

Motion Studio 用户开发手册

V1.2

目录

Motion Studio 用户开发手册	0
第一章 Motion Studio 基本介绍.....	4
1.1 Motion Studio 软件简介	5
1.2 安装环境需求	6
1.3 安装与删除 Motion Studio 软件	7
1.4 软件更新	12
第二章 Motion Studio 项目操作.....	13
2.1 建立 Motion Studio 与 MAS 控制器连接.....	14
2.2 新建项目	18
2.3 基本调机步骤	21
2.4 机器程序	29
第三章 变量使用.....	38
3.1 变量种类	39
3.2 用户自定义变量	40
3.3 VR 变量.....	43
3.4 Table 变量	48
第四章 输入输出控制.....	53
4.1 DIO 控制	54
4.2 AIO 控制	60
第五章 气缸控制.....	68
5.1 如何配置气缸	70
5.2 如何测试气缸控制	74
5.3 如何编写程序控制气缸动作	75
5.4 Cylinder 类.....	78
第六章 运动控制.....	84
6.1 执行运动控制的基本步骤	85
6.2 BASE:指定操作轴	86
6.3 WAIT DONE:等待运动完成	87

6.4	相对运动指令和绝对运动指令.....	90
6.5	基础轴控运动	91
6.6	回原点运动	102
6.7	如何停止运动	106
6.8	高速比较触发	108
6.9	高速位置锁存	115
6.10	附：基本运行控制指令一览	119
第七章	流程控制.....	120
7.1	WHILE 循环.....	121
7.2	IF 条件判断	123
7.3	FOR 循环.....	125
7.4	Case 条件判断.....	127
7.5	TMR 计时器类.....	129
7.6	MS_LOOP 循环	131
第八章	子程序使用	133
8.1	子程序的使用步骤	134
8.2	SUB 和 FUNCTION 的区别.....	135
8.3	SUB 子程序.....	136
8.4	FUNCTION 子程序	137
第九章	多任务编程.....	138
9.1	TASK 相关指令一览	139
9.2	多个 TASK 运行时的关系	140
9.3	如何用 TASK 实现循环任务	141
9.4	如何用 Task 启动另一个 Task	143
9.5	如何用 TASK 实现中断功能	144
9.6	如何用 TASK 指令实现单步运行	147
9.7	子 TASK 的使用	149
第十章	与外部通信.....	152
10.1	外部通信基本步骤	153



10.2 串口通信	154
10.3 TCP/IP 通信	158
第十一章 编辑用户界面	165
11.1 调用 API 文件编辑用户界面	166
11.2 使用 MS HMI.NET 控件编辑用户界面	168
第十二章 应用示例	171
12.1 孔径加工	172
12.2 码垛	174
终章 附录	178
附录 1 手册使用导引	179
附录 2 影响 Motion Runtime 运行的因素	181

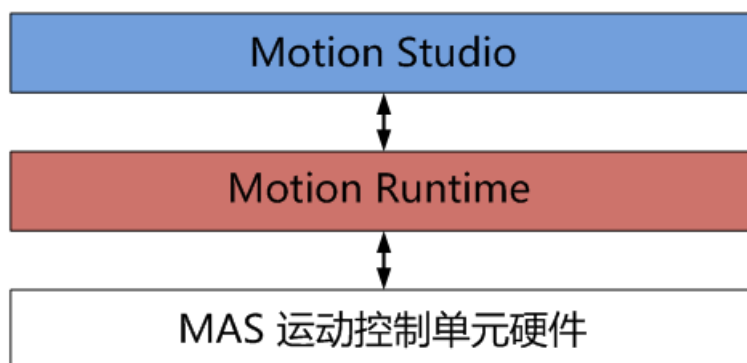
第一章 Motion Studio 基本介绍

1.1 Motion Studio 软件简介

Motion Studio 是用于 MAS 控制器调试和开发的一套工具软件,它集合了运动控制、状态监测、参数设置、轨迹显示、系统调试与编程等功能,让使用者快速使用 MAS 控制器并完成设备应用开发。

Motion Studio 包含两个部分: Motion Studio 和 Motion Runtime

-  • Motion Studio: 集成开发环境。
-  • Motion Runtime: Motion Studio 的后台运行程序。它管控着外部与 MAS 控制单元之间的所有交互。**(MAS 控制器要正常运行, Motion Runtime 一定要运行着)**



1.2 安装环境需求

◆ 硬件需求

1. Pentium 1 GHz(含)以上
2. 1 GB 以上内存
3. 至少 500MB 的硬盘空间

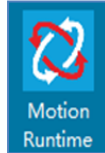
◆ 软件环境

1. Win7/Win8/Win10 操作系统 (32bit/64bit)
2. Microsoft .NET Framework 4.0(含)以上版本。

1.3 安装与删除 Motion Studio 软件

1.3.1 安装 Motion Runtime

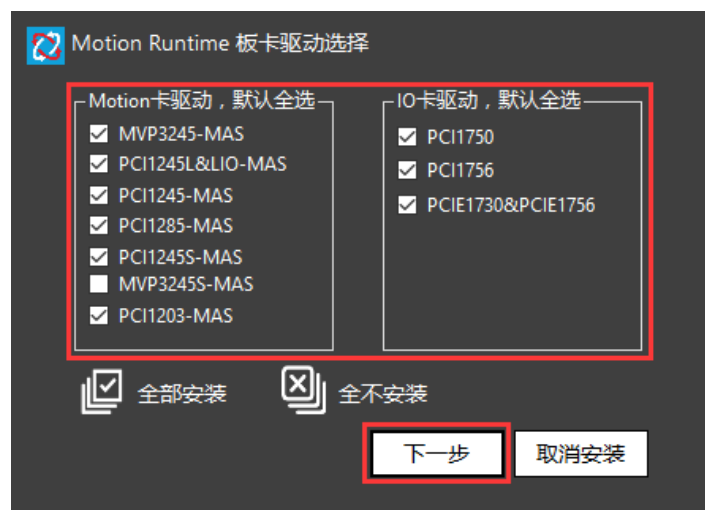
步骤一：双击 Motion Runtime 安装包。



步骤二：点击下一步。



步骤三：选择实际的实体运动控制单元和 I/O 控制单元，点击下一步。

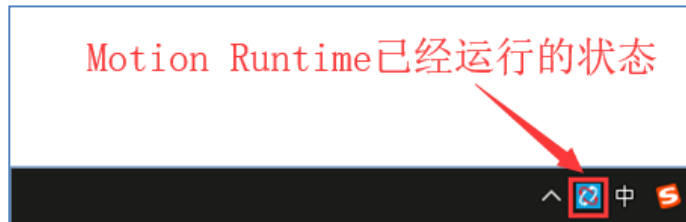


步骤四：等待安装进度完成，点击安装完成。Motion Runtime 会默认安装在本地路径 C:\Advantech\Motion_Runtime 下。



◆ **注意：**

1. Motion Runtime 安装完后需重启电脑。
2. Motion Runtime 会在 Windows 启动时自动运行起来。Motion Runtime 正常运行起来后可以在电脑任务栏看到 Motion Runtime 运行的图标。



1.3.2 安装 Motion Studio

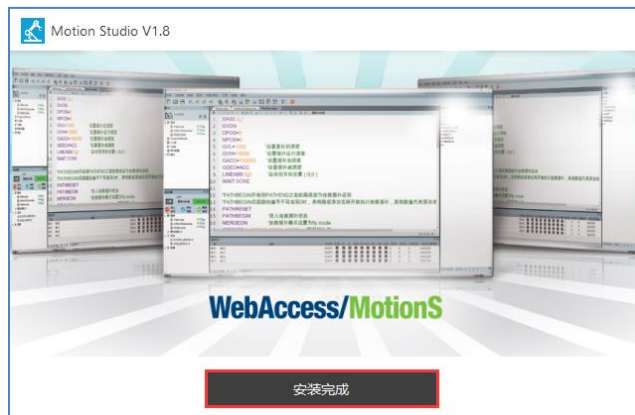
步骤一：双击 Motion Studio 安装包。



步骤二：点击下一步。

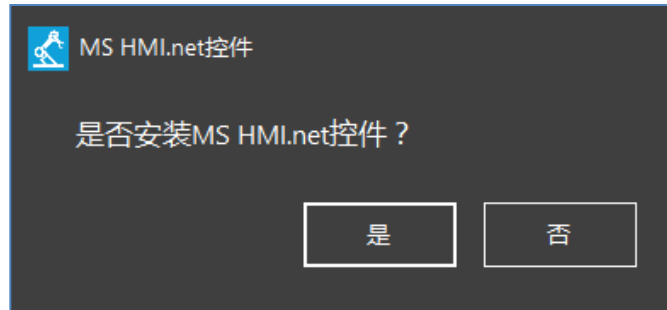


步骤三：等待安装进度完成，点击安装完成。Motion Studio 会默认安装在本地路径 C:\Advantech\Motion Studio 下。



◆ **注意:**

Motion Studio 安装完后, 会弹出是否安装 HMI.NET 控件对话框, 如果此电脑已安装 Visual Studio 2010 版本(含)以上软件, 点击是, 否则点击否。HMI.NET 控件的安装说明请参考第 11 章。



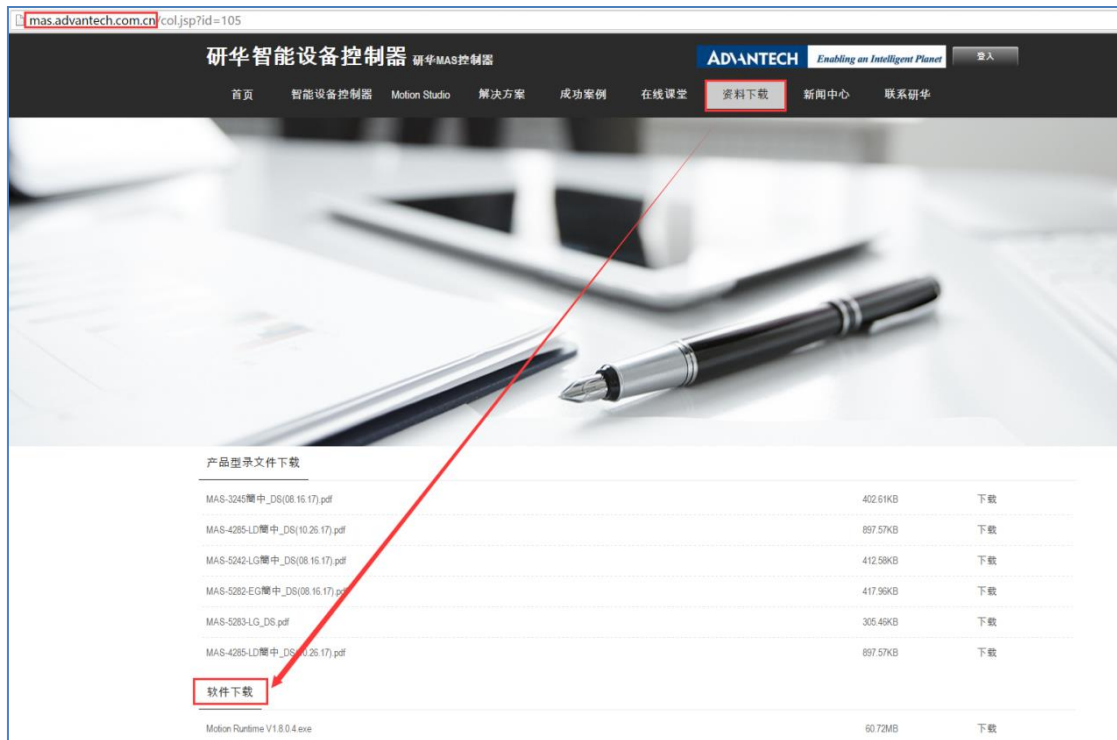
1.3.3 软件删除

软件删除请至电脑“控制面板\程序\程序和功能\卸载或更改程序”。

1.4 软件更新

请至研华 MAS 控制器官方网站 <http://mas.advantech.com.cn/> 下载软件。再按下面步骤更新软件。

1. 卸载原来的 Motion Runtime 和 Motion Studio 软件。
2. 再安装新的 Motion Runtime 和 Motion Studio 软件



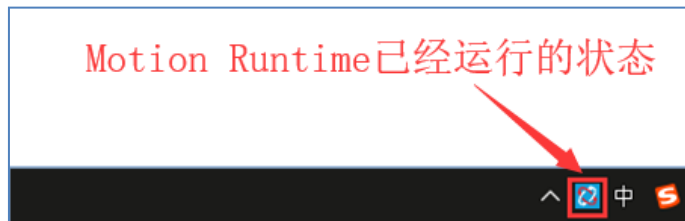
第二章 Motion Studio 项目操作

2.1 建立 Motion Studio 与 MAS 控制器连接

要让 Motion Studio 对 MAS 控制器进行操作，需先建立 Motion Studio 与 Motion Runtime 的连接。以下将介绍 Motion Studio 连接不同类型控制器的方式。

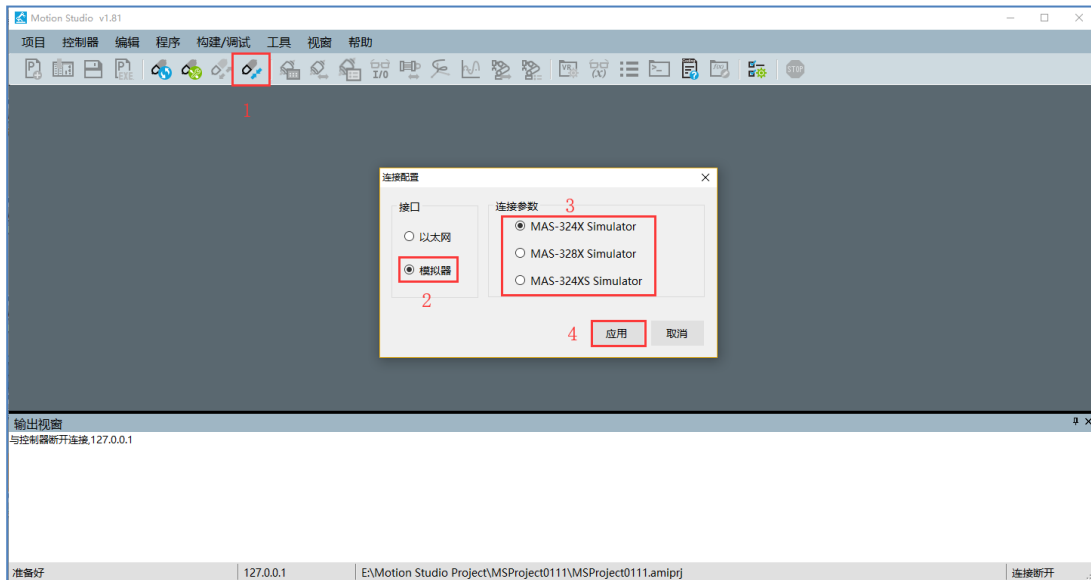
◆ **注意：**

1. Motion Studio 可以和 Motion Runtime 装在不同的电脑上。
2. 安装了 Motion Runtime 的电脑才是要控制的 MAS 控制器。
3. 进行连接之前，要先确认 MAS 控制器中 Motion Runtime 已经运行起来。如下图，可以在电脑任务栏中查看 Motion Runtime 是否已经运行起来。



2.1.1 如何连接虚拟控制器（模拟器）

Motion Studio 如何连接虚拟控制器，只需按下面操作即可。



1. 打开连接设置
2. 选择“模拟器”接口
3. 选择一种模拟器。
 - a) MAS-324X Simulator: 模拟通用 4 轴控制器, 32DI, 32DO
 - b) MAS-328X Simulator: 模拟通用 8 轴控制器, 64DI, 64DO
 - c) MAS-324XS Simulator: 模拟通用 4 轴 SCARA 控制器, 32DI, 32DO
4. 点击应用, 进行连接。

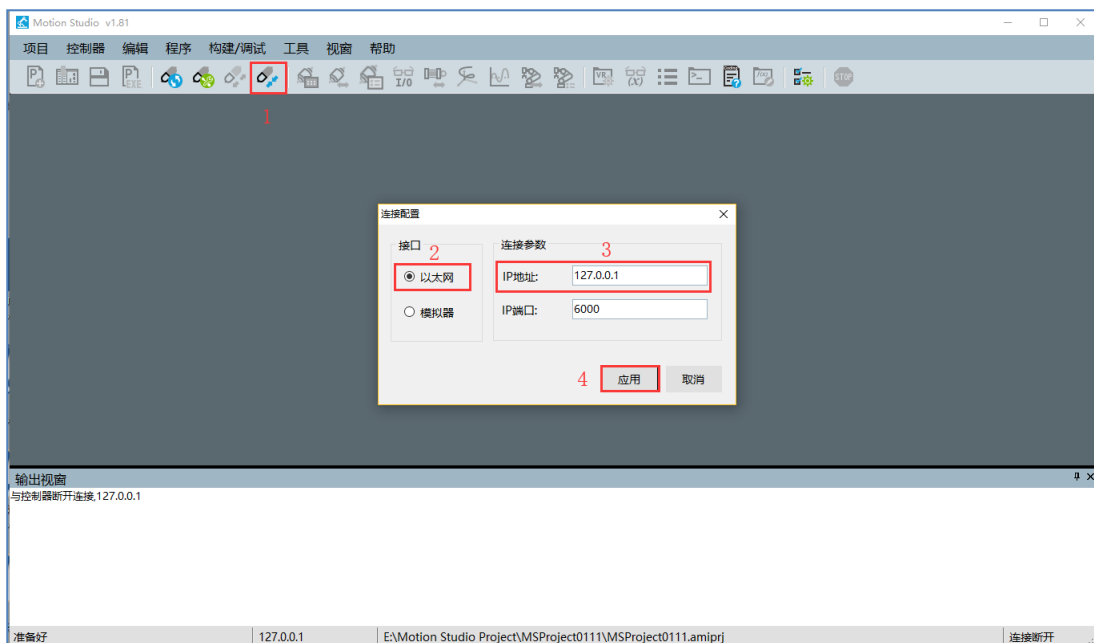
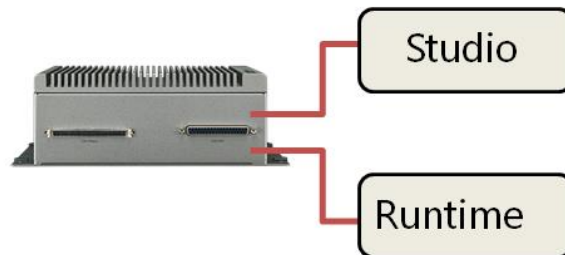
◆ **注意:**

1. 连接虚拟控制器时, Motion Studio 只能连同一台电脑的运动 Runtime。
2. 用户常使用笔记本电脑连虚拟控制器, 用于编程学习和设备动作控制模拟。

2.1.2 如何连接实体控制器

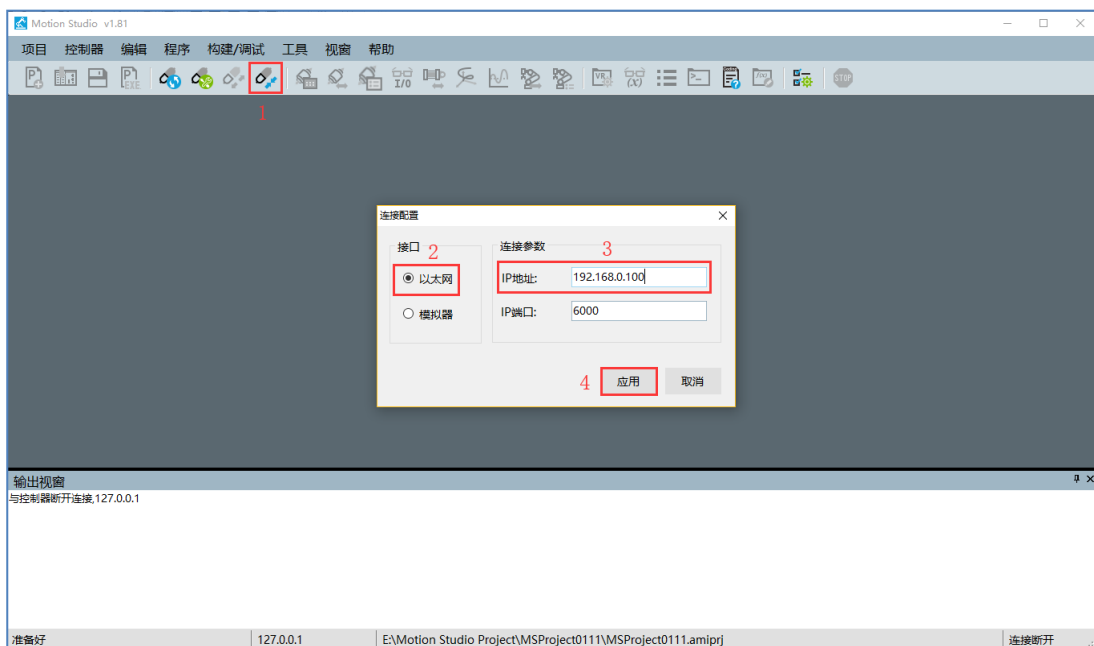
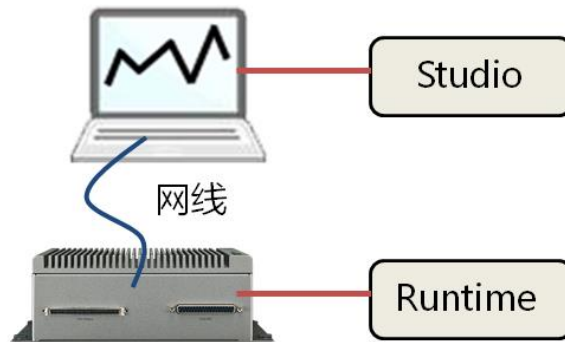
连实体控制器分两种情况。

- ◆ 情况 1: Motion Studio 和 Motion Runtime 在同一台电脑的情况, 连接步骤如下。



1. 打开连接设置
2. 选择“以太网”接口
3. IP 地址填“127.0.0.1”
4. 点击应用, 进行连接。

- ◆ 情况 2: Motion Studio 和 Motion Runtime 在不同电脑的情况, 连接步骤如下。



1. 打开连接设置
2. 选择“以太网”接口
3. IP 地址填“MAS 控制器所在电脑的网口 IP 地址”, 且 Motion Runtime 和 Motion Studio 连接网口的 IP 地址需在同一网段。

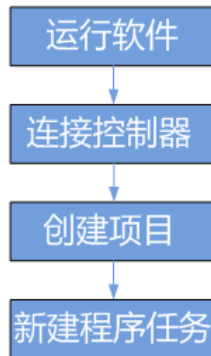
例: Motion Runtime 所在电脑的网口 IP 地址设置为 192.168.0.100, 那 Motion Studio 所在电脑的网口 IP 需设置为 192.168.0.X。X 的范围为 $1 \leq X \leq 255$, 不能为 100, 因为 X 为 100 的话, 两台电脑的网口 IP 就冲突了。

4. 点击应用, 进行连接。

2.2 新建项目

2.2.1 如何新建一个 Motion Studio 项目

要开始一个新的项目，需新建一个 Motion Studio 项目用于控制器的调试和编程。新建一个 Motion Studio 项目按以下步骤即可完成。

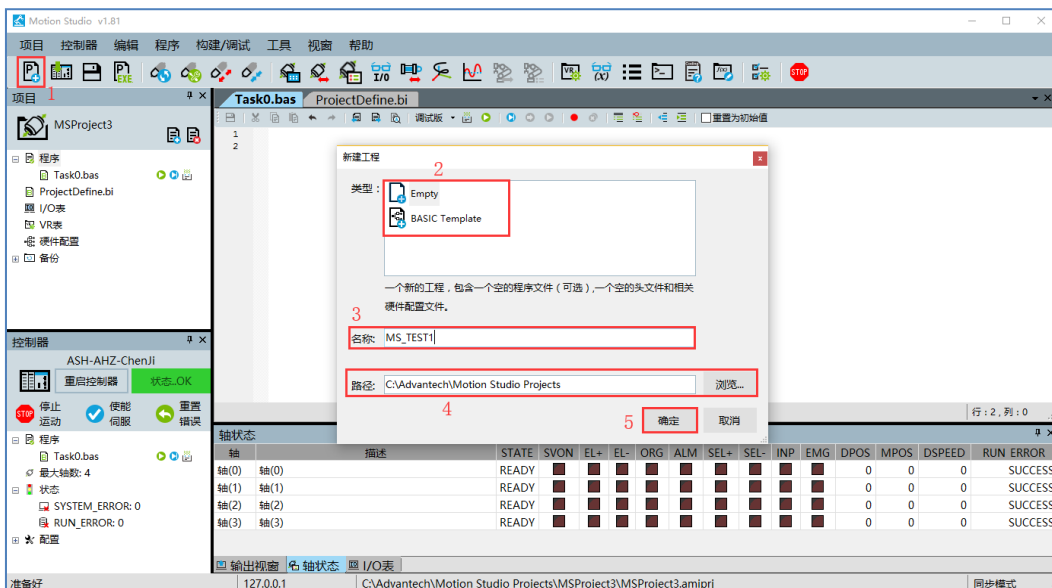


步骤一：运行软件

1. 确认 Motion Runtime 已经运行起来 (Motion Runtime 一般在电脑启动时会自启动)。
2. 如果没有，启动 Motion Runtime。
3. 双击桌面 Motion Studio 图标，启动 Motion Studio。

步骤二：使用 Motion Studio 连接控制器（请参考 2.1 章节的操作步骤）

步骤三：创建项目，依照如下小步创建项目。

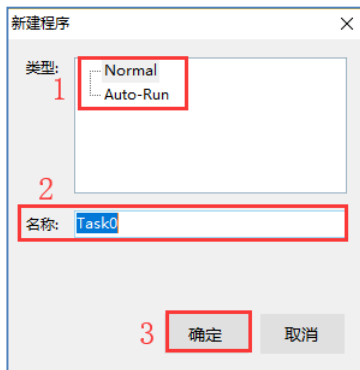


1. 新建项目。
2. 选择项目类型，创建一个空的项目选择“Empty”类型。

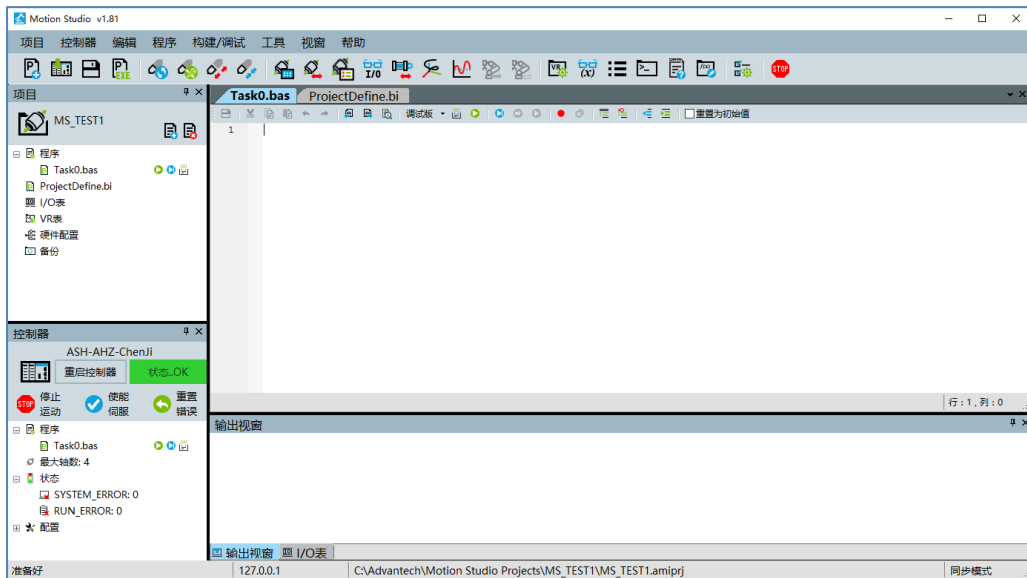
3. 填写项目名称。
4. 填写项目存放的路径。
5. 点击确定，进入下一步。

步骤四：新建程序任务

新建 Motion Studio 项目时，需先新建一个程序任务（Task）。后面可以在 Motion Studio 中添加多个程序任务。新建程序任务的操作步骤如下。

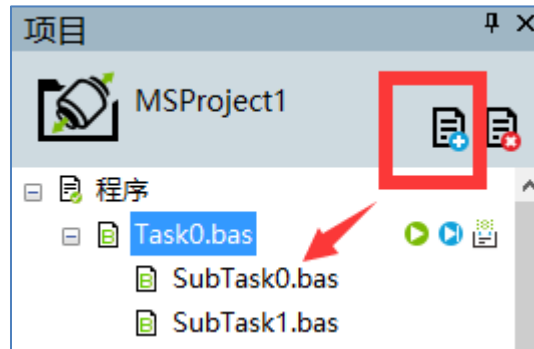


1. 选择 TASK 类型。
2. 填写 TASK 名称。
3. 点击确定。一个新项目创建完毕，Motion Studio 的界面呈现如下。



步骤五：为 TASK 建立子 TASK

子 TASK 并非项目必要，关于子 TASK 的使用请见 9.7 章节。



如上图，选中主 TASK，选择新建，就能为当前选中的主 TASK 建立一个子 TASK。

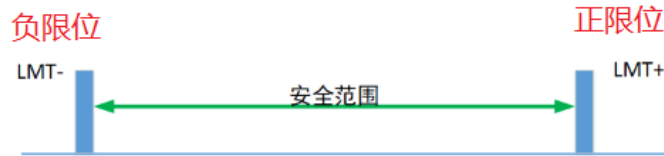
一个主 TASK 可拥有多个子 TASK。

2.3 基本调机步骤

如果只是连接虚拟控制器，学习 Motion Studio 的基本使用和 BASIC 编程，可以不看此章节。这里讲基本调机步骤是针对 MAS 控制器已连实际的电机，设备的硬件接线都已完成的情况。

控制器调机包含很多方面，本章节的基本调机步骤只是讲最基础的几个步骤，经过这几步，使用者可以编写简单的程序让设备动起来。

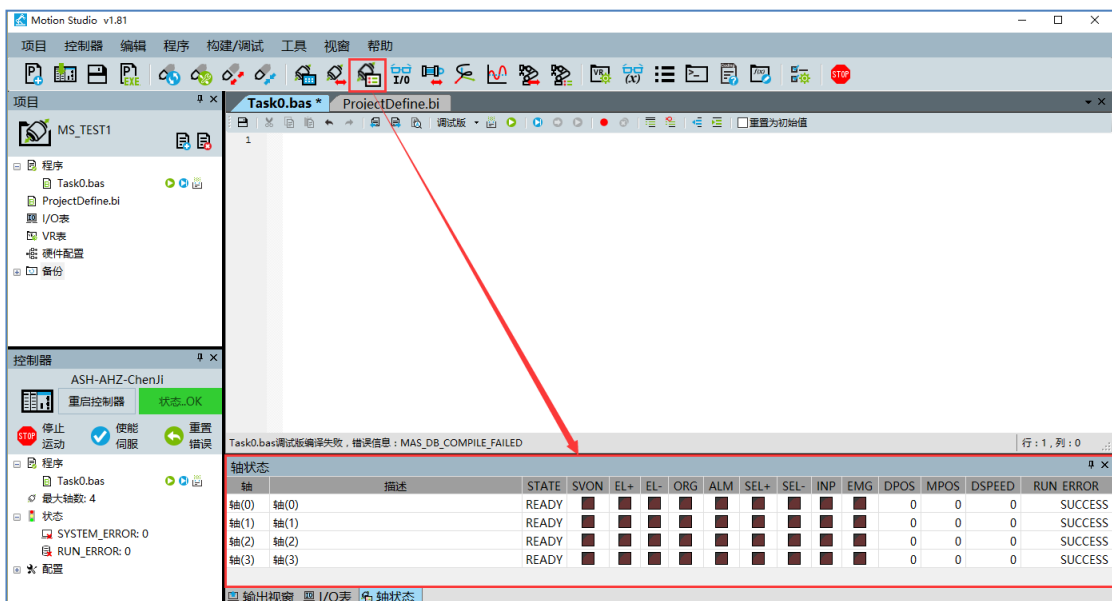
2.3.1 如何确认硬件限位



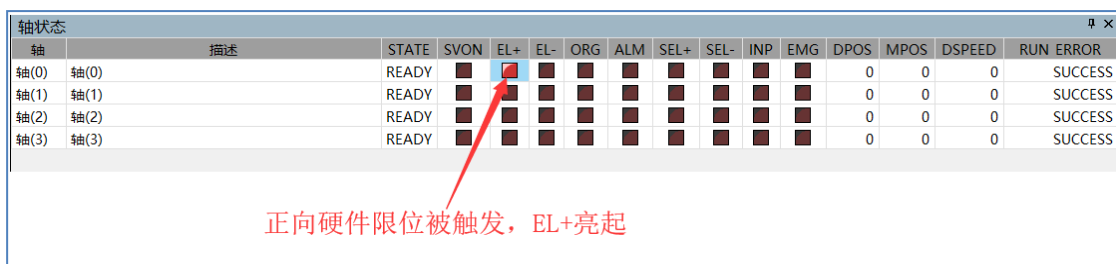
一般的运动控制设备,每个轴都会安装正负硬件限位,硬件限位可以防止运动部件超出限位区域,起到保护的作用。所以在控制电机运动前须先确认各轴硬件限位正常。

用下面步骤确认轴硬件限位正常(以轴 0 为例,其它轴用一样方法确认)。

步骤一：打开轴状态工具



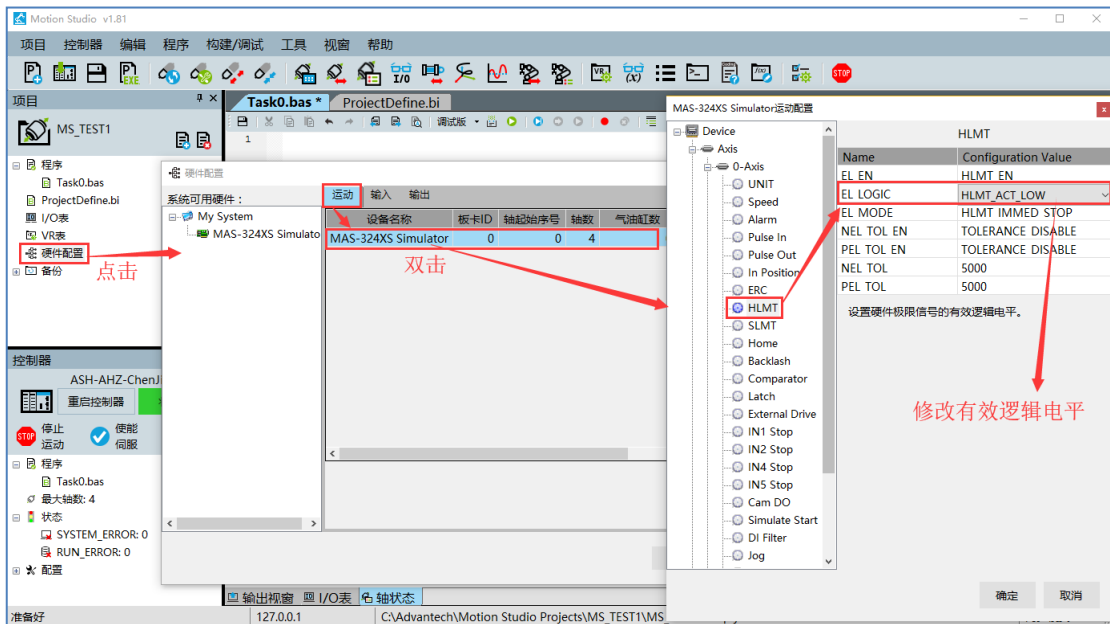
步骤二：手动去感应轴 0 的正向硬件限位，确认轴状态工具中 EL+ 亮起。



三种异常情况处理方法：

1. 正向硬件限位被触发,轴状态工具中 EL-灯亮起。则将正负硬件限位的信号线互换一下。
2. 正向硬件限位无论是否被触发,轴状态工具中 EL+灯都不亮。则检查限位开关是否是坏的,或硬件接线是否正确。

3. 正向硬件限位被触发，轴状态工具中 EL+灯不亮。没被触发，EL+灯反而亮起。则如下图，进入 Motion Studio 硬件配置中修改 EL_LOGIC 的逻辑电平。



步骤三：用步骤二的方法确认轴 0 负向硬件限位是否正常。



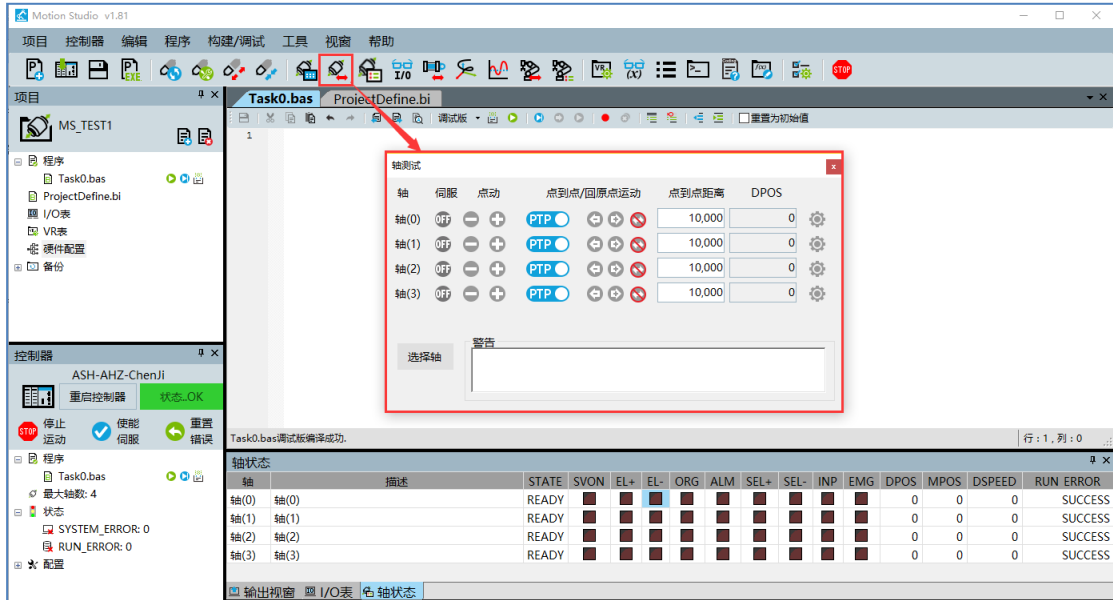
◆ 注意：

硬件限位的相关属性也可以通过程序设置。可以参考"Motion BASIC 使用手册"中的 EL_EN , EL_LOGIC 等相关 BASIC 指令。

2.3.2 如何进行电机试运转

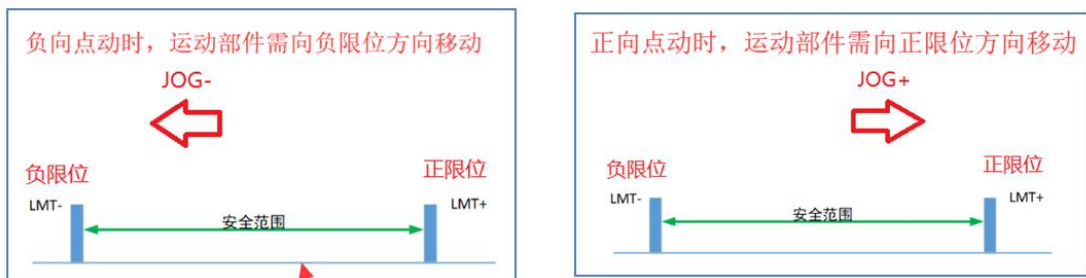
用下面步骤进行电机试运转(以轴 0 为例，其它轴用一样方法确认)，确认电机控制方向正确。

