M218 SoMachine 指令手册

M218 SoMachine
指令手册

SoMachine

Schneider Electric

施耐德电气
善用其效 尽享其能

全球能效管理专家施耐德电气为世界100多个国家提供整体解决方案，其中在能源与基础设施、工业过程控制、楼宇自动化和数据中心与网络等市场处于世界领先地位，在住宅应用领域也拥有强大的市场能力。致力于为客户提供安全、可靠、高效的能源，施耐德电气2010年的销售额为196亿欧元，拥有超过110,000名员工。施耐德电气助您——善用其效，尽享其能！

施耐德电气在中国

1987年，施耐德电气在天津成立第一家合资工厂梅兰日兰，将断路器技术带到中国，取代传统保险丝，使得中国用户用电安全性大为增强，并为断路器标准的建立作出了卓越的贡献。90年代初，施耐德电气旗下品牌奇胜率先将开关面板带入中国，结束了中国使用灯绳开关的时代。

施耐德电气的高额投资有力地支持了中国的经济建设，并为中国客户提供了先进的产品支持和完善的技术服务，中低压电器、变频器、接触器等工业产品大量运用在中国国内的经济建设中，促进了中国工业化的进程。

目前，施耐德电气在中国共建立了77个办事处，26家工厂，6个物流中心，1个研修学院，3个研发中心，1个实验室，700多家分销商和遍布全国的销售网络。施耐德电气中国目前员工数近22,000人。通过与合作伙伴以及大量经销商的合作，施耐德电气为中国创造了成千上万个就业机会。

施耐德电气 EcoStruxure™ 能效管理平台

凭借其对五大市场的的深刻了解、对集团客户的悉心关爱，以及在能效管理领域的丰富经验，施耐德电气从一个优秀的产品和设备供应商逐步成长为整体解决方案提供商。今年，施耐德电气首次集成其在建筑楼宇、IT、安防、电力及工业过程和设备等五大领域的专业技术和经验，将其高质量的产品和解决方案融合在一个统一的架构下，通过标准的界面为各行业客户提供一个开放、透明、节能、高效的EcoStruxure™能效管理平台，为企业客户节省高达30%的投资成本和运营成本。

前　言

尊敬的客户，非常感谢您购买施耐德电气的小型PLC产品，在此向您深表谢意！

M218是施耐德电气基于前瞻性的控制理念和丰富的自动化经验开发的一款一体式高性能的小型PLC，特别适用于小型的自动化设备。

自2011年初发布至今，M218凭借其卓越的性能和具有竞争力的价格，极大的降低了设备生产厂商对于设备的设计、安装、调试和维护的成本。借助于SoMachine自动化平台，M218可以方便的和施耐德电气的HMI，VSD，Motion等产品组成针对OEM控制系统的解决方案。为了便于广大施耐德电气小型PLC的用户更快的了解SoMachine平台并熟练的掌握M218的编程，我们编写了此手册。

手册共分三大部分：其中第1章到第6章为第一部分，主要介绍SoMachine软件的特点，这部分内容详细介绍了SoMachine软件平台的整体结构，程序语言，编辑器，以及如何在SoMachine下对一个项目进行管理等；第7章到第9章为第二部分，介绍M218的通用操作符，通用功能块库以及系统功能块库；第三部分是附录，以实例的方式对M218编程的快速入门，通讯功能，高速计数功能，脉冲输出功能，系统时钟RTC功能，以及结构文本ST语言等进行详尽的介绍。通过这些介绍以及一些实例，我们可以对整个SoMachine软件平台和M218的编程有一个深入的了解。

本手册的编写汇聚了施耐德电气OEM技术部门每一位技术精英的心血！每一个软件特点的介绍和每一条指令的使用方法都是各位技术专家在实际工作中的经验总结。

参与本手册编写的人员有：

蔡民辖　晁广安　杜汕波　方飞宇　龚耀平　胡子健　刘校仕　潘知翼　王晓军
王休才　吴永辉　吴玉华　杨小雷　杨晓萌　岳培劼　张　福　张继轼　张伟毅
张俊杰　吴江华　时勇祥　张英杰　盛积庆

参与本手册校对的人员有：

邓永辉　刘东斌　李银川　弋志伟　唐海丽　王亚峰　王海龙　李　振　秦　晖
杨　合　李　杰　石彦丽　梁天跃

向以上技术人员的辛勤工作与无私奉献表示崇高的敬意！

最后特别感谢施耐德电气工控事业部OEM销售副总裁艾小明先生以及控制与架构市场部经理陆伯德（Charly Lupart）先生对于此手册编写工作的大力支持和指导！

OEM技术中心 LEC市场部
2012-3-31

注意事项

在尝试安装、操作或维护设备之前，请仔细阅读下述说明并通过查看来熟悉设备。 下述特别信息可能会在本文其他地方或设备上出现，提示用户潜在的危险，或者提醒注意有关阐明或简化某一过程的信息。

在“危险”或“警告”安全标签上添加此符号表示存在触电危险，如果不遵守使用说明，将导致人身伤害。

这是提醒注意安全的符号。提醒用户可能存在人身伤害的危险。请遵守所有带此符号的安全注意事项，以避免可能的人身伤害甚至死亡。

⚠ 危险

“危险”表示极可能存在危险，如果不遵守说明，可导致严重的人身伤害甚至死亡。

⚠ 警告

“警告”表示可能存在危险，如果不遵守说明，可导致严重的人身伤害甚至死亡，或设备损坏。

⚠ 警告

“注意”表示可能存在危险，如果不遵守说明，可导致严重的人身伤害或设备损坏。

注意

注意(无安全警告符号)。表示存在潜在的危险，如果忽视，可能导致设备损坏。

请注意

电气设备的安装、操作、维修和维护工作仅限于合格人员执行。对于使用本资料所引发的任何后果，Schneider Electric 概不负责。

专业人员是指掌握与电气设备的制造和操作相关的技能和知识的人员，他们经过安全培训能够发现和避免相关的危险。

目 录

目 录

目 录

目 录

目 录

目 录

目 录

目 录

目 录

第一章 SoMachine编程

1.1.1 SoMachine是什么?

SoMachine是一款专业、高效且开放的OEM解决方案软件，可以在单一的环境中开发，组态和调试整台机器，包括逻辑程序，电机控制，HMI和相关的网络自动化功能。

SoMachine允许你在施耐德电气灵活的控制平台中编程和调试所有的器件设备。这个控制平台包含如下：

控制器
- HMI控制器，XBTGC，XBT GT/GK CANopen
- 逻辑控制器, M218/M238/M258
- 运动控制器, LMC058
- 集成的控制器卡,ATV IMC

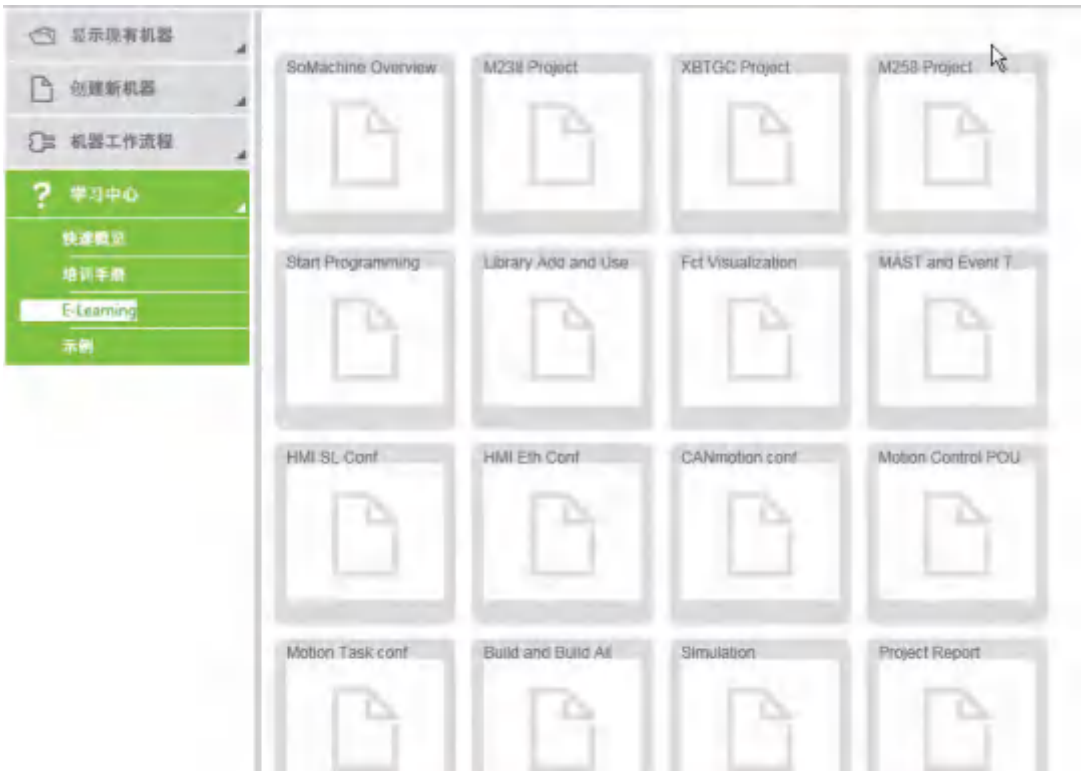
扩展/分布式I/O
- TM2/TM5/TM7

HMI
- XBTGT,XBTGK

1.1.2 SoMachine主要组成部分

可视化的图形用户接口(GUI)
用户可以很容易的组态项目的架构，以及组态架构中的各种设备

强大的学习中心
内嵌的学习中心提供了快速入门手册，完善的培训手册，E-learning以及各种类型的示例程序，初学者可以先浏览快速入门手册，了解关于SoMachine的基本入门知识，然后通过练习各种示例来深入学习SoMachine软件。



项目管理
SoMachine提供了”显示现有项目”,”创建新项目”两种方式来管理用户的项目。
“显示现有项目”(图一)：用户无需打开每个已存在的项目，即可快速的浏览每个项目的基本信息，然后根据这些信息来打开相应的项目。


图一

“创建新项目”(图二)：用户可以通过五种方式来创建新程序，即使用空项目启动，使用标准项目启动，使用TVDA启动，使用AFB启动，使用现有项目启动，大大的节省了用户的开发时间，无需重复创建项目，


图二

项目信息
用户在组态SoMachine项目时，可以手动的添加很多项目附加信息，例如，项目的系统架构图，项目的系统描述，机器实物图，工艺流程图，项目总结文档等等信息，为项目的标准化归档提供了诸多方面。

编程及调试

SoMachine提供了强大的编程工具，支持IEC61131-3的全部编程语言，满足不同用户的编程习惯和需求，内嵌种类众多的函数和功能块库，用户仅需简单调用即可，无需编制大量复杂的程序。仿真，单步执行，断点执行，跟踪等调试工具可供用户调试使用。

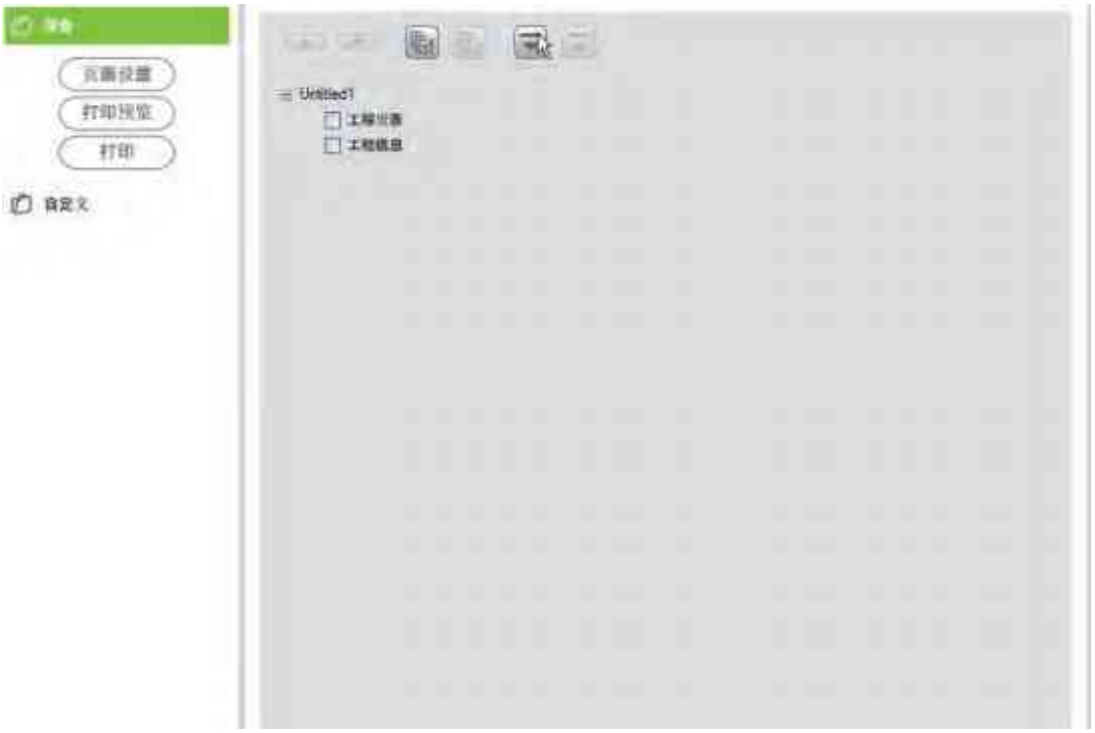
诊断

“在线”菜单提供了许多的在线诊断方式，用户可以在线监控项目的状态,方便解决项目中出现的错误或故障。

归档

项目文件归档是整个项目中很重要的一个环节。在SoMachine平台下，构建和自定义整个项目报告非常灵活：

- 选择条目加入到该报告
- 组织整个报告章节
- 定义页面的布局
- 打印整个报告



透明
SoMachine支持DTM(设备类型管理器,一个现场设备工具容器)。
在SoMachine中插入支持DTM类型的设备后，控制器和现场总线CANopen能够经由SoMachine与每个单个设备直接通讯，这样减少了连接到单个设备电缆数量，节省布线成本。

OEM应用库
SoMachine软件可以安装补丁包Solution Extension.该补丁包内集成了经测试的，验证的，归档的以及支持的专家应用库，这些库特定应用于一些OEM行业。它们的简单组态加速设计，调试，安装和诊断。当前，这些库覆盖如下几个行业：
- 包装行业
- 起重行业
- 物流输送行业

TVDA（测试的，验证的，归档的架构）
SoMachine内置了很多预置项目和可用的架构，你可选择其中之一使适合你的单个需求。这些架构是基于控制器组态的。Solution Extension 将提供详细的和具体的面向TVDA的应用解决方案。

1.1.3.SoMachine主要特征

- 支持IEC 61131-3定义的全部编程语言
- LD (梯形图)
- IL (指令列表)
- FBD (功能块图)
- SFC (顺序功能图)

- ST (结构化文本)
- CFC (连续功能图)

支持种类繁多的控制器服务
- 多任务：Mast,Fast,Event
- 功能和功能块
- 数据单元类型
- 在线下载
- 监视窗口
- 变量的图形化监控
- 断点，一步一步执行
- 仿真

支持基于HMI的服务
- 图形库，包含4000个左右的2D/3D对象
- 简单画图对象，例如，点，线
- 预组态对象，例如，按钮，图形条
- 配方
- 动作表
- 报警
- 打印
- Java脚本
- 多媒体文件支持：wav,png,jpg
- 变量趋势图

支持motion服务
- 嵌入设备的组态和调试
- CAM廓线图编辑器
- 采样应用程序追踪
- 运动和传动功能块库
- 可视化界面

支持全局服务
- 用户权限分配
- 项目文档打印
- 项目比较
- 基于发布/订阅机制的变量共享
- 库版本管理

集成现场总线组态器
- 控制网络：Modbus Serial line/Modbus TCP
- 现场总线：CANopen/CANmotion/AS-Interface
- 网关：Profibus-DP, Ethernet IP

专家和解决方案库
- 用于运动控制的PLCopen功能块，例如：MC_Power,MC_MoveAbsolute

- 包装功能块，例如，analog film tension control，rotary knife
- 起重功能块，例如，anti-sway
- 输送功能块，例如，tracking, turntable

1.1.4 SoMachine编程的优势

1. 变量名的编程方式

传统的PLC编程方式要求编程者先分配寄存器地址，然后在此基础上命名标签名。这样，编程者在编程之前必须先确定详细的I/O点数和中间寄存器数，然后分配相应的寄存器地址，最后再给这些寄存器地址命名标签名，

SoMachine的编程方式不同于传统的PLC编程方式，编程者可以无需考虑先分配具体的寄存器地址，可以直接先声明变量名，在程序的执行代码中使用变量名即可，编程者可以随时将某个特定变量名链接到某个寄存器地址，也可以随时更改链接到该变量名的地址，使整个编程过程更加灵活和方便，大大节省了编程者的开发时间。

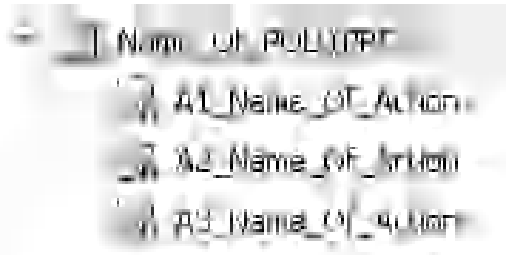
建议编程者采用变量名的方式来编程，在程序代码中尽量不用或少用直接寄存器地址参与编程。

2. 编程语言的优化选择

传统的PLC软件编程语言一般仅支持LD/IL/FBD,ST/SFC/CFC往往需要安装附加的语言包才能支持。SoMachine支持全部的6种编程语言，编程者可以根据实际的需要任意选择编程语言。对于SoMachine编程而言，建议编程者在选择编程语言时应根据实际的编程方便来选择编程语言，而不是在整个程序中仅使用LD.例如，编写算法程序时应优先考虑ST,编写顺序流程时应优先考虑SFC，编写逻辑控制时应优先考虑LD,编写功能块时应优先考虑CFC或FBD.

3. 优化的功能块结构

传统的PLC编程元素中FB都是单个体，FB与FB之间关联性不太大。而SoMachine的编程是面向对象的编程方式，支持如下的FB结构



.

1.2.1 POU创建的原则

POU: Programming Organization Unit,可编程组织单元。可以是程序，功能块或函数中的任一类型。下面一一详细的描述这3种类型的概念和区别：

程序：
在执行时能够返回一个或多个值的POU，程序内所有的变量值能够从本次程序执行结束保持到下一次程序执行。程序可以被其他的POU调用，但是函数中不可以调用程序。程序没有实例。

功能块：
可以提供一个或多个输出值的POU,不同于函数，功能块的输出变量值和内部变量值在每次调用后保持，从而影响下次调用时的运算(调用功能块时输入值一样，但是输出值不一定一样)。功能块有实例，调用功能块其实就是调用功能块的实例。

函数：
只有一个返回值的POU。与其它2种方式不同，函数在每次调用后不保存内部变量的值(本次函数调用时对函数内部变量的改变不会影响下一次调用)..在ST语言中，函数可以作为参数参与表达式运算。

建议广大的编程者，SoMachine编程中可以遵循如下的POU创建原则:
- MAST任务中只能调用程序，建议可以将您的项目按工艺或功能分成几段程序，在MAST中调用。
- 需要反复调用的程序段可以建成功能块，这样，在您的程序中只需调用你的功能块实例即可。例如，有100台开关阀需要通过程序来控制开或关，那么，在项目中只需要建立一个开关阀的功能块，再分别调用100次即可。
- 个别的算法程序可以建成函数，得出的结果可以参与表达式运算。

1.2.2 编程语言的选择

在上章节中讲述了SoMachine软将有别于传统的PLC编程软件，编程语言的多样性是一大优点。那么，编程者在选择编程语言时具体怎么选择呢？从优化程序和编程便利性的角度建议大家，涉及到算法部分请选择ST语言，编写的程序往往简洁而高效；涉及到流程控制部分，请选择SFC语言，编写的程序会条理清晰，逻辑关系不会混乱；涉及到逻辑控制部分，请选择LD语言，编写的联锁，互锁等逻辑简单易懂；涉及到功能块部分，请选择CFC或者FBD，编写的程序会形成一个网络清晰的网状电路图，易于读懂。当然，在实际的编程时，用户也可以根据自己的使用习惯来选择编程语言，虽然实现的方法不同，但是都能得到同一个结果。

1.2.3 变量名的使用

强烈的建议编程者在使用SoMachine时，使用变量名代替直接寄存器地址的方式来编程，其优势这里不再做多赘述。用户可以在项目的前期使用变量名编写程序，待I/O地址确定后再链接到相应的变量上。

变量名的命名是有一定规则的，请给变量命名时请尽量遵守“匈牙利命名法”：
- 每一个变量的基本名字中应该包含一个有意义的简短描述;
- 基本名字中每一个单词的首字母应当大写，其它字母则为小写;

• 依据变量的数据类型，在基本名字之前加上小写字母前缀;
请参照下表:

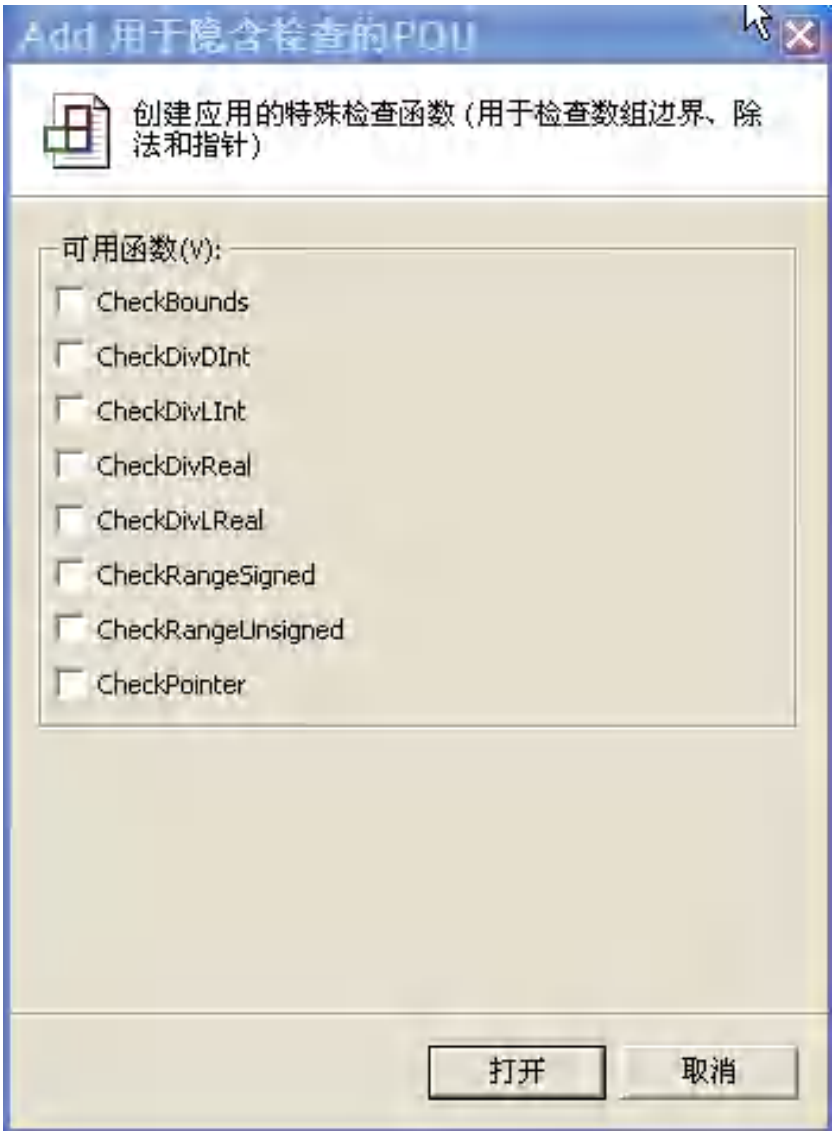
数据类型	下限	上限	数据长度	前缀
BOOL	FALSE	TRUE	1	x*
				b
BYTE			8	by
WORD			16	w
DWORD			32	dw
LWORD			64	lw
SINT	-128	127	8	si
USINT	0	255	8	usi
INT	-32,768	32,767	16	i
UINT	0	65,535	16	ui
DINT	-2,147,483,648	2,147,483,647	32	di
UDINT	0	4,294,967,295	32	udi
LINT	-2^{63}	$2^{63} - 1$	64	li
ULINT	0	$2^{64} - 1$	64	uli
REAL			32	r
LREAL			64	lr
STRING				s
TIME				tim
TIME_OF_DAY				tod
DATE_AND_TIME				dt
DATE				date
ENUM			16	e
POINTER				p
ARRAY				a

1.2.4 “下载”例外的处理方法

很多编程者在编写程序的过程中由于思路往往不够缜密，导致程序在下装运行后，PLC会提示”下装例外错误”,而令编程者烦恼的是如何确认故障的原因。在这里，给大家总结一下产生该错误的可能原因，以免大家在编写的过程中可以提前避免。

1. 算术运算/指针/数组的限制
在程序的编写过程中，可能会发生如下的几种情况：
1.除法运算的被除数在某些情况下会为零;
2.指针在赋值的过程中可能不小心指向空地址；
3.调用数组时数组边界别溢出了；

以上几种情况都会导致PLC的可能停止，因此，在程序中必须加以处理。那么如何处理呢?请参照如下步骤:
1．右键点击”Application”中的”添加对象”，弹出对象菜单；
2．选择菜单中的”用于隐含检查的POU”,弹出如下对话框:



上图中，一些特殊检查函数可供用户选择，请将前面的框中打勾即可加入到现有的程序中，对可能的算法/指针/数组错误进行屏蔽.

2. MAST任务的选择
每个项目程序中仅且只能有一个MAST任务，目的在于循环执行用户的程序。那么任务的类型选择至关重要。SoMachine明确指出MAST任务只能选择自由扫描任务或循环任务的一种，那么该选择那一种呢?

在这里，建议大家优先选择循环任务
注意：
1. 如果为某个循环任务定义的周期过短，则该任务会在写入输出后立即重复，而不会执行其他较低优先级的任务或任何系统处理。这将会影响所有任务的执行并导致控制器超过系统看门狗限制，从而导致系统看门狗例外.

2. 可以使用 GetCurrentTaskCycle 和 SetCurrentTaskCycle 功能通过应用程序获取和设置循环执行任务间隔.

第二章 任务的动作

第二章 任务的动作

2.1 任务的概念：

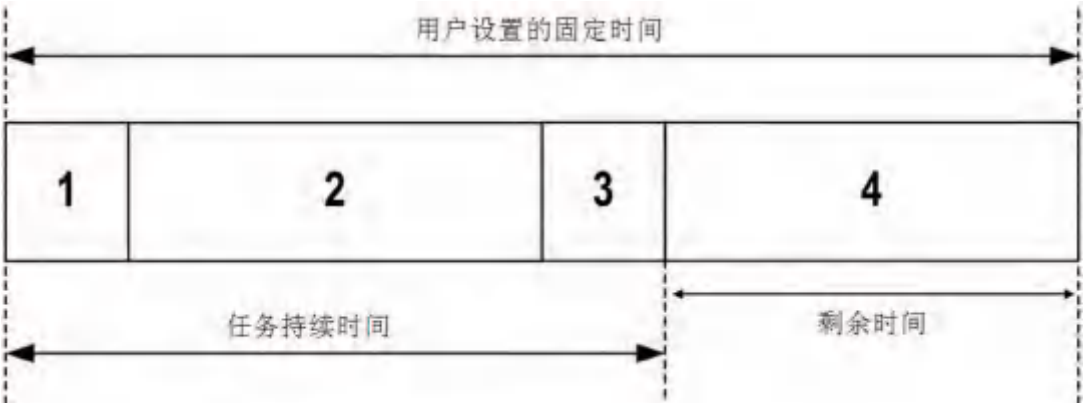
在SoMachine中“任务”指的是交派给M218的工作。SoMachine 设备树中的“ 任务配置” 节点用于定义一个或多个“任务”以控制应用程序的执行。

2.2 任务类型：

- 循环
- 自由运行
- 事件
- 外部事件

2.2.1循环任务

循环任务：是指用户设置一个固定的时间周期，M218按照此时间周期进行执行，如果设置的时间小于程序执行需要的时间，M218将在运行中报错。循环任务类型的执行方式如下：



1. 读取输入：将输入%Ix的状态写入输入存储器，并执行其他系统操作。

2. 任务处理：处理任务中定义的用户代码（POU 等）。在此操作期间， %Qx 输出存储器变量会根据应用程序指令进行更新，但不会写入物理输出。

3. 刷新输出：%Qx 输出存储器变量使用已定义的任何输出强制进行修改，但物理输出的写入取决于输出类型和所用指令。

有关 I/O 行为的详细信息，请参阅控制器状态的详细描述

注意： 扩展 I/O 始终由 MAST 任务进行物理更新。

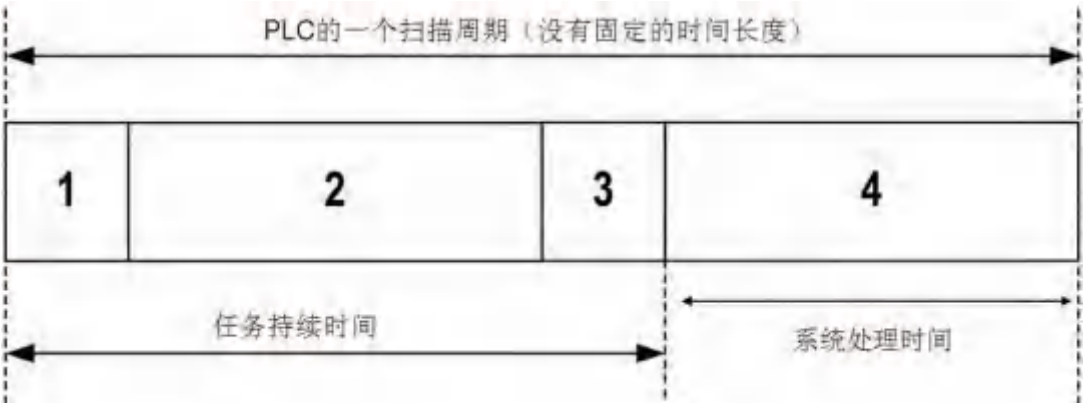
4. 剩余时间：控制器操作系统执行系统处理和任何其他较低优先级的任务。

注意： 如果为某个循环任务定义的周期过短，则该任务会在写入输出后立即重复，而不会执行其他较低优先级的任务或任何系统处理。这将会影响所有任务的执行并导致控制器超过系统看门狗限制，从而导致系统看门狗例外。

注意： 可以使用 GetCurrentTaskCycle 和 SetCurrentTaskCycle 功能（需要增加Toolbox_Advance库），通过应用程序获取和设置循环执行任务间隔。

2.2.2自由运行任务

自由运行任务没有固定持续时间。在自由运行模式下，每个任务扫描都在前一个扫描完成时以及短时间系统处理后开始。每个自由运行任务类型的执行方式如下：



1. 读取输入：将输入%Ix的状态写入输入存储器，并执行其他系统操作。

2. 任务处理：处理任务中定义的用户代码（POU 等）。在此操作期间， %Qx 输出存储器变量会根据应用程序指令进行更新，但不会写入物理输出。

3. 刷新输出：%Qx 输出存储器变量使用已定义的任何输出强制进行修改，但物理输出的写入取决于输出类型和所用指令。

有关 I/O 行为的详细信息，请参阅控制器状态的详细描述

注意：扩展 I/O 始终由 MAST 任务进行物理更新。

4. 系统处理：控制器操作系统执行系统处理和任何其他较低优先级的任务。系统处理周期的长度设置为前 3 个操作总持续时间的 30% (4 = 30 % * (1 + 2 + 3))。在任何情况下，系统处理周期都不会少于 3 毫秒。

注意：在自由运行模式中，每次任务扫描都在上次扫描完成，以及系统处理（自由运行任务总持续时间的30%）一段时间后开始。如果由于其他任务中断而使系统处理时间减少，幅度低于自由运行任务总持续时间的15%且时间超过3秒，则会检测到系统错误。有关详细信息，请参阅系统看门狗。

建议在有高优先级且耗时的任务运行时，不在多任务应用程序中使用自由运行任务

2.2.3事件任务

此类型的任务由内部事件驱动，即由程序变量启动。除非有更高优先级的任务先于此事件任务执行，否则事件任务在与触发事件关联的布尔变量的上升沿时启动。在此情况下，事件任务会根据任务优先级分配的指示启动。

2.2.4外部事件任务

此类型的任务由外部事件驱动，即通过检测到硬件或硬件相关的功能事件启动。除非有更高优先级的任务先于外部事件任务执行，否则它会在事件发生时启动。在此情况下，外部事件任务会根据任务优先级分配的指示启动。

2.3任务数

可为 Modicon M218 Logic Controller 定义的最大任务数为：

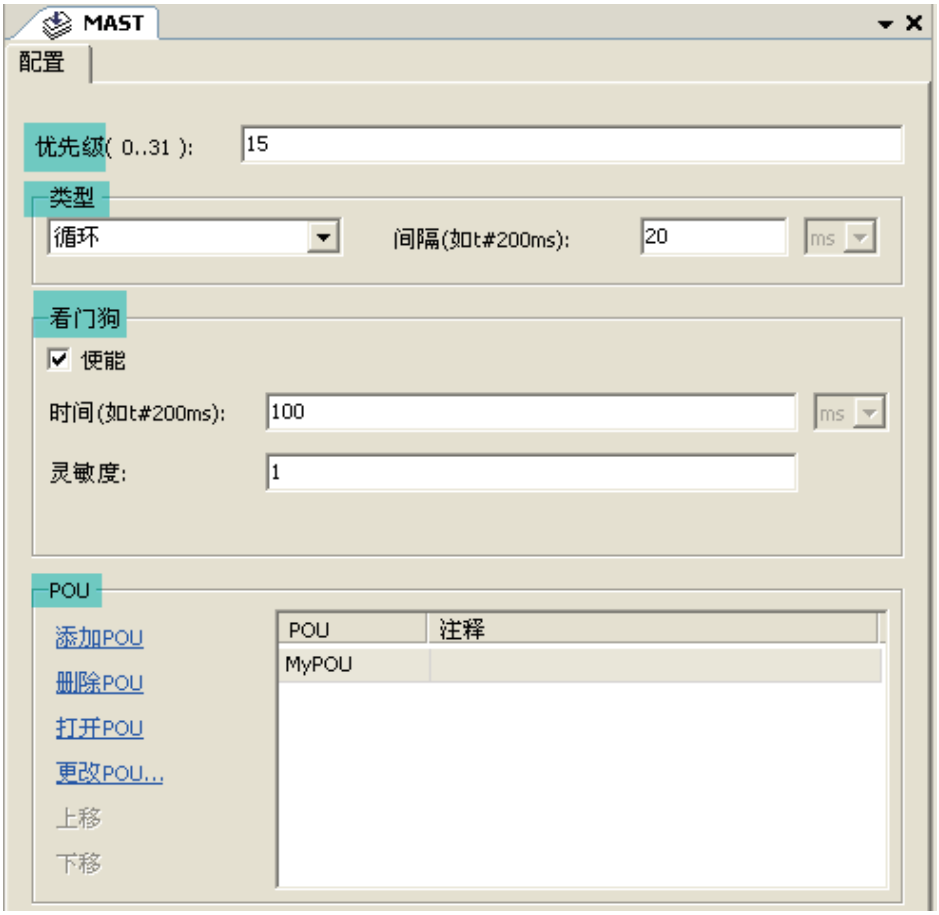
- 总任务数 = 7
- 循环任务数 = 3
- 自由运行任务数 = 1
- 事件任务数 = 2
- 外部事件任务数 = 4

注意：自由运行任务、循环任务和事件任务的总数不应大于 3。

2.4任务配置

按照以下步骤访问任务配置：

步骤	操作
1	单击“程序”菜单
2	在左边的“设备”窗口的“设备树”中双击要配置的任务： 

3	每个配置任务都有自己的参数，这些参数与其他任务无关，需要配置下面四个部分： 1、 优先级 2、 任务类型 3、 看门狗 4、 程序组织单位（POU  	
	优先级	您可以使用0 到31这些数字配置每个任务的优先级（0 表示最高优先级，31 表示最低优先级）。 一次仅能运行一个任务。任务的优先级确定何时运行该任务： ● 优先级高的任务先于优先级低的任务执行 ● 具有相同优先级的任务将轮流运行（每个任务执行2毫秒） 注意： 请勿分配具有相同优先级的任务。如果还存在其他任务试图先于具有相同优先级的任务执行，则结果可能不确定且不可预知。有关详细信息，请参考任务优先级
	类型	可以使用下列 4 种任务类型： ● 循环 ● 自由运行 ● 事件 ● 外部事件
	看门狗	要配置看门狗，必须定义两个参数： ● 时间：如果程序执行时间超过这个设定时间，看门狗执行。

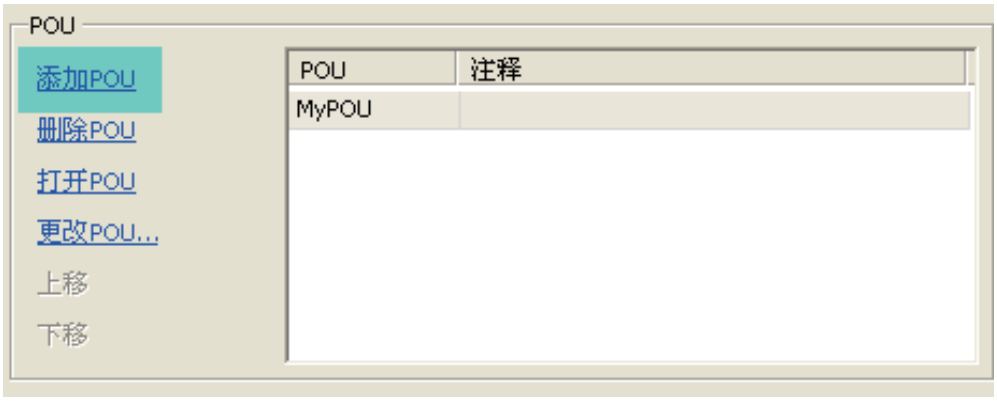
要配置看门狗，必须定义两个参数：

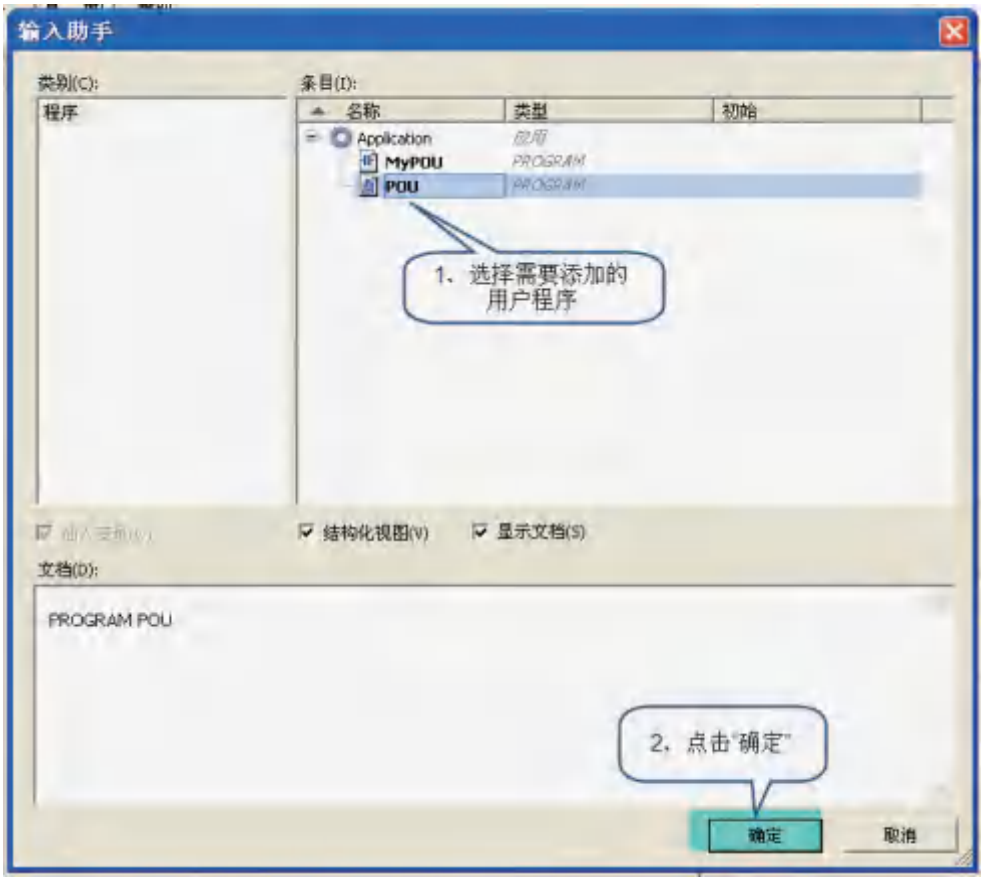
- 时间：如果程序执行时间超过这个设定时间，看门狗执行。
- 灵敏度：如果看门狗的计数次数大于等于设定的灵敏数， PLC进入“停止”状态

由任务控制的 POU 列表在任务配置窗口中定义。要添加POU到任务，请使用命令“添加POU”。要从列表中删除POU，请使用命令“删除POU”。

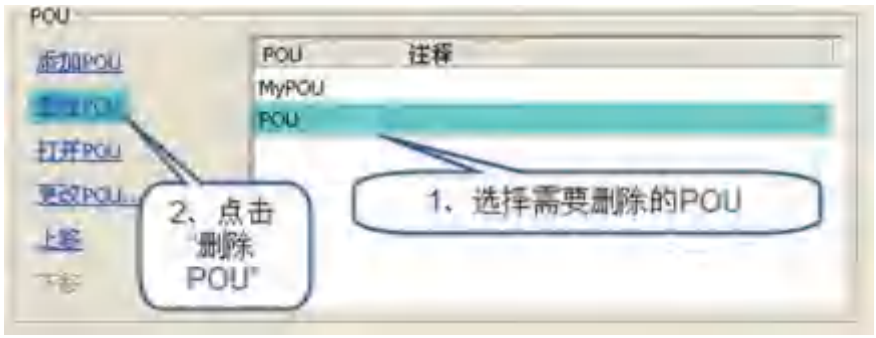
可创建所需数量的POU。如果应用程序使用多个小型 POU（而不是一个大型 POU），则会延长在线模式下的变量刷新时间。

- 添加 POU





- 删除 POU



2.5系统和任务看门狗:

2.5.1简介
M218有两种类型的看门狗：

• 系统看门狗：
这些看门狗在PLC的操作系统（固件）中定义并由其管理，用户无法配置这些看门狗。

• 任务看门狗：
可为每个任务定义可选的看门狗。这些看门狗由应用程序管理，可在SoMachine中进行配置。

2.5.2系统看门狗
M218 有两个系统看门狗。它们由控制器操作系统（固件）进行管理，因此在 SoMachine 在线帮助中有时候也称为硬件看门狗。当其中一个系统看门狗超过其阈值条件时，会检测到错误。

2 个系统看门狗的阈值条件定义如下：
• 如果所有任务需要 80 % 以上的处理器资源，且时间超过 3 秒，将检测到系统错误。控制器进入“ 空” 状态。

• 如果在 20 秒的间隔过程中没有执行优先级最低的系统任务，将检测到系统错误。控制器会以自动重新启动进入“ 空” 状态进行响应。

注意： 用户无法配置系统看门狗。

2.5.3任务看门狗
SoMachine允许您为应用程序中的每个任务配置可选的看门狗。（在SoMachine在线帮助中，任务看门狗有时候也称为软件看门狗或控制定时器）。当您定义的任务看门狗之一达到其阈值条件时，将检测到应用程序错误并且控制器会进入“ 暂停” 状态。

在定义任务看门狗时，可使用以下选项：

• 时间：这定义允许的最长任务执行时间。当任务所用时间超过此值时，控制器将报告任务看门狗例外。

• 灵敏度：“ 灵敏度 ” 字段用于定义必须在控制器检测到应用程序错误之前发生的任务看门狗例外数。

任务看门狗在各个任务的“ 任务配置” 选项卡的“ 配置” 子选项卡上进行配置。要访问此选项卡，请双击设备树中的任务。

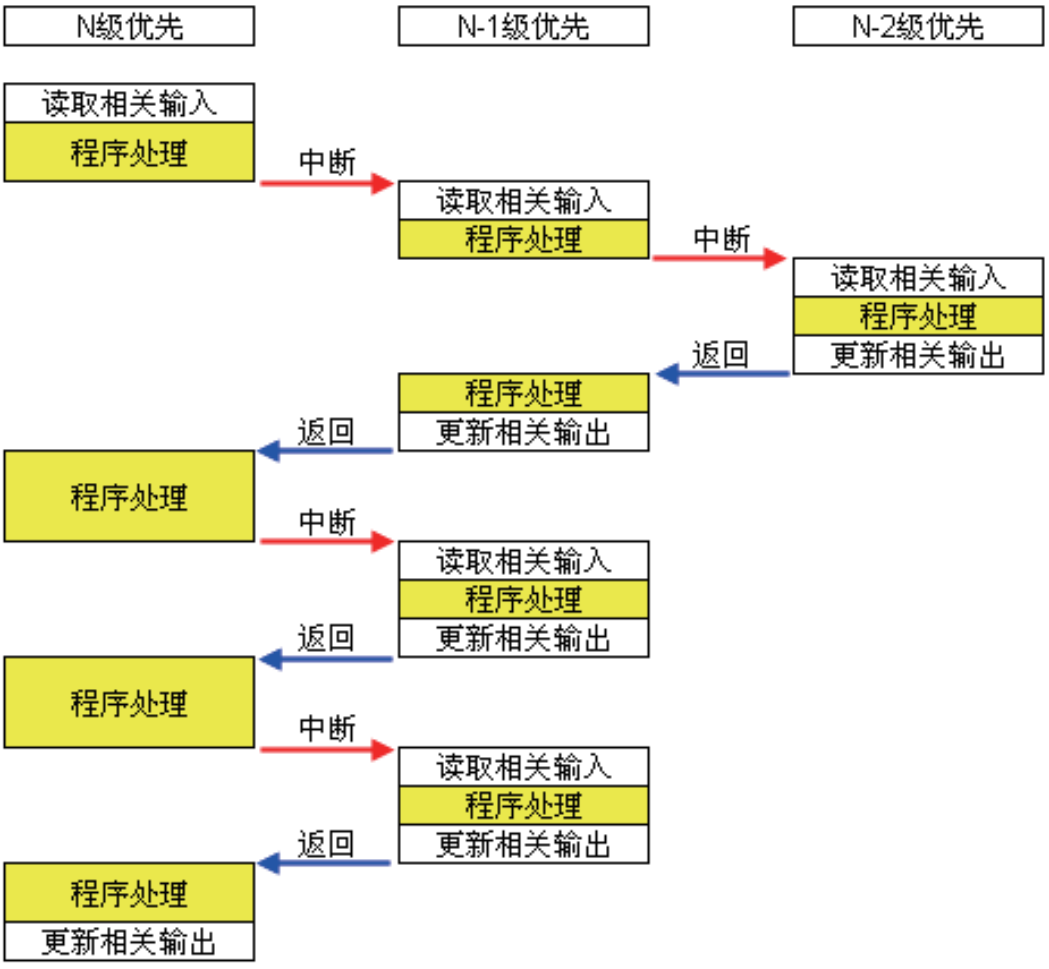
注意： 有关看门狗的详细信息，请参阅 CoDeSys 在线帮助。

2.6任务优先级

可以将各个任务的优先级配置为 0 到 31 （0 表示最高优先级， 31 表示最低优先级）。每个任务的优先级必须唯一。如果向多个任务分配相同的优先级，则这些任务的执行会无法确定且不可预测，这可能会导致意外后果。

警告
意外的设备操作 请勿将相同的优先级分配给不同的任务。 如果不遵守这些说明，将会导致死亡、严重伤害或设备损坏。

某个任务循环开始后，它可以中断任何优先级较低的任务（任务抢占）。优先级较高的任务循环完成后，被中断的任务将恢复。



注意： 如果在不同任务中使用相同输入，则输入映像可能会在较低优先级任务的任务循环期间发生更改。

要提高在多任务期间执行正确输出行为的可能性，则在不同任务中使用相同字节中的输出时，应检测到错误。

警告
意外的设备操作 请勿将相同的优先级分配给不同的任务。 如果不遵守这些说明，将会导致死亡、严重伤害或设备损坏。

2.7缺省任务配置

对于M218：可以在“自由运行”或“循环”模式下配置 MAST 任务。

缺省情况下，MAST 任务在循环模式下自动创建。其预设优先级为中(15)，预设时间间隔为20毫秒，任务看门狗服务已激活，设定的时间为100毫秒，灵敏度为1。有关优先级设置的详细信息，请参阅任务优先级 。有关看门狗的详细信息，请参阅看门狗 。

设计高效应用程序对于实现多任务的系统至关重要。在此类应用程序中，可能难以使资源利用率始终低于系统看门狗阈值。如果只是重新分配优先级不足以保持低于阈值，则当 SysTaskWaitSleep 功能添加到一些较低优先级任务时，可以使这些任务使用较少的系统资源。有关此功能的详细信息，请参阅可选的系统 SysTask 库/ 库的 SysLibs 类别。

注意： 请勿删除或更改 MAST 任务的名称。如果这么做， SoMachine 会在您尝试生成应用程序时检测到错误，您将无法将其下载到控制器。

第三章 编程语言

Somachine支持IEC61131-3所描述的所有语言：
文本化的语言：
- 指令表（IL）
- 结构化文本（ST）

图形化的语言：
- 梯形图（LD）
- 功能模块图（FBD）
- 顺序功能流程图（SFC）
- 连续功能图（CFC）

3.1 指令表（IL）

指令表编程语言与汇编语言类似，是一种助记符编程语言，由操作符和操作数组成。
例如：

```
VAR
    input1: BOOL;
    output1: BOOL;
    input2: BOOL;
    T1: TON;
    T1_en: BOOL;
    T1_PT: TIME := T#10S;
    T1_ET: TIME;
END_VAR
```

```
LD          input1
ST          output1
LD          input2
R           %QX0.0
CAL         T1(
        IN:= T1_en,
        PT:= T1_PT,
        ET=> T1_ET)
```

上述程序中，LD/ST/R/CAL为操作符；input1/output1等为操作数。

操作数可为变量、常量、寄存器地址、函数等，如：input1为变量，T1_PT为常量，%QX0.0为输出点地址。

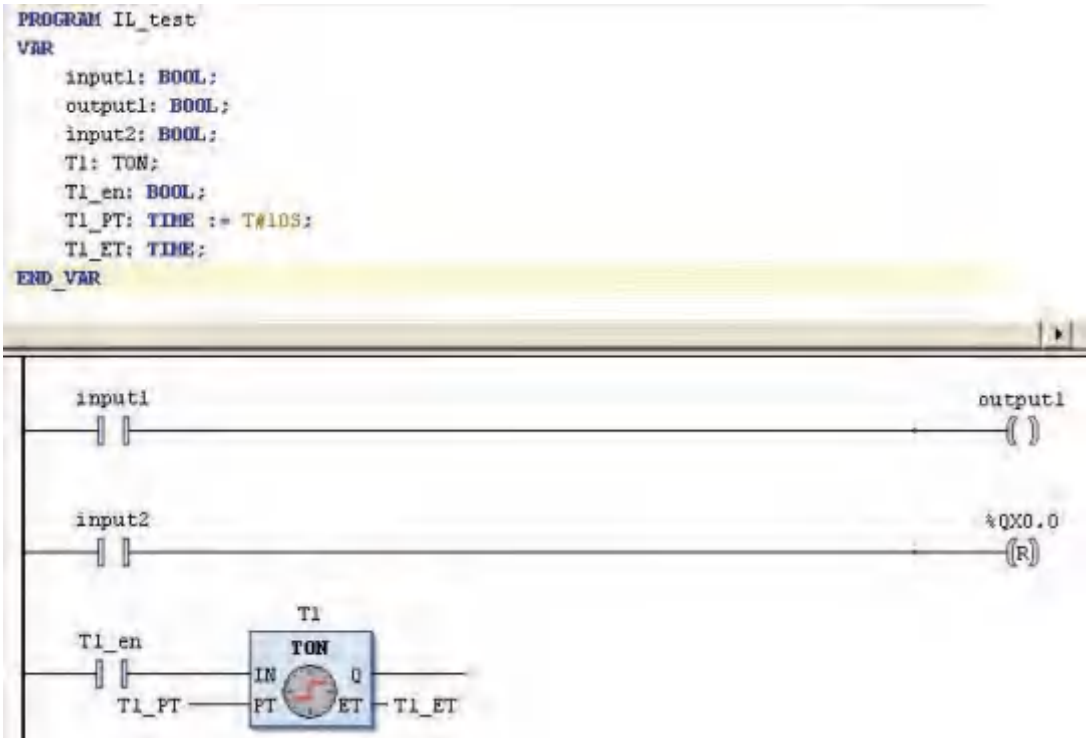
常用操作符如下所示：

ADD	加法
SUB	减法
MUL	乘法
DIV	除法
MOD	取余
AND(N)	与（非）
OR(N)	或（非）
XOR(N)	异或（非）
NOT	非
SHL	左移
SHR	右移
ROL	循环左移
ROR	循环右移
GT	大于
GE	大于等于
LT	小于
LE	小于等于
EQ	等于
NE	不等于

关于操作符的详细说明，请参见手册第七章：《操作符说明》。

在Somachine软件中，指令表（IL）与梯形图（LD）、功能块图（FBD）之间可以相互转换。

上例的IL程序可转换为梯形图如下：



3.2 结构化文本（ST）

3.2.1概述

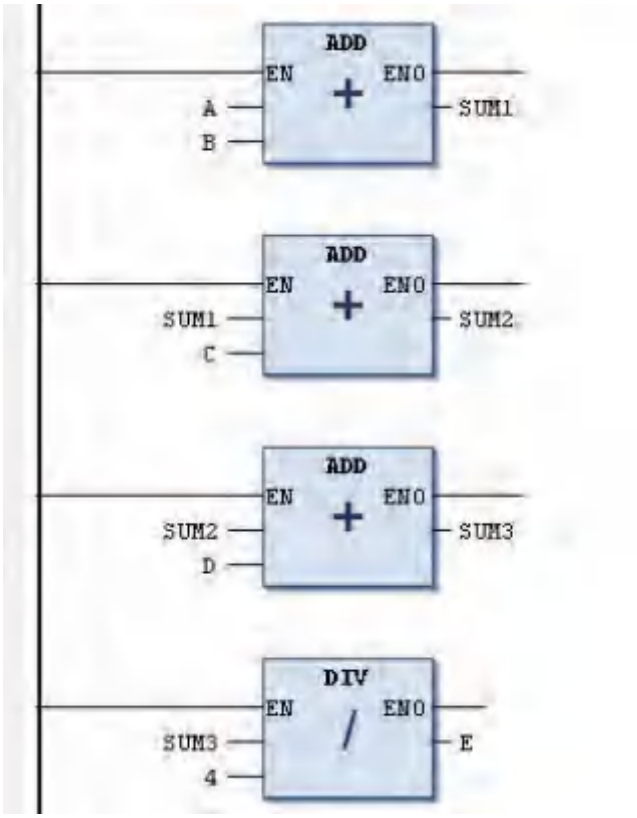
结构化文本，Structured Text，简写为ST，是用结构化的描述文本来编写程序的一种编程语言。

ST语言的特点是“高级文本编程”和“结构化”，适合于算法和结构较为复杂，其它编程语言（如梯形图、FBD等）实现比较困难的情况。具有高效、快捷、简洁的优点。

例：现在有A，B，C，D四个整形变量，要求得到这四个数的平均数，并赋值给E。

梯形图语言编写：

```
PROGRAM POU
VAR
    A: INT;
    B: INT;
    C: INT;
    D: INT;
    SUM1: INT;
    SUM2: INT;
    SUM3: INT;
    E: INT;
END_VAR
```



ST语言编写：

```
PROGRAM POU
VAR
    A: INT;
    B: INT;
    C: INT;
    D: INT;
    E: INT;
END_VAR
```

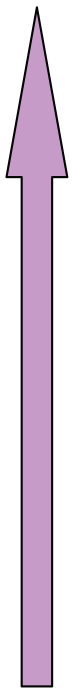
```
E:=(A+B+C+D)/4;
```

3.2.2 语法结构

一、表达式
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。

例如：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。
操作符的优先级如下表所示：

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

于操作符的详细说明，请参见手册第七章：《操作符说明》。

二、指令说明
1. 赋值指令
变量 := 变量或表达式
意义：操作符":="右边变量或表达式的值被赋给了左边的变量
例如：Var2:=Var1*10;
该语句把Var1的值乘以10，再将得到的结果赋值给Var2。

2 IF指令

使用IF指令可以检查条件，并根据此条件执行相应的指令。

常见的IF指令结构有：

(1) IF 条件A THEN

表达式A;

END_IF

当条件A满足时，执行表达式A。

例如：
IF temp<17
THEN heating_on := TRUE;
END_IF

当温度降到17度以下时，开启加热器。

(2) IF 条件A THEN
表达式A;
ELSE
表达式B;
END_IF
当条件A满足时，执行表达式A；否则，执行表达式B。

例如：
IF temp<17
THEN heating_on := TRUE;
ELSE heating_on := FALSE;
END_IF;

当温度降到17度以下时，开启加热器，否则加热器保持关闭。

（3） IF 条件A THEN 表达式A；
ELSIF 条件B THEN 表达式B；
...
ELSIF 条件N-1 THEN 表达式N-1；
ELSE 表达式N；
END_IF

当条件A满足时，执行表达式A；否则，当条件B满足时，执行表达式B；...否则，当(条件N-1)满足时，执行(表达式N-1)；如果以上条件都不满足，则执行表达式N；指令结束。

例如：
IF temp<17
THEN heating_on := TRUE;
ELSIF temp<30

THEN Heating_on := FALSE; Colding_on:= FALSE;
ELSE
Colding_on:= TRUE;
END_IF

当温度降到17度以下时，开启加热器；当温度升高到30度以上时，开启制冷器；当温度在17度~30度之间时，加热器、制冷器均保持关闭。

3 CASE指令

语法：
CASE <控制变量> OF
<数值1>: <表达式1>
<数值2>: <表达式2>
<数值3, 数值4, 数值5>: <表达式3>
<数值6 .. 数值10>: <表达式4>
...
<数值n>: <表达式n>
ELSE <ELSE的表达式>
END_CASE
CASE指令用于将控制变量和若干个操作数进行比较，如果控制变量与其中一个值相同，则执行该值对应的语句。如果与任何一个值都不相同，则执行ELSE指令的语句。

例如：
CASE INT1 OF
1, 5: BOOL1 := TRUE;
2: BOOL2 := TRUE;
10..20: BOOL3:= TRUE;
ELSE
BOOL1 := BOOL2 :=BOOL3 :=FALSE;
END_CASE
当INT1=1或5时，BOOL1为真；
当INT1=2时，BOOL2为真；
当INT1=10~20之间的数值时，BOOL3为真；
否则，BOOL1、BOOL2、BOOL3均为假。

当使用IF指令有过多分层，或者需要使用多个ELSIF，才能完成程序功能时，使用CASE指令替代IF指令，可以简化程序，并且能提高程序的可读性。

4 FOR循环指令

FOR循环指令用于一些需要重复执行的语句，它可以使程序简短并且一目了然。但需要注意避免陷入死循环。

FOR循环指令是有限制的循环指令，当限制条件满足（变量值等于“循环结束时变量值”）时，程序就将退出FOR循环，执行下一条指令。

语法：
FOR <循环控制变量> := <循环开始时变量值> TO <循环结束时变量值>{BY <变量递增步长>}
DO
<表达式>
END_FOR
其中，{}内语句可根据需要省略，省略时步长默认为1。

例如：
FOR Counter:=1 TO 5 BY 1 DO
Var1:=Var1*2;
END_FOR

此程序的循环控制变量为Counter，循环开始时控制变量初值为1，每一次循环Counter+1；当Counter等于5时，执行完FOR循环内容后，退出循环，执行下一条语句。
语句Var1:=Var1*2一共执行5次；假设Var1的初始值是1，那么循环结束后，Var1的值为32。

注意：<循环结束时变量值>不能等于其数据类型的最大值，否则会进入死循环。

例如，假设上例中所使用的计数变量Counter的类型是SINT（范围从-128到127），如果语句为
“FOR Counter:=1 TO 127 BY 1 DO”
则会进入死循环。编程时应避免此类情况的发生。

5 WHILE循环指令
WHILE循环与FOR循环使用方法类似。二者的不同之处是，WHILE循环的结束条件不是指定的循环次数，而是任意的逻辑表达式。当满足该表达式叙述的条件满足时，执行循环。

语法：
WHILE <循环条件>
<表达式>
END_WHILE
WHILE循环执行前先检查<循环条件>是否为TRUE，如果为TRUE，则执行<表达式>；当执行完一次后，再次检查<循环条件>，如果仍为TRUE，则再次执行，直到<循环条件>为FALSE。如果一开始<循环条件>就为FALSE，则不会执行WHILE循环里的指令。

例如：
WHILE Counter<>0 DO
 Var1 := Var1*2;
 Counter := Counter-1;
END_WHILE

此程序只要Counter不等于0，则一直会执行WHILE循环中的指令，直到Counter等于0为止。每执行一次循环，通过指令“Counter := Counter-1”使Counter的值减1，当Counter等于0时，循环结束。

注意：WHILE循环因为没有循环次数的限定，因此相对FOR循环更容易发生死循环；因此

可以在循环指令的内容中，增加语句来避免死循环的产生。如上述程序中的“Counter := Counter-1”即可避免程序进入死循环。

6 REPEAT循环指令
REPEAT循环与WHILE循环一样，也是没有明确循环次数的循环。与WHILE循环的区别在于，REPEAT循环在指令执行以后，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。

语法：
```
REPEAT
<表达式>
UNTIL <循环结束条件>
END_REPEAT
```
语句一直执行，直到<循环结束条件>为TRUE时，REPEAT循环结束。如果<循环结束条件>一开始就为TRUE，则循环只执行一次。

例如，上述WHILE示例程序也可写为：
```
REPEAT
 Var1 := Var1*2;
 Counter := Counter-1;
UNTIL
 Counter=0
END_REPEAT
```
注意：REPEAT循环同样需要避免死循环的发生。

在一定的意义上，WHILE循环和REPEAT循环比FOR循环的功能更加强大，这是因为在执行循环前，WHILE循环和REPEAT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。

7 CONTINUE指令
CONTINUE指令可以在FOR、WHILE和REPEAT三种循环中使用，其作用为中断本次循环，直接执行下次循环。

例如：
```
FOR Counter:=1 TO 5 BY 1 DO
INT1:= INT1/2;
IF INT1=0 THEN
CONTINUE;  (* to avoid division by zero *)
END_IF
Var1:=Var1/INT1;  (* only executed, if INT1 is not "0" *)
END_FOR
```
此程序使用CONTINUE指令，当INT1等于0时，直接结束本次循环，开始下一次循环，以避免指令“Var1:=Var1/INT1”中对INT1的除零操作。

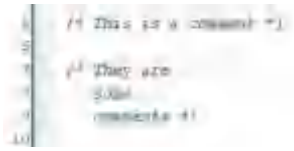
8 EXIT指令
EXIT用于退出FOR循环、WHILE循环、REPEAT循环。例如：
```
FOR Counter:=1 TO 5 BY 1 DO
INT1:= INT1/2;
IF INT1=0 THEN
EXIT;  (* to avoid division by zero *)
END_IF
Var1:=Var1/INT1;
END_FOR
```
当INT1等于0时，FOR循环结束。

9. RETURN指令
返回指令，用于根据一定条件退出POU。
例如：
```
IF b=TRUE THEN
   RETURN;
END_IF
a := a + 1;
```
如果b为TRUE，将不会执行a := a +1，而是直接退出POU。

10. JMP指令
```
<label>:
JMP <label>;
```
跳转指令，跳转到label所在的位置执行程序。
例如：
```
i := 0;
label: i := i+1;
......
IF i<10 THEN
  JMP label;
END_IF
```
当i<10时，跳转回label所在行，执行i:=i+1。
JMP指令容易造成程序结构混乱，降低代码可读性，不建议使用。

三、在ST中调用功能块
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ST中调用TON定时器，假设其实例名为TON1：

四、在ST中添加注释
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） 注释以 (* 开始，以 *) 结束。这种注释方法允许多行注释。
例如：



（2） 注释以“//”开始，一直到本行结束。这是单行注释的方法。
例如：

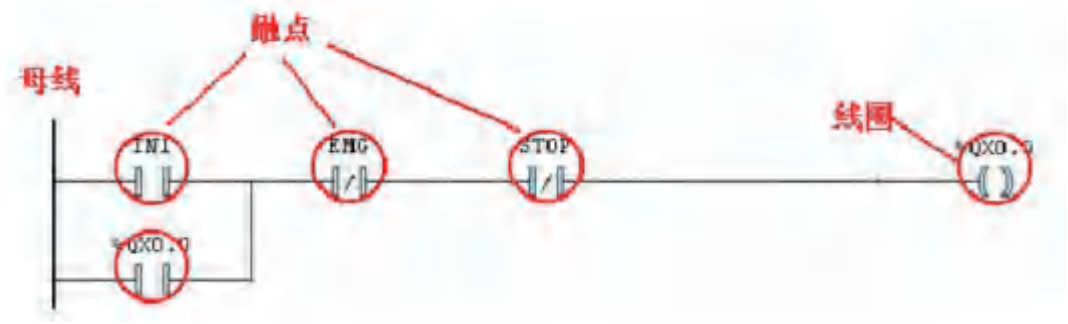
```
PROGRAM POU
VAR
    TON1: TON;
END_VAR

TON1(IN:=EN , PT:=T#500ms , Q=>T1_Out , ET=> );  // Call TON FB
```

3.3 梯形图（LD）

梯形图语言是PLC程序设计中最常用的语言。由于与电气设计人员所熟悉的传统继电器电路图类似，因此梯形图语言得到了广泛的应用。
梯形图包含了一系列的节，左右两边各有一个垂直的电流线（一般称为母线）限制其范围，在中间是由触点、线圈、连接线组成的电路图。每一个节的左边有一系列触点，这些触点根据布尔变量值的TRUE和FALSE来传递从左到右的“ON”和“OFF”的状态。每一个触点是一个布尔变量，如变量值为TRUE，通过连接线从左到右传递“ON”状态。否则传递"OFF"的状态。在节最右边的线圈，根据左边的状态获得“ON”或“OFF”的状态，并相应地赋给一个布尔变量TRUE或FALSE。

如下为一个梯形图程序的示例：



3.3.1 LD元素

常用LD元素如下表所示：

—| |—	触点
—|/|—	取反触点
并联触点	并联触点
并联取反触点	并联取反触点
—|P|—	上升沿触点
—|N|—	下降沿触点
—( )—	线圈
—(S)—	置位线圈
—(R)—	复位线圈

3.3.2 逻辑指令

插入触点

符号：—| |—

快捷键：F4

说明：插入一个常开触点。若选中一个已有触点后，再调用此命令时，插入的触点会出现在选中触点的左侧。

插入取反触点

符号：—|/|—

快捷键：SHIFT+F4

说明：使用此命令可以在程序中插入一个常闭触点。

插入串联右触点

符号：—| |—

快捷键：CTRL+D

说明：选中一个已有的触点，再调用此命令，可以在所选触点的右侧插入一个常开触点。

插入并联下触点
符号： 
快捷键：SHIFT+F5
说明：选中一个已有的触点，再调用此命令，可以在所选触点的下方插入一个常开触点。

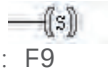
插入并联上触点
符号： 
快捷键：CTRL+P
说明：选中一个已有的触点，再调用此命令，可以在所选触点的上方插入一个常开触点。

插入取反并联下触点
符号： 
快捷键：ALT+F5
说明：选中一个已有的触点，再调用此命令，可以在所选触点的下方插入一个取反触点。

插入上升沿触点
符号： 
快捷键：CTRL+SHIFT+F4
说明：插入一个上升沿检测触点。选中一个已有的触点，调用此命令，新触点会出现在所选中触点的左侧。

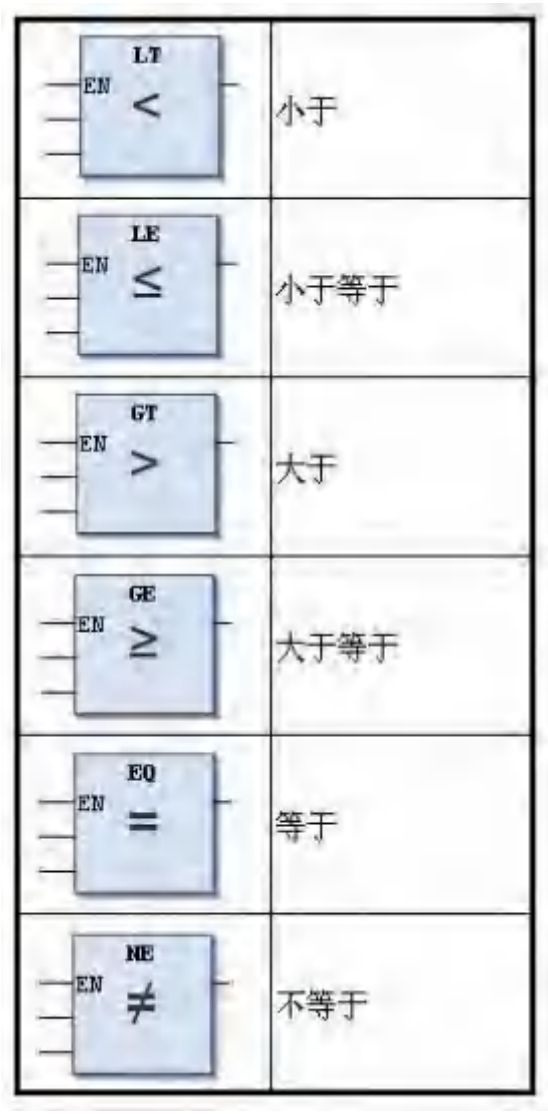
插入下降沿触点
符号： 
快捷键：CTRL+SHIFT+F5
说明：插入一个下降沿检测触点。选中一个已有的触点，调用此命令，新触点会出现在所选中触点的左侧。

插入线圈
符号： 
快捷键：CTRL+F9
说明：插入一个线圈。
如果选中触点和线圈之间的连接线，再调用此命令，则新线圈将添加在所有线圈的下面；如果选中的是线圈，那么新线圈将会添加在所选中线圈的上方。

插入置位线圈
符号： 
快捷键：F9
说明：插入一个置位线圈。
如果选中触点和线圈之间的连接线，再调用此命令，则新线圈将添加在所有线圈的下面；如果选中的是线圈，那么新线圈将会添加在所选中线圈的上方。

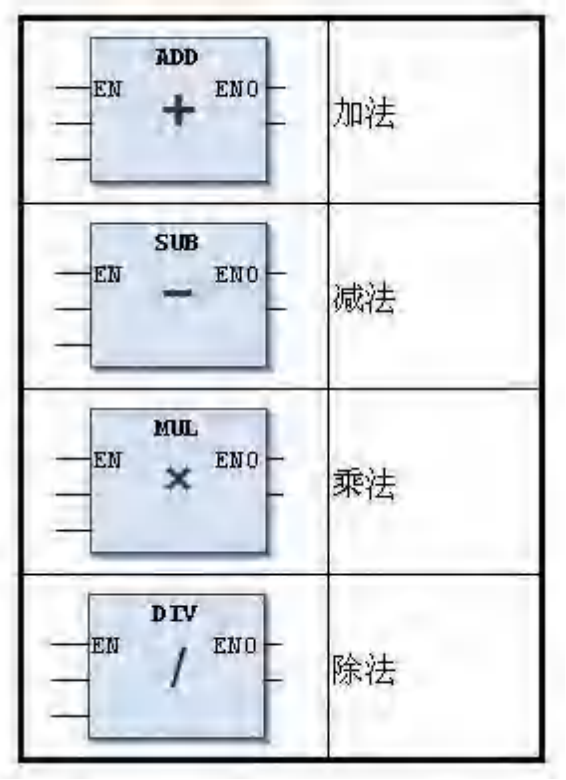
插入复位线圈
符号：
快捷键：SHIFT+F9
说明：插入一个复位线圈。
如果选中触点和线圈之间的连接线，再调用此命令，则新线圈将添加在所有线圈的下面；如果选中的是线圈，那么新线圈将会添加在所选中线圈的上方。

3.3.3 比较指令
常用的比较指令如下表所示：



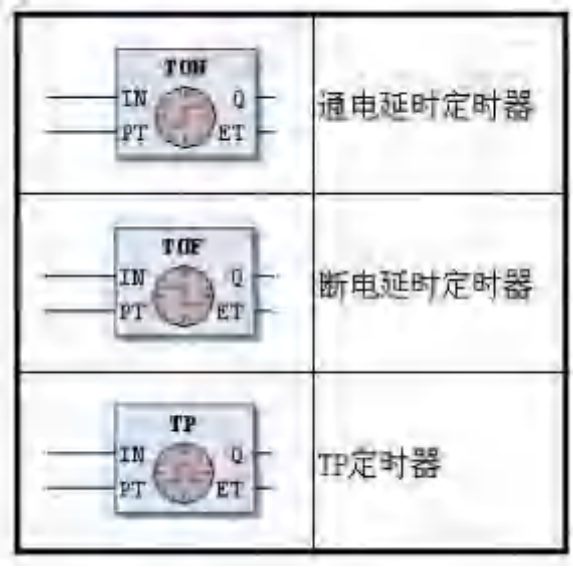
关于比较指令的详细说明，请参考手册第7.5章：《比较操作符》。

3.3.4 算术运算指令
常用的算术指令如下表所示：



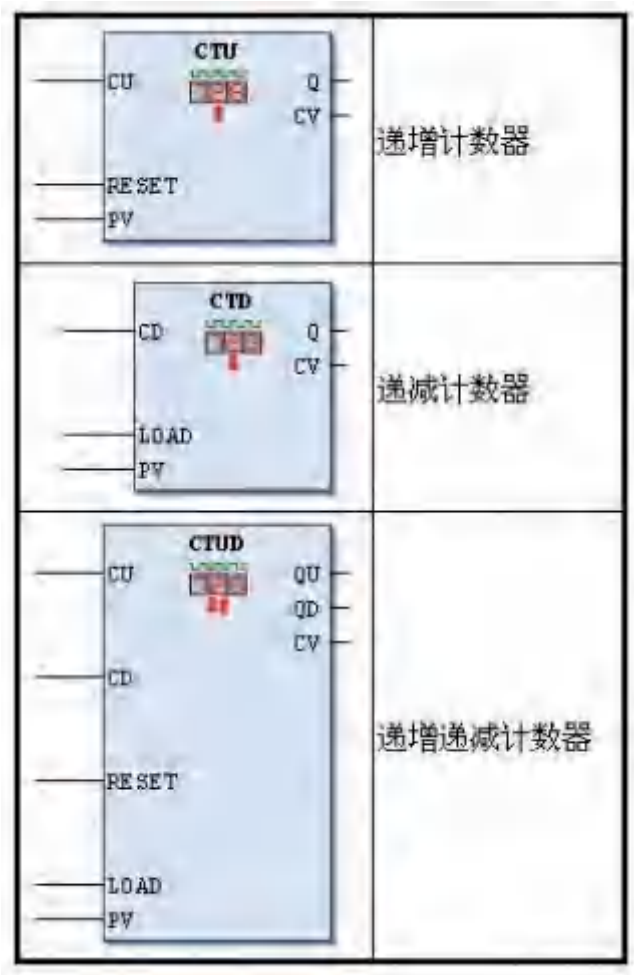
关于算术指令的详细说明，请参考手册第7.1章：《算术操作符》。

3.3.5 定时器指令
常用的定时器指令如下表所示：



关于定时器指令的详细说明，请参考手册第8.1.5章：《定时器》。

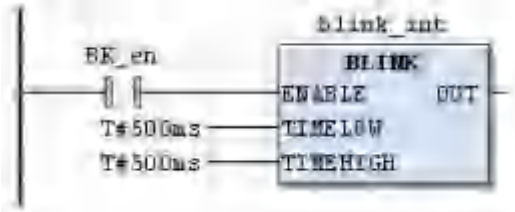
3.3.6 计数器指令
常用的计数器指令如下表所示：



关于计数器指令的详细说明，请参考手册第8.1.4章：《计数器》。

3.3.7 功能块

可以通过命令“插入运算块”或“插入空运算块”调用各功能块，例如：



关于在梯形图中调用各指令以及功能块的操作，可参考手册第4章：《编辑器》。

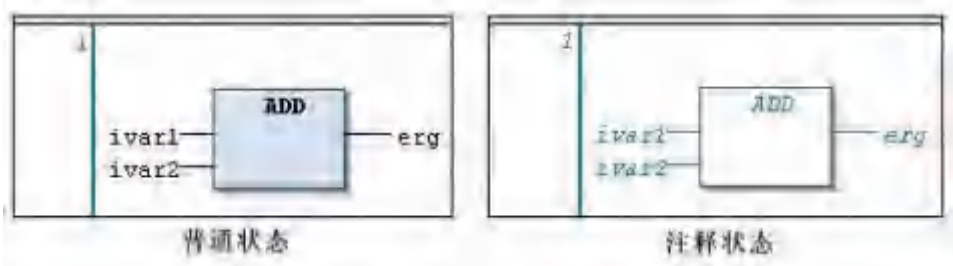
3.3.8 其他指令
常用的其他指令如下表所示：

	插入节
	在下方插入节
	节注释切换
	插入跳转
	插入标签
	插入返回

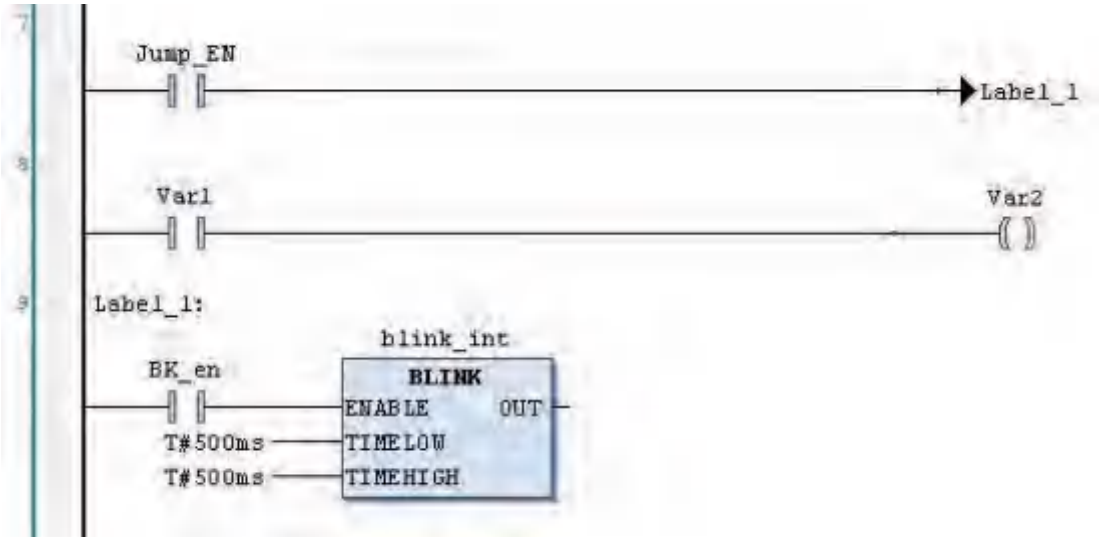
插入节
符号：
快捷键：CTRL+I
说明：使用此命令可以在LD编辑器中插入一个节。
如果光标处于一个节上，此时新插入的节将立即出现在光标所在节的上方。如果光标位于编辑器窗口中，但并不位于任何节上，此时插入节，新插入的节将会出现在编辑器最后一个节的后面。插入新节后，节的编号会自动更新。

在下方插入节
符号：
快捷键：CTRL+SHIFT+I
说明：使用此命令可以在LD编辑器中插入一个节。
如果光标正位于一个节上，此时新插入的节将立即出现在光标所在节的下方。如果光标位于编辑器窗口中，但并不位于任何节上，此时插入节，新插入的节将会出现在编辑器最后一个节的后面。插入新节后，节的编号会自动更新。

节注释切换
符号：
快捷键：CTRL+T
说明：使用此命令可以把节在普通状态和注释状态之间切换。
普通状态和注释状态如下所示：



插入跳转
符号：
快捷键：SHIFT+F11
说明：使用此命令将插入一个跳转运算块。跳转的目的地是另一个节，取决于跳转命令上的节标签。
示例如下所示。当变量Jump_EN为TRUE时，程序将跳过第8节，直接执行第9节。

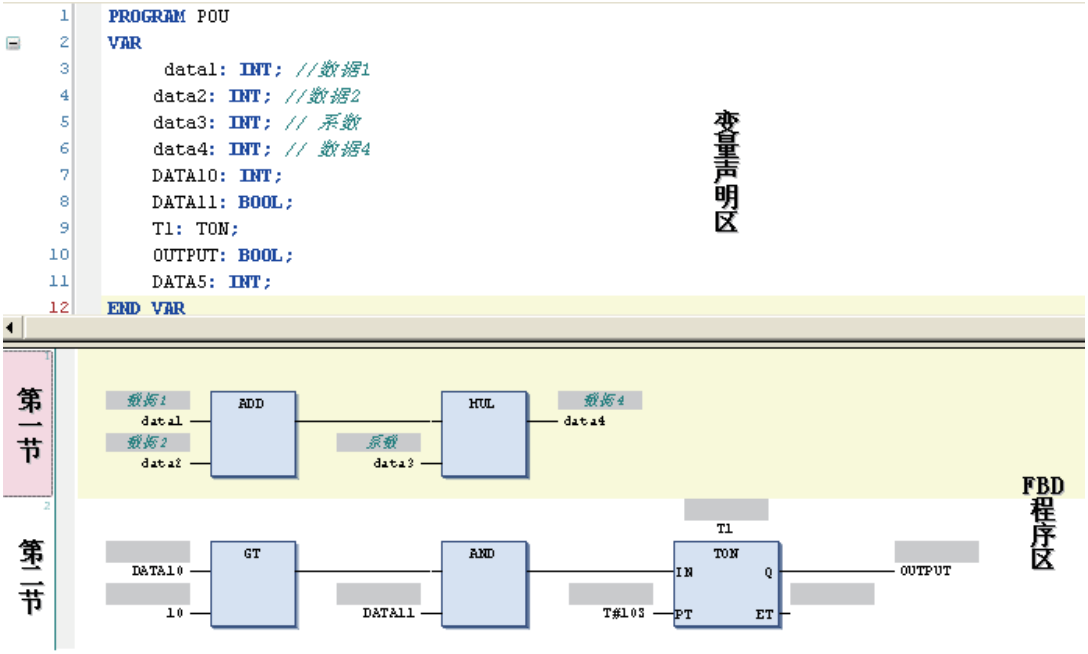


插入标签
符号：
快捷键：CTRL+F11
说明：使用此命令可以为当前选中的节添加标签。

插入返回
符号：
快捷键：CTRL+SHIFT+F11
说明：使用此命令可以插入一条返回命令(RETURN)。

3.4 功能块（FBD）

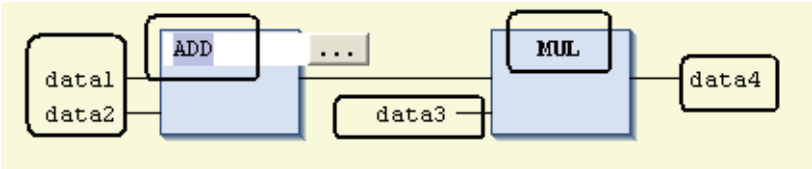
FBD是功能块图（Function Block Diagram）的简称。FBD是一种图形化的编程语言。FBD由一些列“节”组成，每个“节”有许多功能块组成。每节完成一段相对独立的运算，这些运算包括逻辑，算术，功能块，输入，输出，连线，跳转和返回等。
如图所示



3.4.1 FBD的光标位置

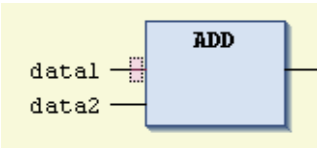
每个文本是一个可能的光标位置，选中的文本以蓝色背景表示并且当前可以被修改。通过点矩形框也能识别当前光标位置，在 FBD 中基本上由虚线矩形显示当前光标的位置，文本和运算块由蓝色或红色阴影显示。光标的位置决定了哪些元素在右键菜单是可插入的。下面是光标所有可能位置的例子：

1. 文本（光标位置1）

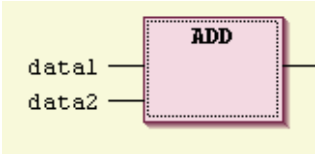


光标在文本处可以输入文本标签或内存地址，在跳出变量声明时可以对标签做注解，可以是中文注解。

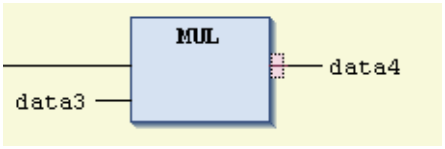
2.输入（光标位置2）：



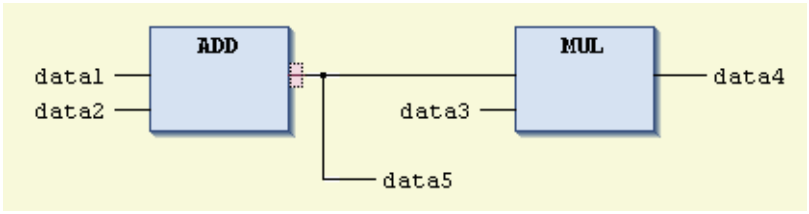
3.操作符、函数或功能块（光标位置3）：



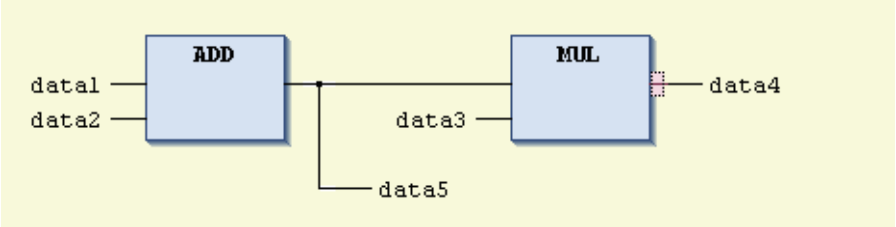
4. 输出（光标位置4，后面紧跟着赋值标签或跳转）：



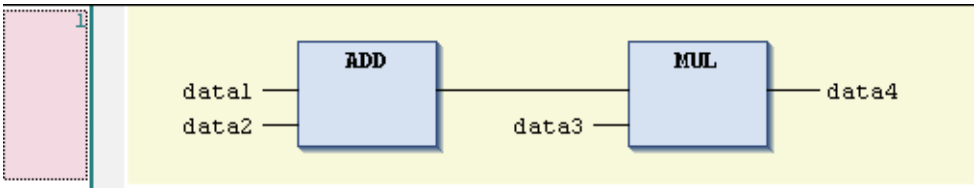
5. 赋值前面的交叉线（光标位置5）：



6.节的末尾输出端（光标位置6）



7. 在节的最右边位置或节的其它光标位置以外的地方。这将选择整个节：



通过单击鼠标或使用键盘可以在特定的位置放置光标，在任何时候，使用箭头键可以在选择的方向上跳到最近的光标位置。通过这种方式可以访问所有的光标位置包括文本区域。如果最近的光标位置被选中， 就可以使用<up>或 <down>方向键来选择先前的网络或随后的网络的最近光标位置。

3.4.2操作说明
在选中的光标处可以做插入和扩展操作
1. 赋值
插入赋值依赖于选中的位置（参照'FBD的光标位置'），在选中的输入端附近（光标位置2）、在选中的输出端（光标位置4）之后或在网络的末端（光标位置6）可以直接插入赋值。

2.跳转
符号： 
这个命令插入一个跳转。
插入依赖于选中的位置（参照'FBD的光标位置'），在选中的输入端（光标位置2）附近、在选中的输出
端（光标位置4）后或在网络的末端（光标位置6）可以直接插入跳转。

3.返回
符号： 
这个命令插入一个返回指令。
插入返回指令依赖于选中的位置（参照“FBD的光标位置”），在选中的输入端附近（光标位置2）、在选中
的输出端（光标位置4）之后或在网络的末端（光标位置6）可以直接插入返回指令。

4. 运算块
符号： 
用这些命令能插入运算符，功能，功能模块和程序。这三个功能块的具体使用参见后续章节

•在某些功能块的上部有一个可以写入文字的区域，在此区域中用户可以写入功能块的实例名。如果已定义实例的功能块被另一个功能块类型所替代，那么用户要重新定义功能块实例。

•运算块的所有未连接输入端都会出现“？？？”，用户必须用常量或变量代替这些问号。

•在插入运算块的地方，如果其右侧已经有分支，则这个分支将被连接到功能块的第一个输出变量。

•插入位置。对于大多数运算快来讲，可以直接插入到用户选中的位置：

•如果选中了一个输入变量（光标位置2），运算块会插入到该输入变量之前。运算块的第一个输入变量和第一个输出变量会在现有的分支中连接起来。

•如果选中了一个输出变量（光标位置4)），运算块会添加到该输出变量之后。运算块的第一个输入变量和第一个输出变量会在分支中连接起来。

•如果选中了一个运算块（光标位置3），它会被新的POU代替，新的运算块的连接和原来运算块的连接相同。如果原来运算块的输入多于新运算块的输入，那么未连接的分支将被删除，对于输出变量而言，情况与之相同。

•如果一个跳转或返回被选中（光标位置3），新运算块会插到跳转块或返回块之前。运算块的第一个输入变量和第一个输出变量会在现有的分支中连接起来。

•如果一个完整节或子节被选中（光标位置8 ），运算块会被插入到节或子节的最后面，它的输入将被连接起来。

5. 扩展输入

符号 ：

这个命令插入一个输入端运算。你必须选择运算符本身（光标位置3），在使用此命令后在功能块的输入测最下方会增加一个输入端。插入的输入端分配了文本“？？？”。

6. 取反

符号 ：

用这个命令可以对输入、输出、跳转或返回指令进行否定操作，否定的符号是在连接处的一个小圆圈。如果选中了一个输入（光标位置2）（查看'FBD的当前位置'），随后这个输入将被否定。如果选中了一个输出端（光标位置4），那么这个输出端将被否定。

如果一个跳转或返回被标记，那么跳转或返回将被否定。

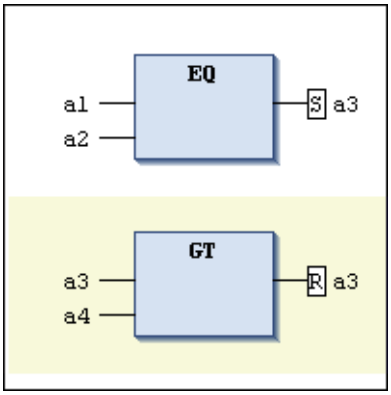
否定可以通过重新否定来取消。

7. 置位/复位'

符号 ：

用这个命令可以象设置或复位那样来定义输出，置位的输出用[S]表示，复位输出端用[R]表示。单击 输出断设置为置位，双击 输出设置为复位，同时输出端的显示为

三击 则输出断恢复为正常输出状态。



8.剪切，复制，粘贴和删除

在菜单项目“编辑”下可以用到“剪切”，“复制”“粘贴”和“删除”，也可以通过鼠标右键来完成。

如果选中了一个交叉线（光标位置5），那么位于交叉行下面的赋值、跳转或返回将会被剪切、删除或复制。如果选中了一个功能块（光标位置3），选中的对象自身将被剪切、删除或复制，同时也包括在输入端的所有独立的分支。

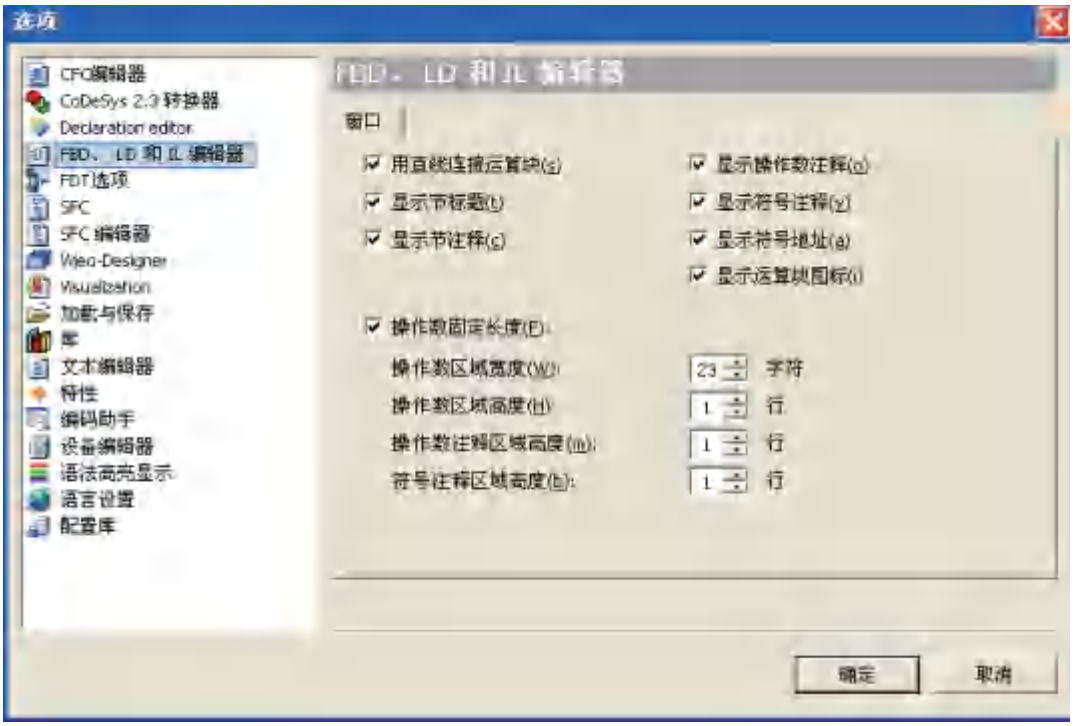
在复制或剪切之后，删除或剪切的部分位于剪贴板上，可以随意粘贴它。

首先选中粘贴点，有效的粘贴点包括输入和输出端。

3.4.3 对程序注释

在FBD编程时可以对节，操作数，符号做注释，可以是中文注释。

在“工具–选项”打开如下画面：



显示节的标题　选中此项后，一个节的标题可视后，节标题会显示在横线上。

显示节的注释　选中此项后，如果用户已经定义了节注释，则这些注释将显示在节左上角的横线下。

显示操作数注释 选中此项后，会在窗口的编辑区域中显示变量的注释，此处操作数的注释仅显示在使用该变量的地方。这与在声明变量时所加“符号注释”不同。

显示符号注释　选中此项后，在声明时加上注释的每个变量，其上方显示它们的符号注释。

注意：在添加符号注释后再为变量添加操作数注释，或是不加符号注释仅添加操作数注释，均是可以的。

显示符号地址　选中此项后，会在每一个变量上方显示它们的地址

操作数固定长度　如果未选中此项，系统默认将节的注释作为一行显示；如果选中此项，可以通过调整下列参数来规定注释显示区域的大小。

操作数注释区域的宽度[字符数]：规定最多用多少字符数来显示操作数名。

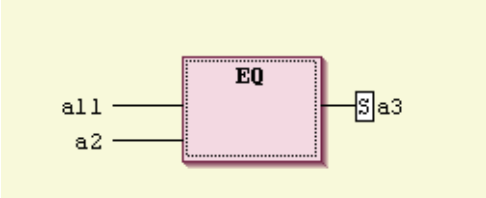
操作数注释区域的高度[行数]：规定最多可以用多少行来显示操作数名。

操作数注释显示区域的高度[行数]：规定最多可以用多少行来显示操作数的注释。

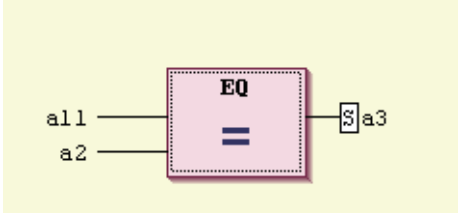
变量注释显示区域的高度：规定最多可以用多少行来显示变量注释。

显示运算块图标　选中此项后，功能块或函数提供了图标（位图），该图标将显示在FBD和LD编辑器的运算块中

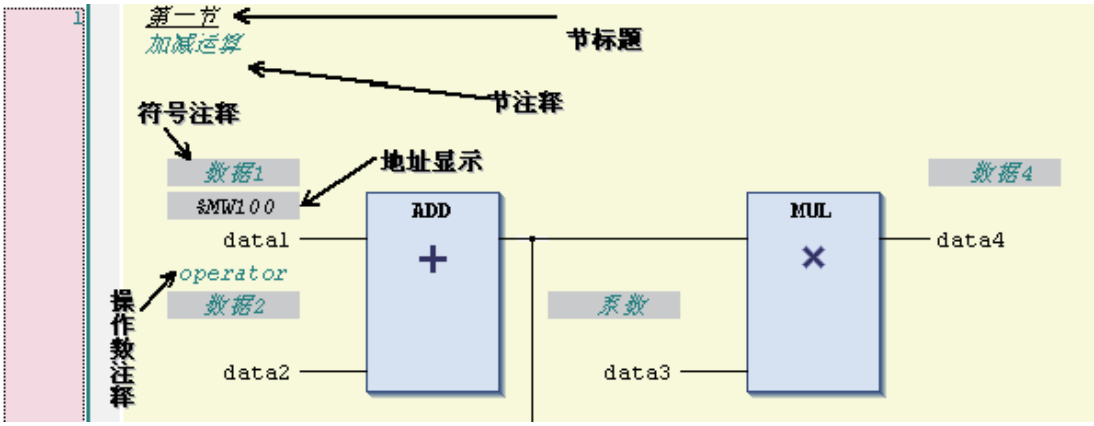
没有选用此功能的：



选用此功能后：



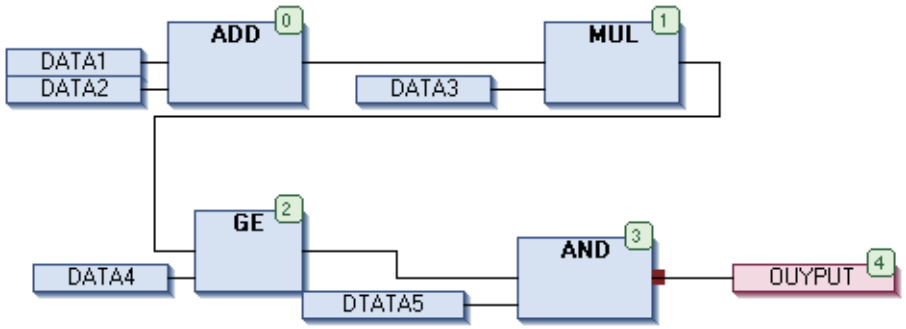
选择所有功能后界面显示如下：



3.5 连续功能图（CFC）

CFC是连续功能块图（Continuous Function Chart）的简称。CFC 是一种图形化的编程语言。CFC 基于FBD 语言，但没有“节”的限制，摆放元素更加灵活。。元素可以摆放在编程区任意位置。用鼠标拖拽在元素之间连线，当元素移动位置时，编辑器会自动调整连线长度。如果连接线因为缺乏空间不能画出，在输入和相关的输出之间出现一个红线，这个红线只有当空间充足时才转化为连接线。

```
PROGRAM POU_1
VAR
    DATA1: INT;
    DATA2: INT;
    DATA4: INT;
    DATA3: INT;
    DTATA5: BOOL;
    OUYPUT: BOOL;
```



可以通过缩放工具改变编辑窗口的尺寸：点击编辑窗口右下角的按钮，在打开的菜单中选择一个缩放倍数；还有另外一个方法，在打开的菜单中选择 ... 打开一个对话框，然后可以输入任意缩放倍数。

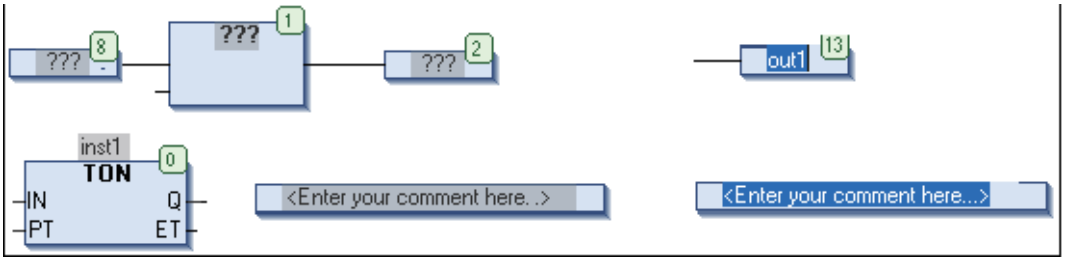
在CFC语言里元素的右上角的数字，显示了在线模式下CFC中元素的执行顺序。执行流程从编号为0的元素开始。移动元素时，它的编号仍保持不变。添加一个新元素时，按照拓扑序列（从左到右，从上到下），该元素将自动获得一个编号。

3.5.1CFC当前光标的位置

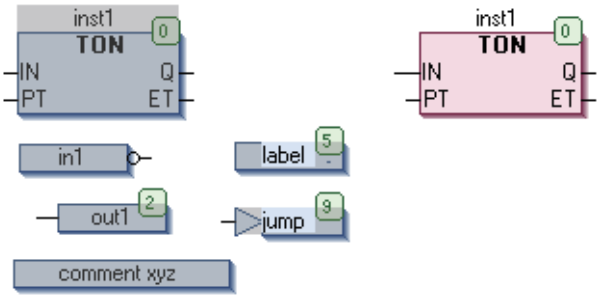
每个文本都是光标可能的位置，选中的文本渐变为蓝色并且可以被修改。

在其它的情况下当前鼠标的位置通过虚线矩形框来显示，下面是光标可能位置的例子：

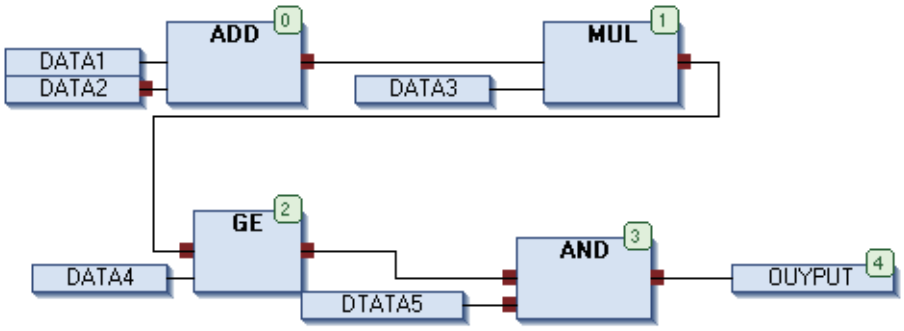
1. 当光标在文本时，文本的背景色为蓝色，且可以编辑。点击 按钮打开输入助手。当插入一个元素后，开始显示的是"???"，用于提示用户输入一个有效的标识符。然后，光标置于变量名或运算块参数名位置，并给出运算块参数或变量的类型提示。如果已经被定义，符号注释将显示在第二行。



2. 当光标在元素（运算块，输入，输出，跳转，标签，返回，注释）上时，该元素显示为红色，且可用鼠标移动。



3. 当光标指在元素的输入或输出连接线上时，连接点变红，可以对该连接线进行取反，复位或置位操作。



3.5.2 操作说明

CFC 的元素包括块、输入、输出、跳转、标记、返回和注释等。其中块分为操作符、函数、功能块和程序四种形式。

1.选中元素

在元素中继线处点击鼠左键，可以选中元素。

如果想同时选中几个元素，按住<Shift>键并选中单个元素。也可以用鼠标左键在编辑器中画矩形区域选中其中几个元素。“其它”/“全选”选中所有元素

2. 移动元素

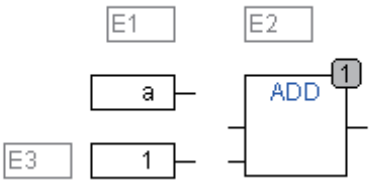
当光标在位置a 时，或按住<Shift>键同时选中

移动元素，到合适的位置后释放左键。如果释放位置处已有其它元素或超出编辑区，被移动元素会跳回原位置，移动失败

3.连线

一个元素的输入引脚只能连一个输出引脚（本元素的输出引脚或其它元素的输出引脚），而一个元素的输出引脚可以连几个输入引脚（本元素的输入引脚或其它元素的输入引脚）。在连线时，编辑器会检查双方的数据类型是否匹配，如果不匹配，光标会变为“禁止”样式，连线失败。若连接线为浅灰色，则表明元素之间有位置重叠

连线操作:



