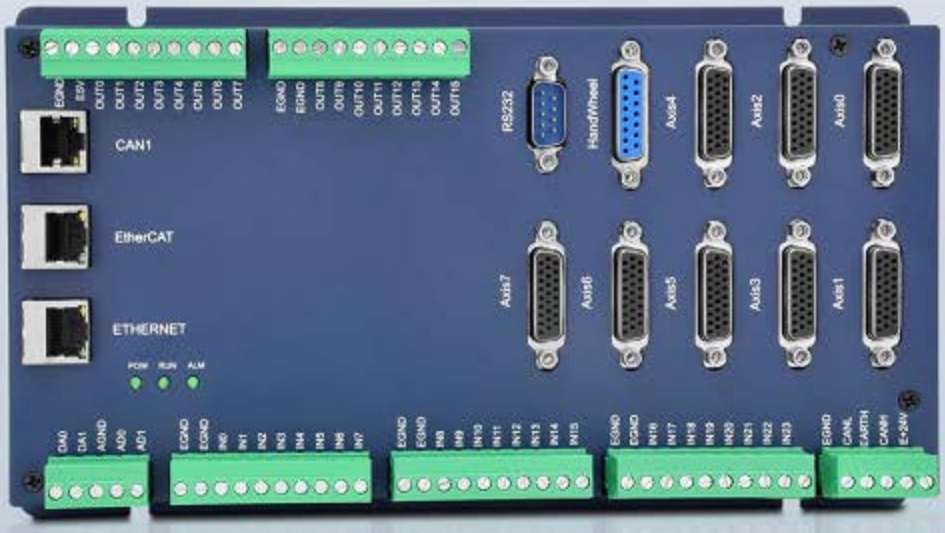
Zmotion®
正运动技术

做最好用的运动控制
DO THE BEST TO USE MOTION CONTROL

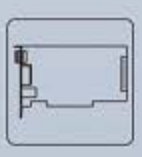
Zmotion PC Function Library Programming Manual

Version2.1.1




Vision Motion Controller


Motion Controller


Motion Control Card


IO Expansion Module


HMI

Zmotion PC Function Library Programming Manual V2.1

(for basic & advanced functions)

Contents

Chapter I Product Description

1.1.Features of Zmotion Controller

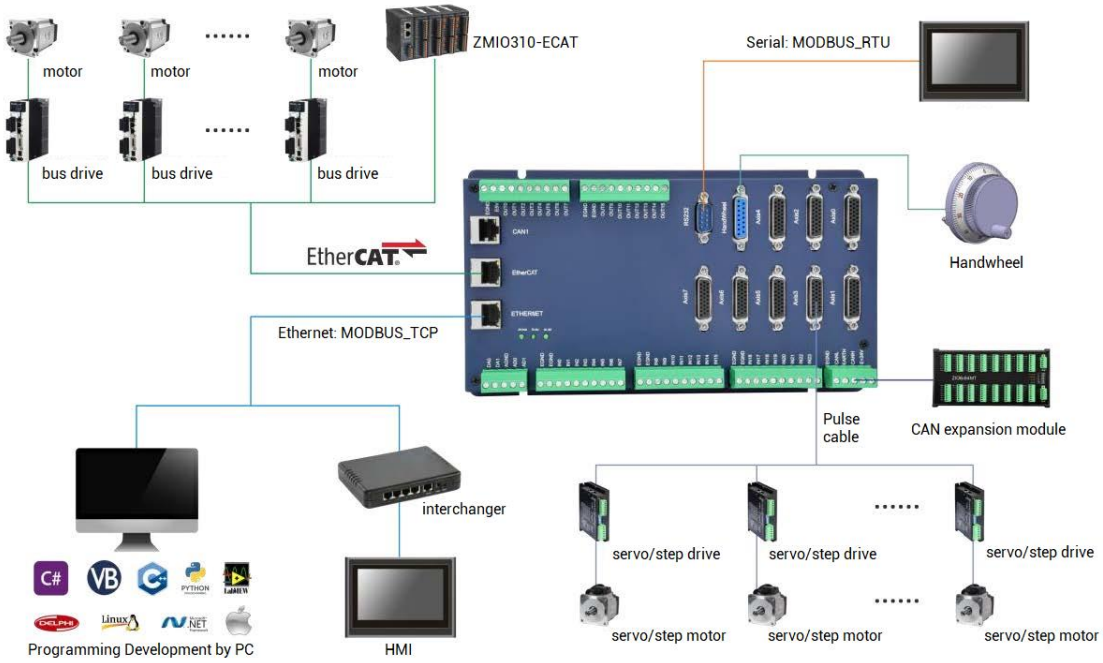
ZMC motion controller supports PC direct online control and provides DLL function library and routines, such as, VC, VB, C#, PYTHON, LABVIEW, etc. The function library supports both WINCE and LINUX. And the library is available for all types of controllers.

Compared with PCI, ZMC motion controller online control has the following advantages:

1. No slot is used, which means the stability is better.
2. Reduce the requirement for PC, no PCI slot is needed.
3. MINI computer or ARM industrial computer can be chosen to reduce the overall cost.
4. The controller is directly used as a wiring board to save space.
5. Programs can be run in parallel on the controller, then the complexity of the PC software can be reduced, because only simple interaction with the PC is required.
6. The motion controller is a special controller that controls the operation mode of the motor. The control structure mode is generally: controller + driver + (step or servo) motor.
7. The controller supports functions such as speed look-ahead, electronic cam, electronic gear, pitch compensation, fixed-step tracking, motion superposition, virtual axis, pulse closed loop, hardware position latch, position comparison output, continuous interpolation, and motion pause, etc.

From above, it can be seen that a motion controller with an Ethernet interface can be used to replace the PCI motion control card for saving space, reducing costs, optimizing programs, and making wiring more convenient, and that is why people prefer using Ethernet now.

1.2. Product Framework



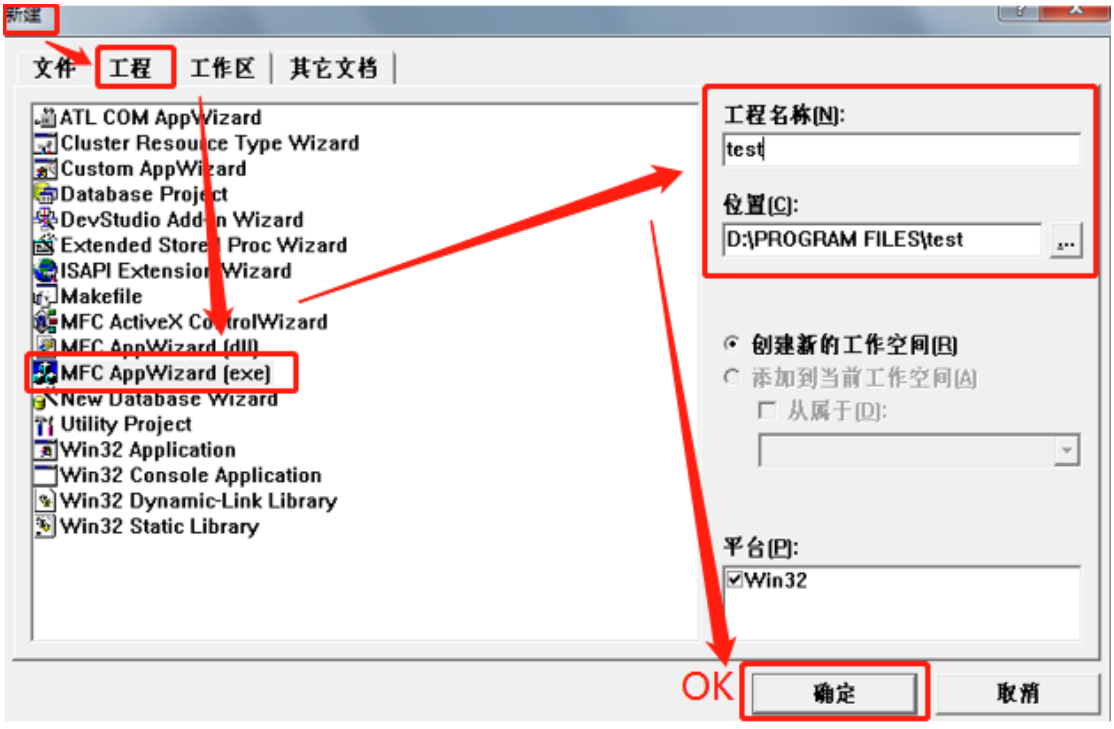
Chapter II Usage and Encapsulation of Function Library

2.1.Function Library Calling

2.1.1.Windows System

2.1.1.1. VC++6.0

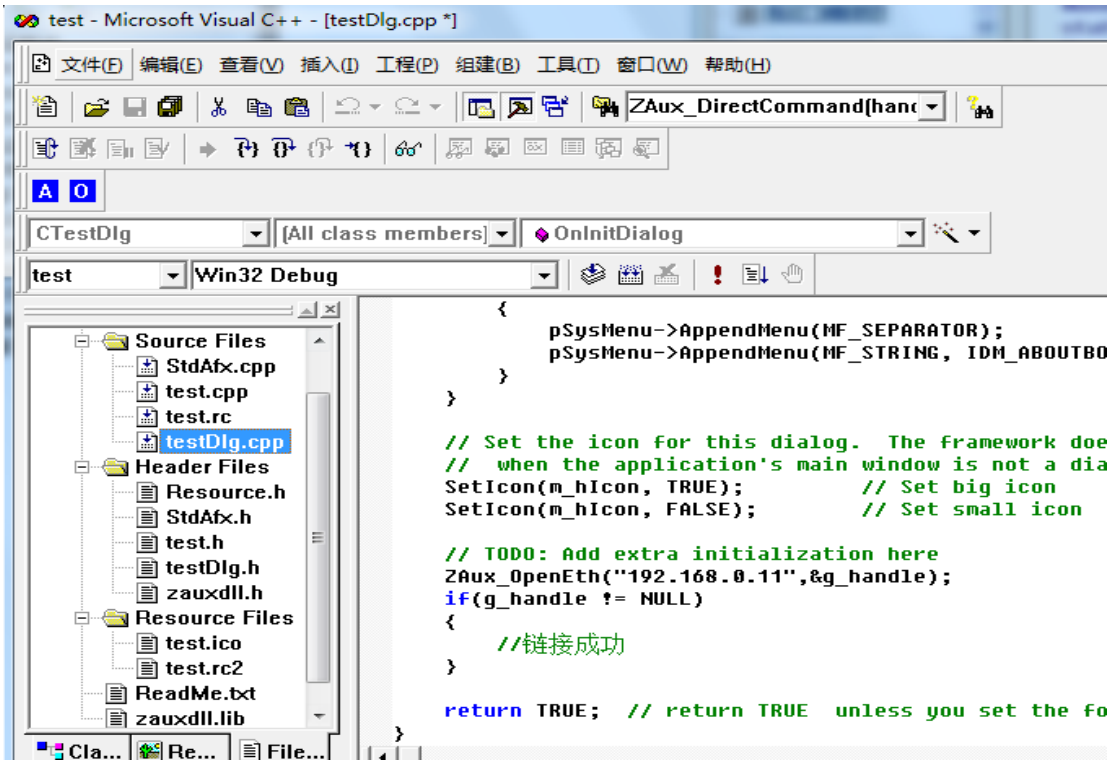
1) Open VC++6.0.
2) Build a new project.
3) Select [MFC APPWizard(exe)]
4) Select project storage route.
5) Set the name of project, select “OK”.



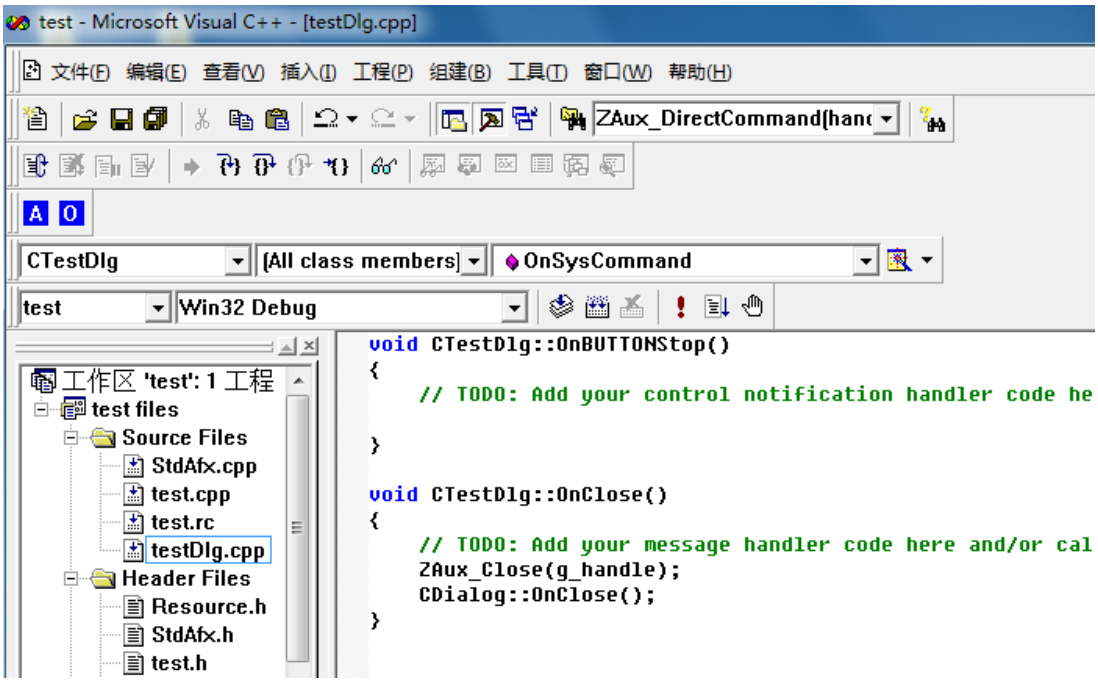
6) Choose “basic” procedure type, then new project is created.
7) Put libraries: zauxdll.h, zauxdll.lib, zauxdll.dll and zmotion.dll under the project path, or directly to import source code through zauxdll.cpp.

8) Add header file #include “zauxdll.h” under testDlg.cpp.

9) Find the CTestDlg::OnInitDialog() function, and add controller link code: ZAux_OpenEth(“192.168.0.11",&g_handle).



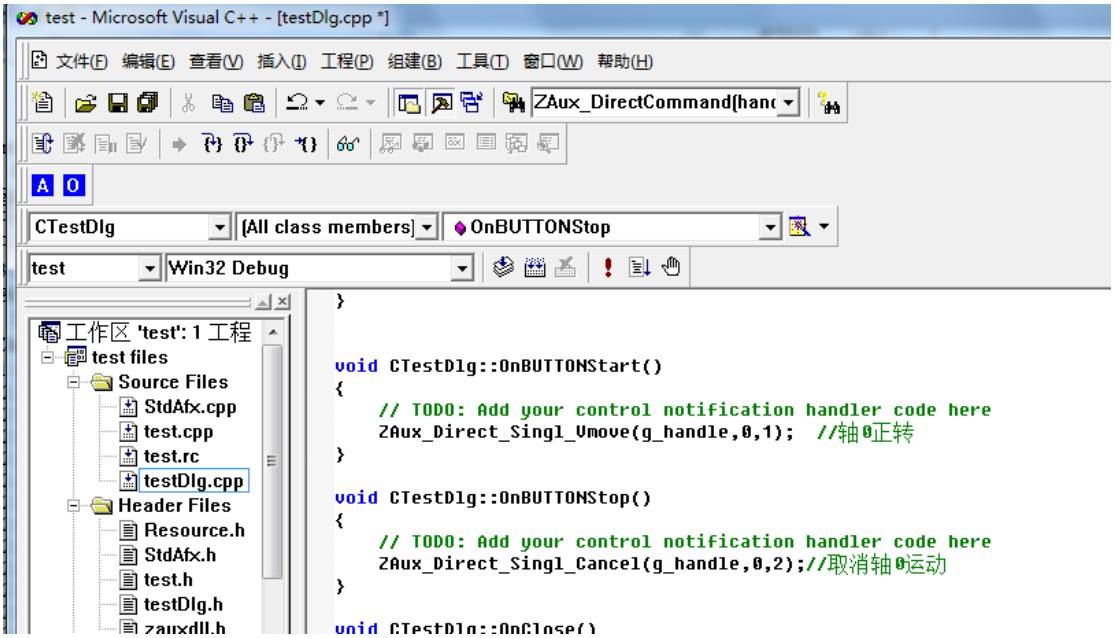
10) Add a member function: CTestDlg::OnClose() under CTestDlg to close window and disconnect controller link, and add function: ZAux_Close(g_handle) under CTestDlg::OnClose() to disconnect link.



11) Double click to start or stop to add relative codes under its corresponding event function:

ZAux_Direct_Single_Vmove(g_handle,0,1);

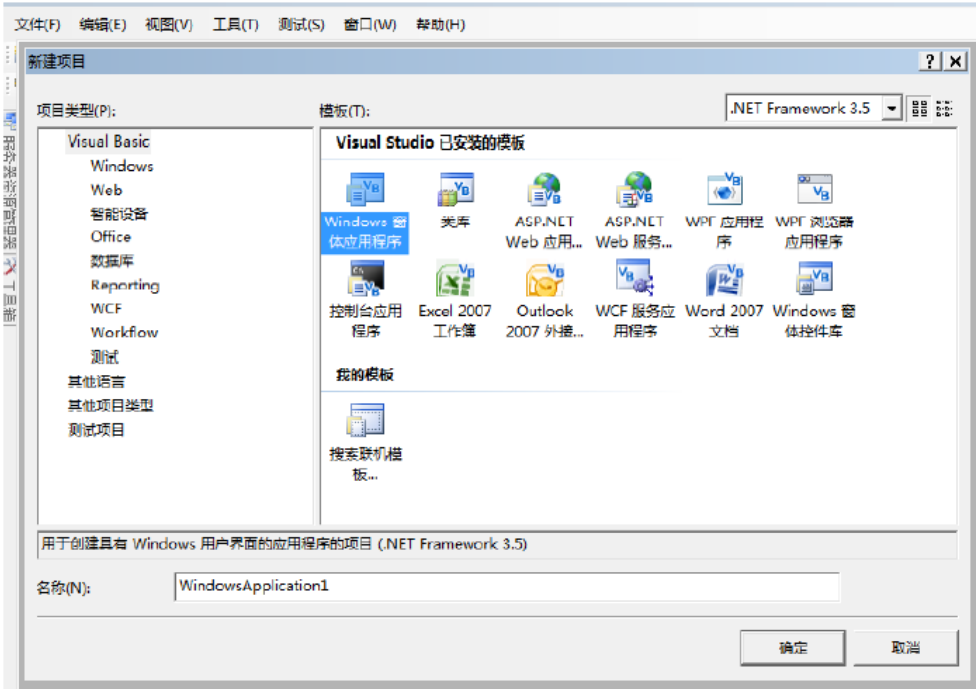
ZAux_Direct_Single_Cancel(g_handle,0,2).



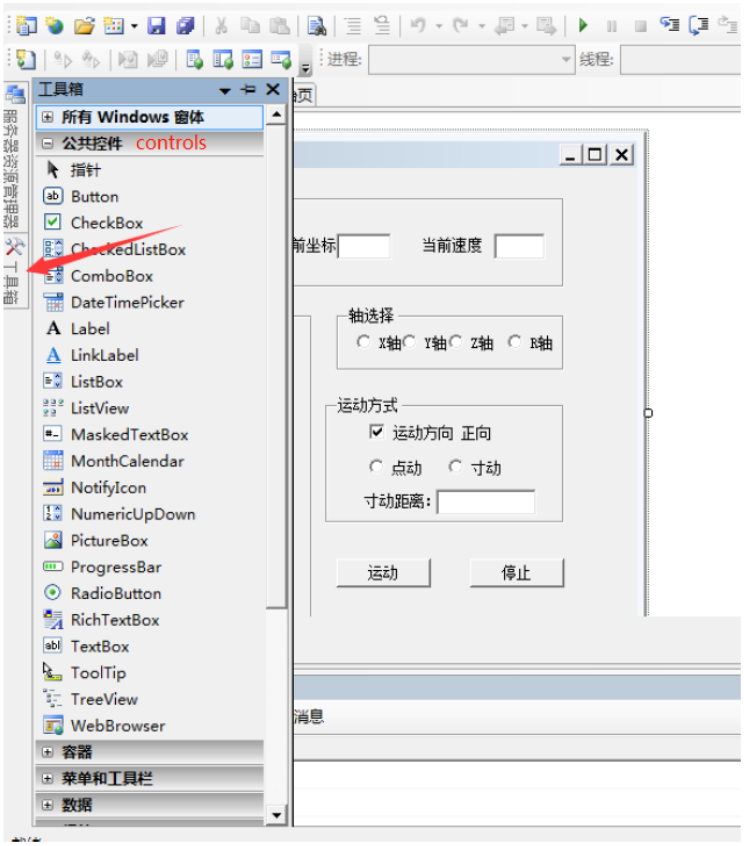
12) After the program was compiled, then run the program. Press “run” button on interface, axis 0 will move forward at its initial speed, press “stop” button, then axis 0 stop moving.

2.1.1.2. VB.NET

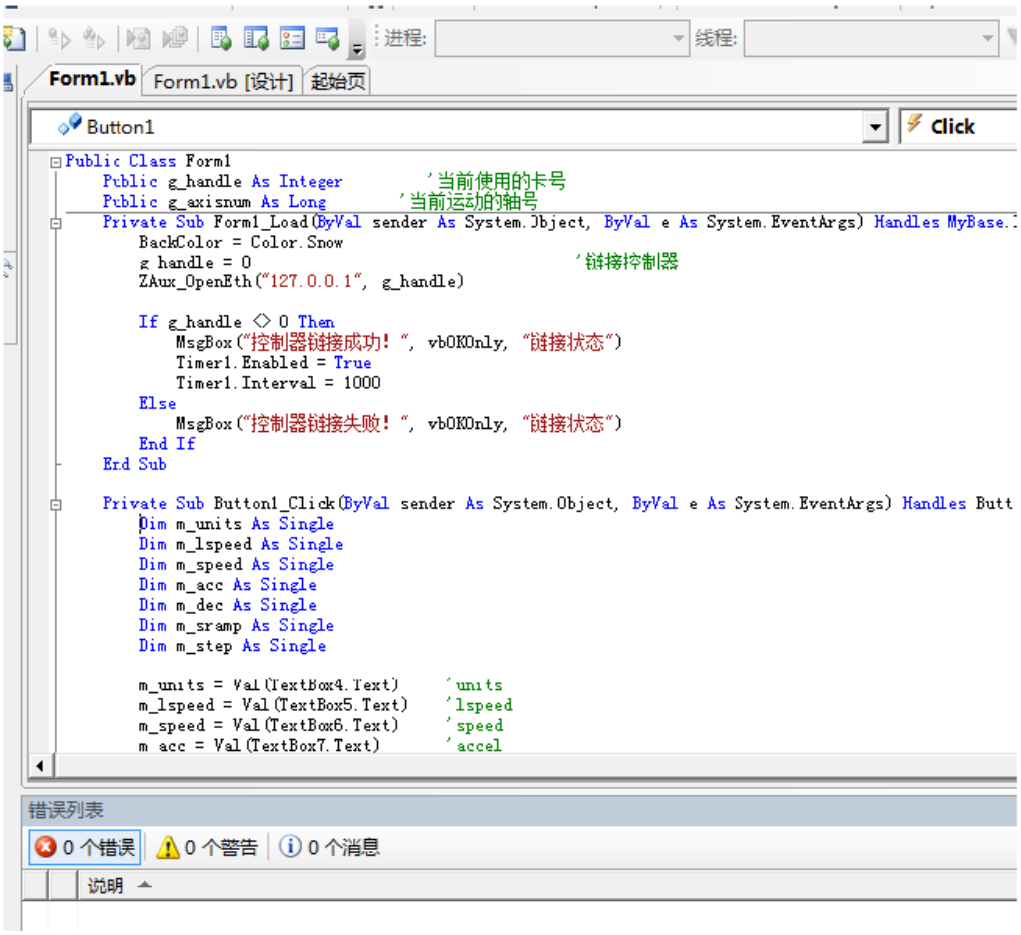
1) Build new project, and save.



2) Select corresponding widgets (controls) to complete operation interface according to required functions and interfaces.

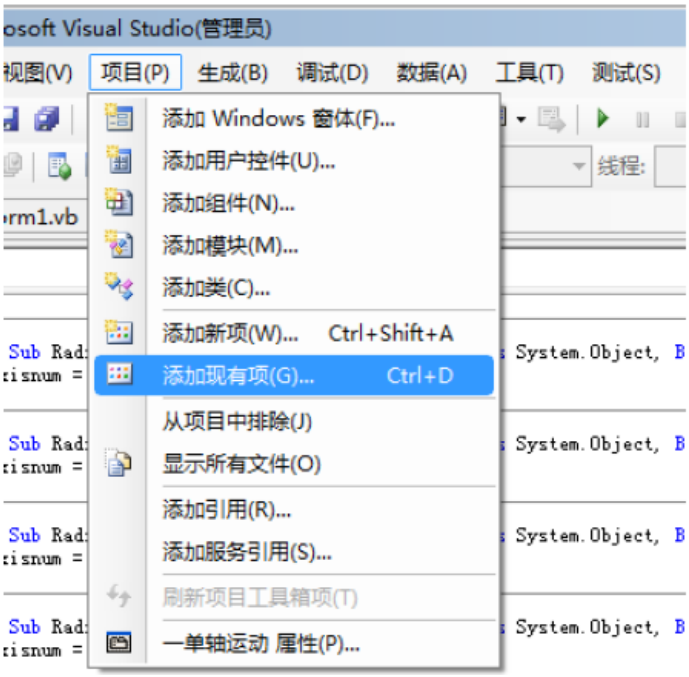


3) VB.Net is object-oriented programming language, choose reasonable trigger events to program, such as Button_Click, Checked, Load, etc.

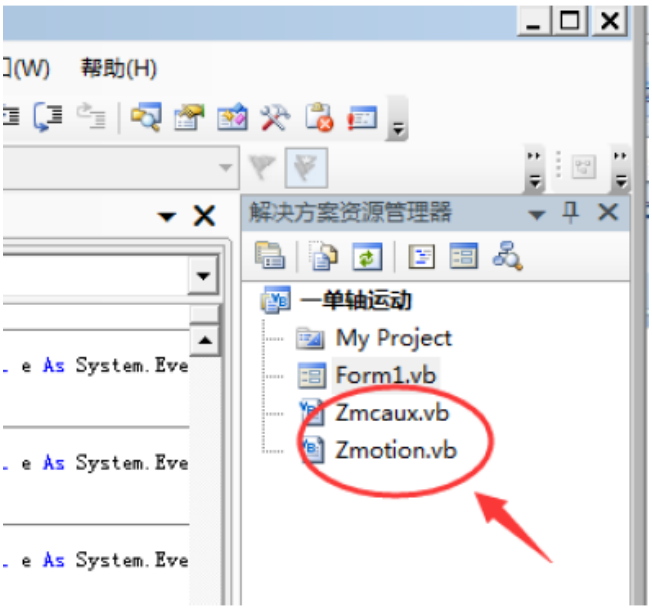


4) Add dll file and header file and put them in the same folder as the project file.

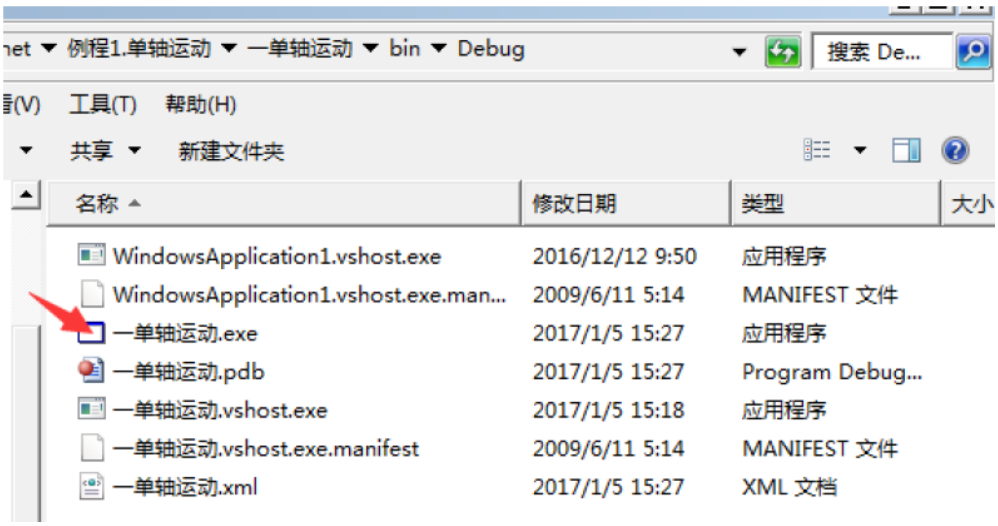
5) Add the header file in the project (2 .vb files)



After add it successfully, it will show:

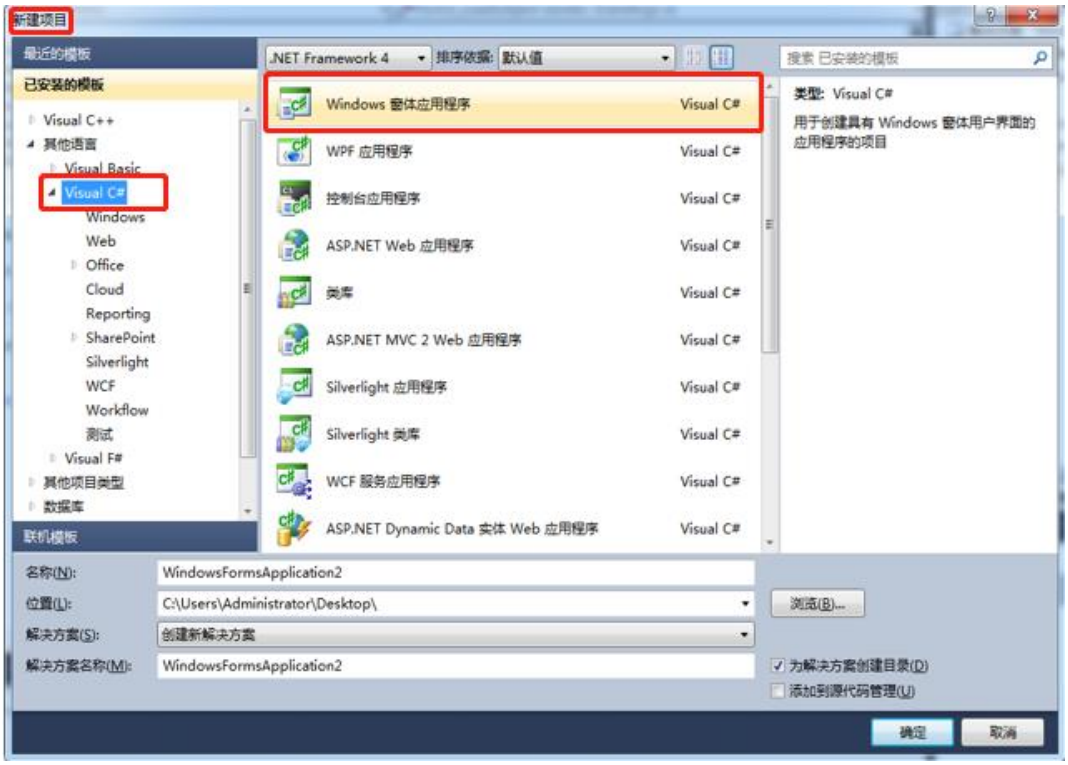


6) After PC links to controller successfully, then set IP well, start to debug.
7) After procedure was run and debugged, there is a .exe file in folder: “bin” \ “Debug”, click it, then it can operate directly.

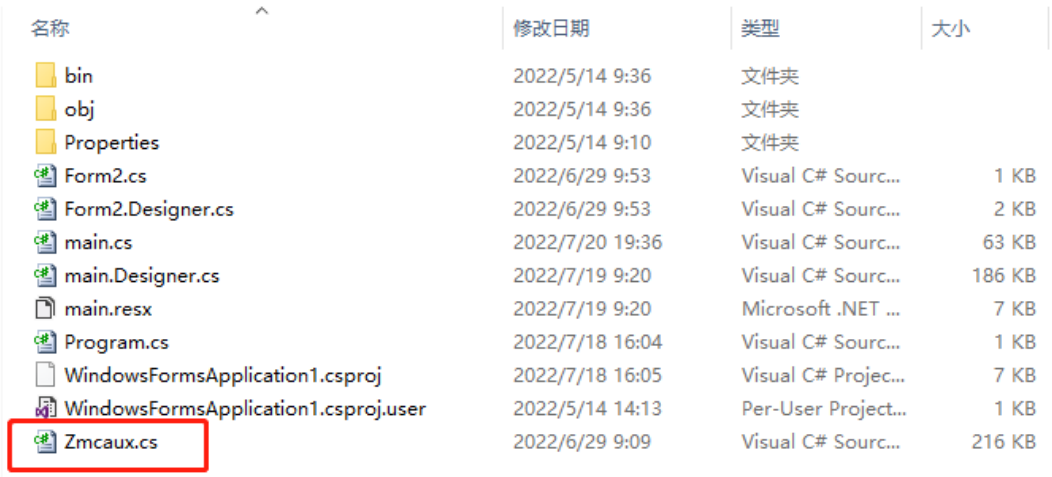


2.1.1.3. C#

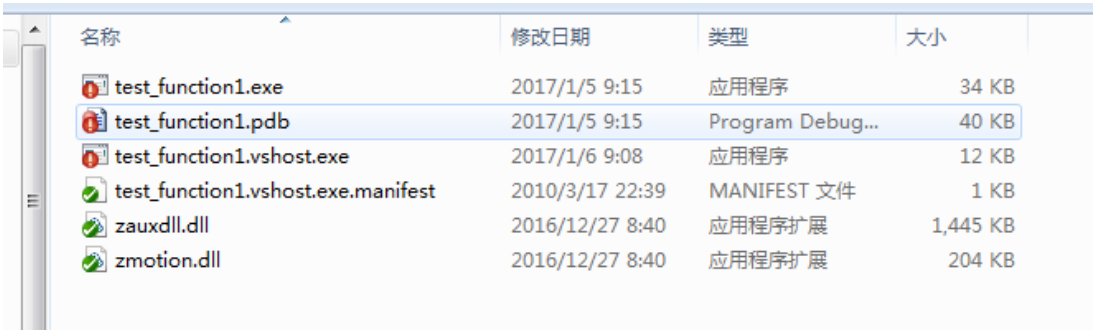
1) Open VS2012, click “File” – “New” – “Project”, select “windows application” type.



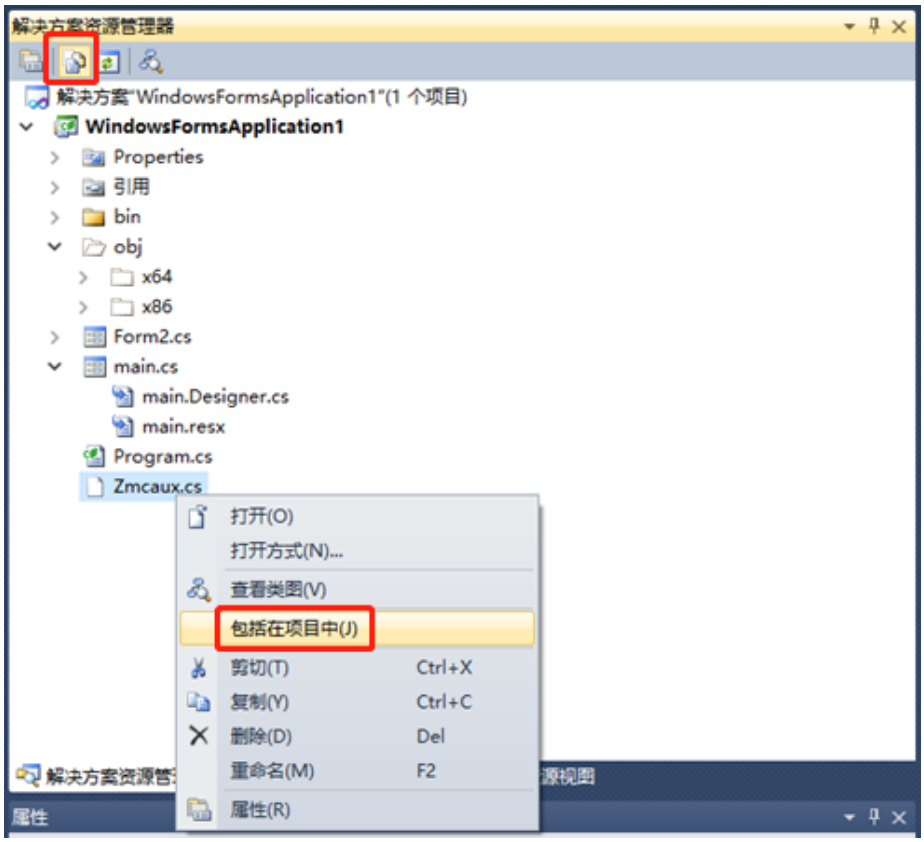
2) Find the folder built just now, then put file zmcaux.cs into folder.



3) Put files zaux.dll and zmotion.dll into the folder “debug”.



4) Use “VS” to open new built project, then click “display all files” in the right “Resource Manager”, then right click zmcaux.cs file – “included in Project”.



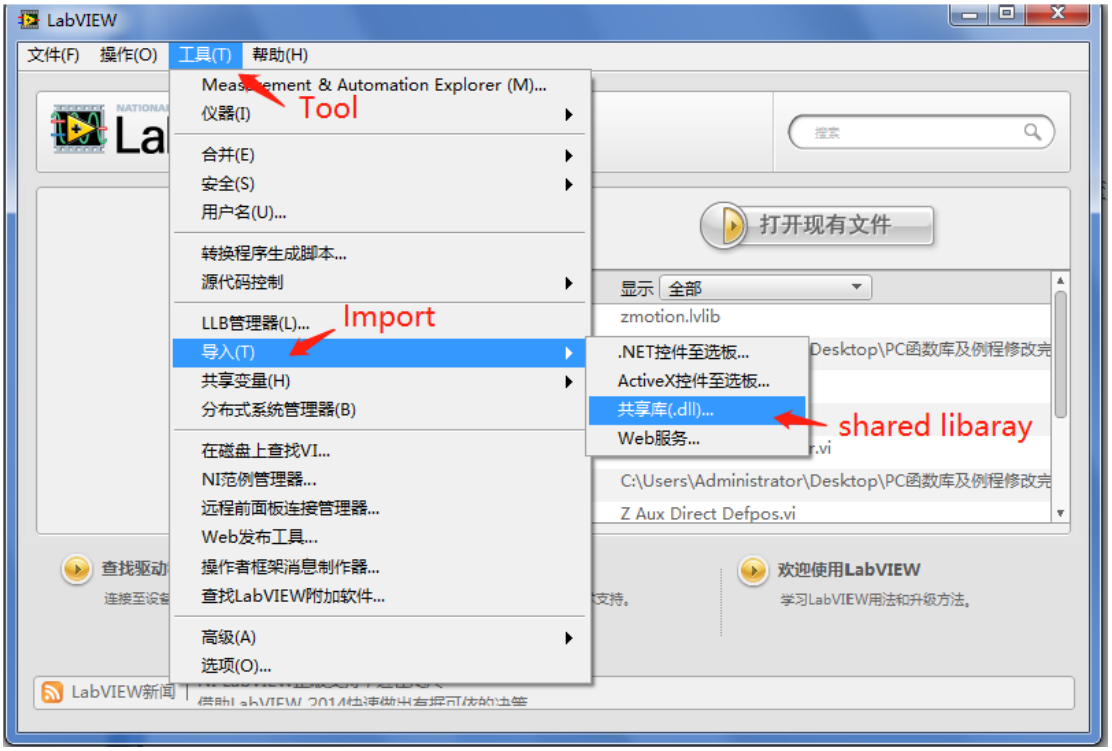
5) Double click “form 1", code editing interface will appear, now, write “using cszmcaux" at the beginning of the file.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Threading;                    //线程声明
using System.Text.RegularExpressions;//正则表达式声明

using cszmcaux;
namespace WindowsFormsApplication1
{
    public partial class main : Form
    {
        public IntPtr g_handle; //声明控制器连接句柄
```

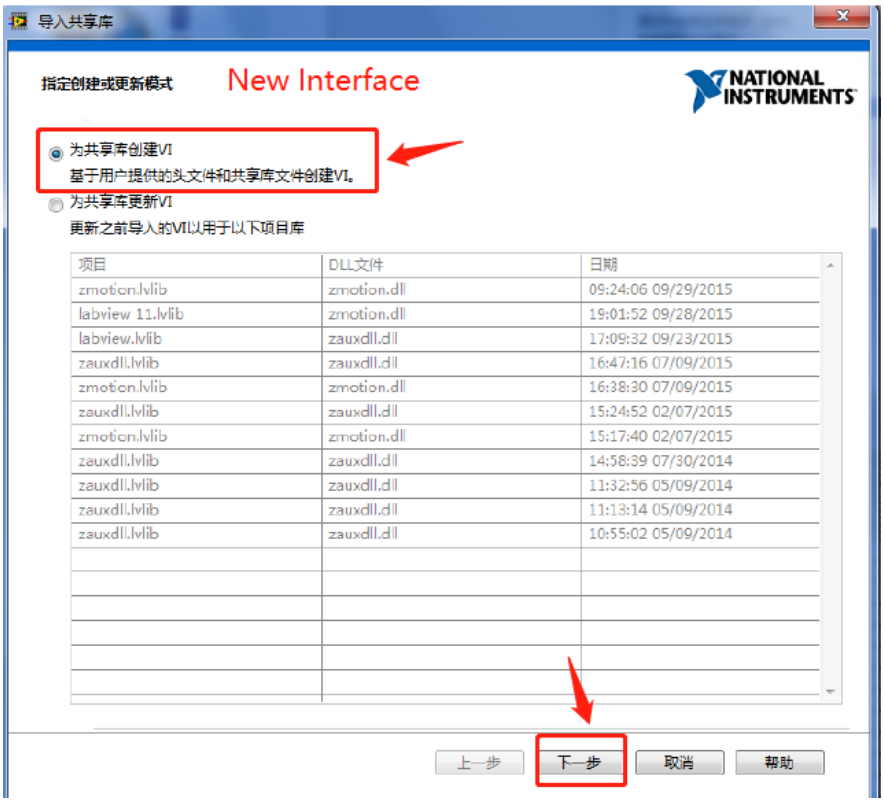
6) Now, it is time to program.

2.1.1.4. LABVIEW

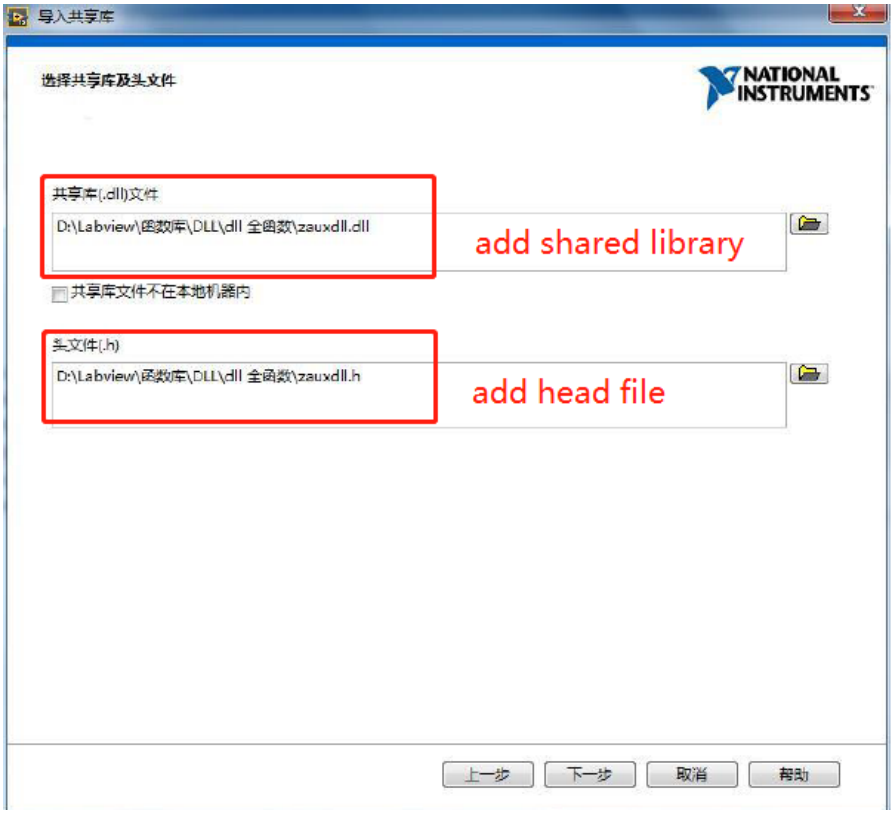
1) Open labview interface. In “Tool”, click “Import” – “shared library (dll)".



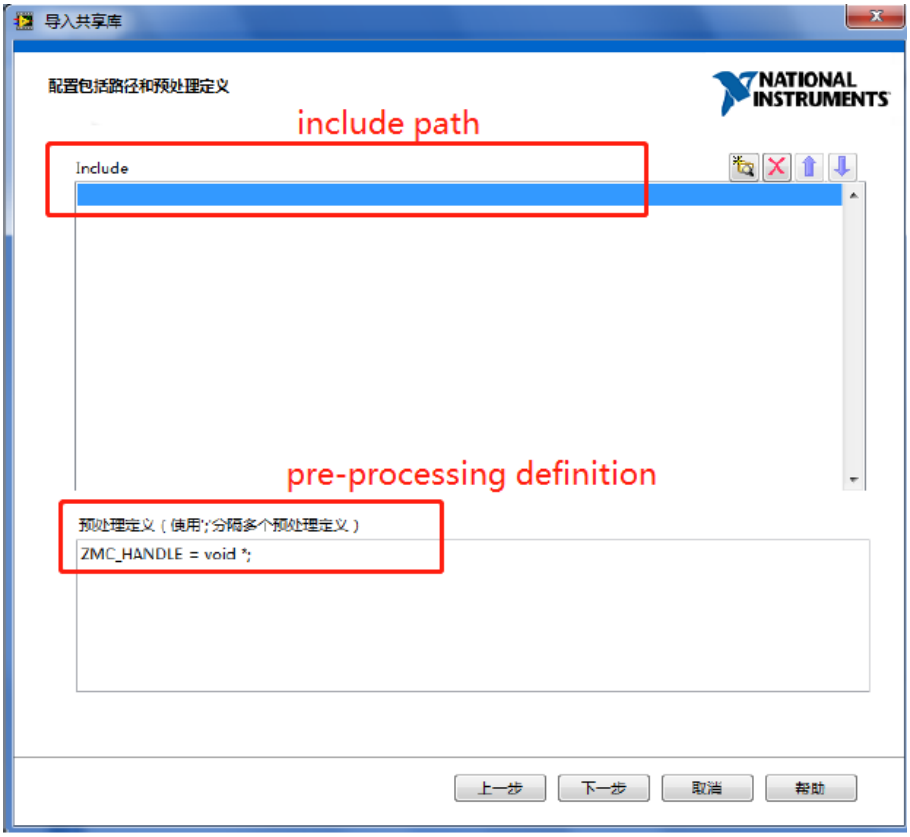
2) Then, a new interface appears, build the new Vi for shared library to enter the next step.



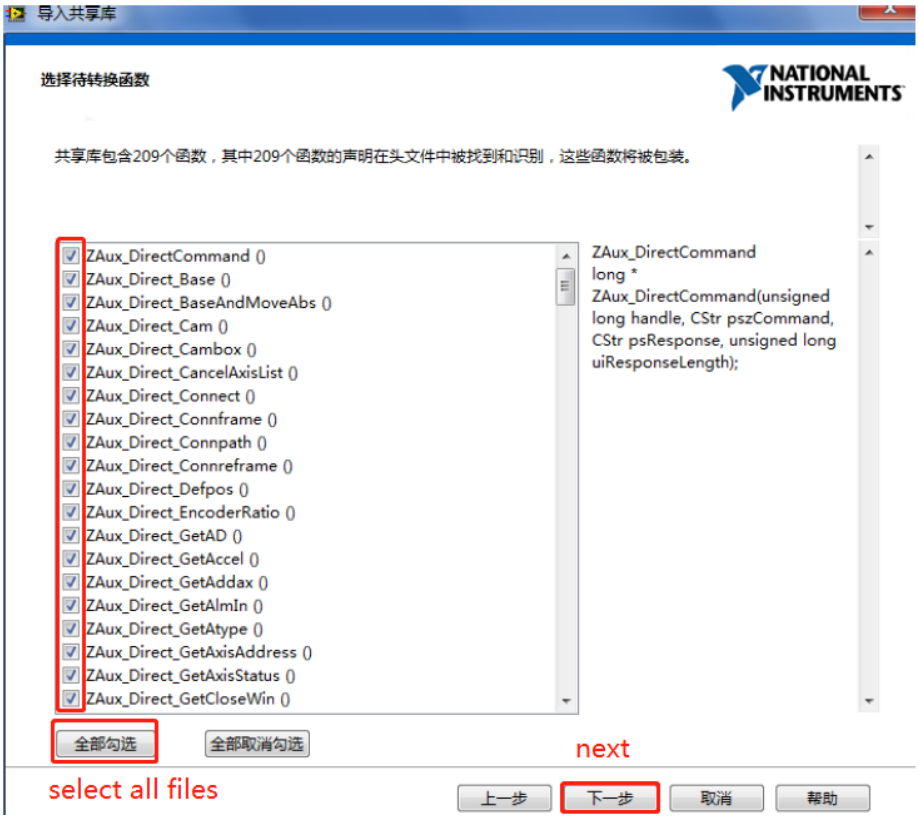
3) Select “shared library (.dll)” and “header file (.h)”, and load zauxdll.h and zauxdll.dll files.



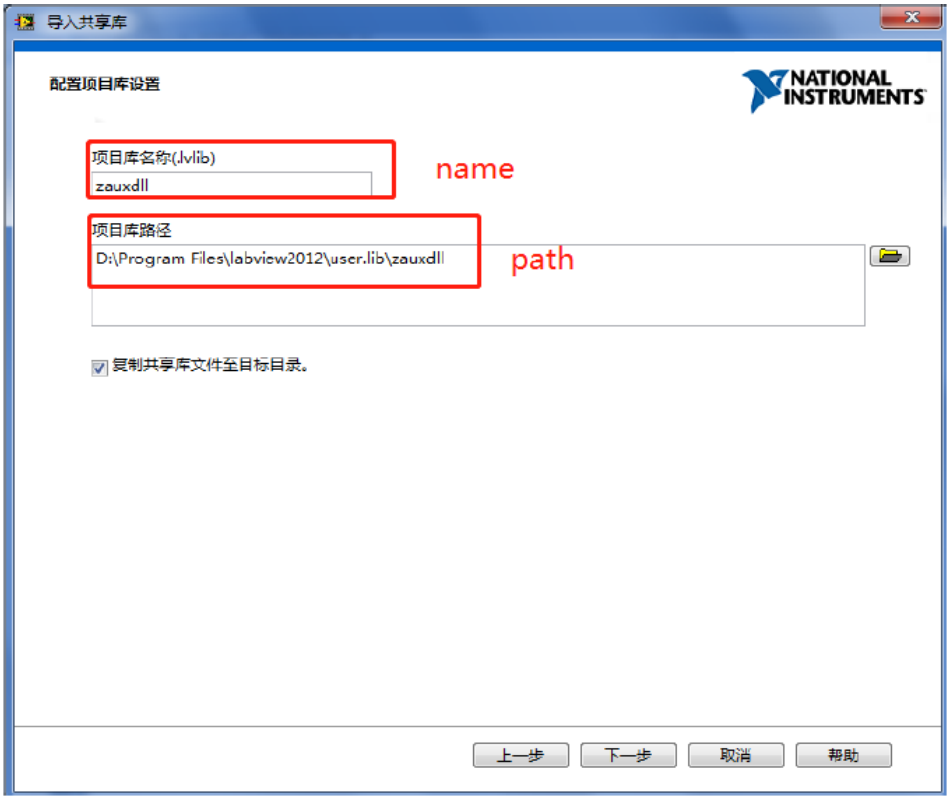
4) Configure the command that includes path and macro definition, and add the preview processing instruction “ZMC_HANDLE = void *;” while importing ZAUXDLL library.



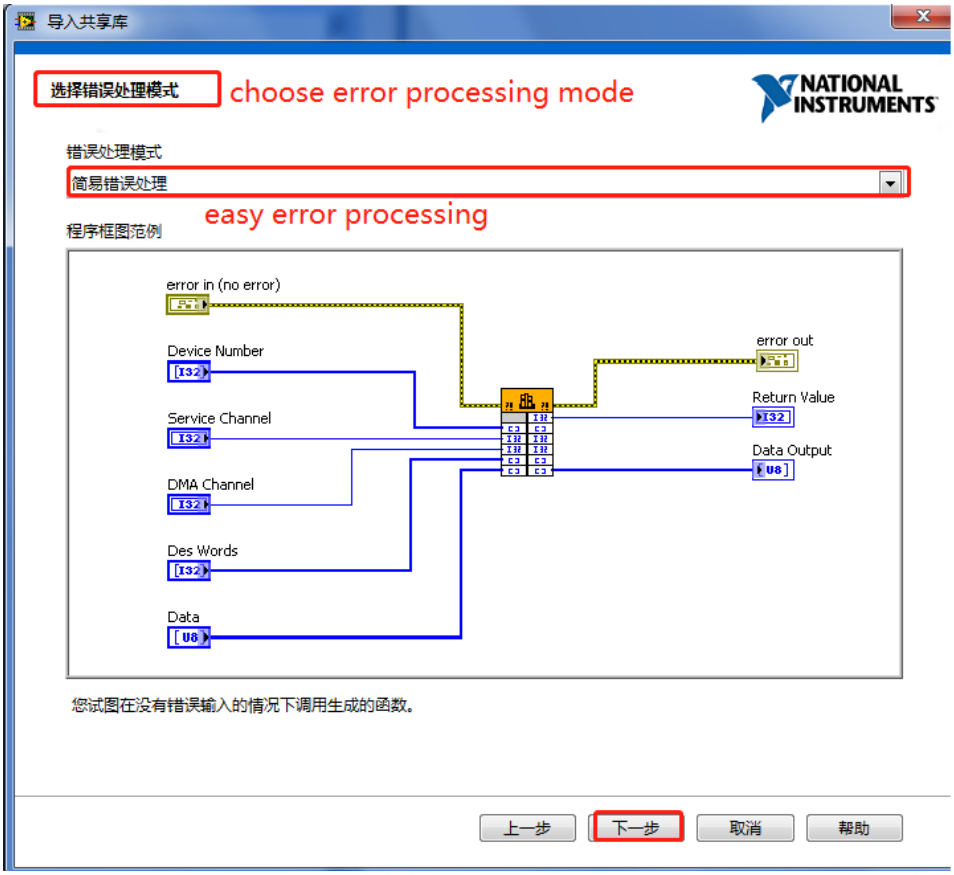
5) Select all function definition in DLL library, then click “next” (some functions in basic library are not used usually, therefore, some functions can be ignored).



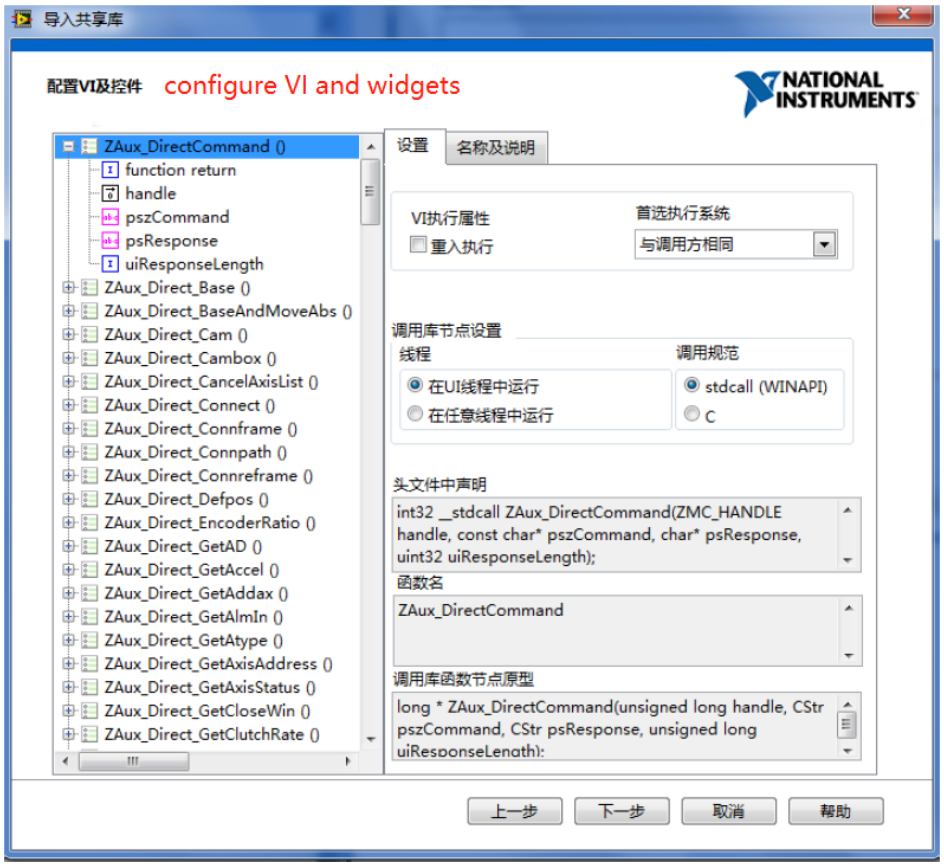
6) Configure the name and path of generated VI library.



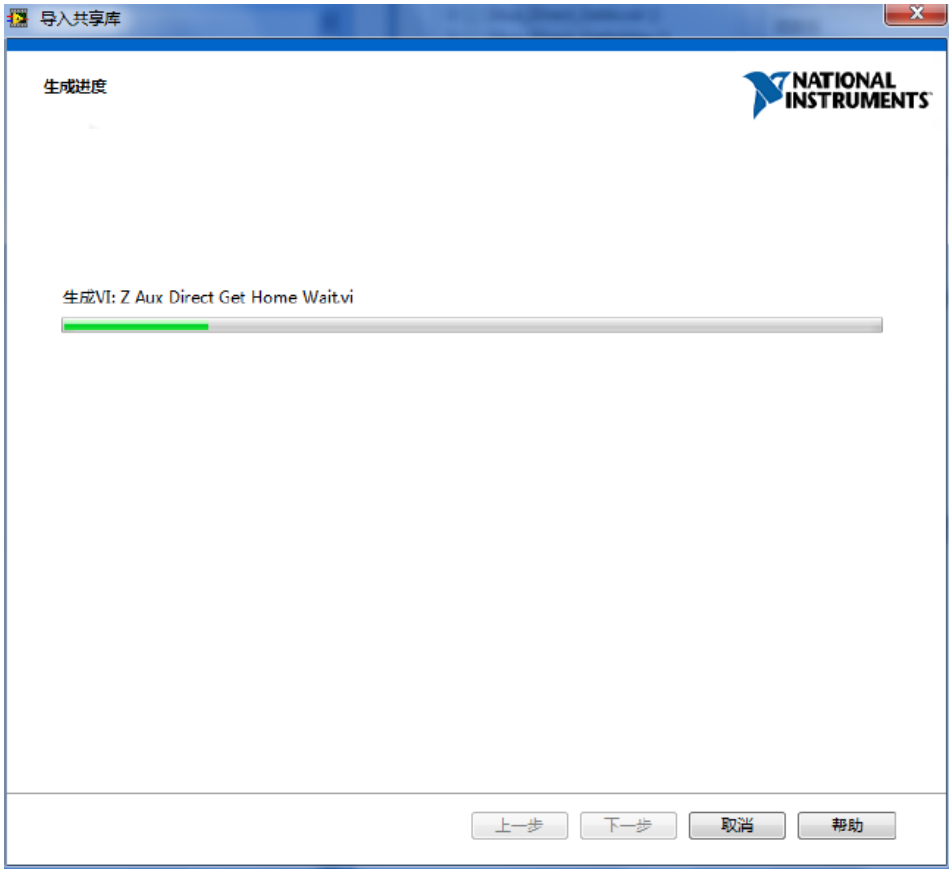
7) There are many error processing methods, choose “easy error processing mode”.



8) Configure VI and widgets, same as .DLL setting.



9) Keep clicking “next” until installation finished.



2.1.1.5. Python

1. Import dynamic library

   1) In Python, it needs to import platform and ctype modules. Platform is used to gain types of former operating system, ctype is used to import relative link library -- dynamic library, mainly is C/C++.

   2) Choose related dynamic library(C/C++) according to different platforms, such as, libraries in widowsX86, windowsX64, Linux, etc.

```python
import platform
import ctypes

systype = platform.system()
if systype == 'Windows':
    if platform.architecture()[0] == '64bit':
        zauxdll = ctypes.WinDLL('./zauxdll64.dll')
        print('Windows x64')
    else:
        zauxdll = ctypes.WinDLL('./zauxdll.dll')
        print('Windows x86')
elif systype == 'Darwin':
    zmcdll = ctypes.CDLL('./zmotion.dylib')
    print("macOS")
elif systype == 'Linux':
    zmcdll = ctypes.CDLL('./zmotion.so')
    print("Linux")
else:
    print("OS Not Supported!!")

```

2. Python function encapsulation

   1) To do function encapsulation together with dynamic libraries.

   2) According to requirements, encapsulate relative functions.

```python
class ZAUXWrapper:
    def __init__(self):
        self.handle = ctypes.c_void_p()
        self.sys_ip = ""
        self.sys_info = ""
        self.is_connected = False

    def search(self, console=[]):
        iplist = ctypes.create_string_buffer(b'', 1024)  # create_string_buffer创建的是一个 ANSI 标准的 C类型字符串
        zauxdll.ZAux_SearchEth(ctypes.byref(iplist), 1024, 200)
        s = iplist.value.decode()
        str_iplist = s.split()
        num = len(str_iplist)
        print(num, "Controller(s) Found:")
        print(*str_iplist, sep='\n')
        console.append("Searching...")
        console.append(str(num) + " Controller(s) Found:")
        return str_iplist, num

    def connect(self, ip, console=[]):
        if self.handle.value is not None:
            self.disconnect()
        ip_bytes = ip.encode('utf-8')
        p_ip = ctypes.c_char_p(ip_bytes)
        print("Connecting to", ip, "...")
        ret = zauxdll.ZAux_OpenEth(p_ip, ctypes.pointer(self.handle))
        msg = "Connected"
        if ret == 0:
            msg = ip + " Connected"
            self.sys_ip = ip
```

Description: refer to this manual, functions can be encapsulated.

3. Encapsulate ZBasic instructions in Python.

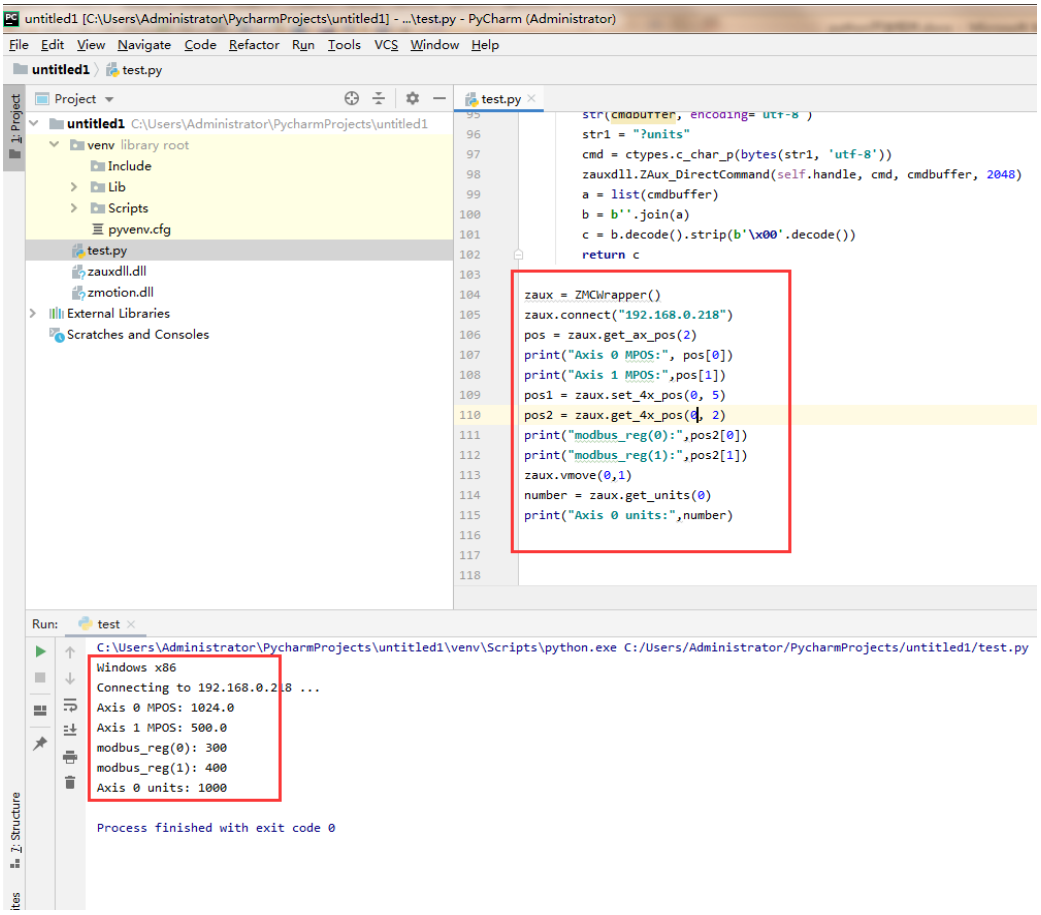
   For example:

```
85      def vmove(self, iaxis, idir):
86          cmdbuffer = (ctypes.c_char * 2048)()
87          str1 = "VMOVE(%d) AXIS(%d)" % (idir, iaxis)
88          cmd = ctypes.c_char_p(bytes(str1, 'utf-8'))
89          zauxdll.ZAux_DirectCommand(self.handle, cmd, cmdbuffer, 2048)
90          return
91
92      def get_units(self, iaxis):
93          cmdbuffer = (ctypes.c_char * 2048)()
94          str(cmdbuffer, encoding='utf-8')
95          str1 = "?units"
96          cmd = ctypes.c_char_p(bytes(str1, 'utf-8'))
97          zauxdll.ZAux_DirectCommand(self.handle, cmd, cmdbuffer, 2048)
98          a = list(cmdbuffer)
99          b = b''.join(a)
100         c = b.decode().strip(b'\x00'.decode())
101         return c
```

Description: some functions can be combined with basic instructions to form string commands, and program through the combination of string commands and ZAux_DirectCommand instruction.

4. Use python function

For example:



Description: build the object to be encapsulated, then call relative functions.

2.1.2.LINUX System

2.1.2.1. QT

1) Start Qt Creator 2.6.1, and build a new project.
2) Copy the dynamic link library (zmotion. so) and header file (zmotion. h) into folder "..\linux\function library\x86" of product matched disk into project file.
3) In the Qt Creator interface, open .pro file and add.

```
greaterThan(QT_MAJOR_VERSION, 4): QT += widgets  (version compatibility problem)
SOURCES + = main.cpp\
        mainwindow.cpp\
        zaux.cpp
HEADERS + = mainwindow.h\
        zmotion.h\
        zaux.h
```

Modify the header file of main.c to #include <QApplication>

4) The way to add the dynamic library
   1> right click the project.
   2> click "add the library".
   3> select the second external library, click OK.
   4> Click "browse library file".

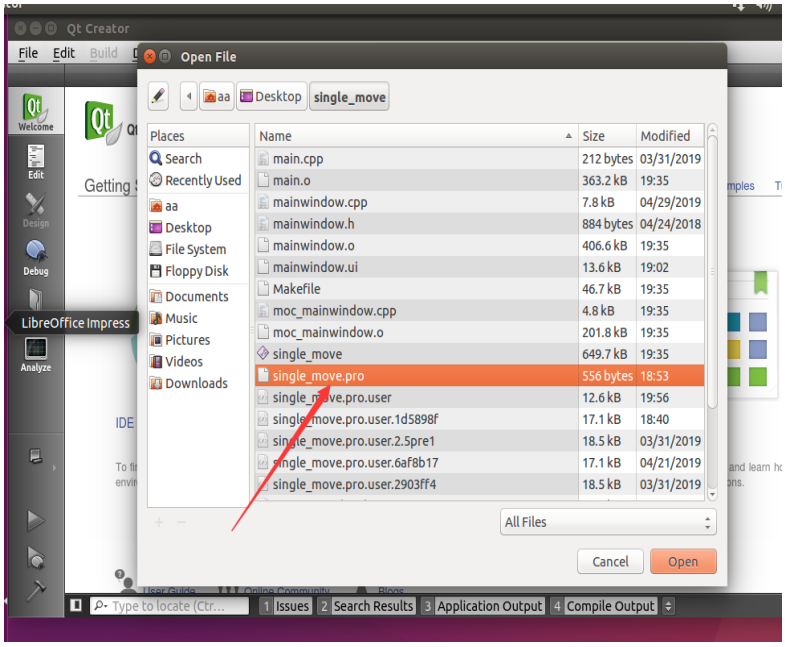
   Note:

   Dynamic library name should start with "lib" to make it identifiable. Then click "next", information of dynamic library will be added into .pro automatically. Following, you can call the header file that includes function, and run program.

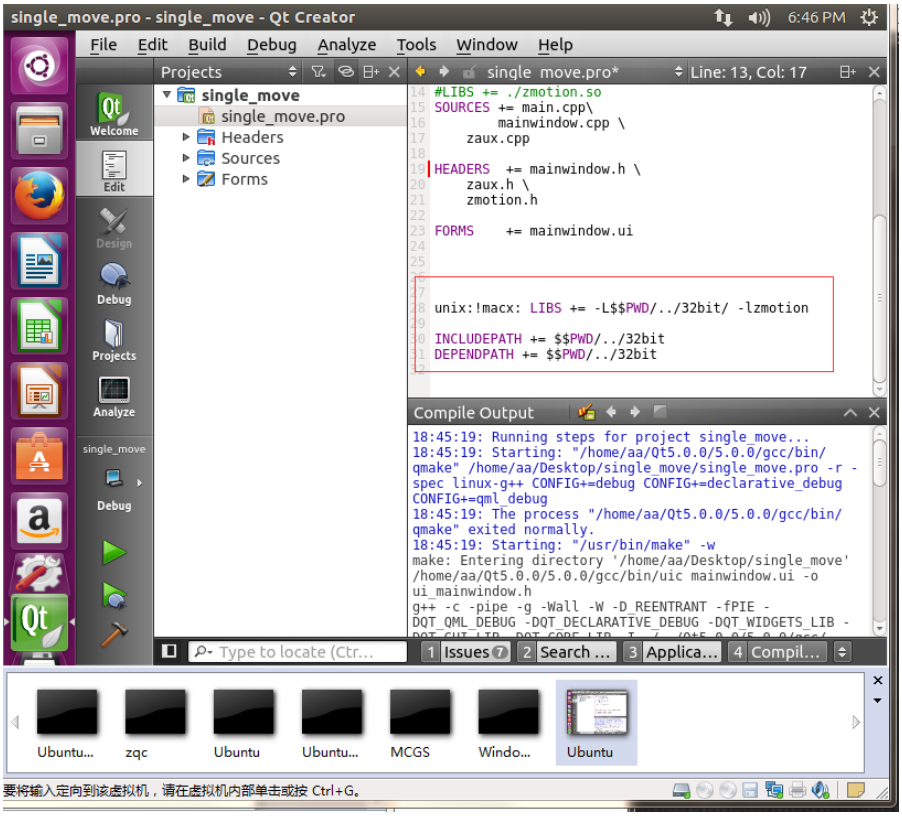
5) Open QT program, click file, find the .pro file in route, then click open.

   Note:

   Path can't contain Chinese characters.



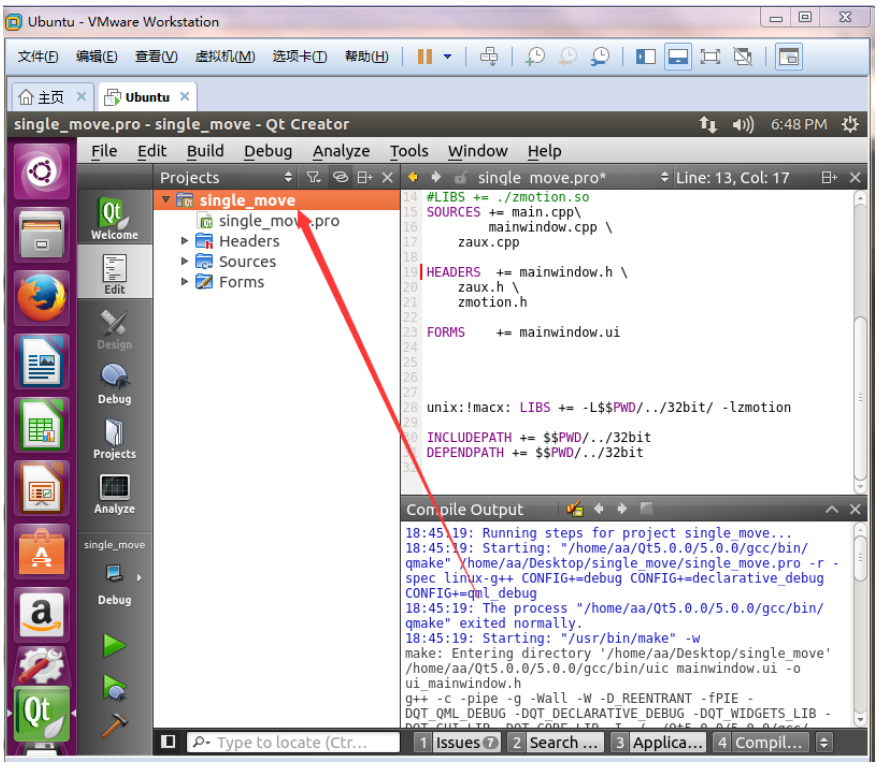
6) Add .pro of QT route in right path dynamic library, double click single_move.pro file, and check the dynamic library path set before, now, set own dynamic library path.



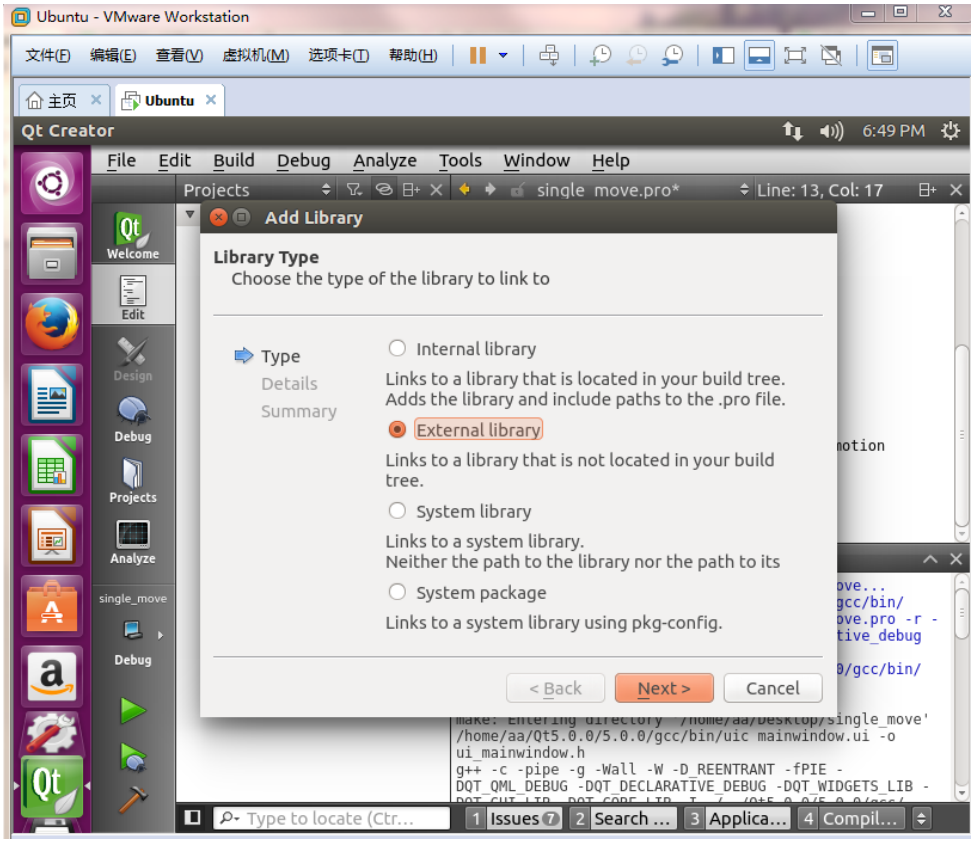
7) Delete dynamic library path in .pro file.

8) Put 32-bit dynamic library of folder into assigned path.

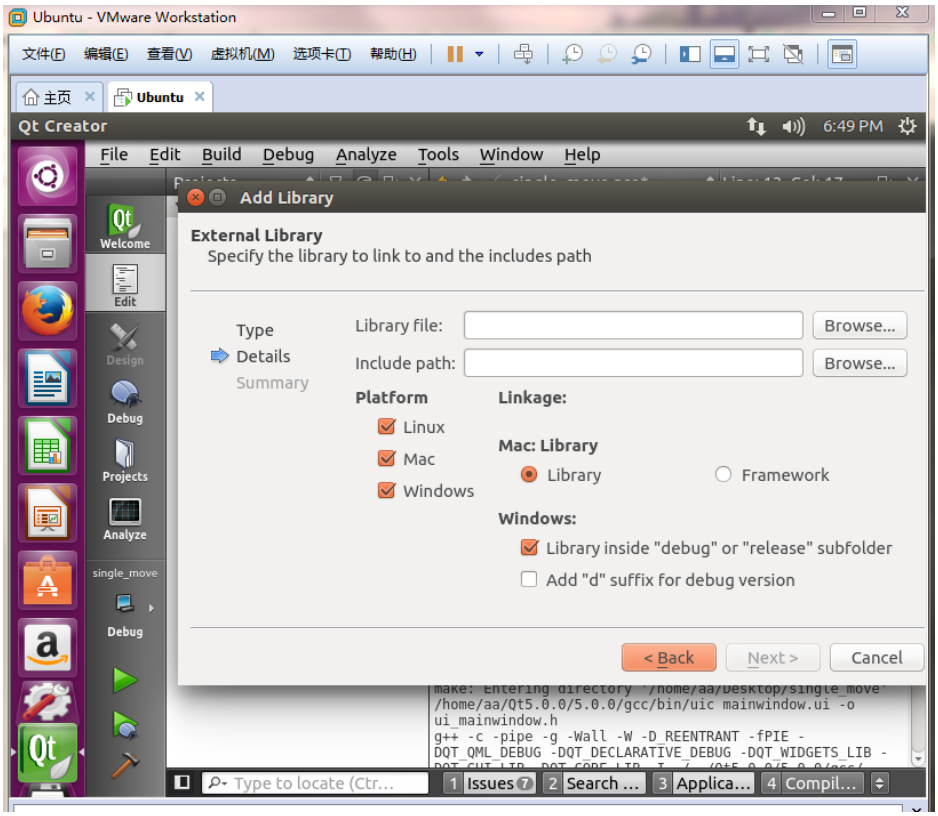
9) Add own dynamic library path, firstly select the project file.



10) Then right click add library.



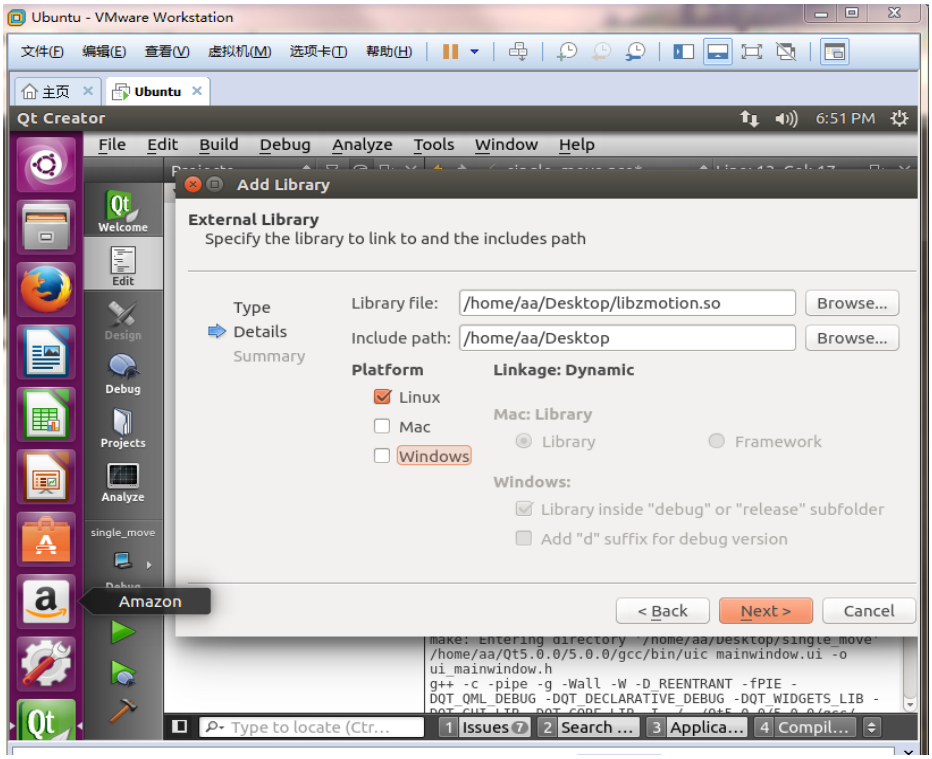
11) Select the second “Details” add external library.



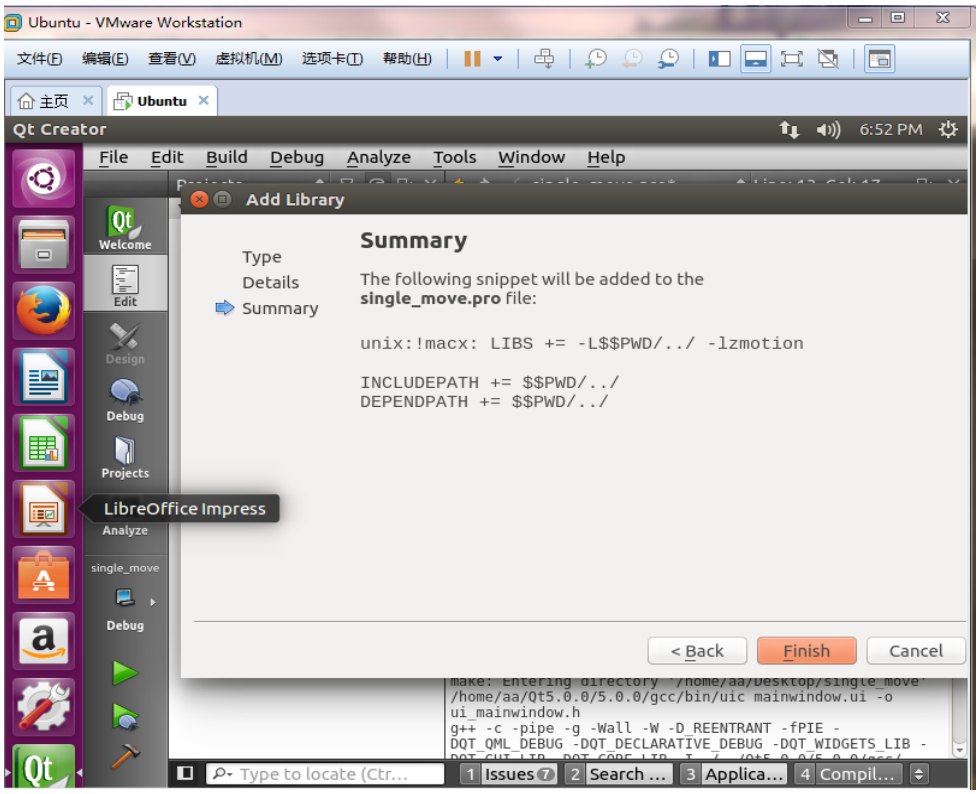
12) Now, find the path that saved the dynamic library just now.

Note:

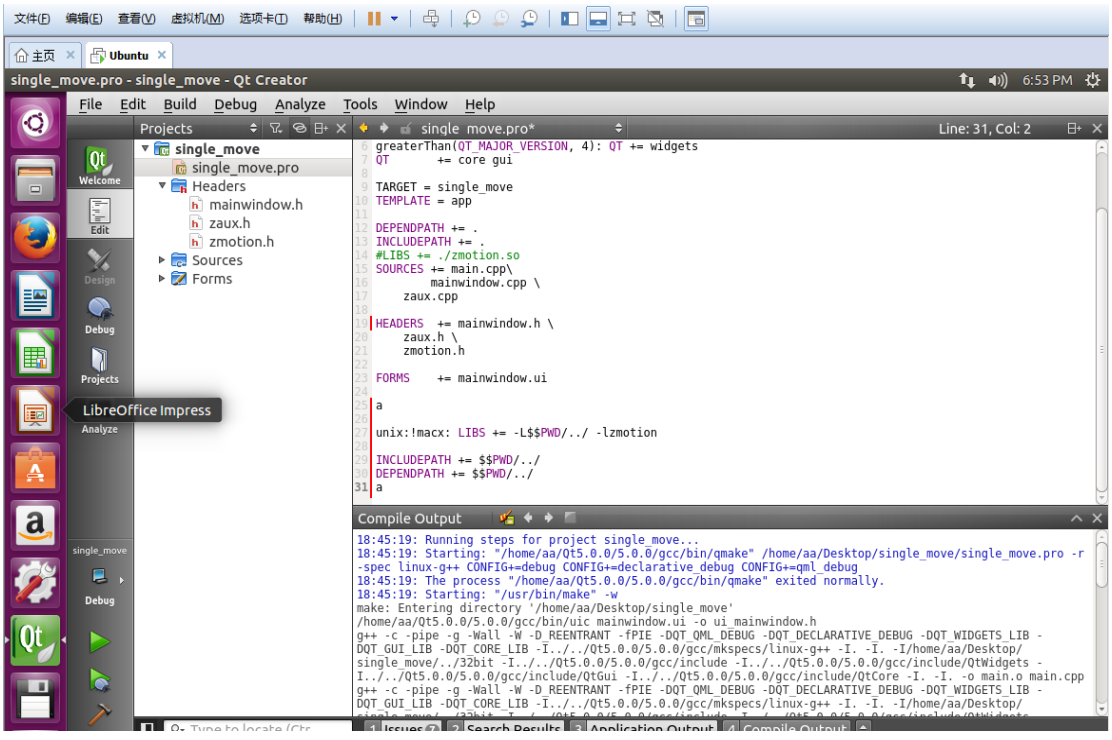
Dynamic library name of Linux should start with “lib” to make it identifiable. Don’t change the file name casually.



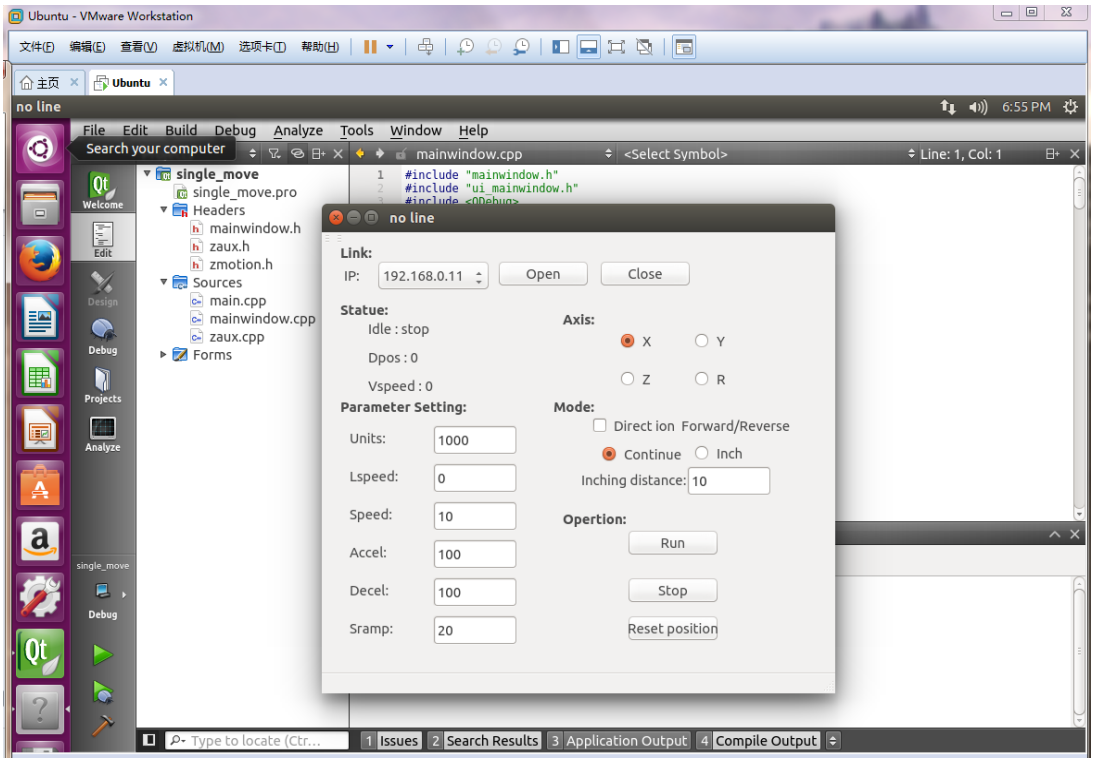
13) Click “next”.



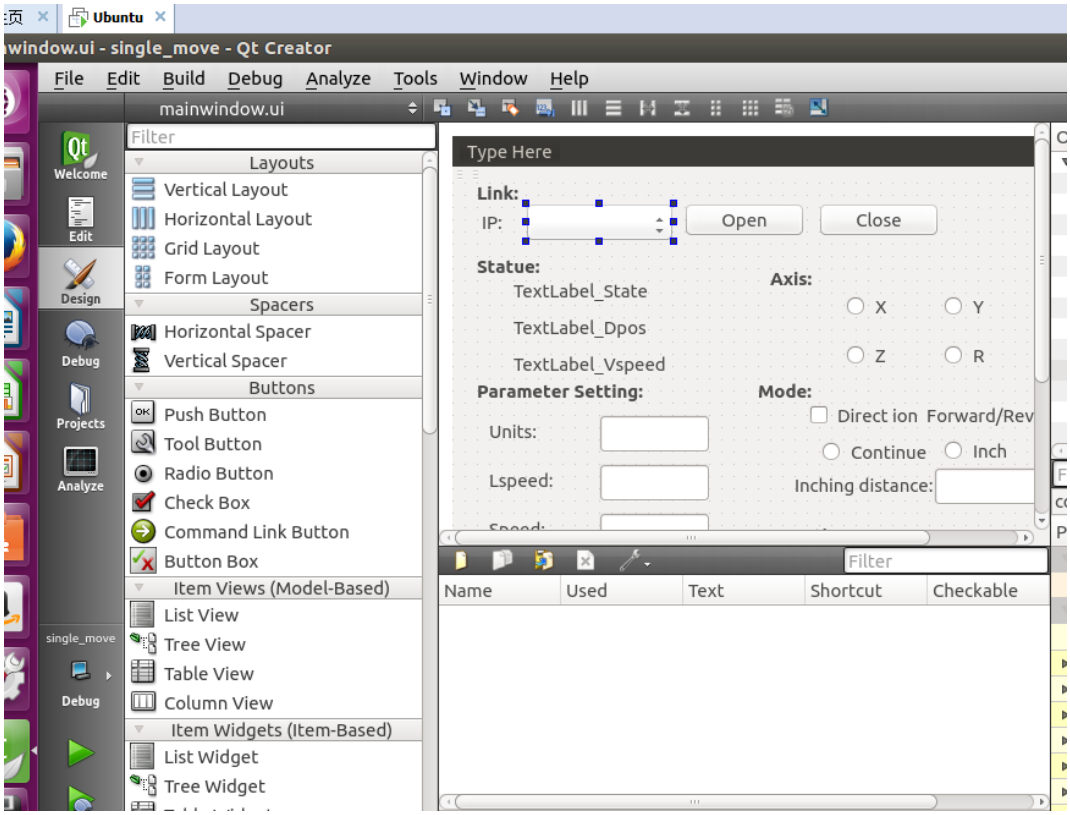
14) Check the path of dynamic library generated in .pro.

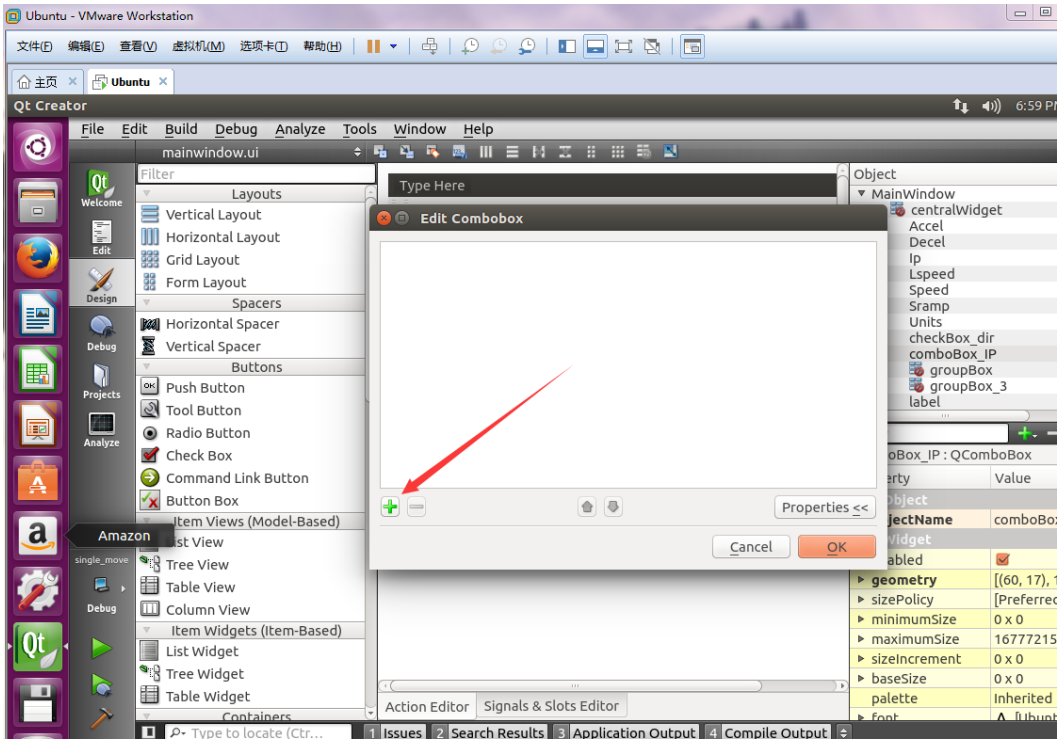


15) Click “run”, then program will appear.



16) IP can be added manually, open ui interface, double click button.

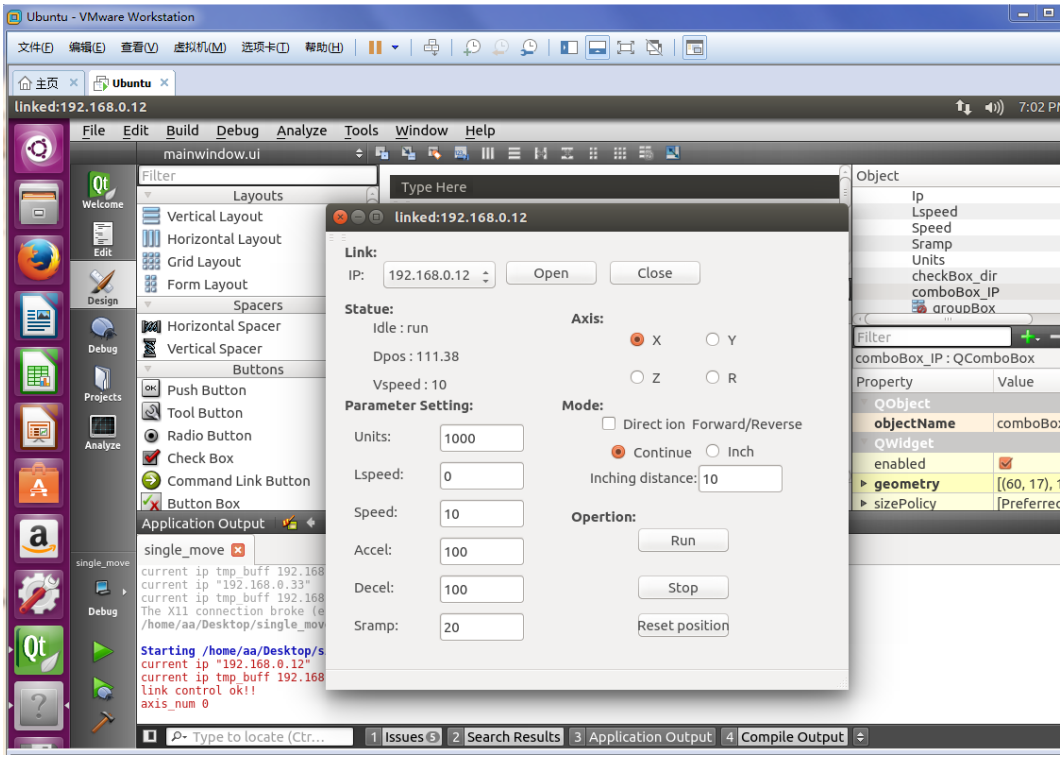




17) Controller's ip network segment is 192.168.0.1

The default IP of controller is 192.168.0.11

Next, you can use the route, click “open” to connect to IP, then click “Run.



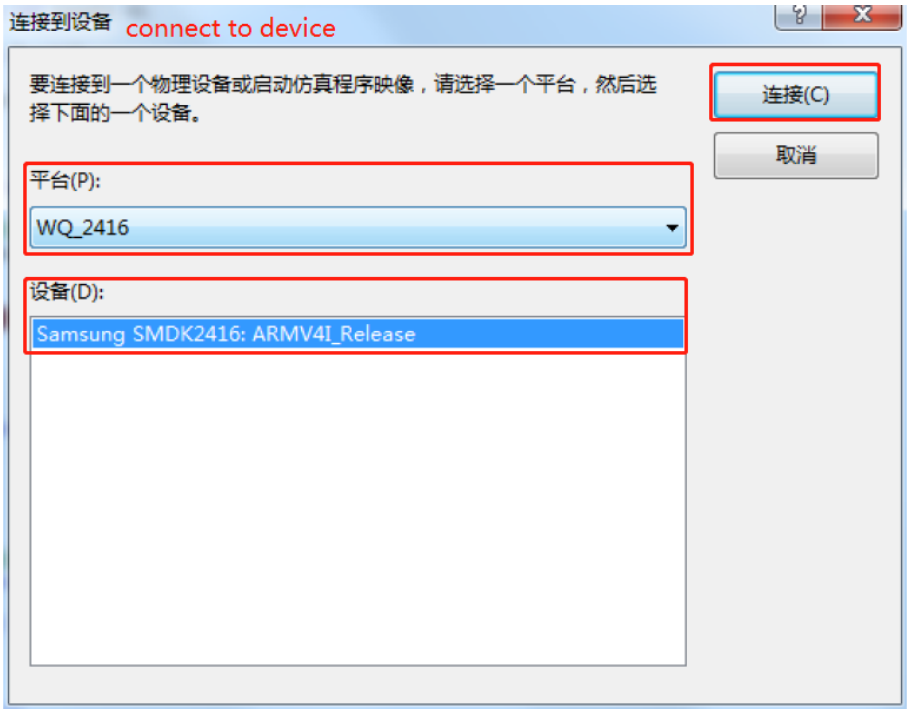
2.1.3.Wince System

2.1.3.1. C#

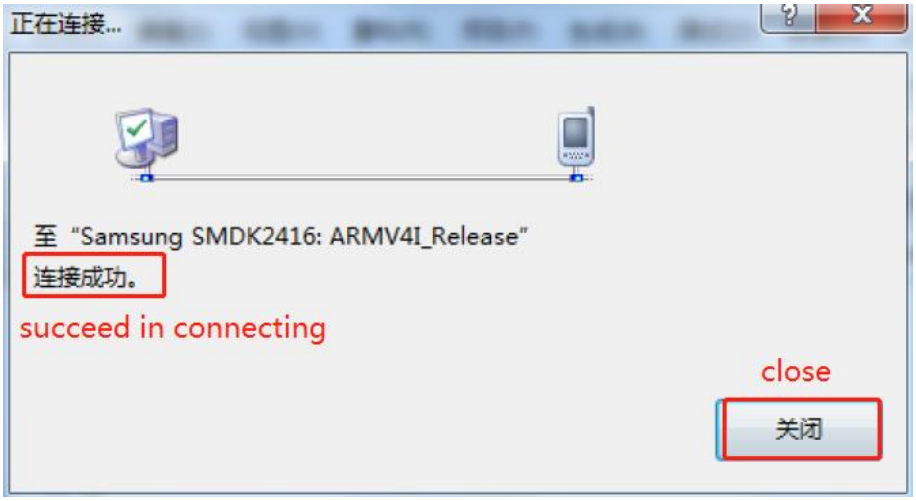
This route uses wince 6 mobile device as external connection device, and uses USB as connection main line.

1) Firstly, it needs to install the synchronization device driver and USB driver on the PC (not explained).
2) Then, use USB cable to build a connection with PC, device will connect to PC automatically. After connecting, Wince will show below information “Connected”.

3) Now, connect to VS2008:
   1> Click “tool” >> “connect to device”.
   2> Open below window, select “platform (P)” and “device (D)” at the same time, then click “connect” button.



3> After connecting successfully, see below:

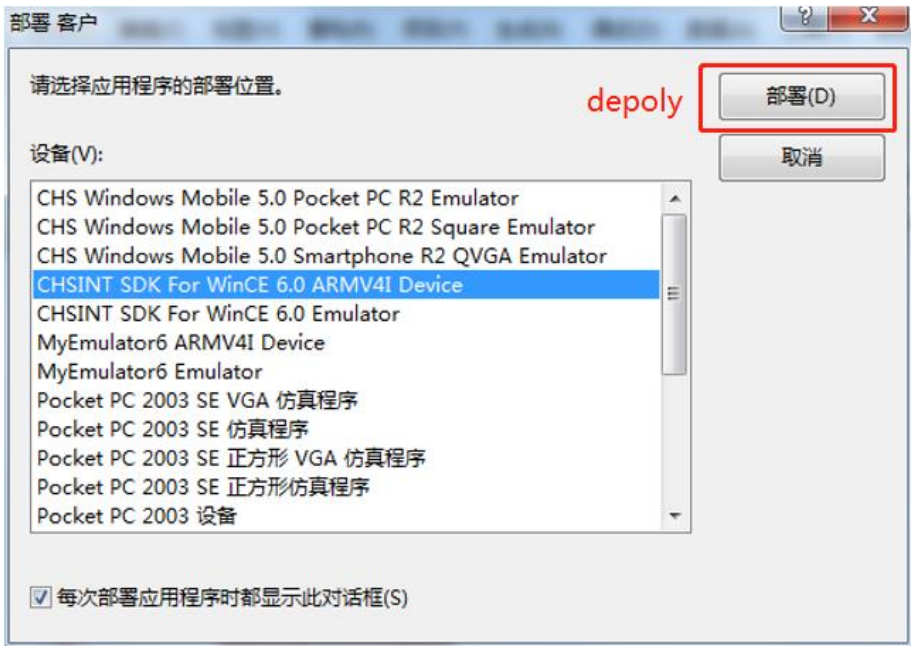


4> Click "close"/"OFF".

4) Start to debug / run program (here, debug is introduced).

5) Click "debug", then start to deploy platform.

6) Choose platform in image and click "deploy", then it will deploy automatically.



7) After deploying successfully, device will show the design window, it's time to debug and run.

8) After window code is finished, link wince6 and PC through USB, click “debug” or “run” to enter design window directly in device.

9) on wince6 interface, we can operate controller directly.
(**Note**: make sure IP of controller and Wince6 and PC are in the same segment. And when set Wince6 IP, don't set its gateway).

10) click “Search IP” to find available device IP, then select searched IP in the drop-down list, there needs 1~3 seconds, then choose required IP address to connect.

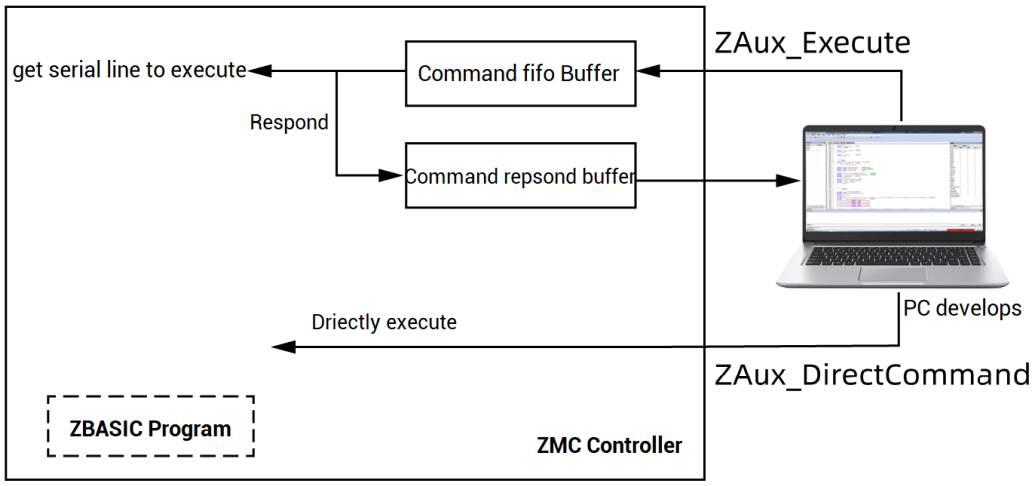
11) click “Connect”, it will link with selected IP address. If succeeds, it will display

"controller connected", or will display: "fail to connect, check IP".

12) Click "Axis X, Y, Z, R" separately to select motion axes, and they relate to "axis 0, 1, 2, 3", then we can choose motion mode "Jog or continuous", click "Move" to start move.
13) "Move / Stop / Position Clear" widgets are control buttons of the whole route, namely, they can control program run, stop, etc. (when controller is in disconnection, these key buttons will print "fail to connect with controller")
14) When operation is finished, click "disconnect" to break the link between device and controller.

2.2. Auxiliary Library Encapsulation

2.2.1. Methods of encapsulating auxiliary library



In ZAux library, motion functions and basic parameters are encapsulated, and open source codes can be viewed. Please refer to source codes for details.

ZAux library can send ZBasic instructions to controllers through ZAux_Execute or ZAux_Direct_Command. Relative functions can be referred by ZBASIC manual. If there is unencapsulated command used, or you want to encapsulate function by yourself, you can send through ZAux_Execute or ZAux_DirectCommand, or you can modify or add corresponding functions according to codes that are used now.

There are 2 ways to send character string command: buffer mode and direct mode.

Direction Mode:

Directly execute one single related command of variable / array / parameter,

now, all transferred parameters must be exact values (not expressions). Please refer to ZAux_DirectCommand function.

ZAux_Direct_Command(controller handle, command string, return string, length of return string).

Buffer Mode:

In this mode, all instructions can be executed, and expression can be as parameters, but speed will be slower, see ZAux_Execute for reference.

ZAux_Execute(controller handle, command string, return string, length of return string), check inputs and outputs of functions and parameters meaning or limitation.

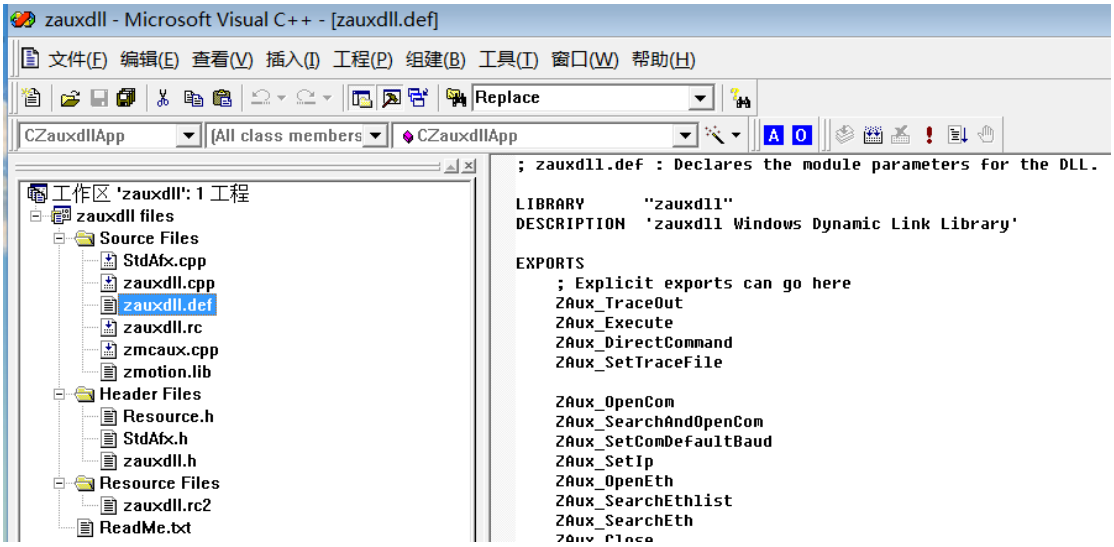
For example, set axis speed function: ZAux_Direct_SetSpeed(ZMC_HANDLE handle, int iaxis, float fValue), it is same as "SPEED(axis number)=value" in ZBASIC command. Then, when encapsulating function, the command string is "speed(%d) =%f".

Encapsulation Steps:

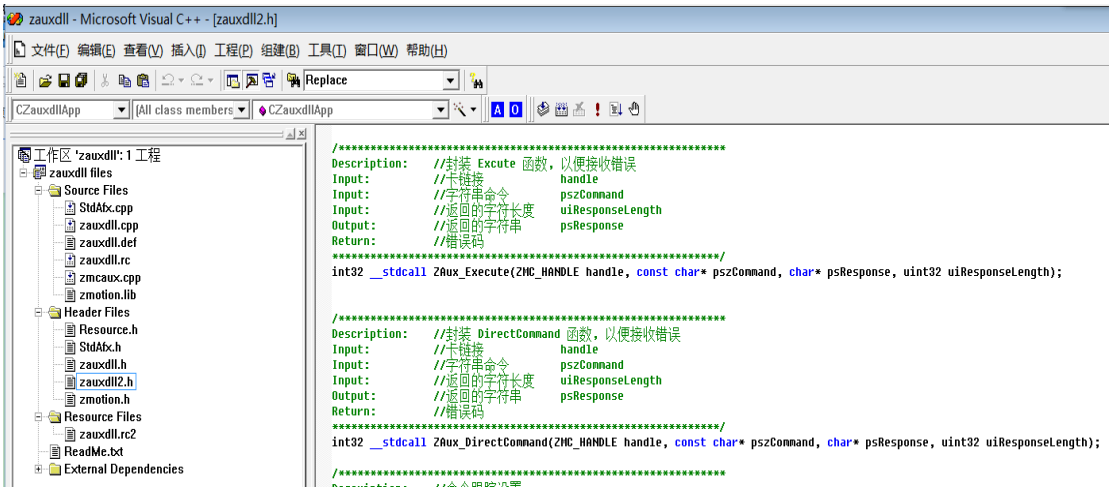
(1) Edit source code: open auxiliary source code, add relevant function in zmcaus.cpp

```
int32 __stdcall ZAux_Direct_SetSpeed(ZMC_HANDLE handle, int iaxis, float fValue)
{
    char  cmdbuff[2048];
    char  cmdbuffAck[2048];
    if( iaxis > MAX_AXIS_AUX)
    {
        return  ERR_AUX_PARAERR;
    }

  //generate the command
    sprintf(cmdbuff, "SPEED(%d)=%f", iaxis, fValue);

    //call the command to execute the function
    return ZAux_DirectCommand(handle, cmdbuff, cmdbuffAck, 2048);
}
```

(2) Define new added function definition in zauxdll.def.



(3) Add corresponding declaration in the header file.



(4) Compile new generated DLL.

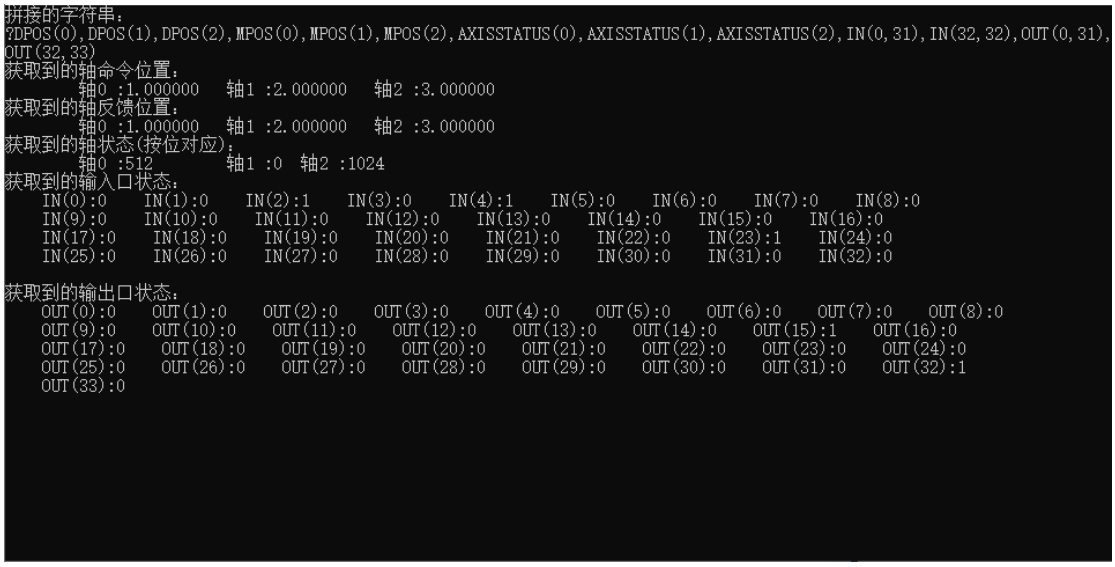
2.2.2. Encapsulation Examples

2.2.2.1. Get multi-type data directly

If the user wants to obtain a variety of data, such as the command position of the axis, the feedback position of the axis, the IO points on the board, etc., they often obtain different data through a variety of separate and independent functions. Such accumulation (so many times of reading and writing) will cause the program to freeze. In order to improve the performance of a high-level program, it is often possible to define a function of its own, and quickly transfer data to the high-level program through

a function, instead of obtaining different types of data through multiple cycles.

For example: suppose there is a simple three-axis platform, it is necessary to read the command position of axis 0, axis 1 and axis 2, the feedback position, and the input port 0-input port 32, output port 0-output port 33 on the controller board, and the state of the three axes. The program for obtaining data is as follows:




```
// test1.cpp: define the entry point of control panel application program.
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//if it is not 0, it fails
	{
		printf("%s fail!return code is %d\n", command, ret);
	}

}

/*************************************************************
Description:	//directly get data function (by own customized)
Input:		//handle		card link
			iaxisNum		total numbers of axes
			iaxislist		axis No. list
			fDposlist		output demand position value (DPOS)
			fMposlist		output measurement position value (MPOS)
			iAxisstatuslist		output axis state position value, bit to bit
			startIn			starting IN No. to be got
			endIn			end IN No. to be got
			iIn			output IN state, bit to bit
			startOut		starting OUT No. to be got
```

```
                    endOut              end OUT No. to be got
                    iOut                output OUT state, bit to bit
Output:     //
Return:     //error codes
************************************************************/
int  Demo_Direct_MyGetData(ZMC_HANDLE handle,int iaxisNum, int* iaxislist, float*
fDposlist,float* fMposlist,int32* iAxisstatuslist,int startIn , int endIn,int *iIn,int startOut ,
int endOut,int *iOut)
{
    char cmdbuff[2048];
    char  tempbuff[2048];
    char cmdbuffAck[20480];

    //If the transferred address is blank, exit.

    if(NULL == iaxislist || NULL == fDposlist || NULL == fMposlist ||  NULL ==
iAxisstatuslist || NULL == iIn || NULL == iOut)
    {
        return  ERR_AUX_PARAERR;
    }
    //If transferred end No. is smaller than starting No., exit.
    if ((endIn<startIn)  ||  (endOut<startOut))
    {
        return  ERR_AUX_PARAERR;
    }

    int ret=0;
    int i;
    //generate the command
    sprintf(cmdbuff, "?");
    //splice DPOS
    for (i=0;i<iaxisNum;i++)
    {
        sprintf(tempbuff,"DPOS(%d),",iaxislist[i]);//generate character string that
corresponds to the command

        strcat(cmdbuff, tempbuff);//splice character string

        if (strlen(cmdbuff)>1000)
        {
            return  ERR_AUX_PARAERR;    //parameter is wrong, character string
splicing is too long
        }
    }
    //splice MPOS
    for (i=0;i<iaxisNum;i++)
    {
        sprintf(tempbuff,"MPOS(%d),",iaxislist[i]);//generate character string that
corresponds to the command

        strcat(cmdbuff, tempbuff);//splice character string

        if (strlen(cmdbuff)>1000)
        {
            return  ERR_AUX_PARAERR;    //parameter is wrong, character string
```

```
splicing is too long
            }
        }
        //splice AXISSTATUS
        for (i=0;i<iaxisNum;i++)
        {
            sprintf(tempbuff,"AXISSTATUS(%d),",iaxislist[i]);//generate character string that
corresponds to the command


            strcat(cmdbuff, tempbuff);//splice character string
            if (strlen(cmdbuff)>1000)
            {
                return  ERR_AUX_PARAERR;     //parameter is wrong, character string
splicing is too long

            }
        }
        int32 ostart,istart,iend,oend;            //32 at most in once time
        bool addflag;

        addflag=false;

        int32 temp;         //32 at most in once time
        int32 temp2;        //32 at most in once time

        temp=endIn-startIn+1;
        if (temp%32 == 0)
        {
            temp=temp/32;
        }
        else
        {
            temp=temp/32+1;
        }

        //splice IN
        for (i=0;i<temp;i++)
        {
            istart = startIn+32*i;
            iend =istart+31;
            if (iend>endIn)
            {
                iend=endIn;
            }
            //generate the command
            sprintf(tempbuff, "IN(%d,%d),", istart,iend);
            strcat(cmdbuff, tempbuff);//splice character string
            if (strlen(cmdbuff)>1000)
            {
                return ERR_AUX_PARAERR ;     //parameter is wrong, character string
splicing is too long
            }

        }
```

```
    temp2=endOut-startOut+1;
    if (temp2%32 == 0)
    {
        temp2=temp2/32;
    }
    else
    {
        temp2=temp2/32+1;
    }
    //splice OUT
    for (i=0;i<temp2;i++)
    {
        ostart = startOut+32*i;
        oend =ostart+31;
        if (oend>endOut)
        {
            oend=endOut;
        }


        // generate the command
        sprintf(tempbuff, "OUT(%d,%d)", ostart,oend);

        strcat(cmdbuff, tempbuff);//splice character string
        if (i<temp-1)
        {
            strcat(cmdbuff, ",");//splice character string
        }

        if (strlen(cmdbuff)>1000)
        {
            return  ERR_AUX_PARAERR;     //parameter is wrong, character string
splicing is too long
        }

    }

    printf("spliced string：\n",cmdbuff);
    printf("%s\n",cmdbuff);

    ret=ZAux_DirectCommand(handle,cmdbuff,cmdbuffAck,2048);
    if(ERR_OK != ret)
    {
        return ret;
    }
    //printf("%s\n",cmdbuffAck);
    //printf("%d\n",strlen(cmdbuffAck));
    //
    if(0 == strlen(cmdbuffAck))
    {
        return ERR_NOACK;
    }
    float ftempbuff[200];
    int itempbuff[200];
```

```
	ZAux_TransStringtoFloat(cmdbuffAck,iaxisNum*2,ftempbuff);//character string is converted to floating
	//DPOS output
	for(i=0;i<iaxisNum;i++)
	{
		//printf("%f\n",ftempbuff[i]);
		fDposlist[i]=ftempbuff[i];

	}
	//MPOS output
	for(i=0;i<iaxisNum;i++)
	{
		//printf("%f\n",ftempbuff[i+iaxisNum]);
		fMposlist[i]=ftempbuff[i+iaxisNum];
	}

	ZAux_TransStringtoInt(cmdbuffAck,iaxisNum*3+temp2+temp,itempbuff);//character string is converted to integer
	//AXISSTATUS output
	for(i=0;i<iaxisNum;i++)
	{
		//printf("%d\n",itempbuff[i+iaxisNum*2]);
		iAxisstatuslist[i]=itempbuff[i+iaxisNum*2];
	}
	//IN output
	for(i=0;i<temp;i++)
	{
		//printf("%d\n",itempbuff[i+iaxisNum*3]);
		iIn[i]=itempbuff[i+iaxisNum*3];
	}
	//OUT output
	for(i=0;i<temp2;i++)
	{
		//printf("%d\n",itempbuff[i+iaxisNum*3+temp]);
		iOut[i]=itempbuff[i+iaxisNum*3+temp];
	}
	return ERR_OK;
}
int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;			//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("fail to connect controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("success to connect controller！ \n");

	int axis[4]={0,1,2,4};
	float d_dpos[4];
```

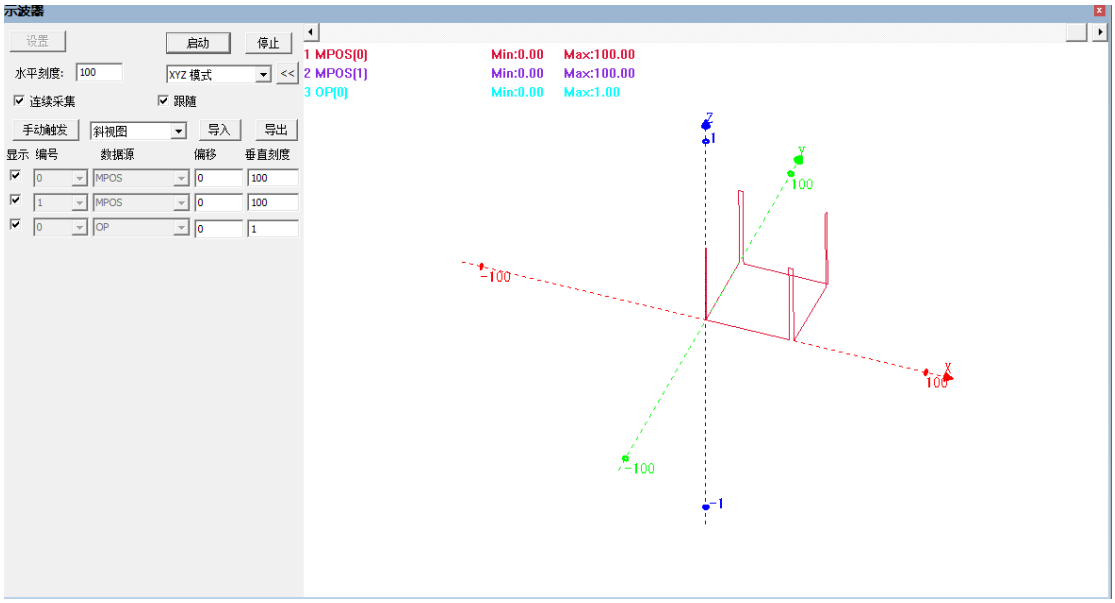
```
	float d_mpos[4];
	int32 d_axis_status[4];
	int d_in[10];
	int d_out[10];

	ret =
Demo_Direct_MyGetData(handle,3,axis,d_dpos,d_mpos,d_axis_status,0,32,d_in,0,33,d_ou
t);
	int i;
	printf("obtained DPOS：\n");
	for (i=0;i<3;i++)
	{
		printf("\taxis%d :%f",i,d_dpos[i]);
	}
	printf("\n");
	printf("obtained MPOS：\n");
	for (i=0;i<3;i++)
	{
		printf("\taxis%d :%f",i,d_mpos[i]);
	}
	printf("\n");
	printf("obtained axis state(bit to bit)：\n");
	for (i=0;i<3;i++)
	{
		printf("\taxis%d :%d",i,d_axis_status[i]);
	}
	printf("\n");

	printf("obtained IN state：\n");
	int j=0;
	int tempval;
	for (i=0;i<=32;i++)
	{
		if (((i%32)==0)&&(i>0) )
		{
			j++;
		}
		//convert to bit
		tempval=d_in[j]>>(i-32*j);
		printf("    IN(%d):%d",i,tempval &(0x01));
		if (((i%8)==0)&&(i>0) )
		{
			printf("\n");
		}
	}
	printf("\n");

	printf("obtained OUT state：\n");
	j=0;

	for (i=0;i<=33;i++)
	{
		if (((i%32)==0)&&(i>0) )
		{
```

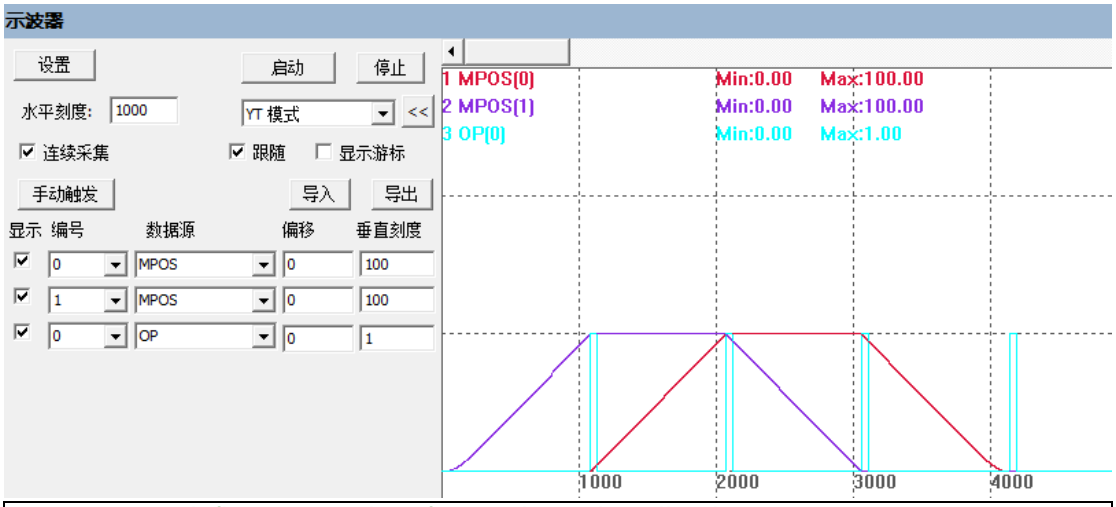
```
                j++;
            }
            // convert to bit
            tempval=d_out[j]>>(i-32*j);
            printf("    OUT(%d):%d",i,tempval &(0x01));
            if (((i%8)==0)&&(i>0) )
            {
                printf("\n");
            }
        }
        printf("\n");

        Sleep(20000);
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;//judge whether command is executed
successfully or not
        printf("connection closed!\n");

        handle = NULL;
        return 0;
}
```

2.2.2.2. Buffer Command: One Command Executes Multi-Type Commands

Generally, the dispensing industry and the woodworking industry are mostly used in the continuous track, and there is an inserted buffer output between the continuous tracks. If the motion and the continuous track motion and the buffer output are sent separately, there will inevitably be limitations. Through encapsulating the motion function separately to achieve the effect of executing multiple functions in one line of commands, as shown in the following example:

Assuming to control an XY two-axis platform, from the coordinate point (0,0) -> (100,0) (output port 0 output 50ms) -> (100,100) (output port 0 output 50ms) -> (0,100) (output port 0 output 50ms) -> (0,0) (output port 0 output 50ms), you can package it by yourself and use a function to send it quickly:






```
// test1.cpp: define entry point of control panel application program.
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//if it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}

}

/*************************************************************
Description:	//customized motion function
Input:		//handle		card link/handle
			iMoveLen		filled motion length
```

```
                    iaxisNum        the total axes in motion
                    iaxislist       axis No. list
                    fPoslist        distance list
                    iout            buffer output port
                    outlist         buffer output list (each motion, whether outputs, 0
means not to output, 1 means output after motion)
                    outtime         buffer output time

Output:     //
Return:     //error code
************************************************************/
int  Demo_Direct_MyMoveABS(ZMC_HANDLE handle,int iMoveLen,int iaxisNum, int*
iaxislist, float* fPoslist,int iout,int *outlist,int outtime)
{
    char cmdbuff[2048];
    char  tempbuff[2048];
    char cmdbuffAck[20480];

    //if transferred address is blank, exit


    int ret=0;
    int i;
    //first to read remain linear buffer
    int iBuffLen = 0;
    ret = ZAux_Direct_GetRemain_LineBuffer(handle,iaxislist[0],&iBuffLen);
    if(iBuffLen <= iMoveLen*2)
    {
        return 1002;            //motion buffer is not enough
    }

    //generate the command
    sprintf(cmdbuff, "BASE(");
    //splice motion axis list
    for (i=0;i<iaxisNum-1;i++)
    {
        sprintf(tempbuff,"%d,",iaxislist[i]);//generate the character string that
corresponds to the command

        strcat(cmdbuff, tempbuff);//splice the character string

        if (strlen(cmdbuff)>1000)
        {
            return  ERR_AUX_PARAERR;    //parameter is wrong, character string
splicing is too long
        }
    }
    sprintf(tempbuff,"%d)\n",iaxislist[i]);//generate the character string that
corresponds to the command
    strcat(cmdbuff,tempbuff);

    //splice the motion
    for (i=0;i<iMoveLen;i++)
    {
        //printf("%d,%d\n",i*iaxisNum,i*iaxisNum+1);
```

```
            if (outlist[i]==0)    //this motion doesn't output
            {
                strcat(cmdbuff, "MoveAbs(");

        sprintf(tempbuff,"%f,%f)\n",fPoslist[i*iaxisNum],fPoslist[i*iaxisNum+1]);//generate
the character string that corresponds to the command
                strcat(cmdbuff, tempbuff);//splice the character string

            }
            else if (outlist[i]==1)
            {
                    strcat(cmdbuff, "MoveAbs(");

        sprintf(tempbuff,"%f,%f)\n",fPoslist[i*iaxisNum],fPoslist[i*iaxisNum+1]);//generate
the character string that corresponds to the command
                    strcat(cmdbuff, tempbuff);//splice the character string

                    strcat(cmdbuff, "Move_op2(");
                    sprintf(tempbuff,"%d,%d,%d)\n",iout,1,outtime);//generate the
character string that corresponds to the command
                    strcat(cmdbuff, tempbuff);//splice the character string
            }
            else
            {
                return  ERR_AUX_PARAERR;//parameter is wrong
            }


        }


        printf("spliced string：\n");
        printf("%s\n",cmdbuff);

        if (strlen(cmdbuff)>1000)
        {
            return  ERR_AUX_PARAERR;     // parameter is wrong, character string
splicing is too long
        }
        ret=ZAux_DirectCommand(handle,cmdbuff,cmdbuffAck,2048);

        return ret;


}
int _tmain(int argc, _TCHAR* argv[])
{
        char *ip_addr = (char *)"127.0.0.1";        //Controller IP address
        ZMC_HANDLE handle = NULL;           //link handle
        int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
        if (ERR_SUCCESS != ret)
        {
            printf("fail to connect controller!\n");
            handle = NULL;
            Sleep(2000);
```

```
            return -1;
        }
        printf("success to connect controller!\n");

        ret =ZAux_Direct_SetAtype(handle,0,1);//set axis type of axis 0 as 1
        commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//judge whether the
command is executed successfully.
        ret =ZAux_Direct_SetAtype(handle,1,1);// set axis type of axis 1 as 1
        commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//judge whether the
command is executed successfully.

     ret =ZAux_Direct_SetUnits(handle,0,100);//set pulse amount (UNITS) of axis 0 as100
        commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//judge whether the
command is executed successfully.
        ret =ZAux_Direct_SetUnits(handle,1,100);// set pulse amount (UNITS) of axis 1
as100
        commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//judge whether the
command is executed successfully.

        ret =ZAux_Direct_SetAccel(handle,0,500);//set axis 0 acceleration
        commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//judge whether the
command is executed successfully.
        ret =ZAux_Direct_SetAccel(handle,1,500);//set axis 1 acceleration
        commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//judge whether the
command is executed successfully.

        ret =ZAux_Direct_SetDecel(handle,0,500);//set axis 0 deceleration
        commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
command is executed successfully.
        ret =ZAux_Direct_SetDecel(handle,1,500);//set axis 1 deceleration
        commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
command is executed successfully.

        ret =ZAux_Direct_SetDpos(handle,0,0);//set DPOS clearing (0) of axis 0
        commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
command is executed successfully.
        ret =ZAux_Direct_SetDpos(handle,1,0);//set DPOS clearing (0) of axis 1
        commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
command is executed successfully.

        ret =ZAux_Direct_SetSpeed(handle,0,100);//set axis 0 speed
        commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
command is executed successfully.

        ret =ZAux_Direct_SetSpeed(handle,1,100);// set axis 1 speed
        commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
command is executed successfully.

        ret =ZAux_Direct_SetMerge(handle,0,1);//set opening continuous interpolation
(only main axis is opened, such as, axis 0 and axis 1 interpolation, axis 0 is the master
axis, the main axis No. is the first axis No. in continuous interpolation motion
instructions)


        int axis[2]={0,1};
```

```
	float POS[12]={0,0,0,100,100,100,100,0,0,0};
	int otlist[5]={0,1,1,1,1};

	ZAux_Trigger(handle);//trigger the oscilloscope

	ret = Demo_Direct_MyMoveABS(handle,5,2,axis,POS,0,otlist,50);//
	commandCheckHandler("Demo_Direct_MyMoveABS", ret) ;//judge whether the command is executed successfully.

