雷赛 PAC9300 运动控制器 ST 语言编程
及可视化界面设计简明手册

2020 年 6 月

目　录

第 1 章　引　言 2
第 2 章　结构化文本（ST）语言简介 5
2.1．ST 语言的数据类型 5
2.2．运算符 6
2.3. 常用的数学函数 6
2.4. 常用语句 7
第 3 章　雷赛 iStudio 软件的使用 10
3.1. 准备工作 10
3.2. 调试第一个 ST 程序（FOR 和 IF 语句、二维数组） 10
第 4 章　EtherCAT 总线及节点的参数设置 20
4.1. 添加设备 20
4.2. 设置 EtherCAT 参数 24
4.3.　IO 的映射与控制（轴使能、JOG 运动模块） 25
第 5 章　可视化界面的设计 35
5.1. 指示灯与按键的设计 35
5.2. 文本框的设计（读位置模块、相对点位运动模块） 41
5.3. 组合框和速度表的设计（定时器、读速度模块、回原点模块） 44
5.4. 单选钮、复选框和开关的设计　（CASE 语句） 50
5.5. 软数字键盘的调用 54
5.6. 最大化窗体改为指定大小的窗体 54
5.7. 程序自启动的设置 55
第 6 章　轨迹控制程序设计 57
6.1. 用直线插补实现轨迹控制（子程序、直线插补模块） 57
6.2. 用样条曲线插补实现轨迹控制（样条曲线插补、函数的调用） 68
第 7 章　ST 程序与 C#程序的通讯 77
7.1. C#程序读写共享内存的方法 77
7.2. ST 程序读写共享内存的方法 80

第 1 章　引　言

PAC（Programmable Automation Controller）是可编程自动化控制器，它将 PLC、DCS（分布式计算机控制系统）和 PC 机技术集成在一体，以满足现代工业应用中越来越复杂的需求。

雷赛 PAC9300 运动控制器外形如图 1.1 所示，它本质为一台专用的嵌入式工业计算机，其 CPU 为 4 核 1.99GHz 主频、内存为 2GB、固态硬盘为 60GB。操作系统为 Windows7。运动控制软件独占一个 CPU 核，另外 3 核运行其他软件。

PAC9300 运动控制器有 1 路 EtherCAT 总线接口，可连接 128 个节点；控制 32 个电机时通讯周期可达 1ms。

PAC9300 运动控制器有 1 个 VGA 显示器接口、4 个 USB 接口，可用于连接显示器、键盘和鼠标等设备；有 2 个 Ethernet 网络接口，用于连接上位机、网口相机等设备；还有 2 路 RS232 接口和 2 路 RS485 接口，可与其他扩展设备连接。



图 1.1　雷赛 PAC9300 运动控制器外观

PAC9300 运动控制器具有多轴直线插补、空间圆弧插补、螺旋线插补、样条曲线插补、电子齿轮、电子凸轮等运动控制功能。可采用符合 IEC61131-3 标准的 5 种编程语言编写程序，5 种编程语言分别为：梯形图（LD）、指令表（IL）、顺序功能图表（SFC）、功能块图（FBD）、结构化文本（ST）。同时可以设计漂亮的可视化界面，如图 1.2 所示。

本文将以图 1.3 所示的运动控制系统为硬件平台。其中，PAC9300 运动控制器通过 EtherCAT 总线与三轴平台连接，上有一个雷赛 L7 总线型交流伺服电机、二个 DM 系列总线型步进电机以及一些 IO 器件；同时，PAC9300 还接有一个显示器和鼠标、键盘。

EtherCAT 总线三轴运动平台电路图如图 1.4 所示。

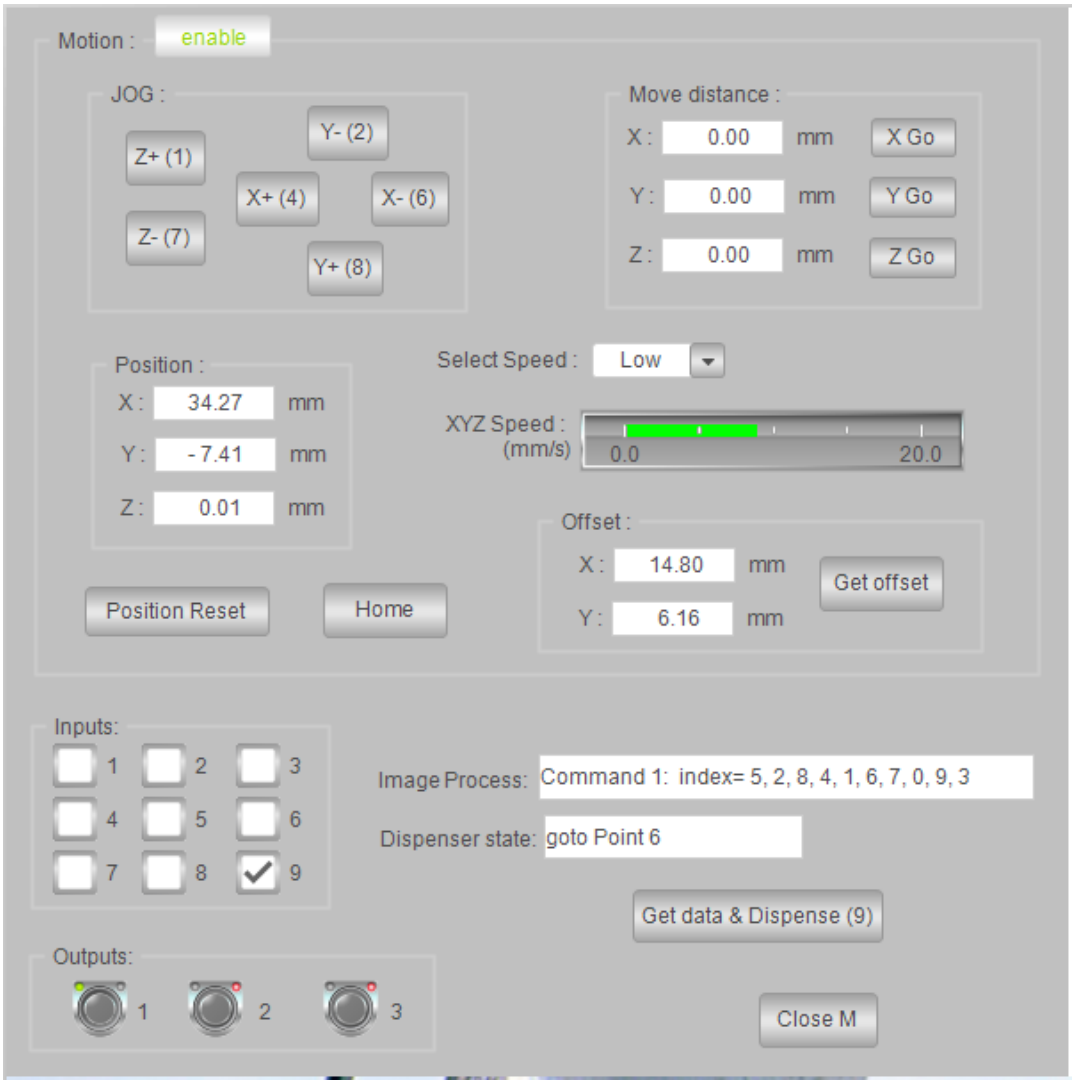

图 1.2　PAC9300 运动控制器的可视化界面

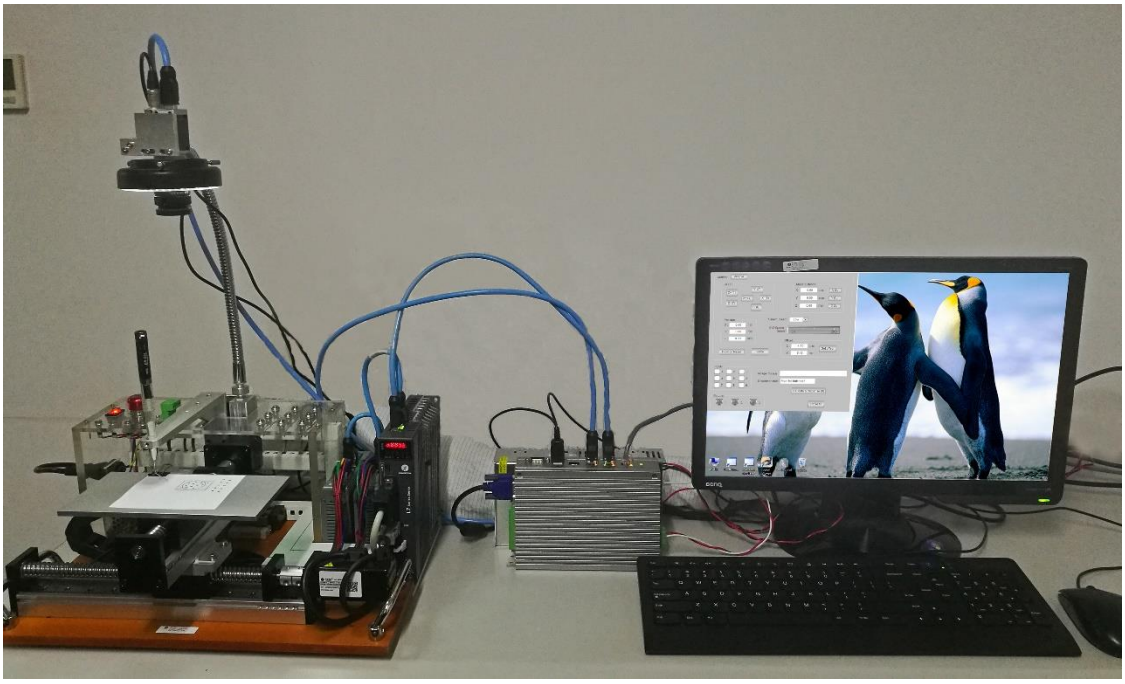

图 1.3　雷赛 PAC9300 运动控制器实验系统

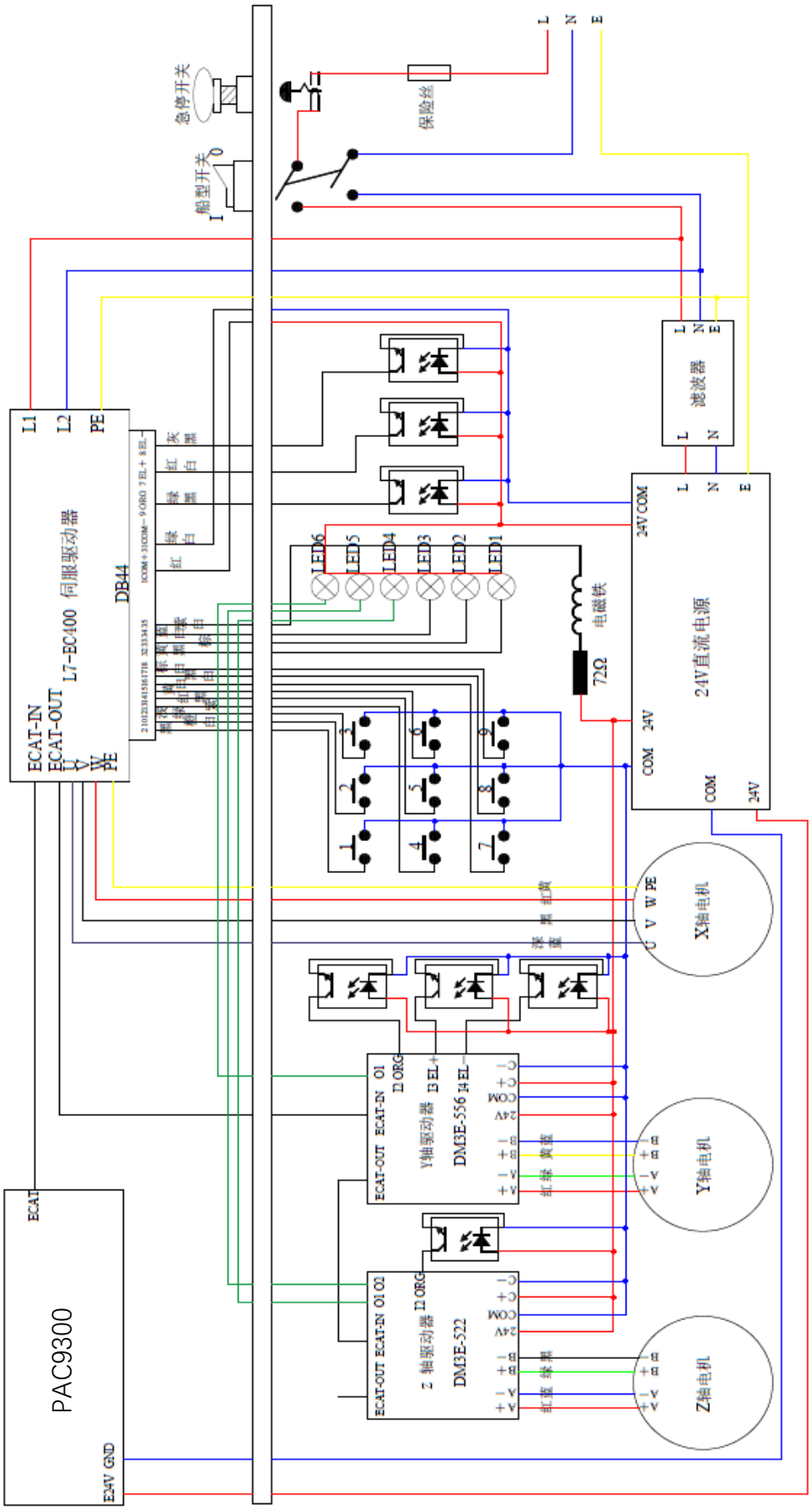

图 1.4 EtherCAT 总线三轴运动平台电路图

第 2 章　结构化文本（ST）语言简介

结构化文本编程语言（Structured Text）是 IEC 61131-3 标准支持的五种语言之一，专为可编程逻辑控制器（PLC）设计。它是一种块结构的高级语言，在语法上与 Pascal 语言相同，和 C 语言相似。

注意：ST 语言中字母不分大小写！

每条语句要由分号“;”结尾。

2.1．ST 语言的数据类型

A．常用的数据类型有：

BOOL:	逻辑量,TRUE 或 FALSE
INT:	16 位有符号整数
DINT:	32 位有符号整数
REAL:	32 位实数
LREAL:	64 位实数
STRING:	字符串
TIME:	时间值

B．常量

数字常量可以定义为一个二进制、十六进制和十进制数。二进制和 16 进制的数有前缀 2#、16#。

```
例：  2#1001_0011   // 二进制数
      16#A2         // 十六进制
      3.14159       // 十进制
```

字符串常量用单引号定义。例：’enable ’

时间常量：以 t#为前缀，时间单位有：d - 天、h - 时、m - 分、s - 秒、ms - 毫秒

```
例：  T#100S12ms
      t#12h34m15s
```

C．变量

变量名必须以字母开头，可以加入数字。

使用变量必须先定义变量。本地变量只能在一个子程序、或一个函数中使用；全局变量可在所有子程序和函数中使用。定义变量时可赋初值。

本地变量定义的格式：

```
VAR
    i: INT;
    Speed: LREAL := 20;
END_VAR
```

全局变量定义的格式：

```
VAR_GLOBAL
    IN1: BOOL;
```

```
    Wcount: INT ;
    motorState: STRING := 'Stop';
END_VAR
```

变量类型转换的函数有很多。通用的转换函数为：ANY_TO_<任意数据类型>

示例：

```
// 从 REAL 数据类型到 INT 类型的转化
re : REAL := 1.234;
i : INT := ANY_TO_INT(re);
```

D. 数组

一维数组的定义格式：数组名：ARRAY[0..n] OF 类型；　　//该数组的长度为 n+1

二维数组的定义格式：数组名：ARRAY[0..m,0..n] OF 类型；//（m+1）行×（n+1）列

例：

定义：

```
LineEnd: ARRAY[0..2] OF LREAL := [0,0,0]; //定义时可以赋初值
answer:ARRAY[0..3,0..3] OF INT;
```

使用：

```
LineEnd[1]:= 10;
answer[i,1]:= hen;
```

2.2．运算符

常用运算符及其优先级如表 2.1 所示。

表 2.1.　运算符及其优先级

优先级	运算符号	描述
1	（……）	括号
2	函数名（……）；	函数的调用
3	- NOT	负号 非
4	* / MOD	乘 除 求余数
5	+ -	加 减
6	<，<=，>，>=	小于，小于等于，大于，大于等于
7	=，<>	等于，不等于
8	AND XOR OR	与 异或 或

2.3. 常用的数学函数

常用的数学函数如表 2.2 所示。

表 2.2.　常用的数学函数

函数名	描述	单位
ABS（x）	绝对值	
SQRT（x）	平方根	
LN（x）	自然对数	
LOG（x）	10 为底的对数	
EXP（x）	指数	
EXPT（y，x）	幂	计算 y 的 x 次方
SIN（x）	正弦	角度单位为弧度
COS（x）	余弦	
TAN（x）	正切	
ASIN（x）	反正弦	函数值的单位为弧度
ACOS（x）	反余弦	
ATAN（x）	反正切	

2.4. 常用语句

A. 赋值语句

:=

注：ST 语言中的赋值语句和 C 语言不同，等号前多一个冒号。

例：DelayOn := TRUE;

```
L := r0 - r0*COS(PI/4);
```

B. 条件语句

```
IF 条件表达式 THEN
    ……
  ELSE
    ……
END_IF;
```

例：

```
IF sum = 100.0  THEN  // 注意：不要写成 sum := 100 ！否则软件不会报错，程序不对。
    i:=i+1;
  ELSE
    i:=i-1;
END_IF;
```

C. 循环语句

```
FOR 循环变量 := 初值 TO 终值 BY 步长 DO
……
END_FOR;
```

注：如果步长为 1，BY1 可省略

D. 条件循环语句

```
WHILE 条件表达式 DO
……
END_WHILE;
```

E. 开关语句

开关语句是一种多分支选择语句。格式如下：

```
CASE <变量> OF
      <数值 1>: <指令块 1>
      <数值 2>: <指令块 2>
      <数值 3, 数值 4, 数值 5>: <指令块 3>
      <数值 6 .. 数值 10>: <指令块 4>
      ……
      <数值 n>: <指令块 n>
   ELSE <其他指令块>
END_CASE;
```

例：

```
CASE INT1 OF
   1, 5: BOOL1 := TRUE;     // 如果 INT1 = 1 或 5，BOOL1 = TRUE，BOOL3 = FALSE
         BOOL3 := FALSE;
      2: BOOL2 := FALSE;   // 如果 INT1 = 2，BOOL2 = FALSE，BOOL3 = TRUE
        BOOL3 := TRUE;
 10..20: BOOL1 := TRUE;    // 如果 INT1 在 10~20 内，BOOL1 = TRUE，BOOL3 = TRUE
         BOOL3:= TRUE;
  ELSE
      BOOL1 := NOT BOOL1;       // 否则，BOOL1 取反，BOOL2 = BOOL1 OR BOOL2
      BOOL2 := BOOL1 OR BOOL2;
END_CASE;
```

F. EXIT 语句

在 FOR、WHILE 或 REPEAT 循环语句中执行 EXIT 语句，循环将立即停止。

F. CONTINUE 语句

在 FOR、WHILE 或 REPEAT 循环语句中执行 CONTINUE 语句，循环将立即开始下一次循环。

```
FOR Counter:=1 TO 5 BY 1 DO
    INT1:=INT1/2;
    IF INT1=0 THEN
        CONTINUE;    // 避免除数为零
    END_IF
    Var1:=Var1/INT1; （* 除数不为零时执行 *）
END_FOR;
```

G. RETURN 语句

退出子程序或函数。

H. 调用功能块、子程序、函数的语句

格式： 功能块或子程序名（参数 1，参数 2，……）;

变量 := 函数名（参数 1，参数 2，……）；

I. 注释符号

单行注释符号为：“//”；多行注释符号为“(*” 语句块 “*)”。

J. iStudio 软件帮助文件

在雷赛的 iStudio 软件中，有完整的帮助文件。其中有关 ST 语言比较完整的内容目录如图 2.1 所示。用户有必要仔细阅读。

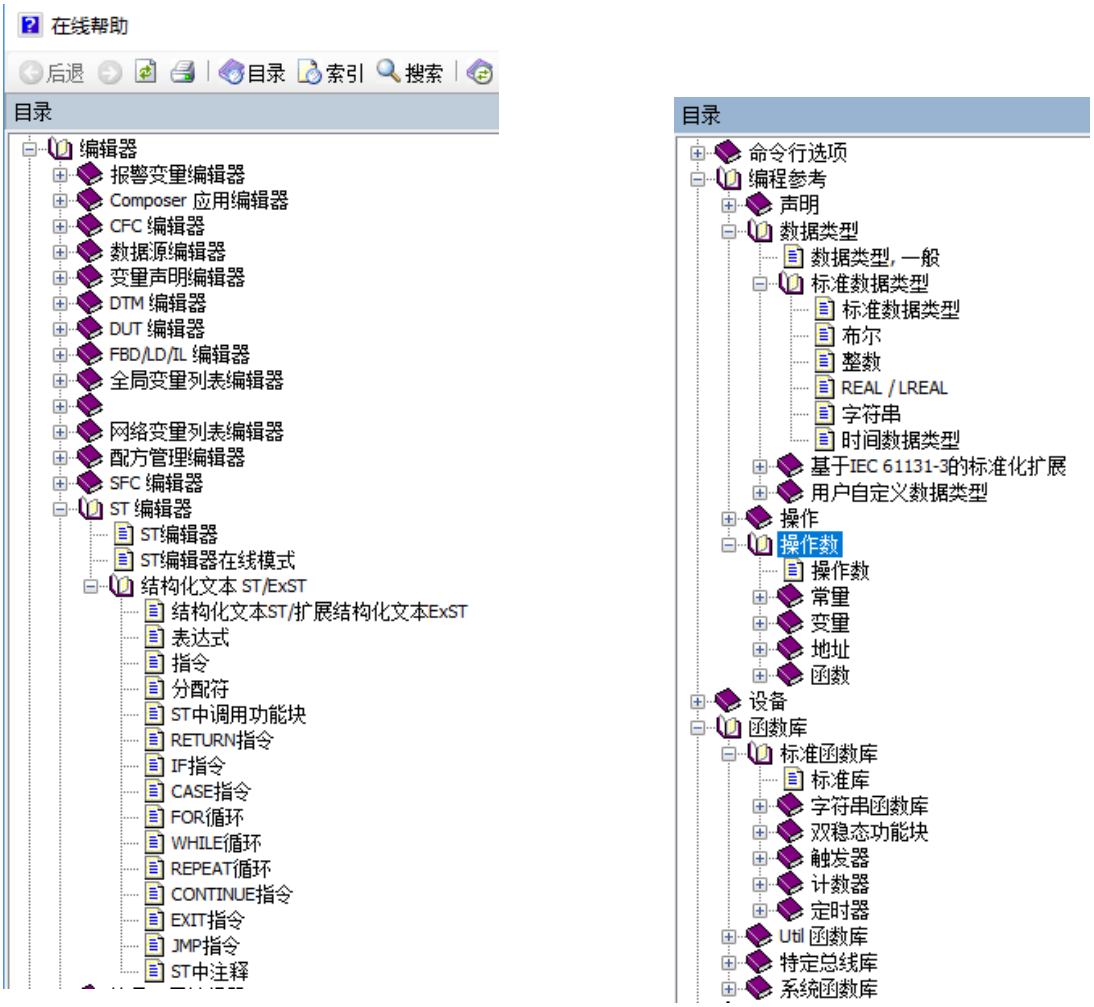

图 2.1 雷赛 iStudio 软件帮助文件中与 ST 语言相关内容的目录

第 3 章　雷赛 iStudio 软件的使用

3.1. 准备工作

首先在一台 PC 机上安装雷赛 iStudio 软件。然后将 PC 机上的网线接入 PAC9300 运动控制器上的网口 LAN1；再用另一条网线将 PAC9300 的网口 LAN2 与 3 轴运动平台上的雷赛交流伺服电机驱动器上的 CN4 接口相连。参见图 1.1、1.3。

给运动平台接入 220V 交流电；给 PAC9300 运动控制器接上直流 24V 电源；运行 PC 机上的 iStudio 软件，给 PAC 运动控制器编写程序、调试程序。

下面通过一个例程，介绍 iStudio 软件的基本用法。

3.2. 调试第一个 ST 程序 (FOR 和 IF 语句、二维数组)

例程一：求解“百钱百鸡”问题。鸡翁一，值钱五，鸡母一，值钱三，鸡雏三，值钱一。百钱买百鸡，问鸡翁、鸡母、鸡雏各几何？

ST 程序代码如下：

```
PROGRAM PLC_PRG
VAR
    cock,hen,chick,i:INT;                // 定义整数型变量
    sum:REAL;                            // 定义实数型变量
    answer:ARRAY[0..3,0..3] OF INT;      // 定义整数型二维数组 4X4
    flag:BOOL:=TRUE;                     // 定义布尔型变量且赋初值
END_VAR

IF flag=TRUE THEN                                  // 只计算一次
        flag:=FALSE;
        i:=0;
        FOR cock := 1 TO 19 DO                     // 公鸡数量在 1～19 之间变化
            FOR hen := 1 TO 31 DO                  // 母鸡数量在 1～31 之间变化
                chick := 100-cock-hen;             // 小鸡数量为 100 - cock - hen
                sum := cock*5+hen*3+chick/3.0;     // 计算购鸡的总价格
                IF sum = 100.0  THEN               // 如果总价格 sum = 100
                    answer[i,0]:=cock;             // 保存结果
                    answer[i,1]:=hen;
                    answer[i,2]:=chick;
                    i:=i+1;                        // 解的数量加 1
                END_IF;
            END_FOR;
        END_FOR;
END_IF;
```

调试程序的步骤如下：

A.　打开 iStudio 软件，如图 3.1 所示。

B. 新建工程

点击菜单中的“文件”，再点击“新建工程”。得图 3.2 所示的界面。在名称、位置栏填写工程的名称和在硬盘存放的路径。



图 3.1　雷赛 iStudio 软件开始界面

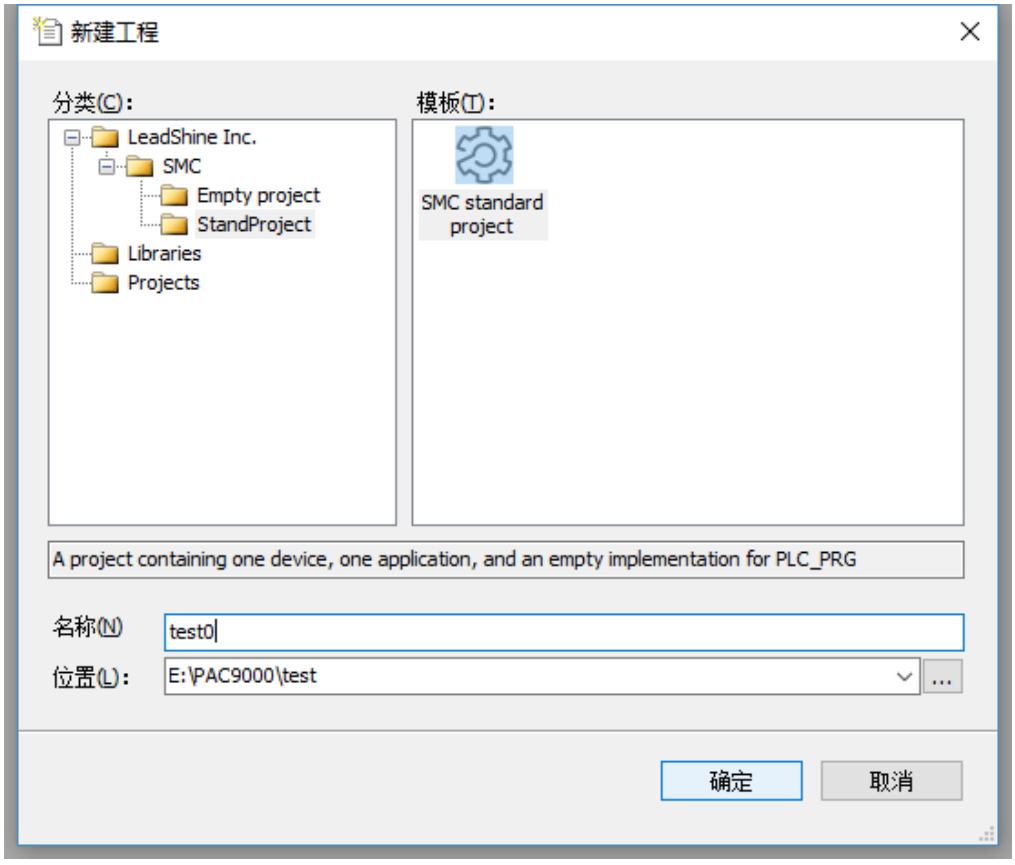

图 3.2　新建工程的界面

C. 选择控制器和编程语言

在图 3.2 中点击“确定”后，弹出如图 3.3 所示的界面，选择运动控制器的型号和编程语言类型。再点击“确定”。

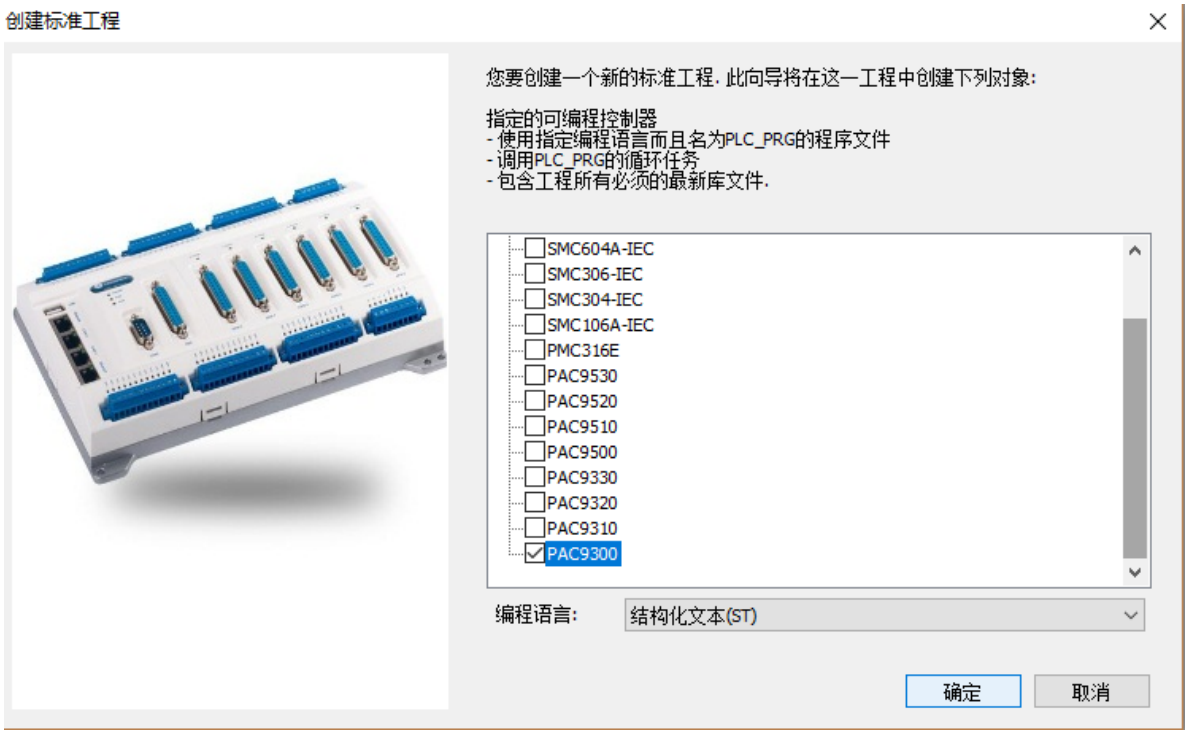

图 3.3　选择控制器和编程语言的界面

D. 编写代码

在设备栏的项目树中，双击“PLC_PRG”。弹出如图 3.4 所示的界面。在上面的框中定义变量；在下面的框中编写程序代码。完成的代码如图 3.5 所示。

在编写代码时，要注意代码的格式。该软件没有“自动对齐”的功能。

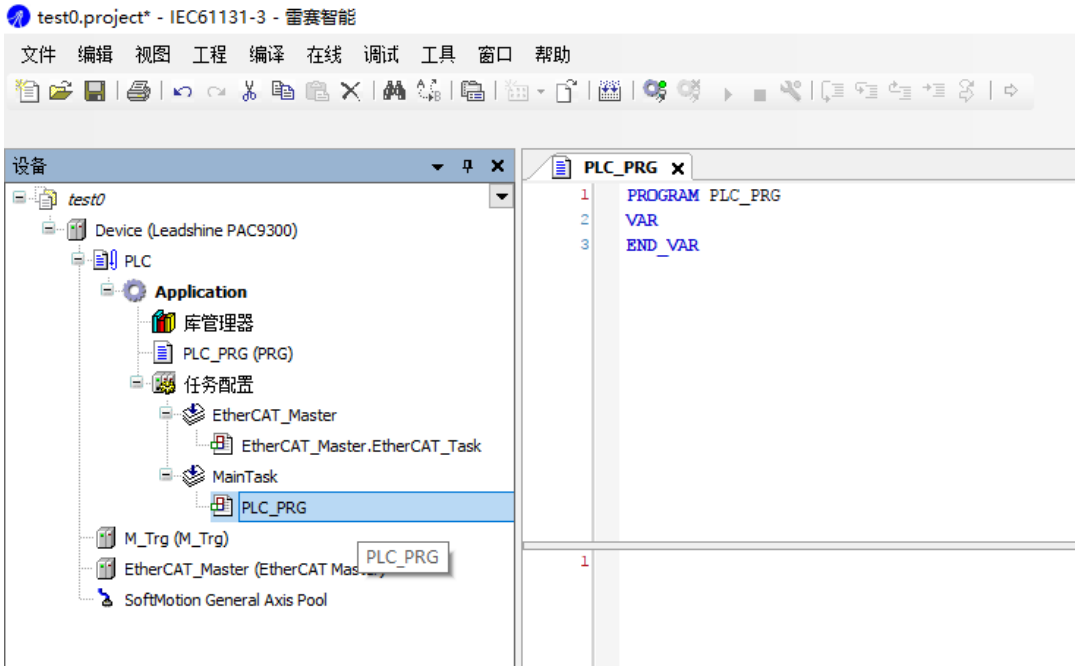

图 3.4　进入编程界面

PLC_PRG

```
PROGRAM PLC_PRG
VAR
        cock,hen,chick,i:INT;                   // 定义整数型变量
        sum:REAL;                               // 定义实数型变量
        answer:ARRAY[0..3,0..3] OF INT;         // 定义整数型二维数组4X4
        flag:BOOL:=TRUE;                        // 定义布尔型变量且赋初值
END_VAR
```

```
IF flag=TRUE THEN                               // 只计算一次
        flag:=FALSE;
        i:=0;
        FOR cock := 1 TO 19 DO                  // 公鸡数量在1～19之间变化
            FOR hen := 1 TO 31 DO               // 母鸡数量在1～31之间变化
                chick := 100-cock-hen;          // 小鸡数量为100 - cock - hen
                sum := cock*5+hen*3+chick/3.0;  // 计算购鸡的总价格
                IF sum = 100.0  THEN            // 如果总价格sum = 100
                    answer[i,0]:=cock;
                    answer[i,1]:=hen;
                    answer[i,2]:=chick;
                    i:=i+1;                     // 解的数量加1
                END_IF;
            END_FOR;
        END_FOR;
END_IF;
```