步骤一：打开轴运动测试工具

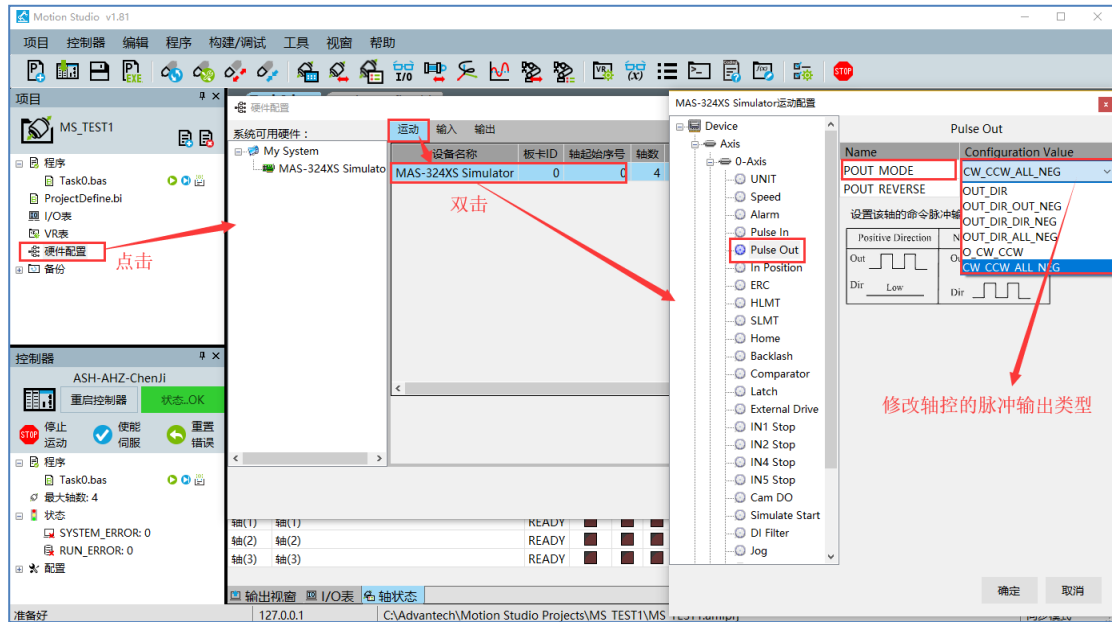


步骤二：在轴测试工具中，使用 JOG+ 和 JOG- 按钮使轴正向点动和负向点动，确认 2 点：

- 正向点动时，运动部件向正限位方向移动
- 负向点动时，运动部件向负限位方向移动



非上述现象时，如下图，进入 Motion Studio 硬件配置中修改轴控的脉冲输出类型。直到正负方向试运转的方向正确才算试运转正常。



2.3.3 如何设置用户单位

新的 Motion Studio 项目创建后，控制器默认控制输出的位移单位是脉冲，但是实际应用中，用脉冲为单位不是很直观，用户希望以实际需求来决定使用单位。比如直线运动，用户常以毫米 (mm) 为单位；旋转运动，用户常以角度为单位。

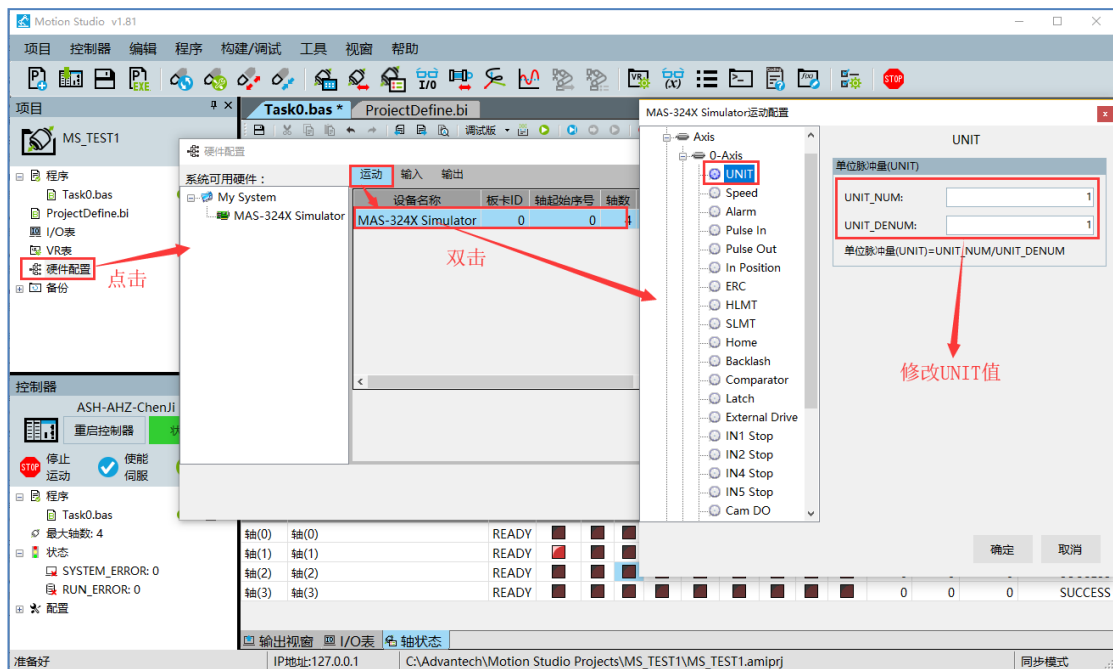
Motion Studio 中使用 UNIT 将脉冲转换成用户单位。说明如下：

- UNIT 的意思是单位脉冲量，即多少个脉冲为一个 UNIT。
- UNIT 是一个分数。由属性 UNIT_NUM (分子) 和 UNIT_DENOM (分母) 共同决定大小。

$$\text{UNIT} = \text{UNIT_NUM} / \text{UNIT_DENOM}$$
- UNIT_NUM 和 UNIT_DENOM 默认值为 1。这时，用户单位为脉冲。

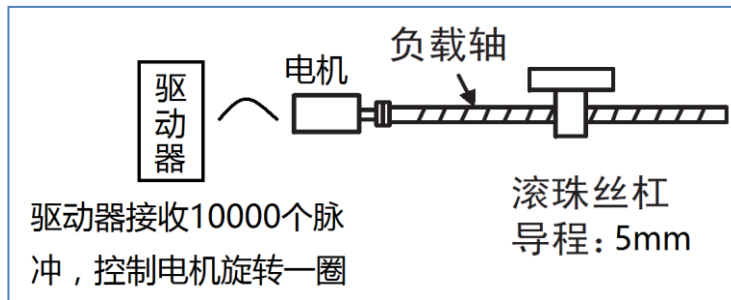
设置 UNIT 有两种方法：

- ◆ 方法 1：硬件配置里配置，如下图(以轴 0 为例)。



- ◆ 方法 2：程序里设置。可以参考"Motion BASIC 使用手册"中的 UNIT_NUM 和 UNIT_DENOM 两条指令。

例子： 如下图，用户伺服驱动器里设置了指令脉冲为 10000 脉冲/转，电机通过联轴器直连丝杠，丝杠的导程为 5mm。用户要以毫米(mm)为用户单位，该如何设置 UNIT？

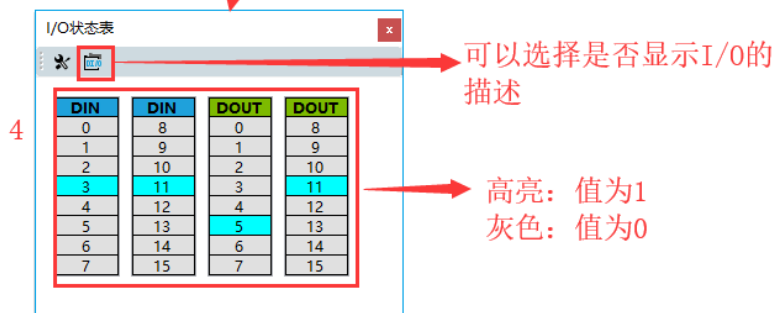
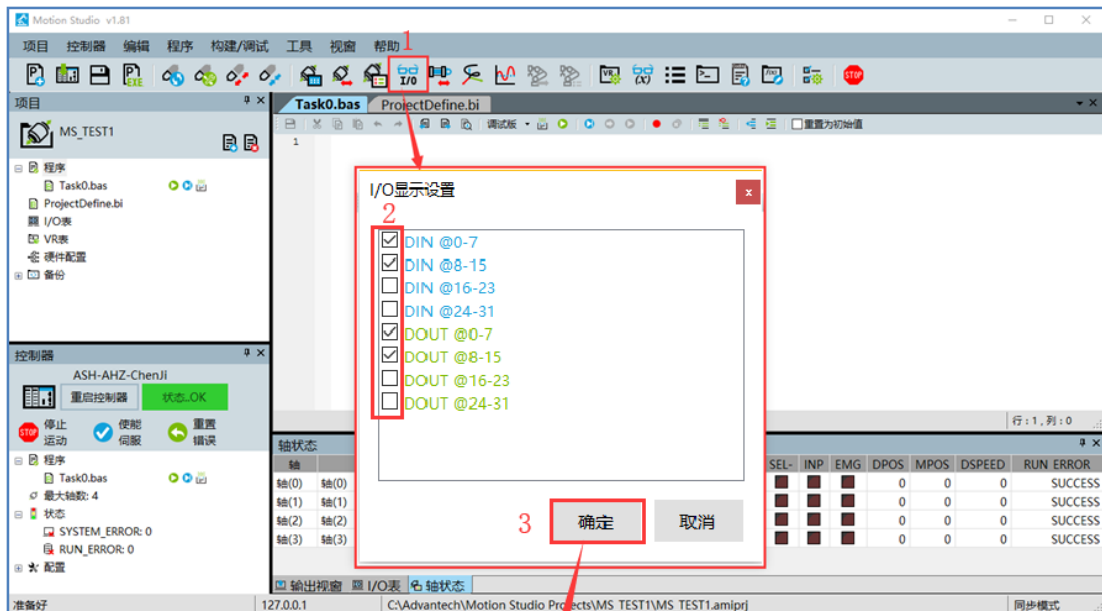


本例用户单位为 mm，所以要确定控制终端移动 1mm，需要多少个控制脉冲输出。从不本例中可知，10000 个脉冲可以控制电机旋转一圈，导程为 5mm，所以控制终端移动 1mm，驱动器需要接收 2000 个脉冲。

所以 UNIT 需设置为 2000，即 UNIT_NUM 设置为 2000，UNIT_DENOM 设置为 1。

2.3.4 如何进行 DIO 测试

设备电气部分初次接好线后，经常需要测试 DI/DO 硬件接线是否正常，以保证后续 I/O 编程控制没有问题。可以打开 Motion Studio 中的 I/O 工具进行测试，按如下步骤即可。

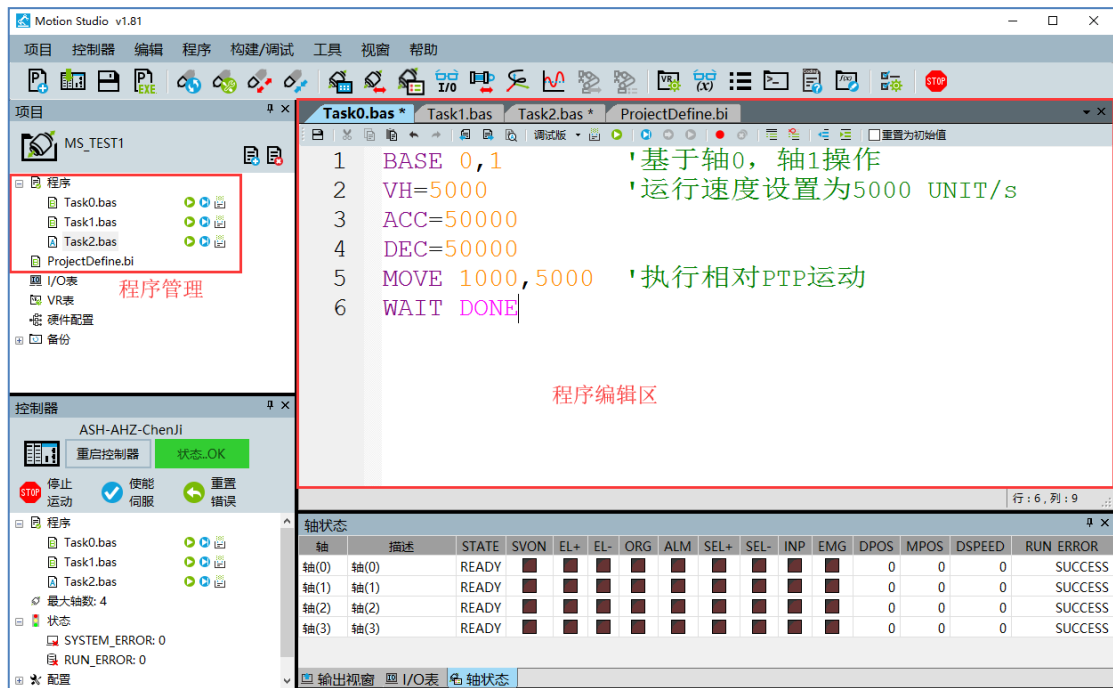


1. 打开 I/O 工具
2. 勾选要测试的 I/O 组别
3. 确定，打开 I/O 测试工具
4. 进行 I/O 测试。
 - a) 实时显示全部 DIO 状态
 - b) DO 可以手动点击置 1 还是置 0

2.4 机器程序

2.4.1 程序机制

Motion Studio 中的程序是在 MAS 控制器上执行指定任务的 BASIC 指令序列，包含任务程序 (Task) 和 ProjectDefine.bi。机器程序与机制说明如下。



- 最多支持 10 个独立任务程序 (Task) 在一个控制器上的执行。
- 每个程序都是独立的，存放于单独的 bas 文件中，可以单独或同时执行。
- 每个程序(Task)中可以编写代码控制另外一个程序 (Task) 运行和停止运行。
- 采用 BASIC 语言编程，是编译型语言，执行效率高。程序需要编译后才能正常执行。
- 每个程序指令由上至下依次执行，遇到跳转或循环指令，会依据相应指令到对对应行后继续执行。
- ProjectDefine.bi 类似于 C/C++ 里面的头文件，所有 Task 共用一个 ProjectDefine.bi。

程序执行顺序举 3 个例程说明。

例子 1: Task0 运行后, 程序从第 1 行指令顺序执行到第 4 行指令结束, Task0 就停止运行。

Task0

```
1  BASE 0
2  VH=5000
3  ACC=50000
4  DEC=50000
```

例子 2: Task1 运行后, 程序顺序执行到第 7 行, 就一直循环执行 7~13 行指令。Task1 在没有外部命令让 Task1 停止的情况下, Task1 会一直运行着。

Task1

```
1  BASE 0
2  VH=5000
3  ACC=50000
4  DEC=50000
5
6  'WHILE循环
7  WHILE 1
8      MOVE 1000
9      WAIT DONE
10     MOVE -1000
11     WAIT DONE
12     SLEEP 10
13 WEND
14
```

例子 3: Task2 运行后, 程序顺序执行到第 14 行后, 会执行 SUB DO_Pulse 中的程序, DO_Pulse 中的程序执行完, 会继续跳回主程序, 从第 15 行继续执行, 直到 16 行指令运行结束, Task2 停止运行。

Task2

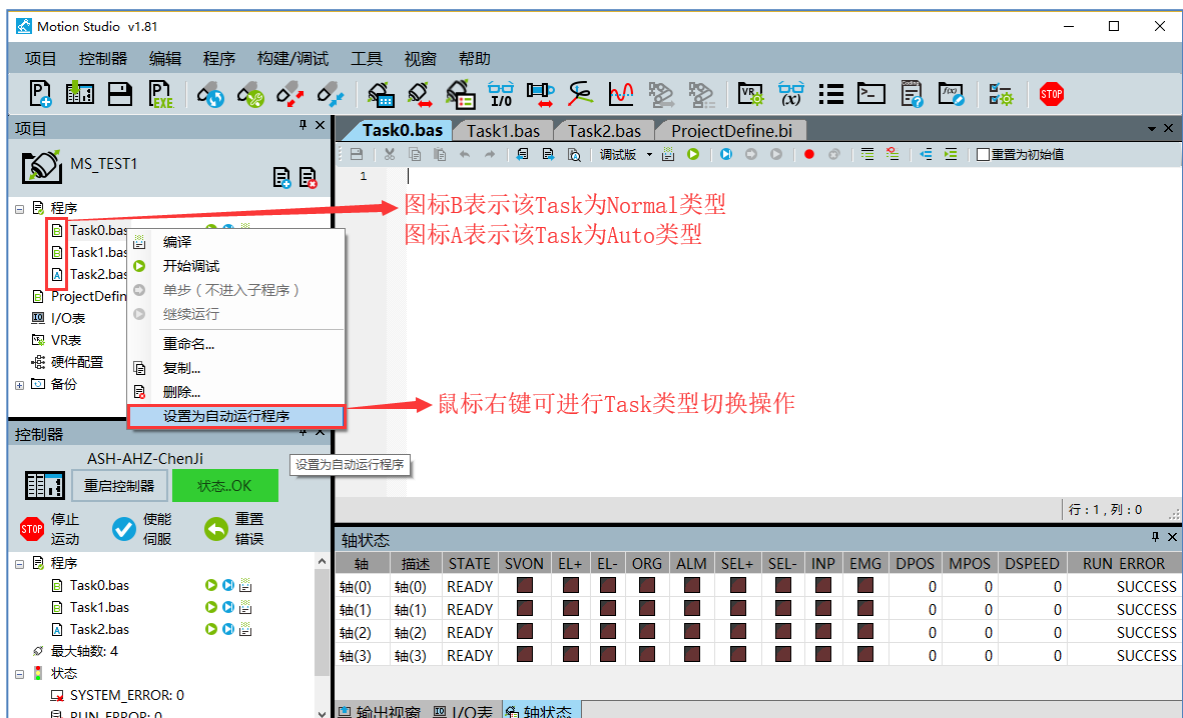
```
1  '定义1个SUB子程序: DO_Pulse
2  SUB DO_Pulse
3      DOUT(1)=1
4      SLEEP 1000
5      DOUT(1)=0
6  END SUB
7
8  BASE 0
9  VH=5000
10 ACC=50000
11 DEC=50000
12 MOVE 1000
13 WAIT DONE
14 DO_Pulse '执行SUB DO_Pulse中程序
15 MOVE -1000
16 WAIT DONE
17
```

2.4.2 一般和自动类型 Task

Motion Studio 的程序 (Task) 分为 2 种类型：一般 (Normal) 和自动 (Auto)。说明如下。

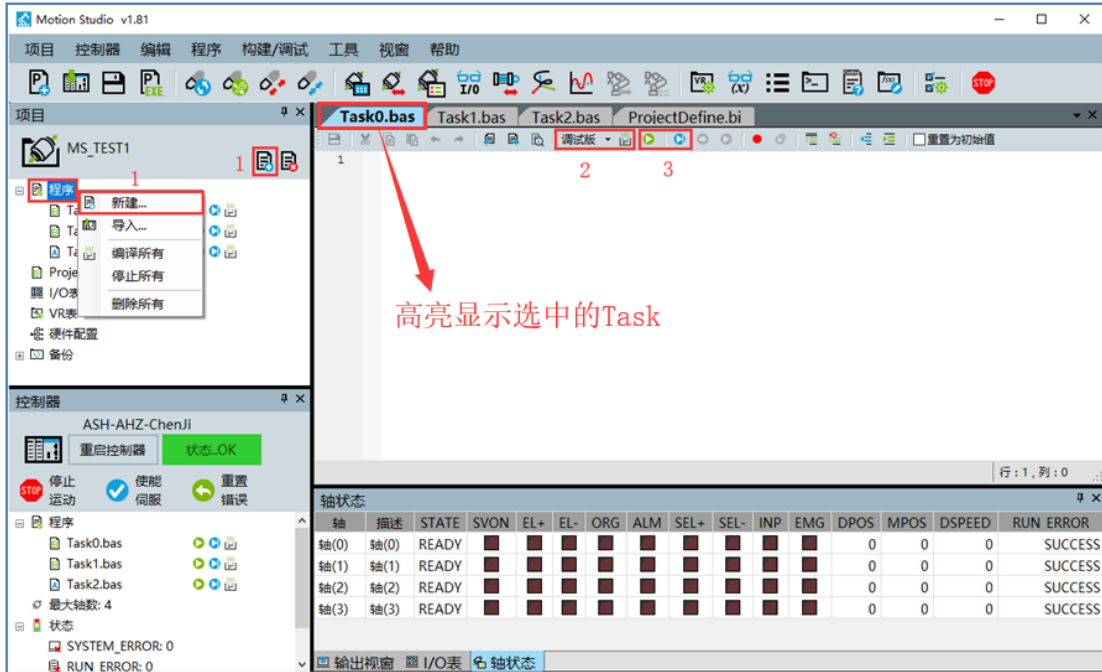
- ◆ Normal Task: 需要在 Motion Studio 手动启动运行或通过其它程序下指令启动运行。
- ◆ Auto Task: 在 Motion Runtime 启动时会自动运行一次。通常一个项目中最少会设置 1 个自动类型的程序。

两种 Task 在 Motion Studio 中呈现方式和类型切换方法如下图。



2.4.3 如何建立 Task, 编译, 执行

Motion Studio 在新建项目时会建立一个 Task, 之后使用者可以再建立 Task, Task 需要编译后才能运行。依据下面步骤可进行建立 Task, 编译, 执行。



1. 鼠标右键“程序”或点击“新建程序图标”进行新建 Task。
2. 选择“调试版”还是“发布版”进行编译。
3. 选择“调试版运行”还是“发布版运行”。点击运行后, 相应的 Task 程序在 MAS 控制器上会执行一次。

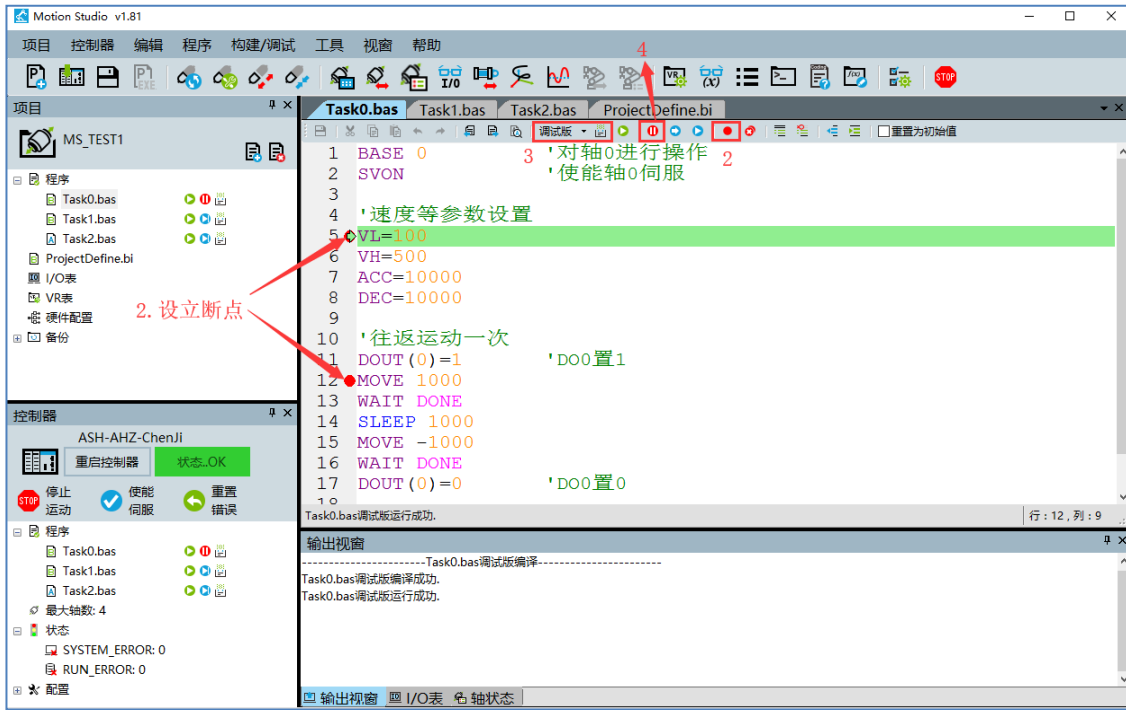
◆ 注意:

初学者常常不知如何选择“调试版”还是“发布版”编译和运行, 简单的看只要注意以下几点就可以, 其它情况随便选一种就可以。

- 断点和单步操作只在调试版运行中起作用。
- Print 指令只在调试版运行中可以将信息打印到 Motion Studio 的“输出视窗”。
- Auto 类型的 Task 需要发布版编译后才能在 Motion Runtime 重启时正常自动运行。

2.4.4 如何设立断点

调机阶段，编写程序后常需要透过断点来中断程序执行，观察相应的变量做错误排查。依据下面步骤可知如何设立断点。



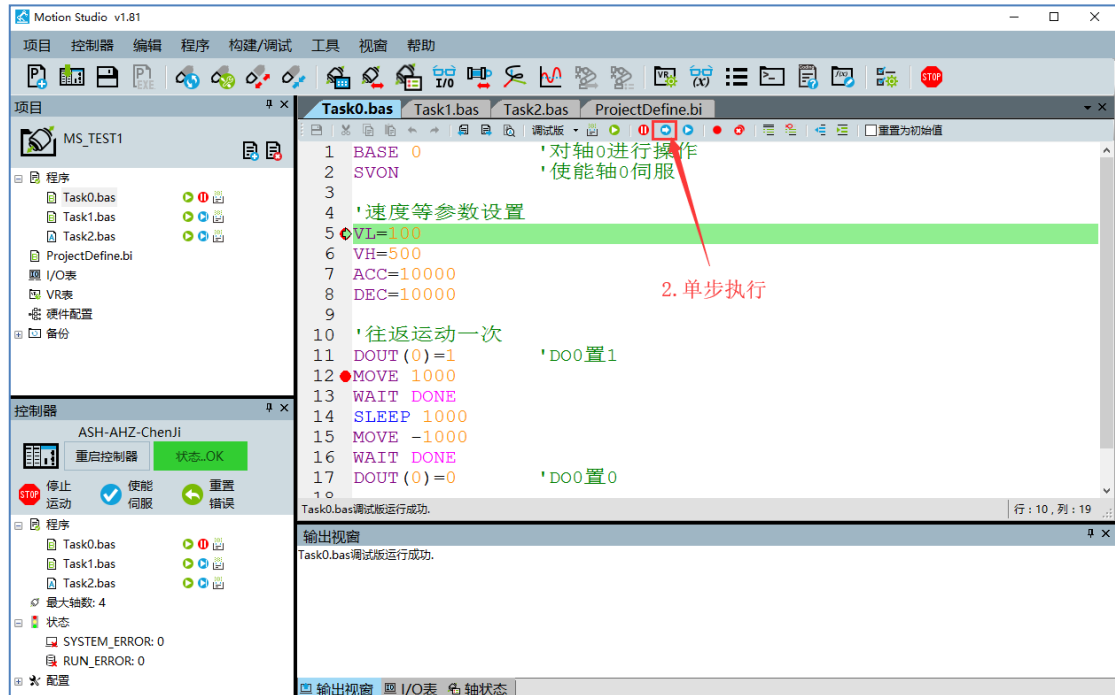
1. 编写程序。
2. 鼠标移至程序行编号区域，点击设立断点。
3. 编译
4. 调试版运行。程序将在中断行停止运行。

◆ 注意：

1. 断点不能设置在程序的第一行。

2.4.5 如何使用单步执行

在排查程序问题时，使用者可以利用单步运行来检测程序每一步带来的变化。单步执行需在程序进入断点中断后才可以进行，依据下面步骤可知如何使用单步执行。



1. 编写程序，设立断点，让程序进入断点（请参照 2.4.4 章节）
2. 按下“单步”，程序执行下一步。每按一次，程序执行一步。

2.4.6 ProjectDefine.bi 与 ProjectDefine.bas

除了全局变量之外，用户也会有一些自定义变量、方法需要同时在多个 TASK 内使用。此时，就需要使用 ProjectDefine.bi、ProjectDefine.bas 的功能。

◆ ProjectDefine.bi

可在 ProjectDefine.bi 内进行：

- a) #Define：宏定义
- b) DIM：定义变量
- c) Declare：声明子程序
- d) Type：声明类

◆ ProjectDefine.bas

在 ProjectDefine.bi 声明的子程序、类，需要在 ProjectDefine.bas 内编写实现方法。

◆ 使用示例

步骤一：在 ProjectDefine.bi 进行一些宏定义、变量与方法声明，如下图：

```

1
2 '宏定义轴号
3 #Define X 0
4 #Define Y 1
5 #Define Z 2
6
7 '宏定义VR编号
8 #Define TotalNum 100
9
10 '声明变量
11 DIM WorkPosFlag AS INTEGER
12
13 '声明子程序
14 DECLARE SUB XY_MOVEABS(X_POS AS DOUBLE,Y_POS AS DOUBLE)
15
16 '声明类
17 Type Work
18     DIM AxState AS INTEGER
19     DECLARE FUNCTION GoWorkPos() AS INTEGER
20     '其它代码
21 END Type
22
  
```

步骤二：在 ProjectDefine.bi 声明的子程序、类，在 ProjectDefine.bas 内编写实现方法，如下图：

```

2 SUB XY_MOVEABS (X_POS AS DOUBLE, Y_POS AS DOUBLE)
3   BASE 0, 1
4   MOVEABS X_POS, Y_POS
5   WAIT DONE
6 END SUB
7
8 FUNCTION Work.GoWorkPos () AS INTEGER
9   DIM Flag AS INTEGER
10  BASE 0, 1
11  MOVEABS 20, 20
12  WAIT DONE
13  IF AxState=0 THEN Flag=0 '轴异常
14  IF AxState=1 THEN Flag=1 '轴正常
15  RETURN Flag '轴正常, 认为动作到位
16 END FUNCTION

```

步骤三：完成上述操作后，便可以在任意 TASK 内使用之前定义的变量、方法，如下图：

```

2 BASE X, Y '此处判断相当于 BASE 0, 1
3 MOVE 80, 70
4 WAIT DONE
5
6 IF VR(TotalNum)=1 THEN XY_MOVEABS (10, 10)
7 '此处判断相当于 IF VR(100)=1 THEN ''''
8
9 DIM MyWork as Work '实例化Work类
10 WorkPosFlag=MyWork.GoWorkPos ()
11 '调用方法去工作点, 并返回工作点标志

```

◆ 注意

在 ProjectDefine.bi 声明的变量，虽然可以在所有的 TASK 内使用，但本质上并不是同一个变量，只是名称相同而已。所以，在 ProjectDefine.bi 声明的变量，并不是全局变量。

若要在多个 TASK 内使用同一个变量，请使用 VR、Table 全局变量。

第三章 变量使用

3.1 变量种类

MAS 控制器的变量可以归类为 3 种，如下做简单说明。

- ◆ 系统变量： 控制器内建并具备特定功能属性的变量。如下表这些是系统变量。

变量	说明
VH	轴的运行速度
ACC	轴的加速度
UNIT_NUM	单位脉冲量分子

- ◆ 用户自定义变量： 用户根据编程需要在 Task 里自定义的 Task 局部变量。
- ◆ VR 变量： 控制器内建的全局变量，各 Task 间通用，无需用户再次定义。
- ◆ Table 变量： 与 UI 界面的 Table 控件结合使用，用于工单参数管理，

- ◆ **注意：**

系统变量本章不再详细说明，请参考 BASIC 使用手册。用户自定义变量和 VR 变量用法在下面章节介绍。

3.2 用户自定义变量

用户自定义变量的主要说明如下。

1. Task 里使用 Dim...As...进行声明
2. 是 Task 的局部变量，不是所有 Task 共同的全局变量。各 Task 间同名的自定义变量不是同一个变量。需要所有 Task 共同的全局变量请使用 VR 变量。
3. 在 ProjectDefine.bi 中声明自定义变量不能作为所有 Task 的全局变量
4. 自定义变量使用时要注意变量类型，类型不对会导致变量值丢失精度或不正确。
5. 自定义变量的类型和类型转换说明请参考 BASIC 使用手册的“数据类型及转换章节”。

下面介绍几种自定义变量的用法。

3.2.1 如何声明变量和赋值

- 声明 1 个变量并赋值

```
DIM a AS DOUBLE '声明1个名为a的64位浮点型变量
a=15.3          'a赋值15.3
```

```
DIM text AS STRING ="Hi,MAS" '声明1个名为text的字符串,并赋值为HI,MAS
```

- 1 行声明多个变量并赋值

```
DIM AS ULONG b,c '定义b、c两个变量,数据类型为无符号长整型
b=4             'b赋值4
c=8             'c赋值8
```

3.2.2 如何声明数组和赋值

- 数组声明和赋值说明

```
'定义1个长度为3,名为Array的BYTE类型数组并赋值。
'Array(0)=1; Array(1)=2; Array(2)=5
DIM Array(2) AS BYTE={1,2,5}
```

- 指定数组区间声明并赋值

```
DIM Array(2 to 5) AS BYTE={10,11,4,5}
PRINT array(2)      '打印出为10
PRINT array(3)      '打印出为11
PRINT array(4)      '打印出为4
PRINT array(5)      '打印出为5
```

3.2.3 如何声明在子程序中使用的变量

- 变量声明在子程序中

```
SUB test
  dim i as INTEGER
  for i=0 to 3
    DOUT(i)=1
  next i
END SUB
```

- 变量声明在子程序外部。需要加“Shared”指令。

```
DIM Shared a AS INTEGER=0

SUB test
  a=a+1
END SUB
```

SUB test中的变量a在SUB外部定义，需使用Shared

◆ **注意:**

自定义变量在 SUB 和 FUNCTION 中的使用情况是一样的。

3.3 VR 变量

3.3.1 VR 变量介绍

VR 变量的主要说明如下。

1. 控制器内建的全局变量，各 Task 间通用，无需用户再次定义
2. 共 10000 个：VR(0)-VR(9999)。数据类型为 Double，可读可写。
3. Motion Studio 中可以利用 VR 表工具进行管理，VR 表工具的主要用途如下。
 - a) 批量分区块管理 VR 变量。
 - b) 将 VR 添加到 VR 表工具中进行监测。
 - c) 对每个 VR 变量进行描述。
 - d) 初始值保存。初始值在电脑重启后不会丢失，可以使用 VRReset 指令将初始值赋给当前值。
 - e) 对 VR 变量建 Modbus 地址。建了 Modbus 地址的 VR 可用于与控制器外部通信。

3.3.2 VR 变量的基本使用

VR 变量可读可写，属于全局变量，可在任意 TASK 内使用。如下例子说明基本的赋值传递。

- 赋值给 VR 变量

```
VR (0) = 100.5      '直接赋值
DIM A AS LONG = -5
VR (1) = A         '通过变量赋值
```

- VR 变量值赋值给其它变量

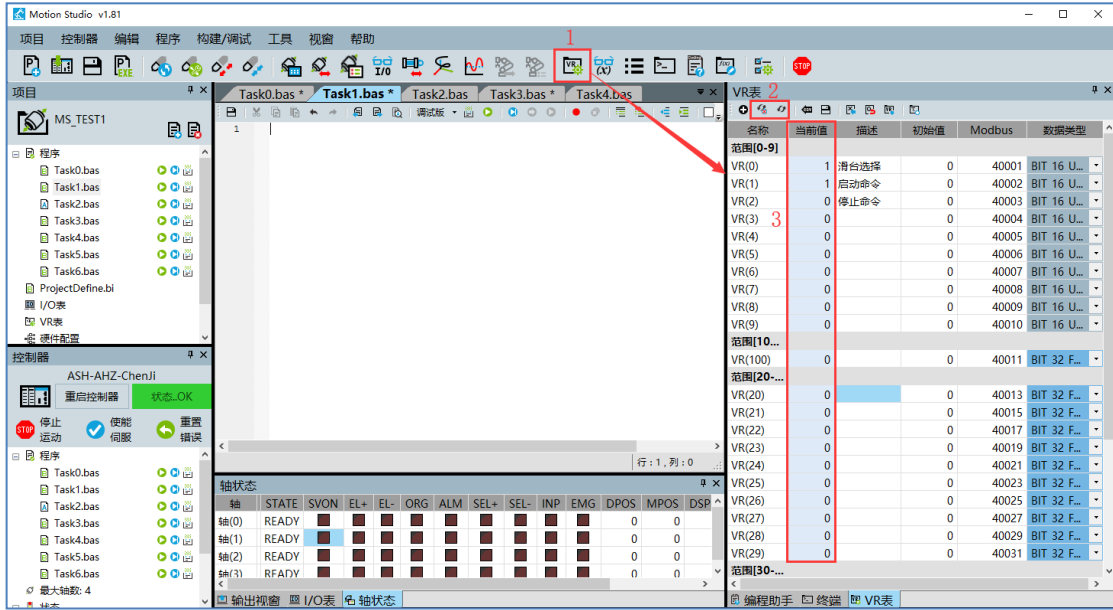
```
DIM A AS LONG = 0
A = VR (0)      '将VR (0) 的值赋给A
```