	Sleep(20000);
	ret = ZAux_Close(handle);    //close the connection
	commandCheckHandler("ZAux_Close", ret) ;//judge whether the command is executed successfully.

	printf("connection closed!\n");

	handle = NULL;
	return 0;
}
```

Chapter III Board Card Connection Initialization

3.1.Link Controller

3.1.1.Emphasis

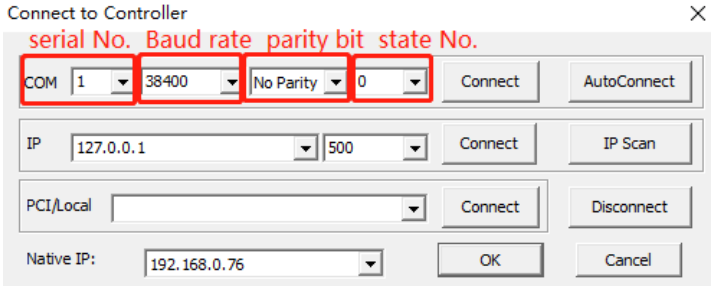
3.1.1.1. General instructions

- **Serial Port:**

  Controller default parameters:

  Boolean rate: 38400, data bit: 8, stop bit: 1, no parity.

  Note: serial port parameters on computer of controller should be same as serial port parameters of controller, then connection can be built.



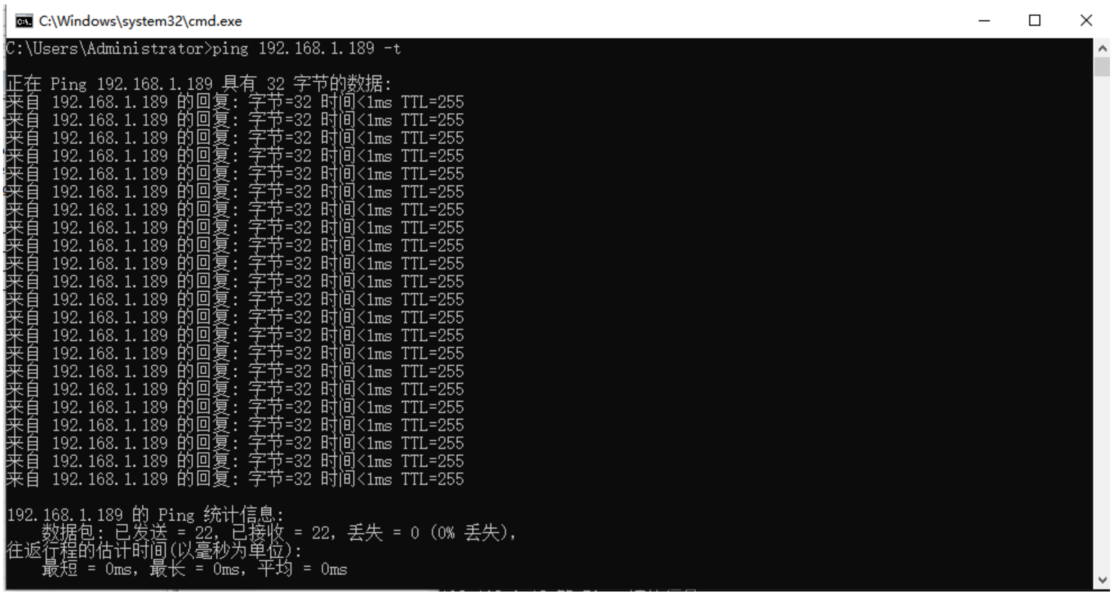
  State No.: when use 485 serial port, there may have several slave stations, through this state No. to select, usually it is ok to fill 0.

- **EtherNET:**

The network port adopts RJ45 standard network cable interface, and the communication rate is 100Mbit/s. The controller comes with a network port, but it supports at least 2 network port channels. An interchanger can be used to expand the network port, and the communication channels can be checked (printed) through "/port".

The factory IP address of the controller is 192.168.0.11, and the port number is 502. When communicating, ensure that the device is in the same network segment (the first three segments are the same), and the IP address can be modified. The network port is often used to connect to a computer or touch screen, and the computer is usually connected to the controller through the network port, if you forget the IP of the controller, you can use the serial port to connect to view or modify it.

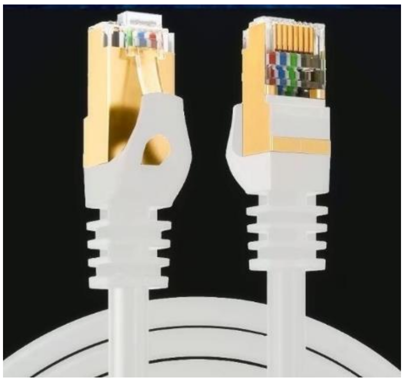
Ethernet connection ability and performance:



Cable recommendations:

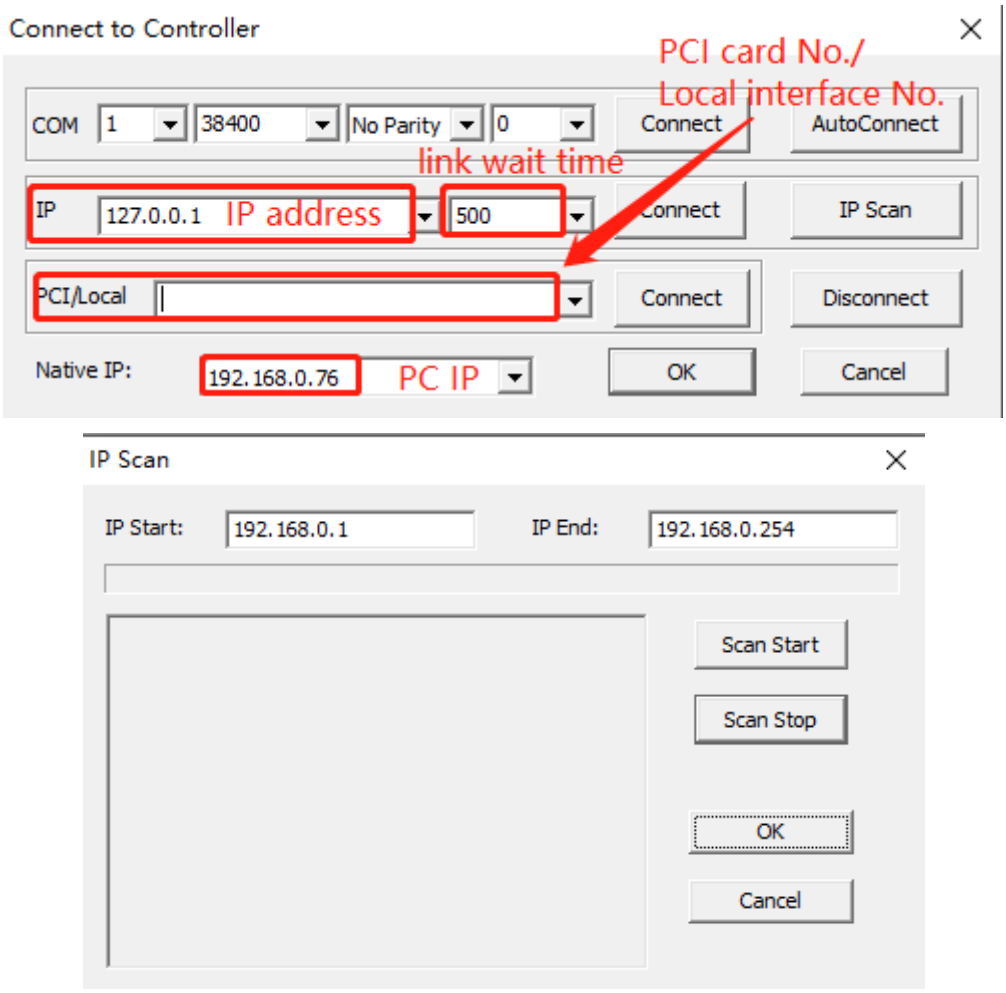
Ethernet cable Type	**Transfer Frequency**	**Transfer Velocity**	**Transfer Distance**	**Cable Type**
CAT5	100Mhz	100Mbps	100m	Shielded/non-shielded
CAT5E	100Mhz	1000Mbps	100m	Shielded/non-shielded
CAT6	250Mhz	1Gbps	100m	Shielded/non-shielded
CAT6E	500Mhz	10Gbps	100m	Shielded/non-shielded
CAT7	600Mhz	10Gbps	100m	Pair shielded
CAT8	2000Mhz	40Gbps	30m	Pair shielded

In order to ensure the quality of communication, it is better to use a double-shielded network cable with a metal head above Category 7 in occasions with large interference. In normal occasions, use a network cable with a shielded metal head of Category 5e. As shown in the figure below: the network cable with metal wire can play the role of grounding protection. If a network cable without a metal headband shielding layer is used in an actual interference site, uncertain things such as packet loss will occur.



Check the return value of the function to judge whether the function is executed

successfully. If it returns 0, it is successful, and if it is not 0, it fails. When the Ethernet port connection returns unsuccessful, check whether the transmitted IP is correct, and ensure that the computer IP address and the controller IP address are in the same network segment.



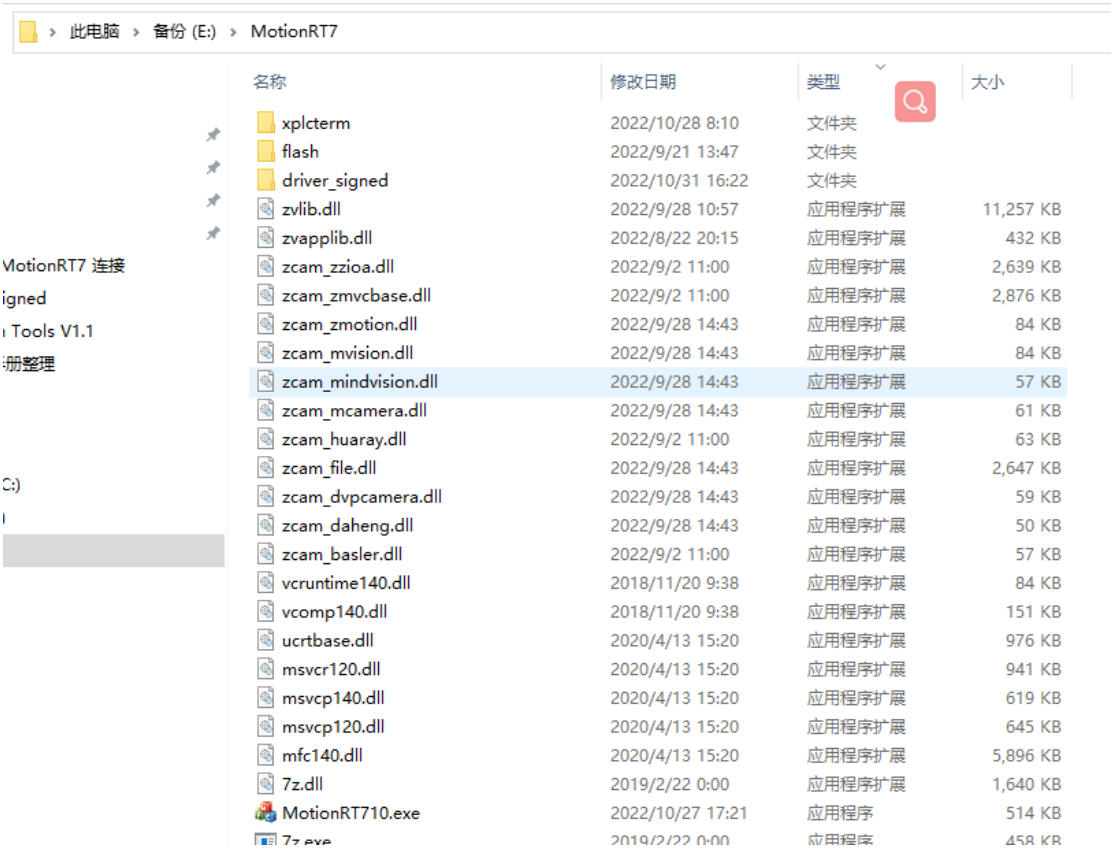
- Connection waiting time: the network port connection is valid. If the response time is short, the value can be increased appropriately to extend the response time
- Start IP, End IP: fill in the corresponding network segment according to the network segment of the control IP set by yourself.
- Station No.: If the serial port connection uses 485, there may be multiple slave stations, which can be selected by station number.
- PCI card number: PCI type control card slot card number (you need to install the driver to scan out).

3.1.1.2. MotonRT7

MotionRT710 is recommended to be used with XPCI/XPCIE control card, which can

give full play to the performance, and the authorization information is stored in the control card. Each control card has a unique number on the card.

Prepare the latest version of the MotionRT7 installation package and decompress it before installation. The figure below shows the contents of the files after decompression of the MotionRT7 installation package.



✧ Driver_signed: The driver folder, which contains MotionRT7 driver installation information files, ECAT protocol installation information, installation wizard software, and may also include security directory files, sys system files, and signature files.

- Dpinst_amd64.exe: installation wizard software.
- ZMotionRT64.cat: digital signature file of the driver.
- ZMotionRT64.inf: MotionRT7 software driver installation information, browse

the folder to select the file when installing the driver.

- ZMotionRT64.sys: system file.
- ZMotionRTPacket.inf: bus protocol installation information, install the EACT bus protocol, browse folder to select this file.

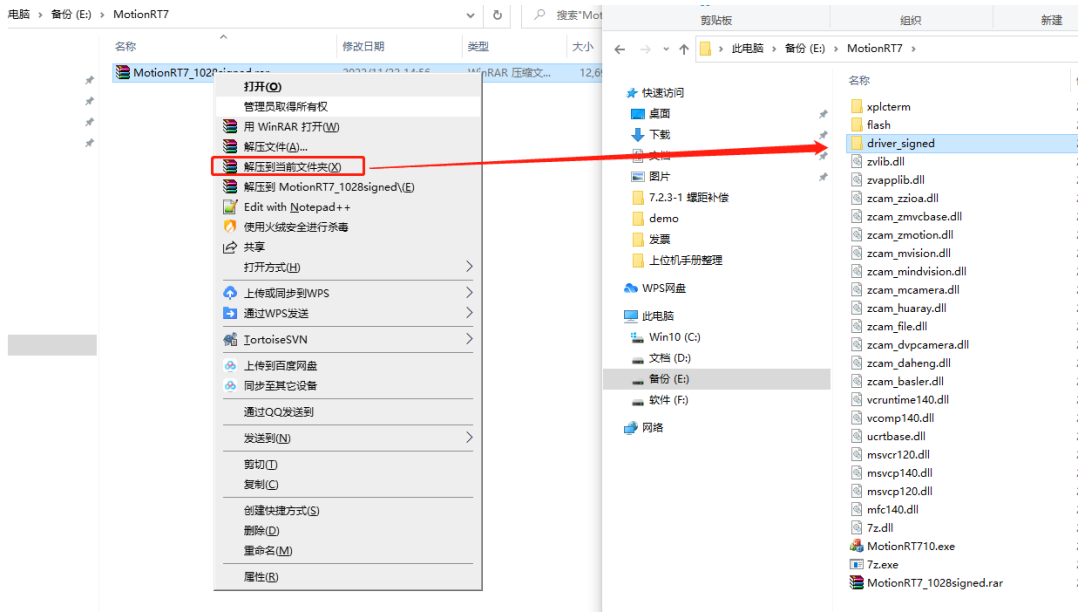
✧ flash: controller system folder.

✧ Udisk: U disk folder.

✧ Xplcterm: xplc screen folder, which contains xplcterm software, which is displayed as a screen when using HMI.

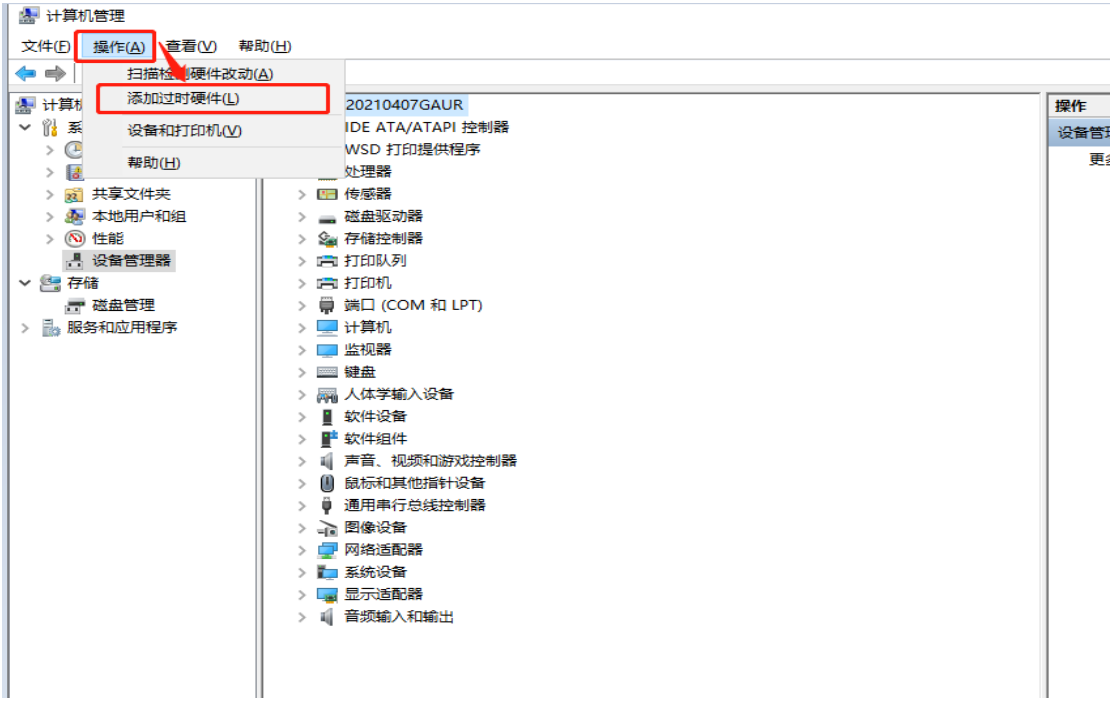
➢ **MotionRT7 Drive Installation:**

After obtaining the installation package of MotionRT7, unzip the installation package to the location of your choice, as shown below:



3.1.1.2.1. When there is no PCI card device.

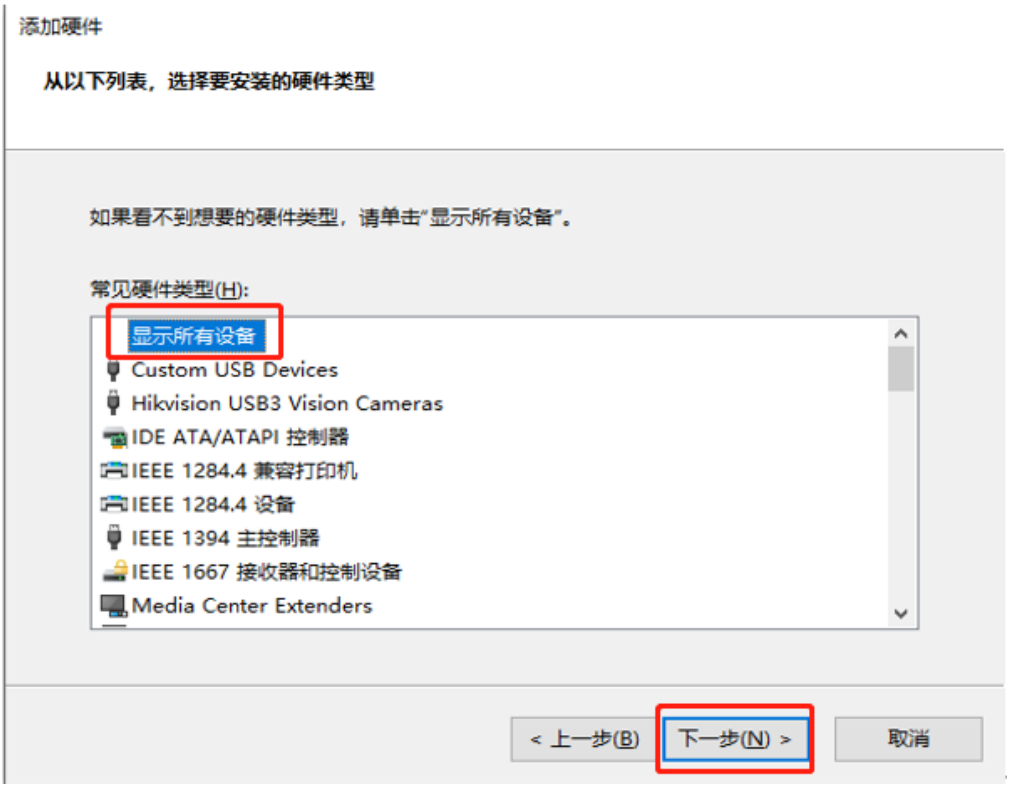
1. In the device manager, the menu: "Operation" - "Add Obsolete Hardware", if there is no "Operation", right click to add it.



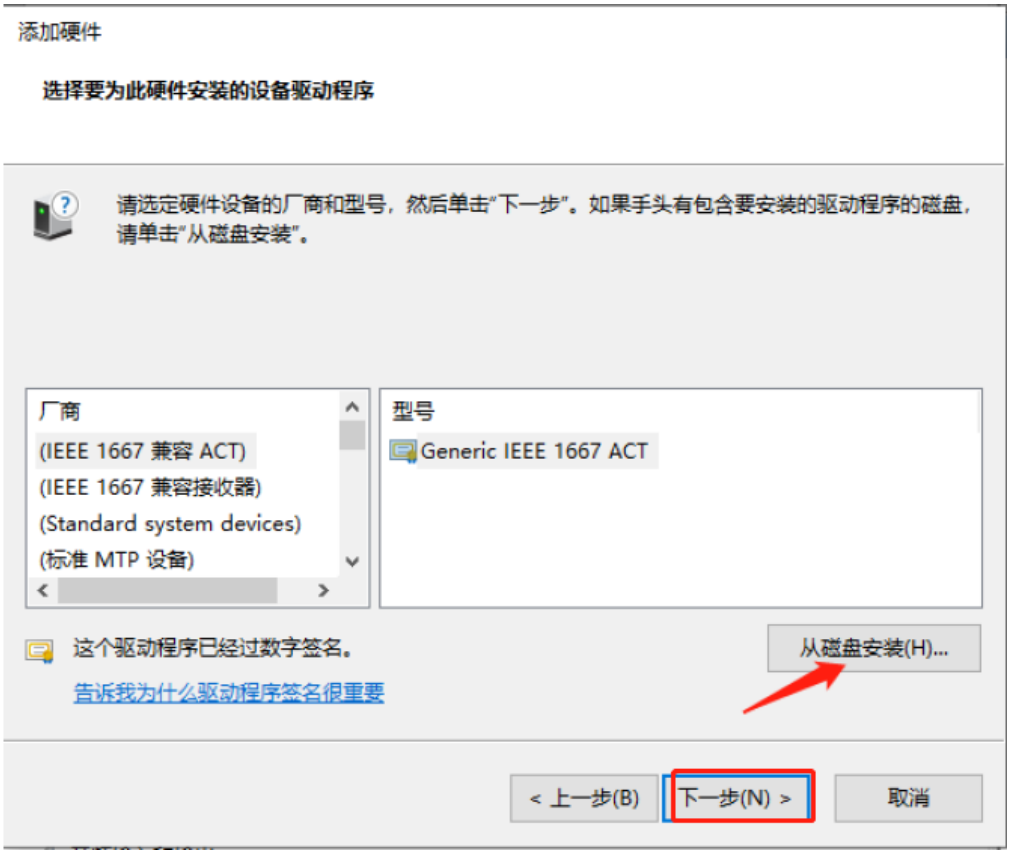
2. After that, below interface will pop up, click “next".

3. Then, in below window, select the second one (install the hardware that is selected from the list manually).

4. Select “display all devices”, then click “next”.



5. Choose to install from disk, (all options are default items, no need to choose manufacturer and model)



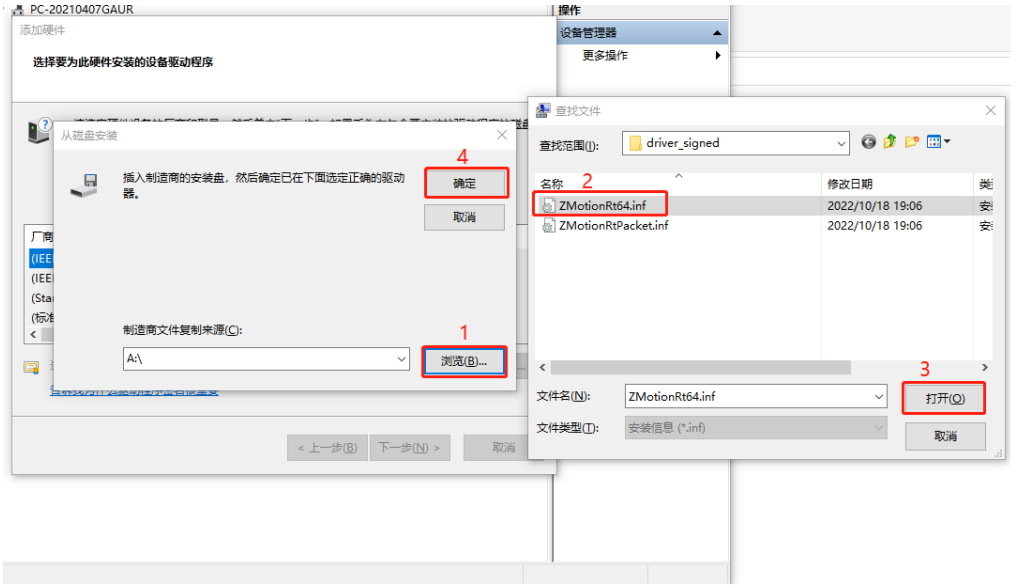
6. Then, below window appears:

   (1) Click "browse".

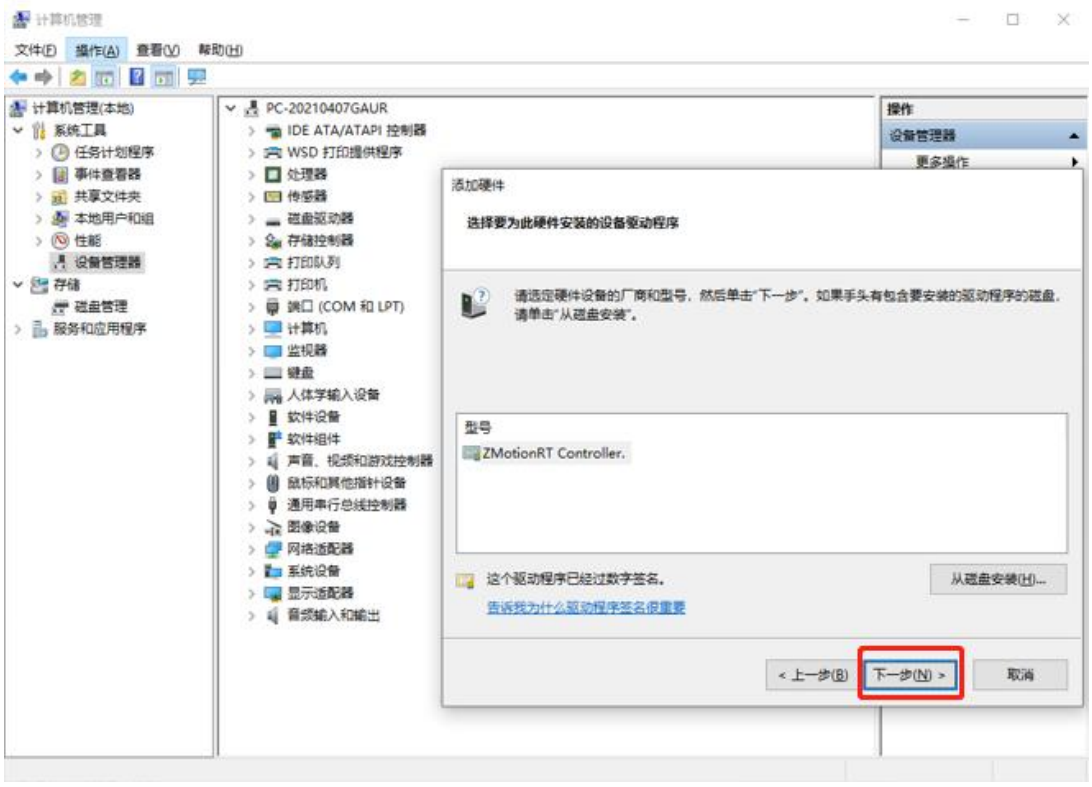
   (2) Open the file path MotionRT7\driver_signed\ZMotionRt64.inf under MotionRT7.

   (3) Open ZMotionRt64.inf file.

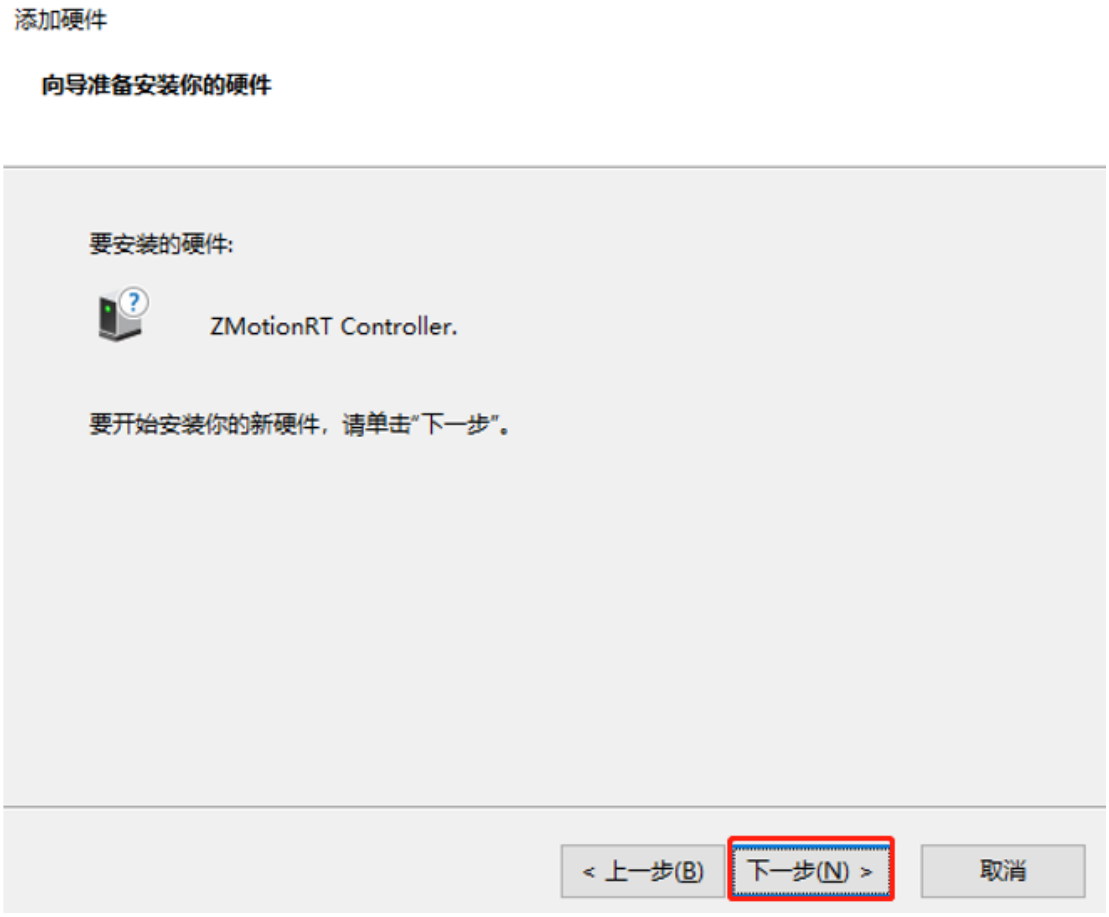
   (4) Click "OK".



7. After that, directly click "Next".

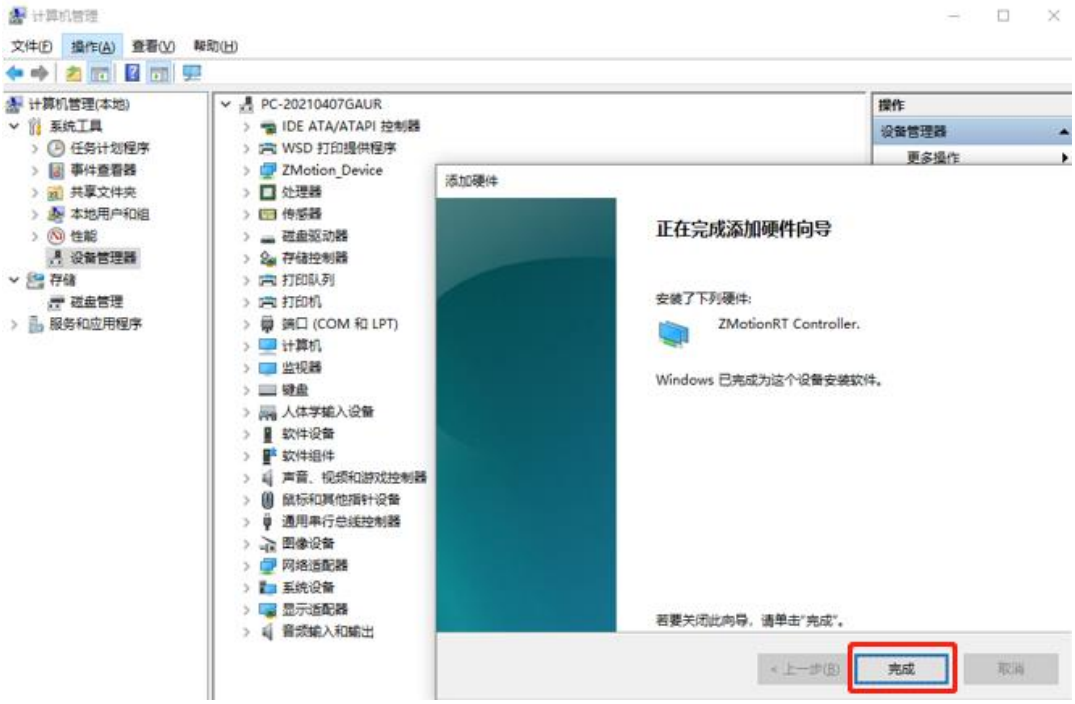


8. The same, click “Next”.



9. The last step, click “complete”. If there is ZMotionRT Controller in the device

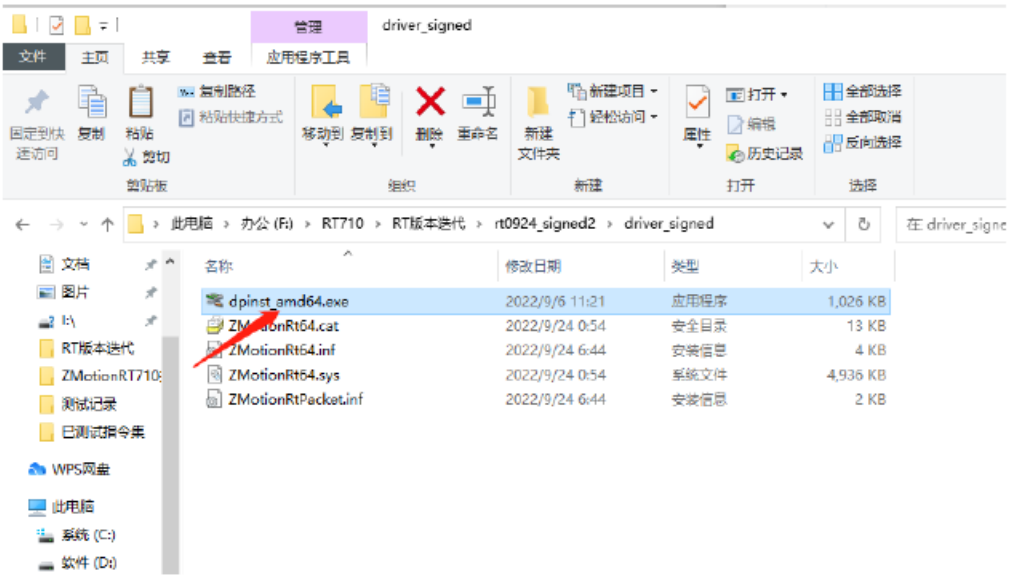
manager, the installation is successful.



3.1.1.2.2. When there is the PCI card device.

Method 1: install automatically

Use the built-in installation wizard software dpinst_amd64.exe in the driver directory to automatically install, and the specific operation is according to the software guide.

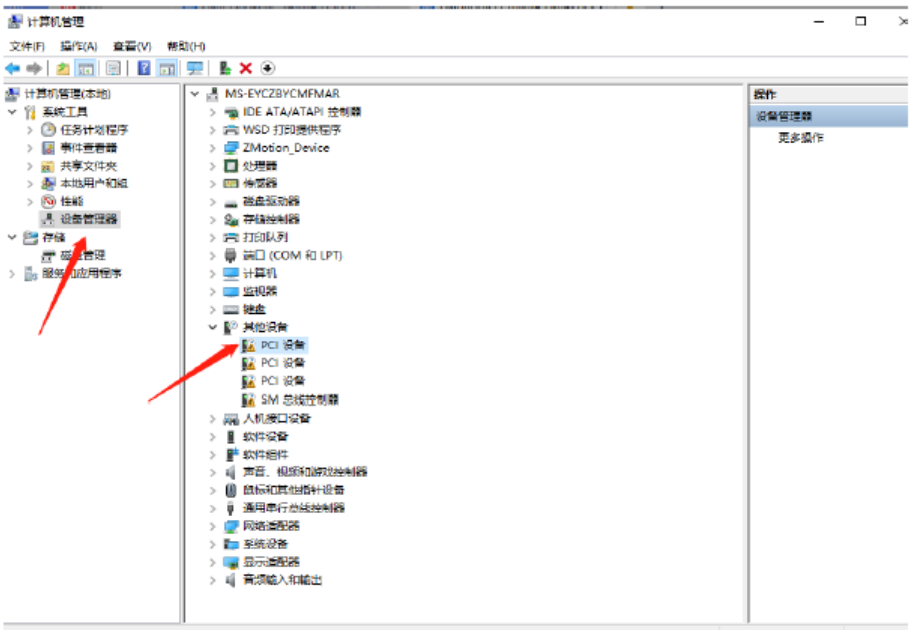


Note: If there is no PCI device, the software cannot be installed successfully, only the

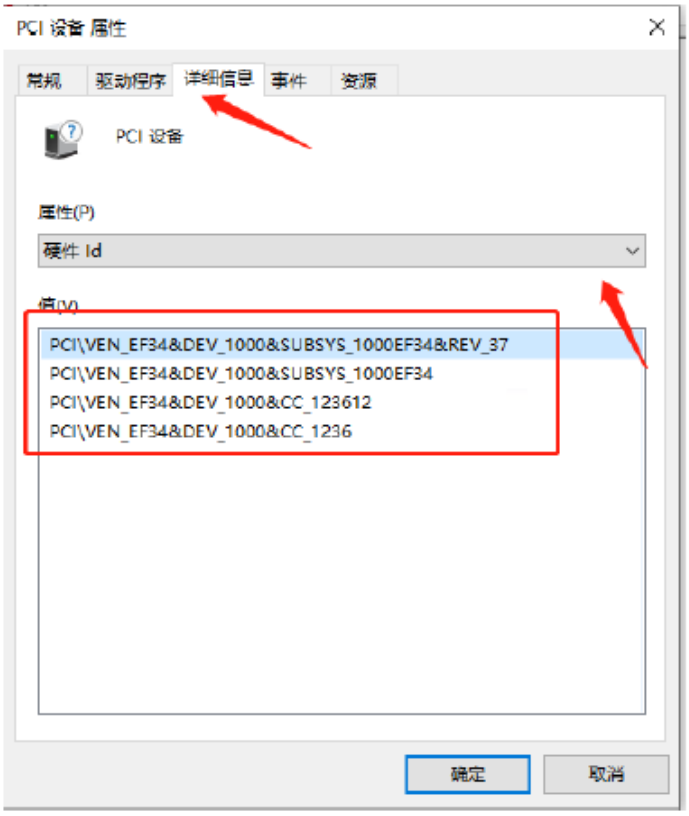
ZMotionRT64.sys file can be installed! !

Method 2: install manually

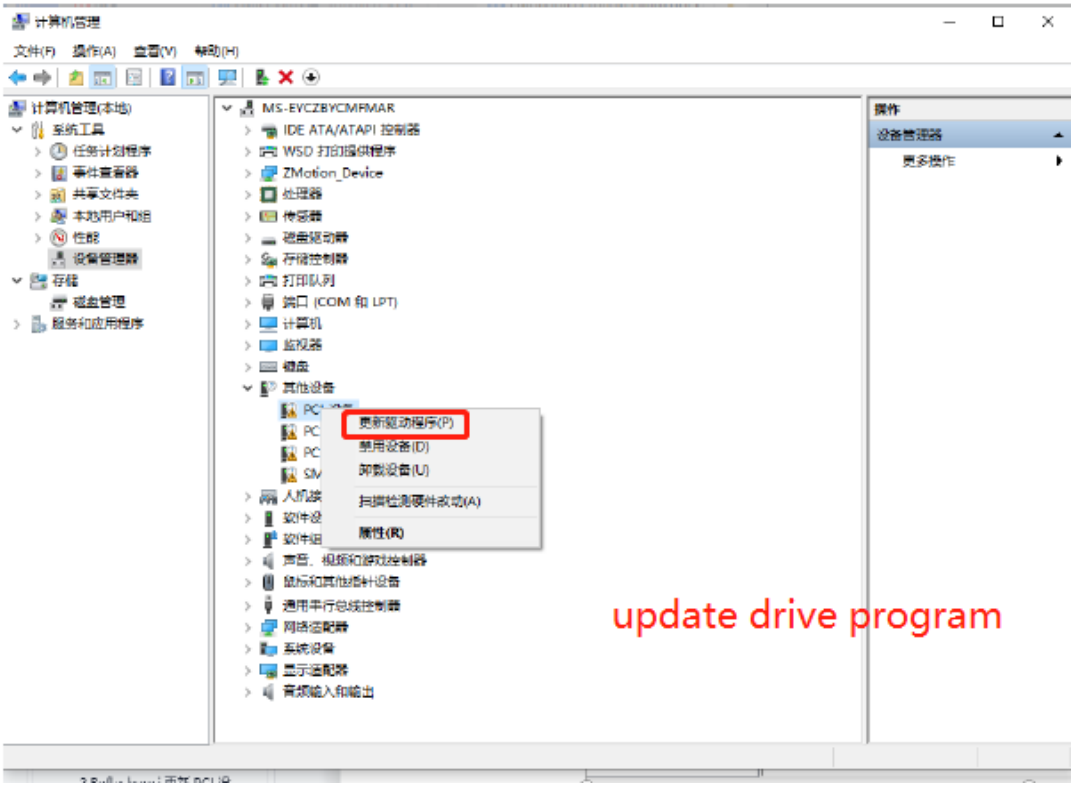
1. Open the Device Manager menu and select the PCI device in Other Devices.



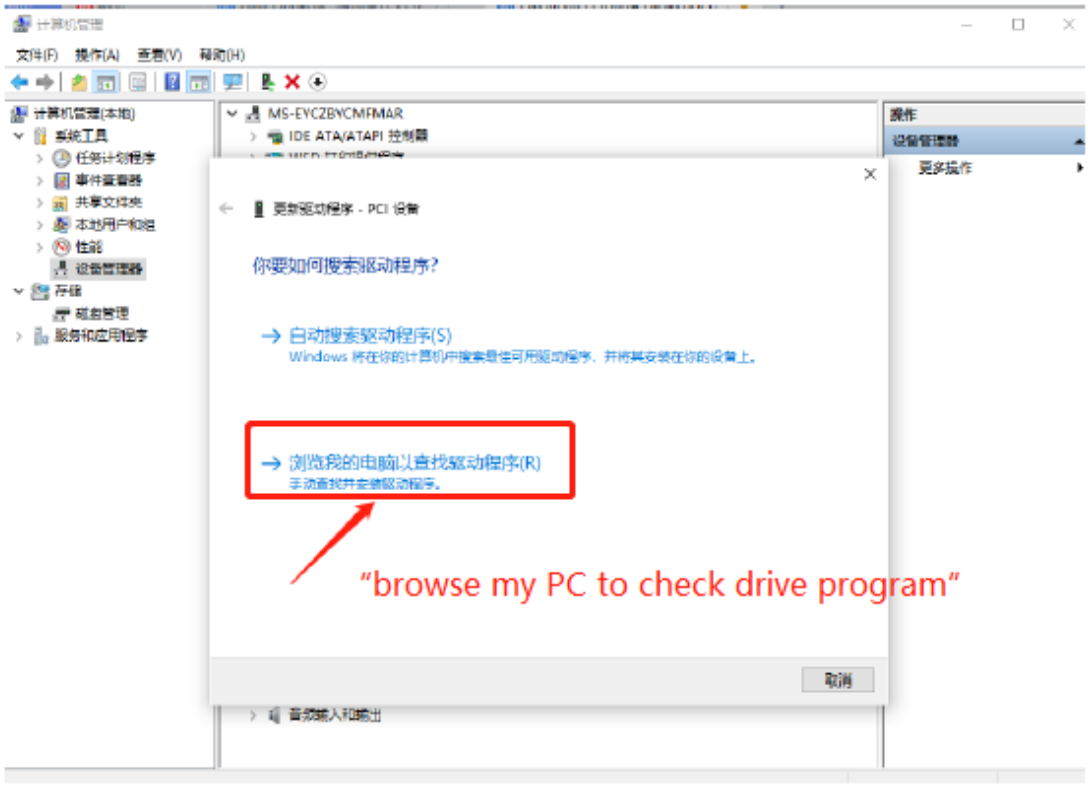
2. If there are multiple PCI devices, right-click "Properties" to view detailed information, select "Hardware ID" for properties, and confirm that it is a PCI device starting with PCI\VEN_EF34&DEV_1000&.



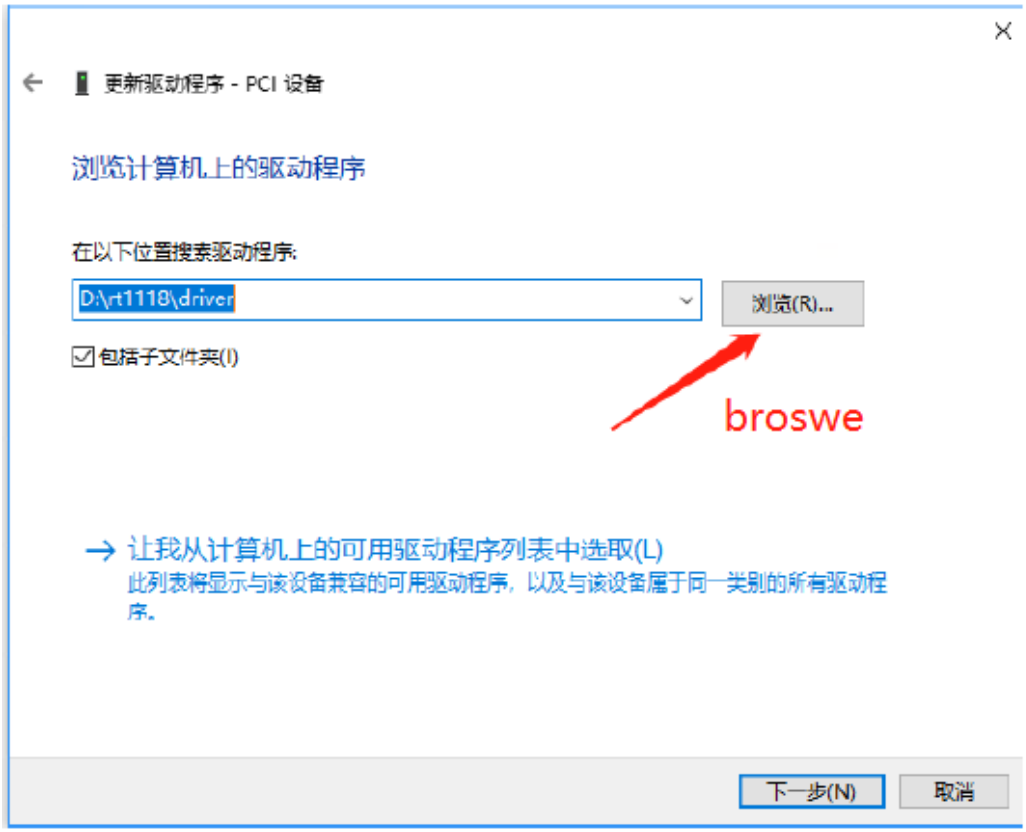
3. Find PCI Device, right-click to select “update drive program".



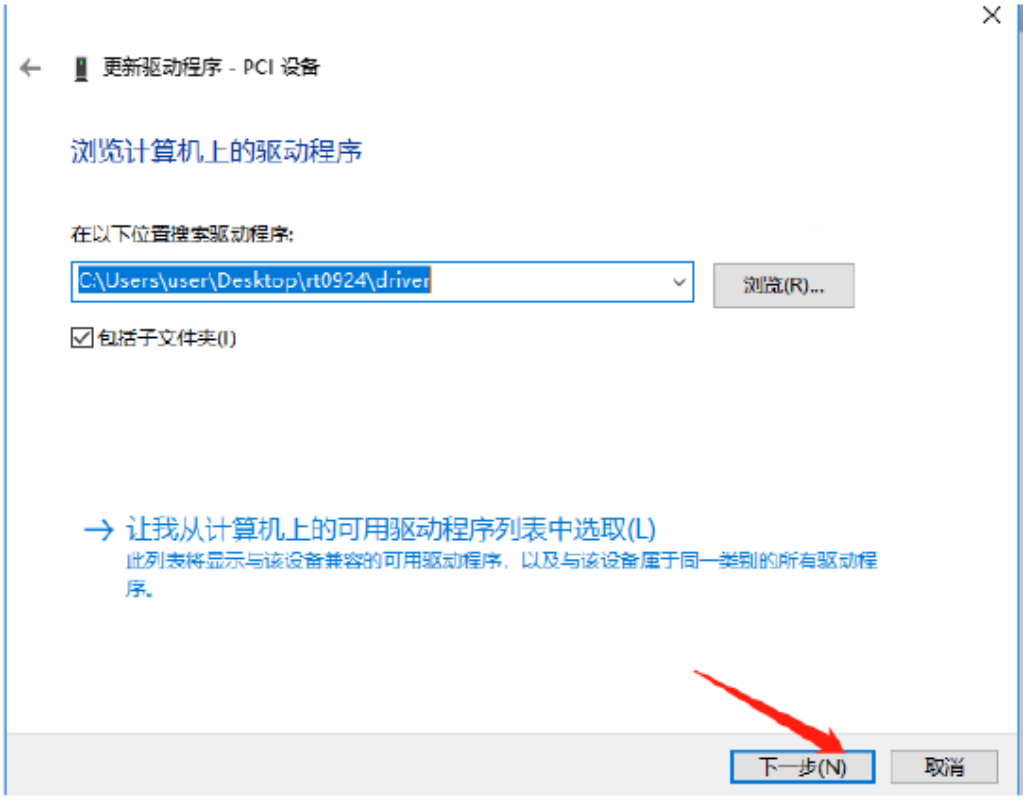
4. Select “browse my PC to check drive program”.



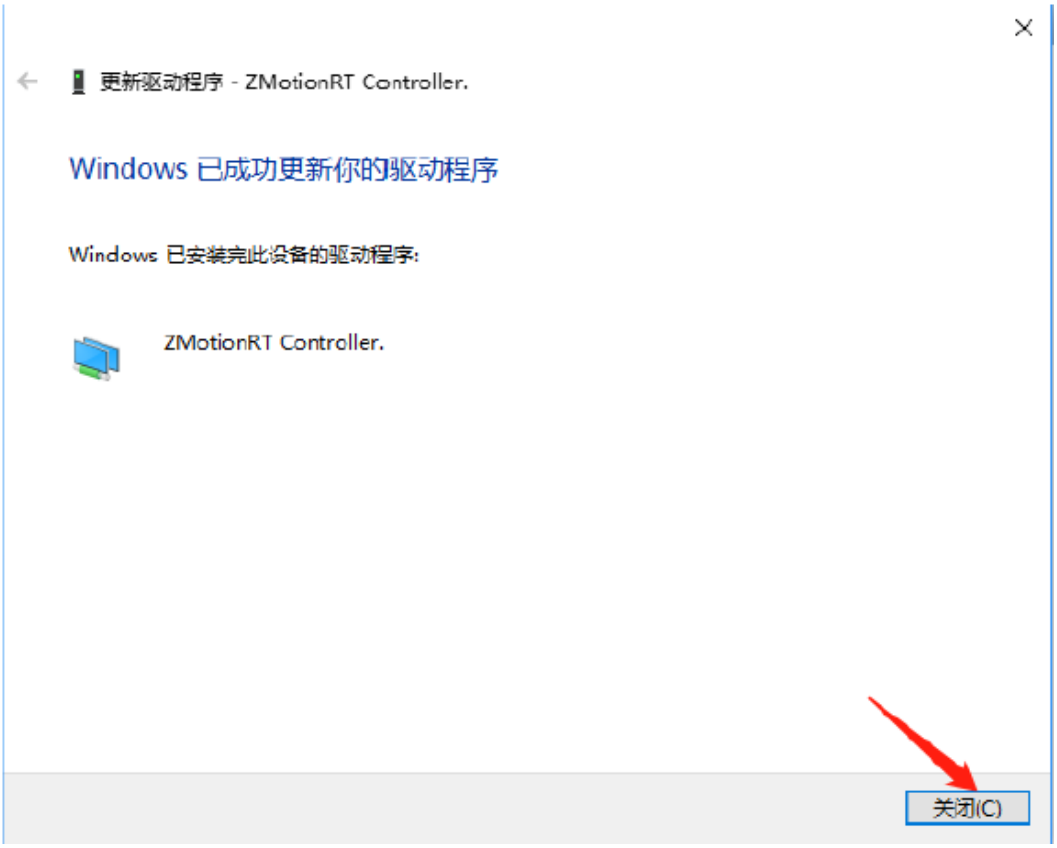
5. Click “browse", and select driver folder.



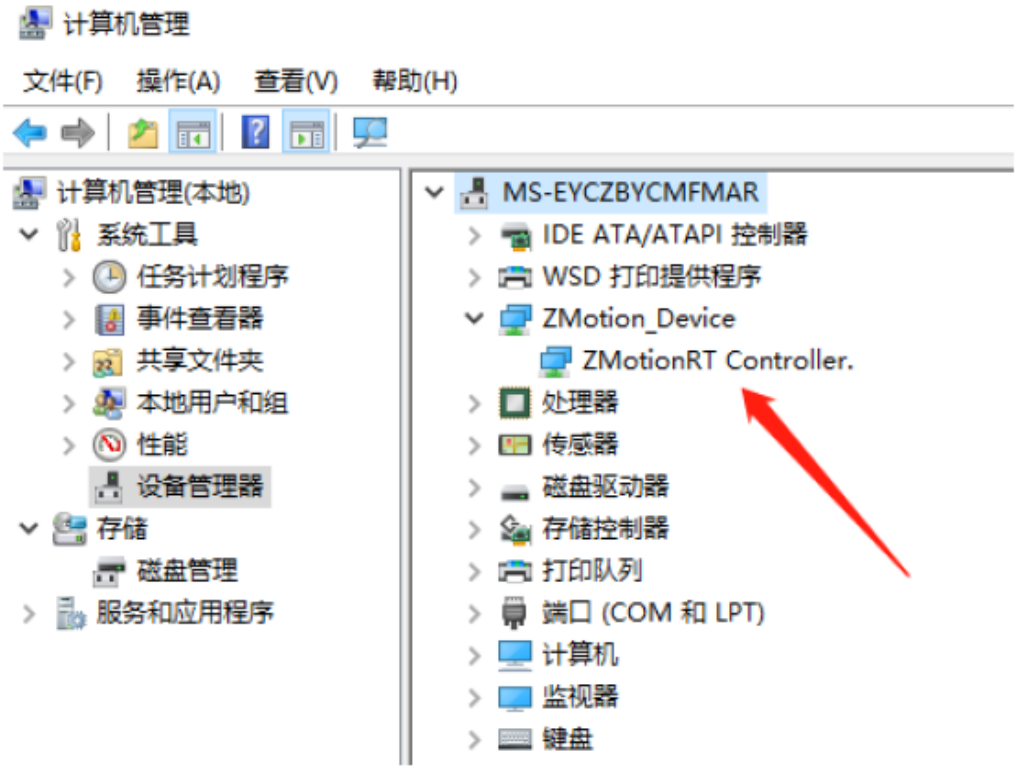
6. Click “next step".



7. Wait until installed, click close.



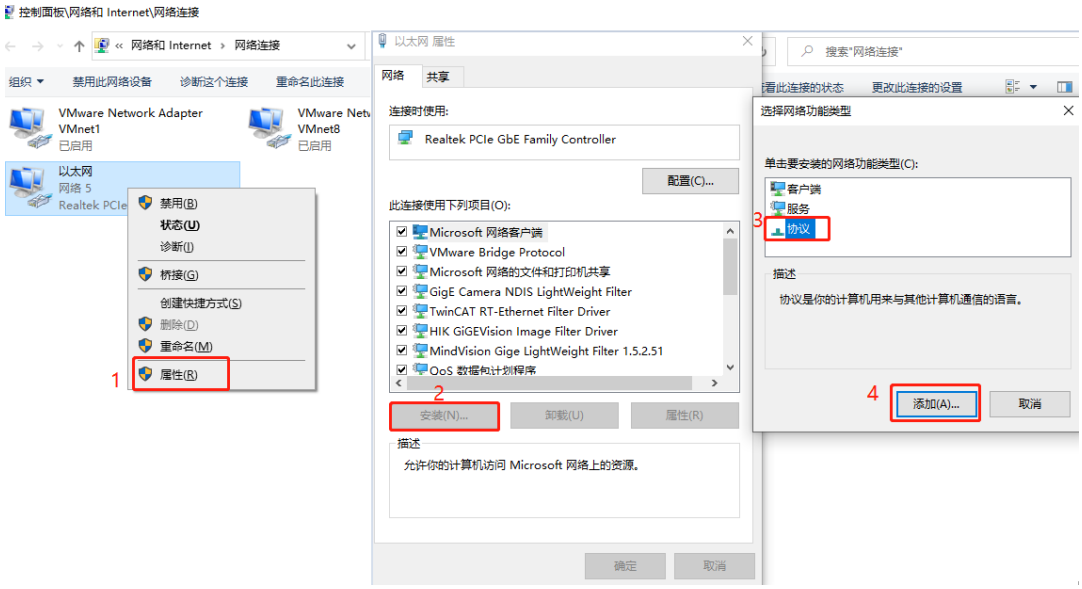
8. If there is ZMotionRTController in the device manager, the installation is successful.



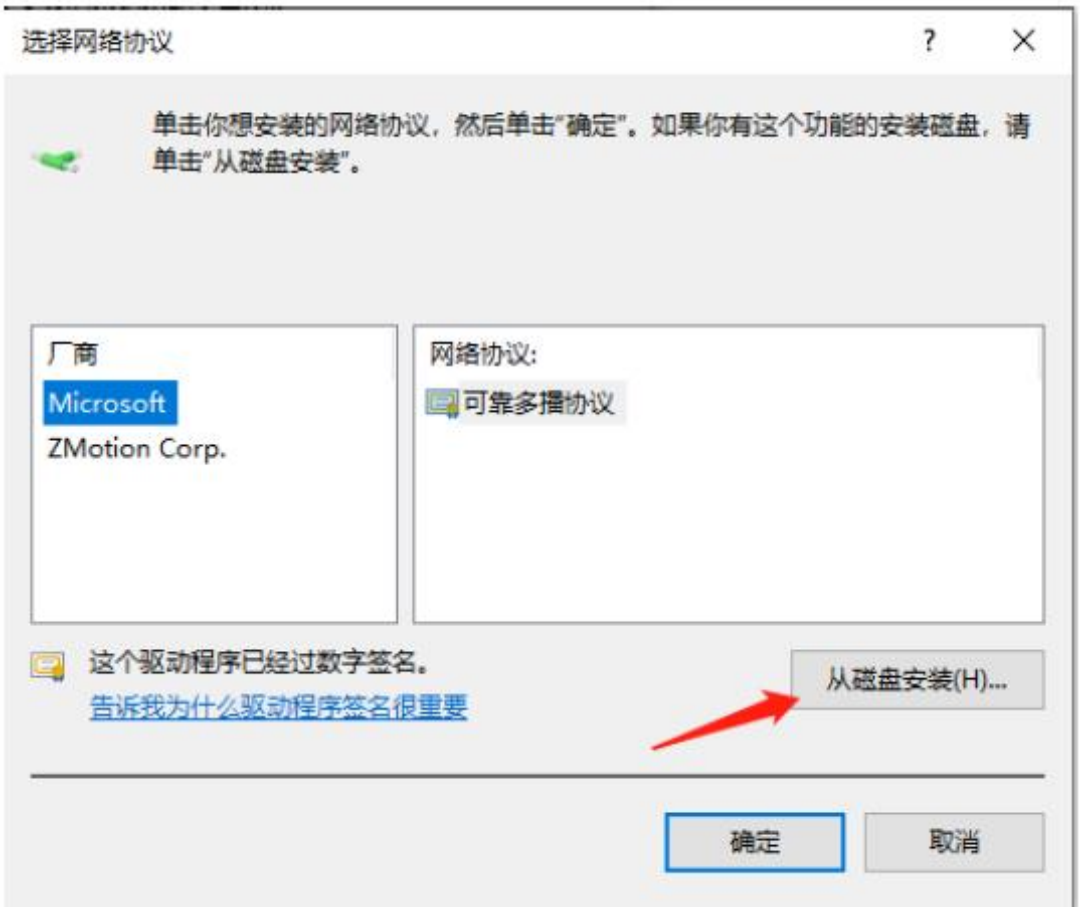
3.1.1.2.3. Ordinary Network Card Install EtherCAT Bus Protocol

MotionRT710 supports the ECAT network port of XPCI/XPCIE, and also supports the common network port of the computer as ECAT.

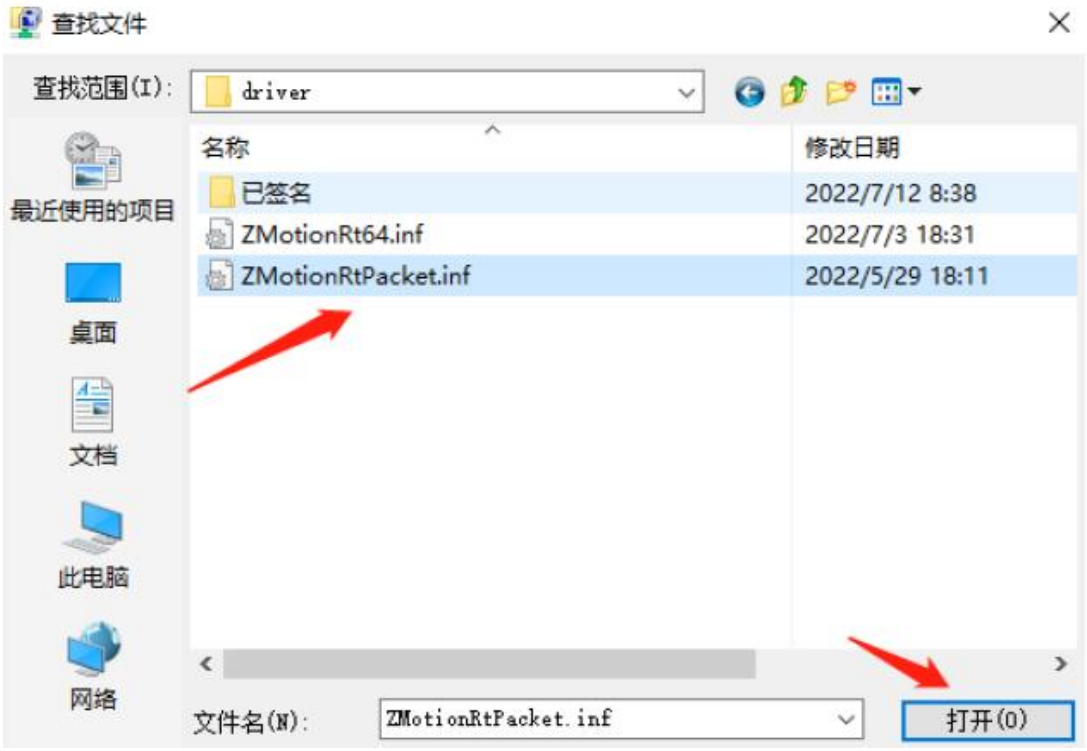
1. Open control panel, in Windows network connection interface, select the network port that needs to be used as the bus, right-click Properties->Installation->Protocol->Add.



2. Select “installation from disk".

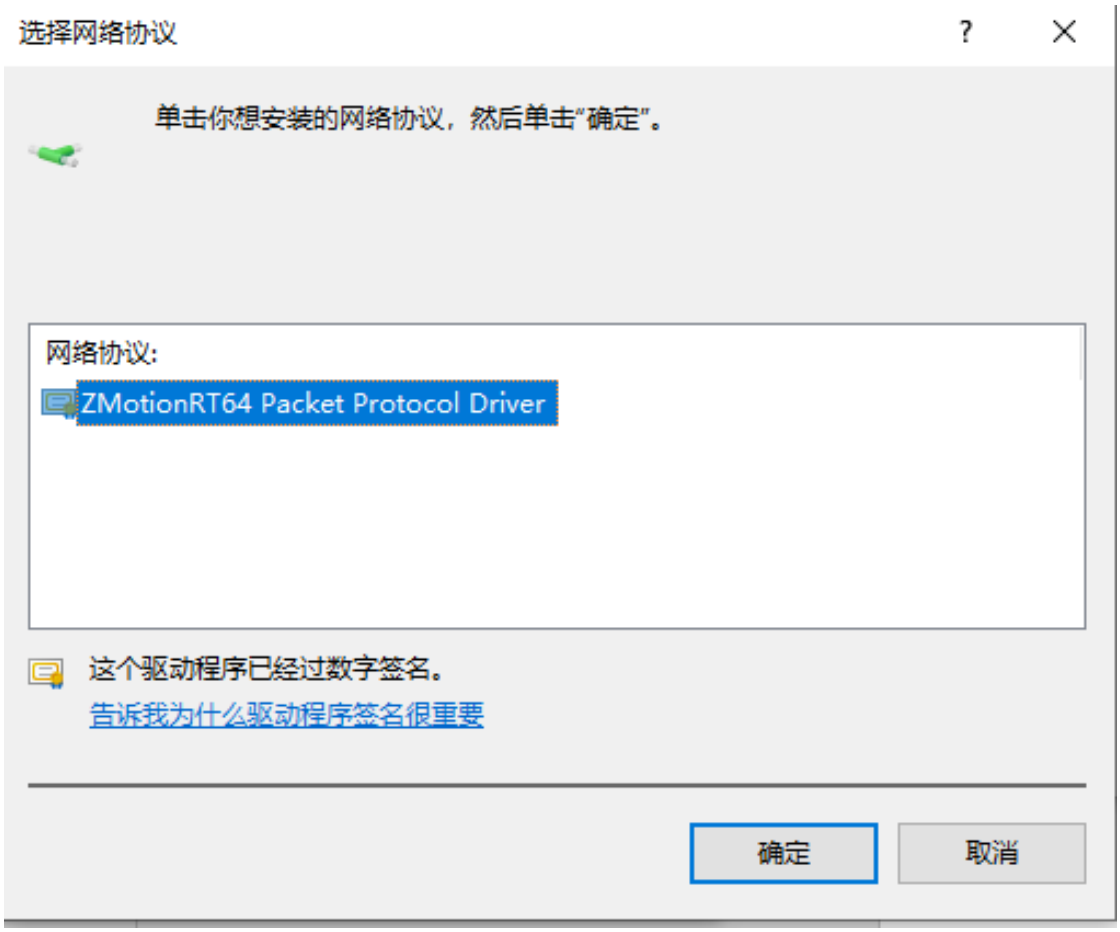


3. Brower drive position, select “ZMotionRtPacket.inf".



4. Click “ok".

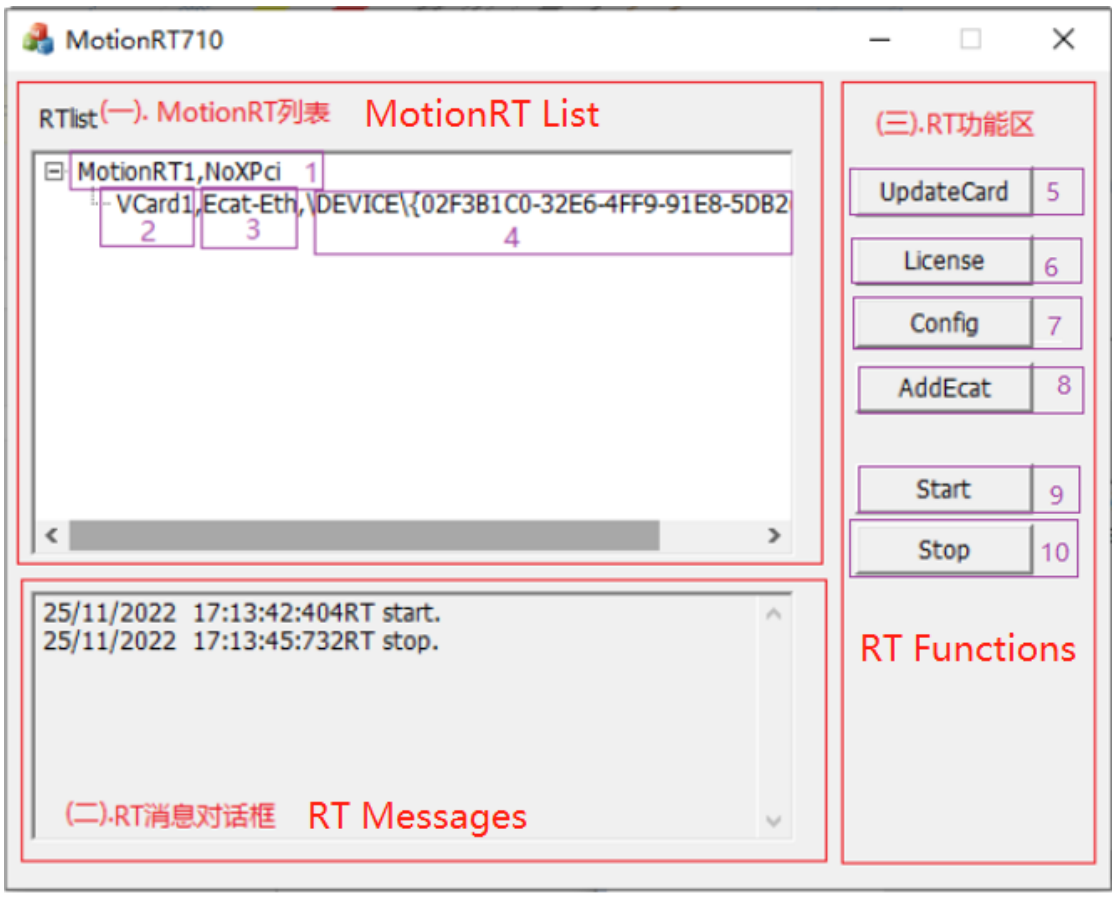
5. Select “ZMotionRT64PacketProtocolDriver" in network protocol, then click “ok".



6. After completion, there will be an additional "ZMotionRT64 Packet Protocol Driver" in the Ethernet attribute protocol, which means the installation is successful.

3.1.1.3. MotionRT Control Panel

- Mian Interface



(1) MotionRT List:

That is, the list of PCI cards currently present on the computer

[1]: PCI card model, NoXPci means no card

[2]: Type of card.

[3]: ID of the card. Ecat-Eth refers to the network port used when there is no card.

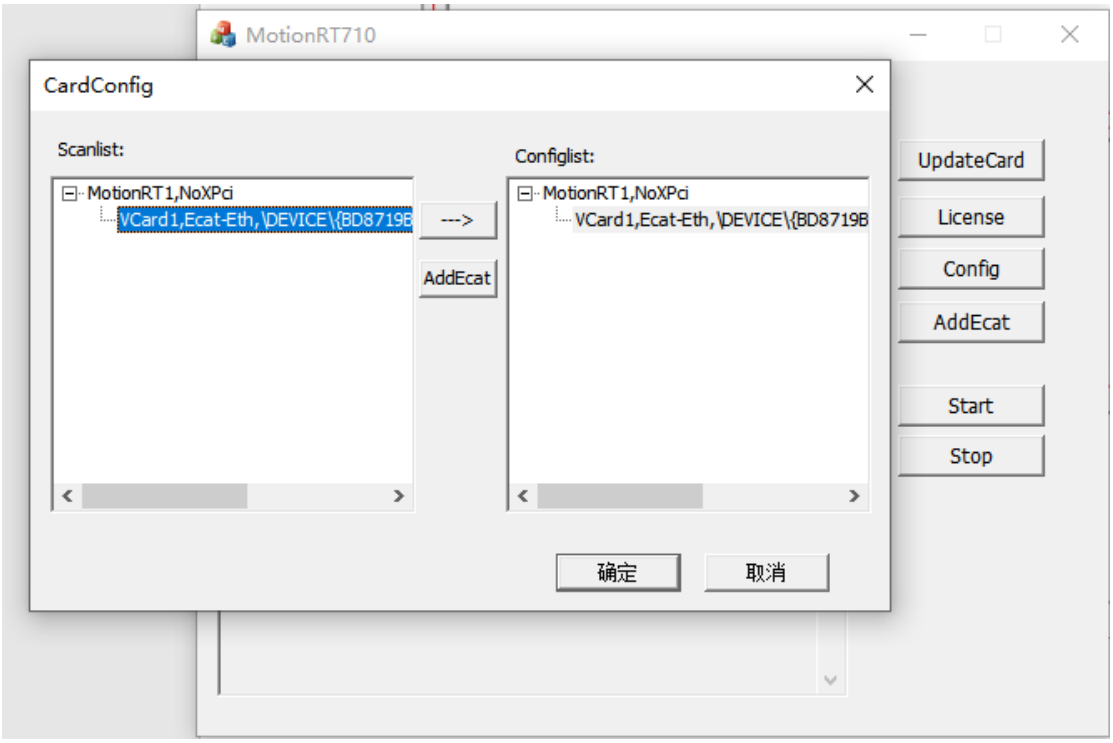
[4]: The dialing address of the card. When there is no card, it means the unique number of the network card

(2) RT Messages (dialogue area):

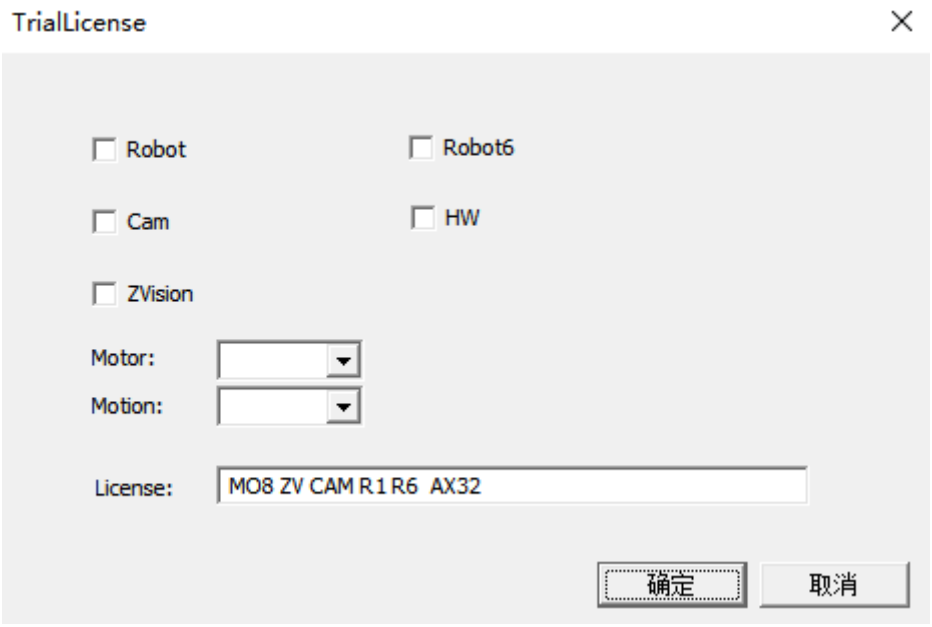
A message dialog box for interaction between the console and each card.

(3) RT Functions Area:

[5]: UpdateCard: refresh the configuration information of the card, and save it, and automatically load the configuration information when the console is opened next time. When using it for the first time, you need to manually add the card number information (AddEcat), as shown in the figure below:



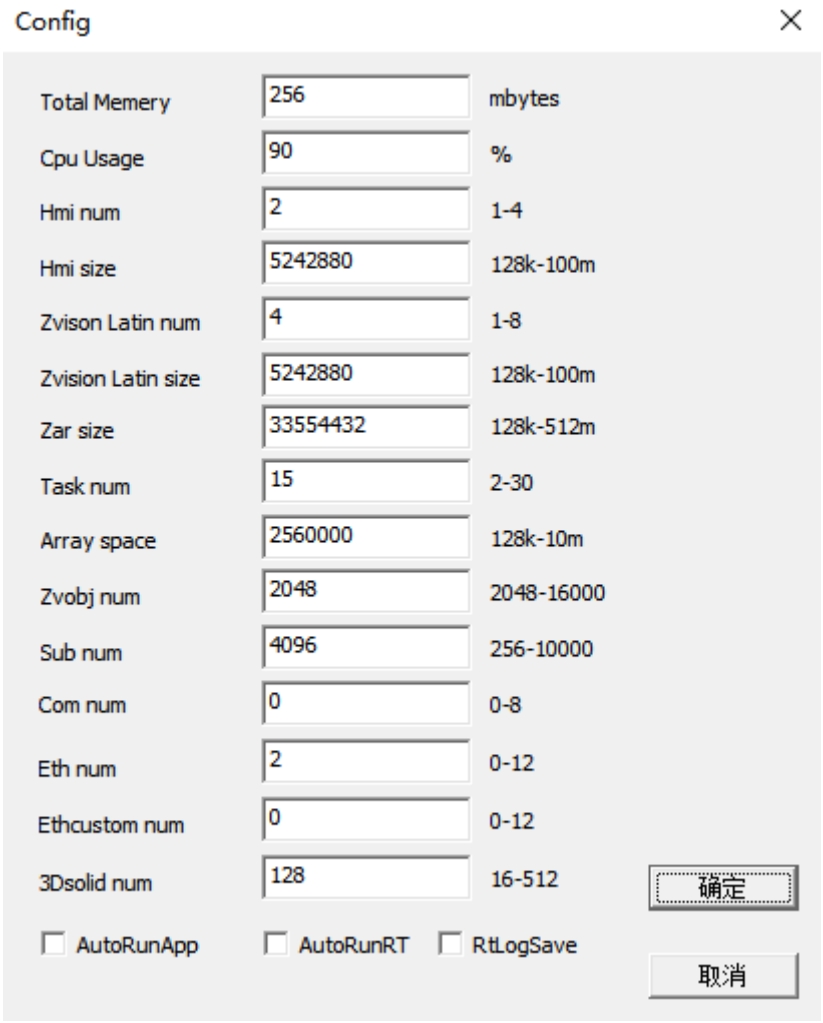
[6]: License function interface.



Function	Description
Robort	Ordinary robotic arm.
Robort6	6 axes and robotic arms are with below 6 axes.
Cam	Camera functions
HW	Hardware position comparison output function.
ZVision	Time instruction function.
Motor	The number of actual motors.

Motion	Valid motion control functions: MOO: point to point MO1: point to point, electronic cam. MO2: point to point, electronic cam, linear interpolation. MO6: point to point, electronic cam, linear interpolation, circular interpolation. MO8: point to point, electronic cam, linear interpolation, circular interpolation, continuous interpolation.
License	It is used to display the widow that saves License parameters configuration.

[7]: Config configuration interface.



Function	Description
Total Memery	Total memory, including the memory occupied by all the spaces in the controller that can save data, such as, array space, Zar file size, channel size, hmi resolution, etc., it is

	best to set value above 200.
Cpu Usage	CPU usage limit.
Hmi num	The number of Hmi that can be used.
Hmi size	The resolution of Hmi.
Zvision Latin num	Vision channels.
Zvision Latin size	Single vision channel size.
Zar size	Zar file size.
Task num	The number of tasks that can be executed.
Array Space	Assigned array space.
Zvobj num	The number of Zvobject.
Sub num	The number of max sub-function.
Com num	The number of serial ports.
Eth num	The number of network ports PORT, the set value should be smaller than the max value.
Ethcustom num	Customized Ethernet PORT numbers.
3Dsolid num	The number of 3D solids.
AutoRunApp	Open the software automatically when power on.
AutoRunRT	Automatically run RT software when power on.
RtLogSave	Save logs into log file.

[8]: AddEcat

Add connectable card.

[9]: Start

Start the MotionRT button, after starting, you can connect the control card.

[10]: Stop

Stop the MotionRT button, after stopping, the control card cannot be connected.

3.1.2.Routine

3.1.2.1. Connect to controller through Ethernet

It shows link controller by Ethernet, if it fails, error codes will be printed.

```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
if (ret)//if it is not 0, fail
{
printf("%s return code is %d\n", command, ret);
}
}

int _tmain(int argc, _TCHAR* argv[])
{
char *ip_addr = (char *)"127.0.0.1";          //Controller IP address is 192.168.0.11,
  simulator IP address is 127.0.0.1, it needs to open simulator, then use it to connect.
ZMC_HANDLE handle = NULL;                 //link handle
int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
if (ERR_SUCCESS != ret)
{
printf("Fail to connect controller!\n");
handle = NULL;
getchar();
return -1;
}
printf("Success to connect controller!\n");

//get motion types}
Sleep(2000);
ret = ZAux_Close(handle);    //close the connection
commandCheckHandler("ZAux_Close", ret) ;//judge whether the command is
executed successfully.
printf("connection closed!\n");
```

```
handle = NULL;
return 0;
}
```

3.1.2.2. Connect to controller through Serial Port

```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//if it is not 0, fail
	{
		printf("%s return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{

	ZMC_HANDLE handle = NULL;  //link the returned handle

	uint32 comid = 1; //connected COM port

	int ret = ZAux_OpenCom(comid,&handle); //COM1 connects to controller
	commandCheckHandler("ZAux_OpenCom", ret);
	if (ERR_SUCCESS != ret)
	{
		printf("serial connect failed!\n");
		handle = NULL;
	}
	else
	{
		printf("serial connect succeeded!\n");
	}

	uint32 dwBaudRate = 38400; //baud rate 38400
	uint32 dwByteSize = 8;    //8-bit data bit
	uint32 dwParity = 0;      //no parity +
```

```
    uint32 dwStopBits = 0; ///stop bit
    ret = ZAux_SetComDefaultBaud(dwBaudRate, dwByteSize, dwParity,
dwStopBits); //set serial port communication parameters
    commandCheckHandler("ZAux_SetComDefaultBaud", ret);
    //do something...

    Sleep(2000);
    ret = ZAux_Close(handle);   //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the command is
executed successfully.
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

3.1.2.3. Connect to Controller through PCI

If it fails, error codes will be printed, and return directly.

```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//if it is not 0, fail
    {
        printf("%s return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    ZMC_HANDLE handle = NULL;  //link returned handle
    uint32 cardnumcardnum = 0;   //PCI0

    int ret = ZAux_OpenPci(cardnumcardnum, &handle); ////connect to controller
through Ethernet
    commandCheckHandler("ZAux_OpenPci", ret);
    if (ERR_SUCCESS != ret)
```

```
    {
        printf("PCI connect failed!\n");
        Sleep(3000);
        handle = NULL;
        return 0;
    }
    else
    {
        printf("PCI connect succeeded!\n");
    }

    //do something.....
    Sleep(3000);

    ret = ZAux_Close(handle);
    commandCheckHandler("ZAux_OpenPci", ret);
    return 0;
}
```

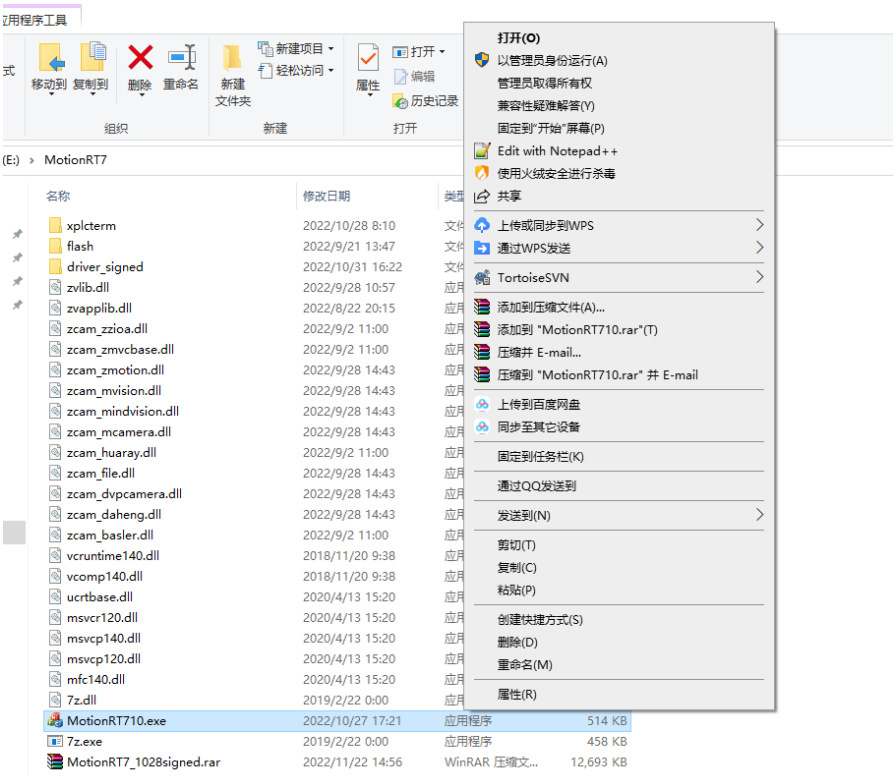
3.1.2.4. Modify IP Address

```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)// if it is not 0, fail
    {
        printf("%s return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";            //simulator IP address, it needs to
open the simulator.
    ZMC_HANDLE handle = NULL;            //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
```

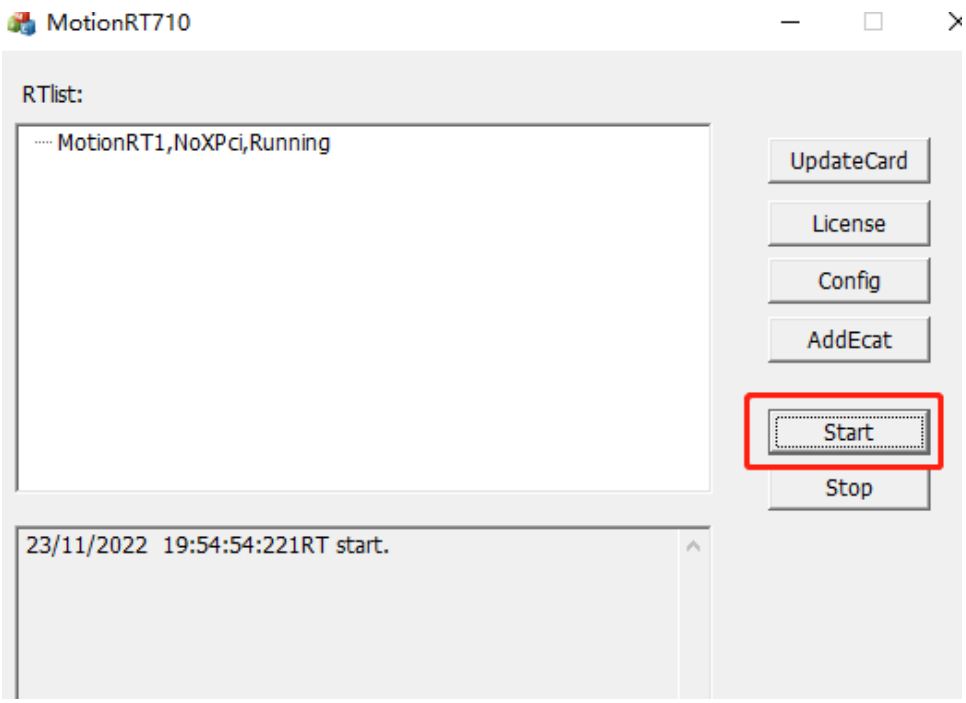
```
            printf("fail to connect controller!\n");
            handle = NULL;
            getchar();
            return -1;
        }
        printf("success to connect controller\n");

        ret = ZAux_SetIp( handle,(char *)"192.168.0.11");//simulator IP can't be modified, only controller IP can be modified, this routine is for introducing, so connect to the simulator.
        commandCheckHandler("ZAux_Close", ret) ;//judge whether the command is executed successfully

        //get the motion type}
        Sleep(2000);
        handle = NULL;
        return 0;
}
```

3.1.2.5. MotionRT Connection

Connect to MotionRT, first to open MotionRT control panel:



Open the console, configure the configuration, and click Start to start the connection.



Note: zmotion.dll version must be above 3.8.8.50.

```
// test1.cpp: define the entry point of control panel application program.
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//if it is not 0, fail.
	{
		printf("%s return code is %d\n", command, ret);
	}

}

int _tmain(int argc, _TCHAR* argv[])
{

	ZMC_HANDLE handle = NULL;		//link handle

	char MotionID[32]= "";			//no requirements for character strings, character strings can be filled.
	int ret=ZAux_FastOpen(5, MotionID,1000 ,&handle);	//link to MotionRT

	if (ERR_SUCCESS != ret)
	{
		printf("fail to connect controller!\n");
		handle = NULL;
		getchar();
		return -1;
	}
```

```
    printf("success to connect controller!\n");



    //get the motion type}
    Sleep(2000);
    handle = NULL;
    return 0;
}
```

3.2. Get Controller Information

3.2.1. Emphasis

In order to facilitate the user to check the relevant information of the motion controller, two functions, ZAux_GetControllerInfo and ZAux_GetSysSpecification, are provided to obtain the relevant information of the controller.

Controller model type, controller software version (firmware version), unique ID of the controller, maximum number of virtual axes of the controller, maximum number of motors of the controller, maximum number of IN, OUT, AD, and DA of the controller, and system date of the controller, controller system time and other parameters can be obtained. If you want to get more controller parameters, you can send “ZAux_Execute" command, and use the string "?*max" or "?*set" to get more controller parameters.

3.2.2. Routines

3.2.2.1. Controller Messages

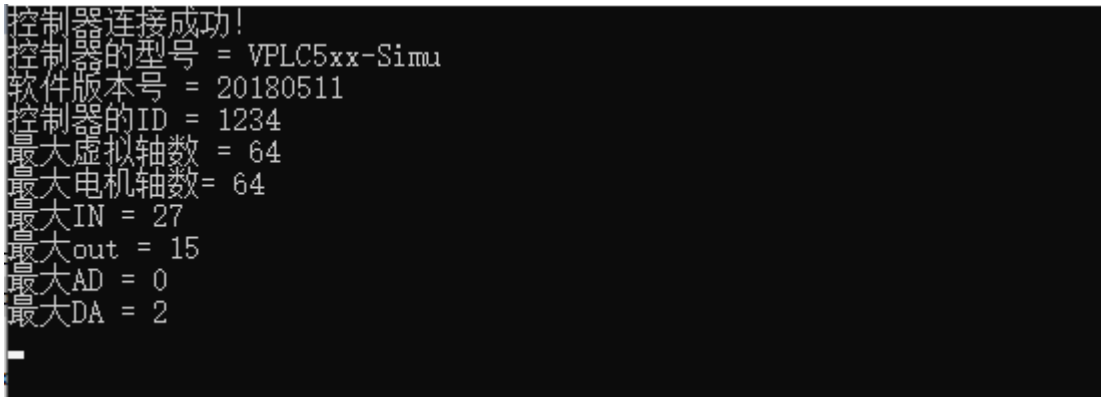
This routine mainly demonstrates how to obtain the relevant information of the controller through instructions.

```
// test1.cpp: define the entry point of control panel application program.
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
```

```
{
	if (ret)//if it is not 0, fail
	{
		printf("%s return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
 char *ip_addr = (char *)"127.0.0.1";		//controller factory default IP address is
 192.168.0.11, simulator IP address is 127.0.0.1, it needs to open the simulator for
 connection through simulator.
	ZMC_HANDLE handle = NULL;		//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("fail to connect controller! \n");
		handle = NULL;
		getchar();
		return -1;
	}
	printf("success to connect controller! \n");

	char SoftTypes[20];
	char SoftVersions[20];
	char ControllerIDs[20];

	ret = ZAux_GetControllerInfo(handle,SoftTypes,SoftVersions,ControllerIDs);//get
control card information
	commandCheckHandler("ZAux_GetControllerInfo", ret) ;// judge whether the
command is executed successfully.
	printf("SoftType = %s\n", SoftTypes);	//print controller model
	printf("SoftVersion = %s\n", SoftVersions);	//print version No.
	printf("ControllerId = %s\n", ControllerIDs); //print controller ID
	uint16 Max_VirtuAxises1;
	uint8 Max_motor1;
	uint8 Max_io1[4];
	ret =
ZAux_GetSysSpecification(handle,&Max_VirtuAxises1,&Max_motor1,Max_io1);//get
the max specification of control card.
	commandCheckHandler("ZAux_GetSysSpecification", ret) ;//judge whether the
command is executed successfully.
	printf("Max_VirtuAxises1 = %u\n", Max_VirtuAxises1); //max virtual axes
	printf("Max_motor1 = %u\n", Max_motor1);	//max motor axes
	printf("Max_io1[0] = %u\n", Max_io1[0]); //max IN
	printf("Max_io1[1] = %u\n", Max_io1[1]); //max out
	printf("Max_io1[2] = %u\n", Max_io1[2]); //max AD
	printf("Max_io1[3] = %u\n", Max_io1[3]); //max DA

	Sleep(2000);
	ret = ZAux_Close( handle);//close the connection
	commandCheckHandler("ZAux_Close", ret) ;//judge whether the command is
executed successfully.
	handle = NULL;
	return 0;
```

```
}
```

Running result:



Chapter IV Set Motion Control Parameters

In the motion control system, the object of motion control is called "axis", and a motor-controlled motion platform in the motion control system is called a motion axis. Each motion axis has only one degree of freedom and can do linear motion or rotary motion.

The classification of Zmotion motion control card axes is as follows:

Axis Type	Description
Motor Axis	Use the pulse axis interface, EtherCAT bus or RTEX bus interface of the controller to connect with the driver, assign the axis No. to the device, and use one motor as one axis.
Virtual Axis	The virtual axis is within the motion controller, and a real drive is not used. Or it is used as a virtual master axis for synchronous control and as a Cartesian axis in the robot algorithm.
Encoder Axis	Use the controller's local encoder axis interface, and assign it as the actual encoder input usage.

Motor axis: it actively operates, the motor moves according to the pulses sent by the controller. The number of pulses sent is determined by (motion parameter variation)*(Units set by ZAux_Direct_SetUnits), and the target demand position is reflected by the planning position parameter of ZAux_Direct_GetDpos.

Encoder axis: it passively operates, the encoder follows the rotation of the motor, generates pulses, and feeds back to the controller. The number of pulses received by the controller is determined by calling ZAux_Direct_GetEncoder, and the encoder feedback position is reflected by the feedback position parameter obtained by ZAux_Direct_GetMpos.

After the user connects a whole system (including motion controller, drive, motor), how to check the status of the whole system? How to configure this system? How to check the alarm information? How to configure speed, acceleration and deceleration? This chapter will introduce which states the user can detect and how to detect, and how to configure motion control parameters.

4.1.Basic Parameters of Motion

4.1.1.Emphasis

4.1.1.1. Axis States

Get the various statuses of the axis through the ZAux_Direct_GetAxisStatus command. The value is displayed in decimal, and the status is judged by the corresponding bit in binary. Multiple errors can occur at the same time.

The axis status word is defined as follows:

Bit	Description	Print Values	
1	Alarm: Follow-sup errors exceed	2	2h
2	Error of communication with remote axis	4	4h
3	Remote driver alarms	8	8h
4	Forward hard position limit	16	10h
5	Reverse hard position limit	32	20h
6	While finding the origin	64	40h
7	HOLD speed keeps signal input	128	80h
8	Error: Follow-up error out of limit	256	100h
9	Exceed forward soft position limit	512	200h
10	Exceed reverse soft position limit	1024	400h
11	CANCEL is in execution	2048	800h
12	Pulse frequency exceeds MAX_SPEED, it needs to modify the speed (slow down) or MAX_SPEED	4096	1000h
14	Robot instruction coordinates error	16384	4000h
18	Abnormal power	262144	40000h
21	Fail to trigger special motion instruction in motion	2097152	200000h
22	Alarm signal inputs	4194304	400000h
23	Axis has entered in pause state	8388608	800000h

Call **ZAux_Direct_GetAxisStopReason** to view the axis history stop reasons, and write 0 to clear, bit by bit latch, and the latch is the axis status information.

Call **ZAux_Direct_GetIfIdle** to judge whether the motion command added to the axis is completed, return 0 during the motion, and return -1 when the motion ends.

4.1.1.2. Axis Type

The axis type calls the ZAux_Direct_SetAtype/ZAux_Direct_GetAtype (ATYPE) command to configure according to the characteristics of the current axis. When the user program is initialized, the configuration of the axis type should be completed as soon as possible. If the type does not match, an error will be reported.

All unassigned axes default to virtual axes with an ATYPE of 0.

The axis types supported by the controller are as follows:

Atype	Description
0	Virtual axis
1	Servo or stepper of pulse direction
2	Servo controlled by analog signal (reserved)
3	Quadrature encoder
4	Pulse directional output + quadrature encoder input
5	Pulse directional output + pulse directional encoder input
6	Encoder of pulse directional method
7	Servo or stepper of pulse direction + EZ signal input
8	Expand servo or stepper of pulse direction through ZCAN
9	Expand quadrature encoder through ZCAN
10	Expand encoder of pulse direction through ZCAN
20	Galvanometer type, with galvanometer status feedback. Bit 2 of AXISSTATUS will be set if the galvanometer cannot be connected, and ENCODER will return to the original sending position, the unit is pulse. ZMC408SCAN supports.
21	Galvanometer axis type, need controller support. The default system period is 250us, and the galvanometer refresh period is 50us, which is related to the firmware. All motion control instructions of common axes can be used, and mixed interpolation of galvanometer axes and other axis types is supported
22	Galvanometer type, with galvanometer status feedback. Bit2 of AXISSTATUS will be set if the galvanometer cannot be connected, and bit3 of AXISSTATUS will be set when the galvanometer alarms. MPOS will return to the feedback position (MPOS), and the anti-correction process will be performed. ENCODER will return to the original feedback position. ZMC408SCAN supports.
24	Remote encoder axis type. The ZHD500X uses the handwheel, and the firmware version of the controller 5 series 20180404 or above is required to support it.
50	RTEX cycle position mode, need RTEX controller.
51	RTEX cycle speed mode, need RTEX controller.

52	RTEX cycle torque mode, need RTEX controller. Please close driver 2 degrees of freedom control mode firstly, and set speed limit.
65	EtherCAT cycle position mode, need EtherCAT controller.
66	EtherCAT cycle speed mode, need EtherCAT controller. DRIVE_PRODILE needs to be set as 20 or above.
67	EtherCAT cycle torque mode, need EtherCAT controller. DRIVE_PRODILE needs to be set as 30 or above.
70	ECAT self-defined operation, it only reads data of encoder, need EtherCAT controller.

1. Axis type (ZAux_Direct_SetAtype)=0: virtual axis

When multi-axis synchronous motion can be used as the main axis of the machine, the slave axes all follow this virtual main axis.

As the superposition axis of other axes, a virtual axis is superimposed on the actual moving axis. These virtual axes can be set through the ADDAX command (axis superposition), and then the motion of each virtual axis is superimposed on the real axis.

2. Axis type (ZAux_Direct_SetAtype)=1 or 7: pulse axis

The movement of the axis is controlled by the controller sending pulses, the direction of the pulses determines the steering of the motor, and the speed of the axis movement is controlled according to the frequency of the pulses sent.

3. Axis type (ZAux_Direct_SetAtype)=3 or 6: encoder axis

When the encoder occupies one axis No. independently, the axis type can be 3 or 6 according to the encoder type.

4. Axis type (ZAux_Direct_SetAtype) = 4: pulse axis and encoder axis share the axis No.

When the current pulse axis has encoder feedback, the axis type is set to 4, and the pulse output and encoder input signals are on the same axis.

5. Axis type (ZAux_Direct_SetAtype)=8: CAN extended axis

When the CAN bus extension axis is used, the axis type of the extended pulse axis is set to 8, and the axis type connected to the encoder axis on the extended axis is set to 9.

6. Axis type (ZAux_Direct_SetAtype)=21: galvanometer axis No.

When connecting to the laser galvanometer device, it needs to set the galvanometer

axis type to 21, which is supported by some models of the laser galvanometer axis.

7. Axis type (ZAux_Direct_SetAtype)=50,51,52 : RTEX bus axis

When using the RTEX bus driver, the axis type can only be selected from the above three, and the axis type (ZAux_Direct_SetAtype) = 50 is the position mode, using motion commands to control the motor operation. The axis type (ZAux_Direct_SetAtype) = 51 means speed mode, under the speed mode, use DAC command to set the running speed of the motor and keep running. The axis type (ZAux_Direct_SetAtype)=52 means torque mode, use the DAC command to set the motor torque in torque mode and keep running.

To switch between speed and torque modes, to prevent accidents, firstly set DAC to 0 and then use the ZAux_Direct_SetAtype command to switch.

Note: before modifying the axis type (ZAux_Direct_SetAtype) and switching to the torque mode, please set the first position of the drive parameter Pr6.47 to 0 and close the 2-DOF control mode. Then set the speed limit by parameter Pr3.17. When the setting value of Pr3.17 (speed limit selection) is 0, the speed limit is set by Pr3.21, and when the setting value is 1, the speed limit value can be Pr3.21 or Pr3.22 when switching to torque control through SL_SW.

8. Axis type (ZAux_Direct_SetAtype)=65,66,67: EtherCAT bus axis No.

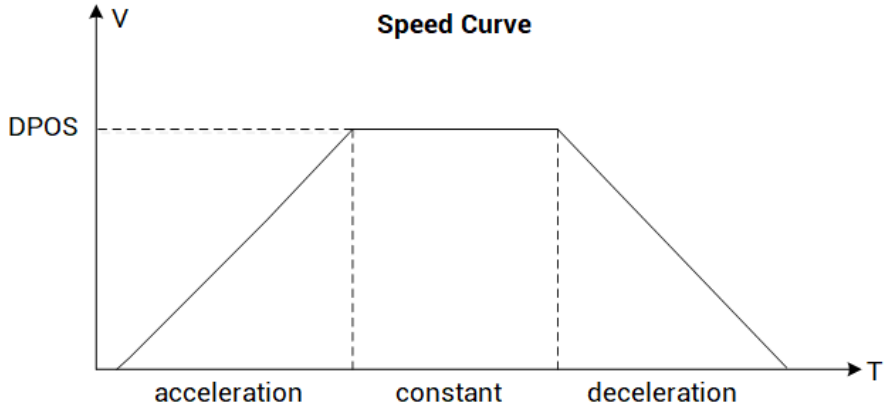
When using the EtherCAT bus driver, the axis type can only be selected from the above three, where the axis type (ZAux_Direct_SetAtype) = 65 is the position mode, using motion commands to control the motor operation. The axis type (ZAux_Direct_SetAtype) = 66 means speed mode, under the speed mode, use DAC command to set the running speed of the motor and keep running. And here are two speed units, pulse number/S and R/MIN, which are determined by the driver. The axis type (ZAux_Direct_SetAtype)=67 means torque mode, use the DAC command to set the motor torque in this torque mode, and run continuously, the DAC value range is 0-1000 in torque control mode, corresponding to 0-100%.

To switch between speed and torque modes, to prevent accidents, firstly set DAC to 0 and then use the ZAux_Direct_SetAtype command to switch.

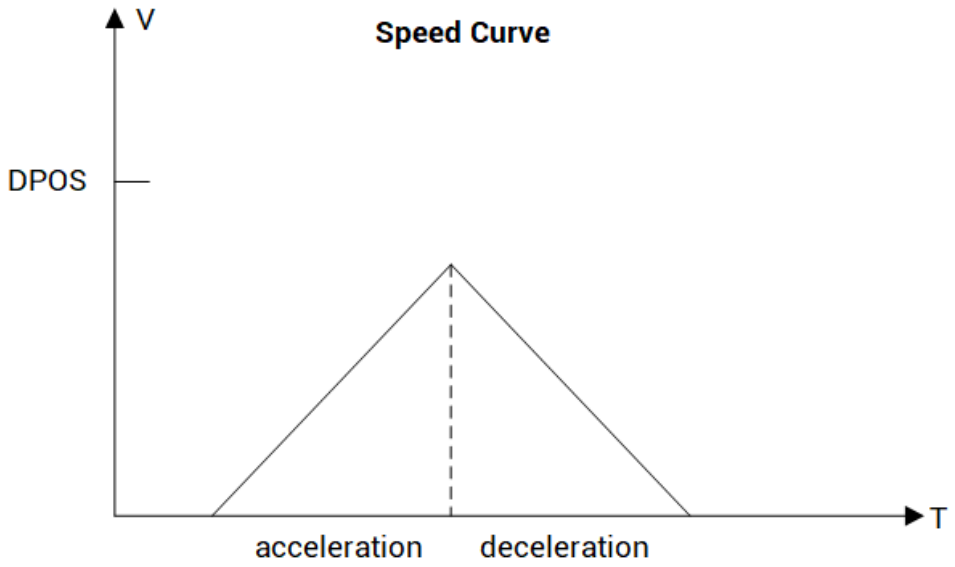
4.1.1.3. Axis Speed

The speed curve is generally divided into three stages: acceleration stage, constant

speed stage, and deceleration stage, as shown in the figure below.



When the displacement is short, there may not be a constant velocity stage, only an acceleration and deceleration stage, as shown in the figure below.



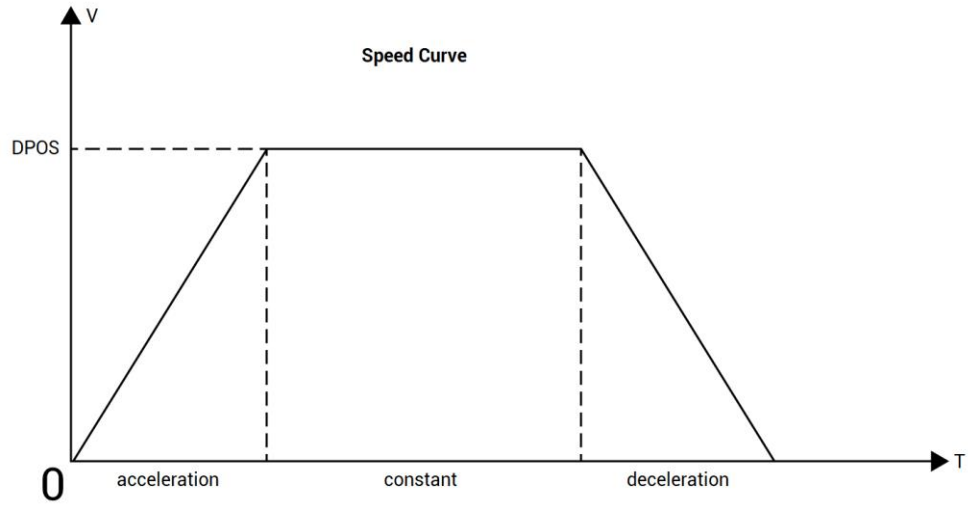
Commonly used speed commands include ZAux_Direct_SetSpeed (motion speed configuration), ZAux_Direct_SetAccel (acceleration onfiguration), ZAux_Direct_SetDecel (deceleration configuration) and ZAux_Direct_SetFastDec (fast deceleration and deceleration configuration, it can be used to stop rapidly when there is emergency stop, alarm or position limit). And the setting is completed when the axis parameters are initialized , as the velocity reference for motion commands.

Ladder diagram curve:

If SRAMP is not set (set SRAMP as 0. Call ZAux_Direct_SetSramp to set), the speed curve is a trapezoidal curve. In this speed planning mode, the speed curve changes according to a trapezoidal curve. Keep the parameter values such as speed, acceleration

and deceleration unchanged.

After the speed reaches the set value, it moves at a constant speed. If only the acceleration is set, and when the deceleration is 0, the deceleration will automatically be equal to the acceleration value. Generally, the corresponding acceleration and deceleration is set before the motion, and do not modify it during the motion. Otherwise, it will cause changes in the trajectory of the movement. As shown in the figure: S curve time (ZAux_Direct_SetSramp)=0 (the acceleration and deceleration process is faster, and the speed change has a greater impact on the machine tool.)



For example: suppose the target speed value (DPOS) is 100, the acceleration is 1000, and the deceleration is 1000, then:

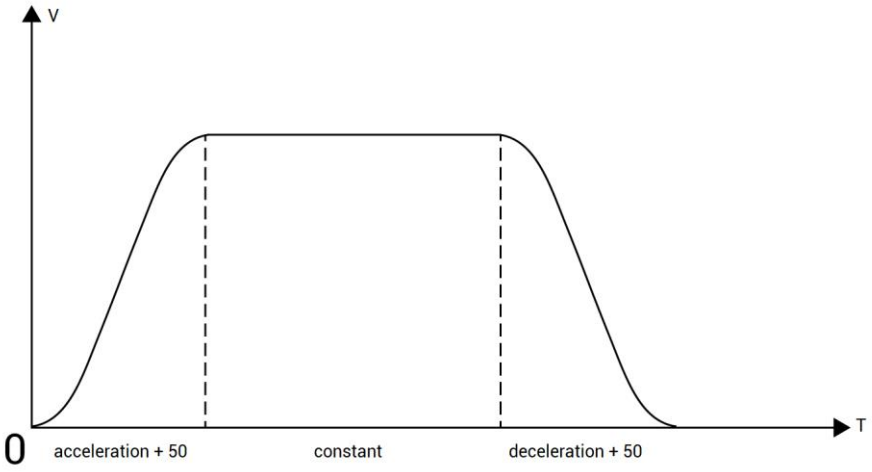
Total acceleration time=100/1000=0.1s

Total deceleration time=100/1000=0.1s

S-curve:

By calling ZAux_Direct_SetSramp to set the appropriate acceleration and deceleration rate of change, so that the speed curve is smooth, and the vibration is reduced when the machine starts and stops or accelerates and decelerates. The value of S-curve time (ZAux_Direct_SetSramp) ranges from 0 to trapezoidal acceleration time (in ms). After setting, the acceleration and deceleration process will take longer for the corresponding time. The longer the time, the smoother the speed curve. If the setting time exceeds the trapezoidal acceleration time, proceed according to the trapezoidal acceleration time.

When the S-curve time (ZAux_Direct_SetSramp)=50, the S-curve is obtained as follows: the acceleration and deceleration are softer.



For example: suppose the target speed value (DPOS) is 100, the acceleration is 1000, the deceleration is 1000, and the S-curve time is 50ms, then:

Total acceleration time=100/1000 + 0.05=0.1s+0.05s=0.15s

Total deceleration time=100/1000 + =0.05=0.1s+0.05=0.15s

4.1.1.4. UNITS (Pulse Amount)

In the motion control device, when the mechanical structure is determined, the transmission relationship between the motor and the mechanical device is fixed, and the mechanical displacement generated by the motor per revolution is also fixed. The UNITS pulse equivalent (ZAux_Direct_SetUnits) is the corresponding number of pulses per unit, which can be unit distance, unit angle, etc., and supports 5 decimal places.

The controller uses the pulse equivalent (ZAux_Direct_SetUnits) as the basic unit, and the target position, speed, acceleration and deceleration of the movement are all performed with the pulse equivalent (ZAux_Direct_SetUnits) as the basic unit. After the pulse equivalent (ZAux_Direct_SetUnits) is modified, the target position, Speed, acceleration and deceleration, etc. will change proportionally with the pulse equivalent (ZAux_Direct_SetUnits).

Assuming a pulse-type motion controller, the controller sends 1000 pulses, and the equipment mechanism moves 1mm, then when the pulse equivalent units is set to 1000, the move-related instructions in motion move 1mm, and 1000 pulses are actually sent.

The speed setting is 100 user unit, that is 100mm/s, when controls axis to move 50 user unit, actually it moves 50mm.

Syntax: ZAux_Direct_SetUnits (controller connection handle, axis No., variable)

Servo pulse equivalent setting:

Reduction ratio: that is, the transmission ratio of the deceleration device, which is a kind of transmission ratio, and refers to the ratio of the instantaneous input speed to the output speed in the deceleration mechanism

Thread lead: it is the axial distance that any point on the thread moves along the same helix for one revolution. The pitch refers to the axial distance between the corresponding points of two adjacent teeth on the thread, and the code is P. The lead of a single-thread thread is equal to the pitch, and the lead of a multi-thread thread is equal to the number of heads (n) multiplied by the pitch, that is, the lead S=nP

If the mechanism has a lead (in mm) and a reducer, the pulse equivalent should be converted into mm as:

Pulse equivalent = electronic gear ratio * lead / encoder resolution * mechanical reduction ratio

Note:

1. The motor rotates a circle, and the reducer output rotates b circles, then the reduction ratio is: a:b

2. If the motor is directly connected to the screw, the mechanical reduction ratio is 1:1

3. If the axis is a rotary shaft, the lead is 360°.

4. The number of pulses per circle = encoder resolution / electronic gear ratio

Step pulse equivalent setting:

Step angle: a pulse, the angle at which the stepper motor rotates.

Subdivision: subdivide the step angle into N equal parts, then one pulse, the angle at which the stepper motor rotates is "step angle/subdivision".

The number of pulses for one revolution of the stepping motor: 360/(step angle/subdivision)=360*subdivision/step angle

Reduction ratio: that is, the transmission ratio of the deceleration device, which is a kind of transmission ratio, and refers to the ratio of the instantaneous input speed to the output speed in the deceleration mechanism

Thread lead: It is the axial distance that any point on the thread moves along the same helix for one revolution. The pitch refers to the axial distance between the corresponding points of two adjacent teeth on the thread, and the code is P. The lead of a single-thread thread is equal to the pitch, and the lead of a multi-thread thread is equal to the number of heads (n) multiplied by the pitch, that is, the lead S=nP

If the mechanism has a lead (in mm) and a reducer, the pulse equivalent should be converted into mm as:

Pulse equivalent = lead/((360/(step angle/subdivision))*reduction ratio)=(lead*step angle)/(360*reduction ratio*subdivision)

Note:

1. The motor rotates a circle, and the reducer output rotates b circles, then the reduction ratio is a:b.

2. If the motor is directly connected to the screw, the mechanical reduction ratio is 1:1

3. If the shaft is a rotary axis, the lead is 360°.

4. Lead: when the motor rotates one revolution, any point on its propulsion surface advances in the axial direction.

5. The stepper motor defaults to take one step and turn at an angle of 1.8°. Then 1.8° is the value of the step angle. Therefore, the stepper motor defaults to 360/1.8=200 pulses per revolution, but in some institutions, the precision requirements are high, and 1.8° is not suitable, so there is a subdivision setting on the driver. For example, there is a 5 subdivision on the driver, that is, the angle is reduced by 5 times, that is, 1.8/5=0.36, which means that the angle of rotation of the motor every step is 0.36°, 360/0.36=1000 pulses per revolution.

The calculation methods of UNITS (pulse equivalent: ZAux_Direct_SetUnits) of several common mechanical transmissions are shown in the table below.