把鼠标放在E1 的输出引脚上，按下左键，拖拽到E2 的输入引脚上，释放左键
把鼠标放在E2 的输入引脚上，按下左键，拖拽到E1 的输出引脚上，释放左键

4.删除连线
如上图所示，有三种方式删除E1（a）和E2（ADD）之间的连线。选中E1的输出引脚，按下<Delete>键或“编辑”“删除”，如果E1 的输出引脚有几条线，则会同时删除。选中E2 的输入引脚，按下<Delete>键或“编辑”“删除”。
5.插入元素
CFC 的元素包括块、输入、输出、跳转、标记、返回和注释等具体参考下表：

	输入	???	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	???	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。
	运算块	???	运算块可用来表示操作符，函数，功能块和程序。选中运算块的‘???’文本框，修改为一个操作符名，函数名，功能块名或者程序名。通过输入助手可以选择输入一个有效的对象。 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。 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。
	跳转	???	跳转用来指示程序下一步执行到哪里，这个位置是由标签定义的（见下）。插入一个新标签后，要用标签名替代“???”。
	标签	???	标签标识程序跳转的位置（见上文“跳转”） 在在线模式下，标识POU结束的返回标签会自动插入.
	返回	RETURN	注意：在线模式下，return自动插入到编辑器第一列的最后那个元素之后。在单步调试中，在离开该POU之前，会自动跳转到该return。
	编排器	???	编排器用于结构体类型的运算块输入。编排器会显示结构体的所有成员，以方便编程人员使用它们。使用方法是：先增加一个编排器到编辑器中，修改“???”为要使用的结构体名字，然后连接编排器的输出引脚和运算块的输入引脚。
	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。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。

	注释	<Enter your comment here...>	用该元素可以为图表添加注释。选中文本，即可以输入注释。用户可以用<ctrl>+<enter>在注释中换行。
	输入引脚		有些运算块可以增加输入引脚。首先在工具箱中选中Input 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。.
	输出引脚		有些运算块可以增加输出引脚。首先在工具箱中选中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出引脚。

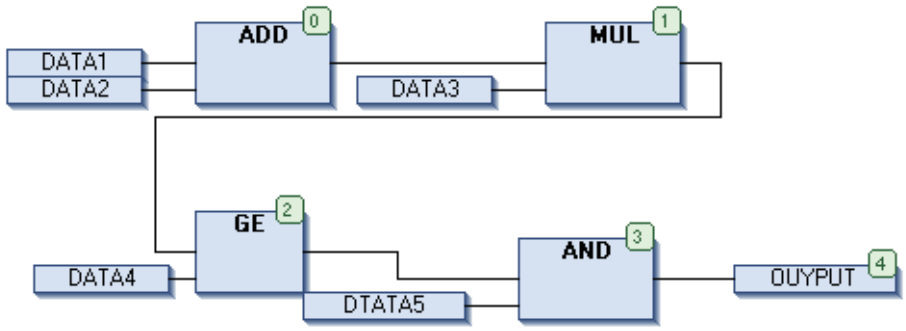
3.5.3 CFC元素的执行顺序

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。

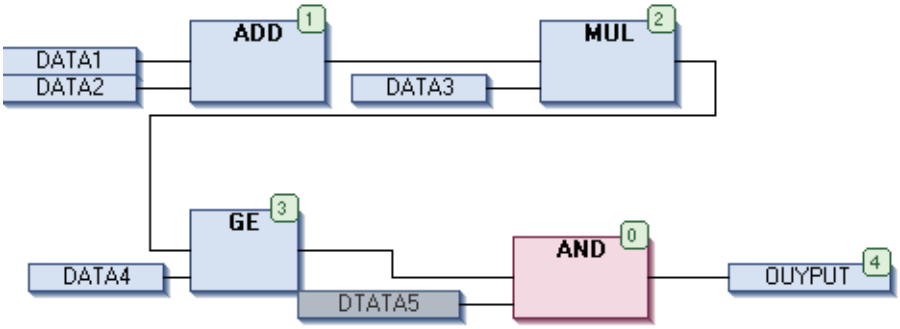
执行顺序包含的命令有：置首、置尾、向上移动、向下移动、设置执行顺序、按数据流排序、按拓扑排序。

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，下图说明此功能：

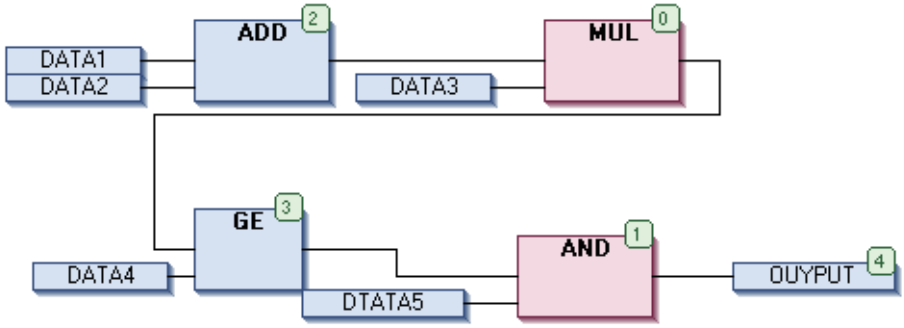


如选中图7中3号元素执行置首命令后执行顺序如下：



命令执行完成后，原3号元素标号变为了0号。其他元素的序号也做了调整，但依旧保持了原先的执行顺序。

如果选中图7中的1号和3号元素执行置首命令后执行顺序如下：



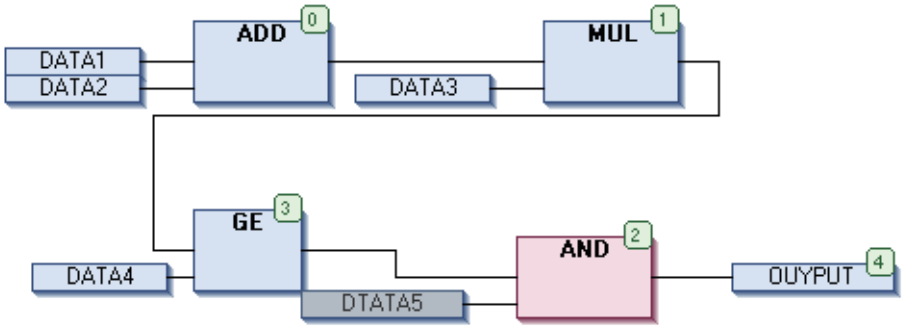
原1号和3号元素标号变成0号和1号，但这两个元素依旧保留原先内部执行顺序。其他未选中的元素也做了标号调整，但也保留这原先的内部执行顺序。

2.置尾

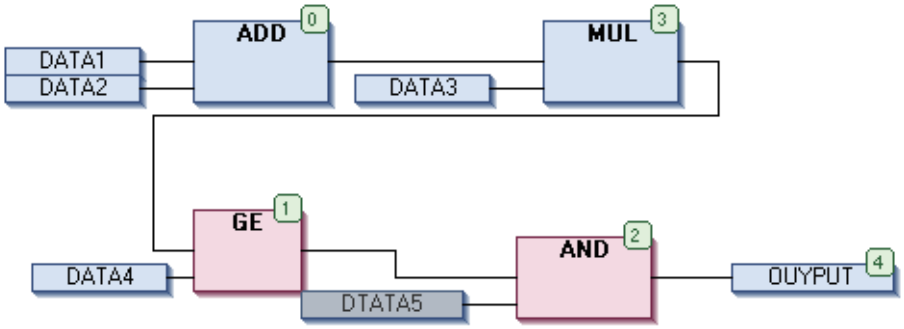
把所有选中元素移到执行顺序的末端。选中元素的内部顺序保持不变；未选中元素的内部顺序也保持不变。具体操作可以参照上述“置首”功能

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。如下：



如果把图7中2号和3好元素都选中执行“向上移动”命令后，其结果是原2号、3号元素变成1号和2号元素，原1号元素编程3号元素。其余不变。如图示：



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.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，编辑器内部做了如下一些操作：首先按照拓扑对所有元素进行排序；然后创建一个新的执行顺序链表；找到那些输入值已知，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：一个算法块执行后，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。如图示：

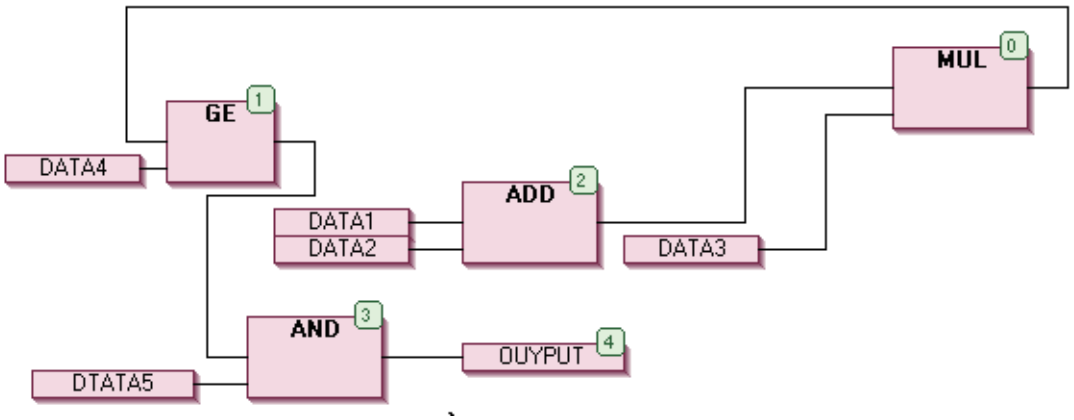
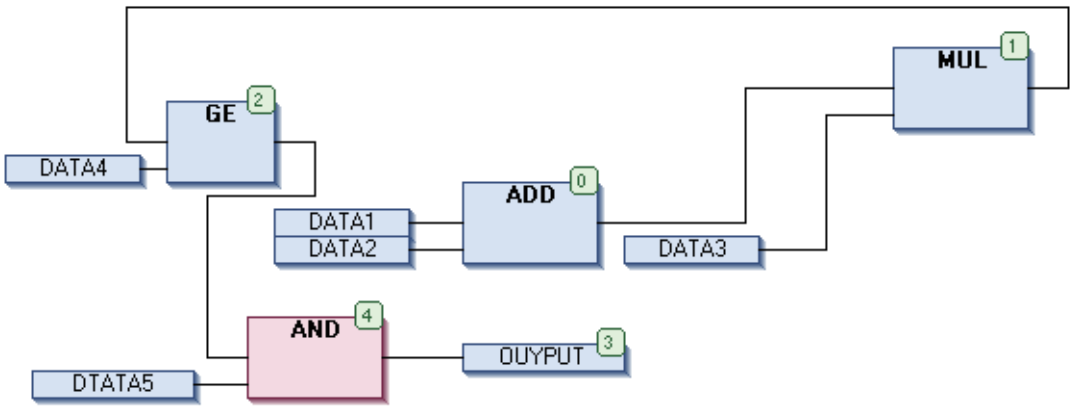


图12是一个拓扑结构顺序，选中全部元素后执行“按数据流排序”后结果如下：



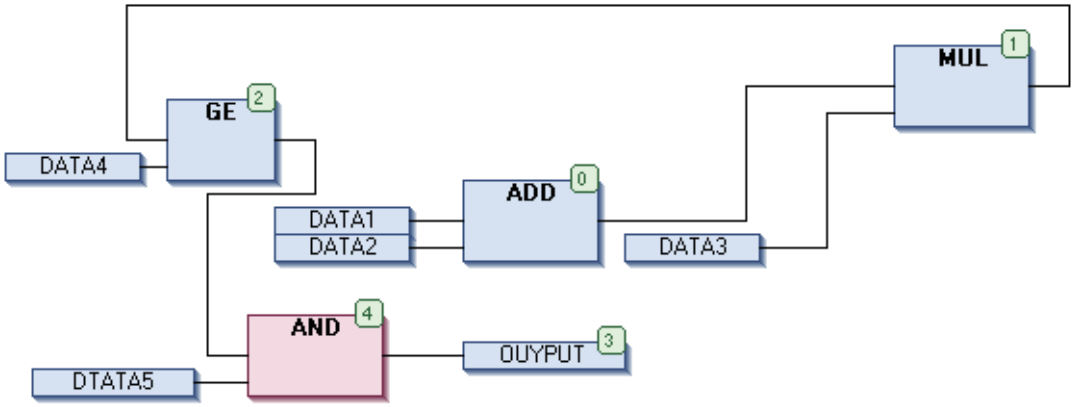
元素的编号按照数据流的流向来编排的。

6、按拓扑排序

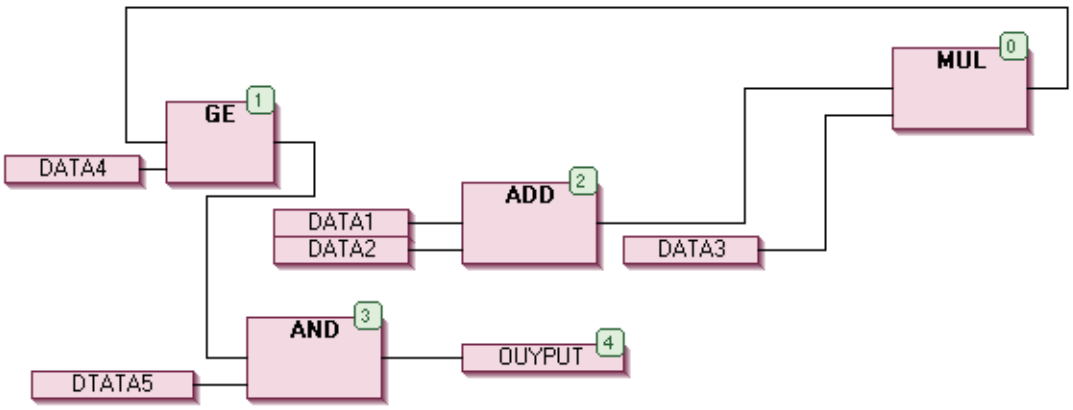
拓扑排序表示各个元素按照拓扑顺序执行，而不是按照元素数据流决定执行顺序。

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。

举例说明,下图打乱元素标号的程序：



选中全部元素执行“按拓扑排序”命令后结果如图示：

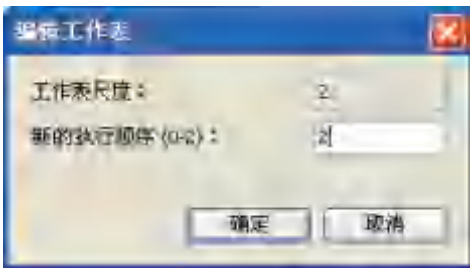


拓扑排列的顺序就是：从左到右，从上到下的顺序执行；左边的元素的执行顺序编号小于右边的，上边的小于下边的。

7、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。

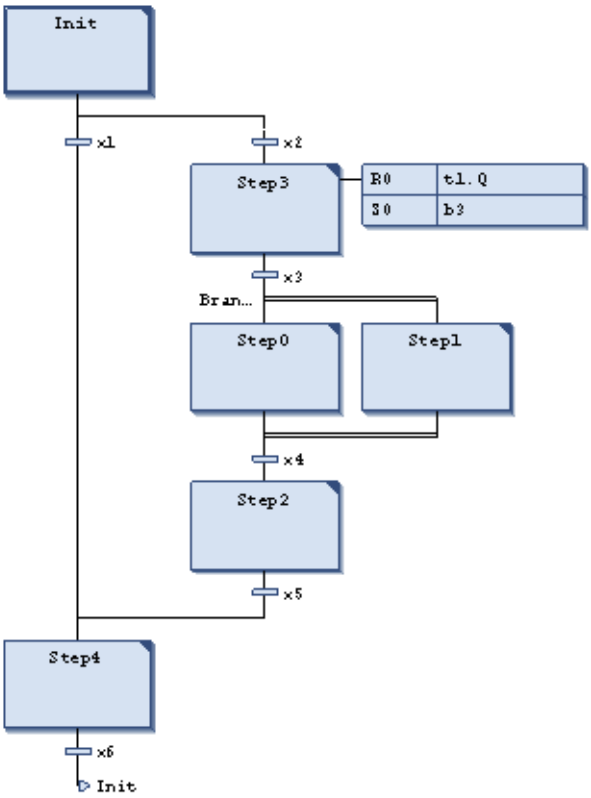
执行“设置执行顺序”命令后，会打开设置执行顺序”对话框。在当前执行次序区域显示当前单元编号，用户可以在新执行次序中输入需要单元编号。括弧中值为可选值



3.6 顺序流程图（SFC）

SFC 是顺序功能图（Sequential Function Chart）的简称，是一种图形化的编程语言，

用来描述程序中不同动作的时间顺序。可以在一个程序内按照时间顺序对动作进行编辑描述。这些动作可以作为独立的编程对象，用任意编程语言进行编写。SFC 由一系列的步和转移组成。



3.6.1 基本概念

1.步

用顺序功能图编写的程序组织单元包含了一系列的步，这些步之间是通过定向连接（转换条件）实现的。

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，那么在步就会编程蓝色的框。。

2.动作

动作是使用其它语言实现的一系列指令，可以是用IL 或ST 语言实现的指令语句，也

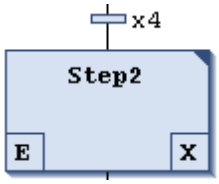
可以是用LD、FBD、CFC 或SFC 实现。用鼠标双击动作所属的步，进行编辑.

各步的动作编辑画面类似与POU的界面，各种语言的编辑界面均可用。所不同的是动作编辑画面里没有变量申明区，所有的局部变量都在总的SFC界面里，如下图所示，所有步的局部变量都在POU_2(PRG)

1. 进入和退出动作

可以额外的为一个步添加一个进入和退出的动作，在一个步激活后，一个进入动作只能执行一次。退出动作只在步失效之前执行一次。

进入动作用左下角一个“E”来表示，退出动作用右下角的“X”表示。



4.转换/转换条件

在步和步之间有所谓的转换。

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。

5.激活步

在调用顺序功能图的POU后，初始化步的动作（被一个双边线包围）将首先执行。动作正在执行的步称为激活步。在线模式下，活动步以蓝色显示

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。

6.限定符

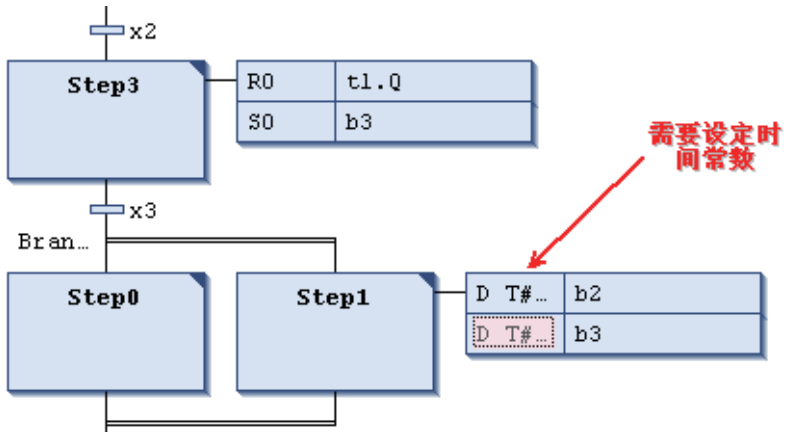
为了关联动作和步，用到下面的限定词。

限定词L、 D、 SD、 DS 和SL 需要一个时间 常量格式的时间值。

时间格式为T#（数值）（单位）。如5秒表示为T#5S。

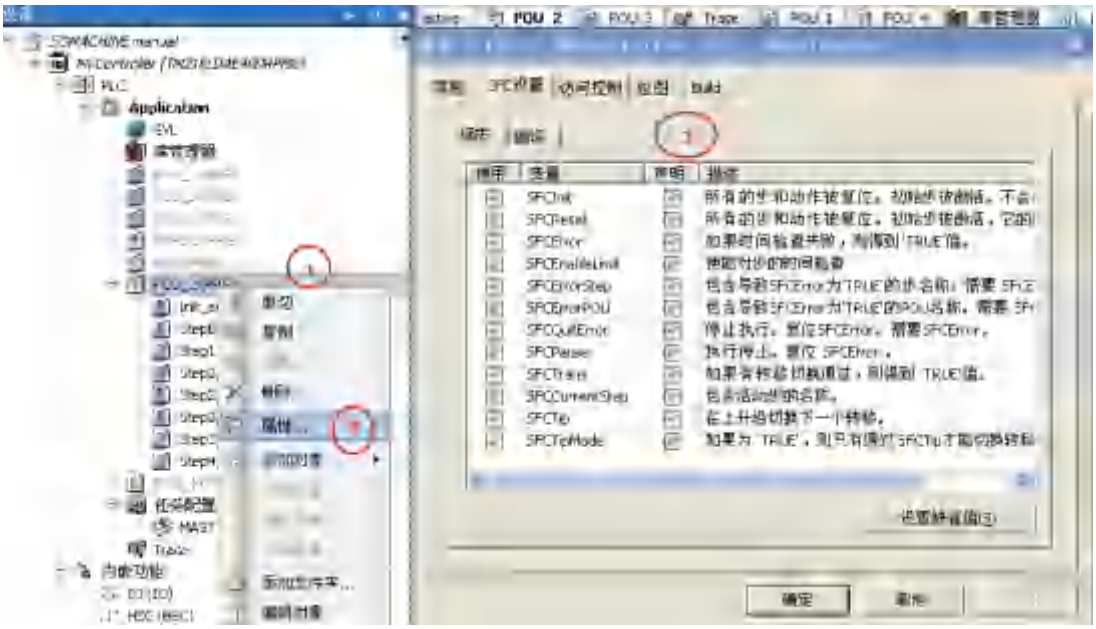
限定符	名称	说明	备注
N	非存储	普通型：动作随步活动，当步开始激活时，动作开始执行；当步退出时，动作停止执行。	
S	设置存储	动作还将继续执行。一直等到有 R限定符给它复位之后，动作才将停止执行。在动作没有置位型：当步开始激活时，动作开始执行；当步退出时，被 R停止之前，任何限定符也不能使得动作停止（包括 P限定符）。	
R	重置	复位型：当步开始激活时，动作停止执行。	
L	时限	限时结束型：动作在一定时间被激活。旳长时间是步活动时间。当步开始激活时，动作开始执行；同时开始计时，设定值时，步的转移条件已经满足，此时，动作也将当时间到达设定值后，动作将停止执行。如果时间还没有到达停止执行。	要加时间常数（设定值）
D	延时	延时开始型：当步开始激活时，开始计时，当时间到达设定值后，动作将开始执行；当步退出时，动作停止执行。如果时间还没有到达设定值时，步的转移条件已经满足，在这种情况下，动作将不可能被执行。	要加时间常数（设定值）
P	脉冲	脉冲型：当步开始激活时，动作开始执行，并且动作只被执行一次，之后动作将停止执行。	
SD	存储和延时	延时绝对开始置位型：当间到达设定值后，动作将开始执行；当步退出时，动作步开始激活时，开始计时，当时还经满足，在这种情况下，计时还在进行，一直到时间达到设定值后，动作将被执行，此时不将执行，一直等到有 R限定符给它复位之后，动作才将停止执行。在动作没有被 R停止之前，任何限定符也不能使得动作停止（包括 P限定符）。如果时间还没有到达设定值时，步的转移条件已管当前步还是不是原来的步了。	要加时间常数（设定值）
DS	延时和存储	延时开始置位型：当步开始激活时，开始计时，当时间到达设定值后，动作将开始执行；当步退出时，动作还将执 P限定符）。如果时间还没有到达设定值时，步的转移条件已经满足，在这种情况下，动作将不可能被行，一直等到有 R限定符给它复位之后，动作才将停止执行。在动作没有被 R停止之前，任何限定符也不能使得动作停止（包括执行。	要加时间常数（设定值）
SL	时限存储和	活动的。当步开始激活时，动作开始执行；同时开始计时，当时间到达设定值后，动作将停止执行。如果时间还没有到达设定值时，件已继续执行，直到行。限制符 L、D、SD、DS和 SL需要 TIME常量格式的时间值。绝对限时结束型：在一定时间内动作是步的转移条经满足，此时，动作还将当时间到达设定值后动作才会停止执	要加时间常数（设定值）

如图所示：



SFC隐形变量

在SFC编程语言里有些隐形变量可以使用。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。右击使用SFC语言的POU属性，弹出属性对话框，点击SFC设置选项，将需要使用变量前打勾。如图所示：



各个变量信息如下：

SFCInit: BOOL;	如果这个变量为TRUE，其对应的SFC程序将被设定回退到"初始步"。所有的步和动作以及其他的SFC标志都将被复位（初始化）。初始步将保持活动，但是只要这个变量为TRUE，其就不会被执行。SFCInit必须被设定回到FALSE，才能恢复正常运行。
SFCReset: BOOL;	这个变量的行为类似于SFCInit。但是最后的地方是不一样的。在初始步的初始化完成之后有进行了更多的处理工作。因此，在这种情况下，例如将SFCReset信号标志复位为FALSE的操作，就可以在初始步中来实现。
SFCError: BOOL;	只要在SFC的任何一步中发生超时，该变量将会变为TRUE。前提是：SFCEnableLimit必须为TRUE。注意，在SFCError被复位前，任何别的超时情况都不会被记录。如果需要使用其它的时间控制标志（SFCErrorStep, SFCErrorPOU, SFCQuitError），SFCError必须被定义。
SFCEnableLimit: BOOL;	该变量用来在步中，通过SFCError实现对时间控制进行激活（TRUE）或禁止（FALSE）的操作。这就意味着，如果这个变量被声明并被激活（SFC设定），则为使SFCError可以起作用，它必须被设定为TRUE。否则，任何步的超时都将不能被记录。这种用法在系统启动或手动操作时是很合理的。如果这个变量没有被定义，SFCError将会自动开始工作。当然，其前提是SFCError必须是已经被定义了！
SFCErrorStep: STRING;	这个变量存储发生超时的步的名称，该步的超时由SFCError.timeout记录。前提是SFCError必须被定义！
SFCErrorPOU: STRING;	该变量存储一个发生超的SFC POU的名称. 前提是SFCError必须被定义！
SFCQuitError: BOOL;	只要这个变量为TRUE，SFC图的执行就会停止，并且SFCError变量会被置为FALSE。当该变量一被复位为FALSE，所有当前活动步的时间状态就将被复位。前提是SFCError必须被定义！
SFCPause: BOOL;	只要这个变量为TRUE，SFC图的执行就会停止。
SFCTrans: BOOL;	当一个转移被激活时，该变量就会为TRUE。
SFCCurrentStep: STRING;	这个变量存储当前活动步的名称，其独立于时间监视。对于同时执行的序列来说，其将会记录处于右侧的步名称。
SFCTip, SFCTipMode: BOOL;	这个变量允许在当前图内使用缓动模式。当这个模式通过SFCTipMode=TRUE被打开时，只有当SFCTip=TRUE时（上升沿），才可以跳转到下一步。只要SFCTipMode被设置为FALSE，就可以通过转移进行跳转。

还有些针对每个步的隐形变量，步的隐形变量是结构体变量，在使用时需要在变量申明区里构建一个这对步的结构体变量。

在监控状态时可以看到STEP1里相关内容。

step1.x 表示当前活动状态

step1._x: 表示下一个循环的活动状态

如果Step1.x = TRUE, 该步将会在本周期被执行

如果 Step1._x = TRUE 并且 Step1.x = FALSE, 该步将在下一个周期中执行。即在一个循环的开始时刻，将Step1._x 的值拷贝到Step1.x 。

Step1.t 步激活到当前时刻所花费的时间

第四章 编辑器

4.1关于编辑器

SoMachine提供给用户一个基于硬件和软件的集成编辑平台

4.1.1控制器设备编辑器：

1.用户主界面：

1.1取决于自定义设置，这个设置存储在一个特殊的文件中，并定义菜单、工具栏和键盘的使用。当前的自定义设置可在工具菜单中自定义对话框中查看并修改。

1.2用户界面外观还取决于"视图"和窗口的排列，用户随时可以移动“视图”和窗口、固定/浮动视图、调整大小或关闭窗口。

用户界面主要由下图所示：

2.设备连接操作界面：

2.1 通讯设置: 用于配置与控制器之间的通信连接。

2.2 应用:显示控制器上当前正在运行的应用程序，并可从控制器中删除应用程序。

2.3 PLC设置：包括以下项目的配置：

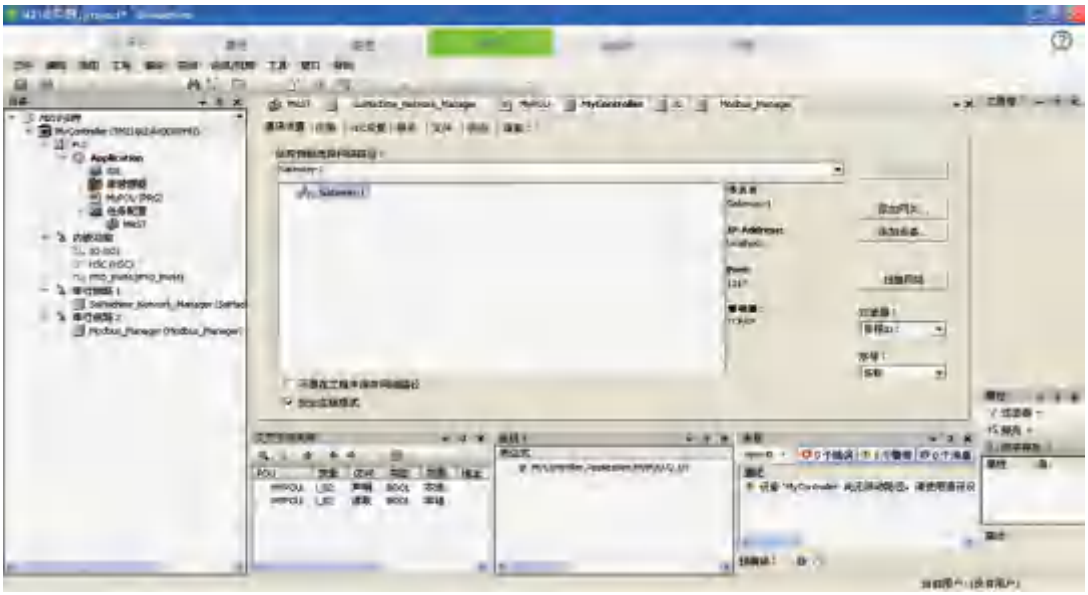
- I/O处理应用：应用程序名称
- PLC设置：1.停止时更新I/O
  2.停止模式时的输出动作（保持当前值&输出为缺省值）
  3.更新所有设备中的所有变量
- 总线周期选项：MAST任务等

2.4 服务：

包括以下项目的配置：
PLC时间（读取）
当地日期时间（写入&与当地的日期/时间同步）

2.5 文件（PC与控制器之间的文件管理）

用户界面如图所示：



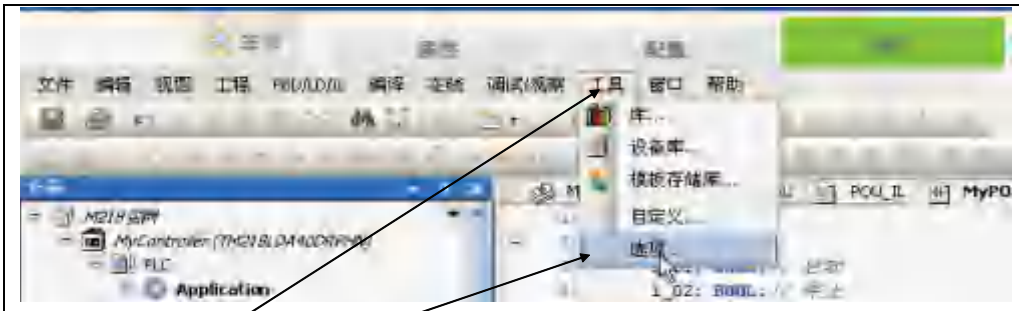
4.1.2软件编辑器：

系统提供6种编程语言的编辑器，分别是以下几种：
1.指令表（IL）编辑器
2.结构化文本(ST)编辑器
3.功能块图形(FBD)编辑器
4.梯形图(LD)编辑器
5.连续功能图(CFC)编辑器
6.顺序流程图(SFC)编辑器

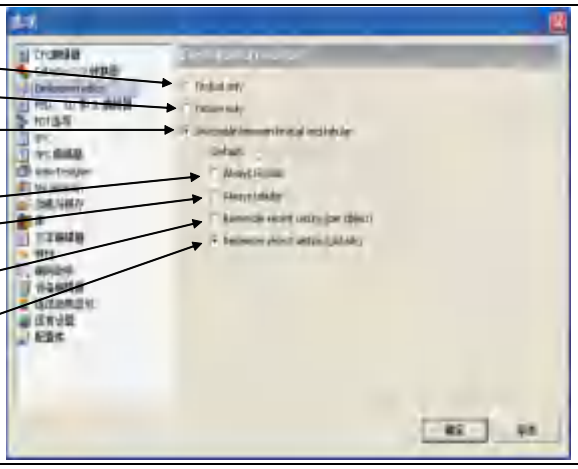
4.2声明编辑器

声明编辑器：
1.文本声明编辑器：是一个用于变量声明的文本编辑器
2.表格声明编辑器：编辑器的表格视图为普通的变量声明的。

声明编辑器操作如图所示：



SoMachine 中点击"**工具**"菜单的"**选项**"进入声明对话框：



1.只有文本

2.只有表格

3.在文本和表格之间切换:如果激活该设置，声明编辑器窗口的两个按钮都可用，可以在声明部分切换文本视图和表格视图。此时可以选择以下设置，以下设置定义了在打开对象进行编辑时，声明编辑器首次缺省显示的是哪种形式：

☒3.1一直是文本

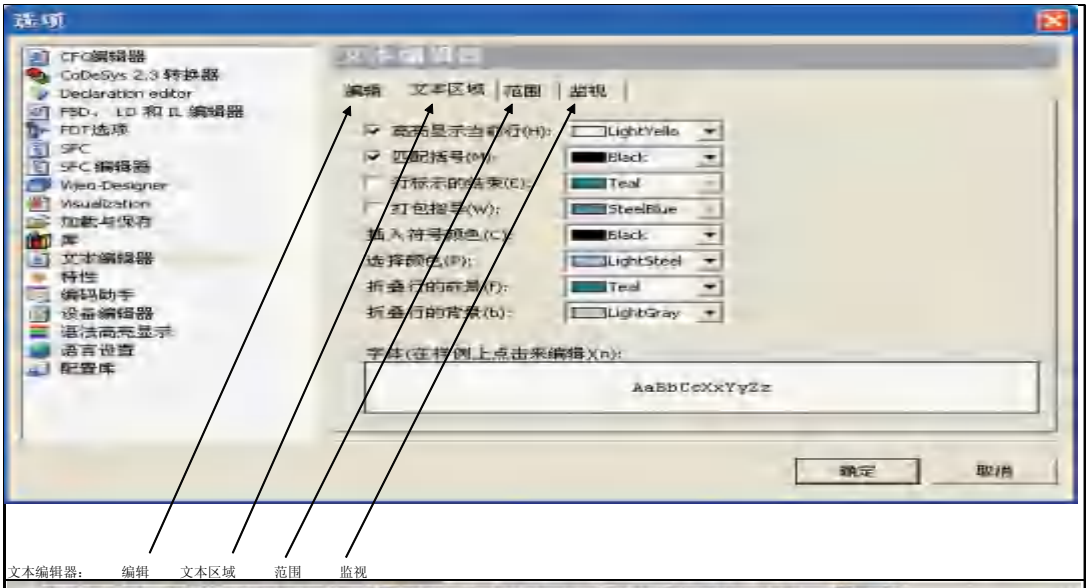
☒3.2一直是表格

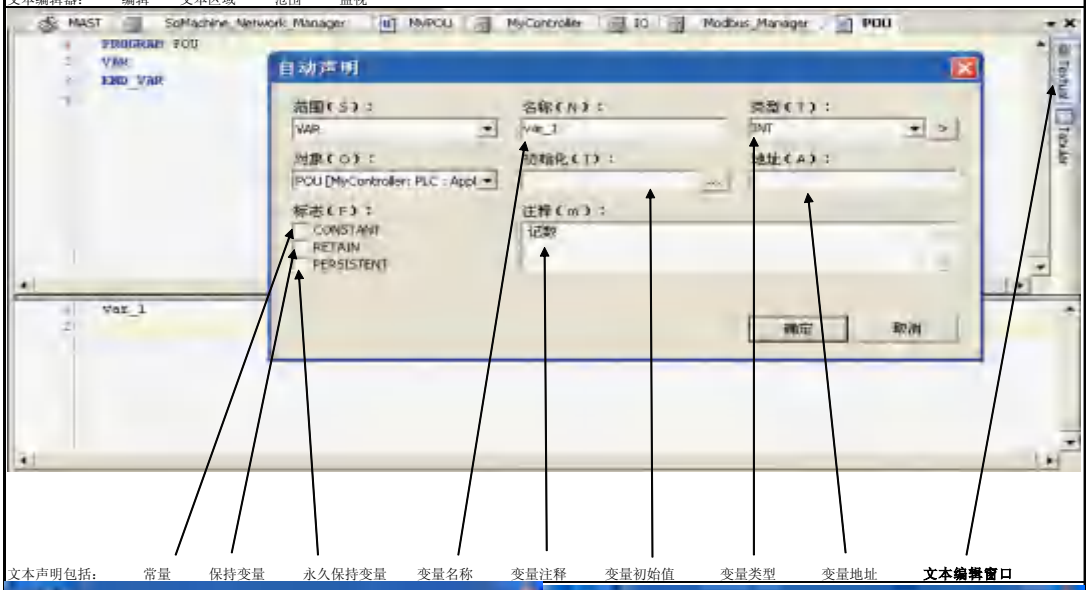
☒3.3记住最近的设置(每个对象):重新打开一个对象时，声明编辑器将以和上次编辑这个对象时相同的格式打开。

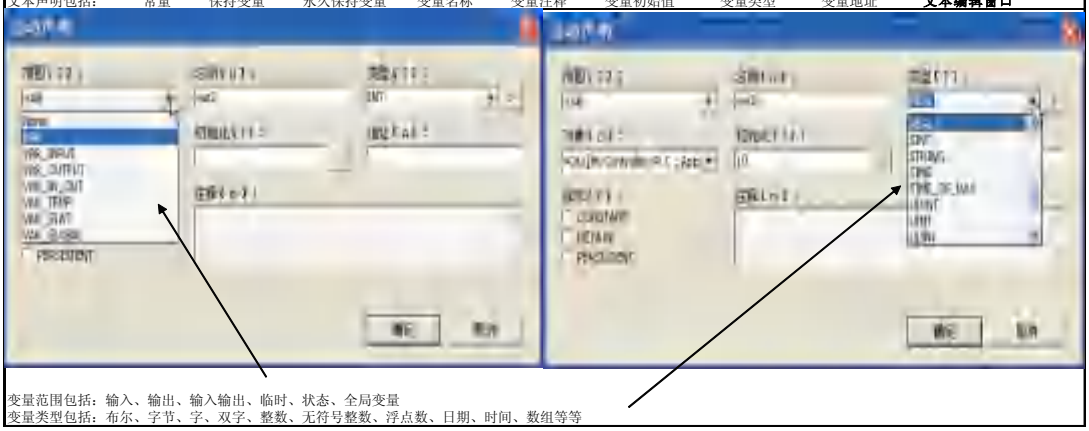
3.4记住最近设置(全局): 当重新打开一个对象时，编辑器将使用最近打开任意对象时的格式打开。

4.2.1 文本声明编辑器

操作和外观由各自当前文本编辑器的设置来确定，这些设置在选项对话框中定义。在对话框中可以对高亮显示颜色、行号、跳格、缩进等进行缺省设置的定义。如果安装了相应的插件，则具有常规Windows的功能，甚至具有智能鼠标的功能。

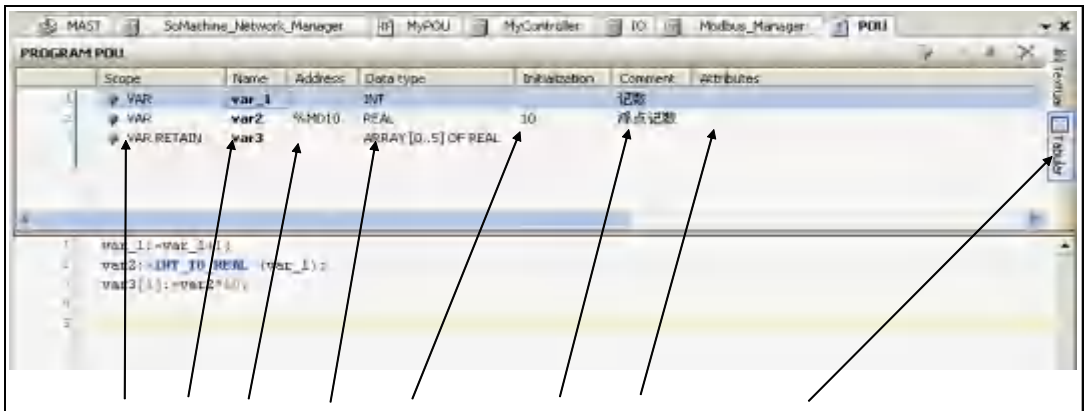






4.2.2 表格声明编辑器

编辑器的表格视图为普通的变量声明的定义提供需要的列：范围、名称、地址、数据类型、初始化、注释和（注解）属性。特殊声明作为编号行插入。



声明包括： 范围、 名称、 地址、 数据类型、 初始化、 注释和（注解） 属性 **表格编辑窗口**

在“声明”下：可以编辑POU对象的类型（从选择列表中选择）和名称。可以输入注释（**通过Ctrl+Enter**换行），属性按钮为指定的注解说明和属性打开属性对话框。

在现存声明上添加一个新行，先选中该行，然后从工具栏或右键菜单中使用插入命令。要在表的末尾添加一个新的声明，先在现存最近的声明行处点击，同样使用插入命令。

在默认下，最新插入的声明首先使用范围"VAR"和最近使用的数据类型。强制变量名称的输入栏将被自动打开，在这里输入一个有效的标识符，并用<Enter>键或在视图的其它地方点击来结束。

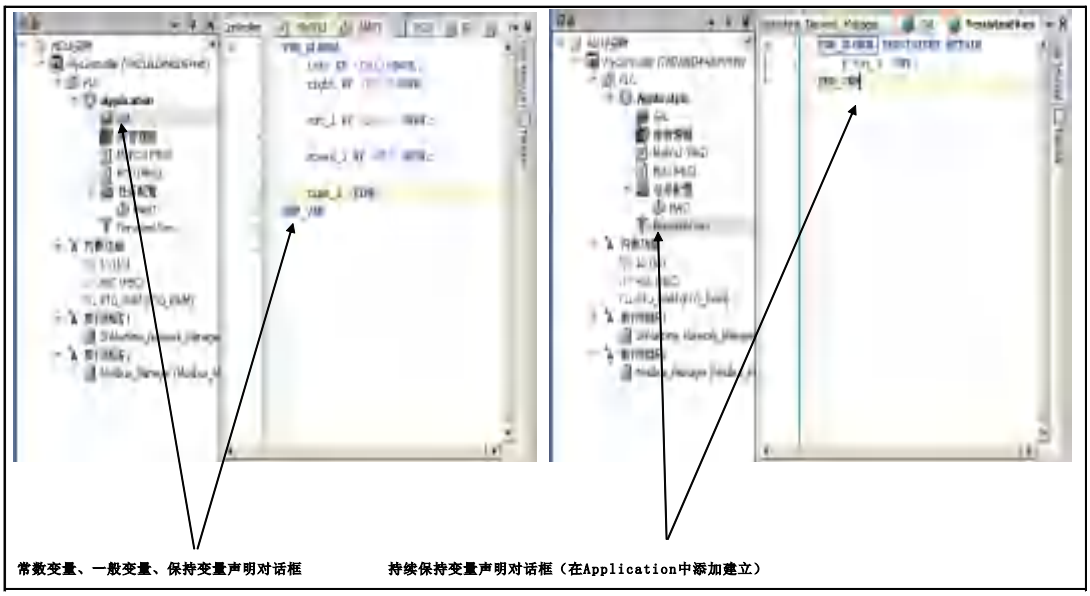
在每个表格单元上双击打开并输入值。

要编辑范围，双击打开列表，从中选择期望范围和范围属性关键字（标志）。

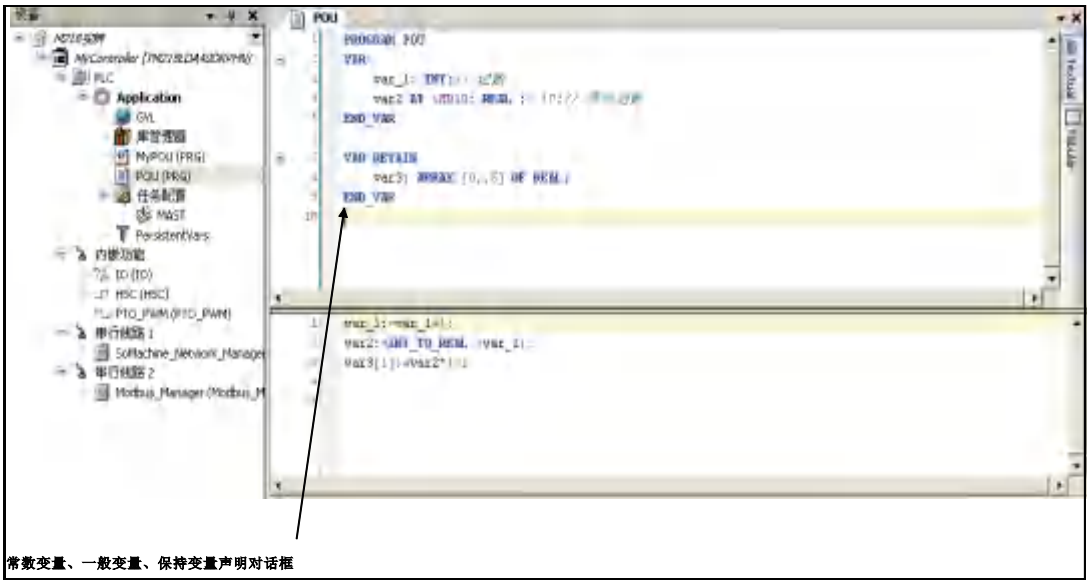
数据类型可以被直接键入或通过选择按钮使用辅助输入或数组向导。

4.2.3变量声明
1.离线变量声明 ：

变量有效性：
1.1全局变量：变量应用于全局程序中。

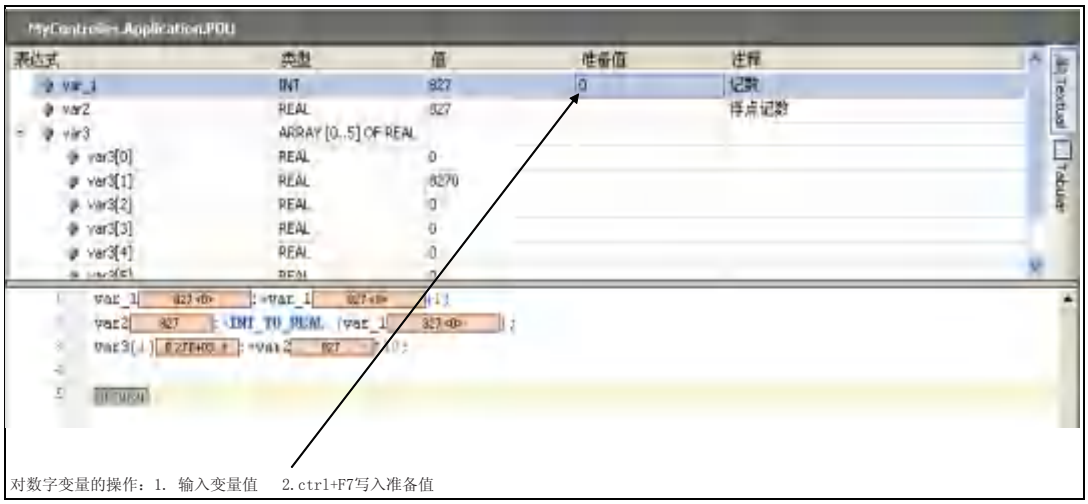


1.2 局部变量：变量只应用于单一程序中。



2. 在线模式下的声明编辑器

声明编辑器在线模式的表格显示与监视窗口类似。
每个监视表达式的表格，显示数据类型和当前值，还显示当前预设定的强制值或写入值。



4.3 文本编辑器

- 指令表（IL）编辑器
- 结构化文本(ST)编辑器

4.3.1 指令表（IL）编辑器

指令表-IL:

指令表是遵循IEC 61131-3标准的编程语言。

此语言是基于累加器的编程语言。支持所有IEC 61131-3标准的操作符、多输入/多输出、取反、 注释、输出的置位/复位和无条件/条件跳转。

每条指令首先使用 LD操作符将值装载到累加器中。然后累加器将被执行。执行的结果存储在累加器中，使用ST指令可以将该结果赋给其它变量。

为了实现条件执行或循环，IL 支持如EQ, GT, LT, GE, LE, NE比较操作符和跳转指令。跳转指令可以分为无条件(JMP)或条件的(JMPC / JMPCN)跳转。条件跳转时需检查累加器的值是真或假。

语法:

指令表由一系列的指令组成。

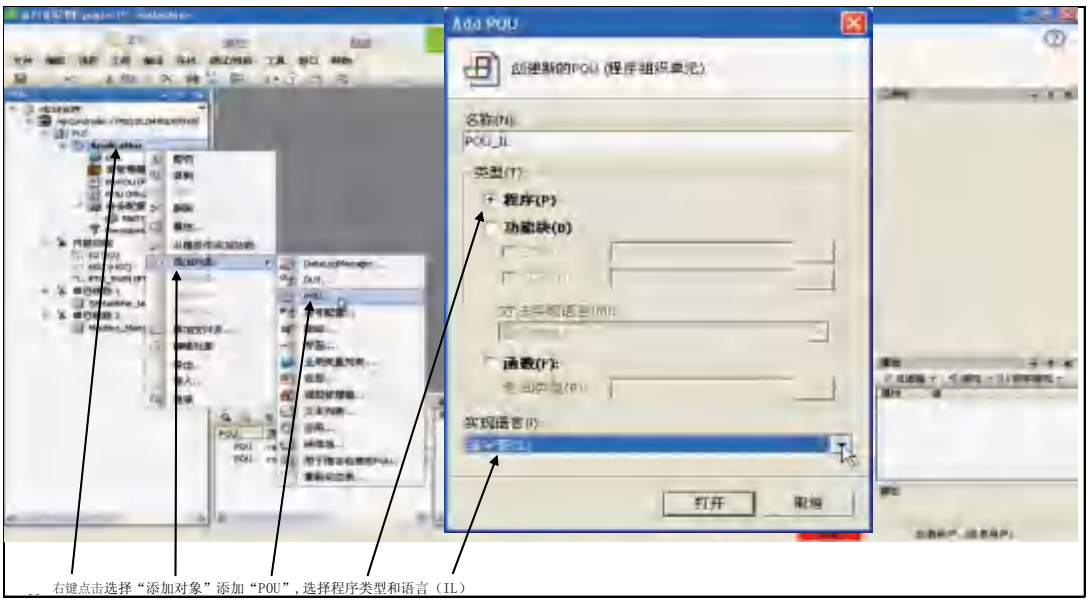
每条指令占据一行，包含一个操作符和一个或多个用逗号隔开的操作数。操作数之间用逗号分隔。

每行的指令前面可以有标题，标题后要有冒号。例如：在下文中的例子"ml:" 标题可以是跳转指令的目的地，在下文中的例子“跳转到下一个”。

每行结束处可以加入注释。每行指令之间可以插入空行。

一、指令表程序建立

类型：程序、功能块、函数



二、表格编辑器的结构

1.操作符：该栏目包括指令列表操作符（LD，ST，CAL，AND,OR等等）或是函数名在调用功能块的同时也指定各自的参数。在这种情况下，在前缀字段必须输入“:=”或是“=>”。

2.操作数：该字段仅能包括一个操作数或是一个跳转标签。如果需用多于一个操作数（多操作数/扩展操作数 “AND

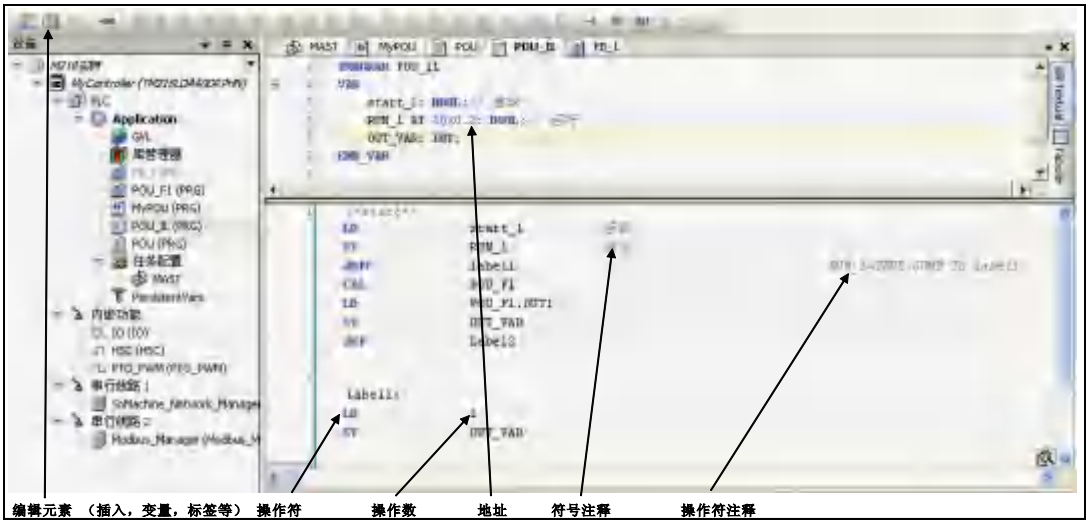
A,B,C”或是调用带有多个参数的函数），那些操作数必须在后面各行中写入并且操作数字段要为空。在这种情况下，加入参数分隔逗号。在调用功能块，程序或动作时，必须加上相应的打开或关闭括号。

3.地址：该字段包括操作数在声明部分定义的地址。该字段不能编辑，并且可以通过“显示符号地址”来切换打开和关闭。

4.符号注释：该字段包括在声明部分为操作数定义的注释。该字段不能编辑，并且能通过“显示符号注释”选项来切换打开或是关闭。

5.操作符注释：该字段包括当前行的注释。是可编辑的，并且可以通过“显示操作符注释”选项.来切换打开或是关闭。

示例如图所示：



操作编辑表格方法:

Up- and Down 箭头键： 向前或后移动

<Tab>:在一行内移动到右边

<Shift>+<Tab>:在一行内移动到左边

<Ctrl>+<Enter>: 在当前行下输入一个新行。

<Delete>: 删除当前行，也就是当前选中的所有域。

Cut, Copy, Paste: 剪切，复制，粘贴：至少选中此行的一个区，使用复制命令可以复制一行或几行。使用剪切命令剪切一行。粘贴将会在当前行被选中前插入到复制剪切行，如果没有任何域被选中，将会插入到节的结尾。

<Ctrl>+<Home> 滚动到文档开始且标记首节。

<Ctrl+End> 滚动到文档的结尾且标记后节。

<PageUp> 滚动到上一个屏幕且标记最上面的矩形。

<PageDown> 滚动到下一个屏幕且标记最上面的矩形。

三、指令表（IL）逻辑运算程序设计

逻辑运算使用"AND"，"AND NOT"，"OR"等语句进行程序设计
1.在一个程序节中，第一行语句必须始终为"LD"
2.如必须使用括号，则无法100％使用布尔代数

注意: 只可以使用圆括号"（"和 "）"

括号规则：
运算结果始终使用"）"语句进行设置
此处得到的结果，将与下一个接点或下一对括号一起用作逻辑上的"AND"

规则：

1. 多个接点并联到另外一个接点上，在括号中必须设置"OR"分支

2. 只有一个接点并联，"AND"应先于"OR"应用，且不需要括号

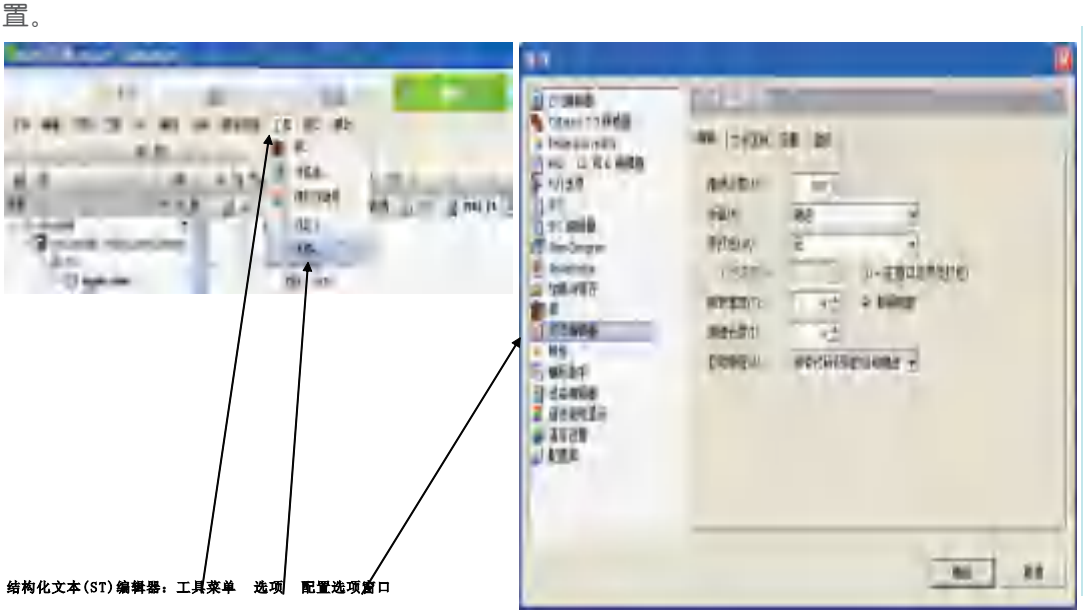
示例如图所示：

4.3.2 结构化文本(ST)编辑器

结构化文本ST编辑器是用来在IEC编程语言的结构化文本（ST）或扩展结构化文本中创建程序对象。其中扩展结构化文本是对IEC61131-3标准的扩展。

ST语言是文本类型的高级编程语言，类似于PASCAL语言或C语言。程序代码由表达式和指令组成。与IL（指令列表）相比，ST可以使用大量的循环编程结构，从而开发出复杂的算法。

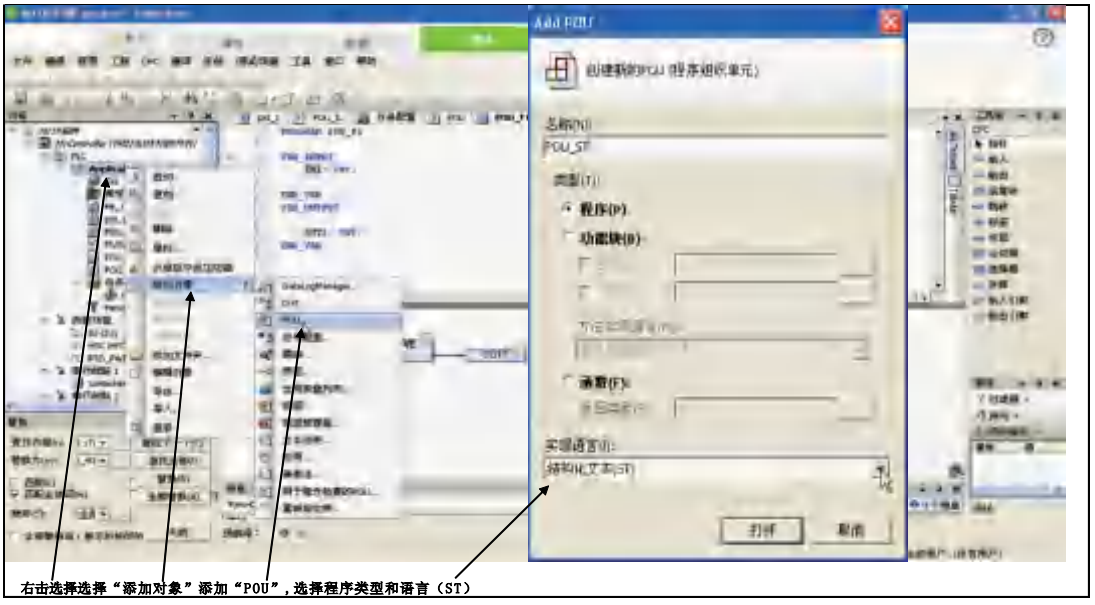
编辑器的设置：在“工具”的“选项”对话框中，有相应的文本编辑器设置，可以用来配置ST编辑器的特性、显示和菜单。在此处，可以对高亮显示颜色、行号、制表、缩进等进行缺省设置。



结构化文本(ST)编辑器：工具菜单　选项　配置选项窗口

一、指令表程序建立

类型：程序、功能块、函数



右击选择选择“添加对象”添加“POU”，选择程序类型和语言（ST）

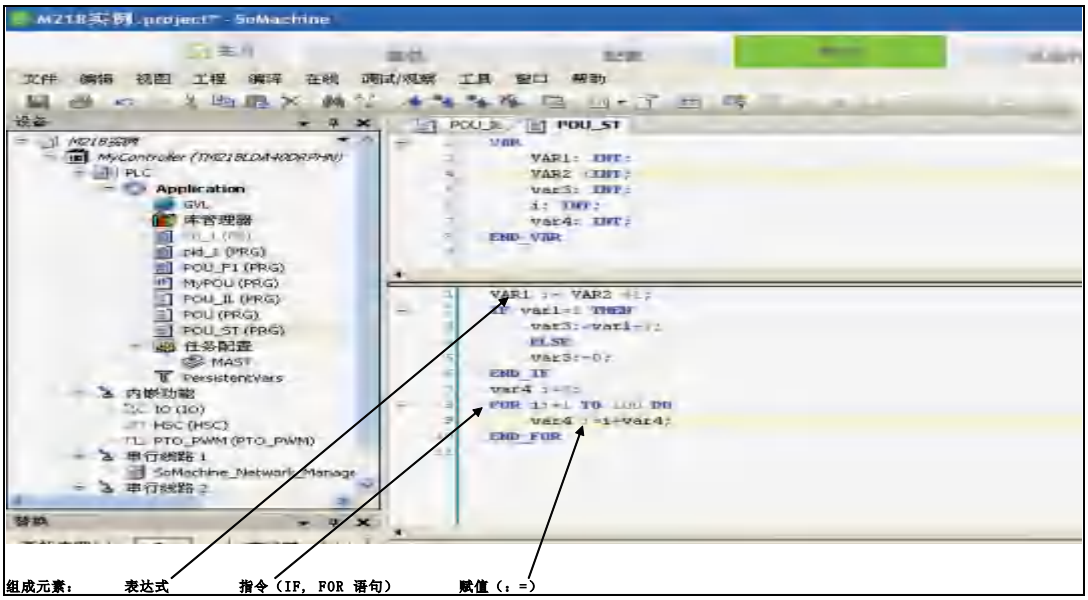
二、文本编辑器的组成

1.表达式：是计算之后获得返回值的结构。在指令中需要使用该返回值。表达式由操作符、操作数、赋值组成。操作数可以是常量、变量、函数调用的返回值或其它表达式。

2.指令：用于对表达式进行某些操作。

3.赋值操作符：作用是将其右侧的表达式产生的值赋给左侧的操作数（变量或地址）。

示例如图所示：



三、ST指令的基本编程设计

1.FOR:循环，编写循环过程。

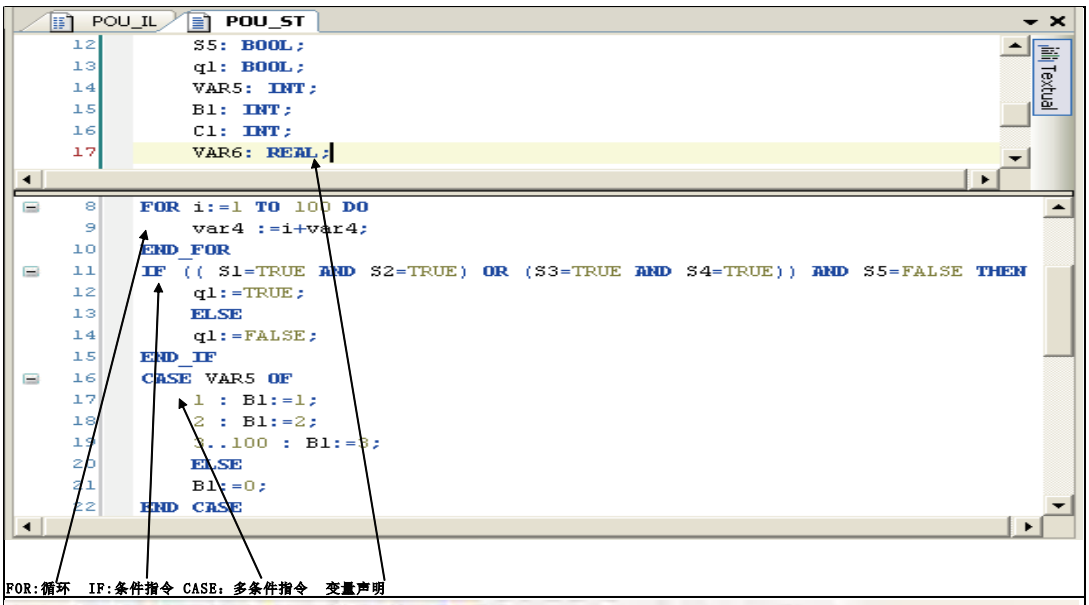
2.IF:条件指令，可以检查条件，并根据此条件执行相应的指令。

3.CASE：多条件指令，在程序中使用相同的条件变量，通过CASE指令可以连接多个条件指令。

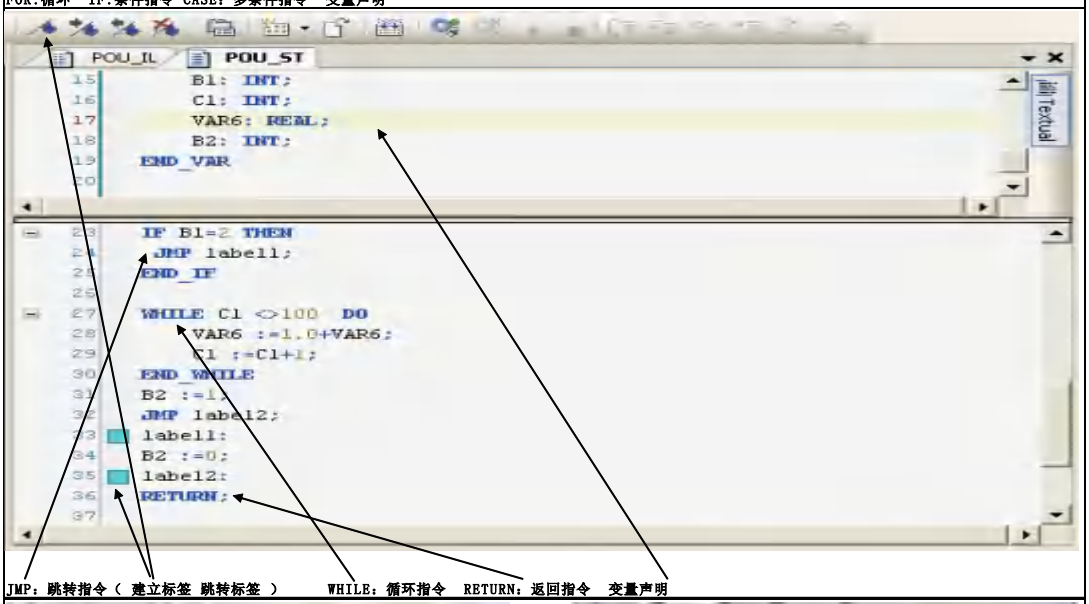
4.JMP：跳转指令，可以用于无条件跳转到使用跳转标号标记的代码行。

5.WHILE：循环指令，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。

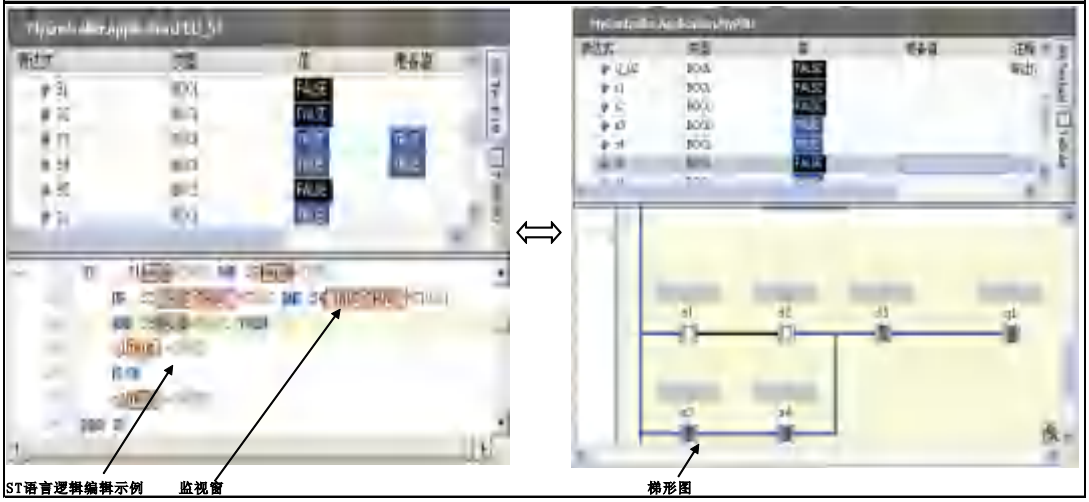
6.RETURN：返回指令，用于退出POU。



FOR:循环 IF:条件指令 CASE：多条件指令 变量声明



JMP：跳转指令（ 建立标签 跳转标签 ） WHILE：循环指令 RETURN：返回指令 变量声明



ST语言逻辑编辑示例 监视窗 梯形图

4.4 图形化编辑器

主要编辑器如下：

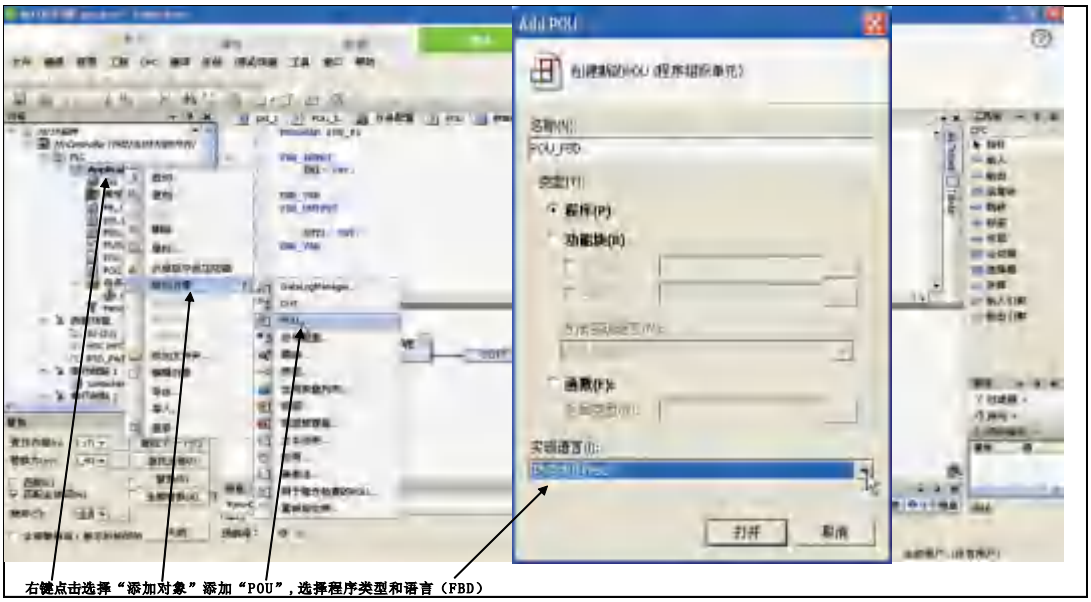
- 功能块图形FBD编辑器
- 梯形图LD编辑器
- 连续功能图CFC编辑器
- 顺序流程图SFC编辑器

4.4.1 功能块图形FBD编辑器

功能块图是一种图形化的编程语言。它由一系列的节组成，每一个节又由图形结构的运算块和连接线的组成，实现逻辑、运算、调用、跳转或返回功能。

一、功能块图形FBD程序建立

类型：程序、功能块、函数



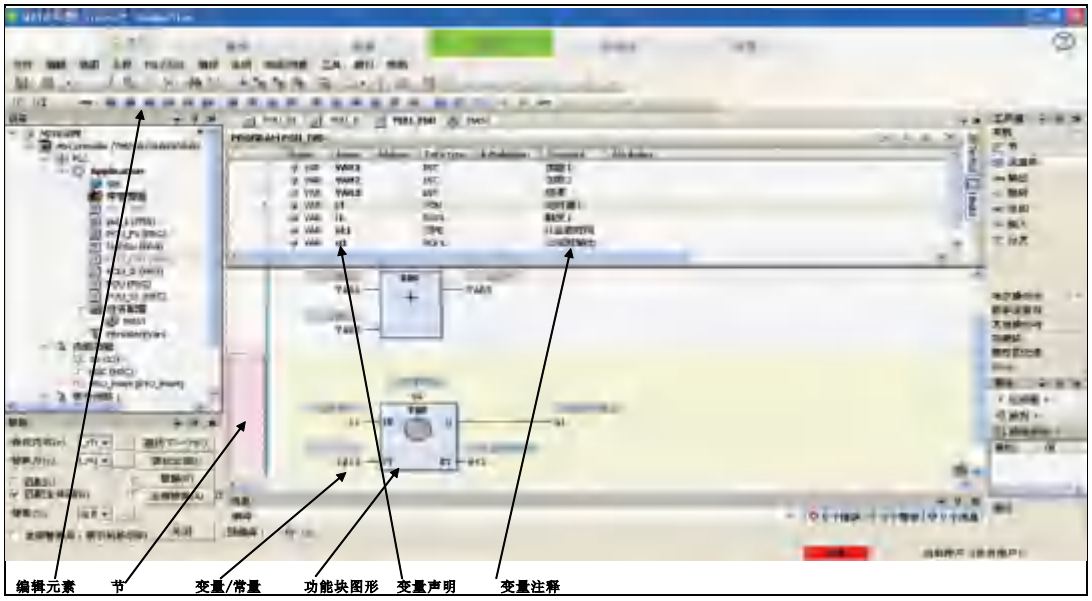
二、功能块图形FBD的组成

“节”是FBD程序的基本组成单元。每个节包含逻辑式或运算表达式，POU（函数，程序，功能块调用等），跳转，返回指令。

当创建一个新的对象时，编辑窗口将会自动添加一个空节。

库或对象属性的功能块或函数提供了图标（位图），该图标将显示在FBD编辑器的运算块中。标准操作符也有图标。

示例如图所示：



三、功能块图形FBD的基本编程设计

1.插入节

2.添加功能块图形

3.填写功能块图形指令

4.定义功能块

5.输入变量

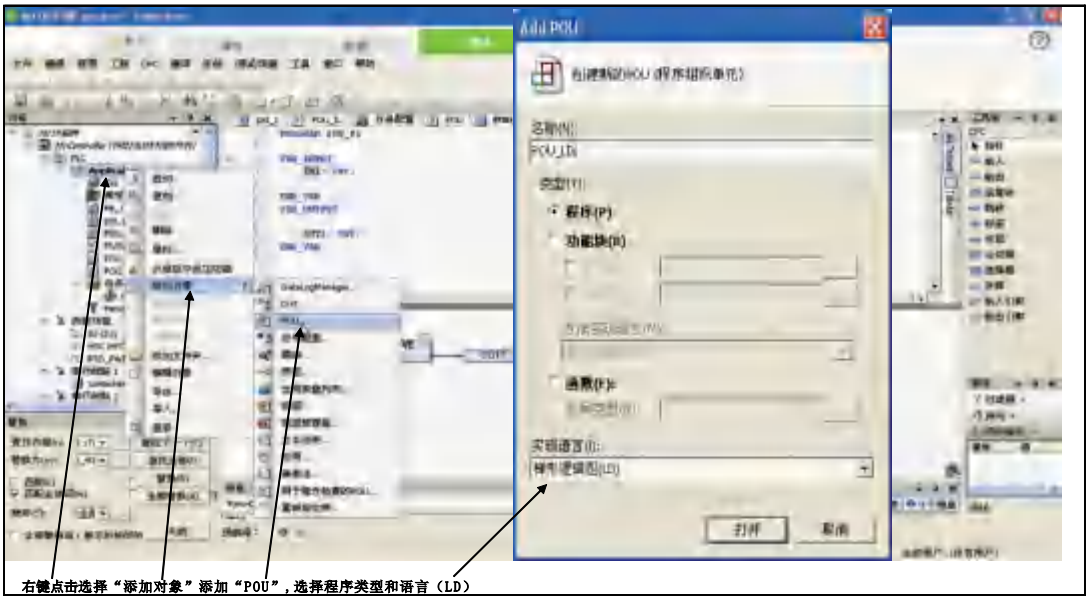
示例如图所示：

4.4.2 梯形图LD编辑器

梯形图是一种图形化的编程语言，它类似于电路的结构。梯形图不但很适用于逻辑的转换，并且它也能创建类似于FBD中的节，所以用梯形图调用程序组织单元也是是很方便的。梯形图包含了一系列的节，左右两边各有一个垂直的电流线（能量线）限制其范围，在中间是由触点、线圈、连接线组成的电路图。每一个节的左边有一系列触点，这些触点根据布尔变量值的TRUE和FALSE来传递从左到右的开和关的状态。每一个触点是一个布尔变量，如变量值为TRUE，通过连接线从左到右传递状态。否则传递"关"的状态。在节最右边的线圈，根据左边的状态获得一个开或关的值，并相应地赋给一个布尔变量真或假值。

一、梯形图LD程序建立

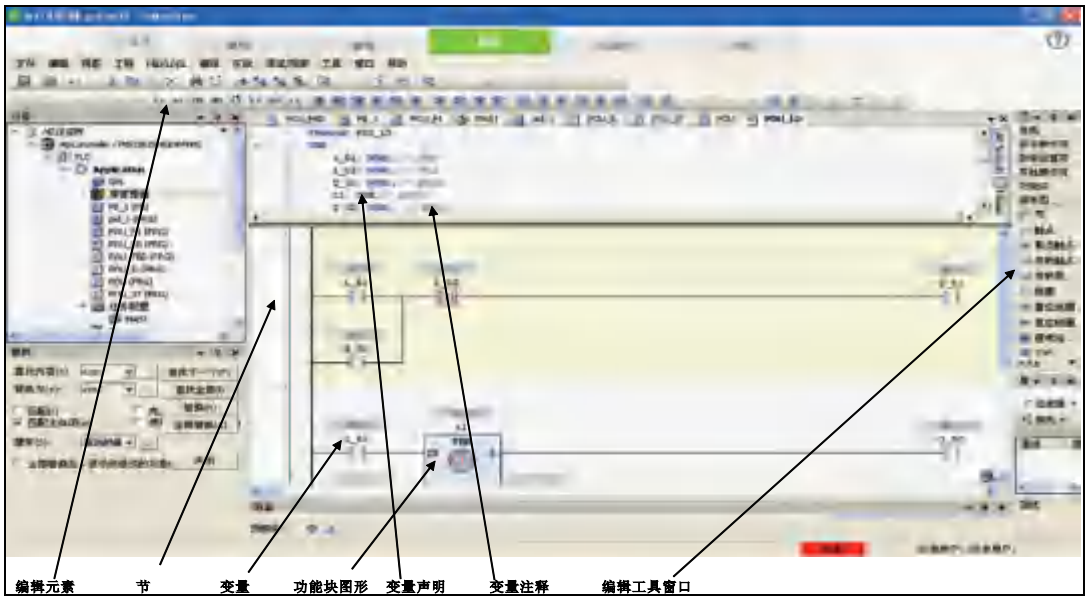
类型：程序、功能块、函数



二、梯形图LD的组成

“节”是LD程序的基本组成单元。每个节包含逻辑式或运算表达式，POU（函数，程序，功能块调用等），跳转，返回指令。当创建一个新的对象时，编辑窗口将会自动添加一个空节。库或对象属性的功能块或函数提供了图标（位图），该图标将显示在LD编辑器的运算块中。标准操作符也有图标。

示例如图所示：



三、梯形图LD的基本编程设计

1.插入节

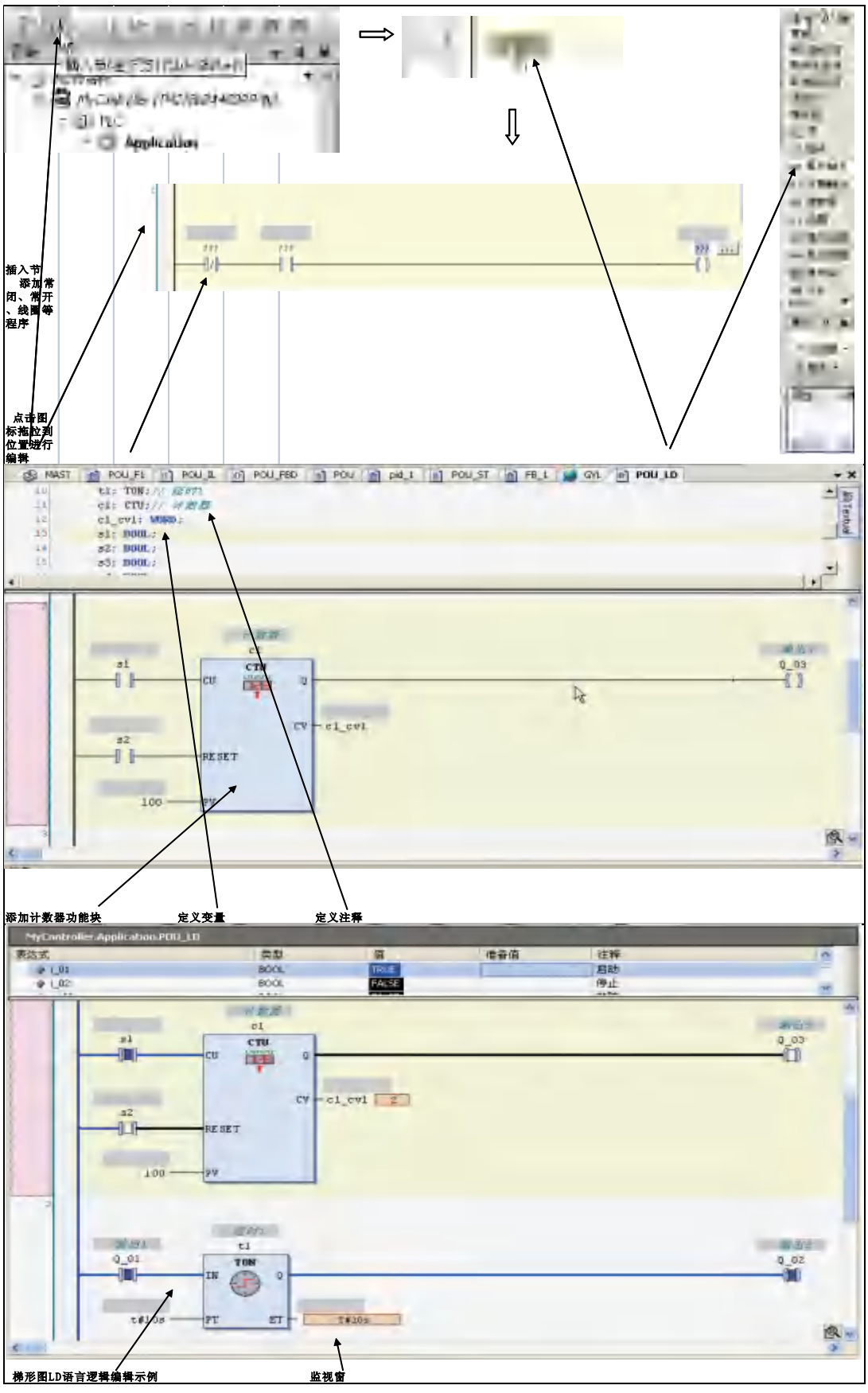
2.添加常闭、常开、线圈、功能块图形等

3.填写功能块图形指令

4.定义功能块

5.输入变量

示例如图所示：

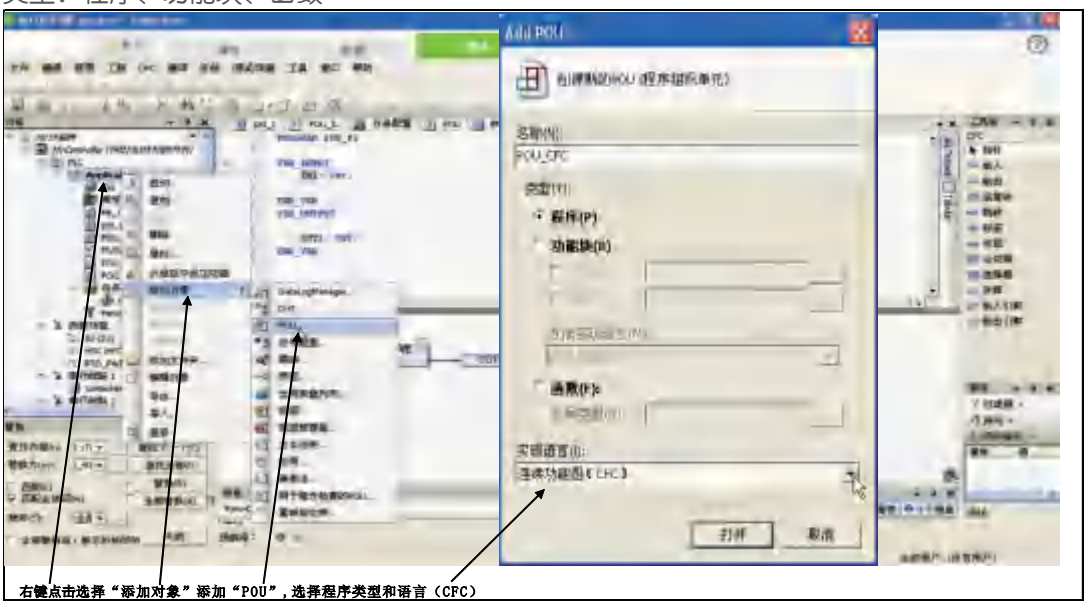

插入节
添加常闭、常开、线圈等程序
点击图标拖拉到位置进行编辑
添加计数器功能块
定义变量
定义注释
梯形图LD语言逻辑编辑示例
监视窗

4.4.3 连续功能图CFC编辑器

连续功能图是IEC61131-3标准编程语言的扩展，是基于功能块图的图形化编程语言，但它没有网络限制，可任意放置元素，例如允许插入反馈回路。与网格编辑器不同，CFC编辑器允许把元素放在任何位置，例如，允许直接插入反馈回路。CFC编辑器内部有一个链表，包含了所有已经插入的元素，链表的顺序决定了CFC元素的执行顺序，但是用户可以改变元素的执行顺序。在CFC程序中，当光标在元素上移动时，以灰色阴影方式显示的区域可能是光标的位置。点击其中一个阴影区域，该区域立刻变为红色。当释放鼠标键后，该区域立刻变成当前光标位置，相关元素或文本被选中并且用红色显示。

一、连续功能图CFC程序建立

类型：程序、功能块、函数



二、连续功能图CFC的组成

系统库或对象属性的功能块或函数提供了图标（位图），该图标将显示在CFC编辑器的运算块中。标准操作符也有图标。

主要组成元素

1.变量

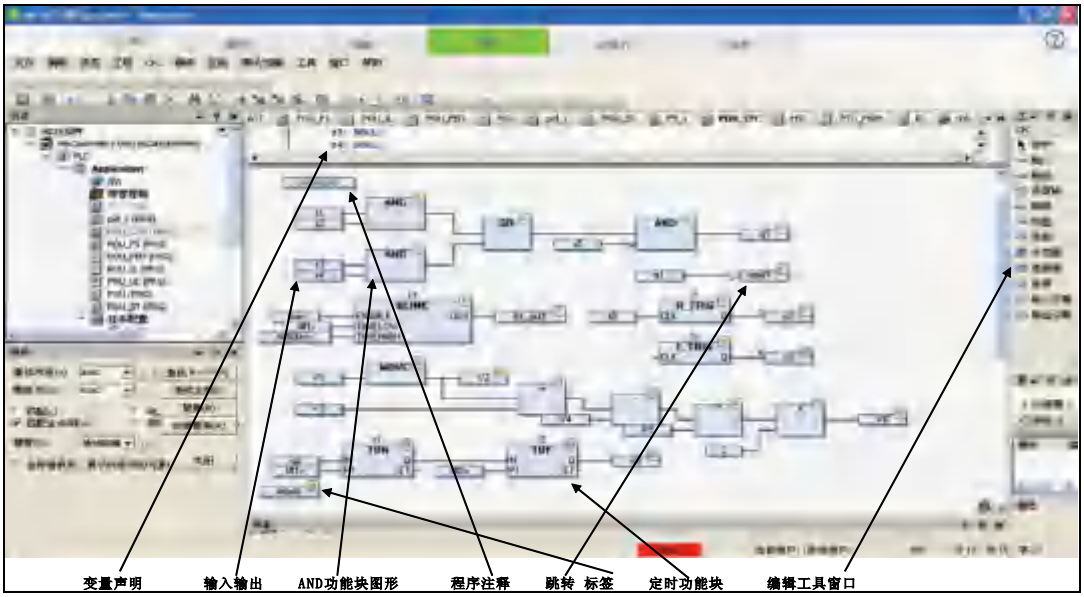
2.输入、输出

3.运算块

4.跳转、标签

5.注释

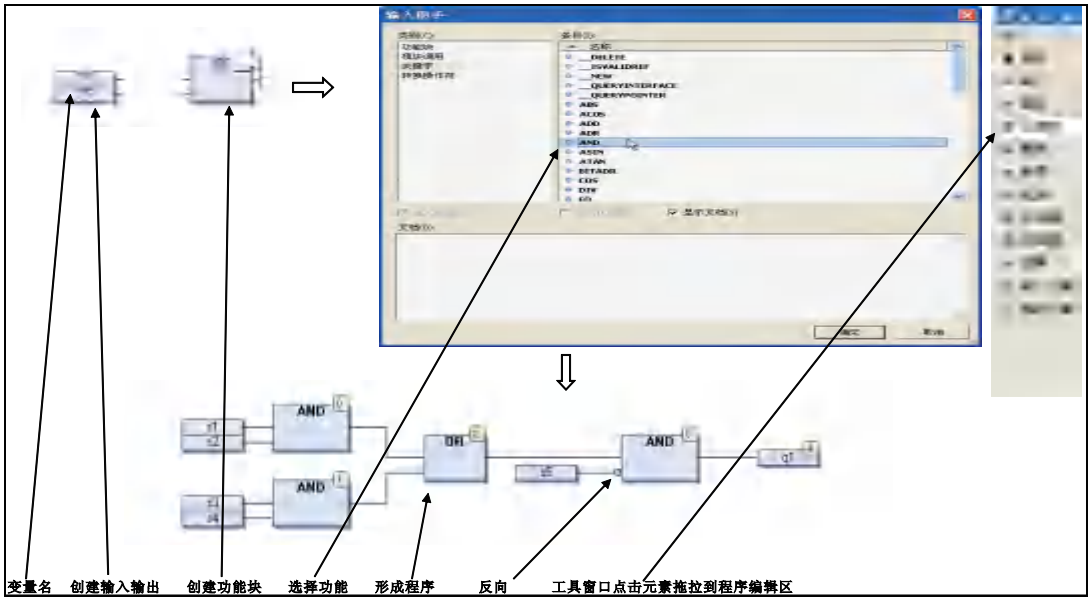
示例如图所示：



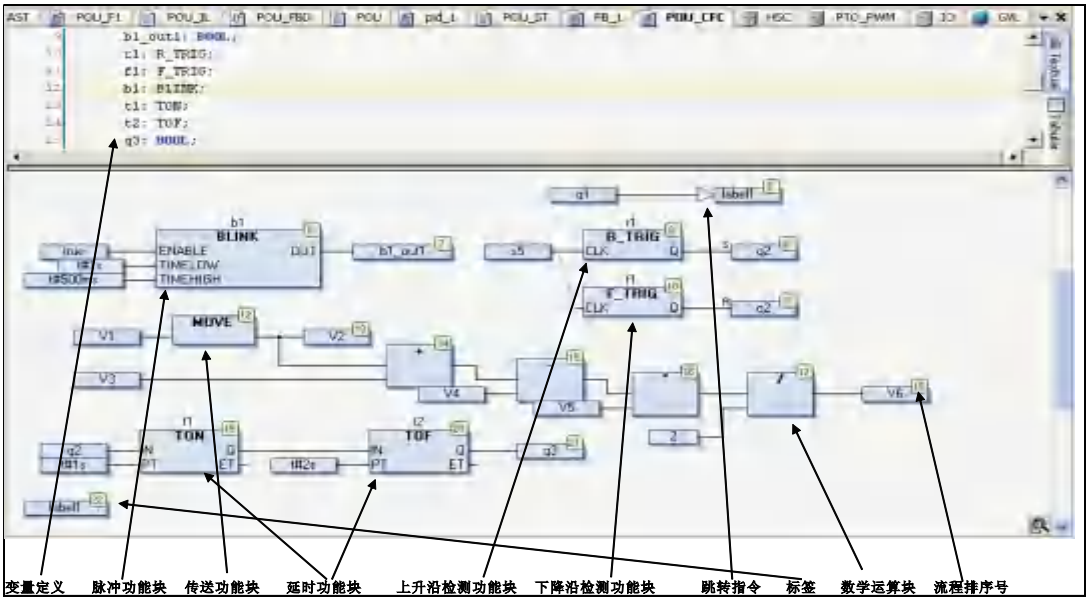
三、连续功能图CFC的基本编程设计

1.逻辑编程
2.常用功能块编程
3.特殊功能块编程（如硬件关联的HSC高速计数、PTO高速输出功能块等）
4.程序完成后，排序。（排序原则：从上到下，从左到右）

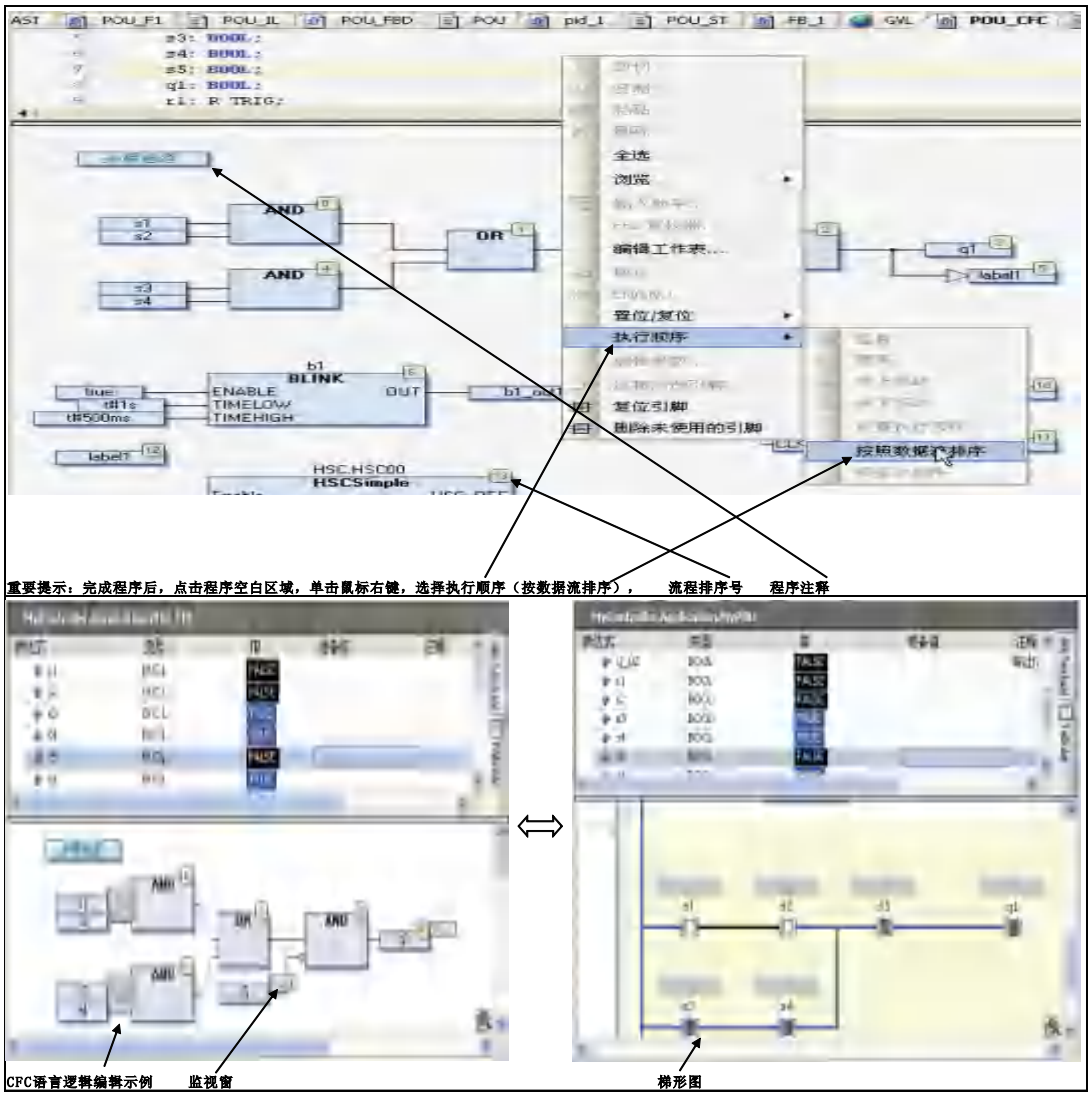
1.逻辑编程示例如图所示：



2.常用功能块编程示例如图所示：



3.特殊功能块编程示例如图所示：

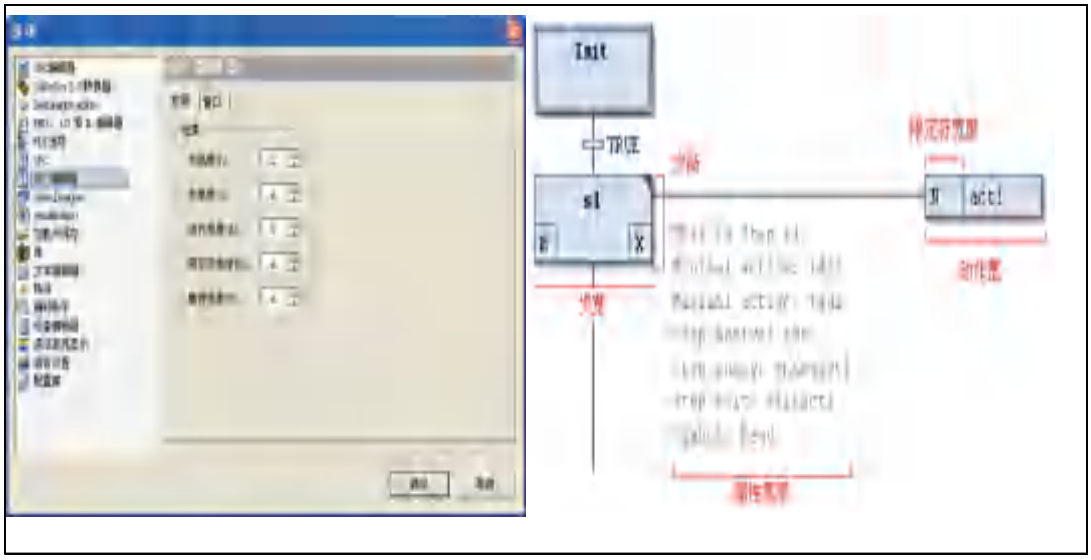


4.4.4 顺序流程图SFC编辑器

顺序流程图是IEC61131-3 标准图形化编程语言，主要用于对程序流程的控制。在一个SFC图表中所用到的元素，缺省情况下都在“SFC菜单”中，并且只要SFC编辑器处于活动状态，这些元素就是可用的。这些元素由“转移”将“步”连接为“串行”或“并行”序列。

