英威腾中大型PLC

编程手册

深圳市英威腾电气股份有限公司
SHENZHEN INVT ELECTRIC CO., LTD.

前言

概述

非常感谢您使用中大型可编程控制器。

本手册记载了使用中大型可编程控制器所必需的信息。使用前请详细阅读本手册，充分理解其功能和性能，完成系统构建，发挥其优越性能。

本手册适用于 AX、TM、TP 系列可编程控制器，但需注意部分功能指令的适用范围。

读者对象

本手册适用于具有电工专业知识的人员（合格的电气工程师或具有同等知识的人员）阅读。

在线支持

除本手册外，还可以通过登录英威腾官方网站获取产品资料和技术支持。

网址：https://www.invt.com.cn

如果最终使用为军事单位，或将本产品用于兵器制造等用途时，本产品将列入《中华人民共和国对外贸易法》规定的出口产品管制对象，在出口时，需要进行严格审查，并办理所需的出口手续。

修改记录

本公司保留对产品不断改进的权利，恕不另行通知。

编号	修改内容摘要	版本	日期
1	创建	V1.0	2024.08

目 录

1 程序结构与执行 1
1.1 程序结构 1
1.2 任务 1
1.3 程序执行过程 2
1.4 任务的执行类型 4
1.5 任务优先级 5
1.6 多子程序的运行 8
1.7 单轴控制 9
1.7.1 单轴控制编程说明 9
1.7.2 单轴控制常用的 MC 功能块 10
1.8 凸轮同步控制 10
1.8.1 凸轮表的周期模式 11
1.8.2 凸轮表的输入方法 12
1.8.3 凸轮表的数据结构 12
1.8.4 凸轮表的引用与切换 13
1.9 编程建议 13
2 EtherCAT 运行机制 14
2.1 EtherCAT 运行原理 14
2.1.1 EtherCAT 协议介绍 14
2.1.2 工作计数器 WKC 14
2.1.3 寻址方式 14
2.1.4 分布时钟 18
2.2 EtherCAT 通信模式 20
2.2.1 周期性过程数据通信 20
2.2.2 非周期性邮箱数据通信 22
2.3 EtherCAT 状态机 23
2.4 EtherCAT 伺服驱动器控制应用协议 25
2.4.1 基于 EtherCAT 的 CAN 应用协议（CoE） 25
2.4.2 IEC 61800-7-204 的伺服驱动行规（SERCOS） 28
3 轴状态机制 32
3.1 轴状态转换 32
4 编程基础 33
4.1 变量 33
4.1.1 变量声明 33
4.1.2 数据类型 34
4.1.3 变量类型 40
4.1.4 保持变量 40
5 编程语言 42
5.1 概述 42
5.2 结构化文本（ST） 42
5.2.1 结构化文本编程语言简介 42
5.2.2 程序执行顺序 43
5.2.3 表达式执行顺序 43
5.2.4 指令语句 44
5.2.5 应用举例 54
5.3 梯形图（LD）、功能块（FBD） 58

5.3.1 梯形图、功能块图编程语言简介 58
5.3.2 程序执行顺序 58
5.3.3 执行控制 59
5.3.4 连接元素 60
5.3.5 应用举例 68
5.4 指令表(IL) 70
5.4.1 指令表编程语言简介 70
5.4.2 连接元素 71
5.4.3 操作指令 74
5.4.4 函数及功能块 79
5.4.5 应用举例 80
5.5 顺序功能流程图（SFC） 82
5.5.1 顺序流程图编程语言简介 82
5.5.2 SFC 的结构 84
5.6 连续功能图（CFC） 95
5.6.1 连续功能图编程语言结构 95
5.6.2 连接元素 97
5.6.3 CFC 的组态 103
6 基础指令 104
6.1 比较指令 104
6.1.1 GT 大于 104
6.1.2 LT 小于 104
6.1.3 GE 大于等于 104
6.1.4 LE 小于等于 104
6.1.5 EQ 等于 105
6.1.6 NE 不等于 105
6.2 选择指令 105
6.2.1 SEL 二选一指令 105
6.2.2 MUX 多选一指令 105
6.2.3 MAX 取最大值指令 106
6.2.4 MIN 取最小值指令 106
6.2.5 LIMIT 极限值指令 106
6.3 计数器指令 107
6.3.1 CTU 向上计数功能块 107
6.3.2 CTD 向下计数功能块 107
6.3.3 CTUD 向上/向下计数功能块 108
6.4 定时器指令 108
6.4.1 TP 触发定时器功能块 108
6.4.2 TON 延时接通定时器 109
6.4.3 TOF 延时关断定时器 110
6.4.4 RTC 时钟计时功能块 110
6.5 位和字逻辑指令 111
6.5.1 AND 与指令 111
6.5.2 OR 或指令 111
6.5.3 NOT 非指令 111
6.5.4 XOR 异或指令 111
6.5.5 SR 置位优先功能块 112
6.5.6 RS 复位优先功能块 112
6.5.7 R_TRIG 上升沿功能块 112
6.5.8 F_TRIG 下降沿功能块 113
6.6 位/字节函数 113

6.6.1 EXTRACT 113
6.6.2 PACK 组合指令 113
6.6.3 PUTBIT 114
6.6.4 UNPACK 拆分功能块 114
6.7 数据移位指令 114
6.7.1 SHL 按位左移 114
6.7.2 SHR 按位右移 114
6.7.3 ROL 按位循环左移 114
6.7.4 ROR 按位循环右移 115
6.8 数据类型转换指令 115
6.8.1 BOOL_TO_<TYPE> 115
6.8.2 BYTE_TO_<TYPE> 115
6.8.3 WORD_TO_<TYPE> 115
6.8.4 DWORD_TO_<TYPE> 115
6.8.5 INT_TO_<TYPE> 116
6.8.6 SINT_TO_<TYPE> 116
6.8.7 DINT_TO_<TYPE> 116
6.8.8 UDINT_TO_<TYPE> 116
6.8.9 REAL_TO_<TYPE> 116
6.8.10 STRING_TO_<TYPE> 116
6.8.11 TIME_TO_<TYPE> 117
6.8.12 TOD_TO_<TYPE> 117
6.8.13 DATE_TO_<TYPE> 117
6.8.14 DT_TO_<TYPE> 117
6.9 数据处理指令 117
6.9.1 MOVE 117
6.9.2 HEXinASCII_TO_BYTE 117
6.9.3 BYTE_TO_HEXinASCII 118
6.9.4 WORD_AS_STRING 118
6.10 数学运算指令 118
6.10.1 ADD 118
6.10.2 SUB 118
6.10.3 MUL 118
6.10.4 DIV 119
6.10.5 MOD 119
6.10.6 ABS 119
6.10.7 SQRT 119
6.10.8 LN 119
6.10.9 LOG 120
6.10.10 EXP 120
6.10.11 EXPT 120
6.10.12 SIN 120
6.10.13 COS 120
6.10.14 TAN 121
6.10.15 ASIN 121
6.10.16 ACOS 121
6.10.17 ATAN 121
6.10.18 RAD/DEG 121
6.10.19 SIZEOF 122
6.11 日期和时间指令 122
6.11.1 SetDateAndTime 122

6.11.2 GetDateAndTime 122
6.12 字符串功能指令 122
6.12.1 LEN 122
6.12.2 LEFT 123
6.12.3 RIGHT 123
6.12.4 MID 123
6.12.5 CONCAT 123
6.12.6 INSERT 124
6.12.7 DELETE 124
6.12.8 REPLACE 124
6.12.9 FIND 124
6.13 地址操作指令 125
6.13.1 ADR/^ 125
6.13.2 BITADR 125
6.14 文件操作指令 125
6.14.1 概述 125
6.14.2 输入输出 125
6.14.3 files_load 文件加载函数 126
6.14.4 Files_Copy 文件复制函数 126
6.14.5 Delete_File 文件删除函数 126
6.14.6 Write_File 文件写入函数 127
6.15 调节器 130
6.15.1 PD 130
6.15.2 PID 131
6.15.3 PID_FIXCYCLE 132
6.16 BCD 转换指令 132
6.16.1 BCD_TO_INT 132
6.16.2 INT_TO_BCD 132
6.17 系统指令 132
6.17.1 控制器故障诊断指令 132
6.17.2 TM 控制器 IP、时间指令 134
6.17.3 TP 控制器 IP、时间指令 134
6.18 信号发生器 135
6.18.1 BLINK 135
6.18.2 FREQ_MEASURE 135
6.18.3 GEN 136
6.19 数学辅助功能块 137
6.19.1 DERIVATIVE 137
6.19.2 INTEGRAL 138
6.19.3 LIN_TRAFO 138
6.19.4 STATISTICS_INT 139
6.19.5 STATISTICS_REAL 139
6.19.6 VARIANCE 139
6.20 操作功能块 139
6.20.1 CHARCURVE 139
6.20.2 RAMP_INT 140
6.20.3 RAMP_REAL 141
6.21 模拟值处理 141
6.21.1 HYSTERESIS 141
6.21.2 LIMITALARM 141
7 运动控制指令 142

7.1 单轴指令 142
7.1.1 MC_Power 142
7.1.2 MC_Halt 143
7.1.3 MC_Home 144
7.1.4 MC_MoveAbsolute 145
7.1.5 MC_AccelerationProfile 147
7.1.6 MC_MoveAdditive 148
7.1.7 MC_MoveRelative 150
7.1.8 MC_MoveSuperImposed 152
7.1.9 MC_MoveVelocity 154
7.1.10 MC_PositionProfile 156
7.1.11 MC_ReadActualPosition 157
7.1.12 MC_ReadBoolParameter 158
7.1.13 MC_ReadAxisError 159
7.1.14 MC_ReadStatus 160
7.1.15 MC_ReadParameter 161
7.1.16 MC_Reset 162
7.1.17 MC_Stop 163
7.1.18 MC_VelocityProfile 164
7.1.19 MC_WriteBoolParameter 166
7.1.20 MC_WriteParameter 167
7.1.21 MC_AbortTrigger 168
7.1.22 MC_ReadActualTorque 169
7.1.23 MC_ReadActualVelocity 170
7.1.24 MC_SetPosition 171
7.1.25 MC_TouchProbe 172
7.1.26 MC_MoveContinuousAbsolute 174
7.1.27 MC_MoveContinuousRelative 175
7.1.28 MC_Jog 177
7.1.29 MC_Inch 179
7.1.30 SMC3_PersistPosition 180
7.1.31 SMC3_PersistPositionSingleturn 182
7.1.32 SMC3_PersistPositionLogical 183
7.1.33 SMC_Homing 184
7.1.34 SMC_SetControllerMode 188
7.1.35 SMC_SetTorque 189
7.2 主从轴指令 190
7.2.1 MC_CamIn 190
7.2.2 MC_Camout 196
7.2.3 MC_CamTableSelect 198
7.2.4 MC_GearIn 199
7.2.5 MC_GearOut 201
7.2.6 MC_GearInPos 202
7.2.7 MC_Phasing 206
8 通信指令 208
8.1 串口自由口指令 208
8.1.1 指令列表 208
8.1.2 ICP_Serial_Comm_hCom 208
8.1.3 ICP_Serial_Comm_Read 209
8.1.4 ICP_Serial_Comm_Write 210
8.2 TCP 自由口指令 212

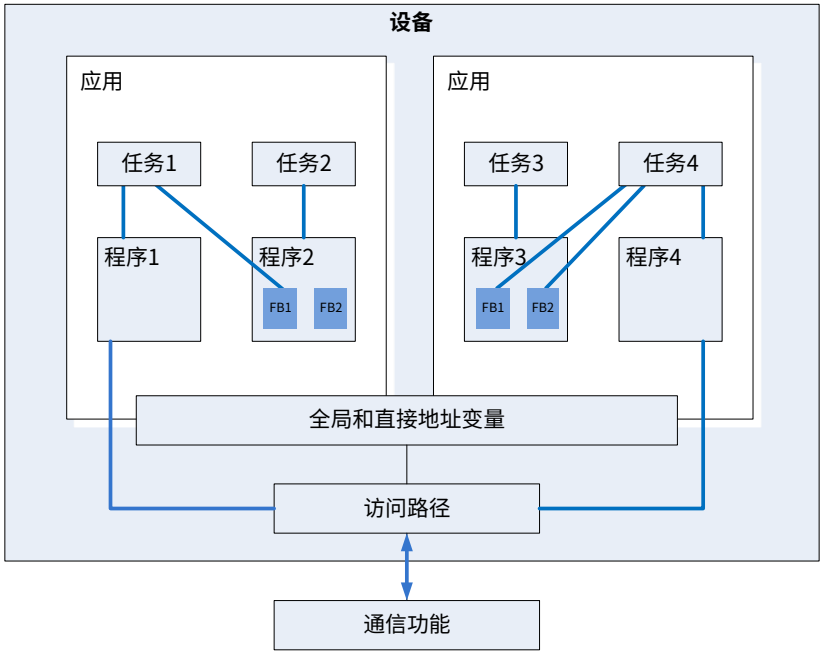
8.2.1 指令列表 212
8.2.2 ICP_TCP_Comm_Client 212
8.2.3 ICP_TCP_Comm_Write 213
8.2.4 ICP_TCP_Comm_Read 214
8.2.5 ICP_TCP_Comm_Server 215
8.2.6 ICP_TCP_Comm_Connect 215
8.3 UDP 自由口指令 216
8.3.1 指令列表 216
8.3.2 ICP_UDP_Comm_Send 216
8.3.3 ICP_UDP_Comm_Receive 217
9 脉冲输出指令 219
9.1 辅助指令 219
9.1.1 IMC_GetSys_P 219
9.1.2 IMC_Axis_P 220
9.1.3 IMC_Power_P 221
9.1.4 IMC_SetPosition_P 222
9.1.5 IMC_ReadCmdPosition_P 222
9.1.6 IMC_ReadParameter_P 223
9.1.7 IMC_ReadStatus_P 224
9.1.8 IMC_SendData_P 224
9.1.9 IMC_Acc2Jerk_P 225
9.1.10 IMC_AccTime2Jerk_P 226
9.2 单轴指令 226
9.2.1 IMC_Jog_P 226
9.2.2 IMC_Inch_P 227
9.2.3 IMC_MoveAbsolute_P 228
9.2.4 IMC_MoveRelative_P 229
9.2.5 IMC_MoveVelocity_P 229
9.2.6 IMC_Home_P 230
9.2.7 IMC_Halt_P 234
9.2.8 IMC_Stop_P 235
9.2.9 IMC_Reset_P 235
9.2.10 IMC_SetOverride_P 236
9.2.11 IMC_MoveSuperImposed_P 237
9.2.12 IMC_ReadCmdSpeed_P 237
10 故障代码 239
10.1 SMC_ERROR 故障代码（402 轴通用错误信息） 239
10.2 PLC 故障代码表（适用 TM、TP 系列 PLC） 244

1 程序结构与执行

1.1 程序结构

软件模型用分层结构表示。每一层隐含其下面层的许多特性。软件模型描述基本的软件元素及其相互关系。这些软件元素包含：设备、应用、任务、全局变量、访问路径和应用对象，其内部结构如图 1-1 所示，该软件模型与 IEC 61131-3 标准的软件模型保持一致。

图 1-1 程序分层结构



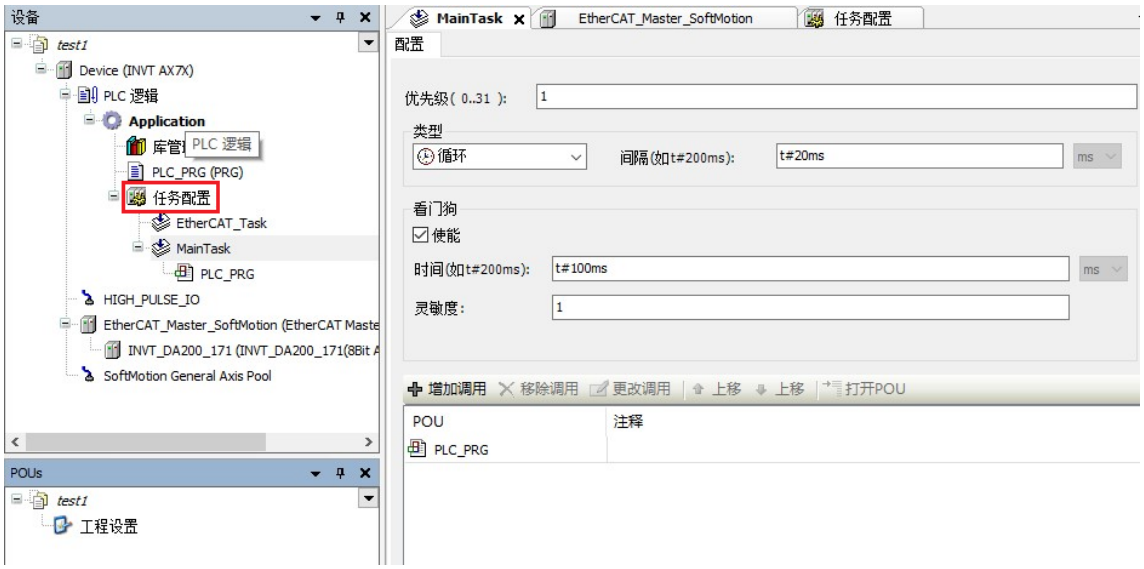
1.2 任务

一个程序可以用不同的编程语言来编写。典型的程序由许多互连的功能块组成，各功能块之间可互相交换数据。在一个程序中不同部分的执行通过“任务”来控制。“任务”被配置以后，可以使一系列程序或功能块周期性地执行或由一个特定的事件触发开始执行。

在设备树中有“任务管理器”选项卡，使用它除了声明特定的 PLC_PRG 程序外，还可以控制工程内其他子程序的执行处理。任务是用于规定程序组织单元在运行时的属性，它是一个执行控制元素，具有调用的能力。在一个任务配置中可以建立多个任务，而一个任务中，可以调用多个程序组织单元，一旦任务被设置，它就可以控制程序周期执行或者通过特定的事件触发开始执行。

在任务配置中，用名称、优先级和任务的启动类型来定义它。这启动类型可以通过时间（周期的，随机的）或通过内部或外部的触发任务时间来定义，例如使用一个布尔型全局变量的上升沿或系统中的某一特定事件。对每个任务，可以设定一串由任务启动的程序。如果在当前周期内执行此任务，那么这些程序会在一个周期的长度内被处理。优先权和条件的结合将决定任务执行的时序，任务设置界面如图 1-2 所示。

图 1-2 任务配置界面



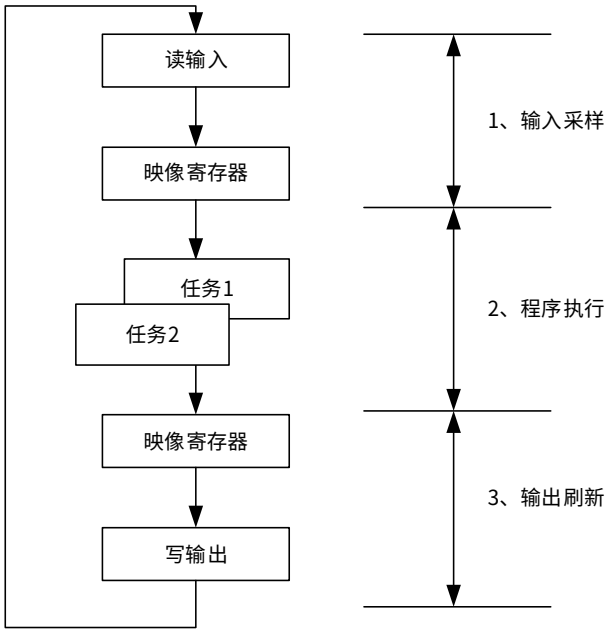
编程者需遵循如下规则：

- 循环任务的最大个数为 100，自由运行任务的最大个数为 100，事件触发任务的最大个数为 100。
- 根据目标系统，PLC_PRG 可能会在任意情况下作为一个自由程序执行，而不用手动插入任务配置中。
- 处理和调用程序是根据任务编辑器内自上而下的顺序所执行的。

1.3 程序执行过程

下图详细的描述了可编程控制器内部执行程序的完整流程，主要的流程有输入采样、程序执行和输出刷新这三个部分组成。

图 1-3 控制器执行流程



- 输入采样

每次扫描周期开始时，可编程控制器检测输入设备（开关、按钮等）的状态，将状态写入输入映像寄存区内。在程序执行阶段，运行系统从输入映像区内读取数据进行程序解算。需要特别注意的是输入的刷新只发生在扫描的开始阶段，在扫描过程中，即使输出状态改变，输入状态也不会发生变化。

- 执行程序

在扫描周期的执行程序阶段，可编程控制器从输入映像区或输出映像区内读取状态和数据，并依照指令进行逻辑和

算术运算，运算的结果保存在输出映像区相应的单元中。在这一阶段内，只有输入映像寄存器的内容保持不变，其他映像寄存器的内容会随着程序的执行而变化。

- 输出刷新

输出刷新阶段也可称为写输出阶段，可编程控制器将输出映像区的状态和数据传送到输出点上，并通过一定的方式隔离和功率放大，驱动外部负载。可编程控制器在一个扫描周期内除了完成上述三个阶段的任务外，还要完成内部诊断、通信、公共处理以及输入/输出服务等辅助任务。

可编程控制器重复执行上述过程，每重复一次的时间就是一个工作周期（或扫描周期）。由可编程控制器的扫描方式可得知，可编程控制器为了迅速响应输入输出数据的变化，完成控制任务扫描时间较短，控制器的工作周期一般都控制在 ms 数量级，因此需要开发稳定、可靠、响应快的实时系统供可编程控制器运行系统所用。

由于可编程控制器采用循环的工作方式，输入信号只会在每个周期的开始阶段进行刷新，输出在每个工作周期的结束阶段进行集中输出，所以必然会产生输出信号相对输入信号的滞后现象。从可编程控制器的输入端有一个信号输入发生到变化到控制器的输出端，对该输入信号的变化做出反应需要一段时间。滞后延时时间是设计可编程控制器控制系统时应了解的一个重要参数。通常，滞后延时时间的长短和以下因素有关。

1、 输入电路的滤波时间，它由硬件 RC 滤波电路的时间常数决定，通过改变时间常数可调整输入延迟时间。

表 1-1 为 AX-EM-1600D 数字量输入模块技术参数，其中“端口滤波时间”表示该输入模块的滤波时间。

表 1-1 AX-EM-1600D 数字量输入模块规格参数

项目	规格
输入通道	16
输入连接方式	18 点接线端子
输入电压等级	24V（最高可达 30V）
输入电流（典型）	4.7mA
ON 电压	＞15VDC
OFF 电压	＜5VDC
端口滤波时间	10ms
输入阻抗	5.4kΩ
输入信号形式	电压直流输入
隔离方式	光耦隔离
输入动态显示	输入有效时，指示灯亮

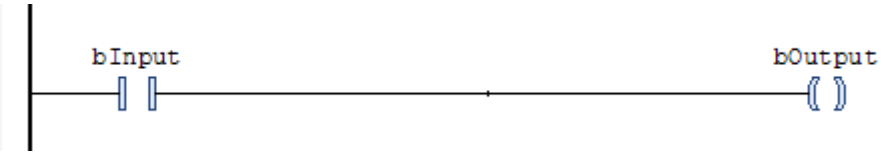
2、 输出电路的滞后时间，与输出电路的方式有关系，继电器输出方式的滞后时间一般为 10ms 左右，晶体管输出方式滞后时间小于 1ms。

3、 控制器循环扫描的工作方式。

4、 用户程序中语句的安排。

为了能够让读者更好的理解这整个过程，如下通过一个简单的梯形图程序例子来反应其输入输出及滞后现象的工作原理。程序逻辑如图 1-4 所示。

图 1-4 TM 系列可编程控制器程序



bInput 与外部的输入按钮有硬件映射关系，当按钮被按下时 bInput 为 ON，bOutput 与外部继电器的线圈也有硬件映射关系。当 bOutput 为 ON 时，继电器的线圈也会得电。在可编程控制器的内部，其处理的关系如图 1-6 所示。当输入按钮按下时，bInput 不会马上被置为 ON，因为输入采样只有在一个工作周期的开始阶段才能被程序执行，由于该按钮信号已经过了采样阶段，通常会在下一个周期开始阶段才被执行。在图 1-6 的程序中将 bInput 的状态赋值给 bOutput，由于在程序运行期间存在一定的程序计算，所以需要一定的程序处理时间 bOutput 才会被置为 ON。由于输出刷新是发生在程序处理的最后阶段，故在该周期的最后阶段 bOutput 通过输出刷新功能将其数值传递至实际硬件，最终线圈才能得电。图 1-5 是比较理想状态，最终的输出只有一个周期的延迟。

图 1-5 输出最快的情况

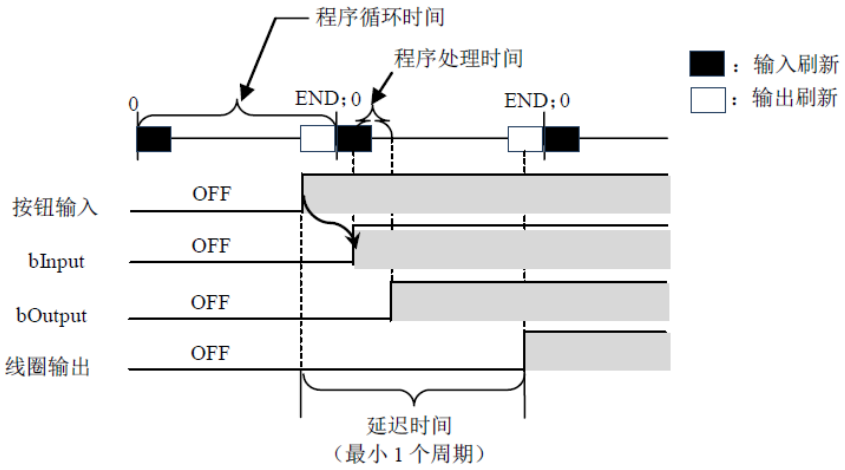
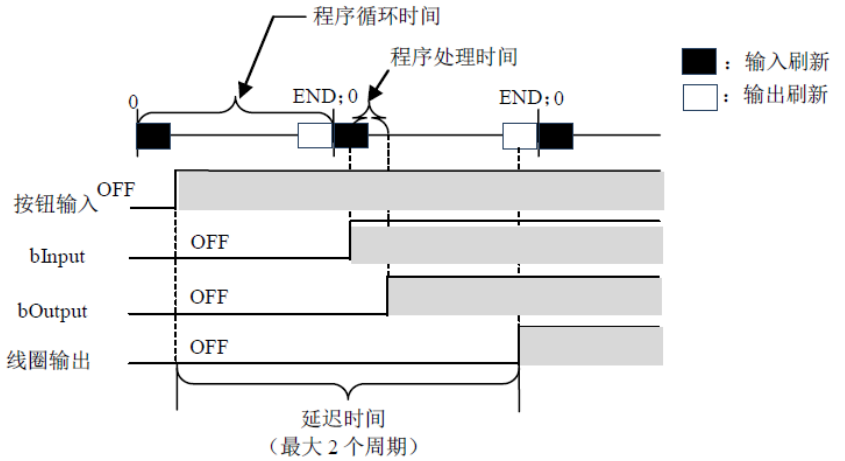


图 1-5 为比较理想的情况，那么也要考虑到比较糟的情况，当一个周期的输入采样刚刚结束的时候，此时外部输入按钮为 ON，由于需要在下一个周期开始时输入信号才能被载入至输入映像区。而实际输出则要等到第二个周期结束时才能被载入输出映像区，故整个过程如图 1-6 所示。在这种情况输出的延时接近于 2 个周期，这种情况为最迟的输出情况。

图 1-6 输出最迟的情况



1.4 任务的执行类型

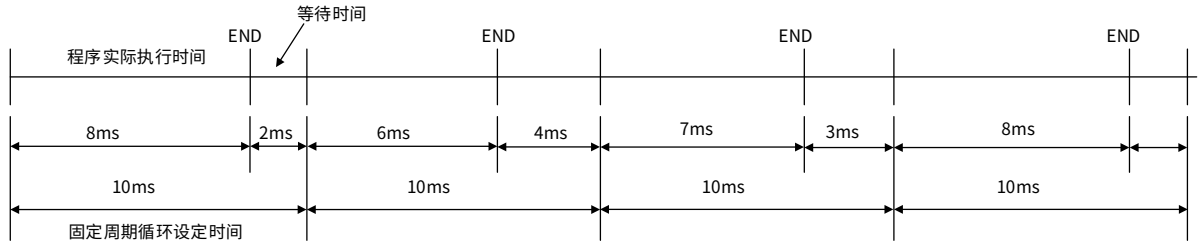
在任务配置树的最顶端有条目“任务配置”。其中的内容是当前定义的任务，每个通过任务名代表。特定任务的 POUs 调用没有显示在任务配置树中。针对每个独立的任务可以对其进行执行的类型编辑及配置。包括固定周期循环、事件触发、自由运行和状态触发 4 种类型。

图 1-7 任务执行的类型

1、 固定周期循环-Cyclic

根据程序中所使用的指令执行与否，程序的处理时间会有所不同，所以实际执行时间在每个扫描周期都发生不同的变化，执行时间有长有短。通过使用固定周期循环方式，能保持一定的循环时间反复执行程序。即使程序的执行时间发生变化，也可以保持一定的刷新间隔时间。在这里，推荐大家优先选择固定周期循环任务启动方式。例如，假设将程序对应的任务设定为固定周期循环方式，间隔时间设定为 10ms 时，实际程序执行的时序图如图 1-8 所示。

图 1-8 固定周期循环执行顺序

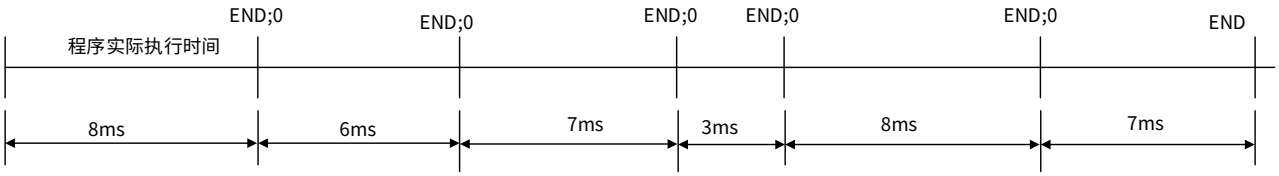


如果程序实际执行时间在规定的固定周期循环设定时间内执行完，则空余时间用作等待。如应用中还有优先级较低的任务未被执行，则剩下的等待时间用来执行相对低优先级的任务。任务的优先级在后文会有详细的说明。

2、 自由运行-Freewheeling

程序一开始运行任务就被处理，一个运行周期结束后任务将在下一个循环中被自动重新启动。该执行方式不受程序扫描周期的影响。即确保每次执行完程序的最后一条指令后才进入下一个循环周期，否则不会结束该程序周期。图 1-9 为自由运行执行顺序的时序图。

图 1-9 自由运行执行顺序



由于自由运行执行方式因为没有固定的任务时间，所以每次执行的时间可能都不一样。故不能保证程序的实时性，在实际的应用中选用此方式的场合较少。

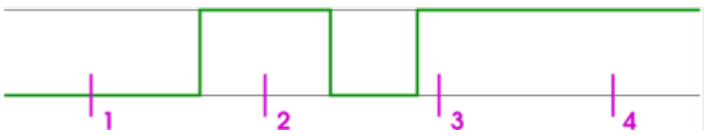
3、 事件触发-Event

如果事件区域的变量得到一个上升沿，任务开始。

4、 状态触发-Status

如果事件区域的变量为 TRUE，任务开始。状态触发方式与事件触发功能类似，区别在于状态触发的触发变量只要为 TRUE 程序就执行，为 FALSE 则不执行。而事件触发只采集触发变量的上升沿有效信号。图 1-10 中针对事件触发和状态触发分别进行了比较，绿色实线为两种触发方式选择的布尔变量状态，表 1-2 为比较的结果。

图 1-10 任务输入触发信号



在采样点 1~4（紫色）不同类型的任务展示了不同的反应。这个具体的事件为 TRUE 完成了状态驱动任务的条件，然而一个事件驱动任务需要事件从 FALSE 变为 TRUE。如果任务计划的采样频率过低，事件的上升沿可能检测不到。

表 1-2 事件触发与状态触发执行结果比较

执行点	1	2	3	4
事件触发（Event）	不执行	执行	执行	执行
状态触发（Status）	不执行	执行	不执行	不执行

1.5 任务优先级

1、 任务优先级设置

可以对任务的优先级进行设置，一共可以设 32 个级别（0~31 之间的一个数字，0 为最高优先级，31 为最低优先级）。当一个程序在执行时，优先级高的任务优先于优先级任务低的任务，高优先级任务 0 能中断同一资源中较低优先级的程序执行，使较低优先级程序执行被放缓。

注意：在任务优先级等级分配时，请勿分配具有相同优先级的任务。如果还存在其他任务视图先于具有相同优先级的任务，则结果可能不确定且不预知。

如果任务的类型为“循环”，则按照“间隔”中的时间循环执行，具体设置如下图 1-11 所示。

图 1-11 固定周期循环配置

例：假设有 3 个不同的任务，分别对应三种不同的优先等级，具体分配如下。

：任务 1 具有优先级 0 和循环时间 10ms。

：任务 2 具有优先级 1 和循环时间 30ms。

：任务 3 具有优先级 2 和循环时间 40ms。

在控制器内部，各任务的时序关系如图 1-12 所示，具体说明如下：

0~10ms：先执行任务 1（优先级最高），如在本周期内已将程序执行完，剩余时间执行任务 2 程序。但是如果此时任务 2 没有被完全执行完，但时间已经到了第 10ms 时，由于任务 1 是每 10ms 执行一次的且优先级更高，此时将会打断任务 2 的执行。

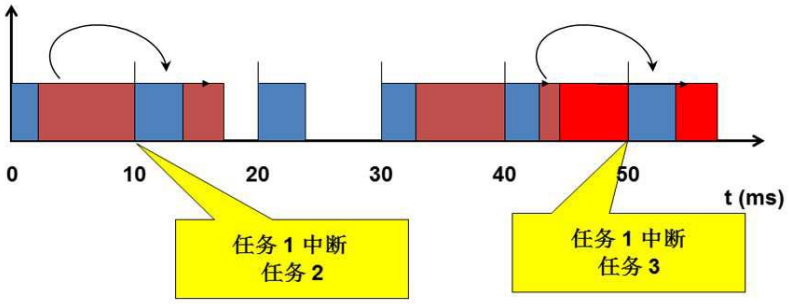
10~20ms：先将任务 1 的程序执行完毕，如有剩余时间，再执行上个周期未完成的任务 2。

20~30ms：由于任务 2 是每 30ms 执行一次的，在 10~20ms 之间任务 2 已经全部执行完毕，此时不需要再执行任务 2，只需将优先级最高的任务 1 执行一次即可。

30~40ms：与之前类似。

40~50ms：此时出现了任务 3，任务 3 的优先级更低，所以只有在确保任务 2 彻底执行完后，才能执行任务 3。

图 1-12 任务优先级中断执行顺序



2、 AX、TM、TP 系列任务的优先级安排

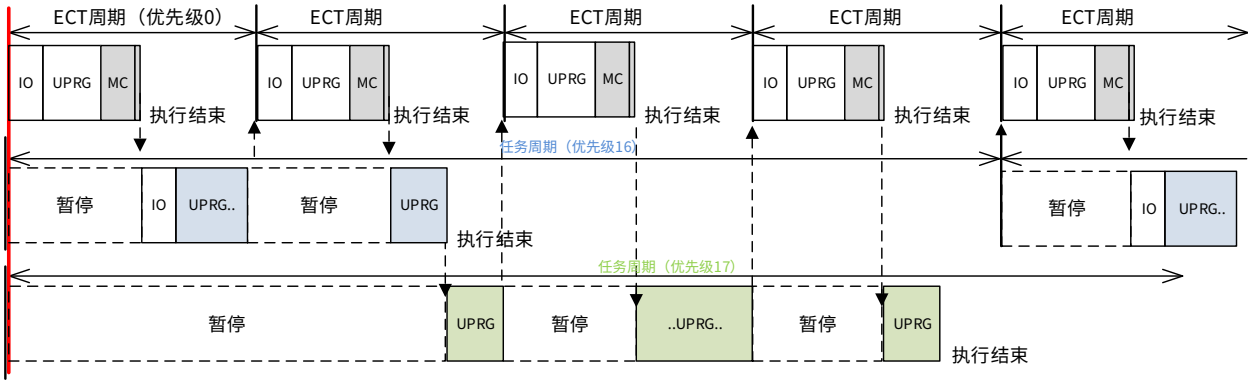
AX、TM、TP 系列控制器的上位机软件创建新的标准工程时在任务配置中默认创建 MainTask，其优先级默认为 1。新创建的任务其优先级默认的也为 1，但是为确保对运动控制等重要任务优先执行，在一些需要高性能运动控制（MC）的应用中，可让控制器的性能得到合理利用。任务优先级顺序设置推荐使用如下顺序（若仅有一个任务，此任务优先级可随意设置）。

表 1-3 任务优先级安排

任务类型	推荐优先级
RTC_Mod 等系统参数模块	31
ModbusTCP	15~30
ModbusRTU	15~30
高速 IO	1~15
模拟量输入输出模块	1~15
温度模块	1~15
EtherCAT	0

优先级设定值越小，优先级越高。高优先级的 POU 可以打断低优先级 POU 的执行，如图 1-13，其中 ECT 代表 EtherCAT。

图 1-13 POU 执行顺序



从图 1-13 可知控制器执行任务时，有一个用户观察不到的时间对准点，如上图左侧，在这个时间点开始，按最高优先级→次高优先级→最低优先级的顺序开始执行。

低优先的任务在执行时，可能被高优先级的任务打断，待高优先级任务执行完成后，返回被打断的任务，继续执行该低优先级任务。

EtherCAT 任务为最高优先级任务，按 EtherCAT 周期进入该任务，完整执行一遍该任务内的所有 POU 后，才返回较低优先级任务。

3、 任务配置中的执行周期设定的要求

中大型可编程控制器系统上位机软件采用多任务方式来执行用户程序的“任务”，而每个“任务”分配了不同的执行周期，有些全局变量可能要在不同的 POU 之间被访问和修改，于是需要对全局变量进行交互同步，也是在任务的“时间对准点”进行的，在设置循环类型任务的周期时，不同类型的循环任务循环时间呈整数倍数的关系。

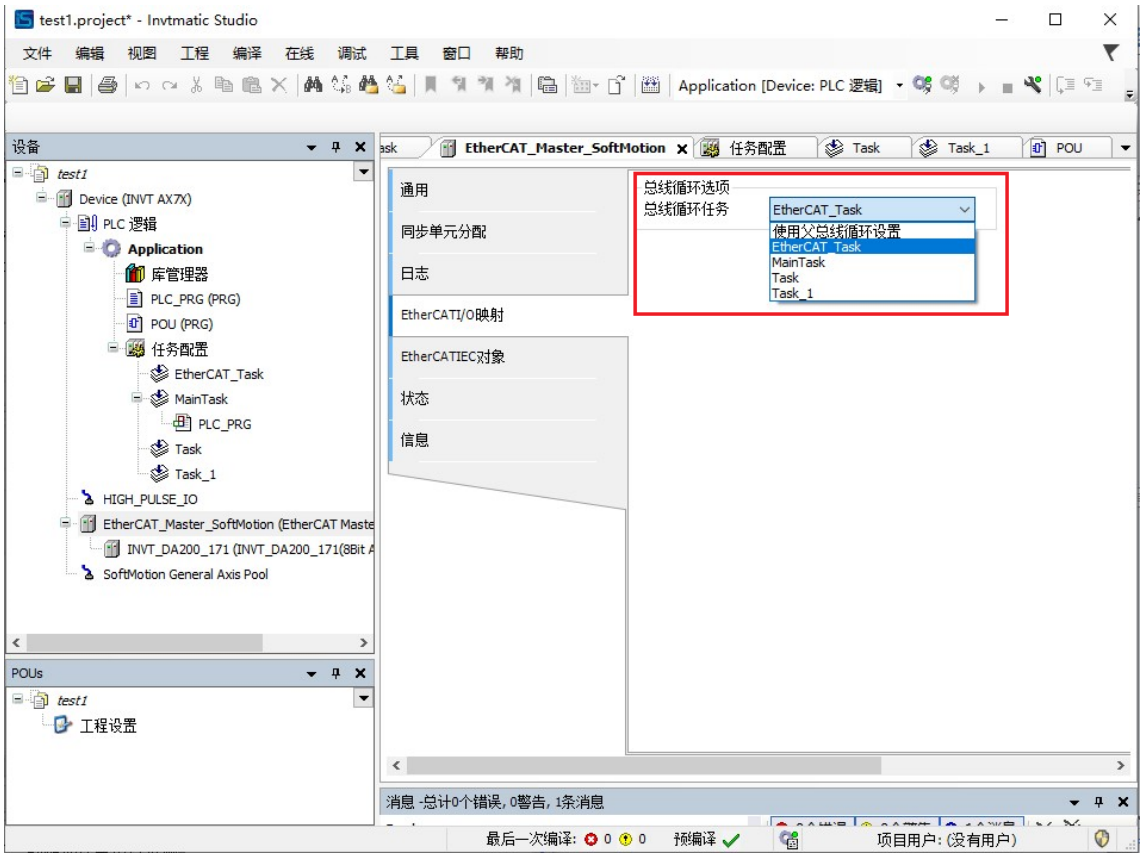
比如，设 EtherCAT 任务周期为 4ms、8ms，而设普通循环任务的周期为 400ms，更低优先级的循环任务周期为 100ms 或 200ms 等等。EtherCAT 周期不要设置为 5ms、7ms、9ms 等，容易造成非 2 的整数倍的异常关系。

4、 子设备总线循环任务（Bus Cycle Options）配置注意事项

在控制器设备“PLC 设置→总线周期→总线周期任务”选项中，总线周期任务：选项列表提供了当前有效工程中任务配置里已定义的任务（例如"MainTask"，"EtherCAT Master"等）。选择其中一个任务作为当前工程的总线周期，或者选择选项<未标明的>，该选项意味着将应用最短的任务周期时间，也就是最快的执行周期。您可以切换到另一种设置，但是请务必注意下列事项：

在修改“<unspecified>”设置之前，您已经清楚其影响。“<unspecified>”是指将应用由设备描述定义的缺省行为。所以请检查这方面的相关描述，缺省方式下，该任务可能被定义为具有最短周期时间，但也可能具有最长周期时间。因此在使用扩展模块以及 EtherCAT 模块时，为提高系统运行稳定性（尤其在使用 EtherCAT_Master_SoftMotion 模块时），在“EtherCAT I/O 映射→总线循环选项”中选择各模块对应的任务，参考例程如图 1-14。

图 1-14 EtherCAT 总线循环任务设置示例



1.6 多子程序的运行

在实际的工程项目中，通常可以将程序按控制流程或者按照设备的对象分割成很多子程序，据此，设计人员将可以按各处理单元分别进行编程。如下图 1-15，以控制流程将主程序拆分为多个不同流程的子程序，拆分的目的主要是使主程序条理更清晰，并且方便今后的调试。

图 1-15 按流程拆分多子程序

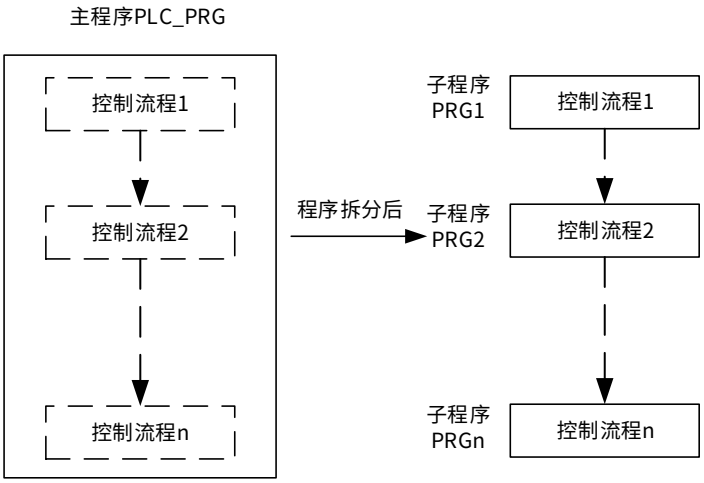


图 1-15 右半部份是按流程进行分类的各子程序 PRG1，PRG2...PRGn，图 1-15 左半部分为主程序 PLC_PRG，在主程序中可以分别调用 PRG1...PRGn 子程序。多子程序运行的方式有两种，第一种在任务配置中添加子程序。第二种方法是在主程序中调用子程序，也是比较常用及灵活的一种方式，如下，分别对两种方式的进行详细说明。

1、 任务配置中添加子程序

用户可以通过在任务配置页面中添加子程序实现多程序的运行。按子程序的执行顺序依次点击“增加调用”进行添加子程序。如图 1-16 所示，添加后，对应的任务即按用户所指定的从上至下的顺序循环执行，也可以通过“上

移”和“下移”功能再对顺序进行手动编辑。

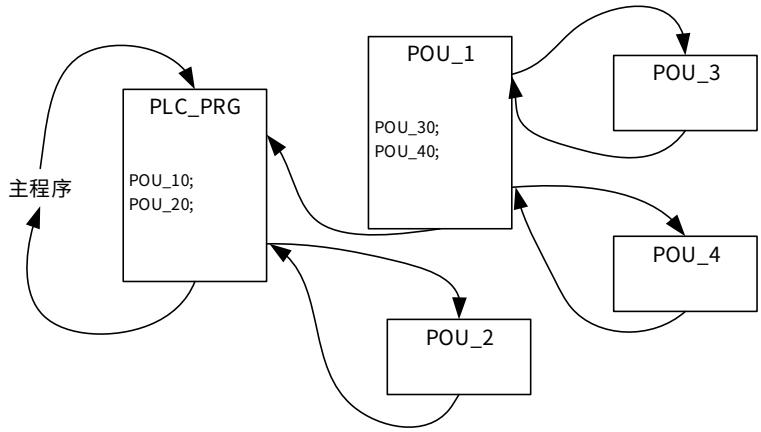
图 1-16 任务中添加子程序

2、 主程序 PLC_PRG 中调用子程序

PLC_PRG 被系统默认为主程序，从某种意义上可以理解为汽车的电瓶，在生产汽车时，将各个零件进行组装，相当于子程序的编写；当汽车组装完成时，就要检查汽车是否可用，如果想启动汽车，就必须通过电瓶来启动汽车的各个部件，如发动机、车灯等，电瓶就相当于启动汽车的入口点。通过这样调用程序使操作性更强及使程序更灵活，能够在程序中加入判断语句等，且能实现嵌套。

PLC_PRG 是一个特殊的 POU，其默认的运行方式为“惯性滑行”。系统默认每个控制周期调用该 POU，用户不需要对其进行额外的任务配置，在任务配置中也可看该 POU 相应的配置。用户可以通过它来实现对其他子程序的调用，更可以在调用时添加必要的条件选择，或实现子程序嵌套，使程序调用更为灵活。如果要实现图 1-17 中的调用关系，可以在主程序 PLC_PRG 中写入如下代码。

图 1-17 POU 调用顺序



如图 1-17 中所示，主程序为 PLC_PRG，该主程序使用的是结构化文本编程语言，其中的程序的内容为 POU_1()；POU_2()。上述程序的主要功能是分别调用执行了 POU_1 和 POU_2 子程序。而 POU_1 中又分别调用了 POU_3 和 POU_4，实际可编程控制器内部实际按如下顺序执行程序：

A、 可编程控制器先执行子程序 POU_1。

B、 由于 POU_1 中依次调用了 POU_3 和 POU_4，故先执行 POU_3。

C、 执行 POU_4，POU_1 执行完成。

D、 最后执行 POU_2，完成一个完整任务周期。

重复上述步骤 A 至 D 即为可编程控制器内部的执行顺序。

1.7 单轴控制

1.7.1 单轴控制编程说明

控制器与伺服轴（如 DA200）配合的运动控制是基于 EtherCAT 总线网络来实现的，每个 EtherCAT 总线周期会进行一次计算、发布一次控制命令，从而实现对伺服的控制，不同与以往的脉冲控制方式，EtherCAT 总线完全通过软件来实现，应用时需要注意以下几点：

- MC 相关的 POU 应配置在 EtherCAT 任务下执行，多数 MC 功能块放在低优先级 Main 任务的 POU 中无法正常运行。

- PDO 配置表需要配置相关的数据对象，否则会由于通信数据对象配置缺省而导致伺服无法运行，也不会有出错报警，排查原因难度加大。
- 控制器可以通过配置 SDO 的方式对伺服进行参数设定。
- MC 功能块实例只能用于唯一的伺服轴控制，如果用于多个会导致控制出错。
- 必须有 MC 功能块来监控运行中的伺服轴，避免程序逻辑跳转无 MC 功能块监控而引发的报错，此类错误通常不易排查。
- 注意调试的安全处理，信号配置与实际应用相符。若伺服系统采用增量式编码器，正常运行之前需要回零，对于在有限范围内运动（如丝杆），应设置极限与安全保护信号。

1.7.2 单轴控制常用的 MC 功能块

MC 功能块（FB）也称为 MC 指令，实际上，用户程序中使用的是 MC 功能块的对象实例，伺服轴通过 MC 对象实例来进行控制，例如：

MC_Power1：MC_Power；//声明实例 MC_Power1

MC_Power1（Axis=Axis1，）；

单轴的控制，一般用于定位的控制，即伺服电机拖动外部机构运动到指定的位置；有时也需要伺服以指定的速度或力矩运行等，在单轴控制中，常用到如下的 MC 功能块：

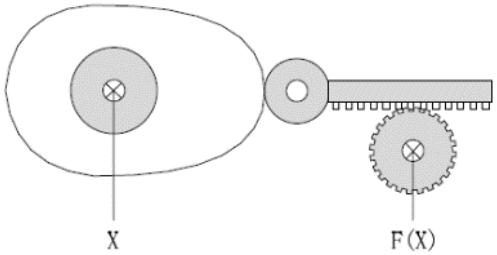
表 1-4 单轴控制常用 MC 功能块

控制操作	需要使用的 MC 指令	说明
伺服使能	MC_Power	运行该指令，使伺服轴使能，才能进行后续的运行控制
绝对定位	MC_MoveAbsolute	命令伺服运行到指定的坐标点
相对定位	MC_MoveRelative	以当前位置为参考，运行指定距离
伺服点动运行	MC_Jog	伺服电机的点动运行，常用于低速试车，用于检验设备或调整伺服电机位置
相对叠加定位	MC_MoveAdditive	在伺服当前运行指令的基础上，再相对运行指定距离
速度控制	MC_MoveVelocity	命令伺服以指定的速度运行
伺服暂停	MC_Halt	命令伺服暂停运行，若 MC_Movexxx 再次触发，伺服可以再运行
紧急停机	MC_Stop	命令伺服紧急停机，只有 stop 命令复位后，触发 MC_Movexxx，伺服才可以再运行
告警复位	MC_Reset	当伺服出现告警停机后，运行该指令进行复位
伺服原点回归	MC_Home	命令伺服开始原点回归操作，应用系统的原点信号、两侧极限信号等都接在伺服的 DI 端口
控制器原点回归	MC_Homing	控制系统开始原点回归操作，应用系统的原点信号、两侧极限信号等都接在控制器的 DI 端口

1.8 凸轮同步控制

电子凸轮(英文简称 ECAM)是利用构造的凸轮曲线来模拟机械凸轮，以达到机械凸轮系统相同的凸轮轴与主轴之间相对运动的软件系统。电子凸轮可以应用在诸如汽车制造、冶金、机械加工、纺织、印刷、食品包装等各个领域。电子凸轮曲线是以主轴脉冲（主动轴）输入为 X，伺服电机（凸轮轴）对应输出为 Y=F(X)的一个函数曲线。

图 1-18 电子凸轮



可编程控制器电子凸轮功能有如下特点：

- 凸轮曲线易于绘制：可通过凸轮表、凸轮曲线或数组描述凸轮且支持多个凸轮表选择和运行中动态切换。
- 凸轮曲线易于修正：支持对运行中的凸轮表进行动态修改。
- 支持一主多从：一个主轴可有多个从轴与之对应。
- 凸轮挺杆：允许有多个凸轮挺杆、多个设置区间。
- 凸轮离合器：用户程序可使之进入与退出凸轮运行。
- 特有功能：支持虚拟主轴、相位偏移和输出叠加。

注意：所谓“在线修改 CAM 曲线”，是指用户编写的程序在执行过程中，根据控制特性的需要，对 CAM 曲线的关键点坐标，进行的修改。修改的内容，一般是修改关键点坐标，但也可以修改关键点的个数、修改主轴的距离范围等。

可编程控制器电子凸轮有三个控制要素：

1、 主轴：同步控制的参考轴。

2、 从轴：按照非线性特性跟随主轴运动的伺服轴。

3、 凸轮表：描述主轴-从轴相对位置与范围、周期性等的数据表或凸轮曲线。

常用的电子凸轮相关功能块见下表：

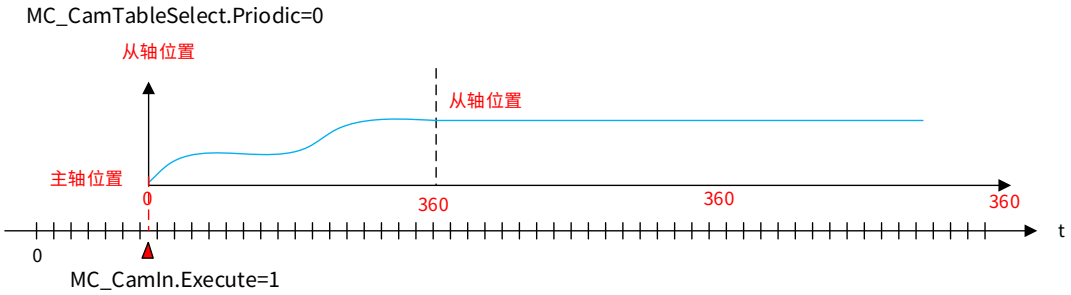
表 1-5 常用电子凸轮功能块

MC 指令	说明
MC_CamTableSelect	运行该指令，关联主轴、从轴凸轮表三者关系
MC_CamIn	让从轴进入凸轮运行
MC_CamOut	让从轴退出凸轮运行
MC_Phasing	主轴相位修改

1.8.1 凸轮表的周期模式

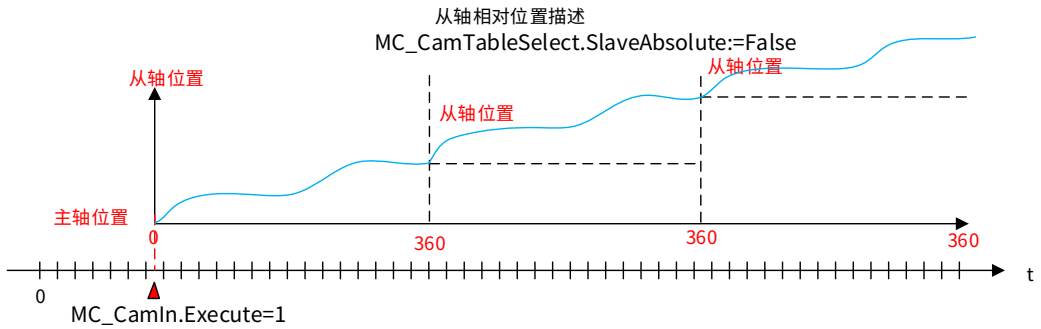
单周期模式（Periodic:=0）：凸轮表周期运行完毕后，从轴脱离凸轮运行状态，如图 1-9 所示。

图 1-19 单周期模式



周期模式（Periodic:=1）：凸轮表周期运行完毕后，从轴又开始下一凸轮周期的运行，直到用户程序命令其退出凸轮运行状态，如图 1-0 所示。

图 1-20 周期模式



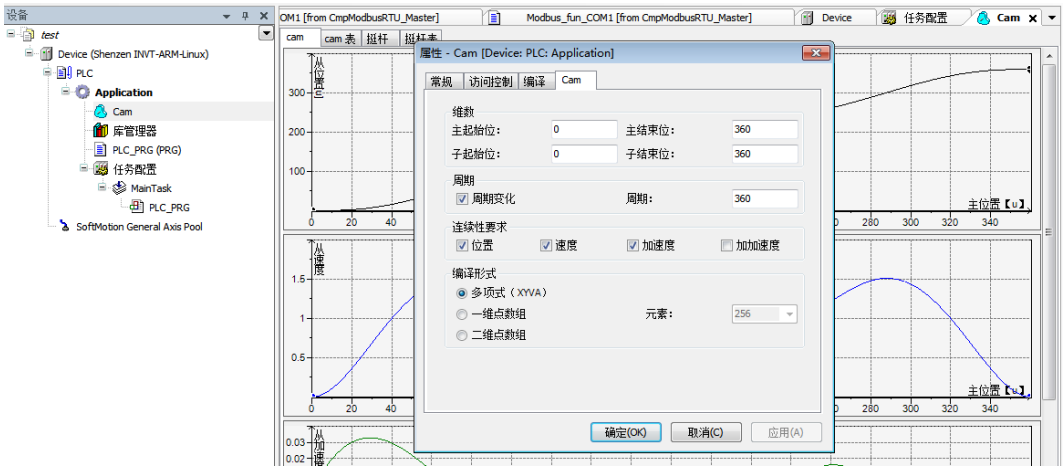
1.8.2 凸轮表的输入方法

新建一个凸轮表时，系统会自动生成一个最简单的凸轮曲线，用户在此基础上进行编辑，形成自己的 CAM 曲线表。

用户可以增减凸轮曲线的关键点个数或修改关键点的坐标。

凸轮曲线两个关键点之间的线型可设置为直线或五次多项式，并且系统会对每条曲线作最佳优化，以尽量减小速度和加速度的突变。

图 1-21 凸轮曲线



1.8.3 凸轮表的数据结构

在 Invtmatic Studio 中，对每一个 CAM 表，都有描述该 CAM 表的数据结构，描述该 CAM 表的特征数据。下图为“CAM0”凸轮表的描述数据结构，请注意其结构各变量名称。

图 1-22 凸轮表数据结构

cam | cam 表 | 挺杆 | 挺杆表

	X	Y	V	A	J	段类型	最小(...	最大(...	最大(|...	最大(|...
	0	0	0	0	0					
						Poly5	0	120	1.5120...	0.0328...
	120	120	1	0	0					
						Poly5	120	240	1	0
	240	240	1	0	0					
						Poly5	240	360	1.512	0.0328...
	360	360	0	0	0					

Invtmatic Studio 内部有一个数据结构来描述 CAM 凸轮表的特征，我们也可以手动编写一个 CAM 表，或者我们可以通过数据结构的访问操作，对 CAM 特性数据进行修改。

注意：我们在声明 CAM0 凸轮表时，系统自动默认声明了全局变量类型的 CAM0 数据结构，同时声明了 CAM0_A[i]数组。

例如在用户程序中，修改 CAM0 凸轮表关键点个数或坐标：

CAM0.nElements：=10；将关键点个数改为 10 个

CAM0.xEnd：=300；将主轴的结束点改为 300

例如在用户程序中，修改其中 2 个关键点的坐标：

CAM0_A[2].dx：=10

CAM0_A[2].dy：=30

CAM0_A[2].dv：=1

CAM0_A[2].da：=0

CAM0_A[3].dx：=30

CAM0_A[3].dy：=50

CAM0_A[3].dv：=1

CAM0_A[3].da：=0

1.8.4 凸轮表的引用与切换

CAM 凸轮表在控制器内部是用一个数组来保存，可以通过特定的 MC_CAM_REF 变量类型来指向。例如声明：

凸轮表 q：MC_CAM_REF；

可以给该变量赋值，也可认为是将其指向某具体的凸轮表：

凸轮表 q：=Cam0；//指向所需的凸轮表

凸轮表 q：MC_CAM_REF；//凸轮表指针；

TableID：uint；//凸轮表选择命令，可由 HMI 设置；

CaseTableIDof

0：凸轮表 q：=凸轮表 A；

1：凸轮表 q：=凸轮表 B；

2：凸轮表 q：=凸轮表 C；

End_case

MC_CamTableSelect_0(//凸轮关系

Master：=虚主轴

Slave：=凸轮从轴

CamTable：=凸轮表 q

Execute：=bSelect，//上升沿触发凸轮表选择

Periodic：=TRUE，

MasterAbsolute：=FALSE，

SlaveAbsolute：=FALSE)；

上面的例程，利用该 MC_CAM_REF 变量的赋值运算，就可以实现多个凸轮表的切换操作了。

1.9 编程建议

在 CODESYS 中可以对任务的优先级进行设置，一共可以设 32 个级别（0~31 之间的一个数字，0 是最高优先级，31 是最低优先级）。当一个程序在执行时，优先级高的任务优先于优先级低的任务，高优先级任务 0 能中断同一资源中优先级较低的程序执行，使优先级较低的程序执行被放缓。在任务分配时，请勿分配优先级相同的任务。

注意：

- 一个任务配置，只能设一种优先级、循环类型、间隔时间，若需要不同的执行特性，需要添加多个任务配置。
- 一个任务配置内，可以包含多个 POU，执行时按该任务中 POU 添加顺序执行。
- EtherCAT 总线通讯的任务优先级一般设置为最高优先级 0，扫描周期一般设置为 1~4ms，设定值越小，运动控制的精度越高，轴较多时，适当放大扫描周期，否则 CPU 负载率会较高，可能出现轴丢失的情况。

任务配置–运行状态监视

进入在线模式后，用户可使用系统自带的监视器，对任务的平均、最大、最小循环时间等任务执行相关参数进行监控。在项目开发阶段，可以使用该功能测试程序最大、最小、平均循环时间，用于测定程序的稳定性及对程序设定任务周期时间的优化。

任务配置中，应遵循如下时间设定关系，按照此设定方法可以更好地优化任务周期及“看门狗”时间，以保证程的稳定性和实时性。

“看门狗”触发时间＞固定周期循环时间＞程序最大循环时间

如果循环时间比固定周期循环时间长，CPU 会检测出程序有超出计数的情况，此时，会影响程序的实时性。如果程序循环时间比看门狗时间设定长的情况下，CPU 会检测出看门狗故障，会停止程序的执行。

2 EtherCAT 运行机制

2.1 EtherCAT 运行原理

2.1.1 EtherCAT 协议介绍

EtherCAT（Ethernet for Control Automation Technology）是一种突破了其他以太网解决方案固有局限性的技术，具有以下几个关键特性：

高效的数据处理：传统以太网方案需要接收数据包、解码，并将过程数据复制到各个设备中，而 EtherCAT 从站设备在报文经过其节点时就能读取带有相应寻址信息的数据，同时在报文经过时插入输入数据。这种处理方式使报文只有几纳秒的时间延迟。

数据传输流程：由主站发出的帧被传输并经过所有从站，直到网段或分支的最后一个从站。最后一个设备检测到其开放端口时，将帧返回给主站。由于发送和接收的以太网帧压缩了大量设备数据，可用数据率可达 90%以上，100Mb/s 全双工特性完全得以利用，有效数据率可以达到大于 100Mb/s。

主站和从站设计：EtherCAT 主站采用标准的以太网介质存取控制器（MAC），无需额外的通讯处理器，这意味着任何集成了以太网接口的设备控制器都可以实现 EtherCAT 主站，无论操作系统或应用环境如何。EtherCAT 从站则采用 EtherCAT 从站控制器（ESC）来高速动态地处理数据，网络性能不依赖于从站使用的微处理器性能，因为所有通讯都是在 ESC 硬件中完成的。过程数据接口（PDI）为从站应用层提供了一个双口随机存储器（DPRAM）来实现数据交换。

精确同步：在需要广泛同步动作的分布过程中，如多个伺服轴同时执行联动任务时，精确同步尤为重要。分布时钟的精确校准是实现同步的有效解决方案。相比完全同步通讯，分布式校准时钟在某种程度上具备错误延迟的容错性。

通过这些特性，EtherCAT 提供了一种高效、灵活且可靠的工业以太网解决方案，适用于各种自动化控制应用。

2.1.2 工作计数器 WKC

每个 EtherCAT 报尾拥有一个 16 位的工作计数器 WKC。WKC 是用于记录对 EtherCAT 从站设备读写次数的工作计数器，EtherCAT 从站控制器在硬件中计算 WKC，主站接收到返回数据后检查子报文中的 WKC。如果不等于预期值，则表示该子报文没有被正确的处理。子报文经过某一个从站时，如果是单独的读或写操作，WKC 加 1。如果是读写操作，读成功时 WKC 加 1，写成功时 WKC 加 2，全部完成时 WKC 加 3，WKC 为各个从站处理结果的累加。关于 WKC 增量的描述如表 2-1 所示。

表 2-1 WKC 增量

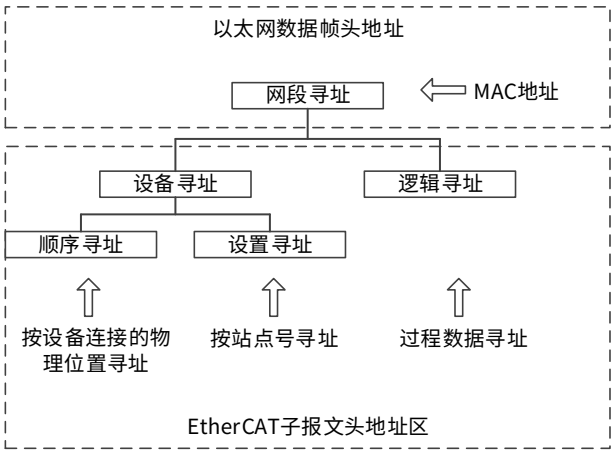
指令	数据类型	增量
读指令	读取失败	无变化
	成功读取	+1
写指令	写入失败	无变化
	成功写入	+1
读/写指令	不成功	无变化
	成功读取	+1
	成功写入	+2
	成功读写	+3

2.1.3 寻址方式

EtherCAT 通信由主站发送 EtherCAT 数据帧读写从站设备的内部存储区来实现，EtherCAT 报文使用多种寻址方式来操作 ESC 内部存储区实现多种通信服务。EtherCAT 的寻址方式如图 2-1 所示。一个 EtherCAT 网段相当于一个以太网设备，主站首先使用以太网数据帧头的 MAC 地址寻址到网段，然后使用 EtherCAT 子报文头中的 32 个位地址寻址到段内设备。段内寻址可以使用两种方式：设备寻址和逻辑寻址。设备寻址针对某一个从站进行读写操作。

逻辑寻址面向过程数据，可以实现多播，同一个子报文可以读写多个从站设备。

图 2-1 EtherCAT 网络寻址模式



2.1.3.1 网段寻址

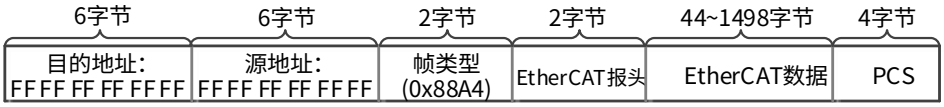
根据 EtherCAT 主站及其网段的连接方式不同，可以使用如下两种方式寻址到网段。

1、 直连模式

一个 EtherCAT 网段直接连接到主站设备的标准以太网端口，如图 2-2 所示。此时，主站使用广播 MAC 地址，EtherCAT 数据帧如图 2-3 所示。

图 2-2 直连模式中的 EtherCAT 网段

图 2-3 直连模式中的 EtherCAT 网段寻址地址内容



2、 开放模式

EtherCAT 网段连接到一个标准以太网交换机上，如图 2-4 所示。此时，一个网段需要一个 MAC 地址，主站发送的 EtherCAT 数据帧中的地址是它所控制的网段的 MAC 地址，如图 2-5 所示。EtherCAT 网段内的第一个从站设备有 ISO/IEC 8802.3 的 MAC 地址，这个地址表示了整个网段，这个从站称为段地址从站，它能够交换以太网内的目的地址区和源地址区。如果 EtherCAT 数据帧通过 UDP 传送，这个设备也会交换源和目的的 IP 地址、以及源和目的的 UDP 端口号，使响应的数据帧完全满足 UDP/IP 协议标准。

图 2-4 开放模式中的 EtherCAT 网段

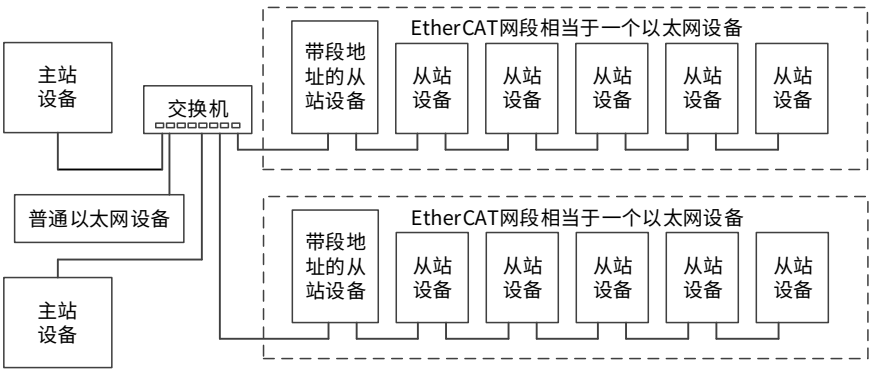
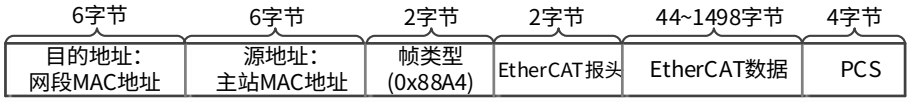


图 2-5 开放模式中的 EtherCAT 网段寻址地址内容

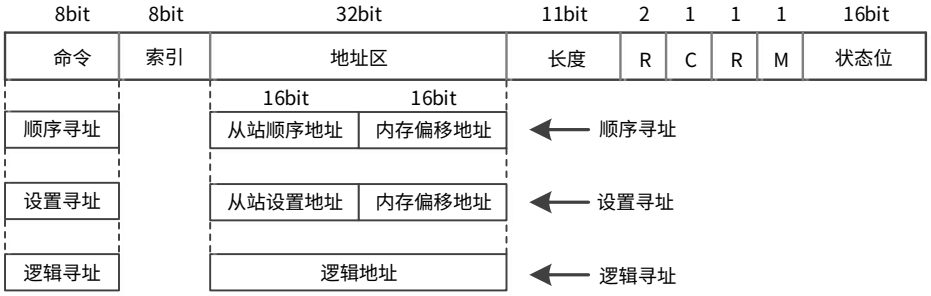


2.1.3.2 设备寻址

在设备寻址时，EtherCAT 子报文头内的 32 位地址分为 16 位从站设备地址和 16 位从站设备内部物理存储空间地址，如图 2-6 所示。16 位从站设备地址可以寻址 65535 个从站设备，每个设备内最多可以有 64 个本地地址空间。

设备寻址时，每个报文只寻址唯一的一个从站设备，但它有两种不同的设备寻址机制（顺序寻址、设置寻址）。

图 2-6 EtherCAT 的设备寻址结构



1、 顺序寻址

顺序寻址时，从站的地址由其在网段内的连接位置确定，用一个负数来表示每个从站在网段内由接线顺序决定的位置。顺序寻址子报文在经过每个从站设备时，其顺序地址加 1；从站在接收报文时，顺序地址为 0 的报文就是寻址到自己的报文。由于这种机制在报文经过时更新设备地址，所以又被称为“自动增量寻址”。

如图 2-7 所示网段中有 3 个从站设备，其顺序寻址的地址为 0、-1、-2，以此类推。主站使用顺序寻址访问从站时子报文的地址变化如图 2-8 所示。主站发出 3 个子报文分别寻址 3 个从站，其中的地址分别是 0、-1 和-2，如图 2-8 数据帧为 1。数据帧达到从站①时，从站①检查到子报文 1 中的地址为 0，从而得知子报文 1 就是寻址到自己的报文。数据帧经过从站①后，所有的顺序地址都增加 1，称为 1、0 和-1，如图 2-8 中的数据帧 2。到达从站②时，从站②发现子报文 2 中的顺序地址为 0，即为自己的报文。同理，后续的从站都按此方法来寻址。如图 2-7 在实际工程应用中，顺序寻址主要用于启动阶段，主站配置站点地址给各个从站。此后，可以使用与物理位置无关的站点地址来寻址从站。使用顺序寻址机制能自动为从站设定地址，如图 2-8。

图 2-7 顺序寻址的从站地址

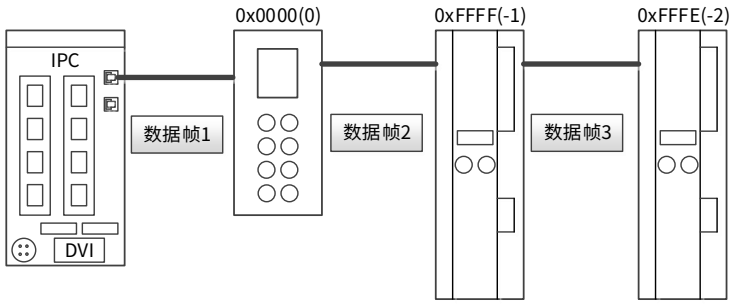


图 2-8 顺序寻址时子报文地址的变化

2、 设置寻址

设置寻址时，从站的地址与其在网段内的连续顺序无关。如图 2-9 所示，地址可以有主站在数据链路启动阶段配置给从站，也可以由从站在上电初始化阶段的配置数据装载，然后由主站在链路启动阶段使用顺序寻址方式读取各个从站的设置地址。其报文结构如图 2-10 所示。

图 2-9 设置寻址时的从站地址

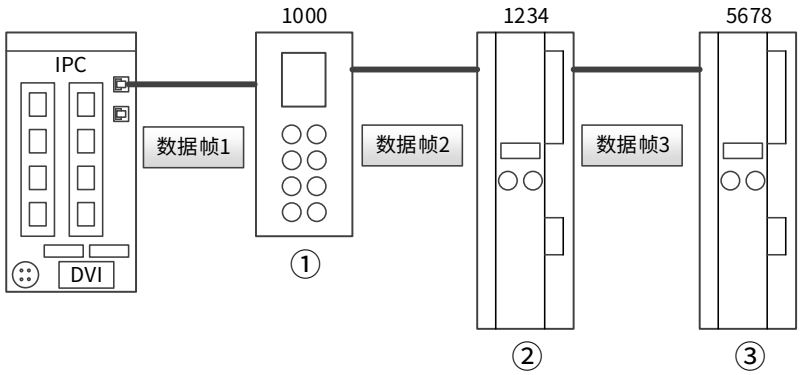
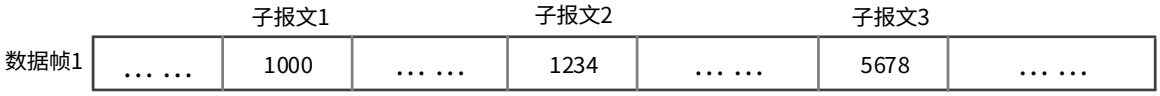


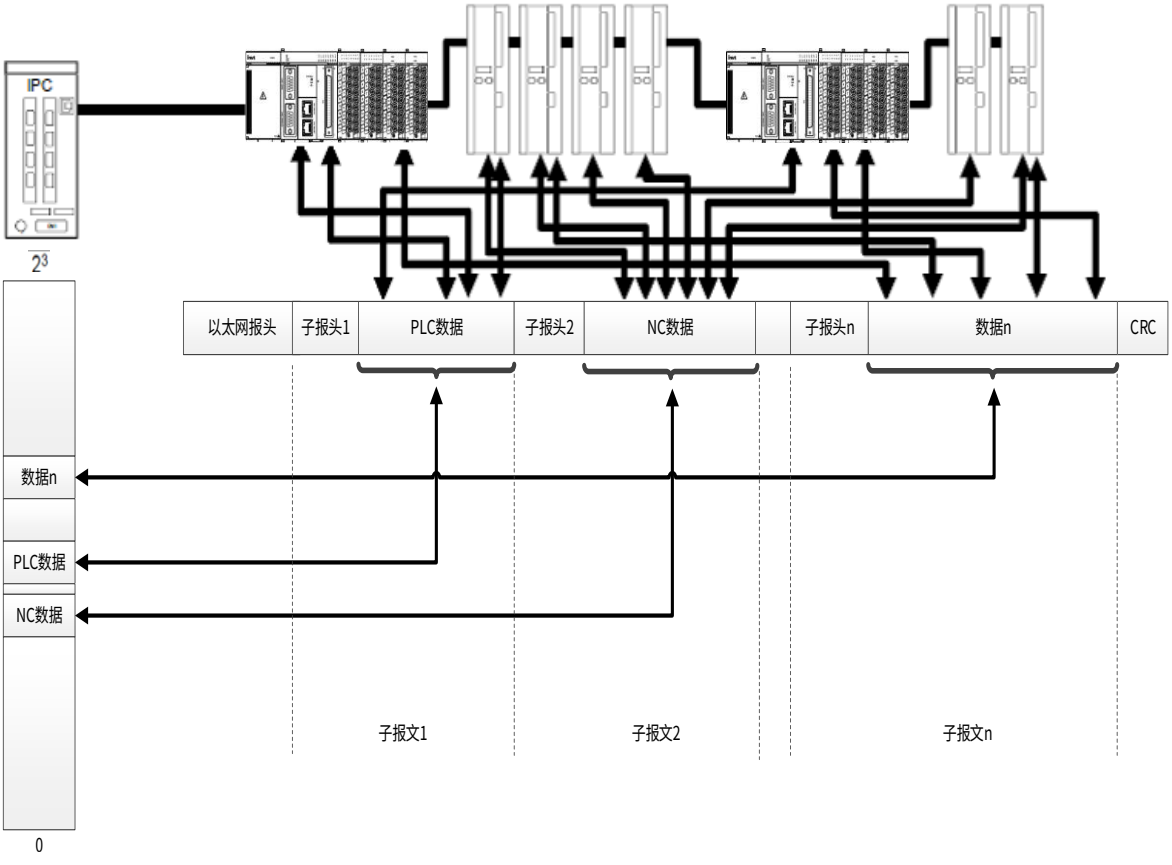
图 2-10 设置寻址时的报文结构



2.1.3.3 逻辑寻址

逻辑寻址时，从站地址并不是单独定义的，而是使用寻址段内 4GB 逻辑地址空间中的一段区域。报文内的 32 位地址区作为整体数据逻辑地址完成设备的逻辑寻址。逻辑寻址方式由现场总线内存管理单元（FMMU，Fieldbus Memory Management Unit）实现，FMMU 功能位于每一个 ESC 内部，将从站本地物理存储地址映射到网段内逻辑地址，其原理图如图 2-11 所示。

图 2-11 现场总线内存管理单元（FMMU）运行原理



从站设备收到一个数据逻辑寻址的 EtherCAT 子报文时，检查是否有 FMMU 单元地址匹配。如果有，它将输入类型数据插入到 EtherCAT 子报文数据区的对应位置，以及从 EtherCAT 子报文数据区的对应位置抽取输出类型数据。

2.1.4 分布时钟

2.1.4.1 分布时钟概念

精确同步对于同时动作的分布式过程而言尤为重要。例如，几个伺服轴同时执行协调运动时，便是如此。分布时钟机制能够使所有的从站都同步于一个参考时钟。主站连接的第一个具有分布时钟功能的从站作为参考时钟，以参考时钟来同步其他设备和主站的从时钟。为了实现精确的时钟同步控制，必须测量和计算数据传输延时和本地时钟偏移，并补偿本地时钟的漂移。同步时钟所涉及到如下 6 个概念。

1、 系统时间

系统时间是分布时钟使用的系统计时。系统时间从 2001 年 1 月 1 日零点开始，使用 64 位二进制变量表示，单位为纳秒（ns），最大可以计时 500 年。也可以使用 32 位二进制变量表示，32 位时间值最大可以表示 4.2s，通常用于通信和时间戳。

2、 参考时钟和从时钟

EtherCAT 协议规定主站连接的第一个具有分布时钟功能的从站作为参考时钟，其他从站的时钟称为从时钟。参考时钟被用于同步其他从站设备的从时钟和主站时钟。参考时钟提供 EtherCAT 系统时间。

3、 主站时钟

EtherCAT 主站也具有计时功能，称为主站时钟。主站时钟可以在分布时钟系统中作为从站时钟被同步。在初始化阶段，主站可以按照系统时间的格式发送主站时钟给参考时钟从站，使分布时钟使用系统时间计时。

4、 本地时钟、其初始偏移量和时钟漂移

每一个 DC 从站都有本地时钟，本地时钟独立运行，使用本地时钟信号计时。系统起动时，各从站的本地时钟和参考时钟之间有一定的差值，称为时钟初始偏移量。在运行过程中，由于参考时钟和 DC 从站时钟使用各自的时钟源等原因，它们的计时周期存在一定的漂移，这将导致时钟运行不同步，本地时钟产生漂移。因此，必须对时钟初始偏移和时钟漂移进行补偿。

5、 本地系统时间

每个 DC 从站的本地时钟经过补偿和同步之后都产生一个本地系统时间，分布时钟同步机制就是使各个从站的本地系统时间保持一致。参考时钟也是相应从站的本地系统时钟。

6、 传输延时

数据帧在从站之间传输时会产生一定的延迟。其中包括设备内部和物理连接延迟。所以在同步从站时钟时，应该考虑参考时钟与多个从站时钟之间的传输延时。

2.1.4.2 时钟同步过程

时钟同步有如下三个步骤组成：

步骤1 传输延时测量

分布时钟初始化时主站会给所有方向的从站初始化传输延时，并计算得到从时钟与参考时钟之间的偏差值，将其写入从站时钟站。

步骤2 参考时钟偏移补偿（系统时间）

每个从站的本地时钟会与系统时间进行比较，然后将不同的比较结果分别写入不同的从站内，这样所有的从站都会得到绝对的系统时间。

步骤3 参考时钟漂移补偿

时钟漂移补偿及本地时间是用于定期补偿本地时钟的误差及微调。如下的图形说明了补偿计算的两个应用案例，图 2-12 为系统时间小于从站本地时钟的案例。图 2-13 为系统时间大于本地时间的案例。

图 2-12 时钟同步过程：系统时间<本地时间

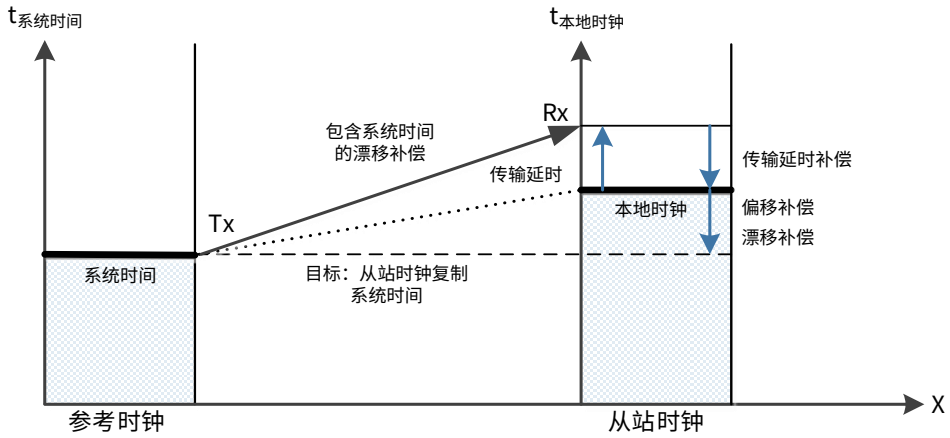
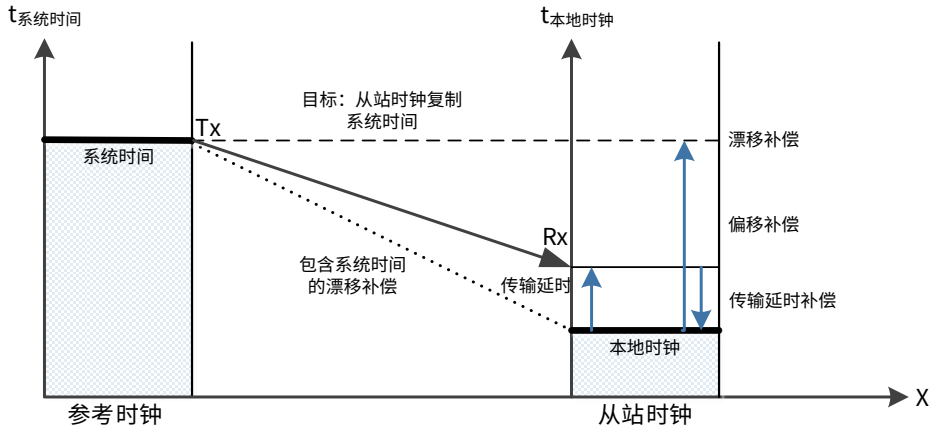
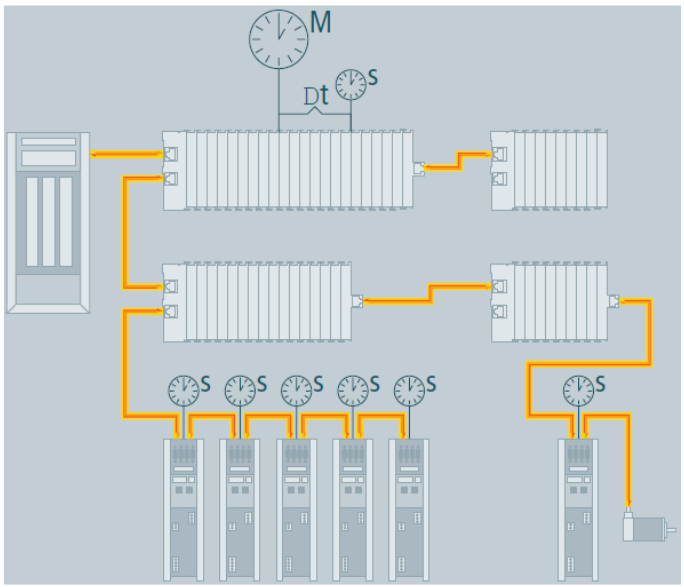


图 2-13 时钟同步过程：系统时间>本地时间



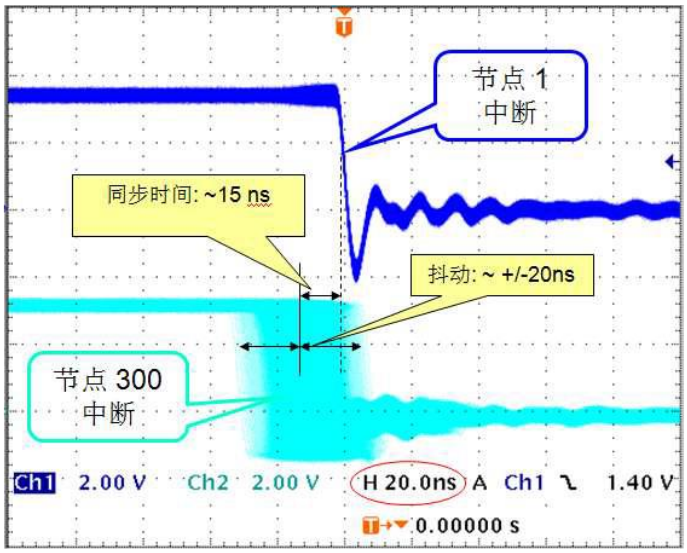
采用 EtherCAT，数据交换就完全基于纯硬件机制。由于通讯采用了逻辑环结构（借助于全双工快速以太网的物理层），主站时钟可以简单、精确地确定各个从站时钟传播的延迟偏移，反之亦然。分布时钟均基于该值进行调整，这意味着可以在网络范围内使用非常精确的、小于 1μs 的、确定性的同步误差时间基。其结构如图 2-14 所示。

图 2-14 同步时钟原理



比如两设备之间相差 300 个节点，线缆的长度为 120m，使用示波器抓取其通信信号，其结果如图 2-15 所示。

图 2-15 同步时钟性能测试



该功能对于运动控制是非常重要的，它通过连续检测到的位置值计算出速度，当采样时间非常短时，即使是位置测量出现一个很小的瞬时抖动，也会导致速度计算出现较大的阶跃变化。在 EtherCAT 中，引入时间戳数据类型作为一个逻辑延伸，可以为测量值附加高分辨率的系统时间，而以太网所提供的巨大带宽使这成为可能。

2.2 EtherCAT 通信模式

在实际自动化控制系统中，应用程序之间通常有两种数据交换形式：时间关键和时间非关键。

时间关键表示特定的动作必须在确定的时间窗口内完成。如果不能再要求的时间窗口内完成通讯，则有可能引起控制失效。时间关键的数据通常周期性的发送，称为周期性过程数据通信。时间非关键数据可以非周期性发送，在 EtherCAT 中采用非周期性邮箱（Mailbox）数据通信。

2.2.1 周期性过程数据通信

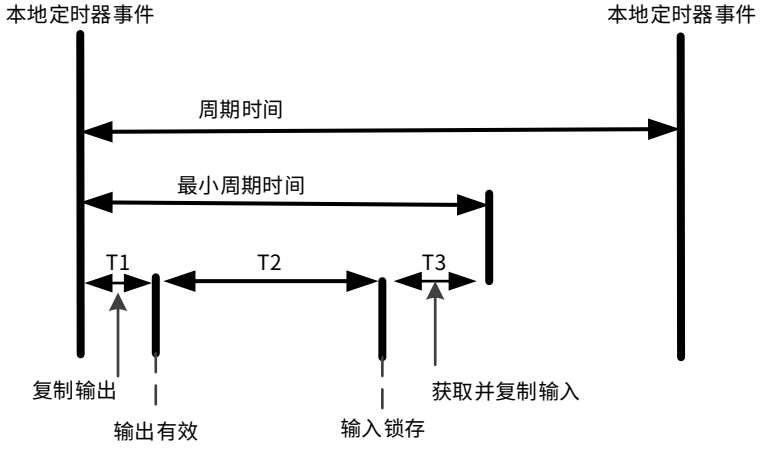
主站可以使用逻辑读、写或读写命令同时控制多个从站。在周期性数据通信模式下，主站和从站有多种同步运行模式。

2.2.1.1 从站设备同步模式

1、 自由运行模式

在自由运行模式下，本地控制周期由一个本地定时器中断产生。周期时间可以由主站设定，这是从站的可选功能。自由运行模式的本地周期如图 2-16 所示。其中 T1 为本地微处理器从 EtherCAT 从站控制器复制数据并计算输出数据的时间；T2 为输出硬件延时；T3 为输入锁存偏移时间。这些参数反映了从站的时间响应性能。