Mechanic Structure	Picture Example	Mechanic Specification	Encoder Resolution	UNITS

4.1.2.Routine

4.1.2.1. Set getting axis basic motion parameters

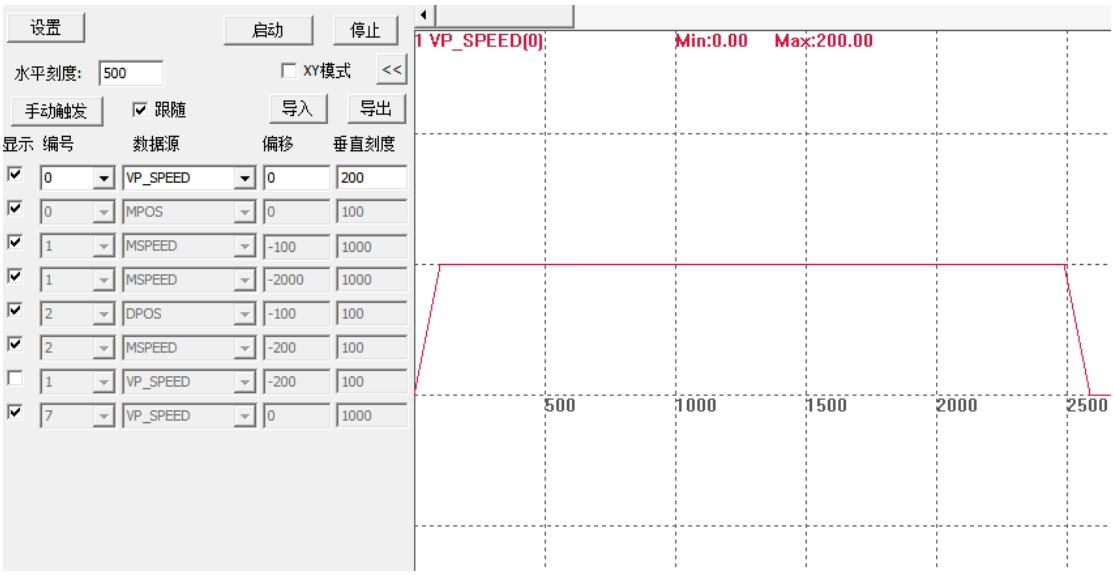
```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s return code is %d\n", command, ret);
	}

}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller factory IP address is 192.168.0.11, simulator IP address is 127.0.0.1, it needs to open the simulator when using simulator.
	ZMC_HANDLE handle = NULL;				//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		getchar();
		return -1;
	}
	printf("Success to connect controller!\n");

	ZAux_Direct_SetAtype(handle, 0, 1);	//set axis type of axis 0 as 1
	ZAux_Direct_SetUnits(handle, 0, 100); //set pulse amount of axis 0 as 100
```

```
    ZAux_Direct_SetSpeed(handle, 0, 200); //set speed of axis 0 as 200units/s
    ZAux_Direct_SetAccel(handle, 0, 2000); //set acceleration of axis 0 as
2000units/s/s
    ZAux_Direct_SetDecel(handle, 0, 2000); //set deceleration of axis 0 as
2000units/s/s

    float dpos;
    int ifIdle;
    int status;
    float DposValue;

    int Atype;
    float Units;
    float Speed;
    float Accel;
    float Decel;
    ret =ZAux_Direct_GetAtype(handle, 0, &Atype);      //get the axis type of axis 0
    commandCheckHandler("ZAux_Direct_GetAtype", ret);//judge whether the
instruction is executed successfully
    printf("axis type of axis 0Atype = %d\n", Atype);

    ret =ZAux_Direct_GetUnits(handle, 0, &Units); //get the pulse amount of axis 0
    commandCheckHandler("ZAux_Direct_GetUnits", ret);//judge whether the
instruction is executed successfully

    printf("pulse amount of axis 0Units = %f\n", Units);

    ret =ZAux_Direct_GetSpeed(handle, 0, &Speed); //get the speed of axis 0
    commandCheckHandler("ZAux_Direct_GetSpeed", ret);//judge whether the
instruction is executed successfully

    printf("speed of axis 0Speed = %f\n", Speed);

    ret =ZAux_Direct_GetAccel(handle, 0, &Accel); //get the acceleration of axis 0
    commandCheckHandler("ZAux_Direct_GetAccel", ret);//judge whether the
instruction is executed successfully

    printf("acceleration of axis 0Accel = %f\n", Accel);

    ret =ZAux_Direct_GetDecel(handle, 0, &Decel); //get the deceleration of axis 0
    commandCheckHandler("ZAux_Direct_GetDecel", ret);//judge whether the
instruction is executed successfully

    printf("deceleration of axis 0Decel = %f\n", Decel);

    ret = ZAux_Direct_SetDpos( handle,  0,  0);//axis instruction position clearing
```

```
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//judge whether the
instruction is executed successfully

    ret = ZAux_Direct_SetMpos( handle,  0,  0);//encoder feedback position clearing
    commandCheckHandler("ZAux_Direct_SetMpos", ret);//judge whether the
instruction is executed successfully


    ZAux_Trigger(handle);//oscilloscope trigger function
    ret =ZAux_Direct_Single_Move( handle,  0,  500);
    printf("drive axis 0 moves 500units\n");
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//judge whether the
instruction is executed successfully

    Sleep(1000);

    ret = ZAux_Direct_GetDpos(handle, 0, &dpos);      //get dpos
    commandCheckHandler("ZAux_Direct_GetDpos", ret) ;//judge whether the
instruction is executed successfully

    printf("after 1 second, axis 0 axis instruction position dpos = %f\n", dpos);

    ret = ZAux_Direct_GetMpos(handle, 0, &DposValue);
    commandCheckHandler("ZAux_Direct_GetMpos", ret) ;//judge whether the
instruction is executed successfully

    printf("after 1 second, axis 0 encoder feedback position Mpos = %f\n",
DposValue);

   ret = ZAux_Direct_GetIfIdle(handle, 0, &ifIdle);   //get whether the current axis is
 idle
    commandCheckHandler("ZAux_Direct_GetIfIdle", ret) ;//judge whether the
instruction is executed successfully

    printf("after 1 second, axis 0 motion state Idle: %s\n", ifIdle ? "Stop" : "Running");

    ret = ZAux_Direct_GetAxisStatus(handle, 0, &status);  //get the axis state
    commandCheckHandler("ZAux_Direct_GetAxisStatus", ret) ;//judge whether the
instruction is executed successfully
    printf("axis state AXISTATUS：  %x h\n", status);

  Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
```

```
    handle = NULL;
    return 0;
}
```

Out put waveform figure:



4.1.2.2. Set getting axis basic parameters (with initial speed & S curve)

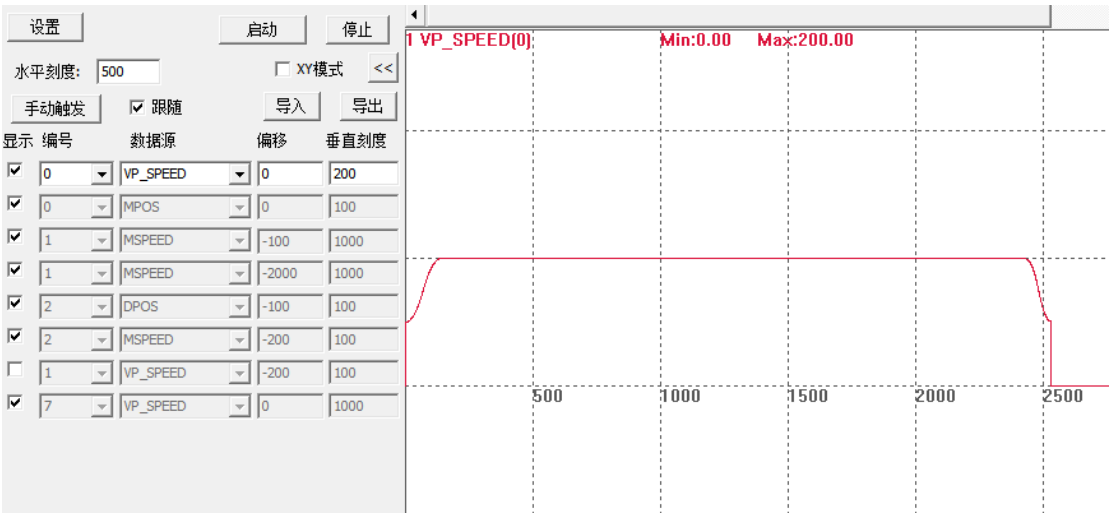
```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)// it is not 0, fail
    {
        printf("%s return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
```

```
    char *ip_addr = (char *)"127.0.0.1";            //controller IP address
    ZMC_HANDLE handle = NULL;                 // link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   // connect to controller
    if (ERR_SUCCESS != ret)
    {

        printf("Fail to connect controller!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("Success to connect controller!\n");

    ZAux_Direct_SetAtype(handle, 0, 1);  //set axis type of axis 0 as 1
    ZAux_Direct_SetUnits(handle, 0, 100); //set pulse amount of axis 0 as 100
    ZAux_Direct_SetSpeed(handle, 0, 200); //set speed of axis 0 as 200units/s
    ZAux_Direct_SetAccel(handle, 0, 2000); //set acceleration of axis 0 as
2000units/s/s
    ZAux_Direct_SetDecel(handle, 0, 2000); //set deceleration of axis 0 as
2000units/s/s
    ZAux_Direct_SetLspeed(handle, 0, 100); //set starting speed of axis 0 as
20units/s
    ZAux_Direct_SetSramp(handle, 0, 200);//set S curve time of axis 0 as 200ms
    float dpos;
    int ifIdle;
    int status;
    float DposValue;

    int Atype;
    float Units;
    float Speed;
    float Accel;
    float Decel;
    float Lspeed;
    float Sramp;

    ret =ZAux_Direct_GetAtype(handle, 0, &Atype);      //get the axis type of axis 0
    commandCheckHandler("ZAux_Direct_GetAtype", ret);//judge whether the
instruction is executed successfully
    printf("axis type of axis 0 Atype = %d\n", Atype);

    ret =ZAux_Direct_GetUnits(handle, 0, &Units); //get pulse amount of axis 0
    commandCheckHandler("ZAux_Direct_GetUnits", ret);//judge whether the
instruction is executed successfully
    printf("pulse amount of axis 0 Units = %f\n", Units);
```

```
    ret =ZAux_Direct_GetSpeed(handle, 0, &Speed); //get the speed of axis 0
    commandCheckHandler("ZAux_Direct_GetSpeed", ret);//judge whether the
instruction is executed successfully
    printf("axis 0 speed Speed = %f\n", Speed);

    ret =ZAux_Direct_GetAccel(handle, 0, &Accel); //get acceleration of axis 0
    commandCheckHandler("ZAux_Direct_GetAccel", ret);//judge whether the
instruction is executed successfully
    printf("axis 0 acceleration Accel = %f\n", Accel);

    ret =ZAux_Direct_GetDecel(handle, 0, &Decel); //get deceleration of axis 0
    commandCheckHandler("ZAux_Direct_GetDecel", ret);//judge whether the
instruction is executed successfully
    printf("axis 0 deceleration Decel = %f\n", Decel);

    ret =ZAux_Direct_GetLspeed(handle, 0, &Lspeed); //get starting speed of axis 0
    commandCheckHandler("ZAux_Direct_GetLspeed", ret);//judge whether the
instruction is executed successfully
    printf("axis 0 starting speed Lspeed = %f\n", Lspeed);

    ret =ZAux_Direct_GetSramp(handle, 0, &Sramp); //get S curve time of axis 0
    commandCheckHandler("ZAux_Direct_GetDecel", ret);//judge whether the
instruction is executed successfully
    printf("axis 0 S curve time Sramp = %f\n", Sramp);

    ret = ZAux_Direct_SetDpos( handle,  0,  0);//axis instruction position clearing
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//judge whether the
instruction is executed successfully
    ret = ZAux_Direct_SetMpos( handle,  0,  0);//encoder feedback position clearing
    commandCheckHandler("ZAux_Direct_SetMpos", ret);//judge whether the
instruction is executed successfully

    ZAux_Trigger(handle);//oscilloscope trigger function
    ret =ZAux_Direct_Single_Move( handle,  0,  500);
    printf("drive axis 0 moves 500units\n");
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//judge whether the
instruction is executed successfully
    Sleep(1000);

    ret = ZAux_Direct_GetDpos(handle, 0, &dpos);      //get dpos
    commandCheckHandler("ZAux_Direct_GetDpos", ret) ;//judge whether the
instruction is executed successfully
    printf("after 1 second, axis 0 instruction position dpos = %f\n", dpos);

    ret = ZAux_Direct_GetMpos(handle, 0, &DposValue);
```

```
    commandCheckHandler("ZAux_Direct_GetMpos", ret) ;//judge whether the
instruction is executed successfully
    printf("after 1 second, axis 0 encoder feedback position Mpos = %f\n",
DposValue);

    ret = ZAux_Direct_GetIfIdle(handle, 0, &ifIdle);   //get whether the current axis is
 idle
    commandCheckHandler("ZAux_Direct_GetIfIdle", ret) ;//judge whether the
instruction is executed successfully
    printf("after 1 second, axis 0 motion state Idle: %s\n", ifIdle ? "Stop" : "Running");

    ret = ZAux_Direct_GetAxisStatus(handle, 0, &status);  //get axis state
    commandCheckHandler("ZAux_Direct_GetAxisStatus", ret) ;//judge whether the
instruction is executed successfully
    printf("axis state AXISSTATUS：%x h\n", status);

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

Output waveform figure:



4.2. Other Parameters

4.2.1. Emphasis

The instruction list mainly classifies the status settings of all other axis parameters, and the planning is divided into parameter categories and status categories. The relatively basic parameter status acquisition complements the acquisition of relevant information in various situations. For example, ZAux_GetModbusDpos, ZAux_GetModbusMpos, and ZAux_GetModbusCurSpeed are obtained in batches by obtaining the value of the register Modbus_IEEE.

4.2.1.1. Axis Address

1. **When ZCAN expands axes, there is an 8-bit DIP switch on the expansion board (hardware version should be V1.3 or above).**

Please note ZCAN expansion axes don't exceed 2 because it is limited by the bus bandwidth.

ZAux_Direct_SetAxisAddress must be used firstly to set the axis address, and then use ZAux_Direct_SetAtype to set the ZCAN extended axis type. After modification, you must use ZAux_Direct_SetAtype to set the axis type again. See example one.

Bit 1-4	CAN address DIP, the combination value is 0-15.
Bit 5-6	CAN speed DIP, different combination values are with different speeds.
Bit 7	Reserved for special functions.
Bit 8	120ohm resistor DIP, dial ON to conduct the resistor.

Rule:

ZAux_Direct_SetAxisAddress (controller connection handle, axis No., (32*0)+ID)//local axis interface 0 of the expansion board

ZAux_Direct_SetAxisAddress (controller connection handle, axis No., (32*1)+ID)//local axis interface 1 of the expansion board

2. **Map the axis No. of the bus driver, and map the connected drivers one by one according to the number.**

The drive numbers are arranged according to the wiring sequence, and the numbers start from 0 to the number of EtherCAT drives minus 1.

The drive No. is different from the device No. The device No. includes all the devices connected to the ECAT interface, but the drive No. only counts the connected drives.

The axis address must be set first, and then the axis type (ZAux_Direct_SetAxisAddress) of the ECAT axis must be set. After modification, the axis type (ZAux_Direct_SetAxisAddress) must be reset. See example two.

Bit 0-15	Drive No. + 1, 0-automatically specify
Bit 16-31	SLOT No. (when in multi-slot)

Rule:

ZAux_Direct_SetAxisAddress (controller connection handle, axis No., (slot No.<<16) + driver No. + 1)

3. Remap local pulse axis No., 4 series controllers support local pulse or encoder axis number remapping, firmware version should be 160608 and above.

When remapping, pay attention to first set the original pulse axis as a virtual axis. After modification, the axis type must be reset (ZAux_Direct_SetAtype). See example four.

Bit 0-15	Mapped local pulse axis No.
Bit 16-31	High 16-bit are all set as 1 (same as “high 16-bit =-1” in decimal system)

Rule:

Note: to remap the axis No. of axis 0 of the physical interface of the controller to axis 3, then axis 3 is the target axis number, and axis 0 is the remapped axis number.

ZAux_Direct_SetParam(controller connection handle, "ATYPE", remapped axis No.0, 0)

ZAux_Direct_SetParam(controller connection handle, "ATYPE", target axis No.3, 0)

ZAux_Direct_SetAxisAddress (controller connection handle, target axis No.3, (-1<<16) + remapped axis No.0)

ZAux_Direct_SetParam(controller connection handle, "ATYPE", target axis No.3, 1 or 7)

4.2.1.2. Motion Type

For Zmotion motion control cards, each motion has different motion types. For example, walking an interpolated straight line is one type of motion, and walking a circle center to draw an arc is another type of motion. At this time, motion type of the current motion can be obtained by calling the ZAux_Direct_GetMtype function. And the motion type of the next motion can be obtained by calling the ZAux_Direct_GetNtype function

The movement type form is as follows:

MTYPE	Motion Instruction Type
0	IDLE (no motion)
1	MOVE (single-axis linear motion or linear interpolation motion)
2	MOVEABS (absolute value single-axis linear motion of or absolute linear interpolation motion)
3	MHELICAL (circle center spiral movement)
4	MOVECIRC (circular interpolation)
5	MOVEMODIFY (modify motion position)
6	MOVESP (single-axis linear motion of SP speed or linear interpolation motion of SP speed)
7	MOVEABSSP (absolute value single-axis linear motion of SP speed or absolute linear interpolation motion of SP speed)
8	MOVECIRCSP (circular interpolation of SP speed)
9	MHELICALSP (circle center spiral movement of SP speed)
10	FORWARD, VMOVE (1) (forward continuous motion)
11	REVERSE, VMOVE(-1) (reverse continuous motion)
12	DATUMING (in homing motion)
13	CAM (cam table motion)
14	FWD_JOG (mapping forward JOG motion)
15	REV_JOG (mapping reverse JOG motion)
20	CAMBOX (follow cam table motion)
21	CONNECT (synchronous motion)
22	MOVELINK (automatic cam motion)
23	CONNPATH (synchronous motion 2, vector type)
25	MOVESLINK (automatic cam motion 2)
26	MOVESPRIAL (involute arc)
27	MECLIPSE, MECLIPSEABS, MECLIPSESP, MECLIPSEABSSP (elliptical motion)
28	MOVE_OP/MOVE_OP2, MOVE_TABLE, MOVE_PWM, MOVE_TASKMOVE_PARA, MOVE_ASYNMOVE, MOVE_AOUT (buffer IO/ buffer register operation, etc.)
29	MOVE_DELAY, MOVE_WAIT, MOVE_SYNMOVE (buffer delay)
31	MSPHERICAL, MSPHERICALSP (space arc)

32	MOVE_PT (motion distance in unit time)
33	CONNFRAME (robotic arm inverse kinematic motion)
34	CONNREFRAME (robotic arm forward kinematic motion)

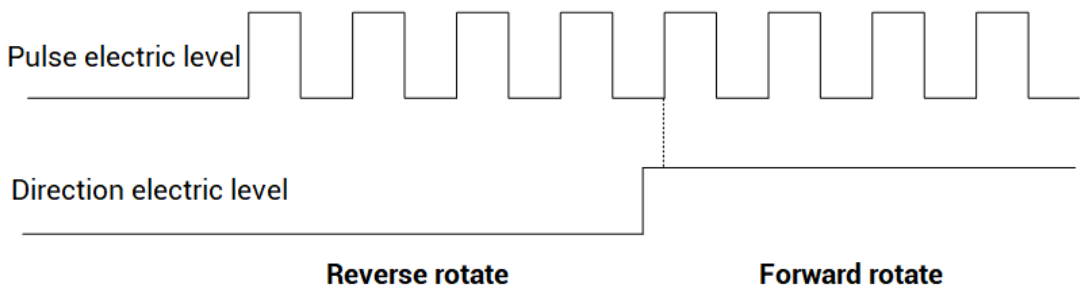
4.2.1.3. Pulse Mode

Pulse direction mode (PUL/DIR):

The PUL+ and PUL- pulse lines output the number of command pulses, the number of pulses corresponds to the running distance of the motor, and the pulse frequency corresponds to the running speed of the motor.

The DIR+ and DIR- direction lines output direction signals, and the different levels of the signals correspond to different rotation directions of the motor. This mode is the most in the driver.

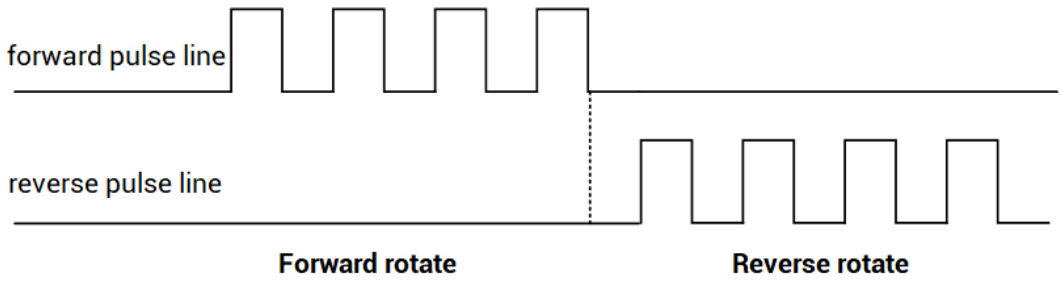
Corresponding to **ZAux_Direct_SetInvertStep** (handle, axis No., (0-3))



Dual pulse mode (CW/CCW):

Both lines output pulse signals, CW is the pulse signal in the forward direction, and CCW is the pulse signal in the reverse direction. They are usually output in differential mode. The phase difference angle of the two signals is determined according to the phase lead or lag.

Corresponding to **ZAux_Direct_SetInvertStep** (handle, axis No., (4-7))

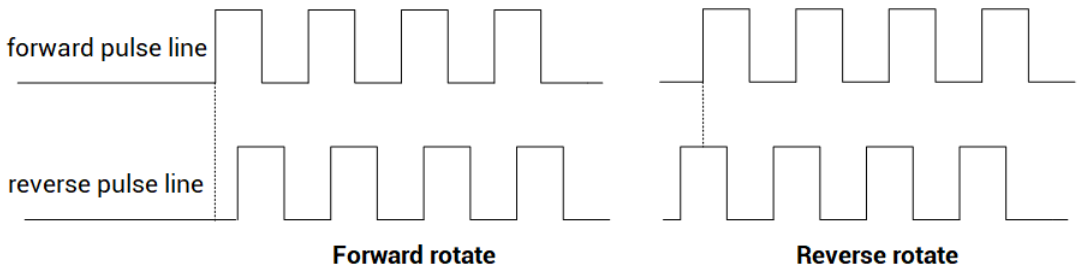


Quadrature pulse mode (A/B phase):

Orthogonal pulse refers to two identical pulse signals that are independent of each other. The positive direction pulse signal is generated before the negative direction pulse signal, and the phase difference between the two is 90 degrees. At this time, it is positive rotation, the negative direction pulse signal is generated before the positive direction pulse signal, the phase difference between the two is 90 degrees, and it is a negative rotation at this time.

The function of counting or encoding is achieved by the phase difference between two pulses.

Corresponding to **ZAux_Direct_SetInvertStep**(handle, axis No., (8-9))



4.2.2.Routine

The following routines will provide simple application examples of all the instructions mentioned in this chapter.

4.2.2.1. Local Pulse Axis No. Remapping

In this routine, remap axis No. of axis 0 to axis 3.

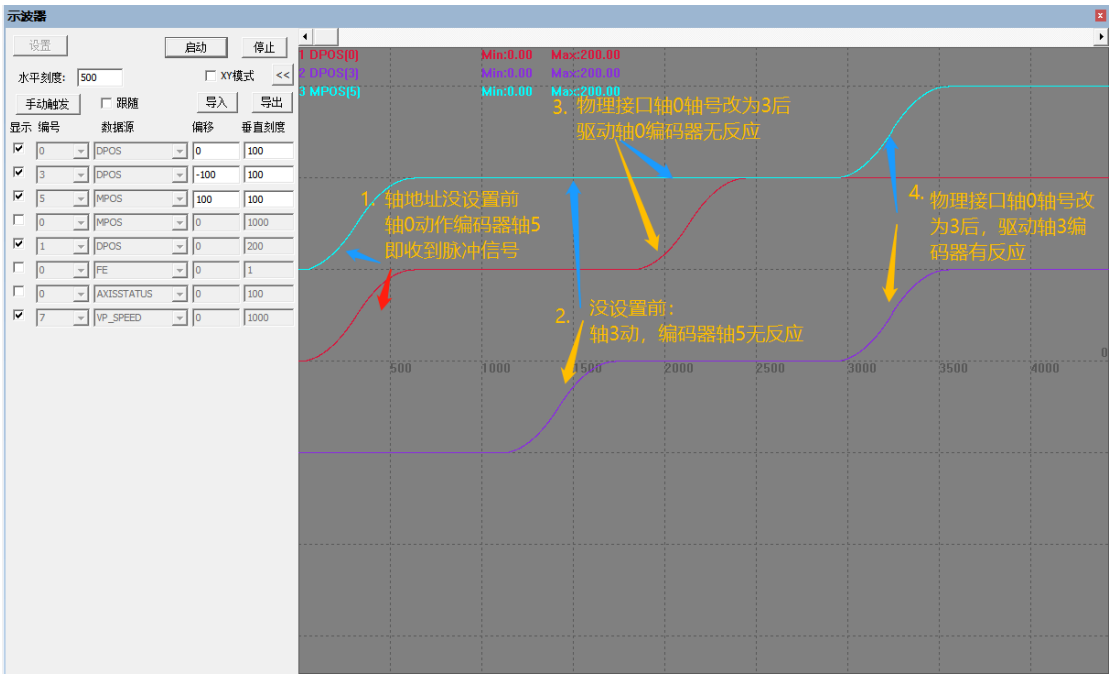
```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s return code is %d\n", command, ret);
	}

}

int _tmain(int argc, _TCHAR* argv[])
{
 char *ip_addr = (char *)"192.168.0.181";		//controller IP address, hardware
 wiring is needed to run the waveform in manual, simulator is useless.
	ZMC_HANDLE handle = NULL;		//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to connect controller！ \n");
	int i=0;
	//axis address is cleared to 0, reset
	for (i=0;i<10;i++)
	{
		ret =ZAux_Direct_SetAtype(handle,i,0);//clear ATYPE to 0 before set axis
address
		commandCheckHandler("ZAux_Direct_SetParam", ret) ;//judge whether the
instruction is executed successfully
		ret =ZAux_Direct_SetAxisAddress(handle,i,0);//set axis address
Axis_Address to clear to zero
		commandCheckHandler("ZAux_Direct_SetParam", ret) ;//judge whether the
instruction is executed successfully
	}
	int axis[2]={0,3};
	for (i=0;i<2;i++)
```

```
    {
    ret = ZAux_Direct_SetParam(handle,"ATYPE",axis[i],1);//use SetParam method to
set axis list, ATYPE=1
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//judge whether the
instruction is executed successfully

    ret = ZAux_Direct_SetParam(handle,"DPOS",axis[i],0);//use SetParam method to
set axis list, dpos is cleared to 0
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//judge whether the
instruction is executed successfully

    ret = ZAux_Direct_SetParam(handle,"ACCEL",axis[i],1000);//use SetParam
method to set axis list, acceleration is 1000
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//judge whether the
instruction is executed successfully

    ret = ZAux_Direct_SetParam(handle,"DECEL",axis[i],1000);//use SetParam
method to set axis list, deceleration is 1000
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//judge whether the
instruction is executed successfully

    ret = ZAux_Direct_SetParam(handle,"SPEED",axis[i],1000);//use SetParam
method to set axis list, speed is 1000
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//judge whether the
instruction is executed successfully

    }
    ret = ZAux_Direct_SetParam(handle,"DPOS",5,0);//use SetParam method to set
axis 5, dpos is cleared to 0
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//judge whether the
instruction is executed successfully


    ret = ZAux_Direct_SetParam(handle,"ATYPE",5,6);//use SetParam method to set
axis 5, axis type is set as 6, encoder axis, setAtype function also can be used.
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//judge whether the
instruction is executed successfully


    float len[2]={100,100};
    int AxisAddress0;
    int AxisAddress3;
    ret =ZAux_Direct_GetAxisAddress(handle,0,&AxisAddress0);//get axis 0 address
    commandCheckHandler("ZAux_Direct_GetAxisAddress", ret) ;//judge whether the
instruction is executed successfully
    printf("axis 0 address before modification：%d\n",AxisAddress0);
```

```
    ret =ZAux_Direct_GetAxisAddress(handle,3,&AxisAddress3);//get axis 3 address
    commandCheckHandler("ZAux_Direct_GetAxisAddress", ret) ;//judge whether the
instruction is executed successfully
    printf("axis 3 address before modification: Axis_Address：%d\n",AxisAddress3);
    printf("*****************************************************\n");
    ZAux_Trigger(handle);//trigger the oscilloscope brought by ZDevelop itself
    float IDLE;
    ZAux_Direct_Single_Move(handle,0,100);//axis 0 moves 100
    while (1)//wait for axis to complete the motion
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }
    Sleep(400);
    ZAux_Direct_Single_Move(handle,3,100);//axis 3 moves 100
    while (1)//wait for axis 3 to complete the motion
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",3,&IDLE);
        if (IDLE<0)break;
    }

    //{remap axis No. of physical interface axis 0 to axis 3, then axis 3 is the axis No.
that is to be modified, axis 0 is the axis No. that is to be remapped.

    ret = ZAux_Direct_SetParam(handle,"ATYPE",0,0);//use SetParam method to
remap axis 0,set axis type as virtual axis, setAtype function also can be used.
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetParam(handle,"ATYPE",3,0);//use SetParam method to set
axis type of target axis No.3 as virtual axis, setAtype function also can be used.
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetAxisAddress(handle,3,(-1<<16)+0);//remap axis No. of axis
0 into axis 3
    commandCheckHandler("ZAux_Direct_SetAxisAddress", ret) ;//judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetParam(handle,"ATYPE",3,1);//set axis type of target axis No.
as 1
    commandCheckHandler("ZAux_Direct_SetParam", ret) ;//judge whether the
instruction is executed successfully
```

```
    ret =ZAux_Direct_GetAxisAddress(handle,0,&AxisAddress0);//get axis 0 address
    commandCheckHandler("ZAux_Direct_GetAxisAddress", ret) ;//judge whether the
instruction is executed successfully
    printf("axis 0 address before modification Axis_Address：%d\n",AxisAddress0);

    ret =ZAux_Direct_GetAxisAddress(handle,3,&AxisAddress3);//get axis 3 address
    commandCheckHandler("ZAux_Direct_GetAxisAddress", ret) ;//judge whether the
instruction is executed successfully
    printf("axis 0 address before modification Axis_Address：%d\n",AxisAddress3);

    //for example: remap axis No. of physical interface axis 0 to axis 3, then axis 3 is
the axis No. that is to be modified, axis 0 is the axis No. that is to be remapped.

    ZAux_Direct_Single_Move(handle,0,100);//axis 0 moves 100
    while (1)
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }
    Sleep(400);
    ZAux_Direct_Single_Move(handle,3,100);//axis 3 moves 100

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

It can be seen the axis No. of axis 0 is changed into 3 from the below waveform.



Output:

```
控制器连接成功!
修改前轴0轴地址为Axis_Address: 0
修改前轴3轴地址为Axis_Address: 0
**************************************************
修改后轴0轴地址为Axis_Address: 0
修改后轴3轴地址为Axis_Address: -65536
```

4.2.2.2. Parameters Getting & Configuration

```
// test1.cpp : define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)// it is not 0, fail
	{
		printf("%s return code is %d\n", command, ret);
	}
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//Controller IP address
	ZMC_HANDLE handle = NULL;			//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to connect controller!\n");

	ZAux_Direct_SetDpos(handle, 1, 100); //set axis 1 dpos=100
	ZAux_Direct_SetDpos(handle, 2, 200); //set axis 2 dpos=200
	ZAux_Direct_SetSpeed(handle, 0, 300); //set axis 0 speed as 300units/s
	ZAux_Direct_SetSpeed(handle, 1, 400); //set axis 1 speed as 400units/s
	ZAux_Direct_SetSpeed(handle, 2, 500); //set axis 2 speed as 500units/s

	float Mpos[3]={0};
	float Dpos[3]={0};
	float Speed[3]={0};
	int MaxSpeed;
	int Invert_Step;
	//{Invert_Step usage
	//0-3 pulse directional mode, pulse line + direction line
	//4-7 dual pulse method (or named as CW/CCW), forward pulse line + reverse
pulse line
	//8-9 AB output (some controllers need to be customized)
	printf("**********************************************\n");
	ZAux_Direct_GetInvertStep(handle, 0, &Invert_Step);//get pulse mode of axis 0
	commandCheckHandler("ZAux_GetModbusMpos", ret) ;//judge whether the
instruction is executed successfully
	printf("axis 0 pulse mode Invert_Step(0)：%d\n",Invert_Step);

	ZAux_Direct_SetInvertStep(handle, 0, 1);//set pulse mode of axis 0 as 1
	commandCheckHandler("ZAux_GetModbusMpos", ret) ;//judge whether the
instruction is executed successfully

	ZAux_Direct_GetInvertStep(handle, 0, &Invert_Step);//get pulse mode of axis 0
	commandCheckHandler("ZAux_GetModbusMpos", ret) ;//judge whether the
instruction is executed successfully
	printf("axis 0 pulse mode Invert_Step(0)(after modification)：%d\n",Invert_Step);
```

```
    ZAux_Direct_SetInvertStep(handle, 0, 0);//set pulse mode of axis as 0 (default mode)
    commandCheckHandler("ZAux_GetModbusMpos", ret) ;//judge whether the instruction is executed successfully
    printf("*********************************************\n");
    //Invert_Step usage}

    //{simple application of max_speed (max speed can not exceed the value)
    ret=ZAux_Direct_GetMaxSpeed(handle, 0, &MaxSpeed);//get max pulse output frequency of axis 0
    commandCheckHandler("ZAux_Direct_GetMaxSpeed", ret) ;//judge whether the instruction is executed successfully
    printf("max pulse output frequency of axis 0 Maxspeed: axis 0=%d\n",MaxSpeed);
    ret=ZAux_Direct_SetMaxSpeed(handle, 0, MaxSpeed+100);//modify max pulse frequency
    commandCheckHandler("ZAux_Direct_SetMaxSpeed", ret) ;//judge whether the instruction is executed successfully

    ret=ZAux_Direct_GetMaxSpeed(handle, 0, &MaxSpeed);
    commandCheckHandler("ZAux_Direct_GetMaxSpeed", ret) ;//judge whether the instruction is executed successfully
    printf("max pulse output frequency of axis 0 Maxspeed (after modification): axis 0=%d\n",MaxSpeed);
    ret=ZAux_Direct_SetMaxSpeed(handle, 0, MaxSpeed-100);//modify to max pulse frequency
    commandCheckHandler("ZAux_Direct_SetMaxSpeed", ret) ;//judge whether the instruction is executed successfully
    //simple application of max_speed (max speed can not exceed this value)}

    //use modbus method to get DPOS,MPOS,VP_SPEED
    printf("*********************************************\n");
    ret=ZAux_GetModbusMpos( handle, 3, Mpos);//read MPOS of axis 0-axis 2
    commandCheckHandler("ZAux_GetModbusMpos", ret) ;//judge whether the instruction is executed successfully
    printf("MPOS: axis 0=%f axis 1=%f axis 2=%f\n",Mpos[0],Mpos[1],Mpos[2]);

    ret=ZAux_GetModbusDpos(handle, 3, Dpos);
    commandCheckHandler("ZAux_GetModbusDpos", ret) ;//judge whether the instruction is executed successfully
    printf("DPOS: axis 0=%f axis 1=%f axis 2=%f\n",Dpos[0],Dpos[1],Dpos[2]);

    ret=ZAux_GetModbusCurSpeed(handle, 3, Speed);//read VP_SPEED of axis 0-axis 2(namely, in motion, pulse speed sent by controller is not ideal)
    commandCheckHandler("ZAux_GetModbusCurSpeed", ret) ;//judge whether the instruction is executed successfully
    printf("current planning speed VP_SPEED (before motion): axis 0=%f axis 1=%f
```

```
axis 2=%f\n\n",Speed[0],Speed[1],Speed[2]);
    //use modbus method to DPOS,MPOS,VP_SPEED}

    //{motion
    ret=ZAux_Direct_Single_Move(handle,0,200);//drive axis 0 moves 200
    commandCheckHandler("ZAux_Direct_Single_Move", ret) ;//judge whether the
instruction is executed successfully
    ret=ZAux_Direct_Single_Move(handle,1,200);// drive axis 1 moves 200
    commandCheckHandler("ZAux_Direct_Single_Move", ret) ;//judge whether the
instruction is executed successfully
    ret=ZAux_Direct_Single_Move(handle,2,200);// drive axis 2 moves 200
    commandCheckHandler("ZAux_Direct_Single_Move", ret) ;//judge whether the
instruction is executed successfully
    Sleep(200);
    //motion}

    //{read information again
    ret=ZAux_GetModbusMpos( handle, 3, Mpos);//read MPOS of axis 0 – axis 2
    commandCheckHandler("ZAux_GetModbusMpos", ret) ;//judge whether the
instruction is executed successfully
    printf("MPOS (after motion: axis 0=%f axis 1=%f axis 2 = %f\n", Mpos[0], Mpos[1],
Mpos[2]);

    ret=ZAux_GetModbusDpos(handle, 3, Dpos);
    commandCheckHandler("ZAux_GetModbusDpos", ret) ;//judge whether the
instruction is executed successfully
    printf("DPOS (after motion): axis 0=%f axis 1=%f axis 2 = %f\n", Dpos[0], Dpos[1],
Dpos[2]);

    ret=ZAux_GetModbusCurSpeed(handle, 3, Speed);//read VP_SPEED of axis 0 –
axis 3 (namely, when in motion, the pulse speed sent by controller)
    commandCheckHandler("ZAux_GetModbusCurSpeed", ret) ;//judge whether the
instruction is executed successfully
    printf("current planning speed VP_SPEED (after motion): axis 0=%f axis 1=%f axis
2=%f\n",Speed[0],Speed[1],Speed[2]);
    printf("**********************************************\n");
    //read information again}

    //{get the motion type
 //value is 0, no motion, value is more than 0, there is the motion, please refer to
 5.2.2.2 for details
    int Mtype=0;
    int Ntype=0;
    ret=ZAux_Direct_GetMtype(handle, 0,&Mtype);
    commandCheckHandler("ZAux_Direct_GetMtype", ret) ;//judge whether the
instruction is executed successfully
```

```
    printf("current motion type of axis 0 Mtype: axis 0 =%d \n",Mtype);

    ret=ZAux_Direct_GetNtype(handle, 0,&Ntype);
    commandCheckHandler("ZAux_Direct_GetNtype", ret) ;//judge whether the
instruction is executed successfully
    printf("next motion type of axis 0 that is running Ntype：=%d \n",Ntype);

    printf("*********************************************\n");
    //get the motion type}
    Sleep(2000);
    ret = ZAux_Close(handle);   //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

Program result outputs:

```
控制器连接成功!
*************************************************
轴0脉冲模式Invert_Step(0)：0
轴0脉冲模式Invert_Step(0)（修改后）：1
*************************************************
轴0最大脉冲输出频率Maxspeed：轴0=1000000
轴0最大脉冲输出频率Maxspeed（修改后）：轴0=1000100
*************************************************
反馈位置MPOS：轴0=-900.000000 轴1=100.000000 轴2=200.000000
规划位置DPOS：轴0=-900.000000 轴1=100.000000 轴2=200.000000
当前运动的规划速度VP_SPEED（运动前）：轴0=0.000000 轴1=0.000000 轴2=0.000000

反馈位置MPOS（运动后）：轴0=-879.900024 轴1=120.099998 轴2=220.100006
规划位置DPOS（运动后）：轴0=-879.900024 轴1=120.099998 轴2=220.100006
当前运动的规划速度VP_SPEED（运动后）：轴0=202.000000 轴1=202.000000 轴2=202.000000
*************************************************
轴0当前运动类型Mtype：轴0=1
轴0当前正在运动的下一条运动指令类型Ntype：=0
*************************************************
```

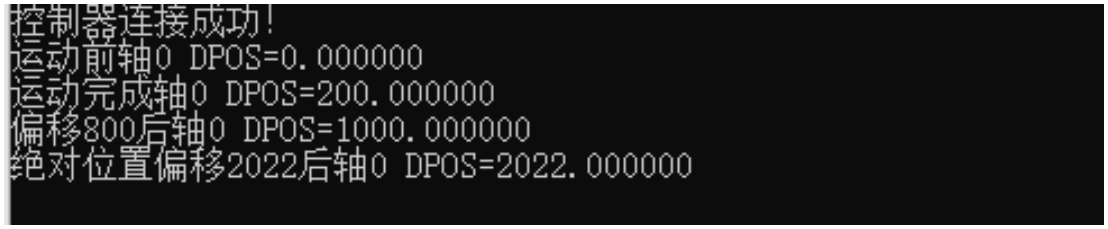
4.2.2.3. Axis Coordinates Configuration

```
// test1.cpp: define the entry point of control panel application program
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)// it is not 0, fail
```

```
    {
        printf("%s return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";          //controller IP address
    ZMC_HANDLE handle = NULL;                 //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to connect controller!\n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//set axis type
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetAccel(handle,0,500);//set axis acceleration
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetDecel(handle,0,500);//set axis deceleration
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetSpeed(handle,0,200);//set axis speed
    commandCheckHandler("ZAux_Direct_SetSpeed", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetDpos(handle,0,0);//set axis DPOS clearing
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully
    float DPOS;
    float OFFPOS;
    ret =ZAux_Direct_GetDpos(handle,0,&DPOS);//get axis DPOS value
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully
    printf("before motion, axis 0 DPOS=%f\n",DPOS);

    float IDLE;
    ZAux_Direct_Single_Move(handle,0,200);//axis 0 moves 200
    while (1)//wait for axis 0 to complete the motion
    {
        Sleep(100);
```

```
            ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
            if (IDLE<0)break;
        }

        ret =ZAux_Direct_GetDpos(handle,0,&DPOS);//get axis DPOS value
        commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully
        printf("after motion completed, axis 0 DPOS=%f\n",DPOS);

        ret =ZAux_Direct_SetOffpos(handle,0,800);//offset relatively axis 0 coordinates
        commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully

        //ret =ZAux_Direct_GetOffpos(handle,0,&OFFPOS);
        //commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully
        //printf("axis 0 offset OFFPOS=%f\n",OFFPOS);

        ret =ZAux_Direct_GetDpos(handle,0,&DPOS);// get axis DPOS value
        commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully
        printf("after offsets 800, axis 0 DPOS=%f\n",DPOS);

        ret =ZAux_Direct_Defpos(handle,0,2022);//offset axis 0 coordinates as absolute
position 2022.
        commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully

        ret =ZAux_Direct_GetDpos(handle,0,&DPOS);//get axis DPOS value
        commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully
        printf("after absolute position offsets 2022, the axis 0 DPOS=%f\n",DPOS);

        Sleep(2000);
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
        printf("connection closed!\n");
        handle = NULL;
        return 0;
}
```

Program result output: (only the coordinates are changed, the axis does not move)



4.2.2.4. Get Follow-up Errors

For the servo control system, under some abnormal conditions, the actual position of the motor may differ greatly from the planned position. At this time, there are usually some dangerous situations, such as motor failure, encoder A and B phase signals are reversed or disconnected, too much mechanical friction or mechanical failure causes the motor to stall, etc. In order to detect these situations in time, enhance the safety of the system and prolong the service life of the equipment, the motion controller provides a safety protection mechanism that can obtain the following error exceeding the limit.

The following routine is to obtain the following error of axis 0.

```
// test1.cpp: define the entry point of control panel application program

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)// it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}

}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;				//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
```

```
            printf("Fail to connect controller!\n");
            handle = NULL;
            Sleep(2000);
            return -1;
        }
           printf("Success to connect controller!\n");

        ret =ZAux_Direct_SetAtype(handle,0,5);//set axis type as 5, the axis type is with
encoder feedback, obvious following errors will be caused because there is no
encoder feedback wiring for simulator
        commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully
        ret =ZAux_Direct_SetAccel(handle,0,500);//set axis acceleration
        commandCheckHandler("ZAux_Direct_SetAccel", ret) ;// judge whether the
instruction is executed successfully
        ret =ZAux_Direct_SetDecel(handle,0,500);//set axis deceleration
        commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully
        ret =ZAux_Direct_SetSpeed(handle,0,200);//set axis speed
        commandCheckHandler("ZAux_Direct_SetSpeed", ret) ;// judge whether the
instruction is executed successfully
        ret =ZAux_Direct_SetDpos(handle,0,0);// set axis DPOS clearing
        commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the
instruction is executed successfully

        float IDLE;
        ZAux_Direct_Single_Move(handle,0,200);//axis 0 moves 200
        while (1)//wait for axis 0 to complete the motion
        {
            Sleep(100);
            ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
            if (IDLE<0)break;
        }

        float FE;
        float DPOS;
        float MPOS;
        ret =ZAux_Direct_GetDpos(handle,0,&DPOS);//set axis DPOS clearing
        commandCheckHandler("ZAux_Direct_GetDpos", ret) ;// judge whether the
instruction is executed successfully
        printf("current DPOS=%f\n",DPOS);
        ret =ZAux_Direct_GetMpos(handle,0,&MPOS);//set axis DPOS clearing
        commandCheckHandler("ZAux_Direct_GetMpos", ret) ;// judge whether the
instruction is executed successfully
        printf("current MPOS=%f\n",MPOS
            );
```

```
    ret =ZAux_Direct_GetFe(handle,0,&FE);//set axis DPOS clearing
    commandCheckHandler("ZAux_Direct_GetFe", ret) ;// judge whether the
instruction is executed successfully
    printf("current FE=%f\n",FE);

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;// judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

4.3. Axis Parameters Instruction List

4.3.1. Emphasis

Motion control parameters list:

Instruction	Description	Read & Write	No.
Atype	Axis type	R & W	0
Units	Pulse amount	R & W	1
Accel	Acceleration	R & W	2
Decel	Deceleration	R & W	3
Speed	Running speed	R & W	4
Creep	Crawling speed	R & W	5
Lspeed	Initial speed	R & W	6
Merge	Continuous interpolation switch	R & W	7
Sramp	Time configuration of acceleration and deceleration curve	R & W	8
Dpos	Axis instruction planning (demand) position	R & W	9
Mpos	Encoder feedback (measurement) position	R & W	10
Endmove	Current motion target position	Read only	11
Fs_Limit	Forward soft position limit setting	R & W	12
Rs_Limit	Reverse soft position limit setting	R & W	13
Datum_In	Map origin input	R & W	14
Fwd_In	Map forward hard position limit input	R & W	15
Rev_In	Map reverse hard position limit input	R & W	16

Idle	Motion state	Read only	17
Loaded	Whether the buffer is in blank state	Read only	18
Mspeed	Actual encoder feedback speed	Read only	19
Mtype	Current motion type	Read only	20
Ntype	Next motion type	Read only	21
Remain	Current motion remain distance	Read only	22
Vp_Speed	Current motion speed	Read only	23
Axisstatus	Axis state	Read only	24
Move_Mark	Motion mark	R & W	25
Move_Curmark	Current motion mark	Read only	26
Vector_Buffered	Current motion remain distance	Read only	27
Axis_StopReason	Axis stop reasons	Read only	28
Moves_Buffered	Current motion buffers	Read only	29
Axis_Address	Axis address	R & W	30
Axis_Enable	Single-axis enable	R & W	31
Force_Speed	Sp motion speed	R & W	32
Startmove_Speed	Sp motion starting speed	R & W	33
Endmove_Speed	Sp motion ending speed	R & W	34
Fastdec	Fast deceleration	R & W	35
Addax_Axis	Superposition axis No.	Read only	36
Link_Axis	Connecting axis No.	Read only	37
Corner_Mode	Corner mode	R & W	38
Decel_Angle	Corner deceleration starts	R & W	39
Stop_Angle	Corner deceleration ends	R & W	40
Full_Sp_Radius	Speed limit radius	R & W	41
Splimit_Radius	Speed limit value	R & W	42
Zsmooth	Chamfer radius	R & W	43
Vector_Moved	Currently moved distance	R & W	44
Endmove_Buffer	The end position of buffer	Read only	45
Homewait	Homing reverse delay	R & W	46
Fast_Jog	Map jog input	R & W	47
Fwd_Jog	Map forward Jog input	R & W	48
Rev_Jog	Map reverse Jog input	R & W	49
Jogspeed	Map Jog speed	R & W	50
Fhold_in	Map holding input	R & W	51
Fhspeed	Holding speed	R & W	52
Encoder	Encoder original value	Read only	53
Encoder_Status	Encoder state	Read only	54
PP_Step	Encoder inner ratio	R & W	55
Reg_Inputs	Latch input latch	R & W	56
Mark	Latch trigger	Read only	57
MarkB	Latch 1 trigger	Read only	58
MarkC	Latch 2 trigger	Read only	59
MarkD	Latch 3 trigger	Read only	60

Reg_Pos	Latch position	Read only	61
Reg_PosB	Latch 2 position	Read only	62
Reg_PosC	Latch 3 position	Read only	63
Reg_PosD	Latch 4 position	Read only	64
Alm_In	Map alarm input	R & W	65
Rep_Option	Coordinates loop mode	R & W	66
Rep_Dist	Coordinates loop position	R & W	67
Invert_Step	Pulse mode configuration	R & W	68
Max_speed	Pulse frequency limit	R & W	69
Axis_Zset	Precision output configuration	R & W	70
Dac	Bus axis analog control	R & W	71
ErrorMask	Operation when in error	R & W	72

The state of the above motion parameters can be set or acquired through the ZAux_Direct_SetParam and ZAux_Direct_GetParam functions, and the string commands are not case-sensitive when setting and acquiring.

4.3.2. Routine

4.3.2.1. Set & Obtain basic parameters by character string

```
// test1.cpp: define the entry point of control panel application program
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)// it is not 0, fail
	{
		printf("%s return code is %d\n", command, ret);
	}

}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;				//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
```

```
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("Success to connect controller!\n");

    ZAux_Direct_SetParam(handle,"ATYPE", 0, 1);       //set axis type of axis 0 as 1, string commands are case insensitive
    ZAux_Direct_SetParam(handle,"Units", 0, 100); //set pulse amount of axis 0 as 100, string commands are case insensitive
    ZAux_Direct_SetParam(handle,"SPEED", 0, 200); //set speed of axis 0 as 200units/s, string commands are case insensitive
    ZAux_Direct_SetParam(handle,"ACCEL",0, 2000); //set acceleration of axis 0 as 2000units/s/s, string commands are case insensitive
    ZAux_Direct_SetParam(handle,"DECEL", 0, 2000); //set deceleration of axis 0 as 2000units/s/s, string commands are case insensitive

    float dpos;
    float ifIdle;
    float status;
    float DposValue;

    float Atype;
    float Units;
    float Speed;
    float Accel;
    float Decel;

    ret =ZAux_Direct_GetParam(handle,"ATYPE", 0, &Atype);     //get axis type of axis 0, the type is 1
    commandCheckHandler("ZAux_Direct_GetAtype", ret);//judge whether the instruction is executed successfully
    printf("axis type of axis 0 Atype = %d\n", Atype);

    ret =ZAux_Direct_GetParam(handle, "Units",0, &Units); //get axis 0 pulse amount, it is 100
    commandCheckHandler("ZAux_Direct_GetUnits", ret);//judge whether the instruction is executed successfully
    printf("pulse amount of axis 0 Units = %f\n", Units);

    ret =ZAux_Direct_GetParam(handle,"Speed", 0, &Speed); //get axis 0 speed, 200units/s
    commandCheckHandler("ZAux_Direct_GetSpeed", ret);//judge whether the
```

```
instruction is executed successfully
    printf("speed of axis 0 Speed = %f\n", Speed);

    ret =ZAux_Direct_GetParam(handle,"Accel", 0, &Accel); //get axis 0 acceleration,
2000units/s/s
    commandCheckHandler("ZAux_Direct_GetAccel", ret);//judge whether the
instruction is executed successfully
    printf("acceleration of axis 0 Accel = %f\n", Accel);

    ret =ZAux_Direct_GetParam(handle,"Decel", 0, &Decel); //get axis 0 deceleration,
2000units/s/s
    commandCheckHandler("ZAux_Direct_GetDecel", ret);//judge whether the
instruction is executed successfully
    printf("deceleration of axis 0 Decel = %f\n", Decel);

    ret = ZAux_Direct_SetParam( handle,"DPOS",  0,  0);//axis DPOS is cleared
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//judge whether the
instruction is executed successfully

    ret = ZAux_Direct_SetParam( handle, "MPOS", 0,  0);//MPOS is cleared
    commandCheckHandler("ZAux_Direct_SetMpos", ret);//judge whether the
instruction is executed successfully

    ZAux_Trigger(handle);//oscilloscope trigger function
    ret =ZAux_Direct_Single_Move( handle,  0,  500);
    printf("drive axis 0 to move 500units\n");
    commandCheckHandler("ZAux_Direct_Single_Move", ret);//judge whether the
instruction is executed successfully
    Sleep(1000);

    ret = ZAux_Direct_GetParam(handle,"DPOS", 0, &dpos);      //get dpos
    commandCheckHandler("ZAux_Direct_GetDpos", ret) ;//judge whether the
instruction is executed successfully
    printf("after 1s, axis 0 instruction demand position dpos = %f\n", dpos);

    ret = ZAux_Direct_GetParam(handle,"MPOS", 0, &DposValue);
    commandCheckHandler("ZAux_Direct_GetMpos", ret) ;//judge whether the
instruction is executed successfully
    printf("after 1s, axis 0 encoder measurement position Mpos = %f\n", DposValue);

    ret = ZAux_Direct_GetParam(handle,"Idle", 0, &ifIdle);   //get whether the current
axis is idle
    commandCheckHandler("ZAux_Direct_GetIfIdle", ret) ;//judge whether the
instruction is executed successfully
    printf("after 1s, axis 0 motion state Idle: %s\n", ifIdle ? "Stop" : "Running");
```

```
    ret = ZAux_Direct_GetParam(handle,"AxisStatus", 0, &status);  //get axis state
    commandCheckHandler("ZAux_Direct_GetAxisStatus", ret) ;//judge whether the
instruction is executed successfully
    printf("axis state AXISSTATUS：%x h\n", status);

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

Chapter V Basic Motion Control

5.1.Motion Buffer

5.1.1.Emphasis

Motion instructions are usually used to achieve point to point, linear interpolation, circular interpolation, gear motion, etc. And do set axis parameters before motion.

Motion instructions have 2 types: single axis motion instruction and multi axes motion instructions. Single axis motion instruction only can operate one axis at one time, and it operates master axis by default, while multi axis instruction can operate multi axes at one time.

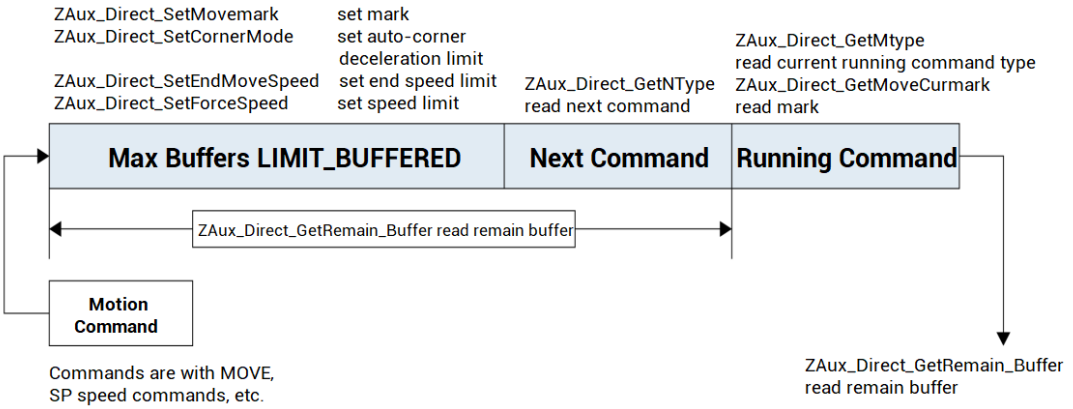
Instruction in cycle: get instruction, analyze instruction, execute instruction, then repeat the process above.

Zmotion motion controller contain multilevel motion buffer, when there is instruction in process, then new called motion instructions will enter buffer in order to avoid program blocking. When the multilevel motion buffer is full, and if new motion instructions are still called in program, the same, program will be blocked until there is space in buffer again.

Motion instructions are linear, circular and all kinds of interpolation motions and some instructions with prefix “MOVE”.

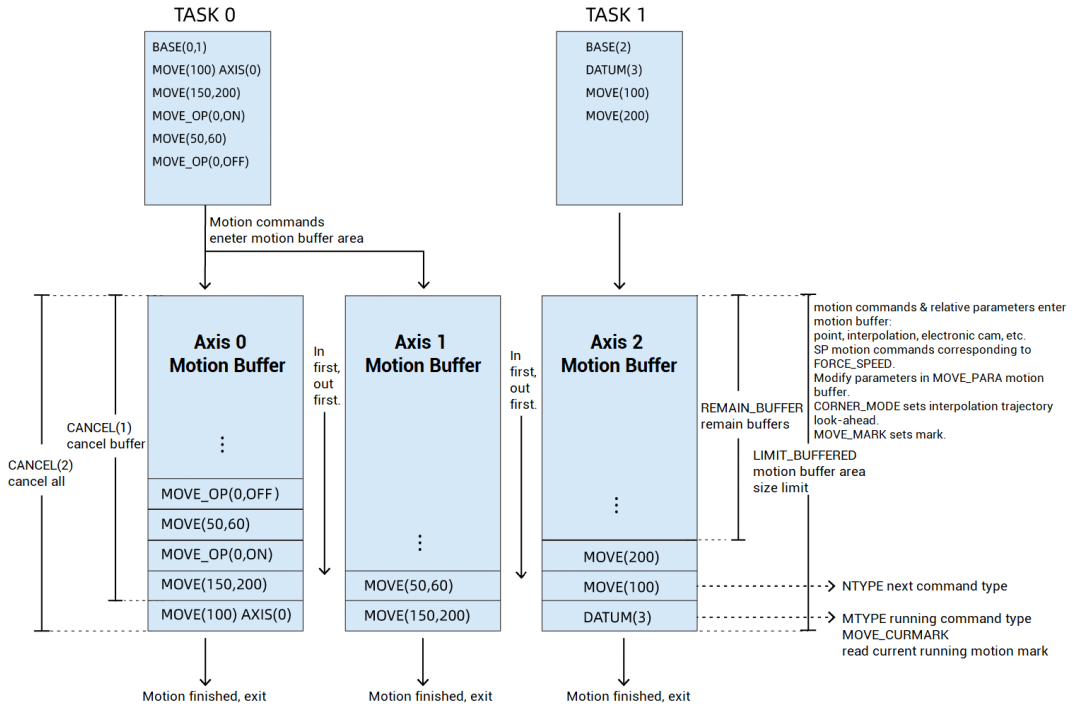
When using instructions with SP, ZAux_Direct_SetForceSpeed, ZAux_Direct_SetEndMoveSpeed and ZAux_Direct_SetStartMoveSpeedFORCE_SPEED instructions will follow SP type motion instructions to enter motion buffer.

Also, ZAux_Direct_MoveOp instruction can be used to enter IO operation command into buffer, in this way, IO output is added automatically between motion instructions. And use ZAux_Direct_MoveDelay to enter delay command into buffer, then, it will automatically delay between motion instructions.



Each axis has independent motion buffer, no conflict between each other, such as ZMC4 series, its motion buffer of each axis is 4096. See related hardware manual for details of motion buffer.

Take Zmotion ZMC406 controller as the example, motion buffer execution system is shown below.



Instructions support motion buffer: ZAux_Direct_MoveSp, ZAux_Direct_MoveAbsSp, ZAux_Direct_MoveCircSpSP, etc., namely, commands are with SP function suffix.

Instructions doesn't support motion buffer: ZAux_Direct_MoveModify, etc.

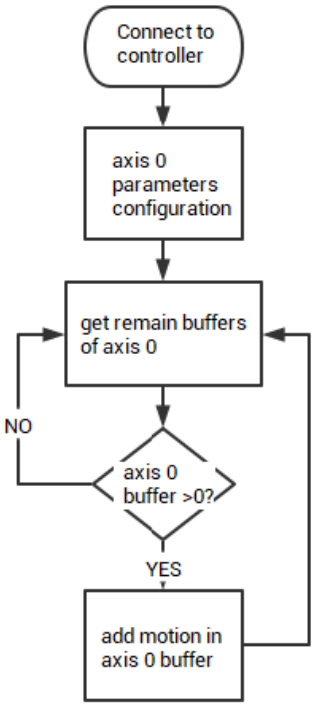
5.1.2.Routine

5.1.2.1. Get remain motion buffers

In the ZDevelop development environment, you can check the maximum motion buffer size by entering "?*max" in the information output window on the command line. For example, the MotionRT7 motion buffer size is max_movebuff:4096.

```
>>?*max

max_axis:64
max_motor:64
max_movebuff:4096
max_in:0, 4096
max_out:0, 4096
max_ain:0, 1000
max_aout:0, 1000
max_pwm:0
max_slot:4
max_slotecat:4, 128
max_comport:0
max_ethport:2
max_ethcustom:0
```

This routine obtains the remaining buffer number of the axis 0 buffer, and adds motion to the buffer according to whether the remaining buffer number is greater than 0.



```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)// it is not 0, fail
	{
		printf("%s return code is %d\n", command, ret);
	}

}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;				//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		getchar();
		return -1;
	}
	printf("Success to connect controller!\n");

	ZAux_Direct_SetAtype(handle, 0, 1);	//set axis type of axis 0 as 1
	ZAux_Direct_SetUnits(handle, 0, 100); //set pulse amount of axis 0 as 100
	ZAux_Direct_SetSpeed(handle, 0, 200); //set speed of axis 0 as 200units/s
	ZAux_Direct_SetAccel(handle, 0, 2000); //set acceleration of axis 0 as
2000units/s/s
	ZAux_Direct_SetDecel(handle, 0, 2000); //set deceleration of axis 0 as
2000units/s/s

	int MoveTime = 0;
	int GetValue;


	ret = ZAux_Direct_SetDpos( handle, 0 , 0);//clear DPOS position to 0
	commandCheckHandler("ZAux_Direct_SetDpos",ret);
```

```
    while (MoveTime < 100)
    {
        ret = ZAux_Direct_GetRemain_Buffer( handle, 0, &GetValue);//get remain
buffers of axis 0
        commandCheckHandler("ZAux_Direct_GetRemain_Buffer", ret);
        printf("Remain_Buffer = %d \n", GetValue);

        if (GetValue > 0)//if the number of axis 0 buffers is > 0
        {
        ret = ZAux_Direct_Single_Move( handle, 0, 10);//add the motion into axis 0
 buffer
            commandCheckHandler("ZAux_Direct_Single_Move", ret);
            MoveTime += 1;
        }
    }

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

5.1.2.2. Trigger Output in Motion (move_op2)

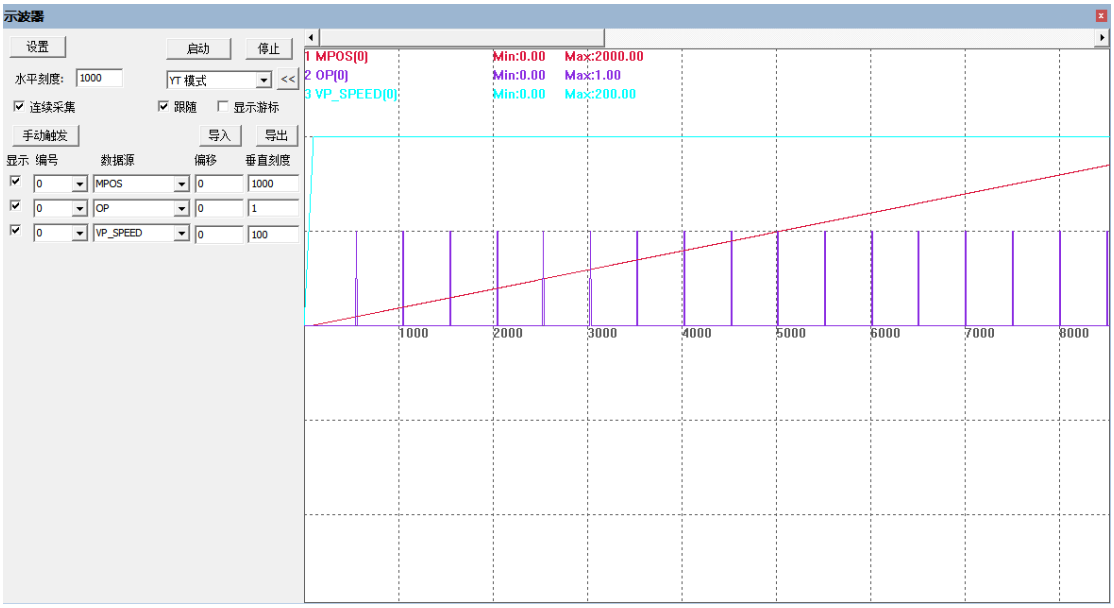
For example, suppose that in a certain actuator, 100 pulses are required to realize the movement of axis 0 by 1mm, the total movement of axis 0 is 2 meters, and the output port 0 is triggered every 100mm for dispensing.

```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)// it is not 0, fail
    {
        printf("%s return code is %d\n", command, ret);
    }
```

```
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;			//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		getchar();
		return -1;
	}
	printf("Success to connect controller!\n");

	ZAux_Direct_SetAtype(handle, 0, 1);	//set axis type of axis 0 as 4
	ZAux_Direct_SetUnits(handle, 0, 100); //set pulse amount of axis 0 as 100,
corresponding to 1mm

	ZAux_Direct_SetSpeed(handle, 0, 200); //set speed of axis 0 as 200units/s
	ZAux_Direct_SetAccel(handle, 0, 2000); //set acceleration of axis 0 as
2000units/s/s
	ZAux_Direct_SetDecel(handle, 0, 2000); //set deceleration of axis 0 as
2000units/s/s

	int MoveTime = 0;
	int GetValue;


	ret = ZAux_Direct_SetDpos( handle, 0 , 0);//clear DPOS position to 0
	commandCheckHandler("ZAux_Direct_SetDpos",ret);
	ret = ZAux_Direct_SetMpos( handle, 0 , 0);// clear MPOS position to 0
	commandCheckHandler("ZAux_Direct_SetMpos",ret);
	ret = ZAux_Direct_SetMerge( handle, 0 , 1);//open continuous interpolation
	commandCheckHandler("ZAux_Direct_SetMerge",ret);

	ret =ZAux_Trigger(handle);
	commandCheckHandler("ZAux_Trigger",ret);

	for (int i=0;i<20;i++)
	{
		ret = ZAux_Direct_Single_Move(handle, 0 , 100);
		commandCheckHandler("ZAux_Direct_Single_Move", ret);
		ret = ZAux_Direct_MoveOp2(handle, 0 , 0,1,10);
		commandCheckHandler("ZAux_Direct_MoveOp2", ret);
	}


	Sleep(2000);
	ret = ZAux_Close(handle);	//close the connection
	commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
	printf("connection closed!\n");
	handle = NULL;
```

```
    return 0;
}
```

It can be observed from the oscilloscope shown in the figure below that, under the condition that the speed does not drop, the OUT port 0 outputs once every 100 pulse equivalent positions:



5.2.Single-axis Motion

5.2.1.Emphasis

Each axis has its independent parameters, and parameters are global, which means it will take effect immediately when speed and acceleration changed, then dynamic speed can be achieved.

When ZAux_Direct_Single_Cancel is called to stop the motion command, it needs to decelerate by the value set by using the deceleration command ZAux_Direct_SetFastDec (the default is equal to the deceleration value). To achieve an emergency stop, the fast deceleration can be set to the maximum allowable value of the drive.

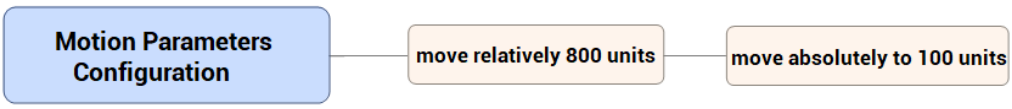
Single-axis continuous movement (ZAux_Direct_Single_Vmove) moves in one direction continuously. If the previous single-axis continuous motion is not stopped, the new single-axis continuous motion command will automatically replace the previous single-axis continuous motion and modify the direction, so there is no need to call ZAux_Direct_Single_Cancel to stop the previous single-axis continuous motion.

5.2.2.Routine

5.2.2.1. Single-axis Point Moiton

When do initialization, DPOS = 0, pulse amount (UNITS) = 100pulse/s, speed SPEED = 200 units/s, acceleration = 2000units/s/s, deceleration = 2000units/s/s, S curve acceleration and deceleration time is 200ms, and to do relative motion firstly, the distance is 800units, then do absolute motion, move absolutely to absolute position 100units.

Ps: the acceleration and deceleration time of the S-shaped curve is the change time of the motion acceleration and deceleration process, and the trajectory is S-shaped, not trapezoidal (T-shaped).

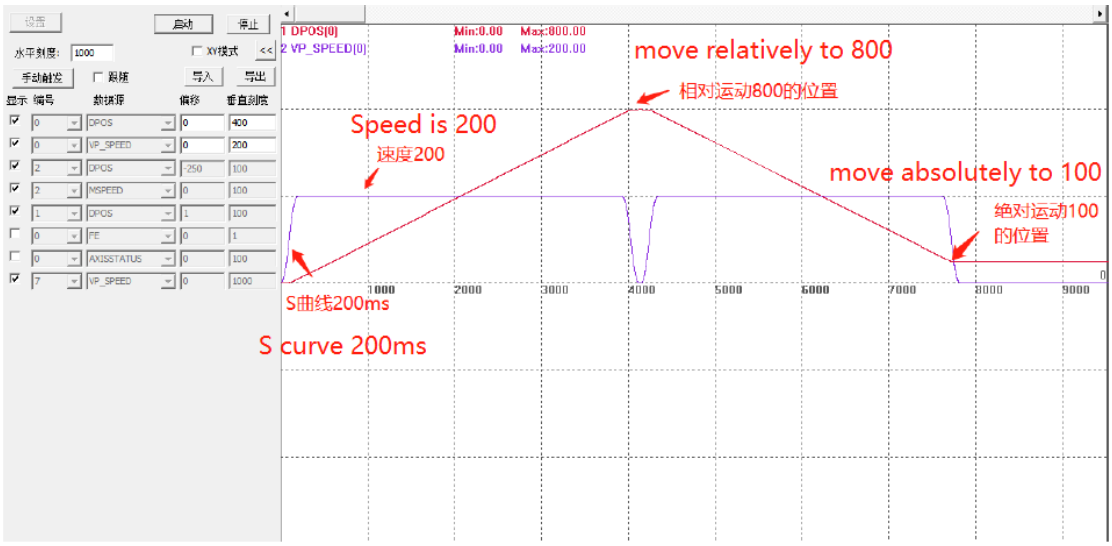



```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)// it is not 0, fail
	{
		printf("%s return code is %d\n", command, ret);
	}

}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;				//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
```

```
        printf("Fail to connect controller!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("Success to connect controller!\n");

    ZAux_Direct_SetAtype(handle, 0, 1);   //set axis type of axis 0 as 1
    ZAux_Direct_SetUnits(handle, 0, 100); //set pulse amount of axis 0 as 100
    ZAux_Direct_SetSpeed(handle, 0, 200); //set speed of axis 0 as 200units/s
    ZAux_Direct_SetAccel(handle, 0, 2000); //set acceleration of axis 0 as
2000units/s/s
    ZAux_Direct_SetDecel(handle, 0, 2000); //set deceleration of axis 0 as
2000units/s/s
    ZAux_Direct_SetSramp(handle, 0, 200); //set S curve of axis 0 as 200ms

    ret = ZAux_Direct_SetDpos( handle,  0,  0);//clear axis instruction position
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//judge whether the
instruction is executed successfully
    ret = ZAux_Direct_SetMpos( handle,  0,  0);//clear encoder feedback position
    commandCheckHandler("ZAux_Direct_SetMpos", ret);//judge whether the
instruction is executed successfully

    ZAux_Trigger(handle);//oscilloscope trigger function
    ret =ZAux_Direct_Single_Move( handle,  0,  800);
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//judge whether the
instruction is executed successfully

    float IDLE;
    while (1)//wait for axis 0 to complete the motion
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }
    ret =ZAux_Direct_Single_MoveAbs( handle,  0, 100);
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//judge whether the
instruction is executed successfully
    while (1)// wait for axis 0 to complete the motion
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

    Sleep(2000);
```

```
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

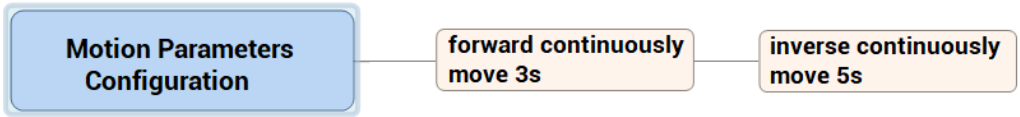
Waveform:



5.2.2.2. Single-axis Continuous Motion

Initially, the (DPOS) command position is 0, the (UNITS) pulse equivalent is set to 100, the speed is set to 200units/s, the acceleration is 2000units/s/s, the deceleration is 2000units/s/s, the acceleration and deceleration time of the S-shaped curve is set to 200ms, and the forward direction is continued first. Exercise for 3 seconds, then do negative continuous exercise for 5 seconds, and then stop.

Ps: S-shaped curve acceleration and deceleration time is the change time of the motion acceleration and deceleration process, and the trajectory is S-shaped, not trapezoidal (T-shaped).

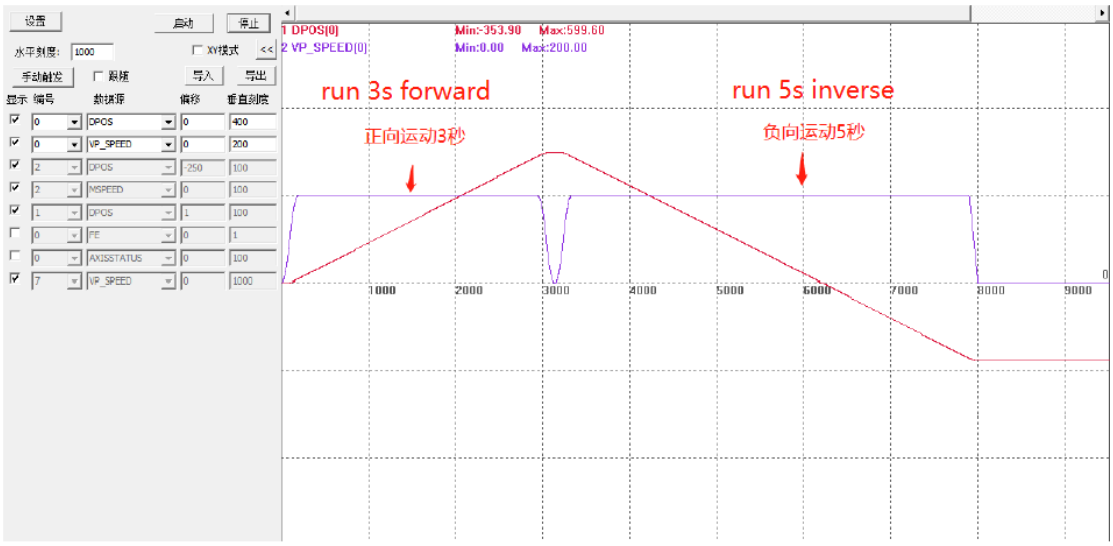



```
// test1.cpp: define the entry point of control panel application program
//
```

```
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//it is not 0, fail
    {
        printf("%s return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";          //controller IP address
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("Success to connect controller!\n");

    ZAux_Direct_SetAtype(handle, 0, 1);   //set axis type of axis 0 as 1
    ZAux_Direct_SetUnits(handle, 0, 100); //set pulse amount of axis 0 as 100
    ZAux_Direct_SetSpeed(handle, 0, 200); //set speed of axis 0 as 200units/s
    ZAux_Direct_SetAccel(handle, 0, 2000); //set acceleration of axis 0 as
2000units/s/s
    ZAux_Direct_SetDecel(handle, 0, 2000); //set deceleration of axis as
2000units/s/s
    ZAux_Direct_SetSramp(handle, 0, 200); //set S curve of axis 0 as 200ms

    ret = ZAux_Direct_SetDpos( handle,  0,  0);//clear axis command position
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//judge whether the
instruction is executed successfully
    ret = ZAux_Direct_SetMpos( handle,  0,  0);//clear encoder feedback position
    commandCheckHandler("ZAux_Direct_SetMpos", ret);//judge whether the
instruction is executed successfully

    ZAux_Trigger(handle);//oscilloscope trigger function
    ret =ZAux_Direct_Single_Vmove( handle,  0,  1);
```

```
        commandCheckHandler("ZAux_Direct_Single_Vmove", ret);//judge whether the
instruction is executed successfully

        Sleep(3000);//delay 3s

        ret =ZAux_Direct_Single_Vmove( handle,  0,  -1);
        commandCheckHandler("ZAux_Direct_SetDpos", ret);//judge whether the
instruction is executed successfully


        Sleep(5000);//delay 5s
        ret = ZAux_Direct_Single_Cancel(handle,0,2);//stop the motion
        commandCheckHandler("ZAux_Direct_Single_Cancel", ret) ;//judge whether the
instruction is executed successfully
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
        printf("connection closed!\n");
        handle = NULL;
        return 0;
}
```

Waveform:

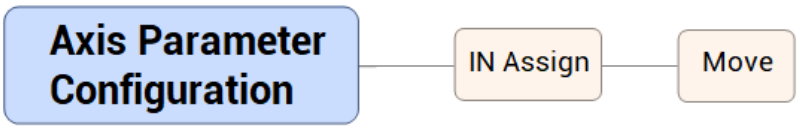


5.2.2.3. Quick Jog (IN Control)

The initial command is 0, the pulse equivalent is set to 100, the speed is set to 200units/s, the acceleration is 2000units/s/s, the deceleration is 2000units/s/s, and the S-curve acceleration and deceleration time is set to 200ms. Set the JOG speed to 100.

Map the input port IN0 as the input port for controlling whether to use JOG speed or SPEED speed, set the input port IN1 as the input port for positive direction motion control, and map the input port IN2 as the input port for negative direction motion control.

Ps: S-shaped curve acceleration and deceleration time is the change time of the motion acceleration and deceleration process, and the trajectory is S-shaped, not trapezoidal (T-shaped).

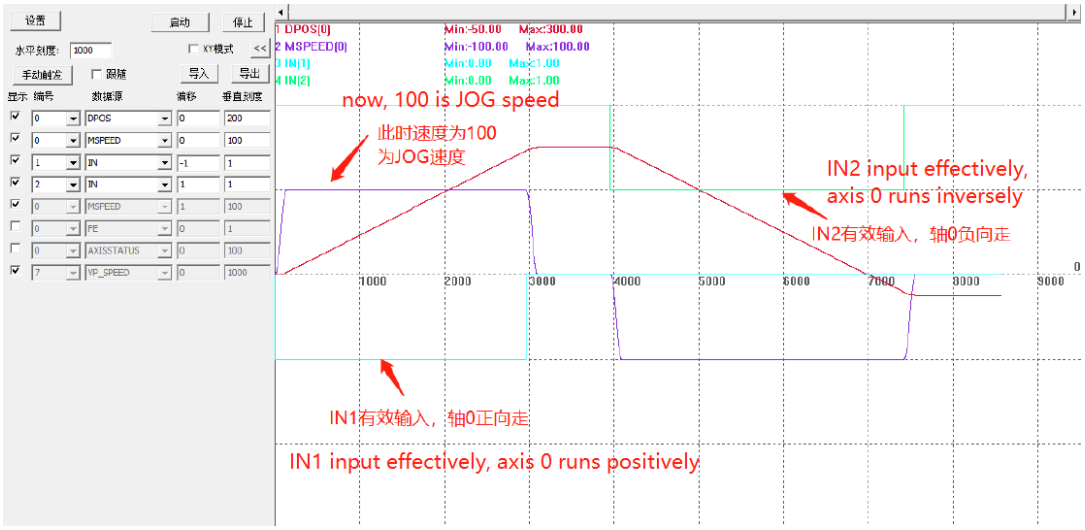



```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s return code is %d\n", command, ret);
	}

}
int32 __stdcall ZAux_Direct_ZSIMU_IN(ZMC_HANDLE handle,int link_axis,int
move_axis)
{


	char  cmdbuff[2048];
	char  cmdbuffAck[2048];
	//generate the command
	sprintf(cmdbuff, "ZSIMU_IN(%d,%d) ", link_axis,move_axis);


	//call command to execute the function
	return ZAux_DirectCommand(handle, cmdbuff, cmdbuffAck, 2048);
}

int _tmain(int argc, _TCHAR* argv[])
```

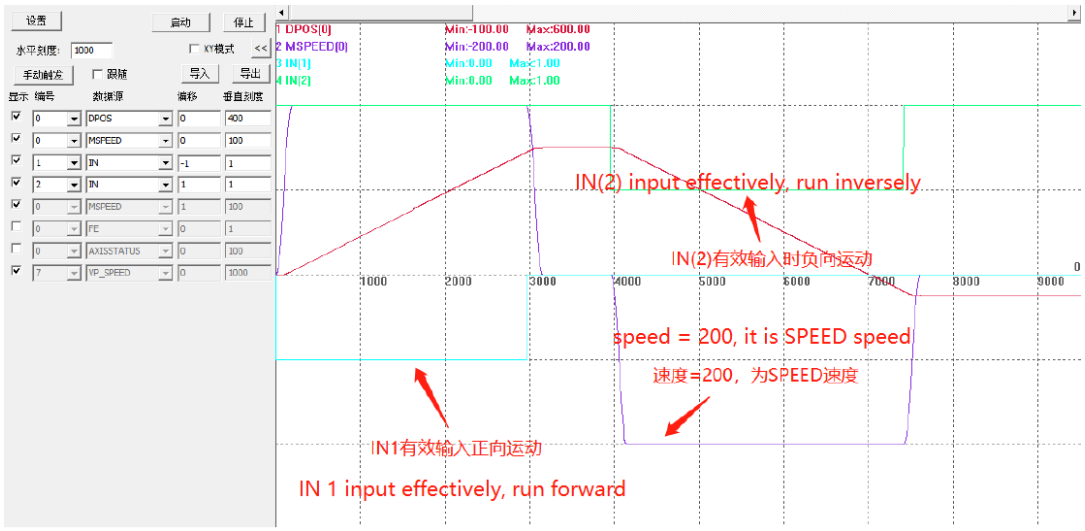
```
{
    char *ip_addr = (char *)"127.0.0.1";            //controller IP address
    ZMC_HANDLE handle = NULL;               //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("Success to connect controller!\n");

    ZAux_Direct_SetAtype(handle, 0, 1);   //set axis type of axis 0 as 1
    ZAux_Direct_SetUnits(handle, 0, 100); //set pulse amount of axis 0 as 100
    ZAux_Direct_SetSpeed(handle, 0, 200); //set speed of axis 0 as 200units/s
    ZAux_Direct_SetAccel(handle, 0, 2000); //set acceleration of axis 0 as
2000units/s/s
    ZAux_Direct_SetDecel(handle, 0, 2000); //set deceleration of axis 0 as
2000units/s/s
    ZAux_Direct_SetSramp(handle, 0, 200); //set S curve of axis 0 as 200ms

    ret = ZAux_Direct_SetDpos( handle,  0,  0);//clear axis command position
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//judge whether the
instruction is executed successfully
    ret = ZAux_Direct_SetMpos( handle,  0,  0);//clear encoder feedback position
    commandCheckHandler("ZAux_Direct_SetMpos", ret);//judge whether the
instruction is executed successfully

    ZAux_Direct_SetJogSpeed(handle, 0, 100);//set speed of JOG as 100
    ZAux_Direct_SetFastJog(handle, 0, 0);//set whether JOG speed opens control IN
    ZAux_Direct_SetFwdJog(handle, 0, 1);//set JOG forward control IN 0
    ZAux_Direct_SetRevJog(handle, 0, 2);// set JOG inverse control IN 1

    ZAux_Direct_SetInvertIn(handle, 0, 1);//reverse IN, valid input, ECI series control
cards doesn't use inverse
    ZAux_Direct_SetInvertIn(handle, 1, 1);// reverse IN, valid input, ECI series control
cards doesn't use inverse
    ZAux_Direct_SetInvertIn(handle, 2, 1);// reverse IN, valid input, ECI series control
cards doesn't use inverse
    ZAux_Trigger(handle);//oscilloscope trigger function

    Sleep(500);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
```

```
        printf("connection closed!\n");
        handle = NULL;
        return 0;
}
```

When IN(0) doesn't have valid input:



When IN(0) has valid input:



5.3.Homing

5.3.1.Emphasis

Original input is configured by **ZAux_Direct_SetDatumIn** command, positive and negative position limit is configured through **ZAux_Direct_SetFwdIn** and **ZAux_Direct_SetRevIn** commands.

Trigger is valid when ZMC series controller is 0. And when input is OFF, which means it achieves original position / limit position, do use **ZAux_Direct_SetFwdIn** to inverse the electricity level for common opened signals.

Trigger is valid when ECI series controller is 1. And when input is ON, which means it achieves original position / limit position, do use **ZAux_Direct_SetRevIn** to reverse the electricity level for common closed signals.

For Z signal homing, the axis type must be configured the type with signal Z.

When multiple axes are homing, each axis needs to use homing function.

When homing is finished in BUS (EtherCAT, RTEX, etc.) motion controller, MPOS is needed to be cleared by manual.

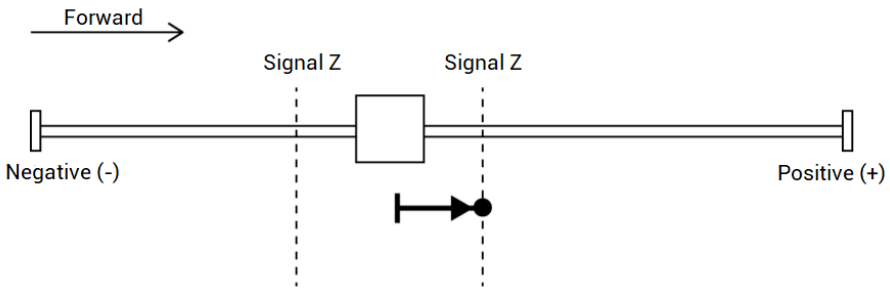
Homing mode:

Mode 1:

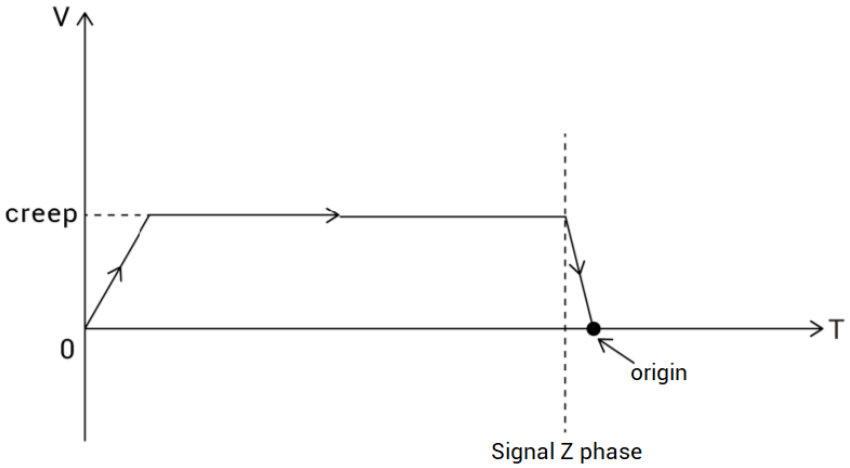
As shown in the figure below, **ZAux_Direct_Single_Datum** mode 1:

The axis runs forward at the speed of ZAux_Direct_SetCreep until the Z signal appears and starts to decelerate. After the deceleration reaches 0, the command position (DPOS) value is reset to 0 at this time, and the position after stopping is the origin. On the way back to zero If it touches the position limit switch, it will stop directly.

Homing mode 2 is the same type as mode 1, only the movement direction of finding the origin is opposite.



Speed Curve:



Z phase homing mode

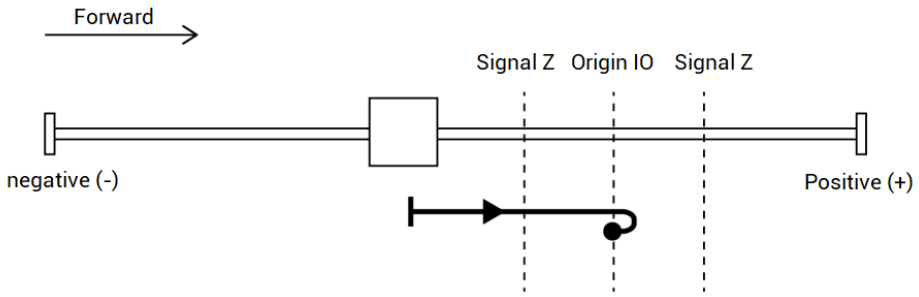
Note: After returning to zero and encountering the Z phase signal, it uses the deceleration of Decel to decelerate to 0 and stop. There is a small error, which can be reduced by reducing the speed of Creep and increasing the deceleration of Decel. If there is a high-precision homing, it needs to use the Z-phase latch and manually do the zero return to achieve the high-precision Z-phase homing.

Mode 3:

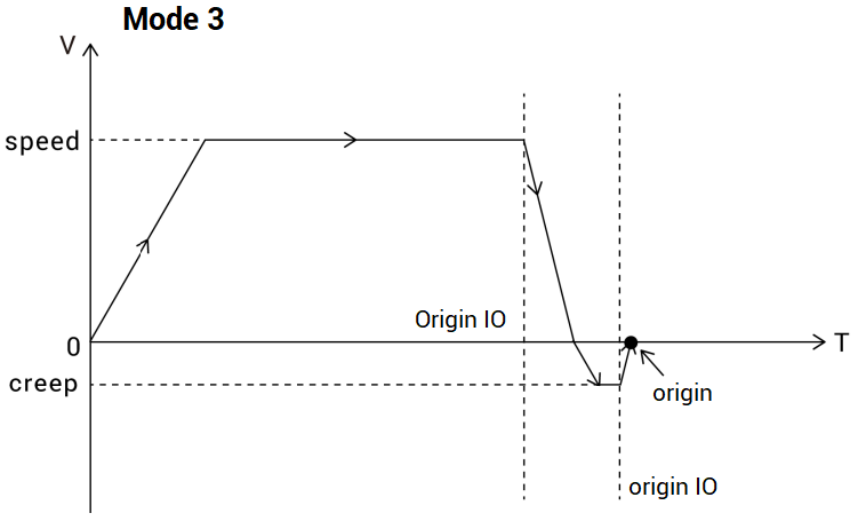
As shown in the figure below, **ZAux_Direct_Single_Datum** mode 3:

The axis runs forward quickly at the speed of **ZAux_Direct_SetSpeed** until it meets the origin switch and starts to decelerate. After decelerating to 0, it reverses to find the origin at the speed of **ZAux_Direct_SetCreep.** When meets the origin again, it will decelerate to stop, and the position value is reset to 0, the current position is the origin. If the limit switch is encountered on the way back to zero, it will stop directly.

Homing mode 4 is the same type as mode 3, only the movement direction of finding the origin is opposite.



Speed Curve:



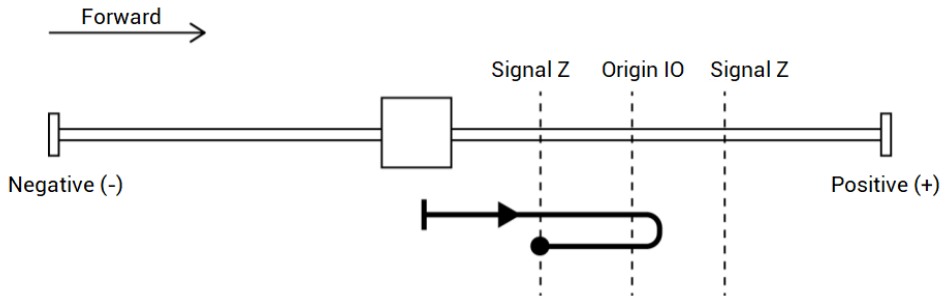
Note: After returning to zero and touching the origin signal, it decelerates to the creep speed, and moves in reverse until it meets the origin IO induction, and then decelerates to 0 to stop. There is a small error between leaving the IO induction and decelerating to 0. The error can be reduced by reducing the speed of Creep and increasing the deceleration of Decel. If there is a high-precision zero return, you need to manually do the zero return through the IO latch to achieve the high-precision.

Mode 5:

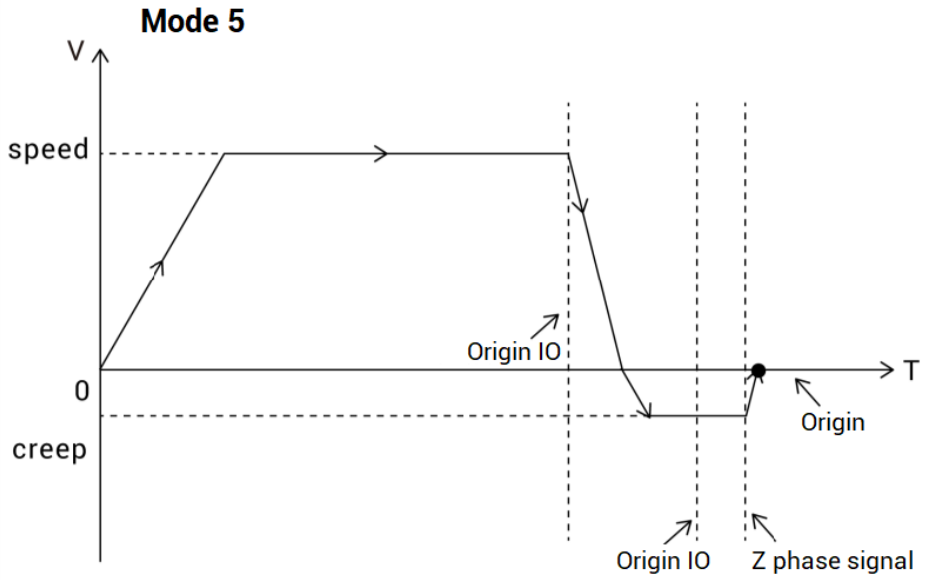
As shown in the figure below, **ZAux_Direct_Single_Datum** mode 5:

The axis runs forward quickly at the speed of **ZAux_Direct_SetSpeed** until it touches the origin switch and starts to decelerate. After decelerating to 0, it moves in the reverse direction at the speed of **ZAux_Direct_SetCreep** until the Z signal appears and then decelerates to stop. Reset the position value to 0, the position where the Z signal appears is the origin, and if it touches the limit switch on the way back to zero, it will stop directly.

Homing mode 6 is the same type as mode 5, only the movement direction of finding the origin is opposite.



Speed Curve:



Note: After returning to zero and touching the origin signal, use the Decel deceleration to decelerate to the creep speed, and move in the reverse direction until it leaves the origin IO induction. After encountering the Z phase signal, use the Decel deceleration to decelerate to 0 to stop. There is a small error between leaving the Z signal and decelerating to 0. But it can be reduced by reducing the speed of Creep and increasing the deceleration of Decel. If there is a high-precision zero return, you need to use the Z-phase latch to manually return to zero to achieve high-precision.

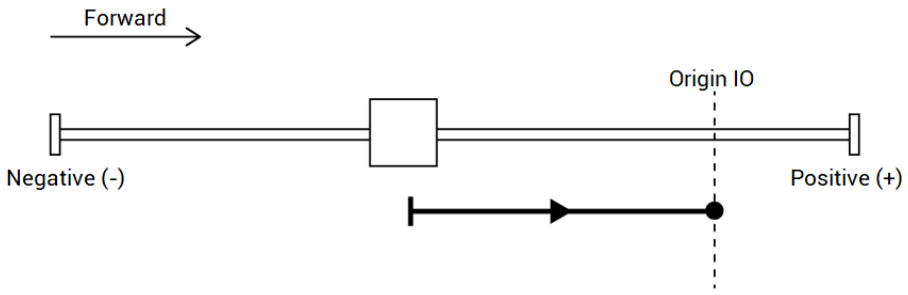
Mode 8:

As shown in the figure below, **ZAux_Direct_Single_Datum** mode 8:

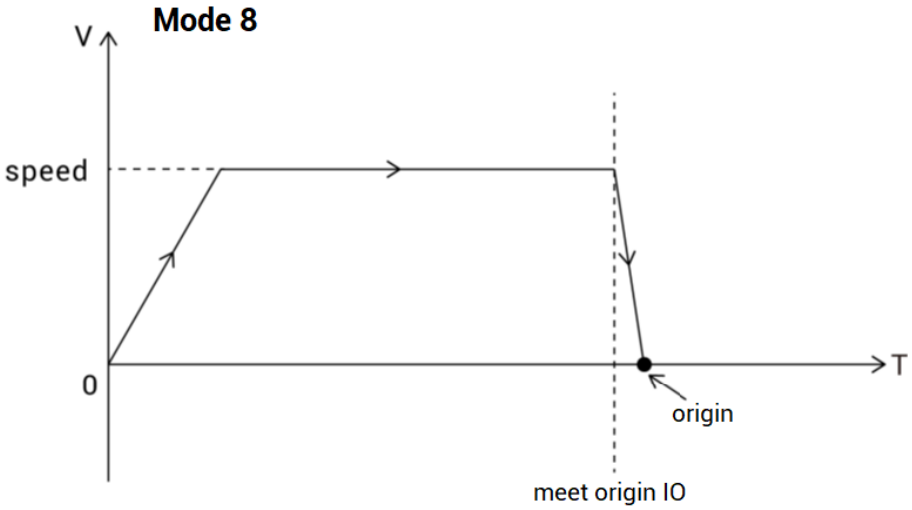
The axis runs forward quickly at the speed of **ZAux_Direct_SetSpeed**, and starts to

decelerate until it touches the origin switch. After decelerating to 0, the position value is reset to 0. After stopping, the position is the origin. It will stop directly when it hits the limit switch in homing process.

Homing mode 9 is the same type as mode 8, only the movement direction of finding the origin is opposite.



Speed Curve:



Note: After returning to zero and touching the origin signal, it uses Decel deceleration to decelerate to 0 and stop. There is a small error, which can be reduced by reducing the speed and increasing the Decel deceleration. If there is a high-precision zero return, you need to manually do the zero return through the IO latch to achieve the high-precision.

5.3.2.Routine

5.3.2.1. Single-axis Homing Motion

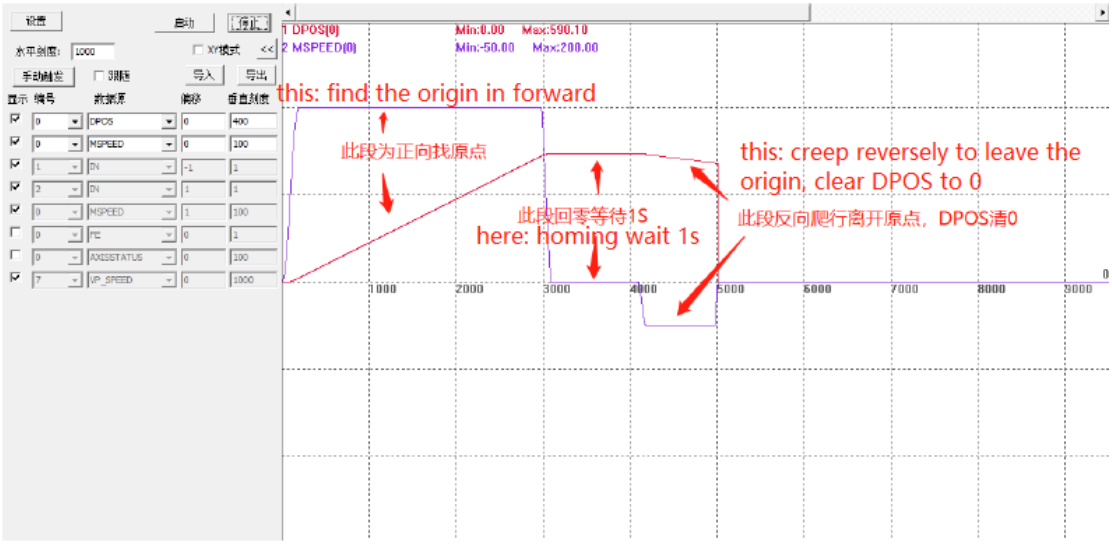
This routine adopts homing mode 3, and the axis runs forward at the speed of

ZAux_Direct_SetSpeed until it touches the origin switch. The axis then moves in reverse at the ZAux_Direct_SetCreep speed until it leaves the home switch. At the same time, the position will be automatically cleared to 0. In the example, IN0 is used as the origin switch, and the level inversion is turned on.


```
// test1.cpp: define the entry point of control panel application program
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)// it is not 0, fail
	{
		printf("%s return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;			//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		getchar();
		return -1;
	}
	printf("Success to connect controller!\n");

	ZAux_Direct_SetAtype(handle, 0, 1);	//set axis type of axis 0 as 1
	ZAux_Direct_SetUnits(handle, 0, 100); //set pulse amount of axis 0 as 100
	ZAux_Direct_SetSpeed(handle, 0, 200); //set speed of axis 0 as 200units/s
	ZAux_Direct_SetAccel(handle, 0, 2000); //set acceleration of axis 0 as
2000units/s/s
	ZAux_Direct_SetDecel(handle, 0, 2000); //set deceleration of axis 0 as
2000units/s/s
	ZAux_Direct_SetSramp(handle, 0, 200); //set S curve time of axis 0 as 200ms
	ret = ZAux_Direct_SetCreep(handle, 0, 50);	//set inverse creep speed when
```

```
homing
    commandCheckHandler("ZAux_Direct_SetCreep", 0);

    ret = ZAux_Direct_SetDatumIn(handle, 0, 0);     //set origin switch
    commandCheckHandler("ZAux_Direct_SetDatumIn", 0);

    ret = ZAux_Direct_SetHomeWait(handle, 0, 1000);  //set homing waiting time
    commandCheckHandler("ZAux_Direct_SetHomeWait", 0);

    ret = ZAux_Direct_SetInvertIn(handle, 0, 1);    //set input 0 electric level inverse
    commandCheckHandler("ZAux_Direct_SetInvertIn", 0);

    ret = ZAux_Direct_SetDpos( handle,  0,  0);//clear axis command position
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//judge whether the
instruction is executed successfully
    ret = ZAux_Direct_SetMpos( handle,  0,  0);//clear encoder feedback position
    commandCheckHandler("ZAux_Direct_SetMpos", ret);//judge whether the
instruction is executed successfully

    ZAux_Trigger(handle);//oscilloscope trigger function
    ret = ZAux_Direct_Single_Datum(handle, 0, 3);   //homing, mode 3
    commandCheckHandler("ZAux_Direct_Single_Datum", 0);

    uint32 homestatus;
    while (1)//wait for axis 0 to complete homing
    {
        Sleep(100);
        ZAux_Direct_GetHomeStatus(handle,0,&homestatus);//get homing motion
finish state
        if (homestatus==1)break;
    }

    Sleep(2000);
    ret = ZAux_Close(handle);   //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

Waveform:



5.4.Electronic Gear

5.4.1.Emphasis

The electronic gear function is used for the connection of two axes. The main axis and the slave axis are connected according to a constant gear ratio. There is no need for physical gears, and the ratio of the electronic gear can be directly set by using instructions. And the electronic gear ratio can be modified at any time because it is realized by the software.

The electronic gear function is realized through the instructions **ZAux_Direct_Connect** and **ZAux_Direct_Connpath**. Connect one axis to another axis for follow-up motion according to a certain ratio, then one motion command can drive the operation of two motors. By detecting the movement of the two motor axes, feedback the displacement deviation to the controller and obtain the synchronization compensation, so that the displacement deviation between the two axes can be controlled within the allowable range of precision.

The electronic gear is connected to the number of pulses. For example, the connection ratio of the master and slave axes is 1:5, and one pulse is sent to the master axis, and 5 pulses are correspondingly sent to the slave axis.

After 2 axes are connected by the synchronous command, it needs to use the “cancel” command to cancel the synchronous motion of the current axis, and then call other motions. When switching the synchronous override, you can directly call **ZAux_Direct_Connect** to replace the previous ZAux_Direct_Connect command. What is

connected is the number of pulses, and the ratio of different axis UNITS should be considered.

It is unlike cams, the connection of electronic gears is linear.

The role of electronic gear:

- Pulse compensation, reducing the burden on the host computer (because the current components used to send pulses have limitations on the frequency of sending pulses).
- Matching the number of pulses sent by the motor and the minimum movement of the machine, the movement of the workpiece (or the motor) corresponding to one pulse of the command input can be set to any value, and the infinitely variable speed of the motor can be realized. When the motor starts and stops, it can prevent out-of-step and overshoot, so that the potential of the motor can be fully utilized.
- Transmit synchronous motion information, realize coordinate linkage, transformation between motion forms (rotation-rotation, rotation-line, line-line), simplified control, etc.

The same point between **ZAux_Direct_Connpath** and **ZAux_Direct_Connect**: the usage syntax of the two is the same, the number of pulses is connected, and the effect of ZAux_Direct_Connpath connected to the movement of a single axis is the same as that of ZAux_Direct_Connect.

The difference between **ZAux_Direct_Connpath** and **ZAux_Direct_Connect**: ZAux_Direct_Connect connects the target position of a single axis. ZAux_Direct_Connpath connects the vector length of interpolation axis. At this time, it needs to be connected to the main axis of the interpolation movement, and it cannot follow the interpolation movement when connected to the interpolation slave axis. ZAux_Direct_Connpath will track the vector length change for XY interpolation instead of tracking individual X or Y axes.

5.4.2.Routine

5.4.2.1. Synchronous Motion

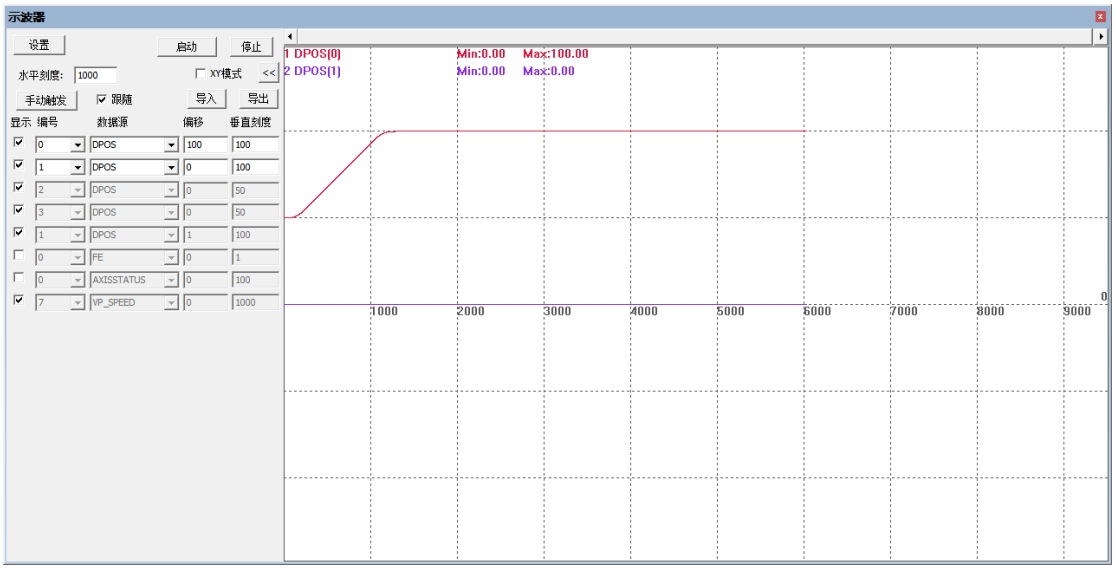
In this routine, axis 0 is driven to move a distance equivalent to 100 pulses, and the parameters of axis 1 and 0 are set to be the same, and the effect before and after synchronization is compared.

```
// test1.cpp: define the entry point of control panel application program.
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"


void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}

}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";  //controller IP address, there may be errors when using simulator to check the curve, it is best to connect to controller for viewing.
	ZMC_HANDLE handle = NULL;		//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		getchar();
		return -1;
	}
	printf("Success to connect controller！ \n");

	ret =ZAux_Direct_SetAtype(handle,0,1);//set axis type of axis 0 as 1
```

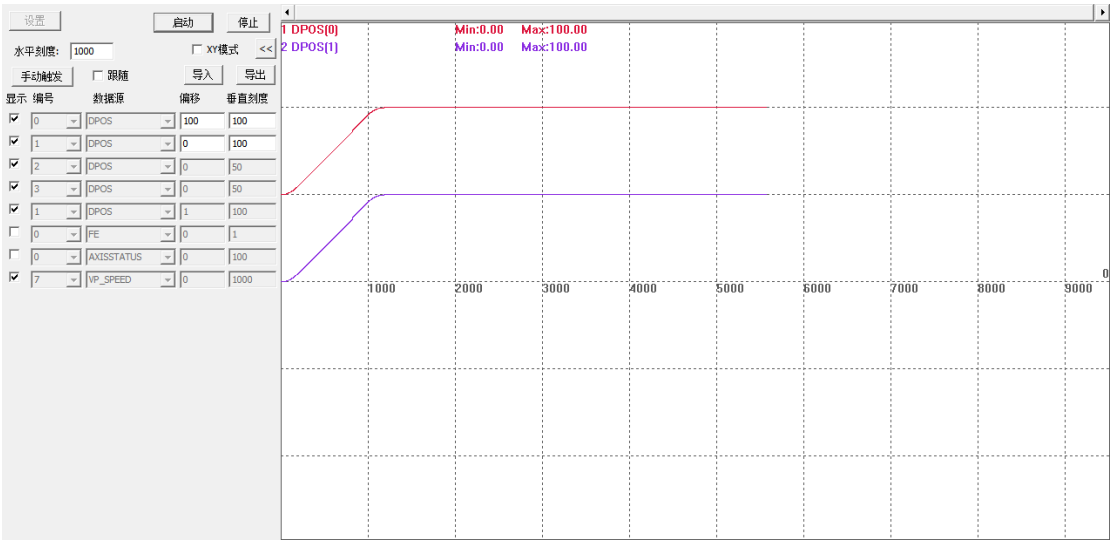
```
 commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//judge whether the
 instruction is executed successfully
     ret =ZAux_Direct_SetAtype(handle,1,1);// set axis type of axis 1 as 1
 commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//judge whether the
 instruction is executed successfully

     ret =ZAux_Direct_SetUnits(handle,0,100);//set pulse amount (UNITS) of axis 0 as
100
 commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//judge whether the
 instruction is executed successfully
     ret =ZAux_Direct_SetUnits(handle,1,100);// set pulse amount (UNITS) of axis 1 as
100
 commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//judge whether the
 instruction is executed successfully

     ret =ZAux_Direct_SetAccel(handle,0,500);//set the acceleration of axis 0
 commandCheckHandler("ZAux_Direct_SetAccel", ret) ;// judge whether the
 instruction is executed successfully
     ret =ZAux_Direct_SetAccel(handle,1,500);// set the acceleration of axis 1
 commandCheckHandler("ZAux_Direct_SetAccel", ret) ;// judge whether the
 instruction is executed successfully

     ret =ZAux_Direct_SetDecel(handle,0,500);//set the deceleration of axis 0
 commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
 instruction is executed successfully
     ret =ZAux_Direct_SetDecel(handle,1,500);//set the deceleration of axis 1
 commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
 instruction is executed successfully

     ret =ZAux_Direct_SetDpos(handle,0,0);//set clearing DPOS of axis 0 (to 0)
 commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the
 instruction is executed successfully
     ret =ZAux_Direct_SetDpos(handle,1,0);//set clearing DPOS of axis 1 (to 0)
 commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the
 instruction is executed successfully

     ret =ZAux_Direct_SetSpeed(handle,0,100);//set the speed of axis 0
 commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
 instruction is executed successfully

     ret =ZAux_Direct_SetSpeed(handle,1,100);// set the speed of axis 1
 commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
 instruction is executed successfully

     ret =ZAux_Direct_Connect(handle,1,0,1);//set open synchronous connection, axis
1 follows with axis 0
```

```
  commandCheckHandler("ZAux_Direct_Connect", ret) ;// judge whether the
  instruction is executed successfully

      float IDLE;
      ZAux_Trigger( handle);//trigger the oscilloscope

      ret = ZAux_Direct_Single_Move(handle, 0,100.0); //axis 1 moves 100
  commandCheckHandler("ZAux_Direct_Single_Move", ret) ;// judge whether the
  instruction is executed successfully


      while (1)//wait for axis 0 motion to complete
      {
            Sleep(100);
            ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
            if (IDLE<0)break;
      }
       ret    =ZAux_Direct_Single_Cancel(handle,0,2);//cancel    the    synchronization
motion
  commandCheckHandler("ZAux_Direct_Connect", ret) ;// judge whether the
  instruction is executed successfully

      getchar();
      ret = ZAux_Close(handle);    //close the link
  commandCheckHandler("ZAux_Close", ret) ;// judge whether the instruction is
  executed successfully
      printf("connection closed!\n");
      handle = NULL;
      return 0;
}
```

Before:



After: ZAux_Direct_Connect (controller link handle, 1, 0, 1)



5.4.2.2. Synchronous Motion 2

This routine is mainly an interpolation synthesis vector motion that drives axis 2 to follow axis 0 and axis 1, and perform synchronous motion with the interpolation synthesis vector.

```
// test1.cpp: define the entry point of control panel application program.
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
```

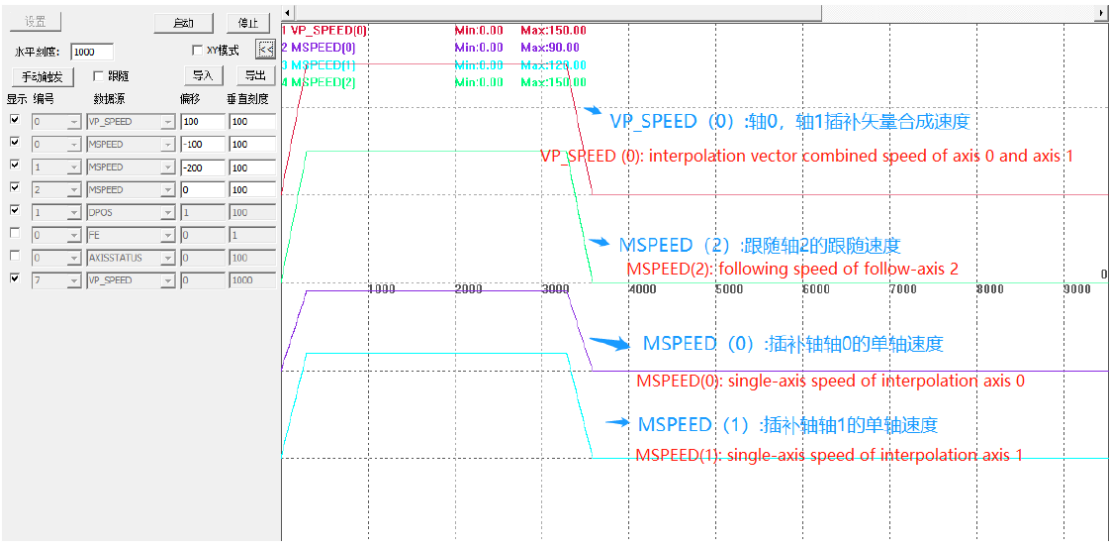
```
void commandCheckHandler(const char *command, int ret)
{
	if (ret)// it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";  // controller IP address, there may be errors when using simulator to check the curve, it is best to connect to controller for viewing.
	ZMC_HANDLE handle = NULL;		//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		getchar();
		return -1;
	}
	printf("Success to connect controller!\n");

	ret =ZAux_Direct_SetAtype(handle,0,1);// set axis type of axis 0 as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//judge whether the instruction is executed successfully
	ret =ZAux_Direct_SetAtype(handle,1,1);// set axis type of axis 1 as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//judge whether the instruction is executed successfully
	ret =ZAux_Direct_SetAtype(handle,2,1);// set axis type of axis 2 as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//judge whether the instruction is executed successfully

	ret =ZAux_Direct_SetUnits(handle,0,100);// set pulse amount of axis 0 as 100

	commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//judge whether the instruction is executed successfully
	ret =ZAux_Direct_SetUnits(handle,1,100);// set pulse amount of axis 1 as 100

	commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//judge whether the instruction is executed successfully
	ret =ZAux_Direct_SetUnits(handle,2,100);// set pulse amount of axis 2 as 100
	commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//judge whether the instruction is executed successfully
```

```
    ret =ZAux_Direct_SetAccel(handle,0,500);// set the acceleration of axis 0
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetAccel(handle,1,500);// set the acceleration of axis 1
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetAccel(handle,2,500);// set the acceleration of axis 2
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetDecel(handle,0,500);// set the deceleration of axis 0
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetDecel(handle,1,500);// set the deceleration of axis 1
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetDecel(handle,2,500);// set the deceleration of axis 2
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetDpos(handle,0,0);// set clearing DPOS of axis 0 (to 0)
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetDpos(handle,1,0);// set clearing DPOS of axis 1 (to 0)
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetDpos(handle,2,0);// set clearing DPOS of axis 2 (to 0)
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetSpeed(handle,0,150);// set the speed of axis 0
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetSpeed(handle,1,100);// set the speed of axis 1
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetSpeed(handle,2,100);// set the speed of axis 2
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
instruction is executed successfully

    ret =ZAux_Direct_Connpath(handle,1,0,2);// set open synchronous connection 2,
axis 2 connects the axis 0 (interpolation master axis)
    commandCheckHandler("ZAux_Direct_Connect", ret) ;//judge whether the
instruction is executed successfully

    float IDLE;
```

```
    ZAux_Trigger( handle);// trigger the oscilloscope
    int axislist[2]={0,1};
    float dposlist[2]={300,400};
    ret = ZAux_Direct_Move(handle, 2,axislist,dposlist); //interpolation motion,
300,400，
    commandCheckHandler("ZAux_Direct_Single_Move", ret) ;//judge whether the
instruction is executed successfully

    while (1)// wait for axis 0 motion to complete
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }
    ret =ZAux_Direct_Single_Cancel(handle,0,2);// cancel the synchronization motion
    commandCheckHandler("ZAux_Direct_Connect", ret) ;//judge whether the
instruction is executed successfully

    getchar();
    ret = ZAux_Close(handle);    // close the link
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

Waveform diagram of interpolation speed curve and following axis speed curve:

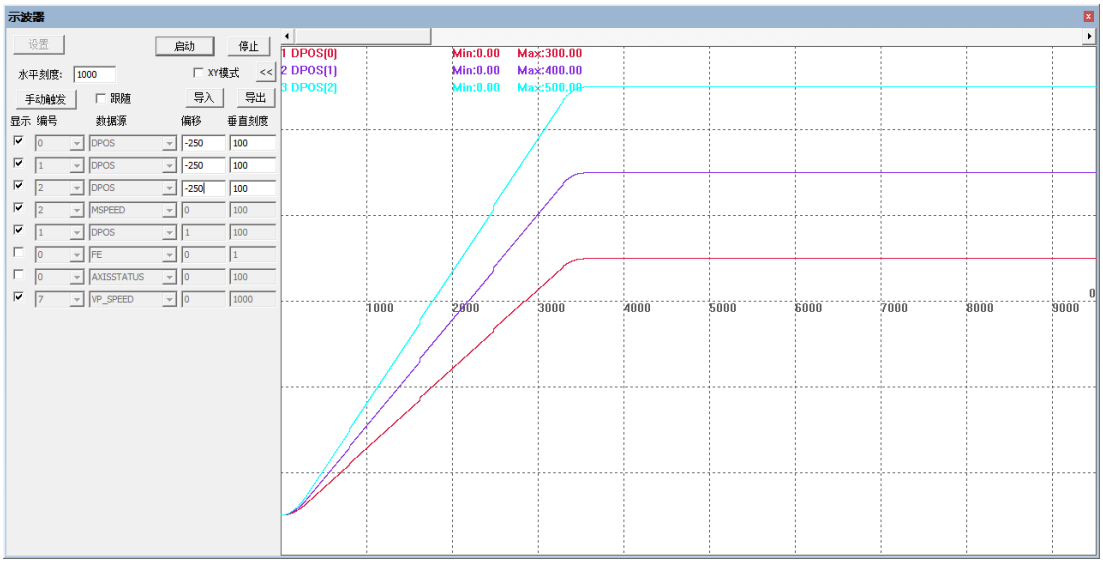
(It can be observed from the waveform diagram that the following axis changes with the speed change of the vector synthesized by the interpolation motion)



Waveform diagram of interpolation motion and following axis motion trajectory curve:

(It can be observed from the waveform diagram that the following axis moves with the trajectory of the vector synthesized by the interpolation motion.)

Synthetic vector distance=DPOS(2)=√(300^2+400^2)=500



5.4.2.3. Set Pulse Output Electronic Gear

In this routine, when the encoder gear ratio defaults to 1, modify the pulse output gear ratio, and connect the pulse output of axis 0 to the encoder input of axis 5, so as to observe the actual situation of the pulse output of axis 0.

Hardware wiring: It needs to connect the axis 0 pulse output to the axis 5 encoder.

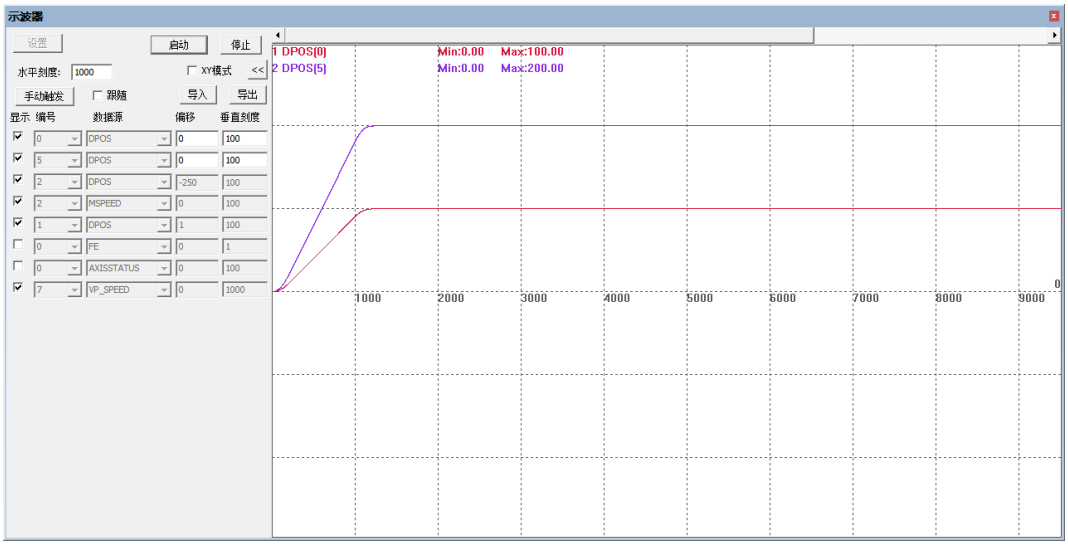
```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)// it is not 0, fail
    {
        printf("%s fail!return code is %d\n", command, ret);
```

```
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.181";  //controller IP address, this routine needs to use encoder for comparing, so it is necessary to connect to actual controller.
    ZMC_HANDLE handle = NULL;               //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("Success to connect controller!\n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//set axis type of axis 0 as 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//judge whether the instruction is executed successfully
    ret =ZAux_Direct_SetAtype(handle,5,6);//set axis type of axis 5 as pulse encoder
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//judge whether the instruction is executed successfully

    ret =ZAux_Direct_SetUnits(handle,0,100);//set pulse amount of axis 0 as 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;// judge whether the instruction is executed successfully
    ret =ZAux_Direct_SetUnits(handle,5,100);// set pulse amount of axis 5 as 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;// judge whether the instruction is executed successfully

    ret =ZAux_Direct_SetAccel(handle,0,500);//set acceleration of axis 0
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;// judge whether the instruction is executed successfully

    ret =ZAux_Direct_SetDecel(handle,0,500);//set deceleration of axis 0
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the instruction is executed successfully

    ret =ZAux_Direct_SetDpos(handle,0,0);//set clear axis 0 DPOS (to 0)
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the instruction is executed successfully
    ret =ZAux_Direct_SetMpos(handle,5,0);// set clear axis 5 MPOS (to 0)
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the instruction is executed successfully
```

```
    ret =ZAux_Direct_SetSpeed(handle,0,100);//set speed of axis 0
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully


    ret =ZAux_Direct_StepRatio(handle,0,2,1);//set pulse gear ratio of axis 0 as 2,
namely, 2 times pulse output
    commandCheckHandler("ZAux_Direct_StepRatio", ret) ;// judge whether the
instruction is executed successfully

    float IDLE;
    ZAux_Trigger( handle);// trigger the oscilloscope

    ret = ZAux_Direct_Single_Move(handle, 0,100); // axis 0 moves 100
    commandCheckHandler("ZAux_Direct_Single_Move", ret) ;// judge whether the
instruction is executed successfully

    while (1)// wait for axis 0 motion to complete
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

    ret =ZAux_Direct_StepRatio(handle,0,1,1);//resume the pulse gear ratio of axis 0
as 1
    commandCheckHandler("ZAux_Direct_StepRatio", ret) ;// judge whether the
instruction is executed successfully

    getchar();
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;// judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

Output waveform:

It can be observed from the waveform diagram that the pulse received by encoder axis 5 is twice that of axis 0, which is just in line with the set pulse output gear ratio.



5.4.2.4. Set Encoder Gear Ratio

In this routine, when the pulse output gear ratio defaults to 1, modify the encoder input gear ratio, and connect the pulse output of axis 0 to the encoder input of axis 5, so as to observe the actual situation of pulse reception of axis 5.

Routine hardware wiring: It is necessary to connect the axis 0 pulse output to the axis 5 encoder.

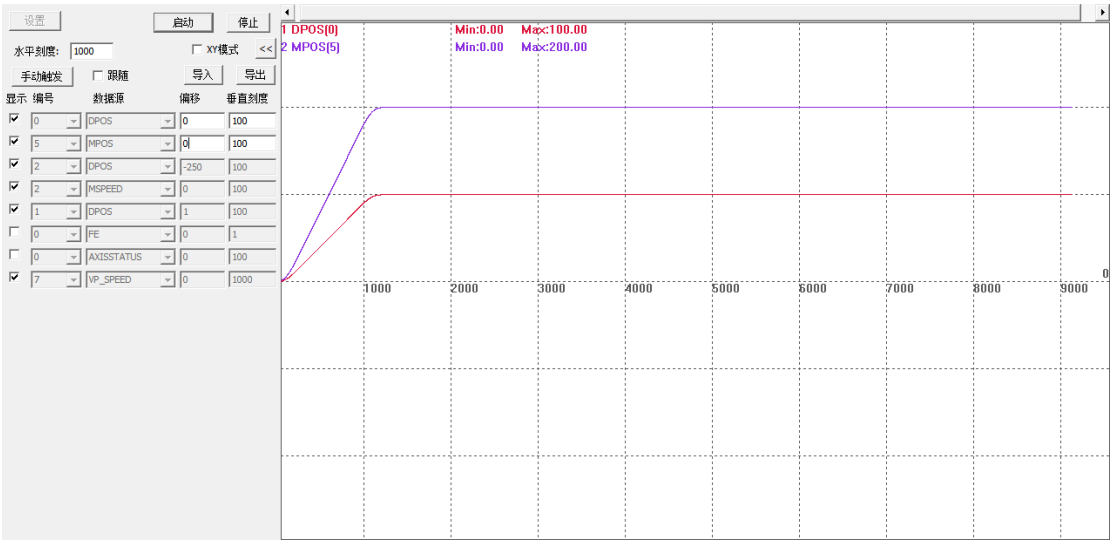
```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)// it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}

}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"192.168.0.181";  //controller IP address, this routine needs to use encoder for comparing, so it is necessary to connect to actual controller.
```

```
    ZMC_HANDLE handle = NULL;               //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("Success to connect controller!\n");

    ret =ZAux_Direct_SetAtype(handle,0,1);// set axis type of axis 0 as 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetAtype(handle,5,6);// set axis type of axis 5 as pulse encoder
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetUnits(handle,0,100);// set pulse amount of axis 0 as 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;// judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetUnits(handle,5,100);// set pulse amount of axis 5 as 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetAccel(handle,0,500);// set acceleration of axis 0
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetDecel(handle,0,500);// set deceleration of axis 0
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetDpos(handle,0,0);// set clear axis 0 DPOS (to 0)
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetMpos(handle,5,0);// set clear axis 5 MPOS (to 0)
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetSpeed(handle,0,100);// set speed of axis 0
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully


    ret =ZAux_Direct_EncoderRatio(handle,5,2,1);// set pulse gear ratio of axis 5 as 2,
namely, 2 times pulse output
```

```
    commandCheckHandler("ZAux_Direct_EncoderRatio", ret) ;// judge whether the instruction is executed successfully
    float IDLE;
    ZAux_Trigger( handle);//trigger the oscilloscope

    ret = ZAux_Direct_Single_Move(handle, 0,100); //axis 0 moves 100
    commandCheckHandler("ZAux_Direct_Single_Move", ret) ;// judge whether the instruction is executed successfully

    while (1)// wait for axis 0 motion to complete
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

    ret =ZAux_Direct_EncoderRatio(handle,5,1,1);// resume the pulse gear ratio of axis 5 as 1
    commandCheckHandler("ZAux_Direct_EncoderRatio", ret) ;// judge whether the instruction is executed successfully

    getchar();
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;// judge whether the instruction is executed successfully

    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

Output waveform:

It can be observed from the waveform diagram that the pulse received by encoder axis 5 is twice that of axis 0, which is just in line with the input gear ratio of the encoder.



5.5.Interpolation Motion

5.5.1.Emphasis

5.5.1.1. SP Speed

1. Difference between SP and non-SP motion

Set the axis speed through **ZAux_Direct_SetSpeed**, the speed changes immediately, and dynamic speed change can be realized, and all interpolation motions have corresponding SP motions with custom speeds, such as **ZAux_Direct_MoveSp**, **ZAux_Direct_MoveAbsSp** and so on. The speed of SP motion is no longer controlled by the global speed ZAux_Direct_SetSpeed, but the start, run, and end speeds of the corresponding segment are set through special speed commands, and these speeds also enter the motion buffer along with the motion.

2. SP speed description:

SP speed represents a function that can perform speed control. Zmotion control card commands with SP suffix mean that the motion can be controlled by a user-defined speed.

The function parameter setting in this chapter is mainly applied to the function after the interpolation motion command with the SP suffix. At this time, the motion speed adopts the set ZAux_Direct_SetForceSpeed parameter speed instead of the set

ZAux_Direct_SetSpeed speed parameter. SP speed applies to interpolation motion commands with SP suffix (eg ZAux_Direct_MoveSp, ZAux_Direct_MoveCircSp).

The starting speed set by ZAux_Direct_SetStartMoveSpeed: the starting speed of the SP motion of the custom speed.

The end speed set by ZAux_Direct_SetEndMoveSpeed: the end speed of SP movement with custom speed.

Forced speed set by ZAux_Direct_SetForceSpeed: forced speed of SP movement with custom speed.

Note: the above three parameters will take effect only when the motion command with SP is used, and the parameters are all brought into the motion buffer. When not in use, please set the command ZAux_Direct_SetStartMoveSpeed for setting the start speed and the command ZAux_Direct_SetEndMoveSpeed for setting the end speed to a larger value, otherwise this parameter will be used in the next movement command.

5.5.1.2. Continuous Interpolation & Trajectory Look-ahead

In the actual processing process, continuous interpolation is enabled in order to pursue machining efficiency. If the corners of the motion trajectory are not decelerated, when the corners are large, it will cause a large impact on the machine and affect the machining accuracy. If the continuous interpolation is turned off, the deceleration at the corner is reduced to 0. Although the machine is protected, the processing efficiency is greatly affected. Therefore, a look-ahead command is provided to automatically judge whether to reduce the corner speed to a reasonable value at the corner. In this way, it will not affect the machining accuracy and can improve the machining speed, which is the role of the trajectory look-ahead function.

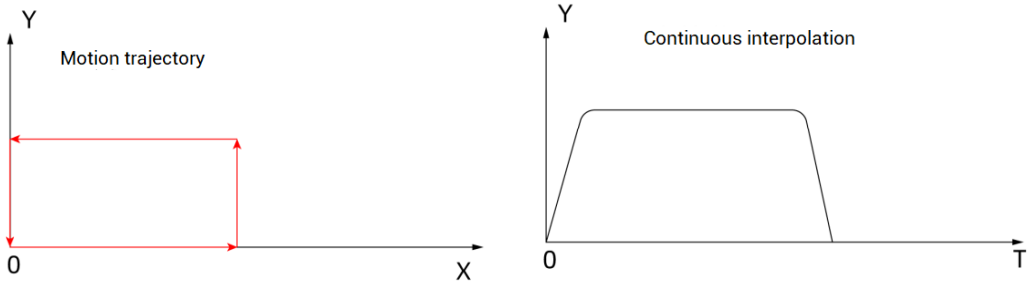
The trajectory look-ahead of the motion controller can automatically calculate a smooth speed plan according to the user's motion path, reducing the impact of the machine, thereby improving the machining accuracy. Automatically analyze the inflection point that will appear in the command trajectory of the motion buffer, and automatically calculate the motion speed at the corner according to the corner conditions set by the user, also calculate the speed plan based on the maximum acceleration value set by the user, so that speed values in any acceleration and deceleration process will not exceed the value of ZAux_Direct_SetAccel and

ZAux_Direct_SetDecel to prevent damage to the mechanical part.

Velocity planning with and without trajectory lookahead:

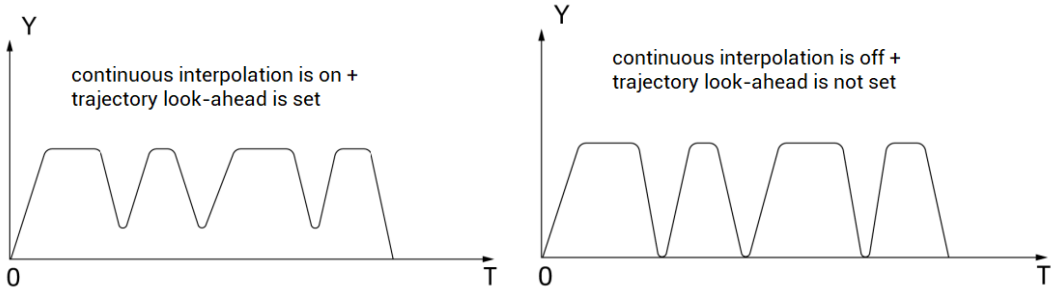
Assume that the motion trajectory is as shown in the left figure, following a rectangular trajectory, which is divided into four segments of linear interpolation motion.

Mode 1: after the continuous interpolation is turned on, the curve of the main axis speed changing with time is as shown in the right figure. The main axis speed is continuous, and the corner of the track still does not decelerate, and the impact at the corner is large when running at high speed.



Mode 2: under the condition of mode 1, if the continuous interpolation is turned off, the resulting curve of the main axis speed changing with time is shown in the left figure below. After moved each straight line, it decelerates to 0 and then starts the second straight line, which is not efficient.

Mode 3: under the condition of mode 1, continuous interpolation is turned on, and the trajectory look-ahead parameter is set, and the obtained curve of the main axis speed changing with time is shown on the right, and the corners are decelerated according to a certain ratio, and the processing efficiency is higher than that of mode 2.



If you want the speed curve to be softer in the above modes, you only need to set the speed to S curve through the **ZAux_Direct_SetSramp** command.

The main command **ZAux_Direct_SetCornerMode** of trajectory look-ahead is used for speed planning at corners. There are three commonly used modes, and different

modes can be selected according to the actual requirements of the processed trajectory.

This parameter takes effect before the interpolation motion command is called, and the corner mode is generally set in parameter initialization. Because the look-ahead motion parameters will be stored in the motion buffer together with the motion command, and the look-ahead motion parameters can be called multiple times, also different modes can also be used in combination. For example, **ZAux_Direct_SetCornerMode**=2+8 means that automatic corner deceleration and small circle speed limit are used at the same time, set the appropriate look-ahead mode according to the requirements of the trajectory segment, and automatically optimize the trajectory when executing the motion command.

Note: once the **ZAux_Direct_SetCornerMode** mode is set, the parameters will be stored in the controller. Set ZAux_Direct_SetCornerMode=0 to cancel it, otherwise the previously set ZAux_Direct_SetCornerMode will still take effect at the next run.

CORNER_MODE instruction parameter description:

Bit	Value	Description
0	1	Reserved
1	2	Automatic corner deceleration. Make acceleration and deceleration take effect According to **ZAux_Direct_SetAccel** and **ZAux_Direct_SetDecel**. This parameter takes effect before the motion function is called. The deceleration angle is set according to the **ZAux_Direct_SetDecelAngle** and **ZAux_Direct_SetStopAngle** functions. The reference speed of the deceleration corner is based on the FORCE_SPEED speed, and a reasonable **ZAux_Direct_SetForceSpeed** must be set.
2	4	Automatic small circle speed limit, speed limit when the radius is less than the set value, no speed limit when it is greater than the limit value. The modification of this parameter takes effect before the motion function is called. Limit speed is related to **ZAux_Direct_SetForceSpeed**. Limit speed = **ZAux_Direct_SetForceSpeed** * actual radius / **ZAux_Direct_SetFullSpRadius.** Speed limit radius is set through **ZAux_Direct_SetFullSpRadius**.
4	16	Reserved
5	32	Automatic chamfer settings. The modification of this parameter takes effect before the linear motion function is called. The setting value of the radius of the chamfer is not the actual

		value, only when it is 90 degrees. This linear motion is automatically chamfered with the previous linear motion, and the chamfer radius refers to **ZAux_Direct_SetZsmooth.**

ZAux_Direct_SetCornerMode=2 corner deceleration application: motion trajectory doesn't be changed, only automatically judge whether to decelerate at the corner, generally it is used to improve the problem of machine vibration, where there are requirements for trajectory accuracy and not so fast speed requirements.

ZAux_Direct_SetCornerMode=8 small circle speed limit application: do not change the motion track, generally used in arc processing, calculate the current limit speed of the arc according to the radius of the arc.

ZAux_Direct_SetCornerMode=32 automatic chamfering application: changing the motion trajectory will not reduce the speed. For the occasions with large trajectory corners, the motion trajectory at the chamfering will be automatically smoothed, so it is generally used in applications that require fast speed and not high accuracy. In order to facilitate the description of the function of each bit, the routines are set individually, and multiple bits can be used at the same time in practical applications. For example, **ZAux_Direct_SetCornerMode**=2+8, turn on automatic corner deceleration and small circle speed limit.

5.5.2.Routine

5.5.2.1. Corner Deceleration

After the corner deceleration is turned on, if the STARTMOVE_SPEED (SP command startinf speed ZAux_Direct_SetStartMoveSpeed) or ENDMOVE_SPEED (SP command ending speed ZAux_Direct_SetEndMoveSpeed) parameter of the custom speed (SP) motion command is set, the corner deceleration will be invalid, and the corner will follow the speed set by above two SP commands to move. In this example, the corner deceleration mode 2 and continuous interpolation are turned on, and the deceleration angle is set between 15-45 degrees, and there are three-segment linear interpolation trajectory.