图 3.5　完成的代码

E. 连接 PAC9300 运动控制器

如图 3.6 所示，双击项目树中的“Device”，得 Device 界面如图 3.7 所示，点击“扫描网络”，得到图 3.8 所示的界面。

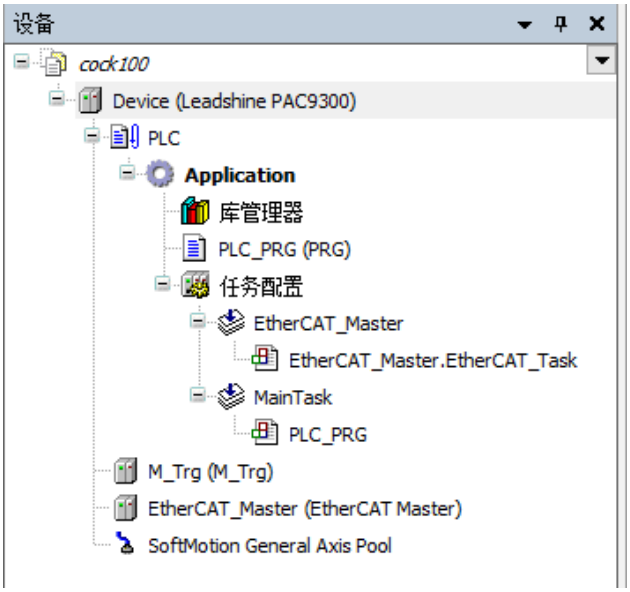

图 3.6　项目树中的“Device”

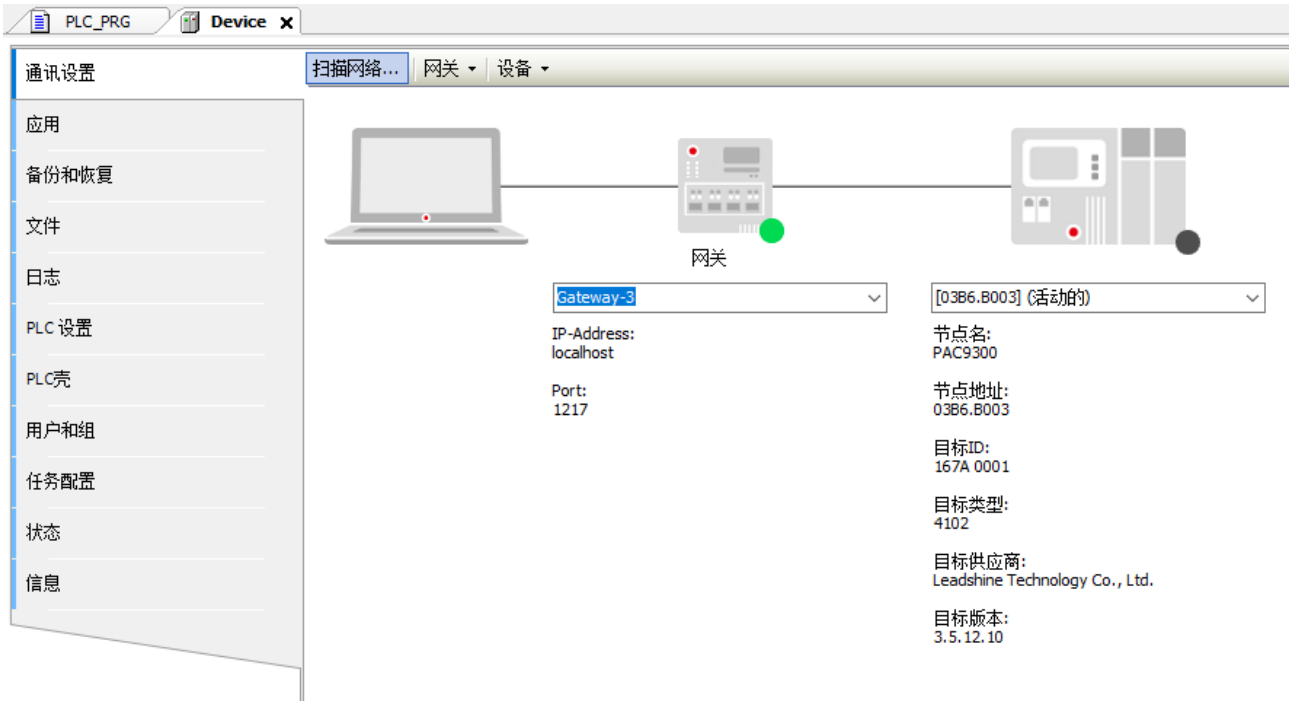

图 3.7　选择“扫描网络”卡

在图 3.8 所示的界面上选择设备“PAC9300”，点击“确定”；PAC 链接成功，如图 3.9 所示，有 2 个绿色指示灯。

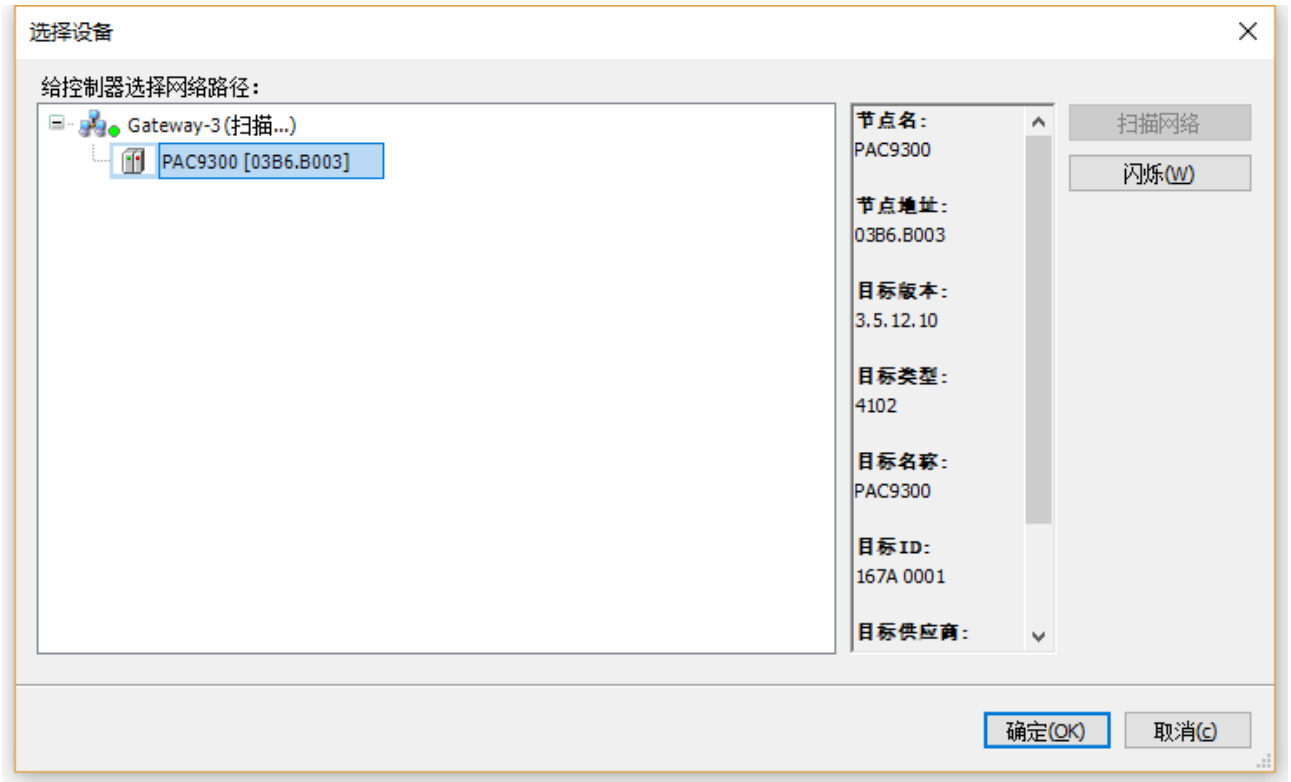

图 3.8　选择“PAC9300”运动控制器

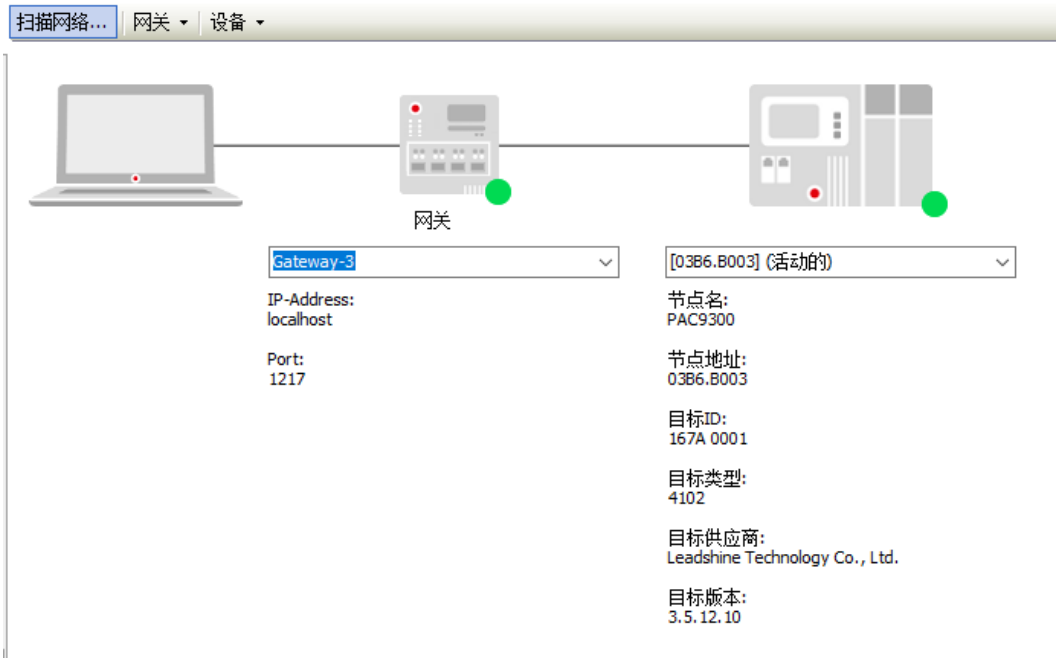

图 3.9　PC 机与 PAC 控制器连接成功

F. 编译程序

点击菜单上面的“编译”图标，如图 3.10 所示。编译完成后，点击界面左下角，弹出编译结果。如图 3.11 所示。

如果代码有语法错误，在该界面有错误提示。根据提示修改程序，直到编译没有错误为止。

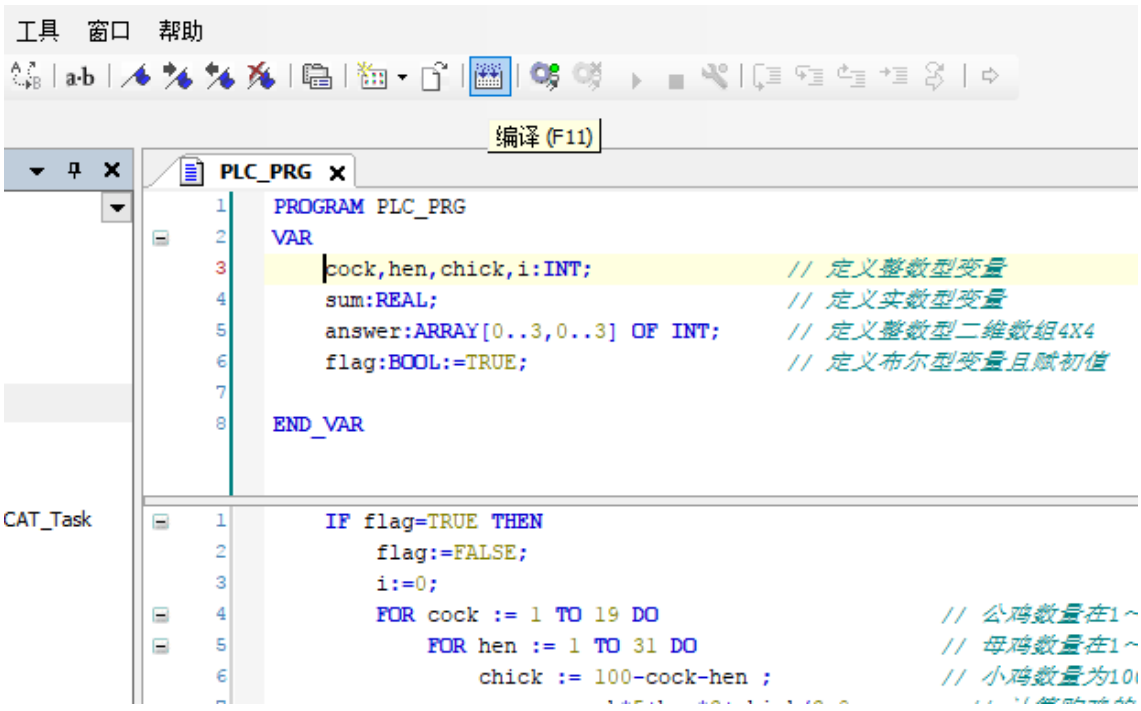

图 3.10　“编译”图标

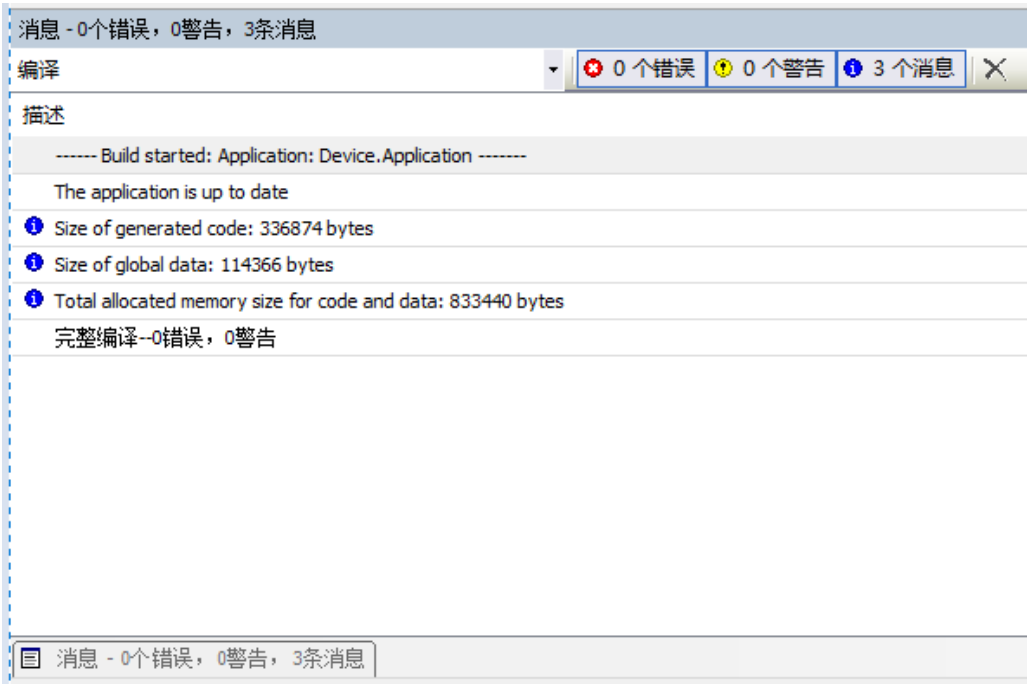

图 3.11　编译结果

G. 下载程序

点击菜单中的“登陆”图标，如图 3.12 所示。如果弹出如图 3.13 所示的对话框，点击“确定”，完成登录；之后又会弹出一个对话框如图 3.14 所示，点击“是”，下载程序。

如果修改程序后再一次编译下载，会弹出图 3.15 所示的对话框，选“登陆并下载”即可。

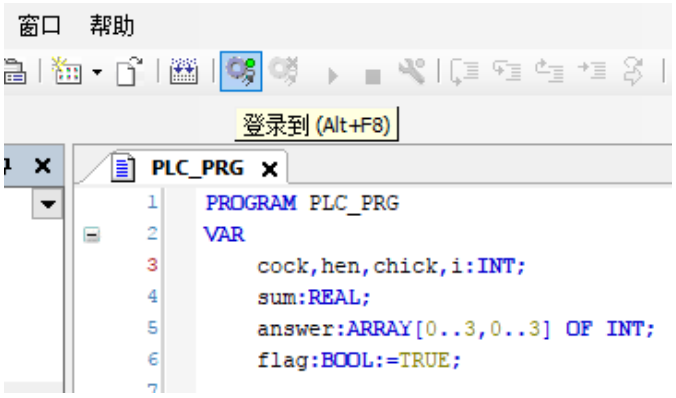

图 3.12　登陆到控制器

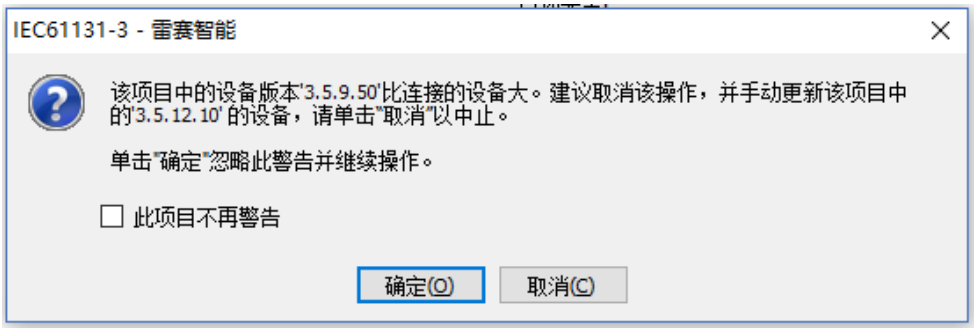

图 3.13　忽略警告

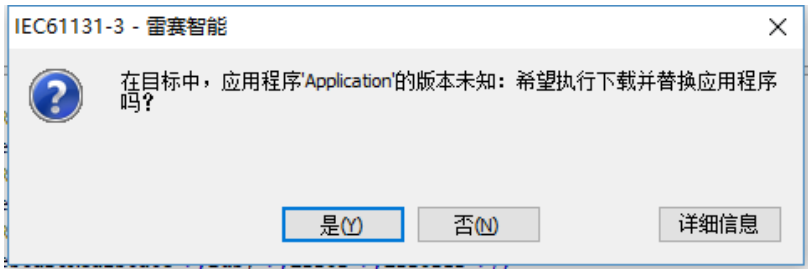

图 3.14　下载程序

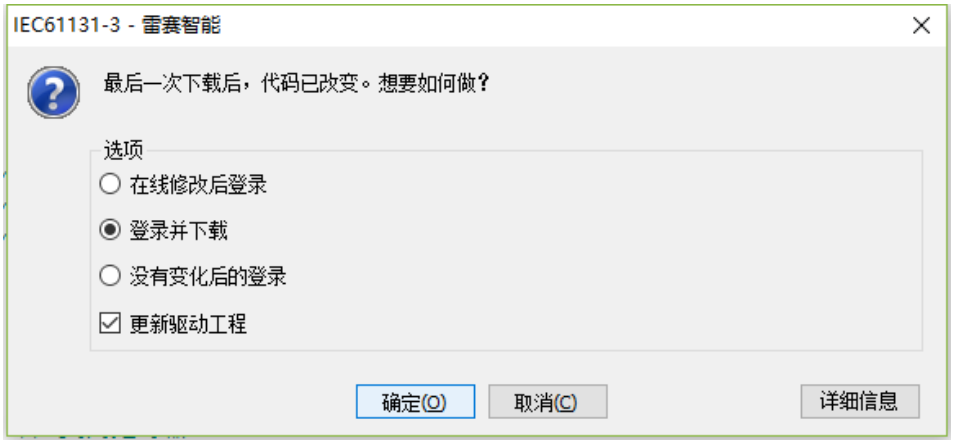

图 3.15　再次下载程序时选“登陆并下载”

H. 运行程序

点击菜单中的“启动”图标，如图 3.16 所示，进入到程序运行、调试状态。

I. 查看运行结果

点击变量表中 answer 前面的加号“+”，开打数组，查看运算结果，如图 3.17 所示。

在程序框内，每个变量后面都有一个小框，显示其当前数值。这些数据对调试程序非常有帮助。

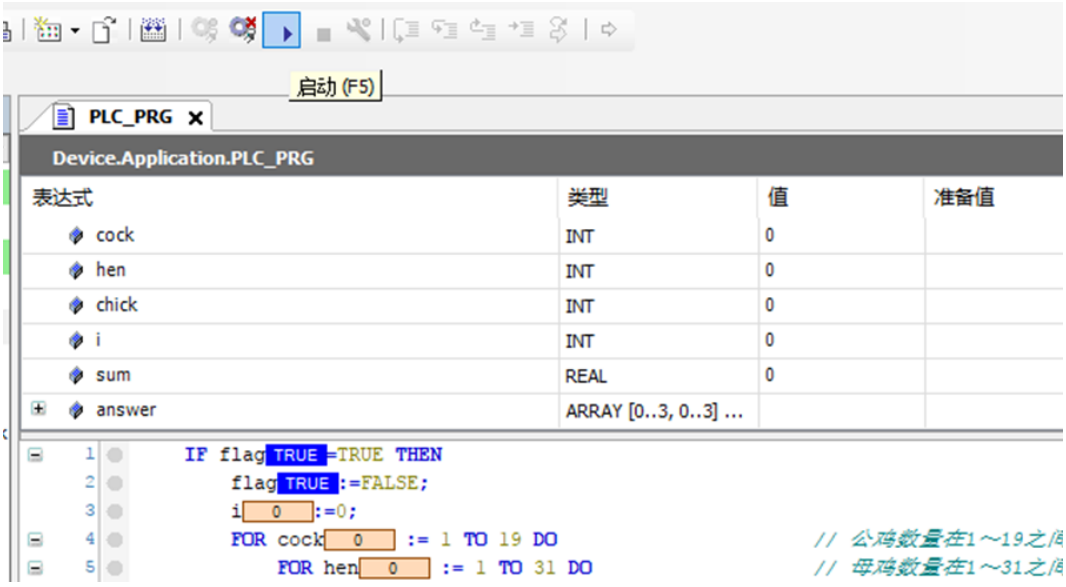

图 3.16　运行程序

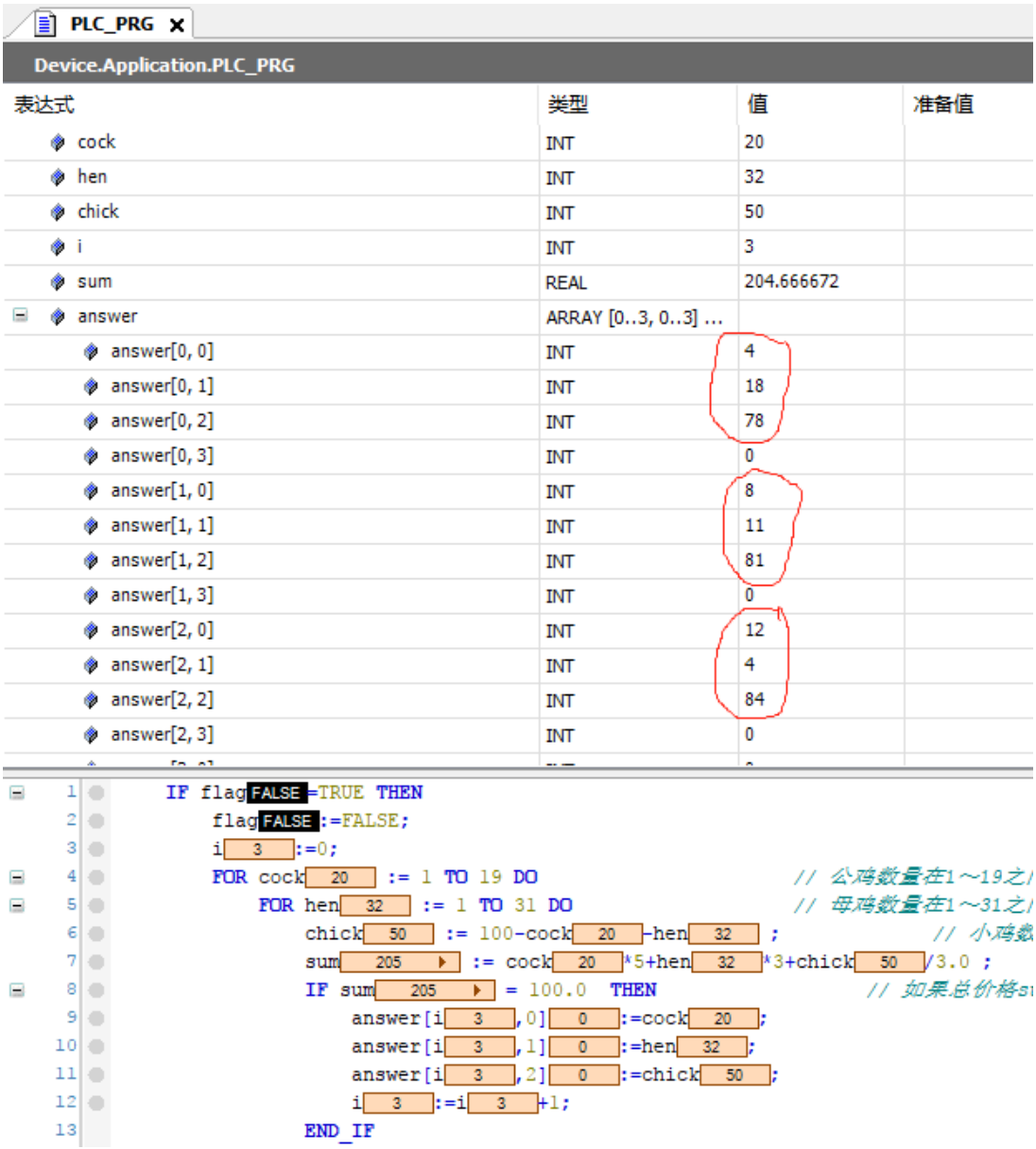

图 3.17　查看运算结果

J. 结束程序

如图 3.18 所示，点击菜单中的“停止”图标，结束程序运行。

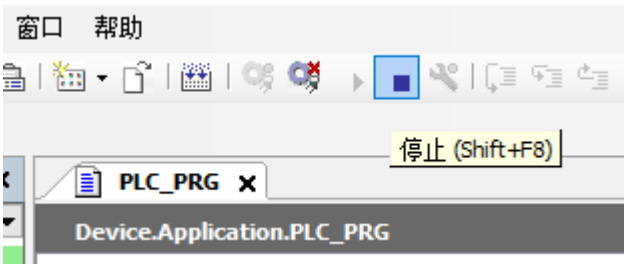

图 3.18　停止程序运行

再点击菜单中的“退出”图标，如图 3.19 所示，软件从调试状态退回到编辑状态。

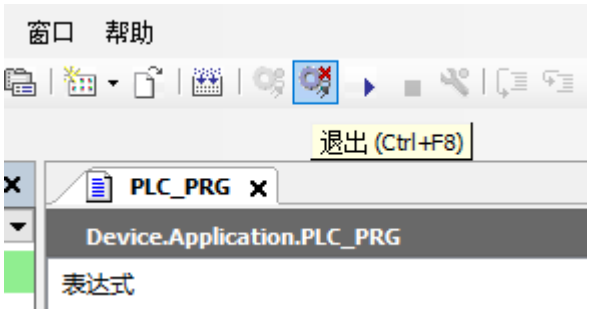

图 3.19 退出运行、调试状态

如果不修改、编译程序，要再一次运行程序，必须先登陆、在“在线”菜单下点击“热复位”，然后可正常启动程序。

注意：ST 编写的程序执行过程和 PLC 的梯形图一样，是反复扫描运行的。

如果将上述 ST 语言编写的百钱百鸡程序翻译成 C 语言，结果如下：

```
while(true)
{
  If (flag == true )
     {
      flag = false;
      for (cock = 1; cock <= 19; cock = cock + 1)      // 公鸡数量在 1～19 之间变化
         {
          for (hen = 1; hen <= 31; hen++)             // 母鸡数量在 1～31 之间变化
             {
              chick = 100 - cock - hen;               // 小鸡数量为 100- cock-hen
              sum = cock * 5 + hen * 3 + chick / 3;   // 计算购鸡的总价格
              if (sum == 100)                         // 如果总价格 sum  = 100
                {
                 result = "cock = " + Convert.ToString(cock);
                 result = result + "  hen = " + Convert.ToString(hen);
                 result = result + "  chick = " + Convert.ToString(chick) + "\n";
                 richTextBox1.AppendText(result);   // 输出符合“百钱百鸡”条件的结果
                }
             }
         }
     }
}
```

该程序是一个死循环程序。如果没有 If flag = true 和 flag := false 这几条语句，PAC 控制器将反复计算该程序。

在编写 PAC 程序时，一定要注意程序的运行时间，尤其是使用循环语句的时候要特别注意，程序执行一遍的时间不能超过 2 毫秒，否则，EtherCAT 总线型伺服电机有可能出错。

第 4 章　EtherCAT 总线及节点的参数设置

下面仍然通过一个例程来介绍 EtherCAT 总线及节点的参数设置方法。如图 1.3 所示，本实验系统有 3 个 EtherCAT 总线电机，即 EtherCAT 总线上有 3 个节点。

例程二：按下运动平台上的数字键 1，X 轴前进，指示灯 1 亮；按下数字键 2，Y 轴后退，指示灯 2 亮。按键抬起，电机停止，指示灯灭。

4.1. 添加设备

按照 3.2 中介绍的步骤 A~E，新建一个工程并连接 PAC9300 运动控制器。

添加 3 个电机及驱动器的步骤如下：

A. 扫描设备

用鼠标右键点击项目树中的“EtherCAT_Master”，如图 4.1 所示。在下拉菜单中选“扫描设备”。

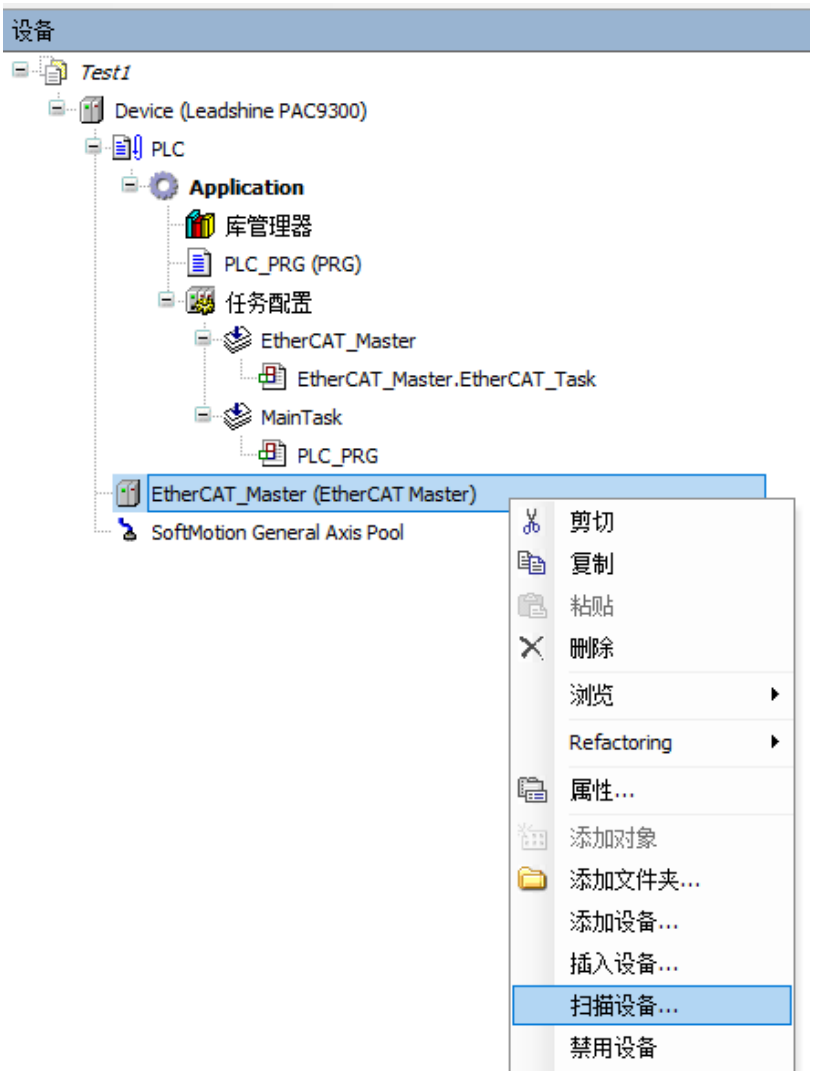

图 4.1　扫描设备

几秒钟后，显示图 4.2 所示的界面，3 个驱动器已经自动找到。点击“复制所有设备到工程中”键。之后在项目树中多出 3 个驱动器，如图 4.3 所示。

如果不能自动扫描到设备，则要先在设备库中添加设备描述文件。方法详见产品手册。

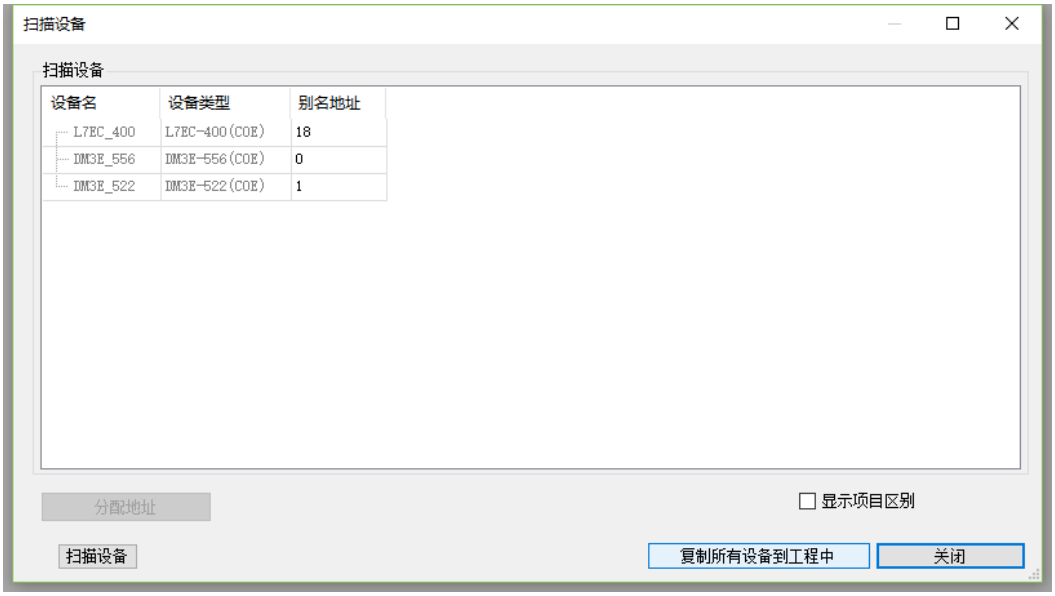

图 4.2　复制所有设备到工程中

B. 添加电机

用鼠标右键点击项目树中的驱动器“L7EC_400”，如图 4.3 所示。在下拉菜单中选“添加 SoftMotion 的 CiA402 轴”，之后弹出如图 4.4 所示的对话框，选择“确定”，使电机添加。

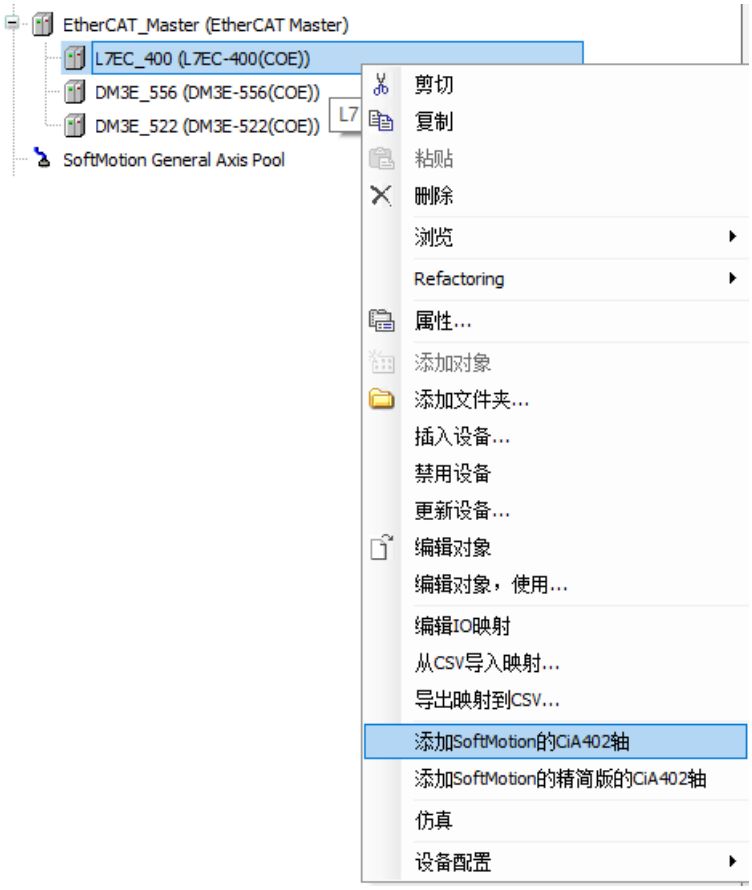

图 4.3　添加电机

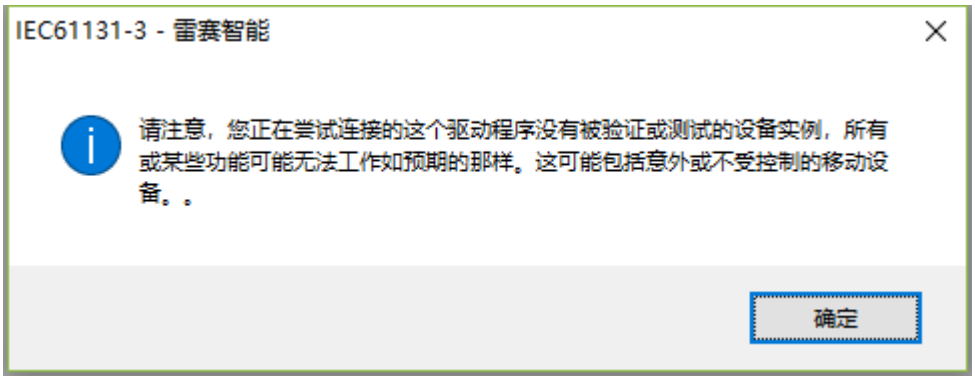

图 4.4　添加电机点“确定”