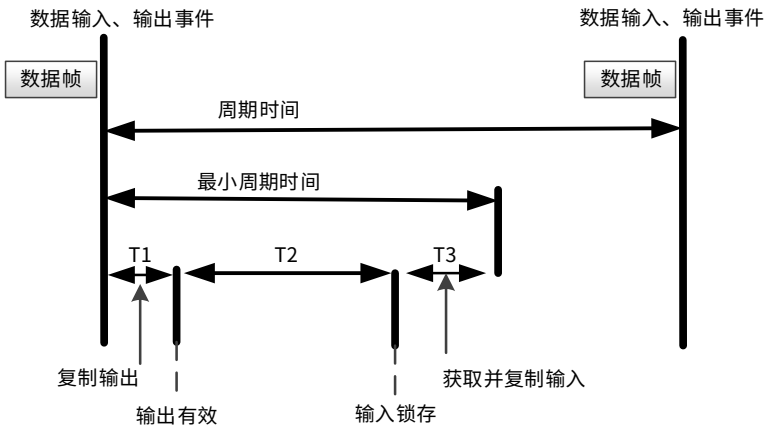
图 2-16 自由运行模式的本地周期



2、 同步于数据输入、输出事件

本地周期在发生数据输入或输出事件的时候触发，如图 2-17 所示，主站可以将过程数据帧的发送周期写给从站，从站可以检查是否支持这个周期时间或对周期时间进行本地优化，从站可以选择支持这个功能。通常同步与数据输出事件，如果从站只有输入数据，则数据同步于输入事件。

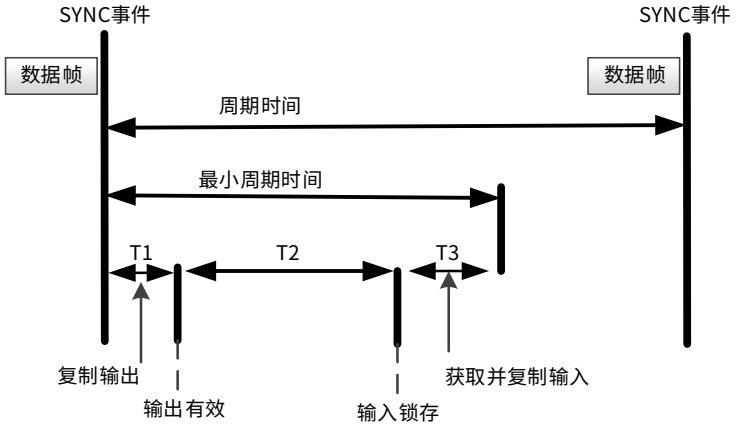
图 2-17 同步于数据输入、输出事件的本地周期



3、 同步于分布式时钟同步事件

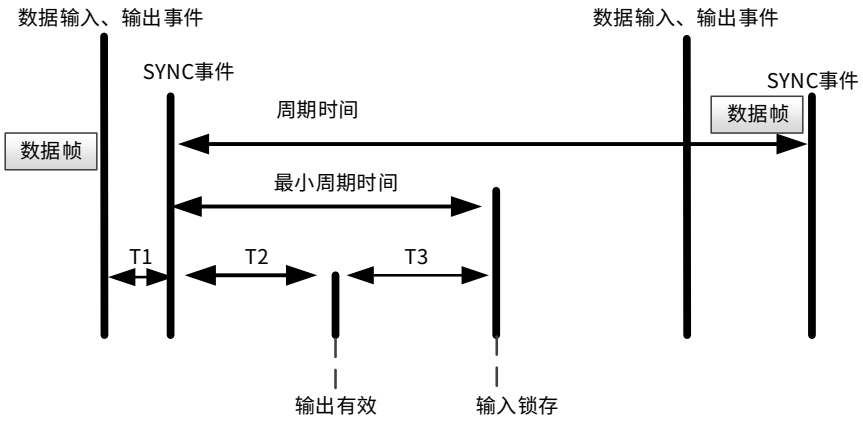
本地周期由 SYNC 事件触发，如图 2-18 所示，主站必须在 SYNC 事件之前完成数据帧的发送，为此要求主站时钟也要同步于参考时钟。

图 2-18 同步于 SYNC 事件的本地周期



为了进一步优化从站同步性能，主站应该在数据收发事件发生时从接收到的过程数据帧复制输出信息，然后等待 SYNC 信号到达后继续本地操作。如图 2-19 所示，数据帧必须比 SYNC 信号提前 T1 时间到达。从站在 SYNC 事件之前已经完成数据交换和控制计算，接收 SYNC 信号后可以马上执行输出操作，从而进一步提高同步性能。

图 2-19 优化的同步于 SYNC 事件的本地周期



2.2.1.2 主站设备同步模式

1、 周期性模式

在周期性模式下，主站周期性的发送过程数据帧。主站周期通常由一个本地定时器控制。从站可以运行在自由运行模式或同步于接收数据事件模式。对于运行在同步模式的从站，主站应该检查相应的过程数据帧的周期时间，保证其大于从站支持的最小周期时间。

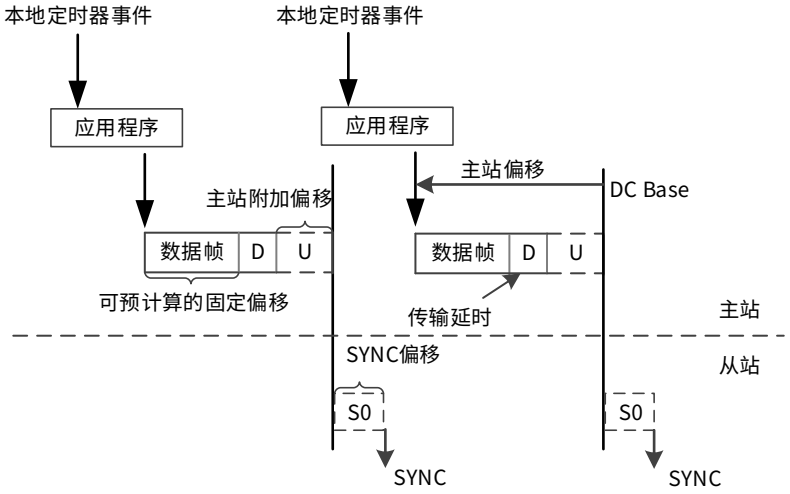
主站可以以不同的周期时间发送多种周期性的过程数据帧，以便获得最优化的带宽。例如，使用比较短的周期发送运动控制数据，比较长的周期用来发送 I/O 数据。

2、 DC 模式

在 DC 模式下，主站运行与周期性模式类似，只是主站本地周期应该和参考时钟同步。主站本地定时器应该根据发布参考时钟的 ARMW 报文进行调整。在运行过程中，用于动态补偿时钟漂移的 ARMW 报文返回主站后，主站时钟可以根据读回的参考时钟时间进行调整，使之大致同步与参考时钟时间。

DC 模式下，所有支持 DC 的从站都应该同步与 DC 系统时间。主站也应该使其他通讯周期同步于 DC 参考时钟时间。本地周期与 DC 参考时钟同步的工作原理如图 2-20 所示。

图 2-20 主站 DC 模式



主站本地运行由一个本地定时器启动。本地定时器应该比 DC 参考时钟定时存在一个提前量，提前量为以下时间之和。

- 控制程序执行时间
- 数据帧传输时间
- 数据帧传输延时 D
- 附加偏移 U（与各从站延时时间的抖动和控制程序执行时间的抖动值有关，用于主站周期的调整）

2.2.2 非周期性邮箱数据通信

EtherCAT 协议中非周期性数据通信称为邮箱数据通信，它可以双向进行—主站到从站和从站到主站。它支持全双工、两个方向独立通信和多用户协议。从站到从站的通信由主站作为路由器来管理。邮箱通信数据头中包括一个地址域，使主站可以重寄邮箱数据。邮箱数据通信是由实现参数交换的标准方式，如果需要配置周期性过程数据通信或需要其他非周期性服务时需要使用邮箱数据通信。

邮箱数据报文结构如图 2-21 所示。通常邮箱通信值对应一个从站，所以报文中使用设备寻址模式。其数据头中各数据元素的解释如表 2-2 所列。

图 2-21 邮箱数据单元结构

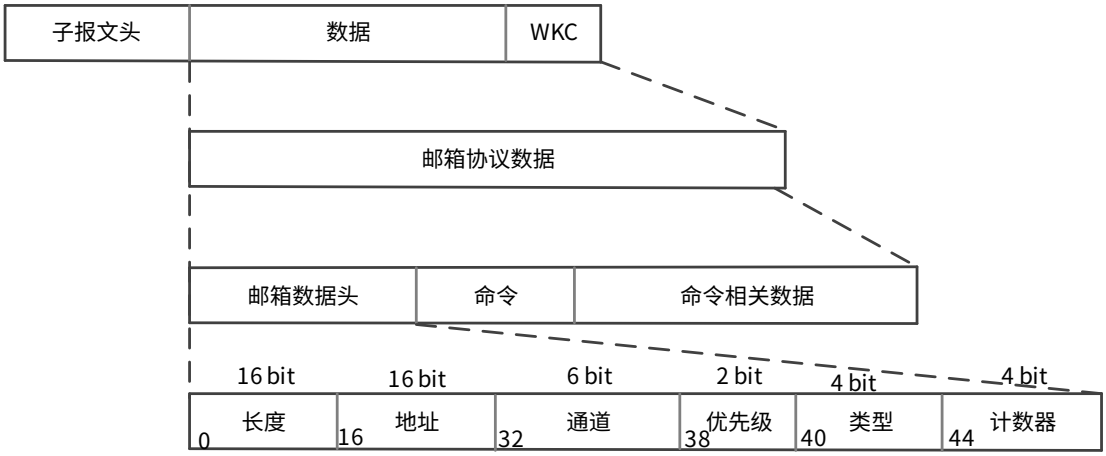


表 2-2 邮箱数据头

数据元素	位数	描述
长度	16 位	跟随的邮箱服务数据长度
地址	16 位	主站到从站通信时，为数据源从站地址 从站到从站通信时，为数据目的从站地址
通道	6 位	保留
优先级	2 位	保留
类型	4 位	邮箱类型，即后续的协议类型： 0：邮箱通信出错 2：EoE（Ethernet over EtherCAT） 3：CoE（CANopen over EtherCAT） 4：FoE（File Access over EtherCAT） 5：SoE（Sercos over EtherCAT） 15：VoE（Vendor Specific Profile over EtherCAT）
计数器 Ctr	4 位	用于重复检测的顺序编号，每个新的邮箱服务将加 1（为了兼容老版本而只使用 1~7）

- 主站到从站通信–写邮箱命令

主站发送写数据区命令将邮箱数据发送从站。主站需要检查从站邮箱命令应答报文中工作计数器 WKC。如果工作计数器为 1，表示写命令成功。反之，如果工作计数器没有增加，通常因为从站没有读完上一个命令，或在限定的时间内没有响应，主站必须重发写邮箱数据命令。

- 从站到主站通信–读邮箱命令

从站有数据要发送给主站，必须先将数据写入输入邮箱缓存区，然后由主站来读取。主站发现从站 ESC 输入邮箱数据区有效数据等待发送时，会尽快发送适当的读命令来读取从站数据。主站有两种方式来测定从站是否已经将邮箱数据填入输入数据区。一种是使用 FMMU 周期性的读取某一个标志位。使用逻辑寻址可以读取多个从站的标志位，但其缺点是每个从站都需要一个 FMMU 单元。另一个方法是将简单的轮训 ESC 输入到邮箱的输入区。读命令的工作计数器增加 1，表示从站已经将新数据填入到了输入数据区。

2.3 EtherCAT 状态机

EtherCAT 状态机（ESM，EtherCAT State Machine）负责协调主站和从站应用程序在初始化和运行时的状态关系。

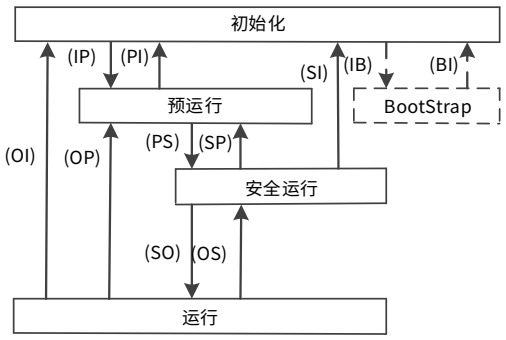
EtherCAT 设备必须支持四种状态，另外还有一个可选的状态。

- Init：初始化，简写为 I
- Pre-Operational：预运行，简写为 P

- Safe-Operational：安全运行，简写为 S
- Operational：运行，简写为 O
- Boot-Strap：引导状态（可选），简写为 B

以上各状态之间的转换关系如图 2-22 所示。从初始化状态向运行状态转化时，必须按照“初始化→预运行→安全运行→运行”的顺序转换，只有从运行状态返回时可以越级转化，其他状态均不可以越级转化。引导状态为可选状态，只允许与初始化状态之间相互转化。所有的状态改变都由主站发起，主站向从站发送状态控制命令请求新的状态，从站响应此命令，执行所请求的状态转换，并将结果写入从站状态指示变量。如果请求的状态转换失败，从站将给出错误标志。

图 2-22 EtherCAT 状态转化关系



- Init：初始化

初始化状态定义了主站与从站在应用层的初始通信关系。此时，主站与从站应用层不可以直接通信，主站使用初始化状态来初始化 ESC 的一些配置寄存器。如果主站支持邮箱通信，则配置邮箱通信参数。

- Pre-Operational：预运行

在预运行状态下，邮箱通信被激活。主站与从站可以使用邮箱通信来交换与应用程序相关的初始化操作和参数。在这个状态下不允许过程数据通信。

- Safe-Operational：安全运行

在安全运行状态下，从站应用程序读入输入数据，但是不产生输出信号。设备无输出，处于“安全状态”。此时，仍然可以使用邮箱通信。

- Operational：运行

在运行状态下，从站应用程序读入数据，主站应用程序发出输出数据，从站设备产生输出信号。此时，仍然可以使用邮箱通信。

- Boot-Strap：引导状态（可选）

引导状态的功能是下载设备固件程序。主站可以使用 FoE 协议的邮箱通信下载一个新的固件程序给从站。

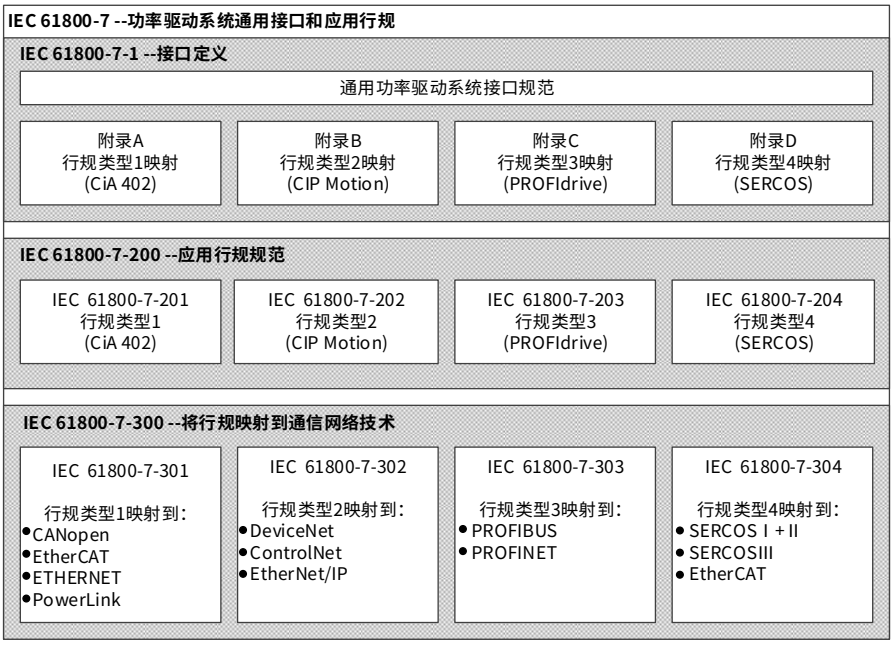
表 2-3 EtherCAT 状态机其转化过程总结过程

状态和状态转化	描述
初始化	应用层没有通信，主站只能读写 ESC 寄存器
初始化向预运行转化 Init to Pre-OP（IP）	主站配置从站站点地址寄存器 如果支持邮箱通信，则配置邮箱通道参数；如果支持分布式时钟，则配置 DC 相关寄存器 主站写状态控制寄存器，以请求“Pre-Op”状态
预运行	应用层邮箱数据通信
Pre-Op to Safe-Op（PS）	主站使用邮箱初始化过程数据映射 主站配置过程数据通信使用的 SM 通道 主站配置 FMMU 主站写状态控制寄存器，以请求“Safe-Op”状态
Safe-Operational	主站发送有效的输出数据 主站写状态控制寄存器，以请求“Op”状态
Operational	输入和输出全部有效，仍然可以使用邮箱通信

2.4 EtherCAT 伺服驱动器控制应用协议

IEC 61800 标准系列是一个可调速电子功率驱动系统通用规范。其中 IEC 61800-7 定义了控制系统和功率驱动系统之间的通信接口标准、包括网络通信技术和应用行规，如图 2-23 所示。EtherCAT 作为网络通信技术，支持了 CANopen 协议中的行规 CiA402 和 SERCOS 协议的应用层，分别称为 CoE 和 SoE。

图 2-23 IEC 61800-7 体系结构



2.4.1 基于 EtherCAT 的 CAN 应用协议（CoE）

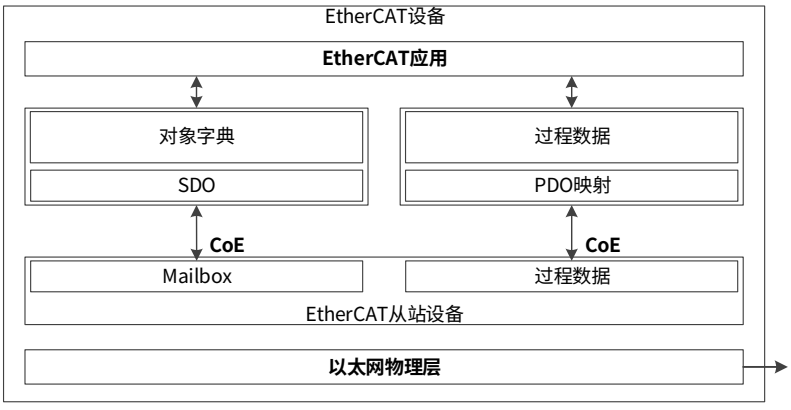
CANopen 设备和应用行规广泛用于多种设备类别和应用，如 I/O 组件、驱动、编码器、比例阀、液压控制器，以及用于塑料或纺织行业的应用行规等。EtherCAT 可以提供与 CANopen 机制相同的通讯机制，包括对象字典、PDO（过程数据对象）、SDO（服务数据对象），甚至相似的网络管理。因此，在已经实施了 CANopen 的设备中，仅需稍加变动即可轻松实现 EtherCAT，绝大部分的 CANopen 固件都得以重复利用。并且，可以选择性地扩展对象，以便利用 EtherCAT 所提供的巨大带宽资源。

EtherCAT 协议在应用层支持 CANopen 协议，并作了相应的补充，其主要功能有：

- 使用邮箱通信访问 CANopen 对象字典和对象，实现网络初始化。
- 使用 CANopen 应用对象和可选的时间驱动 PDO 消息，实现网络管理。
- 使用对象字典映射过程数据，周期性传输指令数据和状态数据。

图 2-24 为 CoE 设备结构图，其通讯方式主要包括周期性过程数据通信及非周期数据通信。下文会分别介绍两者在实际应用中的区别。

图 2-24 CoE 设备结构



2.4.1.1 CoE 对象字典

CoE 协议完全遵从 CANopen 协议，其对象字典的定义也相同，如表 2-4 所示。表 2-5 列举了 CoE 的通信数据对象，其中针对 EtherCAT 通信扩展了相关通信对象 0x1C00~0x1C4F，用于设置存储同步管理器的类型、通信参数和 PDO 数据分配。

表 2-4 CoE 对象字典定义

索引号范围	描述
0x0000~0x0FFF	数据类型描述
0x1000~0x1FFF	通信对象包括：设备类型、标识符、PDO 映射、与 CANopen 兼容 CANopen 专用数据对象，在 EtherCAT 中保留 EtherCAT 扩展数据对象
0x2000~0x5FFF	制造商定义对象
0x6000~0x9FFF	行规定义数据对象
0xA000~0xFFFF	保留

表 2-5 CoE 通信数据对象

索引	描述
0x1000	设备类型
0x1001	错误寄存器
0x1008	设备商设备名称
0x1009	制造商硬件版本
0x100A	制造商软件版本
0x1018	设备标识符
0x1600~0x17FF	RxPDO 映射
0x1A00~0x1BFF	TxPDO 映射
0x1C00	同步管理器通讯类型
0x0x1C10~0x1C2F	过程数据通信同步管理器 PDO 分配
0x0x1C30~0x1C4F	同步管理参数

2.4.1.2 CoE 周期性过程数据通信（PDO）

周期性数据通信中，过程数据可以包含多个 PDO 映射数据对象。CoE 协议使用的数据对象 0x1C10~0x1C2F 定义相应的 PDO 映射通道。表 2-6 为 EtherCAT 协议中对该通讯数据的具体结构。

表 2-6 CoE 通信数据对象

索引	对象类型	描述	类型
0x1C10	数组	SM0 PDO 分配	无符号整型 16 位
0x1C11	数组	SM1 PDO 分配	无符号整型 16 位
0x1C12	数组	SM2 PDO 分配	无符号整型 16 位
0x1C13	数组	SM3 PDO 分配	无符号整型 16 位
…	…	…	…
0x1C2F	数组	SM31 PDO 分配	无符号整型 16 位

以下针对 SM2 PDO（0x1C12）进行分配举例，表 2-7 列出了其取值实例。如 PDO0 中映射了两个数据，第一个通讯变量为控制字，对应映射的索引及子索引地址为 0x6040：00；第二个通讯变量目标位置值，对应映射的索引及子索引地址为 0x607A：00。

表 2-7 SM2 通道 PDO 分配对象数据 0x1C12 举例

0X1C12 子索引	数值	PDO 数据对象映射			
		子索引	数值	字节数	描述
0	3	-	-	1	PDO 映射对象数目
1	PDO0 0x1600	0	2	1	数据映射数据对象数目
		1	0x6040：00	2	控制字

0X1C12 子索引	数值	PDO 数据对象映射			
		子索引	数值	字节数	描述
		2	0x607A：00	4	目标位置
1	PDO1 0x1601	0	2	1	数据映射数据对象数目
		1	0x6071：00	2	目标转矩
		2	0x6087：00	4	目标斜坡
1	PDO2 0x1602	0	2	1	数据映射数据对象数目
		1	0x6073：00	2	最大电流
		2	0x6075：00	4	马达额定电流

PDO 映射有以下几种方式：

1、 简单设备不需要映射协议：使用简单的过程数据；在从站的 EEPROM 中读取。

2、 读取的 PDO 映射：固定过程数据映射；使用 SDO 通信读取。

3、 可选择的 PDO 映射：多组固定的 PDO 通过对象 0x1C1X 选择；通过 SDO 通信读取。

4、 可变的 PDO 映射：通过 CoE 通信配置。

2.4.1.3 CoE 非周期性过程数据通信（SDO）

EtherCAT 主站通过读写邮箱数据 SM 通道实现非周期性数据通信。CoE 协议邮箱数据结构如图 2-25 所示。

图 2-25 CoE 数据头

针对图 2-25 中的编号部分在表 2-8 中有详细的解释。

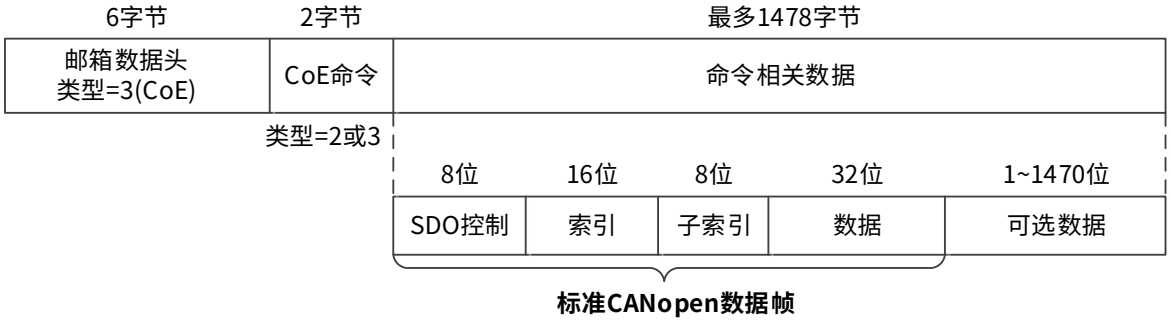
表 2-8 CoE 命令定义

CoE 命令字段编号	描述
编号	PDO 发送时的编号
类型	消息类型： 0：保留　　1：紧急事件信息 2：SDO 请求　　3：SDO 响应 4：TxPDO　　5：RxPDO 6：远程 TxPDO 发送请求　　7：远程 RxPDO 发送请求 8：SDO 信息　　9~15：保留

- SDO 服务

CoE 通信服务类型 2 和 3 为 SDO 通信服务，SDO 数据结构如图 2-26 所示。

图 2-26 SDO 数据帧格式



SDO 按传输方式通常有如下三种类型，表 2-9 为 SDO 数据帧具体的内容，其结构如图 2-27 所示。

1、 快速传输服务：与标准的 CANopen 协议相同，只使用 8 个字节，最多传输 4 个字节有效数据。

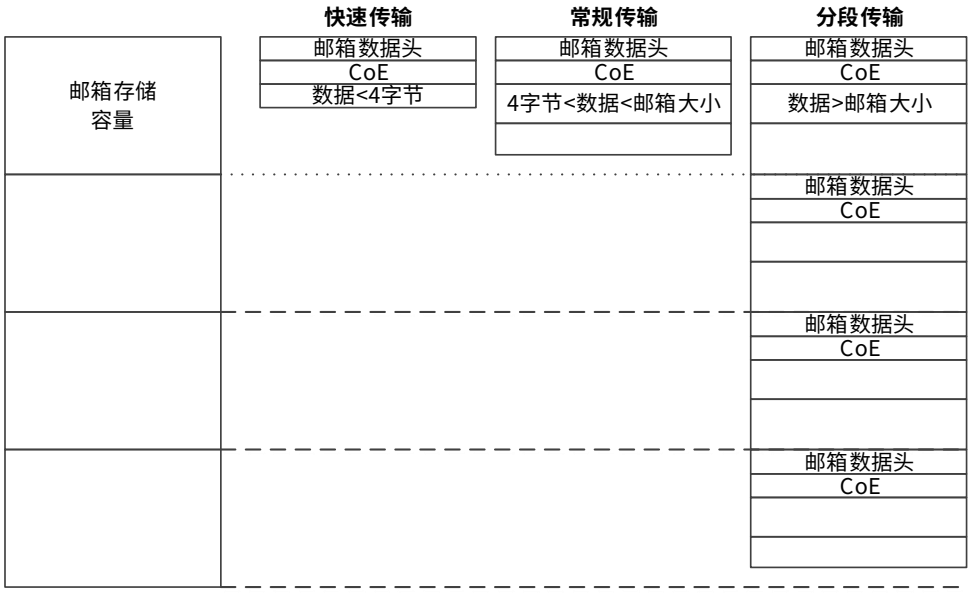
2、 常规传输服务：使用超过 8 个字节，可以传输超过 4 个字节的有效数据，最大可传输的有效数据取决于邮箱 SM 所管理的存储区容量。

3、 分段传输服务：对于超过邮箱容量的情况，使用分段的方式进行传输。

表 2-9 CoE 数据帧内容

SDO 控制	标准 CANopen SDO 服务
索引	设备对象索引
子索引	子索引
数据	SDO 中的数据
数据（可选）	有 4 个字节的可选数据可被加截至数据帧中

图 2-27 SDO 传输类型



如果要传输的数据大于 4 个字节，则使用常规传输服务。在常规传输时用快速传输时的 4 个数据字节表示要传输的数据的完整大小，用扩展数据部分传输有效数据，有效数据的最大容量为邮箱容量减去 16。

2.4.2 IEC 61800-7-204 的伺服驱动行规（SERCOS）

SERCOS（Serial Real-time Communication System）被公认为用于高性能实时系统的通讯接口，特别适用于运动控制应用场合。其伺服驱动和通讯技术的行规属于 IEC 61800-7-204 标准的范畴。以下是 SERCOS 和 EtherCAT 的整合以及兼容性方面的关键点：

SERCOS 和 EtherCAT 的映射（SoE）：SERCOS 的伺服驱动行规到 EtherCAT 的映射在 IEC 61800-7-204 标准的 304 部分中进行了定义。SoE（SERCOS over EtherCAT）为 SERCOS 的伺服驱动参数和功能提供了一种基于 EtherCAT 邮箱的访问方法。

参数访问和服务通道：用于访问驱动中全部参数及功能的服务通道是基于 EtherCAT 邮箱进行的。这种方式确保了 EtherCAT 与现有 SERCOS 协议的兼容性，能够访问 IDN（SERCOS 的标识符）的数值、属性、名称、单位等信息。

数据传输机制：SERCOS 的过程数据（AT 和 MDT 格式数据）通过 EtherCAT 设备协议机制进行传送，其映射方式与 SERCOS 的映射方式相似。这样，SERCOS 的数据可以在 EtherCAT 网络中高效传输。

状态机映射：EtherCAT 从站的状态机可以很容易地映射为 SERCOS 协议的状态。这种状态机映射使得 SERCOS 和 EtherCAT 的整合更加顺畅，确保了两者之间的兼容性和协同工作能力。

扩展性和数据长度限制：在保证兼容性的同时，EtherCAT 还关注与数据长度限制相关的扩展性。这种扩展性确保了在处理复杂应用时，EtherCAT 可以灵活应对不同的数据需求。

2.4.2.1 SoE 状态机

SERCOS 协议的通信阶段与 EtherCAT 状态机的比较图如图 2-28 所示，其特点有以下几个方面：

1、 SERCOS 协议通信阶段 0 和 1 被 EtherCAT 初始化状态覆盖。

2、 通信阶段 2 对应于与运行状态，允许使用邮箱通信实现服务通道，操作 IDN 参数。

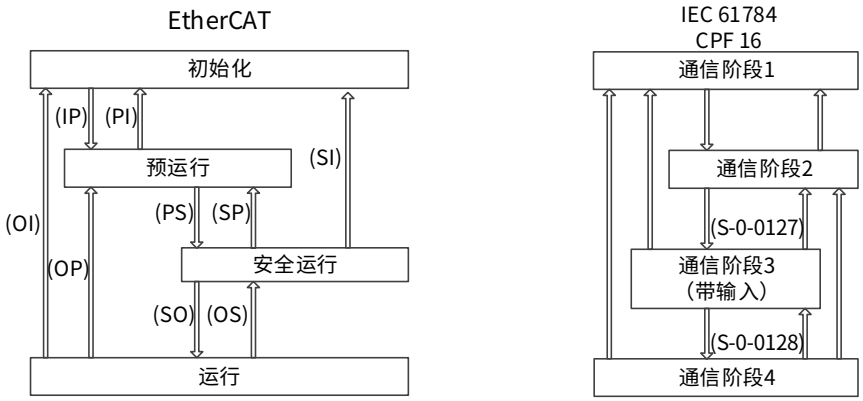
3、 通信阶段 3 对应于安全运行状态，开始传输周期性数据，只有输入数据有效，输出数据被忽略，同时可以实现时钟同步。

4、 通信阶段 4 对应于运行阶段，所有的输入和输出都有效。

5、 不使用 SERCOS 协议的阶段切换过程命令 S-0-0127（通信阶段 3 切换检查）和 S-0-0128（通信阶段 4 切换检查），分别由 PS 和 SO 状态转化取代。

6、 SERCOS 协议只允许高级通信阶段向下切换到通信阶段 0，而 EtherCAT 允许任意的状态向下切换，如图 2-28 的 a）所示。例如从运行状态切换到安全运行状态，或从安全运行状态切换到预运行状态。SoE 也应该支持这种切换如图 2-28 的 b）中所示，如果从站不支持，则应该 EtherCAT AL 状态寄存器中设置错误位。

图 2-28 SoE 状态机



a)EtherCAT状态机 b)SERCOS状态机

2.4.2.2 IDN 继承

SoE 协议继承 SERCOS 协议的 DIN 参数定义。每个 IDN 参数都有一个唯一的 16 位标识号 IDN，对应一个唯一的数据块，保存参数的全部信息。数据块由 7 个元素组成，如表 2-10 所列。IDN 参数分为标准数据和产品数据两部分，每部分又分为 8 个参数组，使用不同的 IDN 表示，如表 2-11 所列。

表 2-10 IDN 数据块结构

编号	名称
元素 1	IDN
元素 2	名称
元素 3	属性
元素 4	单位
元素 5	最小允许值
元素 6	最大允许值
元素 7	数据值

表 2-11 IDN 编号定义

位	15	14~12	11~0
含义	分类	参数组	参数编号
取值	0：标准数据 S；1：产品数据 P	0~7：8 个参数组	0000~4095

在使用 EtherCAT 作为通信网络时，取消了一些 SERCOS 协议中用于通信接口控制的 IDN，如表 2-12 所列。此外，还对一些 IDN 的定义做了些修改，如表 2-13 所列。

表 2-12 删除的 IDN

IDN 描述	IDN 描述
S-0-0003	最小 AT 发送的开始时间
S-0-0004	发送到接收状态切换时间
S-0-0005	最小反馈采样提前时间
S-0-0009	主站数据报文中的开始地址
S-0-0010	主站数据报文长度
S-0-0088	接收 MDT 后准备好接收 MST 所需要的恢复时间
S-0-0090	命令处理时间
S-0-0127	通信阶段 3 切换检查
S-0-0128	通信阶段 4 切换检查

表 2-13 修改的 IDN

IDN	原描述	新描述
S-0-0006	AT 发送的开始时间	在从站内部于同步信号之后应用程序向 ESC 存储区写入 AT 数据的时间偏移
S-0-0014	通信接口状态	映射从站 DL 状态和 AL 状态码
S-0-0028	MST 错误技术	映射从站 RX 错误计数器到丢失计数器
S-0-0089	MDT 发送开始时间	在从站内部于同步信号之后从 ESC 存储区得到新的 MDT 数据的时间偏移

2.4.2.3 SoE 周期性过程数据

输出过程数据（MDT 数据内容）和输入过程数据（AT 数据内容）由 S-0-0015、S-0-0016 和 S-0-0024 配置。过程数据不包括服务通道数据，只有周期性过程数据。输出过程数据包括伺服控制字和指令数据，输入过程包括状态字和反馈数据。S-0-0015 设定了周期性过程数据的类型，如表 2-14 所列，参数 S-0-0016 和参数 S-0-0024 如表 2-15 所列。主站在“预运行”阶段通过邮箱通信写这三个参数，以配置周期性过程数据的内容。

表 2-14 参数 S-0-0015 定义

S-0-0015	指令数据	反馈数据
0：标准类型 0	无	无反馈数据
1：标准类型 1	扭矩指令 S-0-0080（2 字节）	无反馈数据
2：标准类型 2	速度指令 S-0-0036（4 字节）	速度反馈 S-0-0053（4 字节）
3：标准类型 3	速度指令 S-0-0036（4 字节）	位置反馈 S-0-0051（4 字节） 速度反馈 S-0-0053（4 字节）
4：标准类型 4	位置指令 S-0-0047（4 字节）	
5：标准类型 5	位置指令 S-0-0047（4 字节） 速度指令 S-0-0036（4 字节）	位置反馈 S-0-0051（4 字节） 或速度反馈 S-0-0053（4 字节）+ 位置反馈 S-0-0051（4 字节）
6：标准类型 6	速度指令 S-0-0036（4 字节）	无反馈数据
7：自定义	S-0-0024 配置	S-0-0016 配置

表 2-15 参数 S-0-0016 和参数 S-0-0024 定义

数据字	S-0-0024 定义	S-0-0016 定义
0	输出数据最大长度（Word）	输入数据最大长度（Word）
1	输出数据实际长度（Word）	输入数据实际长度（Word）
2	指令数据映射的第一个 IDN	反馈数据映射的第一个 IDN
3	指令数据映射的第二个 IDN	反馈数据映射的第二个 IDN
…	…	…

2.4.2.4 SoE 非周期性服务通道

EtherCAT SoE 服务通道 SSC（SoE Service Channel）由 EtherCAT 邮箱通信功能完成，它用于非周期性数据交

换，如读写 IDN 及其元素。SoE 数据头格式如图 2-29 所示。

图 2-29 SoE 数据头格式

表 2-16 SoE 数据命令描述

数据区	描述
命令	即指令类型： 0x01：读请求 0x02：读响应 0x03：写请求 0x04：写响应 0x05：通报 0x06：从站信息 0x07：保留
后续数据	后续数据信号： 0x00：无后续数据帧 0x01：未完成传输，有后续数据帧
错误	错误信号： 0x00：无错误 0x01：发生错误，数据区有 2 个字节的错误码
地址	从站设备的具体地址
操作元素标识	单个元素操作时为元素选择，按位定义，每一个位对应一个元素；寻址结构体时为元素的数目
IDN	参数的 IDN 编号，或分段操作时的剩余片段

常用的 SSC 操作包括 SSC 读操作、SSC 写操作和 SSC 过程命令。

SSC 读操作：SSC 读操作由主站发起，写 SSC 请求到从站。从站接收到读操作请求后，用所请求的 IDN 编号和数据值作为回答。主站可以同时读多个元素，从站应该同时回答多个元素，如果从站只支持单个元素操作，应该以所请求的第一个元素作为响应。

SSC 写操作：SSC 写操作用于主站下载数据到从站，从站应该以写操作的结果回答。分段操作由一个或多个分段写操作及一个 SSC 写响应服务组成。

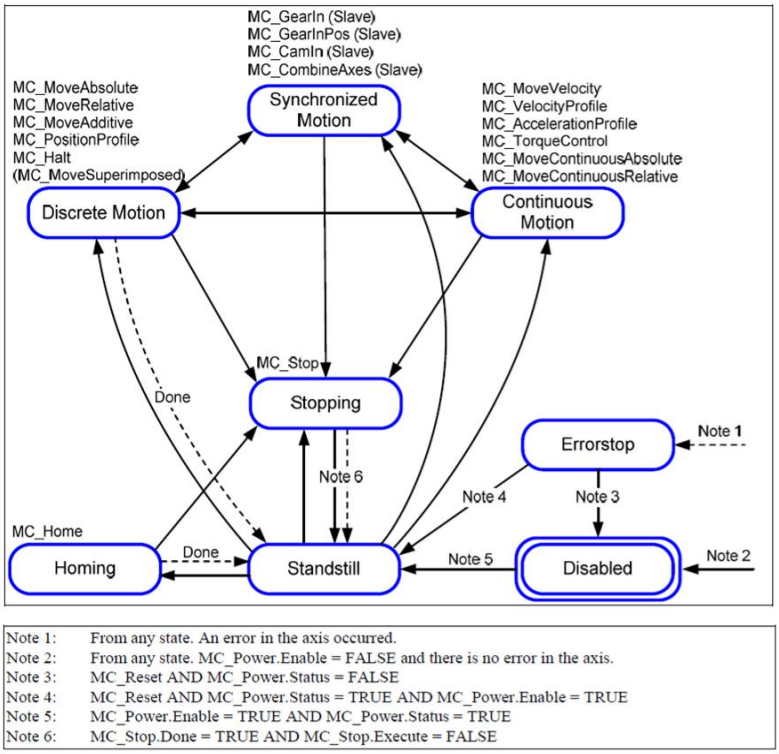
SSC 过程命令：SSC 过程命令是一种特殊的非周期数据，每一个过程命令都有唯一标识的 IDN 和规定的数据元素，用于启动伺服装置的某些特定功能或过程。执行这些功能或过程通常需要一段时间，过程命令只是触发其开始，随后它所占用的服务通道立即变为可用，用以传输其他非周期数据或过程命令，而不用等到被触发的功能或过程执行完毕。

3 轴状态机制

3.1 轴状态转换

轴状态切换以 PLCopen 状态机图为参考设计，具体转换如下图 3-1 所示。

图 3-1 轴状态转换



Note 1:	From any state. An error in the axis occurred.
Note 2:	From any state. MC_Power.Enable = FALSE and there is no error in the axis.
Note 3:	MC_Reset AND MC_Power.Status = FALSE
Note 4:	MC_Reset AND MC_Power.Status = TRUE AND MC_Power.Enable = TRUE
Note 5:	MC_Power.Enable = TRUE AND MC_Power.Status = TRUE
Note 6:	MC_Stop.Done = TRUE AND MC_Stop.Execute = FALSE

- 在轴停止状态可以转移到各种运行状态。
- 可以由多种状态转移到停止状态。
- 离散、同步、连续可以直接相互切换状态。
- 若伺服轴出现报警（错误停止）就必须先运行 MC_Reset 指令、MC_Power 指令使轴进入 Standstill 状态，才能让轴再次运行。
- 若不按上述转移图方式使用 MC 指令命令轴运动，就不会使轴响应，会得到 MC 功能块的错误报警信息。

4 编程基础

4.1 变量

变量是保存在存储器中待处理的抽象数据，是为了识别 PLC 的输入/输出、PLC 内部的存储区域而使用的名称，可以代替物理地址在程序中的使用。可以根据需要随时改变变量中所存储的数据值。在程序执行过程中，变量的值可以发生变化。

使用变量之前必须先声明变量，及指定变量的类型和名称。变量具有名称，类型和值。变量的数据类型确定它所代表的内存大小和类型。变量名即指在程序源代码中的标识符。

4.1.1 变量声明

变量声明就是指定变量的名称、类型和赋初始值，变量的声明非常重要，未经声明的变量是不能通过编译的，所以也无法在程序中使用。用户可以在程序组织单元（POU）、全局变量列表（GVL）和自动声明对话框中进行变量的声明。在 CoDeSys 中变量声明分为两类，普通变量声明和直接变量。

- 普通变量声明

最常用的变量声明，不需要和硬件外设或通讯进行关联的变量，仅供项目内部逻辑使用。普通声明须符合以下规则。

<标识符>:<数据类型>{:=<初值>};

{}中的部分是可选部分，如：nTest:BOOL;，nTest:BOOL:=TRUE;

- 直接变量声明

在 CoDeSys 应用中，当需要和可编程逻辑控制器的 I/O 模块进行变量映射或和外部设备进行网络通讯时，需要采用此声明方法。使用关键字 AT 把变量直接联结到确定地址，直接变量须符合以下规则：

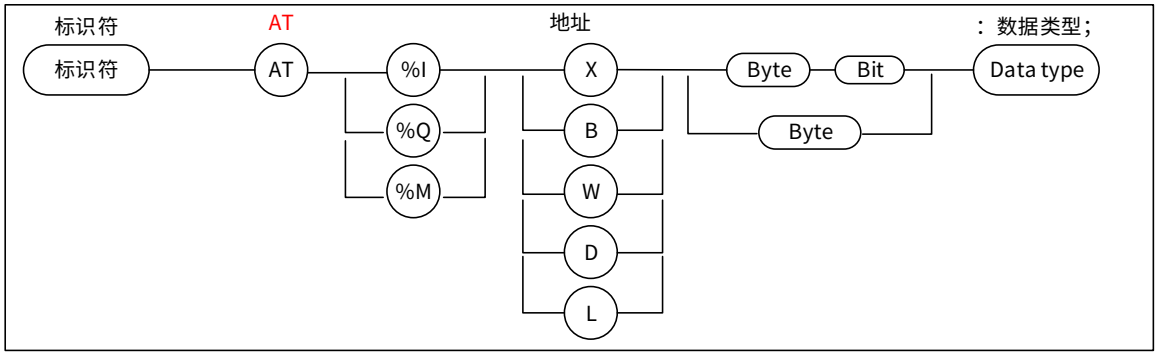
AT<地址>：

<标识符>AT<地址>:<数据类型>{:=<初始化值>};

{}中的部分是可选部分。

使用“%”开始，然后是位置前缀符号和大小前缀符号，如果有分级，则用整数表示分级，并用小数点符号“.”表示，如%IX0.0，%QW0。直接变量声明的具体格式如图 4-1 所示。

图 4-1 直接变量的表示方法



I：表示输入单元；Q：表示输出单元；M：表示存储区单元。大小前缀的定义如表 4-1 所示。

表 4-1 大小前缀定义

前缀符号	定义	约定数据类型
X	位（bit）	BOOL
B	字节（BYTE）	BYTE
W	字（WORD）	WORD
D	双字（DWORD）	DWORD

前缀符号	定义	约定数据类型
L	长字（LWORD）	LWORD
*	未特定位置的内部变量，系统自动分配。	

【例4.1】 在程序中定义了变量双字型 Var1，如需拿取该变量其中的一部分数据，将其转换成布尔、字节、字类型的变量，其首地址为多少，该如何换算。

```
VAR
Var1                    AT%ID48                 :DWORD;
END_VAR
```

%I 说明了该变量属于输入单元，具体的地址为%ID48。在表 4-2 列出了 CoDeSysV3.x 在寻址时，系统会根据数据类型的大小（X:bit，B:byte，W:word，D:dword）来进行分配。

该地址内存映射表中，字地址%IW96 和%IW97 两个字组合后对应%ID48，因为 48*2（字节）后的字节首地址为 96。同样的道理，字节地址%IB192、%IB193、%IB194 和%IB195 这四个字节变量组合后对应%ID48，因为 48*4（字节）后对应的字节首地址正好为 192。

表 4-2 内存映射表

%IX	96.0~96.7	96.8~192.15	97.0~97.7	97.8~97.15
%IB	192	193	194	195
%IW	96		97	
%ID	48			

【例4.2】 相信有了【例 4.1】的基础，就很容易理解如下的地址映射关系。

%MX12.0：是%MB12 的第一位。

%IW4：表示输入字单元 4（字节单元 8 和 9）。

%Q*：输出在一个为特定的位置。

%IX1.3：表述输入第 1 字节单元的第三位。

4.1.2 数据类型

无论声明的是变量还是常量，都必须使用到数据类型。数据类型的标准化是编程语言开放性的重要标志，在 CoDeSys 中数据类型完全符合 IEC61131-3 所定义的标准。CoDeSys 将数据类型分为标准数据类型、IEC1131-3 标准的扩展数据类型和自定义数据类型，数据类型决定了它将占用多大的存储空间以及将存储何种类型的值。

CoDeSys 标准数据类型共分为 5 大类，分别为布尔类型、整数类型、实数类型、字符串类型和时间数据类型，表 4-3 将 CoDeSys 所支持的标准数据类型列举出来。

表 4-3 标准数据类型

数据大类	数据类型	关键字	位数	取值范围
布尔	布尔	BOOL	1	FALSE（0）或 TRUE（1）
整型	字节	BYTE	8	0~255
	字	WORD	16	0~65535
	双字	DWORD	32	0~4294967295
	长字	LWORD	64	0~（2^{64}-1）
	短整型	SINT	8	-128~127
	无符号短整型	USINT	8	0~255
	整型	INT	16	-32768~32767
	无符号整型	UINT	16	0~65535
	双整型	DINT	32	-2147483648~2147483647
	无符号双整型	UDINT	32	0~4294967295
	长整型	LINT	64	-2^{63}~（2^{63}-1）
实数	实数	REAL	32	1.175494351e-38~3.402823466e+38

数据大类	数据类型	关键字	位数	取值范围
	长实数	LREAL	64	2.2250738585072014e-308~ 1.7976931348623158e+308
字符串	字符串	STRING	8*N	-
时间数据	时间	TIME	32	T#0ms~T#71582m47s295ms
		TINE_OF_DAY		T0D#0:0:0~T0D#1993:02:47.295
		DATE		D#1970-1-1~D#2106-02-06
		DATE_AND_TIME		DT#1970-1-1-0:0:0~DT#2106-02-06-06:28:15

4.1.2.1 布尔

布尔型变量用来表示 TRUE/FALSE 值，一个布尔型变量只有 TRUE 或 FALSE 两种状态，在 CoDeSys 还可以使用 0 或 1 来表示。

类型	内存使用
BOOL	8 位

【例4.3】 将开门信号和取料信号的与逻辑结果赋值给布尔型变量 bReady，结构化文本语言代码如下。

```
VAR
bReady,bDoors_Open,bGrip:BOOL;
END_VAR
bReady:=(bDoors_Open and bGrip);
```

在 CoDeSys 中允许将相同类型的变量进行统一声明，用“,”进行分割。

【例4.4】 将十进制数 211 赋值给 bReady 变量，结构化文本语言代码如下。

```
VAR
bReady:BOOL;
END_VAR
bReady:=211;
```

将一个整型数据赋值给布尔型数据，显然这种赋值方式是错误的，通过程序编译后，编译会返回错误提示“C0032: 不能将类型‘USINT’转化为类型‘BOOL’”。

布尔型变量是使用最多的一种变量类型。所以在流程控制语句中（如 IF，CASE，循环）中时常会用到，所以学会正确使用布尔变量至关重要。

注意：如果在内存中的最低位被置位（例如 2#00000001），则 BOOL 类型变量为“TRUE”（真）。如果内存中最低位没有被置位，则 BOOL 变量为 FALSE 例如 2#00000000。所有其它值都不能被正确地进行转换，并显示为(***INVALID:16#xy***，联机监视时)。类似的问题是可能出现的，例如，在 PLC 程序中使用了重叠的内存范围。

例如，如果定义了一个布尔数组，A:ARRAY[0..7]OFBOOL，在系统中，它所占用的总内存并不是一个 8 位字节，而是占用了 8 个 8 位字节。

4.1.2.2 整型

整型类型代表了没有小数点的整数类型。在 CoDeSys 中，整型是一个最大的标准类，其成员最多，没有必要将每个类型的关键字死记硬背，只要了解其中的规律，就非常容易记忆，下面简单的说明了整型前缀其中的规律。

- U_表示无符号数据类型，U 为 Unsigned 的缩写。
- S_表示短数据类型，S 为 Short 的缩写。
- D_表示双数据类型，D 为 Double 的缩写。
- L_表示长数据类型，L 为 Long 的缩写。

如 UINT 为无符号的整型数据，USINT 即为无符号的短整型数，LINT 表示长整型数据。

【例4.5】 整型举例。

```
VAR
nValue1:USINT;
nValue2:LINT;
nValue3:WORD;
END_VAR

nValue1:=4;
nValue3:=16;
nValue2:=nValue1+nValue3;
```

运行程序后，nValue2 最终的输出结果为 20。

无符号（Unsigned）和有符号（Signed）的区别是最高位的区别。

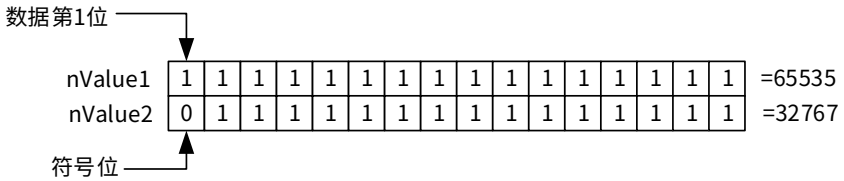
无符号类型数据将全部存储空间全部储存数据本身，没有符号位。如 UINT 型将 16 为全部存储数据本身，即数据范围可以从 0~65535。

有符号类型数据牺牲最高位作为符号位。如 INT 型变量牺牲最高位作为符号位，剩下 15 位作为数据存储，故该数据范围为-32768~32767。所以有符号整型变量中可以存放的正数的范围比无符号的整形变量中正数的范围缩小一倍。

如下通过两个变量对无符号和有符号进行说明：

```
nValue1:UINT;
nValue2:INT;
```

图 4-2 无符号与有符号数据结构



4.1.2.3 实数

实数，也可称为浮点数，这里主要是用于处理含有小数的数值数据，实数类型包含了 REAL 及 LREAL 这 2 种数据类型。REAL 实数占用 32 位存储空间，而 LREAL 长实数占用 64 为存储空间。在 CoDeSys 中，实数和长实数常量有两种表示形式。

1、 十进制小数形式

它由数字和小数点组成。0.123、123.1、0.0 都是十进制小数表现形式。

2、 指数形式

如 123e3 或 123E3 都代表 123×10^3。但注意字母 e（或 E）之前必须有数字，且 e 后面的指数必须为整数，如 e3、2.1e3.5、.e3、e 等都是不合语法的指数形式。

一个浮点数可以有多种指数表示形式，如 123.456 可以表示为 123.456e0、12.3456e1、1.23456e2 等。其中的 1.23456e2 称为“规范化的指数形式”。即在字母 e（或 E）之前的小数部分中，小数点左边应有一位（且只能有一位）非零的数字。

【例4.6】 将 12.3 赋值给 rRealVar1 变量，结构化文本语言代码如下：

```
VAR
rRealVar1:REAL;
END_VAR
RealVar1:=1.23e1;
```

在例 4.6 中，1.23e1 表示就是 12.3，当然也可以直接通过表达式 RealVar1:=12.3 完成上例要求。

此时如果将要求更换为将 0.123 赋值给 rRealVar1 变量，通过上面提到的规律，只需将表达式更换为：

RealVar1:=1.23e-1;

或，

RealVar1:=0.123;

注意：数据类型 LREAL 的支持取决于目标设备。

在编译时 64 位类型的 LREAL 是被转换为 REAL（可能有信息丢失），还是保持不变，需参考不同硬件产品的相应文档。如果 REAL 或 LREAL 类型转换成 SINT,USINT,INT,UINT,DINT,UDINT,LINT 或 ULINT 类型，且实型数据的值超出了整形的范围，结果将会是不确定的并且该值取决于目标系统。

这种情况有可能产生异常为了获取与目标无关的代码，应由应用程序处理所有值域越界问题。如果 REAL,LREAL 型数据在整形的值域范围内，它们之间的转换在所有系统上都可以进行。

4.1.2.4 字符串

一个字符串是一个字符队列。字符串（STRING）常数使用一个单引号作为其前缀和后缀。也可以输入空格和专用字符（如音符）。这些字符将同所有其它字符一样进行处理。在 CoDeSys 中，字符串类型变量可以包含任意一串字符，使用单撇号括起来的一个字符串，如‘Hello’、‘Howareyou’、‘CoDeSys’、‘why?’等都是字符常串量，声明时的大小决定了存储变量所需要的存储空间。这里的存储空间它是指字符串中字符的数量，用圆括号或方括号括起来，具体计算及声明方法如下。

- 如果在定义变量是没有指定字符串大小，系统则默认分配 80 个字符给该变量，系统中实际占用存储空间=[80+1] byte。

例如在变量声明中定义的是 Str1:STRING:=‘a’;虽然实际的 Str1 变量初始值只包含一个字符，但由于再声明中未使用括号来限定字符串大小，故 Str1 在系统内占用的内存空间为 80+1 个字节。

- 如果定义了大小，则系统中实际占用存储空间=[空间的大小为（定义字符串大小）+1] byte。CoDeSys 中一般不限制字符串的长度，但是字符串函数只能处理长度在 1~255 个字符之间的字符串。例如，定义两个字符串，其语句如下：

  Str1:STRING[10]:='a';

  Str2:STRING:='a';

上述两句声明语句中内容是一样的，区别在于多了一个[10]的存储空间限定，图 4-3 中为这两种声明语句的方式在程序的内存部分的区别，左边由于 Str1 限定了大小为 10，故在程序中实际占的字节大小为 10+1，即 11 个。而 Str2 默认分配的是 80 个字符，实际占的大小为 80+1，即 81 个。

图 4-3 字符串的存储方式

地址	Str1
1	a
2	
3	
4	
5	
…	
10	
11	

地址	Str2
1	a
2	
3	
4	
5	
…	
80	
81	

通常而言，使用默认的 80 个字符能满足大部分的应用，但是如果应用中有大量的字符串数据，但每个字符串中实际的字符数据却又很小，这样就会造成数据存储区的大量浪费，限定大小后能够节省很多存储空间供其他变量使用。如果一个变量用字符串初始化，而字符串对于变量的数据类型来说又太长，字符串将会从右至左相应的截断。

字符串在程序中的表示时，为了和普通的变量区分开，需要加单引号‘XXX’。

【例4.7】 将字符串‘HelloCoDeSys’赋值给 str 变量。

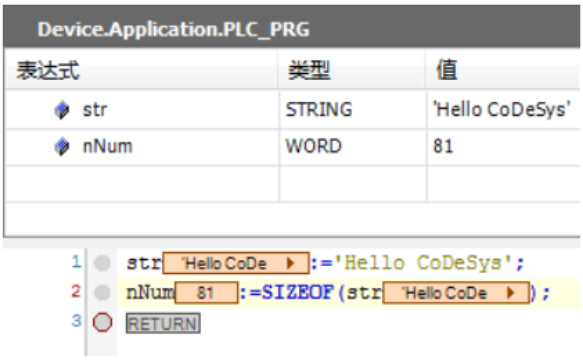
VAR

str:STRING;

```
nNum:WORD;
END_VAR
str:='HelloCoDeSys';
nNum:=SIZEOF(str);（*使用 SIZEOF 指令查看存储空间占用量*）
```

程序运行的结果如图 4-4 所示，'HelloCoDeSys'实际字符数为 13 个字符，占存储空间的大小为 14 个字节，但是使用 SIZEOF 指令看到的输出结果为 81 个 byte，其原因就是因为没有指定字符串大小，系统自动分配了 80 个字符给 str 变量。

图 4-4 字符串的实例运行结果

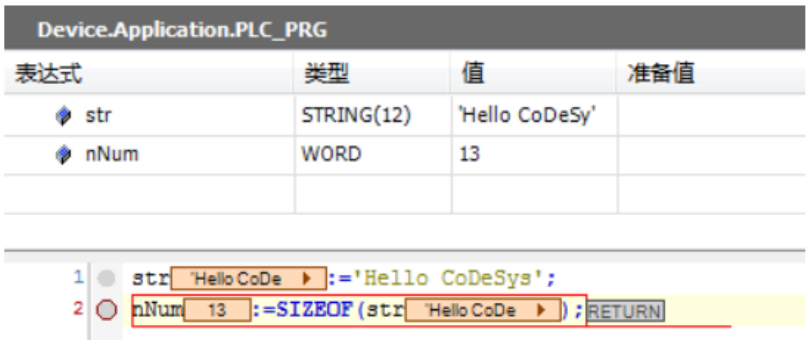


【例4.8】 将字符串‘HelloCoDeSys’赋值给 str 变量，str 定义为 12 个字符大小。

```
VAR
str:STRING[12];
nNum:WORD;
END_VAR
str:='HelloCoDeSys';
nNum:=SIZEOF(str);
```

程序实际运行的结果如图 4-5 所示。

图 4-5 字符串实例运行结果



从运行的结果来看，str 只显示了‘HelloCoDeSy’，缺少了一个‘s’，即多余部分已被系统自动截断。字符串占用的存储空间为 13 个 byte。

4.1.2.5 时间数据

时间数据类型包括 TIME、TIME_OF_DAY/TOD、DATE 和 DATE_AND_TIME/DT。系统内部处理这些数据的方式与双字（DWORD）类型相似。

1、 **TIME：**时间，精度为毫秒（ms），范围从 0~71582m47s295ms。语法格式如下。

t#<时间声明>

一个 TIME 常数总是由一个起始符 T 或 t（或用 TIME 或 time）和一个数字标识符#组成。然后，是跟随的实际时间声明，包括天数（d 标识）、小时（h 标识）、分钟（m 标识）、秒（s 标识）和毫秒（ms 标识）。请注意时间各项必须根据时间长度单元顺序进行设置（d 在 h 之前，h 在 m 之前，m 在 s 之前，s 在 ms 之前），但无须包含所

有的时间长度单位。

在 ST 语言的赋值语句中，正确使用时间常数的示例：

```
TIME1:=T#14ms;
TIME1:=T#100S12ms;
(*最高单位的值可以超出其限制*)
TIME1:=t#12h34m15s;
```

【例4.9】 时间类型变量的定义和使用。

```
VAR
tTime:TIME;
END_VAR
tTime:=T#3d19h27m41s1ms;
```

注意：时间可以溢出，如小时可以超过 24h，如赋值时写入 T#3d29h27m41s1ms，系统会自动校正最终输出的结果为 T#4d5h27m41s1ms。

以下的时间常量赋值是不正确的。

```
tTime:=15ms;（*缺少 T#*）
tTime:=t#4ms13d;（*顺序错误*）
```

2、 **TIME_OF_DAY/TOD：**时刻，精度为毫秒（ms），范围从 0:0:0~1193:02:47.295。时刻声明使用“<时:分:秒>”的格式。语法格式如下。

tod#<时间声明>

除了“tod#”，也可以使用“TOD#”、“time_of_day”、“TIME_OF_DAY”表示。

【例4.10】 时刻类型变量的定义和使用。

```
VAR
tTime_OF_DAY:TIME_OF_DAY;
END_VAR
tTime_OF_DAY:=TOD#21:32:23.123;
```

上式表示的时间为 21 点 32 分 23 秒 123 毫秒。

3、 **DATE：**日期，精度为天，范围从 1970-01-01~2106-02-06。日期声明使用“<年-月-日>”的格式。语法格式如下。

dt#<日期声明>

除了“d#”也可以使用也可以用“D#”、“date”、“DATE”表示。这些常数可用来输入日期。声明一个 DATE 常数时，起始符为 d、D、DATE 或 date 后跟随一个#号。然后你就可按照年-月-日的格式输入任何日期。

【例4.11】 日期类型变量的定义和使用。

```
VAR
tDate:DATE;
END_VAR
tDate:=D#2014-03-09;
```

上式表示的时间为 2014 年 3 月 9 日。

4、 **DATE_AND_TIME/DT：**日期和时间，精度为秒（s），范围从 1970-01-01-00:00~2106-02-06-06:28:15。日期和时间的声明使用“<年-月-日-时:分:秒>”的格式，语法格式如下。

dt#<日期和时间声明>

除了“dt#”，也可以使用“DT#”、“date_and_time”、“DATE_AND_TIME”表示。

【例4.12】 日期和时间类型变量的定义和使用。

```
VAR
tDT:DATE_AND_TIME;
END_VAR
tDT:=DT#2014-03-09-16:22:31.223;
```

上式表示的时间为 2014 年 3 月 9 日 16 点 22 分 31 秒 223 毫秒。

4.1.3 变量类型

表 4-4 变量的类型

变量类型关键字	变量属性	外部读写	内部读写
VAR	局部变量	-	R/W
VAR_INPUT	输入变量，由外部提供	R/W	R
VAR_OUTPUT	输出变量，有内部变量提供给外部	W	R/W
VAR_IN_OUT	输入-输出变量	R/W	R/W
VAR_GLOBAL	全局变量，能在所有配置、资源内使用	R/W	R/W
VAR_TEMP	临时变量，程序和功能块内部存储使用的变量	-	R
VAR_STAT	静态变量	-	-
VAR_EXTERNAL	外部变量，能在程序内部修改，但需由全局变量提供	R/W	R/W

VAR、VAR_INPUT、VAR_OUTPUT 和 VAR_IN_OUT 是在程序组织单元（POU）中被应用的最多的几种变量类型。VAR_GLOBAL 全局变量在实际的工程项目中也需要被大量的使用。

4.1.4 保持变量

表 4-5 变量的属性一览

变量附加属性关键字	变量附加属性
RETAIN	保持型变量，用于掉电保持
PERSISTENT	保持型变量
VAR RETAIN PERSISTENT VAR PERSISTENT RETAIN	两者功能一样，皆为保持性变量，用于掉电保持
CONSTANT	常量

PERSISTENT

目前只有少数 PLC 还保留独立的内存区域用于存放 PERSISTENT 类型数据，在 CoDeSysV3.x 中，取消了其原掉电保持的功能，取而代之的是通过 VARRETAINPERSISTENT 或 VARPERSISTENTRETAIN 来实现，两者从功能上完全一样。

PERSISTENT 类型变量声明格式如下：

```
VARGLOBALPERSISTENTRETAIN
<标识符>:<数据类型>;
END_VAR
```

内存存储位置：与 RETAIN 变量一样，RETAINPERSISTENT 和 PERSISTENTRETAIN 变量也存储在一个独立的内存区中。

持续变量的复位：

保留变量用关键字“RETAIN”来识别，这些变量保持它们的值即使是在控制器的非正常关闭时和正常的关闭和其中的一个控制器或在命令“热复位”时。当程序重新运行时，存储的值将进行进一步的处理。一个具体的例子是生产线上的饼形计数器在电源故障后重新开始计数。所有其它的变量从新初始化，不是用它们的初始化值或标准初始化的值。与永久变量相反，保留变量在程序的一个新的下载时重新初始化。

永久变量通过关键字“**PERSISTENTRETAIN**”来识别。不像保留变量，这些变量在一个重新下载或在执行命令“冷复位”或“原始复位”之后还会继续保留它们的值。如表 4-6 所示，在何在线命令时会复位持续变量。

表 4-6 持续变量在线命令行为一览表

在线命令	VAR	VAR RETAIN	VAR PERSISTENT RETAIN
热复位	-	X	X
冷复位	-	-	X
原始复位	-	-	-
下载	-	-	X
在线改变	X	X	X
重新下载	-	X	X
注： “X”=保留值，“-”=初始值。			

5 编程语言

5.1 概述

不同的工程应用具有不同的最佳编程方式，每种编程语言都具有其不同的特点，可根据实际工程应用的需求选用合适的编程语言，下面简单介绍下 CoDeSys 的 6 种不同特点的语言。

1、 结构化文本（ST）：优点是可实现复杂运算控制，对编程人员的技能要求高，其缺点是编译时需要将代码转换为机器语言，会导致编译时间长、执行速度慢，且直观性和易操作性差。

2、 梯形图（LD）：与电气操作原理图相对应，其优点它的直观性，电气技术人员易于掌握和学习。缺点是在应对复杂的控制系统编程时往往程序描述性不够清晰。梯形图在国内的工业自动化领域中是使用最多的一种 PLC 编程语言。

3、 功能块（FBD）：以功能块为设计单位，能从控制功能入手，优点是使控制方案的分析和理解变得容易，功能块具有直观性强、容易掌握的特点，有较好的操作性。在应对复杂控制系统时仍可用图形方式清晰描述。缺点是每种功能块要占用程序存储空间，并延长程序执行周期。

4、 指令表（IL）：优点是易于记忆及掌握，与梯形图（LD）有对应关系，便于相互转换和对程序的检查，且编程及调试时不受屏幕大小的限制，输入元素不受限制。缺点和梯形图一样，对复杂系统的程序描述不够清晰。

5、 顺序流程功能图（SFC）：已完成的功能为主线，优点是操作过程条理清楚，便于对程序操作过程的理解和思路；对大型程序可分工设计，采用较灵活的程序结构，节省程序设计时间和调试时间，由于只对活动步进行扫描，因此，可缩短程序执行时间。

6、 连续功能图（CFC）：实际上是功能块图（FBD）的另一种形式。在整个程序中可自定义运算块的计算顺序，易于实现大规模、数量庞大但又不易细分功能的流程运算。在连续控制行业中，得到大量使用。

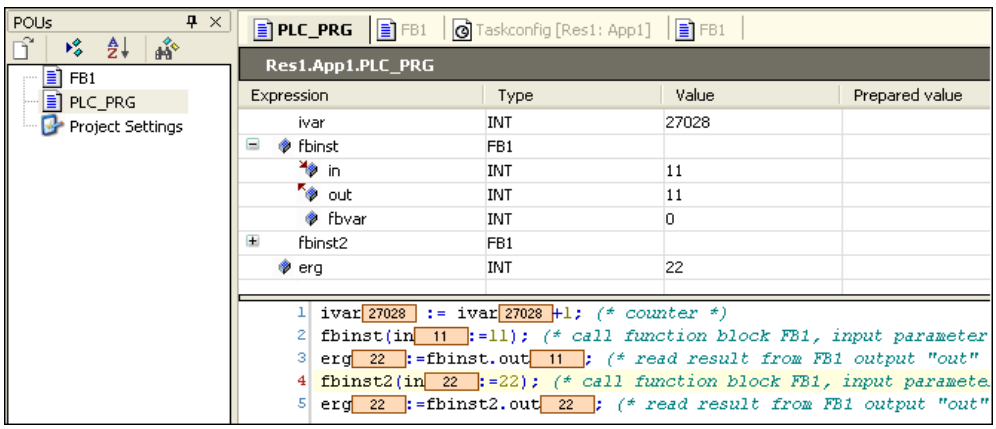
5.2 结构化文本（ST）

5.2.1 结构化文本编程语言简介

结构化文本（ST）是一种高级的文本语言，可以用来描述功能，功能块和程序的行为，还可以在顺序功能流程图中描述步、动作和转变的行为。

结构化文本编程语言是一种高级语言，类似于 Pascal，是一种特别为工业控制应用而开发的一种语言，也是在 CoDeSys 中最常用的一种语言。对于熟悉计算机高级语言开发的人员来说，结构化文本语言更是易学易用，它可以实现选择、迭代、跳转语句等功能。此外，结构化文本语言还易读易理解，特别是当用有实际意义的标识符、批注来注释时，更是这样。在复杂控制系统中，结构化文本可以大大减少其代码量，使复杂系统问题变得简单，缺点是调试不直观，编译速度相对较慢。结构化文本的视图如图 5-1 所示。

图 5-1 结构化文本



5.2.2 程序执行顺序

使用结构化文本的程序执行顺序根据“行号”依次从上至下开始顺序执行，如图 5-2 所示，每个周期开始，先执行行号较小的程序行。

图 5-2 结构化文本程序执行顺序

5.2.3 表达式执行顺序

表达式中包括操作符和操作数，操作数按照操作符指定的规则进行运算，得到结果并返回。操作数可以为变量、常量、寄存器地址、函数等。

【例5.1】 表达式示例。

a+b+c;

3.14*R*R;

ABS(-10)+var1;

如果在表达式中有若干个操作符，则操作符会按照约定的优先级顺序执行：先执行优先级高的操作符运算，再顺序执行优先级低的操作符运算。如果在表达式中具有优先级相同的操作符，则这些操作符按照书写顺序从左至右执行。操作符的优先级如表 5-1 所示。

表 5-1 操作符优先级

操作符	符号	优先级
小括号	()	最高
函数调用	Function name(Parameter list)	
求幂	EXPT	
取反	NOT	
乘法	*	
除法	/	
取模	MOD	
加法	+	
减法	-	
比较	<,>,<=,>=	
等于	=	
不等于	<>	
逻辑与	AND	
逻辑异或	XOR	
逻辑或	OR	最低

5.2.4 指令语句

结构化文本语句表主要有 5 种类型，即赋值语句、函数和功能块控制语句、选择语句、迭代（循环）语句、跳转语句。表 5-2 列举了所有结构化文本用到的语句。

表 5-2 结构化文本语句

指令类型	指令语句	举例
赋值语句	:=	bFan:=TRUE;
函数及功能块控制语句	功能块/函数调名（）；	-
选择语句	IF	IF<布尔表达式>THEN <语句内容>;END_IF
	CASE	CASE<条件变量>OF <数值 1>:<语句内容 1>; … <数值 n>:<语句内容 n>;ELSE <ELSE 语句内容>; END_CASE;
迭代语句	FOR	FOR<变量>:=<初始值>TO<目标值>{BY<步长>}DO <语句内容> END_FOR;
	WHILE	WHILE<布尔表达式> <语句内容>;END_WHILE;
	REPEAT	REPEAT <语句内容>UNTIL <布尔表达式> END_REPEAT;
跳转语句	EXIT	EXIT;
	CONTINUE	CONTINUE;
	JMP	<标识符>: … JMP<标识符>;
返回语句	RETURN	RETURN;
NULL 语句	;	-

1、 赋值语句

赋值语句是结构化文本中最常用的语句之一，作用是将其右侧表达式产生的值赋给左侧的操作数（变量或地址），使用“:=”表示。

具体格式如下：

<变量>:=<表达式>;

【例5.2】 分别给两个布尔型变量赋值，bFan 置 TRUE，bHeater 置 FALSE。

```
VAR
    bFan: BOOL; bHeater:BOOL;
END_VAR
    bFan:=TRUE;
    bHeater:=FALSE;
```

通过使用“:=”赋值语句实现上述功能。

使用时需要注意数据类型的匹配。如果赋值操作符的两侧数据类型不同，应调用数据类型转换函数。例如，rVar1 是 Real 实数类型，iVar1 是 Int 整数类型，当 iVar1 赋值给 rVar1 时，应调用 INT_TO_REAL 的转换函数。

其语句格式如下。

rVar1:= INT_TO_REAL(iVar1);

一行中语句可以有多个，例如，arrData[1]:=3;arrData[2]:=12;该两句指令可以写在一行。

【例5.3】 一行中可有多个数据。

arrData1[i]:=iDataInLine1; arrData2[j]:= iDataInLine2;

函数调用时，函数返回值被赋值作为表达式的值，它应是最新的求值结果。

【例5.4】 函数调用的返回值作为表达式的值。

Str1:=INSERT(IN1:=’CoDe’,IN2:=’Sys’,P:=2);

2、 函数及功能块控制语句

A、 函数控制语句

函数块控制语句用于调用函数，函数调用后直接将返回值作为表达式的值赋值给变量。例如，rVar1:=SIN(rData1);语句中，调用正弦函数 SIN，并将返回值赋值给变量 rVar1。其语句格式如下。

变量:=函数名（参数表）;

【例5.5】 函数控制语句示例。

rResult:=ADD(rData1,rData2);//使用ADD 函数，将 rData1 加 rData2 的结果赋值给变量 rResult。

B、 功能块控制语句

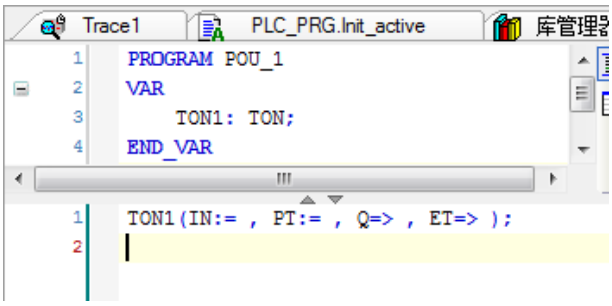
功能块控制语句用于功能块，功能块调用采用将功能块名进行实例化实现调用，如 Timer 为 TON 功能块的实例名，具体格式如下。

功能块实例名:(功能块参数);

如果需要在 ST 中调用功能块，可直接输入功能块的实例名称，并在随后的括号中给功能块的各参数分配数值或变量，参数之间以逗号隔开；功能块调用以分号结束。

例如，在结构化文本中调用功能块 TON 定时器，假设其实例名为 TON1，具体实现如图 5-3 所示。

图 5-3 结构化文本调用功能块选择语句



选择语句是根据规定的条件选择表达式来确定执行它所组成的语句。从大类上可分为IF和CASE两类。

3、 选择语句

A、 IF 语句

用 IF 语句实现单分支选择结构，基本格式如下。

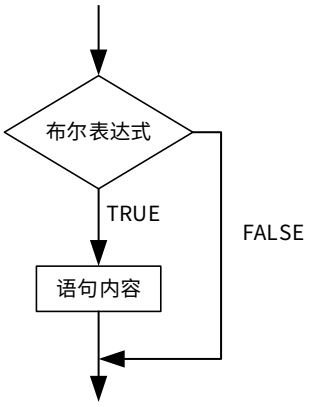
IF <布尔表达式>THEN

<语句内容>;

END_IF

如果使用上述格式，只有当<布尔表达式>为 TRUE 时，才执行语句内容，否则不执行 IF 语句的<语句内容>。语句内容可以为一条语句或者可以为空语句，也可以并列多条语句，该语句表达式执行流程如图 5-4 所示。

图 5-4 简单 IF 语句执行流程



【例5.6】 使用 PLC 判断当前温度是否超过了 60℃，如果超过，始终打开风扇进行散热处理，实现代码如下。

```
VAR
    nTemp:BYTE;      (*当前温度状态信号*)
    bFan:BOOL;       (*风扇开关控制信号*)
END_VAR
    nTemp:=80;
    IF nTemp>60 THEN bFan:=TRUE;
END_IF
```

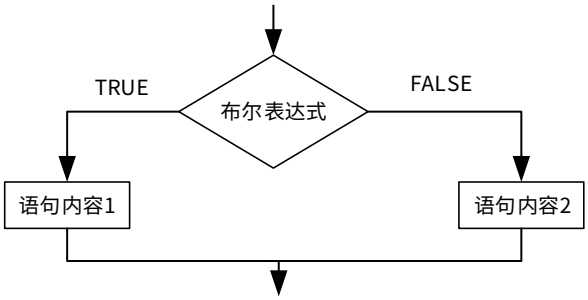
B、 IF…ELSE 语句

用IF…ELSE语句实现双分支选择机构，基本格式如下：

```
IF <布尔表达式> THEN
<语句内容 1>;
ELSE
<语句内容 2>;
END_IF
```

如上表达式先判断<布尔表达式>内的值，如果为 TRUE，则执行<语句内容 1>，如为 FALSE，则执行<语句内容 2>，程序执行流程如图 5-5 所示。

图 5-5 IF…ELSE 语句执行流程



【例5.7】 使用 PLC 判断当温度小于 20℃时，开启加热设备，否则（温度≥20℃）加热设备断开状态。

```
IF nTemp<20 THEN
    bHeating:=TRUE;
ELSE
    bHeating:=FALSE;
END_IF
VAR
    nTemp:BYTE;          (*当前温度状态信号*)
```

```
        bHeating:BOOL;          (*加热器开关控制信号*)
    END_VAR
```

当程序的条件判断式不止一个时，此时，需要再一个嵌套的 IF…ELSE 语句，即多分支选择结构，基本格式如下。

```
    IF<布尔表达式 1>THEN
        IF<布尔表达式 2>THEN
            <语句内容 1>;
            ELSE
            <语句内容 2>;
            END_IF
    ELSE
        <语句内容 3>;
    END_IF
```

如上，在 IF…ELSE 中有放入了一个 IF…ELSE 语句，实现嵌套，下面通过一个例子说明嵌套的使用。

如上表达式先判断<布尔表达式 1>内的值，如果为 TRUE，则继续判断<布尔表达式 2>的值，如果<布尔表达式 1>的值为 FALSE，则执行<语句内容 3>，回到<布尔表达式 2>判断，如果<布尔表达式 2>为 TRUE，则执行<语句内容 1>，反之则执行<语句内容 2>。

【例5.8】 当设备进入自动模式后，如实际温度＞50°C时，才开启风扇并关闭加热器，≤50°C时关闭风扇打开加热器，如在手动模式时，加热器风扇均不动作。

```
    VAR
        bAutoMode: BOOL;        (*手/自动模式状态信号*)
        nTemp:BYTE;             (*当前温度状态信号*)
        bFan:BOOL;              (*风扇开关控制信号*)
        bHeating:BOOL;          (*加热器开关控制信号*)
    END_VAR
    IF bAutoMode=TRUE THEN IF
      nTemp>50 THEN
        bFan:=TRUE;
        bHeating:=FALSE;
    ELSE
        bFan:= FALSE;
        bHeating:= TRUE;
    END_IF
    ELSE
        bFan:= FALSE;
        bHeating:=FALSE;
    END_IF
```

C、 IF…ELSIF…ELSE 语句

此外，多分支选择结构还能通过如下方式来呈现。具体格式如下

```
    IF <布尔表达式 1> THEN
        <语句内容 1>;
    ELSIF <布尔表达式 2> THEN
        <语句内容 2>;
    ELSIF <布尔表达式 3> THEN
```

<语句内容 3>;

...

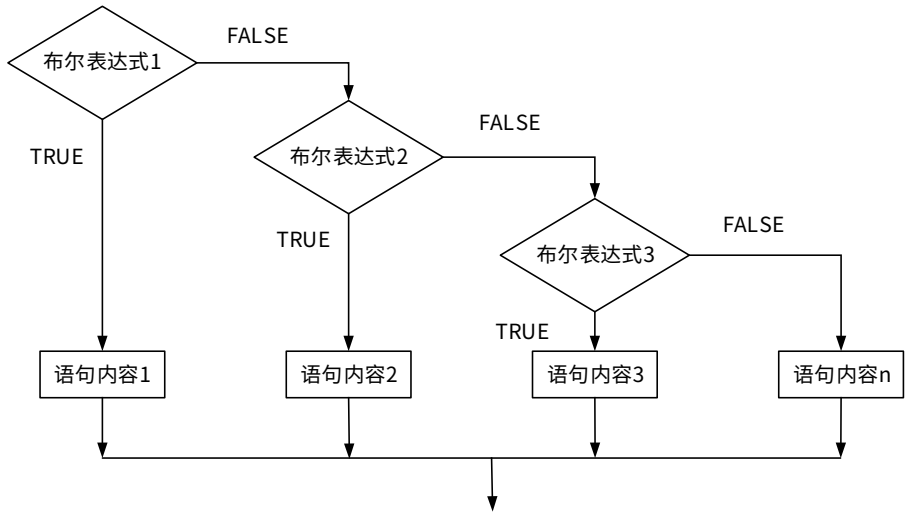
ELSE

<语句内容 n>;

END_IF

如果表达式<布尔表达式 1>为 TRUE，那么只执行指令<语句内容 1>，不执行其它指令。否则，从表达式<布尔表达式 2>开始进行判断，直到其中的一个布尔表达式为 TRUE，然后执行与此布尔表达式对应语句内容。如果布尔表达式的值都不为 TRUE，则只执行指令<语句内容 n>，程序执行流程如图 5-6 所示。

图 5-6 IF...ELSIF...ELSE 语句执行流程



D、 CASE 语句

CASE 语句是多分支选择语句，它根据表达式的值来使程序从多个分支中选择一个用于执行的分支，基本格式如下。

CASE<条件变量>OF

<数值1>:<语句内容1>;

<数值2>:<语句内容2>;

<数值3,数值4,数值5>:<语句内容3>;

<数值6..数值10>:<语句内容4>;

...