- 运动指令中使用 VR 数据

```
BASE 0, 1
MOVEABS VR (0), VR (1)
```

3.3.3 如何监测 VR 变量

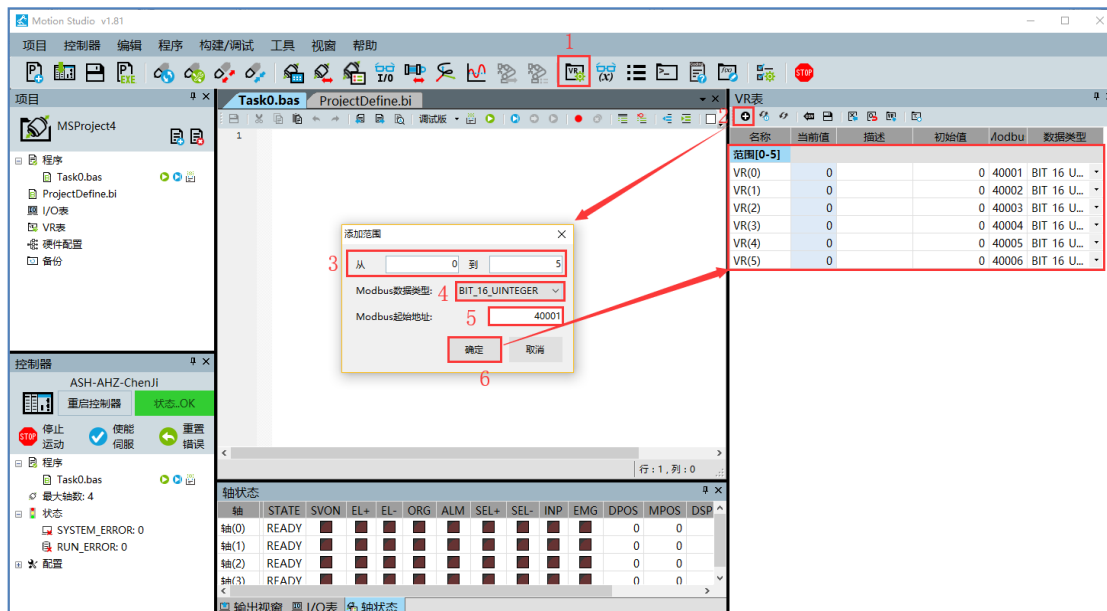
依照下面步骤即可实现 VR 表中批量监测 VR 变量的值。



1. 打开 VR 表工具，添加要管理的 VR 变量。
2. 选择刷新显示：实时刷新还是手动刷新。
3. 进行 VR 变量监测。
 - a) 显示 VR 变量值
 - b) 直接“当前值”列的表格里修改 VR 值

3.3.4 如何建 VR 的 Modbus 地址

控制器与外部进行数据交互是通过 VR 变量来实现的。VR 变量需先分配 Modbus 地址才可与外部进行数据交互。依据下面步骤即可完成 VR 变量的 Modbus 地址分配。



1. 打开 VR 表工具
2. 在 VR 表中，点击“+”添加 VR 范围。
3. 选择要添加到 VR 表中的 VR 区间，如图是 VR(0)~VR(5)
4. 选择要添加的 VR 变量对应的 Modbus 数据类型。
5. 填写起始 VR 变量对应的 Modbus 起始地址。
6. 确定添加。如图，VR 表中添加的 VR 变量对应的 Modbus 地址分配完成。

◆ 注意：

1. VR 变量不是一次就要全部添加到 VR 表中并进行 Modbus 地址分配，可分多次进行。
2. VR 变量可分配给 Modbus 通信使用的数据类型与 Modbus 地址的对应关系如下说明。

VR 变量可分配给 Modbus 通信使用的数据类型分为 16 位和 32 位。16 位的 VR 变量对应 1 个 Modbus 地址，32 位 VR 变量对应 2 个 Modbus 地址，如下图示例：

VR表					
名称	当前值	描述	初始值	Modbus	数据类型
范围[0-3]					
VR(0)	0		0	40001	BIT 16 INT
VR(1)	0		0	40002	BIT 16 INT
VR(2)	0		0	40003	BIT 16 INT
VR(3)	0		0	40004	BIT 16 INT
范围[4-7]					
VR(4)	0		0	40005	BIT 32 FLOAT
VR(5)	0		0	40007	BIT 32 FLOAT
VR(6)	0		0	40009	BIT 32 FLOAT
VR(7)	0		0	40011	BIT 32 FLOAT

上图 VR 表格中，VR(0)-VR(3)分配的数据类型为 16 位有符号整型，其对应的 Modbus 地址为 40001-40004，地址间隔为 1；而 VR(4)-VR(7)分配的数据类型为 32 位浮点型，其对应的 Modbus 地址为 40005-40011，地址间隔为 2。

3.3.5 如何批量存取 VR 数据

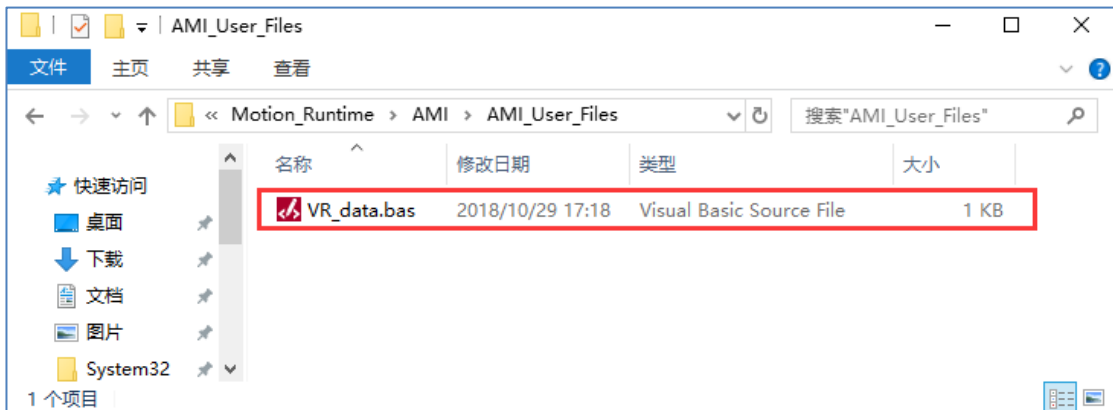
可通过 FILE_WRITEVR 和 FILE_READVR 指令对 VR 表进行保存和读取。

- 使用 FILE_WRITEVR 指令保存 VR 数据示例

```
'将VR(0)~VR(20)的数据写到本地名为VR_data的bas文件
FILE_WRITEVR "VR_data.bas", 0, 20
```

执行上面程序后，存有 VR(0)~VR(20)的表格数据会存在 VR_data.bas 文件中，路径如下：

Motion Studio 安装目录下\Advantech\Motion_Runtime\AMI\AMI_User_Files



- 使用 FILE_READVR 读取保存的 VR 数据示例

```
'将存放VR文件路径下的"VR_data.bas"文件读出并更新VR表的值
FILE_READVR "VR_data.bas"
```

执行上面程序后，会将本地名为 VR_data.bas 文件内的 VR 数据读到控制器里，这些数据将覆盖控制器里对应的 VR 数据

◆ 注意：

1. FILE_WRITEVR 和 FILE_READVR 这两个指令涉及到本地文件的读和写。程序里不能用高频率去使循环使用。比如在无限循环体中使用 FILE_WRITEVR，那会造成一直高频率对硬盘写数据。一定时间后，会对硬盘造成伤害。

3.4 Table 变量

3.4.1 Table 变量介绍

在 Motion Studio 中，Table 变量的基本属性类似于 VR 变量。

Table 变量的特性如下：

1. 用户可用范围：Table (0)- Table (399999)，共 400000 个。
2. 可通过建立 Table 表，以表格形式管理、监测 Table 变量。
3. Table 变量不可分配 ModBus 地址，可通过 MS HMI.NET 中的 MSDataTable 控件，在 UI 界面上对其进行批量的值操作。
4. Motion Studio 提供 Tab 类方法，可让用户对 Table 表进行便捷的数值操作。

◆ Table 变量的基本取值与赋值

该操作类似于 VR 变量，示例如下：

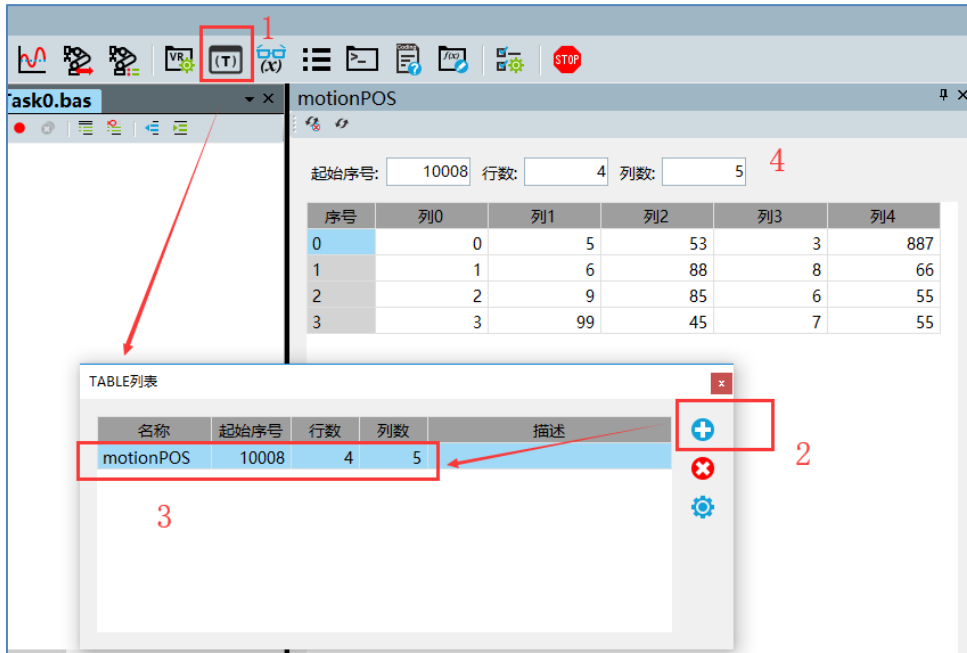
<code>TABLE (0) =1</code>	'将变量TABLE (0) 赋值为1
<code>VR (2) =TABLE (0)</code>	'将变量TABLE (0) 的值赋给VR (2)

执行程序后，VR(2)的值变为 1，结果如下：

范围[0-10]	
VR(0)	0
VR(1)	0
VR(2)	1
VR(3)	0

3.4.2 使用 Table 表监测 Table 变量

对 Table 变量的管理、监测，需要建立 Table 表，依照下面步骤：



如上图：

1. 双击工具栏的 Table 图标，打开 Table 列表工具。
2. 手动添加 Table 表。
3. 双击打开某个 Table 表。
4. 编辑当前 Table 表的起始序号（地址）、行数、列数。

上图中，起始地址设为 Table (10008)，表格 4 行 5 列。

如若表格内，监测显示的内容如上图，则可知对应的 Table()变量值如下：

Table (10008)=0

Table (10009)=5

Table (10010)=53

.....

Table (10026)=7

Table (10027)=55

上述操作后，即可监测 Table 表格内对应 Table 变量。

3.4.3 使用 Tab 方法操作 Table 表

通过 Tab 类, 可对 Table 表内的数据进行一维、二维的数据操作, 步骤如下:

1. 实例化一个 Table 表
2. 对 Table 表的行、列, 进行取值、赋值操作

为更直观地对 Tab 方法进行讲解, 以下面 Table 表为例:

起始序号:	<input type="text" value="10008"/>	行数:	<input type="text" value="5"/>	列数:	<input type="text" value="5"/>
序号	列0	列1	列2	列3	列4
0	0	5	53	3	887
1	1	6	88	8	66
2	2	9	85	6	55
3	3	99	45	7	55
4	4	88	23	66	81

步骤一: 实例化一个 Table 表

实例化为一个 Tab 类型的表格, 名称为 Para1, 如下:

```
DIM AS Tab Para1
Para1=Tab(10008,5,5)
```

上述代码, 首先实例化一个名称为 Para1 的 Table 表。

然后, 从 Table(10008)开始, 取 5*5 共 25 个 Table 变量, 映射为 5 行 5 列的 Para1 表格。

则可知, Para1 表的内容与上面的 Table 表内容一样, Para1 表格内 Table 变量的地址为 Table (10008)至 Table (10032)。

步骤二：使用 Tab 类方法，操作 Table 表内容

◆ Table 表的单个取值、赋值

仍以上述实例化的 Para1 为例，获取 Para1 表格中第一行第二列的数值，将 Para1 表格中第三行第二列的值改成 23，示例如下：

```
DIM A AS DOUBLE  
A=Para1.getvalue(1,2)  
Para1.setvalue(3,2,23)
```

上例中，使用 Tab 类中的 getvalue 方法，获取 Para1 表格中第一行第二列的数值。执行程序后，A 的值是 88。

使用 Tab 类中的 getvalue 方法，将 Para1 表格中第三行第二列的值改成 23。原第三行第二列的值是 45，执行程序后，该值变成 23。

◆ 从 Table 表取出位置点

起始序号:	<input type="text" value="10008"/>	行数:	<input type="text" value="5"/>	列数:	<input type="text" value="5"/>
序号	列0	列1	列2	列3	列4
0	0	5	53	3	887
1	1	6	88	8	66
2	2	9	85	6	55
3	3	99	45	7	55
4	4	88	23	66	81

仍以先前实例化的 Para1 表格为例，取出上表中的 (6, 88, 8) 这个点位，做一个三轴的插补运动。代码如下：

```
Para1.MAPPoint(1,3)
'从实例化的Para1表格里，从列1开始，取出3列
PickPlace=Para1.Point(1)
'从取出的列中，取出行1，赋值给PickPlace
BASE 0,1,2
LINEABS PickPlace
WAIT DONE
```

上例中，使用到了 Tab 类中的 MAPPoint 方法与 Point 方法

1. MAPPoint (m,n)：从第 m 列开始，取出共 n 列，映射为一个新的表格，新表格的行数等于原表格的行数。

则可知上例中执行 Para1.MAPPoint(1,3)后，获得的新表格内容如下：

列1	列2	列3
6	53	3
6	88	8
9	85	6
99	45	7
88	23	66

2. Point (r) 方法：对应“新表格”中第 r 行内容。放入 Point(r)中。

◆ 说明：

上例中，Point(r)本质上是一个 P 点数据类型，有关 P 点，详见 6.5.5 章节。Tab 类方法也可结合 P 点进行数据操作，详见 6.5.5 章节。

第四章 输入输出控制

4.1 DIO 控制

此章节介绍的 DIO 控制方法，适用于运动控制卡、IO 卡上的 DIO 信号，如下：

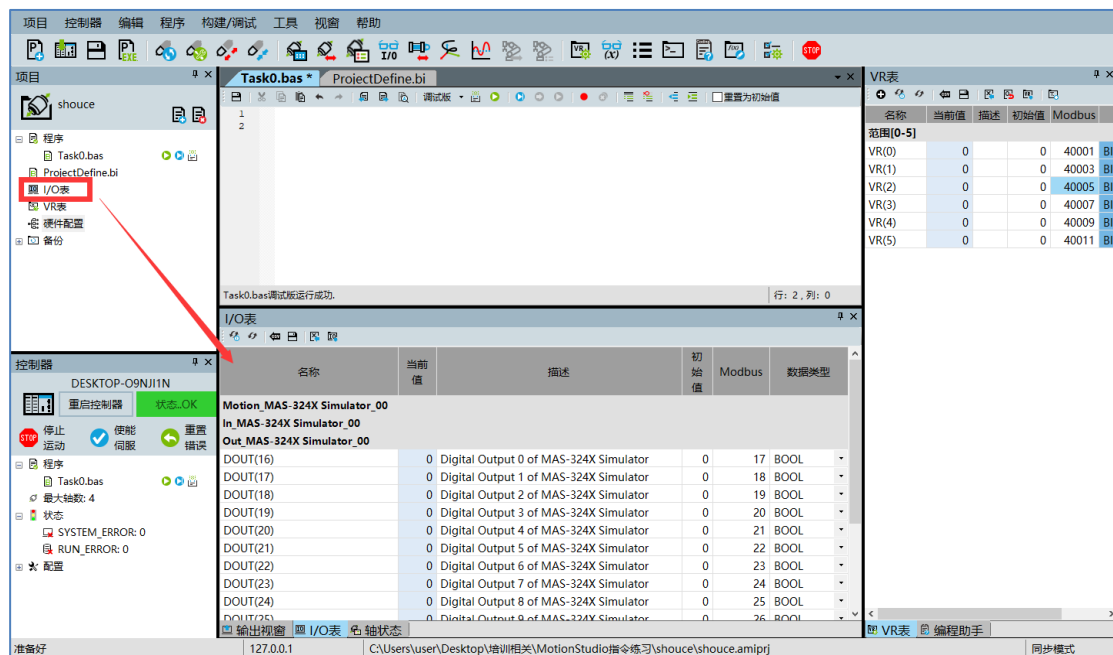
- 1) MAS 运动控制卡
- 2) 普通 IO 卡: PCI-1750、PCI-1756 PCIe-1730、PCIe-1756

并不适用于其它卡片，如模拟量卡上的 DIO，有关模拟量卡上的 DIO 控制，请参见 4.2 章节。

4.1.1 如何查看系统 IO 数量和分配

研华 MAS 运动控制卡的 IO 信号分为：运动控制专用 IO，普通 IO。

◆ 查看系统 IO 数量和分配



如上图：

双击 I/O 表，弹出 I/O 表窗口，点击相应分类可查看相应 IO 数量及功能分配。

4.1.2 开放给用户使用的运动控制 IO

在研华运动控制器提供的运动控制 IO 中：

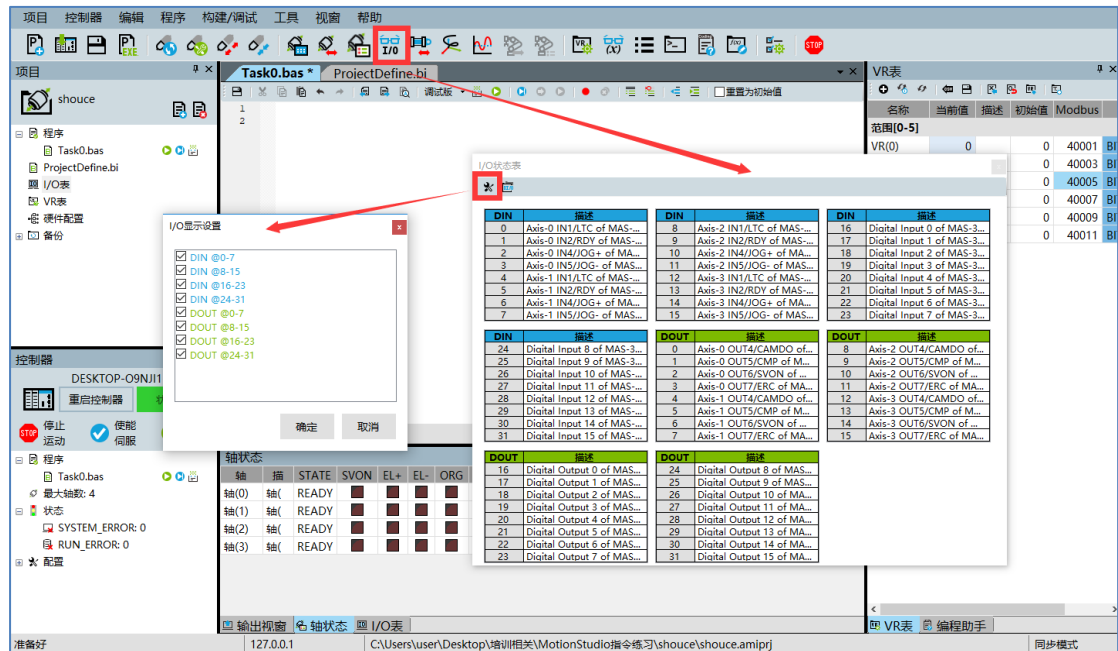
1. 有些 IO 点固定使用，用户不能用作其它用途，例如 LMT(硬极限信号输入)、ORG(原点感应信号输入)、脉冲输出信号等。
2. 有些 IO 点可以功能多用，开放给用户，用户可根据实际情况选择相应的功能使用。

◆ 开放给用户使用的运动控制 IO 一览

下表中 IO 适用于每个轴。

IN1	默认为一般输入，供用户使用 此输入点还可做高速位置锁存用，详见 6.9 高速位置锁存章节。
IN2	默认为一般输入，供用户使用。 如需接入伺服准备完成信号，建议使用该输入点。
IN4 IN5	默认为一般输入，供用户使用。 这两个输入点还可用于：外部输入控制轴 JOG+/ JOG-运动，此时需执行 JOG ON 指令启用该功能。
OUT4	默认为一般输入，供用户使用。 此输出点还可用作 CAM-DO(范围比较触发功能)，需通过硬件配置，或者执行 CAMDO_EN=1 指令启用该功能。
OUT5	默认为一般输出，供用户使用。 此输出点还可用作 CMP(高速比较触发)功能，需通过硬件配置，或者执行 CMP_EN=1 指令启用该功能，详见 5.8 章节。
OUT6	默认为一般输出，供用户使用。 如需外接伺服使能，建议使用该输出点。
OUT7	默认为一般输出，供用户使用。 如需外接伺服的偏差计数清除功能，建议使用该输出点。

4.1.3 如何监测 DIO 状态



如上图：

1. 双击功能菜单中的“I/O 监测”快捷功能，打开 IO 状态表，
2. 在 IO 状态表中，点击 IO 配置工具，弹出 I/O 显示设置
3. 在 I/O 显示设置窗口中，勾选要监测的 IO，点击确定后，即可对 IO 进行监测。

◆ 注意

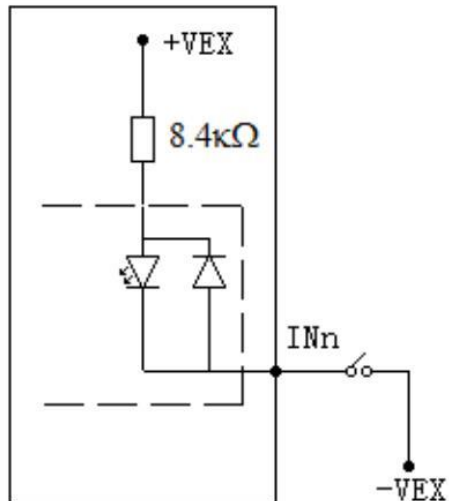
在 IO 状态表中，点击相应的输出点，可强制输出状态转换。

4.1.4 DIO 硬件接线

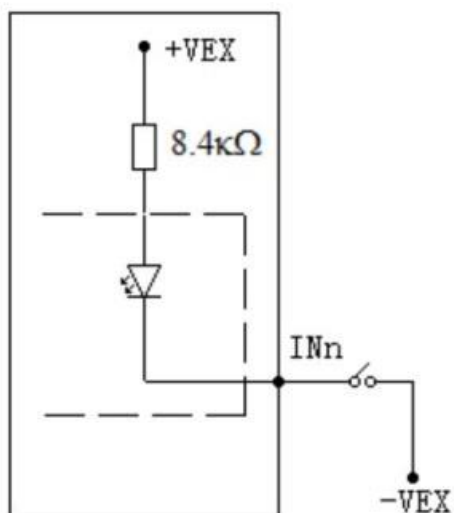
◆ DI 输入点接线

DI 触发机制：输入点的有效电平是低电平，当输入点接入低电平时，控制器内部的发光二极管两端产生电压差，发光二极管导通产生光源信号，控制器接收到光源信号后，令输入信号为 ON。具体见下图的讲解。

一般的数字量输入，接线如下图：



高速数字量输入，接线如下图：

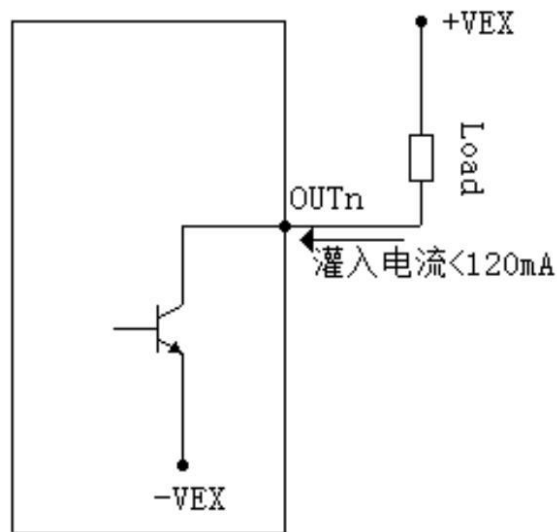


◆ DO 输出点接线

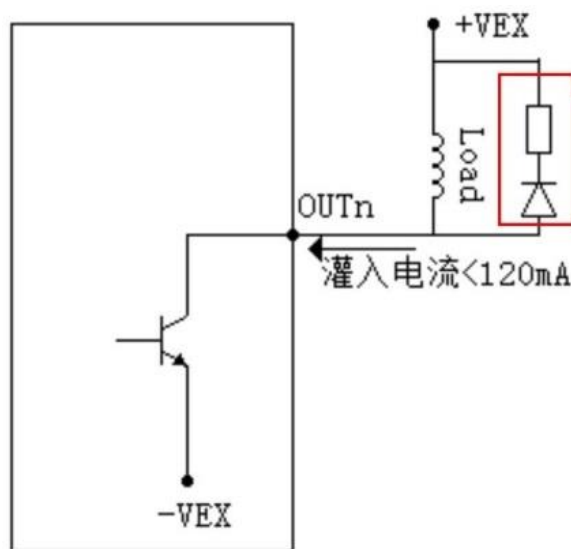
DO 输出时，输出点会与控制器内部的“-VEX (24V 低电平)”导通，即此时输出点的电平为低电平。

输出接负载时，负载的一端与输出点连接，一端要与“+VEX (24 高电平)”连接，具体的方法见下面的讲解。

输出点接一般负载时，接线图如下：



输出点接感性负载时，建议增加感性能量释放的回路，如下图：



DO 输出点接入负载时，其最大灌入电流不得超过 120mA。

4.1.5 如何编写程序控制 DIO

使用 DIN()读取输入状态；使用 DOUT()控制输出状态，DOUT()的输出控制具有锁存性质，执行一次即可令输出状态保持。

◆ IO 输入读取示例

读取输入点 1 的状态并将其赋值给 VR(0)，如下：

```
VR(0)=DIN(1)
```

执行上面的程序的后，若 DIN(1)的为 ON，则 VR(0)值为 1；若 DIN(1)的为 OFF，则 VR(0)值为 0。

◆ IO 输出控制示例

让输出点 2 为 ON，保持 5 秒，5 秒后让输出为 OFF，如下：

```
DOUT(2)=1
```

```
SLEEP 5000
```

```
DOUT(2)=0
```

因 DOUT()的输出控制具有锁存性质,所以只需执行一次指令，便可让状态保持。

◆ 注意：

由于 IO 的输出具有锁存性质，因此不需要一直让输出条件满足。举例说明，让输出 1 保持为 ON，程序如下：

```
DOUT(1)=1
```

有些用户可能会有下面的错误的用法，需要注意：

```
WHILE 1
```

```
    DOUT(1)=1
```

```
WEND
```

4.2 AIO 控制

用 Motion Studio 进行 AIO 控制，主要针对研华的模拟量卡，步骤如下：

1. 安装 DAQ 软件与驱动，用于识别、配置、测试模拟量卡。
2. 模拟量卡的 AIO 编程控制。

同时，因模拟量卡上也会有 DIO，所以本章节也会涉及模拟量卡上的 DIO 控制方法。

4.2.1 识别与配置模拟量卡

Motion Studio 要控制模拟量卡上的 AIO，另外要安装如下软件：

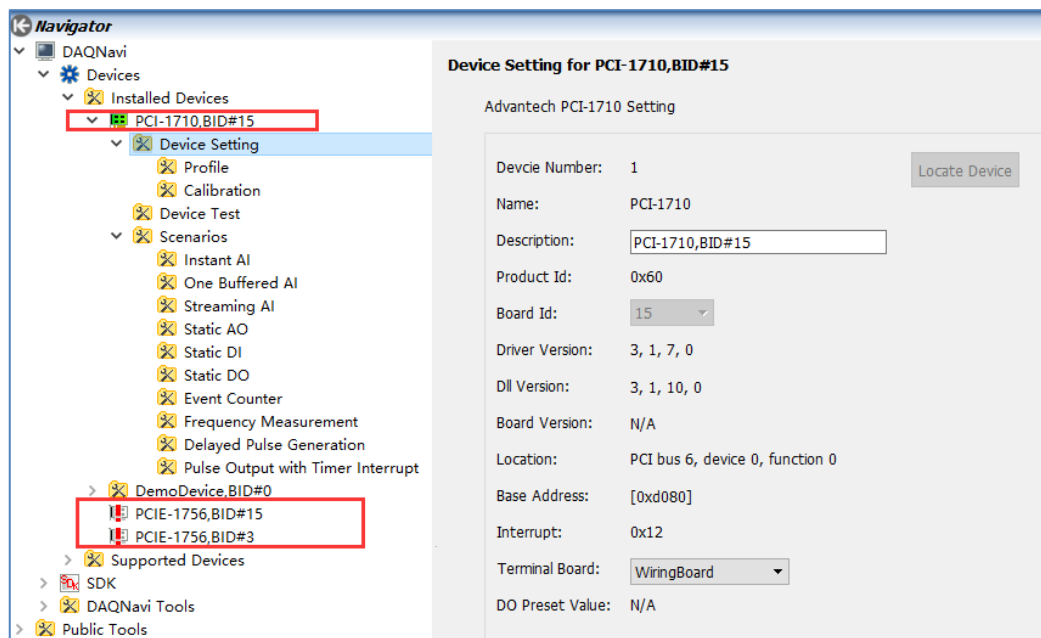
- 模拟量卡对应的驱动：用于识别模拟量卡
- DAQNavi_SDK：模拟量卡调试与配置工具

以上软件可至研华的官网 (<http://www.advantech.com.cn>) 下载。

步骤一：模拟量卡的识别

安装 DAQNavi_SDK 软件后，会有一个 Navigator 工具，此工具可用于对模拟量卡进行基本的配置与调试。

打开 Navigator 软件，展开 Devices 项可查看当前识别到的控制卡，如下图所示：

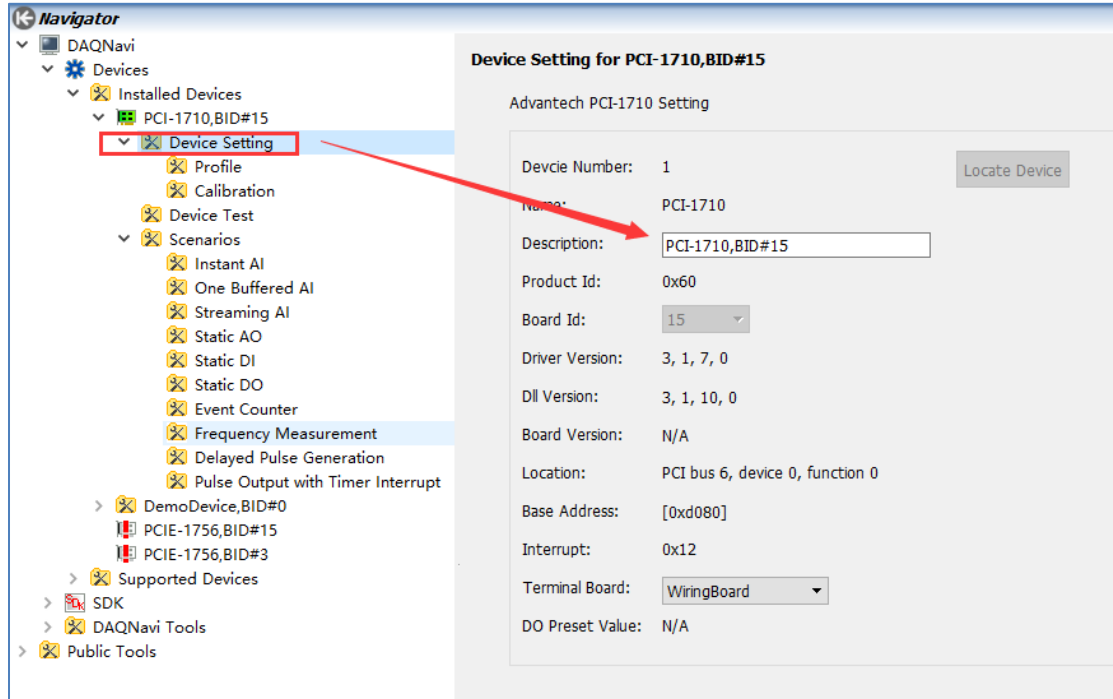


◆ 注意：

若无法识别模拟量卡，请确认驱动是否正确安装。

步骤二：使用 Navigator 软件配置模拟量卡

如下图，在 Navigator 工具中，选中相应模拟量卡下面的 Device Setting 项，可查看与修改模拟量卡的基本配置，比如模拟量卡的名称描述、输入信号的种类配置、单端输入与差分输入的选择等。

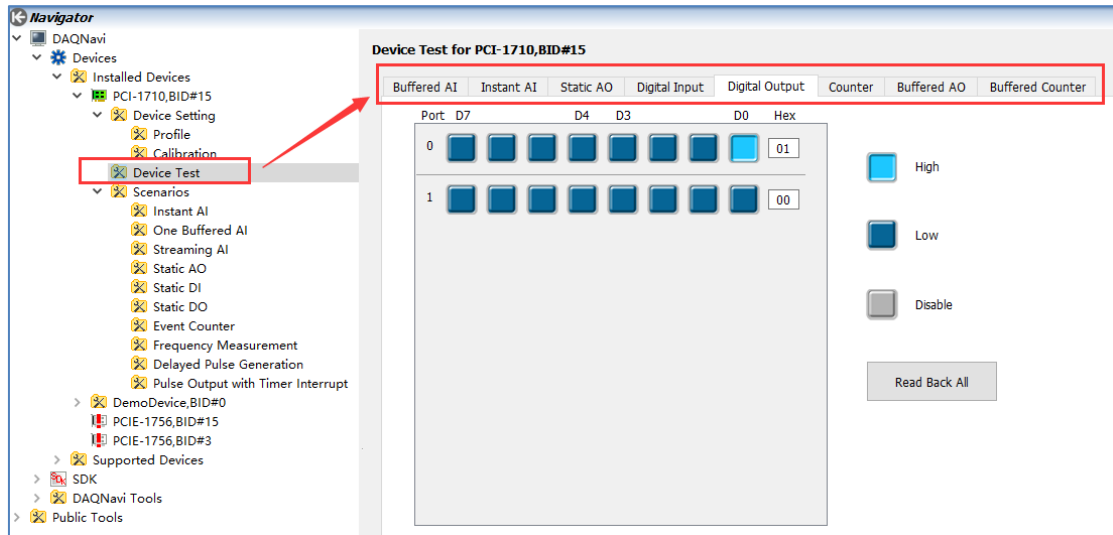
**◆ 注意：**

Description 为模拟量卡的名称配置，在 Motion Studio 中编程控制模拟量卡时，代码中的模拟量卡名称需与此名称保持一致。

步骤三：使用 Navigator 软件查看、监测、测试模拟量卡

如下图，在 Navigator 工具中，选中相应模拟量卡下面的 Device Test 项，可进行如下操作：

- 查看、监测模拟量卡的输入输出分配、当前状态。
- 进行基本的手动调试工作，以确认模拟量卡可以正常运行。



4.2.2 编程控制模拟量卡

根据 4.2.1 章节的操作，通过 Navigator 工具确认模拟量卡可以被识别、并完成配置后，即可通过 Motion Studio 编写程序对其进行控制。

◆ **注意：**

在运行 Motion Studio 程序时，请退出 Navigator 及其它 DAQNavi_SDK 所安装的软件运行，否则在控制模拟量卡时会冲突。

编写模拟量卡的程序，步骤如下：

- 1) 引用库文件
- 2) 实例化模拟量卡控制对象
- 3) 初始化模拟量卡
- 4) 模拟量卡控制，包含 DI 与 DO 控制、AI 与 AO 控制
- 5) 模拟量卡资源释放

接下来，依上述步骤进行讲解。

步骤一： 引用库文件

```
#include once "ExPCIBoard.bi"
```

步骤二： 实例化模拟量卡控制对象

代码示例如下：

```
DIM VOL AS DAQ
```

代码解析：实例化一个名称为 VOL 的对象，该名称用户可自行修改，类型为 DAQ。

接下来的讲解，均按照 VOL 对象名称进行。

步骤三： 初始化模拟量卡

代码示例如下：

```
DIM AS WSTRING * 64 DEVICENAME => "PCI-1710,BID#15"  
VOL.INIT 0,DEVICENAME
```

代码解析：将名称为"PCI-1710,BID#15"的模拟量卡与 VOL 对象匹配，并执行初始化；接下来，就可以使用 VOL 对象控制模拟量卡。

此处名称要与模拟量卡的识别名称保持一致，识别名称详见 4.2.1 章节的步骤二。

◆ 说明

以上代码，用户只需要根据实际应用，修改对象的名称（本例中为 VOL）和模拟量卡名称（本例中为"PCI-1710,BID#15"）即可，无需对代码的其它部分进行改动。

步骤四：模拟量卡上的 DI 与 DO 控制

仍以名称为 VOL 的对象为例，进行讲解，n 为通道索引号。

- 读取输入 DI 的状态，方法为 VOL.ReadDI(n)。
- 控制 DO 的输出为 OFF，方法为 VOL.WriteDO(n,0) 。
- 控制 DO 的输出为 ON，方法为 VOL.WriteDO(n,1)。
- 读取 DO 输入状态，方法为 VOL.ReadDO(n)。

◆ 程序示例

获取 DI(1)的状态，如果为真，则令 DO(0)为 OFF；如果为假，则令 DO(0)为 ON，代码如下：

```
IF VOL.ReadDI(1)=0 THEN
    VOL.WriteDO(0,1)
ELSE
    VOL.WriteDO(0,0)
END IF
```

步骤五：AI 与 AO 控制

仍以名称为 VOL 的对象为例，进行讲解，n 为通道索引号。

- 读取 AI 的值，方法为 VOL.ReadAI(n)。
- 控制 AO 的输出，方法为 VOL.WriteAO(n,Value) ，即控制通道 n 输出电压值 Value。

■ 程序示例

获取 AI(0)和 AI(4)通道的数值，并在输出视窗显示；控制通道 AO(0)、AO(1)分别输出 4V、3V 的电压值。代码如下：

```
PRINT "AI0:";VOL.ReadAI(0)
PRINT "AI4:";VOL.ReadAI(4)
VOL.WriteAO(0,4)
VOL.WriteAO(1,3)
```