用同样的方法添加 2 个步进电机，结果如图 4.5 所示。

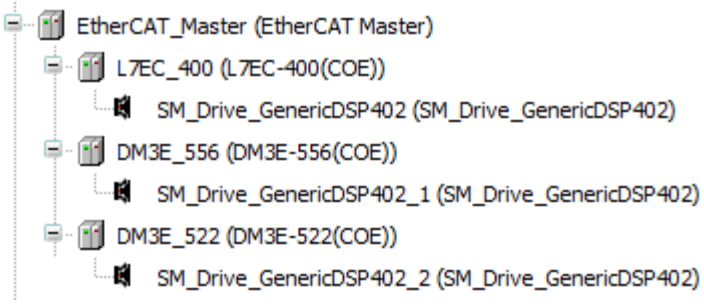

图 4.5　完成电机的添加

为了便于编程，可将 3 个电机的名称改为 X、Y、Z。点击电机名称“SM_Dirive_GenericDSP402”,会出现一个文本框，写入 X，如图 4.6 所示。

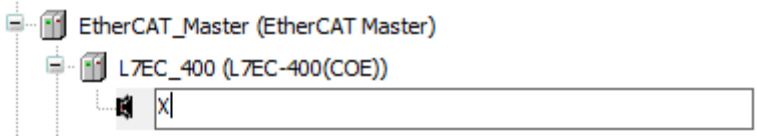

图 4.6　修改电机名称

之后，会弹出 2 个对话框，如图 4.7、4.8 所示。点“确定”即可。最后结果如图 4.9 所示。



图 4.7　电机重命名 1

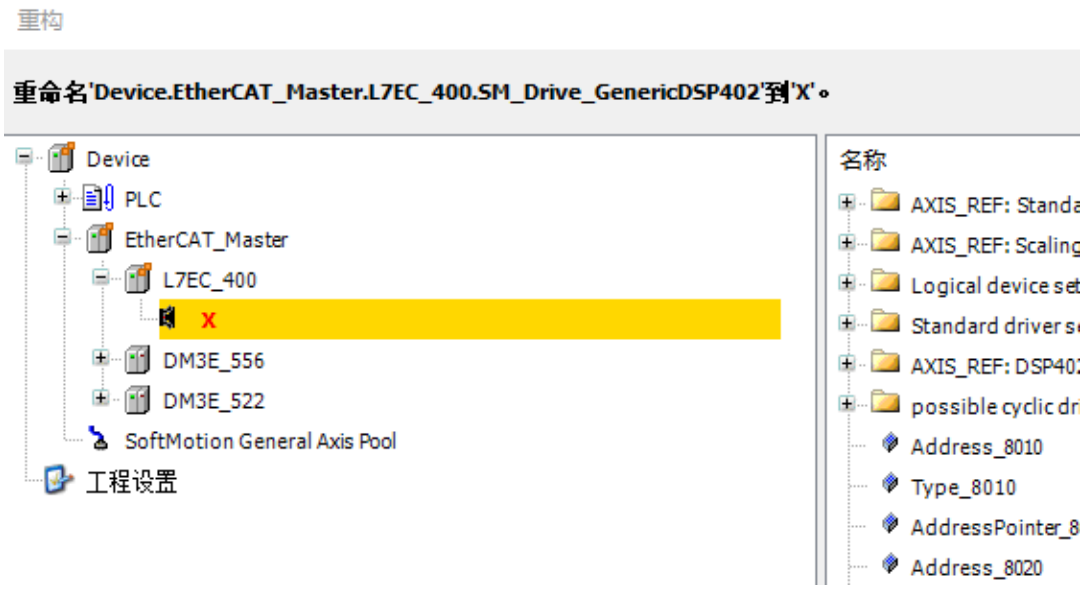

图 4.8　电机重命名 2

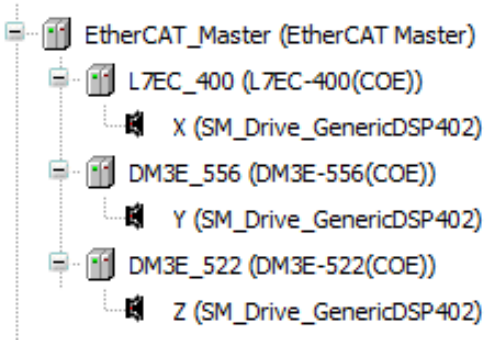

图 4.9　完成电机重命名

C. 设置电机参数

在项目树中双击电机“X”，弹出 X 轴的参数设置界面如图 4.10 所示。

在“基本”卡中，将“速度斜坡式”选为 $\sin^2$；将“CNC 限制”的速度设为 60。即最大速度为 60mm/s。

在“缩放/映射”卡中，“增量”设为 3000，其他均为 1。这样，增量即为脉冲当量。

伺服电机已设为每转 30000 个脉冲，丝杠导程为 10mm，故脉冲当量为 3000 脉冲/mm。

Y 轴和 Z 轴的设置也相同，见图 4.10。只有 Z 轴丝杠导程为 4mm，所以 Z 轴的脉冲当量为 7500 脉冲/mm。



图 4.10　电机参数的设置

4.2. 设置 EtherCAT 参数

如图 4.11 所示，用鼠标将任务配置中的“EtherCAT_Master.EtherCAT_Task”图标拖入“MainTask”之下，然后将“EtherCAT_Master”删除。如图 4.12 所示。

鼠标双击项目树中的“EtherCAT_Master”，弹出 EtherCAT_Master 参数界面，如图 4.13 所示。

选择“常规卡”，点击“资源地址”后面的“浏览”键，弹出对话框“选择网络适配器”，选中第一个 MAC 地址。再点击“确认”。

可以将循环时间改为 1000 微秒。但不要小于 500 微秒。

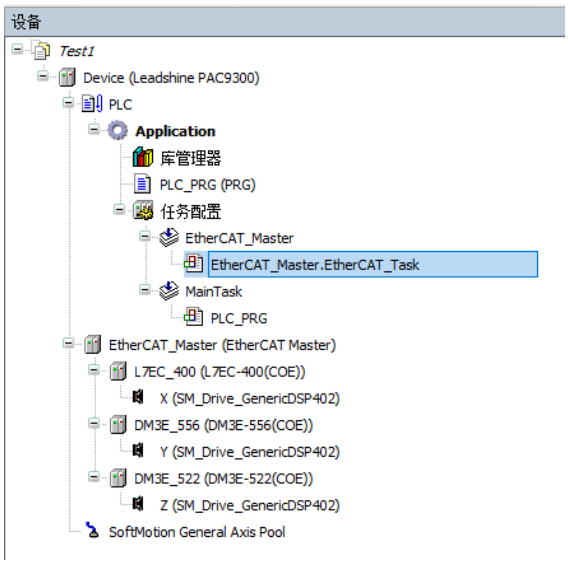

图 4.11　修改任务配置

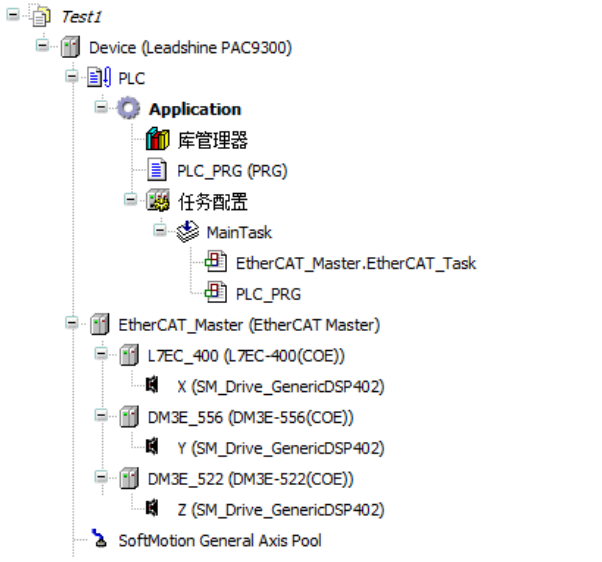

图 4.12　修改完成的任务配置

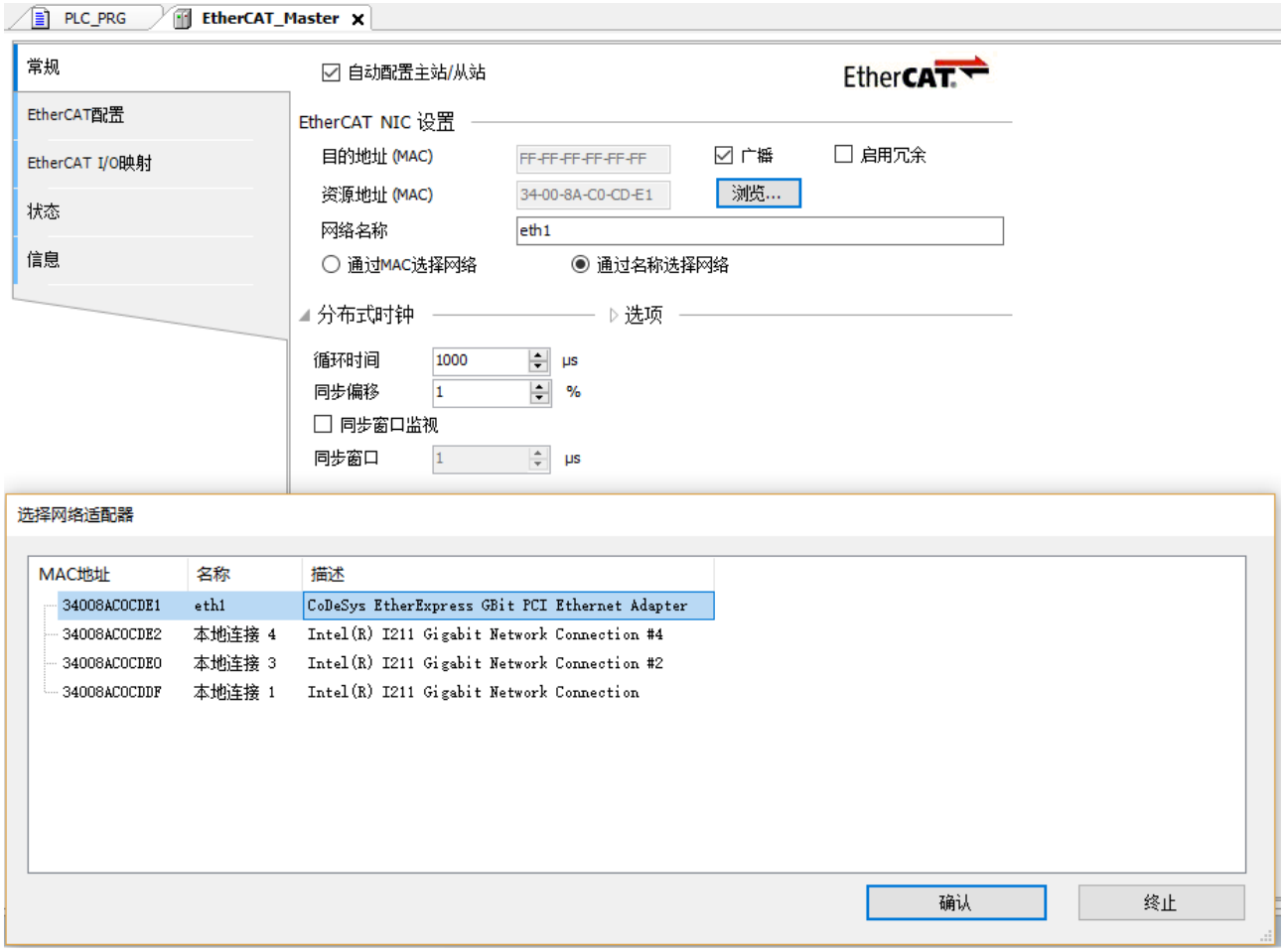

图 4.13　EtherCAT 网络参数的设置

4.3.　IO 的映射与控制（轴使能、JOG 运动模块）

A．数字输入口的映射

首先，在程序中定义输入输出变量如下：

```
PROGRAM PLC_PRG
VAR
    PowerX: MC_Power;          // 定义 X 轴使能模块名称
    JogX: MC_Jog;              // 定义 X 轴的 Jog 运动模块
    JogFdX: BOOL;              // 电机正转信号
    JogBdX: BOOL;              // 电机反转信号
    Input1:BOOL;               // 按键 1
    Input2:BOOL;               // 按键 2
    Output1:BOOL;              // 指示灯 1
    Output2:BOOL;              // 指示灯 2
    OUTenable:BOOL :=TRUE;     // 输出口使能
END_VAR
```

然后，再设置雷赛 L7 交流伺服电机驱动器的参数。该驱动器的数字输入接口电路图如图 4.14 所示。用雷赛 L7 伺服电机调试软件将 Pin2、Pin10、Pin12~18 口定义为数字输入口 SI1、SI5、SI7~13，它们对应于三轴平台上的 9 个按键#1~#9。

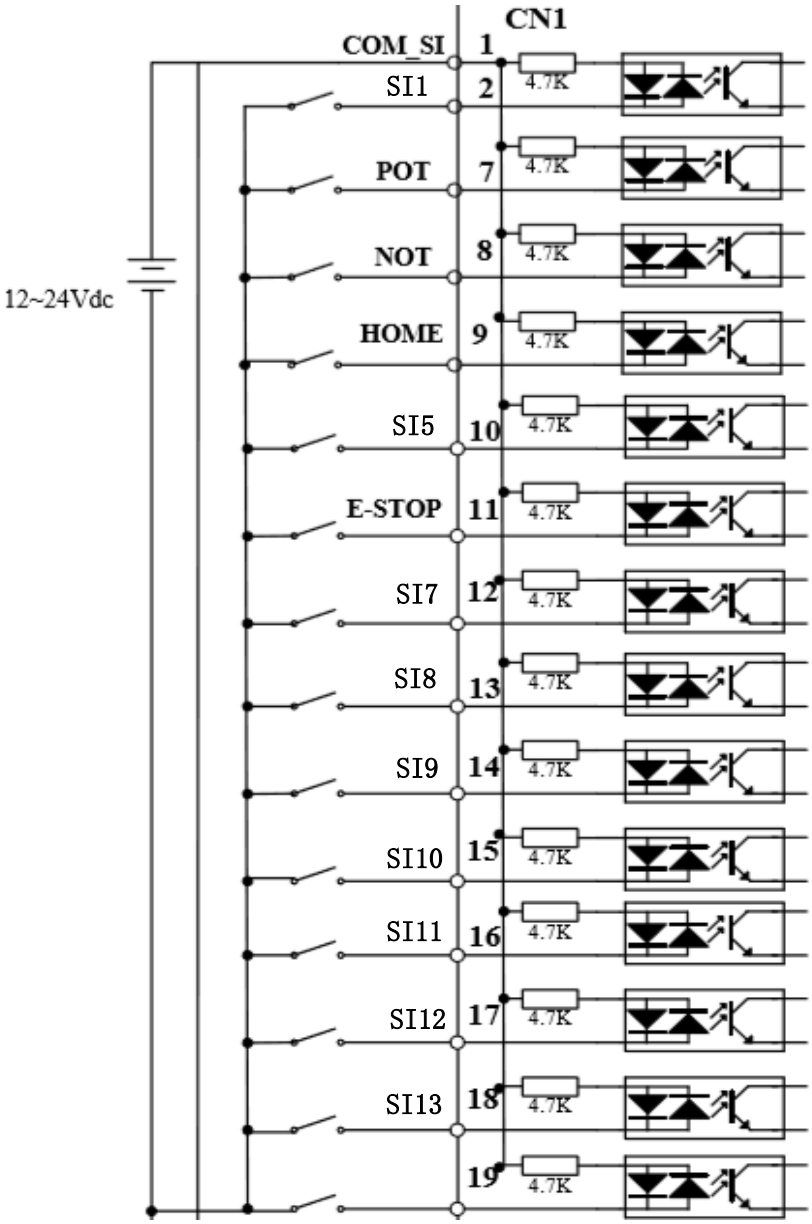

图 4.14　雷赛 L7 交流伺服电机驱动器数字输入接口电路图

IEC61800-7CiA402 标准协议定义的输入 IO 状态映射关系如表 4.1 所示。

表 4.1　数字输入状态映射表

60FD	参数名称	输入 IO 状态映射			关联模式							F
	设定范围			单位		标准出厂设定			对象字典索引			
	60FDh 对象为符合 IEC61800-200 标准的输入 IO 状态映射对象，60FDh 对象的位是按功能定义的。											

Bit31	Bit30	Bit29	Bit28	Bit27	Bit26	Bit25	Bit24
Z 信号	保留	保留	保留	探针 2	探针 1	BRAKE	INP/V-C OIN /TLC
Bit23	Bit22	Bit21	Bit20	Bit19	Bit18	Bit17	Bit16
E-STOP	保留	保留	保留	保留	保留	SI14	SI13
Bit15	Bit14	Bit13	Bit12	Bit11	Bit10	Bit9	Bit8
SI12	SI11	SI10	SI9	SI8	SI7	SI6	SI5
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SI4	SI3	SI2	SI1	保留	HOME	POT	NOT

打开驱动器的“EtherCAT I/O 映射”卡，找到“Digital inputs”，点击前面的“+”号后，在 Bit4 那行的“变量”栏中双击鼠标，然后点击“…”符号，如图 4.15 所示。弹出对话框如图 4.16 所示。

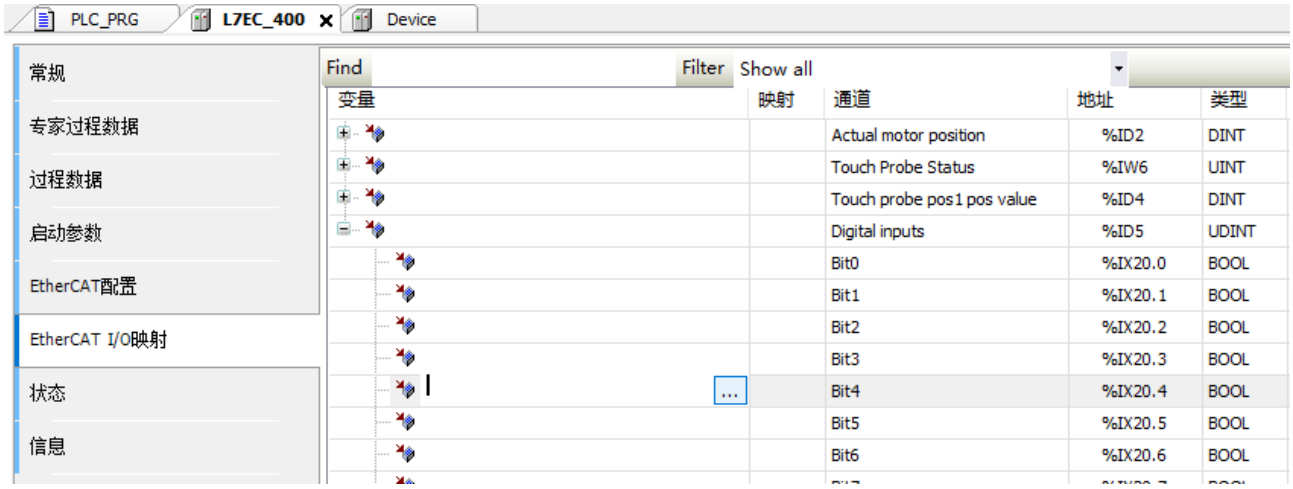

图 4.15　映射输入变量

打开“Application”、“PLC_PRG”，选中“Input1”，再点“确定”。完成输入口 SI1 的映射。然后，在 Bit8 行，添加 Input2。结果如图 4.17 所示。

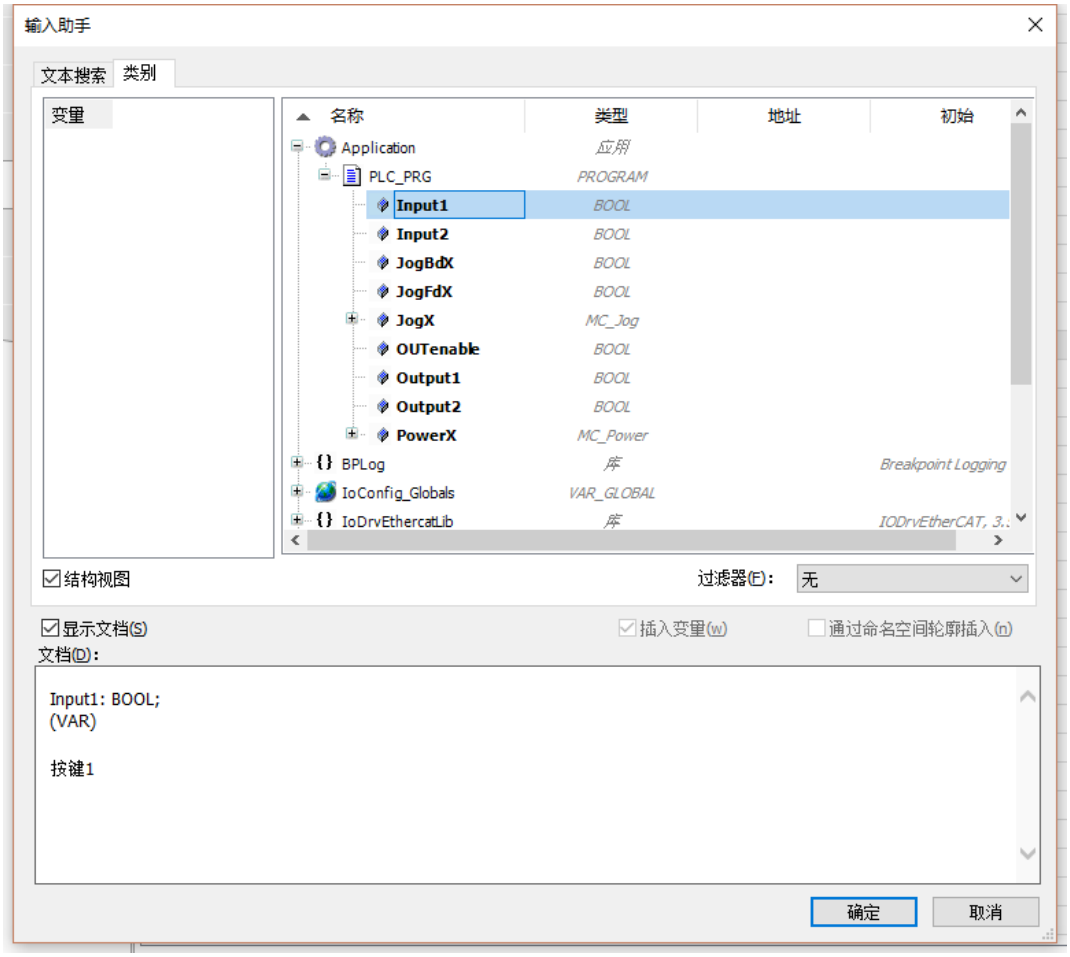

图 4.16　选择输入变量

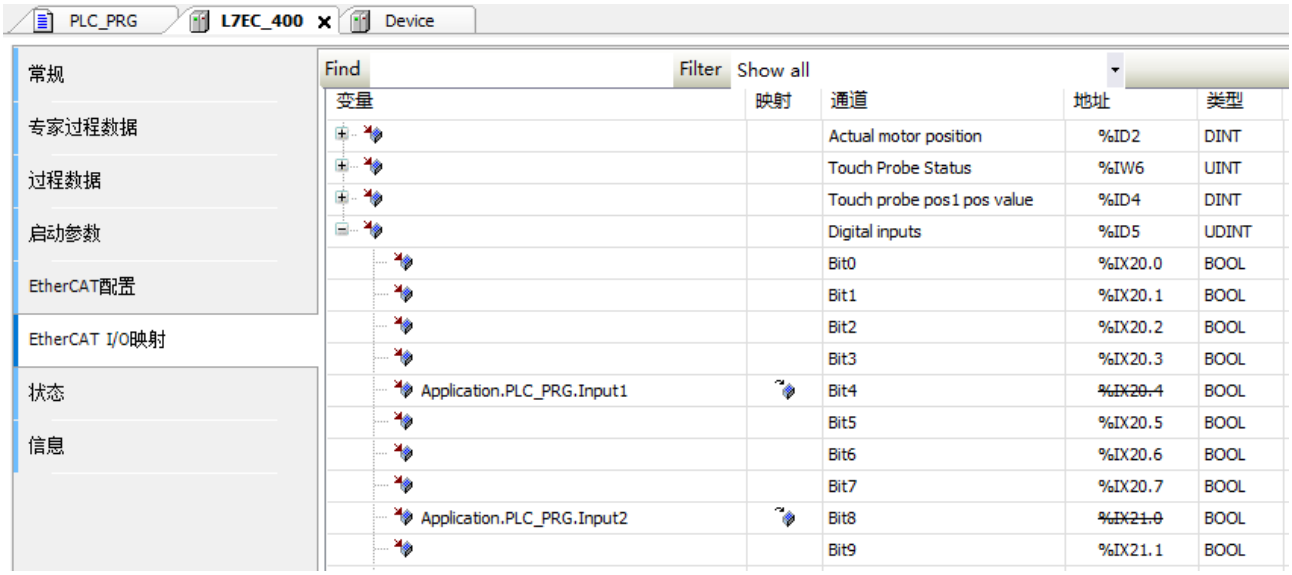

图 4.17　完成输入变量的映射

B. 数字输出口的映射

雷赛 L7 交流伺服电机驱动器的数字输出接口电路图如图 4.18 所示。用雷赛 L7 伺服电机调试软件将 Pin33、Pin32、Pin34 口更改为数字输出口 SO1、SO2、SO3，它们对应于三轴平台上的指示灯 1、2、3。

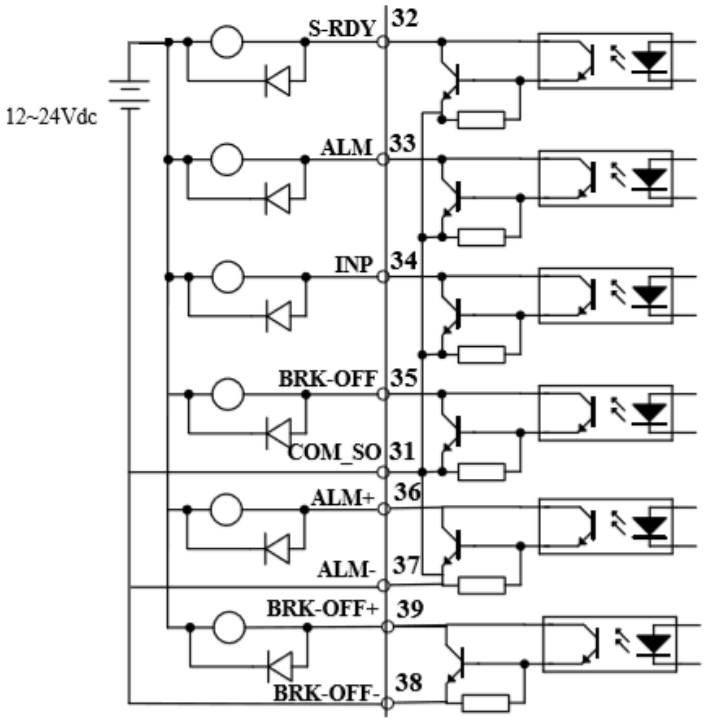

图 4.18　雷赛 L7 数字输出接口电路图

打开 L7 驱动器的参数设置界面中的“常规”卡，在“使能专家设置”打上勾，如图 4.19 所示。

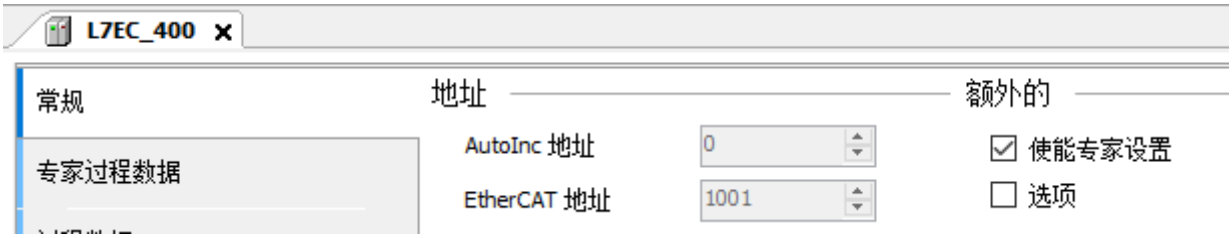

图 4.19　选中“使能专家设置

再打开 L7 驱动器的参数设置界面中的“专家过程数据卡”，如图 4.20 所示。“同步管理器”中选“输出”；PDO 列表中选“16#1600”；PDO 分配中选“16#1600”。在“PDO 内容”中点击“插入”。弹出如图 4.21 所示的对话框。

找到 16#60FE16:#01，选中 Physical outputs，点击“确定”；同样方法，找到 16#60FE16:#02，插入 Bit mask。结果如图 4.22 所示。

IEC61800-7CiA402 标准协议定义的数字输出的对象 IO 状态映射关系如表 4.2 所示。要使用 SO1、SO2 和 SO3，首先要打开输出口的使能，即把 Bit mask 对应位设为 TRUE。然后，在 Physical outputs 中映射输出变量。方法与输入变量映射相同，结果如图 4.23 所示。

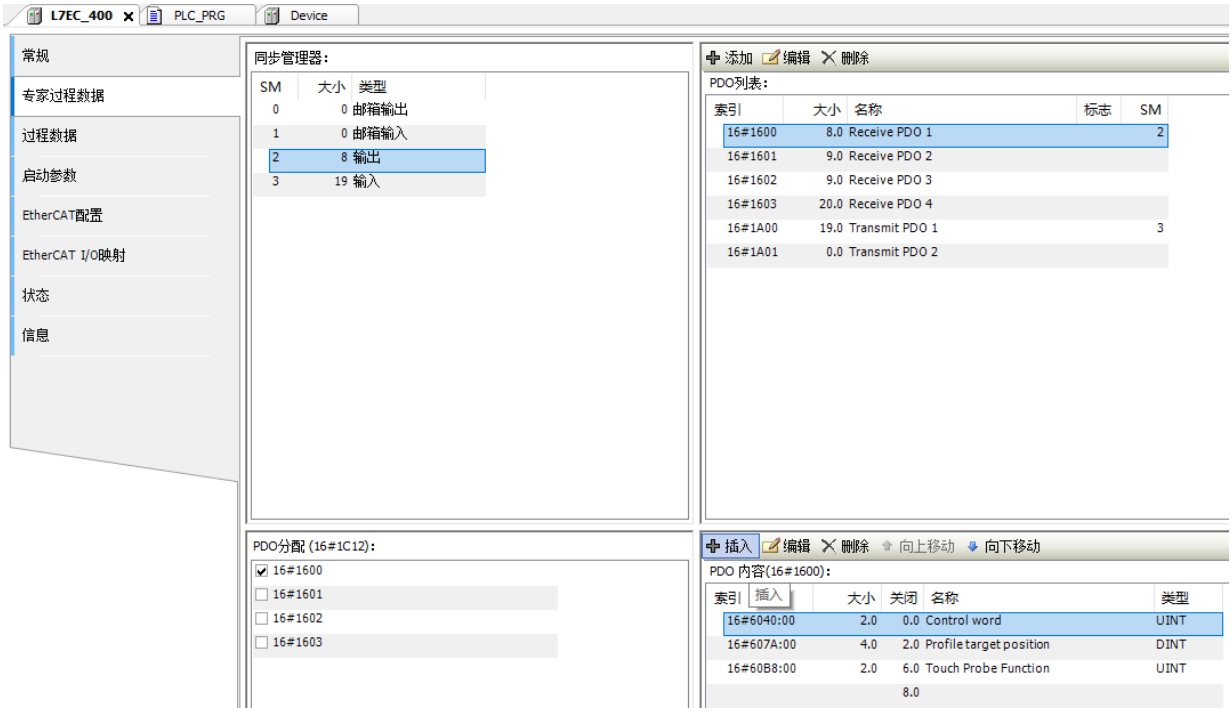

图 4.20　插入 PDO 数据

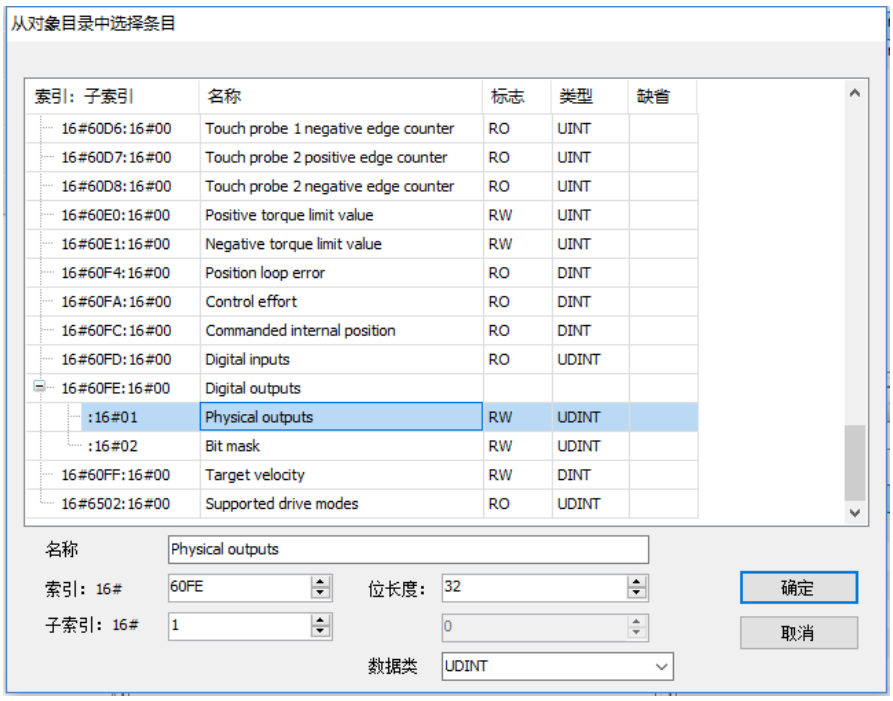

图 4.21　插入 Physical outputs、Bit mask

表 4.2　数字输出的 PDO 数据

子索引 \ 位	31~21	21	20	19	18	17	16	15~0
PhysicalOutputs 60FE.01	保留	SO6 有效	SO5 有效	SO4 有效	SO3 有效	SO2 有效	SO1 有效	保留
Bit Masks 60FE.02		SO6 使能	SO5 使能	SO4 使能	SO3 使能	SO2 使能	SO1 使能	