<数值n>:<语句内容n>;

ELSE

<ELSE 语句内容>;

END_CASE;

CASE 语句按照下面的模式进行执行：

- 如果<条件变量>的值为<数值 i>，则执行指令<语句内容 i>。
- 如果<条件变量>没有任何指定的值，则执行指令<ELSE 语句内容>。
- 如果条件变量的几个值都需要执行相同的指令，那么可以把几个值相继写在一起，并且用逗号分开。这样，共同的指令被执行，如上程序第四行。
- 如果需要条件变量在一定的范围内执行相同的指令，可以通过写入初、终值，以两个点分开。这样，共同的指令被执行，如上程序第五行。

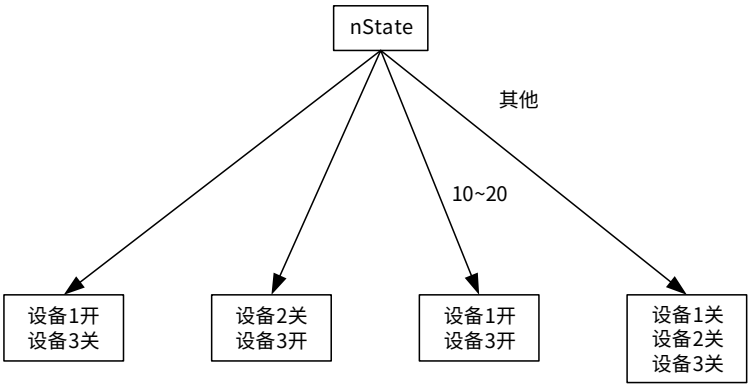
【例5.9】 当前状态为 1 或 5 时，设备 1 运行和设备 3 停止；状态为 2 时，设备 2 停止和设备 3 运行；如当前状态在 10 至 20 之间，设备 1 和设备 3 均运行，其他情况时要求设备 1，2，3 均停止，具体实现的代码如下：

VAR

```
        nDevice1,nDevice2,nDevice3:BOOL;      (*设备 1..3 开关控制信号*)
        nState:BYTE;          (*当前状态信号*)
    END_VAR

    CASE nState OF 1,
    5:
        nDevice1:=TRUE;
        nDevice3:=FALSE;
    2:
        nDevice2:=FALSE;
        nDevice 3:=TRUE;
    10..20:
        nDevice1:=TRUE;
        nDevice 3:=TRUE;
    ELSE
        nDevice1:=FALSE;
        nDevice2:=FALSE;
        nDevice3:=FALSE;
    END_CASE;
```

CASE 语句流程见图 5-7 所示，当 nState 为 1 或者 5 时，设备 1 开，设备 3 关；nState 为 2 时，设备 2 关，设备 3 开；nState 为 10~20 时，设备 1 关，设备 3 开；其他情况则设备 1 关，设备 2 关，设备 3 关。

图 5-7 CASE 语句例流程



4、 迭代语句

迭代语句主要用于重复执行的程序，在 CoDeSys 中，常见的迭代语句有 FOR，REPEAT 及 WHILE 语句，下面对这种语句做详细解释。

A、 FOR 循环

FOR 循环语句用于计算一个初始化序列，当某个条件为 TRUE 时，重复执行嵌套语句并计算一个迭代表达式序列，如果为 FALSE，则终止循环，具体格式如下。

```
FOR<变量>:=<初始值>TO<目标值>{BY<步长>}DO
    <语句内容>
END_FOR;
```

FOR 循环的执行顺序如下：

- 计算<变量>是否在<初始值>与<目标值>的范围内。
- 当<变量>小于<目标值>，执行<语句内容>。

- 当<变量>大于<目标值>，则不会执行<语句内容>。
- 当每次执行<语句内容>时，<变量>总是按照指定的步长增加其值。步长可以是任意的整数值。如果不指定步长，则其缺省值是 1。当<变量>大于<目标值>时，退出循环。

从某种意义上理解，FOR 循环的原理就像复印机，在复印机上先预设要复印的份数，在此即循环的条件，当条件满足时，即复印的张数等于设置的张数，停止复印。

FOR 循环是循环语句中最常用的一种，FOR 循环体现了一种规定次数、逐次反复的功能，但由于代码编写方式不同，也可以实现其他循环功能，下面通过一个实例，演示如何使用 FOR 循环。

【例5.10】 使用 FOR 循环实现 2 的 5 次方计算。

```
VAR
    Counter:BYTE;       (*循环计数器*)
    Var1:WORD;          (*输出结果*)
END_VAR
FOR Counter:=1 TO 5 BY 1 DO
    Var1:=Var1*2;
END_FOR;
```

假设 Var1 的初始值是 1，那么循环结束后，Var1 的值的为 32。

注意：如果<目标值>等于<变量>的极限值，则会进入死循环。假设【例 5.10】中的计数变量 Counter 的类型为 SINT（-128 至 127），将<目标值>设定为 127 时，控制器会进入死循环。故不能对<目标值>设极限值。

B、 WHILE 循环

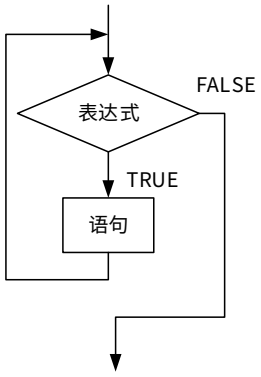
WHILE 循环与 FOR 循环使用方法类似。二者的不同之处是，WHILE 循环的结束条件可以是任意的逻辑表达式。即可以指定一个条件，当满足该条件时，执行循环，具体格式如下。

```
WHILE <布尔表达式>
    <语句内容> ;
END_WHILE;
```

WHILE 循环的执行顺序如下：

- 计算<布尔表达式>的返回值。
- 当<布尔表达式>的值为 TRUE 时，重复执行<语句内容>。
- 当<布尔表达式>初始值为 FALSE，那么指令<语句内容>不会被执行，跳转至 WHILE 语句的结尾。其流程图见图 5-8 所示。

图 5-8 WHILE 语句流程



WHILE 语句在像在工程中控制一台电机，当按下“启动”按钮时（布尔表达式为 TRUE 时），电机不停的旋转，当按下停止按钮后（布尔表达式为 FALSE 时），电机也立即停止。下面通过一个实例，演示如何使用 WHILE 循环。

注意：如果<布尔表达式>的值始终为 TRUE，那么将会产生死循环，应当避免死循环的产生。可以通过改变循环指令的条件来避免死循环的产生。例如：利用可增减的计数器避免死循环的产生。

【例5.11】 只要计数器不为零，则始终执行循环体内的程序。

```
VAR
    Counter: BYTE;          (*计数器*)
    Var1:WORD;
END_VAR
WHILE Counter<>0 DO
    Var1 := Var1*2;
    Counter := Counter-1;
END_WHILE
```

在一定的意义上，WHILE 循环比 FOR 循环的功能更加强大，这是因为在执行循环之前，WHILE 循环不需要知道循环的次数。因此，在有些情况下，只使用这两种循环就可以了。然而，如果清楚地知道了循环的次数，那么 FOR 循环更好，因为 FOR 循环可以避免产生死循环。

C、 REPEAT 循环

REPEAT 循环与 WHILE 循环不同，因为只有在指令执行以后，REPEAT 循环才检查结束条件。这就意味着无论结束条件怎样，循环至少执行一次。

具体格式如下。

```
REPEAT
    <语句内容>
UNTIL
    <布尔表达式>
END_REPEAT;
```

REPEAT 循环的执行顺序如下：

- 当<布尔表达式>的值为 FALSE 时，执行<语句内容>。
- 当<布尔表达式>的值为 TRUE 时，停止执行<语句内容>。
- 在第一次执行<语句内容>后，如果<布尔表达式>的值为 TRUE，那么<语句内容>只被执行一次。

注意：如果<布尔表达式>的值始终为 TRUE，那么将会产生死循环，应当避免死循环的产生。可以通过改变循环指令部分的条件来避免死循环的产生。例如：利用可增减的计数器避免死循环的产生。

下面通过一个实例，演示如何使用 REPEAT 循环。

【例5.12】 REPEAT 循环举例，当计数器为 0 时，则停止循环。

```
VAR
    Counter: BYTE;
END_VAR


REPEAT
    Counter := Counter+1;
UNTIL
    Counter=0
END_REPEAT
```

此例的结果为，每个程序周期都进入该 REPEAT 循环，Counter 为 BYTE（0~255），即每个周期内都进行了 256 次自加一计算。

因为之前提到的“这就意味着无论结束条件怎样，循环至少执行一次”，所以每当进入该 REPEAT 语句时，Counter 先为 1，每个周期内都执行了 256 遍的 Counter:=Counter+1 指令，直到将 Counter 变量累加至溢出为 0，跳出循环。再被加到溢出，如此往复。

5、 跳转语句

A、 EXIT 语句

如果FOR、WHILE和REPEAT三种循环中使用了EXIT指令，那么无论结束条件如何，内循环立即停止，具体格式如下。

```
EXIT;
```

【例5.13】 使用 EXIT 指令避免当使用迭代语句时中出现除零。

```
FOR Counter:=1 TO 5 BY 1 DO INT1:= INT1/2;
IF INT1=0 THEN
    EXIT;            (*避免程序除零*)
END_IF
    Var1:=Var1/INT1;
END_FOR
```

当 INT1 等于 0 时，则 FOR 循环结束。

B、 CONTINUE 语句

该指令为 IEC61131-3 标准的扩展指令，CONTINUE 指令可以在 FOR、WHILE 和 REPEAT 三种循环中使用。

CONTINUE 语句中断本次循环，忽略位于它后面的代码而直接开始一次新的循环。当多个循环嵌套时，CONTINUE 语句只能使直接包含它的循环语句开始一次新的循环，具体格式如下。

```
CONTINUE;
```

【例5.14】 使用 CONTINUE 指令避免当使用迭代语句时中出现除零。

```
VAR
    Counter: BYTE;          (*循环计数器*)
    INT1,Var1: INT;         (*中间变量*)
    Erg: INT;               (*输出结果*)
END_VAR
FOR Counter:=1 TO 5 BY 1 DO
INT1:= INT1/2;
IF INT1=0 THEN
    CONTINUE;               (* 为了避免除零 *)
END_IF
    Var1:=Var1/INT1;        (*只有当 INT1 不等于 0 的情况下才执行*)
END_FOR;
Erg:=Var1;
```

C、 JMP 语句

跳转语句，跳转指令可以用于无条件跳转到使用跳转好标记的代码行，具体格式如下。

```
<标识符>:
.
JMP <标识符>;
```

<标识符>可以是任意的标识符，它被放置在程序行的开始。JMP 指令后面为跳转目的地，即一个预先定义的标识符。当执行到 JMP 指令时，将跳转到标识符所对应的程序行。

注意：必须避免制造死循环，可以配合使用 IF 条件控制跳转指令。

【例5.15】 使用 JMP 语句实现计数器在 0..10 范围内循环。

```
VAR
    nCounter: BYTE;
```

```
END_VAR
Label1:nCounter:=0;
Label2:nCounter:=nCounter+1;
IF nCounter<10 THEN
    JMP Label2;
ELSE
    JMP Label1;
END_IF
```

上例中的 Label1 和 Label2 属于标签，不属于变量，故在程序中不需要进行变量声明。

通过 IF 语句判断计数器是否在 0-10 的范围内，如果在范围内，则执行语句 JMPLabel2，程序会在下一个周期跳转到至 Label2，执行程序 nCounter:=nCounter+1，将计数器进行自加 1，反之，则会跳转至 Label1，执行 nCounter:=0，将计数器清零。

此例子中的功能同样可以通过使用 FOR，WHILE 或者 REPEAT 循环来实现。通常情况下，应该避免使用 JMP 跳转指令，因为这样降低了代码的可读性和可靠性。

6、 RETURN 指令

RETURN 指令是返回指令，用于退出程序组织单元（POU），具体格式如下。

```
RETURN;
```

【例5.16】 使用 IF 语句作为判断，当条件满足时，立即终止执行本程序。

```
VAR
    nCounter: BYTE;
    bSwitch: BOOL;          (*开关信号*)
END_VAR
IF bSwitch=TRUE THEN
RETURN;
END_IF;
nCounter:= nCounter +1;
```

当 bSwitch 为 FALSE 时，nCounter 始终执行自加 1，如 bSwitch 为 TRUE 时，nCounter 保持上一周期的数值，立刻退出此程序组织单元（POU）。

7、 空语句

空语句（；）即什么内容都不执行。

8、 注释

注释是程序中非常重要的一部分，它使程序更加具有可读性，同时不会影响程序的执行。在 ST 编辑器的声明部分或执行部分的任何地方，都可以添加注释。在 ST 语言中，有两种注释方法。

1、 多行注释以（*开始，以*）结束。这种注释方法允许多行注释，如图 5-9 所示。

2、 单行注释以“//”开始，一直到本行结束。这是单行注释的方法，如图 5-10 所示。注意 CoDeSysV2 暂不支持此注释方式。

图 5-9 结构化文本语言注释（多行注释）

```
(*
    bOperationActive:=FALSE;
    bOrderActive:=FALSE;
    bRecipeActive:=FALSE;
    bInfoActive:=FALSE;
    bServiceActive:=FALSE;
    bSimulationActive:=FALSE;
*)
    IF iMainAreaIndex = 0 THEN
        bOperationActive:=TRUE;
    ELSIF iMainAreaIndex = 1 THEN
        bOrderActive:=TRUE;
```

图 5-10 结构化文本语言注释（单行注释）

```
// gesture handling:
// only when mouseup was done
IF xRight AND bDragCanStart = FALSE THEN
    xRight := FALSE;
    IF iMainAreaIndex < MAX_MODULES-1 THEN
        iMainAreaIndex := iMainAreaIndex + 1;
        bIndexChanged := TRUE;
    END_IF
```

5.2.5 应用举例

【例5.17】 磁滞功能块 FB_Hystersis。

1、 控制要求

该功能块有三个输入信号，分别为当前实时值输入信号，比较设定值输入信号和偏差值输入信号。此外需要有一个输出值，当输出为 TRUE 时，只有当输入信号 IN1 小于 VAL-HYS 时，输出才切换到 FALSE。当输出信号为 FALSE 时，只有当输入信号 IN1 大于 VAL+HYS 时输出才切换到 TRUE。

功能块 FB_Hystersis 的输入输出变量定义如下。

```
FUNCTION_BLOCK FB_Hysteresis
VAR_INPUT
    IN1:REAL;          //输入信号
    VAL:REAL;          //比较信号
    HYS:REAL;          //磁滞偏差信号
END_VAR
VAR_OUTPUT
    Q:BOOL;
END_VAR
```

图 5-11 磁滞过程

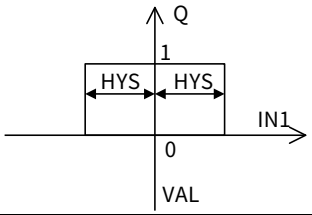
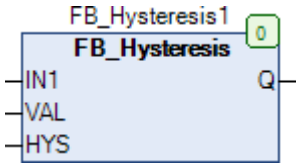


图 5-12 功能块图形



2、 功能块编程

功能块本体用于判断输入信号的程序如下。

```
IF Q THEN
  IF IN1<(VAL-HYS) THEN
      Q:=FALSE;          //IN1 减小
  END_IF
  ELSIF IN1>(VAL+HYS) THEN
      Q:=TRUE;           //IN1 增加
END_IF
```

3、 功能块应用

FB_Hysteresis 功能块可以用于位信号控制，其中 IN1 连接过程变量 rActuallyValue，VAL 链接过程设定值 rSetValue，rTolerance 为所需的控制偏差，程序的声明部分如下。

```
PROGRAM POU
VAR
    fbHysteresis:FB_Hysteresis;   //fbHysteresis 是FB_Hysteresis 功能块的实例
    rActuallyValue:REAL;          //实际测量值
    rSetValue:REAL;               //过程设定值
    rTolerance:REAL;              //偏差设定值
    bOutput AT%QX0.0:BOOL;        //位信号输出
END_VAR
```

程序的本体如下：

```
fbHysteresis(IN1:=rActuallyValue , VAL:=rSetValue , HYS:=rTolerance , Q=>bOutput );
```

如上的程序部分同样也可以用如下的程序表示，其结果是一样的。

```
fbHysteresis(IN1:=rActuallyValue , VAL:=rSetValue , HYS:=rTolerance); bOutput:=fbHysteresis.Q;
```

图 5-13 磁滞功能块程序运行结果

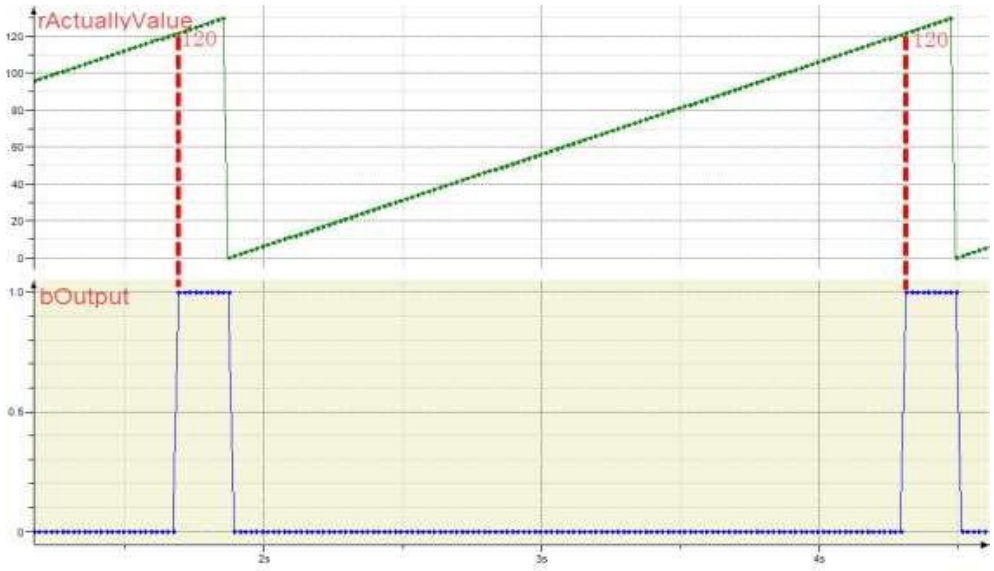


图 5-13 为实际程序运行的结果图，程序中将 rSetValue 设定为 100，rTolerance 设定为 20。当 rActuallyValue 的数值由 0 开始递增，达到 120 时，bOutput 信号被置为 TRUE，随后 rActuallyValue 降为 0 时，bOutput 也变为 FALSE，理论上降到 80 时，bOutput 就会变为 FALSE。

【例5.18】 时滞功能块 FB_Delay。

功能块 FB_Delay 是时滞功能块，它与 FB_Hystersis 磁滞功能块不同。输出信号在时间上滞后输入信号的时间称为时滞。生产过程的被控对象常用一阶滤波环节加时滞描述。这里只介绍时滞功能块，不对一阶滤波做过多介绍。

时滞环节的传递函数是：

$$Y(s)=e^{-s\tau}X(s)$$

假设采样周期为 Ts，则离散化后，得到，

$$Y(k)=X(k-N)$$

式中，X 是时滞环节的输入信号；Y 是时滞环节的输出信号。设离散化所采用的采样周期是 Ts，则时滞τ与采样周期 Ts 之比为滞后拍数 N。

- 功能块 FB_Delay 的变量声明

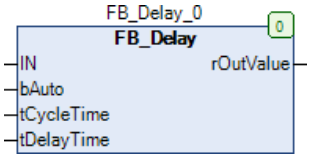
程序中采用数组存储输入信号，数组中存储不同时刻的采样数据，即第 1 单元存储时刻 1×Ts 的采样值，第 i 单元存储时刻 i×Ts 的采样值。时滞时间τ与采样周期 Ts 之比的整数值是 N（N 的小数部分去除后，用 N 表示）。因

此，如果某时刻，输入信号存储在第 N 单元，则经时滞的输出信号应从第 1 存储单元输出。

```
FUNCTION_BLOCK FB_Delay
VAR_INPUT
    IN:REAL;                    //输入信号
    bAuto: BOOL;                //自动手动标志信号
    tCycleTime:TIME;            //采样周期
    tDelayTime:TIME;            //时滞时间
END_VAR
VAR_OUTPUT
    rOutValue:REAL;             //经过时滞后环节处理后的输出
END_VAR
VAR
    N:INT;                      //滞后拍数
    arrValue:ARRAY[0..2047] OF REAL;        //先进先出的数组堆栈
    i:INT;                      //数组的下标，用于输入
    j:INT;                      //数组的下标，用于输出
    fbTrig:R_TRIG;              //将自动信号转化为脉冲
    fbTon:TON;
END_VAR
```

当填写完上述输入输出参数后，通过图形化编程语言调用该功能块图，可以看到图 5-14的效果示意图。

图 5-14 FB_Delay 功能块图形



- 功能块 FB_Delay 的程序本体

```
N:=TIME_TO_INT(tDelayTime)/TIME_TO_INT(tCycleTime);
fbTrig(CLK:= bAuto);
IF fbTrig.Q THEN
    i:=N;
    j:=0;
END_IF
fbTon(IN:= NOT fbTon.Q  , PT:=tCycleTime);
IF fbTon.Q AND bAuto THEN
    i:=(i+1)MOD 2000;
    arrValue[i]:=i;
    j:=(j+1)MOD 2000;
    rOutValue:=arrValue[j];
END_IF
```

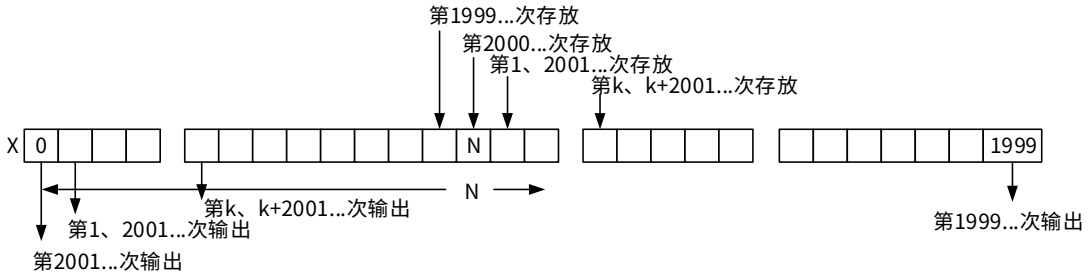
功能块本体采用两个下标窗口来管理输入和输出信号的存取和输出。输入信号数据存放在数组 X 的 i 下标地址，初始值等于滞后拍数。输出信号在数组 X 的 j 下标地址，输出初始值等于 0。采用取模的方法确定每次的存放和输出的地址，并在每次执行操作后，将原地址进行加 1 运算。保证下次执行操作时，存放该次的输入和前 N 次输入信号作为该次输出。

数组存储单元的数量决定时滞大小和采样周期有关。当时滞越大，采样周期越小时，所需的存储器单元越多。一般

可根据应用的大小使滞后拍数 N 大于存储器单元总数即可。

示例中，要求滞后拍数 N 小于 2000（数组长度为 2048）。此外，数组的存储单元从 0 地址开始，实际应用从地址 0 开始。图 5-15 显示了输入窗口和输出窗口的关系。

图 5-15 输入/输出窗口的关系



使用功能 FB_Delay 注意事项：

- 滞后拍数 N 与时滞、采样周期有关，程序中用运行状态切换到自动状态的信号作为初始值设定的脉冲信号。
- 该功能块可与一阶滤波环节组合，用于模拟实际生产过程，进行控制系统仿真研究。

【例5.19】 计算最大、最小和平均值。

在一些工业控制中，常常需要计算若干测量值的平均值，最大值及最小值。下面使用结构化文本编程语言实现此类应用。

1、 控制要求

有一个窑炉，内共有 32 个点需要测量温度值，分别将这 32 个点的最大值，最小值及平均值计算出来。

2、 程序编写

程序中分别定义最大值、最小值、累计总和及平均值，具体变量的定义如下所示。

```
PROGRAM PLC_PRG
VAR
    rMaxValue:REAL;     //最大值
    rMinValue:REAL;     //最小值
    rSumValue:LREAL;    //总和
    rAvgValue:REAL;     //平均值
    arrInputBuffer AT%IW100 :ARRAY[1..32] OF REAL;  //输入源数据
    i:INT;
END_VAR
```

主体程序如下，使用 FOR…DO 语句扫描所有输入通道，计算平均值、最大和最小值，此外计算总和。

```
rSumValue:=0;
FOR i:=1 TO 32 BY 1 DO
rSumValue:=REAL_TO_LREAL(arrInputBuffer[i])+rSumValue;
    IF arrInputBuffer[i]> rMaxValue THEN
        rMaxValue:=arrInputBuffer[i];
    END_IF
    IF arrInputBuffer[i]< rMinValue THEN
        rMinValue:=arrInputBuffer[i];
    END_IF
END_FOR;
rAvgValue:=rSumValue/32;
```

5.3 梯形图（LD）、功能块（FBD）

5.3.1 梯形图、功能块图编程语言简介

在IEC61131-3的标准中定义了两种图形类编程语言。即梯形图（LadderDiagram，LD）编程语言和功能块图（FunctionBlockDiagram，FBD）编程语言。梯形图编程语言用一系列梯级组成梯形图，表示工业控制逻辑系统中各变量之间的关系。功能块图编程语言用一系列功能块的连接表示程序组织单元的本体部分。

- 梯形图（LD）

梯形图起源于美国，最初基于图形表示的继电器逻辑，用于可编程逻辑控制器（PLC）的编程。它是 PLC 编程中最广泛使用的图形化语言之一。

梯形图的基本结构：

1、 **电力轨线：**左侧电力轨线，名义上为功率流的起点；右侧电力轨线，功率流的终点。功率流从左向右沿着水平梯级通过各个触点、功能、功能块、线圈等提供能量。

2、 **触点与线圈：**触点，代表布尔变量的状态（如开关的状态）；线圈，代表实际设备的状态（如电动机的启动状态）。每一个触点和线圈对应 PLC 存储器中的一个存储单元。

3、 **功能与功能块：**对应 IEC1131-3 中的标准库或用户自定义的功能或功能块。

梯形图的逻辑解算：根据梯形图中各触点的状态和逻辑关系，求出与图中各线圈对应的编程元件的状态，这一过程称为梯形图的逻辑解算。

软继电器：在梯形图中，某些编程元件沿用了传统继电器的名称，如线圈和触点，但它们实际上是存储单元（软继电器），每一个软继电器与 PLC 存储器中的映像寄存器的一个存储单元相对应。

TRUE/ON 状态：如果存储单元为“TRUE”，对应的软继电器的线圈“通电”，其常开触点接通，常闭触点断开。

FALSE/OFF 状态：如果存储单元为“FALSE”，对应的软继电器的线圈和触点的状态与上述相反。

- 功能块图（FBD）

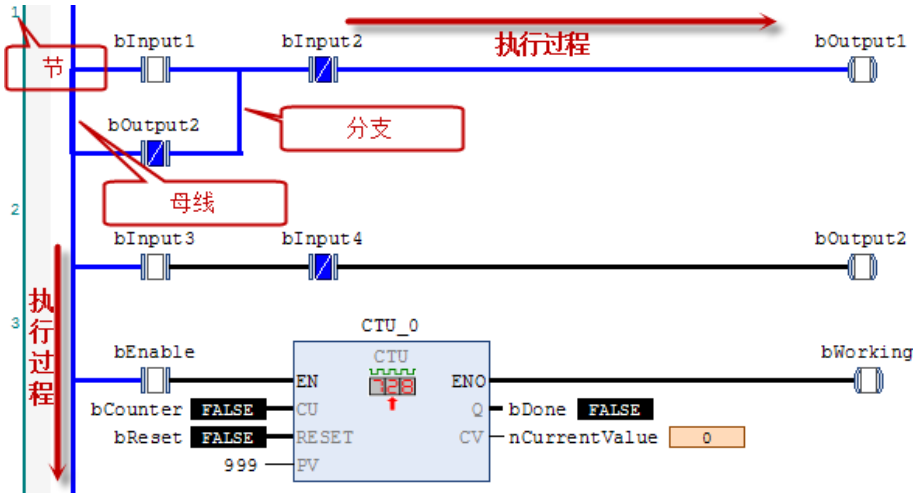
功能块图用来描述功能、功能块和程序的行为特征，还可以在顺序功能流程图中描述步、动作和转变的行为特征。功能块图与电子线路图中的信号流图非常相似，在程序中，它可看作两个过程元素之间的信息流。功能块图普遍地应用在过程控制领域。

功能块用矩形块来表示，每一功能块的左侧有不少于一个的输入端，在右侧有不少于一个的输出端，功能块的类型名称通常写在块内，但功能块实例的名称通常写在块的上部，功能块的输入输出名称写在块内的输入输出点的相应地方。

5.3.2 程序执行顺序

梯形图和功能块图的执行顺序相仿，都是按照从左至右，从上到下的顺序进行执行，如图 5-16 所示。

图 5-16 程序执行顺序



母线：梯形图采用网络结构，一个梯形图的网络以左母线为界。在分析梯形图的逻辑关系时，为了借用继电器电路图的分析方法，可以想象左右两侧母线（左母线和右母线）之间有一个左正右负的直流电源电压，母线之间有“能流”从左向右流动。右母线不显示。

节：梯形图网络结构中最小单位，从输入条件开始，到一个线圈的有关逻辑的网络称为一个节。在编辑器中，节垂直排列。在 CoDeSys 中，每个节通过左侧的一系列节号表示，包含输入指令和输出指令，由逻辑式、算术表达式、程序、跳转、返回或功能块调用指令所构成。要插入一个节，可以使用命令插入节或从工具箱拖动它。一个节所包含的元素都可以通过在编辑器中拖放来进行复制或移动。梯形图执行时，从标号最小的节开始执行，从左到右确定各元素的状态，并确定其右侧连接元素的状态，逐个向右执行，操作执行的结果由执行控制元素输出。然后进行下一节的执行过程。图 5-16 显示了梯形图的执行过程。

能流：图 5-16 中左侧的加粗蓝色线即为能流，可以理解为一个假想的“概念电流”或“能流”（PowerFlow）从左向右流动，这一方向与执行用户程序时的逻辑运算的顺序是一致的。能流只能从左向右流动。利用能流这一概念，可以帮助我们更好地理解和分析梯形图。

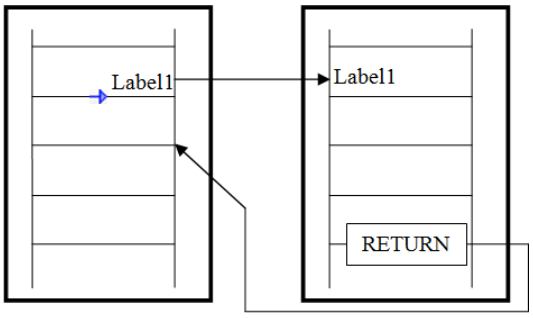
分支：当梯形图中有分支出现时，同样依据从上到下，从左至右的执行顺序分析各图形元素的状态，对垂直连接元素根据上述有关规定确定其右侧连接元素的状态，从而逐个从左向右，从上向下执行求值过程。在梯形图中，没有反馈路径的求值不是很明确。其所有外部输入值与这些有关的触点必须在每个梯级以前被求值。

5.3.3 执行控制

跳转及返回

当满足跳转条件时，程序跳转到 Label 标号的节开始执行，直至该部分程序执行到 RETURN 时，返回原来的节并继续执行。其结构图如图 5-17 所示。

图 5-17 跳转指令执行过程

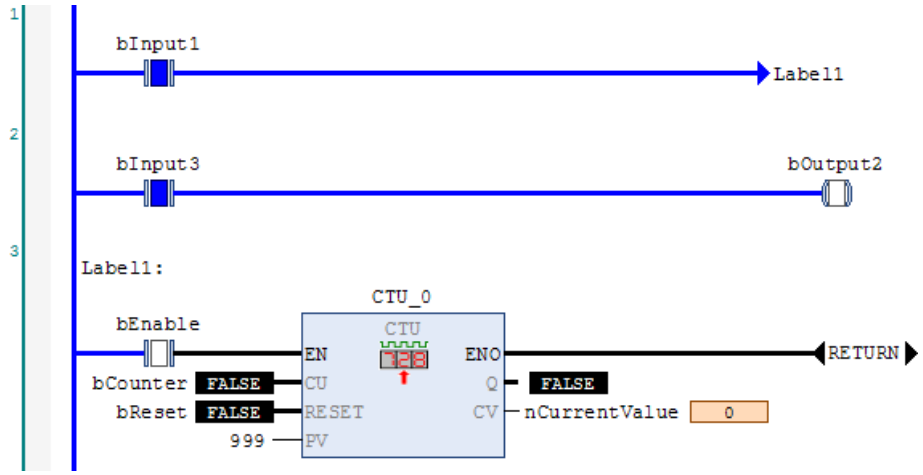


当程序执行到图 5-17 左侧的 Label1 时，此时程序开始执行跳转，直接跳转至图 5-17 右侧，找到以 Label1 为标号的程序段并开始执行接下来的程序，直至程序运行至 RETURN，则跳转程序执行完成并回到图左侧的主程序循环中。

CoDeSys 中使用梯形图的跳转指令和返回指令如下，如图 5-18 所示。

【例5.20】 使用跳转指令执行程序的示例。

图 5-18 跳转指令执行



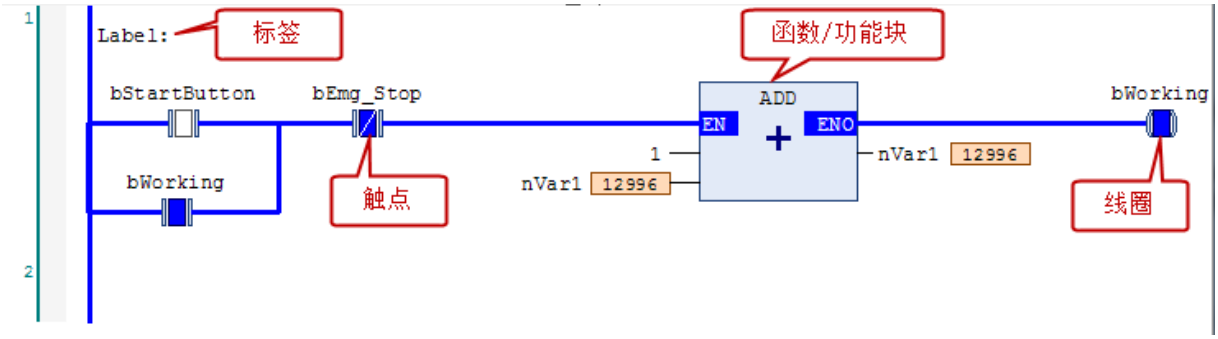
如图 5-18 所示，当 bInput1 被置为 TRUE 时，主程序执行跳转语句，根据标签 Label1，程序跳转至第 3 节的 Label1 程序段，从图中不难看出，尽管第 2 节的 bInput3 被置为 ON，但 bOutput2 始终不会被置为 TRUE，因为程序直接跳过了该语句。只有当 bInput1 为 FALSE 时，且 bInput3 为 TRUE 时，bOutput2 才会 TRUE。

5.3.4 连接元素

IEC1131-3 中的梯形图语言是对各 PLC 厂家的梯形图语言合理地吸收、借鉴，语言中的各图形符号与各 PLC 厂家的基本一致，梯形图编辑器视图如图 5-19 所示。IEC61131-3 的主要的图形符号包括以下内容。

- 基本连接类：电源柜先、连接元素
- 触点类：常开触点、常闭触点、正转换读出触点、负转换触点
- 线圈类：一般线圈、取反线圈、置位（锁存）线圈、复位去锁线圈、保持线圈、置位保持线圈、复位保持线圈、正转换读出线圈、负转换读出线圈
- 功能和功能块：包括标准的功能和功能块以及用户自己定义的功能块

图 5-19 梯形图编辑器

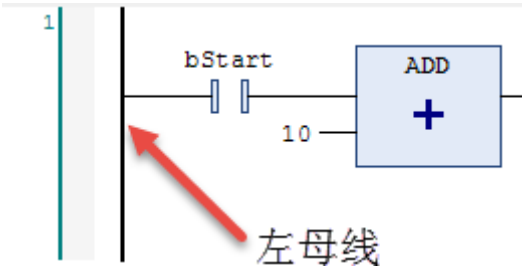


5.3.4.1 线元素

1、 电源轨线（母线）

梯形图电源轨线（Power Rail）的图形元素亦称为母线。其图形表示是位于梯形图左侧，也可称其为左电源母线。左母线的图形如图 5-20 所示。

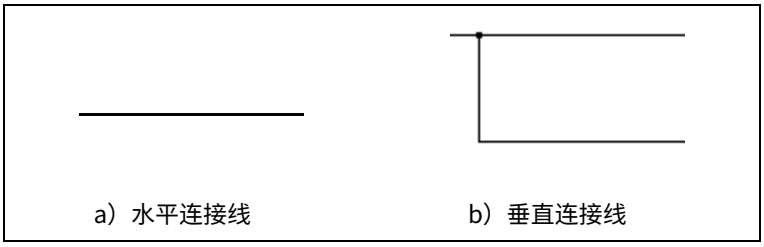
图 5-20 左母线



2、 连接线

在梯形图中，各图形符号用连接线连接，连接线的图形符号有水平线和垂直线，它是构成梯形图的最基本元素。水平和垂直连接线的图形如图 5-21 所示。

图 5-21 连接线



3、 连接元素的传递规则

连接元素的状态从左向右传递，实现能流的流动。状态的传递遵守如下规则：连接到左电源轨线的连接元素，其状

态任何时刻都为 TRUE，它表示左侧电源轨线是能流的起点。右电源轨线类似于电气图中的零电位。

水平连接元素用水平线表示，水平连接元素从它紧靠左侧的图形开始将图形元素的状态传递到靠它右侧的图形元素。

垂直连接元素总是与一个或多个水平连接元素连接，即它由一个或多个水平连接元素在每一侧与垂直线相交组成。垂直连接元素的状态根据与其连接的各左侧水平连接元素的状态或运算表示。

因此，垂直连接元素的状态根据以下列规则确定：

如果左侧所有连接的水平连接元素的状态为 FALSE，则该垂直连接元素的状态为 FALSE。

如果左侧的一个或多个水平连接元素的状态为 TRUE，则该垂直连接元素的状态为 TRUE。

垂直连接元素的状态被传递到与其连接的所有水平连接元素，但不能传递到与其左侧连接元素的所有水平连接元素。

【例5.21】 连接元素及状态传递的示例。

图 5-22 连接元素及状态的示例

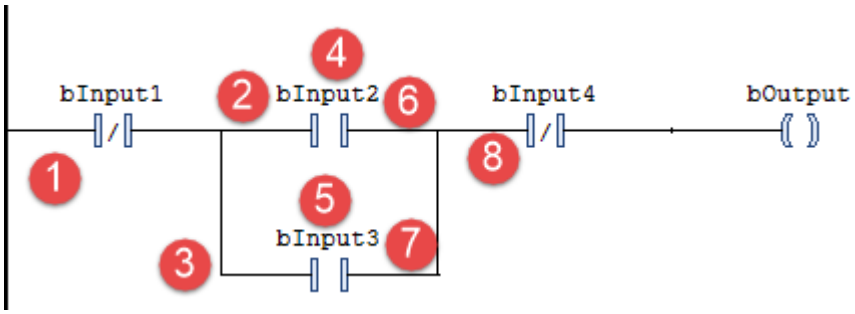


图 5-22 是连接元素及其状态的示例。连接元素 1 与左电源轨线连接，其状态为 TRUE。连接元素 2 与连接元素 1 相连，其状态从连接元素 1 传递，因此，其状态为 TRUE。连接元素 3 是垂直连接元素，它与水平连接元素 1 连接，其状态为 TRUE。

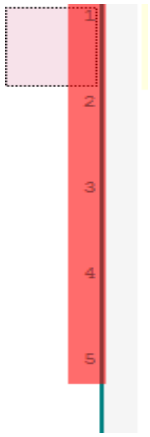
连接元素 2 和 3 分别传递 4，5，由于图形元素 4 和 5 对应的变量 bInput2 和 bInput3 分别为常开触点，因此连接元素 6 和 7 的状态经图形元素的传递而成为 FALSE；由于连接元素 8 的左侧所有的状态都为 FALSE。

连接元素的输入和输出数据类型必须相同。标准中，触点和线圈等图形元素的数据类型并不局限于布尔类型。因此，连接元素的输入输出数据类型相同才能保证状态的正确传递。

5.3.4.2 节

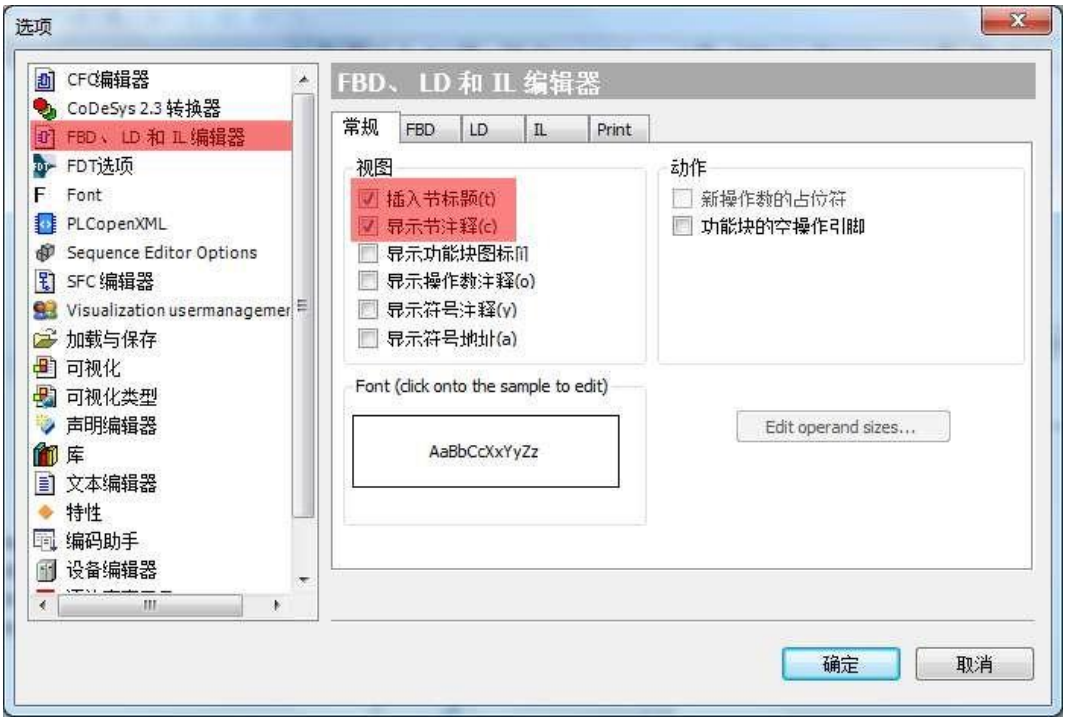
节是 LD 和 FBD 的基本实体，在 LD/FBD 编辑器中，节是数值顺序安排的。每个节在左侧开始有一个节编号，并有一个由逻辑或算数表达式，程序，函数，功能块调用，跳转或返回指令组成的结构。节的示意图如图 5-23 中的红色阴影部分所示，序号依次排列。

图 5-23 节视图



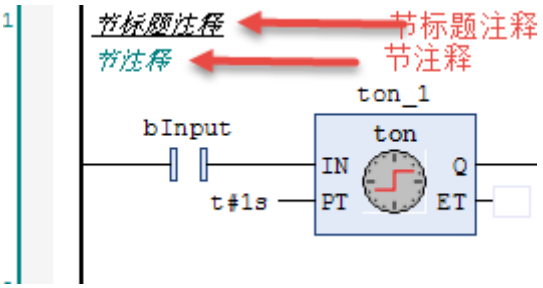
节注释：一个节同时还可以分配一个标题，注释和标号。标题和注释区域的可用性可以通过“选项”→“FBD，LD 和 IL 编辑器”对话框打开或关闭，如图 5-24 所示。

图 5-24 节标题注释和标号功能



如果上述选项被激活，用户可通过鼠标单击节上边界下面，为标题打开一个编辑区域。输入注释则打开标题区域下相应的编辑区域。注释可以是多行的。换行可以通过回车键实现，注释文本的输入是通过[Ctrl]+[Enter]终止的。图 5-25 为节标题注释和节注释的视图。

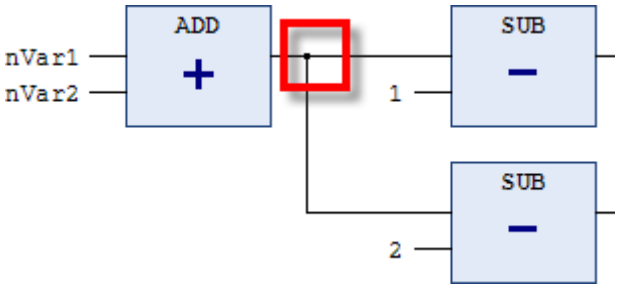
图 5-25 节标题注释与节注释



节标题注释：节也可以通过“切换节注释状态”设置为“注释状态”，设置后该节会像注释一样显示，不会被执行。

节分支：通过在工具箱中插入“ 分支 ”来创建“子节”，如图 5-26 所示则使用了分支功能。

图 5-26 通过分支功能创建子节



5.3.4.3 标签

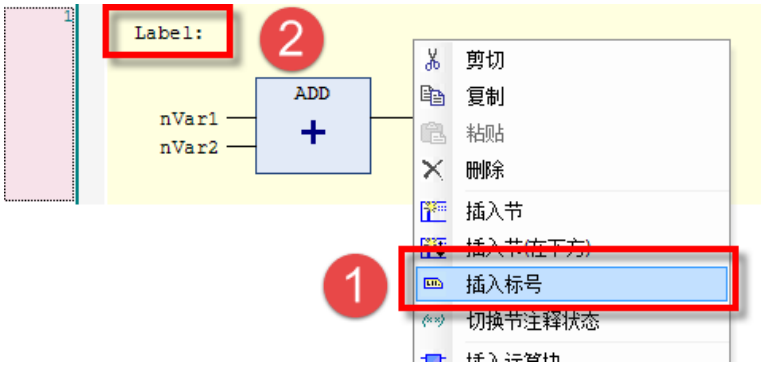
标签是一个可选的标识符且当定义跳转时可以确定其地址。它可以包含任何字符。

在节区域下，每个 FBD，LD 或 IL 节都有一个文本输入区域来定义一个标号。标号是节的一个选择性辨识符，可以在定义跳转时寻址到，它可以包含任意顺序的字符。

FBD 中的使用

在节的空白处鼠标右击，选择“插入标号”，如图 5-27 中的 1 所示，然后会在 2 中弹出 Label: 的标号，用户可以对其进行编辑。

图 5-27 在节中添加标号



5.3.4.4 触点

1、 触点类型

触点是梯形图的图形元素。梯形图的触点（Contact）沿用了电气逻辑图的触点术语，用于表示布尔型变量的状态变化。触点是向其右侧水平连接元素传递一个状态的梯形图元素。

触点可以分为常开触点（Normally Open Contact，NO）和常闭触点（Normally Closed Contact，NC）。常开触点指在正常工况下，触点断开，其状态为 FALSE。常闭触点指在正常工况下，触点闭合，其状态为 TRUE。表 5-3 列出了 CoDeSys 梯形图中常用的触点图形符号及说明。

表 5-3 触点元素的图形符号与说明

类型	图形符号	说明
常开触点		如果该触点对应当前布尔变量值为 TRUE 时，则该触点吸合，如触点左侧连接元素的状态为 TRUE 时，则状态 TRUE 被传递至该触点右侧，使右侧连接元素的状态为 TRUE。反之，当布尔变量值为 FALSE 时，右侧连接元素状态为 FALSE。
常闭触点		如果该触点对应当前布尔变量值为 FALSE 时，则该常闭触点处于吸合状态，如触点左侧连接元素的状态为 TRUE 时，则状态 TRUE 被传递至该触点右侧，使右侧连接元素的状态为 TRUE。反之，当布尔变量值为 TRUE 时，触点断开，则右侧连接元素状态为 FALSE。
插入右触点		可以进行多个触点的串联，在右侧插入触点。多个串联的触点都为吸合状态时，最后一个触点才能传输 TRUE。
插入并联下常开触点		可以进行多个触点的并联，在触点下侧并联插入常开触点。两个并联触点中只需一个触点为 TRUE，则平行线传输 TRUE。
插入并联下常闭触点		可以进行多个触点的并联，在触点下侧并联插入常闭触点。常闭触点默认为吸合状态，如该触点对应当前布尔变量值为 FALSE 时，左侧连接元素的状态为 TRUE 时，则该并联触点右侧传输 TRUE。
插入并联上常开触点		可以进行多个触点的并联，在触点上侧并联插入常开触点。两个并联触点中只需一个触点为 TRUE，则平行线传输 TRUE。

2、 状态传递规则

根据触点的状态和该触点连接的左侧连接元素的状态，按下列规则确定其右侧图形符号的状态。

当触点左侧图形元素状态为 TRUE 时，才能将其状态传递到出点右侧图形元素，根据下列原则进行传递：

- 如果触点状态为 TRUE，则该触点右侧图形元素的状态为 TRUE。
- 如果触点状态为 FALSE，则该触点右侧图形元素的状态为 FALSE。

当触点左侧图形元素状态为 FALSE 时，不管触点的状态如何，都不能将其状态传递到触点右侧图形元素，即右侧图形元素的状态为 FALSE。

从 FALSE→TRUE 在触点左侧图形左侧变化时，其有关变量也从 FALSE→TRUE，则该触点的右侧图形状态从

FALSE→TRUE，并保持一个周期，然后再变为 FALSE，则这称为上升沿触发。

从 TRUE→FALSE 在触点左侧图形左侧变化时，其有关变量也从 TRUE→FALSE，则该触点的右侧图形状态从 TRUE→FALSE，并保持一个周期，然后再变为 TRUE，则这称为下降沿触发。

5.3.4.5 线圈

1、 线圈类型

线圈是梯形图的图形元素。梯形图中的线圈沿用了电气逻辑图的线圈术语，用于表示布尔型变量的状态变化。

根据线圈的不同特性，可以分为瞬时线圈和锁存线圈，锁存线圈分为置位线圈和复位线圈。表 5-4 列出了 CoDeSys 梯形图中常用的线圈图形符号及说明。

表 5-4 线圈元素的图形符号与说明

类型	图形符号	说明
线圈		左侧连接元素的状态被传递到有关的布尔变量和右侧连接元素，如果线圈左侧连接元素的状态为 TRUE，则线圈的布尔变量为 TRUE，反之线圈为 FALSE。
置位线圈		线圈中有一个 S。当左侧连接元素的状态为 TRUE 时，该线圈的布尔变量被置位并且保持，直到由 Reset（复位）线圈的复位。
复位线圈		线圈中有一个 R。当左侧连接元素的状态为 TRUE 时，该线圈的布尔变量被复位并且保持，直到由 Set（置位）线圈的置位。

2、 线圈的传递规则

线圈是将左侧水平或垂直连接元素的状态毫无改变的传递到其右侧水平连接元素的梯形图元素。在传递过程中，将左侧连接的有关变量和直接地址的状态存储到合适的布尔变量中。相反，取反线圈是将其左侧水平或垂直连接元素的状态取反后传递到传递到其右侧水平连接元素的梯形图元素。

置位线圈和复位线圈将左侧水平连接元素状态从 FALSE 至 TRUE 和从 TRUE 至 FALSE 的瞬间保持一个求值周期，其他时间将其左侧水平连接元素状态传递至右侧水平连接元素。

上升沿和下降沿跳变线圈在其左侧水平连接元素状态从从 FALSE 至 TRUE 和从 TRUE 至 FALSE 的瞬间，将有关线圈的变量保持一个求值周期，其他时间将其左侧水平连接元素状态传递至其右侧连接水平元素。

在右侧没有制定只能链接一个元素，因此，用户可在右侧进行扩展，从而达到简化程序的目的。例如，可以在右侧并联其它线圈，如【例 5.22】所示。

【例5.22】 线圈状态的传递。

图 5-28 线圈状态的传递

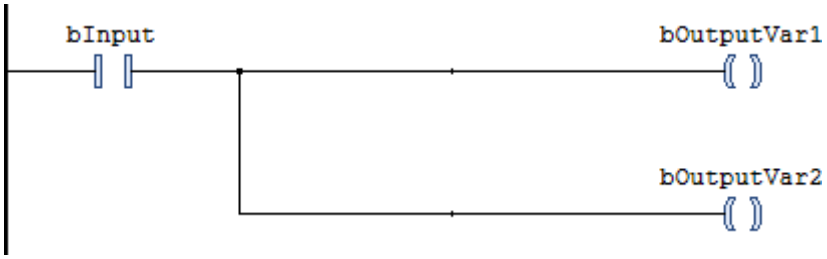


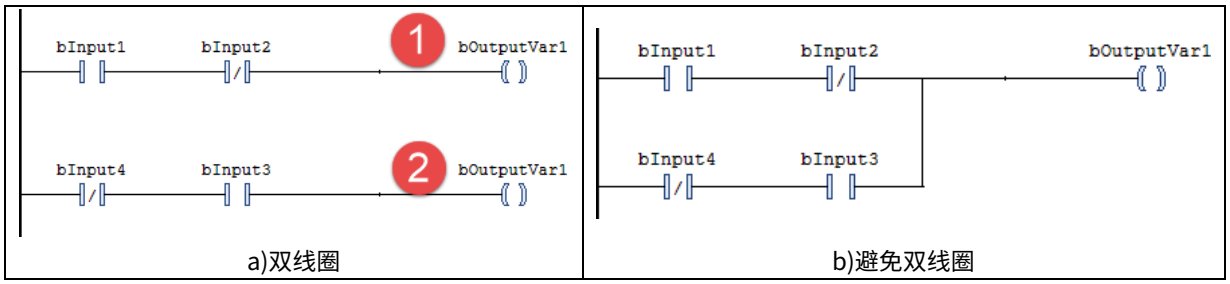
图 5-28 显示了线圈状态的传递过程。图中，当触点 bInput 闭合时，其触点右侧的连接元素状态也为 TRUE，分别经过水平和垂直连接元素后连接到线圈 bOutputVar1 和 bOutputVar2，并将其状态也置为 TRUE。

3、 双线圈

所谓双线圈就是在用户程序中，同一个线圈使用了两次或两次以上，该现象称为双线圈输出。图 5-29 的 a）中有输出变量“bOutputVar1”两个线圈，在同一个扫描周期，两个线圈的逻辑运算结果可能刚好相反，即变量 bOutputVar1 的线圈一个可能“通电”，另外一个可能“断电”。对于变量 bOutputVar1 控制来说，真正起作用的是最后一个 bOutputVar1 的线圈的状态。

bOutputVar1 的线圈的通断状态除了对外部负载起作用外，通过它的触点，还可能对程序中别的变量的状态产生影响。所以一般应避免出现双线圈输出现象，尽量使用图 5-29 中的 b）采用的并联方式来解决双线圈问题。

图 5-29 双线圈示例



只要能保证在同一扫描周期内只执行其中一个线圈对应的逻辑运算，这样的双线圈输出是允许的。下列 3 种情况允许双线圈输出。

- 在判断条件相反的两个程序段（例如自动程序和手动程序）中，允许出现双线圈输出，即同一变量的线圈可以在两个程序段中分别出现一次。实际上 PLC 只执行正在处理的程序段中双线圈元件的一个线圈输出指令。
- 在调用条件相反的两个子程序（例如自动程序和手动程序）中，允许出现双线圈现象。即同一变量的线圈可以在两个子程序中分别出现一次。子程序中的指令只是在该子程序被调用时才执行，没有调用时不执行。
- 为了避免双线圈输出，针对同一变量可以多次使用置位/复位指令。

5.3.4.6 函数及功能块调用

如要实现函数或功能块的调用就要用到运算块，运算块可以代表所有的 POU，包括功能块、函数甚至包括程序。功能块如计时器，计数器等。可以在 FBD，LD 的节中插入。运算块可以有任意的输入，任意输出。函数及功能块的图形符号说明详见表 5-5。

与接点和线圈一起，用户也可以插入功能块和程序。在网络中，它们必须有带布尔值的一个输入和一个输出，并可在相同位置上像接点那样使用，也就是说在 LD 网络的左侧。

表 5-5 函数及功能块图形符号与说明

类型	图形符号	说明
插入运算快		插入函数或功能块，根据弹出的对话框通过鼠标选择想要使用的函数及功能块。适用于对函数和功能块不太熟悉者使用。
插入空运算快		直接插入矩形块，在“？？？”处直接输入函数或功能块名，适用于对函数及功能块较为熟悉的用户。
插入带 EN/ENO 的运算快		只有当 EN 为 TRUE 时，才执行函数或功能块并允许将状态传递至下游。适用于对函数和功能块不太熟悉者使用。
插入带 EN/ENO 的空运算快		插入带 EN/ENO 矩形块，在“？？？”处直接输入函数或功能块名，只有当 EN 为 TRUE 时，才执行函数或功能块并允许将状态传递至下游。适用于对函数及功能块较为熟悉的用户。

梯形图编程语言支持函数和功能块的调用。在函数和功能块调用时，需要注意如下事项：

1、 梯形图中，函数和功能块用一个矩形框表示。函数可以有多个输入参数但只能有一个返回参数。功能块可以有多个输入参数和多个输出参数。

2、 输入列于矩形框的左侧，输出列于矩形框的右侧。

3、 函数和功能块的名称显示在框内的上中部，功能块需要将其实例化，实例名列于框外的上中部。用功能块的实例名作为其在项目中的唯一标识。

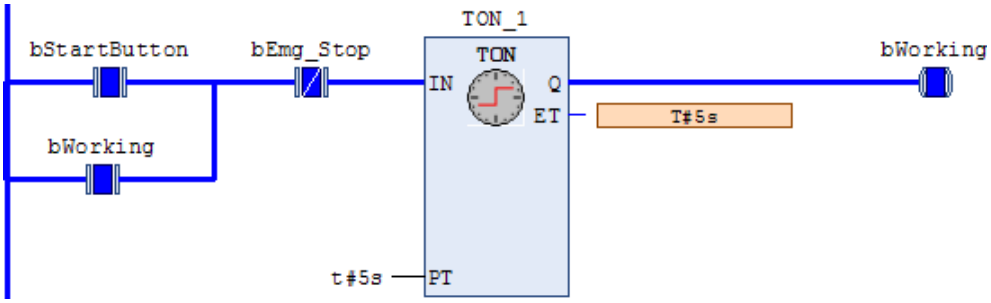
4、 为了保证能流可以通过函数或功能块，每个被调用的函数或功能块至少应有一个输入和输出参数。为了使被连接的功能块执行，至少应有一个布尔输入经水平梯级连接到垂直的左电源轨线。

5、 功能块调用时，可以直接将实际参数值填写在该内部形参变量名的功能块外部连接线处。

【例5.23】 功能块调用实参的设置。

图 5-30 调用 TON 延时 ON 功能块，TON_1 为功能块 TON 实例化后的实例名。功能块的输入形参 PT 设置为 t#5s。输出形参 Q 及 ET，当不需要输出形参如示例中的 ET 时可以不连接变量。

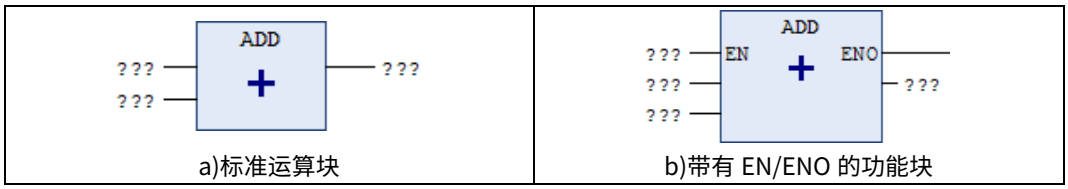
图 5-30 调用功能块时实参的设置



可以看到，功能块 TON 的输出 Q 连接到了线圈 bWorking。表示当触点 bStartButton 为 TRUE 且 bEmg_Stop 为 FALSE 持续时间超过 5s 后，bWorking 为 TRUE。当 bEmg_Stop 断开即为 TRUE 时，bWorking 为 FALSE。

如果没有 EN 和 ENO 的专用输入输出参数，则函数和功能块会被自动执行，且将状态传递至下游。【例 5.23】中，调用了带 EN 和 ENO 的功能块。在工具箱中可以选择插入标准运算块“ 运算块 ”，或插入带有 EN/ENO 的功能块“ 带有EN/ENO的功能块 ”。在编辑器中通过拖拽释放复制或移动。图 5-31 的 a）和 b）为标准的运算块与带有 EN/ENO 的运算块的示图比较。

图 5-31 FBD 中两种运算块比较

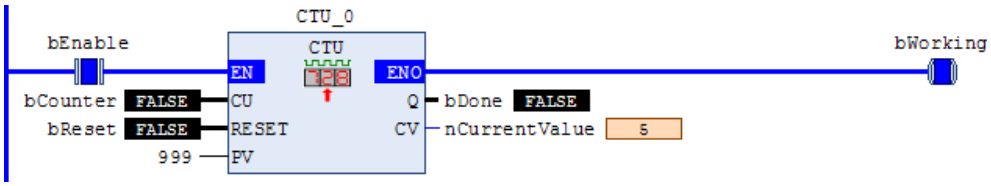


在图 5-31 的 a）中只要前端条件满足，该功能块则会被直接执行，而 b）中，功能块只有当 EN 为 TRUE 时，该功能块才会被执行，否则，即使前端条件全部满足，该功能块不会被程序执行。如果将 b）中的 EN 的输入信号置为常数“TRUE”，此时 a）和 b）的作用是完全一致的。

【例5.24】 调用带 EN 和 ENO 的功能块。

图 5-32 为带 EN 和 ENO 的功能块，布尔输入量 bEnable 用于计数器功能块 CTU_0 的启动，bWorking 作为该功能块被启用的状态变量信号。

图 5-32 调用带 EN 和 ENO 的功能块



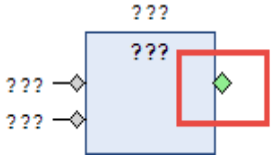
可以看出，当 bCounter 有上升沿触发信号时，则形参输出变量 CV 进行加 1 计算。

- 当 EN 为 FALSE 时，功能块本体定义的操作不被执行，ENO 的值也相应为 FALSE。
- 当 ENO 的值为 TRUE，则说明该功能块的正在被执行。

5.3.4.7 分配

可以将分配功能理解为运算块分配输入/输出。在工具箱中可以选择插入“ 分配 ”工具，将其拖拽至程序的编辑区中，此时编辑区中的预算块对应的输入输出接口处会出现灰色的菱形小图案，读者可以直接将其拖拽至接口处，插入之后，文本字符串“???”可以用要分配的变量的名字替换，也可以使用 ... 按钮调用“输入助手”，此时已完成了对运算块输入/输出接口变量的分配。分配视图如图 5-33 所示。

图 5-33 分配视图



5.3.4.8 跳转执行

跳转执行控制元素：跳转（Jump）执行控制元素用终止与双箭头的布尔信号线表示。跳转信号线开始与一个布尔变量、一个函数或功能块的布尔输出或梯形图的能流线。

跳转分为有条件跳转和无条件跳转。

当跳转信号开始与一个布尔变量、函数或功能块输出时，该跳转是条件跳转。只有程序控制执行到特定网络标号的跳转信号线及该布尔值为TRUE时才发生跳转。

无条件跳转如跳转信号线开始于梯形图的左电源轨线时，该跳转是无条件的。功能块图编程语言中，如果开始于布尔常数1时，该跳转也是无条件的。跳转控制元素图形如表 5-6所列举。

表 5-6 跳转控制元素图形

执行控制类型		执行控制元素的图形符号	说明
无条件跳转	LD 语言	TRUE —▶ Label	直接无条件跳转至 Label
	FBD 语言	—▶ Label	
条件跳转	LD 语言	bInput —┤ ├—▶ Label	当 bInput 为 1 时条件跳转至 Label
	FBD 语言	bInput —▶ Label	
条件跳转返回	LD 语言	bInput —┤ ├—◀ RETURN	当 bInput 为 1 时条件跳转返回
	FBD 语言	bInput —◀ RETURN	

跳转目标：在程序组织单元中，跳转目标是发生跳转的该程序组织单元内的一个标号。它表示跳转发生后，程序将从该目标开始执行。

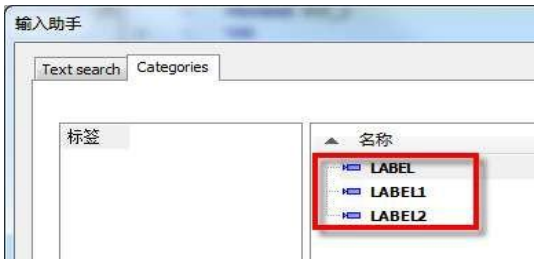
跳转返回：跳转返回（Return）分条件跳转返回和无条件跳转返回两类。

适用于从函数、功能块的条件返回，当条件跳转返回的布尔输入量为TRUE时，程序执行将跳转返回到调用的实体。当布尔输入为FALSE时，程序执行将继续在正常方式进行无条件跳转返回由函数或功能块的物理结束来提供。

如表5-6所示，将RETURN语句直接连接到左轨线表示无条件返回。

跳转执行的组态：在工具箱中插入“→”，插入→跳转之后，自动输入的“???”，跳转目标的标号替换。、可以直接输入目标的标号或者通过点击“→| ... ”浏览键使用输入助手进行选择，如图 5-34 所示，系统会自动筛选可以使用的标号供用户选择。

图 5-34 跳转输入助手



【例 5.25】跳转语句示例。

在气缸控制中，控制气缸电磁阀的伸出信号为 bExtrent，如果发出伸出信号 bExtrent 后 5s 内没有收到伸出传感器的反馈信号 bExtrented_Sensor1，则跳转至报警程序 Alarm，变量声明及程序如下。

```
PROGRAM PLC_PRG
VAR
    bExtrent:BOOL;
    bExtrented_Sensor1:BOOL;
    fb_TON:ton;
```

END_VAR

图 5-35 跳转语句示例程序

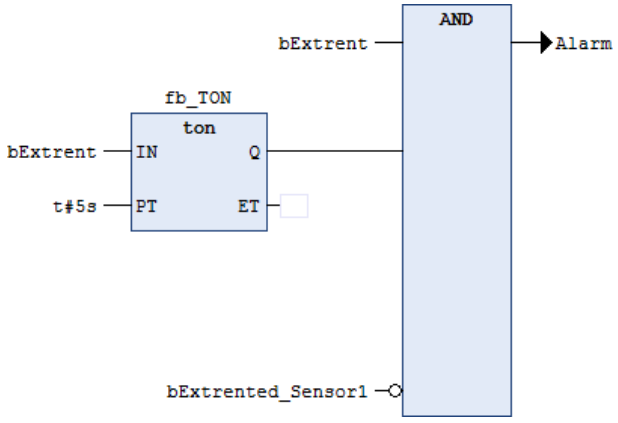


图 5-35 为跳转语句的示例程序，最终当 fb_TON 功能块的输出信号 Q 和 bExtrent 信号同时满足时，与逻辑的输出信号 Alarm 被置为 TRUE。

5.3.5 应用举例

【例5.25】 信号闪烁灯。

控制要求

使用定时器和逻辑函数构成信号闪烁灯系统。该线路输出可使信号灯按一定的周期 ON 和 OFF。

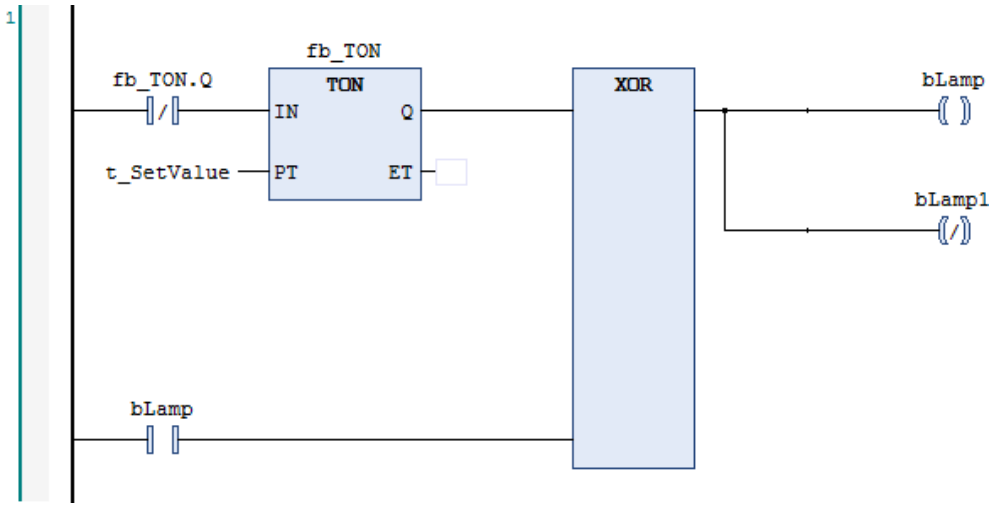
编程

程序实现将 bLamp 和 bLamp1 的交替 ON 和 OFF 实现闪烁信号灯的控制要求。程序如图 5-36 所示的梯形图来实现。

用户可以通过 t_SetValue 用来设定 ON 和 OFF 切换的时间，如下将其设定为 500ms，具体变量的定义如下所示。

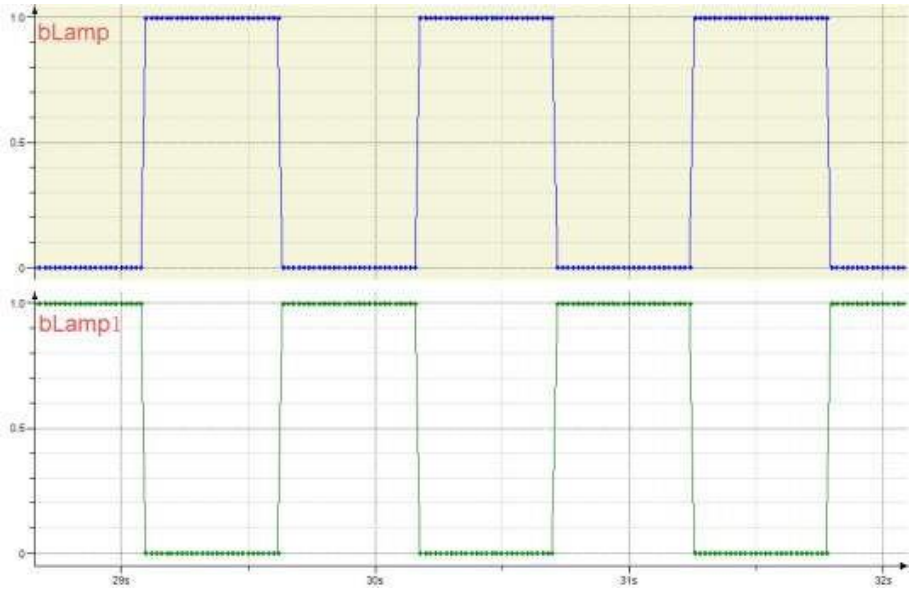
```
PROGRAM
PLC_PRG VAR
    fb_TON:ton;                    //TimeDelay
    t_SetValue:TIME:=t#500ms;      //SetTime
    bLamp AT%QX0.0:BOOL;           //Output0
    bLamp1 AT%QX0.1:BOOL;          //Output1
END_VAR
```

图 5-36 闪烁信号灯的梯形图程序



其输出效果如图 5-37 所示，bLamp 与 bLamp1 的输出曲线正好相反，其状态切换的时间正好为 1s。

图 5-37 闪烁信号灯的输出图形



【例5.26】 pH 值控制系统。

控制要求

在废水处理或者发酵的过程中常常需要采用 pH 控制。pH 控制系统的被控对象具有非线性和时滞性，常用非线性和时滞补偿控制方案。但在简单控制方案中也可以采用如下的控制策略进行控制：当 pH 值测量超过设定的酸度值时，等待一定的时间然后加入一定时间的碱性液体。当 pH 超过设定值时，接点 PHH 闭合，反之，当小于设定值时，其加碱阀为 bValves1。其控制方案为“看一看，调一调”。

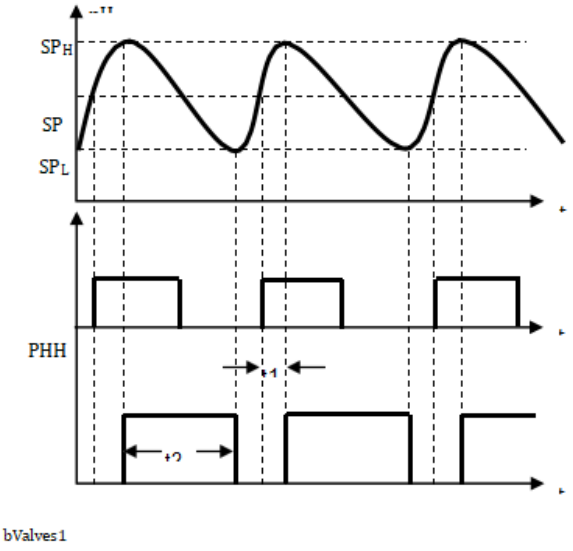
当 pH 控制在线性区域时，可假设控制过程中 pH 的变化呈现线性性特性，即加碱液或酸液进行中和时，pH 的变化呈线性特性。通常，当设定值上限 SPH 和设定值下限 SPL 之间较小时，线性关系成立。

设置发酵过程中 pH 值从 SPL 变化到 SPH 所需时间为 t，加碱后 pH 值从 SPH 变化到 SPL 的时间为 t2，则延时时间可设置为 t1=t/2，加碱控制阀门打开的时间为 t2。

实际 pH 控制的设定值 SP=（SPH+SPL）/2。减少 SPH 和 SPL 之间的差值有利于提高控制精度。

加碱阀 bValves1 的启动条件是定时器 t1 的设定时间到，因此，程序中用 t1.Q 作为启动条件，加碱阀 bValves1 的停止条件是 t2 的设定时间到，因此，程序中用 t2.Q 作为停止条件。

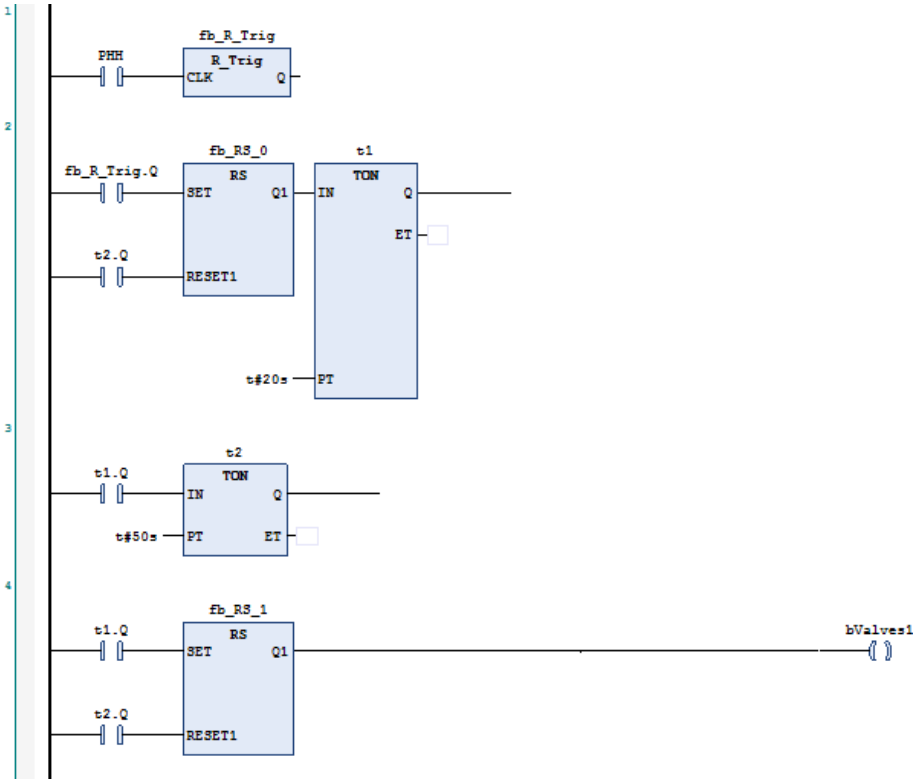
定时器 t1 的启动条件是 pH 到设定值 SP，因此，用触点 PHH 的上升沿触发 fb_Trigger 功能块，并用 RS 功能块将其信号暂存，定时器 t2 的启动条件是定时器 t1 的计时达到。



编程

根据上述的控制要求，使用梯形图编程语言编写 pH 值控制程序，其变量声明及程序如图所示。程序中分别采用了两个定时器。

```
PROGRAM PLC_PRG
VAR
    t1,t2:ton;              //定时器 t1，t2
    PHH:BOOL;               //超过设定值信号
    bValves1 AT%QX0.0 :BOOL;          //加碱阀
    fb_R_Trig:R_Trig;
    fb_RS_0,fb_RS_1:RS;
END_VAR
```



5.4 指令表(IL)

指令表语言（IL）是 IEC 61131-3 标准中的一种低级编程语言，类似于汇编语言。它简单易学、易于实现，并且可以直接下载到 PLC 中。然而，IL 对大型复杂控制问题缺少有效的工具，因此在这些场景中通常不采用 IL 编程语言。尽管如此，IL 作为基础编程语言，其通用性和简单性使其在 PLC 编程中占有重要地位。

5.4.1 指令表编程语言简介

指令表语言是一种由一系列指令组成的编程语言，每条指令在新的一行开始。完整的指令由操作符和紧随其后的操作数组成，操作数是 IEC 61131-3 标准中定义的变量和常量。

指令表语言是一种面向行的语言，类似于汇编语言。一条指令是 PLC 可执行的一项命令，要求严格按行描述，允许空白行作为空指令。

基本格式：

1、 指令格式：操作符，执行具体操作的指令；操作数，指令作用的变量和常量；标号，可选，指令前加标号，后跟冒号；注释，操作数之后可加注释。

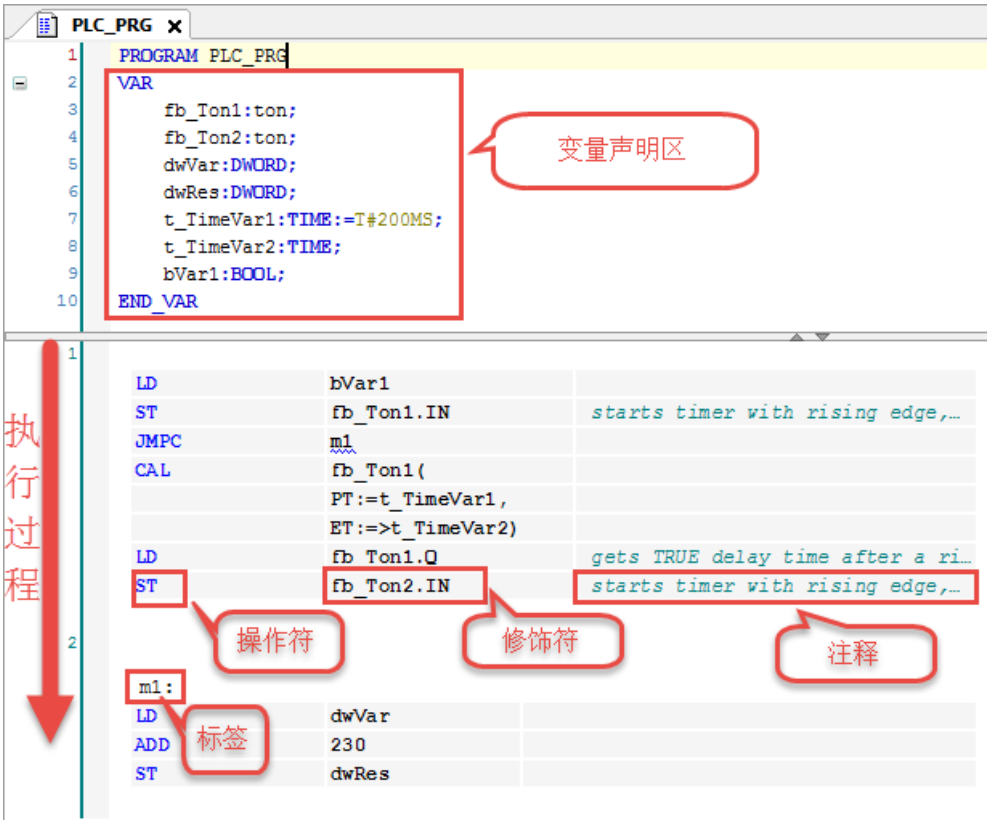
2、 多操作数：有些操作符可带若干个操作数，操作数之间用逗号隔开。

指令表编程语言的特点是：

- 简单易学：指令操作简单，易于掌握，适用于小型、较简单控制系统的编程。
- 操作符功能强大：操作符用于操纵所有基本数据类型的变量、调用函数和功能块。
- 直接解释执行：指令表语言能够直接在 PLC 内部解释执行，适用于大多数 PLC 制造商。
- 错误检出：大多数指令表编程语言编写的程序要在运行时才能检测出错误。
- 语言转换：指令表语言编写的程序较难转换为其他编程语言，其他编程语言编写的程序易于转换为指令表语言。

程序执行顺序

指令表编程语言的执行过程是从上至下的顺序进行执行，如下图所示。



指令表 IL 编辑器中的编程样例代

指令的格式

在指令表编程语言中，指令具有如下的格式。

标号：操作符/函数 操作数注释

【例5.27】 使用指令表实现电机的起保停控制。

```
START:
LD    bStart
OR    bHold
ANDN  bStop
ST    bDone
```

【例 5.27】中的程序用于对某设备的电机进行起保停控制。程序中，标号为 START，指令第 1 行将变量 bStart 的结果存放至累加器中。第 2 行的指令用于将第 1 行指令的结果和 bHold 输出保持信号进行或逻辑运算，结果仍覆盖到累加器中。第 3 行指令用于将第 2 行运算结果和停止信号 bStop 进行取反后的逻辑与运算，结果仍存放在累加器中。第 4 行指令用于将当前累加器中的结果输出至变量 bDone 中。

5.4.2 连接元素

指令表包含一系列指令，每个指令在新一行开始，包含操作符，根据操作的类型，一个或多个操作域用逗号分开，

操作符可以用修饰符扩展。表 5-7 所示的操作符和修饰符。

表 5-7 操作符及修饰符含义

操作符	修饰符	含义	举例
LD	N	将操作数（取反）载入累加器	LD iVar
ST	N	将累加器中的数（取反）存入操作数变量中	ST iErg
S	-	当累加器中的数为真，将操作数（布尔型）设为真	S bVar1
R	-	当累加器中的数为假，将操作数（布尔型）设为真	R bVar1
AND	N,(	累加器中的数和（取反的）操作数逐位进行与运算	AND bVar2
OR	N,(	累加器中的数和（取反的）操作数逐位进行或运算	OR xVar
XOR	N,(	累加器中的数和（取反的）操作数逐位进行异或运算	XOR N,(bVar1,bVar2)
NOT	-	累加器中的数逐位取反	-
ADD	(	将累加器中的数和操作数相加，其结果复制到累加器中	ADD (iVar1,iVar2)
SUB	(	累加器中的数减去操作数，其结果复制到累加器中	SUB iVar2
MUL	(	累加器中的数和操作数相乘，其结果复制到累加器中	MUL iVar2
DIV	(	累加器中的数除以操作数，其结果复制到累加器中	DIV 44
GT	(	检查累加器中的数是否大于操作数，其结果（布尔型）复制到累加器中；>	GT 23
GE	(	检查累加器中的数是否大于或等于操作数，其结果（布尔型）复制到累加器中>=	GE iVar2
EQ	(	检查累加器中的数是否等于操作数，其结果（布尔型）复制到累加器中;=	EQ iVar2
NE	(	检查累加器中的数是否不等于操作数，其结果（布尔型）复制到累加器中;<>	NE iVar1
LE	(	检查累加器中的数是否小于或等于操作数，其结果（布尔型）复制到累加器中;<=	LE 5
LT	(	检查累加器中的数是否小于操作数，其结果（布尔型）复制到累加器中无条件（有条件）跳转到标签;<	LT cVar1
JMP	CN	无条件（有条件）跳转到标签	JMPN next
CAL	CN	（有条件）调用一个程序或功能块（当累加器中的数为正时）	CAL prog1
RET		从当前 POU 中返回，跳回到调用的 POU 中	RET
RET	C	有条件：仅当累加器中的数为真时，从当前 POU 中返回，跳回到调用的 POU 中	RETC
RET	CN	有条件-仅当累加器中的数为假时，从当前 POU 中返回，跳回到调用的 POU 中	RETCN
)	-	评估延迟的操作数	-

注意：

- 累加器总是存储当前值，有后续操作时产生。CAL 的操作数应是一个被调用的功能块实例名。
- NOT 操作的结果是当前结果的位取反，修饰符 N 表示取反操作。RET 操作符不需要操作数
- 修饰符 C 表示有关指令只有在当前运算的结果是布尔为 TRUE（或当操作符布尔值为 FALSE，并与“N”修饰符结合时）才执行。
- 操作符可多于一个修正符，即可同时，也可以只用一个或不用。例如，JMP 操作符可以有 JMP、JMPC、JMPN 三种格式。

左圆括号“（”表示操作符的运算被延缓直到遇到右圆括号“）”。因此，对传统 PLC 中的程序块操作，主控操作等都可以采用该操作符实现。

5.4.2.1 操作数

操作数可以直接表示变量或符号的符号变量。例如：

LDA：表示设置当前值等于符号变量 A 所对应的数值。

AND%IX1.3：表示将当前结果与输入单元 1 的第 3 位进行与逻辑运算，结果作为当前值。

JMPABC：表示当前计算值为布尔值 1 时，从标号 ABC 的位置开始执行。

RET：是无操作数的操作符，当执行到该指令时，程序将返回到原断点后的指令处执行。断点是由于函数调用、功能块调用或中断子程序等造成的。

5.4.2.2 指令

IEC61131-3 标准的指令表编程语言对传统指令表编程语言进行了总结，取长补短，采用函数和功能块，使用数据类型的超载属性等，使编程语言更简单灵活，指令更简单。其主要优点如下：

函数和功能块调用：

标准库调用，定时器和计数器等功能块指令可以通过标准库在指令表编程语言中直接调用，使得复杂的功能实现变得更加简便。

简化编程，通过调用预定义的函数和功能块，可以减少编程工作量，提高代码的可读性和可维护性。

数据类型的过载属性：

运算简化，数据类型的过载属性允许对不同数据类型的变量进行相同的操作，简化了运算过程，使得代码更为简洁和直观。

程序块组合：

使用圆括号，可以方便地使用圆括号将程序块组合在一起，并实现主控等指令的功能，使得复杂逻辑的实现变得更加直观。

边沿检测属性：

微分功能，通过边沿检测属性的方法，对信号进行微分处理，简化了指令集，使得对输入信号的边沿检测变得更加简便。

数据传送指令：

赋值函数 MOVE，数据传送指令可以直接使用赋值函数 MOVE 来实现，使得数据传输操作更加直接和清晰。

5.4.2.3 操作符

在引入操作符的概念之前，需要介绍一个概念，即累加器，他在指令表编程语言中尤为重要。

指令表编程语言提供了一个存储当前结果的累加器，与传统的可编程控制器使用的累加器不同，这种标准累加器存储位数是可变的，即标准指令表编程语言提供了一种存储位数可变的虚拟累加器，其存储位数取决于正在处理的操作数和数据类型。同样，虚拟累加器的数据类型也可以发生变化，以适应最新运算结果的操作数的数据类型。

指令执行过程中，数据存储采用的方法如下：

运算结果：=当前运算结果 操作 操作数

因此，在操作符规定的操作下，当前运算结果与操作数进行由操作符规定的操作运算。运算结果作为新的运算将结果存放回当前运算结果的累加器。

5.4.2.4 修饰符

修饰符有三种，C、N 和 N,(，如表 5-8 中所示，修饰符本身不能独立构成，需要配合之前的操作符才能构成一句完整的语句。

表 5-8 修饰符指令

修饰符	使用	功能
C	结合 JMP，CAL，RET 使用	本指令仅当前面的达式结果为真时执行
N	结合 JMPC，CALC，RETC 使用	本指令仅当前面的达式结果为假时执行
N,(	其他	操作数（而非累加器中的数）取反

修饰符 C 表示有关指令只有当运算结果为 TRUE（或当操作符为 FALSE，并与“N”修饰符结合使用时）才执行。修饰符 N 的逻辑与 C 正好相反。

【例5.28】 修饰符示例。

```
1   LD      TRUE
    ANDN    bVar1
    JMPC    m1
    LDN     bVar2
    st      bRes
2
    m1:
    ld      bVar2
    st      bRes
```

首先，装载 TRUE 至累加器，其次将变量 bVar1 的值取反和累加器的值进行与运算，在此使用了 ANDN 指令，如果使用 AND 则表示直接进行与运算。如果结果为 TRUE，则程序跳转至 m1，反之将变量 bVar2 取反加载至累加器并输出，此条指令使用的是 LDN，也使用了修饰符 N，在此也是取反的意思。

5.4.3 操作指令

指令表编程语言共有 9 大类，其指令说明分别如下。

5.4.3.1 数据存取类指令

数据存取类指令是读取数据存储单元内的操作。标准指令采用 LD 和 LDN 指令表示存取和存

取取反指令。编程语言的格式如下：

LD 操作数 //将操作数规定的数据存储单元中的内容作为当前结果存储。

LDN 操作数 //将操作数规定的数据存储单元中的内容取反后作为当前结果存储。

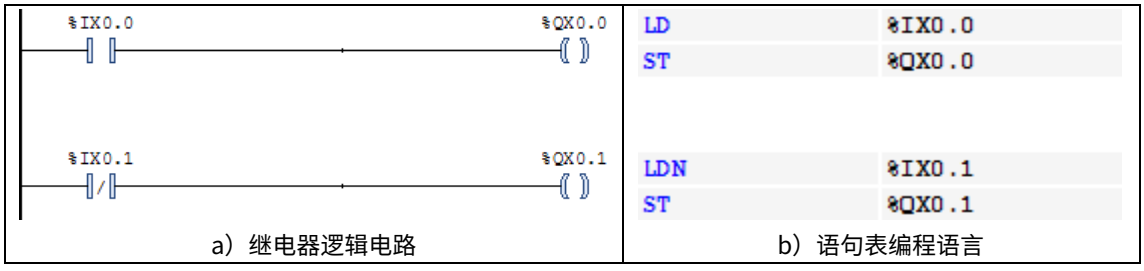
LD 是 Load 的缩写，LDN 是 LoadNot 的缩写。

数据存取类指令的操作对象，即 LD 或 LDN 的操作对象，是操作数。它是对操作数对应的数据存储单元内容进行的读取操作。读取的数据被存放在运算结果累加器，该数据也被称为当前值。

对常开触点的数据读取用 LD 指令，对常闭触点的数据读取用 LDN 指令。

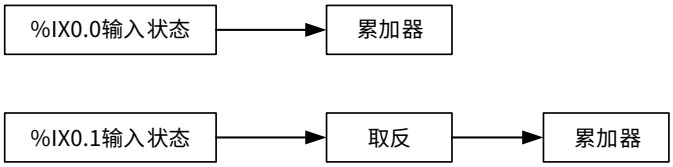
与继电器逻辑电路相似，对常开触点，即动点接点，采用存储 LD 指令，例如，LD%IX0.0 指令执行存取操作数地址为%IX0.0 接点状态的操作。从寄存器看，该操作过程是把地址为%IX0.0 的输入状态传送到了运算结果累加器。图 5-38 的 a）和 b）为继电器逻辑电路和语句表编程语言指令的示例。

图 5-38 LD 和 LDN 指令的示例和操作过程



a）继电器逻辑电路 b）语句表编程语言

图 5-39 LD 和 LDN 指令的操作过程



图为数据存取的执行过程。对常闭接点，即动端接点，采用逻辑取反 LDN 指令，例如 LDN%IX0.1 指令执行存取操作数地址为%IX0.1 接点状态的操作。从寄存器看，该操作过程是把地址为%IX0.1 的输入状态寄存器状态取反，并把取反结果传送到运算结果累加器中。

表 5-9 LD 和 LDN 指令的示例

指令	说明	累加器的数据类型
LD FALSE	当前值等于 FALSE	布尔量
LD TRUE	当前值等于 TRUE	布尔量
LD 3.14	当前值等于 3.14	实数
LD 100	当前值等于 100	整数
LD T#0.5s	当前值等于时间常数 0.5s	时间数据
LD START	当前值等于变量 START 的状态	根据变量 START 的类型而定

5.4.3.2 输出类指令

输出类指令用于将运算结果累加器中的内容传送到输出状态寄存器。标准指令采用 ST 和 STN

指令表示存取和存取取反指令。编程语言的格式如下：

ST 操作数 //将当前结果存储到操作数规定的数据存储单元。

STN 操作数 //将当前结果取反后，存储到操作数规定的数据存储单元。

须指出，在执行 ST 或 STN 指令后，当前运算结果仍被保留在运算结果累加器的存储单元，ST 是 Store 的意思，STN 是 StoreNot 的缩写。

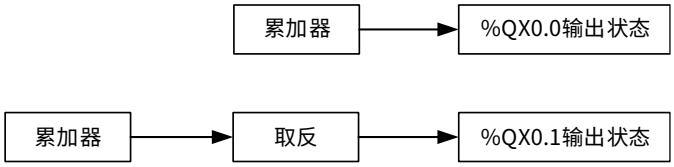
与继电器逻辑电路相似，对线圈用 ST 指令，例如 ST%QX0.0 指令执行输出到%QX0.0 的线圈的操作：从寄存器看，该操作过程是把运算结果累加器的状态传送到地址为%QX0.0 的输出地址中。图 5-40 的 a）和 b）为继电器逻辑电路和语句表编程语言指令的示例。

图 5-40 ST 和 STN 指令的示例和操作过程

a）继电器逻辑电路 b）语句表编程语言

为数据存取的执行过程。使用 STN%QX0.1 指令执行输出到%QX0.1 失励线圈的操作；该操作过程是把运算结果累加器的状态取反，并把取反的结果传送至输出为%QX0.1 的地址。

图 5-41 ST 和 STN 指令的操作过程



5.4.3.3 置位复位类指令

标准指令集采用 S 和 R 指令表示置位和复位类指令，其编程语言的格式如下：

S 操作数 //当前结果为 False 时，将操作数对应的数据存储单元内容设置为 True，并保持。

R 操作数 //当前结果为 True 时，将操作数对应的数据存储单元内容设置为 False，并保持。

这类指令具有记忆属性。执行 S 操作数后，操作数对应的数据存储单元内容被设置为 TRUE，并且该数据存储单元内容被记忆和保持到执行 R 操作数指令，执行 R 操作数指令使操作数对应的数据存储单元内容被设置为 FALSE。同样，该存储单元的内容要保持到执行 S 操作数的指令，并使其内容设置到 TRUE 为止。

S 和 R 指令可以用 SR 和 RS 功能块的调用实现。与功能块比较，其不同点是：S 指令和 R 指令的执行时根据程序中的先后位置确定执行先后次序，因此，优先级的确定和 RS 和 SR 有所不同。此外，功能块要先设置 S 和 R 端，才能执行调用指令。

S 是 Set 的缩写，R 是 Reset 的缩写。S 和 R 指令的示例如表 5-10。

表 5-10 S 和 R 指令的示例

指令			说明
SET:	LD	TRUE	当前值等于 TRUE
	S	START	当前值等于 TRUE,START 变量值置 TRUE，并保持
	LD	FALSE	当前值等于 FALSE
	S	STOP	当前值等于 FALSE,STOP 变量值置 FALSE，并保持
RESET:	LD	TRUE	当前值等于 TRUE
	R	STOP	当前值等于 TRUE,STOP 变量值置 FALSE，并保持

S 指令有条件的输出 STC 指令，R 表是有条件的输出 STCN 指令。因此，当前结果存储器为 TRUE 时，S 操作数指令执行设置输出操作数为置位操作，同样，R 操作数指令执行设置输出操作数为复位的操作，即置位取反的操作。

5.4.3.4 逻辑运算类指令

标准的逻辑运算有：AND(N)、OR(N)、XOR(N)和 NOT 等。编程语言的格式如下：

逻辑运算操作符 操作数 或 逻辑运算操作符 N 操作数

逻辑运算操作符。操作数用于将当前结果存储器内容与操作数对应的数据存储单元内容进行规定的逻辑运算，运算结果作为新的当前结果，存放在当前结果存储器内。

逻辑运算操作符 N。操作数用于将当前结果存储器内容与操作数对应的数据存储单元内容的取反结果进行规定的逻辑运算，运算结果作为当前结果，存放在当前结果存储器内。

【例5.29】 电动机控制程序示例。

```
LD      A     //存取符号变量，启动按钮A的信号
OR      C     //与输出变量C进行或运算，实现触点自保
ANDN    B     //与停止按钮B的取反信号进行与逻辑运算
ST      C     //输出至输出变量C
```

此例为典型的电机控制程序，输入变量 A，B 和输出变量 C 都是符号变量，须在声明部分对其实际地址的赋值。

5.4.3.5 算术运算类

这类指令包括 ADD、SUB、MUL、DIV 和 MOD 等。编程语言的格式如下：

ADD 操作数 //当前结果加操作数对应的数据存储单元内容，运算结果存在当前结果存储器。

SUB 操作数 //当前结果减操作数对应的数据存储单元内容，运算结果存在当前结果存储器。

MUL 操作数 //当前结果乘操作数对应的数据存储单元内容，运算结果存在当前结果存储器。

DIV 操作数 //当前结果除操作数对应的数据存储单元内容，运算结果（商）存在当前结果存储器。

MOD 操作数 //当前结果以操作数对应的数据存储单元内容为模进行模除运算结果（取余数），将运算结果存在当前结果存储器。

【例5.30】 温度补补偿系数的运算示例。

```
LD      273.15        //存取开尔文零度的温度值，即273.15
ADD     rTem1         //与实数变量rTem1进行加法运算
DIV     373.15        //处于设计温度100℃对应的开尔文温度值
ST      rCompensate   //将结果输出至变量rCompensate作为补偿系数
```

【例 5.30】中用于对气体流量进行温度补偿，其中 rTem1 是实际温度，单位是°C。程序在第一行读取 273.15；第二行将温度实际值 rTem1 与 273.15 进行累加并作为当前值。第三行将该当前值除以

设计温度值，结果存放在当前值存储器；第四行将运算结果作为温度补偿值存放在 rCompensate 中。可以看到，程序中 ADD 和 DIV 的运算都是实数数据类型的运算。

5.4.3.6 比较运算类指令

比较指令包括：GT(>)、GE(≥)、EQ(=)、NE(≠)、LE(<)和 LT(≤)。编程语言的格式如下：

GT 操作数 //当前操作数>操作数对应数据存储单元内容，运算结果 TRUE 送当前结果寄存器。

GE 操作数 //当前操作数≥操作数对应数据存储单元内容，运算结果 TRUE 送当前结果寄存器。

EQ 操作数 //当前操作数=操作数对应数据存储单元内容，运算结果 TRUE 送当前结果寄存器。

NE 操作数 //当前操作数≠操作数对应数据存储单元内容，运算结果 TRUE 送当前结果寄存器。

LE 操作数 //当前操作数<操作数对应数据存储单元内容，运算结果 TRUE 送当前结果寄存器。

LT 操作数 //当前操作数≤操作数对应数据存储单元内容，运算结果 TRUE 送当前结果寄存器。

这类指令用于将他当前结果与操作数对应的数据存储单元内容进行比较，满足操作符规定的比较条件时，当前结果被置为 TRUE，反之为 FALSE。比较累指令将当前结果存储器的数据类型改变为布尔数据类型。