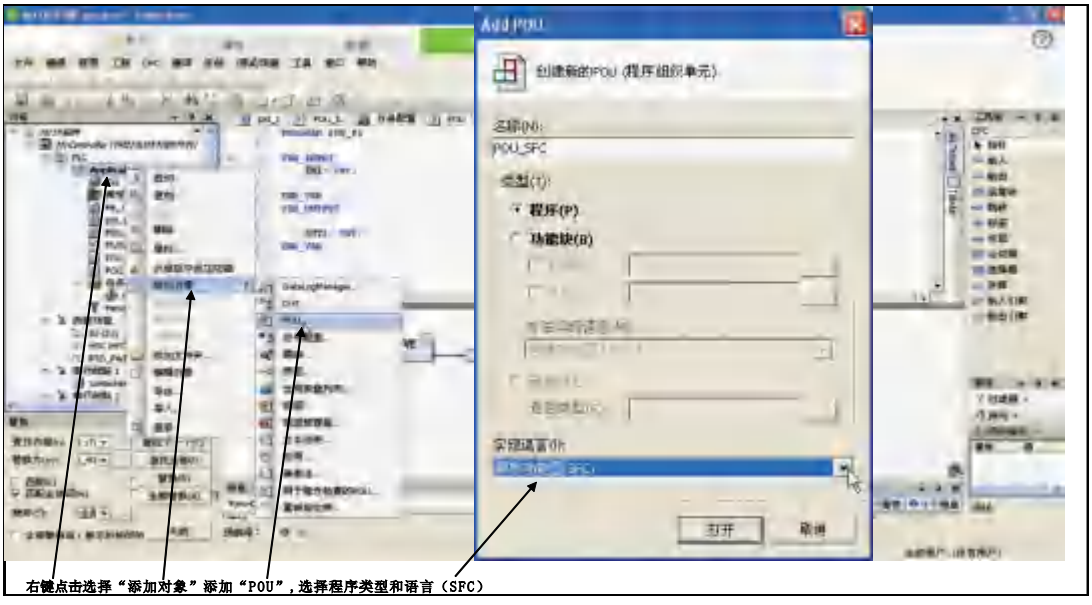
一、顺序流程图SFC编辑器设置

操作和外观由SFC的设置来确定，这些设置在“工具”-“选项”对话框中布局和窗口定义。



二、顺序流程图SFC程序建立

类型只有：程序、功能块二种

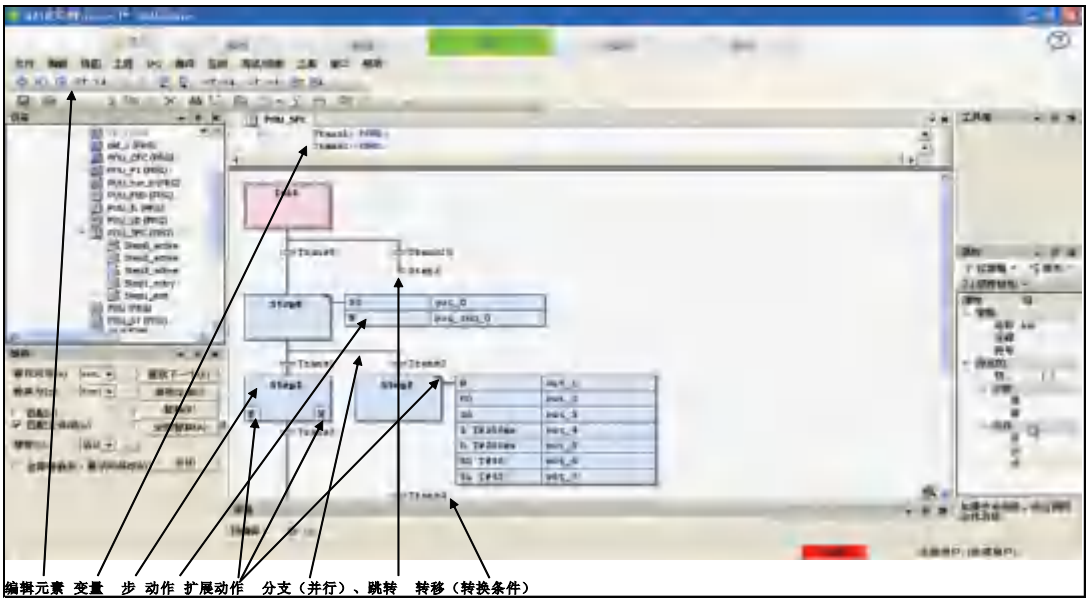


三、顺序流程图SFC的组成

系统库或对象属性的功能块或函数提供了图标（位图），该图标将显示在SFC编辑器的运算块中。标准操作符也有图标。

主要组成元素：步、转移、动作、分支（可选）、分支（并行）、跳转、宏。

示例如图所示：



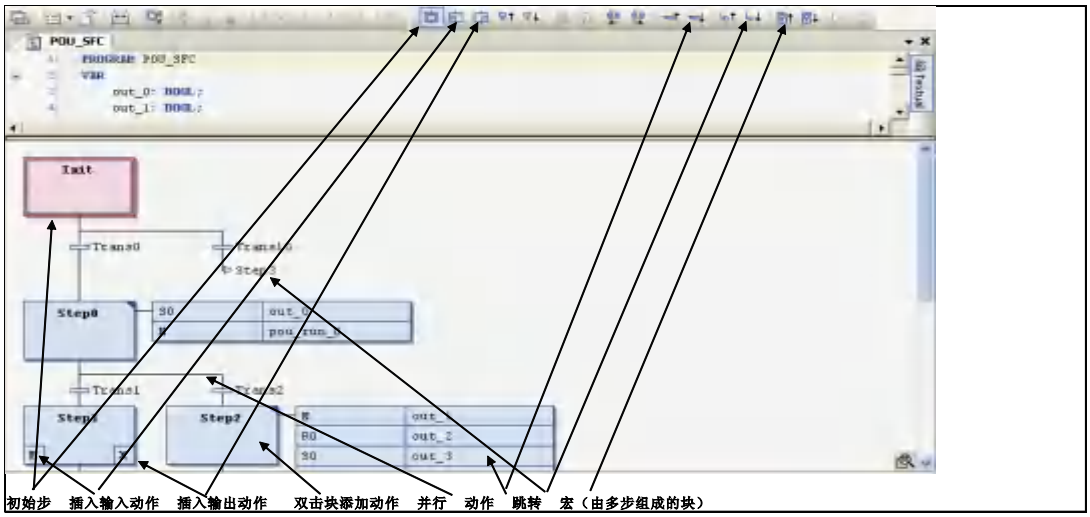
四、顺序流程图SFC的基本编程设计

1.步编程

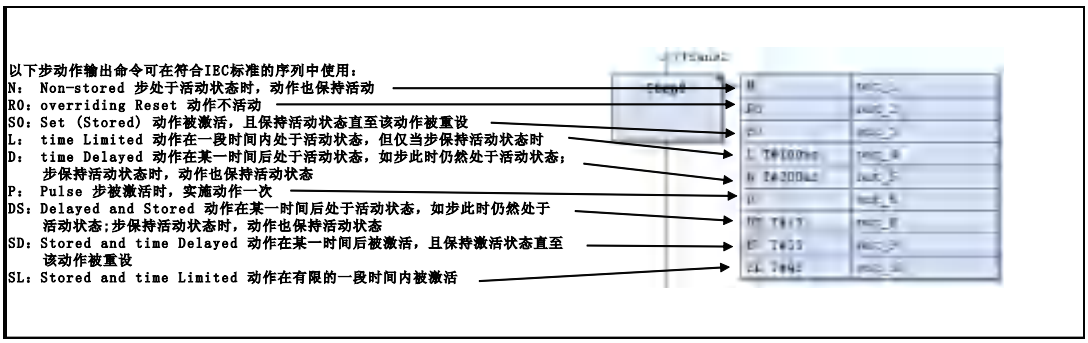
2.常用步动作编程

3.动作调用和宏调用编程

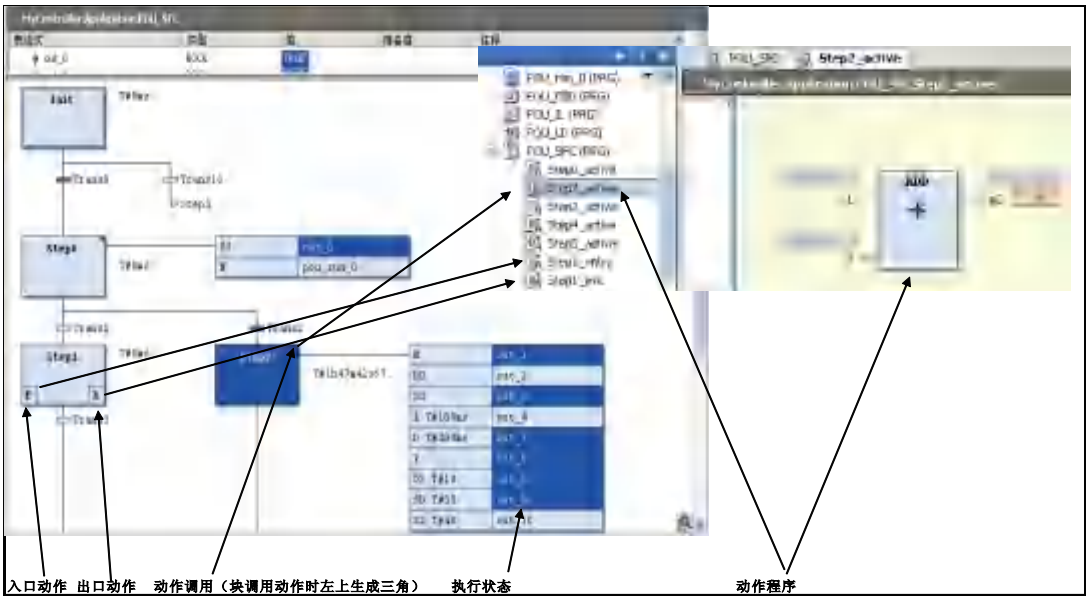
1.示例如图所示：



2.常用步动作编程：采用IEC－步动作命令



3..动作调用和宏调用编程



第五章 SoMachine项目管理

5.1主页

一、启动 SoMachine 软件

三种启动方式：

1、依次点击 开始 → 程序 → Schneider Electric → SoMachine 菜单；

2、双击桌面上的 SoMachine 图标；

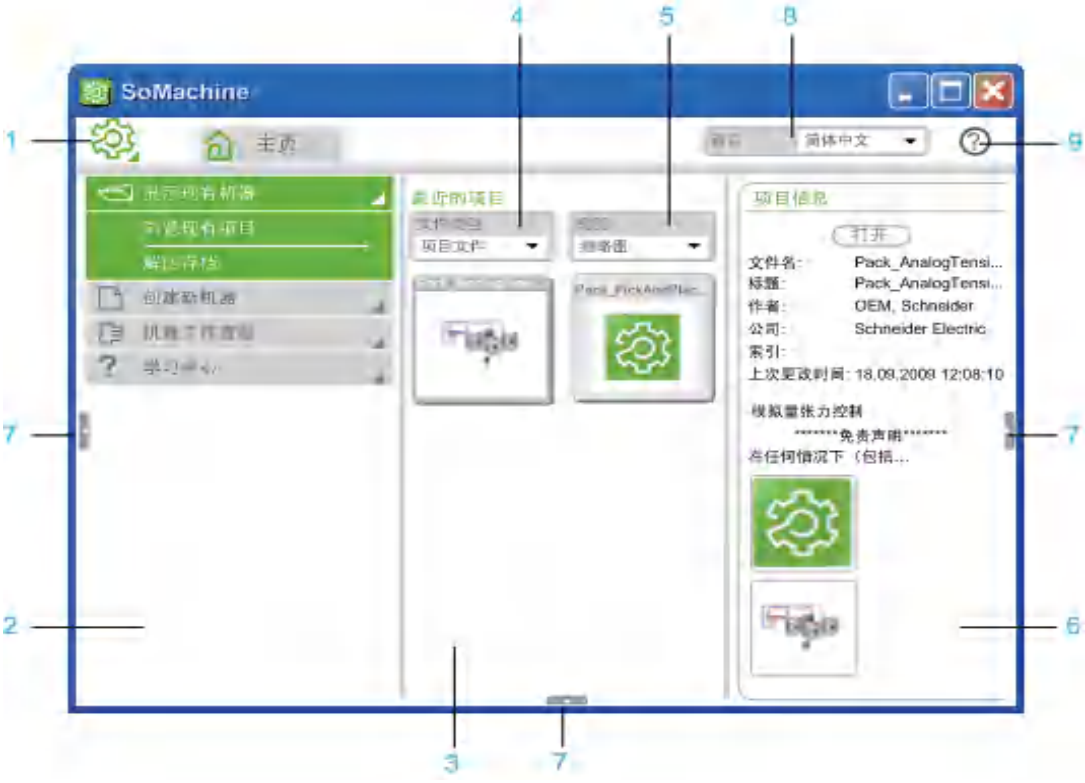
3、双击一个已有 SoMachine 项目文件启动 SoMachine。

注意： 如果通过开始菜单或通过双击 SoMachine 图标启动 SoMachine，则主选择屏幕将显示为图5.1形式。如果通过双击已有 SoMachine 项目文件启动 SoMachine，则 SoMachine 将带程序 选项卡启动。此时显示一个启动屏幕，表示该程序正在启动。

主选择屏幕的元素

成功启动 SoMachine 后，将显示主选择屏幕（主页选项卡），通过该屏幕可以访问软件的相应功能。

SoMachine 启动后将显示如图5.1所示窗口：



1、通过 SoMachine 图标可以访问常规功能菜单，单击鼠标左键可以展开下拉菜单。

2、左侧任务的选择面板显示主页屏幕提供的任务，单击图柄（图5.1中7所示）可隐藏该面板。

3、中间的工作区域始终显示。通过它可以访问最近打开过的项目。

4、通过文件类型列表，可以在此工作区域中显示所有文件，或者只显示项目文件或库文件或 CoDeSys 项目文件 (3.0 版以前) 或 CoDeSys 库文件 (3.0 版以前)。

5、通过视图列表可以从项目的缩略图视图更改为更加详细的列表视图。

6、右侧的信息面板显示有关工作区中当前选定项目的详细信息。它包含用于打开选定项目的打开按钮。你可通过单击图柄隐藏此面板。

7、图柄用于隐藏或显示右侧的任务选择面板，缺省情况下，启动屏幕不显示左侧的信息面板和窗口底部的消息面板。

8、语言列表用于更改 SoMachine 用户界面的语言，在更改语言后必须重新启动 SoMachine，更改才能生效。

9、点击‘？’图标可快速打开 SoMachine 在线帮助系统。

主页选项卡的任务

借助主选择屏幕左侧的任务选择面板，您可以根据 SoMachine 功能可以执行的任务，将这些功能分组到不同的文件夹。

主页屏幕的功能	子功能
显示现有机器	此文件夹提供的子任务用于打开已经存在的 SoMachine 项目并用于打开项目存档文件。
创建新机器	此文件夹提供的子任务用于创建新的SoMachine 项目文件或库文件（从头开始或使用模板）。
机器工作流程	此文件夹提供的子任务专用于试运行机器。此文件夹中不提供试运行任务以外的命令 。
学习中心	此文件夹提供有关 SoMachine 的详细信息 。

关闭 SoMachine 项目

主页选项卡还可以用于关闭 SoMachine 项目。处于项目中时，只需单击主页选项卡便会关闭项目，并使您返回到上图所示的主选择屏幕。

二、常规功能菜单

单击在 SoMachine 窗口左上角可见的 SoMachine 图标可以访问常规功能菜单（图）。

常规功能菜单提供下列命令：

命令	描述
主页	执行此命令可以返回到 主页选项卡（关闭当前打开的项目），该选项卡是 SoMachine 的主要选择屏幕。
保存	执行此命令可以保存对当前打开的项目的更改。
另存为	执行此命令可以将当前打开的项目 /库保存到其他位置或其他名称下面。会显示标准 Windows 保存项目对话框，您可以浏览至新文件夹和 /或输入新文件名称。
保存/发送存档	执行此命令可以创建当前打开的 SoMachine 项目的存档文件，以及： • 使用项目存档 对话框的 保存 按钮将存档文件保存到连接的驱动器。 或 • 使用项目存档 对话框的 发送 按钮，创建要附加到空电子邮件的临时存档文件。如果消息应用程序编程接口 (MAPI) 安装正确，则 SoMachine 会自动创建此电子邮件。请与当地管理员联系以获得详细信息 。
首选项	执行此命令可以配置： • **首选路径**，用于打开和保存 SoMachine 项目。 • **在线轮询间隔**（毫秒），为 2 次尝试轮询连接的设备之间必须经过的时间间隔。 • **路由步宽度**，定义图形配置编辑器的网格大小。选择一个介于 15（宽网格）和 50（小网格）之间的值，请记住，较小的网格（较大的值）会导致刷新配置的图形表示形式的时间较长。 • **路由超时**，定义为创建直角连接线而允许 图形配置编辑器 对配置的图形表示形式进行路由（重新路由）的时间。如果在路由过程完成之前超时，则剩余的线将显示为不带直角。 • Stop Gateway on Exit 复选框定义网关在关闭 SoMachine 最后一个实例后的行为： o 复选框处于选中状态（缺省） 网关在您退出最后一个SoMachine实例后自动停止。如果使用网关的其他 SoMachine 实例仍在运行，则网关不会停止。 o 复选框未选中 退出最后一个SoMachine实例后网关不会自动停止。 • 单击选项按钮进行常规设置。
帮助	执行此命令可以打开 SoMachine 在线帮助。
关于	执行此命令可以打开 关于 对话框，该对话框提供有关当前安装的 SoMachine 版本的信息以及许可证和技术信息。
退出	执行此命令可以关闭 SoMachine。

三、显示现有机器

通过**显示现有机器**任务，您可以打开已存在的项目。

该任务包括以下子任务：

•**浏览现有项目**，用于打开现有 SoMachine 项目
•**解压存档**，用于打开现有 SoMachine 项目存档文件

打开最近的项目

工作区使您可以快速访问之前打开的 SoMachine 项目。您可以在选项对话框中定义要在此处显示的项目数，该对话框可通过常规功能菜单**首选项 → 选项 → 加载和保存 → 显示最近使用的列表中的 [1-16]** 项进行访问。

1、最近打开的项目将显示在 SoMachine 工作区中。

2、通过文件类型列表，可以在此工作区域中显示所有文件，或者只显示项目文件或库文件或 CoDeSys 项目文件 (3.0 版以前) 或 CoDeSys 库文件 (3.0 版以前)。

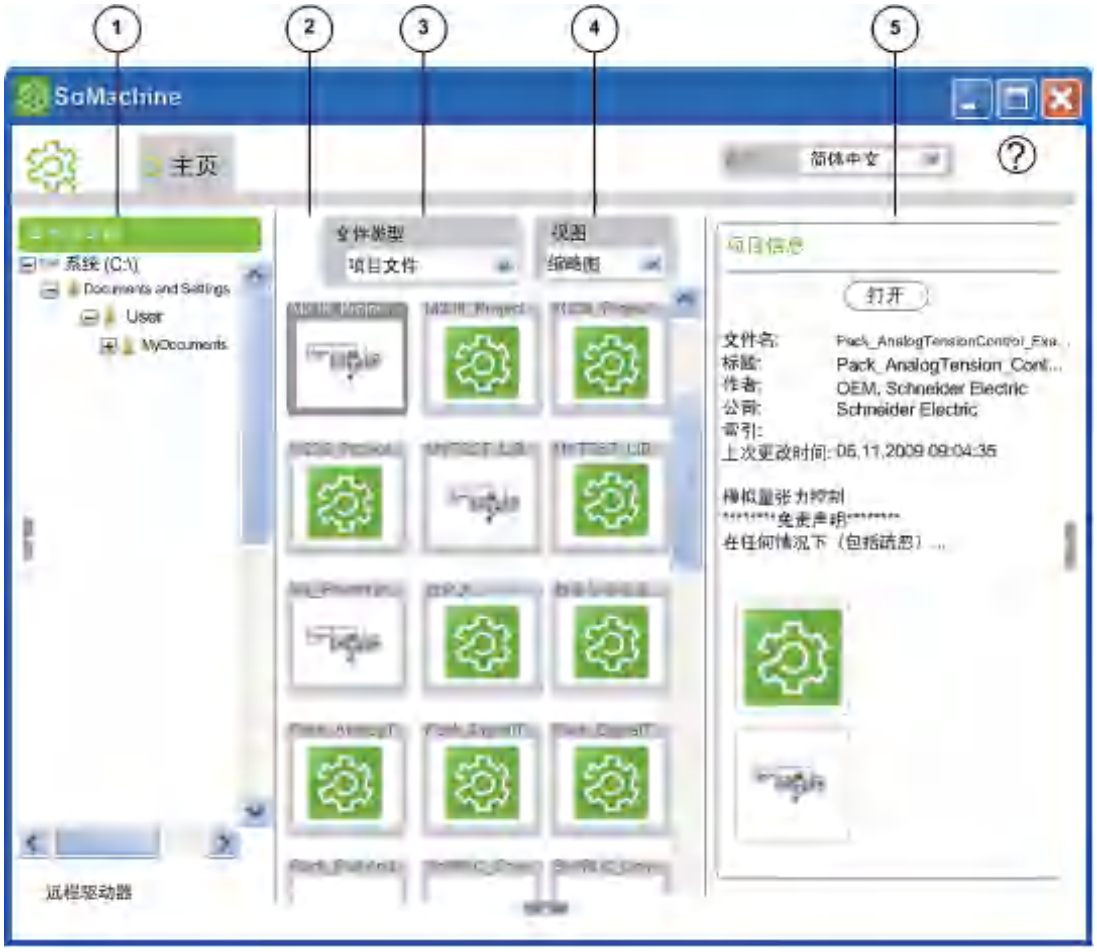
3、通过视图列表可以从项目的缩略图视图更改为更加详细的列表视图。

4、信息面板显示与在工作区中所选的项目相关的项目信息。要打开选定项目，请单击打开按钮。

在工作区中双击项目图标，或在工作区中选择项目并单击信息面板中的打开按钮，便可在 SoMachine 中打开项目。SoMachine将选择属性选项卡，以分配更多信息。

浏览现有项目

使用浏览现有项目子任务可以浏览本地驱动器或连接的网络驱动器上的现有项目。



1、任务选择面板显示已连接驱动器（与 Windows 资源管理器类似）。

2、工作区显示在选择面板中所选的文件夹中可用的 SoMachine 项目。

3、通过**文件类型**列表，可以在此工作区域中显示**所有文件**，或者只显示**项目文件**或**库文件**或 CoDeSys 项目文件 (3.0 版以前) 或 CoDeSys 库文件 (3.0 版以前)。

4、通过视图列表可以从项目的缩略图视图更改为更加详细的列表视图。

5、信息面板显示与在工作区中所选的项目相关的项目信息。要打开选定项目，请单击打开按钮。

在任务选择面板中，浏览至包含要打开的 SoMachine 项目的文件夹。在工作区中双击项目图标，或在工作区中选择项目并单击信息面板中的打开按钮，便可在 SoMachine 中打开项目。SoMachine将选择属性选项卡，以分配更多信息。

要访问未在 PC 上映射的网络驱动器，请执行以下步骤：

步骤	操作
1	在任务选择面板上单击 **远程驱动器** 任务。
2	按照以下语法，在文本框中输入服务器名称和子文件夹名称： \\服务器名称 \文件夹名称
3	单击**打开**按钮。 结果：服务器/子文件夹的内容将显示在任务选择面板中（ 类似于 Windows 资源管理器）。

打开项目存档

使用解压存档子任务可以打开现有 SoMachine 项目存档文件。
要打开现有 SoMachine 项目存档文件，请执行以下步骤：

步骤	操作
1	从任务选择面板中选择**解压存档**子任务。 结果 ：将显示**解压存档**对话框。
2	浏览至包含要打开的项目存档文件的文件夹。
3	选择项目存档文件，然后单击打开按钮。 结果 ： **工程存档**对话框随即打开。
4	在 **工程存档**对话框中，选择要从存档文件中解压的 **所引用设备**和**所引用的库** ，然后单击 解压按钮。 结果 ： SoMachine 会从存档文件中解压所选项目，并自动打开项目。

四、创建新机器

1、使用空项目启动

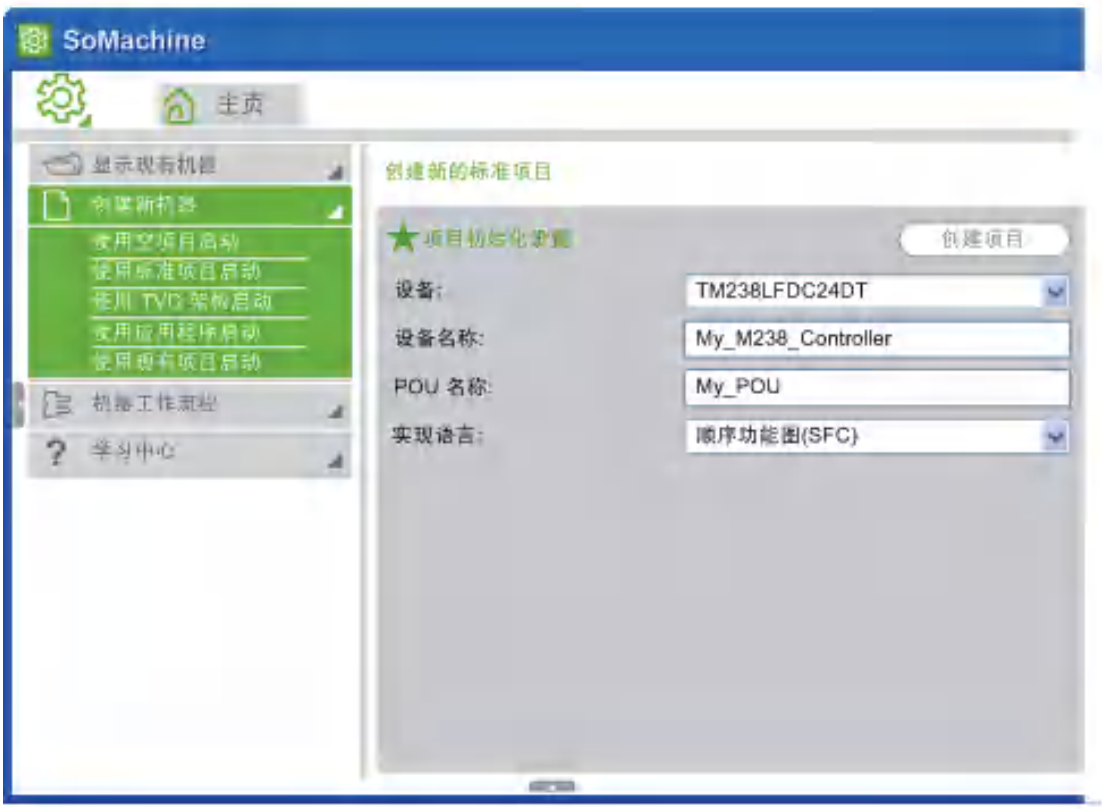
用于创建没有任何预配置设备或设置的全新项目。它可打开将项目另存为对话框，在该对话框中，您可以浏览目标文件夹，并为新项目分配名称。

单击保存可将项目保存到所选文件夹。SoMachine 会打开项目并选择属性选项卡以分配更多信息。

2、使用标准项目启动

子任务用于为您选择的控制器或 HMI 创建新项目。在单一步骤中，您可以为程序组织单元 (POU) 输入名称（适用于控制器和具有控制器的 HMI），然后为此 POU 选择编程语言。

SoMachine 会用您输入的名称创建一个项目，该项目包括所选设备、编程语言和该 POU，并且自动与主站 (MAST) 任务关联。



创建新的标准项目

要创建新的标准项目，请执行以下步骤：

步骤	动作	注释
1	在 主页 选项卡中，选择 **创建新机器** → **使用标准项目启动** 。	–
2	从 **设备** 列表中，选择要为其创建新项目的控制器或 HMI 。	–
3	在 **设备名称** 文本框中，为所选控制器或 HMI 输入名称。	**设备名称** 不能包含任何空格。HMI设备名称的长度不能超过 32 个字符。
4	如果要为控制器或具有控制器的 HMI 创建标准项目，请在 POU 名称文本框中为程序组织单元 (POU) 输入名称。	仅当您在 **设备** 列表中选择了控制器或具有控制器的 HMI 时， POU **名称** 文本框才能进行编辑。如果选择没有控制器的 HMI ，则此文本框将不可用。 POU 名称 文本框不能包含任何空格。
5	如果要为控制器或具有控制器的 HMI 创建标准项目，请从 实现语言 列表中选择一种编程语言。	–
6	单击 **创建项目** 按钮。 结果 ：打开将**项目另存为** 对话框，可让您浏览至目标文件夹并为新项目分配名称。	–
7	单击 **保存** 即可将您的项目保存到选定的文件夹中。 结果 ： SoMachine 将打开项目并选定 **属性** 选项卡 。	如果单击 **程序** 选项卡，您将看到新项目已经包含以下项： • 所选类型的设备，具有指定名称 • 所选语言的 POU ，具有指定名称 • 调用该 POU 的 MAST 任务

3、使用TVD架构启动

可以基于经过测试、验证和归档的架构创建新项目。SoMachine 提供各种具有即用型配置的 TVDA 项目，您可以进行调整，从而满足您的个人要求。可以直接选择经过测试、验证和归档的合适架构，也可以使用 TVD 架构查找器功能查找最符合您个人要求的 TVD 架构。

注意： 所指定的 TVD 架构已经在实际服务条件下进行过测试。当然，您的特定应用要求可能不同于针对这些项目假定的要求。在此情况下，您必须调整该项目以满足您的特定需要。为此，您需要查阅您发现有必要修改或调整的硬件、软件组件和控制器逻辑的特定产品文档。尤其要注意并遵守您的修改和/或调整适用的任何安全信息、不同电气要求和规范标准。这些 TVD 架构中的全部或部分可能包含在您所在国家/地区或当地并不提供的推荐产品，也可能暗示或推荐了与您当地、地区或国家的电气或安全法规和/或规范标准相抵触的布线、产品、规程或控制器逻辑和/或功能。

⚠警告

合规性问题

确保采用的全部设备和设计的系统均符合并遵循所有适用的当地、地区和国家法规及标准。

如果不遵守这些说明，将会导致死亡、严重伤害或设备损坏。

TVD 架构的使用和应用要求具备自动控制系统的设计和编程方面的专业知识。只有设计师或集成人员才能清楚知道机器或过程安装和设置、运行及维护过程中可能出现的各种情况和因素，因此才能确定可以有效并正确使用的自动化和关联设备、控制逻辑和功能、相关安全装置及互锁设备。为特定应用选择自动化和控制设备及任何其他相关设备或软件时，还必须考虑任何适用的当地、地区或国家标准和/或法规。

1、通过回答问题，选择面板中的 TVD 架构查找器可帮助您查找经过测试、验证和归档的合适架构。

2、工作区提供经过测试、验证和归档的可用架构列表，在回答完 TVD 架构查找器中的问题后会突出显示最符合您要求的项目。

3、对于在工作区域选择的经过测试、验证和归档的架构，信息面板可提供与之相关的更多信息。它包含用于打开选定项目的打开按钮。

基于 TVD 架构创建新项目

要基于 TVD 架构创建新项目，请执行以下步骤：

步骤	操作
1	在 主页 选项卡中，选择创建新机器 → 使用 TVD 架构启动 。
2	回答选择面板中的问题并单击 建议的项目 按钮。 结果 ：在工作区中以绿色突出显示最符合您要求的 TVD架构。
3	在工作区域双击要作为新项目模板的 TVD 架构，或在工作区域选中它并单击信息面板中的 打开 按钮。 结果 ：将显示 将项目另存为 对话框，在该对话框中您可以浏览至目标文件夹并为新项目分配名称。
4	单击 保存 可将项目保存到所选文件夹。 SoMachine 会打开项目并选择 属性 选项卡 ，以分配更多信息。

4、使用应用程序启动

可以基于专用应用程序的示例项目创建新项目。

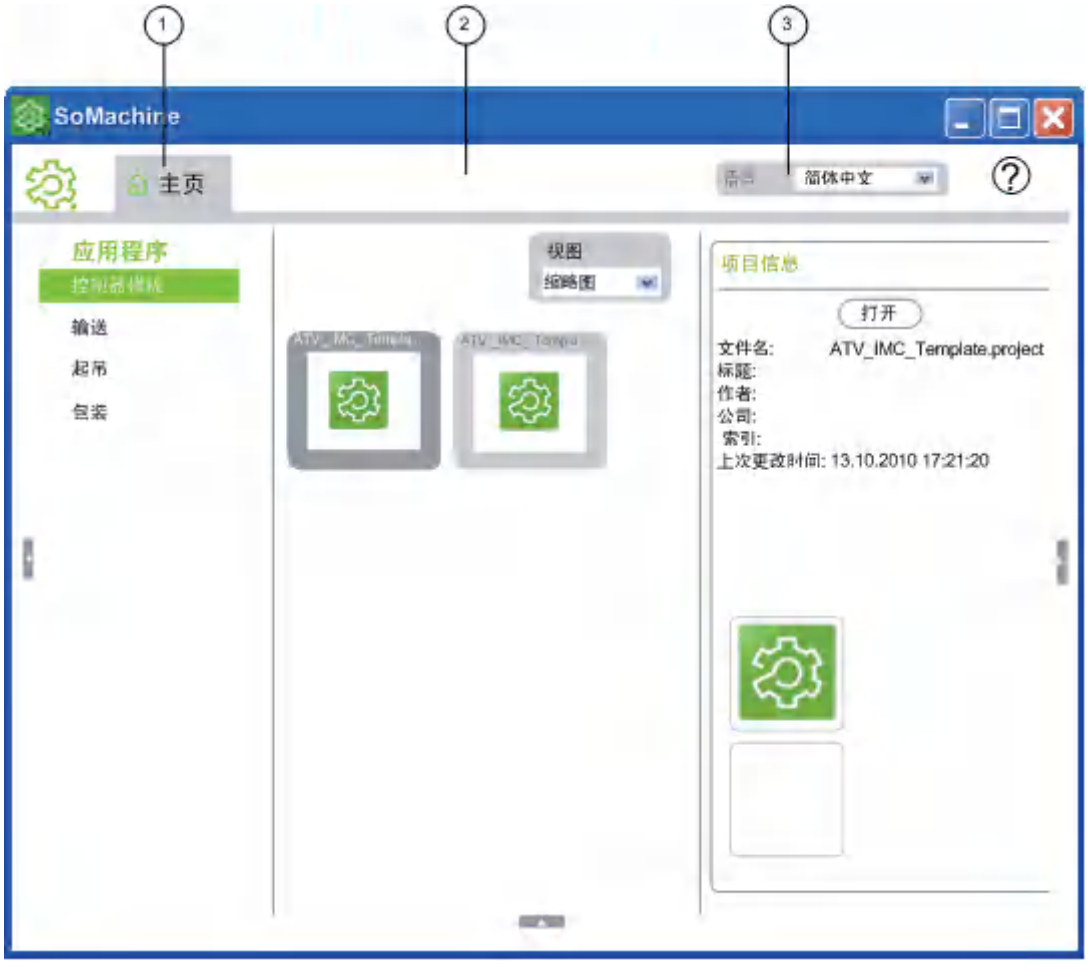
SoMachine 提供多种应用程序项目示例，这些示例是专用于以下应用的项目外壳，其中包括控制器、HMI、现场设备和应用程序功能：

•控制器模板

•输送

•起吊

•包装



1、选择面板列出由 SoMachine 为其提供应用程序项目的应用程序。

2、对于在选择面板中选定的应用程序，工作区会显示可用的应用程序项目。

3、对于在工作区中选定的应用程序项目，信息面板会提供与之相关的更多信息。它包含用于打开选定项目的**打开**按钮。

基于应用程序项目**创建新项目**

要基于应用程序项目创建新项目，请执行以下步骤：

步骤	操作
1	在主页选项卡中选择**创建新机器** → **使用应用程序启动**。
2	从应用程序列表为您的机器选择合适的应用程序类型。 结果：可用于所选应用程序的应用程序项目会显示在工作区中。
3	在工作区中双击要作为新项目模板的应用程序项目，或在工作区中选择该项目并单击信息面板中的打开按钮。 结果：将显示将**项目另存为**对话框，可让您浏览至目标文件夹并为新项目分配名称。
4	单击**保存**可将项目保存到所选文件夹。SoMachine 会打开项目并选择**属性**选项卡，以分配更多信息。

5、使用现有项目启动

可以基于已存在的项目创建新项目。在本地驱动器或连接的网络驱动器上浏览现有项目和/或库文件。

为防止覆盖现有项目/库，保存新项目/库时会自动显示另存为对话框。输入新名称可创建新项目/库文件。

此子任务与显示现有机器任务中的浏览现有项目子任务相同。有关更多信息，请参阅此子任务的描述。

5.2属性

一、属性选项卡的概述

属性选项卡

属性选项卡用于输入附加的项目信息。此处输入的文本信息和图形信息是可选的。对于在工作区中选定的项目，由于此信息始终显示在信息面板中，因此它将有助于今后识别各个项目，而无需打开项目。仅在打开项目之后才显示此选项卡。

属性选项卡提供以下任务：
•常规
•描述
•自定义信息

属性选项卡的元素

1、任务选择面板显示属性选项卡的任务。

2、工作区为文件信息和作者信息提供输入字段。

3、信息面板将显示在工作中区输入的信息，您可以用其中的另存为按钮将项目保存到其他位置或以其他名称保存。

要保存您的输入，请单击 SoMachine 图标并从常规功能菜单中选择保存。

二、常规任务的描述

常规任务
常规任务用于将项目文件保存到其他位置或其他名称下面，并提供用于添加作者信息的选项。它提供以下部分：
•文件信息
•作者信息
•项目信息（带另存为按钮）

文件信息
文件信息部分显示项目文件的名称以及保存它的文件夹。

作者信息
作者信息部分提供输入字段，用于输入您认为相关的可选信息。您的输入会显示在右侧的项目信息部分中。

注： 标题和作者文本框中不允许使用下列字符：
• (（左括号）
•)（右括号）
• ,（逗号）

要保存在属性选项卡中输入的信息，请单击 SoMachine 图标并从常规功能菜单选择保存。

更改项目名称或项目位置

要以其他名称保存项目或将项目保存到其他位置，请在项目信息面板中单击另存为按钮，浏览到首选文件夹并保存项目文件。

三、描述任务的说明

描述任务
描述任务用于将图像添加到项目。此可选的客户图像可帮助您识别项目。

工作区提供添加按钮以将需要的客户图像添加到 SoMachine 项目，并显示与配置选项卡中的设置相对应的配置视图。

项目的图像显示优先级

SoMachine 会检查每个项目的图像可用性，并按下述优先级顺序显示图像：

优先级	项目图像
高	客户图像
中	配置视图
低	标准 SoMachine 图标

客户图像

客户图像部分用于将图像添加到您的 SoMachine 项目中。

添加客户图像

要将新图像添加到项目，请执行以下步骤：

步骤	操作
1	单击 添加 按钮。 结果 ： 将客户图像添加到项目对话框打开。
2	浏览至目标文件夹，选择所需的图像并单击打开 。 结果 ： 客户图像显示在工作区和信息面板中。

注意： SoMachine 支持 JPG 图形。图形大小不应超过 1 MB，否则将缩减大小进行保存。

更改或删除客户图像

要替换项目的客户图像，请单击更改按钮并浏览至新图像的位置。

单击删除按钮删除图像。

配置视图

配置视图部分概述与在配置选项卡中的设置相对应的项目配置。

四、自定义信息任务的说明

自定义信息任务

自定义信息任务用于自定义您认为相关的项目信息。该工作区提供自定义信息的输入字段且

自定义信息字段

自定义信息字段是一些文本框，用于对项目进行个性化而提供的任何可选信息。您在此处定义的字段及其值在您激活相应的**信息**复选框后，会显示在右侧的信息面板中。

在自定义字段文本框中，输入要创建的新字段的名称。在值文本框中输入应在此字段中显示的内容。

对于要作为**项目信息**显示在右侧信息面板中的每行信息，选中**信息**列中的复选框。

附加的文档

单击添加附件按钮，将一个或多个文档附加到项目文件。

将显示一个对话框，您可以在其中浏览至目标文件夹。

将文件附加到项目后，**添加附件**部分会显示**重要信息**的复选框以及以下命令的图标：

- **查看附件**，该命令打开附加的文件。
- **将附件另存为文件**，该命令打开一个**另存为**对话框。
- **将附件从项目中删除**，该命令删除项目和附加的文件之间的链接。

重要信息

附件的文档部分为每个附加的文档提供重要信息复选框。对于您认为重要而需要读取的各个文档，请选中该复选框。然后，任务菜单会在自定义信息任务旁边显示一个小的 SoMachine 图标，以指示该任务包括重要信息。

5.3 配置

5.3.1添加控制器

配置页面

在SoMachine中配置完工程属性之后，点击“配置”功能选项卡，进入配置页面，如下图所示：

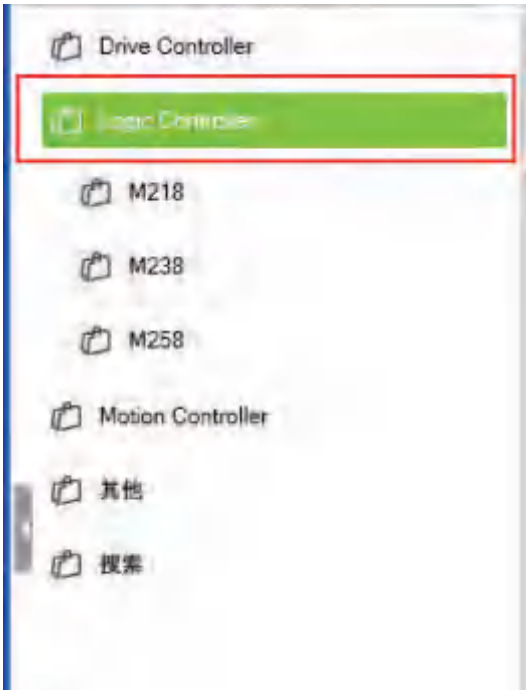
在页面的左边有五个功能选择文件夹，分别为：

1、Drive Controller（驱动控制器）
2、Logic Controller（逻辑控制器）
3、Motion Controller（运动控制器）
4、其他
5、搜索

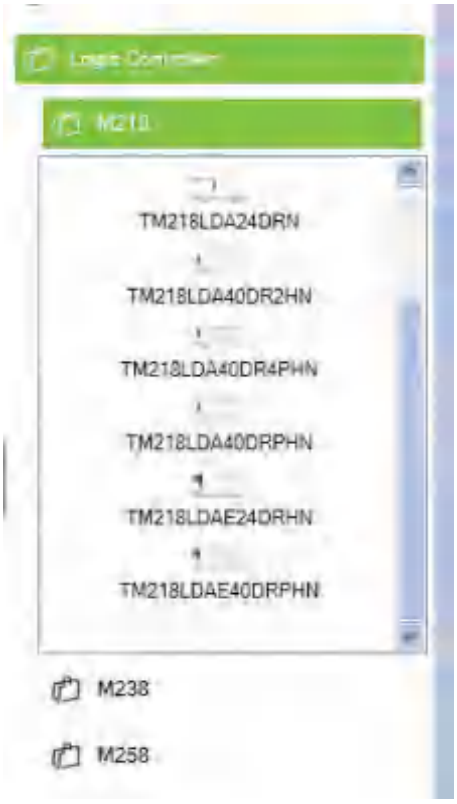
在每个文件夹下可以看到基于SoMachine平台的各个产品线相应的产品选项。在“Drive Controller”选项文件夹里面可以选择到“ATV-IMC”传动控制器。在“Logic Controller”选项文件夹里面可以选择到“M218”、“M238”、“M258”逻辑控制器。在“Motion Controller”选项文件夹里面可以选择到“LMC058”运动控制器。

下面我们以创建“Logic Controller”工程为例，添加逻辑控制器“M218”：、

1、双击“Logic Controller”文件夹图标，在下拉菜单中将可以看到“M218”“M238”“M258”三个系列PLC的选择文件夹。



2、 双击“M218”选项文件夹，可以看到在下拉菜单中列出了M218系列PLC的所有型号。共八个型号的PLC供选择（根据软件版本的不同，可以选择的控制器型号可能会有所差别）。



3、添加“TM218LDAE40DRPHN”到配置页面的中间区域，即配置区域进行控制器配置。用鼠标左键点击“TM218LDAE40DRPHN”，并拖拽到页面中间的配置区域，如下图所示，在配置区域将会出现“TM218LDAE40DRPHN”的图片

4、 用鼠标左键点击配置区域中间的“TM218LDAE40DRPHN”图片，在配置页面的右边区域将显示“TM218LDAE40DRPHN”的基本信息，包括控制器的“名称”、“供应商”、“版本”、“顺序号”、“说明”等信息。如上图右侧信息栏所示。

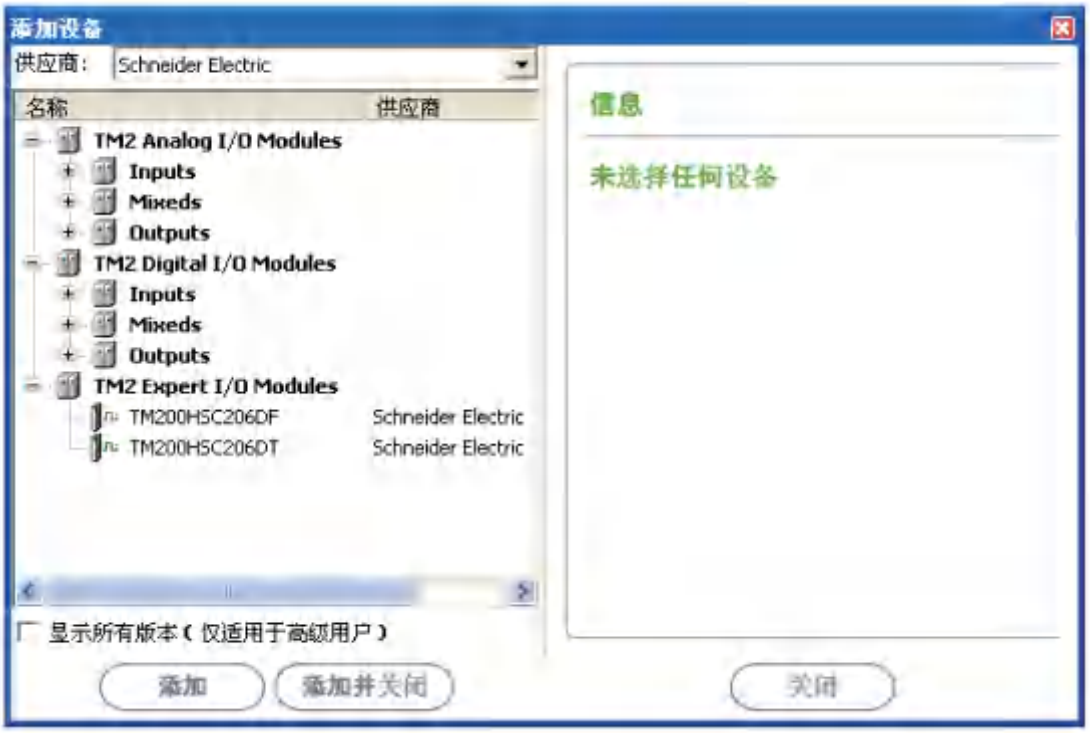
5.3.2添加扩展模块

扩展模块

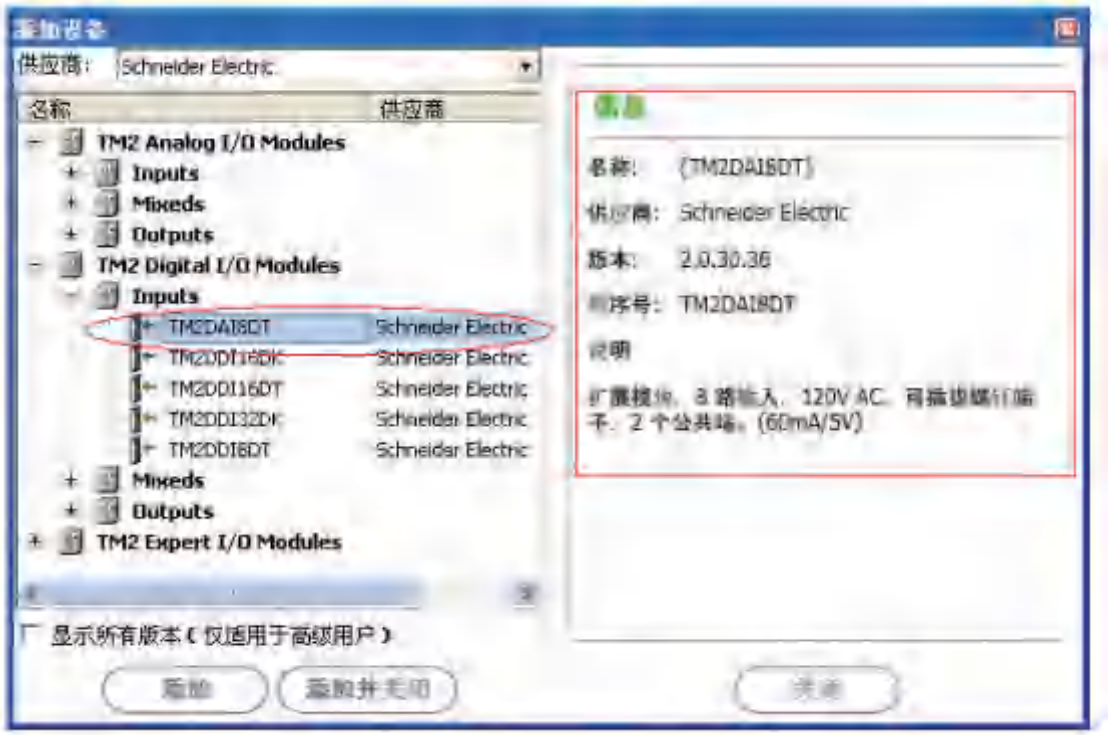
控制器支持外部扩展模块，可以通过配置页面进行添加和删除扩展模块，以及对扩展模块进行配置。

1、添加扩展模块，通过鼠标点击PLC控制器的本体图片的右侧选项“添加扩展”，如下图所示：

点击之后系统将会自动弹出扩展模块选择窗口。在窗口中包括“供应商”下拉菜单，此菜单可以进行模块供应商的选择，默认供应商为施耐德电气“Schneider Electric”。在窗口的中部为可供选择的扩展模块。M218支持扩展TM2系列的扩展模块，在列表中可以看到三大类模块：“TM2 Analog I/O Modules”、“TM2 Digital I/O Modules”、“TM2 Expert I/O Modules”。即：“模拟量模块”、“数字量模块”、“专家模块”三大类。在每个类别的扩展模块下面对应有该系列的具体模块。如下图所示：



2、点击选中要添加的扩展模块，在窗口的右面将显示该模块的基本信息。



3、点击页面左下角的“添加”选项按钮即可将所选扩展模块添加到本体PLC的后面，并可以继续选择添加扩展模块。如果点击“添加并关闭”选项按钮，将会把模块添加到PLC本体后面，并自动关闭扩展模块选择窗口，跳转到PLC配置页面。并且在本体PLC后面可以看见扩展的模块图片。

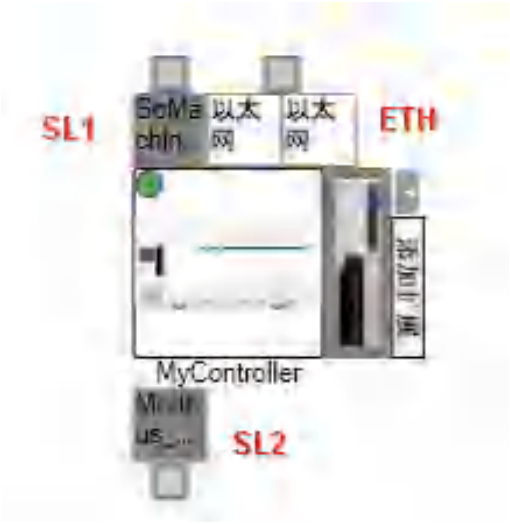
4、如果要删除已经添加的扩展模块，可以先通过鼠标左键选中需要删除的模块，然后点击鼠标的右键来进行删除操作。点击右键后将弹出一个对话框，类似于Windows操作系统的文件操作。

5.3.3参数配置

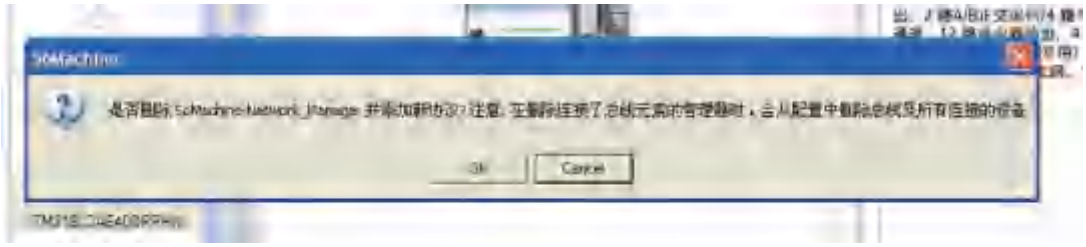
1、通讯端口设置

添加完控制器的本体和所需要的扩展模块之后，可以对所选择控制器的通讯端口进行配置，在此页面可以删除各个端口默认的通信协议设置，并可以添加自己实际需要的通讯协议。

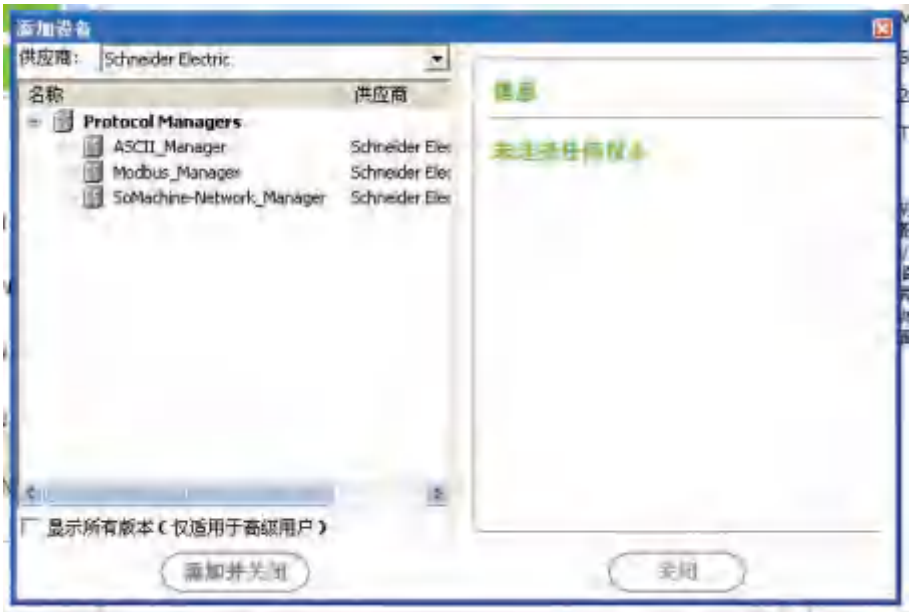
“TM218LDAE40DRPHN”控制器包含三个通讯端口，分别为串口1、串口2和以太网通讯端口。串口1（SL1）默认通讯协议管理器为“SoMachine Network Manager”；串口2（SL2）默认为“Modbus Manager”；以太网（ETH）端口不能进行修改，默认以太网通讯协议。



修改串口通讯协议设置可以通过鼠标左键点击对应的串口图标，系统将会弹出一个提示窗口，请您确认是否对端口的通信协议进行更改，提示框如下图所示：



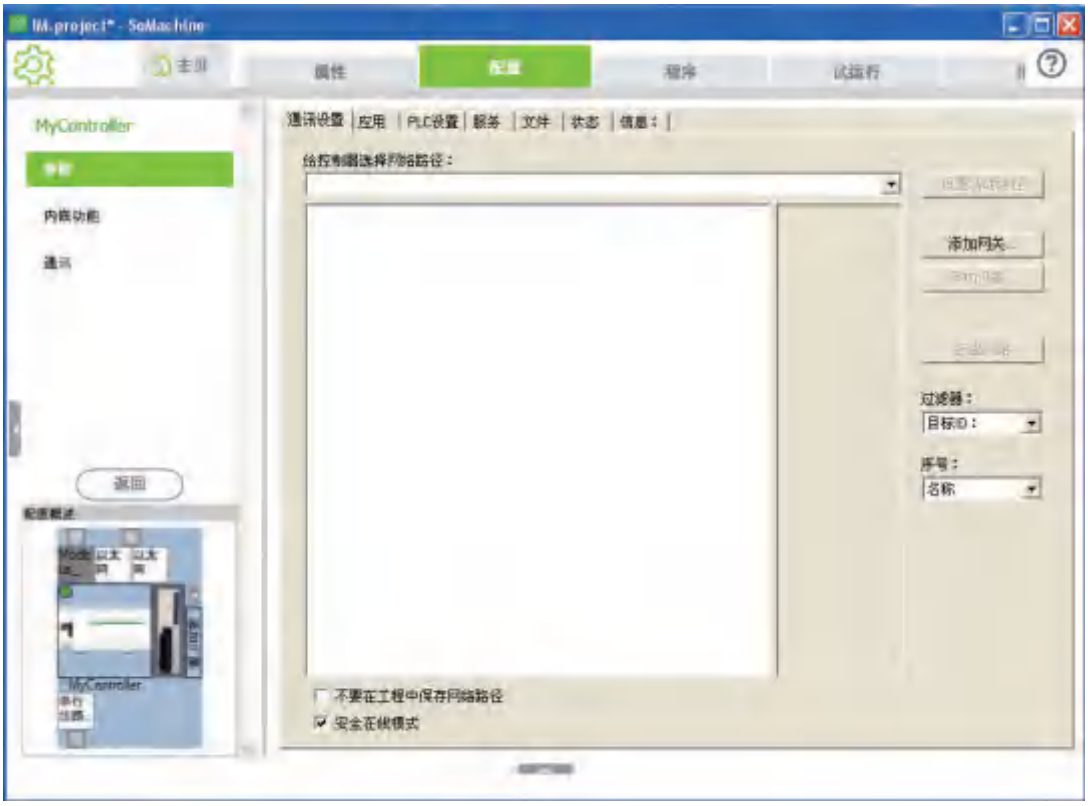
如果点击“OK”按钮，则会删除当前的通讯协议设置，并自动弹出通讯协议管理器配置窗口，配置窗口如下图所示，可以添加需要设置的通讯协议管理器，其中包括“ASCII_Manager”、“Modbus_Manager”、“SoMachine-Network_
Manager”三个通讯管理器，选择需要使用的管理器，并点击窗口下方的“添加并关闭窗口”跳出串口通信协议的选择界面：



注意：对应的端口只能添加各自支持的通讯协议管理器，例如在串口2上不能添加“SoMachine-Network_Manager”管理器，因为串口2不支持该协议。

2、参数配置页面

鼠标左键双击配置区的PLC配置图片，将打开如下所示的参数配置页面：



页面中包含如下三个选项框：

1、参数

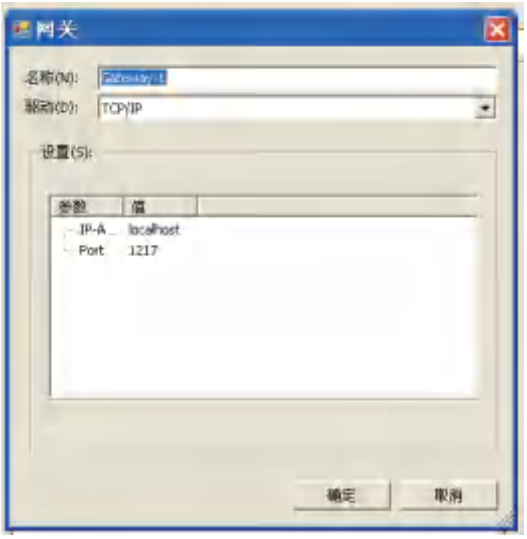
2、内嵌功能

3、通讯

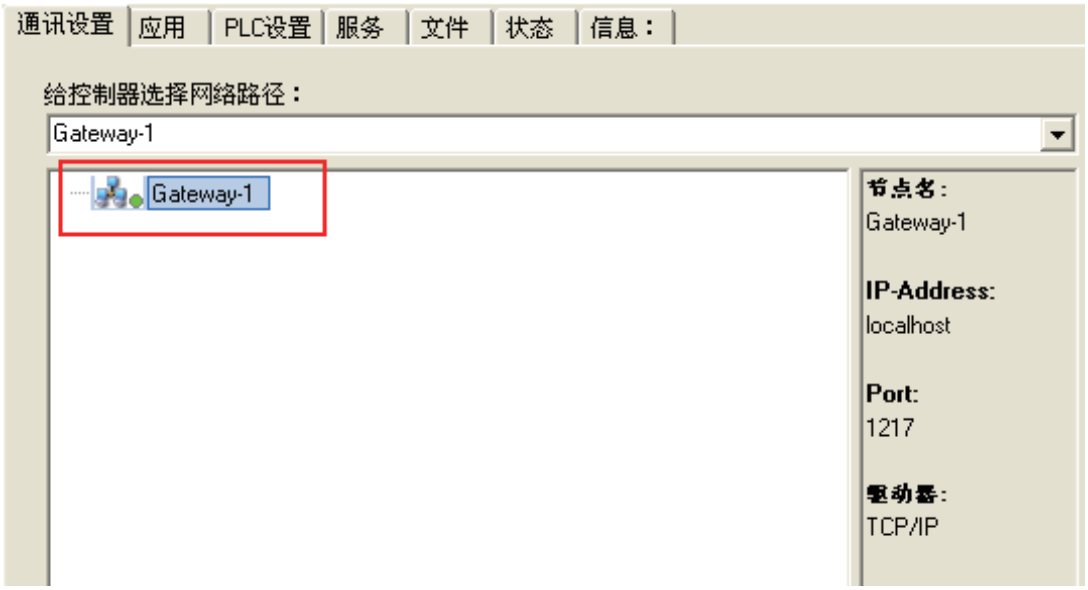
（1）参数

通讯设置：此功能用于设置活动路径，只有设置了活动路径才可以对PLC进行连接，用于程序下载、在线修改和在线监控等（只有在有外接PLC存在的情况下才可以设置活动路径）。

点解页面右边功能栏里的“添加网关”按钮，将会弹出如下图所示提示框：

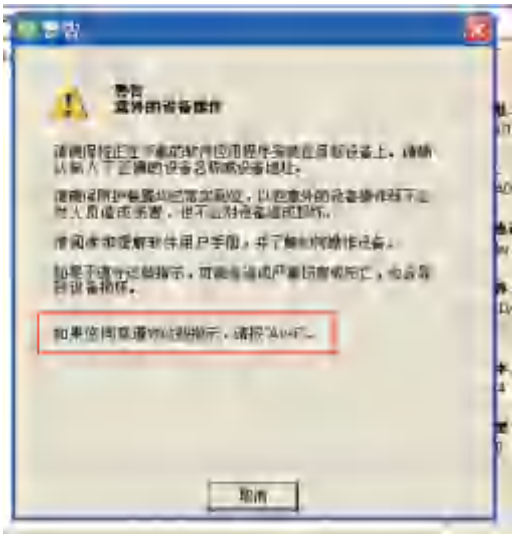


默认的网关名称为“Gateway-1”，驱动为“TCP/IP”，点击确定即可，其他参数可以均为默认值。可以看到在软件的中间区域出现名称为“Gateway-1”的网络图标，如下图所示：



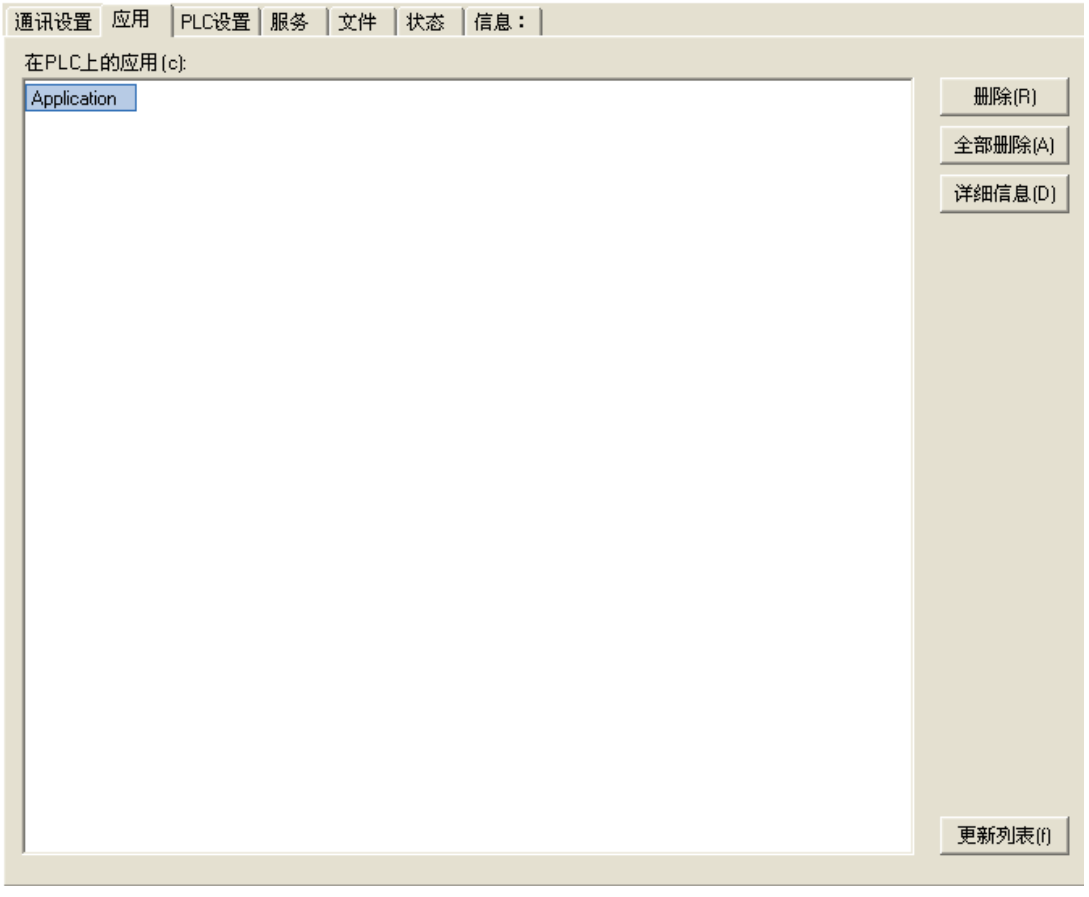
鼠标左键单击选中此图标。通过鼠标左键双击此图标，或者由鼠标右键弹出下拉菜单中选择“扫描网络”以及通过点击右边功能栏上“扫描网络”按钮均可以进行网络扫描。（只有在电脑通过USB电缆连接了PLC的情况下才可以通过扫描网络进行活动路径设置）。

执行完扫描网络功能后，可以看到连接到此台电脑的PLC设备，其中包括上图中所示的PLC名称：TM218，节点地址：0000.2001，目标ID：16#101A0507，目标供应商：SchneiderElectric，目标名称：TM218LDAE40DRPHN，目标版本号：2.0.30.44（即Firmware版本号），目标类型：16#1000。



选中设备节点可以由鼠标右键下拉菜单中选择“设置活动路径”或者点击界面右边的功能按钮“设置活动路径”进行路径设置。可以看到系统会弹出一个如下图所示警告提示框，按照提示信息操作，通过键盘按“ALT+F”关闭即可（注意：不要点击取消按钮）。这样活动路径就设置好了，设置之后就可以进行在线连接设备了。

应用：如果已经设置了活动路径，可以通过点击此界面中的“更新列表”按钮查看当前控制器中的应用程序。通过“删除”按钮删除设备中已经存在的应用程序。通过“详细信息”按钮查看当前设备中应用程序的相关信息，包括创建日期等应用信息。（注：只有实际连接了外部控制器并设置了活动路径的情况下才能刷新的到应用信息）。下图中可以看到目前设备中的应用程序为“Application”。

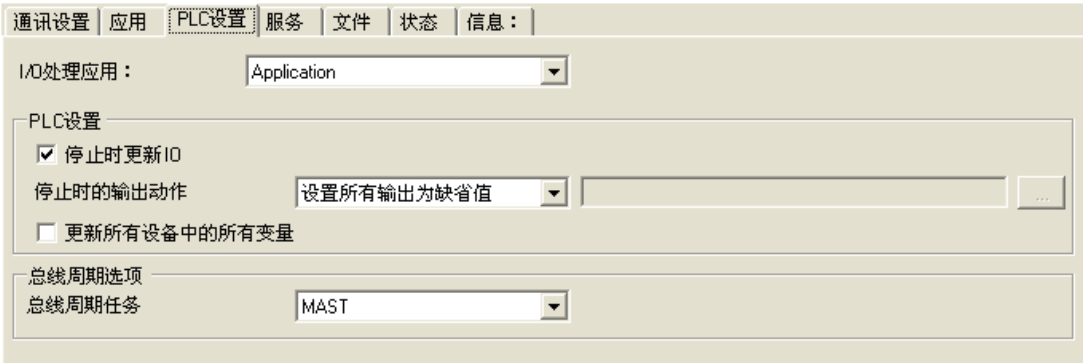


PLC设置：

I/O处理应用，用于选择处理I/O的应用程序，控制器默认为“Application”，不需要更改为其他。

PLC设置，用于选择设备停止时是否进行I/O更新，已经停止时“设置所有输出为缺省值”或者“保持当前值”。

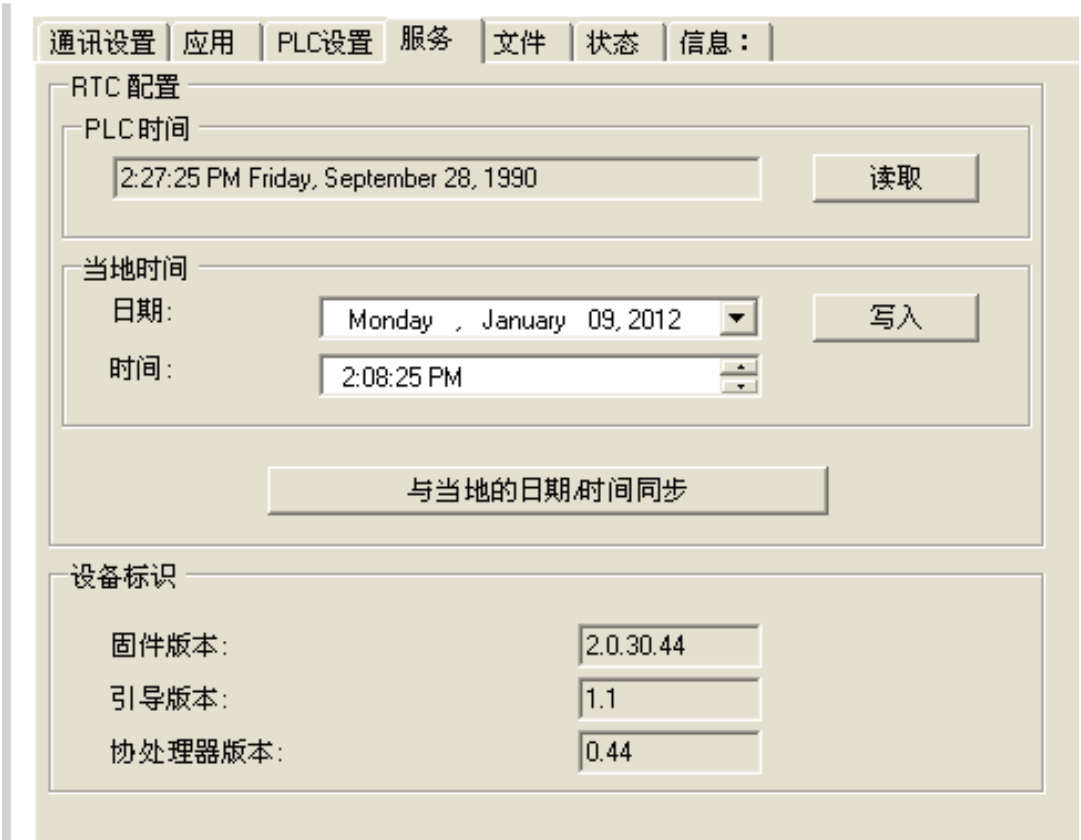
总线周期选项，可以将程序中的任务选为总线周期任务，系统默认为“MAST”任务。



服务：

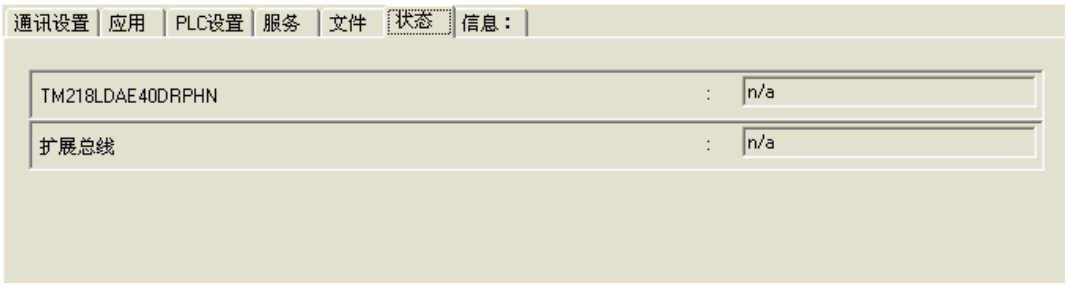
通过此页面可以配置RTC（实时时钟）时间，可以读取当前设备时间，也可以写入时间或者与当前电脑时间同步。（注意：只有在有外接设备并已经设置了活动路径的在线情况下才可以读取和写入PLC时间）。

设别标识：提供设备的一些基本信息。



状态：

显示总线的状态，在线的情况下将能够显示总线是否故障。



信息：

显示所选择控制器的基本信息，以及缩略图，使有一个直观的印象。

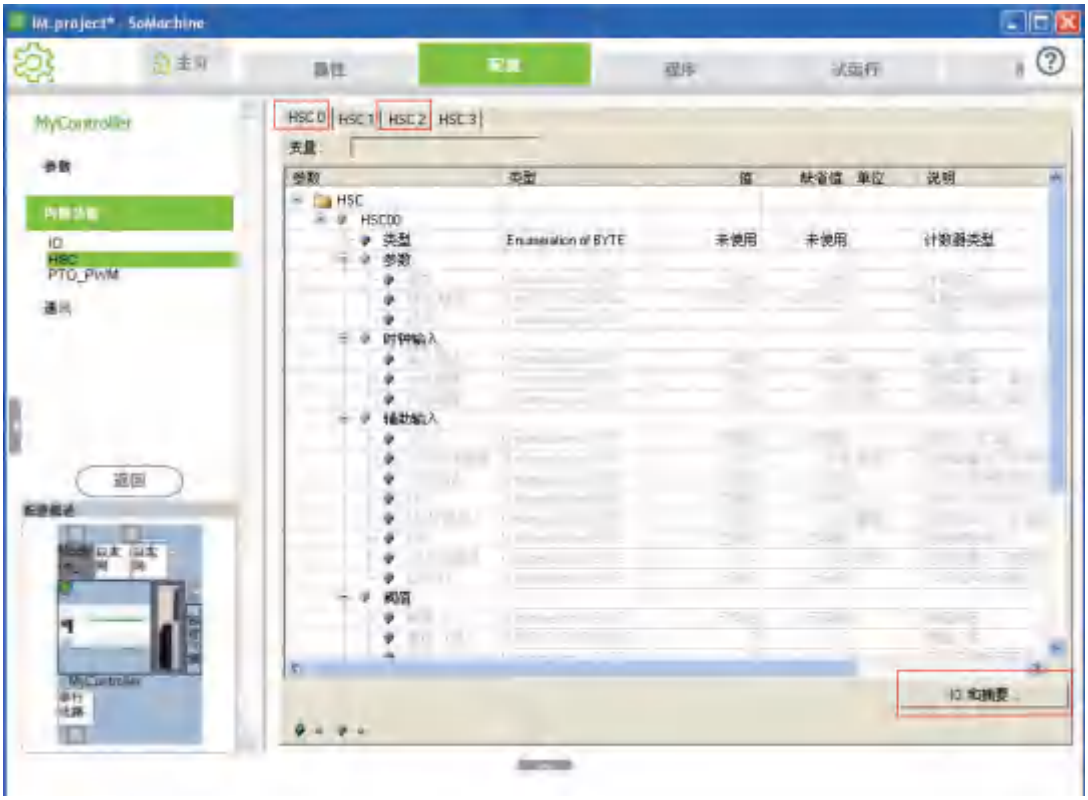
（2）内嵌功能

点击“内嵌功能”选项框，可以看到所选PLC支持的内嵌功能，包括IO（本体输入输出）、HSC（高速计数功能）、PTO_PWM（脉冲输出）。

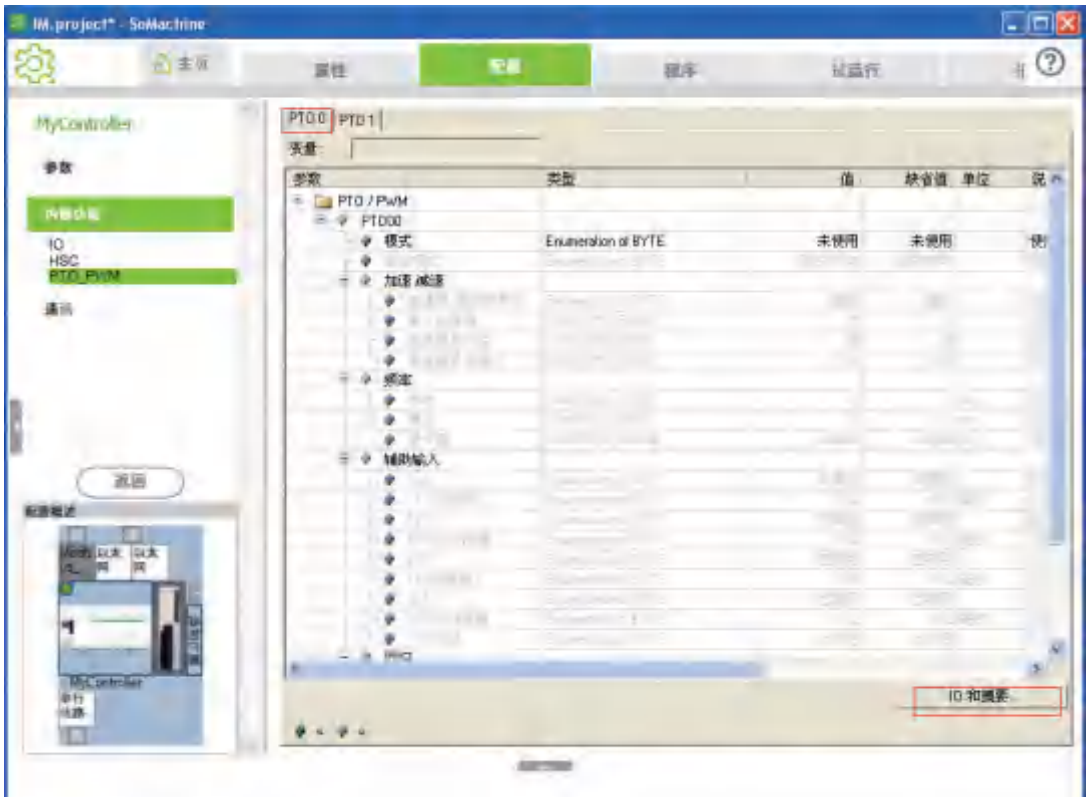
注意：根据PLC类型的不同，可能有些内嵌功能不支持，详细说明请参考所选PLC的参考手册。

IO参数界面如下图所示，可以通过此界面对PLC本体IO点进行设置：

通过HSC配置界面可以设置高速计数功能的相关参数，其中包括高速计数功能的启用与否和启用后的参数设置。如果HSC包含支持多路高速计数功能，可以通过页面上方的选择框“HSC*”进行选择。查看已经分配的IO通道情况，可以通过点击右下角的“IO和摘要”按钮进行查看。高速计数功能配置页面如下图所示：



PTO_PWM功能配置页面可以对支持脉冲输出功能的PLC进行脉冲输出功能相关参数的设置，设置内容包括是否启用脉冲输出功能以及启用后的一些功能参数设置。根据支持输出通道数的不同，可以通过页面上方的选择框“PTO*”进行选择。查看已经配置好的IO通道情况，可以点击右下角的“IO和摘要”按钮进行查看。



（3）通讯

在此界面可以对PLC的串口和以太网口（如果支持以太网）进行通讯参数设置。点击“参数”选项框之后，下拉菜单将显示当前所选择的PLC本体支持的通讯端口（实例中包含：串口1、串口2、以太网）。在串口1和串口2的选项下面还分别包括“物理设置”和“协议设置”两个选项。

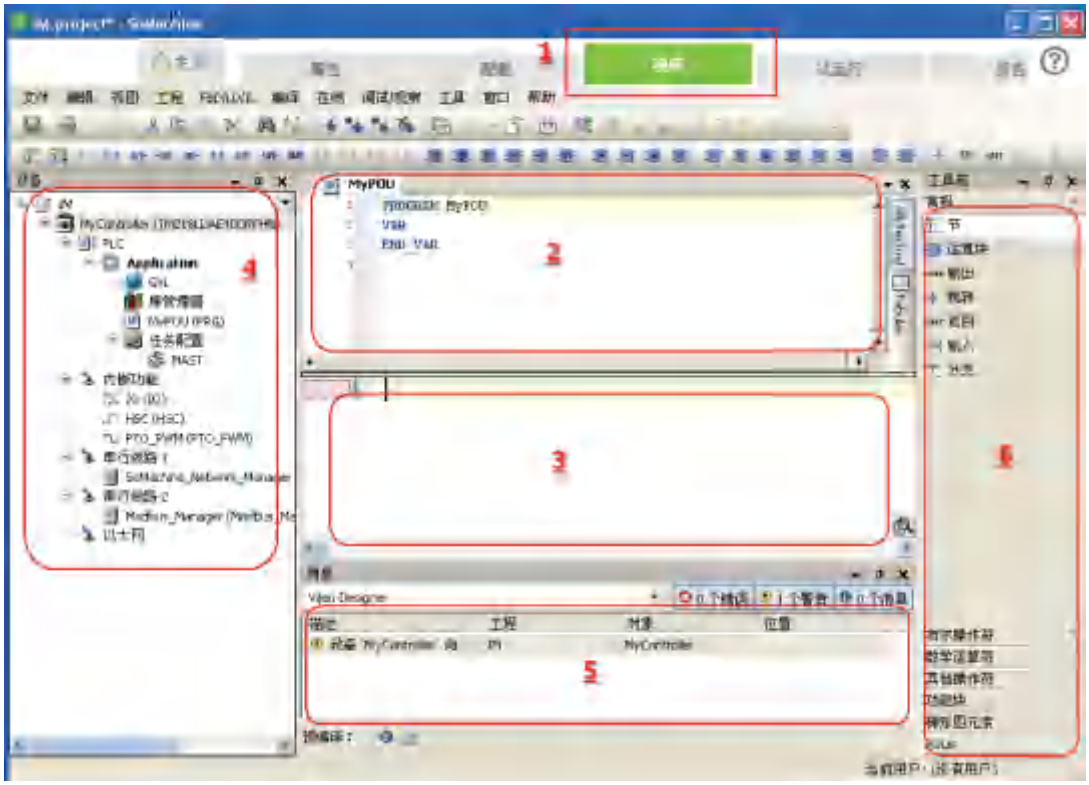
“物理设置”包括：通讯波特率、校验位、数据位、停止位、物理介质选择（是否支持RS232和RS485通讯，视PLC具体型号而定）。在物理设置界面可以选择是否启用内置“极化电阻”功能。

“协议设置”包括：传输模式（RTU或者ASCII）、寻址（主站模式或从站模式）、地址（如果做从站，站点号设置）、帧间时间。

5.4编程

5.4.1窗口介绍

在选择了控制器并进行了基本配置之后，就可以开始进行编写程序了。下图即为“程序”界面：



1、SoMachine软件上面的“程序”选项卡，对控制器进行编程。

2、变量声明区，对程序中用到的变量进行声明，在此区域声明的变量均为局部变量，只能在当前POU（程序组织单元）内使用。全局变量要创建在“GVL”（全局变量列表）里面，在变量创建章节将会进行详细的介绍。

3、程序编辑区，在此区域进行程序的编写，根据所选择的编程语言的不同，此区域的表现形式也会有所变化，图片中所示为选择梯形图编程语言下的编程区。

4、设备区，此区域将设备的所有功能以列表的形式进行了表示，可以在此区域对设备的功能进行配置和修改（功能等同于上一章节的配置功能，可以通过鼠标的左右键的点击来进行相应操作）。

5、消息区，启动应用、活动路径、编译信息等各种提示信息均显示在此区域。

6、工具箱，提供程序编制的基本控件和基本编程元素（根据所选编程语言的不同，变现形式有所不同）。

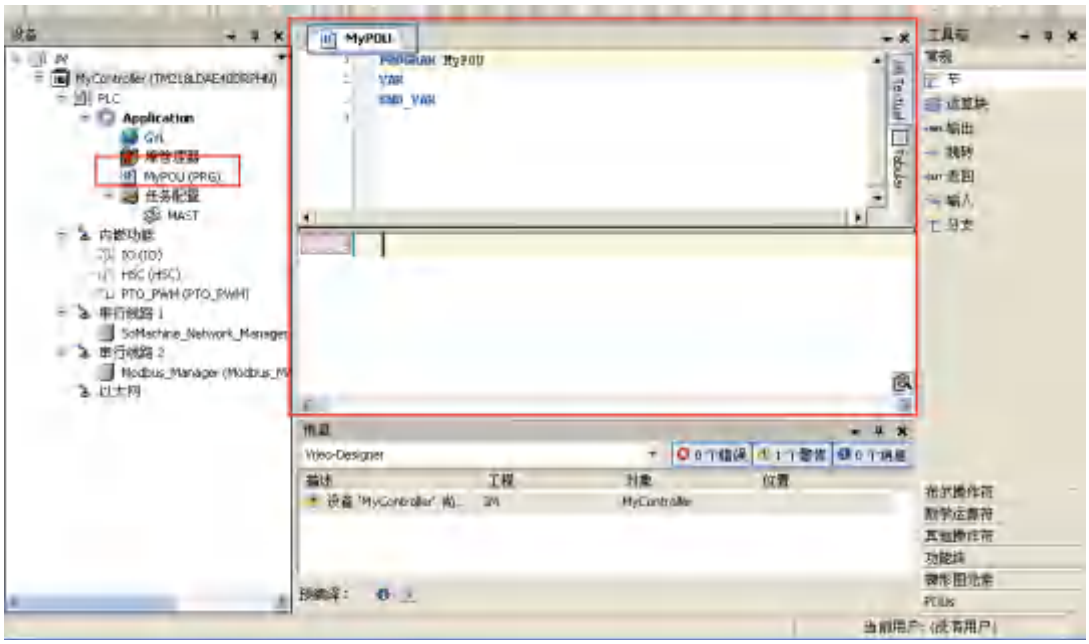
注：此页面的布局以及各种基本操作类似于所有基于Windows操作系统的软件的基本操作，便于用户灵活的使用和排列页面布局。

5.4.2创建POU

POU是Program Organization Unit 的缩写，意思是“程序组织单元”，它是创建PLC程序时必须用到的对象。

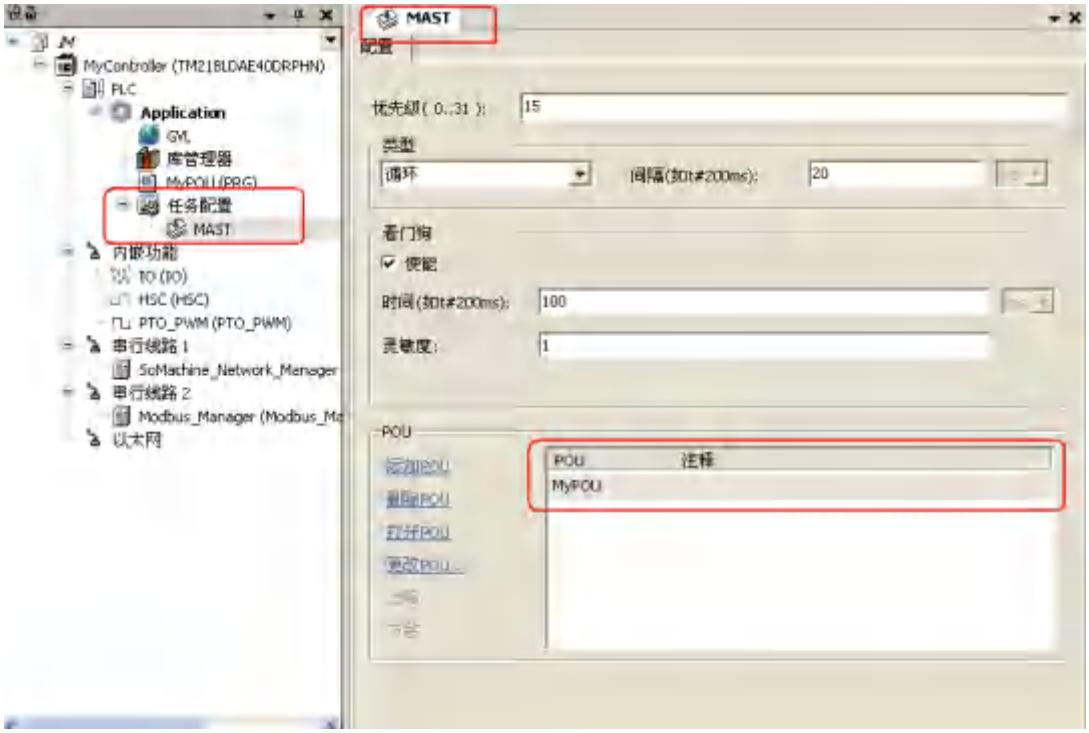
一个程序组织单元对象（POU），大体上来说可以认为就是一个编程单元，不同的程序段可以分别编辑在不同的POU内，在“任务”可以调用各自需要执行的POU，来进行程序的执行。如果POU不被添加到“任务”中或者不被其他“POU”在整个程序结构中是不会被执行的。所以一定要将POU添加到“任务”当中，以便能够被编译以及执行。

添加M218系列的可编程控制器后，在“程序”编辑界面会自动生成一个默认的POU，名字为“MyPOU”，编程语言默认为梯形图。同时，“MyPOU”也被默认的添加到了“MAST”任务里面，如下图所示：



鼠标双击“MAST”任务，可以看到在“MAST”任务管理器里面已经自动添加了“MyPOU”，如果不要要使用默认生成的POU名字以及编程语言等，可以将系统自动生成的POU进行删除，并需要将任务里面的POU进行移除操作。POU的添加与删除等操作可以参考任务配置章节。

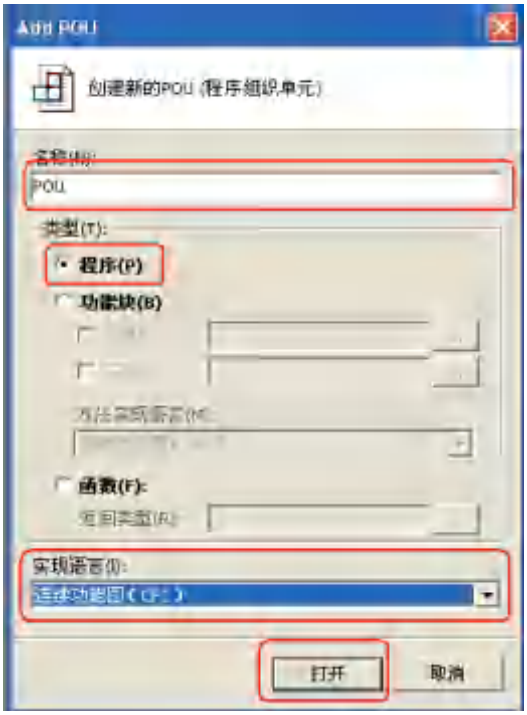
注：POU的名称不支持中文，所以不能将POU的名称设置为中文字符。



创建多个POU

根据程序的需要，可以自行添加多个POU。鼠标左键点击选中“设备”列表中的“Application”，在下拉菜单中将鼠标放置在“添加对象”选项上，选择列表里面的“POU”选项。系统将自动弹出POU创建窗口。

系统弹出“Add POU”窗口，可以对POU的名字进行更改（不能使用中文字符）。在类型选项里面可以决定是用来创建“程序”“功能块”“函数”。



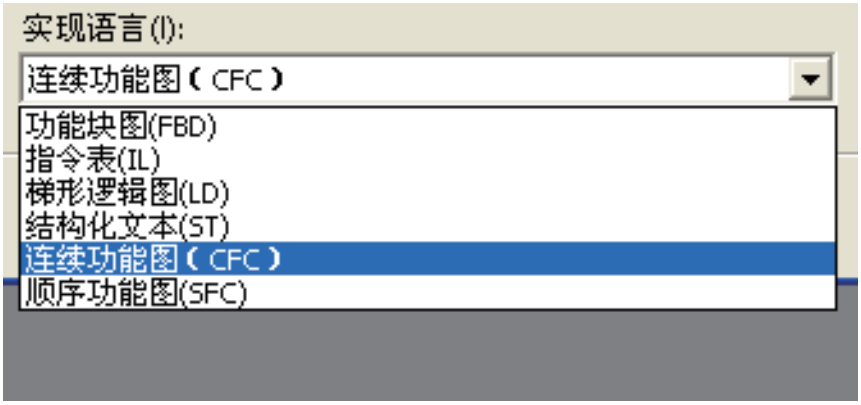
程序：它在操作期间返回一个或多个值。程序上次运行的所有值都保留到程序的下一次运行。可由另一个 POU 调用。

功能块：在程序处理期间，它提供一个或多个值。与功能不同，输出变量值和必要的内部变量值将从执行功能块后一直持续到下次执行功能块。因此，使用相同的参数（输入参数）调用功能块无需始终产生相同的输出值。

功能：在对其进行处理时，它只能产生单个数据元素（可以包含多个元素，如字段或结构）。在表达式中输入某个运算符可以在文本语言中调用功能。

实现语言：可以选择所提供的六种编程语言进行编写，在实现语言下拉菜单中可以看到六种语言列表。选择完所使用的编程语言之后，点击“打开”功能按钮，即可创建一个POU。

编程语言选择，提供的六种编程语言，各种语言的介绍请参考编程语言章节的相关内容



可以创建多个POU，如下图所示创建了六个不同语言组成的POU

注：建议在编写程序的时候整个工程的程序分散到多个POU中，这样程序的可读性会比较好，有利于程序的编写、修改等操作。

5.4.3变量声明

SoMachine软件支持变量名编程方式，灵活易用，有助于程序的编写，本章将介绍如何创建变量，以及对变量进行声明。

局部变量：局部变量只在声明的POU内部调用，在其他POU内调用时不生效。

全局变量（GVL）：全局变量声明在GVL里面，变量可以在应用程序的所有变量POU内调用。

保留变量（Retain）：局部变量和全局变量均可以设置为保留变量，保留变量区根据所选择控制器类型的不同，数据区容量会有所差别，具体保留区大小可以参照控制器样本手册或咨询施耐德电气技术人员。

持久变量（Persistent）：持久变量为全局变量，而且必须声明在“持续变量”（PersistentVars）里面，保留变量区根据所选择控制器类型的不同，数据区容量会有所差别，具体保留区大小可以参照控制器样本手册或咨询施耐德电气技术人员。

保留变量和持久变量等同于传统意义上的断电保持型变量，下表列出了普通变量、保留变量和持久变量（保留持久）变量在对控制器进行不同操作时保持的状态。

操作	普通变量	保留变量 RETAIN	持久变量（ PERSISTENT ） 和 保留持久变量（ RETAIN -PERSISTENT ）
对应用程序进行在线修改	X	X	X
停止	X	X	X
电源重置	–	X	X
热复位	–	X	X
冷复位	–	–	X
初始值复位	–	–	-
应用程序下载	–	–	X
X 保持值 – 重新初始化值			

局部变量声明:

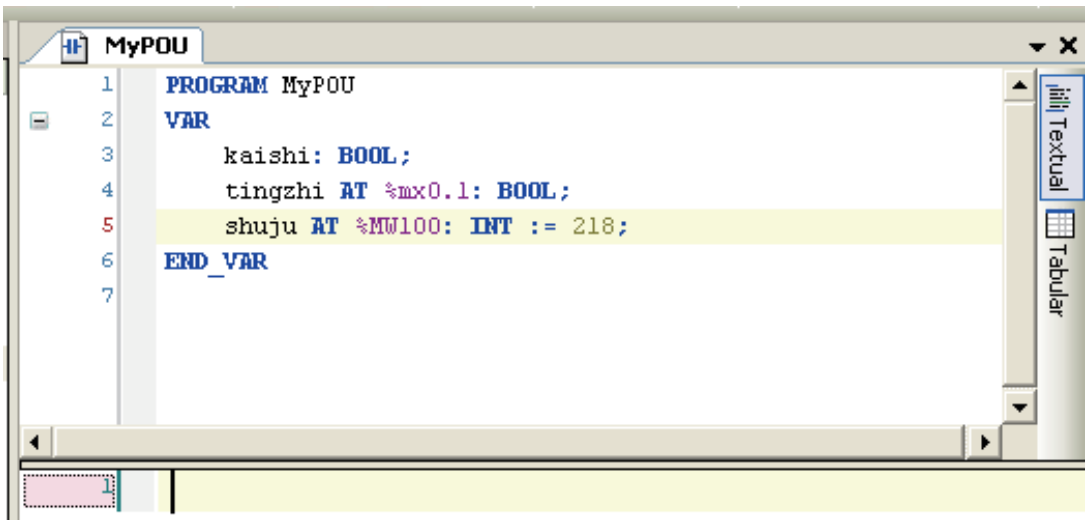
（1）直接在POU的变量声明区按照变量声明的语法进行编写，此种方法适合于已经知道所需变量并熟悉变量声明语法的客户进行使用。

语法:
<标识符> {AT<地址>}:<类型>{:=<初始化值>};其中大括号{}中为可选部分。

例如:

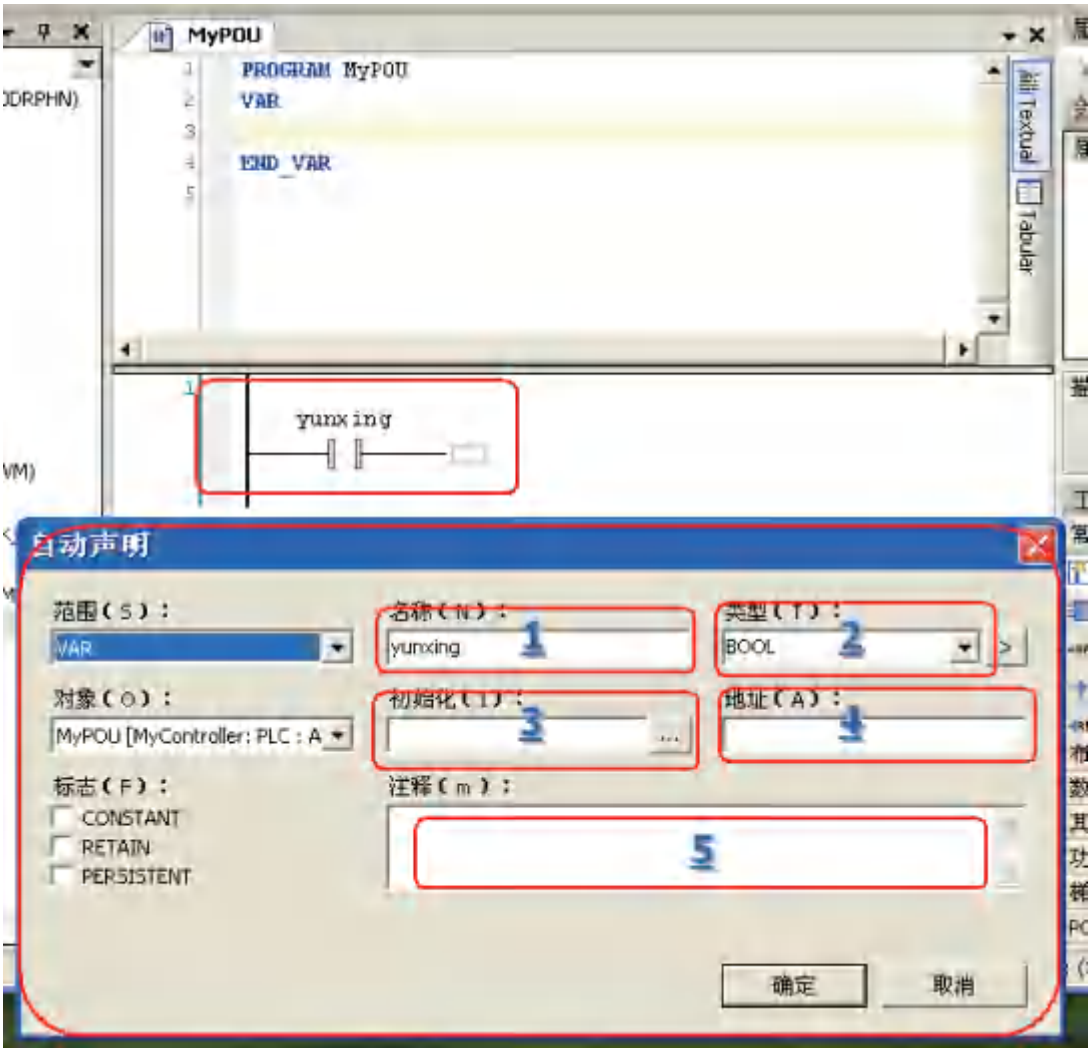
```
kaishi: BOOL;
tingzhi AT %MX0.1: BOOL;
shuju AT %MW100: INT := 218;
```

其中：kaishi、tingzhi、shuju为三个标示符，即变量。Kaishi为BOOL类型变量；tingzhi为BOOL类型变量并且被映射到了实际地址%MX0.1；shuju为INT类型变量，映射到实际地址%MW100，并且被赋予了初始值218。三个变量声明在POU的变量声明区，三个变量即可以在“MyPOU”中进行使用，截图如下所示。



（2）在程序编辑区直接输入变量，利用自动声明窗口进行声明，此种方法适合大多数用户使用，并且简单方便，不容易出现错误。

例如，在程序编辑区添加触点，在触点的“？？？”处写入需要的变量名“yunxing”，将鼠标点击到编程区的其他任意空白处即可自动弹出变量“自动声明”窗口，可以在窗口中填入或选择需要的内容。



1：声明的变量名

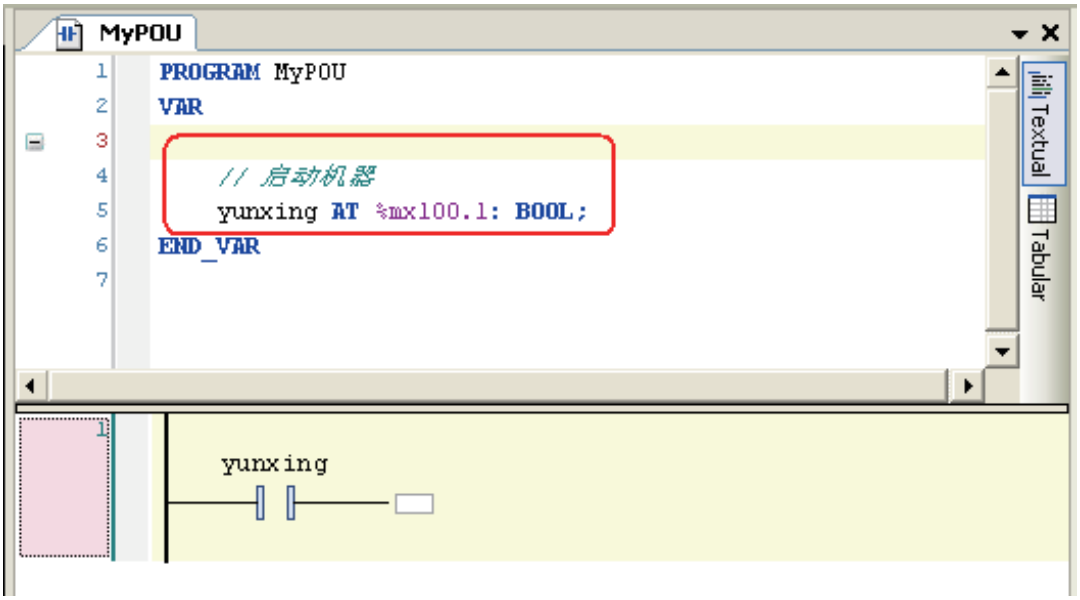
2：变量对应的数据类型，可以通过下拉菜单进行选择（必须选择一种数据类型）

3：初始化值，即初值（可以选择填与不填）

4：地址，即映射的实际地址（可以选择填与不填）

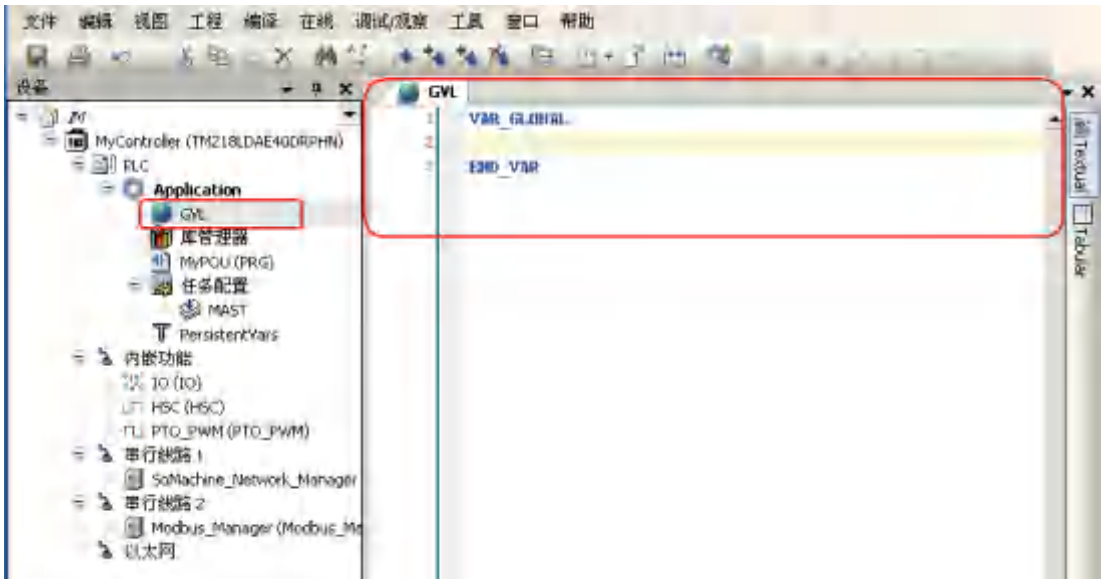
5：注释，即对此变量名的注释解释，可以使用中文注释（可以选择填与不填）

声明完成之后，变量的信息会自动添加到POU的变量声明区，如下图所示：

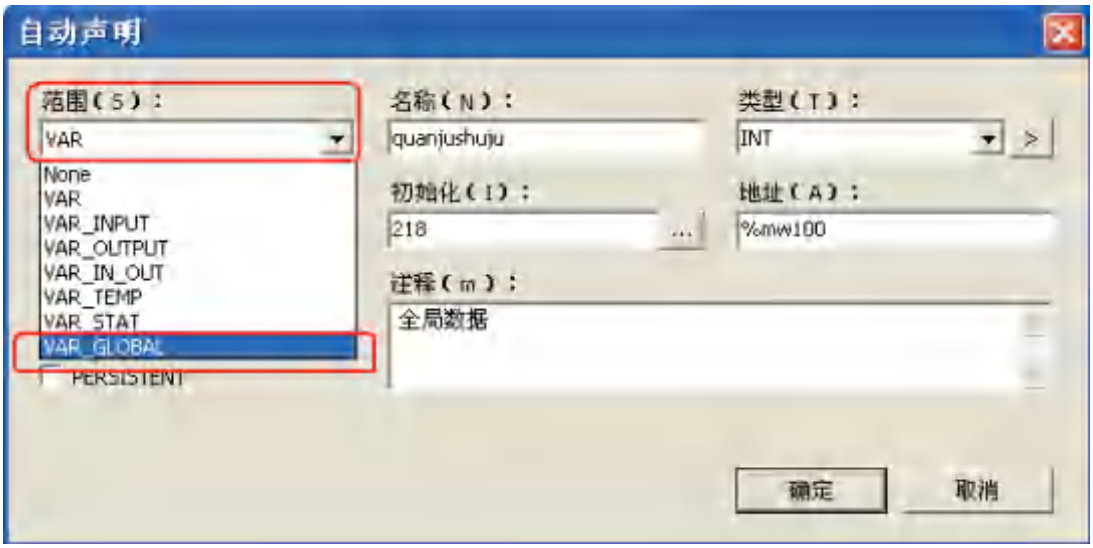


全局变量声明：

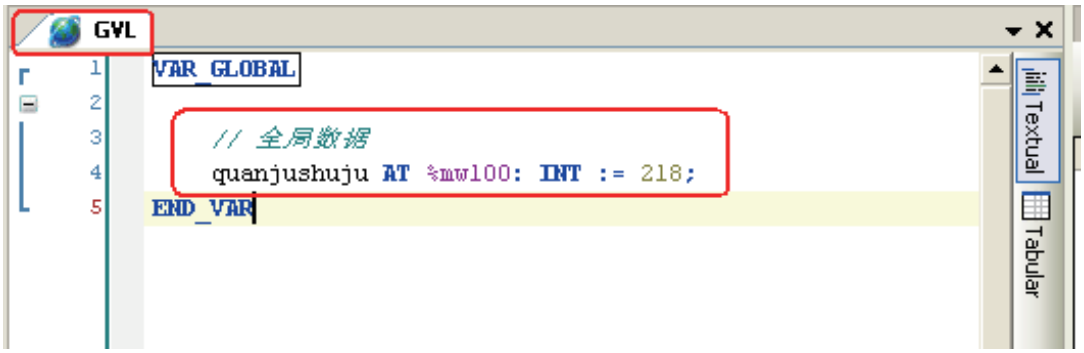
（1）与局部变量声明方式类似，可以直接在GVL里面按照变量声明的语法格式进行声明，与局部变量的区别在于，全局变量能够在功能内的所有POU内被调用。双击软件左面设备栏“Application”菜单里面的“GVL”，可以看到会软件的中部打开一个“GVL”变量声明窗口，所有的全局变量都可以声明在这个声明区里面，在程序的各个POU里面都可以调用声明在此声明区里的变量，声明的具体方式可以参照局部变量声明方式。