步骤六：模拟量卡资源释放

在程序运行结束前，要对模拟量卡进行资源的释放。

仍以名称为 VOL 的对象为例，资源的释放方法如下：

```
VOL.DESTY 'DAQ卡资源释放'
```

4.2.2 模拟量卡控制代码示例

整合前章节示例，所有代码如下：

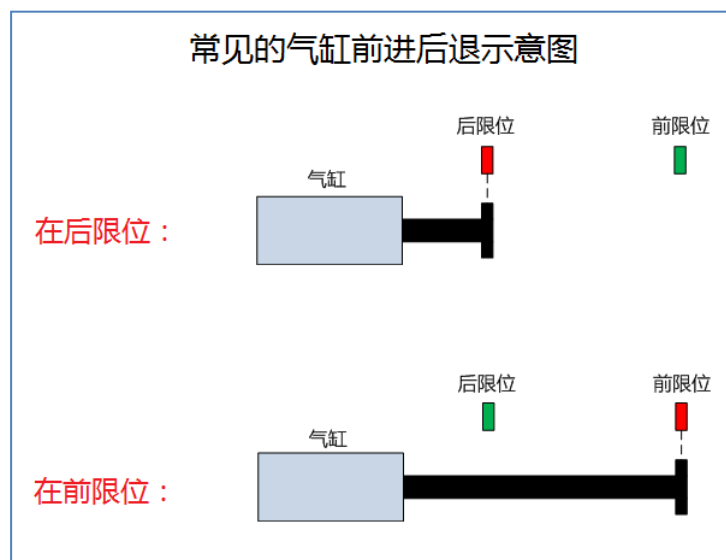
```
#include once "ExPCIBoard.bi"
DIM VOL AS DAQ
DIM AS WSTRING * 64 DEVICENAME => "PCI-1710,BID#15"
VOL.INIT 0,DEVICENAME
VOL.WriteAO(0,4)
VOL.WriteAO(1,3)
WHILE 1
    IF VOL.ReadDI(1)=0 THEN
        VOL.WriteDO(0,1)
    ELSE
        VOL.WriteDO(0,0)
    END IF
    PRINT "AI0:";VOL.ReadAI(0)
    PRINT "AI4:";VOL.ReadAI(4)
    SLEEP 100
WEND
VOL.DESTY
```

第五章 气缸控制

实际的设备应用中, 气缸控制是经常遇到的 (油缸的控制跟气缸一样, 本章仅介绍气缸的控制)。控制器一般通过 DO 控制电磁阀, 然后由电磁阀换向引导气动来使气缸动作。根据是几位几通的电磁阀, 简单可将电磁阀分为单线圈 (1 个 DO 控制) 和双线圈 (2 个 DO 控制), 再加上气缸常常需要利用前后限位来确定是否动作到位。所以气缸控制对控制器来看, 可以看成是 DI/O 的控制。

Motion Studio 中可以根据实际应用, 快速配置每个气缸的控制端口, 到位方式, 报警条件等, 然后再通过简单的指令即可实现气缸的顺序动作控制, 大大简化了高级语言实现气缸控制的难度。

下面几个小章节将介绍如何配置, 测试以及编程实现气缸控制。



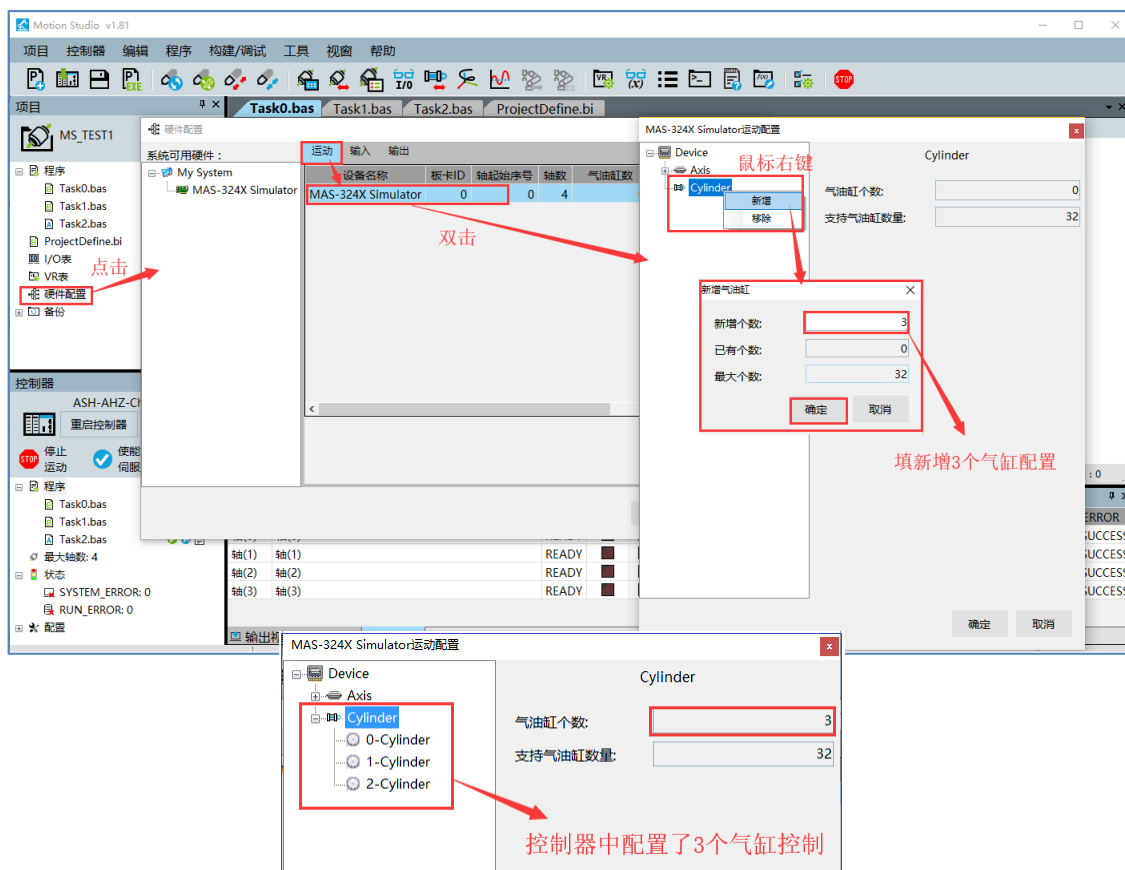
5.1 如何配置气缸

使用 Motion Studio 的气缸控制指令，需先进行配置。如下示例，说明如何配置气缸的控制。

◆ 配置气缸示例

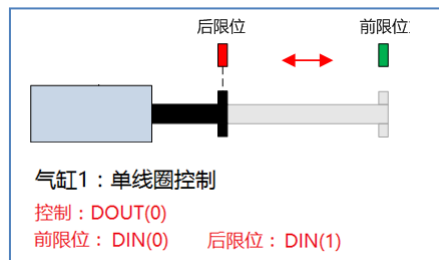
设备中有 3 个气缸要控制。依据下面步骤即可完成 3 个气缸的控制配置。

步骤一：硬件配置中添加 3 个气缸控制。

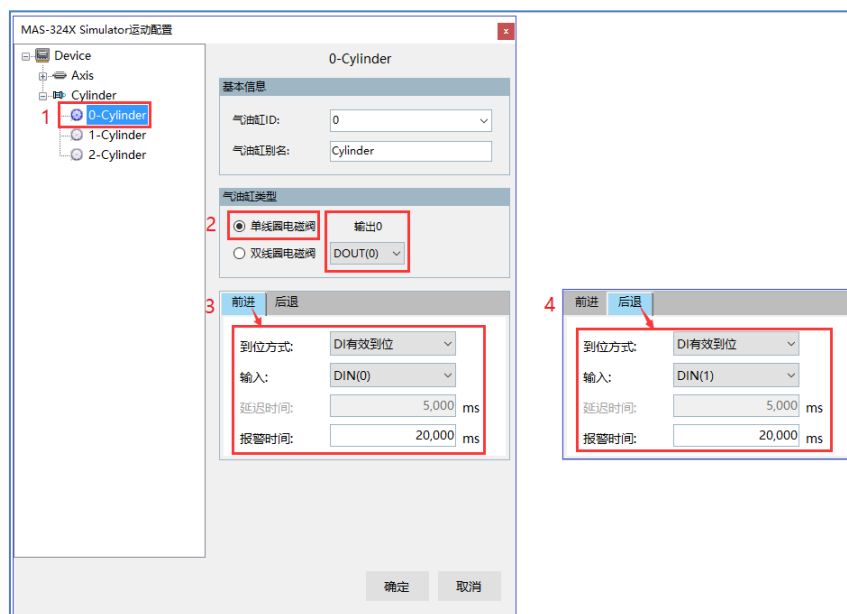


步骤二：依据各气缸的硬件规格进行配置

- 气缸1：如下图，单线圈电磁阀控制，前后都有安装限位。



按下面步骤进行气缸1的配置。

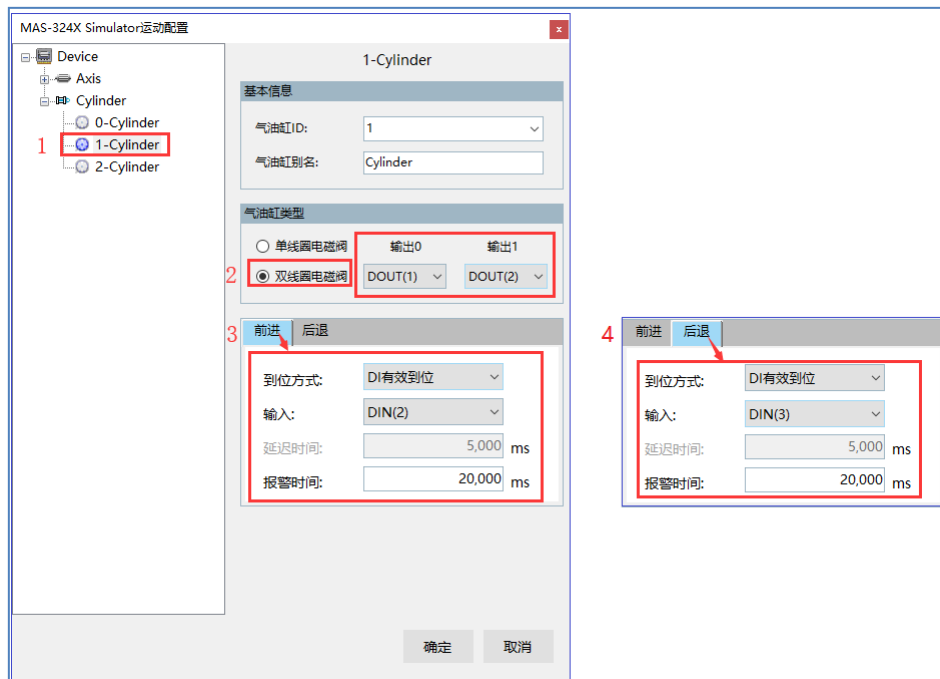


1. 点击进入 ID 为 0 的气缸
2. 选择“单线圈电磁阀”，用 DOUT(0)作为电磁阀的控制。
3. 进入“气缸前进”动作的配置，选择“DI 有效到位”，前限位接入 DIN(0)
4. 进入“气缸后退”动作的配置，选择“DI 有效到位”，后限位接入 DIN(1)

- 气缸2：如下图，双线圈电磁阀控制，前后都有安装限位。

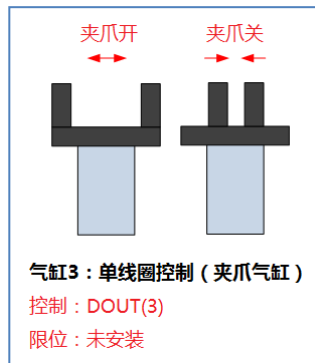


按下面步骤进行气缸2的配置。



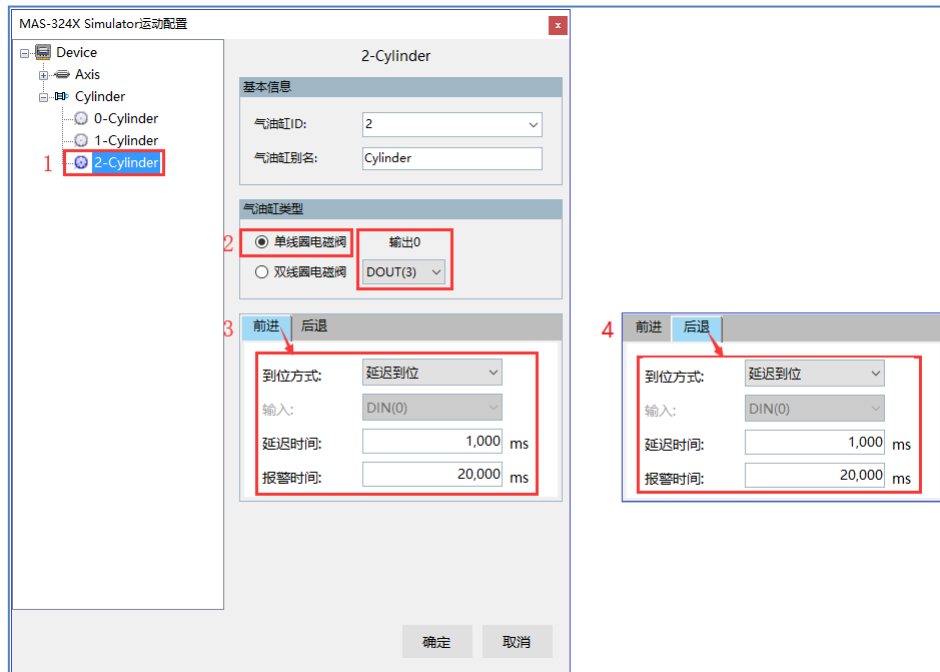
1. 点击进入 ID 为 1 的气缸
2. 选择“双线圈电磁阀”，用 DOUT(1), DOUT(2)作为电磁阀的控制。
3. 进入“气缸前进”动作的配置，选择“DI 有效到位”，前限位接入 DIN(2)
4. 进入“气缸后退”动作的配置，选择“DI 有效到位”，后限位接入 DIN(3)

- 气缸 3：如下图，单线圈电磁阀控制，前后都没有安装限位。使用延时到位，即气缸动作后延时一段时间就认为气缸到位了。



注意：一般情况下，不推荐气缸不安装限位，因控制器无法确认气缸是否正常动作。

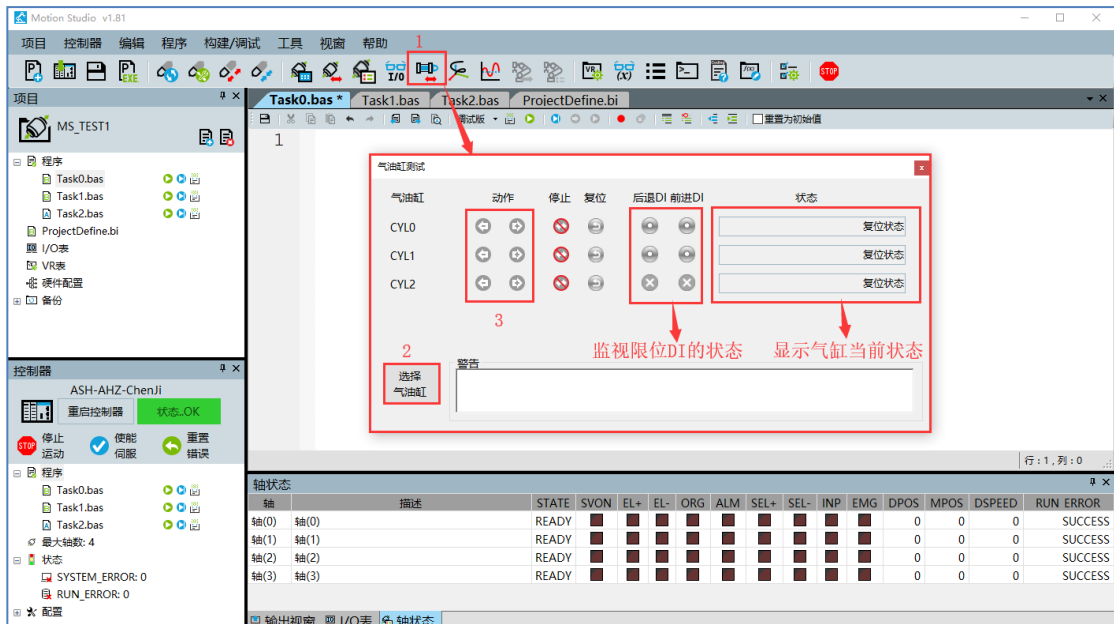
按下面步骤进行气缸 3 的配置。



1. 点击进入 ID 为 2 的气缸
2. 选择“单线圈电磁阀”，用 DOUT(3)作为电磁阀的控制。
3. 进入“气缸前进”动作的配置，选择“延迟到位”，延迟时间 1 秒（根据实际情况填延迟时间）
4. 进入“气缸后退”动作的配置，选择“延迟到位”，延迟时间 1 秒（根据实际情况填延迟时间）

5.2 如何测试气缸控制

据 9.1 章节说明，配置好气缸的规格后，在进行编程控制前，可以使用气缸测试工具进行手动测试，测试配置和实际的硬件规格是否匹配，硬件接线是否正常，气缸能否正常动作。如下步骤即可进行气缸控制测试。



1. 打开气缸测试工具
2. 选择要测试哪些气缸，已配置的气缸才会在选择列表中。
3. 点击前进或后退按钮进行气缸动作测试。
4. 观察限位的DI状态和气缸当前状态，同时结合实际气缸动作，判断气缸动作是否正常。

5.3 如何编写程序控制气缸动作

编写程序之前，需要先参照 9.1 章节配置气缸控制，配置后可知每个气缸 ID 对应的气缸控制规格。根据气缸 ID 就可控制气缸动作。如下举例说明气缸动作编程。

◆ 控制单个气缸动作

```
'使用CYL_BASE控制气缸
CYL_BASE 1      '基于ID为1的气缸
CYL_MOVE 1      '气缸1执行"前进"动作
WAIT CYLDONE    '等待气缸1前进到位
CYL_MOVE 0      '气缸1执行"后退"动作
WAIT CYLDONE    '等待气缸1后退到位
```

```
'指定气缸ID控制气缸
CYL_MOVE CYL(2),1      '气缸2执行"前进"动作
WAIT CYL(2),CYLDONE    '等待气缸2前进到位
CYL_MOVE CYL(2),0      '气缸2执行"后退"动作
WAIT CYL(2),CYLDONE    '等待气缸2后退到位
```

◆ 多个气缸同时动作

```
CYL_BASE 2,3      '两个气缸一起动作
CYL_MOVE 1,0      '气缸2,3分别执行"前进"、"后退"动作
WAIT CYLDONE      '等待气缸2,3动作到位完成

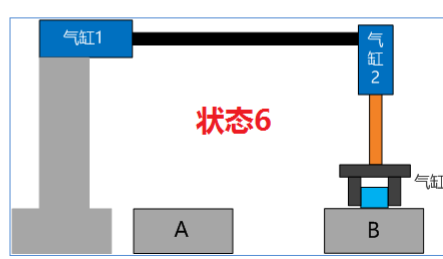
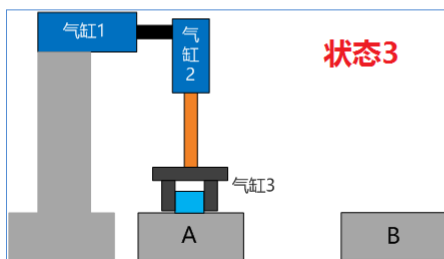
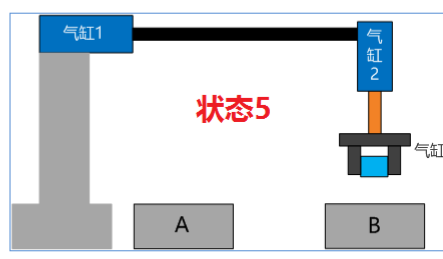
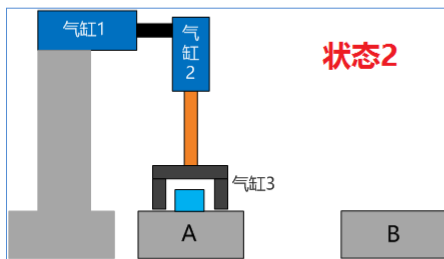
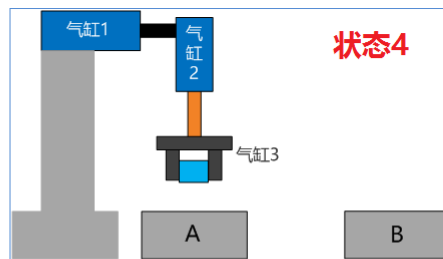
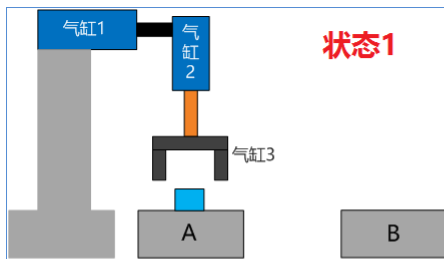
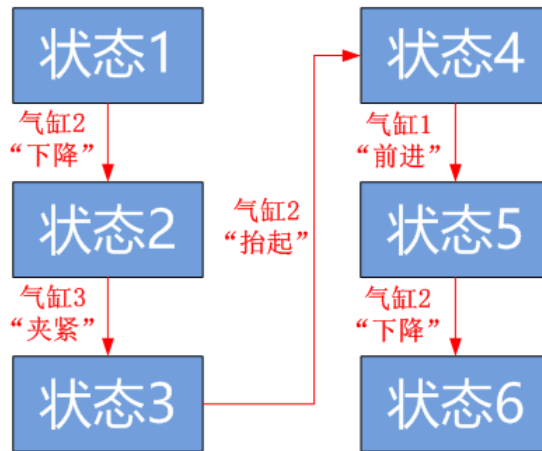
CYL_BASE 0,1,4    '三个气缸一起动作
CYL_MOVE 1,1,1    '气缸0,1,4都执行"前进"动作
WAIT CYLDONE      '等待气缸0,1,4动作到位完成
```

◆ 注意

CYLBASE 一次性最多指定 8 个气缸 ID。

◆ 气缸顺序流程控制编程

如下图，要做下面顺序流程控制，从状态 1 顺序动作到状态 6。



程序代码:

CYL_MOVE CYL(2),1	'气缸2"下降"
WAIT CYL(2),CYLDONE	'等待气缸2动作到位
CYL_MOVE CYL(3),1	'气缸3"夹紧"
WAIT CYL(3),CYLDONE	'等待气缸3动作到位
CYL_MOVE CYL(2),0	'气缸2"抬起"
WAIT CYL(2),CYLDONE	'等待气缸2动作到位
CYL_MOVE CYL(1),1	'气缸1"前进"
WAIT CYL(1),CYLDONE	'等待气缸1动作到位
CYL_MOVE CYL(2),1	'气缸2"下降"
WAIT CYL(2),CYLDONE	'等待气缸2动作到位

5.4 Cylinder 类

在 Motion Studio 程序中,用户基于专用运动卡片上的 IO,可以使用前面介绍的气缸控制功能,高效、便捷地进行气缸控制。

但是当用户使用 PCI-1750、PCI-1730 等非运动控制卡时,无法使用上面介绍的气缸控制功能。此时, Motion Studio 提供气缸控制的 Cylinder 类方法,供用户用于气缸控制。

5.4.1 Cylinder 类基本属性

Cylinder 类的基本属性与方法如下表：

成员	成员属性	说明
FwCoil	Type_DO	双线圈时：指定前进线圈的输出点 DO
BwCoil	Type_DO	双线圈时：指定后退线圈的输出点 DO
Coil	Type_DO	单线圈时：指定线圈的输出点 DO
Fwsensor	Type_DI	指定前进到位的感应器信号输入点
Bwsensor	Type_DI	指定后退到位的感应器信号输入点
FwDoneType	ULONG	设定气缸前进到位方式，如下： 0：延时到位：气缸动作后，延时指定时间到，即认为气缸动作到位 1：限位到位：气缸动作后，遇到指定的限位有效，即认为气缸动作到位计时器启动后，执行此方法刷新计时器 2：（限位+延时）到位：气缸动作后，遇到指定的限位有效，再延时指定时间后，即认为气缸动作到位 3：（延时+限位）到位：气缸动作后，延时指定时间后，再检测到指定的限位有效，即认为气缸动作到位
BwDoneType	ULONG	设定气缸后退到位方式，同上
FwTime	ULONG	在延时到位方式下，设定前进到位延时时间
BwTime	ULONG	在延时到位方式下，设定后退到位延时时间
FwAlmTime	ULONG	前进超时时间，如超过该时间还未到位，则气缸报警
BwAlmTime	ULONG	后退超时时间，如超过该时间还未到位，则气缸报警
Forward()	FUNCTION	无形参，执行气缸前进动作
Backward()	FUNCTION	无形参，执行气缸后退动作
Stop()	FUNCTION	无形参，执行气缸停止动作
Reset()	FUNCTION	无形参，用于复位气缸的状态，并重置内部计时器
Status()	FUNCTION	无形参，返回气缸状态，详见气缸状态获取的介绍
WaitDone()	FUNCTION	无形参，等待气缸完成，此为阻塞方式，谨慎使用

有关类的详细说明请见 BASIC 手册。

5.4.2 用 Cylinder 类方法控制气缸

用 Cylinder 类方法控制气缸的步骤如下：

1. 实例化气缸
2. 配置气缸参数
3. 执行气缸动作
4. 判断气缸动作完成

下面实例化 Cly1、Cly2 两个气缸，作为示例讲解。

步骤一：实例化气缸

如下示例，实例化两个名称为 Cly1、Cly2 的气缸用于控制。

```
DIM Cly1 AS Cylinder
DIM Cly2 AS Cylinder
```

步骤二：配置气缸参数

◆ 线圈输出配置

以 Cly1 为双线圈气缸，Cly2 为单线圈气缸，对其输出点进行配置，示例如下：

```
Cly2.Coil=DOUT(20)      '单输出线圈配置
Cly1.FwCoil=DOUT(16)   '前进输出线圈
Cly1.BwCoil=DOUT(17)   '后退输出线圈
```

◆ 气缸到位方式、到位信号配置

将气缸 Cly1 设为“限位+延时”到位方式，并进行感应器信号与延时时间配置，示例如下：

```
Cly1.FwDoneType=2      '前进到位方式 :限位+延时
Cly1.BwDoneType=2      '后退到位方式 :限位+延时
Cly1.Fwsensor=DIN(16)  '前进到位感应器信号
Cly1.Bwsensor=DIN(17)  '后退到位感应器信号
Cly1.FwTime=5000       '前进到位延时
Cly1.BwTime=5000       '后退到位延时
```

◆ 超时报警时间配置

当气缸动作时间超过报警时间后，会触发超时报警。

为气缸 Cly1 设置超时报警时间，示例如下：

```
Cly1.FwAlmTime=20000      '前进报警延时
Cly1.BwAlmTime=20000      '后退报警延时
```

注：气缸报警后，气缸自动停止动作。

步骤三：执行气缸动作

◆ 气缸复位

- 初次使用气缸时，需要复位，初始化气缸状态。
- 超时报警后，需要复位，清除报警。

复位气缸 Cly1，示例如下：

```
Cly1.Reset()
```

注：对气缸执行复位，不会改变线圈的输出状态。

◆ 控制气缸前进、后退

控制气缸 Cly1 后退，Cly2 前进，示例如下：

```
Cly1.Backward()
Cly2.Forward()
```

也可以用以下这种写法：

```
Cly1.move 0
Cly2.move 1
```

注：对单线圈气缸来说，只有前进动作。

◆ 气缸停止

停止气缸 Cly1 动作，示例如下：

```
Cly1.Stop()
```

步骤四：判断气缸动作完成

- ◆ 通过读取气缸状态，判断是否动作完成

气缸状态，STATUS 值与对应的状态如下：

- 0：复位：原始状态。执行 CYL_Stop、CYL_Reset 后的气缸状态都为复位状态
- 1：前进到位
- 2：后退到位
- 3：前进中
- 4：后退中
- 9：超过到位时间报警

做一个简单气缸控制，启动按钮 VR(1)按下，气缸便会后退，示例如下：

```

DIM Cly1 AS Cylinder
DIM StepFlag AS INTEGER           '定义流控步骤
MS_LOOP(10)
  SELECT CASE StepFlag
  CASE 0
    IF MS_EDGER(VR(1)) THEN StepFlag=1 '启动按钮按下
  CASE 1
    Cly1.Backward()
    StepFlag=2
  CASE 2
    IF Cly1.Status()=2 THEN
      Cly1.Stop()
      StepFlag=0
    END IF
  CASE 3
    .....
  END SELECT
MS_LEND

```

上面示例中，在 CASE 2 中，气缸状态为 2 时，判断气缸后退完成。

实时扫描气缸 Cly1 的状态为例，当其报警时，另报警标志为 1，示例如下：

```

DIM CylAlarmFlag AS INTEGER '定义报警标志
WHILE 1
  IF Cly1.STATUS=9 THEN CylAlarmFlag=1
  SLEEP 10
WEND

```

◆ 等待气缸完成事件: WaitDone()

除了上述的判断气缸状态外, 也可用 WaitDone()事件等待气缸停止。

执行 WaitDone()方法, 程序会一直等待在 WaitDone()处, 直至气缸动作停止(包含异常停止)。

仍以实例化的气缸 Cly1 为例, 示例 WaitDone()使用方法:

```

Cly1.Forward()
Cly1.WaitDone()
```

事实上, 上面代码只有 2 行, 而这 2 行程序的执行时间却等于气缸前进的时间。

◆ 集中配置气缸参数

上述的到位方式设定、时间设定都是单个设定, 显得复杂。

Cylinder 类提供另外一种集合配置方法, 非常方便。

成员	成员属性	说明
SetTime (FwTime, BwTime, FwAlmTime, BwAlmTime)	SUB	集中配置气缸时间参数
DoneType(FwDoneType, BwDoneType)	SUB	集中配置气缸到位方式参数

详细说明请见 BASIC 手册。

示例如下:

```

DIM Cly3 AS Cylinder
Cly3.DoneType(2,3)
    '集中配置:前进到位方式为(限位+延时),后退到位方式为(延时+限位)
Cly3.Settime(5000,5000,20000,20000)
    '集中配置时间,前进/后退超时为5000ms,报警超时20000ms
```

第六章 运动控制

6.1 执行运动控制的基本步骤

进行一段运动控制，须有以下步骤要素：

1. 指定操作轴
2. 使能轴
3. 配置运动参数
4. 执行运动轨迹
5. 等待运动结束

下面将分别进行讲解。

6.2 BASE:指定操作轴

运动控制的对象是轴，在进行运动控制之前，要指定控制的对象，即用 BASE 指定操作轴。

◆ BASE 使用示例

例程 1: 指定轴 0，设定其运行速度为 10，如下：

```
BASE 0  
VH=10
```

例程 2: 设定轴 0、2 运行速度为 10，然后设定轴 1、3 运行速度为 15，如下：

```
BASE 0, 2  
VH=10  
BASE 1, 3  
VH=15
```

◆ 注意

使用 BASE 指定操作轴后，其后的运动控制都是针对指定的轴。如果要改变操作的轴，只需再次执行 BASE 指令即可。

◆ 注意

BASE 一次性最多指定 8 个操作轴。

6.3 WAIT DONE:等待运动完成

◆ WAIT DONE 的用处

运动控制中，可能会面临以下状况

- a) 两段运动轨迹不能同时执行，即等待一个轨迹执行完后，再执行下一个轨迹
- b) 不能对正在运动的轴再次下发运动指令

此时，就要用 WAIT DONE 指令，用于等待运动完成。

◆ WAIT DONE 指令执行完成的机制

执行运动轨迹时，当轴的运动状态从 BUSY 变为 READY 后，认为运动结束，此时 WAIT DONE 执行完成。

• 轴状态说明

在轴未报警的前提下：

- a) 轴在执行运动时，状态为 BUSY
- b) 轴未运动时，或者运行停止后，状态为 READY

◆ 注意：

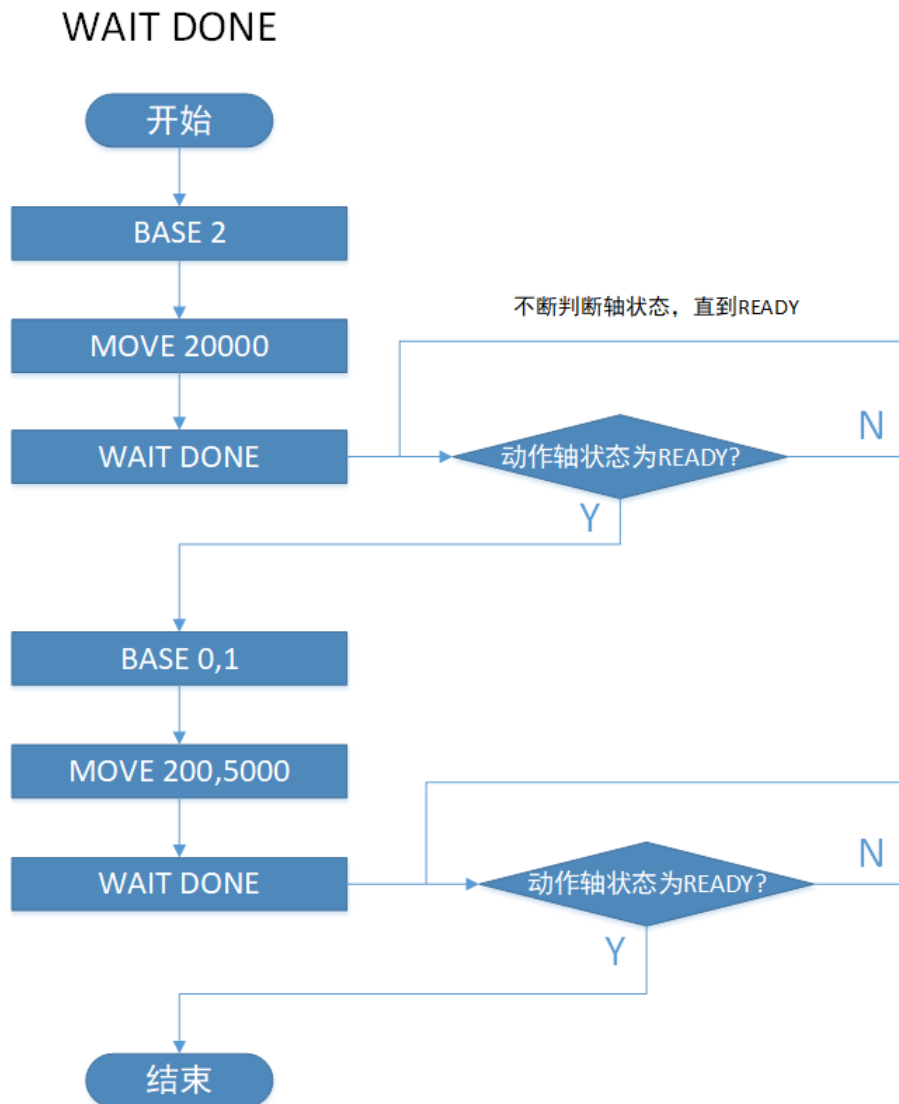
启用轴 INP（伺服驱动器的到位信号）情况下，WAIT DONE 的执行完成条件还要以接收到 INP 信号为首要条件。

◆ WAIT DONE 应用示例

先让轴 2 正向移动 20000，动作结束后，再让轴 0、轴 1 分别移动至 200，5000，代码如下：

```
BASE 2
SVON
MOVE 20000
WAIT DONE
BASE 0, 1
SVON
MOVE 200, 5000
WAIT DONE
```

代码的执行流程如下：



上述代码，在执行第 4 行 WAIT DONE 的时候，只有轴 2 停止后，即轴 2 处于 READY 后，WAIT DONE 才算执行完成，才能继续向下执行。

◆ 注意

由 WAIT DONE 指令执行机制可知，若上述代码中轴 2 在动作时，对其下 STOPDEC 或者 STOPEMG 令其停止运动后，轴 2 的状态也会变为 READY，也会导致第 4 行 WAIT DONE 执行完成。

6.4 相对运动指令和绝对运动指令

◆ 相对运动指令与绝对运动指令

在 Motion Studio 编程环境中，运动指令有“相对运动指令”和“绝对运动指令”之分，“绝对运动指令”后面会加上 ABS 三位字母。

例如，“LINE”是相对直线插补指令，“LINEABS”是绝对直线插补指令。

◆ 相对运动指令与绝对运动指令差别

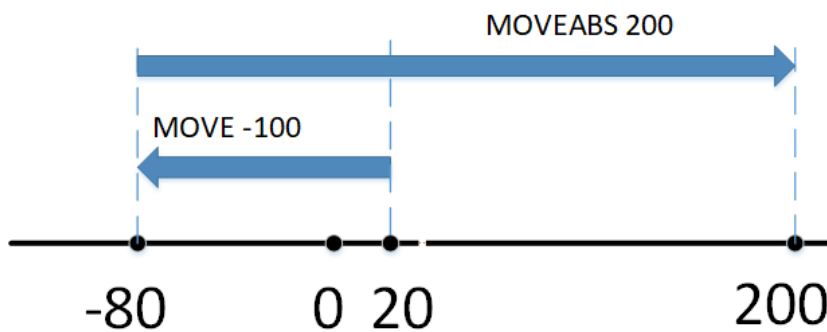
相对运动指令：直接指定运动方向与运动距离。

绝对运动指令：指定目标点。板卡内部计算当前位置与目标点位的向量差，得出运动方向和运动距离。

◆ 例程：

```
BASE 0  
MOVE -100  
WAIT DONE  
MOVEABS 200  
WAIT DONE
```

若当前位置为 20，上述代码中，运动轨迹如下：



“MOVE -100”直接给出：运动的方向为负向，运动的距离是 100 个单位，属于相对运动指令。

“MOVEABS 200”则是直接给出运动的目标点位 200，而运动的方向与距离则由“当前点位与目标点位的连线向量”计算可得，属于绝对运动指令。

6.5 基础轴控运动

6.5.1 如何进行单轴点位运动

单轴点位运动，运行速度设定和位移设定都是针对单个轴的设定，各轴之间运行时互不关联。

单轴相对点位运动使用指令 MOVE，单轴绝对点位运动使用指令 MOVEABS。

◆ 单轴点位运动示例

例程 1:让轴 0 正向移动距离为 1000

```
BASE 0  
MOVE 1000  
WAIT DONE
```

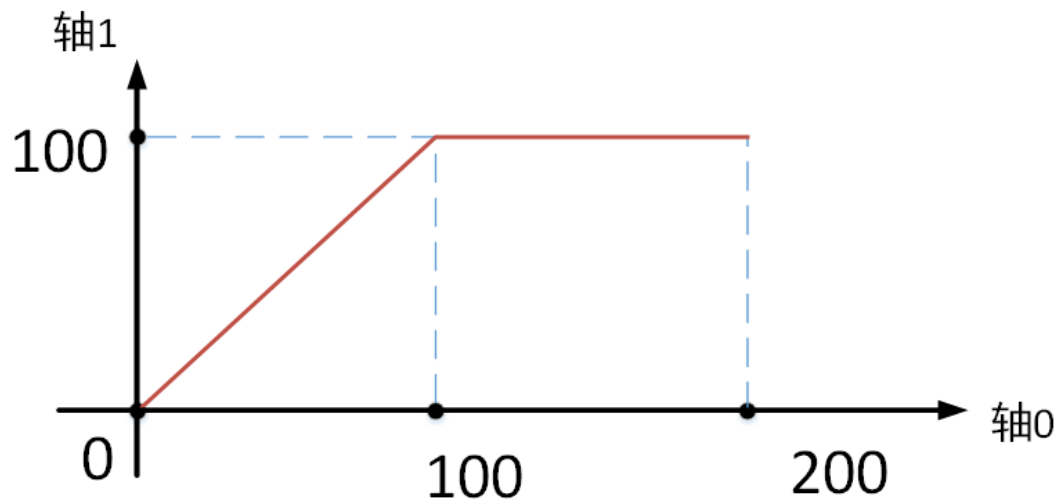
例程 2: 指定轴 1，移动至绝对位置为 5000 的点位

```
BASE 0  
MOVEABS 5000  
WAIT DONE
```