图 4.22　完成 Physical outputs、Bit mask 的插入

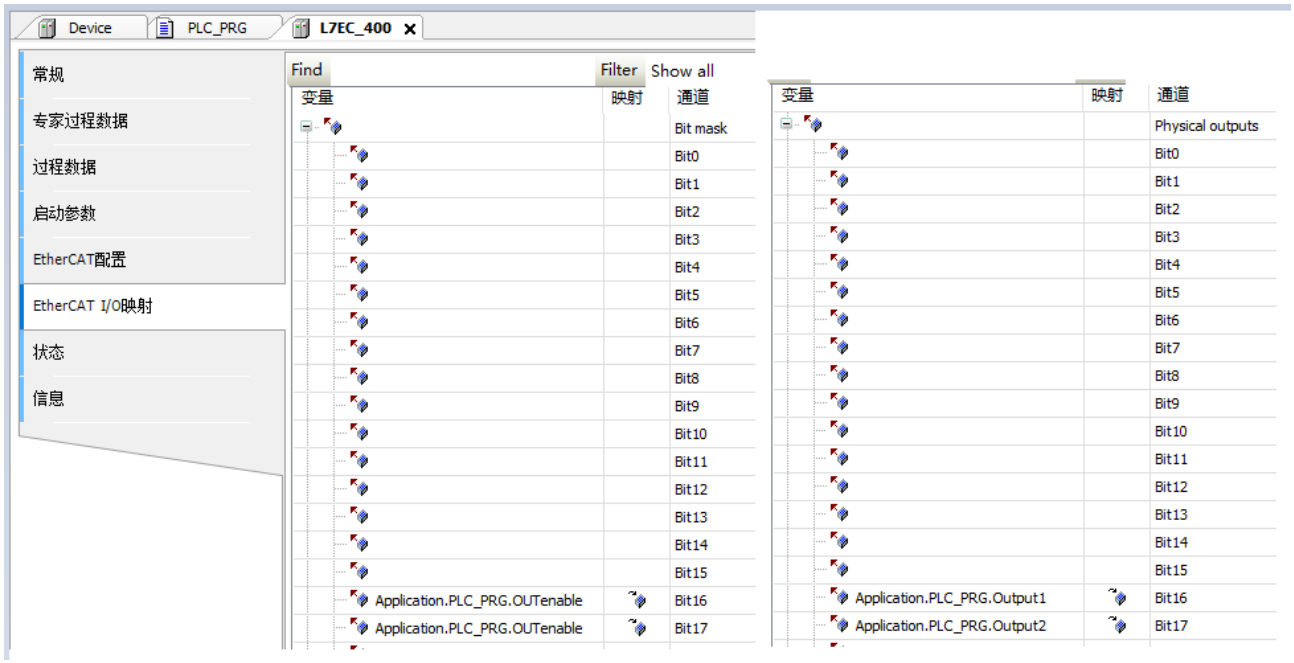

图 4.23　输出使能和输出变量的映射

C．实现例程二的程序

将如下程序代码输入至 PLC_PRG；编辑、下载、运行；即可实现例程二的功能。

```
IF   Input1=TRUE   THEN          // 如果按键 1 按下
     JogFdX:=TRUE;               // 电机正转
     Output1 :=TRUE;             // 开指示灯 1
   ELSE
     JogFdX:=FALSE;              // 否则，电机不可正转
     Output1 :=FALSE;            // 关指示灯 1
END_IF

IF   Input2=TRUE   THEN          // 如果按键 2 按下
      JogBdX:=TRUE;              // 电机反转
      Output2 :=TRUE;            // 开指示灯 2
    ELSE
      JogBdX:=FALSE;             // 否则，电机不可反转
      Output2 :=FALSE;           // 关指示灯 2
END_IF
```

```
PowerX(Axis:= X,Enable:= TRUE, bRegulatorOn:= TRUE,bDriveStart:=TRUE,
        Status=>,bRegulatorRealState=>,bDriveStartRealState=>,Busy=>,Error=>,ErrorID=>);
         // 调用电机使能模块
JogX(Axis:=X,JogForward:=JogFdX,JogBackward:=JogBdX,Velocity:=5,Acceleration:=30,
       Deceleration:=30,Jerk:=,Busy=>,CommandAborted=>,Error=>,ErrorId=>);
      // 调用 X 轴 Jog 运动模块
```

代码中调用了 2 个模块。电机运动前，必须调用电机使能模块。Jog 运动模块相当于雷赛运动控制卡中的匀速运动函数，只控制运动速度，不控制运动距离。

2 个模块的参数定义参见产品手册。也可以用鼠标右键在变量定义栏中点击模块名称，然后点“浏览”、“转到定义处”，就可以查看该模块的参数定义。如图 4.24、4.25 所示。

注意：调用模块时，先要在变量定义段里为模块定义一个模块名；然后在代码段里用该模块名调用模块。



图 4.24　查看模块的定义

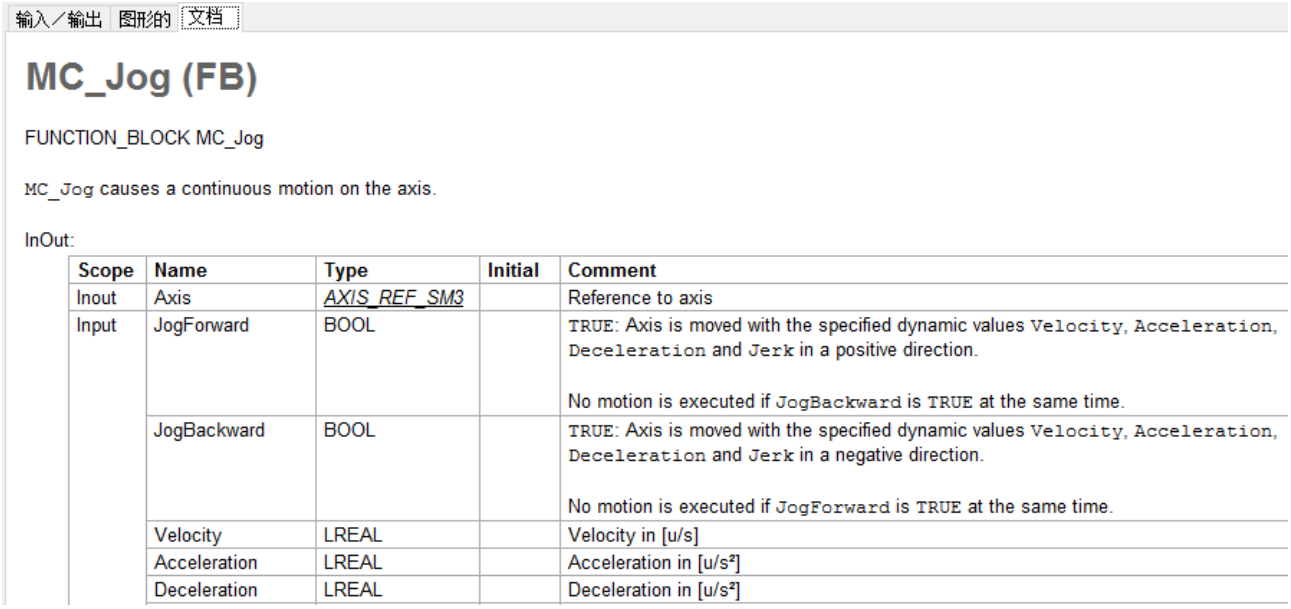

图 4.25　模块的定义

输入模块时，可使用“输入助手”。在编辑区点鼠标右键，弹出对话框如图 4.26 所示。选“输入助手…”。之后，弹出对话框如图 4.27 所示。选“文本搜索”，然后输入模块名称，如：MC_Power，下面会出现搜索结果，选择所需模块，点“确定”后得 MC_Power 模块的代码，如图 4.28 所示，然后修改模块的名称、设置其参数即可。



图 4.26　选“输入助手”

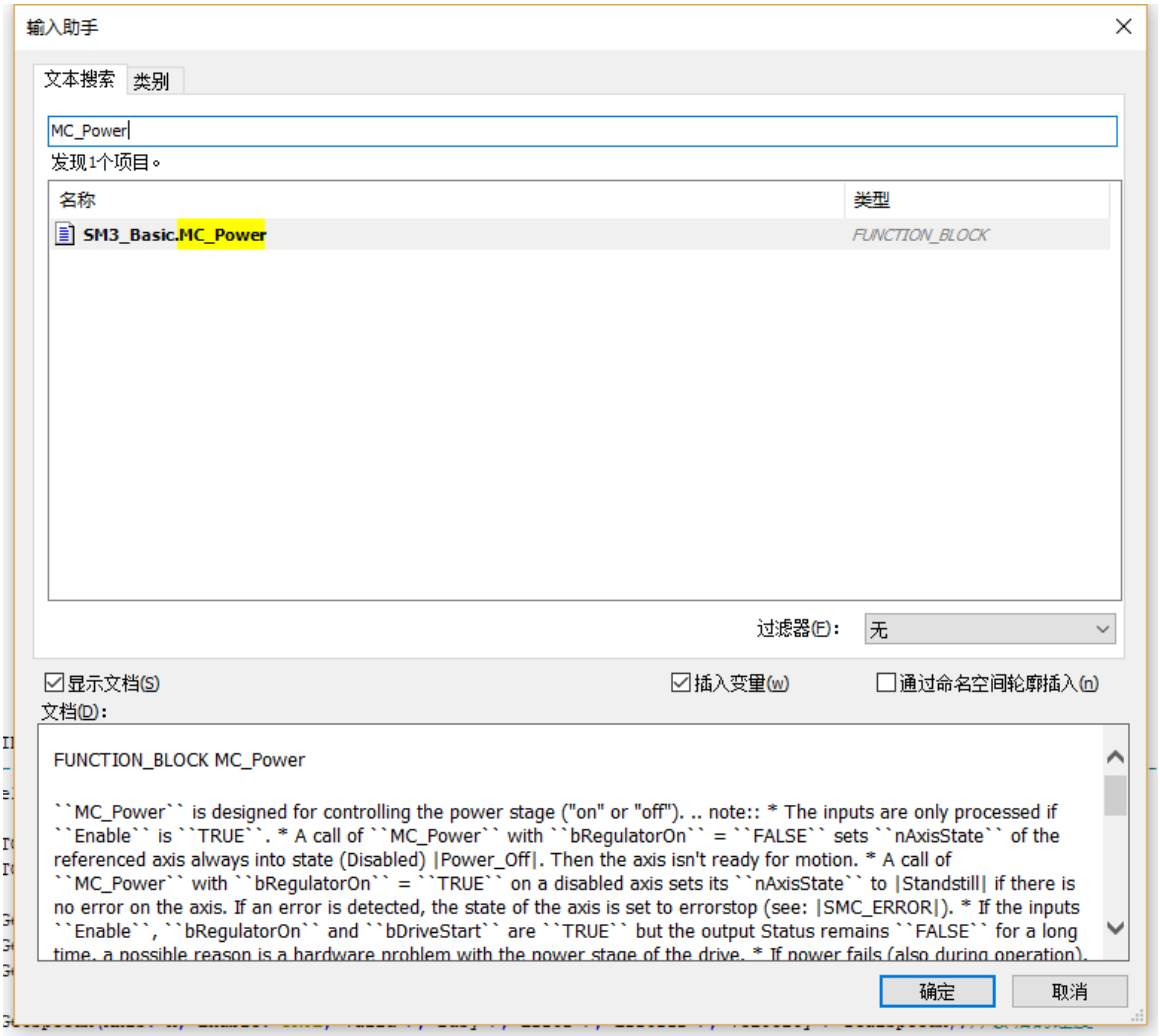

图 4.27　“输入助手”的对话框

PLC_PRG

```
1  PROGRAM PLC_PRG
2  VAR
3      PowerX: MC_Power;        // 定义轴使能模块名称
```

```
1
2  MC_Power(
3      Axis:= ,
4      Enable:= ,
5      bRegulatorOn:= ,
6      bDriveStart:= ,
7      Status=> ,
8      bRegulatorRealState=> ,
9      bDriveStartRealState=> ,
10     Busy=> ,
11     Error=> ,
12     ErrorID=> );
13
```

图 4.28　由“输入助手”得到的模块代码

第 5 章　可视化界面的设计

5.1. 指示灯与按键的设计

给例程二设计一个可视化界，如图 5.1 所示。步骤如下：

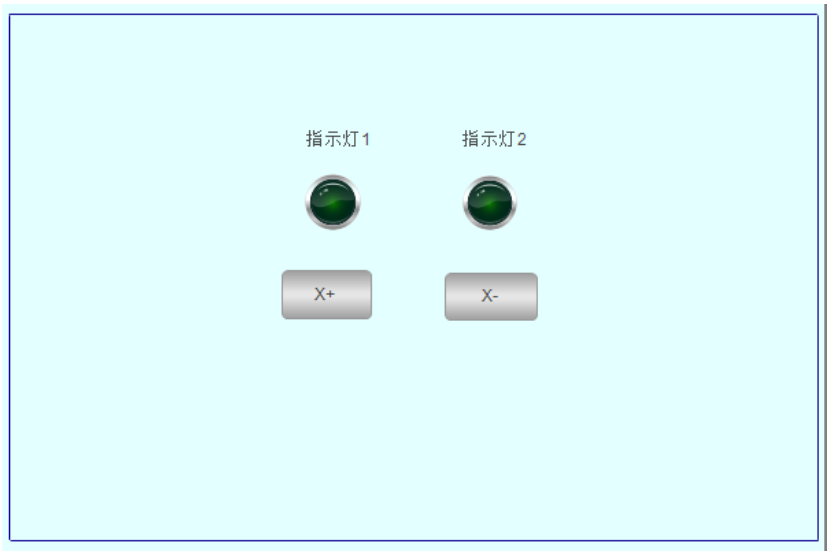

图 5.1　例程二的可视化界面

A. 添加视图管理器

如图 5.2 所示，用鼠标右键点击项目树中的“Application”，选“添加对象”，再选“视图管理器”。弹出如图 5.3 所示的对话框，点击“激活”下面的方框。然后点击“打开”。

视图管理器 VisualizationManager 的界面如图 5.4 所示。在“设置”卡中，点击“使用 Unicode 字符串”前的方框，将其选中；在“选择的语言”后的下拉菜单中选择“zh-CHS”，即简体中文。

B. 添加视图

添加视图的方法和添加视图管理器的方法一样，参见图 5.2。视图的名称可以自己定义，如图 5.5 所示。

C. 设置目标视图和网络视图的参数

用鼠标左键双击项目树中 VisualizationManager 下的“目标视图”，将其界面打开，如图 5.6 所示。

点击“起始视图”后的“…”键，选择视图名称为刚刚添加的视图名，Visualization；视图的更新速率选为 200 毫秒；“最佳匹配”下选“固定的”，视图的宽度和高度分别选 600 和 400 像素；“默认文本输入”，选“键盘”。

同样，设置“网络视图”的参数，如图 5.7 所示。

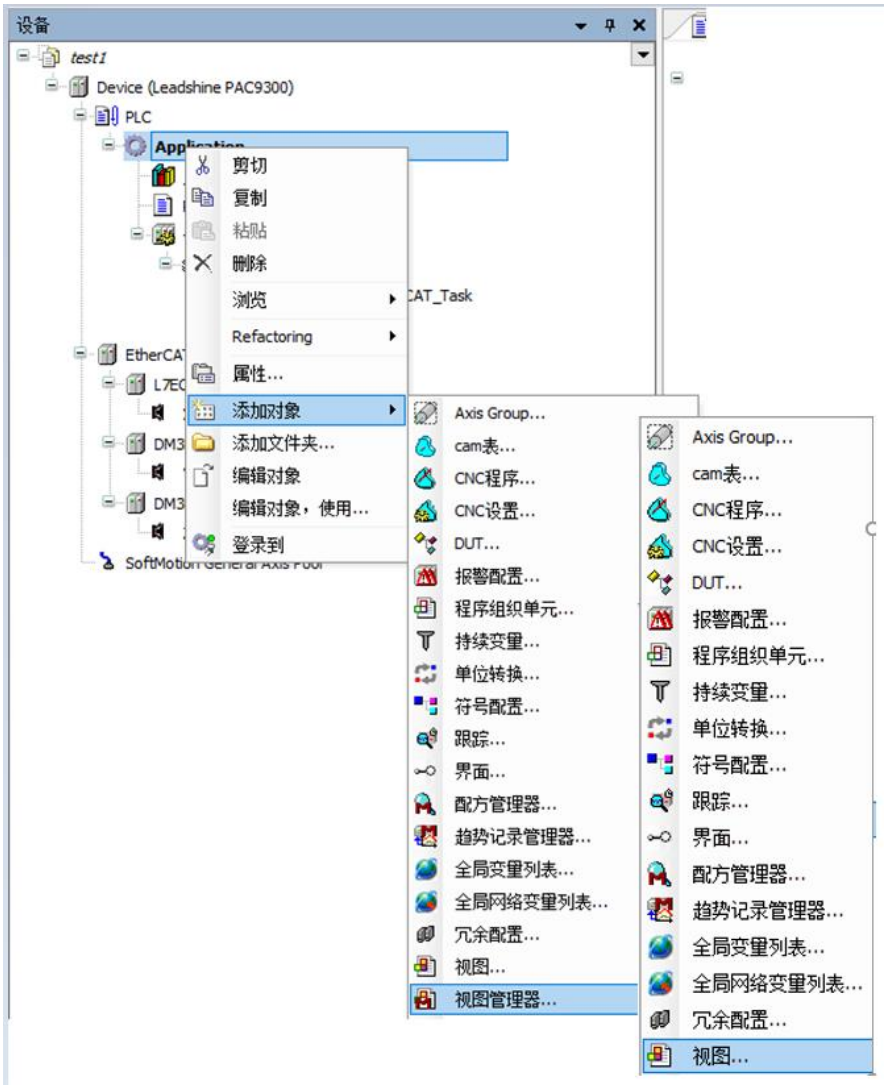

图 5.2　添加视图管理器和视图

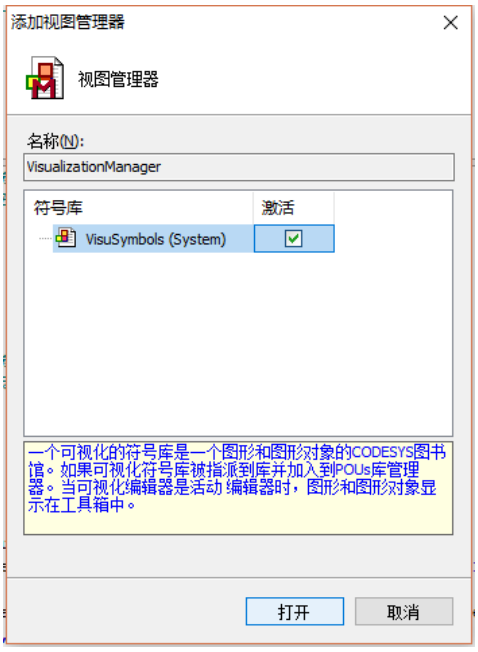

图 5.3　打开视图管理器



图 5.4　设置视图管理器中的参数

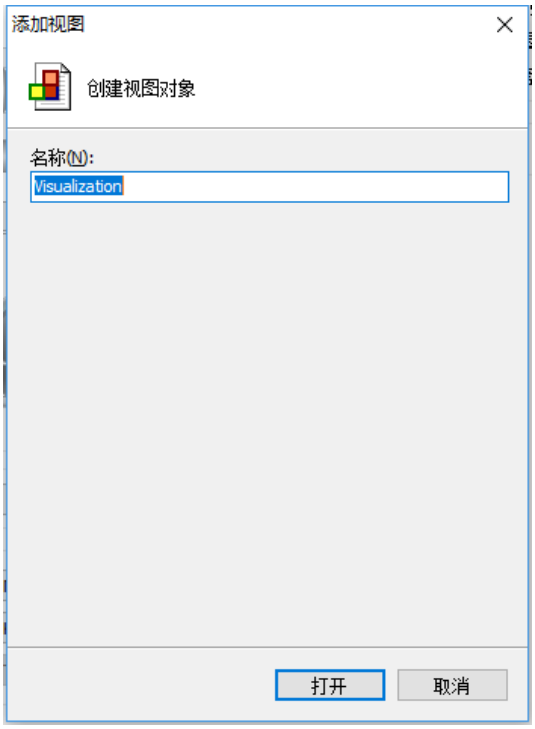

图 5.5　设置视图的名称



图 5.6　设置目标视图的参数

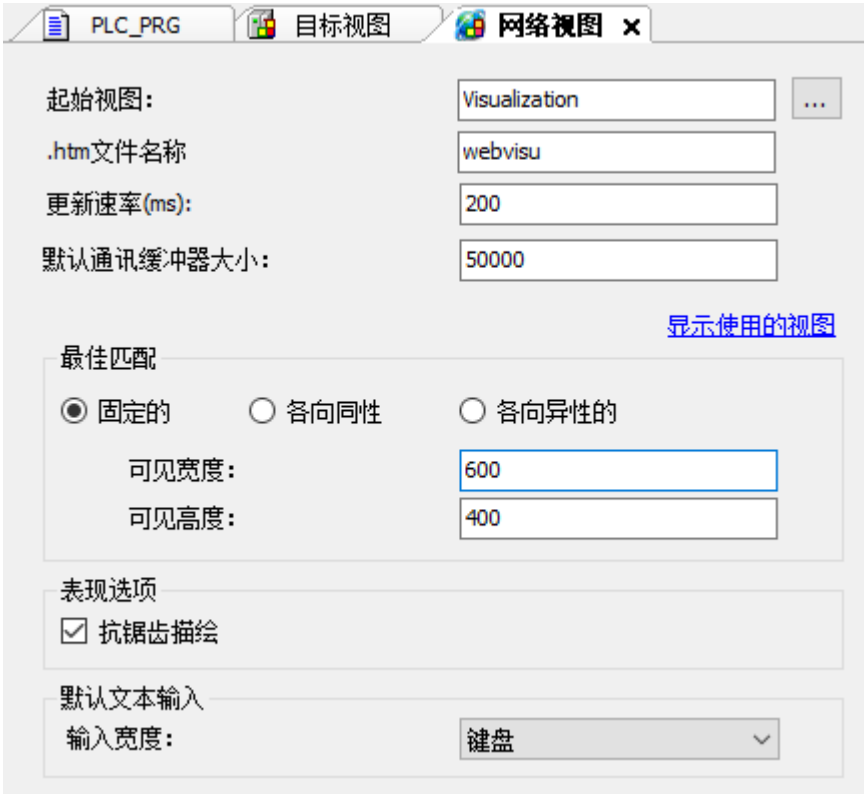

图 5.7　设置网络视图的参数

D. 设计可视化界面

在项目树中用鼠标左键双击视图名 Visualization，将其界面打开如图 5.8 所示。

用鼠标右键点击界面后，选“背景”，弹出对话框如图 5.9 所示。可选择背景颜色。

在“工具箱”栏下选择“基本的”工具，可以画线段和图形。

如画直线：在工具箱中选中“Line”，用鼠标将线段拖入界面，再用鼠标调整线段的起点和终点；然后，在 Line 的属性框中选“Color”，可设置线段的颜色；在“Element look”下，可修改线宽 Line width、修改线型 Line style 为实线、虚线或点划线。如图 5.8 所示。

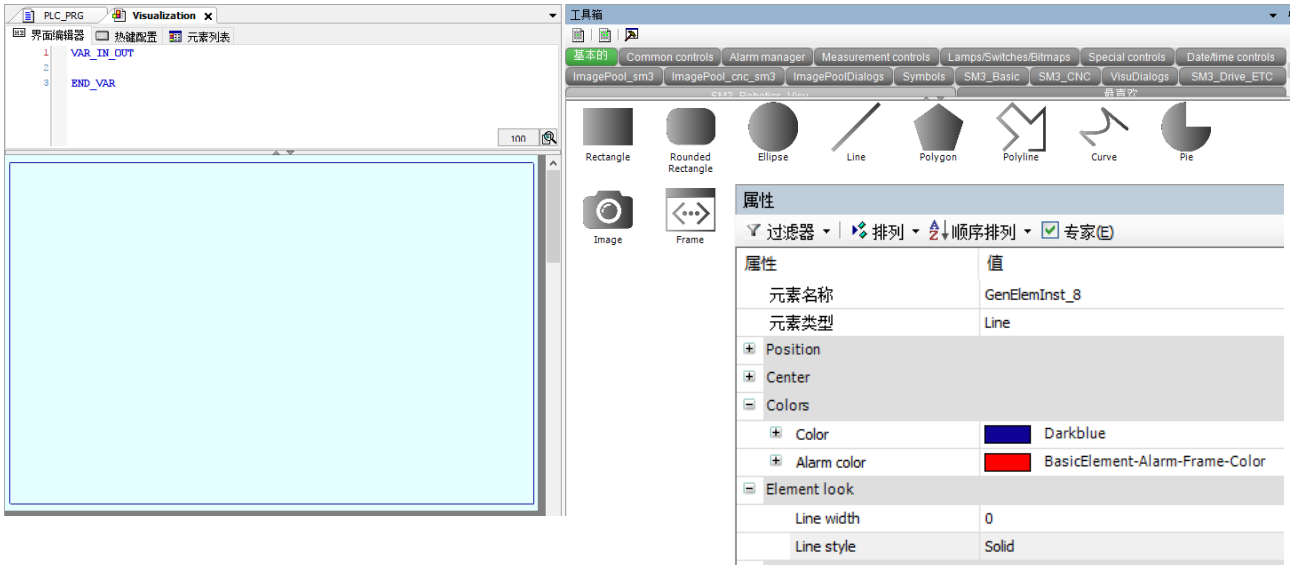

图 5.8　设置可视化界面、画线段

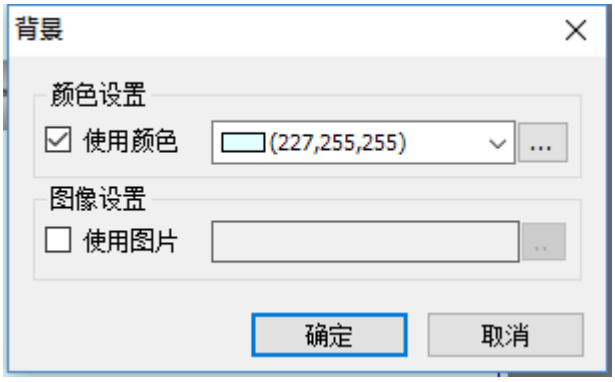

图 5.9　设置可视化界面的背景色

E. 添加指示灯控件

在“工具箱”栏下选择“Lamps/Swithches/Bitmaps”（灯/开关/位图）工具，然后选中“Lamp1”，用鼠标将该指示灯拖入界面，再用鼠标调整至合适的大小和位置。

然后，在 Lamp1 的属性表中找到“Variable”，点击其后的框，出现一个“…”键，点击后出现对话框，选 Application、PLC_PRG 中的变量 Output1。这将该指示灯与变量 Output1 联系起来。Output1=TRUE 时，指示灯亮；Output1=FALSE 时，指示灯灭。

在 Lamp1 的属性表中找到“Background”其下的“Image”，点击其后的框，可在下拉菜单中选择指示灯的颜色。如图 5.10 所示。

同样，再添加一个指示灯，变量设为 Output2。

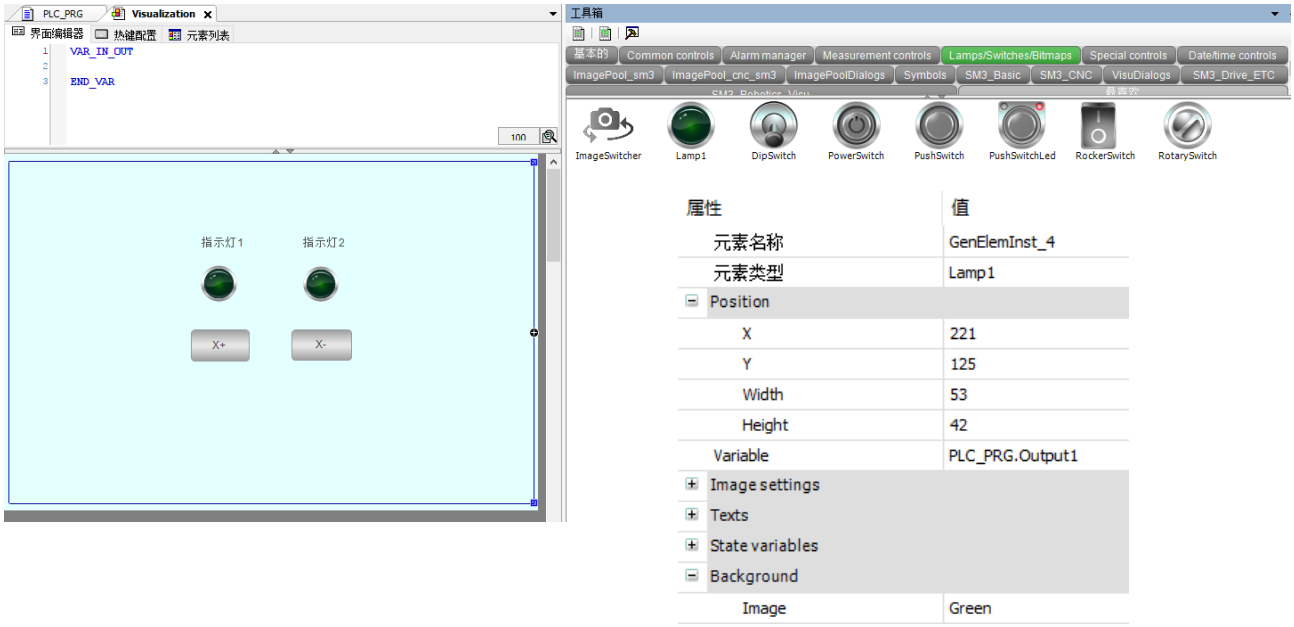

图 5.10　添加指示灯控件

F. 添加按键控件

在添加按键之前，要在 PLC_PRG 的变量定义栏中定义 2 个按键的变量：

```
ButtonXfd :BOOL;  // X 轴前进键
ButtonXbd :BOOL;  // X 轴后退键
```

如图 5.11 所示，在“工具箱”栏下选择“Common controls”（普通控件）工具，然后选中“Button”，用鼠标将该按键拖入界面，再用鼠标调整至合适的大小和位置。

在 Button 属性表中找到“Texts”，在其下面的 Text 栏中写入按键上面显示的字符，“X+”。

在属性表的“输入配置”中，找到“Tap”（无自锁的按键），在其下面的“Variable”后面添加变量 ButtonXfd。

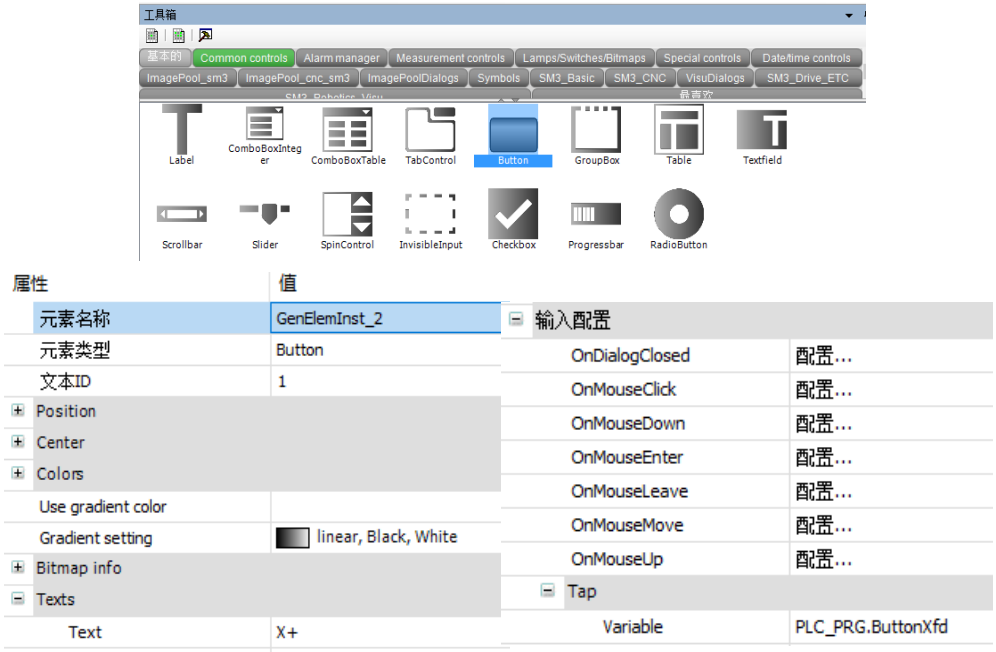

图 5.11　添加按键控件

注意：Tap 按键　--鼠标键按下，变量为 True；抬起为 False。
　　Toggle 按键--鼠标键按下，变量为 True，抬起还为 True；再次按下为 False。
　　即为带自锁的按键。

同样，再添加一个按键，变量设置为 ButtonXbd。如图 5.10 所示。

将 PLC_PRG 程序中的 2 条 IF 语句稍作修改如下：

```
IF  Input1=TRUE OR ButtonXfd=TRUE THEN        // 如果按键 1 按下或 X 轴前进键按下
IF  Input2=TRUE OR ButtonXbd=TRUE THEN        // 如果按键 2 按下或 X 轴后退键按下
```

G. 添加标签控件

在“Common controls”工具中选中“Label”，用鼠标将标签拖入界面，再用鼠标调整至合适的大小和位置；在其中间输入字符即可。如图 5.1 中的“指示灯 1”、“指示灯 2”。

完成以上操作后，编译、下载并运行程序，图 5.1 所示的界面就在 PAC9300 运动控制器连接的显示器上出现，用鼠标点击 X+、X-按键，同样可使 X 轴做 Jog 运动。

5.2. 文本框的设计 (读位置模块、相对点位运动模块)

如图 5.12 所示，给例程二添加显示平台移动位置和相对点位运动的功能，使用文本框显示变量、用文本框输入变量。



图 5.12　添加文本框、按键控件

A. 文本框显示变量

读取轴当前位置的模块为 MC_ReadActualPosition，使用前要定义变量如下：

```
GetPositionX: MC_ReadActualPosition;
PositionX: LREAL;
```

调用该模块的代码如下：

```
GetPositionX(Axis:=X,Enable:=TRUE,Valid=>,Busy=>,Error=>,ErrorID=>,
             Position=>PositionX);    //读轴的位置
```

在“工具箱”栏下选择“Common controls”（普通控件）工具，然后选中“Textfield”，

用鼠标将文本框拖入界面，再用鼠标调整至合适的大小和位置。

在 Textfield 属性表中找到“Texts”，在其下面的 Text 栏中写入%3.3f，意思为整数位 3 位，小数位 3 位；在“Text variable”后加入 PLC_PRG 的变量 PositionX。如图 5.13 所示。

属性	值
元素名称	GenElemInst_44
元素类型	Textfield
文本ID	12
⊞ Position	
⊞ Colors	
⊞ Element look	
Shadow type	From style
⊟ Texts	
Text	%3.3f
Tooltip	
⊞ Text properties	
⊟ Text variables	
Text variable	PLC_PRG.PositionX

图 5.13　文本框的属性表

B. 文本框输入变量

相对点位运动的模块为 MC_MoveRelative。使用前要定义变量如下：

```
MoveX: MC_MoveRelative;
StartX: BOOL;
DistanceX: LREAL;        // 运动距离
SpeedX: LREAL := 10;     // 运动速度为 10mm/s
DoneX: BOOL;
ButtonXrun :BOOL;        // X 轴启动键
```

在程序中添加代码如下：

```
IF  ButtonXrun = TRUE  THEN   // X 轴启动键按下
     StartX:=TRUE;            // X 轴开始相对点位运动
END_IF

IF DoneX = TRUE THEN          // X 轴停止
   StartX:=FALSE;             // 禁止点位运动
END_IF
MoveX(Axis:=X,Execute:=StartX,Distance:=DistanceX,Velocity:=SpeedX,
      Acceleration:=30,Deceleration:=30,Jerk:=,Done=>DoneX,Busy=>,
      CommandAborted=>,Error=>,ErrorID=>); // 调用点位运动模块
```

在“工具箱”栏中选取“Textfield”，用鼠标将文本框拖入界面，再用鼠标调整至合适的大小和位置。

在 Textfield 属性表中找到“Texts”，在其下面的 Text 栏中写入%3.3f；在“Text variable”后加入 PLC_PRG 的变量 DistanceX。在“输入配置”下面，找到“OnMouseClick”，点击其后的“配置…”，弹出对话框如图 5.14 所示。在左边框中选中“写变量”，点击“>”键，然后，点击“确定”。