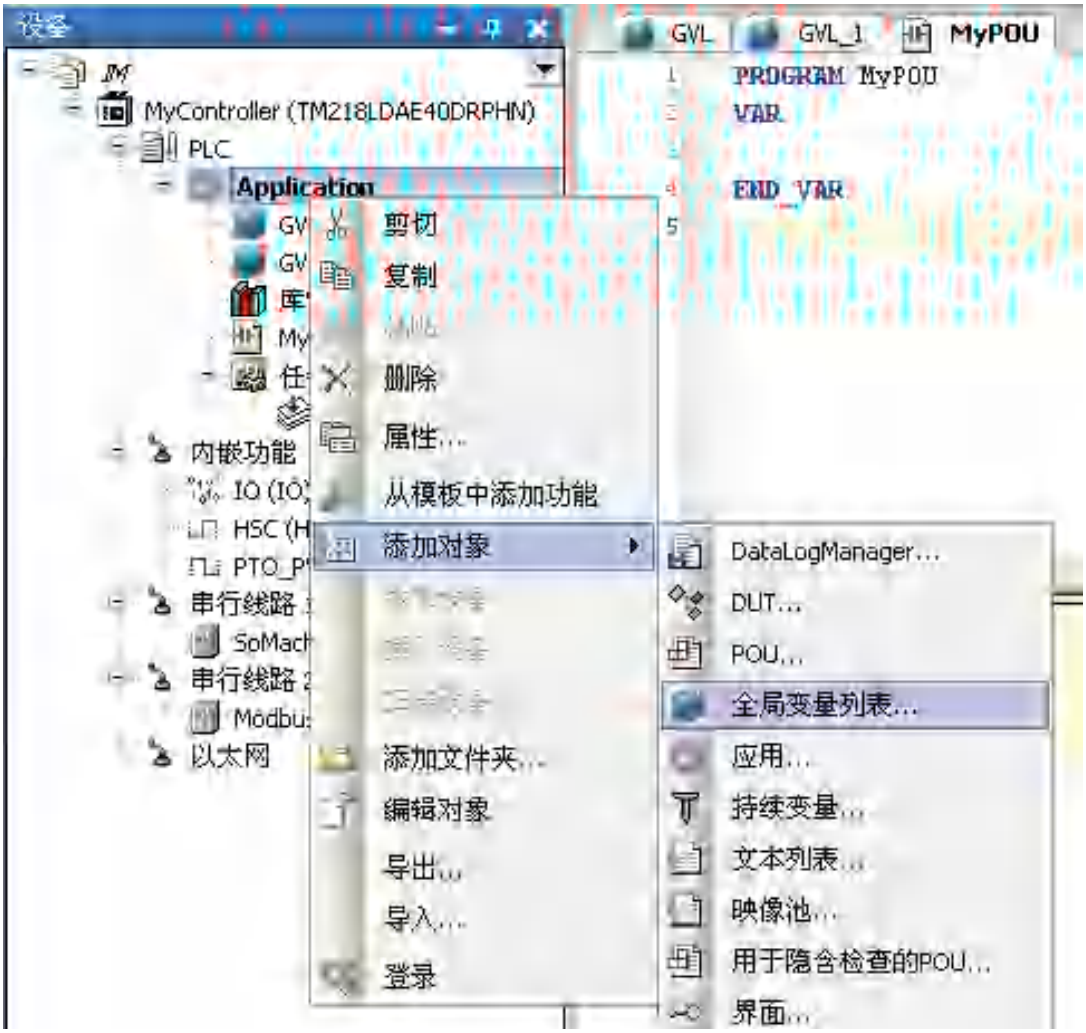
（2）类似于局部变量声明方式，可以在编程区创建变量的同时利用自动声明窗口进行全局变量的创建，需要注意的是需要选择为全局变量的格式，具体设置方法见下图所示，只需要在“自动声明”弹出框左边的“范围”选项栏的下拉菜单里面选择为“VAR_GLOBAL”（全局变量）即可。



点击确定之后可以看到在“GVL”声明区自动添加了声明的变量“quanjushuju”，如下图所示，全局变量“quanjushuju”数据类型为INT，映射到实际地址“%MW100”，初始值为218。此数据即可在所有的POU中被调用。



系统默认会生成一个“GVL”对象，如果变量数据比较多的情况下，我们还可以通过添加多个全局变量列表的形式来对全局数据进行分类，这样就增加了程序的可读性和灵活性。点击选中“Application”并从下拉列表中选择“添加对象”功能，通过鼠标选择“全局变量列表”即可，系统将自动弹出一个添加全局列表对话框。

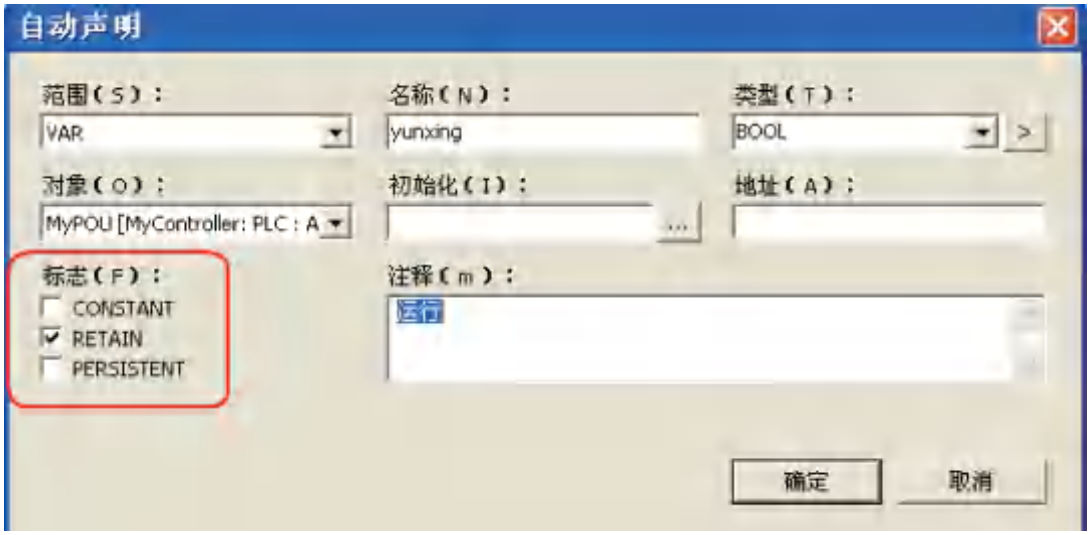


名称即为自定义的全局变量列表名称（不支持中文），点击打开之后即可以自动添加一个新的全局变量列表，有助于变量的管理。

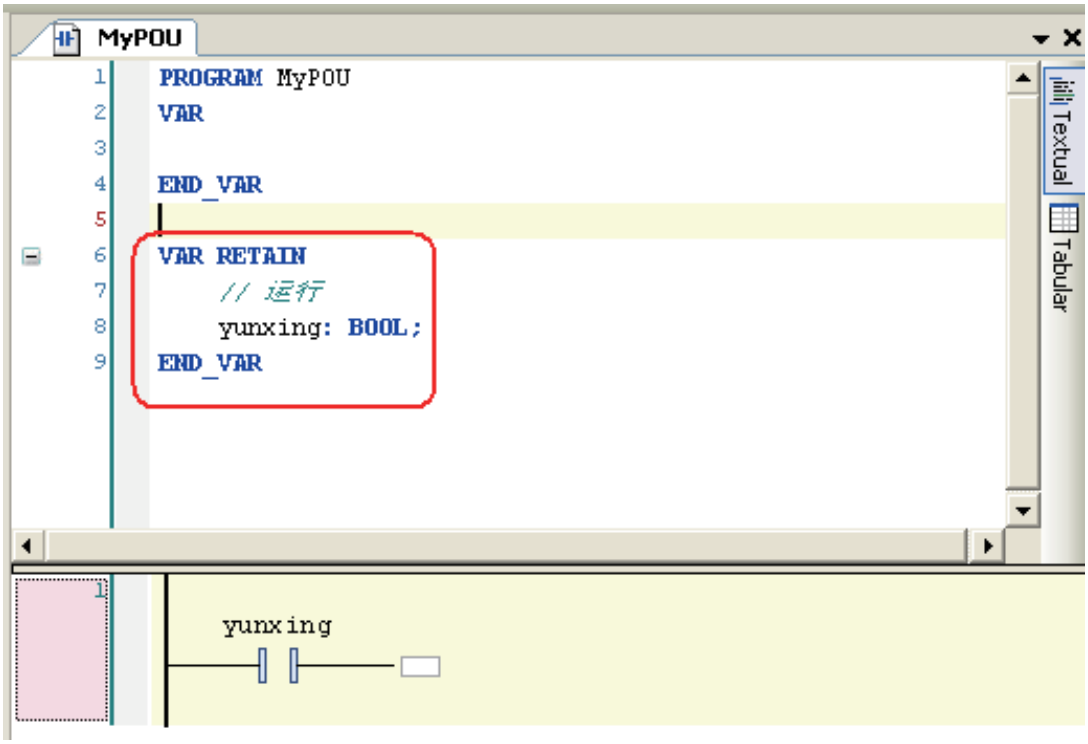
在自动声明时，选择多个全局变量列表里面的一个时，通过自动变量声明窗口的“对象”菜单进行选择，如下图所示，可以看到在“对象”的下拉菜单里面有三个已填加的变量列表供选择，选中其中一个即可将所声明的变量加入到对应的全局变量列表里。

保留变量声明：

保留变量可以设置为局部变量也可以设置为全局变量，基本设置方法与本章上面介绍的声明方式相同，如果需要将变量声明为保留型变量，需要在自动声明窗口中的“标志”选项中将“Retain”勾选上即可。

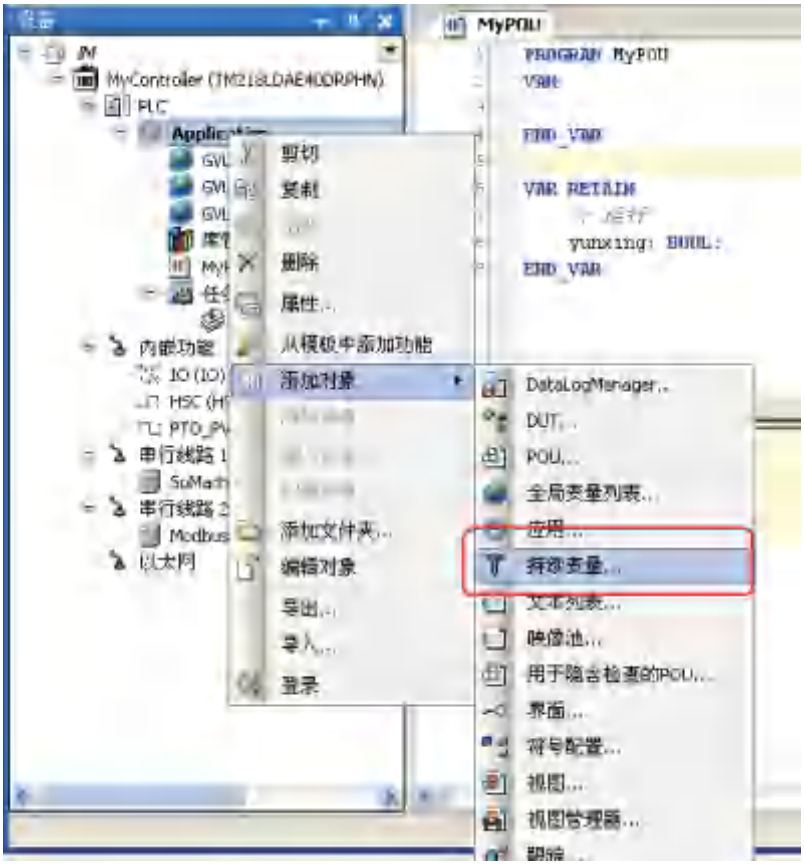


声明的变量会自动在声明区添加，并且与普通变量有所区别，保留变量会声明在“VAR RETAIN——END_VAR”声明区，即声明在此区间的变量为保留型变量。如下图所示，“yunxing”即为保留型局部变量。如果需要声明为全局变量，即可按照声明全局变量的方法，勾选上“RETAIN”标志即可。

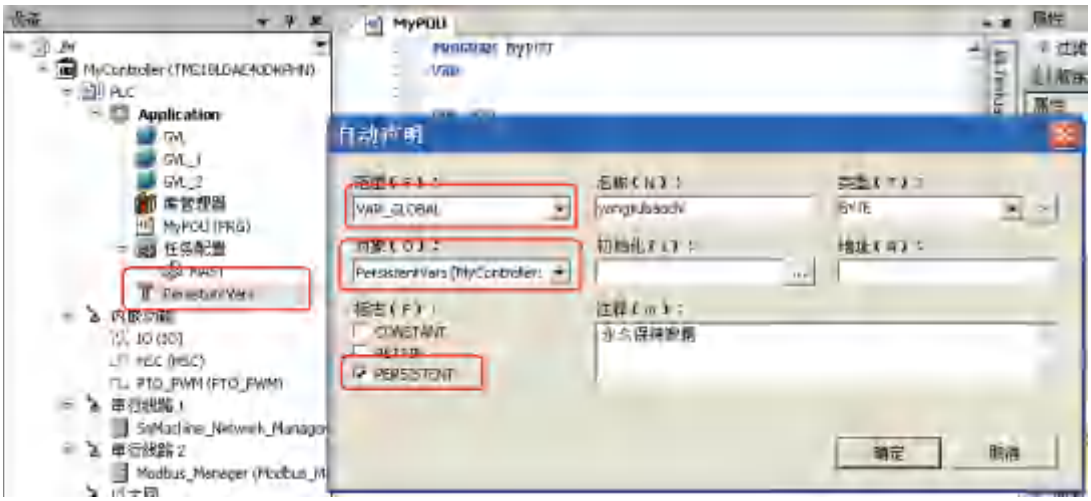


持久变量声明：

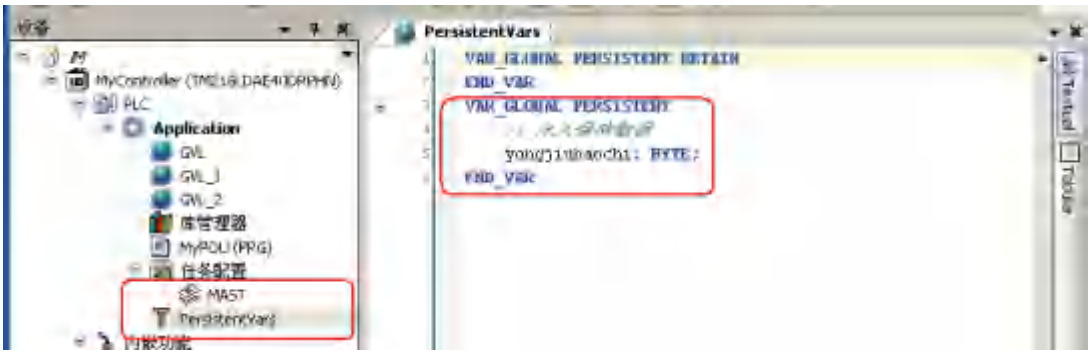
持久变量必须为全局变量，并且需要在“Application”里面添加“持续变量”对象，添加方法与添加POU类似，如下图所示，鼠标点击后将自动弹出添加窗口，点击打开即可在“Application”下添加对象“PersistentVars”（注：此名称不能更改）。



在变量自动声明窗口中将“标志”勾选为“PERSISTENT”，在“范围”下拉菜单中选择“VAR_GLOBAL”，在“对象”中选为“PersistentVars”即可。变量“yoongjiubaochi”将被声明到持久保持数据声明区，并可以作为全局变量被所有的POU调用。



添加完毕后，双击对象“PersistentVars”可以看到变量已经添加到了变量声明区，如下图所示。如果将自动声明窗口中的“RETAIN”和“PERSISTENT”全部勾选上，则会声明到PersistentVars里面的“VAR_GOLBAL PERSISTENT RETAIN——END_VAR”，其类型为“持久保留变量”，与“持久变量”性质相同。

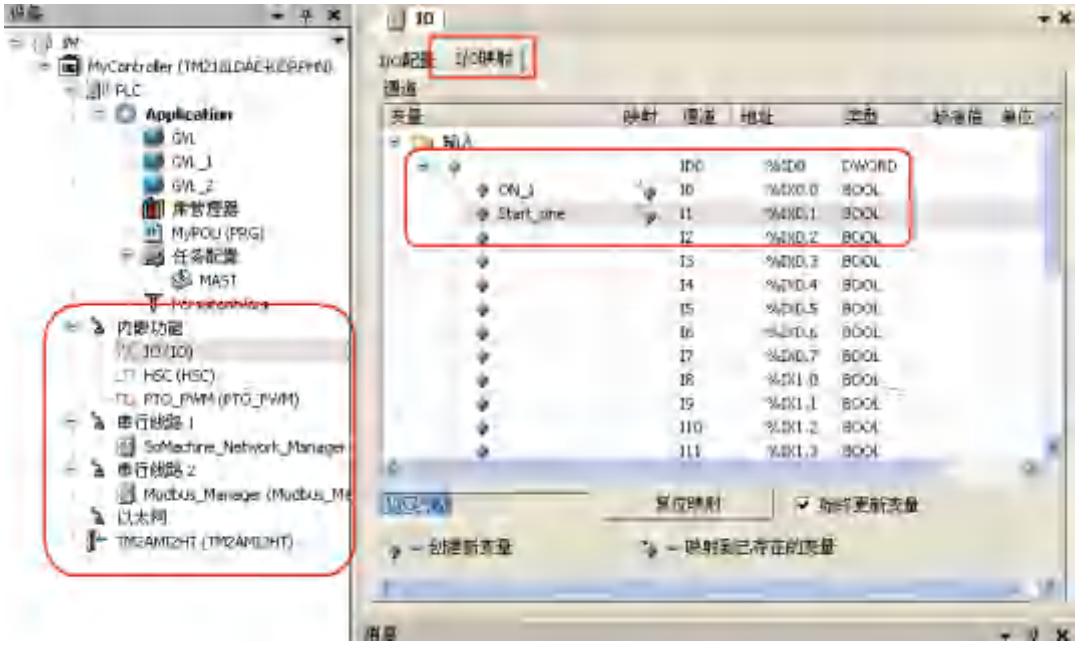


IO地址声明及映射

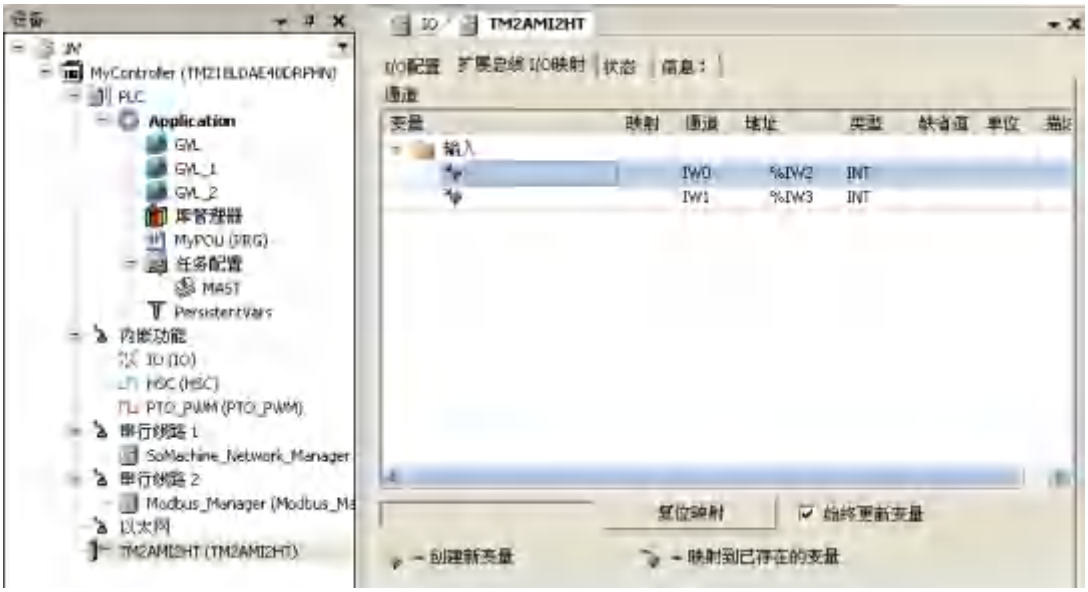
IO地址与变量名之间的映射关系可以通过前面介绍的直接在变量声明区通过功能符号AT的方式进行对应，也可以通过在“自动声明”窗口中通过添加“地址”的方式进行映射。

下面我们介绍第三种映射方式，也是我们推荐的IO映射方式，希望用户采用此种方式进行IO映射。

双击“内嵌功能”菜单里面的“IO”图标即可打开“IO”配置界面，在“IO映射”页面，我们只需要在对应的IO地址之前输入变量名即可实现地址的映射。如下图所示“ON_1”对应的实际地址为“%IX0.0”，“Start_one”对应的实际地址为“%IX0.1”。通过这样的映射关系即可在所有POU里面直接调用这两个变量，并且是作为全局变量使用。建议用户采用此种方法进行IO地址的映射。输出变量设置方法类似。



扩展模块的映射同样采取此种方法，在“%IW2”前面的变量名区输入需要的变量名即可实现映射关系，并且可以在所有的POU中被调用。

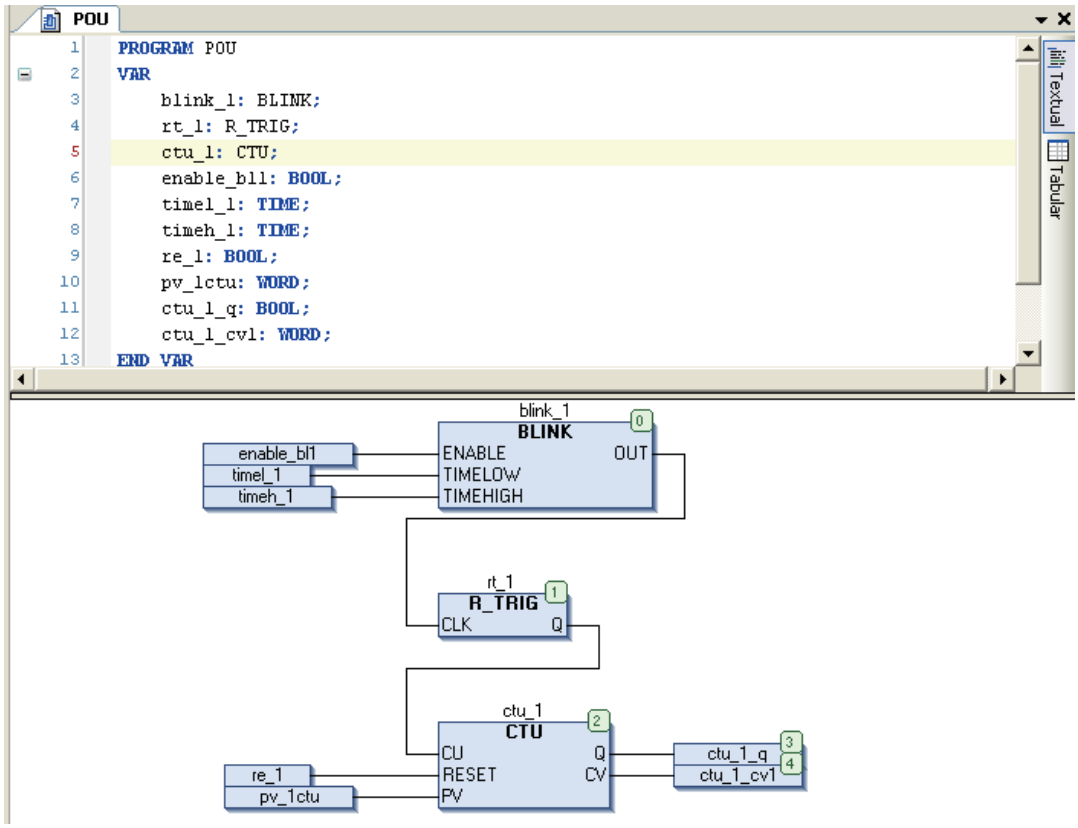


5.4.4编写程序

SoMachine软件支持六种编程语言，编程方式符合IEC61131-3标准。程序的编写在相关编程语言界面的程序编辑区进行。具体六种语言的编辑界面介绍请参见相关章节。

在程序编写之前请注意先将需要的功能配置完成，根据程序的结构创建所需要的POU和相应的任务。在编辑的过程中注意变量声明，以及变量的类型等事宜。

下图为基于CFC编程语言的加计数器示例，供参考：



5.4.5任务配置

SoMachine 设备树中的"任务配置"节点用于定义一个或多个任务以控制应用程序的执行。

可用的任务类型有：

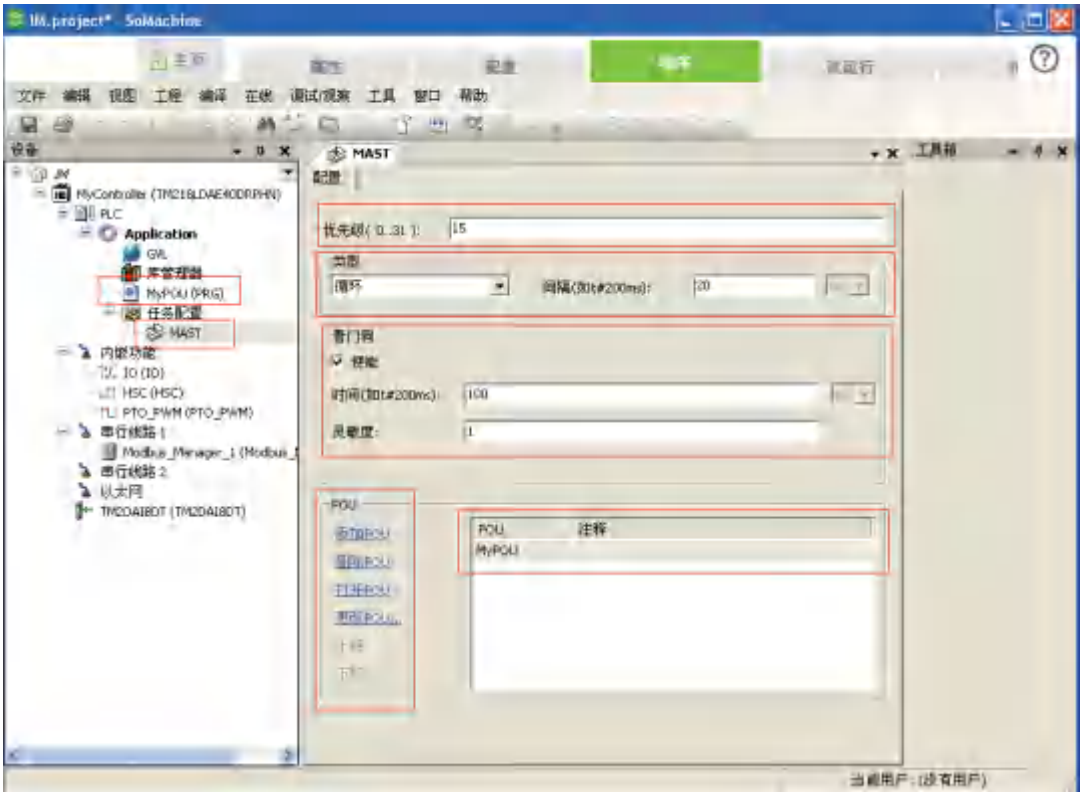
循环
自由运行

事件
外部事件

关于各种任务之间的具体说明和详细介绍请参见手册的“任务”这一章节，本章将介绍如何在工程项目中进行任务的配置。

创建SoMachine项目之后，系统中会默认生成一个“MAST”任务。这此任务中可以添加需要在此任务中运行的POU（程序组织单元）。

双击“MAST”任务图标可以进入任务设置页面，界面如下图所示：



在“POU”操作区可以进行POU的添加、POU的删除、打开POU、更改POU、上移、下移等操作。

在“看门狗”操作区可以进行看门狗功能的启用以及时间和灵敏度设置。

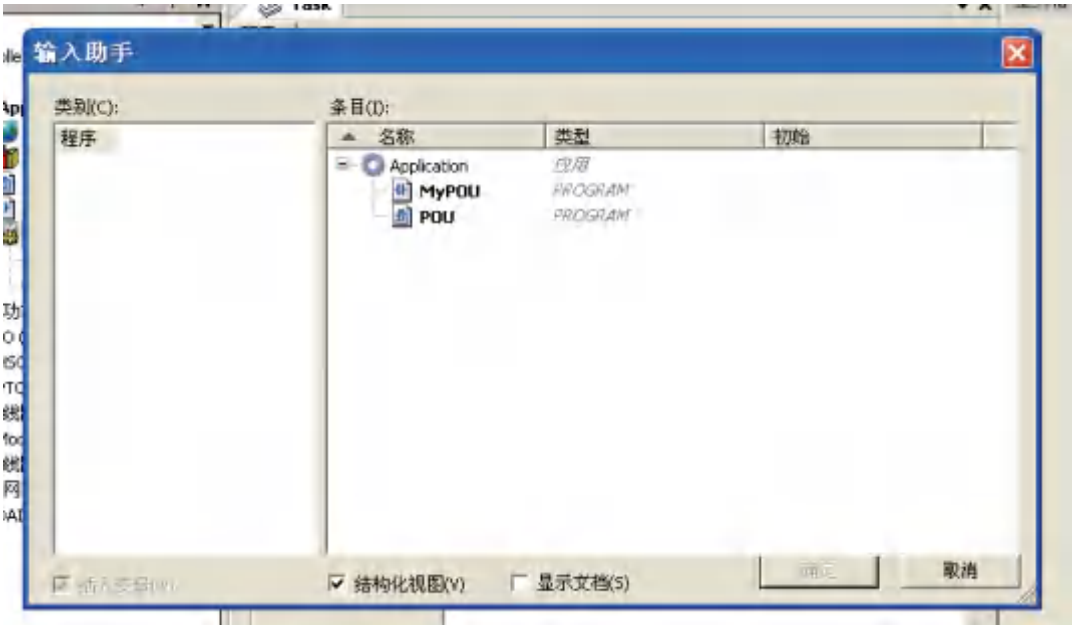
在“类型”操作区可以进行任务类型的选择，包括：循环、事件、外部的、自用运行。以及根据所选循环类型不同而想对应的循环事件设置、事件标志位设置、外部事件对应的IO设置。

添加一个新的任务

选中左边设备栏中的“任务配置”，点击鼠标右键，从下拉菜单中选择“添加对象”中的“任务”。

鼠标点击后将会自动弹出任务添加窗口，可以设置任务的名称。点击下方的“打开”按钮，将弹出类似“MAST”任务的设置窗口。

一个任务中可以添加多个POU，每个POU在任务里按照由上到下的顺序执行。可以通过“上移”和“下移”操作更改POU的执行顺序。



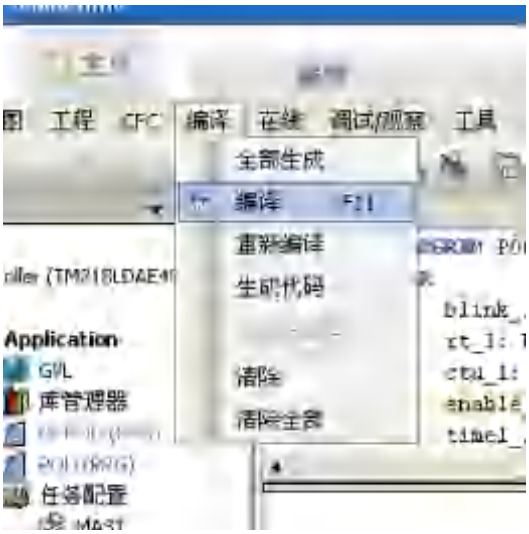
新添加任务的配置界面和“MAST”任务的配置界面相同。

5.4.6程序编译

程序编写完成后，在下载之前对程序进行编译。编译命令对编写的程序进行语法检查，并且只编译添加到任务中的程序。即，如果创建的POU没有添加到任务中，编译命令不对此POU进行语法检查。编译命令不生成代码，只进行语法检查。

如果直接执行设备登录命令，系统将默认执行编译指令（相当于先手动执行编译命令），在编译检查没有语法错误后执行连接登录指令。同样，在编译中不对没有添加到任务中的POU进行语法检查。执行登录命令同时会生成代码。

SoMachine中编译命令下拉菜单如下图所示：



全部生成：对当前应用和HMI应用同时进行编译。

编　　译：对当前应用进行编译。

重新编译：如果需要对已经编译过的应用再次编译，可以通过重新编译指令进行。

生成代码：执行此命令后生成当前应用的机器代码，执行登录命令时，生成代码默认执行。

清　　除：删除当前应用的编译信息，如果再次登录设备时需要重新生成编译信息。

清除全部：删除工程中所有的编译信息。

执行编译命令后可以看到，添加到任务里面的“POU”显示为蓝色，没有添加到任务里面的“MyPOU”则显示为灰色。编译指令没有对“MyPOU”里面的程序进行语法检查，因为该程序单元没有处于活动状态，编译指令只针对于处于活动状态的程序组织单元进行语法检查。如果在编译的过程中发现需要运行的程序单元显示为灰色，可以检查该程序单元是否已经被成功的添加到了所需要运行的任务当中。

编译命令执行完成之后可以在消息栏看到编译生成的信息，其中可以看到编译的程序是否有错误或者警告，以及错误和警告的数量。如果有错误和警告产生可以通过消息窗口进行查看和查找，根据提示信息对程序进行修改。

5.4.7应用程序下载与运行

简介

要运行应用程序，首先必须将 PC 连接到控制器，然后将应用程序下载到控制器。

下载项目的作用是让您将当前项目从 SoMachine 复制到控制器存储器。

注意： 某些控制器由于存储器大小的限制，无法存储应用程序源，只能存储执行的生成应用程序。这表示您无法将应用程序源从控制器上载到 PC。

警告

意外的设备操作

每次登录控制器后，请确保：

- 正在下载的软件应用程序必须安装在目标设备上。确认已输入正确的设备名称或设备地址。
- 防护措施必须到位，以便意外的设备操作不会导致人身伤害或设备损坏。
- 您必须已阅读并理解所涉及软件和设备的用户文档，且必须了解如何操作设备。

如果不遵守这些说明，将会导致死亡、严重伤害或设备损坏。

前提条件

在将应用程序下载到控制器之前，应用程序必须完全符合下列条件：

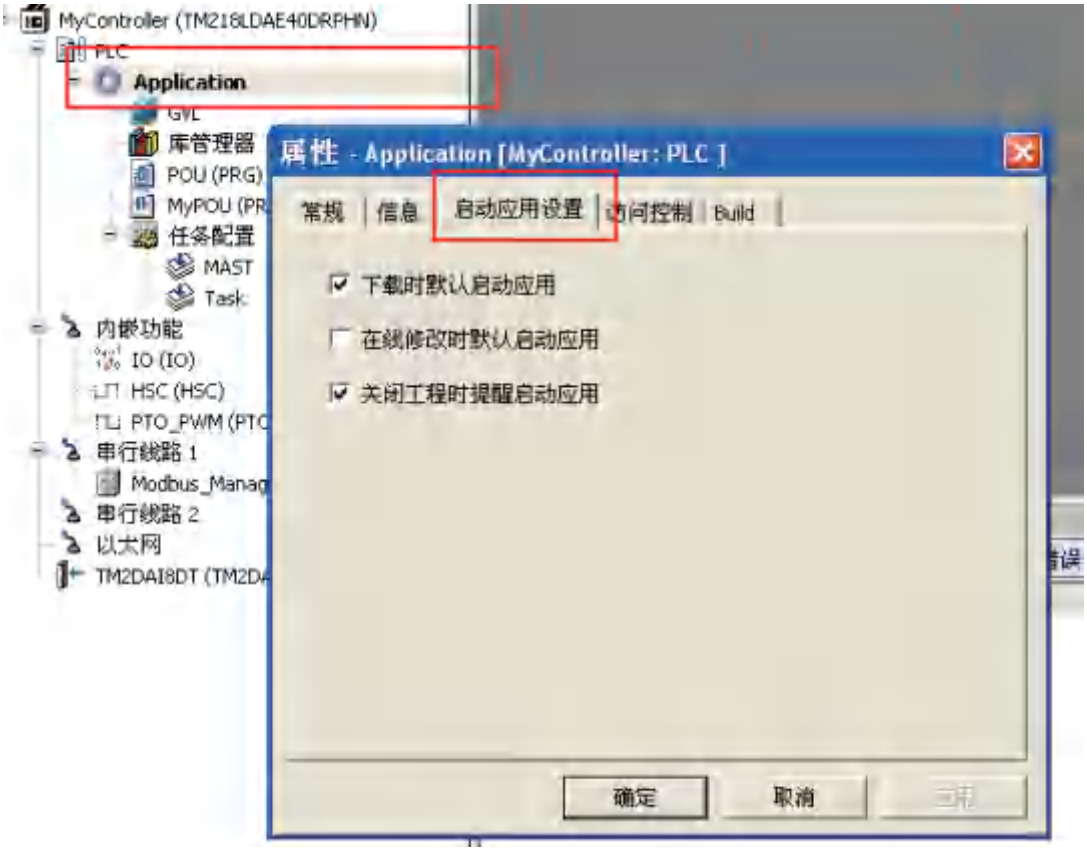
•必须针对控制器设置活动路径。

•要下载的应用程序必须处于活动状态。(SoMachine 中的控制器型号与连接的控制器型号一致)

•应用程序不能有任何编译错误。

启动应用程序

启动应用程序是在控制器启动时启动的应用程序。此类应用程序存储在控制器存储器中。要配置启动应用程序的下载，请右键单击设备视图中的应用程序节点，然后选择属性命令。



“下载时默认启动应用”：如果勾选上，在执行程序下载操作时，自动创建启动应用，不需要再次自行创建（建议勾选此项）

“在线修改时默认启动应用”：如果勾选上，在执行在线修改操作时会自动创建启动应用（不建议勾选此项）

“关闭工程时提醒启动应用”：如果勾选上此选项，在关闭工程时如果没有创建启动应用系统会提示您创建启动应用（建议勾选此项）

成功下载新应用程序结束时，会显示一条消息，询问您是否要创建启动应用程序。

可以通过下列方式手动创建启动应用程序：

•离线模式：**单击在线 → 创建启动应用**，将启动应用程序保存到文件中。

•在在线模式下，使用处于"停止"模式下的应用程序：**单击在线 → 创建启动应用**，将启动应用程序下载到控制器中。

操作模式

鉴于已加载应用程序和要下载的应用程序之间的关系，下载方法各有不同。存在 3 种情况：

•第一种情况：控制器中的应用程序与您要加载的应用程序相同。在这种情况下，您只能将 SoMachine 连接到控制器，而不能进行任何下载。

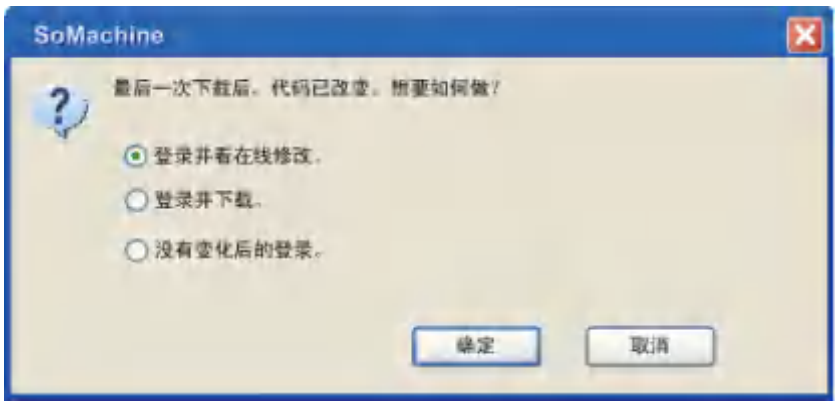
•第二种情况：您修改了控制器中加载的应用程序。在这种情况下，您可以指定是要全部或部分下载已修改的应用程序，还是让控制器中的应用程序保持原样。

•第三种情况：控制器应用程序与您要加载的应用程序不同。在这种情况下，您必须指定是否要连接已加载的应用程序，或者是否要加载新的应用程序。

将应用程序下载到控制器：第一种情况

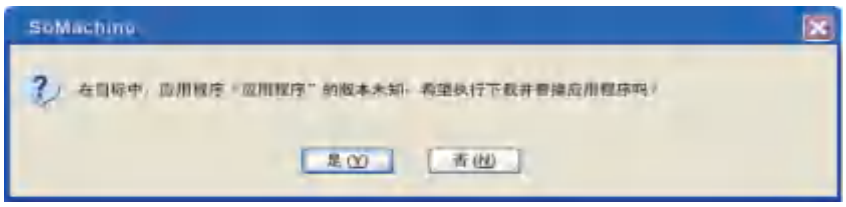
步骤	操作
1	要连接到控制器，请选择在线 → 登录到" 应用程序 [YourApplicationName; Plc Logic]" 。
2	此时连接到控制器。

将应用程序下载到控制器：第二种情况

步骤	操作
1	要连接到控制器，请选择在线 → 登录到" 应用程序 [YourApplicationName; Plc Logic]" 。
2	如果您修改了应用程序，并想要将其重新加载到控制器，则会出现以下消息：  登录并在线修改 只有正在运行项目的已修改部分会重新加载到控制器。 登录并下载 整个修改的应用程序将重新加载到控制器。 登录但不进行任何更改 不加载修改。 选择您要执行的操作，单击确定。

注意： 有关应用程序下载的重要安全相关信息，请参见您控制器的编程指南。

将应用程序下载到控制器：第三种情况

步骤	操作
1	要连接到控制器，请选择 在线 → 登录到 " 应用程序 [YourApplicationName; Plc Logic]" 。
2	如果要加载的应用程序不是当前控制器中的应用程序，则会出现以下消息：  要下载新应用程序到控制器，单击 确定 ，否则单击 取消 。

注意： 有关应用程序下载的重要安全相关信息，请参见您控制器的编程指南。

5.5试运行

一、试运行功能概述

注意：仅在打开 SoMachine 项目之后才会显示试运行选项卡。

该选项卡包含图形配置编辑器（类似于配置选项卡），但是仅提供试运行机器所需的功能：

•输入最后的参数

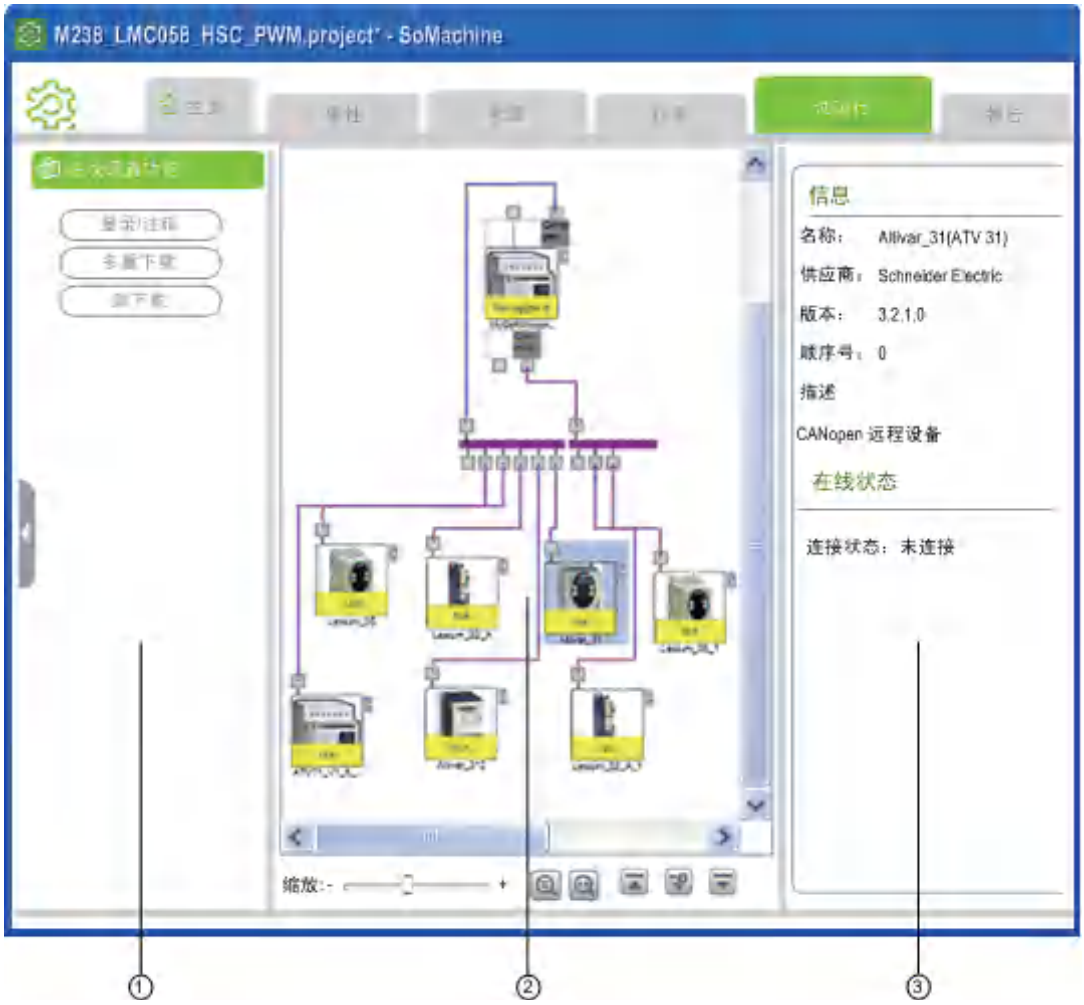
•查看在线状态

•登录机器

•下载到设备或从设备下载

试运行选项卡的元素

下图显示包含示例配置的试运行选项卡：



1、任务选择面板列出可用的任务。

2、工作区包括图形配置编辑器，其中的功能局限于试运行任务。

3、信息面板提供有关在工作区中选择的元素的更多信息以及此设备的在线状态。

在线状态信息

在图形配置编辑器中，用不同颜色分别指示每个设备的在线状态：

在线状态	颜色
未登录	黄色
运行	绿色
已停止	红色

对于在图形配置编辑器中选定的设备，在线状态信息会显示在信息面板中。

编辑设备和通讯参数

编辑用于试运行机器的设备参数

•双击图形配置编辑器中的设备图标，

或

•右键单击设备图标并从上下文菜单中选择编辑参数。

结果：会为选定的设备显示参数配置屏幕。

编辑用于试运行机器的通讯参数

•双击图形配置编辑器中的网络连接行，

或

•右键单击网络连接行并从上下文菜单中选择编辑参数。

结果：会为选定的网络连接显示网络配置屏幕。

注意： 有关编辑设备和通讯参数的重要安全相关信息，请参见您控制器的编程指南。

二、登录/注销

使用登录/注销任务，您可以在运行 SoMachine 的 PC 与连接的设备间建立连接，以上载/下载项目程序。（不是所有的PLC都支持程序的上载，具体信息请参阅产品手册或咨询施耐德工程师）。

在建立此连接之前，您必须将设备物理连接到 PC。有关此硬件连接的详细信息，请参阅设备的硬件手册。

⚠警告

意外的设备操作

每次登录到控制器时，都请确保以下事项：

- 确认已输入正确的设备名称或设备地址。
- 必须实施防护措施，以便意外的设备操作不会导致人身伤害或设备损坏。
- 必须阅读并了解所涉及软件和设备的用户文档，并且必须清楚如何操作设备。

如果不遵守这些说明，将会导致死亡、严重伤害或设备损坏。

根据项目中是包含一个或多个应用程序，登录过程会有所不同。

项目包含 1 个控制器和 1 个应用程序时的登录过程

要登录到连接的设备，请执行以下步骤：

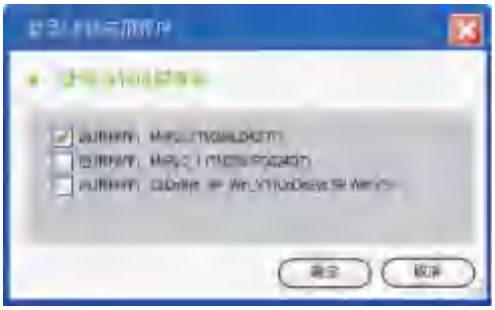
步骤	操作
1	在试运行 选项卡中，从任务选择面板单击 登录/注销 按钮。 结果：成功登录后，各个设备图标底部的信息行将通过不同的颜色指示设备的状态（例如，以红色指示停止）。

要注销，请在任务选择面板中再次单击登录/注销按钮。

结果：成功注销后，各个设备图标底部的信息行将以黄色显示未登录或无状态。

项目包含多个控制器且每个控制器对应 1 个应用程序时的登录步骤

要登录到连接的设备，请执行以下步骤：

步骤	操作
1	在试运行 选项卡中，从任务选择面板单击 登录/注销 按钮。 结果：将显示 登录/注销应用程序 对话框。 
2	在登录/注销应用程序 对话框中，选择您要登录的应用程序并单击 确定 按钮。 结果：成功登录后，各个设备图标底部的信息行将通过不同的颜色指示设备的状态（例如，以红色指示停止）。

要注销，请在任务选择面板中再次单击登录/注销按钮，取消选择选定的应用程序，然后单击关闭按钮。

结果：成功注销后，各个设备图标底部的信息行将以黄色显示未登录或无状态。

三、多重下载

对于多重下载操作，您必须指定 2 个设置：

•选择要下载的项目应用程序。

•指定在线修改选项和其他设置。

注意： 所有物理目标（与所选应用程序相关联的控制器和 HMI）都必须连接到 PC，且端口的通讯协议为SoMachine Network。

过程

按如下步骤执行多重下载任务：

步骤	操作
1	在试运行选项卡中，在任务选择面板中单击多重下载按钮。 结果：将显示多重下载对话框。 
2	在**请选择要下载的应用程序**部分中，选中要下载到新的源项目的应用程序。
3	选择适合您应用程序的**在线修改选项**。有关更多详细信息，请参阅在线帮助的 CoDeSys 部分。
4	选择适合您应用程序的**其他操作**选项。有关更多详细信息，请参阅在线帮助的 CoDeSys 部分。
5	单击**确定**按钮以启动下载过程。 结果：将显示**多重下载 - 结果**消息框，指示是否已成功完成每个应用程序的下载过程。

注意： 有关多重下载功能的重要安全相关信息，请参见您控制器的编程指南。

四、源下载任务

源下载：用于将存档的 SoMachine 项目版本从 PC 下载到连接的控制器。

注意：M218不支持源下载功能，控制器上必须提供额外的存储器扩展才能使用此功能。因此，此功能仅适用于 ATV IMC 和 M258 控制器以及将 USB 闪存驱动器作为辅助存储设备的 XBTGC 类型控制器。通过 SoMachine 3.0，XBTGT/GK 控制器可以将 CF 卡用作辅助存储设备，进行源下载。

过程：要将 SoMachine 项目下载到控制器，请执行以下步骤：

步骤	操作
1	按 **登录 /注销** 任务一章中的说明，登录到要将 SoMachine 项目下载的控制器。
2	从任务选择面板单击 **源下载** 按钮。 结果 ：当前已打开项目的项目存档会自动创建并下载到控制器。

5.6报告

注意：仅在打开 SoMachine 项目之后，报告选项卡才可用。

该选项卡提供创建和自定义项目报告的功能。

左侧的任务选择面板显示报告创建步骤：

•页面设置

•打印预览

•打印

报告选项卡的元素

此选项卡还提供自定义任务，使您可根据个人喜好修改报告。

下图显示报告选项卡的元素：

1、用于创建报告任务

2、用于选择的项目树结构

3、用于自定义报告任务

4、有关选定对象的详细信息

报告选项卡的按钮

工作区中的按钮可用于查看项目的树结构和排列此结构中的项。每个项目项都提供一个复选框。选中要包含在报告中的项的复选框。

报告设置的按钮	含义
上移 	单击**上移**可以在项目的树结构中向上移动项目项。
下移 	单击**下移**可以在项目的树结构中向下移动项。
全选 	单击**全选**可以选中所有项目项的复选框
全不选 	单击**全不选**可以取消选中所有项目项的复选框
全部展开 	单击**全部展开**可以展开所有项目项的树结构的详细信息
全部折叠 	单击**全部折叠**可以关闭树结构视图。

设置打印参数

要设置报告的打印参数，请单击左侧任务选择面板中的**页面设置**按钮。根据您的个人要求调整标准 Windows 页面设置对话框的设置。有关详细信息，请参阅 Windows 在线帮助。

打印报告

要将报告打印出来，请执行以下步骤：

步骤	操作
1	单击 **打印预览** 可以查看选择的项。 结果：**打印预览** 随即生成。使用 页面 文本框在 **打印预览** 中浏览。
2	单击 打印 可以启动打印。 结果： 用于 打印 的 Windows 对话框随即打开。 选择您的设置并单击 **确定** 以启动打印。

对于页面设置，请使用 Windows 打印对话框。

自定义任务

自定义任务用于根据个人喜好自定义报告。如果单击**自定义**按钮，则会打开下面的屏幕：

1、工作区的上半部分提供用于自定义报告的选项。

2、工作区的下半部分显示页面配置的预览。预览周围的框使您可以将信息放置在报告中。

用于自定义报告的选项

用于自定义的选项	步骤
添加徽标	单击**添加**可以将徽标添加到报告中。 结果：选择徽标 对话框打开，使您可以浏览至要用作徽标的图片文件。 徽标可以显示在报告的页眉或页脚。
更改或删除徽标	单击**更改**可替换为另一个徽标，单击 **删除**可删除徽标。
添加文本	在文本框 **文本 1** 和**文本 2** 中，输入要在报告打印输出的页眉或页脚显示的可选文本。所有条目都显示在窗口下半部分的预览中。
排列报告中的信息	预览的顶部和底部提供了一些框，使用这些框可以将选定的信息 **文本 1**、**文本 2**、**徽标**、**页标题**、**日期**、**项目名称**、Project Number 放置在报告页眉或页脚的所需位置处。从列表中选择要放在给定位置上的信息。

第六章 M218固件更新

第六章 M218固件更新

简介

正常情况下，执行固件更新并不会删除设备中现有的应用程序，包括闪存中的引导应用程序。但为以防万一，请您在执行固件更新之前，准备好您的应用程序，以备恢复设备应用程序。

⚠ 小心

应用程序数据丢失

- 在尝试固件更新之前需备份应用程序，将其备份到 PC 的硬盘。
- 固件更新成功后，恢复设备的应用程序。

如果不遵守这些说明，将会导致受伤或设备损坏。

如果在应用程序传输或固件更新的过程中出现断电或通讯中断，那么您的设备可能无法正常工作。如果出现断电或通讯中断，请再次尝试传输。

注意

设备无法操作

- 传输一旦开始，不要中断应用程序的传输或固件更新。
- 在传输成功完成之前不要将设备投入使用。

如果不遵守这些说明，则会导致设备损坏。

当执行固件更新之后，控制器的串行线路端口 1 会被恢复到缺省的 SoMachine 协议进行配置。 SoMachine 协议与其他协议（如 Modbus 串行线路）不兼容。 如果将新控制器连接到某个配置了 Modbus 的串行线路，或更新连接到该串行线路的控制器的固件，则可能会导致该串行线路上的其他设备停止通讯。 在下载针对预期端口协议正确配置之前，请确保控制器未连接到活动 Modbus 串行线路网络。

注意

意外的设备操作

在将控制器物理连接到正常运行的 Modbus 串行线路网络之前，请确保应用程序针对 Modbus 正确配置了串行线路端口。

如果不遵守这些说明，则会导致设备损坏。

在开始固件更新过程之前，请确保您已准备好：

- USB编程电缆（TCS XCNA MUM3P）
- Modicon M218 Logic Controller

此更新过程属于维护操作。 需要将控制器从会受其影响的系统和应用程序断开。 在此操作过程中，PC 和控制器必须保持接连状态。

注意：如果在固件更新期间 PC 和控制器意外断开连接，则在成功执行新的固件更新操作前，控制器无法正常工作。

电缆安装步骤

步骤	操作
1	将 USB编程电缆（TCS XCNA MUM3P）插入 PC 的 USB 端口。
2	将电缆的另一端插入控制器的 USB 端口。
3	启动Exec Loader for M218(启动位置:开始菜单->程序->Schneider-Electric->Somachine->Tool->Exec Loader for M218)

Exec Loader for M218使用方法

简介

M218 Exec Loader 向导是一个基于 Windows 的向导，用于指导您完成更新 Schneider Electric 控制器中的固件所必需的步骤。

注意
要启动 Exec Loader 向导，请完成以下步骤： 1: 关闭所有 Windows 应用程序（包括虚拟计算机）。 2: 如果网关正在运行，请在任务栏中右键单击 CoDeSys Gateway Sys Try (running) 图标并选择 Stop Gateway。 停用网关后，CoDeSys Gateway Sys Tray (stopped 图标会出现在任务栏中: 3: 启动Exec Loader for M218(启动位置:开始菜单->程序->Schneider-Electric->Somachine->Tool->Exec Loader for M218

M218固件升级步骤

步骤	屏幕	功能
1	Welcome(欢迎)	Exec Loader 向导简介。
2	Setting(设置)	选择要传输到控制器的正确固件文件。
3	Connecting M218(M218连接中)	自动连接M218 PLC
4	File and Device Exec Properties	比较固件文件和控制器的硬件 ID 和固件版本信息。
5	Transfer Progress	监控固件文件到控制器的传输。

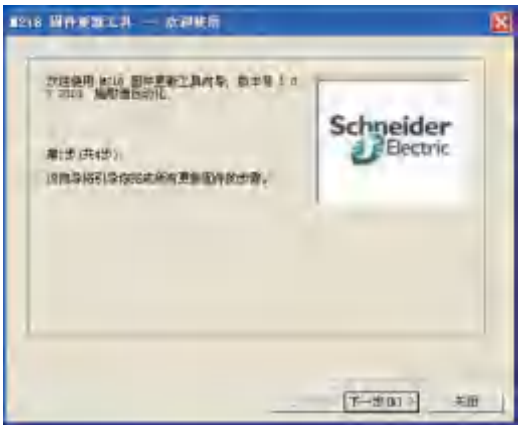
M218固件升级步骤详细介绍

步骤1-Welcome(欢迎)

该向导为每个步骤提供了一个屏幕。Welcome屏幕是 Exec Loader 向导的简介。

要继续，请执行以下操作：

- 选择“下一步”继续此过程并显示下一个屏幕(步骤 2 Setting)，并手动关闭M218 PLC电源。
- 选择“关闭”以关闭屏幕而不更改控制器上的固件。



步骤2-Setting(设置)

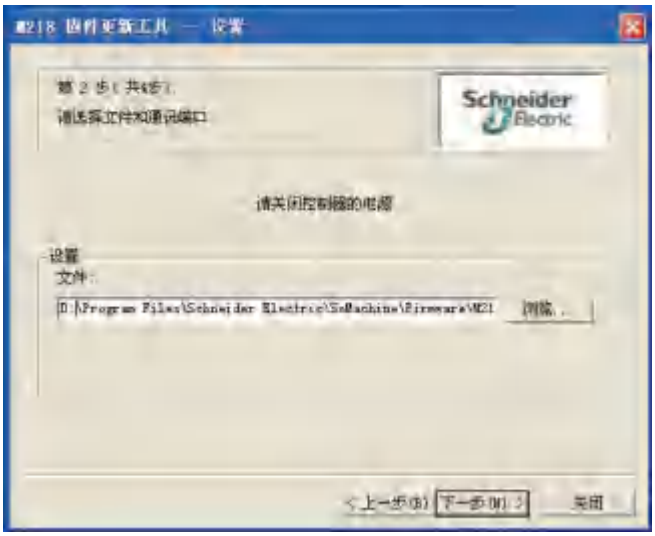
- 在设置界面中，单击“浏览”按钮为控制器选择正确的固件文件。示例(例如软件安装在D盘)：D:\Program Files\Schneider Electric\Firmware\M218\M218_V2.0.30.36_20111116.mfw
- 选择“上一步返回到“欢迎界面”。
  选择“关闭”以关闭屏幕而不更改控制器上的固件。

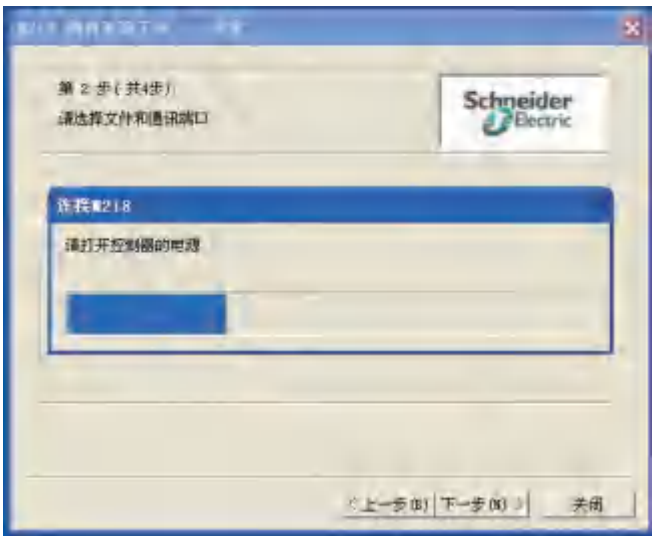
只有通过“浏览”按钮选择了M218的固件后，“下一步”按钮才从灰色不可操作状态变成黑色可操作状态。

注意事项

Setting界面开启“下一步”按钮只是通过判断是否选择了M218的固件，并不判断选择的固件和PLC型号是否匹配

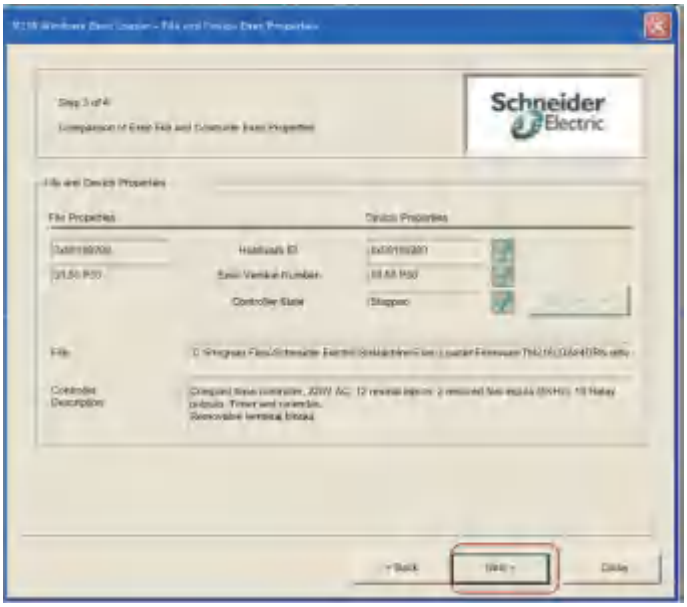
在Setting(设置)界面中，用户在点击“下一步”按钮之前，请确认M218 PLC已经断电，且PLC上所有的灯处于熄灭状态，电脑与M218 PLC之间的通讯电缆连接良好。





步骤3-Connecting M218(M218连接中)

进入Connecting M218(M218 连接中)界面后，用户在进度条运行过程中，打开M218 PLC的控制电源，当Exec Loa-der for M218软件成功打开与M218的连接后，该向导会自动进入步骤4。

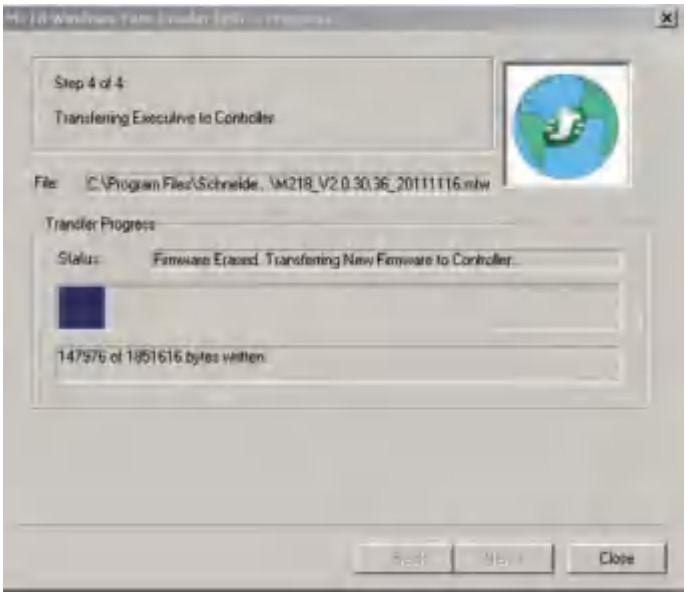


File and Device Exec Properties(固件和设备版本属性)

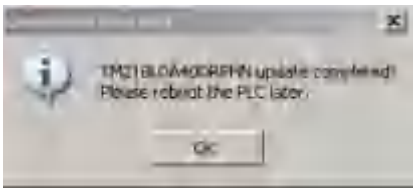
在此步骤中，Exec Loader 向导会先检查固件文件和控制器的以下信息，然后继续后面的操作:

- Hardware ID(硬件ID) - 对于目标控制器，选择的固件文件是否正确。
  Hardware ID(硬件ID) 是每个控制器参考号的唯一标识符。
  - Green Check mark(绿色对号)-确定
  - Red Cross(红色差号)-错误的固件文件。请选择与控制器参考号对应的固件文件(通过“上一步”按钮返回到步骤2)
- Exec Version Number(识别固件的版本) - 选择的固件文件是否比当前安装的固件更新。
  - Green Check mark(绿色对号)-确定
  - Yellow Check mark(黄色对号)- 将控制器降级到较旧版本的固件，或使用当前固件的相同版本升级控制器点击“下一步”按钮开始升级固件

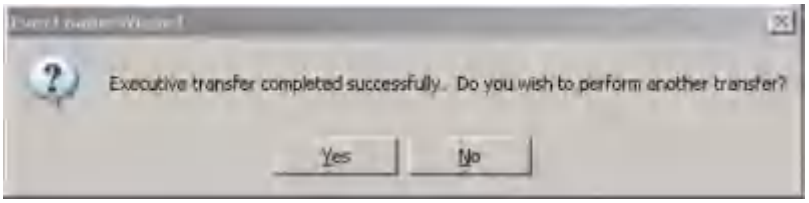
步骤4-Transfer Progress(传输过程)



通过此屏幕可以监控传输进度。经过一段时间后会提供剩余时间信息。
更新完成，弹出以下对话框：



如果传输成功，则显示消息窗框，以允许进行其他传输。有以下两个选择可以使用：
是 - 该向导将返回到步骤2，然后用户可以设置其他传输
否 – 单击“关闭”按钮以退出向导。该操作将完成更新过程



如果传输不成功
如果传输中断（例如，由于通讯中断），则会显示消息框，以允许重试传输。有两个选项可用：
是 - 该向导返回步骤4(固件和设备版本属性)，用户可以尝试再次传输
否 – 单击“关闭”按钮可以退出向导

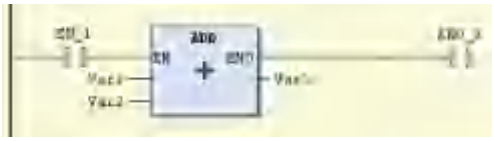
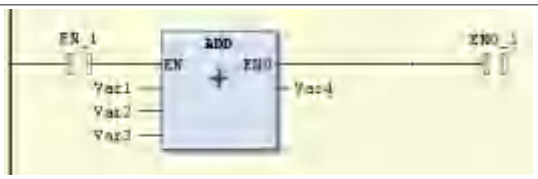
注意

设备无法操作
- 传输一旦开始，不要中断应用程序的传输或固件更新。
- 在传输成功完成之前不要将设备投入使用。

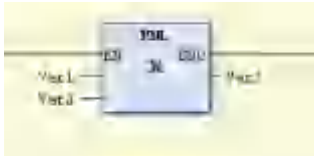
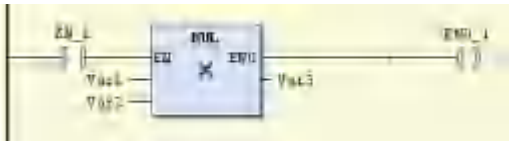
如果不遵守这些说明，则会导致设备损坏。

第七章 操作符说明

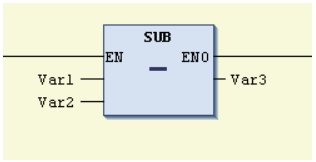
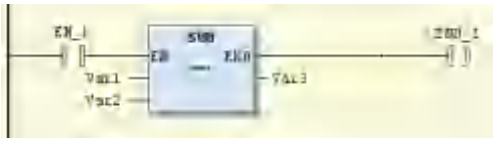
7.1.1加法

操作符	ADD
功能说明	该功能块用作变量或常量相加，Var3=Var1+Var2，输入输出的变量类型必须相同
图形	
管脚定义	输入： EN：功能块使能（BOOL型）；当其为高电平时，ADD功能块被激活 Var1：加数1 Var2：加数2 输出： ENO：辅助输出；一旦EN为高电平时，其值为高电平 Var3：和；即输入值相加后的结果 当EN为TRUE时，ADD功能块被激活，ENO输出为TRUE；加数Var1、Var2相加后，将其结果赋值给Var3 （*例：Var1=4，Var2=2，Var3=6*）
变量声明	VAR EN_1： BOOL; Var1:: INT; Var2:: INT; ENO_1:: BOOL; Var3： INT; END_VAR
CFC语言示例	
ST语言示例	1 Var3:=Var1+Var2;(*Var1为4，Var2为2，结果Var3为6*)
LD语言示例	
时序图	
使用限制	"输入/输出适用的数据类型：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL、TIME"
注意事项	 TIME型量也可以使用加法功能，两个TIME型量相加得到一个新的时间量。例如：t#45s + t#50s= t#1m35 s。 被选择的输出数据类型应可以存储输出结果，否则可能引起数据错误。MUL、SUB、DIV 指令同样。 加法功能块还可以增加输入管脚，例如：Var4=Var1+Var2+Var3。MUL、SUB、DIV 指令同样。

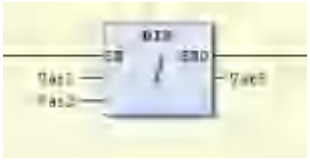
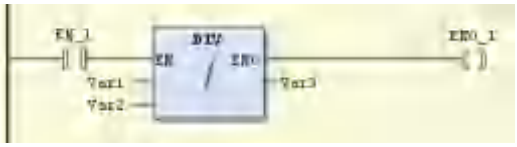
7.1.2乘法

操作符	MUL
功能说明	该功能块用作变量或常量相乘；Var3=Var1*Var2，输入输出的数据类型必须相同
图形	
管脚定义	输入： EN：功能块使能（BOOL型）；当其为高电平时，MUL功能块被激活 Var1：乘数1 Var2：乘数2 输出： ENO：辅助输出；一旦EN为高电平时，其值为高电平 Var3：积；即输入变量相乘的结果 当EN为TRUE时，MUL功能块被激活，ENO输出为高电平；乘数Var1、Var2相乘后，将其结果赋值给Var3； （*例：Var1=4，Var2=2，Var3=8*）
变量声明	VAR EN_1： BOOL; Var1:: INT; Var2:: INT; ENO_1:: BOOL; Var3： INT; END_VAR
CFC语言示例	
ST语言示例	1 Var3:=Var1*Var2;(*Var1为4，Var2为2，结果Var3为8*)
LD语言示例	
时序图	
使用限制	"输入/输出适用的数据类型：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL"
注意事项	

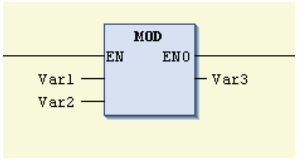
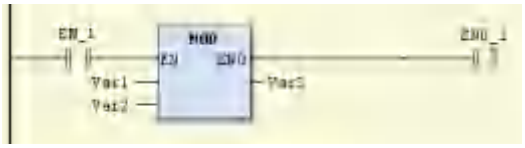
7.1.3减法

操作符	SUB
功能说明	该功能块用作变量或常量相减；Var3=Var1-Var2；输入输出的数据类型必须相同
图形	
管脚定义	输入： EN：功能块使能（BOOL型）；当其为高电平时，SUB功能块被激活 Var1：被减数 Var2：减数 输出： ENO：辅助输出（BOOL型）；一旦EN为高电平时，其值为高电平 Var3：差；被减数减去减数后的结果 当EN为1时，SUB功能块被激活，ENO输出为1；减数Var1、Var2相减后，将其结果赋值给Var3 ， （*例Var1=4，Var2=2，Var3=2*）
变量声明	VAR EN_1： BOOL； Var1:: INT； Var2:: INT； ENO_1:: BOOL； Var3： INT； END_VAR
CFC语言示例	
ST语言示例	1 Var3:=Var1-Var2;(*Var1为4，Var2为2，结果Var3为2*)
LD语言示例	
时序图	
使用限制	"输入/输出适用的数据类型：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL、TIME、TOD"
注意事项	"TIME 型变量也可以使用减法功能，两个TIME 型量相减得到一个新的时间量。例如：t#1m35s - t#50s = t#45s ，但时间结果不能有负值。TOD 型变量也可以使用减法功能，两个TOD 型量相减得到一个新的TIME 型数据，例如：TOD#23:40:30- TOD#00:30:20=T#1390m10s0ms ,但时间结果不能有负值。"

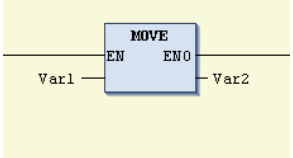
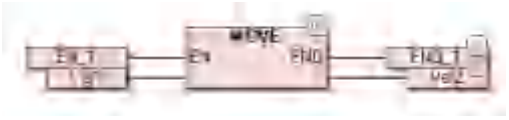
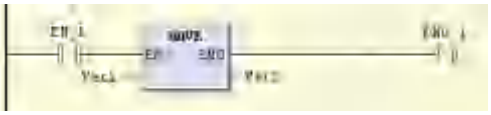
7.1.4除法

操作符	DIV
功能说明	该功能块用作变量或常量相除；Var3=Var1/Var2；输入输出变量类型必须相同
图形	
管脚定义	输入： EN：功能块使能（BOOL型）；当其为高电平时，MUL功能块被激活 Var1：被除数 Var2：除数 输出： ENO：辅助输出（BOOL型）；一旦EN为高电平时，其值为高电平 Var3：商；即被除数和除数相除后的结果 当EN为TRUE时，DIV功能块被激活，ENO输出为TRUE；Var1、Var2相除后，将其结果赋值给Var3，（*例Var1=4，Var2=2，Var3=2*）
变量声明	VAR EN_1： BOOL； Var1:: INT； Var2:: INT； ENO_1:: BOOL； Var3： INT； END_VAR
CFC语言示例	
ST语言示例	1 Var3:=Var1/Var2;(*Var1为4，Var2为2，结果Var3为2*)
LD语言示例	
时序图	"输入/输出适用的数据类型：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL"
使用限制	
注意事项	"在工程中使用DIV 指令时，可使用CheckDivByte、CheckDivWord、CheckDivDWord 和 CheckDivReal 等指令，检查除数是否为零，避免除数为零的现象。"

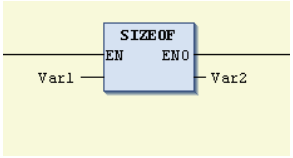
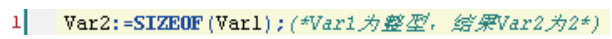
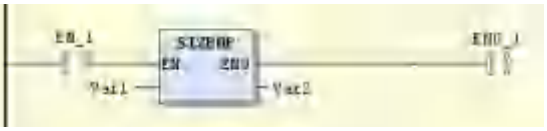
7.1.5取余

操作符	MOD
功能说明	该功能块用作Var1的值除以Var2的值取余数，Var3=Var1 MOD Var2；输入值和输出值的数据类型必须是相同的。
图形	
管脚定义	输入： EN：功能块使能（BOOL型）；当其为高电平时，MOD功能块被激活 Var1：被除数； Var2：除数； 输出： ENO：辅助输出；一旦EN为高电平时，其值为高电平 Var3：模数，即相除后的余数； 当EN为高电平时，MOD功能块被激活，ENO输出为高电平，连续执行MOD功能时EN需保持高电平；被除数Var1除以除数Var2后，将结果中的余数赋值给Var3，公式为Var3:=Var1 MOD Var2；（*例：Var1=5，Var2=2，Var3=1*）
变量声明	VAR EN_1： BOOL； Var1:: INT； Var2:: INT； ENO_1:: BOOL； Var3： INT； END_VAR
CFC语言示例	
ST语言示例	1 Var3:=Var1 MOD Var2;(*Var1为5，Var2为2，结果Var3为1*)
LD语言示例	
时序图	
使用限制	
注意事项	"输入/输出适用的数据类型：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT"

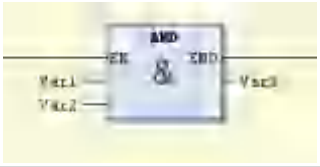
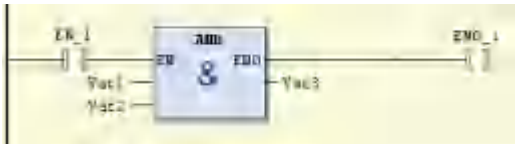
7.1.6赋值

操作符	MOVE
功能说明	该功能块用作将一个常量或者变量的值赋给另外一个变量
图形	
管脚定义	输入： EN：布尔型（BOOL）；当其为TRUE时，MOVE功能块被激活 Var1：变量1 输出： ENO：布尔型（BOOL）；一旦EN为TRUE时，其值为TRUE Var2：变量2 当EN为TRUE时，MOVE功能块被激活，ENO输出为TRUE；把变量1的数据传送到指定的变量2中，Var2=Var1； （*例：Var1为5，结果Var2为5*）
变量声明	VAR EN_1： BOOL; Var1：INT; ENO_1：BOOL; Var2： INT; END_VAR
CFC语言示例	
ST语言示例	1 Var2:=Var1;(*Var1为5，结果Var2为5*)
LD语言示例	
时序图	
使用限制	"输入/输出适用的数据类型：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL、TIME、DT、BOOL、STRING、ARRAY"
注意事项	

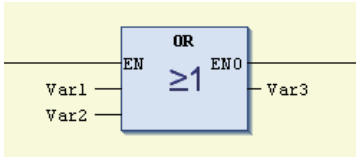
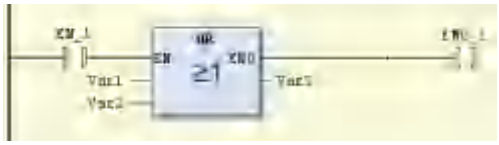
7.1.7 SIZEOF

操作符	SIZEOF
功能说明	该功能块用作取得输入变量所占用的字节数
图形	
管脚定义	输入： EN：布尔型（BOOL）；当其为TRUE时，SIZEOF功能块被激活 Var1：变量1 输出： ENO：布尔型（BOOL）；一旦EN为TRUE时，其值为TRUE Var2：变量2，存储变量1数据类型所需字节数的数据 当EN为TRUE时，SIZEOF功能块被激活，ENO输出为TRUE；将变量1所需的字节数存储到变量2中，Var2:=SIZEOF(Var1)； （*例：Var1为整型，结果Var2为2，一个整型数据类型需要2个字节*）
变量声明	VAR EN_1： BOOL; Var1：INT; ENO_1：BOOL; Var2： INT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	SIZEOF操作符通常返回一个无符号数。返回值的类型与变量Var1的大小相匹配。 SIZEOF(Var1)的返回值 0 <= Var1< 256 USINT 256 <= Var1 < 65536 UINT 65536 <Var1 < 4294967296 UDINT 4294967296 <= Var1 ULINT

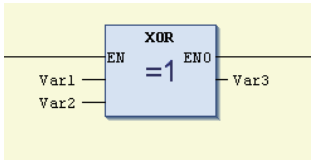
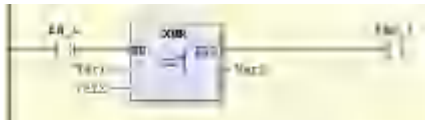
7.2.1与

操作符	AND
功能说明	该功能块用作变量或常量的相与运算，输入输出变量类型必须一致。
图形	
管脚定义	输入： EN：布尔型（BOOL）；当其为TRUE时，AND功能块被激活 Var1：变量1 Var2：变量2 输出： ENO：布尔型（BOOL）；一旦EN为TRUE时，其值为TRUE Var3：单字（WORD）；变量3，变量1和变量2相与运算后的结果 当EN为TRUE时，AND功能块被激活，ENO输出为TRUE；Var1变量1和Var2变量2进行相与运算，将其结果赋值给Var3，Var3:=Var1 AND Var2； (*例：Var1为2#10010011，Var2为2#10001010，结果Var3为2#10000010*)
变量声明	VAR EN_1：BOOL； Var1：WORD； Var2：WORD； ENO_1：BOOL； Var3：WORD； END_VAR
CFC语言示例	
ST语言示例	`1 Var3:=Var1 AND Var2;(*Var1为2#10010011，Var2为2#10001010，结果Var3为2#10000010*)`
LD语言示例	
时序图	
使用限制	输入/输出适用的数据类型：BOOL、BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT
注意事项	

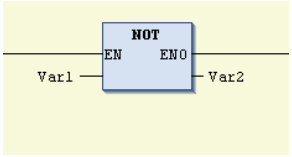
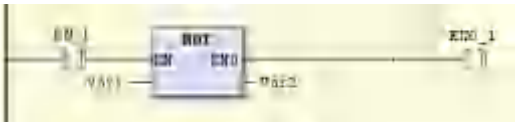
7.2.2或

操作符	OR
功能说明	该功能块用作变量或常量的相或运算，输入输出数据类型必须一致
图形	
管脚定义	输入： EN：布尔型（BOOL）；当其为TRUE时，OR功能块被激活 Var1：变量1 Var2：变量2 输出： ENO：布尔型（BOOL）；一旦EN为TRUE时，其值为TRUE Var3：单字（WORD）；变量3，变量1和变量2相或运算后的结果 当EN为TRUE时，OR功能块被激活，ENO输出为TRUE；Var1变量1和Var2变量2进行相或运算，将其结果赋值给Var3，Var3:=Var1 OR Var2； (*例：Var1为2#10010011，Var2为2#10001010，结果Var3为2#10011011*)
变量声明	VAR EN_1： BOOL; Var1： WORD; Var2： WORD; ENO_1： BOOL; Var3： WORD; END_VAR
CFC语言示例	
ST语言示例	1 Var3:=Var1 OR Var2;(*Var1为2#10010011，Var2为2#10001010，结果Var3为2#10011011*)
LD语言示例	
时序图	
使用限制	输入/输出适用的数据类型：BOOL、BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT
注意事项	

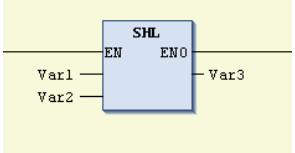
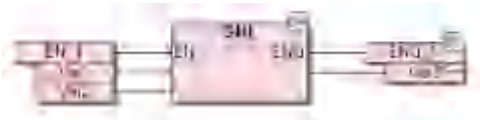
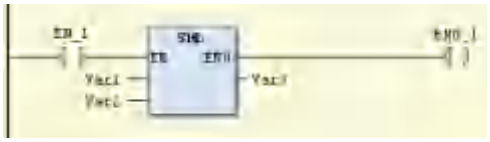
7.2.3异或

操作符	XOR
功能说明	该功能块用作变量或常量的异或运算，输入输出数据类型必须一致
图形	
管脚定义	输入： EN：布尔型（BOOL）；当其为TRUE时，XOR功能块被激活 Var1：单字（WORD）；变量1 Var2：单字（WORD）；变量2 输出： ENO：布尔型（BOOL）；一旦EN为TRUE时，其值为TRUE Var3：单字（WORD）；变量3，变量1和变量2异或运算后的结果 当EN_1为TRUE时，XOR功能块被激活，ENO_1输出为TRUE；Var1变量1和Var2变量2进行相或运算，将其结果赋值给Var3，Var3:=Var1 XOR Var2； (*例：Var1为2#10010011，Var2为2#10001010，结果Var3为2#00011001*)
变量声明	VAR EN_1： BOOL; Var1：WORD; Var2：WORD; ENO_1：BOOL; Var3： WORD; END_VAR
CFC语言示例	
ST语言示例	1 Var3:=Var1 XOR Var2;(*Var1为2#10010011，Var2为2#10001010，结果Var3为2#00011001*)
LD语言示例	
时序图	
使用限制	输入/输出适用的数据类型：BOOL、BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT
注意事项	

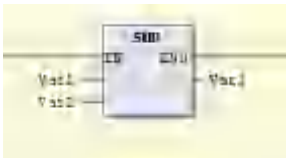
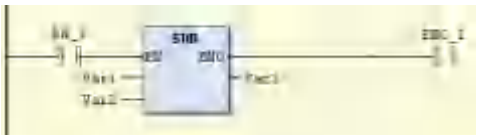
7.2.4非

操作符	NOT
功能说明	该功能块用作变量或常量的取非运算， 逐位取非，输入输出的数据类型必须一致
图形	
管脚定义	输入： EN：布尔型（BOOL）；当其为TRUE时，NOT功能块被激活 Var1：变量1 输出： ENO：布尔型（BOOL）；一旦EN为TRUE时，其值为TRUE Var2：变量2，变量1和变量2取非运算后的结果 当EN为TRUE时，NOT功能块被激活，ENO输出为TRUE；Var1变量1进行取非运算，将其结果赋值给Var2，Var2:=NOT Var1； (*例：Var1为2#10010011，结果Var2为2#01101100*)
变量声明	VAR EN_1： BOOL; Var1：WORD; ENO_1：BOOL; Var2：WORD; END_VAR
CFC语言示例	
ST语言示例	1 Var2:=NOT Var1;(*Var1为2#10010011，结果Var2为2#01101100*)
LD语言示例	
时序图	
使用限制	输入/输出适用的数据类型：BOOL、BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT
注意事项	

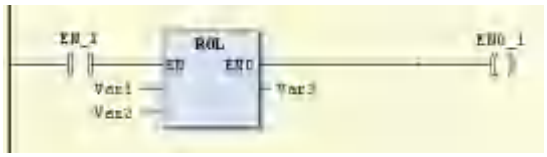
7.3.1左移

操作符	SHL
功能说明	该功能块用作对操作数进行按位左移，左边移出的位不作处理，右边自动补0
图形	
管脚定义	输入： EN：布尔型（BOOL）；当其为TRUE时，SHL功能块被激活 Var1：变量1需要左移的操作数 Var2：变量2操作数左移的位数 输出： ENO：布尔型（BOOL）；一旦EN为TRUE时，其值为TRUE Var3：变量3，对操作数进行按位左移后的结果 当EN为TRUE时，SHL功能块被激活，ENO输出为TRUE；对操作数变量1进行按变量2的位数左移后的结果赋值给Var3，Var3:=SHL(Var1,Var2)； (*例：Var1为16#45，Var2为2，结果Var3为16#0114*)
变量声明	VAR EN_1：BOOL； Var1：WORD； Var2：WORD； ENO_1：BOOL； Var3：WORD； END_VAR
CFC语言示例	
ST语言示例	1 Var3:=SHL(Var1,Var2);(*Var1为16#45，Var2为2，结果Var3为16#0114*)
LD语言示例	
时序图	
使用限制	输入/输出适用的数据类型：BYTE、INT、WORD、DWORD、SINT、UINT
注意事项	Var1:=SHL(16#45,2)；（*Var1：BYTE；结果Var1 为16#14*） Var2:=SHL(16#45,2)；（*Var2：WORD；结果Var2 为16#0114*） 注意：上面例子中，虽然输入变量的值一样，但因为输出数据类型的大小不同，所以得到的结果Var1和Var2 不同。如果左移的位数超出了数据本身的位数，BYTE、WORD和DWORD类型的操作数将会补0，而有符号类型的操作数（例如INT）将会进行算术移位。也就是说会将这些数的最高位的值补在空出的二进制位上。

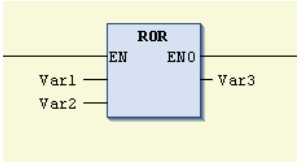
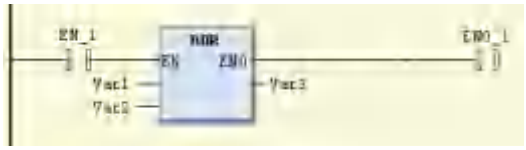
7.3.2右移

操作符	SHR
功能说明	该功能块用作对操作数进行按位右移，右边移出的位不作处理，左边自动补0
图形	
管脚定义	输入： EN：布尔型（BOOL）；当其为TRUE时，SHR功能块被激活 Var1：变量1需要右移的操作数。 Var2：变量2右移的位数 输出： ENO：布尔型（BOOL）；一旦EN为TRUE时，其值为TRUE Var3：变量3，对操作数进行按位右移后的结果 当EN为TRUE时，SHR功能块被激活，ENO输出为TRUE；对操作数变量1进行按变量2的位数右移后的结果赋值给Var3，Var3:=SHR(Var1,Var2)； (*例：Var1为16#45，Var2为2，结果Var3为16#0011*)
变量声明	VAR EN_1： BOOL； Var1：WORD； Var2：WORD； ENO_1：BOOL； Var3：WORD； END_VAR
CFC语言示例	
ST语言示例	`1  Var3:=SHR(Var1,Var2);(*Var1为16#45，Var2为2，结果Var3为16#0011*)`
LD语言示例	
时序图	
使用限制	输入/输出适用的数据类型：BYTE、INT、WORD、DWORD、SINT、UINT
注意事项	Var1:=SHL(16#45,2)；（*Var1：BYTE；结果Var1 为16#14*） Var2:=SHL(16#45,2)；（*Var2：WORD；结果Var2 为16#0114*） 注意：上面例子中，虽然输入变量的值一样，但因为输出数据类型的大小不同，所以得到的结果Var1和Var2 不同。如果左移的位数超出了数据本身的位数，BYTE、WORD和DWORD类型的操作数将会补0，而有符号类型的操作数（例如INT）将会进行算术移位。也就是说会将这些数的最高位的值补在空出的二进制位上。

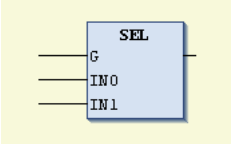
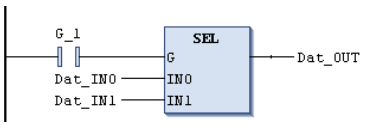
7.3.3循环左移

操作符	ROL
功能说明	该功能块用作对操作数进行按位循环左移，左边移出的位直接补充到右边最低位
图形	
管脚定义	输入： EN：布尔型（BOOL）；当其为TRUE时，ROL功能块被激活 Var1：变量1循环左移的操作数 Var2：变量2循环左移的位数 输出： ENO：布尔型（BOOL）；一旦EN为TRUE时，其值为TRUE Var3：变量3，对操作数进行按位左移后的结果 当EN为TRUE时，ROL功能块被激活，ENO输出为TRUE；对操作数变量1进行按变量2的位数循环左移后的结果赋值给Var3，Var3:=ROL(Var1,Var2)； (*例：Var1为16#45，Var2为2，结果Var3为16#0114*)
变量声明	VAR EN_1：BOOL； Var1：WORD； Var2：WORD； ENO_1：BOOL； Var3：WORD； END_VAR
CFC语言示例	
ST语言示例	1 Var3:=ROL(Var1,Var2);(*Var1为16#45，Var2为2，结果Var3为16#0114*)
LD语言示例	
时序图	
使用限制	输入/输出适用的数据类型：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT
注意事项	Var1:=SHL(16#45,2)；（*Var1：BYTE；结果Var1 为16#14*） Var2:=SHL(16#45,2)；（*Var2：WORD；结果Var2 为16#0114*） 注意：上面例子中，虽然输入变量的值一样，但因为输出数据类型的大小不同，所以得到的结果Var1和Var2 不同。如果左移的位数超出了数据本身的位数，BYTE、WORD和DWORD类型的操作数将会补0，而有符号类型的操作数（例如INT）将会进行算术移位。也就是说会将这些数的最高位的值补在空出的二进制位上。

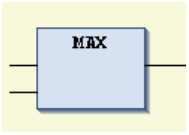
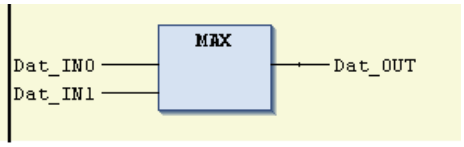
7.3.4循环右移

操作符	ROR
功能说明	该功能块用作对操作数进行按位循环右移，右边移出的位直接补充到左边最高位
图形	
管脚定义	输入： EN：布尔型（BOOL）；当其为TRUE时，ROR功能块被激活 Var1：变量1循环右移的操作数 Var2：变量2循环右移的位数 输出： ENO：布尔型（BOOL）；一旦EN为TRUE时，其值为TRUE Var3：变量3，对操作数进行按位右移后的结果 当EN为TRUE时，ROR功能块被激活，ENO输出为TRUE；对操作数变量1进行按变量2的位数循环右移后的结果赋值给Var3，Var3:=ROR(Var1,Var2)； (*例：Var1为16#45，Var2为2，结果Var3为16#40011*)
变量声明	VAR EN_1：BOOL； Var1：WORD； Var2：WORD； ENO_1：BOOL； Var3：WORD； END_VAR
CFC语言示例	
ST语言示例	1 Var3:=ROR(Var1,Var2);(*Var1为16#45，Var2为2，结果Var3为16#40011*)
LD语言示例	
时序图	
使用限制	输入/输出适用的数据类型：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT
注意事项	Var1:=SHL(16#45,2)；（*Var1：BYTE；结果Var1 为16#14*） Var2:=SHL(16#45,2)；（*Var2：WORD；结果Var2 为16#0114*） 注意：上面例子中，虽然输入变量的值一样，但因为输出数据类型的大小不同，所以得到的结果Var1和Var2 不同。如果左移的位数超出了数据本身的位数，BYTE、WORD和DWORD类型的操作数将会补0，而有符号类型的操作数（例如INT）将会进行算术移位。也就是说会将这些数的最高位的值补在空出的二进制位上。

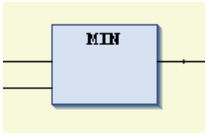
7.4.1二选一

操作符	SEL
功能说明	该功能块用于从两个输入端变量选择一个输入端变量作为输出
图形	
管脚定义	输入： G：布尔型（BOOL）；该输入端为FALSE，输出OUT：=IN0；输入端为TRUE，输出OUT：=IN1 IN0：任意类型；二选一的候选变量一 IN1：任意类型；二选一的候选变量二 输出： 任意类型；将二选一的候选变量选中一个后作为输出。
变量声明	VAR G_1: BOOL; Dat_IN0: INT; Dat_IN1: INT; Dat_OUT: INT; END_VAR
CFC语言示例	
ST语言示例	3 Dat_OUT:=SEL(G_1, Dat_IN0, Dat_IN1); 4
LD语言示例	
时序图	
使用限制	
注意事项	输入端IN0与IN1的变量类型可以不同，但必须是可以比较的数据类型，如IN0为BYTE，IN1为Real，Out为Real是允许的；如IN0为BOOL（其值为False或True），IN1为Real，Out为Real是不允许的，因为Bool型与整数或浮点数无法比较。如果输入端变量类型不一样，输出端的变量类型必须是可以涵盖输入端变量长度的变量，例如，输入端变量分别为BYTE和INT，则输出端变量类型至少为INT，不能为BYTE（有关变量数据类型长度的介绍请参考变量类型章节）。无论选择输入端变量G的值是False或True，输出端均会有输出

7.4.2取最大值

操作符	MAX
功能说明	该功能块用于从两个或多个输入端变量的值当中选出最大数作为输出值
图形	
管脚定义	输入： 任意类型 输出： 任意类型
变量声明	VAR Dat_IN0: INT; Dat_IN1: INT; Dat_OUT: INT; END_VAR
CFC语言示例	
ST语言示例	2 3 Dat_OUT:=MAX(Dat_IN0, Dat_IN1, Dat_IN2); 4
LD语言示例	
时序图	
使用限制	输入数量最大不能超过480个，否则在编译时，会引起软件异常退出，为保证程序正常运行，建议不参与比较的数据数目不要超过300个。
注意事项	数据输入数量可以为两个以上，在LD语言中通过菜单项“插入输入”/CFC语言中通过工具箱中“输入引脚”快捷工具项增加输入数量。 多个输入端的变量类型可以不同，但必须是可以比较的数据类型，如一个输入引脚为BYTE，另外一个输入引脚为Real，Out为Real是允许的；如一个输入引脚为BOOL（其值为False或True），另外一个输入引脚为Real，输出端变量为Real是不允许的，因为Bool型与整数或浮点数无法比较。 如果输入端类型不一样，输出端的变量类型必须是可以涵盖输入端变量长度的变量，例如，输入端分别为BYTE和INT，则输出端变量类型至少为INT，不能为BYTE（有关变量数据类型长度的介绍请参考变量类型章节）。

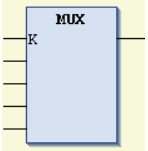
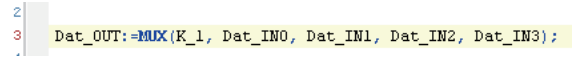
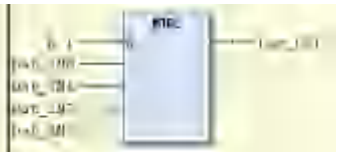
7.4.3取最小值

操作符	MIN
功能说明	该功能块用于从两个或多个数据中选出最小数
图形	
管脚定义	输入： 任意类型 输出： 任意类型
变量声明	VAR Dat_IN0: INT; Dat_IN1: INT; Dat_OUT: INT; END_VAR
CFC语言示例	
ST语言示例	1 Dat_OUT:=MIN(Dat_IN0, Dat_IN1); 2 3
LD语言示例	
时序图	
使用限制	输入数量最大不能超过480个，否则在编译时，会引起软件异常退出，为保证程序正常运行，建议不参与比较的数据数目不要超过300个。
注意事项	数据输入数量可以为两个以上，在LD语言中通过菜单项“插入输入”/CFC语言中通过工具箱中“输入引脚”快捷工具项增加输入数量。 多个输入端的变量类型可以不同，但必须是可以比较的数据类型，如一个输入引脚为BYTE，另外一个输入引脚为Real，Out为Real是允许的；如一个输入引脚为BOOL（其值为False或True），另外一个输入引脚为Real，输出端变量为Real是不允许的，因为Bool型与整数或浮点数无法比较。 如果输入端类型不一样，输出端的变量类型必须是可以涵盖输入端变量长度的变量，例如，输入端分别为BYTE和INT，则输出端变量类型至少为INT，不能为BYTE（有关变量数据类型长度的介绍请参考变量类型章节）。

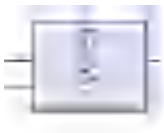
7.4.4取极限值

操作符	LIMIT
功能说明	该功能块用于对输入变量的值作限制输出，即无论输入端变量IN怎样变化，只输出MN及MX之间的数值
图形	
管脚定义	输入： 任意类型 输出： 任意类型 如果输入端变量IN的值大于输入端变量MN的值，并小于输入端变量MX的值，输出值等于输入端变量IN的值 如果输入端变量IN的值小于等于输入端变量MN的值，输出值等于MN的值 如果输入端变量IN的值大于等于输入端变量MX的值，输出值等于MX的值
变量声明	VAR Dat_Min: INT; Dat_IN1: INT; Dat_Max: INT; Dat_OUT: INT; END_VAR
CFC语言示例	
ST语言示例	1 2 Dat_OUT:=LIMIT(Dat_Min, Dat_IN1, Dat_Max); 3
LD语言示例	
时序图	
使用限制	
注意事项	多个输入端的变量类型可以不同，但必须是可以比较的数据类型，如一个输入引脚为BYTE，另外一个输入引脚为Real，Out为Real是允许的；如一个输入引脚为BOOL（其值为False或True），另外一个输入引脚为Real，输出端变量为Real是不允许的，因为Bool型与整数或浮点数无法比较。 如果输入端类型不一样，输出端的变量类型必须是可以涵盖输入端变量长度的变量，例如，输入端分别为BYTE和INT，则输出端变量类型至少为INT，不能为BYTE（有关变量数据类型长度的介绍请参考变量类型章节）。 如果输入端变量MN上变量的数值大于输入端变量MX上变量的数值，无论IN的值为多少，输出端均输出输入端变量MX上变量的数值。

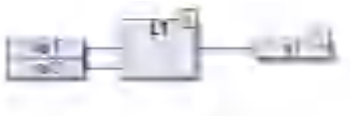
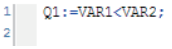
7.4.5多选一

操作符	MUX
功能说明	该功能块用于从多个输入引脚中选择一个变量输出
图形	
管脚定义	输入： K：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT或UDINT类型，K的值会决定选中第几个输入引脚作为输出 多个选择输入端：任意类型；多选一的候选变量 输出： OUT：任意类型；将多选一的候选变量选中一个后输出
变量声明	VAR K_1: BYTE; Dat_IN0: INT; Dat_IN1: INT; Dat_IN2: INT; Dat_IN3: INT; Dat_OUT: INT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	数据输入数量可以为两个以上，在LD语言中通过菜单项“插入输入”/CFC语言中通过工具箱中“输入引脚”快捷工具项增加输入数量。 多个输入端的变量类型可以不同，但必须是可以比较的数据类型，如一个输入引脚为BYTE，另外一个输入引脚为Real，Out为Real是允许的；如一个输入引脚为BOOL（其值为False或True），另外一个输入引脚为Real，输出端变量为Real是不允许的，因为Bool型与整数或浮点数无法比较。 如果输入端类型不一样，输出端的变量类型必须是可以涵盖输入端变量长度的变量，例如，输入端分别为BYTE和INT，则输出端变量类型至少为INT，不能为BYTE（有关变量数据类型长度的介绍请参考变量类型章节）。 如果选择输入K的值大于供选择的输入端变量的数目，输出端会有输出最后一个输入端的变量。 如果供选择的输入端变量的数目大于选择输入K的值的范围，超过K值的输入端变量的变量将不起作用，例如K为BYTE型，其最大取值为255,如果供选择的输入端变量有300个，第255个输入端变量后面的输入端变量永远不会被选中。

7.5.1大于

操作符	GT
功能说明	这是一个布尔量操作符。当第一个操作数比第二个大时，返回值为TRUE。
图形	
管脚定义	输入： 操作数可以是任何基本数据类型，如：BOOL, BYTE,WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL,TIME, DATE, TIME_OF_DAY,DATE_AND_TIME 和STRING. 输出： 布尔型（BOOL）；
变量声明	VAR var1: INT; var2: INT; q1: BOOL; END_VAR
CFC语言示例	
ST语言示例	`Q1:=VAR1>VAR2;`
LD语言示例	
时序图	
使用限制	
注意事项	

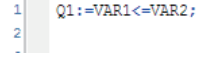
7.5.2小于

操作符	LT
功能说明	这是一个布尔量操作符。当第一个操作数比第二个小时，返回值为TRUE。
图形	
管脚定义	输入： 操作数可以是任何基本数据类型，如：BOOL, BYTE,WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL,TIME, DATE, TIME_OF_DAY,DATE_AND_TIME 和STRING. 输出： 布尔型（BOOL）；
变量声明	VAR var1: INT; var2: INT; q1: BOOL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

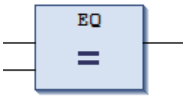
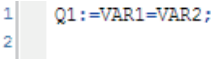
7.5.3小于等于

操作符	LE
功能说明	这是一个布尔量操作符。当第一个操作数比第二个小或者等于时，返回值为TRUE。
图形	
管脚定义	输入： 操作数可以是任何基本数据类型，如：BOOL, BYTE,WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL,TIME, DATE, TIME_OF_DAY,DATE_AND_TIME 和STRING. 输出： 布尔型（BOOL）；
变量声明	VAR var1: INT; var2: INT; q1: BOOL; END_VAR
CFC语言示例	
ST语言示例	1 Q1:=VAR1<=VAR2; 2
LD语言示例	
时序图	
使用限制	
注意事项	