例程 3:

```
BASE 0, 1  
DPOS=0  
VH=100  
MOVEABS 200, 100  
WAIT DONE
```

例程 3 的中运行合成轨迹如下图:



轨迹分析：“VH=100”设定两个轴的运行速度皆为 100；设定轴 0 的运行距离为 200，轴 1 的运动距离为 100，因两轴速度一样，所以两个轴不会同时到达终点，合成轨迹出现了图中的拐角。

所以，多个轴进行单轴运动时，其合成位移曲线可能是曲线，也有可能是直线。

6.5.2 如何进行直线插补运动

直线插补运动中，运行速度设定是对参与轴的合成速度设定，运行位移设定是对参与轴的合成位移设定。

直线插补运动中各轴之间配合联动，同时启动和停止，所以，多个轴进行直线插补运动时，其合成位移曲线必是直线。

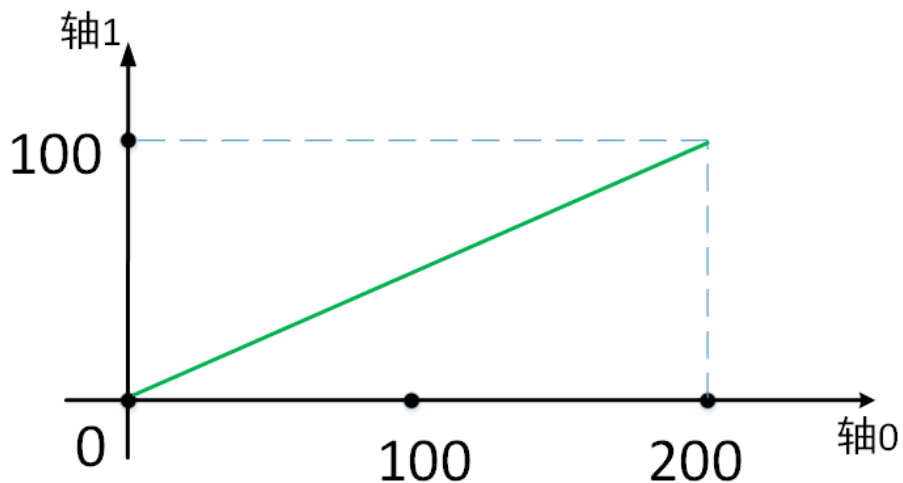
直线插补运动的参与轴数为 2-3 轴，相对直线插补使用指令 LINE，绝对直线插补使用指令 LINEABS。

◆ 直线插补运动示例

轴 0、1 的绝对插补运动

```
BASE 0, 1
DPOS=0
MPOS=0
GVH=100
LINEABS 200, 100
WAIT DONE
```

运行轨迹如下图：



例程轨迹分析：可与单轴点位运动章节中例 3 的运动轨迹比较，此例中，GVH=100 设定两个轴的合成运行速度为 100；两轴的起始坐标为 (0, 0)，设定两轴的终点坐标为 (200, 100)。两轴协同插补，同时到达终点，合成轨迹是直线。

6.5.3 如何进行圆弧插补运动

和直线插补运动相同，在圆弧插补运动中，运行速度设定、位移设定是对参与轴的合成速度、位移设定。

◆ 圆弧插补运动的轨迹指定

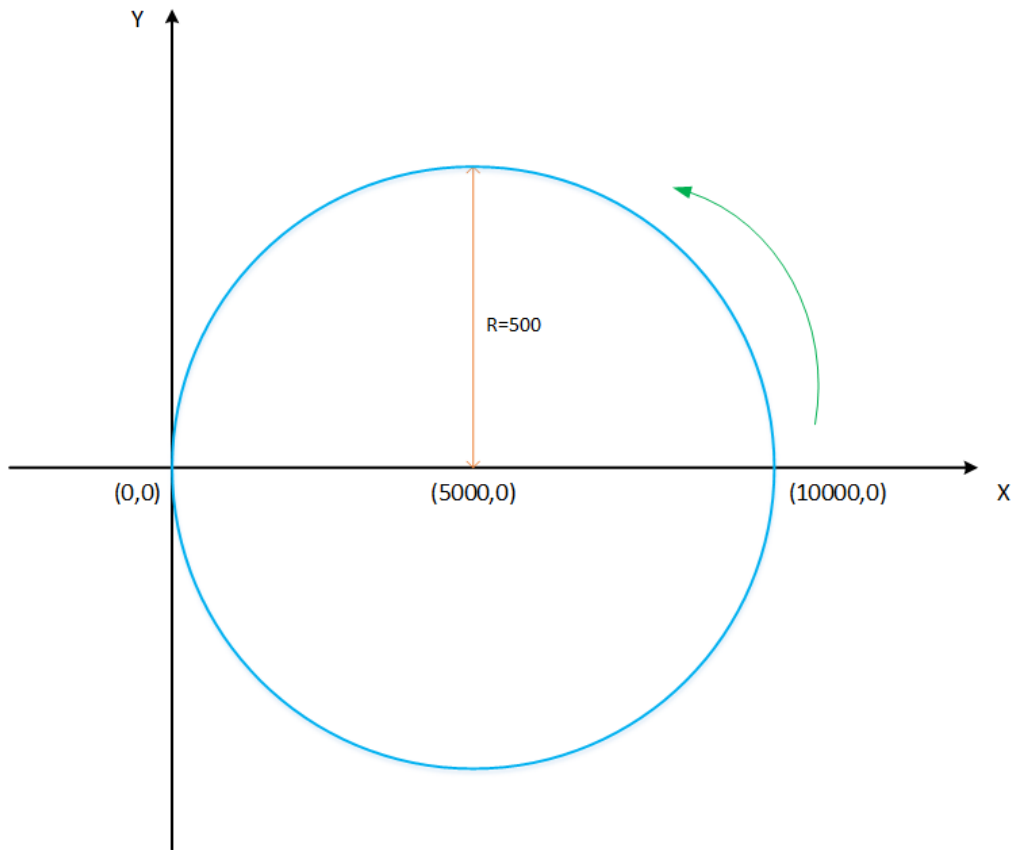
圆弧插补运动中，需要指定要素来确定圆弧轨迹，具体指定方法如下：

指定圆弧方向、圆心、终点	指令为 CIRC、CIRCABS
指定圆弧方向、圆弧上 3 点	指令为 CIRC_3P、CIRCABS_3P
指定圆弧方向、圆心、角度	指令为 CIRC_A 、CIRCABS_A

指令用法见 BASIC 手册。

◆ 圆弧插补运动示例

下图轨迹，以 (0, 0) 为起始点，以 CIRCABS 为例，走出图中圆弧轨迹。



分析：使用轨迹指定的第一种方法，指定圆弧方向为逆时针，圆心位置 (5000, 0)，终点位置为 (0, 0)

程序如下：

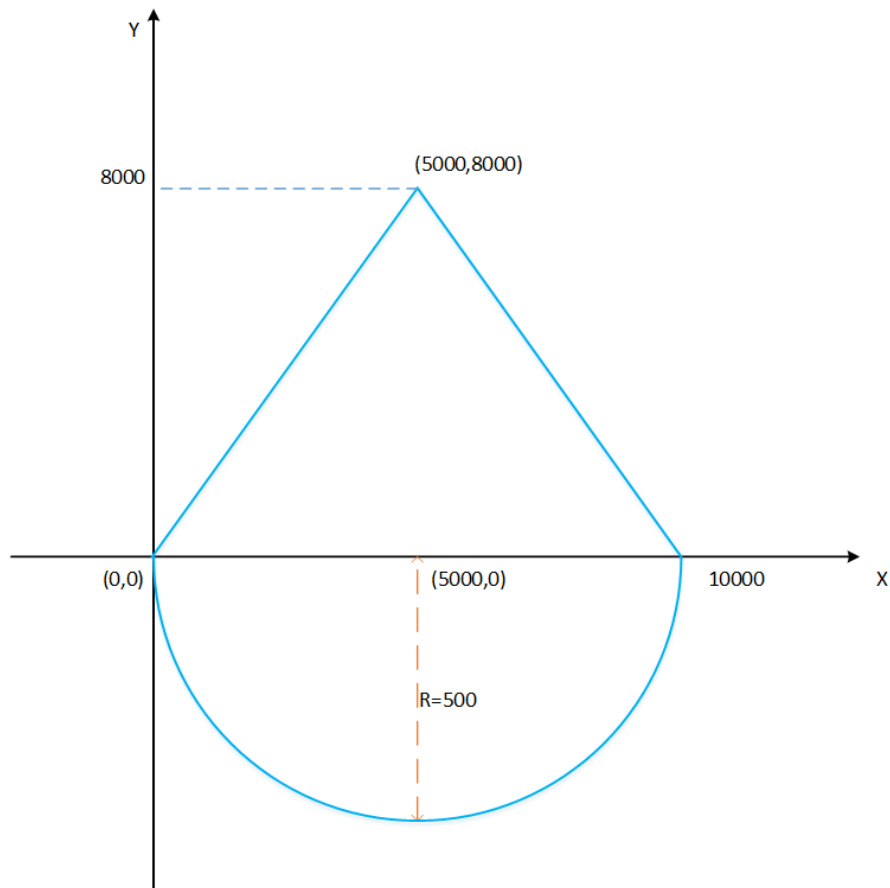
```
BASE 0, 1
SVON
DPOS=0
MPOS=0
CIRCABS 1, 5000, 0, 0, 0
WAIT DONE
```


6.5.4 如何进行连续插补运动

◆ 连续插补运动轨迹说明

连续插补运动中，包含多段插补轨迹，各个轨迹可以是直线或者曲线。

以下图，举个简单的例程：



在上图中，若当前位置为 (0, 0) ，以逆时针走完图中轨迹，则共有三段插补轨迹：

阶段 1，圆弧插补： CIRCABS 1, 5000, 0, 10000, 0

阶段 2，直线插补： LINECABS 5000, 8000

阶段 3，直线插补： LINECABS 0, 0

◆ 连续插补指令介绍

运行连续插补模式，要使用连续插补指令，相关指令如下：

- PATHRESET 清除连续路径缓存
- PATHBEGIN 开始进入连续插补模式
- PATHEND PATHBEGIN 对应的结束指令
- MERGEON 以 fly mode 速度交接模式执行插补运动
- MERGEOFF 以 buffer mode 速度交接模式执行连续插补运动

指令详细用法见 BASIC 手册。

◆ 连续插补运动示例

使用连续插补指令，运行上述轨迹的代码如下

```
BASE 0, 1
SVON
GVL=10
GVH=100
PATHRESET
PATHBEGIN
MERGEON
    CIRCABS 1, 5000, 0, 10000, 0
    LINECABS 5000, 8000
    LINECABS 0, 0
PATHEND
WAIT DONE
```

上面代码中，PATHBEGIN 与 PATHEND 之间的执行轨迹即是连续插补轨迹。

◆ 连续插补运动的速度交接模式

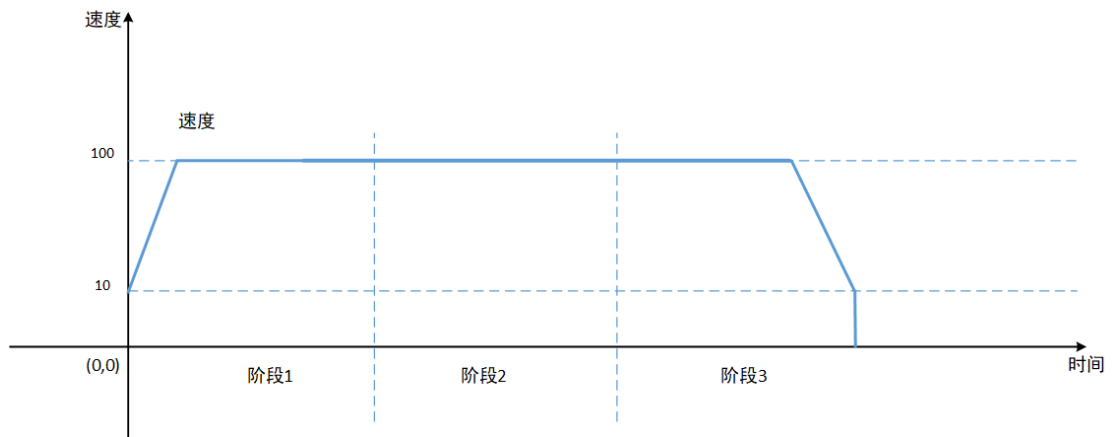
执行连续插补轨迹，涉及速度交接模式的设定，如上面代码中的 MERGEON 的使用，速度交接模式有两种，如下：

MERGEON : fly mode , 指前一段路径的 GVH 加速或减速到后一段路径的 GVH 的速度交接方式。

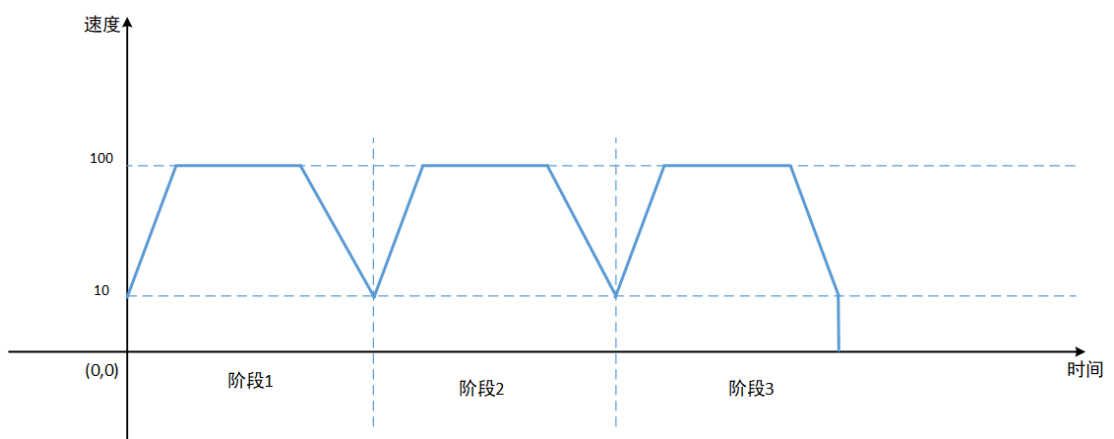
MERGEOFF : buffer mode, 指前一段路径的 GVH 加速或减速到后一段路径的 GVL 的速度交接方式

◆ 速度交接模式对比说明

上述代码中使用 fly mode，其运行速度变化如下：



如上述代码将 MERGEON 换成 MERGEOFF，即使用 buffer mode，其运行速度变化如下：



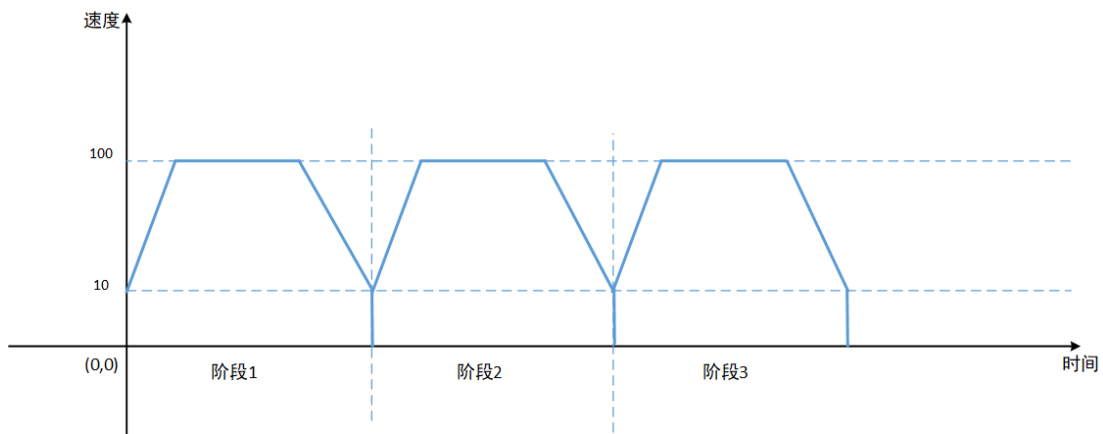
对比可知，fly Mode 在轨迹过渡阶段的速度转换时间短，效率更高，即 MERGEON 效率高于 MERGEOFF。

◆ 普通模式与连续插补模式的比较

如果以普通模式（非连续插补模式）走完上述轨迹，其代码如下：

```
BASE 0, 1
SVON
GVL=10
GVH=100
CIRCABS 1, 5000, 0, 10000, 0
WAIT DONE
LINECABS 5000, 8000
WAIT DONE
LINECABS 0, 0
WAIT DONE
```

运行中的速度变化过程如下：



从图中可知，普通模式下，每一段轨迹结束时，速度都会减速到零，需要反应时长更多。

可知，执行连续轨迹时，连续插补模式的效率高于普通模式。

6.5.5 轴运动的 P 点操作

之前介绍的多轴运动指令，在指定位置时需要在指令后逐一写出点位值，比较繁琐，且不直观。Motion Studio 提供 PT 类方法，可以简化运动指令的位置操作，称之为 P 点。

先用一个简单的例子，示例如何使用 PT 方法：

```
DIM AS PT PickPlace=PT(200,500,300,400)
BASE 0,1,2,3
MOVE PickPlace '前往取料点位
WAIT DONE
```

上面的代码中，直接用 PickPlace 代替点位，非常方便直观。再对比下面的代码：

```
BASE 0,1,2,3
MOVE 200,500,300,400 '前往取料点位
WAIT DONE
```

显而易见，使用 PT 方法操作点位，更加高效、便捷、直观。

◆ PT 方法的基本操作

使用 DIM 实例化 PT 方法，并进行 P 点操作，如下：

```
DIM P0 AS PT '定义一个P0点
P0=PT(0.1,0.1,0.1) 'P0点赋值
DIM AS PT P1=PT(60,60,60) '定义一个P1点并赋值
DIM AS PT P2,P3,P4,P5 '定义多个P点
P2=PT(100,100)
P3=PT(700,400,200)
P3=P1-P2+P0 '多个P点可进行位置运算
BASE 0,1 'P点进行绝对运动
MOVEABS P0
WAIT DONE
MOVE P1 'P点进行相对运动
WAIT DONE
```

1 个 P 点最多存储 8 个 DOUBLE 类型的数值。

使用 P 点时，根据轴的个数从前往后取出存储值，上图中“MOVE P1”实际上只用到了 P1 内存储的前面两位数据，即等同于“MOVE 60,60”。

◆ PT 类与 Tab 类的结合使用

有关 Tab 类的介绍，请参考 3.4.3 章节。

起始序号:	<input type="text" value="10008"/>	行数:	<input type="text" value="5"/>	列数:	<input type="text" value="5"/>
序号	列0	列1	列2	列3	列4
0	0	5	53	3	887
1	1	6	88	8	66
2	2	9	85	6	55
3	3	99	45	7	55
4	4	88	23	66	81

以上面的 Table 表为例，取出上表中 (6, 88, 8) 点位，用于做运动控制。以此来展示 PT 类与 Tab 类的结合使用，如下示例

```

DIM AS Tab Para1
DIM PickPlace AS PT
Para1=Tab(10008,5,5)    '实例化Para1表格
Para1.MAPPoint(1,3)
'从实例化的Para1表格里，从列1开始，取出3列
PickPlace=Para1.Point(1)
'从取出的列中，取出行1，赋值给PickPlace
BASE 0,1,2
LINEABS PickPlace
WAIT DONE

```

图示例中，Tab 类与 PT 类结合使用，使 Table 表在参数管理和应用上有更好的效果。

6.6 回原点运动

6.6.1 回原模式的类型

实际应用中，有多种信号可以用来回原点，最基本的有如下三种：

- a) 原点感应器信号
- b) 极限感应器信号
- c) 编码器 Z 相信号

结合这三种回原点信号，衍生出多种回原类型。

研华的运动控制卡提供的回原模式类型如下：

- 0: MODE1_Abs
- 1: MODE2_Lmt
- 2: MODE3_Ref
- 3: MODE4_Abs_Ref
- 4: MODE5_Abs_NegRef
- 5: MODE6_Lmt_Ref
- 6: MODE7_AbsSearch
- 7: MODE8_LmtSearch
- 8: MODE9_AbsSearch_Ref
- 9: MODE10_AbsSearch_NegRef
- 10: MODE11_LmtSearch_Ref
- 11: MODE12_AbsSearchReFind
- 12: MODE13_LmtSearchReFind
- 13: MODE14_AbsSearchReFind_Ref
- 14: MODE15_AbsSearchReFind_NegR
- 15: MODE16_LmtSearchReFind_Ref
- 101~137: CiA402_MODE1 ~ CiA402_MODE37 (EtherCAT 伺服)

各个回原模式不再详细介绍，用户可在 Motion Studio 软件中的硬件配置中，通过轴 HOME 类型设置的相关介绍自行了解。

◆ 回原模式选择示例

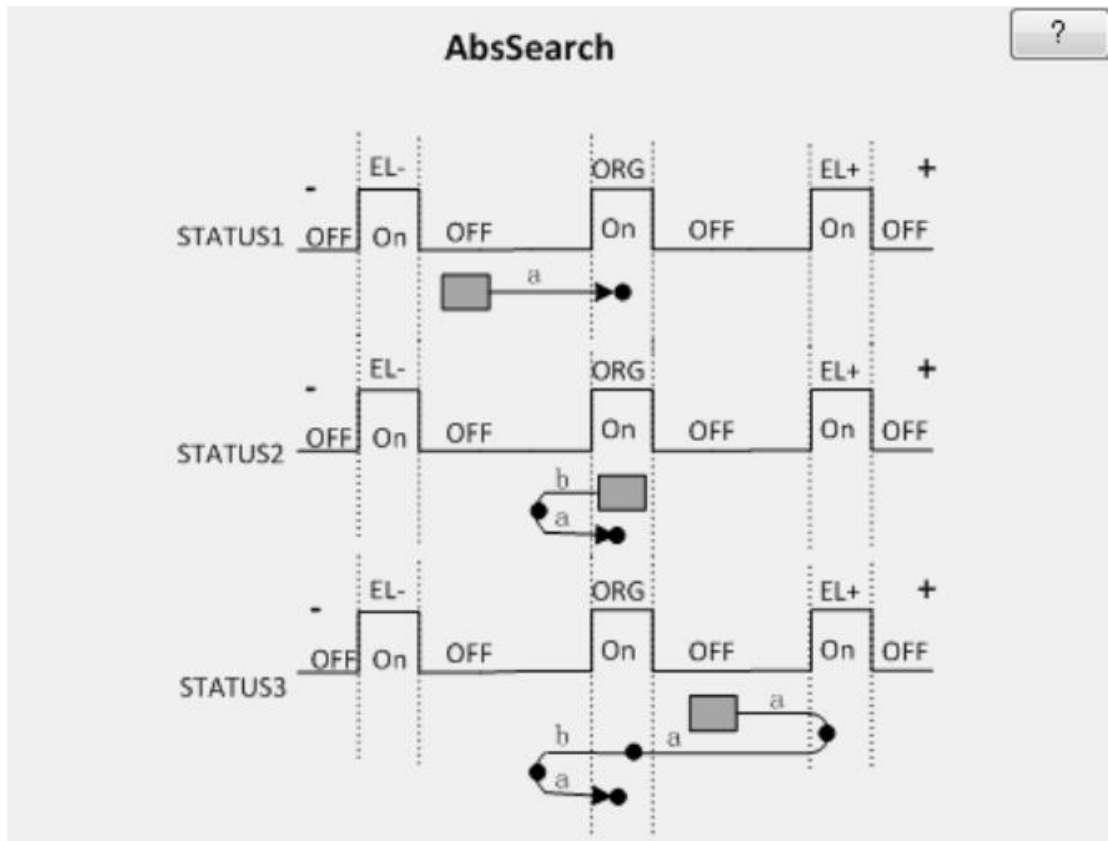
选择回原模式用 HOME_MODE 指令。

选择轴 0 的回原模式为 MODE7_AbsSearch，如下：

BASE 0

HOME_MODE=6

AbsSearch 模式介绍如下图：



上图中，说明了在 AbsSearch 回原模式下，三种不同情况的正向回原轨迹。

6.6.2 如何编写回原程序

◆ 回原程序的步骤

1. 指定操作轴
2. 回原速度设定
3. 回原模式选择
4. 指定回原方向，进行回原。HOME_P 执行正向回原，HOME_N 执行反向回原。
5. 等待回原结束

◆ 注意：

在进行回原之前，请确认相关感应器（极限位感应、原点感应）信号正常。

◆ 回原程序示例

先让轴 0、1 以 AbsSearch 模式正向回完原点后，再让轴 0 以 LmtSearch 模式进行负向回原，程序如下：

```
BASE 0, 1, 2
HOME_VL=500
HOME_VH=10000
HOME_ACC=50000
HOME_DEC=50000
BASE 0, 1
HOME_MODE=6
HOMEP
WAIT DONE
BASE 2
HOME_MODE=7
HOMEN
WAIT DONE
```

执行完上述程序后，轴 0，轴 1 停在了原点感应器的位置，轴 2 停在了负向极限感应器的位置。

6.6.3 回原时的异常处理

- ◆ 原点感应器损坏时，怎么回原点？

可选择通过极限感应器或者编码器 Z 相信号回原点

- ◆ 选择原点感应器模式回原点时，无法找到原点？

首先确认硬件是否正常。

确认硬件正常后，请查看原点感应器信号的接入点电平是否设置正确。

- ◆ 选择极限感应器模式回原点时，如何让电机不停留在极限感应器的位置？

使用原点偏移功能，设置回原点运动完成后，再移动一段偏移距离，即使用 HOME_OFFSETDIST 指令，用法如下。

例 程：

```
BASE 0  
HOME_OFFSETDIST =1000
```

- ◆ 使用 REFIND 类型回原点时，找到原点后，回原进程却无法结束？

REFIND 类型回原点时，要保证 HOME_VL 值大于 0，请确认 HOME_VL 是否为 0。

6.7 如何停止运动

◆ 让轴停止的方法

运动事件在到达目的地后会自行停止，但也可以用以下方法让轴中途停止运动。

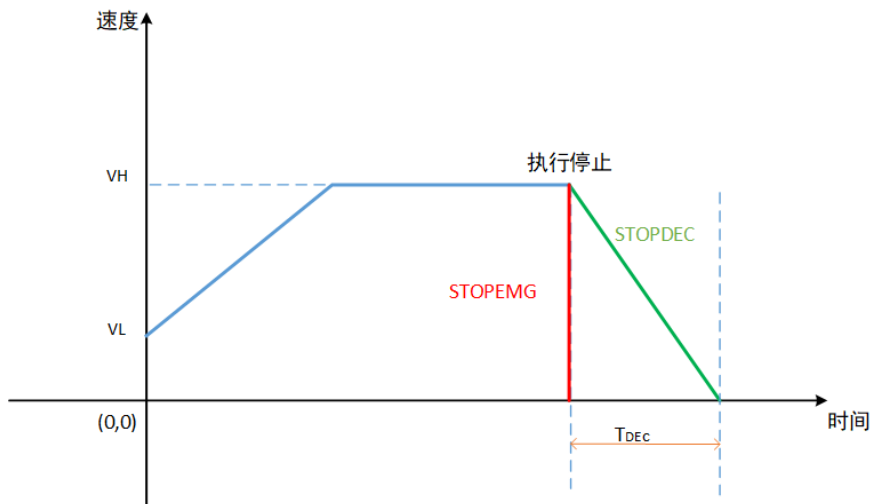
- 外部硬件停止，如外部急停按钮，用户可自行接线
- 通过轴的 INI，即立即停止功能
- 指令停止，使用 STOPDEC 或者 STOPEMG

◆ STOPDEC 与 STOPEMG 的区别

STOPDEC：减速停止

STOPEMG：立即停止

二者速度曲线如下：



上图中，红色线为执行 STOPEMG 时的减速曲线，其速度立即减速至 0，无减速时间；绿色曲线为执行 STOPDEC 时的减速曲线，其速度不会立即减速至 0，而是有一个减速过程。

◆ 指令停止轴应用示例

让轴 0、1、2 同时进行单轴点位运动，1 秒后让三个轴减速停止，如下：

```
BASE 0, 1, 2
SVON
MOVE 40000, 20000, 35000
SLEEP 1000
STOPDEC
WAIT DONE
```

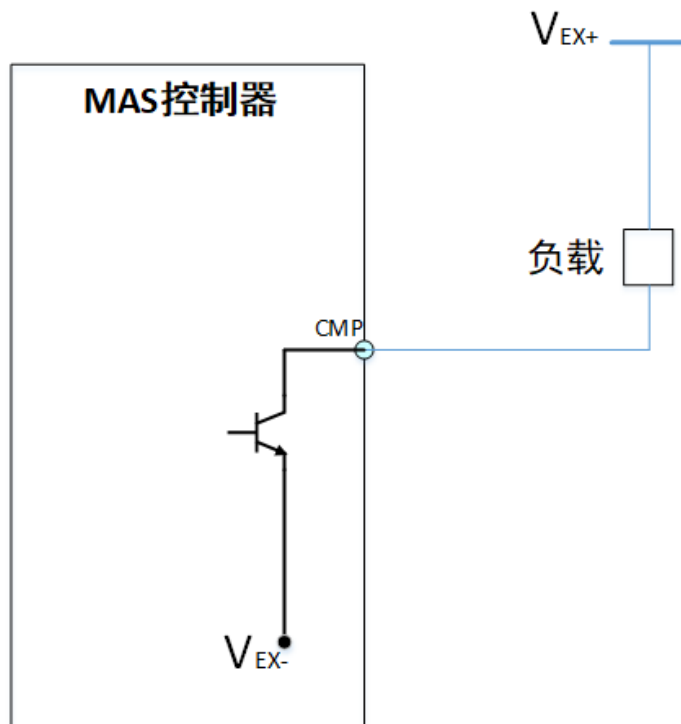
6.8 高速比较触发

使用高速比较触发功能的步骤如下：

1. 硬件接线，连接负载
2. 启用并配置高速触发的输出功能
3. 比较触发程序

6.8.1 比较触发输出的硬件接线

以使用 X 轴的高速比较触发功能为例，硬件接线如下：

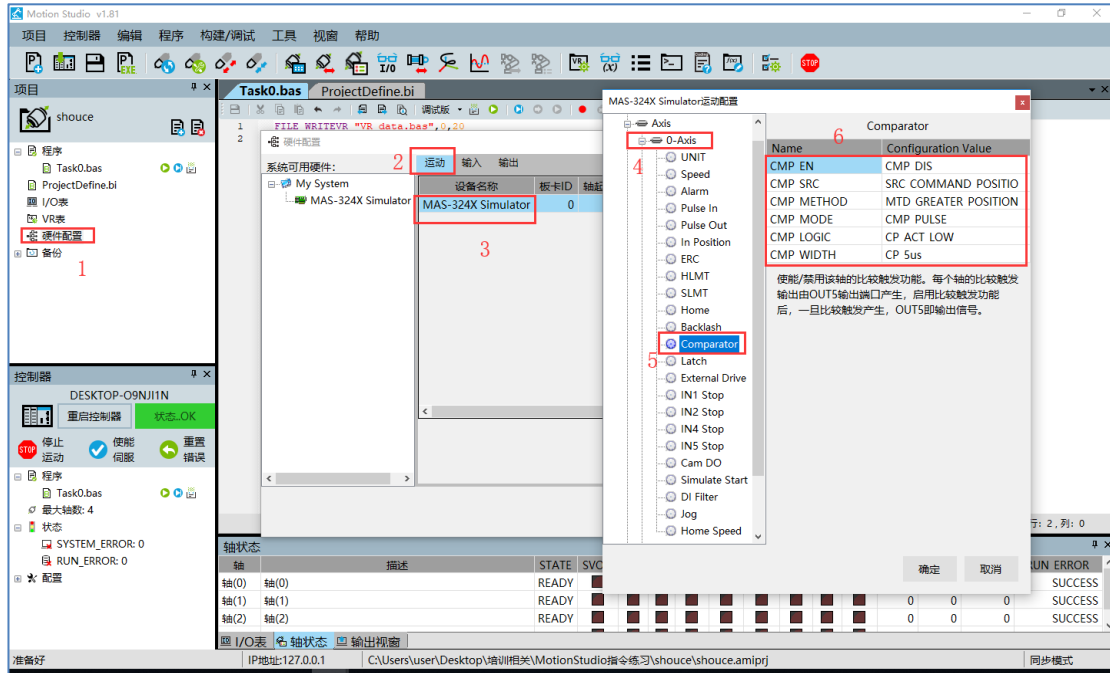


如上图所示，以线圈负载为例，线圈的一端与高电平 24V 信号连接，另一端接入 CMP 引脚。

6.8.2 启用比较触发输出

有两种方式启用比较触发功能，其一通过 Motion Studio 软件中的硬件配置栏启用，其二通过程序指令代码启用。

◆ 方法一：通过 Motion Studio 软件中的硬件配置栏



1. 打开硬件配置
2. 选择运动
3. 双击设备名称
4. 选择要操作的轴
5. 选择比较触发功能
6. 配置相关参数，完成后点击确定即可

通过以上操作，完成了对轴 0 的高速比较触发配置。

◆ 方法二：见下面的比较触发控制程序。

6.8.3 如何进行单轴比较触发

进行单轴比较触发的步骤如下：

1. 硬件接线 (请参考 6.8.1 章节)
2. 启用比较触发 (请参考 6.8.2 章节)
3. 单轴比较触发程序

接下来，以 0 轴为例，讲解如何编写单轴比较触发程序。

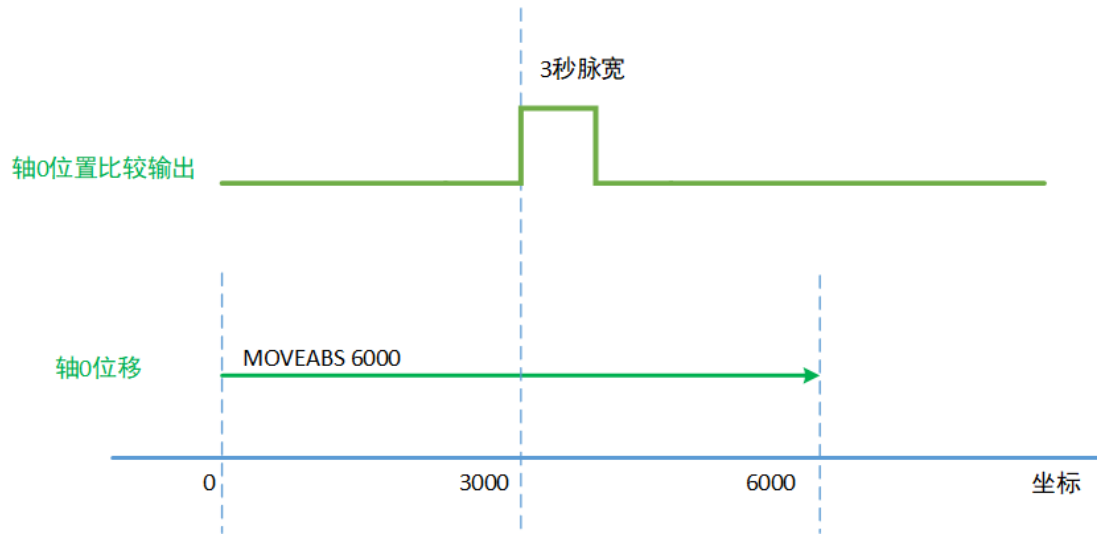
◆ 单轴比较触发控制程序示例

轴 0 从位置 0 运行至位置 6000，此过程中启用轴 0 的比较触发功能，设置比较触发输出为脉冲，输出脉冲宽度 3S，位置比较源为理论位置 3000，如下：

```
BASE 0
CMP_EN =1           '使能高速比较触发
CMP_LOGIC =0       '设置输出状态，比较触发时输出 ON，未触发时输出 OFF
CMP_METHOD=1       '设置比较方法
CMP_MODE =0        '设置输出模式为脉冲输出
CMP_WIDTH=3000000  '设置输出时的脉冲宽度
CMP_SRC =0         '设置比较数据源为轴理论位置
CMP AX(0), 3000    '使能高速比较触发
DPOS=0
MOVEABS 6000
WAIT DONE
```

更详细的指令说明见 BASIC 手册。

上面的代码中，输出如下图所示：



轴 0 的位置移至 3000 时，触发比较输出，输出脉冲宽度为 3000000 微秒。

6.8.4 如何进行多轴比较触发

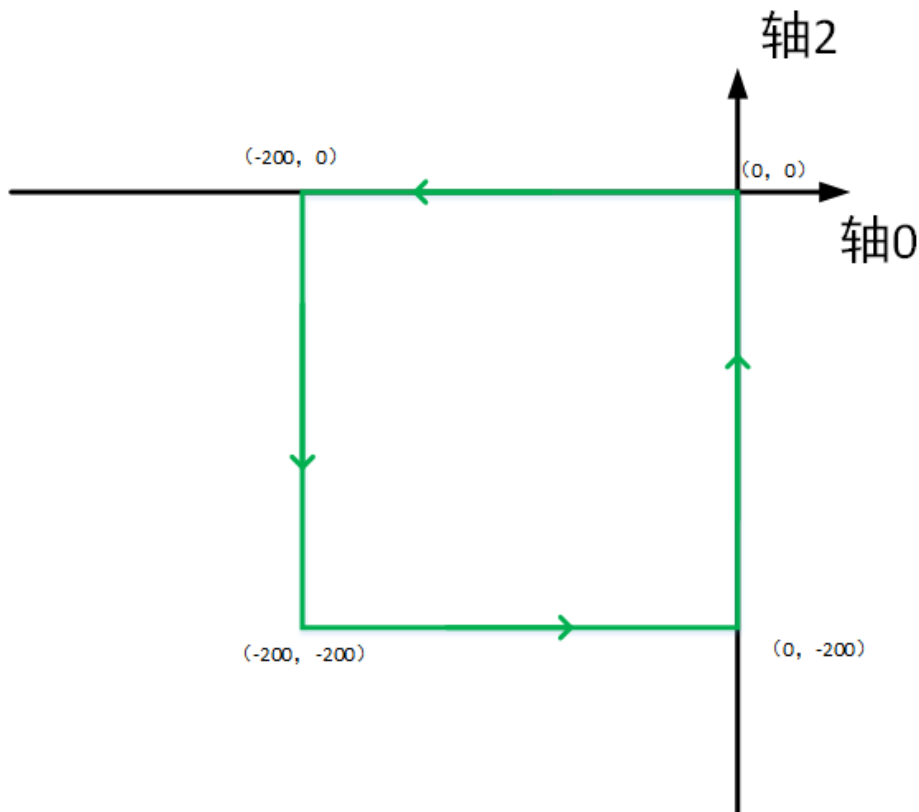
进行多轴比较触发的步骤如下：

1. 硬件接线 (请参考 6.8.1 章节)
2. 启用比较触发 (请参考 6.8.2 章节)
3. 多轴比较触发程序

接下来, 以 0、2 轴为例, 讲解如何编写多轴比较触发程序。

◆ 多轴比较触发控制程序示例

先看以下一段运行轨迹：



轴 0 和轴 2 进行连续插补 (LINEABS) , 从 (0, 0) 开始走完上述正方形轨迹, 最终回至 (0, 0) 位置。

在此轨迹中, 取 10 个点位进行比较触发, 10 个点位分别是: (-50,0), (-100,0), (-150,0), (-200,0), (-200,-50), (-200,-100), (-200,-150), (-200,-200), (-100,-200), (0,-100)。