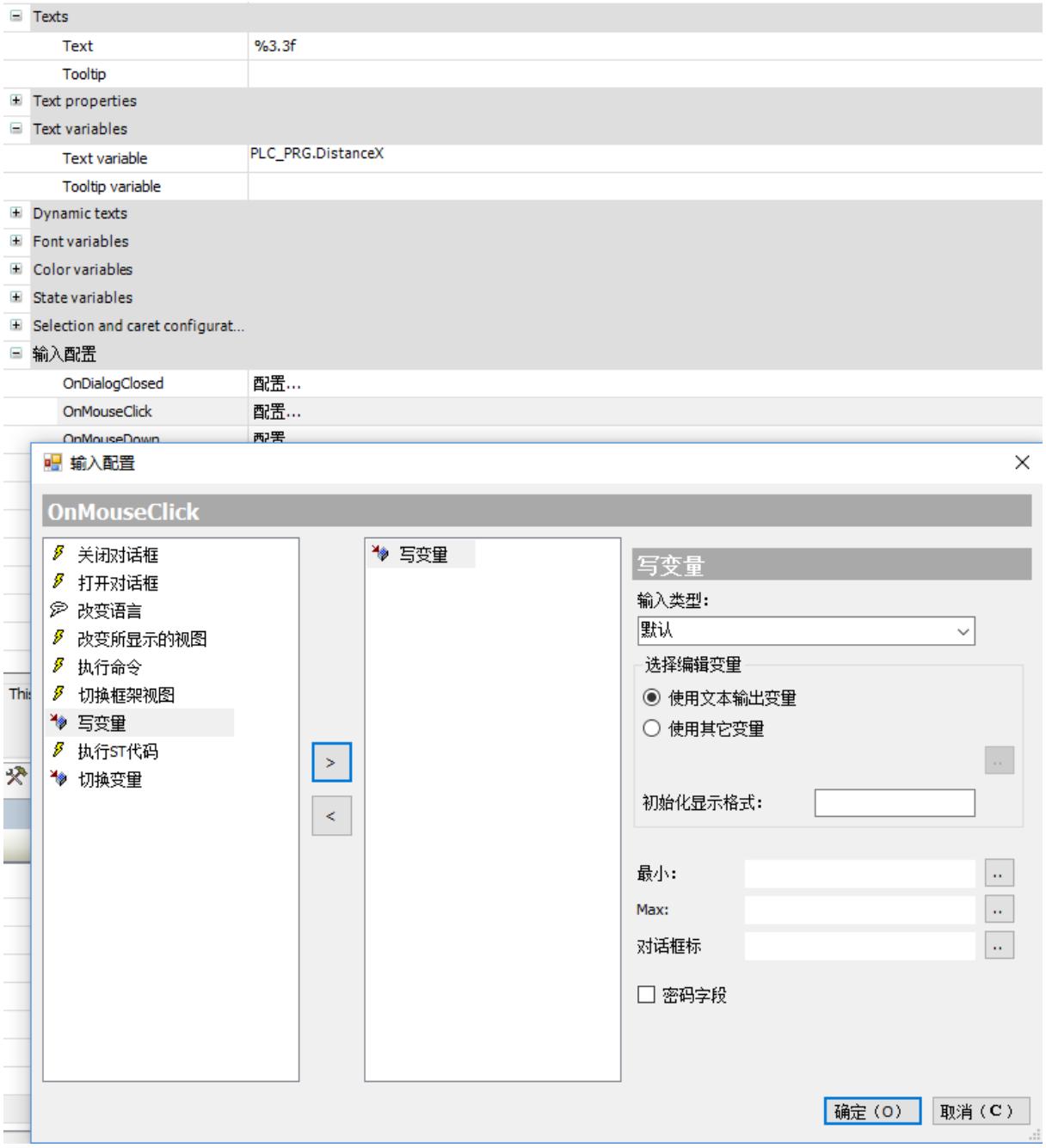

图 5.14　文本框的输入配置

添加一个 Tap 型的按键“确定”，其变量为 ButtonXrun。最后，按图 5.12 所示，添加几个标签。

完成以上操作后，编译、下载并运行程序，用鼠标点击“X 轴运动距离：”后的文本框，用键盘输入-23.433，鼠标点击“确定”，X 轴即向反方向移动 23.433 毫米；移动过程中，“X 轴当前位置：”后的文本框不断显示平台的新位置。

注意：每执行一遍程序，都会读取一次 X 轴的当前位置；执行一次程序的时间远小于 EtherCAT 的循环时间（1000 微秒），如图 4.12 所示。但是，可视化界面上的 X 轴的当前位置是每 200 毫秒刷新一次，如图 5.6 所示。

5.3. 组合框和速度表的设计（定时器、读速度模块、回原点模块）

例程三：手动调整运动平台三个轴的位置且速度可调。其界面如图 5.15 所示。该程序每 50 毫秒更新一次界面的数据，这样可以提高程序的运行效率。

下面通过例程三介绍组合框的设计方法和定时器、读速度模块、回原点模块的使用方法。



图 5.15　例程三的界面

A. 数组型组合框

参照例程二的步骤，新建工程、设置电机驱动器参数、EtherCAT 网络参数、添加可视化界面。显示三个轴的当前位置的方法和例程二相同，这里只是用了一个 GroupBox 控件将三个文本框及标签圈在一起。

在做 JOG 运动、点位运动、回原点运动时，必须先选择轴号，才能进行相应的单轴运动。采用数组型组合框选择轴号。

首先，为数组型组合框定义 2 个变量，代码如下所示：

```
AxisNum : INT :=1;                                    // 轴号，默认轴为 X
AxisName:ARRAY [1..3] OF STRING:= ['X','Y','Z'];  // 组合框显示的数组
```

在“工具箱”栏下选择“Common controls”工具，然后选中“ComboBoxTable”，用鼠标将该组合框拖入界面，再用鼠标调整至合适的大小和位置。

然后，在 ComboBoxTable 的属性表中找到“Variable”，添加变量 AxisNum；在“Data array”中添加数组 AxisName。再将“Number visible rows（可见行数）”改为 3。如图 5.16 所示。

程序运行中，选 Y，则 AxisNum = 2；选 Z，则 AxisNum = 3。

注意：为数组型组合框定义的数组中不可出现汉字，否则，显示不正常。

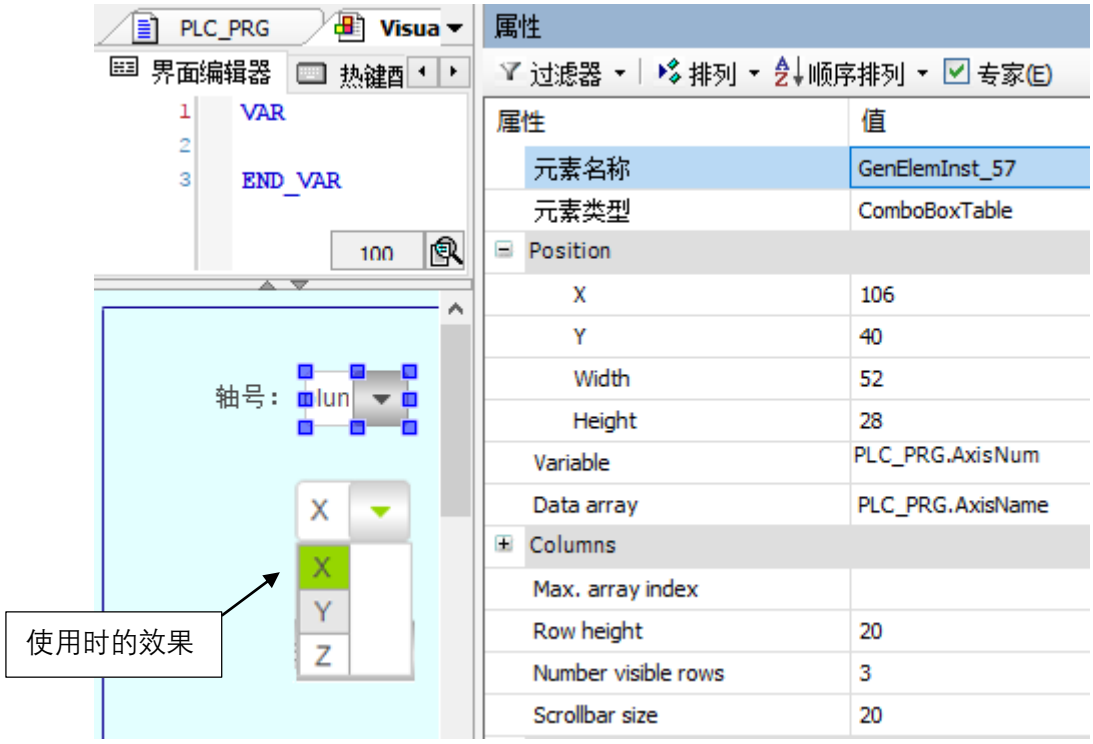

图 5.16　数组型组合框的设置及使用效果

B. 整数型组合框

首先，为整数型组合框定义 1 个变量，代码如下所示：

```
SpeedRatio: INT := 1;   // 速度倍率
```

在“Common controls”中选“ComboBoxInteger”，用鼠标将该组合框拖入界面，再用鼠标调整至合适的大小和位置。

然后，在 ComboBoxInteger 的属性表中找到“Variable”，添加变量 SpeedRatio；将“Settings of the list”中的“Number visible rows”改为 3；再将“Value range（取值范围）”下的“Minimum value”改为 1，“Maximum value”改为 3。如图 5.17 所示。

程序运行中，变量 SpeedRatio 的值等于整数型组合框所选数值。

C. 长方形仪表盘

使用一个仪表盘显示运动平台的合速度 absSpeed。3 个轴的合速度的计算公式如下：

$$absSpeed = (Vx^2+Vy^2+Vz^2)^{0.5}$$

在“Measurement controls”中选“BarDisplayImage”，用鼠标将该仪表盘拖入界面，再用鼠标调整至合适的大小和位置。

然后，在其属性表中找到“Value”，添加变量 absSpeed；将“Bar”中的“Diagram type”改为 Scale in bar。将“Scale”中的“Scale start”改为 0，将“Scale end”改为 10，将“Main scale”改为 10，将“Sub scale”改为 2。如图 5.18 所示。

D. 程序设计

由于可视化界面 200ms 才刷新一次，因此，用定时器控制时间 50ms 处理一次和界面有关的参数即可。

On-Delay 型定时器由上升沿触发开启，所以，代码中要先关定时器再开定时器。代码如下：

```
TON1(IN:=FALSE,PT:=T#50MS,Q=>Delay1Off,ET=>); // 关定时器
TON1(IN:=TRUE,PT:=T#50MS,Q=>Delay1Off,ET=>);  // 开定时器
```

注意:回原点模块中的速度、加速度的单位为脉冲/秒、脉冲/秒 ^{2}。其中 Home_Pos、HomeAxisNo 这 2 个参数设为 0 即可。

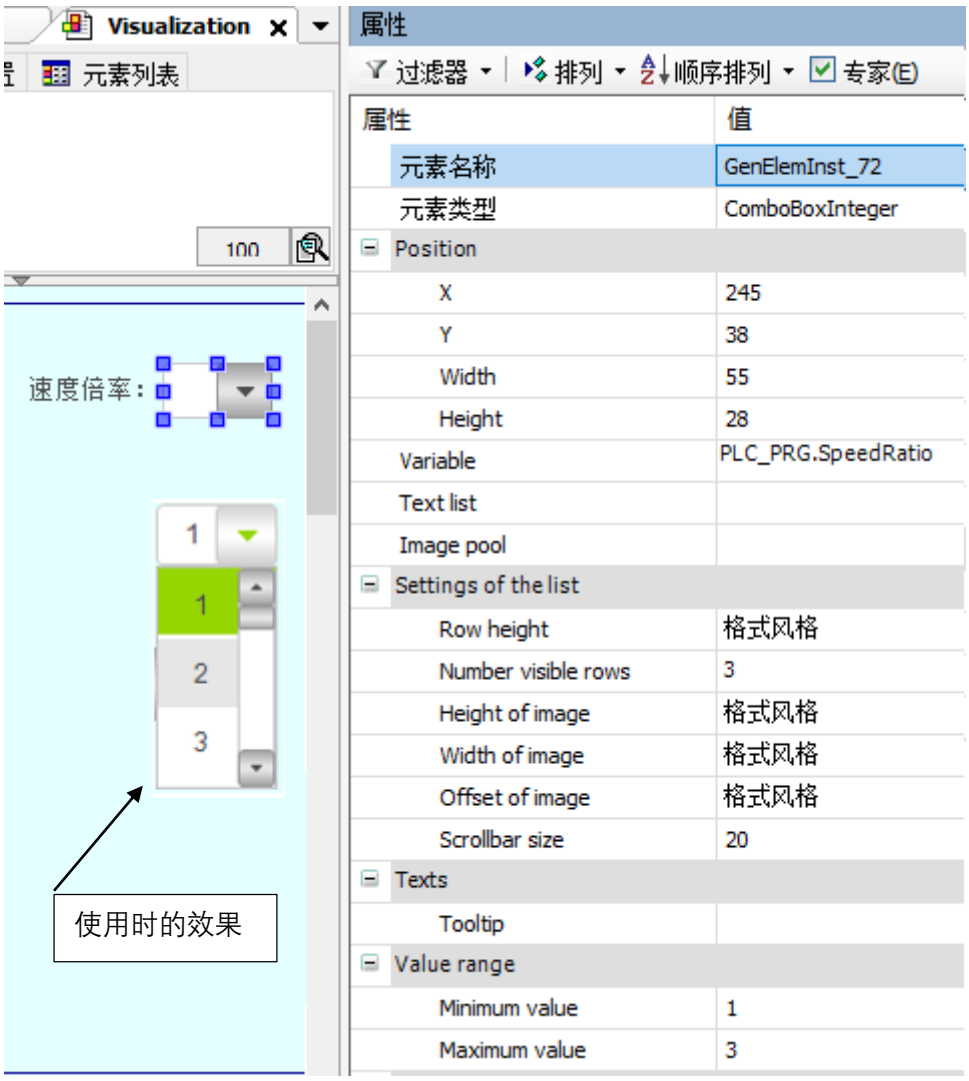

图 5.17　整数型组合框的设置及使用效果

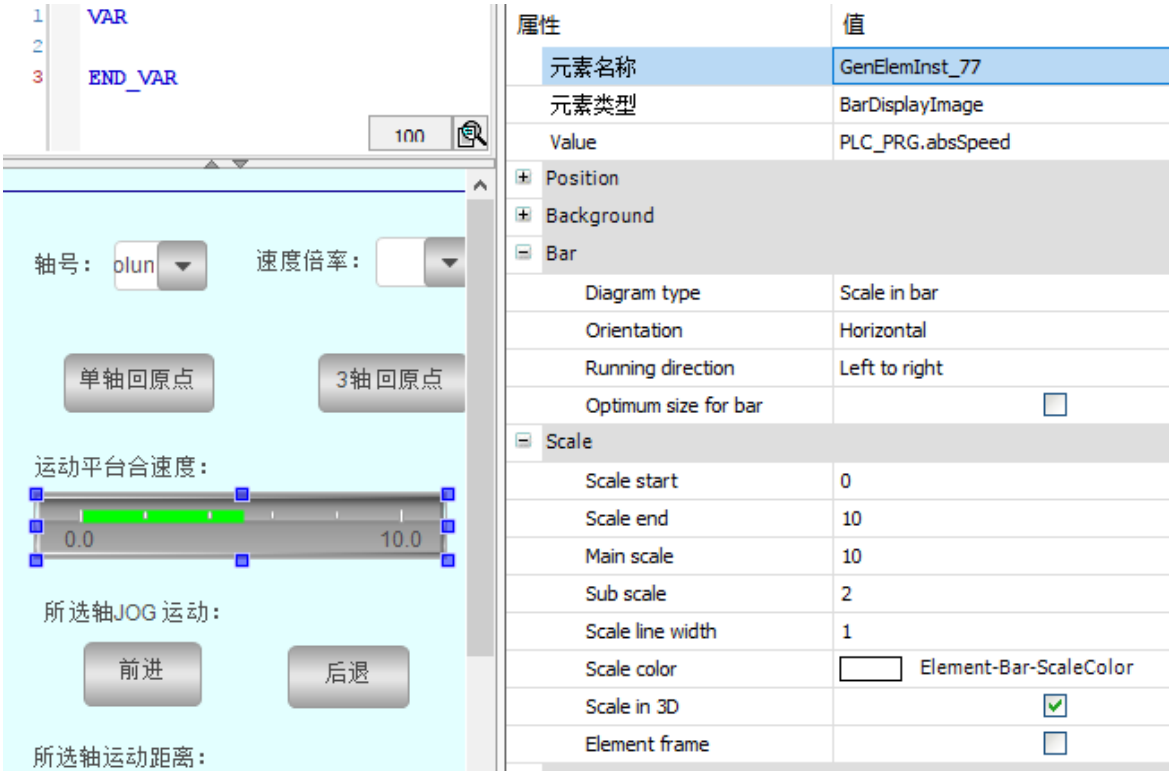

图 5.18　长方形仪表盘的参数设置

例程三的完整代码如下：

```
PROGRAM PLC_PRG
VAR
    PowerX: MC_Power;            // 定义轴使能模块名称
    PowerY: MC_Power;
    PowerZ: MC_Power;

    JogX: MC_Jog;                          // 定义 X 轴的 Jog 运动模块
    JogY: MC_Jog;                          // 定义 Y 轴的 Jog 运动模块
    JogZ: MC_Jog;                          // 定义 Z 轴的 Jog 运动模块
    JogFd: ARRAY[1..3] OF BOOL;           // 电机正转信号
    JogBd: ARRAY[1..3] OF BOOL;           // 电机反转信号

    ButtonFd :BOOL;                        // 前进键
    ButtonBd :BOOL;                        // 后退键

    GetPositionX: MC_ReadActualPosition; // 读轴的位置模块
    GetPositionY: MC_ReadActualPosition;
    GetPositionZ: MC_ReadActualPosition;
    PositionX: LREAL;
    PositionY: LREAL;
    PositionZ: LREAL;

    GetSpeedX: MC_ReadActualVelocity;     // 读轴的速度模块
    GetSpeedY: MC_ReadActualVelocity;
    GetSpeedZ: MC_ReadActualVelocity;
    realSpeedX: LREAL ;
    realSpeedY: LREAL ;
```

```
    realSpeedZ: LREAL ;
    absSpeed: LREAL;                    // 3 轴的合速度

    MoveX: MC_MoveRelative;             // X 轴相对点位运动模块
    MoveY: MC_MoveRelative;             // Y 轴相对点位运动模块
    MoveZ: MC_MoveRelative;             // Z 轴相对点位运动模块
    MoveStart: ARRAY[1..3] OF BOOL;     // 点位运动开始信号
    MoveDone:  ARRAY[1..3] OF BOOL;     // 点位运动结束信号
    Distance: LREAL;                    // 运动距离
    Speed0: LREAL := 2;                 // 运动初速度为 2mm/s
    Speed: LREAL ;                      // 运动速度
    SpeedRatio: INT := 1;   // 速度倍率
    ButtonRun :BOOL;        // 启动键

    AxisNum : INT :=1;      //轴号，默认轴为 X
    AxisName:ARRAY [1..3] OF STRING:= ['X','Y','Z']; // 组合框显示的数组

    HomeX: LT_EtherCATHomeMove;         // 定义 X 轴的 Home 运动模块
    HomeY: LT_EtherCATHomeMove;         // 定义 Y 轴的 Home 运动模块
    HomeZ: LT_EtherCATHomeMove;         // 定义 Z 轴的 Home 运动模块
    HomeStart: ARRAY[1..3] OF BOOL;     // Home 开始信号
    HomeDone:  ARRAY[1..3] OF BOOL;     // Home 结束信号
    HomeSpeed: DWORD;                   // Home 速度
    ButtonHome: BOOL;                   // 单轴 Home 键
    ButtonHome3: BOOL;                  // 3 轴 Home 键

    TON1: TON;                  // 定义一个定时器
    Delay1Off: BOOL;            // 定时器的输出信号，定时结束

END_VAR


TON1(IN:=TRUE,PT:=T#50MS,Q=>Delay1Off,ET=>);    // 开定时器

IF Delay1Off = TRUE THEN    // 50ms 执行一次

  TON1(IN:=FALSE,PT:=T#50MS,Q=>Delay1Off,ET=>); // 关定时器
  TON1(IN:=TRUE,PT:=T#50MS,Q=>Delay1Off,ET=>);  // 开定时器

  //读轴的位置
  GetPositionX(Axis:=X,Enable:=TRUE,Valid=>,Busy=>,Error=>,ErrorID=>,Position=>PositionX);
  GetPositionY(Axis:=Y,Enable:=TRUE,Valid=>,Busy=>,Error=>,ErrorID=>,Position=>PositionY);
  GetPositionZ(Axis:=Z,Enable:=TRUE,Valid=>,Busy=>,Error=>,ErrorID=>,Position=>PositionZ);

  //读轴的速度
  GetSpeedX(Axis:=X,Enable:=TRUE,Valid=>,Busy=>,Error=>,ErrorID=>,Velocity=>realSpeedX);
  GetSpeedY(Axis:=Y,Enable:=TRUE,Valid=>,Busy=>,Error=>,ErrorID=>,Velocity=>realSpeedY);
  GetSpeedZ(Axis:=Z,Enable:=TRUE,Valid=>,Busy=>,Error=>,ErrorID=>,Velocity=>realSpeedZ);

  Speed := Speed0*SpeedRatio;                   // 速度 = 初速度 X 速度倍率
  HomeSpeed := ANY_TO_DWORD(Speed);             // LREAL 转为 DWORD
  absSpeed:=SQRT(realSpeedX*realSpeedX+realSpeedY*realSpeedY+realSpeedZ*realSpeedZ);
                                    // 计算合速度
  IF  ButtonFd=TRUE THEN            // 如果前进键按下
      JogFd[AxisNum]:=TRUE;         // 电机正转
    ELSE
```

```
      JogFd[AxisNum]:=FALSE;              // 否则，电机不可正转
  END_IF;

  IF  ButtonBd=TRUE THEN                  // 如果后退键按下
      JogBd[AxisNum]:=TRUE;               // 电机反转
    ELSE
      JogBd[AxisNum]:=FALSE;              // 否则，电机不可反转
  END_IF;

  IF  ButtonRun=TRUE  THEN               // 点位运动启动键按下
      MoveStart[AxisNum]:=TRUE;          // 开始相对点位运动
  END_IF;

  IF  ButtonHome = TRUE  THEN            //  单轴 Home 键按下
      HomeStart[AxisNum]:=TRUE;          // 启动回原点运动
  END_IF

  IF  ButtonHome3 = TRUE  THEN           //  3 轴 Home 键按下
      HomeStart[1]:=TRUE;                // 启动 3 轴回原点运动
      HomeStart[2]:=TRUE;
      HomeStart[3]:=TRUE;
  END_IF

END_IF

IF MoveDone[AxisNum] = TRUE THEN     // 点位运动停止
   MoveStart[AxisNum]:=FALSE;        // 禁止点位运动
END_IF;

IF  HomeDone[1]=TRUE  THEN             // 如果 X 轴回原点运动结束
    HomeStart[1]:=FALSE;               // 禁止回原点运动
END_IF
IF  HomeDone[2]=TRUE  THEN             // 如果 Y 轴回原点运动结束
    HomeStart[2]:=FALSE;               // 禁止回原点运动
END_IF
IF  HomeDone[3]=TRUE  THEN             // 如果 Z 轴回原点运动结束
    HomeStart[3]:=FALSE;               // 禁止回原点运动
END_IF

// 调用电机使能模块
PowerX(Axis:=X,Enable:=TRUE,bRegulatorOn:=TRUE,bDriveStart:=TRUE,Status=>,
       bRegulatorRealState=>,bDriveStartRealState=>,Busy=>,Error=>,ErrorID=>);
PowerX(Axis:=Y,Enable:=TRUE,bRegulatorOn:=TRUE,bDriveStart:=TRUE,Status=>,
       bRegulatorRealState=>,bDriveStartRealState=>,Busy=>,Error=>,ErrorID=>);
PowerX(Axis:=Z,Enable:=TRUE,bRegulatorOn:=TRUE,bDriveStart:=TRUE,Status=>,
       bRegulatorRealState=>, bDriveStartRealState=>, Busy=>, Error=>, ErrorID=>);

// 调用 Jog 运动模块
JogX(Axis:=X,JogForward:=JogFd[1],JogBackward:=JogBd[1],Velocity:=Speed,Acceleration:=30,
     Deceleration:=30,Jerk:=,Busy=>,CommandAborted=>,Error=>,ErrorId=>);
JogY(Axis:=Y,JogForward:=JogFd[2],JogBackward:=JogBd[2],Velocity:=Speed,Acceleration:=30,
     Deceleration:=30,Jerk:=,Busy=>,CommandAborted=>,Error=>,ErrorId=>);
JogZ(Axis:=Z,JogForward:=JogFd[3],JogBackward:=JogBd[3],Velocity:=Speed,Acceleration:=30,
     Deceleration:=30,Jerk:=,Busy=>,CommandAborted=>,Error=>,ErrorId=>);

// 调用点位运动模块
```

```
MoveX(Axis:=X,Execute:=MoveStart[1],Distance:=Distance,Velocity:=Speed,Acceleration:=30,
      Deceleration:=30,Jerk:=,Done=>MoveDone[1],Busy=>,CommandAborted=>,Error=>,ErrorID=>);
MoveY(Axis:=Y,Execute:=MoveStart[2],Distance:=Distance,Velocity:=Speed,Acceleration:=30,
      Deceleration:=30,Jerk:=,Done=>MoveDone[2],Busy=>,CommandAborted=>,Error=>,ErrorID=>);
MoveZ(Axis:=Z,Execute:=MoveStart[3],Distance:=Distance,Velocity:=Speed,Acceleration:=30,
      Deceleration:=30,Jerk:=,Done=>MoveDone[3],Busy=>,CommandAborted=>,Error=>,ErrorID=>);

// 调用 Home 运动模块。注意:该模块中速度、加速度的单位为脉冲/秒、脉冲/秒平方
HomeX(Axis:=X,bExecute:=HomeStart[1],Home_Method:=21,Home_FastVel:=3000*HomeSpeed,
      Home_SlowVel:=900,Home_Acc:=3000000,Home_Pos:=0,HomeAxisNo:=0,
      bBusy=>,bDone=>HomeDone[1],bCommandAborted=>,Error=>,ErrorID=>);
HomeY(Axis:=Y,bExecute:=HomeStart[2],Home_Method:=21,Home_FastVel:=3000*HomeSpeed,
      Home_SlowVel:=900,Home_Acc:=3000000,Home_Pos:=0,HomeAxisNo:=0,
      bBusy=>,bDone=>HomeDone[2],bCommandAborted=>,Error=>,ErrorID=>);
HomeZ(Axis:=Z,bExecute:=HomeStart[3],Home_Method:=21,Home_FastVel:=7500*HomeSpeed,
      Home_SlowVel:=900,Home_Acc:=3000000,Home_Pos:=0,HomeAxisNo:=0,
      bBusy=>,bDone=>HomeDone[3],bCommandAborted=>,Error=>,ErrorID=>);
```

5.4. 单选钮、复选框和开关的设计 （CASE 语句）

下面在例程三的基础上再添加几个控件。

A. 单选钮

我们也可以用单选钮来选择轴号。首先，为单选钮定义一个变量如下所示：

```
AxisNumS : INT :=0;      //速度表的轴号，默认轴为 X
```

在“Common controls”中选“RadioButton”，用鼠标将该控件拖入界面，再用鼠标调整至合适的位置。

这时，只有 2 个选择钮，在其属性表中找到“Radio button settings”，点击“Radio button”后面的按钮可以再添加一个选择钮，如图 5.19 所示。在 3 个选择钮的“Text”中填入 X 轴、Y 轴、Z 轴。在“Variable”后添加变量 AxisNumS。

注意：和组合框不同，单选钮的编号只能从 0 开始。

B. 半圆形仪表盘

用一个半圆形的仪表盘显示各轴的运动速度。

先在程序中定义一个变量 realSpeed: LREAL ;

在“Measurement controls”中选“Meter180”，用鼠标将该仪表盘拖入界面，再用鼠标调整至合适的大小和位置。

然后，在其属性表中找到“Value”，添加变量 realSpeed；将“Scale”中的“Scale start”改为-10，将“Scale end”改为 10，将“Main scale”改为 20，将“Sub scale”改为 2。如图 5.20 所示。

在程序中添加如下代码：

```
CASE AxisNumS OF
    0: realSpeed := realSpeedX;   // 如果选 X 轴，则显示 X 轴的速度
    1: realSpeed := realSpeedY;   // 如果选 Y 轴，则显示 Y 轴的速度
    2: realSpeed := realSpeedZ;   // 如果选 Z 轴，则显示 Z 轴的速度
END_CASE;
```

当程序运行后，选择 Y 轴，速度倍率选 3，点击“后退”键，半圆形速度表上显示的速度为-6，运动平台合速度为 6 mm/s（默认速度是 2 mm/s）。如图 5.21 所示。

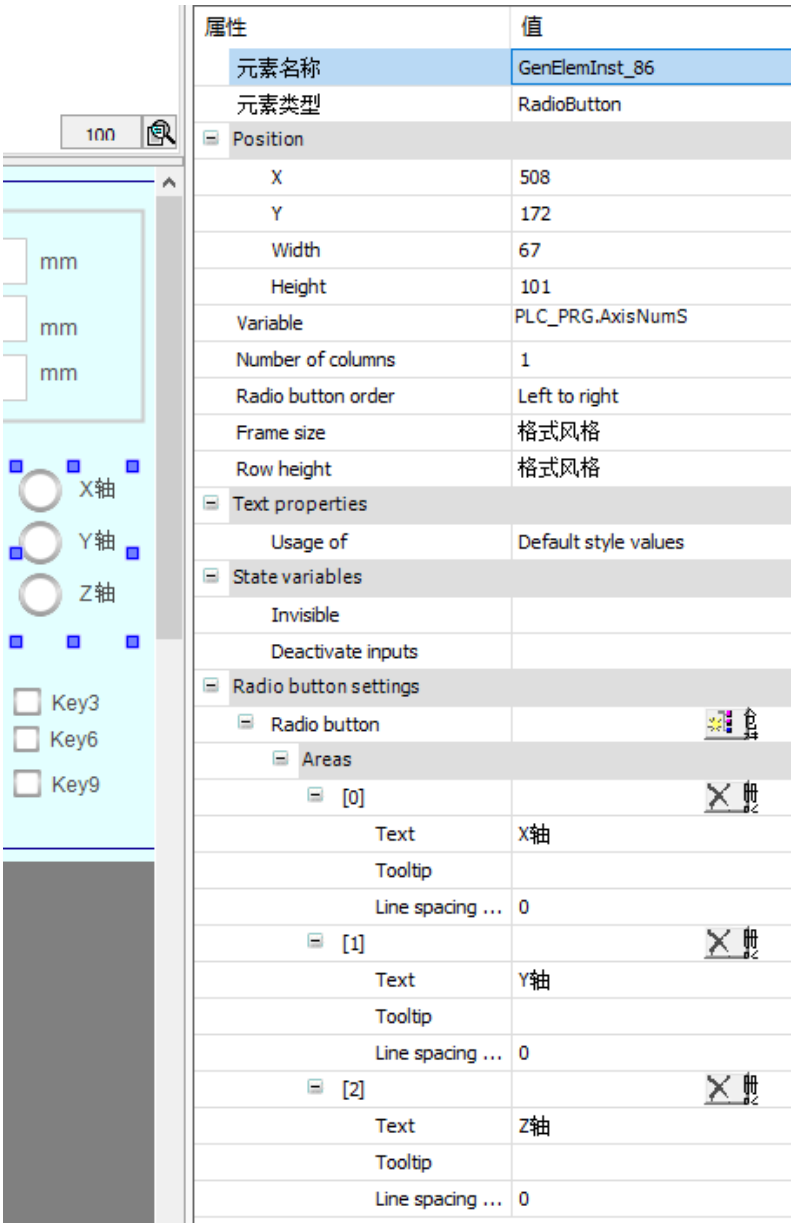

图 5.19　单选钮的参数设置

C. 复选框

用 9 个复选框显示运动平台上 9 个按键的状态，如图 5.21 所示。数字输入口的设置方法请见本文 4.3 节。

在“Common controls”中选“Checkbox”，用鼠标将该控件拖入界面，再用鼠标调整至合适的位置。

在其属性表中找到“Variable”后添加变量 Input1，在“Text”后添加 Key1。如图 5.22 所示。

用相同的方法再添加 8 个复选框，并设置好相应的变量。不需要编写代码，当程序运行后，如果运动平台上有键按下，相应的复选框中就显示符号“√”，如图 5.21 所示。