7.5.4大于等于

操作符	GE
功能说明	这是一个布尔量操作符。当第一个操作数比第二个大或者等于时，返回值为TRUE。
图形	
管脚定义	输入： 操作数可以是任何基本数据类型，如：BOOL, BYTE,WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL,TIME, DATE, TIME_OF_DAY,DATE_AND_TIME 和STRING. 输出： 布尔型（BOOL）；
变量声明	VAR var1: INT; var2: INT; q1: BOOL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

7.5.5等于

操作符	EQ
功能说明	这是一个布尔量操作符。当第一个操作数与第二个等于时，返回值为TRUE。
图形	
管脚定义	输入： 操作数可以是任何基本数据类型，如：BOOL, BYTE,WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL,TIME, DATE, TIME_OF_DAY,DATE_AND_TIME 和STRING. 输出： 布尔型（BOOL）；
变量声明	VAR var1: INT; var2: INT; q1: BOOL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

7.5.6不等于

操作符	NE
功能说明	这是一个布尔量操作符。当第一个操作数与第二个不等于时，返回值为TRUE。
图形	
管脚定义	输入： 操作数可以是任何基本数据类型，如：BOOL, BYTE,WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL,TIME, DATE, TIME_OF_DAY,DATE_AND_TIME 和STRING. 输出： 布尔型（BOOL）；
变量声明	VAR var1: INT; var2: INT; q1: BOOL; END_VAR
CFC语言示例	
ST语言示例	1 Q1:=VAR1<>VAR2; 2
LD语言示例	
时序图	
使用限制	
注意事项	

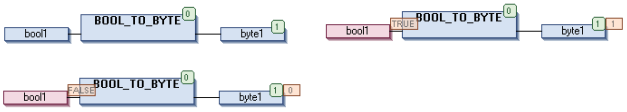
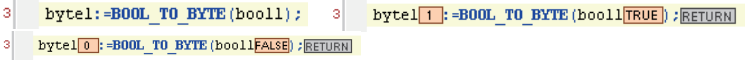
7.6.1取地址

操作符	ADR
功能说明	ADR返回变量自身的地址，数据类型为DWORD。这个地址可以作为指针传递给操作函数，也可以赋给工程内的某个指针
图形	
管脚定义	输入： 操作数为的数据类型为： BYTE,WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL,TIME, DATE, TIME_OF_DAY,DATE_AND_TIME ， STRING和ARRAY 输出： 指针地址(DWORD)；
变量声明	VAR VAR1: INT; VAR2: POINTER TO INT; END_VAR
CFC语言示例	
ST语言示例	1 var2:=ADR(VAR1); 2
LD语言示例	
时序图	
使用限制	
注意事项	在线修改之后，某些地址中的数据可能会被改变。请在使用地址指针时务必注意

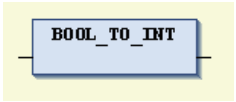
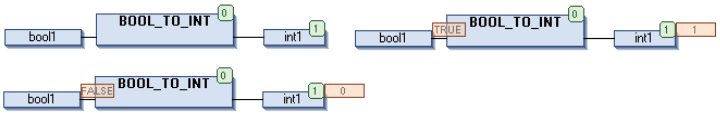
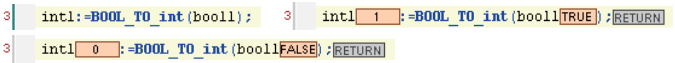
7.6.2取位地址

操作符	BITADR
功能说明	"BITADR的返回值为DWORD中的位偏移量。请注意偏移量的值取决于目标设置中的字节编址是否被激活。"
图形	
管脚定义	输入: 操作数为的数据类型为: BOOL ; 输出: 指针地址(DWORD);
变量声明	VAR A1_BOOL:BOOL; VAR1 AT %IX0.2: BOOL; VAR2: POINTER TO WORD END_VAR
CFC语言示例	
ST语言示例	1 var2:=BITADR(VAR1); 2
LD语言示例	
时序图	
使用限制	
注意事项	在线修改之后，某些地址中的数据可能会被改变。请在使用地址指针时务必注意

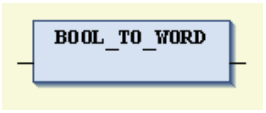
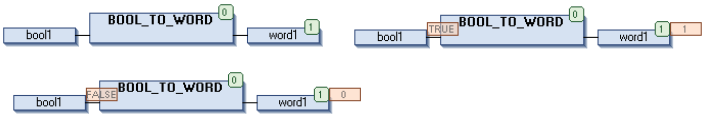
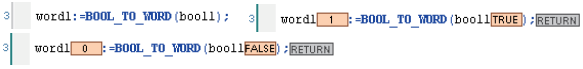
7.7.1布尔类型转换-1

操作符	BOOL_TO_BYTE
功能说明	该功能块用作布尔类型数据向字节数据类型的转换
图形	
管脚定义	若操作数为TRUE，结果为1；若操作数为FALSE，结果为0.
变量声明	VAR bool1: BOOL; byte1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	
使用限制	
注意事项	

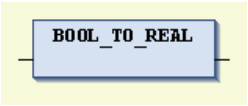
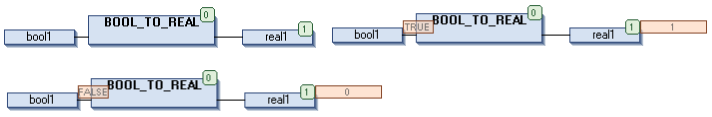
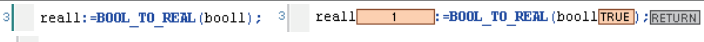
7.7.1布尔类型转换-2

操作符	BOOL_TO_INT
功能说明	该功能块用作布尔类型数据向整型类型数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：布尔型(BOOL); 输出： 整型(INT);
变量声明	VAR bool1: BOOL; int1: INT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	BOOL_TO_DINT 用作布尔类型数据向双整型类型数据的转换 BOOL_TO_LINT 用作布尔类型数据向长整型类型数据的转换 BOOL_TO_SINT 用作布尔类型数据向短整型类型数据的转换 BOOL_TO_UINT 用作布尔类型数据向无符号整型类型数据的转换 BOOL_TO_UDINT 用作布尔类型数据向无符号双整型类型数据的转换 BOOL_TO_ULINT 用作布尔类型数据向无符号长整型类型数据的转换 BOOL_TO_USINT 用作布尔类型数据向无符号短整型类型数据的转换 若操作数为TRUE，结果为1；若操作数为FALSE，结果为0，如上所示
使用限制	。
注意事项	

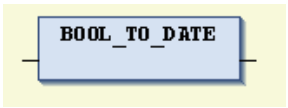
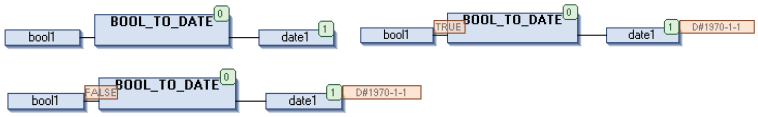
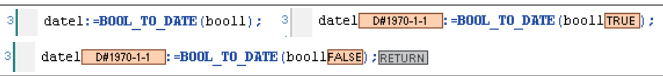
7.7.1布尔类型转换-3

操作符	BOOL_TO_WORD
功能说明	该功能块用作布尔类型数据向字类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为：布尔型(BOOL); 输出: 字型(WORD);
变量声明	VAR bool1: BOOL; word1: word; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	BOOL_TO_DWORD 用作布尔类型数据向双字类型数据的转换 BOOL_TO_LWORD 用作布尔类型数据向长字类型数据的转换 若操作数为TRUE，结果为1；若操作数为FALSE，结果为0，如上所示 。
使用限制	
注意事项	

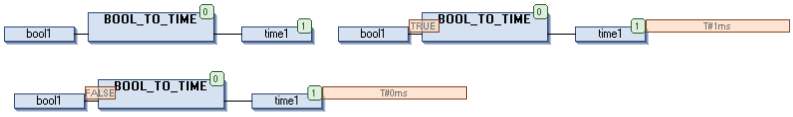
7.7.1布尔类型转换-4

操作符	BOOL_TO_REAL
功能说明	该功能块用作布尔类型数据向实数型类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为：布尔型(BOOL); 输出: 实数型(REAL);
变量声明	VAR bool1: BOOL; real1: REAL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	BOOL_TO_LREAL 用作布尔类型数据向长实数型类型数据的转换 若操作数为TRUE，结果为1；若操作数为FALSE，结果为0，如上所示。
使用限制	
注意事项	

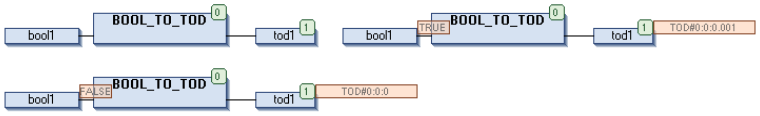
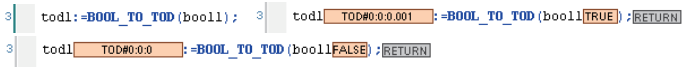
7.7.1布尔类型转换-5

操作符	BOOL_TO_DATE
功能说明	该功能块用作布尔类型数据向日期类型数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：布尔型(BOOL); 输出： 日期类型(DATE);
变量声明	VAR bool1: BOOL; date1: DATE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	"若操作数为FALSE，结果为D#1970-01-01;若操作数为TRUE，结果为D#1970-01-01."
使用限制	
注意事项	

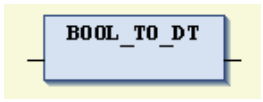
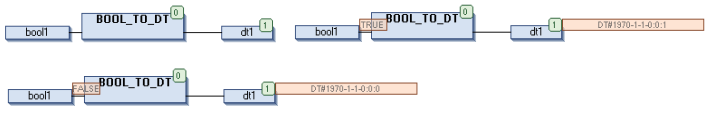
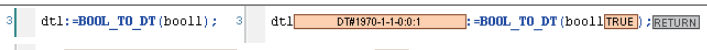
7.7.1布尔类型转换-6

操作符	BOOL_TO_TIME
功能说明	该功能块用作布尔类型数据向时间类型数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：布尔型(BOOL); 输出： 时间类型(TIME);
变量声明	VAR bool1: BOOL; time1: TIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	若操作数为FALSE，结果为0ms；若操作数为TRUE，结果为1ms. BOOL_TO_LTIME 用作布尔类型数据向长时间类型数据的转换 若操作数为FALSE，结果为0ns；若操作数为TRUE，结果为1ns，如上所示。
使用限制	
注意事项	

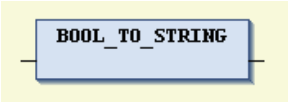
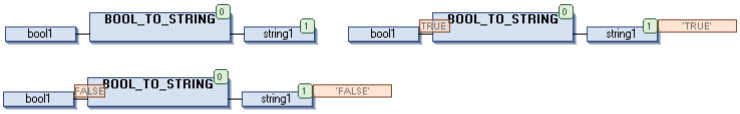
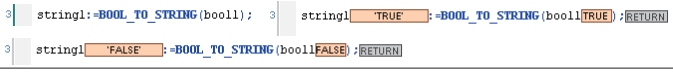
7.7.1布尔类型转换-7

操作符	BOOL_TO_TOD
功能说明	该功能块用作布尔类型数据向时刻类型数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：布尔型(BOOL); 输出： 时刻类型(TOD);
变量声明	VAR bool1: BOOL; tod1: TOD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	"若操作数为FALSE，结果为TOD#0:0:0.0；若操作数为TRUE，结果为TOD#0:0:0.001."
使用限制	
注意事项	

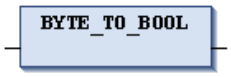
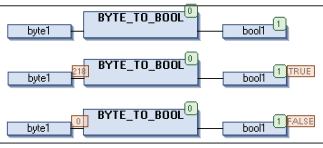
7.7.1布尔类型转换-8

操作符	BOOL_TO_DT
功能说明	该功能块用作布尔类型数据向日期时间类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 布尔型(BOOL); 输出: 日期时间类型(DT);
变量声明	VAR bool1: BOOL; dt1: DT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	"若操作数为FALSE，结果为DT#1970-1-1-0:0:0；若操作数为TRUE，结果为DT#1970-1-1-0:0:1."
使用限制	
注意事项	

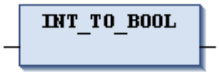
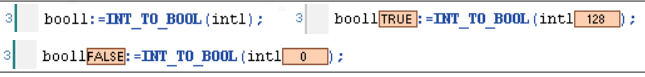
7.7.1布尔类型转换-9

操作符	BOOL_TO_STRING
功能说明	该功能块用作布尔类型数据向字符串类型数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：布尔型(BOOL); 输出： 字符串类型(STRING);
变量声明	VAR bool1: BOOL; string1: STRING; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	BOOL_TO_WSTRING 用作布尔类型数据向宽字符类型数据的转换 若操作数为TRUE，结果为“TRUE”；若操作数为FALSE，结果为“FALSE”。
使用限制	
注意事项	

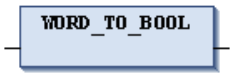
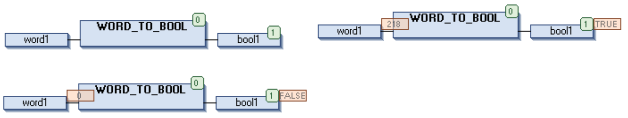
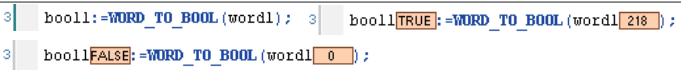
7.7.2转换为布尔类型-1

操作符	BYTE_TO_BOOL
功能说明	该功能块用作字节类型数据向布尔类型数据的转换
图形	
管脚定义	若操作数不为0，结果为TRUE；若操作数为0，结果为FALSE.
变量声明	VAR bool1: BOOL; byte1: BYTE; END_VAR
CFC语言示例	
ST语言示例	3 bool1:=BYTE_TO_BOOL(byte1); 3 bool1TRUE:=BYTE_TO_BOOL(byte1218); 3 bool1FALSE:=BYTE_TO_BOOL(byte10);
LD语言示例	
备注	
使用限制	
注意事项	

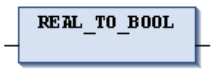
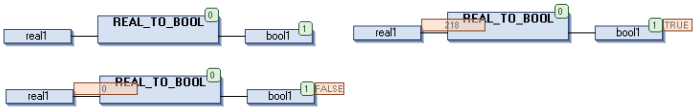
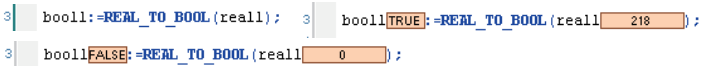
7.7.2转换为布尔类型-2

操作符	INT_TO_BOOL
功能说明	该功能块用整型类型数据向布尔类型数据的转换
图形	
管脚定义	若操作数不为0，结果为TRUE；若操作数为0，结果为FALSE.
变量声明	VAR bool1: BOOL; int1: INT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	DINT_TO_BOOL 用作双整型类型数据向布尔类型数据的转换 LINT_TO_BOOL 用作长整型类型数据向布尔类型数据的转换 SINT_TO_BOOL 用作短整型类型数据向布尔类型数据的转换 UINT_TO_BOOL 用作无符号整型类型数据向布尔类型数据的转换 UDINT_TO_BOOL 用作无符号双整型类型数据向布尔类型数据的转换 ULINT_TO_BOOL 用作无符号长整型类型数据向布尔类型数据的转换 USINT_TO_BOOL 用作无符号短整型类型数据向布尔类型数据的转换 若操作数不为0，结果为TRUE；若操作数为0，结果为FALSE.。
使用限制	
注意事项	

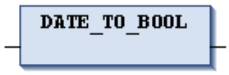
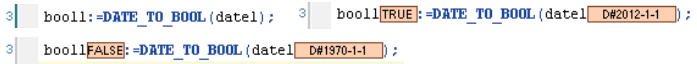
7.7.2转换为布尔类型-3

操作符	WORD_TO_BOOL
功能说明	该功能块用作字类型数据向布尔类型数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：字型(WORD); 输出： 布尔类型(BOOL);
变量声明	VAR bool1: BOOL; word1: word; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	DWORD_TO_BOOL 用作双字数据类型向布尔类型数据的转换 LWORD_TO_BOOL 用作长字数据类型向布尔类型数据的转换 若操作数不为0，结果为TRUE；若操作数为0，结果为FALSE。
使用限制	
注意事项	

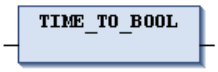
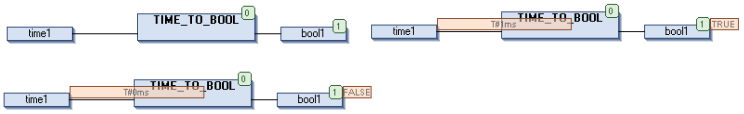
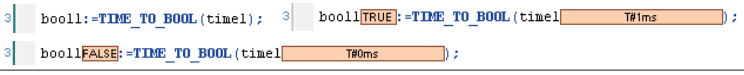
7.7.2转换为布尔类型-4

操作符	REAL_TO_BOOL
功能说明	该功能块用作实数类型数据向布尔类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为：实数型(REAL); 输出: 布尔类型(BOOL);
变量声明	VAR bool1: BOOL; real1: REAL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	LREAL_TO_BOOL 用作长实数类型数据向布尔类型数据的转换 若操作数不为0，结果为TRUE；若操作数为0，结果为FALSE。
使用限制	
注意事项	

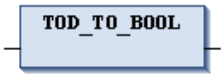
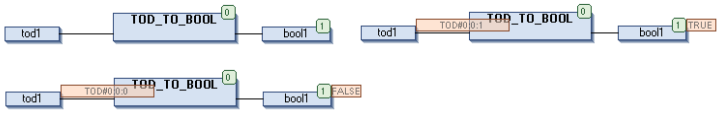
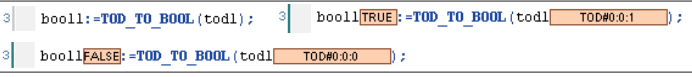
7.7.2转换为布尔类型-5

操作符	DATE_TO_BOOL
功能说明	该功能块用作日期类型数据向布尔类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为：日期类型(DATE); 输出: 布尔类型(BOOL);
变量声明	VAR bool1: BOOL; date1: DATE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	若操作数不为D#1970-1-1，结果为TRUE；若操作数为D#1970-1-1，结果为FALSE。 操作数日期必须大于D#1970-1-1,若操作数日期小于D#1970-1-1, 操作数会显示大于当前日期的一个数值，结果为TRUE。
使用限制	
注意事项	

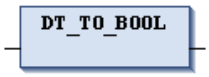
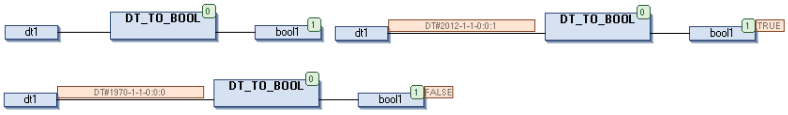
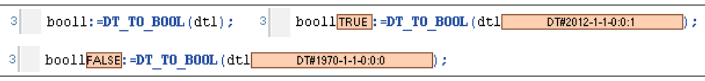
7.7.2转换为布尔类型-6

操作符	TIME_TO_BOOL
功能说明	该功能块用作时间类型数据向布尔类型数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：时间类型(TIME); 输出： 布尔类型(BOOL);
变量声明	VAR bool1: BOOL; time1: TIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	若操作数为0ms，结果为FALSE；若操作数不为0ms，结果为TRUE. LTIME_TO_BOOL 用作长时间类型数据向布尔类型数据的转换。 若操作数为0ns，结果为FALSE；若操作数不为0ns，结果为TRUE.
使用限制	
注意事项	

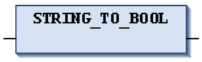
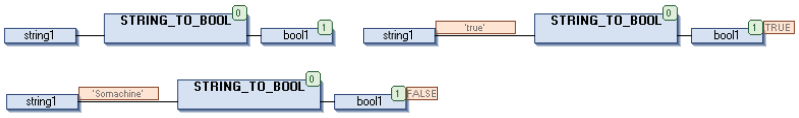
7.7.2转换为布尔类型-7

操作符	TOD_TO_BOOL
功能说明	该功能块用作时刻类型数据向布尔类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 时刻类型(TOD); 输出: 布尔类型(BOOL);
变量声明	VAR bool1: BOOL; tod1: TOD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	"若操作数为TOD#0:0:0，结果为FALSE；若操作数不为TOD#0:0:0，结果为TRUE."
使用限制	
注意事项	

7.7.2转换为布尔类型-8

操作符	DT _TO_BOOL
功能说明	该功能块用作日期时间类型数据向布尔类型数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：日期时间类型(DT); 输出： 布尔类型(BOOL);
变量声明	VAR bool1: BOOL; dt1: DT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	若操作数为DT#1970-1-1-0:0:0，结果为FALSE；若操作数不为DT#1970-1-1-0:0:0，结果为TRUE. 操作数日期必须大于DT#1970-1-1-0:0:0, 若操作数日期小于D#1970-1-1-0:0:0, 操作数会显示大于当前日期的一个数值，结果为TRUE。
使用限制	
注意事项	

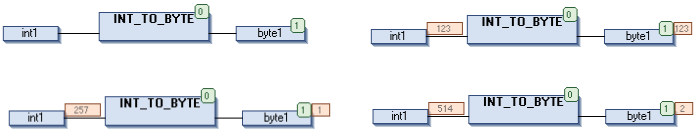
7.7.2转换为布尔类型-9

操作符	STRING_TO_BOOL
功能说明	该功能块用作字符串类型数据向布尔类型数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：字符串类型(STRING); 输出： 布尔类型(BOOL);
变量声明	VAR bool1: BOOL; string1: STRING; END_VAR
CFC语言示例	

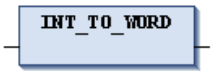
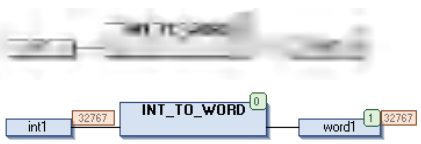
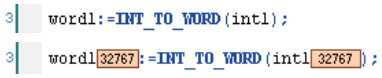
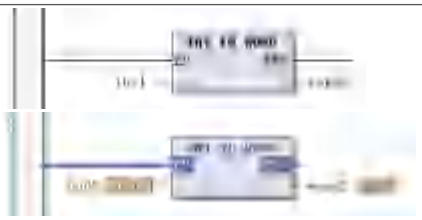
ST语言示例	bool1:=STRING_TO_BOOL(string1);
	bool1TRUE:=STRING_TO_BOOL(string1'true');
	bool1FALSE:='Somachine');

LD语言示例	
备注	若操作数为'TRUE'，结果为TRUE；若操作数不为'TRUE'，结果为FALSE. WSTRING_TO_BOOL 用作宽字符类型数据向布尔类型数据的转换 若操作数为"TRUE"，结果为TRUE；若操作数不为"TRUE"，结果为FALSE.
使用限制	
注意事项	

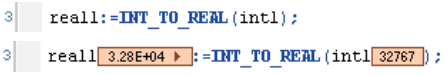
7.7.3整数类型之间的转换-1

操作符	INT_TO_BYTE
功能说明	该功能块用作整型类型数据向字节类型数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：整型(INT); 输出： 字节类型(BYTE);
变量声明	VAR int1: INT; byte1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	字节以256为一个单位，操作数大于256，结果为（操作数/256）取余数。 DINT_TO_BYTE 用作双整型类型数据向字节类型数据的转换 LINT_TO_BYTE 用作长整型类型数据向字节类型数据的转换 SINT_TO_BYTE 用作短整型类型数据向字节类型数据的转换 UINT_TO_BYTE 用作无符号整型类型数据向字节类型数据的转换 UDINT_TO_BYTE 用作无符号双整型类型数据向字节类型数据的转换 ULINT_TO_BYTE 用作无符号长整型类型数据向字节类型数据的转换 USINT_TO_BYTE 用作无符号短整型类型数据向字节类型数据的转换
使用限制	
注意事项	从较大的数据类型转换为较小的数据类型时，有可能丢失部分信息。 如果有符号数据类型向无符号数据类型转换时，先把有符号数据类型转换为补码的形式，再进行数据转换。 如果需转换的数超过了数值范围，结果则取相应类型的字节数，忽略超出数值范围的高字节数。

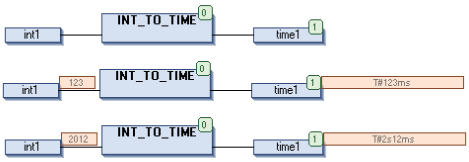
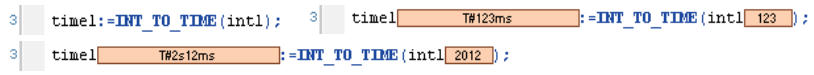
7.7.3整数类型之间的转换-2

操作符	INT_TO_WORD
功能说明	该功能块用整型类型数据向字类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为：整型(INT); 输出: 字类型(WORD);
变量声明	VAR int1: INT; word1: WORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	INT_TO_LWORD、UINT_TO_LWORD 用作整型类型数据、无符号整型类型数据向长字类型数据的转换 INT_TO_DWORD、UINT_TO_DWORD 用作整型类型数据、无符号整型类型数据向双字类型数据的转换 DINT_TO_WORD、UDINT_TO_WORD 用作双整型类型数据、无符号双整型类型数据向字类型数据的转换 DINT_TO_LWORD、UDINT_TO_LWORD 用作双整型类型数据、无符号双整型类型数据向长字类型数据的转换 DINT_TO_DWORD、UDINT_TO_DWORD 用作双整型类型数据、无符号双整型类型数据向双字类型数据的转换 LINT_TO_WORD、ULINT_TO_WORD 用作长整型类型数据、无符号长整型类型数据向字类型数据的转换 LINT_TO_LWORD、ULINT_TO_LWORD 用作长整型类型数据、无符号长整型类型数据向长字类型数据的转换 LINT_TO_DWORD、 ULINT_TO_DWORD用作长整型类型数据、无符号长整型类型数据向双字类型数据的转换 SINT_TO_WORD、USINT_TO_WORD 用作短整型类型数据、无符号短整型类型数据向字类型数据的转换 SINT_TO_LWORD、USINT_TO_LWORD 用作短整型类型数据、无符号短整型类型数据向长字类型数据的转换 SINT_TO_DWORD、USINT_TO_DWORD 用作短整型类型数据、无符号短整型类型数据向双字类型数据的转换
使用限制	
注意事项	从较大的数据类型转换为较小的数据类型时，有可能丢失部分信息。 如果有符号数据类型向无符号数据类型转换时，先把有符号数据类型转换为补码的形式，再进行数据转换。 如果需转换的数超过了数值范围，结果则取相应类型的字节数，忽略超出数值范围的高字节数。

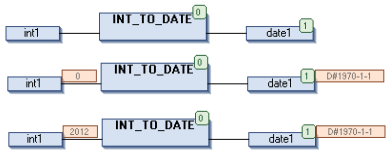
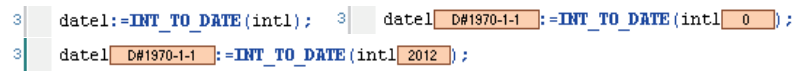
7.7.3整数类型之间的转换-3

操作符	INT_TO_REAL
功能说明	该功能块用作整型类型数据向实数类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 整型(INT); 输出: 实数类型(REAL);
变量声明	VAR int1: INT; real1: REAL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	INT_TO_LREAL、UINT_TO_LREAL 用作整型类型数据、无符号整型类型数据向长实数类型数据的转换 DINT_TO_REAL、UDINT_TO_REAL 用作双整型类型数据、无符号双整型类型数据向实数类型数据的转换 DINT_TO_LREAL、UDINT_TO_LREAL 用作双整型类型数据、无符号双整型类型数据向长实数类型数据的转换 LINT_TO_REAL、ULINT_TO_REAL 用作长整型类型数据、无符号长整型类型数据向实数类型数据的转换 LINT_TO_LREAL、ULINT_TO_LREAL 用作长整型类型数据、无符号长整型类型数据向长实数类型数据的转换 SINT_TO_REAL、USINT_TO_REAL 用作短整型类型数据、无符号短整型类型数据向实数类型数据的转换 SINT_TO_LREAL、USINT_TO_LREAL 用作短整型类型数据、无符号短整型类型数据向长实数类型数据的转换
使用限制	
注意事项	从较大的数据类型转换为较小的数据类型时，有可能丢失部分信息。 如果有符号数据类型向无符号数据类型转换时，先把有符号数据类型转换为补码的形式，再进行数据转换。 如果需转换的数超过了数值范围，结果则取相应类型的字节数，忽略超出数值范围的高字节数。

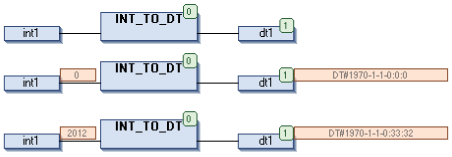
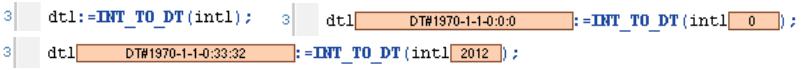
7.7.3整数类型之间的转换-4

操作符	INT_TO_TIME
功能说明	该功能块用作整型类型数据向时间类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为：整型(INT); 输出: 时间类型(TIME); 时间将以毫秒为单位
变量声明	VAR int1: INT; time1: TIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	INT_TO_TIME、UINT_TO_TIME 用作整型、无符号整型类型数据向时间类型数据的转换，结果为**m**s**ms。 INT_TO_LTIME、UINT_TO_LTIME 用作整型、无符号整型类型数据向长时间类型数据的转换，结果为**ms**us**ns。 如 int:=2012,则time1:=2us12ns. DINT_TO_TIME、UDINT_TO_TIME 用作双整型、无符号双整型类型数据向时间类型数据的转换，结果为**m**s**ms。 DINT_TO_LTIME、UDINT_TO_LTIME 用作双整型、无符号双整型类型数据向长时间类型数据的转换，结果为**ms**us**ns。 LINT_TO_TIME、ULINT_TO_TIME 用作长整型、无符号长整型类型数据向时间类型数据的转换，结果为**m**s**ms。 LINT_TO_LTIME、ULINT_TO_LTIME 用作长整型、无符号长整型类型数据向长时间类型数据的转换，结果为**ms**us***ns。 SINT_TO_TIME、USINT_TO_TIME 用作短整型、无符号短整型类型数据向时间类型数据的转换，结果为***m***s***ms。 SINT_TO_LTIME、USINT_TO_LTIME 用作短整型、无符号短整型类型数据向长时间类型数据的转换，结果为**ms**us***ns。
使用限制	
注意事项	从较大的数据类型转换为较小的数据类型时，有可能丢失部分信息。 如果有符号数据类型向无符号数据类型转换时，先把有符号数据类型转换为补码的形式，再进行数据转换。 如果需转换的数超过了数值范围，结果则取相应类型的字节数，忽略超出数值范围的高字节数。

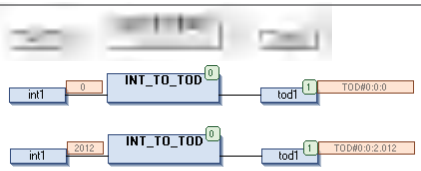
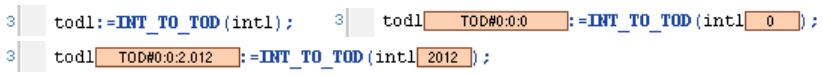
7.7.3整数类型之间的转换-5

操作符	INT_TO_DATE
功能说明	该功能块用作整型类型数据向日期数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 整型(INT); 输出: 日期类型(DATE);
变量声明	VAR int1: INT; date1: DATE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	SINT_TO_DATE、USINT_TO_DATE 用作短整型、无符号短整型类型数据向日期类型数据的转换 LINT_TO_DATE、ULINT_TO_DATE 用作长整型、无符号长整型类型数据向日期类型数据的转换 DINT_TO_DATE、UDINT_TO_DATE 用作双整型、无符号双整型类型数据向日期类型数据的转换 UINT_TO_DATE 用作无符号整型类型数据向日期类型数据的转换
使用限制	
注意事项	从较大的数据类型转换为较小的数据类型时，有可能丢失部分信息。 如果有符号数据类型向无符号数据类型转换时，先把有符号数据类型转换为补码的形式，再进行数据转换。 如果需转换的数超过了数值范围，结果则取相应类型的字节数，忽略超出数值范围的高字节数。

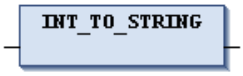
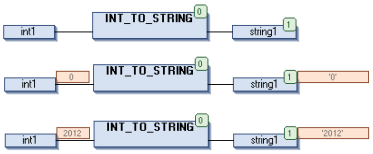
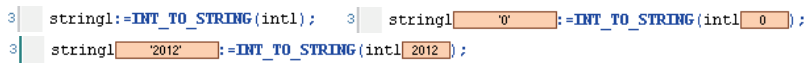
7.7.3整数类型之间的转换-6

操作符	INT_TO_DT
功能说明	该功能块用作整型类型数据向日期时间类型数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：整型(INT); 输出： 日期时间类型(DT);
变量声明	VAR int1: INT; dt1: DT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	日期以秒为单位，从1970年1月1日起存储在内部。 LINT_TO_DT、ULINT_TO_DT 用作长整型、无符号长整型类型数据向时间日期类型数据的转换。 DINT_TO_DT、UDINT_TO_DT 用作双整型、无符号双整型类型数据向时间日期类型数据的转换。 SINT_TO_DT、USINT_TO_DT 用作短整型、无符号短整型类型数据向时间日期类型数据的转换。
使用限制	
注意事项	

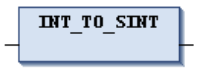
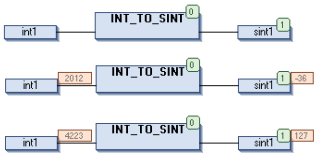
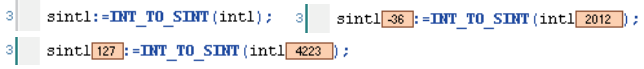
7.7.3整数类型之间的转换-7

操作符	INT_TO_TOD
功能说明	该功能块用作整型类型数据向时刻类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 整型(INT); 输出: 时刻类型(TOD);
变量声明	VAR int1: INT; tod1: TOD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	时刻以豪秒为单位。 LINT_TO_TOD、ULINT_TO_TOD 用作长整型、无符号长整型类型数据向时刻类型数据的转换。 DINT_TO_TOD、UDINT_TO_TOD 用作双整型、无符号双整型类型数据向时刻类型数据的转换。 SINT_TO_TOD、USINT_TO_TOD 用作短整型、无符号短整型类型数据向时刻类型数据的转换。
使用限制	
注意事项	

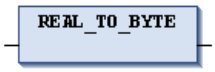
7.7.3整数类型之间的转换-8

操作符	INT _TO_STRING
功能说明	该功能块用作整型类型数据向字符串类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 整型(INT); 输出: 字符串类型(STRING);
变量声明	VAR int1: INT; string1: STRING; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	LINT_TO_STRING、ULINT_TO_STRING 用作长整型、无符号长整型类型数据向字符串类型数据的转换。 DINT_TO_STRING、UDINT_TO_STRING 用作双整型、无符号双整型类型数据向字符串类型数据的转换。 SINT_TO_STRING、USINT_TO_STRING 用作短整型、无符号短整型类型数据向字符串类型数据的转换。 LINT_TO_WSTRING、ULINT_TO_WSTRING 用作长整型、无符号长整型类型数据向短字符串类型数据的转换。 DINT_TO_WSTRING、UDINT_TO_WSTRING 用作双整型、无符号双整型类型数据向短字符串类型数据的转换。 SINT_TO_WSTRING、USINT_TO_WSTRING 用作短整型、无符号短整型类型数据向短字符串类型数据的转换。
使用限制	
注意事项	

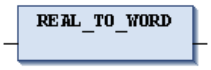
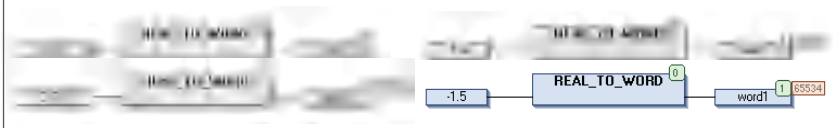
7.7.3整数类型之间的转换-9

操作符	INT _TO_SINT
功能说明	该功能块用作整型类型数据向短整型类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为：整型(INT); 输出: 短整型(SINT);
变量声明	VAR int1: INT; sint1: SINT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	INT_TO_LINT、INT_TO_DINT 用作整型类型数据向长整型、双整型类型数据的转换。 INT_TO_USINT、INT_TO_ULINT、INT_TO_UDINT 用作整型类型数据向无符号短整型、无符号长整型、无符号双整型类型数据的转换。
使用限制	
注意事项	从较大的数据类型转换为较小的数据类型时，有可能丢失部分信息。 如果有符号数据类型向无符号数据类型转换时，先把有符号数据类型转换为补码的形式，再进行数据转换。 如果需转换的数超过了数值范围，结果则取相应类型的字节数，忽略超出数值范围的高字节数。

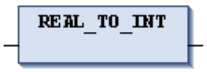
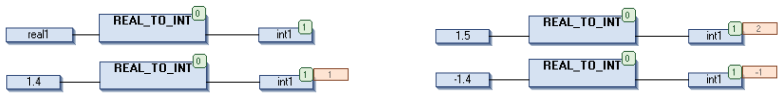
7.7.4实数/长实数类型的转换-1

操作符	REAL_TO_BYTE
功能说明	该功能块用作实数类型数据向字节类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为：实数型(REAL); 输出: 字节类型(BYTE);
变量声明	VAR real1: REAL; byte1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	操作数将被四舍五入为近似的整数值，然后转换成字节变量类型。 LREAL_TO_BYTE 用作长实数类型数据向字节类型数据的转换
使用限制	
注意事项	如果 REAL 或 LREAL 类型转换成BYTE类型，且实型数据的值超出了字节的范围，结果将会是不确定的并且该值取决于目标系统。这种情况即使产生一个异常也是可能的! 为了获取与目标无关的代码，应由应用程序进行值域越界处理。如果real/lreal 型数据在字节的值域范围内，他们之间的转换在所有系统上都可以进行。

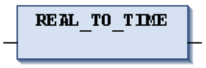
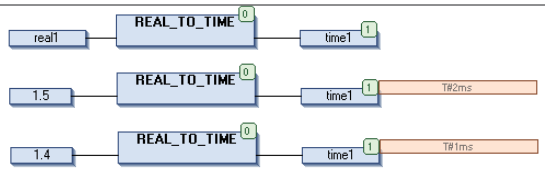
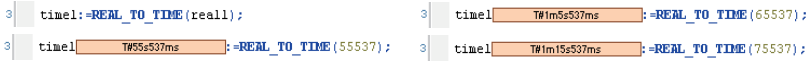
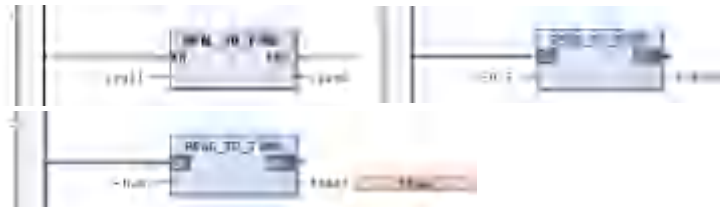
7.7.4实数/长实数类型的转换-2

操作符	REAL_TO_WORD
功能说明	该功能块用作实数类型数据向字类型数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：实数型(REAL); 输出： 字类型(WORD);
变量声明	VAR real1: REAL; word1: WORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	操作数将被四舍五入为近似的整数值，然后转换成字变量类型。 LREAL_TO_WORD、LREAL_TO_DWORD、LREAL_TO_LWORD 用作长实数类型数据向字、双字、长字类型数据的转换 REAL_TO_DWORD、REAL_TO_LWORD 用作实数类型数据向双字、长字类型数据的转换。
使用限制	
注意事项	如果 REAL 或 LREAL 类型转换成WORD类型，且实型数据的值超出了字的范围，结果将会是不确定的并且该值取决于目标系统。这种情况即使产生一个异常也是可能的! 为了获取与目标无关的代码，应由应用程序进行值域越界处理。如果real/lreal 型数据在字的值域范围内，他们之间的转换在所有系统上都可以进行。

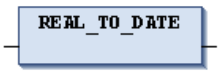
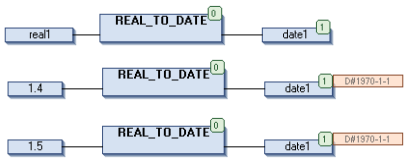
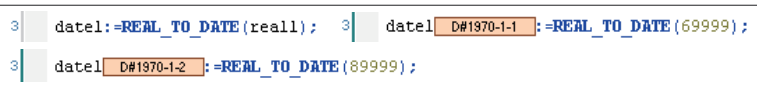
7.7.4实数/长实数类型的转换-3

操作符	REAL_TO_INT
功能说明	该功能块用作实数类型数据向整数类型数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：实数型(REAL); 输出： 整型(INT);
变量声明	VAR real1: REAL; int1: INT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	操作数将被四舍五入为近似的整数值，然后转换成整数变量类型。 REAL_TO_SINT、REAL_TO_USINT 用作实数类型数据向短整型、无符号短整数类型数据的转换 REAL_TO_DINT、REAL_TO_UDINT 用作实数类型数据向双整型、无符号双整数类型数据的转换 REAL_TO_LINT、REAL_TO_ULINT 用作实数类型数据向长整型、无符号长整数类型数据的转换 LREAL_TO_SINT、LREAL_TO_USINT 用作长实数类型数据向短整型、无符号短整数类型数据的转换 LREAL_TO_DINT、LREAL_TO_UDINT 用作长实数类型数据向双整型、无符号双整数类型数据的转换 LREAL_TO_LINT、LREAL_TO_ULINT 用作长实数类型数据向长整型、无符号长整数类型数据的转换
使用限制	
注意事项	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。这种情况即使产生一个异常也是可能的! 为了获取与目标无关的代码，应由应用程序进行值域越界处理。如果real/lreal型数据在整形的值域范围内，他们之间的转换在所有系统上都可以进行。

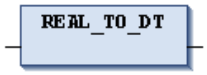
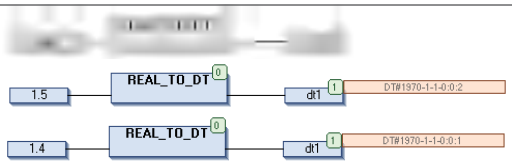
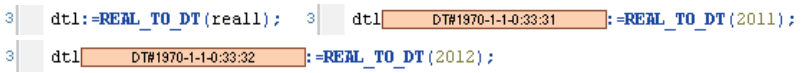
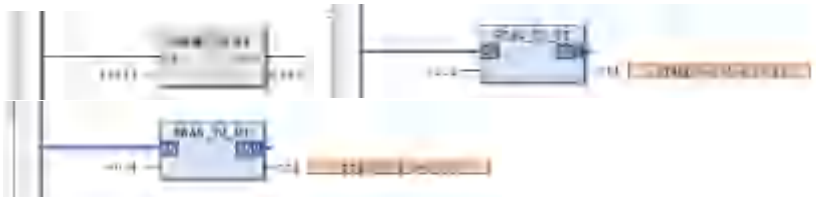
7.7.4实数/长实数类型的转换-4

操作符	REAL_TO_TIME
功能说明	该功能块用作实数类型数据向时间类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 实数型(REAL); 输出: 时间类型(TIME);
变量声明	VAR real1: REAL; time1: TIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	操作数将被四舍五入为近似的整数值，然后转换成时间变量类型。时间变量以毫秒为单位。 REAL_TO_LTIME 用作实数类型数据向长时间类型数据的转换. LREAL_TO_TIME、LREAL_TO_LTIME 用作长实数类型数据向时间、长时间类型数据的转换.
使用限制	
注意事项	如果 REAL 或 LREAL 类型转换成TIME类型，且实型数据的值超出了时间的范围，结果将会是不确定的并且该值取决于目标系统。这种情况即使产生一个异常也是可能的! 为了获取与目标无关的代码，应由应用程序进行值域越界处理。如果real/lreal 型数据在时间的值域范围内，他们之间的转换在所有系统上都可以进行。

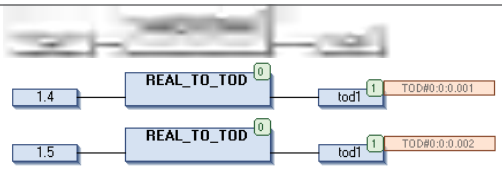
7.7.4实数/长实数类型的转换-5

操作符	REAL_TO_DATE
功能说明	该功能块用作实数类型数据向日期数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：实数型(REAL); 输出： 日期类型(DATE);
变量声明	VAR real1: REAL; date1: DATE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	操作数将被四舍五入为近似的整数值，然后转换成日期变量类型。 LREAL_TO_DATE 用作长实数类型数据向日期类型数据的转换。
使用限制	
注意事项	如果 REAL 或 LREAL 类型转换成DATE类型，且实型数据的值超出了日期的范围，结果将会是不确定的并且该值取决于目标系统。这种情况即使产生一个异常也是可能的! 为了获取与目标无关的代码，应由应用程序进行值域越界处理。如果real/lreal 型数据在日期的值域范围内，他们之间的转换在所有系统上都可以进行。

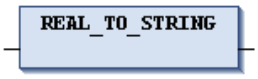
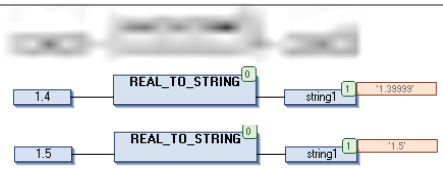
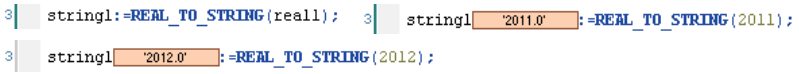
7.7.4实数/长实数类型的转换-6

操作符	REAL_TO_DT
功能说明	该功能块用作实数类型数据向日期时间类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 实数型(REAL); 输出: 日期时间类型(DT);
变量声明	VAR real1: REAL; dt1: DT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	操作数将被四舍五入为近似的整数值，然后转换成时间日期变量类型。时间日期变量以秒为单位，从1970年1月1日起存储在内部。 LREAL_TO_DT 用作长实数类型数据向时间日期类型数据的转换。
使用限制	
注意事项	

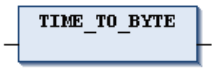
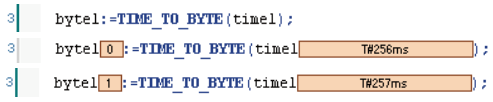
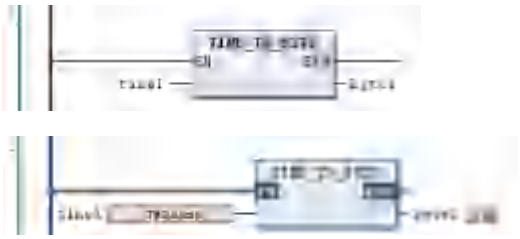
7.7.4实数/长实数类型的转换-7

操作符	REAL_TO_TOD
功能说明	该功能块用作实数类型数据向时刻类型数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：实数型(REAL); 输出： 时刻类型(TOD);
变量声明	VAR real1: REAL; tod1: TOD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	操作数将被四舍五入为近似的整数值，然后转换成时间日期变量类型。时刻以豪秒为单位。 LREAL_TO_TOD 用作长实数类型数据向时刻类型数据的转换。
使用限制	
注意事项	如果 REAL 或 LREAL 类型转换成TOD类型，且实型数据的值超出了时刻的范围，结果将会是不确定的并且该值取决于目标系统。这种情况即使产生一个异常也是可能的! 为了获取与目标无关的代码，应由应用程序进行值域越界处理。如果real/lreal 型数据在时刻的值域范围内，他们之间的转换在所有系统上都可以进行。

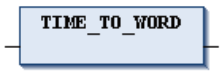
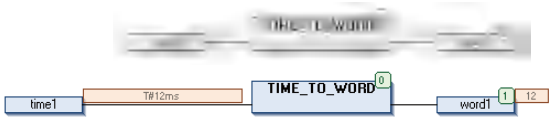
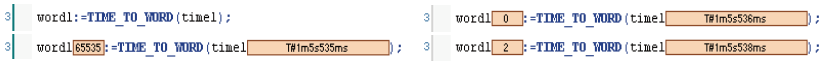
7.7.4实数/长实数类型的转换-8

操作符	REAL _TO_STRING
功能说明	该功能块用作实数类型数据向字符串类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 实数型(REAL); 输出: 字符串类型(STRING);
变量声明	VAR real1: REAL; string1: STRING; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	LREAL_TO_STRING 用作长实数类型数据向字符串类型数据的转换。 REAL_TO_WSTRING 用作实数类型数据向短字符串类型数据的转换。
使用限制	
注意事项	

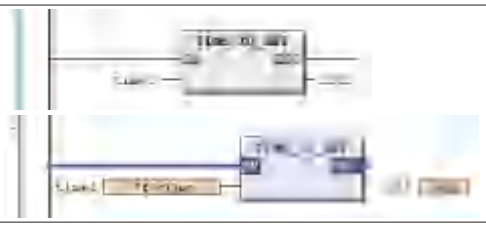
7.7.5时间/时刻类型转换-1

操作符	TIME_TO_BYTE
功能说明	该功能块用作时间类型数据向字节类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 时间类型(TIME); 输出: 字节类型(BYTE);
变量声明	VAR time1: TIME; byte1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	LTIME_TO_BYTE 用作长时间类型数据向字节类型数据的转换
使用限制	
注意事项	

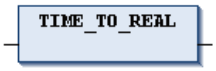
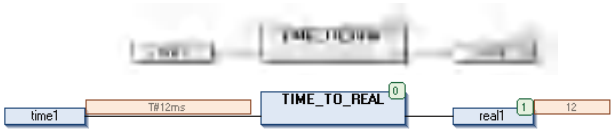
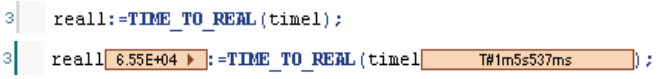
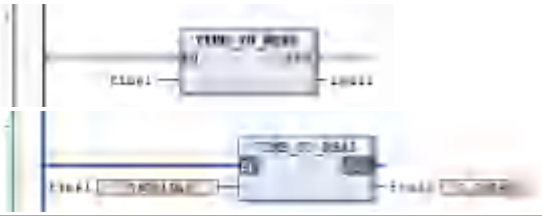
7.7.5时间/时刻类型转换-2

操作符	TIME_TO_WORD
功能说明	该功能块用作时间类型数据向字类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 时间类型(TIME); 输出: 字类型(WORD);
变量声明	VAR time1: TIME; word1: WORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	TIME_TO_LWORD、TIME_TO_DWORD 用作时间类型数据向长字、双字类型数据的转换 LTIME_TO_WORD、LTIME_TO_LWORD、LTIME_TO_DWORD 用作长时间类型数据向字、长字、双字类型数据的转换。
使用限制	
注意事项	

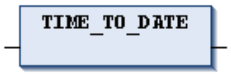
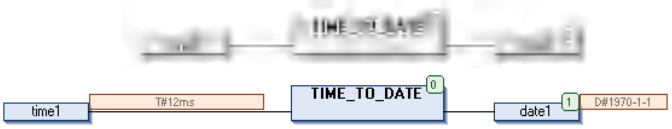
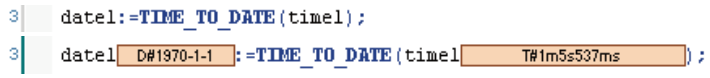
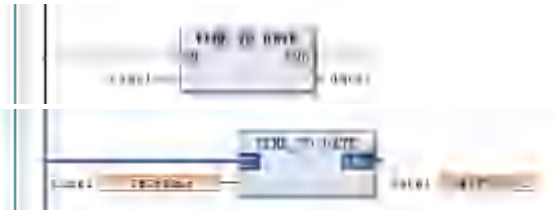
7.7.5时间/时刻类型转换-3

操作符	TIME_TO_INT
功能说明	该功能块用作时间类型数据向整数类型数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：时间类型(TIME); 输出： 整数类型(INT);
变量声明	VAR time1: TIME; int1: INT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	TIME_TO_SINT、TIME_TO_USINT 用作时间类型数据向短整型、无符号短整数类型数据的转换 TIME_TO_DINT、TIME_TO_UDINT 用作时间类型数据向双整型、无符号双整数类型数据的转换 TIME_TO_LINT、TIME_TO_ULINT 用作时间类型数据向长整型、无符号长整数类型数据的转换 LTIME_TO_SINT、LTIME_TO_USINT 用作长时间类型数据向短整型、无符号短整数类型数据的转换 LTIME_TO_DINT、LTIME_TO_UDINT 用作长时间类型数据向双整型、无符号双整数类型数据的转换 LTIME_TO_LINT、LTIME_TO_ULINT 用作长时间类型数据向长整型、无符号长整数类型数据的转换
使用限制	
注意事项	

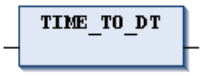
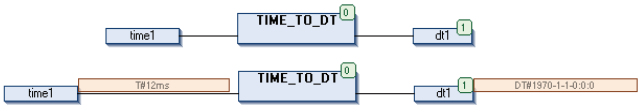
7.7.5时间/时刻类型转换-4

操作符	TIME_TO_REAL
功能说明	该功能块用作时间类型数据向实数类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 时间类型(TIME); 输出: 实数类型(REAL);
变量声明	VAR time1: TIME; real1: REAL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	TIME_TO_LREAL 用作时间类型数据向长实数类型数据的转换. LTIME_TO_REAL、LTIME_TO_LREAL 用作长时间类型数据向实数、长实数类型数据的转换.
使用限制	
注意事项	

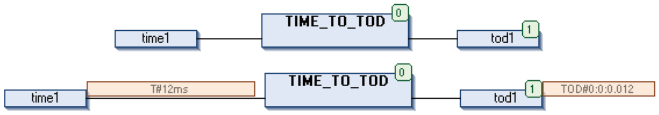
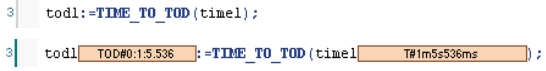
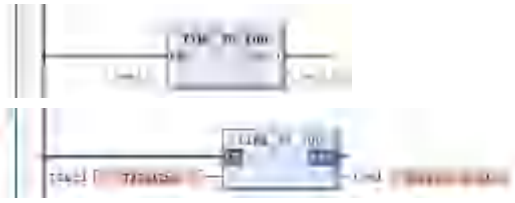
7.7.5时间/时刻类型转换-5

操作符	TIME_TO_DATE
功能说明	该功能块用作时间类型数据向日期数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 时间类型(TIME); 输出: 日期类型(DATE);
变量声明	VAR time1: TIME; date1: DATE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	LTIME_TO_DATE 用作长时间类型数据向日期类型数据的转换。
使用限制	
注意事项	

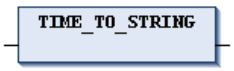
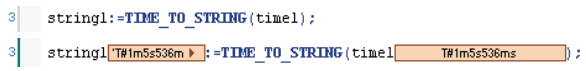
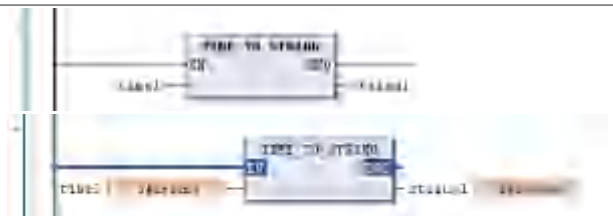
7.7.5时间/时刻类型转换-6

操作符	TIME_TO_DT
功能说明	该功能块用作时间类型数据向日期时间类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 时间类型(TIME); 输出: 日期时间类型(DT);
变量声明	VAR time1: TIME; dt1: DT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	LTIME_TO_DT 用作长时间类型数据向日期时间类型数据的转换。
使用限制	
注意事项	

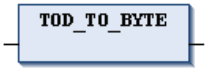
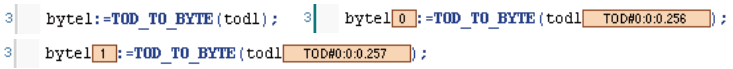
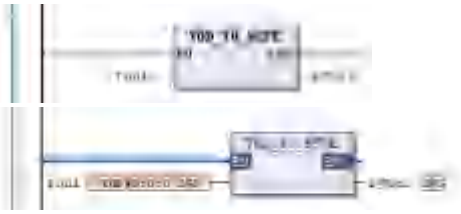
7.7.5时间/时刻类型转换-7

操作符	TIME_TO_TOD
功能说明	该功能块用作时间类型数据向时刻类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 时间类型(TIME); 输出: 时刻类型(TOD);
变量声明	VAR time1: TIME; tod1: TOD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	LTIME_TO_TOD 用作长时间类型数据向时刻类型数据的转换。
使用限制	
注意事项	

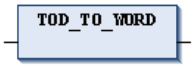
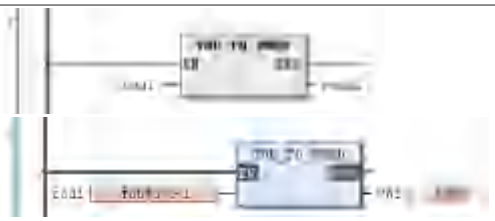
7.7.5时间/时刻类型转换-8

操作符	TIME _TO_STRING
功能说明	该功能块用作时间类型数据向字符串类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 时间类型(TIME); 输出: 字符串类型(STRING);
变量声明	VAR time1: TIME; string1: STRING; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	LTIME_TO_STRING 用作长时间类型数据向字符串类型数据的转换。 TIME_TO_WSTRING 用作时间类型数据向短字符串类型数据的转换。
使用限制	
注意事项	

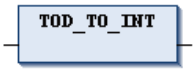
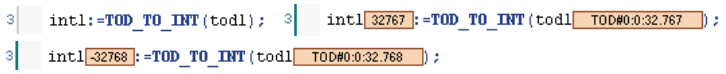
7.7.5时间/时刻类型转换-9

操作符	TOD_TO_BYTE
功能说明	该功能块用作时刻类型数据向字节类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 时刻类型(TOD); 输出: 字节类型(BYTE);
变量声明	VAR tod1: TOD; byte1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	
使用限制	
注意事项	

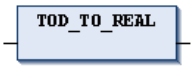
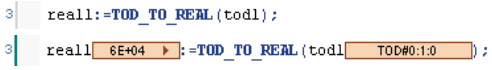
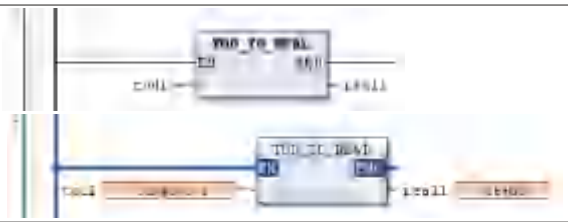
7.7.5时间/时刻类型转换-10

操作符	TOD_TO_WORD
功能说明	该功能块用作时刻类型数据向字类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 时刻类型(TOD); 输出: 字类型(WORD);
变量声明	VAR tod1: TOD; word1: WORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	TOD_TO_LWORD、TOD_TO_DWORD 用作时刻类型数据向长字、双字类型数据的转换。
使用限制	
注意事项	

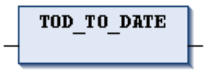
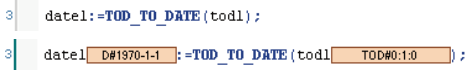
7.7.5时间/时刻类型转换-11

操作符	TOD_TO_INT
功能说明	该功能块用作时刻类型数据向整数类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 时刻类型(TOD); 输出: 整型(INT);
变量声明	VAR tod1: TOD; int1: INT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	TOD_TO_SINT、TOD_TO_USINT 用作时刻类型数据向短整型、无符号短整数类型数据的转换 TOD_TO_DINT、TOD_TO_UDINT 用作时刻类型数据向双整型、无符号双整数类型数据的转换 TOD_TO_LINT、TOD_TO_ULINT 用作时刻类型数据向长整型、无符号长整数类型数据的转换
使用限制	
注意事项	

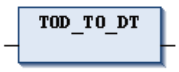
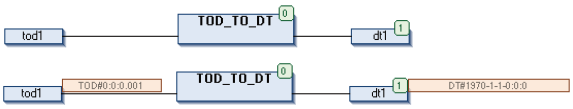
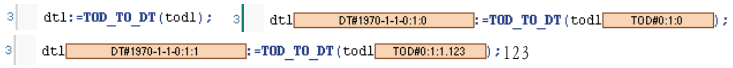
7.7.5时间/时刻类型转换-12

操作符	TOD_TO_REAL
功能说明	该功能块用作时刻类型数据向实数类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 时刻类型(TOD); 输出: 实数型(REAL);
变量声明	VAR tod1: TOD; real1: REAL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	TOD_TO_LREAL 用作时刻类型数据向长实数类型数据的转换.
使用限制	
注意事项	

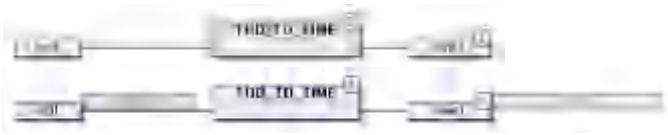
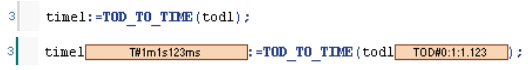
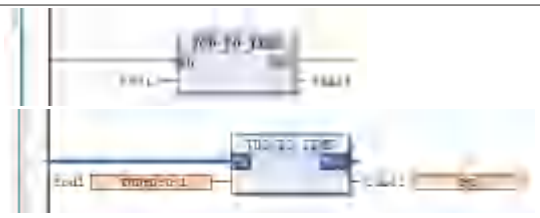
7.7.5时间/时刻类型转换-13

操作符	TOD_TO_DATE
功能说明	该功能块用作时刻类型数据向日期数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 时刻类型(TOD); 输出: 日期类型(DATE);
变量声明	VAR tod1: TOD; date1: DATE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	
使用限制	
注意事项	

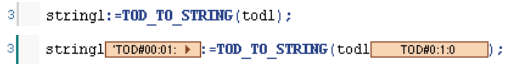
7.7.5时间/时刻类型转换-14

操作符	TOD_TO_DT
功能说明	该功能块用作时刻类型数据向时间日期类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 时刻类型(TOD); 输出: 时间日期类型(DT);
变量声明	VAR tod1: TOD; dt1: DT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	
使用限制	
注意事项	

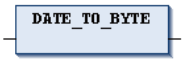
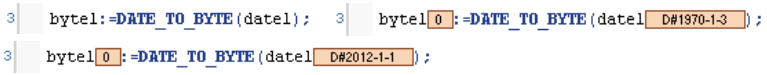
7.7.5时间/时刻类型转换-15

操作符	TOD_TO_TIME
功能说明	该功能块用作时刻类型数据向时间类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 时刻类型(TOD); 输出: 时间类型(TIME);
变量声明	VAR tod1: TOD; time1: TIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	TOD_TO_LTIME 用作时刻类型数据向长时间类型数据的转换。
使用限制	
注意事项	

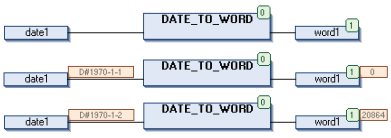
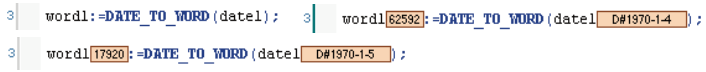
7.7.5时间/时刻类型转换-16

操作符	TOD _TO_STRING
功能说明	该功能块用作时刻类型数据向字符串类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 时刻类型(TOD); 输出: 字符串类型(STRING);
变量声明	VAR tod1: TOD; string1: STRING; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	TOD_TO_WSTRING 用作时刻类型数据向短字符串类型数据的转换。
使用限制	
注意事项	

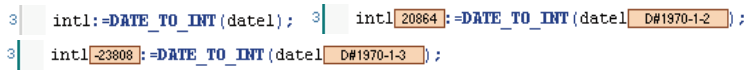
7.7.6日期/日期时间类型转换-1

操作符	DATE_TO_BYTE
功能说明	该功能块用作日期类型数据向字节类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 日期类型(DATE); 输出: 字节类型(BYTE);
变量声明	VAR date1: DATE; byte1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	
使用限制	
注意事项	

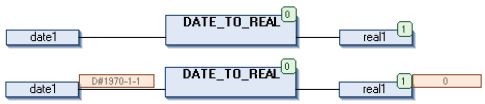
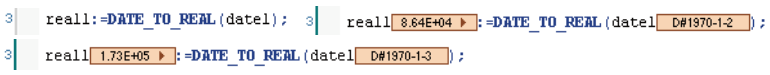
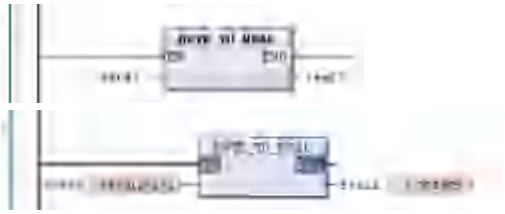
7.7.6日期/日期时间类型转换-2

操作符	DATE_TO_WORD
功能说明	该功能块用作日期类型数据向字类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 日期类型(DATE); 输出: 字类型(WORD);
变量声明	VAR date1: DATE; word1: WORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	DATE_TO_LWORD、TIME_TO_DWORD 用作日期类型数据向长字、双字类型数据的转换。
使用限制	
注意事项	

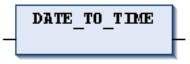
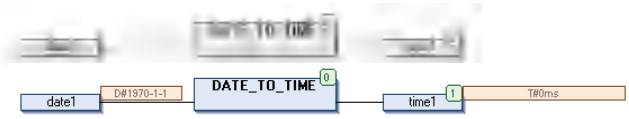
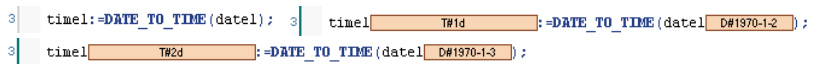
7.7.6日期/日期时间类型转换-3

操作符	DATE_TO_INT
功能说明	该功能块用作日期类型数据向整数类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 日期类型(DATE); 输出: 整数类型(INT);
变量声明	VAR date1: DATE; int1: INT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	DATE_TO_SINT、DATE_TO_USINT 用作日期类型数据向短整型、无符号短整数类型数据的转换 DATE_TO_DINT、DATE_TO_UDINT 用作日期类型数据向双整型、无符号双整数类型数据的转换 DATE_TO_LINT、DATE_TO_ULINT 用作日期类型数据向长整型、无符号长整数类型数据的转换
使用限制	
注意事项	

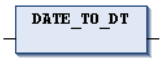
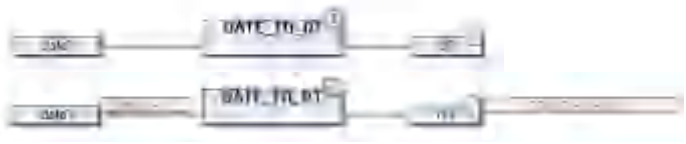
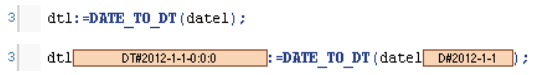
7.7.6日期/日期时间类型转换-4

操作符	DATE_TO_REAL
功能说明	该功能块用作日期类型数据向实数类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 日期类型(DATE); 输出: 实数类型(REAL);
变量声明	VAR date1: DATE; real1: REAL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	DATE_TO_LREAL 用作日期类型数据向长实数类型数据的转换.
使用限制	
注意事项	

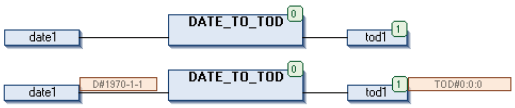
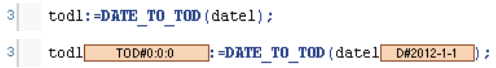
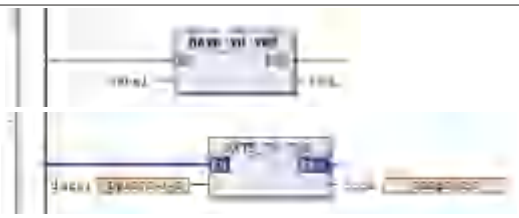
7.7.6日期/日期时间类型转换-5

操作符	DATE_TO_TIME
功能说明	该功能块用作日期类型数据向时间数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 日期类型(DATE); 输出: 时间类型(TIME);
变量声明	VAR date1: DATE; time1: TIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	DATE_TO_LTIME 用作日期类型数据向长时间类型数据的转换。
使用限制	
注意事项	

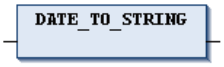
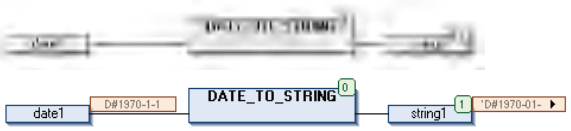
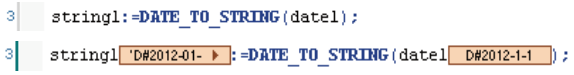
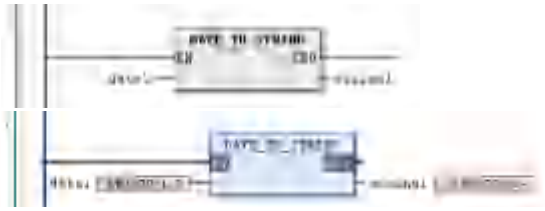
7.7.6日期/日期时间类型转换-6

操作符	DATE_TO_DT
功能说明	该功能块用作日期类型数据向日期时间类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 日期类型(DATE); 输出: 日期时间类型(DT);
变量声明	VAR date1: DATE; dt1: DT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	
使用限制	
注意事项	

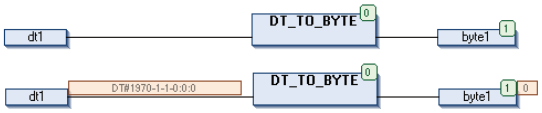
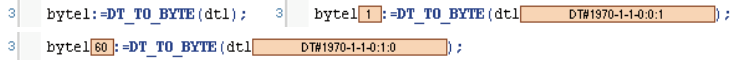
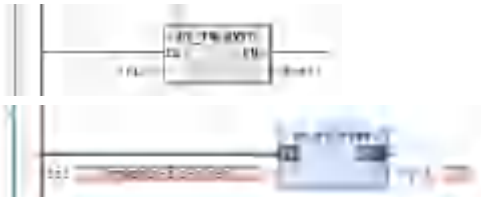
7.7.6日期/日期时间类型转换-7

操作符	DATE_TO_TOD
功能说明	该功能块用作日期类型数据向时刻类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 日期类型(DATE); 输出: 时刻类型(TOD);
变量声明	VAR date1: DATE; tod1: TOD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	
使用限制	
注意事项	

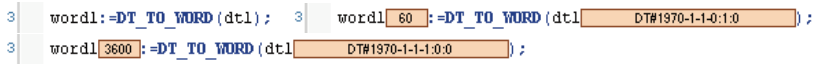
7.7.6日期/日期时间类型转换-8

操作符	DATE _TO_STRING
功能说明	该功能块用作日期类型数据向字符串类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 日期类型(DATE); 输出: 字符串类型(STRING);
变量声明	VAR date1: DATE; string1: STRING; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	DATE_TO_WSTRING 用作日期类型数据向短字符串类型数据的转换。
使用限制	
注意事项	

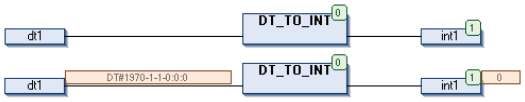
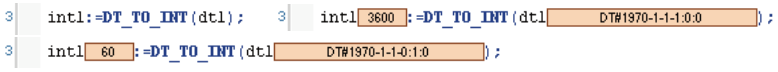
7.7.6日期/日期时间类型转换-9

操作符	DT_TO_BYTE
功能说明	该功能块用作日期时间类型数据向字节类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 日期时间类型(DT); 输出: 字节类型(BYTE);
变量声明	VAR dt1: DT; byte1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	
使用限制	
注意事项	

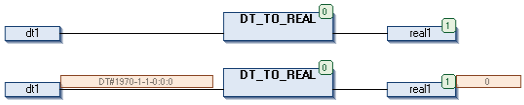
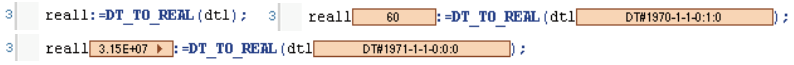
7.7.6日期/日期时间类型转换-10

操作符	DT_TO_WORD
功能说明	该功能块用作日期时间类型数据向字类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 日期时间类型(DT); 输出: 字类型(WORD);
变量声明	VAR dt1: DT; word1: WORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	DT_TO_LWORD、DT_TO_DWORD 用作日期时间类型数据向长字、双字类型数据的转换。
使用限制	
注意事项	

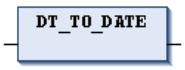
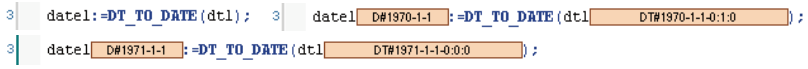
7.7.6日期/日期时间类型转换-11

操作符	DT_TO_INT
功能说明	该功能块用作日期时间类型数据向整数类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 日期时间类型(DT); 输出: 整数类型(INT);
变量声明	VAR dt1: DT; int1: INT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	DT_TO_SINT、DT_TO_USINT 用作日期时间类型数据向短整型、无符号短整数类型数据的转换 DT_TO_DINT、DT_TO_UDINT 用作日期时间类型数据向双整型、无符号双整数类型数据的转换 DT_TO_LINT、DT_TO_ULINT 用作日期时间类型数据向长整型、无符号长整数类型数据的转换
使用限制	
注意事项	

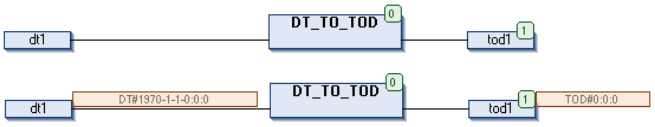
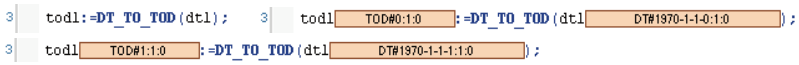
7.7.6日期/日期时间类型转换-12

操作符	DT_TO_REAL
功能说明	该功能块用作日期时间类型数据向实数类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 日期时间类型(DT); 输出: 实数类型(REAL);
变量声明	VAR dt1: DT; real1: REAL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	DT_TO_LREAL 用作日期时间类型数据向长实数类型数据的转换.
使用限制	
注意事项	

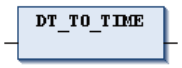
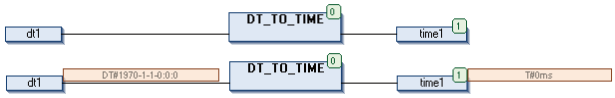
7.7.6日期/日期时间类型转换-13

操作符	DT_TO_DATE
功能说明	该功能块用作日期时间类型数据向日期数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 日期时间类型(DT); 输出: 日期类型(DATE);
变量声明	VAR dt1: DT; date1: DATE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	
使用限制	
注意事项	

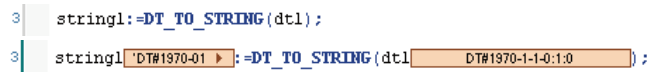
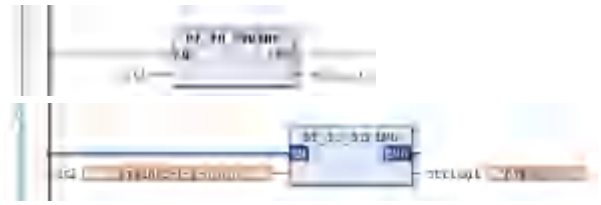
7.7.6日期/日期时间类型转换-14

操作符	DT_TO_TOD
功能说明	该功能块用作日期时间类型数据向时刻类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 日期时间类型(DT); 输出: 时刻类型(TOD);
变量声明	VAR dt1: DT; tod1: TOD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	
使用限制	
注意事项	

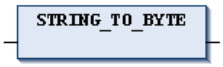
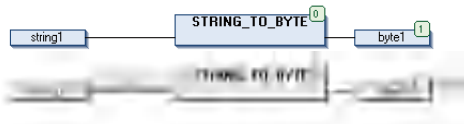
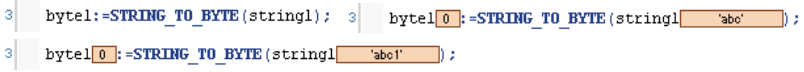
7.7.6日期/日期时间类型转换-15

操作符	DT_TO_TIME
功能说明	该功能块用作日期时间类型数据向时间类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 日期时间类型(DT); 输出: 时间类型(TIME);
变量声明	VAR dt1: DT; time1: TIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	DT_TO_LTIME 用作日期时间类型数据向长时间类型数据的转换。
使用限制	
注意事项	

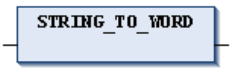
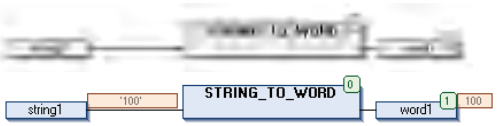
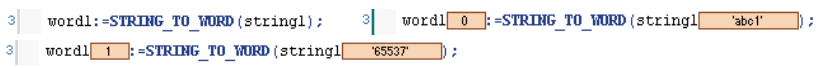
7.7.6日期/日期时间类型转换-16

操作符	DT _TO_STRING
功能说明	该功能块用作日期时间类型数据向字符串类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 日期时间类型(DT); 输出: 字符串类型(STRING);
变量声明	VAR dt1: DT; string1: STRING; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	DT_TO_WSTRING 用作日期时间类型数据向短字符串类型数据的转换。
使用限制	
注意事项	

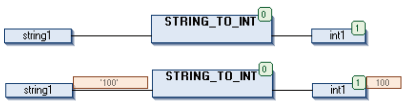
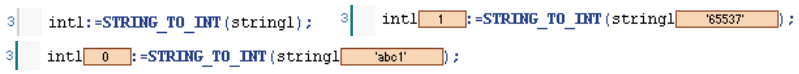
7.7.7字符串类型转换命令-1

操作符	STRING_TO_BYTE
功能说明	该功能块用作字符串类型数据向字节数据类型的转换
图形	
管脚定义	输入: 操作数的数据类型为: 字符串类型(STRING); 输出: 字节类型(BYTE);
变量声明	VAR string1: STRING; byte1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	
使用限制	
注意事项	转换的过程依据C语言标准的编译机制进行: 先把STRING转换为INT类型变量, 然后把INT转换为BYTE类型。由于高字节将被截去, 因此结果将介于0-255之间。由于一条单独的机器指令就可以完成这一转换, 因此在绝大多数处理中都可以由这种方法生成最优的代码。

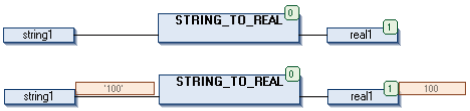
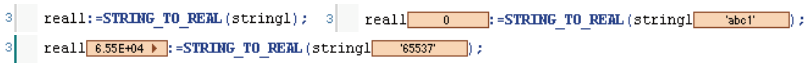
7.7.7字符串类型转换命令-2

操作符	STRING_TO_WORD
功能说明	该功能块用作字符串类型数据向字类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 字符串类型(STRING); 输出: 字类型(WORD);
变量声明	VAR string1: STRING; word1: WORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	STRING_TO_DWORD 用作字符串类型数据向双字类型数据的转换 STRING_TO_LWORD 用作字符串类型数据向长字类型数据的转换
使用限制	
注意事项	转换的过程依据C语言标准的编译机制进行: 先把STRING转换为INT类型变量，然后把INT转换为WORD类型。结果将介于0-65535之间。由于一条单独的机器指令就可以完成这一转换，因此在绝大多数处理中都可以由这种方法生成最优的代码。

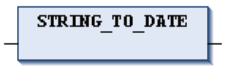
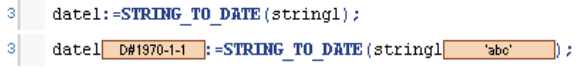
7.7.7字符串类型转换命令-3

操作符	STRING_TO_INT
功能说明	该功能块用作字符串类型数据向整型类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 字符串类型(STRING); 输出: 整数类型(INT);
变量声明	VAR string1: STRING; int1: INT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	STRING_TO_UINT 用作字符串类型数据向无符号整型类型数据的转换 STRING_TO_DINT、STRING_TO_UDINT 用作字符串类型数据向双整型、无符号双整型类型数据的转换 STRING_TO_LINT、STRING_TO_ULINT 用作字符串类型数据向长整型、无符号长整型类型数据的转换 STRING_TO_SINT、STRING_TO_USINT 用作字符串类型数据向短整型、无符号短整型类型数据的转换
使用限制	
注意事项	

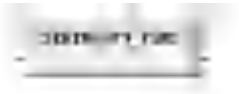
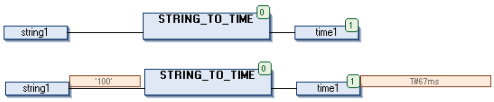
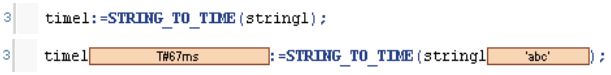
7.7.7字符串类型转换命令-4

操作符	STRING_TO_REAL
功能说明	该功能块用作字符串类型数据向实数型类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 字符串类型(STRING); 输出: 实数类型(REAL);
变量声明	VAR string1: STRING; real1: REAL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	STRING_TO_LREAL 用作字符串类型数据向长实数型类型数据的转换。
使用限制	
注意事项	转换的过程依据C语言标准的编译机制进行: 先把STRING转换为INT类型变量, 然后把INT转换为REAL类型。 由于一条单独的机器指令就可以完成这一转换, 因此在绝大多数处理中都可以由这种方法生成最优的代码。

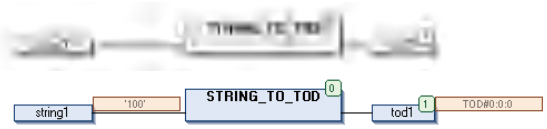
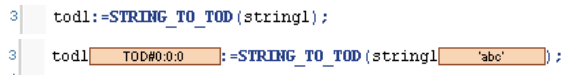
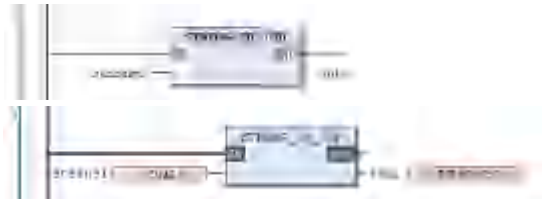
7.7.7字符串类型转换命令-5

操作符	STRING_TO_DATE
功能说明	该功能块用作字符串类型数据向日期类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 字符串类型(STRING); 输出: 日期类型(DATE);
变量声明	VAR string1: STRING; date1: DATE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	
使用限制	
注意事项	转换的过程依据C语言标准的编译机制进行: 先把STRING转换为INT类型变量, 然后把INT转换为DATE类型。 由于一条单独的机器指令就可以完成这一转换, 因此在绝大多数处理中都可以由这种方法生成最优的代码。

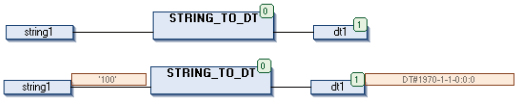
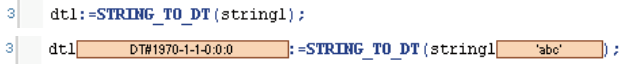
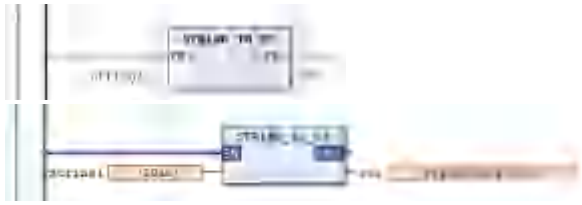
7.7.7字符串类型转换命令-6

操作符	STRING_TO_TIME
功能说明	该功能块用作字符串类型数据向时间类型数据的转换
图形	
管脚定义	输入: 操作数的数据类型为: 字符串类型(STRING); 输出: 时间类型(TIME);
变量声明	VAR string1: STRING; time1: TIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	STRING_TO_LTIME 用作字符串类型数据向长时间类型数据的转换
使用限制	
注意事项	转换的过程依据C语言标准的编译机制进行: 先把STRING转换为INT类型变量, 然后把INT转换为TIME类型。 由于一条单独的机器指令就可以完成这一转换, 因此在绝大多数处理中都可以由这种方法生成最优的代码。

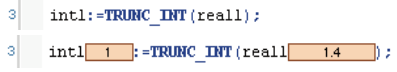
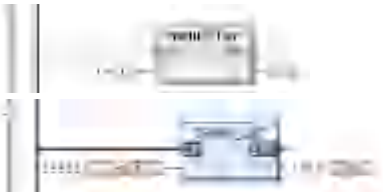
7.7.7字符串类型转换命令-7

操作符	STRING_TO_TOD
功能说明	该功能块用作字符串类型数据向时刻类型数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：字符串类型(STRING); 输出： 时刻类型(TOD);
变量声明	VAR string1: STRING; tod1: TOD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	STRING_TO_LTIME 用作字符串类型数据向长时间类型数据的转换
使用限制	
注意事项	转换的过程依据C语言标准的编译机制进行：先把STRING转换为INT类型变量，然后把INT转换为TOD类型。 由于一条单独的机器指令就可以完成这一转换，因此在绝大多数处理中都可以由这种方法生成最优的代码。

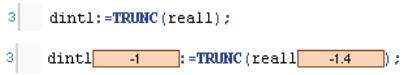
7.7.7字符串类型转换命令-8

操作符	STRING_TO_DT
功能说明	该功能块用作字符串类型数据向日期时间类型数据的转换
图形	
管脚定义	输入： 操作数的数据类型为：字符串类型(STRING); 输出： 日期时间类型(DT);
变量声明	VAR string1: STRING; dt1: DT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	STRING_TO_LTIME 用作字符串类型数据向长时间类型数据的转换
使用限制	
注意事项	转换的过程依据C语言标准的编译机制进行：先把STRING转换为INT类型变量，然后把INT转换为DT类型。 由于一条单独的机器指令就可以完成这一转换，因此在绝大多数处理中都可以由这种方法生成最优的代码。

7.7.8取整

操作符	TRUNC_INT
功能说明	该功能块用作将REAL类型转化为INT类型，将使用该值的整数部分.
图形	
管脚定义	输入: 操作数的数据类型为：实数类型(REAL); 输出: 整数类型(INT);
变量声明	VAR real1: REAL; int1: INT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	
使用限制	
注意事项	注意：TRUNC_INT对应于CoDeSys V2.3版本中的TRUNC，所以当导入工程是2.3版本时会自动替换。须注意v3版本中转换函数TRUNC的变化。

7.7.9截尾取整

操作符	TRUNC
功能说明	该功能块用作将REAL类型转化为DINT类型，将使用该值的整数部分.
图形	
管脚定义	操作数的数据类型为：实数类型(REAL); 输出： 双整数类型(DINT);
变量声明	VAR real1: REAL; dint1: DINT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
备注	
使用限制	
注意事项	

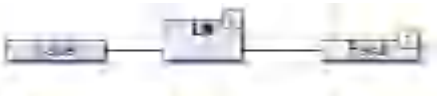
7.8.1绝对值

操作符	ABS		
功能说明	该函数计算输入值的绝对值，并将结果分配给输出。标准IEC操作符。		
数学公式	Result = | Value |;		
图形			
参数描述	输入参数描述		
	变量	数据类型	含义
	Value	基本数据类型	输入值
	输出参数描述		
	变量	数据类型	含义
	Result	基本数据类型	输出值
变量声明	VAR Value: INT; Result: INT; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=ABS(Value);		
使用限制			
注意事项			

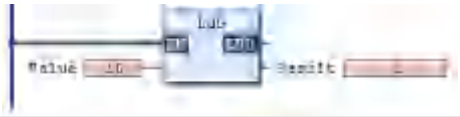
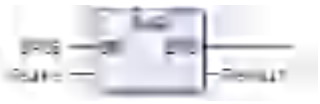
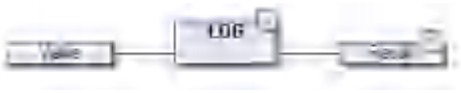
7.8.2平方根

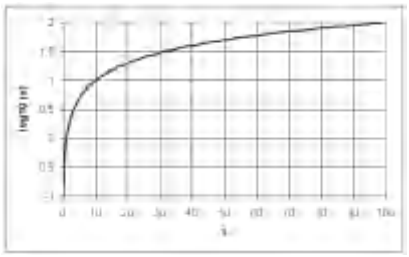
操作符	SQRT		
功能说明	该函数计算输入值的平方根值，并将结果分配给输出。标准IEC操作符。		
数学公式	$Result = \sqrt{Value}$ ；		
图形			
参数描述	输入参数描述		
	变量	数据类型	含义
	Value	基本数据类型	输入值
	输出参数描述		
	变量	数据类型	含义
	Result	REAL，LREAL	输出值
变量声明	VAR Value: INT; Result: REAL; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=SQRT(Value);		
使用限制	输入值不能小于0		
注意事项			

7.8.3自然对数

操作符	LN		
功能说明	该函数计算输入值的自然对数（以常数e为底数）。标准IEC操作符。		
数学公式	$Result = Ln(Value)$ ；		
图形			
参数描述	输入参数描述		
	变量	数据类型	含义
	Value	基本数据类型	输入值
	输出参数描述		
	变量	数据类型	含义
	Result	REAL，LREAL	输出值
变量声明	VAR Value: INT; Result: REAL; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=LN(Value);		
使用限制			
注意事项			

7.8.4常用对数

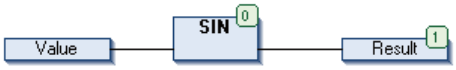
操作符	LOG		
功能说明	该函数计算输入值的常用对数（以10为底数）。标准IEC操作符。		
数学公式	$Result = Log(Value)$ 或 $Result = Lg(Value)$ ；		
图形			
参数描述	输入参数描述		
	变量	数据类型	含义
	Value	基本数据类型	输入值
	输出参数描述		
	变量	数据类型	含义
	Result	REAL，LREAL	输出值
变量声明	VAR Value: INT; Result: REAL; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=LOG(Value);		
使用限制			
注意事项			

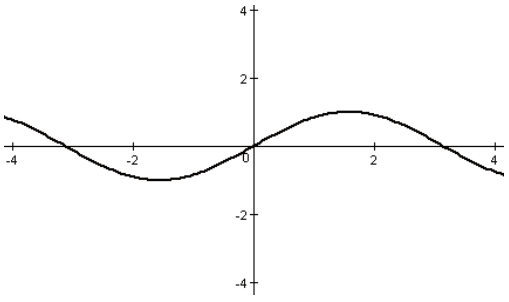


7.8.5指数

操作符	EXP		
功能说明	该函数计算输入值的自然指数。标准IEC操作符。其反函数为自然对数Ln(x)。		
数学公式	$Result = Exp(Value)$ ；		
图形			
参数描述	输入参数描述		
	变量	数据类型	含义
	Value	基本数据类型	输入值
	输出参数描述		
	变量	数据类型	含义
	Result	REAL，LREAL	输出值
变量声明	VAR Value: INT; Result: REAL; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=EXP(Value);		
使用限制			
注意事项			

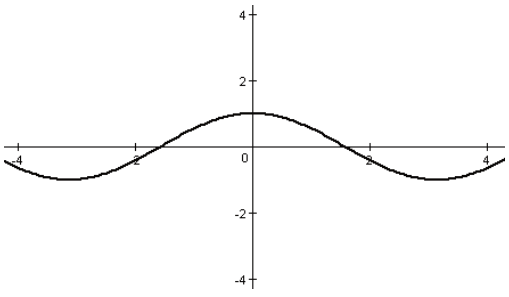
7.8.6正弦

操作符	SIN		
功能说明	该函数计算输入值的正弦值；用弧度表示。标准IEC操作符。		
数学公式	$Result = Sin(Value)$ ；		
图形			
参数描述	输入参数描述		
	变量	数据类型	含义
	Value	基本数据类型	输入值
	输出参数描述		
	变量	数据类型	含义
	Result	REAL，LREAL	输出值
变量声明	VAR Value: INT; Result: REAL; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=SIN(Value);		
使用限制			
注意事项			



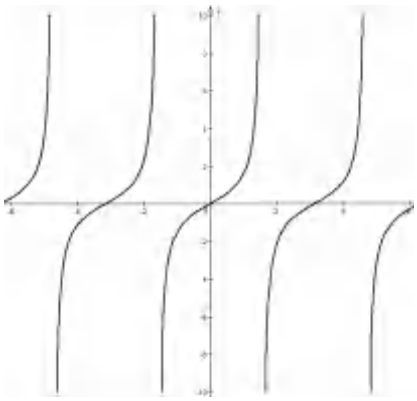
7.8.7余弦

操作符	COS		
功能说明	该函数计算输入值的余弦值；用弧度表示。标准IEC操作符。		
数学公式	$Result = Cos(Value)$;		
图形			
参数描述	输入参数描述		
	变量	数据类型	含义
	Value	基本数据类型	输入值
	输出参数描述		
	变量	数据类型	含义
	Result	REAL，LREAL	输出值
变量声明	VAR Value: INT; Result: REAL; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=COS(Value);		
使用限制			
注意事项			



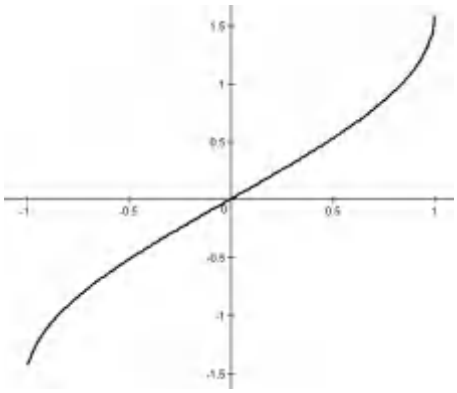
7.8.8正切

操作符	TAN		
功能说明	该函数计算输入值的正切值；用弧度表示。标准IEC操作符。		
数学公式	$Result = Tan(Value)$；		
图形			
参数描述	输入参数描述		
	变量	数据类型	含义
	Value	基本数据类型	输入值
	输出参数描述		
	变量	数据类型	含义
	Result	REAL，LREAL	输出值
变量声明	VAR Value: INT; Result: REAL; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=TAN(Value);		
使用限制			
注意事项			



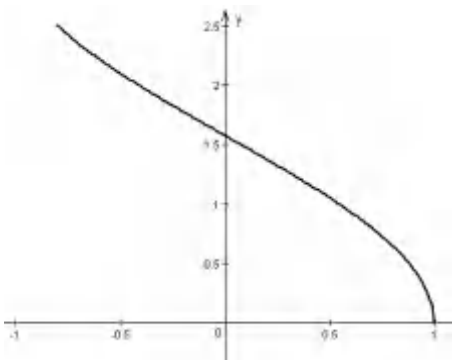
7.8.9反正弦

操作符	ASIN		
功能说明	该函数计算输入值的反正弦值，正弦函数的反函数；用弧度表示。标准IEC操作符。		
数学公式	$Result = Asin(Value)$;		
图形			
参数描述	输入参数描述		
	变量	数据类型	含义
	Value	基本数据类型	输入值
	输出参数描述		
	变量	数据类型	含义
	Result	REAL，LREAL	输出值
变量声明	VAR Value: INT; Result: REAL; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=ASIN(Value);		
使用限制	-1 ≤ Value ≤ 1		
注意事项			



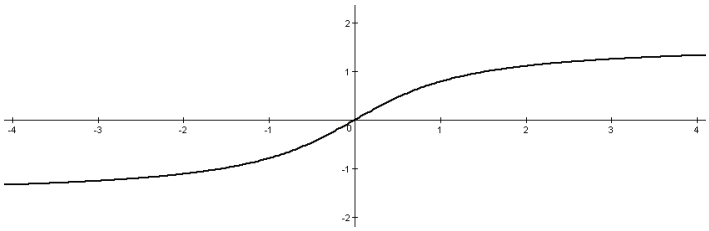
7.8.10反余弦

操作符	ACOS		
功能说明	该函数计算输入值的反余弦值，余弦函数的反函数；用弧度表示。标准IEC操作符。		
数学公式	$Result = Acos(Value)$；		
图形			
参数描述	输入参数描述		
	变量	数据类型	含义
	Value	基本数据类型	输入值
	输出参数描述		
	变量	数据类型	含义
	Result	REAL，LREAL	输出值
变量声明	VAR Value: INT; Result: REAL; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=ACOS(Value);		
使用限制	-1 ≤ Value ≤ 1		
注意事项			

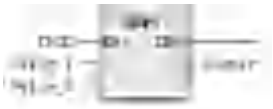


7.8.11反正切

操作符	ATAN		
功能说明	该函数计算输入值的反正切值，正切函数的反函数；用弧度表示。标准IEC操作符。		
数学公式	$Result = A\tan(Value)$；		
图形			
参数描述	输入参数描述		
	变量	数据类型	含义
	Value	基本数据类型	输入值
	输出参数描述		
	变量	数据类型	含义
	Result	REAL，LREAL	输出值
变量声明	VAR Value: INT; Result: REAL; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=ATAN(Value);		
使用限制			
注意事项			

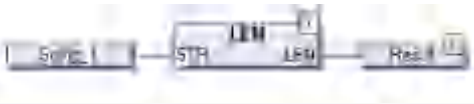


7.8.12幂

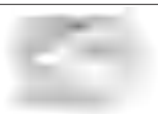
操作符	EXPT		
功能说明	该函数为输入值1对输入值2的求幂计算。标准IEC操作符。		
数学公式	$Result = (Value\_1)^{Value\_2}$;		
图形			
参数描述	输入参数描述		
	变量	数据类型	含义
	Value_1	基本数据类型	输入值
	Value_2	基本数据类型	输入值
	输出参数描述		
	变量	数据类型	含义
	Result	REAL，LREAL	输出值
变量声明	VAR Value_1: INT; Value_2: INT; Result: REAL; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=EXPT(Value_1,Value_2);		
使用限制			
注意事项			

第八章 SoMachine通用库指南

8.1.1字符串函数-字符串长度

操作符	LEN		
功能说明	返回输入字符串的长度。位于Standard. Library中。		
图形			
管脚定义	输入管脚描述		
	标识符	数据类型	含义
	STR	String	输入的字符串
	输出管脚描述		
	标识符	数据类型	含义
		INT	输出字符串长度
变量声明	VAR String_1: STRING; Result: INT; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=LEN(String_1);		
使用限制			
注意事项			

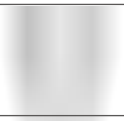
8.1.1字符串函数-左边取字符串

操作符	LEFT		
功能说明	从字符串最左侧开始向右提取特定数目的字符；组成一个新的字符串输出。		
图形			
管脚定义	输入管脚描述		
	标识符	数据类型	含义
	STR	String	输入的字符串
	SIZE	INT	提取的位数
	输出管脚描述		
	标识符	数据类型	含义
		String	输出字符串
变量声明	VAR String_1: STRING; Size：INT; Result: String; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=LEFT(String_1, Size);		
使用限制			
注意事项			

8.1.1字符串函数-右边取字符串

操作符	RIGHT		
功能说明	从字符串最右侧开始向左提取特定数目的字符；组成一个新的字符串输出。位于Standard. Library中。		
图形			
管脚定义	输入管脚描述		
	标识符	数据类型	含义
	STR	String	输入的字符串
	SIZE	INT	提取的位数
	输出管脚描述		
	标识符	数据类型	含义
		String	输出字符串
变量声明	VAR String_1: STRING; Size：INT； Result: String; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=RIGHT(String_1, Size);		
使用限制			
注意事项			

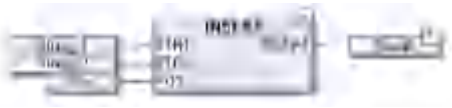
8.1.1字符串函数-中间取字符串

操作符	MID		
功能说明	从字符串中间提取特定数目的字符；组成一个新的字符串输出。位于Standard. Library中。		
图形			
管脚定义	输入管脚描述		
	标识符	数据类型	含义
	STR	String	输入的字符串
	LEN	INT	提取的位数
	POS	INT	开始提取的位数，从左计数
	输出管脚描述		
	标识符	数据类型	含义
		String	输出字符串
变量声明	VAR String_1: STRING; Legth：INT； Pos：INT； Result: String; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=MID(String_1, Legth, Pos);		
使用限制			
注意事项			

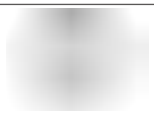
8.1.1字符串函数-合并字符串

操作符	CONCAT		
功能说明	将两个字符串合并组成一个新的字符串输出。位于Standard. Library中。		
图形			
管脚定义	输入管脚描述		
	标识符	数据类型	含义
	STR1	String	合并字符串的头部
	STR2	String	合并字符串的尾部
	输出管脚描述		
	标识符	数据类型	含义
		String	输出合并后的字符串
变量声明	VAR String_1: STRING; String_2: STRING; Result: String; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=CONCAT(String_1, String_2);		
使用限制			
注意事项			

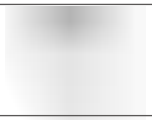
8.1.1字符串函数-插入字符串

操作符	INSERT		
功能说明	在一个字符串中插入另一个字符串，组成新的字符串输出。位于Standard. Library中。		
图形			
管脚定义	输入管脚描述		
	标识符	数据类型	含义
	STR1	String	初始字符串
	STR2	String	准备插入STR1的字符串
	POS	INT	STR1从左计数，在此位置后插入STR2
	输出管脚描述		
	标识符	数据类型	含义
		String	输出合并后的字符串
变量声明	VAR String_1: STRING; String_2: STRING; Pos: INT; Result: String; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=INSERT(String_1, String_2, Pos);		
使用限制			
注意事项			

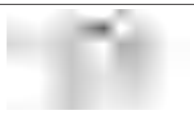
8.1.1字符串函数-删除字符串

操作符	DELETE		
功能说明	在一个字符串中删除特定的连续的部分，剩余的字符组成字符串输出。位于Standard. Library中。		
图形			
管脚定义	输入管脚描述		
	标识符	数据类型	含义
	STR	String	初始字符串
	LEN	INT	删除字符的长度
	POS	INT	从左计数开始删除的位置(包含此位置)
	输出管脚描述		
	标识符	数据类型	含义
		String	输出合并后的字符串
变量声明	VAR String_1: STRING; Legth: INT; Pos: INT; Result: String; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=DELETE(String_1, Legth, Pos);		
使用限制			
注意事项			

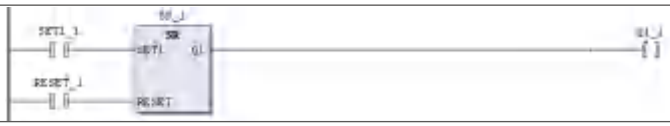
8.1.1字符串函数-替换字符串

操作符	REPLACE		
功能说明	用一个字符串替换掉长度更长的字符串中的一部分，组成新的字符串输出。位于Standard. Library中。		
图形			
管脚定义	输入管脚描述		
	标识符	数据类型	含义
	STR 1	String	初始字符串
	STR 2	String	用于替换的字符串
	L	INT	替换掉的字符的长度
	P	INT	从左计数开始替换的位置(包含此位置)
	输出管脚描述		
	标识符	数据类型	含义
		String	输出合并后的字符串
变量声明	VAR String_1: STRING; String_2 : STRING; Legth: INT; Pos: INT; Result: String; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Result:=REPLACE(String_1, String_2, Legth, Pos);		
使用限制			
注意事项			

8.1.1字符串函数-查找字符串

操作符	FIND		
功能说明	从一个字符串找出特定字符串的初始位置。位于Standard. Library中。		
图形			
管脚定义	输入管脚描述		
	标识符	数据类型	含义
	STR 1	String	初始字符串
	STR 2	String	被查找的字符串
	输出管脚描述		
	标识符	数据类型	含义
		INT	STR2在STR1中的开始位置，如果在STR1中没有找到STR2则输出“0”。
变量声明	VAR String_1: STRING; String_ 2 : STRING; Pos: INT; END_VAR		
在LD中表示形式			
在FBD中表示形式			
在CFC中表示形式			
在ST中表示形式	Pos:=FIND(String_1, String_2);		
使用限制			
注意事项	注意字符串字母的大小写		