```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
```

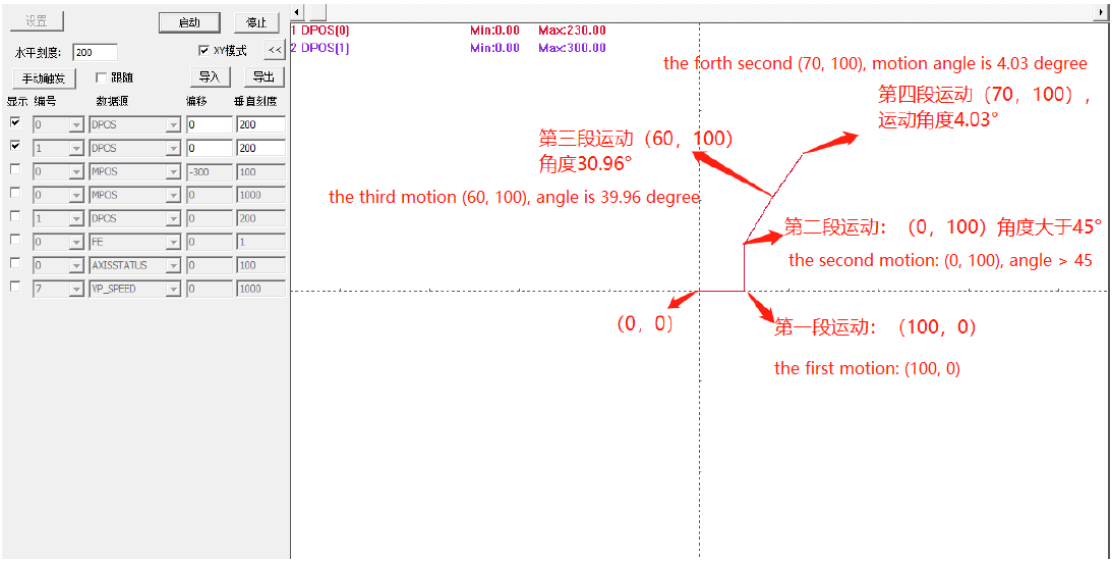
```
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#define PI 3.1415926

void commandCheckHandler(const char *command, int ret)
{
    if (ret)// it is not 0, fail
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.181";  //controller IP address, there may be error if checking the curve through the simulator, so it is better to connect to actual controller to check.
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);  //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("Success to connect controller!\n");

    ret =ZAux_Direct_SetAtype(handle,0,1);// set axis type of axis 0 as 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the instruction is executed successfully
    ret =ZAux_Direct_SetAtype(handle,1,1);// set axis type of axis 1 as 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the instruction is executed successfully

    ret =ZAux_Direct_SetUnits(handle,0,100);// set pulse amount of axis 0 as 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;// judge whether the instruction is executed successfully
    ret =ZAux_Direct_SetUnits(handle,1,100);// set pulse amount of axis 1 as 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;// judge whether the instruction is executed successfully

    ret =ZAux_Direct_SetAccel(handle,0,500);// set acceleration of axis 0
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;// judge whether the instruction is executed successfully
```

```
    ret =ZAux_Direct_SetAccel(handle,1,500);// set acceleration of axis 1
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetDecel(handle,0,500);// set deceleration of axis 0
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetDecel(handle,1,500);// set deceleration of axis 1
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetDpos(handle,0,0);// set clear axis 0 DPOS (to 0)
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetDpos(handle,1,0);// set clear axis 1 DPOS (to 0)
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetSpeed(handle,0,100);// set speed of axis 0
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetSpeed(handle,1,100);// set speed of axis 1
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetMerge(handle,0,1);//set open continuous interpolation (only
main axis is opened, for example, axis 0 and axis 1 interpolate, axis 0 is the main axis.
The main axis is determined by the first axis No. in continuous interpolation motion
command list)
    commandCheckHandler("ZAux_Direct_SetMerge", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetCornerMode(handle,0,2);// set open corner mode (only main
axis is opened, for example, axis 0 and axis 1 interpolate, axis 0 is the main axis. The
main axis is determined by the first axis No. in continuous interpolation motion
command list)
    commandCheckHandler("ZAux_Direct_SetCornerMode", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetDecelAngle(handle,0,15 * (PI/180));//set starting
deceleration angle (only main axis is opened, for example, axis 0 and axis 1
interpolate, axis 0 is the main axis. The main axis is determined by the first axis No. in
continuous interpolation motion command list)
    commandCheckHandler("ZAux_Direct_SetDecelAngle", ret) ;// judge whether the
instruction is executed successfully
```

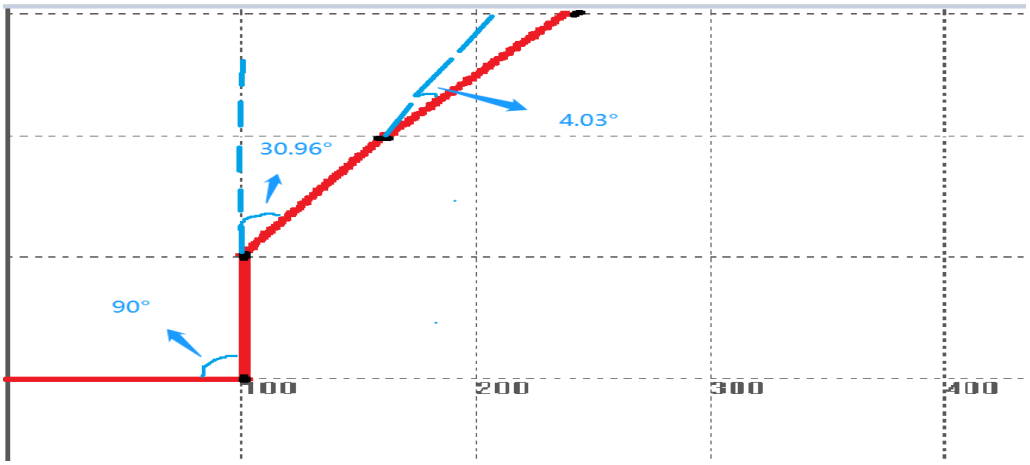
```
    ret =ZAux_Direct_SetStopAngle(handle,0,45 * (PI/180));//set end deceleration angle (only main axis is opened, for example, axis 0 and axis 1 interpolate, axis 0 is the main axis. The main axis is determined by the first axis No. in continuous interpolation motion command list)
    commandCheckHandler("ZAux_Direct_SetStopAngle", ret) ;// judge whether the instruction is executed successfully

    ret =ZAux_Direct_SetForceSpeed(handle,0,100);//set the speed when in equal deceleration (only main axis is opened, for example, axis 0 and axis 1 interpolate, axis 0 is the main axis. The main axis is determined by the first axis No. in continuous interpolation motion command list)
    commandCheckHandler("ZAux_Direct_SetForceSpeed", ret) ;// judge whether the instruction is executed successfully

    float IDLE;
    float DposList[2]={0};
    int Axislist[2]={0,1};

    ZAux_Trigger( handle);// trigger the oscilloscope
    DposList[0]=100;
    DposList[1]=0;
    ret =ZAux_Direct_Move(handle,2,Axislist,DposList);//interpolation (100,0)
    commandCheckHandler("ZAux_Direct_SetForceSpeed", ret) ;// judge whether the instruction is executed successfully
    DposList[0]=0;
    DposList[1]=100;
    ret =ZAux_Direct_Move(handle,2,Axislist,DposList);//interpolation (0,100), motion angle is more than 45°, full deceleration
    commandCheckHandler("ZAux_Direct_SetForceSpeed", ret) ;// judge whether the instruction is executed successfully
    DposList[0]=60;
    DposList[1]=100;
    ret =ZAux_Direct_Move(handle,2,Axislist,DposList);//interpolation (60, 100), motion angle 30.96° is 15°~45°, full deceleration
    commandCheckHandler("ZAux_Direct_SetForceSpeed", ret) ;// judge whether the instruction is executed successfully
    DposList[0]=70;
    DposList[1]=100;
    ret =ZAux_Direct_Move(handle,2,Axislist,DposList);//interpolation (70, 100), motion angle 4.03° is 15°, not to decelerate
    commandCheckHandler("ZAux_Direct_SetForceSpeed", ret) ;// judge whether the instruction is executed successfully

    while (1)// wait for axis 0 motion to complete
    {
```

```
            Sleep(100);
            ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
            if (IDLE<0)break;
        }

        getchar();
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret);    //judge whether the instruction is executed successfully
        printf("connection closed!\n");
        handle = NULL;
        return 0;
}
```

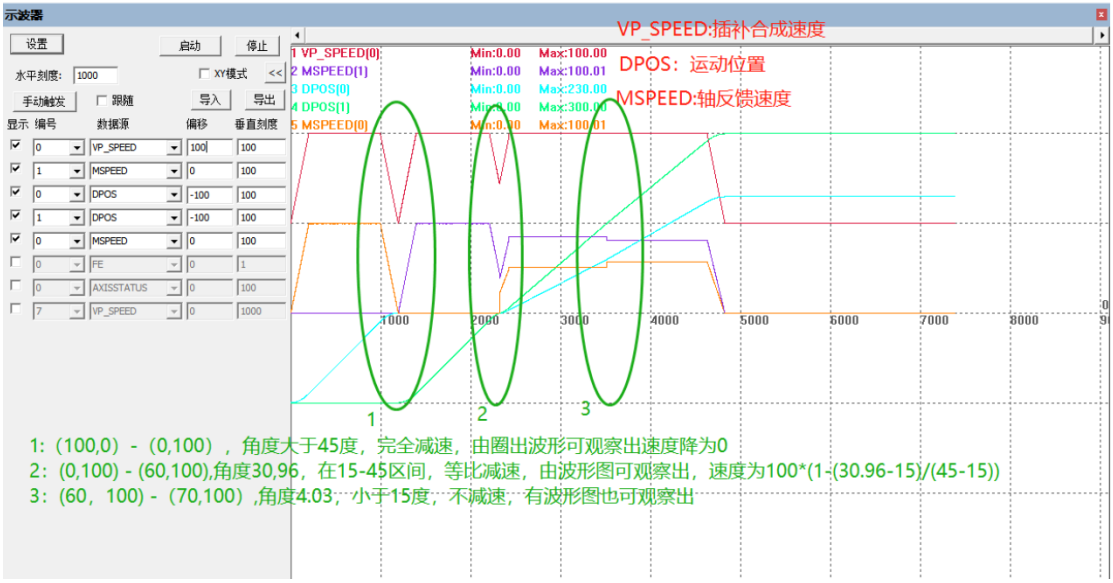
- Trajectory Curve:



- Angle analysis:



- Speed Curve:



- VP_SPEED: interpolation composite speed
- DPOS: motion position
- MSPEED: axis feedback speed
- 1. (100,0) – (0,100), angle is more than 45 degrees, full decelerate, it can be seen the speed is decelerated to 0 from the marked waveform (green, 1)
- 2. (0,100) – (70,100), angle is 90.96 degrees, which is between 15 to 45, proportionally decelerate, it can be seen the speed is 100*(1-(30.96-15)/(45-15)) from marked green 2 waveform.
- 3. (60,100) – (70,100), angle is 4.03 degrees, which is less than 15 degrees, not to decelerate, please check marked green 3 waveform

5.5.2.2. Draw A Circle with The Plane Center & Small-Circle Speed Limit

The small circle is limited to 60. When the radius is greater than or equal to the limit value, the speed is not limited, and run at the speed of SPEED, at this time, the speed is 100.

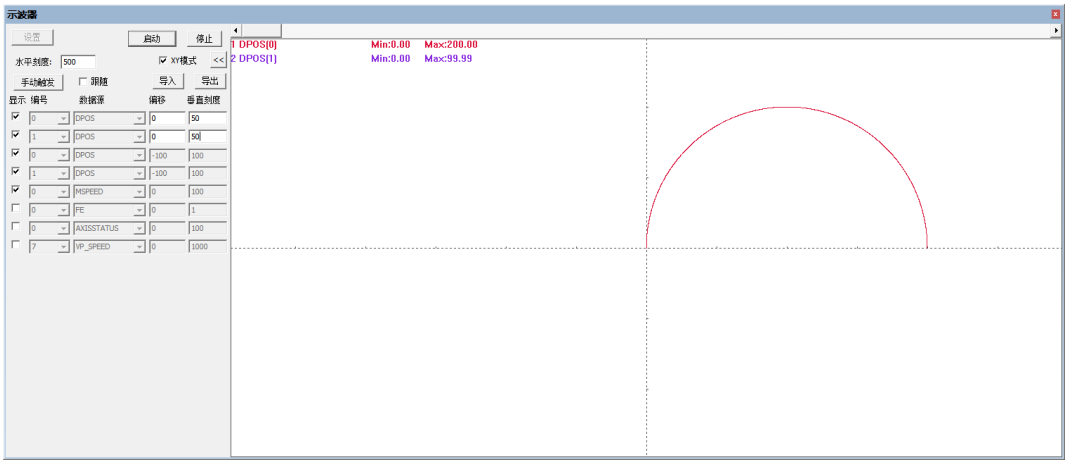
```
// test1.cpp: define the entry point of control panel application program
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#define PI 3.1415926
```

```
void commandCheckHandler(const char *command, int ret)
{
    if (ret)// it is not 0, fail
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.181";  //controller IP address, there may be error if checking the curve through the simulator, so it is better to connect to actual controller to check.
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);  //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("Success to connect controller!\n");

    ret =ZAux_Direct_SetAtype(handle,0,1);// set axis type of axis 0 as 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//judge whether the instruction is executed successfully
    ret =ZAux_Direct_SetAtype(handle,1,1);// set axis type of axis 1 as 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//judge whether the instruction is executed successfully

    ret =ZAux_Direct_SetUnits(handle,0,100);// set pulse amount of axis 0 as 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//judge whether the instruction is executed successfully
    ret =ZAux_Direct_SetUnits(handle,1,100);// set pulse amount of axis 1 as 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;//judge whether the instruction is executed successfully

    ret =ZAux_Direct_SetAccel(handle,0,500);// set acceleration of axis 0
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//judge whether the instruction is executed successfully
    ret =ZAux_Direct_SetAccel(handle,1,500);// set acceleration of axis 1
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//judge whether the instruction is executed successfully

    ret =ZAux_Direct_SetDecel(handle,0,500);// set deceleration of axis 0
```

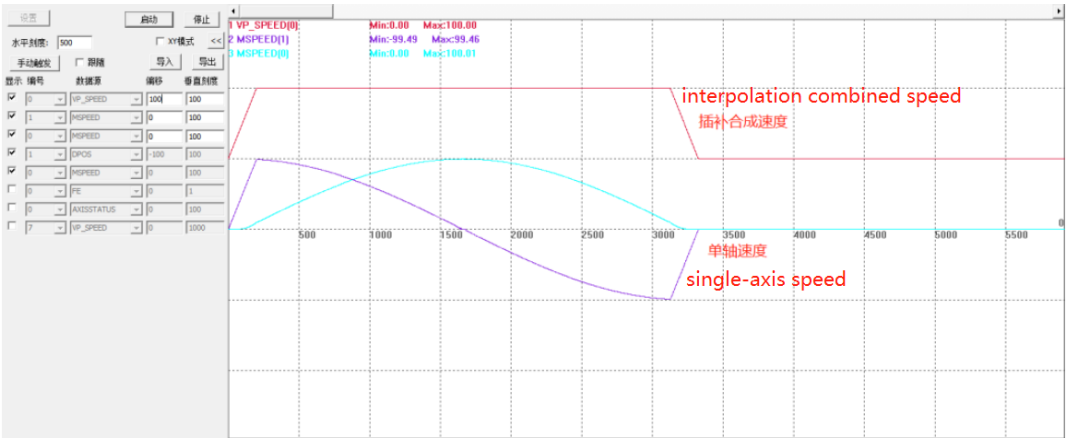
```
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetDecel(handle,1,500);// set deceleration of axis 1
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetDpos(handle,0,0);// set clear axis 0 DPOS (to 0)
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetDpos(handle,1,0);// set clear axis 1 DPOS (to 0)
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetSpeed(handle,0,100);// set speed of axis 0
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetSpeed(handle,1,100);// set speed of axis 1
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetMerge(handle,0,1);// set open continuous interpolation (only
main axis is opened, for example, axis 0 and axis 1 interpolate, axis 0 is the main axis.
The main axis is determined by the first axis No. in continuous interpolation motion
command list)
    commandCheckHandler("ZAux_Direct_SetMerge", ret) ;//judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetCornerMode(handle,0,8);//set open small circle speed limit
(only main axis is opened, for example, axis 0 and axis 1 interpolate, axis 0 is the
main axis. The main axis is determined by the first axis No. in continuous
interpolation motion command list)
    commandCheckHandler("ZAux_Direct_SetCornerMode", ret) ;//judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetFullSpRadius(handle,0,60);// set starting deceleration angle
(only main axis is opened, for example, axis 0 and axis 1 interpolate, axis 0 is the
main axis. The main axis is determined by the first axis No. in continuous
interpolation motion command list)
    commandCheckHandler("ZAux_Direct_SetFullSpRadius", ret) ;//judge whether
the instruction is executed successfully

    ret =ZAux_Direct_SetForceSpeed(handle,0,100);//set the speed when in equal
deceleration (only main axis is opened, for example, axis 0 and axis 1 interpolate, axis
0 is the main axis. The main axis is determined by the first axis No. in continuous
interpolation motion command list)
```

```
	commandCheckHandler("ZAux_Direct_SetForceSpeed", ret) ;//judge whether the instruction is executed successfully

	float IDLE;
	int Axislist[2]={0,1};

	ZAux_Trigger( handle);//trigger the oscilloscope
	//the coordinate is 0,0 before the motion
	//interpolation trajectory end point coordinate is 100,0
	//circle center coordinate is 50,0，
	//the radius is 50 from above
	ret =ZAux_Direct_MoveCirc(handle,2,Axislist,100,0,50,0,1);//circular interpolation motion
	commandCheckHandler("ZAux_Direct_MoveCirc", ret) ;//judge whether the instruction is executed successfully

	while (1)//wait for axis 0 motion to complete
	{
		Sleep(100);
		ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
		if (IDLE<0)break;
	}

	getchar();
	ret = ZAux_Close(handle);    //close the connection
	commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is executed successfully
	printf("connection closed!\n");
	handle = NULL;
	return 0;
}
```

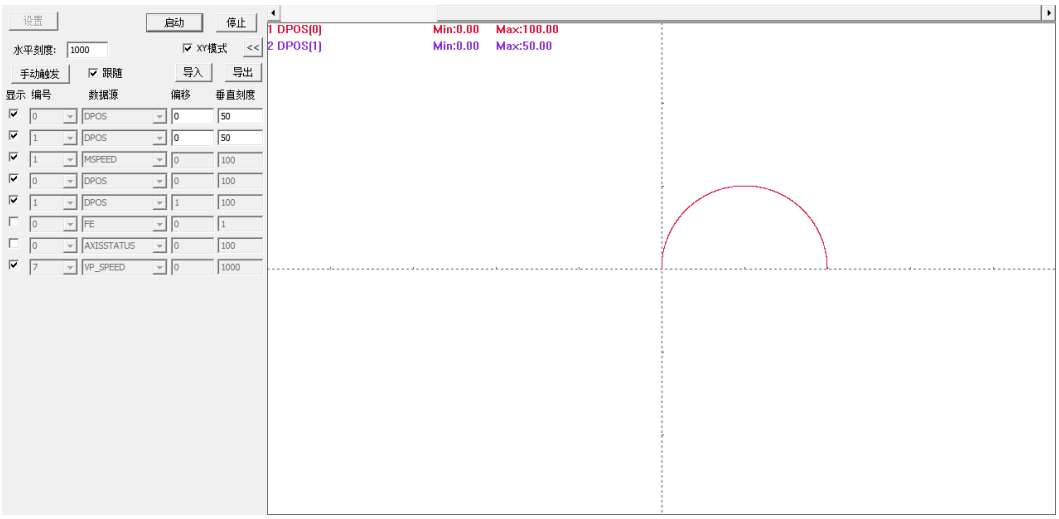
- The radius is 100, 100>60, trajectory waveform:



- Set the radius as 100, 100>60, speed curve waveform (not to decelerate totally):



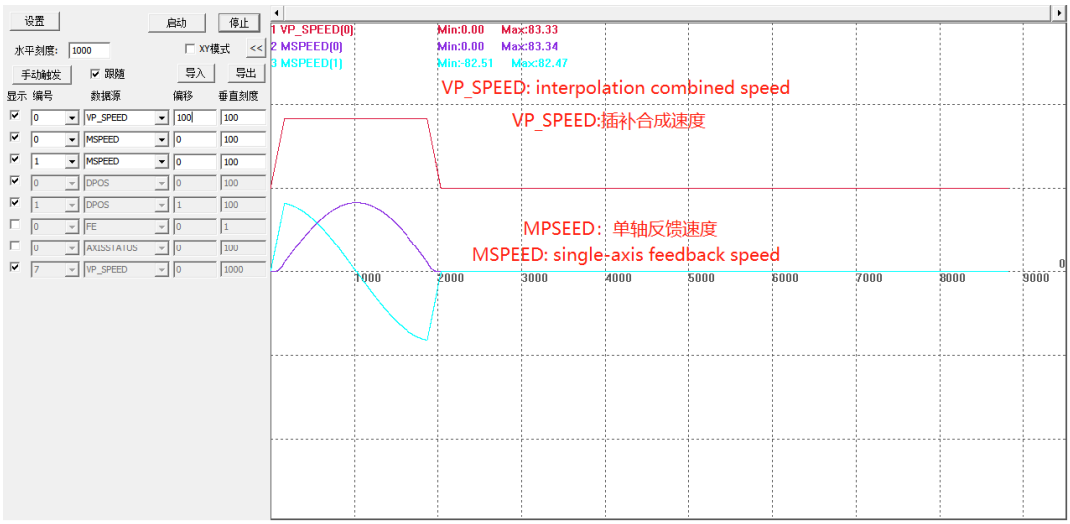
- The radius is 50, 50<60, trajectory waveform:



- The radius is 50, 50<60, speed curve waveform (small circle deceleration)

  //when radius is equal to limit value, limit = FORCE_SPEED * actual radius / FULL_SP_RADIUS

  //Speed = 100*50*60=83.33



5.5.2.3. Chamfer

Chamfering can be used for motions such as straight lines, arcs, and spirals. For intuitive display, only straight line corners are made in the routine. Automatically calculate the actual corner radius size based on the corner angle. Limits to 50% when path is exceeded. (Example: move two placements (100,0), (0,100), the maximum chamfer radius is 50)

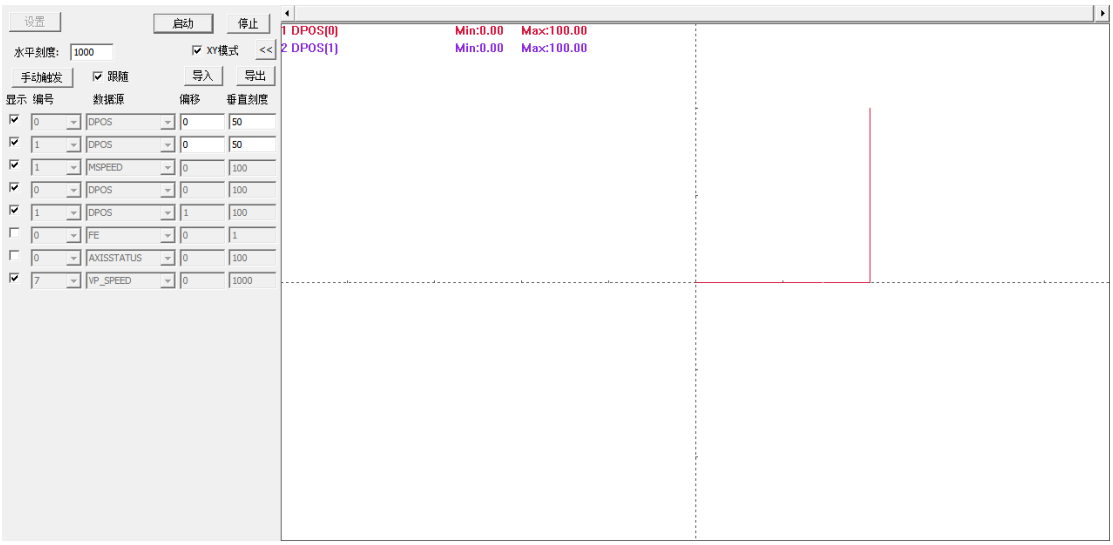
```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#define PI 3.1415926

void commandCheckHandler(const char *command, int ret)
{
    if (ret)// it is not 0, fail
    {
        printf("%s fail!return code is %d\n", command, ret);
    }

}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.181";  //controller IP address, there may be error if checking the curve through the simulator, so it is better to connect to actual
```

```
controller to check.
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("Success to connect controller!\n");

    ret =ZAux_Direct_SetAtype(handle,0,1);// set axis type of axis 0 as 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetAtype(handle,1,1);// set axis type of axis 1 as 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetUnits(handle,0,100);// set pulse amount of axis 0 as 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;// judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetUnits(handle,1,100);// set pulse amount of axis 1 as 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetAccel(handle,0,500);// set acceleration of axis 0
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;// judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetAccel(handle,1,500);// set acceleration of axis 1
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetDecel(handle,0,500);// set deceleration of axis 0
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetDecel(handle,1,500);// set deceleration of axis 1
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetDpos(handle,0,0);// set clear axis 0 DPOS (to 0)
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetDpos(handle,1,0);// set clear axis 1 DPOS (to 0)
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the
instruction is executed successfully
```

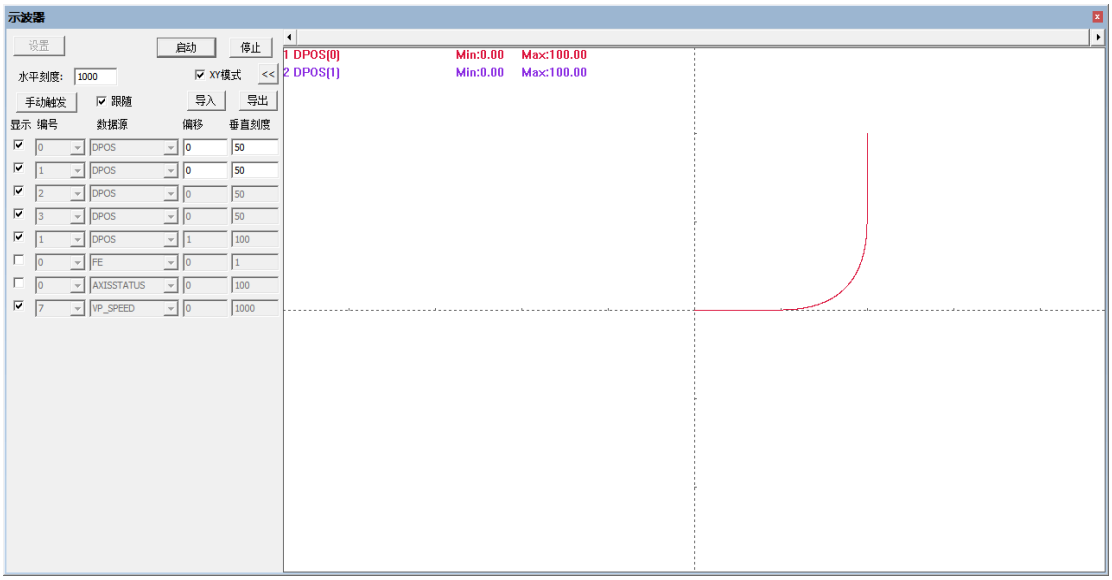
```
    ret =ZAux_Direct_SetSpeed(handle,0,100);// set speed of axis 0
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetSpeed(handle,1,100);// set speed of axis 1
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetMerge(handle,0,1);//set open continuous interpolation (only
main axis is opened, for example, axis 0 and axis 1 interpolate, axis 0 is the main axis.
The main axis is determined by the first axis No. in continuous interpolation motion
command list)
    commandCheckHandler("ZAux_Direct_SetMerge", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetCornerMode(handle,0,32);//set automatic chamfer (only
main axis is opened, for example, axis 0 and axis 1 interpolate, axis 0 is the main axis.
The main axis is determined by the first axis No. in continuous interpolation motion
command list)
    commandCheckHandler("ZAux_Direct_SetCornerMode", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetZsmooth(handle,0,50);//set chamfer radius (only main axis
is opened, for example, axis 0 and axis 1 interpolate, axis 0 is the main axis. The main
axis is determined by the first axis No. in continuous interpolation motion command
list)

    commandCheckHandler("ZAux_Direct_SetZsmooth", ret) ;// judge whether the
instruction is executed successfully

    float IDLE;
    int Axislist[2]={0,1};
    float Dposlist[2]={0};
    ZAux_Trigger( handle);// trigger the oscilloscope

    Dposlist[0]=100;
    Dposlist[1]=0;
    ret =ZAux_Direct_Move(handle,2,Axislist,Dposlist);//linear interpolation
    commandCheckHandler("ZAux_Direct_MoveCirc", ret) ;// judge whether the
instruction is executed successfully
    Dposlist[0]=0;
    Dposlist[1]=100;
    ret =ZAux_Direct_Move(handle,2,Axislist,Dposlist);//linear interpolation
    commandCheckHandler("ZAux_Direct_MoveCirc", ret) ;// judge whether the
instruction is executed successfully
```

```
    while (1)// wait for axis 0 motion to complete
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

    getchar();
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;// judge whether the instruction is executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

➢ Interpolation trajectory, before using chamfer, trajectory waveform:



➢ Interpolation trajectory, after using chamfer, chamfer radius is set as the max value 50 of this trajectory, trajectory waveform:



5.5.2.4. Axis Holding Input Application

When the axis 0 moves a SP point of 1500, when the signal of IN (0) is encountered during the movement, the speed is automatically switched to the holding speed.

```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;				//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
```

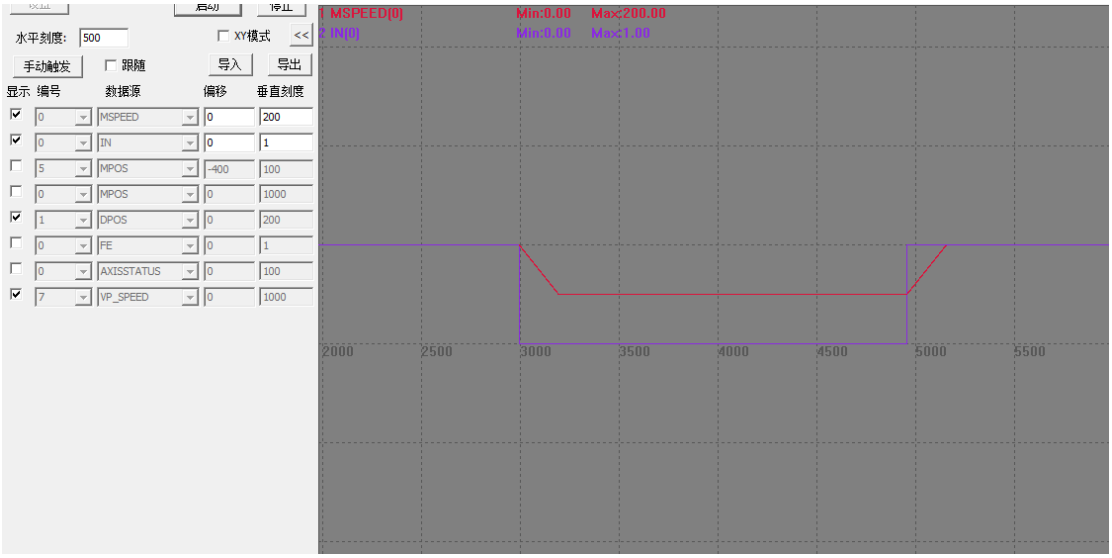
```
            printf("Fail to connect controller!\n");
            handle = NULL;
            getchar();
            return -1;
        }
        printf("Success to connect controller!\n");

        ret =ZAux_Direct_SetAtype(handle,0,1);//set axis type as 1
        commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully
        ret =ZAux_Direct_SetAccel(handle,0,500);//set axis acceleration
        commandCheckHandler("ZAux_Direct_SetAccel", ret) ;// judge whether the
instruction is executed successfully
        ret =ZAux_Direct_SetDecel(handle,0,500);//set axis deceleration
        commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully
        ret =ZAux_Direct_SetDpos(handle,0,0);//set clear DPOS (to 0)
        commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the
instruction is executed successfully

        ret =ZAux_Direct_SetForceSpeed(handle,0,200);//set axis SP running speed
        commandCheckHandler("ZAux_Direct_SetSpeed", ret) ;// judge whether the
instruction is executed successfully

        ret =ZAux_Direct_SetFhspeed(handle,0,100);//set holding speed as 100
        commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the
instruction is executed successfully

        ret =ZAux_Direct_SetFholdIn(handle,0,0);//set holding speed mapping IN as 0
        commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the
instruction is executed successfully

        ret =ZAux_Direct_SetInvertIn(handle,0,1);//set holding speed mapping IN to
reverse electrical level
        commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the
instruction is executed successfully

        float IDLE;
        int axislist=0;
        float dposlist=1500;
        ZAux_Trigger( handle);//trigger the oscilloscope
        ZAux_Direct_MoveSp(handle,1,&axislist,&dposlist);//axis 0 moves 1500
        while (1)//wait for axis 0 to complete the motion
        {
            Sleep(100);
            ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
```

```
            if (IDLE<0)break;
        }

        getchar();
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;// judge whether the instruction is executed successfully
        printf("connection closed!\n");
        handle = NULL;
        return 0;

}
```

➢ Speed Curve:

MSPEED (encoder feedback real-time speed): vertical scale 200

IN (0): vertical scale 1



5.5.2.5. Continuous Interpolation SP Speed Motion Control

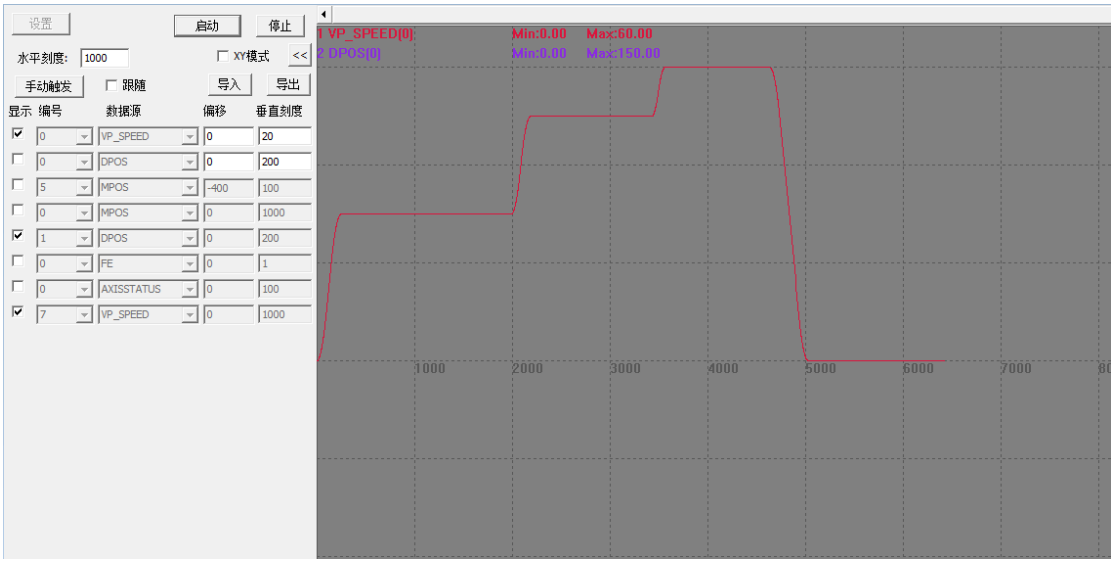
```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
```

```
void commandCheckHandler(const char *command, int ret)
{
	if (ret)// it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;		//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		getchar();
		return -1;
	}
	printf("Success to connect controller!\n");

	ret =ZAux_Direct_SetAtype(handle,0,1);//set axis type as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully
	ret =ZAux_Direct_SetAccel(handle,0,500);//set axis acceleration
	commandCheckHandler("ZAux_Direct_SetAccel", ret) ;// judge whether the
instruction is executed successfully
	ret =ZAux_Direct_SetDecel(handle,0,500);//set axis deceleration
	commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully
	ret =ZAux_Direct_SetDpos(handle,0,0);//set clear DPOS to 0
	commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the
instruction is executed successfully

	ret =ZAux_Direct_SetMerge(handle, 0,1);//set continuous interpolation
	commandCheckHandler("ZAux_Direct_SetMerge", ret) ;// judge whether the
instruction is executed successfully
	float IDLE;

	int axislist=0;
	float dposlist=40;
	ZAux_Trigger( handle);//trigger the oscilloscope
	//the first segment
	ret =ZAux_Direct_SetForceSpeed(handle,0,30);//set axis SP running speed
	commandCheckHandler("ZAux_Direct_SetSpeed", ret) ;// judge whether the
```

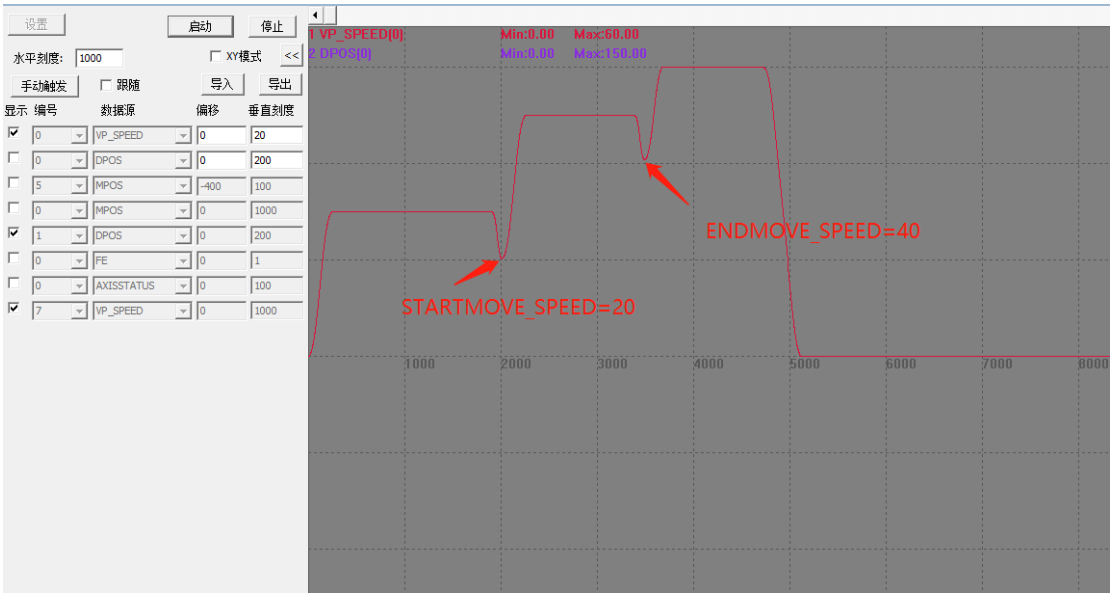
```
instruction is executed successfully

    ret =ZAux_Direct_SetStartMoveSpeed(handle,0,1000);//cancel setting axis
starting running speed, the set value should be more than ForceSpeed.
    commandCheckHandler("ZAux_Direct_SetStartMoveSpeed", ret) ;// judge
whether the instruction is executed successfully

    ret =ZAux_Direct_SetEndMoveSpeed(handle,0,1000);// cancel setting axis end
running speed, the set value should be more than ForceSpeed.
    commandCheckHandler("ZAux_Direct_SetEndMoveSpeed", ret) ;// judge whether
the instruction is executed successfully

    ZAux_Direct_MoveSp(handle,1,&axislist,&dposlist);//axis 0 moves 40
    //the second segment

    ret =ZAux_Direct_SetForceSpeed(handle,0,50);//set axis SP running speed
    commandCheckHandler("ZAux_Direct_SetSpeed", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetStartMoveSpeed(handle,0,20);//set starting running speed
as 20
    commandCheckHandler("ZAux_Direct_SetStartMoveSpeed", ret) ;// judge
whether the instruction is executed successfully

    ret =ZAux_Direct_SetEndMoveSpeed(handle,0,40);//set axis end running speed
as 40
    commandCheckHandler("ZAux_Direct_SetEndMoveSpeed", ret) ;// judge whether
the instruction is executed successfully

    dposlist=50;
    ZAux_Direct_MoveSp(handle,1,&axislist,&dposlist);//axis 0 moves 50
    //the third segment
    ret =ZAux_Direct_SetForceSpeed(handle,0,60);//set axis SP running speed
    commandCheckHandler("ZAux_Direct_SetSpeed", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetStartMoveSpeed(handle,0,1000);// cancel setting axis
starting running speed, the set value should be more than ForceSpeed.
    commandCheckHandler("ZAux_Direct_SetStartMoveSpeed", ret) ;// judge
whether the instruction is executed successfully

    ret =ZAux_Direct_SetEndMoveSpeed(handle,0,1000);// cancel setting axis end
running speed, the set value should be more than ForceSpeed.
    commandCheckHandler("ZAux_Direct_SetEndMoveSpeed", ret) ;// judge whether
the instruction is executed successfully
```

```
    dposlist=60;
    ZAux_Direct_MoveSp(handle,1,&axislist,&dposlist);//axis 0 moves 60

    getchar();
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;// judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

➢ When SP speed is not used:



➢ When SP speed is used:



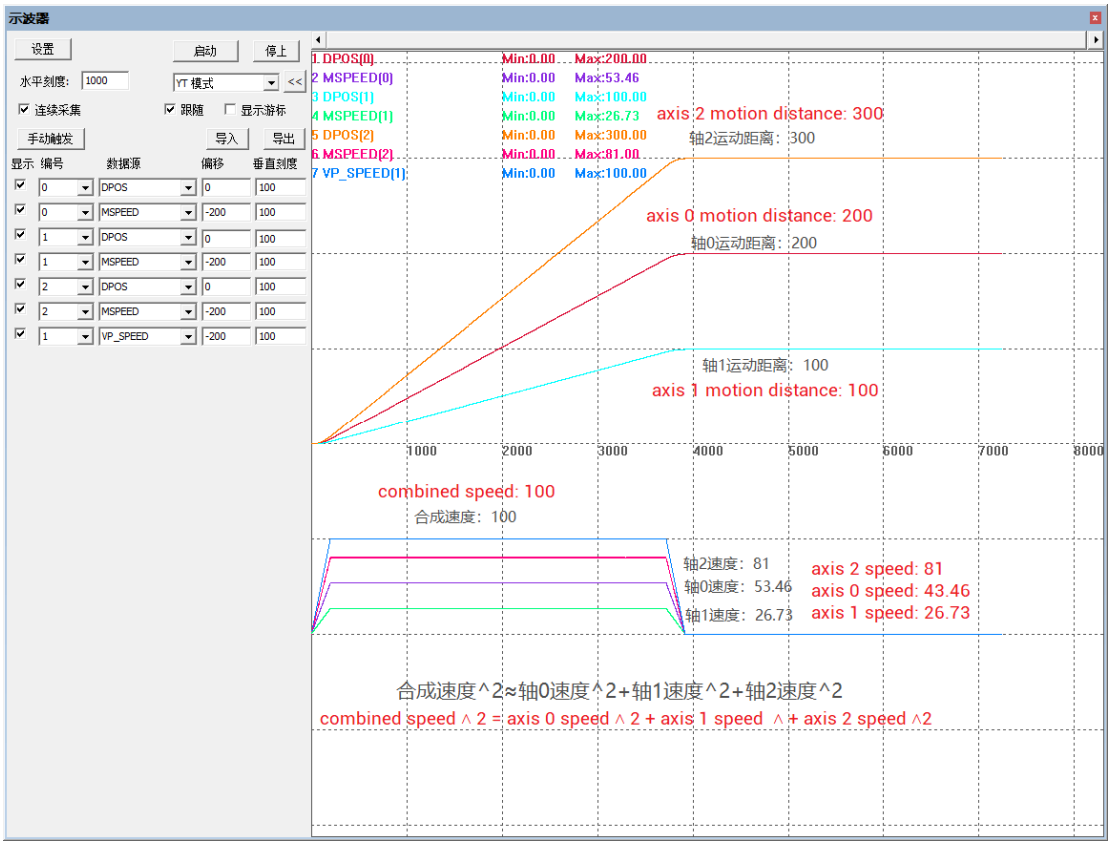
5.5.2.6. Multi-axis Linear Interpolation

```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)// it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//simulator IP address
	ZMC_HANDLE handle = NULL;			//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to connect controller!\n");

	ret =ZAux_Direct_SetAtype(handle,0,1);//set axis type of axis 0 as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully

	ret =ZAux_Direct_SetAtype(handle,1,1);// set axis type of axis 1 as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully

	ret =ZAux_Direct_SetAtype(handle,2,1);// set axis type of axis 2 as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully

	ret = ZAux_Direct_Single_Cancel(handle, 0, 0); //axis 0 motion stops
```

```
        commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
        ret = ZAux_Direct_Single_Cancel(handle, 1, 0); // axis 1 motion stops
        commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
        ret = ZAux_Direct_Single_Cancel(handle, 2, 0); // axis 2 motion stops
        commandCheckHandler("ZAux_Direct_Single_Cancel", ret);

        ret = ZAux_Direct_SetDpos(handle, 0, 0);       //set axis 0 dpos
        commandCheckHandler("ZAux_Direct_SetDpos", ret);
        ret = ZAux_Direct_SetDpos(handle, 1, 0);       // set axis 1 dpos
        commandCheckHandler("ZAux_Direct_SetDpos", ret);
        ret = ZAux_Direct_SetDpos(handle, 2, 0);       // set axis 2 dpos
        commandCheckHandler("ZAux_Direct_SetDpos", ret);

        ret =ZAux_Direct_SetMerge(handle, 0,1);//set continuous interpolation
        commandCheckHandler("ZAux_Direct_SetMerge", ret) ;// judge whether the
instruction is executed successfully


        int axislist[3] = { 1,0,2 };                  //motion axis list, axis 1 is the master axis
        float poslist[3] = { 100,200,300 };//motion list, axis 1-100, axis 0-200, axis 2-300
        ZAux_Trigger( handle);//trigger the oscilloscope

        ret = ZAux_Direct_SetSpeed(handle, axislist[0], 100);      //set interpolation speed
100, the setting is on the master axis
        commandCheckHandler("ZAux_Direct_SetSpeed", ret);

        ret = ZAux_Direct_SetAccel(handle, axislist[0], 500);       // set interpolation
acceleration 500, the setting is on the master axis
        commandCheckHandler("ZAux_Direct_SetAccel", ret);

        ret = ZAux_Direct_SetDecel(handle, axislist[0], 500);       // set interpolation
deceleration 500, the setting is on the master axis
        commandCheckHandler("ZAux_Direct_SetDecel", ret);

        ret = ZAux_Direct_MoveAbs(handle, 3, axislist, poslist);      //call the motion,
axis0, 1, 2 move to corresponding absolute position
        commandCheckHandler("ZAux_Direct_MoveAbs", ret);

        int state[3];

        do
        {
            ZAux_Direct_GetIfIdle(handle, axislist[0], state);
            ZAux_Direct_GetIfIdle(handle, axislist[1], state + 1);
            ZAux_Direct_GetIfIdle(handle, axislist[2], state + 2);
```

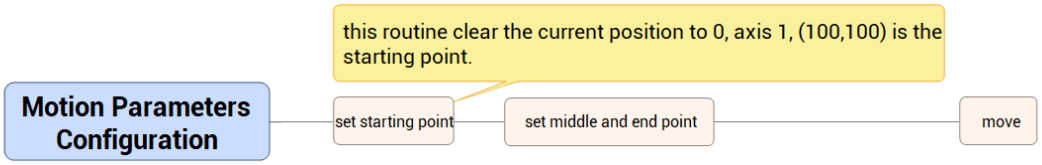
```
            if (!(state[0] || state[1] || state[2]))
            {
                Sleep(100);
            }
            else
            {
                break;
            }
        } while (true);//wait for the motion to stop

        printf("Idle\n");


        Sleep(2000);
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;// judge whether the instruction is executed successfully
        printf("connection closed!\n");
        handle = NULL;
        return 0;
}
```

The figure shows the movement speed changes of axes 0, 1, and 2, and the meaning of the line segment is shown in the figure.



5.5.2.7. Draw the Arc with 3-Point in Plane

This routine uses 2 axes to do planar circular arc movement, starting point (100, 100), end point (300, 100), middle point (200, 200), and three points to draw an arc.



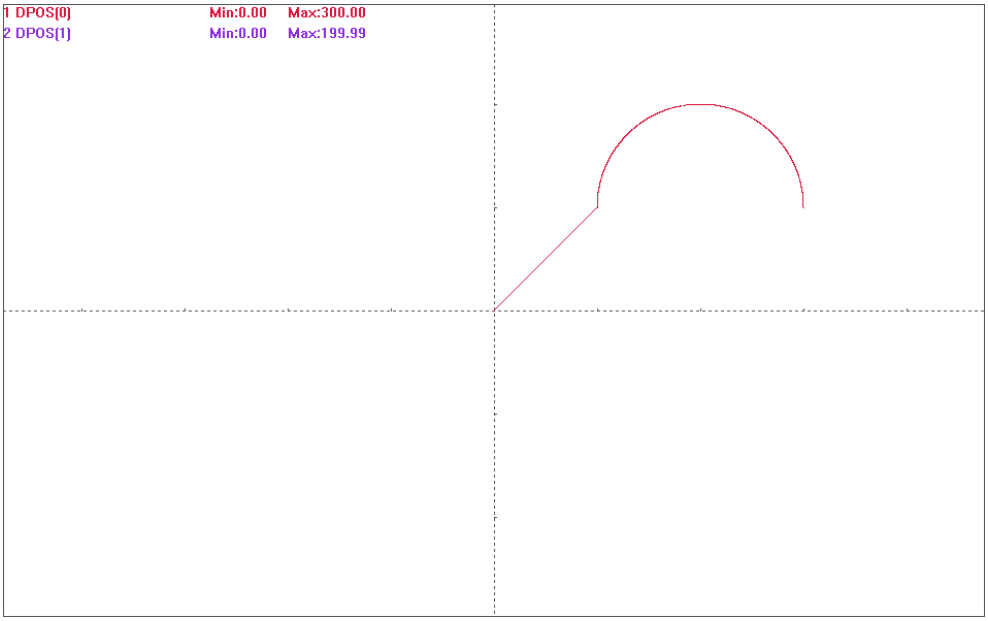

```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)// it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//simulator IP address
	ZMC_HANDLE handle = NULL;				//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to connect controller!\n");

	float UnitsValue = 100;
	float DposValue = 0;
	float SpeedValue = 100;
	float AccelValue = 10000;
	float DecelValue = 10000;
```

```
    int AxisList[2] = { 0,1 };
    float DiatanceValue = 100;

    ret =ZAux_Direct_SetAtype(handle,0,1);//set axis type of axis 0 as 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetAtype(handle,1,1);// set axis type of axis 1 as 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully

    ret = ZAux_Direct_SetUnits(handle, AxisList[0], UnitsValue);//set pulse amount of
axis 0
    commandCheckHandler("ZAux_Direct_SetUnits", ret);

    ret = ZAux_Direct_SetUnits(handle, AxisList[1], UnitsValue);// set pulse amount of
axis 1
    commandCheckHandler("ZAux_Direct_SetUnits", ret);

    ret = ZAux_Direct_SetSpeed(handle, AxisList[0], SpeedValue);//set speed of axis 0
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);

    ret = ZAux_Direct_SetSpeed(handle, AxisList[1], SpeedValue);//set speed of axis 1
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);

    ret = ZAux_Direct_SetAccel(handle, AxisList[0], AccelValue);//set acceleration of
axis 0
    commandCheckHandler("ZAux_Direct_SetAccel", ret);

    ret = ZAux_Direct_SetAccel(handle, AxisList[1], AccelValue);// set acceleration of
axis 1
    commandCheckHandler("ZAux_Direct_SetAccel", ret);

    ret = ZAux_Direct_SetDecel(handle, AxisList[0], DecelValue);//set deceleration of
axis 0
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Direct_SetDecel(handle, AxisList[1], DecelValue);// set deceleration of
axis 1
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Direct_SetDpos(handle, AxisList[0], 0);//set axis 0 dpos as 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);

    ret = ZAux_Direct_SetDpos(handle, AxisList[1], 0);// set axis 1 dpos as 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
```

```
    ret = ZAux_Direct_Single_Cancel(handle, 0, 0); //axis 0 motion stops
    commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
    ret = ZAux_Direct_Single_Cancel(handle, 1, 0); //axis 1 motion stops
    commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
    ret = ZAux_Direct_Single_Cancel(handle, 2, 0); //axis 2 motion stops
    commandCheckHandler("ZAux_Direct_Single_Cancel", ret);

    int run_state = 0; //axis motion state

    ZAux_Trigger( handle);//trigger the oscilloscope

    //move to starting position (100,100)
    ret = ZAux_Direct_Single_Move(handle, AxisList[0], DiatanceValue);
    commandCheckHandler("ZAux_Direct_Single_Move", ret);

    ret = ZAux_Direct_Single_Move(handle, AxisList[1], DiatanceValue);
    commandCheckHandler("ZAux_Direct_Single_Move", ret);
    //move to starting position (100,100)
    int state[2];
    do
    {
        ret = ZAux_Direct_GetIfIdle(handle, AxisList[0], state);
        commandCheckHandler("ZAux_Direct_GetIfIdle", ret);

        ret = ZAux_Direct_GetIfIdle(handle, AxisList[1], state + 1);
        commandCheckHandler("ZAux_Direct_GetIfIdle", ret);

        if (!(state[0] || state[1] ))
        {
            Sleep(100);
        }
        else
        {
            break;
        }
    } while (true);//wait for motion to stop

    //3 points to draw the arc motion, starting point (100, 100), the middle point
(100,100), end point (300,100)
    ret = ZAux_Direct_MoveCirc2(handle, 2, AxisList, 200-100, 200-100, 300-100,
100-100);
    commandCheckHandler("ZAux_Direct_MoveCirc2", ret);

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
```

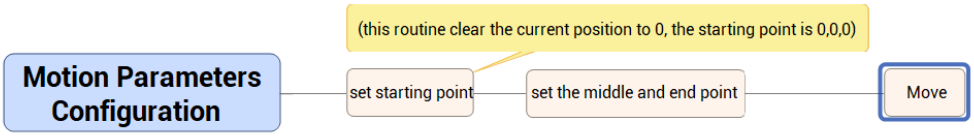
```
    commandCheckHandler("ZAux_Close", ret) ;// judge whether the instruction is executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

- XY plane result:



5.5.2.8. Spatial 3-Point Arc Drawing

This routine uses 3 axes to do space arc movement, the current position point is used as the starting point (0, 0, 0), the end point (120, 160, 400), the middle point (240, 320, 300), mode 0, and three points to draw an arc.



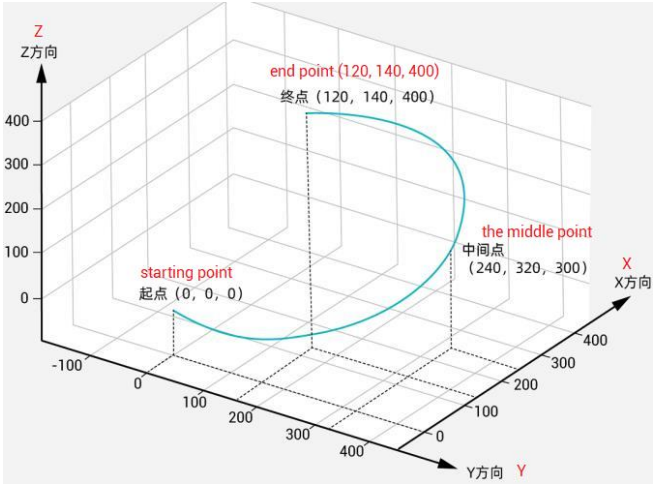

```
// test1.cpp: define the entry point of control panel application program
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
```

```
void commandCheckHandler(const char *command, int ret)
{
	if (ret)// it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//simulator IP address
	ZMC_HANDLE handle = NULL;		//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to connect controller!\n");

	ret =ZAux_Direct_SetAtype(handle,0,1);//set axis type of axis 0 as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully

	ret =ZAux_Direct_SetAtype(handle,1,1);//set axis type of axis 1 as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully

	ret =ZAux_Direct_SetAtype(handle,2,1);//set axis type of axis 2 as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully

	ret = ZAux_Direct_SetSpeed(handle, 0, 200); //set speed of main axis 0 as
200units/s
	commandCheckHandler("ZAux_Direct_SetSpeed", ret);
	ret = ZAux_Direct_SetAccel(handle, 0, 2000); // set acceleration main axis 0 as
2000units/s/s
	commandCheckHandler("ZAux_Direct_SetAccel", ret);
	ret = ZAux_Direct_SetDecel(handle, 0, 2000); // set deceleration of main axis 0 as
2000units/s/s
	commandCheckHandler("ZAux_Direct_SetDecel", ret);

	ret = ZAux_Direct_Single_Cancel(handle, 0, 0); //axis 0 motion stops
	commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
```

```
    ret = ZAux_Direct_Single_Cancel(handle, 1, 0); //axis 1 motion stops
    commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
    ret = ZAux_Direct_Single_Cancel(handle, 2, 0); //axis 2 motion stops
    commandCheckHandler("ZAux_Direct_Single_Cancel", ret);

    int run_state = 0; //axis motion state

    ret = ZAux_Direct_SetDpos(handle, 0, 0);    //set axis 0 dpos as 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, 1, 0);    //set axis 1 dpos as 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, 2, 0);    //set axis 2 dpos as 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);

    ZAux_Trigger( handle);//trigger the oscilloscope

    //the list of axes that participate in the motion
    int axisList[3] = { 0,1,2};

    //starting point (0,0,0), end point (120, 160, 400), the middle point (240, 320, 300),
mode 0, these three points draw the arc
    ret = ZAux_Direct_MSpherical(handle, 3, axisList, 120, 160, 400, 240, 320, 300, 0,
0, 0);
    commandCheckHandler("ZAux_Direct_MSpherical", ret);

    int state[3];

    do
    {
        ret = ZAux_Direct_GetIfIdle(handle, axisList[0], state);
        commandCheckHandler("ZAux_Direct_GetIfIdle", ret);

        ret = ZAux_Direct_GetIfIdle(handle, axisList[1], state + 1);
        commandCheckHandler("ZAux_Direct_GetIfIdle", ret);

        ret = ZAux_Direct_GetIfIdle(handle, axisList[2], state + 2);
        commandCheckHandler("ZAux_Direct_GetIfIdle", ret);

        if (!(state[0] || state[1] || state[2]))
        {
            Sleep(100);
        }
        else
        {
            break;
        }
```

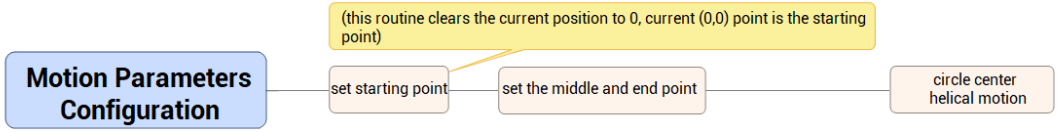
```
        } while (true);//wait for motion to stop

        printf("Idle\n");

        Sleep(2000);
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;// judge whether the instruction is
executed successfully
        printf("connection closed!\n");
        handle = NULL;
        return 0;
}
```

➢ Trajectory effects:



5.5.2.9. Helical Curve

This routine uses 3 axes to do space arc movement, the current position point, the origin as the starting point (0, 0), the end point (200, -200), the middle point (200, 0), the Z axis participates in the movement, it moves 100, makes a clockwise spiral movement.



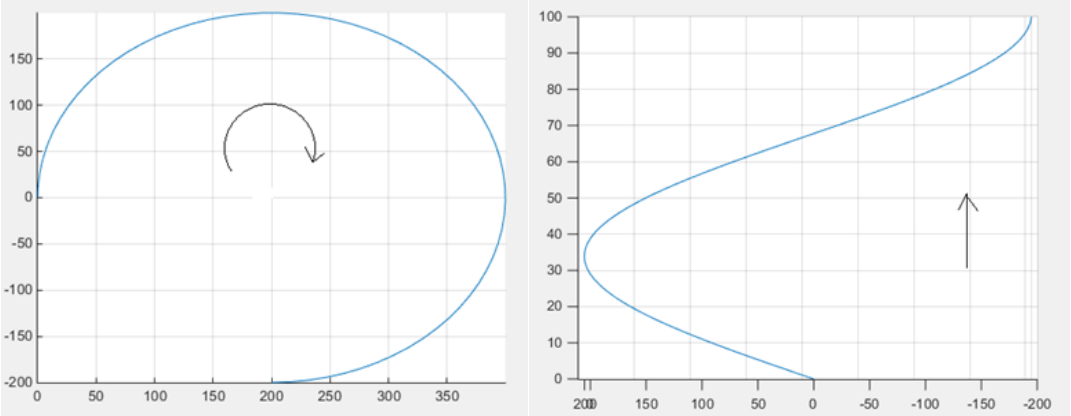

```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
```

```
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)// it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//simulator IP address
	ZMC_HANDLE handle = NULL;			//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to connect controller!\n");

	ret =ZAux_Direct_SetAtype(handle,0,1);//set axis type of axis 0 as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully

	ret =ZAux_Direct_SetAtype(handle,1,1);//set axis type of axis 1 as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully

	ret =ZAux_Direct_SetAtype(handle,2,1);//set axis type of axis 2 as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully

	ret = ZAux_Direct_Single_Cancel(handle, 0, 0); //axis 0 motion stops
	commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
	ret = ZAux_Direct_Single_Cancel(handle, 1, 0); //axis 1 motion stops
	commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
	ret = ZAux_Direct_Single_Cancel(handle, 2, 0); //axis 2 motion stops
	commandCheckHandler("ZAux_Direct_Single_Cancel", ret);

	int run_state = 0; //axis motion state
```

```
    ret = ZAux_Direct_SetDpos(handle, 0, 0);    //set axis 0 dpos as 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, 1, 0);    //set axis 1 dpos as 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, 2, 0);    //set axis 2 dpos as 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);

    ret = ZAux_Direct_SetSpeed(handle, 0, 100); //set master axis 0 speed as
100units/s
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ret = ZAux_Direct_SetAccel(handle, 0, 10000); // set master axis 0 acceleration as
10000units/s/s
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ret = ZAux_Direct_SetDecel(handle, 0, 10000); // set master axis 0 deceleration as
10000units/s/s
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    //the list of axes that participate in motion
    int axisList[3] = { 0,1,2 };
    ZAux_Trigger( handle);//trigger the oscilloscope

    //the origin is used as the starting point, the center is (200,0), the end point is
(200,-200), in clockwise, the Z axis participates in the speed calculation, the
movement is 100, and then do the circle center spiral movement
    ret = ZAux_Direct_MHelical(handle, 3, axisList, 200,-200, 200, 0, 1, 100,0);
    commandCheckHandler("ZAux_Direct_MoveCirc", ret);

    int state[3];
    do
    {
        ret = ZAux_Direct_GetIfIdle(handle, axisList[0], state);
        commandCheckHandler("ZAux_Direct_GetIfIdle", ret);

        ret = ZAux_Direct_GetIfIdle(handle, axisList[1], state + 1);
        commandCheckHandler("ZAux_Direct_GetIfIdle", ret);

        ret = ZAux_Direct_GetIfIdle(handle, axisList[2], state + 2);
        commandCheckHandler("ZAux_Direct_GetIfIdle", ret);

        if (!(state[0] || state[1] || state[2]))
        {
            Sleep(100);
        }
        else
        {
            break;
```

```
            }
        } while (true);//wait for the motion to stop

        printf("Idle\n");

        Sleep(2000);
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;// judge whether the instruction is
executed successfully
        printf("connection closed!\n");
        handle = NULL;
        return 0;
}
```

➢ Interpolation trajectory:



5.6. Axis Pause / Resume / Stop

5.6.1. Emphasis

During single-axis or multi-axis interpolation motion, the motion control card supports functions such as motion pause and axis stop.

Note: when using the axis stop function, the movement will stop at the fast deceleration (**ZAux_Direct_SetFastDec**). If the fast deceleration is not set, the fast deceleration will be equal to the set deceleration (**ZAux_Direct_SetDecel**) by default, so when using the axis stop function, deceleration or fast deceleration must be set. ZAux_Direct_Single_Cancel, when using mode 3, it cannot be used to stop the superimposed axis, otherwise it may cause the position of the superimposed axis to be

inconsistent.

5.6.2.Routine

5.6.2.1. Axis Pause & Resume

This routine uses axis 0 to do single-axis motion, it moves 2000 units in positive direction, motion starts, and it will pause after 1s, then resume after 4s.


```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;				//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		getchar();
		return -1;
	}
	printf("Success to connect controller!\n");
```

```
    ZAux_Direct_SetAtype(handle, 0, 1);   //set axis type of axis 0 as 1
    ZAux_Direct_SetUnits(handle, 0, 100); //set pulse amount of axis 0 as 100
    ZAux_Direct_SetSpeed(handle, 0, 200); //set speed of axis 0 as 200units/s
    ZAux_Direct_SetAccel(handle, 0, 2000); //set acceleration of axis 0 as
2000units/s/s
    ZAux_Direct_SetDecel(handle, 0, 2000); //set deceleration of axis 0 as
2000units/s/s
    ZAux_Direct_SetSramp(handle, 0, 200); //set S curve of axis 0 as 200ms

    ret = ZAux_Direct_SetDpos( handle,  0,  0);//clear axis command position to 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//judge whether the
instruction is executed successfully
    ret = ZAux_Direct_SetMpos( handle,  0,  0);//clear encoder feedback position to 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);//judge whether the
instruction is executed successfully
    int GetVal;
    ret =ZAux_Direct_Single_Move( handle,  0,  2000);
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//judge whether the
instruction is executed successfully

    Sleep(1000);//wait 1s
    ZAux_Direct_MovePause(handle, 0,0);//pause the current motion of axis 0

    Sleep(2000);//wait 2s

    ZAux_Direct_GetAxisStatus(handle, 0,&GetVal);//get the axis stop reason
    printf("current axis state: %x h\n",GetVal);

    Sleep(2000);//wait 2s
    ZAux_Direct_MoveResume(handle, 0);//resume axis 0 motion

    float IDLE;
    while (1)//wait for axis 0 to complete the motion
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
```

```
    return 0;
}
```

Chapter VI Advanced Motion Control

6.1.Coordinates Loop

6.1.1.Emphasis

The instructions in this chapter can be used to limit the coordinate cycle range of the cam main axis to realize the continuity of multiple cam curves.

When using the absolute motion mode, if the target position is within the range of the coordinate cycle, it can move correctly, if it is outside the range of the coordinate cycle, the motion is incorrect.

Relative motion is not influenced.

ZAux_Direct_SetRepOption (coordinate cycle mode): bit by bit to represent different meanings

Bit	Value	Description
0	1	0 – loop range: -coordinate loop position (**ZAux_Direct_SetRepDist**) ~ + coordinate loop position (**ZAux_Direct_SetRepDist**) 1 – loop range: 0 ~ coordinate loop position (**ZAux_Direct_SetRepDist**)
1	2	The repeat motion between this position 1 follow cam table motion (**ZAux_Direct_Cambox**) and automatic cam (**ZAux_Direct_MoveLink**), and it restores to 0 when prohibit to be valid.
2	4	Reserved
4	16	1 – not to use **ZAux_Direct_SetRepDist.** 0 – use **ZAux_Direct_SetRepDist.**

6.1.2.Routine

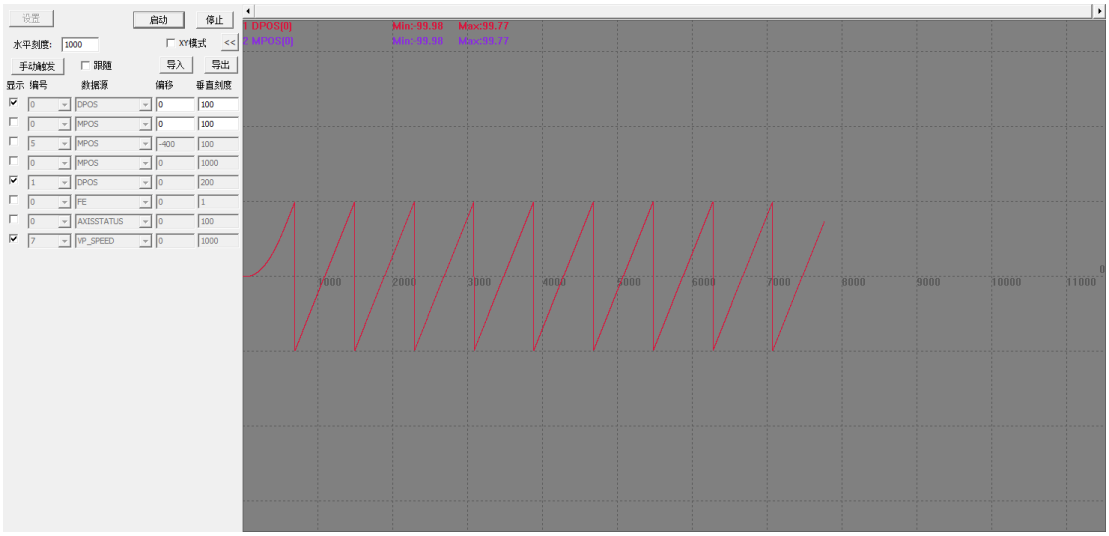
6.1.2.1. Coordinates Loop

This routine shows coordinates loop motion, axis 0 starts to do loop motion from -100 to 100 from the origin position and to do direct loop motion from 0 to 100.

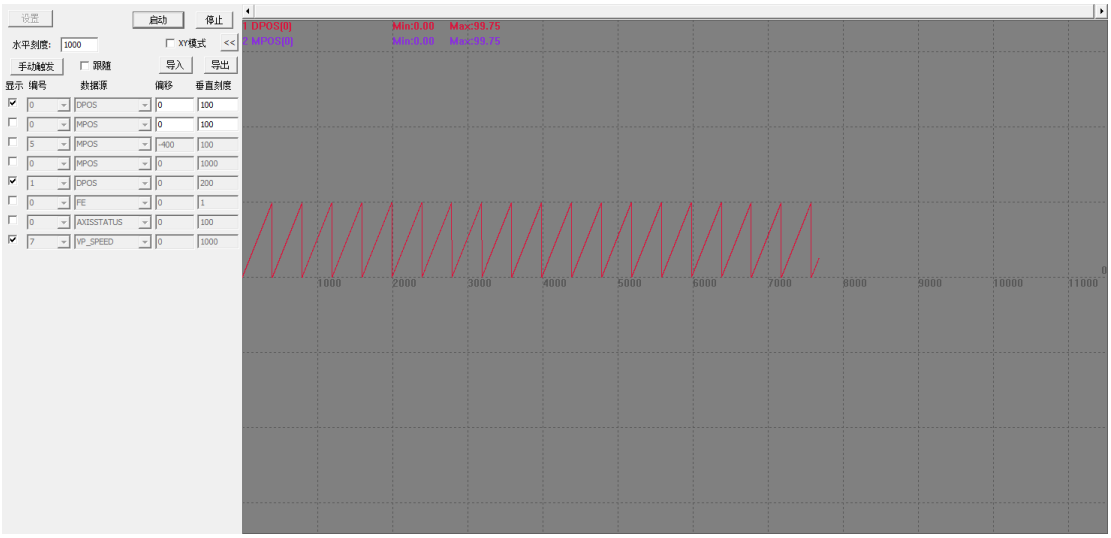

```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}

}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;			//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		getchar();
		return -1;
	}
	printf("Success to connect controller!\n");

	ret =ZAux_Direct_SetAtype(handle,0,1);//set axis type as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//judge whether the
instruction is executed successfully
	ret =ZAux_Direct_SetAccel(handle,0,500);//set axis acceleration
	commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//judge whether the
instruction is executed successfully
	ret =ZAux_Direct_SetDecel(handle,0,500);//set axis deceleration
	commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
instruction is executed successfully
	ret =ZAux_Direct_SetDpos(handle,0,0);//set clear axis DPOS to 0
```

```
	commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully

	ret =ZAux_Direct_SetSpeed(handle,0,250);//set axis speed
	commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
instruction is executed successfully


	ret =ZAux_Direct_SetRepDist(handle,0,100);//set loop coordinates as 100
	commandCheckHandler("ZAux_Direct_SetRepDist", ret) ;//judge whether the
instruction is executed successfully
	ret =ZAux_Direct_SetRepOption(handle,0,1);//set loop mode as 1
	commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully


	float IDLE;
	int mode;
	int AxisStatus;
	ZAux_Trigger( handle);//trigger the oscilloscope

	ZAux_Direct_Single_Vmove(handle,0,1);//axis 0 keeps moving

	while (1)//wait for axis 0 to complete the motion
	{
		Sleep(100);
		ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
		if (IDLE<0)break;
	}

	getchar();
	ret = ZAux_Close(handle);    //close the connection
	commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
	printf("connection closed!\n");
	handle = NULL;
	return 0;
}
```

- **ZAux_Direct_SetRepOption** (coordinates loop mode) = 0, it loops from -100 to 100.



➢ **ZAux_Direct_SetRepOption** (coordinates loop mode) = 1, it loops from 0 to 100.



6.2. Pitch Compensation

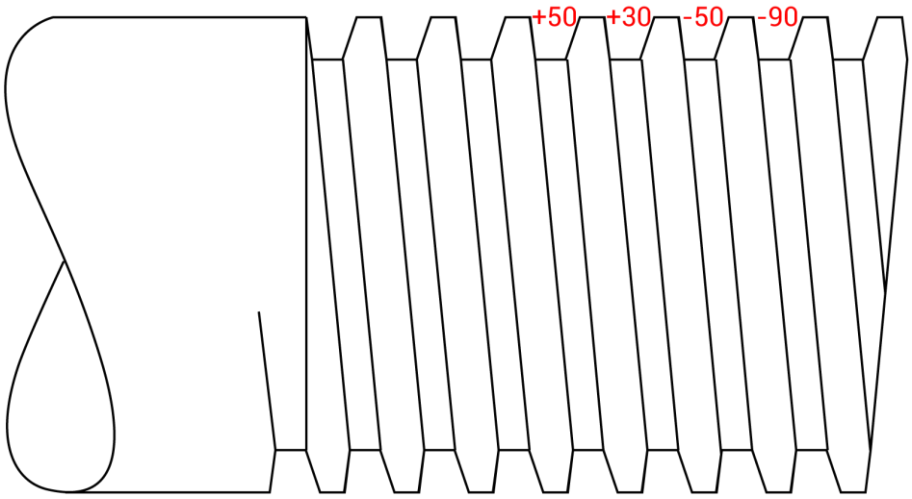
6.2.1. Emphasis

For open-loop and semi-closed-loop systems, the positioning accuracy is largely affected by the accuracy of the ball screw. Although the accuracy of the ball screw is high, there are always manufacturing errors. In order to obtain a motion accuracy exceeding the accuracy of the ball screw, the pitch error compensation function must be used in the system, and the system is used to compensate and correct the error to ensure the machining accuracy.

The instructions in this chapter are mainly functions of the pitch compensation, which are mainly used when the equipment mechanism has manufacturing errors, then the pitch compensation can be used to ensure the accuracy.

6.2.2.Routine

6.2.2.1. Pitch Compensation



There is a slight error in the pitch of each circle, and it is not strictly uniform, so pitch compensation is required for adjustment in high-precision applications. This routine drives axis 0 to move back and forth with a distance of 700, and the corresponding pulse number needs to be compensated in the following list.

Starting compensation point position (MPOS)	Compensation value (the number of pulses when the distance is 100)
100	0
200	50 pulses
300	30 pulses
400	-50 pulses
500	-90 pulses
600	0

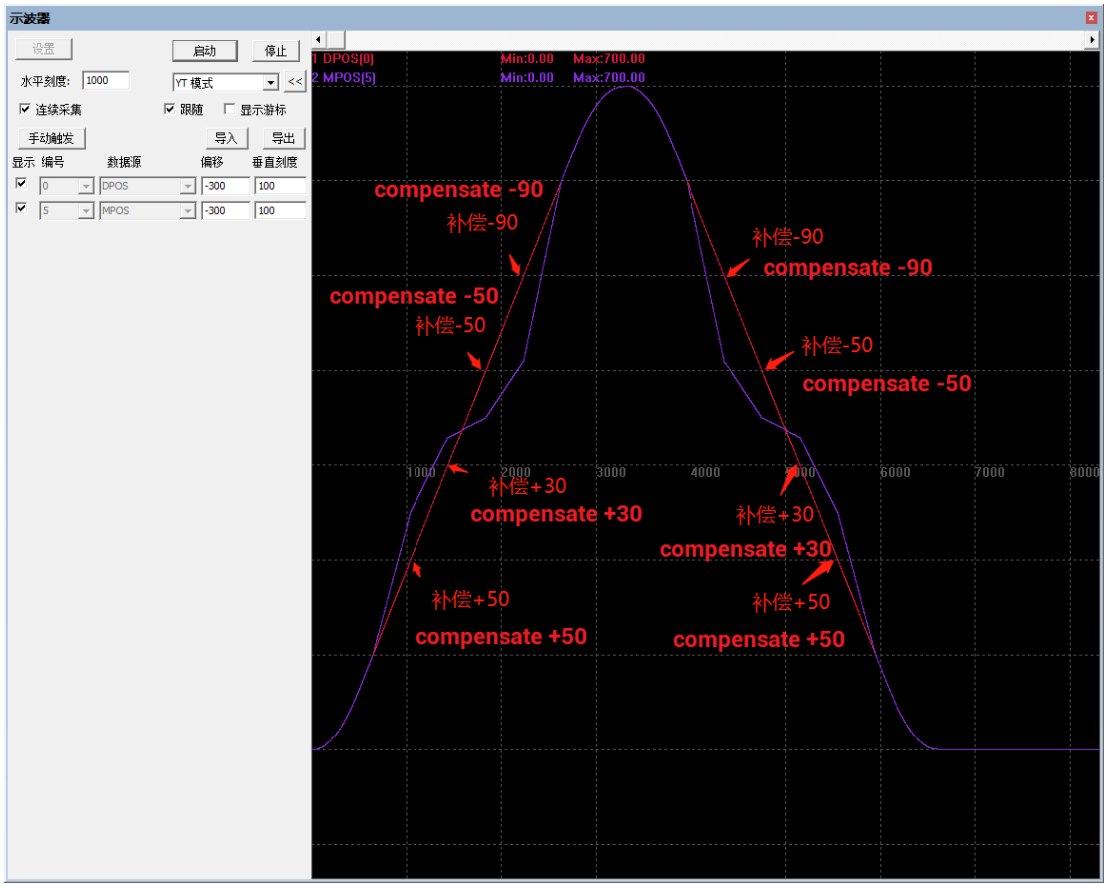
```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
```

```
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}

}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"192.168.0.11";		//controller IP address
	ZMC_HANDLE handle = NULL;		//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		getchar();
		return -1;
	}
	printf("Success to connect controller!\n");

	ret =ZAux_Direct_SetAtype(handle,0,1);//set axis type as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//judge whether the
instruction is executed successfully
	ret =ZAux_Direct_SetAtype(handle,5,6);//set axis type as 6
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//judge whether the
instruction is executed successfully

	ret =ZAux_Direct_SetAccel(handle,0,500);//set axis acceleration
	commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//judge whether the
instruction is executed successfully
	ret =ZAux_Direct_SetDecel(handle,0,500);//set axis deceleration
	commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
instruction is executed successfully
	ret =ZAux_Direct_SetDpos(handle,0,0);//set clear axis DPOS to 0
	commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully
	ret =ZAux_Direct_SetMpos(handle,0,0);//set clear axis MPOS to 0
	commandCheckHandler("ZAux_Direct_SetMpos", ret) ;//judge whether the
instruction is executed successfully
```

```
    ret =ZAux_Direct_SetDpos(handle,5,0);// set clear axis DPOS to 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetMpos(handle,5,0);// set clear axis MPOS to 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret) ;//judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetSpeed(handle,0,250);//set axis speed
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
instruction is executed successfully

    float numlist[4]={50,100,50,25};

    //the position of starting compensation (MPOS)
      compensation value (the number of pulses when the distance is 100)
    //   100                              0
    //   200                              50 pulses
    //   300                              30 pulses
    //   400                              -50 pulses
    //   500                              -90 pulses
    //   600                               0
    ret =ZAux_Direct_Pitchset(handle,0,1,100,4,100,0,numlist);//set pitch
compensation
    commandCheckHandler("ZAux_Direct_Pitchset", ret) ;//judge whether the
instruction is executed successfully

    float IDLE;
    int mode;
    int AxisStatus;
    ZAux_Trigger( handle);//trigger the oscilloscope

    ZAux_Direct_Single_Move(handle,0,700);//axis 0 moves 700

    while (1)//wait for axis 0 to complete the motion
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }
    ZAux_Direct_Single_Move(handle,0,-700);//axis 0 moves -700

    while (1)//wait for axis 0 to complete the motion
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
```

```
            if (IDLE<0)break;
        }

        ret =ZAux_Direct_Pitchset(handle,0,0,100,4,100,0,numlist);//close pitch
compensation
        commandCheckHandler("ZAux_Direct_Pitchset", ret) ;//judge whether the
instruction is executed successfully

        getchar();
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
        printf("connection closed!\n");
        handle = NULL;
        return 0;
}
```

Waveform diagram: (in this waveform diagram, in order to better display the compensation effect, the pulse output of axis 0 is connected to the encoder input of axis 5, so that it is convenient to observe the difference between the planned position and the actual position to see the compensation effect)



6.3. Backlash Compensation

6.3.1. Emphasis

Because there must be a certain gap between the screw and the nut, so when it is converted into reverse from forward, within a certain angle, although the screw rotates, the nut can only move the work table after the gap is eliminated. And this gap is the backlash, but it should be reflected in the rotation angle of the lead screw.

On the equipment, there is a backlash in each link of the transmission chain, such as gear transmission, ball screw nut pair, etc. The backlash is one of the factors that affect the machining accuracy. When the CNC machine tool table changes direction in its direction of motion, the existence of the backlash will cause the servo motor to idle without actually moving the table, which is called loss of motion. If the value of the backlash is small, there is no big effect on the machining accuracy, so no measures need to be taken, but if the value is large, the stability of the system will be significantly reduced, and the machining accuracy will be obviously reduced, especially for curve processing, which will affect the dimensional tolerance and the consistency of the curve, at this time, the elimination or compensation of the backlash must be carried out to improve the machining accuracy.

6.3.2. Routine

6.3.2.1. Backflash Compensation

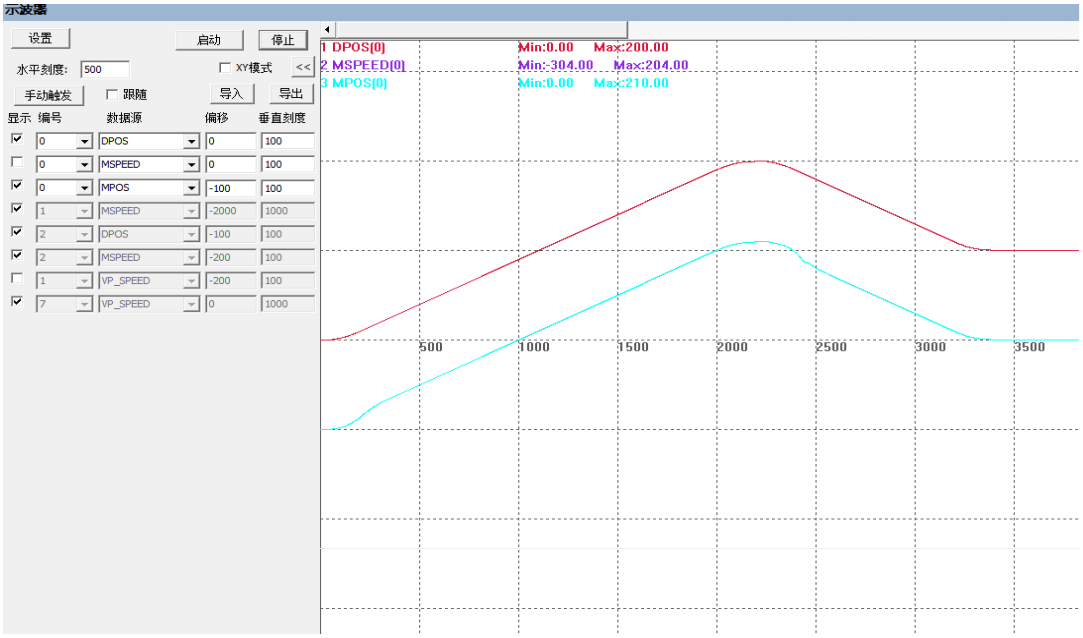
For example: assume that a mechanism has a backlash of 10 pulse equivalents. The following example demonstrates that the mechanism moves 200 in the forward direction and then moves 100 in the reverse direction. Using backlash compensation, the backlash is automatically compensated.

```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
```

```
void commandCheckHandler(const char *command, int ret)
{
	if (ret)// it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"192.168.0.11";		//controller IP address
	ZMC_HANDLE handle = NULL;		//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to connect controller!\n");

	ret =ZAux_Direct_SetAtype(handle,0,4);//set axis type as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully
	ret =ZAux_Direct_SetUnits(handle,0,1);//set pulse amount as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully

	ret =ZAux_Direct_SetAccel(handle,0,500);//set axis acceleration
	commandCheckHandler("ZAux_Direct_SetAccel", ret) ;// judge whether the
instruction is executed successfully
	ret =ZAux_Direct_SetDecel(handle,0,500);//set axis deceleration
	commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully
	ret =ZAux_Direct_SetDpos(handle,0,0);//set clear axis DPOS to 0
	commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the
instruction is executed successfully
	ret =ZAux_Direct_SetMpos(handle,0,0);//set clear axis MPOS to 0
	commandCheckHandler("ZAux_Direct_SetMpos", ret) ;// judge whether the
instruction is executed successfully

	ret =ZAux_Direct_SetSpeed(handle,0,1000);//set axis speed
	commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully
```

```
    ret =ZAux_Direct_SetMerge(handle,0,1);//set axis speed
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_Backlash(handle,0,1,10,100,100);//open axis 0 backflash
compensation, compensate 10
    commandCheckHandler("ZAux_Direct_Backlash", ret) ;// judge whether the
instruction is executed successfully

    float IDLE;
    int mode;
    int AxisStatus;
    ZAux_Trigger( handle);//trigger the oscilloscope

    ZAux_Direct_Single_Move(handle,0,200);//axis 0 moves 200
    ZAux_Direct_Single_Move(handle,0,-100);//axis 0 moves -100

    while (1)//wait for axis 0 to complete the motion
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }
    ret =ZAux_Direct_Backlash(handle,0,0,0,100,100);//close axis 0 backflash
compensation
    commandCheckHandler("ZAux_Direct_Backlash", ret) ;// judge whether the
instruction is executed successfully

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;// judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

Motion waveform: it can be seen when the direction is switched, the blue one actual position moves more than read one planning position:


示波器
设置
启动
停止
水平刻度:
500
XY模式
<<
手动触发
跟随
导入
导出
显示 编号 数据源 偏移 垂直刻度
1 DPOS[0]
2 MSPEED[0]
3 MPOS[0]
Min:0.00 Max:200.00
Min:-304.00 Max:204.00
Min:0.00 Max:210.00
500
1000
1500
2000
2500
3000
3500

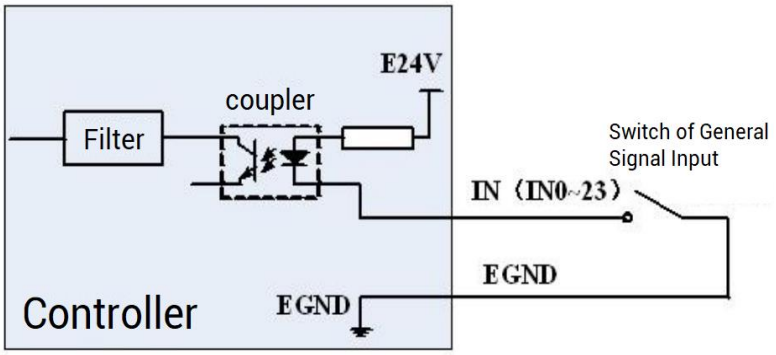
Chapter VII Hardware Interface Access and Configuration

7.1.Digital Inputs and Outputs

7.1.1.Emphasis

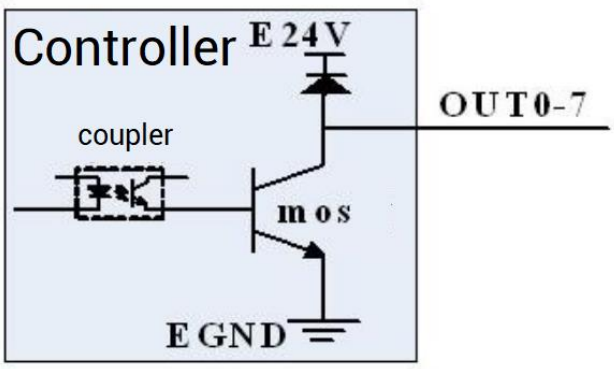
Take the ZMC406 controller as an example, the input and output ports of the controller do not need to be powered separately by an external IO power supply, and the IO points of the expansion module can be expanded up to 512 points.

The signal type allowed to be connected to the digital input port IN of the controller is NPN, 24V signal, and the internal circuit diagram is as follows. IN port voltage below 7v is low level. So the input port is divided into high-speed input port and common input port. The high-speed input port hardware allows an input signal frequency of 500K, and the other common IN port hardware allows an input frequency of about 10K.



The OUT signal type of the digital output port of the controller is NPN, 0V signal, and the output current is 300mA. The internal circuit diagram is as follows.

The high-speed output port outputs signals, the maximum output frequency of the hardware is 1M, and the hardware output frequency of the ordinary output port is about 10k.



Output Circuit

➢ Please refer to corresponding hardware manuals for detailed parameters.

7.1.2.Routine

7.1.2.1. IO Read & Settings

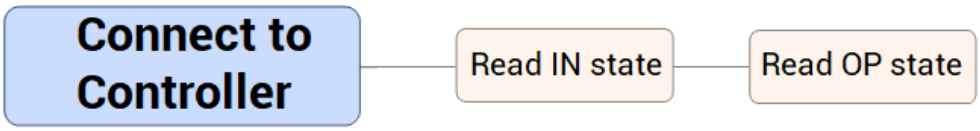
This routine gets input state of simulator IN0, and gets the state of OP0, then resetsOP0 according to OP0 input state, at the last, reads OP0 state.


```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
```

```
    char *ip_addr = (char *)"127.0.0.1";          //controller IP address
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to connect controller!\n");

    unsigned int in_value;
    unsigned int out_value;
    ret = ZAux_Direct_GetIn(handle, 0, &in_value);    //read IN0 state
    commandCheckHandler("ZAux_Direct_GetIn", ret);
    printf("IN0 status: %d\n", in_value);
    ret = ZAux_Direct_GetOp(handle, 0, &out_value);   //read OP0 state
    commandCheckHandler("ZAux_Direct_GetOp", 0);
    printf("OP0 status: %d\n", out_value);

    if(out_value==0)
    {
        ret = ZAux_Direct_SetOp(handle, 0, 1);          //set OP0 state
        commandCheckHandler("ZAux_Direct_SetOp", 0);
    }else
    {
        ret = ZAux_Direct_SetOp(handle, 0, 0);          // set OP0 state
        commandCheckHandler("ZAux_Direct_SetOp", 0);
    }
    ret = ZAux_Direct_GetOp(handle, 0, &out_value);   //read OP0 state
    commandCheckHandler("ZAux_Direct_GetOp", 0);
    printf("OP0 status: %d\n", out_value);

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

7.1.2.2. Read Multi-IO (GetModbusOut/GetModbusIn)

This routine gets input states of simulator IN0-7, then gets states of OP0-OP7.

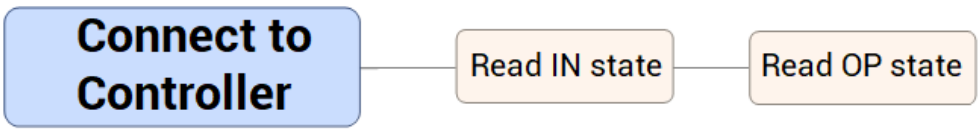



```
// test1.cp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)// it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;				//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to connect controller!\n");

	uint8 GetValue[8];						//read set value

	ret = ZAux_GetModbusIn( handle, 0 , 7, GetValue);

	commandCheckHandler("ZAux_GetModbusIn", ret);

	for (int i = 0; i <= 7; i++)
```

```
    {
        printf("IN(%d) = %d  ", i, (GetValue[0] & (0x1 << i)) >> i);
    }
    printf("\n");
    ret = ZAux_GetModbusOut(handle, 0 , 7 , GetValue);
    commandCheckHandler("ZAux_GetModbusOut", ret);

    for (int i = 0; i <= 7; i++)
    {
        printf("OP(%d) = %d  ", i, (GetValue[0] & (0x1 << i)) >> i);
    }

    printf("\n");

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

7.1.2.3. Multi-IO Reading Configuration

This routine gets input states of simulator IN0-7, then gets states of OP0-OP7.




```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)// it is not 0, fail
    {
        printf("%s fail!return code is %d\n", command, ret);
```

```
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";          //controller IP address
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to connect controller\n");

    int32 GetValue;                                    //read set value
    uint32 uValue;

    ret = ZAux_Direct_GetInMulti( handle, 0 , 7, &GetValue);

    commandCheckHandler("ZAux_GetModbusIn", ret);

    for (int i = 0; i <= 7; i++)
    {
        printf("IN(%d) = %d  ", i, (GetValue & (0x1 << i)) >> i);
    }
    printf("\n");
    uValue=1;
    ret = ZAux_Direct_SetOutMulti(handle, 0 , 7 , &uValue);
    commandCheckHandler("ZAux_GetModbusOut", ret);

    ret = ZAux_Direct_GetOutMulti(handle, 0 , 7 , &uValue);
    commandCheckHandler("ZAux_GetModbusOut", ret);

    for (int i = 0; i <= 7; i++)
    {
        printf("OP(%d) = %d  ", i, (uValue & (0x1 << i)) >> i);
    }
    printf("\n");

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
```

```
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

7.2. Analog Input & Output

7.2.1. Emphasis

The controller AD/DA uses an internal power supply. For example, the resolution of ZMC416BE analog is 12 bits, 0-10V voltage type, and the corresponding scale is 0-4095. The controller has 2 channels of AD and 2 channels of DA, which can be expanded by analog expansion module (up to 256 channels of AD and 128 channels of DA). There are expansion modules with 12-bit resolution and 16-bit resolution. The controller defaults to 12-bit resolution.

PIN	Name	Description
1	DA0	0-10V analog output 0
2	DA1	0-10V analog output 1
3	AGND	Analog GND
4	AD0	0-10V analog input 0
5	AD1	0-10V analog input 1

7.2.2. Routine

7.2.2.1. AD/DA Setting & Reading

This routine obtains the input state of the simulator AD0, then obtains the state of DA0, and then re-reads the state of DA0 after setting DA0


```
// test1.cpp: define the entry point of control panel application program
//
```

```
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)// it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;				//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to connect controller!\n");

	float AD_value;
	float DA_value;
	ret = ZAux_Direct_GetAD(handle,0,&AD_value);//read AD0 state
	commandCheckHandler("ZAux_Direct_GetAD", ret);
	printf("read AD_value: %f\n",AD_value);

	ret = ZAux_Direct_GetDA(handle,0,&DA_value);//read DA0 state
	commandCheckHandler("ZAux_Direct_GetDA", ret);
	printf("read DA_value: %f\n",DA_value);
	Sleep(1000);
	ret = ZAux_Direct_SetDA(handle,0,3000);//set DA0 state
	commandCheckHandler("ZAux_Direct_SetDA", ret);
	Sleep(1000);
	ret = ZAux_Direct_GetDA(handle,0,&DA_value);// read DA0 state
	commandCheckHandler("ZAux_Direct_GetDA", ret);
	printf("read DA_value: %f\n",DA_value);

	Sleep(2000);
	ret = ZAux_Close(handle);	//close the connection
```

```
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

7.3.PWM Control

7.3.1.Emphasis

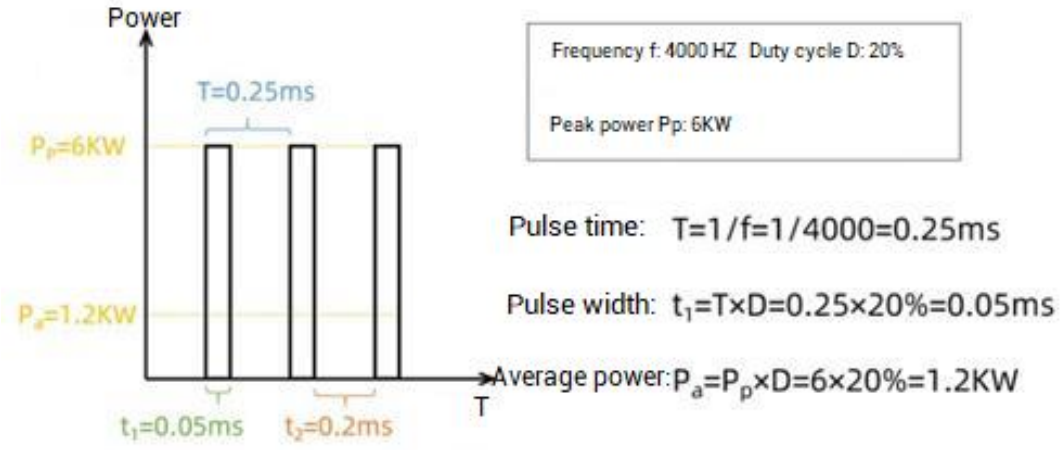
PWM is the abbreviation of "Pulse Width Modulation" in English, it is a very effective technology to control analog circuits by using the digital output of a microprocessor.

PWM (Pulse Width Modulation) is to adjust the pulse width by adjusting the duty cycle. Only controllers that support PWM function can be used

When adjusting the process parameters of laser cutting, the pulse frequency and duty cycle are often used. Many people who are not familiar with the laser cutting process may not understand the role of these two parameters. Next, let's see the detailed explanation.

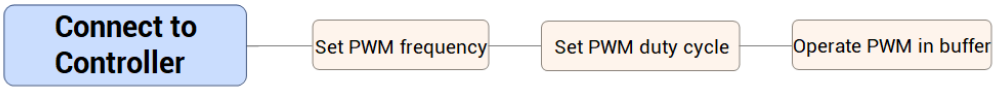
- cycle signal: one cycle is from high level to low level and back to high level
- Pulse frequency: how many cycles of PWM in 1 second.
- Pulse duty cycle: the ratio of the high level time to the entire cycle time in one cycle.
- Pulse time: the time used for one cycle.
- Pulse width: the time used for high level in one cycle.
- Laser peak power: the instantaneous energy output within a pulse width.
- Average power: the energy output by the laser in one repetition period.

Next, we use an example to illustrate how to calculate the various parameters of PWM.



7.3.2.Routine

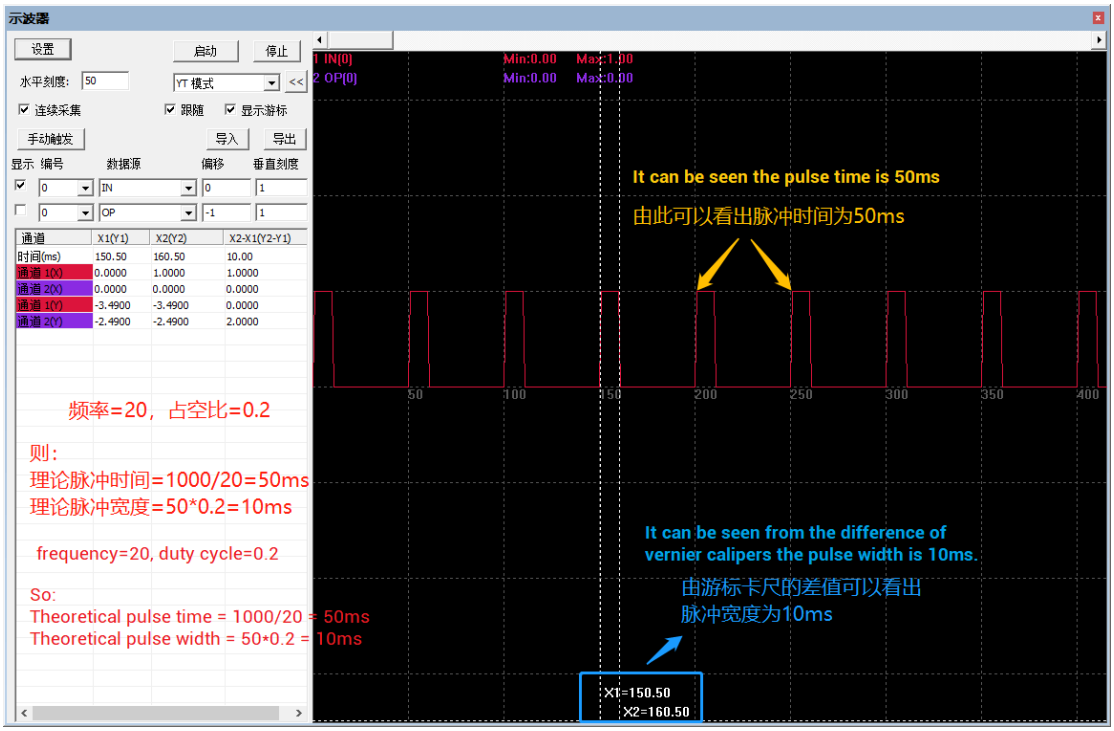
7.3.2.1. Use PWM




```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)// it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"192.168.0.11";		//controller IP address
	ZMC_HANDLE handle = NULL;			//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
```

```
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to connect controller!\n");

    float freq_value;
    float duty_value;
    ret = ZAux_Direct_SetPwmFreq(handle, 0,20);        //set the PWM frequency, it must be set before setting the duty cycle, here it is set to 20, so that the Zdevelop oscilloscope can capture the waveform
    commandCheckHandler("ZAux_Direct_SetPwmFreq", ret);
    ZAux_Trigger( handle);//trigger the oscilloscope

    ret = ZAux_Direct_SetPwmDuty(handle, 0, 0.2);        //set PWM duty cycle, it is bigger than 0, output PWM
    commandCheckHandler("ZAux_Direct_SetPwmDuty", ret);

    Sleep(1000);

    ret = ZAux_Direct_SetPwmDuty(handle, 0, 0);        //set PWM duty cycle as 0, close PWM output
    commandCheckHandler("ZAux_Direct_SetPwmDuty", ret);

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

- Waveform captured by the oscilloscope:


示波器
设置
启动
停止
水平刻度: 50
YT 模式
连续采集
跟随
显示游标
手动触发
导入
导出
显示 编号 数据源 偏移 垂直刻度
通道 X1(Y1) X2(Y2) X2-X1(Y2-Y1)
时间(ms) 150.50 160.50 10.00
通道 1(X) 0.0000 1.0000 1.0000
通道 2(X) 0.0000 0.0000 0.0000
通道 1(Y) -3.4900 -3.4900 0.0000
通道 2(Y) -2.4900 -2.4900 2.0000
1 IN[0] Min:0.00 Max:1.00
2 OP[0] Min:0.00 Max:0.00
It can be seen the pulse time is 50ms
由此可以看出脉冲时间为50ms
50 100 150 200 250 300 350 400
频率=20，占空比=0.2
则：
理论脉冲时间=1000/20=50ms
理论脉冲宽度=50*0.2=10ms
frequency=20, duty cycle=0.2
So:
Theoretical pulse time = 1000/20 = 50ms
Theoretical pulse width = 50*0.2 = 10ms
It can be seen from the difference of vernier calipers the pulse width is 10ms.
由游标卡尺的差值可以看出脉冲宽度为10ms
X1=150.50
X2=160.50

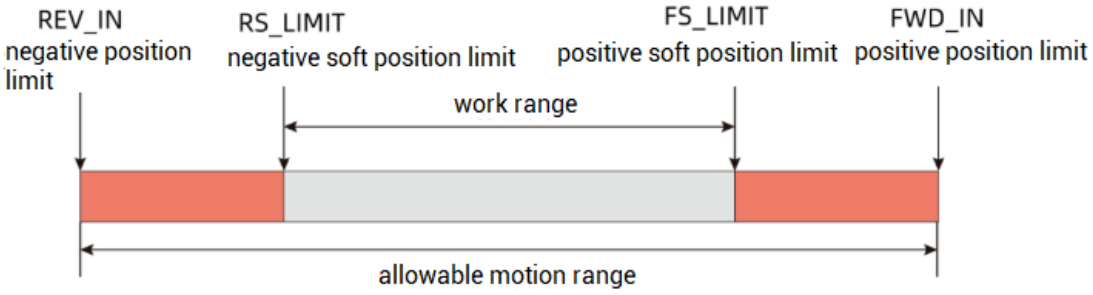
Chapter VIII Security Mechanism

8.1.Axis Software Position Limit Function

8.1.1.Emphasis

Positive software limit and negative software limit are both the position limit function of the controller. They are software limits and have nothing to do with hardware. They limit the range of the command position. When **ZAux_Direct_SetFsLimit** (positive software limit) or **ZAux_Direct_SetRsLimit** (negative software limit) is greater than **ZAux_Direct_SetRepDist** (coordinate cycle position, refer to Chapter 7.1 for details), the parameter does not work, which means the positive and negative software limits are prohibited. When canceling the soft limit, it is recommended not to modify the value of **ZAux_Direct_SetRepDist**, but to set a larger value for **ZAux_Direct_SetFsLimit** or **ZAux_Direct_SetFsLimit**. And the value of **ZAux_Direct_SetFsLimit** or **ZAux_Direct_SetFsLimit** is 200000000/-200000000 by default.

The motion controller can limit the range of motion of each axis by installing limit switches or setting soft limits. The hardware and software limit switches are used for the permissible range of motion and working range of the axes of the technology object.



Soft is unlike hard limit switches, soft limit switches are only realized through software program settings without external switching elements. The soft limit switch will limit the "working range" of the axis, and the limit position is directly set by the command. After the axis reaches the set position, it will immediately stop moving with the fast deceleration STDEC (**ZAux_Direct_SetFastDec**), and the corresponding axis state position is 1, they should be located inside the associated hardware limit switch that limits the range of travel of the machine tool. Because the position of the soft limit switch is relatively flexible, the working range of the axis can be adjusted according to

the current running track and specific requirements.

When the workbench touches the limit switch or the planned position exceeds the soft limit, the motion controller will stop the movement of the workbench urgently. After the limit is triggered, the axis cannot continue to move. At this time, the position of the axis needs to be adjusted to make it far away from the limit position to restart the movement.

The axis will only generate a stop signal when it hits the limit. At this time, it takes a certain amount of time to decelerate, and the actual axis position will exceed the limit by a certain distance. Assume that the SPEED (**ZAux_Direct_SetSpeed**) speed is v0 when stopping, and the fast deceleration STDEC (**ZAux_Direct_SetFastDec**) is a, the calculation formula is $v_t^2-v_0^2=2as$, then bring in the following data: $0-100^2=2*(-1000)*s$, get the overshoot distance s=5, from this, increase STDEC (**ZAux_Direct_SetFastDec**) and reducing SPEED (**ZAux_Direct_SetSpeed**) can achieve the purpose of reducing overshoot.

8.1.2.Routine

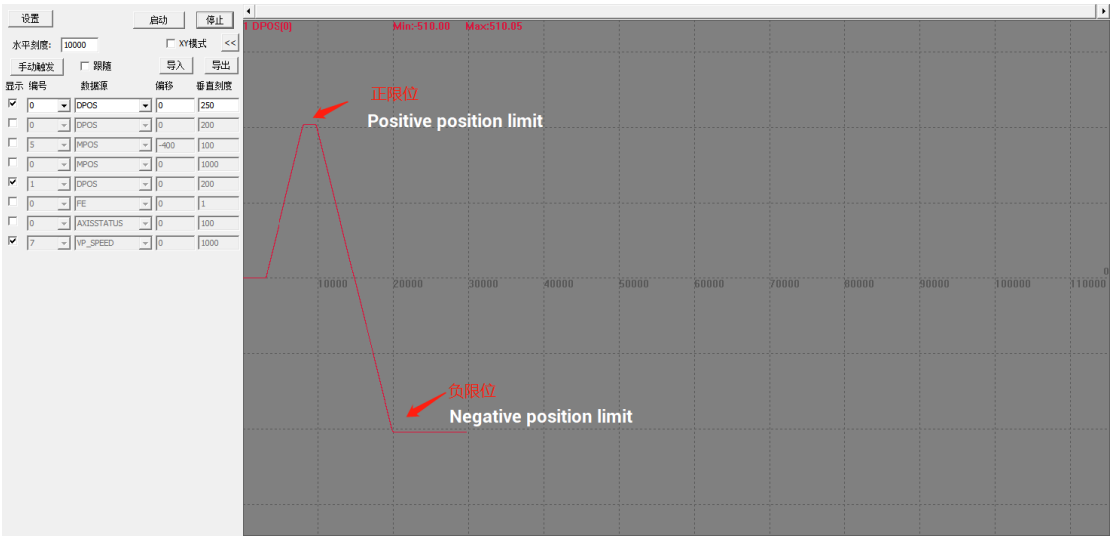
8.1.2.1. Positive & Negative Limit Setting

```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}

}

int _tmain(int argc, _TCHAR* argv[])
{
```

```
    char *ip_addr = (char *)"127.0.0.1";            //controller IP address
    ZMC_HANDLE handle = NULL;               //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        getchar();
        return -1;
    }
    printf("Success to connect controller!\n");

    ret =ZAux_Direct_SetAtype(handle,0,1);//set axis type as 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetAccel(handle,0,500);//set axis acceleration
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetDecel(handle,0,500);//set axis deceleration
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetDpos(handle,0,0);//set clear axis DPOS to 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully


    ret =ZAux_Direct_SetFsLimit(handle,0,500);//set forward soft position limit as
500
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetRsLimit(handle,0,-500);//set inverse soft position limit as -
500
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;//judge whether the
instruction is executed successfully

    float IDLE;
    int mode;
    int AxisStatus;
    ZAux_Trigger( handle);//trigger the oscilloscope
    while(1)
    {
        printf("please enter motion mode, 1-forward, -1-inverse: \n");
        scanf("%d",&mode);

        if (mode!=1 && mode!=-1 && mode!=0)
```

```
        {
            printf("please input motion mode error,%d\n",mode);
            Sleep(2000);
            return 0;
        }
        if (mode==0)
        {
            printf("exit\n");
            break;
        }
        ZAux_Direct_Single_Vmove(handle,0,mode);//axis 0 continuous motion

        while (1)//wait for axis 0 to complete the motion
        {
            Sleep(100);
            ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
            if (IDLE<0)break;
        }
        ZAux_Direct_GetAxisStatus(handle,0,&AxisStatus);
        printf("AxisStatus axis state: %s %d：,\n",AxisStatus>0 ? "axis alarm":"axis
normal",AxisStatus);
    }
    getchar();
    ret = ZAux_Close(handle);   //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

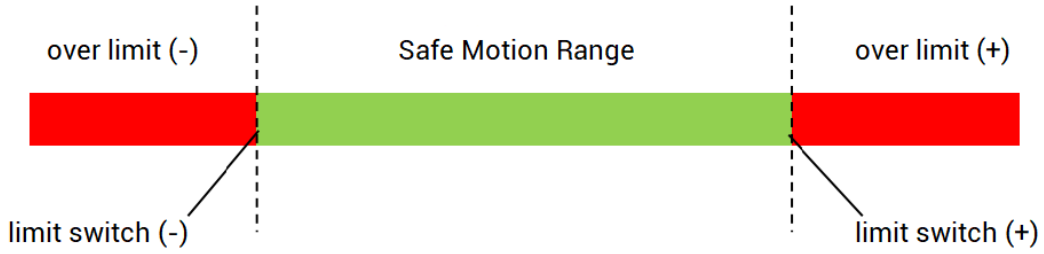
- Output waveform:



8.2. Axis Hardware Position Limit Function

8.2.1. Emphasis

The controller provides two inputs for each axis as travel control switches, and hardware sensors need to be installed. If the axis touches the limit switch during movement, it will immediately alarm and stop.



Note 1:

If the limit sends an alarm and reports an error AXISSTATUS: 10h (positive limit alarm), then the axis can only run in the negative direction when continuing to operate the axis.

If the limit sends an alarm and reports an error AXISSTATUS: 20h (negative limit alarm), then the axis can only run in the positive direction when continuing to operate the axis.

If the limit sends an alarm and reports an error AXISSTATUS: 30h (positive limit alarm + negative limit alarm), it is necessary to check whether the effective level of the photoelectric switch and other external equipment is opposite to the controller setting.

Note 2:

Remember to set the fast deceleration STDEC (**ZAux_Direct_SetFastDec**)

After the axis is configured with the positive and negative limits valid, if the axis moves beyond the position of the limit switch and the limit switch is triggered, the motion controller prohibits movement in the direction of the position limit, and the axis will stop in an emergency. At this time, because deceleration takes a certain amount of time, the actual axis position will exceed the limit by a certain distance. Assume that the SPEED (**ZAux_Direct_SetSpeed**) speed is v_0 when stopping, and the fast deceleration STDEC (**ZAux_Direct_SetFastDec**) is a. The calculation formula is: $v_t^2-v_0^2=2as$, then enter the data below: $0-100^2=2*(-1000)*s$, and get the overshoot distance s=5. From this, it

can be obtained that increasing STDEC (**ZAux_Direct_SetFastDec**) and reducing SPEED (**ZAux_Direct_SetSpeed**) can reduce overshoot purpose.

If the deceleration or the fast deceleration in not set, the axis will not stop when it hits the limit.

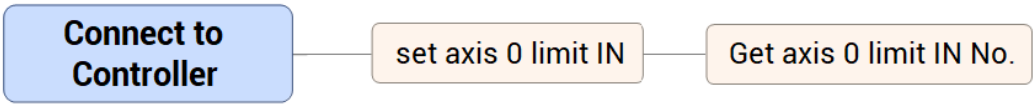
Note 3:

Remember that the positive and negative limits should not be set inversely.

When the positive limit is encountered, the axis can run in the negative direction normally. When the negative limit is encountered, the axis can run in the positive direction normally. If the positive and negative limit settings are opposite, the axis cannot move normally when it encounters the positive and negative limits.

8.2.2.Routine

8.2.2.1. Positive & Negative Hard Position Limit Setting




```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
```

```
    char *ip_addr = (char *)"127.0.0.1";          //controller IP address
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to connect controller!\n");

    int axislist[1] = { 0 };


    /*set the parameter*/
    ret = ZAux_Direct_SetAtype(handle, axislist[0], 1);       //set axis 0 type
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetAtype(handle, axislist[0], 1);   //set axis type of axis 0 as 1
    ZAux_Direct_SetUnits(handle, axislist[0], 100); //set pulse amount of axis 0 as
100
    ZAux_Direct_SetSpeed(handle, axislist[0], 200); //set axis 0 speed as 200units/s
    ZAux_Direct_SetAccel(handle, axislist[0], 2000); //set axis 0 acceleration as
2000units/s/s
    ZAux_Direct_SetDecel(handle, axislist[0], 2000); //set axis 0 deceleration as
2000units/s/s
    ZAux_Direct_SetFastDec(handle, axislist[0],3000);      //set fast deceleration of
axis 0
    float nvalue;
    ZAux_Direct_GetFastDec(handle, axislist[0],&nvalue);//obtain axis 0 fast
deceleration

    ret = ZAux_Direct_SetFwdIn(handle,axislist[0],1);       //map positive limit input
and correspond to input 1
    commandCheckHandler("ZAux_Direct_SetFwdIn", ret);

    ZAux_Direct_SetRevIn(handle,axislist[0],2);                  //map negative limit
input and correspond to input 2
    commandCheckHandler("ZAux_Direct_SetRevIn", ret);

    ret = ZAux_Direct_SetInvertIn(handle,1, 0);             //set origin signal, which
means low level is valid
    commandCheckHandler("ZAux_Direct_SetInvertIn", ret);

    ret = ZAux_Direct_SetInvertIn(handle,2, 0);             // set origin signal, which
means low level is valid
```

```
    commandCheckHandler("ZAux_Direct_SetInvertIn", ret);

    /*obtain parameters*/
    int GetValue;

    ret = ZAux_Direct_GetAtype(handle, axislist[0], &GetValue);            //set axis
type of axis 0
    printf("ZAux_Direct_GetAtype = %d \n", GetValue);

    ZAux_Direct_GetFwdIn(handle,axislist[0],&GetValue);        //get IN No. that sets
positive limit input
    printf("ZAux_Direct_GetFwdIn = %d \n", GetValue);

    ZAux_Direct_GetRevIn(handle,axislist[0],&GetValue);        //get IN No. that sets
negative limit input
    printf("ZAux_Direct_GetRevIn = %d \n", GetValue);

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

8.3. Axis Alarm Function

8.3.1. Emphasis

The controller provides a dedicated drive alarm signal input interface. This interface should be connected to the alarm output signal interface of the driver. After detecting the driver alarm signal, the motion controller will turn off the servo enable of the axis.

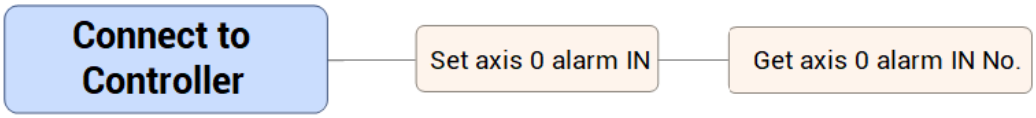
Note:

After setting the alarm input port, the ZMC controller is OFF by default, and the normally open signal uses **ZAux_Direct_SetInvertIn** to invert the level.

The ECI controller is ON by default, and the normally closed signal uses **ZAux_Direct_SetInvertIn** to invert the level.

8.3.2.Routine

8.3.2.1. Alarm IN Setting




```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;				//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to connect controller!\n");

	/*set axis related parameters*/
	ZAux_Direct_SetAlmIn(handle,0,0);//define axis 0 alarm signal into IN0
	commandCheckHandler("ZAux_Direct_SetAlmIn ",ret);
```

```
    ZAux_Direct_SetInvertIn(handle,0 ,0);//set input 0 low level as valid
    commandCheckHandler("ZAux_Direct_SetInvertIn ", ret);

    /*obtain axis related parameters */
    int GetValue;

    ZAux_Direct_GetAlmIn(handle, 0, &GetValue);//get axis 0's alarm signal input port
    printf("ZAux_Direct_GetAlmIn = %d\n ", GetValue);

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;// judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

Chapter IX Board Card Data Interaction

9.1.VR Register

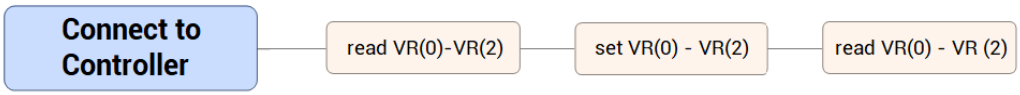
9.1.1.Emphasis

The controller has battery protection registers or ferroelectric memory VR, which can be used to save data when power is lost. The ONPOWER power-down interrupt can be used to trigger recording the position at power-down to VR.

- Only ZMC and xplc series have VR power-down storage, and ECI series have this register, but they do not actually have the function of power-down storage.
- VR registers are stored ferroelectrically and can be stored for up to 10 years or so. The key parameters of the machine are recommended to be stored in Flash.
- Some ZMC, xplc1/xplc2 series old products are battery storage, which can be used normally for several years without any problem, and they are rechargeable. If they are not turned on for a long time, they will be affected.

9.1.2.Routine

9.1.2.1. Use VR Register




```
// test1.cpp:
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//it is not 0, fail
```

```
    {
        printf("%s fail!return code is %d\n", command, ret);
    }

}
int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";          //controller IP address
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to connect controller!\n");
    float getVrfData[3] = {0.0};

    ret = ZAux_Direct_GetVrf(handle, 0, 3, getVrfData);//read VR(0)-VR(2)
    commandCheckHandler("ZAux_Direct_GetVrf", ret);

    printf("getVrfData[0] = %f\n",getVrfData[0]);
    printf("getVrfData[1] = %f\n", getVrfData[1]);
    printf("getVrfData[2] = %f\n", getVrfData[2]);

    float setVrfData[3] = {1.1,2.2,3.3};
    //write values in array into vr
    ret = ZAux_Direct_SetVrf(handle, 0, 3, setVrfData);//set VR(0)-VR(2)
    commandCheckHandler("ZAux_Direct_SetVrf", ret);

    //get values that was written just now
    ret = ZAux_Direct_GetVrf(handle, 0, 3, getVrfData);//read VR(0)-VR(2)
    commandCheckHandler("ZAux_Direct_GetVrf", ret);

    printf("getVrfData[0] = %f\n",getVrfData[0]);
    printf("getVrfData[1] = %f\n", getVrfData[1]);
    printf("getVrfData[2] = %f\n", getVrfData[2]);

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
```

```
    return 0;
}
```

9.2. Table

9.2.1. Emphasis

System array Table register

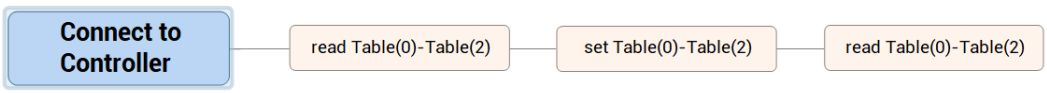
TABLE is a super large array that comes with the controller. The data type is 32-bit floating point (64-bit floating point for 4xx series and above), but data will not be saved when power off. When writing a program, the TABLE array does not need to be defined again, it can be used directly, and the index subscript starts from 0. The global array created by default in the system can be accessed by all programs. Data recording buffer, cam data table, pitch compensation table, manipulator parameters, etc. are all stored in table.

Some commands of ZBasic can directly read the value in TABLE as a parameter, such as cam motion, cam table motion, synchronous motion, forward and reverse kinematic, position comparison output and other commands.

The parameters sampled by the oscilloscope are also stored in TABLE. Therefore, pay attention to the allocation and use of multiple TABLE areas during development and application, and do not overlap with the data storage area sampled by the oscilloscope.

9.2.2. Routine

9.2.2.1. Use Table Register




```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
```

```
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//it is not 0, fail
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";          //controller IP address
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);  //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to connect controller!\n");
    float getTableValue[3] = {0.0};
    //obtain table value
    ret = ZAux_Direct_GetTable(handle, 0, 3, getTableValue);
    commandCheckHandler("ZAux_Direct_GetTable", ret);
    //print the values obtained from table
    printf("getTableValue[0] = %f\n",getTableValue[0]);
    printf("getTableValue[1] = %f\n", getTableValue[1]);
    printf("getTableValue[2] = %f\n", getTableValue[2]);

    float setTableValue[3] = {3.14,1.44,1.73};
    //set table
    ret = ZAux_Direct_SetTable(handle, 0, 3, setTableValue);
    commandCheckHandler("ZAux_Direct_SetTable", ret);

    //get table values
    ret = ZAux_Direct_GetTable(handle, 0, 3, getTableValue);
    commandCheckHandler("ZAux_Direct_GetTable", ret);

    //print the values obtained from table
    printf("getTableValue[0] = %f\n",getTableValue[0]);
    printf("getTableValue[1] = %f\n", getTableValue[1]);
    printf("getTableValue[2] = %f\n", getTableValue[2]);
```

```
    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

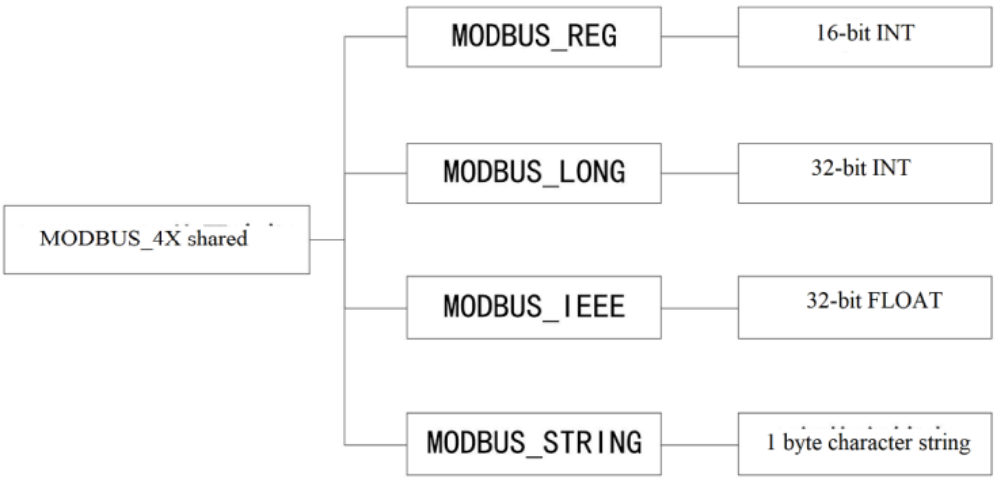
9.3.Modbus Register

9.3.1.Emphasis

The MODBUS register conforms to the MODBUS standard communication protocol, and it is divided into two types: bit register and word register.

Bit register: MODBUS_BIT, the touch screen is generally called MODBUS_0X, Boolean

Word registers: MODBUS_REG, MODBUS_LONG, MODBUS_IEEE, MODBUS_STRING, the touch screen is generally called MODBUS_4X, the type is as shown below.



The word registers in the controller occupy the same storage space, and one LONG occupies two REGs, so pay attention to the staggered word register numbers.

MODBUS_LONG(0) occupies two REG addresses MODBUS_REG (0) and MODBUS_REG(1).

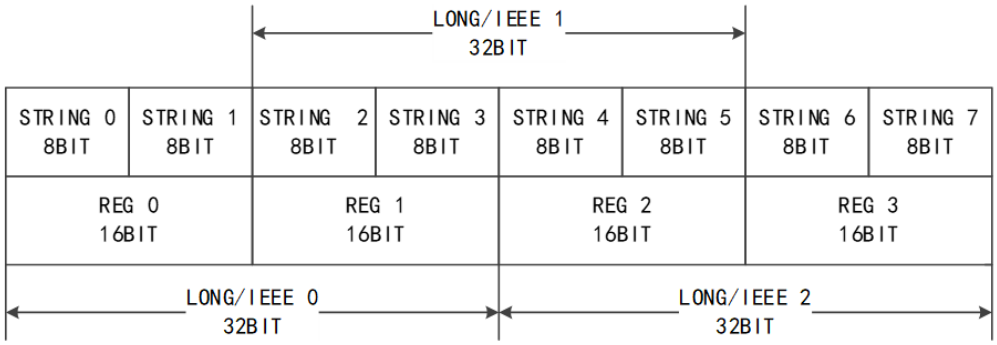
MODBUS_ LONG (1) occupies two REG addresses MODBUS_ REG (1) and MODBUS_ REG (2).

MODBUS_IEEE(0) occupies two REG addresses MODBUS_REG (0) and MODBUS_REG (1).

MODBUS_ IEEE (1) occupies two REG addresses MODBUS_ REG (1) and MODBUS_ REG (2).

Therefore, it should be noted that the addresses of MODBUS_reg, MODBUS_long and MODBUS_ieee cannot overlap.