每次比较触发发生时, 让轴 0 的比较触发输出 (CMP) 反转。

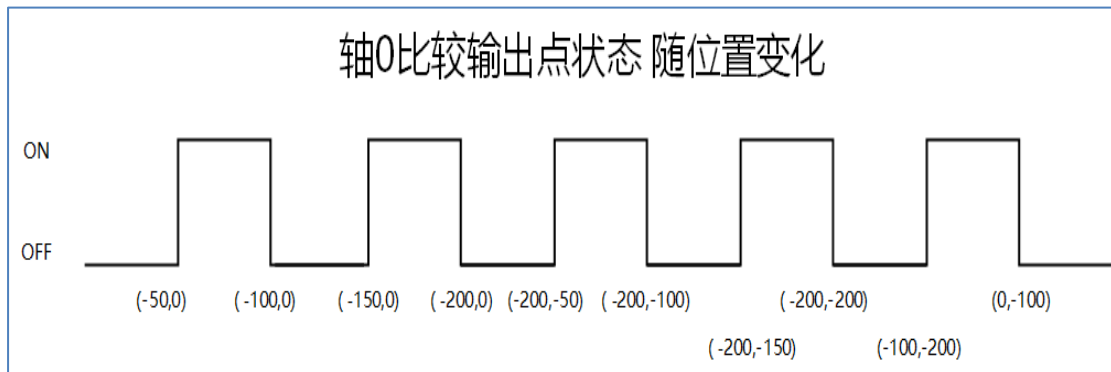
程序代码如下:

```

DIM AS DOUBLE CMP_XTable(20) ={-50, -100, -150, -200, -200, -200, -200, -200,
-100, 0}
DIM AS DOUBLE CMP_YTable(20)={ 0, 0, 0, 0, -50, -100, -150, -200, -200, -100}
BASE 0,2
GVL=1
GVH=5
LINEABS 0,0
WAIT DONE
MCMP_EN=1           '启用比较触发
RESETCMP AX(0)      '清除轴 0 比较触发标志
RESETCMP AX(2)      '清除轴 2 比较触发标志
MCMPSETDO 0        '手动设置轴 0 比较触发输出点状态变为 OFF
CMP AX(0),CMP_XTable(),10  '设置轴 0 多点比较触发位置值
CMP AX(2),CMP_YTable(),10  '设置轴 2 多点比较触发位置值
MCMP_MODE=1        '设定输出模式为反转当前输出
MCMP_CH=1          '设置双轴比较触发的是轴 0 的 OUT5 输出比较信号
MCMP_EMPTY=1       '自动清除多轴比较数据
MCMP_DEVIA=10      '多轴比较误差范围为 10
CMP_METHOD=1       '设置比较模式: 位置相同时比较触发
CMP_SRC=0          '设置比较数据源为轴理论位置
PATHRESET          '清除路径缓存
PATHBEGIN          '进入连续插补状态
    LINEABS -200,0
    LINEABS -200,-200
    LINEABS 0,-200
    LINEABS 0,0
PATHEND            '连续插补路径段结束指令, 退出连续状态
WAIT DONE

```

程序运行后，轴 0 比较输出点的状态随位置的变化如下图：



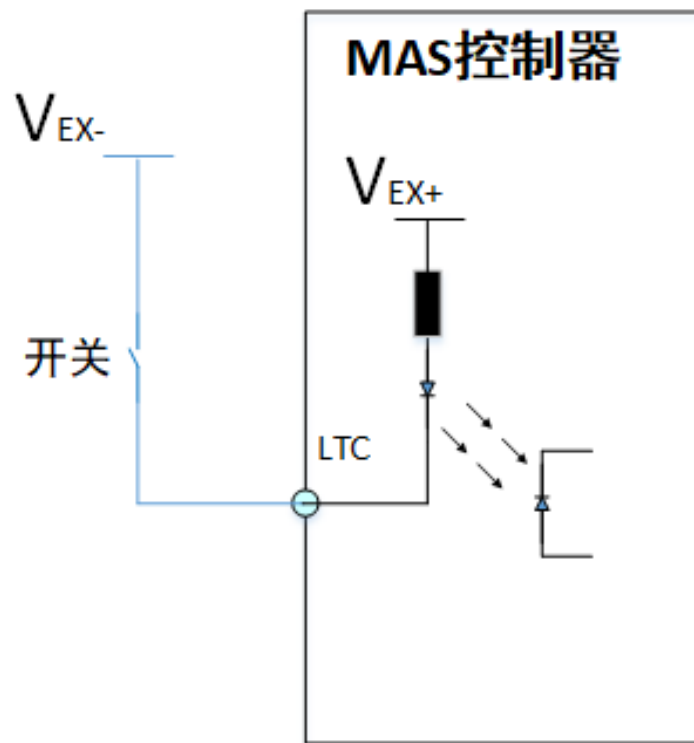
◆ **注意：**

编写比较触发程序时，比较数据（CMP）要放在比较使能（CMP_EN=1）之后。如果，比较数据（CMP）放在比较使能（CMP_EN=1）之前，会导致某些点位漏触发。

6.9 高速位置锁存

6.9.1 位置锁存硬件接线

以使用 X 轴的位置锁存功能为例，若输入器件为开关，其硬件接线如下：

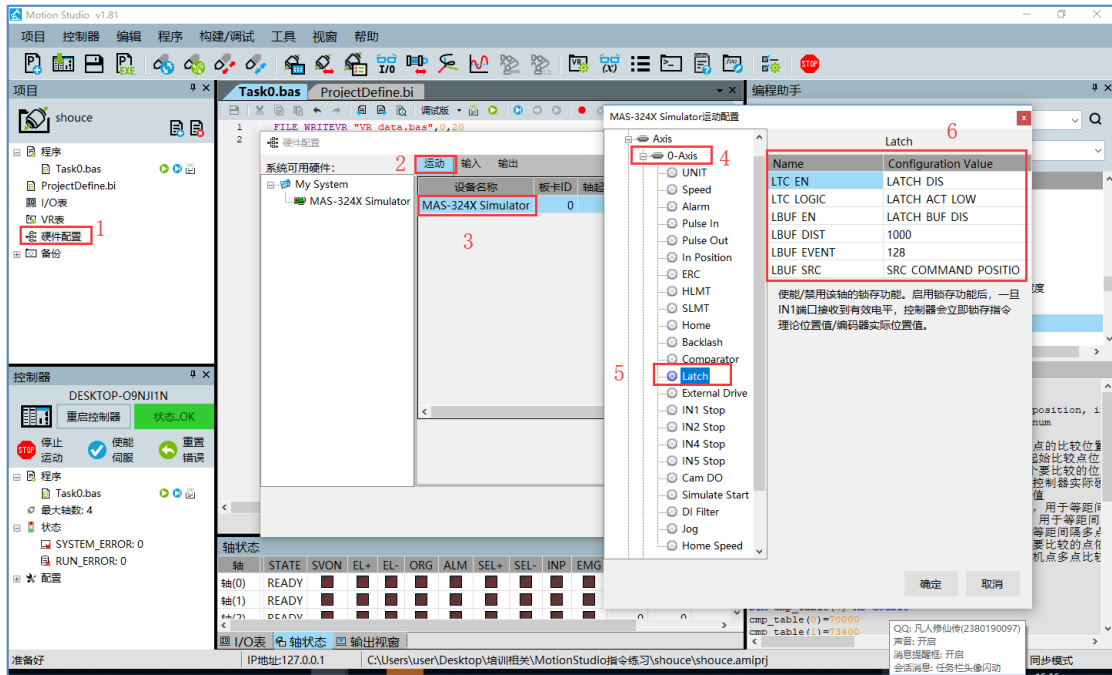


如上图所示，机械开关触点的一端与低电平信号连接，另一端接入 X_IN1/LTC 引脚。如此，开关闭合时，X_IN1/LTC 信号为 ON。

6.9.2 位置锁存功能启用

有两种方式启用位置锁存功能，其一通过 Motion Studio 软件中的硬件配置栏启用，其二通过程序指令代码启用。

◆ 方法一：通过 Motion Studio 软件中的硬件配置栏



如上图，步骤如下：

1. 打开硬件配置
2. 选择运动
3. 双击设备名称
4. 选择要启用锁存功能的轴
5. 选择锁存功能
6. 设置相关参数即可

◆ 方法二：见下面的位置锁存控制程序。

6.9.3 如何进行位置锁存

使用位置锁存功能的步骤如下：

1. 位置锁存硬件接线（请参考 6.9.1 章节）
2. 启用并配置高速位置锁存功能（请参考 6.9.2 章节）
3. 位置锁存程序

接下来，以 0 轴为例，讲解如何编写位置锁存程序。

◆ 位置锁存程序例程

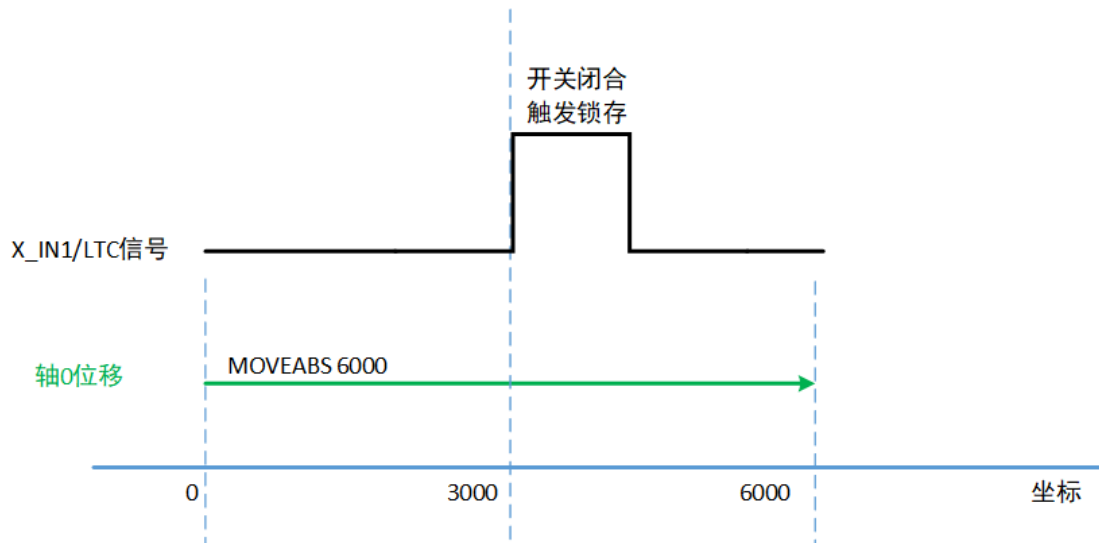
轴 0 在向位置 6000 运行的过程中，会触发某个机械开关动作，当开关动作时要记录此时轴 0 的位置。实现此要求需要如下操作：

硬件方面，要按照上述接线图，将开关信号与锁存输入连接。

程序代码如下：

```
DIM A AS DOUBLE
BASE 0
LTC_EN=1           '启用轴的锁存功能
LTC_LOGIC=0       '设置轴 0 锁存输入信号低电平有效
DPOS=0            '轴 0 的当前理论位置置 0
MOVEABS 6000
WAIT DONE
A=LDPOS(AX(0))    '获取轴 0 的锁存数据，并赋值给变量 A
RESETLTC AX(0)    '清除锁存数据、锁存标记
```

上例中，假设在轴 0 运行到 3000 位置的时候，触发了开关，如下：



开关闭合后，X_IN1/LTC 信号输入为 ON，触发锁存，则保存的理论位置就是 3000，最终 A 的值是 3000。

6.10 附：基本运行控制指令一览

指令	说明
BASE, R_BASE	指定要操作的轴
UNIT_NUM, UNIT_DENOM	用于设定用户单位
POUT_MODE	设定指令脉冲输出类型
EL_EN, EL_LOGIC, EL_MODE	硬极限参数设定
ALM_EN, ALM_LOGIC, ALM_MODE	伺服报警功能设定
VL, VH, GVL, GVH HOME_VL, HOME_VH JOG_VL, JOG_VH	设定相应运行模式下的“启动速度和运行速度”
ACC, DEC, GACC, GDEC HOME_ACC, HOME_DEC JOG_ACC, JOG_DEC	设定相应运行模式下的“加速度和减速度”
SVON, SVOFF	使能伺服、禁用伺服使能
JOGP, JOGN, MOVE, LINE, CIR	执行相应模式下的相对运动
MOVEABS, LINEABS, CIRABS	执行相应模式下的绝对运动
PATHBEGIN, PATHEND	连续插补运动
HOME_P, HOMEN	回原运动
WAIT DONE	等待运动完成
STOPDEC	减速停止轴
STOPEMG	紧急停止轴

第七章 流程控制

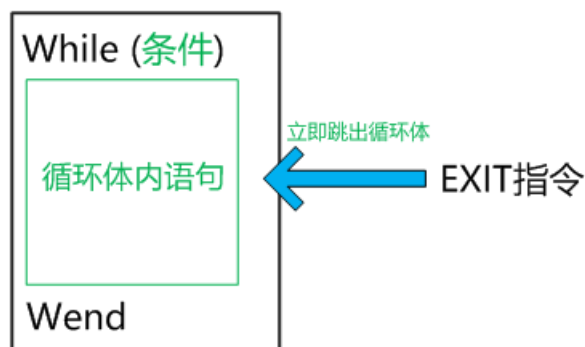
7.1 WHILE 循环

语法: WHILE condition
 commands
 WEND

参数:

condition: 条件表达式

commands: 循环体内的语句。



While 循环是最基本的循环模式，做如下几点使用说明。

1. 基本使用：条件满足，进入循环。条件不满足，执行完循环体内的语句后再跳出循环（而不是立即跳出循环）。

<pre> 1 '轴0一直循环做往返运动 2 BASE 0 3 SVON 4 WHILE 1 5 MOVE 1000 6 WAIT DONE 7 MOVE -1000 8 WAIT DONE 9 WEND 10 </pre>	<pre> 1 'VR(0)<5时，轴0一直做往返运动 2 BASE 0 3 SVON 4 WHILE (VR(0)<5) 5 MOVE 1000 6 WAIT DONE 7 MOVE -1000 8 WAIT DONE 9 WEND 10 </pre>
---	--

2. 退出循环：循环体内遇到 EXIT While 语句时，立即跳出循环。

```

1  '轴0一直循环做往返运动,直到VR(0)>100,立即跳出While循环
2  BASE 0
3  SVON
4  VR(0)=95
5  WHILE 1
6      VR(0)=VR(0)+1
7      MOVE 1000
8      WAIT DONE
9      IF(VR(0)>100) THEN
10         EXIT WHILE      '跳出while循环
11     END IF
12     MOVE -1000
13     WAIT DONE
14 WEND
15

```

3. 注意使用 Sleep：WHILE 循环体中没有任何等待相关的指令时，需要在循环体中加 Sleep 指令，以免程序高速一直循环执行，占用完 CPU 的资源，导致电脑卡住。

```

1  WHILE 1
2      IF(VR(0)=1) THEN
3          VR(0)=0
4          BASE 0
5          MOVEABS 1000|
6      END IF
7      SLEEP 10      '延时10ms
8  WEND
9

```

仅是避免占用太多CPU资源的话，延时1~10ms都没关系

7.2 IF 条件判断

语法: IF <condition1> THEN

 commands1

 [ELSEIF <condition2> THEN]

 commands2

 [ELSE commands3]

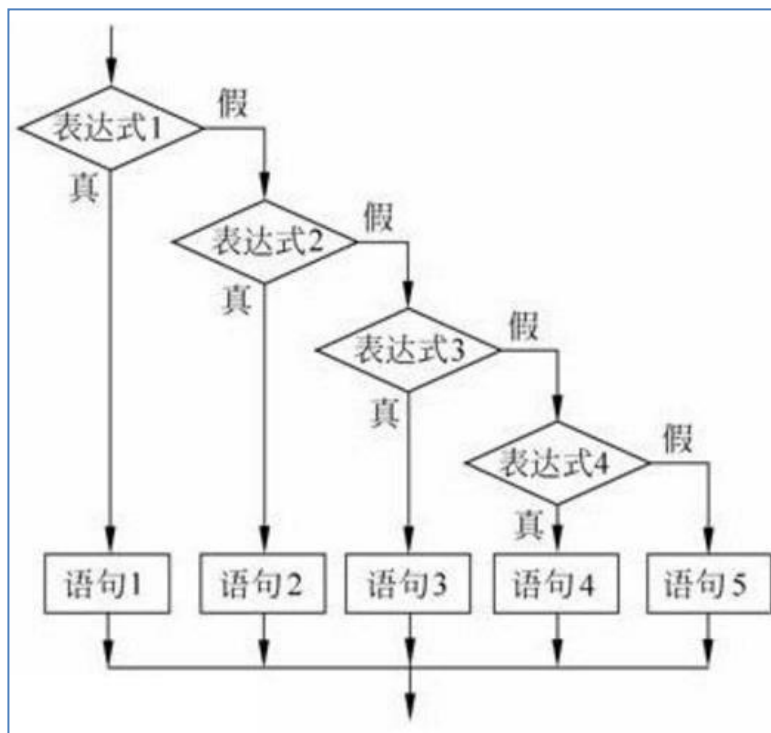
END IF

参数:

condition: 条件表达式

commands : 条件满足后需执行的语句

IF 条件判断指令用来判定所给定的条件是否满足, 根据判定的结果 (真或假) 决定执行哪块指令。下面举几个常用的例子说明如何使用 IF 条件判断语句。



例程 1: 只有 1 个判断条件的情况。

```
'条件下的语句只有1句可在一行写完，无需End IF
IF VR(0)=1 THEN VR(1)=VR(1)+1
```

```
'循环检测1个条件，条件下的语句有多个语句
WHILE 1
  IF VR(0)=1 THEN 'VR(0)为1时，执行该条件下的语句
    BASE 0
    MOVE 100
    WAIT DONE
    VR(0)=0      '将VR(0)清零，避免循环执行
  END IF
  SLEEP 10
WEND
```

例程 2: 有 2 个判断条件，且 2 个条件互斥的情况。

```
'判断，如果a>5，则DO0置1，否则置0
Dim a AS ULONG
IF a>5 THEN
  DOUT(0)=1
ELSE
  DOUT(0)=0
END IF
```

例程 3: 有 3 个或以上判断条件的情况。

```
'例：判断变量A是大于10，还是小于1，还是大于等于1且小于等于10
'如果还有其它条件，可以增加ELSEIF
Dim A AS ULONG
A=5
IF A>10 THEN      '判断条件1
  PRINT "A>10"    '即A大于10；打印：A > 10
ELSEIF A<1 THEN   '条件1不成立，即A小于10；此句为判断条件2
  PRINT "A<1"    '条件2成立，即A小于1，则打印：A < 1
ELSE PRINT "1<=A<=10" '条件2不成立，则打印1<=A <= 10
END IF
'运行的结果是 1<=A<=10
```

7.3 FOR 循环

语法: For `varname` = `startvalue` TO `endvalue` [STEP `stepvalue`]
 [`commands`]
 NEXT `varname`

参数:

`varname`: 循环体变量名

`startvalue`: 循环起始值

`endvalue`: 循环结束值

`stepvalue`: 循环变量的增量, 可选; 缺省时, 增量为1。增量也可以是小数或负数。增量为负数时, 循环起始值要大于循环结束值。

Commands: 循环体语句

注意: 该指令最多嵌套八层 (为程序可读性, 建议不要嵌套太多层)

For 循环是一种开界的循环体, 界限由循环体变量决定; 可用 EXIT 指令退出 For 循环。

如果循环变量小于循环结束值, 则执行指令块到 NEXT 时, 循环变量自动加一个增量, 再一次执行指令块; 当循环变量大于等于循环结束值时, 则停止循环;

下面举几个常用的例子说明如何使用 For 循环。

例程 1： 无需中途退出循环体的情况。

```
'循环增量为1，STEP可不写。  
DIM i AS ULONG  
FOR i=0 To 7  
    DOUT (i) =1 '将DO0~DO7全部置1  
NEXT i
```

```
'循环增量不为1  
DIM i AS ULONG  
FOR i=0 To 7 STEP 2  
    DOUT (i) =0 '将DO0、DO2、DO4、DO6置0  
NEXT i
```

例程 2： 需中途退出循环，使用 EXIT For 语句。

```
'轴0往返运动100次，每次往返运动前先判断VR(0)。  
'如果VR(0)值变为1，则退出For循环，不会执行剩余的往返运动。  
BASE 0  
DIM i AS ULONG  
FOR i=1 To 100  
    IF VR(0)=1 THEN  
        EXIT FOR  
    END IF  
    MOVE 100  
    WAIT DONE  
    MOVE -100  
    WAIT DONE  
NEXT i
```

判断是否要退出For循环

7.4 Case 条件判断

语法: Select Case `expression`

```
[ Case expressionlist
    [commands]
.....
[ Case expressionlist
    [commands]
[ Case Else ]
    [commands]
End Select
```

参数:

`expression`: 表达式, 最简单的情况为 1 个变量, 变量类型需为整型。

`expressionlist`: case 分支表达式条件

`commands`: case 条件满足的情况下, 要执行的语句。

注意: 各 case 分支表达式内容需互斥

下面举几个常用的例子说明如何使用 Select CASE。

例程 1：只判断表达式的其中几个条件。

```
'循环判断：VR(0)为1时，轴1运动。VR(0)为2时，轴0运动。
'因VR变量的数据类型为double,但Select Case后面的表达式值需为整型，需使用CINT指令转换。
'这里CASE分支并没有判断VR(0)所有情况，值判断了值为1和2的情况。这是允许的。
WHILE 1
  Select Case CINT (VR(0))
    Case 1
      BASE 1
      MOVE 1000
      WAIT DONE
      VR(0)=0 '将VR(0)赋0，否则一旦其它地方将VR(0)赋为1，将一直进入CASE 1
    Case 2
      BASE 0
      MOVE 1000
      WAIT DONE
      VR(0)=0 '将VR(0)赋0，否则一旦其它地方将VR(0)赋为2，将一直进入CASE 2
  End Select
  SLEEP 10
WEND
```

使用CINT将VR(0)的值转为整型

例程 2：判断表达式的所有条件，常使用 Case Else 来做 CASE 分支外的条件判断。

```
'判断变量choice的值，依据不同的结果执行相应的分支语句。
'Select case 语句相当于多个If判断语句组成，只是条件需互斥而已。
'下面程序运行的结果是"number is 5 to 10"
Dim choice As Long
choice=7
Select Case choice
Case 1
  Print "number is 1" 'choice为1时，打印number is 1
Case 2
  Print "number is 2" 'choice为2时，打印number is 2
Case 3, 4
  Print "number is 3 or 4" 'choice为3或4时，打印number is 3 or 4
Case 5 To 10
  Print "number is 5 to 10" 'choice为5~10时，打印number is 5 to 10
Case Else
  Print "number is outside the range" '非以上情况，打印number is outside the range
End Select
```

7.5 TMR 计时器类

TMR 是一个封装的类，用户可通过实例化该类，进行计时的应用。

7.5.1 TMR 类基本属性

TMR 类的基本属性与方法如下表：

成员	成员属性	说明
On(T_MS)	FUNCTION	T_MS, 计时器的计时时间 计时器未启动时，执行此方法启动计时器 计时器启动后，执行此方法刷新计时器 计时未完成，返回值为 FALSE 计时完成，返回值为 TRUE
Reset()	SUB	计时器复位

有关类的详细说明请见 BASIC 手册。

7.5.2 使用 TMR 类做计时应用

使用 TMR 类做计时的应用，有以下几个基本步骤：

1. 实例化一个计时器
2. 执行 On (T_MS)，启动计时器，设定计时时间
3. 执行 On (T_MS)，刷新计时器时间，获取返回值
4. 执行 Reset()，重置计时器

◆ 流程控制中的计时器使用示例

定义 VR(1)为启动按钮，按钮按下，延时 1S 进入启动程序，代码如下：

```

DIM T1 AS TMR           '实例化一个计时器
DIM StepFlag AS INTEGER '定义流控步骤
MS_LOOP(10)
  SELECT CASE StepFlag
  CASE 0
    IF MS_EDGER(VR(1)) THEN StepFlag=1 '启动按钮按下
  CASE 1
    IF T1.ON(1000)=TRUE THEN '启动并判断计时器的状态
      StepFlag=2           '计时完成，进入下一步
      T1.Reset()          '重置计时器
    END IF
  CASE 2
    '启动程序
    StepFlag=0
  CASE 3
    '
  END SELECT
MS_LEND

```

上述代码中，程序第一次进入 CASE 1 就启动了计时器；接下来，每次进入 CASE 1 都会刷新计时器的时间值，并返回状态。

MS_LOOP()类似于 WHILE 循环，用法见 7.6 章节。

7.6 MS_LOOP 循环

MS_LOOP 循环语句本质上类似于 WHILE，但在 MS_LOOP 循环内，可使用：

- 1) 上升沿的判断指令：MS_EDGER()
- 2) 下降沿的判断指令：MS_EDGEF()
- 3) 上升沿或下降沿的判断指令：MS_PULSE()
- 4) 定义好的流程控制步骤变量：MS_STEP0~MS_STEP10，无需用户再次定义

使用上述指令，用户可更方便、高效的做流程控制，实现按钮防呆、状态机等功能。

7.6.1 MS_LOOP 循环基本用法

语法：MS_LOOP(time)

commands

MS_LEND

相当于：WHILE 1

commands

SLEEP time

WEND

参数：

time：回圈中 Sleep 的时间，单位为 ms

commands：指令块

例程：

```
MS_LOOP (10)
  BASE 0
  MOVE 1000
  WAIT DONE
  IF (VR (5) =1) THEN MS_LEXIT '当VR (5) 等于1时，退出循环
MS_LEND
```

◆ 注意

MS_LOOP 循环语句无循环条件，若需要退出请执行 MS_LEXIT 指令，如上例所示。

相关指令的详细用法见 BASIC 手册。

7.6.1 MS_LOOP 循环用于流程控制

MS_LOOP 用于流程控制，需要结合 Select Case 语句。

◆ 程序示例

下面顺序流程动作中，用 MS_STEP1 来控制步骤，定义用户界面按钮如下：

VR(2)的值为 1 时，代表上位界面“回原点按钮”按下。

VR(50)的值为 1 时，代表回原点动作结束。

VR(3)的值为 1 时，代表上位界面“开始加工按钮”按下。

VR(51)的值为 1 时，代表加工程序动作结束。

程序示例如下：

```

SUB MyInit() '初始化子函数
END SUB
SUB MyHome() '回原点动作子函数
END SUB
SUB MyOrg() '去工作点子函数
END SUB
SUB MyRun() '加工动作子函数
END SUB
MS_LOOP(10)
  SELECT CASE CINT(MS_STEP1)
    CASE 0
      IF MS_PULSE(MS_STEP1) THEN MyInit()
      IF MS_EDGER(VR(2)) THEN MS_STEP1 = 1
    CASE 1
      IF MS_PULSE(MS_STEP1) THEN MyHome()
      IF VR(50) = 1 THEN MS_STEP1 = 2
    CASE 2
      IF MS_PULSE(MS_STEP1) THEN MyOrg()
      IF MS_EDGER(VR(3)) THEN MS_STEP1 = 3
    CASE 3
      IF MS_PULSE(MS_STEP1) THEN MyRun()
      IF VR(51) = 1 THEN MS_STEP1 = 2
  END SELECT
MS_LEND

```

从上述示例中，可以看出：

- 1) 通过对按钮、变量的上升沿或下降沿判断，实现了防呆处理，避免同一个动作被重复执行。
- 2) 通过 Select Case 语句，判断 MS_STEP1 的值做流程控制，实现状态机的功能，可避免多个动作同时运行的状况发生。
- 3) 代码非常简洁、层次分明，便于查看与维护。

第八章 子程序使用

8.1 子程序的使用步骤

子程序的使用有如下步骤：

- 1、声明子程序
 - 2、编写子程序
 - 3、调用子程序
- 声明子程序，需要使用 DECLARE 指令。若子程序写在调用它的程序之前，可不进行声明。
 - 定义编写子程序，使用 SUB 或者 FUNCTION 函数。
 - 调用子程序时，直接使用子程序名称即可。

8.2 SUB 和 FUNCTION 的区别

SUB 用于定义无返回值的子程序，而 FUNCTION 用于定义有返回值的子程序。

◆ 应用示例

例程 1: 定义 Test1()为无返回值的子函数，如下：

```
DECLARE SUB Test1()
```

例程 2: 定义 Test2()为返回 INTEGER 类型值的子函数，如下：

```
DECLARE FUNCTION Test2() AS INTEGER
```


8.3 SUB 子程序

SUB 子程序在使用时，可分为无形参与有形参两种。

子程序在调用时，直接使用子程序名即可。

8.3.1 SUB 无形参应用示例

- ◆ 定义一个输出 2 秒脉宽的子程序并调用，程序如下：

```
DECLARE SUB Do2sPulse '声明子程序名 Do2sPulse
Do2sPulse()          '调用子程序 Do2sPulse
SUB Do2sPulse        '子程序体
    DOUT(1)=1
    SLEEP 2000
    DOUT(1)=0
END SUB
```

8.3.2 SUB 有形参应用示例

- ◆ 定义一个子程序，可以操作指定轴移动固定距离，程序如下

```
DECLARE SUB AXIS_MOVE(ax AS USHORT) '声明子程序名 AXIS_MOVE
AXIS_MOVE(0)                        '调用子程序，让轴 0 正向移动 500
AXIS_MOVE(2)                        '调用子程序，让轴 2 正向移动 500
SUB AXIS_MOVE(ax AS USHORT)         '子程序体
    BASE ax
    MOVE 100
    WAIT DONE
END SUB
```

8.4 FUNCTION 子程序

8.4.1 FUNCTION 使用

类似于 SUB，FUNCTION 也可分为有形参和无形参两种；不同的是，FUNCTION 可以定义有返回值的子程序。

◆ FUNCTION 应用示例

定义一个子程序，实现输入一个值 N，N 的类型为 DOUBLE，进行如下运算 $N*(N+1)*(N+2)$ ，然后将结果写入 Result，程序如下：

```

DIM AS DOUBLE N, Result           '声明子程序
DECLARE FUNCTION MyMUL(num as DOUBLE) AS DOUBLE
Result=MyMUL(N)                   '调用子程序
FUNCTION MyMUL(num as DOUBLE) AS DOUBLE '子程序体
    MyMUL=num*(num+1)*(num+2)
END FUNCTION

```

8.4.2 FUNCTION 中返回值的三种写法

◆ 写法一、直接赋值给子程序名称

```

FUNCTION ADD1(a as DOUBLE, b as DOUBLE) as DOUBLE
    ADD1=a+b '（第 1 种，直接赋值给子程序名 ADD1）
END FUNCTION
VR(0)=ADD1(3, 4)

```

◆ 写法二、使用 RETURN

```

FUNCTION ADD2(a as DOUBLE, b as DOUBLE) as DOUBLE
    RETURN a+b '（第 2 种，使用 RETURN）
END FUNCTION
VR(0)=ADD2(3, 4)

```

◆ 写法三、使用 FUNCTION

```

FUNCTION ADD2(a as DOUBLE, b as DOUBLE) as DOUBLE
    FUNCTION=a+b '（第 3 种，使用 FUNCTION）
END FUNCTION
VR(0)=ADD2(3, 4)

```

第九章 多任务编程

9.1 TASK 相关指令一览

多任务编程，即多个 TASK 之间协同运行，实现特定功能。

在多任务编程中，会涉及下面的指令，详细的用法见 BASIC 手册

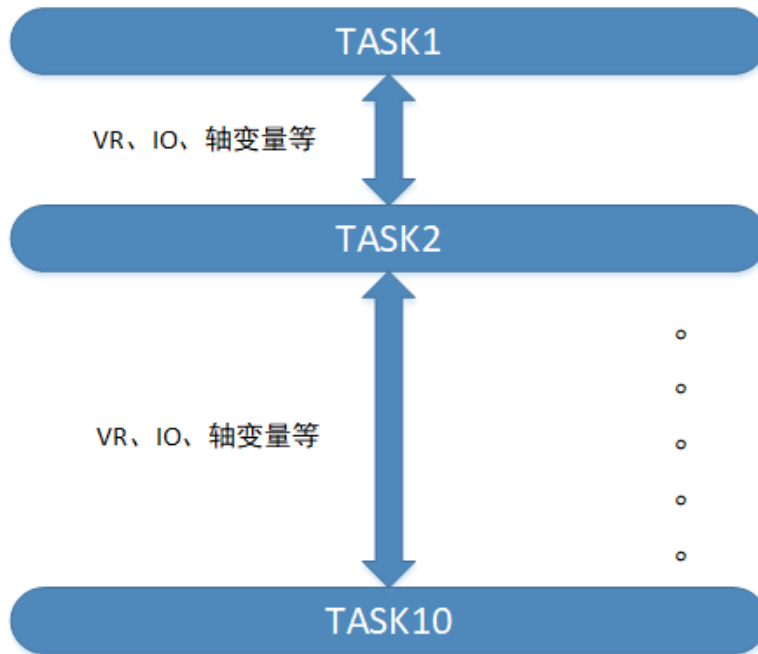
指令	说明
RUN_TASK	运行指定的 Task
STOP_TASK	停止运行指定的 Task
STOP_ALL	停止运行所有 Task，并停止所有轴的运动
Task_Status	读取 Task 的状态
Task_Pause	执行相应模式下的相对运动
Task_Resume	恢复运行对应编号的 Task 暂停程序

◆ 注意：

以上指令只能对“主 TASK 名称”进行操作，即接下来讲解的多个 TASK 之间的协同运行，都是针对“主 TASK 名称”名称进行的讲解。

9.2 多个 TASK 运行时的关系

多个 TASK 同时运行的关系如下图：



- 每个 TASK 是一个独立的进程
- 多个 TASK 相互之间是并行运行的状态；
- VR 变量、轴变量、IO 变量等全局变量在各个 TASK 之间共享操作
- 一个 TASK 可以通过 TASK 指令暂停、恢复、启动、停止其它的 TASK