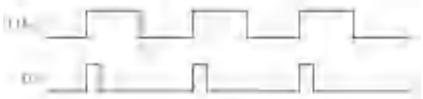
8.1.2双稳态功能块-置位优先触发器

操作符	SR
功能说明	该功能块用作一个置位优先触发器
图形	
管脚定义	输入： SET1：布尔型（BOOL）；该输入端为为置位输入，TRUE时置位输入，FALSE时置位输入消失 RESET：布尔型（BOOL）；该输入端为为复位输入，TRUE时复位输入，FALSE时复位输入消失 输出： Q1：布尔型（BOOL）；触发器状态输出，一旦出现SET1输入，则Q1输出TRUE 当SET1为TRUE时，不论RESET是否为TRUE，Q1输出都为TRUE； 当SET1为FALSE时，如果Q1输出为TRUE,一旦RESET为TRUE，Q1输出立刻复位为FALSE。如果Q1输出为FALSE,不论RESET为TRUE或者FALSE，Q1输出保持为FALSE。 功能块逻辑表达式：Q1 = (NOT RESET AND Q1) OR SET1
变量声明	VAR SR_1: SR; SET1_1:BOOL; RESET_1: BOOL; Q1_1: BOOL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

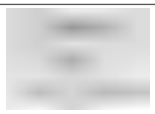
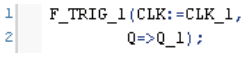
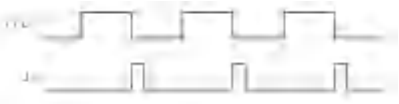
8.1.2双稳态功能块-复位优先触发器

操作符	RS
功能说明	该功能块用作一个复位优先触发器
图形	
管脚定义	输入： SET：布尔型（BOOL）；该输入端为为置位输入，TRUE时置位输入，FALSE时置位输入消失 RESET1：布尔型（BOOL）；该输入端为为复位输入，TRUE时复位输入，FALSE时复位输入消失 输出： Q1：布尔型（BOOL）；触发器状态输出，一旦出现RESET1输入，则Q1输出FALSE 当RESET1为TRUE时，不论SET是否为TRUE，Q1输出都为FALSE； 当RESET1为FALSE时，如果Q1输出为FALSE,一旦SET为TRUE，Q1输出立刻置位为TRUE。如果Q1输出为TRUE,不论SET为TRUE或者FALSE，Q1输出保持为TRUE。 功能块逻辑表达式：Q1 = NOT RESET1 AND (Q1 OR SET)
变量声明	VAR RS_1: RS; SET_1:BOOL; RESET1_1: BOOL; Q1_1: BOOL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

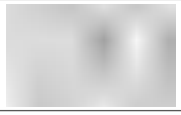
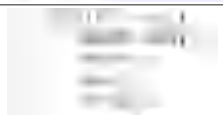
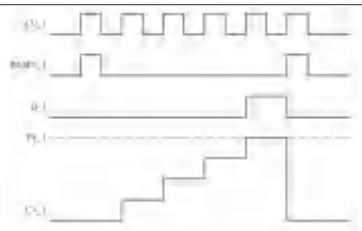
8.1.3触发器功能块-上升沿检测触发器

操作符	R_TRIG
功能说明	该功能块用作一个上升沿检测
图形	
管脚定义	输入： CLK：布尔型（BOOL）；该输入端为布尔型被检测上升沿的信号输入 输出： Q：布尔型（BOOL）；触发器状态输出，一旦检测到CLK有上升沿信号输入，则Q输出TRUE 当CLK从FALSE变为TRUE时，Q输出先变为TRUE然后Q输出变为FALSE； 如果CLK持续保持为FALSE或者TRUE，Q输出一直保持为FALSE;
变量声明	VAR R_TRIG_1: R_TRIG; CLK_1: BOOL; Q_1: BOOL; END_VAR
CFC语言示例	
ST语言示例	1 R_TRIG_1(CLK:=CLK_1, 2 Q=>Q_1);
LD语言示例	
时序图	
使用限制	
注意事项	

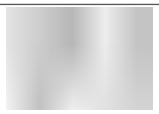
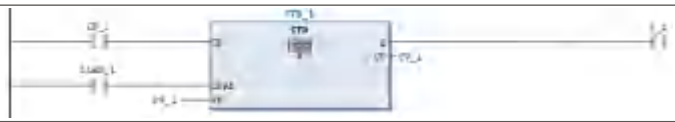
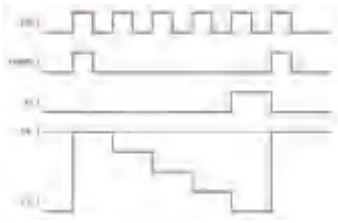
8.1.3触发器功能块-下降沿检测触发器

操作符	F_TRIG
功能说明	该功能块用作一个下降沿检测
图形	
管脚定义	输入： CLK：布尔型（BOOL）；该输入端为布尔型被检测下降沿的信号输入 输出： Q：布尔型（BOOL）；触发器状态输出，一旦检测到CLK有下降沿信号输入，则Q输出TRUE 当CLK从TRUE变为FALSE时，Q输出先变为TRUE然后Q输出变为FALSE； 如果CLK持续保持为FALSE或者TRUE，Q输出一直保持为FALSE;
变量声明	VAR F_TRIG_1: F_TRIG; CLK_1: BOOL; Q_1: BOOL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

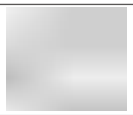
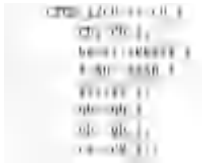
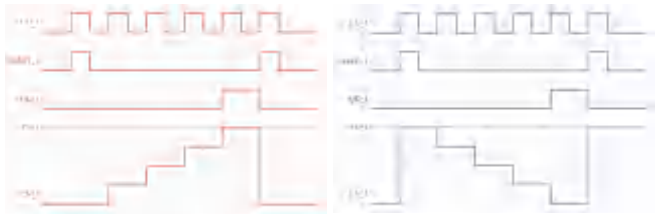
8.1.4计数器-递增计数器

操作符	CTU
功能说明	该功能块用作递增计数器
图形	
管脚定义	输入： CU：布尔型（BOOL）；该输入端为布尔型，检测上升沿的信号输入触发输出CV递增 RESET:布尔型（BOOL）；该输入端为布尔型，用来复位计数器，如果为TRUE时CV复位为0 PV:字（WORD）；该输入为一个字，用来设置输出CV递增计数上限 输出： Q：布尔型（BOOL）；计数器状态输出，输出CV递增到计数上限PV时，则Q输出TRUE CV：字（WORD）；该输出为递增计数实时值，按照CU上升沿依次显示从0到PV递增的数值 当RESET为TRUE时，不论CU是否检测到上升沿，都不能触发递增计数，CV保持0； 当RESET为FALSE时，当CU端有一个从FALSE变为TRUE的上升沿时，CV将加1，当CV递增到上限PV时，Q输出为TRUE；
变量声明	VAR CTU_1: CTU; CU_1: BOOL; RESET_1: BOOL; PV_1: WORD; Q_1: BOOL; CV_1: WORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

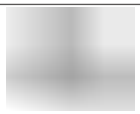
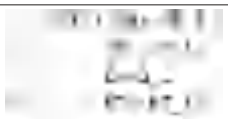
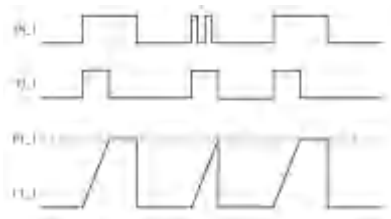
8.1.4计数器-递减计数器

操作符	CTD
功能说明	该功能块用作递减计数器
图形	
管脚定义	输入： CD：布尔型（BOOL）；该输入端为布尔型，检测上升沿的信号输入触发输出CV递减 LOAD:布尔型（BOOL）；该输入端为布尔型，用来复位计数器，如果为TRUE时CV复位为上限值PV PV:字（WORD）；该输入为一个字，用来设置输出CV递减计数上限 输出： Q：布尔型（BOOL）；计数器状态输出，输出CV递减到0时，则Q输出TRUE CV：字（WORD）；该输出为递减计数实时值，按照CD上升沿依次显示从PV到0递增的数值 当LOAD为TRUE时，不论CD是否检测到上升沿，都不能触发递减计数，CV保持递减计数上限PV值； 当LOAD为FALSE时，当CD端有一个从FALSE变为TRUE的上升沿时，若CV大于0时，CV将减1,CV等于0时，Q输出为TRUE；
变量声明	VAR CTD_1: CTD; CD_1: BOOL; LOAD_1: BOOL; PV_1: WORD; Q_1: BOOL; CV_1: WORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

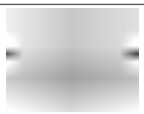
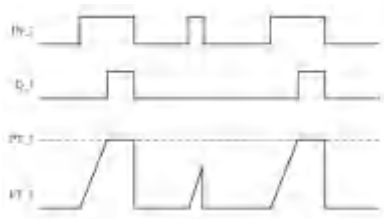
8.1.4计数器-递增递减计数器

操作符	CTUD
功能说明	该功能块用作一个递增递减计数器
图形	
管脚定义	输入： CU：布尔型（BOOL）；该输入端为布尔型，检测上升沿的信号输入触发输出CV递增 CD：布尔型（BOOL）；该输入端为布尔型，检测上升沿的信号输入触发输出CV递减 RESET:布尔型（BOOL）；该输入端为布尔型，用来复位递增计数器，如果为TRUE时CV复位为0 LOAD:布尔型（BOOL）；该输入端为布尔型，用来复位递减计数器，如果为TRUE时CV复位为上限值PV PV:字（WORD）；该输入为一个字，用来设置输出CV递减或递减的计数上限 输出： QU：布尔型（BOOL）；计数器状态输出，输出CV递增到计数上限PV时，则QU输出TRUE QD：布尔型（BOOL）；计数器状态输出，输出CV递减到0时，则QD输出TRUE CV：字（WORD）；该输出为递增或递减的计数实时值 当RESET为TRUE时，不论CU或CD是否检测到上升沿，都不能触发计数器，CV保持0； 当LOAD为TRUE时，不论CU或CD是否检测到上升沿，都不能触发计数器，CV保持递减计数上限PV值；
变量声明	VAR CTUD_1: CTUD; CU_1: BOOL; CD_1: BOOL; RESET_1: BOOL; LOAD_1: BOOL; PV_1: WORD; QU_1: BOOL; QD_1: BOOL; CV_1: WORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

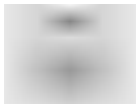
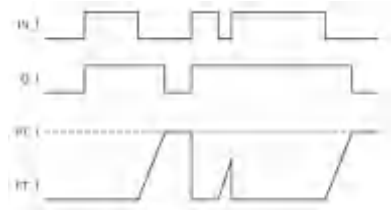
8.1.5定时器-TP定时器

操作符	TP
功能说明	该功能块用作定时器
图形	
管脚定义	输入： IN：布尔型（BOOL）；该输入端为布尔型，检测上升沿的信号输入触发定时器 PT：时间常量（TIME）；该输入为一个时间常量，用来设置输出ET计时上限 输出： Q：布尔型（BOOL）；定时器状态输出，输出ET到计数上限PT时，则Q输出TRUE ET：时间常量（TIME）；该输出为计时实时值，从IN上升沿开始计时的时间值 当检测到IN上升沿后，不论IN是否保持为TRUE输出ET以毫秒精度开始计时，在ET计时到达设定PT计时上限之前Q都输出TRUE，只要ET计时到达设定计时上限PT时，Q输出为FALSE； 如果IN没有检测到上升沿时，Q输出为FALSE；
变量声明	VAR TP_1:TP; IN_1:BOOL; PT_1: TIME; Q_1: BOOL; ET_1: TIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.1.5定时器-通电延时定时器

操作符	TON
功能说明	该功能块用作通电延时定时器
图形	
管脚定义	输入： IN：布尔型（BOOL）；该输入端为布尔型，检测上升沿的信号输入触发通电延时定时器 PT：时间常量（TIME）；该输入为一个时间常量，用来设置通电延时间 输出： Q：布尔型（BOOL）；定时器状态输出，输出ET到延时时间PT时，则Q输出TRUE ET：时间常量（TIME）；该输出为延时实时值，从IN上升沿开始计时的时间值 当检测到IN上升沿后输出ET开始计时，只有输入IN持续为TRUE计时到达设定时间PT后，定时器状态输出Q为TRUE； 如果在计时到达设定时间PT之前输入IN由TRUE变为FALSE,则定时器状态输出Q还是为FALSE；
变量声明	VAR TON_1: TON; IN_1: BOOL; PT_1: TIME; Q_1: BOOL; ET_1: TIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.1.5定时器-断电延时定时器

操作符	TOF
功能说明	该功能块用作一个断电延时定时器
图形	
管脚定义	输入： IN：布尔型（BOOL）；该输入端为布尔型，检测下降沿的信号输入触发断电延时定时器 PT：时间常量（TIME）；该输入为一个时间常量，用来设置断电延时间 输出： Q：布尔型（BOOL）；定时器状态输出，输出ET到延时时间PT时，则Q输出FALSE ET：时间常量（TIME）；该输出为延时实时值，从IN下降沿开始计时的时间值 当IN为TRUE时,Q为TRUE，ET为0； 当检测到IN下降沿后输出ET开始计时，只有输入IN持续为FALSE计时到达设定时间PT后，定时器状态输出Q为TRUE； 如果在计时到达设定时间PT之前输入IN由FALSE变为TRUE,则定时器状态输出Q还是为TRUE；
变量声明	VAR TOF_1: TOF; IN_1: BOOL; PT_1: TIME; Q_1: BOOL; ET_1: TIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

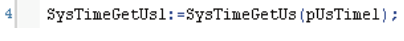
8.1.5定时器-实时时钟

操作符	RTC
功能说明	该功能块用作一个实时时钟
图形	
管脚定义	输入： EN：布尔型（BOOL）；该输入端为布尔型，检测上升沿的信号输入设置PDT值为实时时钟 PDT：日期时间常量（DATE_AND_TIME）；该输入为一个日期时间常量，用来设置日期和时间 输出： Q：布尔型（BOOL）；实时时钟状态输出，EN输入为TRUE后CDT开始计时，则Q输出TRUE CDT：日期时间常量（DATE_AND_TIME）；该输出为实时时钟显示 当EN为TRUE时，CDT输出将被设置为PDT的日期和时间，并且以秒开始计时，Q输出为TRUE； 当EN为FALSE时，CDT输出将被复位为初始日期和时间：DT#1970-01-01-00:00:00，Q输出为FALSE；
变量声明	VAR RTC_1: RTC; EN_1: BOOL; PDT_1: DATE_AND_TIME; Q_1: BOOL; CDT_1: DATE_AND_TIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

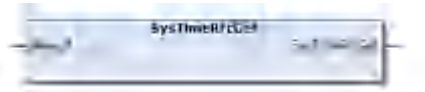
8.2.1计时功能-SysTimeGetMs

操作符	SysTimeGetMs
功能说明	此功能用作返回电源接通后所经过的时间，以毫秒 (ms) 为单位。
图形	
管脚定义	输出： SysTimeGetMs：无符号双整型（UDINT）；电源接通后所经过的时间（以毫秒为单位）
变量声明	VAR SysTimeGetMs1: UDINT;; END_VAR
CFC语言示例	
ST语言示例	4 SysTimeGetMs1:=SysTimeGetMs();
LD语言示例	
时序图	
使用限制	
注意事项	要访问 SysTime 功能，需手动添加该库。操作步骤：1.在"设备"窗口中，双击控制器应用程序节点的库管理器节点；2.单击添加库；3.浏览至系统 → SysLibs（如果公司过滤器的设置为系统或全部公司，则可见）；4. 选择 SysTime 库并按确定。

8.2.1计时功能-SysTimeGetUs

操作符	SysTimeGetUs
功能说明	此功能用作返回电源接通后所经过的时间，以微秒 (µs) 为单位。
图形	
管脚定义	输入/输出： pUsTime：无符号长整型（ULINT）；电源接通后所经过的时间（以微秒为单位） 输出： SysTimeGetUs：无符号双整型（UDINT）；功能运行诊断：0 = 未检测到错误；2 = 在输入变量上检测到错误，返回的值 (pUsTime) 无效。
变量声明	VAR pUsTime1: ULINT; SysTimeGetUs1: UDINT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	要访问 SysTime 功能，需手动添加该库。操作步骤：1.在"设备"窗口中，双击控制器应用程序节点的库管理器节点；2.单击添加库；3.浏览至系统 → SysLibs（如果公司过滤器的设置为系统或全部公司，则可见）；4. 选择 SysTime 库并按确定。

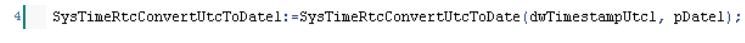
8.2.2标准Real Time Clock功能-SysTimeRtcGet

操作符	SysTimeRtcGet
功能说明	此功能用作返回 Universal Time Coordinated (UTC) Timestamp 值（自 1970 年 1 月 1 日 00:00:00 到现在所经过的秒数）形式的控制器 Real Time Clock (RTC)
图形	
管脚定义	输入/输出： pResult：无符号双整型（UDINT）；功能运行诊断：0 = 未检测到错误；1 = 检测到内部操作错误，返回的值 (SysTimeRtcGet) 无效 (0)。 输出： SysTimeRtcGet：双字型（DWORD）；UTC Timestamp 值（自 1970 年 1 月 1 日 00:00:00 到现在所经过的秒数）形式的控制器 RTC
变量声明	VAR pResult1: UDINT; SysTimeRtcGet1: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	要访问 SysTime 功能，需手动添加该库。操作步骤：1.在"设备"窗口中，双击控制器应用程序节点的库管理器节点；2.单击添加库；3.浏览至系统 → SysLibs（如果公司过滤器的设置为系统或全部公司，则可见）；4. 选择 SysTime 库并按确定。

8.2.2标准Real Time Clock功能-SysTimeRtcSet

操作符	SysTimeRtcSet
功能说明	此功能用作通过提供的 Timestamp 值（自 1970 年 1 月 1 日 00:00:00 到现在所经过的秒数）设置控制器 Real Time Clock。
图形	
管脚定义	输入： ulTimestamp：双字型（ DWORD）； Timestamp 值（自 1970 年 1 月 1 日 00:00:00 到现在经过的秒数） 输出： SysTimeRtcSet：无符号双整型（UDINT）； 功能运行诊断：0 = 未检测到错误。
变量声明	VAR ulTimestamp1:　DWORD; SysTimeRtcSet1：UDINT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	要访问 SysTime 功能，需手动添加该库。操作步骤：1.在"设备"窗口中，双击控制器应用程序节点的库管理器节点；2.单击添加库；3.浏览至系统 → SysLibs（如果公司过滤器的设置为系统或全部公司，则可见）；4. 选择 SysTime 库并按确定。

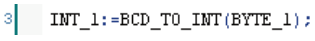
8.2.2标准Real Time Clock功能-SysTimeRtcConvertUtcToDate

操作符	SysTimeRtcConvertUtcToDate
功能说明	此功能用作将 timestamp 值（自 1970 年 1 月 1 日 00:00:00 到现在所经过的秒数）转换为 SYSTIMEDATE 格式的相应日期和时间。
图形	
管脚定义	输入： dwTimestampUtc：双字型（ DWORD）；要转换的 Timestamp（自 1970 年 1 月 1 日 00:00:00 到现在所经过的秒数）。 输出： SysTimeRtcConvertUtcToDate：无符号双整型（UDINT）；功能运行诊断：0 = 未检测到错误；2 = 在输入变量上检测到错误，返回的值 (pDate) 无效。 输入/输出： SYSTIMEDATE.wYear：UINT；年。 SYSTIMEDATE.wMonth：UINT；月。 SYSTIMEDATE.wDay：UINT；日。 SYSTIMEDATE.wHour：UINT；小时。 SYSTIMEDATE.wMinute：UINT；分钟。 SYSTIMEDATE.wSecond：UINT；秒。 SYSTIMEDATE.wMilliseconds：UINT；毫秒。 SYSTIMEDATE.wDayOfWeek：UINT；星期几。 SYSTIMEDATE.wYday：UINT；一年中的第几天。
变量声明	VAR dwTimestampUtc1: DWORD; SysTimeRtcConvertUtcToDate1 : UDINT; pDate1: SYSTIMEDATE; END_VAR
CFC语言示例	
ST语言示例	 4 SysTimeRtcConvertUtcToDate1:=SysTimeRtcConvertUtcToDate(dwTimestampUtc1, pDate1);
LD语言示例	
时序图	
使用限制	
注意事项	要访问 SysTime 功能，需手动添加该库。操作步骤：1.在"设备"窗口中，双击控制器应用程序节点的库管理器节点；2.单击添加库；3.浏览至系统 → SysLibs（如果公司过滤器的设置为系统或全部公司，则可见）；4. 选择 SysTime 库并按确定。

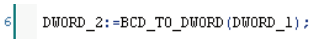
8.2.2标准Real Time Clock功能-SysTimeRtcConvertDateToUtc

操作符	SysTimeRtcConvertDateToUtc
功能说明	此功能用作将 SYSTIMEDATE 格式的日期和时间转换为相应的 Timestamp 值（自 1970 年 1 月 1 日 00:00:00 到现在所经过的秒数）。
图形	
管脚定义	输出: SysTimeRtcConvertDateToUtc：无符号双整型（UDINT）；功能运行诊断：0 = 未检测到错误；2 = 在输入变量上检测到错误，返回的值 (pdwTimestampUtc) 无效。 输入/输出: pDate1：系统日期时间格式（SYSTIMEDATE）；要转换的日期和时间 。显示形式为年、月、日、小时、分钟、秒值。 pdwTimestampUtc1：双字型（DWORD）；从输入的日期和时间值计算。 其中：SYSTIMEDATE.wYear：UINT；年。 SYSTIMEDATE.wMonth：UINT；月。 SYSTIMEDATE.wDay：UINT；日。 SYSTIMEDATE.wHour：UINT；小时。 SYSTIMEDATE.wMinute：UINT；分钟。 SYSTIMEDATE.wSecond：UINT；秒。 SYSTIMEDATE.wMilliseconds：UINT；毫秒。 SYSTIMEDATE.wDayOfWeek：UINT；星期几。 SYSTIMEDATE.wYday：UINT；一年中的第几天。
变量声明	VAR SysTimeRtcConvertDateToUtc1: UDINT; pdwTimestampUtc1: DWORD ; pDate1: SYSTIMEDATE; END_VAR
CFC语言示例	
ST语言示例	 4 SysTimeRtcConvertDateToUtc1:=SysTimeRtcConvertDateToUtc(pDate1, pdwTimestampUtc1);
LD语言示例	
时序图	
使用限制	
注意事项	要访问 SysTime 功能，需手动添加该库。操作步骤：1.在"设备"窗口中，双击控制器应用程序节点的库管理器节点；2.单击添加库；3.浏览至系统 → SysLibs（如果公司过滤器的设置为系统或全部公司，则可见）；4. 选择 SysTime 库并按确定。

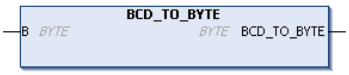
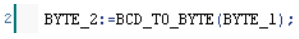
8.3.1 BCD转换-BCD_TO_INT

操作符	BCD_TO_INT
功能说明	该函数将BCD码转换成INT值。由应用库UTIL.Library提供
图形	
管脚定义	输入: B：字节型（BYTE）；该输入端为Byte类型的BCD码 输出: 有符号整数型（INT）；输出值为INT值
变量声明	VAR Byte_1: BYTE; INT_1: INT; END_VAR
CFC语言示例	
ST语言示例	 3 INT_1:=BCD_TO_INT(BYTE_1);
LD语言示例	
时序图	
使用限制	
注意事项	如果输入值不是BCD码，则返回错误值为-1。

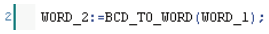
8.3.1 BCD转换-BCD_TO-DOWARD

操作符	BCD_TO_DWORD
功能说明	该函数将BCD码转换成DWORD值。由应用库UTIL.Library提供
图形	
管脚定义	输入： X：双字型（DWORD）；该输入端为双字类型的BCD码 输出： 双字型（DWORD）；输出值为DWORD值
变量声明	VAR DWORD_1: DWORD; DWORD_2: DWORD; END_VAR
CFC语言示例	
ST语言示例	 6 DWORD_2:=BCD_TO_DWORD(DWORD_1);
LD语言示例	
时序图	
使用限制	
注意事项	如果输入值不是BCD码，则返回错误值为-1。

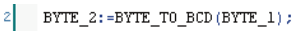
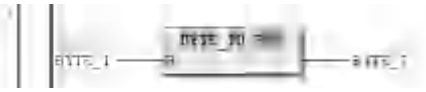
8.3.1 BCD转换-BCD_TO_BYTE

操作符	BCD_TO_BYTE
功能说明	该函数将BCD码转换成字节的二进制值。由应用库UTIL.Library提供
图形	
管脚定义	输入: B：字节型（BYTE）；该输入端为Byte类型的BCD码 输出: BYTE类型（BYTE）；输出值为字节的二进制值
变量声明	VAR BYTE_1: BYTE; BYTE_2: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

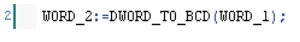
8.3.1 BCD转换-BCD_TO_WORD

操作符	BCD_TO_WORD
功能说明	该函数将BCD码转换成WORD值。由应用库UTIL.Library提供
图形	
管脚定义	输入: W：单字（WORD）；该输入端为单字的BCD码 输出: 单字（WORD）；输出值为WORD值
变量声明	VAR WORD_1: WORD; WORD_2: WORD; END_VAR
CFC语言示例	
ST语言示例	 WORD_2:=BCD_TO_WORD(WORD_1);
LD语言示例	
时序图	
使用限制	
注意事项	

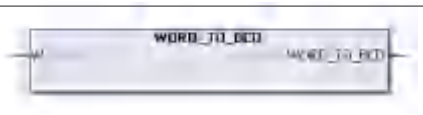
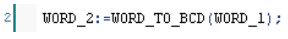
8.3.1 BCD转换-BYTE_TO_BCD

操作符	BYTE_TO_BCD
功能说明	该函数将数值范围为0-99的字节值转换成BCD码格式。由应用库UTIL.Library提供
图形	
管脚定义	输入: B：字节型（BYTE）；该输入端为范围0-99的Byte类型的值 输出: 字节型（BYTE）；输出值为BYTE类型的BCD值
变量声明	VAR BYTE_1: BYTE; BYTE_2: BYTE; END_VAR
CFC语言示例	
ST语言示例	 2 BYTE_2:=BYTE_TO_BCD(BYTE_1);
LD语言示例	
时序图	
使用限制	
注意事项	输入端的字节值范围为：0-99，如果超过99，则循环到0-99范围输出。

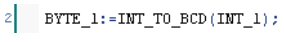
8.3.1 BCD转换-DWORD_TO-BCD

操作符	DWORD_TO_BCD
功能说明	该函数将DWORD转换成BCD值。由应用库UTIL.Library提供
图形	
管脚定义	输入: X：双字型（DWORD）；该输入端为范围为0-9999的DWORD值 输出: 双字型（DWORD）；输出值为DWORD的BCD值
变量声明	VAR DWORD_1: DWORD; DWORD_2: DWORD; END_VAR
CFC语言示例	
ST语言示例	 2 WORD_2:=DWORD_TO_BCD(WORD_1);
LD语言示例	
时序图	
使用限制	
注意事项	输入值的范围为0-9999.

8.3.1 BCD转换-WORD_TO-BCD

操作符	WORD_TO_BCD
功能说明	该函数将WORD值转换成BCD值。由应用库UTIL.Library提供
图形	
管脚定义	输入: W：单字（WORD）；该输入端为单字的WORD值 输出: 单字（WORD）；输出值为BCD码
变量声明	VAR WORD_1: WORD; WORD_2: WORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	输入值的范围为0-9999.

8.3.1 BCD转换-INT_TO_BCD

操作符	INT_TO_BCD
功能说明	该函数将整数值转换成BCD码格式。由应用库UTIL.Library提供
图形	
管脚定义	输入: I：整数型（INT）；该输入端为INT类型的整数值 输出: 字节型（BYTE）；输出值为字节类型的BCD值
变量声明	VAR Byte_1: BYTE; INT_1: INT; END_VAR
CFC语言示例	
ST语言示例	 2 BYTE_1:=INT_TO_BCD(INT_1);
LD语言示例	
时序图	
使用限制	
注意事项	如果输入值小于0或者大于99，则返回错误值为255。

8.3.2 Gray 转换-BYTE_TO_GRAY

操作符	BYTE_TO_GRAY
功能说明	该功能块用作将一个字节数据转换成格雷特码
图形	
管脚定义	输入: B：字节型（BYTE）；该输入端为要转化的字节数据 输出: BYTE_TO_GRAY：字节型（BYTE）；转换后的格雷特码
变量声明	VAR BYTE_1:BYTE;; GRAY_1:BYTE; END_VAR
CFC语言示例	
ST语言示例	GRAY_1:=BYTE_TO_GRAY(BYTE_1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.2 Gray 转换-DWORD_TO_GRAY

操作符	DWORD_TO_GRAY
功能说明	该功能块用作将一个双字数据转换成格雷特码
图形	
管脚定义	输入： X：字节型（DWORD）；该输入端为要转化的双字数据 输出： DWORD_TO_GRAY：双字型（DWORD）；转换后的格雷特码
变量声明	VAR DWORD_1:DWORD;; GRAY_1:DWORD; END_VAR
CFC语言示例	
ST语言示例	GRAY_1:=DWORD_TO_GRAY(DWORD_1);
LD语言示例	
时序图	
使用限制	
注意事项	

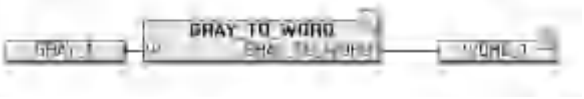
8.3.2 Gray 转换-GRAY_TO_BYTE

操作符	GRAY_TO_BYTE
功能说明	该功能块用作将一个格雷特码转换成一个字节数据
图形	
管脚定义	输入： B：字节型（BYTE）；该输入端为要转化的格雷特码 输出： GRAY_TO_BYTE：字节型（BYTE）；转换后的字节数据
变量声明	VAR GRAY_1:BYTE; BYTE_1:BYTE;; END_VAR
CFC语言示例	
ST语言示例	`BYTE_1:=GRAY_TO_BYTE(GRAY_1);`
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.2 Gray 转换-GRAY_TO_DWORD

操作符	GRAY_TO_DWORD
功能说明	该功能块用作将一个格雷特码转换成一个双字数据
图形	
管脚定义	输入： X：双字型（DWORD）；该输入端为要转化的格雷特码 输出： GRAY_TO_DWORD：双字型（DWORD）；转换后的双字数据
变量声明	VAR GRAY_1:DWORD; DWORD_1:DWORD;; END_VAR
CFC语言示例	
ST语言示例	DWORD_1:=GRAY_TO_DWORD(GRAY_1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.2 Gray 转换-GRAY_TO_WORD

操作符	GRAY_TO_WORD
功能说明	该功能块用作将一个格雷特码转换成一个字数据
图形	
管脚定义	输入: W：字型（WORD）；该输入端为要转化的格雷特码 输出: GRAY_TO_WORD：字型（WORD）；转换后的字型数据
变量声明	VAR GRAY_1:WORD; WORD_1:WORD;; END_VAR
CFC语言示例	
ST语言示例	`WORD_1:=GRAY_TO_WORD(GRAY_1);`
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.2 Gray 转换-WORD_TO_GRAY

操作符	WORD_TO_GRAY
功能说明	该功能块用作将一个字型数据转换成格雷特码
图形	
管脚定义	输入: W：字型（WORD）；该输入端为要转化的字数据 输出: WORD_TO_GRAY：字型（WORD）；转换后的格雷特码
变量声明	VAR WORD_1:WORD;; GRAY_1:WORD; END_VAR
CFC语言示例	
ST语言示例	GRAY_1:=WORD_TO_GRAY(WORD_1);
LD语言示例	
时序图	
使用限制	
注意事项	

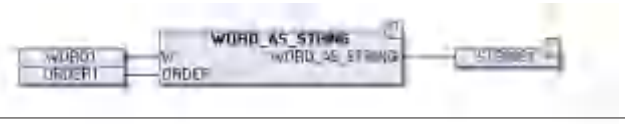
8.3.3 HEX/ASCII 功能-BYTE_TO_HEXinASCII

操作符	BYTE_TO_HEXinASCII
功能说明	该功能块用作一个二进制的字节转化成十六进制的ASCII码字
图形	
管脚定义	输入: B：字节型（BYTE），该输入为要转化的字节型数据 输出: BYTE_TO_HEXinASCII：字型（WORD），该输出为转化后的十六进制ASCII码字
变量声明	VAR BYTE1:BYTE; HEXinASCII1:WORD; END_VAR
CFC语言示例	
ST语言示例	HEXinASCII1:=BYTE_TO_HEXinASCII(BYTE1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.3 HEX/ASCII 功能-HEXinASCII_TO_BYTE

操作符	HEXinASCII_TO_BYTE
功能说明	该功能块用作一个十六进制的ASCII码字转化成二进制的字节
图形	
管脚定义	输入： W：字型（WORD）；该输入为要转化的十六进制ASCII码字 输出： HEXinASCII_TO_BYTE：字型（BYTE）；该输出为转化后的二进制字节
变量声明	VAR WORD1:WORD; BYTE1:BYTE; END_VAR
CFC语言示例	
ST语言示例	BYTE1:=HEXinASCII_TO_BYTE(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

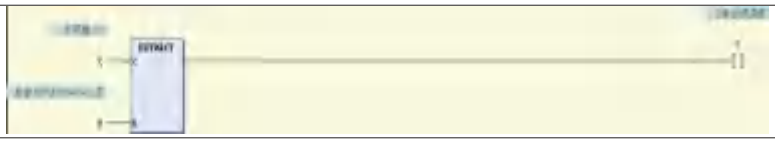
8.3.3 HEX/ASCII 功能-WORD_AS_STRING

操作符	WORD_AS_STRING
功能说明	该功能块用作一个十六进制的字转化成字符串
图形	
管脚定义	输入： W：字型（WORD）；该输入为要转化的字 ORDER:布尔型（BOOL）：该输入决定了转化的字是否高低字节互换 输出： WORD_AS_STRING：字符串型（STRING）；该输出为转化后的字符串
变量声明	VAR WORD1:WORD; ORDER1:BOOL; STRING1:STRING; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

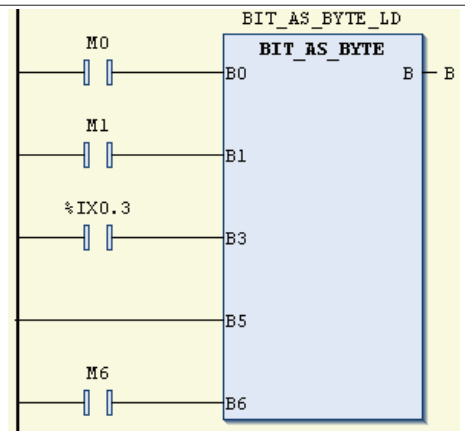
8.3.4位/字节算法功能块-EXTRACT

操作符	EXTRACT
功能说明	位提取(函数)--提取指定的双字X中的某一位N(有效范围从0到31)的状态,将该位的状态True(1)或False(0)输出
图形	
管脚定义	输入: X: 双整型（DWORD）；要提取的数据,连接双整型变量或双整型直接地址(如%MD2) N: 字节型（BYTE）；指定的提取位,连接字节型变量或字节型直接地址(如%MB3) 输出: EXTRACT:布尔型(BOOL);提取出的位状态,连接Bool型变量或直接地址(如%QX0.4),输出结果为True或Flase

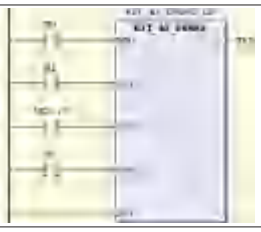
变量声明	VAR
	X: DWORD;// 双字输入X
	N: BYTE;// 指定双字X的N=0-31位
	
	Y: BOOL;// Y输出真或假

CFC语言示例	 举例:如果X=2,N=1时,输出Y=True; N=3时,输出Y=Flase;
ST语言示例	`Y:=EXTRACT(X, N);`
LD语言示例	
时序图	
使用限制	
注意事项	1.指定位N的有效数据范围是0到31, N值在32到255之间时,N/32的余数值是N的有效数据,例如N=33时相当于N=1; 2.该函数在CFC中输出引脚有名称定义,在LD梯形图和ST中只有引脚.无名称;

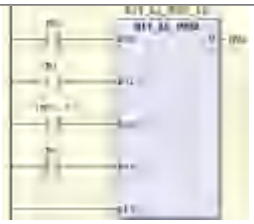
8.3.4位/字节算法功能块-BIT_AS_BYTE

操作符	BIT_AS_BYTE
功能说明	位整合字节(功能块)--将输入位B0到B7转换为字节B(B0对应字节的0位,B7对应字节的7位)
图形	
管脚定义	输入: B0：布尔型（BOOL）；要整合的位变量,连接布尔型变量或布尔型直接地址(如%IX0.3); B1到B7的定义同上,B0到B7管脚对应输入字节B的0到7位; 输出: B: 字节型(BYTE);整合后的字节,连接BYTE型变量或字节型直接地址(如%MB2)
变量声明	VAR BIT_AS_BYTE_LD: BIT_AS_BYTE; M0: BOOL; M1: BOOL; M6: BOOL; B: BYTE;
CFC语言示例	 举例:如果输入M0,M1,M5,%IX0.3都=False,则输出B=32或16#20;
ST语言示例	BIT_AS_BYTE_ST(B0:= M0, B1:= M1, B3:= %IX0.3, B5:= TRUE, B6:= %M6, B=>B);
LD语言示例	
时序图	
使用限制	
注意事项	1.示例中的程序假设输入引脚B2,B4,B7不使用,因此删除了不显示,不影响正常使用; 2.在变量声明中仅声明了M0,M1,M6,B,其它没有声明是因为不使用或直接地址%IX0.3; 3.使用功能块在BOOL类型和BYTE,WORD,DWORD整数类型间转换时,使用16进制显示方式调试更直观(在菜单'调试/观察''下有'显示模式可选择显示为'十进制'或'十六进制'';

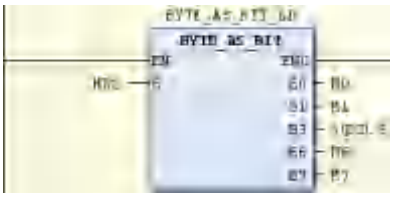
8.3.4位/字节算法功能块-BIT_AS_DWORD

操作符	BIT_AS_DWORD
功能说明	位整合双字(功能块)--将输入位B00到B31转换为双整型字X(B00对应双整型字X的0位,B31对应双整型字X的31位)
图形	
管脚定义	输入: B00：布尔型（BOOL）；要整合的位变量,连接布尔型变量或布尔型直接地址(如%MX0.3); B01到B31的定义同上,B00到B31管脚对应输入双字X的0到31位; 输出: X:　双整型(DWORD);整合后的双字,连接DWORD型变量或DWORD型直接地址(如%MD2)
变量声明	VAR BIT_AS_BYTE_LD: BIT_AS_DWORD; M0: BOOL; M1: BOOL; M6: BOOL; MD2: DWORD;　END_VAR
CFC语言示例	 举例:如果输入M0,M1,M6,%MX0.3都=False,则输出MD2=214748364816#80000000,即2的31次方;
ST语言示例	BIT_AS_DWORD_ST(B00:= M0, B01:= M1, B03:= %MX0.3, B06:= M6, B31:= TRUE, X=>MD2);
LD语言示例	
时序图	
使用限制	
注意事项	1.示例中的程序假设仅使用了输入引脚B00,B01,B03,B06,B31,其它引脚不使用,因此删除了不显示,不影响正常使用; 2.在变量声明中仅声明了M0,M1,M6,MD2其它没有声明是因为不使用或直接地址%MX0.3; 3.使用功能块在BOOL类型和BYTE,WORD,DWORD整数类型间转换时,使用16进制显示方式调试更直观(在菜单'调试/观察'下有'显示模式可选择显示为'十进制'或'十六进制'';

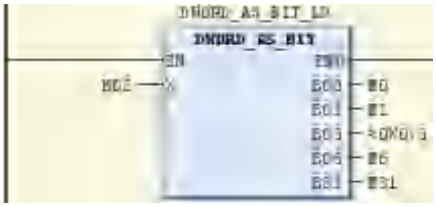
8.3.4位/字节算法功能块-BIT_AS_WORD

操作符	BIT_AS_WORD
功能说明	位整合字(功能块)--将输入位B00到B15转换为整型字W(B00对应整型字W的0位,B15对应整型字W的15位)
图形	
管脚定义	输入: B00：布尔型（BOOL）；要整合的位变量,连接布尔型变量或布尔型直接地址(如%MX0.3); B01到B15的定义同上,B00到B15管脚对应输入字W的0到15位; 输出: W:　整型(WORD);整合后的字,连接WORD型变量或WORD型直接地址(如%MW2)
变量声明	VAR BIT_AS_BYTE_LD: BIT_AS_WORD; M0: BOOL; M1: BOOL; M6: BOOL; MW2: WORD;　END_VAR
CFC语言示例	 举例:如果输入M0,M1,M6,%MX0.3都=False,则输出MW2=32768或16#8000;即2的15次方;
ST语言示例	BIT_AS_WORD_ST(B00:= M0, B01:= M1, B03:= %MX0.3, B06:= M6, B15:= TRUE, W=>MW2);
LD语言示例	
时序图	
使用限制	
注意事项	1.示例中的程序假设仅使用了输入引脚B00,B01,B03,B06,B15,其它引脚不使用,因此删除了不显示,不影响正常使用; 2.在变量声明中仅声明了M0,M1,M6,MW2,其它没有声明是因为不使用或直接地址%MX0.3; 3.使用功能块在BOOL类型和BYTE,WORD,DWORD整数类型间转换时,使用16进制显示方式调试更直观(在菜单'调试/观察'下有'显示模式可选择显示为'十进制'或'十六进制';

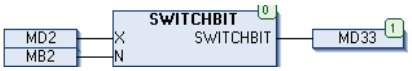
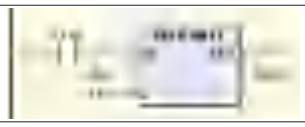
8.3.4位/字节算法功能块-BYTE_AS_BIT

操作符	BYTE_AS_BIT
功能说明	字节拆位(功能块)--将输入字节型变量B转换为Bool型变量位B0到B7(B0对应整型字B的0位,B7对应整型字B的7位)
图形	
管脚定义	输入: B: 字节型(BYTE);拆分的字节,连接BYTE型变量或BYTE型直接地址(如%MB2) 输出: B0：布尔型（BOOL）；拆分后的位,连接布尔型变量或布尔型直接地址(如%MX0.3); B1到B7的定义同上,B0到B7管脚对应输入字节B的0到7位;
变量声明	VAR BYTE_AS_BIT_LD: BYTE_AS_BIT; MB2: BYTE:=3; M0: BOOL; M1: BOOL; M6,M7: BOOL; END_VAR
CFC语言示例	 举例:如果输入MB2=3时,则输出M0=TRUE,M1=TRUE,其余%MX0.3,M6,M7等都=False;
ST语言示例	BYTE_AS_BIT_ST(B:= MB2, B00=> M0, B01=> M1, B03=> %QX0.3, B06=> M6, B7=> M7);
LD语言示例	
时序图	
使用限制	
注意事项	1.示例中的程序假设仅使用了输入引脚B0,B1,B3,B6,B7,其它引脚不使用,因此删除了不显示,不影响正常使用; 2.在变量声明中仅声明了M0,M1,M6,M7,MB2,其它没有声明是因为不使用或直接地址%QX0.3; 3.使用功能块在BOOL类型和BYTE,WORD,DWORD整数类型间转换时,使用16进制方式调试更直观(在菜单'调试/观察'下有'显示模式可选择显示为'十进制'或'十六进制';

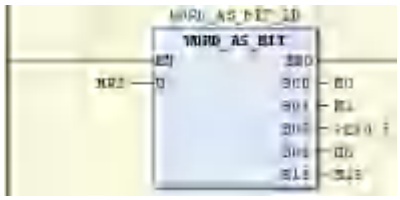
8.3.4位/字节算法功能块-DWORD_AS_BIT

操作符	DWORD_AS_BIT
功能说明	双字拆位(功能块)--将输入双整型字X转换为Bool位B00到B31(B00对应双整型字X的0位,B31对应双整型字X的31位)
图形	
管脚定义	输入: X: 双整型(DWORD);拆分的双字,连接DWORD型变量或DWORD型直接地址(如%MD2) 输出: B00: 布尔型（BOOL）; 拆分后的位,连接布尔型变量或布尔型直接地址(如%MX0.3); B01到B31的定义同上,B00到B31管脚对应输入双字X的0到31位;
变量声明	VAR DWORD_AS_BIT_LD: DWORD_AS_BIT; M0: BOOL; M1: BOOL; M6,M31: BOOL; MD2: DWORD :=3; END_VAR
CFC语言示例	 举例:如果输入MD2=3时,则输出M0=TRUE,M1=TRUE,其余%MX0.3,M6,M31等都=False;
ST语言示例	DWORD_AS_BIT_ST(X:= MD2, B00=> M0, B01=> M1, B03=> %QX0.3, B06=> M6, B31=> M31);
LD语言示例	
时序图	
使用限制	
注意事项	1.示例中的程序假设仅使用了输入引脚B00,B01,B03,B06,B31,其它引脚不使用,因此删除了不显示,不影响正常使用; 2.在变量声明中仅声明了M0,M1,M6,M31,MD2,其它没有声明是因为不使用或直接地址%QX0.3; 3.使用功能块在BOOL类型和BYTE,WORD,DWORD整数类型间转换时,使用16进制显示方式调试更直观(在菜单'调试/观察'下有'显示模式可选择显示为'十进制'或'十六进制';

8.3.4位/字节算法功能块-SWITCHBIT

操作符	SWITCHBIT
功能说明	双字指定位取反(函数)--将双字X中的某一位N(有效范围从0到31)的开关状态切换翻转后,输出切换后的双字值
图形	
管脚定义	输入: X: 双整型（DWORD）；双字数据,连接双整型变量或双整型直接地址(如%MD2) N: 字节型（BYTE）；指定的位,连接字节型变量或字节型直接地址(如%MB3) 输出: SWITCHBIT: 双整型（DWORD）;指定位取反后的双字,连接双整型变量或双整型直接地址(如%MD33)
变量声明	VAR MD2: DWORD; MB2: BYTE; MD33: DWORD; END_VAR
CFC语言示例	 举例:如果X=7,N=0时,函数输出值=6; N=3时,函数输出值=15;
ST语言示例	MD33:=SWITCHBIT(MD2, MB2);
LD语言示例	
时序图	
使用限制	
注意事项	1.指定位N的有效数据范围是0到31, N值在32到255之间时,N/32的余数值是N的有效数据,例如N=33时相当于N=1; 2.该函数在CFC中输出引脚有名称定义,在LD梯形图和ST中只有引脚.无名称;

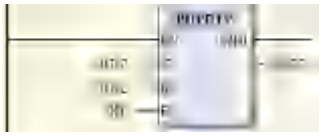
8.3.4位/字节算法功能块-WORD_AS_BIT

操作符	WORD_AS_BIT
功能说明	字拆位(功能块)--将输入整型字W转换为Bool位B00到B15(B00对应整型字W的0位,B15对应整型字W的15位)
图形	
管脚定义	输入: W: 拆分的字,双整型(WORD);连接WORD型变量或WORD型直接地址(如%MW2) 输出: B00：布尔型（BOOL）；拆分后的位,连接布尔型变量或布尔型直接地址(如%MX0.3); B01到B15的定义同上,B00到B15管脚对应输入字W的0到15位;
变量声明	VAR WORD_AS_BIT_LD: WORD_AS_BIT; MW2: WORD:=3; M0: BOOL; M1: BOOL; M6,M15: BOOL; END_VAR
CFC语言示例	 举例:如果输入MW2=3时,则输出M0=TRUE,M1=TRUE,其余%MX0.3,M6,M15等都=False;
ST语言示例	WORD_AS_BIT_ST(W:= MW2, B00=> M0, B01=> M1, B03=> %QX0.3, B06=> M6, B15=> M15);
LD语言示例	
时序图	
使用限制	
注意事项	1.示例中的程序假设仅使用了输入引脚B00,B01,B03,B06,B15,其它引脚不使用,因此删除了不显示,不影响正常使用; 2.在变量声明中仅声明了M0,M1,M6,MW2,其它没有声明是因为不使用或直接地址%QX0.3; 3.使用功能块在BOOL类型和BYTE,WORD,DWORD整数类型间转换时,使用16进制显示方式调试更直观(在菜单'调试/观察'下有'显示模式可选择显示为'十进制'或'十六进制';

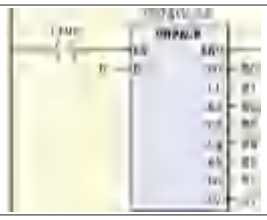
8.3.4位/字节算法功能块-PACK

操作符	PACK
功能说明	位整合字节(函数)--将8个布尔类型的输入位B0, B1, ..., B7整合为1个字节,是UNPACK的逆操作
图形	
管脚定义	输入: B0: 布尔型(BOOL);整合前的位,连接BOOL型变量或直接地址(如%IX0.4); B1到B7的定义同上,B0到B7管脚对应输出字节PACK的0到7位; 输出: PACK:字节型(BYTE);整合的字节值,连接字节型变量或字节型直接地址(如%MB2)
变量声明	VAR M0,M1,M2,M3,M4,M5,M6,M7: BOOL; B:BYTE; END_VAR
CFC语言示例	 举例:如果M0到M7=True,输出B=255;如果M0=TRUE,M1到M7=FALSE,输出B=1;
ST语言示例	B:=PACK(B0:=M0, B1:=M1, B2:=M2, B3:=M3, B4:=M4, B5:=M5, B6:=M6, B7:=M7);
LD语言示例	
时序图	
使用限制	
注意事项	1.输入B0到B7不使用的引脚可以不写,例如:只用到B0和B7位时,只需将变量M0,M7或直接地址输入连接相应引脚即可,不使用的引脚状态为FALSE; 2.函数PACK在CFC中使用输出引脚为PACK,在LD和ST中使用时输出引脚没有标识;

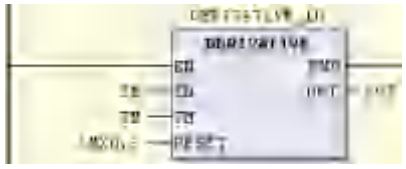
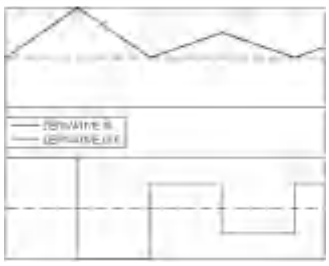
8.3.4位/字节算法功能块-PUTBIT

操作符	PUTBIT
功能说明	双字指定位赋值(函数)--设置X中的第N位为B,B是BOOL量,状态为TRUE或FALSE，从X的第0位开始算起
图形	
管脚定义	输入: X：双整型（DWORD）；双字数据,连接双整型变量或双整型直接地址(如%MD2) N：字节型（BYTE）；指定的赋值位,连接字节型变量或字节型直接地址(如%MB3) B: 布尔型(BOOL);赋值位的状态,连接BOOL型变量或BOOL型直接地址(如%MX0.2) 输出: PUTBIT: 布尔型(BOOL);指定位赋值后的双字,连接双整型变量或双整型直接地址(如%MD33)
变量声明	VAR MD2: DWORD; MB2:BYTE; M0: BOOL; MD33: DWORD; END_VAR
CFC语言示例	 举例:如果MD2=15,MB2=3;当M0=TRUE,则MD33=15,当M0=FALSE,则MD33=7;
ST语言示例	MD33:=PUTBIT(X:=MD2, N:=MB2, B:=M0);
LD语言示例	
时序图	
使用限制	
注意事项	1.指定位N的有效数据范围是0到31, N值在32到255之间时,N/32的余数值是N的有效数据,例如N=33时相当于N=1; 2.该函数在CFC中输出引脚有名称定义,在LD梯形图和ST中只有引脚.无名称;

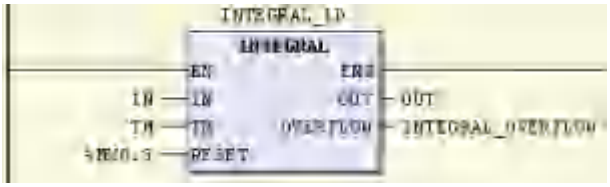
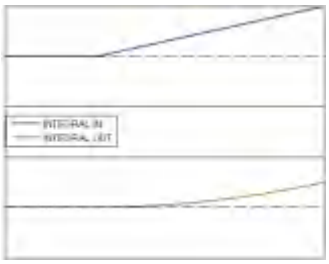
8.3.4位/字节算法功能块-UNPACK

操作符	UNPACK
功能说明	字节拆位(功能块)--将输入B由BYTE类型转换为8个BOOL类型的输出变量B0,...,B7,是PACK的逆操作
图形	
管脚定义	输入: B：字节型（BYTE）；拆分的字节,连接字节型变量或字节型直接地址(如%MB2) 输出: B0: 布尔型(BOOL);拆分后的位,连接BOOL型变量或直接地址(如%QX0.4),输出结果为TRUE或FALSE; B1到B7的定义同上,B0到B7管脚对应输入字节B的0到7位;
变量声明	VAR UNPACK_CFC: UNPACK; B: BYTE; M0: BOOL; M1: BOOL; M2 : BOOL; M3,M4,M5,M6,M7: BOOL; END_VAR
CFC语言示例	 举例:如果%MB2=255,输出M0到M7=TRUE;如果%MB2=2,输出M1=TRUE,其它输出FALSE;
ST语言示例	UNPACK_ST(B:=%MB2 , B0=>M0 , B1=>M1 , B2=>M2 , B3=>M3 , B4=>M4 , B5=>M5 , B6=>M6 , B7=>M7);
LD语言示例	
时序图	
使用限制	
注意事项	输出B0到B7不使用的引脚可以不写,例如:只用到B0和B7位时,只需将M0,M7输出关联即可;

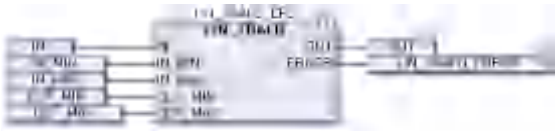
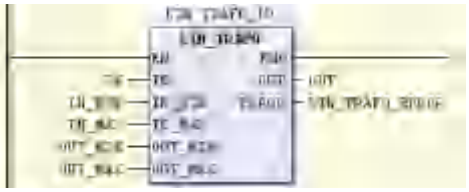
8.3.5数学辅助功能块-DERIVATIVE

操作符	DERIVATIVE
功能说明	微分(功能块)--对连续输入的变量IN按照时间TM(单位ms)微分,结果输出到OUT, 微分计算公式:OUT={3*[IN(N)-IN(N-3)]+IN(N-1)+IN(N-2)}/[3*TM(N-2)+4*TM(N-1)+3*TM(N)];
图形	
管脚定义	输入: IN: 实数型（REAL);连续输入的变量,连接实数型变量或立即数(如0.2)或整数型直接地址(如%MW2); TM: 双整型(DWORD);微分时间(毫秒),连接双整型变量或立即数(如5)或整数型直接地址(如%MD9); 数值越大微分效果越弱; RESET:布尔型(BOOL);复位信号,TRUE时OUT:=0;连接布尔型变量或布尔型直接地址(如%IX0.3); 输出: OUT:实数型（REAL）； 微分的结果,连接实数型变量;
变量声明	VAR DERIVATIVE_CFC: DERIVATIVE; IN: REAL; TM: DWORD; OUT: REAL; END_VAR
CFC语言示例	
ST语言示例	DERIVATIVE_ST(IN:=IN , TM:= TM, RESET:=%MX0.3 , OUT=> OUT);
LD语言示例	
时序图	
使用限制	
注意事项	

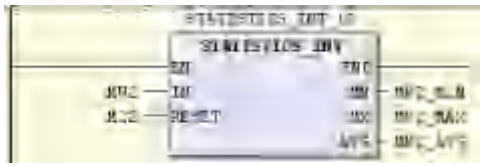
8.3.5数学辅助功能块-INTEGRAL

操作符	INTEGRAL
功能说明	积分(功能块)--对连续输入的变量IN按照时间TM(单位ms)积分,结果输出到OUT,当积分值OUT超出实数范围3.4028-23466e+38时OVERLOAD变为TRUE; 积分运算公式: OUT(N)=[A(N)+B(N)]/2, A(N)=A(N-1)+TM*IN(N-1); B(N)=B(N-1)+TM*IN(N-1); N,N-1为连续两次输入值;
图形	
管脚定义	输入: IN: 实数型（REAL）; 连续输入的变量,连接实数型变量或立即数(如0.2)或整数型直接地址(如%MW2); TM: 双整型(DWORD); 积分时间(毫秒),连接双整型变量或立即数(如5)或整数型直接地址(如%MD9);数值越大积分效果越强; RESET:布尔型(BOOL);复位信号,TRUE时OUT:=0,OVERLOAD:=FALSE;连接布尔型变量或布尔型直接地址(如%IX0.3); 输出: OUT:实数型（REAL）; 积分的结果,连接实数型变量; OVERLOAD:布尔型(BOOL);积分值溢出;连接布尔型变量或布尔型直接地址(如%QX0.3),超出实数范围3.402823466e+38时变为TRUE;
变量声明	VAR DERIVATIVE_CFC: DERIVATIVE; IN: REAL; TM: DWORD; OUT: REAL; INTEGRAL_OVERFLOW:BOOL; END_VAR
CFC语言示例	
ST语言示例	INTEGRAL_ST(IN:= IN, TM:= TM, RESET:= %MX0.3, OUT=> OUT, OVERFLOW=>INTEGRAL_OVERFLOW);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.5数学辅助功能块-LIN_TRAFO

操作符	LIN_TRAFO
功能说明	"线性转换(功能块)--将在原始下限和上限值范围内的输入实数，转换为由新的工程下限和上限值确定的范围内的实数。下面的公式是转换的基础:(IN - IN_MIN) : (IN_MAX - IN) = (OUT - OUT_MIN) : (OUT_MAX - OUT)"
图形	
管脚定义	输入: IN：实数型（REAL）；输入值,连接实数型变量或立即数(如0.2)或整数型直接地址(如%MW2); IN_MIN(输入值下限),IN_MAX(输入值上限),OUT_MIN(输出值下限),OUT_MAX(输入值上限)的数据类型同上;(注意需要满足IN_MIN<IN_MAX) 输出: OUT:实数型（REAL）；转换后输出值,连接实数型变量; ERROR:布尔型(BOOL); 错误输出, 如果IN_MIN >= IN_MAX，或者输入值IN超出了 设定的输入值范围,即IN<IN_MIN或>IN_MAX，则发生错误，输出为TRUE;
变量声明	VAR LIN_TRAFO_CFC: LIN_TRAFO; IN: REAL; IN_MIN: REAL; IN_MAX: REAL; OUT_MIN: REAL; OUT_MAX: REAL; OUT: REAL; LIN_TRAFO_ERROR: BOOL; END_VAR
CFC语言示例	 举例:一个温度传感器量程为-50到+50度,精度12位的模拟输入量程为0到4095;则当输入值IN=2048时,输出值OUT=(50-(-50))/(4095-0)*2048=0度
ST语言示例	LIN_TRAFO_ST(IN:= IN, IN_MIN:= IN_MIN, IN_MAX:= IN_MAX , OUT_MIN:= OUT_MIN, OUT_MAX:= OUT_MAX, OUT=> OUT, ERROR=> LIN_TRAFO_ERROR);
LD语言示例	
时序图	
使用限制	
注意事项	1.IN_MAX必须 >= IN_MIN,否则ERROR引脚会报错; 2.IN需要满足IN_MIN<=IN<=IN_MAX,否则ERROR引脚会报错;

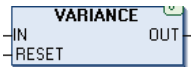
8.3.5数学辅助功能块-STATISTICS_INT

操作符	STATISTICS_INT
功能说明	整型数据统计最大.最小.平均值(功能块)--按统计方法计算输入整数的最大值,最小值和平均值;
图形	
管脚定义	输入: IN：整数型（INT）；连续输入的整数,连接整数型变量或直接地址(如%MW2); RESET:布尔型(BOOL);复位信号,TRUE将输入IN赋值给输出MN,MX,AVG,连接布尔型变量或布尔型直接地址(如%IX0.3); 输出: MN:整数型（INT）；最小值,连接整数型变量或直接地址(如%MW33); MX（最大值）和AVG（平均值）的定义同上
变量声明	VAR STATISTICS_INT_CFC: STATISTICS_INT; MW2: INT; M22: BOOL; MW2_MIN: INT; MW2_MAX: INT; MW2_AVG: INT; END_VAR
CFC语言示例	
ST语言示例	STATISTICS_INT_ST(IN:= MW2, RESET:=M22 , MN=>MW2_MIN , MX=>MW2_MAX , AVG=>MW2_AVG);
LD语言示例	
时序图	
使用限制	
注意事项	1.输入复位RESET=TRUE时,输出引脚的值MN:=MX:=AVG:=IN; 2.输出引脚AVG的值是以时间积分出的统计平均数,不是求和平均值;

8.3.5数学辅助功能块-STATISTICS_REAL

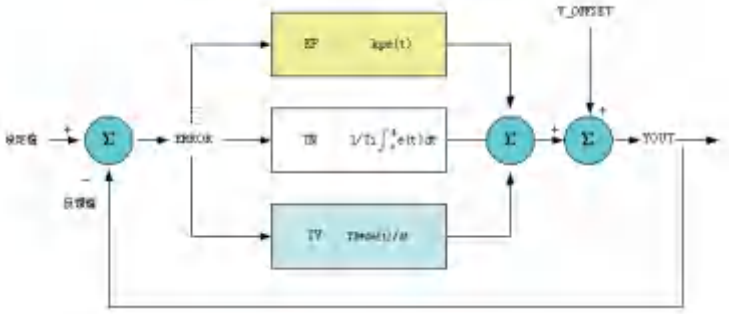
操作符	STATISTICS_REAL
功能说明	实型数据统计最大.最小.平均值(功能块)--按统计方法计算输入实数的最大值,最小值和平均值;
图形	
管脚定义	输入: IN：数型（REAL）；连续输入的实数,连接整数型变量或直接地址(如%MF2); RESET:布尔型(BOOL);复位信号,TRUE将输入IN赋值给输出MN,MX,AVG,连接布尔型变量或布尔型直接地址(如%IX0.3); 输出: MN:实数型（REAL）；最小值, 连接实数型变量; MX（最大值）和AVG（平均值）的定义同上
变量声明	VAR STATISTICS_REAL_CFC: STATISTICS_REAL; MF2: REAL; M22: BOOL; MF2_MIN: REAL; MF2_MAX: REAL; MF2_AVG: REAL; END_VAR
CFC语言示例	
ST语言示例	STATISTICS_REAL_ST(IN:= MF2, RESET:=M22 , MN=>MF2_MIN , MX=>MF2_MAX , AVG=>MF2_AVG);
LD语言示例	
时序图	
使用限制	
注意事项	1.输入复位RESET=TRUE时,输出引脚的值MN:=MX:=AVG:=IN; 2.输出引脚AVG的值是以时间积分出的统计平均数,不是求和平均值;

8.3.5数学辅助功能块-VARIANCE

操作符	VARIANCE
功能说明	平方偏差(功能块)--计算输入值的方差;方差是一个统计函数，其定义为一组样本数据的偏离程度，等于标准差的平方。
图形	
管脚定义	输入: IN：实数型（REAL）；连续输入的实数,连接实数型变量或直接地址(如%MD2); RESET:布尔型(BOOL);复位信号,TRUE时OUT=0,连接布尔型变量或布尔型直接地址(如%IX0.3); 输出: OUT:实数型（REAL）；平均方差值,连接实数型变量;
变量声明	VAR VARIANCE_CFC: VARIANCE; MF2: REAL; M22: BOOL; MF33: REAL; END_VAR
CFC语言示例	 举例:当M22=FALSE时,MF2数据有变化时MF33输出连续变化的方差值,当M22=TRUE时,输出MF33=0;
ST语言示例	VARIANCE_ST(IN:=MF2 , RESET:=M22 , OUT=>MF33);
LD语言示例	
时序图	
使用限制	
注意事项	1.输入复位reset=true时，输出引脚的out值为0

8.3.6调节器-PID调节功能介绍

PID控制是用于过程控制的一种常用方法，通过对被控对象的反馈信号与目标信号的差量进行比例、积分、微分运算，来调节目标信号，达到快速准确响应的目的。适用于流量控制，压力控制、温度控制等过程控制。其控制框图如下:

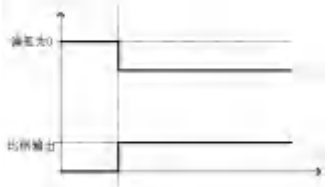


PID控制器由比例单元（P）、积分单元（I）和微分单元（D）组成。其输入e (t)与输出u (t)的关系为u(t)=kp(e(t)+1/Ti∫e(t)dt+Td*de(t)/dt) 式中积分的上下限分别是0和t。其中kp为比例系数； Ti为积分时间常数； Td为微分时间常数

比例（P）控制

比例控制是一种最简单的控制方式。属于当前控制，控制的是当前误差，系统一旦出现偏差，比例调节马上产生调节作用用来减少偏差，比例调节作用大，可以加快调节，减少偏差。但是比例作用过大会产生震荡导致系统不稳定。比例控制，只能调节当前时刻新产生的误差，而无法调节之前累积的误差，所以，当系统当前时刻的无法依靠比例控制完全消除时，误差就会随着时间的累积越来越大。这种误差我们称为稳态误差。因此，我们说当仅有比例控制时系统输出存在稳态误差。而且大部分系统中都会存在稳态误差。

比例控制的效果如下



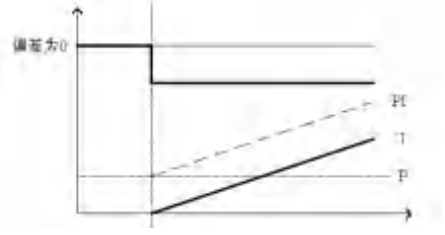
积分（I）控制

积分控制的作用就是消除系统的稳态误差，属于过去控制，即对过去存在误差的累积。在积分控制中，当系统存在偏差时，积分控制就进行控制，积分项的值取决于PLC每个扫描周期所记录的当时误差的积分，随着时间的增加，积分项会随误差的累积而改变。这样，无论系统在过去哪一时刻产生的比例控制未能消除的误差，积分项都会将它记录下来，它推动控制器的输出增大或减小使稳态误差进一步减小，直到趋向于零。因此，比例+积分(PI)控制器，可以使系统在进入稳态后无稳态误差。

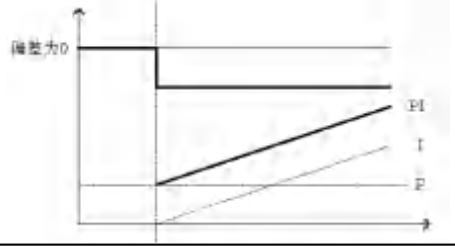
积分控制效果主要影响因素包括比例系数、积分时间和当前的偏差大小。

积分时间：积分时间表示积分控制的强弱的单位，积分时间越大表示积分的强度越弱，反之则越强，积分强度的增大可以快速达到目标值，强度过弱会导致达到目标值的时间变长，无法实现快速响应。但是强度的增加也会导致系统处于震荡状态，无法快速达到稳定状态。公式中的Ti时积分时间常数，与积分时间成反比。下图所示，当误差一定时，比例控制和比例积分控制的输出的区别。这只是一种假设的模型，便于观察控制器的输出。现实系统中误差是会收敛的。

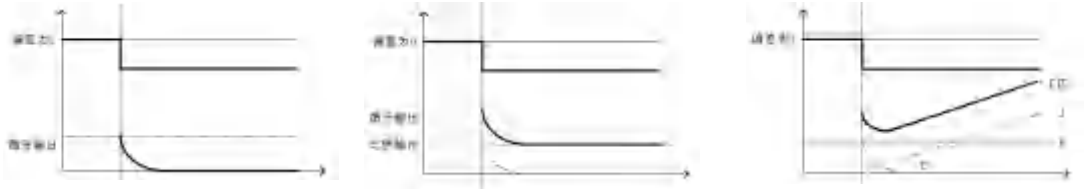
单纯积分效果如下:



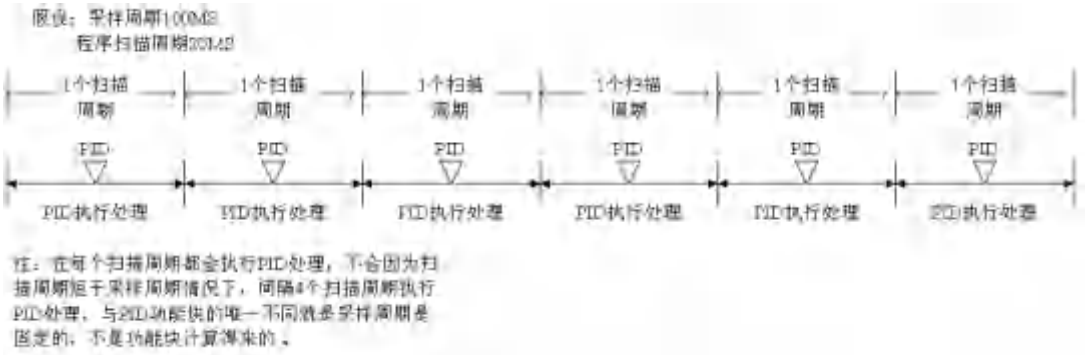
比例+积分效果如下:



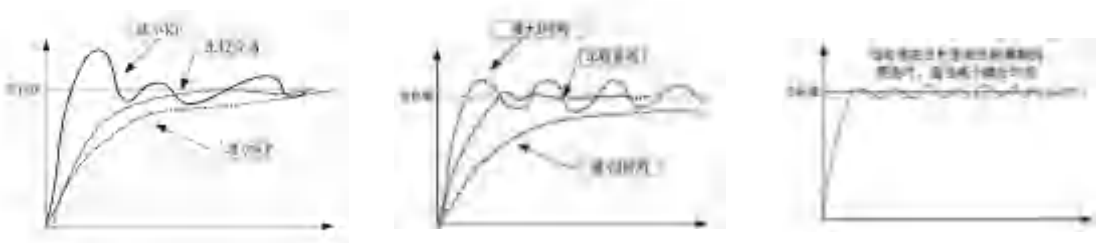
微分（D）控制
微分控制的作用主要是反映当前偏差的变化率，并且能根据当前的变化率来推测即将产生的偏差趋势从而进行预先控制，减少超调或误控制。因此微分控制属于预测将来控制。在微分控制中，控制器的输出与输入误差信号的微分（即误差的变化率）成正比关系。能够提前抑制误差，从而避免了被控量的严重超调。所以对有较大惯性或滞后的被控对象，比例+微分控制器能改善系统在调节过程中的动态特性。
微分控制效果的影响因素主要包括微分时间、比例增益和误差变化率。
微分时间：微分时间是微分控制强度的单位，微分时间越大，微分的强度越强，反之则越弱，但是微分时间调节并不总是改善系统的稳定性，过度的微分控制也会导致系统的震荡，无法实现快速稳定的效果。
单纯的微分控制效果如下：　　比例+微分控制效果如下：　　比例+积分+微分效果如下：



采样周期与扫描周期对PID功能块的影响
目前在PD、PID及PID_FIXCYCLE中前两个功能块的采样周期均取决于扫描周期。其数据由功能块的内部计数器计算出来的用于产生积分控制。而在PID_FIXCYCLE中采样周期由输入端子输入的，不是通过功能块内部计算得来的。采样周期的长短并不影响扫描周期内功能块的执行。如下图所示：

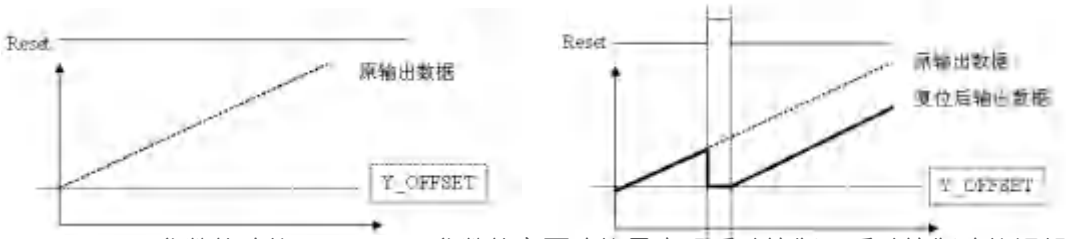


PID调节一般规律
1.在输出不振荡时，且稳态时误差大，增大比例增益P。如振荡，减小P。
2.在输出不振荡时，且稳态误差太大，减小积分时间常数Ti。
3.在输出不振荡时，且系统的跟随性差时增大微分时间常数Td。如振荡，减小Td。
在调节PID时首先调节比例项，然后调节积分项，最后调节微分项。
首先对比例项进行调节，如下图所示　其次调节积分项，如下图所示：　　最后调节微分项，调节如下：

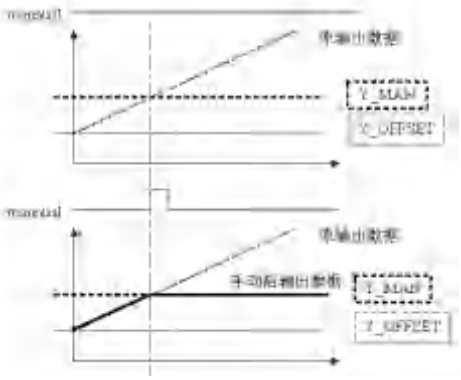


下面对PID功能块的各项功能进行说明

1．RESET参数的功能：RESET的作用是起到复位作用，包括KP、TV等参数的修改生效，以及MANUAL控制的复位。功能逻辑如下图所示：

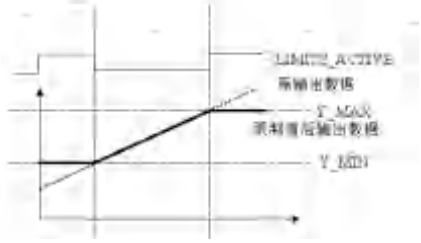


2．MANUAL参数的功能：MANUAL参数的主要功能是实现手动控制，手动控制功能逻辑如下：

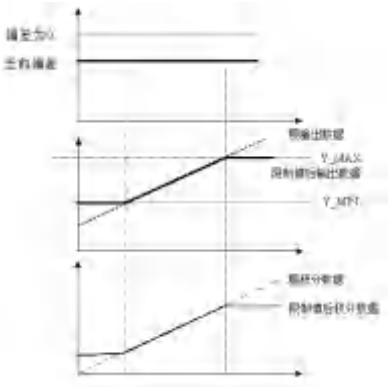


3.最大最小值的限制功能（Y_MIN,Y_MAX）

此功能用于限制输出数据在规定的范围之内，避免因为错误参数的输入或控制本身的问题而导致的错误等。该功能的使能没有单独的控制端子，而是通过Y_MIN<Y_MAX来使能该功能，当计算出的输出值小于Y_MIN或大于Y_MAX时，限制其为最小值或最大值，并且使能超范围布尔量limits_active，如不需要此功能则只要保持参数Y_MIN=Y_MAX=0即可；



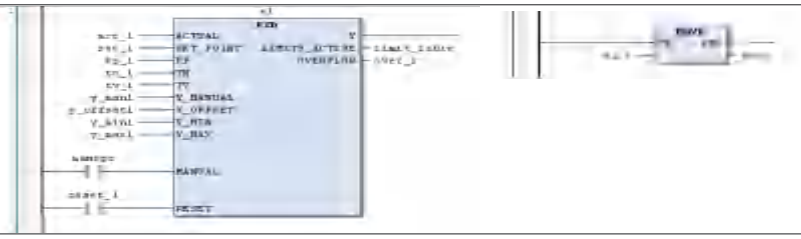
4.当输出达到最大最小限制值时，此时积分项会停止积分功能，用以保持当前积分值。只到计算出的输出值在最大值最小值范围之内后，积分项会立即再次激活用来消除稳态误差。



8.3.6调节器-PD

操作符	PD
功能说明	该功能块用作比例微分控制器
图形	
管脚定义	输入 类型 注释 ACTUAL REAL 实际值，过程变量 SET_POINT REAL 目标值，设定点 KP REAL 比例系数 (P) TV REAL 微分时间 (D)单位：秒 Y_MANUAL REAL 当手动信号MANUAL=TRUE时功能块输出值 Y_OFFSET REAL 非手动时输出偏移量 Y_MIN REAL 输出最小值 Y_MAX REAL 输出最大值 MANUAL BOOL 功能块输出值Y=Y_MANUAL,相当于旁路PID功能 FALSE: 功能块输出值Y由功能块运算得出 RESET BOOL TRUE:复位功能块输出Y=Y_OFFSET 输出 类型 注释 Y REAL 功能块计算输出值 LIMITS_ACTIVE BOOL 当输出值小于最小值或大于最大值时为TURE,否则为FALSE
变量声明	VAR act_1: REAL; set_1: REAL; kp_1: REAL; tn_1: REAL; tv_1: REAL; y_man1: REAL; y_offset1: REAL; y_min1: REAL; y_max1: REAL; manopr: BOOL; reset_1: BOOL; yout: REAL; limit_lable: BOOL; over_1: BOOL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	Y_OFFSET, Y_MIN和Y_MAX用于在规定的范围之内转换操作值。 MANUAL可以用于切换打开和关闭手动操作；RESET用于重新初始化控制器。 在正常操作时（MANUAL = RESET = LIMITS_ACTIVE = FALSE），控制器计算控制器偏差，也就是SET_POINT与ACTUAL的差值，同时得到相对于时间的导数dD/dt，并且在内部保存这些值。除了当前值变化以外，其他值的改变如KP，TV等变化均在重新初始化后才能生效！ 输出值，也就是操作值Y，按照以下公式计算： Y = KP × (D + TV dD/dt) + Y_OFFSET 这里 D=SET_POINT-ACTUAL 所以除了P部分以外，当前控制器误差的改变（D部分）也影响操作值。 另外，Y被由 Y_MIN 和 Y_MAX.规定很多场合中如果不做限制，|Y|可能会过大，将导致系统不稳定。的范围所限制。 如果Y超出范围，LIMITS_ACTIVE将为TRUE。如果不希望限制操作值， Y_MIN 和 Y_MAX必须设置为0。 一旦MANUAL=TRUE，Y_MANUAL将赋给Y。 通过将TV设置为0，可以容易的创建一个P控制器。

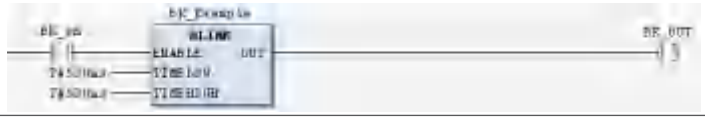
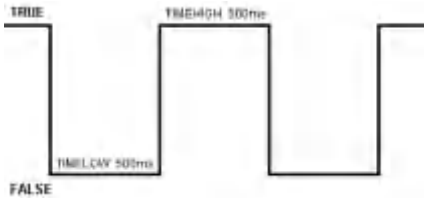
8.3.6调节器-PID

操作符	PID
功能说明	该功能块用作比例积分微分控制器，比PD多个积分项
图形	
管脚定义	输入　　类型　注释 ACTUAL　　REAL　实际值，过程变量 SET_POINT　　REAL　目标值，设定点 KP　　REAL　比例增益 TN　　REAL　积分时间，I部分单位增益的倒数，以秒为单位，例如0.5是500毫秒 TV　　REAL　微分时间 (D)单位：秒 Y_MANUAL　　REAL　当手动信号MANUAL=TRUE时功能块输出值 Y_OFFSET　　REAL　非手动时输出偏移量 Y_MIN　　REAL　限制输出最小值 Y_MAX　　REAL　限制输出最大值 MANUAL　　BOOL　TRUE: 功能块输出值Y=Y_MANUAL,相当于旁路PID功能 FALSE: 功能块输出值Y由功能块运算得出 RESET　　BOOL　TRUE:复位功能块输出Y=Y_OFFSET，同时复位积分项 输出　　类型　注释 Y　　REAL　功能块计算输出值 LIMITS_ACTIVE BOOL　当输出值小于最小值或大于最大值时为TURE,否则为FALSE OVERFLOW　　BOOL　此值为TRUE，表示积分项溢出
变量声明	VAR x1: PID; act_1　: REAL; set_1　: REAL; kp_1　: REAL; tn_1　: REAL; tv_1　: REAL; y_man1　: REAL; y_offset1: REAL; y_min1　: REAL; y_max1　: REAL; manopr　: BOOL; reset_1　: BOOL; yout　: REAL; limit_lable: BOOL; over_1　: BOOL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	TN≠0
注意事项	Y_OFFSET, Y_MIN和Y_MAX用于在规定的范围之内转换操作值。 MANUAL可以用于切换到手动操作；RESET用于重新初始化控制器。 在正常操作时（MANUAL = RESET = LIMITS_ACTIVE = FALSE），控制器计算控制器偏差，也就是SET_POINT 与 ndash; ACTUAL的差值，得到关于时间的导数de/dt，并且在内部保存这些值。 和PD控制器不同的是，输出操作值Y中包含了积分部分，按照以下公式计算： Y = KP × (D + 1/TN ∫edt + TV dD/dt) + Y_OFFSET 所以除了P部分以外，当前控制器误差的改变（D部分）和历史控制器误差（I部分）都影响操作值。 通过将TV设置为0，PID控制器可以容易的转换为一个PI控制器。如果需要用P控制器，请使用PD控制器转换。 因为存在积分部分，如果误差D的积分变的过大，则可能由于控制器参数不正确而产生溢出。因此为了安全，提供一个BOOL型输出OVERFLOW，在溢出时该值为TRUE。这个仅仅发生在控制系统由于不正确的参数而不稳定时。此时，控制器将暂停，并且只有重新初始化才能再次运行。

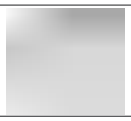
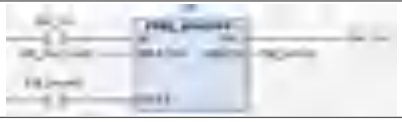
8.3.6调节器-PID_FIXCYCLE

操作符	PID_FIXCYCLE
功能说明	该功能块用作比例积分微分控制器，与PID不同之处是采样周期是固定的，不是由功能块的内部计数器计算出来的
图形	
管脚定义	输入 类型 注释 ACTUAL REAL 实际值，过程变量 SET_POINT REAL 目标值，设定点 KP REAL 比例增益 TN REAL 积分时间，I部分单位增益的倒数，以秒为单位，例如0.5是500毫秒 TV REAL 微分时间 (D)单位：秒 Y_MANUAL REAL 当手动信号MANUAL=TRUE时功能块输出值 Y_OFFSET REAL 非手动时输出偏移量 Y_MIN REAL 限制输出最小值 Y_MAX REAL 限制输出最大值 MANUAL BOOL TRUE: 功能块输出值Y=Y_MANUAL,相当于旁路PID功能 FALSE: 功能块输出值Y由功能块运算得出 RESET BOOL TRUE:复位功能块输出Y=Y_OFFSET ，同时复位积分项 CYCLE REAL 采样周期时间，单位为秒。例如：0.1为100ms 输出 类型 注释 Y REAL 功能块计算输出值 LIMITS_ACTIVE BOOL 当输出值小于最小值或大于最大值时为TURE,否则为FALSE OVERFLOW BOOL 此值为TRUE，表示积分项溢出
变量声明	VAR x1: PID_FIXCYCLE; act_1: REAL; set_1: REAL; kp_1: REAL; tn_1: REAL; tv_1: REAL; y_man1: REAL; y_offset1: REAL; y_min1: REAL; y_max1: REAL; manopr: BOOL; reset_1: BOOL; yout: REAL; limit_lable: BOOL; over_1: BOOL; cycle_1:REAL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	"与PID功能块的区别就是采样周期是固定的"

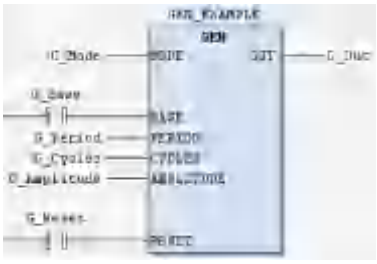
8.3.7信号发生器-BLINK

操作符	BLINK
功能说明	此功能块输出方波脉冲信号。
图形	
管脚定义	输入　　类型　注释 ENABLE　　BOOL　功能块使能信号，高电平有效。 TIMELOW　　TIME　输出脉冲信号一个周期中，低电平保持时间。 TIMEHIGH　　TIME　输出脉冲信号一个周期中，高电平保持时间。 输出　　类型　注释 OUT　　BOOL　输出脉冲信号。
变量声明	VAR BK_Example: BLINK; BK_en: BOOL; BK_OUT: BOOL; END_VAR
CFC语言示例	
ST语言示例	BK_Example(ENABLE:=BK_en , TIMELOW:=T#500ms , TIMEHIGH:=T#500ms , OUT=>BK_OUT);
LD语言示例	
注意事项	上述功能块OUT引脚输出波形如图所示： 

8.3.7信号发生器-FREQ_MEASURE

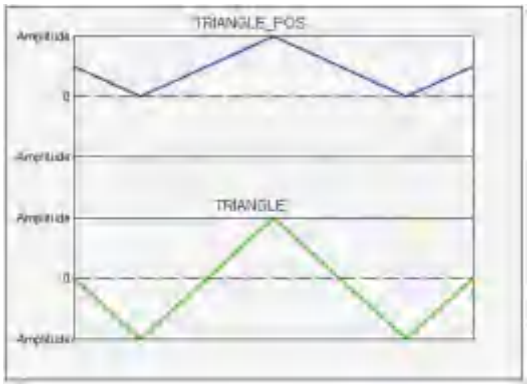
操作符	FREQ_MEASURE
功能说明	此功能块测量一个布尔型输入信号的频率，单位Hz。
图形	
管脚定义	输入　　类型　注释 IN　　BOOL　输入信号，脉冲上升沿有效。 PERIODS　INT　输入信号采样周期，取值范围：1~ 10。 RESET　　BOOL　复位所有参数到0，高电平有效。 输出　　类型　注释 OUT　　REAL　测量频率值，单位Hz。 VALID　　BOOL　功能块标志位；参数设置正确，且功能块工作时，为TURE；否则为FALSE。
变量声明	VAR FM: FREQ_MEASURE; FM_In: BOOL; FM_Periods: INT; FM_Reset: BOOL; FM_Out: REAL; FM_Valid: BOOL; END_VAR
CFC语言示例	
ST语言示例	FM(IN:=FM_In, PERIODS:=FM_Periods , RESET:=FM_Reset , OUT=>FM_Out , VALID=>FM_Valid);
LD语言示例	
使用限制	推荐测量频率范围：0~10Hz；超出此范围会有一定误差。
注意事项	① 在编写功能块时，如果直接在PERIODS引脚定义变量，默认的变量类型为INT (1..10)； 注意此时需要将变量类型改为INT，否则程序编译时会报错。 ② PERIODS设置范围：1~ 10，否则功能块将不能正常工作； 此参数意义为：设置输入信号若干个周期测量频率的平均值；例如PERIODS=3，表示OUT输出值为3个上升沿周期 测量频率平均值。 ③ VALID正常工作时为TRUE；以下情况时，VALID将为FALSE： 功能块参数设置不正确； 功能块没有工作； 功能块工作时，输入信号频率为0（当输入信号2次上升沿间隔时间>3*OUT）。

8.3.7信号发生器-GEN

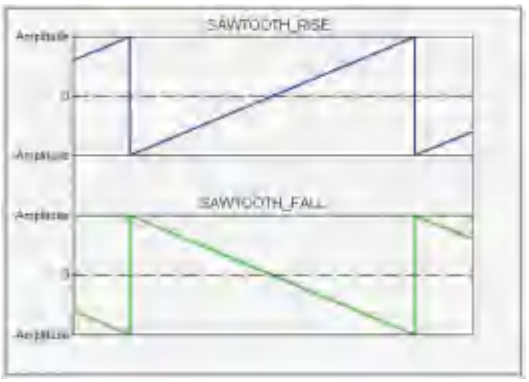
操作符	GEN
功能说明	此功能块提供函数发生器功能，可产生7种不同波形，并以INT变量形式输出。
图形	
管脚定义	输入　　类型　　注释 MODE　　GEN_MODE　函数波形选择引脚。可选择的函数模式： TRIANGLE:　三角波函数，幅值-Amplitude~ +Amplitude； TRIANGLE_POS: 三角波函数，幅值0~ +Amplitude； SAWTOOTH_RISE: 上升锯齿函数； SAWTOOTH_FALL: 下降锯齿函数； RECTANGLE:　矩形函数； SINE:　正弦函数； COSINE:　余弦函数。 （文后见不同MODE类型的输出图示） BASE　　BOOL　　当BASE=TRUE时，函数波形的周期取决于PERIOD参数； 当BASE=FALSE时，函数波形的周期取决于CYCLE。 PERIOD　　TIME　　函数波形的周期，单位为时间。 CYCLES　　INT　　函数波形的周期，以GEN功能块被调用的次数表示。 AMPLITUDE　INT　　函数波形的幅值。 RESET　　BOOL　　当RESET=TRUE时，函数发生器重新设置为0。 输出　　类型　　注释 OUT　　INT　　函数发生器生成的波形输出变量。
变量声明	VAR GEN_EXAMPLE: GEN; G_Mode: GEN_MODE; G_Base: BOOL; G_Period: TIME; G_Cycles: INT; G_Amplitude: INT; G_Reset: BOOL; G_Out: INT; END_VAR
CFC语言示例	
ST语言示例	GEN_EXAMPLE(MODE:=G_Mode , BASE:=G_Base , PERIOD:=G_Period , CYCLES:=G_Cycles , AMPLITUDE:=G_Amplitude , RESET:=G_Reset , OUT=>G_Out);
LD语言示例	
注意事项	① 当BASE=FALSE时，函数周期取决于CYCLES，例如CYCLES=1000，则表示该函数波形经GEN功能块执行1000次后输出一个周期； 此时函数周期时间等于PLC1000次扫描周期的时间总和（前提是每一个扫描周期，GEN只被调用一次） 当BASE=FALSE时，CYCLES的值不能为0，否则PLC会报程序异常错误；可在程序中对CYCLES赋予初始值。 ② 当BASE=TRUE时，即函数周期取决于PERIOD设定值，PERIOD的值不能为0，否则PLC会报程序异常错误；可在程序中对PERIOD赋予初始值。

不同MODE类型的输出图示:

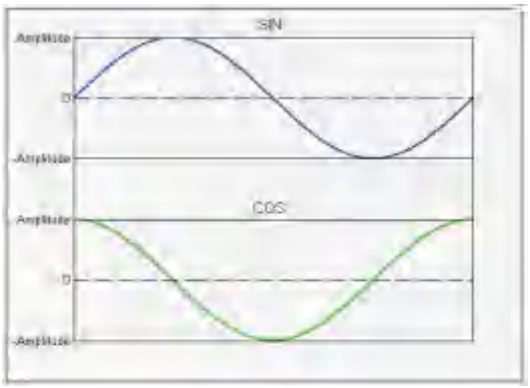
图(1)



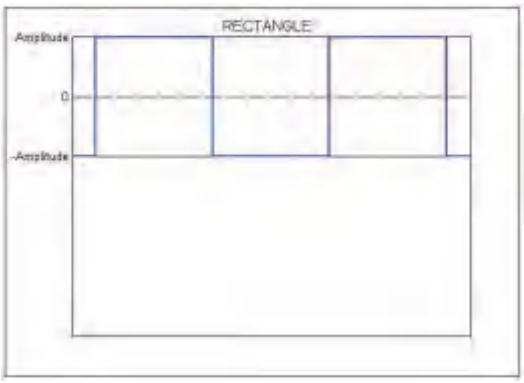
图(2)



图(3)



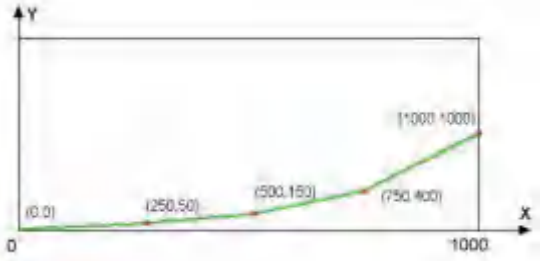
图(4)



8.3.8函数操纵功能块-CHARCURVE

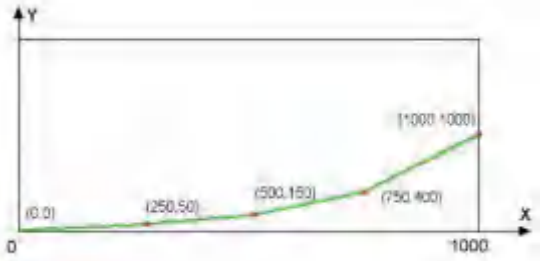
操作符	CHARCURVE
功能说明	此功能块将给定的若干个数据点，分段进行线性化。
图形	
管脚定义	输入 类型 注释 IN INT 线性化的点的X坐标值； N BYTE 线性化的点个数，范围为2≤N≤11； P ARRAY [0..10] OF POINT POINT类型变量由两个INT值(X 和 Y)组成，表示一个点；此数组一共可存储11个点，分别为P[0]~ P[10]； 输出 类型 注释 OUT INT 点列线性化后输出的操作值； ERR BYTE 错误代码；正常时为0。
变量声明	VAR Charcurve_Example: CHARCURVE; CHV_Input: INT; CHV_N: BYTE; CHV_P: ARRAY [0..10] OF POINT; CHV_OUT: INT; CHV_ERR: BYTE; END_VAR
CFC语言示例	
ST语言示例	Charcurve_Example(IN:=CHV_Input , N:=CHV_N , P:=CHV_P , OUT=>CHV_OUT , ERR=>CHV_ERR);
LD语言示例	
注意事项	① 输入N指定了使用P中的点的数量，如N=3，则P[0]~ P[2]这3个点用于线性化； ② 数组中的点 P[0]..P[N-1] 必须依照X值的大小，升序存储，否则ERR为1； ③ 如果输入IN不在P[0].X 和 P[N-1].X之内，则ERR=2，此时OUT=P[0]. Y或者 P[N-1].Y； 如果输入N超出了2~11之间的允许值，则ERR=4。

编程示例：
程序要求：生成如下图所示的线段曲线：

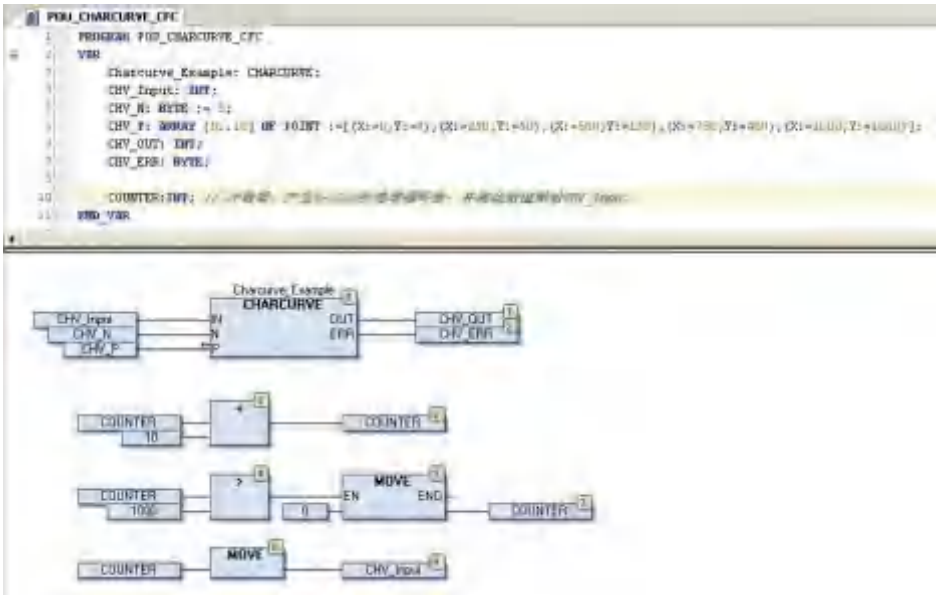


相关CFC程序如下：

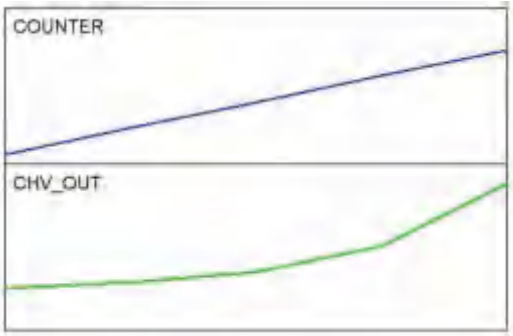
编程示例：
程序要求：生成如下图所示的线段曲线：



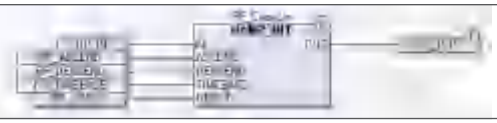
相关CFC程序如下：



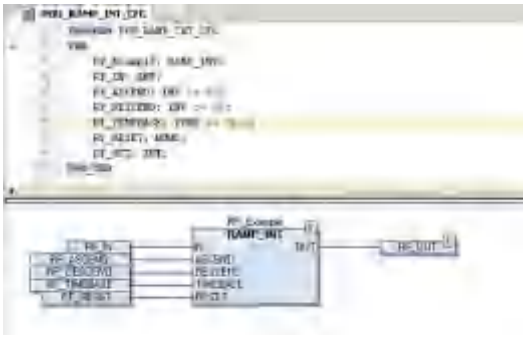
程序中COUNTER与CHV_OUT数值曲线：



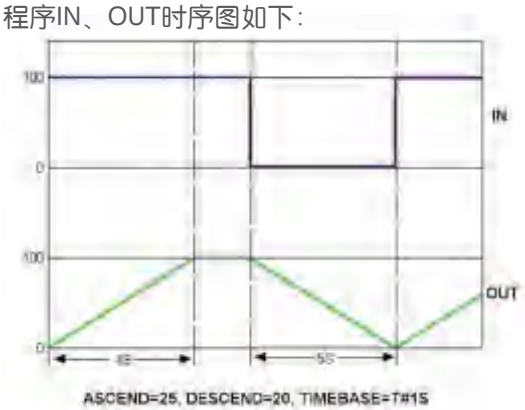
8.3.8函数操纵功能块-RAMP_INT

操作符	RAMP_INT
功能说明	此功能块根据输入IN值的变化，OUT输出斜坡上升或下降函数；输入输出数据类型为INT。
图形	
管脚定义	输入 类型 注释 IN INT 输入数值; ASCEND INT 输出OUT的上升斜坡; DESCEND INT 输出OUT的下降斜坡; TIMEBASE TIME 上升/下降斜坡的时间基准; RESET BOOL 功能块复位，高电平有效; 输出 类型 注释 OUT INT 输出数值。
变量声明	VAR RP_Example: RAMP_INT; RP_IN: INT; RP_ASCEND: INT; RP_DESCEND: INT; RP_TIMEBASE: TIME; RP_RESET: BOOL; RP_OUT: INT; END_VAR
CFC语言示例	
ST语言示例	RP_Example(IN:=RP_IN , ASCEND:=RP_ASCEND , DESCEND:=RP_DESCEND , TIMEBASE:=RP_TIMEBASE , RESET:=RP_RESET , OUT=>RP_OUT);
LD语言示例	
注意事项	① ASCEND/TIMEBASE，即为上升斜率; ② DESCEND/TIMEBASE，即为下降斜率。

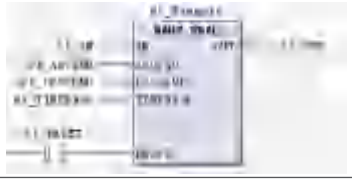
编程示例:



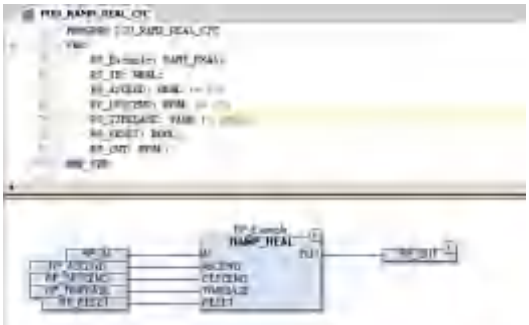
程序IN、OUT时序图如下:



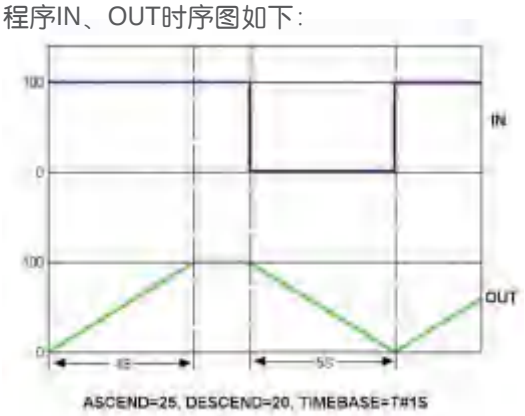
8.3.8函数操纵功能块-RAMP_REAL

操作符	RAMP_REAL
功能说明	此功能块根据输入IN值的变化，OUT输出斜坡上升或下降函数；输入输出数据类型为REAL。
图形	
管脚定义	输入 类型 注释 IN REAL 输入数值； ASCEND REAL 输出OUT的上升斜坡； DESCEND REAL 输出OUT的下降斜坡； TIMEBASE TIME 上升/下降斜坡的时间基准； RESET BOOL 功能块复位，高电平有效； 输出 类型 注释 OUT REAL 输出数值。
变量声明	VAR RP_Example: RAMP_REAL; RP_IN: REAL; RP_ASCEND: REAL; RP_DESCEND: REAL; RP_TIMEBASE: TIME; RP_RESET: BOOL; RP_OUT: REAL; END_VAR
CFC语言示例	
ST语言示例	RP_Example(IN:=RP_IN , ASCEND:=RP_ASCEND , DESCEND:=RP_DESCEND , TIMEBASE:=RP_TIMEBASE , RESET:=RP_RESET , OUT=>RP_OUT);
LD语言示例	
注意事项	① ASCEND/TIMEBASE，即为上升斜率； ② DESCEND/TIMEBASE，即为下降斜率。

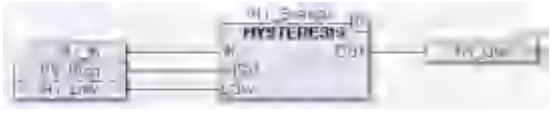
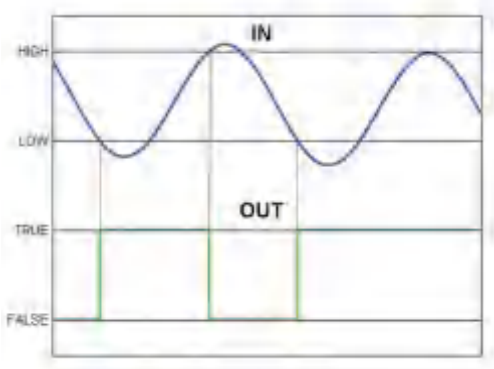
编程示例：



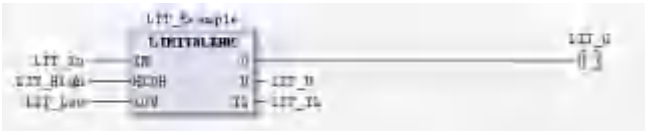
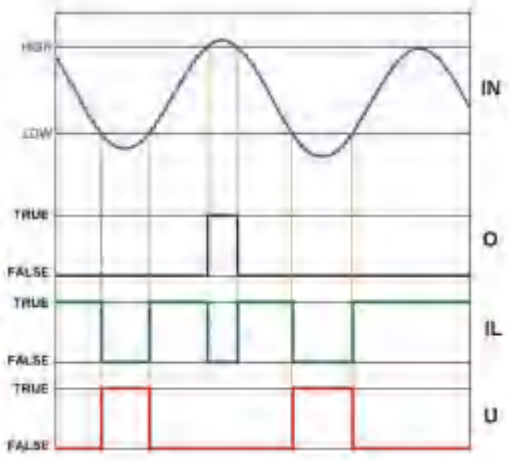
程序IN、OUT时序图如下：



8.3.9模拟量监视功能块-HYSTERESIS

操作符	HYSTERESIS
功能说明	此功能块监视输入模拟量IN的变化，并根据设定的上下限阈值，控制BOOL型输出量OUT的状态。
图形	
管脚定义	输入 类型 注释 IN INT 输入模拟量数值; HIGH INT IN的上限阈值; LOW INT IN的下限阈值; 输出 类型 注释 OUT BOOL 输出量。
变量声明	VAR HY_Example: HYSTERESIS; HY_In: INT; HY_High: INT; HY_Low: INT; HY_Out: BOOL; END_VAR
CFC语言示例	
ST语言示例	HY_Example(IN:=HY_In , HIGH:=HY_High , LOW:=HY_Low , OUT=>HY_Out);
LD语言示例	
注意事项	功能说明: ① 如果IN低于限值LOW，OUT变为TRUE。如果IN高于上限HIGH，OUT变为为FALSE; ② 如果IN数值介于LOW和HIGH之间，OUT输出TRUE或FALSE均有可能；此时OUT状态保持为：IN经过阈值点之前，当时OUT的状态。 可参考下图Hysteresis.IN和Hysteresis.OUT的图解比较: 