注意：

- 该指令直接将比较结果存放在存储单元，用户可根据存储单元的状态进行后续程序的执行。
- 比较运算类指令适用于不同数据类型的变量比较，而不局限与单一位的比较，可扩大其应用范围。

【例5.31】 比较运算类指令示例。

```
LD    rRealVar    //读取实数变量rRealVar
GT    50.0        //变量rRealVar与50做大于比较
ST    bRed        //如果大于50，则rRealVar数值超限，bRed为TRUE
STN   bGreen      //不大于等于50，则绿色bGreen置为TRUE
```

【例 5.31】中，变量 rRealVar 为过程测量值，当其值大于 50 时，表示测量值超限，红色警报等 bRed 为 TRUE，反之，bGreen 为 TRUE。

5.4.3.7 跳转与返回指令

跳转指令为 JMP，返回指令为 RET。两者的编程语言格式如下：

JMP 标号 //跳转到标号的位置继续执行

RET //返回到跳转时的断点后继续执行

跳转指令的操作数是标号，不是操作数对应的数据存储单元地址。

返回指令是没有操作数的指令，用于调用函数、功能块及程序的返回。

JMP 是 Jump 缩写。执行该指令时，如果当前结果为 TRUE，则跳转条件满足，程序在该点中断，并跳转到该标号所在的程序行继续执行。它与 RET 指令配合，用于实现子程序的执行。可以带修饰符 C 或 N，表示根据当前结果存储器内容执行或取反。

RET 是 Return 的缩写。执行该指令后，程序返回，并从原断电后第一条指令开始执行。可以带修饰符 C 或 N，表示根据当前结果存储器内容执行或取反。

注意：

- 跳转指令是从主程序跳转到子程序的指令。子程序不能用跳转指令跳转到主程序，只能用返回指令返回。子程序开始标志是标号，子程序结束标志是 RET 指令。
- 程序中标号具有唯一性。

【例5.32】 跳转指令的示例

```
LD    AUTO        //存取布尔变量AUTO
JMPC  AUTOPRO     //如果AUTO为TRUE，则程序跳转到AUTOPRO子程序
JMP   MANPRO      //如果为FALSE，则程序跳转到MANNPRO子程序
```

【例 5.32】用于自动和手动程序的切换控制。当 AUTO 开关切到自动位置，则 AUTO 为 TRUE，程序则会执行跳转指令 JMPC。因此程序跳转到 AUTOPRO 子程序，及执行在自动条件下的有关程序。当跳转条件不满足时，执行 JMP 指令，因此，程序跳转到 MANPRO 子程序，即执行手动操作时的有关程序。

在此需要特别注意的是 AUTOPRO 和 MANPRO 为子程序的标号，并非程序名。

5.4.3.8 调用指令

IEC61131-3 的标准调用指令为 CAL 指令。编程语言的格式如下。

CAL 操作数 //调用操作数表示的函数、功能块或程序

通过执行该指令，可以调用函数、功能块和程序，使程序结构简洁化，程序描述清晰。调用通用格式如下。

CAL 是 Call 的缩写，表是调用。CAL 指令的操作数是函数名或功能块实例名。实例名中的参数用逗号分隔。

5.4.3.9 圆括号指令

IEC61131-3 的标准采用圆括号对指令进行修正，即进行优先执行的操作。

左圆括号“(”用于将当前累加器的内容压入堆栈，并将操作符的操作命令存储，这时，堆栈的其他内容向下移一层。右圆括号“)”用于将堆栈最上层的内容弹出，并于当前累加器内容进行对应的操作，操作结果放在当前累加器内，此时，堆栈的内容向上移一层。因此左圆括号被称为操作延迟，它产生的瞬时结果不影响当前累加器。

表 5-11 圆括号的表达特性

序号	描述/示例	
1	圆括号表达式开始于显示操作符	AND(LD %IX0.1 OR %IX0.2)
2	圆括号表示（短格式）	AND(LD %IX0.1 OR %IX0.2)

【例5.33】 圆括号对算术运算操作的修正。

```
LD      rVar1   //将rVar1的值送入累加器
ADD(    rVar2   //rVar1压入堆栈，进行加法运算，此时累加器内容为rVar2
MUL(    rVar3   //rVar2压入堆栈，进行乘法运算，此时累加器内容为rVar3
ADD     rVar4   //将rVar3加上rVar4
)               //当前累加器的内容为rVar2*（rVar3+rVar4）
)               /将rVar2*（rVar3+rVar4）加上rVar1
ST      rVar5   //将rVar1+rVar2*（rVar3+rVar4）的结果送至rVar5
```

从【例 5.33】中，其最终实现的算法为 rVar1+rVar2*(rVar3+rVar4)，整个操作过程中，数据类型需保持一致。此外，数据类型被传递。操作从最里层的可圆括号开始，逐层向外操作，直到最外层圆括号。

```
LD      %IX1.1  //将%IX1.1的内容送至累加器
AND(    %IX1.2  //%IX1.1压入堆栈，此时累加器的内容为%IX1.2的内容
OR(     %IX1.3  //%IX1.2压入堆栈，此时累加器的内容为%IX1.3的内容
AND     %IX1.4  //%IX1.3和%IX1.4的与运算结果放入累加器
)               //堆栈原%IX1.2内容与累加器进行或运算
)               //堆栈原%IX1.1内容与累加器进行或运算
ST      bOutput //将累加器内容送至bOutput
```

【例5.34】 圆括号对逻辑运算操作的修正。

```
LD      FALSE   //将FALSE加载至累加器中
OR(     %IX0.0  //读取%IX0.0的数值
AND(    %IX0.1  //与%IX0.1进行与运算
)               //运算结果压入堆栈
OR(     %IX0.2  //读取%IX0.2的值
AND     %IX0.3  //与%IX0.3进行与运算
)               //运算结果与堆栈内容进行或运算
ST      bOutput //最终输出结果放在bOutput中
```

【例5.35】 圆括号在程序块并联中的应用。

【例 5.35】中，两个 OR 开始的指令是两个程序块，是两个触点串联的程序，最后经或运算后，将运算结果存放到 bOutput 变量中。

数学运算中，圆括号具有括号相类似的功能，即括号外的操作被延迟执行。

```
LD      rVar1   //读取变量rVar1的值至累加器
ADD     rVar2   //加rVar2，将结果rVar1+rVar2存放至累加器
MUL(    rVar3   //对rVar3的乘法被延迟，将rVar3压入堆栈
SUB     rVar4   //将rVar3-rVar4的结果放在堆栈
)               //之前的乘法运算被执行，结果放在累加器中
```

【例5.36】 圆括号的延迟功能。

【例 5.36】中，其运算结果为(rVar1+rVar2)*(rVar3-rVar4)。

通过如下的示例说明累加器和堆栈之间的关系。

```
LD      rVar1   //读取rVar1的值，并存在累加器中
ADD(    rVar2   //延迟加，将rVar2压入堆栈
MUL(    rVar3   //延迟乘，将rVar3压入堆栈
SUB     rVar4   //rVar3-rVar4运算，并存回堆栈
)               //堆栈内数据rVar2弹出，乘运算被执行
)               //当前累加器中的数据与堆栈内数据rVar1弹出，执行加法运算
```

【例5.37】 累加器和堆栈之间的关系。

【例 5.37】中堆栈内数据和当前累加器数据内容如表 5-12 所示。

表 5-12 堆栈内数据和当前累加器数据内容的变化

指令	1	2	3	4	5	6
当前累加器	rVar1	rVar1	rVar1	rVar1	rVar1	rVar1+ rVar2*(rVar3- rVar4)
堆栈 1	-	rVar2	rVar2	rVar2	rVar2*(rVar3-rVar4)	-
堆栈 2	-	-	rVar3	rVar3- rVar4	-	-

最终，【例 5.37】的运算结果为 rVar1+rVar2*(rVar3-rVar4)。

所以对较复杂运算关系使用结构化文本或梯形图语言实现较简单。从圆括号指令内进行跳转有时会产生不可预测的结果，故在使用时需要谨慎应对。

5.4.4 函数及功能块

5.4.4.1 函数的调用

指令表语言中，函数的调用相对简单。

函数调用方法

在操作符区输入函数名，第一个输入参数作为 LD 的操作数。如果有更多的参数，下一个必须是在同一行作为函数名输入，之后的参数可以添加到这一行并用逗号隔开或添加到下面的行里。函数返回值将被保存在累加器中。需要注意的是，根据 IEC 标准的规定，返回值只能有一个。函数调用的编程格式和示例如表 5-13 所示。

表 5-13 函数调用的编程格式及示例

方式	编程格式		示例	
单参数	LD	参数	LD	0.5 //读取弧度 0.5
	函数名		COS	//调用 COS 函数
	ST	返回值	ST	Var1 //将运算结果 0.87758 存放在变量 Var1 中
双参数	LD	参数 1	LD	Var1 //读取变量 Var1 的值
	函数名	参数 2	ADD	Var2 //与变量 Var2 的值相加
	ST	返回值	ST	Var3 //将运算结果返回值存放在 Var3
多参数	LD	参数 1	LD	Var1 //读取变量 Var1 的值
	函数名	参数 2,…,参数 n	SEL	IN0，IN1//根据 Var1 的值选择 IN0 或 IN1 作为返回值
	ST	返回值	ST	Var2 //将运算结果返回值存放在 Var2

带非形参表的函数调用其格式如下。

函数名非形参，非形参，…，非形参，

带形参表的函数调用，编程语言的格式如下。

函数名（第一形参 := 实参，…，最后形参 := 实参）

函数调用示例

下面通过 2 个调用函数的示例让读者有更清晰地理解。

```
LD      10      //将10送当前累加器
ADD     10,
        12,
        14      //将10，12，14与当前结果存储器内容1相加，结果36送当前累.
ST      iVar1   //当前累加器36送变量iVar
```

【例5.38】 非形参的函数调用示例。

【例 5.38】中，用 ADD 函数直接实现多个数值的想加运算。因此，与传统 PLC 的相加运算比较，程序被简化。须注意，一些传统 PLC 产品只允许有一个操作数。例如，ADD10。而 CoDeSys，可以直接进行叠加。

```
LD      strVar1     //将strVar1的值送至当前累加器
Right(  strVar1     //右移指令放入堆栈
LEN                 //取变量strVar1的长度
SUB     1           //将strVar1的长度-1
)                   //将 (strVar1) 右移一位
ST      strVar1     //将当前累加器的数值输出至strVar1
```

【例5.39】 带形参的函数调用示例。

调用 RIGHT 函数，该函数第一个参数为 LEN，取的是变量 strVar1 的数据长度，第二个参数是 sub1，将当前累加器中的数据长度-1，在函数中的功能为右移的位数，即每次移 1 位。上述程序如果转换为结构化文本其程序为：

```
strVar1 := RIGHT(strVar1, (LEN(strVar1) - 1));
```

5.4.4.2 功能块的调用

功能块及程序调用方法

非形参编程语言中，功能块调用有下列 2 种方式。

1、 带形参表的功能块调用，编程语言的格式如下。

CAL 功能块实例名（形参表）

2、 带参数读/存储的功能块调用，编程语言的格式如下。

CAL 功能块实例名

功能块及程序调用示例

下面以调用 TON 功能块为例说明功能块的调用方法。

【例5.40】 带参数读/存储的功能块调用，程序如下。

```
LD      %IX1.1          //读取%IX1.1的数据
ST      fb_Timer1.IN    //赋值给功能块TON实例fb_Timer1的IN变量
LD      T#500ms         //读取时间t#500ms
ST      fb_Timer1.PT    //赋值给功能块TON实例fb_Timer1的PT变量
CAL     fb_Timer1       //调用功能块fb_Timer1
```

5.4.5 应用举例

【例5.41】 称重显示示例。

在实际工业生产中很多设备拥有称重筛选设备，当实际产品的重量不符合设定值时，需要将该产品作为次品并触发剔除信号将该产品剔除，这种设备中就有称重相关程序。

控制要求

称重装置将物料称后的毛重数据存储在 PLC 寄存器内，使用称重函数将毛重减去皮重，最终将净重以 REAL 类型变量输出。

假设：毛重变量：rGrossWeight；皮重变量：rTareWeigh；净重变量：rActuallyWeight。

为了控制称重信号的执行，需要设置手动触发信号作为称重命令，使用布尔变量 bStart 作为启动命令。

数据的显示类型毛重 rGrossWeight，皮重 rTareWeight 及净重 rActuallyWeight 均为 REAL 数据类型。

编程

编写功能块 FB_Weight，如下部分为功能块声明区域。

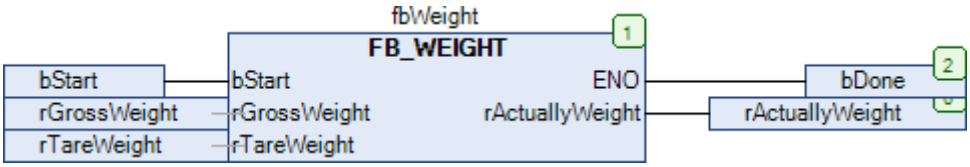
```
FUNCTION_BLOCK FB_WEIGHT
VAR_INPUT
    bStart: BOOL;
    rGrossWeight:REAL;
    rTareWeight:REAL;
```

```
END_VAR VAR_OUTPUT
    ENO:BOOL;
    rActuallyWeight:REAL;
END_VAR
VAR
END_VAR
```

如下部分为功能块的逻辑程序部分：

```
LD      bStart          //加载启动信号
ST      ENO             //将当前运行状态输出
JMPC    WEIGHTING       //如有启动信号程序跳转至WEIGHTI...
LD      0
ST      rActuallyWeight //如没有启动信号时，将输出数值清...
RET

WEIGHTING:
LD      rGrossWeight    //称重程序段，装载毛重重量
SUB     rTareWeight     //将毛重减去皮重重量
ST      rActuallyWeight //将最终的结果输出值净重
```

当功能块部分的程序编写完成后，实际测试，新添加一个程序，在程序中调用功能块 FB_Weight 并填写相应的输入输出参数，最终的程序如下图所示。



称重程序示例

例如，当毛重的重量为 5（g）时，皮重设定的值为 1（g），只有当 bStart 被触发变为 TRUE 时，最终的净重才会有出 4（g），否则一直为 0。

【例5.42】 循环计算的示例。

控制要求

计算 1 到 10 的累加和及阶乘的程序，程序中可以使用 JMPC 跳转指令。

编程

变量声明如下：

```
PROGRAM PLC_PRG
VAR
    diSum,diProduct:DINT;
    i:BYTE;
END_VAR
```

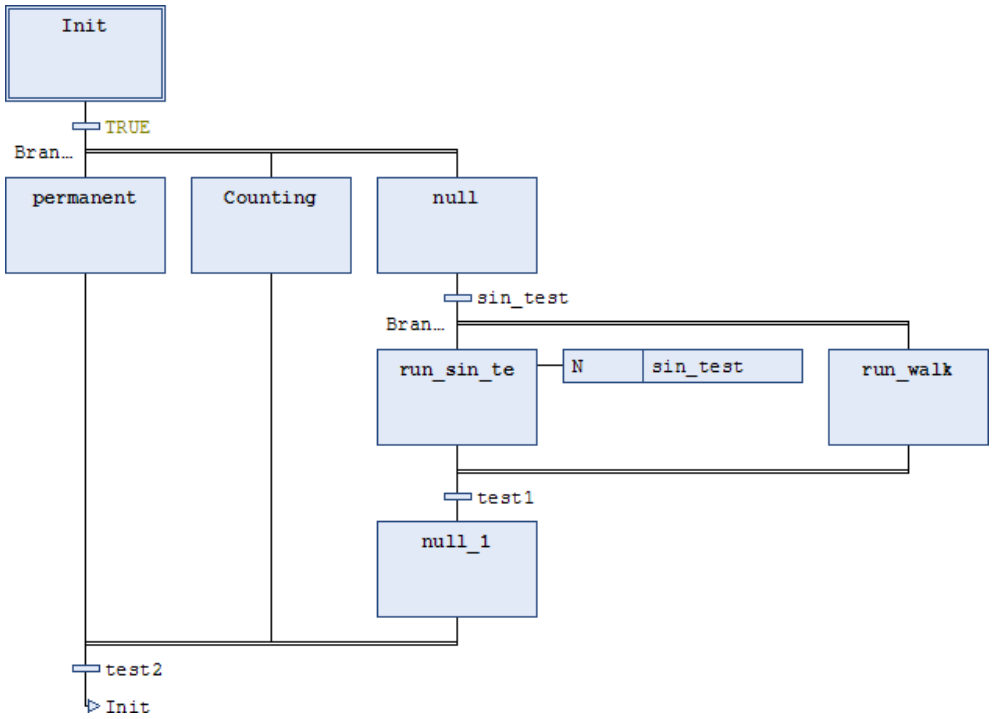
```
LD      1       //读取双整形数据1作为计数值i和乘积值diProduct的初始值
ST      i       //将i置为1
ST      diProduct   //将乘积值diProduct置为1
LD      0       //读取双整数据0，并作为累加和diSum的初始值
ST      diSum   //将diSum置为0

LOOP:
LD      diSum   //循环开始，读取diSum的当前值
ADD     i       //将i与diSum进行加法运算
ST      diSum   //将累加和返回给变量diSum
LD      diProduct   //进行乘积的循环，读取diProduct的当前数值
MUL     i       //乘以i，进行阶乘运算
ST      diProduct   //将结果返回给变量diProduct变量中
LD      i       //读取i
ADD     1       //将变量i进行递加运算
ST      i       //将加完的的结果返回给变量i
LE      10      //如果当前i的累加值小于10，则当前值置1
JMPC    LOOP    //如果值为1，则进行循环，跳转至LOOP处。
RET             //返回
```

上述程序可以简单的进行累加及阶乘运算，运算结果在 diSum 中为 55，在变量 diProduct 中存放 3628800。需要注意的是，当运算结果大于变量设置的数据类型允许范围时，结果会被置为 0。例如，如果计算 1 到 50 的累加和及阶乘时，阶乘的结果超过了双整长整数的允许范围，此时，计算结果被置为 0，为此，可将变量的数据类型改为实数或双精度实数类型。

5.5 顺序功能流程图（SFC）

顺序功能流程图语言是为了满足顺序逻辑控制而设计的编程语言。编程时将顺序流程动作的过程分成步和转换条件，根据转移条件对控制系统的功能流程顺序进行分配，一步一步的按照顺序动作，如图所示。每一步则代表一个控制功能任务，用方框表示。在方框内含有用于完成相应控制功能任务的梯形图逻辑。这种编程语言使程序结构清晰，易于阅读及维护，可以大大减轻编程的工作量，缩短编程和调试时间。用于系统的规模校大，程序关系较复杂的场合。它的特点是：以功能为主线，按照功能流程的顺序分配，条理清楚，便于对用户程序理解。

图 5-42 顺序功能流程图语言



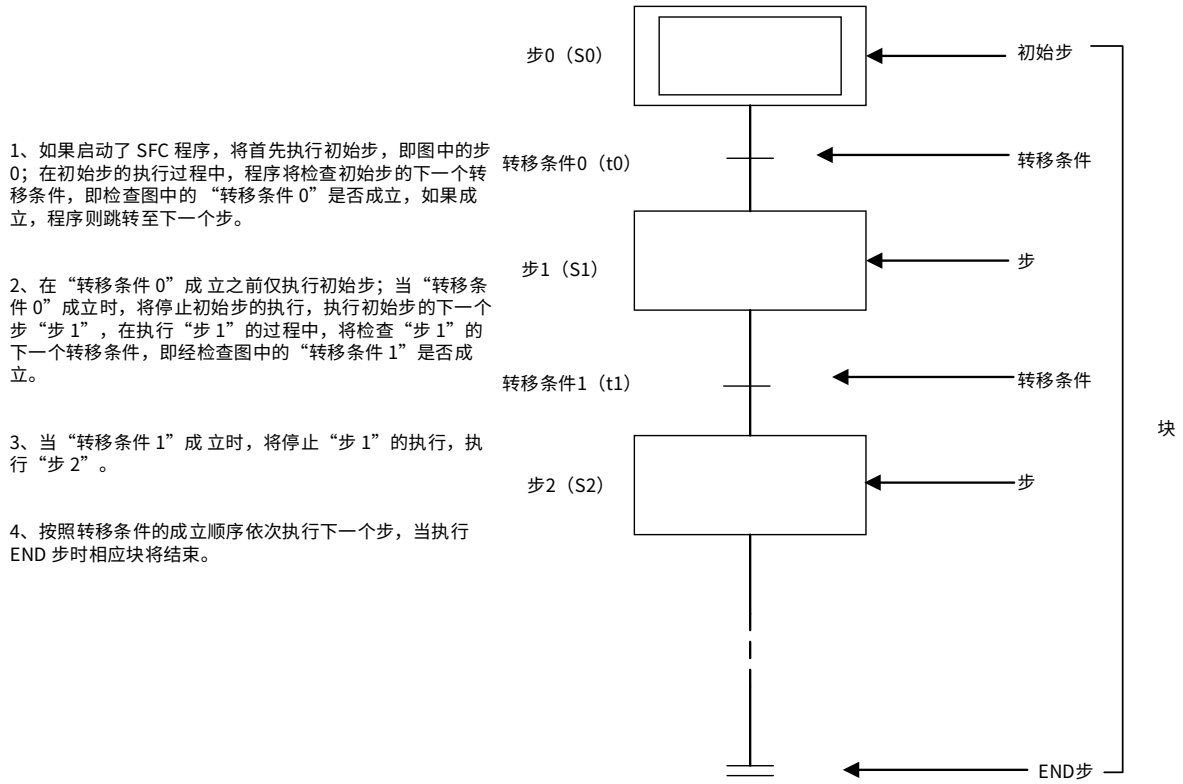
5.5.1 顺序流程图编程语言简介

基本结构

在 SFC 程序中，从初始步开始，当转移条件成立时按顺序执行转移条件的下一个步，通过 END 步结束一系列的动

作，完整的过程如图 5-43 所示。

图 5-43 SFC 基本流程



程序特点

1、 SFC 优点

- 执行顺控的最佳选择

将自动运行的顺序就照原样转换成图形描述即可，因此便于编程，便于理解程序。

- 易于理解的结构化程序

可以采用图形进行层次化和模块化的编写工作，所以便于试运行以及维护工作。如图所示，图的左边是设备运行的流程图，通过 SFC 编程语言，能够直接将左边的流程图转换为程序，编程人员只需要在每个动作中加入相应的了逻辑，在流程跳转时加入适当的转换条件即可。

图 5-44 设备运行流程图

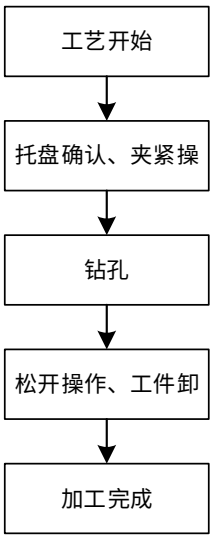
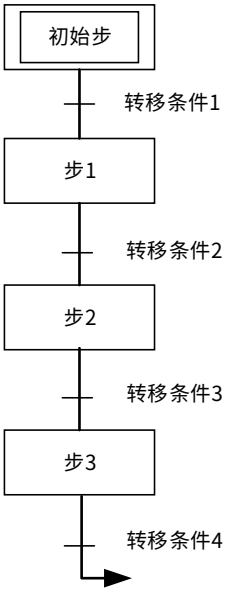


图 5-45 SFC 图



- 工序（步）间不需要互锁

由于 CPU 只对动作中的步进行运算，所以前进动作和后退动作逻辑不需要互锁。而使用 SFC 顺序流程图编程时并不需考虑互锁，因为在前后步条件不满足的情况下，程序并不会执行其他步，故不需要在触点互锁这方面考虑太多。

- 在多个工序（步）中可使用同一线圈

由于 CPU 只对动作中的步进行运算，所以即使在没有动作的步中存在相同的线圈，也不会被作为双线圈处理。（与主顺控程序中的线圈相同时，会变为双线圈处理。）

- 用图形监控动作状态

机械设备因为故障而停机时，通过监控，可以在画面上显示当前停止中的步，从而在第一时间查找出停机的原因，便于排除故障。此外，如果在各步上附有注释，那么动作时为什么会停止的原因就一目了然了。

- 设计的标准化

按照控制流程以图形方式编程，所以无论是谁编写的，都会是几乎相同的没有个人差别的程序，从而达到设计图的标准化。

- 由多人分担程序编写

可以将控制内容分成多个，由不同的人编写程序，然后编辑成一个。

- 按工序进行运算处理

由于 CPU 只对动作中的步进行运算，所有通过好的编程方法可以缩短扫描时间。

- 更容易设计和维护系统

因为整个系统和各个站以及机器本身的控制都是在一对一的基础上与 SFC 程序和步对应，所以即使顺控程序经验较少的人也可以设计和维护系统。此外其它程序员也使该格式设计的程序，相比其他编程语言，可读性更强。

- 此外，有效地使用 SFC 具备的功能，可以缩短机械动作的节拍时间

2、 SFC 缺点

- 不适用的控制内容

采用中断处理方式的紧急停止、一直监控、从计算机接收数据等程序都不是顺序控制，所以不适合编写 SFC 程序。(对这些内容的控制如果在主程序中编写梯形图程序，则便于汇总掌握。)

- 不习惯导致的不放心

因为不习惯，所以有不能在极其复杂的控制中使用的先入为主的成见。（通过 SFC 进行结构化和模块化，可以整理要控制的内容，所以可以只用梯形图编程）。

5.5.2 SFC 的结构

在 SFC 编程语言中的“工具箱”中可以添加 SFC 中的工具，SFC 由表 5-14 中的 6 大部分组成。其中，SFC 的基本元素是步配合转换条件，整合各基本元素形成了几种基本结构，任何一个复杂的或简单的 SFC 结构都是以这些基本部件构成的，如图 5-46 所示。

图 5-46 基本组成结构

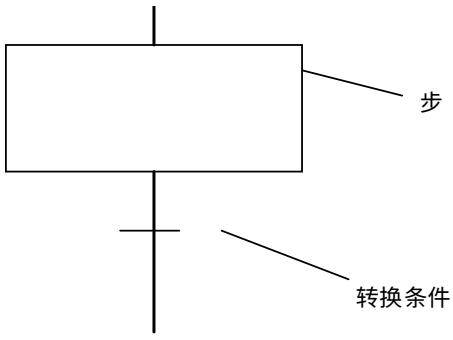


表 5-14 SFC 工具箱

类型	图形符号	说明
步		SFC 包含一系列的步，这些步通过有向的连接彼此联系
转移		动作是使用其他语言实现的一系列指令，可以用 IL 或 ST 语言实现的指令语句，也可以是用 LD、FBD 或 SFC 实现的网络
动作		动作命令可以在步中加入入口动作和出口动作
跳转		步之间的切换就是转换。只有当步的转换条件为真时，步的转换才进行
宏		添加宏
分支		添加并行分支

5.5.2.1 步

步的定义

步表示整个工业过程中的某个主要功能。它可以是特定的时间、特定的阶段或者是几个设备执行的动作。步属于 SFC 的执行体，所有的实现执行的逻辑代码都包含在其中，转换条件则是决定步的状态，当上一步的转换条件成立，则这个步就被激活，被激活的步将进入执行的状态。

被激活期间，这个步被反复扫描，直到这个步的转换条件成立，步的激活被解除，退出执行到下一个步，下一个步则被激活。

SFC 顺序功能图中的步由一个方框表示，该方框内表示“步名称”以及由连接线表示的上下转于关系。步的名称可以在当前位置直接编辑，并且步的名称必须在其所在的 POU 内是独一无二的，尤其是在 SFC 内使用动作编程时，需要特别注意。

步的组态

步分为两种：初始步和普通步。下文会对这两种不同类型的步做逐一介绍。

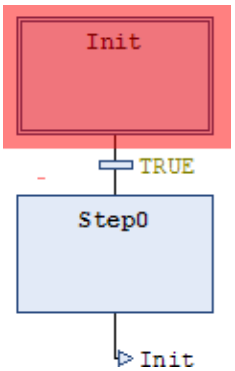
1、 初始步

初始步是表示各个块的开始的步。可以选中对应的“步”，单击鼠标右键，选择“初始步”或通过快捷菜单中的“ ”按钮来进行初始步的设定。如图 5-47 中的 a）所示，初始步的视图和普通步略有不同，初始步为双矩形方框（被一个双边线包围）的视图，可以通过鼠标右键进行设定，如图 5-47 的 b）所示。

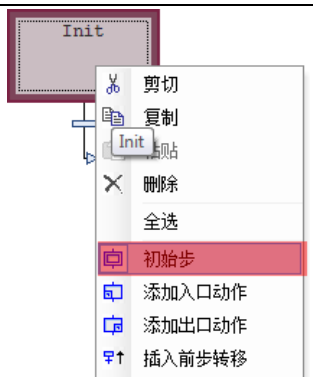
表 5-15 步的编辑

类型	图形符号	说明
初始步		用于将当前 SFC 编辑器中选中的步，设置为初始步

图 5-47 设定初始步



a)初始步视图



b)设定初始步

2、 普通步

当前正在被执行的“普通步”称为“激活步”。在线模式下，“激活步”被填充成蓝色。

每步包括一个动作和一个标记，这个标记用来表示此步是否激活。如果单步动作正在执行，那么所在的步就会显示为蓝色的框，如下图所示。

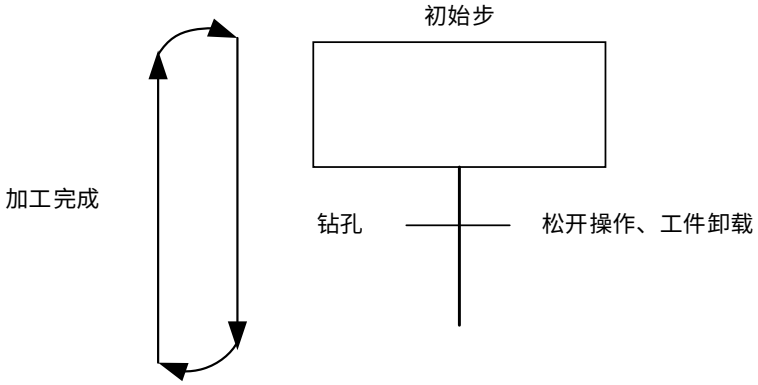
普通步在一个控制循环中激活步的所有动作都将执行。所以，当激活步之后的转换条件是 TRUE 时，它之后的步被激活。当前激活的步将在下个循环中再执行。

5.5.2.2 动作

动作的定义

在本节的开始就介绍到，SFC 的执行过程最基本的结构就是步配合转换条件，每当步被激活时，将执行步的操作，直到转换条件成立才转至下一步。下一步被激活开始新的执行动作，直到自己的转换条件成立才离开，如此顺序往下执行，如下图所示。

图 5-48 SFC 执行步骤



动作是步具体要执行的操作，例如阀门的开启、电动机的起动或停止，工作或产品的移动等。每个步里面，可以是多个动作在执行，转换条件也是执行判定，所以，建立好 SFC 运行的流程结构，其中非常重要的一部就是确定好步和对其动作的组态，。

此外，将产生如下的标签：

- 每当创建一个步，会自动分配一个步的结构标签。
- 每当创建一个动作，会自动分配一个动作的结构标签。
- 每当创建一个转换条件，会自动分配一个 BOOL 量的标签。这些标签的数据可以在编程中引用。

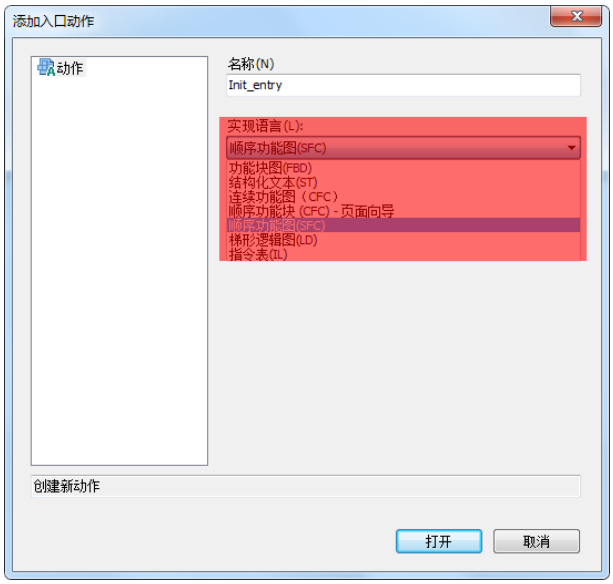
每个步都可以定义多个（或单个）动作，动作包括了此步执行的详细描述，动作可以使用梯形图、FBD、ST、SFC 等语言编写，用户可以对入口及出口动作进行编辑，其编辑动作的元素如表 5-16 所示。

表 5-16 动作

类型	图形符号	说明
添加入口动作		步激活前执行的动作
添加出口动作		这个动作会在步执行完的下一周期执行（“步出口”）

选择添加动作后，系统会自动弹出提示框，如图 5-49 所示，用户可以选择想要的编程语言来编写动作程序。

图 5-49 步动作支持的编程语言



限定符

限定符是用来配置一个动作将以什么样的方式与 IEC 步相关联的。限定符是被插入到一个动作元素的限定符区域内，这些限定符有 IecSfc.library 的 SFC Action Control 功能块来进行处理的，并通过 SFC 插件 IecSfc.library 可以被自动的包含到一个工程中去，SFC 限定符如表 5-17 所示。

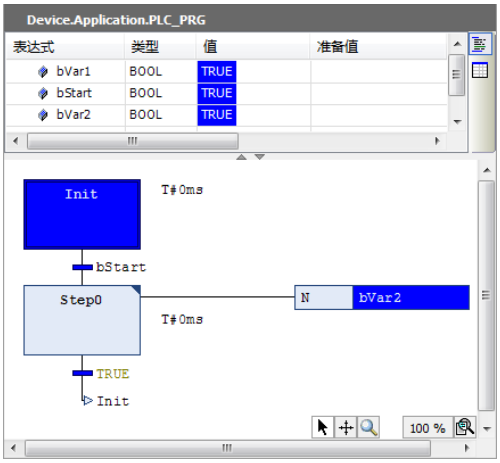
表 5-17 限定符

限定符	名称	说明
N	Non-stored	只要步激活动作就激活
S0	Set (Stored)	当步激活时动作启动，步失效时仍继续，直到动作重启
R0	Overriding Reset	动作失效
L	Time Limited	当步激活时动作启动，动作会继续直到步不活跃或设定时间已到
D	Time Delayed	激活后延时器启动。延时后如果步仍激活，动作开始直到步失效
P	Pulse	当步激活/失效时动作开始执行，且值执行一遍
SD	Stored and time Delayed	延时后动作开始，它会继续直到重启
DS	Delayed and Stored	具体设定的时间延时之后如果步仍激活，动作开始并继续，直到重启
SL	Stored and time limited	步激活后动作开始，并继续一定时间或重启

当使用限定符 L、D、SD、DS 或 SL 时，需要一个时间值，格式为 TIME 的类型。

【例5.43】 限定符 N 的应用举例。

图 5-50 动作限定符 N



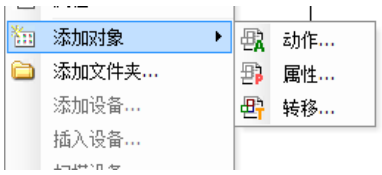
如表 5-17 中的说明，限定符“N”的作用是：只要对应的步被激活，则相应的连接的变量也被激活。如图中的 bVar2，每次 Step0 被执行，bVar2 就会被置位 ON，反之为 OFF，该限定符可以用于监控步的执行状态。

限定符 L、D、SD、DS 和 SL 需要一个时间常量格式的时间值。格式为 T#（数值）（单位）。如 5 秒的表示为：T#5S。

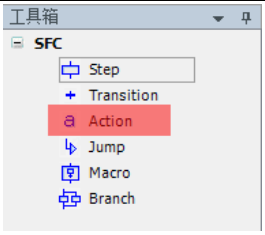
动作的组态

用户可以在设备树中找到该 SFC 编程语言的 POU，鼠标右键选择“添加对象”，可以选择动作，如图 5-51 中的 a），或者可以直接使用工具栏，在工具栏中选择“ Action ”按钮进行动作的添加，工具栏如图 5-51 的 b）所示。

图 5-51 添加“动作”1



a）设备树中添加动作



b）工具箱中添加动作

使用第二种方式，在工具栏中选择“Action”，将其拖拽至步的上方后，会显示如图 5-52 中的四个灰色方框，可以将动作拖拽至对应的方框中，拖拽后会在步对应的“步属性”中也添加相应的设置，如图 5-52 右所示。

图 5-52 中的‘1’为 IEC 标准步的动作，CoDeSys 控制平台扩展了 IEC 标准的动作，额外加入了“步入口”，“步出口”和“步活动”三个动作，三种扩展动作分别为‘2’，‘3’，‘4’。

图 5-52 添加“动作”2

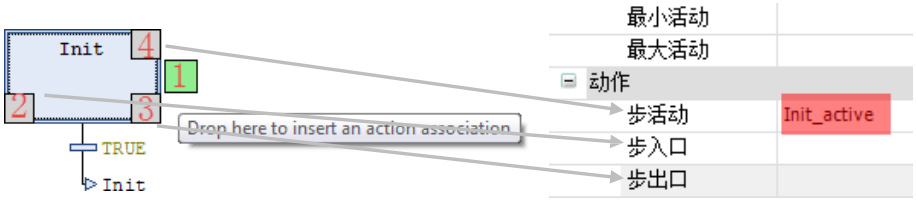
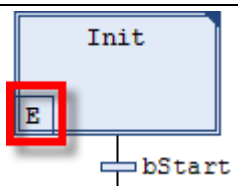


图 5-52 中所示的‘1’，‘2’，‘3’，‘4’对应具体的步动作解释如下。

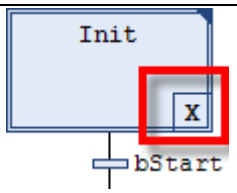
1、 步入口

该步被激活前执行的动作。只要该步被程序激活，这些步动作就会被执行，且在“步活动”动作之前。该动作通过步属性的“步入口”区域的入口关联到步。它通过步左下角的“E”显示，如图 5-53的a）所示。

图 5-53“步入口”及“步出口”状态



a）步入口”状态



b）“步出口”状态

2、 步出口

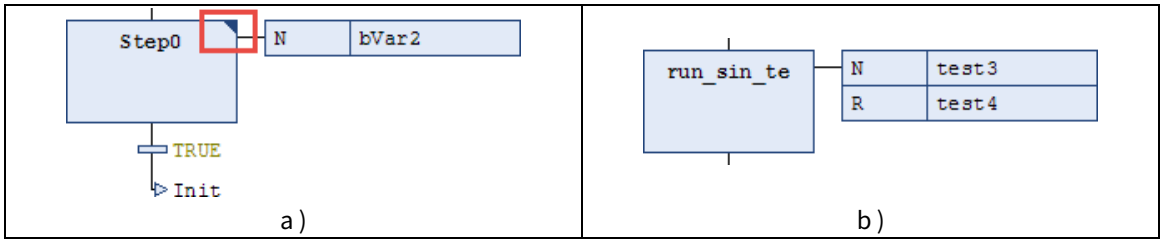
这个动作会在步执行完的下一周期执行。当步无效时它会被执行一遍。但是，执行不会在同一个周期，在下一步执行周期的开始。动作通过步属性的“步入口”区域的入口关联到步。它通过步右下角的“X”显示，如图 5-53的b）所示。

3、 步动作

步激活时，步动作会被执行，可能的入口已经执行完。在这一步的“步入口”执行完之后，这种步动作会在步激活后被执行。但是，不同于IEC步动作，当它无效时不会被再执行，且它不可分配资格。

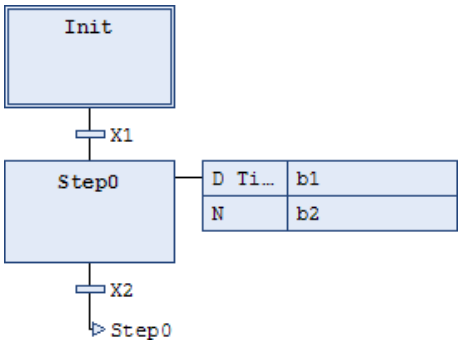
动作通过步属性的“步活动”区域的入口关联到步。它通过步右上角的小三角显示，如图 5-54的a）所示。

图 5-54 “步活动”状态




```
PROGRAM PLC_PRG
VAR
    b1,b2,b3: BOOL;
    X1, X2: BOOL;
    Time1:TIME:=T#5S;
END_VAR
```

图 5-55“步激活”状态



程序如图所示，当转移条件变量X1为TRUE后，程序则会执行步Step0，与此同时，对应的步激活状态变量b2则为TRUE，由于b1对应的限定符为D，而具体的时间在程序的声明区定义的是变量Time，即为5s，则当Step0执行5s后，b1变量由FALSE置为TRUE。

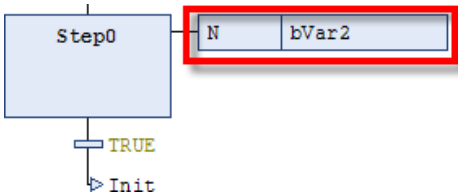
4、 步关联动作

步关联动作有前插入和后插入，如表 5-19所示。使用当前步成为IEC标准步时，先鼠标点击步，例如Step0，然后选择菜单栏“SFC”→“插入前关联动作”命令，将IEC步动作与步进行关联，一个步可以关联一个或多个动作。新动作的位置由当前光标位置及使用的命令决定，动作必须在工程内是可用的，并且被插入时必须采用一个唯一的动作名称，如图 5-56所示。

表 5-18 步关联动作

类型	图形符号	说明
插入前关联动作		在步中添加前关联动作
插入后关联动作		在步中添加后关联动作

图 5-56 IEC 标准步动作添加关联动作



IEC的标准步至少被执行两次，第一次执行是在当他们被激活时，第二次执行是在下个周期它们被禁止时。

由于一个步中可以分配多个动作，这些动作按照从上到下的顺序进行执行。例如，动作Action_AS1关联到步AS1，

添加一个步动作和添加一个有限定符N的IEC动作，在这两种情况下，假设转换条件都已经满足，初始步再次到达需要花费2个循环周期的时间。假设，一个变量iCounter在Action_AS1中增加。再次激活步Init之后，步动作例子中的iCounter值为1。拥有限定符N的IEC动作的iCounter的值却是2。

5.5.2.3 转移

在步和步之间的转换条件简称为转换。转换条件的值必须是TRUE或FALSE，因而它可以是一个布尔变量、布尔地址或布尔常量。只有当步的转换条件为真时，步的转换才能进行。即前步的动作执行完后，如果有出口动作则执行一次出口动作，后步如果有入口动作则执行一次后步入口的动作，然后按照控制周期执行该活动步的所有动作。

用顺序功能图编写的程序组织单元包含了一系列的步，步之间是通过定向连接（转换条件）实现的。在顺序功能图中和步转移相关的操作如表 5-19所示。

表 5-19 SFC 中的转移操作

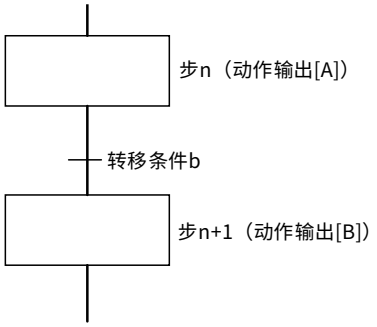
类型	图形符号	说明
插入前步转移		在步之前插入转移条件
插入后步转移		在步之后插入转移条件

通常而言，转换拥有不同的方式，如下是在顺序功能图中常会用到的几种转移方式，下文中会逐一介绍。

1、 串行转移

串行转移是指，通过转移条件成立转移至串行连接的下一步执行处理的方法。

图 5-57 串行分支转移



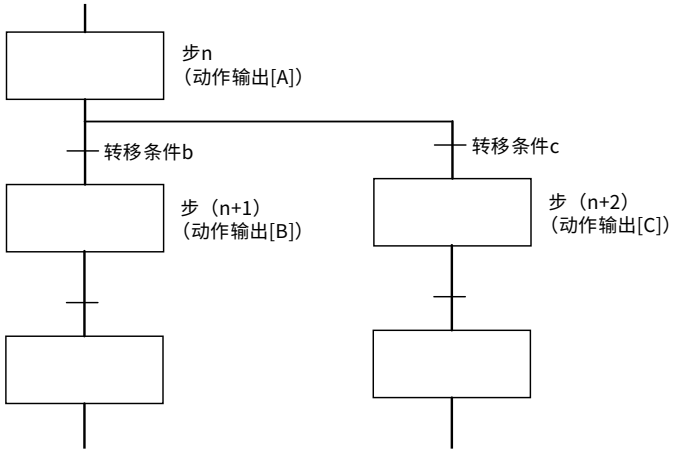
如图所示，执行步 n 的动作输出[A]时，如果转移条件 b 成立，则对动作输出[A]进行非执行处理，执行步(n＋1)的动作输出[B]。

2、 选择转移

选择转移是指，在并行连接的多个步中，仅执行最先成立的转移条件的步的方式。

A、 选择分支转移

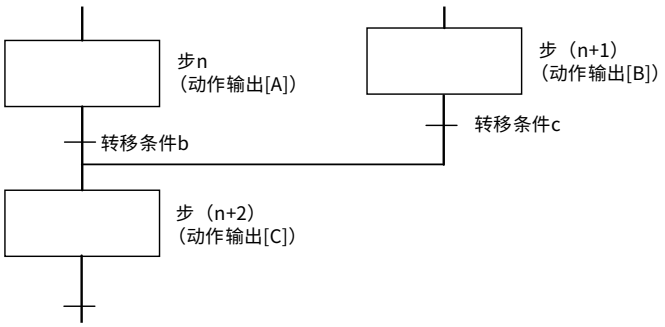
图 5-58 选择分支转移



- 执行步 n 的动作输出[A]时，选择转移条件 b 或者 c 中最先成立的条件的步(步(n+1)或者步(n+2))，执行该步的动作输出([B]或者[C])。
- 转移条件同时成立时，左侧的转移条件优先。对步 n 的动作输出[A]进行非执行处理。
- 选择后，依次执行所选列的各个步，直至进行合并为止。

B、 选择合并转移

图 5-59 选择合并转移



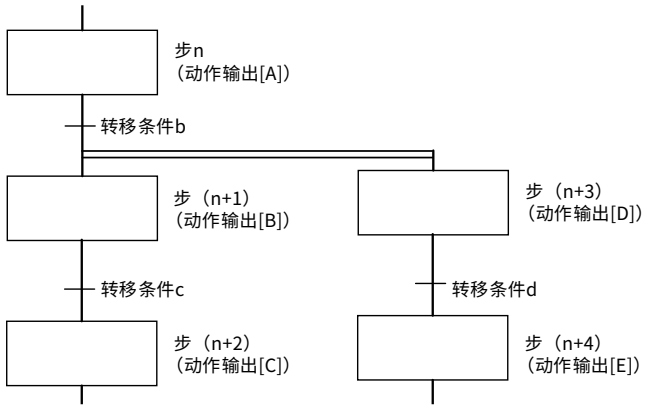
如果分支中执行列的转移条件（b 或者 c）成立，则对执行步的动作输出（[A]或者[B]）进行非执行处理，执行步(n+2)的动作输出[C]。

3、 并行转移

并行转移是指，通过转移条件成立同时执行并行连接的多个步的方式。

A、 并行分支转移

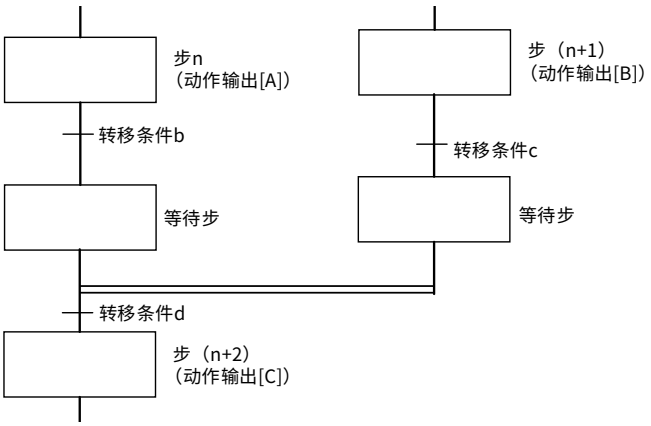
图 5-60 并行分支转移



- 执行步 n 的动作输出[A]时，如果转移条件 b 成立，则同时执行步(n+1)的动作输出[B]、步(n+3)的动作输出[D]。
- 转移条件 c 成立时转移至步(n+2)，转移条件 d 成立时转移至步(n+4)。

B、 并行合并转移

图 5-61 并行合并转移 1



- 执行步 n 的动作输出[A]、步(n+1)的动作输出[B]时，如果转移条件 b、转移条件 c 成立，则分别对步 n 的动作输出[A]、步(n+1)的动作输出[B]进行非执行处理，转移至等待步。
- 等待步是用于使并行处理的步同步的步，通过将并行处理的所有的步转移至等待步，对转移条件 d 进行检查，如果转移条件 d 成立，则执行步(n+2)的动作输出[C]。
- 等待步被作为虚拟步，即使没有动作输出梯形图也没有关系。

5.5.2.4 跳转

跳转是指通过转移条件成立转移至同一个 POU 内的指定步的执行处理方式。它是通过一条竖线和水平箭头及跳转目标名显示的。如 Step0 。

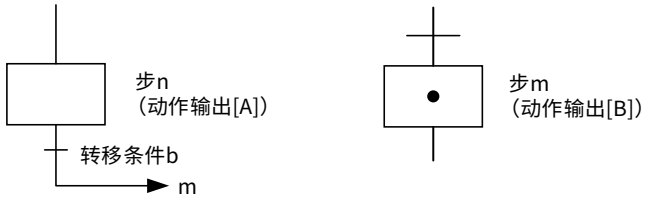
跳转定义了后续转换为 TRUE 时将要执行的步。因为按照程序的执行顺序，程序不能交叉或往上执行，故在此就有跳转存在的必要了。跳转只能用在分支的最后。当最后一个转换被选中时，可以通过“插入前跳转”命令插入，跳转的可执行的操作如图所示。

表 5-20 SFC 中的跳转元素

类型	图形符号	说明
插入前跳转		在步前添加跳转。
插入后跳转		在步后添加跳转。

跳转的目标可以通过关联的文本字符串给定，可内引编辑。它可以是一个步名或平行分支的标签，如图所示。

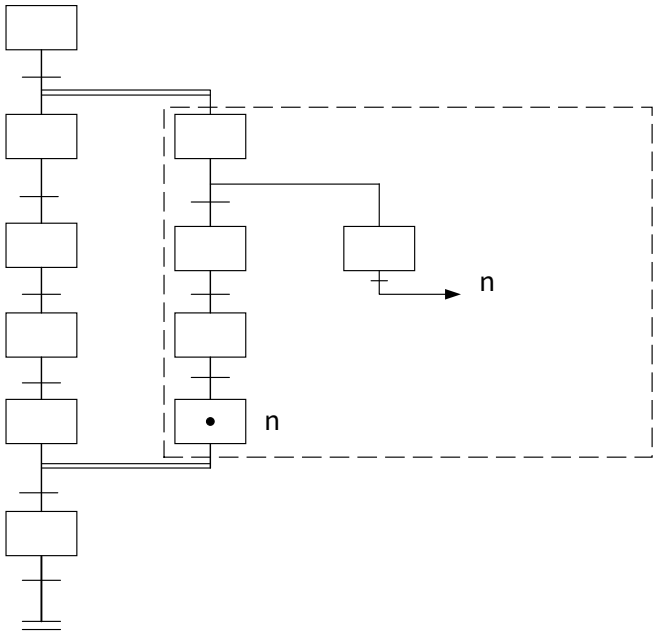
图 5-62 并行合并转移 2



执行步 n 的动作输出[A]时，如果转移条件 b 成立，则对动作输出[A]进行非执行处理，执行步 m 的动作输出[B]。

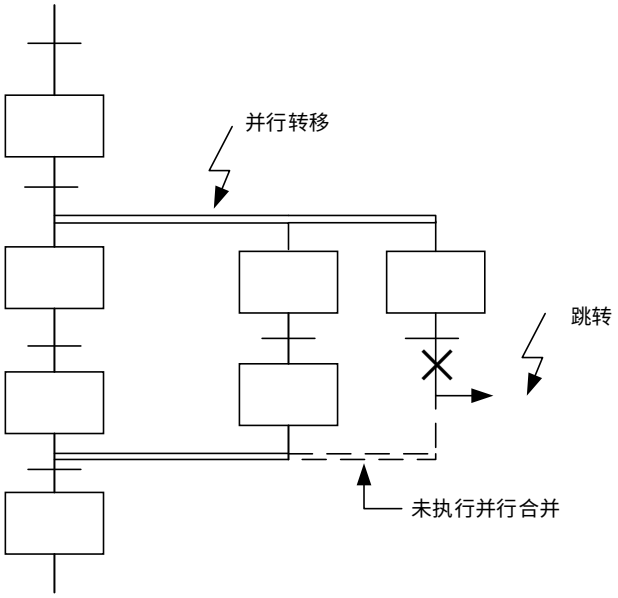
在并行转移内进行跳转时，只能在分支的各个纵方向进行跳转。例如，从分支起至合并为止的纵方向内的跳转程序，如图 5-63 所示。

图 5-63 并行合并转移 3



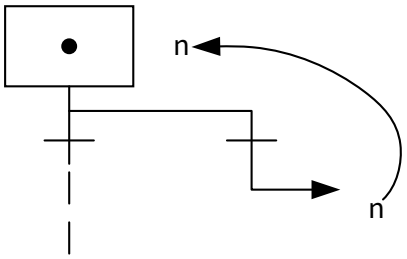
不能创建如图 5-64所示的跳转程序：向分支内的其它纵向梯形图的跳转，向并行分支外部的跳及从并行分支外部向并行分支内的跳转。例如，向并行分支外部的跳转程序（不能指定）。

图 5-64 并行合并转移 4



例如，如图 5-65所示的转移条件成立时，不要指定至当前步的跳转。如果指定了至当前步的跳转，则无法正常动作。

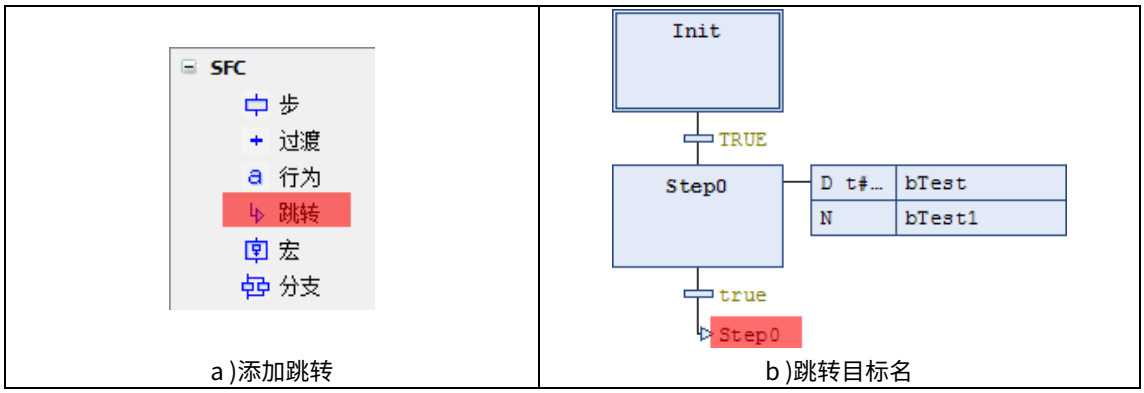
图 5-65 并行合并转移 5



跳转的建立

在工具箱中找到“跳转”即可实现跳转的插入，如图 5-66的a）所示，添加后，只需要输入跳转目标名即可；如图5-66的b）所示，跳转的目标是Step0，就直接写入Step0即可。

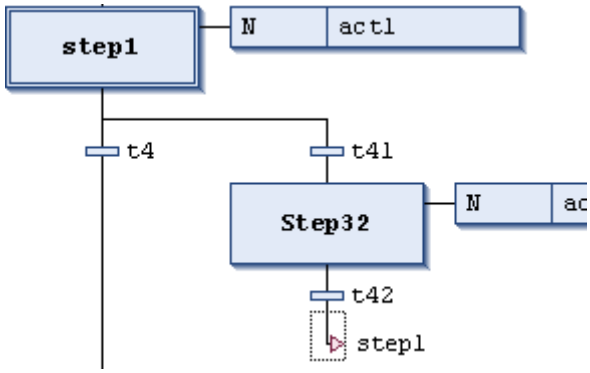
图 5-66 跳转的建立



a)添加跳转　　b)跳转目标名

图 5-67 所示的是跳转指令的一个典型应用，当跳转指令 t42 条件满足后，根据程序指示，则会自动跳转至step1，并重新进入程序循环。

图 5-67 SFC 跳转指令典型应用



5.5.2.5 宏

同其他软件对宏的定义一样，SFC 程序中的宏主要功能也是为了避免很多重复性的工作，用户只需提前将宏事先定义好，随后在程序中对其调用即可。对宏常见的操作如表 5-21 所示。

表 5-21 SFC 中的宏元素

类型	图形符号	说明
插入宏		插入宏
添加宏		添加宏
进入宏		即打开宏编辑器视图
退出宏		可以返回到 SFC 标准视图

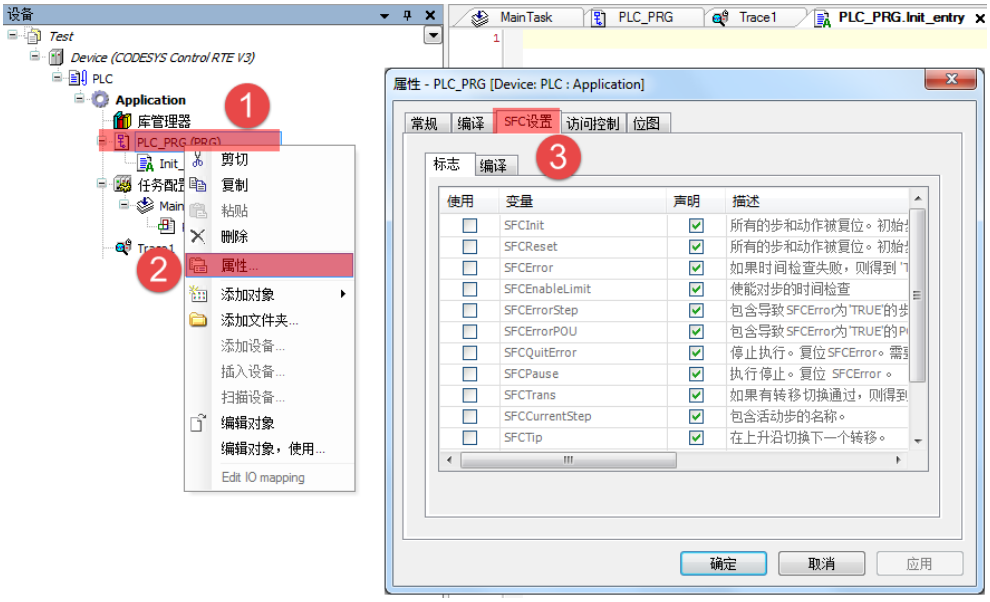
隐形变量

每个 SFC 步和 IEC 动作提供隐含生成变量供运行时监视步和 IEC 动作。还可以定义变量来监视和控制 SFC 执行（超时，重启，尖模式）。这些隐含变量的类型实在库 IecSFC.library 中定义的。

SFC 对象添加时这个库会自动被添加。

在 SFC 编程语言里有些隐形变量可以外部调用。正常情况下这些变量不显示出来。要使用这些变量需要对 SFC 属性做设置。右击 SFC 语言的 POU 的“属性”，弹出属性对话框中点击“SFC 设置”选项，将需要使用变量前打勾。如图 5-68 所示。

图 5-68 SFC 隐形属性



为了访问这些标识使它们工作，它们必须声明和激活。这在“SFC 设置”对话框中设置。它是对象属性对话框的子对话框。如需使用该变量，则必须在变量前的“使用”前打上勾，该变量在其描述中也说明了其具体的使用方法。

5.6 连续功能图（CFC）

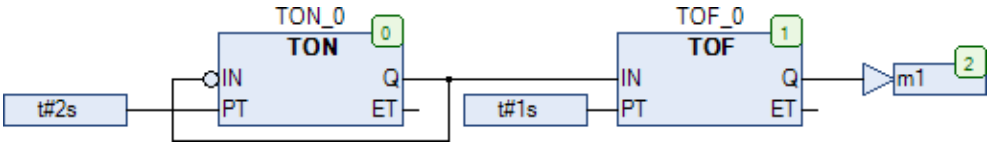
5.6.1 连续功能图编程语言结构

5.6.1.1 简介

连续功能图（CFC）实际上是 FBD 的另一种形式。在整个程序中，可以自定义运算块的顺序，易于实现流程运算，它用于描述资源的顶层结构以及程序和功能块对任务的分配。

连续功能流程图和功能块图之间的主要区别是资源和任务分配的不同。每一功能用任务的名称来描述，如图所示。如果一个程序内的功能块象它的父程序一样在相同的任务下执行，任务关联是隐含的。连续功能图 CFC 如图 5-69 所示。

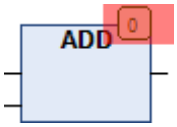
图 5-69 连续功能图 CFC



5.6.1.2 执行顺序

在 CFC 语言里元素的右上角的数字，显示了在线模式下 CFC 中元素的执行顺序。执行流程从编号为 0 的元素开始，在每个 PLC 运算周期内，0 号元素令总是第一个被执行。当手动移动该元素时，它的编号仍保持不变。当添加一个新元素时，系统自动按照拓扑序列（从左到右，从上到下）会自动获得一个编号，如图 5-70 红色部分所示。

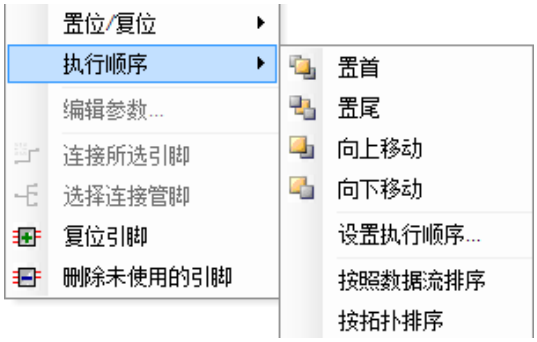
图 5-70 CFC 编程语言顺序编号



CFC 语言中运算块、输出、跳转、返回和标签元素的右上角的数字，显示了在线模式下 CFC 中元素的执行顺序。执行流程从编号为 0 的元素开始。考虑到执行顺序会影响到结果，在一定情况下可以改变执行顺序。操作在菜单“CFC”下的“执行顺序”中的子菜单命令可以改变元素的执行顺序。

执行顺序包含的命令有：置首、置尾、向上移动、向下移动、设置执行顺序、按数据流排序、按拓扑排序，如图 5-71 所示。

图 5-71 CFC 顺序编排



1、 置首

把选中元素移到执行顺序的首端。如果选中多个元素执行这个命令时，选中元素的原有的内部顺序保持不变；未选中元素的内部顺序也保持不变。

2、 置尾

把所有选中元素移到执行顺序的末端。选中元素的内部顺序保持不变；未选中元素的内部顺序也保持不变。具体操作可以参照上述“置首”功能。

3、 向上移动

把所有选中元素（如果某个元素已在执行顺序的首端，除去该元素）在执行顺序上向前移动一位。如选中图 7 中的

3 号元素执行“向上移动”命令，结果是 2 号元素与 3 号元素的执行顺序互换了一下，其余都不变。

4、 向下移动

把所有选中元素（如果某个元素已在执行顺序的末端，除去该元素）在执行顺序上向后移动一位。具体操作参考“向上移动”。

5、 设置执行顺序

该命令可以对选中元素重新编号，调整元素的执行顺序。执行“设置执行顺序”命令后，会打开“设置执行顺序”对话框。在当前执行次序区域显示当前单元编号，用户可以在“新处理顺序”中输入需要单元编号，括弧内的值为可选值，如下图所示。

6、 按数据流排序

数据流排序表示各个元素按照数据流顺序执行，而不是按照元素所在位置（拓扑）决定执行顺序。执行数据流排序命令后，CFC 编辑器内部做如下操作。

步骤1 按照拓扑对所有元素进行排序。

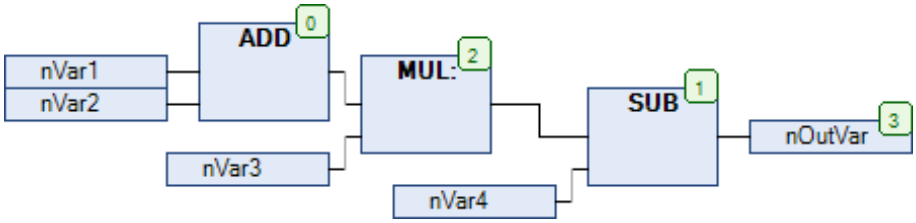
步骤2 创建一个新的执行顺序链表。

步骤3 找到那些已知的输入值，下一步可以被执行但还没有放入到链表中的元素。

数据流排序的优点是：一个算法执行后，连接到它的输出引脚上算法块会立刻执行；但是在拓扑排序中却不一定是这样。拓扑排序的执行结果可能和数据流排序的执行结果不同。

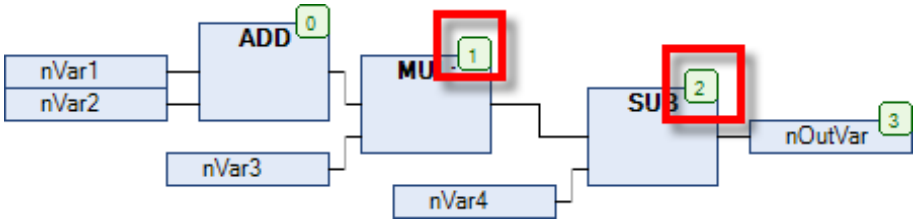
【例5.44】 图为打乱元素标号的程序，使用“按数据流排序”的排序方法查看结果。

图 5-72 使用数据流排序之前



选中全部元素后执行“按数据流排序”后结果如图 5-73 所示。

图 5-73 使用数据流排序之后



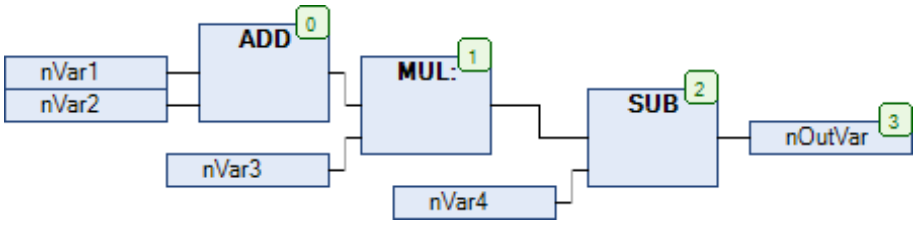
元素的编号按照数据流的流向重新编排的，函数 MUL 和 SUB 的执行顺序较之前有了改变。

7、 按拓扑排序

拓扑排序表示各个元素按照拓扑顺序执行，而不是按照元素数据流决定执行顺序。拓扑排序后，元素按照从左到右，从上到下的顺序执行；左边的元素的执行顺序编号小于右边的，上边的小于下边的。拓扑排序依据的是元素的位置坐标，与连线位置无关。

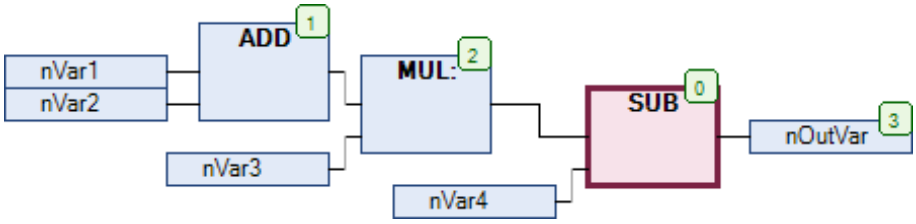
【例5.45】 图 5-74 为打乱元素标号的程序，使用“按拓扑排序”的排序方法查看结果。

图 5-74 使用按拓扑排序之前



选中 SUB 函数，右键执行“按拓扑排序”命令后结果如图 5-75 所示。

图 5-75 使用按拓扑排序之后



拓扑排列的顺序就是：从左到右，从上到下的顺序执行；左边的元素的执行顺序编号小于右边的，上边的小于下边的。

5.6.2 连接元素

CFC 的元素包括块、输入、输出、跳转、标记、返回和注释等。

图 5-76 CFC 工具箱

5.6.2.1 指针

指针默认在工具箱最上方。一旦这个条目被选中，光标编程箭头形状，就可以选择编辑窗口中的元素来放置并编辑它们。

5.6.2.2 输入、输出

输入

在 CFC 工具箱中插入“”符号添加合成器功能。在 CFC 编辑器中插入后的图示为“???”。

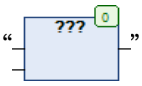
选中“???”文本，然后修改为变量或者常量。通过输入助手可以选择输入一个有效标识符。

输出

在 CFC 工具箱中插入“”符号添加合成器功能。在 CFC 编辑器中插入后的图示为“??? 0”。

选中“???”文本，然后修改为变量或者常量。通过输入助手可以选择输入一个有效标识符。

5.6.2.3 运算块

在CFC工具箱中插入“”符号添加合成器功能。在CFC编辑器中插入后的图示为“ 

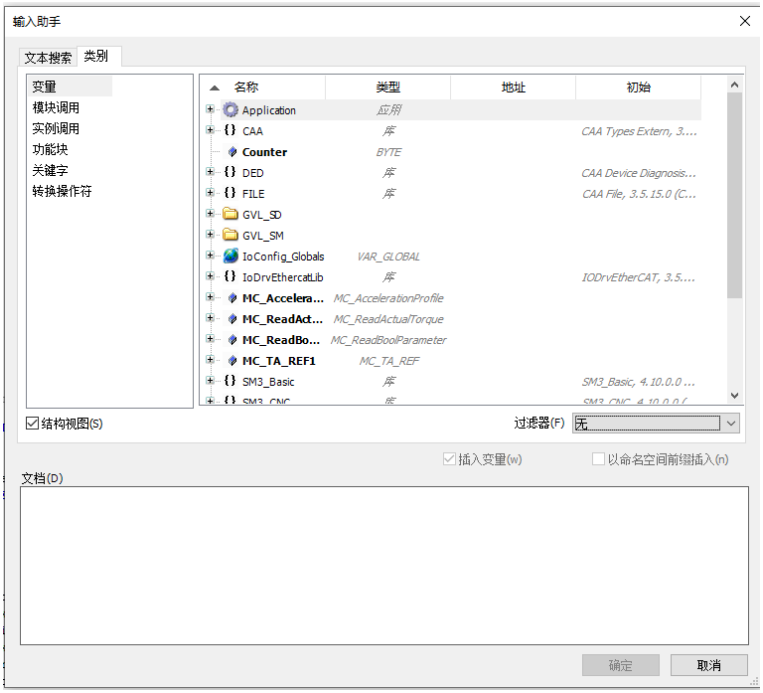
”。

运算块可用来表示操作符，函数，功能块和程序。插入运算块后，选中运算块的“???”文本框，将其修改为操作符名，函数名，功能块名或者程序名。或者可以通过“输入助手”选择输入一个有效的对象。

【例5.46】 通过运算块，在 CFC 编程语言中调用定时器功能块。

新建POU，使用CFC编程语言，添加“运算块”，点击“???”输入“F2”弹出输入助手，在其中选择功能块，找到定时器功能块，如图 5-77所示。

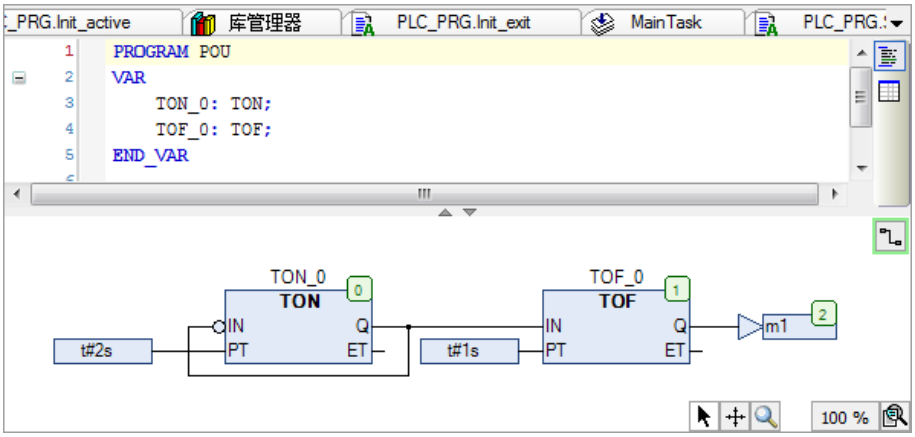
图 5-77 CFC 输入助手工具



如果需要在 ST 中调用功能块，可直接输入功能块的实例名称，并在随后的括号中给功能块的各参数分配数值或变量。参数之间以逗号（，）隔开，功能块调用以分号（；）结束。

例如，在结构化文本中调用功能块 TON 定时器，假设其实例名为 TON1，具体实现如图 5-78 所示。

图 5-78 连续功能图调用功能块



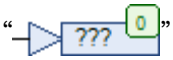
在例子中，当插入一个功能块，随即运算块上出现另一个“???”，这时要把“???”修改为功能块实例名 0，在本例中实例名为 TON_0 和 TOF_0。

若运算块被修改为另一个运算块（通过修改运算块名），而且新运算块的最大输入或输出引脚数，或者最小输入或输出引脚数与前者不同。运算块的引脚会自动做相应的调整。若要删除引脚，则首先删除最下面的引脚。

5.6.2.4 跳转&标签

CFC 程序的跳转由跳转指令和标签两部分组成，如下会对这两部分做详细介绍。

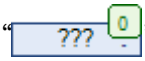
跳转

在 CFC 工具箱中插入“”符号可以实现跳转功能。在 CFC 编辑器中插入后的图示为“ 

”。

跳转用来指示程序下一步执行到哪，具体去哪最终“标签”所定义，如下会介绍“标签”的使用方法。插入一个新跳转后，要用跳转名替代“???”。

标签

在 CFC 工具箱中插入“”符号可以添加标签。在 CFC 编辑器中插入后的图示为“ 

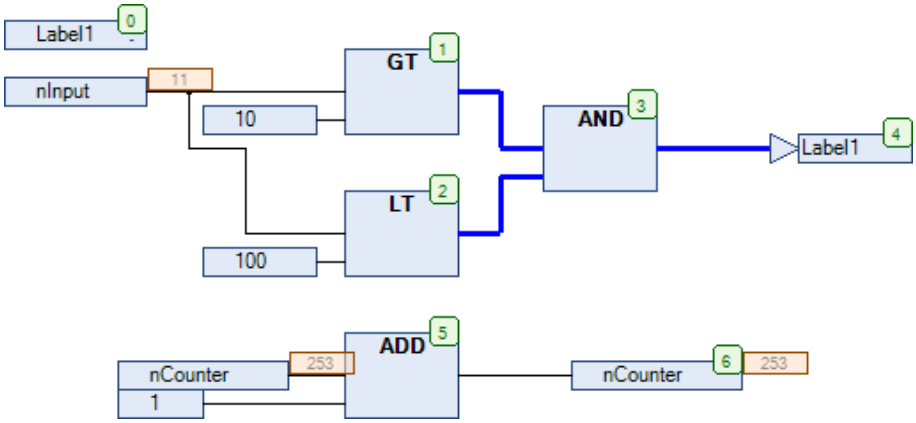
”。

“标签”可以标识程序跳转的位置在在线模式下，如果跳转被激活，则可以进入跳转对应的标签，如此循环执行。

标签名不属于变量，故在程序声明区中不需要对其定义，【例 5.49】说明了如何正确使用跳转指令和标签。

【例5.47】 CFC 跳转指令和标签功能示例。

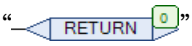
图 5-79 CFC 跳转功能示例



程序启动后，当输入值 nInput 大于 10 且小于 100 时，程序执行跳转功能，转至标签 Label1，由于 Label1 的执行序号为 0，故在该程序中执行的顺序为：4→0→1→2→3→4，如此循环执行。

由于在此程序中还有自加 1 的功能，但执行序号为 5 和 6，故当跳转指令被执行时，该自累加功能不会被程序执行，反之 nCounter 则进行自累加。

5.6.2.5 返回

在 CFC 工具箱中插入“”符号可以实现返回功能。在 CFC 编辑器中插入后的图示“ 

”。

需特别注意该执行顺序号，当条件满足时，则会直接返回程序。

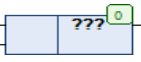
命令返回一个入令插这个命注意在联机模式下带 RETURN 名字的跳转标签自动插入到第一列和编辑器最后元素的后面，在分支中，它自动跳转到执行将离开 POU 之前的地方。

RETURN 指令是返回指令，用于退出程序组织单元（POU）。

注意：在线模式下，RETURN 自动插入到编辑器最后那个元素之后。在单步调试中，在离开该 POU 之前，会自动跳转到该 RETURN。

5.6.2.6 合成器

合成器用于结构体类型的运算块输入。合成器会显示结构体的所有成员，以方便编程人员使用它们。

在CFC工具箱中插入“”符号添加合成器功能。在CFC编辑器中插入后的图示为“ 

”。

使用方法是：先增加一个合成器到编辑器中，修改“???”为要使用的结构体名字，然后连接合成器的输出引脚和运算块的输入引脚。

【例5.48】 CFC 程序 CFC_PROG 处理一个功能块实例 fubblo1，有一个结构体类型输入变量 struvar。通过使用

合成器元素，结构体变量可以访问：

结构stru1定义:

```
TYPE stru1 :
STRUCT
ivar:INT;
strvar:STRING:='hallo';
END_STRUCT
END_TYPE
```

功能块fublo1,声明和实现:

```
FUNCTION_BLOCK fublo1
VAR_INPUT
struvar:STRU1;
END_VAR
VAR_OUTPUT
fbout_i:INT;
fbout_str:STRING;
END_VAR
VAR
fbvar:STRING:='world';
END_VAR
fbout_i:=struvar.ivar+2;
fbout_str:=CONCAT (struvar.strvar,fbvar);
```

程序CFC_PROG的声明和实现:

```
PROGRAM PLC_PRG
VAR
intvar: INT; stringvar: STRING;
fbinst: fublo1;
erg1: INT;
erg2: STRING;
END_VAR
```

程序如图 5-80所示，‘1’为合成器，‘2’为含有结构体输入变量的stru1，实现了结构体类型的输入块运算。

图 5-80 CFC 合成器程序示例

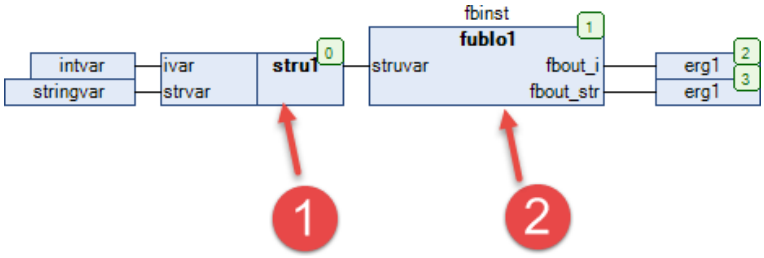
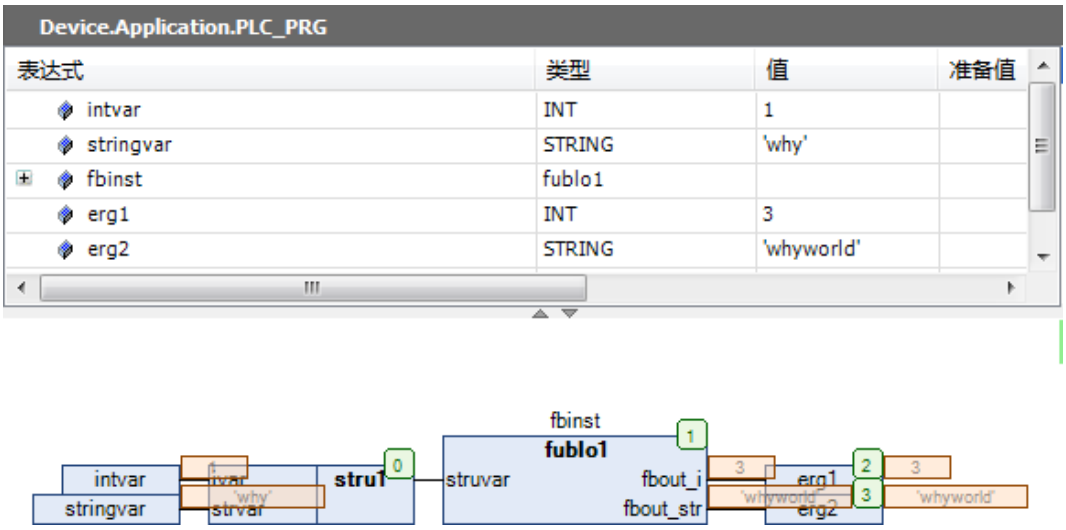
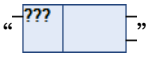


图 5-81 CFC 合成器示例运行结果



5.6.2.7 选择器

选择器用于结构体类型的运算块输出。选择器会显示结构体的所有成员，以方便编程人员使用它们。

在 CFC 工具箱中插入“ ”符号添加选择器功能。在 CFC 编辑器中插入后的图示为“  ”。

使用方法是：先增加一个选择器到编辑器中，修改“???”为要使用的结构体名字，然后连接选择器的输出引脚和运算块的输出引脚。

【例5.49】 CFC 程序 cfc_prog 处理功能块实例 fubblo2,它有一个 stru1 结构的输出变量 fbout。通过选择元素，结构体成员变量可以被访问：

结构定义 stru1:

```
TYPE stru1 :
STRUCT
ivar:INT;
strvar:STRING:='hallo';
END_STRUCT
END_TYPE
```

功能块 fublo1，声明与实现：

```
FUNCTION_BLOCK fublo2
VAR_INPUT
fbin : INT;
fbin2:STRING;
END_VAR
VAR_OUTPUT
fbout : stru1;
END_VAR
VAR
fbvar:INT:=2;
fbin3:STRING:='Hallo';
END_VAR
```

程序 PLC_PRG_1，声明与实现：

```
PROGRAM PLC_PRG_1 VAR
```

intvar: INT;
stringvar: STRING;
fbinst: fublo2;
erg1: INT;
erg2: STRING;
fbinst2: fublo2;
END_VAR

程序如图 5-82 所示，‘1’为拥有 stru1 结构体类型输出变量 fbout 的功能块，‘2’选择器。

图 5-82 CFC 选择器程序示例

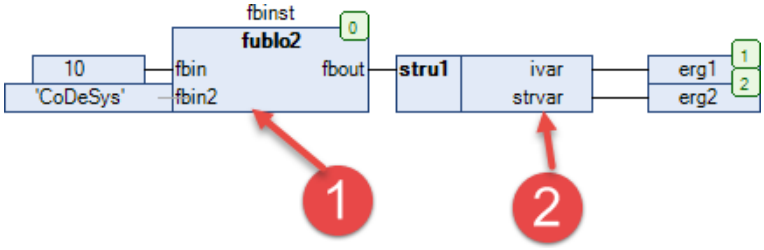
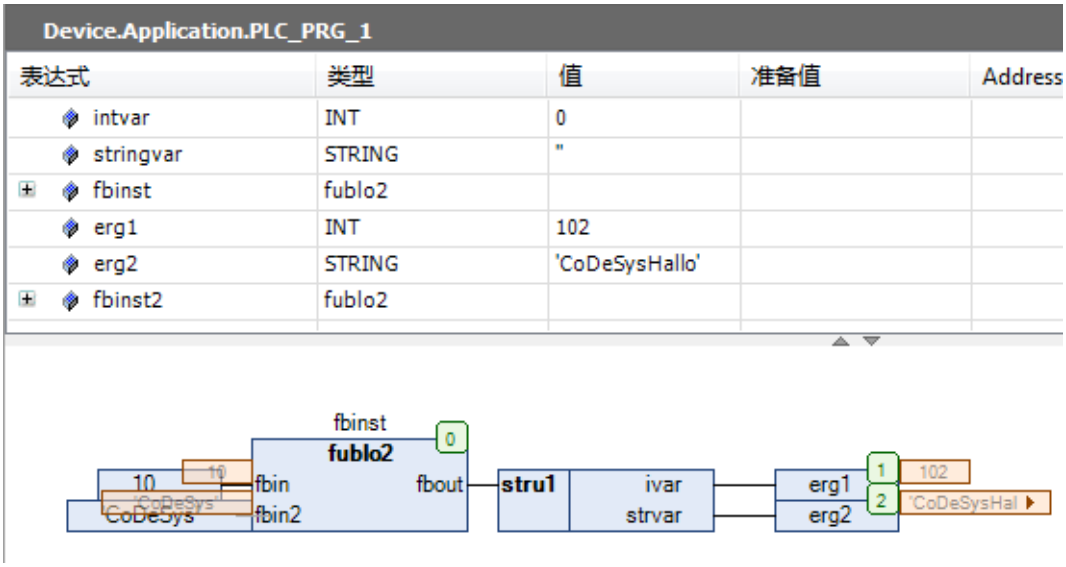


图 5-83 CFC 选择器示例运行结果

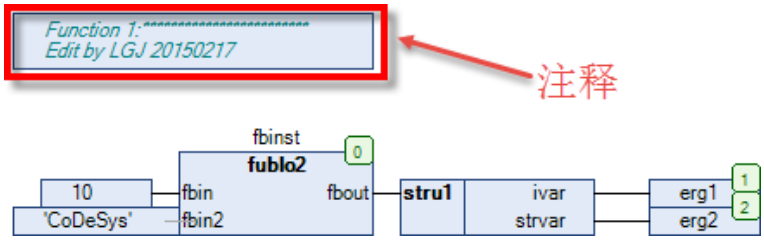


5.6.2.8 注释

在 CFC 工具箱中插入“▭”符号可以实现添加注释功能。

在 CFC 编辑器中插入后的图示为“ <在此处添加注释> ”。

通过添加该元素，可以在 CFC 程序中为图表添加注释。选中文本，直接输入注释即可。如需要换行，输入<ctrl>+<enter>即可实现换行，下图为 CFC 中的注释视图。



5.6.2.9 输入、输出引脚

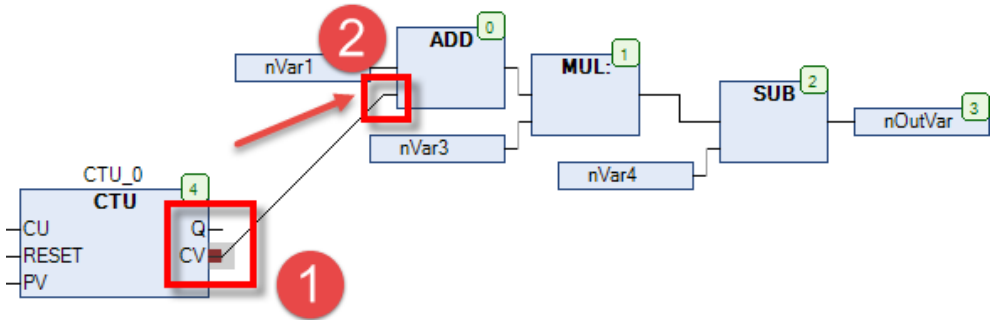
有些运算块可以增加输入引脚（或输出引脚）。首先在工具箱中选中 Input Pin（或 Output Pin），然后拖放到在 CFC 编辑器中的算法块上，该运算块就会增加一个输入引脚（或输出引脚）。

5.6.3 CFC 的组态

1、 在 CFC 程序中添加连接

添加连接时，首先激活连接程序块的管脚，激活后在管脚出会有一个方形区域出现，鼠标左键选中这个方形区域，如图 5-84的‘1’所示，按住鼠标至要连接的另一个连接点处松开鼠标，连接至图 5-84中的‘2’处，即完成了两部分的连接。

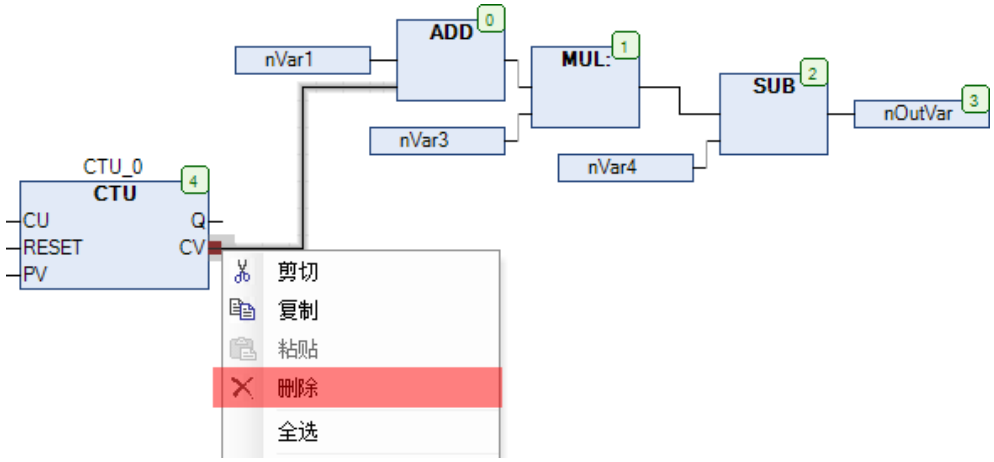
图 5-84 CFC 程序添加连接



2、 在 CFC 程序中删除连接

删除连接时，首先激活连接程序块的管脚，激活后在管脚出会有一个方形区域出现，鼠标右键选中这个方形区域，在随后显示菜单栏中选择“删除”即可，如图 5-85 中框出部分所示。也可以在快捷菜单栏中选择“ ”按钮，删除程序中的连接线。