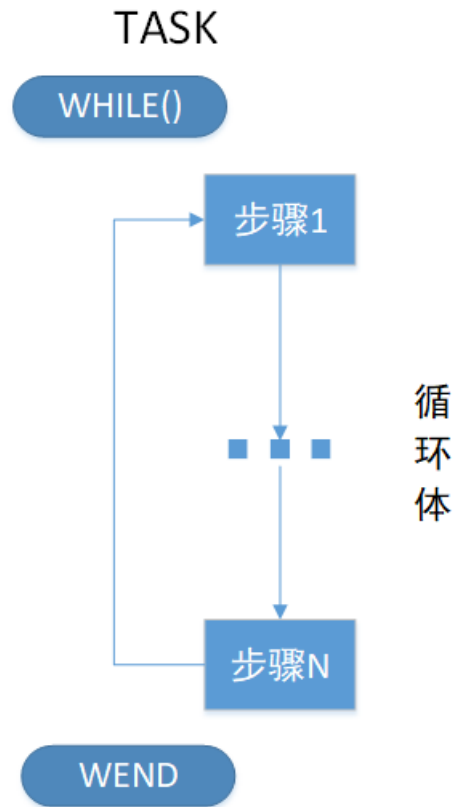
◆ **注意：**

最多只能同时添加 10 个 TASK 任务。

9.3 如何用 TASK 实现循环任务

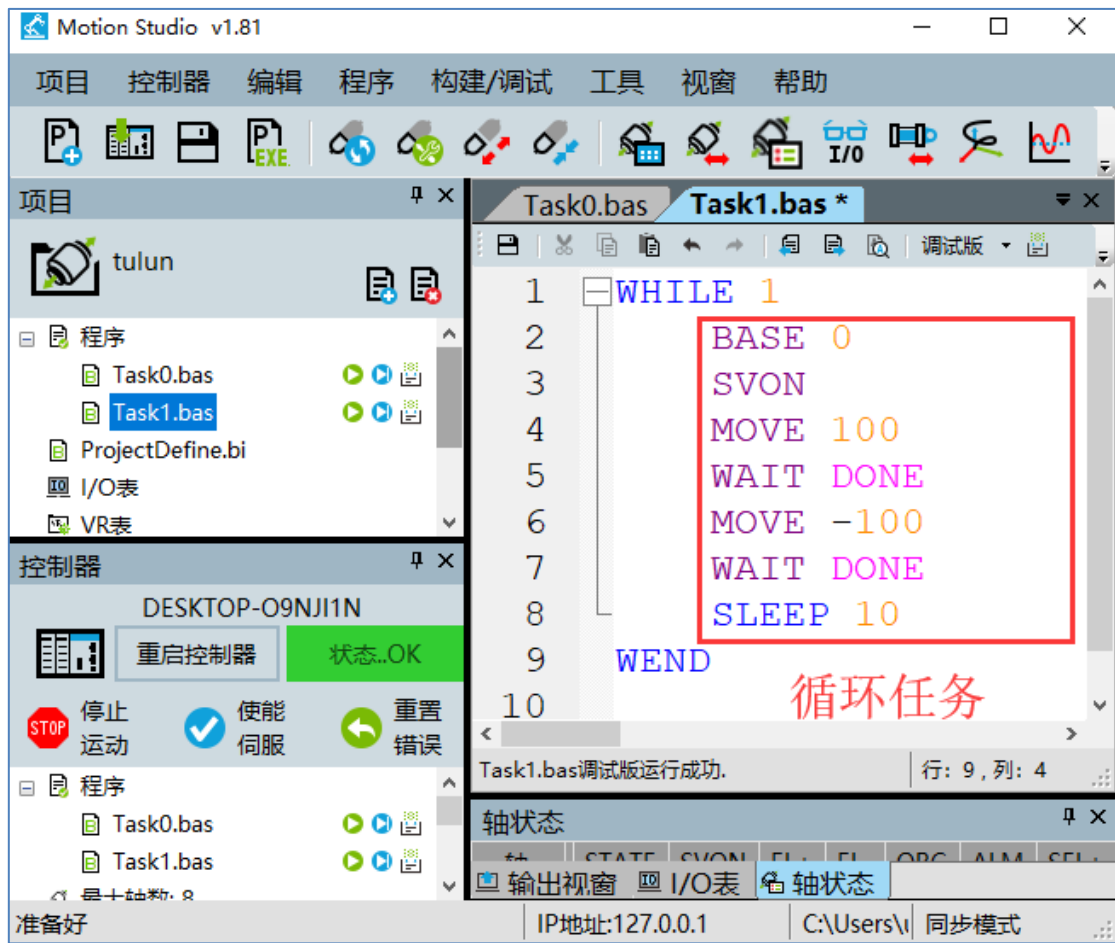
◆ 循环任务实现方法

实现循环任务，需要在 TASK 内使用 WHILE 语句。



如上图所示，在 TASK 任务内，添加 WHILE 循环体，循环体内的任务即循环任务。

◆ 循环任务实现示例



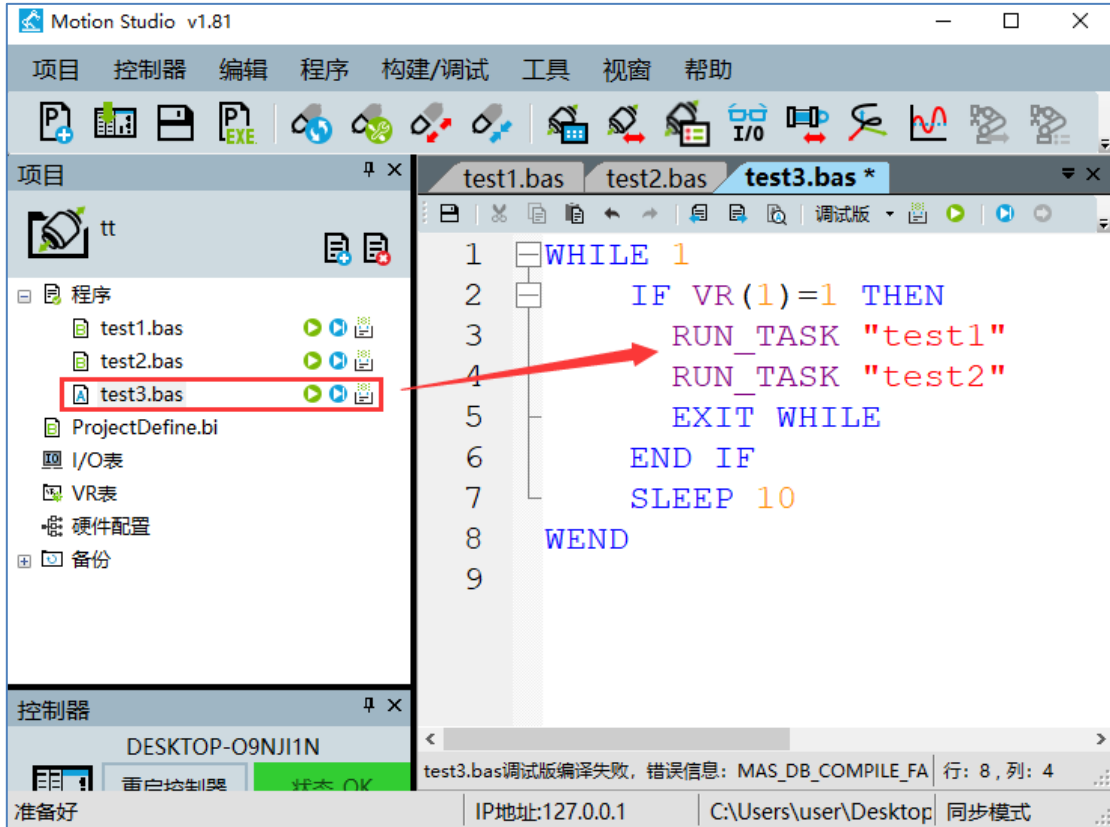
上图中，Task1.bas 程序的 WHILE 循环体内，写入循环内容即可形成循环任务。

执行程序，轴 0 会先进 100，然后后退 100，并一直循环执行该往复动作。

9.4 如何用一个 Task 启动另一个 Task

使用 RUN_TASK 指令，可启动处于未运行状态的 TASK。

◆ TASK 调用示例：



• 上图示例说明：

用户创建了 3 个 TASK，分别为 test1.bas，test2.bas，test3.bas。

test1.bas 和 test2.bas 的类型为 Normal，不会自启动。

test3.bas 的类型为 Auto-Run，为自启动任务。

以 VR(1)作为另外两个 TASK 的启动条件，当 VR(1)=1 时，通过 test3.bas 启动 test1.bas，test2.bas 这两个 TASK。

• 示例实现 TASK 调用的方法：

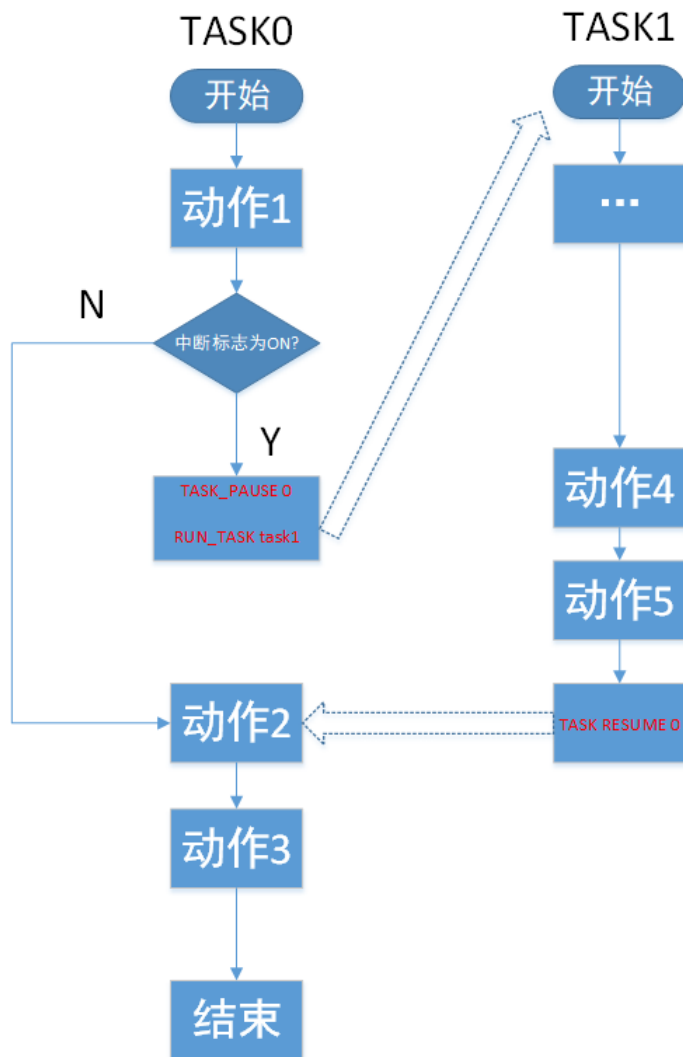
按上图中所示，可在 test3.bas 循环体内加入一个 IF 判断语句，如此，便可以不断扫描并判断 VR(1)的状态。当 VR(1)=1 条件满足后，便可使用 RUN_TASK 指令，启用 test1.bas 和 test2.bas，然后执行 EXIT 跳出循环。

9.5 如何用 TASK 实现中断功能

◆ 中断功能实现方法

实现中断功能，需要使用指令 `Task_Pause` 与 `Task_Resume`。

以下的流程图说明如何实现中断功能。

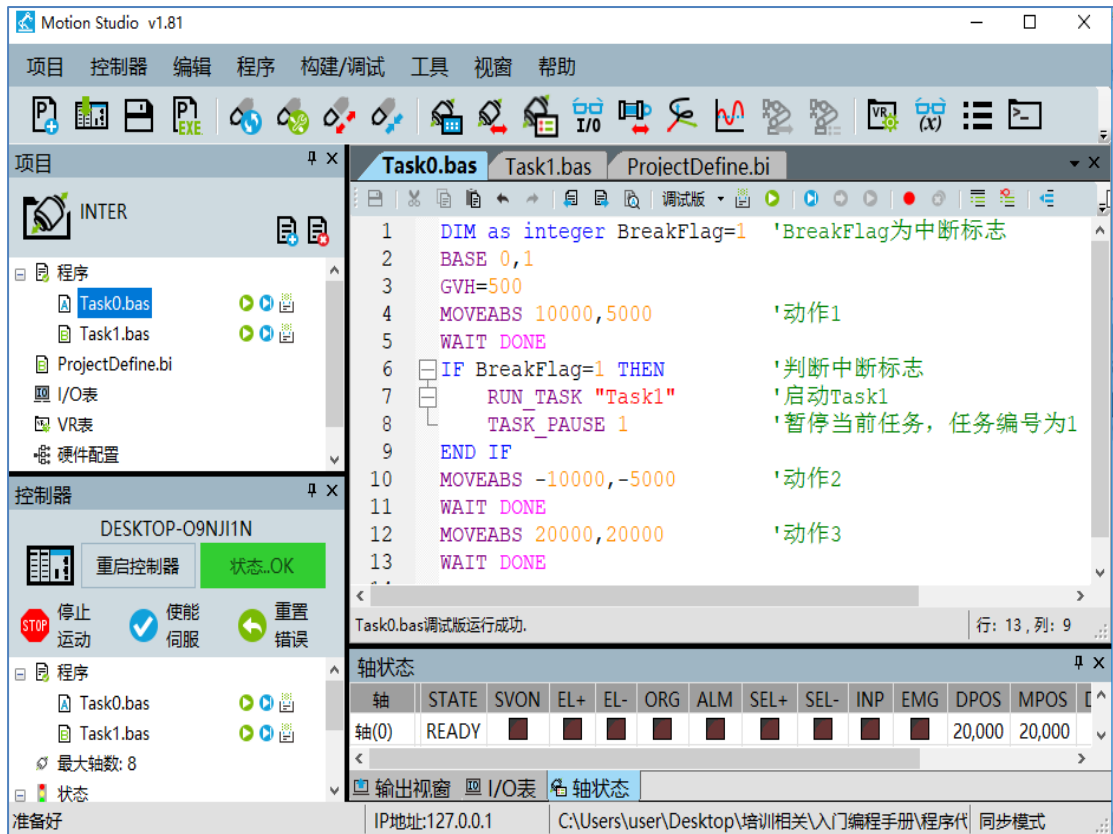


1. TASK0 的动作 1 完成后，若判断中断标志为 ON，则执行 `TASK_PAUSE` 指令暂停 TASK0 的运行，同时通过 `RUN_TASK` 指令启动 TASK1。
2. TASK1 启动，动作 5 完成后，执行指令 `TASK_RESUME` 恢复 TASK0 的运行。于是，TASK0 会继续向下运行动作 2，直至结束。
3. 由此，实现了对 TASK0 的中断功能。

◆ 中断功能实现示例

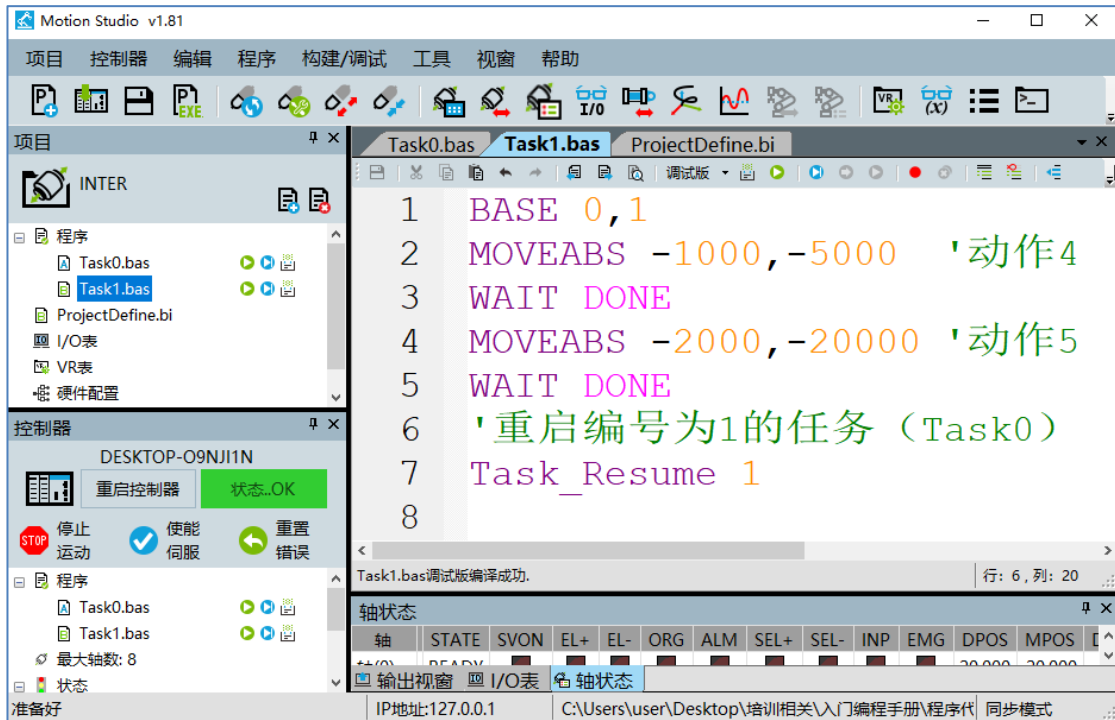
以上面的流程图为例，进行编程。

- 在 TASK0 内:



程序代码中，为方便讲解，直接令中断标志值为 1。实际应用要结合实际情况，处理中断标志的值。

- 在 TASK1 内



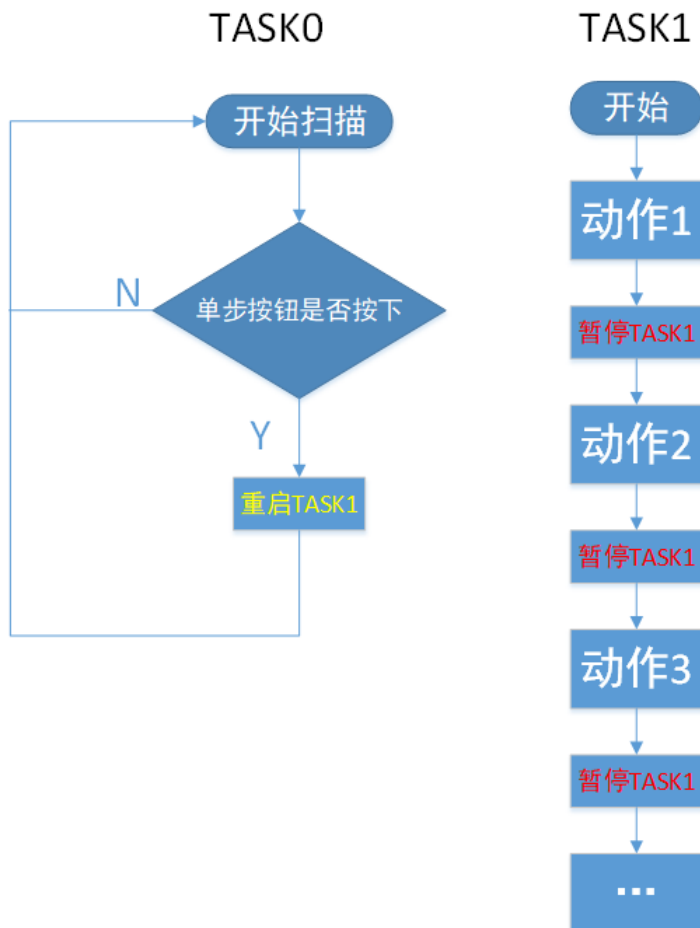
程序中，TASK0 类型为 Auto_Run，为自启动任务；TASK1 类型设为 Normal，不会自启动。

9.6 如何用 TASK 指令实现单步运行

◆ 单步运行实现方法

单步运行，需要使用指令 Task_Pause 与 Task_Resume。

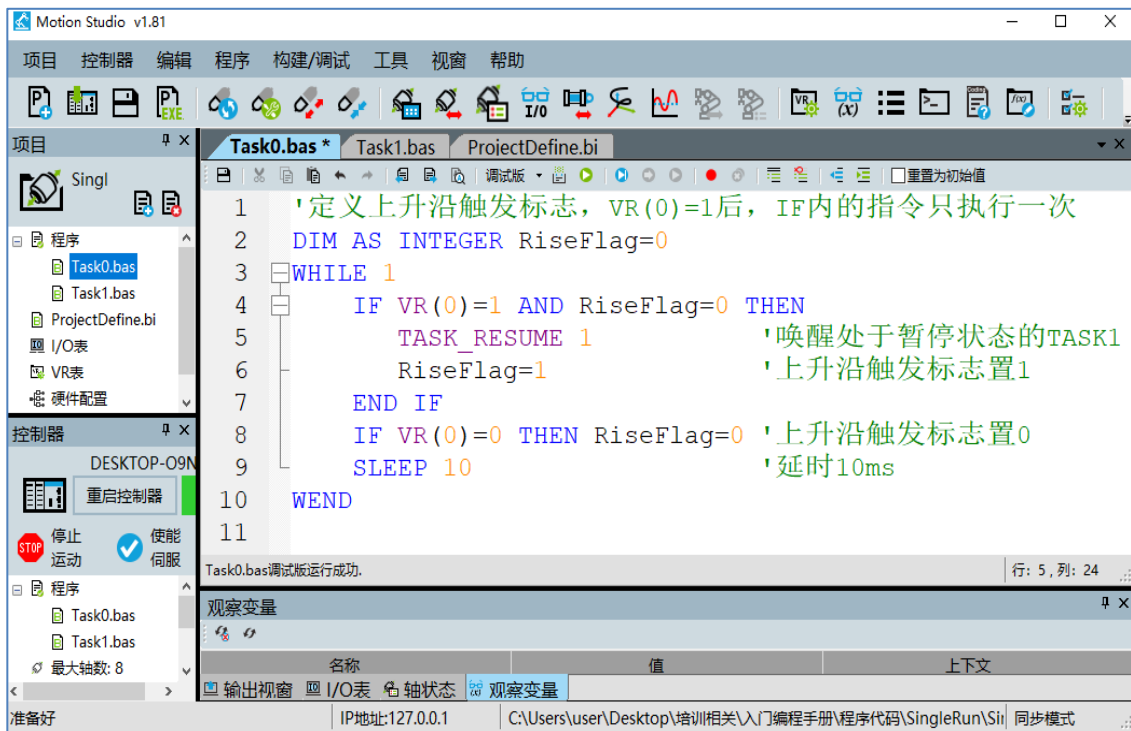
以下的流程图说明如何实现单步运行。



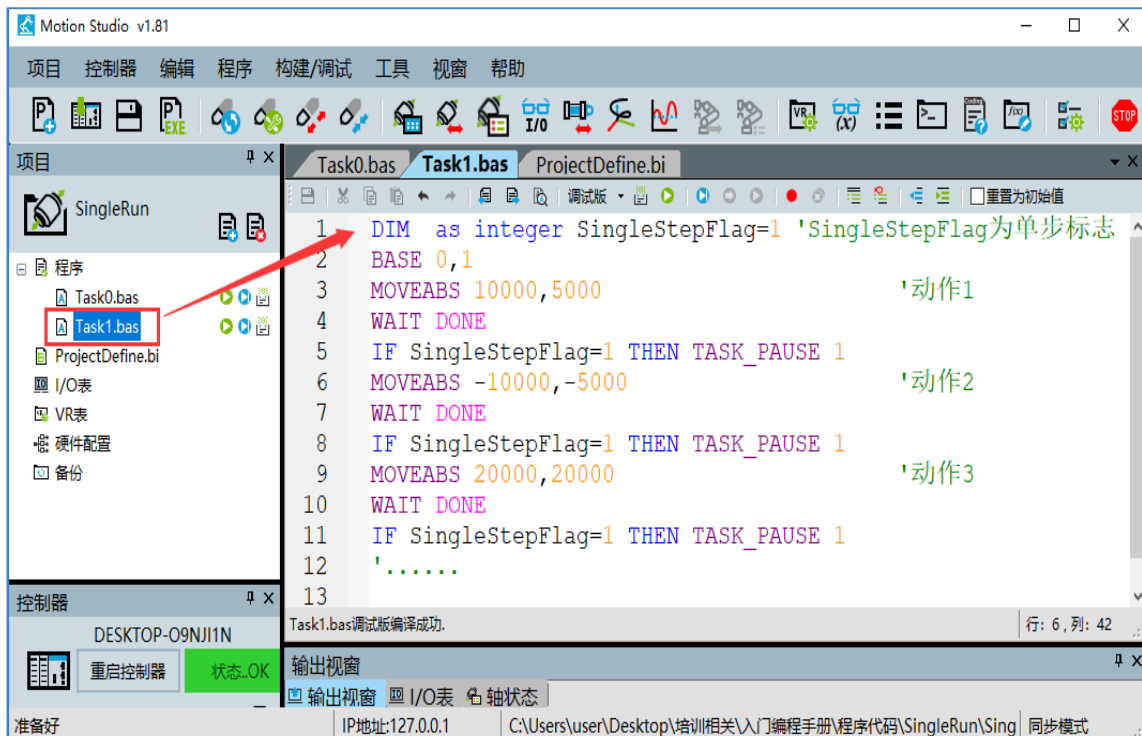
1. TASK1 内，在每一步动作后都添加暂停任务指令。则每次执行完成一个动作后，TASK1 被暂停。
2. TASK0 内，用循环任务实时扫描“单步按钮”的状态。当每次按下“单步按钮”后，就让 TASK1 任务重启。
3. 如此，TASK1 每执行一个动作后，就会被暂停；在按下“单步按钮”后，TASK1 又被重启，继续执行完成下一个动作后，再次被暂停.....如此，即可实现单步运行。

◆ 单步运行实现示例

以上面的流程图为例，进行编程。在 TASK0 内：



在 TASK1 内：



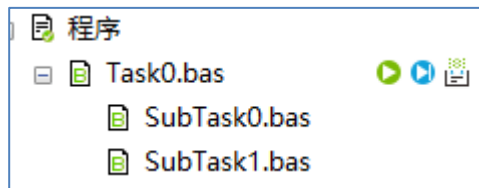
9.7 子 TASK 的使用

关于子 TASK 如何建立，详见 2.2.1 章节的步骤五，不再赘述。

9.7.1 子 TASK 与主 TASK 的关系

当一个 TASK 的代码量非常庞大时，查看起来会很吃力。

如果为这个 TASK 建立子 TASK，将主 TASK 中的某部分代码抽出来，放在子 TASK 中，每个子 TASK 执行相应的任务。这样就能将复杂的 TASK 变得层次分明，这就是子 TASK 的概念。



关于主 TASK 与子 TASK，有以下两点：

- a) 子 TASK 作为主 TASK 的一部分来执行。

上图中，运行 Task0 时，SubTask1、SubTask0 会被先执行。

- b) TASK 执行时，按照从下至上的机制，会先执行“子 TASK”里面的代码；“子 TASK”执行完成后，接着执行“主 TASK”里面的代码。

所以，上图中，运行 Task0 时，TASK 执行顺序是 SubTask1、SubTask0、Task0。

9.7.2 子 TASK 应用

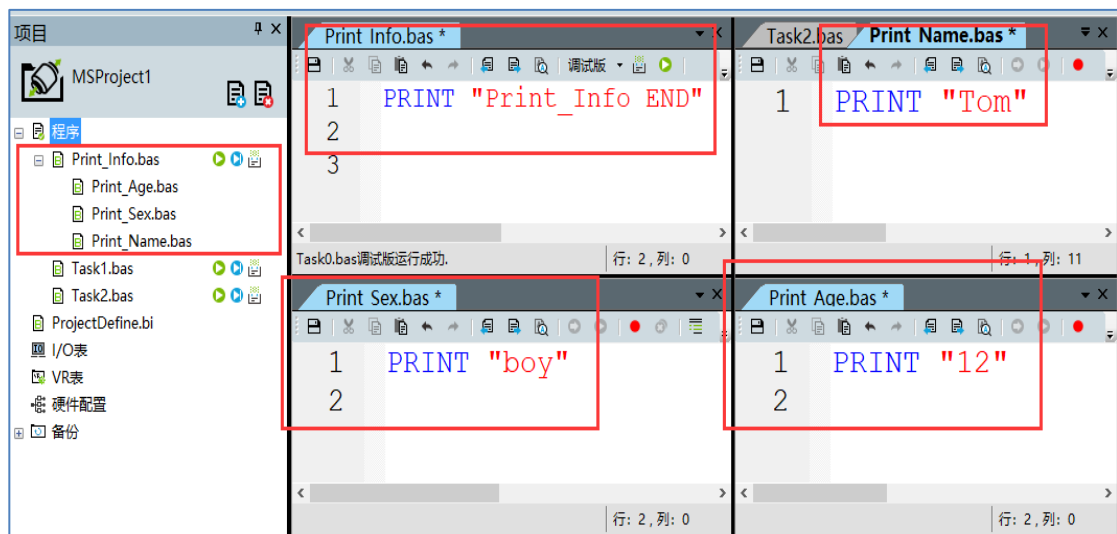
合理运用子 TASK，可使程序层次分明，易于查看与维护。

接下来，用一个简单的程序说明子 TASK 的应用。

◆ 子 TASK 应用示例

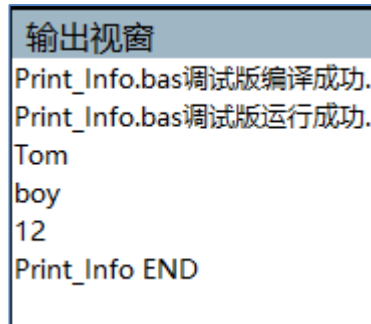
一个 TASK 任务，需要打印出人员的姓名、性别、年龄信息。为了让这个 TASK 的代码看起来层次分明，可按照如下方式编写程序：

如下图：



- 1) 新建一个主 TASK，命名为：Print_Info。
在主 TASK 中写入代码：PRINT "Print_Info END"，表示程序执行结束。
- 2) 建立第一个子程序，命名为：Print_Age。
在该子 TASK 中写入代码：PRINT "12"，即打印出年龄信息。
- 3) 建立第二个子程序，命名为：Print_Sex。
在该子 TASK 中写入代码：PRINT "boy"，即打印出性别信息。
- 4) 建立第三个子程序，命名为 Print_Name。
在该子 TASK 中写入代码：PRINT "Tom"，即打印出姓名信息。

执行 Print_Info 这个 TASK，执行完成后，输出视窗的结果如下：



```
输出视窗
Print_Info.bas调试版编译成功.
Print_Info.bas调试版运行成功.
Tom
boy
12
Print_Info END
```

可知，程序依次打印出了姓名、性别、年龄及程序结束的信息。

通过以上例程，可以看出：

使用子 TASK，并对子 TASK 合理命名，可让 TASK 程序结构层次分明，易于查看、维护。

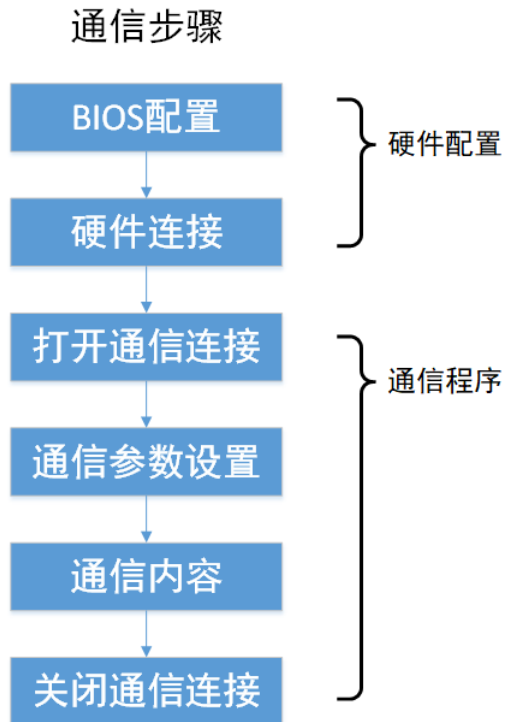
第十章 与外部通信

10.1 外部通信基本步骤

在 Motion Studio 编程环境中，常用以下方式与外部进行通信：

- 1、串口通信
- 2、TCP/IP 通信

进行外部通信，基本的步骤（BIOS 是否要设置根据实际情况而定）如下图：



10.2 串口通信

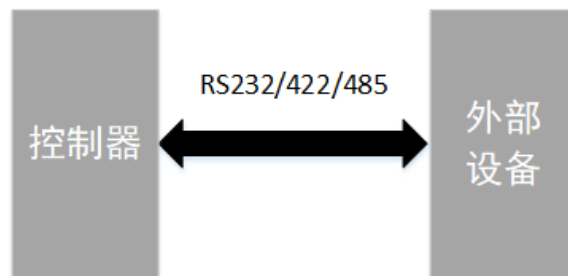
10.2.1 串口通信硬件配置与连接

◆ 配置串口通信模式

串口通信模式有 RS232、RS-422、RS-485 等通信模式，用户需要根据实际硬件及通信需求，正确地设定串口的通信模式。

◆ 串口通信硬件连接

串口通信方式设定完成后，即可进行硬件连接。



根据实际硬件的串口引脚分布，完成通信接线后，即可进行串口通信。

10.2.2 如何进行串口通信

进行串口通信的步骤如下：

1. 串口通信配置与硬件连接（请参考 10.2.1 章节）
2. 串口通信程序

接下来，讲解如何编写串口通信程序。

◆ 串口通信程序的基本架构

串口通信程序包含以下 4 个基本部分：

- a) 打开串口
- b) 通信参数配置
- c) 发送与接收程序
- d) 关闭串口

◆ 串口通信指令一览

指令	说明
COM_OPEN	打开串口
COM_SET	设置串口通讯参数
COM_READSTREAM	串口自由协议读操作，通过串口读数据
COM_WriteStream	串口自由协议写操作，通过串口写数据
COM_ResetBuf	清除串口缓存区数据
COM_CLOSE	关闭串口

各指令用法详见 BASIC 手册。

◆ 串口通信程序例程

通信要求：

打开串口 2，设置串口波特率 9600，校验位无，停止位 1 位，数据位 8 位。

通信连接后，发送“OK”出去。

然后不断地接收数据，每次接收 3 个字符，当接收字符串不为空时，在 Motion Studio 的输出视窗中显示接收的字符。

当接收的字符为“END”时，结束通信。

程序如下：

```

DIM ReadStr AS STRING
DIM WriteStr AS STRING="OK"
DIM COMFlag AS INTEGER=1
COM_Open 2                '打开串口 2
COM_SET 2,9600,0,1,8      '设置通信参数
COM_WriteStream 2,WriteStr,2
                          '向串口 2 发送 WriteStr 中的数据，即发送“OK”

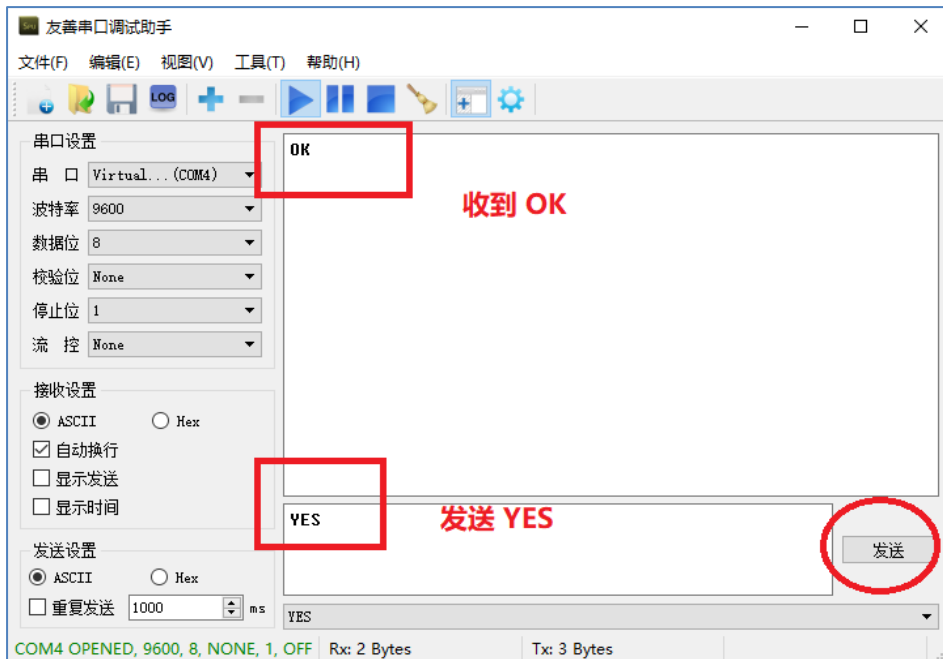
WHILE COMFlag
  COM_ReadStream 2,ReadStr,3,30
                          '从串口 2 中接收 3 个字节，存入 ReadStr 中
  IF ReadStr="END" THEN COMFlag=0
                          '判断接收到的内容，如果为 END，则结束通信循环
  IF ReadStr<>" " THEN PRINT ReadStr
                          '判断接收到的内容，如果不为空，则打印出来

  COM_ResetBuf 2
                          '清空串口 2 的缓存区数据

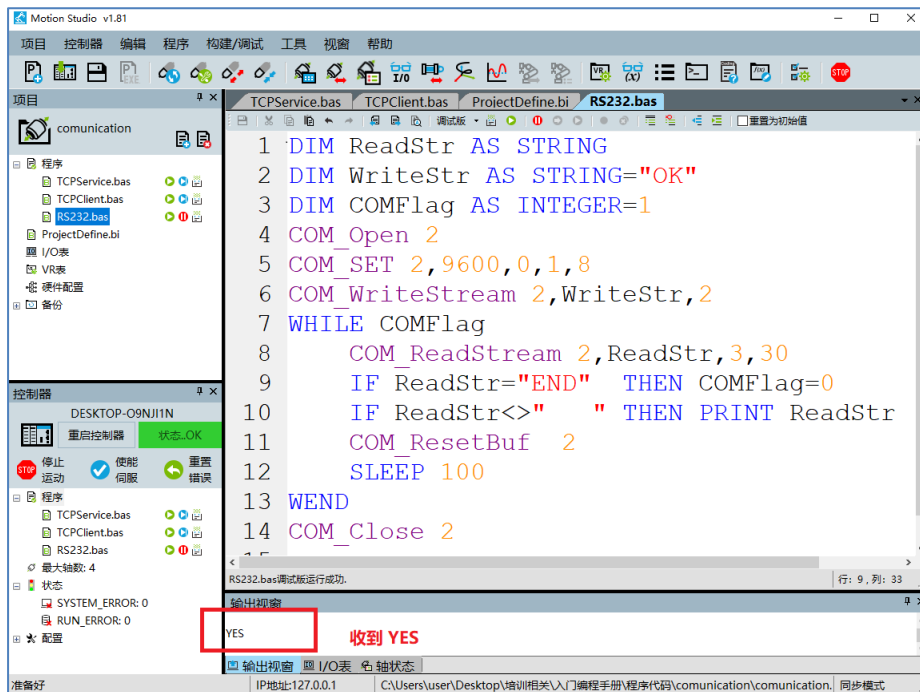
  SLEEP 100
WEND
COM_Close 2                '关闭串口 2

```

在此以调试工具作为通信对象, Motion Studio 程序运行后, 通信连接后, 调试工具会收到“OK”如下图。



然后, 让调试工具发出“YES”, Motion Studio 程序会接收到“YES”, 并在输出视窗中显示出来, 运行结果如下图。



当调试工具发出“END”后, Motion Studio 程序运行结束, 通信结束。

10.3 TCP/IP 通信

10.3.1 TCP/IP 通信硬件连接与配置



如上图，通过标准网线将通信双方连接后，即完成硬件上的连接。

根据实际需求，设定通信双方的 IP 地址。

◆ **注意：**

通信双方的 IP 地址必须在同一个段内，但不能重复。

比如，双方地址可设为 192.168.1.1 与 192.168.1.2。

10.3.2 如何进行 TCP/IP 通信

进行 TCP/IP 通信的步骤如下：

1. TCP/IP 硬件连接与配置（请参考 10.3.1 章节）
2. TCP/IP 通信程序

接下来，讲解如何编写 TCP/IP 通信程序。

◆ TCP/IP 通信程序基本框架

网络连接后，即可进行 TCP/IP 通信。TCP/IP 通信程序包含以下 4 个基本部分：

1. 打开一个 TCP 通讯连接
2. 等待 TCP 连接完成
3. 发送与接收程序
4. 关闭 TCP 通讯连接

◆ TCP/IP 通信指令一览

指令	说明
TCP_OPEN	打开一个 TCP 通信连接
TCP_WAIT	等待 TCP 连接完成
TCP_STATUS	检查 TCP 通讯连接状态
TCP_Check	获取 TCP 通讯接收缓存区的字符个数
TCP_ReadSTR	TCP/IP 自由协议读操作, 控制器从 TCP 接收缓存区读字符串指令
TCP_WriteSTR	TCP/IP 自由协议写操作, 控制器发送出字符串指令
TCP_Read	TCP/IP 自由协议读操作, 控制器接收数据
TCP_Write	TCP/IP 自由协议写操作, 控制器发送出数据
TCP_ReadVR	TCP/IP 自由协议读操作, 控制器用 VR 变量接收数据
TCP_WriteVR	TCP/IP 自由协议写操作, 控制器把 VR 变量中的数据发送出去
TCP_ResetBuf	清除 TCP 缓存区数据
TCP_CLOSE	指定 TCP 通讯编号, 关闭对应 TCP 通信端口