4X Space Diagram:



9.3.2.Routine

9.3.2.1. Use Modbus Register

```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//it is not 0, fail
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";          //controller IP address
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to connect controller!\n");
    uint16 getValue[3];
    ZAux_Modbus_Get4x(handle, 0, 3, getValue);
    commandCheckHandler("ZAux_Modbus_Get4x", ret);
    printf("****************************Modbus_4X
setting****************************\n");
    //print obtained value
    printf("getValue[0] = %d\n",getValue[0]);
    printf("getValue[1] = %d\n", getValue[1]);
    printf("getValue[2] = %d\n", getValue[2]);

    uint16 setValue[3] = {4,5,6};
    //set modbus_4x data, starting from 0, there are 3 values
    //modbus_4x(0) is set to 4
    //modbus_4x(1) is set to 5
    //modbus_4x(2) is set to 6
    ret = ZAux_Modbus_Set4x(handle, 0, 3, setValue);
    commandCheckHandler("ZAux_Modbus_Set4x", ret);

    //obtain modbus_4x's value, here get above set values
    //getValue[0]'s value is the value in modbus_4x(0)
    //getValue[1] ]'s value is the value in modbus_4x(1)
    //getValue[2] ]'s value is the value in modbus_4x(2)
    ZAux_Modbus_Get4x(handle, 0, 3, getValue);
    commandCheckHandler("ZAux_Modbus_Get4x", ret);

    //print obtained values
    printf("getValue[0] = %d\n",getValue[0]);
    printf("getValue[1] = %d\n", getValue[1]);
    printf("getValue[2] = %d\n", getValue[2]);
    printf("****************************Modbus_String setting
****************************\n");
    char getStringData[13] = {0};
```

```
    //get "100-113" content from the modbus
    ZAux_Modbus_Get4x_String(handle,100,13,getStringData);
    commandCheckHandler("ZAux_Modbus_Get4x_String", 0);

    //print obtained character string
    printf("getStringData = %s\n",getStringData);

    char setStringData[13] = "hello world!";
    //write "hello world!" in modbus, the second parameter is the address that starts
to write.
    //the character string length is 13, so the third parameter value is 13
    ret = ZAux_Modbus_Set4x_String(handle, 100, 13, setStringData);
    commandCheckHandler("ZAux_Modbus_Set4x_String", ret);

    //get "hello world!" from modbus
    ZAux_Modbus_Get4x_String(handle,100,13,getStringData);
    commandCheckHandler("ZAux_Modbus_Get4x_String", 0);

    //print obtained character string
    printf("getStringData = %s\n",getStringData);
    uint8 GetValue[3];
    ret = ZAux_Modbus_Get0x( handle, 0, 3, GetValue);
    commandCheckHandler("ZAux_Modbus_Get0x" ,ret);

    printf("**************************Modbus_0x setting
**************************\n");

    printf("GetValue[0] = %d\n", GetValue[0]);
    printf("GetValue[1] = %d\n", GetValue[1]);
    printf("GetValue[2] = %d\n", GetValue[2]);
    uint8 SetValue[3] = {1, 2, 3};

    ret = ZAux_Modbus_Set0x( handle, 0, 3, SetValue);
    commandCheckHandler("ZAux_Modbus_Set0x",ret);

    ret = ZAux_Modbus_Get0x( handle, 0, 3, GetValue);
    commandCheckHandler("ZAux_Modbus_Get0x" ,ret);

    printf("GetValue[0] = %d\n", GetValue[0]);
    printf("GetValue[1] = %d\n", GetValue[1]);
    printf("GetValue[2] = %d\n", GetValue[2]);

    Sleep(2000);
    ret = ZAux_Close(handle);   //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
```

```
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

9.4.Flash/File Read & Write

9.4.1.Emphasis

There are U disk interface and flash on the controller.

The user can communicate with the controller through the U disk. The file format of the U disk and the format/order of reading and writing files must be consistent.

Store Flash block (enter online command ? FLASH_SECTES parameter in ZDevelop can check), which can be used to save power-off data.

- Flash has a write life limit and cannot be erased and written infinitely. It is recommended to write data that is frequently rewritten into VR.

USB interface:

Standard U-disk interface is used for inserting U-disk devices and supports data interaction with U-disk. Different types of controllers have the same method of using the U disk interface. Just insert the U disk into the UDISK port on the controller. When the controller is powered on and a U disk is inserted, the U disk indicator light will be on.

PIN	Name	Description
1	V	Internal +5V power
2	D-	Differential data D-
3	D+	Differential data D+
4	GND	Internal power ground

Before operating the U disk, first use the U_STATE command to judge the state of the U disk to ensure that the U disk can communicate successfully, and then use the U disk related instructions to operate. (It is recommended to use a 2.0 U disk)

The U disk interface mainly has the following three aspects of use:

- program upgrade

Through the USB port, download the packaged zar encrypted program package,

which is convenient for customers to update the system program.

- file loaded three times

Use the U disk command to load the three files saved in the U disk for execution.

- U disk and register data interaction

U disk supports reading and writing variables and arrays.

➢ Some U disk formats do not be supported, U disk read file system supports fat32 and fat16, does not support ntfs, exFAT formats.

FLASH data copy: the data stored in FLASH in multiple controllers can be transferred to each other through U disk.

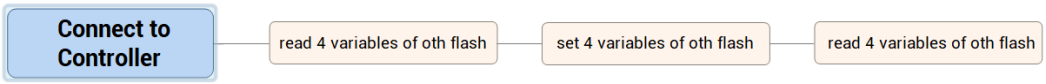
The VR register, TABLE register and the data in the U disk are transferred to each other.

The file type for reading and writing is SD(filenum).BIN or SD(filenum).CSV, and the types of files that can be operated by different commands are different.

9.4.2.Routine

9.4.2.1. Flash Writing & Reading

The following example shows an example of flash reading and writing, first write data into the 0th flash of the controller, and then read the data in the flash.



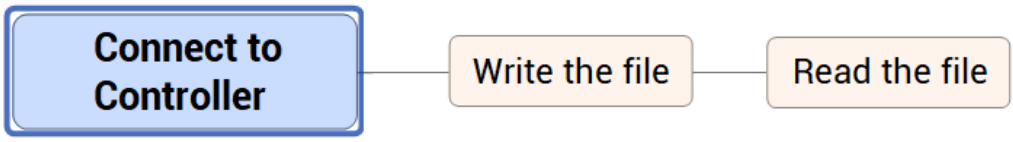

```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
```

```
    if (ret)//it is not 0, fail
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";          //controller IP address
    ZMC_HANDLE handle = NULL;                     //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to connect controller!\n");
    float readValues[4] = {0};
    uint32 readValuesNum = 0;
    //read flash data from block 0
    ret = ZAux_FlashReadf(handle, 0,4,readValues,&readValuesNum);
    commandCheckHandler("ZAux_FlashReadf", ret);

    //print obtained data
    printf("readValues[0] = %f\n",readValues[0]);
    printf("readValues[1] = %f\n", readValues[1]);
    printf("readValues[2] = %f\n", readValues[2]);
    printf("readValues[3] = %f\n", readValues[3]);

    float writeValues[4] = { 1.0,2.0,3.0,4.0 };

    //write data into the block 0 of flash
    ret = ZAux_FlashWritef(handle, 0, 4, writeValues);
    commandCheckHandler("ZAux_FlashWritef", ret);

    //read flash data from block 0
    ret = ZAux_FlashReadf(handle, 0,4,readValues,&readValuesNum);
    commandCheckHandler("ZAux_FlashReadf", ret);

    //print obtained data
    printf("readValues[0] = %f\n",readValues[0]);
    printf("readValues[1] = %f\n", readValues[1]);
    printf("readValues[2] = %f\n", readValues[2]);
    printf("readValues[3] = %f\n", readValues[3]);
```

```
    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

9.4.2.2. U Disk File Writing & Reading

Write the data to the file, the format of the file must be consistent with the file format of the U disk, and then read the written data.




```
// test1.cpp: define the entry point of control panel application program //


#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//it is not 0, fail
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";          //controller IP address
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
```

```
            handle = NULL;
            Sleep(2000);
            return -1;
        }
        printf("Success to connect controller!\n");


        //save data into test.txt file, it will generate in same program path
        float writepvarlist[10] = {1.1,2.2,3.3,4.4,5.5,6.6,7.7,8.8,9.9,10.2};
        ret = ZAux_WriteUFile("test123.txt",writepvarlist,10);
        commandCheckHandler("ZAux_WriteUFile", ret);

        float readpvarlist[10] = {0};
        int readnum = 0;
        //read data that has been saved into file just now to floating array
        ret = ZAux_ReadUFile("test123.txt", readpvarlist, &readnum);
        commandCheckHandler("ZAux_ReadUFile", ret);

        //print read results:
        printf("readpvarlist[0] = %f\n",readpvarlist[0]);
        printf("readpvarlist[1] = %f\n", readpvarlist[1]);
        printf("readpvarlist[2] = %f\n", readpvarlist[2]);
        printf("readpvarlist[3] = %f\n", readpvarlist[3]);
        printf("readpvarlist[4] = %f\n", readpvarlist[4]);
        printf("readpvarlist[5] = %f\n", readpvarlist[5]);
        printf("readpvarlist[6] = %f\n", readpvarlist[6]);
        printf("readpvarlist[7] = %f\n", readpvarlist[7]);
        printf("readpvarlist[8] = %f\n", readpvarlist[8]);
        printf("readpvarlist[9] = %f\n", readpvarlist[9]);

        Sleep(2000);
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
        printf("connection closed!\n");
        handle = NULL;
        return 0;
}
```

Chapter X Bus

10.1. Bus Initialization

10.1.1. Emphasis

10.1.1.1. EtherCAT Protocol Description

EtherCAT is an industrial Ethernet technology with high performance, low cost, simple application and flexible topology. It can be used as an ultra-high-speed 1/0 network at the industrial field level, using standard Ethernet physical layer, and transmission media twisted pair or optical fiber (100Base-TX or 100Base-FX). EtherCAT system consists of master station and slave station.

Bus Card Ethercat Communication Specifications:

Item		Specification
Communication protocol		IEC61800-7 CiA 402 Drive Profile
Application Layer	SDO	SDO request, SDO response
	PDO	changeable PDO mapping
	CiA 402	Synchronous Cylce Position Mode (CSP) Synchronous Cycle Velocity Mode (CSV) Synchronous Cyclic Torque Mode (CST) HM (homing) TouchProbe (probe)
Physical Layer	Transfer protocol	100BASE-TX(IEEE802.3)
	Max distance	100M
	Interface	RJ45(EtherCat 口)

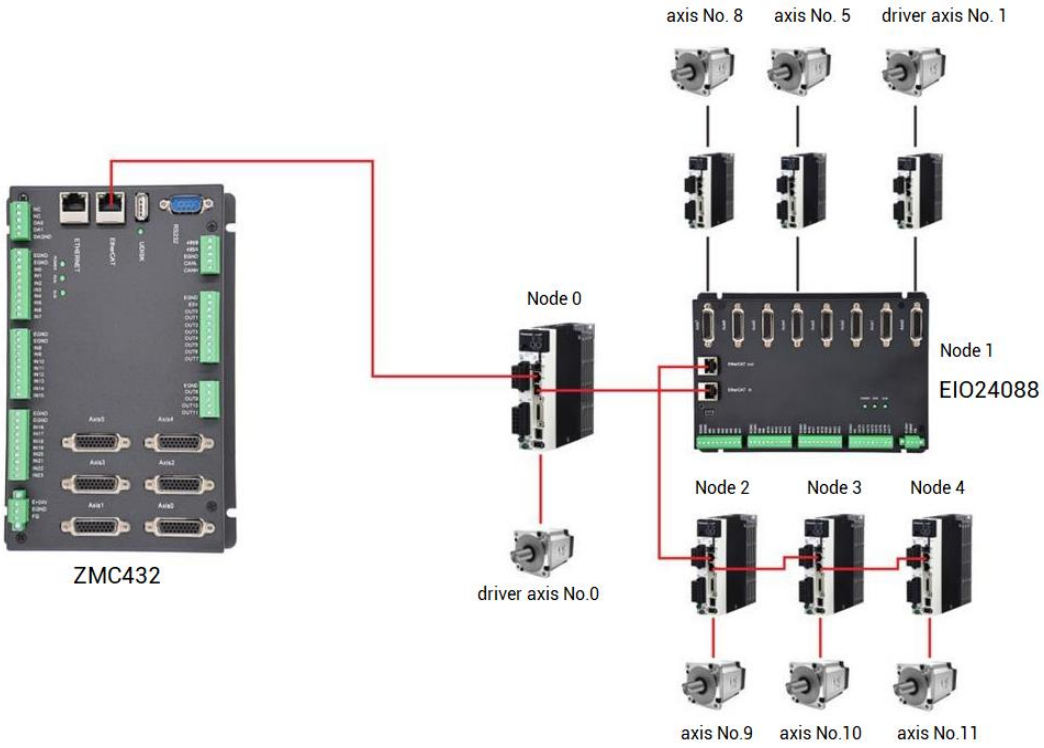
10.1.1.2. EtherCAT Bus Interface

The EtherCAT bus interface uses the same RJ45 standard network cable interface as the EtherNet network port, which is used to connect the EtherCAT bus driver and expansion module. Some controller models do not support the EtherCAT bus.

After the EtherCAT bus driver is connected to the driver, it needs to perform the bus initialization operation before use, and use the command to map the axis No. of the

driver, enable the driver, etc. For details, refer to the bus initialization reference program.

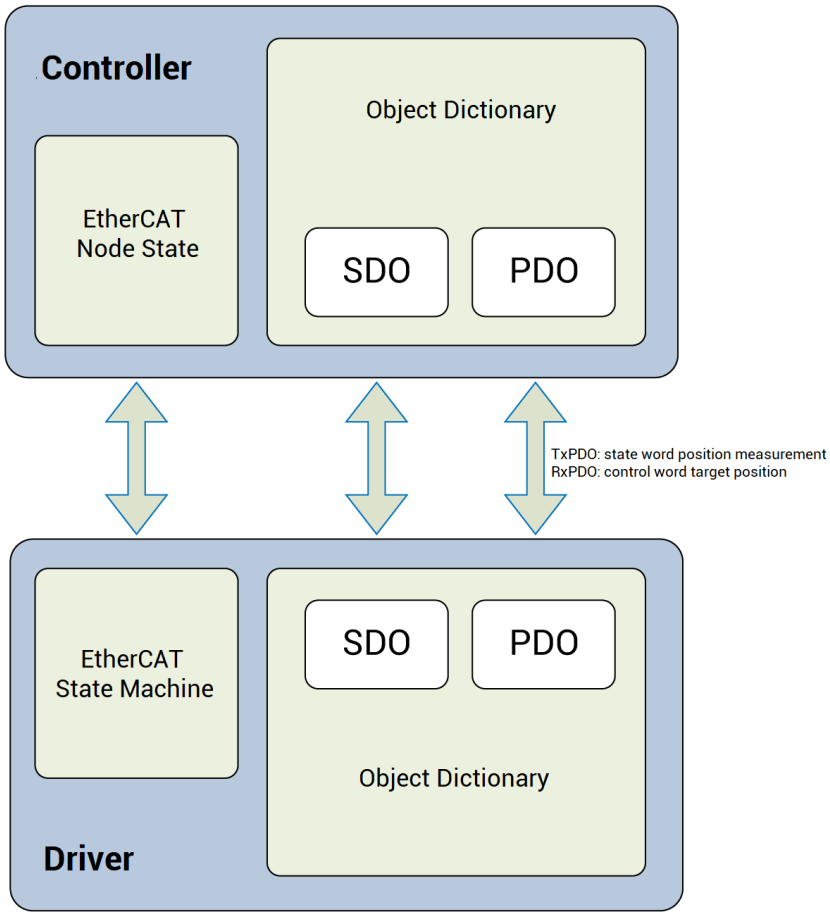
Mixed use of bus axis and pulse axis is supported. The EtherCAT bus wiring reference diagram is as follows.



Basic concepts:

- Slot number: the slot number is the slot number of the EtherCAT network port on the controller. There is only one EtherCat slot by default, and the default slot number is 0.
- Node number: it is arranged according to the wiring sequence of EtherCAT devices, the number starts from 0 (to the number of devices -1). Configure the axis number and IO number by judging the number of node motors and the number of IOs.
- Axis number: axis No. for the controller, starting from 0 (to the total number of axes connected to -1), the axis number can be mapped to any connected drive device through the AXIS_ADDRESS command.
- Bus cycle time: 1ms by default. Controllers above four series support modification of the bus synchronization cycle time 500us-4ms.

10.1.1.3. EtherCAT Communication Structure



In the structure diagram, the PDO process data object contains the real-time data during the operation of the servo drive, and is accessed periodically for read and write. For SDO mailbox communication, access and modify some communication parameter objects and PDO process data objects non-periodically.

10.1.1.4. Bus Initialization

(1) The difference between pulse controller and bus controller

Bus control: Bus control is to send command planning position values through the network cable. The bus controller needs an initialization program to enable the motor and then control the motor.

Pulse control: Pulse control is to directly open the corresponding output port to

enable the motor, and then send pulses to control the motor.

(2) Bus initialization

The process of bus initialization (this process is the basic file initialization process):

- Scan the nodes, scan all nodes on the EtherCat network.
- For EtherCat nodes, perform PDO preset mapping, such as mapping the target position 607ah to the planned position of the axis, and mapping the feedback position 6064h to the feedback position of the axis, etc.
- Start the bus, switch the drive state machine, "initialization—>pre-running—>safety running—>running"

(3) Bus homing

Some Ethercat drives support the drive itself to return to zero, and provide a variety of homing methods. When the driver returns to zero, the origin and the limit sensor switch need to be directly connected to the corresponding IO port of the driver (when returning to zero in the controller mode, it needs to be connected to the IO port of the controller). The homing action can be completed by controlling the master station to send the corresponding homing command.

10.1.1.5. Bus Mode

There are three commonly used control modes for the EtherCAT bus, namely CSP cycle position mode, CSV cycle velocity mode, and CST cycle torque mode.

The setting of CSP, CSV, and CST mode needs to set the PDO list of the driver in advance, and only when the PDO contains the corresponding data dictionary can it be switched to this mode. The built-in data dictionary of the drive's default PDO list needs to be determined by checking the drive manual.

1. When the driver PDO list contains data dictionary 607Ah, the axis type can be set to 65, CSP periodical position control mode, at this time use the position movement command to control the motor movement, and set the movement speed.

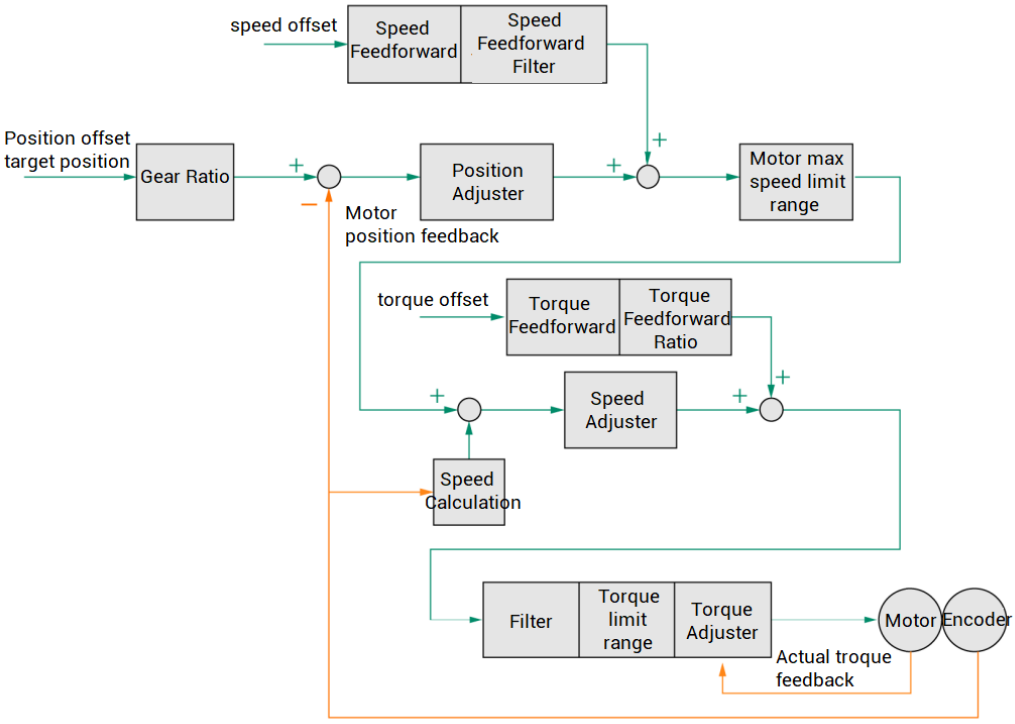
2. When the PDO list of the drive contains the data dictionary 60FFh, the axis type can be set to 66, and the CSV periodical velocity control mode. At this time, use the dedicated speed command to control the motor to run at the speed of the set value, and

the position motion command cannot be used.

3. When the drive PDO list contains the data dictionary 6071h, the axis type can be set to 67, CST cycle torque control mode, at this time, use the dedicated torque command to control the motor to run at the set value torque, and the position motion command cannot be used.

CSP mode:

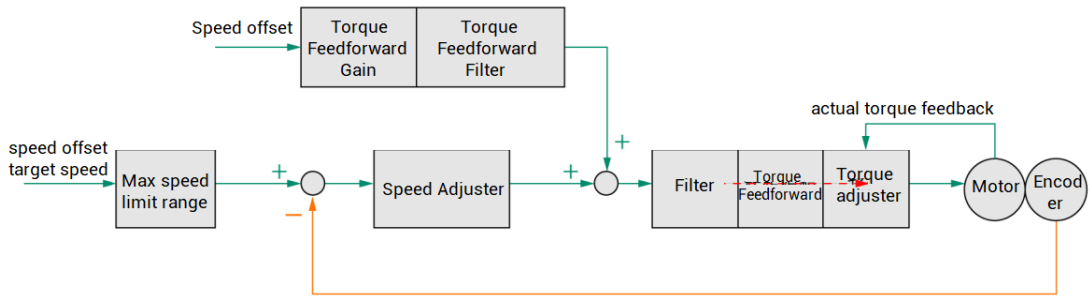
In the periodic synchronous position mode, the upper controller completes the position command planning, and then periodically sends the planned target position to the servo driver, and the position, speed, and torque control are completed by the servo driver.



CSP Mode

CSV mode:

In the periodic synchronous speed mode, the host controller periodically and synchronously sends the calculated target speed to the servo drive, and the speed and torque adjustments are performed internally by the servo.

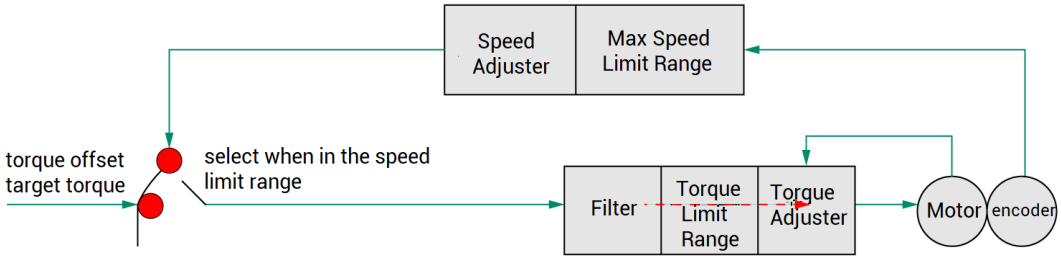


Speed control needs to see the specific unit of the drive.

Modify the corresponding ATYPE type, and then modify the corresponding profile (speed control is set to 20), and then modify the value of dac through online commands or programs.

CST mode:

In the periodic synchronous torque mode, the host controller periodically and synchronously sends the calculated target torque to the servo driver, and the torque adjustment is performed internally by the servo.



EtherCAT torque control needs to set the corresponding axis type (67), and correspondingly modify the profile to be more than 30.

Then modify the value of dac to control the rotation of the motor through online commands or programs (adjusting the value of dac is not easy to span too large)

RTEX torque mode:

Corresponding mode is modified through modifying corresponding axis type, and then modify the corresponding profile. For details, please refer to the ZBasic manual.

Send the value of dac to control the rotation of the motor through the online command, or control the rotation of the motor through the program. The unit of dac is one thousandth of the torque in torque control, and dac=1000 means 100%

Command: dac	Description
Size	With symbol 32bit

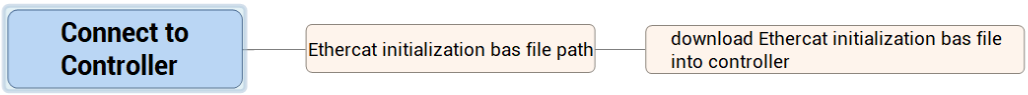
Unit	0.1%
Set range	-motor max torque ~ max motor torque * max torque limit = 100 x Pr9.07 / (Pr9.06 x ✓2)

If the dac (torque) is too small, the motor cannot overcome the friction and cannot rotate.

10.1.2. Routine

10.1.2.1. EtherCAT Bus Initialization (bas method)

(If you are not familiar with our company's Zbasic language, you can use the basic file in the CD-ROM provided by our company for free, please contact us)

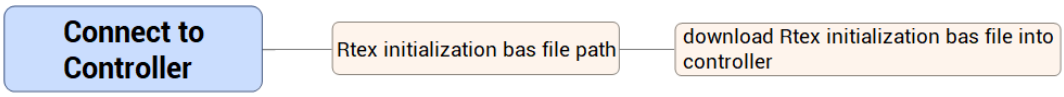



```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"192.168.0.11";		//controller IP address
	ZMC_HANDLE handle = NULL;		//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
```

```
            handle = NULL;
            Sleep(2000);
            return -1;
        }
        printf("Success to connect controller!\n");

        const char * filename= ".\\ECAT init.bas";

        //download the program into ROM, 0-RAM 1-ROM
        ret = ZAux_BasDown(handle,filename,0);//download initialization bas file into
controller to initialize

        commandCheckHandler("ZAux_BasDown",ret);

        Sleep(2000);
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
        printf("connection closed!\n");
        handle = NULL;
        return 0;
}
```

10.1.2.2. Rtex Bus Initialization (bas method)

(If you are not familiar with our company's Zbasic language, you can use the basic file in the CD-ROM provided by our company for free, please contact us)




```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
```

```
    if (ret)//it is not 0, fail
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.11";          //controller IP address
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to connect controller!\n");
    //under program path
    const char * filename= ".\\RTEX init.bas";

    //download the program into ROM, 0-RAM 1-ROM
    ret = ZAux_BasDown(handle,filename,0);//download initialization bas file into
controller to initialize
    commandCheckHandler("ZAux_BasDown",ret);

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

10.1.2.3. EtherCAT Bus Axis Enable

Before enabling the Ethrcat bus axis, the initial configuration of the bus axis is required. There are two methods for initial configuration of the EthrCat bus:

- Method 1: use the Zbasic language of Zmotion to write a bas file for initial configuration of the EthrCat bus, and download the Bas file to the controller

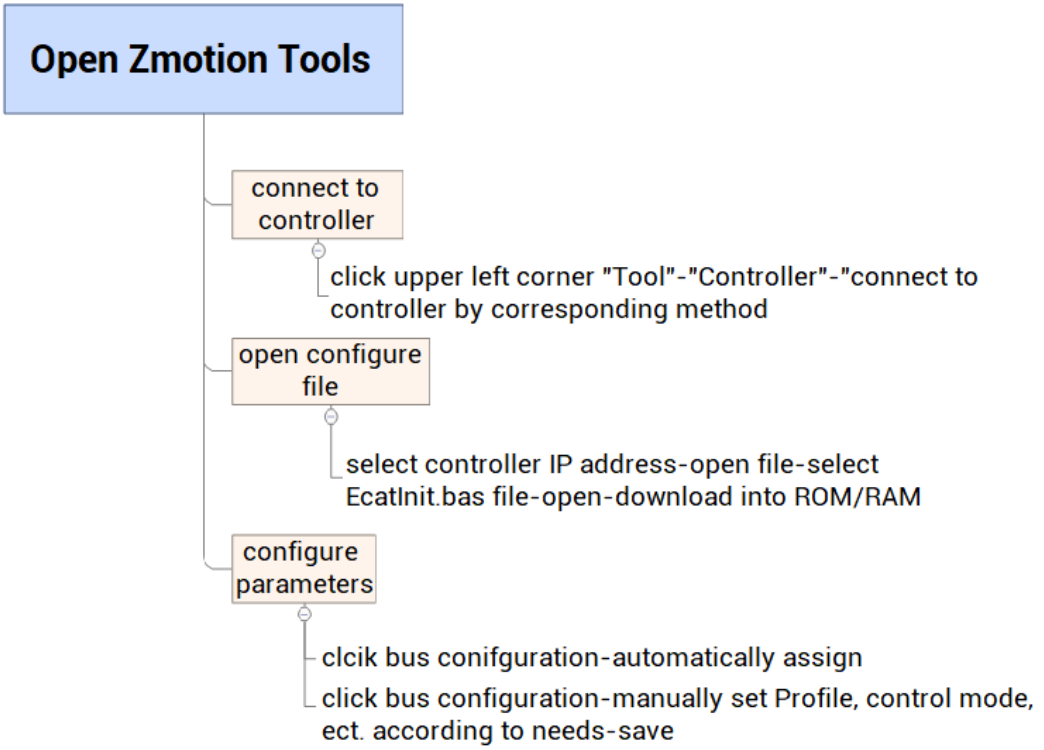
through the ZAux_BasDown function (if not familiar with our company's Zbasic language, please contact us) to initialize the bus configuration.

- Method 2: Initialize the configuration of the bus by using the Zmotion Tools gadget provided by our company for free.

This routine uses method 2 to initialize the configuration of the bus through the Zmotion Tools tool.

Scenario: the current controller is connected to an Ethercat bus motor, and the motor needs to be enabled.

Zmotion Tools tool configuration flow chart:




```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//it is not 0, fail
```

```
    {
        printf("%s fail!return code is %d\n", command, ret);
    }

}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.11";          //controller IP address
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to connect controller!\n");

    //ZAux_BusCmd_InitBus(handle);//bus initialization (it is valid in controller that has configured bus parameters through "Zmotion tools")

     int GetValue;
     ret = ZAux_BusCmd_GetInitStatus(handle, &GetValue); //obtain bus initialization state, 0 means failure, 1 means success (it is valid in controller that has configured bus parameters through "Zmotion tools")

     commandCheckHandler("ZAux_BusCmd_GetInitStatus", ret);
     printf("Bus initial state= %d \n", GetValue);
     int AxisEnable;
     ret = ZAux_Direct_GetAxisEnable(handle, 0, &AxisEnable);//set axis enable, 0 means off, 1 means on
     commandCheckHandler("ZAux_BusCmd_GetInitStatus", ret);
     printf("axis 0 bus init enable state: %d \n", AxisEnable);

     if (AxisEnable<1)
     {
         ret = ZAux_Direct_SetAxisEnable(handle, 0, 1);//set axis enable, 0 means off, 1 means on
         commandCheckHandler("ZAux_Direct_SetAxisEnable", ret);
         printf("open enable\n");
     }
     else
     {
        ret = ZAux_Direct_SetAxisEnable(handle, 0, 0);//set axis enable, 0 means off,
```

```
1 means on
            commandCheckHandler("ZAux_Direct_SetAxisEnable", ret);
            printf("close enable\n");
        }

        Sleep(2000);
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
        printf("connection closed!\n");
        handle = NULL;
        return 0;
}
```

10.2. SDO/PDO Writing & Reading

10.2.1. Emphasis

The data exchange of SDO uses Mailbox communication. So please note that the data refresh time of SDO becomes unstable.

The master side reads and writes data in the records in the object dictionary, and can set objects and monitor various statuses of the slaves.

Note:

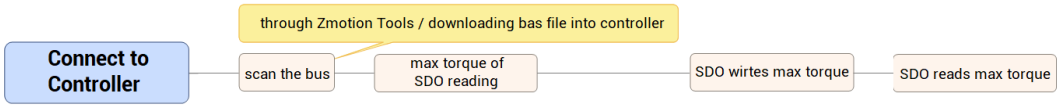
It takes time to respond to the read and write operations of SDO.

ZAux_BusCmd_SDOWrite and ZAux_BusCmd_SDORead bus commands only can be used for EtherCAT, write SDO to the slot number through the device number.

10.2.2. Routine

10.2.2.1. EtherCAT Bus SDO Writing & Reading

This routine is operated after the bus is initialized and a new project is created.




```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"192.168.0.11";		//controller IP address
	ZMC_HANDLE handle = NULL;		//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to connect controller!\n");
	int slot = 0;
	int node = 0;
	int index = 0x6041;
	int subindex=0;
	int type = 6;
	int value;
	int setValue = 8;

	/*read the state word on slot 0 */
	ret = ZAux_BusCmd_SDORead(handle,slot,node, index, subindex,type,&value);
	commandCheckHandler("ZAux_BusCmd_SDORead",ret);
	printf("Value=%d\n",value);
```

```
    /*set the control mode on slot 0 */
    index = 0x6060;
    ret = ZAux_BusCmd_SDOWrite(handle,slot,node, index,subindex,2, setValue);
    commandCheckHandler("ZAux_BusCmd_SDOWrite", ret);

    /*read the control mode on slot 0 */
    ret = ZAux_BusCmd_SDORead(handle,slot,node, index, subindex,type,&value);
    commandCheckHandler("ZAux_BusCmd_SDORead",ret);
    printf("Value=%d\n",value);

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the conntion
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

10.2.2.2. Rtex Parameters Related Information

This routine is operated after the bus is initialized and a new project is created.

```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//it is not 0, fail
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.11";          //controller IP address
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
```

```
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to connect controller!\n");
    float GetValue;
    int ipara = 7 * 256 + 11;
    /*set the value of Pr7.11*/
    ret = ZAux_BusCmd_RtexWrite(handle,0,ipara,500);

    /*read the value of Pr7.11*/
    ret = ZAux_BusCmd_RtexRead(handle,0, ipara,&GetValue);
    commandCheckHandler("ZAux_BusCmd_RtexRead", ret);
    printf("GetValue=%f\n", GetValue);

    Sleep(2000);
    ret = ZAux_Close(handle);   //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

10.3. Bus Torque

10.3.1. Emphasis

Motor torque refers to the size of the rotating force. But the torque of the motor is proportional to the strength of the rotating magnetic field and the current in the rotor cage, and is proportional to the square of the power supply voltage, so the torque is determined by the factors of current and voltage.

Motor torque and calculation: T=9550 * P / n.

P is the rated (output) power of the motor in kilowatts (KW).

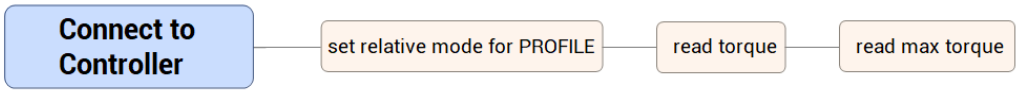
The denominator is the rated speed n and the unit is revolutions per minute (r/min)

P and n can be found directly from the motor nameplate. Since P and n are the rated values of the motor, T is the rated torque of the motor. Generally speaking, the force that the motor drives the machine to rotate is the torque of the motor. The torque of the motor = the force of the pulley to drag the belt x the radius of the pulley.

Note: the settings can only be read after setting the correct ATYPE and drive PDO.

10.3.2. Routine

10.3.2.1. Get Current Torque of Current Bus Drive




```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"192.168.0.11";		//controller IP address
	ZMC_HANDLE handle = NULL;			//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
```

```
    }
    printf("Success to connect controller!\n");

    int gValue;
    ret=ZAux_BusCmd_GetDriveTorque(handle,0,&gValue);//get the current torque,
PROFILE needs to be configured corresponding mode.
    commandCheckHandler("ZAux_BusCmd_GetDriveTorque", ret) ;//judge whether
the instruction is executed successfully
    printf("axis 0 current torque: %d\n",gValue);

    ret=ZAux_BusCmd_GetMaxDriveTorque(handle,0,&gValue);//get the current
torque, PROFILE needs to be configured corresponding mode.
    commandCheckHandler("ZAux_BusCmd_GetDriveTorque", ret) ;//judge whether
the instruction is executed successfully
    printf("axis 0 current torque: %d\n",gValue);

    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

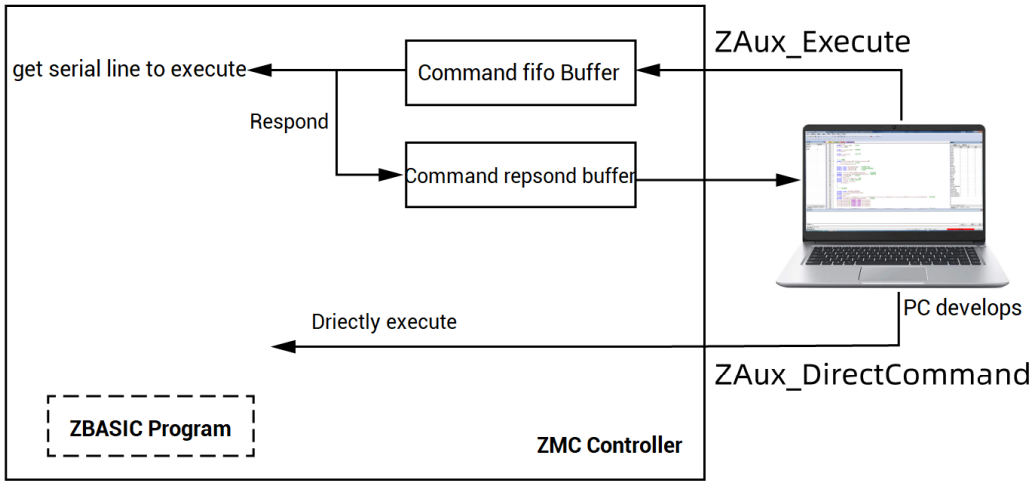
Chapter XI Other Commands

11.1. Online Commands

11.1.1. Emphasis

ZAux_Execute or ZAux_DirectCommand can encapsulate basic commands. If you use unencapsulated commands or want to encapsulate your own functions, you can send them through ZAux_Execute or ZAux_DirectCommand, or refer to existing codes to modify and add corresponding functions.

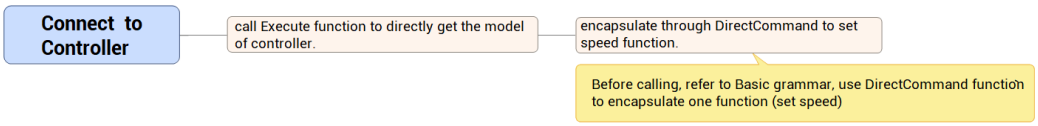
There are two ways to send string commands, buffered and direct. Specifically as shown in the figure:



11.1.2. Routine

11.1.2.1. Use Online Command Function

This routine mainly demonstrates how to use the Execute function to directly obtain the model of the controller, and how to use the DirectCommand function to encapsulate an example of setting the speed function command.




```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail

	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}
int  Test_Direct_SetSpeed(ZMC_HANDLE handle, int iaxis, float fValue)
{
	char cmdbuff[2048];
	char cmdbuffAck[2048];
	if (iaxis> MAX_AXIS_AUX)
	{
		return ERR_AUX_PARAERR;
	}
	sprintf(cmdbuff,"SPEED(%d)=%f",iaxis,fValue);//generate corresponding
command's character string

	ZAux_DirectCommand(handle,cmdbuff,cmdbuffAck,2048);
}
/*parameters:
	mode		25, 26, 2D comparison mode
	Opnum		corresponding output
	Opstate		output state of the comparison point.
	maxerr		the pulse deviation of comparison position's each axis left and right,
when entered the deviation range, start to compare.
	num			saved comparison points in TABLE
	tablepos	TABLE No. of the first comparison point's coordinate

	when it is used with hwtimer, hwtimer parameter can be adjusted dynamically.
	ModePara1: pulse time
	ModePara2: the number of pulses
	ModePara3: pulse period
	*/
int32 __stdcall ZAux_Direct_HwPswitch2_2D(ZMC_HANDLE handle, int Axisnum,int
```

```
Mode,int Opnum , int Opstate,int maxerr,int num, int tablepos, float ModePara1=0,
float ModePara2=0,float ModePara3=0)
{
    if(0 > Axisnum || Axisnum > MAX_AXIS_AUX)
    {
        return  ERR_AUX_PARAERR;
    }
    char  cmdbuff[2048];
    char  cmdbuffAck[2048];
    //generate the command
    switch(Mode)
    {
    case 25:
        sprintf(cmdbuff, "HW_PSWITCH2(%d,%d,%d,%d,%d,%d) AXIS(%d)", Mode,
Opnum, Opstate, maxerr,num,tablepos,Axisnum);
        break;
    case 26:
        sprintf(cmdbuff, "HW_PSWITCH2(%d,%d,%d,%d,%d,%d,%f,%f,%f) AXIS(%d)",
Mode, Opnum, Opstate,
maxerr,num,tablepos,ModePara1,ModePara2,ModePara3,Axisnum);
        break;

    }
    //call the command to execute the function

    return ZAux_Execute(handle, cmdbuff, cmdbuffAck,2048);
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";          //controller IP address
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);  //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to connect controller!\n");

    const char * command="?control";  //send the command that obtains controller
model
    char psRespons[20];        //the array is used to receive the controller model
    int length=20;      //controller model length, which must be larger than received
character string
    float GetValue1;
    ret = ZAux_Execute(handle, command, psRespons, length);      //obtain controller
model through execute command directly
```

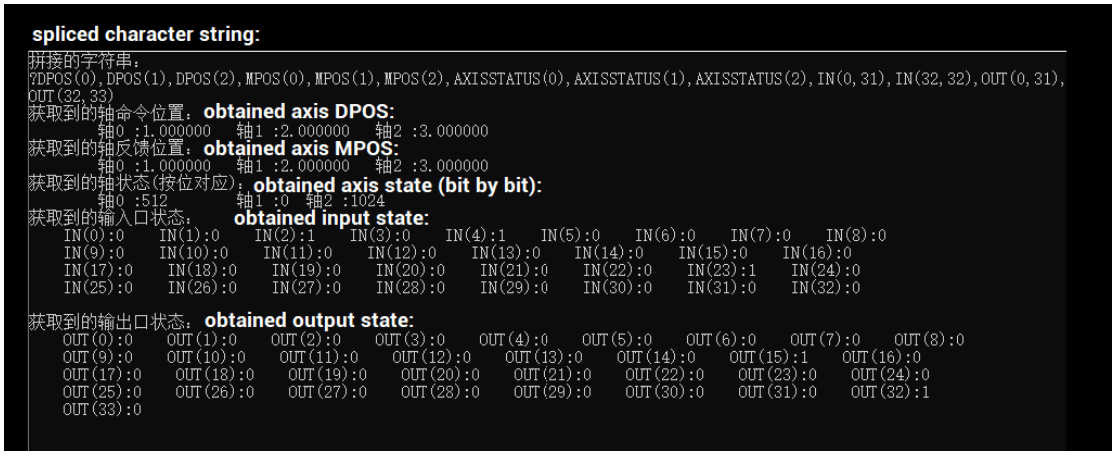
```
    commandCheckHandler("ZAux_Execute",ret);

    printf("psRespons = %s\n", psRespons); //print returned character string
    printf("length = %d\n",length);

    ret=ZAux_Direct_GetSpeed(handle,0,&GetValue1);//obtain speed value before
setting the speed
    commandCheckHandler("ZAux_Direct_GetSpeed",ret);
    printf("the speed value before setting the speed = %f\n",GetValue1);

    ret=Test_Direct_SetSpeed(handle,0,1231);//use own encapsulated speed
command to set the speed of axis 0
    commandCheckHandler("ZAux_Direct_GetSpeed",ret);

    ret=ZAux_Direct_GetSpeed(handle,0,&GetValue1);//obtain the speed value after
setting the speed
    commandCheckHandler("ZAux_Direct_GetSpeed",ret);
    printf("speed value after setting the speed = %f\n",GetValue1);
    Sleep(2000);
    ret = ZAux_Close(handle);    //close the connection

    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

11.1.2.2. Use Online Command Function 2

If the user wants to obtain a variety of data, such as the command position (DPOS) of the axis, the feedback position (MPOS) of the axis, the IO points on the board, etc., these different data are obtained through a variety of separate and independent functions. Such accumulation will lead to reach the number of write and read limits, then will cause the program to freeze. In order to improve the performance of a high-level program, it is often possible to define a function of its own, and quickly transfer data to the high-level program through a function, instead of obtaining different types of data through multiple cycles. For example:

Suppose there is a simple three-axis platform, and it is necessary to read the command position of axis 0, axis 1 and axis 2, the feedback position, and the input port

0-input port 32, output port 0-output port 33, and the state of the three axes. The procedure for obtaining data is as follows:




```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}

}

/*************************************************************
Description:	//customized function that obtains data directly
Input:		//handle		card link
			iaxisNum		axis total numbers
			iaxislist		axis No. list
			fDposlist		output command position value (DPOS)
			fMposlist		output feedback position value (MPOS)
			iAxisstatuslist	output axis state position value, bit by bit
			startIn		starting IN No. to be obtained
			endIn		end IN No. to be obtained
			iIn		output IN state, bit by bit
			startOut		starting OUT No. to be obtained
			endOut		end OUT No. to be obtained
			iOut		output OUT state, bit by bit
Output:		//
Return:		//error codes
*************************************************************/
int  Demo_Direct_MyGetData(ZMC_HANDLE handle,int iaxisNum, int* iaxislist, float*
fDposlist,float* fMposlist,int32* iAxisstatuslist,int startIn , int endIn,int *iIn,int startOut ,
int endOut,int *iOut)
```

```
{
    char cmdbuff[2048];
    char  tempbuff[2048];
    char cmdbuffAck[20480];

    //if transferred address is blank, exit

    if(NULL == iaxislist || NULL == fDposlist || NULL == fMposlist ||  NULL ==
iAxisstatuslist || NULL == iIn || NULL == iOut)
    {
        return  ERR_AUX_PARAERR;
    }
    //if transferred end No. is smaller than starting No., exit
    if ((endIn<startIn)  ||  (endOut<startOut))
    {
        return  ERR_AUX_PARAERR;
    }

    int ret=0;
    int i;
    //generate the command
    sprintf(cmdbuff, "?");
    //splice DPOS
    for (i=0;i<iaxisNum;i++)
    {
        sprintf(tempbuff,"DPOS(%d),",iaxislist[i]);//generate corresponding command's
character string

        strcat(cmdbuff, tempbuff);//splice character string

        if (strlen(cmdbuff)>1000)
        {
            return  ERR_AUX_PARAERR;     //parameters are wrong, character string
splice is too long
        }
    }
    //splice MPOS
    for (i=0;i<iaxisNum;i++)
    {
        sprintf(tempbuff,"MPOS(%d),",iaxislist[i]);//generate corresponding command's
character string

        strcat(cmdbuff, tempbuff);//splice character string

        if (strlen(cmdbuff)>1000)
        {
            return  ERR_AUX_PARAERR;     //parameters are wrong, character string
splice is too long
        }
    }
    //splice AXISSTATUS
    for (i=0;i<iaxisNum;i++)
    {
        sprintf(tempbuff,"AXISSTATUS(%d),",iaxislist[i]);//generate corresponding
command's character string
```

```
            strcat(cmdbuff, tempbuff);//splice character string

            if (strlen(cmdbuff)>1000)
            {
                return  ERR_AUX_PARAERR;     //parameters are wrong, character string
splice is too long
            }
        }
        int32 ostart,istart,iend,oend;          //max 32 at once time
        bool addflag;

        addflag=false;

        int32 temp;         //max 32 at once time
        int32 temp2;        //max 32 at once time

        temp=endIn-startIn+1;
        if (temp%32 == 0)
        {
            temp=temp/32;
        }
        else
        {
            temp=temp/32+1;
        }

        //splice IN
        for (i=0;i<temp;i++)
        {
            istart = startIn+32*i;
            iend =istart+31;
            if (iend>endIn)
            {
                iend=endIn;
            }
            //generate the command
            sprintf(tempbuff, "IN(%d,%d),", istart,iend);
            strcat(cmdbuff, tempbuff);//splice character string
            if (strlen(cmdbuff)>1000)
            {
                return ERR_AUX_PARAERR ;     //parameters are wrong, character string
splice is too long
            }

        }

        temp2=endOut-startOut+1;
        if (temp2%32 == 0)
        {
            temp2=temp2/32;
        }
        else
        {
            temp2=temp2/32+1;
```

```
    }
    //splice OUT
    for (i=0;i<temp2;i++)
    {
        ostart = startOut+32*i;
        oend =ostart+31;
        if (oend>endOut)
        {
            oend=endOut;
        }


        //generate the command
        sprintf(tempbuff, "OUT(%d,%d)", ostart,oend);

        strcat(cmdbuff, tempbuff);//splice character string
        if (i<temp-1)
        {
            strcat(cmdbuff, ",");//splice character string
        }

        if (strlen(cmdbuff)>1000)
        {
        return  ERR_AUX_PARAERR;     //parameters are wrong, character string splice
 is too long
        }

    }

    printf("spliced character string: \n",cmdbuff);
    printf("%s\n",cmdbuff);

    ret=ZAux_DirectCommand(handle,cmdbuff,cmdbuffAck,2048);
    if(ERR_OK != ret)
    {
        return ret;
    }
    //printf("%s\n",cmdbuffAck);
    //printf("%d\n",strlen(cmdbuffAck));
    //
    if(0 == strlen(cmdbuffAck))
    {
        return ERR_NOACK;
    }
    float ftempbuff[200];
    int itempbuff[200];

    ZAux_TransStringtoFloat(cmdbuffAck,iaxisNum*2,ftempbuff);//character string is
converted into floating
    //DPOS output
    for(i=0;i<iaxisNum;i++)
    {
        //printf("%f\n",ftempbuff[i]);
        fDposlist[i]=ftempbuff[i];
```

```
	}
	//MPOS output
	for(i=0;i<iaxisNum;i++)
	{
		//printf("%f\n",ftempbuff[i+iaxisNum]);
		fMposlist[i]=ftempbuff[i+iaxisNum];
	}

	ZAux_TransStringtoInt(cmdbuffAck,iaxisNum*3+temp2+temp,itempbuff);//characte
r string is converted into integer
	//AXISSTATUS output
	for(i=0;i<iaxisNum;i++)
	{
		//printf("%d\n",itempbuff[i+iaxisNum*2]);
		iAxisstatuslist[i]=itempbuff[i+iaxisNum*2];
	}
	//IN output
	for(i=0;i<temp;i++)
	{
		//printf("%d\n",itempbuff[i+iaxisNum*3]);
		iIn[i]=itempbuff[i+iaxisNum*3];
	}
	//OUT output
	for(i=0;i<temp2;i++)
	{
		//printf("%d\n",itempbuff[i+iaxisNum*3+temp]);
		iOut[i]=itempbuff[i+iaxisNum*3+temp];
	}
	return ERR_OK;
}
int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;			//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to connect controller!\n");

	int axis[4]={0,1,2,4};
	float d_dpos[4];
	float d_mpos[4];
	int32 d_axis_status[4];
	int d_in[10];
	int d_out[10];

	ret =
Demo_Direct_MyGetData(handle,3,axis,d_dpos,d_mpos,d_axis_status,0,32,d_in,0,33,d_ou
t);
	int i;
```

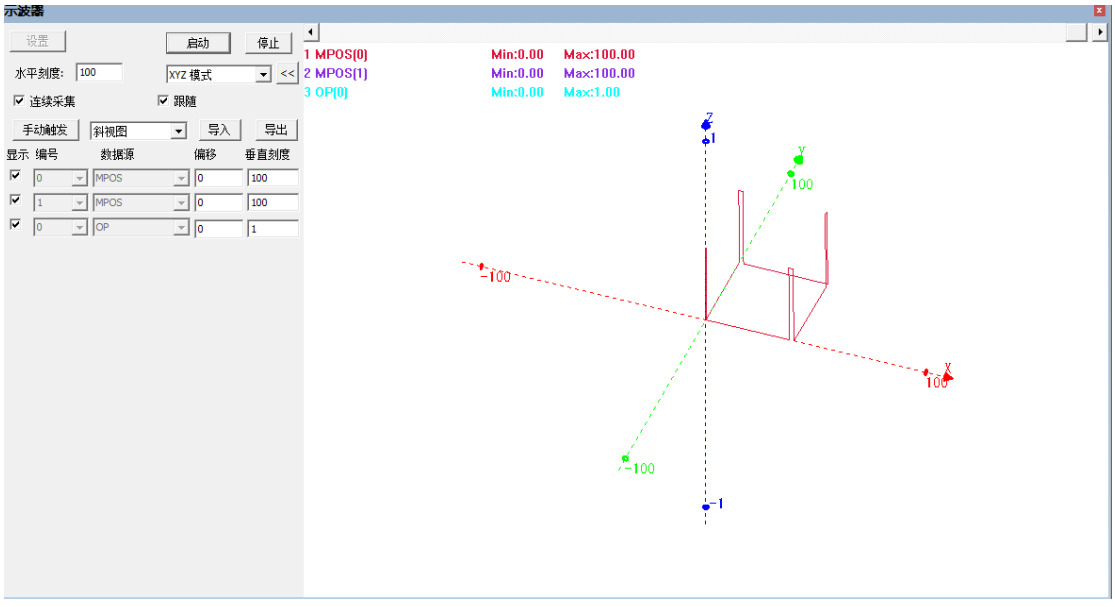
```
printf("obtained axis command position: \n");
for (i=0;i<3;i++)
{
	printf("\taxis%d :%f",i,d_dpos[i]);
}
printf("\n");
printf("obtained axis feedback position: \n");
for (i=0;i<3;i++)
{
	printf("\taxis%d :%f",i,d_mpos[i]);
}
printf("\n");
printf("obtained axis state (bit by bit): \n");
for (i=0;i<3;i++)
{
	printf("\taxis%d :%d",i,d_axis_status[i]);
}
printf("\n");

printf("obtained input state: \n");
int j=0;
int tempval;
for (i=0;i<=32;i++)
{
	if (((i%32)==0)&&(i>0) )
	{
		j++;
	}
	//convert into bit
	tempval=d_in[j]>>(i-32*j);
	printf("    IN(%d):%d",i,tempval &(0x01));
	if (((i%8)==0)&&(i>0) )
	{
		printf("\n");
	}
}
printf("\n");

printf("obtained output state: \n");
j=0;

for (i=0;i<=33;i++)
{
	if (((i%32)==0)&&(i>0) )
	{
		j++;
	}
	//convert to bit
	tempval=d_out[j]>>(i-32*j);
	printf("    OUT(%d):%d",i,tempval &(0x01));
	if (((i%8)==0)&&(i>0) )
	{
		printf("\n");
	}
}
```

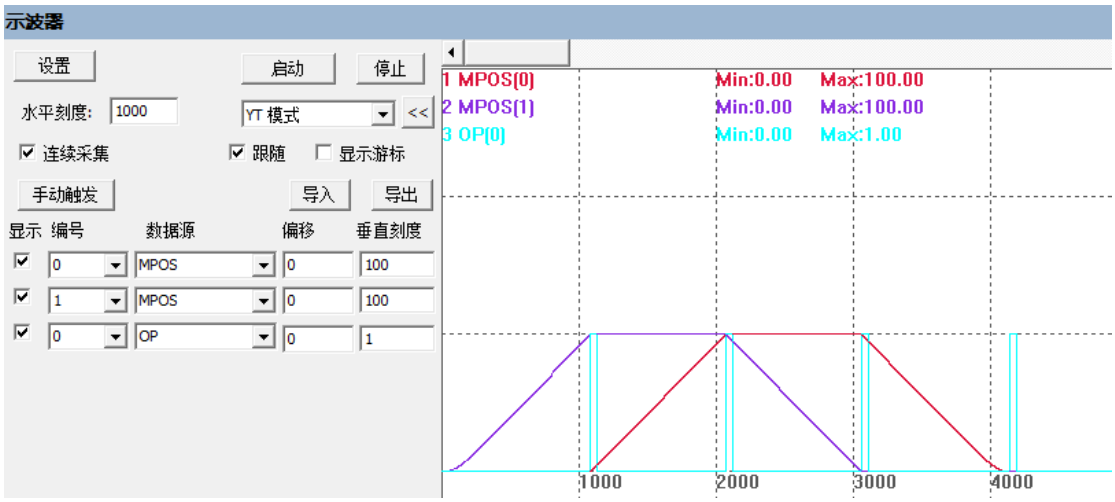
```
        printf("\n");

        Sleep(20000);
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
        printf("connection closed!\n");

        handle = NULL;
        return 0;
}
```

11.1.2.3. Use Online Command Function 3

Generally, the dispensing industry and the woodworking industry are mostly used in the continuous track, and there is an inserted buffer output between the continuous tracks. If the motion and the continuous track motion and the buffer output are sent separately, there will inevitably be limitations. Actually, encapsulate the motion function separately can achieve the effect of executing multiple functions in one line of commands, as shown in the following example:

Assuming to control an XY two-axis platform, from the coordinate point (0,0) -> (100,0) (output port 0 outputs 50ms) -> (100,100) (output port 0 outputs 50ms) -> (0,100) (output port 0 outputs 50ms) -> (0,0) (output port 0 outputs 50ms), then encapsulate it by yourself and use a function to send it quickly:






```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}

}

/*************************************************************
Description:	//customized motion function
Input:		//handle		card link
			iMoveLen		filled motion length
			iaxisNum		total axes that joined in motion
			iaxislist		axis No. list
			fPoslist		distance list
			iout			buffer output
			outlist			buffer output list (each motion, whether outputs, 0
means not to output, 1 means output after the motion)
			outtime			buffer output time

Output:		//
Return:		//error codes
*************************************************************/
int  Demo_Direct_MyMoveABS(ZMC_HANDLE handle,int iMoveLen,int iaxisNum, int*
iaxislist, float* fPoslist,int iout,int *outlist,int outtime)
{
	char cmdbuff[2048];
	char  tempbuff[2048];
	char cmdbuffAck[20480];
```

```
    //if transferred address is blank, exit


    int ret=0;
    int i;
    //firstly read remaining linear buffers
    int iBuffLen = 0;
    ret = ZAux_Direct_GetRemain_LineBuffer(handle,iaxislist[0],&iBuffLen);
    if(iBuffLen <= iMoveLen*2)
    {
        return 1002;            //motion buffer is not enough
    }

    //generate the command
    sprintf(cmdbuff, "BASE(");
    //splice motion axes list
    for (i=0;i<iaxisNum-1;i++)
    {
        sprintf(tempbuff,"%d,",iaxislist[i]);//generate corresponding command's
character string

        strcat(cmdbuff, tempbuff);//splice character string

        if (strlen(cmdbuff)>1000)
        {
        return  ERR_AUX_PARAERR;     //parameters are wrong, character string
 splice is too long.
        }
    }
    sprintf(tempbuff,"%d)\n",iaxislist[i]);//generate corresponding command's
character string

    strcat(cmdbuff,tempbuff);

    //splice motion
    for (i=0;i<iMoveLen;i++)
    {
        //printf("%d,%d\n",i*iaxisNum,i*iaxisNum+1);
        if (outlist[i]==0)    //this motion doesn't output
        {
            strcat(cmdbuff, "MoveAbs(");

    sprintf(tempbuff,"%f,%f)\n",fPoslist[i*iaxisNum],fPoslist[i*iaxisNum+1]);//generate
corresponding command's character string
            strcat(cmdbuff, tempbuff);//splice the character string

        }
        else if (outlist[i]==1)
        {
                strcat(cmdbuff, "MoveAbs(");

    sprintf(tempbuff,"%f,%f)\n",fPoslist[i*iaxisNum],fPoslist[i*iaxisNum+1]);//generate
corresponding command's character string
                strcat(cmdbuff, tempbuff);//splice the character string
```

```
                    strcat(cmdbuff, "Move_op2(");
                    sprintf(tempbuff,"%d,%d,%d)\n",iout,1,outtime);//generate
corresponding command's character string
                    strcat(cmdbuff, tempbuff);//splice the character string
          }
          else
          {
               return  ERR_AUX_PARAERR;// parameters are wrong
          }


     }


     printf("spliced character string:\n");
     printf("%s\n",cmdbuff);

     if (strlen(cmdbuff)>1000)
     {
          return  ERR_AUX_PARAERR;     //parameters are wrong, character string
splice is too long.
     }
     ret=ZAux_DirectCommand(handle,cmdbuff,cmdbuffAck,2048);

     return ret;


}
int _tmain(int argc, _TCHAR* argv[])
{
     char *ip_addr = (char *)"127.0.0.1";          //controller IP address
     ZMC_HANDLE handle = NULL;              //link handle
     int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
     if (ERR_SUCCESS != ret)
     {
          printf("Fail to connect controller!\n");
          handle = NULL;
          Sleep(2000);
          return -1;
     }
     printf("Success to connect controller!\n");

     ret =ZAux_Direct_SetAtype(handle,0,1);//set axis type of axis 0 as 1
     commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully
     ret =ZAux_Direct_SetAtype(handle,1,1);//set axis type of axis 1 as 1
     commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
instruction is executed successfully

     ret =ZAux_Direct_SetUnits(handle,0,100);//set pulse amount of axis 0 as 100
     commandCheckHandler("ZAux_Direct_SetUnits", ret) ;// judge whether the
instruction is executed successfully
     ret =ZAux_Direct_SetUnits(handle,1,100);//set pulse amount of axis 1 as 100
     commandCheckHandler("ZAux_Direct_SetUnits", ret) ;// judge whether the
```

```
instruction is executed successfully

    ret =ZAux_Direct_SetAccel(handle,0,500);//set acceleration of axis 0
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;// judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetAccel(handle,1,500);//set acceleration of axis 1
    commandCheckHandler("ZAux_Direct_SetAccel", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetDecel(handle,0,500);//set deceleration of axis 0
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetDecel(handle,1,500);//set deceleration of axis 1
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetDpos(handle,0,0);//set clear axis 0 DPOS to 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetDpos(handle,1,0);//set clear axis 1 DPOS to 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetSpeed(handle,0,100);//set speed of axis 0
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetSpeed(handle,1,100);//set speed of axis 1
    commandCheckHandler("ZAux_Direct_SetDecel", ret) ;// judge whether the
instruction is executed successfully

    ret =ZAux_Direct_SetMerge(handle,0,1);//set open continuous interpolation (only
main axis is opened, for example, axis 0 and axis 1 interpolate, axis 0 is the master
axis. The main axis is determined by the first axis No. in axis list of continuous
interpolation motion commands.


    int axis[2]={0,1};
    float POS[12]={0,0,0,100,100,100,100,0,0,0};
    int otlist[5]={0,1,1,1,1};

    ZAux_Trigger(handle);//trigger the oscilloscope

    ret = Demo_Direct_MyMoveABS(handle,5,2,axis,POS,0,otlist,50);//
    commandCheckHandler("Demo_Direct_MyMoveABS", ret) ;// judge whether the
instruction is executed successfully

    Sleep(20000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;// judge whether the instruction is
executed successfully
    printf("connection closed!\n");

    handle = NULL;
    return 0;
```

```
}
```

11.2. Controller ZBASIC Related

11.2.1. Emphasis

11.2.1.1. ZBasic Program Description

The control program referred to by ZBasic is a program downloaded and runs independently in the controller. It is necessary for users to know about some corresponding commands and grammars and downloading steps, then they can use control program. And there is a big difference in the size of the program space for independent operation of different series of controllers. For specific resources, please refer to the product manual of the corresponding model.

Advantages of independent control program:

- **Independence**: the control program can be executed independently from the PC without occupying PC task resources. The PC only needs simple logic control to complete the entire motion control.
- **Real-time:** since WINDOWS is not a real-time system, the task scheduling of the application program is completely controlled by the PC, and data interaction between the application program and the controller needs to be carried out continuously, so often in some high-speed response control, the application program through the PC is unable to respond in time.
- **Confidentiality**: the development can download the non-exportable independent control program in advance or generate a download package and put it on the application program, and the file cannot be uploaded, and can be encrypted as a zar file for download. The encryption of the application program can be implemented by encrypting the control program, or some control functions can be encrypted to protect the interests of developers.

11.2.1.2. How to Use ZBasic for PC

1. Control program writing

The writing of the control program needs to be written in the xxx.bas file, and the syntax format is ZBasic. (You can find the corresponding technical support, or download the corresponding bas file from the CD-ROM, or contact us directly)

2. Download the control program

Use the **ZAux_BasDown** command to download the program. (Note: The xxx.bas file path cannot be filled incorrectly, and the file path should not be too long. (within 1000 bytes))

11.2.2. Routine

11.2.2.1. Download & Edit Control Program

This routine mainly demonstrates how to use the PC library function to download the written xxx.bas program to the controller. First write the control program, then save the written program to the specified path, and then use ZAux_BasDown to download it.

Note: the file path must be represented by "\\", not "/" or "\".

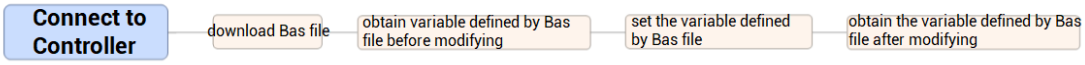
For example: the path of the .bas file is "F:\111\Basic1.bas", in C language, the "\" in the path should be replaced by "\\", namely: "F:\\111\\Basic1 .bas".

Example: The bas file of this routine is placed in the current path, then "\" in C language is replaced by ".\\ECAT initializes .bas"

The bas file used in this routine is an initialization program for the EtherCat bus axis. The purpose of this routine is to use the Bas file and the ZMC432 controller to initialize the configuration of the bus axis connected to the controller.

The functions involved in this routine are: download the "ECAT initialization.bas" file to controller, and configure three variables, they are global variable (PUL_AxisStart) (local pulse starting axis No.), PUL_AxisNum (local pulse axis numbers) and Bus_AxisStart (bus axis starting axis No.).

```
'*********************************************************ECAT总线初始化
global  BUS_TYPE            '总线类型。可用于上位机区分当前总线类型
global  Bus_Slot            '槽位号0（单总线控制器缺省0）
global  PUL_AxisStart       '本地脉冲轴起始轴号
global  PUL_AxisNum         '本地脉冲轴轴数量
global  Bus_AxisStart       '总线轴起始轴号
global  Bus_NodeNum         '总线配置节点数量,用于判断实际检测到的从站数量是否一致

BUS_TYPE        = 0     '总线类型。可用于上位机区分当前总线类型
Bus_Slot        = 0     '槽位号0（单总线控制器缺省0）
PUL_AxisStart   = 0     '本地脉冲轴起始轴号
PUL_AxisNum     = 0     '本地脉冲轴轴数量
Bus_AxisStart   = 0     '总线轴起始轴号
Bus_NodeNum     = 2     '总线配置节点数量,用于判断实际检测到的从站数量是否一致

```




```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
    if (ret)//it is not 0, fail
    {
        printf("%s fail!return code is %d\n", command, ret);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"192.168.0.11";          //controller IP address
    ZMC_HANDLE handle = NULL;               //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to connect controller!\n");

    const char * filename= ".\\ECAT Init.bas";

    //download into ROM, 0-RAM, 1-ROM
    ret = ZAux_BasDown(handle,filename,1);//download the ECAT init bas file to the
```

```
controller for initialization. After the program is downloaded, after a 7S delay (waiting for the driver to power on the reserved time) to initialize the Ethercat bus
    commandCheckHandler("ZAux_BasDown",ret);
    Sleep(500);//delay 500ms, wait for the program is downloaded to controller
    //PUL_AxisStart, PUL_AxisNum and Bus_AxisStart are 3 global variables defined byECAT init.bas file

    float GetUserVar;
    ret = ZAux_Direct_GetUserVar(handle,"PUL_AxisStart",&GetUserVar);//obtain controller local pulse axis starting axis No. before modifying
    commandCheckHandler("ZAux_BasDown",ret);
    printf("before set PUL_AxisStart=%f\n",GetUserVar);
    ret = ZAux_Direct_GetUserVar(handle,"PUL_AxisNum",&GetUserVar);//obtain controller local pulse total axes before modifying
    commandCheckHandler("ZAux_BasDown",ret);
    printf("before set PUL_AxisNum=%f\n",GetUserVar);
    ret = ZAux_Direct_GetUserVar(handle,"Bus_AxisStart",&GetUserVar);//obtain controller bus axis starting axis No. before modifying
    commandCheckHandler("ZAux_BasDown",ret);
    printf("before set Bus_AxisStart=%f\n",GetUserVar);

    //Because this routine uses ZMC432 this time, the controller body has 6 local pulse axes, so set the bus axis number to 0，
    //modify some for variables:
    //controller local pulse axis starting axis No.: PUL_AxisStart is modified as 0, 0 is by default
    //controller local pulse total axes.: PUL_AxisNum is modified as 6, 0 is by default
    //controller bus axis starting axis No.": Bus_AxisSt is modified as 6, 0 is by default
    ret = ZAux_Direct_SetUserVar(handle,"PUL_AxisStart",0);//set controller local pulse axis starting axis No.
    commandCheckHandler("ZAux_BasDown",ret);
    ret = ZAux_Direct_SetUserVar(handle,"PUL_AxisNum",6);//set controller local pulse total axes
    commandCheckHandler("ZAux_BasDown",ret);
    ret = ZAux_Direct_SetUserVar(handle,"Bus_AxisStart",6);//set controller bus axis starting axis No.
    commandCheckHandler("ZAux_BasDown",ret);

    ret = ZAux_Direct_GetUserVar(handle,"PUL_AxisStart",&GetUserVar);//obtain controller local pulse axis starting axis No. after modifying
    commandCheckHandler("ZAux_BasDown",ret);
    printf("after set PUL_AxisStart=%f\n",GetUserVar);
    ret = ZAux_Direct_GetUserVar(handle,"PUL_AxisNum",&GetUserVar);//obtain controller local pulse total axes after modifying
    commandCheckHandler("ZAux_BasDown",ret);
```

```
        printf("after set PUL_AxisNum=%f\n",GetUserVar);
        ret = ZAux_Direct_GetUserVar(handle,"Bus_AxisStart",&GetUserVar);//obtain
controller bus axis starting axis No. after modifying
        commandCheckHandler("ZAux_BasDown",ret);
        printf("after set Bus_AxisStart=%f\n",GetUserVar);

        Sleep(2000);
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
        printf("connection closed!\n");
        handle = NULL;
        return 0;
}
```

11.3. Controller Auto-Reporting

11.3.1. Emphasis

Take the initiative to report:

1. The controller actively reports.

2. Actively report the message: the controller sends a message to the PC through SEND_AUTOUP.

3. DLL support after 3.1.6.68, as shown in the below.

4. To check which PORTs support active reporting, you can use the ZDevelop online command?*IFAUTOUP_PORT to check, and set it to -1 to automatically select the PORT that can be reported.

Periodic reporting:

1. Both zmotion.dll and controller firmware support.

2. By setting the active periodic reporting of frequently read parameters in advance, the time for PC active polling can be reduced.

11.3.2. Routine

11.3.2.1. Report Actively

BASIC program:

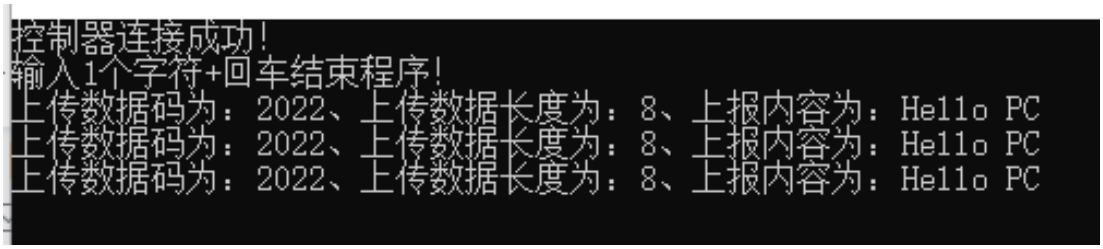
SEND_AUTOUP(-1,2022,"Hello PC")

Run the program on the PC first (it will automatically open the port to be reported), and then execute SEND_AUTOUP(-1,2022,"Hello PC") through ZDevelop, and report once every time it is executed.

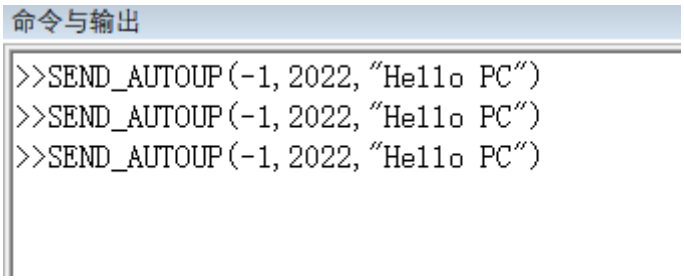

```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
```

```
#include "zauxdll2.h"
//this command is to check the returned value, if the returned value is not 0, print the returned value to screen.
void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}
}
//define the call-back function of reporting periodically
void AutoUpCallBack(ZMC_HANDLE handle, int32 itypecode, int32 idatalength, uint8 *pdata)
{
	printf("upload data code: %d, upload data length: %d, upload content: %s\n", itypecode,idatalength,(char*)pdata);
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;			//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to connect controller!\n");
	ret = ZAux_SetAutoUpCallBack( handle, AutoUpCallBack);
	commandCheckHandler("ZAux_SetAutoUpCallBack", ret);

	printf("input 1 character + “Enter” button to end program!\n");
	char a[128];
	scanf("%s",&a);

	ret = ZAux_Close(handle);	//close the connection
	commandCheckHandler("ZAux_Close", ret) ;// judge whether the instruction is executed successfully
	printf("connection closed!\n");
	handle = NULL;
	return 0;
}
```

- PC side:

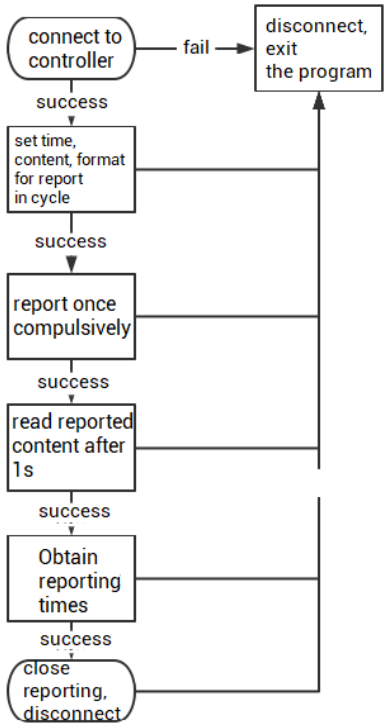


- Controller side



11.3.2.2. Report in Cycle

Periodic reporting, by setting the active periodic reporting of frequently read parameters in advance, it can reduce the active polling time of the PC. The specific reporting process is shown in the figure below.



```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include  <time.h>
//this command is to check the returned value, if the returned value is not 0, print the returned value to screen.
void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
		Sleep(2000);
		exit(0);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address

	ZMC_HANDLE handle = NULL;			//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to connect controller!\n");

	ret = ZAux_CycleUpEnable(handle, 0, 700, "idle, mpos(0,3), dpos(1)");//report motion state of axis 0, DPOS of axis 1 and MPOS of axis 0-2 in cycle
	commandCheckHandler("ZAux_CycleUpEnable", ret) ;//judge whether the instruction is executed successfully

	ret = ZAux_CycleUpForceOnce(handle, 0);//report once compulsively, 0 is the channel No.
	commandCheckHandler("ZAux_CycleUpForceOnce", ret) ;//judge whether the instruction is executed successfully

	double dtemp;//read periodical reported content in floating type
	int32 itemp,addrnum;//read periodical reported content in integer type
```

```
    addrnum=0;

    Sleep(1000);
    ret = ZAux_CycleUpReadBuff(handle, 0, "mpos", 0, &dtemp);//read MPOS (0) in floating type
    commandCheckHandler("ZAux_CycleUpReadBuff", ret) ;//judge whether the instruction is executed successfully

    ret = ZAux_CycleUpReadBuffInt(handle, 0, "dpos", 1, &itemp);//read DPOS (1) in integer
    commandCheckHandler("ZAux_CycleUpReadBuff", ret) ;//judge whether the instruction is executed successfully
    printf("itemp=%d,mpos=%f\n",itemp,dtemp);

    addrnum = ZAux_CycleUpGetRecvTimes(handle, 0);//obtain report times
    printf("report times=%d\n",addrnum);

    Sleep(2000);

    ret = ZAux_CycleUpDisable(handle, 0);//close periodical reporting
    commandCheckHandler("ZAux_CycleUpEnable", ret) ;//judge whether the instruction is executed successfully

    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

11.4. Debug Related

11.4.1. Emphasis

11.4.1.1. Debug Messages

The ZAux_SetTraceFile command needs to be used together with the ZAux_Execute command. After using the debugging information, the information sent by the ZAux_Execute command and the returned content will be saved by default, and saved in \ZmotionLog\ under the current program path, and saved in the form of a log.

Printout mode:

0: off

1: output error command

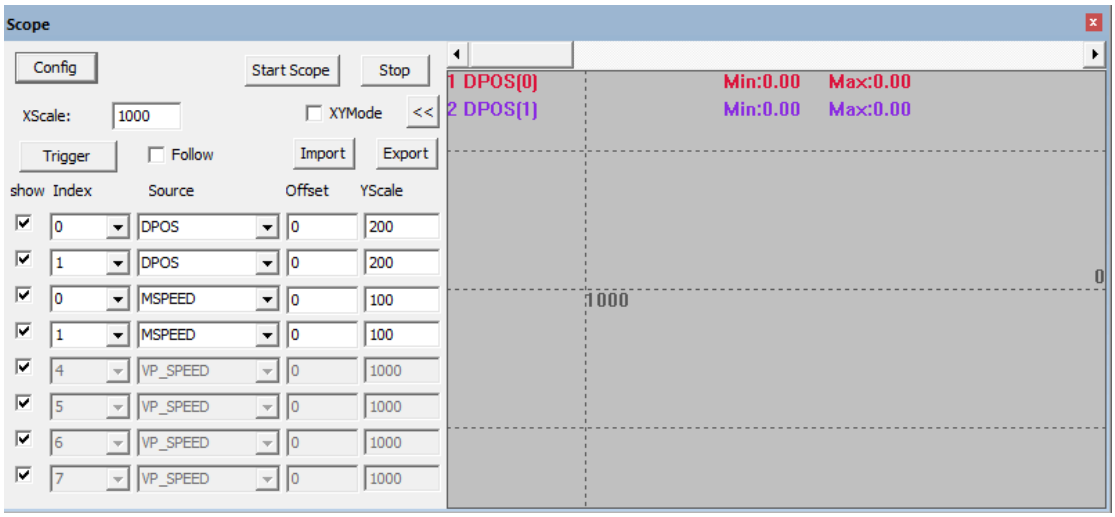
2: only the motion and parameter setting commands are output

3: output all commands

Note: this command is only used during debugging. When not in use, remember to turn off the print output mode, otherwise it will affect the communication efficiency.

11.4.1.2. Oscilloscope

Oscilloscope is extremely important of program debugging and running. It is used to transfer signals that can't be seen by naked eyes into graphics, so it is convenient to analyze change processes of all kinds of signals. Oscilloscope shows controller internal data in graph, it can display different signals, like, axis parameter, axis status, etc. “View” – “Scope", then oscilloscope window can be opened, or click the shortcut button .



The oscilloscope must be triggered to successfully sample. Turn on the oscilloscope, click "Start Scope" after setting the relevant parameters. Also, it could be manually triggered sampling, or add the "TRIGGER" instruction to the program to automatically trigger the oscilloscope sampling.

Oscilloscope basic setting button function:

Config: open oscilloscope configuration window, set relevant parameters

Start Scope: start oscilloscope (but not to trigger oscilloscope capture)

Stop: stop oscilloscope capture

XY mode: when checked, switch to the XY plane to display the interpolated composite track of the two axes.

<<: press to hide the channel name and peak value, and only display the channel number.

Trigger: manually trigger oscilloscope capture button (use TIGGER instruction to trigger automatically)

Follow: After the following is enabled, the horizontal axis automatically moves to the real-time sampling point and follows the waveform display.

Show: choose whether the present channel curve shows or not

Index: select the data source number that needs to be captured, such as, axis number, digital IO number, analog IO number, TABLE number, VR number, MODUBUS number, etc.

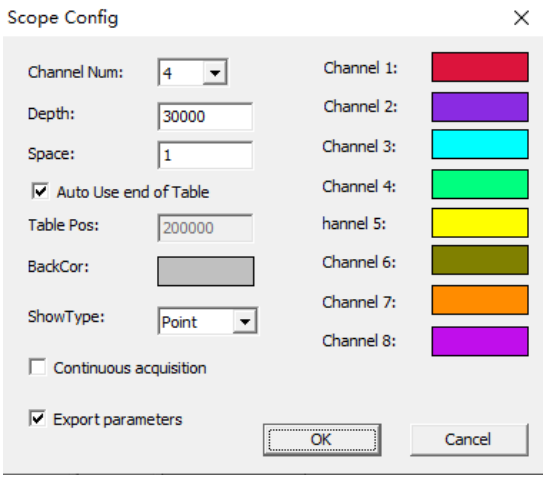
Source: select captured data type

Offset: Set the offset of the vertical axis of the waveform.

Xscale: The scale of one grid on the vertical axis.

Yscale: The scale of one grid on the horizontal axis.

If sets oscilloscope parameters, like, axis number, data source and start oscilloscope configuration window, it should stop firstly, then to do configure.



Chanel Num: channel total numbers to be sampled.

Depth: total sampling numbers, the depth is bigger, the sampling ranger is bigger.

Space: sampling time space. The unit is system cycle, which is related to the firmware version of the controller. Generally, default is 1ms, and see it through SERVO_PERIOD instruction. Usually, the smaller the space, the more accurate the sampled data, and the larger the amount of data per unit time.

TABLE Pos: Set the position where the captured data is stored. Generally, the space at

the end of the TABLE data is automatically used by default, and you can also customize the configuration, but be careful not to overlap with the TABLE data area used by the program when setting.

BackCor/Channel Color: set background and the color related to each channel's waveform.

ShowType: there are two curve types to be chosen, point and line. Abnormal data is easily found out by Line.

Continuous acquisition: when the continuous acquisition is not enabled, the sampling will stop when the sampling depth is reached. After the continuous acquisition is enabled, the oscilloscope will continue to sample.

Export parameters: export oscilloscope parameters.

Import and Export Data of Oscilloscope

Import: data can be imported when oscilloscope stops, and if it is imported successfully, sampling waveform can reappear.

The method of importing sampling data: click "Import", import data file that should be the former file type exported from oscilloscope, then open it.

Export: export parameters, including oscilloscope parameter configuration, data type of each channel and each sampling point's data.

The method of exporting data: click "Export parameters" in "Scope Config", then start scope. After sampling, click "Export", and select folder to store oscilloscope data, the exported data is text file.

Oscilloscope Sampling Method

- Open project, connect to controller or simulator, then open the oscilloscope window (note: first, connect to controller or simulator, then operate the oscilloscope window).
- Click "Scope Config" in oscilloscope window, select sampling channel numbers, sampling depth, sampling space, sampling data TABLE stored position (generally the end position of TABLE array will be used automatically) and sampling type, etc. Then, click "OK" for saving this configuration.
- Select sampling Index and Source, click "Start Scope".
- Download program into controller, the program should include TRIGGER, the trigger oscilloscope sampling automatically instruction. Now, it starts sampling and shows different data source's waveforms. It can adjust Show Scale and Wave Offset, this is for observing different waveforms.

If the waveform accuracy is not high or the display is incomplete, click the "Stop" button

and then open the "Scope Config", and adjust the sampling space and sampling depth, and perform the above sampling process again.

If the sampling time is long, start “Continuous acquisition" function.

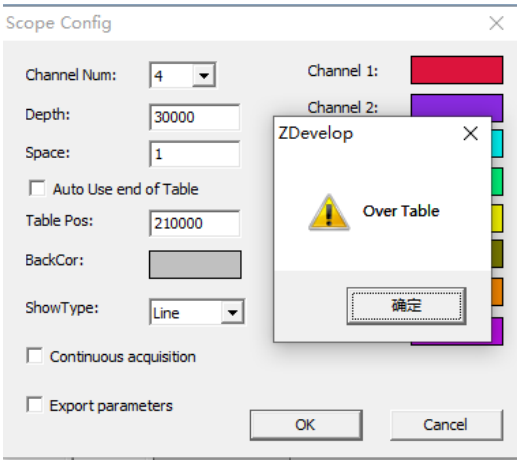
Precautions for Using Oscilloscope

Oscilloscope sampling time calculation:

- For example, depth: 1000, space: 5
- If system cycle is SERVO_PERIOD=1000, which means 1ms trajectory planning cycle. Space 5 means sampling one data point per 5ms. Total sampling data number is 10000, so sampling time length is 50s.

TABLE data end storage space calculation:

- Set the position where the captured data is stored. Generally, the space at the end of the TABLE data is automatically used by default, now starting space address is calculated automatically according to captured data space.
- Calculation method: captured data space = channel numbers * depth
- For example, if TABLE space of controller is 320000, there are 4 sampling channels, depth is 30000, each sampling point occupies one TABLE, so it will occupy 4*30000=120000 TABLE positions. 320000-120000=200000, which means starting position of TABLE is 200000.
- The position for storing data also can be self-defined, if according to above channel number and depth, TABLE starting space can't exceed 200000 when it is self-defined, otherwise, it can't be configured, please see below picture:



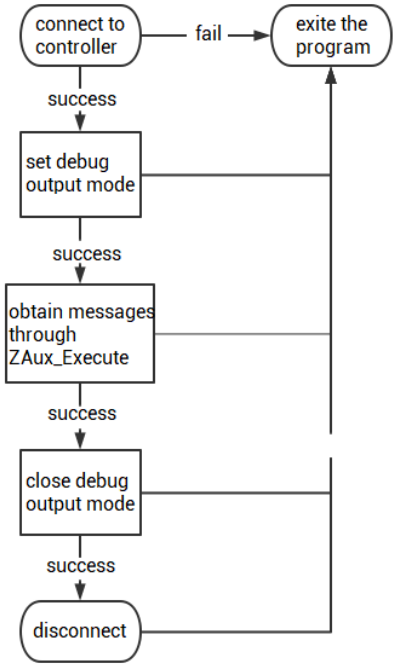
- The space occupied by the oscilloscope sample data should not overlap with the TABLE data area used by the program.
- Controller TABLE space can be read through TSIZE instruction, check “Controller Status" or input ?*max to print and check.

Continuous acquisition:

- ✧ When continuous acquisition is not selected, the oscilloscope automatically stops sampling when the sampling depth is reached.
- ✧ First select "Continuous acquisition" in "Scope Config", then start oscilloscope, it will continue to sampling after triggered, and sampling even if it reached the depth. It will stop until press "Stop" button manually.
- ✧ All waveforms and captured data from continuous acquisition can be exported.

11.4.2. Routine

11.4.2.1. Debug Print Messages

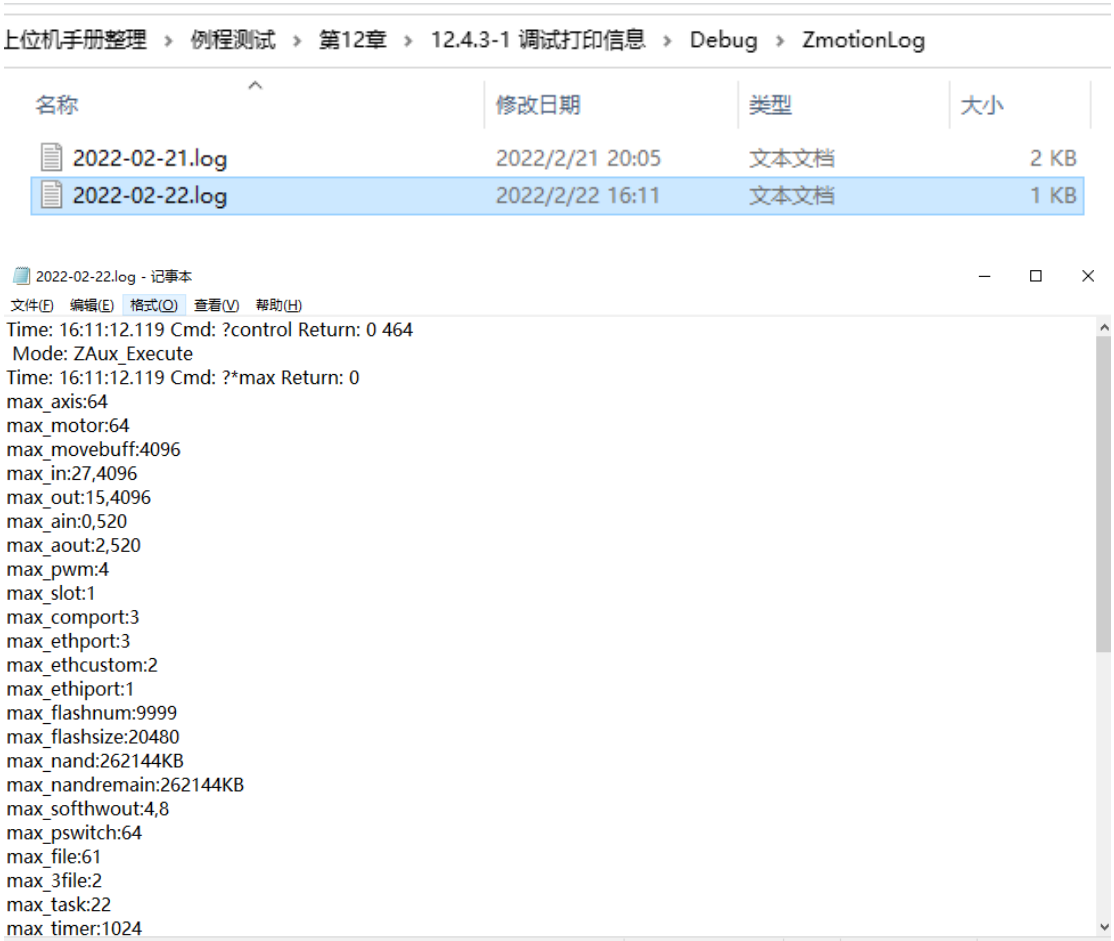



```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include  <time.h>
//this command is to check the returned value, if the returned value is not 0, print the returned value to screen.
void commandCheckHandler(const char *command, int ret)
{
```

```
    if (ret)//it is not 0, fail
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    //char *ip_addr = (char *)"192.168.1.11";          //controller IP address
    char *ip_addr = (char *)"127.0.0.1";          //controller IP address
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);  //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to connect controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to connect controller!\n");

    //0 close  1-only error commands are output  2-only motion and setting commands are output  3 all commands are output
    int bifTofile = 3;    //set debug output mode as 3

    ret = ZAux_SetTraceFile(bifTofile, "reserved");//set print output mode
    commandCheckHandler("ZAux_SetTraceFile",ret);

    const char * command = "?control";//send the command that gets controller model

    const char * command1 = "?*max"; //send the command that sends controller messages

    char psRespons[1024];          //the array is used to receive controller model
    int length = 1024;             //controller model length, which must be larger than received character string length

    ret = ZAux_Execute(handle,command, psRespons,length);//obtain controller model
    commandCheckHandler("ZAux_Execute", ret);

    ret = ZAux_Execute(handle,command1, psRespons,length);//obtain controller model
```

```
        commandCheckHandler("ZAux_Execute", ret);

        Sleep(2000);
        ret = ZAux_SetTraceFile(0, "reserved");//close print output mode, after debugged, print output mode must be closed
        commandCheckHandler("ZAux_Execute", ret);
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;// judge whether the instruction is executed successfully
        printf("connection closed!\n");
        handle = NULL;
        return 0;
}
```

➢ Output the result:



11.4.2.2. Use Oscilloscope

This routine demonstrates that through axis 0 and axis 1, walk from the position of

plane point (0,0) to the position of plane point (100,100), and capture the motion trajectory diagram and speed curve diagram through the oscilloscope.

PC program:

```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include  <time.h>
//this command is to check the returned value, if the returned value is not 0, print the returned value to screen.
void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
		Sleep(2000);
		exit(0);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	//char *ip_addr = (char *)"192.168.1.11";		//controller IP address
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;				//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to connect controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to connect controller!\n");

	ret =ZAux_Direct_SetAtype(handle,0,1);//set axis type of axis 0 as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the instruction is executed successfully

	ret =ZAux_Direct_SetAtype(handle,1,1);//set axis type of axis 1 as 1
	commandCheckHandler("ZAux_Direct_SetAtype", ret) ;// judge whether the
```

```
instruction is executed successfully
    commandCheckHandler("ZAux_Execute", ret);

    ret = ZAux_Direct_SetDpos(handle, 0, 0);       //set dpos of axis 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, 1, 0);       //set dpos of axis 1
    commandCheckHandler("ZAux_Direct_SetDpos", ret);

    int axislist[2] = { 0,1 };                     //motion axes list, axis 0 is the main axis
    float poslist[2] = { 100,100};                 //motion axes list, axis 0-100, axis1-100
    ZAux_Trigger( handle);//trigger the oscilloscope

    ret = ZAux_Direct_SetSpeed(handle, axislist[0], 100);      //set interpolation speed
as 100, set it on the main axis
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);

    ret = ZAux_Direct_SetAccel(handle, axislist[0], 500);      //set interpolation
acceleration as 500, set it on the main axis
    commandCheckHandler("ZAux_Direct_SetAccel", ret);

    ret = ZAux_Direct_SetDecel(handle, axislist[0], 500);      //set interpolation
deceleration as 100, set it on the main axis
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Direct_MoveAbs(handle, 2, axislist, poslist);     //call the motion, axis
0 and axis 0 move to corresponding 100, 100 absolute position
    commandCheckHandler("ZAux_Direct_MoveAbs", ret);

    int state[2];

    do
    {
        ZAux_Direct_GetIfIdle(handle, axislist[0], state);
        ZAux_Direct_GetIfIdle(handle, axislist[1], state + 1);

        if (!(state[0] || state[1]))
        {
            Sleep(100);
        }
        else
        {
            break;
        }
    } while (true);//wait for the motion to complete

    Sleep(2000);
```

```
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;// judge whether the instruction is
executed successfully
    printf("connection closed!\n");
    handle = NULL;
    return 0;
}
```

Chapter XII Details of Instructions

<table>
<tr><td>Instruction 1</td><td>ZAux_OpenCom</td></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_OpenCom(uint32 comid,ZMC_HANDLE *phandle)</td></tr>
<tr><td>Description</td><td>Connect to controller through serial port.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>Comid</td><td>Connected COM No.</td></tr></table></td></tr>
<tr><td>Output parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>Phandle</td><td>Controller link handle</td></tr></table></td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Connect to Controller through Serial Port</td></tr>
<tr><td>Details</td><td>1. There are several ways to connect to controller, and this instruction is serial port connection, but it is with lower communication speed compared to ethernet connection.
2. ZMC_HANDLE type: in Zmotion library, it is specified for type definition of control card connection data.
3. Controller default serial port parameters:<table><tr><td>Baud rate</td><td>Data bit</td><td>Stop bit</td><td>Parity</td></tr><tr><td>38400</td><td>8</td><td>1</td><td>No</td></tr></table></td></tr>
</table>

<table>
<tr><td>Instruction 2</td><td>ZAux_SetComDefaultBaud</td></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_SetComDefaultBaud(uint32 dwBaudRate, uint32 dwByteSize, uint32 dwParity, uint32 dwStopBits)</td></tr>
<tr><td>Description</td><td>Set serial port communication parameters.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>dwBaudRate</td><td>Baud rate (default): 38400</td></tr><tr><td>dwByteSize</td><td>Data bit size (default): 8-bit</td></tr><tr><td>dwParity</td><td>Parity (default): 0-no parity</td></tr><tr><td>dwStopBits</td><td>Stop bit (default): 1-bit</td></tr></table></td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Connect to Controller through Serial Port</td></tr>
</table>

Details	It is not consistent with SetCome in ZBasic, this function is only used for setting serial port parameters.

Instruction 3	ZAux_OpenEth
Original Format	int32 __stdcall ZAux_OpenEth(char *ipaddr, ZMC_HANDLE * phandle)
Description	Connect to controller through ethernet port.
Input parameters	Parameter name: Ipaddr — Description: Connected IP address
Output parameters	Parameter name: Phandle — Description: Returned link handle
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Connect to Controller through Ethernet Port
Details	1. The network port adopts RJ45 standard network cable interface, and the communication rate is 100Mbit/s. 2. The factory IP address of the controller is 192.168.0.11, and the port number is 502. The peer communication device must be in the same network segment as the controller before it can be connected. 3. It belongs to the most commonly used controller connection method. 4. ZMC_HANDLE type: in the Zmotion library, it is specially used to define the control card connection data type.

Instruction 4	ZAux_OpenPci
Original Format	int32 __stdcall ZAux_OpenPci(uint32 cardnum, ZMC_HANDLE *phandle)
Description	Connect to controller through PCI card.
Input parameters	Parameter name: Cardnum — Description: PCI card No., default starts from 0.
Output parameters	Parameter name: Phandle — Description: link handle
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Connect to Controller through PCI

Details	1. PCI connection requires a PCI type control card for PCI connection. This type of connection has the highest communication rate, but this connection requires an industrial computer with a PCI card slot interface. 2. ZMC_HANDLE type: in the Zmotion library, it is specially used to define the control card connection data type.

Instruction 5	**ZAux_Close**
Original Format	int32 __stdcall ZAux_Close(ZMC_HANDLE handle)
Description	Close controller connection
Input parameters	Parameter name: Handle — Description: Link handle
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Connect to Controller through Serial Port
Details	/

Instruction 6	**ZAux_SetIp**
Original Format	int32 __stdcall ZAux_SetIp(ZMC_HANDLE handle, char * ipaddress)
Description	Set controller IP address
Input parameters	Parameter name: Handle — Description: Link handle Parameter name: Ipaddress — Description: Set IP address value
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Modify IP address
Details	If the controller is connected through the network port, after the IP address is successfully modified, the controller will automatically disconnect from the PC, and the PC needs to be replaced with the corresponding network segment before reconnecting

Instruction 7	**ZAux_SearchEthlist**

<table>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_SearchEthlist(char *ipaddrlist, uint32 addrbufflength, uint32 uims)</td></tr>
<tr><td>Description</td><td>Search IP addresses under the current network segment.</td></tr>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>Addrbufflength</td><td>Search returned IP address total length</td></tr>
<tr><td>Uims</td><td>Search timeout time.</td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>Ipaddrlist</td><td>Searched all IP addresses</td></tr>
</table>
</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>
<pre>
//search IP address automatically
    char buffer[4096+256+1];
    int32 iresult;
    iresult = ZAux_SearchEthlist(buffer, 4096+256+1, 100);//search IP address to Buffer
    if(0!=iresult){return 0;}//fail and exit

    //converted from character string
    int ipos =0;
    const char * pstring;
    pstring = buffer;
    char buffer2[256];
    buffer2[0] = '\0';
    for(int j= 0; j< 20;j++)
    {
        //skip blank space
        while(' ' == pstring[0])
        {
            pstring++;

        }
        memset(buffer2, 0, sizeof(buffer2));
        ipos = sscanf(pstring , "%s", &buffer2);//obtain the address to buffer2
        if(-1 == ipos){break;}
        printf("%s\n",buffer2);
        //point to IP address ending
        while((' ' != pstring[0]) && ('\t' != pstring[0]) && ('\0' != pstring[0]))
        {
            pstring++;
        }
    }
</pre>
</td></tr>
</table>

Details	1. Controller default IP address is 192.168.0.11 2. If the IP addresses of the controller and the PC are not in the same network segment, the IP address will not be scanned

Instruction 8	**ZAux_SearchAndOpenCom**
Original Format	int32 __stdcall ZAux_SearchAndOpenCom(uint32 uimincomidfind, uint32 uimaxcomidfind, uint *pcomid, uint32 uims, ZMC_HANDLE *phandle)
Description	Search serial port automatically and connect to controller.
Input parameters	Parameter name: Description uimincomidfind: Searched minimal COM No. uimaxcomidfind: Searched maximum COM No. uims: Connection respond time MS
Output parameters	Parameter name: Description Pcomid: Searched COM No. phandle: Returned handle
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	/

Instruction 9	**ZAux_SearchEth**
Original Format	int32 __stdcall ZAux_SearchEth(const char *ipaddress, uint32 uims)
Description	Search whether the current IP address is in this network segment.
Input parameters	Parameter name: Description Pcomid: Search judged IP address phandle: Search timeout time
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	/

Instruction 10	**ZAux_GetMaxPciCards**
Original Format	int32 __stdcall ZAux_GetMaxPciCards( )

Description	Search current max PCI card numbers.
Input parameters	/
Output parameters	/
Return value	Max card numbers.
Example	Int Card; Card=ZAux_GetMaxPciCards();//obtain max card numbers
Details	/

Instruction 11	**ZAux_FastOpen**
Original Format	int32 __stdcall ZAux_FastOpen(int type, char *pconnectstring, uint32 uims ,ZMC_HANDLE * phandle)
Description	Build the connection with controller, it can specify connection waiting time.
Input parameters	Parameter name: Description type: Connection type type: 1-COM, 2-ETH, 4-PCI, 5-MotionRT pconnectstring: Connection character string: type=1: COM + serial port No. type=2: IP address type=4: Pci + card No. type=5: no request uims: Connection timeout time uims, unit is ms.
Output parameters	Parameter name: Description phandle: Link handle
Return value	If it is successful, return value is 0, if not, please refer to error codes.

Example	Serial port connection:
	ZMC_HANDLE phandle;//controller link handle
	Char comID[32]= "0";//serial port ID, serial port No. cannot be written directly, Com is required before the serial port No.
	ZAux_FastOpen(1, comID,1000s ,&phandle);
	Ethernet port connection:
	ZMC_HANDLE phandle;//controller link handle
	Char EthID[32]= "192.168.0.11";//ethernet port ID
	ZAux_FastOpen(1, EthID,1000s ,&phandle);

Details	"type" is set to 5 that requires Zmotion.dll version above 3.8.8.50.

Instruction 12	**ZAux_GetControllerInfo**

Original Format	int32 __stdcall ZAux_GetControllerInfo(ZMC_HANDLE handle ,char *SoftType ,char *SoftVersion ,char *ControllerId)
Description	Read controller manufacturer information
Input parameters	Parameter name: handle — Description: Controller link handle
Output parameters	Parameter name: SoftType — Description: Controller software model Parameter name: SoftVersion — Description: Controller firmware version No. Parameter name: ControlllerId — Description: Controller unique ID
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Controller Information
Details	/

Instruction 13	**ZAux_GetSysSpecification**
Original Format	int32 __stdcall ZAux_GetSysSpecification(ZMC_HANDLE handle, uint16 *Max_VirtuAxises,uint8 *Max_motor,uint8 *Max_io)
Description	Read the controller max specification.
Input parameters	Parameter name: handle — Description: Controller link handle
Output parameters	Parameter name: Max_VirtuAxises — Description: The number of controller max virtual axes Parameter name: Max_motor — Description: The number of controller actual axes. Parameter name: Max_io — Description: IN, OUT, AD, DA, each occupies one unit8.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Controller Information
Details	/

Instruction 14	**ZAux_GetRtcTime**
Original Format	int32 __stdcall ZAux_GetRtcTim(ZMC_HANDLE handle, char *RtcDate, char * RtcTime)
Description	Read controller system time and date. Default starts from 2000.1.1.
Input parameters	Parameter name: handle — Description: Controller link handle

Output parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>RtcData</td><td>Controller system data. Format: YYMMDD / YYYYMMDD</td></tr><tr><td>RtcTime</td><td>Controller system time. Format: HHMMSS</td></tr></table>
Return value	Please refer to error codes.
Example	char RtcDate[6]; char RtcTime[6]; ZAux_GetRtcTime(handle,RtcDate,RtcTime);
Details	YYMMDD format: for example, 220101 – 1st Jan, 2022 YYYYMMDD format: for example, 20220101 – 1st Jan, 2022 HHMMSS format: for example, 131400 – thirteen fourteen / fourteen past thirteen (13:14:00)

Instruction 15	**ZAux_SetRtcTime**
Original Format	int32 __stdcall ZAux_SetRtcTim(ZMC_HANDLE handle, char *RtcDate, char * RtcTime)
Description	Set controller system time and date, they are saved when power-off. Default starts from 2000.1.1. And simulator time cannot be modified.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Controller link handle</td></tr><tr><td>RtcData</td><td>Controller system data. Format: YYMMDD / YYYYMMDD</td></tr><tr><td>RtcTime</td><td>Controller system time. Format: HHMMSS</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	char RtcDate[20]="20220101"; char RtcTime[20]="000000"; ZAux_SetRtcTime(handle,RtcDate,RtcTime);
Details	YYMMDD format: for example, 220101 – 1st Jan, 2022 YYYYMMDD format: for example, 20220101 – 1st Jan, 2022 HHMMSS format: for example, 131400 – thirteen fourteen / fourteen past thirteen (13:14:00)

Instruction 16	**ZAux_Direct_SetAtype**
Original Format	int32 __stdcall ZAux_Direct_SetAtype(ZMC_HANDLE handle, int iaxis, int iValue)
Description	Set the axis type of specified axis

Input parameters	Parameter name: handle — Controller link handle Parameter name: iaxis — Axis No. Parameter name: iValue — Axis type
Output parameters	/
Return value	Please refer to error codes.
Example	Get axis basic motion parameters configuration
Details 1	1. It is best to set ATYPE when the program is initialized. 2. The ZCAN expansion axis must first set AXIS_ADDRESS, and after setting, delay 2 task cycles before calling the motion command. Due to the limitation of the bus bandwidth, do not set more than 2 ZCAN expansion axes. 3. For some product models with independent encoder, the corresponding virtual axis can be used as the encoder axis. For example, the motor axis of ZMC206 is 0-5 axis, and the encoder can be controlled by axis 6-11. For details, please refer to controller state after the ZDevelop software is connected to the controller.
Details 2	Atype setting

Instruction 17	**ZAux_Direct_GetAtype**
Original Format	int32 __stdcall ZAux_Direct_GetAtype(ZMC_HANDLE handle, int iaxis, int *piValue)
Description	Read the axis type of specified axis
Input parameters	Parameter name: handle — Controller link handle Parameter name: iaxis — Axis No.
Output parameters	Parameter name: piValue — Axis type
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration
Details	/

Instruction 18	**ZAux_Direct_SetUnits**
Original Format	int32 __stdcall ZAux_Direct_SetUnits(ZMC_HANDLE handle,int iaxis, float fValue)
Description	Set the pulse equivalent. When it is set to 1, it means that the unit is 1 pulse.

	Pulse equivalent, the basic unit of the controller. It specifies the number of pulses sent per unit, supporting 5 decimal places of precision. The controller takes the pulse equivalent as the basic unit, and the coordinate display will change proportionally with the change of the pulse equivalent after modification.
Input parameters	Parameter name | Description handle | Controller link handle iaxis | Axis No. fValue | pulse amount that is set.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration

Details	How to set ?
	Assuming that the motor U=3600 pulses make one revolution, and the screw pitch P=2mm per revolution:
	The pulse equivalent UNITS corresponding to the motor turning 1°:
	UNITS=U/360=3600/360=10;//MOVE(1) at this time, the motor rotates 1°
	The pulse equivalent UNTIS corresponding to the workbench walking 1mm:
	UNITS=U/P=3600/2=1800, //MOVE(1) at this time, the workbench moves 1mm
	When the machine has a reduction ratio, the reduction ratio should be counted, assuming that the reduction ratio i=2:1
	UNITS=U*i/P=3600*2/2=3600

Instruction 19	**ZAux_Direct_GetUnits**
Original Format	int32 __stdcall ZAux_Direct_GetUnits(ZMC_HANDLE handle,int iaxis, float *pfValue)
Description	Read the pulse equivalent. Pulse equivalent, the basic unit of the controller. It specifies the number of pulses sent per unit, supporting 5 decimal places of precision. The controller takes the pulse equivalent as the basic unit, and the coordinate display will change proportionally with the change of the pulse equivalent after modification.
Input parameters	Parameter name | Description handle | Controller link handle iaxis | Axis No.

Output parameters	Parameter name: pfValue — Description: Returned pulse amount
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration
Details	/

Instruction 20	**ZAux_Direct_SetAccel**
Original Format	int32 __stdcall ZAux_Direct_SetAccel(ZMC_HANDLE handle,int iaxis, float fValue)
Description	Set acceleration, the unit is units/s/s.
Input parameters	Parameter name: handle — Description: Controller link handle Parameter name: iaxis — Description: Axis No. Parameter name: pfValue — Description: Acceleration that is set.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration
Details	1. When multi-axis motion, the acceleration of the interpolation motion sets the acceleration of the main axis. (main axis: the axis No. specified by the 0th data in the axis list array) 2. It is recommended to set the acceleration and deceleration before moving, and do not modify it during the exercise. Adjusting during the exercise will cause the speed curve to change.

Instruction 21	**ZAux_Direct_GetAccel**
Original Format	int32 __stdcall ZAux_Direct_GetAccel(ZMC_HANDLE handle,int iaxis, float *pfValue)
Description	Read acceleration, unit is units/s/s.
Input parameters	Parameter name: handle — Description: Controller link handle Parameter name: iaxis — Description: Axis No.
Output parameters	Parameter name: pfValue — Description: Returned acceleration
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration

Details	1. When multi-axis motion, the acceleration of the interpolation motion sets the acceleration of the main axis. (main axis: the axis No. specified by the 0th data in the axis list array) 2. It is recommended to set the acceleration and deceleration before moving, and do not modify it during the exercise. Adjusting during the exercise will cause the speed curve to change.

Instruction 22	**ZAux_Direct_SetDecel**
Original Format	int32 __stdcall ZAux_Direct_SetDecel(ZMC_HANDLE handle,int iaxis, float fValue)
Description	Set deceleration, unit is units/s/s
Input parameters	Parameter name: handle — Controller link handle iaxis — Axis No. pfValue — Deceleration that is set.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration
Details	1. When multi-axis motion, the deceleration of the interpolation motion sets the deceleration of the main axis. (main axis: the axis No. specified by the 0th data in the axis list array) 2. It is recommended to set the acceleration and deceleration before moving, and do not modify it during the exercise. Adjusting during the exercise will cause the speed curve to change. 3. When it is set to 0, it is equal to acceleration ACCEL value automatically, and it accelerates and decelerates symmetrically.

Instruction 23	**ZAux_Direct_GetDecel**
Original Format	int32 __stdcall ZAux_Direct_GetDecel(ZMC_HANDLE handle,int iaxis, float *pfValue);
Description	Read deceleration, unit is units/s/s
Input parameters	Parameter name: handle — Controller link handle iaxis — Axis No.

Output parameters	Parameter name: pfValue — Description: Get deceleration
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration
Details	1. When multi-axis motion, the deceleration of the interpolation motion sets the deceleration of the main axis. (main axis: the axis No. specified by the 0th data in the axis list array) 2. It is recommended to set the acceleration and deceleration before moving, and do not modify it during the exercise. Adjusting during the exercise will cause the speed curve to change. 3. When it is set to 0, it is equal to acceleration ACCEL value automatically, and it accelerates and decelerates symmetrically.

Instruction 24	**ZAux_Direct_SetSpeed**
Original Format	int32 __stdcall ZAux_Direct_SetSpeed(ZMC_HANDLE handle,int iaxis, float fValue)
Description	Set axis speed, unit is units/s.
Input parameters	Parameter name: handle — Description: Controller link handle Parameter name: iaxis — Description: Axis No. Parameter name: pfValue — Description: Speed that is set.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration
Details	1. When multi-axis motion, the speed of the interpolation motion sets the deceleration of the main axis. (main axis: the axis No. specified by the 0th data in the axis list array) 2. After the speed is modified, it will take effect immediately, and dynamic speed change can be realized, but it will shake on the instant of changing. If you want to change speed smoothly, please use **SPEED_RATIO** (zbasic command, PC can use this function: ZAux_Direct_SetParam(handle, "SPEED_RATIO", axis number, current speed ratio)) to replace the speed function, so that the online speed change becomes relatively smooth.

Instruction 25	**ZAux_Direct_GetSpeed**

Original Format	int32 __stdcall ZAux_Direct_GetSpeed(ZMC_HANDLE handle,int iaxis, float *pfValue)
Description	Read axis speed.
Input parameters	Parameter name: handle — Controller link handle; iaxis — Axis No.
Output parameters	Parameter name: pfValue — Get speed
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration
Details	When multi-axis motion, the speed of the interpolation motion sets the deceleration of the main axis. (main axis: the axis No. specified by the 0th data in the axis list array)

Instruction 26	**ZAux_Direct_SetDpso**
Original Format	int32 __stdcall ZAux_Direct_SetDpos(ZMC_HANDLE handle,int iaxis, float fValue)
Description	Set axis current position or the demand position sent by controller, the unit is units.
Input parameters	Parameter name: handle — Controller link handle; iaxis — Axis No.; pfValue — Current axis position
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration
Details	It will automatically convert into OFFPOS offset when write DPOS, motor will not be moved.

Instruction 27	**ZAux_Direct_GetDpso**
Original Format	int32 __stdcall ZAux_Direct_GetDpos(ZMC_HANDLE handle, int iaxis, float *pfValue)
Description	Read axis current position or the demand position sent by controller, the unit is units.

Input parameters	Parameter name: handle — Controller link handle; iaxis — Axis No.
Output parameters	Parameter name: pfValue — Get current axis position.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration
Details	/

Instruction 28	**ZAux_Direct_SetMpso**
Original Format	int32 __stdcall ZAux_Direct_SetMpos(ZMC_HANDLE handle, int iaxis, float fValue)
Description	Set axis measurement position, the unit is units.
Input parameters	Parameter name: handle — Controller link handle; iaxis — Axis No.; pfValue — Current axis measurement position value
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration
Details	It will automatically convert into OFFPOS offset when write MPOS, motor will not be moved.

Instruction 29	**ZAux_Direct_GetMpso**
Original Format	int32 __stdcall ZAux_Direct_GetMpos(ZMC_HANDLE handle, int iaxis, float *pfValue)
Description	Read axis measurement position, the unit is units.
Input parameters	Parameter name: handle — Controller link handle; iaxis — Axis No.
Output parameters	Parameter name: pfValue — Get current axis measurement position.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration

Details	/

Instruction 30	ZAux_Direct_GetEncoder
Original Format	int32 __stdcall ZAux_Direct_GetEncoder(ZMC_HANDLE handle,int iaxis, float *pfValue);
Description	Read internal encoder value, the unit is pulse
Input parameters	Parameter name: handle — Controller link handle; iaxis — Axis No.
Output parameters	Parameter name: pfValue — Return to current axis measurement position whose unit is pulse.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	float fValue; ret = ZAux_Direct_GetEncoder(handle, 0, &fValue); //read axis 0 inner encoder value
Details	1. the internal parameters can only be read when the ATYPE of the encoder is configured. 2. When the drive has multi-turn absolute value encoders, the multi-turn value is read.

Instruction 31	ZAux_Direct_GetFastDec
Original Format	int32 __stdcall ZAux_Direct_GetFastDec(ZMC_HANDLE handle, int iaxis, float * fValue)
Description	Read fast deceleration, and this speed is used automatically when in abnormal stop state. The unit is units/s/s.
Input parameters	Parameter name: handle — Controller link handle; iaxis — Axis No.
Output parameters	Parameter name: pfValue — Get fast deceleration value
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Position limit setting
Details	It is used automatically when CANCEL or abnormally stops. When the value is set to 0, it uses DECEL automatically.

Instruction 32	ZAux_Direct_SetFastDec

Original Format	int32 __stdcall ZAux_Direct_SetFastDec(ZMC_HANDLE handle, int iaxis, float fValue)
Description	set fast deceleration, and this speed is used automatically when in abnormal stop state. The unit is units/s/s.
Input parameters	Parameter name: handle — Controller link handle; iaxis — Axis No.; pfValue — Fast deceleration value
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Position limit setting
Details	It is used automatically when CANCEL or abnormally stops. When the value is set to 0, it uses DECEL automatically.

Instruction 33	**ZAux_Direct_SetLspeed**
Original Format	int32 __stdcall ZAux_Direct_SetLspeed(ZMC_HANDLE handle, int iaxis, float fValue)
Description	Set starting speed, the unit is units.
Input parameters	Parameter name: handle — Controller link handle; iaxis — Axis No.; pfValue — Starting speed that is set
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration (with initial speed & S curve)
Details	1. When in multi-axis motion, set the main axis as the initial speed of interpolation motion. 2. When you need to pursue efficiency, you can consider setting the starting speed.

Instruction 34	**ZAux_Direct_GetLspeed**
Original Format	int32 __stdcall ZAux_Direct_Ge tLspeed(ZMC_HANDLE handle, int iaxis, float *pfValue)
Description	Read starting speed, the unit is units.

Input parameters	Parameter name: handle — Controller link handle; iaxis — Axis No.
Output parameters	Parameter name: pfValue — Returned starting speed
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration (with initial speed & S curve)
Details	1. When in multi-axis motion, set the main axis as the initial speed of interpolation motion. 2. When you need to pursue efficiency, you can consider setting the starting speed.

Instruction 35	**ZAux_Direct_SetSramp**
Original Format	int32 __stdcall ZAux_Direct_SetSramp(ZMC_HANDLE handle, int iaxis, float fValue)
Description	Set S curve time, the unit is ms, 0 means ladder diagram acceleration and deceleration
Input parameters	Parameter name: handle — Controller link handle; iaxis — Axis No.; pfValue — S curve that is set
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration (with initial speed & S curve)
Details	1. After setting, the acceleration and deceleration process will prolong the corresponding time. 2. Set the maximum value as the acceleration time of the original T-shaped acceleration.

Instruction 36	**ZAux_Direct_GetSramp**
Original Format	int32 __stdcall ZAux_Direct_GetSramp(ZMC_HANDLE handle, int iaxis, float *pfValue)
Description	Read S curve time configuration.

Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Controller link handle</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr></table>
Output parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>pfValue</td><td>Returned S curve time value</td></tr></table>
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration (with initial speed & S curve)
Details	1. When in multi-axis motion, set the main axis as the initial speed of interpolation motion. 2. When you need to pursue efficiency, you can consider setting the starting speed.

Instruction 37	**ZAux_Direct_GetMspeed**
Original Format	int32 __stdcall ZAux_Direct_GetMspeed(ZMC_HANDLE handle, int iaxis, float *pfValue)
Description	Read current real-time axis measurement speed, the unit is units.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Controller link handle</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr></table>
Output parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>pfValue</td><td>Returned real-time axis measurement speed.</td></tr></table>
Return value	If it is successful, return value is 0, if not, please refer to error codes.

Example	float fValue;
	ret = ZAux_Direct_GetMspeed(handle, 0, &fValue);
	//read axis 0 current measurement speed.
	handle: controller link handle

Details	/

Instruction 38	**ZAux_Direct_GetVpSpeed**
Original Format	int32 __stdcall ZAux_Direct_GetVpSpeed (ZMC_HANDLE handle, int iaxis, float *pfValue)
Description	Read current real-time axis running speed sent by the controller, the unit is units.

<table>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Controller link handle</td></tr>
<tr><td>iaxis</td><td>Axis No.</td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>pfValue</td><td>Returned real-time axis speed.</td></tr>
</table>
</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>float fValue;
ret = ZAux_Direct_GetVpSpeed(handle, 0, &fValue);
//read axis 0 current command motion speed.
handle: controller link handle</td></tr>
<tr><td>Details</td><td>1. When multi-axis is in motion, the main axis returns the speed of the interpolation motion, not the sub-speed of the main axis.
2. The non-spindle returns the corresponding component speed, which is consistent with the feedback speed effect.
3. By default, the planning speed is designed to display the multi-axis composite speed, and there is no negative value, unless the SYSTEM_ZSET(zbasic command is used, the PC can use this function: ZAux_Direct_SetParam(handle, "SYSTEM_ZSET", axis No., setting value)) bit0 value is set to 0, it can be used to display the command speed of a single axis, which can be positive or negative.</td></tr>
</table>

<table>
<tr><td>Instruction 39</td><td>ZAux_Direct_GetIfIdle</td></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_GetIfIdle(ZMC_HANDLE handle, int iaxis, int *piValue)</td></tr>
<tr><td>Description</td><td>Read whether the current axis is in motion</td></tr>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Controller link handle</td></tr>
<tr><td>iaxis</td><td>Axis No.</td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>pfValue</td><td>Returned state: 0-in motion, -1-no motion</td></tr>
</table>
</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Multi-axis linear interpolation</td></tr>
<tr><td>Details</td><td>When the axis is associated with a manipulator, when CONNFRAME is reversed, the joint axis always returns 0, when CONNREFRAME is forward, the virtual axis always returns 0.</td></tr>
</table>

Instruction 40	ZAux_Direct_GetAxisStatus
Original Format	int32 __stdcall ZAux_Direct_GetAxisStatus(ZMC_HANDLE handle, int iaxis, int *piValue)
Description	Read the state of the current axis. The value is displayed in decimal, and the state is judged by the corresponding bit in binary.
Input parameters	Parameter name: handle — Controller link handle Parameter name: iaxis — Axis No.
Output parameters	Parameter name: pfValue — Returned state: corresponding bit means different states.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get axis basic motion parameters configuration (with initial speed & S curve)
Details	Returned state: corresponding bit means different states.

Instruction 41	ZAux_Direct_GetAxisStopReason
Original Format	int32 __stdcall ZAux_Direct_GetAxisStopReason(ZMC_HANDLE handle, int iaxis, int *piValue)
Description	Read axis history state.
Input parameters	Parameter name: handle — Controller link handle Parameter name: iaxis — Axis No.
Output parameters	Parameter name: pfValue — Returned state: corresponding bit means different states.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	float fValue; ZAux_Direct_GetAxisStopReason(handle, 0, &fValue); //read axis 0 stop reasons. Handle: controller link handle
Details	Returned state: corresponding bit means different states.

Instruction 42	ZAux_GetModbusDpos
Original	int32 __stdcall ZAux_GetModbusDpos (ZMC_HANDLE handle , int

Format	imaxaxises, float * pValueList);
Description	Rapidly read current DPOS of multi-axis. Modbus register method.
Input parameters	Parameter name: handle — Controller link handle; iaxis — Axis numbers
Output parameters	Parameter name: pfValue — DPOS coordinates value that is read, starting from 0.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get and set parameters
Details	start to read from 0 by default, it is the fastest way to read speed.

Instruction 43	**ZAux_GetModbusMpos**
Original Format	int32 __stdcall ZAux_GetModbusMpos (ZMC_HANDLE handle , int imaxaxises, float * pValueList);
Description	Rapidly read current MPOS of multi-axis. Modbus register method.
Input parameters	Parameter name: handle — Controller link handle; iaxis — Axis numbers
Output parameters	Parameter name: pfValue — Measurement coordinates value that is read, starting from 0.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get and set parameters
Details	start to read from 0 by default, it is the fastest way to read speed.

Instruction 44	**ZAux_GetModbusCurSpeed**
Original Format	int32 __stdcall ZAux_GetModbusCurSpeed (ZMC_HANDLE handle , int imaxaxises, float * pValueList);
Description	Rapidly read current speed of multi-axis. Modbus register method.
Input parameters	Parameter name: handle — Controller link handle; iaxis — Axis numbers

Output parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>pfValue</td><td>Current speed that is read, starting from 0.</td></tr></table>
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get and set parameters
Details	start to read from 0 by default, it is the fastest way to read speed.

Instruction 45	**ZAux_Direct_SetAxisAddress**
Original Format	int32 __stdcall ZAux_Direct_SetAxisAddress (ZMC_HANDLE handle, int iaxis, int iValue)
Description	Set axis address.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Controller link handle</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr><tr><td>iValue</td><td>Configured address</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Local pulse axis No. remapping

Details	1. When ZCAN expands the axis, there is an 8-bit DIP switch on the expansion board (hardware version above V1.3).
	Limited by the bus bandwidth, do not set more than 2 ZCAN expansion axes.
	And AXIS_ADDRESS must be set firstly, and then the ATYPE type of ZCAN extended axis must be set, also ATYPE must be reset after modification. See example one.
	Bit 1-4: CAN address dial code, combination value 0-15
	Digit 5-6: CAN speed dial code, different combination values have different speeds.
	Bit 7: reserved for special functions
	Bit 8: 120 ohm resistance dial code, the resistance is connected when dialing ON
	Rule:
	ZAux_Direct_SetAxisAddress(handle, axis number, (32*0)+ID)
	//The local axis interface 0 of the expansion board
	ZAux_Direct_SetAxisAddress(handle, axis number, (32*1)+ID)
	//The local axis interface 1 of the expansion board
	
	2. The axis No. is mapped to the bus drive, and the connected

	drives are mapped one by one according to the number.
	The drive numbers are arranged according to the wiring sequence, and the numbers start from 0 to the number of EtherCAT drives minus 1.
	The drive number is different from the device number. The device number includes all the devices connected to the ECAT interface, and the drive number only counts the connected drives.
	AXIS_ADDRESS must be set firstly, and then the ATYPE type of the ECAT axis must be set, and ATYPE must be set again after modification.
	Bit 0-15: drive number plus 1, 0-automatic assignment
	Bits 16-31: SLOT number (for multiple slots)
	Rule:
	ZAux_Direct_SetAxisAddress(handle, axis number, (slot number<<16)+driver number+1)
	
	3. Local pulse axis No. remapping, 4 series controllers support local pulse or encoder axis number remapping, firmware version 160608 and above support.
	When remapping, pay attention to first set the original pulse axis as a virtual axis. ATYPE must be reset after modification. See command example: Local pulse axis No. remapping
	Bit 0-15: mapped local pulse axis No.
	Bit 16-31: the upper 16 bits are all set to 1 (equivalent to the upper 16 bits in decimal = -1)
	Rule:
	ZAux_Direct_SetAtype(handle, remapped axis number, 0)
	//Set the axis type to 0, if the lower version is not set, an error will be reported
	ZAux_Direct_SetAtype(handle, local pulse axis number to be modified, 0)
	//Set the axis type to 0, if the lower version is not set, an error will be reported
	ZAux_Direct_SetAxisAddress(handle, remapped axis number, (-1<<16) + local pulse axis number to be modified)
	ZAux_Direct_SetAtype(handle, remapped axis number, 1 or 7)

Instruction 46	ZAux_Direct_GetAxisAddress
Original Format	int32 __stdcall ZAux_Direct_GetAxisAddress (ZMC_HANDLE handle, int iaxis, int *piValue)
Description	Read axis address.
Input parameters	Parameter name: handle — Controller link handle Parameter name: iaxis — Axis No.

Output parameters	Parameter name: pfValue — Description: Returned axis address
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Local pulse axis No. remapping
Details	/

Instruction 47	**ZAux_Direct_SetAxisEnable**
Original Format	int32 __stdcall ZAux_Direct_SetAxisEnable(ZMC_HANDLE handle, int iaxis, int iValue);
Description	Set axis enable.
Input parameters	Parameter name: handle — Description: Link handle Parameter name: axis — Description: Slot No. Parameter name: iValue — Description: State: 0-off, 1-on
Output parameters	One parameter
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	EtherCAT Bus axis enable

Instruction 48	**ZAux_Direct_GetAxisEnable**
Original Format	int32 __stdcall ZAux_Direct_GetAxisEnable (ZMC_HANDLE handle , int iaxis, int * piValue);
Description	Read EtherCAT axis enable state
Input parameters	Parameter name: handle — Description: Controller link handle Parameter name: iaxis — Description: Axis No.
Output parameters	Parameter name: piValue — Description: Returned enable state
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	EtherCAT Bus axis enable
Details	/

Instruction 49	**ZAux_Direct_SetInvertStep**
Original Format	int32 __stdcall ZAux_Direct_SetInvertStep(ZMC_HANDLE handle, int iaxis, int iValue)

Description

Set pulse output mode.

Input parameters

Parameter name	Description
handle	Link handle
axis	Slot No.
iValue	Mode selection, set bit by bit, default is 0

Output parameters

/

Return value

If it is successful, return value is 0, if not, please refer to error codes.

Example

Get and Set parameters

Details

Mode selection, default 0

The mode value represented by the lower 8 bits (bit 0-bit 7) is as follows:

0-3: pulse direction mode, pulse line + direction line

4-7: double pulse mode (or CW/CCW), positive pulse line + negative pulse line

8-9: AB output (customized by some controllers)

The levels corresponding to each mode are as follows:

Mode value	Description	Panasonic Setting		Mitsubishi Setting
		Pr0.06	Pr0.07	PA13
0	Pulse/direction (pulse positive logic) (+)	0	3	××01h
1	Pulse/direction (pulse negative logic) (+)	/	/	××11h
2	Pulse/direction (pulse positive logic) (-)	1	3	××01h
3	Pulse/direction (pulse negative logic) (-)	/	/	××11h
4	Double pulse (pulse negative logic) (+)	/	/	××10h
5	Double pulse (pulse negative logic) (-)	/	/	××10h
6	Double pulse (pulse positive logic) (+)	1	1	××00h (default)
7	Double pulse (pulse positive logic) (-)	0(default)	1(default)	××00h (default)

The upper 8 bits (bit 8-bit 15) indicate the direction change protection time, in microseconds: 0-255

Commonly used modes are 0, 2, 6, 7.

If the mode is not set correctly, the stepper motor may lose a step

	position when commutating. When you are not sure about the setting of the stepper motor, you can set a protection time of about 100 microseconds.

Instruction 50	ZAux_Direct_GetInvertStep
Original Format	int32 __stdcall ZAux_Direct_GetInvertStep(ZMC_HANDLE handle, int iaxis, int *piValue)
Description	Read pulse output mode.
Input parameters	Parameter name: handle — Controller link handle; iaxis — Axis No.
Output parameters	Parameter name: piValue — Returned current mode.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get and Set parameters
Details	Same as above.

Instruction 51	ZAux_Direct_GetMaxSpeed
Original Format	int32 __stdcall ZAux_Direct_GetMaxSpeed (ZMC_HANDLE handle , int iaxis, int * piValue);
Description	Read max frequency of pulse output
Input parameters	Parameter name: handle — Controller link handle; iaxis — Axis No.
Output parameters	Parameter name: piValue — Returned pulse frequency
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get and Set parameters
Details	1. Once it is found to exceed this setting value, it will be forced and an axis alarm will be generated. 2. For the encoder axis, when the setting value is lower than 500K, the encoder filter will be enabled, and when the setting value is higher than 1M, the encoder filter setting will be cancelled. The default value is 1000000 (the default pulse frequency of old firmware is 500000). 3. When the linear motor is used at a high speed, the pulse frequency is generally easy to exceed the limit, so the value should be appropriately set to a larger value.

<table>
<tr><th>Instruction 52</th><th>ZAux_Direct_SetMaxSpeed</th></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_SetMaxSpeed (ZMC_HANDLE handle , int iaxis, int iValue);</td></tr>
<tr><td>Description</td><td>Set max frequency of pulse output</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link handle</td></tr><tr><td>axis</td><td>Slot No.</td></tr><tr><td>iValue</td><td>Max pulse frequency that is set.</td></tr></table></td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Get and Set parameters</td></tr>
<tr><td>Details</td><td>1. Once it is found to exceed this setting value, it will be forced and an axis alarm will be generated.
2. For the encoder axis, when the setting value is lower than 500K, the encoder filter will be enabled, and when the setting value is higher than 1M, the encoder filter setting will be cancelled. The default value is 1000000 (the default pulse frequency of old firmware is 500000).
3. When the linear motor is used at a high speed, the pulse frequency is generally easy to exceed the limit, so the value should be appropriately set to a larger value.</td></tr>
</table>

<table>
<tr><th>Instruction 53</th><th>ZAux_Direct_GetMtype</th></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_GetMtype(ZMC_HANDLE handle, int iaxis, int *piValue)</td></tr>
<tr><td>Description</td><td>Read instruction type of current motion.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Controller link handle</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr></table></td></tr>
<tr><td>Output parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>piValue</td><td>Returned motion instruction type</td></tr></table></td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Get and Set parameters</td></tr>
<tr><td>Details</td><td>Please refer to 5.5.2.2 for details.</td></tr>
</table>

Instruction 54	ZAux_Direct_GetNtype

Original Format	int32 __stdcall ZAux_Direct_GetNtype(ZMC_HANDLE handle, int iaxis, int *piValue)
Description	Read the next instruction type of current motion.
Input parameters	Parameter name: handle — Controller link handle Parameter name: iaxis — Axis No.
Output parameters	Parameter name: piValue — Returned motion instruction type
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get and Set parameters
Details	Please refer to 5.5.2.2 for details.