图 5-85 CFC 程序删除连接用



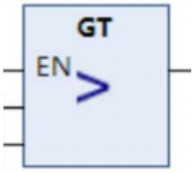
6 基础指令

6.1 比较指令

6.1.1 GT 大于

判断两个输入值的大小，当第一个输入值大于第二个输入值时输出 TRUE，否则输出 FALSE。

FBD 中示例：

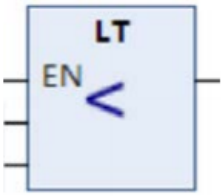


注意：当两个输入变量数据类型不一致时编译会报错。

6.1.2 LT 小于

判断两个输入值的大小，当第一个输入值小于第二个输入值时输出为 TRUE，否则输出为 FALSE。

FBD 中示例：

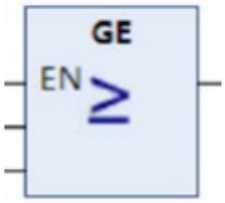


注意：当两个输入变量数据类型不一致时编译会报错。

6.1.3 GE 大于等于

判断两个输入值的大小，当第一个输入值小于第二个输入值时输出为 TRUE，否则输出为 FALSE。

FBD 中示例：

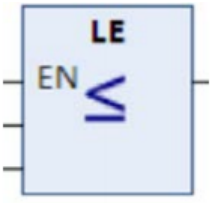


注意：当两个输入变量数据类型不一致时编译会报错。

6.1.4 LE 小于等于

判断输入值的大小，当第一个输入值小于或等于第二个输入值时输出 TRUE，否则输出 FALSE。

FBD 中示例：

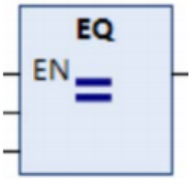


注意：当两个输入变量数据类型不一致时编译会报错。

6.1.5 EQ 等于

判断两个输入值大小，当第一个输入值等于第二个输入值时输出为 TRUE，否则输出 FALSE。

FBD 中示例：

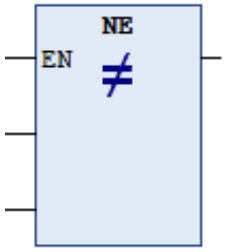


注意：当两个输入变量数据类型不一致时编译会报错。

6.1.6 NE 不等于

判断两个输入值的大小，当第一个数不等于第二个数时输出为 TRUE，否则输出为 FALSE。

FBD 中示例：



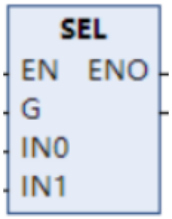
注意：当两个输入变量数据类型不一致时编译会报错。

6.2 选择指令

6.2.1 SEL 二选一指令

当 G=FLASE 时，输出 IN0；当 G=TRUE 时，输出 IN1。

FBD 中示例：

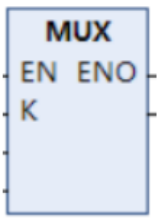


注意：当 G 为真时,CODESYS 不计算 IN0 之前的表达式。当 G 为假时，CODESYS 不会计算 IN1 之前的表达式。

6.2.2 MUX 多选一指令

从一组值中选择第 K 个值。第一个值是 K=0。如果 K 比其他输入的值大，那么 CODESYS 传递最后的一个值(INn)。

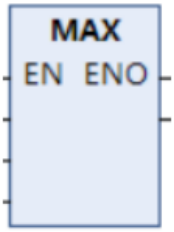
FBD 中示例：

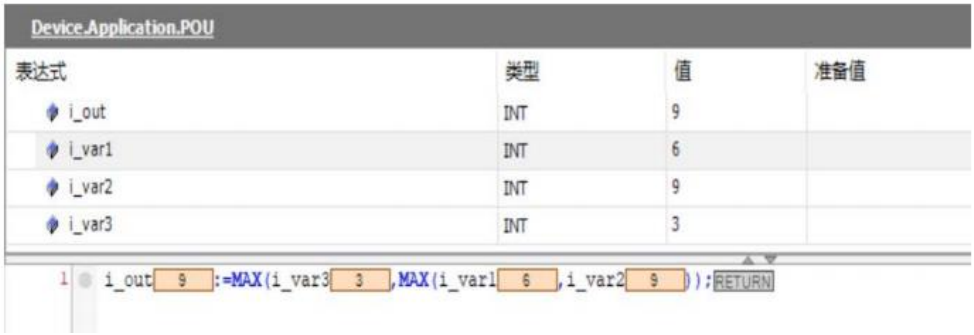


6.2.3 MAX 取最大值指令

取两个输入值中的最大值，并将最大值从右侧输出。

FBD 中示例：

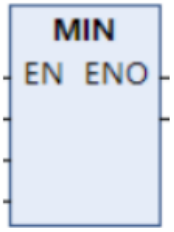




6.2.4 MIN 取最小值指令

取两个输入值中的最小值，并将最小值从右侧输出。

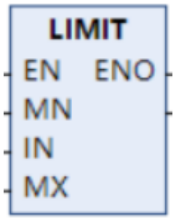
程序实例：



6.2.5 LIMIT 极限值指令

MX 是结果的上限值，MN 是结果的下限值。如果 IN 得值超过了 MX（上限值），则输出运算结果为 MX，如果 IN 的值小于 MN（下限值），则输出运算结果为 MN。当 IN 的值在 MN 和 MX 范围内的时候，则输出的值为输入的 IN 的值。

FBD 中示例：



注意： MX 和 MN 数据类型必须相同。

6.3 计数器指令

6.3.1 CTU 向上计数功能块

该功能块为向上计数功能块。

输入：

CU:BOOL;上升沿触发 CV 的递增计数

RESET:BOOL;为 TRUE 时，CV 被复位为 0

PV:WORD;CV 计数的上限

输出：

Q:BOOL;当 CV 达到计数上限 PV 时，其值为 TRUE

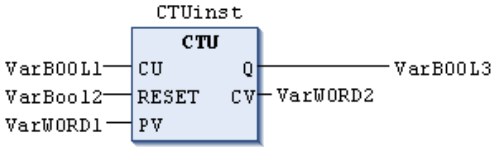
CV:WORD;不断加 1 的值，直至 CV

如果 RESET 的值为 TRUE，计数值 CV 将初始化为 0。如果 CU 从 FALSE 到 TRUE 有一个上升沿，CV 的值加 1。当 CV 大于等于计数上限值 PV 时，Q 的值为 TRUE。

声明示例：

CTUInst:CTU;

FBD 中示例：



ST 中示例：

CTUInst(CU:=VarBOOL1,RESET:=VarBOOL2,PV:=VarWORD1);

VarBOOL3:=CTUInst.Q;

VarWORD2:=CTUInst.CV;

6.3.2 CTD 向下计数功能块

该功能块为向下计数功能块。

输入：

CD:BOOL;上升沿触发 CV 的递减计数

LOAD:BOOL;为 TRUE 时，CV 被置为上限值 PV

PV:WORD;CV 开始递减时的初始值

输出：

Q:BOOL;当 CV 达到 0 时，其值为 TRUE

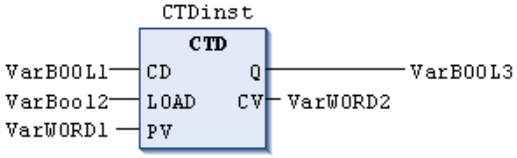
CV:WORD;不断减 1 的值，从 PV 开始直至 0

当 LOAD 的值为 TRUE 时，计数值 CV 将初始化为计数上限值 PV。如果 CD 从 FALSE 到 TRUE 有一个上升沿，且 CV 大于 0，则 CV 的值减 1（即 CV 的值不能小于 0）。当 CV 等于 0 时，Q 的值为 TRUE。

声明示例：

CTDInst:CTD;

FBD 中示例：



ST 中示例：

```
CTDInst(CD:=VarBOOL1,LOAD:=VarBOOL2,PV:=VarWORD1);
VarBOOL3:=CTDInst.Q;
VarWORD2:=CTDInst.CV;
```

6.3.3 CTUD 向上/向下计数功能块

该功能块为向上/向下计数功能块。

输入：

CU:BOOL;上升沿触发 CV 的递增计数

CD:BOOL;上升沿触发 CV 的递减计数

RESET:BOOL;为 TRUE 时，CV 被复位为 0

LOAD:BOOL;为 TRUE 时，CV 被置为上限值 PV

PV:WORD;CV 开始递增时的上限值，或 CV 递减时的初始值

输出：

QU:BOOL;当 CV 达到计数上限 PV 时，其值为 TRUE

QD:BOOL;当 CV 达到 0 时，其值为 TRUE

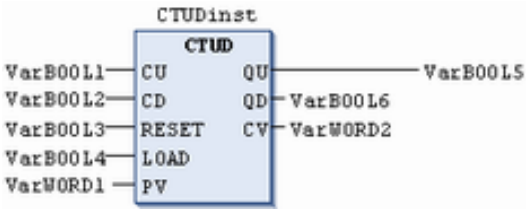
CV:WORD;不断减 1 的值，从 PV 开始直至 0

如果 CU 从 FALSE 到 TRUE 有一个上升沿，CV 的值加 1。如果 CD 从 FALSE 到 TRUE 有一个上升沿，且 CV 大于 0，则 CV 的值减 1。当 CV 大于等于 PV 时，QU 的值为 TRUE。当 CV 等于 0 时，QD 的值为 TRUE。

声明示例：

```
CTUDInst:CUTD;
```

FBD 中示例：



ST 中示例：

```
CTUDInst(CU:=VarBOOL1,CD:=VarBOOL2,RESET:=VarBOOL3,LOAD:=VarBOOL4,PV:=VarWORD1);
VarBOOL5:=CTUDInst.QU;
VarBOOL6:=CTUDInst.QD;
VarWORD2:=CTUDInst.CV;
```

6.4 定时器指令

6.4.1 TP 触发定时器功能块

该功能块是一个触发定时器功能块。

输入：

IN:BOOL;上升沿触发 ET 端计时

PT:TIME;ET 计时时间的上限值

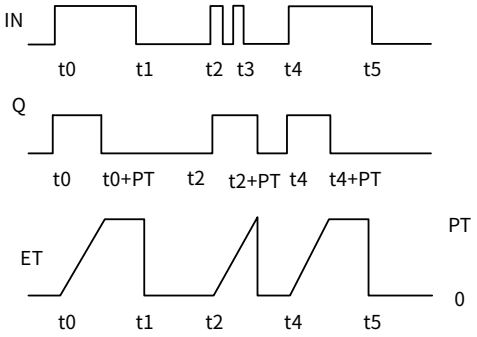
输出：

Q:BOOL;当 ET 在计时的时候，其值为 TRUE

ET:TIME;时间的当前状态

如果 IN 的值为 FALSE，则 Q 的值为 FALSE，且 ET 的值为 0。当 IN 的值为 TRUE 时，ET 中的时间将以毫秒为单位开始计数，直至 ET 等于 PT。ET 等于 PT 之后将一直保持这个常量。如果 IN 为 TRUE 且 ET 小于等于 PT 时，则 Q 为 TRUE，否则 Q 为 FALSE。

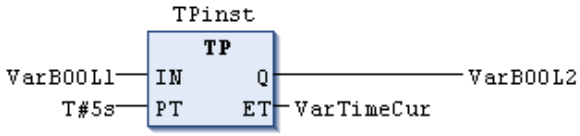
在 PT 所定义的时间周期中，Q 为 TRUE。TP 的时间顺序图如下：



声明示例：

TPInst:TP;

FBD 中示例：



ST 中示例：

TPInst(IN:=VarBOOL1,PT:=T#5s);

VarBOOL2:=TPInst.Q;

6.4.2 TON 延时接通定时器

该功能块是一个延时接通定时器。

输入：

IN:BOOL;上升沿触发 ET 端计时

PT:TIME;ET 计时时间的上限值（即延时时间）

输出：

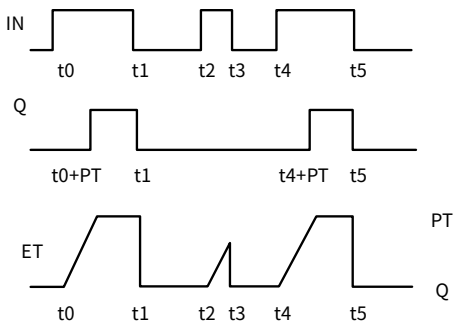
Q:BOOL;当 ET 计时达到上限值 PT，输出一个上升沿

ET:TIME;时间的当前状态

TP(IN,PT,Q,ET)的意思是：IN 与 PT 分别为 BOOL 类型与 TIME 类型的输入变量。Q 与 ET 分别为 BOOL 类型与 TIME 类型的输出变量。如果 IN 的值为 FALSE，则 Q 的值为 FALSE，且 ET 的值为 0。

当 IN 的值为 TRUE 时，ET 中的时间将以毫秒为单位开始计数，直至 ET 等于 PT。ET 等于 PT 之后将一直保持这个常量。如果 IN 为 TRUE 且 ET 等于 PT，则 Q 为 TRUE。否则 Q 为 FALSE。因此，当延时（PT 所定义的时间）完成后，Q 出现一个上升沿。

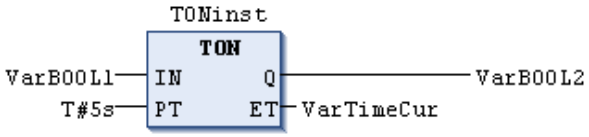
TON 的时间顺序图如下：



声明示例：

TONInst:TON;

FBD 中示例：



ST 中示例：

TONInst(IN:=VarBOOL1,PT:=T#5s);

6.4.3 TOF 延时关断定时器

该功能块是一个延时关断定时器。

输入：

IN:BOOL;下降沿触发 ET 端计时

PT:TIME;ET 计时时间的上限值（即延时时间）

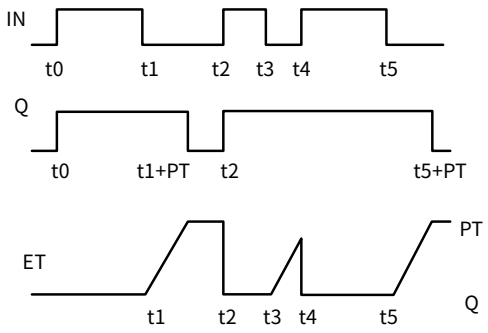
输出：

Q:BOOL;当 ET 计时达到上限值 PT，输出一个下降沿

ET:TIME;时间的当前状态

TOF(IN,PT,Q,ET)的意思是：如果 IN 为 TRUE，则 Q 为 TRUE。当 IN 的值为 FALSE 时，ET 中的时间将以毫秒为单位开始计数，直至 ET 等于 PT。ET 等于 PT 之后将一直保持这个常量。如果 IN 为 FALSE 且 ET 等于 PT，则 Q 为 FALSE，否则 Q 为 TRUE。因此当延时完成后，Q 出现一个下降沿。

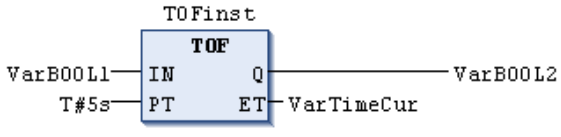
TOF 的时间顺序图如下：



声明示例：

TOFInst:TOF;

FBD 中示例：



ST 中示例：

TOFInst(IN:=VarBOOL1,PT:=T#5s);

VarBOOL2:=TOFInst.Q;

6.4.4 RTC 时钟计时功能块

该功能块具有时钟功能，从设定时间开始计时。

输入：

EN:BOOL;上升沿触发 CDT 端的计时

PDT:DATE_AND_TIME;计时开始的日期和时间

输出：

Q:BOOL;当 CDT 端开始计时，输出为 TRUE

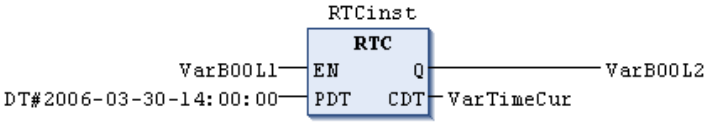
CDT:DATE_AND_TIME;计时的当前日期和时间

VarBOOL2:=RTC(EN,PDT,Q,CDT)表示：当 EN 为 FALSE 时，输出变量 Q 为 FALSE，CDT 为 DT#1970-01-01-00:00:00。一旦 EN 变为 TRUE（上升沿到来），只要 EN 一直保持为 TRUE，CDT 端以 PDT 为初始值以秒为精度递增计时。当 EN 复位为 FALSE，CDT 复位为初始值 DT#1970-01-01-00:00:00。

声明示例：

RTCInst:RTC;

FBD 中示例：



ST 中示例：

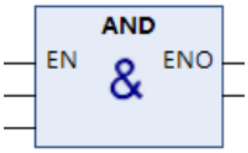
RTCInst(EN:=VarBOOL1,PDT:=DT#2006-03-30-14:00:00,Q=>VarBOOL2,CDT=>VarTimeCur);

6.5 位和字逻辑指令

6.5.1 AND 与指令

当左侧两个输入位非 0，右侧输出位也输出 1，否则输出 0。

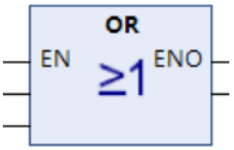
FBD 中示例：



6.5.2 OR 或指令

当左侧两个输入位至少有一个为非零值，右侧输出位的值为 1，否则为 0。

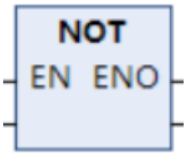
FBD 中示例：



6.5.3 NOT 非指令

当左侧输入位输入 0，右侧输出位输出 1，当左侧输入位输入 1，右侧输出位输出 0。

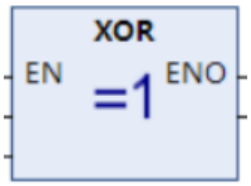
FBD 中示例：



6.5.4 XOR 异或指令

当左侧两输入位其中一位输入 1，另一位输入 0，输出 1；当两个输入值同为 1 或 0 时，输出 0。

FBD 中示例：



6.5.5 SR 置位优先功能块

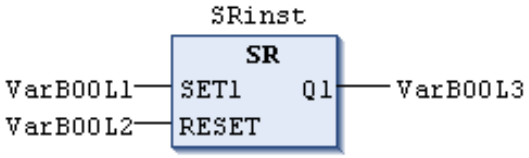
该功能块是置位优先的双稳态功能块。Q1=SR(SET1,RESET)的意思是：Q1=(NOTRESETANDQ1)ORSET1。

输入变量 SET1 与 RESET 以及输出变量 Q1 均为 BOOL 类型。

声明示例：

SRInst:SR;

FBD 中示例：



ST 中示例：

SRInst(SET1:=VarBOOL1,RESET:=VarBOOL2);

VarBOOL3:=SRInst.Q1;

6.5.6 RS 复位优先功能块

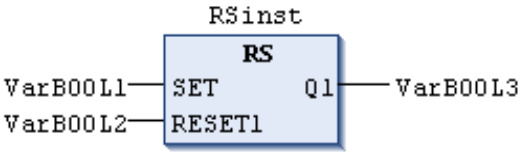
该功能块是复位优先的双稳态功能块。Q1=RS(SET,RESET1)的意思是：Q1=NOTRESET1AND(Q1ORSET)。

输入变量 SET1 与 RESET 以及输出变量 Q1 均为 BOOL 类型。

声明示例：

RSInst:RS;

FBD 中示例：



ST 中示例：

RSInst(SET:=VarBOOL1,RESET1:=VarBOOL2);

VarBOOL3:=RSInst.Q1;

6.5.7 R_TRIG 上升沿功能块

检测上升沿功能块。

输入：

CLK:BOOL;布尔输入信号用于检测上升沿

输出：

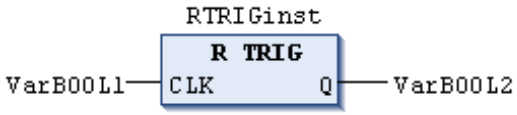
Q:BOOL;如果 CLK 检测到上升沿，输出为 TRUE

当 CLK 从“FALSE”变为“TRUE”时，该上升沿检测器开始启动，Q 输出先为“TRUE”然后输出变为“FALSE”，持续一个 PLC 运算周期；如果 CLK 持续保持为“TRUE”或者“FALSE”，Q 输出一直保持为“FALSE”。

声明示例：

RTRIGInst:R_TRIG;

FBD 中示例：



ST 中示例：

RTRIGInst(CLK:=VarBOOL1);

VarBOOL2:=RTRIGInst.Q;

6.5.8 F_TRIG 下降沿功能块

检测下降沿功能块。

输入：

CLK:BOOL;布尔输入信号用于检测下降沿

输出：

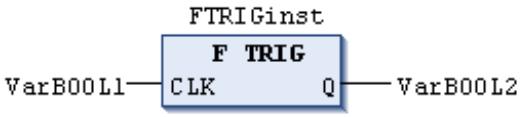
Q:BOOL;如果 CLK 检测到下降沿输出为 TRUE

当 CLK 从“TRUE”变为“FALSE”时，该下降沿检测器开始启动，Q 输出先为“TRUE”然后输出变为“FALSE”，持续一个 PLC 运算周期；如果 CLK 持续保持为“TRUE”或者“FALSE”，Q 输出一直保持为“FALSE”。

声明示例：

FTRIGInst:F_TRIG;

FBD 中示例：



ST 中示例：

FTRIGInst(CLK:=VarBOOL1);

VarBOOL2:=FTRIGInst.Q;

6.6 位/字节函数

6.6.1 EXTRACT

该函数的输入变量 X 为 DWORD 类型，N 为 BYTE 类型。输出变量为 BOOL 类型，输出为输入变量 X 的第 N 位的数值，N 从第 0 位开始计算。

ST 中示例：

FLAG:=EXTRACT(X:=81,N:=4);

(*Result:TRUE,because81isbinary1010001,sothe4thbitis1*)

FLAG:=EXTRACT(X:=33,N:=0);

(*Result:TRUE,because33isbinary100001,sothebit'0'is1*)

6.6.2 PACK 组合指令

该函数用于将 8 个 BOOL 类型输入变量 B0，B1，…，B7 组合为一个 BYTE 类型数据。

功能块 UNPACK 与该功能有着密切的关系。

6.6.3 PUTBIT

该函数的输入变量 X，N 与 B 分别为 DWORD 类型，BYTE 类型与 BOOL 类型。

PUTBIT 用于将 X 的第 N 位设置为值 B，N 从第 0 位开始计算。

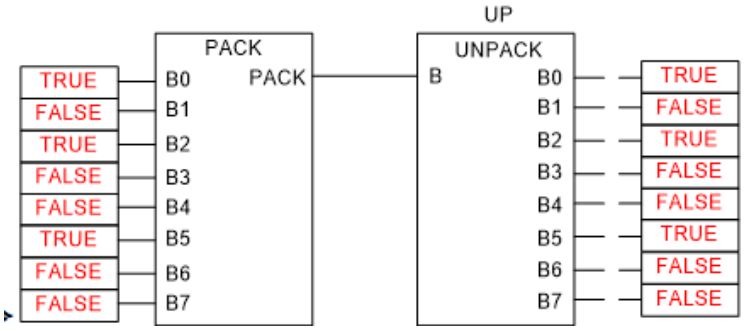
ST 中示例：

```
var1:=38;(*binary100110*)
var2:=PUTBIT(A,4,TRUE);(*Result:54=2#110110*)
var3:=PUTBIT(A,1,FALSE);(*Result:36=2#100100*)
```

6.6.4 UNPACK 拆分功能块

UNPACK 用于将 BYTE 类型的输入变量 B 拆分为 8 个 BOOL 类型的输出变量 B0，B1，…B7。它与 PACK 的功能相反。

FBD 中示例：

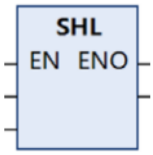


6.7 数据移位指令

6.7.1 SHL 按位左移

对输入值进行按位左移，左边移出的位不作处理，右边自动补 0。

FBD 中示例：

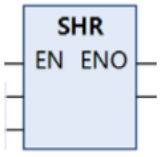


注意：数据只能为整型，若为浮点型则会报错。

6.7.2 SHR 按位右移

对输入值进行按位右移，右边移出的位不作处理，左边自动补 0。

FBD 中示例：

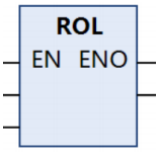


注意：数据只能为整型，若为浮点型则会报错。

6.7.3 ROL 按位循环左移

对输入值进行按位循环左移，左边移出的位直接补充到右边最低位。

FBD 中示例：

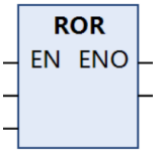


注意：指令支持数据类型为整型。

6.7.4 ROR 按位循环右移

对输入值进行按位循环右移，右边移出的位直接补充到左边最高位。

FBD 中示例：



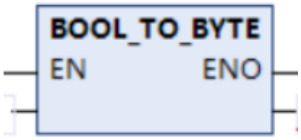
注意：指令支持数据类型为整型。

6.8 数据类型转换指令

6.8.1 BOOL_TO_<TYPE>

将布尔类型变量转换为其他类型变量。

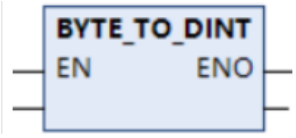
FBD 中示例：



6.8.2 BYTE_TO_<TYPE>

将字节类型变量转换为其他类型变量

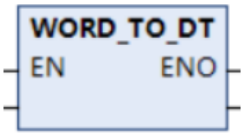
FBD 中示例：



6.8.3 WORD_TO_<TYPE>

将字类型变量转换为其他类型变量。

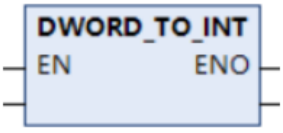
FBD 中示例：



6.8.4 DWORD_TO_<TYPE>

将双字类型变量转换为其他类型变量。

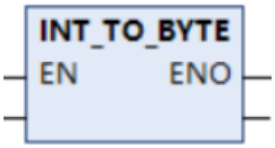
FBD 中示例：



6.8.5 INT_TO_<TYPE>

将整数类型变量转换为其他类型变量。

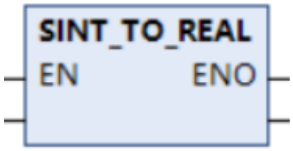
FBD 中示例：



6.8.6 SINT_TO_<TYPE>

将短整型类型变量转换为其他类型变量。

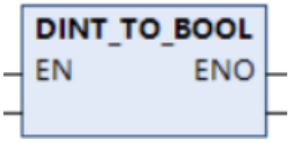
FBD 中示例：



6.8.7 DINT_TO_<TYPE>

将长整型类型变量转换为其他类型变量。

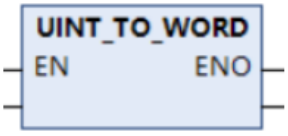
FBD 中示例：



6.8.8 UDINT_TO_<TYPE>

将无符号长整型类型变量转换为其他类型变量。

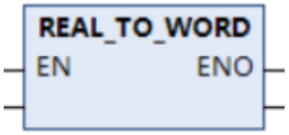
FBD 中示例：



6.8.9 REAL_TO_<TYPE>

将实数类型变量转换为其他类型变量。

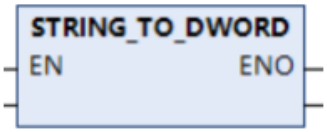
FBD 中示例：



6.8.10 STRING_TO_<TYPE>

将字符类型变量转换为其他类型变量。

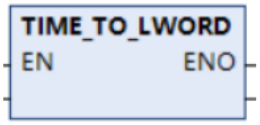
FBD 中示例：



6.8.11 TIME_TO_<TYPE>

将时钟类型变量转换为其他类型变量。

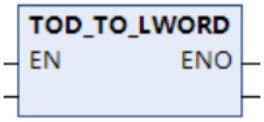
FBD 中示例：



6.8.12 TOD_TO_<TYPE>

将时间类型变量转换为其他类型变量。

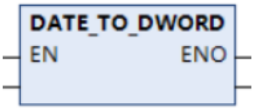
FBD 中示例：



6.8.13 DATE_TO_<TYPE>

将日期类型变量转换为其他类型变量。

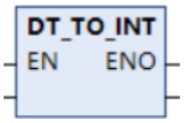
FBD 中示例：



6.8.14 DT_TO_<TYPE>

将日期时间类型变量转换为其他类型变量。

FBD 中示例：

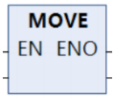


6.9 数据处理指令

6.9.1 MOVE

此运算符用于将一个变量的值分配到另一个相同类型的变量中。

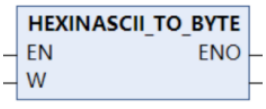
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
MOVE	赋值指令	FC		a2:=MOVE(a1);

6.9.2 HEXinASCII_TO_BYTE

当指令触发时，将源数据的 HEXinASCII 转换成字节类型的数据。

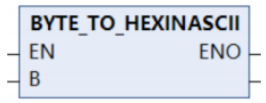
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
HEXinASCII_TO_BYTE	HEXinASCII 转字节	FC		HEXinASCII_TO_BYTE(W:=);

6.9.3 BYTE_TO_HEXinASCII

当指令触发时，将源数据字节类型的数据转换成 HEXinASCII。

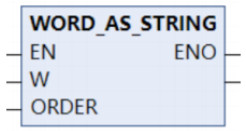
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
BYTE_TO_HEXinASCII	字节转 HEXinASCII	FC		BYTE_TO_HEXinASCII(B:=);

6.9.4 WORD_AS_STRING

当指令触发时，将源数据的 WORD 类型数据转换成 STRING 类型的数据。

指令格式：

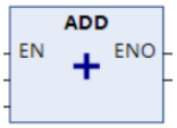
指令	名称	FB/FC	LD 表现	ST 表现
WORD_AS_STRING	数字转字符指令	FC		WORD_AS_STRING(W:=,ORDER:=);

6.10 数学运算指令

6.10.1 ADD

将左侧两个输入相加，从右侧输出加法运算结果。

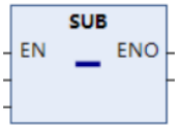
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
ADD	加法指令	FC		a1:=a2+a3;

6.10.2 SUB

将左侧两个输入相减，从右侧输出减法运算结果。

指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
SUB	减法指令	FC		a1:=a2-a3;

6.10.3 MUL

将左侧两个输入相乘，从右侧输出乘法运算结果。

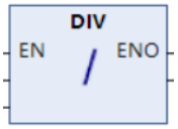
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
MUL	乘法指令	FC		a1:=a2*a3;

6.10.4 DIV

将左侧两个输入相除，从右侧输出除法运算后的商值。

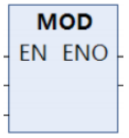
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
DIV	除法指令	FC		a1:=a2/a3;

6.10.5 MOD

将左侧两个输入相除，从右侧输出相除后非负的整数余数。

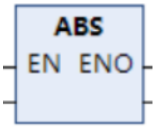
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
MOV	取余指令	FC		a1:=a2 MOD a3;

6.10.6 ABS

将输入数据取绝对值后赋值给输出变量。

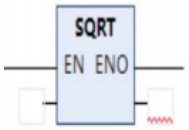
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
ABS	绝对值指令	FC		q:=ABS();

6.10.7 SQRT

将输入值进行平方根运算，输出平方根运算结果。

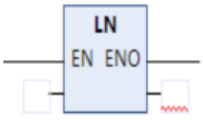
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
SQRT	计算平方根值	FC		q:=SQRT();

6.10.8 LN

将输入值进行对数运算，输出对数运算结果。

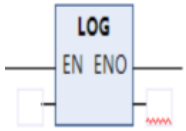
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
LN	计算自然对数 e 的对数	FC		q:=LN();

6.10.9 LOG

将输入值进行以 10 为底的对数运算，输出对数运算结果。

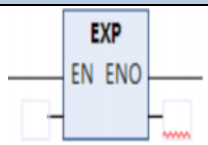
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
LOG	计算 10 的对数	FC		q:=LOG();

6.10.10 EXP

将输入值进行指数函数运算，输出函数运算结果。

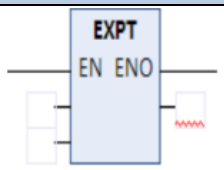
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
EXP	计算指数值	FC		q:=EXP();

6.10.11 EXPT

将输入变量 2 当作输入变量 1 的幂，输出幂运算结果。

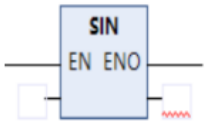
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
EXPT	计算次方值	FC		q:=EXPT();

6.10.12 SIN

将输入值进行正弦运算，输出正弦运算结果。

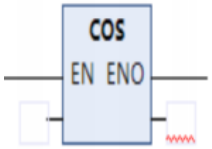
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
SIN	计算正弦值	FC		q:=SIN();

6.10.13 COS

将输入值进行余弦运算，输出余弦运算结果。

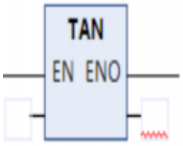
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
COS	计算余弦值	FC		q:=COS();

6.10.14 TAN

将输入值进行正切运算，输出正切运算结果。

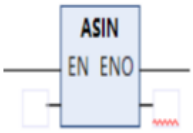
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
TAN	计算正切值	FC		q:=TAN();

6.10.15 ASIN

将输入值进行反正弦运算，输出反正弦运算结果。

指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
ASIN	计算反正弦值	FC		q:=ASIN();

6.10.16 ACOS

将输入值进行反余弦运算，输出反余弦运算结果，运算结果被计算成弧度。

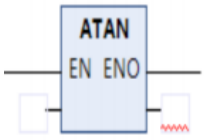
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
ACOS	计算反余弦值	FC		q:=ACOS();

6.10.17 ATAN

将输入值进行反正切运算，输出反正切运算结果，运算结果被计算成弧度。

指令格式：

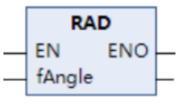
指令	名称	FB/FC	LD 表现	ST 表现
ATAN	计算反正切值	FC		q:=ATAN();

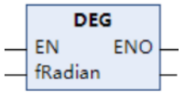
6.10.18 RAD/DEG

RAD：浮点数角度转换成弧度的运算，其计算公式为[弧度单位=角度单位×л/180]。

DEG：浮点数弧度转换成角度的运算，其计算公式为[角度单位=弧度单位×л/180]。

指令格式：

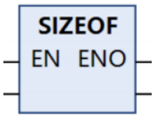
指令	名称	FB/FC	LD 表现	ST 表现
RAD	角度转换为弧度	FC		q:=RAD();

指令	名称	FB/FC	LD 表现	ST 表现
DEG	弧度转换为角度	FC		q:=DEG();

6.10.19 SIZEOF

输入值用于定义“变量”需要的字节数量。SIZEOF 运算符总是产生一个无符号值。返回变量的类型适应于“变量”长度检测。

指令格式：

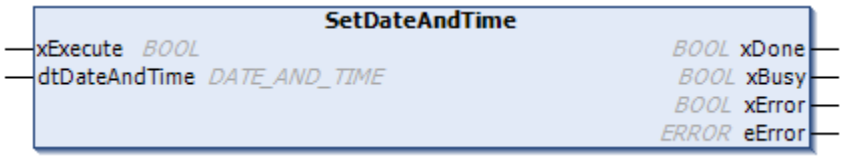
指令	名称	FB/FC	LD 表现	ST 表现
SIZEOF	数据类型大小指令	FC		SIZEOF();

6.11 日期和时间指令

6.11.1 SetDateAndTime

设置当前系统的时区、日期、时间。

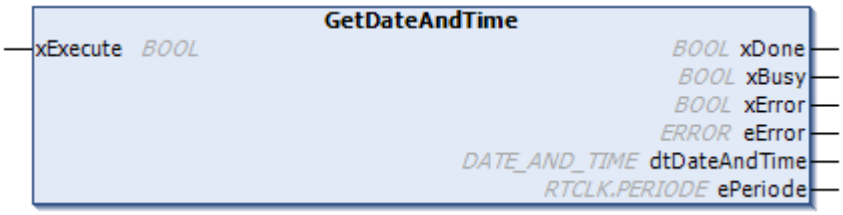
指令格式：



6.11.2 GetDateAndTime

获取当前系统的时区、日期、时间。

指令格式：



6.12 字符串功能指令

6.12.1 LEN

该函数用于获取字符串的长度。输入变量 STR 为 STRING 类型，返回值为 INT 类型。

FBD 中示例：

ST 中示例：

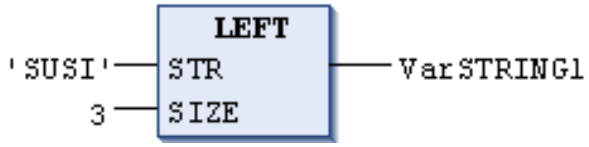
```
VarINT1:=LEN('SUSI');
```

6.12.2 LEFT

该函数功能是获取源字符串中从左边开始的部分字符。输入变量 STR 为 STRING 类型，SIZE 为 INT 类型，返回值为 STRING 类型。

LEFT(STR,SIZE)意思是：获取从字符串 STR 左边开始，长度为 SIZE 的部分字符。

FBD 中示例：



ST 中示例：

```
VarSTRING1:=LEFT('SUSI',3);
```

6.12.3 RIGHT

该函数功能是获取源字符串中从右边开始的部分字符。输入变量 STR 为 STRING 类型，SIZE 为 INT 类型，返回值为 STRING 类型。

RIGHT(STR,SIZE)意思是：获取从字符串 STR 右边开始，长度为 SIZE 的部分字符。

FBD 中示例：

ST 中示例：

```
VarSTRING1 := RIGHT ('SUSI',3);
```

6.12.4 MID

该函数用于获取源字符串中的一部分字符。输入变量 STR 为 STRING 类型，LEN 与 POS 为 INT 类型，返回值为 STRING 类型。

MID (STR, LEN, POS) 的意思是：获取字符串 STR 中从第 POS 个字符位置开始，长度为 LEN 的部分字符。

FBD 中示例：

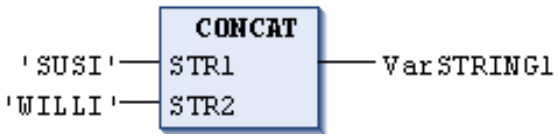
ST 中示例：

```
VarSTRING1:=MID('SUSI',2,2);
```

6.12.5 CONCAT

该函数功能是将两个字符串进行串联组合。输入变量 STR1 与 STR2 以及返回值均为 STRING 类型。

FBD 中示例：



ST 中示例：

```
VarSTRING1:=CONCAT('SUSI','WILLI');
```

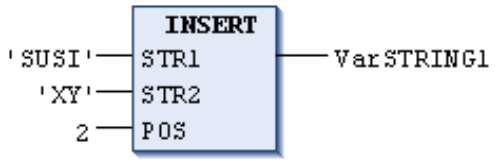
6.12.6 INSERT

该函数用于在一个字符串中所指定的位置插入另一个字符串。

输入变量 STR1 与 STR2 都为 STRING 类型，POS 为 INT 类型，返回值为 STRING 类型。

INSERT(STR1,STR2,POS)的意思是：在字符串 STR1 指定的 POS 位置之后插入字符串 STR2。

FBD 中示例：



ST 中示例：

```
VarSTRING1:=INSERT('SUSI','XY',2);
```

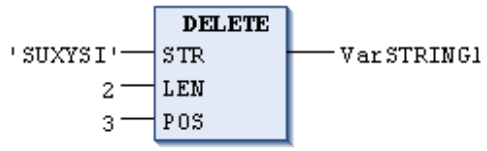
6.12.7 DELETE

该函数用于从一个字符串中的指定位置开始删除给定的部分字符。

输入变量 STR 为 STRING 类型，LEN 与 POS 为 INT 类型，返回值为 STRING 类型。

DELETE(STR,L,POS)意思是：在 STR 中删除 L 字符，在 POS 字符位置开始。

FBD 中示例：



ST 中示例：

```
Var1:=DELETE('SUXYSI',2,3);
```

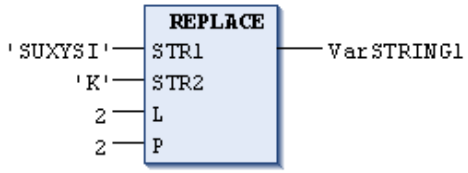
6.12.8 REPLACE

该函数用于将一个字符串中的部分字符在指定的位置替换为给定的另一个字符串。

输入变量 STR1 与 STR2 都为 STRING 类型，L 与 P 都为 INT 类型，返回值为 STRING 类型。

REPLACE(STR1,STR2,L,P)的意思是：用字符串 STR2 替换字符串 STR1 中指定的第 P 个字符位置开始，给定长度为 L 的部分字符。

FBD 中示例：



ST 中示例：

```
VarSTRING1:=REPLACE(’SUXYSI’,’K’,2,2);
```

6.12.9 FIND

该函数用于在字符串中查找部分字符。输入变量 STR1 与 STR2 均为 STRING 类型，返回值为 INT 类型。

FIND(STR1,STR2)意思是：寻找到 STR2 首次出现在 STR1 的第一个字符的位置。如果在 STR1 中没有找到 STR2，那么 OUT：=0。

FBD 中示例：

ST 中示例：

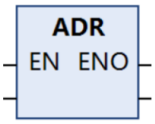
```
arINT1:=FIND('abcdef','de');
```

6.13 地址操作指令

6.13.1 ADR/^

ADR：取得输入变量的内存地址，结果赋值给输出变量，运算符，此运算符是 IEC61131-3 标准的扩展^：取得输入变量的地址内容，结果赋值给输出变量。

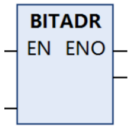
指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
ADR	取地址指令	FC		ADR();
^	取地址内容指令	FC		^

6.13.2 BITADR

取 BOOL 型变量的内存地址，结果赋值给输出变量。

指令格式：

指令	名称	FB/FC	LD 表现	ST 表现
BITADR	位地址指令	FC		BITADR();

6.14 文件操作指令

6.14.1 概述

该库主要用于 SD 与 AX7X 本地文件/文件夹的导入与导出功能，以及文件/文件夹删除与写入等操作。

其中 SD 卡路径默认：/home/root/temp/，若该路径无法使用，请尝试使用另一版本 SD 卡路径：/home/root/sdcard/。PLC 本地路径默认：/home/CODESYS/PlcLogic/。

注意：由于 PLC 内存较小，建议合理使用内存。

6.14.2 输入输出

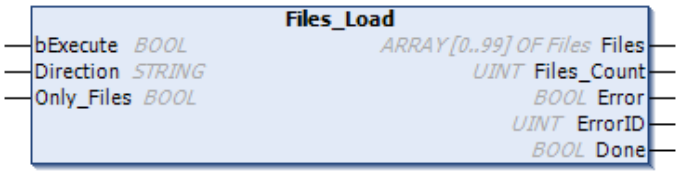
Files 结构体说明：

为了方便存储与操作文件信息，自定义 Files 结构体，该结构体与 FILE_DIR_ENTRY 结构体有所不同，新增了 Direction（文件路径）参数。

名称	数据类型	注释
Name	STRING	文件名称（包括文件后缀）
Direction	STRING	文件路径

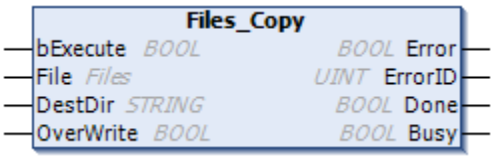
名称	数据类型	注释
isDirectory	BOOL	文件夹标志，TRUE：文件夹，FALSE：文件
Size	CAA.SIZE	文件内存大小，单位：Byte
LastModifiaction	DATE_AND_TIME	文件最终修改时间

6.14.3 files_load 文件加载函数



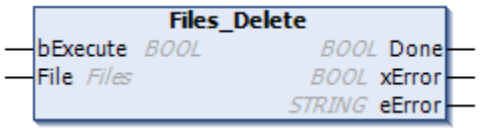
类别	名称	数据类型	初值	注释
输入	bExecute	BOOL	-	上升沿触发
	Direction	STRING	'/home/CODESYS /PlcLogic/_cnc/'	目标路径
	Only_Files	BOOL	-	TRUE：只加载文件 FALSE：文件和文件夹都加载
输出	Files	ARRAY[0…20] OF Files	-	加载的文件列表，最大 20 个
	Files_Count	UINT	-	加载到的文件/文件夹数量
	Error	BOOL	-	报警标志
	ErrorID	UINT	-	报警码
	Done	BOOL	-	加载完成标志

6.14.4 Files_Copy 文件复制函数



类别	名称	数据类型	初值	注释
输入	bExecute	BOOL	-	上升沿触发
	File	Files	-	待复制的文件
	DestDir	STRING	'/home/CODESYS /PlcLogic/_cnc/'	默认路径：PLC 本地_cnc 文件夹， SD 卡默认路径：/home/root/temp/
	OverWrite	BOOL	-	是否覆盖已有文件 TRUE：覆盖，FALSE：不覆盖
输出	Error	BOOL	-	报警标志
	ErrorID	UINT	-	报警码
	Done	BOOL	-	复制完成标志
	Busy	BOOL	-	复制中

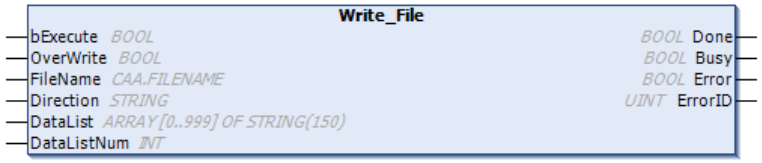
6.14.5 Delete_File 文件删除函数



类别	名称	数据类型	初值	注释
输入	bExecute	BOOL	-	上升沿触发
	File	Files	-	待删除的文件

类别	名称	数据类型	初值	注释
输出	Done	BOOL	-	删除完成标志
	xError	BOOL	-	报警标志
	eError	STRING	-	报警码

6.14.6 Write_File 文件写入函数



类别	名称	数据类型	初值	注释
输入	bExecute	BOOL	-	上升沿触发
	OverWrite	FALSE	FALSE	TRUE：覆盖文件写入 FALSE：文件未写入
输入	FileName	CAA.FILENAME	‘xx.cnc’	待写入的文件名称（包括后缀）
	Direction	STRING	'/home/CODESYS/PlcLogic/_cnc/'	文件路径
	DataList	ARRAY[0..999] OF STRING(150)	-	需写入的数据列表，按行写入，最大一次性存储 1000 条数据
	DataListNum	INT	-	DataList 数据列表数量
输出	Done	BOOL	-	写入完成标志
	Busy	BOOL	-	写入中标志
	Error	BOOL	-	报警标志
	ErrorID	UINT	-	报警码

功能块使用

ST 语言实现如下：

```
VAR
    Load:files_load;
    Load_Execite:BOOL;
    Load_Direction:STRING:=’ /home/CODESYS/PlcLogic/_cnc/’ ;
    Load_Only_File:BOOL;
    files:ARRAY[0..19] OF files;
    files_Cunt:UINT;
    Load_Error:BOOL;
    Load_ErrorID:UINT;
    Load_Done:BOOL;

    Copy:Files_Copy;
    Copy_Execute:BOOL;
    File_Index:UINT:=0;
    Copy_File:Files;
    Copy_DestDir:STRING:=’ /home/CODESYS/PlcLogic/_cnc/112/’ ;
    Copy_OverWrite:BOOL;
    Copy_Error:BOOL;
    Copy_ErrorID:UINT;
```

```
        Copy_Done:BOOL;
        Copy_Busy:BOOL;

        Delete:Files_Delete;
        Delete_Execute:BOOL;
        Delete_File:Files;
        Delete_Done:BOOL;
        Delete_Error:BOOL;
        Delete_ErrorID:UINT;

        Write:Write_File;
        Write_Execute:BOOL;
        Write_OverWrite:BOOL;
        Write_FileName:STRING;
        Write_Direction:STRING:=’/home/CODESYS/PlcLogic/_cnc/’;
        Write_DataList:ARRAY [0..999] OF STRING(150);
        Write_DataListNum:UINT;
        Write_Done:BOOL;
        Write_Busy:BOOL;
        Write_Error:BOOL;
        Write_ErrorID:UINT;
END_VAR

Load(
        bExecute:= Load_Execite,
        Direction:= Load_Direction,
        Only_Files:= Load_Only_File,
        Files=> files,
        Files_Count=> files_Cunt,
        Error=> Load_Error ,
        ErrorID=>Load_ErrorID ,
        Done=> Load_Done);
IF Load_Done THEN
        Load_Execite:=FALSE;
END_IF

Copy_File:=files[File_Index];
Copy(
        bExecute:= Copy_Execute,
        File:= Copy_File,
        DestDir:= Copy_DestDir,
        OverWrite:= Copy_OverWrite,
        Error=> Copy_Error,
        ErrorID=> Copy_ErrorID,
```

```
        Done=> Copy_Done,
        Busy=> Copy_Busy);
IF Copy_Done THEN
        Copy_Execute:=FALSE;
END_IF

//Delete_File.Direction:=Copy_DestDir;
Delete(
        bExecute:= Delete_Execute,
        File:= Delete_File,
        Done=> Delete_Done,
        Error=>Delete_Error ,
        ErrorID=> Delete_ErrorID);
IF Delete_Done THEN
        Delete_Execute:=FALSE;
END_IF

Write_DataListNum:=3;
Write_DataList[0]:=‘00’;
Write_DataList[1]:=‘111’;
Write_DataList[2]:=‘2222’;
Write_FileName:=files[File_Index].Name;
Write(
        bExecute:= Write_Execute,
        OverWrite:= Write_OverWrite,
        FileName:= Write_FileName,
        Direction:= Write_Direction,
        DataList:= Write_DataList,
        DataListNum:=Write_DataListNum,
        Done=> Write_Done,
        Busy=> Write_Busy,
        Error=> Write_Error,
        ErrorID=> Write_ErrorID);
IF Write_Done THEN
        Write_Execute:=FALSE;
END_IF
```

错误码	出错类型	解决办法
16#0000	无报警	-
16#0001	Direction 目标路径不存在	重新确认目标路径
16#0002	获取文件名称失败	确认加载的文件名称
16#0003	目标路径关闭超时	确认关闭时其他函数未对该路径进行操作
16#0004	文件/文件夹复制失败	确认文件与目标路径是否存在
16#0005	文件/文件夹删除失败	确认文件与目标路径是否存在
16#0006	文件开启失败	确认文件路径与文件名称是否正确

错误码	出错类型	解决办法
16#0007	文件写入失败	确认写入数据量，避免写入超时
16#0008	文件关闭失败	确认关闭文件时未对该文件进行其他操作
16#0009	复制的文件超过 100KB	确定待复制的大小是否超过 100kB

6.15 调节器

6.15.1 PD

该功能块为比例-微分调节功能块。

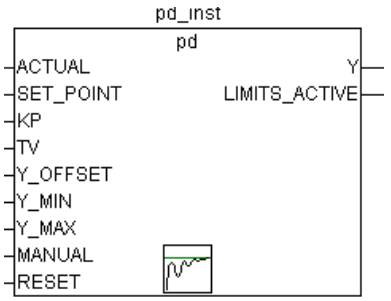
输入变量：

变量	数据类型	描述
ACTUAL	REAL	控制变量的当前值。
SET_POINT	REAL	描述值，命令值。
KP	REAL	比例系数，P-部分的比例增益。
TV	REAL	微分时间，以秒定义的 D-部分时间，例如"0.5"表示 500 秒。
Y_MANUAL	REAL	如果 MANUAL=TRUE 定义输出值 Y。
Y_OFFSET	REAL	操作值 Y 的偏移值。
Y_MIN,Y_MAX	REAL	操作值 Y 的低限制值以及高限制值。如果 Y 到达限制值，输出 LIMITS_ACTIVE 将会被设置为 TRUE 并且 Y 将会保持在制定的范围内。这个功能块只在 Y_MIN<Y_MAX 时工作。
MANUAL	BOOL	如果为 TRUE，手动操作将会被激活，那么输出值将会通过 Y_MANUAL 进行定义。
RESET	BOOL	TRUE 复位控制器；在重新初始化时 Y=Y_OFFSET。

输出变量：

变量	数据类型	描述
Y	REAL	操作值，由功能块定义（参阅下文）。
LIMITS_ACTIVE	BOOL	TRUE 表明 Y 到达给定的限位值（Y_MIN，Y_MAX）。

FBD 中示例：



Y_OFFSET，Y_MIN 以及 Y_MAX 用于在指定的范围内进行操作数的转换。

MANUAL 可以用于启动或者关闭手动操作。RESET 用于复位控制器。

正常操作时(MANUAL=RESET=LIMITS_ACTIVE=FALSE)，控制器计算偏差值 SET_POINT-ACTUAL，产生时间的相关微分 de/dt 并将其作为内部变量存储。

输出值 Y，通过以下方式获得：

$$\mathrm{Y} = \mathrm{KP} \cdot \left(\Delta + \mathrm{TV}\frac{\delta\Delta}{\delta t}\right) + \mathrm{Y_OFFSET}$$

其中Δ=SET_POINT-ACTUAL

所以除了 P-部分以及控制器当前的偏差(D-部分)都将会影响计算输出。

另外 Y 限制在 Y_MIN 和 Y_MAX 定义的范围内。如果 Y 处理时到达这个限制值，LIMITS_ACTIVE 将会变为 TRUE。

如果没有计算的限制值，Y_MIN 和 Y_MAX 必须被设置为 0。

一旦 MANUAL=TRUE，Y_MANUAL 将会被写入 Y。

一个 P 调节可以通过设置 TV=0 实现。

6.15.2 PID

该功能块为比例-积分-微分调节功能块。

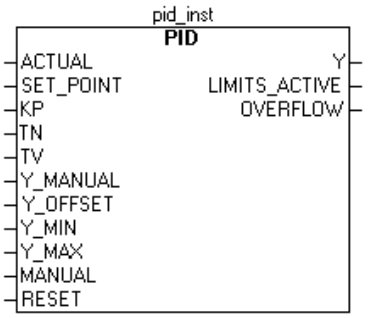
输入变量：

变量	数据类型	描述
ACTUAL	REAL	控制变量当前的值
SET_POINT	REAL	期望值，命令变量
KP	REAL	比例系数，在 P-部分的单位增益这个值不能为 0，否则功能块将不会执行任何计算
TN	REAL	重置时间，i 部分的单位增益定在几秒钟内，例如“0.5”为 500 毫秒值必须大于 0，否则功能块将不会执行任何计算；TN 越小，就会得到更多积分部分，包括操作变量的值；TN 增加越多，这种情况越少
TV	REAL	微分作用的时候，D-部分的单位增益定在几秒钟内，例如“0.5”为 500ms
Y_MANUAL	REAL	如果 MANUAL=TRUE 则定义输出值 Y
Y_OFFSET	REAL	偏移操作变量 Y
Y_MIN Y_MAX	REAL	resp 越低，操作变量 Y 的上限越高 如果 Y 超过这些限制，输出 LIMITS_ACTIVE 将会设置成真并且 Y 将会保持在规定范围内 如果 Y_MIN＜Y_MAX，这些控制才会起作用
MANUAL	BOOL	如果 TRUE，手动操作将会被激活，操作变量将会通过 Y_MANUAL 被定义
RESET	BOOL	在初始化过程 Y=Y_OFFSET 过程中，TRUE 重置控制器

输出变量：

变量	数据类型	描述
Y	REAL	操作变量值，功能块计算（参见下文）。
LIMITS_ACTIVE	BOOL	TRUE 意味着 Y 已经超出了给出的限制（Y_MIN，Y_MAX）。
OVERFLOW	BOOL	TRUE 意味着溢出（参见下文）。

FBD 中示例：



Y_OFFSET，Y_MIN 以及 Y_MAX 用于在指定的范围内进行操作数的转换。

MANUAL 可以用于启动或者关闭手动操作。RESET 用于复位控制器。

正常操作时(MANUAL=RESET=LIMITS_ACTIVE=FALSE)，控制器计算偏差值 SET_POINT-ACTUAL，产生时间的相关微分 de/dt 并将其作为内部变量存储。

输出值 Y，通过以下方式获得：

$$\mathrm{Y} = \mathrm{KP} \cdot \left(\Delta + \frac{1}{TN}\int e\mathrm{dt} + \mathrm{TV}\frac{\delta\Delta}{\delta t}\right) + \mathrm{Y_OFFSET}$$

其中Δ=SET_POINT-ACTUAL

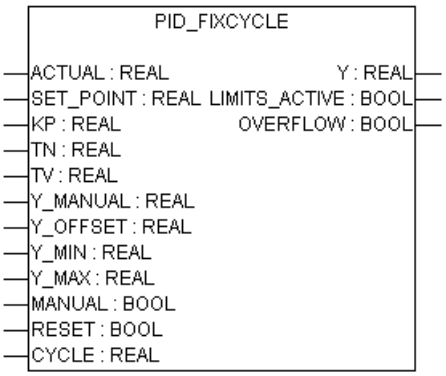
所以除了 P-部分以及控制器当前的偏差(D-部分)都将会影响计算输出。

PID 控制器可以通过设置 TV=0 轻松地转变成 PI 控制器。

因为另外的积分部分，如果错误 e 的积分部分变得更大，控制器错误的参数设置可能造成溢出。因此为了安全起见，输出调用 OVERFLOW 是可以的，在这种情况下值是 TRUE。仅由于不正确的参数设置控制系统不稳定时这种情况才会发生。在相同的时间，控制器将被暂停，将只有通过重新初始化才能再次激活。

6.15.3 PID_FIXCYCLE

FBD 中示例：



该功能模块功能和 PID 控制器一致，区别是其周期时间由 CYCLE（秒）设置，而不是通过一个内部函数自动测量。

6.16 BCD 转换指令

6.16.1 BCD_TO_INT

该函数用于将 BCD 格式下的一个字节转换为一个整数值。输入变量为 BYTE 类型，输出变量为 INT 类型。

当所要转换的字节不是 BCD 格式时，输出为-1。

ST 中示例：

```
i:=BCD_TO_INT(73);  (* Result is 49 *)
k:=BCD_TO_INT(151); (* Result is 97 *)
l:=BCD_TO_INT(15);  (* Output -1, because it is not in BCD format *)
```

6.16.2 INT_TO_BCD

该函数用于将一个整数值转换为 BCD 格式下的一个字节。输入变量为 INT 类型，输出变量为 BYTE 类型。

当所要转换的整数值不能转换为 BCD 格式时，输出为 255。

ST 中示例：

```
i:=INT_TO_BCD(49);  (* Result is 73 *)
k:=BCD_TO_INT(97);  (* Result is 151 *)
l:=BCD_TO_INT(100); (* Error! Output: 255 *)
```

6.17 系统指令

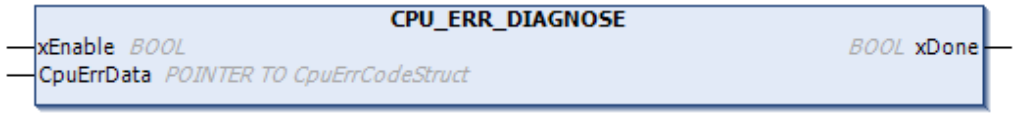
6.17.1 控制器故障诊断指令

该故障诊断指令适用 TM、TP 系列 PLC，错误代码参考 10.2PLC 故障代码表（适用 TM、TP 系列 PLC），查找错误说明。

6.17.1.1 CPU_ERR_DIAGNOSE

该功能块用于读写 CPU 故障信息。

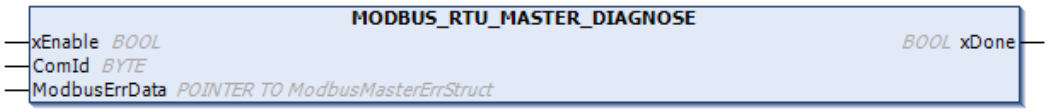
FBD 中示例：



6.17.1.2 MODBUS_RTU_MASTER_DIAGNOSE

该功能块用于读写 Modbus_RTU_Master 故障信息。

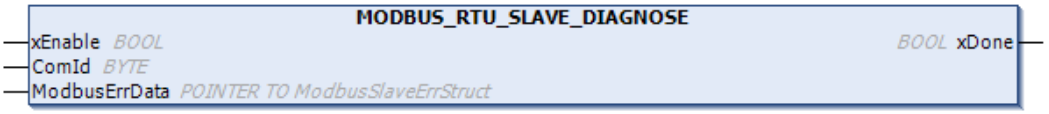
FBD 中示例：



6.17.1.3 MODBUS_RTU_SLAVE_DIAGNOSE

该功能块用于读写 Modbus_RTU_Slave 故障信息。

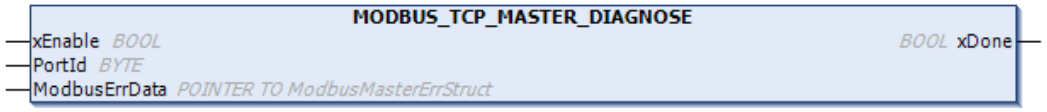
FBD 中示例：



6.17.1.4 MODBUS_TCP_MASTER_DIAGNOSE

该功能块用于读写 Modbus_TCP_Master 故障信息。

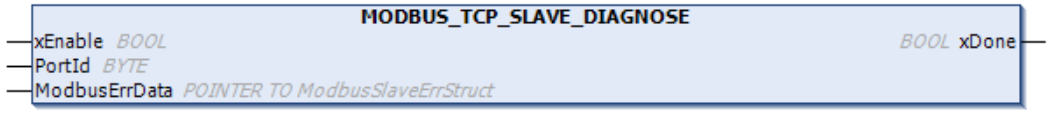
FBD 中示例：



6.17.1.5 MODBUS_TCP_SLAVE_DIAGNOSE

该功能块用于读写 Modbus_TCP_Slave 故障信息。

FBD 中示例：



ST 用法示例：

功能块的使用例程如下，用户可以根据需要自行选择使用哪个功能块。在功能块实例调用时一定要将结构体指针指向对应的错误信息结构体数组地址，该数组存放的是对应的错误诊断信息；当诊断出多个错误时，该数组能存放多个错误。数组的大小取决于用户的需要，可自行定义，但一定要大于诊断的错误数量。

```
PROGRAM PLC_PRG
VAR
    cpuerrcode : CPU_ERR_DIAGNOSE;
    ioerrcode : IO_ERR_DIAGNOSE;
    modbuserrcode : MODBUS_ERR_DIAGNOSE;

    cputmp : ARRAY[1..5] OF CpuErrCodeStruct; // CPU错误信息结构体数组
    iotmp : ARRAY[1..12] OF IoErrCodeStruct;  // IO错误信息结构体数组
    modbustmp : ARRAY[1..5] OF ModbusErrCodeStruct; // Modbus错误信息结构体数组
END_VAR
```

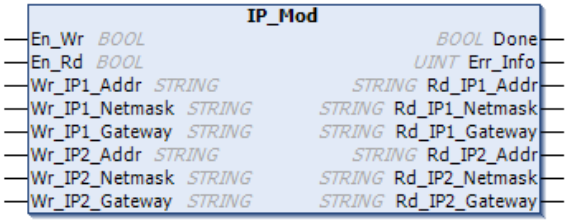
```
cpuerrcode(xEnable:= ,
           CpuErrData:= ADR(cputmp),     // 注：指针要指向错误信息结构体数组地址
           xDone=> );

ioerrcode(xEnable:= ,
          IoErrData:= ADR(iotmp),
          xDone=> );

modbuserrcode(ModbusEnable:= ,
              ModbusErrData:= ADR(modbustmp) ,
              xDone=> );
```

6.17.2 TM 控制器 IP、时间指令

6.17.2.1 IP_Mod（仅适用 TM 系列 PLC）

该功能块用于读写网络参数信息，包括 IP 地址、子网掩码、网关地址。

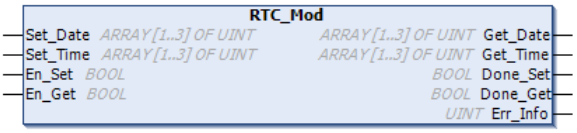
FBD 中示例：



6.17.2.2 RTC_Mod（仅适用 TM 系列 PLC）

该功能块用于读写控制器时间。

FBD 中示例：



6.17.3 TP 控制器 IP、时间指令

6.17.3.1 RTC_Mod（仅适用 TP 系列 PLC）

该功能块用于读取控制器时间。

FBD 中示例：

6.17.3.2 Sys_NetworkConfig（仅适用 TP 系列 PLC）

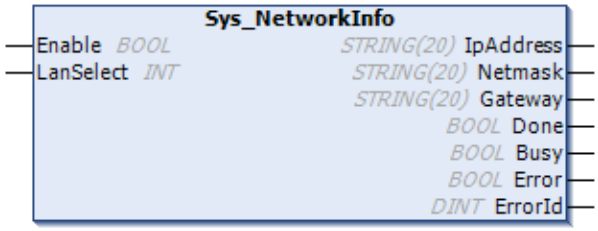
该功能块用于设置网络参数信息，包括 IP 地址、子网掩码、网关地址。

FBD 中示例：

6.17.3.3 Sys_NetworkInfo（仅适用 TP 系列 PLC）

该功能块用于读取网络参数信息，包括 IP 地址、子网掩码、网关地址。

FBD 中示例：



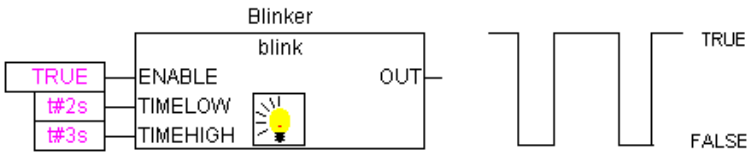
6.18 信号发生器

6.18.1 BLINK

该功能块用于产生一个脉冲信号。输入变量 ENABLE 为 BOOL 类型，TIMELOW 和 TIMEHIGH 为 TIME 类型。输出变量 OUT 为 BOOL 类型。

如果 ENABLE 的数值为 TRUE，BLINK 启动。在 TIMEHIGH 中所设定的时间周期内 OUT 为 TRUE，在 TIMELOW 中所设定的时间周期内 OUT 为 FALSE。

CFC 中示例：



6.18.2 FREQ_MEASURE

这个功能块测量布尔输入信号的（平均）频率值(Hz)。可以指定测量周期，一个周期是两个信号上升沿直接的间隔。

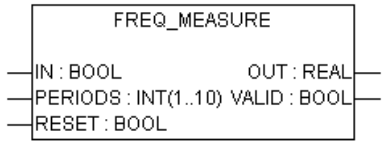
输入变量：

变量	数据类型	描述
IN	BOOL	输入信号
PERIODS	INT	周期序号，两个上升沿之间的时间间隔，通过这个计算输入信号的平均频率，可能的值：1~10
RESET	BOOL	复位所有参数到 0

输出变量：

变量	数据类型	描述
OUT	REAL	结果频率(Hz)
VALID	BOOL	FALSE 直到第一次测量结束，或者如果周期>3*OUT(表示输入由错误)

FBD 中示例：



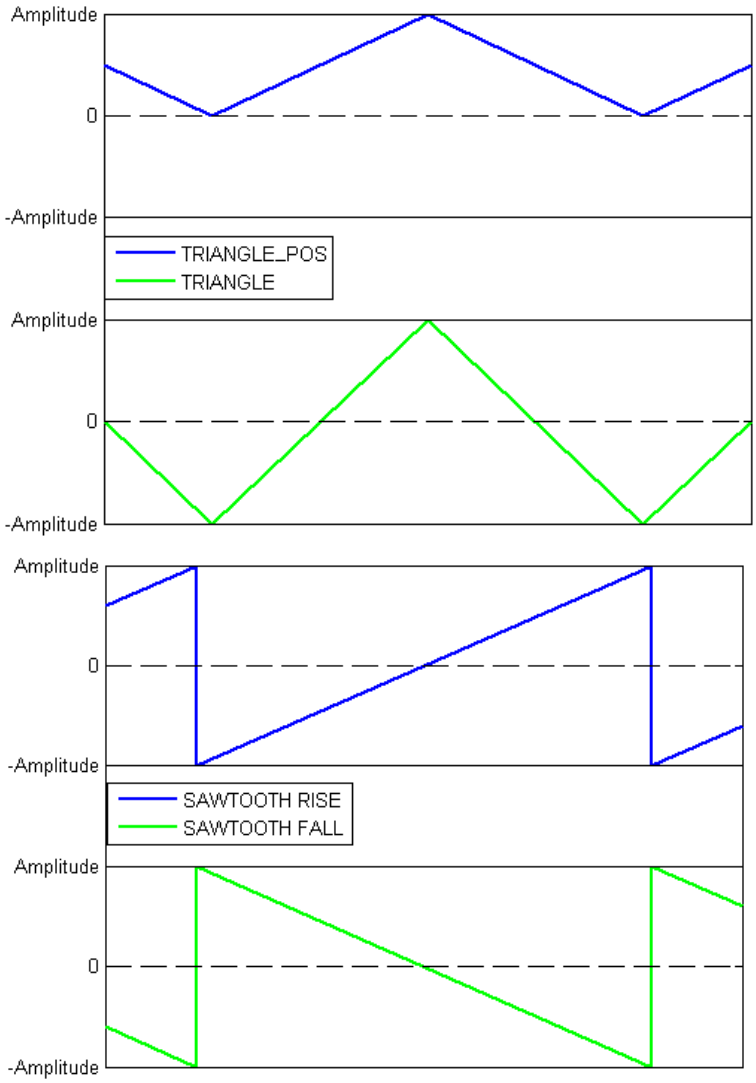
6.18.3 GEN

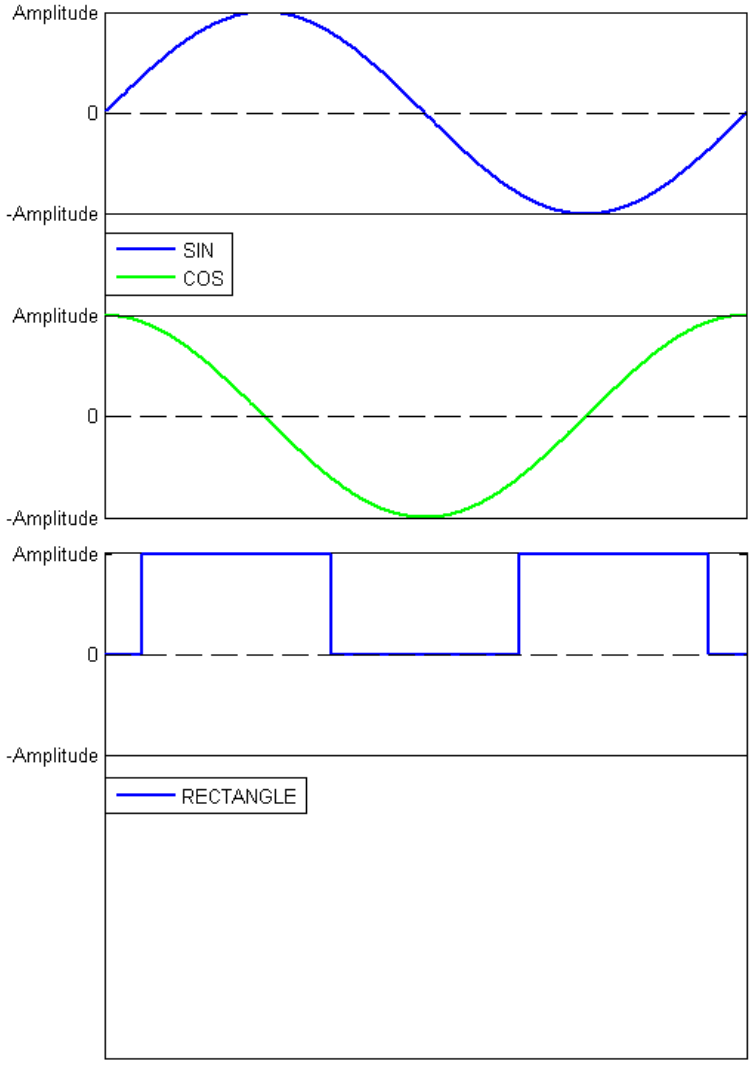
该功能块用于生成标准振荡周期。

输入变量 MODE 可以预定义 GEN_MODE 类型；BASE 为 BOOL 类型；PERIOD 为 TIME 类型；CYCLES 与 AMPLITUDE 为 INT 类型；RESET 为 BOOL 类型。

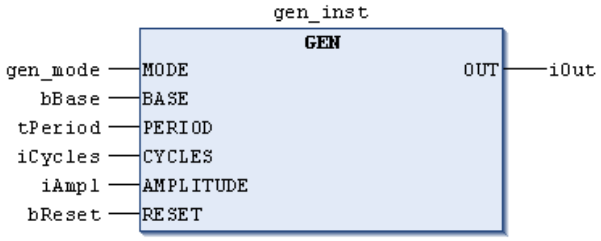
MODE 用于定义所产生的振荡周期模式。其中，枚举值 TRIANGLE 与 TRIANGLE_POS 为三角波，SAWTOOTH_RISE 为递增锯齿波，SAWTOOTH_FALL 为递减锯齿波，RECTANGLE 为方波，SINUS 与 COSINUS 分别为正弦波与余弦波。

BASE 用于定义是否使用所设置的时间定义循环周期（BASE=TRUE）或者使用表示调用功能块次数的特定周期数值定义循环周期（BASE=FALSE）。PERIOD 或 CYCLES 用于定义相应的循环周期。AMPLITUDE 用于定义所产生的振幅。当 RESET=TRUE，信号发生器复位为 0。





CFC 中示例：



6.19 数学辅助功能块

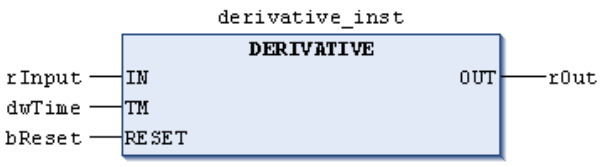
6.19.1 DERIVATIVE

该功能块用于确定局部近似导数。

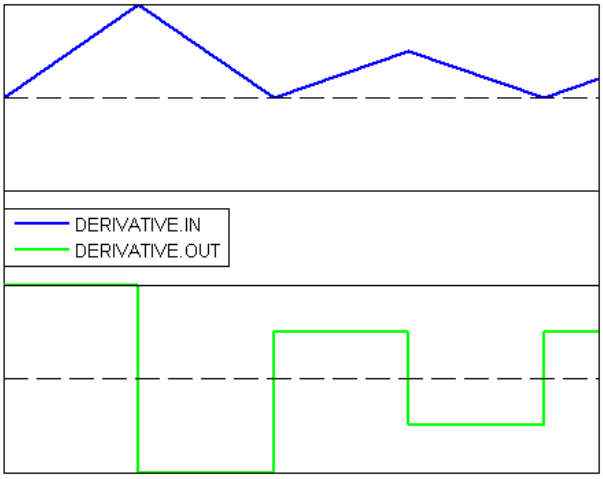
输入变量 IN 为 REAL 类型，TM 为 DWORD 类型，以毫秒为单位表示时间；RESET 为 BOOL 类型，其值为 TRUE 时，功能块复位。输出变量 OUT 为 REAL 类型。

为了取得最精确的结果，DERIVATIVE 近似地使用最后 4 个数值，以便输入参数中所产生的不精确性尽可能降低。

FBD 中示例：



DERIVATIVE 输入输出：



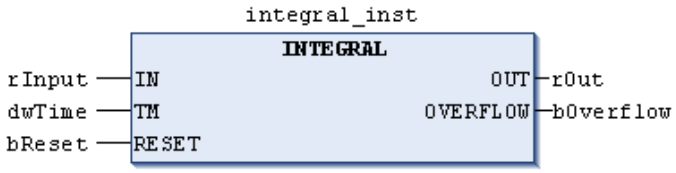
6.19.2 INTEGRAL

该功能块用于近似积分。

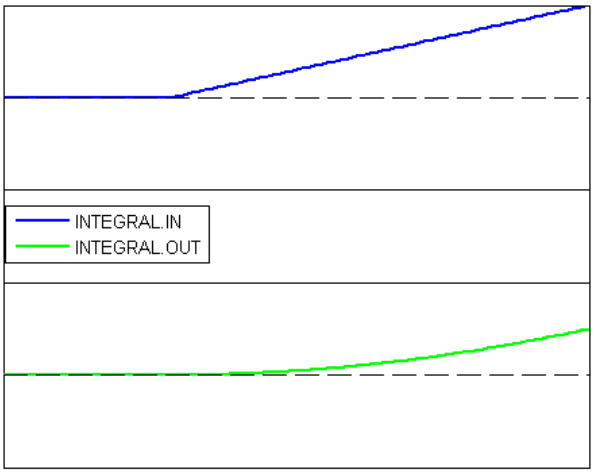
与 DERIVATIVE 类似，输入变量 IN 为 REAL 类型，TM 为 DWORD 类型，以毫秒为单位表示时间；RESET 为 BOOL 类型，当该值为 TRUE 时，功能块复位。输出变量 OUT 为 REAL 类型。

积分为两个阶跃函数的近似值，数据的平均值为近似积分。

FBD 中示例：



INTEGRAL 输入输出：

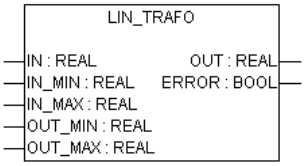


6.19.3 LIN_TRAFO

该功能块将由上限值和下限值确定的范围内的实数，转换为由另外的上限值和下限值确定的范围内的实数。

以下表达式基于这种变换：

(IN-IN_MIN):(IN_MAX-IN)=(OUT-OUT_MIN):(OUT_MAX-OUT)



输入变量：

变量	数据类型	描述
IN	REAL	输入变量
IN_MIN	REAL	变量范围低值
IN_MAX	REAL	变量范围高值
OUT_MIN	REAL	输出范围的低值
OUT_MAX	REAL	输出范围的高值

输出变量：

变量	数据类型	描述
OUT	REAL	输出值
ERROR	BOOL	错误发生：TRUE，如果 IN_MIN=IN_MAX，或者 IN 超出指定的输入范围

应用示例：

一个温度是由电压值提供的(输入 IN)。然而这些温度值将会被转化为温度值(输出 OUT)。输入(电压)值的范围是通过 IN_MIN=0 和 IN_MAX=10 进行限制。输出(温度值)值的范围是通过 OUT_MIN=-20 和 OUT_MAX=40 进行限制的。因此对应一个 5V 电压值将会产生一个 10°C的温度输出。

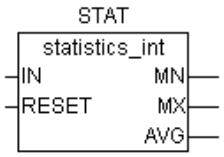
6.19.4 STATISTICS_INT

该功能块用于计算一些标准统计数值。

输入变量 IN 为 INT 类型。当 BOOL 类型的输入变量 RESET 为 TRUE 时，所有数值重新初始化。

输出变量 MN 为 IN 的最小值，MX 为 IN 的最大值，AVG 为平均值。三个输出变量均为 INT 类型。

FBD 中示例:



6.19.5 STATISTICS_REAL

该功能块和 STATISTICS_INT 相似，只是输入变量 IN，输出变量 MN、MX、AVG 均为 REAL 类型。

6.19.6 VARIANCE

VARIANCE 用于计算输入数值的方差。

输入变量 IN 为 REAL 类型，RESET 为 BOOL 类型，输出变量 OUT 为 REAL 类型。

这个功能块用于计算输入数值的方差。当 RESET=TRUE 时，VARIANCE 将被复位。

通过计算方差平方根，可以很容易地得到标准偏差。

6.20 操作功能块

6.20.1 CHARCURVE

该功能块用于以折线的形式，表示数值。

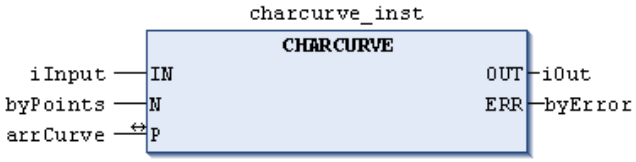
输入 IN 为 INT 类型，用于设置进行处理数值；N 为 BYTE 类型，用于设置点的数量。P 为基于两个整数值（X 与 Y）的预定义 POINT 类型，数组 P[0...10]用于产生特征线。

输出变量 OUT 为 INT 类型，用于输出已处理的数据；ERR 为 BYTE 类型，用于显示错误。

数组中的点 P[0]...P[N-1]必须依照它们的 X 值的大小来存储，否则 ERR 返回值 1。如果输入 IN 的数值不在 P[0].X 与 P[N-1].X 之间，ERR=2 ，并且 OUT 为相应的限值 P[0].Y 或 P[N-1].Y。

如果 N 的数值在允许值 2~11 之外，那么 ERR=4。

FBD 中示例：



ST 中示例：

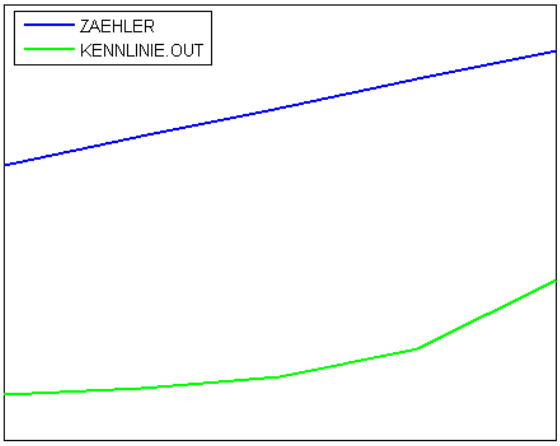
首先定义数组 P

```
VAR
...
CHARACTERISTIC_LINE:CHARCURVE;
KL:ARRAY[0..10]OFPOINT:=[(X:=0,Y:=0),(X:=250,Y:=50),
(X:=500,Y:=150),(X:=750,Y:=400),7((X:=1000,Y:=1000))];
COUNTER:INT;
...
END_VAR
```

然后定义 CHARCURVE 一个增加值例如：

```
COUNTER:=COUNTER+10;
CHARACTERISTIC_LINE(IN:=COUNTER,N:=5,P:=KL);
```

跟踪显示效果：



6.20.2 RAMP_INT

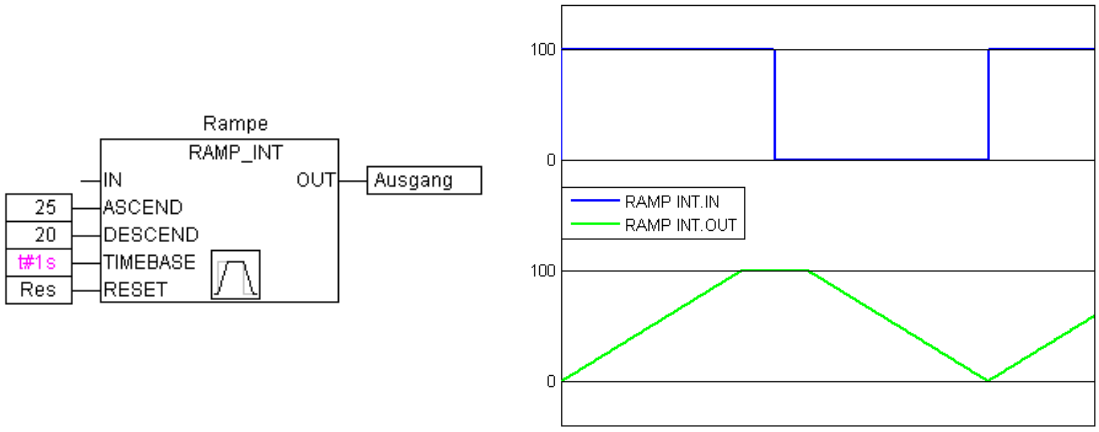
该功能块用于限制输入数值的上升率或下降率。

输入变量 IN，ASCEND，DESCEND 为 INT 类型：IN 为输入值，ASCEND 与 DESCEND 分别为给定时间内的最大增量值与减量值。TIMEBASE 为 TIME 类型，用于设置给定时间。当 RESET 的数值为 TRUE 时，RAMP_INT 将重新初始化。

输出变量 OUT 为 INT 类型，包含已被限制上升率和下降率的数值。

当 TIMEBASE 的数值为 t#0s 时，输出 OUT 与 ASCEND 和 DESCEND 无关，与 IN 保持相同。

CFC 中示例：



6.20.3 RAMP_REAL

RAMP_REAL 与 RAMP_INT 功能相似，两者之间的区别是 RAMP_REAL 的输入 IN，ASCEND，DESCEND 和输出 OUT 是 REAL 类型。

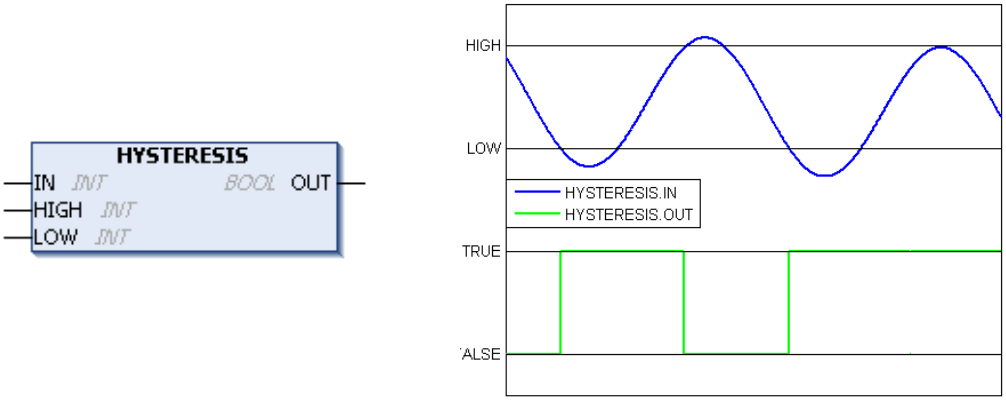
6.21 模拟值处理

6.21.1 HYSTERESIS

这个功能块的输入包含三个 INT 变量 IN，HIGH 和 LOW。输出 OUT 的类型为 BOOL。

如果 IN 低于下限值 LOW，OUT 的数值为 TRUE。如果 IN 高于上限值 HIGH，OUT 的值为 FALSE。

FBD 中示例：



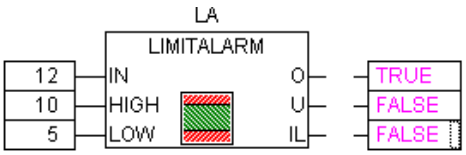
6.21.2 LIMITALARM

该功能块用于检查输入数值处于哪个范围内。

输入变量 IN、HIGH 和 LOW 均为 INT 型。输出变量 O、U 与 IL 均为 BOOL 型。

如果 IN 到达输入上界 HIGH，O 将会被置为 TRUE，当 IN 位 LOW 之下时，U 将会被置为 TRUE。如果 IN 位于 LOW 和 HIGH 之间 IL 将会被置为 TRUE。

FBD 中示例：



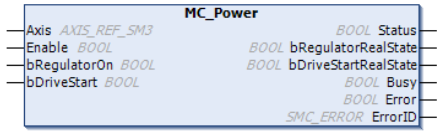
7 运动控制指令

7.1 单轴指令

7.1.1 MC_Power

MC_Power：将伺服驱动器切换为使能状态。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_Power	轴使能		MC_Power(Axis:=, Enable:=, bRegulatorOn:=, bDriveStart:=, Status=>, bRegulatorRealState=>, bDriveStartRealState=>, Busy=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Enable	执行条件	BOOL	TRUE,FALSE	FALSE	输入的一个上升沿将启动功能块的处理
bRegulatorOn	执行条件	BOOL	TRUE,FALSE	FALSE	输入高电平信号 TRUE，激活轴的使能状态
bDriveStart	执行条件	BOOL	TRUE,FALSE	FALSE	输入高电平信号 TRUE

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Status	使能	BOOL	TRUE,FALSE	FALSE	进入使能状态时变为 TRUE
bRegulatorRealState	使能有效	BOOL	TRUE,FALSE	FALSE	bRegulatorOn 置为 TRUE 后该位置为 TRUE
bDriveStartRealState	驱动有效	BOOL	TRUE,FALSE	FALSE	bDriveStart 置为 TRUE 后该位置为 TRUE
Busy	执行中	BOOL	TRUE,FALSE	FALSE	接受指令后变为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	发生异常时变为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

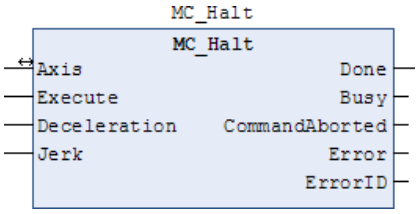
3、 功能说明

将 Enable（有效）设为 TRUE 时，Axis（轴）指定的轴进入可运行状态。将轴设为可运行状态，可实现轴控制。将 Enable（有效）设为 FALSE，可解除 Axis（轴）指定轴的可运行状态。解除可运行状态后，轴不接收动作指令，无法实现轴控制。并且，解除可运行状态的轴相应的动作指令表现为异常。但是，即使是解除状态也可以执行 MC_Power（可运行）指令、MC_Reset（轴错误复位）指令。

7.1.2 MC_Halt

MC_Halt：停止指定轴的运动。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_Halt	轴正常停止命令		MC_Halt(Axis:=, Execute:=, Deceleration:=, Jerk:=, Done=>, Busy=>, CommandAborted=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行条件	BOOL	TRUE,FALSE	FALSE	输入的一个上升沿将启动功能块的处理
Deceleration	减速度	LREAL	正数或 0	0	功能块的减速度(μ/S2)
Jerk	执行条件	LREAL	正数或 0	0	指定跃度［指令单位/S3］

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	指令执行完成	BOOL	TRUE,FALSE	FALSE	轴指令执行完成，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
CommandAborted	指令被中断	BOOL	TRUE,FALSE	FALSE	当前指令被中断，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

本功能块启动可以停止一个轴的运动，而在这操作后，其它一个运动轴指令再启动运行时可以终止功能块的运行。

本功能块在轴为运行状态下才能运行，其他状态下启动指令无效。

本功能块在输入变量执行条件的上升沿启动。

本功能块执行过程中轴的状态为离散运动状态，功能运行完成后轴状态变为停机静止状态。

7.1.3 MC_Home

MC_Home：确定轴的零点位置。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_Home	轴回零指令		MC_Home(Axis:=, Execute:=, Position:=, Done=>, Busy=>, CommandAborted=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行条件	BOOL	TRUE,FALSE	FALSE	输入的一个上升沿将启动功能块的处理
Position	轴到达位置	LREAL	数据范围	0	代表轴的回零位置

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	指令执行完成	BOOL	TRUE,FALSE	FALSE	轴指令执行完成，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
CommandAborted	指令被中断	BOOL	TRUE,FALSE	FALSE	当前指令被中断，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

本功能块为回零操作，启动指令里输入变量执行条件的上升沿触发，输入变量轴到达的位置为零点位置。本功能块只有轴在停机静止状态才能运行，且在执行指令前需要设置伺服回零模式，指令运行时运动过程中轴的状态为回零状态。