各指令用法详见 BASIC 手册。

◆ TCP/IP 客户端通信例程

创建一个客户端连接，对接 IP 为"127.0.0.1"的服务器，连接完成后，向外发送 "I'm Ready" 信号，然后实时读取缓存区的数据。

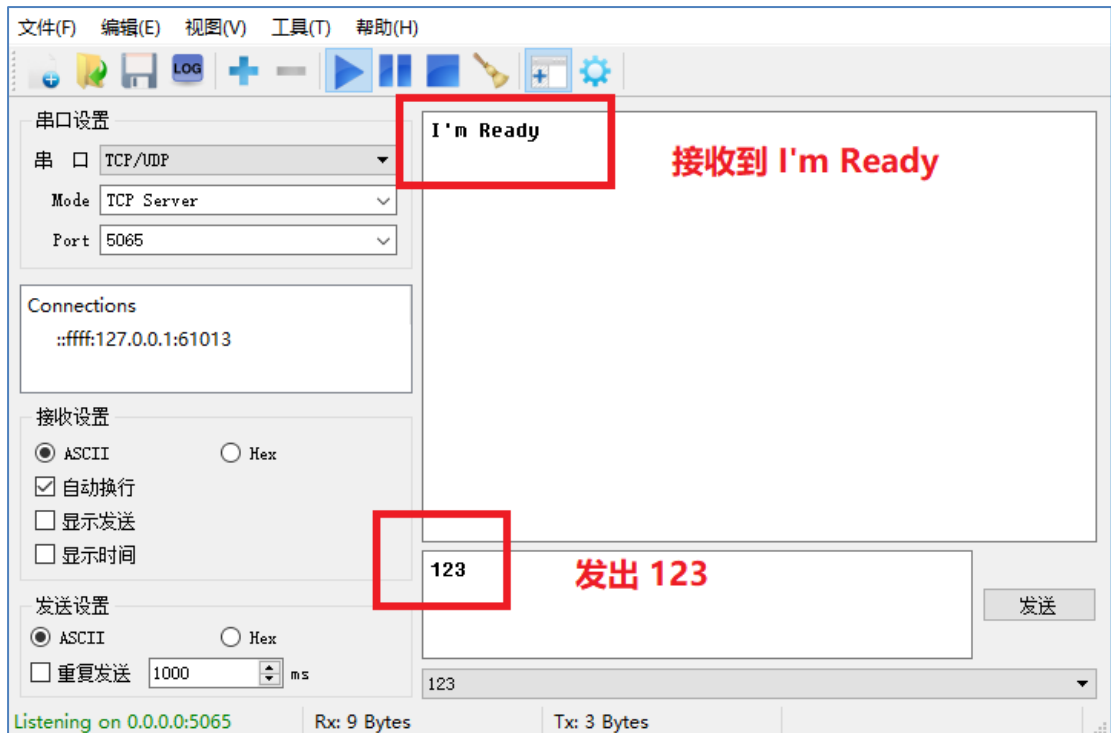
并以 VR(0)作为通信条件，当 VR(0)=1 时，结束通信，程序如下：

```

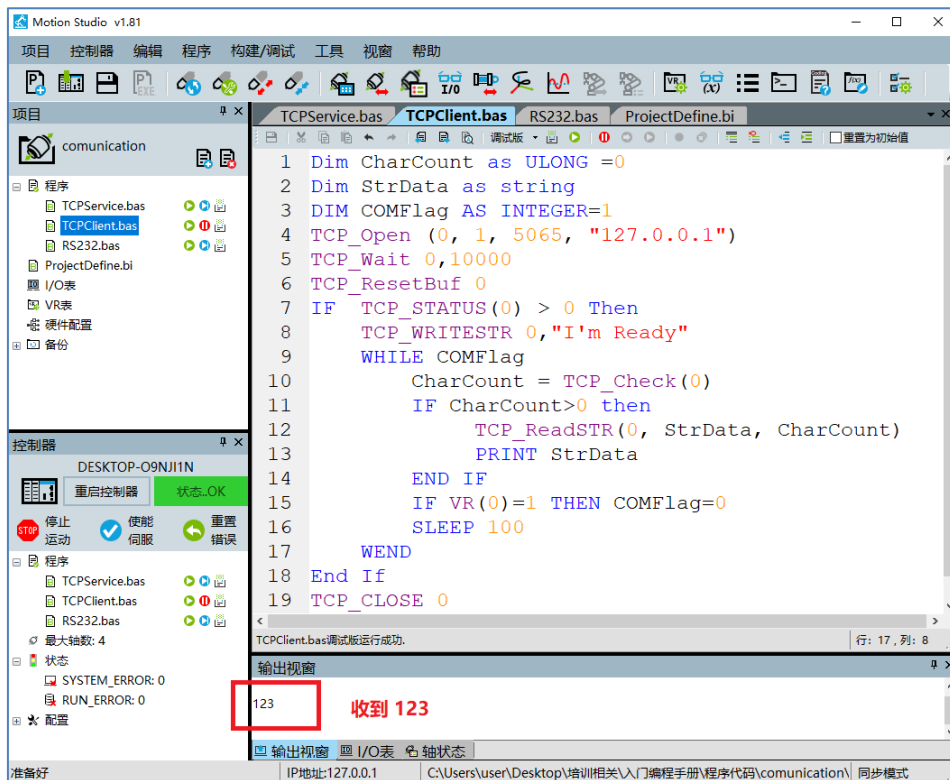
Dim CharCount as ULONG =0
Dim StrData as string
DIM COMFlag AS INTEGER=1
TCP_Open (0, 1, 5065, "127.0.0.1")
    '创建一个编号为 0 的客户端连接，对接 IP 为"127.0.0.1"的服务器
TCP_Wait 0,10000
    '等待编号为 0 的 TCP 通信连接完成，等待超时为 10000ms
TCP_ResetBuf 0                    '清除 TCP 缓存区数据
IF  TCP_STATUS(0) > 0 Then        '判断 TCP 是否连接成功
    TCP_WRITESTR 0,"I'm Ready"    '向外发送"I'm Ready"
    WHILE COMFlag                '通信循环
        CharCount = TCP_Check(0)
                                '获取 TCP 通讯接收缓存区的字符个数
        IF CharCount>0 then
            '判断 TCP 通讯接收缓存区是否有数据
            TCP_ReadSTR(0, StrData, CharCount)
                                '读取 TCP 通讯接收缓存区的数据，存入 StrData 中
            PRINT StrData
        END IF
        IF VR(0)=1 THEN COMFlag=0 '是否结束通信的判断
        SLEEP 100
    WEND
End If
TCP_CLOSE 0                       '关闭编号为 0 的客户端连接

```

在此以调试工具作为服务器，Motion Studio 程序运行后，TCP 连接成功，调试工具会收到 “I'm Ready” 如下图。



然后，让调试工具发出 “123”，Motion Studio 程序会接收到 “123”，并在输出视窗中显示出来，运行结果如下图。



◆ TCP/IP 服务器通信例程

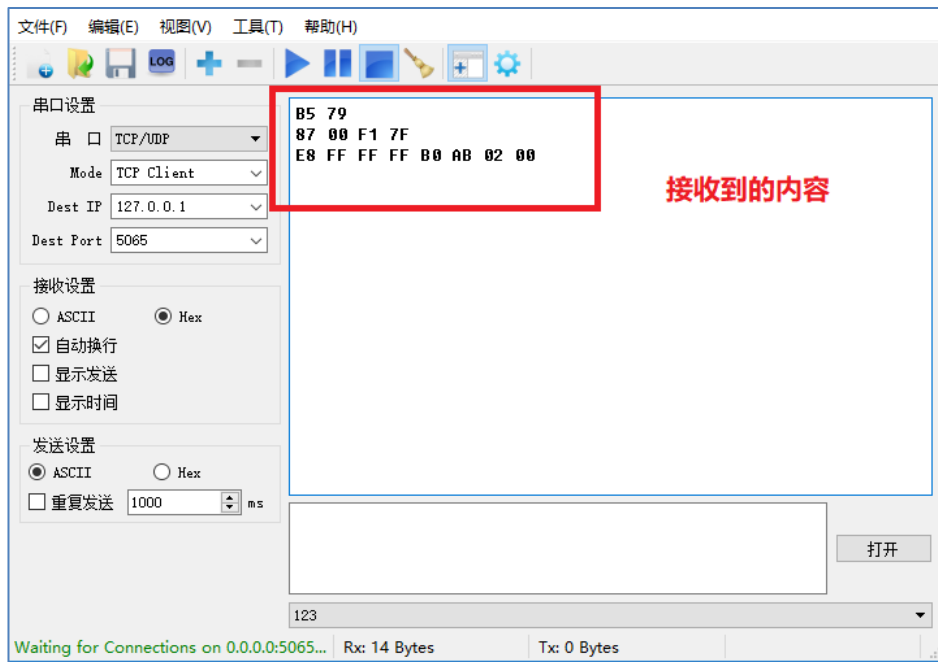
创建一个 TCP 服务器，通信编号为 0，待客户端连接成功又后，相隔 3 秒，依次发送 VR(0)与 VR(1)、VR(2)与 VR(3)、VR(4)与 VR(5)，三次发送完成后，关闭服务器连接。程序如下：

```
TCP_Open(0,0,5025) '创建一个编号为 0 的服务器
TCP_WAIT 0          '编号为 0 的服务器，等待客户端连接
VR(0)=-75
VR(1)=121
VR(2)=135
VR(3)=32753
VR(4)=-24
VR(5)=175024
If TCP_STATUS(0) > 0 Then '确认通讯编号为 0 的连接是否连接成功
    TCP_WriteVR 0,0,2,0
    SLEEP 3000
    TCP_WriteVR 0,2,2,1
    SLEEP 3000
    TCP_WriteVR 0,4,2,2
End If
TCP_Close 0          '关闭一个编号为 0 的服务器
```

上面的程序中，待客户端连接后，服务器三次发送数据：

- 1) 将 VR(0),VR(1)发送出去，服务器端收到的数据为十六进制数：B5 79。对应-75、121
- 2) 将 VR(2),VR(3)发送出去，服务器端收到的数据为十六进制数：87 00 F1 7F。对应 135、32753
- 3) 将 VR(4),VR(5)发送出去，服务器端收到的数据为十六进制数：E8 FF FF FF B0 AB 02 00。对应-24、175024

在此以调试工具作为客户端，Motion Studio 程序运行后，客户端连接成功，调试工具会收到相应容，如下图。



上图中，接收到的三行数据，分别对应 Motion Studio 程序三次发送的内容。

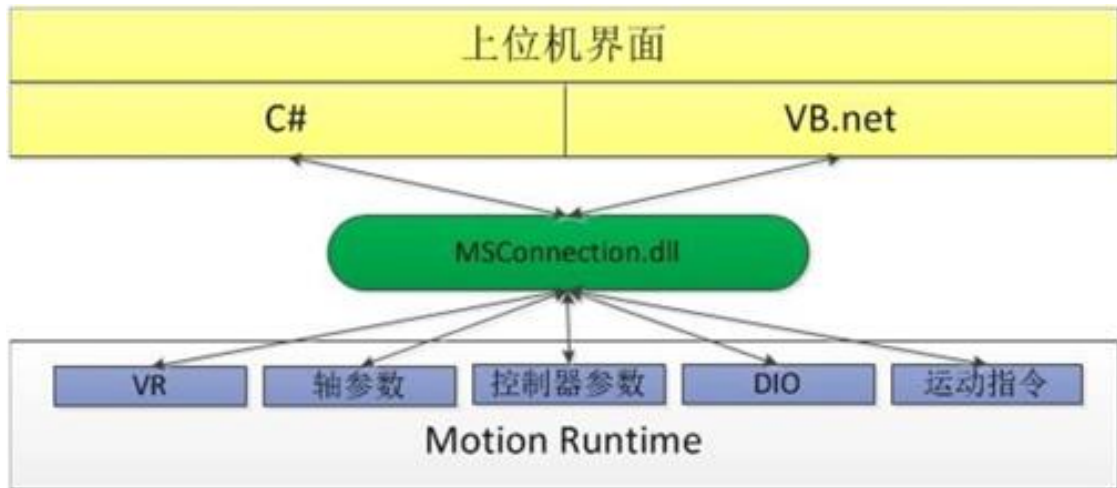
第十一章 编辑用户界面

11.1 调用 API 文件编辑用户界面

实现用户界面与运动控制器之间的数据交换，需要一个编程接口来对接双方的程序。

研华为用户提供应用程序编程接口，即 API 文件，为上位机与控制器双方提供数据交换的平台。

如下图所示，用户通过调用 MSConnection.dll(API 文件)中的相应函数，可以对 Motion Runtime 中的 VR、轴参数、控制器参数和 DIO 进行读/写操作，从而实现上位机界面与控制器之间的数据交换。



◆ 调用 API 函数示例

调用 API 函数，读取 Motion Runtime 中 VR(2)、VR(3)、VR(4)的数据，数据如下图：

名称	当前值	描述	初始值	Modbu	数据类型
范围[0-1...					
VR(0)	0		0	40001	BIT_32_F...
VR(1)	0		0	40003	BIT_32_F...
VR(2)	-100		0	40005	BIT_32_F...
VR(3)	1		0	40007	BIT_32_F...
VR(4)	200		0	40009	BIT_32_F...
VR(5)	0		0	40011	BIT_32_F...

上图 Motion Studio 中的 VR 表,可以看出,VR 数据类型为 32 位 Float,获取的数据长度为 3,开始地址为 40005, 程序代码如下：

```
float [] a = null; //声明一个 F32 类型的数组, 并赋值 null
a = Wrapper.ReadList_F32(3, 40005); //将 API 返回结果赋值给 a
```

上面的代码中, 调用 API 方法中的 ReadList 函数来获取 VR(2)、VR(3)、VR(4)的值, 并将获取值赋给数组 a, 最终结果 a[0]、a[1]、a[2]的值分别等于-100、1 和 200。

◆ 说明：

API 文件支持的语言包括 C#和 VB.net, 更详细用法请参见《MSConnection》手册。

11.2 使用 MS HMI.NET 控件编辑用户界面

在 11.1 章节讲解的“调用 API 编辑用户界面”的方式中，用户需要自己编写调用 API 的程序。

研华出于提高用户的编程效率考虑，提供一种更方便、快捷的方式，即为用户提供封装好的控件，不需要用户自己编写调用 API 的程序，这就是此章节要讲的 MS HMI.NET 控件。

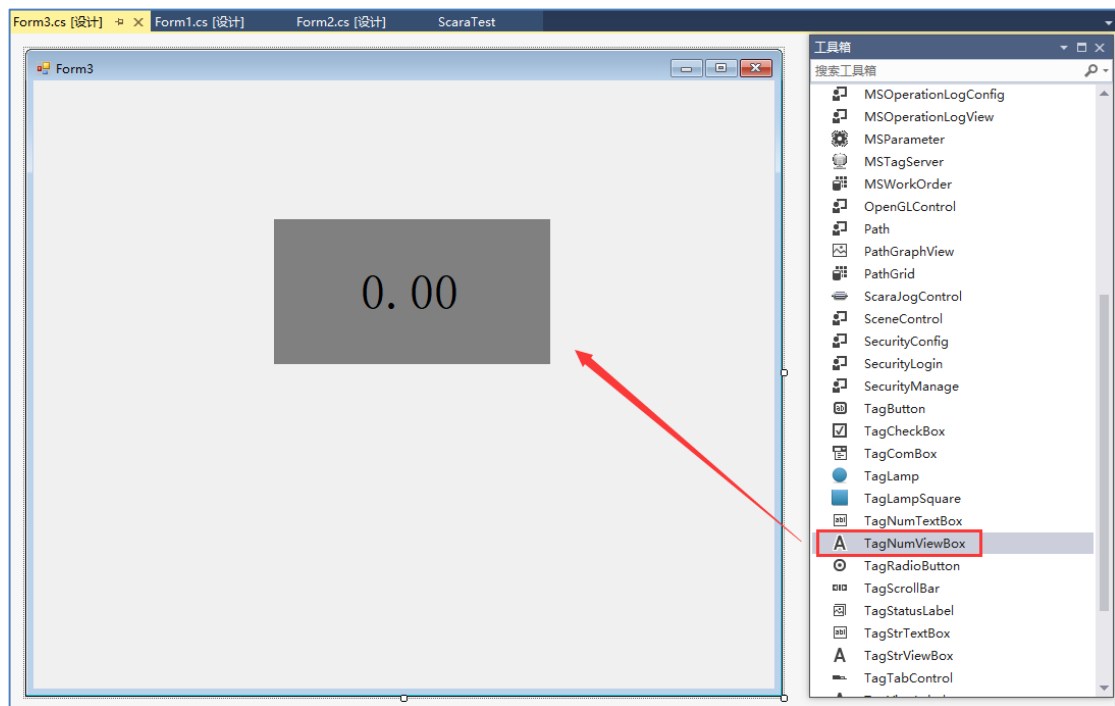
使用 MS HMI.NET 控件时，用户只需要将相应控件拖拉到界面上，然后将控件的属性地址和 Motion Runtime 中的地址绑定后，即可通过控件对相应地址进行读写操作。此种操作类似于触摸屏或者一些组态软件的操作，非常方便。

◆ MS HMI.NET 控件使用示例

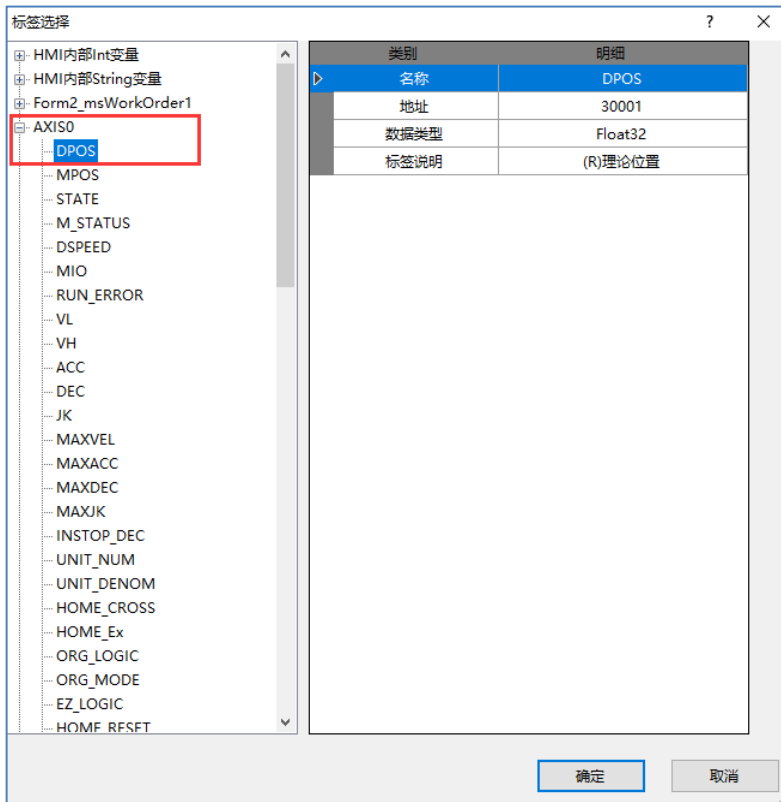
如何在用户界面添加一个控件，让该控件显示电机轴 0 的当前理论位置？

以 Viso Studio2015 界面编程环境为例，使用 MS HMI.NET 控件，只需要进行以下两步操作，即可实现功能：

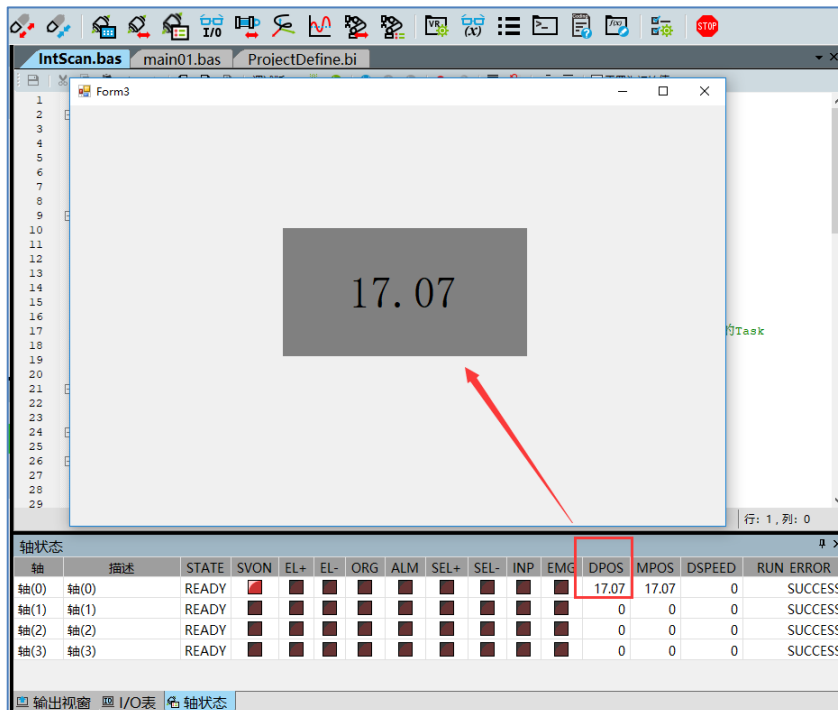
步骤一：在界面上添加一个 MS HMI.NET 提供的数值显示控件



步骤二：在控件的属性里面，将标签地址与轴 0 的理论位置绑定



通过以上两个简单的步骤就完成了操作，运行界面程序，如下：



从上图中可以看出，通过和 Motion Studio 中的轴状态显示比较，添加的控件正确的显示出了轴 0 当前的理论位置。

◆ 说明

MS HMI.NET 控件简化界面开发流程,可为用户节约大量的开发时间,更为详细的用法参见《MS HMI.NET》使用手册。

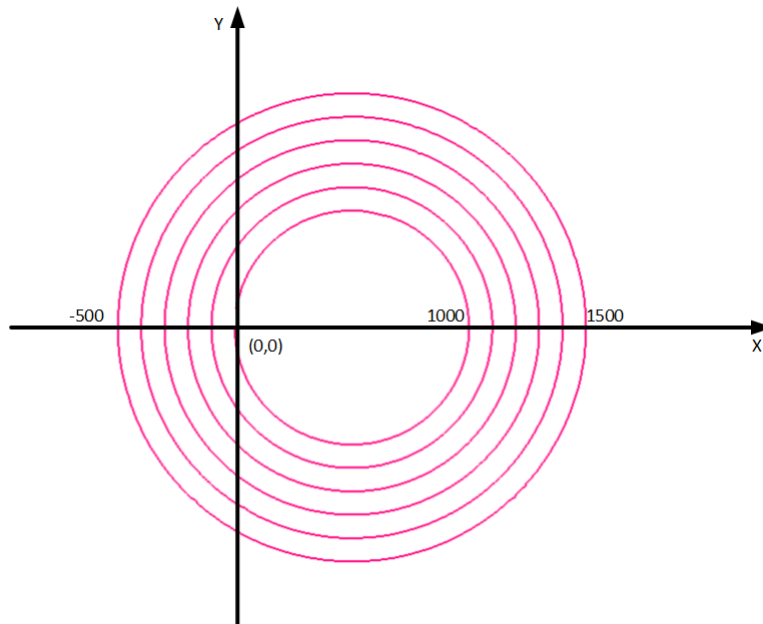
第十二章 应用示例

12.1 孔径加工

12.1.1 孔径加工需求

如下图，在加工平面（XY 平面）上，加工出 6 个圆的轨迹，6 个圆的圆心重合。以内圆最左侧的点位 $(0, 0)$ 为工作点，建立坐标系。

最小内圆的半径是 500，其余圆的半径依次增大 100



在两个圆之间加工时，需要 Z 轴控制工具抬起与下压。加工时，Z 轴控制工具处于下压状态；不加工时，Z 轴控制工具抬起。

12.1.2 孔径加工程序代码

在 Motion Studio 中编写运动控制程序，代码如下：

```

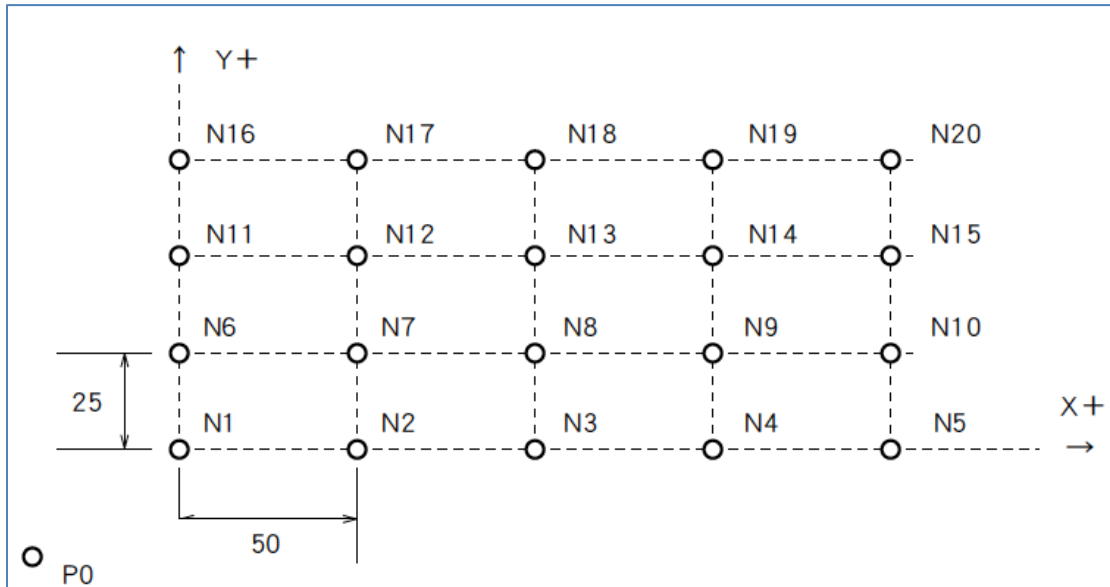
DIM AS INTEGER i, count      '定义变量，记录加工圆的次数
DIM AS DOUBLE rad, offset   '定义半径（相对值），两个圆之间的位置差
offset=100                  '两个圆之间的距离相差 100
rad=500                    '第一个圆的半径为 500
count=5                    '从 0-5，共 6 个圆
BASE 0,1
GVH=1000
SUB AX_Z_DOWN              '定义让 Z 轴下降的子程序
  BASE 2
  VH=1000
  MOVE 20
  WAIT DONE
END SUB
SUB AX_Z_UP                '定义让 Z 轴抬起的子程序
  BASE 2
  VH=1000
  MOVE -20
  WAIT DONE
END SUB
FOR i=0 To count
  AX_Z_DOWN()              'Z 轴下降，让工具接触工件
  BASE 0,1
  CIRC 1,rad,0,0,0        '逆时针旋转,相对圆心为(rad,0),相对圆弧终点(0,0)
  WAIT DONE
  AX_Z_UP()                'Z 轴上升，让工具离开工件
  BASE 0,1
  LINE -offset,0          '运动至下一个圆的位置
  WAIT DONE
  rad=rad+offset          '新的相对半径，要增加一个 offset 值
NEXT i
BASE 0,1
LINE offset*(count+1),0   '回到工作点
WAIT DONE

```

12.2 码垛

12.2.1 码垛需求

码垛位置图如下所示。



各点位置数据如下：

P0 点为取料点，坐标为 (0, 0)；N1 为第 1 个码垛点，坐标为 (100, 50)。

码垛点 N1-N20，各点 X 方向间隔 50，Y 方向间隔 25。

动作顺序要求：

- 1) 从 P0 位置取料，放至 N1 点。
- 2) 回到 P0 位置取料，放至 N2 点。
- 3) 回到 P0 位置取料，放至。。。。。
- 4) 如此，按顺序将 N1-N20 位置都放置物料。

12.2.2 码垛轨迹分析

在 X 轴方向，两个点之间的偏移量 $x_offset=50$ ；在 Y 轴方向，两个点之间的偏移量为 $y_offset=25$ 。

可定义一个数组 `PosTarget ()`，用于存储码垛点位置数据，包括 X 值和 Y 值。如果下一个码垛点在 X 方向，则 X 值增加 x_offset ；如果下一个码垛点在 Y 方向，则 Y 值增加 y_offset 。每码垛完成一行，就重置 X 值。

使用 FOR 循环，控制 X 或者 Y 方向的码垛点个数。

12.2.3 码垛程序代码

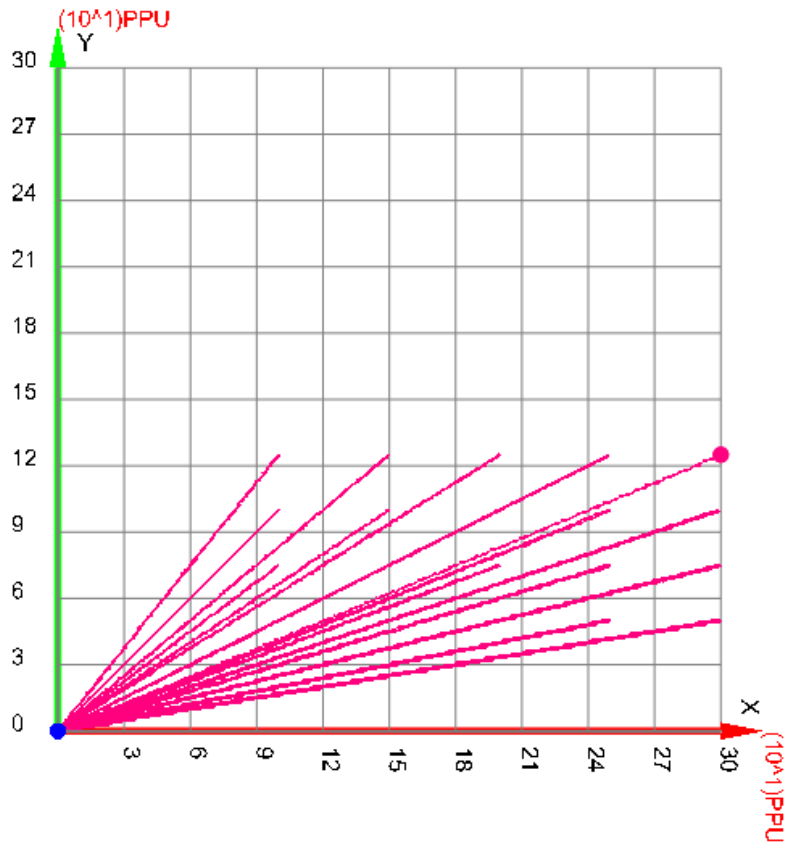
在 Motion Studio 中编写运动控制程序，代码如下：

```

DIM AS DOUBLE P0(2)={0,0}           ' P0 点（取料点）
DIM AS DOUBLE P1(2)={100,50}       ' 第一个码垛点 N1 点位置
DIM AS DOUBLE PosTarget(2)         ' 定义数组存储码垛点位
DIM AS DOUBLE  x_offset=50         ' x 方向码垛间隔距离
DIM AS DOUBLE  y_offset=25         ' y 方向码垛间隔距离
DIM AS INTEGER J,K
'定义子程序：将 AR_S()数组点的坐标值给 AR_T()，可用于位置数组间的数据传递
SUB COPY_P( AR_T() AS DOUBLE, AR_S() AS DOUBLE)
  AR_T(0) = AR_S(0)
  AR_T(1) = AR_S(1)
END SUB
BASE 0,1
COPY_P(PosTarget(), P1())          ' 将 N1 点的坐标值给 PosTarget()
FOR J=1 to 4
  FOR K=1 to 5
    LINEABS P0()                   ' 直线运动到 N1 点
    WAIT DONE
    LINEABS PosTarget()            ' 直线运动到 PosTarget 点
    WAIT DONE
    PosTarget(0) = PosTarget(0)+x_offset ' X 值偏移 x_offset
  NEXT K
  PosTarget(0) = P1(0)              ' X 值重置
  PosTarget(1) = PosTarget(1) + y_offset ' y 值偏移 y_offset
NEXT J

```

在 Motion Studio 打开 3D 轨迹显示，运行程序轨迹显示如下图：

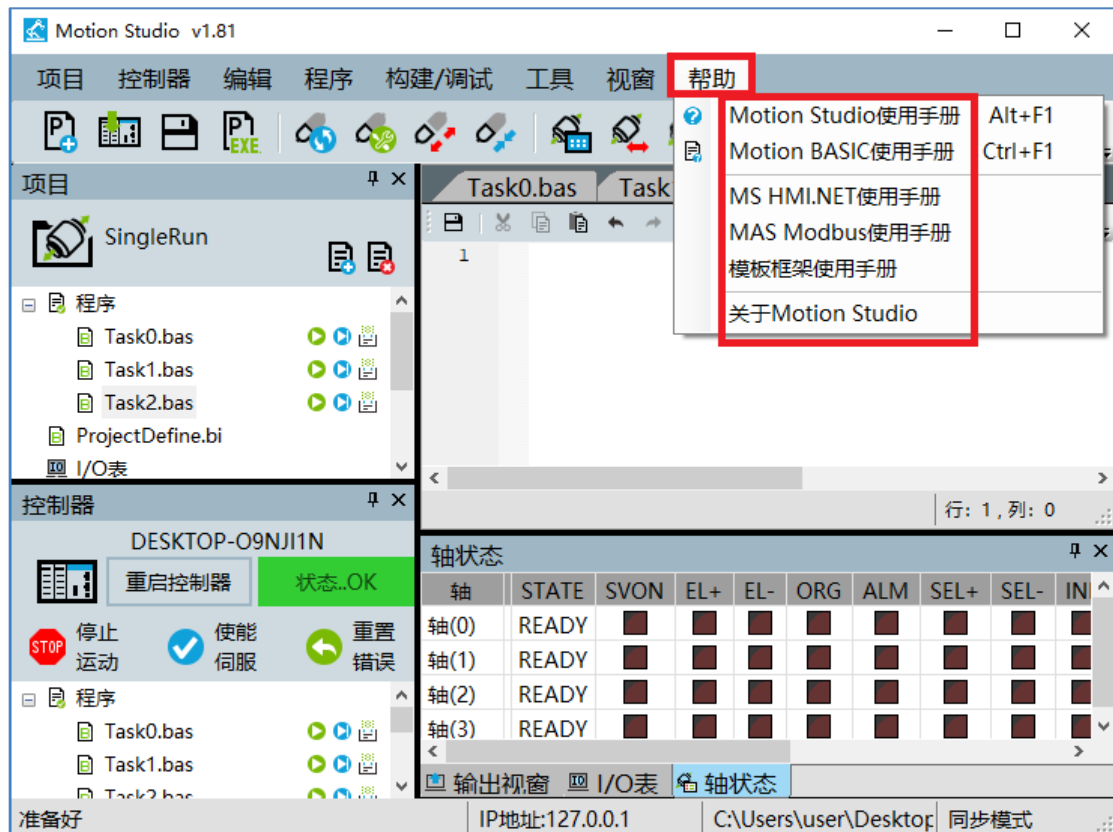


终章 附录

附录 1 手册使用导引

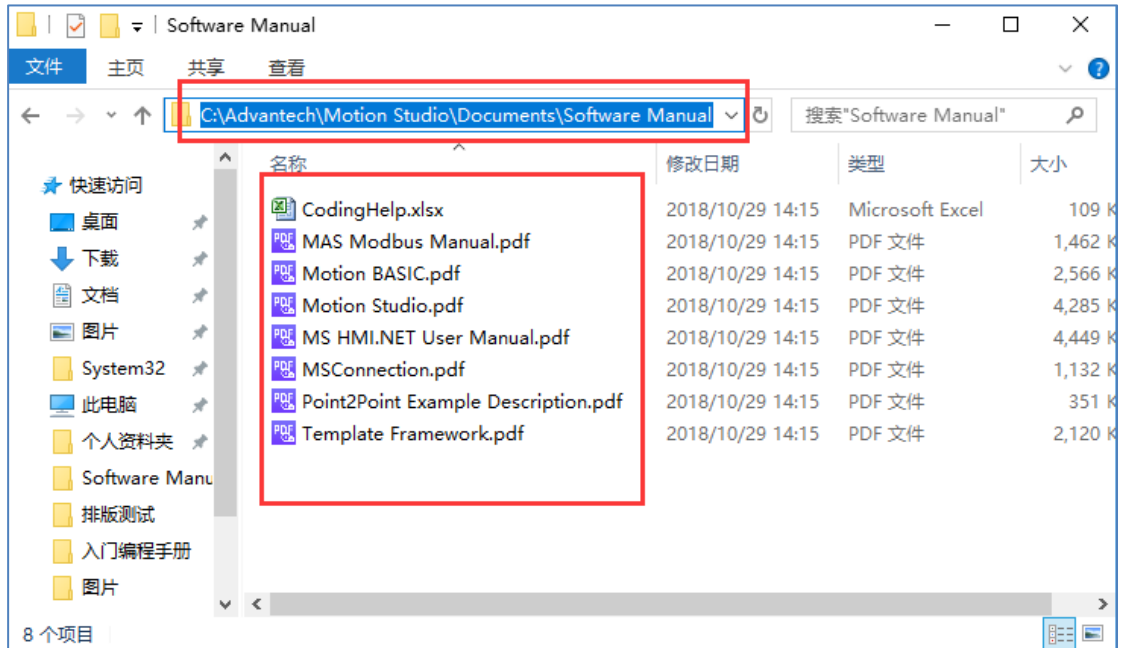
研华提供详细的手册供用户使用，用户可通过以下方式查看手册。

1.1 通过 Motion Studio 软件的帮助工具查看



如上图，打开菜单栏中的帮助，即可查看相应手册。

1.2 在 Motion Studio 安装路径下查看



如上图，在 Motion Studio 安装目录下：

\Advantech\Motion Studio\Documents\Software Manual 路径下，也可查看相应的手册。

附录 2 影响 Motion Runtime 运行的因素

Motion Runtime 软件在运行时，会受以下因素影响：

◆ 系统防火墙

请关闭电脑病毒防护、系统防火墙。



◆ 杀毒软件

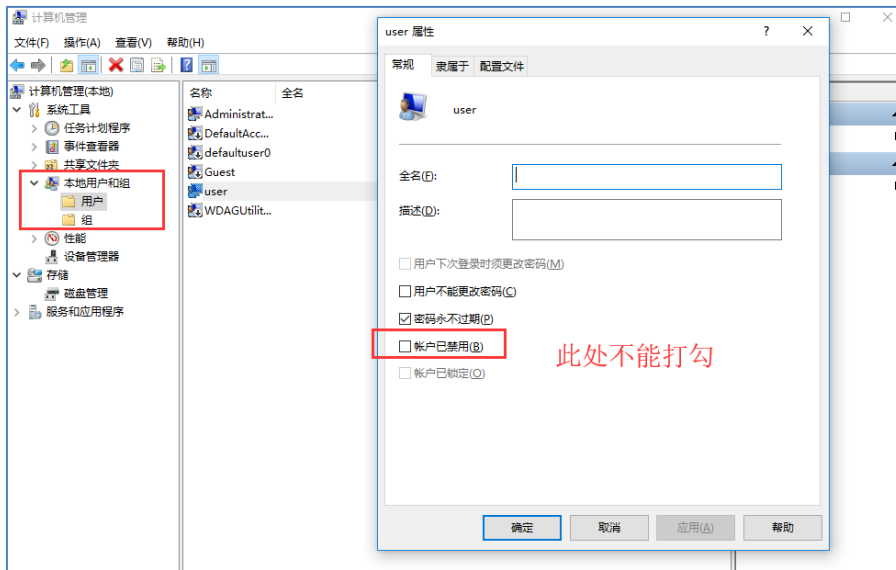
一些杀毒软件会错误地禁止 Motion Runtime 的运行，这些软件如下：

- a) 360
- b) 鲁大师
- c) 驱动精灵
- d) 其它软件等

所以，Motion Runtime 在运行时，请务必禁用相关杀毒软件。

◆ 用户权限

如下图，在计算机管理中，当前登陆用户要拥有管理员权限。



◆ 用户帐户控制设置

请将用户账户控制设置成“从不通知”。