Instruction 55	**ZAux_Direct_SetOffpos**
Original Format	int32 __stdcall ZAux_Direct_SetOffpos (ZMC_HANDLE handle , int iaxis, int fValue);
Description	Set modifying offset position.
Input parameters	Parameter name: handle — Link handle Parameter name: axis — Axis No. Parameter name: fValue — Offset value that is set
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Axis Coordinates Configuration
Details	Relative offsets modify all coordinates and have no effect on motion already running/entering the buffer. After the modification is completed, OFFPOS will be restored to 0.

Instruction 56	**ZAux_Direct_GetOffpos**
Original Format	int32 __stdcall ZAux_Direct_GetOffpos (ZMC_HANDLE handle , int iaxis, int *piValue);
Description	Read modified offset position.
Input parameters	Parameter name: handle — Controller link handle Parameter name: iaxis — Axis No.

Output parameters	Parameter name: fValue; Description: Obtained offset value
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Axis Coordinates Configuration
Details	After the modification is completed, OFFPOS will be restored to 0.

Instruction 57	**ZAux_Direct_Defpos**
Original Format	int32 __stdcall ZAux_Direct_Defpos(ZMC_HANDLE handle, int iaxis, float pfDpos)
Description	The current position is as one new absolute position value according to the definition.
Input parameters	Parameter name: handle; Description: Link handle Parameter name: axis; Description: Axis No. Parameter name: fValue; Description: Distance coordinates that is set
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Axis Coordinates Configuration
Details	No effect on motion already running/entering the buffer.

Instruction 58	**ZAux_Direct_GetFe**
Original Format	int32 __stdcall ZAux_Direct_GetFe (ZMC_HANDLE handle , int iaxis, float*pfValue);
Description	Read follow-up errors.
Input parameters	Parameter name: handle; Description: Controller link handle Parameter name: iaxis; Description: Axis No.
Output parameters	Parameter name: fValue; Description: Obtained offset value
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Axis Coordinates Configuration
Details	Follow-up error = controller DPOS – encoder MPOS

Instruction 59	**ZAux_Direct_GetFeLimit**

Original Format	int32 __stdcall ZAux_Direct_GetFeLimit (ZMC_HANDLE handle, int iaxis, float*pfValue);
Description	Read the following error at the maximum allowed. (Requires closed-loop firmware to take effect, reserved function)
Input parameters	Parameter name: handle — Controller link handle; iaxis — Axis No.
Output parameters	Parameter name: fValue — Max allowable follow-up error values
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	float fValue; ZAux_Direct_GetFeLimit(handle, 0, &fValue); //read the maximum allowable following error of axis 0. handle: controller link handle
Details	/

Instruction 60	**ZAux_Direct_SetFeLimit**
Original Format	int32 __stdcall ZAux_Direct_SetFeLimit (ZMC_HANDLE handle , int iaxis, float fValue);
Description	Set following error at the maximum allowed. (Requires closed-loop firmware to take effect, reserved function)
Input parameters	Parameter name: handle — Link mark; axis — Axis No.; fValue — Max allowable follow-up error values
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.

Example	float fValue=100;
	ZAux_Direct_SetFeLimit(handle, 0, fValue);
	//set the maximum allowable following error of axis 0 as 100
	handle: controller continuous handle

Details	/

Instruction 61	**ZAux_Direct_GetFeRange**
Original Format	int32 __stdcall ZAux_Direct_GetFeRange (ZMC_HANDLE handle , int iaxis, int*pfValue);
Description	Read the following error when in alarm. (Requires closed-loop

	firmware to take effect, reserved function)
Input parameters	Parameter name: handle — Description: Controller link mark; Parameter name: iaxis — Description: Axis No.
Output parameters	Parameter name: fValue — Description: Follow-up error values when in alarm.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Int iValue; ZAux_Direct_GetFeRanget(handle, 0, & iValue); //read follow-up error when axis 0 alarms. handle: controller continuous handle
Details	/

Instruction 62	**ZAux_Direct_SetFeRange**
Original Format	int32 __stdcall ZAux_Direct_SetFeRange (ZMC_HANDLE handle , int iaxis, float fValue)
Description	Set following error when in alarm. (Requires closed-loop firmware to take effect, reserved function)
Input parameters	Parameter name: handle — Description: Link mark; Parameter name: axis — Description: Axis No.; Parameter name: fValue — Description: Follow-up error values when in alarm
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Int iValue=100; ZAux_Direct_SetFeRanget(handle, 0, iValue); //set the maximum allowable following error of axis 0. handle: controller link handle
Details	/

Instruction 63	**ZAux_Direct_GetAllAxisPara**
Original Format	int32 __stdcall ZAux_Direct_GetAllAxisPara(ZMC_HANDLE handle, const char *sParam,int imaxaxis,float *pfValue)
Description	Read multi-axis axis parameters at once time.

Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>sParam</td><td>Character string name of Basic grammar parameter</td></tr><tr><td>imaxaxis</td><td>Axis numbers</td></tr></table>
Output parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>fValue</td><td>Returned parameter values</td></tr></table>
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	float fAtype[8]; ZAux_Direct_GetAllAxisPara(handle,"ATYPE",8,fAtype); //get the axis type of axis 0-7
Details	/

Instruction 64	**ZAux_Direct_GetAllAxisInfo**
Original Format	int32 __stdcall ZAux_Direct_GetAllAxisInfo(ZMC_HANDLE handle,int imaxaxis,int *IdleStatus,float *DposStatus,float *MposStatus, int *AxisStatus)
Description	Read basic axis status for multiple axes. The value is displayed in decimal, and the state is judged by the corresponding bit in binary.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>imaxaxis</td><td>Axis numbers</td></tr></table>
Output parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>IdleStatus</td><td>Axis running state</td></tr><tr><td>DposStatus</td><td>Axis command coordinates</td></tr><tr><td>MposStatus</td><td>Axis measurement coordinates</td></tr><tr><td>AxisStatus</td><td>Axis state</td></tr></table>
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	int iIdle[8]; Float iDpos[8]; Float iMpos[8]; int iStatus[8]; ZAux_Direct_GetAllAxisPara(handle,,8,iIdle,iDpos,iMpos,iStatus); //get the axis type of axis 0-7
Details	Axis state, bit by bit, please refer to 5.1.2.1 for details.

Instruction 65	**ZAux_Direct_SetParam**
Original	int32 __stdcall ZAux_Direct_SetParam(ZMC_HANDLE handle,const

Format	char *sParam,int iaxis, float fset)
Description	General parameters modify function "sParm" to fill in parameter name.
Input parameters	Parameter name: handle — Link mark sParam — Character string name of Basic grammar parameter. axis — Axis No. fset — Parameter values
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get basic parameters by character string.
Details	The upper computer uses the Basic instruction to obtain the controller value, which is often used for the unencapsulated Basic instruction of the upper computer. Using this function instead of some instruction encapsulating can quickly obtain the corresponding values.

Instruction 66	**ZAux_Direct_GetParam**
Original Format	int32 __stdcall ZAux_Direct_GetParam(ZMC_HANDLE handle,const char *sParam, int iaxis, float *pfValue)
Description	Parameter, general parameter reads function "sparam", filling in the parameter name.
Input parameters	Parameter name: handle — Link mark sParam — Character string parameter name in Basic grammar iaxis — Axis No.
Output parameters	Parameter name: pfValue — Returned parameter value
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get basic parameters by character string.
Details	The upper computer uses the Basic instruction to obtain the controller value, which is often used for the unencapsulated Basic instruction of the upper computer. Using this function instead of some instruction encapsulating can quickly obtain the corresponding values.

Instruction 67	**ZAux_Direct_GetEndMove**

Original Format	int32 __stdcall ZAux_Direct_GetEndMove(ZMC_HANDLE handle, int iaxis, float *pfValue)
Description	Read the end position of the current motion, the unit is units.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr></table>
Output parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>pfValue</td><td>Returned parameter value</td></tr></table>
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Float fValue; ZAux_Direct_GetEndMove(handle,0,&fValue); //get the end position of axis 0 current motion
Details	For non-fixed distance movements such as VMOVE and DATUM, ENDMOVE is not accurate, or it changes all the time.

Instruction 68	**ZAux_Direct_GetEndMoveBuffer**
Original Format	int32 __stdcall ZAux_Direct_GetEndMoveBuffer(ZMC_HANDLE handle, int iaxis, float *pfValue)
Description	Read the end position of the current motion and motion in buffer which can be used to convert absolute and relative, the unit is units.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr></table>
Output parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>pfValue</td><td>Return to end target positions of current motion and motions in buffer.</td></tr></table>
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Float fValue; **ZAux_Direct_GetEndMoveBuffer**(handle,0,&fValue); //get the end position of axis 0 motions in buffer.
Details	For non-fixed distance movements such as VMOVE and DATUM, ENDMOVE_BUFFER is not accurate, or it changes all the time. After the circular coordinate instruction REP_OPTION is used, ENDMOVE_BUFFER decreases in turn according to the REP_DIST setting value in REP_OPTION mode, that is, the minimum precision is REP_DIST (mode 1) or 2 times REP_DIST (mode 0)

Instruction 69	ZAux_Direct_GetLoaded
Original Format	int32 __stdcall ZAux_Direct_GetLoaded (ZMC_HANDLE handle , int iaxis,int *pfValue);
Description	Read whether there is motion buffer except the current motion.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.
Output parameters	Parameter name: pfValue — Returned value: -1: there is no remain motion buffer 0: there is remain motion buffer
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Int iValue; ZAux_Direct_GetEndMoveBuffer(handle,0,&iValue); //get whether axis 0 has motion buffer or not.
Details	This instruction cannot judge whether the axis is stopped, please use IDLE instruction to judge stop state.

Instruction 70	ZAux_Direct_GetMovesBuffered
Original Format	int32 __stdcall ZAux_Direct_GetMoveBuffered (ZMC_HANDLE handle , int iaxis, int * piValue);
Description	Read the number of motions buffered currently
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.
Output parameters	Parameter name: pfValue — Buffers
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Int iValue; ZAux_Direct_GetMoveBuffered(handle,0,&iValue); //get the number of motions that has been buffered of axis 0
Details	Do not subtract the parameter MOVES_BUFFERED from the total buffer space to determine whether there is remaining buffer space. Special motion instructions may occupy multiple buffer spaces. It is more accurate to use the REMAIN_BUFFER state function to judge.

Instruction 71	ZAux_Direct_GetMoveCurmark

Original Format	int32 __stdcall ZAux_Direct_GetMoveCurmark (ZMC_HANDLE handle , int iaxis, int * piValue);
Description	Read MOVE_MARK mark that is running currently.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.
Output parameters	Parameter name: piValue — Current Mark mark No.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Int iValue; ZAux_Direct_GetMoveCurmark(handle,0,&iValue); //get the Mark mark No. that is running of axis 0
Details	/

Instruction 72	**ZAux_Direct_SetMoveMark**
Original Format	int32 __stdcall ZAux_Direct_SetMoveMark (ZMC_HANDLE handle , int iaxis, int iValue);
Description	Set the MARK label of the next motion instruction to be called. This mark No. will be written into the motion buffer together with the motion command. Each time a motion command is called, MOVE_MARK will automatically increase by one.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.; piValue — MARK value that is set.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	ZAux_Direct_GetMoveCurmark(handle,0,1); //set next command's Mark as 1 of axis 0
Details	/

Instruction 73	**ZAux_Direct_GetRemain**
Original Format	int32 __stdcall ZAux_Direct_GetRemain(ZMC_HANDLE handle,int iaxis, float *pfValue)
Description	Read remain distance of current motion, the unit is units.

Input parameters	Parameter name: handle — Description: Link mark Parameter name: iaxis — Description: Axis No.
Output parameters	Parameter name: piValue — Description: Returned remain distance
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Float fValue; ZAux_Direct_GetRemain(handle,0,&fValue); //get motion remain distance of axis 0 current motion
Details	/

Instruction 74	**ZAux_Direct_GetRemain_LineBuffer**
Original Format	int32 __stdcall ZAux_Direct_GetRemain_LineBuffer (ZMC_HANDLE handle , int iaxis, int *piValue);
Description	Axis remaining buffers, it calculates according to linear segment, REMAIN_BUFFER the unique one that can add AXIS and can be obtained through Zaux_DirectCommand.
Input parameters	Parameter name: handle — Description: Link mark Parameter name: iaxis — Description: Axis No.
Output parameters	Parameter name: piValue — Description: Remaining linear buffers
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Int iValue; ZAux_Direct_GetRemain_LineBuffer(handle,0,&iValue); //get axis 0 current remaining linear motion buffers.
Details	/

Instruction 75	**ZAux_Direct_GetRemain_Buffer**
Original Format	int32 __stdcall ZAux_Direct_GetRemain_Buffer(ZMC_HANDLE handle, int iaxis, int *piValue)
Description	Read remaining axis motion buffers, it is obtained from the most complex remaining space arc buffers.
Input parameters	Parameter name: handle — Description: Link mark Parameter name: iaxis — Description: Axis No.

Output parameters	Parameter name: piValue — Description: Remaining space arc buffers
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get remain buffers.
Details	/

Instruction 76	**ZAux_Direct_GetVectorBuffered**
Original Format	int32 __stdcall ZAux_Direct_GetVectorBuffered(ZMC_HANDLE handle , int iaxis, float *pfValue);
Description	Get distance that are completed by the current motion and buffer motions.
Input parameters	Parameter name: handle — Description: Link mark Parameter name: iaxis — Description: Axis No.
Output parameters	Parameter name: piValue — Description: Remaining space arc buffers
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Float fValue; ZAux_Direct_GetMoveCurmark(handle,0,&fValue); //get the distance that axis 0's current movement and buffer movement have completed.
Details	For multi-axis interpolation, it is vector distance.

Instruction 77	**ZAux_Direct_MoveOp**
Original Format	int32 __stdcall ZAux_Direct_MoveOp(ZMC_HANDLE handle, int iaxis, int ioutnum, int ivalue);
Description	Write output assignment into motion buffer.
Input parameters	Parameter name: handle — Description: Link mark Parameter name: iaxis — Description: Axis No. Parameter name: ioutnum — Description: IO No. Parameter name: ivalue — Description: IO state
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Example 1: ZAux_Direct_Single_Move(handle, 0, 100);//axis 0 moves 100 in

	positive diection.
	ZAux_Direct_MoveOp(handle, 0,1, 1);//set the state of out1 to 1 after axis 0 moves forward 100
	Example 2: Continuous Position Latching

Details	When this instruction LOAD is executed, it does not perform any movement, but only operates the output port. The syntax is the same as the OP command. In normal mode, the error is within one scan period, and all controllers can be used. Precise position output mode, the error is within 1 microsecond. This is valid in 4xx series with firmware above 20170421. 1. Only the OP port that supports hardware comparison output supports the precise output function. 2. Each precise output MOVE_OP that takes effect needs an interval of 1 cycle to continue to take effect. During this interval, the new MOVE_OP automatically uses the normal method. After this interval, the new MOVE_OP can continue to take effect. For continuous MOVE_OP, only the first one takes effect because there is no interval time. (Some controllers can trigger multiple precise outputs at the same time, please refer to the controller hardware manual for details, for example, the first 8 output ports of ZMC420SCAN support precise output, and each output port can use precise output at the same time) 3. In the case that the controller supports independent OP ports, different OP ports will not conflict with the MOVE_OP precision output of multiple axes. But in the case where the precise output function of the OP port is not independent, using the precision function at the same time may conflict. 4. The precise function of MOVE_OP is based on the BASE main axis. When interpolating multiple axes, the slave axis whose ATYPE type is different from that of the BASE main axis cannot guarantee accurate position output.

Instruction 78	**ZAux_Direct_MoveOpMulti**
Original Format	int32 __stdcall ZAux_Direct_MoveOpMulti(ZMC_HANDLE handle, int iaxis, int ioutnumfirst, int ioutnumend, int ivalue)
Description	Write multiple output instructions into motion buffer.
Input parameters	

Parameter name	Description
handle	Link mark
iaxis	Axis No.
ioutnumfirst	The first output channel to be operated
ioutnumend	The last output channel to be operated
ivalue	IO state

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.

Example	ZAux_Direct_Single_Move(handle, 0, 100);//axis 0 moves 100 in the positive direction
	ZAux_Direct_MoveOpMulti((handle, 0,0,8, 1);
	//after axis moves 100 in the positive direction, set out0-out4 state as 1

Details	/

Instruction 79	**ZAux_Direct_MoveOp2**
Original Format	int32 __stdcall ZAux_Direct_MoveOp2(ZMC_HANDLE handle, int iaxis, int ioutnum, int ivalue, int iofftimems)
Description	Write output instructions into motion buffer. And invert after iofftimems time to produce pulse output.
Input parameters	Parameter name: Description handle: Link mark iaxis: Axis No. ioutnum: IO No. ivalue: IO state iofftimems: invert after iofftimems time to produce pulse output.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Fly-shooting
Details	When this instruction LOAD is executed, it does not perform any movement, but only operates the output port. A single axis only supports one pulse output at a time, and the second MOVE_OP2 instruction will automatically turn off the pulse of the previous instruction.

Instruction 80	**ZAux_Direct_MoveAout**
Original Format	int32 __stdcall ZAux_Direct_MoveAout(ZMC_HANDLE handle, int iaxis, int ioutnum, float fvalue)
Description	Add analog output instructions in motion buffer.

<table>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link mark</td></tr>
<tr><td>iaxis</td><td>Axis No.</td></tr>
<tr><td>ioutnum</td><td>AOUT No.</td></tr>
<tr><td>ivalue</td><td>IO state</td></tr>
<tr><td>fvalue</td><td>Output analog values.
12-bit scale value range 0~4095 corresponds to 0~10V voltage.
16-bit scale value range 0~65536 corresponds to 0~10V voltage.</td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>ZAux_Direct_Single_Move(handle, 0, 100);//axis 0 moves 100 in the positive direction
ZAux_Direct_MoveAout(handle, 0,1, 2048);//after axis 0 moved 100 forward, set DA1 state to 2048.</td></tr>
<tr><td>Details</td><td>This instruction LOAD does not perform any movement, only modifies the AOUT value. The type of this instruction is consistent with MOVE_OP.</td></tr>
</table>

<table>
<tr><td>Instruction 81</td><td>ZAux_Direct_MoveDelay</td></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_MoveDelay(ZMC_HANDLE handle, int iaxis, int itimems)</td></tr>
<tr><td>Description</td><td>Add delay in motion buffer.</td></tr>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link mark</td></tr>
<tr><td>iaxis</td><td>Axis No.</td></tr>
<tr><td>itimems</td><td>Delay time MS</td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Example 1:
ZAux_Direct_Single_Move(handle, 0, 100);//axis 0 moves 100 forward
ZAux_Direct_MoveDelay(handle, 0,1000);//after axis 0 moves 100 forward, delay 1s, then execute the next buffer.</td></tr>
</table>

	Example 2: Continuous Position Latch
Details	When former motion instructions end, the speed will automatically become 0.

Instruction 82	**ZAux_Direct_MoveWait**
Original Format	int32 __stdcall ZAux_Direct_MoveWait(ZMC_HANDLE handle,uint32 base_axis,char *paraname,int inum,int Cmp_mode,float fvalue)
Description	Add one condition judge blocking in motion buffer, it is valid in firmware above 150802 or XPLC160405 above versions.
Input parameters	Parameter name: Description handle: Link mark base_axis: Axis No. of motion axis paraname: Character string names: DPOS MPOS IN AIN VPSPEED MSPEED MSPEED MODBUS_REG MODBUS_IEEE MODBUS_BIT NVRAM VECT_BUFFED REMAIN inum: Axis parameter No. Cmp_mode: Comparison mode: 1: ≥. 0: =. -1: ≤ Fvalue: Comparison values
Return value	If it is successful, return value is 0, if not, please refer to error codes.

Example	Example 1:
	ZAux_Direct_Single_Move(handle, 0, 100);//axis 0 moves 100 forward
	ZAux_Direct_MoveWait(handle, 0,"IN",1,0,1);//after axis 0 moves 100 forward, waiting for IN1 signal, then execute the next buffer.
	Example 2: position latch

Details	This instruction LOAD does not do any movement when it is executed, it only waits for the specified condition to be met. When former motion instructions end, the speed will automatically become 0.

Instruction 83	**ZAux_Direct_MoveTask**
Original Format	int32 __stdcall ZAux_Direct_MoveTask(ZMC_HANDLE handle,uint32 base_axis,uint32 tasknum,char *labelname)
Description	The motion buffer is added to a TASK task. When the task has been started, an error will be reported, but it will not affect the program execution. Some control cards support this function

Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>base_axis</td><td>Axis No. of motion axis</td></tr><tr><td>tasknum</td><td>Task No.</td></tr><tr><td>lablename</td><td>Triggered global SUB function/lable</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	This instruction LOAD does not do any movement when it is executed, it only turns on the task, and this command's type is same as MOVE_OP. When task has been opened, there is error in ZDevelop inside, but no influence on program execution.

Instruction 84	**ZAux_Direct_MovePara**
Original Format	int32 __stdcall ZAux_Direct_MovePara(ZMC_HANDLE handle,uint32 base_axis,char *paraname,uint32 iaxis,float fvalue)
Description	To write parameter settings into the motion buffer, firmware version 20170503 or above is required.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>base_axis</td><td>Axis No. of motion axis</td></tr><tr><td>paraname</td><td>Parameter name in character string</td></tr><tr><td>iaxis</td><td>Axis No. that modifies the parameter</td></tr><tr><td>fvalue</td><td>Parameter set value.</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.

Example	Example: The total movement of axis 0 is 300. When the movement reaches 100, the speed is changed to 100 online to complete the total movement.
	ZAux_Direct_SetMerge(handle, 0,1);//Open continuous interpolation
	ZAux_Direct_Single_Move(handle, 0, 100);//axis 0 positive movement 100
	ZAux_Direct_MovePara(handle, 0,"SPEED",0,100);//After the axis 0 moves forward 100, the speed is changed to 100

	ZAux_Direct_Single_Move(handle, 0, 200);//axis 0 positive movement 200
Details	This instruction LOAD does not perform any movement, but only modifies the parameters. The type of this instruction is consistent with MOVE_OP.

Instruction 85	**ZAux_Direct_MovePw**
Original Format	int32 __stdcall ZAux_Direct_MovePwm(ZMC_HANDLE handle,uint32 base_axis,uint32 pwm_num,float pwm_duty,float pwm_freq)
Description	To PWM output into the motion buffer, firmware version 20170503 or above is required.
Input parameters	Parameter name: Description handle: Link mark base_axis: Axis No. of motion axis Pwm_num: PWM No. Pwm_duty: Duty cycle that is set Pwm_freq: The frequency that is set
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	How to use PWM
Details	This instruction LOAD does not perform any movement, but only modifies the parameters. The type of this instruction is consistent with MOVE_OP.

Instruction 86	**ZAux_Direct_MoveSynmove**
Original Format	int32 __stdcall ZAux_Direct_MoveSynmove(ZMC_HANDLE handle,uint32 base_axis, uint32 iaxis, float fdist, uint32 ifsp)
Description	Trigger other axes motions to write into buffer when in motion, current axis waits.
Input parameters	Parameter name: Description handle: Link mark base_axis: Axis No. of motion axis iaxis: Synchronous axis No. fdist: Synchronous motion distance ifsp: Whether the synchronous motion is SP motion. 0: NO, 1: YES.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error

	codes.

Example	Example: The total movement of axis 0 is 300. When the movement reaches 100, trigger axis 1 to move 100. After the movement of axis 1 is completed, axis 0 continues to complete the rest of the movement
	ZAux_Direct_SetMerge(handle, 0,1);//Open continuous interpolation
	ZAux_Direct_Single_Move(handle, 0, 100);//axis 0 positive movement 100
	ZAux_Direct_MoveSynmove(handle, 0,1,100,0);//Axis 0 moves forward 100 and starts axis 1 movement
	ZAux_Direct_Single_Move(handle, 0, 200);//axis 0 positive movement 200

Details	This instruction LOAD does not perform any movement, but only modifies the parameters. The type of this instruction is consistent with MOVE_OP.

Instruction 87	**ZAux_Direct_MoveASynmove**
Original Format	int32 __stdcall ZAux_Direct_MoveASynmove(ZMC_HANDLE handle, uint32 base_axis, uint32 iaxis,float fdist,uint32 ifsp)
Description	Trigger other axes motions when in motion, current axis doesn't wait. It is valid in 20170503 above firmware versions.
Input parameters	Parameter name: Description handle: Link mark base_axis: Axis No. of motion axis iaxis: Triggered axis No. fdist: Triggered motion distance ifsp: Whether the trigger motion is SP motion.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.

Example	Example: The total movement of axis 0 is 300. When the movement reaches 100, trigger axis 1 to move 100. After the movement of axis 1 is completed, axis 0 continues to complete the rest of the movement
	ZAux_Direct_SetMerge(handle, 0,1);//Open continuous interpolation
	ZAux_Direct_Single_Move(handle, 0, 100);//axis 0 positive movement 100
	ZAux_Direct_MoveSynmove(handle, 0,1,100,0);//Axis 0 moves forward 100 and starts axis 1 movement
	ZAux_Direct_Single_Move(handle, 0, 200);//axis 0 positive

	movement 200
Details	This instruction LOAD does not perform any movement, but only modifies the parameters. The type of this instruction is consistent with MOVE_OP.

Instruction 88	**ZAux_Direct_MoveTable**
Original Format	int32 __stdcall ZAux_Direct_MoveTable(ZMC_HANDLE handle,uint32 base_axis,uint32 table_num,float fvalue)
Description	Modify Table value in motion buffer, in this way, Table value can be modified while moving.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>base_axis</td><td>Axis No. of motion axis</td></tr><tr><td>Table_num</td><td>TABLE No.</td></tr><tr><td>fvalue</td><td>Value that is set</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.

Example	Example: the total movement of axis 0 is 300, when the movement reaches 100, trigger modification table(1)=10.
	ZAux_Direct_Single_Move(handle, 0, 100);//axis 0 positive movement 100
	ZAux_Direct_MoveTable(handle, 0,1,10);//After axis 0 moves 100 forwards, wait until IN1 has a signal, then execute the next buffer
	ZAux_Direct_Single_Move(handle, 0,200);//axis 0 positive movement 200

Details	This instruction LOAD does not perform any movement, but only modifies the parameters. The type of this instruction is consistent with MOVE_OP.

Instruction 89	**ZAux_Direct_MoveCancel**
Original Format	int32 __stdcall ZAux_Direct_MoveCancel(ZMC_HANDLE handle, int32 base_axis, int32 Cancel_Axis,int iMode)
Description	Cancel other axes motions in motion buffer, in this way, other axes motions can be canceled while moving.

Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>base_axis</td><td>Axis No. of motion axis</td></tr><tr><td>Cancel_axis</td><td>Axis No. of stop axis</td></tr><tr><td>iMode</td><td>Stop mode: 0: cancel current motion 1: cancel buffer motion 2: cancel current motion and buffered motion 3: interrupt pulse sending immediately</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.

Example	Example: the total movement of axis 0 is 300, and when the movement reaches 100, stop axis 1
	ZAux_Direct_Single_Move(handle, 0, 100);//axis 0 positive movement 100
	ZAux_Direct_MoveCancel(handle,0,1,0)//Insert the stop axis 1 instruction in the axis 0 buffer
	ZAux_Direct_Single_Move(handle, 0,200);//axis 0 positive movement 200

Details	/

Instruction 90	**ZAux_Direct_MoveModify**
Original Format	int32 __stdcall ZAux_Direct_MoveModify(ZMC_HANDLE handle, int iaxis, float pfDisance)
Description	Modify the target position of the previous movement, the effect is the same as MOVEABS when there is no movement in front. Using MOVEMODIFY during continuous interpolation will break velocity continuity.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr><tr><td>pfDistance</td><td>Motion distance</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Example: axis 0 moves 300 at a very slow speed, after 1 second,

	modify the total movement of axis 0 from 300 to 200
	ZAux_Direct_Single_Move(handle, 0, 300);//axis 0 positive movement 100
	Sleep(1000);
	ZAux_Direct_MoveModify(handle, 0,200);//modify the target position of axis 0

Details	Same effect as MOVEABS when there is no movement in front, but will not enter the motion buffer Using MOVEMODIFY during continuous interpolation will break velocity continuity. MOVEMODIFY is not necessarily linear interpolation for multi-axis motion at the same time.

Instruction 91	**ZAux_Direct_MoveLimit**
Original Format	int32 __stdcall ZAux_Direct_MoveLimit(ZMC_HANDLE handle, int iaxis,float limitspeed)
Description	Add speed limit at the end position of current motion to decelerate corner compulsively.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr><tr><td>limitspeed</td><td>Limited speed</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.

Example	ZAux_Direct_SetMerge(handle, 0,1);//Open continuous interpolation
	ZAux_Direct_Single_Move(handle, 0, 200);//axis 0 positive movement 200
	ZAux_Direct_MoveLimit(handle, 0,100);//Slow down to 100 between two movements
	ZAux_Direct_Single_Move(handle, 0, 200);//axis 0 positive movement 200

Details	/

Instruction 92	**ZAux_Direct_Single_Move**
Original Format	int32 __stdcall ZAux_Direct_Single_Move(ZMC_HANDLE handle, int iaxis, float fdistance)
Description	Single-axis relative motion.

Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr><tr><td>fdistance</td><td>Distance</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Single-axis Point to Point
Details	/

Instruction 93	**ZAux_Direct_Single_MoveAbs**
Original Format	int32 __stdcall ZAux_Direct_Single_MoveAbs(ZMC_HANDLE handle, int iaxis, float fdistance)
Description	Single-axis absolute motion.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr><tr><td>fdistance</td><td>Absolute distance</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Single-axis Point to Point
Details	/

Instruction 94	**ZAux_Direct_Single_Vmove**
Original Format	int32 __stdcall ZAux_Direct_Single_Vmove(ZMC_HANDLE handle, int iaxis, int idir)
Description	Single-axis continuous motion instruction, it moves towards one direction continuously.

Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr><tr><td>idir</td><td>Direction: -1: negative, 1-positive</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Single-axis Continuous Motion
Details	When the previous VMOVE movement does not stop, this VMOVE command will automatically replace the previous VMOVE command and modify the direction, so there is no need to CANCEL the previous VMOVE command.

Instruction 95	**ZAux_Direct_GetJogSpeed**
Original Format	int32 __stdcall ZAux_Direct_GetJogSpeed (ZMC_HANDLE handle , int iaxis, float *pfValue);
Description	Read Jog speed when in FastJog motion.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr></table>
Output parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>pfValue</td><td>Returned JOG speed value</td></tr></table>
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Fast Jog (IN Control)
Details	When REV_JOG/FWD_JOG is set, when the corresponding input point is pressed and the current input state is maintained, the motor will move at the speed set by this function, and the motion will stop when the input point is released.

Instruction 96	**ZAux_Direct_SetJogSpeed**
Original Format	int32 __stdcall ZAux_Direct_SetJogSpeed (ZMC_HANDLE handle , int iaxis, float fValue);
Description	Set Jog speed when in FastJog motion.

Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr><tr><td>fValue</td><td>Speed value that is set</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Fast Jog (IN Control)
Details	When REV_JOG/FWD_JOG is set, when the corresponding input point is pressed and the current input state is maintained, the motor will move at the speed set by this function, and the motion will stop when the input point is released.

Instruction 97	**ZAux_Direct_GetFastJog**
Original Format	int32 __stdcall ZAux_Direct_GetFastJog (ZMC_HANDLE handle , int iaxis, int*piValue);
Description	Read axis fast jog speed mapping input point. When the set value is -1, it doesn't take effect.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr></table>
Output parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>piValue</td><td>Returned JOG input No.</td></tr></table>
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Fast Jog (IN Control)
Details	If fast jog input is set, speed is given by “SPEED” parameter. If no, speed is given by “JOGSPEED” parameter.

Instruction 98	**ZAux_Direct_SetFastJog**
Original Format	int32 __stdcall ZAux_Direct_SetFastJog (ZMC_HANDLE handle , int iaxis, int iValue);
Description	Set axis fast jog speed mapping input point. When the set value is -1, it doesn't take effect.

Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr><tr><td>iValue</td><td>Set fast JOG input No.</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Fast Jog (IN Control)
Details	If fast jog input is set, speed is given by "SPEED" parameter. If no, speed is given by "JOGSPEED" parameter.

Instruction 99	**ZAux_Direct_SetFwdJog**
Original Format	int32 __stdcall ZAux_Direct_SetFwdJog (ZMC_HANDLE handle , int iaxis, int iValue);
Description	Set JOG input point in positive direction. When the set value is -1, it doesn't take effect.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr><tr><td>iValue</td><td>Set forward JOG input No.</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Fast Jog (IN Control)
Details	When there is signal, corresponding axis moves forward at the speed of JOGSPEED.

Instruction 100	**ZAux_Direct_GetFwdJog**
Original Format	int32 __stdcall ZAux_Direct_GetFwdJog(ZMC_HANDLE handle, int iaxis, int *piValue);
Description	Read forward JOG input No. When the set value is -1, it doesn't take effect.

Input parameters	Parameter name: handle — Description: Link mark Parameter name: iaxis — Description: Axis No.
Output parameters	Parameter name: piValue — Description: Returned forward JOG input No.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Fast Jog (IN Control)
Details	When there is signal, corresponding axis moves forward at the speed of JOGSPEED.

Instruction 101	**ZAux_Direct_GetRevJog**
Original Format	int32 __stdcall ZAux_Direct_GetRevJog (ZMC_HANDLE handle, int iaxis, int * piValue);
Description	Read negative JOG input No., -1 means invalid.
Input parameters	Parameter name: handle — Description: Link mark Parameter name: iaxis — Description: Axis No.
Output parameters	Parameter name: piValue — Description: Returned inverse JOG input No.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Fast Jog (IN Control)
Details	When there is signal, corresponding axis moves inverse at the speed of JOGSPEED. When REV_JOG and FWD_JOG both have signal input, axis moves according to FWD_JOG.

Instruction 102	**ZAux_Direct_SetRevJog**
Original Format	int32 __stdcall ZAux_Direct_SetRevJog (ZMC_HANDLE handle, int iaxis, int iValue);
Description	Set JOG input No. in negative direction. When the set value is -1, it doesn't take effect.
Input parameters	Parameter name: handle — Description: Link mark Parameter name: iaxis — Description: Axis No. Parameter name: iValue — Description: Set inverse JOG input No.

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Fast Jog (IN Control)
Details	When there is signal, corresponding axis moves inverse at the speed of JOGSPEED. When REV_JOG and FWD_JOG both have signal input, axis moves according to FWD_JOG.

Instruction 103	**ZAux_Direct_Single_Datum**
Original Format	int32 __stdcall ZAux_Direct_Single_Datum(ZMC_HANDLE handle, int iaxis, int imode)
Description	Single-axis homing motion
Input parameters	Parameter name: handle — Link mark iaxis — Axis No. imode — Finding origin mode (please refer to details)
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Single-axis Homing Motion
Details	mode: find the origin mode, adding 10 means turn around and look back after hitting the limit, and will not stop when it hits the limit, for example, 13 = mode 3 + limit turn and look back, it is used when the origin is in the middle.

Mode	Description
0	Clear error states of all axes.
1	Axis runs forward at the speed of CREEP until signal Z appeared, it will directly stop when meeting limit switch, DPOS value will be reset to 0, at the same time, modify MPOS.
2	Axis runs reverse at the speed of CREEP until signal Z appeared, it will directly stop when meeting limit switch, DPOS value will be reset to 0, at the same time, modify MPOS.
3	Axis runs forward at the speed of SPEED, until meeting origin switch, then axis runs reverse at the speed of CREEP until away from origin switch. When in the finding origin process, it will directly stop when meeting negative limit switch, when in the creeping process, it will directly stop when meeting positive position limit. DPOS value will be reset to 0, at the same time, modify MPOS.
4	Axis runs reverse at the speed of SPEED, until meeting origin switch, then axis runs forward at the speed of CREEP until away from origin switch. When in the finding origin process, it will directly stop when meeting negative limit switch, when in the creeping process, it will directly stop when meeting positive position limit. DPOS value will be reset to 0, at the same time, modify MPOS.
5	Axis runs forward at the speed of SPEED, until meeting origin switch, then axis runs reverse at the speed of CREEP until away from origin switch. Then, continuing at CREEP speed forward until meeting signal Z. It stops immediately when met limit switch. DPOS value will be reset to 0, at the same time, modify MPOS.
6	Axis runs reverse at the speed of SPEED, until meeting origin switch, then axis runs forward at the speed of CREEP until away from origin switch. Then, continuing at CREEP speed reverse until meeting signal Z. It stops immediately when met limit switch. DPOS value will be reset to 0, at the same time, modify MPOS.
8	Axis runs forward at SPEED speed, until meeting origin switch, it will stop immediately when met limit switch.
9	Axis runs reverse at SPEED speed, until meeting origin switch, it will stop immediately when met limit switch.
21	Use EtherCAT drive homing function, now mode 2 is valid, Set drive homing mode (6098h), default 0 means using drive current homing mode. Using axis SPEED, CREEP, ACCEL and DECEL to multiple UNITS, then automatically set drive 6099h and 609Ah. Action sequence: 6098 homing mode – 6099 speed – 609A acceleration – 6060 switch to

Instruction 104	**ZAux_BusCmd_Datum**
Original Format	int32 __stdcall ZAux_BusCmd_Datum (ZMC_HANDLE handle, uint32 iaxis , uint32 homemode);
Description	Bus driver homing
Input parameters	

Parameter name	Description
handle	Link mark
iaxis	Axis No.
Homemode	Homing mode, check driver manual

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	Set the drive's zero return mode (6098h), the default 0 means use the drive's current zero return mode. Use the SPEED, CREEP, ACCEL, DECEL of the axis, multiplied by UNITS to automatically set the drive's 6099h, 609Ah Action sequence: 6098 homing mode → 6099 Speed → 609A Acceleration → 6060 Switch current mode

Instruction 105	**ZAux_Direct_UserDatum**
Original Format	int32 __stdcall ZAux_Direct_UserDatum(ZMC_HANDLE handle, int iaxis, int imode,float HighSpeed,float LowSpeed,float DatumOffset);
Description	Custom homing in twice.
Input parameters	Parameter name: Description handle: Link mark iaxis: Axis No. imode: Finding origin mode HihSpeed: Homing in high-speed LowSpeed: Homing in low-speed Datumoffset: Offset distance of twice homing
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	Homing mode (it meets position limit to reversely find by default)

Instruction 106	**ZAux_Direct_SetHomeWait**
Original Format	int32 __stdcall ZAux_Direct_SetHomeWait (ZMC_HANDLE handle, int iaxis, int fValue);
Description	Set homing reverse finding waiting time.
Input parameters	Parameter name: Description handle: Link mark iaxis: Output No. fValue: Homing reverse waiting time MS

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Single-axis Homing Motion
Details	For the servo driver in the pulse mode, when returning to the origin, it needs to wait for a certain period of time when finding reversely. The controller default value is 2ms.

Instruction 107	**ZAux_Direct_GetHomeWait**
Original Format	int32 __stdcall ZAux_Direct_GetHomeWait (ZMC_HANDLE handle, int iaxis, int *pfValue);
Description	Read reverse homing waiting time. For the pulse mode servo driver, when returning to the origin, it needs to wait for a certain period of time when finding inversely. The controller default value is 2ms.
Input parameters	Parameter name: handle — Link mark; iaxis — Output No.
Output parameters	Parameter name: piValue — Obtained homing reverse waiting time MS
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Int iValue; ZAux_Direct_GetHomeWait (handle, 0, &iValue); //read the setting value of the homing and homing waiting time of axis 0
Details	For the servo driver in the pulse mode, when returning to the origin, it needs to wait for a certain period of time when finding reversely. The controller default value is 2ms.

Instruction 108	**ZAux_BusCmd_SetDatumOffpos**
Original Format	int32 __stdcall ZAux_BusCmd_SetDatumOffpos (ZMC_HANDLE handle, int iaxis , float fValue);
Description	Set homing offset distance
Input parameters	Parameter name: handle — Link mark; axis — Output No.; fvalue — Set offset distance

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	ZAux_BusCmd_SetDatumOffpos(handle, 0, 100); //set offset 100 after axis 0 homing completed.
Details	It is mainly for bus motors, if it is a pulse motor, some controllers support it, it is determined by the firmware.

Instruction 109	**ZAux_BusCmd_GetDatumOffpos**
Original Format	int32 __stdcall ZAux_BusCmd_GetDatumOffpos (ZMC_HANDLE handle, int iaxis , float *fValue);
Description	Get homing offset distance of homing that is set.
Input parameters	Parameter name: handle — Link mark; iaxis — Output No.
Output parameters	Parameter name: piValue — Obtained homing offset distance
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	float fValue; ZAux_BusCmd_GetDatumOffpos(handle, 0, &fValue); //read homing offset set value of axis 0
Details	It is mainly for bus motors, if it is a pulse motor, some controllers support it, it is determined by the firmware.

Instruction 110	**ZAux_Direct_SetHomeWait**
Original Format	int32 __stdcall ZAux_Direct_SetHomeWait (ZMC_HANDLE handle, int iaxis, int fValue);
Description	Set homing creep speed, which is used to search for the origin. The unit is units/s.
Input parameters	Parameter name: handle — Link mark; iaxis — Output No.; fValue — Creep speed that is set
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error

	codes.
Example	Single-axis Homing Motion
Details	/

Instruction 111	**ZAux_Direct_GetHomeStatus**
Original Format	int32 __stdcall ZAux_Direct_GetHomeStatus (ZMC_HANDLE handle , uint32 iaxis, uint32 *homestatus);
Description	Homing completion state.
Input parameters	Parameter name: handle — Link mark; iaxis — Output No.
Output parameters	Parameter name: homestatue — Read homing state value: 0-abnormal homing. 1-normal homing.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Single-axis Homing Motion
Details	/

Instruction 112	**ZAux_BusCmd_GetHomeStatus**
Original Format	int32 __stdcall ZAux_BusCmd_GetHomeStatus (ZMC_HANDLE handle, uint32 iaxis , uint32 * homestate);
Description	Bus driver homing completion state.
Input parameters	Parameter name: handle — Link mark; iaxis — Output No.
Output parameters	Parameter name: homestatue — Read homing state value: 0-abnormal homing. 1-normal homing.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	/

Instruction 113	**ZAux_Direct_SetDatumIn**
Original Format	int32 __stdcall ZAux_Direct_SetDatumIn(ZMC_HANDLE handle, int iaxis , int iValue)
Description	Set axis mapping origin IN, -1 means cancel.

Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Output No.</td></tr><tr><td>iValue</td><td>IO No., when it is -1, setting is cancelled.</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Single-axis Homing Motion
Details	After setting the origin switch, ZMC controller considers that there is a signal input when the input of the ZMC controller is OFF. To reverse the effect, you can use INVERT_IN to invert the level. (ECI series controllers are opposite).

Instruction 114	**ZAux_Direct_GetDatumIn**
Original Format	int32 __stdcall ZAux_Direct_GetDatumIn(ZMC_HANDLE handle, int iaxis, int *piValue)
Description	Read axis mapping origin IN.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Output No.</td></tr></table>
Output parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>piValue</td><td>Corresponding input No., -1 means no setting.</td></tr></table>
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Single-axis Homing Motion
Details	After setting the origin switch, ZMC controller considers that there is a signal input when the input of the ZMC controller is OFF. To reverse the effect, you can use INVERT_IN to invert the level. (ECI series controllers are opposite).

Instruction 115	**ZAux_Direct_SetClutchRate**
Original Format	int32 __stdcall ZAux_Direct_SetClutchRate(ZMC_HANDLE handle, int iaxis, float fValue)
Description	Speed of connect, ratio/s, default value is 1000000.

<table>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link mark</td></tr>
<tr><td>iaxis</td><td>Output No.</td></tr>
<tr><td>pfValue</td><td>Synchronous connection ratio</td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Robotic arm</td></tr>
<tr><td>Details</td><td>When it is set to 0, the connection is tracked according to the speed/acceleration parameters of the axis, which is more suitable for handwheel movement (when the speed is not high enough, it may cause continuous movement for a period of time to end).
If the setting value cannot be much larger than the CONNECT connection ratio, the actual connection ratio will be reduced
It is generally used to define the change time of the connection rate from 0 to the setting rate</td></tr>
</table>

<table>
<tr><td>Instruction 116</td><td>ZAux_Direct_GetClutchRate</td></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_GetClutchRate(ZMC_HANDLE handle, int iaxis, float *pfValue)</td></tr>
<tr><td>Description</td><td>Read connection speed ratio.</td></tr>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link mark</td></tr>
<tr><td>iaxis</td><td>Output No.</td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>pfValue</td><td>Current synchronous connection ratio</td></tr>
</table>
</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>/</td></tr>
<tr><td>Details</td><td>When it is set to 0, the connection is tracked according to the speed/acceleration parameters of the axis, which is more suitable for handwheel movement (when the speed is not high enough, it may cause continuous movement for a period of time to end).
If the setting value cannot be much larger than the CONNECT connection ratio, the actual connection ratio will be reduced
It is generally used to define the change time of the connection rate from 0 to the setting rate</td></tr>
</table>

<table>
<tr><th>Instruction 117</th><th>ZAux_Direct_EncoderRatio</th></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_EncoderRatio(ZMC_HANDLE handle, int iaxis,int output_count,int input_count)</td></tr>
<tr><td>Description</td><td>Encoder input gear ratio setting, default (1,1), set a negative value to switch direction.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Output No.</td></tr><tr><td>Output_count</td><td>molecular</td></tr><tr><td>Input_count</td><td>denominator</td></tr></table></td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Encoder gear ratio configuration</td></tr>
<tr><td>Details</td><td>/</td></tr>
</table>

<table>
<tr><th>Instruction 118</th><th>ZAux_Direct_StepRatio</th></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_StepRatio(ZMC_HANDLE handle, int iaxis, int output_count,int input_count)</td></tr>
<tr><td>Description</td><td>Set stepper output gear ratio, default (1,1), range is 1-65535.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Output No.</td></tr><tr><td>Output_count</td><td>molecular</td></tr><tr><td>Input_count</td><td>denominator</td></tr></table></td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Pulse output electronic gear ratio configuration</td></tr>
<tr><td>Details</td><td>Setting it to a negative value can modify the motor direction, but is generally not recommended.
The pulse motor can use the INVERT_STEP command, and the bus motor is best modified on the driver.
It is recommended not to modify this parameter casually, and the same effect can be achieved by modifying the pulse equivalent.</td></tr>
</table>

Instruction 119	ZAux_Direct_Connect

Original Format	int32 __stdcall ZAux_Direct_Connect(ZMC_HANDLE handle, float ratio, int link_axis, int move_axis)
Description	Synchronous movement. The electronic gear connects the target position of the current axis to the interpolation vector length of the idring_axis axis through the electronic gear.
Input parameters	Parameter name: handle — Link mark ratio — Ratio, be positive or negative, pay attention the ratio is the number of pulses Link_axis — Axis No. of connecting axis (main axis) Move_axis — Axis No. of motion axis
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Synchronous motion
Details	1. The number of pulses is connected, and the ratio of different axis UNITS should be considered. Use CANCEL to cancel the connection. 2. Suppose the UNIST of the connecting axis 0 is 100, and the UNITS of the connected axis 1 is 10. 3. Use ZAux_Direct_Connect to connect, the ratio is 1, when the axis 0 moves s1=100, the axis 1 moves = s1*UNITS(0)*ratio/UNITS(1), and the movement is 1000 at this time. 4. The ratio can be dynamically changed by calling the instruction repeatedly. 5. Often used in handwheels.

Instruction 120	**ZAux_Direct_Connpath**
Original Format	int32 __stdcall ZAux_Direct_Connpath(ZMC_HANDLE handle, float ratio, int link_axis, int move_axis)
Description	Synchronous movement 2. The electronic gear connects the target position of the current axis to the interpolation vector length of the link_axis axis through the electronic gear.
Input parameters	Parameter name: handle — Link mark ratio — Ratio, be positive or negative, pay attention the ratio is the number of pulses Link_axis — Axis No. of connecting axis (main axis) Move_axis — Axis No. of motion axis

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Synchronous motion 2
Details	1. Connection between interpolation motion and interpolation vector length can be built when the main axis of interpolation motion is connect, and the effect of slave axis connection is same as CONNECT. 2. The number of pulses is connected, and the ratio of different axis UNITS should be considered. Use CANCEL to cancel the connection. 3. The ratio can be dynamically changed by calling the instruction repeatedly.

Instruction 121	**ZAux_Direct_GetLinkax**
Original Format	int32 __stdcall ZAux_Direct_GetLinkax (ZMC_HANDLE handle , int iaxis, int *piValue);
Description	Read current connected and referred axis No.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.
Output parameters	Parameter name: pfValue — Returned connected reference axis No.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Float fValue; ZAux_Direct_GetLinkax (handle ,0, &fValue); //check axis 0 connection axis axis No.
Details	Returns the reference axis number of the current joint motion, or -1 if there is no joint.

Instruction 122	**ZAux_Direct_Move**
Original Format	int32 __stdcall ZAux_Direct_Move(ZMC_HANDLE handle, int imaxaxises, int *piAxislist,float *pfDisancelist)
Description	Multi-axis relative linear interpolation motion.
Input parameters	Parameter name: handle — Link mark; imaxaxises — Motion axes; piAxislist — Axis list array; pfDistancelist — Distance list of motion

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Multi-axis linear interpolation
Details	Interpolation movement distance X=$\sqrt{X_0^2+X_1^2+X_2^2+....+X_n^2}$ Movement time T=X/main axis SPEED Only the main axis speed parameter is valid during interpolation movement, the main axis is the first axis in the axis list array, and the movement refers to the parameters of this axis.

Instruction 123	**ZAux_Direct_MoveSp**
Original Format	int32 __stdcall ZAux_Direct_MoveSp(ZMC_HANDLE handle, int imaxaxises , int *piAxislist, float *pfDisancelist)
Description	SP motion command corresponding to relative linear interpolation.
Input parameters	Parameter name: Description handle: Link mark imaxaxises: Motion axes piAxislist: Axis list array pfDistancelist: Distance list of motion
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	continuous interpolation SP speed motion control
Details	Interpolation movement distance X=$\sqrt{X_0^2+X_1^2+X_2^2+....+X_n^2}$ Movement time T=X/main axis SPEED Only the main axis speed parameter is valid during interpolation movement, the main axis is the first axis in the axis list array, and the movement refers to the parameters of this axis.

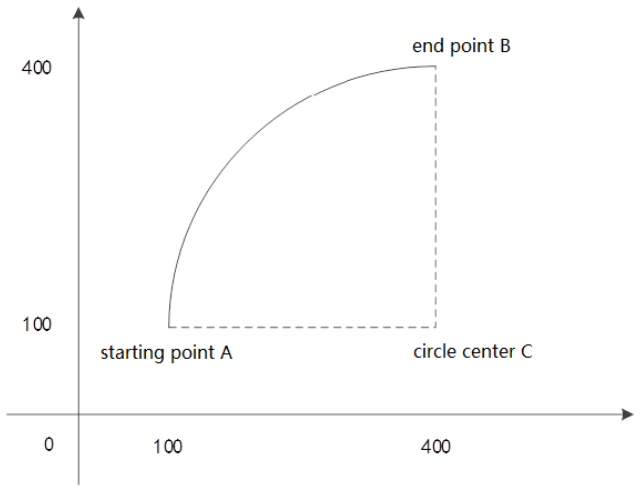
Instruction 124	**ZAux_Direct_MoveAbs**
Original Format	int32 __stdcall ZAux_Direct_MoveAbs(ZMC_HANDLE handle,int imaxaxises, int *piAxislist, float *pfDisancelist)
Description	Absolute linear interpolation.
Input parameters	Parameter name: Description handle: Link mark imaxaxises: Motion axes piAxislist: Axis list array pfDistancelist: Distance list of motion
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error

	codes.
Example	multi-axis linear interpolation
Details	Interpolation movement distance X=$\sqrt{X_0^2+X_1^2+X_2^2+....+X_n^2}$ Movement time T=X/main axis SPEED Only the main axis speed parameter is valid during interpolation movement, the main axis is the first axis in the axis list array, and the movement refers to the parameters of this axis.

Instruction 125	**ZAux_Direct_MoveAbsSp**	
Original Format	int32 __stdcall ZAux_Direct_MoveAbsSp(ZMC_HANDLE handle,int imaxaxises, int *piAxislist, float *pfDisancelist)	
Description	SP motion command corresponding to absolute linear interpolation.	
Input parameters	Parameter name	Description
	handle	Link mark
	imaxaxises	Motion axes
	piAxislist	Axis list array
	pfDistancelist	Distance list of motion
Output parameters	/	
Return value	If it is successful, return value is 0, if not, please refer to error codes.	
Example	/	
Details	Interpolation movement distance X=$\sqrt{X_0^2+X_1^2+X_2^2+....+X_n^2}$ Movement time T=X/main axis SPEED Only the main axis speed parameter is valid during interpolation movement, the main axis is the first axis in the axis list array, and the movement refers to the parameters of this axis.	

Instruction 126	**ZAux_Direct_MoveCirc**	
Original Format	int32 __stdcall ZAux_Direct_MoveCirc(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection)	
Description	The circle center draws the arc. To do circular interpolation of the first axis and the second axis in axis list, it is relative motion method.	
Input parameters	Parameter name	Description
	handle	Link mark
	imaxaxises	Total axes
	piAxislist	Axis list array
	fend1	Motion end coordinate of the first axis
	fend2	Motion end coordinate of the second axis
	fcenter1	The first axis motion circle center,

		corresponding to starting point.
	fcenter2	The second axis motion circle center, corresponding to starting point.
	idirection	0-anticlockwise, 1-clockwise

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Draw the circle by plane center + small-round speed limit
Details	To do circular interpolation of the first axis and the second axis in axis list, it is relative motion method. When the end point distance is 0, it is a full circle. When using, circle center and circular end point that correspond to starting point coordinate should be obtained. Please make sure coordinate positions are correct, otherwise, actual motion trajectory will be wrong.  Suppose starting point A coordinate is (100,100), circle center C coordinate is (400,100), enc point coordinate is (400,400). The coordinate of circler center C that corresponds to starting point A is (300,0), and the coordinate of end point B that corresponds to starting point A is (300,300).

Instruction 127	**ZAux_Direct_MoveCircSp**
Original Format	int32 __stdcall ZAux_Direct_MoveCircSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection)
Description	Draw arc with the circle center (relative) – SP motion command

Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>imaxaxises</td><td>Total axes</td></tr><tr><td>piAxislist</td><td>Axis list array</td></tr><tr><td>fend1</td><td>Motion end coordinate of the first axis</td></tr><tr><td>fend2</td><td>Motion end coordinate of the second axis</td></tr><tr><td>fcenter1</td><td>The first axis motion circle center, corresponding to starting point.</td></tr><tr><td>fcenter2</td><td>The second axis motion circle center, corresponding to starting point.</td></tr><tr><td>idirection</td><td>0-anticlockwise, 1-clockwise</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Reference: example of ZAux_Direct_MoveCirc
Details	Reference: example of ZAux_Direct_MoveCirc

Instruction 128	**ZAux_Direct_MoveCircAbs**
Original Format	int32 __stdcall ZAux_Direct_MoveCircAbs(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection)
Description	The circle center draws the arc. To do circular interpolation of the first axis and the second axis in axis list, it is absolute motion method.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>imaxaxises</td><td>Total axes</td></tr><tr><td>piAxislist</td><td>Axis list array</td></tr><tr><td>fend1</td><td>Motion end coordinate of the first axis</td></tr><tr><td>fend2</td><td>Motion end coordinate of the second axis</td></tr><tr><td>fcenter1</td><td>The first axis motion circle center, which is absolute to starting point.</td></tr><tr><td>fcenter2</td><td>The second axis motion circle center, which is absolute to starting point.</td></tr><tr><td>idirection</td><td>0-anticlockwise, 1-clockwise</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Reference: example of ZAux_Direct_MoveCirc
Details	Reference: example of ZAux_Direct_MoveCirc

<table>
<tr><th>Instruction 129</th><th>ZAux_Direct_MoveCircAbsSp</th></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_MoveCircAbsSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection)</td></tr>
<tr><td>Description</td><td>The circle center draws the arc. Absolute circular motion relates to SP motion command.</td></tr>
<tr><td>Input parameters</td><td><table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link mark</td></tr>
<tr><td>imaxaxises</td><td>Total axes</td></tr>
<tr><td>piAxislist</td><td>Axis list array</td></tr>
<tr><td>fend1</td><td>Motion end coordinate of the first axis</td></tr>
<tr><td>fend2</td><td>Motion end coordinate of the second axis</td></tr>
<tr><td>fcenter1</td><td>The first axis motion circle center, which is absolute to starting point.</td></tr>
<tr><td>fcenter2</td><td>The second axis motion circle center, which is absolute to starting point.</td></tr>
<tr><td>idirection</td><td>0-anticlockwise, 1-clockwise</td></tr>
</table></td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Reference: example of ZAux_Direct_MoveCirc</td></tr>
<tr><td>Details</td><td>Reference: example of ZAux_Direct_MoveCirc</td></tr>
</table>

<table>
<tr><th>Instruction 130</th><th>ZAux_Direct_MoveCirc2</th></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_MoveCirc2(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fmid1, float fmid2, float fend1, float fend2)</td></tr>
<tr><td>Description</td><td>Three points draw the arc. To do circular interpolation of the first axis and the second axis in axis list, it is relative motion method.</td></tr>
<tr><td>Input parameters</td><td><table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link mark</td></tr>
<tr><td>imaxaxises</td><td>Total axes</td></tr>
<tr><td>piAxislist</td><td>Axis list array</td></tr>
<tr><td>fmid1</td><td>Middle point of the first axis, which relates to starting point distance.</td></tr>
<tr><td>fmid2</td><td>Middle point of the second axis, which relates to starting point distance.</td></tr>
<tr><td>fend1</td><td>End point of the first axis, which relates to starting point distance.</td></tr>
<tr><td>fend2</td><td>End point of the second axis, which relates to starting point distance.</td></tr>
</table></td></tr>
</table>

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Plane 3-point Arc Drawing
Details	Note: this instruction cannot be used for full-circle interpolation movement, use MOVECIRC relative to the arc for the full circle, or use two such instructions continuously.

Instruction 131	**ZAux_Direct_MoveCirc2Sp**
Original Format	int32 __stdcall ZAux_Direct_MoveCirc2Sp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fmid1, float fmid2, float fend1, float fend2)
Description	Three points draw the arc. To do circular interpolation of the first axis and the second axis in axis list, it is relative motion method. SP motion command that corresponds to relative motion.
Input parameters	Parameter name: Description handle: Link mark imaxaxises: Total axes piAxislist: Axis list array fmid1: Middle point of the first axis, which relates to starting point distance. fmid2: Middle point of the second axis, which relates to starting point distance. fend1: End point of the first axis, which relates to starting point distance. fend2: End point of the second axis, which relates to starting point distance.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Refer to ZAux_Direct_MoveCirc2.
Details	Note: this instruction cannot be used for full-circle interpolation movement, use MOVECIRC relative to the arc for the full circle, or use two such instructions continuously.

Instruction 132	**ZAux_Direct_MoveCirc2Abs**
Original Format	int32 __stdcall ZAux_Direct_MoveCirc2Abs(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fmid1, float fmid2, float fend1, float fend2)
Description	Three points draw the arc. To do circular interpolation of the first axis and the second axis in axis list, it is relative motion method. SP motion command that corresponds to absolute motion.

Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>imaxaxises</td><td>Total axes</td></tr><tr><td>piAxislist</td><td>Axis list array</td></tr><tr><td>fmid1</td><td>Middle point of the first axis, which is absolute to starting point distance.</td></tr><tr><td>fmid2</td><td>Middle point of the second axis, which is absolute to starting point distance.</td></tr><tr><td>fend1</td><td>End point of the first axis, which is absolute to starting point distance.</td></tr><tr><td>fend2</td><td>End point of the second axis, which is absolute to starting point distance.</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Refer to ZAux_Direct_MoveCirc2.
Details	Note: this instruction cannot be used for full-circle interpolation movement, use MOVECIRC relative to the arc for the full circle, or use two such instructions continuously.

Instruction 133	**ZAux_Direct_MoveCirc2AbsSp**
Original Format	int32 __stdcall ZAux_Direct_MoveCirc2AbsSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fmid1, float fmid2, float fend1, float fend2)
Description	To do circular interpolation of the first axis and the second axis in axis list, it is absolute motion method. Corresponding SP motion command.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>imaxaxises</td><td>Total axes</td></tr><tr><td>piAxislist</td><td>Axis list array</td></tr><tr><td>fmid1</td><td>Middle point of the first axis, which is absolute to starting point distance.</td></tr><tr><td>fmid2</td><td>Middle point of the second axis, which is absolute to starting point distance.</td></tr><tr><td>fend1</td><td>End point of the first axis, which is absolute to starting point distance.</td></tr><tr><td>fend2</td><td>End point of the second axis, which is absolute to starting point distance.</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.

Example	Refer to ZAux_Direct_MoveCirc2.
Details	Note: this instruction cannot be used for full-circle interpolation movement, use MOVECIRC relative to the arc for the full circle, or use two such instructions continuously.

Instruction 134	**ZAux_Direct_MHelical**
Original Format	int32 __stdcall ZAux_Direct_MHelical(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fDistance3, int imode)
Description	Center circle helical. To do circular interpolation of the first axis and the second axis in axis list, the third axis does helical interpolation, which are relative to starting point.
Input parameters	

Parameter name	Description
handle	Link mark
imaxaxises	Total axes
piAxislist	Axis list array
fend1	Motion coordinate of the first axis
fend2	Motion coordinate of the second axis
fcentre1	Circle center of the first axis motion which is relative to starting point distance.
fcentre2	Circle center of the second axis motion which is relative to starting point distance.
idirection	0-anticlockwise, 1-clockwise
fdistance3	Motion distance of the third axis
imode	The third axis' speed calculation: 0(default): the third axis participates in speed calculation 1: the third axis doesn't participate in speed calculation

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Helical curve
Details	It can complete a full spiral circle, and it can be viewed as a complete circle from the Z direction. Please make sure the coordinate position is correct, otherwise the actual motion trajectory will be wrong.

Instruction 135	**ZAux_Direct_MHelicalSp**
Original Format	int32 __stdcall ZAux_Direct_MHelicalSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1,

<table>
<tr><td></td><td>float fcenter2, int idirection, float fDistance3, int imode)</td></tr>
<tr><td>Description</td><td>Center circle helical motion corresponds to SP motion command.</td></tr>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link mark</td></tr>
<tr><td>imaxaxises</td><td>Total axes</td></tr>
<tr><td>piAxislist</td><td>Axis list array</td></tr>
<tr><td>fend1</td><td>Motion coordinate of the first axis</td></tr>
<tr><td>fend2</td><td>Motion coordinate of the second axis</td></tr>
<tr><td>fcentre1</td><td>Circle center of the first axis motion which is relative to starting point distance.</td></tr>
<tr><td>fcentre2</td><td>Circle center of the second axis motion which is relative to starting point distance.</td></tr>
<tr><td>idirection</td><td>0-anticlockwise, 1-clockwise</td></tr>
<tr><td>fdistance3</td><td>Motion distance of the third axis</td></tr>
<tr><td>imode</td><td>The third axis' speed calculation:
0(default): the third axis participates in speed calculation
1: the third axis doesn't participate in speed calculation</td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>refer to ZAux_Direct_MHelical.</td></tr>
<tr><td>Details</td><td>It can complete a full spiral circle, and it can be viewed as a complete circle from the Z direction.
Please make sure the coordinate position is correct, otherwise the actual motion trajectory will be wrong.</td></tr>
</table>

<table>
<tr><td>Instruction 136</td><td>ZAux_Direct_MHelicalAbs</td></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_MHelicalAbs(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fDistance3, int imode)</td></tr>
<tr><td>Description</td><td>Center circle helical. To do circular interpolation of the first axis and the second axis in axis list, the third axis does helical interpolation. It is absolute motion method.</td></tr>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link mark</td></tr>
<tr><td>imaxaxises</td><td>Total axes</td></tr>
<tr><td>piAxislist</td><td>Axis list array</td></tr>
<tr><td>fend1</td><td>Motion coordinate of the first axis</td></tr>
<tr><td>fend2</td><td>Motion coordinate of the second axis</td></tr>
</table>
</td></tr>
</table>

	fcentre1	Circle center of the first axis motion, absolute coordinate.
	fcentre2	Circle center of the second axis motion absolute coordinate.
	idirection	0-anticlockwise, 1-clockwise
	fdistance3	Motion distance of the third axis
	imode	The third axis' speed calculation: 0(default): the third axis participates in speed calculation 1: the third axis doesn't participate in speed calculation
Output parameters	/	
Return value	If it is successful, return value is 0, if not, please refer to error codes.	
Example	refer to ZAux_Direct_MHelical.	
Details	It can complete a full spiral circle, and it can be viewed as a complete circle from the Z direction. Please make sure the coordinate position is correct, otherwise the actual motion trajectory will be wrong.	

Instruction 137	**ZAux_Direct_MHelicalAbsSp**	
Original Format	int32 __stdcall ZAux_Direct_MHelicalAbsSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fDistance3, int imode)	
Description	Center circle helical. To do circular interpolation of the first axis and the second axis in axis list, the third axis does helical interpolation. It is absolute motion method. Correspond to SP motion command.	
Input parameters	Parameter name	Description
	handle	Link mark
	imaxaxises	Total axes
	piAxislist	Axis list array
	fend1	Motion coordinate of the first axis
	fend2	Motion coordinate of the second axis
	fcentre1	Circle center of the first axis motion, absolute coordinate.
	fcentre2	Circle center of the second axis motion absolute coordinate.
	idirection	0-anticlockwise, 1-clockwise
	fdistance3	Motion distance of the third axis
	imode	The third axis' speed calculation: 0(default): the third axis participates in speed calculation

	1: the third axis doesn't participate in speed calculation
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	refer to ZAux_Direct_MHelical.
Details	It can complete a full spiral circle, and it can be viewed as a complete circle from the Z direction. Please make sure the coordinate position is correct, otherwise the actual motion trajectory will be wrong.

Instruction 138	**ZAux_Direct_MHelical2Abs**
Original Format	int32 __stdcall ZAux_Direct_MHelical2Abs(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fmid1, float fmid2, float fend1, float fend2, float fDistance3, int imode)
Description	Three points helical. To do circular interpolation of the first axis and the second axis in axis list, the third axis does helical interpolation. It is absolute motion method.
Input parameters	Parameter name: Description handle: Link mark imaxaxises: Total axes piAxislist: Axis list array fmid1: Middle point coordinate of the first axis, absolute coordinate. fmid2: Middle point coordinate of the second axis, absolute coordinate. fend1: End point coordinate of the first axis, absolute coordinate. fend2: End point coordinate of the second axis, absolute coordinate. fdistance3: Motion distance of the third axis imode: The third axis' speed calculation: 0(default): the third axis participates in speed calculation 1: the third axis doesn't participate in speed calculation
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/

Details	It can't turn one helical, please use MHELICAL or MHELICALABS if you need to make a turn.

Instruction 139	**ZAux_Direct_MHelical2AbsSp**
Original Format	int32 __stdcall ZAux_Direct_MHelical2AbsSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fmid1, float fmid2, float fend1, float fend2, float fDistance3, int imode)
Description	Function of the function. To do circular interpolation of the first axis and the second axis in axis list, the third axis does helical interpolation. It is absolute motion method. Correspond to SP motion command.
Input parameters	

Parameter name	Description
handle	Link mark
imaxaxises	Total axes
piAxislist	Axis list array
fmid1	Middle point coordinate of the first axis, absolute coordinate.
fmid2	Middle point coordinate of the second axis, absolute coordinate.
fend1	End point coordinate of the first axis, absolute coordinate.
fend2	End point coordinate of the second axis, absolute coordinate.
fdistance3	Motion distance of the third axis
imode	The third axis' speed calculation: 0(default): the third axis participates in speed calculation 1: the third axis doesn't participate in speed calculation

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	It can't turn one helical, please use MHELICAL or MHELICALABS if you need to make a turn.

Instruction 140	**ZAux_Direct_MHelical2**
Original Format	int32 __stdcall ZAux_Direct_MHelical2(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fmid1, float fmid2, float fend1, float fend2, float fDistance3, int imode)
Description	To do circular interpolation of the first axis and the second axis in axis list, the third axis does helical interpolation. It is relative motion method.

<table>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link mark</td></tr>
<tr><td>imaxaxises</td><td>Total axes</td></tr>
<tr><td>piAxislist</td><td>Axis list array</td></tr>
<tr><td>fmid1</td><td>Middle point coordinate of the first axis, which relates to starting point.</td></tr>
<tr><td>fmid2</td><td>Middle point coordinate of the second axis, which relates to starting point.</td></tr>
<tr><td>fend1</td><td>End point coordinate of the first axis, which relates to starting point.</td></tr>
<tr><td>fend2</td><td>End point coordinate of the second axis, which relates to starting point.</td></tr>
<tr><td>fdistance3</td><td>Motion distance of the third axis</td></tr>
<tr><td>imode</td><td>The third axis' speed calculation:
0(default): the third axis participates in speed calculation
1: the third axis doesn't participate in speed calculation</td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>/</td></tr>
<tr><td>Details</td><td>It can't turn one helical, please use MHELICAL or MHELICALABS if you need to make a turn.</td></tr>
</table>

<table>
<tr><td>Instruction 141</td><td>ZAux_Direct_MHelical2Sp</td></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_MHelical2Sp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fmid1, float fmid2, float fend1, float fend2, float fDistance3, int imode)</td></tr>
<tr><td>Description</td><td>To do circular interpolation of the first axis and the second axis in axis list, the third axis does helical interpolation. It is relative motion method. Correspond to SP motion command.</td></tr>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link mark</td></tr>
<tr><td>imaxaxises</td><td>Total axes</td></tr>
<tr><td>piAxislist</td><td>Axis list array</td></tr>
<tr><td>fmid1</td><td>Middle point coordinate of the first axis, which relates to starting point.</td></tr>
<tr><td>fmid2</td><td>Middle point coordinate of the second axis, which relates to starting point.</td></tr>
<tr><td>fend1</td><td>End point coordinate of the first axis, which relates to starting point.</td></tr>
<tr><td>fend2</td><td>End point coordinate of the second axis,</td></tr>
</table>
</td></tr>
</table>

		which relates to starting point.
	fdistance3	Motion distance of the third axis
	imode	The third axis' speed calculation: 0(default): the third axis participates in speed calculation 1: the third axis doesn't participate in speed calculation
Output parameters	/	
Return value	If it is successful, return value is 0, if not, please refer to error codes.	
Example	/	
Details	It can't turn one helical, please use MHELICAL or MHELICALABS if you need to make a turn.	

Instruction 142	**ZAux_Direct_MEclipse**	
Original Format	int32 __stdcall ZAux_Direct_MEclipse(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fADis, float fBDis)	
Description	Eclipse interpolation, draw the arc with the center. To do eclipse interpolation of the first axis and the second axis in axis list, the third axis does helical interpolation. It is relative motion method.	
Input parameters	Parameter name	Description
	handle	Link mark
	imaxaxises	Total axes
	piAxislist	Axis list array
	fend1	Motion coordinate of the first axis.
	fend2	Motion coordinate of the second axis.
	fcentre1	Circle center of the first axis, which relates to starting point.
	fcertre2	Circle center of the second axis, which relates to starting point.
	idirection	0-anticlockwise, 1-clockwise
	fADis	The radius of the ellipse on the first axis can be semi-major or semi-minor.
	fBDis	The radius of the ellipse on the second axis can be semi-major or semi-minor. When AB is equal, it is automatically an arc or a spiral.
Output parameters	/	
Return value	If it is successful, return value is 0, if not, please refer to error codes.	
Example	/	

Details	Three-axis synchronous helical can be selected. Full ellipse can be drawn. It can only draw an ellipse whose major axis (short axis) is parallel or perpendicular to X

Instruction 143	**ZAux_Direct_MEclipseSp**
Original Format	int32 __stdcall ZAux_Direct_MEclipseSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fADis, float fBDis)
Description	Eclipse interpolation motion relates to SP motion command.
Input parameters	

Parameter name	Description
handle	Link mark
imaxaxises	Total axes
piAxislist	Axis list array
fend1	Motion coordinate of the first axis.
fend2	Motion coordinate of the second axis.
fcentre1	Circle center of the first axis, which relates to starting point.
fcertre2	Circle center of the second axis, which relates to starting point.
idirection	0-anticlockwise, 1-clockwise
fADis	The radius of the ellipse on the first axis can be semi-major or semi-minor.
fBDis	The radius of the ellipse on the second axis can be semi-major or semi-minor. When AB is equal, it is automatically an arc or a spiral.

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	Three-axis synchronous helical can be selected. Full ellipse can be drawn. It can only draw an ellipse whose major axis (short axis) is parallel or perpendicular to X

Instruction 144	**ZAux_Direct_MEclipseAbs**
Original Format	int32 __stdcall ZAux_Direct_MEclipseAbs(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fADis, float fBDis)
Description	To do eclipse interpolation of the first axis and the second axis in axis list. It is absolute motion method.

Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>imaxaxises</td><td>Total axes</td></tr><tr><td>piAxislist</td><td>Axis list array</td></tr><tr><td>fend1</td><td>Motion coordinate of the first axis.</td></tr><tr><td>fend2</td><td>Motion coordinate of the second axis.</td></tr><tr><td>fcentre1</td><td>Circle center of the first axis, which is absolute to starting point.</td></tr><tr><td>fcertre2</td><td>Circle center of the second axis, which is absolute to starting point.</td></tr><tr><td>idirection</td><td>0-anticlockwise, 1-clockwise</td></tr><tr><td>fADis</td><td>The radius of the ellipse on the first axis can be semi-major or semi-minor.</td></tr><tr><td>fBDis</td><td>The radius of the ellipse on the second axis can be semi-major or semi-minor. When AB is equal, it is automatically an arc or a spiral.</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	Three-axis synchronous helical can be selected. Full ellipse can be drawn. It can only draw an ellipse whose major axis (short axis) is parallel or perpendicular to X

Instruction 145	**ZAux_Direct_MEclipseAbsSp**
Original Format	int32 __stdcall ZAux_Direct_MEclipseAbsSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fADis, float fBDis)
Description	To do eclipse interpolation of the first axis and the second axis in axis list. It is absolute motion method. Correspond to SP motion command.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>imaxaxises</td><td>Total axes</td></tr><tr><td>piAxislist</td><td>Axis list array</td></tr><tr><td>fend1</td><td>Motion coordinate of the first axis.</td></tr><tr><td>fend2</td><td>Motion coordinate of the second axis.</td></tr><tr><td>fcentre1</td><td>Circle center of the first axis, which is absolute to starting point.</td></tr><tr><td>fcertre2</td><td>Circle center of the second axis, which is absolute to starting point.</td></tr></table>

	idirection	0-anticlockwise, 1-clockwise
	fADis	The radius of the ellipse on the first axis can be semi-major or semi-minor.
	fBDis	The radius of the ellipse on the second axis can be semi-major or semi-minor. When AB is equal, it is automatically an arc or a spiral.
Output parameters	/	
Return value	If it is successful, return value is 0, if not, please refer to error codes.	
Example	/	
Details	Three-axis synchronous helical can be selected. Full ellipse can be drawn. It can only draw an ellipse whose major axis (short axis) is parallel or perpendicular to X	

Instruction 146	**ZAux_Direct_MEclipseHelical**	
Original Format	int32 __stdcall ZAux_Direct_MEclipseHelical(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fADis, float fBDis, float fDistance3)	
Description	Interpolation, ellipse + helical	
Input parameters	Parameter name	Description
	handle	Link mark
	imaxaxises	Total axes
	piAxislist	Axis list array
	fend1	Motion coordinate of the first axis.
	fend2	Motion coordinate of the second axis.
	fcentre1	Circle center of the first axis, which relates to starting point.
	fcertre2	Circle center of the second axis, which relates to starting point.
	idirection	0-anticlockwise, 1-clockwise
	fADis	The radius of the ellipse on the first axis can be semi-major or semi-minor.
	fBDis	The radius of the ellipse on the second axis can be semi-major or semi-minor. When AB is equal, it is automatically an arc or a spiral.
	fDistance3	Motion coordinate of the third axis
Output parameters	/	
Return value	If it is successful, return value is 0, if not, please refer to error	

	codes.
Example	/
Details	Please make sure the coordinate position is correct, otherwise the actual motion trajectory will be wrong.

Instruction 147	**ZAux_Direct_MEclipseHelicalSp**
Original Format	int32 __stdcall ZAux_Direct_MEclipseHelicalSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fADis, float fBDis, float fDistance3)
Description	Interpolation, ellipse + helical. Corresponding SP motion command.
Input parameters	Parameter name / Description handle: Link mark imaxaxises: Total axes piAxislist: Axis list array fend1: Motion coordinate of the first axis. fend2: Motion coordinate of the second axis. fcentre1: Circle center of the first axis, which relates to starting point. fcertre2: Circle center of the second axis, which relates to starting point. idirection: 0-anticlockwise, 1-clockwise fADis: The radius of the ellipse on the first axis can be semi-major or semi-minor. fBDis: The radius of the ellipse on the second axis can be semi-major or semi-minor. When AB is equal, it is automatically an arc or a spiral. fDistance3: Motion coordinate of the third axis
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	Please make sure the coordinate position is correct, otherwise the actual motion trajectory will be wrong.

Instruction 148	**ZAux_Direct_MEclipseHelicalAbs**
Original Format	int32 __stdcall ZAux_Direct_MEclipseHelicalAbs(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fcenter1, float fcenter2, int idirection, float fADis, float fBDis,float fDistance3)
Description	To do eclipse interpolation of the first axis and the second axis in axis list. It is absolute motion method. The third axis does helical

	synchronously (absolute).
Input parameters	Parameter name: Description handle: Link mark imaxaxises: Total axes piAxislist: Axis list array fend1: Motion coordinate of the first axis. Absolute coordinate. fend2: Motion coordinate of the second axis. Absolute coordinate. fcentre1: Circle center of the first axis, which is absolute to starting point. fcertre2: Circle center of the second axis, which is absolute to starting point. idirection: 0-anticlockwise, 1-clockwise fADis: The radius of the ellipse on the first axis can be semi-major or semi-minor. fBDis: The radius of the ellipse on the second axis can be semi-major or semi-minor. When AB is equal, it is automatically an arc or a spiral. fDistance3: Motion coordinate of the third axis
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	Please make sure the coordinate position is correct, otherwise the actual motion trajectory will be wrong.

Instruction 149	**ZAux_Direct_MEclipseHelicalAbsSp**
Original Format	int32 __stdcall ZAux_Direct_MEclipseHelicalAbsSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1,float fend2, float fcenter1, float fcenter2, int idirection, float fADis, float fBDis,float fDistance3)
Description	To do eclipse interpolation of the first axis and the second axis in axis list. It is absolute motion method. The third axis does helical synchronously (absolute). Corresponding SP motion command.
Input parameters	Parameter name: Description handle: Link mark imaxaxises: Total axes piAxislist: Axis list array fend1: Motion coordinate of the first axis. Absolute coordinate. fend2: Motion coordinate of the second axis.

		Absolute coordinate.
	fcentre1	Circle center of the first axis, which is absolute to starting point.
	fcertre2	Circle center of the second axis, which is absolute to starting point.
	idirection	0-anticlockwise, 1-clockwise
	fADis	The radius of the ellipse on the first axis can be semi-major or semi-minor.
	fBDis	The radius of the ellipse on the second axis can be semi-major or semi-minor. When AB is equal, it is automatically an arc or a spiral.
	fDistance3	Motion coordinate of the third axis
Output parameters	/	
Return value	If it is successful, return value is 0, if not, please refer to error codes.	
Example	/	
Details	Please make sure the coordinate position is correct, otherwise the actual motion trajectory will be wrong.	

Instruction 150	**ZAux_Direct_MSpherical**	
Original Format	int32 __stdcall ZAux_Direct_MSpherical(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fend3, float fcenter1, float fce nter2, float fcenter3, int imode, float fcenter4, float fcenter5)	
Description	Space circular interpolation motion, which is relative motion method.	
Input parameters	Parameter name	Description
	handle	Link mark
	imaxaxises	Total axes
	piAxislist	Axis list array
	fend1	Motion distance parameter 1 of the first axis.
	fend2	Motion distance parameter 1 of the second axis.
	fend3	Motion distance parameter 1 of the third axis.
	fcentre1	Motion distance parameter 2 of the first axis.
	fcertre2	Motion distance parameter 2 of the second axis.
	fcertre3	Motion distance parameter 2 of the third axis.

	imode	Specify the meaning of the previous parameters 0: The current point, the middle point, and the end point define the arc: Distance parameter 1: the end point distance. Distance parameter 2: the distance between the middle points. 1: The current point, the center of the circle, and the end point determine the arc, and take the smallest arc. Distance parameter 1: the end point distance. Distance parameter 2: the distance from the center of the circle. 2: The current point, the middle point, and the end point determine the circle: Distance parameter 1: the end point distance. Distance parameter 2: the distance between the middle points. 3: The current point, the center of the circle, and the end point determine the circle, first walk the smallest arc, and then continue to walk the complete circle: Distance parameter 1: the end point distance Distance parameter 2: the distance from the center of the circle
	Fcertre4	Helical distance of the fourth axis.
	Fcertre5	Helical distance of the fifth axis.
Output parameters	/	
Return value	If it is successful, return value is 0, if not, please refer to error codes.	
Example	space arc	
Details	Please make sure the coordinate position is correct, otherwise the actual motion trajectory will be wrong.	

Instruction 151	**ZAux_Direct_MSphericalSp**
Original Format	int32 __stdcall ZAux_Direct_MSphericalSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float fend1, float fend2, float fend3, float fcenter1, float fcenter2, float fcenter3, int imode, float fcenter4, float fcenter5)

<table>
<tr><td>Description</td><td>Space circular interpolation motion, which is relative motion method. Corresponding SP motion command.</td></tr>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link mark</td></tr>
<tr><td>imaxaxises</td><td>Total axes</td></tr>
<tr><td>piAxislist</td><td>Axis list array</td></tr>
<tr><td>fend1</td><td>Motion distance parameter 1 of the first axis.</td></tr>
<tr><td>fend2</td><td>Motion distance parameter 1 of the second axis.</td></tr>
<tr><td>fend3</td><td>Motion distance parameter 1 of the third axis.</td></tr>
<tr><td>fcentre1</td><td>Motion distance parameter 2 of the first axis.</td></tr>
<tr><td>fcertre2</td><td>Motion distance parameter 2 of the second axis.</td></tr>
<tr><td>fcertre3</td><td>Motion distance parameter 2 of the third axis.</td></tr>
<tr><td>imode</td><td>Specify the meaning of the previous parameters
0: The current point, the middle point, and the end point define the arc:
Distance parameter 1: the end point distance.
Distance parameter 2: the distance between the middle points.
1: The current point, the center of the circle, and the end point determine the arc, and take the smallest arc.
Distance parameter 1: the end point distance.
Distance parameter 2: the distance from the center of the circle.
2: The current point, the middle point, and the end point determine the circle:
Distance parameter 1: the end point distance.
Distance parameter 2: the distance between the middle points.
3: The current point, the center of the circle, and the end point determine the circle, first walk the smallest arc, and then continue to walk the complete circle:
Distance parameter 1: the end point</td></tr>
</table>
</td></tr>
</table>

		distance Distance parameter 2: the distance from the center of the circle
	Fcertre4	Helical distance of the fourth axis.
	Fcertre5	Helical distance of the fifth axis.
Output parameters	/	
Return value	If it is successful, return value is 0, if not, please refer to error codes.	
Example	Refer to ZAux_Direct_MsPherical.	
Details	Please make sure the coordinate position is correct, otherwise the actual motion trajectory will be wrong.	

Instruction 152	**ZAux_Direct_MoveSpiral**	
Original Format	int32 __stdcall ZAux_Direct_MoveSpiral(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float centre1, float centre2, float circles, float pitch,float distance3, float distance4)	
Description	Involute circular interpolation movement, relative movement mode, starts from angle 0 when the initial radius is 0 and spreads directly.	
Input parameters	Parameter name	Description
	handle	Link mark
	imaxaxises	Total axes
	piAxislist	Axis list array
	centre1	Relative distance of first axis circle center.
	centre2	Relative distance of second axis circle center.
	circles	Circles to be rotated, can be decimal circles, negative value means clockwise.
	pitch	Diffusion distance per circle, can be negative.
	distance3	The function of the 3rd axis spiral, specify the relative distance of the 3rd axis, this axis does not participate in the speed calculation.
	distance4	The function of the 4th axis spiral,

		specify the relative distance of the 4th axis, this axis does not participate in the speed calculation.
Output parameters	/	
Return value	If it is successful, return value is 0, if not, please refer to error codes.	
Example	/	
Details	The distance between the current point and the center of the circle determines the starting radius. When the starting radius is 0, the angle cannot be determined, and the angle starts directly from 0.	

Instruction 153	**ZAux_Direct_MoveSpiralSp**	
Original Format	int32 __stdcall ZAux_Direct_MoveSpiralSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float centre1, float centre2, float circles, float pitch,float distance3, float distance4)	
Description	Involute circular interpolation movement, relative movement mode, starts from angle 0 when the initial radius is 0 and spreads directly.	
Input parameters	Parameter name	Description
	handle	Link mark
	imaxaxises	Total axes
	piAxislist	Axis list array
	centre1	Relative distance of first axis circle center.
	centre2	Relative distance of second axis circle center.
	circles	Circles to be rotated, can be decimal circles, negative value means clockwise.
	pitch	Diffusion distance per circle, can be negative.
	distance3	The function of the 3rd axis spiral, specify the relative distance of the 3rd axis, this axis does not participate in the speed calculation.

	distance4 — The function of the 4th axis spiral, specify the relative distance of the 4th axis, this axis does not participate in the speed calculation.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	The distance between the current point and the center of the circle determines the starting radius. When the starting radius is 0, the angle cannot be determined, and the angle starts directly from 0.

Instruction 154	**ZAux_Direct_MoveSmooth**
Original Format	int32 __stdcall ZAux_Direct_MoveSmooth(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float end1, float end2, float end3, float next1, float next2, float next3, float radius)
Description	Spatial straight line round interpolation movement, according to the absolute coordinates of the next straight line movement, automatically inserts a circular arc at the corner.
Input parameters	

Parameter name	Description
handle	Link mark
imaxaxises	Total axes
piAxislist	Axis list array
end1	Absolute coordinate of 1st axis
end2	Absolute coordinate of second axis
end3	Absolute coordinate of 3rd axis
Next1	Next linear motion absolute coordinate of 1st axis.
Next2	Next linear motion absolute coordinate of second axis.
Next3	Next linear motion absolute coordinate of 3rd axis.

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	This command is an early development command and has a limit on the number of axes. It is recommended to use the CORNER_MODE command, which has more functions. According to the absolute coordinates of the next linear motion, the arc is automatically inserted at the corner. After adding the arc, the end point of the motion will be inconsistent with the end point of the straight line. And when the corner is too large, the arc will not be inserted. Also when the distance is not enough, the radius will be automatically reduced.

Instruction 155	**ZAux_Direct_MoveSmoothSp**
Original Format	int32 __stdcall ZAux_Direct_MoveSmoothSp(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, float end1, float end2, float end3, float next1, float next2, float next3, float radius)
Description	Spatial straight line round interpolation movement, according to the absolute coordinates of the next straight line movement, automatically inserts a circular arc at the corner.
Input parameters	Parameter name: Description handle: Link mark imaxaxises: Total axes piAxislist: Axis list array end1: Absolute coordinate of 1st axis end2: Absolute coordinate of second axis end3: Absolute coordinate of 3rd axis Next1: Next linear motion absolute coordinate of 1st axis. Next2: Next linear motion absolute coordinate of second axis. Next3: Next linear motion absolute coordinate of 3rd axis. radius: Insert the radius of circular, when it is overlarge, automatically reduce.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	This command is an early development command and has a limit on the number of axes. It is recommended to use the

	CORNER_MODE command, which has more functions. According to the absolute coordinates of the next linear motion, the arc is automatically inserted at the corner. After adding the arc, the end point of the motion will be inconsistent with the end point of the straight line. And when the corner is too large, the arc will not be inserted. Also when the distance is not enough, the radius will be automatically reduced.

Instruction 156	**ZAux_Direct_McircTurnabs**
Original Format	int32 __stdcall ZAux_Direct_McircTurnabs (ZMC_HANDLE handle , int tablenum, float refpos1, float refpos2, int mode, float end1, float end2, int maxaxises, int *piAxislist, float * pfDistancelist);
Description	Rotary table circular arc + helical interpolation movement. Valid in controller versions after 20130901.
Input parameters	Parameter name: Description handle: Link mark tablenum: TABLE No. that stores rotary parameters. refpos1: Reference point of first axis, absolute position. refpos2: Reference point of second axis, absolute position. end1: End point of first axis, absolute position. end2: End point of second axis, absolute position. Maxaxises: The number of axes that join in the motion piAxislist: Axis list pfDistancelist: Rotary axis distance list
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	The rotation function means that the working platform rotates on a plane parallel to XY, and the positive direction of rotation must be consistent with the positive direction of XY (right-hand rule). The rotation parameters are stored in the TABLE table in sequence, which stores the R-axis number, the pulse number of R-axis one circle, the X-axis number, the Y-axis number, the X circle center, and the Y circle center.

Instruction 157	**ZAux_Direct_MoveTurnabs**
Original Format	int32 __stdcall ZAux_Direct_MoveTurnabs (ZMC_HANDLE handle , int tablenum, int imaxaxises, int *piAxislist, float * pfDistancelist);

Description	Rotary table linear interpolation movement. Valid in controller versions after 20130901.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>tablenum</td><td>TABLE No. that stores rotary parameters.</td></tr><tr><td>Maxaxises</td><td>The number of axes that join in the motion</td></tr><tr><td>piAxislist</td><td>Axis No. list</td></tr><tr><td>pfDistancelist</td><td>Distance list</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	The rotation function means that the working platform rotates on a plane parallel to XY, and the positive direction of rotation must be consistent with the positive direction of XY (right-hand rule). The rotation parameters are stored in the TABLE table in sequence, which stores the R-axis number, the pulse number of R-axis one circle, the X-axis number, the Y-axis number, the X circle center, and the Y circle center.

Instruction 158	**ZAux_Direct_MultiMove**
Original Format	int32 __stdcall ZAux_Direct_MultiMove(ZMC_HANDLE handle,int iMoveLen, int imaxaxises, int *piAxislist, float *pfDisancelist)
Description	Several relative multi-axis linear interpolation.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iMoveLen</td><td>Filled motion length</td></tr><tr><td>Maxaxises</td><td>The number of axes that join in the motion</td></tr><tr><td>piAxislist</td><td>Axis No. list array</td></tr><tr><td>pfDistancelist</td><td>Distance list array</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	/

Instruction 159	**ZAux_Direct_MultiMoveAbs**

Original Format	int32 __stdcall ZAux_Direct_MultiMoveAbs(ZMC_HANDLE handle,int iMoveLen, int imaxaxises, int *piAxislist, float *pfDisancelist)
Description	Several absolute multi-axis linear interpolation.
Input parameters	Parameter name: Description handle: Link mark iMoveLen: Filled motion length Maxaxises: The number of axes that join in the motion piAxislist: Axis No. list array pfDistancelist: Distance list array
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	/

Instruction 160	**ZAux_Direct_MultiLineN**
Original Format	int32 __stdcall ZAux_Direct_MultiLineN(ZMC_HANDLE handle,int imode,int iMoveLen, int imaxaxises, int *piAxislist, float *pfDisancelist,int *iReBuffLen)
Description	Several relative multi-axis linear interpolation.
Input parameters	Parameter name: Description handle: Link mark imode: bit0- bifabs absolute mode bit1- bifsp custom speed mode bit2- bifresume small line segment servo pause bit3- bifmovescan galvanometer mode iMoveLen: Filled motion length Maxaxises: The number of axes that join in the motion piAxislist: Axis No. list array pfDistancelist: Distance list array
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	/

Instruction 161	ZAux_Direct_SetFhspeed
Original Format	int32 __stdcall ZAux_Direct_SetFhspeed (ZMC_HANDLE handle , int iaxis, float fValue);
Description	Set axis holding speed.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.; fValue — Speed value that is set
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Axis Holding Input
Details	Only when the corresponding input is in the hold state can it keep moving at this speed.

Instruction 162	ZAux_Direct_GetFhspeed
Original Format	int32 __stdcall ZAux_Direct_GetFhspeed (ZMC_HANDLE handle , int iaxis, float *pfValue);
Description	Read axis holding speed.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.; fValue — Speed value that is set
Output parameters	Parameter name: fValue — Speed value that is read.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Axis Holding Input
Details	Only when the corresponding input is in the hold state can it keep moving at this speed.

Instruction 163	ZAux_Direct_SetEndMoveSpeed
Original Format	int32 __stdcall ZAux_Direct_SetEndMoveSpeed(ZMC_HANDLE handle, int iaxis, float fValue);
Description	Set end speed of SP motion that customs the speed.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.; fValue — Speed value that is set

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	continuous interpolation SP motion
Details	It will take effect only when the motion command with SP is used. Please set a larger value when not in use. The controller default value is 1000.

Instruction 164	**ZAux_Direct_GetEndMoveSpeed**
Original Format	int32 __stdcall ZAux_Direct_GetEndMoveSpeed(ZMC_HANDLE handle, int iaxis, float *fValue);
Description	Read end speed of SP motion that customs the speed.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.
Output parameters	Parameter name: fValue — Speed value that is read.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	continuous interpolation SP motion
Details	It will take effect only when the motion command with SP is used. Please set a larger value when not in use. The controller default value is 1000.

Instruction 165	**ZAux_Direct_GetForceSpeed**
Original Format	int32 __stdcall ZAux_Direct_GetForceSpeed (ZMC_HANDLE handle , int iaxis, float *pfValue);
Description	Read SP running speed.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.
Output parameters	Parameter name: fValue — SP speed value that is read.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Axis Holding Input
Details	This parameter is carried into the motion buffer. When FORCE_SPEED is greater than SPEED, SPEED will also take effect to limit the maximum speed (SPEED does not work in

	versions after 140716). If FORCE_SPEED has been reduced to the corresponding speed when entering this section, please set STARTMOVE_SPEED.

Instruction 166	ZAux_Direct_SetForceSpeed
Original Format	int32 __stdcall ZAux_Direct_SetForceSpeed (ZMC_HANDLE handle , int iaxis, float fValue);
Description	Set SP running speed.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.; fValue — Speed value that is set
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Axis Holding Input
Details	This parameter is carried into the motion buffer. When FORCE_SPEED is greater than SPEED, SPEED will also take effect to limit the maximum speed (SPEED does not work in versions after 140716). If FORCE_SPEED has been reduced to the corresponding speed when entering this section, please set STARTMOVE_SPEED.

Instruction 167	ZAux_Direct_SetStartMoveSpeed
Original Format	int32 __stdcall ZAux_Direct_SetStartMoveSpeed(ZMC_HANDLE handle, int iaxis, float fValue);
Description	The starting speed of SP movement with custom speed, this parameter is brought into the motion buffer, The unit is units/s. See the "STARTMOVE_SPEED" command in the software manual.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.; fValue — Speed value that is set
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	continuous interpolation SP motion
Details	It will take effect only when the motion command with SP is used.

	Please set a larger value when not in use. The controller default value is 1000.

Instruction 168	ZAux_Direct_GetStartMoveSpeed
Original Format	int32 __stdcall ZAux_Direct_GetStartMoveSpeed(ZMC_HANDLE handle, int iaxis, float * pfValue);
Description	Read starting speed of SP motion of custom speed.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.
Output parameters	Parameter name: fValue — SP starting speed value that is read.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	continuous interpolation SP motion
Details	It will take effect only when the motion command with SP is used. Please set a larger value when not in use. The controller default value is 1000.

Instruction 169	ZAux_Direct_GetFholdIn
Original Format	int32 __stdcall ZAux_Direct_GetFholdIn (ZMC_HANDLE handle , int iaxis, into*piValue);
Description	Read holding input.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.
Output parameters	Parameter name: fValue — Return input FHOLDIN input No.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	ZAux_Direct_SetFhspeed
Details	If there is an input signal, the speed of the moving axis is set by the FHSPEED parameter. And the current motion is not cancelled. When the input is canceled, the motion speed in process returns to the program speed. When the input is OFF, it is considered that there is a signal input. To reverse the effect, you can use INVERT_IN to invert the level (except for ECI series controllers).

	This parameter is only applicable to speed control mode (command with SP suffix). Therefore it does not work if movement is not velocity controlled (CAMBOX, CONNECT, MOVELINK).

Instruction 170	ZAux_Direct_SetFholdIn
Original Format	int32 __stdcall ZAux_Direct_SetFholdIn (ZMC_HANDLE handle, int iaxis, int iValue);
Description	Set hold input. Input No. corresponding to hold input, and -1 is invalid. If there is an input signal, the speed of the moving axis is set by the FHSPEED parameter. And the current motion is not cancelled. When the input is canceled, the motion speed in process returns to the program speed. It is only valid for SP motions.
Input parameters	Parameter name: handle — Link mark iaxis — Axis No. iValue — Set Input No.
Output parameters	Parameter name: fValue — Return input FHOLDIN input No.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Axis Holding Input
Details	If there is an input signal, the speed of the moving axis is set by the FHSPEED parameter. And the current motion is not cancelled. When the input is canceled, the motion speed in process returns to the program speed. When the input is OFF, it is considered that there is a signal input. To reverse the effect, you can use INVERT_IN to invert the level (except for ECI series controllers). This parameter is only applicable to speed control mode (command with SP suffix). Therefore it does not work if movement is not velocity controlled (CAMBOX, CONNECT, MOVELINK).

Instruction 171	ZAux_Direct_SetCornerMode
Original Format	int32 __stdcall ZAux_Direct_SetCornerMode(ZMC_HANDLE handle, int iaxis, int iValue);
Description	Set corner mode.
Input parameters	Parameter name: handle — Link mark iaxis — Axis No. iValue — Mode setting: Bit 0 — Value 1 — Reserved Bit 1 — Value 2 — Auto corner

				deceleration
		2	4	Reserved
		3	8	Auto small-round speed limit

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Corner deceleration
Details	Model: different bits represent different meanings. Please 6.5.2.2 -- CORNER_MODE instruction parameter description:

Instruction 172	**ZAux_Direct_GetCornerMode**
Original Format	int32 __stdcall ZAux_Direct_GetCornerMode (ZMC_HANDLE handle, int iaxis, int * piValue);
Description	Read corner deceleration mode.

Input parameters

Parameter name	Description
handle	Link mark
iaxis	Axis No.

Output parameters

Parameter name	Description
iValue	Mode setting:

Bit	Value	Description
0	1	Reserved
1	2	Auto corner deceleration
2	4	Reserved
3	8	Auto small-round speed limit

Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Corner deceleration
Details	Model: different bits represent different meanings. Please 6.5.2.2 -- CORNER_MODE instruction parameter description:

Instruction 173	**ZAux_Direct_SetDecelAngle**
Original Format	int32 __stdcall ZAux_Direct_SetDecelAngle(ZMC_HANDLE handle, int iaxis, float fValue);
Description	Set corner deceleration angle, starting deceleration angle, the unit is radian.

Input parameters

Parameter name	Description
handle	Link mark
iaxis	Axis No.
fValue	Corner deceleration angle that is set.

Output parameters

/

Return value

If it is successful, return value is 0, if not, please refer to error codes.

Example

Corner deceleration

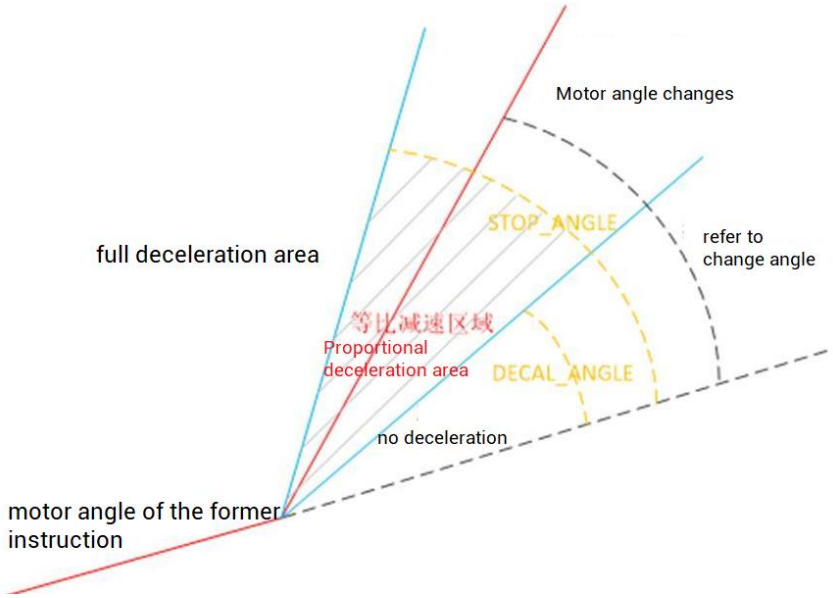
Details

The reference speed of the deceleration corner is based on the FORCE_SPEED speed, and a reasonable FORCE_SPEED must be set.

Angle conversion radian formula: angle value * (PI/180)

The deceleration angle refers to the change value of the reference angle of the motor relative to the previous movement. As shown below.

This angle value is not the angle of the actual trajectory, but the angle converted to the motor transformation. This angle value is only for reference.

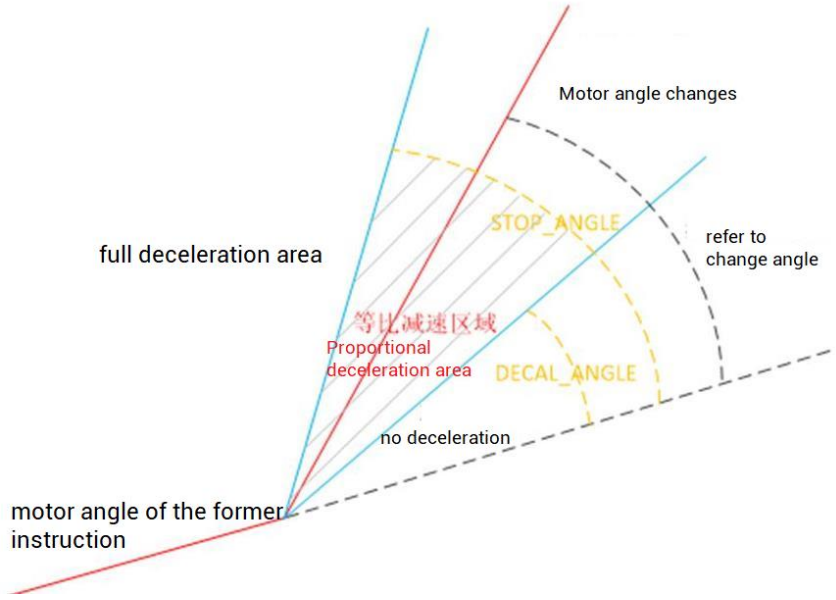


The absolute value is taken when the next interpolation motion track is below.

When a straight line is connected with an arc, the angle is calculated according to the tangent direction of the arc.

Used together with STOP_ANGLE, it will decelerate when the actual motion angle is between DECEL_ANGLE (upper limit) and STOP_ANGLE (lower limit).

Instruction 174	ZAux_Direct_GetDecelAngle

<table>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_GetDecelAngle (ZMC_HANDLE handle, int iaxis, float *pfValue);</td></tr>
<tr><td>Description</td><td>Read corner starting decelerating angle, the unit is radian.</td></tr>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link mark</td></tr>
<tr><td>iaxis</td><td>Axis No.</td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>fValue</td><td>Corner deceleration angle that is read.</td></tr>
</table>
</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Corner deceleration</td></tr>
<tr><td>Details</td><td>The reference speed of the deceleration corner is based on the FORCE_SPEED speed, and a reasonable FORCE_SPEED must be set.
Angle conversion radian formula: angle value * (PI/180)
The deceleration angle refers to the change value of the reference angle of the motor relative to the previous movement. As shown below.
This angle value is not the angle of the actual trajectory, but the angle converted to the motor transformation. This angle value is only for reference.
 

The absolute value is taken when the next interpolation motion track is below.
When a straight line is connected with an arc, the angle is calculated according to the tangent direction of the arc.
Used together with STOP_ANGLE, it will decelerate when the actual motion angle is between DECEL_ANGLE (upper limit) and</td></tr>
</table>

	STOP_ANGLE (lower limit).

Instruction 175	ZAux_Direct_SetFullSpRadius
Original Format	int32 __stdcall ZAux_Direct_SetFullSpRadius(ZMC_HANDLE handle, int iaxis, float fValue);
Description	Set small round speed limit radius.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.; fValue — Small-round radius
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Draw the circle with plane circle center + small-round speed limit
Details	When the radius is greater than the radius of the small circle speed limit, the speed is the speed value specified by the user program. When the radius is smaller than the radius of the small circle speed limit, the controller will reduce the speed proportionally.

Instruction 176	ZAux_Direct_GetFullSpRadius
Original Format	int32 __stdcall ZAux_Direct_GetFullSpRadius (ZMC_HANDLE handle , int iaxis, float *pfValue);
Description	Read the min radius of small-round speed limit.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.
Output parameters	Parameter name: fValue — Returned speed limit radius
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Draw the circle with plane circle center + small-round speed limit
Details	When the radius is greater than the radius of the small circle speed limit, the speed is the speed value specified by the user program. When the radius is smaller than the radius of the small circle speed limit, the controller will reduce the speed proportionally.

Instruction 177	ZAux_Direct_GetInterpFactor
Original	int32 __stdcall ZAux_Direct_GetInterpFactor (ZMC_HANDLE

<table>
<tr><td>Format</td><td>handle , int iaxis, int *piValue);</td></tr>
<tr><td>Description</td><td>When reading interpolation, whether the axis participates in the speed calculation, the default (1) means YES. This parameter only works for the third axis of the line and the helix.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr></table></td></tr>
<tr><td>Output parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>fValue</td><td>Returned speed calculation mode</td></tr></table></td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>/</td></tr>
<tr><td>Details</td><td>This parameter only affects any axis of the line and the third axis of the helix. Please restore it in time after motion, otherwise it will cause incorrect following motions.
When some axes do not participate in the speed calculation, first calculate the sub-velocity and total movement time of the participating axes according to the interpolation of the participating axes, and then divide the movement distance of the non-participating axes by the total time in turn to obtain corresponding speed of each axis that don't participate the calculation.
It is not possible to set all the axes of the interpolated actual motion to 0, which will cause the actual speed to be infinite.</td></tr>
</table>

<table>
<tr><td>Instruction 178</td><td>ZAux_Direct_SetInterpFactor</td></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_SetInterpFactor (ZMC_HANDLE handle , int iaxis, int iValue);</td></tr>
<tr><td>Description</td><td>When setting interpolation, whether the axis participates in the speed calculation, the default (1) means YES. This parameter only works for the third axis of the line and the helix.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr><tr><td>piValue</td><td>Mode: 0-not to participate, 1-participate</td></tr></table></td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>/</td></tr>
</table>

Details	This parameter only affects any axis of the line and the third axis of the helix. Please restore it in time after motion, otherwise it will cause incorrect following motions. When some axes do not participate in the speed calculation, first calculate the sub-velocity and total movement time of the participating axes according to the interpolation of the participating axes, and then divide the movement distance of the non-participating axes by the total time in turn to obtain corresponding speed of each axis that don't participate the calculation. It is not possible to set all the axes of the interpolated actual motion to 0, which will cause the actual speed to be infinite.

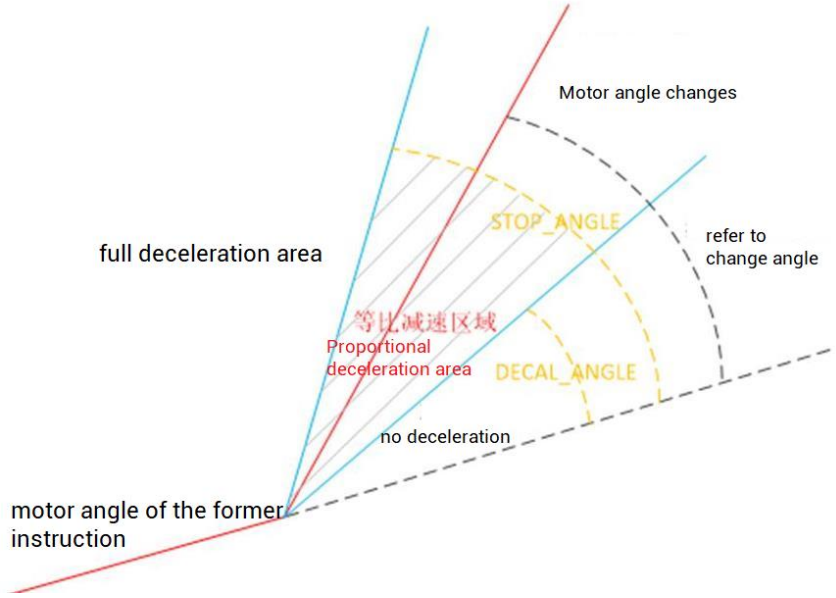
Instruction 179	**ZAux_Direct_SetMerge**
Original Format	int32 __stdcall ZAux_Direct_SetMerge(ZMC_HANDLE handle, int iaxis, int iValue)
Description	Set continuous interpolation switch.
Input parameters	Parameter name: Description handle: Link mark iaxis: Axis No. piValue: 0-OFF, 1-ON
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Corner deceleration
Details	When MERGE is set to ON, the multi-segment interpolation still slows down, the possible reasons are as follows: 1. Maybe MERGE is not set successfully, you can print it to check. 2. The controller is a point motion model, which does not support continuous interpolation, please contact the manufacturer. 3. Set CORNER_MODE corner deceleration, print to confirm. 4. The motion command with SP is used, and ENDMOVE_SPEED and STARTMOVE_SPEED are set. At this time, the speed is determined by these two commands. 5. The main axis is switched between multiple interpolations, and the master axis speed parameter is changed. 6. The MOVE_DELAY delay instruction is added between multiple interpolations, even if the delay is written to 0, it will cause deceleration.

Instruction 180	**ZAux_Direct_GetMerge**
Original	int32 __stdcall ZAux_Direct_GetMerge(ZMC_HANDLE handle, int

Format	iaxis, int *piValue)
Description	Read continuous interpolation switch state.
Input parameters	Parameter name: handle — Description: Link mark Parameter name: iaxis — Description: Axis No.
Output parameters	Parameter name: piValue — Description: Obtained continuous interpolation state: 0-OFF, 1-ON
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Corner deceleration
Details	When MERGE is set to ON, the multi-segment interpolation still slows down, the possible reasons are as follows: 1. Maybe MERGE is not set successfully, you can print it to check. 2. The controller is a point motion model, which does not support continuous interpolation, please contact the manufacturer. 3. Set CORNER_MODE corner deceleration, print to confirm. 4. The motion command with SP is used, and ENDMOVE_SPEED and STARTMOVE_SPEED are set. At this time, the speed is determined by these two commands. 5. The main axis is switched between multiple interpolations, and the master axis speed parameter is changed. 6. The MOVE_DELAY delay instruction is added between multiple interpolations, even if the delay is written to 0, it will cause deceleration.

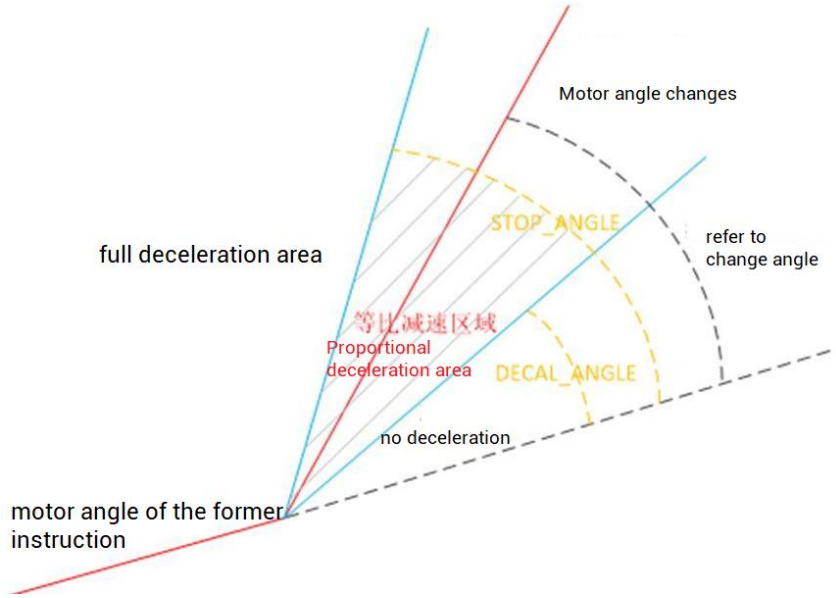
Instruction 181	**ZAux_Direct_SetStopAngle**
Original Format	int32 __stdcall ZAux_Direct_SetStopAngle(ZMC_HANDLE handle, int iaxis, float pfValue);
Description	Set stop deceleration angle.
Input parameters	Parameter name: handle — Description: Link mark Parameter name: iaxis — Description: Axis No. Parameter name: pfValue — Description: Stop deceleration angle
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Corner deceleration
Details	The reference speed of the deceleration corner is based on the FORCE_SPEED speed, and a reasonable FORCE_SPEED must be

set.
Angle conversion radian formula: angle value * (PI/180)
The deceleration angle refers to the change value of the reference angle of the motor relative to the previous movement. As shown below.
This angle value is not the angle of the actual trajectory, but the angle converted to the motor transformation. This angle value is only for reference.



The absolute value is taken when the next interpolation motion track is below.
When a straight line is connected with an arc, the angle is calculated according to the tangent direction of the arc.
Used together with STOP_ANGLE, it will decelerate when the actual motion angle is between DECEL_ANGLE (upper limit) and STOP_ANGLE (lower limit).

Instruction 182	ZAux_Direct_GetStopAngle
Original Format	int32 __stdcall ZAux_Direct_GetStopAngle (ZMC_HANDLE handle , int iaxis, float * pfValue);
Description	Corner stop deceleration angle, the unit is radian. The reference speed of the deceleration corner is based on the FORCE_SPEED speed, and a reasonable FORCE_SPEED must be set.
Input parameters	Parameter name: handle — Description: Link mark Parameter name: iaxis — Description: Axis No.
Output parameters	Parameter name: pfValue — Description: Returned corner stop angle

Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Corner deceleration
Details	The reference speed of the deceleration corner is based on the FORCE_SPEED speed, and a reasonable FORCE_SPEED must be set. Angle conversion radian formula: angle value * (PI/180) The deceleration angle refers to the change value of the reference angle of the motor relative to the previous movement. As shown below. This angle value is not the angle of the actual trajectory, but the angle converted to the motor transformation. This angle value is only for reference.  The absolute value is taken when the next interpolation motion track is below. When a straight line is connected with an arc, the angle is calculated according to the tangent direction of the arc. Used together with STOP_ANGLE, it will decelerate when the actual motion angle is between DECEL_ANGLE (upper limit) and STOP_ANGLE (lower limit).

Instruction 183	**ZAux_Direct_SetZsmooth**
Original Format	int32 __stdcall ZAux_Direct_SetZsmooth (ZMC_HANDLE handle, int iaxis, float fValue);
Description	Set deceleration chamfer radius.
Input parameters	

Parameter name	Description
handle	Link mark
iaxis	Axis No.
pfValue	The value is set.

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Chamfer
Details	Automatically calculate the actual corner radius based on the corner angle. Limits to 50% when path is exceeded. At 90°, the corner radius is the set value.

Instruction 184	**ZAux_Direct_GetZsmooth**
Original Format	int32 __stdcall ZAux_Direct_GetZsmooth(ZMC_HANDLE handle, int iaxis, float *pfValue);
Description	Read global chamfer radius.
Input parameters	Parameter name: handle — Link mark Parameter name: iaxis — Axis No.
Output parameters	Parameter name: pfValue — Description: Returned chamfer radius
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Chamfer
Details	Automatically calculate the actual corner radius based on the corner angle. Limits to 50% when path is exceeded. At 90°, the corner radius is the set value.

Instruction 185	**ZAux_Direct_MovePause**
Original Format	int32 __stdcall ZAux_Direct_MovePause(ZMC_HANDLE handle,int iaxis, int imode)
Description	Pause of BASE axis movement is only valid during single-axis or multi-axis interpolation movement, and pause together when multi-axis is linked.
Input parameters	Parameter name: handle — Link mark Parameter name: iaxis — Axis No. Parameter name: imode — Pause method: 0: pause current motion. 1: pause when preparing to execute the next motion command after the current motion is completed. 2: pause when the next motion command is about to be executed after the current

		motion is completed, and the MARK marks of the two commands are different. This mode can be used when an action is implemented by multiple instructions, and can be paused after an entire action is completed. 3: forced pause, it can also enter the pause state in IDLE mode, 20170513 firmware version adds this function.	
Output parameters	/		
Return value	If it is successful, return value is 0, if not, please refer to error codes.		
Example	axis pause & resume		
Details	1. It is only valid during single-axis or multi-axis interpolation motion, and it will pause together when multi-axis is linked. 2. It can use AXISSTATUS to check whether there is a pause. 3. When the axis has paused or is not in motion, calling this command will have a warning output, but it will not affect the program running. Some motion does not support pause, such as VMOVE, synchronous motion instructions, etc.		

Instruction 186	**ZAux_Direct_Rapidstop**	
Original Format	int32 __stdcall ZAux_Direct_Rapidstop(ZMC_HANDLE handle, int imode)	
Description	All axes in axis list stop immediately, and the interpolation movement is also stopped if the axes are involved in the interpolation.	
Input parameters	Parameter name	Description
	handle	Link mark
	imode	model 0 cancel the current motion 1 cancel motions in buffer 2 cancel current motion and buffer motion 3 Immediately interrupt pulse sending
Output parameters	/	
Return value	If it is successful, return value is 0, if not, please refer to error codes.	
Example	/	
Details	Mode0~2 deceleration follows the maximum value of FASTDEC and DECEL. To call absolute position movement after RAPIDSTOP, WAIT IDLE	

	must first wait for the stop to complete.

Instruction 187	ZAux_Direct_MoveResume
Original Format	int32 __stdcall ZAux_Direct_MoveResume (ZMC_HANDLE handle , int iaxis);
Description	Cancel motion pause.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	axis pause & resume
Details	When the BASE axis pauses, continue to moving from where it paused. AXISSTATUS can be used to check whether there is a pause.

Instruction 188	ZAux_Direct_CancelAxisList
Original Format	int32 __stdcall ZAux_Direct_CancelAxisList(ZMC_HANDLE handle, int imaxaxises, int *piAxislist, int imode)
Description	All axes in axis list stop immediately, and the interpolation movement is also stopped if the axes are involved in the interpolation.
Input parameters	Parameter name: handle — Link mark; imaxaxises — Total motion axes; piAxislist — Axis list array; imode — model 0 cancel the current motion 1 cancel motions in buffer 2 cancel current motion and buffer motion 3 Immediately interrupt pulse sending
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	Mode0~2 deceleration follows the maximum value of FASTDEC and DECEL.

<table>
<tr><th>Instruction 189</th><th>ZAux_Direct_Single_Cancel</th></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_Single_Cancel (ZMC_HANDLE handle, int iaxis, int imode);</td></tr>
<tr><td>Description</td><td>Stop single-axis motion.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr><tr><td>imode</td><td>model
0 cancel the current motion (default)
1 cancel motions in buffer
2 cancel current motion and buffer motion
3 Immediately interrupt pulse sending</td></tr></table></td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>single-axis continuous motion</td></tr>
<tr><td>Details</td><td>If the specified axis is in the interpolation motion axis list, regardless of the main axis or other axes, stop the interpolation motion of the axis group
MODE0~2 deceleration follows the maximum value of FASTDEC and DECEL.
To call absolute position movement after CANCEL, you must wait for the stop to complete.</td></tr>
</table>

<table>
<tr><th>Instruction 190</th><th>ZAux_Direct_GetRepDist</th></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_GetRepDist (ZMC_HANDLE handle , int iaxis, float *pfValue);</td></tr>
<tr><td>Description</td><td>Read DPOS and MPOS coordinates that is set to cycle automatically through REP_OPTION configuration.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr></table></td></tr>
<tr><td>Output parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>pfValue</td><td>Returned cyclic coordinate values</td></tr></table></td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Coordinate Cycle</td></tr>
<tr><td>Details</td><td>/</td></tr>
</table>

<table>
<tr><th>Instruction 191</th><th>ZAux_Direct_SetRepDist</th></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_SetRepDist (ZMC_HANDLE handle , int iaxis, float fValue);</td></tr>
<tr><td>Description</td><td>Automatically cycle DPOS and MPOS coordinates according to REP_OPTION configuration.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr></table></td></tr>
<tr><td>Output parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>pfValue</td><td>Configured cyclic coordinate values</td></tr></table></td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Coordinate Cycle</td></tr>
<tr><td>Details</td><td>/</td></tr>
</table>

<table>
<tr><th>Instruction 192</th><th>ZAux_Direct_GetRepOption</th></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_GetRepOption (ZMC_HANDLE handle , int iaxis, int *piValue);</td></tr>
<tr><td>Description</td><td>Read coordinates repeated configuration.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr></table></td></tr>
<tr><td>Output parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>pfValue</td><td>Returned mode:<table><tr><td>Bit</td><td>Value</td><td>Description</td></tr><tr><td>0</td><td>1</td><td>0: loop range: – REP_DIST ~ + REP_DIST
1: loop range: 0 ~ + REP_DIST</td></tr><tr><td>1</td><td>2</td><td>1: prohibit the repeated movement of CAMBOX and MOVELINK, and return to 0 after the prohibition takes effect</td></tr><tr><td>2</td><td>4</td><td>Reserved</td></tr><tr><td>4</td><td>16</td><td>1: do not use REP_DIST</td></tr></table></td></tr></table></td></tr>
</table>

				0: use REP_DIST

Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Coordinate Cycle
Details	It can be used to limit the coordinate cycle range of the cam's main axis to realize the continuity of multiple cam curves. When using the absolute motion mode, if the target position is within the range of the coordinate cycle, it can move correctly, if it is outside the range of the coordinate cycle, the motion is incorrect. Relative motion is not affected.

Instruction 193	**ZAux_Direct_SetRepOption**
Original Format	int32 __stdcall ZAux_Direct_SetRepOption (ZMC_HANDLE handle , int iaxis, int iValue);
Description	Set coordinate loop mode.

Input parameters

Parameter name	Description
handle	Link mark
iaxis	Axis No.
iValue	Different bits represent different meanings:

Bit	Value	Description
0	1	0: loop range: – REP_DIST ~ + REP_DIST 1: loop range: 0 ~ + REP_DIST
1	2	1: prohibit the repeated movement of CAMBOX and MOVELINK, and return to 0 after the prohibition takes effect
2	4	Reserved
4	16	1: do not use REP_DIST 0: use REP_DIST

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.

Example	Coordinate Cycle
Details	It can be used to limit the coordinate cycle range of the cam's main axis to realize the continuity of multiple cam curves. When using the absolute motion mode, if the target position is within the range of the coordinate cycle, it can move correctly, if it is outside the range of the coordinate cycle, the motion is incorrect. Relative motion is not affected.

Instruction 194	**ZAux_Direct_Pitchset**
Original Format	int32 __stdcall ZAux_Direct_Pitchset(ZMC_HANDLE handle,int iaxis,int iEnable,float StartPos,uint32 maxpoint,float DisOne ,uint32 TablNum,float * pfDisancelist)
Description	Set axis' pitch compensation, it is invalid for expansion axes.
Input parameters	Parameter name: Description handle: Link mark iaxis: Axis No. iEnable: Whether to open compensation, 1: ON, 0: OFF. StartPos: Staring compensation MPOS position maxpoint: Total points in compensation area DisOne: Each compensation point's gap TablNum: Compensation coordinate values are filled into starting guidance address of TABLE system array. pfDisancelist: Area compensation value list
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Pitch Compensation
Details	The number of compensated pulses of each point are stored in TABLE.

Instruction 195	**ZAux_Direct_Pitchset2**
Original Format	int32 __stdcall ZAux_Direct_Pitchset2(ZMC_HANDLE handle,int iaxis,int iEnable,float StartPos,uint32 maxpoint,float DisOne ,uint32 TablNum,float * pfDisancelist ,uint32 RevTablNum,float * RevpfDisancelist)
Description	Set axis' pitch compensation 2, it is invalid for expansion axes.

Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr><tr><td>iEnable</td><td>Whether to open compensation, 1: ON, 0: OFF.</td></tr><tr><td>StartPos</td><td>Staring compensation MPOS position</td></tr><tr><td>maxpoint</td><td>Total points in compensation area</td></tr><tr><td>DisOne</td><td>Each compensation point's gap</td></tr><tr><td>TabINum</td><td>Compensation coordinate values are filled into starting guidance address of TABLE system array.</td></tr><tr><td>pfDisancelist</td><td>Area compensation value list</td></tr><tr><td>RevTabINum</td><td>reverse compensation coordinate values are filled into starting guidance address of TABLE system array.</td></tr><tr><td>RevpfDisancelist</td><td>Reverse area compensation list, compensation data direction is consistent with forward.</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	/

Instruction 196	**ZAux_Direct_GetPitchStatus**
Original Format	int32 __stdcall ZAux_Direct_GetPitchStatus(ZMC_HANDLE handle,int iaxis,int * IfEnable,float * PitchDist)
Description	Set reading axis' pitch compensation state, it is invalid for expansion axes.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr></table>
Output parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>iEnable</td><td>Whether to open compensation, 1: ON, 0: OFF.</td></tr><tr><td>PicthDist</td><td>Compensation distance</td></tr></table>
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/

Details	/

Instruction 197	ZAux_Direct_GetIn
Original Format	int32 __stdcall ZAux_Direct_GetIn(ZMC_HANDLE handle,int ionum,uint32 *piValue)
Description	Read input state, please refer to IN command in ZBasic Manual.
Input parameters	Parameter name: handle — Link mark; ionum — Input No.
Output parameters	Parameter name: piValue — Get input state values
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	IO reading and setting.
Details	1. If level-inverse is set, what is read is the state after INVERT_IN flipping. 2. The IO channel number of the ZIO expansion board is related to the dial code. The initial value is (16 + dial code combination value * 16). The EIO bus expansion IO uses the NODE_IO command, which can only be set to a multiple of 8. Please refer to the hardware manual for details. Note: the IO mapping number must be greater than the maximum IO number of the controller itself, and cannot coincide with the number of the controller.

Instruction 198	ZAux_Direct_SetOp
Original Format	int32 __stdcall ZAux_Direct_SetOp(ZMC_HANDLE handle, int ionum,uint32 iValue);
Description	Open output, please refer to OP command in ZBasic Manual.
Input parameters	Parameter name: handle — Link mark; ionum — Input No.; piValue — Output port's state values that are set.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	IO reading and setting.
Details	1. If level-inverse is set, what is read is the state after INVERT_IN

	flipping. 2. The IO channel number of the ZIO expansion board is related to the dial code. The initial value is (16 + dial code combination value * 16). The EIO bus expansion IO uses the NODE_IO command, which can only be set to a multiple of 8. Please refer to the hardware manual for details. Note: the IO mapping number must be greater than the maximum IO number of the controller itself, and cannot coincide with the number of the controller.

Instruction 199	**ZAux_Direct_GetOp**
Original Format	int32 __stdcall ZAux_Direct_GetOp(ZMC_HANDLE handle, int ionum, uint32 *piValue);
Description	Read output state, please refer to IN command in ZBasic Manual.
Input parameters	Parameter name: handle — Link mark; ionum — output No.
Output parameters	Parameter name: piValue — Get output state values
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	IO reading and setting.
Details	The IO channel number of the ZIO expansion board is related to the dial code. The initial value is (16 + dial code combination value * 16). The EIO bus expansion IO uses the NODE_IO command, which can only be set to a multiple of 8. Please refer to the hardware manual for details. Note: the IO mapping number must be greater than the maximum IO number of the controller itself, and cannot coincide with the number of the controller. 32 outputs can be operated at most.

Instruction 200	**ZAux_GetModbusIn**
Original Format	int32 __stdcall ZAux_GetModbusIn (ZMC_HANDLE handle,int ionumfirst, int ionumend, uint8 * pValueList);
Description	Rapidly read current multiple input states.
Input parameters	Parameter name: handle — Link mark; ionumfirst — Input starting No.; ionumend — Input end No.

Output parameters	Parameter name: pValueList; Description: Bit state, store by bit
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Multiple IO reading
Details	The state obtained by the Modbus method is the state before inversion. If INVERT_IN is used to reverse, the read status may be wrong.

Instruction 201	**ZAux_GetModbusOut**
Original Format	int32 __stdcall ZAux_GetModbusOut (ZMC_HANDLE handle,int ionumfirst, int ionumend, uint8 * pValueList);
Description	Rapidly read multiple current output states. The state obtained by the Modbus method is the state before inversion. If INVERT_IN is used to reverse, the read status may be wrong.
Input parameters	handle: Link mark ionumfirst: Input starting No. ionumend: Input end No.
Output parameters	pValueList: Bit state, store by bit
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Multiple IO reading
Details	The state obtained by the Modbus method is the state before inversion. If INVERT_IN is used to reverse, the read status may be wrong.