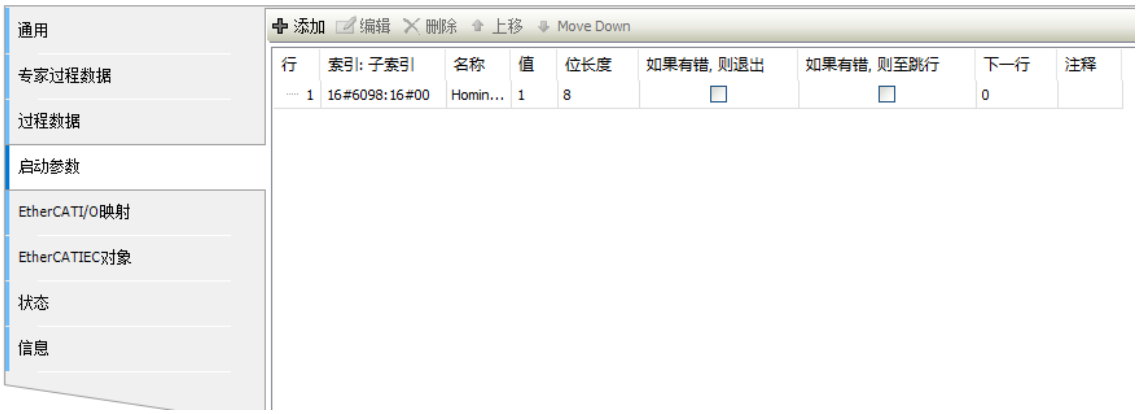
回零模式设置方法主要有两种：

- 通过手动设置伺服功能码设置回零模式。在英威腾伺服 DA200 设置 P5.10 来设置回零模式。
- 通过 AX 系列从站的启动参数设置相应的回零模式，采用通讯的方式必须设置如下索引和子索引的数据。

项目	索引	子索引	描述
回零方式	0x6098	-	根据伺服手册可以选择设定的具体参数
找原点速度	0x6099	0x01	一般定义的速度相对比较高，减少归零时间
找零点速度	0x6099	0x02	一般定义的速度相对比较低

项目	索引	子索引	描述
原点回归加减速	0x609A	-	在原点回归时的加减速变化
原点回归超时时间	0x2005	0x24	回归时间超过设定时间，系统报Err.601错误

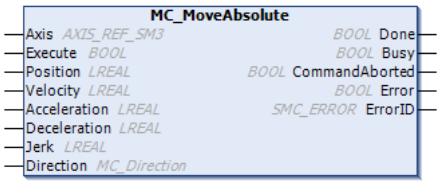
AX 系列相应的设置界面参考如下：



7.1.4 MC_MoveAbsolute

MC_MoveAbsolute：指定绝对坐标的目标位置，进行定位。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_ MoveAbsolute	轴绝对位置控制指令		MC_MoveAbsolute(Axis:=, Execute:=, Position:=, Velocity:=, Acceleration:=, Deceleration:=, Jerk:=, Direction:=, Done=>, Busy=>, CommandAborted=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行条件	BOOL	TRUE,FALSE	FALSE	输入的一个上升沿将启动功能块的处理
Position	轴到达位置	LREAL	数据范围	0	代表轴的绝对位置数据
Velocity	运行速度	LREAL	数据范围	0	轴运行到目标位置的最大速度
Acceleration	加速度	LREAL	数据范围	0	速度变大时加速度值
Deceleration	减速度	LREAL	数据范围	0	速度变小时减速度值

输入变量	名称	数据类型	有效范围	初始值	描述
Jerk	跃度	LREAL	数据范围	0	曲线加减速的斜率变化值
Direction	指令极性	MC_DIRECTION	Negative，Shortest，Positive，Current，Fastest	Shortest	Negative：反向移动； Shortest：根据最短路径选择方向； Positive：正向移动； Current：按当前方向移动； Fastest：自动选择最快的方向移动。 **注意：**此功能轴在旋转模式下有效。

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	指令执行完成	BOOL	TRUE,FALSE	FALSE	轴指令执行完成，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
CommandAborted	指令被中断	BOOL	TRUE,FALSE	FALSE	当前指令被中断，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

本功能块为轴绝对定位指令，Position 为轴的绝对位置。本功能块运行状态为 Standstill 中，当启动指令 Execute 上升沿触发后，运行状态切换为 Discrete Motion，并运动到指定 Position 位置处。当 Jerk 为 0 时，轴做梯形加减速运动；当 Velocity、Acceleration、Deceleration 和 Jerk 有数据（不为 0 的数据）时，则做 S 型曲线加减速运动。

图 7-1 梯形加减速动作

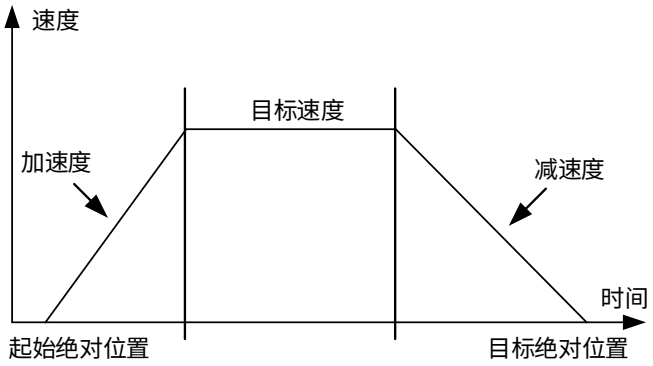
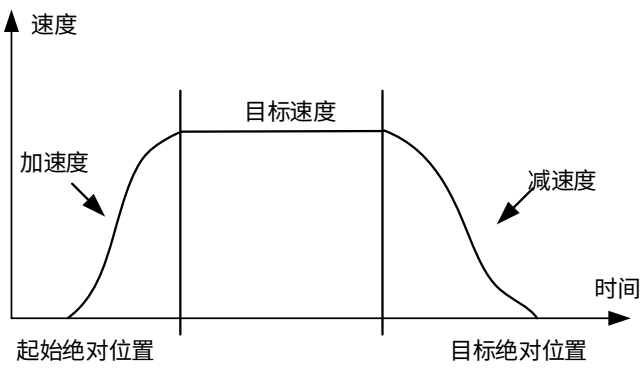
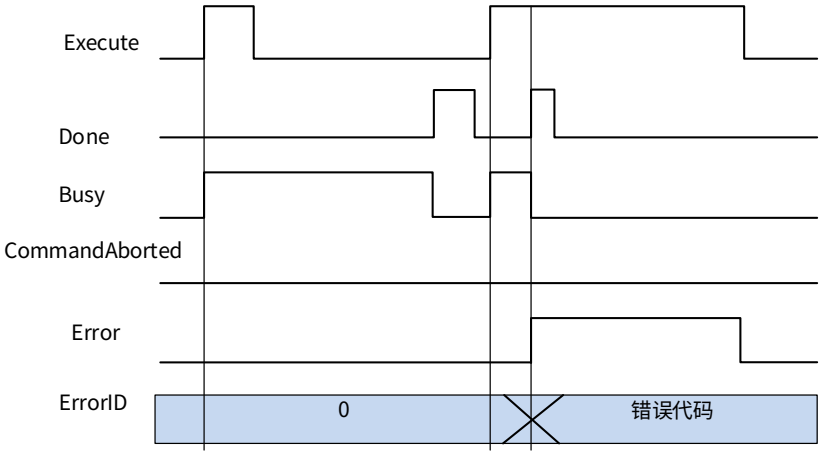


图 7-2 S 曲线加减速动作



4、 时序图

- 轴必须处于 Standstill 状态。
- 功能块的 Execute 上升沿触发。
- 功能块 Done 为 TRUE 时表示执行完成，否则 Busy 为 TRUE。



7.1.5 MC_AccelerationProfile

MC_AccelerationProfile：时间段和加减速度的轮廓运动模型

1、 指令格式

指令	名称	图形表现	ST 表现
MC_AccelerationProfile	加速度轮廓指令	MC_AccelerationProfile Axis — Done TimeAcceleration — Busy Execute — CommandAborted ArraySize — Error AccelerationScale — ErrorID Offset	MC_AccelerationProfile(Axis:=, TimeAcceleration:=, Execute:=, ArraySize:=, AccelerationScale:=, Offset:=, Done=>, Busy=>, CommandAborted=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例
TimeAcceleration	轴加速时间和加速度描述	MC_TA_REF	-	-	轴加速时间和加速度数据描述，加速数据由多组数据组成

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行条件	BOOL	TRUE,FALSE	FALSE	输入的一个上升沿将启动功能块的处理
ArraySize	动态数组	INT	数据范围	0	运行轮廓中使用的数组个数
AccelerationScale	综合因子	LREAL	正数或 0	1	MC_TA_REF 中加速度或减速度的比例因子

输入变量	名称	数据类型	有效范围	初始值	描述
Offset	偏移	LREAL	-	0	加减速度的总体偏移值

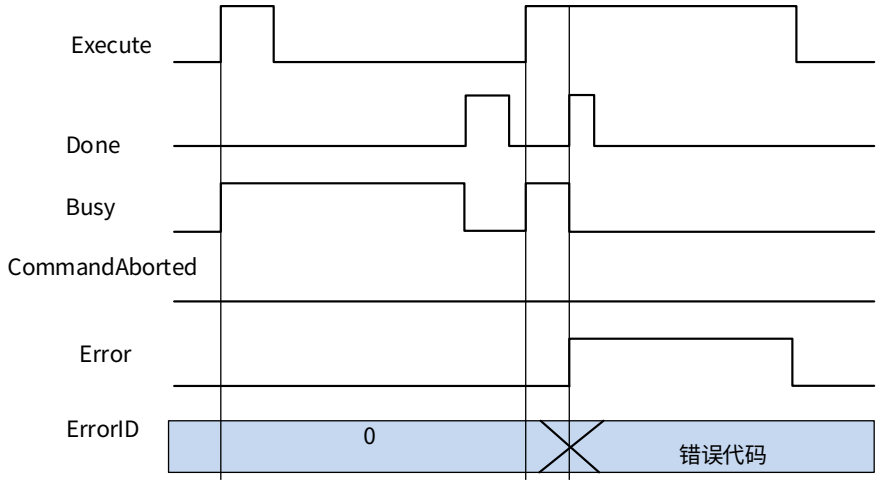
输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	指令执行完成	BOOL	TRUE,FALSE	FALSE	轴指令执行完成，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
CommandAborted	指令被中断	BOOL	TRUE,FALSE	FALSE	当前指令被中断，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

本功能块为时间段和加减速度的轮廓运动模型，运行模式为 Discrete Motion，按用户在 TimeAcceleration 变量中设定的数据运行。本功能块运行状态为 Standstill 中，指令运行时的状态为 Discrete Motion，其他状态无法运行。启动指令为 Execute 的上升沿启动，本指令在 Discrete Motion 运行速度都是在上一次的叠加，容易引起系统故障。

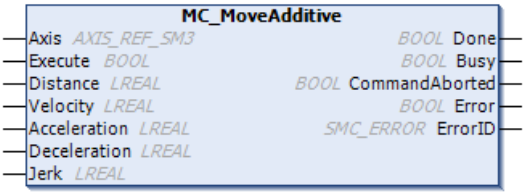
4、 时序图



7.1.6 MC_MoveAdditive

MC_MoveAdditive：轴在原来指令位置上再叠加 Distance 的距离，进行定位。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_MoveAdditive	叠加绝对运动指令		MC_MoveAdditive(Axis:=, Execute:=, Distance:=, Velocity:=, Acceleration:=, Deceleration:=, Jerk:=, Done=>, Busy=>, CommandAborted=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行条件	BOOL	TRUE,FALSE	FALSE	输入的一个上升沿将启动功能块的处理
Distance	轴到达位置	LREAL	数据范围	0	轴的叠加位置数据
Velocity	运行速度	LREAL	数据范围	0	轴运行到目标位置的最大速度
Acceleration	加速度	LREAL	数据范围	0	速度变大时加速度值
Deceleration	减速度	LREAL	数据范围	0	速度变小时减速度值
Jerk	跃度	LREAL	数据范围	0	曲线加减速的斜率变化值

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	指令执行完成	BOOL	TRUE,FALSE	FALSE	轴指令执行完成，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
CommandAborted	指令被中断	BOOL	TRUE,FALSE	FALSE	当前指令被中断，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

启动指令为 Execute，上升沿触发功能块，Distance 数据为轴的叠加数据。本功能块运行状态如果为 Discrete Motion，使用情况会把其他指令的 CommandAbort 置位。在 Standstill 状态，本指令能独立运行，实现相对定位需求。Acceleration 或 Deceleration 为 0，指令运行都为异常状态，但轴的状态为 Discrete Motion。当 Jerk 为 0 时，轴做梯形加减速运动；当 Velocity、Acceleration、Deceleration 和 Jerk 有数据（不为 0 的数据）时，则做 S 型曲线加减速运动。

图 7-3 梯形加减速动作

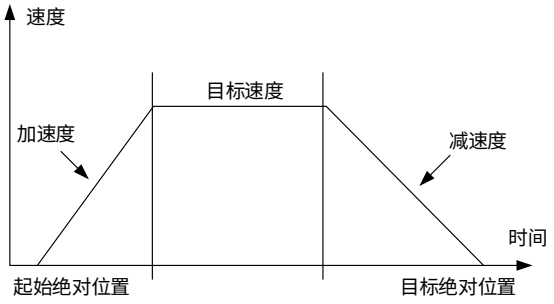
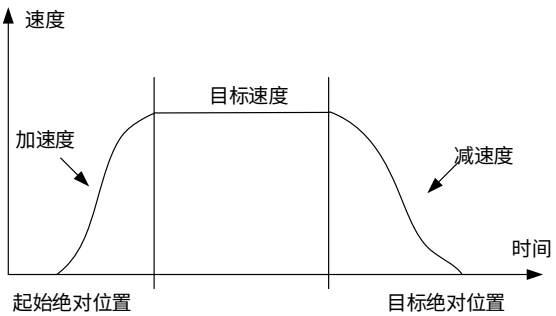
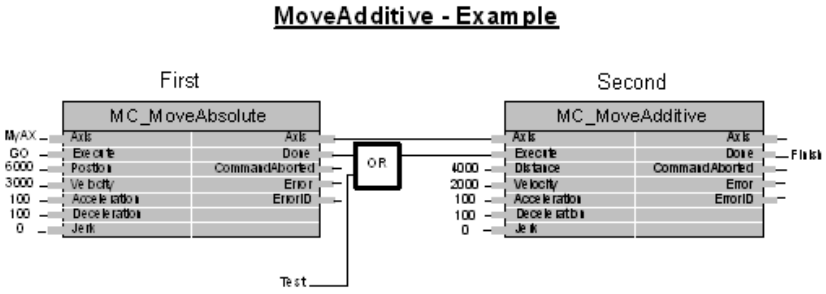


图 7-4 S 曲线加减速动作

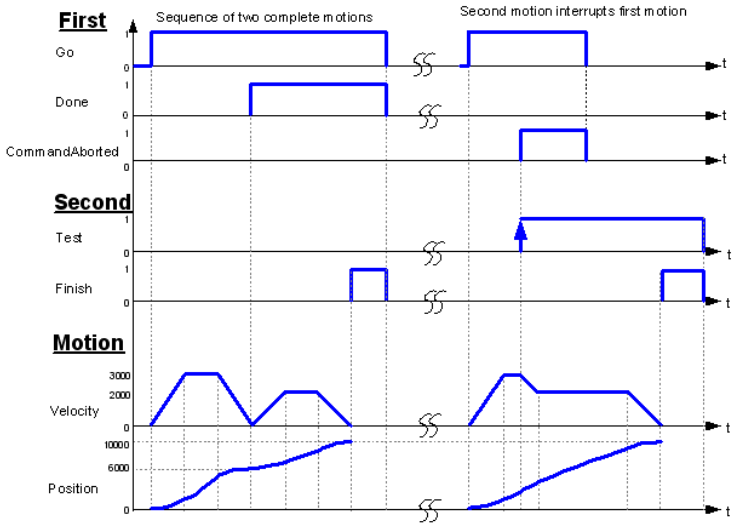


4、 时序图

举例说明



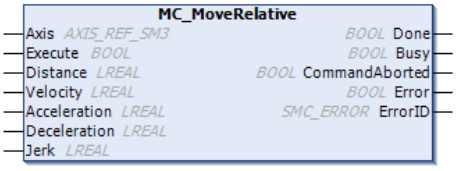
时序操作说明



7.1.7 MC_MoveRelative

MC_Move Relative：指定自指令当前位置起的移动距离，进行定位。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_MoveRelative	轴相对定位指令		MC_MoveRelative(Axis:=, Execute:=, Distance:=, Velocity:=, Acceleration:=, Deceleration:=, Jerk:=, Done=>, Busy=>, CommandAborted=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行条件	BOOL	TRUE,FALSE	FALSE	输入的一个上升沿将启动功能块的处理（FALSE→TRUE）
Distance	运动相对位置	LREAL	数据范围	0	此数据为运动的相对位置
Velocity	运行速度	LREAL	数据范围	0	轴运行到目标位置的最大速度
Acceleration	加速度	LREAL	数据范围	0	速度变大时加速度值
Deceleration	减速度	LREAL	数据范围	0	速度变小时减速度值
Jerk	跃度	LREAL	数据范围	0	曲线加减速的斜率变化值

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	指令执行完成	BOOL	TRUE,FALSE	FALSE	轴指令执行完成，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
CommandAborted	指令被中断	BOOL	TRUE,FALSE	FALSE	当前指令被中断，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

本功能块运行状态为 Standstill 中，指令运行时的状态为 Discrete Motion，在指令执行中关注本轴的运行状态，避免打断本轴的其他指令或被其他指令打断本轴的执行。启动指令为 Execute 的上升沿（FALSE→TRUE）启动，本指令在 Discrete Motion 可以重复上升沿有效，每次都可以刷新最新的 Position 位置。Acceleration 或 Deceleration 为 0，指令运行都为异常状态，但轴的状态为 Discrete Motion；

图 7-5 梯形加减速动作

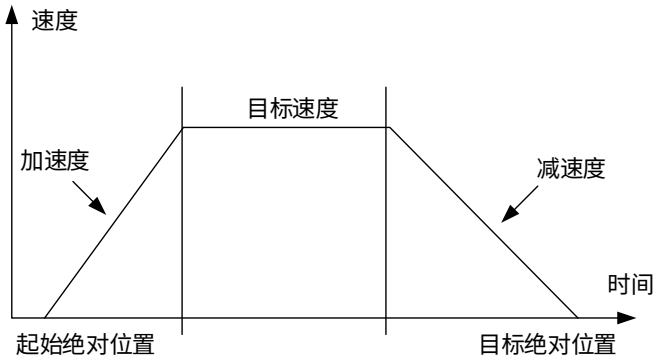
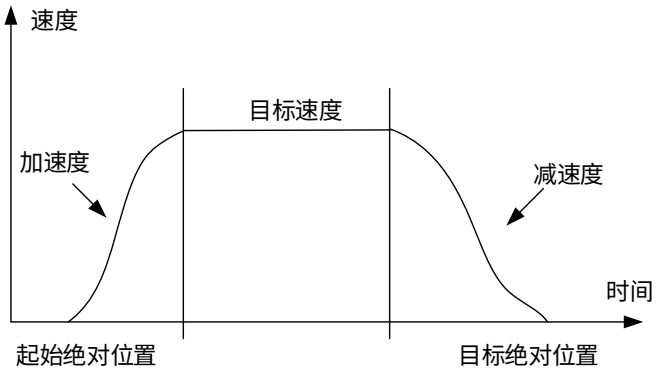
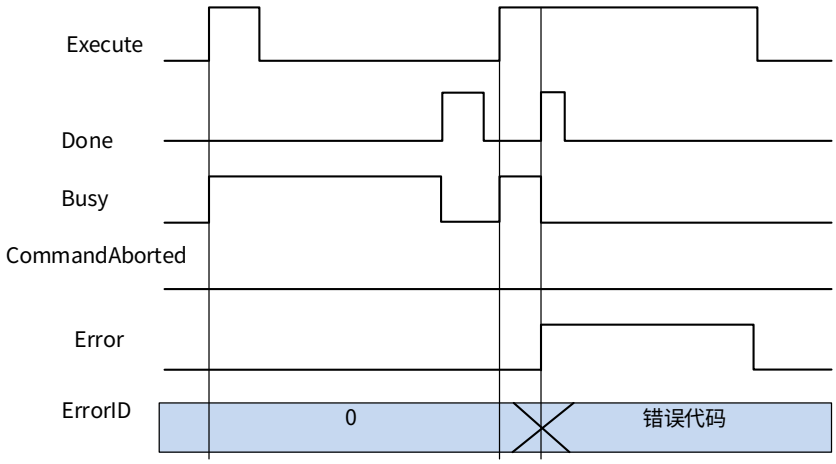


图 7-6 S 曲线加减速动作



4、 时序图



Execute 上升沿触发功能块，Busy 置位表示功能块执行中，执行完成后，置位 Done。

7.1.8 MC_MoveSuperImposed

MC_MoveSuperImposed：轴在原来指令速度和位置的基础上叠加速度和位置的数据在运行的指令上，对整个原来的指令执行时间模型上没有变化。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_MoveSuperImposed	叠加相对运动指令	MC_MoveSuperImposed Axis AXIS_REF_SM3 — BOOL Done Execute BOOL — BOOL Busy Distance LREAL — BOOL CommandAborted VelocityDiff LREAL — BOOL Error Acceleration LREAL — SMC_ERROR ErrorID Deceleration LREAL Jerk LREAL	MC_MoveSuperImposed(Axis:=, Execute:=, Distance:=, VelocityDiff:=, Acceleration:=, Deceleration:=, Jerk:=, Done=>, Busy=>, CommandAborted=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行条件	BOOL	TRUE,FALSE	FALSE	输入的一个上升沿将启动功能块的处理（FALSE→TRUE）
Distance	运动相对位置	LREAL	数据范围	0	运动相对位置
VelocityDiff	叠加速度	LREAL	数据范围	0	轴运行叠加速度
Acceleration	加速度	LREAL	数据范围	0	速度变大时加速度值
Deceleration	减速度	LREAL	数据范围	0	速度变小时减速度值

输入变量	名称	数据类型	有效范围	初始值	描述
Jerk	跃度	LREAL	数据范围	0	曲线加减速的斜率变化值

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	指令执行完成	BOOL	TRUE,FALSE	FALSE	轴指令执行完成，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
CommandAborted	指令被中断	BOOL	TRUE,FALSE	FALSE	当前指令被中断，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

本功能块为叠加位置和速度指令，Execute 上升沿触发，VelocityDiff 和 Distance 分别为叠加在其他指令上的速度和位置。在运动模式下 MC_MoveSuperImposed 可以叠加在任何其他指令。通过此指令能解决我们实际运行中由皮带和齿轮间隙误差补偿，能保证运动的一致性。指令运行时需要设置参数叠加位置。

图 7-7 梯形加减速动作

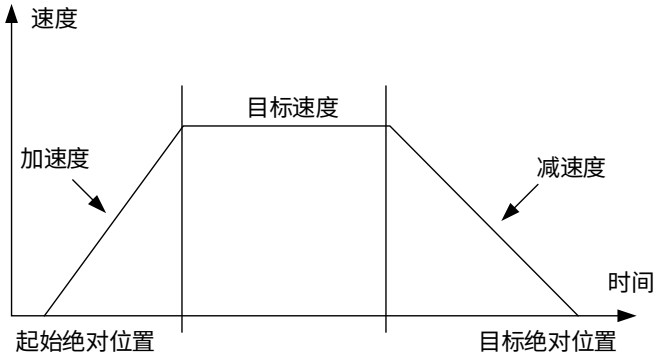
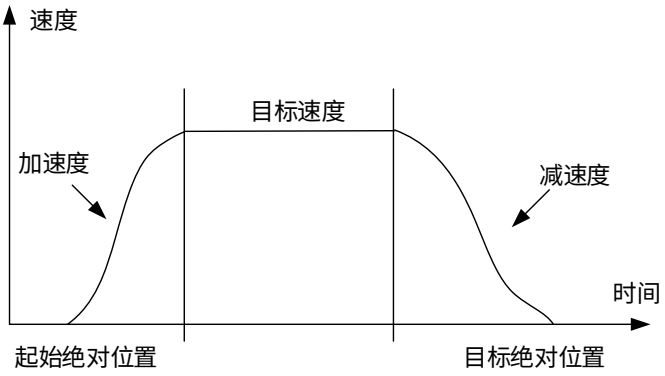
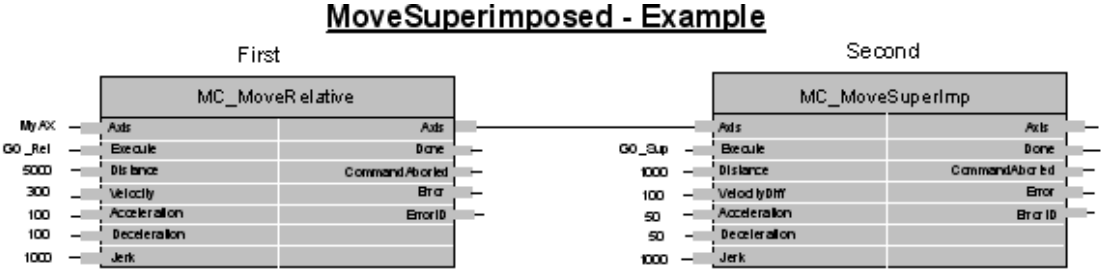


图 7-8 S 曲线加减速动作

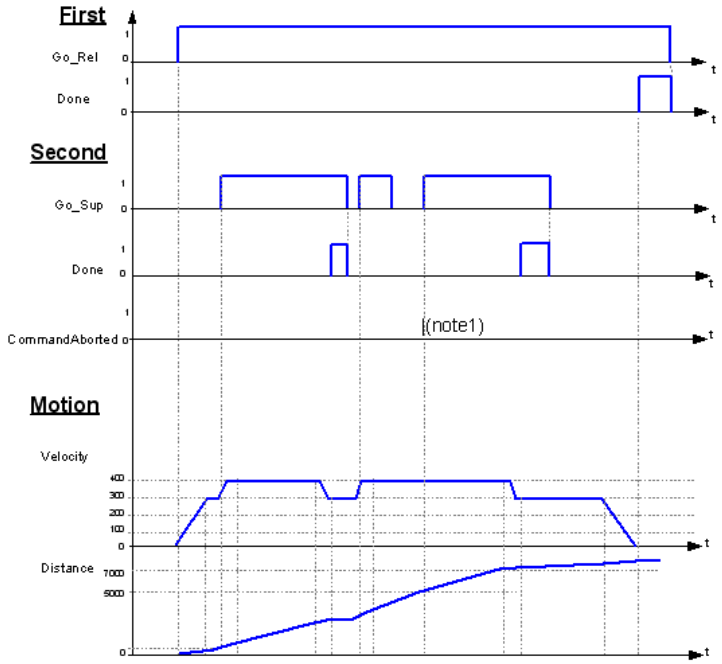


4、 时序图

举例说明



时序操作说明



7.1.9 MC_MoveVelocity

MC_MoveVelocity：使用伺服驱动器的位置控制模式，进行模拟速度控制。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_MoveVelocity	速度控制指令	MC_MoveVelocity Axis AXIS_REF_SM3 — BOOL InVelocity Execute BOOL — BOOL Busy Velocity LREAL — BOOL CommandAborted Acceleration LREAL — BOOL Error Deceleration LREAL — SMC_ERROR ErrorID Jerk LREAL Direction MC_Direction	MC_MoveVelocity(Axis:=, Execute:=, Velocity:=, Acceleration:=, Deceleration:=, Jerk:=, Direction:=, InVelocity=>, Busy=>, CommandAborted=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行条件	BOOL	TRUE,FALSE	FALSE	输入的一个上升沿将启动功能块的处理（FALSE→TRUE）
Velocity	运行速度	LREAL	数据范围	0	指定目标速度运行

输入变量	名称	数据类型	有效范围	初始值	描述
Acceleration	加速度	LREAL	数据范围	0	速度变大时加速度值
Deceleration	减速度	LREAL	数据范围	0	速度变小时减速度值
Jerk	跃度	LREAL	数据范围	0	曲线加减速的斜率变化值
Direction	运行方向	MC_Direction	Positive, Negative, Current	Current	为运行方向的指令操作

输出变量

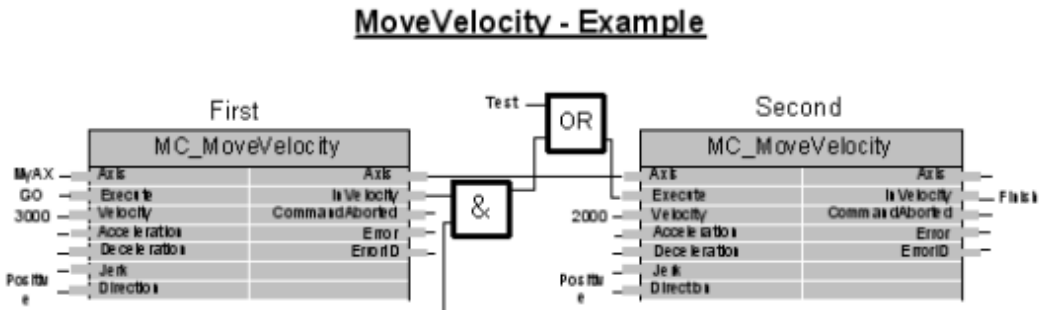
输出变量	名称	数据类型	有效范围	初始值	描述
Done	指令执行完成	BOOL	TRUE,FALSE	FALSE	轴指令执行完成，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
CommandAborted	指令被中断	BOOL	TRUE,FALSE	FALSE	当前指令被中断，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

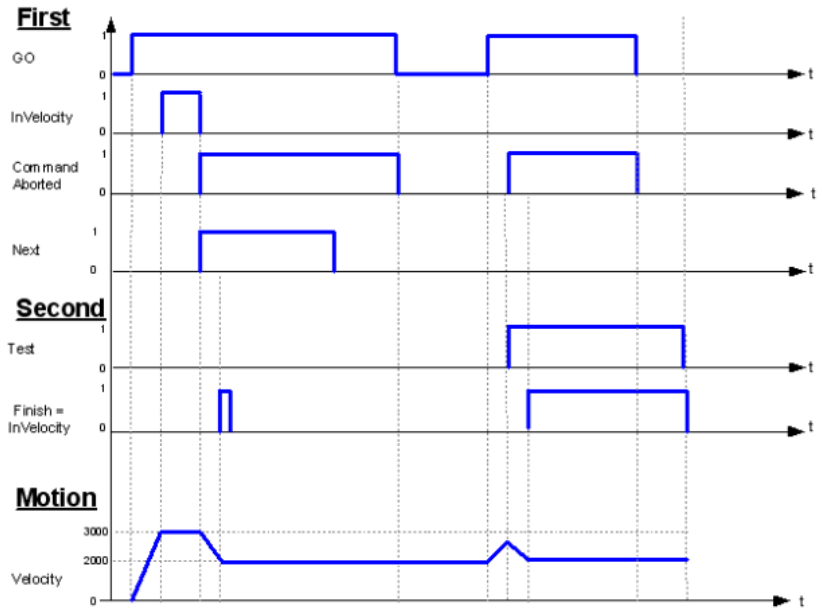
Eexcute 上升沿触发该功能块，使驱动器按照 Velocity 值进行速度控制，功能块的 InVelocity 表示指令的运行速度达到设定值。

4、 时序图

举例说明



时序操作说明



7.1.10 MC_PositionProfile

MC_PositionProfile：时间段和位置的轮廓运动模型。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_PositionProfile	位置轮廓指令		MC_PositionProfile(Axis:=, TimePosition:=, Execute:=, ArraySize:=, PositionScale:=, Offset:=, Done=>, Busy=>, CommandAborted=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例
TimePosition	轴位置运行时间和位置描述	MC_TP_REF	-	-	轴位置运行时间和位置数据描述，数据由多组数据组成

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行条件	BOOL	TRUE,FALSE	FALSE	输入的一个上升沿将启动功能块的处理（FALSE→TRUE）
ArraySize	动态数	INT	数据范围	0	运行轮廓中使用的数组个数
PositionScale	综合因子	LREAL	正数或 0	0	MC_TP_REF 中位置的比例因子
Offset	偏移	LREAL	-	0	位置的总体偏移值

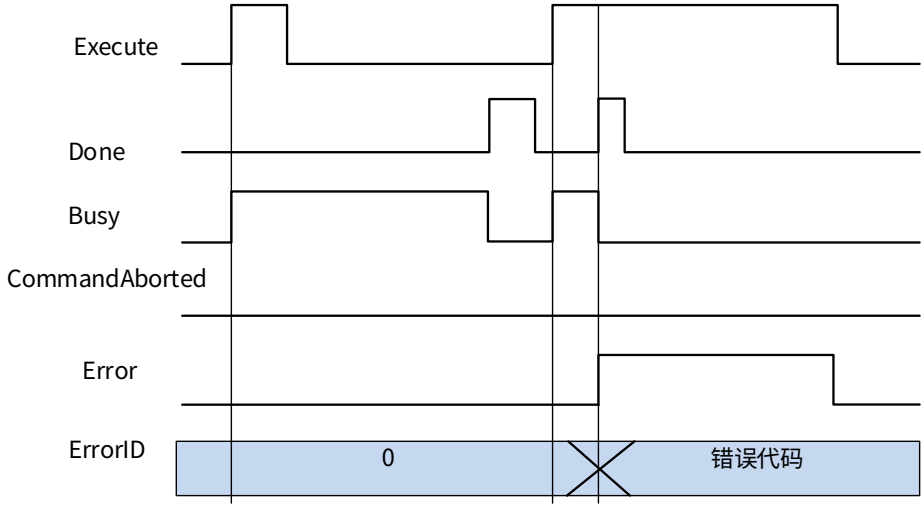
输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	指令执行完成	BOOL	TRUE,FALSE	FALSE	轴指令执行完成，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
CommandAborted	指令被中断	BOOL	TRUE,FALSE	FALSE	当前指令被中断，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

本功能块为时间段和位置的轮廓运动模型，按用户在 TimePosition 变量中设定的数据运行。本功能块运行在状态 Stanstill 中，Execute 上升沿触发功能块运行，功能块执行过程中状态为 Discrete Motion。

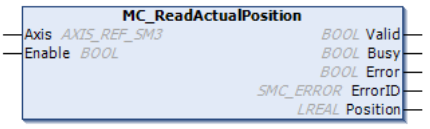
4、 时序图



7.1.11 MC_ReadActualPosition

MC_ReadActualPosition：读取驱动器运行的实际位置，并保存在自定义变量中。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_ ReadActualPosition	实际位置读取指令		MC_ReadActualPosition(Axis:=, Enable:=, Valid=>, Busy=>, Error=>, ErrorID=>, Position=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Enable	执行条件	BOOL	TRUE,FALSE	FALSE	输入的一个上升沿将启动功能块的处理（FALSE→TRUE）

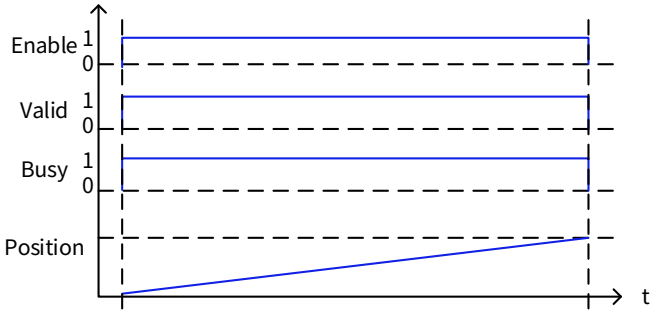
输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Valid	位置数据可获取标志	BOOL	TRUE,FALSE	FALSE	能正确的获取驱动器的位置，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码
Position	获取到的轴位置	LREAL	轴位置	0	指令读出来的轴位置数据

3、 功能说明

Eexcute 上升沿触发该功能块，通过本指令可读取轴的位置值；Valid 为 TRUE 时表示读取的位置值是有效值。该功能块可重复调用，互不影响。

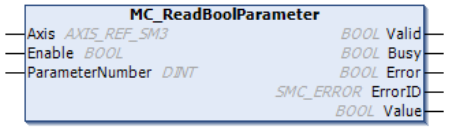
4、 时序图



7.1.12 MC_ReadBoolParameter

MC_ReadBoolParameter：指令读取驱动轴的位参数，并保存在自定义变量中。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_ReadBoolParameter	读取轴的位参数		MC_ReadBoolParameter(Axis:=, Enable:=, ParameterNumber:=, Valid=>, Busy=>, Error=>, ErrorID=>, Value=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Enable	执行条件	BOOL	TRUE,FALSE	FALSE	置为 TRUE 将启动功能块的处理
ParameterNumber	轴参数的序号	DINT	-	0	访问轴参数的索引、子索引和序号

注意：

- ParameterNumber(DINT)=-DWORD_TO_DINT(SHL(USINT_TO_DOWRD(usiDataLength),24)（对象字典中的数据长度）+SHL(UINT_TO_DWORD(uiIndex),8)(对象字典中的索引-16bit)+usisubIndex(对象字典中的子索引-8bit))。
- usiDataLength：按字节数填写；1 字节为 16#01；2 字节为 16#02；4 字节为 16#04 等。

输出变量

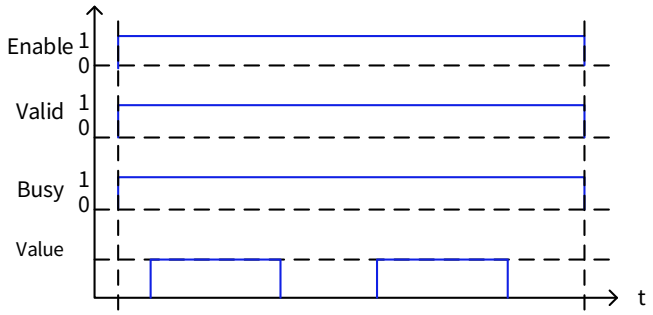
输出变量	名称	数据类型	有效范围	初始值	描述
Valid	位置数据可获取标志	BOOL	TRUE,FALSE	FALSE	能正确的获取驱动器的位置，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE

输出变量	名称	数据类型	有效范围	初始值	描述
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码
Value	获取到的轴位状态	BOOL	TRUE,FALSE	FALSE	读出 ParameterNumber 参数的值

3、 功能说明

通过 MC_ReadBoolParam 读取驱动器中的位数据状态，指令为 Enable 电平使能有效。指令可以重复多次使用，互不影响.功能块的 Valid 为 TRUE 表示读出值为有效的位状态数据；功能块的 Busy 为 TRUE 表示当前功能块正在执行中。

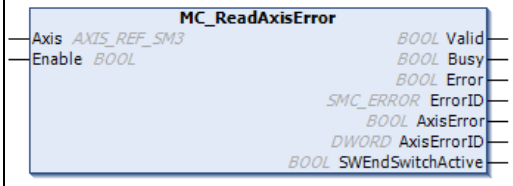
4、 时序图



7.1.13 MC_ReadAxisError

MC_ReadAxisError：读取轴的错误信息，并保存在自定义变量中。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_ReadAxisErro	读取轴的错误信息		MC_ReadAxisError(Axis:=, Enable:=, Valid=>, Busy=>, Error=>, ErrorID=>, AxisError=>, AxisErrorID=>, SWEndSwitchActive=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Enable	执行条件	BOOL	TRUE,FALSE	FALSE	置为 TRUE 将启动功能块的处理

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Valid	错误数据可获取标志	BOOL	TRUE,FALSE	FALSE	能正确的获取驱动器的位置，置为 TRUE

输出变量	名称	数据类型	有效范围	初始值	描述
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	TRUE,FALSE	FALSE	发生异常时，输出错误代码
AxisError	轴错误标示	BOOL	TRUE,FALSE	FALSE	读出轴为错误，对应的标示置位
AxisErrorID	轴错误码	DWORD	-	0	读出轴的错误码
SWEndSwitch Active	软限位开关有效	BOOL	TRUE,FALSE	FALSE	在指令读取中，检查软限位开关状态

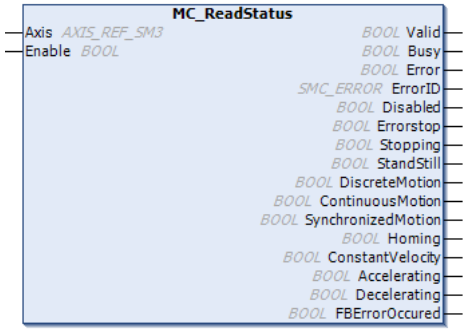
3、 功能说明

通过该功能块读取轴的错误信息，Enable 为 TRUE 有效；功能块的 Valid 为 TRUE 表示读出的 AxisError 和 AxisErrorID 为有效的数据值；功能块的 Busy 为 TRUE 表示当前功能块正在执行中；指令可重复调用互不影响。

7.1.14 MC_ReadStatus

MC_ReadStatus：指令读取轴的状态数据，并保存在自定义变量中。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_ReadStatus	读取轴的状态		MC_ReadStatus(Axis:=, Enable:=, Valid=>, Busy=>, Error=>, ErrorID=>, Disabled=>, Errorstop=>, Stopping=>, StandStill=>, DiscreteMotion=>, ContinuousMotion=>, SynchronizedMotion=>, Homing=>, ConstantVelocity=>, Accelerating=>, Decelerating=>, FBErrorOccured=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Enable	执行条件	BOOL	TRUE,FALSE	FALSE	置为 TRUE 将启动功能块的处理

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Valid	错误数据可获取标志	BOOL	TRUE,FALSE	FALSE	能正确的获取驱动器的位置，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE

输出变量	名称	数据类型	有效范围	初始值	描述
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码
Disabled	轴未使能	BOOL	TRUE,FALSE	FALSE	轴在未使能状态为 TRUE
Errorstop	轴错误状态	BOOL	TRUE,FALSE	FALSE	轴在错误运行状态为 TRUE
Stoping	轴停止过程状态	BOOL	TRUE,FALSE	FALSE	轴在停止过程中为 TRUE
StandStill	轴标准状态	BOOL	TRUE,FALSE	FALSE	轴在标准（能运行）状态中为 TRUE
DiscreteMotion	轴离散运动状态	BOOL	TRUE,FALSE	FALSE	轴在离散运动状态中为 TRUE
ContinuousMotion	轴连续运动状态	BOOL	TRUE,FALSE	FALSE	轴在连续运动状态中为 TRUE
SynchronizedMotion	轴同步运行状态	BOOL	TRUE,FALSE	FALSE	轴在同步运动状态中为 TRUE
Homing	轴回原点状态	BOOL	TRUE,FALSE	FALSE	轴在回原点状态中为 TRUE
ConstantVelocity	轴运行速度到达	BOOL	TRUE,FALSE	FALSE	轴达到运行速度中为 TRUE
Accelerating	轴加速过程状态	BOOL	TRUE,FALSE	FALSE	轴加速过程状态为 TRUE
Dccelerating	轴减速过程状态	BOOL	TRUE,FALSE	FALSE	轴减速过程状态为 TRUE
FBErrorOccured	轴功能块错误出现标志	BOOL	TRUE,FALSE	FALSE	轴功能块错误标志为 TRUE

3、 功能说明

通过 MC_ReadStatus 对应轴的各种状态，指令 Enable 为 TRUE 使能效应。指令可以重复多次使用，互不影响。功能块的 Valid 为 TRUE 表示读出的后面各状态标志的各种数据；功能块的 Busy 为 TRUE 表示当前功能块正在执行中。

7.1.15 MC_ReadParameter

MC_ReadParameter：指令读取驱动轴的参数，并保存在自定义变量中。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_ReadParameter	读取轴的参数		MC_ReadParameter(Axis:=, Enable:=, ParameterNumber:=, Valid=>, Busy=>, Error=>, ErrorID=>, Value=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Enable	执行条件	BOOL	TRUE,FALSE	FALSE	置为 TRUE 将启动功能块的处理
ParameterNumber	轴参数的序号	DINT	-	0	访问轴参数的索引、子索引和序号

注意：

- ParameterNumber(DINT)=-DWORD_TO_DINT(SHL(USINT_TO_DOWRD(usiDataLength),24)(对象字典中的数据长度)+SHL(UINT_TO_DWORD(uiIndex),8)(对象字典中的索引-16bit)+usisubIndex(对象字典中的子索引-8bit))。
- usiDataLength：按字节数填写；1 字节为 16#01；2 字节为 16#02；4 字节为 16#04 等。

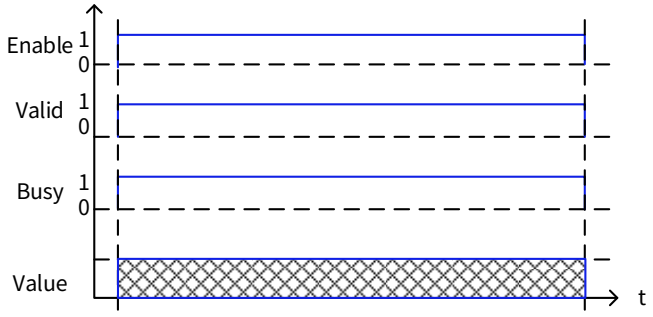
输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Valid	位置数据可获取标志	BOOL	TRUE,FALSE	FALSE	能正确的获取驱动器的位置，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码
Value	获取到的轴位状态	BOOL	TRUE,FALSE	FALSE	读出 ParameterNumber 参数的值

3、 功能说明

通过 MC_ReadBoolParameter 读取驱动器中的位数据状态，Enable 高电平有效。指令可以重复多次使用，互不影响；功能块的 Valid 为 TRUE 表示读出的 Valid 为有效的位状态数据；功能块的 Busy 为 TRUE 表示当前功能块正在执行中。

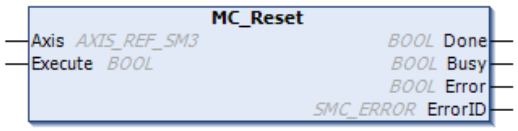
4、 时序图



7.1.16 MC_Reset

MC_Reset：指令解除轴的异常。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_Reset	轴错误状态复位指令		MC_Reset(Axis:=, Execute:=, Done=>, Busy=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行条件	BOOL	TRUE,FALSE	FALSE	上升沿将启动功能块的处理

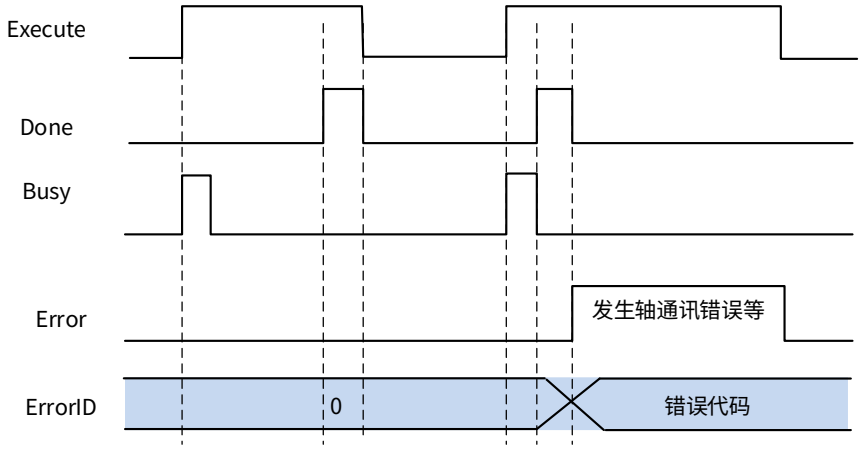
输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	指令执行完成	BOOL	TRUE,FALSE	FALSE	轴指令执行完成，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

本功能块在轴通讯正常的情况下，把轴状态处于 Errorstop 变为 Standstill,把轴的异常状态变为正常可运行的状态。如果遇到轴 Errorstop 无法复位，Axis.bCommunication 为 FLASE 状态，必须要重新建立主站和从站轴的通讯。

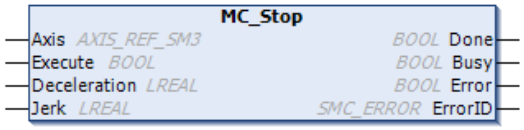
4、 时序图



7.1.17 MC_Stop

MC_Stop：指令使轴减速停止。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_Stop	轴停止命令		MC_Stop(Axis:=, Execute:=, Deceleration:=, Jerk:=, Done=>, Busy=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行条件	BOOL	TRUE,FALSE	FALSE	上升沿将启动功能块的处理
Deceleration	减速度	LREAL	正数或 0	0	功能块的减速度(μ/S2)

输入变量	名称	数据类型	有效范围	初始值	描述
Jerk	跃度	LREAL	正数或 0	0	指定跃度［指令单位/S3］

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	指令执行完成	BOOL	TRUE,FALSE	FALSE	轴指令执行完成，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

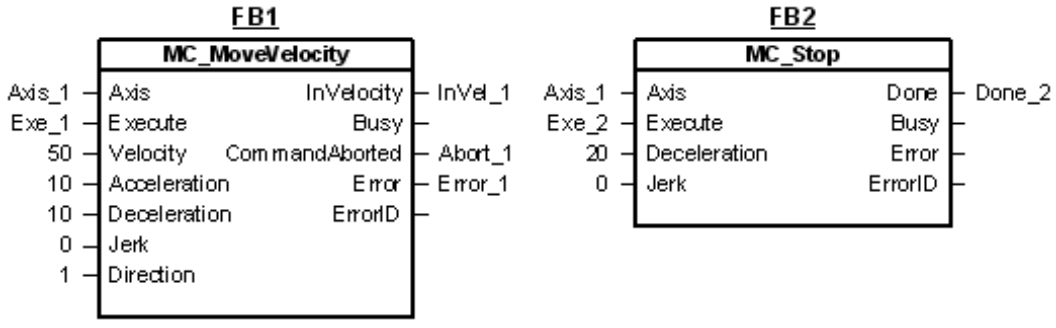
本功能块为在正常运行的情况下停止一个轴的运动，当轴处于 Stopping 状态，对本轴的任何指令都是无效的。

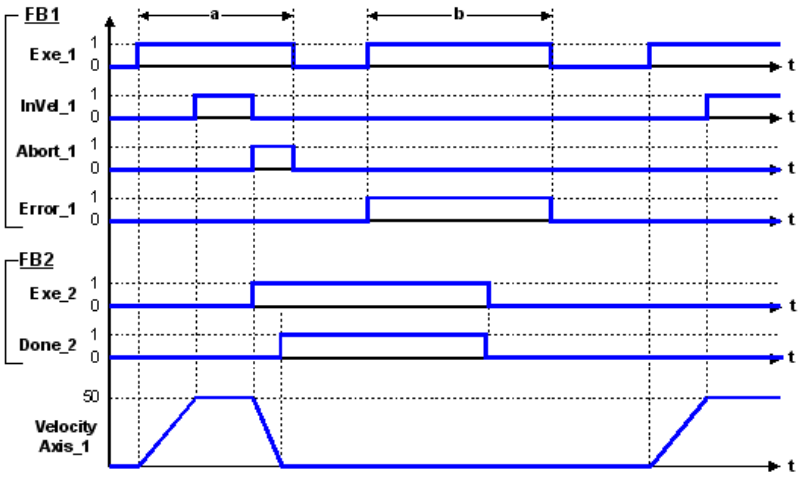
当轴状态处于 Stopping 时，Execute 为 Flase 状态，Done 输出状态为 TRUE，轴状态变为 Standstill。本功能块在 Execute 的上升沿启动。在 MC_Stop 有效执行的过程中（Busy 为 TRUE），重复启动 MC_Stop 指令，系统处于 Errorstop 状态。当启动 MC_Stop(强制停止)指令，处于运行中的指令执行 CommandAborted(执行中断)。

4、 时序图

时序操作例程如下

在执行 MC_MoveVelocity 指令和 MC_Stop 指令在不同的时序操作中对应的标志位的变化。





7.1.18 MC_VelocityProfile

MC_VelocityProfile：本功能块为时间段和速度的轮廓运动模型。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_VelocityProfile	速度轮廓指令	MC_VelocityProfile Axis AXIS_REF_SM3 TimeVelocity MC_TV_REF Execute BOOL ArraySize INT VelocityScale LREAL Offset LREAL BOOL Done BOOL Busy BOOL CommandAborted BOOL Error SMC_ERROR ErrorID	MC_VelocityProfile(Axis:=, TimeVelocity:=,

指令	名称	图形表现	ST 表现
			Execute:=, ArraySize:=, VelocityScale:=, Offset:=, Done=>, Busy=>, CommandAborted=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例
TimeVelocity	轴速度运行时间和速度描述	MC_TV_REF	-	-	轴速度运行时间和速度数据描述，由多组数据组成

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行条件	BOOL	TRUE,FALSE	FALSE	上升沿将启动功能块的处理
ArraySize	动态数组	INT	-	0	运行轮廓中使用的数组个数
VelocityScale	速度因子	LREAL	正数或 0	1	速度的比例因子
Offset	偏移	LREAL	-	0	速度值的总体偏移值

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	指令执行完成	BOOL	TRUE,FALSE	FALSE	轴指令执行完成，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

本功能块为时间段和速度的轮廓运动模型，轴的运行状态为 ContinuousMotion，按用户在 TimeVelocity 变量中设定的数据运行。本功能块轴的运行状态为 Standstill 中，指令运行时轴的状态为 DiscreteMotion，其他状态无法运行。启动指令为 Execute 的上升沿启动,本指令在 DiscreteMotion 重复运行。TimeVelocity 为 MC_TV_REF 数据类型。

MC_TV_REF 具体描述如下：

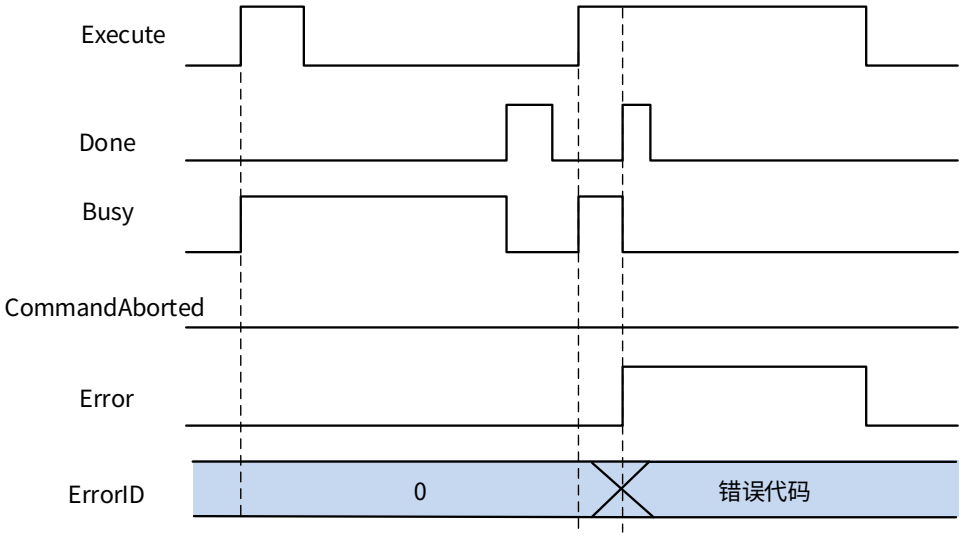
成员	类型	初始值	描述
Number_of_pairs	INT	0	轮廓路径的段数
IsAbsolute	BOOL	TRUE	绝对运动(TRUE)和相对运动选择
MC_TV_Array	ARRAY[1...N] OF SMC_TV	-	时间和速度的数组

SMC_TV 具体描述如下：

成员	类型	初始值	描述
delta_time	TIME	TIME#0ms	速度值段的时间
Velocity	LREAL	0	当前记录的速度值

注：整个速度的过程为 S 曲线加减速的方式，每段轮廓的速度都为叠加的计算方式；在指令重复运行时，速度也为叠加方式，在指令使用时避免速度超限的出现；重复运行一定把本轴的状态重回到 Standstill 状态。

4、 时序图



7.1.19 MC_WriteBoolParameter

MC_WriteBoolParameter：指令设置驱动轴的位参数。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_WriteBool Parameter	设置轴的位参数		MC_WriteBoolParameter(Axis:=, Execute:=, ParameterNumber:=, Value:=, Done=>, Busy=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Enable	执行条件	BOOL	TRUE,FALSE	FALSE	置为 TRUE，启动功能块处理
ParameterNumber	轴参数的序号	DINT	-	0	访问轴参数的索引、子索引和序号
Value	设定值	BOOL	TRUE,FALSE	FALSE	设置位参数值

注意：

- ParameterNumber(DINT)=-DWORD_TO_DINT(SHL(USINT_TO_DOWRD(usiDataLength),24)(对象字典中的数据长度)+SHL(UINT_TO_DWORD(uiIndex),8)(对象字典中的索引-16bit)+usisubIndex(对象字典中的子索引-8bit))。
- usiDataLength:按字节数填写；1 字节为 16#01；2 字节为 16#02；4 字节为 16#04 等。

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	设置操作成功	BOOL	TRUE,FALSE	FALSE	设置操作成功置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

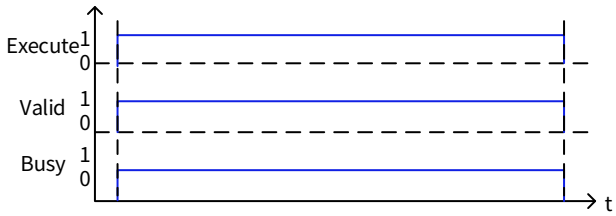
3、 功能说明

通过 MC_WriteBoolParameter 设置轴的位参数，指令为 Execute 上升沿触发。指令可以重复多次使用，互不影响。

4、 时序图

- 功能块的 Execute 必须为上升沿触发条件。
- 功能块的 Done 表示设置操作成功。
- 功能块的 Busy 表示当前功能块正在执行中。

时序操作说明：



7.1.20 MC_WriteParameter

MC_WriteParameter：指令设置驱动轴的参数。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_WriteParameter	设置轴的参数		MC_WriteParameter(Axis:=, Execute:=, ParameterNumber:=, Value:=, Done=>, Busy=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Enable	执行条件	BOOL	TRUE,FALSE	FALSE	置为 TRUE，启动功能块的处理
ParameterNumber	轴参数的序号	DINT	-	0	访问轴参数的索引、子索引和序号

输入变量	名称	数据类型	有效范围	初始值	描述
Value	设定值	BOOL	TRUE,FALSE	FALSE	设置位参数值

注意：

- ParameterNumber(DINT)=-DWORD_TO_DINT(SHL(USINT_TO_DOWRD(usiDataLength),24)(对象字典中的数据长度)+SHL(UINT_TO_DWORD(uiIndex),8)(对象字典中的索引-16bit)+usisubIndex(对象字典中的子索引-8bit))。
- usiDataLength：按字节数填写；1 字节为 16#01；2 字节为 16#02；4 字节为 16#04 等。

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	设置操作成功	BOOL	TRUE,FALSE	FALSE	设置操作成功置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

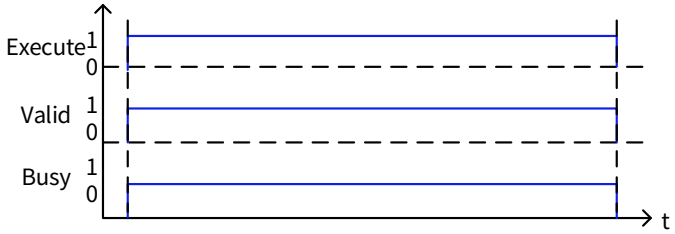
3、 功能说明

通过 MC_WriteParameter 设置轴的位参数，指令为 Execute 上升沿触发。指令可以重复多次使用，互不影响。

4、 时序图

- 功能块的 Execute 必须为上升沿触发条件。
- 功能块的 Done 表示设置操作成功。
- 功能块的 Busy 表示当前功能块正在执行中。

时序操作说明:



7.1.21 MC_AbortTrigger

MC_AbortTrigger：功能块终止输入锁存相关事件的关联特性，和 MC_Touchprobe 配套使用。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_AbortTrigger	功能块终止事件关联	MC_AbortTrigger Axis AXIS_REF_SM3 — BOOL Done TriggerInput TRIGGER_REF — BOOL Busy Execute BOOL — BOOL Error SMC_ERROR ErrorID	MC_AbortTrigger(Axis:=, TriggerInput:=, Execute:=, Done=>, Busy=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入输出变量	名称	数据类型	有效范围	初始值	描述
TriggerInput	触发信号	TRIGGER_REF			触发信号、触发属性等描述

TRIGGER_REF 说明

输入输出变量	名称	数据类型	初始值	描述
TRIGGER_REF	iTriggerNumber	INT	-1	在驱动器模式下，锁定功能中的哪一个。 0：探针 1 上升沿锁存 1：探针 1 下降沿锁存 2：探针 2 上升沿锁存 3：探针 2 下降沿锁存
	bFastLatching	BOOL	TRUE	指定锁存触发的模式： TRUE：驱动器模式 FALSE：控制器模式
	bInput	BOOL	FALSE	当 bFastLatching=FALSE 时，由控制器 Input 信号触发
	bActive	BOOL	FALSE	触发的有效信号

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行条件	BOOL	TRUE,FALSE	FALSE	上升沿触发启动功能块的处理

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	设置操作成功	BOOL	TRUE,FALSE	FALSE	设置操作成功置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

通过 MC_AbortTrigger 功能块，把触发信号或属性的指令以及相关的触发指令的关联操作终止。功能块的 Execute 必须为上升沿触发条件；功能块的 Done 为 TRUE 表示设置操作成功，Busy 为 TRUE 表示当前功能块正在执行中。

7.1.22 MC_ReadActualTorque

MC_ReadActualTorque：指令读取驱动器运行的当前力矩值，读取的当前力矩值保存在自己定义的变量单元中。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_ReadActualTorque	当前力矩值读取指令		MC_ReadActualTorque(Axis:=, Enable:=, Valid=>, Busy=>, Error=>, ErrorID=>, Torque=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Enable	执行条件	BOOL	TRUE,FALSE	FALSE	置为 TRUE，启动功能块的处理

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Valid	当前力矩值可获取标志	BOOL	TRUE,FALSE	FALSE	能正确的获取驱动器的力矩值，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码
Torque	获取的当前力矩值	LREAL	力矩值	0	指令读出来的当前力矩数据

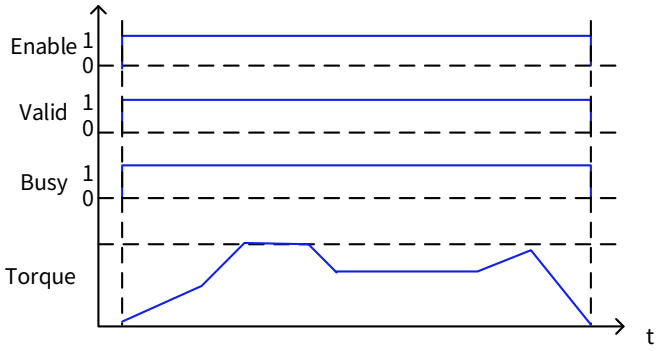
3、 功能说明

通过 MC_ReadActualTorque 读取驱动器中的当前力矩值指令，指令为 Enable 高电平使能效应。指令可以重复多次使用，互不影响。

4、 时序图

- 功能块的 Enable 必须为 TRUE 的条件。
- 功能块的 Valid 为 TRUE 表示读出的 Torque 为有效的数据值。
- 功能块的 Busy 为 TRUE 表示当前功能块正在执行中。

时序操作说明



7.1.23 MC_ReadActualVelocity

MC_ReadActualVelocity：指令读取驱动器运行的当前速度值，读取的当前力矩值保存在自己定义的变量单元中。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_ReadActualVelocity	当前速度读取指令	MC_ReadActualVelocity Axis AXIS_REF_SM3 — BOOL Valid Enable BOOL — BOOL Busy BOOL Error SMC_ERROR ErrorID LREAL Velocity	MC_ReadActualVelocity(Axis:=, Enable:=, Valid=>, Busy=>, Error=>, ErrorID=>, Velocity=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Enable	执行条件	BOOL	TRUE,FALSE	FALSE	置为 TRUE，启动功能块的处理

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Valid	当前力矩值可获取标志	BOOL	TRUE,FALSE	FALSE	能正确的获取驱动器的力矩值，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码
Velocity	获取的当前速度值	LREAL	速度值	0	指令读出来的当前速度数据

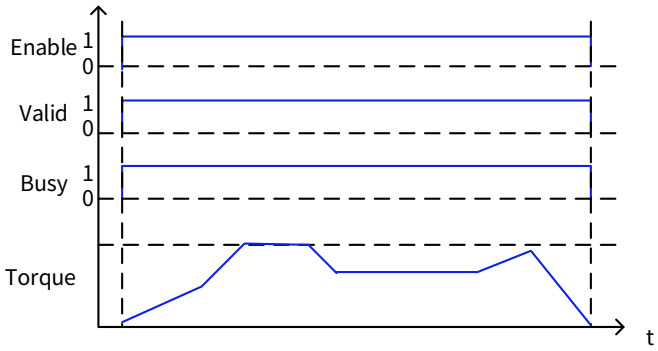
3、 功能说明

通过 MC_ReadActualVelocity 读取驱动器中的当前速度值指令，Enable 高电平有效。指令可以重复多次使用，互不影响。

4、 时序图

- 功能块的 Enable 必须为 TRUE 的条件。
- 功能块的 Valid 为 TRUE 表示读出的 Torque 为有效的数据值。
- 功能块的 Busy 为 TRUE 表示当前功能块正在执行中。

时序操作说明



7.1.24 MC_SetPosition

MC_SetPosition：将指令中的位置数据设为当前轴的位置数据，对设置位置数据操作不会产生任何位移移动，用于产生坐标系的位移。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_SetPosition	当前位置变更	MC_SetPosition Axis AXIS_REF_SM3 — BOOL Done Execute BOOL — BOOL Busy Position LREAL — BOOL Error Mode BOOL — SMC_ERROR ErrorID	MC_SetPosition(Axis:=, Execute:=, Position:=, Mode:=, Done=>,

指令	名称	图形表现	ST 表现
			Busy=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行条件	BOOL	TRUE,FALSE	FALSE	上升沿触发启动功能块的处理
Position	轴位置数据	LREAL	-	0	位置数据
Mode	设定值	BOOL	TRUE,FALSE	FALSE	位置模式 TRUE：相对位置(RELATIVE) FALSE：绝对位置(ABSOLUTE)

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	设置操作成功	BOOL	TRUE,FALSE	FALSE	设置操作成功置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

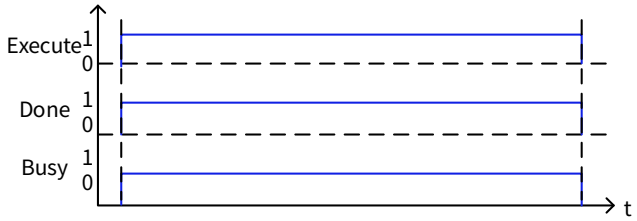
3、 功能说明

通过 MC_SetPosition 设置轴的位置参数，不产生任何位移，但形成了坐标偏移，Execute 上升沿触发。指令可重复多次使用，互不影响。

4、 时序图

- 功能块的 Execute 必须为上升沿触发条件。
- 功能块的 Done 为 TRUE 表示设置操作成功。
- 功能块的 Busy 为 TRUE 表示当前功能块正在执行中。

时序操作说明



7.1.25 MC_TouchProbe

MC_TouchProbe：指令通过外部信号触发，保存当前轴的位置数据。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_ TouchProbe	启用外部锁定	MC_TouchProbe Axis AXIS_REF_SM3 — BOOL Done TriggerInput TRIGGER_REF — BOOL Busy Execute BOOL — BOOL Error WindowOnly BOOL — SMC_ERROR ErrorID FirstPosition LREAL — LREAL RecordedPosition LastPosition LREAL — BOOL CommandAborted	MC_TouchProbe(Axis:=, TriggerInput:=, Execute:=,

指令	名称	图形表现	ST 表现
			WindowOnly:=, FirstPosition:=, LastPosition:=, Done=>, Busy=>, Error=>, ErrorID=>, RecordedPosition=>, CommandAborted=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例
TriggerInput	触发信号	TRIGGER_REF	-	-	触发信号或触发属性等关联属性

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行条件	BOOL	TRUE,FALSE	FALSE	上升沿触发启动功能块的处理
WindowOnly	触发窗口	BOOL	TRUE,FALSE	FALSE	-
FirstPosition	触发开始位置	LREAL	-	0	指定接收触发的开始位置
LastPosition	触发结束位置	LREAL	-	0	指定接收触发的结束位置

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	设置操作成功	BOOL	TRUE,FALSE	FALSE	设置操作成功置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码
RecordedPosition	触发记录位置	LREAL	-	-	触发发生时当前的位置
CommandAbort	指令被中断	BOOL	TRUE,FALSE	FALSE	当前指令被中断，置为 TRUE

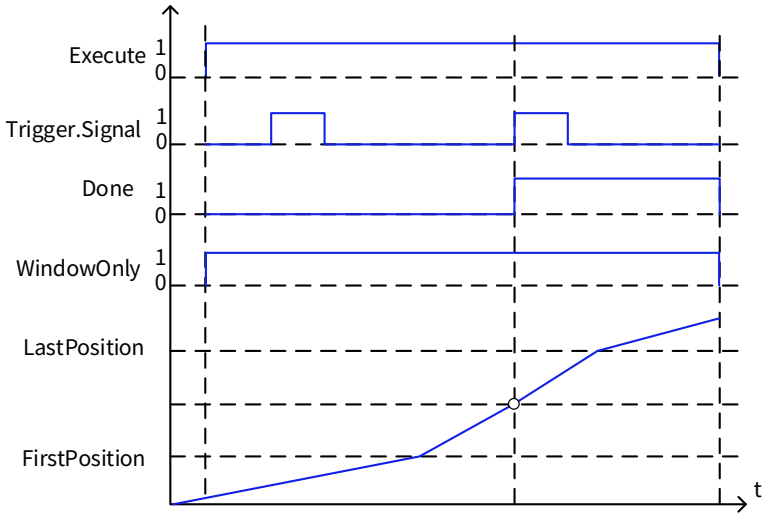
3、 功能说明

通过 MC_TouchProbe 功能块 TriggerInput 的信号触发时记录运行轴的当前位置。Execute 上升沿执行驱动器锁存时：驱动器采集到锁存信号会在记录位置。

4、 时序图

- 功能块的 Execute 必须为上升沿触发条件。
- 功能块的 Done 为 TRUE 表示设置操作成功。

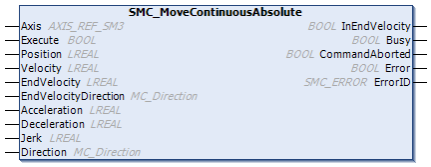
时序操作说明



7.1.26 MC_MoveContinuousAbsolute

MC_MoveContinuousAbsolute：轴按绝对位置连续运行(单位按轴设置),绝对位置由 Distance 指定，最后的运行速度 EndVelocity 运行。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_MoveContinuousAbsolute	轴绝对位置连续控制指令		SMC_MoveContinuousAbsolute(Axis:=, Execute:=, Position:=, Velocity:=, EndVelocity:=, EndVelocityDirection:=, Acceleration:=, Deceleration:=, Jerk:=, Direction:=, InEndVelocity=>, Busy=>, CommandAborted=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行条件	BOOL	TRUE,FALSE	FALSE	上升沿触发将启动功能块的处理
Distance	运动相对位置	LREAL	数据范围	0	此数据为运动的相对位置
Velocity	运行速度	LREAL	数据范围	0	轴运行到目标位置的最大速度
EndVelocity	运行结束速度	LREAL	数据范围	0	指令执行完成后的运行速度

输入变量	名称	数据类型	有效范围	初始值	描述
EndVelocity-Direction	结束速度的方向	MC_Direction	positive, negative, current;	Current	可以使用：positive, negative, current; 不可使用:shortest,fastest
Acceleration	加速度	LREAL	数据范围	0	速度变大时加速度值
Deceleration	减速度	LREAL	数据范围	0	速度变小时减速度值
Direction	运行方向	shortest	数据范围	shortest	对于线性/直线轴：positive, negative； 对于旋转/圆周轴：positive, negative，current, shortest, fastest

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
InEndVelocity	指令位置到达	BOOL	TRUE,FALSE	FALSE	轴指令执行位置到达，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码
CommandAbort	指令被中断	BOOL	TRUE,FALSE	FALSE	当前指令被中断，置为 TRUE

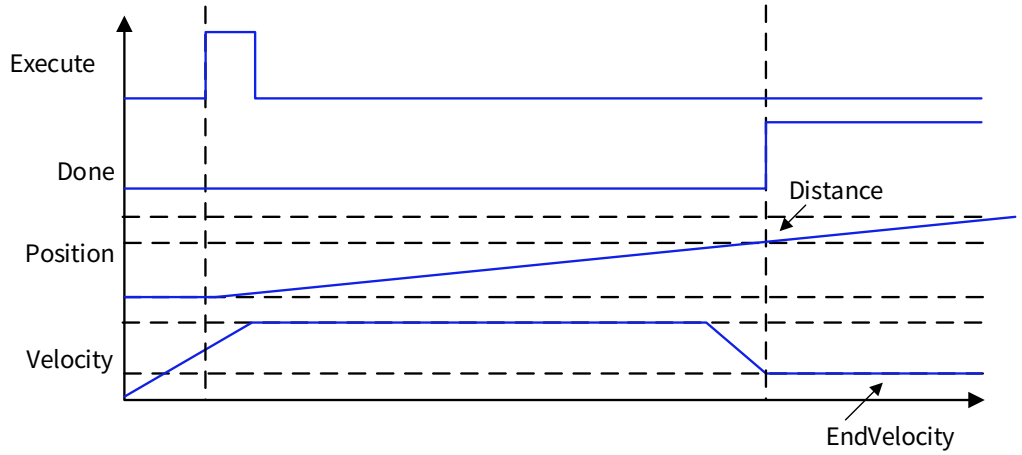
3、 功能说明

本功能块为轴绝对定位指令，Distance 数据为轴的绝对位置。本功能块运行状态为 Standstill 中，指令运行时的状态为 DiscreteMotion，一个完整的运行过程一定要控制轴的不同运动状态。启动指令为 Execute 的上升沿启动，本指令在 DiscreteMotion 可以重复上升沿有效，每次都可以刷新最新的 Position 位置。Acceleration 或 Deceleration 为 0，指令运行都为异常状态，但轴状态为 DiscreteMotion。

4、 时序图

- 轴必须处于 Standstill 状态指令才能运行。
- 功能块的 Execute 必须为上升沿触发条件。
- 功能块的 Busy 表示当前功能块正在执行中。

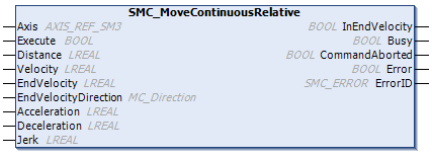
时序操作说明



7.1.27 MC_MoveContinuousRelative

MC_MoveContinuousRelative：轴按相对位置连续运行（单位按轴设置），绝对位置由 Distance 指定，最后的运行速度 EndVelocity 运行。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_MoveContinuousRelative	轴相对位置连续控制指令		SMC_MoveContinuousRelative(Axis:=, Execute:=, Distance:=, Velocity:=, EndVelocity:=, EndVelocityDirection:=, Acceleration:=, Deceleration:=, Jerk:=, InEndVelocity=>, Busy=>, CommandAborted=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行条件	BOOL	TRUE,FALSE	FALSE	上升沿触发启动功能块的处理
Distance	运动相对位置	LREAL	数据范围	0	此数据为运动的相对位置
Velocity	运行速度	LREAL	数据范围	0	轴运行到目标位置的最大速度
EndVelocity	运行结束速度	LREAL	数据范围	0	指令执行完成后的运行速度
EndVelocity-Direction	结束速度的方向	MC_Direction	Positive，Negative，Current	Current	可以使用：Positive，Negative，Current 不可使用：Shortest，Fastest
Acceleration	加速度	LREAL	数据范围	0	速度变大时加速度值
Deceleration	减速度	LREAL	数据范围	0	速度变小时减速度值
Direction	运行方向	Shortest	数据范围	Shortest	对于线性/直线轴：Positive，Negative 对于旋转/圆周轴：Positive，Negative，Current，Shortest，Fastest

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
InEndVelocity	指令位置到达	BOOL	TRUE,FALSE	FALSE	轴指令执行位置到达，置为 TRUE
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
CommandAbort	指令被中断	BOOL	TRUE,FALSE	FALSE	当前指令被中断，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

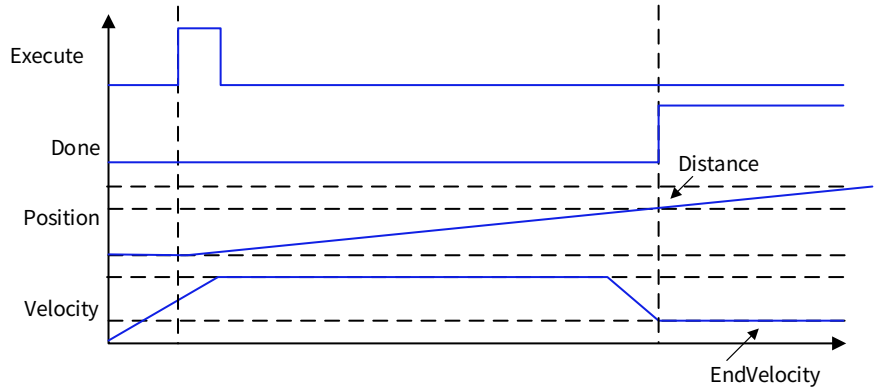
3、 功能说明

本功能块运行状态为 Standstill 中，指令运行时的状态为 DiscreteMotion，在指令执行中关注本轴的运行状态，避免打断本轴的其他指令或被其他指令打断本轴的执行。启动指令为 Execute 的上升沿启动,本指令在 DiscreteMotion 可以重复上升沿有效,每次都可以刷新最新的 Position 位置。Acceleration 或 Deceleration 为零，指令运行都为异常状态，但轴的状态为 DiscreteMotion。

4、 时序图

- 轴必须处于 Standstill 状态指令才能运行。
- 功能块的 Execute 必须为上升沿触发条件。
- 功能块的 Busy 表示当前功能块正在执行中。

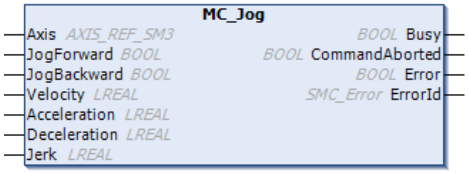
时序操作说明



7.1.28 MC_Jog

MC_Jog：根据指定目标速度执行微动移动

1、 指令格式

指令	名称	图形表现	ST 表现
MC_Jog	轴点动指令		MC_Jog(Axis:=, JogForward:=, JogBackward:=, Velocity:=, Acceleration:=, Deceleration:=, Jerk:=, Busy=>, CommandAborted=>, Error=>, ErrorId=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
JogForward	正向有效	BOOL	TRUE,FALSE	FALSE	设置为 TRUE 则开始正向移动；设置为 FALSE 则停止正向移动

输入变量	名称	数据类型	有效范围	初始值	描述
JogBackward	负向有效	BOOL	TRUE,FALSE	FALSE	设置为 TRUE 则开始反向移动；设置为 FALSE 则停止反向移动
Velocity	目标速度	LREAL	正数或 0	0	指定目标速度。 单位：[指令单位/s]
Acceleration	加速度	LREAL	正数或 0	0	指定加速度。 单位：[指令单位/s]
Deceleration	减速度	LREAL	正数或 0	0	指定减速度。 单位：[指令单位/s]
Jerk	跃度	LREAL	数据范围	0	曲线加减速的斜率变化值

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
CommandAborted	指令被中断	BOOL	TRUE,FALSE	FALSE	当前指令被中断，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorId	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

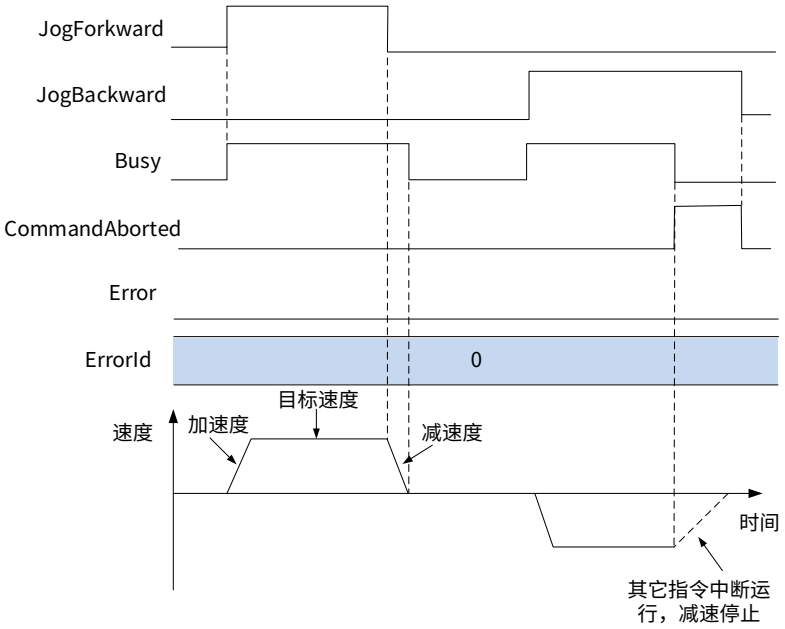
根据指定 Velocity（目标速度）执行点动运行。需要正向运行时，将 JogForward（正向运行有效）置为 TRUE；需要反向运行时，将 JogBackward（负向运行有效）置为 TRUE。同时将 JogForward（正向运行有效）和 JogBackward（负向运行有效）置为 TRUE，将不会有运动发生。如果 MC_Jog 指令的指令速度设置值超过轴参数中的点动最高速度，则以点动最高速度执行。

4、 时序图

在启动 JogForward（正向运行有效）或 JogBackward（负向运行有效）的同时，Busy（执行中）变为 TRUE；在 JogForward（正向运行有效）或 JogBackward（负向运行有效）的下降沿开始减速至停止轴的同时，Busy（执行中）变为 FALSE。

利用其它指令中止本指令时，CommandAborted（执行中断）变为 TRUE，Busy（执行中）变为 FALSE。

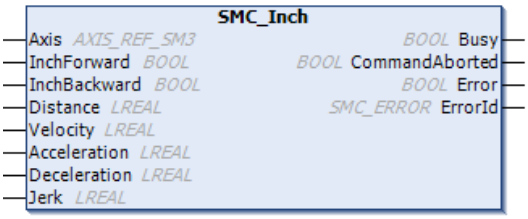
时序操作说明



7.1.29 MC_Inch

MC_Inch：轴单步运动控制，通过程序能实现一步接一步的单步控制。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_Inch	轴相对定位指令		SMC_Inch(Axis:=, InchForward:=, InchBackward:=, Distance:=, Velocity:=, Acceleration:=, Deceleration:=, Jerk:=, Busy=>, CommandAborted=>, Error=>, ErrorId=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
InchForward	正向执行	BOOL	TRUE,FALSE	FALSE	如果 InchForward 为 TRUE，轴将按照给定的速度正向运行至设定位置。输入被指定为 False，要二次触发为 TRUE，才能再次启动。 如果 InchForward 在到达位置之前被设置为 FALSE，那么轴将立即减速到 0，并且 Busy 将会被设置为 FALSE。如果输入 InchBackward 在仿真情况下被设置为 TRUE，那么将不会有运动产生。
InchBackward	反向执行	BOOL	TRUE,FALSE	FALSE	如果 InchBackward 为 TRUE，轴将会按照给定的速度值进行反向运动至设定位置。然后输入必须被设置为 FALSE，再设置为 TRUE，启动另一个运动。 **注意：**如果输入信号 InchForward 同时被设置为 TRUE，那么将不会有轴运动。
Distance	移动的距离	LREAL	数据范围	0	此数据为移动的距离
Velocity	运行速度	LREAL	数据范围	0	轴运行到目标位置的最大速度

输入变量	名称	数据类型	有效范围	初始值	描述
Acceleration	加速度	LREAL	数据范围	0	速度变大时加速度值
Deceleration	减速度	LREAL	数据范围	0	速度变小时减速度值

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
CommandAborted	指令被中断	BOOL	TRUE,FALSE	FALSE	当前指令被中断，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorId	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

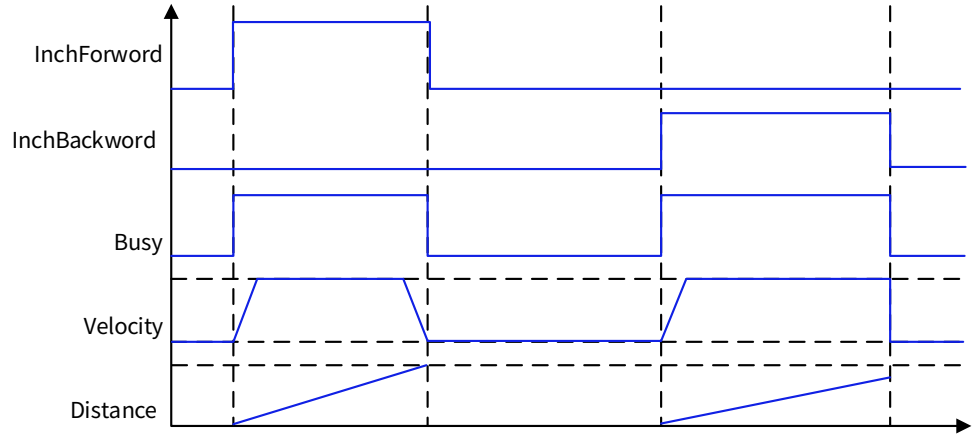
3、 功能说明

本功能块运行状态为 Standstill 中，指令运行时的状态为 DiscreteMotion，在指令执行中关注本轴的运行状态，避免打断本轴的其他指令或被其他指令打断本轴的执行。Acceleration 或 Deceleration 为零，指令运行都为异常状态，但轴的状态为 DiscreteMotion；

4、 时序图

- 功能块的 InchForward/InchBackward 必须有 TRUE/FALSE 的条件。
- 功能块的 Busy 表示当前功能块正在执行中。

时序操作说明



7.1.30 SMC3_PersistPosition

该指令用来保持纪录实轴绝对值编码器的位置（断电重启控制器后，恢复断电前位置记录值）。如果伺服电机使用的是绝对值编码器，使用该功能块配合使用。

1、 指令格式

指令	名称	图形表现	ST 表现
SMC3_PersisitPosition	轴位置保持		SMC3_PersistPosition(Axis:=, PersistentData:=, bEnable:=, bPositionRestored=>, bPositionStored=>, bBusy=>, bError=>, eErrorID=>, eRestoringDiag=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例
PersistentData	保持数据	SMC3_PersistPosition_Data	-	-	存储位置信息的断电保持型数据结构

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Enable	执行	BOOL	TRUE,FALSE	FALSE	TRUE 功能块执行，FALSE 不执行功能块

输出变量

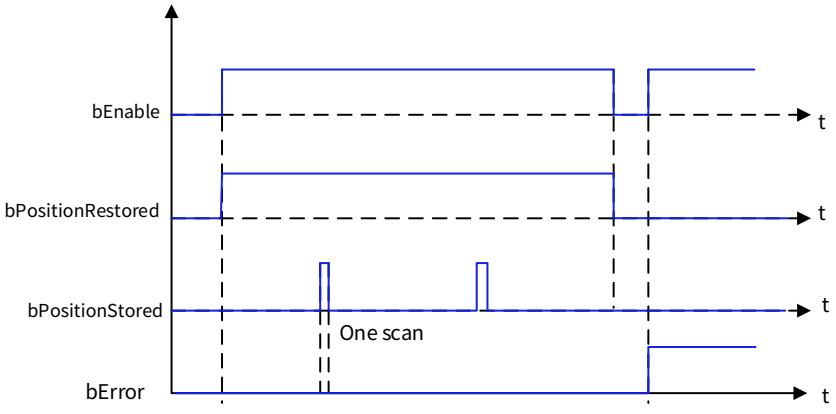
输出变量	名称	数据类型	有效范围	初始值	描述
bPositionRestored	位置恢复	BOOL	TRUE,FALSE	FALSE	TRUE,轴重启后位置恢复完成
bPositionStored	位置保存	BOOL	TRUE,FALSE	FALSE	TRUE,调用功能块后保存位置完成
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码
eRestoringDiag	恢复诊断	SMC3_Persist-PositionDiag	-	-	位置恢复中的诊断信息 SMC3_PPD_RESTORING_OK：位置成功恢复 SMC3_PPD_AXIS_PROP_CHANGED：轴参数有更改，无法恢复位置 SMC3_PPD_DATA_STORED_DURING_WRITING：功能块从轴参数数据结构复制数据，而不是从 PersistentData 数据中复制。可能原因：非同步性持续变量、控制器崩溃死机。

3、 功能说明

PLC 重启 bEnable 信号为 TRUE，则 bPositionRestroed 输出为 TRUE。

4、 时序图

功能块的 Busy 为 TRUE 表示当前功能块正在执行中。



7.1.31 SMC3_PersistPositionSingleturn

该指令用来保持纪录实轴绝对值编码器（单圈绝对值）的位置（断电重启控制器后，恢复断电前位置记录值）。如果伺服电机使用的是单圈绝对值编码器，使用该功能块配合使用。

1、 指令格式

指令	名称	图形表现	ST 表现
SMC3_PersisitPositionSingleturn	轴位置保持		SMC3_PersistPositionSingleturn(Axis:=, PersistentData:=, bEnable:=, usiNumberOfAbsoluteBits:=, bPositionRestored=>, bPositionStored=>, bBusy=>, bError=>, eErrorID=>, eRestoringDiag=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例
PersistentData	保持数据	SMC3_PersistPosition_Data	-	-	存储位置信息的断电保持型数据结构

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Enable	执行	BOOL	TRUE,FALSE	FALSE	TRUE 功能块执行，FALSE 不执行功能块

输出变量

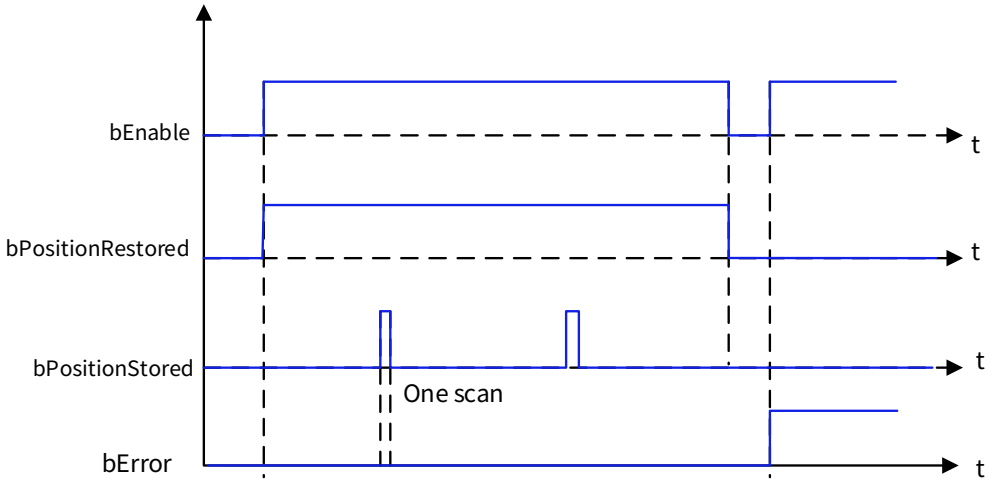
输出变量	名称	数据类型	有效范围	初始值	描述
bPositionRestored	位置恢复	BOOL	TRUE,FALSE	FALSE	TRUE,轴重启后位置恢复完成
bPositionStored	位置保存	BOOL	TRUE,FALSE	FALSE	TRUE,调用功能块后保存位置完成
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码
eRestoringDiag	恢复诊断	SMC3_Persist-PositionDiag	-	-	位置恢复中的诊断信息 SMC3_PPD_RESTORING_OK：位置成功恢复；SMC3_PPD_AXIS_PROP_CHANGED：轴参数有更改，无法恢复位置；SMC3_PPD_DATA_STORED_DURING_WRITING：功能块从轴。 参数数据结构复制数据，而不是从 PersistentData 数据中复制。可能原因：非同步性持续变量、控制器崩溃死机。