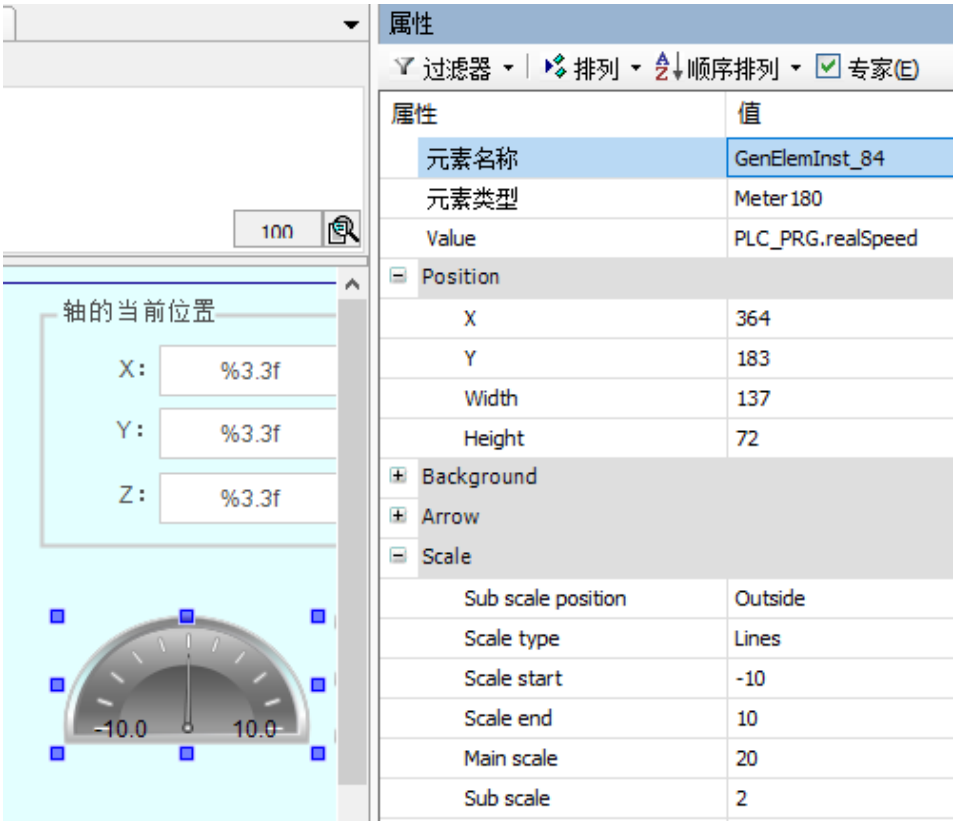

图 5.20　半圆形仪表盘的参数设置

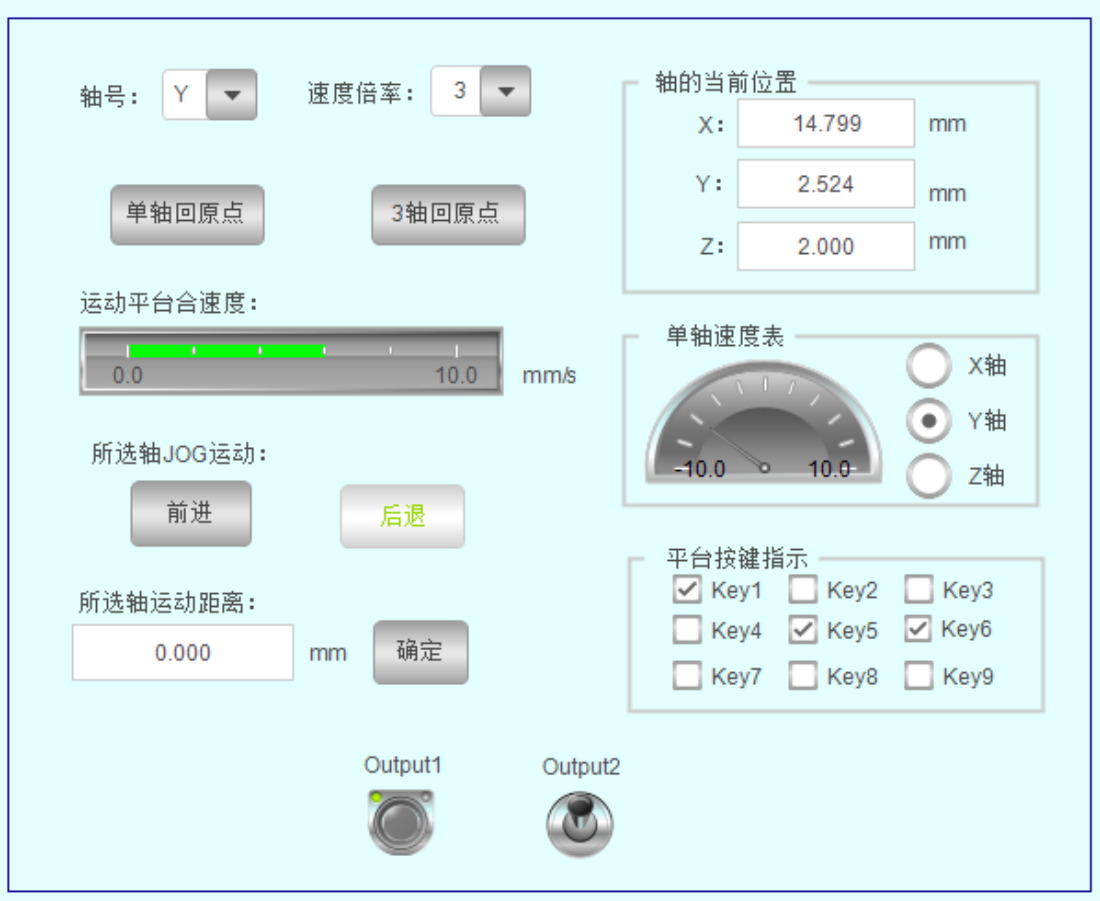

图 5.21　例程三运行界面

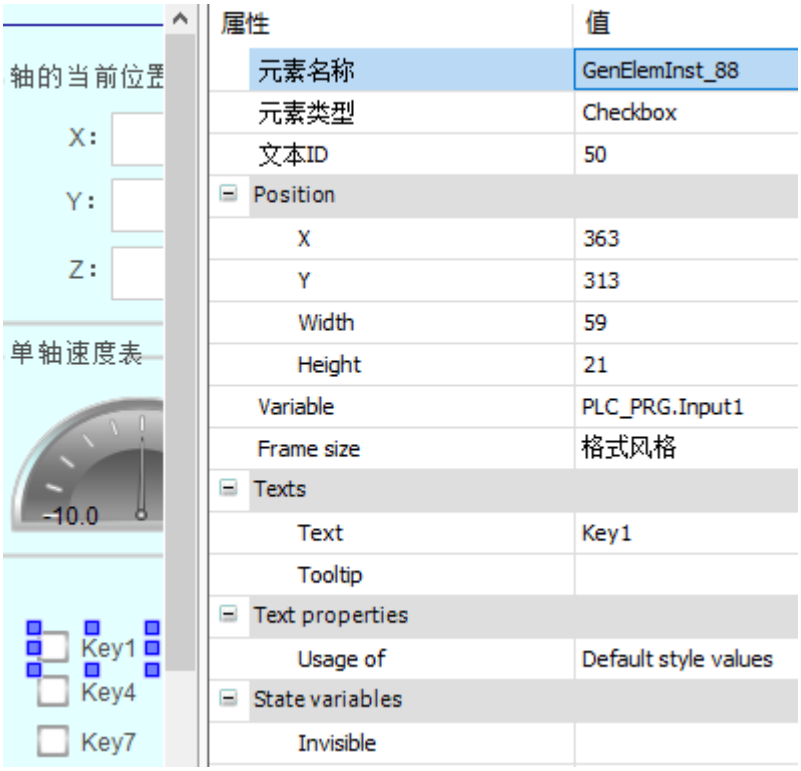

图 5.22　复选框的参数设置

D. 2 种开关控件

“工具箱”里有许多漂亮的控件，这里介绍 2 个开关控件，可用于数字输出口的控制。输出口的设置方法请见本文 4.3 节。

1）带 LED 灯的按键

在“Lamps/Swithches/Bitmaps”中选“PushSwitchLed”，用鼠标将该控件拖入界面，再用鼠标调整至合适的位置。

在其属性表中找到“Variable”后添加变量 Output1。该控件参数设置比普通按键“Button”的简单。在其上方添加标签“Output1”，如图 5.23 所示。

程序运行后，鼠标点击该按钮，其左边的 LED 灯变绿，如图 5.21 所示，变量 Output1 变为 TRUE，运动平台上的指示灯 1 亮；再点击该按钮，其右边的 LED 灯变红，变量 Output1 变为 FALSE，运动平台上的指示灯 1 灭。

2）钮子开关

在“Lamps/Swithches/Bitmaps”中选“DipSwitch”，用鼠标将该控件拖入界面，再用鼠标调整至合适的位置。

在其属性表中找到“Variable”后添加变量 Output2 即可。在其上方添加标签“Output2”，如图 5.23 所示。

程序运行后，鼠标点击该开关，其钮子方向朝上，如图 5.21 所示，变量 Output2 变为 TRUE，运动平台上的指示灯 2 亮；再点击该按开关，其钮子方向朝下，变量 Output2 变为 FALSE，运动平台上的指示灯 2 灭。

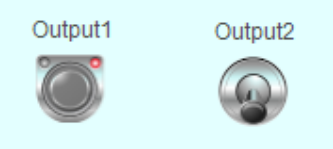

图 5.23　2 种开关控件

5.5. 软数字键盘的调用

在使用 PAC 运动控制器时，如果不需要输入大量信息，可以省略键盘，使用软键盘即可，方法如下。

在设置文本框输入方式时，点击文本框属性表中的“输入配置”下的“OnMouseClick”，弹出如图 5.24 所示的对话框，在左边的选项中选“写变量”，再点击“>”符号；在“输入类型”中选“VisuDialogs.Numpad”。即用软数字键盘输入数字。

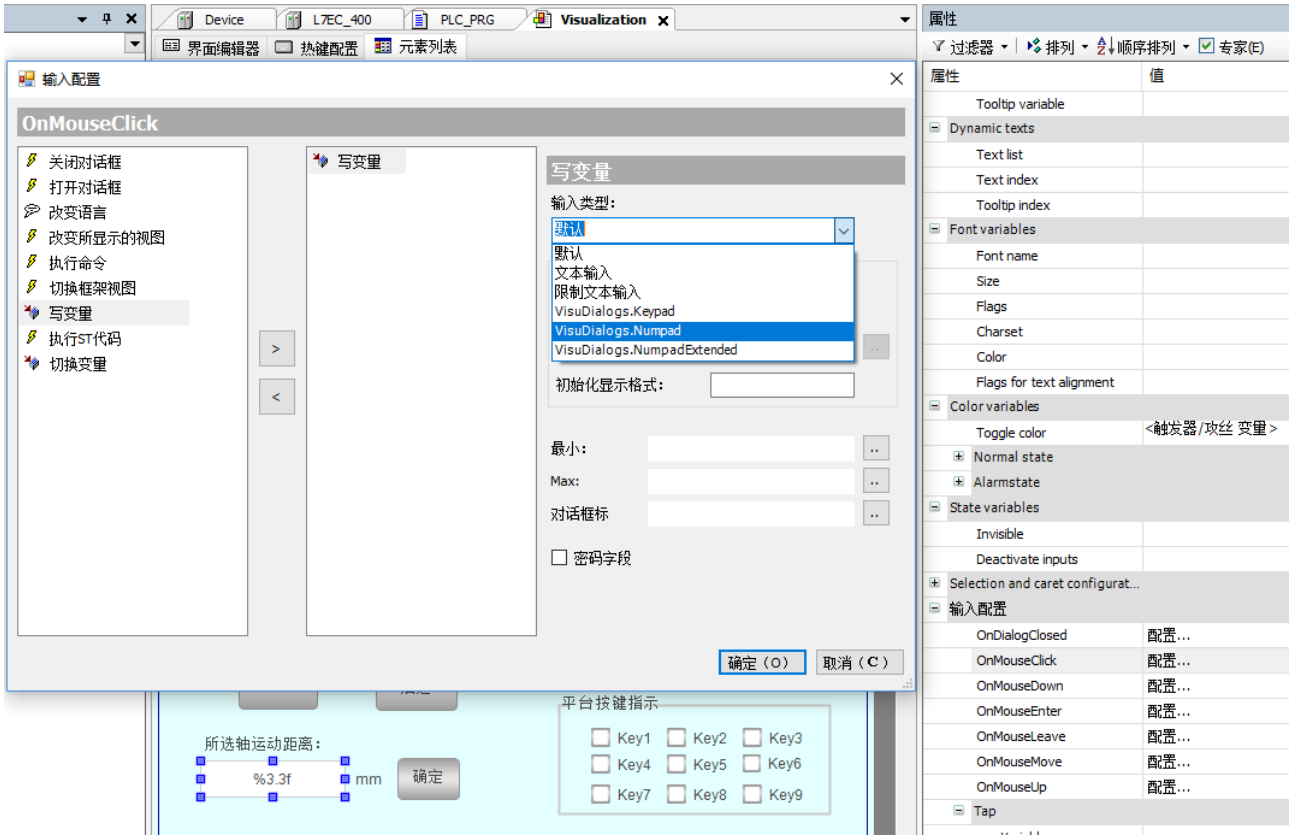

图 5.24　选择软数字键盘为输入方式

程序运行中，用鼠标点击文本框，立即弹出如图 5.26 所示的数字键盘。数字输入的方法和普通键盘一样。点击“OK”后，数字键盘自动关闭。

5.6. 最大化窗体改为指定大小的窗体

可视化界面在 PAC 上显示时，默认为最大化窗体。该模式不利于操作 Windows 其它程序。

将最大化窗体改为指定大小窗体的方法如下。

用记事本打开 PAC 运动控制器 D 盘上的 RTE_File 文件夹中的 CODESYSControl.cfg 文件。找到如下指令：

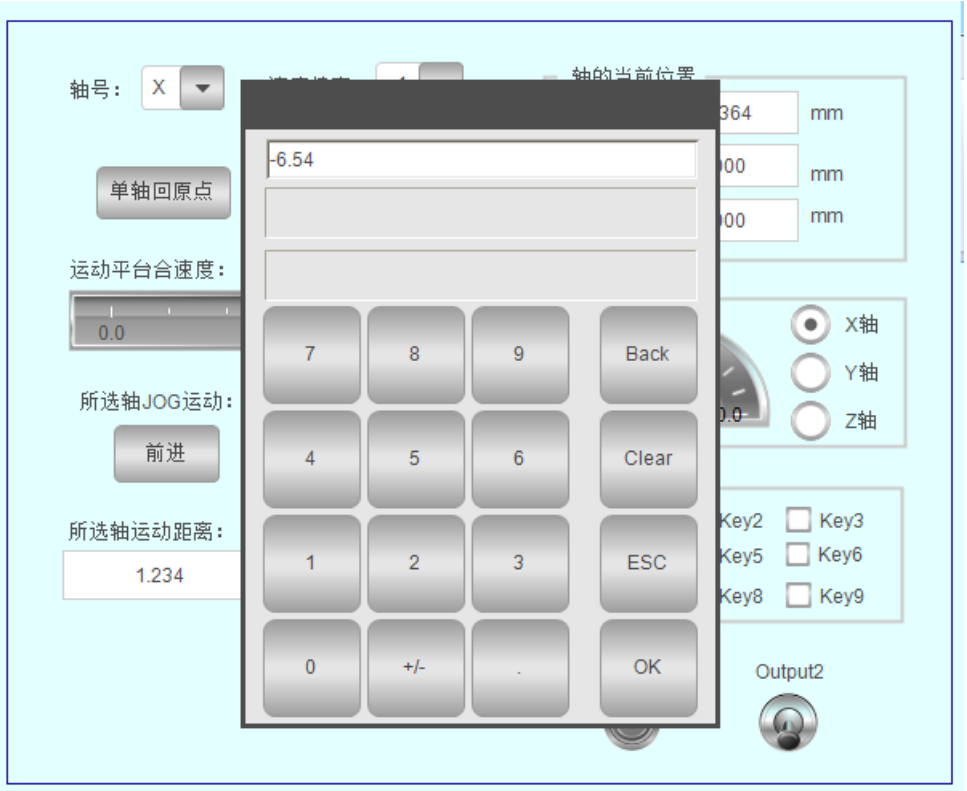

图 5.25　程序运行中，用软数字键盘输入数据

```
[CmpTargetVisuStub]
Program="CoDeSysControlStaticTV.exe"
CommandLine=-r targetvisuextern.cfg
```

然后在其下方添加如下指令：

```
[CmpTargetVisu]
WindowType=2
WindowPositionX=0
WindowPositionY=0
WindowSizeWidth=600
WindowSizeHeight=400
```

其中 WindowSizeWidth 和 WindowSizeHeight 为窗体的宽和高，这 2 个参数要大于等于图 5.6 上设置的窗体宽和高。保存该文件，重启 PAC。之后可视化界面窗体大小即为所设的尺寸。

5.7. 程序自启动的设置

将编译好的程序下载到 PAC 后，可以让 PAC 开机时自启动该程序。设置方法如下。

A.　程序调试完成后，点击菜单中的“在线”，选“创建启动应用”，如图 5.26 所示；弹出对话框如图 5.27 所示，点击“保存”。

B.　在本项目文件夹中找到 Application.app 和 Application.crc2 个文件，如图 5.28 所示，将其拷贝至 U 盘。

C.　将 U 盘上的 2 个 Application 文件替换 PAC 控制器 D:\RTE_File\PlcLogic\Application 中的文件。

D.　重新启动 PAC，该程序即自启动。之后再修改该程序，编译下载后，不需要重复 A~C 步

骤，程序也可自启动。



图 5.26　创建启动应用

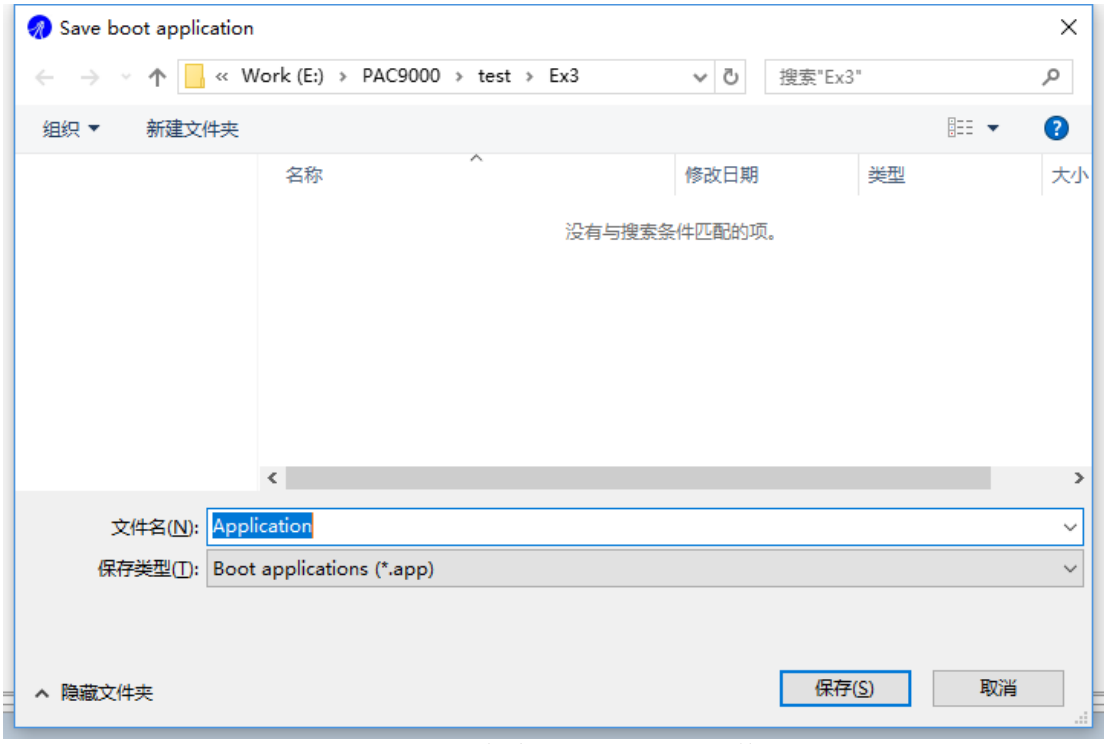

图 5.27　保存 Application 文件

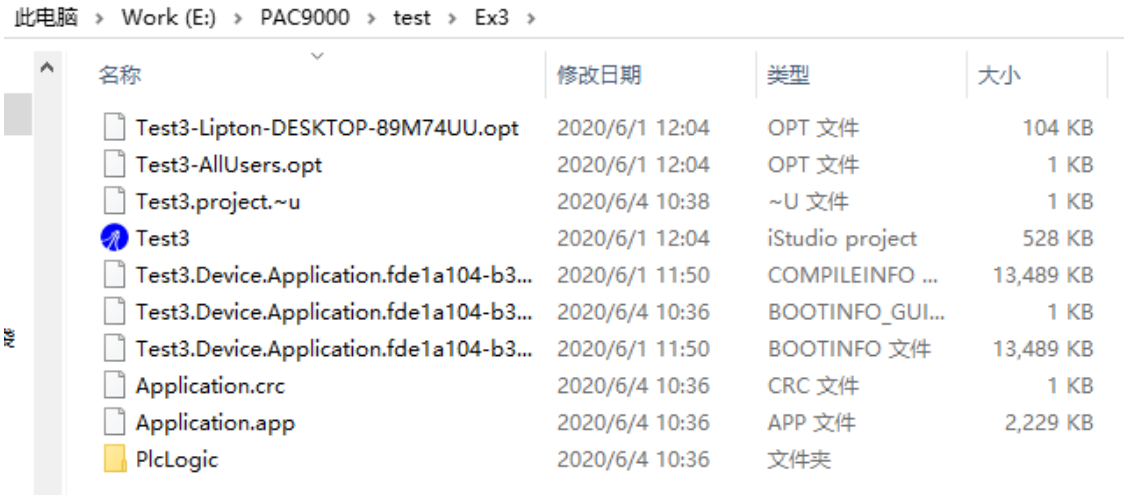

图 5.28　拷贝 2 个 Application 文件

第 6 章　轨迹控制程序设计

下面通过例程四用三轴运动平台模拟点胶机的运动，介绍 PAC9300 运动控制器的轨迹控制功能。

例程四：如图 6.1 所示，在工装有 10 工件需要点胶。已知 10 工件的位置在工件坐标系中的坐标为：

W0（9.23，6.52）；　W1（11.99，11.47）；　W2（7.57，14.46）；
W3（11.51，18.43）；　W4（17.92，20.16）；　W5（19.05，14.69）；
W6（24.14，15.44）；　W7（23.53，7.72）；　W8（19.48，3.27）；
W9（16.59，7.53）。

开始时，X、Y 轴停在工件坐标系原点上，Z 轴距离 XY 平面 5mm 处，此点即为加工起点。当“开始点胶”按键按下后，XY 轴运动到 W0 点，Z 轴向下移动 5mm，打开点胶阀，延时 0.5 秒，Z 轴再上移 5 毫米。然后，以此方式对后续工件点胶，完成 10 个工件点胶后，Z 轴回到加工起点。

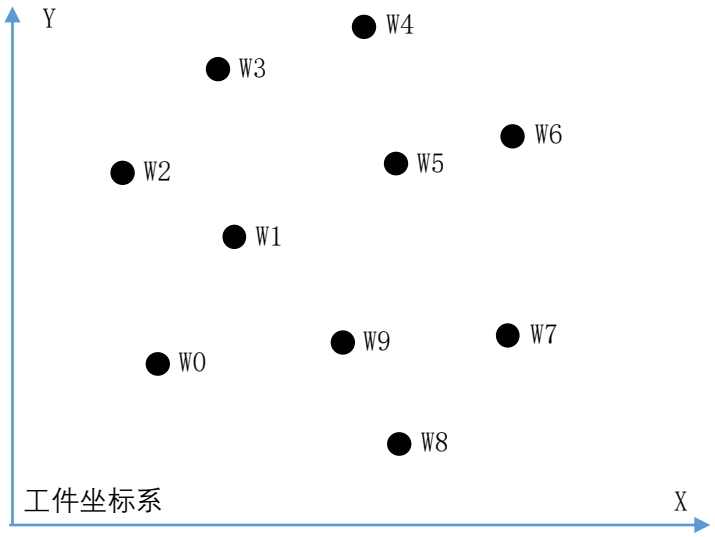

图 6.1　工件位置

6.1. 用直线插补实现轨迹控制（子程序、直线插补模块）

在例程三的基础上设计例程四。在界面上添加一个按键“开始点胶”、一个文本框，如图 6.2 所示。主程序的内容是手动控制运动平台，和例程三基本一样。子程序是自动点胶过程。在编写子程序之前，要设置一下全局变量，供主程序和子程序共用。

A. 全局变量的定义

鼠标右键点击项目树中的“Application”，选择“全局变量列表”，如图 6.3 所示。给“全局变量列表”一个名称：GlobalVariable。

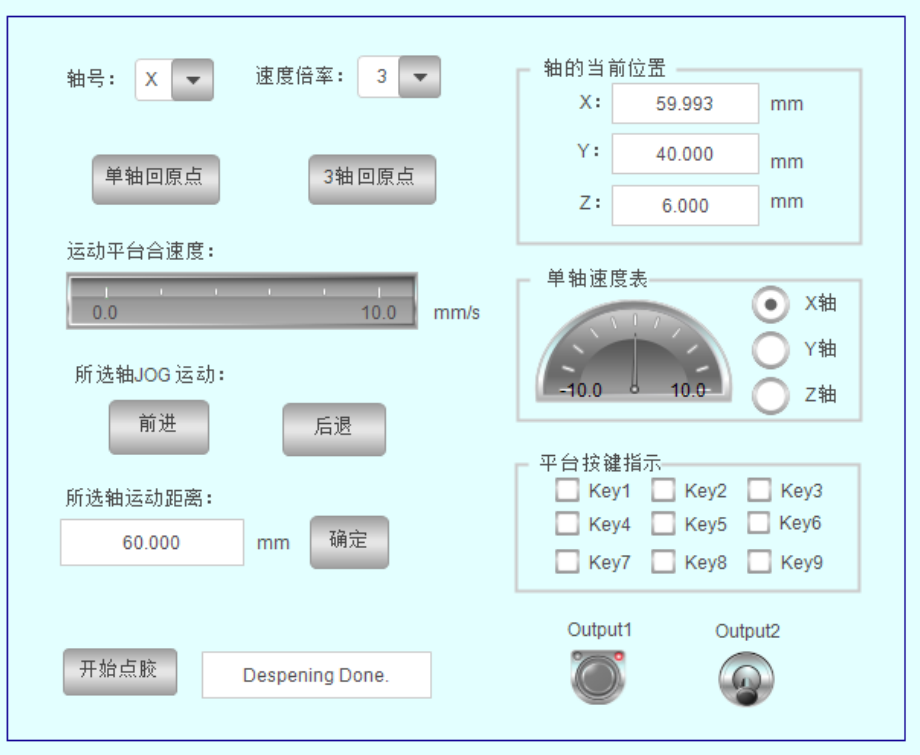

图 6.2　例程四的界面

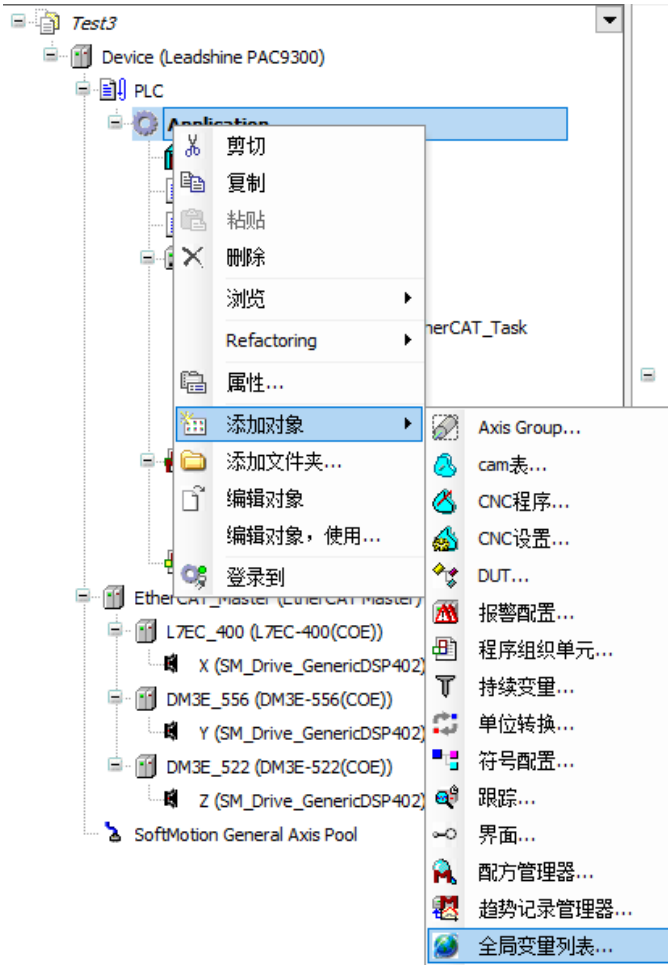

图 6.3　添加全局变量列表

然后，在项目树中鼠标双击“GlobalVariable”打开全局变量列表的编辑界面。

将数字输入、输出变量从主程序 PLC_PRG 中剪贴过来，并修改 IO 映射中的变量。如图 6.4 所示。

注意：全局变量 VAR_GLOBAL 前的{attribute 'qualified_only'}要删除！

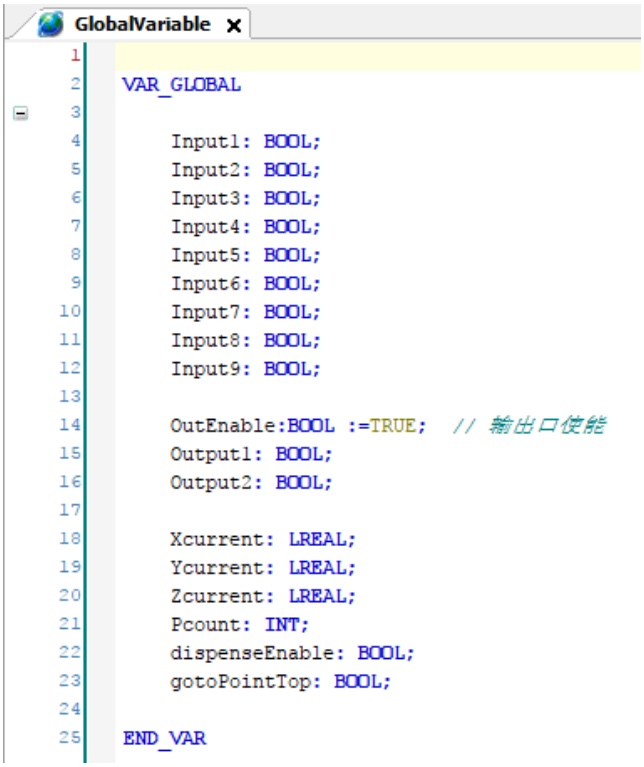

图 6.4　全局变量列表界面

B. 子程序的定义

鼠标右键点击项目树中的“Application”，选择“程序组织单元”，如图 6.5 所示。

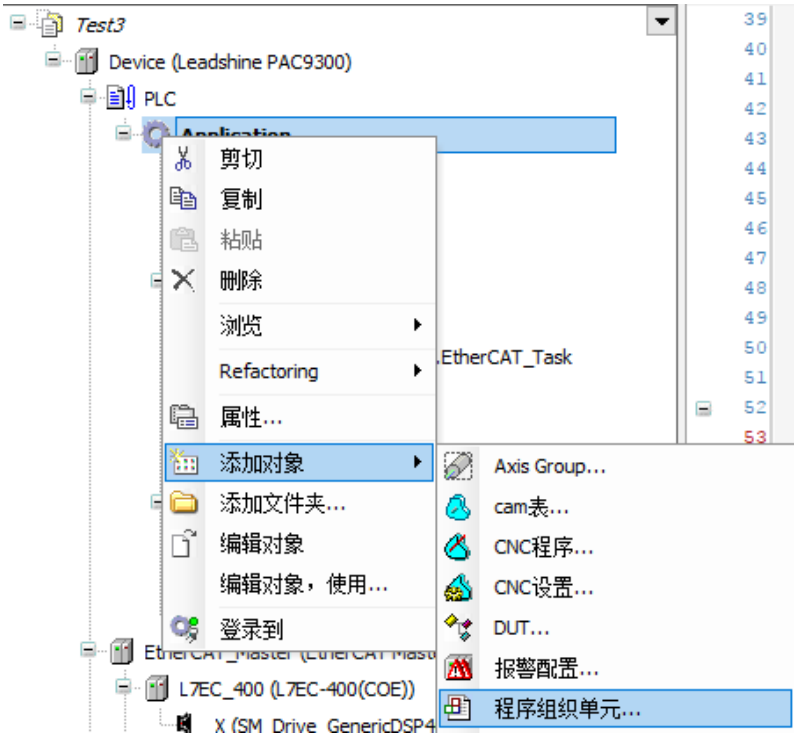

图 6.5　添加子程序

在弹出的对话框中选择“程序”、填写子程序的名称，选子程序的语言为“结构化文本”，然后点“打开”。如图 6.6 所示。在项目树中鼠标双击“Dispensing”，可打开子程序的编辑界面。



图 6.6　设置子程序的名称

C. 触发器的应用

当按下“开始点胶”按键时，要记录 XY 平台的当前位置。为了准确地记录平台的当前位置，需要使用触发器 R_TRIG 检测按键的上升沿。程序代码如下：

```
RTRIGbuttonStart : R_TRIG ;                    // 定义上升沿检测触发器

RTRIGbuttonStart(CLK:= ButtonStart);           // 获取按键按下时的上升沿
IF  RTRIGbuttonStart.Q  = TRUE  THEN           // 如果点胶开始键按下
     dispenseEnable:=TRUE;                     // 可以启动点胶子程序
     gotoPointTop:=TRUE;
     Xcurrent:=PositionX;                      // 记录起点位置
     Ycurrent:=PositionY;
     Zcurrent:=PositionZ;
     Wcount:=0;
 END_IF;
```

D. 点胶子程序的设计

点胶轨迹控制过程有以下几个状态：

1）启动插补模块使平台运动到下一个点的上方，状态变量为 gotoPointTop；

2）等待 XY 平台运动到下一个点的上方；

3）到达点的上方，启动点位运动模块使 Z 轴向下运动，状态变量为 gotoBottom；

4）等待 Z 轴运动到下方；

5）到达下方，开定时器 TON_2，状态变量为 startPause；

6）等待定时器时间到；

7）定时器时间到，启动点位运动模块使 Z 轴向上运动，状态变量为定时器的到时信号 DelayOff2；

8）到达点的上方，准备去下一个点，状态变量为 arriveTop；

9）如果点胶次数小于 10 次，转到 1）继续点胶。

10）点胶次数大于 10 次，回到起点，结束点胶过程。状态变量为 backStartPoint。

根据以上状态变量和运动模块的到位信号，就很容易控制点胶过程一步一步地自动运行。

点胶子程序的完整代码如下：

```
PROGRAM Dispensing
VAR

    TON_2: TON;                         // 定义一个定时器
    DelayOn2: BOOL;                     // 定时器的输入信号，控制其开关
    DelayOff2: BOOL;                    // 定时器的输出信号，显示定时器的状态
    DelayTime2: TIME := T#500MS;        // 定时器的延时时间
    elapsedT2: TIME;                    // 定时器启动后走过的时间

    Px:ARRAY [0..9] OF LREAL :=[9.23,11.99,7.57,11.51,17.92,19.05,24.14,23.53,19.48,16.59];
    Py:ARRAY [0..9] OF LREAL :=[6.52,11.47,14.46,18.43,20.16,14.69,15.44,7.72,3.27,7.53];
    // 10 个工件的坐标数据
    motorState: STRING;
    LineAbs: LT_2AxisLineAbs;
    lineStart: BOOL;
    lineHalt: BOOL;
    lineStop: BOOL;
    LineEnd: ARRAY [0..1] OF LREAL;
    lineDone: BOOL;
    VectorSpeed: LREAL :=12;

    MoveZ: MC_MoveAbsolute;
    StartZ: BOOL;
    EndPositionZ: LREAL;
    DoneZ: BOOL;

    gotoBottom: BOOL;
    startPause: BOOL;
    arriveTop: BOOL;
    backStartPoint: BOOL;

END_VAR
```

```
IF  gotoPointTop = TRUE THEN                                    // 运动到下一个点的上方
    LineEnd[0]:=Px[Wcount]+Xcurrent;                            // 设置直线插补终点坐标
    LineEnd[1]:=Py[Wcount]+Ycurrent;
    motorState:='Goto Point ';
    motorState:=CONCAT (motorState,DINT_TO_STRING(Wcount));  // 显示运动状态
    motorState:=CONCAT (motorState,' . ');
    lineStart:=TRUE;                                            // 开始直线插补
    gotoBottom :=TRUE;
    gotoPointTop :=FALSE;
END_IF

IF lineDone = TRUE AND gotoBottom = TRUE THEN  //  到达下一个点上方，开始向下运动
    LineStart:=FALSE;
    EndPositionZ:=Zcurrent-5.0;                                 // 设置点位运动的终点
    motorState:='Goto Point ';
    motorState:=CONCAT (motorState,DINT_TO_STRING(Wcount));  // 显示运动状态
    motorState:=CONCAT (motorState,' Bottom. ');
    StartZ:=TRUE;                                               //开始 Z 轴点位运动
    startPause :=TRUE;
    gotoBottom := FALSE;
END_IF

IF DoneZ = TRUE AND startPause = TRUE THEN        // 到达点的下方，开始延时
    StartZ:=FALSE;
    DelayOn2:=TRUE;                               // 开定时器 2，延时 0.5 秒
    Output1:=TRUE;                                // 打开点胶阀
    motorState:='Delay 0.5 S. ';
    startPause := FALSE;
END_IF

IF DelayOff2 = TRUE THEN                          // 定时器时间到,开始向上方运动
    DelayOn2:=FALSE;                              // 关定时器
    EndPositionZ:=Zcurrent;                       // 设置点位运动的终点
    motorState:='Goto Point ';
    motorState:=CONCAT (motorState,DINT_TO_STRING(Wcount));   // 显示运动状态
    Output1:=FALSE;                                           // 关闭点胶阀
    motorState:=CONCAT (motorState,' Top. ');
    StartZ:=TRUE;                                             //开始点位运动
    arriveTop :=TRUE;
END_IF

IF DoneZ = TRUE AND arriveTop = TRUE THEN    // 到达点的上方，准备去下一个点
    StartZ:=FALSE;
    Wcount := Wcount+1;
    IF Wcount <  10 THEN
        gotoPointTop := TRUE;
      ELSE
        LineEnd[0]:=Xcurrent;
        LineEnd[1]:=Ycurrent;
        motorState:='Back to start Point.';
        lineStart:=TRUE;                                        //开始直线插补
```

```
            backStartPoint:=TRUE;
        END_IF
        arriveTop := FALSE;
    END_IF

    IF lineDone = TRUE AND backStartPoint = TRUE THEN    // 回到起点，结束点胶过程
        LineStart:=FALSE;
        backStartPoint := FALSE;
        motorState:='Despening Done.';
        dispenseEnable := FALSE;
    END_IF

LineAbs(Axis0:=X,Axis1:=Y,Exec:=lineStart,Halt:=lineHalt,Stop:=lineStop,IpoCycle:=1000,
        EndPos:=LineEnd,VelocityMode:=1,Velocity:=VectorSpeed,Acceleration:=1000,
        Deceleration:=1000,Jerk:=,Vel=>,Done=>lineDone,Busy=>,CommandAborted=>,
        CommandHalt=>,Error=>,ErrorID=>);                    // 调用 2 轴直线插补运动模块
MoveZ(Axis:=Z,Execute:=StartZ ,Position:=EndPositionZ,Velocity:=VectorSpeed,
      Acceleration:=1000,Deceleration:=1000,Jerk:= ,Direction:= ,
      Done=>DoneZ,Busy=>,CommandAborted=>,Error=>,ErrorID=>);// 调用绝对坐标的点位运动模块

TON_2(IN:=DelayOn2, PT:=DelayTime2, Q=>DelayOff2, ET=>elapsedT2); // 调用定时器模块
```