Instruction 202	**ZAux_Direct_SetOutMulti**
Original Format	int32 __stdcall ZAux_SetOutMulti (ZMC_HANDLE handle, uint16 iofirst, uint16 ioend, uint32 *istate);
Description	Io interface is set to multiple outputs.
Input parameters	handle: Link mark ionumfirst: Input starting No. ionumend: Input end No. istate: Output state, istate set bit by bit, one UNIT corresponds to 32 outputs.
Output parameters	/

Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Multiple IO reading
Details	/

Instruction 203	**ZAux_Direct_GetOutMulti**
Original Format	int32 __stdcall ZAux_direct_GetOutMulti (ZMC_HANDLE handle, uint16 iofirst, uint16 ioend, uint32 * istate);
Description	Io interface is set to multiple outputs.
Input parameters	Parameter name: handle — Link mark; iofirst — IO starting No.; ioend — IO end No.
Output parameters	Parameter name: istate — Output state, istate set bit by bit, one UNIT corresponds to 32 outputs.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Multiple IO reading
Details	/

Instruction 204	**ZAux_Direct_GetInMulti**
Original Format	int32 __stdcall ZAux_Direct_GetInMulti (ZMC_HANDLE handle, int startio , int endio, int32 *piValue);
Description	Io interface is set to multiple outputs.
Input parameters	Parameter name: handle — Link mark; iofirst — IO starting No.; ioend — IO end No.
Output parameters	Parameter name: istate — Output state, istate set bit by bit, one UNIT corresponds to 32 outputs.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Multiple IO reading
Details	/

Instruction 205	**ZAux_Direct_SetInvertIn**
Original Format	int32 __stdcall ZAux_Direct_SetInvertIn(ZMC_HANDLE handle, int ionum, int bifInvert)
Description	Set reverse input state, please refer to INVERT_IN command in

	ZBasic.
Input parameters	Parameter name: handle — Description: Link mark Parameter name: ionum — Description: Input No. Parameter name: BifInvert — Description: State 0: inverse is ON, 1: inverse is OFF.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Position limit settings
Details	For ZMC series controllers, it considers that there is signal input when the input is OFF (it is opposite for ECI series controllers).

Instruction 206	**ZAux_Direct_GetInvertIn**
Original Format	int32 __stdcall ZAux_Direct_GetInvertIn(ZMC_HANDLE handle, int ionum, int * piValue)
Description	Read reverse input state.
Input parameters	Parameter name: handle — Description: Link mark Parameter name: ionum — Description: Input No.
Output parameters	Parameter name: piValue — Description: Returned inverse input state. 0: inverse is ON, 1: inverse is OFF.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Position limit settings
Details	For ZMC series controllers, it considers that there is signal input when the input is OFF (it is opposite for ECI series controllers).

Instruction 207	**ZAux_Direct_GetAD**
Original Format	int32 __stdcall ZAux_Direct_GetAD(ZMC_HANDLE handle, int ionum , float *pfValue);
Description	Read analog input values. Please refer to AIN instruction in ZBasic.
Input parameters	Parameter name: handle — Description: Link mark Parameter name: ionum — Description: AIN No.
Output parameters	Parameter name: pfValue — Description: Returned analog values

Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	AD/DA Setting & Getting
Details	The 12-bit scale value ranges from 0 to 4095 corresponding to a voltage of 0 to 10V. 16-bit scale value range 0~65536 corresponds to 0~10V voltage. The AD channel number of the ZAIO expansion board is related to the dial code. The initial value is (8 + dial code combination value * 8). ZMIO bus expansion AD can only be set to a multiple of 8 through the NODE_AIO command. Please refer to the hardware manual for details. Note: the AIO mapping number must be greater than the maximum AIO number of the controller itself, and cannot coincide with the number of the controller.

Instruction 208	**ZAux_Direct_SetDA**
Original Format	int32 __stdcall ZAux_Direct_SetDA(ZMC_HANDLE handle, int ionum, float fValue);
Description	Read analog output values. Please refer to AOUT instruction in ZBasic.
Input parameters	Parameter name: Description handle: Link mark ionum: DA output No. fValue: Configured value.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	AD/DA Setting & Getting
Details	12-bit scale value ranges from 0 to 4095 corresponding to a voltage of 0 to 10V. 16-bit scale value range 0~65536 corresponds to 0~10V voltage.

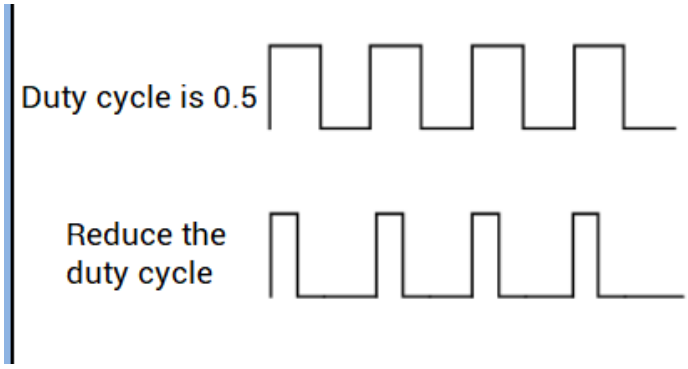
Instruction 209	**ZAux_Direct_GetDA**
Original Format	int32 __stdcall ZAux_Direct_GetDA(ZMC_HANDLE handle, int ionum, float *pfValue);
Description	Read analog output values. Please refer to AOUT instruction in ZBasic.

Input parameters	Parameter name: handle — Link mark; ionum — DA output No.
Output parameters	Parameter name: piValue — Returned output values
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	AD/DA Setting & Getting
Details	12-bit scale value ranges from 0 to 4095 corresponding to a voltage of 0 to 10V. 16-bit scale value range 0~65536 corresponds to 0~10V voltage.

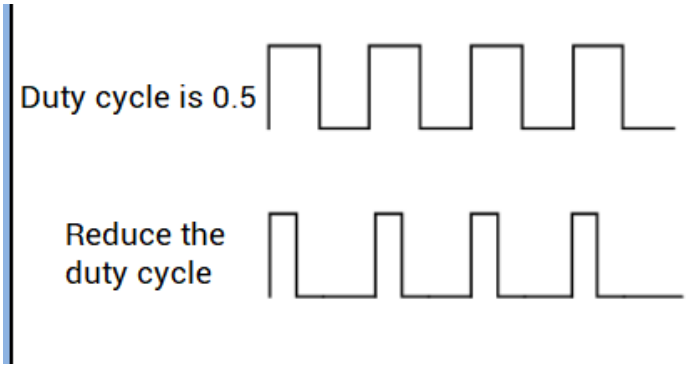
Instruction 210	**ZAux_Direct_SetPwmFreq**
Original Format	int32 __stdcall ZAux_Direct_SetPwmFreq(ZMC_HANDLE handle, int ionum, float fValue);
Description	Set PWM frequency, please refer to "PWM_DREQ" command in ZBasic.
Input parameters	Parameter name: handle — Link mark; ionum — PWM No.; fValue — Configured value.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	PWM Usage
Details	PWM can only be turned off by setting the duty cycle to 0, not by setting the PWM frequency to 0, and do not set the frequency to 0, the PWM frequency must be adjusted before the PWM switch.

Instruction 211	**ZAux_Direct_GetPwmFreq**
Original Format	int32 __stdcall ZAux_Direct_GetPwmFreq(ZMC_HANDLE handle, int ionum, float *pfValue);
Description	Read PWM frequency.
Input parameters	Parameter name: handle — Link mark; ionum — PWM No.

Output parameters	Parameter name: fValue — Description: Configured value.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	PWM Usage
Details	PWM can only be turned off by setting the duty cycle to 0, not by setting the PWM frequency to 0, and do not set the frequency to 0, the PWM frequency must be adjusted before the PWM switch.

Instruction 212	**ZAux_Direct_SetPwmDuty**
Original Format	int32 __stdcall ZAux_Direct_SetPwmDuty(ZMC_HANDLE handle, int ionum, float fValue);
Description	Set PWM duty cycle, please refer to "PWM_DUTY" command in ZBasic.
Input parameters	Parameter name: handle — Description: Link mark Parameter name: ionum — Description: PWM No. Parameter name: fValue — Description: Configured value.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	PWM Usage
Details	PWM can only be turned off by setting the duty cycle to 0, not by setting the PWM frequency to 0, and do not set the frequency to 0, the PWM frequency must be adjusted before the PWM switch. The duty cycle refers to the ratio of the active level to the entire period. In one cycle, the active level is output first, and then the invalid level is output.  The actual output of the PWM is controlled by the output port, and the PWM output must be turned on when the output port is turned on, otherwise the output will be shielded. The PWM function can be

	turned on first, and then the output port can be turned on, so as to realize the first pulse suppression function of the laser power supply.

Instruction 213	ZAux_Direct_GetPwmDuty
Original Format	int32 __stdcall ZAux_Direct_GetPwmDuty(ZMC_HANDLE handle, int ionum, float *pfValue);
Description	Read pwm duty cycle.
Input parameters	Parameter name: handle — Link mark; ionum — PWM No.
Output parameters	Parameter name: fValue — Configured value.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	PWM Usage
Details	PWM can only be turned off by setting the duty cycle to 0, not by setting the PWM frequency to 0, and do not set the frequency to 0, the PWM frequency must be adjusted before the PWM switch. The duty cycle refers to the ratio of the active level to the entire period. In one cycle, the active level is output first, and then the invalid level is output.  The actual output of the PWM is controlled by the output port, and the PWM output must be turned on when the output port is turned on, otherwise the output will be shielded. The PWM function can be turned on first, and then the output port can be turned on, so as to realize the first pulse suppression function of the laser power supply.

Instruction 214	ZAux_Direct_SetFsLimit
Original Format	int32 __stdcall ZAux_Direct_SetFsLimit(ZMC_HANDLE handle, int iaxis, float fValue)

<table>
<tr><td>Description</td><td>Set the positive soft limit of the axis, and set a larger value when canceling the soft limit. the unit is units.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr><tr><td>fValue</td><td>Configured positive position limit value.</td></tr></table></td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Position limit settings</td></tr>
<tr><td>Details</td><td>When FS_LIMIT is greater than REP_DIST, the parameter has no effect, and the positive soft limit is prohibited.
When canceling the soft limit, it is recommended not to modify the value of REP_DIST, just set FS_LIMIT to a larger value. The value of FS_LIMIT is 200000000 by default.
The soft limit cannot be used as a reference for the limit signal when DATUM returns to zero.
Note: before setting the limit, it needs to check whether there is a corresponding acceleration and deceleration, and fast deceleration. If there is no setting, it will not stop when the limit is encountered.</td></tr>
</table>

<table>
<tr><td>Instruction 215</td><td>ZAux_Direct_GetFsLimit</td></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_GetFsLimit(ZMC_HANDLE handle, int iaxis, float *fValue)</td></tr>
<tr><td>Description</td><td>Read the positive soft limit value.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr></table></td></tr>
<tr><td>Output parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>fValue</td><td>Returned positive position limit value</td></tr></table></td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Position limit settings</td></tr>
<tr><td>Details</td><td>When FS_LIMIT is greater than REP_DIST, the parameter has no effect, and the positive soft limit is prohibited.
When canceling the soft limit, it is recommended not to modify the value of REP_DIST, just set FS_LIMIT to a larger value. The value of FS_LIMIT is 200000000 by default.</td></tr>
</table>

	The soft limit cannot be used as a reference for the limit signal when DATUM returns to zero. Note: before setting the limit, it needs to check whether there is a corresponding acceleration and deceleration, and fast deceleration. If there is no setting, it will not stop when the limit is encountered.

Instruction 216	ZAux_Direct_SetRsLimit
Original Format	int32 __stdcall ZAux_Direct_SetRsLimit(ZMC_HANDLE handle, int iaxis, float fValue)
Description	Set the negative soft limit of the axis, and set a larger value when canceling the soft limit. the unit is units.
Input parameters	Parameter name: Description handle: Link mark iaxis: Axis No. fValue: Configured negative position limit value.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Position limit settings
Details	When FS_LIMIT is greater than REP_DIST, the parameter has no effect, and the positive soft limit is prohibited. When canceling the soft limit, it is recommended not to modify the value of REP_DIST, just set FS_LIMIT to a larger value. The value of FS_LIMIT is 200000000 by default. The soft limit cannot be used as a reference for the limit signal when DATUM returns to zero. Note: before setting the limit, it needs to check whether there is a corresponding acceleration and deceleration, and fast deceleration. If there is no setting, it will not stop when the limit is encountered.

Instruction 217	ZAux_Direct_GetRsLimit
Original Format	int32 __stdcall ZAux_Direct_GetRsLimit(ZMC_HANDLE handle, int iaxis, float *fValue)
Description	Read the negative soft limit value.
Input parameters	Parameter name: Description handle: Link mark iaxis: Axis No.

Output parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>fValue</td><td>Returned negative position limit value</td></tr></table>
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Position limit settings
Details	When FS_LIMIT is greater than REP_DIST, the parameter has no effect, and the positive soft limit is prohibited. When canceling the soft limit, it is recommended not to modify the value of REP_DIST, just set FS_LIMIT to a larger value. The value of FS_LIMIT is 200000000 by default. The soft limit cannot be used as a reference for the limit signal when DATUM returns to zero. Note: before setting the limit, it needs to check whether there is a corresponding acceleration and deceleration, and fast deceleration. If there is no setting, it will not stop when the limit is encountered.

Instruction 218	**ZAux_Direct_SetFwdIn**
Original Format	int32 __stdcall ZAux_Direct_SetFwdIn(ZMC_HANDLE handle, int iaxis,int iValue)
Description	Set the positive limit signal, -1 means cancel, refer to the "FWD_IN" command in ZBasic. Note: before setting the limit, it needs to check whether there is a corresponding acceleration and deceleration, and fast deceleration. If there is no setting, it will not stop when the limit is encountered.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr><tr><td>iValue</td><td>IO No., -1 means cancelling the setting.</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Position limit settings
Details	1. Before setting the limit, it needs to check whether there is a corresponding acceleration and deceleration, and fast deceleration. If there is no setting, it will not stop when the limit is encountered. 2. When input is OFF, it considers there is signal input. INVERT_IN can be used to get opposite effect (and the situation is opposite to ECI series controllers).

	3. Please note don't set incorrectly, otherwise, it will keep moving when encountered position limit.

Instruction 219	ZAux_Direct_GetFwdIn
Original Format	int32 __stdcall ZAux_Direct_GetFwdIn (ZMC_HANDLE handle, int iaxis, int * piValue)
Description	Select positive position limit IO of axis. Note: before setting the limit, it needs to check whether there is a corresponding acceleration and deceleration, and fast deceleration. If there is no setting, it will not stop when the limit is encountered.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.
Output parameters	Parameter name: piValue — Returned corresponding IN No., -1 means not to set.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Position limit settings
Details	1. Before setting the limit, it needs to check whether there is a corresponding acceleration and deceleration, and fast deceleration. If there is no setting, it will not stop when the limit is encountered. 2. When input is OFF, it considers there is signal input. INVERT_IN can be used to get opposite effect (and the situation is opposite to ECI series controllers). 3. Please note don't set incorrectly, otherwise, it will keep moving when encountered position limit.

Instruction 220	ZAux_Direct_SetRevIn
Original Format	int32 __stdcall ZAux_Direct_SetRevIn(ZMC_HANDLE handle, int iaxis,int iValue);
Description	Set the negative limit signal, -1 means cancel, refer to the "FWD_IN" command in ZBasic. Note: before setting the limit, it needs to check whether there is a corresponding acceleration and deceleration, and fast deceleration. If there is no setting, it will not stop when the limit is encountered.

Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr><tr><td>iValue</td><td>IO No., -1 means cancelling the setting.</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Position limit settings
Details	1. Before setting the limit, it needs to check whether there is a corresponding acceleration and deceleration, and fast deceleration. If there is no setting, it will not stop when the limit is encountered. 2. When input is OFF, it considers there is signal input. INVERT_IN can be used to get opposite effect (and the situation is opposite to ECI series controllers). 3. Please note don't set incorrectly, otherwise, it will keep moving when encountered position limit.

Instruction 221	**ZAux_Direct_GetRevIn**
Original Format	int32 __stdcall ZAux_Direct_GetRevIn(ZMC_HANDLE handle, int iaxis,int* piValue);
Description	Select negative position limit IO of axis. Note: before setting the limit, it needs to check whether there is a corresponding acceleration and deceleration, and fast deceleration. If there is no setting, it will not stop when the limit is encountered.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr></table>
Output parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>piValue</td><td>Returned corresponding IN No., -1 means not to set.</td></tr></table>
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Position limit settings
Details	1. Before setting the limit, it needs to check whether there is a corresponding acceleration and deceleration, and fast deceleration. If there is no setting, it will not stop when the limit is encountered. 2. When input is OFF, it considers there is signal input. INVERT_IN can be used to get opposite effect (and the situation is

	opposite to ECI series controllers). 3. Please note don't set incorrectly, otherwise, it will keep moving when encountered position limit.

Instruction 222	ZAux_Direct_SetAlmIn
Original Format	int32 __stdcall ZAux_Direct_SetAlmIn(ZMC_HANDLE handle, int iaxis,int iValue)
Description	Set the axis alarm signal, -1 means cancel, refer to the "ALM_IN" command in ZBasic.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.; iValue — IO No., -1 means cancelling the setting.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Alarm IN Settings
Details	1. Before setting the alarm signal it needs to check whether there is a corresponding acceleration and deceleration, and fast deceleration. If there is no setting, it will not stop when the limit is encountered. 2. When input is OFF, it considers there is signal input. INVERT_IN can be used to get opposite effect (and the situation is opposite to ECI series controllers).

Instruction 223	ZAux_Direct_GetAlmIn
Original Format	int32 __stdcall ZAux_Direct_GetAlmIn(ZMC_HANDLE handle, int iaxis,int * piValue)
Description	Read alarm IO No. of selected axis.
Input parameters	Parameter name: handle — Link mark; iaxis — Axis No.
Output parameters	Parameter name: piValue — Returned corresponding IN No., -1 means not to set.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Alarm IN Settings
Details	1. Before setting the alarm signal it needs to check whether there is a corresponding acceleration and deceleration, and fast

	deceleration. If there is no setting, it will not stop when the limit is encountered. 2. When input is OFF, it considers there is signal input. INVERT_IN can be used to get opposite effect (and the situation is opposite to ECI series controllers).

Instruction 224	ZAux_Direct_SetVrf
Original Format	int32 __stdcall ZAux_Direct_SetVrf(ZMC_HANDLE handle,int vrstartnum, int numes, float *pfValue)；
Description	Set data in VR.
Input parameters	Parameter name: handle — Link mark vrstartnum — Starting No. that starting to operate VR numes — The number of writing pfValue — Data list
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	VR Register Usage
Details	32-bit float. Power-down storage register Note that the number of controllers of different models is different. Store floating-point numbers, sharing a space with VR_INT and VRSTRING.

Instruction 225	ZAux_Direct_GetVrf
Original Format	int32 __stdcall ZAux_Direct_GetVrf(ZMC_HANDLE handle, int vrstartnum, int numes, float *pfValue)
Description	Read data in VR.
Input parameters	Parameter name: handle — Link mark vrstartnum — Starting No. that starting to operate VR numes — The number of writing
Output parameters	Parameter name: pfValue — Data list
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	VR Register Usage
Details	32-bit float. Power-down storage register Note that the number of controllers of different models is different. Store floating-point numbers, sharing a space with VR_INT and VRSTRING.

<table>
<tr><th>Instruction 226</th><th>ZAux_Direct_GetVrInt</th></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_GetVrInt (ZMC_HANDLE handle ,int vrstartnum, int numes, int * piValue);</td></tr>
<tr><td>Description</td><td>Read VR data in integer. It is valid to read through VR_INT “DirectCommand” with firmware version above 150401.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>vrstartnum</td><td>Starting No. that starting to operate VR</td></tr><tr><td>numes</td><td>The number of writing</td></tr></table></td></tr>
<tr><td>Output parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>pfValue</td><td>Data list</td></tr></table></td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>/</td></tr>
<tr><td>Details</td><td>/</td></tr>
</table>

<table>
<tr><th>Instruction 227</th><th>ZAux_Direct_SetTable</th></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_SetTable(ZMC_HANDLE handle,int tabstart, int numes, float *pfValue)</td></tr>
<tr><td>Description</td><td>Write data into TABLE.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>tablestart</td><td>Starting No. that starting to operate TABLE</td></tr><tr><td>numes</td><td>The number of writing</td></tr><tr><td>pfValue</td><td>Data list</td></tr></table></td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>TABLE Register Usage</td></tr>
<tr><td>Details</td><td>It is a super large array that comes with the controller. The data type is 32-bit floating point (64-bit floating point for 4 series and above), and it will not be saved when power off.</td></tr>
</table>

<table>
<tr><th>Instruction 228</th><th>ZAux_Direct_GetTable</th></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_GetTable(ZMC_HANDLE handle, int tabstart, int numes, float *pfValue)</td></tr>
<tr><td>Description</td><td>Read data in TABLE.</td></tr>
</table>

Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>tablestart</td><td>Starting No. that starting to operate TABLE</td></tr><tr><td>numes</td><td>The number of writing</td></tr></table>
Output parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>pfValue</td><td>Data list</td></tr></table>
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	TABLE Register Usage
Details	It is a super large array that comes with the controller. The data type is 32-bit floating point (64-bit floating point for 4 series and above), and it will not be saved when power off.

Instruction 229	**ZAux_Modbus_Set0x**
Original Format	int32 __stdcall ZAux_Modbus_Set0x(ZMC_HANDLE handle, uint16 start, uint16 inum, uint8* pdata)
Description	Set modbus bit register. MODBUS_REG setting.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>start</td><td>Starting No. that starting to operate modbus_bit</td></tr><tr><td>inum</td><td>The number of writing</td></tr><tr><td>pdata</td><td>Data list</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	MODBUS Register Usage
Details	/

Instruction 230	**ZAux_Modbus_Get0x**
Original Format	int32 __stdcall ZAux_Modbus_Get0x(ZMC_HANDLE handle, uint16 start, uint16 inum, uint8* pdata)
Description	Read modbus bit register. MODBUS_REG reading.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark</td></tr><tr><td>start</td><td>Starting No. that starting to operate modbus_bit</td></tr><tr><td>inum</td><td>The number of writing</td></tr></table>

Output parameters	Parameter name: pdata; Description: Data list
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	MODBUS Register Usage
Details	/

Instruction 231	**ZAux_Modbus_Set4x**
Original Format	int32 __stdcall ZAux_Modbus_Set4x(ZMC_HANDLE handle, uint16 start, uint16 inum, uint16* pdata)
Description	Set modbus word register.
Input parameters	handle: Link mark start: Starting No. that starting to operate modbus_reg inum: The number of writing pdata: Data list
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	MODBUS Register Usage
Details	MODBUS word register, this function corresponds to modbus_reg. Please refer to 10.3

Instruction 232	**ZAux_Modbus_Get4x**
Original Format	int32 __stdcall ZAux_Modbus_Get4x(ZMC_HANDLE handle, uint16 start, uint16 inum, uint16* pdata)
Description	Read modbus word register.
Input parameters	handle: Link mark start: Starting No. that starting to operate modbus_reg inum: The number of writing
Output parameters	pdata: Data list
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	MODBUS Register Usage
Details	MODBUS word register, this function corresponds to modbus_reg.

	Please refer to 10.3

Instruction 233	ZAux_Modbus_Set4x_Long
Original Format	int32 __stdcall ZAux_Modbus_Set4x_Long (ZMC_HANDLE handle, uint16 start, uint16 inum, int32 * pidata);
Description	Set MODBUS_LONG.
Input parameters	Parameter name: handle — Link mark start — Starting No. that starting to operate modbus_long inum — The number of writing pdata — Data list
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	One MODBUS_LONG occupies two MODBUS_REG addresses, for example, MODBUS_LONG (0) occupies MODBUS_REG(0) and MODBUS_REG(1). MODBUS word register, this function corresponds to modbus_long. Please refer to 10.3

Instruction 234	ZAux_Modbus_Get4x_Long
Original Format	int32 __stdcall ZAux_Modbus_Get4x_Long (ZMC_HANDLE handle, uint16 start, uint16 inum, int32 * pidata);
Description	Read modbus_long.
Input parameters	Parameter name: handle — Link mark start — Starting No. that starting to operate modbus_long inum — The number of reading.
Output parameters	Parameter name: pdata — Data list
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	One MODBUS_LONG occupies two MODBUS_REG addresses, for example, MODBUS_LONG (0) occupies MODBUS_REG(0) and MODBUS_REG(1). MODBUS word register, this function corresponds to modbus_long.

	Please refer to 10.3

Instruction 235	ZAux_Modbus_Set4x_String
Original Format	int32 __stdcall ZAux_Modbus_Set4x_String (ZMC_HANDLE handle , uint16 start,uint16 inum, char * pidata);
Description	Set modbus_string.
Input parameters	Parameter name: handle — Link mark; start — Modbus starting address; inum — Length; pdata — Written character string.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	MODBUS word register, this function corresponds to modbus_long. Please refer to 10.3

Instruction 236	ZAux_Modbus_Get4x_String
Original Format	int32 __stdcall ZAux_Modbus_Get4x_String (ZMC_HANDLE handle , uint16 start,uint16 inum, char * pidata);
Description	Read modbus_string.
Input parameters	Parameter name: handle — Link mark; start — Modbus starting address; inum — Length
Output parameters	Parameter name: pdata — Read character string.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	MODBUS word register, this function corresponds to modbus_long. Please refer to 10.3

Instruction 237	ZAux_Modbus_Set4x_Float
Original Format	int32 __stdcall ZAux_Modbus_Set4x_Float (ZMC_HANDLE handle, uint16 start, uint16 inum, float * pfdata);
Description	Set MODBUS_IEEE

Input parameters	Parameter name: handle — Description: Link mark Parameter name: start — Description: Starting No. that starting to operate modbus_long Parameter name: inum — Description: The number of writing Parameter name: pdata — Description: Data list
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	One MODBUS_IEEE occupies two MODBUS_REG addresses, for example, MODBUS_IEEE (0) occupies MODBUS_REG(0) and MODBUS_REG(1). MODBUS word register, this function corresponds to modbus_long. Please refer to 10.3

Instruction 238	**ZAux_Modbus_Get4x_Float**
Original Format	int32 __stdcall ZAux_Modbus_Get4x_Float(ZMC_HANDLE handle, uint16 start, uint16 inum, float * pfdata);
Description	Read MODBUS_IEEE.
Input parameters	Parameter name: handle — Description: Link mark Parameter name: start — Description: Starting No. that starting to operate modbus_long Parameter name: inum — Description: The number of writing
Output parameters	Parameter name: pdata — Description: Data list
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	One MODBUS_IEEE occupies two MODBUS_REG addresses, for example, MODBUS_IEEE (0) occupies MODBUS_REG(0) and MODBUS_REG(1). MODBUS word register, this function corresponds to modbus_long. Please refer to 10.3

Instruction 239	**ZAux_WriteUFile**
Original Format	int32 __stdcall ZAux_WriteUFile(const char *sFilename, float *pVarlist, int inum)
Description	Store variables list that is in float formation into file, and the controller USB file's format is consistent.

Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>sFilename</td><td>File name.</td></tr><tr><td>pVarlist</td><td>Data list.</td></tr><tr><td>inum</td><td>Data length.</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	File reading and writing test
Details	/

Instruction 240	**ZAux_ReadUFile**
Original Format	int32 __stdcall ZAux_ReadUFile(const char *sFilename, float *pVarlist, int*pinum)
Description	Read variables list that is in float formation, and the controller USB file's format is consistent.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>sFilename</td><td>File name.</td></tr><tr><td>inum</td><td>Data length.</td></tr></table>
Output parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>pVarlist</td><td>Data list.</td></tr></table>
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	File reading and writing test
Details	/

Instruction 241	**ZAux_FlashWritef**
Original Format	int32 __stdcall ZAux_FlashWritef(ZMC_HANDLE handle, uint16 uiflashid, uint32 uinumes, float *pfvlue)
Description	Write float data into Flash block.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link handle</td></tr><tr><td>uiflashid</td><td>FLASH block No.</td></tr><tr><td>uinumes</td><td>The number of writing</td></tr><tr><td>pfvlue</td><td>Data list</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.

Example	Flash reading and writing test
Details	The internal FLASH adopts the method of sequential storage, and the order of reading must be consistent with the order of storage. The internal FLASH has a limit on the number of storage times, so do not operate in a random cycle. Be careful not to operate the FLASH during the motion, it will affect the execution of the motion.

Instruction 242	**ZAux_FlashReadf**
Original Format	int32 __stdcall ZAux_FlashReadf(ZMC_HANDLE handle, uint16 uiflashid, uint32 uibuffnum, float *pfvlue, uint32* puinumesread)
Description	Read user data in float type from Flash block.
Input parameters	Parameter name: Description handle: Link handle uiflashid: FLASH block No. uibuffnum: The number of buffer variables
Output parameters	Parameter name: Description Pfvlue: Data list. puinumesread: The number of variables that is read.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Flash reading and writing test
Details	The internal FLASH adopts the method of sequential storage, and the order of reading must be consistent with the order of storage. The internal FLASH has a limit on the number of storage times, so do not operate in a random cycle. Be careful not to operate the FLASH during the motion, it will affect the execution of the motion.

Instruction 243	**ZAux_BusCmd_GetNodeNum**
Original Format	int32 __stdcall ZAux_BusCmd_GetNodeNum(ZMC_HANDLE handle,int slot,int *piValue);
Description	Read the number of nodes.
Input parameters	Parameter name: Description handle: Link handle slot: Slot No.
Output parameters	Parameter name: Description pivalue: Returned number of nodes.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	bus initialization (zmotiontools method)

Details	/

Instruction 244	ZAux_BusCmd_GetNodeInfo
Original Format	int32 __stdcall ZAux_BusCmd_GetNodeInfo(ZMC_HANDLE handle, int slot, int node, int sel, int *piValue);
Description	Read node information.
Input parameters	Parameter name: handle — Link handle slot — Slot No. node — Node No. sel — Information No.
Output parameters	Parameter name: pivalue — Returned parameter information.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	bus initialization (zmotiontools method)
Details	/

Instruction 245	ZAux_BusCmd_GetNodeStatus
Original Format	int32 __stdcall ZAux_Direct_GetNodeStatus (ZMC_HANDLE handle, uint32 slot , uint32 node , uint32 *nodestatus);
Description	Read node bus state.
Input parameters	Parameter name: handle — Link handle slot — Slot No. node — Node No.
Output parameters	Parameter name: nodestatus — nodestatus processes bit by bit. Bit0: whether the node exists. Bit1: communication state. Bit2: node state.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	Nodestatus: When the value is 1, bit 0 is 1, bit 1 and bit 2 are 0, the device communicates normally. When the value is 3, bit 0 and bit 1 are 1, bit 2 is 0, the device communicates abnormally.

BIT	Meaning
0	Indicate whether node exists or not. 1-YES
1	Communication state. 1-error.

	2	Node state. 1-error.

Instruction 246	ZAux_BusCmd_DriveClear
Original Format	int32 __stdcall ZAux_BusCmd_DriveClear(ZMC_HANDLE handle, int iaxis, uint32 mode);
Description	Set clearing bus servo alarms.
Input parameters	Parameter name: handle — Link handle iaxis — Axis No. mode — 0- clear current alarm; 1- clear history alarm; 2- clear external input alarm
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	When there is no error for driver, the command is used, controller will report 6015, but it will not influence program execution.

Instruction 247	ZAux_BusCmd_InitBus
Original Format	int32 __stdcall ZAux_BusCmd_InitBus(ZMC_HANDLE handle);
Description	Bus initialization.
Input parameters	Parameter name: handle — Link handle
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	bus initialization (zmotiontools method)
Details	Bus parameters configured by "Zmotion tools" tool software are valid for controller, and the basic file in the tool can also be downloaded to the controller through the ZAux_BasDown function for use.

Instruction 248	ZAux_BusCmd_GetInitStatus
Original Format	int32 __stdcall ZAux_BusCmd_GetInitStatus(ZMC_HANDLE handle,int *piValue);
Description	Get bus initialization state.

Input parameters	Parameter name: handle — Description: Link handle
Output parameters	Parameter name: pivalue — Description: Returned state, 0: fail to initialize, 1: success to initialize.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	bus initialization (zmotiontools method)
Details	Bus parameters configured by "Zmotion tools" tool software are valid for controller, and the basic file in the tool can also be downloaded to the controller through the ZAux_BasDown function for use.

Instruction 249	**ZAux_Direct_SetErrormask**
Original Format	int32 __stdcall ZAux_Direct_SetErrormask (ZMC_HANDLE handle, int iaxis , int iValue);
Description	Set error mark.
Input parameters	Parameter name: handle — Description: Link handle Parameter name: iaxis — Description: Axis No. Parameter name: iValue — Description: The value that is set.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	To do "and" operation with AXISSTATUS to decide which errors' WDOG need to be closed.

Instruction 250	**ZAux_Direct_GetErrormask**
Original Format	int32 __stdcall ZAux_Direct_GetErrormask (ZMC_HANDLE handle, int iaxis , int * piValue);
Description	Get error mark. To do "and" operation with AXISSTATUS to decide which errors' WDOG need to be closed.
Input parameters	Parameter name: handle — Description: Link handle Parameter name: iaxis — Description: Axis No. Parameter name: iValue — Description: The value that is set.
Output parameters	Parameter name: pivalue — Description: Returned mark value
Return value	If it is successful, return value is 0, if not, please refer to error

	codes.
Example	/
Details	To do "and" operation with AXISSTATUS to decide which errors' WDOG need to be closed.

Instruction 251	**ZAux_BusCmd_SDOWrite**
Original Format	int32 __stdcall ZAux_BusCmd_SDOWrite(ZMC_HANDLE handle, uint32 slot, uint32 node, uint32 index, uint32 subindex, uint32 type, int value);
Description	Write SDO through device No. and slot No.
Input parameters	Parameter name: Description handle: Link handle slot: Slot No. node: Node No. index: Object dictionary No. subindex: Object dictionary sub No. type: Data type: 1 boolean 2 integer 8 3 integer 16 4 integer 32 5 unsigned 8 6 unsigned 16 7 unsigned 32 value: Written data value
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Max torque of SDO writing and reading.
Details	Before execution, it needs to connect to devices well and scan the bus. It only can write the data dictionary that can be written.

Instruction 252	**ZAux_BusCmd_SDORead**
Original Format	int32 __stdcall ZAux_BusCmd_SDORead(ZMC_HANDLE handle, uint32 slot, uint32 node, uint32 index, uint32 subindex, uint32 type, int *value);
Description	Read SDO through device No. and slot No.
Input parameters	Parameter name: Description handle: Link handle slot: Slot No. node: Node No.

<table>
<tr><td></td><td>
<table>
<tr><td>index</td><td>Object dictionary No.</td></tr>
<tr><td>subindex</td><td>Object dictionary sub No.</td></tr>
<tr><td>type</td><td>Data type:
<table>
<tr><td>1</td><td>boolean</td></tr>
<tr><td>2</td><td>integer 8</td></tr>
<tr><td>3</td><td>integer 16</td></tr>
<tr><td>4</td><td>integer 32</td></tr>
<tr><td>5</td><td>unsigned 8</td></tr>
<tr><td>6</td><td>unsigned 16</td></tr>
<tr><td>7</td><td>unsigned 32</td></tr>
</table></td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>value</td><td>data value that is read.</td></tr>
</table>
</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Max torque of SDO writing and reading.</td></tr>
<tr><td>Details</td><td>Before execution, it needs to connect to devices well and scan the bus.</td></tr>
</table>

<table>
<tr><td>Instruction 253</td><td>ZAux_BusCmd_RtexWrite</td></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_BusCmd_RtexWrite(ZMC_HANDLE handle, uint32 iaxis, uint32 ipara, float value);</td></tr>
<tr><td>Description</td><td>Write RTEX parameter information.</td></tr>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link handle</td></tr>
<tr><td>iaxis</td><td>Axis No.</td></tr>
<tr><td>ipara</td><td>Servo parameter No.
<table>
<tr><td>Value</td><td>Description</td></tr>
<tr><td>1</td><td>Parameter type * 256 + parameter No. (Pr7.20 = 7 * 256 + 20)</td></tr>
<tr><td>2</td><td>Special parameter = 128, current parameter is written into EEPROM (current value = 1)</td></tr>
<tr><td>3</td><td>Special parameter = 4000h + homing clamp mode + (set value * 256), use the homing clamp function of the driver.</td></tr>
</table></td></tr>
<tr><td>value</td><td>Parameter values</td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error</td></tr>
</table>

codes.

Example

RTEX parameters related information.

Details

1. Servo Parameters

Parameter	Function	Value
Pr0.00	Set motor rotation direction	0: CW 1: CCW
Pr0.01	Set control mode.	Generally set as 0: half-closed loop control
Pr0.08	The number of pulses for motor in one cycle.	0-8388608 (according to actual motor)
Pr0.09	gear ratio numerator setting	0-1073741824
Pr0.10	Set gear ratio denominator.	1-1073741824
Pr4.01	Set positive position limit signal	OFF commonly: 00818181h (8487297) ON commonly: 00010101h (65793)
Pr4.02	Set negative position limit signal	OFF commonly: 00828282h (8553090) ON commonly: 00020202h (131586)
Pr4.03	Set Home signal	OFF commonly: 00A2A2A2h (10658466) ON commonly: 00222222h (2236962)
Torque related	/	/
Pr0.13	First torque limit	0-500%
Pr5.21	Select torque limit, when in torque control, it is fixed as Pr0.13 (first torque limit)	As shown in below.

Set value	TL_SW=0		TL_SW=1	
	-	+	-	+
0，[1]	Pr0.13			
2	Pr5.22	Pr0.13	Pr5.22	Pr0.13
3	Pr0.13		Pr5.22	
4	Pr5.22	Pr0.13	Pr5.22	Pr5.25

Pr5.22	Second torque limit	0-500%

Pr5.25	Forward torque limit	0-500%
Pr5.26	Inverse torque limit	0-500%
Speed related	/	/
Pr3.12	Set acceleration time	0~10000ms (reach 1000.r/min)
Pr3.13	Set deceleration time	
Pr3.14	Set S acceleration and deceleration	0~1000ms
Pr3.17	Select speed limit. Mode selection of speed limit value during torque control.	Set value / SL_SW: 0 → Pr3.21; 1 → Pr3.21 / Pr3.22. When it is set to 1, select according to RTEX communication command SL_SW.
Pr3.21	Speed limit value 1	0~20000r/min
Pr3.22	Speed limit value 2	0~20000r/min

2. RTEX Communication Parameter

Pr7.20	Rtex communication period	-1: take effect Pr7.91 setting. 3: 0.5ms 6: 1.0ms
Pr7.21	Rtex command update period ratio	1: 1 time 2: 2 times
Pr7.91	Rtex communication period expansion	62500 ns 125000 ns 250000 ns 500000 ns 1000000 ns 2000000 ns

3. Homing clamp mode

typecode	Description
$50	Position clamp state monitor.
$51	Position clamp 1 is ON.
$52	Position clamp 2 is ON.
$53	Position clamp 1 and 2 are ON.
$54	Position clamp 1 is OFF.
$58	Position clamp 2 is OFF.
$5c	Position clamp 1 and 2 are OFF.

4. Set value

Set value = clamp 1 setting + ($10 * clamp 2 setting)

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0
LATCH_SEL2				LATCH_SEL1			

Clamp 1 / 2 setting	Description
$0	Phase Z signal trigger
$1	Logic rising edge of EXT1
$2	Logic rising edge of EXT2
$3	Logic rising edge of EXT3
$9	Logic falling edge of EXT1
$10	Logic falling edge of EXT2
$11	Logic falling edge of EXT3

Instruction 254	**ZAux_BusCmd_RtexRead**
Original Format	int32 __stdcall ZAux_BusCmd_RtexRead(ZMC_HANDLE handle, uint32 iaxis, uint32 ipara, float *value);
Description	Read RTEX parameter information.
Input parameters	Parameter name: handle — Link handle Parameter name: iaxis — Axis No. Parameter name: ipara — Servo parameter No.

Value	Description
1	Parameter type * 256 + parameter No. (Pr7.20 = 7 * 256 + 20)

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>
<table>
<tr><td>2</td><td>Special parameter = 128, current parameter is written into EEPROM (current value = 1)</td></tr>
<tr><td>3</td><td>Special parameter = 4000h + homing clamp mode + (set value * 256), use the homing clamp function of the driver.</td></tr>
</table>
</td></tr>
<tr><td>value</td><td>Parameter values</td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>value</td><td>Data that is read.</td></tr>
</table>
</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>RTEX parameters related information.</td></tr>
</table>

Details

1. Servo Parameters

Parameter	Function	Value
Pr0.00	Set motor rotation direction	0: CW 1: CCW
Pr0.01	Set control mode.	Generally set as 0: half-closed loop control
Pr0.08	The number of pulses for motor in one cycle.	0-8388608 (according to actual motor)
Pr0.09	gear ratio numerator setting	0-1073741824
Pr0.10	Set gear ratio denominator.	1-1073741824
Pr4.01	Set positive position limit signal	OFF commonly: 00818181h (8487297) ON commonly: 00010101h (65793)
Pr4.02	Set negative position limit signal	OFF commonly: 00828282h (8553090) ON commonly: 00020202h (131586)
Pr4.03	Set Home signal	OFF commonly: 00A2A2A2h (10658466) ON commonly: 00222222h (2236962)
Torque related	/	/
Pr0.13	First torque limit	0-500%
Pr5.21	Select torque limit, when in torque control, it is fixed as Pr0.13 (first torque limit)	As shown in below.

Set value	TL_SW=0		TL_SW=1	
	-	+	-	+
0，[1]	Pr0.13			
2	Pr5.22	Pr0.13	Pr5.22	Pr0.13
3	Pr0.13		Pr5.22	
4	Pr5.22	Pr0.13	Pr5.22	Pr5.25

Pr5.22	Second torque limit	0-500%
Pr5.25	Forward torque limit	0-500%
Pr5.26	Inverse torque limit	0-500%

<table>
<tr><th>Speed related</th><th>/</th><th>/</th></tr>
<tr><td>Pr3.12</td><td>Set acceleration time</td><td rowspan="2">0~10000ms (reach 1000.r/min)</td></tr>
<tr><td>Pr3.13</td><td>Set deceleration time</td></tr>
<tr><td>Pr3.14</td><td>Set S acceleration and deceleration</td><td>0~1000ms</td></tr>
<tr><td>Pr3.17</td><td>Select speed limit. Mode selection of speed limit value during torque control.</td><td>
<table>
<tr><td rowspan="2">Set value</td><td colspan="2">SL_SW</td></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>0</td><td colspan="2">Pr3.21</td></tr>
<tr><td>1</td><td>Pr3.21</td><td>Pr3.22</td></tr>
</table>
When it is set to 1, select according to RTEX communication command SL_SW.
</td></tr>
<tr><td>Pr3.21</td><td>Speed limit value 1</td><td>0~20000r/min</td></tr>
<tr><td>Pr3.22</td><td>Speed limit value 2</td><td>0~20000r/min</td></tr>
</table>

2. RTEX Communication Parameter

Pr7.20	Rtex communication period	-1: take effect Pr7.91 setting. 3: 0.5ms 6: 1.0ms
Pr7.21	Rtex command update period ratio	1: 1 time 2: 2 times
Pr7.91	Rtex communication period expansion	62500 ns 125000 ns 250000 ns 500000 ns 1000000 ns 2000000 ns

3. Homing clamp mode

typecode	Description
$50	Position clamp state monitor.
$51	Position clamp 1 is ON.
$52	Position clamp 2 is ON.
$53	Position clamp 1 and 2 are ON.
$54	Position clamp 1 is OFF.
$58	Position clamp 2 is OFF.
$5c	Position clamp 1 and 2 are OFF.

4. Set value

Set value = clamp 1 setting + ($10 * clamp 2 setting)

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0

LATCH_SEL2	LATCH_SEL1

Clamp 1 / 2 setting	Description
$0	Phase Z signal trigger
$1	Logic rising edge of EXT1
$2	Logic rising edge of EXT2
$3	Logic rising edge of EXT3
$9	Logic falling edge of EXT1
$10	Logic falling edge of EXT2
$11	Logic falling edge of EXT3

Instruction 255	**ZAux_BusCmd_SDOReadAxis**
Original Format	int32 __stdcall ZAux_BusCmd_SDOReadAxis (ZMC_HANDLE handle, uint32 iaxis,uint32 index,uint32 subindex ,uint32 type ,int32 * value)
Description	Read SDO through axis No.
Input parameters	Parameter name: handle — Link handle slot — Slot No. index — Object dictionary No. subindex — Object dictionary sub No. type — Data type: 1 boolean

	2 integer 8 3 integer 16 4 integer 32 5 unsigned 8 6 unsigned 16 7 unsigned 32
Output parameters	Parameter name: value — Description: data value that is read.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Int32 iValue; ZAux_BusCmd_SDOReadAxis(0,0x6061,0,2,&iValue); //get axis 0's control mode.
Details	Before execution, it needs to connect to devices well and scan the bus. Only readable data dictionaries can be read.

Instruction 256	**ZAux_BusCmd_SDOWriteAxis**
Original Format	int32 __stdcall ZAux_BusCmd_SDOWriteAxis (ZMC_HANDLE handle, uint32 iaxis, uint32 index,uint32 subindex ,uint32 type ,int32 value)
Description	Write SDO through axis No.
Input parameters	Parameter name — Description: handle — Link handle iaxis — Axis No. index — Object dictionary No. subindex — Object dictionary sub No. type — Data type: 1 boolean 2 integer 8 3 integer 16 4 integer 32 5 unsigned 8 6 unsigned 16 7 unsigned 32 value — data value that is written.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	ZAux_BusCmd_SDOWriteAxis(0,0x6060,0,2,8); //axis 0 control mode is 8, position control

Details	Before execution, it needs to connect to devices well and scan the bus. Only readable data dictionaries can be read.

Instruction 257	**ZAux_BusCmd_NodePdoWrite**
Original Format	int32 __stdcall ZAux_BusCmd_NodePdoWrite (ZMC_HANDLE handle, uint32 inode,uint32 index,uint32 subindex ,uint32 type ,int value)
Description	PDO writing operation.
Input parameters	Parameter name: Description handle: Link handle inode: Device No. index: Object dictionary No. subindex: Object dictionary sub No. type: Data type: 1 boolean 2 integer 8 3 integer 16 4 integer 32 5 unsigned 8 6 unsigned 16 7 unsigned 32 value: data value that is written.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	ZAux_BusCmd_NodePdoWrite(0,0x6040,0,3,15); //Set device 0 control word to 15
Details	For non-axis and IO devices, use this command to read and write PDO, such as power supply devices. For axis and IO type devices, they can access PDO through axis parameters and IO instructions, but this instruction cannot be used. When the bus is turned on, the current PDO list and the current value of the writable PDO will be automatically read in advance. The PDO list or the current value of the related data dictionary can be modified through SDO before calling bus open. It needs to be started before it can be modified. You can use SOD_START to start to SAFEOP first, then set the initial PDO state, and then start to OP.

Instruction 258	**ZAux_BusCmd_NodePdoRead**

<table>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_BusCmd_NodePdoRead(ZMC_HANDLE handle,uint32 inode,uint32 index,uint32 subindex ,uint32 type ,int* ivalue)</td></tr>
<tr><td>Description</td><td>PDO reading operation.</td></tr>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link handle</td></tr>
<tr><td>inode</td><td>Device No.</td></tr>
<tr><td>index</td><td>Object dictionary No.</td></tr>
<tr><td>subindex</td><td>Object dictionary sub No.</td></tr>
<tr><td>type</td><td>Data type:
<table>
<tr><td>1</td><td>boolean</td></tr>
<tr><td>2</td><td>integer 8</td></tr>
<tr><td>3</td><td>integer 16</td></tr>
<tr><td>4</td><td>integer 32</td></tr>
<tr><td>5</td><td>unsigned 8</td></tr>
<tr><td>6</td><td>unsigned 16</td></tr>
<tr><td>7</td><td>unsigned 32</td></tr>
</table></td></tr>
<tr><td>value</td><td>data value that is written.</td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>value</td><td>Read data value.</td></tr>
</table>
</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Int32 iValue;
ZAux_BusCmd_NodePdoRead(0,0x6041,0,3,&iValue);
//get device 0's state word</td></tr>
<tr><td>Details</td><td>For non-axis and IO devices, use this command to read and write PDO, such as power supply devices.
For axis and IO type devices, they can access PDO through axis parameters and IO instructions, but this instruction cannot be used.</td></tr>
</table>

<table>
<tr><td>Instruction 259</td><td>ZAux_BusCmd_GetDriveTorque</td></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_BusCmd_GetDriveTorque (ZMC_HANDLE handle , int iaxis,int *piValue);</td></tr>
<tr><td>Description</td><td>Read current bus torque, it needs to set corresponding DRIVE_PROFILE type.</td></tr>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link handle</td></tr>
<tr><td>iaxis</td><td>Axis No.</td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>piValue</td><td>Current torque.</td></tr>
</table>
</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error</td></tr>
</table>

	codes.
Example	Get current torque of current bus drive
Details	The correct ATYPE (set to 65/66/67) and DRIVE_PROFILE must be set before reading.

Instruction 260	**ZAux_BusCmd_SetMaxDriveTorque**
Original Format	int32 __stdcall ZAux_BusCmd_SetMaxDriveTorque (ZMC_HANDLE handle, int iaxis , int piValue);
Description	Set max torque of current bus drive, it needs to set corresponding DRIVE_PROFILE type.
Input parameters	Parameter name: handle — Link handle; iaxis — Axis No.; fValue — Set max torque limit
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	The correct ATYPE (set to 65/66/67) and DRIVE_PROFILE must be set before reading.

Instruction 261	**ZAux_BusCmd_GetMaxDriveTorque**
Original Format	int32 __stdcall ZAux_BusCmd_GetMaxDriveTorque (ZMC_HANDLE handle, int iaxis , int *piValue);
Description	Get max torque of current bus torque, it needs to set corresponding DRIVE_PROFILE type.
Input parameters	Parameter name: handle — Link handle; iaxis — Axis No.
Output parameters	Parameter name: piValue — Obtained max torque.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Get current torque of current bus drive
Details	The correct ATYPE (set to 65/66/67) and DRIVE_PROFILE must be set before reading.

Instruction 262	**ZAux_Direct_SetDAC**
Original Format	int32 __stdcall ZAux_Direct_SetDAC (ZMC_HANDLE handle, int iaxis , float fValue);
Description	The servo axis DA directly controls the torque. ps: The bus driver

	needs to set corresponding DRIVE_PROFILE type, and ATYPE needs to be set to speed mode.
Input parameters	Parameter name: Description handle: Link handle iaxis: Axis No. fValue: Set analog output value.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	The unit is the scale of the DA module, 12 or 16 bits. The speed control depends on the specific unit of the drive. The unit of torque control is one thousandth, equal to 1000 means 100% torque.

Instruction 263	**ZAux_Direct_GetDAC**
Original Format	int32 __stdcall ZAux_Direct_GetDAC (ZMC_HANDLE handle, int iaxis , float *fValue);
Description	The servo axis DA directly controls the torque. ps: The bus driver needs to set corresponding DRIVE_PROFILE type, and ATYPE needs to be set to speed mode.
Input parameters	Parameter name: Description handle: Link handle iaxis: Axis No. fValue: Set analog output value.
Output parameters	Parameter name: Description fValue: Analog returned value.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	The unit is the scale of the DA module, 12 or 16 bits. The speed control depends on the specific unit of the drive. The unit of torque control is one thousandth, equal to 1000 means 100% torque.

Instruction 264	**ZAux_Execute**
Original Format	int32 __stdcall ZAux_Execute(ZMC_HANDLE handle, const char *pszCommand, char *psResponse, uint32 uiResponseLength)
Description	Send character string command to controller, in buffer method (it automatically blocks when there is no buffer for controller).
Input	Parameter name: Description

parameters	handle: Link handle pszCommand: Sent command character string. uiResponseLength: Returned character string length.
Output parameters	Parameter name: Description psResponse: Returned character string.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Use online command function
Details	Upper computer uses unencapsulated Basic instruction function.

Instruction 265	**ZAux_DirectCommand**
Original Format	int32 __stdcall ZAux_DirectCommand(ZMC_HANDLE handle, const char *pszCommand,char *psResponse, uint32 uiResponseLength)
Description	Send character string command to controller, in direct method (it doesn't enter buffer area, there are few commands don't support this currently).
Input parameters	Parameter name: Description handle: Link handle pszCommand: Sent command character string. uiResponseLength: Returned character string length.
Output parameters	Parameter name: Description psResponse: Returned character string.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Use online command function
Details	Upper computer uses unencapsulated Basic instruction function.

Instruction 266	**ZAux_Pause**
Original Format	int32 __stdcall ZAux_Pause(ZMC_HANDLE handle)
Description	Pause BASIC program that is running inside the controller.
Input parameters	Parameter name: Description handle: Link handle
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	When the task is resumed after pausing, task continues to

	executing

Instruction 267	ZAux_Resume
Original Format	int32 __stdcall ZAux_Resume(ZMC_HANDLE handle)
Description	Continue to run controller internal BASIC program.
Input parameters	Parameter name: handle — Description: Link handle
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	When the task is resumed after pausing, task continues to executing

Instruction 268	ZAux_BasDown
Original Format	int32 __stdcall ZAux_BasDown(ZMC_HANDLE handle,const char *Filename,uint32 run_mode)
Description	One single .bas file generates ZAR and is downloaded into controller for operation.
Input parameters	handle — Link handle Filename — BAS file name that is with path. run_mode — Download mode, 0-RAM, 1-ROM.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Download & edit controller program
Details	.bas file, it can directly run Zmotion motion control card's Basic file.

Instruction 269	ZAux_ZarDown
Original Format	int32 __stdcall ZAux_ZarDown(ZMC_HANDLE handle,const char *Filename,uint32 run_mode)
Description	Download ZAR program into controller for operation.
Input parameters	handle — Link handle Filename — BAS file name that is with path. run_mode — Download mode, 0-RAM, 1-ROM.

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Download & edit controller program
Details	Zar file, it can directly run Zmotion motion control card program. It is the file that is generated after encryption.

Instruction 270	**ZAux_Direct_GetVariableInt**
Original Format	int32 __stdcall ZAux_Direct_GetVariableInt (ZMC_HANDLE handle , const char * pname, int * piValue);
Description	Read Basic integer global variables, or axis parameters.
Input parameters	Parameter name: handle — Description: Link handle Parameter name: pname — Description: Global variables name / axis parameter name with custom axis No. DPOS(0).
Output parameters	Parameter name: piValue — Description: Returned value.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	int piValue; ZAux_Direct_GetVariableInt (handle , "DPOS（0）", &piValue); //get axis 0 DPOS value in integer type
Details	Global variable refers to the name of the global variable that needs to be set in the ZDevelop software, and then set and read the value of the global variable through the host computer. Does not support setting and reading of Basic structures.

Instruction 271	**ZAux_Direct_GetVariablef**
Original Format	int32 __stdcall ZAux_Direct_GetVariablef (ZMC_HANDLE handle , const char * pname, float* pfValue);
Description	Read Basic floating global variables, or axis parameters.
Input parameters	Parameter name: handle — Description: Link handle Parameter name: pname — Description: Global variables name / axis parameter name with custom axis No. DPOS(0).
Output parameters	Parameter name: pfValue — Description: Returned value.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Float pfValue;

	ZAux_Direct_GetVariablef (handle , "DPOS（0）", &pfValue); // get axis 0 DPOS value in float type
Details	Global variable refers to the name of the global variable that needs to be set in the ZDevelop software, and then set and read the value of the global variable through the host computer. Does not support setting and reading of Basic structures.

Instruction 272	**ZAux_Direct_SetUserArray**
Original Format	int32 __stdcall ZAux_Direct_SetUserArray(ZMC_HANDLE handle, char* arrayname, int arraystart, int numes, float * pfValue);
Description	Set Basic global array.
Input parameters	Parameter name: Description handle: Link handle arrayname: Array name. arraystart: Array starting element. numes: The number of arrays pfValue: The value that is set.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	Global variable refers to the name of the global variable that needs to be set in the ZDevelop software, and then set and read the value of the global variable through the host computer. Does not support setting and reading of Basic structures.

Instruction 273	**ZAux_Direct_GetUserArray**
Original Format	int32 __stdcall ZAux_Direct_GetUserArray(ZMC_HANDLE handle, char* arrayname, int arraystart, int numes, float *pfValue);
Description	Read Basic global array.
Input parameters	Parameter name: Description handle: Link handle arrayname: Array name. arraystart: Array starting element. numes: The number of arrays
Output parameters	Parameter name: Description pfValue: Read array element's values.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	Global variable refers to the name of the global variable that needs

	to be set in the ZDevelop software, and then set and read the value of the global variable through the host computer. Does not support setting and reading of Basic structures.

Instruction 274	ZAux_Direct_SetUserVar
Original Format	int32 __stdcall ZAux_Direct_SetUserVar(ZMC_HANDLE handle,char *varname , float pfValue);
Description	Set Basic custom global variables.
Input parameters	Parameter name: Description handle: Link handle varname: Variables' name. pfValue: The value that is set.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Download & edit controller program
Details	Custom global variable refers to the name of the global variable that needs to be set in the ZDevelop software, and then set and read the value of the global variable through the host computer. Does not support setting and reading of Basic structures.

Instruction 275	ZAux_Direct_GetUserVar
Original Format	int32 __stdcall ZAux_Direct_GetUserVar(ZMC_HANDLE handle,char *varname , float *pfValue);
Description	Read Basic custom global variables.
Input parameters	Parameter name: Description Handle: Link handle varname: Variables' name.
Output parameters	Parameter name: Description pfValue: Returned value
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Download & edit controller program
Details	Custom global variable refers to the name of the global variable that needs to be set in the ZDevelop software, and then set and read the value of the global variable through the host computer. Does not support setting and reading of Basic structures.

Instruction 276	ZAux_SetTimeOut
Original Format	int32 __stdcall ZAux_SetTimeOut(ZMC_HANDLE handle, uint32 timems)

Description	Commanded delay waiting time.
Input parameters	Parameter name: handle — Description: Link handle Parameter name: timems — Description: Waiting time MS.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	/

Instruction 277	**ZAux_TransStringtoInt**
Original Format	int32 __stdcall ZAux_TransStringtoInt (const char * pstringin, int inues, int * pivlaue);
Description	Convert character string into int.
Input parameters	Parameter name: pstringin — Description: Character string of data Parameter name: inumes — Description: The number of data converting.
Output parameters	Parameter name: pivalue — Description: Converted data
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	/

Instruction 278	**ZAux_TransStringtoFloat**
Original Format	int32 __stdcall ZAux_TransStringtoFloat (const char * pstringin, int inues, int * pfvlaue);
Description	Convert character string into float.
Input parameters	Parameter name: pstringin — Description: Character string of data Parameter name: inumes — Description: The number of data converting.
Output parameters	Parameter name: pfvalue — Description: Converted data
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	/

Instruction 279	***PZMCAutoUpCallBack**

Original Format	typedef void (*PZMCAutoUpCallBack)(ZMC_HANDLE handle, int32 itypecode, int32 idatalength, uint8 *pdata);
Description	Call back function format of active reporting.
Input parameters	Parameter name: Description handle: Link handle itypecode: Upload type code idatalength: Data length pdata: Data pointer
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Active reporting.
Details	/

Instruction 280	**ZAux_SetAutoUpCallBack**
Original Format	int32 __stdcall ZAux_SetAutoUpCallBack (ZMC_HANDLE handle, PZMCAutoUpCallBack pcallback);
Description	Controller reports automatically.
Input parameters	Parameter name: Description handle: Link handle pcallback: Function format
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Automatically report
Details	Report automatically: Controller auto-report function. Automatically report messages: controller sends messages to PC.

Instruction 281	**ZAux_CycleUpEnable**
Original Format	int32 __stdcall ZAux_CycleUpEnable(ZMC_HANDLE handle, uint32 cycleindex, float fintervalms, const char* psetesname);
Description	Enable cyclic reporting.
Input parameters	Parameter name: Description handle: Link handle cycleindex: Report channel No., 0-max value (PS: ECI series max values = 0, ZMC4XX series max values = 1) fintervalms: Reporting gap time, the unit is ms. And it can not lower than controller bus

		communication period.
	psetesname	Select parameters to reporting, grammar: Parameter 1: for example: IDLE: report IDLE of default axis. Parameter 2(index): for example: IDLE(1): report IDLE of axis 1. Parameter 3(index,numes): for example: IDLE(0,3): report IDLE of axis 0-2.

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Periodically report
Detail 1	(1) The number of bytes uploaded by each channel cannot exceed 1000 bytes. (2) The space occupied by each parameter is different. One IO only occupies one bit. Products above 4 series use the double data type by default, and 3 series and XPLC products use the float data type by default. (3) Only the network port connection can be uploaded periodically (4) It needs zmotion.dll and controller firmware supports. (5) By setting the parameters that are frequently read in advance to report actively and periodically, it can reduce the time for active polling by the PC
Detail 2	Report support parameters: VP_SPEED (command current speed), MPOS (feedback / measurement position), DPOS (command / demand position), MSPEED (feedback current speed), AXISSTATUS (axis status), FE (following error), MOVES_BUFFERED (current buffer number), IDLE (motion state), MOVE_CURMARK (current movement mark), MARK (latch trigger), VECTOR_BUFFERED(), MTYPE (current motion type), DRIVER_FE (driver's current follow-up error), DRIVER_TORCH (driver current torque), DRIVER_STATUS (driver status), IN (input), OUT (output), AIN (analog input), AOUT (analog output), TABLE(register). The 4 series controller adds parameters: REMAIN_BUFFER1, the remaining number of linear motion buffers, in the case of MTYPE=1.

Instruction 282	**ZAux_CycleUpDisable**
Original Format	int32 __stdcall ZAux_CycleUpDisable(ZMC_HANDLE handle, uint32 cycleindex);
Description	Close periodic-reporting enable.
Input parameters	Parameter name | Description

	handle	Link handle
	cycleindex	Report channel No., 0-max value-1
Output parameters	/	
Return value	If it is successful, return value is 0, if not, please refer to error codes.	
Example	Periodically report	
Detail	(1) It needs zmotion.dll and controller firmware supports. (2) By setting the parameters that are frequently read in advance to report actively and periodically, it can reduce the time for active polling by the PC	

Instruction 283	**ZAux_CycleUpGetRecvTimes**	
Original Format	uint32 __stdcall ZAux_CycleUpGetRecvTimes(ZMC_HANDLE handle, uint32 cycleindex);	
Description	Obtain the number of packages that is received by periodical reporting.	
Input parameters	Parameter name	Description
	handle	Link handle
	cycleindex	Report channel No., 0-max value-1
Output parameters	/	
Return value	The number of packages that is received by periodical reporting	
Example	Periodically report	
Detail	It needs zmotion.dll and controller firmware supports. Exceed overflow, for debugging.	

Instruction 284	**ZAux_CycleUpForceOnce**	
Original Format	int32 __stdcall ZAux_CycleUpForceOnce(ZMC_HANDLE handle, uint32 cycleindex);	
Description	Report once compulsively.	
Input parameters	Parameter name	Description
	handle	Link handle
	cycleindex	Report channel No., 0-max value-1
Output parameters	/	
Return value	The number of packages that is received by periodical reporting	
Example	Periodically report	
Detail	It is called when idle may be wrong after the motion command.	

Instruction 285	**ZAux_CycleUpReadBuff**
Original	uint32 __stdcall ZAux_CycleUpReadBuff(ZMC_HANDLE handle,

Format	uint32 cycleindex, const char *psetname, uint32 isetindex, double *pvalue);
Description	Read the content from periodical reporting, and the content data type is floating type.
Input parameters	Parameter name: handle — Link handle cycleindex — Report channel No., 0-max value-1 psetname — Parameter name isetindex — Parameter No.
Output parameters	Parameter name: pvalue — Read parameter content in periodical reporting.
Return value	The number of packages that is received by periodical reporting
Example	Periodically report
Detail	(1) It needs zmotion.dll and controller firmware supports. (2) By setting the parameters that are frequently read in advance to report actively and periodically, it can reduce the time for active polling by the PC

Instruction 286	**ZAux_CycleUpReadBuffInt**
Original Format	uint32 __stdcall ZAux_CycleUpReadBuffInt(ZMC_HANDLE handle, uint32 cycleindex, const char *psetname, uint32 isetindex, int32*pvalue);
Description	Read the content from periodical reporting, and the content data type is integer type.
Input parameters	Parameter name: handle — Link handle cycleindex — Report channel No., 0-max value-1 psetname — Parameter name isetindex — Parameter No.
Output parameters	Parameter name: pvalue — Read parameter content in periodical reporting.
Return value	The number of packages that is received by periodical reporting
Example	Periodically report
Detail	(1) It needs zmotion.dll and controller firmware supports. (2) By setting the parameters that are frequently read in advance to report actively and periodically, it can reduce the time for active polling by the PC

Instruction 287	**ZAux_SetTraceFile**
Original Format	int32 __stdcall ZAux_SetTraceFile(int bifTofile,const char *pFilePathName)
Description	Print out the called PC function library command to the specified

	file.
Input parameters	Parameter name: bifTofile — Description: Print output mode. 0- OFF 1- Only error command is output 2- Only motion and parameter configuration command are output 3- All commands are output Parameter name: pFilePathName — Description: File name is with absolute path. This is reserved parameter, which means it doesn't take effect currently, any character string can be filled in.
Output parameters	/
Return value	The number of packages that is received by periodical reporting
Example	Debug and print information
Detail	The file is saved under \ZmotionLog\ under the current program path, and is saved as a log. Note: This command is only used during debugging. When not in use, remember to turn off the print output mode, otherwise it will affect the communication efficiency.

Instruction 288	**ZAux_Trigger**
Original Format	int32 __stdcall ZAux_Trigger(ZMC_HANDLE handle)
Description	Oscilloscope triggering function (it is valid in firmware version above 150723).
Input parameters	Parameter name: handle — Description: Link handle
Output parameters	/
Return value	The number of packages that is received by periodical reporting
Example	Oscilloscope Usage
Detail	It needs to use ZDevelop software to open oscilloscope.

Instruction 289	**ZAux_Direct_GetCreep**
Original Format	int32 __stdcall ZAux_Direct_GetCreep(ZMC_HANDLE handle, int iaxis,float *pfValue);
Description	Read 2 times homing creep speed.
Input parameters	Parameter name: handle — Description: Link handle Parameter name: iaxis — Description: Axis No.

Output parameters	Parameter name: *piValue — Description: Returned creep speed.
Return value	The number of packages that is received by periodical reporting
Example	Periodically report
Detail	While in homing, it is the creep speed when leave the origin after found the origin.

Instruction 290	**ZAux_Direct_Base**
Original Format	int32 __stdcall ZAux_Direct_Base(ZMC_HANDLE handle, int imaxaxises, int *piAxislist)
Description	Set default axis No. of motion controller.
Input parameters	Parameter name: handle — Description: Link handle Parameter name: imaxaxises — Description: The number of participated axes Parameter name: puAxislist — Description: Axis No. array
Output parameters	/
Return value	The number of packages that is received by periodical reporting
Example	/
Detail	/

Instruction 291	**ZAux_Direct_Backlash**
Original Format	int32 __stdcall ZAux_Direct_Backlash(ZMC_HANDLE handle, int iaxis, bool enable,float dist,float speed,float accel)
Description	Set axis backlash, but it is invalid in expansion axis.
Input parameters	Parameter name: handle — Description: Link handle Parameter name: iaxis — Description: Axis No. that is participated in motion. Parameter name: enable — Description: Enable switch: 0-OFF, 1-ON Parameter name: dist — Description: Compensation distance, unit: units Parameter name: speed — Description: Compensation speed of backflash Parameter name: accel — Description: Compensation acceleration of backflash
Output parameters	/
Return value	The number of packages that is received by periodical reporting
Example	Backflash Compensation
Detail	/

Chapter XIII Instruction Returned Value

13.1. Instruction Returned Value

User can send the command through calling function that is encapsulated by dynamical library in program, then, motion controller will move according to received motion commands.

After the instruction is accepted by motion controller, return instruction execution result for the host computer that sends the command to indicate whether the instruction is executed correctly.

13.2. Return Values Details (Error Codes)

Error Code	Meaning	Example
210	File is too large	
212	State error	It is non-suspended state in Resume
213	File downloading and uploading errors, and package loss	When PC function is called, return this error.
214	The length of downloaded file is checked that is wrong.	
215	Buffer length or storage is not enough	When the character string command is over long, return this error.
217	Functions are not supported or prohibited by controllers.	
218	Called and transferred parameters are wrong.	
219	Download appears conflict, multiple files are downloaded simultaneously.	
220	Filename error, there are special characters.	
221	Filename error, more than the length.	
222	File does not exist	
223	Password protection limits.	
224	Password protection limits 2.	
260	Hardware error	
261	Disk is not formatted	
262	RTC error	
263	NORFLASH error	

264	RAM error	
265	NANDFLASH error	
266	USB Drive error	
267	FPGA error	
268	Ethernet hardware error	
271	Backup power supply error	
272	Sub-card does not exist	
273	File is missing	
274	System File Error	
275	No master, generating from sub-card	
276	Program file correction error	
277	Program file error causes it can't start	
278	ZAR check, APPPASS error	
279	ZAR check, ID error	
280	BAS file exceeds the maximum number	
281	Sub-card ID conflicts, or multi-main conflict	
282	Unsupported functions	
284	The controller does not match with zar	
285	Image File Error	
286	Font file error	
288	Controller appears above errors, it will cause alarm when ON next time.	
1000	Motion module returns wrong offset	
1002	No motion buffer	
1004	Slave axis is moving	
1005	Motion functions that are not supported	
1006	Arc position error	
1007	Parameter ellipse AB errors	
1008	The motion module input parameter error	
1009	It is in motion process, which means it can't be operated	
1010	Pause and others are operated repeatedly	
1011	IDLE can't do pause, this kind of operation.	
1012	The current movement doesn't support pause	
1013	Pause point can't be found	
1014	ATYPE are not supported	
1015	ATYPE of ZCAN conflicts	
1016	Functions are not supported by axis	
1017	FRAME correction data error	
1018	FRAME correction data is not enough	
1019	Data meeting conditions in FRAME correction data is not enough	
1020	FRAME auxiliary parameter correction data is not enough	

1021	The interval of FRAME correction data is too small, smaller than the number of joint axes	
1022	FRAME coordinate error	
1023	Coordinate can't be modified compulsively under FRAME status	
1024	FRAME abnormal inverse solution	
1025	Not FRAME status	
1026	FRAME HAND error	
1027	Attitude can't be switched in interpolation	
1028	Axis equivalent of special joint axis and virtual axis are the same.	
1030	CORNERMODE 7-bit is set well but it does not support this motion	
1031	CORNERMODE 7-bit is set well but it is not in FRAME status	
1032	AXIS_ADDRESS error	
2000	ZBASIC module offset.	
2021	Manually stop	
2022	This task stops caused by errors in other tasks	
2023	Attempt to modify as read-only parameter	
2024	Array is over bound	
2025	The number of variables is more than controller specifications	
2026	The number of arrays is more than controller specifications	
2027	The number array space is more than controller specifications	
2028	The number of SUB is more than controller specifications	
2029	Identifier naming error	
2030	Identifier name is too long	
2031	No right parenthesis	
2032	Unknown characters	
2033	Expressions encountered unknown name	
2034	SUB can't be used in expression	
2043	Unknown command identifier, and it is the first identifier name of current line.	
2044	Stack Overflow	
2045	Math expression is too complex, different controllers have different specifications	
2046	End reference numerals are not found	
2047	Command does not have return value, not be used for expression calculation.	
2048	Function must return a value, no need at the beginning of the line.	

2049	Special instruction must be in a separate line	
2050	Parameters or arrays need to be indexed.	
2051	Variables can not use index	
2052	Redefine array, and the length is inconsistent.	
2053	The "length" parameter of array definition is wrong, it is negative or too large.	
2054	Identifiers have been defined as the SUB process, they can not do others	SUB process mark symbol can't be defined again.
2055	Identifiers have been defined as parameters, can not do others.	
2056	Identifiers reserved and can not be used.	
2057	Unrecognizable character appears	Such as, "&" can't be identified
2058	SUB calling is out stake repeatedly	
2060	Syntax format error	There is no TO after FOR
2062	Function parameters range Error	When the task No. exceeds the range, it will return this error. Auto run task No. error is the error code.
2063	Too many function parameters	Like, CONNECT(1,1,1)
2064	Too less function parameters	Like, CONNECT(1)
2065	Lack operands	
2066	Lack operands behind operators	
2067	Lack operands before operators	
2068	Unknown operators	
2069	Lack binary operators	An operator is needed between two instructions that are in same line.
2070	CALL must call SUB	
2072	Require assignment symbol	It will report error when use space between data instead of comma.
2073	Empty file	
2074	SUB-defined identifier name conflicts.	
2075	Task to be opened has been running.	
2076	Multiple parameters are separated by commas.	
2077	Parentheses are not matched, no left parenthesis.	
2078	"IF" nesting too much	
2079	"Loops" nested too much.	
2080	The number of interpolation axes is too less	
2081	CONST constant can not be modified.	It will report error when defined constant data is assigned again.
2082	Command can not be sent from the PC	

	online.	
2083	Too many SUB-defined parameters.	
2084	SUB with parameters, they can not be used GOTO statement.	
2085	Local identifier defined too much	
2086	LOCAL variable names and variable names or other identifiers file name conflict.	
2087	LOCAL does not support the array definition.	
2088	GSUB defined parameters letters repeated.	
2089	GSUB defined parameters can only be a single letter.	
2090	Modify read-only parameter is not allowed	
2091	GSUB_IFPARA function uses the wrong occasion.	
2092	Division by zero	
2093	Over buffer	
2094	Command-line blocking time is too long.	
2095	Parameter same name	
2096	Values are used without initialization	
2097	Axis No. conflicts	
2099	internal error	
2100	The number of SCANEDGE is too many	
2101	ZINDEX type mismatch	
2901	System errors, defined identifiers are too many, including variables, arrays, process, process parameters and so on.	
3201	Over buffer	
3202	File ends abnormally	
3204	Internal state error	
3205	Unsupported functions	
3206	Internal call parameters error	
3231	Resource is not enough	
3242	“os” error	
3243	U disk is not inserted	
3244	File is opened repeatedly	
3245	File is oversize	
3248	Filename error	
3249	Filename is over long	
3250	File doesn't exist	
3301	Three points in a circular arc line.	
3302	Two parallel straight lines, there is no intersection.	
3401	MODBUS master station parameter is wrong, and the normal length is over	
3402	Message Response Timeout	

3407	MODBUS return parameter error	
3408	MODBUS return is not supported	
3421	MODBUS slave station returns invalid function codes	
3422	MODBUS slave station returns address space error	
3423	MODBUS slave station returned wrong length data	
3424	MODBUS salve station returned length is too long	
3501	ZCAN return without sub card	
3502	No relative axis when ZCAN returns sub card	
Error: 4000-4500 PLC module		
4002	Parameter error	
4003	Unknown type	
4004	Unknown function	
4005	Push too many STL	
4006	Push too many stakes	
4007	The program is too complex, and BLOCK is too many	
4008	No push BLOCK	
4009	No push STL	
4010	No push	
4014	File content errors	
4015	RET must be behind STL	
4016	Over range	
4017	Below range	
4018	L is not defined	
4019	Not support G code function	
4020	GOTO can not cross PLC and BASIC	
4021	PLC is only one main task	
4022	Grammatical errors	
4023	FOR NEXT errors, mismatched	
4024	FOR NEXT error, no NEXT	
4026	FOR MC mix	
4027	FOR STL mix	
4030	Must be used in PLC main task	
4031	Must be used in interrupt	
4032	The number of parameters is less	
4033	The number of parameters is more	
4034	To be a multiple of 8	
4035	The register indicates an error	
4036	Register type error	
4037	More than the number of LVs	
4038	Read-only	
4500-5000 PLC PC side error		
4503	Not enough memory	

4504	Return to the bus cable	
4505	Reflux	
4506	AND type can not be directly connected to the bus cable	
4510	Dangling	
4511	The far right must be the type of output	
Error: 5000-5500 HMI module		
5000	No. of LCD is wrong	
5002	LCD No. conflicts	
5003	Object that is not supported	
5004	Memory is not enough	
5005	Control Hierarchy Error	
5006	Window NO. exceeds	
5007	Invalid window number	
5010	Object attribute is missing	
5011	There are several display elements on input window.	
5012	ACTION type error	
5013	Too many events.	
5014	Fail to return to the previous window	
5015	Basic window can't close	
5016	Font is not found in the related character	
5017	It must be used in HMI tasks	
5020	Control ID Conflict	
5021	Error No. LCD	
5022	Available LCD can't be found	
5023	LCD does not open	
5024	LCD has no data	
5025	Program reset	
5026	LCD has been opened	
5027	Not a network LCD	
5028	It does not support compression	
5029	Color depth is not supported	
5030	Unsupported Data Types	
5031	Device No. error	
5032	LCD_SEL can not be used	
5033	REDRAW can't be set in DRAW stage	
5034	DRAW function only can be called in DRAW stage	
5035	Operation can't be called in DRAW stage	
5036	Internal LCD resolution is fixed	
5037	LCD resolution exceeds	
5038	Library filename error	
5039	Too many characters	
5501-5599	PC side PLC file compile errors	
5501-5599: PC-side PLC file compilation error		

5503	Memory is not enough	
5504	Return to bus wire	
5505	Reflux	
5506	AND type instruction can not be directly connected to the bus	
5510	Right is hanging in the air, there is no output instruction	
5511	The far right type of instruction is not output	
5512	The far right can not be connected together	
5513	Output type of instruction must be the rightmost	
5514	It does not support the type of instruction	
5517	No value of register	
5518	DOT value exceeds the range	
5519	Index register exceeds the range	
5520	Excessive number of characters	
5521	Register type error	
5522	Register value error	
5523	Excessive number of registers	
5524	The number of registers is too less	
5525	Use STL wrongly	
5526	Use RET wrongly	
5527	Repeat RET	
5528	Position error of END or LBL	
5529	Function can't be directly connected to the bus	
5530	Out the stake without pushing the stake	
5531	Too many MPP	
5532	Using the wrong type register	
5533	ANB error, insufficient number of blocks	
5534	ORB error, insufficient number of blocks	
5535	ANB error, it can't be combined after output operation	
5536	ORB error, it can't be combined after output operation	
5537	AND directly connects to bus	
5538	OR directly connects to the bus	
5539	OR can't be behind OUT	
5540	STL can not be shared with MC	
5541	MC can not directly access the bus	
5542	_ @ register needs brackets	
5543	Notes error	
5544	Ladder excessive number of columns	
5545	Output types can not be directly connected to bus	
6000: ECAT bus error		

6000	ECAT module error, SLOT No. error	
6001	Internal error, function is not supported.	
6005	Parameter error	
6006	The number of device types that are supported is over the limit	
6009	NODE operation times exceeds	
6012	Insufficient resources	
6013	Slave device response timeout	
6014	Buffer is not enough	
6015	Response packet error WKC	
6016	Long answer of SDO	
6017	SDO wrong response	
6018	SDO response data length error	
6019	WKC timeout	
6020	STATE switch timeout	
6021	SDO ABORT, drive return error	Data dictionary write & read error or unsupported data of drive is written.
6023	NODE PROFILE error	
6024	Axis PROFILE error	
6025	The number of axes exceeds	
6029	PDO list length exceeds the number of system specification	
6031	The number of devices exceeds	
6042	Device does not support	
6045	E-mail timeout	
6047	Data type error	
6049	Sub module is not supported by module	
6050	Submodule numbers exceeds the limit	
6051	Unknown submodule by module	
6208	RTEX drive ID conflict	
6209	Scan timeout	Usually, net cable appears problem
6210	RTEX fails to initialize	
6211	RTEX scan result error	
20000: PC side error		
20000	There is wrong offset on PC side	
20002	Wrong parameters	
20003	time out	Fifo buffer may be blocked
20006	Operating system error	
20007	Fail to open serial	
20008	Fail to turn Ethernet on	
20009	Handle Error	
20010	Send Error	
20011	File Error	
20012	File length error	
20013	File name is too long	
20014	file does not exist	

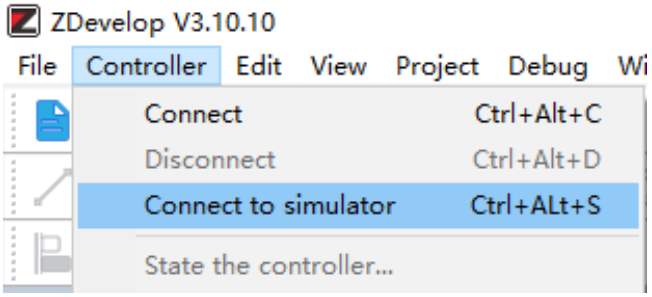
20015	ZLB library file errors	
20016	File does not compile	General PLC file needs to compile
20020	Firmware file does not match	
20021	Unsupported functions	
20030	Input buffer length is not enough	
20100	Response buffer length is not enough	
30000	Above 30000 are the error codes generated from ZAUX auxiliary library.	
30001	No respond.	
30002	Respond error.	
30003	Parameter error.	
30004	Unsupported	
30005	Parameter error.	

13.3. Others: Quick Start / Common Problems

13.3.1. Quick Start

13.3.1.1. Simulation Connection

1. Open ZDevelop software (it can be downloaded from Zmotion website or contact us)
2. Connect to Simulator.

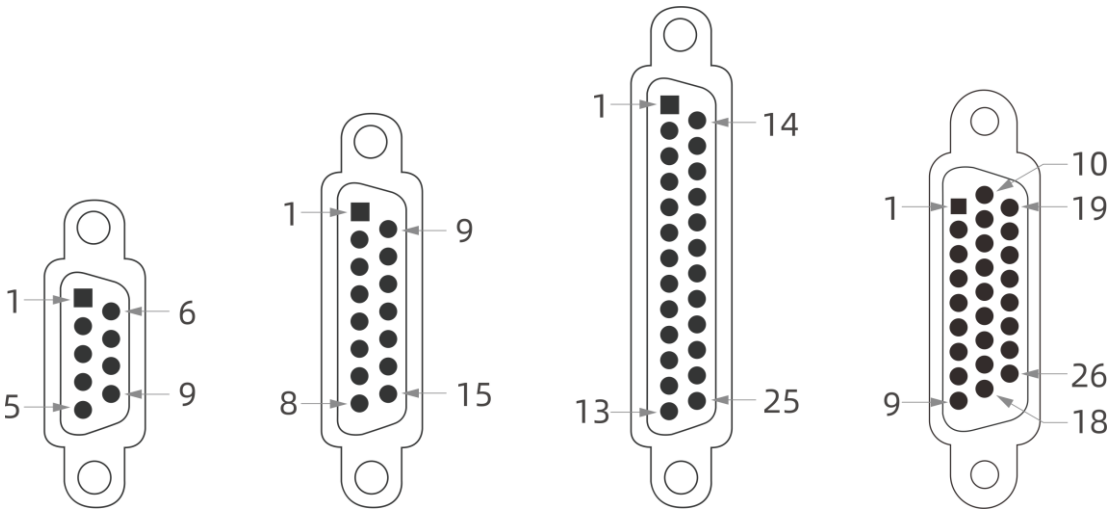


3. Connect to simulator through ethernet, IP address is 127.0.0.1.

```
// test1.cpp: define the entry point of control panel application program
//
#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
```

```
void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s return code is %d\n", command, ret);
	}

}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address, simulator IP address, it needs to open the simulator.
	ZMC_HANDLE handle = NULL;		//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to Controller!\n");
		handle = NULL;
		getchar();
		return -1;
	}
	printf("Success to Controller!\n");

	//obtain motion type}
	Sleep(2000);
	ret = ZAux_Close(handle);	//close the connection
	commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is executed successfully
	printf("connection closed!\n");
	handle = NULL;
	return 0;
}
```

13.3.1.2. Pulse Servo Diver Wiring



9PinDb head 15PinDb head 25PinDb head 26PinDb head

13.3.2. Common Problems

13.3.2.1. Not to Connect to Controller

The default IP address of the controller is 192.168.0.11, please change the IP address of the computer to the same network segment, and then connect. Some laptops have conflicts between the wireless connection and the network port. At this time, please don't try to connect through wireless.

(1) check the IP address of the PC, which needs to be on the same network segment.

(2) check the IP address of the controller, it can use the serial port to connect to check.

(3) check the PC firewall settings.

(4) Ping the IP of the controller to see if the controller can be pinged through. If the Ping fails, check the physical interface or the cable

If the PC cannot connect to the internet and the controller at the same time

Method 1: modify the IP address of the controller to be on the same network segment as the PC and the gateway.

Method 2: modify the subnet mask of the PC so that the IP address of the gateway and the controller are in the same network segment, for example, change it to: 255.255.128.0

13.3.2.2. Call the Motion Command, Axis Doesn't Move

Possibility 1:

The limit or alarm input, or the level configuration of the limit alarm is wrong.

The special signal is normally closed by default, and some Chinese products are normally open. It needs to set INVERT_IN to flip the corresponding port.

Possibility 2:

The atype of the axis is misconfigured.

13.3.2.3. How to Calculate IO Expansion Module No.

It is determined according to the ID of the module, and the ID is determined by the dial switch.

For ZIO1608:

ID0 - Input 16 - 31 Output 16-23

ID1 - Input 32 - 47 Output 32-39

And so on.

13.3.2.4. Whether All Commands are Sent Directly Through PC

Some commands such as WAIT and CAN are restricted from being sent from the PC, because it will cause the PC command channel to block.

It is also not possible to define a variable array through the PC command.

13.3.2.5. How to Stop the Motion Immediately

There is no absolute immediate stop. By setting the fastdel parameter to a large value, it can stop quickly when the limit or rapidstop occurs.

If move again after stopping, it is best to add the judgment of wait idle

rapidstop ignores base and stops all axes

13.3.2.6. How to Achieve Handwheel

It can use “connect" command to achieve, and there is reference handwheel routine, please go to Zmotion website – technical support / download or contact us.

13.3.2.7. How to Access Some System States Directly By modbus

IO, position etc. can be accessed through special modbus registers.

Bits 10000 and 10000 are inputs and bits 20000 and 20000 are outputs.

Each axis of the word register occupies two words, in float format.

13.3.2.8. Motion Still Executes After Cancel

The default of cancel is to cancel the current movement, and the buffered movement will not be canceled. If all motions are cancelled, it should be cancel(2).

Check whether the acceleration and deceleration is set. There is no absolute immediate stop. By setting the fastdel parameter to a large value, the limit or rapidstop can be stopped quickly. If the deceleration or rapid deceleration is not set, then it can not stop.

13.3.2.9. How to Achieve Power Failure Storage

The VR data is saved when the power is turned off. If other data needs to be saved, it can be written into the flash and read out when the power is turned on. Note that the flash should not be read and written frequently, and there is a limit on the number of reads and writes.

13.3.2.10. How to Check Errors in Program Motion

Do not restart, use the ZDevelop tool to connect to it (select the method of

attaching to the current program), and now it can view the internal status of the program and the location of the error.

Enter "?*task" in the command bar also can refer to the cause and location of the error.

13.3.2.11. Position Limit Overshoot

1. Increase FASTDEC, or enlarge induction block sensing range of position limit signal.
2. Check whether inversion is set for position limit.

13.3.2.12. Continuous Interpolation is Invalid

1. Check if MERGE is set on the first BASE axis of the MOVESP instruction, or other axis parameter settings.
2. If the movement command is a small line segment, check whether the movement time of the small line segment is too short, resulting in insufficient movement speed on the PC side.

13.3.2.13. Sensor Signal Jumps, Other Signals are Affected

1. Check whether the power supply of the IO sensor is not the same as the IO power supply of the controller. At this time, the ground cables of the two need to be connected.
2. If the IO mapping of the drive, the IO signal jumps, it can detect whether the IO mapping is repeated.

13.3.2.14. Inverse Finding Slowly Near the Origin While Homing

The controller has added special protection for the situation that the servo motor may cross the origin. At this time, the controller may detect that the origin decelerates and stops, but after a complete stop, the origin cannot be detected, so it is mistakenly thought that the axis has passed the origin sensing range.

Solution: LSPEED is set to 0 when returning to zero, so that returning to zero

produces a certain deceleration distance.

13.3.2.15. No to Invert Level (0X System Input State Register on HMI)

Actually it reads the original state, but inverse instruction only be valid in IN reading.

13.3.2.16. No Execution for Whole Round Interpolation

The full-round interpolation must use relative circular interpolation, absolute circular can not be used.

13.3.2.17. How to Find Problem When ZAR File Downloading Error

Through latest ZDevelop software, directly download zar file, debug window will show error information.

13.3.2.18. No To Move When PC Sends Motion

1. Stop command is called continuously for controller BASIC program.
2. Check the wiring.
3. Check the axis state to see whether there is the error.

Zmotion PC Program Manual

For advanced functions

Chapter I Axis Superposition / Electronic Cam

1.1.Motion Superposition

1.1.1. Emphasis

Motion superposition, which superimposes the motion of one axis to another axis.

The ADDAX command superimposes the number of pulses, not the set pulse amount.

Conversion relation: motion distance of superimposing axis * pulse amount of superimposing axis / pulse amount of superimposed axis = motion distance of superimposed axis.

- ✧ Suppose the pulse amount of axis A is 100, the pulse amount of axis B is 50, and the superimposing axis movement is 100
- ✧ Superimpose the movement of axis A to axis B. At this time, axis A shows a movement of 100, and axis B moves 100*100/50=200.
- ✧ The movement of axis B is superimposed on axis A. At this time, axis B shows a movement of 100, and axis A moves 100*50/100=50.

The axes cannot be superimposed on each other at the same time. That is, after A is superimposed on B, B can no longer be superimposed on A.

Support series superposition, A is superimposed to B, B is superimposed to C.

Support parallel superposition, A superimposes to B and C at the same time.

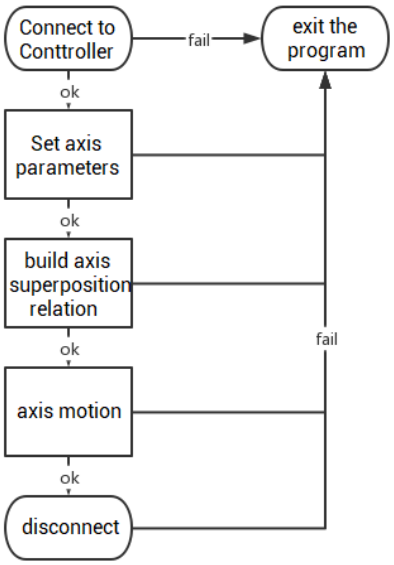
When superimposing, the speed changes from the superimposed axis, and the acceleration and deceleration are determined according to the acceleration and deceleration of the superimposing axis and the ratio of the units of the two axes.

Note: ZAux_Direct_Single_Addax function does not work when the current state of the axis is the positive solution state (forward kinematic) or the negative solution state (inverse kinematic).

1.1.2.Routine

1.1.2.1. Motion Superposition

The pulse amount of axis 0 is 100, the pulse amount of axis 1 is 50. Superpose axis 0 on axis 1, axis 0 moves 100.

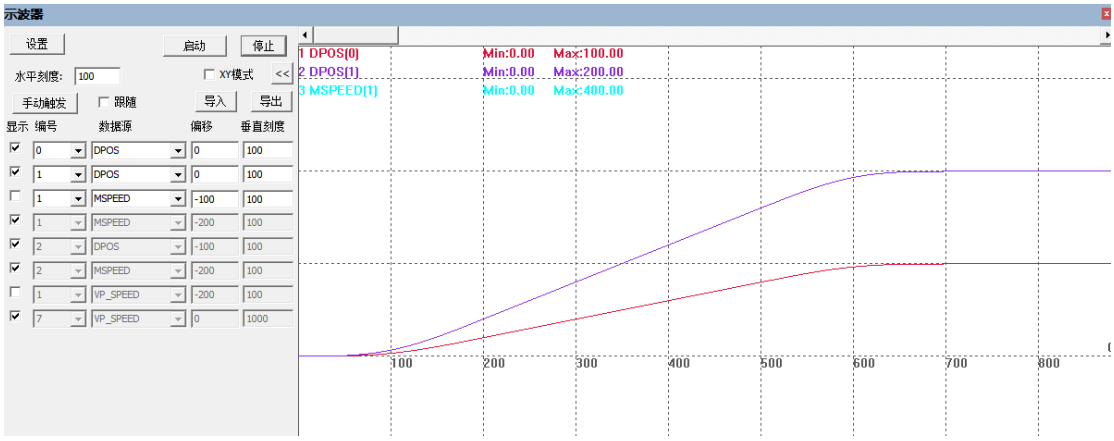



```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
	}

}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	ZMC_HANDLE handle = NULL;			//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
```

```
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to Controller! \n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to Controller!\n");

    ZAux_Direct_SetAtype(handle, 0, 1);  //set axis 0 axis type as 1
    ZAux_Direct_SetUnits(handle, 0, 100); //set axis 0 pulse amount as 100
    ZAux_Direct_SetAtype(handle, 1, 1);  //set axis 1 axis type as 1
    ZAux_Direct_SetUnits(handle, 1, 50); //set axis 1 pulse amount as 50

    ZAux_Direct_SetSpeed(handle, 0, 200); //set axis 0 speed as 200units/s
    ZAux_Direct_SetAccel(handle, 0, 2000); //set axis 0 acceleration as 2000units/s/s
    ZAux_Direct_SetDecel(handle, 0, 2000); //set axis 0 deceleration as
2000units/s/s
    ZAux_Direct_SetSramp(handle, 0, 200); //set axis 0 S curve time as 200ms

    ret = ZAux_Direct_SetDpos( handle,  0,  0);//clear axis command position to 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//judge whether the
instruction is executed successfully
    ret = ZAux_Direct_SetMpos( handle,  0,  0);//clear encoder feedback position to 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);//judge whether the
instruction is executed successfully

    ZAux_Direct_Single_Addax(handle,1,0);//axis 0 pulse is superposed on axis 1

    ZAux_Trigger(handle);
    ZAux_Direct_Single_Move(handle,0,100);//axis 0 moves 100
    float IDLE;
    while (1)//wait for axis 0 to complete the motion
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }
    ZAux_Direct_Single_Addax(handle,1,-1);//cancel superposition

    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");

    handle = NULL;
```

```
    return 0;
}
```

➢ Waveform:



1.2. Electronic Cam & Synchronous Follow

1.2.1. Emphasis

(1) Electronic Cam

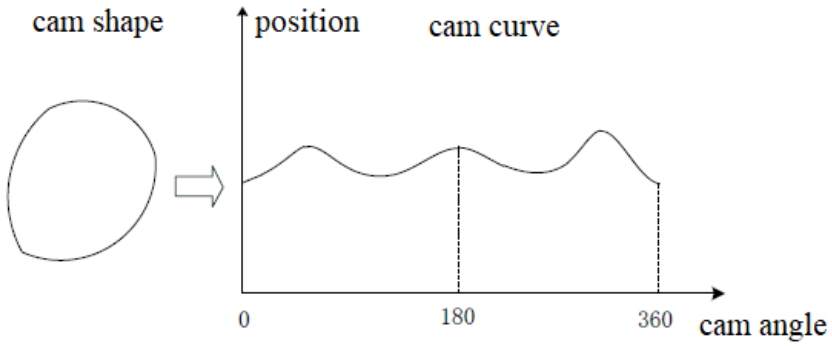
➢ **Components**

Electronic cam is used to simulate mechanical cam through constructed cam curve, in this way, we can achieve the software system of relative motion between cam axes and master axes in same mechanical cam system. Namely, through controller controls servo motor, it can simulate mechanical cam, no need to install other mechanical structures. It is also can be written as Electronic CAM.

➢ **The theory**

Electronic cam actually belongs to multi-axes synchronous motion, which is based on adding one or multiple slave axis systems, that is, it is developed on the basis of mechanical cam. Electronic cam is usually used for periodical curve motions.

As following graphic, motion trajectory between one rotating angle and processing position is gained from the contour of mechanic cam. This trajectory is called arc, and decompose this arc line as countless linear trajectory, and then constitute one series trajectory approaching arc motion. When electronic cam directly adds this segment trajectory motion parameters into motion instructions, then, it can control axes go out the target trajectory.



Put one motion program of electronic cam into controller, then feedback the position signal to controller through encoder. When position signals are received and processed by the controller, then all are sent to servo driver. Then, the driver controls multi axes synchronous motion to complete preset trajectory.

➢ **Advantages**

There are many advantages of electronic cam.

- promote control precision of machine, make control distance longer, low error ratio, promote the reliability.
- simplify institution and make it more flexible, make debug and maintain easier.
- electronic cam uses software to control signals, relative motion parameters of program are changed, motion curve will be changed, which means it has high flexibility.
- Easy to install, no need extra mechanical components, so no tool wear situation.

(2) Synchronous Follow

Synchronous follow is a kind of electronic cam application, which is similar with “flying sheer", both need position synchronization between mater and slave axes, the difference is that, in “flying sheer", feeding material (master) is continuous, but in “synchronous follow", feeding material is random (materials appear randomly), and it usually will add sensor as signal to trigger synchronization, each time signal comes, synchronization motion will only happen once. It will go back to starting position after achieving set distance, and follow again until next trigger. This is commonly applied in assembly line, such as, synchronous paint spraying, container material injection, material clamping, etc.

(3) Automatic Cam

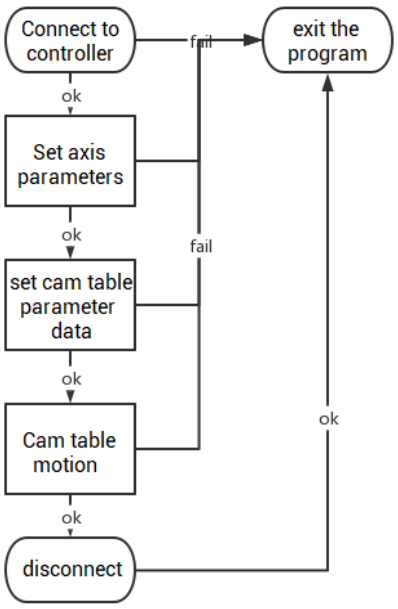
Automatic cam is a kind of cam motion. It is mainly aimed at the master-slave follow-up movement between two axes. The user can simply set a few related

parameters to build the motion relationship between the master axis and the slave axis. The position relationship is not stored in the TABLE table, the following distance and speed change process of each segment are set by command parameters. The speed of the slave axis is automatically calculated to match the main axis during the motion process. Common motion processes include following acceleration, deceleration, and synchronization.

Automatic cam commands include MOVELINK, MOVESLINK, FLEXLINK, etc. Common applications include chasing shear, flying shear, and wheel cutting.

1.2.2.Routine

1.2.2.1. Cam Table Routine

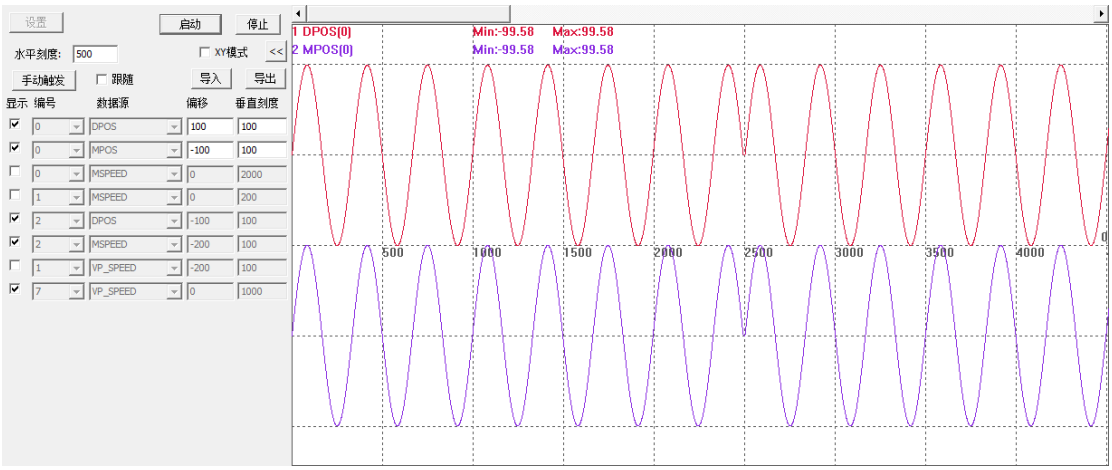



```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
#define PI 3.1415
void commandCheckHandler(const char *command, int ret)
{
```

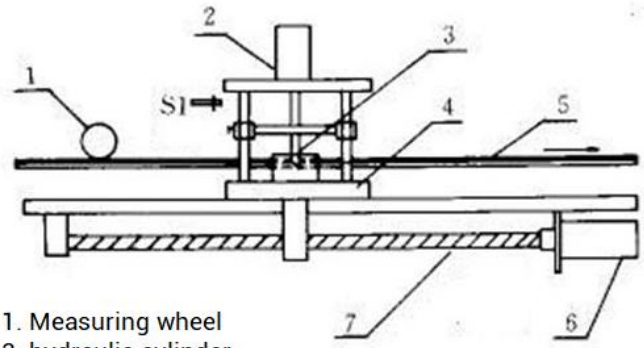
```
    if (ret)//it is not 0, fail
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }
}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";          //controller IP address
    //char *ip_addr = (char *)"192.168.1.11";          //controller IP address
    ZMC_HANDLE handle = NULL;               //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to Controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to Controller!\n");

    ret =ZAux_Direct_SetAtype(handle, 0, 1);  //set axis 0 axis type as 1
    commandCheckHandler("ZAux_Direct_SetAtype", ret);//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetUnits(handle, 0, 100); //set axis 0 pulse amount as 100
    commandCheckHandler("ZAux_Direct_SetUnits", ret);//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetSpeed(handle, 0, 100); //set axis 0 speed as 100units/s
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetAccel(handle, 0, 1000); //set axis 0 acceleration as
1000units/s/s
    commandCheckHandler("ZAux_Direct_SetAccel", ret);//judge whether the
instruction is executed successfully
    ret =ZAux_Direct_SetDecel(handle, 0, 1000); //set axis 0 deceleration as
1000units/s/s
    commandCheckHandler("ZAux_Direct_SetDecel", ret);//judge whether the
instruction is executed successfully

    ret = ZAux_Direct_SetDpos( handle,  0,  0);//clear axis command position to 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);//judge whether the
instruction is executed successfully
    ret = ZAux_Direct_SetMpos( handle,  0,  0);//clear encoder feedback position to 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);//judge whether the
```

```
instruction is executed successfully

    int TableStartPoint = 10;                    //Table starting position
    int TableEndPoint = 10+1024-1;                    //Table end position
    float ftablemulti = 100;      //position ratio
    float ReferDistance = 1000;                    //refer to movement distance
    int i,j;

    float setTableValue[1024];//cam table data

    //write sine curve data
    for ( j=0;j<4;j++)
    {
        for ( i=0;i<256;i++)
        {
            setTableValue[i+j*256]=(sin(((2*PI)/34)*i))*100;
        }
    }

    ZAux_Direct_SetTable(handle,TableStartPoint,1024,setTableValue);//write cam
table data into Table register value

    ZAux_Trigger(handle);
    ret = ZAux_Direct_Cam( handle, 0, TableStartPoint, TableEndPoint, ftablemulti,
ReferDistance); //cam table motion
    commandCheckHandler("ZAux_Direct_Cam", ret);
    float IDLE;
    while (1)//wait for axis 0 to complete the motion
    {
        Sleep(100);
        ZAux_Direct_GetParam(handle,"IDLE",0,&IDLE);
        if (IDLE<0)break;
    }

    Sleep(6000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

- Waveform:



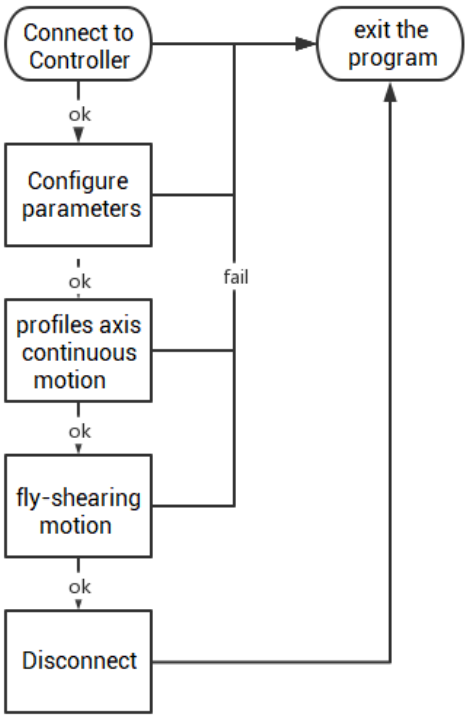
1.2.2.2. Fly-Shearing



1. Measuring wheel
2. hydraulic cylinder
3. knives
4. workbench
5. profiles
6. servo motor
7. Screw

The profile continues to move, and the workbench stops first, until the profile continues to move for a certain distance, the workbench starts to accelerate, when the speed of the workbench is consistent with the profile, then switch the S1 working tool to cut down, and the tool picks up after cutting, the workbench starts to decelerate, and back to the starting point. Repeat the process and cut to obtain the profile of the set length.

Suppose the length of the profile to be cut is 4m, the working distance of the table is 1m, the axis 1 is the basic axis (profile transmission), the axis 0 is the following axis (following shear table), and the OUT0 port controls the tool. The flying shear part program is as follows:

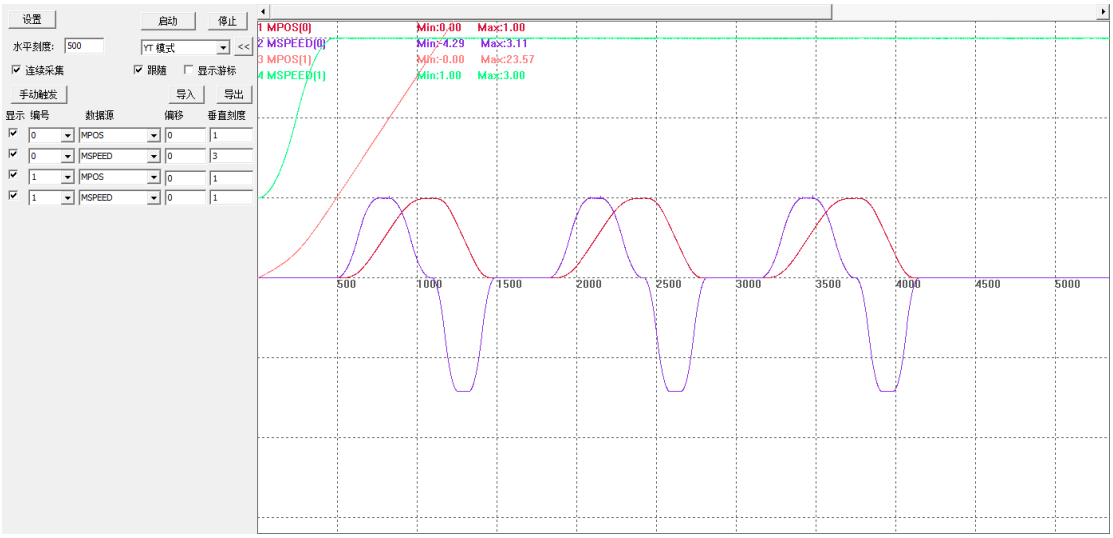



```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"

void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
		Sleep(2000);
		exit(0);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	//char *ip_addr = (char *)"192.168.0.11";		//controller IP address
	ZMC_HANDLE handle = NULL;				//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to Controller!\n");
```

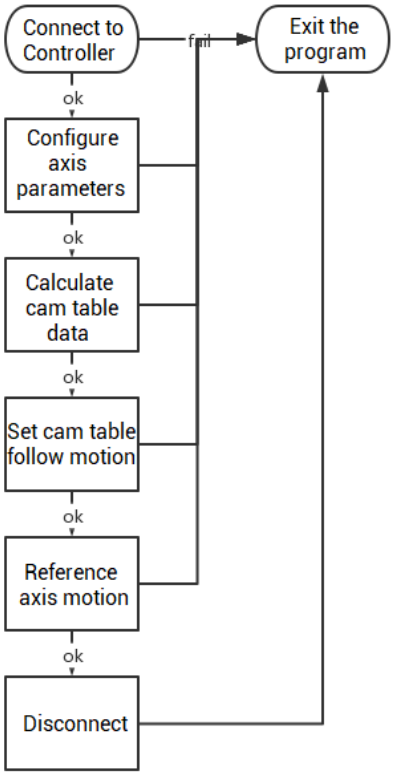
```
            handle = NULL;
            Sleep(2000);
            return -1;
        }
        printf("Success to Controller!\n");

        int FirstAxis = 0, SecondAxis = 1;                  //define axis 0, axis 1
        int AType = 1;                                      //set axis type
        float DposValue = 0;                                //set DPOS
        float SpeedValue = 3;                               //set the value of speed, profile
running speed  is 1m/s,60m/min
        float AccelValue = 10;                              //set acceleration value
        float DecelValue = 10;                              //set deceleration value
        float UnitsValue = 10000;                           //set units value

        ret = ZAux_Direct_SetAtype(handle, FirstAxis, AType);           //set axis 0 type
        commandCheckHandler("ZAux_Direct_SetAtype", ret);
        ret = ZAux_Direct_SetAtype(handle, SecondAxis, AType);          //set axis 1 type
        commandCheckHandler("ZAux_Direct_SetAtype", ret);
        ret = ZAux_Direct_SetSpeed( handle, FirstAxis, SpeedValue);     //set axis 0 speed
        commandCheckHandler("ZAux_Direct_SetSpeed", ret);
        ret = ZAux_Direct_SetSpeed(handle, SecondAxis, SpeedValue); //set axis 1 speed
        commandCheckHandler("ZAux_Direct_SetSpeed", ret);
        ret = ZAux_Direct_SetDpos( handle, FirstAxis, DposValue);       //set axis 0 DPOS
        commandCheckHandler("ZAux_Direct_SetDpos", ret);
        ret = ZAux_Direct_SetDpos(handle, SecondAxis, DposValue);    //set axis 1 DPOS
        commandCheckHandler("ZAux_Direct_SetDpos", ret);
        ret = ZAux_Direct_SetUnits(handle, FirstAxis, UnitsValue);       //set axis 0 UNITS
        commandCheckHandler("ZAux_Direct_SetUnits", ret);
        ret = ZAux_Direct_SetUnits(handle, SecondAxis, UnitsValue);    //set axis 1 UNITS
        commandCheckHandler("ZAux_Direct_SetUnits", ret);
        ret = ZAux_Direct_SetAccel(handle, FirstAxis, AccelValue);       //set axis 0
acceleration
        commandCheckHandler("ZAux_Direct_SetAccel", ret);
        ret = ZAux_Direct_SetAccel(handle, SecondAxis, AccelValue);   //set axis 1
acceleration
        commandCheckHandler("ZAux_Direct_SetAccel", ret);
        ret = ZAux_Direct_SetDecel(handle, FirstAxis, DecelValue);       //set axis 0
deceleration
        commandCheckHandler("ZAux_Direct_SetDecel", ret);
        ret = ZAux_Direct_SetDecel(handle, SecondAxis, DecelValue);   //set axis 1
deceleration
        commandCheckHandler("ZAux_Direct_SetDecel", ret);
        ret = ZAux_Direct_SetSramp(handle, FirstAxis, 400);          //set axis 0 Sramp
        commandCheckHandler("ZAux_Direct_SetDecel", ret);
        ret = ZAux_Direct_SetSramp(handle, SecondAxis, 400);      //set axis 1 Sramp
```

```
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Trigger( handle);                    //trigger the oscilloscope
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Direct_Single_Vmove(handle, SecondAxis, 1); //profile continuous
motion
    commandCheckHandler("ZAux_Direct_Single_Vmove", ret);

    int i;
    for (i=0;i<3;i++)//fly-shearing three times
    {
        ret = ZAux_Direct_Movelink( handle, FirstAxis, 0 , 1 , 0 , 0 , SecondAxis,8,
0);//before profile motion 1m, workbench is static.
        commandCheckHandler("ZAux_Direct_Movelink1", ret);
        ret = ZAux_Direct_Movelink( handle, FirstAxis, 0.4 , 0.8 , 0.8 , 0 , SecondAxis,
8, 0);//workbench is in acceleration stage: profile moves 0.8m, workbench accelerates
0.4m.
        commandCheckHandler("ZAux_Direct_Movelink2", ret);
        ret = ZAux_Direct_Movelink( handle, FirstAxis, 0.2 , 0.2 , 0 , 0 , SecondAxis, 8,
0);//synchronous follow 0.2m
        commandCheckHandler("ZAux_Direct_Movelink3", ret);
    ZAux_Direct_MoveOp2(handle, FirstAxis,0,1,1000);//operate OP to make tool
 down to cut, lift back after 1s (the time needs to be calculated)
        commandCheckHandler("ZAux_Direct_MoveOp2", ret);
        ret = ZAux_Direct_Movelink( handle, FirstAxis, 0.4 , 0.8 , 0 , 0.8, SecondAxis, 8,
0);//workbench is in deceleration stage: profile moves 0.8m, workbench accelerates
0.4m.
        commandCheckHandler("ZAux_Direct_Movelink2", ret);
        ret = ZAux_Direct_Movelink( handle, FirstAxis, -1 , 1.2 , 0.5 , 0.5, SecondAxis,
8, 0);//workbench goes to starting point: profile moves 1.2m, workbench moves 1m
inversely.
        commandCheckHandler("ZAux_Direct_Movelink2", ret);
    }

    Sleep(5000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

➢ Waveform:



1.2.2.3. Electronic Cam Synchronous Motion (Follow Cam Table)



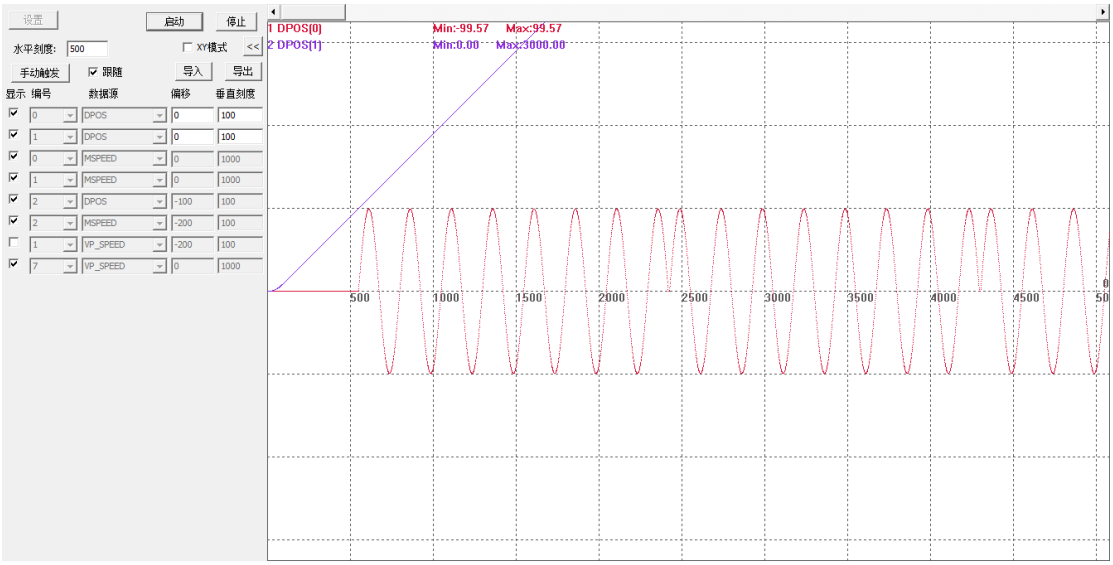

```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
```

```
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//it is not 0, fail
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
        exit(0);
    }


}

int _tmain(int argc, _TCHAR* argv[])
{
    char *ip_addr = (char *)"127.0.0.1";          //controller IP address
    //char *ip_addr = (char *)"192.168.0.11";          //controller IP address
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to Controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to Controller!\n");

    int firstAxis = 0, secondAxis = 1;          //define axis 0 and axis 1
    int AType = 1;                              //set axis type
    int DposValue = 0;                          //set DPOS
    int UnitValue = 100;                        //set pulse amount (UNITS)
    int SpeedValue = 200;                       //set speed
    int AccelValue = 2000;                      //set acceleration
    int DecelValue = 2000;                      //set deceleration

    ret = ZAux_Direct_SetAtype(handle, firstAxis, AType);       //set axis 0 type
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ret = ZAux_Direct_SetAtype(handle, secondAxis, AType);    //set axis 1 type
    commandCheckHandler("ZAux_Direct_SetAtype",ret);
    ret = ZAux_Direct_SetDpos(handle, firstAxis, DposValue);  //set axis 0 DPOS
value
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, secondAxis, DposValue);    //set axis 1 DPOS
```

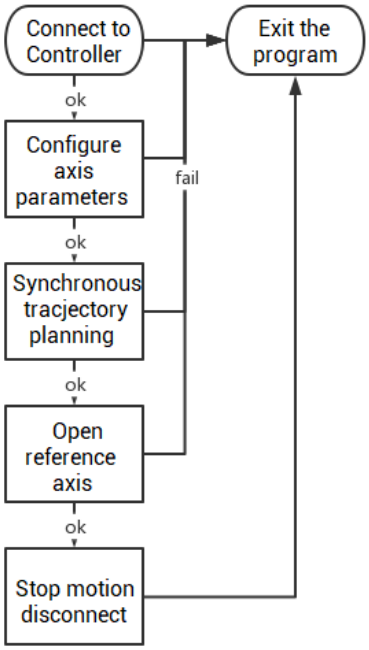
```
value
    commandCheckHandler("ZAux_Direct_SetDpos",ret);
    ZAux_Direct_SetUnits(handle, firstAxis, UnitValue);          //set axis 0 UNITS
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetUnits(handle, secondAxis, UnitValue);      //set axis 1 UNITS
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, firstAxis,SpeedValue);       //set axis 0 speed
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetSpeed(handle, secondAxis, SpeedValue);  //set axis 1 speed
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, firstAxis, AccelValue);   //set axis 0 acceleration
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetAccel(handle, secondAxis, AccelValue); //set axis 1 acceleration
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, firstAxis, DecelValue);  //set axis 0 deceleration
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ZAux_Direct_SetDecel(handle, secondAxis, DecelValue); //set axis 1 deceleration
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    //{calculate Table data
    float deg;
    float rad;
    float x;
    int stepdeg = 2;
    float PI = 3.1416;

    uint32 IN_Status; //receive state value of input
    int piValue;         //read motion status of axis 1
    int i,j;
    int TableStartPoint = 10;                    //Table starting position
    int TableEndPoint = 10+1024-1;                        //Table end position
    float setTableValue[1024];//cam table data

    //write sine curve data
    for ( j=0;j<4;j++)
    {
        for ( i=0;i<256;i++)
        {
            setTableValue[i+j*256]=(sin(((2*PI)/34)*i))*100;
        }
    }

    ZAux_Direct_SetTable(handle,TableStartPoint,1024,setTableValue);//write cam
table data into Table register value
    //calculate Table data}
```

```
        ret = ZAux_Trigger(handle);
        commandCheckHandler("ZAux_Trigger", ret);

        ret = ZAux_Direct_Cambox(handle, firstAxis, TableStartPoint,
TableEndPoint,100,1500,1,2,100);//when reference axis 1 moves to position 100,
following axis 0 opens cam table motion

        ret = ZAux_Direct_Single_Move(handle, secondAxis, 3000);
        commandCheckHandler("ZAux_Direct_Single_Move", ret);

        Sleep(5000);
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
        printf("connection closed!\n");

        handle = NULL;
        return 0;
}
```

➢ Waveform:



1.2.2.4. Electronic Cam Synchronous Motion (movelink)



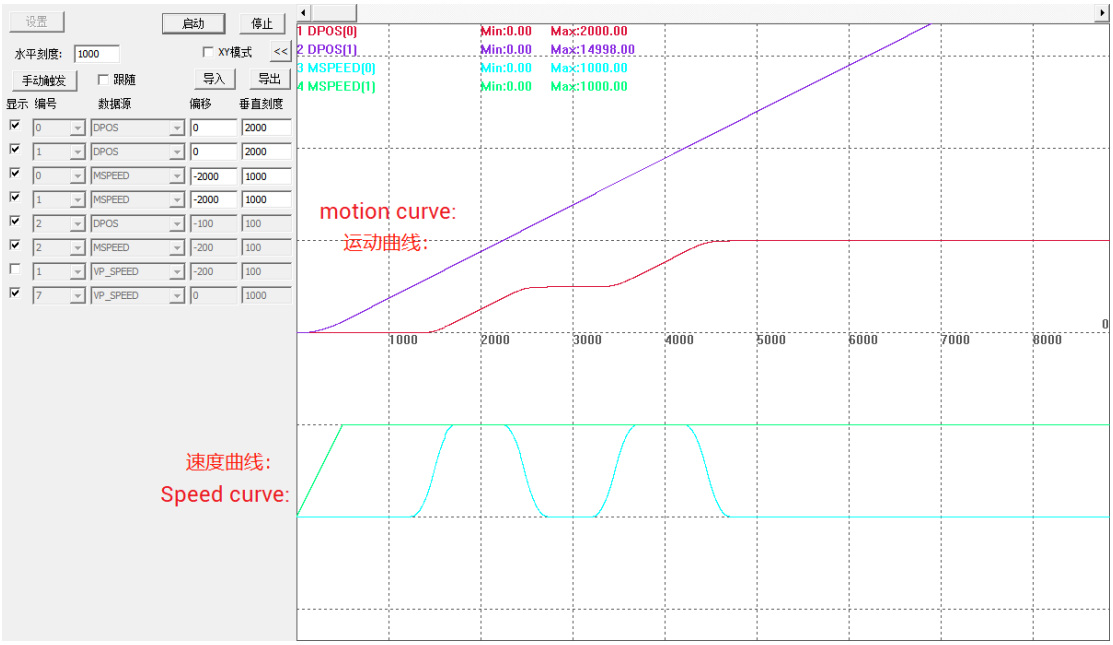

```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
		Sleep(2000);
		exit(0);
	}

}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	//char *ip_addr = (char *)"192.168.0.11";		//controller IP address
	ZMC_HANDLE handle = NULL;				//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
```

```
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to Controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to Controller!\n");

    int FirstAxis = 0, SecondAxis = 1;          //define axis 0 and axis 1
    int AType = 1;                              //set axis type
    float DposValue = 0;                        //set DPOS
    float SpeedValue = 100;                     //set speed
    float AccelValue = 2000;                    //set acceleration
    float DecelValue = 2000;                    //set deceleration
    float UnitsValue = 100;                     //set Units value

    char psRespons[1024];  //used for array that is received controller model
    int length = 1024;        //the length of controller model must be larger than
received character string length
    float DistanceValue = 150;

    float fDistance = 50;
    float fLinkDis = 100;
    float startsp = 0;
    float endsp = 1;
    int iLinkaxis = SecondAxis;
    int ioption = 2;
    float flinkstartpos = 50;

    ret = ZAux_Direct_SetAtype(handle, FirstAxis, AType);  //set axis 0 type
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ret = ZAux_Direct_SetAtype(handle, SecondAxis, AType); //set axis 1 type
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ret = ZAux_Direct_SetDpos( handle,  FirstAxis, DposValue); //set axis 0 DPOS
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, SecondAxis, DposValue); //set axis 1 DPOS
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetUnits(handle, FirstAxis, UnitsValue);  //set axis 0 UNITS
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ret = ZAux_Direct_SetUnits(handle, SecondAxis, UnitsValue);//set axis 1 UNITS
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ret = ZAux_Direct_SetAccel(handle, FirstAxis, AccelValue); //set axis 0
acceleration
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ret = ZAux_Direct_SetAccel(handle, SecondAxis, AccelValue);   //set axis 1
```

```
acceleration
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ret = ZAux_Direct_SetDecel(handle, FirstAxis, DecelValue);//set axis 0
deceleration
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDecel(handle, SecondAxis, DecelValue);   //set axis 1
deceleration
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Trigger(handle);                 //trigger the oscilloscope
    commandCheckHandler("ZAux_Trigger", ret);

    ret = ZAux_Direct_Moveslink(handle, 0, 250, 500, 0, 1, 1, 2, 1000);//when axis 1 is
at position 1000-1500, axis 0 accelerates to synchronize with axis 1, the acceleration
process motion is all 250.
    commandCheckHandler("ZAux_Direct_Moveslink", ret);
    ret = ZAux_Direct_Moveslink(handle, 0, 500, 500, 1, 1, 1, 2, 1500);//when axis 1 is
at position 1500-2000, axis 0 synchronizes with axis 1 to move 500.
    commandCheckHandler("ZAux_Direct_Moveslink", ret);
    ret = ZAux_Direct_Moveslink(handle, 0, 250, 500, 1, 0, 1, 2, 2000);//when axis 1 is
at position 2000-2500, axis 0 and axis 1 decrease to 0, the deceleration process all
moves 250.
    commandCheckHandler("ZAux_Direct_Moveslink", ret);

    ret = ZAux_Direct_Moveslink(handle, 0, 250, 500, 0, 1, 1, 2, 3000);//when axis 1 is
at position 3000-3500, axis 0 accelerates to synchronize with axis 1, the acceleration
process motion is all 250.
    commandCheckHandler("ZAux_Direct_Moveslink", ret);
    ret = ZAux_Direct_Moveslink(handle, 0, 500, 500, 1, 1, 1, 2, 3500);//when axis 1 is
at position 3500-4000, axis 0 synchronize with axis 1 to move 500.
    commandCheckHandler("ZAux_Direct_Moveslink", ret);
    ret = ZAux_Direct_Moveslink(handle, 0, 250, 500, 1, 0, 1, 2, 4000);//when axis 1 is
at position 4000-4500, axis 0 and axis 1 decrease to 0, the deceleration process all
moves 250.
    commandCheckHandler("ZAux_Direct_Moveslink", ret);

    ZAux_Direct_Single_Vmove(handle, 1,1);//open axis 1
    Sleep(15000);
    ZAux_Direct_Single_Cancel(handle, 1,2);//stop axis 1
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");

    handle = NULL;
    return 0;
```

```
}
```

➢ Waveform:

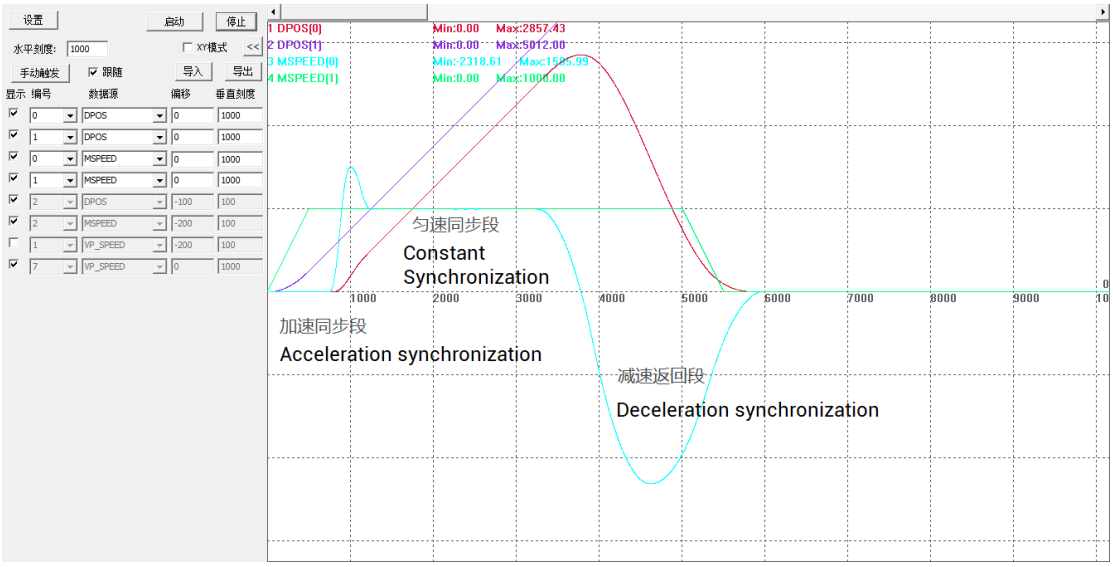


1.2.2.5. Belt Synchronous Follow Motion (MOVESYNC)

```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
		Sleep(2000);
		exit(0);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//controller IP address
```

```
    //char *ip_addr = (char *)"192.168.0.11";          //controller IP address
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to Controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to Controller!\n");

    int FirstAxis = 0, SecondAxis = 1;              //define axis 0 axis 1
    int AType = 1;                                  //set axis type
    float DposValue = 0;                            //set DPOS
    float SpeedValue = 100;                         //set speed
    float AccelValue = 2000;                        //set acceleration
    float DecelValue = 2000;                        //set deceleration
    float UnitsValue = 100;                         //set Units

    ret = ZAux_Direct_SetAtype(handle, FirstAxis, AType);           //set axis 0 type
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ret = ZAux_Direct_SetAtype(handle, SecondAxis, AType);          //set axis 1 type
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ret = ZAux_Direct_SetDpos( handle,  FirstAxis, DposValue);      //set axis 0 DPOS
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetDpos(handle, SecondAxis, DposValue);   //set axis 1 DPOS
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetUnits(handle, FirstAxis, UnitsValue);  //set axis 0 UNITS
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ret = ZAux_Direct_SetUnits(handle, SecondAxis, UnitsValue);   //set axis 1 UNITS
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ret = ZAux_Direct_SetAccel(handle, FirstAxis, AccelValue); //set axis 0
acceleration
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ret = ZAux_Direct_SetAccel(handle, SecondAxis, AccelValue);   //set axis 1
acceleration
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ret = ZAux_Direct_SetDecel(handle, FirstAxis, DecelValue);//set axis 0
deceleration
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDecel(handle, SecondAxis, DecelValue);   //set axis 1
deceleration  commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Trigger(handle);                 //trigger the oscilloscope
    commandCheckHandler("ZAux_Trigger", ret);
```

```
    ZAux_Direct_Single_Vmove(handle, 1,1);//open axis 1, simulate belt axis motion

    ZAux_Direct_MoveWait(handle,0,"MPOS",1,1,500);//when axis 1 moves to 500, open synchronous motion
    float fpos = 500;
    ZAux_Direct_MoveSync(handle,0,0,1000,1,1,&FirstAxis,&fpos);//the belt axis moves to 1000, the following axis 0 moves to 500, and the speed has been synchronized, the synchronization time is automatically calculated according to the acceleration and deceleration

    ZAux_Direct_MoveSync(handle,0,2000,1000,1,1,&FirstAxis,&fpos);//continue to synchronize 2s
    fpos=0;
    ZAux_Direct_MoveSync(handle,-1,0,300,1,1,&FirstAxis,&fpos);//following axis 0 stops synchronization, returns to initial position

    Sleep(5000);
    ZAux_Direct_Single_Cancel(handle, 1,2);//stop axis 1
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is executed successfully
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```



Chapter II High-Speed Latch Application

2.1.High-Speed Latch

2.1.1.Emphasis

ZAux_Direct_Regist command is used to latch axis' measurement feedback position.

Encoder axis latching is supported, and virtual axis and pulse axis latching are supported by ZMC4XX series controllers with the latest firmware.

EtherCAT supports drive latch, now, latch is achieved by drive IO.

For RTEX, it only supports controller latching.

When latching occurs, axis status ZAux_Direct_GetMark will become ON, at the same time, latched position will be saved into ZAux_Direct_GetRegPos.

Each axis has R0, R1 and EZ signals to trigger latch action. When 2 signals are used, the latch value from the second signal will be used in ZAux_Direct_GetMarkB and ZAux_Direct_GetRegPosB to latch.

- Single-time latch mode: **ZAux_Direct_Regist**

  When input signal R0 in controller is mapped into IN0 by default (used controller supports latching).

  When input signal R1 in controller is mapped into IN1 by default (used controller supports latching).

  When input signal R2 in controller is mapped into IN2 by default (used controller supports latching whose input should be larger than 2).

  When input signal R3 in controller is mapped into IN3 by default (used controller supports latching whose input should be larger than 2).

  1. Save absolute position in REG_POS when meet rising edge of Z pulse.
  2. Save absolute position in REG_POS when meet falling edge of Z pulse.
  3. Save absolute position in REG_POS when meet rising edge of input signal R0.
  4. Save absolute position in REG_POS when meet falling edge of input signal R0.
  6. Save absolute position in REG_POS when meet rising edge of input signal R0, and save absolute position in REG_POSB when meet rising edge of Z signal.

7. Save absolute position in REG_POS when meet rising edge of input signal R0, and save absolute position in REG_POSB when meet falling edge of Z signal.
8. Save absolute position in REG_POS when meet falling edge of input signal R0, and save absolute position in REG_POSB when meet rising edge of Z signal.
9. Save absolute position in REG_POS when meet falling edge of input signal R0, and save absolute position in REG_POSB when meet falling edge of Z signal.
10. Save absolute position in REG_POS when meet rising edge of input signal R0, and save absolute position in REG_POSB when meet rising edge of signal R1.
11. Save absolute position in REG_POS when meet rising edge of input signal R0, and save absolute position in REG_POSB when meet falling edge of signal R1..
12. Save absolute position in REG_POS when meet falling edge of input signal R0, and save absolute position in REG_POSB when meet rising edge of signal R1.
13. Save absolute position in REG_POS when meet falling edge of input signal R0, and save absolute position in REG_POSB when meet falling edge of signal R1.
14. Save absolute position in REG_POSB when meet rising edge of input signal R1.
15. Save absolute position in REG_POSB when meet falling edge of input signal R1.
16. Save absolute position in REG_POSB when meet rising edge of Z signal.
17. Save absolute position in REG_POSB when meet falling edge of Z signal.
18. Save absolute position in REG_POSB when meet rising edge of input signal R2.
19. Save absolute position in REG_POSB when meet falling edge of input signal R2.
20. Save absolute position in REG_POSB when meet rising edge of input signal R3.
21. Save absolute position in REG_POSB when meet falling edge of input signal R3.

Note: for pulse axis type, usually it uses R0, R1 and Z pulse signals to latch. For bus axis type, it uses R2 and R3 to latch.

- Continuous Latch: **ZAux_Direct_CycleRegist**(ZMC_HANDLE handle,int iaxis, int imode,int iTabStart,int iTabNum)

Parameters:

handle: connection handle

iaxis: controller axis

imode: latch method

iTabStart: continuous latched values are saved in table position, the first table element saves the number of latched, behinds save latched coordinates, the max save numbers = numes-1, and it will cycle write if exceeds limit.

iTabNum: occupied number of table

Mode + 100 means continuous latch, and latched data are saved in TABLE register.

2 channels are latched continuously and separately, and continuous latch of up and down edge can be achieved.

ECI: valid in firmware above 20150829

4xx controllers: valid in firmware above 20170523

100 + mode: single channel mode can be used, add 100 means continuous latch.

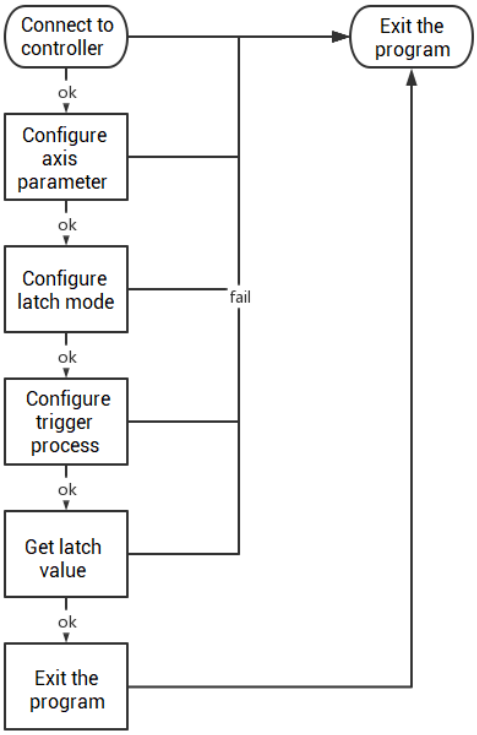
1. Save absolute position in REG_POS when meet rising edge of Z pulse.
2. Save absolute position in REG_POS when meet falling edge of Z pulse.
3. Save absolute position in REG_POS when meet rising edge of input signal R0.
4. Save absolute position in REG_POS when meet falling edge of input signal R0.

14. Save absolute position in REG_POSB when meet rising edge of input signal R1.
15. Save absolute position in REG_POSB when meet falling edge of input signal R1.
16. Save absolute position in REG_POSB when meet rising edge of input Z signal.
17. Save absolute position in REG_POSB when meet falling edge of Z signal.
23. Save absolute position in REG_POSB when meet rising edge of input signal R0.
24. Save absolute position in REG_POSB when meet falling edge of input signal R0.
33. Save absolute position in REG_POS when meet rising edge of input R0, and switch to falling edge next time, switch between these two in turn.
34. Save absolute position in REG_POS when meet falling edge of input R0, and switch to rising edge next time, switch between these two in turn.
35. Save absolute position in REG_POS when meet rising edge of input R1, and switch to falling edge next time, switch between these two in turn.
36. Save absolute position in REG_POS when meet falling edge of input R1, and switch to rising edge next time, switch between these two in turn.

2.1.2.Routine

2.1.2.1. Position Latch

Axis 0 moves continuously, when IN0 senses the signal, then latch axis 0 encoder position value at high speed.

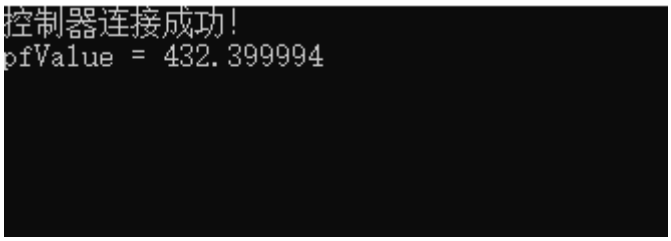



```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
		Sleep(2000);
		exit(0);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	//char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	char *ip_addr = (char *)"192.168.0.11";		//real controller IP address is
192.168.0.11 by default
	ZMC_HANDLE handle = NULL;				//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
```

```
            printf("Fail to Controller!\n");
            handle = NULL;
            Sleep(2000);
            return -1;
        }
        printf("Success to Controller!\n");

        int iaxis = 0;
        float DposValue = 0;
        float MposValue = 0;
        int mode = 3;       //wait for the absolute position of R0 rising edge
        int piValue;
        float pfValue;
        ret = ZAux_Direct_SetDpos( handle, iaxis, DposValue);//clear axis 0 dpos to 0
        commandCheckHandler("ZAux_Direct_SetDpos", ret);

        ret = ZAux_Direct_SetMpos( handle, iaxis, MposValue);//clear axis 0 mpos to 0
        commandCheckHandler("ZAux_Direct_SetMpos", ret);

        ret = ZAux_Direct_Single_Vmove( handle, iaxis, 1);//axis 0 moves forward
        commandCheckHandler("ZAux_Direct_Single_Move", ret);

        ret = ZAux_Direct_Regist( handle, iaxis, mode);//set single-time latch, latch mode:
trigger latching when IN0 is rising edge
        commandCheckHandler("ZAux_Direct_Regist", ret);

        ret = ZAux_Direct_MoveWait(handle, 1,"Mpos",iaxis,1,432);//in axis 1 buffer, add
the block (here axis 1 buffer is occupied) that executes the next command until axis 0
MPOS is larger than 432
        commandCheckHandler("ZAux_Direct_Regist", ret);

        /*after MPOS equals to 432, write rising edge into axis 1 buffer, this routine
connects ZMC series controllers, low level is valid level, so the rising edge is formed
when IN(0) is set from 1 to 0. (if it is ECI series, it is opposite, from 0 to1)

        ret = ZAux_Direct_MoveOp(handle, 1,0,1);//write that OUT0 is set to 1 into axis 1
buffer (here, the used controller has been connected OUT0 to IN0 directly)
        commandCheckHandler("ZAux_Direct_Regist", ret);
        ret = ZAux_Direct_MoveOp(handle, 1,0,0);//write that OUT0 is set to 0 into axis 1
buffer (here, the used controller has been connected OUT0 to IN0 directly)
        commandCheckHandler("ZAux_Direct_Regist", ret);

        while (1)
        {
            Sleep(2000);
            ZAux_Direct_GetMark(handle, iaxis, &piValue);
```

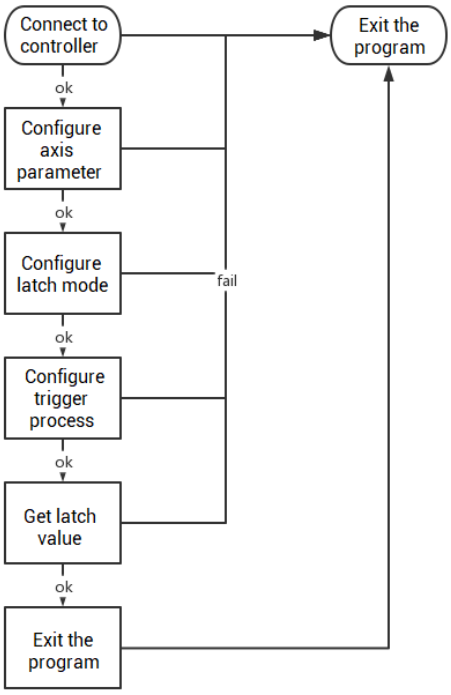
```
            if (piValue == -1)//trigger latching
            {
                ret =ZAux_Direct_GetRegPos(handle,0,&pfValue);
                   commandCheckHandler("ZAux_Direct_GetRegPos", ret);
                printf("pfValue = %f\n", pfValue);
                break;
            }
        }

        ret = ZAux_Direct_Single_Cancel( handle, 0, 2);//stop moving
        commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
        Sleep(5000);
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
        printf("connection closed!\n");

        handle = NULL;
        return 0;
}
```

- Output Result: Success to Connect!



2.1.2.2. Continuous Position Latch

Axis 0 moves continuously, when IN0 senses the signal, then latch axis 0 encoder position value at high speed continuously.



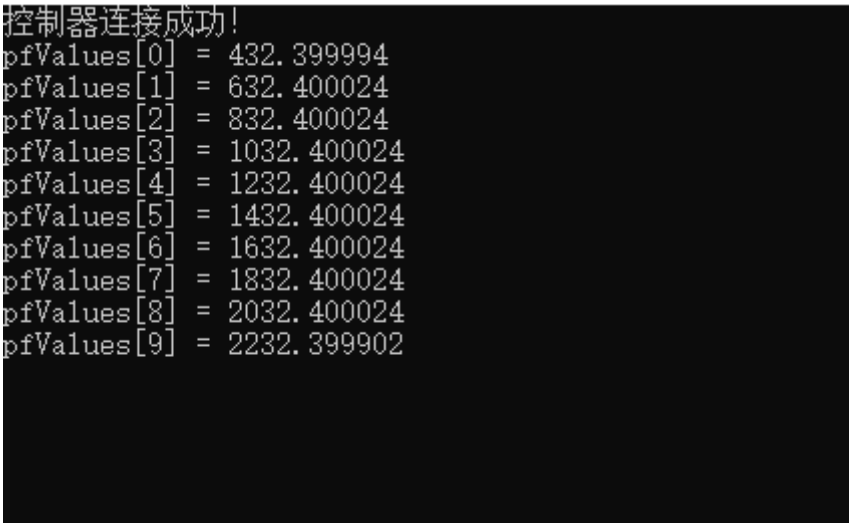

```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
		Sleep(2000);
		exit(0);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	//char *ip_addr = (char *)"127.0.0.1";		//controller IP address
	char *ip_addr = (char *)"192.168.0.11";		//real controller IP address is
192.168.0.11 by default
	ZMC_HANDLE handle = NULL;				//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to Controller!\n");
```

```
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to Controller!\n");

    int iaxis = 0;
    float DposValue = 0;
    float MposValue = 0;
    int mode = 103;          //wait for absolute position of R0 rising edge
    float pfValues[10];
    float pfValue;
    int AType = 1;                          //set axis type
    int UnitValue = 100;                    //set UNITS
    int SpeedValue = 200;                   //set speed
    int AccelValue = 2000;                  //set acceleration
    int DecelValue = 2000;                  //set deceleration

    ret = ZAux_Direct_SetAtype(handle, iaxis, AType);       //set axis 0 type
    commandCheckHandler("ZAux_Direct_SetAtype", ret);

    ZAux_Direct_SetUnits(handle, iaxis, UnitValue);         //set axis 0 pulse amount
    commandCheckHandler("ZAux_Direct_SetUnits", ret);

    ZAux_Direct_SetSpeed(handle, iaxis,SpeedValue);             //set axis 0 speed
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);

    ZAux_Direct_SetAccel(handle, iaxis, AccelValue);        //set axis 0 acceleration
    commandCheckHandler("ZAux_Direct_SetAccel", ret);

    ZAux_Direct_SetDecel(handle, iaxis, DecelValue);        //set axis 0 deceleration
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, iaxis, DposValue);//clear axis 0 dpos to 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);

    ret = ZAux_Direct_SetMpos( handle, iaxis, MposValue);//clear axis 0 mpos to 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ret = ZAux_Direct_Single_Vmove( handle, iaxis, 1);//axis 0 moves forward
    commandCheckHandler("ZAux_Direct_Single_Move", ret);

    ret = ZAux_Direct_CycleRegist( handle, iaxis, mode,0,100);//set continuous latch,
latch  mode: trigger latching when IN0 is in rising edge, latch value saves table(0)-
table(99)
    commandCheckHandler("ZAux_Direct_CycleRegist", ret);
```

```
	ret = ZAux_Direct_MoveWait(handle, 1,"Mpos",iaxis,1,432);//in axis 1 buffer, add the block (here axis 1 buffer is occupied) that executes the next command until axis 0 MPOS is larger than 432
	commandCheckHandler("ZAux_Direct_MoveWait", ret);

	/*after MPOS equals to 432, write rising edge into axis 1 buffer, this routine connects ZMC series controllers, low level is valid level, so the rising edge is formed when IN(0) is set from 1 to 0. (if it is ECI series, it is opposite, from 0 to1)*/
	int i;

	for (i=0;i<10;i++)//trigger 1 latch every 1 second, trigger 10 times
	{
		ret = ZAux_Direct_MoveOp(handle, 1,0,1);//write that OUT0 is set to 1 into axis 1 buffer (here, the used controller has been connected OUT0 to IN0 directly)
		commandCheckHandler("ZAux_Direct_MoveOp ", ret);
		ret = ZAux_Direct_MoveOp(handle, 1,0,0);////write that OUT0 is set to 0 into axis 1 buffer (here, the used controller has been connected OUT0 to IN0 directly)
		commandCheckHandler("ZAux_Direct_MoveOp ", ret);
		ret = ZAux_Direct_MoveDelay(handle, 1,1000);////write that OUT0 is set to 0 into axis 1 buffer (here, the used controller has been connected OUT0 to IN0 directly)
		commandCheckHandler("ZAux_Direct_MoveDelay", ret);
	}

	while (1)
	{
		Sleep(2000);
		ZAux_Direct_GetTable(handle, 0, 1,&pfValue);

		if (pfValue == 10)//exit the cycle when 10 times latches are triggered.
		{
			ret = ZAux_Direct_GetTable(handle, 1, 10,pfValues);
			  commandCheckHandler("ZAux_Direct_GetTable", ret);
			for (i=0;i<10;i++)//trigger 1 latch every 1 second, trigger 10 times
			{
				printf("pfValues[%d] = %f\n", i,pfValues[i]);
			}
			break;
		}
	}

	ret = ZAux_Direct_Single_Cancel( handle, 0, 2);//stop moving
	commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
	Sleep(5000);
	ret = ZAux_Close(handle);    //close the connection
	commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is executed successfully
```

```
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

- Output Result: Success to Connect!