3、 功能说明

PLC 重启 bEnable 信号为 TRUE，则 bPositionRestroed 输出为 TRUE。

4、 时序图

功能块的 Busy 为 TRUE 表示当前功能块正在执行中。



7.1.32 SMC3_PersistPositionLogical

该指令用来保持纪录实轴绝对值编码器（单圈绝对值）的位置（断电重启控制器后，恢复断电前位置记录值）。如果伺服电机使用的是单圈绝对值编码器，使用该功能块配合使用。

1、 指令格式

指令	名称	图形表现	ST 表现
SMC3_PersisitPositionLogical	轴位置保持	SMC3_PersistPositionLogical Axis bPositionRestored PersistentData bPositionStored bBusy bError bEnable eErrorID eRestoringDiag	SMC3_PersistPositionLogical(Axis:=, PersistentData:=, bEnable:=, bPositionRestored=>, bPositionStored=>, bBusy=>, bError=>, eErrorID=>, eRestoringDiag=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例
PersistentData	保持数据	SMC3_Persist Position_Data	-	-	存储位置信息的断电保持型数据结构

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Enable	执行	BOOL	TRUE,FALSE	FALSE	TRUE 功能块执行，FALSE 不执行功能块

输出变量

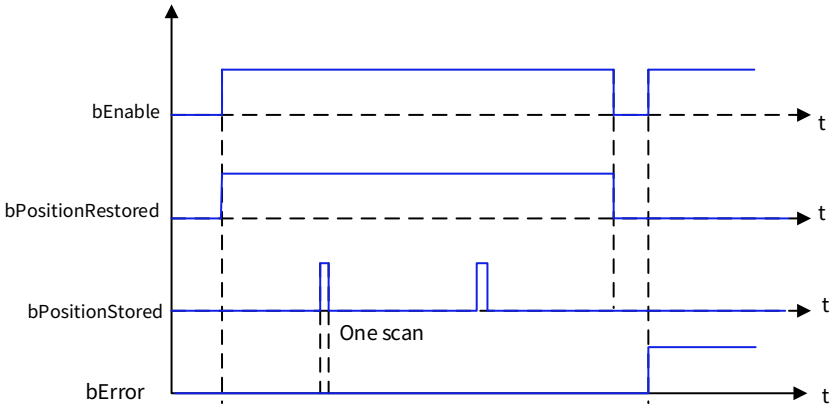
输出变量	名称	数据类型	有效范围	初始值	描述
bPositionRestored	位置恢复	BOOL	TRUE,FALSE	FALSE	TRUE,轴重启后位置恢复完成
bPositionStored	位置保存	BOOL	TRUE,FALSE	FALSE	TRUE,调用功能块后保存位置完成
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	当前指令正在执行中，置为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码
eRestoringDiag	恢复诊断	SMC3_Persist-PositionDiag	-	-	位置恢复中的诊断信息 SMC3_PPD_RESTORING_OK：位置成功恢复 SMC3_PPD_AXIS_PROP_CHANGED：轴参数有更改，无法恢复位置 SMC3_PPD_DATA_STORED_DURING_WRITING：功能块从轴参数数据结构复制数据，而不是从 PersistentData 数据中复制。可能原因：非同步性持续变量、控制器崩溃死机。

3、 功能说明

PLC 重启 bEnable 信号为 TRUE，则 bPositionRestroed 输出为 TRUE。

4、 时序图

功能块的 Busy 为 TRUE 表示当前功能块正在执行中。



7.1.33 SMC_Homing

轴回零指令，与 MC_Home 有区别，MC_Home 为伺服驱动器控制的回零方式，SMC_Homing 指令为 PLC 控制器控制的回零方式。

1、 指令格式

指令	名称	图形表现	ST 表现
SMC_Homing	轴回零指令	SMC_Homing Axis AXIS_REF_SM3 bExecute BOOL fHomePosition LREAL fVelocitySlow LREAL fVelocityFast LREAL fAcceleration LREAL fDeceleration LREAL fJerk LREAL nDirection MC_Direction bReferenceSwitch BOOL fSignalDelay LREAL nHomingMode SMC_HOMING_MODE bReturnToZero BOOL bIndexOccured BOOL fIndexPosition LREAL bIgnoreHWLimit BOOL BOOL bDone BOOL bBusy BOOL bCommandAborted BOOL bError SMC_ERROR nErrorID BOOL bStartLatchingIndex	SMC_Homing(Axis:=, bExecute:=, fHomePosition:=, fVelocitySlow:=, fVelocityFast:=, fAcceleration:=, fDeceleration:=,

指令	名称	图形表现	ST 表现
			fJerk:=, nDirection:=, bReferenceSwitch:=, fSignalDelay:=, nHomingMode:=, bReturnToZero:=, bIndexOccured:=, fIndexPosition:=, bIgnoreHWLimit:=, bDone=>, bBusy=>, bCommandAborted=>, bError=>, nErrorID=>, bStartLatchingIndex=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
bExecute	执行	BOOL	TRUE,FALSE	FALSE	TRUE 功能块执行，FALSE 不执行功能块
fHomePosition	原点设定位置	LREAL	-	0	回零后原点设定位置，单位为用户标定后的单位
fVelocitySlow	慢速	LREAL	-	0	离开参考开关后慢速设定速度
fVelocityFast	快速	LREAL	-	0	接近参考开关置位时，快速设定速度
fAcceleration	加速度	LREAL	-	0	加速度设定值
fDeceleration	减速度	LREAL	-	0	减速度设定值
fJerk	加加速度	LREAL	-	0	加加速度设定值
nDirection	回零方向	MC_DIRECTION	-	negative	回零开始方向
bReferenceSwitch	参考开关	BOOL	TRUE,FALSE	FALSE	连接参考开关，TRUE:参开开关触发，FALSE:参考开关闭合
fSignalDelay	延迟	LREAL	-	0	参考开关的传输时间，用来补偿死区时间。单位为秒。
nHomingMode	回零模式	SMC_HOMING_MODE	FAST_BSLOW_S_STOP, FAST_BSLOW_STOP_S, FAST_BSLOW_I_S_STOP, FAST_SLOW_S_STOP, FAST_SLOW	0	回零模式

输入变量	名称	数据类型	有效范围	初始值	描述
			_STOP_S, FAST_SLOW_I_S_STOP		
bReturnTozero	返回零位	BOOL	TRUE,FALSE	FALSE	TRUE：回零完成后轴运行到位置零（**注意：**如果 fHome Position=10，则回零完成后轴位置变为 10，bReturnTozero 为 TRUE 则回零完成后轴反向走 10 个单位到 0 位）
bIndexOccured	脉冲信号	BOOL	TRUE,FALSE	FALSE	TRUE：标志脉冲记录，回零模式为 FAST_BSLOW_I_S_STOP，FAST_SLOW_I_S_STOP 时生效
fIndexPosition	记录位置	LREAL	-	0	标志脉冲时记录的位置
bIgnoreHWLimit	忽略硬限位	BOOL	TRUE,FALSE	FALSE	TRUE，设定硬件限位开关使能为 false，如果相同的物理开关用于硬件限位开关和参考开关，那么硬件控制将被设置为假。

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
bDone	设置操作成功	BOOL	TRUE,FALSE	FALSE	TRUE，回零完成
bBusy	指令正在执行	BOOL	TRUE,FALSE	FALSE	TRUE，功能块正在执行
bCommandAborted	指令被中断	BOOL	TRUE,FALSE	FALSE	TRUE，功能块被其他动作指令中断
bError	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
nErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码
bStartLatchingIndex	启动锁存	BOOL	TRUE,FALSE	FALSE	由“bIndexOccured”和“fIndexPosition”共同作用产生

具体的回零模式描述，见下表：

模式	类型	初始值	描述
FAST_BSLOW_S_STOP	SMC_HOMING_MODE	0	轴按照设定方向以快速度走向原点开关，碰到原点开关后反向以慢速离开原点开关，离开后先执行 MC_setPosition 将当前位置设为 fHomePosition 设定值，然后执行 MC_stop
FAST_BSLOW_STOP_S	SMC_HOMING_MOD	1	按照设定方向以快速走向原点开关，碰到原点开关后反向以慢速离开原点开关，离开后先执行 MC_stop 将轴停止，然后执行 MC_setPosition，将当前位置设为 fHomePosition 设定值
FAST_BSLOW_I_S_STOP	SMC_HOMING_MOD	2	按照设定方向以快速走向原点开关，碰到原点开关后反向以慢速离开原点开关，bIndexOccured 信号到时先执行 MC_setPosition 再执行 MC_stop
FAST_SLOW_S_STOP	SMC_HOMING_MOD	4	按照设定方向以快速走向原点开关，碰到原点开关后以慢速离开原点开关，离开后先执行 MC_setPosition 将当前位置设为 fHomePosition 设定值，然后执行 MC_stop
FAST_SLOW_STOP_S	FAST_SLOW_STOP_S	5	按照设定方向以快速走向原点开关，碰到原点

模式	类型	初始值	描述
	SMC_HOMING_MOD		开关后以慢速离开原点开关，离开后先执行 MC_stop，然后执行 MC_setPosition 将当前位置设为 fHomePosition 设定值
FAST_SLOW_I_S_STOP	SMC_HOMING_MOD	6	按照设定方向以快速走向原点开关，碰到原点开关后原方向以慢速离开原点开关，bIndexOccured 信号到时先执行 MC_setPosition 再执行 MC_stop

3、 功能说明

SMC_HOMING 通过 bExecute 的上升沿启动之后，轴将会按照速度 fVelocityFast 并以 nDirection 定义的方向开始运动，直到 bReferenceSwitch=FALSE。然后轴将会缓慢停止并按照相反的方向以速度 fVelocitySlow 离开参考开关。bReferenceSwitch=TRUE 后回零完成。

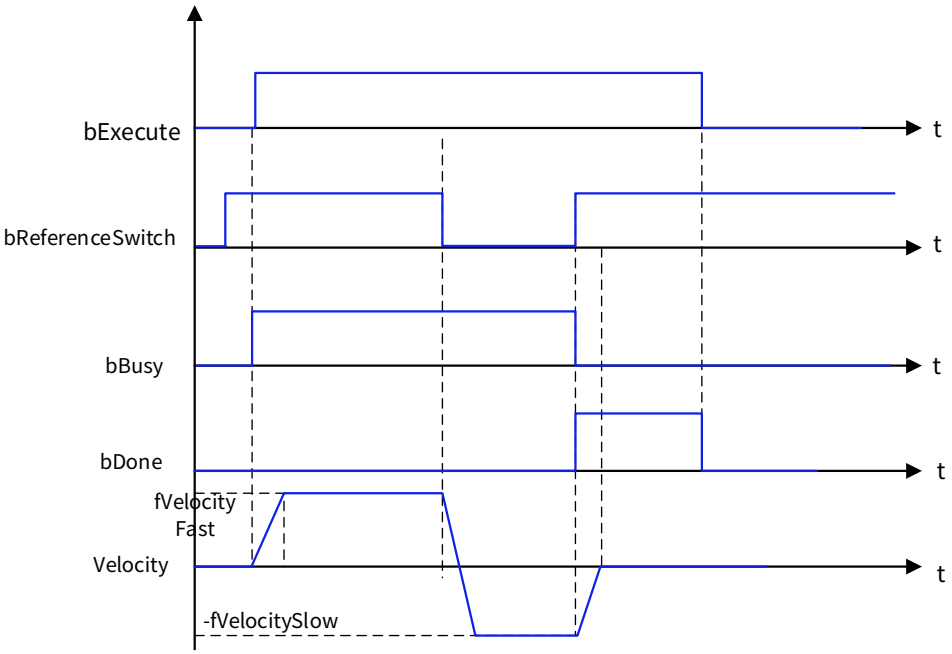
使能回零指令后 bReferenceSwitch 的状态为 ON→OFF→ON，在 OFF→ON 的上升沿回零完成，设置参考位置。参考位置=fHomePostion+（（fSignalDelay*1000+1 个 DC 时钟周期）/1000）*fVelocitySlow 实际就是补偿了设置的 bReferenceSwitch 采样延迟和一个通讯周期位移延迟。

如果 bReturnToZero=TRUE，bReferenceSwitch 的状态在 OFF→ON 的上升沿将参考位置设置为 fHomePostion+（（fSignalDelay*1000+1 个 DC 时钟周期）/1000）*fVelocitySlow，然后按照速度 fVelocityFast 运行到 0 位置。

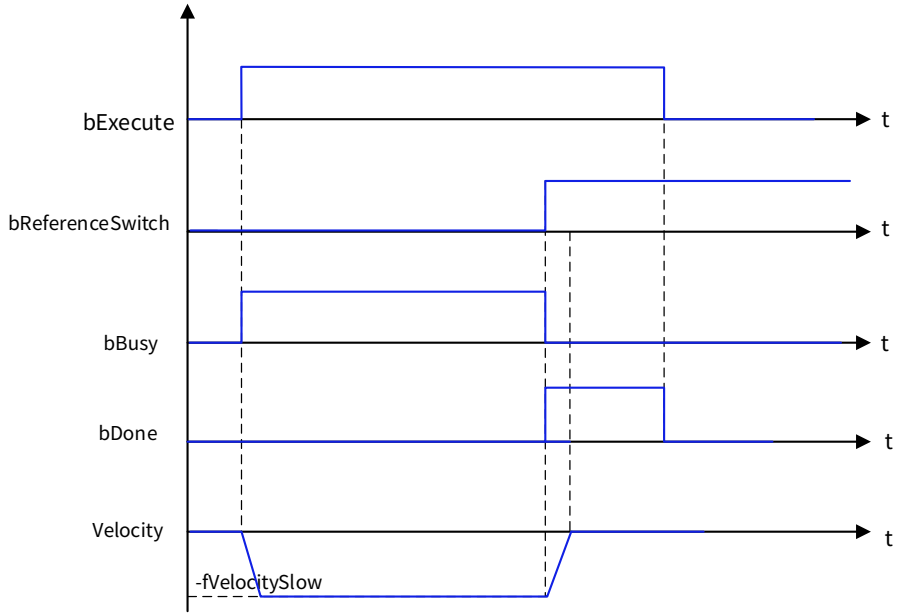
注意：Done 完成信号后，轴位置设定为：fHomePosition。设定的时机跟 nHomingMode 有关。

4、 时序图

- 指令执行时 bReferenceSwitch 为 TRUE 时



- 指令执行时 bReferenceSwitch 为 FALSE 时



7.1.34 SMC_SetControllerMode

设定伺服当前运行模式，默认为同步周期位置控制，控制模式相关设置结合伺服手册说明，DA200，位置模式为 8，速度模式为 9，转矩模式为 10。

1、 指令格式

指令	名称	图形表现	ST 表现
SMC_SetControllerMode	设定轴控制模式	SMC_SetControllerMode Axis bDone bBusy bError bExecute nErrorID nControllerMode	SMC_SetControllerMode(Axis:=, bExecute:=, nControllerMode:=, bDone=>, bBusy=>, bError=>, nErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
bExecute	执行	BOOL	TRUE,FALSE	FALSE	TRUE 功能块执行，FALSE 不执行功能块
nControllerMode	控制模式	SMC_Controller_MODE	-	SMC_Position	轴控制模式 1：转矩控制模式，SMC_torque 2：速度控制模式，SMC_Velocity 3：位置控制模式，SMC_Position

输入变量	名称	数据类型	有效范围	初始值	描述
					4：电流控制模式，SMC_Current

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
bDone	设置操作成功	BOOL	TRUE,FALSE	FALSE	TRUE，回零完成
bBusy	指令正在执行	BOOL	TRUE,FALSE	FALSE	TRUE，功能块正在执行
bError	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
nErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

功能块使用需满足条件：

1、轴必须满足这些控制条件，比如虚轴不能使用该功能块。

2、各模式支持的同步周期必须保证一致。

3、执行指令时轴必须在非“errorstop”，“stopping”，“homing”状态，否则会产生错误。

4、如果指令执行 1000 个任务周期后，轴仍然没有变为设定控制方式，则指令报错，bError 由 false 变为 true。

5、当轴的控制方式由低级向高级转变时(torque→velocity,torque→position,velocity→position)，功能块会计算高级方式的设定值，例如当有转矩模式变为位置模式时，功能块会根据当前轴实际位置叠加一个预期位置距离（通过当前实际的速度和任务周期内的时间偏移来计算），来补偿实际和设定值之间的时间滞后。

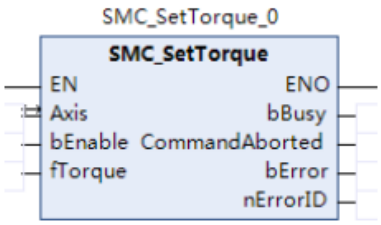
6、指令执行后，当轴的实际控制方式变为设定控制方式，bDone 信号触发，在指令触发与 bDone 信号触发之间的时间内轴仍然会运行，并且在这段时间内功能块会按照设定控制方式计算合适的设定值，但是如果一旦 bDone 信号触发而没有其他的控制指令继续给轴设定值，则轴会立即停止并报错，因此需使用 bDone 信号上升沿来触发 MC_Halt、MC_MoveVelocity、MC_MoveAbsolute 等指令来平滑控制轴。

注意：当控制模式转换为力矩控制时，后需要接力矩控制指令（如 SMC_SetTorque 等）来平滑控制轴。

7.1.35 SMC_SetTorque

该指令功能为：设定轴转矩（转矩控制模式时有效）。

1、 指令格式

指令	名称	图形表现	ST 表现
SMC_SetTorque	力矩设定		SMC_SetTorque(Axis:=, bEnable:=, fTorque:=, bBusy=>, bError=>, nErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Axis	轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
bEnable	执行	BOOL	TRUE,FALSE	FALSE	TRUE 功能块执行，FALSE 不执行功能块
fTorque	设定的力矩	LREAL	-	0	单位为 0.1%

输入变量	名称	数据类型	有效范围	初始值	描述
					(Axis.fFactorTor:=1;)

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Busy	指令正在执行	BOOL	TRUE,FALSE	FALSE	TRUE，功能块正在执行
bError	错误	BOOL	TRUE,FALSE	FALSE	异常发生时，置为 TRUE
nErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

bEnable 上升沿，无错误则，bBusy 输出为 TURE。该指令只是给轴设定转矩值用并不是转矩控制功能，轴控制模式在转矩控制模式下有效。

力矩设定指令只能在同步力矩模式下运行，在使能本指令时，需先利用 MC_SetControlMode 系统将控制模式切换为同步力矩模式。

驱动器实际力矩受配置参数中设置的最大正负力矩限制。

要停止执行本指令，可使用 MC_Stop（强制停止）指令，停止后，驱动器转换到同步位置模式。

4、 错误说明

轴报错，Error 输出；无效的轴输入，Error 输出。

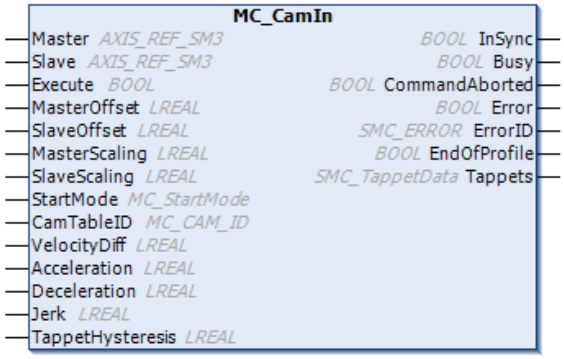
轴控制模式错误，Error 输出，错误代码 SMC_ST_WRONG_CONTROLLER_MODE。

7.2 主从轴指令

7.2.1 MC_CamIn

MC_CamIn：使用指定的凸轮表开始执行电子凸轮动作，可根据应用需求指定主从轴偏置值、缩放比和工作模式。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_Camin	凸轮动作开始		MC_CamIn(Master:=, Slave:=, Execute:=, MasterOffset:=, SlaveOffset:=, MasterScaling:=, SlaveScaling:=, StartMode:=, CamTableID:=, VelocityDiff:=, Acceleration:=, Deceleration:=, Jerk:=, TappetHysteresis:=, InSync=>, Busy=>, CommandAborted=>, Error=>, ErrorID=>, EndOfProfile=>, Tappets=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Master	主轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例
Slave	从轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

注意：主/从轴必须指定不同轴，否则可能报错。

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行凸轮功进入能块	BOOL	TRUE,FALSE	FALSE	上升沿，执行电子凸轮
MasterOffset	主轴偏置	LREAL	负数， 正数，0	0	以指定的偏移值移动主轴的相位
SlaveOffset	从轴偏执	LREAL	负数， 正数，0	0	以指定的偏移值移动从轴的相位
MasterScaling	主轴预编译比例	LREAL	＞0.0	1	以指定的比例放大/缩小主轴的相位
SlaveScaling	从轴预编译比例	LREAL	＞0.0	1	以指定的比例放大/缩小从轴轴的相位
StartMode	从轴相对凸轮输出模式	MC_StartMode	-	absolute	0：absolute 绝对位置 1：relative 相对位置 2：ramp_in(斜坡切入) 3：ramp_in_pos（正向斜坡切入） 4：ramp_in_neg 反向斜坡切 入
CamTableID	表格编号	MC_CAM_ID	-	-	定义 cam 表格的使用，与 MC_CamTableSelect 的输出点 CamTableID 配合使用
VelocityDiff	速度	LREAL	-	-	与 ramp_in 不同的最大速度
Acceleration	加速度	LREAL	-	-	ramp_in 时加速度
Deceleration	减速度	LREAL	-	-	ramp_in 时减速度
Jerk	加加速度	LREAL	-	-	ramp_in 的加速度
TappetHysteresis	挺杆阻尼	LREAL	-	-	挺杆的阻尼系数

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
InSync	凸轮生效	BOOL	TRUE,FALSE	FALSE	主轴和从轴建立凸轮关系后，InSync 被置位，当指令的执行条件 OFF 时，InSync 被复位。
Busy	同步运行中	BOOL	TRUE,FALSE	FALSE	Execute 输入上升沿时，置位 TRUE，TRUE 时表示凸轮关系耦合中，需用 Cam_out 指令复位，指令执行条件复位不能复位该状态。

输出变量	名称	数据类型	有效范围	初始值	描述
CommandAborted	指令被中断	BOOL	TRUE,FALSE	FALSE	从轴被其他控制指令中断输出为 TRUE
Error	错误	BOOL	TRUE,FALSE	FALSE	如果检测到有错误，Error 位被置位；当指令的执行条件 OFF 时，Error 位被复位。
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码
EndOfProfile	曲线完成	BOOL	TRUE,FALSE	FALSE	如果 MC_CamTableSelect 指令执行时 Periodic 参数为 0（非周期），当凸轮曲线执行完一次后，EndOfProfile 位被置位，当指令的执行条件 OFF 时，EndOfProfile 位被复位。
Tappets	挺杆表	SMC_TappetData	-	-	关联一个凸轮挺杆，可被 MC_GetTappetValue 指令读出

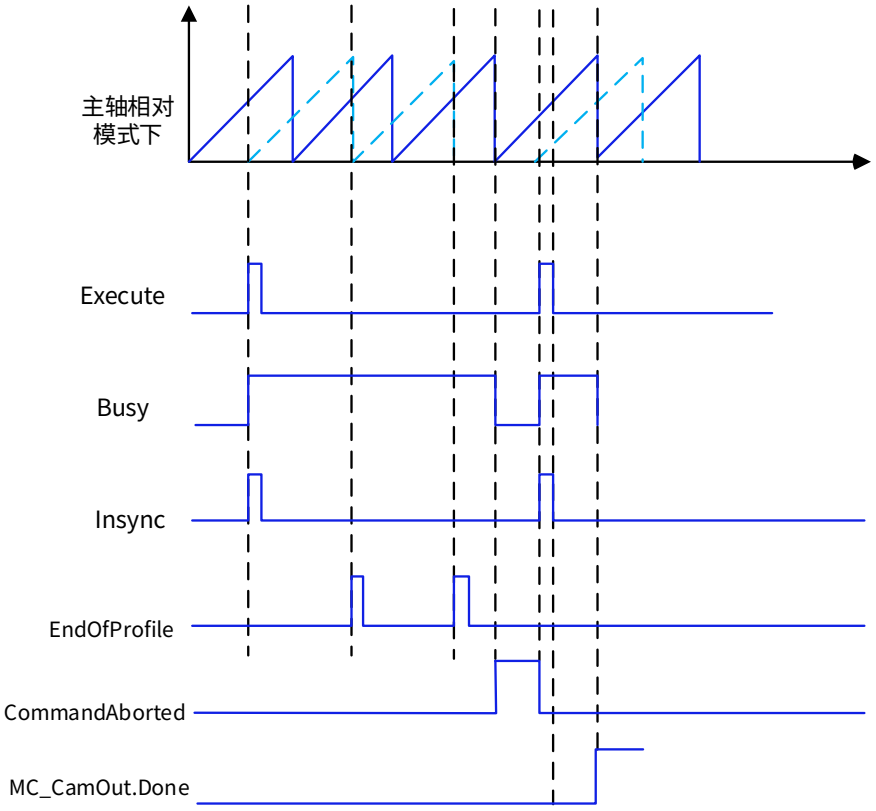
3、 功能说明

在选择正确的凸轮表以及轴无报错的条件下，Execute 上升沿触发该功能块；在一个凸轮运动系统中，要调用一条凸轮曲线，先调用 MC_CamTableSelect 指令选择相应的凸轮表，再执行 MC_CamIn；如要更换凸轮曲线，则再调用 MC_CamTableSelect 指令重新选择凸轮表。需使用 Camout 指令断开主从轴的凸轮耦合关系。该指令执行时,该指令的从轴再执行其它运动指令时，从轴和主轴之间的凸轮关系会解除，并且 Command-Aborted 输出为 TRUE。

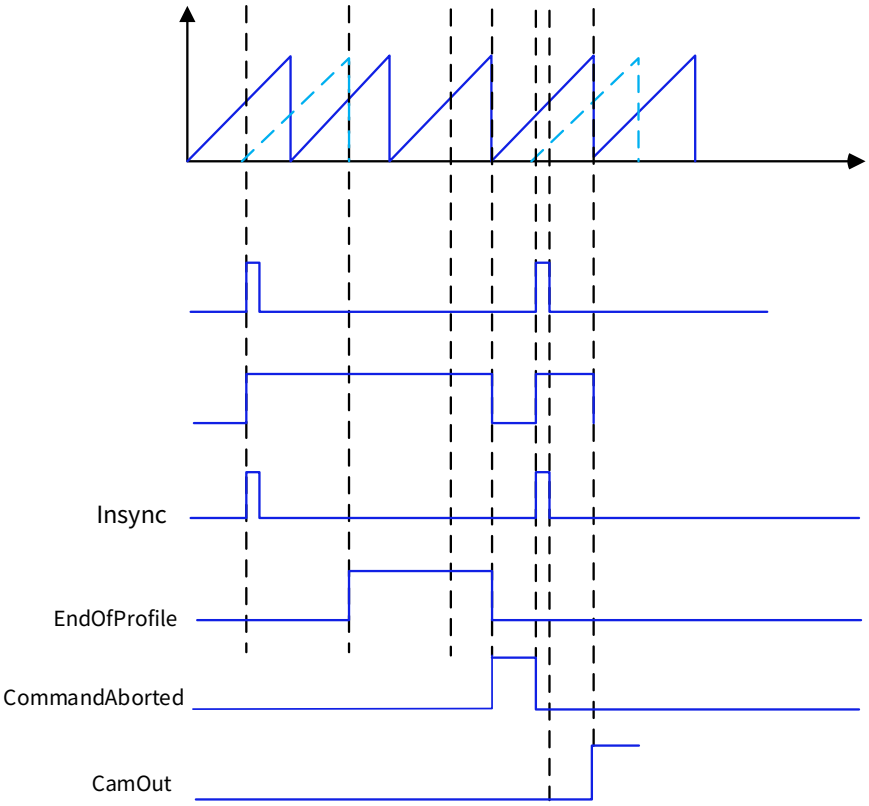
4、 时序图

周期模式（MC_CamTableSelect.Periodic 设为 TRUE）如下图。

注意：MC_Camout 指令只切断主从轴的凸轮耦合关系，如果断开时候从轴速度不为 0，则从轴不会自动减速为 0，需配合使用 MC_STOP 指令。



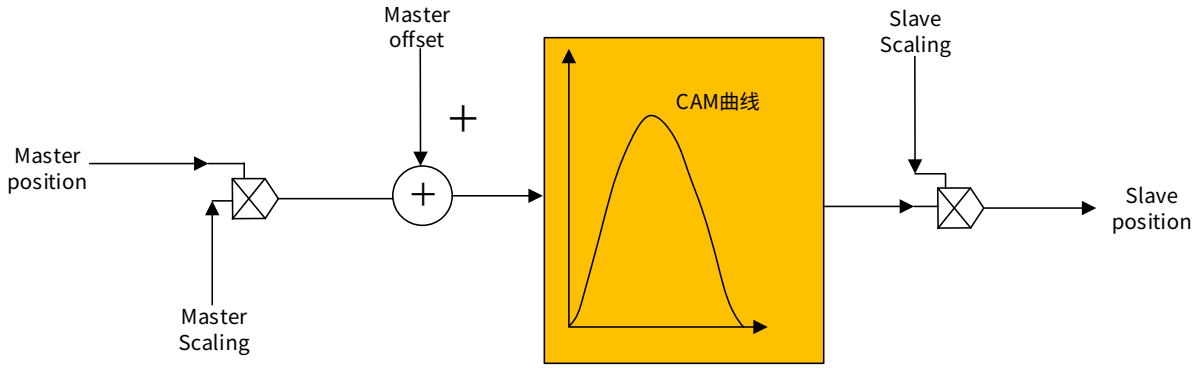
非周期模式（MC_CamTableSelect.Periodic 设为 FALSE）如下图。



5、 功能块详解

指令启动条件：在主轴停止中、位置控制中、速度控制中、同步控制中任意状态下都可以启动本指令。

凸轮曲线中齿合点的计算方法如下：



由上图得出计算方法如下：

Position_Slave=SlaveScaling*CAM(MasterScaling*MasterPosition+MasterOffset)+SlaveOffset

该公式中的主轴位置、从轴位置并不是代表实际物理轴的位置，而是凸轮函数曲线相关的主从轴位置。

主从轴位置跟主从实轴位置之间的关系有详细描述。

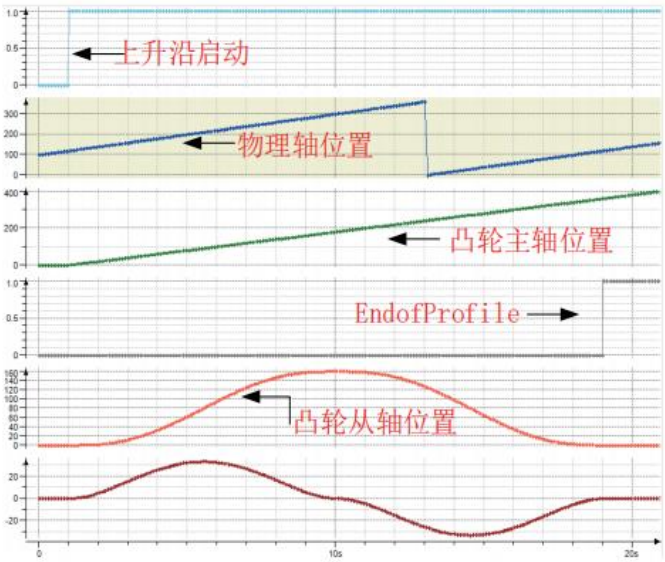
注意：主轴位置、从轴位置均指的是凸轮函数曲线所需的主从轴位置，而不是主轴、从轴物理实轴的位置。

周期模式与 EndOfProfile 的关系：周期模式非周期模式决定了主轴到终止位置后电子凸轮是否要再次进行。

非周期模式：MC_CamTableSelect 指令 Periodic 选择 FALSE，非周期模式时，凸轮完成 EndofProfile 信号输出为 TRUE，执行输入 FALSE 则 EndofProfile 输出 FALSE。此时凸轮只执行一个主轴周期。

注意：主轴周期指的是电子凸轮主轴位置从起始位置到终止位置的范围。

周期模式：MC_CamTableSelect 指令 Periodic 选择 TRUE。

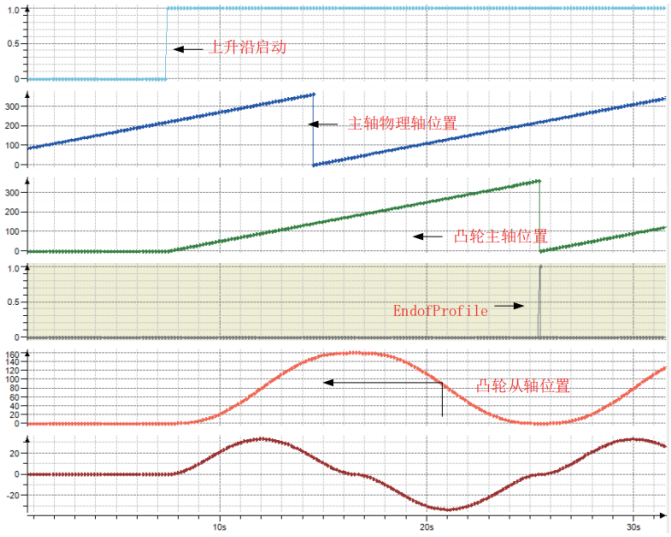


此时凸轮一个主轴周期完成后会接着执行下一个周期，且 EndofProfile 信号 TRUE 输出只维持一个任务周期。

注意：当凸轮主轴位置大于等于凸轮终止位置时 EndofProfile 信号输出为 TRUE，并且凸轮主轴位置更新为凸轮起始位置+当前位置-终止位置。

例如：电子凸轮主轴起始位置为 0、终止位置为 360、主从轴缩放设置为 1、主从轴偏移值设为 0、任务周期 2ms，主轴速度为 100。当某个任务周期凸轮主轴位置为 359.99，那么下个周期 EndofProfile 输出为 TRUE 且主轴位置变为 359.99+100*0.002-360=0.19。

周期模式下设计的凸轮曲线起始位置跟终止位置最好能保持平滑过渡，否则会产生跳变。比如说起始速度为 0，终止速度不为 0，会导致主轴在周期完成和新周期开始处产生跳变。



StartMode 与 MC_CamTableSlect 中主从轴绝对相对模式的关系：

绝对模式：在新的电子凸轮循环开始时，电子凸轮的计算与当前从轴的位置无关。如果从轴相对于主轴起始位置不同于从轴相对于主轴的终止位置，其将造成一个跳变。

相对模式：新的电子凸轮会根据当前从轴的位置进行改变。也就是说，从轴在上一个电子凸轮循环结束的位置，会被现在的电子凸轮运动作为“从轴偏移”进行位置相加的计算。但是，如果在电子凸轮定义中，与主轴起始位置对应的从轴位置不是 0，其将造成一个跳变。

斜坡输入：通过增加一个补偿运动（根据极限值 VelocityDiff、加速度、减速度得出的运动）来防止电子凸轮在开始时的潜在跳变。因此，只要从轴是旋转的方式，正向斜坡输入选项只进行正向补偿，而反向斜坡输入则只进行反向补偿。对于线性运动的从轴，补偿方向可以自动实现，也就是说，正向斜坡输入和反向斜坡输入可以被用斜坡输入的方式进行解释。

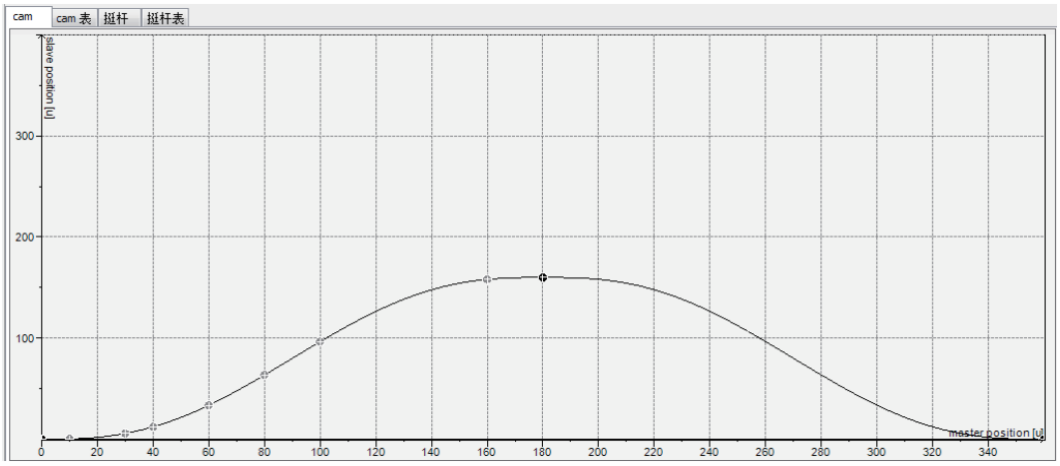
关系表如下所示：

MC_CamTableSelect.MasterAbsolute	主轴模式
absolute	绝对模式
relative	相对模式

MC_CamIn.StartMode	MC_CamTableSelect.SlaveAbsolute	从轴模式
absolute	TRUE	绝对模式
absolute	FALSE	相对模式
relative	TRUE	相对模式
relative	FALSE	相对模式
ramp_in	TRUE	斜坡切入绝对模式
ramp_in	FALSE	斜坡切入相对模式
ramp_in_pos	TRUE	正向斜坡切入绝对模式
ramp_in_pos	FALSE	正向斜坡切入相对模式
ramp_in_neg	TRUE	反向斜坡切入绝对模式
ramp_in_neg	FALSE	反向斜坡切入相对模式

详细关系描述如下：

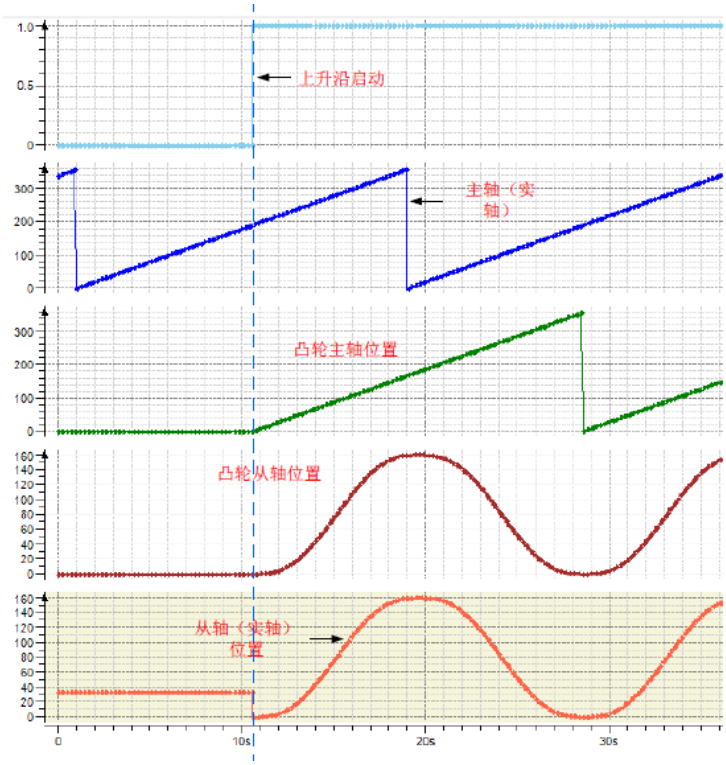
凸轮主轴范围（0-360）、凸轮从轴范围为（0-180）、周期模式、主从轴偏移值 0、主从轴缩放比 1。设计的凸轮表如下图所示。



StartMode 为 0（绝对模式）

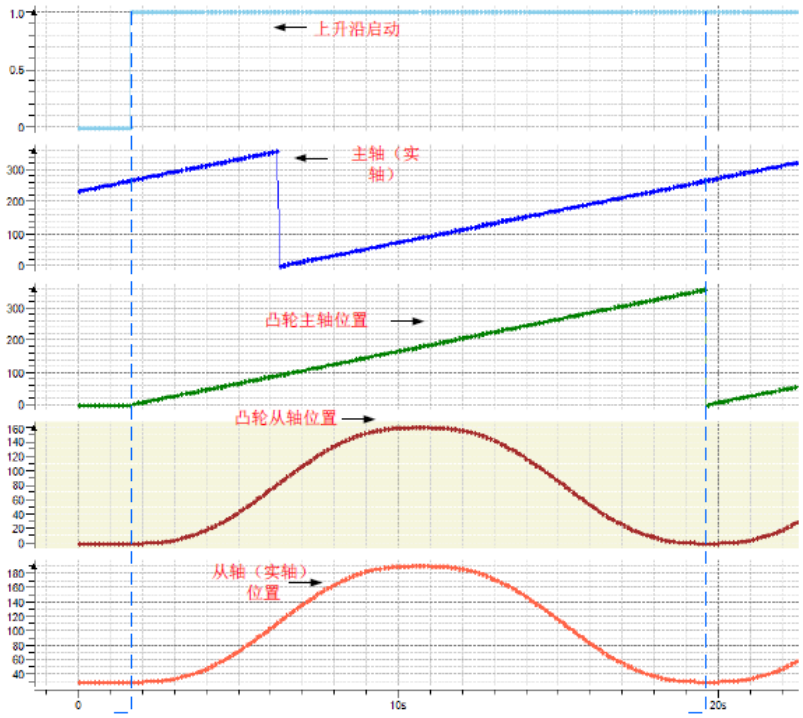
当 MC_CamTableSlect 指令 MasterAbsolute 设置为 FALSE，SlaveAbsolute 设置为 TRUE 时，此时主轴工作在相对模式，从轴工作在绝对模式。当 Excute 上升沿，凸轮启动时，凸轮主轴从凸轮表“起始位置”（0）开始，凸轮从轴按照上述的“凸轮表齿合公式”计算输出，从轴实轴指令位置等于齿合计算输出值。比如说凸轮从轴起始位置为 0，凸轮启动时从轴实轴位置为 20，则启动开始从轴实轴的位置指令为 0 产生跳变。

注意：该情况下从轴（实轴）开始位置不在凸轮从轴开始位置则会产生跳变。



当 MC_CamTableSlect 指令 MasterAbsolute 设置为 FALSE，SlaveAbsolute 设置为 FALSE 时，此时主轴工作在相对模式，从轴工作在相对模式。当 Excute 上升沿，凸轮启动时，凸轮主轴从凸轮表“起始位置”（0）开始，凸轮从轴按照上述的“凸轮表齿合公式”计算输出，从轴实轴指令位置等于齿合计算输出值（凸轮从轴位置）+启动时从轴实轴位置。

比如说凸轮启动时从轴实轴位置为 20，凸轮表从轴起始位置为 0，则启动凸轮时从轴实轴位置指令为 20，后续为 20+凸轮表计算值，最高峰值为 20+凸轮表计算最大值（此处为 180）=200。

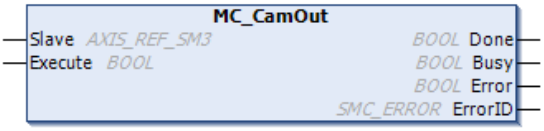


7.2.2 MC_Camout

MC_Camout：断开从轴的凸轮耦合关系。

注意：执行该指令后从轴会按照分离前的速度继续运行，所以需要跟 MC_Stop 等指令配合使用。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_Camout	断开凸轮耦合		MC_CamOut(Slave:=, Execute:=, Done=>, Busy=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Slave	从轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行凸轮功退出能块	BOOL	TRUE,FALSE	FALSE	上升沿，执行电子凸轮

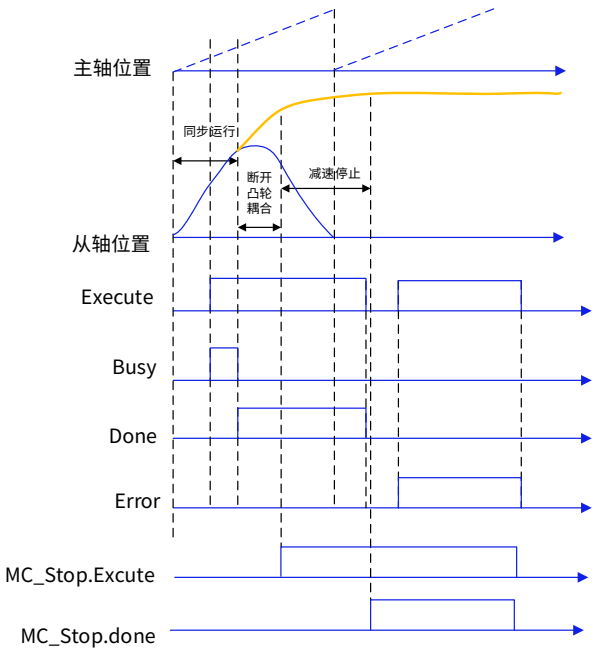
输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	完成	BOOL	TRUE,FALSE	FALSE	完成与主轴的凸轮耦合断开
Busy	同步运行中	BOOL	TRUE,FALSE	FALSE	指令执行中
Error	错误	BOOL	TRUE,FALSE	FALSE	如果检测到有错误，Error 位被置位
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

执行本指令解除从轴的凸轮耦合关系。Excute 上升沿时执行从轴的凸轮耦合关系断开；凸轮关系断开后，从轴并不一定是停止的。如果从轴在执行该指令前速度不为 0，则指令 DONE 信号完成后凸轮耦合关系断开，但是从轴仍然按照凸轮关系断开前速度运行。假如从轴没有凸轮耦合关系时执行该指令，输出 Error。

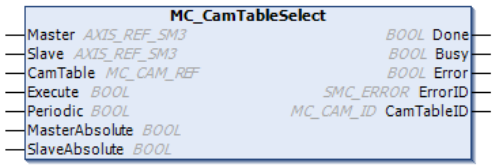
4、 时序图



7.2.3 MC_CamTableSelect

MC_CamTableSelect：指定凸轮表，与 MC_CamIn 指令配合使用。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_CamTableSelect	凸轮表指定		MC_CamTableSelect(Master:=, Slave:=, CamTable:=, Execute:=, Periodic:=, MasterAbsolute:=, SlaveAbsolute:=, Done=>, Busy=>, Error=>, ErrorID=>, CamTableID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Master	主轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例
Slave	从轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例
CamTable	选择表	MC_CAM_REF	-	-	映射到 CAM 表格描述，即 MC_CAM_REF 的一个实例

注意：主轴和从轴不能指定为同一轴，否则会有报错输出，CamTable 所对应的凸轮表编辑需要正确无误，否则也会导致指令报错。主轴、从轴可以为实轴也可以是虚轴。

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行	BOOL	TRUE,FALSE	FALSE	上升沿，执行电子凸轮
Periodic	重复模式	BOOL	TRUE,FALSE	FALSE	指定是反复执行指定的凸轮表还是只执行一次： TRUE：重复 FALSE：不重复
MasterAbsolute	主轴绝对模式	BOOL	TRUE,FALSE	FALSE	指定主轴跟踪距离坐标系是按绝对位置还是相对位置： 1：绝对位置 0：相对位置
SlaveAbsolute	从轴绝对模式	BOOL	TRUE,FALSE	FALSE	与 MC_CamIn 指令中的 StartMode 综合指定从轴当前指令位置为凸轮表输出的绝对（当前主轴位置对应的凸轮表输出值）还是相对（凸轮表输出值叠加指令开始时从轴位置）： 1：绝对位置；0：相对位置

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	完成	BOOL	TRUE,FALSE	FALSE	完成与主轴的凸轮耦合断开
Busy	同步运行中	BOOL	TRUE,FALSE	FALSE	指令执行中
Error	错误	BOOL	TRUE,FALSE	FALSE	如果检测到有错误，Error 位被置位
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码
CamTableID	生效 CAMID	MC_CAM_ID	-	-	选择生效的 Cam_ID，与 MC_CamIn 指令中的 CamTableID 配合使用

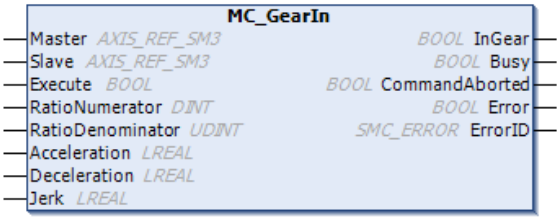
3、 功能说明

本指令指定电子凸轮运行所需凸轮表，所以在使用本指令之前先要将凸轮表编辑好（凸轮编辑器编辑或者在线编辑好）。Excute 上升沿，执行指定凸轮表，亦可凸轮表更新后刷新指定凸轮表；Done 信号输出为 TRUE 时，则输出变量“CamTableID”生成并且生效；指令执行中，Busy 信号输出 TRUE，Done 信号 TRUE、Busy 信号为 FALSE；MasterAbsolute、SlaveAbsolute、Periodic 参数具体作用参考 MC_CamIn 指令详情。

7.2.4 MC_GearIn

MC_GearIn：设定从轴与主轴间的齿轮比，进行电子齿轮动作。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_GearIn	电子齿轮功能块		MC_GearIn(Master:=, Slave:=, Execute:=, RatioNumerator:=, RatioDenominator:=, Acceleration:=, Deceleration:=, Jerk:=, InGear=>, Busy=>, CommandAborted=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Master	主轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例
Slave	从轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行	BOOL	TRUE,FALSE	FALSE	上升沿，触发功能块
RatioNumerator	齿轮比分子	DINT	正数、负数	1	齿轮比分子
RatioDenominator	齿轮比分母	UDINT	正数	1	齿轮比分母
Acceleration	加速度	LREAL	正数或 0	-	指定加速度

输入变量	名称	数据类型	有效范围	初始值	描述
Deceleration	减速度	LREAL	正数或 0	-	指定减速度
Jerk	跃度	LREAL	正数或 0	-	加加速度

输出变量

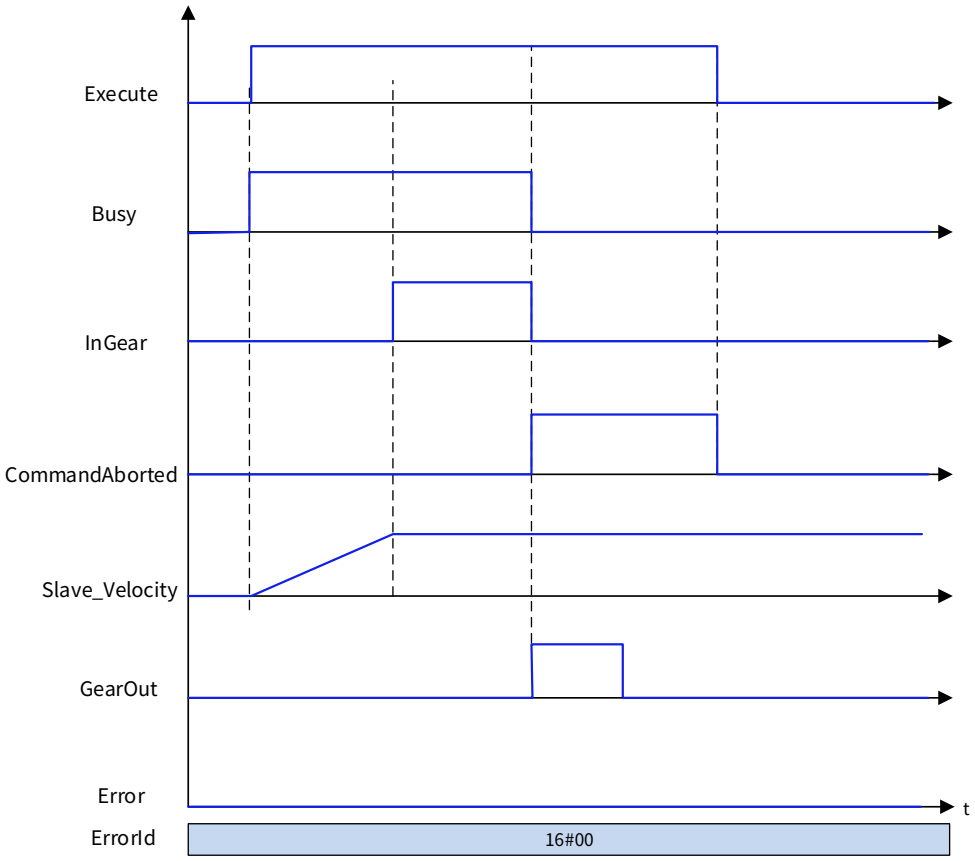
输出变量	名称	数据类型	有效范围	初始值	描述
InGear	齿轮比到达	BOOL	TRUE,FALSE	FALSE	TRUE，从轴达到目标速度
Busy	同步运行中	BOOL	TRUE,FALSE	FALSE	指令执行中。
CommandAborted	中断	BOOL	TRUE,FALSE	FALSE	TRUE，被其他控制指令中断
Error	错误	BOOL	TRUE,FALSE	FALSE	如果检测到有错误，Error 位被置位
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

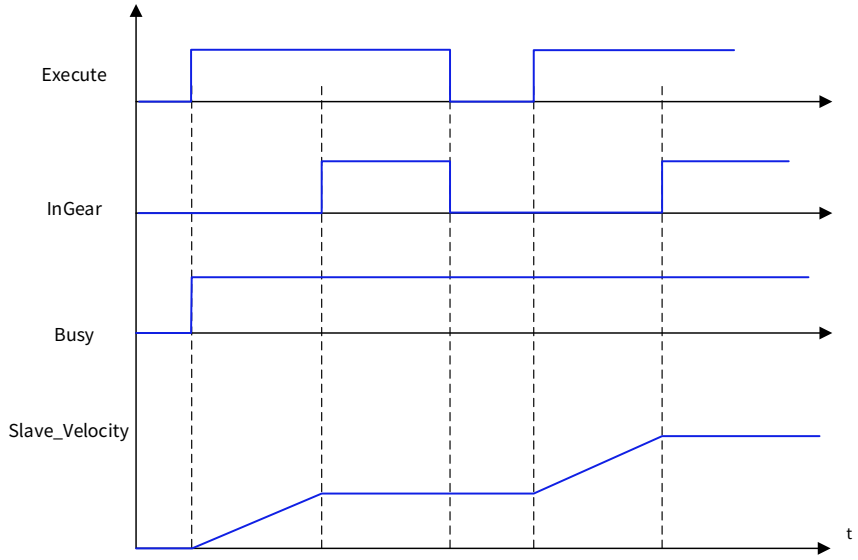
Execute 上升沿，开始电子齿轮动作。执行电子齿轮后要解除耦合必须通过 GearOut 指令。该指令为速度电子齿轮功能，加速过程中造成的同步距离丢失不会自动补偿。指令执行中 Busy 信号为 TRUE 时，如果从轴目标速度没有达到此时 Execute 新的上升沿不会影响它。指令执行中 Busy 信号为 TRUE 时，如果从轴目标速度达到此时 Execute 新的上升沿不会影响它。到达目标速度，InGear 为 TRUE，此后从轴移动量=主轴移动量 *RatioNumerator/RatioDenominator。如果主轴速度实时变化的情况下请注意慎重使用该指令。

注意：执行指令过程中请不要使用 MC_SetPosition 指令以免引起电机急速运转产生事故。

4、 时序图



变更齿轮比参数后重启指令时序图如下：



7.2.5 MC_GearOut

MC_GearOut：终止执行中的 MC_GearIn,MC_GearInPos 指令。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_GearOut	电子齿轮耦合断开		MC_GearOut(Slave:=, Execute:=, Done=>, Busy=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Slave	从轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行	BOOL	TRUE,FALSE	FALSE	上升沿触发功能块

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	完成	BOOL	TRUE,FALSE	FALSE	为 TRUE 时从轴与主轴电子齿轮耦合断开
Busy	同步运行中	BOOL	TRUE,FALSE	FALSE	指令执行中
Error	错误	BOOL	TRUE,FALSE	FALSE	如果检测到有错误，Error 位被置位
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

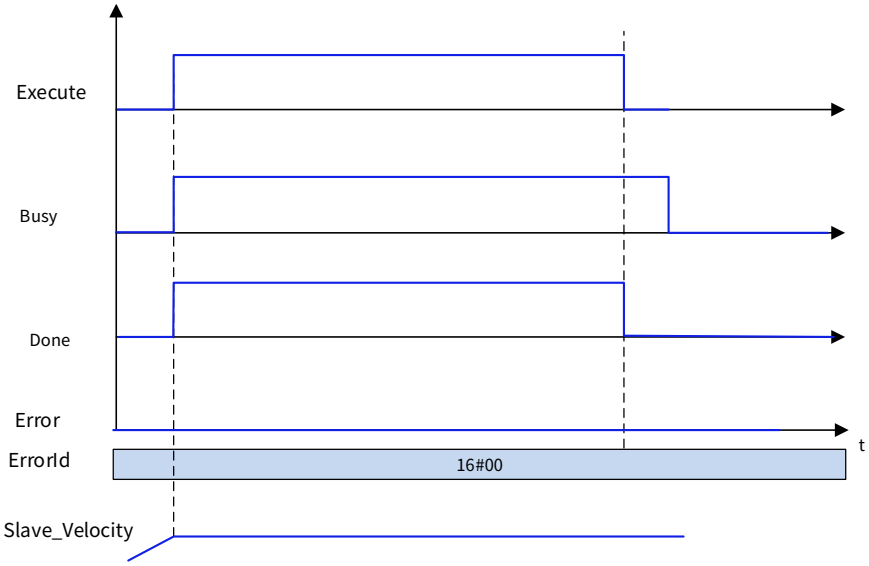
3、 功能说明

Execute 上升沿，执行切出电子齿轮动作。Excute 为 TRUE，ERROR 为 FALSE 则 Busy 输出为 TRUE，Done 输出

为 TRUE。

切出电子齿轮动完成后此时从轴的速度为切出前的速度，所以需配合以 MC_Stop 指令停止从轴。Execute 下降沿则，Done 为 FALSE。

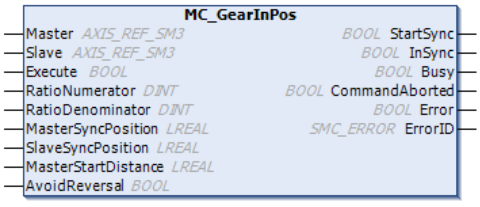
4、 时序图



7.2.6 MC_GearInPos

MC_GearInPos：设定主轴与从轴之间的电子齿轮比，进行电子齿轮动作。指定开始同步的主轴位置、从轴位置、主轴开始同步距离，并以此来完成切入电子齿轮动作。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_GearInPos	指定位置切入电子齿轮耦合		MC_GearInPos(Master:=, Slave:=, Execute:=, RatioNumerator:=, RatioDenominator:=, MasterSyncPosition:=, SlaveSyncPosition:=, MasterStartDistance:=, AvoidReversal:=, StartSync=>, InSync=>, Busy=>, CommandAborted=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Master	主轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例
Slave	从轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行	BOOL	TRUE,FALSE	FALSE	上升沿，触发功能块
RatioNumerator	齿轮比分子	DINT	TRUE,FALSE	-	主从轴速度比的分子
RatioDenominator	齿轮比分母	DINT	-	-	主从轴速度比的分母
MasterSync Position	主轴同步位置	LREAL	-	-	主从轴齿轮比耦合时主轴位置
SlaveSyncPosition	从轴同步位置	LREAL	-	-	主从轴齿轮比耦合时从轴位置
MasterStartDistance	执行同步主轴位置	LREAL	-	-	从轴按照该位置值和-MasterSync Position 以及 SlaveSyncPosition 值计算一条平滑曲线使从轴在 SlaveSyncPosition 时跟主轴齿轮同步，曲线主轴范围为 [MasterStartDistance，MasterSyncPosition]
AvoidReversal	禁止反转	BOOL	TRUE,FALSE	FALSE	设置为 FALSE，如果从轴物理位置超前的情况下进行反转。设置为 TRUE 如果从轴物理上不能实现反转或者导致危险。只在模态轴下适用。如果反转不能被避免，那么轴将错误停止。

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
StartSync	开始耦合处理	BOOL	TRUE,FALSE	FALSE	TRUE,开始电子齿轮耦合处理
InSync	耦合中	BOOL	TRUE,FALSE	FALSE	TRUE,电子齿轮耦合处理完成，主从轴齿轮比耦合中
Busy	同步运行中	BOOL	TRUE,FALSE	FALSE	指令执行中
CommandAborted	指令中断	BOOL	TRUE,FALSE	FALSE	被其它控制指令中断
Error	错误	BOOL	TRUE,FALSE	FALSE	如果检测到有错误，Error 位被置位
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

Execute 上升沿信号，开始执行指令。

开始动作后，Slave（从轴）以 Master（主轴）速度乘以齿轮比得到的速度为目标速度，进行加减速动作。

该功能块同步开始到同步结束的过程本质为同步区间内从轴跟随主轴的一个电子凸轮、此时根据主轴范围（MasterSyncPosition-MasterStartDistance，MasterSyncPosition），从轴范围（当前位置，SlaveSyncPosition），指令会根据设置齿轮比以及上述三个参数自动设计一条凸轮曲线，执行同步时从轴跟随主轴完成凸轮动作。

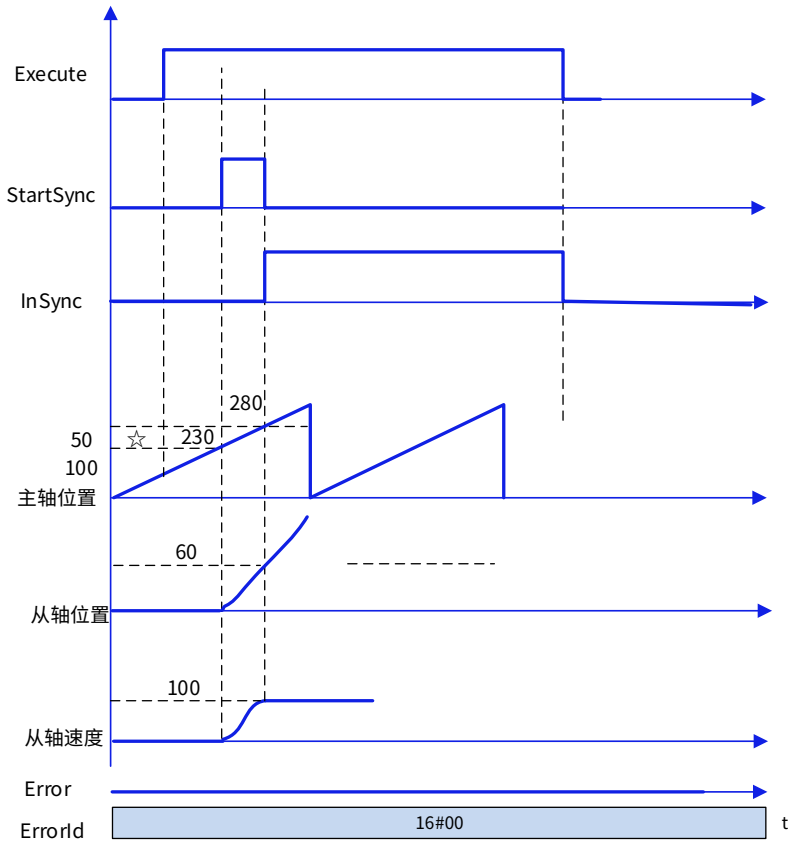
注意：如果主从轴工作在线性模式，需保证需保证上述几个参数设置合理，否则齿轮动作无法正确进行。故而建议使用该指令时主从轴工作在周期模式。

例如：主从轴线性工作模式都向正向运动，如果执行指令时主轴位置>MasterSyncPosition-MasterStartDistance，或者从轴位置>SlaveSyncPosition 则无法切入电子齿轮动作。

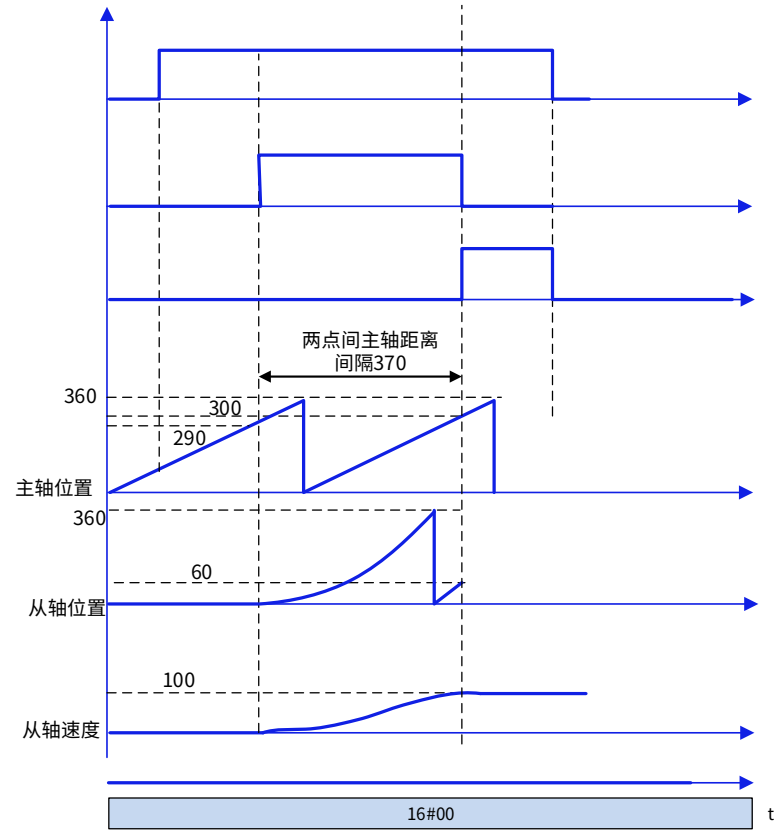
下面给出了几种不同参数下的样例时序图：

当主轴工作在周期模式（360 循环）、从轴工作在周期模式（360 循环）。

- MasterSyncPosition=280、MasterStartDistance=50、SlaveSyncPosition=60，主轴速度为 50、AvoidReversal=FALSE。

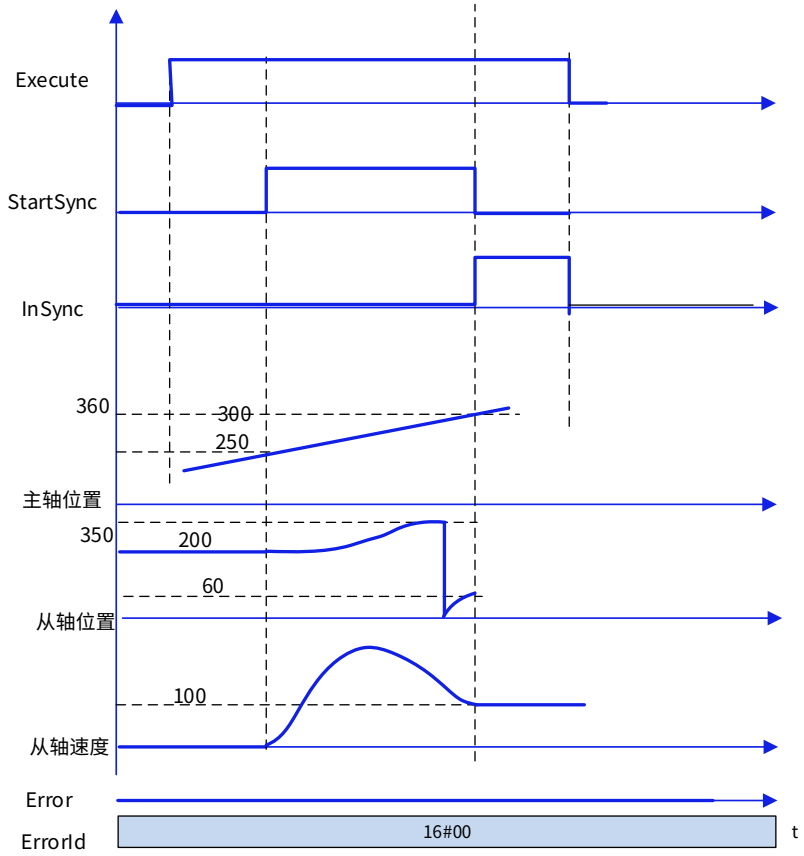


- MasterSyncPosition=300、MasterStartDistance=370、SlaveSyncPosition=60，主轴速度为 50、AvoidReversal=FALSE。



- MasterSyncPosition=300、MasterStartDistance=50、SlaveSyncPosition=60，主轴速度为 50、

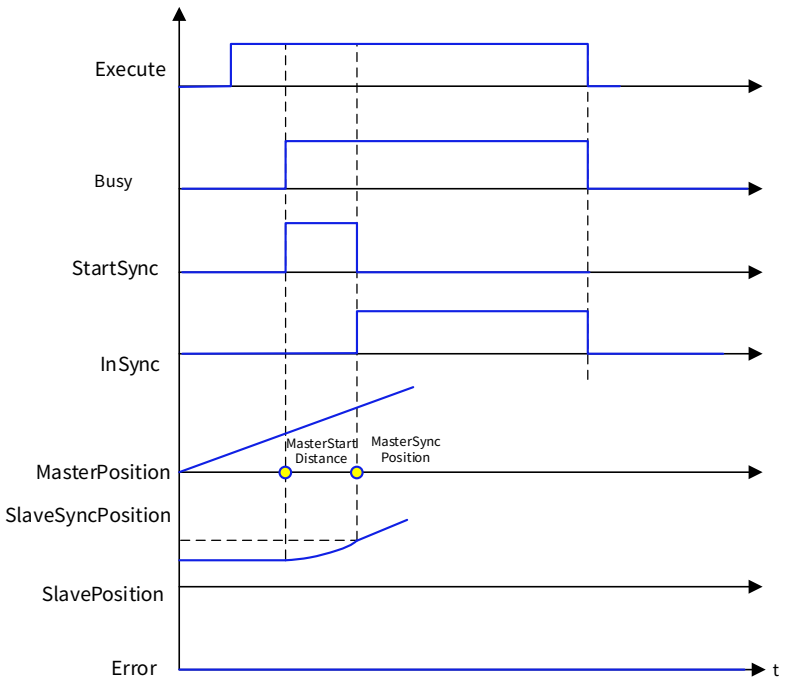
AvoidReversal=FALSE、从轴起始位置大于 60。



同步完成（InSync 为 TRUE）的同时达到目标速度，此后从轴移动量=主轴移动量 *RatioNumerator/RatioDenominator。

对于 AvoidReversal：如果从轴是模态轴并且主轴速度（齿轮比的倍数关系）不是相对于从轴的速度关系，那么 MC_GearInPos 会尝试着避免从轴的反转。它试图通过增加 5 个从轴周期“拉伸”从轴的运动。如果这个“拉伸”无效，那么会有错误出现并且从轴错误停止。如果从轴速度关联主轴速度（是齿轮比的倍数），那么会有错误出现，并且轴错误停止。如果从轴是线性轴模式轴，那么一个错误会产生在 Execute 输入上升沿处理时。

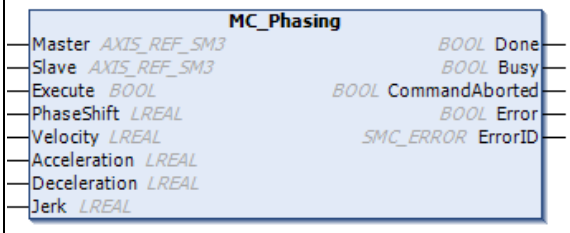
4、 时序图



7.2.7 MC_Phasing

MC_Phasing：指定主从轴之间的相位偏差。

1、 指令格式

指令	名称	图形表现	ST 表现
MC_Phasing	电子齿轮耦合断开		MC_Phasing(Master:=, Slave:=, Execute:=, PhaseShift:=, Velocity:=, Acceleration:=, Deceleration:=, Jerk:=, Done=>, Busy=>, CommandAborted=>, Error=>, ErrorID=>);

2、 相关变量

输入输出变量

输入输出变量	名称	数据类型	有效范围	初始值	描述
Master	主轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例
Slave	从轴	AXIS_REF	-	-	映射到轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	描述
Execute	执行	BOOL	TRUE,FALSE	FALSE	上升沿，触发功能块
PhaseShift	主从轴相位偏差值	LREAL	-	0	主从轴相位偏差值，正数代表从轴滞后
Velocity	速度	LREAL	-	0	执行相位偏移时最大速度值
Acceleration	加速度	LREAL	-	0	执行相位偏移时最大加速度值
Deceleration	减速度	LREAL	-	0	执行相位偏移时最大减速度值
Jerk	速度二次导数	LREAL	-	0	执行相位偏移时最大 Jerk 值

输出变量

输出变量	名称	数据类型	有效范围	初始值	描述
Done	完成	BOOL	TRUE,FALSE	FALSE	TRUE，从轴与主轴电子齿轮耦合断开
Busy	同步运行中	BOOL	TRUE,FALSE	FALSE	指令执行中
Command Aborted	指令中断	BOOL	TRUE,FALSE	FALSE	被其它控制指令中断
Error	错误	BOOL	TRUE,FALSE	FALSE	如果检测到有错误，Error 位被置位
ErrorID	错误代码	SMC_ERROR	-	0	发生异常时，输出错误代码

3、 功能说明

Execute 上升沿执行相位偏移，从轴自动计算一条平滑曲线，完成从轴对主轴的相位偏移，主从轴相位差为输入信号的 PhaseShift 值，正值为从轴滞后于主轴。

完成偏移后 Done 信号输出为 TRUE。

根据设定的 PhaseShift、Velocity、Acceleration、Deceleration 对主从轴相位差进行补偿。

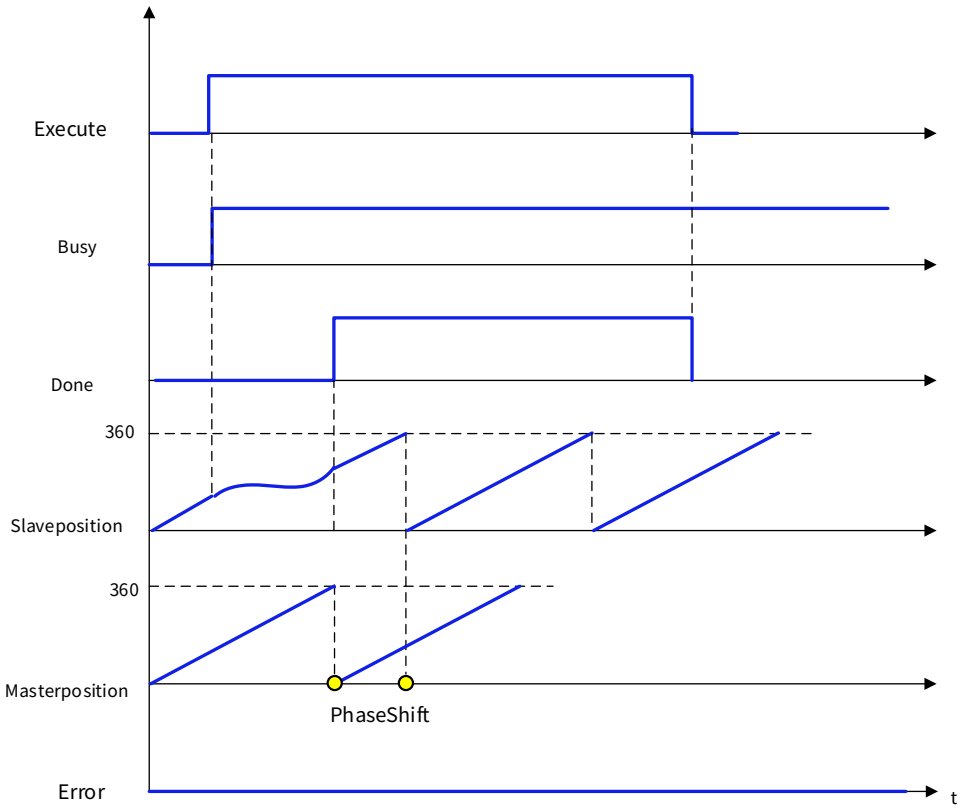
主从轴相位差达到 PhaseShift 时，Done 信号输出。

执行指令时主轴指令位置与反馈位置不变，从轴进行调整，完成后，从轴跟主轴之间相位差为 PhaseShift。

该指令最终结果为轴给定值之间的相位偏移，所以实轴的实际反馈值与最终的偏移可能不一致。

4、 时序图

以主从轴都按 360 周期运动，Execute 信号上升沿执行调整，调整完成后从轴与主轴之间相位偏差为 PhaseShift 设定的值。



8 通信指令

自由口通信是一种基于点对点的通信方式，它采用两端对等的方式进行数据传输，每一端都可以接收和发送数据。它的传输方式是全双工的，即可以在两端同时发送和接收数据。

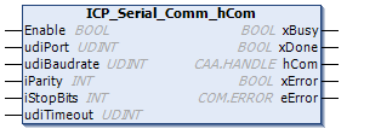
8.1 串口自由口指令

8.1.1 指令列表

指令类别	名称	功能
RS485 自由口通信指令	ICP_Serial_Comm_hCom	创建 RS485 自由口通信连接
	ICP_Serial_Comm_Read	RS485 自由口通信读数据
	ICP_Serial_Comm_Write	RS485 自由口通信写数据

8.1.2 ICP_Serial_Comm_hCom

指令格式：

指令	名称	图形表现	ST 表现
ICP_Serial_Comm_hCom	RS485 自由口通信连接指令		ICP_Serial_Comm_hCom(Enable:=, udiPort:=, udiBaudrate:=, iParity:=, iStopBits:=, udiTimeout:=, xBusy=>, xDone=>, hCom=>, xError=>, eError=>);

相关变量：

输入类型	名称	数据类型	有效范围	初始值	描述
Enable	使能	BOOL	TRUE,FALSE	FALSE	高电平使能功能块
udiPort	硬件串口号	UDINT	-	-	对应硬件串口 1 和 2
udiBaudrate	波特率	UDINT	-	-	串口波特率
iParity	校验位	INT	-	-	串口校验位
iStopBits	停止位	INT	-	-	串口停止位
udiTimeout	超时时间	UDINT	-	-	超时时间
输出类型	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
xBusy	功能块运行	BOOL	TRUE,FALSE	FALSE	运行标志
xDone	完成信号	BOOL	TRUE,FALSE	FALSE	完成标志
hCom	句柄	CAA.HANDLE	-	-	通讯建立句柄
xError	错误标志	BOOL	TRUE,FALSE	FALSE	错误
eError	错误	COM.ERROR	-	0	错误码

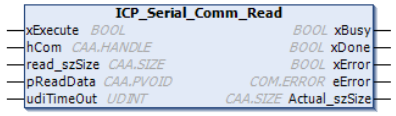
程序示例：

RS485 自由口通信连接指令，当 ICP_Serial_Comm_hCom 指令 Enable 为 TRUE 时，会将创建 RS485 自由口通信有效句柄（hCom 大于 0），xBusy 以及 xDone 为 TRUE。

```
////打开串口
ICP_Serial_Comm_hCom_1(
    Enable:= Open,                                  //功能块使能
    udiPort:= 2,                                    //PLC硬件端口号
    udiBaudrate:= 19200,                            //波特率
    iParity:= COM.PARITY.NONE ,                     //校验位     0=COM.PARITY.EVEN, 1=COM.PARITY.ODD, 2=COM.PARITY.NONE ,
    iStopBits:= COM.STOPBIT.ONESTOPBIT ,            //停止位     0=COM.STOPBIT.ONESTOPBIT, 1=ONE5STOPBITS, 2=COM.STOPBIT.TWOSTOPBITS,
    udiTimeout:= 1000,                              //超时时间
    xBusy=> ,                                   //功能块正在执行
    xDone=> ,                                   //功能块完成信号
    hCom=> ,                                    //通讯建立句柄
    xError=> ,                                  //功能块错误
    eError=> );                                 //功能块错误代码
```

8.1.3 ICP_Serial_Comm_Read

指令格式：

指令	名称	图形表现	ST 表现
ICP_Serial_Comm_Read	RS485 自由口通信读数据		ICP_Serial_Comm_Read(xExecute:=, hCom:=, read_szSize:=, pReadData:=, udiTimeOut:=, xBusy=>, xDone=>, xError=>, eError=>, actual_szSize=>);

相关变量：

输入类型	名称	数据类型	有效范围	初始值	描述
xExecute	触发	BOOL	TRUE,FALSE	FALSE	上升沿触发功能块
hCom	连接句柄	CAA.HANDLE	-	-	通讯建立句柄
read_szSize	数据长度	CAA.SIZE	-	-	读数据长度
pReadData	数据存储	CAA.PVOID	-	-	读数据存储地址
udiTimeOut	超时时间	UDINT	-	-	通讯超时时间
输出类型	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
xBusy	功能块运行	BOOL	TRUE,FALSE	FALSE	功能块运行标志
xDone	完成标志	BOOL	TRUE,FALSE	FALSE	完成标志
xError	错误标志	BOOL	TRUE,FALSE	FALSE	读错误
eError	错误	COM.ERROR	-	0	错误码
actual_szSize	数据长度	CAA.SIZE	-	-	读数据实际长度

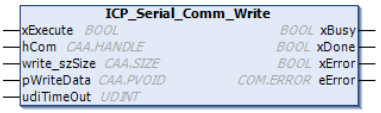
程序示例：

RS485 自由口通信读数据指令。当 ICP_Serial_Comm_Read 指令 xExecute 为 TRUE 时，将从 RS485 自由口通信缓冲区读取数据，xBusy 为 TRUE。若读取成功，xDone 置位一个扫描周期；读取的数据会被放置到地址为 pReadData 的变量里。

```
//读数据模块
ICP_Serial_Comm_Read_1(
    xExecute:= Read_xExecute,                       //上升沿触发功能块
    hCom:= ICP_Serial_Comm_hCom_1.hCom,             //通讯建立句柄
    read_szSize:= read_szSize_1,                    //读数据长度
    pReadData:= ADR(bReadData_1),                   //读数据指针
    udiTimeOut:= ,                                  //超时时间
    xBusy=> ,                                       //功能块正在执行
    xDone=> ,                                       //功能块完成信号
    xError=> ,                                      //功能块错误
    eError=> ,                                      //功能块错误代码
    actual_szSize=> );                              //实际读数据长度
```

8.1.4 ICP_Serial_Comm_Write

指令格式：

指令	名称	图形表现	ST 表现
ICP_Serial_Comm_Write	RS485 自由口通信写数据		ICP_Serial_Comm_Write(xExecute:=, hCom:=, write_szSize:=, pWriteData:=, udiTimeOut:=, xBusy=>, xDone=>, xError=>, eError=>);

相关变量：

输入类型	名称	数据类型	有效范围	初始值	描述
xExecute	触发	BOOL	TRUE,FALSE	FALSE	上升沿触发功能块
hCom	连接句柄	CAA.HANDLE	-	-	通讯建立句柄
write_szSize	数据长度	CAA.SIZE	-	-	写数据长度
pWriteData	数据存储	CAA.PVOID	-	-	写数据存储地址
udiTimeOut	超时时间	UDINT	-	-	通讯超时时间
输出类型	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
xBusy	功能块运行	BOOL	TRUE,FALSE	FALSE	功能块运行标志
xDone	完成标志	BOOL	TRUE,FALSE	FALSE	完成标志
xError	错误标志	BOOL	TRUE,FALSE	FALSE	写错误
eError	错误	COM.ERROR	-	0	错误码

程序示例：

RS485 自由口通信读数据指令。当 ICP_Serial_Comm_Write 指令 xExecute 为 TRUE 时，会将用户设置的发送缓冲区 pWriteData 为首地址的长度为 write_szSize 的数据发送到目标设备。如果在超时时间 udiTimeOut 内发送成功，则 Done 置 TRUE。

```
30  //写数据模块
31  ICP_Serial_Comm_Write_1(
32      xExecute FALSE := Write_xExecute FALSE ,                    //上升沿触发功能块
33      hCom 0 := ICP_Serial_Comm_hCom_1.hCom 0 ,                   //通讯建立句柄
34      write_szSize 0 := write_szSize_1 6 ,                        //写数据长度
35      pWriteData 0 := ADR(bWriteData_1),                          //写数据指针
36      udiTimeOut:= ,                                              //超时时间
37      xBusy=> ,                                                   //功能块正在执行
38      xDone=> ,                                                   //功能块完成信号
39      xError=> ,                                                  //功能块错误
40      eError=> );                                                 //功能块错误代码
```

参考程序逻辑：

- 变量声明

```
PROGRAM COM2
VAR
    ICP_Serial_Comm_hCom_1              :ICP_Serial_Comm_hCom.ICP_Serial_Comm_hCom;
    ICP_Serial_Comm_Read_1              :ICP_Serial_Comm_Read.ICP_Serial_Comm_Read;
    ICP_Serial_Comm_Write_1             :ICP_Serial_Comm_Write.ICP_Serial_Comm_Write;

    //变量
    Open                                :BOOL:=TRUE;                    //使能功能块
    Read_xExecute                       :BOOL;
    Write_xExecute                      :BOOL;

    bReadData_1                         :ARRAY[1..6] OF BYTE;           //读数据
    read_szSize_1                       :CAA.SIZE:=6;                   //读数据长度

    bWriteData_1                        :ARRAY[1..6] OF BYTE:=[7,8,9,4,5,6];        //写数据
    write_szSize_1                      :CAA.SIZE:=6;                   //写数据长度

    //状态机
    iState                              :UINT;


END_VAR
```

- 程序例程

```
//建议任务配置间隔时间大于10mS,否则无法读数据

////打开串口
ICP_Serial_Comm_hCom_1(
    Enable:= Open,                                  //功能块使能
    udiPort:= 2,                                    //PLC硬件端口号
    udiBaudrate:= 19200,                            //波特率
    iParity:= COM.PARITY.NONE ,                     //校验位    0=COM.PARITY.EVEN, 1=COM.PARITY.ODD, 2=COM.PARITY.NONE ,
    iStopBits:= COM.STOPBIT.ONESTOPBIT ,            //停止位    0=COM.STOPBIT.ONESTOPBIT, 1=ONE5STOPBITS, 2=COM.STOPBIT.TWOSTOPBITS,
    udiTimeout:= 1000,                              //超时时间
    xBusy=> ,                                       //功能块正在执行
    xDone=> ,                                       //功能块完成信号
    hCom=> ,                                        //通讯建立句柄
    xError=> ,                                      //功能块错误
    eError=> );                                     //功能块错误代码

//读数据模块
ICP_Serial_Comm_Read_1(
    xExecute:= Read_xExecute,                       //上升沿触发功能块
    hCom:= ICP_Serial_Comm_hCom_1.hCom,             //通讯建立句柄
    read_szSize:= read_szSize_1,                    //读数据长度
    pReadData:= ADR(bReadData_1),                   //读数据指针
    udiTimeOut:= ,                                  //超时时间
    xBusy=> ,                                       //功能块正在执行
    xDone=> ,                                       //功能块完成信号
    xError=> ,                                      //功能块错误
    eError=> ,                                      //功能块错误代码
    actual_szSize=> );                              //实际读数据长度

//写数据模块
ICP_Serial_Comm_Write_1(
    xExecute:= Write_xExecute,                      //上升沿触发功能块
    hCom:= ICP_Serial_Comm_hCom_1.hCom,             //通讯建立句柄
    write_szSize:= write_szSize_1,                  //写数据长度
    pWriteData:= ADR(bWriteData_1),                 //写数据指针
    udiTimeOut:= ,                                  //超时时间
    xBusy=> ,                                       //功能块正在执行
    xDone=> ,                                       //功能块完成信号
    xError=> ,                                      //功能块错误
    eError=> );                                     //功能块错误代码
```

```
//状态机
CASE iState 0 OF
0:
  IF ICP_Serial_Comm_hCom_1.hCom 0 <>0 AND Open TRUE THEN
     iState 0 :=1;
     ELSE
     iState 0 :=0;
  END_IF
1:
  Write_xExecute FALSE :=TRUE;
  IF ICP_Serial_Comm_Write_1.xDone FALSE THEN
     Write_xExecute FALSE :=FALSE;
     iState 0 :=2;
  END_IF
2:
  Read_xExecute FALSE :=TRUE;
  IF ICP_Serial_Comm_Read_1.xDone FALSE THEN
     Read_xExecute FALSE :=FALSE;
     iState 0 :=0;
  END_IF
END_CASE

IF Open TRUE =FALSE THEN
    iState 0 :=0;
END_IF
```

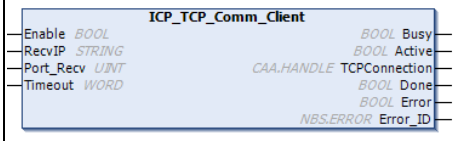
8.2 TCP 自由口指令

8.2.1 指令列表

指令类别	名称	功能
自由 TCP 通信指令	ICP_TCP_Comm_Client	创建 TCP 客户端通信服务
	ICP_TCP_Comm_Write	TCP 通信数据发送
	ICP_TCP_Comm_Read	TCP 通信数据接收
	ICP_TCP_Comm_Connect	创建 TCP 连接，并连接到服务器
	ICP_TCP_Comm_Server	创建 TCP 服务器端通信服务

8.2.2 ICP_TCP_Comm_Client

指令格式：

指令	名称	图形表现	ST 表现
ICP_TCP_Comm_Client	创建 TCP 客户端通信服务指令		ICP_TCP_Comm_Client(Enable:=, RecvIP:=, Port_Recv:=, Timeout:=, Busy=>t, Active=>, TCPConnection=>, Done=>, Error=>, Error_ID=>);