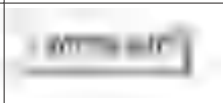
8.3.9模拟量监视功能块-LIMITALARM

操作符	LIMITALARM
功能说明	此功能块根据设定的上下限阈值，输出BOOL型状态标志位，表示输入模拟量IN当前所属范围，是否超出上下限。
图形	
管脚定义	输入 类型 注释 IN INT 输入模拟量数值; HIGH INT IN的上限阈值; LOW INT IN的下限阈值; 输出 类型 注释 O BOOL 当IN>HIGH时，O=TRUE; U BOOL 当IN<LOW时，U=TRUE; IL BOOL 当LOW≤IN≤HIGH时，IL=TRUE;
变量声明	VAR LIT_Example: LIMITALARM; LIT_In: INT; LIT_High: INT; LIT_Low: INT; LIT_O: BOOL; LIT_U: BOOL; LIT_IL: BOOL; END_VAR
CFC语言示例	
ST语言示例	LIT_Example(IN:=LIT_In , HIGH:=LIT_High , LOW:=LIT_Low , O=>LIT_O , U=>LIT_U , IL=>LIT_IL);
LD语言示例	
注意事项	输入IN和输出O/U/IL之间的关系，参考下面时序图: 

8.3.10 数制转换-BYTE_TO_BOOL

操作符	BYTE_TO_BOOL
功能说明	该功能块用作字节型变量转换成布尔型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据； 输出: 布尔型（BOOL），该输出为转换后的布尔型数据； 当输入不为0时，结果为TRUE。当操作数为0时，结果为FALSE。
变量声明	VAR BYTE1: BYTE; BOOL1: BOOL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	如果输入值小于0或者大于99，则返回错误值为255。

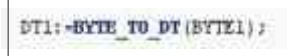
8.3.10 数制转换-BYTE_TO_DATA

操作符	BYTE_TO_DATA
功能说明	该功能块用作字节型数据转换成日期型数据
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据； 输出: 日期型（DATE），该输出为转换后的日期型数据； 输入转换成以秒为单位，PLC内部日期基准为1970年1月1日12:00AM，输出为1970年1月1日再加上转换后的秒数。
变量声明	VAR BYTE1: BYTE; DATE1: DATE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	如果输入值小于0或者大于99，则返回错误值为255。

8.3.10 数制转换-BYTE_TO_DINT

操作符	BYTE_TO_DINT
功能说明	该功能块用作字节型变量转换成双整数型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据； 输出: 双整数型（DINT），该输出为转换后的双整数型数据；
变量声明	VAR BYTE1: BYTE; DINT1: DINT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-BYTE_TO_DT

操作符	BYTE_TO_DT
功能说明	该功能块用作字节型变量转换成日期时间型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据； 输出: 日期时间型（DAY_OF_TIME），该输出为转换后的日期时间型数据； 输入转换成以秒为单位，PLC内部日期基准为1970年1月1日12:00AM，输出为1970年1月1日12:00AM再加上转换后的秒数。
变量声明	VAR BYTE1: BYTE; DT1:DAY_OF_TIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

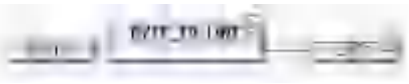
8.3.10 数制转换-BYTE_TO_DWORD

操作符	BYTE_TO_DWORD
功能说明	该功能块用作字节型变量转换成双字型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据; 输出: 双字型（DWORD），该输出为转换后的双字型数据;
变量声明	VAR BYTE1: BYTE; DWORD1： DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

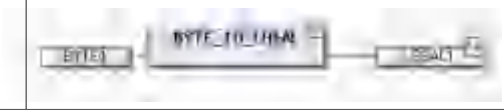
8.3.10 数制转换-BYTE_TO_INT

操作符	BYTE_TO_INT
功能说明	该功能块用作字节型变量转换成整数型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据; 输出: 整数型（INT），该输出为转换后的整数型数据;
变量声明	VAR BYTE1: BYTE; INT1;INT; END_VAR
CFC语言示例	
ST语言示例	INT1:=BYTE_TO_INT(BYTE1);
LD语言示例	
时序图	
使用限制	
注意事项	

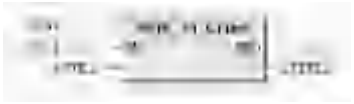
8.3.10 数制转换-BYTE_TO_LINT

操作符	BYTE_TO_LINT
功能说明	该功能块用作字节型变量转换成长整数型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据; 输出: 长整数型（LINT），该输出为转换后的布尔型数据;
变量声明	VAR BYTE1: BYTE; LINT1: LINT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-BYTE_TO_LREAL

操作符	BYTE_TO_LREAL
功能说明	该功能块用作字节型变量转换成长实数型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据； 输出: 长实数型（LREAL），该输出为转换后的长实数型数据；
变量声明	VAR BYTE1: BYTE; LREAL1:LREAL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-BYTE_TO_LTIME

操作符	BYTE_TO_LTIME
功能说明	该功能块用作字节型变量转换成长时间型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据; 输出: 长时间型（LTIME），该输出为转换后的长时间型数据;
变量声明	VAR BYTE1: BYTE; LTIME1:LTIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-BYTE_TO_LWORD

操作符	BYTE_TO_LWORD
功能说明	该功能块用作字节型变量转换成长字型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据; 输出: 长字型（LWORD），该输出为转换后的长字型数据;
变量声明	VAR BYTE1: BYTE; LWORD1:LWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-BYTE_TO_REAL

操作符	BYTE_TO_REAL
功能说明	该功能块用作字节型变量转换成实数型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据; 输出: 实数型（REAL），该输出为转换后的实数型数据;
变量声明	VAR BYTE1: BYTE; REAL1: REAL; END_VAR
CFC语言示例	
ST语言示例	REAL1:=BYTE_TO_REAL(BYTE1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-BYTE_TO_SINT

操作符	BYTE_TO_SINT
功能说明	该功能块用作字节型变量转换成短整数型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据； 输出: 短整数型（SINT），该输出为转换后的短整数型数据；
变量声明	VAR BYTE1: BYTE; SINT1: SINT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-BYTE_TO_STRING

操作符	BYTE_TO_STRING
功能说明	该功能块用作字节型变量转换成字符串型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据; 输出: 字符串型（STRING），该输出为转换后的字符串型数据;
变量声明	VAR BYTE1: BYTE; STRING1:STRING; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

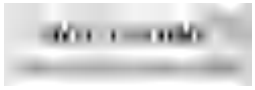
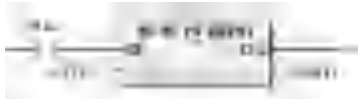
8.3.10 数制转换-BYTE_TO_TIME

操作符	BYTE_TO_TIME
功能说明	该功能块用作字节型变量转换成时间型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据； 输出: 时间型（TIME），该输出为转换后的时间型数据；
变量声明	VAR BYTE1: BYTE; TIME1:TIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

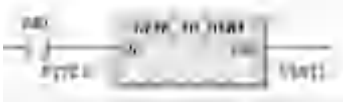
8.3.10 数制转换-BYTE_TO_TOD

操作符	BYTE_TO_TOD
功能说明	该功能块用作字节型变量转换成时间日期型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据; 输出: 时间日期型（TIME_OF_DAY），该输出为转换后的时间日期型数据;
变量声明	VAR BYTE1: BYTE; TOD1:TIME_OF_DAY; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

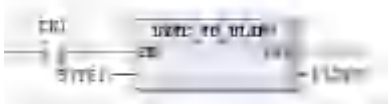
8.3.10 数制转换-BYTE_TO_UDINT

操作符	BYTE_TO_UDINT
功能说明	该功能块用作字节型变量转换成无符号双整数型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据; 输出: 无符号双整数型（UDINT），该输出为转换后的无符号双整数型数据;
变量声明	VAR BYTE1: BYTE; UDINT1:UDINT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-BYTE_TO_UINT

操作符	BYTE_TO_UINT
功能说明	该功能块用作字节型变量转换成无符号整数型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据; 输出: 无符号整数型（UINT），该输出为转换后的无符号整数型数据;
变量声明	VAR BYTE1: BYTE; UDINT1:UDINT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-BYTE_TO_ULINT

操作符	BYTE_TO_ULINT
功能说明	该功能块用作字节型变量转换成无符号长整数型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据; 输出: 无符号双整数型（ULINT），该输出为转换后的无符号长整数型数据;
变量声明	VAR BYTE1: BYTE; ULINT1:ULINT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-BYTE_TO_USINT

操作符	BYTE_TO_USINT
功能说明	该功能块用作字节型变量转换成无符号短整数型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据; 输出: 无符号短整数型（USINT），该输出为转换后的无符号短整数型数据;
变量声明	VAR BYTE1: BYTE; USINT1:USINT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

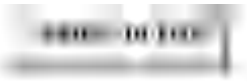
8.3.10 数制转换-BYTE_TO_WORD

操作符	BYTE_TO_WORD
功能说明	该功能块用作字节型变量转换成字型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR BYTE1: BYTE; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

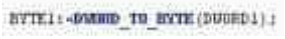
8.3.10 数制转换-BYTE_TO_WSTRING

操作符	BYTE_TO_WSTRING
功能说明	该功能块用作字节型变量转换成双字节字符串型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据； 输出: 双字节字符串型（WSTRING），该输出为转换后的双字节字符串型数据；
变量声明	VAR BYTE1: BYTE; WSTRING1: WSTRING; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DWORD_TO_BOOL

操作符	DWORD_TO_BOOL
功能说明	该功能块用作双字型变量转换成布尔型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据； 输出: 布尔型（BOOL），该输出为转换后的布尔型数据； 当输入不为0时，结果为TRUE。当操作数为0时，结果为FALSE。
变量声明	VAR DWORD1:DWORD; BOOL1: BOOL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DWORD_TO_BYTE

操作符	DWORD_TO_BYTE
功能说明	该功能块用作双字型变量转换成字节型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据; 输出: 字节型（BYTE），该输出为转换后的字节型数据;
变量声明	VAR DWORD1:DWORD; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DWORD_TO_DATA

操作符	DWORD_TO_DATA
功能说明	该功能块用作双字型数据转换成日期型数据
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据； 输出: 日期型（DATE），该输出为转换后的日期型数据； 输入转换成以秒为单位，PLC内部日期基准为1970年1月1日12:00AM，输出为1970年1月1日再加上转换后的秒数。
变量声明	VAR DWORD1:DWORD; DATE1: DATE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

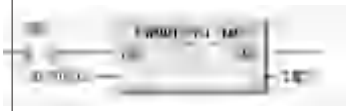
8.3.10 数制转换-DWORD_TO_DINT

操作符	DWORD_TO_DINT
功能说明	该功能块用作双字型变量转换成双整数型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据； 输出: 双整数型（DINT），该输出为转换后的双整数型数据；
变量声明	VAR DWORD1:DWORD; DINT1: DINT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DWORD_TO_DT

操作符	DWORD_TO_DT
功能说明	该功能块用作双字型变量转换成日期时间型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据； 输出: 日期时间型（DAY_OF_TIME），该输出为转换后的日期时间型数据； 输入转换成以秒为单位，PLC内部日期基准为1970年1月1日12:00AM，输出为1970年1月1日12:00AM再加上转换后的秒数。
变量声明	VAR DWORD1:DWORD; DT1:DAY_OF_TIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DWORD_TO_INT

操作符	DWORD_TO_INT
功能说明	该功能块用作双字型变量转换成整数型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据; 输出: 整数型（INT），该输出为转换后的整数型数据;
变量声明	VAR DWORD1:DWORD; INT1;INT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DWORD_TO_LINT

操作符	DWORD_TO_LINT
功能说明	该功能块用作双字型变量转换成长整数型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据; 输出: 长整数型（LINT），该输出为转换后的布尔型数据;
变量声明	VAR DWORD1:DWORD; LINT1: LINT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

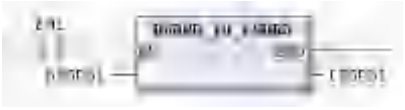
8.3.10 数制转换-DWORD_TO_LREAL

操作符	DWORD_TO_LREAL
功能说明	该功能块用作双字型变量转换成长实数型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据； 输出: 长实数型（LREAL），该输出为转换后的长实数型数据；
变量声明	VAR DWORD1:DWORD; LREAL1:LREAL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

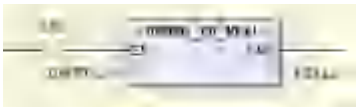
8.3.10 数制转换-DWORD_TO_LTIME

操作符	DWORD_TO_LTIME
功能说明	该功能块用作双字型变量转换成长时间型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据; 输出: 长时间型（LTIME），该输出为转换后的长时间型数据;
变量声明	VAR DWORD1:DWORD; LTIME1:LTIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

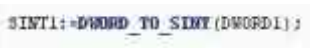
8.3.10 数制转换-DWORD_TO_LWORD

操作符	DWORD_TO_LWORD
功能说明	该功能块用作双字型变量转换成长字型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据; 输出: 长字型（LWORD），该输出为转换后的长字型数据;
变量声明	VAR DWORD1:DWORD; LWORD1:LWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DWORD_TO_REAL

操作符	DWORD_TO_REAL
功能说明	该功能块用作双字型变量转换成实数型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据; 输出: 实数型（REAL），该输出为转换后的实数型数据;
变量声明	VAR DWORD1:DWORD; REAL1: REAL; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

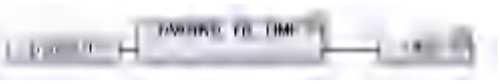
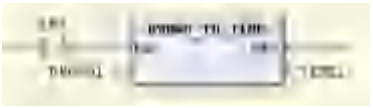
8.3.10 数制转换-DWORD_TO_SINT

操作符	DWORD_TO_SINT
功能说明	该功能块用作双字型变量转换成短整数型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据； 输出: 短整数型（SINT），该输出为转换后的短整数型数据；
变量声明	VAR DWORD1:DWORD; SINT1: SINT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DWORD_TO_STRING

操作符	DWORD_TO_STRING
功能说明	该功能块用作双字型变量转换成字符串型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据; 输出: 字符串型（STRING），该输出为转换后的字符串型数据;
变量声明	VAR DWORD1:DWORD; STRING1:STRING; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DWORD_TO_TIME

操作符	DWORD_TO_TIME
功能说明	该功能块用作双字型变量转换成时间型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据; 输出: 时间型（TIME），该输出为转换后的时间型数据;
变量声明	VAR DWORD1:DWORD; TIME1:TIME; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

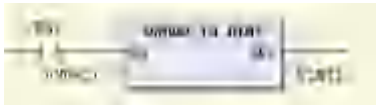
8.3.10 数制转换-DWORD_TO_TOD

操作符	DWORD_TO_TOD
功能说明	该功能块用作双字型变量转换成时间日期型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据； 输出: 时间日期型（TIME_OF_DAY），该输出为转换后的时间日期型数据；
变量声明	VAR DWORD1:DWORD; TOD1:TIME_OF_DAY; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

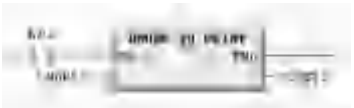
8.3.10 数制转换-DWORD_TO_UDINT

操作符	DWORD_TO_UDINT
功能说明	该功能块用作双字型变量转换成无符号双整数型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据； 输出: 无符号双整数型（UDINT），该输出为转换后的无符号双整数型数据；
变量声明	VAR DWORD1:DWORD; UDINT1:UDINT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DWORD_TO_UINT

操作符	DWORD_TO_UINT
功能说明	该功能块用作双字型变量转换成无符号整数型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据; 输出: 无符号整数型（UINT），该输出为转换后的无符号整数型数据;
变量声明	VAR DWORD1:DWORD; UDINT1:UDINT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

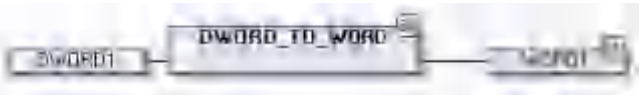
8.3.10 数制转换-DWORD_TO_ULINT

操作符	DWORD_TO_ULINT
功能说明	该功能块用作双字型变量转换成无符号长整数型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据; 输出: 无符号双整数型（ULINT），该输出为转换后的无符号双整数型数据;
变量声明	VAR DWORD1:DWORD; ULINT1:ULINT; END_VAR
CFC语言示例	
ST语言示例	 ULINT1:=DWORD_TO_ULINT(DWORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

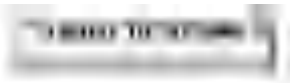
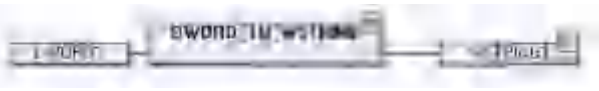
8.3.10 数制转换-DWORD_TO_USINT

操作符	DWORD_TO_USINT
功能说明	该功能块用作双字型变量转换成无符号短整数型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据; 输出: 无符号短整数型（USINT），该输出为转换后的无符号短整数型数据;
变量声明	VAR DWORD1:DWORD; USINT1:USINT; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DWORD_TO_WORD

操作符	DWORD_TO_WORD
功能说明	该功能块用作双字型变量转换成字型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR DWORD1:DWORD; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=DWORD_TO_WORD(DWORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DWORD_TO_WSTRING

操作符	DWORD_TO_WSTRING
功能说明	该功能块用作双字型变量转换成双字节字符串型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据； 输出: 双字节字符串型（WSTRING），该输出为转换后的双字节字符串型数据；
变量声明	VAR DWORD1:DWORD; WSTRING1: WSTRING; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

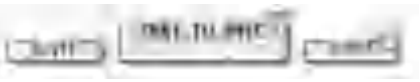
8.3.10 数制转换-BOOL_TO_BYTE

操作符	BOOL_TO_BYTE
功能说明	该功能块用作布尔型变量转换成字节型
图形	
管脚定义	输入: 布尔型（BOOL），该输入为要转换的布尔型数据； 输出: 字节型（BYTE），该输出为转换后的字节型数据；
变量声明	VAR BOOL1: BOOL; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

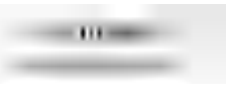
8.3.10 数制转换-DATE_TO_BYTE

操作符	DATE_TO_BYTE
功能说明	该功能块用作日期型变量转换成字节型
图形	
管脚定义	输入: 日期型（DATE），该输入为要转换的日期型数据; 输出: 字节型（BYTE），该输出为转换后的字节型数据;
变量声明	VAR DATE1: DATE; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

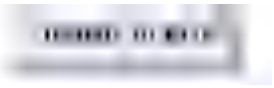
8.3.10 数制转换-DINT_TO_BYTE

操作符	DINT_TO_BYTE
功能说明	该功能块用作双整数型变量转换成字节型
图形	
管脚定义	输入: 双整数型（DINT），该输入为要转换的双整数型数据； 输出: 字节型（BYTE），该输出为转换后的字节型数据；
变量声明	VAR DINT1: DINT; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DT_TO_BYTE

操作符	DT_TO_BYTE
功能说明	该功能块用作日期时间型变量转换成字节型
图形	
管脚定义	输入: 日期时间型（DAY_OF_TIME），该输入为要转换的日期时间型数据； 输出: 字节型（BYTE），该输出为转换后的字节型数据；
变量声明	VAR DT1:DAY_OF_TIME; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DWORD_TO_BYTE

操作符	DWORD_TO_BYTE
功能说明	该功能块用作双字型变量转换成字节型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据; 输出: 字节型（BYTE），该输出为转换后的字节型数据;
变量声明	VAR DWORD1：DWORD; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

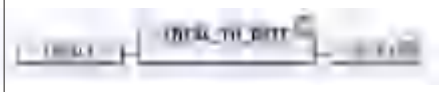
8.3.10 数制转换-INT_TO_BYTE

操作符	INT_TO_BYTE
功能说明	该功能块用作整数型变量转换成字节型
图形	
管脚定义	输入: 整数型（INT），该输入为要转换的整数型数据; 输出: 字节型（BYTE），该输出为转换后的字节型数据;
变量声明	VAR INT1;INT; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

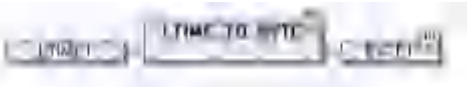
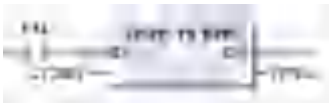
8.3.10 数制转换-LINT_TO_BYTE

操作符	LINT_TO_BYTE
功能说明	该功能块用作长整数型变量转换成字节型
图形	
管脚定义	输入: 长整数型（LINT），该输入为要转换的长整数型数据; 输出: 字节型（BYTE），该输出为转换后的字节型数据;
变量声明	VAR LINT1: LINT; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

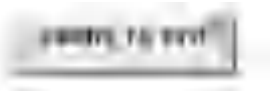
8.3.10 数制转换-LREAL_TO_BYTE

操作符	LREAL_TO_BYTE
功能说明	该功能块用作长实数型变量转换成字节型
图形	
管脚定义	输入: 长实数型（LREAL），该输入为要转换的长实数型数据； 输出: 字节型（BYTE），该输出为转换后的字节型数据；
变量声明	VAR LREAL1:LREAL; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-LTIME_TO_BYTE

操作符	LTIME_TO_BYTE
功能说明	该功能块用作长时间型变量转换成字节型
图形	
管脚定义	输入: 长时间型（LTIME），该输入为要转换的长时间型数据; 输出: 字节型（BYTE），该输出为转换后的字节型数据;
变量声明	VAR LTIME1:LTIME; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

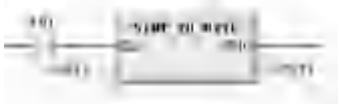
8.3.10 数制转换-LWORD_TO_BYTE

操作符	LWORD_TO_BYTE
功能说明	该功能块用作长字型变量转换成字节型
图形	
管脚定义	输入: 长字型（LWORD），该输入为要转换的长字型数据; 输出: 字节型（BYTE），该输出为转换后的字节型数据;
变量声明	VAR LWORD1:LWORD; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-REAL_TO_BYTE

操作符	REAL_TO_BYTE
功能说明	该功能块用作实数型变量转换成字节型
图形	
管脚定义	输入: 实数型（REAL），该输入为要转换的实数型数据; 输出: 字节型（BYTE），该输出为转换后的字节型数据;
变量声明	VAR REAL1: REAL; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-SINT_TO_BYTE

操作符	SINT_TO_BYTE
功能说明	该功能块用作短整数型变量转换成字节型
图形	
管脚定义	输入: 短整数型（SINT），该输入为要转换的短整数型数据； 输出: 字节型（BYTE），该输出为转换后的字节型数据；
变量声明	VAR SINT1: SINT; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-STRING_TO_BYTE

操作符	STRING_TO_BYTE
功能说明	该功能块用作字符串型变量转换成字节型
图形	
管脚定义	输入: 字符串型（STRING），该输入为要转换的字符串型数据; 输出: 字节型（BYTE），该输出为转换后的字节型数据;
变量声明	VAR STRING1:STRING; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

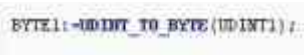
8.3.10 数制转换-TIME_TO_BYTE

操作符	TIME_TO_BYTE
功能说明	该功能块用作时间型变量转换成字节型
图形	
管脚定义	输入: 时间型（TIME），该输入为要转换的时间型数据； 输出: 字节型（BYTE），该输出为转换后的字节型数据；
变量声明	VAR TIME1:TIME; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-TOD_TO_BYTE

操作符	TOD_TO_BYTE
功能说明	该功能块用作时间日期型变量转换成字节型
图形	
管脚定义	输入: 时间日期型（TIME_OF_DATE），该输入为要转换的时间日期型数据； 输出: 字节型（BYTE），该输出为转换后的字节型数据；
变量声明	VAR TOD1:TIME_OF_DAY; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-UDINT_TO_BYTE

操作符	UDINT_TO_BYTE
功能说明	该功能块用作无符号双整数型变量转换成字节型
图形	
管脚定义	输入: 无符号双整数型（UDINT），该输入为要转换的无符号双整数型数据； 输出: 字节型（BYTE），该输出为转换后的字节型数据；
变量声明	VAR UDINT1:UDINT; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-UINT_TO_BYTE

操作符	UINT_TO_BYTE
功能说明	该功能块用作无符号整数型变量转换成字节型
图形	
管脚定义	输入: 无符号整数型（UINT），该输入为要转换的无符号整数型数据; 输出: 字节型（BYTE），该输出为转换后的字节型数据;
变量声明	VAR UINT1：UINT; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-ULINT_TO_BYTE

操作符	ULINT_TO_BYTE
功能说明	该功能块用作无符号长整数型变量转换成字节型
图形	
管脚定义	输入: 无符号长整数型（ULINT），该输入为要转换的无符号长整数型数据; 输出: 字节型（BYTE），该输出为转换后的字节型数据;
变量声明	VAR ULINT1:ULINT; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

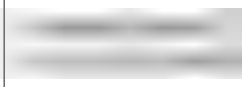
8.3.10 数制转换-USINT_TO_BYTE

操作符	USINT_TO_BYTE
功能说明	该功能块用作无符号短整数型变量转换成字节型
图形	
管脚定义	输入: 无符号短整数型（USINT），该输入为要转换的无符号短整数型数据; 输出: 字节型（BYTE），该输出为转换后的字节型数据;
变量声明	VAR USINT1:USINT; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

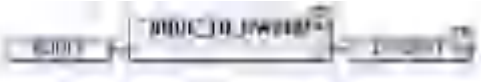
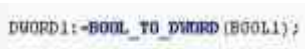
8.3.10 数制转换-WORD_TO_BYTE

操作符	WORD_TO_BYTE
功能说明	该功能块用作字型变量转换成字节型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 字节型（BYTE），该输出为转换后的字节型数据;
变量声明	VAR WORD1: WORD; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-WSTRING_TO_BYTE

操作符	WSTRING_TO_BYTE
功能说明	该功能块用作双字节字符串型变量转换成字节型
图形	
管脚定义	输入: 双字节字符串型（WSTRING），该输入为要转换的双字节字符串型数据; 输出: 字节型（BYTE），该输出为转换后的字节型数据;
变量声明	VAR WSTRING1: WSTRING; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-BOOL_TO_DWORD

操作符	BOOL_TO_DWORD
功能说明	该功能块用作布尔型变量转换成双字型
图形	
管脚定义	输入: 布尔型（BOOL），该输入为要转换的布尔型数据； 输出: 双字型（DWORD），该输出为转换后的双字型数据；
变量声明	VAR BOOL1: BOOL; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-BYTE_TO_DWORD

操作符	BYTE_TO_DWORD
功能说明	该功能块用作字节型变量转换成双字型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据; 输出: 双字型（DWORD），该输出为转换后的双字型数据;
变量声明	VAR BYTE1：BYTE; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

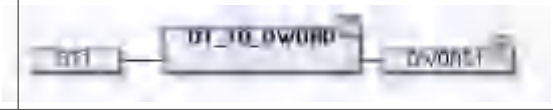
8.3.10 数制转换-DATE_TO_DWORD

操作符	DATE_TO_DWORD
功能说明	该功能块用作日期型变量转换成双字型
图形	
管脚定义	输入: 日期型（DATE），该输入为要转换的日期型数据; 输出: 双字型（DWORD），该输出为转换后的双字型数据;
变量声明	VAR DATE1: DATE; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

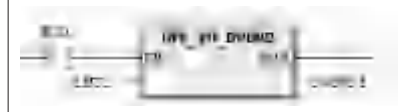
8.3.10 数制转换-DINT_TO_DWORD

操作符	DINT_TO_DWORD
功能说明	该功能块用作双整数型变量转换成双字型
图形	
管脚定义	输入: 双整数型（DINT），该输入为要转换的双整数型数据； 输出: 双字型（DWORD），该输出为转换后的双字型数据；
变量声明	VAR DINT1: DINT; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DT_TO_DWORD

操作符	DT_TO_DWORD
功能说明	该功能块用作日期时间型变量转换成双字型
图形	
管脚定义	输入: 日期时间型（DAY_OF_TIME），该输入为要转换的日期时间型数据； 输出: 双字型（DWORD），该输出为转换后的双字型数据；
变量声明	VAR DT1:DAY_OF_TIME; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

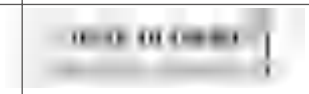
8.3.10 数制转换-INT_TO_DWORD

操作符	INT_TO_DWORD
功能说明	该功能块用作整数型变量转换成双字型
图形	
管脚定义	输入: 整数型（INT），该输入为要转换的整数型数据; 输出: 双字型（DWORD），该输出为转换后的双字型数据;
变量声明	VAR INT1;INT; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

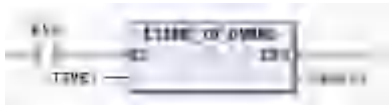
8.3.10 数制转换-LINT_TO_DWORD

操作符	LINT_TO_DWORD
功能说明	该功能块用作长整数型变量转换成双字型
图形	
管脚定义	输入: 长整数型（LINT），该输入为要转换的长整数型数据; 输出: 双字型（DWORD），该输出为转换后的双字型数据;
变量声明	VAR LINT1: LINT; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	DWORD1:=LINT_TO_DWORD(LINT1);
LD语言示例	
时序图	
使用限制	
注意事项	

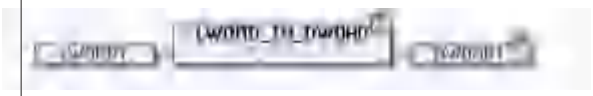
8.3.10 数制转换-LREAL_TO_DWORD

操作符	LREAL_TO_DWORD
功能说明	该功能块用作长实数型变量转换成双字型
图形	
管脚定义	输入: 长实数型（LREAL），该输入为要转换的长实数型数据； 输出: 双字型（DWORD），该输出为转换后的双字型数据；
变量声明	VAR LREAL1:LREAL; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

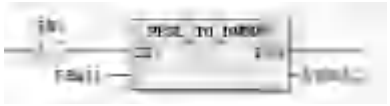
8.3.10 数制转换-LTIME_TO_DWORD

操作符	LTIME_TO_DWORD
功能说明	该功能块用作长时间型变量转换成双字型
图形	
管脚定义	输入: 长时间型（LTIME），该输入为要转换的长时间型数据; 输出: 双字型（DWORD），该输出为转换后的双字型数据;
变量声明	VAR LTIME1:LTIME; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	DWORD1:=LTIME_TO_DWORD(LTIME1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-LWORD_TO_DWORD

操作符	LWORD_TO_DWORD
功能说明	该功能块用作长字型变量转换成双字型
图形	
管脚定义	输入: 长字型（LWORD），该输入为要转换的长字型数据; 输出: 双字型（DWORD），该输出为转换后的双字型数据;
变量声明	VAR LWORD1:LWORD; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-REAL_TO_DWORD

操作符	REAL_TO_DWORD
功能说明	该功能块用作实数型变量转换成双字型
图形	
管脚定义	输入: 实数型（REAL），该输入为要转换的实数型数据; 输出: 双字型（DWORD），该输出为转换后的双字型数据;
变量声明	VAR REAL1: REAL; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-SINT_TO_DWORD

操作符	SINT_TO_DWORD
功能说明	该功能块用作短整数型变量转换成双字型
图形	
管脚定义	输入: 短整数型（SINT），该输入为要转换的短整数型数据; 输出: 双字型（DWORD），该输出为转换后的双字型数据;
变量声明	VAR SINT1: SINT; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	DWORD1:=SINT_TO_DWORD(SINT1);
LD语言示例	
时序图	
使用限制	
注意事项	

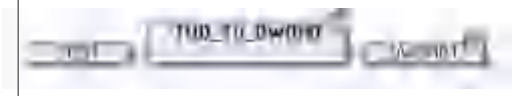
8.3.10 数制转换-STRING_TO_DWORD

操作符	STRING_TO_DWORD
功能说明	该功能块用作字符串型变量转换成双字型
图形	
管脚定义	输入: 字符串型（STRING），该输入为要转换的字符串型数据; 输出: 双字型（DWORD），该输出为转换后的双字型数据;
变量声明	VAR STRING1:STRING; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	DWORD1:=STRING_TO_DWORD(STRING1);
LD语言示例	
时序图	
使用限制	
注意事项	

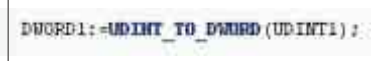
8.3.10 数制转换-TIME_TO_DWORD

操作符	TIME_TO_DWORD
功能说明	该功能块用作时间型变量转换成双字型
图形	
管脚定义	输入: 时间型（TIME），该输入为要转换的时间型数据； 输出: 双字型（DWORD），该输出为转换后的双字型数据；
变量声明	VAR TIME1:TIME; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

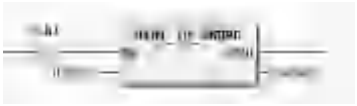
8.3.10 数制转换-TOD_TO_DWORD

操作符	TOD_TO_DWORD
功能说明	该功能块用作时间日期型变量转换成双字型
图形	
管脚定义	输入: 时间日期型（TIME_OF_DATE），该输入为要转换的时间日期型数据； 输出: 双字型（DWORD），该输出为转换后的双字型数据；
变量声明	VAR TOD1:TIME_OF_DAY; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

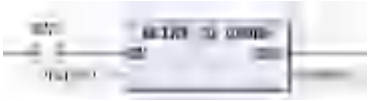
8.3.10 数制转换-UDINT_TO_DWORD

操作符	UDINT_TO_DWORD
功能说明	该功能块用作无符号双整数型变量转换成双字型
图形	
管脚定义	输入: 无符号双整数型（UDINT），该输入为要转换的无符号双整数型数据； 输出: 双字型（DWORD），该输出为转换后的双字型数据；
变量声明	VAR UDINT1:UDINT; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

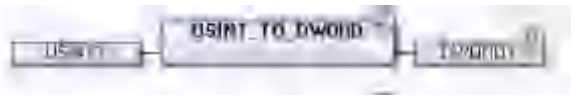
8.3.10 数制转换-UINT_TO_DWORD

操作符	UINT_TO_DWORD
功能说明	该功能块用作无符号整数型变量转换成双字型
图形	
管脚定义	输入: 无符号整数型（UINT），该输入为要转换的无符号整数型数据； 输出: 双字型（DWORD），该输出为转换后的双字型数据；
变量声明	VAR UINT1:UINT; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	DWORD1:=UINT_TO_DWORD(UINT1);
LD语言示例	
时序图	
使用限制	
注意事项	

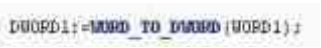
8.3.10 数制转换-ULINT_TO_DWORD

操作符	ULINT_TO_DWORD
功能说明	该功能块用作无符号长整数型变量转换成双字型
图形	
管脚定义	输入: 无符号长整数型（ULINT），该输入为要转换的无符号长整数型数据; 输出: 双字型（DWORD），该输出为转换后的双字型数据;
变量声明	VAR ULINT1:ULINT; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-USINT_TO_DWORD

操作符	USINT_TO_DWORD
功能说明	该功能块用作无符号短整数型变量转换成双字型
图形	
管脚定义	输入: 无符号短整数型（USINT），该输入为要转换的无符号短整数型数据; 输出: 双字型（DWORD），该输出为转换后的双字型数据;
变量声明	VAR USINT1:USINT; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

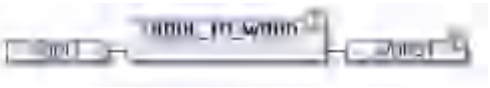
8.3.10 数制转换-WORD_TO_DWORD

操作符	WORD_TO_DWORD
功能说明	该功能块用作字型变量转换成双字型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 双字型（DWORD），该输出为转换后的双字型数据;
变量声明	VAR WORD1: WORD; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-WSTRING_TO_DWORD

操作符	WSTRING_TO_DWORD
功能说明	该功能块用作双字节字符串型变量转换成双字型
图形	
管脚定义	输入: 双字节字符串型（WSTRING），该输入为要转换的双字节字符串型数据； 输出: 双字型（DWORD），该输出为转换后的双字型数据；
变量声明	VAR WSTRING1:WTRING; DWORD1: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-BOOL_TO_WORD

操作符	BOOL_TO_WORD
功能说明	该功能块用作布尔型变量转换成字型
图形	
管脚定义	输入: 布尔型（BOOL），该输入为要转换的布尔型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR BOOL1: BOOL; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=BOOL_TO_WORD(BOOL1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-BYTE_TO_WORD

操作符	BYTE_TO_WORD
功能说明	该功能块用作字节型变量转换成字型
图形	
管脚定义	输入: 字节型（BYTE），该输入为要转换的字节型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR BYTE1: BYTE; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=BYTE_TO_WORD(BYTE1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DATE_TO_WORD

操作符	DATE_TO_WORD
功能说明	该功能块用作日期型变量转换成字型
图形	
管脚定义	输入: 日期型（DATE），该输入为要转换的日期型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR DATE1: DATE; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=DATE_TO_WORD(DATE1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DINT_TO_WORD

操作符	DINT_TO_WORD
功能说明	该功能块用作双整数型变量转换成字型
图形	
管脚定义	输入: 双整数型（DINT），该输入为要转换的双整数型数据； 输出: 字型（WORD），该输出为转换后的字型数据；
变量声明	VAR DINT1: DINT; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=DINT_TO_WORD(DINT1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-DT_TO_WORD

操作符	DT_TO_WORD
功能说明	该功能块用作日期时间型变量转换成字型
图形	
管脚定义	输入: 日期时间型（DAY_OF_TIME），该输入为要转换的日期时间型数据； 输出: 字型（WORD），该输出为转换后的字型数据；
变量声明	VAR DT1:DAY_OF_TIME; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=DT_TO_WORD(DT1);
LD语言示例	
时序图	
使用限制	
注意事项	

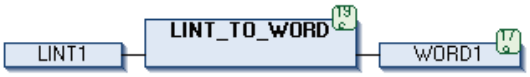
8.3.10 数制转换-DWORD_TO_WORD

操作符	DWORD_TO_WORD
功能说明	该功能块用作双字型变量转换成字型
图形	
管脚定义	输入: 双字型（DWORD），该输入为要转换的双字型数据； 输出: 字型（WORD），该输出为转换后的字型数据；
变量声明	VAR DWORD1:DWORD; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=DWORD_TO_WORD(DWORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

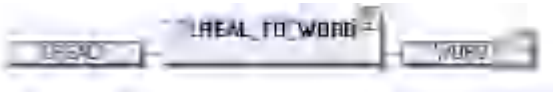
8.3.10 数制转换-INT_TO_WORD

操作符	INT_TO_WORD
功能说明	该功能块用作整数型变量转换成字型
图形	
管脚定义	输入: 整数型（INT），该输入为要转换的整数型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR INT1;INT; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=INT_TO_WORD(INT1);
LD语言示例	
时序图	
使用限制	
注意事项	

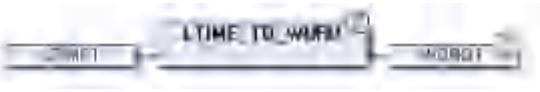
8.3.10 数制转换-LINT_TO_WORD

操作符	LINT_TO_WORD
功能说明	该功能块用作长整数型变量转换成字型
图形	
管脚定义	输入: 长整数型（LINT），该输入为要转换的长整数型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR LINT1: LINT; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=LINT_TO_WORD(LINT1);
LD语言示例	
时序图	
使用限制	
注意事项	

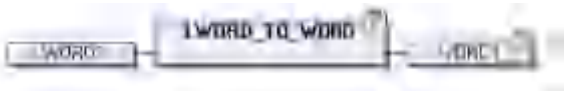
8.3.10 数制转换-LREAL_TO_WORD

操作符	LREAL_TO_WORD
功能说明	该功能块用作长实数型变量转换成字型
图形	
管脚定义	输入: 长实数型（LREAL），该输入为要转换的长实数型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR LREAL1:LREAL; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=LREAL_TO_WORD(LREAL1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-LTIME_TO_WORD

操作符	LTIME_TO_WORD
功能说明	该功能块用作长时间型变量转换成字型
图形	
管脚定义	输入: 长时间型（LTIME），该输入为要转换的长时间型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR LTIME1:LTIME; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=LTIME_TO_WORD(LTIME1);
LD语言示例	
时序图	
使用限制	
注意事项	

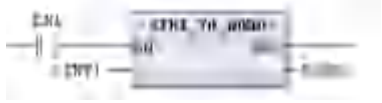
8.3.10 数制转换-LWORD_TO_WORD

操作符	LWORD_TO_WORD
功能说明	该功能块用作长字型变量转换成字型
图形	
管脚定义	输入: 长字型（LWORD），该输入为要转换的长字型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR LWORD1:LWORD; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=LWORD_TO_WORD(LWORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

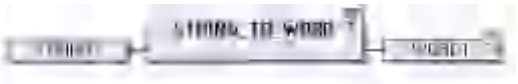
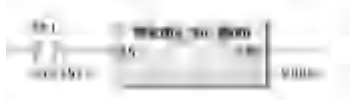
8.3.10 数制转换-REAL_TO_WORD

操作符	REAL_TO_WORD
功能说明	该功能块用作实数型变量转换成字型
图形	
管脚定义	输入: 实数型（REAL），该输入为要转换的实数型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR REAL1: REAL; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=REAL_TO_WORD(REAL1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-SINT_TO_WORD

操作符	SINT_TO_WORD
功能说明	该功能块用作短整数型变量转换成字型
图形	
管脚定义	输入: 短整数型（SINT），该输入为要转换的短整数型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR SINT1: SINT; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=SINT_TO_WORD(SINT1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-STRING_TO_WORD

操作符	STRING_TO_WORD
功能说明	该功能块用作字符串型变量转换成字型
图形	
管脚定义	输入: 字符串型（STRING），该输入为要转换的字符串型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR STRING1:STRING; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=STRING_TO_WORD(STRING1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-TIME_TO_WORD

操作符	TIME_TO_WORD
功能说明	该功能块用作时间型变量转换成字型
图形	
管脚定义	输入: 时间型（TIME），该输入为要转换的时间型数据； 输出: 字型（WORD），该输出为转换后的字型数据；
变量声明	VAR TIME1:TIME; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=TIME_TO_WORD(TIME1);
LD语言示例	
时序图	
使用限制	
注意事项	

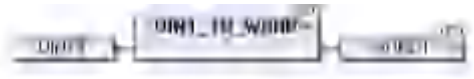
8.3.10 数制转换-TOD_TO_WORD

操作符	TOD_TO_WORD
功能说明	该功能块用作时间日期型变量转换成字型
图形	
管脚定义	输入: 时间日期型（TIME_OF_DATE），该输入为要转换的时间日期型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR TOD1:TIME_OF_DAY; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=TOD_TO_WORD(TOD1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-UDINT_TO_WORD

操作符	UDINT_TO_WORD
功能说明	该功能块用作无符号双整数型变量转换成字型
图形	
管脚定义	输入: 无符号双整数型（UDINT），该输入为要转换的无符号双整数型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR UDINT1:UDINT; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=UDINT_TO_WORD(UDINT1);
LD语言示例	
时序图	
使用限制	
注意事项	

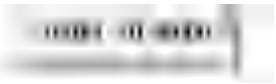
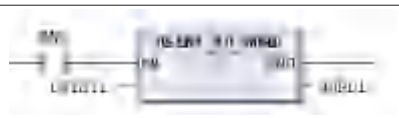
8.3.10 数制转换-UINT_TO_WORD

操作符	UINT_TO_WORD
功能说明	该功能块用作无符号整数型变量转换成字型
图形	
管脚定义	输入: 无符号整数型（UINT），该输入为要转换的无符号整数型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR UDINT1:UDINT; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=UINT_TO_WORD(UINT1);
LD语言示例	
时序图	
使用限制	
注意事项	

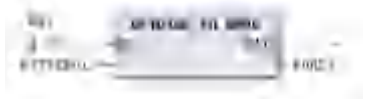
8.3.10 数制转换-ULINT_TO_WORD

操作符	ULINT_TO_WORD
功能说明	该功能块用作无符号长整数型变量转换成字型
图形	
管脚定义	输入: 无符号长整数型（ULINT），该输入为要转换的无符号长整数型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR ULINT1:ULINT; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=ULINT_TO_WORD(ULINT1);
LD语言示例	
时序图	
使用限制	
注意事项	

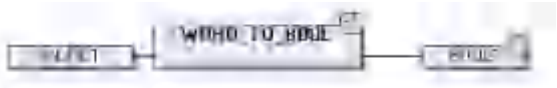
8.3.10 数制转换-USINT_TO_WORD

操作符	USINT_TO_WORD
功能说明	该功能块用作无符号短整数型变量转换成字型
图形	
管脚定义	输入: 无符号短整数型（USINT），该输入为要转换的无符号短整数型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR USINT1:USINT; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	`WORD1:=USINT_TO_WORD(USINT1);`
LD语言示例	
时序图	
使用限制	
注意事项	

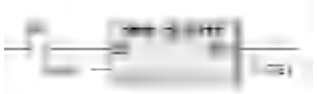
8.3.10 数制转换-WSTRING_TO_WORD

操作符	WSTRING_TO_WORD
功能说明	该功能块用作双字节字符串型变量转换成字型
图形	
管脚定义	输入: 双字节字符串型（WSTRING），该输入为要转换的双字节字符串型数据; 输出: 字型（WORD），该输出为转换后的字型数据;
变量声明	VAR WSTRING1: WSTRING; WORD1: WORD; END_VAR
CFC语言示例	
ST语言示例	WORD1:=WSTRING_TO_WORD(WSTRING1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-WORD_TO_BOOL

操作符	WORD_TO_BOOL
功能说明	该功能块用作字型变量转换成布尔型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 布尔型（BOOL），该输出为转换后的布尔型数据; 当输入不为0时，结果为TRUE。当操作数为0时，结果为FALSE。
变量声明	VAR WORD1:WORD; BOOL1: BOOL; END_VAR
CFC语言示例	
ST语言示例	BOOL1:=WORD_TO_BOOL(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-WORD_TO_BYTE

操作符	WORD_TO_BYTE
功能说明	该功能块用作字型变量转换成字节型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 字节型（BYTE），该输出为转换后的布尔型数据; 当输入不为0时，结果为TRUE。当操作数为0时，结果为FALSE。
变量声明	VAR WORD1:WORD; BYTE1: BYTE; END_VAR
CFC语言示例	
ST语言示例	BYTE1:=WORD_TO_BYTE(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-WORD_TO_DATA

操作符	WORD_TO_DATA
功能说明	该功能块用作字型数据转换成日期型数据
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据； 输出: 日期型（DATE），该输出为转换后的日期型数据； 输入转换成以秒为单位，PLC内部日期基准为1970年1月1日12:00AM，输出为1970年1月1日再加上转换后的秒数。
变量声明	VAR WORD1:WORD; DATE1: DATE; END_VAR
CFC语言示例	
ST语言示例	DATE1:=WORD_TO_DATE(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

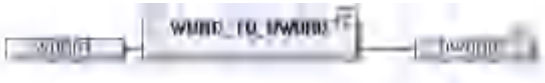
8.3.10 数制转换-WORD_TO_DINT

操作符	WORD_TO_DINT
功能说明	该功能块用作字型变量转换成双整数型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 双整数型（DINT），该输出为转换后的双整数型数据;
变量声明	VAR WORD1:WORD; DINT1: DINT; END_VAR
CFC语言示例	
ST语言示例	DINT1:=WORD_TO_DINT(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-WORD_TO_DT

操作符	WORD_TO_DT
功能说明	该功能块用作字型变量转换成日期时间型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 日期时间型（DAY_OF_TIME），该输出为转换后的日期时间型数据; 输入转换成以秒为单位，PLC内部日期基准为1970年1月1日12:00AM，输出为1970年1月1日12:00AM再加上转换后的秒数。
变量声明	VAR WORD1:WORD; DT1:DAY_OF_TIME; END_VAR
CFC语言示例	
ST语言示例	`DT1:=WORD_TO_DT(WORD1);`
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-WORD_TO_DWORD

操作符	WORD_TO_DWORD
功能说明	该功能块用作字型变量转换成双字型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 双字型（DWORD），该输出为转换后的双字型数据;
变量声明	VAR WORD1:WORD; DWORD1：DWORD; END_VAR
CFC语言示例	
ST语言示例	DWORD1:WORD_TO_DWORD(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-WORD_TO_INT

操作符	WORD_TO_INT
功能说明	该功能块用作字型变量转换成整数型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 整数型（INT），该输出为转换后的整数型数据;
变量声明	VAR WORD1:WORD; INT1;INT; END_VAR
CFC语言示例	
ST语言示例	INT1:=WORD_TO_INT(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-WORD_TO_LINT

操作符	WORD_TO_LINT
功能说明	该功能块用作字型变量转换成长整数型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 长整数型（LINT），输出为转换后的长整型数据;
变量声明	VAR WORD1:WORD; LINT1: LINT; END_VAR
CFC语言示例	
ST语言示例	LINT1:=WORD_TO_LINT(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

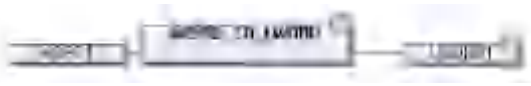
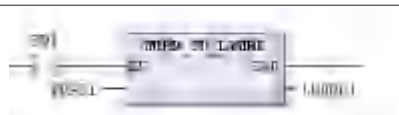
8.3.10 数制转换-WORD_TO_LREAL

操作符	WORD_TO_LREAL
功能说明	该功能块用作字型变量转换成长实数型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 长实数型（LREAL），该输出为转换后的长实数型数据;
变量声明	VAR WORD1:WORD; LREAL1:LREAL; END_VAR
CFC语言示例	
ST语言示例	LREAL1:=WORD_TO_LREAL(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-WORD_TO_LTIME

操作符	WORD_TO_LTIME
功能说明	该功能块用作字型变量转换成长时间型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 长时间型（LTIME），该输出为转换后的长时间型数据;
变量声明	VAR WORD1:WORD; LTIME1:LTIME; END_VAR
CFC语言示例	
ST语言示例	LTIME1:=WORD_TO_LTIME(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

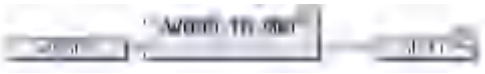
8.3.10 数制转换-WORD_TO_LWORD

操作符	WORD_TO_LWORD
功能说明	该功能块用作字型变量转换成长字型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 长字型（LWORD），该输出为转换后的长字型数据;
变量声明	VAR WORD1:WORD; LWORD1:LWORD; END_VAR
CFC语言示例	
ST语言示例	LWORD1:=WORD_TO_LWORD(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-WORD_TO_REAL

操作符	WORD_TO_REAL
功能说明	该功能块用作字型变量转换成实数型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 实数型（REAL），该输出为转换后的实数型数据;
变量声明	VAR WORD1:WORD; REAL1: REAL; END_VAR
CFC语言示例	
ST语言示例	REAL1:=WORD_TO_REAL(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-WORD_TO_SINT

操作符	WORD_TO_SINT
功能说明	该功能块用作字型变量转换成短整数型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 短整数型（SINT），该输出为转换后的短整数型数据;
变量声明	VAR WORD1:WORD; SINT1: SINT; END_VAR
CFC语言示例	
ST语言示例	SINT1:=WORD_TO_SINT(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

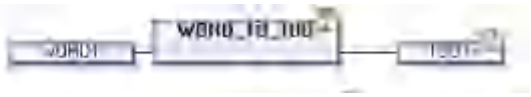
8.3.10 数制转换-WORD_TO_STRING

操作符	WORD_TO_STRING
功能说明	该功能块用作字型变量转换成字符串型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 字符串型（STRING），该输出为转换后的字符串型数据;
变量声明	VAR WORD1:WORD; STRING1:STRING; END_VAR
CFC语言示例	
ST语言示例	STRING1:=WORD_TO_STRING(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

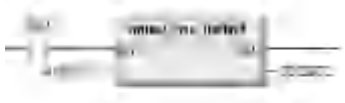
8.3.10 数制转换-WORD_TO_TIME

操作符	WORD_TO_TIME
功能说明	该功能块用作字型变量转换成时间型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据； 输出: 时间型（TIME），该输出为转换后的时间型数据；
变量声明	VAR WORD1:WORD; TIME1:TIME; END_VAR
CFC语言示例	
ST语言示例	TIME1:=WORD_TO_TIME(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-WORD_TO_TOD

操作符	WORD_TO_TOD
功能说明	该功能块用作字型变量转换成时间日期型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 时间日期型（TIME_OF_DAY），该输出为转换后的时间日期型数据;
变量声明	VAR WORD1:WORD; TOD1:TIME_OF_DAY; END_VAR
CFC语言示例	
ST语言示例	TOD1:=WORD_TO_TOD(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-WORD_TO_UDINT

操作符	WORD_TO_UDINT
功能说明	该功能块用作字型变量转换成无符号双整数型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 无符号双整数型（UDINT），该输出为转换后的无符号双整数型数据;
变量声明	VAR WORD1:WORD; UDINT1:UDINT; END_VAR
CFC语言示例	
ST语言示例	UDINT1:=WORD_TO_UDINT(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

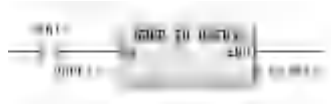
8.3.10 数制转换-WORD_TO_UINT

操作符	WORD_TO_UINT
功能说明	该功能块用作字型变量转换成无符号整数型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 无符号整数型（UINT），该输出为转换后的无符号整数型数据;
变量声明	VAR WORD1:WORD; UDINT1:UDINT; END_VAR
CFC语言示例	
ST语言示例	UINT1:=WORD_TO_UINT(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

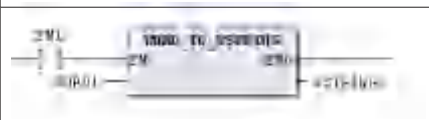
8.3.10 数制转换-WORD_TO_ULINT

操作符	WORD_TO_ULINT
功能说明	该功能块用作字型变量转换成无符号长整数型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 无符号双整数型（ULINT），该输出为转换后的无符号长整数型数据;
变量声明	VAR WORD1:WORD; ULINT1:ULINT; END_VAR
CFC语言示例	
ST语言示例	ULINT1:=WORD_TO_ULINT(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-WORD_TO_USINT

操作符	WORD_TO_USINT
功能说明	该功能块用作字型变量转换成无符号短整数型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 无符号短整数型（USINT），该输出为转换后的无符号短整数型数据;
变量声明	VAR WORD1:WORD; USINT1:USINT; END_VAR
CFC语言示例	
ST语言示例	USINT1:=WORD_TO_USINT(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.3.10 数制转换-WORD_TO_WSTRING

操作符	WORD_TO_WSTRING
功能说明	该功能块用作字型变量转换成双字节字符串型
图形	
管脚定义	输入: 字型（WORD），该输入为要转换的字型数据; 输出: 双字节字符串型（WSTRING），该输出为转换后的双字节字符串型数据;
变量声明	VAR WORD1:WORD; WSTRING1: WSTRING; END_VAR
CFC语言示例	
ST语言示例	WSTRING1:=WORD_TO_WSTRING(WORD1);
LD语言示例	
时序图	
使用限制	
注意事项	

8.4.1 ADDM

说明: 将字符串地址转换为ADDRESS类型的目标地址

操作符

ADDM

功能描述

将字符串地址转换为ADDRESS类型的目标地址

图形表示形式

管脚定义

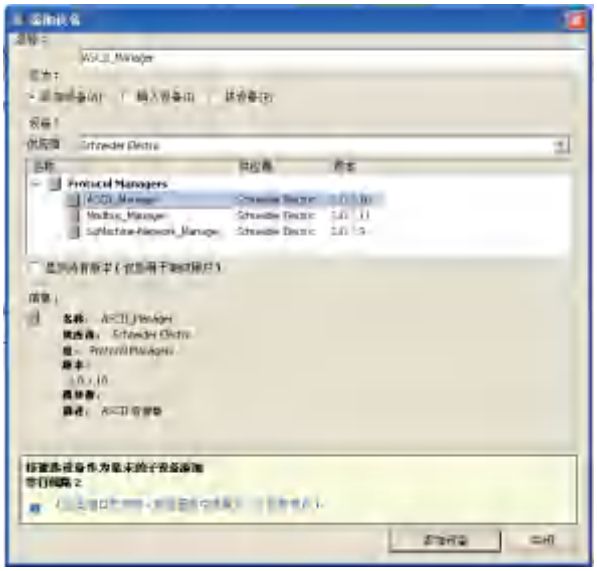
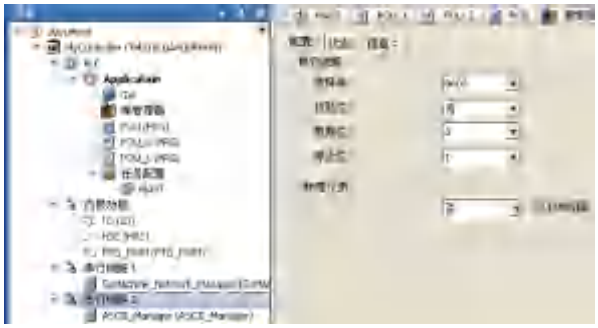
输入	类型	注释
AddrTable	ADDRESS	这是由功能块填充的 ADDRESS 结构
Execute	BOOL	启动引脚,上升沿触发(注意:如果在冷复位、热复位的第一个任务运行周期中将Excute置位True,则检测不到上升沿)
Addr	STRING	要转换为 ADDRESS 类型的 STRING 类型地址(参见下面的详细信息)

输出	类型	注释
Done	BOOL	功能成功完成后，Done 设置为 TRUE
Error	BOOL	功能块出错标志位，终止正在执行的操作。
CommError	BYTE	CommError为通讯错误代码(参见下面的详细信息)

ASCII码串行通讯

(1)系统配置方式

步骤	操作
1	鼠标右键选择要配置的串型线路，选择添加设备 

步骤	操作
2	选择添加设备后，弹出添加设备对话框，选择ASCII-Manager驱动，点击添加设备 
3	双击“设备”导航窗口内串行线路(示例是串口2)，弹出配置界面  配置物理设置：波特率，校验位，数据位，停止位，极化电阻器
4	双击“设备” 导航串口内的ASCII_Manager(ASCII_Manager),弹出协议配置界面  针对ASCII通讯的协议配置参数，请参阅下面的详细介绍
5	做完以上几步后，ASCII串行通讯的配置就完成了

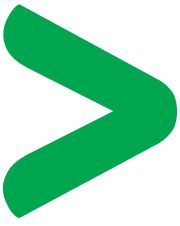
(2)ASCII管理器的配置参数介绍

参数	描述
起始字符	如果为 0，则帧中不使用起始字符。否则，将在接收模式下使用相应的 ASCII 字符以检测帧的开头。在发送模式下，在帧的开始位置添加此字符。
第一个结束字符	如果为 0，则帧中不使用第一个结束字符。否则，将在接收模式下使用相应的 ASCII 字符以检测帧的结尾。在发送模式下，在帧的结束位置添加此字符。
第二个结束字符	如果为 0，则帧中不使用第二个结束字符。否则，将在接收模式下使用相应的 ASCII 字符以检测帧的结尾。在发送模式下，在帧的结束位置添加此字符。
收到的帧长度	如果为 0，则不使用此参数。此参数使系统在控制器收到指定字符数后，在接收时关闭帧结尾。注：此参数不能与帧收到超时（毫秒）同时使用。
帧收到超时(毫秒)	如果为 0，则不使用此参数。此参数使系统经过指定的无收发时间（毫秒）后，在接收时关闭帧结尾。
串行参数设置	请参阅上面的ASCII系统配置方式

(2)Ascii地址格式的Addr String
对于 ASCII 寻址，只请求通讯端口号：'<communication port number>'
例如，要在串行线路 2 上发送用户定义的消息，请使用字符串 '2'。
下表定义了 ASCII 地址格式的 ADDM 输出中的字段：

字段	类型	值	示例
_TYPE	BYTE	保留	未使用
_ClilD	BYTE	保留	未使用
Rack	BYTE	机架编号(始终为 0)	0
Module	BYTE	模块编号(始终为 0)	0
Link	LinkNumber	<communication port number>	2
_PortId	BYTE	未使用	未使用
AddrLen	BYTE	0	0
UnitId	BYTE	未使用	未使用
AddrExt	ADDR_EXT	未使用	未使用

(3)Modbus串行地址格式的Addr String
对于 Modbus 串行寻址，请使用通讯端口和目标从站地址(0 到 247)，之间用句点分隔：
'<communication port number>.<slave address>'
例如，使用以下语法通过串行端口 2 上向从站 1发送消息： '2.1'
ADDM 功能使用以下这些值填充 AddrTable 输入/输出：



字段	类型	值	示例
_TYPE	BYTE	保留	未使用
_ClilD	BYTE	保留	未使用
Rack	BYTE	机架编号(始终为 0)	0
Module	BYTE	模块编号(始终为 0)	0
Link	LinkNumber	<communication port number>	2
_PortId	BYTE	0(对于Modbus)	未使用
AddrLen	BYTE	1	1
UnitId	BYTE	<Slave Address>	1
AddrExt	ADDR_EXT	未使用	未使用

(4)Modbus TCP地址格式的Addr String

- Modbus TCP 标准从站的地址

对于 Modbus TCP 标准从站地址格式，会请求通讯端口号(对于嵌入式以太网端口为 3)和目标 IP 地址 {A.B.C.D}(用花括号括起)：

'<communication port number>{<IP address A.B.C.D>}'

注意： Modbus TCP 标准从站使用 Modbus 地址 255(UnitId 默认值)。但是，Modbus TCP 设备的值可能不同(例如，Tesys 具有 Modbus 地址 1)。在这种情况下，请添加 UnitId 值。默认情况下使用 TCP 端口 502。也可以使用非标准端口，方法是将请求的端口号添加到 IP 地址：

'<communication port number A.B.C.D>{<IP address A.B.C.D>:<port>}'

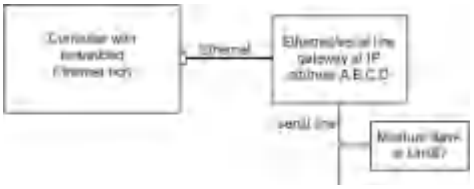
例如，要使用标准 TCP 端口 502 在 Modbus TCP 从站 IP 地址 192.168.1.2 上发送消息，请使用以下字符串： '3{192.168.1.2}'

ADDM 功能使用以下这些值填充 AddrTable 输入/输出：

字段	类型	值	示例
_TYPE	BYTE	保留	未使用
_ClilD	BYTE	保留	未使用
Rack	BYTE	机架编号(始终为 0)	0
Module	BYTE	模块编号(始终为 0)	0
Link	LinkNumber	<communication port number>	3
_PortId	BYTE	0(对于Modbus)	未使用
AddrLen	BYTE	UnitID + AdrExt 长度(以字节为单位)	7
UnitId	BYTE	Modbus 地址(默认为 255)	255
AddrExt	TCP_ADDR_EXT	A	192
		B	168
		C	1
		D	2
		<端口>(默认值=502)	502

- Modbus TCP 标准主站对从站进行寻址

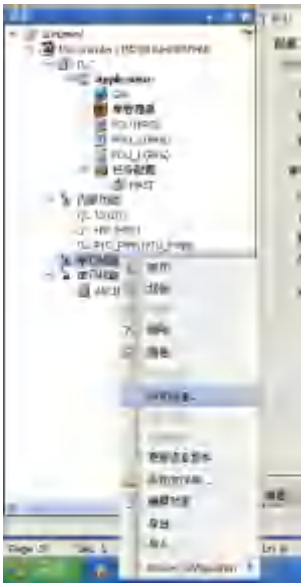
也可以通过以太网/串行线路网关对 Modbus 从站进行寻址：

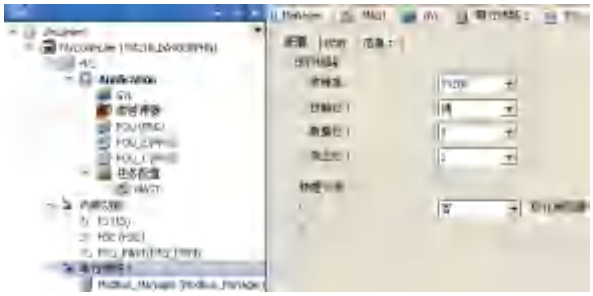


请求包括通讯端口号、网关 IP 地址 {A.B.C.D}(用花括号括起，包括或不包括 TCP 端口)以及 Modbus 串行从站地址(UnitId 参数)：
'<communication port number>{<IP address A.B.C.D>}<slave address>'
例如，要通过以太网/串行线路网关在 Modbus 串行从站地址 5(IP 地址为 192.168.1.2)处，使用标准 TCP 端口 502 发送消息，请使用以下字符串： '3{192.168.1.2}5'
ADDM 功能使用以下这些值填充 AddrTable 输入/输出：

字段	类型	值	示例
_TYPE	BYTE	保留	未使用
_CliID	BYTE	保留	未使用
Rack	BYTE	机架编号(始终为 0)	0
Module	BYTE	模块编号(始终为 0)	0
Link	LinkNumber	<communication port number>	3
_PortId	BYTE	0(对于Modbus)	0
AddrLen	BYTE	UnitID + AdrExt 长度(以字节为单位)	7
UnitId	BYTE	<从站地址>	5
AddrExt	TCP_ADDR_EXT	A	192
		B	168
		C	1
		D	2
		<端口>(默认值=502)	502

Modbus串行通讯
(1)系统配置方式

步骤	操作
1	鼠标右键选择要配置的串型线路，选择添加设备 

步骤	操作
2	选择添加设备后，弹出添加设备对话框，选择Modbus-Manager驱动，点击添加设备 
3	双击“设备”导航窗口内串行线路(示例是串口1)，弹出配置界面  配置物理设置：波特率，校验位，数据位，停止位，极化电阻器
4	双击“设备” 导航串口内的Modbus_Manager(Modbus_Manager),弹出协议配置界面  针对Modbus串行通讯的协议配置参数，请参阅下面的详细介绍
5	根据用户的需求配置Modbus的传输模式是RTU或者ASCII，寻址方式是主站还是从站，如果配置的是从站则需要输入从站的地址，从站地址的范围是1-247.最后配置帧间时间。帧间时间是指主站从接收上一个响应数据帧(或者超时)到下一个请求数据帧之间，应该等待的时间间隔(单位：毫秒)。这个参数可以用来调节传输速率。

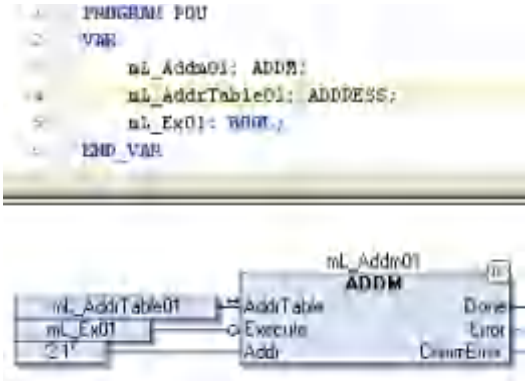
变量声明

```
VAR
mL_Addm01: ADDM;
mL_AddrTable01: ADDRESS;
mL_Ex01: BOOL;
END_VAR
```

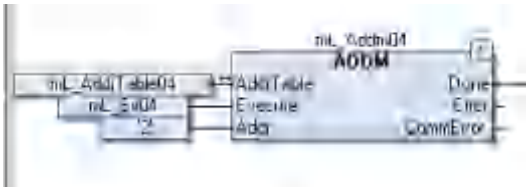
编程示例

每种语言有两个示例，一个是串行口2配置为Modbus_Manager驱动，作为Modbus主站的Addm功能块上的使用示例；另一个是串行口2配置为Ascii_Mananger驱动，作为自由口通讯的Addm功能块上的使用示例

CFC语言示例



Modbus配置下的ADDM功能块使用



Ascii配置下的ADDM功能块使用

ST语言示例

```
mL_Addm01(
    Execute:= NOT mL_Ex01,
    Addr:= '2.1',
    AddrTable:= mL_AddrTable01,
    Done=> ,
    Error=> ,
    CommError=> );
```

Modbus配置下的ADDM功能块使用

```
mL_Addm04(
    Execute:= mL_Ex04,
    Addr:= '2',
    AddrTable:= mL_AddrTable04,
    Done=> ,
    Error=> ,
    CommError=> );
```

Ascii配置下的ADDM功能块使用

LD语言示例

Modbus配置下的ADDM功能块使用

Ascii配置下的ADDM功能块使用

注意事项

在ASCII配置界面中，如果配置了多个帧的结束字符，则当收到第一个结束字符的时候，接收立即停止。

8.4.2 Read_Var
说明: 从Modbus从站设备读取数据
操作符
Read_Var
功能描述
读取Modbus从站设备数据
图形表示形式

管脚定义

输入	类型	注释
Excute	BOOL	启动引脚,上升沿触发(注意:如果在冷复位、热复位的第一个任务运行周期中将Excute置位True,则检测不到上升沿)
Abort	BOOL	终止正在执行的操作
Addr	ADDRESS	目标设备的地址(ADDM功能块的输出)
Timeout	WORD	超时时间为100ms的倍数(0表示无限)
ObjType	ObjectType	接收数据类型(0-MW保持寄存器,1-I数字量输入,2-Q内部位或数字量输出,3-IW输入寄存器)(详见下表)
FirstObj	DINT	FirstObj为要写入的第一个对象的索引。
Quantity	UINT	读取对象的数量 ● 1-125寄存器(MW，IW类型) ●1-2000位(I,Q类型)
Buffer	POINTER TO BYTE	Buffer为要发送数据的缓冲区地址。用户可以通过ADR标准功能指令获取缓冲区的地址(详见下面示例)。缓冲区是一个表，用户要发送的数据写入缓冲区中。例如，4个寄存器的写入值存储在包含4个字的表中，而32位的写入值则需要包含2个字或4个字节的的表中，其中每个位都设置为对应值。

输出	类型	注释
Done	BOOL	功能块完成后Done设置为True
Busy	BOOL	功能块在执行，Busy引脚为True
Aborted	BOOL	当Abort输入终止操作后，终止操作完成Aborted设置为True
Error	BOOL	当功能块检测到错误的时候，Error引脚设置为True，同时CommError和OperError包含相应的错误信息
CommError	BYTE	通讯错误代码(详见下表)
OperError	DWORD	操作错误代码(详见下表)

• ObjectType 枚举数据类型包括可用于读取和/或写入的对象类型。

			读取/写入功能和相关联的 Modbus 请求功能代码			
枚举器	值(十六进制)	对象类型	READ_VAR	WRITE_VAR	SINGLE_WARITE	WRITE_READ_VAR
MW	00	保持寄存器(16 位)	#3(读取保持寄存器)	#16(写入多个寄存器)	#6(写入单个寄存器)	#23(写入读取多个寄存器)
I	01	数字量输入(1 位)	#2(读取数字量输入)			
Q	02	内部位或数字量输出(线圈)(1 位)	#1(读取线圈)	#15(写入多个线圈)		
IW	03	输入寄存器(16 位)	#4(读取输入寄存器)			

• CommError通讯错误代码

枚举器	值(16进制)	描述
CommunicationOK	00	通讯OK
TimeOut	01	功能块在超时时间到期时停止
Canceled	02	功能块通过用户请求(Abort命令)停止
BadAddress	03	地址格式不正确
BadRemoteAddr	04	远程地址不正确
BadMgtTable	05	管理表格式不正确
BadParameters	06	特定参数不正确
ProblemSendingRq	07	向目标发送请求时出现问题
RecvBufferTooSmall	09	接受缓冲区大小太小
SendBufferTooSmall	0A	发送缓冲却大小太小
SystemRessourceMissing	0B	缺少系统资源
BadTransactionNb	0C	事务编号不正确
BadLength	0E	长度不正确
ProtocolSpecificError	FE	操作错误代码中包含特定于协议的代码
Refused	FF	消息被拒绝

• OperError操作错误代码
当CommError通讯错误代码为十六进制的00时,OperError操作错误代码可能返回以下这些值:

枚举器	值(16进制)	描述
OperationOK	00	通讯OK
NotProcessed_or_TargetResourceMissing	01	尚未处理请求
BadResponse	02	收到的响应不正确

当CommError通讯错误代码为十六进制的FF时,OperError操作错误代码可能返回以下这些值:

枚举器	值(16进制)	描述
NotProcessed_or_TargetResourceMissing	01	目标系统的资源不存在
BadLength	05	长度不正确
CommChannelErr	06	通讯通道与检测到的错误关联
BadAddr	07	地址不正确
SystemResourceMissing	0B	缺少系统资源
TargetCommInactive	0C	目标通讯功能未处于活动状态
TargetMissing	0D	目标不存在
ChannelNotConfigured	0F	通道未配置

当CommError通讯错误代码为十六进制的FE时,OperError操作错误代码包含特定于协议的错误检测代码。

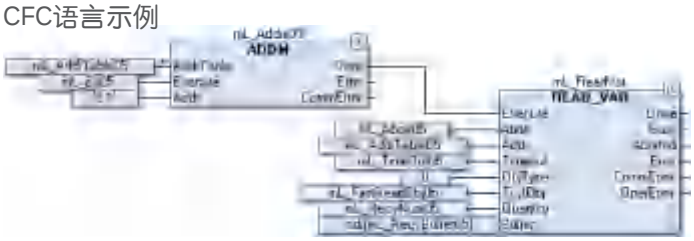
变量声明

```
VAR
mL_Addm05: ADDM;
mL_AddrTable05: ADDRESS;
mL_Ex05: BOOL;
mL_ReadVar: Read_Var;
mL_Abort05: BOOL;
mL_TimeOut05: WORD := 2;
mL_FirstReadObj05: DINT := 8502;
mL_RecvNum05: UINT := 1;
mL_RecvBuffer: INT;
END_VAR
```

编程示例

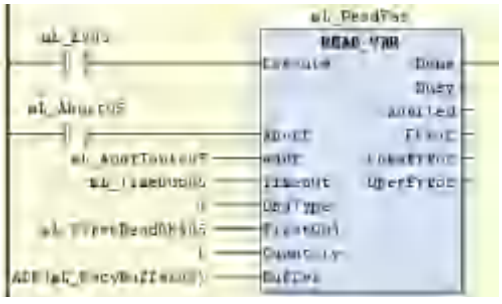
此段程序是通过M218串口2读取从站站号为1的Modbus从站内部寄存器，内部寄存器的地址是8502。

CFC语言示例



ST语言示例

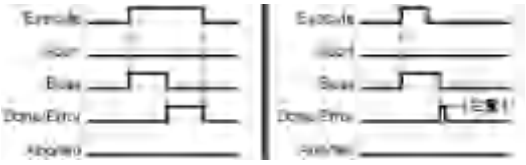
LD语言示例



时序图

- 功能执行

功能在 Execute 输入的上升沿启动。Busy 输出随后会设置为 TRUE。此图显示操作自动完成后(无论是否检测到错误)功能块的行为：



注意1：仅当 Execute 在操作结束后复位为 FALSE 后，Done 或 Error 位在功能块执行完成后，设置为TRUE一个任务循环时间。

- 功能中止

此图显示由应用程序中止的功能。Abort 输入的上升沿取消正在执行的功能。在这种情况下，中止的输出会设置为 1，并且 CommError 包含代码 Canceled - 16#02(由用户请求停止交换)：

注意1：仅当 Execute 在发生中止请求后复位为 FALSE，Abort 位在功能块执行完成后，设置为TRUE一个任务循环时间。

8.4.3 Write_Var
说明: 向Modbus从站设备写入数据
操作符
Write_Var
功能描述
Write_Var功能块将用户数据写入到用户指定的Modbus从站的设备中。
图形表示形式

管脚定义

输入	类型	注释
Excute	BOOL	启动引脚,上升沿触发(注意:如果在冷复位、热复位的第一个任务运行周期中将Excute置位True,则检测不到上升沿)
Abort	BOOL	终止正在执行的操作
Addr	ADDRESS	目标设备的地址(ADDM功能块的输出)
Timeout	WORD	超时时间为100ms的倍数(0表示无限)
ObjType	ObjectType	发送数据类型(0-MW保持寄存器,16Bit;1-I数字量输入,1Bit;2-Q数字量输出,1Bit;3-IW输入寄存器,16Bit)
FirstObj	DINT	FirstObj为要写入的第一个对象的索引。
Quantity	UINT	写入对象的数量 • 1-123寄存器(MW类型) • 1-1968位(Q类型)
Buffer	POINTER TO BYTE	Buffer为要发送数据的缓冲区地址。用户可以通过ADR标准功能指令获取缓冲区的地址(详见下面示例)。缓冲区是一个表，用户要发送的数据写入缓冲区中。例如，4个寄存器的写入值存储在包含4个字的表中，而32位的写入值则需要包含2个字或4个字节的的表中，其中每个位都设置为对应值。

输出	类型	注释
Done	BOOL	功能块完成后Done设置为True
Busy	BOOL	功能块在执行，Busy引脚为True
Aborted	BOOL	当Abort输入终止操作后，终止操作完成Aborted设置为True
Error	BOOL	当功能块检测到错误的时候，Error引脚设置为True，同时CommError和OperError包含相应的错误信息
CommError	BYTE	通讯错误代码(详见Read_Var中介绍)
OperError	DWORD	操作错误代码(详见Read_Var中介绍)

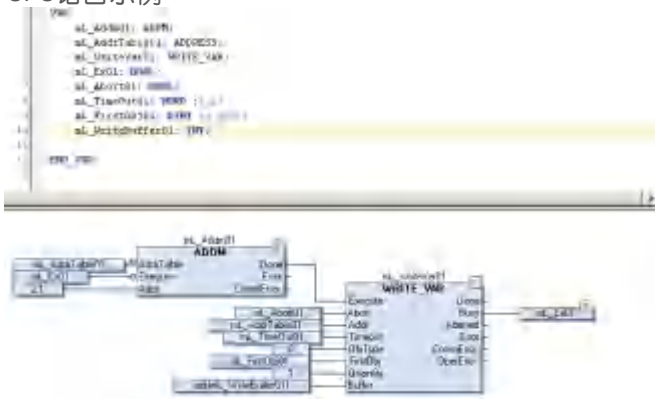
变量声明

```
VAR
mL_Addm01: ADDM;
mL_AddrTable01: ADDRESS;
mL_WriteVar01: WRITE_VAR;
mL_Ex01: BOOL;
mL_Abort01: BOOL;
mL_TimeOut01: WORD := 2;
mL_FirstObj01: DINT := 8502;
mL_WriteBuffer01: POINTER TO int;
END_VAR
```

编程示例

此段程序是通过M218的串口2端口向站号为1的Modbus从站设备的内部8502寄存器写入数据。

CFC语言示例

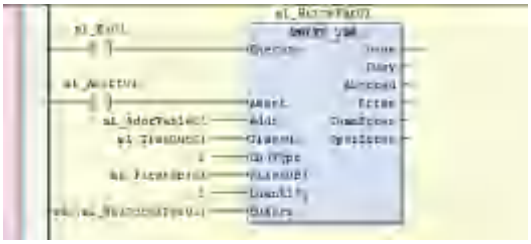


ST语言示例

```
mL_Addm01(
    Execute:= NOT mL_Ex01,
    Addr:= 1,
    AddrTable:= mL_AddrTable01,
    Done=> ,
    Error=> ,
    CommError=> );

mL_WriteVar01(
    Execute:= mL_Addm01.Done,
    Abort:= mL_Abort01,
    Addr:= mL_AddrTable01,
    Timeout:= mL_TimeOut01,
    ObjType:= 0,
    FirstObj:= mL_FirstObj01,
    Quantity:= 1,
    Buffer:= ADR(mL_WriteBuffer01),
    Done=> mL_Ex01,
    Busy=> ,
    Aborted=> ,
    Error=> ,
    CommError=> ,
    OperError=> );
```

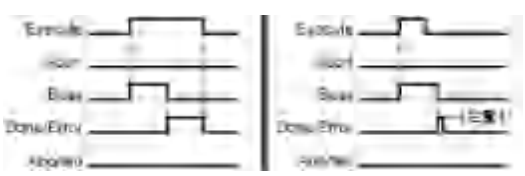
LD语言示例



时序图

- 功能执行

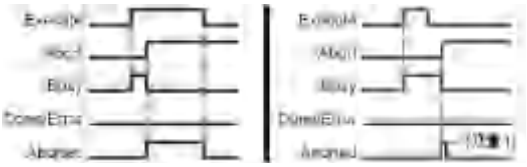
功能在 Execute 输入的上升沿启动。Busy 输出随后会设置为 TRUE。此图显示操作自动完成后（无论是否检测到错误）功能块的行为：



注意1：仅当 Execute 在操作结束后复位为 FALSE 后，Done 或 Error 位在功能块执行完成后，设置为TRUE一个任务循环时间。

- 功能中止

此图显示由应用程序中止的功能。Abort 输入的上升沿取消正在执行的功能。在这种情况下，中止的输出会设置为 TRUE，并且 CommError 包含代码 Canceled - 16#02（由用户请求停止交换）：



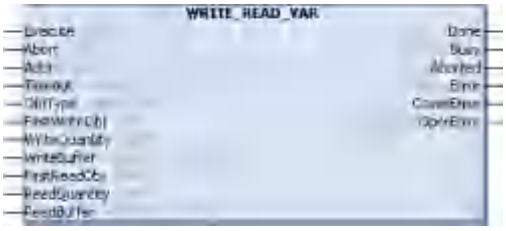
注意1：仅当 Execute 在发生中止请求后复位为 FALSE，Abort 位在功能块执行完成后，设置为TRUE一个任务循环时间。

使用限制

Write_Var功能块中的ObjType只有0-MW和02-Q两种发送数据类型，而且Write_Var功能块的作用是写入多个寄存器(MW)或者写入多个线圈(Q)。其对应的Modbus功能码是#16(写入多个寄存器MW)，#15(写入多个线圈Q)。

8.4.4 Write_Read_Var
说明: 读取和写入Modbus从站设备上的寄存器
操作符
Write_Read_Var
功能描述
Write_Read_Var功能块可以读取Modbus从站设备的内部寄存器(仅限MW类型)并将用户需要发送的数据写入到Modbus从站设备中。读取和写入操作在同一个功能块中完成。
首先执行写入操作。Write_Read_Var功能块随后可以:
- 写入连续内部寄存器并立即读回这些寄存器的值以作验证
- 在同一个唯一请求中，写入某些连续的内部寄存器，并读取其他寄存器

图形表示形式



管脚定义

输入	类型	注释
Excute	BOOL	启动引脚,上升沿触发(注意:如果在冷复位、热复位的第一个任务运行周期中将Excute置位True,则检测不到上升沿)
Abort	BOOL	终止正在执行的操作
Addr	ADDRESS	目标设备的地址(ADDM功能块的输出)
Timeout	WORD	超时时间为100ms的倍数(0表示无限)
ObjType	ObjectType	发送数据类型仅限0-MW保持寄存器
FirstWriteObj	DINT	FirstWriteObj为要写入的第一个对象的索引。
WriteQuantity	UINT	写入对象的数量 ● 1-123寄存器(MW类型)
WriteBuffer	POINTER TO BYTE	WriteBuffer为要发送数据的缓冲区地址。用户可以通过ADR标准功能指令获取缓冲区的地址(详见下面示例)。缓冲区是一个表，用户要发送的数据写入缓冲区中。例如，4个寄存器的写入值存储在包含4个字的表中，而32位的写入值则需要包含2个字或4个字节的的表中，其中每个位都设置为对应值。
FirstReadObj	DINT	FirstReadObj为要读取的第一个对象的索引。
ReadQuantity	UINT	读取对象的数量 ● 1-125 MW类型寄存器
ReadBuffer	POINTER TO BYTE	ReadBuffer为读取数据的接收缓冲区地址。用户可以通过ADR标准功能指令获取缓冲区的地址。接收缓冲区是一个表，用于接收在从站设备中读取的值。

输出	类型	注释
Done	BOOL	功能块完成后Done设置为True
Busy	BOOL	功能块在执行，Busy引脚为True
Aborted	BOOL	当Abort输入终止操作后，终止操作完成Aborted设置为True
Error	BOOL	当功能块检测到错误的时候，Error引脚设置为True，同时CommError和OperError包含相应的错误信息
CommError	BYTE	通讯错误代码(详见Read_Var中介绍)
OperError	DWORD	操作错误代码(详见Read_Var中介绍)

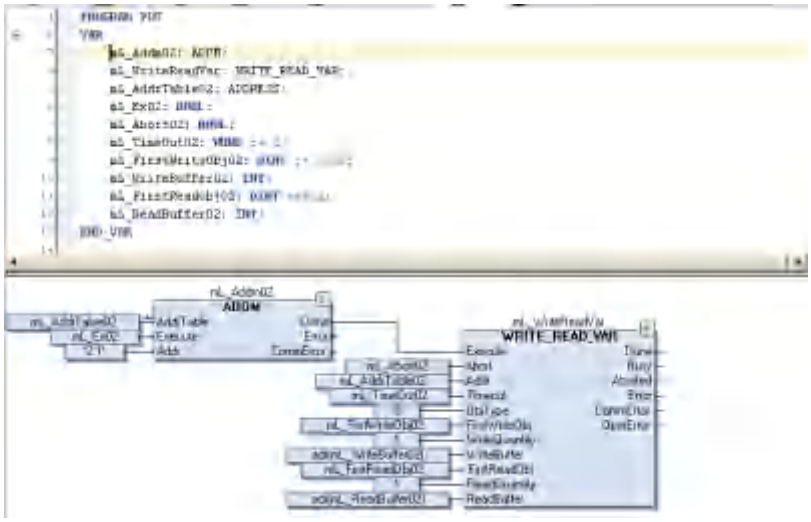
变量声明

```
VAR
mL_Addm02: ADDM;
mL_WriteReadVar: WRITE_READ_VAR;
mL_AddrTable02: ADDRESS;
mL_Ex02: BOOL;
mL_Abort02: BOOL;
mL_TimeOut02: WORD := 2;
mL_FirstWriteObj02: DINT := 8502;
mL_WriteBuffer02: INT;
mL_FirstReadObj02: DINT:=8502;
mL_ReadBuffer02: INT;
END_VAR
```

编程示例

此段程序是通过M218的串口2端口向站号为1的Modbus从站设备的内部8502寄存器写入数据，然后再读取从站设备的内部8502寄存器的内容。

CFC语言示例

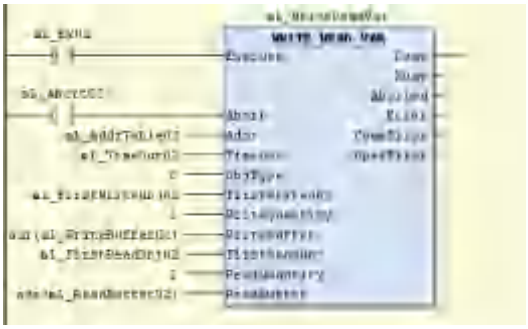


ST语言示例

```
mL_Addm02(
    Execute:= mL_Ex02,
    Addr:= '...',
    AddrTable:= mL_AddrTable02,
    Done=> ,
    Error=> ,
    CommError=> );

mL_WriteReadVar(
    Execute:= mL_Addm02.Done,
    Abort:= mL_Abort02,
    Addr:= mL_AddrTable02,
    Timeout:= mL_TimeOut02,
    ObjType:= 0,
    FirstWriteObj:= mL_FirstWriteObj02,
    WriteQuantity:= 1,
    WriteBuffer:= ADR(mL_WriteBuffer02),
    FirstReadObj:= mL_FirstReadObj02,
    ReadQuantity:= 1,
    ReadBuffer:= ADR(mL_ReadBuffer02),
    Done=> ,
    Busy=> ,
    Aborted=> ,
    Error=> ,
    CommError=> ,
    OperError=> );
```

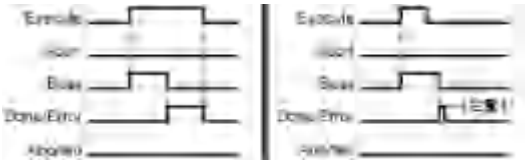
LD语言示例



时序图

功能执行

功能在 Execute 输入的上升沿启动。Busy 输出随后会设置为 TRUE。此图显示操作自动完成后(无论是否检测到错误)功能块的行为：



注意1：仅当 Execute 在操作结束后复位为 FALSE 后，Done 或 Error 位在功能块执行完成后，设置为TRUE一个任务循环时间。

功能中止

此图显示由应用程序中止的功能。Abort 输入的上升沿取消正在执行的功能。在这种情况下，中止的输出会设置为 1，并且 CommError 包含代码 Canceled - 16#02(由用户请求停止交换)：

注意1：仅当 Execute 在发生中止请求后复位为 FALSE，Abort 位在功能块执行完成后，设置为TRUE一个任务循环时间。

使用限制

Write_Read_Var功能块只能写入和读取从设备的保持型寄存器(MW).

8.4.5 Single _Write
说明: 写入Modbus从站设备上的单个寄存器
操作符
Single_Write
功能描述
Single_Write功能块将单个字的内容写入到Modbus从站设备上的单个寄存器中。
图形表示形式

管脚定义

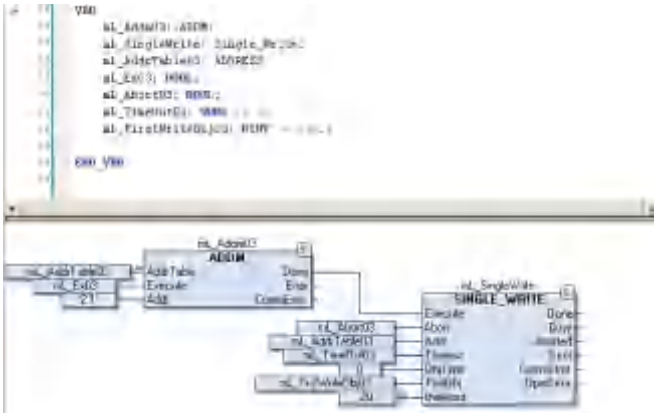
输入	类型	注释
Excute	BOOL	启动引脚,上升沿触发(注意:如果在冷复位、热复位的第一个任务运行周期中将Excute置位True,则检测不到上升沿)
Abort	BOOL	终止正在执行的操作
Addr	ADDRESS	目标设备的地址(ADDM功能块的输出)
Timeout	WORD	超时时间为100ms的倍数(0表示无限)
ObjType	ObjectType	发送数据类型仅限0-MW保持寄存器
FirstObj	DINT	FirstObj为要写入的第一个对象的索引。
theWord	Word	写入Modbus从站设备的数据

输出	类型	注释
Done	BOOL	功能块完成后Done设置为True
Busy	BOOL	功能块在执行，Busy引脚为True
Aborted	BOOL	当Abort输入终止操作后，终止操作完成Aborted设置为True
Error	BOOL	当功能块检测到错误的时候，Error引脚设置为True，同时CommError和OperError包含相应的错误信息
CommError	BYTE	通讯错误代码(详见Read_Var中介绍)
OperError	DWORD	操作错误代码(详见Read_Var中介绍)

变量声明

```
VAR
mL_Addm03: ADDM;
mL_SingleWrite: Single_Write;
mL_AddrTable03: ADDRESS;
mL_Ex03: BOOL;
mL_Abort03: BOOL;
mL_TimeOut03: WORD := 2;
mL_FirstWriteObj03: DINT := 8502;
END_VAR
```

编程示例
此段程序是通过M218的串口2端口向站号为1的Modbus从站设备的内部8502寄存器写入数据，数据值为20。CFC语言示例

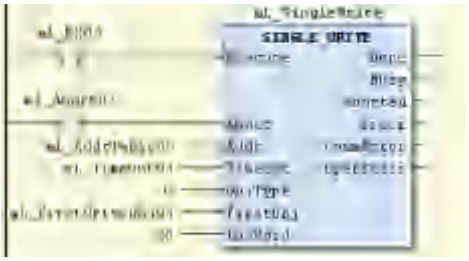


ST语言示例

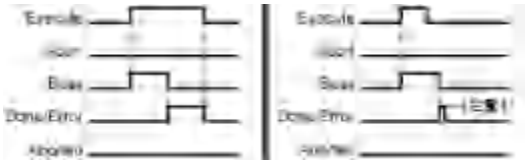
```
mA_Addm03(
    Execute:= mA_Ex03,
    Addr:= 1,
    AddrTable:= mA_AddrTable03,
    Done=> ,
    Error=> ,
    CommError=> );

mA_SingleWrite(
    Execute:= mA_Addm03.Done,
    Abort:= mA_Abort03,
    Addr:= mA_AddrTable03,
    Timeout:= mA_Timeout03,
    ObjType:= 0,
    FirstObj:= mA_FirstWriteObj03,
    theWord:= 20,
    Done=> ,
    Busy=> ,
    Aborted=> ,
    Error=> ,
    CommError=> ,
    OperError=> );
```

LD语言示例



时序图
功能执行
功能在 Execute 输入的上升沿启动。Busy 输出随后会设置为 TRUE。此图显示操作自动完成后(无论是否检测到错误)功能块的行为：



注意1：仅当 Execute 在操作结束后复位为 FALSE 后，Done 或 Error 位在功能块执行完成后，设置为TRUE一个任务循环时间。

功能中止
此图显示由应用程序中止的功能。Abort 输入的上升沿取消正在执行的功能。在这种情况下，中止的输出会设置为 1，并且 CommError 包含代码 Canceled - 16#02(由用户请求停止交换)：

注意1：仅当 Execute 在发生中止请求后复位为 FALSE，Abort 位在功能块执行完成后，设置为TRUE一个任务循环时间。

使用限制
Single_Write只能写入Modbus从站的单个保持型寄存器MW。此时，Modbus的功能码为#6。

8.4.6 SEND_RECV_MSG
说明: 发送和/或者接收用户自定义的消息
操作符
SEND_RECV_MSG
功能描述
SEND_RECV_MSG功能块用于发送和/或者接收用户自定义的消息。它在用户选定的介质(如串行线路)上发送消息，然后等待响应。另外，它也可以发送消息，但不等待响应，或者仅接收消息而不发送消息。此功能块应和ASCII码管理器配合使用图形表示形式

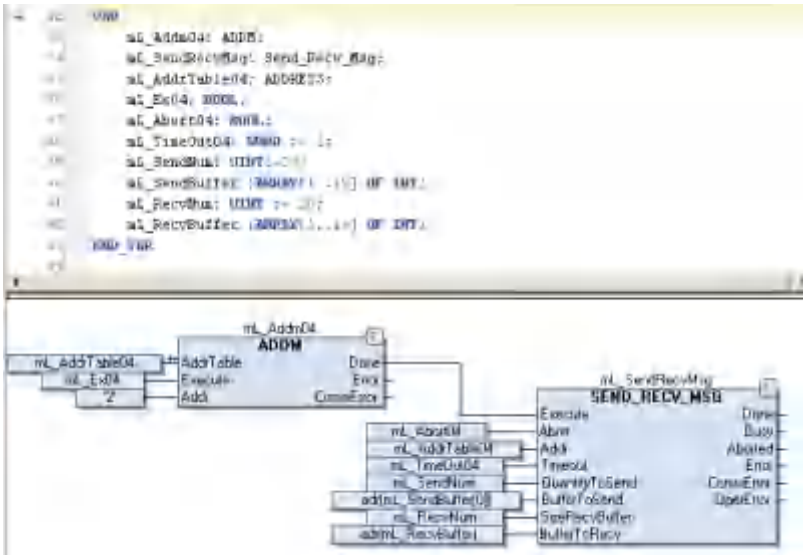
管脚定义

输入	类型	注释
Excute	BOOL	启动引脚,上升沿触发(注意:如果在冷复位、热复位的第一个任务运行周期中将Excute置位True,则检测不到上升沿)
Abort	BOOL	终止正在执行的操作
Addr	ADDRESS	目标设备的地址(ADDM功能块的输出)
Timeout	WORD	超时时间为100ms的倍数(0表示无限)
QuantityToSend	UINT	要发送的字节数量
BufferToSend	POINTER TO BYTE	BufferToSend为存储要发送的消息的缓冲区地址。用户可以通过ADR标准指令获得缓冲区的地址。当BufferToSend设置为0时，Send_Recv_Msg功能块仅作接收使用
SizeRecvBuffer	UINT	SizeRecvBuffer为接收缓冲区可以使用的大小(字节为单位)。接收数据的大小(以字节为单位)显示在功能块实例内部属性(内部变量)中：<实例名称>.NbRecvBytes
BufferToRecv	POINTER TO BYTE	BufferToRecv 为存储收到的消息的缓冲区(SizeRecvBuffer 字节数组)的地址。ADR 标准可以获取接收缓冲区的地址。(请参见下面的示例。)如果为0，则该功能进行"仅发送"操作

输出	类型	注释
Done	BOOL	功能块完成后Done设置为True
Busy	BOOL	功能块在执行，Busy引脚为True
Aborted	BOOL	当Abort输入终止操作后，终止操作完成Aborted设置为True
Error	BOOL	当功能块检测到错误的时候，Error引脚设置为True，同时CommError和OperError包含相应的错误信息
CommError	BYTE	通讯错误代码(详见Read_Var中介绍)
OperError	DWORD	操作错误代码(详见Read_Var中介绍)

变量声明
VAR
mL_Addm04: ADDM;
mL_SendRecvMsg: Send_Recv_Msg;
mL_AddrTable04: ADDRESS;
mL_Ex04: BOOL;
mL_Abort04: BOOL;
mL_TimeOut04: WORD := 2;
mL_SendNum: UINT:=20;
mL_SendBuffer :ARRAY[1..19] OF INT;
mL_RecvNum: UINT := 20;
mL_RecvBuffer :ARRAY[1..19] OF INT;
END_VAR

编程示例
此示例是通过M218的串口2发送、接收消息
CFC语言示例

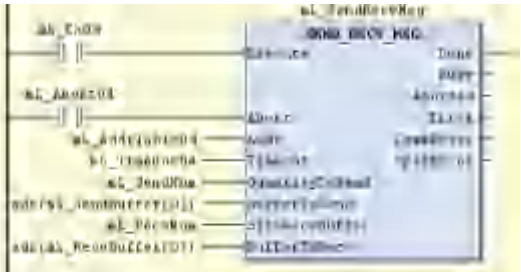


ST语言示例

```
mL_Addm04(
    Execute:= mL_Ex04,
    Addr:= 2,
    AddrTable:= mL_AddrTable04,
    Done=> ,
    Error=> ,
    CommError=> );

mL_SendRecvMsg(
    Execute:= mL_Addm04.Done,
    Abort:= mL_Abort04,
    Addr:= mL_AddrTable04,
    Timeout:= mL_TimeOut04,
    QuantityToSend:= mL_SendNum,
    BufferToSend:= ADR(mL_SendBuffer[1]),
    SizeRecvBuffer:= mL_RecvNum,
    BufferToRecv:= ADR(mL_RecvBuffer[1]),
    Done=> ,
    Busy=> ,
    Aborted=> ,
    Error=> ,
    CommError=> ,
    OperError=> );
```

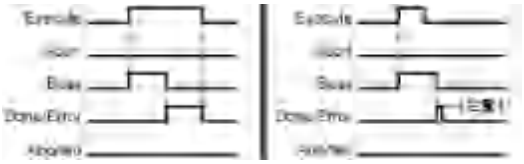
LD语言示例



时序图

功能执行

功能在 Execute 输入的上升沿启动。Busy 输出随后会设置为 TRUE。此图显示操作自动完成后(无论是否检测到错误)功能块的行为：



注意1：仅当 Execute 在操作结束后复位为 FALSE 后，Done 或 Error 位在功能块执行完成后，设置为TRUE一个任务循环时间。

功能中止

此图显示由应用程序中止的功能。Abort 输入的上升沿取消正在执行的功能。在这种情况下，中止的输出会设置为 1，并且 CommError 包含代码 Canceled - 16#02（由用户请求停止交换）：

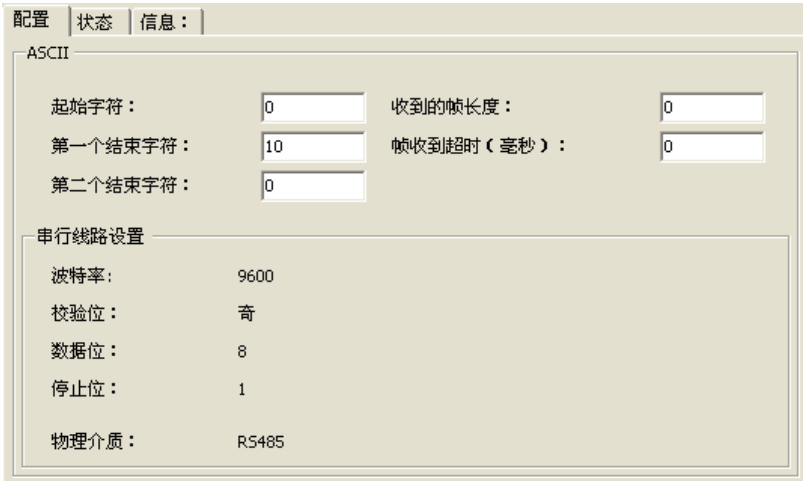
注意1：仅当 Execute 在发生中止请求后复位为 FALSE，Abort 位在功能块执行完成后，设置为TRUE一个任务循环时间。

注意事项

对于"仅发送"操作，当向线路发送完所有数据（包括最后的起始和停止字符）后，交换完成（Busy 复位为 0）。

对于发送/接收或"仅接收"操作，系统直到出现结束条件时才接收字符。当达到结束条件时，交换完成（Busy 复位为 0）。接收到的字符之后将被复制到接收缓冲区（最多 sizeRecvBuffer 个字符），并且接收数据的大小（以字节为单位）显示在功能块实例内部属性（内部变量）中：<实例名称>.NbRecvBytes。sizeRecvBuffer 输入不表示结束条件。

用户定义的消息的起始和结束条件在 ASCII 管理器的配置对话框中配置：



此示例中结束字符是10，接收帧的结束条件是检测到字符10。
当第一个结束字符和第二个结束字符都设置为0，帧收到超时(毫秒)设置为250时，接收帧的结束条件是超时250毫秒。
当第一个结束字符是10，帧收到超时(毫秒)设置为250时，接收帧的结束条件是收到结束字符10或者接收超时250毫秒。

第九章 M218系统库指南

9.1.1PLC_R/W
PLC_R 和 PLC_W 结构中包含以下的系统变量数据类型。
PLC_R_STATUS:控制器状态代码
PLC_R_APPLICATION_ERROR：检测到的应用程序错误状态代码
PLC_R_BOOT_PROJECT_STATUS:引导项目状态代码
PLC_R_IO_STATUS:I/O状态代码
PLC_R_STOP_CAUSE:从“运行”向其它状态转换的原因代码
PLC_W_COMMAND:控制命令代码

1、PLC_R_STATUS:控制器状态代码
PLC_R_STATUS 枚举数据类型包含以下值:

枚举器	值(十六进制)	注释
PLC_R_EMPTY	00	控制器未进行编程
PLC_R_STOPPED	01	控制器已停止
PLC_R_RUNNING	02	控制器正在运行
PLC_R_HALT	04	控制器处于“暂停”状态
PLC_R_BREAKPOINT	08	控制器已在断点处暂停

2、PLC_R_APPLICATION_ERROR：检测到的应用程序错误状态代码
PLC_R_APPLICATION_ERROR 枚举数据类型包含以下值

枚举器	值(十六进制)	注释
PLC_R_APP_ERR_UNKMOWN	FFFF	未知错误
PLC_R_APP_ERR_NOEXEPTION	0000	未检测到错误
PLC_R_APP_ERR_WATCHDOG	0010	任务的应用程序看门狗已过期
PLC_R_APP_ERR_HARDWAREWATCHDOG	0011	硬件看门狗已过期
PLC_R_APP_ERR_IO_CONFIG_ERROR	0012	检测到不正确的I/O配置参数
PLC_R_APP_ERR_UNRESOLVED_EXTREFS	0018	检测打未知功能
PLC_R_APP_ERR_IEC_TASK_CONFIG_ERROR	0025	检测到不正确的任务配置参数
PLC_R_APP_ERR_ILLEGAL_INSTRUCTION	0050	检测到未知指令
PLC_R_APP_ERR_ACCESS_VIOLATION	0051	对保留存储器区域的访问
PLC_R_APP_ERR_PROCESSORLOAD_WATCHDOG	0105	处理器由于应用程序任务而过载

3、PLC_R_BOOT_PROJECT_STATUS:引导项目状态代码
PLC_R_BOOT_PROJECT_STATUS 枚举数据类型包含以下值

枚举器	值(十六进制)	注释
PLC_R_NO_BOOT_PROJECT	0000	
PLC_R_ BOOT_PROJECT_CREATION_IN_PROGRESS	0001	正在创建引导项目
PLC_R_DIFFERENT_BOOT_PROJECT	0002	
PLC_R_VALID_BOOT_PROJECT	FFFF	闪存中的引导项目与RAM中加载的项目相同