主程序中调用子程序的代码如下：

```
      IF  dispenseEnable = TRUE  THEN
          Dispensing();                    // 启动点胶子程序
      END_IF;
```

E. 字符串的处理

在点胶子程序中用一个文本框显示点胶过程的状态。其中使用了 2 个函数。其定义如下：

字符串相加函数 CONCAT ：字符串 1 := CONCAT (字符串 2, 字符串 3);

即：字符串 1 = 字符串 2 + 字符串 3

整型数转字符串函数 DINT_TO_STRING ：字符串 ：= DINT_TO_STRING (整型数变量);

注意：

1. 文本框要显示字符串，必须在文本框 Text 属性中填%s，如图 6.7 所示。
2. 字符串不能显示汉字。



图 6.7　显示字符串的文本框

F. 变量的跟踪曲线

在程序调试过程中，雷赛 iStudio 软件可显示变量变化的跟踪曲线。使用方法如下。

1. 鼠标右键点击项目树中的“Application”，选择“跟踪”，如图 6.8 所示。
2. 在弹出的对话框中填写一个跟踪曲线的名称，如图 6.9 所示的 Trace。

3. 在项目树中用鼠标双击跟踪曲线名，打开跟踪曲线的界面。
4. 在跟踪曲线界面的右上角，点击“配置”，弹出对话框如图 6.10 所示。将其中的“任务”选“Main Task”；“分辨率”选“μs”。
5. 再点击界面右上角的“添加变量”，弹出如图 6.11 所示的对话框。在“跟踪变量”中选“IOConfig_Globals”下的“X”，找到 X 轴的位置变量 fActPosion，点“确定”。

   这里还有 X 轴的速度变量 fActVelocity，也较常用。

   在“Y”、“Z”中选其位置变量，并添加。
6. 再添加运动平台的合速度 absSpeed。该变量在“Application”下的“PLC_PRG”中。
7. 运行程序后，在变量跟踪界面上点击鼠标右键，选“下载跟踪”，如图 6.12 所示，即可开始记录所添加变量的变化曲线。
8. 要暂停曲线跟踪，在界面上点击右键，选“停止跟踪”。
9. 停止跟踪后，转动鼠标滚轮可放大、缩小曲线，按住鼠标左键可拖拽曲线。

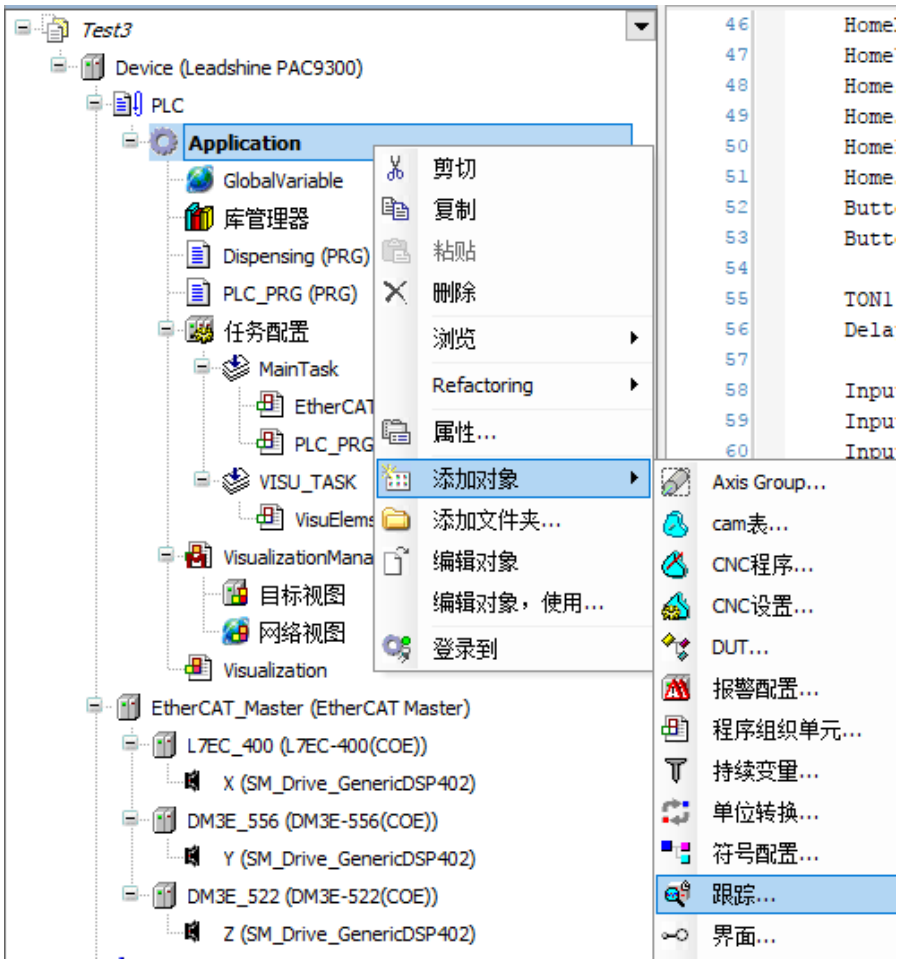

图 6.8　添加跟踪曲线

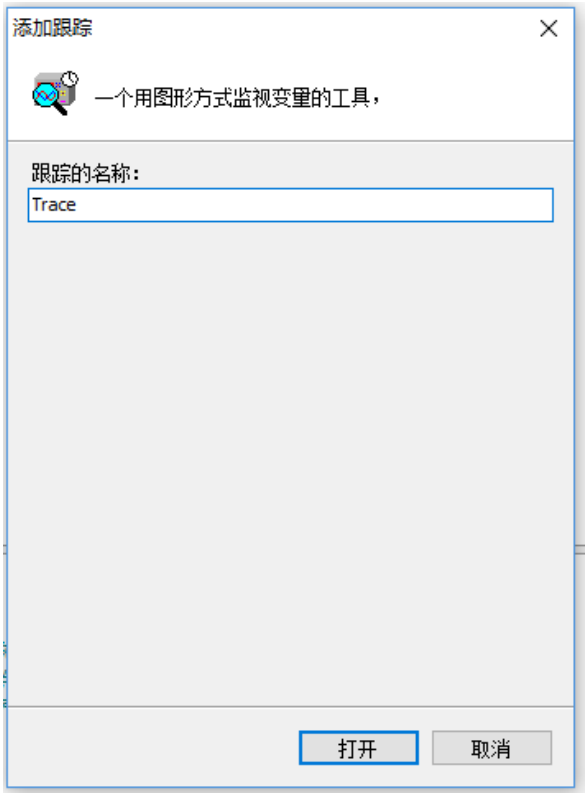

图 6.9　跟踪曲线的名称

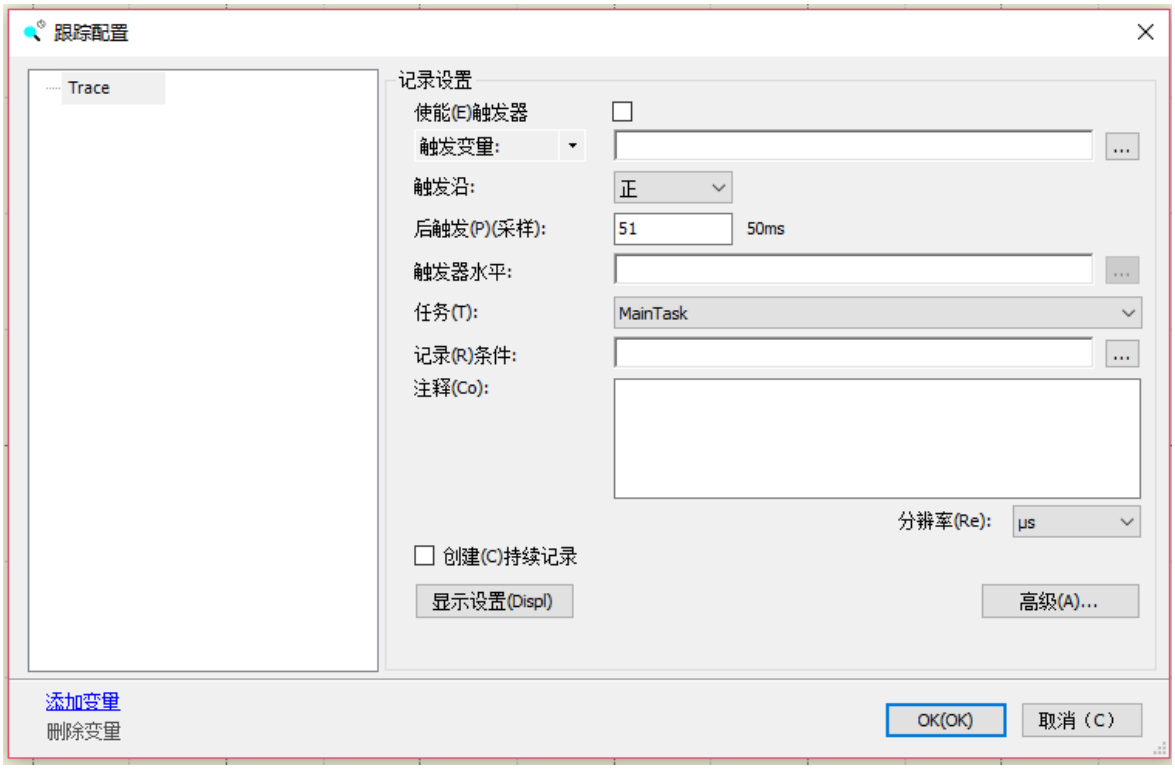

图 6.10　跟踪配置对话框

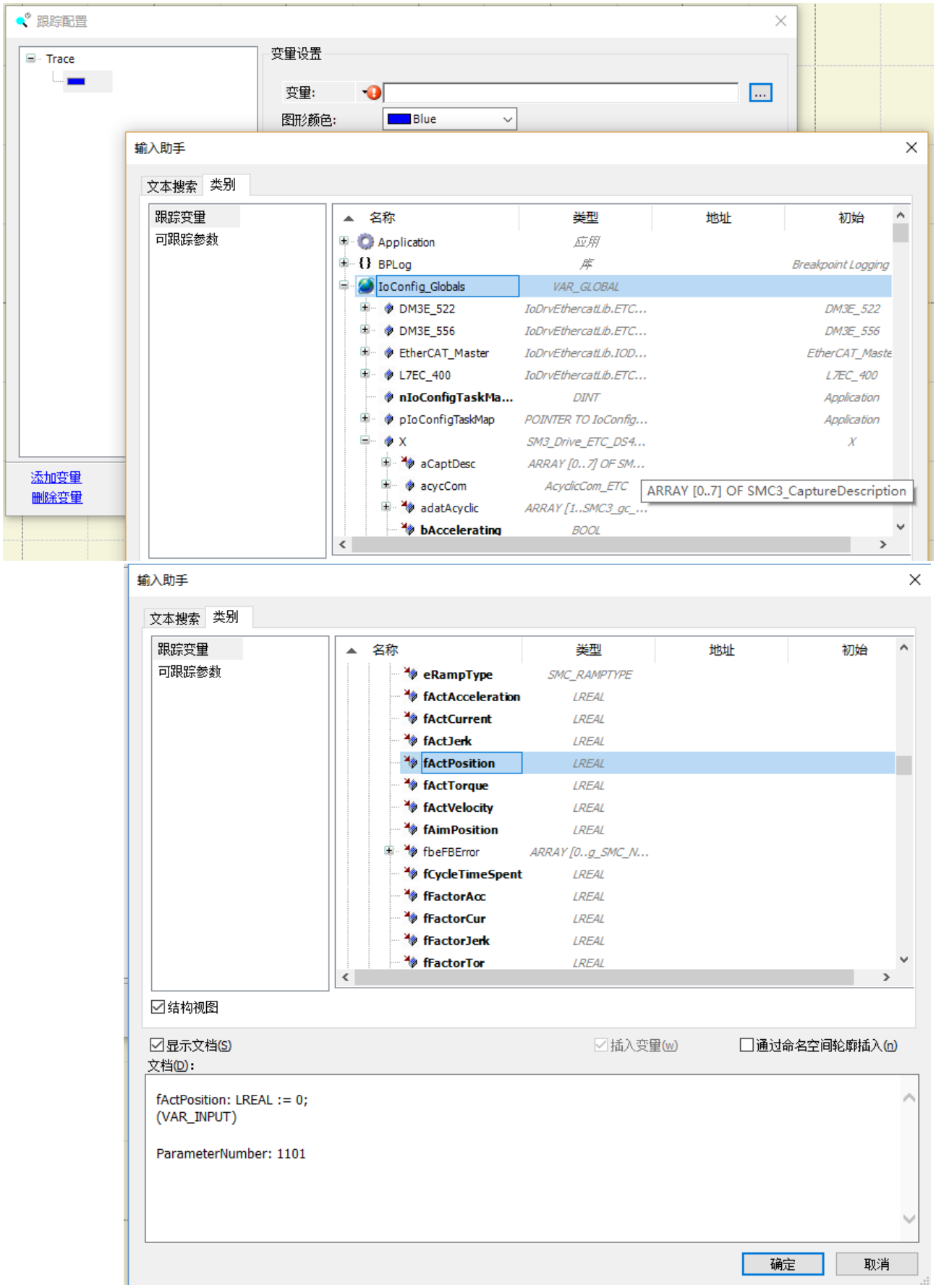
跟踪配置
Trace
变量设置
变量:
图形颜色:
Blue
添加变量
删除变量
输入助手
文本搜索
类别
跟踪变量
可跟踪参数
名称
类型
地址
初始
Application
应用
BPLog
库
Breakpoint Logging
IoConfig_Globals
VAR_GLOBAL
DM3E_522
DM3E_556
EtherCAT_Master
L7EC_400
nIoConfigTaskMa...
DINT
pIoConfigTaskMap
POINTER TO IoConfig...
X
SM3_Drive_ETC_DS4...
aCaptDesc
ARRAY [0..7] OF SM...
acycCom
AcyclicCom_ETC
ARRAY [0..7] OF SMC3_CaptureDescription
adatAcyclic
bAccelerating
BOOL
输入助手
文本搜索
类别
跟踪变量
可跟踪参数
名称
类型
地址
初始
eRampType
SMC_RAMPTYPE
fActAcceleration
fActCurrent
fActJerk
fActPosition
fActTorque
fActVelocity
fAimPosition
fbeFBError
fCycleTimeSpent
fFactorAcc
fFactorCur
fFactorJerk
fFactorTor
LREAL
结构视图
显示文档(S)
插入变量(w)
通过命名空间轮廓插入(n)
文档(D):
fActPosition: LREAL := 0;
(VAR_INPUT)
ParameterNumber: 1101
确定
取消

图 6.11　添加变量

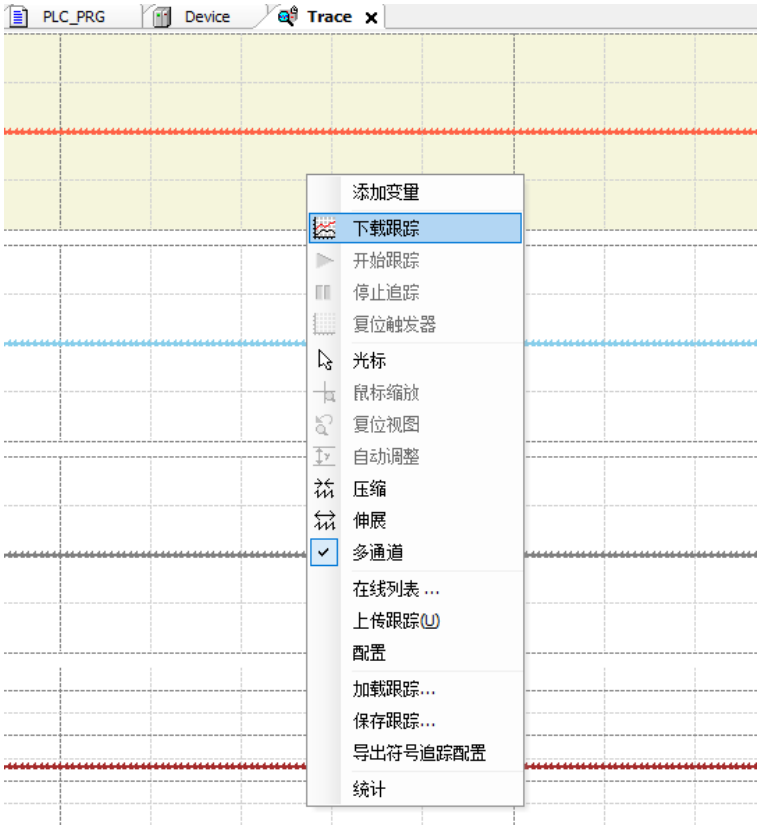

图 6.12　下载跟踪曲线

G. 程序运行结果

如图 6.2 所示，平台的加工起点的当前位置为（60，40，6）。从图 6.13 中可看出起点和终点相同。平台的合速度为 12，和设置的速度一样。

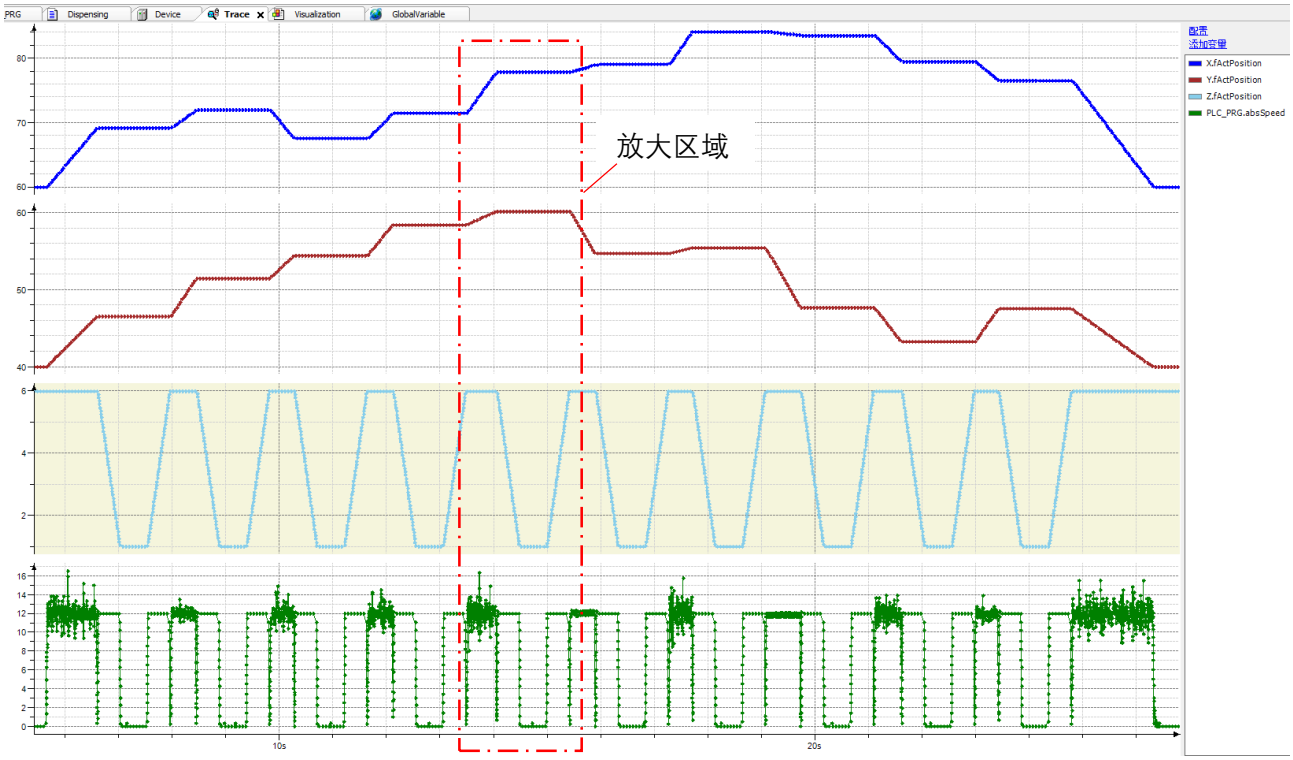

图 6.13　点胶轨迹曲线

如图 6.14 所示，XY 轴插补速度波动较大。Z 轴位置曲线上可以明显看出点胶延时 0.5 秒。

从图中还可看出，从一个点移动到下一个点的过程中，平台的合速度不连续。XY 轴插补完成，速度为零后，Z 再开始运动；Z 轴运动结束，速度为零后，XY 轴插补运动才开始。这样影响了点胶的效率。

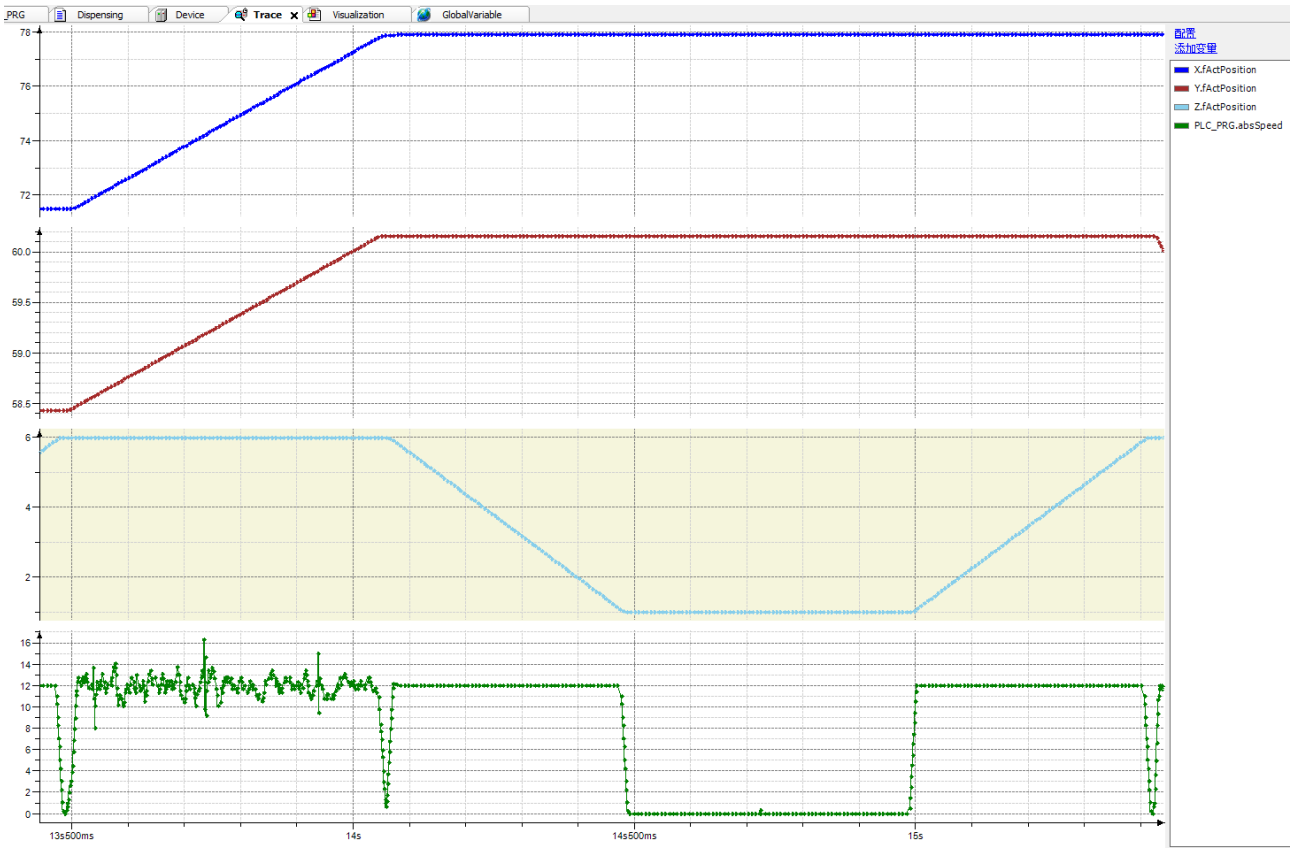

图 6.14　点胶轨迹曲线局部放大图

6.2. 用样条曲线插补实现轨迹控制（样条曲线插补、函数的调用）

例程五：用样条曲线插补实现例程四的点胶过程。

A. 样条曲线轨迹的设计

为了让点胶过程运动平稳、快速，点胶头在工件之间的运动轨迹可以设计为如图 6.15 所示的曲线。采用样条曲线插补函数控制该轨迹运动，可保证矢量速度连续平稳。样条曲线共 8 个点；但从加工原点出发的样条曲线 P0~P2 三个点的 Z 轴坐标和 P3 点的相同；最后回加工原点的样条曲线 P5~P7 三个点的 Z 轴坐标和 P4 点的相同。

在调用样条曲线插补函数前，必须计算样条曲线各点的坐标；还要计算点与点之间距离，得到点与点之间的运动时间。

P0~P7 的坐标计算公式如下：

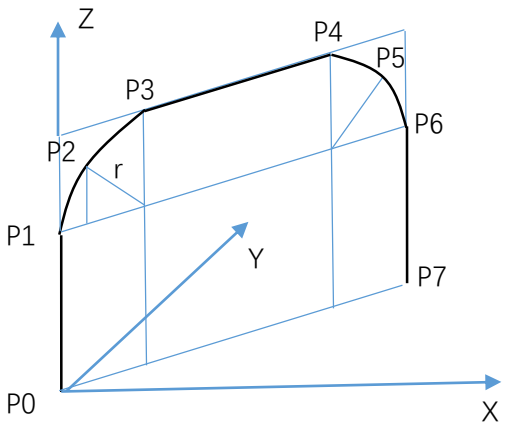

图 6.15　点胶头从一个工件移动到另一个工件的运动轨迹

起点为：P0(0,0,0)；

终点为：P7(x1,y1,0)；

P1、P6 点的高度设为 h1，其坐标为：P1(0,0,h1)、P6(x1,y1,h1)；

P3、P4 点的高度设为 h1+r，其坐标为：P3(p,q, h1+r)、P4(x1-p,y1-q, h1+r)；

其中：q = k p

k = y1/x1

$p^2 + q^2 = r^2$

$p = (r^2/(1+k^2))^{0.5}$，　如果 x1 小于 0，则 p 为负

P2、P5 点为圆弧的中点，其坐标为：P2(m,n,h2)、P5(x1-m,y1-n,h2)；

其中：h2 = h1 + r*sin(PI/4)

$m^2 + n^2 = L^2$

L = r - r*cos(PI/4)

n = k*m

$m = (L^2/(1+k^2))^{0.5}$，　如果 x1 小于 0，则 m 为负

如果 x1=0，则：

m = 0

n = r - r*cos(PI/4)，如果 y1 小于 0，则 n 为负

p = 0

q = r，　　如果 y1 小于 0，则 q 为负

计算样条曲线各段运动时间的公式如下：

设样条曲线插补速度为 v。则：

P0~P1 点的时间为 t1 = h1/v

P1~P2 点的时间为 t2 = PI*r/4/v

P2~P3 点的时间为 t3 = t2

P3~P4 点的时间为 $t4 = (P4(x1-2p)^2+(y1-2q)^2)^{0.5}/v$

P4~P5 点的时间为 t5 = t2

P5~P6 点的时间为 t6 = t2

P6~P7 点的时间为 t6 = t1

B. 设置位置坐标模块

为了便于计算，在开始点胶前，在图 6.1 所示的工件坐标系原点位置，用设置位置坐标模块将三个轴的坐标清零。即平台坐标系等于工件坐标系。在主程序中修改代码如下。

定义部分：

```
SetPosX: MC_SetPosition ; //定义设置位置坐标模块
SetPosY: MC_SetPosition ;
SetPosZ: MC_SetPosition ;
runFlag: BOOL;
```

程序部分：

```
RTRIGbuttonStart(CLK := ButtonStart);      // 获取按键按下的上升沿
IF  RTRIGbuttonStart.Q = TRUE  THEN        // 点胶开始键按下
  dispenseEnable:= TRUE;                   // 可以启动点胶子程序
  splineBusy := TRUE;

  runFlag := TRUE;
    SetPosX(Axis:=X,Execute:=runFlag,Position:=0,Mode:=FALSE,
            Done=>,Busy=>,Error=>,ErrorID=>);
    SetPosY(Axis:=Y,Execute:=runFlag,Position:=0,Mode:=FALSE,
            Done=>,Busy=>,Error=>,ErrorID=>);
    SetPosZ(Axis:=Z,Execute:=runFlag,Position:=0,Mode:=FALSE,
            Done=>,Busy=>,Error=>,ErrorID=>);    // 位置坐标清零
  runFlag := FALSE;
  Wcount:=0;
END_IF;
```

C. 函数的定义

定义函数的过程和定义子程序的过程类似，鼠标右键点击项目树中的“Application”，选择“程序组织单元”，如图 6.5 所示。



图 6.16　添加函数

在弹出的对话框中选择“函数”、填写函数名称，选择函数返回值的类型，选实现函数的语言为“结构化文本”,然后点“打开”。如图 6.16 所示。在项目树中鼠标双击“SplinePoints”，可打开函数的编辑界面。

D. 计算样条曲线各点坐标的函数

计算样条曲线各点坐标的函数代码如下。

```
FUNCTION SplinePoints : ARRAY [0..23] OF LREAL
                            // 函数的返回值为一个数组
VAR_INPUT
    x0,y0,x1,y1: LREAL;  // 函数的输入参数为曲线的起点和终点坐标
END_VAR

VAR
    dx,dy,k: LREAL;
    h: LREAL :=3.5;
    r0: LREAL :=1.5;
    Z: LREAL :=5.0;
    h2,L,m,n,p,q: LREAL;
END_VAR

dx:=x1-x0;
dy:=y1-y0;
h2:= h + r0*SIN(PI/4);      //PI 是系统定义的 LREAL 型常数，3.141592654
L := r0 - r0*COS(PI/4);

IF dx<>0 THEN
   k:= dy/dx;
   m := SQRT(L*L/(1+k*k));
   n := k*m;

   p := SQRT(r0*r0/(1+k*k));
   q := k*p;

  ELSE
   m := 0;
   n := L;

   p := 0;
   q := r0;

   IF dy<0 THEN
      n := -L;
      q := -r0;
   END_IF

END_IF

IF dx<0 THEN
    m := -m;
```

```
        n := -n;
        p := -p;
        q := -q;
    END_IF

    SplinePoints[0]:=x0;      // P0
    SplinePoints[1]:=y0;
    SplinePoints[2]:=0;

    SplinePoints[3]:=x0;      // P1
    SplinePoints[4]:=y0;
    SplinePoints[5]:=h;

    SplinePoints[6]:=m+x0;    // P2
    SplinePoints[7]:=n+y0;
    SplinePoints[8]:=h2;

    SplinePoints[9]:=p+x0;    // P3
    SplinePoints[10]:=q+y0;
    SplinePoints[11]:=Z;

    SplinePoints[12]:=x1-p;   // P4
    SplinePoints[13]:=y1-q;
    SplinePoints[14]:=Z;

    SplinePoints[15]:=x1-m;   // P5
    SplinePoints[16]:=y1-n;
    SplinePoints[17]:=h2;

    SplinePoints[18]:=x1;     // P6
    SplinePoints[19]:=y1;
    SplinePoints[20]:=h;

    SplinePoints[21]:=x1;     // P7
    SplinePoints[22]:=y1;
    SplinePoints[23]:=0;
```

E. 点胶子程序及样条曲线插补模块

样条曲线插补模块就是让用户对一个轴定义一条特殊的位移曲线。当 X、Y、Z 轴同时按其特殊的位移曲线运动，就可以得到一条特定的空间曲线。

用样条曲线插补控制点胶轨迹，过程比用直线插补的简单。代码如下。

```
PROGRAM Dispensing
VAR
    i:INT;
    Px:ARRAY [0..11] OF REAL :=
                    [0,9.23,11.99,7.57,11.51,17.92,19.05,24.14,23.53,19.48,16.59,0];
    Py:ARRAY [0..11] OF LREAL :=
                    [0,6.52,11.47,14.46,18.43,20.16,14.69,15.44,7.72,3.27,7.53,0];
                      // 起点、10 个工件、终点的坐标数据
```

```
    Points: ARRAY [0..23] OF LREAL;
    XYZSpeed: LREAL :=12;              // mm/s
    ms: TIME:= T#1MS;
    deltaT: ARRAY [0..7] OF INT;
    h:LREAL :=3.5;                    // mm
    r0:LREAL :=1.5;
    dxy:LREAL ;

    PTx: MC_PositionProfile;           // 定义样条曲线插补模块
    arNUMx: ARRAY [1..100] OF SMC_TP;  // 数据长度必须是 1~100
    arPTx: MC_TP_REF;
    PTxStart: BOOL;
    PTxDone: BOOL;

    PTy: MC_PositionProfile;
    arNUMy: ARRAY [1..100] OF SMC_TP;
    arPTy: MC_TP_REF;
    PTyStart: BOOL;
    PTyDone: BOOL;

    PTz: MC_PositionProfile;
    arNUMz: ARRAY [1..100] OF SMC_TP;
    arPTz: MC_TP_REF;
    PTzStart: BOOL;
    PTzDone: BOOL;

    x0,y0,x1,y1: LREAL;
    PVTbusy: BOOL :=FALSE;
    PVTdone: BOOL;
    motorState: STRING;

    TON_2: TON;                       // 定义一个定时器
    DelayOn2: BOOL;                   // 定时器的输入信号，控制其开关
    DelayOff2: BOOL;                  // 定时器的输出信号，显示定时器的状态
    DelayTime2: TIME := T#500MS;      // 定时器的延时时间

END_VAR


IF splineBusy=TRUE THEN           // 计算 PT 曲线各点的数据
    IF   Wcount<10 THEN
        motorState:='goto Point ';
        motorState:=CONCAT (motorState,DINT_TO_STRING(Wcount));  // 显示点胶的顺序
      ELSE
        motorState:='go home. ';
    END_IF

        x0 := Px[Wcount];
        y0 := Py[Wcount];
        x1 := Px[Wcount+1];
        y1 := Py[Wcount+1];
```

```
    Points := SplinePoints(x0,y0,x1,y1);   //调用函数，计算样条曲线的坐标值

    deltaT[0] := 0;
    deltaT[1] := LREAL_TO_INT(1000*h/XYZSpeed);
    deltaT[2] := LREAL_TO_INT(250*PI*r0/XYZSpeed);
    deltaT[3] := deltaT[2];
    deltaT[5] := deltaT[2];
    deltaT[6] := deltaT[2];
    deltaT[7] := deltaT[1];
    dxy := SQRT((Points[12]-Points[9])*(Points[12]-Points[9])+
                (Points[13]-Points[10])*(Points[13]-Points[10]));
    deltaT[4] := LREAL_TO_INT(1000*dxy/XYZSpeed);
     //计算 2 点之间的距离和时间（扣除 2 个小圆弧）

FOR i := 1 TO 8 DO
    arNUMx[i].position:=Points[3*(i-1)];    // 保存绝对坐标
    arNUMy[i].position:=Points[3*(i-1)+1];
    arNUMz[i].position:=Points[3*(i-1)+2]-5;

    arNUMx[i].delta_time:=deltaT[i-1]*ms;    // 时间变量的设置方法，变量×时间常数
    arNUMy[i].delta_time:=deltaT[i-1]*ms;
    arNUMz[i].delta_time:=deltaT[i-1]*ms;
END_FOR

IF   Wcount =0 THEN                    // 修正第 1 条曲线的 Z 轴坐标,无前面的上升段
    arNUMz[1].position:=Points[3*(1-1)+2];
    arNUMz[1].delta_time:=50*ms;
    arNUMz[2].position:=Points[3*(2-1)+2]-3.5;
    arNUMz[3].position:=Points[3*(3-1)+2]-4.56;
END_IF

IF   Wcount =10 THEN                // 修正第 10 条曲线的 Z 轴坐标,无后面的下降段
    arNUMz[6].position:=Points[3*(6-1)+2]-4.56;
    arNUMz[7].position:=Points[3*(7-1)+2]-3.5;
    arNUMz[8].position:=Points[3*(8-1)+2];
END_IF

arPTx.IsAbsolute:=TRUE ;          // 绝对位置
arPTx.MC_TP_Array:=arNUMx ;       // PT 点的数据
arPTx.Number_of_pairs:=7 ;        // PT 点的个数

arPTy.IsAbsolute:=TRUE ;          // 绝对位置
arPTy.MC_TP_Array:=arNUMy ;        // PT 点的数据
arPTy.Number_of_pairs:=7 ;        // PT 点的个数

arPTz.IsAbsolute:=TRUE ;          // 绝对位置
arPTz.MC_TP_Array:=arNUMz ;       // PT 点的数据
arPTz.Number_of_pairs:=7 ;        // PT 点的个数

PTxStart:=TRUE;                  // 启动 PT 模块
PTyStart:=TRUE;
```

```
        PTzStart:=TRUE;

        splineBusy :=FALSE;
        PVTbusy :=TRUE;

    END_IF

    IF  PVTbusy = TRUE THEN
        PVTdone := (PTxDone AND PTyDone) AND PTzDone;

        IF PVTdone=TRUE  THEN      // 如果 3 个 PT 模块执行完毕，启动 PT 模块的信号复位
            PTxStart:=FALSE;
            PTyStart:=FALSE;
            PTzStart:=FALSE;
            PVTbusy :=FALSE;
            DelayOn2:=TRUE;                    // 开定时器，延时 0.5 秒
            Output1 :=TRUE;                    // 打开点胶阀
        END_IF
     END_IF

     IF DelayOff2 = TRUE  THEN      // 等待延时时间到
        DelayOn2:=FALSE;            // 关定时器
        Output1 :=FALSE;            // 关闭点胶阀
        Wcount := Wcount+1;         // 点胶序号加 1
        splineBusy :=TRUE;          // 开始计算移动到下一个点的 PT 曲线坐标
        IF Wcount >10 THEN
           dispenseEnable :=FALSE;  // 点胶结束
           splineBusy :=FALSE;
        END_IF
     END_IF

    PTx(Axis:=X,TimePosition:=arPTx,Execute:=PTxStart,ArraySize:=8,PositionScale:=1,
        Offset:=0,Done=>PTxDone,Busy=>,CommandAborted=>,Error=>,ErrorID=>);
    PTy(Axis:=Y,TimePosition:=arPTy,Execute:=PTyStart,ArraySize:=8,PositionScale:=1,
        Offset:=0,Done=>PTyDone,Busy=>,CommandAborted=>,Error=>,ErrorID=>);
    PTz(Axis:=Z,TimePosition:=arPTz,Execute:=PTzStart,ArraySize:=8,PositionScale:=1,
        Offset:=0,Done=>PTzDone,Busy=>,CommandAborted=>,Error=>,ErrorID=>); // 调用 PT 模块

    TON_2(IN:= DelayOn2, PT:= DelayTime2, Q=>DelayOff2, ET=> );          // 调用定时器模块
```