相关变量：

输入类型	名称	数据类型	有效范围	初始值	描述
Enable	使能	BOOL	TRUE,FALSE	FALSE	高电平使能功能块
RecvIP	接收 IP	STRING	-	-	接收控制器 IP
Port_Recv	接收端口	UINT	-	-	接收控制器端口
Timeout	超时时间	WORD	-	1000	请求连接的超时时间
输出类型	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Busy	功能块运行	BOOL	TRUE,FALSE	FALSE	运行信号
Active	成功建立连接标志位	BOOL	TRUE,FALSE	-	服务器与客户端建立通信成功标志
TCPConnection	连接句柄	CAA.HANDLE	-	-	服务器与客户端建立通信的连接句柄
Done	完成标志	BOOL	TRUE,FALSE	FALSE	完成信号
Error	错误标志	BOOL	TRUE,FALSE	FALSE	连接错误
Error_ID	错误	NBS.ERROR	-	0	错误码

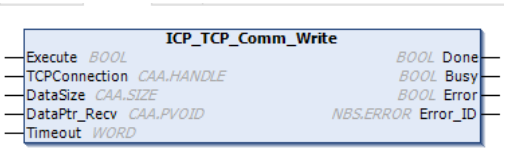
程序示例：

创建 TCP 客户端通信服务指令，当 ICP_TCP_Comm_Client 指令 Enable 为 TRUE 时，本地客户端监听远端服务器端的连接请求。当客户端与服务器连接成功后，将创建客户与远端服务器的通信有效句柄（TCPConnection 大于 0）。

```
//需注意，服务器接收端口号建议设为5000
ICP_TCP_Comm_Client_1(
    Enable FALSE := Enable_Client FALSE,                    //输入的高电平将启动功能块
    RecvIP "" := Recv_IP "",                                //接收控制器IP
    Port_Recv 0 := Port_Recv 0,                             //接收控制器端口号
    Timeout 10000 := Timeout_Client 10000,                  //请求连接的超时时间，单位 us
    Busy FALSE => Busy_Client FALSE,                        //功能块执行标志
    Active FALSE => Active_Client FALSE,                    //成功建立连接标志位
    TCPConnection=> ,                                       //客户端与服务器建立通信的连接句柄
    Done=> ,
    Error FALSE => Error_Client FALSE,                      //功能块错误标志
    Error_ID NO_ERROR => ErrorID_Client NO_ERROR );         //功能块错误码
```

8.2.3 ICP_TCP_Comm_Write

指令格式：

指令	名称	图形表现	ST 表现
ICP_TCP_Comm_Write	TCP 通信数据发送指令		ICP_TCP_Comm_Write(Execute:=, TCPConnection:=, DataSize:=, DataPtr_Recv:=, Timeout:=, Done=>, Busy=>, Error=>, Error_ID=>);

相关变量：

输入类型	名称	数据类型	有效范围	初始值	描述
Enable	使能	BOOL	TRUE,FALSE	FALSE	高电平使能功能块
TCPConnection	连接句柄	CAA.HANDLE	-	-	连接句柄
DataSize	数据长度	CAA.SIZE	-	-	数据长度，Byte
DataPtr_Recv	数据地址	CAA.PVOID	-	-	数据地址
Timeout	超时时间	WORD	-	1000	请求连接的超时时间
输出类型	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Done	完成标志	BOOL	TRUE,FALSE	-	完成信号
Busy	功能块运行	BOOL	TRUE,FALSE	FALSE	运行信号
Error	错误标志	BOOL	TRUE,FALSE	FALSE	发送错误
Error_ID	错误	NBS.ERROR	-	0	错误码

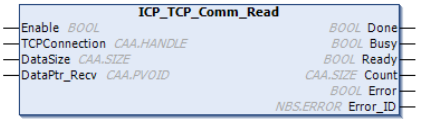
程序示例：

TCP 通信数据发送指令，当 ICP_TCP_Comm_Write 指令 Enable 为 TRUE 时，会将用户设置的发送缓冲区 DataPtr_Recv 为首地址的长度为 DataSize 的数据发送到 TCPConnection 句柄连接的目标设备。如果在超时时间 Timeout 内发送成功，则 Done 置 TRUE。

```
ICP_TCP_Comm_Write_1(
    Execute FALSE := Execute_Write FALSE,                   //输入的上升沿将触发功能块
    TCPConnection 0 := ICP_TCP_Comm_Client_1.TCPConnection 0 ,   //客户端与服务器建立通信的连接句柄
    DataSize 10 := Data_Length_Write 0,                     //设定写入的数据个数
    DataPtr_Recv 0 := ADR(Data_Write),                      //写入的数据地址
    Timeout 60000 := Timeout_Write 10000,                   //请求连接的超时时间，单位 us
    Done FALSE => Done_Write FALSE,                         //写入完成标志位
    Busy FALSE => Busy_Write FALSE,                         //功能块执行标志
    Error FALSE => Error_Write FALSE,                       //功能块错误标志
    Error_ID=> );                                           //功能块错误码
```

8.2.4 ICP_TCP_Comm_Read

指令格式：

指令	名称	图形表现	ST 表现
ICP_TCP_Comm_Read	TCP 通信数据接收指令		ICP_TCP_Comm_Read(Enable:=, TCPConnection:=, DataSize:=, DataPtr_Recv:=, Done=>, Busy=>, Ready=>, Count=>, Error=>, Error_ID=>);

相关变量：

输入类型	名称	数据类型	有效范围	初始值	描述
Enable	使能	BOOL	TRUE,FALSE	FALSE	高电平使能功能块
TCPConnection	连接句柄	CAA.HANDLE	-	-	连接句柄
DataSize	数据长度	CAA.SIZE	-	-	数据长度，Byte
DataPtr_Recv	数据地址	CAA.PVOID	-	-	数据地址
输出类型	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Done	完成标志	BOOL	TRUE,FALSE	FALSE	完成信号
Busy	功能块运行	BOOL	TRUE,FALSE	FALSE	运行信号
Ready	连接成功	BOOL	TRUE,FALSE	FALSE	从缓冲区内读取数据，如果数据不为空，置位一个扫描周期的标志位
Count	数据长度	CAA.SIZE	-	-	接收数据实际长度
Error	错误标志	BOOL	TRUE,FALSE	FALSE	接收错误
Error_ID	错误	NBS.ERROR	-	0	错误码

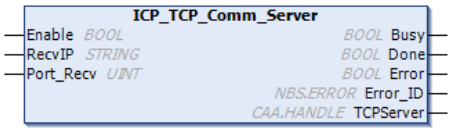
程序示例：

TCP 通信数据接收指令，当 ICP_TCP_Comm_Read 指令 Enable 为 TRUE 时，将从 TCP 通信缓冲区读取数据，Busy 为 TRUE。若读取成功，Done 置位一个扫描周期；读取的数据会被放置到地址为 DataPtr_Recv 的变量里；同时 Ready 置位一个扫描周期；接收数据区域的实际大小值将会赋给 Count，在一个扫描周期后 Count 的值会被清零。

```
14  ICP_TCP_Comm_Read_1(
15      Enable FALSE := Enable_Read FALSE,                          //输入的高电平将启动功能块
16      TCPConnection 0 := ICP_TCP_Comm_Client_1.TCPConnection 0,   //客户端与服务器建立通信的连接句柄
17      DataSize 0 := Data_Length_Read 0,                           //设定读取的数据个数
18      DataPtr_Recv 3030930148 := ADR(Data_Read),                  //读取的数据地址
19      Done=> ,                                                    //服务器端中止标志，检查功能块配置
20      Busy FALSE => Busy_Read FALSE,                              //功能块执行标志
21      Ready FALSE => Ready_Read FALSE,                            //成功建立连接标志位
22      Count=> ,                                                   //实际读取的数据个数
23      Error FALSE => Error_Read FALSE,                            //功能块错误标志
24      Error_ID NO_ERROR => ErrorID_Read NO_ERROR );               //功能块错误码
25
```

8.2.5 ICP_TCP_Comm_Server

指令格式：

指令	名称	图形表现	ST 表现
ICP_TCP_Comm_Server	创建 TCP 服务器端通信服务指令		ICP_TCP_Comm_Server(Enable:=, RecvIP:=, Port_Recv:=, Busy=>, Done=>, Error=>, Error_ID=>, TCPServer=>);

相关变量：

输入类型	名称	数据类型	有效范围	初始值	描述
Enable	使能	BOOL	TRUE,FALSE	FALSE	高电平使能功能块
RecvIP	控制器 IP	STRING	-	-	服务器控制器 IP
Port_Recv	控制器端口号	UINT	-	-	服务器控制器端口号
输出类型	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Busy	功能块运行	BOOL	TRUE,FALSE	FALSE	运行标志
Done	完成标志	BOOL	TRUE,FALSE	FALSE	完成信号
Error	错误标志	BOOL	TRUE,FALSE	FALSE	连接错误
Error_ID	错误	NBS.ERROR	-	-	错误码
TCPServer	服务器句柄	CAA.HANDLE	-	-	TCP 服务器句柄

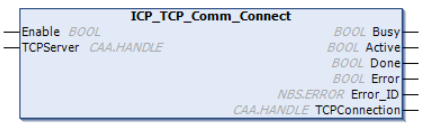
程序示例：

创建 TCP 服务器端通信服务指令。当 ICP_TCP_Comm_Server 指令 Enable 为 TRUE 时，将创建服务器与远端客户的的通信有效句柄（TCPServer 不等于 0）。

```
1   //需注意，服务器接收端口号建议设为5000
2
3   ICP_TCP_Comm_Server_1(
4       Enable:= Enable_Server,                          //输入的高电平将启动功能块
5       RecvIP:= Recv_IP,                                //接收控制器IP
6       Port_Recv:= Port_Recv,                           //接收控制器端口号
7       Busy=> ,                                         //功能块执行标志
8       Done=> ,                                         //中止标志，检查功能块配置
9       Error=> Error_Server,                            //功能块错误标志
10      Error_ID=> ErrorID_Server,                       //功能块错误码
11      TCPServer=> );                                   //TCP 服务器句柄
```

8.2.6 ICP_TCP_Comm_Connect

指令格式：

指令	名称	图形表现	ST 表现
ICP_TCP_Comm_Connect	创建 TCP 连接，并连接到服务器指令		ICP_TCP_Comm_Connect(Enable:=, TCPServer:=, Busy=>, Active=>, Done=>, Error=>, Error_ID=>, TCPConnection=>);

相关变量：

输入类型	名称	数据类型	有效范围	初始值	描述
Enable	使能	BOOL	TRUE,FALSE	FALSE	高电平使能功能块
TCPServer	连接句柄	CAA.HANDLE	-	-	TCP 连接句柄
输出类型	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Busy	功能块运行	BOOL	TRUE,FALSE	FALSE	运行标志
Active	连接成功	BOOL	TRUE,FALSE	FALSE	服务器端监听到远端客户端连接句柄标志
Done	完成标志	BOOL	TRUE,FALSE	FALSE	完成信号
Error	错误标志	BOOL	TRUE,FALSE	FALSE	连接错误
Error_ID	错误	NBS.ERROR	-	-	错误码
TCPConnection	连接句柄	CAA.HANDLE	-	-	服务器与客户端建立通信的连接句柄

程序示例：

创建 TCP 连接，并连接到服务器指令。当 ICP_TCP_Comm_Connect 指令 Enable 为 TRUE 时，本地服务器端监听远端客户端的连接请求。当客户端通过服务器句柄 TCPServer 与服务器连接成功后，将创建服务器与远端客户的的通信有效句柄（TCPConnection 大于 0）。

```
13  ICP_TCP_Comm_Connect_1(
14      Enable FALSE := Enable_Server FALSE,                    //输入的高电平将启动功能块
15      TCPServer  0  := ICP_TCP_Comm_Server_1.TCPServer  0 ,          //TCP 服务器句柄
16      Busy FALSE => Busy_Connect FALSE,                       //功能块执行标志
17      Active FALSE => Active_Connect FALSE,                   //服务器端监听到远端客户端连接句柄标志
18      Done=> ,                                          //中止标志，检查功能块配置
19      Error FALSE => Error_Connect FALSE,                     //功能块错误标志
20      Error_ID  NO_ERROR  => ErrorID_Connect  NO_ERROR ,             //功能块错误码
21      TCPConnection=> );                                //服务器与客户端建立通信的连接句柄
```

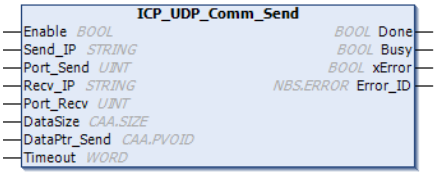
8.3 UDP 自由口指令

8.3.1 指令列表

指令类别	名称	功能
自由 UDP 通信指令	ICP_UDP_Comm_Send	UDP 通信数据发送
	ICP_UDP_Comm_Receive	UDP 通信数据接收

8.3.2 ICP_UDP_Comm_Send

指令格式：

指令	名称	图形表现	ST 表现
ICP_UDP_Comm_Send	UDP 数据发送指令		ICP_UDP_Comm_Send(Enable:=, Send_IP:=, Port_Send:=, Recv_IP:=, Port_Recv:=, DataSize:=, DataPtr_Send:=, Timeout:=, Done=>, Busy=>, xError=>);

相关变量：

输入类型	名称	数据类型	有效范围	初始值	描述
Enable	使能	BOOL	TRUE,FALSE	FALSE	高电平使能功能块
Send_IP	发送 IP	STRING	-	-	发送控制器 IP
Port_Send	发送端口	UINT	-	-	发送控制器端口
Recv_IP	接收 IP	STRING	-	-	接收控制器 IP
Port_Recv	接收端口	UINT	-	-	接收控制器端口
DataSize	数据长度	CAA.SIZE	-	-	需要发送的数据长度
DataPtr_Send	数据存储	CAA.PVOID	-	-	接收数据存储地址
Timeout	超时时间	WORD	-	1000	请求连接的超时时间
输出类型	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Done	完成标志	BOOL	TRUE,FALSE	FALSE	完成信号
Busy	功能块运行	BOOL	TRUE,FALSE	FALSE	运行标志
xError	错误标志	BOOL	TRUE,FALSE	FALSE	发送错误
Error_ID	错误	NBS.ERROR	-	0	错误码

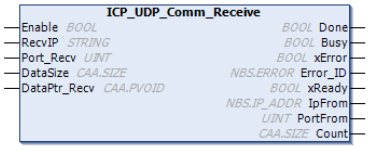
程序示例：

UDP 通讯数据发送指令，当 ICP_UDP_Comm_Send 指令 Enable 为 TRUE 时，会将用户设置的发送缓冲区 DataPtr_Send 为首地址的长度为 DataSize 的数据发送到接收 IP 为 Recv_IP 的目标设备。如果在超时时间 Timeout 内发送成功，则 Done 置 TRUE。

```
1
2   //发送UDP功能块
3   ICP_UDP_Comm_Send_1(
4       Enable TRUE := ON TRUE ,                          //高电平使能功能块
5       Send_IP '192.168.1. ▸ := '192.168.1.10',          //发送控制器IP
6       Port_Send 5000 := 5000,                           //发送控制器端口
7       Recv_IP '192.168.1. ▸ := '192.168.1.40',          //接收控制器IP
8       Port_Recv 3000 := 3000,                           //接收控制器端口
9       DataSize 50 := 50, //SIZEOF(Data)                 //需要发送的数据长度 BYTE
10      DataPtr_Send 139651632122400 := ADR(Data),        //接收数据存储地址
11      Timeout:=,                                        //请求连接的超时时间
12      Done=> ,                                          //完成信号
13      Busy=> ,                                          //运行标志
14      xError=> );                                       //发送错误
15  RETURN
```

8.3.3 ICP_UDP_Comm_Receive

指令格式：

指令	名称	图形表现	ST 表现
ICP_UDP_Comm_Receive	UDP 数据接收指令		ICP_UDP_Comm_Receive(Enable:=, RecvIP:=, Port_Recv:=, DataSize:=, DataPtr_Recv:=, Done=>, Busy=>, xError=>, Error_ID=>, xReady=>, IpFrom=>, PortFrom=>, Count=>);

相关变量：

输入类型	名称	数据类型	有效范围	初始值	描述
Enable	使能	BOOL	TRUE,FALSE	FALSE	高电平使能功能块
Recv_IP	接收 IP	STRING	-	-	接收控制器 IP
Port_Recv	接收端口	UINT	-	-	接收控制器端口
DataSize	数据长度	CAA.SIZE	-	-	需要发送的数据长度
DataPtr_Recv	数据存储	CAA.PVOID	-	-	接收数据存储地址
输出类型	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Done	完成标志	BOOL	TRUE,FALSE	FALSE	完成信号
Busy	功能块运行	BOOL	TRUE,FALSE	FALSE	运行标志
xError	错误标志	BOOL	TRUE,FALSE	FALSE	发送错误
Error_ID	错误	NBS.ERROR	-	0	错误码
xReady	成功标志	BOOL	TRUE,FALSE	FALSE	接收成功标志
IpFrom	发送 IP	NBS.IP_ADDR	-	-	数据发送控制器 IP
PortFrom	发送端口	UINT	-	-	数据发送控制器端口
Count	数据长度	CAA.SIZE	-	-	接收数据实际长度

程序示例：

UDP 通信数据接收指令。当 ICP_UDP_Comm_Receive 指令 Enable 为 TRUE 时，将从 UDP 通信缓冲区读取数据，Busy 为 TRUE。若读取成功，Done 置位一个扫描周期；读取的数据会被放置到地址为 DataPtr_Recv 的变量里；同时 xReady 置位一个扫描周期；接收数据区域的实际大小值将会赋给 Count，在一个扫描周期后 Count 的值会被清零。

```
1  //接收UDP功能块
2  ICP_UDP_Comm_Receive_1(
3      Enable TRUE := ON TRUE ,                    //高电平使能功能块
4      RecvIP '192.168.1. ▸ := '192.168.1.10',     //接收控制器IP
5      Port_Recv 3000 := 3000,                     //接收控制器端口
6      DataSize 100 := SIZEOF(Data),               //需要接收的数据长度
7      DataPtr_Recv 3030897112 := ADR(Data),       //接收数据存储地址
8      Done=> ,                                    //接收完成信号
9      Busy=> ,                                    //运行标志
10     xError=> ,                                  //端口或IP配置错误
11     Error_ID=> ,                                //错误码
12     xReady=> ,                                  //接收成功标志
13     IpFrom=> ,                                  //数据发送控制器IP
14     PortFrom=> ,                                //数据发送控制器端口
15     Count=> );                                  //接收数据实际长度
16     RETURN
```

9 脉冲输出指令

9.1 辅助指令

注意：此章节内容仅适用 TM 系列 PLC。

9.1.1 IMC_GetSys_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_GetSys_P	获取系统信息功能块		IMC_GetSys_P(Enable:= , ValueMode:= , ReturnValue=> Version=> , Busy=> , Error=> , ErrorID=>);

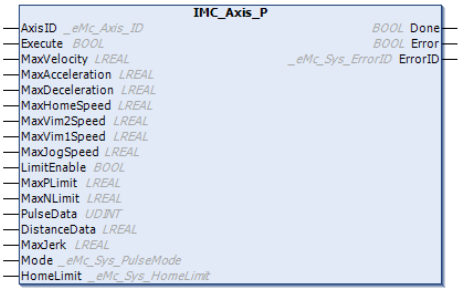
相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
Enable	功能块使能位	BOOL	TRUE,FALSE	TRUE	高电平有效，低电平无效
ValueMode	值的组数类型	_eMc_Sys_ValueMode	_mcSysAxisNum，_mcSysGroupNum，_mcSysCAMNum，_mcSysGearNum，_mcSysFlyCutNum，_mcSysTraceNum	_mcSysAxisNum	值的组数类型，系统支持的轴数：_mcSysAxisNum；系统支持的轴组数：_mcSysGroupNum；系统支持的凸轮表数：_mcSysCAMNum；系统支持的电子齿轮组数：_mcSysGearNum；系统支持的飞剪组数：_mcSysFlyCutNum；系统支持的追剪组数：_mcSysTraceNum
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
ReturnValue	返回值	UINT	-	0	返回值，如：4（轴）、2（轴组）、3（凸轮/齿轮/飞剪/追剪组数）
Version	版本号	STRING	-	V0.0.0.1	返回版本号
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE-功能块执行中 FALSE-功能块未执行
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE-出错，FALSE-未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcError_NULL	错误码

功能说明：获取系统信息的功能块。

9.1.2 IMC_Axis_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_Axis_P	轴参数配置功能块		IMC_Axis_P(AxisID:= , Execute:= , MaxVelocity:= , MaxAcceleration:= , MaxDeceleration:= , MaxHomeSpeed:= , MaxVim2Speed:= , MaxVim1Speed:= , MaxJogSpeed:= , LimitEnable:= , MaxPLimit:= , MaxNLimit:= , PulseData:= , DistanceData:= , MaxJerk:= , Mode:= , HomeLimit:= , Done=> , Error=> , ErrorID=>);

相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴号	_eMc_Axis_ID	0~3	255	轴号(注：初始值 255 是保护措施，避免未赋初值的情况，下同)
Execute	功能块使能位	BOOL	TRUE,FALSE	FALSE	上升沿有效， 下降沿无效
MaxVelocity	轴最大运行速度	LREAL	>1	5000	当前轴最大运行速度(unit/s)，由用户设定
MaxAcceleration	轴运行最大加速度	LREAL	>0	500	当前轴最大允许加速度(unit/s²)
MaxDeceleration	轴运行最大减速度	LREAL	>0	500	当前轴最大允许减速度(unit/s²)
MaxHomeSpeed	轴运行最大回零速度	LREAL	1~80	10	当前轴最大回零速度(unit/s)
MinVim2Speed	第二阶段最大回零速度	LREAL	1~2	1	当前轴第二阶段最大回零速度(unit/s)
MaxVim1Speed	第一阶段最大回零速度	LREAL	1~10	3	当前轴第一阶段最大回零速度(unit/s)
MaxJogSpeed	轴运行最大手动速度	LREAL	1~5000	5000	当前轴最大手动速度(unit/s)
LimitEnable	软限位使能标志	BOOL	TRUE,FALSE	TRUE	TRUE->软限位开启； FALSE->软限位关闭（单轴速度模式不考虑)
MaxPLimit	最大正限位位置	LREAL	>0	9999999 99.999	当前轴最大正限位位置(unit)，单轴速度模式不考虑

输入变量	名称	数据类型	有效范围	初始值	描述
MaxNLimit	最大负限位位置	LREAL	<0	-999999999.999	当前轴最大负限位位置(unit)，单轴速度模式不考虑
PulseData	当前轴旋转一圈所需脉冲数	UDINT	>0	10000	当前轴运行一转需要脉冲数(pulse)
DistanceData	当前轴运行一转运行距离	LREAL	>0	10	当前轴运行一转运行距离（unit）
MaxJerk	当前轴最大加加速度	LREAL	10~400	100	当前轴最大加加速度(unit/s^3)
Mode	当前轴脉冲控制方式	_eMc_Sys_PulseMode	_mcPulseSign, _mcCWCCW, _mcQEP	_mcPulseSign	当前轴脉冲控制方式：_mcPulseSign：脉冲+符号方式；_mcCWCCW：正/反转脉冲列方式；_mcQEP ：正交编码脉冲方式
HomeLimit	回零过程中检测到硬限位的停止方式	_eMc_Sys_HomeLimit	_mcLimitDecStop,_mcLimitImmediatelyStop	_mcLimitDecStop	_mcLimitDecStop：减速停 _mcLimitImmediatelyStop：立即停
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Done	功能块完成状态	BOOL	TRUE,FALSE	FALSE	TRUE-轴定义完成 FALSE-轴定义未完成
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE-出错，FALSE-未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcErr, _NULL	错误码

功能说明：设定指令轴，设置指令轴的相关参数，脉冲型仅支持 4 个轴。

9.1.3 IMC_Power_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_Power_P	轴使能功能块		IMC_Power_P(AxisID:= , Enable:= , AxisError:= , AxisEnable:= , Status=> , Valid=> , Busy=> , Error=> , ErrorID=>);

相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴号	_eMc_Axis_ID	0-3	255	轴号
Enable	功能块使能位	BOOL	TRUE,FALSE	TRUE	高电平有效，低电平无效
AxisError	轴报警信号	BOOL	TRUE,FALSE	FALSE	TRUE：轴报警 FALSE：轴未报警
AxisEnable	轴使能完成信号	BOOL	TRUE,FALSE	TRUE	TRUE：轴使能完成 FALSE：轴未使能

输出变量	名称	数据类型	有效范围	初始值	描述
Status	轴状态	BOOL	TRUE,FALSE	FALSE	TRUE：轴已准备好 FALSE：轴未准备好
Valid	轴使能状态	BOOL	TRUE,FALSE	FALSE	TRUE：轴使能有效 TRUE：轴使能无效
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行中 FALSE：功能块未执行
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE：出错 FALSE：未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcError_NULL	错误码

功能说明：Enable 高电平触发功能块执行，AxisEnable 高电平触发轴使能。

注意：本功能块只能使内部轴使能。

9.1.4 IMC_SetPosition_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_SetPosition_P	设置轴当前指令位置功能块		IMC_SetPosition_P(AxisID:= , Execute:= , Position:= , Relative:= , Done=> , Busy=> , Error=> , ErrorID=>);

相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴号	_eMc_Axis_ID	0-3	255	轴号
Execute	功能块使能位	BOOL	TRUE,FALSE	FALSE	上升沿触发，下降沿无效
Position	位置	LREAL	实数	0	当前位置设定值
Relative	位置模式	_eMc_Position_Mode	_mcAbsolute，_mcRelative	_mcAbsolute	_mcAbsolute-绝对模式 _mcRelative-相对模式
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Done	功能块完成状态	BOOL	TRUE,FALSE	FALSE	TRUE-位置设置完成， FALSE-位置设置未完成
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE-功能块执行中 FALSE-功能块未执行
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE-出错，FALSE-未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcError_NULL	错误码

功能说明：Execute 上升沿触发，Relative 设置为_mcAbsolute 表示绝对位置模式，Relative 设置为_mcRelative 表示相对位置模式。

9.1.5 IMC_ReadCmdPosition_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_ReadCmdPosition_P	读取轴实际位置功能块		IMC_ReadCmdPosition_P(AxisID:= , Enable:= , Busy=> , Value=> , AxisDirection=> , Error=> , ErrorID=>);

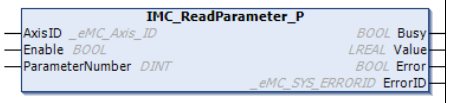
相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴号	_eMc_Axis_ID	0~3	255	轴号
Enable	功能块使能位	BOOL	TRUE,FALSE	TRUE	高电平有效，低电平无效
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行中 FALSE：功能块未执行
Value	位置值	LREAL	实数	0	读取指定轴的实际位置值
AxisDirection	轴方向	_eMc_Sys_Direction	_mcNegative, _mcForward	_mcNegative	_mcNegative：负向 _mcForward：正向
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE：出错 FALSE：未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcError_NULL	错误码

功能说明：Enable 上升沿触发功能块读取指定轴的实际位置，由 Value 返回轴的实际位置值。

9.1.6 IMC_ReadParameter_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_ReadParameter_P	读取轴定义参数功能块		IMC_ReadParameter_P(AxisID:= , Enable:= , ParameterNumber:= , Busy=> , Value=> , Error=> , ErrorID=>);

相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴号	_eMc_Axis_ID	0~3	255	轴号
Enable	功能块使能位	BOOL	TRUE,FALSE	TRUE	高电平有效 低电平无效
ParameterNumber	参数号	DINT	1000-1012	1000	轴定义参数序号 如：1000 为运行最大速度
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行中 FALSE：功能块未执行
Value	返回值	LREAL	1000-1012	0	轴定义参数输出
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE：出错， FALSE：未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcErr, _NULL	错误码

功能说明：Enable高电平有效，根据如下列出的参数序列号读取相应的值。

参数序列号如下：

序列号	名称	描述
1000	MaxVelocity	当前轴最大运行速度（unit/s）
1001	MaxAcceleration	当前轴最大允许加速度（unit/s²）
1002	MaxHomeSpeed	当前轴最大回零速度（unit/s）
1003	MinVim2Speed	当前轴第二阶段最大回零速度（unit/s）
1004	MaxVim1Speed	当前轴第二阶段最小回零速度（unit/s）
1005	MaxJogSpeed	当前轴最大手动速度（unit/s）
1006	LimitEnable	软限位使能标志 TRUE：软限位开启；FALSE：软限位关闭
1007	MaxPLimit	当前轴最大正限位位置（unit）
1008	MaxNLimit	当前轴最大负限位位置（unit）
1009	PulseData	当前轴运行一转需要脉冲数（pulse）
1010	DistanceData	当前轴运行一转运行距离（unit）
1011	MaxJerk	当前轴最大允许加加速度
1012	Mode	当前轴脉冲控制方式： _mcPulseSign：脉冲+符号方式 _mcCWCCW：正/反转脉冲列方式 _mcQEP：正交编码脉冲方式

9.1.7 IMC_ReadStatus_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_ReadStatus_P	读取轴的运行状态功能块		IMC_ReadStatus_P(AxisID:= , Enable:= , Valid=> , Busy=> , Status=> , Error=> , ErrorID=>);

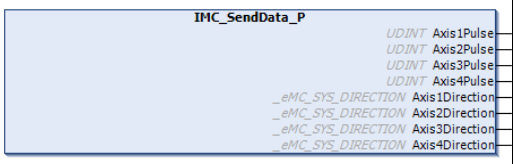
相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴号	_eMc_Axis_ID	0~3	255	轴号
Enable	功能块使能位	BOOL	TRUE,FALSE	TRUE	高电平有效，低电平无效
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Valid	有效状态	BOOL	TRUE,FALSE	FALSE	TRUE-有效输出， FALSE-无效输出
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE-功能块执行中 FALSE-功能块未执行
Status	轴状态	_eMc_Axis_Status	-	_mcDisabled	返回轴所处状态
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE-出错，FALSE-未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcError_NULL	错误码

功能说明：高电平触发功能块返回轴当前状态。

9.1.8 IMC_SendData_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_SendData_P	发送数据功能块		IMC_SendData_P(Axis1Pulse=> , Axis2Pulse=> , Axis3Pulse=> , Axis4Pulse=> , Axis1Direction=> , Axis2Direction=> , Axis3Direction=> , Axis4Direction=>);

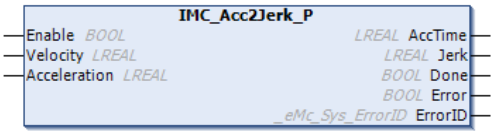
相关变量：

输出变量	名称	数据类型	有效范围	初始值	描述
Axis1Pulse	轴 1 脉冲反馈	UINT	正数	0	轴 1 脉冲反馈
Axis2Pulse	轴 2 脉冲反馈	UINT	正数	0	轴 2 脉冲反馈
Axis3Pulse	轴 3 脉冲反馈	UINT	正数	0	轴 3 脉冲反馈
Axis4Pulse	轴 4 脉冲反馈	UINT	正数	0	轴 4 脉冲反馈
Axis1Direction	轴 1 方向	_eMc_Sys_Direction	_mcNegative, mcForward	_mcNegative	_mcNegative：负向 _mcForward：正向
Axis2Direction	轴 2 方向	_eMc_Sys_Direction	_mcNegative, mcForward	_mcNegative	_mcNegative：负向 _mcForward：正向
Axis3Direction	轴 3 方向	_eMc_Sys_Direction	_mcNegative, mcForward	_mcNegative	_mcNegative：负向 _mcForward：正向
Axis4Direction	轴 4 方向	_eMc_Sys_Direction	_mcNegative,_ mcForward	_mcNegative	_mcNegative：负向 _mcForward：正向

功能说明：将数据发送到底层，脉冲型调用完所有的运动控制功能块后必须添加的模块。

9.1.9 IMC_Acc2Jerk_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_Acc2Jerk_P	通过轴加速度计算加加速度功能块		IMC_Acc2Jerk_P(Enable:= , Velocity:= , Acceleration:= , AccTime=> , Jerk=> , Done=> , Error=> , ErrorID=>);

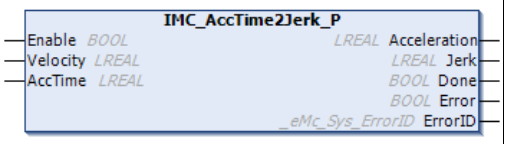
相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
Enable	功能块使能位	BOOL	TRUE,FALSE	TRUE	高电平有效，低电平无效
Velocity	速度	LREAL	正数	100	目标速度
Acceleration	加速度	LREAL	正数	10	目标加速度，unit/s^2
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
AccTime	加速时间	LREAL	正数	0	加速时间，s
Jerk	加加速度	LREAL	正数	0	加加速度，unit/s^3
Done	功能块完成状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行完成 FALSE：功能块未执行
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE：出错 FALSE：未出错
ErrorID	错误码	_eMc_Sys _ErrorID	-	_mcError_NULL	错误码

功能说明：根据速度，加速度的值计算加速时间 AccTime 与加加速度 Jerk 的值。

9.1.10 IMC_AccTime2Jerk_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_AccTime2Jerk_P	通过加速度时间计算加加速度功能块		IMC_AccTime2Jerk_P(Enable:= , Velocity:= , AccTime:= , Acceleration=> , Jerk=> , Done=> , Error=> , ErrorID=>);

相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
Enable	功能块使能位	BOOL	TRUE,FALSE	TRUE	高电平有效，低电平无效
Velocity	速度	LREAL	正数	100	目标速度，unit/s
AccTime	加速时间	LREAL	正数	1	目标加速时间，s
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Acceleration	加速度	LREAL	正数	0	目标加速度，unit/s^2
Jerk	加加速度	LREAL	正数	0	加加速度，unit/s^3
Done	功能块完成状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行完成 FALSE：功能块未执行
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE：出错 FALSE：未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcError_NULL	错误码

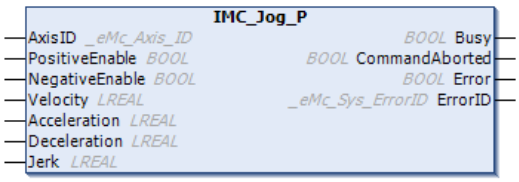
功能说明：根据速度，加速时间的值计算加速度 Accleration 与加加速度 Jerk 的值。

9.2 单轴指令

注意：此章节内容仅适用 TM 系列 PLC。

9.2.1 IMC_Jog_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_Jog_P	手动连续功能块		IMC_Jog_P(AxisID:= , PositiveEnable:= , NegativeEnable:= , Velocity:= , Acceleration:= , Deceleration:= , Jerk:= , Busy=> , CommandAborted=> , Error=> , ErrorID=>);

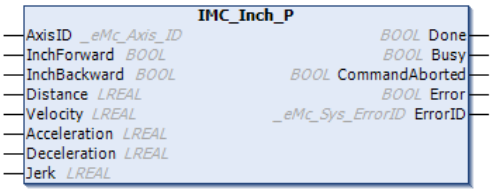
相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴号	_eMc_Axis_ID	0~3	255	轴号
PositiveEnable	正向运行	BOOL	TRUE,FALSE	FALSE	上升沿起动，下降沿停止
NegativeEnable	负向运行	BOOL	TRUE,FALSE	FALSE	上升沿起动，下降沿停止
Velocity	速度	LREAL	正数	10	轴运行速度(unit/s)
Acceleration	加速度	LREAL	正数	10	轴运行加速度(unit/s^2)
Deceleration	减速度	LREAL	正数	10	轴运行减速度(unit/s^2)
Jerk	加加速度	LREAL	正数或 0	0	加加速度(unit/s^3)
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行中 FALSE：功能块未执行
CommandAborted	功能块中断状态	BOOL	TRUE,FALSE	FALSE	TRUE：被中断 FALSE：未被中断
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE：出错 FALSE：未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcError_NULL	错误码

功能说明：轴起始静止状态，PositiveEnable 参数控制轴正方向起停，NegativeEnable 参数控制轴负方向起停。

9.2.2 IMC_Inch_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_Inch_P	手动点动功能块		IMC_Inch_P(AxisID:= , InchForward:= , InchBackward:= , Distance:= , Velocity:= , Acceleration:= , Deceleration:= , Jerk:= , Done=> , Busy=> , CommandAborted=> , Error=> , ErrorID=>);

相关变量：

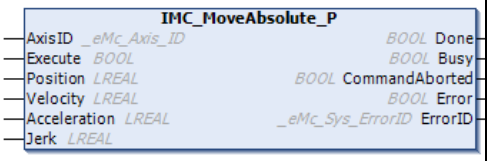
输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴号	_eMc_Axis_ID	0~3	255	轴号
InchForward	正向运行	BOOL	TRUE,FALSE	FALSE	上升沿起动，下降沿停止
InchBackward	负向运行	BOOL	TRUE,FALSE	FALSE	上升沿起动，下降沿停止
Distance	距离	LREAL	实数	0.5	轴运行距离(unit)
Velocity	速度	LREAL	正数	10	轴运行速度(unit/s)
Acceleration	加速度	LREAL	正数	10	轴运行加速度(unit/s^2)
Deceleration	减速度	LREAL	正数	10	轴运行减速度(unit/s^2)
Jerk	加加速度	LREAL	正数或 0	0	加加速度(unit/s^3)

输出变量	名称	数据类型	有效范围	初始值	描述
Done	功能块完成状态	BOOL	TRUE,FALSE	FALSE	TRUE：点动完成， FALSE：点动未完成
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行中 FALSE：功能块未执行
CommandAborted	功能块中断状态	BOOL	TRUE,FALSE	FALSE	TRUE：被中断 FALSE：未被中断
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE：出错 FALSE：未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcError_NULL	错误码

功能说明：InchForward 参数控制正方向的点位运动，InchBackward 参数控制负方向的点位运动，支持 S 型后加减速功能。

9.2.3 IMC_MoveAbsolute_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_MoveAbsolute_P	单轴绝对运动功能块		IMC_MoveAbsolute_P(AxisID:= , Execute:= , Position:= , Velocity:= , Acceleration:= , Jerk:= , Done=> , Busy=> , CommandAborted=> , Error=> , ErrorID=>);

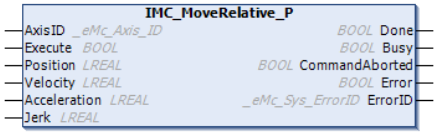
相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴号	_eMc_Axis_ID	0~3	255	轴号
Execute	功能块的使能位	BOOL	TRUE,FALSE	FALSE	上升沿有效， 下降沿无效
Position	绝对位置	LREAL	实数	0	轴绝对运行距离(unit)
Velocity	运行速度	LREAL	正数	10	速度(unit/s)
Acceleration	加速度	LREAL	正数	10	轴运行加速度(unit/s^2)
Jerk	加加速度	LREAL	正数或 0	0	加加速度(unit/s^3)
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Done	功能块完成状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行完成 FALSE：功能块未执行完成
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行中 FALSE：功能块未执行
CommandAborted	功能块中断状态	BOOL	TRUE,FALSE	FALSE	TRUE：被中断 FALSE：未被中断
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE：出错 FALSE：未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcError_NULL	错误码

功能说明：Execute 上升沿触发功能块执行，本功能块位置指令按绝对位置模式运行，支持梯形、S 曲线加减速方式。

9.2.4 IMC_MoveRelative_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_MoveRelative_P	单轴相对运动功能块		IMC_MoveRelative_P(AxisID:= , Execute:= , Position:= , Velocity:= , Acceleration:= , Jerk:= , Done=> , Busy=> , CommandAborted=> , Error=> , ErrorID=>);

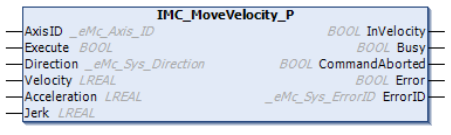
相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴号	_eMc_Axis_ID	0~3	255	轴号
Execute	功能块的使能位	BOOL	TRUE,FALSE	FALSE	上升沿有效， 下降沿无效
Position	相对位置	LREAL	实数	0	轴相对运行距离(unit)
Velocity	运行速度	LREAL	正数	10	速度(unit/s)
Acceleration	加速度	LREAL	正数	10	轴运行加速度(unit/s^2)
Jerk	加加速度	LREAL	正数或 0	0	加加速度(unit/s^3)
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Done	功能块完成状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行完成 FALSE：功能块未执行完成
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行中 FALSE：功能块未执行
CommandAborted	功能块中断状态	BOOL	TRUE,FALSE	FALSE	TRUE：被中断 FALSE：未被中断
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE：出错 FALSE：未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcError_NULL	错误码

功能说明：Execute 上升沿触发功能块执行，功能块位置指令按相对位置模式运行，支持梯形、S 曲线加减速方式。

9.2.5 IMC_MoveVelocity_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_MoveVelocity_P	速度模式功能块		IMC_MoveVelocity_P(AxisID:= , Execute:= , Direction:= , Velocity:= , Acceleration:= , Jerk:= , InVelocity=> , Busy=> , CommandAborted=> , Error=> , ErrorID=>);

相关变量：

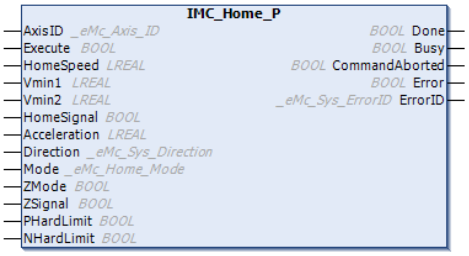
输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴号	_eMC_Axis_ID	0~3	255	轴号
Execute	功能块的使能位	BOOL	TRUE,FALSE	FALSE	上升沿有效， 下降沿无效
Direction	轴运动方向	_eMc_Sys _Direction	_mcNegative， _mcForward	_mcNegative	_mcNegative：负向运行 _mcForward：正向运行
Velocity	运行速度	LREAL	正数	10	速度(unit/s)
Acceleration	加速度	LREAL	正数	10	轴运行加速度(unit/s^2)
Jerk	加加速度	LREAL	正数或 0	0	加加速度(unit/s^3)
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
InVelocity	功能块完成状态	BOOL	TRUE,FALSE	FALSE	命令速度值第一次达到
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行中 FALSE：功能块未执行
Command Aborted	功能块中断状态	BOOL	TRUE,FALSE	FALSE	TRUE：被中断 FALSE：未被中断
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE：出错 FALSE：未出错
ErrorID	错误码	_eMc_Sys _ErrorID	-	_mcError _NULL	错误码

功能说明：

Execute 上升沿触发功能块进行轴匀速控制模式。

9.2.6 IMC_Home_P

指令格式：

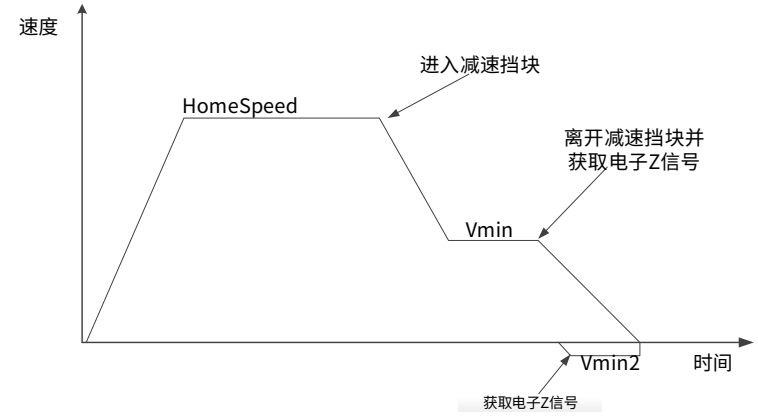
指令	名称	图形表现	ST 表现
IMC_Home_P	回零功能块		IMC_Home_P(AxisID:= , Execute:= , HomeSpeed:= , Vmin1:= , Vmin2:= , HomeSignal:= , Acceleration:= , Direction:= , Mode:= , ZMode:= , ZSignal:= , PHardLimit:= , NHardLimit:= , Done=> , Busy=> , CommandAborted=> , Error=> , ErrorID=>);

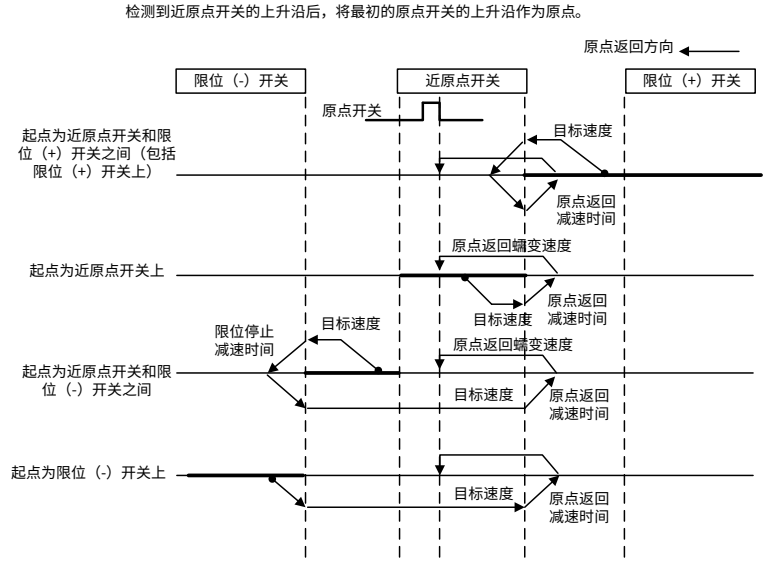
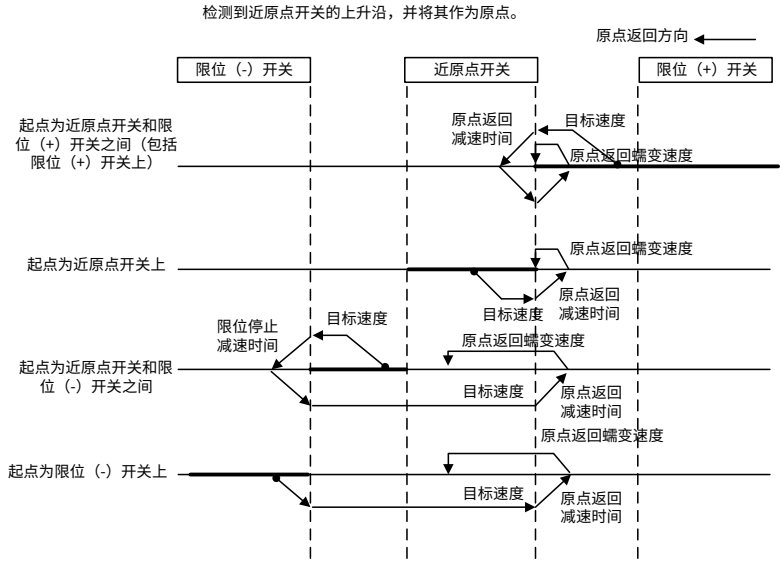
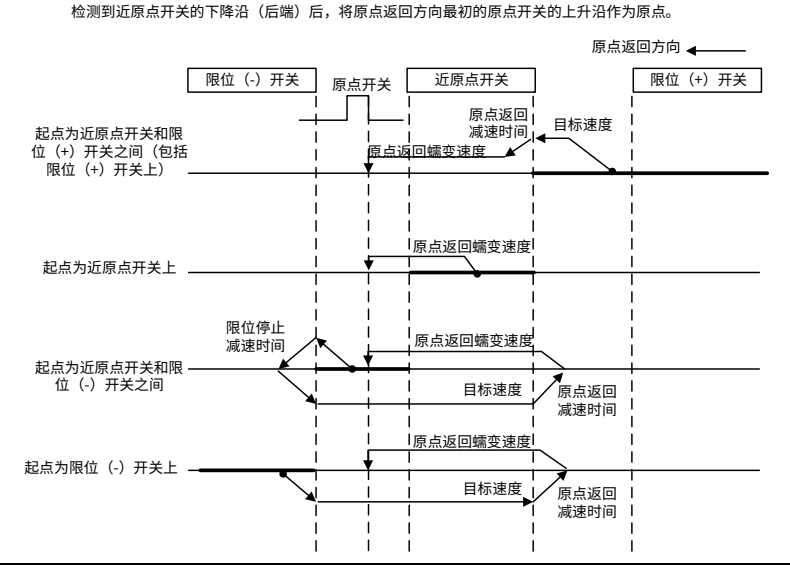
相关变量：

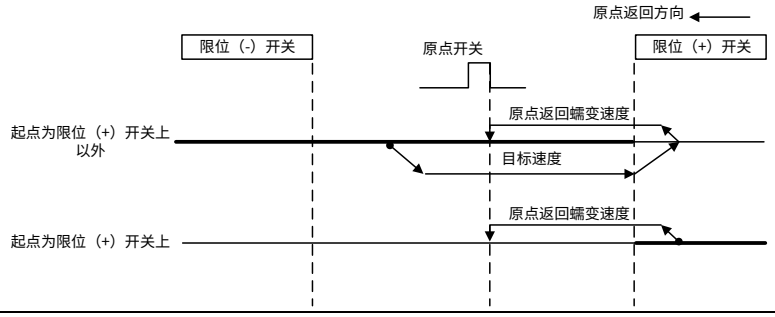
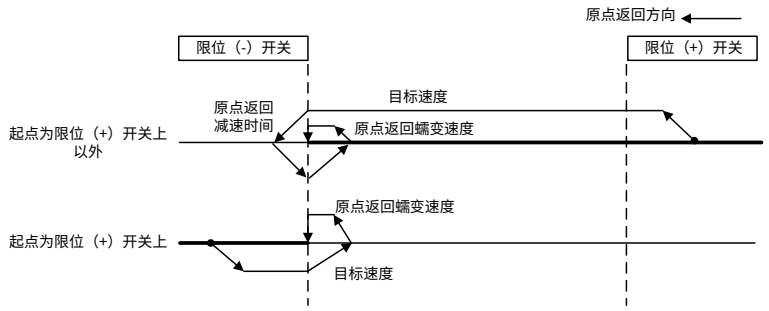
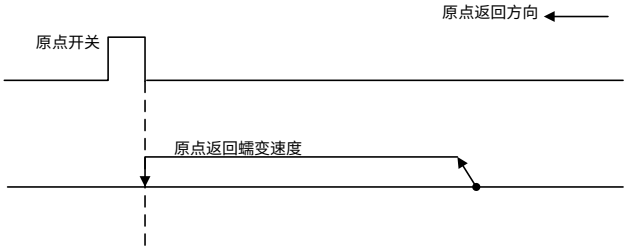
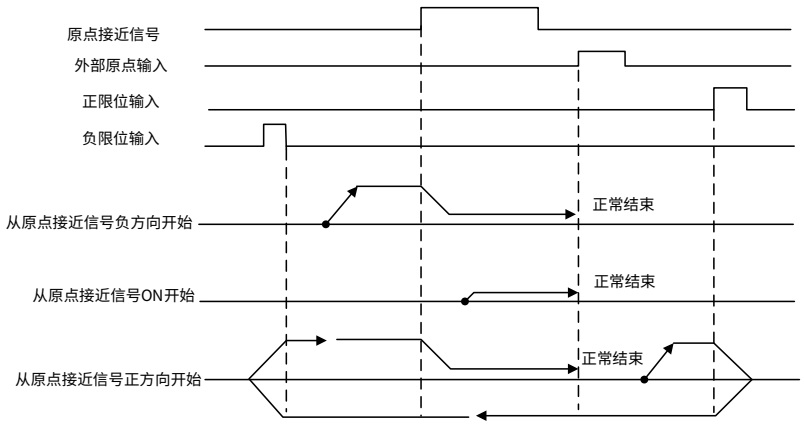
输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴号	_eMc_Axis_ID	0~3	255	轴号
Execute	功能块的使能位	BOOL	TRUE,FALSE	FALSE	上升沿触发，下降沿无效
HomeSpeed	回零速度	LREAL	正数	15	回零速度(unit/s)
Vmin1	在减速挡块的运行速度	LREAL	正数	3	在减速挡块的运行速度，(unit/s)
Vmin2	等待电机 Z 信号运行速度	LREAL	正数	1	等待电机 Z 信号运行速度，(unit/s)

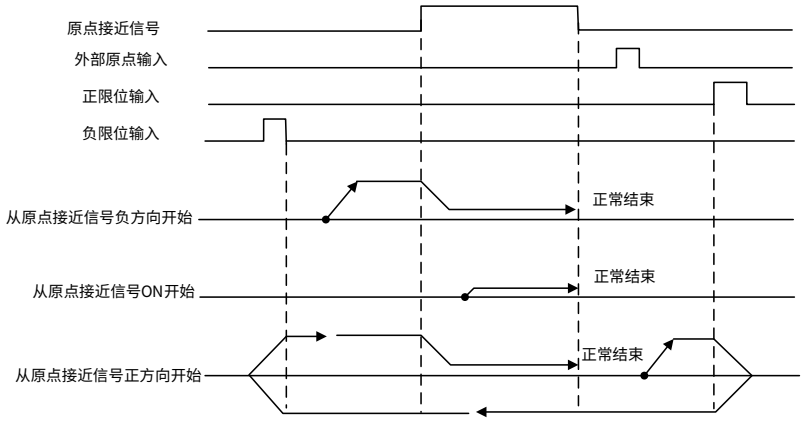
输入变量	名称	数据类型	有效范围	初始值	描述
HomeSignal	回零信号	BOOL	TRUE,FALSE	FALSE	回零信号
Acceleration	加速度	LREAL	正数	10	加速度(unit/s²)
Direction	回零方向	_eMc_Sys_Direction	_mcNegative, _mcForward	_mcForward	_mcNegative：负向运行 _mcForward：正向运行
Mode	回零模式	_eMc_Home_Mode	1~9	2	回零模式
ZMode	Z 信号获取方式	BOOL	TRUE,FALSE	FALSE	FALSE：Z 信号内部（电机）传入 TRUE：Z 信号外部传入
ZSignal	外部 Z 信号	BOOL	TRUE,FALSE	FALSE	ZMode 为 TRUE 时有效，ZSignal 输入上升沿 Z 信号有效
PHardLimit	正硬限位信号	BOOL	TRUE,FALSE	FALSE	正硬限位信号
NHardLimit	负硬限位信号	BOOL	TRUE,FALSE	FALSE	负硬限位信号
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Done	功能块完成状态	BOOL	TRUE,FALSE	FALSE	TRUE：回零完成 FALSE：回零未完成
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行中 FALSE：功能块未执行
Commandborted	功能块中断状态	BOOL	TRUE,FALSE	FALSE	TRUE-被中断 FALSE：未被中断
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE：出错 FALSE：未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcError_NULL	错误码

功能说明：控制内部当前轴位置回到零点位置，共 9 种回零模式。

模式	时序图	简略描述
模式 1	近原点开关的脉冲沿检测后端基准＋原点开关(Z 信号) 	如：速度加速至 HomeSpeed→碰减速板块信号→速度降为 Vmin→离开减速板块且获取电机 Z 信号→减速降至 0 反向加速至 Vmin2→获取电机 Z 信号→回零完成

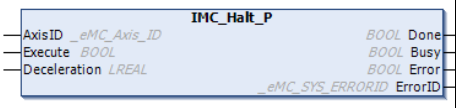
模式	时序图	简略描述
模式 2	近原点开关的脉冲沿检测前端基准＋原点开关(Z 信号) 	情况 1：目标速度(HomeSpeed)碰挡块（近原点开关）→0 反向→目标速度(HomeSpeed)离挡块→0 反向→Vim2 碰挡块→碰原点开关（电机 Z 信号）→回零完成
模式 3	近原点开关的脉冲沿检测前端基准 	情况 1：目标速度(HomeSpeed)碰挡块（近原点开关）→0 反向→目标速度(HomeSpeed)离挡块→0 反向→Vim2 碰挡块→回零完成
模式 4	近原点开关的脉冲沿检测后端基准＋原点开关(Z 信号) 	情况 1：目标速度(HomeSpeed)碰挡块（近原点开关）→Vim2 离挡块→碰原点开关（电机 Z 信号）→回零完成

模式	时序图	简略描述
模式 5	限位开关的脉冲沿检测＋原点开关(Z 信号) 检测到近原点开关的下降沿（后端）后，将原点返回方向最初的原点开关的上升沿作为原点。 	情况 1：目标速度(HomeSpeed)反向碰负限→0 反向→Vim2 正向→碰原点开关（电机 Z 信号）→回零完成
模式 6	限位开关的脉冲沿检测 检测到近原点开关的下降沿（后端）后，将原点返回方向最初的原点开关的上升沿作为原点。 	情况 1：目标速度(HomeSpeed)→碰负限位→0 反向→目标速度(HomeSpeed)离负限→0 负向→Vim2 碰正限位→回零完成
模式 7	原点开关的脉冲沿检测 从当前值向原点返回方向移动，检测到最初的原点开关的上升沿后停止，将其作为原点。 	如：正、负向回零-20→碰原点开关（电机 Z 信号）→回零完成
模式 8	近原点开关的脉冲沿检测后端基准＋原点开关(Z 信号) 	情况 1：正、负向回零→原点信号（减速挡块信号）→Vim2 离开原点信号（减速挡块信号）→碰外部原点信号（电机 Z 信号）→回零完成

模式	时序图	简略描述
模式 9	近原点开关的脉冲沿检测后端基准 	情况 1：正、负向回零→原点信号（减速挡块信号）→Vim2 离开原点信号（减速挡块信号）→回零完成

9.2.7 IMC_Halt_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_Halt_P	单轴停止功能块		IMC_Halt_P(AxisID:= , Execute:= , Deceleration:= , Done=> , Busy=> , Error=> , ErrorID=>);

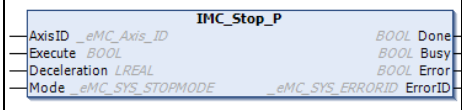
相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴 ID	_eMC_Axis_ID	0~3	255	轴号
Execute	功能块的使能位	BOOL	TRUE,FALSE	FALSE	上升沿有效， 下降沿无效
Deceleration	减速度	LREAL	正数	10	减速度(unit/s²)
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Done	功能块完成状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行完成， FALSE：功能块未执行完成
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行中 FALSE：功能块未执行
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE：出错，FALSE：未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcError_NULL	错误码

功能说明：Execute 上升沿触发，单轴停止后无需复位，再次触发单轴运动类功能块，单轴运动类功能块正常运行。

9.2.8 IMC_Stop_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_Stop_P	单轴停止功能块		IMC_Stop_P(AxisID:= , Execute:= , Deceleration:= , Mode:= , Done=> , Busy=> , Error=> , ErrorID=>);

相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴 ID	_eMc_Axis_ID	0~3	255	轴号
Execute	功能块的使能位	BOOL	TRUE,FALSE	FALSE	上升沿有效， 下降沿无效
Deceleration	减速度	LREAL	正数	10	减速度(unit/s²)
Mode	停止模式	_eMc_Sys_StopMode	_mcDecStop, _mcImmediatelyStop	_mcDecStop	_mcDecStop：减速停 _mcImmediatelyStop：立即停
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Done	功能块完成状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行完成， FALSE：功能块未执行完成
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行中 FALSE：功能块未执行
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE：出错 FALSE：未出错
ErrorID	错误码	_eMc_SysErrorID	-	_mcErrorNULL	错误码

功能说明：Execute 上升沿触发，模式设置为_mcDecStop，轴停止后，轴状态切为 Stopping，需将 Execute 置为 FALSE 即可解除轴 Stopping 状态；模式设置为_mcImmediatelyStop，轴停止后，轴状态切为 Stoppend 状态，轴报错，此时需借助 IMC_Reset 功能块复位解除。

9.2.9 IMC_Reset_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_Reset_P	单轴复位功能块		IMC_Reset_P(AxisID:= , Execute:= , Done=> , Busy=> , Error=> , ErrorID=>);

相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴 ID	_eMc_Axis_ID	0~3	255	轴号
Execute	功能块的使能位	BOOL	TRUE,FALSE	FALSE	上升沿有效，下降沿无效
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Done	功能块完成状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行完成 FALSE：功能块未执行完成
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行中 FALSE：功能块未执行
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE：出错， FALSE：未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcError NULL	错误码

功能说明：Execute 上升沿触发轴进行复位，一般在正、负限位或者当轴发生急停操作后，需要用到单轴复位功能块。

9.2.10 IMC_SetOverride_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_SetOverride_P	单轴调速功能块		IMC_SetOverride_P(AxisID:= , Execute:= , VelFactor:= , AccFactor:= , JerkFactor:= , Done=> , Busy=> , Error=> , ErrorID=>);

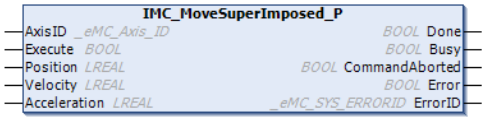
相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴 ID	_eMc_Axis_ID	0~3	255	轴号
Execute	功能块的使能位	BOOL	TRUE,FALSE	FALSE	上升沿有效，下降沿无效
VelFactor	指令速度	LREAL	正数	10	调速后的指令速度，(unit/s)
AccFactor	加速度值	LREAL	正数	10	调速后的加速度值，(unit/s^2)
JerkFactor	跃度值	LREAL	正数或 0	0	调速后的跃度值(unit/s^3)
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Done	功能块完成状态	BOOL	TRUE,FALSE	FALSE	TRUE：调速完成， FALSE：调速未完成
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行中 FALSE：功能块未执行
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE：出错 FALSE：未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcError _NULL	错误码

功能说明：Execute 上升沿触发功能块，将运动参数（速度、加速度、加速度变化时间等）调节到用户的设定值。

9.2.11 IMC_MoveSuperImposed_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_MoveSuperImposed_P	单轴位置叠加功能块		IMC_MoveSuperImposed_P(AxisID:= , Execute:= , Position:= , Velocity:= , Acceleration:= , Done=> , Busy=> , CommandAborted=> , Error=> , ErrorID=>);

相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴 ID	_eMc_Axis_ID	0~3	255	轴号
Execute	功能块的使能位	BOOL	TRUE,FALSE	FALSE	上升沿有效， 下降沿无效
Position	补偿距离	LREAL	≥-10,≤10	0.2	叠加的补偿距离(unit)
Velocity	运行速度	LREAL	＞0,≤5	2	速度(unit/s)
Acceleration	加速度	LREAL	正数	10	加速度(unit/s^2)
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Done	功能块完成状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行完成 FALSE：功能块未执行完成
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE：功能块执行中 FALSE：功能块未执行
CommandAborted	功能块中断状态	BOOL	TRUE,FALSE	FALSE	TRUE：被中断， FALSE：未被中断
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE：出错， FALSE：未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcError_NULL	错误码

功能说明：单轴位置叠加功能，在当前轴轨迹上叠加一段轨迹，仅支持单轴绝对运动、单轴相对运动、电子凸轮、飞剪、追剪功能块的位置叠加。

9.2.12 IMC_ReadCmdSpeed_P

指令格式：

指令	名称	图形表现	ST 表现
IMC_ReadCmdSpeed_P	单轴指令速度读取功能块		IMC_ReadCmdSpeed_P(AxisID:= , Enable:= , Busy=> , Error=> , ErrorID=> , ActiveSpeed=>);

相关变量：

输入变量	名称	数据类型	有效范围	初始值	描述
AxisID	轴 ID	_eMc_Axis_ID	0~3	255	轴号
Enable	功能块的使能位	BOOL	TRUE,FALSE	FALSE	高电平有效，低电平无效
输出变量	**名称**	**数据类型**	**有效范围**	**初始值**	**描述**
Busy	功能块执行状态	BOOL	TRUE,FALSE	FALSE	TRUE-功能块执行中 FALSE-功能块未执行
Error	功能块出错标志	BOOL	TRUE,FALSE	FALSE	TRUE-出错 FALSE-未出错
ErrorID	错误码	_eMc_Sys_ErrorID	-	_mcError_NULL	错误码
ActiveSpeed	速度值	LREAL	正数或 0	0	轴对应的实际速度，(unit/s)

功能说明：读取指定单轴的速度指令，由 ActiveSpeed 返回轴的实际速度指令。

注意：单轴速度读取不可在多轴运行状态下调用。

10 故障代码

10.1 SMC_ERROR 故障代码（402 轴通用错误信息）

错误序号	模块	枚举变量	描述
0	All function blocks	SMC_NO_ERROR	没有错误
1	DriveInterface	SMC_DI_GENERAL_COMMUNICATION_ERROR	通讯错误（例如 Sercos 环断裂）
2	DriveInterface	SMC_DI_AXIS_ERROR	轴错误
10	DriveInterface	SMC_DI_SWLIMITS_EXCEEDED	允许范围内的位置输出(SWLimit)
11	DriveInterface	SMC_DI_HWLIMITS_EXCEEDED	硬件限位开关被激活
13	DriveInterface	SMC_DI_HALT_OR_QUICKSTOP_NOT_SUPPORTED	驱动器状态停止或者不支持快速停止
14	DriveInterface	SMC_DI_VOLTAGE_DISABLED	驱动器没有使能
15	DriveInterface	SMC_DI_IRREGULAR_ACTPOSITION	驱动器当前给予的位置格式不正确。检查通讯。
16	DriveInterface	SMC_DI_POSITIONLAGERROR	位置滞后错误。在设置和当前位置超过限制值
20	All motion generating function blocks	SMC_REGULATOR_OR_START_NOT_SET	控制器没有使能或者抱闸没有打开
21	Axis in wrong controller mode	SMC_WRONG_CONTROLLER_MODE	轴不是一个正确的控制方式
30	DriveInterface	SMC_FB_WASNT_CALLED_DURING_MOTION	运动控制创建的模块在运动结束之前没有被调用。
31	All function blocks	SMC_AXIS_IS_NO_AXIS_REF	给出的 AXIS_REF 变量不是 AXIS_REF 类型
32	Axis in wrong controller mode	SMC_AXIS_REF_CHANGED_DURING_OPERATION	AXIS_REF-变量的返回值在模块激活前被处理
33	DriveInterface	SMC_FB_ACTIVE_AXIS_DIABLED	轴在移动时没有被激活(MC_Power.bRegulatorOn)
34	All motion generating function blocks	SMC_AXIS_NOT_READY_FOR_MOTION	在当前状态下周不能处理当前命令
40	VirtualDrive	SMC_VD_MAX_VELOCITY_EXCEEDED	达到最大速度(fMaxVelocity)
41	VirtualDrive	SMC_VD_MAX_ACCELERATION_EXCEEDED	达到最大加速度(fMaxAcceleration)
42	VirtualDrive	SMC_VD_MAX_DECELERATION_EXCEEDED	达到最大减速度(fMaxDeceleration)
50	SMC_Homing	SMC_3SH_INVALID_VELACC_VALUES	无效的速度或者加速度值
51	SMC_Homing	SMC_3SH_MODE_NEEDS_HWLIMIT	模块需要使用结束限位开关（安全用途）
70	SMC_SetControllerMode	SMC_SCM_NOT_SUPPORTED	模式不支持
71	SMC_SetControllerMode	SMC_SCM_AXIS_IN_WRONG_STATE	在当前模式下使用的控制模式不支持

错误序号	模块	枚举变量	描述
75	SMC_SetTorque	SMC_ST_WRONG_CONTROLLER_MODE	轴不是一个正确的控制模式
80	SMC_ResetAxisGroup	SMC_RAG_ERROR_DURING_STARTUP	在轴组启动时发生错误
90	SMC_ChangeGearingRatio	SMC_CGR_ZERO_VALUES	不正确的变量
91	SMC_ChangeGearingRatio	SMC_CGR_DRIVE_POWERED	驱动器控制模式下不能更改传动比
92	SMC_ChangeGearingRatio	SMC_CGR_INVALID_POSPERIOD	不合适的位置周期(<=0)
110	MC_Power	SMC_P_FTASKCYCLE_EMPTY	轴在扫描周期内不包含任何信息(fTaskCycle=0)
120	MC_Reset	SMC_R_NO_ERROR_TO_RESET	轴没有错误复位
121	MC_Reset	SMC_R_DRIVE_DOESNT_ANSWER	轴没有执行错误复位
122	MC_Reset	SMC_R_ERROR_NOT_RESETTABLE	错误不能被复位
123	MC_Reset	SMC_R_DRIVE_DOESNT_ANSWER_IN_TIME	与轴之间的通讯没有回应
130	MC_ReadParameter, MC_ReadBoolParameter	SMC_RP_PARAM_UNKNOWN	参数序号未知
131	MC_ReadParameter, MC_ReadBoolParameter	SMC_RP_REQUESTING_ERROR	在将参数传送到驱动器过程中发生错误，参阅功能块实例 ReadDriveParameter 的错误 (SM_DriveBasic.lib)
140	MC_WriteParameter, MC_WriteBoolParameter	SMC_WP_PARAM_INVALID	参数序号未知或者不允许进行写操作
141	MC_WriteParameter, MC_WriteBoolParameter	SMC_WP_SENDING_ERROR	参阅模块实例 WriteDriveParameter 的错误 (Drive_Basic.lib)
170	MC_Home	SMC_H_AXIS_WASNT_STANDSTILL	轴不是标准状态
171	MC_Home	SMC_H_AXIS_DIDNT_START_HOMING	在执行回零时发生错误
172	MC_Home	SMC_H_AXIS_DIDNT_ANSWER	通讯错误
173	MC_Home	SMC_H_ERROR_WHEN_STOPPING	执行回零错误停止。查阅是否设置减速度
180	MC_Stop	SMC_MS_UNKNOWN_STOPPING_ERROR	停止时发生完未知错误
181	MC_Stop	SMC_MS_INVALID_ACCDEC_VALUES	不合适的速度或者加速度值
182	MC_Stop	SMC_MS_DIRECTION_NOT_APPLICABLE	Direction=shortest 不可用
183	MC_Stop	SMC_MS_AXIS_IN_ERRORSTOP	轴位于错误停止状态，停止不能被处理
184	MC_Stop	SMC_BLOCKING_MC_STOP_WASNT_CALLED	一个 MC_Stop 的实例，锁定轴(Execute=TRUE)，不能进行调用。请调用 MC_Stop(Execute=FALSE).

错误序号	模块	枚举变量	描述
201	MC_MoveAbsolute	SMC_MA_INVALID_VELACC_VALUES	不合适的速度或者加速度值
202	MC_MoveAbsolute	SMC_MA_INVALID_DIRECTION	方向错误
226	MC_MoveRelative	SMC_MR_INVALID_VELACC_VALUES	不合适的速度或者加速度值
227	MC_MoveRelative	SMC_MR_INVALID_DIRECTION	方向错误
251	MC_MoveAdditive	SMC_MAD_INVALID_VELACC_VALUES	不合适的速度或者加速度值
252	MC_MoveAdditive	SMC_MAD_INVALID_DIRECTION	方向错误
276	MC_MoveSuperImposed	SMC_MSI_INVALID_VELACC_VALUES	不合适的速度或者加速度值
277	MC_MoveSuperImposed	SMC_MSI_INVALID_DIRECTION	方向错误
301	MC_MoveVelocity	SMC_MV_INVALID_ACCDEC_VALUES	不合适的速度或者加速度值
302	MC_MoveVelocity	SMC_MV_DIRECTION_NOT_APPLICABLE	Direction=shortest/fastest 不支持
325	MC_PositionProfile	SMC_PP_ARRAYSIZE	错误排列尺寸
326	MC_PositionProfile	SMC_PP_STEP0MS	步时间=t#0s
350	MC_VelocityProfile	SMC_VP_ARRAYSIZE	错误排列尺寸
351	MC_VelocityProfile	SMC_VP_STEP0MS	步时间=t#0s
375	MC_AccelerationProfile	SMC_AP_ARRAYSIZE	错误排列尺寸
376	MC_AccelerationProfile	SMC_AP_STEP0MS	步时间=t#0s
400	MC_TouchProbe	SMC_TP_TRIGGEROCCUPIED	触发条件已经被激活
401	MC_TouchProbe	SMC_TP_COULDNT_SET_WINDOW	驱动器接口不支持窗口功能
402	MC_TouchProbe	SMC_TP_COMM_ERROR	通讯错误
410	MC_AbortTrigger	SMC_AT_TRIGGERNOTOCCUPIED	触发条件已经被终止
426	SMC_MoveContinuousRelative	SMC_MCR_INVALID_VELACC_VALUES	不合适的速度或者加速度值
427	SMC_MoveContinuousRelative	SMC_MCR_INVALID_DIRECTION	方向错误
451	SMC_MoveContinuousAbsolute	SMC_MCA_INVALID_VELACC_VALUES	不合适的速度或者加速度值
452	SMC_MoveContinuousAbsolute	SMC_MCA_INVALID_DIRECTION	方向错误
453	SMC_MoveContinuousAbsolute	SMC_MCA_DIRECTION_NOT_APPLICABLE	Direction=fastest 不可用
600	SMC_CamRegister	SMC_CR_NO_TAPPETS_IN_CAM	CAM 中不包含任何挺杆
601	SMC_CamRegister	SMC_CR_TOO_MANY_TAPPETS	挺杆组 ID 达到 MAX_NUM_TAPPETS
602	SMC_CamRegister	SMC_CR_MORE_THAN_32_ACCESSES	在一个 CAM_REF 中多于 32 个接口
625	MC_CamIN	SMC_CI_NO_CAM_SELECTED	没有 CAM 被选中

错误序号	模块	枚举变量	描述
626	MC_CamIN	SMC_CI_MASTER_OUT_OF_SCALE	主轴超出范围
627	MC_CamIN	SMC_CI_RAMPIN_NEEDS_VELACC_VALUES	针对 ramp_in 功能块速度和加速度必须被精确指定
628	MC_CamIN	SMC_CI_SCALING_INCORRECT	比例变量 fEditor/TableMasterMin/Max 不正确
640	SMC_CAMBounds, SMC_CamBounds_Pos	SMC_CB_NOT_IMPLEMENTED	给予的 CAM 格式的功能块不支持
675	MC_GearIn	SMC_GI_RATIO_DENOM	RatioDenominator=0
676	MC_GearIn	SMC_GI_INVALID_ACC	加速度不合适
677	MC_GearIn	SMC_GI_INVALID_DEC	减速度不合适
725	MC_Phase	SMC_PH_INVALID_VELACCDEC	速度，加速度，减速度不合适
726	MC_Phase	SMC_PH_ROTARYAXIS_PERIOD0	旋转轴 fPositionPeriod=0
750	All modules using MC_CAM_REF as input	SMC_NO_CAM_REF_TYPE	给予的 CAM 不是类型 MC_CAM_REF
751	MC_CamTableSelect	SMC_CAM_TABLE_DOES_NOT_COVER_MASTER_SCALE	如果从 CamTable 中获取的数据不是通过数据转化得到的主轴区域(xStartandxEnd)。
775	MC_GearInPos	SMC_GIP_MASTER_DIRECTION_CHANGE	在从轴耦合过程中主轴改变旋转方向
800	SMC_BacklashCompensation	SMC_BC_BL_TOO_BIG	齿轮返回比(fBacklash)太大 (>positionperiode/2)
1000	CNC function blocks which are supervising the licensing	SMC_NO_LICENSE	目标没有进行 CNC 的授权。
1001	SMC_Interpolator	SMC_INT_VEL_ZERO	路径不能被处理因为速度=0.
1002	SMC_Interpolator	SMC_INT_NO_STOP_AT_END	上一个路径对象 Vel_End>0.
1003	SMC_Interpolator	SMC_INT_DATA_UNDERRUN	警告：GEOINFO-列表在 DataIn 进行处理，但是列表最后没有被设置。理由：忘记在 DataIn 中设置 EndOfList 或者 SMC_Interpolator 比路径编译模块处理速度快
1004	SMC_Interpolator	SMC_INT_VEL_NONZERO_AT_STOP	停止速度>0.
1005	SMC_Interpolator	SMC_INT_TOO_MANY_RECURSIONS	使用太多 SMC_Interpolator 调用 SoftMotion-错误。
1006	SMC_Interpolator	SMC_INT_NO_CHECKVELOCITIES	Input-OutQueueDataIn 没有作为 SMC_CHeckVelocities 的最后处理模块
1007	SMC_Interpolator	SMC_INT_PATH_EXCEEDED	内部/数值错误错误
1008	SMC_Interpolator	SMC_INT_VEL_ACC_DEC_ZERO	速度，加速度或者减速度为空或者太低。
1009	SMC_Interpolator	SMC_INT_DWIPOTIME_ZERO	FB 调用 dwIpoTime=0
1050	SMC_Interpolator2Dir	SMC_INT2DIR_BUFFER_TOO_SMALL	数据缓冲区太小

错误序号	模块	枚举变量	描述
1051	SMC_Interpolator2Dir	SMC_INT2DIR_PATH_FITS_NOT_IN_QUEUE	路径没有完全包含在队列中
1100	SMC_CheckVelocities	SMC_CV_ACC_DEC_VEL_NON POSITIVE	速度，减速度或者加速度值不为正向
1120	SMC_Controlaxisbypos	SMC_CA_INVALID_ACCDEC_VALUES	变量 offGapVelocity/ fGapAcceleration/ fGapDeceleration 不是正值
1200	SMC_NCDecoder	SMC_DEC_ACC_TOO_LITTLE	加速度值不允许
1201	SMC_NCDecoder	SMC_DEC_RET_TOO_LITTLE	减速度值不允许
1202	SMC_NCDecoder	SMC_DEC_OUTQUEUE_RAN_EMPTY	低于 Queue 的数据被读取并且为空。
1203	SMC_NCDecoder	SMC_DEC_JUMP_TO_UNKNOWN_LINE	因为行号未知所以跳转的行号不能执行
1204	SMC_NCDecoder	SMC_DEC_INVALID_SYNTAX	语法错误
1205	SMC_NCDecoder	SMC_DEC_3DMODE_OBJECT_NOT_SUPPORTED	这些对象不支持 3D 模式
1300	SMC_GCodeViewer	SMC_GCV_BUFFER_TOO_SMALL	缓冲区太小
1301	SMC_GCodeViewer	SMC_GCV_BUFFER_WRONG_TYPE	缓冲区元素类型错误
1302	SMC_GCodeViewer	SMC_GCV_UNKNOWN_IPO_LINE	当前插补行不能被找到
1500	All function blocks using SMC_CNC_REF	SMC_NO_CNC_REF_TYPE	给定的 CNC 程序不是类型 SMC_CNC_REF
1501	All function blocks using SMC_OUTQUEUE	SMC_NO_OUTQUEUE_TYPE	给定的 OutQueue 不是类型 SMC_OUTQUEUE
1600	CNC function blocks	SMC_3D_MODE_NOT_SUPPORTED	这个功能块只在 2D 路径中可用
2000	SMC_ReadNCFile	SMC_RNCF_FILE_DOESNT_EXIST	文件不存在
2001	SMC_ReadNCFile	SMC_RNCF_NO_BUFFER	没有缓冲分配
2002	SMC_ReadNCFile	SMC_RNCF_BUFFER_TOO_SMALL	缓冲区太小
2003	SMC_ReadNCFile	SMC_RNCF_DATA_UNDERRUN	缓冲区中低缓冲数据被读取，为空
2004	SMC_ReadNCFile	SMC_RNCF_VAR_COULDNT_BE_REPLACED	占位符变量不能被替换
2005	SMC_ReadNCFile	SMC_RNCF_NOT_VARLIST	输入的 pvl 不能指向 SMC_VARLIST 对象
2050	SMC_ReadNCQueue	SMC_RNCQ_FILE_DOESNT_EXIST	文件不能打开
2051	SMC_ReadNCQueue	SMC_RNCQ_NO_BUFFER	没有缓冲区定义
2052	SMC_ReadNCQueue	SMC_RNCQ_BUFFER_TOO_SMALL	缓冲区太小
2053	SMC_ReadNCQueue	SMC_RNCQ_UNEXPECTED_EOF	未知文件结尾
2100	SMC_AxisDiagnosticLog	SMC_ADL_FILE_CANNOT_BE_OPENED	文件不能被打开

错误序号	模块	枚举变量	描述
2101	SMC_AxisDiagnosticLog	SMC_ADL_BUFFER_OVERRUN	超过范围的缓冲；WriteToFile 必须更经常的调用
2200	SMC_ReadCAM	SMC_RCAM_FILE_DOESNT_EXIST	文件不能打开
2201	SMC_ReadCAM	SMC_RCAM_TOO_MUCH_DATA	保存到 CAM 数据太多
2202	SMC_ReadCAM	SMC_RCAM_WRONG_COMPILE_TYPE	错误编译模式
2203	SMC_ReadCAM	SMC_RCAM_WRONG_VERSION	文件版本错误
2204	SMC_ReadCAM	SMC_RCAM_UNEXPECTED_EOF	未知的文件结尾
3001	SMC_WriteDriveParamsToFile	SMC_WDPF_CHANNEL_OCCUPIED	文件频道发生占用
3002	SMC_WriteDriveParamsToFile	SMC_WDPF_CANNOT_CREATE_FILE	文件不能被创建
3003	SMC_WriteDriveParamsToFile	SMC_WDPF_ERROR_WHEN_READING_PARAMS	读取文件参数的时候错误
3004	SMC_WriteDriveParamsToFile	SMC_WDPF_TIMEOUT_PREPARING_LIST	准备参数列表时时间错误
5000	SMC_Encoder	SMC_ENC_DENOM_ZERO	译码器参考的转换因子(dwRatioTechUnitsDenom)为 0
5001	SMC_Encoder	SMC_ENC_AXISUSEDBYOTHERFB	其他模块正在处理译码轴
5002	DriveInterface	SMC_ENC_FILTER_DEPTH_INVALID	过滤器选择不合适

10.2 PLC 故障代码表（适用 TM、TP 系列 PLC）

错误类型		错误定位	主错误码	子错误码	错误说明
CPU	系统相关	硬件故障	0001	0001	纽扣电池未安装或电池电压过低
				0002	设备供电电压过低（小于 19V）
	系统组件相关	时钟系统组件故障	0008	0001	设置时间错误
				0002	写 RTC 时钟出错
				0003	读 RTC 时钟出错
		IP 系统组件故障	0009	0001	IP1 与 IP2 的 IP 网段重复报错
				XXX	预留
				0011	读：IP1 模块--打开文件报错
				0012	读：IP1 模块--无法获取 IP 信息
				0013	写：IP1 模块--IP 地址配置出错
				0014	写：IP1 模块--掩码配置出错
				0015	写：IP1 模块--网关配置出错
				0016	写：IP1 模块--与 USB 网段重复报错
				0017	写：IP1 模块--IP 与网关不再同一网段报错
				XXX	预留
				0021	读：IP2 模块--打开文件报错
				0022	读：IP2 模块--无法获取 IP 信息
				0023	写：IP2 模块--IP 地址出错
				0024	写：IP2 模块--掩码出错
				0025	写：IP2 模块--网关出错
				0026	写：IP2 模块--与 USB 网段重复报错

错误类型		错误定位	主错误码	子错误码	错误说明
				0027	写：IP2 模块--IP 与网关不再同一网段报错
背板总线	背板总线相关	CPU IO 故障	0030	0001	模块组态故障
				0002	模块参数配置故障
		数字量故障	0031	0001	DI--模块组态故障
				0002	DI--模块参数配置故障
				XXX	预留
				2001	DO--模块组态故障
				2002	DO--模块参数配置故障
				2003	DO--模块输出端口供电故障
				2004	DO--模块输出故障
				XXX	预留
				XXX	预留
		模拟量故障	0032	0001	模块组态故障
				XXX	预留
				0012	AD--通道 0 参数配置故障
				0015	AD--通道 0 信号源开路故障
				0016	AD--通道 0 采样信号超极限故障
				0017	AD--通道 0 采样信号超量程上限故障
				0018	AD--通道 0 采样信号超量程下限故障
				XXX	预留
				0022	AD--通道 1 参数配置故障
				0025	AD--通道 1 信号源开路故障
				0026	AD--通道 1 采样信号超极限故障
				0027	AD--通道 1 采样信号超量程上限故障
				0028	AD--通道 1 采样信号超量程下限故障
				XXX	预留
				0032	AD--通道 2 参数配置故障
				0035	AD--通道 2 信号源开路故障
				0036	AD--通道 2 采样信号超极限故障
				0037	AD--通道 2 采样信号超量程上限故障
				0038	AD--通道 2 采样信号超量程下限故障
				XXX	预留
				0042	AD--通道 3 参数配置故障
				0045	AD--通道 3 信号源开路故障
				0046	AD--通道 3 采样信号超极限故障
				0047	AD--通道 3 采样信号超量程上限故障
				0048	AD--通道 3 采样信号超量程下限故障
				XXX	预留
				0003	模块输出端口供电故障
				XXX	预留
				2012	通道 0 参数配置故障
				2014	通道 0 输出故障
				XXX	预留
				2022	通道 1 参数配置故障
				2024	通道 1 输出故障
				XXX	预留
				2032	通道 2 参数配置故障
				2034	通道 2 输出故障
				XXX	预留
				2042	通道 3 参数配置故障
				2044	通道 3 输出故障
				XXX	预留

错误类型		错误定位	主错误码	子错误码	错误说明
		测温模块故障	0033	0001	模块组态故障
				XXX	预留
				0012	通道 0 参数配置故障
				0015	通道 0 信号源开路故障
				0017	通道 0 采样信号超量程上限故障
				0018	通道 0 采样信号超量程下限故障
				XXX	预留
				0022	通道 1 参数配置故障
				0025	通道 1 信号源开路故障
				0027	通道 1 采样信号超量程上限故障
				0028	通道 1 采样信号超量程下限故障
				XXX	预留
				0032	通道 2 参数配置故障
				0035	通道 2 信号源开路故障
				0037	通道 2 采样信号超量程上限故障
				0038	通道 2 采样信号超量程下限故障
				XXX	预留
				0042	通道 3 参数配置故障
				0045	通道 3 信号源开路故障
				0047	通道 3 采样信号超量程上限故障
				0048	通道 3 采样信号超量程下限故障
现场总线	Modbus 相关	Modbus_RTU Master1	0040	0001	非法功能码
				0002	非法地址
				0003	数据个数错误
				0004	从站设备故障
				0005	通讯超时，通讯时间超过了用户设定的通讯最大时间发生了错误
				XXX	预留
				0008	接收到的数据帧不符合 Modbus 协议
				0009	CRC/LRC 校验错误
				XXX	预留
				000B	接收到的数据长度不符合协议规定 或者数量超过该功能码规定的最大限制
				000C	接收到的从站地址和请求的从站地址不匹配
				000D	接收到的功能码和请求的功能码不匹配
				000E	命令执行失败
		Modbus_RTU Master2	0041	0001	非法功能码
				0002	非法地址
				0003	数据个数错误
				0004	从站设备故障
				0005	通讯超时，通讯时间超过了用户设定的通讯最大时间发生了错误
				XXX	预留
				0008	接收到的数据帧不符合 Modbus 协议
				0009	CRC/LRC 校验错误
				XXX	预留
				000B	接收到的数据长度不符合协议规定 或者数量超过该功能码规定的最大限制
				000C	接收到的从站地址和请求的从站地址不匹配
				000D	接收到的功能码和请求的功能码不匹配
				000E	命令执行失败
		Modbus_RTU	0042	0001	非法功能码

错误类型		错误定位	主错误码	子错误码	错误说明
		Slave1		0002	非法地址
				0003	数据个数错误
				0004	从站设备故障
				0005	通讯超时，通讯时间超过了用户设定的通讯最大时间发生了错误
				XXX	预留
				0008	接收到的数据帧不符合 Modbus 协议
				0009	CRC/LRC 校验错误
				XXX	预留
				000B	接收到的数据长度不符合协议规定 或者数量超过该功能码规定的最大限制
				000C	接收到的从站地址和请求的从站地址不匹配
				000D	接收到的功能码和请求的功能码不匹配
				000E	命令执行失败
		Modbus_RTU Slave2	0043	0001	非法功能码
				0002	非法地址
				0003	数据个数错误
				0004	从站设备故障
				0005	通讯超时，通讯时间超过了用户设定的通讯最大时间发生了错误
				XXX	预留
				0008	接收到的数据帧不符合 Modbus 协议
				0009	CRC/LRC 校验错误
				XXX	预留
				000B	接收到的数据长度不符合协议规定 或者数量超过该功能码规定的最大限制
				000C	接收到的从站地址和请求的从站地址不匹配
				000D	接收到的功能码和请求的功能码不匹配
				000E	命令执行失败
		ModbusTCP Master1	00A0	0001	非法功能码
				0002	非法地址
				0003	数据个数错误
				0004	从站设备故障
				0005	通讯超时，通讯时间超过了用户设定的通讯最大时间发生了错误
				XXX	预留
				0008	接收到的数据帧不符合 Modbus 协议
				0009	CRC/LRC 校验错误
				XXX	预留
				000B	接收到的数据长度不符合协议规定 或者数量超过该功能码规定的最大限制
				000C	接收到的从站地址和请求的从站地址不匹配
				000D	接收到的功能码和请求的功能码不匹配
	ModbusTCP 相关			000E	命令执行失败
		ModbusTCP Master2	00A1	0001	非法功能码
				0002	非法地址
				0003	数据个数错误
				0004	从站设备故障
				0005	通讯超时，通讯时间超过了用户设定的通讯最大时间发生了错误
				XXX	预留
				0008	接收到的数据帧不符合 Modbus 协议

错误类型		错误定位	主错误码	子错误码	错误说明
				0009	CRC/LRC 校验错误
				XXX	预留
				000B	接收到的数据长度不符合协议规定 或者数量超过该功能码规定的最大限制
				000C	接收到的从站地址和请求的从站地址不匹配
				000D	接收到的功能码和请求的功能码不匹配
				000E	命令执行失败
		ModbusTCP Slave1	00A2	0001	非法功能码
				0002	非法地址
				0003	数据个数错误
				0004	从站设备故障
				0005	通讯超时，通讯时间超过了用户设定的通讯最大时间发生了错误
				XXX	预留
				0008	接收到的数据帧不符合 Modbus 协议
				0009	CRC/LRC 校验错误
				XXX	预留
				000B	接收到的数据长度不符合协议规定 或者数量超过该功能码规定的最大限制
				000C	接收到的从站地址和请求的从站地址不匹配
				000D	接收到的功能码和请求的功能码不匹配
				000E	命令执行失败
		ModbusTCP Slave2	00A3	0001	非法功能码
				0002	非法地址
				0003	数据个数错误
				0004	从站设备故障
				0005	通讯超时，通讯时间超过了用户设定的通讯最大时间发生了错误
				XXX	预留
				0008	接收到的数据帧不符合 Modbus 协议
				0009	CRC/LRC 校验错误
				XXX	预留
				000B	接收到的数据长度不符合协议规定 或者数量超过该功能码规定的最大限制
				000C	接收到的从站地址和请求的从站地址不匹配
				000D	接收到的功能码和请求的功能码不匹配
				000E	命令执行失败

值得信赖的工控与能效解决方案提供者

深圳市英威腾电气股份有限公司
地址：深圳市光明区马田街道松白路英威腾光明科技大厦

苏州英威腾电力电子有限公司
地址：苏州高新区科技城昆仑山路1号

服务热线：400-700-9997　网址：www.invt.com.cn

英威腾微信公众号　英威腾电子手册

66001-01366

产品资料可能有所改动，恕不另行通知。版权所有，仿冒必究。

202408 (V1.0)