Chapter III Position Comparison Output

3.1.Position Comparison Output Application

3.1.1.Emphasis

1. Soft level position comparison output

Operate relative output according to axis position, if there are several ZAux_Direct_Pswitch operate one output, and need to make sequence as No. order.

2. Precision position output

Only controllers that support the hardware comparison output function can use the precise output function, and the command is ZAux_Direct_SetParam(handle, "AXIS_ZSET", axis No., mode)

The instruction sets whether to enable precise output. Normal output operations need to wait for one controller cycle before they can be executed. The precise output operation can respond within one pulse sent by the motor, which greatly improves the accuracy of the dispensing process. At the same time, the ZAux_Direct_SetParam(handle,"MOVEOP_DELAY",axis No., delay time) function can be used to adjust the response time (advance or delay).

Mode:

Bit 0	1- current motion speed uses interpolation speed by default. 0- current motion speed uses single-axis speed.
Bit 1	1- use ZAux_Direct_MoveOp precision output function. 0- use ZAux_Direct_MoveOp original method.
Bit 4	1- for axis with encoder function, use encoder position ZAux_Direct_MoveOp precision method.

When multi-axis interpolate, set master axis.

Usage:

ZAux_Direct_SetParam(handle,"AXIS_ZSET",0,19) //open axis 0 precision output

3. Hardware level position comparison output

Through setting feedback position table, write into TABLE. In motion, compare values in table to operate corresponding outputs.

There are 1024 HW buffers, and 1024 HW commands can be called continuously.

After calling HW instruction, which won't be affected by following coordinate modify function. And coordinate saved in TABLE through HW commands must be correct when calling. Therefore, it's better to modify coordinate manually to avoid random clashing caused by HW commands and coordinates cyclic auto modification.

Due to coordinate modification in auto and cyclic is not controlled by program, HW is in the front or behind can't be determined, which means coordinates in TABLE also can't be determined. Please pay attention, for bus axes, ZAux_Direct_HwPswitch2 command should be used.

(note: valid in 4xx series products with the latest firmware).

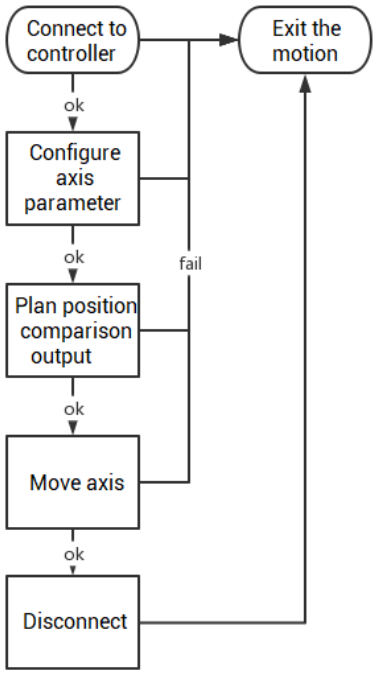
➢ Controller models that support hardware comparison output:

Controller model	Compare channels	Hardware position comparison HwPswitch	Bus hardware position comparison HwPswitch2	Hardware timer HwTimer
ZMC406/432	4 (2 are low speed)	Support	Support / each channel is independent.	
ZMC432N	4	Support		
ZMC412	4	Support		
ZMC464	4	Support		
ZMC430/460	12	Doesn't support		
ZMC420SCAN	8	Doesn't support		
ZMC408SCAN	10	Support		
ZMC408CE	8	Support encoder axis, doesn't support pulse and handwheel.		
ZMC416BE	4	Support encoder axis		
PCI464	2	Support		
ZMC306E	4	Doesn't support	Support / each channel is not independent.	
ZMC304X-HW	8	Doesn't support		
ZMC308B	2	Doesn't support		
ECI2418-HW	4	Doesn't support		
ECI2820-HW	4	Support		
VPLC516E	2	Doesn't support		
VPLC532E	4	Doesn't support	Support / each channel is independent.	

3.1.2.Routine

3.1.2.1. Software Position Comparison Output

Axis 0 moves continuously, output OP0 in the position of 100-200.




```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
		Sleep(2000);
		exit(0);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//simulator IP address
```

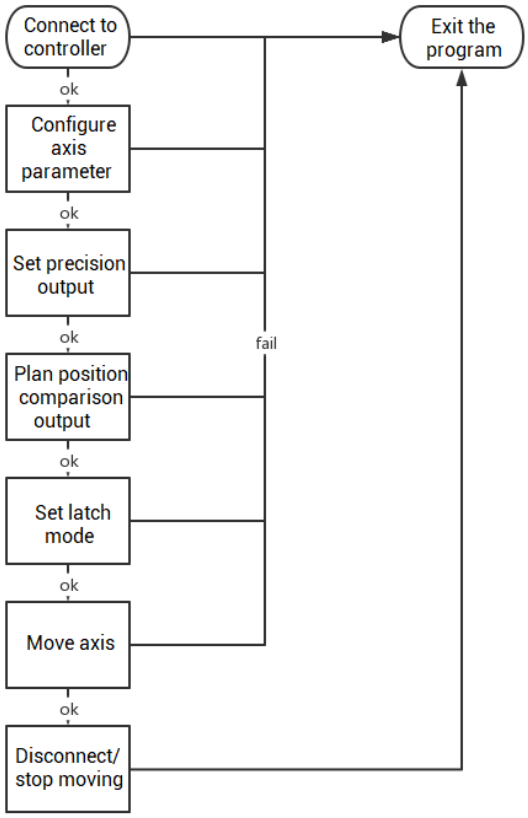
```
    //char *ip_addr = (char *)"192.168.0.11";          //real controller IP address is
192.168.0.11 by default
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to Controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to Controller!\n");

    int iaxis = 0;
    float DposValue = 0;
    float MposValue = 0;
    int AType = 1;                          //set axis type
    int UnitValue = 100;                    //set UNITS
    int SpeedValue = 100;                   //set speed
    int AccelValue = 2000;                  //set acceleration
    int DecelValue = 2000;                  //set deceleration

    ret = ZAux_Direct_SetAtype(handle, iaxis, AType);       //set axis 0 type
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, iaxis, UnitValue);         //set axis 0 pulse amount
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, iaxis,SpeedValue);      //set axis speed
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, iaxis, AccelValue);        //set axis 0 acceleration
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, iaxis, DecelValue);        //set axis 0 deceleration
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, iaxis, DposValue);//clear axis 0 dpos to 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, iaxis, MposValue);//clear axis 0 mpos to 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    float GetValue;
    uint32 piValue;
    ret = ZAux_Direct_Pswitch(handle,0,1,iaxis,0,1,100,200);    //operate the output in
the range of DPOS 0-201 of axis 0
    commandCheckHandler("ZAux_Direct_Pswitch",ret);

    ZAux_Trigger(handle);//open the oscilloscope
    ret = ZAux_Direct_Single_Vmove( handle, iaxis, 1);//axis 0 moves forward
    commandCheckHandler("ZAux_Direct_Single_Move", ret);
```

```
    while (1)
    {
        Sleep(200);
        ret = ZAux_Direct_GetOp(handle,iaxis,&piValue);        //get status of OUT1, 1 means ON, 0 means OFF.
        commandCheckHandler("ZAux_Direct_GetDA",ret);
        printf("OP status piValue=%d\n", piValue);
        ret = ZAux_Direct_GetDpos(handle,iaxis,&GetValue);   //get current position
        printf("position value GetValue=%f\n", GetValue);
        if (GetValue>300)//exit print when position is bigger than 300
        {
            break;
        }
    }

    ret = ZAux_Direct_Single_Cancel( handle, 0, 2);//stop moving
    commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
    Sleep(5000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is executed successfully
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

➢ Waveform:

3.1.2.2. Fly-shooting (ZAux_Direct_HwPswitch)

Assume axis 0 is the conveyor belt axis of continuous motion, it needs to trigger camera to shoot when axis 0 is at the position of 100, 150, 200, 250 and 300, and to latch actual position value of encoder when shooting.



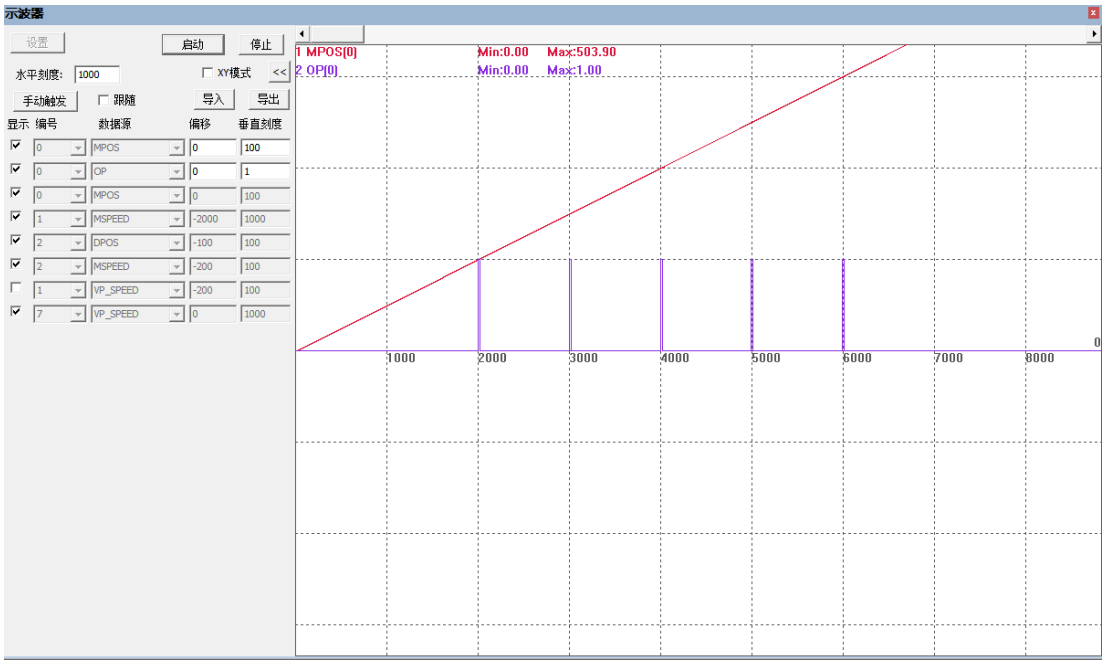

```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
		Sleep(2000);
		exit(0);
	}
}
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    //char *ip_addr = (char *)"127.0.0.1";          //simulator IP address
    char *ip_addr = (char *)"192.168.0.11";          //real controller IP address is
192.168.0.11 by default
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to Controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to Controller!\n");

    int iaxis = 0;
    float DposValue = 0;
    float MposValue = 0;
    int AType = 1;                          //set axis type
    int UnitValue = 100;                    //set UNITS
    int SpeedValue = 50;                    //set speed
    int AccelValue = 2000;                  //set acceleration
    int DecelValue = 2000;                  //set deceleration

    ret = ZAux_Direct_SetAtype(handle, iaxis, AType);       //set axis 0 type
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, iaxis, UnitValue);         //set axis 0 units
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, iaxis,SpeedValue);             //set axis 0 speed
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, iaxis, AccelValue);        //set axis 0 acceleration
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, iaxis, DecelValue);        //set axis 0 deceleration
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, iaxis, DposValue);//clear axis 0 dpos to 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, iaxis, MposValue);//clear axis 0 mpos to 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ret =ZAux_Direct_SetParam(handle,"AXIS_ZSET",0, 19);//open axis 0 precision
output
    commandCheckHandler("ZAux_Direct_SetParam",ret);

    float TableValue[10]={100,101,150,151,200,201,250,251,300,301};//plan bit output
port
```

```
    ret =ZAux_Direct_SetTable(handle,0,10,TableValue );//set Table value
    commandCheckHandler("ZAux_Direct_SetTable",ret);

    ret = ZAux_Direct_HwPswitch(handle,0,2,0,0,0,0);//stop and delete uncompleted comparison point, to prevent unfinished comparison points from being deleted
    commandCheckHandler("ZAux_Direct_HwPswitch",ret);
    ret = ZAux_Direct_HwPswitch(handle,0,1,1,0,0,10);//configure position comparison output
    commandCheckHandler("ZAux_Direct_HwPswitch",ret);

    ret = ZAux_Direct_Regist( handle, iaxis, 4);//set single-time latch, latch mode: trigger latching when IN0 is in falling edge
    commandCheckHandler("ZAux_Direct_Regist", ret);

    ZAux_Trigger(handle);//open oscilloscope
    ret = ZAux_Direct_Single_Vmove( handle, iaxis, 1);//axis 0 moves forward
    commandCheckHandler("ZAux_Direct_Single_Move", ret);

    float GetValue,pfValue;
    int piValue;

    while (1)
    {
        Sleep(200);
        ZAux_Direct_GetMark(handle, iaxis, &piValue);//get whether the latch is triggered

        if (piValue == -1)//trigger latching
        {
            ZAux_Direct_GetRegPos(handle,0,&pfValue);
            printf("pfValue = %f\n", pfValue);
            ret = ZAux_Direct_Regist( handle, iaxis, 4);//set single-time latch, latch mode: trigger latching when IN0 is in falling edge
            commandCheckHandler("ZAux_Direct_Regist", ret);
        }

        ret = ZAux_Direct_GetDpos(handle,iaxis,&GetValue);          //get position value

        if (GetValue>500)//exit when position is larger than 500
        {
            printf("position value GetValue=%f\n", GetValue);
            break;
        }
    }
```

```
        ret = ZAux_Direct_Single_Cancel( handle, 0, 2);//stop moving
        commandCheckHandler("ZAux_Direct_Single_Cancel", ret);
        Sleep(5000);
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
        printf("connection closed!\n");

        handle = NULL;
        return 0;
}
```

➢ Waveform:



3.1.2.3. Fly-Shooting (bind one axis to multiple cameras) (ZAux_Direct_HwPswitch2)

Assume axis 0 is the conveyor belt axis of continuous motion, it needs to trigger camera 1, camera 2, camera 3 and camera 4 to shoot. Camera 1, camera 2, camera 3 and camera 4 corresponds to OUT0, OUT1, OUT2 and OUT3 respectively.

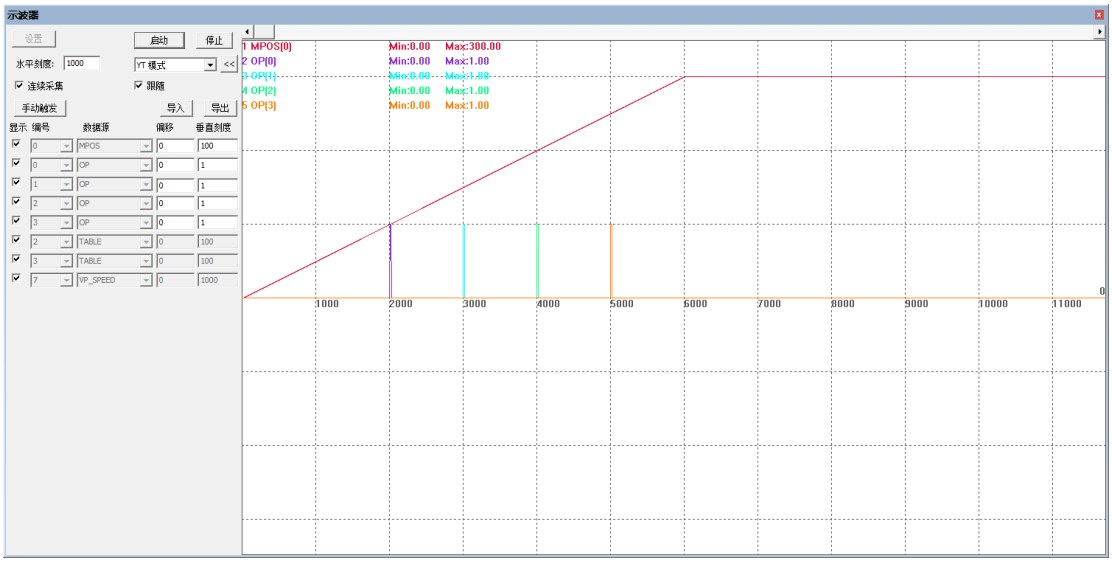
```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
```

```
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
		Sleep(2000);
		exit(0);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	//char *ip_addr = (char *)"127.0.0.1";		//simulator IP address
	char *ip_addr = (char *)"192.168.0.11";		//real controller IP address is
192.168.0.11 by default
	ZMC_HANDLE handle = NULL;				//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to Controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to Controller!\n");

	int iaxis = 0;
	float DposValue = 0;
	float MposValue = 0;
	int AType = 1;					//set axis type
	int UnitValue = 100;				//set UNITS
	int SpeedValue = 50;				//set speed
	int AccelValue = 2000;				//set acceleration
	int DecelValue = 2000;				//set deceleration

	ret = ZAux_Direct_SetAtype(handle, iaxis, AType);	//set axis 0 type
	commandCheckHandler("ZAux_Direct_SetAtype", ret);
	ZAux_Direct_SetUnits(handle, iaxis, UnitValue);		//set axis 0 UNITS
	commandCheckHandler("ZAux_Direct_SetUnits", ret);
	ZAux_Direct_SetSpeed(handle, iaxis,SpeedValue);		//set axis 0 speed
	commandCheckHandler("ZAux_Direct_SetSpeed", ret);
	ZAux_Direct_SetAccel(handle, iaxis, AccelValue);	//set axis 0 acceleration
	commandCheckHandler("ZAux_Direct_SetAccel", ret);
	ZAux_Direct_SetDecel(handle, iaxis, DecelValue);	//set axis 0 deceleration
	commandCheckHandler("ZAux_Direct_SetDecel", ret);
	ret = ZAux_Direct_SetDpos( handle, iaxis, DposValue);//clear axis 0 dpos to 0
	commandCheckHandler("ZAux_Direct_SetDpos", ret);
	ret = ZAux_Direct_SetMpos( handle, iaxis, MposValue);//clear axis 0 mpos to 0
	commandCheckHandler("ZAux_Direct_SetMpos", ret);

	ZAux_Direct_SetParam(handle,"HW_PS2AXISNUM",1,0);//use axis 1 buffer,
compare axis 0 position
```

```
	commandCheckHandler("ZAux_Direct_SetParam",ret);

	ZAux_Direct_SetParam(handle,"HW_PS2AXISNUM",2,0);//use axis 2 buffer, compare axis 0 position
	commandCheckHandler("ZAux_Direct_SetParam",ret);

	ZAux_Direct_SetParam(handle,"HW_PS2AXISNUM",3,0);//use axis 3 buffer, compare axis 0 position
	commandCheckHandler("ZAux_Direct_SetParam",ret);

	uint32 uValue=0;

	ZAux_Direct_SetOutMulti(handle,0,3,&uValue);//initialize OUT0-3 status
	commandCheckHandler("ZAux_Direct_SetOutMulti",ret);

	float TableValue[8]={100,101,150,151,200,201,250,251};//
	ret =ZAux_Direct_SetTable(handle,0,8,TableValue );//set Table value
	commandCheckHandler("ZAux_Direct_SetTable",ret);

	ret = ZAux_Direct_HwPswitch2(handle,0,2,0,0,0,0,0,0);//stop and delete uncompleted comparison point of axis 0, to prevent unfinished comparison points from being deleted
	commandCheckHandler("ZAux_Direct_HwPswitch",ret);

	ret = ZAux_Direct_HwPswitch2(handle,0,1,0,1,0,1,-1,0);//configure position comparison output, axis 0, mode 1, OUT0, output 1, output positions are saved into table(0)-table(1), -1 means no direction used
	commandCheckHandler("ZAux_Direct_HwPswitch",ret);

	ret = ZAux_Direct_HwPswitch2(handle,1,2,0,0,0,0,0,0);//stop and delete uncompleted comparison point of axis 1, to prevent unfinished comparison points from being deleted
	commandCheckHandler("ZAux_Direct_HwPswitch",ret);

	ret = ZAux_Direct_HwPswitch2(handle,1,1,1,1,2,3,-1,0);//configure position comparison output, mode 1, axis 1, OUT1, output 1, output positions are saved into table(2)-table(3), -1 means no direction used
	commandCheckHandler("ZAux_Direct_HwPswitch",ret);

	ret = ZAux_Direct_HwPswitch2(handle,2,2,0,0,0,0,0,0);//stop and delete uncompleted comparison point of axis 2, to prevent unfinished comparison points from being deleted
	commandCheckHandler("ZAux_Direct_HwPswitch",ret);

	ret = ZAux_Direct_HwPswitch2(handle,2,1,2,1,4,5,-1,0);//configure position comparison output, axis 2, mode 1, OUT2, output 1, output positions are saved into table(4)-table(5), -1 means no direction used
	commandCheckHandler("ZAux_Direct_HwPswitch",ret);

	ret = ZAux_Direct_HwPswitch2(handle,3,2,0,0,0,0,0,0);//stop and delete uncompleted comparison point of axis 3, to prevent unfinished comparison points from being deleted
	commandCheckHandler("ZAux_Direct_HwPswitch",ret);

	ret = ZAux_Direct_HwPswitch2(handle,3,1,3,1,6,7,-1,0);//configure position
```

```
comparison output, axis 3, mode 1, OUT1, output 1, output positions are saved into
table(6)-table(7), -1 means no direction used
        commandCheckHandler("ZAux_Direct_HwPswitch",ret);

        ZAux_Trigger(handle);//open the oscilloscope
        ret = ZAux_Direct_Single_Move( handle, iaxis, 300);//axis 0 moves forward
        commandCheckHandler("ZAux_Direct_Single_Move", ret);

        Sleep(5000);
        ret = ZAux_Close(handle);    //close the connection
        commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
        printf("connection closed!\n");

        handle = NULL;
        return 0;
}
```

➢ Waveform:



3.1.2.4. Fly-shooting (2D Comparison Output)

Assuming that axis 0 and axis 1 are moving axes, a plane coordinate system composed of axes 0 and 1 is required, and the camera is triggered to take pictures at positions (100, 100), (200, 200), (300, 300). It is valid in firmware version 4 series 170706 and above.

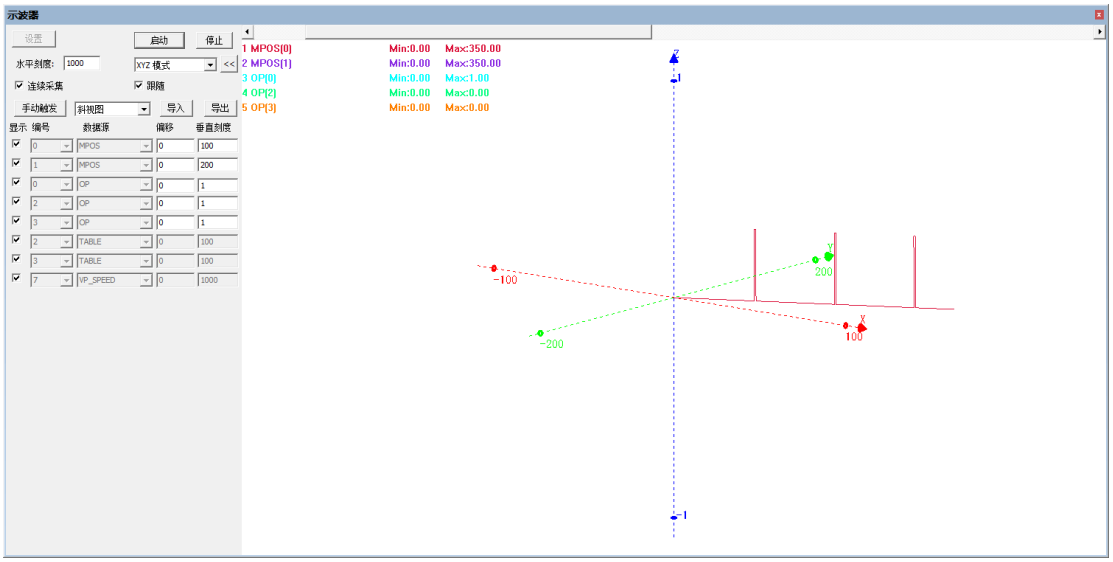
```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
```

```
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
		Sleep(2000);
		exit(0);
	}
}
/*parameter:
	mode		comparison mode of 25, 26 2D
	Opnum		corresponding output
	Opstate		the output state of the first comparison point
	maxerr		pulse deviation of comparison position's each axis left and right
position
	num		the number of comparison points saved in TABLE
	tablepos	Table No. of the first comparison point coordinate

	when it is used together with hwtimer, hwtimer can be adjusted dynamically.
	ModePara1: pulse time
	ModePara2: the number of pulses
	ModePara3: pulse period
	*/
int32 __stdcall ZAux_Direct_HwPswitch2_2D(ZMC_HANDLE handle, int Axisnum,int
Mode,int Opnum , int Opstate,int maxerr,int num, int tablepos, float ModePara1=0, float
ModePara2=0,float ModePara3=0)
{
	if(0 > Axisnum || Axisnum > MAX_AXIS_AUX)
	{
		return  ERR_AUX_PARAERR;
	}
	char  cmdbuff[2048];
	char  cmdbuffAck[2048];
	//generate the command
	switch(Mode)
	{
	case 25:
		sprintf(cmdbuff, "HW_PSWITCH2(%d,%d,%d,%d,%d,%d) AXIS(%d)", Mode,
Opnum, Opstate, maxerr,num,tablepos,Axisnum);
		break;
	case 26:
		sprintf(cmdbuff, "HW_PSWITCH2(%d,%d,%d,%d,%d,%d,%f,%f,%f) AXIS(%d)",
Mode, Opnum, Opstate,
maxerr,num,tablepos,ModePara1,ModePara2,ModePara3,Axisnum);
		break;

	}
	//call the command to execute the function
	return ZAux_Execute(handle, cmdbuff, cmdbuffAck,2048);
}
int _tmain(int argc, _TCHAR* argv[])
{
```

```
    //char *ip_addr = (char *)"127.0.0.1";          //simulator IP address
    char *ip_addr = (char *)"192.168.0.11";          //real controller address is
192.168.0.11 by default
    ZMC_HANDLE handle = NULL;              //link handle
    int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
    if (ERR_SUCCESS != ret)
    {
        printf("Fail to Controller!\n");
        handle = NULL;
        Sleep(2000);
        return -1;
    }
    printf("Success to Controller!\n");

    int iaxis = 0;
    float DposValue = 0;
    float MposValue = 0;
    int AType = 1;                          //set axis type
    int UnitValue = 100;                    //set UNITS
    int SpeedValue = 50;                    //set speed
    int AccelValue = 2000;                  //set acceleration
    int DecelValue = 2000;                  //set deceleration

    ret = ZAux_Direct_SetAtype(handle, iaxis, AType);       //set axis 0 type
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, iaxis, UnitValue);         //set axis 0 UNITS
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, iaxis,SpeedValue);             //set axis 0 speed
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, iaxis, AccelValue);        //set axis 0 acceleration
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, iaxis, DecelValue);        //set axis 0 deceleration
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, 0, 0);//clear axis 0 dpos to 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, 0, 0);//clear axis 0 mpos to 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ret = ZAux_Direct_SetAtype(handle, 1, AType);           //set axis 1 type
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, 1, UnitValue);         //set axis 1 UNITS
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, 1,SpeedValue);         //set axis 1 speed
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, 1, AccelValue);            //set axis 1 acceleration
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, 1, DecelValue);            //set axis 1 deceleration
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, 1, DposValue);//clear axis 1 dpos to 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, 1, MposValue);//clear axis 1 mpos to 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ZAux_Direct_SetOp(handle,0,0);//initialize OUT0 state
    commandCheckHandler("ZAux_Direct_SetOp",ret);
```

```
    float TableValue[12]={100,100,102,102,200,200,202,202,300,300,302,302};//planbit output, each 2 table saves one point
    ret =ZAux_Direct_SetTable(handle,10,12,TableValue );//set Table value
    commandCheckHandler("ZAux_Direct_SetTable",ret);

    ret = ZAux_Direct_HwPswitch2(handle,0,2,0,0,0,0,0,0);//stop and delete uncompleted comparison point, to prevent unfinished comparison points from being deleted
    commandCheckHandler("ZAux_Direct_HwPswitch2",ret);
    ret = ZAux_Direct_HwPswitch2(handle,1,2,0,0,0,0,0,0);//stop and delete uncompleted comparison point, to prevent unfinished comparison points from being deleted
    commandCheckHandler("ZAux_Direct_HwPswitch2",ret);

    ret = ZAux_Direct_HwPswitch2_2D(handle,0,25,0,1,10,6,10);
    commandCheckHandler("ZAux_Direct_HwPswitch2_2D",ret);

    int iaxislist[2]={0,1};
    float dposlist[2]={350,350};

    ZAux_Trigger(handle);//open the oscilloscope
    ret = ZAux_Direct_Move( handle,2,iaxislist , dposlist);//axis 0 moves forward
    commandCheckHandler("ZAux_Direct_Move", ret);

    Sleep(15000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is executed successfully
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

- Waveform:



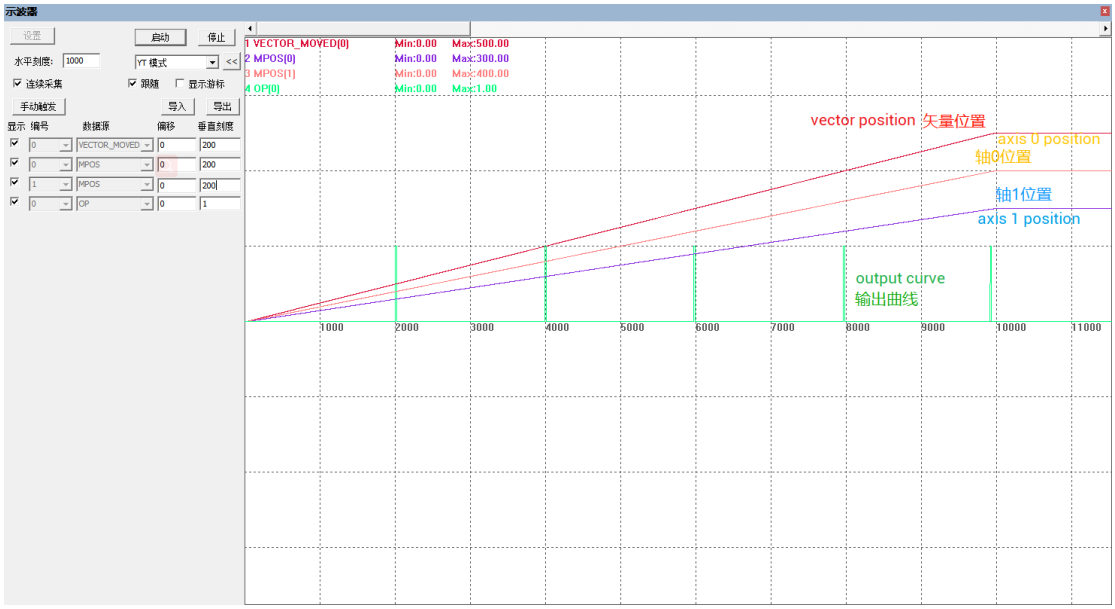
3.1.2.5. Vector Position Comparison Output

```
// test1.cpp: define the entry point of control panel application program
//physical wiring is required, waveform will appear by connecting OUT0 to IN0.

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
		Sleep(2000);
		exit(0);
	}


}

int _tmain(int argc, _TCHAR* argv[])
{
	//char *ip_addr = (char *)"127.0.0.1";		//simulator IP address
	char *ip_addr = (char *)"192.168.0.147";		//real controller address is
192.168.0.11 by default
	ZMC_HANDLE handle = NULL;		//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to Controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to Controller!\n");


	int iaxis[2] = {0,1};
	float DposValue = 0;
	float MposValue = 0;
	int AType = 1;				//set axis type
	int UnitValue = 100;			//set UNITS
	int SpeedValue = 50;			//set speed
	int AccelValue = 2000;			//set acceleration
	int DecelValue = 2000;			//set deceleration

	ret = ZAux_Direct_SetAtype(handle, 0, AType);		//set axis 0 type
	commandCheckHandler("ZAux_Direct_SetAtype", ret);
	ZAux_Direct_SetUnits(handle, 0, UnitValue);		//set axis 0 UNITS
	commandCheckHandler("ZAux_Direct_SetUnits", ret);
	ZAux_Direct_SetSpeed(handle, 0,SpeedValue);		//set axis 0 speed
```

```
        commandCheckHandler("ZAux_Direct_SetSpeed", ret);
        ZAux_Direct_SetAccel(handle, 0, AccelValue);            //set axis 0 acceleration
        commandCheckHandler("ZAux_Direct_SetAccel", ret);
        ZAux_Direct_SetDecel(handle, 0, DecelValue);            //set axis 0 deceleration
        commandCheckHandler("ZAux_Direct_SetDecel", ret);
        ret = ZAux_Direct_SetDpos( handle, 0, DposValue);//clear axis 0 dpos to 0
        commandCheckHandler("ZAux_Direct_SetDpos", ret);
        ret = ZAux_Direct_SetMpos( handle, 0, MposValue);//clear axis 0 mpos to 0
        commandCheckHandler("ZAux_Direct_SetMpos", ret);

        ret = ZAux_Direct_SetAtype(handle, 1, AType);           //set axis 1 type
        commandCheckHandler("ZAux_Direct_SetAtype", ret);
        ZAux_Direct_SetUnits(handle, 1, UnitValue);         //set axis 1 UNITS
        commandCheckHandler("ZAux_Direct_SetUnits", ret);
        ZAux_Direct_SetSpeed(handle, 1,SpeedValue);             //set axis 1 speed
        commandCheckHandler("ZAux_Direct_SetSpeed", ret);
        ZAux_Direct_SetAccel(handle, 1, AccelValue);            //set axis 1 acceleration
        commandCheckHandler("ZAux_Direct_SetAccel", ret);
        ZAux_Direct_SetDecel(handle, 1, DecelValue);            //set axis 1 deceleration
        commandCheckHandler("ZAux_Direct_SetDecel", ret);
        ret = ZAux_Direct_SetDpos( handle, 1, DposValue);//clear axis 1 dpos to 0
        commandCheckHandler("ZAux_Direct_SetDpos", ret);
        ret = ZAux_Direct_SetMpos( handle, 1, MposValue);//clear axis 1 mpos to 0
        commandCheckHandler("ZAux_Direct_SetMpos", ret);

        ret = ZAux_Direct_SetParam( handle, "VECTOR_MOVED",0,0);// 'set vector starting
position
        commandCheckHandler("ZAux_Direct_SetParam", ret);



        float TableValue[10]={100,101,200,201,300,301,400,401,498,499};//
        ret =ZAux_Direct_SetTable(handle,0,10,TableValue );//set Table value
        commandCheckHandler("ZAux_Direct_SetTable",ret);

        ret = ZAux_Direct_HwPswitch2(handle,0,2,0,0,0,0,0,0);//stop and delete
uncompleted comparison point, to prevent unfinished comparison points from being
deleted
        commandCheckHandler("ZAux_Direct_HwPswitch2",ret);

        ret = ZAux_Direct_HwPswitch2(handle,0,3,0,1,0,10,0,0);//configure position
comparison output, master axis is axis 0, mode 3, OUT0, output status 1
        commandCheckHandler("ZAux_Direct_HwPswitch2",ret);


        ZAux_Trigger(handle);//open the oscilloscope
        float pos[2]={300,400};




        ret = ZAux_Direct_Move( handle, 2,iaxis,pos);//axis 0 moves forward
        commandCheckHandler("ZAux_Direct_Move", ret);
```

```
    Sleep(10000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is executed successfully
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

➢ Waveform:



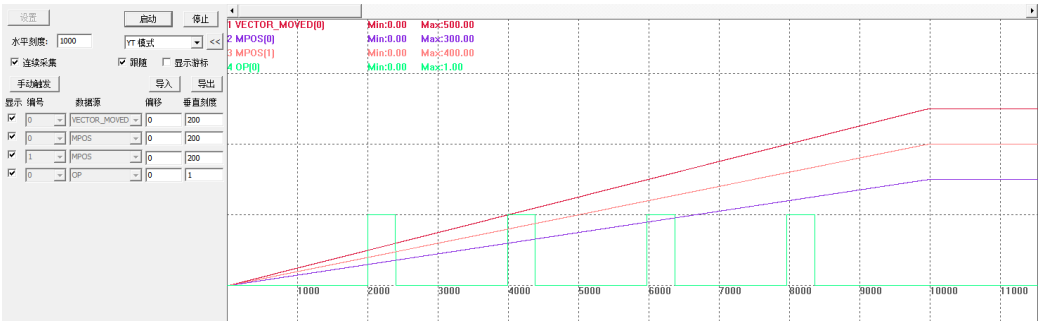
3.1.2.6. Period Comparison Mode (Distance Reset)

```
// test1.cpp: define the entry point of control panel application program
// this routine uses ZMC4 series controllers, and physical wiring is required, waveform will appear by connecting OUT0 to IN0


#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
    if (ret)//it is not 0, fail
    {
        printf("%s fail!return code is %d\n", command, ret);
        Sleep(2000);
```

```
            exit(0);
        }


}

int _tmain(int argc, _TCHAR* argv[])
{
        //char *ip_addr = (char *)"127.0.0.1";          //simulator IP address
        char *ip_addr = (char *)"192.168.0.11";           //real controller IP address is
192.168.0.11 by default
        ZMC_HANDLE handle = NULL;               //link handle
        int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
        if (ERR_SUCCESS != ret)
        {
            printf("Fail to Controller!\n");
            handle = NULL;
            Sleep(2000);
            return -1;
        }
        printf("Success to Controller!\n");


        int iaxis[2] = {0,1};
        float DposValue = 0;
        float MposValue = 0;
        int AType = 1;                              //set axis type
        int UnitValue = 100;                        //set UNITS
        int SpeedValue = 50;                        //set speed
        int AccelValue = 2000;                      //set acceleration
        int DecelValue = 2000;                      //set deceleration

        ret = ZAux_Direct_SetAtype(handle, 0, AType);           //set axis 0 type
        commandCheckHandler("ZAux_Direct_SetAtype", ret);
        ZAux_Direct_SetUnits(handle, 0, UnitValue);         //set axis 0 UNITS
        commandCheckHandler("ZAux_Direct_SetUnits", ret);
        ZAux_Direct_SetSpeed(handle, 0,SpeedValue);         //set axis 0 speed
        commandCheckHandler("ZAux_Direct_SetSpeed", ret);
        ZAux_Direct_SetAccel(handle, 0, AccelValue);            //set axis 0 acceleration
        commandCheckHandler("ZAux_Direct_SetAccel", ret);
        ZAux_Direct_SetDecel(handle, 0, DecelValue);            //set axis 0 deceleration
        commandCheckHandler("ZAux_Direct_SetDecel", ret);
        ret = ZAux_Direct_SetDpos( handle, 0, DposValue);//clear axis 0 dpos to 0
        commandCheckHandler("ZAux_Direct_SetDpos", ret);
        ret = ZAux_Direct_SetMpos( handle, 0, MposValue);//clear axis 0 mpos to 0
        commandCheckHandler("ZAux_Direct_SetMpos", ret);

        ret = ZAux_Direct_SetAtype(handle, 1, AType);           //set axis 1 type
        commandCheckHandler("ZAux_Direct_SetAtype", ret);
        ZAux_Direct_SetUnits(handle, 1, UnitValue);         //set axis 1 UNITS
        commandCheckHandler("ZAux_Direct_SetUnits", ret);
        ZAux_Direct_SetSpeed(handle, 1,SpeedValue);         //set axis 1 speed
        commandCheckHandler("ZAux_Direct_SetSpeed", ret);
        ZAux_Direct_SetAccel(handle, 1, AccelValue);            //set axis 1 acceleration
        commandCheckHandler("ZAux_Direct_SetAccel", ret);
```

```
	ZAux_Direct_SetDecel(handle, 1, DecelValue);		//set axis 1 deceleration
	commandCheckHandler("ZAux_Direct_SetDecel", ret);
	ret = ZAux_Direct_SetDpos( handle, 1, DposValue);//clear axis 1 dpos to 0
	commandCheckHandler("ZAux_Direct_SetDpos", ret);
	ret = ZAux_Direct_SetMpos( handle, 1, MposValue);//clear axis 1 mpos to 0
	commandCheckHandler("ZAux_Direct_SetMpos", ret);

	ret = ZAux_Direct_SetParam( handle, "VECTOR_MOVED",0,0);// 'set vector starting
position
	commandCheckHandler("ZAux_Direct_SetParam", ret);




	ret = ZAux_Direct_HwPswitch2(handle,0,2,0,0,0,0,0,0);//stop and delete
uncompleted comparison point, to prevent unfinished comparison points from being
deleted
	commandCheckHandler("ZAux_Direct_HwPswitch2",ret);
	//start comparing from position 100, compare 4 times, the period distance is 100,
output valid distance 20.

	ret = ZAux_Direct_HwPswitch2(handle,0,5,0,1,100,4,100,20);//configure position
comparison output, master axis is axis 0, mode 5
	commandCheckHandler("ZAux_Direct_HwPswitch2",ret);


	ZAux_Trigger(handle);//open the oscilloscope
	float pos[2]={300,400};




	ret = ZAux_Direct_Move( handle, 2,iaxis,pos);//axis 0 moves forward
	commandCheckHandler("ZAux_Direct_Move", ret);


	Sleep(10000);
	ret = ZAux_Close(handle);    //close the connection
	commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
	printf("connection closed!\n");

	handle = NULL;
	return 0;
}
```

- Waveform:



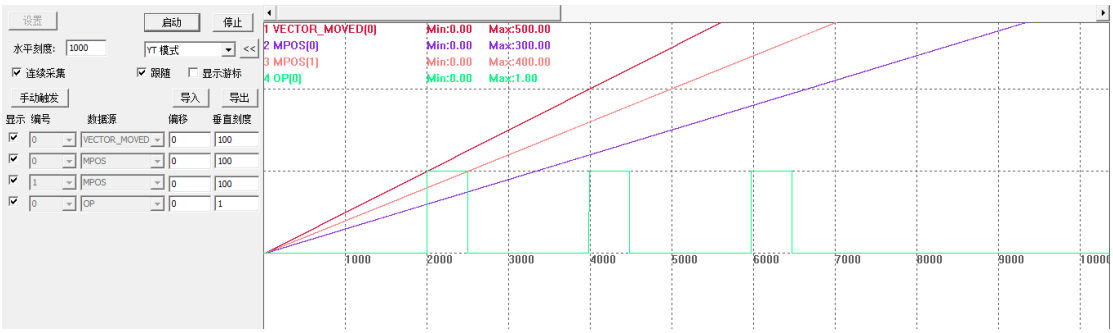
3.1.2.7. Period Comparison Mode (Time Reset)

```
// test1.cpp: define the entry point of control panel application program
// this routine uses ZMC4 series controllers, and physical wiring is required,
waveform will appear by connecting OUT0 to IN0

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
		Sleep(2000);
		exit(0);
	}


}

int _tmain(int argc, _TCHAR* argv[])
{
	//char *ip_addr = (char *)"127.0.0.1";		//simulator IP address
	char *ip_addr = (char *)"192.168.0.147";		//real controller address is
192.168.0.11 by default
	ZMC_HANDLE handle = NULL;			//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to Controller!\n");
		handle = NULL;
		Sleep(2000);
		return -1;
	}
	printf("Success to Controller!\n");


	int iaxis[2] = {0,1};
	float DposValue = 0;
```

```
    float MposValue = 0;
    int AType = 1;                          //set axis type
    int UnitValue = 100;                    //set UNITS
    int SpeedValue = 50;                    //set speed
    int AccelValue = 2000;                  //set acceleration
    int DecelValue = 2000;                  //set deceleration

    ret = ZAux_Direct_SetAtype(handle, 0, AType);          //set axis 0 type
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, 0, UnitValue);        //set axis 0 UNITS
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, 0,SpeedValue);          //set axis 0 speed
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, 0, AccelValue);          //set axis 0 acceleration
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, 0, DecelValue);          //set axis 0 deceleration
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, 0, DposValue);//clear axis 0 dpos to 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, 0, MposValue);//clear axis 0 mpos to 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ret = ZAux_Direct_SetAtype(handle, 1, AType);          //set axis 1 type
    commandCheckHandler("ZAux_Direct_SetAtype", ret);
    ZAux_Direct_SetUnits(handle, 1, UnitValue);        //set axis 1 UNITS
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
    ZAux_Direct_SetSpeed(handle, 1,SpeedValue);          //set axis 1 speed
    commandCheckHandler("ZAux_Direct_SetSpeed", ret);
    ZAux_Direct_SetAccel(handle, 1, AccelValue);          //set axis 1 acceleration
    commandCheckHandler("ZAux_Direct_SetAccel", ret);
    ZAux_Direct_SetDecel(handle, 1, DecelValue);          //set axis 1 deceleration
    commandCheckHandler("ZAux_Direct_SetDecel", ret);
    ret = ZAux_Direct_SetDpos( handle, 1, DposValue);//clear axis 1 dpos to 0
    commandCheckHandler("ZAux_Direct_SetDpos", ret);
    ret = ZAux_Direct_SetMpos( handle, 1, MposValue);//clear axis 1 mpos to 0
    commandCheckHandler("ZAux_Direct_SetMpos", ret);

    ret = ZAux_Direct_SetParam( handle, "VECTOR_MOVED",0,0);// 'set vector
starting position
    commandCheckHandler("ZAux_Direct_SetParam", ret);



    float TableValue[10]={100,101,200,201,300,301,400,401,498,499};//plan bit
output point
    ret =ZAux_Direct_SetTable(handle,0,10,TableValue );//set Table value
    commandCheckHandler("ZAux_Direct_SetTable",ret);

    ret = ZAux_Direct_HwPswitch2(handle,0,2,0,0,0,0,0,0);//stop and delete
uncompleted comparison point, to prevent unfinished comparison points from being
deleted
    commandCheckHandler("ZAux_Direct_HwPswitch2",ret);

    ret = ZAux_Direct_HwPswitch2(handle,0,6,0,1,100,3,100,0);//configure position
comparison output, master axis 0, mode 3, OUT0, output status 1, start to compare
```

```
from position 100, compare 3 times, the period distance is 100, the output valid time is determined according to HW_TIMER.
    commandCheckHandler("ZAux_Direct_HwPswitch2",ret);

    ret = ZAux_Direct_HwTimer(handle,2,1000000,500000,1,0,0);//period is 1s, when output becomes ON, after 500ms, it becomes OFF.
    commandCheckHandler("ZAux_Direct_HwTimer",ret);

    ZAux_Trigger(handle);//open the oscilloscope
    float pos[2]={300,400};




    ret = ZAux_Direct_Move( handle, 2,iaxis,pos);//vector position moves 500
    commandCheckHandler("ZAux_Direct_Move", ret);


    Sleep(10000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is executed successfully
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

➢ Waveform:



Chapter IV PT Motion

4.1.PT Motion

4.1.1.Emphasis

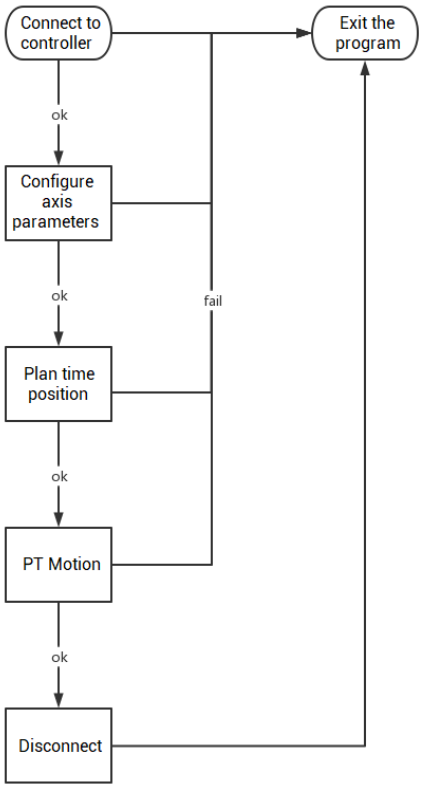
- The drive motor moves a set distance over a period of time.
- The acceleration, speed and deceleration of PT movement are all planned according to the set time and position.
- Generally, the PC calculates the corresponding coordinates every cycle, and then transmits them to the controller.
- Speed during motion=(moving distance/time length)*1000 units/ms.
- Do not move a large distance in a very short time, the pulse frequency will be too high, and the motor will stall. It can be broken down into small segments and sent repeatedly.

Note: when using this command, fast deceleration or deceleration speed need to be configured, otherwise, if an exception occurs, the stop motion command will not stop.

4.1.2.Routine

4.1.2.1. PT Motion

Axis 0 moves from -100 to 0 position in the first 1s, axis 0 moves to 100 from 1s, axis 1 moves to -100, and finally axis 0 and axis 1 return to absolute position 0.

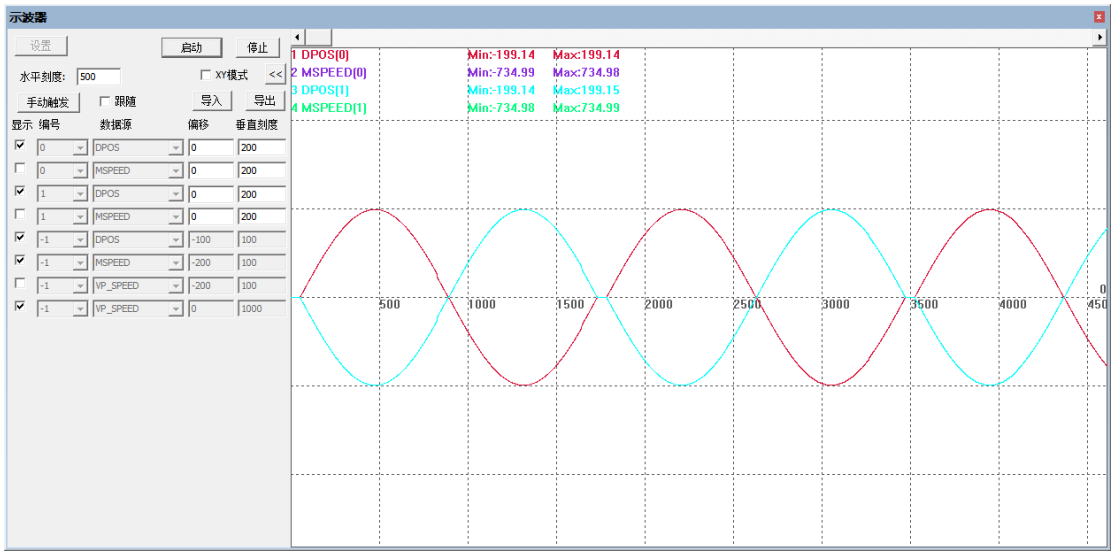



```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
#define PI 3.1415
void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
		Sleep(2000);
		exit(0);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//simulator IP address
	//char *ip_addr = (char *)"192.168.1.11";		//real controller IP address is
192.168.0.11 by default
	ZMC_HANDLE handle = NULL;		//link handle
```

```
int ret = ZAux_OpenEth(ip_addr, &handle);   //connect to controller
if (ERR_SUCCESS != ret)
{
    printf("Fail to Controller!\n");
    handle = NULL;
    Sleep(2000);
    return -1;
}
printf("Success to Controller!\n");

int iaxis[2] = {0,1};

float DposValue = 0;
float MposValue = 0;
int AType = 1;                          //set axis type
int UnitValue = 100;                    //set UNITS

ret = ZAux_Direct_SetAtype(handle, iaxis[0], AType);        //set axis 0 type
commandCheckHandler("ZAux_Direct_SetAtype", ret);
ZAux_Direct_SetUnits(handle, iaxis[0], UnitValue);          //set axis UNITS
commandCheckHandler("ZAux_Direct_SetUnits", ret);
ZAux_Direct_SetFastDec(handle, iaxis[0], 10000);      //set fast deceleration
commandCheckHandler("ZAux_Direct_SetFastDec", ret);

commandCheckHandler("ZAux_Direct_SetDecel", ret);
ret = ZAux_Direct_SetDpos( handle, iaxis[0],0);     //set axis 0 DPOS to -100
commandCheckHandler("ZAux_Direct_SetDpos", ret);
ret = ZAux_Direct_SetMpos( handle, iaxis[0],0);     // set axis 0 MPOS to -100
commandCheckHandler("ZAux_Direct_SetMpos12", ret);
ret = ZAux_Direct_SetAtype(handle, iaxis[1], AType);        //set axis 1 type
commandCheckHandler("ZAux_Direct_SetAtype", ret);
ZAux_Direct_SetUnits(handle, iaxis[1], UnitValue);          //set axis 1 UNITS
commandCheckHandler("ZAux_Direct_SetUnits", ret);
ZAux_Direct_SetFastDec(handle, iaxis[1], 10000);      //set fast deceleration
commandCheckHandler("ZAux_Direct_SetFastDec", ret);

ret = ZAux_Direct_SetDpos( handle, iaxis[1], DposValue);//clear axis 1 dpos to 0
commandCheckHandler("ZAux_Direct_SetDpos", ret);
ret = ZAux_Direct_SetMpos( handle, iaxis[1], MposValue);//clear axis 1 mpos to 0
commandCheckHandler("ZAux_Direct_SetMpos", ret);

uint32 Tims[35];//time planning of relative or absolute PT motion
float DposList[35];//distance planning of relative PT motion
float DposList1[35];//distance planning of relative PT motion
int i;
for ( i=0;i<35;i++)
```

```
    {
        Tims[i]=50;
        DposList[i]=(sin(((2*PI)/34)*i))*200;
        DposList1[i]=(cos(((2*PI)/34)*i+0.5*PI))*200;
    }
    printf("DposList[i]=%f\n",DposList[34]);
    ZAux_Trigger(handle);
    for ( i=0;i<3;i++)
    {
        ret =ZAux_Direct_MultiMovePtAbs(handle, 35,1,
iaxis,Tims,DposList);//35*50ms, move one sine curve with a period of y=sin(x)
        commandCheckHandler("ZAux_Direct_MultiMovePtAbs", ret);
        ret =ZAux_Direct_MultiMovePtAbs(handle, 35,1,
&iaxis[1],Tims,DposList1);////35*50ms, move one sine curve with a period of
y=cos(x+0.5pi)
        commandCheckHandler("ZAux_Direct_SetMpos", ret);
        Sleep(34*50);//send once with a gap of 34*50ms
    }
    Sleep(5000);
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

➢ Waveform:



Chapter V Robot Model

5.1.Robot Model

5.1.1.Emphasis

In Zmotion, the number of robotic arm's motors define the name of robot. For example, if there are 6 motors, it is called 6-joint robot. And the axis that controls motor motion is called the joint axis of robot. And axis that controls robot terminal do linear motion under space coordinate system is called virtual axis.

Joint axis and virtual axis are indicated through **ZAux_Direct_Connreframe** or **ZAux_Direct_Connframe**, controller supports several robots when there is enough axis number.

Program can control motions of joint axes and virtual axes through motion commands (all are valid), but axes of two different types can't move at the same time. When joint axis is moving, virtual axis should be in **ZAux_Direct_Connreframe** mode, then it will automatically indicate the current space coordinate. When virtual axis moves, joint axis should be in **ZAux_Direct_Connframe** mode, then **ZAux_Direct_Connreframe** and **ZAux_Direct_Connframe** commands are dealt as controller connection commands, also can check whether the assigned axis is in relative mode through MTYPE. Robot mode can be cancelled through **ZAux_Direct_Single_Cancel** instruction.

- Inverse Kinematic:

  ZAux_Direct_Connframe means inverse solution motion, and it is used for joint axis. At this time, virtual axis can do linear, circular, spherical motions, and joint axis will automatically move to corresponding joint axis position.

- Forward Kinematic:

  ZAux_Direct_Connreframe means forward solution motion, this is used for virtual axis. At this time, joint axis can do all kinds of motions, but the actual trajectory is not linear or circular. This mode is usually used for manually adjust joint position or upper point homing.

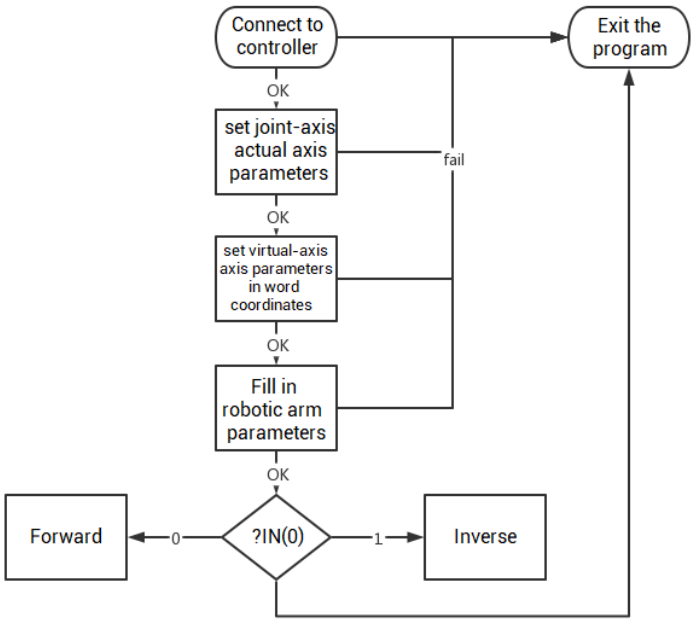
- Three kinds of corner modes for kinematics:

Corner mode of terminal point motion axis under inverse kinematics ZAux_Direct_SetCornerMode(ZMC_HANDLE handle, int iaxis, int iValue)	Connection speed of joint-axis **ZAux_Direct_SetClutchRate**	Motion Mode Description
BIT7 = 1 of iValue value	Invalid	Hybrid mode: speeds and accelerations of virtual axis and joint axis take effect at the same time. In this mode, all virtual axes automatically join in motion all the time, it is invalid in motion superposition of virtual axis. For version above 150718, it supports linear motion, but doesn't support continuous interpolation. Acceleration or deceleration in motion process is invalid.
BIT 7 = 0 of iValue value	0	Smooth mode: joint axis uses itself speed and acceleration, it will do speed planning, and the trajectory will deform in high-speed.
	Non - 0	Force mode: joint axis plans speed and acceleration fully according to virtual axis'.

5.1.2.Routine

5.1.2.1. Build Robotic Arm Model

This routine mainly demonstrates the routine of the standard scara manipulator. Modify the robot type through the value corresponding to the robot type. For the robot type, please refer to the manual. Then manually move the corresponding virtual axis in the ZDevelop software.




```
// test1.cpp: define the entry point of control panel application program
//

#include "stdafx.h"
#include <windows.h>
#include "zmotion.h"
#include "zauxdll2.h"
#include <math.h>
void commandCheckHandler(const char *command, int ret)
{
	if (ret)//it is not 0, fail
	{
		printf("%s fail!return code is %d\n", command, ret);
		Sleep(2000);
		exit(0);
	}
}

int _tmain(int argc, _TCHAR* argv[])
{
	char *ip_addr = (char *)"127.0.0.1";		//simulator IP address
	//char *ip_addr = (char *)"192.168.1.11";		//real controller address is
192.168.0.11 by default
	ZMC_HANDLE handle = NULL;			//link handle
	int ret = ZAux_OpenEth(ip_addr, &handle);	//connect to controller
	if (ERR_SUCCESS != ret)
	{
		printf("Fail to Controller!\n");
		handle = NULL;
```

```
        Sleep(2000);
        return -1;
    }
    printf("Success to Controller!\n");

    int JointAxisList[4] = { 0, 1, 2, 3 };//   //define the type of joint-axis

    //set axis type
    ret = ZAux_Direct_SetAtype(handle, JointAxisList[0], 1);
    commandCheckHandler("ZAux_Direct_SetAtype", ret);

    ret = ZAux_Direct_SetAtype(handle, JointAxisList[1], 1);
    commandCheckHandler("ZAux_Direct_SetAtype", ret);

    ret = ZAux_Direct_SetAtype(handle, JointAxisList[2], 1);
    commandCheckHandler("ZAux_Direct_SetAtype", ret);

    ret = ZAux_Direct_SetAtype(handle, JointAxisList[3], 1);
    commandCheckHandler("ZAux_Direct_SetAtype", ret);

    //set UNITS
    ret = ZAux_Direct_SetUnits(handle, JointAxisList[0], 3600 * 2 / 360);
    commandCheckHandler("ZAux_Direct_SetUnits", ret);

    ret = ZAux_Direct_SetUnits(handle, JointAxisList[1], 3600 * 2 / 360);
    commandCheckHandler("ZAux_Direct_SetUnits", ret);

    ret = ZAux_Direct_SetUnits(handle, JointAxisList[2], 3600 * 2 / 360);
    commandCheckHandler("ZAux_Direct_SetUnits", ret);

    ret = ZAux_Direct_SetUnits(handle, JointAxisList[3], 3600 * 2 / 1.5);
    commandCheckHandler("ZAux_Direct_SetUnits", ret);

    //set DPOS
    ret = ZAux_Direct_SetDpos(handle, JointAxisList[0], 0);
    commandCheckHandler("ZAux_Direct_SetUnits", ret);

    ret = ZAux_Direct_SetDpos(handle, JointAxisList[1], 0);
    commandCheckHandler("ZAux_Direct_SetUnits", ret);

    ret = ZAux_Direct_SetDpos(handle, JointAxisList[2], 0);
    commandCheckHandler("ZAux_Direct_SetUnits", ret);

    ret = ZAux_Direct_SetDpos(handle, JointAxisList[3], 0);
    commandCheckHandler("ZAux_Direct_SetUnits", ret);
```

```
//set acceleration
ret = ZAux_Direct_SetAccel(handle, JointAxisList[0], 1000);
commandCheckHandler("ZAux_Direct_SetAccel", ret);

ret = ZAux_Direct_SetAccel(handle, JointAxisList[1], 1000);
commandCheckHandler("ZAux_Direct_SetAccel", ret);

ret = ZAux_Direct_SetAccel(handle, JointAxisList[2], 1000);
commandCheckHandler("ZAux_Direct_SetAccel", ret);

ret = ZAux_Direct_SetAccel(handle, JointAxisList[3], 1000);
commandCheckHandler("ZAux_Direct_SetAccel", ret);

//set deceleration
ret = ZAux_Direct_SetDecel(handle, JointAxisList[0], 1000);
commandCheckHandler("ZAux_Direct_SetDecel", ret);

ret = ZAux_Direct_SetDecel(handle, JointAxisList[1], 1000);
commandCheckHandler("ZAux_Direct_SetDecel", ret);

ret = ZAux_Direct_SetDecel(handle, JointAxisList[2], 1000);
commandCheckHandler("ZAux_Direct_SetDecel", ret);

ret = ZAux_Direct_SetDecel(handle, JointAxisList[3], 1000);
commandCheckHandler("ZAux_Direct_SetDecel", ret);

//set connection ratio
ret = ZAux_Direct_SetClutchRate(handle, JointAxisList[0], 0);
commandCheckHandler("ZAux_Direct_SetClutchRate", ret);

ret = ZAux_Direct_SetClutchRate(handle, JointAxisList[1], 0);
commandCheckHandler("ZAux_Direct_SetClutchRate", ret);

ret = ZAux_Direct_SetClutchRate(handle, JointAxisList[2], 0);
commandCheckHandler("ZAux_Direct_SetClutchRate", ret);

ret = ZAux_Direct_SetClutchRate(handle, JointAxisList[3], 0);
commandCheckHandler("ZAux_Direct_SetClutchRate", ret);

int VirtualAxisList[4] = { 6, 7, 8, 9 };//       //define the type of virtual-axis
float TableList[7] = { 10, 10, 3600 * 2, 3600 * 2, 3600 * 2, 0, 0 };
float GetTablelist[8];
ret = ZAux_Direct_GetTable(handle, 0, 8, GetTablelist);

//set axis type
ret = ZAux_Direct_SetAtype(handle, VirtualAxisList[0], 0);
```

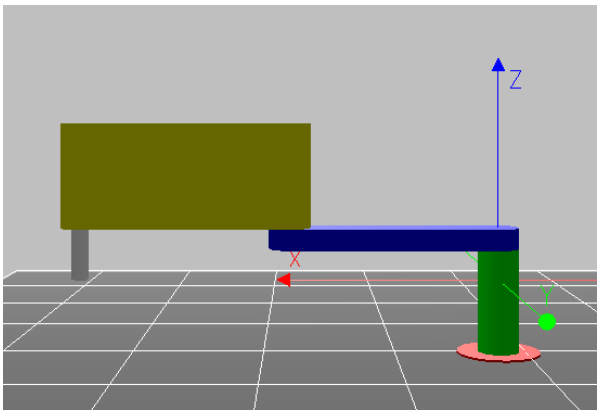
```
commandCheckHandler("ZAux_Direct_SetAtype", ret);

ret = ZAux_Direct_SetAtype(handle, VirtualAxisList[1], 0);
commandCheckHandler("ZAux_Direct_SetAtype", ret);

ret = ZAux_Direct_SetAtype(handle, VirtualAxisList[2], 0);
commandCheckHandler("ZAux_Direct_SetAtype", ret);

ret = ZAux_Direct_SetAtype(handle, VirtualAxisList[3], 0);
commandCheckHandler("ZAux_Direct_SetAtype", ret);

//set system table value
ret = ZAux_Direct_SetTable(handle, 0, 8, TableList);
commandCheckHandler("ZAux_Direct_SetAtype", ret);

//set UNITS
ret = ZAux_Direct_SetUnits(handle, VirtualAxisList[0], 1000);
commandCheckHandler("ZAux_Direct_SetUnits", ret);

ret = ZAux_Direct_SetUnits(handle, VirtualAxisList[1], 1000);
commandCheckHandler("ZAux_Direct_SetUnits", ret);

ret = ZAux_Direct_SetUnits(handle, VirtualAxisList[2], 3600 * 2 / 360);
commandCheckHandler("ZAux_Direct_SetUnits", ret);

ret = ZAux_Direct_SetUnits(handle, VirtualAxisList[3], 1000);
commandCheckHandler("ZAux_Direct_SetUnits", ret);

//set speed
ret = ZAux_Direct_SetSpeed(handle, VirtualAxisList[0], 200);
commandCheckHandler("ZAux_Direct_SetSpeed", ret);

ret = ZAux_Direct_SetSpeed(handle, VirtualAxisList[1], 200);
commandCheckHandler("ZAux_Direct_SetSpeed", ret);

ret = ZAux_Direct_SetSpeed(handle, VirtualAxisList[2], 200);
commandCheckHandler("ZAux_Direct_SetSpeed", ret);

ret = ZAux_Direct_SetSpeed(handle, VirtualAxisList[3], 200);
commandCheckHandler("ZAux_Direct_SetSpeed", ret);

//set acceleration
ret = ZAux_Direct_SetAccel(handle, VirtualAxisList[0], 1000);
commandCheckHandler("ZAux_Direct_SetAccel", ret);

ret = ZAux_Direct_SetAccel(handle, VirtualAxisList[1], 1000);
```

```
    commandCheckHandler("ZAux_Direct_SetAccel", ret);

    ret = ZAux_Direct_SetAccel(handle, VirtualAxisList[2], 1000);
    commandCheckHandler("ZAux_Direct_SetAccel", ret);

    ret = ZAux_Direct_SetAccel(handle, VirtualAxisList[3], 1000);
    commandCheckHandler("ZAux_Direct_SetAccel", ret);

    //set deceleration
    ret = ZAux_Direct_SetDecel(handle, VirtualAxisList[0], 1000);
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Direct_SetDecel(handle, VirtualAxisList[1], 1000);
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Direct_SetDecel(handle, VirtualAxisList[2], 1000);
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    ret = ZAux_Direct_SetDecel(handle, VirtualAxisList[3], 1000);
    commandCheckHandler("ZAux_Direct_SetDecel", ret);

    uint32 GetValue = 0;    //1 means IN is ON, 0 means OFF
    int Jogmaxaxises = 4; //the number of joint-axis
    int frame = 1;       //robot type, 1-standard scara
    int Virmaxaxises = 4;   //the number of virtual-axis
    int   piValue;//define the variable to get the motion buffer, -1 means that there is
no remaining motion buffer at present, 0 means that there are remaining motion
buffers

    //uint32 GetValue = 0;//1 means IN is ON, 0 means OFF
    while (1)
    {
        //ZAux_Direct_GetIn(ZMC_HANDLE handle, int ionum, uint32 *piValue); get
the status value of input
        ret = ZAux_Direct_GetIn(handle, 0, &GetValue);//1 means IN is ON, 0 means
OFF
        commandCheckHandler("ZAux_Direct_GetIn", ret);
        //printf("GetValue 1 means IN is ON, 0 means OFF  GetValue = %d \n",
GetValue);

        if (GetValue == 1)
        {
            ret = ZAux_Direct_Connframe(handle, Jogmaxaxises, JointAxisList,
frame, 0, Virmaxaxises, VirtualAxisList);
            commandCheckHandler("ZAux_Direct_Connreframe", ret);
            printf("open inverse\n");
```

```
                Sleep(1000);

                ret = ZAux_Direct_GetLoaded(handle, 0, &piValue);
                commandCheckHandler("ZAux_Direct_GetLoaded", ret);
                printf("piValue = %d\n", piValue);
                //printf("piValue  means current motion buffers, -1 means no motion
buffer, 0 means remaining buffers piValue = %d\n", piValue);
                //0 means no motion, -1 means with motion
                //wait for the number of tasks in the motion buffer to complete
                while (piValue == -1)
                {
                    ret = ZAux_Direct_GetLoaded(handle, 0, &piValue);
                    commandCheckHandler("ZAux_Direct_GetLoaded", ret);
                    printf("piValue = %d\n", piValue);
                    Sleep(1000);
                    if (piValue == 0)
                    {
                        printf("inverse---while loop piValue = %d\n ", piValue);
                        break;
                    }
                }

            }
            else if (GetValue == 0)
            {
                ret = ZAux_Direct_Connreframe(handle, Virmaxaxises, VirtualAxisList,
frame, 0, Jogmaxaxises, JointAxisList);
                commandCheckHandler("ZAux_Direct_Connreframe", ret);
                printf("open forward\n");
                Sleep(1000);

                ret = ZAux_Direct_GetLoaded(handle, 0, &piValue);
                commandCheckHandler("ZAux_Direct_GetLoaded", ret);
                printf("piValue = %d\n", piValue);

                //wait for the number of tasks in the motion buffer to complete
                while (piValue == -1)
                {
                    if (piValue == 0)
                    {
                        printf("forward---while loop piValue = %d\n ", piValue);
                        break;
                    }
                }
            }
        }
```

```
    ret = ZAux_Close(handle);    //close the connection
    commandCheckHandler("ZAux_Close", ret) ;//judge whether the instruction is
executed successfully
    printf("connection closed!\n");

    handle = NULL;
    return 0;
}
```

- Simulator Model Building:



Chapter VI Instruction Details

<table>
<tr><td>Instruction 1</td><td>ZAux_Direct_GetAddax</td></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_GetAddax (ZMC_HANDLE handle , int iaxis, int * piValue);</td></tr>
<tr><td>Description</td><td>Read superposition axis.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link handle</td></tr><tr><td>iaxis</td><td>Axis No.</td></tr></table></td></tr>
<tr><td>Output parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>piValue</td><td>Read axis' superposition axis No., when there is no superposition axis, it is -1.</td></tr></table></td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Int iValue;
ZAux_Direct_GetAddax(handle, 0, &iValue);
//read axis 0's superposition axis No.</td></tr>
<tr><td>Details</td><td>Conversion relationship: superposition axis motion distance * superposition axis UNITS / superposed axis UNITS = superposed axis motion distance.
Suppose axis A UNITS is 100, axis B UNITS is 50, superposition axis moves 50.
Superpose axis A to axis B, at this time, axis A moves 100, axis B moves 100*100/50=200.
Superpose axis B to axis A, at this time, axis B moves 100, axis A moves 100*50/100=50.
Notes:
1. It can't superpose mutually between axes, that is, when A is superposed into B, B can't be superposed A.
2. It supports serial superposition, that is, motion A is superposed into B, then B is superposed into C.
3. It supports parallel superposition, that is, motion A can be superposed into B and C at the same time.
4. When superposing, speed starts to change from superposed axis, acceleration and deceleration are determined by superposition axis acceleration and deceleration and two axes units ratio.
5. ADDAX is invalid when axis MTYPE is FRAME or REFRAME.</td></tr>
</table>

<table>
<tr><td>Instruction 2</td><td>ZAux_Direct_Single_Addax</td></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_Single_Addax (ZMC_HANDLE handle , int iaxis, int iaddaxis);</td></tr>
<tr><td>Description</td><td>Axis superposition motion, iaddaxis motion is superposed into</td></tr>
</table>

<table>
<tr><td></td><td>iaxis, and the number of pulses is superposed by ADDAX instruction.</td></tr>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link handle</td></tr>
<tr><td>iaxis</td><td>Superposed axis No.</td></tr>
<tr><td>iaddaxis</td><td>Superposition axis No.</td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Motion Superposition</td></tr>
<tr><td>Details</td><td>Conversion relationship: superposition axis motion distance * superposition axis UNITS / superposed axis UNITS = superposed axis motion distance.
Suppose axis A UNITS is 100, axis B UNITS is 50, superposition axis moves 50.
Superpose axis A to axis B, at this time, axis A moves 100, axis B moves 100*100/50=200.
Superpose axis B to axis A, at this time, axis B moves 100, axis A moves 100*50/100=50.
Notes:
1. It can't superpose mutually between axes, that is, when A is superposed into B, B can't be superposed A.
2. It supports serial superposition, that is, motion A is superposed into B, then B is superposed into C.
3. It supports parallel superposition, that is, motion A can be superposed into B and C at the same time.
4. When superposing, speed starts to change from superposed axis, acceleration and deceleration are determined by superposition axis acceleration and deceleration and two axes units ratio.
ADDAX is invalid when axis MTYPE is FRAME or REFRAME.</td></tr>
</table>

<table>
<tr><td>Instruction 3</td><td>ZAux_Direct_Cam</td></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_Cam(ZMC_HANDLE handle,int iaxis, int istartpoint, int iendpoint, float ftablemulti, float fDistance)</td></tr>
<tr><td>Description</td><td>Electronic cam, the CAM command determines axis motion according to data stored in TABLE.</td></tr>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link handle</td></tr>
<tr><td>iaxis</td><td>Axis No.</td></tr>
<tr><td>istartpoint</td><td>Starting point TABLE No., the position where stores the first point.</td></tr>
<tr><td>iendpoint</td><td>Ending point TABLE No.</td></tr>
</table>
</td></tr>
</table>

	ftablemulti	The position multiples this ratio, usually it is the pulse amount.
	fDistance	The distance of reference motion, which is used to calculate total motion time.
Output parameters	/	
Return value	If it is successful, return value is 0, if not, please refer to error codes.	
Example	Electronic Cam	
Details	Note: 2 or more CAM commands can use the same one TABLE data area to operate, and the total motion time is determined by configured speed and the fourth parameter, the actual motion speed is matched automatically according to TABLE trajectory and time. TABLE data needs to be set manually, the first data is the guidance point, which is recommended to be 0. ATBLE data * table multiplier this ratio = sent pulse amounts. Please ensure distance parameter transferred by the instruction * units should be the integer type pulse, otherwise, the floating type will cause small error in motion.	

Instruction 4	**ZAux_Direct_Cambox**	
Original Format	int32 __stdcall ZAux_Direct_Cambox(ZMC_HANDLE handle,int iaxis,int istartpoint, int iendpoint, float ftablemulti, float fDistance, int ilinkaxis, int ioption, float flinkstartpos)	
Description	Electronic cam, the CAM command determines axis motion according to data stored in TABLE. Motion trajectory positions corresponding to Table data values are relative to motion starting point. And following axis motion is done according to reference axis motion.	
Input parameters	Parameter name	Description
	handle	Link handle
	iaxis	Axis No.
	istartpoint	Starting point TABLE No., the position where stores the first point.
	iendpoint	Ending point TABLE No.
	ftablemulti	The position multiples this ratio, usually it is the pulse amount.
	fDistance	The distance of reference motion, which is used to calculate total motion time.
	Ilinkaxis	Reference axis.
	ioption	The connection method of reference axis. Different binary values represent different meanings.

		Bit 0: when the reference axis latch signal (Regist) event is triggered, the current axis and the reference axis start to perform joint motion. Bit 1: when the reference axis moves to the set absolute position, the current axis and the reference axis start to move in conjunction. Bit 2: automatically repeats continuous bi-directional operation. (repeat can be canceled by setting REP_OPTION=1). Bit 5: connected only for positive movement of the reference axis. Bit8: when the reference axis MARKB signal event is triggered, the current axis and the reference axis start to connect and move, and the latched axis number is the axis number of the reference axis, which requires the latest firmware.	
	Flinkstartpos	When ioption parameter bit 1 is set to 1, which means the connection starts when reference axis is at the absolute position value.	
Output parameters	/		
Return value	If it is successful, return value is 0, if not, please refer to error codes.		
Example	Electronic Cam Synchronous Motion		
Details	Note: 2 or more CAMBOX commands can use the same one TABLE data area to operate, and the total motion time is determined by configured speed and the fourth parameter, the actual motion speed is matched automatically according to TABLE trajectory and time. TABLE data needs to be set manually, the first data is the guidance point, which is recommended to be 0. ATBLE data * table multiplier this ratio = sent pulse amounts. Please ensure distance parameter transferred by the instruction * units should be the integer type pulse, otherwise, the floating type will cause small error in motion.		

Instruction 5	**ZAux_Direct_Movelink**
Original Format	int32 __stdcall ZAux_Direct_Movelink(ZMC_HANDLE handle, int iaxis, float fDistance, float fLinkDis, float fLinkAcc, float fLinkDec,int iLinkaxis, int ioption, float flinkstartpos)

<table>
<tr><td>Description</td><td>Automatic cam. It is custom cam motion, which is with acceleration and deceleration that can be set.</td></tr>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link handle</td></tr>
<tr><td>iaxis</td><td>Axis No.</td></tr>
<tr><td>fDistance</td><td>The distance that is moved by current axis when in connection, using user unit.</td></tr>
<tr><td>fLinkDis</td><td>The positive distance that is moved by reference axis when in connection, the unit is units.</td></tr>
<tr><td>fLinkAcc</td><td>The positive distance that is moved by reference axis when current axis is in acceleration, the unit is units.</td></tr>
<tr><td>fLinkDec</td><td>The positive distance that is moved by reference axis when current axis is in deceleration, the unit is units.</td></tr>
<tr><td>iLinkaxis</td><td>Reference axis axis No.</td></tr>
<tr><td>ioption</td><td>The axis No. of reference axis.
Different binary values represent different meanings.
Bit 0: when the reference axis latch signal (Regist) event is triggered, the current axis and the reference axis start to perform joint motion.
Bit 1: when the reference axis moves to the set absolute position, the current axis and the reference axis start to move in conjunction.
Bit 2: automatically repeats continuous bi-directional operation. (repeat can be canceled by setting REP_OPTION=1).
Bit 5: connected only for positive movement of the reference axis.
Bit8: when the reference axis MARKB signal event is triggered, the current axis and the reference axis start to connect and move, and the latched axis number is the axis number of the reference axis, which requires the latest firmware.</td></tr>
<tr><td>Flinkstartpos</td><td>When ioption parameter bit 1 is set to 1, which means the connection starts when reference axis is at the absolute position value.</td></tr>
</table>
</td></tr>
</table>

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Fly-shearing
Details	1. The axis that is connected is the reference axis, the axis that connects is the following axis. 2. There are 3 stages of connecting axis distance for reference axis motion, they are acceleration, constant speed and deceleration. 3. In acceleration and deceleration stages, basic axis distance must be two times of following axis motion distance for matching speed. 4. Please ensure distance parameter transferred by the instruction * units should be the integer type pulse, otherwise, the floating type will cause small error in motion.

Instruction 6	**ZAux_Direct_Moveslink**
Original Format	int32 __stdcall ZAux_Direct_Moveslink(ZMC_HANDLE handle, int iaxis,float fDistance, float fLinkDis, float startsp, float endsp,int iLinkaxis, int ioption, float flinkstartpos)
Description	Automatic cam 2. It is used for custom cam motion, and this motion automatically plans middle curves, no need to calculate the cam table.
Input parameters	

Parameter name	Description
handle	Link mark.
iaxis	Axis No.
fDistance	The distance that is moved by current axis when in connection, using user unit.
fLinkDis	The positive distance that is moved by reference axis when in connection, the unit is units.
startsp	The speed ratio between following axis and reference axis when opening, the units is units/units, negative value means following axis moves in negative direction.
endsp	The speed ratio between following axis and reference axis when ending, the units is units/units, negative value means following axis moves in negative direction. Ps: when start sp = end sp = fdistance/flink dis, it is constant speed.
iLinkaxis	Reference axis axis No.

	ioption	The axis No. of reference axis. Different binary values represent different meanings. Bit 0: when the reference axis latch signal (Regist) event is triggered, the current axis and the reference axis start to perform joint motion. Bit 1: when the reference axis moves to the set absolute position, the current axis and the reference axis start to move in conjunction. Bit 2: automatically repeats continuous bi-directional operation. (repeat can be canceled by setting REP_OPTION=1). Bit 5: connected only for positive movement of the reference axis. Bit8: when the reference axis MARKB signal event is triggered, the current axis and the reference axis start to connect and move, and the latched axis number is the axis number of the reference axis, which requires the latest firmware.	
	Flinkstartpos	When ioption parameter bit 1 is set to 1, which means the connection starts when reference axis is at the absolute position value.	
Output parameters	/		
Return value	If it is successful, return value is 0, if not, please refer to error codes.		
Example	Electronic Cam Synchronous Motion		
Details	1. The axis that is connected is the reference axis, the axis that connects is the following axis. 2. There are 3 stages of connecting axis distance for reference axis motion, they are acceleration, constant speed and deceleration. 3. In acceleration and deceleration stages, basic axis distance must be two times of following axis motion distance for matching speed. 4. Please ensure distance parameter transferred by the instruction * units should be the integer type pulse, otherwise, the floating type will cause small error in motion.		

Instruction 7	**ZAux_Direct_Regist**

<table>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_Regist(ZMC_HANDLE handle,int iaxis, int imode)</td></tr>
<tr><td>Description</td><td>Single position latch.</td></tr>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link mark.</td></tr>
<tr><td>iaxis</td><td>Axis No.</td></tr>
<tr><td>fDistance</td><td>The distance that is moved by current axis when in connection, using user unit.</td></tr>
<tr><td>imode</td><td>
<table>
<tr><td>value</td><td>Description</td></tr>
<tr><td>1</td><td>Save absolute position in RegPos when meets rising edge of Z pulse</td></tr>
<tr><td>2</td><td>Save absolute position in RegPos when meets falling edge of Z pulse</td></tr>
<tr><td>3</td><td>Save absolute position in RegPos when meets rising edge of R0 signal</td></tr>
<tr><td>4</td><td>Save absolute position in RegPos when meets falling edge of R0 signal</td></tr>
<tr><td>6</td><td>Save absolute position in RegPos when meets rising edge of R0 signal, and save absolute position in RegPosB when meets rising edge of Z signal.</td></tr>
<tr><td>7</td><td>Save absolute position in RegPos when meets rising edge of R0 signal, and save absolute position in RegPosB when meets falling edge of Z signal.</td></tr>
<tr><td>8</td><td>Save absolute position in RegPos when meets falling edge of R0 signal, and save absolute position in RegPosB when meets rising edge of Z signal.</td></tr>
<tr><td>9</td><td>Save absolute position in RegPos when meets falling edge of R0 signal, and save absolute position in RegPosB when meets falling edge of Z signal.</td></tr>
</table>
</td></tr>
</table>
</td></tr>
</table>

		10	Save absolute position in RegPos when meets rising edge of R0 signal, and save absolute position in RegPosB when meets rising edge of R1.
		11	Save absolute position in RegPos when meets rising edge of R0 signal, and save absolute position in RegPosB when meets falling edge of R1.
		12	Save absolute position in RegPos when meets falling edge of R0 signal, and save absolute position in RegPosB when meets rising edge of R1.
		13	Save absolute position in RegPos when meets falling edge of R0 signal, and save absolute position in RegPosB when meets rising edge of R1.
		14	Save absolute position in RegPosB when meets rising edge of R1 signal.
		After mode 14, only 150804 firmware supports, each latch channel is independent, and 4-channel latch is supported)	
		15	Save absolute position in RegPosB when meets falling edge of R1 signal
		16	Save absolute position in RegPosB when meets rising edge of Z signal
		17	Save absolute position in RegPosB when meets falling edge of Z signal
		18	Save absolute position in RegPosC when meets rising edge of R2 signal
		19	Save absolute position in RegPosC when meets falling edge of R2 signal
		20	Save absolute position in RegPosD when meets rising edge of R3 signal

		21	Save absolute position in RegPosD when meets falling edge of R3 signal	

Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Position Latch
Details	1. Rising and falling edges refer to the internal state of the controller. If it is set as a rising edge trigger, the latch will be triggered at the moment when the external input port changes from the conduction state to the off state. If it is set as a falling edge trigger, the latch will be triggered at the moment when the external input port changes from the off state to the on state, whether to set it as a rising edge or a falling edge trigger, it depends on the actual needs of the external input signal transition. The pulse axis type generally adopts R0, R1, and Z pulse latches, and the bus axis type adopts R2, R3 latches. 2. Support encoder axis latching, the latest firmware of 4 series and above controllers supports virtual axis and pulse axis latching. 3. EtherCAT supports driver latching. At this time, the driver IO points are used to realize the latching. See the command syntax for the specific mode. 4. Rtex only supports controller latching. 5. 4 series and above controllers support 4-channel latching. 6. ZMC432 has 2 latch input ports, ZMC432N has 4 latch inputs, ZMC412 has 8 latch inputs. 7. It supports simultaneous use of EtherCAT driver latch and controller latch, and requires 4 latch channel functions. 8. The 4 channels refer to MARK, MARKB, MARKC, and MARKD, and the latch channel corresponding to the latch input port is specified through REG_INPUTS. 9. When the latch occurs, the axis state MARK will be set to ON, and the latched position will be stored in the parameter REG_POS. 10. Each axis has input signals R0, R1, and EZ signals that can be used for the latch function. When using two signal latches, the second signal latch uses MARKB and REG_POSB. 11. R0, R1 input generally correspond to input ports 0 and 1, please refer to the general input chapter in the hardware manual of the controller for details.

Instruction 8	**ZAux_Direct_CycleRegist**

Original Format	int32 __stdcall ZAux_Direct_CycleRegist(ZMC_HANDLE handle,int iaxis, int imode,int iTabStart,int iTabNum)
Description	Position continuous latch.

Input parameters

Parameter name	Description
handle	Link mark.
iaxis	Axis No.
imode	

value	Description
101	Save absolute position in RegPos when meets rising edge of Z pulse
102	Save absolute position in RegPos when meets falling edge of Z pulse
103	Save absolute position in RegPos when meets rising edge of R0 signal
104	Save absolute position in RegPos when meets falling edge of R0 signal
114	Save absolute position in RegPosB when meets rising edge of R1 signal.
115	Save absolute position in RegPosB when meets falling edge of R1 signal
116	Save absolute position in RegPosB when meets rising edge of Z signal
117	Save absolute position in RegPosB when meets falling edge of Z signal
123	Save absolute position in RegPosB when meets rising edge of R0 signal
124	Save absolute position in RegPosB when meets falling edge of R0 signal
133	Save absolute position in RegPos when meets falling edge of R0, next time switch into falling edge, switch in turn.
134	Save absolute position in RegPos when meets rising edge of R0, next time switch

Save absolute position
RegPosB when meets
edge of R0 signal

<table>
<tr><td></td><td>
<table>
<tr><td></td><td>
<table>
<tr><td></td><td>into rising edge, switch in turn.</td></tr>
<tr><td>135</td><td>Save absolute position in RegPos when meets falling edge of R1, next time switch into falling edge, switch in turn.</td></tr>
<tr><td>136</td><td>Save absolute position in RegPos when meets rising edge of R1, next time switch into rising edge, switch in turn.</td></tr>
</table>
</td></tr>
<tr><td>iTabStart</td><td>The table position where the content of the continuous latch is stored, the first table element stores the number of latches, and the coordinates of the latches are stored later, the maximum number of saved = numes-1, and it is written in a loop when overflowing</td></tr>
<tr><td>iTabNum</td><td>The number of occupied table</td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Continuous position latch</td></tr>
<tr><td>Details</td><td>Latch results are stored in TABLE.
Latch two-channel continuously and respectively, which means it can achieve continuous latch for upper and bottom edge.
ECI: valid in 20150829 firmware and above.
ZMC4XX series controller: valid in 20170523 firmware and above.</td></tr>
</table>

<table>
<tr><td>Instruction 9</td><td>ZAux_Direct_GetCloseWin</td></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_GetCloseWin(ZMC_HANDLE handle,int iaxis, float * pfValue);</td></tr>
<tr><td>Description</td><td>Read end coordinate range point of latch trigger. Reserved</td></tr>
<tr><td>Input parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>handle</td><td>Link mark.</td></tr>
<tr><td>iaxis</td><td>Axis No.</td></tr>
</table>
</td></tr>
<tr><td>Output parameters</td><td>
<table>
<tr><td>Parameter name</td><td>Description</td></tr>
<tr><td>pfValue</td><td>Returned range value</td></tr>
</table>
</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>/</td></tr>
<tr><td>Details</td><td>Reversed function</td></tr>
</table>

Instruction 10	ZAux_Direct_SetCloseWin
Original Format	int32 __stdcall ZAux_Direct_SetCloseWin(ZMC_HANDLE handle,int iaxis, float fValue);
Description	Set end coordinate range point of latch trigger. Reserved
Input parameters	Parameter name: Description handle: Link mark. iaxis: Axis No. iValue: Range value that is set.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	Reversed function

Instruction 11	ZAux_Direct_GetRegPosB
Original Format	int32 __stdcall ZAux_Direct_GetRegPosB (ZMC_HANDLE handle, int iaxis, float *pfValue);
Description	Return to measurement position (ZAux_Direct_GetMpos) of latch register 2 (ZAux_Direct_GetMarkB). Unit: pulse amount
Input parameters	Parameter name: Description handle: Link mark. iaxis: Axis No.
Output parameters	Parameter name: Description pfValue: Returned coordinate position
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	Only encoder can be latched by default. ZMC4XX series controllers support pulse and virtual latching.

Instruction 12	ZAux_Direct_GetRegPos
Original Format	int32 __stdcall ZAux_Direct_GetRegPosB (ZMC_HANDLE handle, int iaxis, float *pfValue);
Description	Return to measurement position (ZAux_Direct_GetMpos) of latch register 1 (ZAux_Direct_GetMark).
Input parameters	Parameter name: Description handle: Link mark. iaxis: Axis No.

Output parameters	Parameter name: pfValue — Description: Latched coordinate position
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Position Latch
Details	Only encoder can be latched by default. ZMC4XX series controllers support pulse and virtual latching.

Instruction 13	**ZAux_Direct_GetOpenWin**
Original Format	int32 __stdcall ZAux_Direct_GetOpenWin (ZMC_HANDLE handle, int iaxis, float *pfValue);
Description	Read end coordinate range point of latch trigger. Reserved
Input parameters	Parameter name: handle — Description: Link mark. Parameter name: iaxis — Description: Axis No.
Output parameters	Parameter name: pfValue — Description: Returned range value
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	Reversed function

Instruction 14	**ZAux_Direct_GetOpenWin**
Original Format	int32 __stdcall ZAux_Direct_GetOpenWin (ZMC_HANDLE handle, int iaxis, float *pfValue);
Description	Set end coordinate range point of latch trigger. Reserved
Input parameters	Parameter name: handle — Description: Link mark. Parameter name: iaxis — Description: Axis No. Parameter name: iValue — Description: The range value that is set.
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	Reversed function

Instruction 15	**ZAux_Direct_GetMark**
Original Format	iint32 __stdcall ZAux_Direct_GetMark(ZMC_HANDLE handle,int iaxis, int *piValue);

Description	Read returned state of latch event 0.
Input parameters	Parameter name: handle — Link mark. Parameter name: iaxis — Axis No.
Output parameters	Parameter name: piValue — Mode configuration, 0-doesn't occur, 1-occur.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Position latch
Details	1. When latch instruction is executing, piValue becomes true, that is return -1. When completed, it becomes false, return 0. 2. Corresponding latch channel R0. 3. Controller IN0 corresponds to latch channel R0 by default, IN1 corresponds to latch channel R1.

Instruction 16	**ZAux_Direct_GetMarkB**
Original Format	int32 __stdcall ZAux_Direct_GetMarkB(ZMC_HANDLE handle,int iaxis, int *piValue);
Description	Read returned state of latch event 1.
Input parameters	Parameter name: handle — Link mark. Parameter name: iaxis — Axis No.
Output parameters	Parameter name: piValue — Mode configuration, 0-doesn't occur, 1-occur.
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Position latch
Details	1. When latch instruction is executing, piValue becomes true, that is return -1. When completed, it becomes false, return 0. 2. Corresponding latch channel R1. 3. Controller IN0 corresponds to latch channel R0 by default, IN1 corresponds to latch channel R1.

Instruction 17	**ZAux_Direct_Pswitch**
Original Format	int32 __stdcall ZAux_Direct_Pswitch(ZMC_HANDLE handle,int num, int enable,int axisnum,int outnum,int outstate,float setpos,float resetpos)
Description	Software position comparison output.
Input parameters	Parameter name: handle — Link mark. Parameter name: num — Software comparer No., different modes are with different comparers, for

		example, ZMC1XX series has 16.
	enable	Comparer enable operation, 0-OFF, 1-ON.
	axisnum	Motion axis No. of comparison output.
	outnum	Output No. to be operated.
	outstate	Output state, 1-ON, 0-OFF.
	setpos	Starting position of comparison output
	resetpos	Ending position of comparison output
Output parameters	/	
Return value	If it is successful, return value is 0, if not, please refer to error codes.	
Example	Software position comparison	
Details	If multiple PSWITCH operate same one output, No. sequence should be arranged together. When using pulse type motor, it is comparison measurement position (MPOS) when ATYPE is 4. Default is 1/7, it compares the command position (DPOS).	

Instruction 18	**ZAux_Direct_HwPswitch**	
Original Format	int32 __stdcall ZAux_Direct_HwPswitch(ZMC_HANDLE handle,int Axisnum,int Mode, int Direction, int Reserve, int Tablestart, int Tableend)	
Description	Hardware position comparison output, ZMC4XX series products support pulse axis and encoder axis.	
Input parameters	Parameter name	Description
	handle	Link mark.
	Axisnum	Comparison output axis No.
	Mode	Comparer operation, 1-open comparer, 2-stop and delete uncompleted point
	Direction	Comparison direction, 0-negative, 1-positive.
	Reserve	Reserved
	Tablestart	TABLE starting address
	Tableend	TABLE ending address
Output parameters	/	
Return value	If it is successful, return value is 0, if not, please refer to error codes.	
Example	Fly-shooting application	
Details	1. If HW_PSWITCH has not compared all the points, be sure to set the mode value to 2, stop and delete the unfinished comparison points through the HW_PSWITCH(2) command, otherwise the output channel will not work properly.	

	2. The default axes 0-5 correspond to output ports 0 1 2 3 0 1 respectively. A total of 4 compare output ports. 3. Two HW_PSWITCH instructions can be called continuously, and the number of instructions that can be called can be obtained through the function method. 4. Each comparison point trigger will cause the level of the current output port to flip. 5. The number of HW buffers is 1024, and 1024 HW instructions can be called continuously. 6. After the HW command is called, it will not be affected by the subsequent coordinate modification function. The coordinates stored in the HW command TABLE must be correct when it is called. Therefore, try to manually modify the coordinates to avoid random conflicts between the HW command and the automatic modification of the coordinate cycle. 7. Because the coordinates of the automatic cycle modification are not controlled by the program, it is impossible to determine whether it is in front of or behind the HW, so the coordinates in the TABLE cannot be determined. 8. This command only supports the pulse axis hardware position comparison output, please use the HW_PSWITCH2 command for the bus axis. 9. When using a pulse-type motor, only when the ATYPE is 4 is the comparison feedback position (MPOS), and the default factory ATYPE is 1 or 7, and the comparison is the command position (DPOS).

Instruction 19	**ZAux_Direct_HwPswitch2**
Original Format	int32 __stdcall ZAux_Direct_HwPswitch2(ZMC_HANDLE handle,int Axisnum,int Mode, int Opnum, int Opstate, float ModeParal, float ModePara2, float Mode Para3 , float ModePara4)
Description	Hardware position comparison output2, ZMC4XX series products with firmware 20170513 or above version support. And it is valid in ZMC306E/ZMC306N.
Input parameters	

Parameter name	Description
handle	Link mark.
Axisnum	Comparison output axis No.
Mode	Mode 1 - enable comparator ModePara1 = TABLE No. where the coordinates of the first comparison point are located ModePara2 = TABLE No. where the coordinates of the last comparison point are located ModePara3= The direction of the first

		point, 0-negative, 1-positive, -1-do not use the direction ModePara4 = Reserved Mode 2 - stop and delete incomplete comparison points ModePara1 = Reserved ModePara2 = Reserved ModePara3 = Reserved ModePara4 = Reserved Mode 3 - vector comparison method ModePara1 = TABLE No. where the coordinates of the first comparison point are located ModePara2 = TABLE No. where the coordinates of the last comparison point are located ModePara3= Reserved ModePara4 = Reserved Mode 4 - vector comparison method single comparison point ModePara1 = compare point coordinates ModePara2 = Reserved ModePara3 = Reserved ModePara4 = Reserved Mode 5 - Vector comparison mode periodic pulse mode ModePara1 = comparison point coordinates ModePara2 = repeat cycle, compare twice in one cycle, first output valid state, then output invalid state ModePara3 = periodic distance, output Opstate for each distance, output valid state distance (ModePara4) and restore to invalid state ModePara4=Output valid state distance, (ModePara1-ModePara4) bit invalid state distance Mode 6 - vector comparison mode periodic mode, this mode is used	

<table>
<tr><td></td><td><table>
<tr><td></td><td>together with HW_TIMER
ModePara1 = compare point coordinates
ModePara2 = repeat cycle, a coordinate is only compared once
ModePara3 = cycle distance, output every time this distance
ModePara4 = Reserved</td></tr>
<tr><td>Opnum</td><td>Output No., 4 series, out0 – hardware position comparison output.</td></tr>
<tr><td>Opstate</td><td>Output state of the first comparison point. 0-OFF, 1-ON.</td></tr>
<tr><td>ModePara1</td><td>Multiple function parameter.</td></tr>
<tr><td>ModePara2</td><td>Multiple function parameter.</td></tr>
<tr><td>ModePara3</td><td>Multiple function parameter.</td></tr>
<tr><td>ModePara4</td><td>Multiple function parameter.</td></tr>
</table></td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>Fly-shooting application</td></tr>
<tr><td>Details</td><td>1. 4 series products have 4 comparison output ports, it can choose different comparison output ports, generally OUT0/1/2/3 ports.
2. When comparing the main axis that is with encoder input, the encoder position is automatically used to trigger, and the MOVEOP_DELAY parameter can be used to adjust the exact time of output.
3. The effect of different bus drivers may be different, and it can also be adjusted by the MOVEOP_DELAY parameter.
4. HW_PSWITCH2 and MOVE_OP precision use the same hardware resources, so it is not recommended to use them on the same channel at the same time, but they can be used on different channels at the same time.
5. It can only be compared once in each system cycle, and the system cycle is queried through SERVO_PERIOD.
6. Do not modify the TABLE position data until all comparison points are completed.
7. Both pulse axis and bus axis support this command.
8. When using a pulse-type motor, only when the ATYPE is 4 is the comparison feedback position (MPOS). The default factory ATYPE is 1 or 7, and the comparison is the command position (DPOS).</td></tr>
</table>

<table>
<tr><th>Instruction 20</th><th>ZAux_Direct_GetHwPswitchBuff</th></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_GetHwPswitchBuff(ZMC_HANDLE handle, int axisnum,int *buff)</td></tr>
<tr><td>Description</td><td>Read remain buffers of hardware position comparison output.
It is valid in 4xx series pulse axis and encoder axis.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark.</td></tr><tr><td>Axisnum</td><td>Axis No. of comparison output</td></tr></table></td></tr>
<tr><td>Output parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>Buff</td><td>Remaining buffer</td></tr></table></td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>/</td></tr>
<tr><td>Details</td><td>/</td></tr>
</table>

<table>
<tr><th>Instruction 21</th><th>ZAux_Direct_HwTimer</th></tr>
<tr><td>Original Format</td><td>int32 __stdcall ZAux_Direct_HwTimer(ZMC_HANDLE handle,int mode, int cyclonetime, int optime, int reptimes, int opstate, int opnum)</td></tr>
<tr><td>Description</td><td>Hardware timer is used to restore electric level after one certain time when hardware comparison output, which is valid in 4xx series products.</td></tr>
<tr><td>Input parameters</td><td><table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark.</td></tr><tr><td>mode</td><td>0-OFF, 2-ON</td></tr><tr><td>cyclonetime</td><td>Period time, us unit.</td></tr><tr><td>optime</td><td>Valid time, us unit</td></tr><tr><td>reptime</td><td>Repeat times</td></tr><tr><td>opstate</td><td>It starts timing when output becomes the state that is not the state.</td></tr><tr><td>opnum</td><td>The port must support comparison output.</td></tr></table></td></tr>
<tr><td>Output parameters</td><td>/</td></tr>
<tr><td>Return value</td><td>If it is successful, return value is 0, if not, please refer to error codes.</td></tr>
<tr><td>Example</td><td>//period is adjusted as 2, output twice.
ZAux_Trigger(g_handle);
ZAux_Direct_SetOp(g_handle, 0, 0);
ZAux_Direct_HwTimer(g_handle, 2, 1000000, 500000, 2, 0, 0);//after OUT0 became ON, hardware timer is triggered to timing, after 500ms, it will switch to off.</td></tr>
</table>

	ZAux_Direct_SetOp(g_handle, 0, 1);
Details	1. There is only one HW_TIMER, and each call will forcibly stop the previous call. 2. The HW_TIMER function of each output port of ZMC420SCAN is independent. 3. Some ZMC3 series, 4 series and above products support this function. 4. OP and MOVE_OP operations will turn off the ongoing HW_TIMER pulse, so that HW_TIMER can be used to achieve a PWM function. The OP outputs, it turns on the pulse output, and the next OP outputs, it turns off the pulse output. When MOVE_OP is used for precise output, precise PWM output infinite pulse function output can be achieved.

Instruction 22	**ZAux_Direct_MultiMovePt**
Original Format	int32 __stdcall ZAux_Direct_MultiMovePt(ZMC_HANDLE handle,int iMoveLen, int imaxaxises, int *piAxislist, uint32 *pTickslist, float *pfDisancelist)
Description	Set the movement distance for diver motor within the specified ticks. Ticks: the length of time, about 1ms
Input parameters	Parameter name: Description handle: Link mark. iMoveLen: Filled motion numbers imaxaxises: Total axes that participated in motion. piAxislist: Axis No. list pTicklist: Ticks time list pfDistancelist: Motion distance list
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Pt motion
Details	Generally, the PC calculates the corresponding coordinates every cycle, and then transmits them to the controller. Speed during motion = (movement distance/time length)*1000 units/s. Do not move a large distance in a very short time, the pulse frequency will be too high, and the motor will stall. It can be broken down into small segments and sent repeatedly.

Instruction 23	**ZAux_Direct_MultiMovePtAbs**
Original Format	int32 __stdcall ZAux_Direct_MultiMovePtAbs(ZMC_HANDLE handle, int iMoveLen, int imaxaxises, int *piAxislist, uint32 *pTickslist, float *pfDisancelist)
Description	Set the absolute distance for diver motor within the specified ticks.

	Ticks: the length of time, about 1ms
Input parameters	Parameter name: Description handle: Link mark. iMoveLen: Filled motion numbers imaxaxises: Total axes that participated in motion. piAxislist: Axis No. list pTicklist: Ticks time list pfDistancelist: Motion distance list
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Pt motion
Details	Generally, the PC calculates the corresponding coordinates every cycle, and then transmits them to the controller. Speed during motion = (movement distance/time length)*1000 units/s. Do not move a large distance in a very short time, the pulse frequency will be too high, and the motor will stall. It can be broken down into small segments and sent repeatedly.

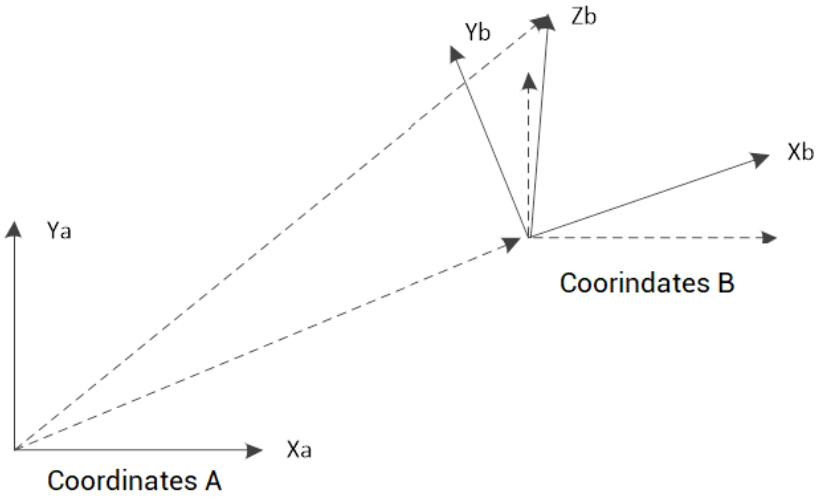
Instruction 24	**ZAux_Direct_Connframe**
Original Format	int32 __stdcall ZAux_Direct_Connframe(ZMC_HANDLE handle, int Jogmaxaxises, int *JogAxislist, int frame, int tablenum , int Virmaxaxises , int *VirAxislist)
Description	Robotic arm reverse kinematic instruction.
Input parameters	Parameter name: Description handle: Link mark. Jogmaxaxises: The number of joint axes JogAxislist: The list of joint-axis frame: Robotic arm type, 1-scara, 3-Palletizing tablenum: The starting place to store robot related parameter in TABLE Virmaxaxises: The number of virtual axes VirAxislist: The list of virtual axes
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Build Robotic Arm
Details	When the connection speed is 0, the motion speed and acceleration of the joint coordinate system are limited by parameters such as SPEED.

	➢ When there is an error such as joint axis alarm, this movement will be cancelled by CANCEL. ➢ Do not CANCEL this movement during the high-speed motion of the virtual axis, which will cause the axis to stop during high-speed motion. ➢ When this command is LOAD, it will automatically modify the coordinates of the virtual axis to make it consistent with the joint axis. Therefore, after calling, it needs WAIT LOADED before starting to move. ➢ Do not modify the coordinates of the virtual axis during the connection process, or call DATUM and other movements that may change the coordinates, which will cause the joint axis to quickly move to the new virtual position. ➢ After CONNFRAME takes effect, the MTYPE of the joint axis is 33. At this time, the joint axis cannot be moved directly, and the joint axis must be moved indirectly by running the virtual axis. When the joint axis needs to be moved directly, first call CANCEL to cancel the CONNFRAME, and then move the joint axis. ➢ When both the virtual axis and the actual axis are rotating axes, such as the end rotating axis, the pulse equivalent of the virtual axis and the actual axis should be consistent.

Instruction 25	**ZAux_Direct_Connreframe**
Original Format	int32 __stdcall ZAux_Direct_Connreframe(ZMC_HANDLE handle, int Virmaxaxises, int *VirAxislist, int frame, int tablenum, int Jogmaxaxises, int *JogAxislist)
Description	Robotic arm forward kinematic instruction.
Input parameters	<table><tr><td>Parameter name</td><td>Description</td></tr><tr><td>handle</td><td>Link mark.</td></tr><tr><td>Virmaxaxises</td><td>The number of virtual axes</td></tr><tr><td>VirAxislist</td><td>The list of virtual-axis</td></tr><tr><td>frame</td><td>Robotic arm type, 1-scara, 3-Palletizing</td></tr><tr><td>tablenum</td><td>The starting place to store robot related parameter in TABLE</td></tr><tr><td>Jogmaxaxises</td><td>The number of joint axes</td></tr><tr><td>JogAxislist</td><td>The list of joint axes</td></tr></table>
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	Build Robotic Arm
Details	This is the opposite instruction of inverse kinematic motion. ➢ When virtual axis CAONNREFRAME moves LOAD, joint axis

	CONNFRAME motion will be CANCEL automatically. ➢ When joint axis CONNFRAME moves LOAD, virtual axis CONNREFRAME motion will be CANCEL automatically.

Instruction 26	ZAux_Direct_FrameRotate
Original Format	int32 __stdcall ZAux_Direct_FrameRotate(ZMC_HANDLE handle,int iaxis, float *pfRotatePara)
Description	Robotic arm coordinates rotation is used for translation and rotation of workpiece coordinate system.
Input parameters	Parameter name: Description handle: Link mark. iaxis: Axis No., joint axis / virtual axis pfRotateParat: Translate rotatory parameter
Output parameters	/
Return value	If it is successful, return value is 0, if not, please refer to error codes.
Example	/
Details	At present, only FRAME6 can rotate the attitude at the same time. Others with XYZ virtual axis can support the rotation of XYZ three axes, but the attitude axis cannot be rotated. After rotation, the virtual axis WORLD_DPOS indicates that the world coordinates will not change, and the virtual axis DPOS indicates that the workpiece coordinates will change. And when using, it requires an existing mechanical connection to the controller. The BASE axis can be any one of the virtual axis and the joint axis, when the BASE axis is not connected with a robot, an error 1025 will be reported. In the case of multiple manipulator superposition, identify which robot mode is based on the BASE axis, if axis_1 in BASE (axis_1, axis_2) is the manipulator axis of mode 1, and axis_2 is the manipulator axis of mode 2, then the result is that the manipulator of mode 1 performs coordinate calculations, namely, they are calculated in the order of the BASE axis.



Instruction 27	ZAux_Direct_MoveSync
Original Format	int32 __stdcall ZAux_Direct_MoveSync(ZMC_HANDLE handle,float imode,int synctime, float syncposition, int syncaxis, int imaxaxises, int *piAxislist, float *pfDisancelist)
Description	Synchronous motion, the object on the belt follows, this motion is non-interpolation motion, which means it can't ensure the trajectory is a straight line.
Input parameters	

Parameter name	Description
handle	Link mark.
imode	Sync mode: imode =-1: end mode, move to the specified absolute position, if the movement in this mode is followed by other ZAux_Direct_MoveSync commands, it will be overwritten, and the belt axis is invalid in this mode. imode =-2: forced end, the original ZAux_Direct_MoveSync is forced to stop when called, and the movement reaches the specified end position. If this mode movement is followed by other ZAux_Direct_MoveSync instructions, it will be overwritten. In this mode, the belt axis is invalid. imode =-10: the first axis follow in following-axis list imode =-20: the second axis follow in following-axis list imode =-30: the third axis follow in following-axis list imode=0+angle:

		Belt rotation angle. angle: belt rotation angle, angle = positive rotation angle between the belt and the 1/2 axis in the list of following axes. For example, imode = PI/4, the belt is in the direction of 45 degrees. imode = PI/2, the belt is in the y direction. imode = PI, the belt is in the negative x direction; imode = (PI*1.75), the belt is in the direction of -45 degrees imode=1000+tableindex, the tracking ratio of the axis number list is stored in the table in turn, so that three-dimensional belt tracking can be realized when the belt is inclined. (Firmware version 20180209 adds this function) (tableindex: table register starting No.)
	synctime	Synchronization time, in ms unit. The movement is completed within the specified time, and the speed of the axis is consistent with that of the object on the belt axis when it is completed. 0 indicates that the synchronization time is estimated based on the velocity and acceleration of the motion axis, which may not be accurate.
	syncposition	Belt axis position when an object is sensed. This command supports the belt axis coordinate cycle, but when the command is called, ensure that there is no coordinate modification or cycle operation between the parameter position and the current belt axis position, so this command should not be near the coordinate cycle point when calling
	syncaxis	Belt axis No.
	imaxaxises	The number of total slave axes that participated in synchronization.
	piAxislist	Following axis axis No. list
	pfDistancelist	Slave axis absolute coordinates position when belt axis object is sensed.
Output parameters	/	
Return value	If it is successful, return value is 0, if not, please refer to error	

	codes.
Example	Belt-axis Synchronization Motion Following
Details	Imode: synchronization mode