F. 程序运行结果分析

程序设置的平台合速度为 12，和例程四的直线插补速度一样。

从图 6.17 中可看出，点胶过程用时 19.8 秒，而图 6.12 的点胶过程用时 21.7 秒。采用样条曲线插补比直线插补快了 1.9 秒，点胶效率提高 8.8%。

原因在于 XYZ 三轴的合速度是连续变化的。不象图 6.12，2 个工件之间的运动速度有 2 次停顿。从图 6.18 也可明显看出 X、Y、Z 三轴的位移曲线不是象图 6.13 那样的折线，而是平滑的过渡曲线。这不但提高了点胶速度，平台运动时的振动和噪音也减小很多。

唯一不足之处是速度变化时有一点超调。

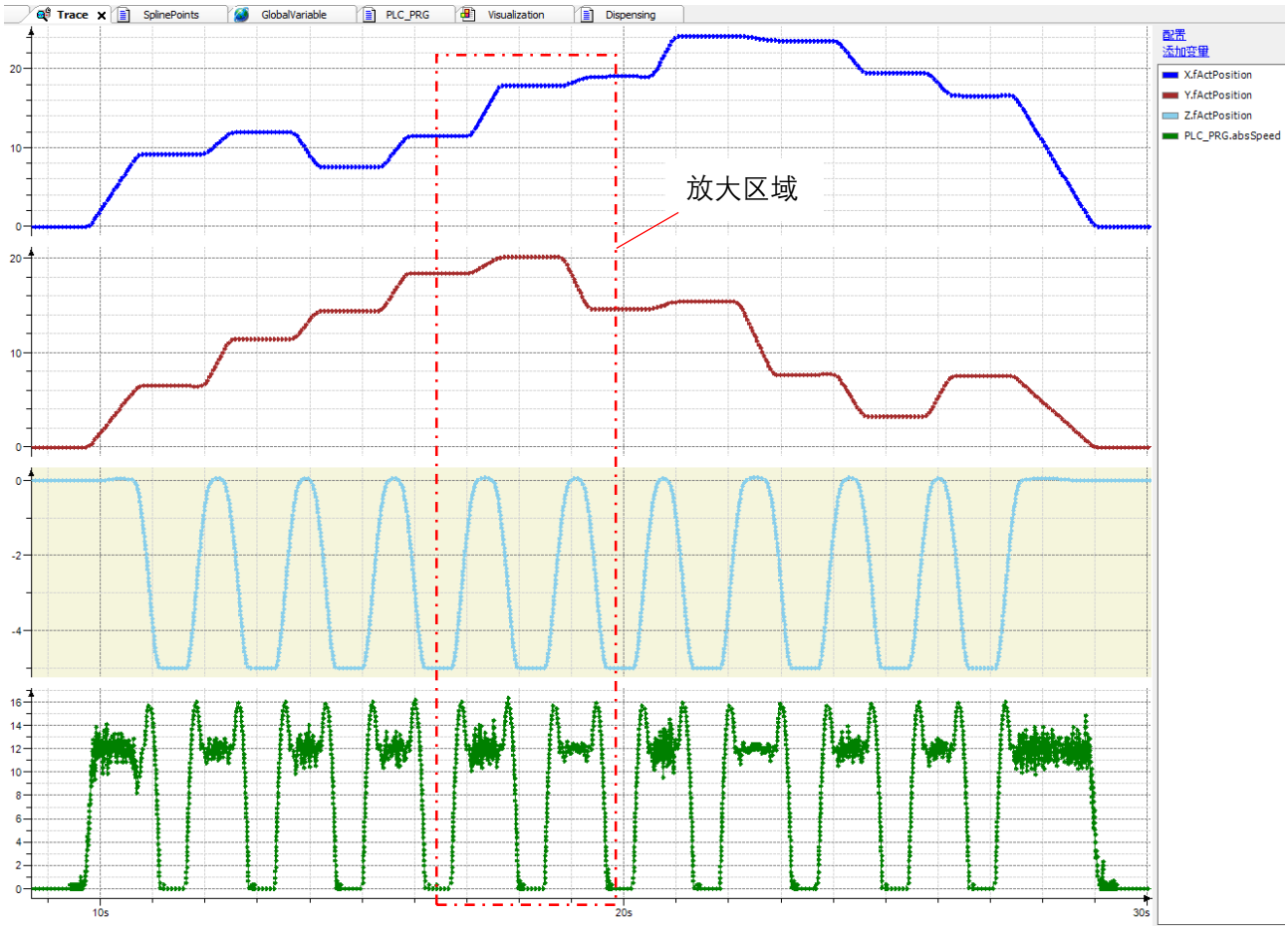

图 6.17　样条曲线插补点胶轨迹曲线

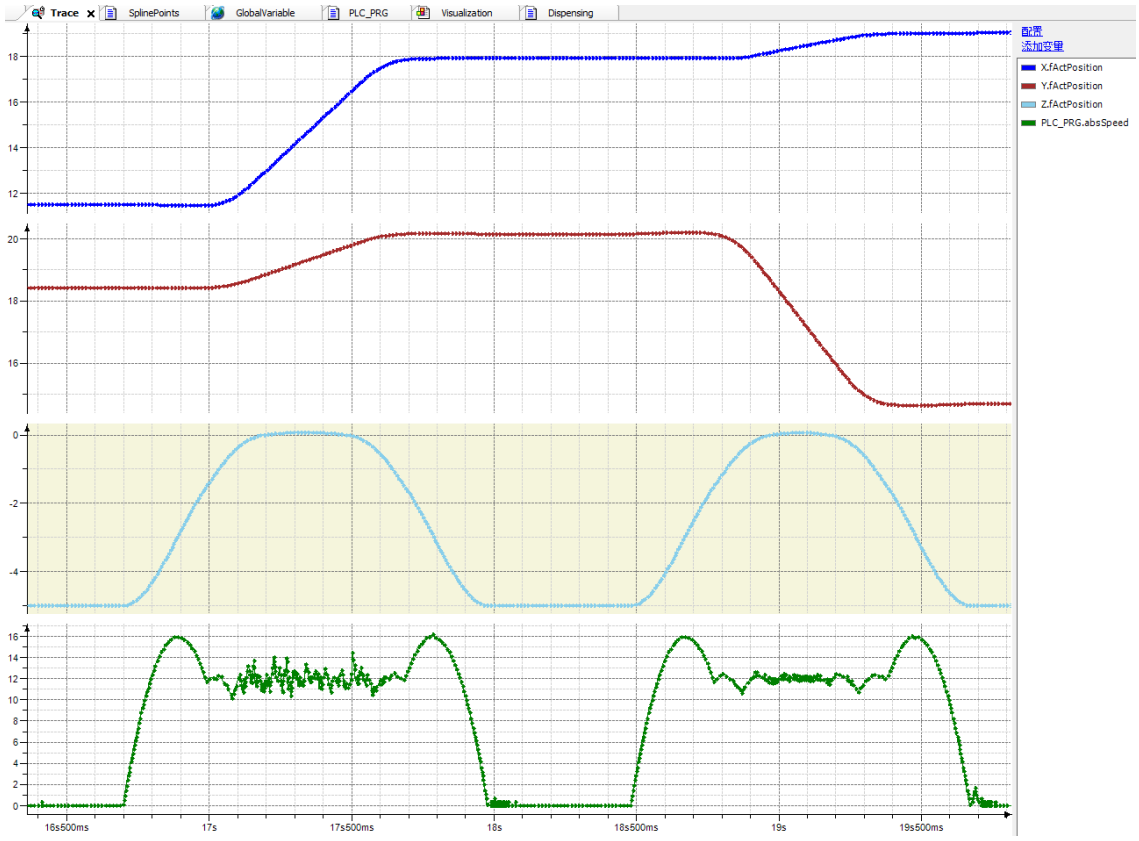

图 6.18　图 6.17 局部放大图

第 7 章　ST 程序与 C#程序的通讯

由于雷赛 PAC9300 运动控制器的运动控制软件独占一个 CPU 核。所以，可以充分利用 CPU 资源，在运行运动控制程序的同时，也可以使用 CPU 的另外 3 核运行其他软件完成自动化设备上所需的工作，如：加工轨迹的预处理、图像处理、生产管理、质量管理等。

PAC 中的运动控制程序与其他软件之间的数据交换通过读写共享内存得以实现。

共享内存是内存映射文件的一种特殊形式。如图 7.1 所示，共享内存映射的是一块物理内存，而非磁盘上的文件。2 个进程通过虚拟地址空间及用户级页表可以读、写物理内存中的同一块区域，从而实现进程间的通讯。该方式通讯效率高。

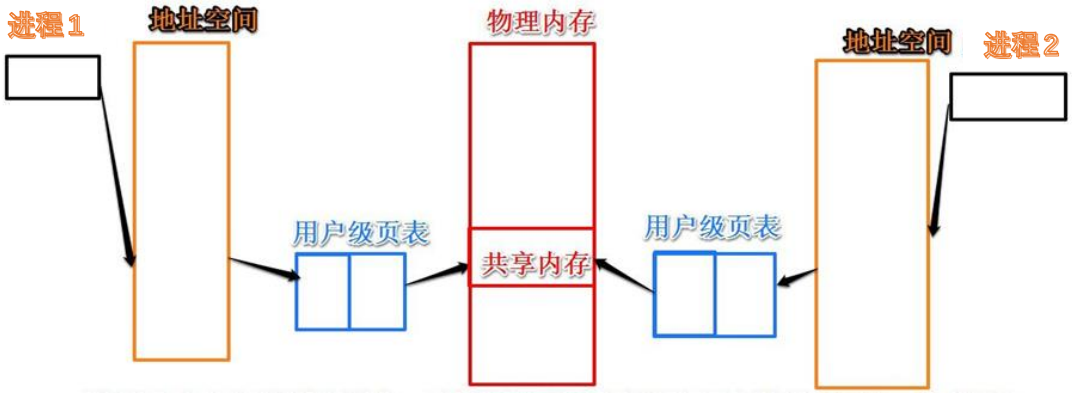

图 7.1　PAC 中 2 个进程通过共享内存通讯

下面通过例程 6 介绍 ST 程序与 C#程序进行通讯的方法。

例程 6 的功能为：每隔 1 秒钟 ST 程序读共享内存，获得 2 个 C#程序发送过来的数据；同时将 2 个数据写入共享内存，让 C#程序读取。

同样，C#程序每隔 1 秒通过共享内存给 ST 程序发送 2 个数据，再接受 2 个数据。

为了提高软件的可靠性，2 个程序不同时写一个共享内存。

7.1. C#程序读写共享内存的方法

存放在共享内存的数据为结构类数据。因此，在使用共享内存前，要定义自己的结构体。

程序界面如图 7.2 所示，代码如下。

```
using System;
using System.IO.MemoryMappedFiles;
using System.Runtime.InteropServices;
using System.Threading;
using System.Windows.Forms;

namespace AppB
{
```

```
public struct DataExchangeCommand        // 声明一个结构体
{
    internal int i1;                     // 对应与ST的DINT
    internal double i2;                  // 对应与ST的LREAL
}

public struct DataExchangeResults        // 声明一个结构体
{
    internal int i1;
    internal double i2;
}

public partial class Form1 : Form
{
    private Thread Job1; //  定义线程名 Job1

    public Form1()
    {
        InitializeComponent();
    }

    private void button1_Click(object sender, EventArgs e)  // 开始
    {
        Job1 = new Thread(MyJob1);               //  创建线程
        Job1.Priority = ThreadPriority.Highest; //  线程Job1优先级最高
        Job1.Start();                            //  启动线程
    }

    private void button2_Click(object sender, EventArgs e)  // 停止
    {
        Job1.Abort();  //  终止线程
    }

    private void MyJob1()   //  线程1的方法
    {
        string Wdata = "", Rdata = "";
        int writeValue1 = 0, readValue1 = 0;
        double writeValue2 = 0, readValue2 = 0;

        DataExchangeCommand dataExchangeW;
        DataExchangeResults dataExchangeR;      // 实例化2个结构体

        int dataSizeW = Marshal.SizeOf(typeof(DataExchangeCommand));
        int dataSizeR = Marshal.SizeOf(typeof(DataExchangeResults));
        //给结构体实例分配空间
        // using：定义一个范围，在范围结束时处理对象。 var：是匿名类型
        using (var commandWrite = MemoryMappedFile.CreateOrOpen
                    ("SharedMemoryPAC_Read", dataSizeW))
        using (var resultRead = MemoryMappedFile.CreateOrOpen
                    ("SharedMemoryPAC_Write", dataSizeR))
        // CreateOrOpen:  在系统内存中创建或打开一个具有指定容量的内存映射文件
        {
```

```
                while (true)
                {
                 using (var accessorWrite = commandWrite.CreateViewAccessor
                             (0, dataSizeW, MemoryMappedFileAccess.Write))
                 using (var accessorRead = resultRead.CreateViewAccessor
                             (0, dataSizeR, MemoryMappedFileAccess.Read))
  //创建MemoryMapperdViewAccessor,映射到视图的内存映射文件中，并指定偏移、大小和访问限制。
                 {
                    writeValue1 = writeValue1 + 2;
                    writeValue2 = writeValue2 + 2.2222;
                    dataExchangeW.i1 = writeValue1;
                    dataExchangeW.i2 = writeValue2;
                    accessorWrite.Write(0, ref dataExchangeW);
                       // 写入共享内存SharedMemoryPAC_Read
                   Wdata = "  "+writeValue1.ToString()+"\n\n  "+writeValue2.ToString();
                   this.Invoke((MethodInvoker)delegate {this.richTextBox1.Text = Wdata;});
                                        //  跨线程操作，在Form1线程上执行指定的委托
                   accessorRead.Read(0, out dataExchangeR);
                   // 读共享内存SharedMemoryPAC_Write
                   readValue1 = dataExchangeR.i1;
                   readValue2 = dataExchangeR.i2;
                   Rdata = "  " + readValue1.ToString() + "\n\n  "+readValue2.ToString();
                   this.Invoke((MethodInvoker)delegate{this.richTextBox2.Text = Rdata;});
                   Thread.Sleep(1000);// 延时1000ms
                    }
                }
            }
        }
    }
}
```

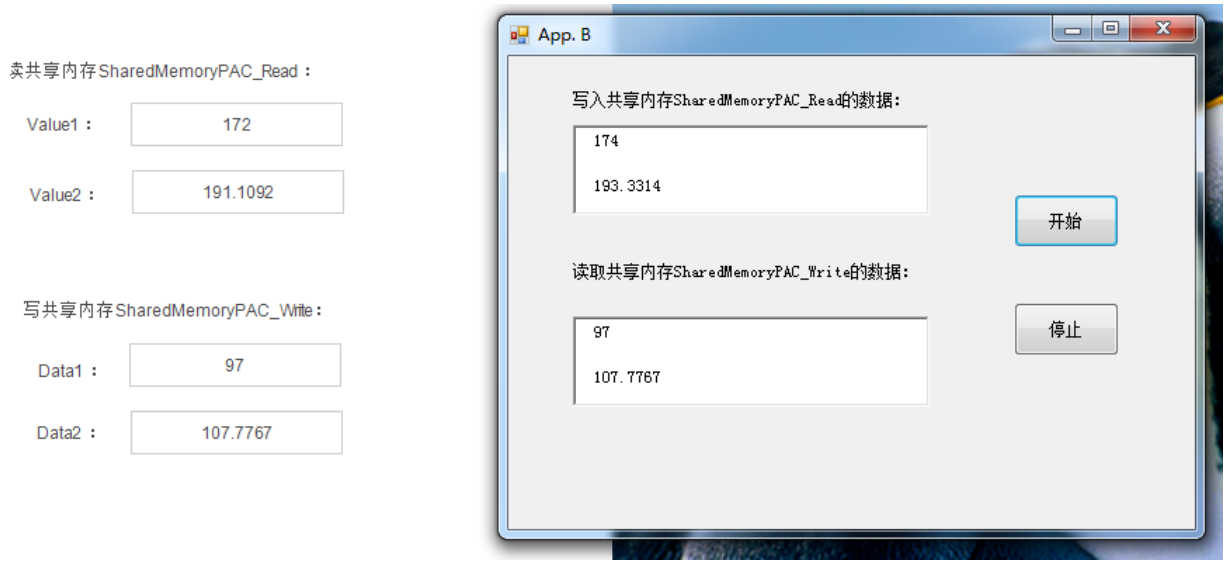

图 7.2　PAC 运行 2 个程序并通过共享内存交换数据

注意：用 MemoryMappedFile.CreateOrOpen 和 CreateViewAccessor 函数初始化共享内存后，就可以反复读写共享内存了。在退出程序前要释放共享内存空间。

C#程序在 PC 机上编写、调试完成后，生成安装文件；然后在 PAC 上安装、运行即可。

7.2. ST 程序读写共享内存的方法

A. 定义结构类数据

如图 7.3 所示，右键点击“Application”，在“添加对象”中选“DUT”，弹出对话框如图 7.4 所示。填入结构类数据的名称。

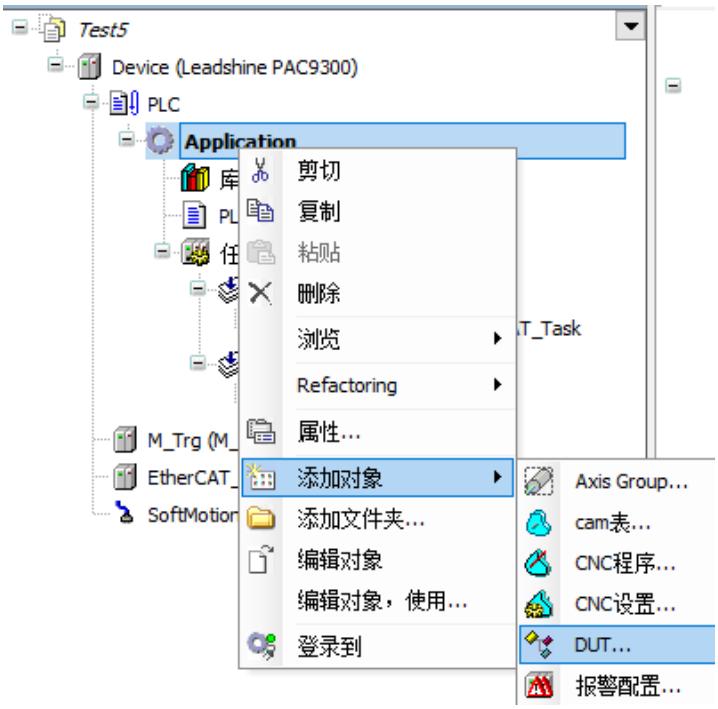

图 7.3　添加结构类数据

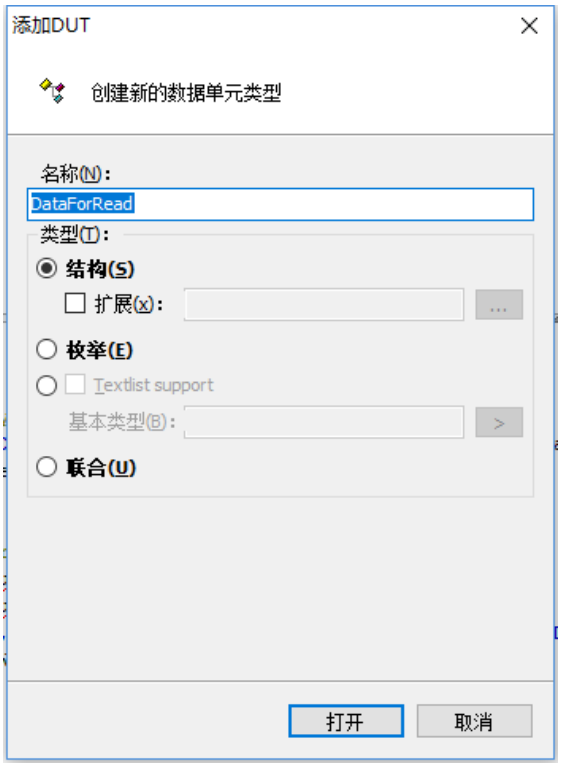

图 7.3　设置结构类数据的名称

在项目树中双击结构类数据的名称，打开结构类数据的编辑界面。定义结构类数据如下：

```
TYPE DataForRead :
STRUCT
  Value1 : DINT;
  Value2 : LREAL;
END_STRUCT
END_TYPE
```

以同样方法再定义一个结构类数据 DataForWrite。

B. 添加 2 个和共享内存有关的库

点击项目树中的“库管理器”，在“库管理器”界面里点击“添加库”，如图 7.4 所示。点击“高级”，弹出对话框如图 7.5 所示。

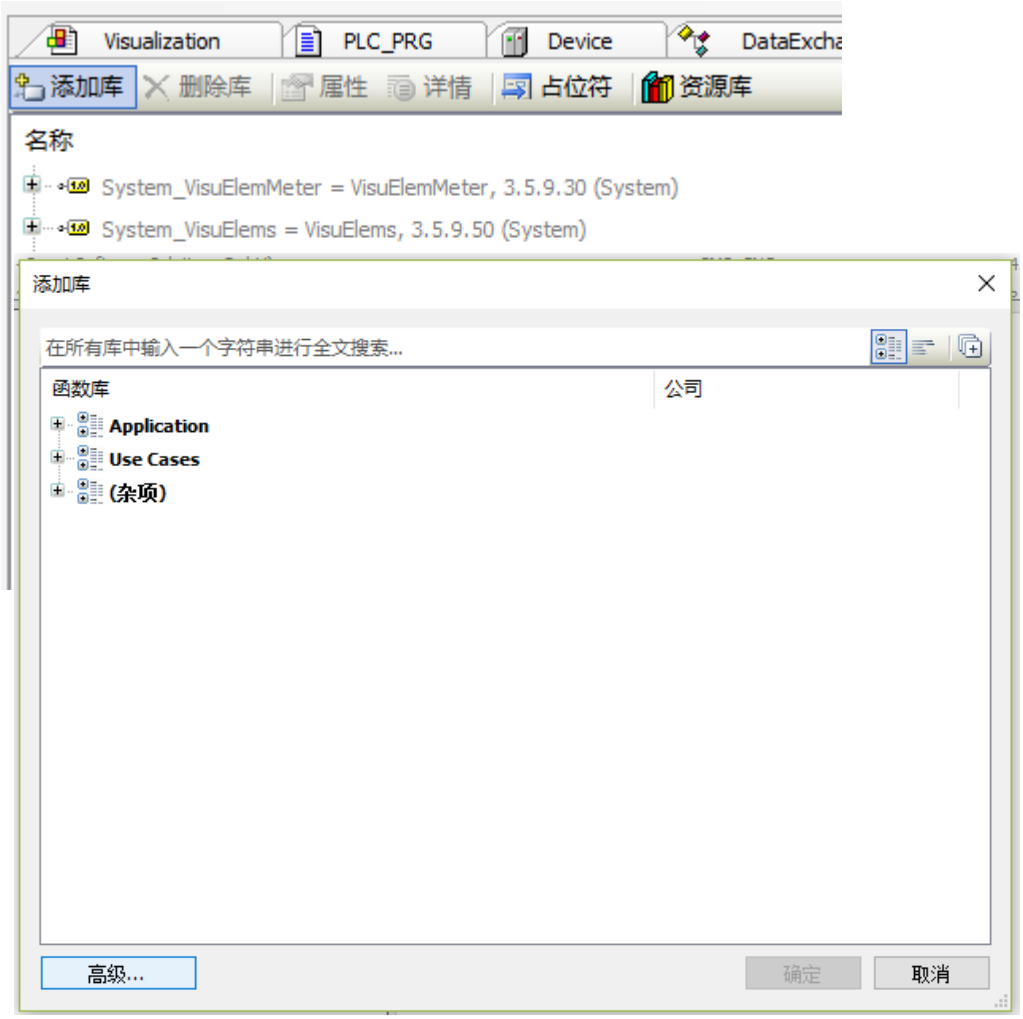

图 7.4　添加库

在“System”下的“SysLibs”中选“SysShm”，再点“确定”。用同样的方法再添加一个 SysTypesInterfaces 库。

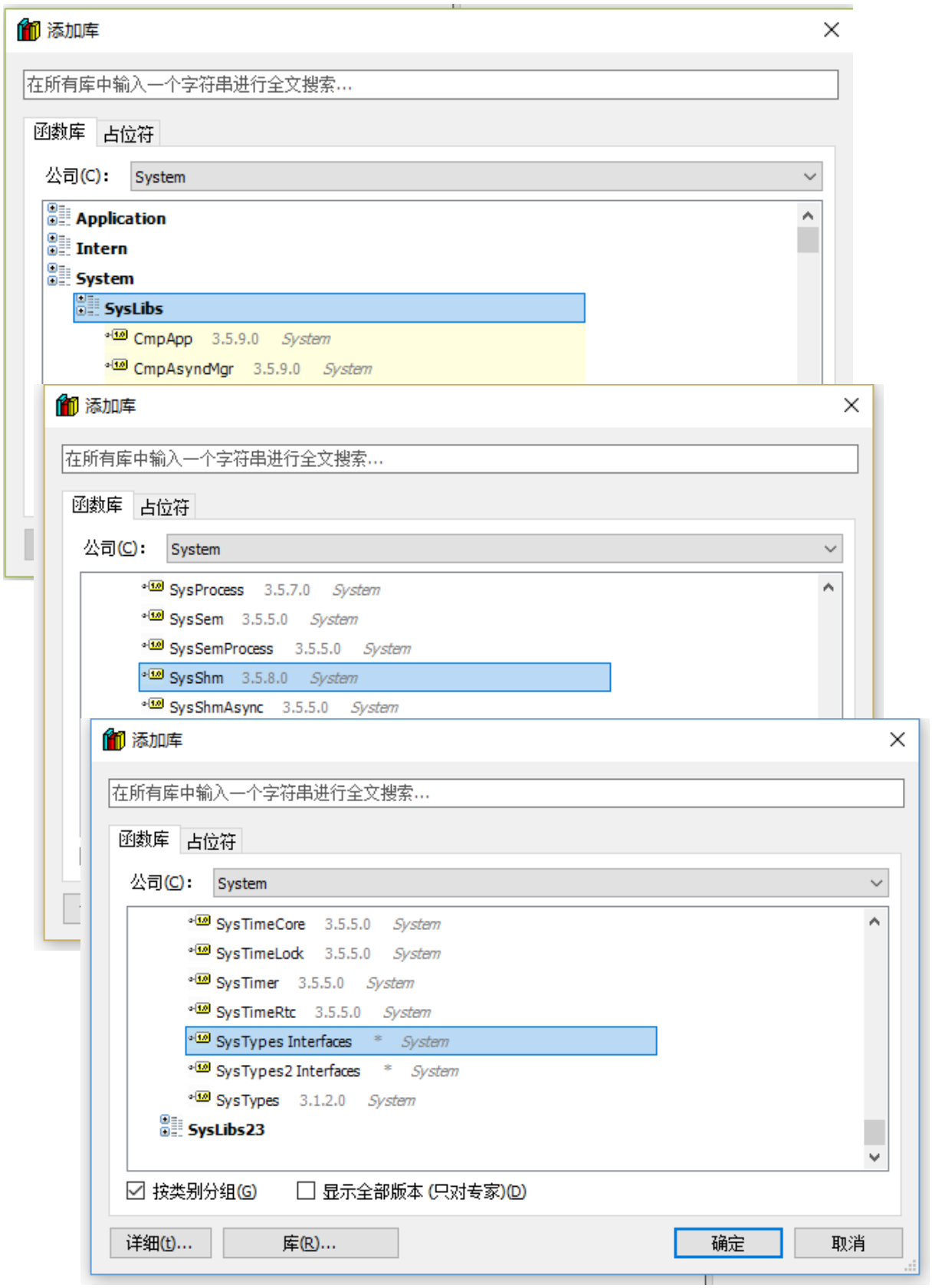

图 7.5　添加 2 个与共享内存有关的库

C. 编程代码

结构化文本语言有关共享内存的函数，与 C#的类似，但没有创建 View 对象的函数。在创建内存映射文件时，文件名要和 C#的一样。详细代码如下：

```
PROGRAM PLC_PRG
VAR
  tt: Ton;

  MemRead: RTS_IEC_HANDLE;
  ResultRead: RTS_IEC_RESULT;
  ReadStructData : DataForRead;
  ReadResult: RTS_IEC_RESULT;
  ReadData: __UXINT;
  RDataSize: UDINT := SIZEOF(DataForRead);

  MemWrite: RTS_IEC_HANDLE;
  ResultWrite: RTS_IEC_RESULT;
  WriteStructData : DataForWrite;
  WriteResult: RTS_IEC_RESULT;
  WriteData: __UXINT;
  WDataSize: UDINT := SIZEOF(DataForWrite);

  MemVal1 : DINT;
  MemVal2 : LREAL;

  MemData1 : DINT;
  MemData2 : LREAL;

END_VAR


tt(IN := TRUE,PT:=T#1000MS);
IF tt.Q THEN
  // 从共享内存 SharedMemory_Read 中读数据
  MemRead := SysSharedMemoryCreate('SharedMemoryPAC_Read', 0, ADR(RDataSize),
                                   ADR(ResultRead));
  ReadData := SysSharedMemoryRead(MemRead, 0, ADR(ReadStructData),
                                  SIZEOF(ReadStructData), ADR(ReadResult));
  ReadData := SysSharedMemoryClose(hShm := MemRead);

  MemVal1 := ReadStructData.Value1;
  memVal2 := ReadStructData.Value2;

  MemData1 := MemData1 +1;
  MemData2 := MemData2 +1.1111;

  // 向共享内存 SharedMemory_Write 中写数据
  MemWrite := SysSharedMemoryCreate('SharedMemoryPAC_Write', 0, ADR(WDataSize),
                                    ADR(ResultWrite));
  WriteStructData.Data1 := MemData1;
  WriteStructData.Data2 := MemData2;
  WriteData := SysSharedMemoryWrite(MemWrite, 0, ADR(WriteStructData),
```

```
                                        SIZEOF(WriteStructData), ADR(WriteResult));
    WriteData := SysSharedMemoryClose(hShm := MemWrite);

    tt(IN:=FALSE);
    tt(IN:=TRUE);
  END_IF
```

D. 编写界面

该程序的界面及文本框对应的变量如图 7.6 所示。

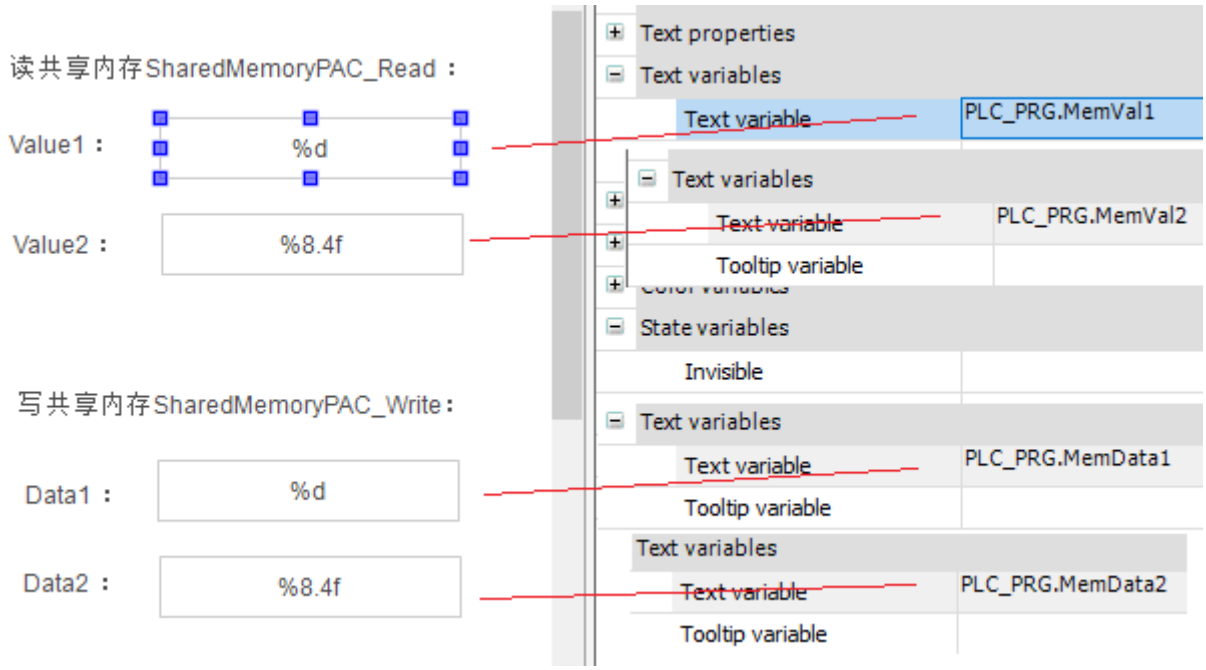

图 7.6　程序界面及文本框的变量

E. 运行结果

在 PAC 上同时运行以上 2 个程序，结果如图 7.2 所示。