4、PLC_R_IO_STATUS:I/O状态代码
PLC_R_IO_STATUS 枚举数据类型包含以下值

枚举器	值(十六进制)	注释
PLC_R_IO_OK	FFFF	输入输出运行正常
PLC_R_IO_NO_INIT	0001	输入输出未初始化
PLC_R_IO_CONF_FAULT	0002	检测到不正确的I/O配置参数
PLC_R_IO_SHORTCUT_FAULT	0003	检测到输入输出短路

5、PLC_R_STOP_CAUSE:从“运行”向其它状态转换的原因代码
PLC_R_STOP_CAUSE 枚举数据类型包含以下值

枚举器	值(十六进制)	注释
PLC_R_STOP_REASON_UNKNOWN	00	未定义初始值或停止原因
PLC_R_STOP_REASON_HW_WATCHDOG	01	硬件看门狗后停止
PLC_R_STOP_REASON_RESET	02	复位后停止
PLC_R_STOP_REASON_EXCEPTION	03	例外后停止
PLC_R_STOP_REASON_USER	04	用户请求后停止
PLC_R_STOP_REASON_IECPROGRAM	05	程序命令后停止
PLC_R_STOP_REASON_DELETE	06	删除应用程序命令后停止
PLC_R_STOP_REASON_DEBUGGING	07	进入调试模式后停止
PLC_R_STOP_FROM_NETWORK_REQUEST	0A	从网络进行请求后停止
PLC_R_STOP_FROM_INPUT	0B	控制器输入要求停止

6、PLC_W_COMMAND:控制命令代码
PLC_W_COMMAND 枚举数据类型包含以下值

枚举器	值(十六进制)	注释
PLC_W_STOP	01	用于停止控制器的命令
PLC_W_RUN	02	用于运行控制器的命令
PLC_W_RESET_COLD	04	用于启动控制器冷复位的命令
PLC_W_RESET_WARM	08	用于启动控制器热复位的命令

9.1.2 系统功能数据类型
系统功能数据类型包含了以下主题:
FIRMARE_VERSION:GetFirmwareVersion功能输出类型
BOOT_PROJECT_STATUS:GetBOOtProjectStatus功能输出代码
STOP_WHY:GetLasStopCause功能输出代码
LOCAL_IO_GET_STATUS:GetLocallOStatus功能参数代码
LOCAL_IO_GEN_STATUS:GetLocallOStatus功能输出代码
RIGHTBUS_GET_STATUS:GetRightBusStatus功能参数代码
DAY_OF_WEEK:SetRTCDrift功能日期参数代码
HOUR:SetRTCDrift功能小时参数类型
MINUTE:SetRTCDrift功能分钟参数类型

1、FIRMARE_VERSION:GetFirmwareVersion功能输出类型
该数据结构包含下列变量

变量	类型	说明
FwVersion	DWORD	固件版本
BootVersion	WORD	引导版本
AsicVersion	WORD	协处理器版本

2、BOOT_PROJECT_STATUS:GetBOOtProjectStatus功能输出代码
该数据结构包含下列变量

枚举器	值(十六进制)	说明
NO_BOOT_PROJECT	0000	未使用
BOOT_PROJECT_CREATION_IN_PROGRESS	0001	正在创建引导项目
DIFFERENT_BOOT_PROJECT	0002	闪存中的引导项目与RAM中加载的项目不同或者不存在
VALID_BOOT_PROJECT	FFFF	闪存中的引导项目与RAM中加载的项目相同

3、STOP_WHY:GetLasStopCause功能输出代码
该数据结构包含下列变量

枚举器	值(十六进制)	说明
STOP_REASON_UNKNOWN	00	为定义初始值或停止原因
STOP_REASON_HW_WATCHDOG	01	硬件看门狗后停止
STOP_REASON_RESET	02	复位后停止
STOP_REASON_EXCEPTION	03	例外后停止
STOP_REASON_USER	04	用户请求后停止
STOP_REASON_IECPROGRAM	05	程序命令后停止
STOP_REASON_DELETE	06	删除应用程序命令后停止
STOP_REASON_DEBUGGING	07	进入调试模式后停止
STOP_FROM_NETWORK_REQUEST	0A	从网络进行请求后停止
STOP_FROM_INPUT	0B	控制器输入要求停止

4、LOCAL_IO_GET_STATUS:GetLocalIOStatus功能参数代码
该数据结构包含下列变量

枚举器	值(十六进制)	说明
LOCAL_IO_GET_GEN_STATUS	00	用于请求嵌入式I/O的常规状态的值

5、LOCAL_IO_GEN_STATUS:GetLocalIOStatus功能输出代码
该数据结构包含下列变量

枚举器	值(十六进制)	说明
LOCAL_IO_OK	00	输入输出运行正常
LOCAL_IO_NO_INIT	01	检测到错误的I/O配置参数
LOCAL_IO_COM_LOST	02	协处理器的通讯错误，未更新嵌入式I/O
LOCAL_IO_CONF_FAULT	03	检测到错误的I/O配置参数

6、RIGHTBUS_GET_STATUS:GetRightBusStatus功能参数代码
该数据结构包含下列变量

枚举器	值(十六进制)	说明
RIGHTBUS_GET_GEN_STATUS	00	扩展总线配置诊断的参数
RIGHTBUS_GET_STATUS1	01	扩展总线模块1诊断的参数
RIGHTBUS_GET_STATUS2	02	扩展总线模块2诊断的参数
RIGHTBUS_GET_STATUS3	03	扩展总线模块3诊断的参数
RIGHTBUS_GET_STATUS4	04	扩展总线模块4诊断的参数
RIGHTBUS_GET_STATUS5	05	扩展总线模块5诊断的参数
RIGHTBUS_GET_STATUS6	06	扩展总线模块6诊断的参数
RIGHTBUS_GET_STATUS7	07	扩展总线模块7诊断的参数

7、DAY_OF_WEEK:SetRTCDrift功能日期参数代码
该数据结构包含下列变量

枚举器	值(十六进制)	说明
MONDAY	01	将“星期几”设置为星期一
TUESDAY	02	将“星期几”设置为星期二
WEDNESDAY	03	将“星期几”设置为星期三
THURSDAY	04	将“星期几”设置为星期四
FRIDAY	05	将“星期几”设置为星期五
SATURDAY	06	将“星期几”设置为星期六
SUNDAY	07	将“星期几”设置为星期日

8、HOUR:SetRTCDrift功能小时参数类型
该数据类型包含从 0 到 23 的小时值。

9、MINUTE:SetRTCDrift功能分钟参数类型
该数据类型包含从 0 到 59 的分钟值。

9.1.3ETH_R/W系统变量数据类型
系统功能数据类型包含了以下主题:
ETH_R_IP_MODE:IP地址源代码
ETH_R_FRAME_PROTOCOL:帧传输协议代码
ETH_R_PORT_DUPLEX_STATUS:传输模式代码
ETH_R_PORT_LINK_STATUS:通讯链路方向代码
ETH_R_PORT_SPEED:以太网端口的通讯速度代码
1、ETH_R_IP_MODE:IP地址源代码
ETH_R_IP_MODE 枚举数据类型包含以下值:

枚举器	值(十六进制)	说明
ETH_R_STORED	00	使用存储的IP地址
ETH_R_BOOTP	01	使用引导程序协议获取IP地址
ETH_R_DHCP	02	使用DHCP协议获取IP地址
ETH_DEFAULT_IP	FF	使用缺省IP地址

2、ETH_R_FRAME_PROTOCOL:帧传输协议代码
ETH_R_FRAME_PROTOCOL 枚举数据类型包含以下值:

枚举器	值(十六进制)	说明
ETH_R_802_3	00	用于帧传输的协议为802.3
ETH_R_ETHERNET_II	01	用于帧传输的协议为Ethernet II

3、ETH_R_PORT_DUPLEX_STATUS:传输模式代码
ETH_R_PORT_DUPLEX_STATUS 枚举数据类型包含以下值:

枚举器	值(十六进制)	说明
ETH_R_PORT_HALF_DUPLEX	00	使用半双工传输模式
ETH_R_FULL_DUPLEX	01	使用全双工传输模式

4、ETH_R_PORT_LINK_STATUS:通讯链路方向代码
ETH_R_PORT_LINK_STATUS 枚举数据类型包含以下值:

枚举器	值(十六进制)	说明
ETH_R_LINK_DOWN	00	通讯链路从服务器到设备
ETH_R_LINK_UP	01	通讯链路从设备到服务器

5、ETH_R_PORT_SPEED:以太网端口的通讯速度代码
ETH_R_PORT_SPEED 枚举数据类型包含以下值:

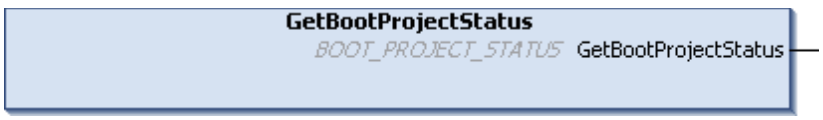
枚举器	值(十进制)	说明
ETH_R_SPEED_10_MB	10	网络速度为每秒10兆位
ETH_R_100_MB	100	网络速度为每秒100兆位

9.1.4M218读/写功能
1、M218读功能
GetBatteryLevel:返回电池的剩余电量
GetBootProjectStatus:返回引导项目状态
GetEventsNumber:返回检测到的外部事件数
GetFirmwareVersion:返回有关固件、引导和协处理器版本的信息
GetHardwareVersion:返回硬件版本
GetLastStopCause:返回上次停止的原因
GetLastStopTime:返回上次检测到停止的日期和时间
GetLocalIOStatus:返回嵌入式I/O状态
GetLocalAIOStatus:返回嵌入式模拟量I/O状态
GetPlcFault:返回控制器I/O上检测到的错误
GetRightBusStatus:返回扩展总线的状态
GetSerialNumber:返回控制器的序列号
GetShortCutStatus:返回嵌入式输出上的短路状态
IsFirstMastColdCycle:指示循环是否为第一个MAST冷启动循环
IsFirstMastCycle:指示循环是否为第一个MAST循环
IsFirstMastWarmCycle:指示循环是否为第一个MAST热启动循环
1.1 GetBatteryLevel:返回电池的剩余电量
功能描述
此功能返回外部备用电池的剩余电量（以百分比表示）。
图形表示形式

I/O 变量描述
下表对输出变量进行了描述:

输出	类型	说明
GetBatteryLevel	WORD	电池中剩余电量的百分比。 范围: 0..100: 100%= 3.5 V 50% = 2.75 V 0%= 2 V 电池 LED 的闪烁阈值: 100%= 常灭 50%= 定期闪烁 0%= 常亮

1.2 GetBootProjectStatus:返回引导项目状态
功能描述
此功能返回引导项目状态。
图形表示形式



I/O 变量描述
下表对输出变量进行了描述:

输出	类型	说明
GetBootProjectStatus	BOOT PROJECT STATUS	引导项目的状态。

1.3 GetEventsNumber:返回检测到的外部事件数
功能描述
该功能返回自上一次冷启动以来发生的事件数（包括输入上检测到的事件和 HSC 阈值比较事件）。
图形表示形式

I/O 变量描述
下表对输出变量进行了描述：

输出	类型	说明
GetEventsNumber	DWORD	该值表示自上一次冷启动以来发生的事件数。通过调用 RestEventsNumber功能复位为 0。

1.4 GetFirmwareVersion:返回有关固件、引导和协处理器版本的信息
功能描述
此功能返回有关控制器的以下信息：
• 固件版本
• 引导版本（固件更新的内核）
• 协处理器版本（专用于嵌入式 I/O 管理的 ASIC）
图形表示形式

I/O 变量描述
下表对输出变量进行了描述：

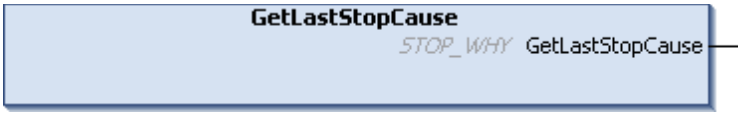
输出	类型	说明
GetFirmwareVersion	FIRMWARE VERSION	固件、引导和协处理器版本。

1.5 GetHardwareVersion:返回硬件版本
功能描述
此功能返回控制器的硬件版本。
图形表示形式

I/O 变量描述
下表对输出变量进行了描述：

输出	类型	说明
GetHardwareVersion	WORD	控制器的硬件版本

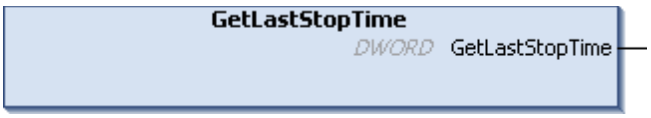
1.6 GetLastStopCause:返回上次停止的原因
功能描述
此功能返回上次从"运行"转换为其他状态的原因。
图形表示形式



I/O 变量描述
下表对输出变量进行了描述：

输出	类型	说明
GetLastStopCause	STOP_WHY	上次从"运行"转换为其他状态的原因

1.7 GetLastStopTime:返回上次检测到停止的日期和时间
功能描述
此功能返回上次从"运行"转换为其他状态的日期和时间。
图形表示形式



I/O 变量描述
下表对输出变量进行了描述：

输出	类型	说明
GetLastStopTime	DWORD	上次检测到"停止"的时间（以秒为单位，从 1970 年 1 月 1 日 00:00 开始计起）

1.8 GetLocalIOStatus:返回嵌入式I/O状态
功能描述
此功能返回嵌入式 I/O 状态。
图形表示形式

I/O 变量描述
下表对输入参数进行描述：

输入	类型	说明
Mode	LOCAL_IO_GET_STATUS	功能的参数：当前只能使用LOCAL_IO_GET_GEN_STATUS(00十六进制)

下表对输出变量进行了描述：

输出	类型	说明
GetLocalIOStatus	LOCAL_IO_GEN_STATUS	嵌入式 I/O 的状态

示例 1
此示例演示如何将 LOCAL_IO_GET_GEN_STATUS 枚举类型的 LOCAL_IO_GET_STATUS 枚举器直接用于 Mode 输入
参数：
VAR MyLocalIOStatus :LOCAL_IO_GEN_STATUS; END_VAR

MyLocalIOStatus := GetLocalIOStatus(LOCAL_IO_GET_GEN_STATUS);
示例 2
此示例演示如何将中间变量用于 Mode 输入参数。
VAR MyLocalIOStatus :LOCAL_IO_GEN_STATUS; MyMode :LOCAL_IO_GEN_STATUS; END_VAR

MyMode := LOCAL_IO_GET_GEN_STATUS; MyLocalIOStatus := GetLocalIOStatus(MyMode);

1.9 GetLocalAIOStatus:返回嵌入式模拟量I/O状态
功能描述
此功能返回本地输入和输出的状态。
图形表示形式

I/O 变量描述
下表对输出变量进行了描述:

输出	类型	说明
位0	WORD	模拟量初始化失败
位1	WORD	模拟量通讯丢失
位2	WORD	模拟量15V丢失
位3	WORD	模拟量5V丢失
位4	WORD	模拟量通道1输出错误
位5	WORD	模拟量通道2输出错误
位6	WORD	模拟量通道1输出超出范围
位7	WORD	模拟量通道2输出超出范围
位8到位15	WORD	保留

1.10 GetPlcFault:返回控制器I/O上检测到的错误
功能描述
此功能采用位域返回控制器 I/O 的常规诊断（十六进制的 FFFF 表示未检测到错误）:
图形表示形式

I/O 变量描述
下表对输出变量进行了描述:

输出	类型	说明
GetPlcFault	WORD	采用位域的控制器 I/O 诊断（十六进制的 FFFF 表示未检测到错误）。

下表详细介绍输出位域中包含的位:

位	说明
0	FALSE = 在 I/O 扩展总线上检测到错误。
1	FALSE = 在嵌入式 I/O 上检测到错误。
2到15	保留位，始终为 1

1.11 GetRightBusStatus:返回扩展总线的状态

功能描述

此功能采用位域返回 I/O 扩展总线状态。根据输入参数 (Mask)，该功能返回对 I/O 扩展总线的配置诊断（I/O 扩展总线配置与插入的模块不匹配）或所请求扩展模拟量 I/O 模块的详细诊断。

图形表示形式

I/O 变量 说明

下表对输入参数进行描述：

输入	类型	说明
Mask	RIGHTBUS GET STATUS	定义功能返回的 I/O 扩展总线诊断的类型：总线配置或 7 个可能的扩展模块之一

下表对输出变量进行了描述：

输出	类型	说明
GetRightBusStatus	字	I/O 扩展总线诊断。

I/O 扩展总线配置诊断

下表描述当扩展总线配置诊断的输入参数 (Mask) 为RIGHTBUS GET STATUS时，GetRightBus-Status 功能返回的位域

（如果未检测到错误，则为十六进制的 0000）

位	说明
0	保留位（始终为 0）
1	如果 TM2 模块 1 配置与插入的模块不匹配，则为 TRUE
2	如果 TM2 模块 2 配置与插入的模块不匹配，则为 TRUE
3	如果 TM2 模块 3 配置与插入的模块不匹配，则为 TRUE
4	如果 TM2 模块 4 配置与插入的模块不匹配，则为 TRUE
5	如果 TM2 模块 5 配置与插入的模块不匹配，则为 TRUE
6	如果 TM2 模块 6 配置与插入的模块不匹配，则为 TRUE
7	如果 TM2 模块 7 配置与插入的模块不匹配，则为 TRUE
8到15	保留位（始终为 0）

I/O 扩展总线模块诊断

下表介绍当扩展模块"x"的详细诊断的输入参数 (Mask) 为RIGHTBUS GET STATUSx（其中 x=1 至 7）时，GetRightBusStatus 功能返回的位域：

注意： 详细诊断仅对模拟量 I/O 模块有效，位域含义取决于相关模拟量模块的类型。

当关联模块为标准模拟量 I/O 模块时（最多 2 路输入通道）：

- TM2AMM3HT
- TM2ALM3LT
- TM2AMI2HT
- TM2AMI2LT
- TM2AVO2HT
- TM2AMO1HT

位	说明
0	所有模拟量通道处于正常状态
1	模块处于初始化状态
2	电源未正常运行
3	配置不正确 - 需要分析
4	输入通道 0 处于转换过程
5	输入通道 1 处于转换过程
6	输入通道 0 的参数无效
7	输入通道 1 的参数无效
8	未使用
9	未使用
10	输入通道 0 的值溢出
11	输入通道 1 的值溢出
12	输入通道 0 的值下溢
13	输入通道 1 的值下溢
14	未使用
15	输出通道的参数无效

关联模块为 4 路或 8 路模拟量输入通道模块之一时：
- TM2ARI8HT
- TM2AMI8HT
- TM2ARI8LT
- TM2ARI8LRJ
- TM2AMI4LT
- TM2AMM6HT

位	说明	含义
0，1	通道 0 状态	00: 模拟量通道处于正常状态 01: 输入通道的参数无效 10: 输入值不可用（模块处于初始化状态、转换过程） 11: 输入通道的值无效（值溢出或下溢）
2，3	通道 1 状态	请参见位 0、1
4，5	通道 2 状态	请参见位 0、1
6，7	通道 3 状态	请参见位 0、1
8，9	通道 4 状态	请参见位 0、1（仅用于 8 路输入通道模块）
10，11	通道 5 状态	请参见位 0、1（仅用于 8 路输入通道模块）
12，13	通道 6 状态	请参见位 0、1（仅用于 8 路输入通道模块）
14，15	通道 7 状态	请参见位 0、1（仅用于 8 路输入通道模块）

注意： 当目标扩展模块是数字量 I/O 模块、高速计数器模块或 AS-i 主模块时，返回的诊断无效（十六进制的 0000）。
要获取对 HSC 和 AS-i 扩展模块的诊断，可以使用：
- 用于 TM200 HSC206D• 模块的HSCGetDiag功能块
- 用于 TWDNOI10M3 模块的 ASI_MasterStatusCheck

示例
下面的示例介绍使用 GetRightBusStatus 诊断 I/O 扩展总线和模拟量模块的方法：

```
VAR (*模块 1 至 7 配置诊断 = MyRightBusStatus 位 1 至 7*) MyRightBusStatus:WORD; (*模块 1 至 7 诊断代码*) ModuleError:Array [1..7] of WORD; END_VAR

(*扩展总线上的配置诊断*)
MyRightBusStatus:=GetRightBusStatus(RIGHTBUS_GET_GEN_STATUS);

IF MyRightBusStatus<>0 THEN (*检测到配置不匹配 => 设置警报，检查位值...*)END_IF;

(*获取模块诊断；检测到错误，如果诊断 <> 0*) (*将列表限制为仅用于已配置的模拟量模块*)
ModuleError[1]:=GetRightBusStatus(RIGHTBUS_GET_STATUS1);
ModuleError[2]:=GetRightBusStatus(RIGHTBUS_GET_STATUS2);
ModuleError[3]:=GetRightBusStatus(RIGHTBUS_GET_STATUS3);
ModuleError[4]:=GetRightBusStatus(RIGHTBUS_GET_STATUS4);
ModuleError[5]:=GetRightBusStatus(RIGHTBUS_GET_STATUS5);
ModuleError[6]:=GetRightBusStatus(RIGHTBUS_GET_STATUS6);
ModuleError[7]:=GetRightBusStatus(RIGHTBUS_GET_STATUS7);
```

1.12 GetSerialNumber:返回控制器的序列号

功能描述
此功能返回控制器的序列号。
图形表示形式

I/O 变量描述
下表对输出变量进行了描述：

输出	类型	说明
GetSerialNumber	WORD	控制器序列号

1.13 GetShortCutStatus:返回嵌入式输出上的短路状态

功能描述
此功能返回嵌入式输出上的短路或过载诊断。
图形表示形式

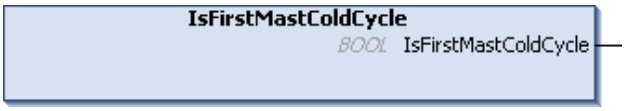
I/O 变量描述
下表对输出变量进行了描述：

输出	类型	说明
GetShortCutStatus	WORD	请参见下面的位域说明

下表描述 TM218LDA40DR4PHN、TM218LDA40DRPHN 以及 TM218LDAE40DRPHN 的位域：

位	说明
0	TRUE= 输出组 1（Q0 至 Q3）上短路。

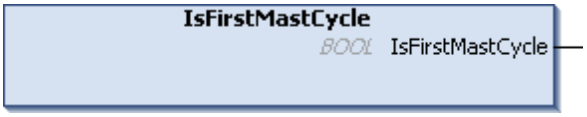
1.14 IsFirstMastColdCycle:指示循环是否为第一个MAST冷启动循环
功能描述
此功能在冷启动之后的第一个 MAST 循环期间（下载或冷复位后的第一个循环）返回 TRUE。
图形表示形式



I/O 变量描述
下表对输出变量进行描述:

输出	类型	说明
IsFirstMastColdCycle	BOOL	冷启动之后的第一个 MAST 任务循环期间为 TRUE

1.15 IsFirstMastCycle:指示循环是否为第一个MAST循环
功能介绍
此功能在启动后的第一个 MAST 循环期间返回 TRUE。
图形表示形式



I/O 变量介绍

输出	类型	说明
IsFirstMastCycle	BOOL	启动之后的第一个 MAST 任务循环期间为 TRUE

1.1.6 IsFirstMastWarmCycle:指示循环是否为第一个MAST热启动循环
功能描述
此功能在热启动之后的第一个 MAST 循环期间返回 TRUE。
图形表示形式

I/O 变量描述
下表对输出变量进行描述

输出	类型	说明
IsFirstMastWarmCycle	BOOL	热启动之后的第一个 MAST 任务循环期间为 TRUE

2、M218写功能
InhibtBatLowLed：禁用或者重行启动电池LED
ResetEventsNumber：复位事件数
SetRTCDrift：每周调整实时时钟
2.1 InhibtBatLowLed：禁用或者重行启动电池LED
功能描述
此功能禁用或重新启用低电池电量 LED 指示灯（电池）的显示。
在控制器不使用外部电池的情况下可以使用此功能，以避免出现持久的错误信号。在执行复位到起点命令或从编程面板下载到控制器后，会自动恢复低电池电量 LED 指示灯（电池）的显示。
图形表示形式

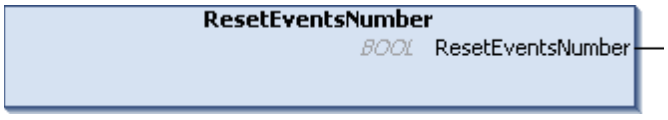
I/O 变量描述
下表对输入参数进行描述：

输入	类型	说明
Inbibit	BOOL	TRUE = 禁用电池 LED 的显示 FALSE = 重新启用电池 LED 的显示

下表对输出参数进行描述

输出	类型	说明
InhibitBatLowLed	WORD	00 十六进制 = 操作成功 否则 = 操作不成功

2.2 ResetEventsNumber：复位事件数
功能描述
该功能将自上一次冷启动以来发生的事件数（包括输入上检测到的事件和 HSC 阈值比较事件）复位。图形表示形式



I/O 变量描述
下表对输出变量进行了描述：

输出	类型	说明
ResetEventsNumber	BOOL	如果计数器成功复位为 0，则为 TRUE

2.3 SetRTCDrift：每周调整实时时钟
功能介绍
此功能用于在每周的指定日的某时某分 30 秒对实时时钟增加或减去指定的秒数。
注意： 必须至少调用一次 SetRTCDrift 功能需要对 进行编程，使其只能在第一个 Mast 循环期间执行。 图形表示形式

I/O 变量介绍
下表介绍了输入参数:

输入	类型	说明
RTCDrift	SINT(-29..29)	按秒修正 (-29 ... +29)
日	DAY OF WEEK	在一周中的哪一天执行
时	HOUR	在几点钟进行更改
分	MINUTE	在第几分钟进行更改。 在每周的指定日的几时几分 30 秒，RTC 被设置成几时几分 (30+RTCDrift) 秒。

注意： 如果为 RTCDrift、日、时、分输入的值超过了限制值，则控制器固件会将所有值设置为其最大值。 下表介绍了输出变量:

输出	类型	说明
SetRTCDrift	WORD	如果命令正常运行，则返回十六进制的 00，否则返回检测到的错误的 ID 代码

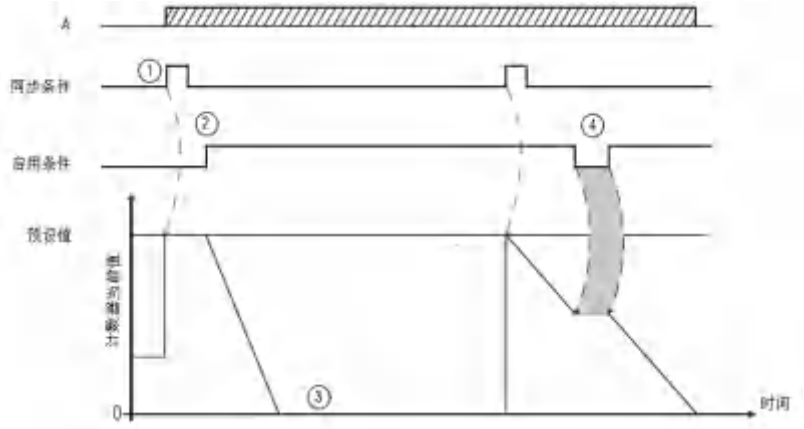
示例
在此示例中，第一个 MAST 任务循环期间仅调用此功能一次，每个星期二的上午 5:45:30 为 RTC 增加 20 秒:

```
VAR
MyRTCDrift :SINT (-29..29) := 0;
MyDay :DAY_OF_WEEK;
MyHour :HOUR;
MyMinute :MINUTE;
END_VAR
IF IsFirstMastCycle()
THEN MyRTCDrift := 20; MyDay := TUESDAY; MyHour := 5; MyMinute := 45; SetRTCDrift(MyR-
TCDrift, MyDay, MyHour,
MyMinute);
END_IF
```

9.2.1 高速计数功能概述

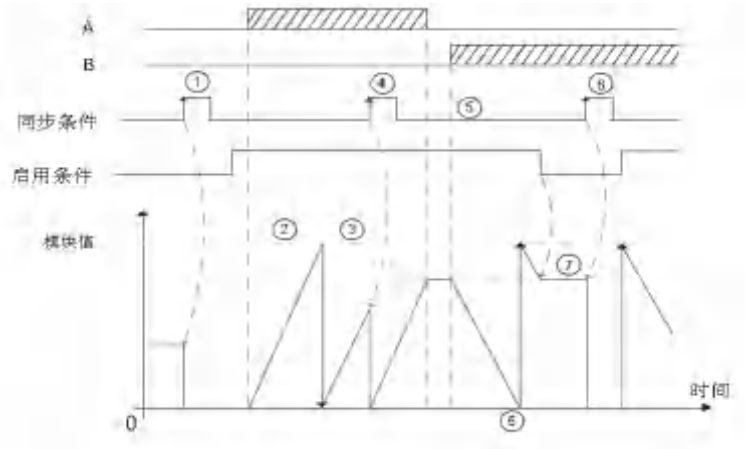
HSC（High Speed Counter）功能可以对来自与快速输入管脚相连的传感器、编码器、光电开关等原件产生的高速开关量信号进行精确计数。
M218本体的HSC可以设置为Simple （简单计数） 和Main （复杂计数）两种类型，对于TM200HSC206DT/F高速计数模块称为Specialized型，其功能与Main型类似。以上三种高速计数器全部为32位计数器。
Simple类型：单路计数模式，可接入速度传感器，接近开关或者编码器的单个信号（A或者B）等。计数方式为减计数，工作模式支持一次性模式和模数回路模式，不支持快速捕捉功能。Simple类型高速计数器都是通过软件来进行复位或启动，不支持通过外部输入信号进行复位或启动操作，最大计数频率为100KHZ
Main类型：可接入速度传感器，接近开关或者AB相增量型编码器，支持数值比较快速输出（Reflex）功能，快速中断捕捉功能。工作模式支持一次性，模数回路，自由大型计数，事件及频率计。Main类型高速计数器的复位和启动支持外部物理输入信号或软件强制控制。最大计数频率为100KHZ。另外每一个Main类型计数器需要占用两个高速计数通道。举例说明：TM218LDAE40DRPHN具有四路高速计数通道，当使用Simple型时，具有4路单相高速计数功能，如果采用Main型时，只具有两路高速计数功能。

一、 M218高速计数器模式简介
M218本体的高速计数器支持以下几种计数模式：一次性，模数回路，自由大型，事件，频率计。
（1）一次性：计数器通过同步（SYNC）信号触发后，将预设值加载到当前值并开始计数，每个脉冲导致当前计数值减1。当前值达到 0 时计数器自动停止。如果要再次启动计数器，需要新的同步（SYNC）信号。计数范围0至2147483647
应用举例：生产线传送物料，要求每20个分为一组进行打包，可采用一次性模式进行计数操作。



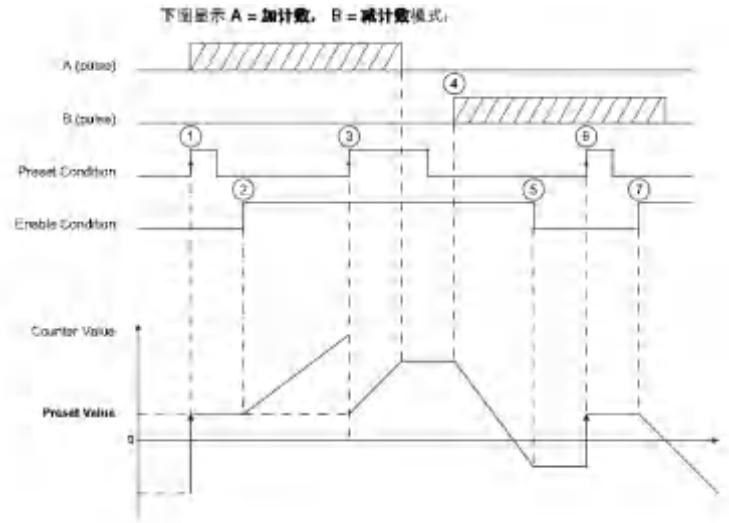
①在同步（SYNC）信号的上升沿，预设值加载到当前值。
②当启动（EN）条件=TRUE时，计数器开始递减计数直到0
③计数到0后，计数器停止
④启动（EN）条件=FALSE时，计数器暂停，保持当前数值

（2）模数回路：在此模式下，若为增计数，则计数器从 0 计数到用户定义的模数值，然后返回到 0 循环计数。如果为减计数，计数器从预设模数值减计数到 0循环计数 。Simple类型只支持减计数，Main类型支持双方向计数。计数范围0至2147483647
应用举例：某机型有一转盘并同轴装有一个编码器，编码器转一圈反馈到PLC 4096个脉冲，可采用模数回路方式进行控制，分辨转盘是否走过一圈
时序图：



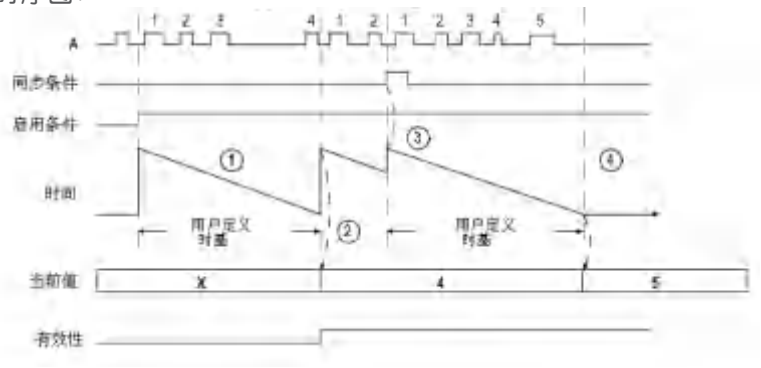
①同步（SYNC）信号上升沿，当前值清零。
②启用（EN）条件为TRUE时，若为增计数（A），则计数值增加，若为减计数（B），则计数值从预设模数值开始递减。
③当计数到达预设模数值后自动清零，自动重新开始计数，并且功能块Modulo_Flag信号置位。
④计数过程中如果同步（SYNC）信号输入上升沿则当前计数值清零。
⑤计数过程中可以改变计数方向。
⑥计数值减到0后，下一个周期自动从预设模数值开始递减。
⑦启用（EN）条件为FALSE时，计数器暂停。
⑧同步（SYNC）信号清零与启动（EN）状态无关。

（3）自由大型：只有Main类型支持自由大型计数。在此模式下，计数器从用户预设值开始向上或向下进行计数，最大计数值为-2147483648至2147484647。
应用举例：某线材生产设备需要通过编码器来计算生产了多少米的线材，可采用自由大型计数方式
时序图：



①同步（SYNC）信号上升沿，预设值赋给当前值。
②启用（EN）条件为TRUE时，若为增计数（A），则计数值增加，若为减计数（B则计数值从预设值开始递减。
③同步（SYNC）信号上升沿，预设值再次赋给当前值。
④加减计数可以随时改变。
⑤启用（EN）条件为FALSE时，计数器暂停。
⑥同步（SYNC）信号清零与启动（EN）状态无关。
⑦启用（EN）条件为TRUE时，继续计数。

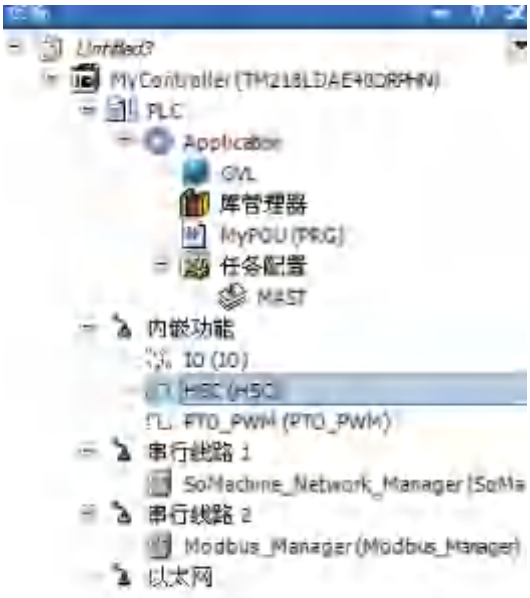
（4）事件计数：只有Main类型才支持事件计数。在此模式下，计数器对在用户配置的时基期间接收的事件数进行累计，并且可以通过命令或输入管脚进行事件计数复位。时基可设定为0.1s，1s，10s，60s。
应用举例：某机构转动一圈会产生1个脉冲，如果设定时基为1s，即可知道该机构一秒钟转了多少圈。
时序图：



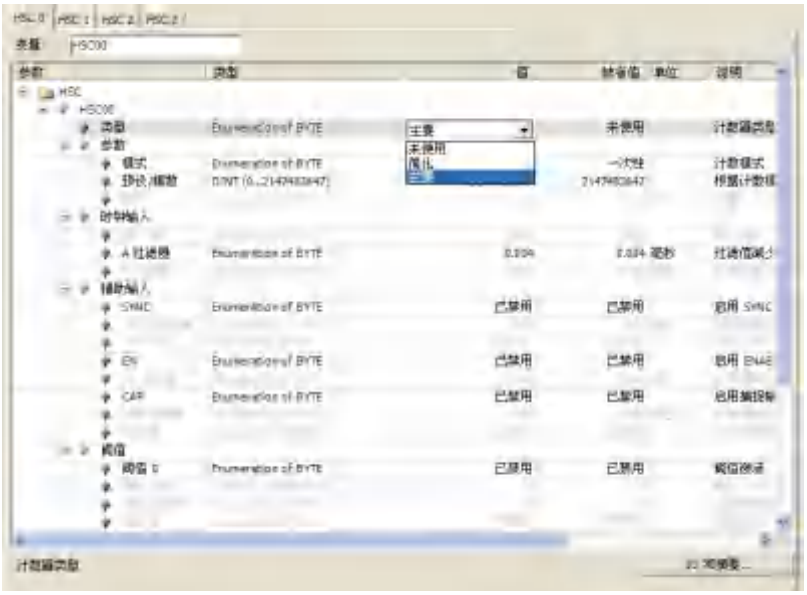
①启用（EN）条件为TRUE时，按照用户设定时基开始计数。
②当达到一次计数时基后，功能块Validity管脚置位，代表完成了一次有效计数。
③同步（SYNC）信号上升沿时，计数器重新启动，重新按照定义时基进行计数，计数值保持为上一个计数周期数值。
④计数器当前值保持为上一个计数周期的值。

（5）频率计：只有Main模式支持频率计模式。此模式与事件计数模式基本相同，唯一的区别就是无法强制进行频率计工作复位。时基可设定为0.1s，1s，10s，60s。最大频率为100KHZ
应用举例：某机构转动一圈会产生1个脉冲，如果设定时基为1s，即可知道该机构一秒钟转了多少圈（工作频率）。

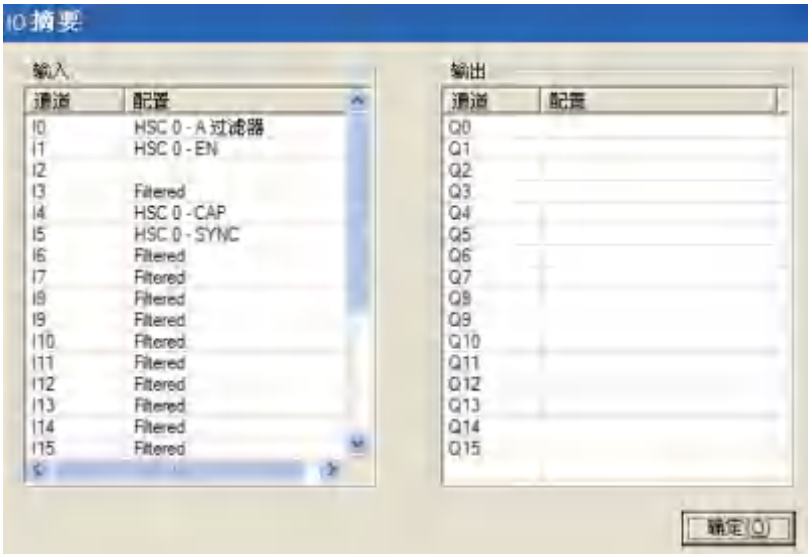
二、高速计数器配置方法：
（1）鼠标单击HSC功能按钮



（2）设置界面

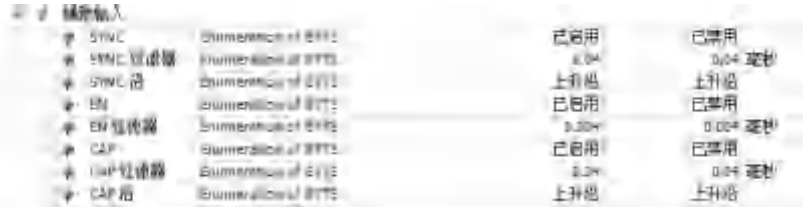


1.左上角HSC 0-HSC 3代表该PLC具有的所有高速计数通道数
2.“变量：HSC00”为该通道映射的变量名称，在使用高速计数功能块时需要指向HSC_REF，此名称是该计数器通道的
唯一标识
3.点击右下角“IO和摘要”可以查看使用高速计数器时所占用的PLC输入输出管脚。

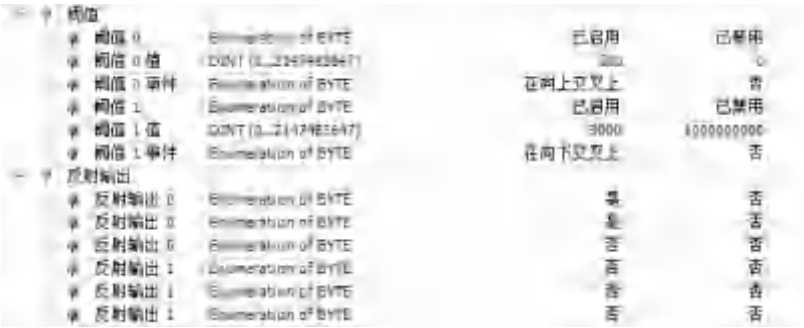


（3）输入设置具体说明

1.A=向上，B=向下：通过两个管脚来实现两个方向的计数
2.A=脉冲，B=方向：一个管脚进行计数，另一个管脚来控制计数方向
3.正常积分：接收增量式编码器AB向信号，支持X1，X2（两倍频），X4（四倍频）三种方式，并且可以设定默认计数方向。



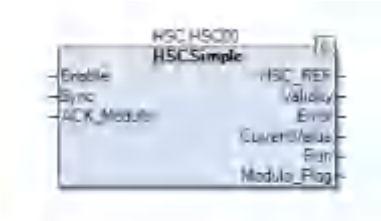
1.SYNC：高速计数器同步（SYNC）信号设置。每次SYNC信号上升沿会触发高速计数器复位。SYNC信号在某种意义上可以认为是高速计数器的初始化信号，所以要想使用计数器必须保证PLC上电后至少有一次SYNC信号上升沿。SYNC功能可通过硬件输入或软件来强制进行。
2.EN：设置是否通过PLC物理输入来启用高速计数器。EN信号与SYNC信号不冲突，SYNC代表初始化高速计数器，EN代表允许高速计数器运行。EN功能可通过PLC物理输入或者软件来强制进行。
3.CAP：用来开启高速计数器的中断捕捉功能。



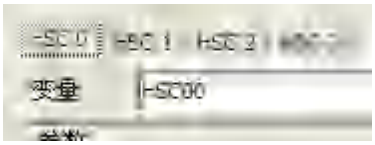
1.M218的比较器具有两个阈值，我们可以分别设定两个阈值的数值及比较条件
2.反射输出用来设定由数值比较功能触发的快速输出。两个反射输出管脚可以根据计数器当前值与两个阈值的比较结果触发对应的信号。每个MAIN类型计数器可以设定两个反射输出
下表概述了所有HSC 模式及其模式下的各种硬件规格：

模式	功能	Simple 类型	Main 类型
一次性	计数模式	减计数	减计数
	最大计数频率	100 kHz	100 kHz
	通过 HSC 物理输入启用	否	是
	通过 HSC 物理输入同步/预设	否	是
	比较功能	否	是，2 个阈值，2 个输出和事件
	捕捉功能	否	是，1 个捕捉寄存器
模数回路	计数模式	加计数	单相，加/减计数；脉冲/方向，积分
	最大计数频率	100 kHz	100 kHz
	通过 HSC 物理输入启用	否	是
	通过 HSC 物理输入同步/预设	否	是
	比较功能	否	是，2 个阈值，2 个输出和事件
	捕捉功能	否	是，1 个捕捉寄存器
自由大型	计数模式	-	加/减计数，脉冲/方向，积分
	最大计数频率	-	100 kHz
	通过 HSC 物理输入启用	-	否
	通过 HSC 物理输入同步/预设	-	是
	比较功能	-	是，2 个阈值，2 个输出和事件
	捕捉功能	-	是，1 个捕捉寄存器
事件	计数模式	-	给定时间段内的脉冲计数
	最大计数频率	-	100 kHz
	通过 HSC 物理输入启用	-	否
	通过 HSC 物理输入同步/预设	-	是
	比较功能	-	否
	捕捉功能	-	否
频率计	计数模式	-	时基内的脉冲计数
	最大计数频率	-	100 kHz
	通过 HSC 物理输入启用	-	否
	通过 HSC 物理输入同步/预设	-	否
	比较功能	-	否
	捕捉功能	-	否

9.2.2 HSCSimple



HSCSimple是控制Simple型计数器的功能块。与一般功能块不同的是HSCSimple功能块不允许随便起名，该名称必须与硬件配置中Simple型高速计数通道的变量名称链接起来。一般命名为HSC.变量名称。



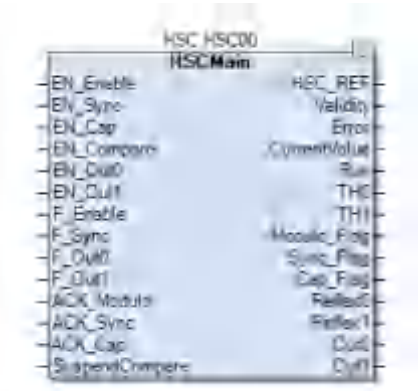
下表介绍了输入变量：

输入	类型	注释
Enable	BOOL	TRUE 表示启用当前SIMPLE型高速计数器
Sync	BOOL	同步信号，上升沿将初始化当前计数器，复位计数器的当前值
ACK_Modulo	BOOL	在模数回路模式中，上升沿将复位Modulo_Flag管脚。

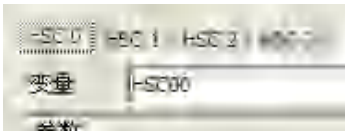
下表介绍了输入变量：

输入	类型	注释
HSC_REF	HSC_REF	功能块根据指定的通道（范例为HSC.HSC00）自动生成的该通道HSC实例名称，每一个高速计数器都会有一个唯一的名称与其对应，在使用其他高速计数相关功能块时必须将此变量赋给其他功能块的对应管脚，以告知哪一个高速计数器将要被操作。
Validity	BOOL	TRUE 表示功能块有效。
HSC_Err	BOOL	TRUE 表示检测到错误，可以使用 HSCGetDiag 功能块获取有关检测到的错误信息。
Run	BOOL	TRUE 表示计数器正在运行。 一次性模式中，当前值达到 0 时该管脚置位，高速计数器自动停止，要重新启动一次性计数器，SYNC管脚要重新输入上升沿。
Modulo_Flag	BOOL	在模数回路模式中，计数器达到一次模数时置位。
CurrentValue	DWORD	计数器的当前计数值。

9.2.3 HSCMain



HSCMain是控制Main型计数器的功能块。与一般功能块不同的是HSCMain功能块不允许随便起名，该名称必须与硬件配置中Main型高速计数通道的变量名称链接起来。一般命名为HSC.变量名称。



下表介绍了输入变量:

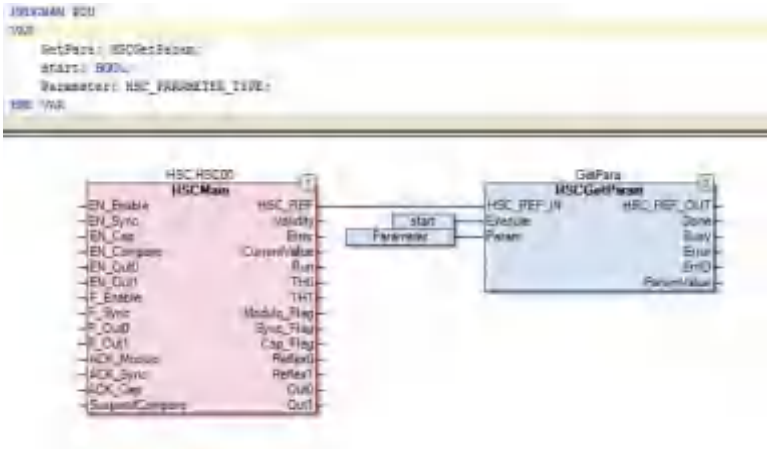
输入	类型	描述
EN_Enable	BOOL	TRUE代表允许通过物理输入来控制计数器运行
EN_Sync	BOOL	TRUE代表允许通过物理输入来控制计数器同步初始化
EN_Cap	BOOL	TRUE代表开启高速计数器硬件中断捕捉功能
EN_Compare	BOOL	TRUE 代表启用阈值比较功能
EN_Out0	BOOL	TRUE 代表允许OUT0进行比较结果输出
EN_Out1	BOOL	TRUE 代表允许OUT1进行比较结果输出
F_Enable	BOOL	TRUE代表强制运行高速计数器（忽略硬件输入EN）
F_Sync	BOOL	在上升沿时，可强制同步初始化高速计数器（忽略硬件输入SYNC）
F_Out0	BOOL	软件强制比较结果OUT0为TRUE
F_Out1	BOOL	软件强制比较结果OUT1为TRUE
ACK_Modulo	BOOL	在上升沿时，复位 Modulo_Flag信号
ACK_Sync	BOOL	在上升沿上，复位 Sync_Flag信号
ACK_Cap	BOOL	在上升沿时，复位 Cap_Flag信号
SuspendCompare	BOOL	TRUE 时暂停比较功能，相关输出信号保留上一次比较结果 • TH0、TH1、Reflex0、Reflex1、Out0、Out1、物理输出 Q0 和 Q1 均保持上一个状态。

下表介绍了输出变量:

输出	类型	描述
HSC_REF	HSC_REF	功能块根据指定的通道（范例为HSC.HSC00）自动生成的该通道HSC实例名称，每一个高速计数器都必须有一个唯一的名称与其对应，在使用其他高速计数相关功能块时必须将此变量赋给其他功能块的对应管脚，以告知哪一个高速计数器将要被操作。
Validity	BOOL	TRUE 表示高速计数器运行正常
HSC_Err	BOOL	TRUE 表示检测到错误。 使用 HSCGetDiag 功能块获得有关此检测到的错误的详细信息。
CurrentValue	DINT	计数器的当前值。
Run	BOOL	TRUE 表示计数器正在运行。 在一次性模式下，当前值达到 0 时RUN清零。
TH0	BOOL	当前计数器值 > 阈值 0 时为TRUE；只有在设置了 EN_Compare 后才处于活动状态。
TH1	BOOL	当前计数器值 > 阈值 1 时为TRUE；只有在设置了 EN_Compare 后才处于活动状态。
Modulo_Flag	BOOL	当发生一次以下情况时该输出置TRUE; 模数回路：当计数器达到模数或 0 时； 自由大型：当计数器达到其限制时
Sync_Flag	BOOL	当计数器通过物理输入或软件强制发生同步操作后该输出置TRUE
Cap_Flag	BOOL	TRUE 代表成功进行了一次高速捕捉，在进行新的捕捉之前，须先复位此标志。
Reflex0	BOOL	高速中断输出Reflex0 （Q0）的状态;只有在设置了 EN_Compare 后才处于活动状态。
Reflex1	BOOL	高速中断输出Reflex1 （Q1）的状态;只有在设置了 EN_Compare 后才处于活动状态。
Out0	BOOL	如果开启了阈值比较功能，则通过OUT0显示比较结果。当开启了反射输出功能时Out0与Reflex0相等。
Out1	BOOL	如果开启了阈值比较功能，则通过OUT1显示比较结果。当开启了反射输出功能时Out1与Reflex1相等。

9.2.4 HSCGetParam

此功能块主要用于读取当前高速计数器的设置参数
使用方法示例如下：假设要读取HSC00高速计数器的相关参数



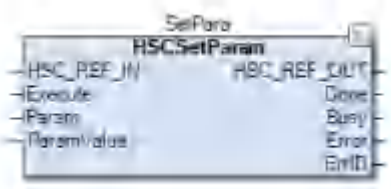
下表介绍了输入变量：

输入	类型	注释
HSC_REF_IN	HSC_REF	HSC 的参考，应与HSCSimple，HSCMain或HSCSpecialized的HSC_REF管脚相连
Execute	BOOL	上升沿时，功能块开始执行。下降沿，功能块复位。
Param	HSC_PARAMETER_TYPE	参考附录HSC_PARAMETER_TYPE型结构变量的介绍

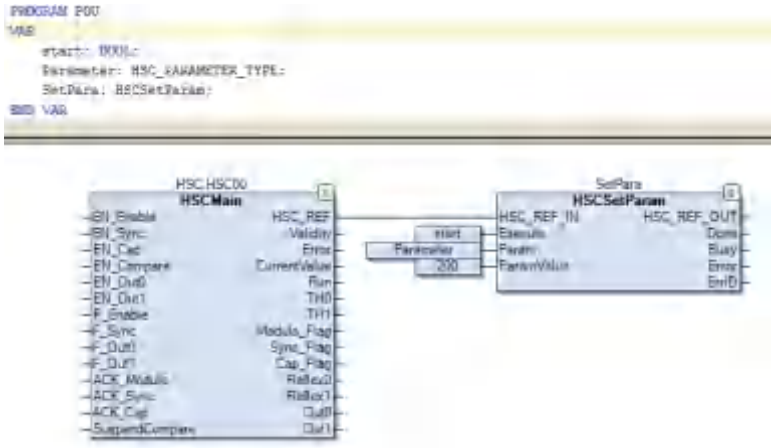
下表介绍了输出变量：

输出	类型	注释
HSC_REF_OUT	HSC_REF	HSC参考的输出，与HSC_REF_IN意义相同
Done	BOOL	TRUE 表示功能块执行完成。
Busy	BOOL	TRUE 表示功能正在进行。
Error	BOOL	TRUE 表示检测到错误
ErrID	HSC_ERR_TYPE	当 Error 为 TRUE 时代表检测到的错误代码，查询附录HSC_ERR_TYPE型结构变量的描述得到错误信息
ParamValue	DINT	已读取的参数值。

9.2.5 HSCSetParam



该功能块用于在程序中在线更改高速计数器的设置参数
使用方法示例如下：假设要设置HSC00高速计数器的参数



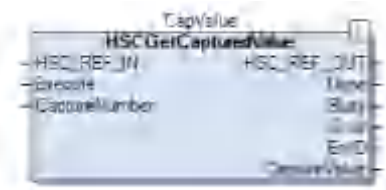
下表介绍了输入变量：

输入	类型	注释
HSC_REF_IN	HSC_REF	HSC 的参考，应与HSCSimple，HSCMain或HSCSpecialized的HSC_REF管脚相连
Execute	BOOL	升沿时，功能块开始执行。下降沿，功能块输出复位。
Param	HSC_PARAMETER_TYPE	要设置的参数。参考附录HSC_PARAMETER_TYPE型结构变量的介绍
ParamValue	DINT	准备要写入的数值。

下表介绍了输出变量：

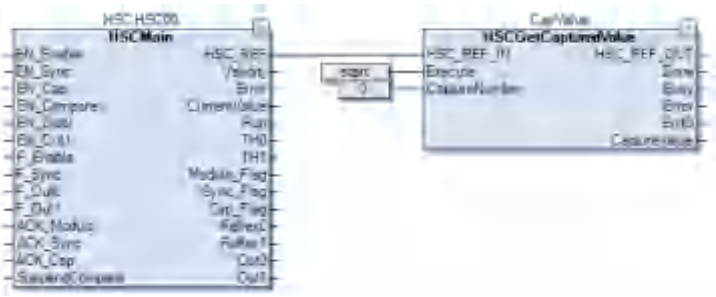
输出	类型	注释
HSC_REF_OUT	HSC_REF	HSC参考的输出，与HSC_REF_IN意义相同
Done	BOOL	TRUE 表示参数已成功写入。
Busy	BOOL	TRUE 表示功能块正在执行。
Error	BOOL	TRUE 表示检测到错误。

9.2.6 HSCGetCapturredValue



HSCGetCaptureValue功能块用来对高速计数器捕捉（CAP）后的数据进行读取。读取的对象包括PLC本体的MAIN型高速计数器捕捉的数据或TM200HSC206DT/F高速计数扩展模块捕捉的数据。

当需要读取HSC00通道捕捉到的数值时使用方法如下：



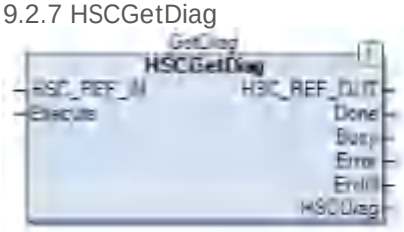
下表介绍了输入变量：

输入	类型	注释
HSC_REF_IN	HSC_REF	HSC 的参考，应与HSCSpecialized或HSCMain的HSC_REF管脚相连
Execute	BOOL	在上升沿上，启动功能块执行。 在下降沿上，功能块执行终止后，复位输出。
CaptureNumber	BYTE	捕捉寄存器的索引值：对于 Main类型始终为0，对于Specialized（HSC扩展模块）为0 或 1

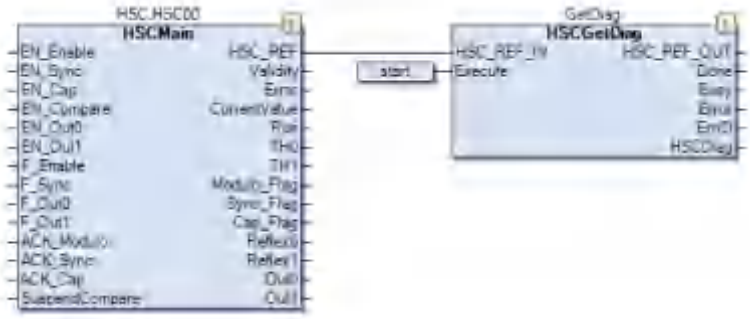
下表介绍了输出变量：

输出	类型	注释
HSC_REF_OUT	HSC_REF	HSC参考的输出，与HSC_REF_IN意义相同
Done	BOOL	TRUE 表示捕捉执行完成。
Busy	BOOL	TRUE 表示捕捉功能正在执行。
Error	BOOL	TRUE 表示检测到错误。
ErrID	HSC_ERR_TYPE	TRUE表示发生错误，请参考附录HSC_ERR_TYPE型结构变量的介绍
CaptureValue	DINT	捕捉到的高速计数器数值。

9.2.7 HSCGetDiag



HSCGetDiag功能块用来读取高速计数器运行中产生错误时的诊断代码。
当需要检测HSC00通道产生的错误时，使用方法如下：



下表介绍了输入变量：

输入	类型	注释
HSC_REF_IN	HSC_REF	HSC 的参考，应与HSCSimple，HSCMain或HSCSpecialized的HSC_REF管脚相连
Execute	BOOL	上升沿时启动功能块。 下降沿时功能块所有输出信息复位

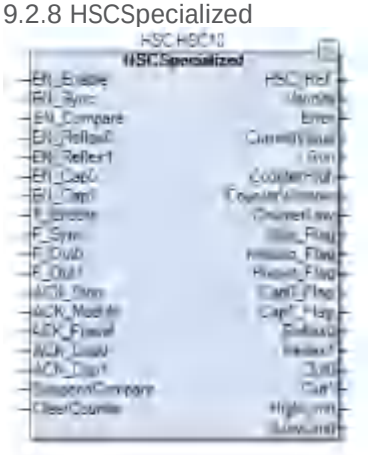
下表介绍了输出变量：

输出	类型	注释
HSC_REF_OUT	HSC_REF	HSC参考的输出，与HSC_REF_IN意义相同
Done	BOOL	TRUE 表示 HSCDiag 成功运行
Busy	BOOL	TRUE 表示功能块正在执行。
Error	BOOL	TRUE 表示检测到错误。
ErrID	HSC_ERR_TYPE	TRUE代表发生错误，请参考附录HSC_ERR_TYPE型结构变量的介绍
EXPERTDiag	DWORD	当 Done 为 TRUE 时可通过此值来判断相关硬件错误

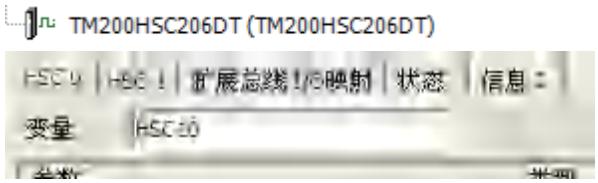
下表指示EXPERTDiag返回DWORD型变量的意义：

位	本体（HSCMain 或 HSCSimple）	TM200HSC206DT/F 模块 (HSCSpecialized)
0	–	检测到输入错误
1	–	检测到输出错误
2	–	–
3	–	–
4	–	自检时检测到内部错误
5	–	–
6	–	检测到总线错误
7	检测到配置错误	检测到配置错误
8	–	–
9	–	配置初始化
10	–	传感器电源电压低
11	–	执行器电源电压低
12	–	检测到 Output0 短路或过流
13	–	检测到 Output1 短路或过流
14	–	–
15	–	–

9.2.8 HSCSpecialized



HSCSpecialized是用来控制TM200HSC206DT/F 高速计数器扩展模块上的两路高速计数通道。与一般功能块不同的是HSCSpecialized功能块不允许随便起名，该名称必须与高速计数扩展模块内高速计数通道的变量名称链接起来。一般命名为HSC.变量名称。



下表介绍了输入变量：

输入	类型	注释
EN_Enable	BOOL	TRUE 允许通过物理输入来启用高速计数器。
EN_Sync	BOOL	TRUE 允许通过物理输入对高速计数器进行同步操作。
EN_Compare	BOOL	TRUE 表示启动数值比较功能。
EN_Reflex0	BOOL	TRUE 表示启用 Reflex0 和 Output0。
EN_Reflex1	BOOL	TRUE 表示启用 Reflex1 和 Output1。
EN_Cap0	BOOL	TRUE 表示开启0通道中的捕捉功能
EN_Cap1	BOOL	TRUE 表示开启1通道中的捕捉功能
F_Enable	BOOL	TRUE 表示强制运行高速计数器运行
F_Sync	BOOL	在上升沿上，强制进行下列计数模式中的计数功能的同步操作。要想启动高速计数器，至少要进行一次同步操作： •一次性计数器：当前值变为预设置并启动计数器 •模数回路计数器：当前值复位并启动计数器 •事件计数器：重新启动与时基对应的内部定时器 •周期计：启用周期计操作。

F_Out0	BOOL	TRUE 时强制Output0 为 TRUE。
F_Out1	BOOL	TRUE 时强制Output1 为 TRUE。
ACK_Stop	BOOL	在上升沿上，复位 Stop_Flag信号。
ACK_Modulo	BOOL	在上升沿上，复位 Modulo_Flag信号
ACK_Preset	BOOL	在上升沿上，复位 Preset_Flag信号
ACK_Cap0	BOOL	在上升沿时，复位 Cap0_Flag信号
ACK_Cap1	BOOL	在上升沿时，复位 Cap1_Flag信号
SuspendCompare	BOOL	TRUE时暂停数值比较功能 •CounterLow、CounterWindow、CounterHigh、Reflex0、Reflex1 输出保持其上一个状态 •物理输出 Output0 和 Output1 保持上一个状态
ClearCounter	BOOL	在上升沿上，将CurrentValue 清零

下表介绍输出变量：

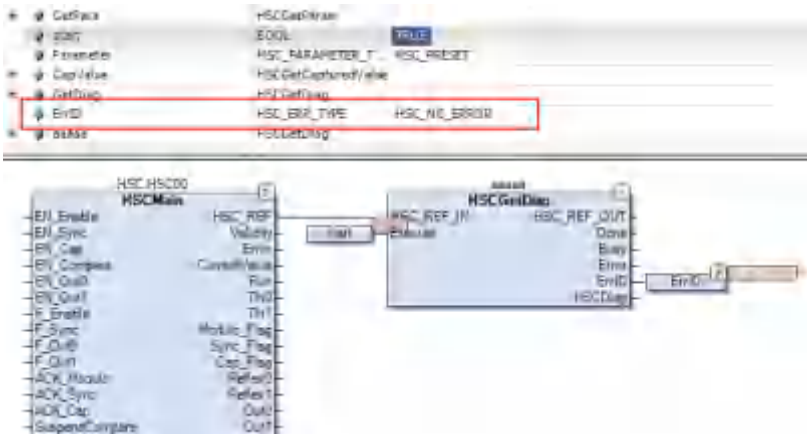
输出	类型	注释
HSC_REF	HSC_REF	HSC 的参考。功能块根据指定的通道自动生成的该通道HSC实例名称，每一个高速计数器都必须有一个唯一的名称与其对应，在使用其他高速计数相关功能块时必须将此变量赋给其他功能块的对应管脚，以告知哪一个高速计数器将要被操作。
Validity	BOOL	TRUE 表示该高速计数器通道有效
Error	BOOL	TRUE 表示检测到错误。 可使用 HSCGetDiag 功能块读取相关错误信息
CurrentValue	DINT	计数器的当前值。
Run	BOOL	TRUE表示高速计数器正在运行。 在一次性模式下，CurrentValue 达到 0 时，计数器自动停止,Run变为False
CounterHigh	BOOL	CurrentValue 大于阈值 1 值时为TRUE。
CounterWindow	BOOL	CurrentValue 介于阈值0与阈值1之间或等于阈值0或阈值1时为TRUE：
CounterLow	BOOL	CurrentValue 小于阈值 0 值时为TRUE。
Stop_Flag	BOOL	在一次性模式下CurrentValue 达到 0 时计数器停止，置为TRUE。
Modulo_Flag	BOOL	在进行模数回转时为TRUE： 模数回路计数器：当计数器达到模数时 自由大型计数器：当计数器达到其最大限制值时
Preset_Flag	BOOL	成功进行同步（SYNC）操作时置TRUE。
Cap0_Flag	BOOL	有新的捕捉值存储到0通道的寄存器时置TRUE。
Cap1_Flag	BOOL	有新的捕捉值存储到1通道的寄存器时置TRUE。

Reflex0	BOOL	Reflex0 的状态
Reflex1	BOOL	Reflex1 的状态
Out0	BOOL	Output0 的状态
Out1	BOOL	Output1 的状态
HighLimit	BOOL	自由大型计数器：在锁定限制模式下，当计数器达到 +2,147,483,647时置TRUE 事件计数器：事件记数超过计数器限制值时置TRUE，如果未达到限制值，则在下一个周期复位。 频率计:当输入频率信号超出可接收范围时置TRUE。 周期记: 周期超过计数器限制值时置TRUE，如果未达到限制值，则在下一个周期复位。 比率:输入 A 的频率到达限制值时置TRUE，输入 A 的频率正常时复位
LowLimit	BOOL	自由大型计数器：在锁定限制模式下：当计数器达到 -2,147,483,648 时置TRUE 事件计数器：在 5 毫秒周期内收到同步（SYNC）信号时置TRUE，如果未达到限制，则在下一个周期复位 周期计: 在 5 毫秒周期内收到同步（SYNC）信号时置TRUE，如果未达到限制，则在下一个周期复位 比率:输入 B 的频率太快时置TRUE，输入 B 的频率正常时复位

9.2.9高速计数器相关结构变量说明

（1）HSC_ERR_TYPE： 高速计数器功能块ErrID输出管脚对应的意义

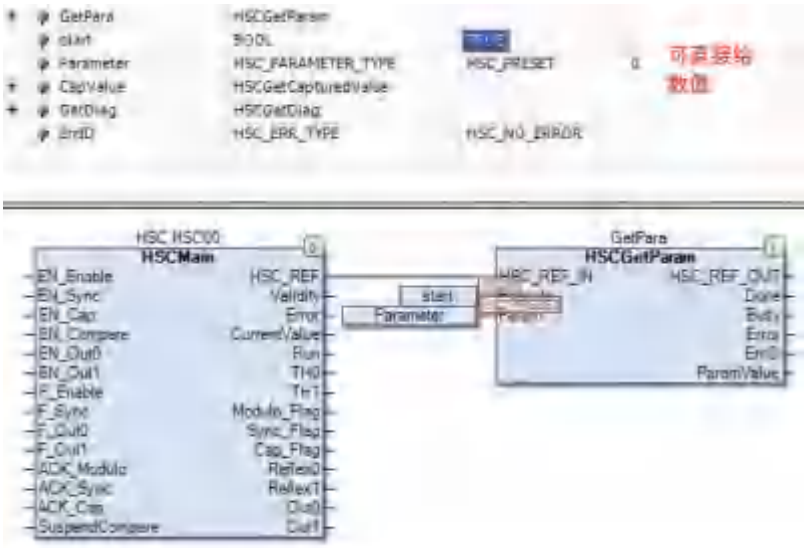
名称	值	描述
HSC_NO_ERROR	00 hex	没检测到错误
HSC_UNKNOWN	01 hex	HSC_REF 名称错误或该通道未配置
HSC_UNKNOWN_PARAMETER	02 hex	未知的参数名称。例如通过HSCSetParam设置参数时参数项目不存在
HSC_INVALID_PARAMETER	03 hex	设置参数不正确。例如通过HSCSetParam设置参数时该参数设置范围超限
HSC_COM_ERROR	04 hex	HSC 模块存在硬件通讯错误。
HSC_CAPTURE_NOT_CONFIGURED	05 hex	执行了捕捉命令但该通道并未配置捕捉功能



（2）HSC_PARAMETER_TYPE

执行HSCSetParam或HSCGetParam功能块时Param输入管脚的意义

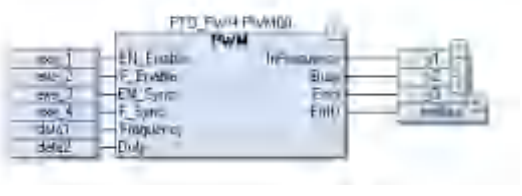
名称	值	描述
HSC_PRESET	00 hex	读取或设置当前指定通道的预设值。
HSC_MODULO	01 hex	读取或设置当前指定通道的模数值。
HSC_TIMEBASE	02 hex	读取或设置当前指定通道的时基值。
HSC_SLACK	03 hex	读取或设置当前指定通道的延迟值。（仅适用于 TM200HSC206DT/F 模块）。
HSC_CALIBRATION	04 hex	读取或设置当前指定通道的校准值（仅适用于 TM200HSC206DT/F 模块）。
HSC_THRESHOLD0	05 hex	读取或设置当前指定通道的的阈值 0
HSC_THRESHOLD1	06 hex	读取或设置当前指定通道的阈值 1
HSC_THRESHOLD2	07 hex	读取或设置当前指定通道的阈值 2
HSC_THRESHOLD3	08 hex	读取或设置当前指定通道的阈值 3

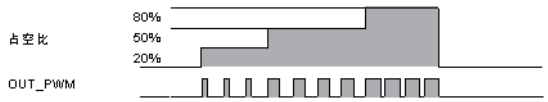


(3) HSC_PARAMETER_TYPE.HSC_TIMEBASE.

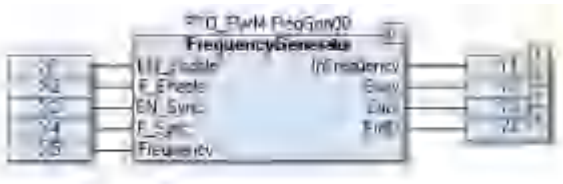
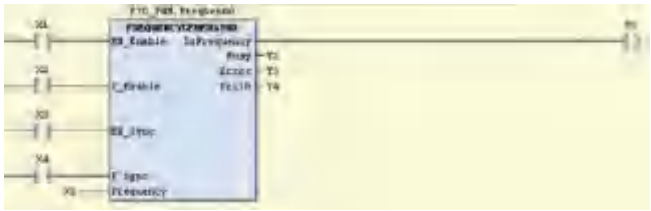
名称	值
HSC_100ms	00 hex
HSC_1s	01 hex
HSC_10s	02 hex
HSC_60s	03 hex

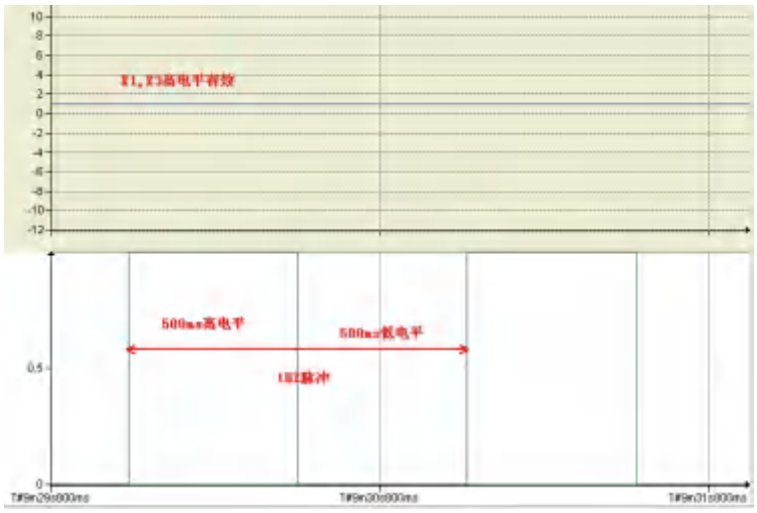
9.3.1 PWM

操作符	PWM 脉冲宽度调制
功能说明	脉冲宽度调制 (PWM) 功能生成具有可变频率和可变宽度（占空比）的方波信号。该模块使用一个内部时钟发生器，并在专用输出通道上提供输出信号。此输出信号可以直接命令轴的变速运动。目标频率始终为正。
图形	
管脚定义	输入　　类型　注释 EN_Enable　BOOL　使能，通过外部事件触发启用PWM输出，如果在通道0配置该功能，当I8=TRUE,PWM启用，如果在通道1配置该功能，I10=TRUE，PWM启用。 F_Enable　BOOL　通过内部变量使能PWM。 EN_SYNC　BOOL　允许通过硬件触发重新启动，如果通道0配置了该功能，I9=TRUE，PWM重新启动。如果通道1配置了该功能，I11=TRUE，PWM重新启动。 F_SYNC　BOOL　通过内部变量重启PWM。 Frequency　DWORD　PWM输出信号的频率（以0.1 Hz 为单位）。（范围：　200...10,000) Duty　　WORD　PWM 输出信号的占空比 (%)（范围：　最小 10(1%) 到最大 990(99%)）。 输出　　类型　注释 InFrequency　BOOL　TRUE = PWM以指定频率输出信号。 Busy　　BOOL　如果设置了启用命令，并且频率或占空比发生了更改，则输出TRUE。 　　如果设置了 InFrequency 或 Error，或者启用命令被复位，则复位为 FALSE。 Error　　BOOL　TRUE = 表示检测到一个错误。 ErrID　　PTOPWM_ERR_TYPE　错误代码类型。
管脚定义	PWM功能块输出管脚ErrID具体内容如下： 
变量声明	VAR exe_1: BOOL; exe_2: BOOL; exe_3: BOOL; exe_4: BOOL; data1: DWORD; data2: WORD; y1: BOOL; y2: BOOL; y3: BOOL; errdata: PTOPWM_ERR_TYPE; END_VAR
LD语言示例	
ST语言示例	PTO_PWM.PWM00(EN_Enable:=exe_1 , F_Enable:=exe_2 , EN_Sync:=exe_3 , F_Sync:=exe_4 , Frequency:=data1 , Duty:=data2 , InFrequency=>y1 , Busy=>y2 , Error=> y3, ErrID=>errdata);
CFC语言示例	

时序图	
基本概念	占空比=TP/T TP:脉冲宽度 T:脉冲周期 
使用限制	1.输出信号的占空比范围为1%~99% 2.PTO、PWM 和频率发生器功能使用相同的专用输出。在同一个通道上只能使用这三个功能的其中一个。通道 0 和通道 1 上可以使用不同的功能。
注意事项	必须通过将 F_Enable 设为 1，或者使用通过外部事件启用该功能，否则输出(OUT_PWM) 会保持为 0。

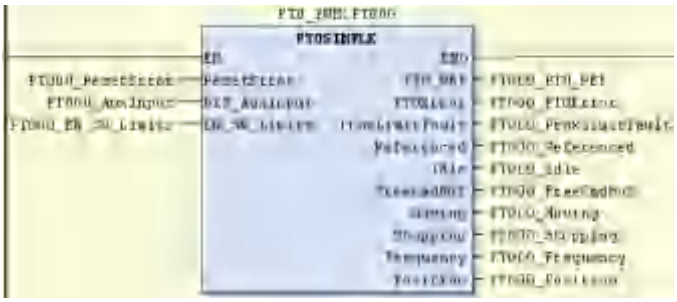
9.3.2 FrequencyGenerator

操作符	FrequencyGenerator 频率发生器
功能说明	此功能块按指定频率控制方波信号输出.频率发生器功能可以在专用输出通道上生成具有固定占空比 (50%) 的方波信号。频率可配置范围为 1 Hz 到 100 kHz，步长为 1 Hz
图形	
管脚定义	输入　　类型　注释 EN_Enable　BOOL　使能，如果已在频率发生器通道配置，可以通过外部事件触发频率发生器。 F_Enable　BOOL　通过软件变量使能频率发生器。 EN_SYNC　BOOL　如果已配置，可以通过外部事件重启频率发生器。 F_SYNC　BOOL　通过软件变量重启频率发生器。 Frequency　DWORD　频率发生器输出信号的频率（以 Hz 为单位）。（范围： 1...100,000) 输出　　类型　注释 InFrequency　BOOL　TRUE = 频率发生器以指定频率输出信号。 Busy　　BOOL　如果设置了启用命令，并且频率发生了更改，则设置为 TRUE。 　　如果设置了 InFrequency 或 Error，或者启用命令被复位，则复位为 FALSE。 Error　　BOOL　TRUE 表示检测到一个错误。 ErrID　检测到的错误的错误代码。
管脚定义	ErrID管脚具体内容如下： 
变量声明	VAR X1: BOOL; X2: BOOL; X3: BOOL; X4: BOOL; X5: DWORD; Y1: BOOL; Y2: BOOL; Y3: BOOL; Y4: PTOPWM_ERR_TYPE; END_VAR
CFC语言示例	
ST语言示例	PTO_PWM.FreqGen00(EN_Enable:=X1 , F_Enable:=X2 , EN_Sync:= X3, F_Sync:=X4 , Frequency:=X5 , InFrequency=>Y1 , Busy=> Y2, Error=> Y3, ErrID=>Y4);
LD语言示例	

时序图	
使用限制	PTO、PWM 和频率发生器功能使用相同的专用输出。在同一个通道上只能使用这三个功能的其中一个。通道 0 和通道 1 上可以使用不同的功能。
注意事项	1.输出脉冲是50%的固定占空比。 2.频率可配置范围为 1 Hz 到 100 kHz，步长为 1 Hz

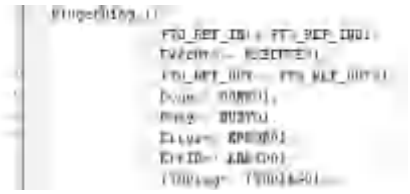
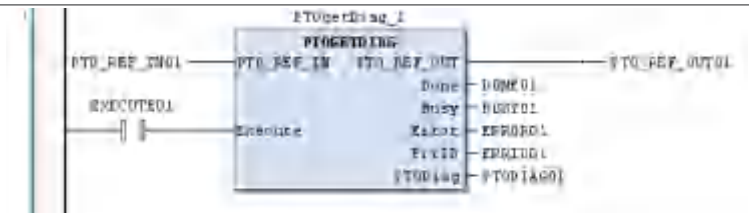
9.3.3 PTO-PTOSimple

操作符	PTOSimple
功能说明	脉冲输出管理(功能块)--该功能块用于PTO脉冲输出功能的管理;可查看脉冲输出轴的运行状态、错误、当前输出脉冲频率和位置等信息,可对故障复位,对辅助输入点管理;
图形	
管脚定义	输入 类型 注释 ResetError :BOOL; 复位错误,连接布尔型变量或直接地址(如%MX0.2),复位当前PTO通道的故障,上升沿有效; DIS_AuxInput :BOOL; 禁止辅助输入,禁止PTO轴配置的辅助输入点有效,如果在配置中没有选择辅助输入,该位不起作用; EN_SW_Limits :BOOL; 使能软限位,启用PTO轴配置中的软限位功能(软件上下限)有效; 输出 类型 注释 PTO_REF :PTO_REF; 轴通道参数,连接在其它PTO功能块的PTO_REF_IN输入端,用来识别当前PTO的轴通道参数; PTOError :BOOL; 轴通道错误,TRUE时表示检测到一个错误。具体错误可通过功能块PTOGetDiag读取; ProxLimitFault :BOOL; 限位触发错误,TRUE时表示运行中触发了限位开关(如果在PTO配置中没有启用PROX,该引脚无效); Referenced :BOOL; 原点标识,TRUE时表示PTO轴已标记参考点,可执行绝对位置命令,否则只能执行速度及相对位置命令; idle :BOOL; 命令发送状态,TRUE时表示PTO轴无命令发送,FALSE时表示正在发送命令, 需要与FreeCmdBuf命令缓冲组合使用,判断是否触发新命令; FreeCmdBuf :BOOL; 命令缓冲状态,TRUE时表示PTO轴无命令缓冲,FALSE时表示已有命令在缓冲区等待执行,不能发送新的命令; Moving :BOOL; 脉冲输出状态,TRUE时表示PTO轴正在移动,有脉冲输出; Stopping :BOOL; 停止过程中,TRUE时表示PTO轴正在停止过程中; Frequency :DWORD; PTO轴当前输出频率,<=PTO轴配置中的上限频率; Position :DINT; PTO轴当前脉冲位置,在相对模式中表示运动的相对距离（在开始新移动之前进行复位为0),在绝对模式中:用绝对模式表示运动的绝对位置;
变量声明	VAR PTO00_ResetError: BOOL; PTO00_EN_SW_Limits: BOOL; PTO00_AuxInput: BOOL; PTO00_PTO_REF: PTO_REF; PTO00_PTOError: BOOL; PTO00_ProxLimitFault: BOOL; PTO00_Referenced: BOOL; PTO00_idle: BOOL; PTO00_FreeCmdBuf: BOOL; PTO00_Moving: BOOL; PTO00_Stopping: BOOL; PTO00_Frequency: DWORD; PTO00_Position: DINT; END_VAR
CFC语言示例	
ST语言示例	PTO_PWM.PTO00(ResetError:= PTO00_ResetError, DIS_AuxInput:= PTO00_AuxInput, EN_SW_Limits:= PTO00_EN_SW_Limits, PTO_REF=> PTO00_PTO_REF, PTOError=> PTO00_PTOError, ProxLimitFault=> PTO00_ProxLimitFault, Referenced=> PTO00_Referenced, Idle=> PTO00_idle, FreeCmdBuf=> PTO00_FreeCmdBuf, Moving=> PTO00_Moving, Stopping=> PTO00_Stopping, Frequency=> PTO00_Frequency, Position=> PTO00_Position);

LD语言示例	
时序图	
使用限制	该功能块必须在MAST中调用才有效;
注意事项	1.PTOSimple功能块不需要在变量声明中实例化,它的实例化名称为PTO轴配置的变量名,如本例中使用的PTO_PWM.PTO00; 2.Idle命令空闲和FreeCmdBuf缓冲区空闲信号在组合使用时,实际只要在缓冲区空闲时就可以触发新的命令; 3.常用引脚为ResetError,PTO_REF,PTOError,Referenced,Moving,Frequency和Position,其它可悬空或删除; 4.只有Referenced=TRUE时,才可以触发绝对位置命令,否则报错PTOError; 5.一般用Moving的下降沿来判断当前运动的停止;

9.3.3 PTO-PTOGetDiag

操作符	PTOGetDiag		
功能说明	此功能块可返回检测到的PTO错误的详细信息。由应用库PTO/PWM.Library中Administrative提供。 该管理类型的功能块按照先入先出的顺序被执行请求。功能块的执行状态可通过输出管脚Done、Busy、Error的状态来判断。该功能块的执行可能会被延时若干个循环周期，该延迟取决于任务循环周期和请求目前所在执行队列的位置。		
图形			
管脚定义	输入 类型 注释 PTO_REF_IN PTO_REF 对PTO的参考。连接到PTOSimple的PTO_REF,或管理或运动输出引脚功能块的PTO_REF_OUT。 Execute BOOL 在上升沿启动功能块执行。 在下降沿时，则在其执行终结时，复位功能块的输出。 输出 类型 注释 PTO_REF_OUT PTO_RE 对PTO的参考。连接到管理或运动输出引脚功能块的PTO_REF_IN输入引脚。 Done BOOL TRUE = 表示 PTODiag 有效。功能块执行结束。 Busy BOOL TRUE = 表示功能块执行正在进行中。 Error BOOL TRUE = 表示检测到一个错误。功能块执行结束。 ErrID PTOPWM_ERR_TYPE 当 Error 为 TRUE 时：检测到的错误的类型。 PTODiag DWORD 当 Done 为 TRUE 时：诊断值有效（请参阅下表）。		
错误代码	枚举器 NO_ERROR PTO_UNKNOW_REF PTO_UNKNOW_PARAMETER PTO_INVALID_PARAMETER PTO_COM_ERROR PTO_AXIS_ERROR PTO_CMD_ERROR PTO_LIMIT_FAULT	值（16进制） 00 01 02 03 04 05 06 07	说明 未检测到错误 未知轴参考或配置不当的轴。 未知参数类型。 用于所请求移动的无效参数值或不正确的参数值组合。 检测到 PTO 接口通讯错误。 检测到轴错误（例如状态机无效）。 缓冲区已满。 回归模式中出现近似错误。
PTODiag诊断值	DWORD值 0..3 4 5,6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	含义 未使用 检测到内部错误 未使用 检测到配置错误 未使用 检测到近似限制错误 命令缓冲区已满 检测到缓冲区速度错误 轴未被引用 回归近似被禁用 FastPTO 停止例外 FastPTO 重新配置 FastPTO 过量 驱动器未就绪（辅助输入 DriveReady 为 FALSE） 检测到软件上限 检测到软件下限 没有为 FastPTO 分配触发引脚 检测到回归错误 频率无效 加速度无效 减速度无效 命令被拒绝 距离无效 位置无效 回归模式无效 方向无效 反向 检测到配置文件错误	

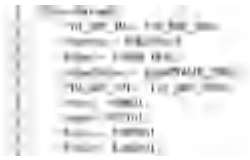
变量声明	VAR PTOgetDiag_1: PTOGetDiag; PTO_REF_IN01: PTO_REF; EXECUTE01: BOOL; PTO_REF_OUT01: PTO_REF; DONE01: BOOL; BUSY01: BOOL; ERROR01: BOOL; ERRID01: PTOPWM_ERR_TYPE; PTODIAG01: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	
使用限制	
注意事项	· 为了在功能块执行过程中获取功能块的最新执行状态,必须重复调用该功能块。 · 当功能块在执行过程中,不得改变PTO_REF_IN的输入。 · 在不同的任务中,不能使用相同的功能块实例。 · 在同一个应用(Application)中，如果有太多的管理类型或者运动类型的功能块被同时调用，该功能块将会在执行结束之后输出一个COM_ERROR错误类型。（意思指请求队列已满）
	输入变量管理： · 该功能块在 Execute 输入的上升沿上启动。 · 此时无需对输入变量进行任何进一步的修改。 · 按照 IEC 61131-3 标准，如果功能块有任何变量输入缺失（即断开或未连接），则使用上一次调用功能块实例的值。在此情 况下，第一次调用时将应用初始配置值。因此，功能块最好始终带有特定于其输入的已知值，这样有助于消除调试程序的麻烦。对于 HSC 和 PTO 功能块，最好只使用一次实例，且该实例必须位于主任务中。
	输出变量管理： ·Done、InVelocity 或 InFrequency 输出与 Busy、CommandAborted 和 Error 输出相互排斥：在一个功能块上，只能有一个输出为 TRUE。如果 Execute 输入为 TRUE，则其中一个输出为 TRUE。 ·在 Execute 输入的上升沿，会设置 Busy 输出。在功能块执行过程中，此 Busy 输出保持已设置状态，并在某一其他输出（Done、InVelocity、InFrequency、CommandAborted 和 Error）的上升沿复位。 ·当功能块成功执行完毕时，会设置 Done、InVelocity 或 InFrequency 输出。 ·当功能块的执行被另一功能块中断时，则改为设置 CommandAborted 输出。 ·当功能块的执行由于检测到错误而结束时，则会设置 Error 输出，并通过 ErrId 输出给出检测到的错误编号。 ·Done、InVelocity、InFrequency、Error、ErrID 和 CommandAborted 输出在 Execute 的下降沿复位。如果 Execute 输入在执行完成之前复位，则在执行结束时，这些输出的设置状态将持续一个任务循环。 ·当功能块的某个实例在完成之前收到新的 Execute 时，则对于以前的操作，功能块不返回任何反馈，比如 Done。
	错误处理： 所有功能块都有 2 个输出，可以报告在执行功能块期间检测到的错误。 ·检测到错误时，Error = TRUE。 ·ErrID 在 Error = TRUE 时返回检测到的错误 ID。

9.3.3 PTO-PTOGetParam

操作符	PTOGetParam
功能说明	此功能块返回 PTO 轴的指定参数的值。由应用库PTO/PWM.Library中Administrative提供。 该管理类型的功能块按照先入先出的顺序被执行请求。功能块的执行状态可通过输出管脚Done、Busy、Error的状态来判断。该功能块的执行可能会被延时若干个循环周期，该延迟取决于任务循环周期和请求目前所在执行队列的位置。
图形	
管脚定义	输入 类型 注释 PTO_REF_IN PTO_REF 对PTO的参考。连接到PTOSimple的PTO_REF,或管理或运动输出引脚功能块的PTO_REF_OUT。 Execute BOOL 在上升沿启动功能块执行。 在下降沿时，则在其执行终结时，复位功能块的输出。 Param PTO_PARAMETER_TYPE 要读取的参数 输出 类型 注释 PTO_REF_OUT PTO_RE 对PTO的参考。连接到管理或运动输出引脚功能块的PTO_REF_IN输入引脚。 Done BOOL TRUE = 表示 PTOGetParam 有效。功能块执行结束。 Busy BOOL TRUE = 表示功能块执行正在进行中。 Error BOOL TRUE = 表示检测到一个错误。功能块执行结束。 ErrID PTOPWM_ERR_TYPE 当 Error 为 TRUE 时：检测到的错误的类型。 ParamValue DWORD 当 Done 为 TRUE 时：参数值有效（请参阅下表）。
Param值	枚举器 / 值（16进制） / 说明 PTO_START_FREQUENCY / 00 / PTO 运动的启动速度。 PTO_STOP_FREQUENCY / 01 / PTO 运动的停止速度。 PTO_EMY_DEC / 02 / PTO 紧急停止的减速度。
错误代码	枚举器 / 值（16进制） / 说明 NO_ERROR / 00 / 未检测到错误 PTO_UNKNOW_REF / 01 / 未知轴参考或配置不当的轴。 PTO_UNKNOW_PARAMETER / 02 / 未知参数类型。 PTO_INVALID_PARAMETER / 03 / 用于所请求移动的无效参数值或不正确的参数值组合。 PTO_COM_ERROR / 04 / 检测到 PTO 接口通讯错误。 PTO_AXIS_ERROR / 05 / 检测到轴错误（例如状态机无效）。 PTO_CMD_ERROR / 06 / 缓冲区已满。 PTO_LIMIT_FAULT / 07 / 回归模式中出现近似错误。
变量声明	VAR PTOGetPara01: PTOGetParam; PTO_REF_IN01: PTO_REF; EXECUTE01: BOOL; PTO_REF_OUT01: PTO_REF; DONE01: BOOL; BUSY01: BOOL; ERROR01: BOOL; ERRID01: PTOPWM_ERR_TYPE; PARAM01: PTO_PARAMETER_TYPE; PARAMVALUE01: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	

使用限制	
注意事项	·为了在功能块执行过程中获取功能块的最新执行状态,必须重复调用该功能块。 ·当功能块在执行过程中,不得改变PTO_REF_IN的输入。 ·在不同的任务中,不能使用相同的功能块实例。 ·在同一个应用(Application)中，如果有太多的管理类型或者运动类型的功能块被同时调用，该功能块将会在执行结束之后输出一个COM_ERROR错误类型。〔意思指请求队列已满〕
	输入变量管理： ·该功能块在 Execute 输入的上升沿上启动。 ·此时无需对输入变量进行任何进一步的修改。 ·按照 IEC 61131-3 标准，如果功能块有任何变量输入缺失（即断开或未连接），则使用上一次调用功能块实例的值。在此情 况下，第一次调用时将应用初始配置值。因此，功能块最好始终带有特定于其输入的已知值，这样有助于消除调试程序的麻烦。对于 HSC 和 PTO 功能块，最好只使用一次实例，且该实例必须位于主任务中。
	输出变量管理： ·Done、InVelocity 或 InFrequency 输出与 Busy、CommandAborted 和 Error 输出相互排斥：在一个功能块上，只能有一个输出为 TRUE。如果 Execute 输入为 TRUE，则其中一个输出为 TRUE。 ·在 Execute 输入的上升沿，会设置 Busy 输出。在功能块执行过程中，此 Busy 输出保持已设置状态，并在某一其他输出（Done、InVelocity、InFrequency、CommandAborted 和 Error）的上升沿复位。 ·当功能块成功执行完毕时，会设置 Done、InVelocity 或 InFrequency 输出。 ·当功能块的执行被另一功能块中断时，则改为设置 CommandAborted 输出。 ·当功能块的执行由于检测到错误而结束时，则会设置 Error 输出，并通过 ErrId 输出给出检测到的错误编号。 ·Done、InVelocity、InFrequency、Error、ErrID 和 CommandAborted 输出在 Execute 的下降沿复位。如果 Execute 输入在执行完成之前复位，则在执行结束时，这些输出的设置状态将持续一个任务循环。 ·当功能块的某个实例在完成之前收到新的 Execute 时，则对于以前的操作，功能块不返回任何反馈，比如 Done。
	错误处理： 所有功能块都有 2 个输出，可以报告在执行功能块期间检测到的错误。 ·检测到错误时，Error = TRUE。 ·ErrID 在 Error = TRUE 时返回检测到的错误 ID。

9.3.3 PTO-PTOSetParam

操作符	PTOSetParam
功能说明	此功能块可修改 PTO 轴的指定参数的值。由应用库PTO/PWM.Library中Administrative提供。 该管理类型的功能块按照先入先出的顺序被执行请求。功能块的执行状态可通过输出管脚Done、Busy、Error的状态来判断。该功能块的执行可能会被延时若干个循环周期，该延迟取决于任务循环周期和请求目前所在执行队列的位置。
图形	
管脚定义	输入 类型 注释 PTO_REF_IN PTO_REF 对PTO的参考。连接到PTOSimple的PTO_REF,或管理或运动输出引脚功能块的PTO_REF_OUT。 Execute BOOL 在上升沿启动功能块执行。 在下降沿时，则在其执行终结时，复位功能块的输出。 Param PTO_PARAMETER_TYPE 要设置的值。 Param_Value DWORD 要写入的参数值。 输出 类型 注释 PTO_REF_OUT PTO_RE 对PTO的参考。连接到管理或运动输出引脚功能块的PTO_REF_IN输入引脚。 Done BOOL TRUE = 表示 PTOSetParam 有效。功能块执行结束。 Busy BOOL TRUE = 表示功能块执行正在进行中。 Error BOOL TRUE = 表示检测到一个错误。功能块执行结束。 ErrID PTOPWM_ERR_TYPE 当 Error 为 TRUE 时：检测到的错误的类型。
Param值	枚举器 / 值（16进制） / 说明 PTO_START_FREQUENCY / 00 / PTO 运动的启动速度。 PTO_STOP_FREQUENCY / 01 / PTO 运动的停止速度。 PTO_EMY_DEC / 02 / PTO 紧急停止的减速度。
错误代码	枚举器 / 值（16进制） / 说明 NO_ERROR / 00 / 未检测到错误 PTO_UNKNOW_REF / 01 / 未知轴参考或配置不当的轴。 PTO_UNKNOW_PARAMETER / 02 / 未知参数类型。 PTO_INVALID_PARAMETER / 03 / 用于所请求移动的无效参数值或不正确的参数值组合。 PTO_COM_ERROR / 04 / 检测到 PTO 接口通讯错误。 PTO_AXIS_ERROR / 05 / 检测到轴错误（例如状态机无效）。
变量声明	VAR PTOSetParam01: PTOSetParam; PTO_REF_IN01: PTO_REF; EXECUTE01: BOOL; PTO_REF_OUT01: PTO_REF; DONE01: BOOL; BUSY01: BOOL; ERROR01: BOOL; ERRID01: PTOPWM_ERR_TYPE; PARAM_IN01: PTO_PARAMETER_TYPE; PARAMVALUE_IN01: DWORD; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	

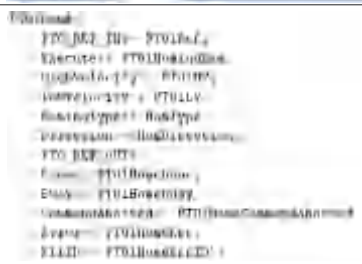
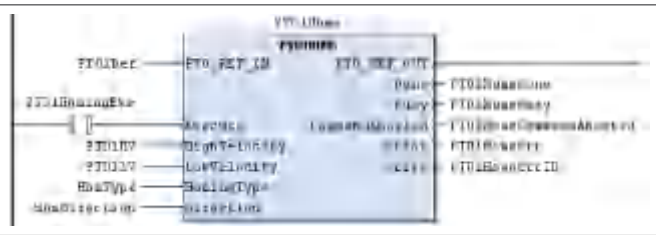
使用限制	
注意事项	·为了在功能块执行过程中获取功能块的最新执行状态,必须重复调用该功能块。 ·当功能块在执行过程中,不得改变PTO_REF_IN的输入。 ·在不同的任务中,不能使用相同的功能块实例。 ·在同一个应用(Application)中，如果有太多的管理类型或者运动类型的功能块被同时调用，该功能块将会在执行结束之后输出一个COM_ERROR错误类型。〔意思指请求队列已满〕
	输入变量管理： ·该功能块在 Execute 输入的上升沿上启动。 ·此时无需对输入变量进行任何进一步的修改。 ·按照 IEC 61131-3 标准，如果功能块有任何变量输入缺失（即断开或未连接），则使用上一次调用功能块实例的值。在此情 况下，第一次调用时将应用初始配置值。因此，功能块最好始终带有特定于其输入的已知值，这样有助于消除调试程序的麻烦。对于 HSC 和 PTO 功能块，最好只使用一次实例，且该实例必须位于主任务中。
	输出变量管理： ·Done、InVelocity 或 InFrequency 输出与 Busy、CommandAborted 和 Error 输出相互排斥：在一个功能块上，只能有一个输出为 TRUE。如果 Execute 输入为 TRUE，则其中一个输出为 TRUE。 ·在 Execute 输入的上升沿，会设置 Busy 输出。在功能块执行过程中，此 Busy 输出保持已设置状态，并在某一其他输出（Done、InVelocity、InFrequency、CommandAborted 和 Error）的上升沿复位。 ·当功能块成功执行完毕时，会设置 Done、InVelocity 或 InFrequency 输出。 ·当功能块的执行被另一功能块中断时，则改为设置 CommandAborted 输出。 ·当功能块的执行由于检测到错误而结束时，则会设置 Error 输出，并通过 ErrId 输出给出检测到的错误编号。 ·Done、InVelocity、InFrequency、Error、ErrID 和 CommandAborted 输出在 Execute 的下降沿复位。如果 Execute 输入在执行完成之前复位，则在执行结束时，这些输出的设置状态将持续一个任务循环。 ·当功能块的某个实例在完成之前收到新的 Execute 时，则对于以前的操作，功能块不返回任何反馈，比如 Done。
	错误处理： 所有功能块都有 2 个输出，可以报告在执行功能块期间检测到的错误。 ·检测到错误时，Error = TRUE。 ·ErrID 在 Error = TRUE 时返回检测到的错误 ID。

9.3.3 PTO-PTOSetPosition

操作符	PTOSetPosition
功能说明	此功能块可修改 PTO 轴的位置值。由应用库PTO/PWM.Library中Administrative提供。 该管理类型的功能块按照先入先出的顺序被执行请求。功能块的执行状态可通过输出管脚Done、Busy、Error的状态来判断。该功能块的执行可能会被延时若干个循环周期，该延迟取决于任务循环周期和请求目前所在执行队列的位置。
图形	
管脚定义	输入 类型 注释 PTO_REF_IN PTO_REF 对PTO的参考。连接到PTOSimple的PTO_REF,或管理或运动输出引脚功能块的PTO_REF_OUT。 Execute BOOL 在上升沿启动功能块执行。 在下降沿时，则在其执行终结时，复位功能块的输出。 Position DINT 位置的值。 输出 类型 注释 PTO_REF_OUT PTO_RE 对PTO的参考。连接到管理或运动输出引脚功能块的PTO_REF_IN输入引脚。 Done BOOL TRUE = PTOSetPosition有效。功能块执行结束。 Busy BOOL TRUE = 表示功能块执行正在进行中。 Error BOOL TRUE = 表示检测到一个错误。功能块执行结束。 ErrID PTOPWM_ERR_TYPE 当 Error 为 TRUE 时： 检测到的错误的类型。
错误代码	枚举器 / 值（16进制） / 说明 NO_ERROR / 00 / 未检测到错误 PTO_UNKNOW_REF / 01 / 未知轴参考或配置不当的轴。 PTO_UNKNOW_PARAMETER / 02 / 未知参数类型。 PTO_INVALID_PARAMETER / 03 / 用于所请求移动的无效参数值或不正确的参数值组合。 PTO_COM_ERROR / 04 / 检测到 PTO 接口通讯错误。 PTO_AXIS_ERROR / 05 / 检测到轴错误（例如状态机无效）。 PTO_CMD_ERROR / 06 / 缓冲区已满。 PTO_LIMIT_FAULT / 07 / 回归模式中出现近似错误。
变量声明	VAR PTOHOME01: PTOHome; PTO_REF_IN01: PTO_REF; EXECUTE01: BOOL; HIGHVELO01: DWORD; LOWVELO01: DWORD; PTOSETPOSITION01: PTOSetPosition; EXECUTE02: BOOL; POSITIONVALUE: DINT; DONE01: BOOL; BUSY01: BOOL; COMMANDABORTED01: BOOL; ERR01: BOOL; ERRID01: PTOPWM_ERR_TYPE; PTO_REF_OUT01: PTO_REF; DONE02: BOOL; BUSY02: BOOL; ERROR02: BOOL; ERRID02: PTOPWM_ERR_TYPE; HOMINGTYPE01: PTO_HOMING_TYPE; DIRECTION01: PTO_DIRECTION; END_VAR
CFC语言示例	
ST语言示例	

LD语言示例	
使用限制	
注意事项	·为了在功能块执行过程中获取功能块的最新执行状态,必须重复调用该功能块。 ·当功能块在执行过程中,不得改变PTO_REF_IN的输入。 ·在不同的任务中,不能使用相同的功能块实例。 ·在同一个应用(Application)中，如果有太多的管理类型或者运动类型的功能块被同时调用，该功能块将会在执行结束之后输出一个COM_ERROR错误类型。（意思指请求队列已满）
	输入变量管理： ·该功能块在 Execute 输入的上升沿上启动。 ·此时无需对输入变量进行任何进一步的修改。 ·按照 IEC 61131-3 标准，如果功能块有任何变量输入缺失（即断开或未连接），则使用上一次调用功能块实例的值。在此情 况下，第一次调用时将应用初始配置值。因此，功能块最好始终带有特定于其输入的已知值，这样有助于消除调试程序的麻烦。对于 HSC 和 PTO 功能块，最好只使用一次实例，且该实例必须位于主任务中。
	输出变量管理： ·Done、InVelocity 或 InFrequency 输出与 Busy、CommandAborted 和 Error 输出相互排斥：在一个功能块上，只能有一个输出为 TRUE。如果 Execute 输入为 TRUE，则其中一个输出为 TRUE。 ·在 Execute 输入的上升沿，会设置 Busy 输出。在功能块执行过程中，此 Busy 输出保持已设置状态，并在某一其他输出（Done、InVelocity、InFrequency、CommandAborted 和 Error）的上升沿复位。 ·当功能块成功执行完毕时，会设置 Done、InVelocity 或 InFrequency 输出。 ·当功能块的执行被另一功能块中断时，则改为设置 CommandAborted 输出。 ·当功能块的执行由于检测到错误而结束时，则会设置 Error 输出，并通过 ErrId 输出给出检测到的错误编号。 ·Done、InVelocity、InFrequency、Error、ErrID 和 CommandAborted 输出在 Execute 的下降沿复位。如果 Execute 输入在执行完成之前复位，则在执行结束时，这些输出的设置状态将持续一个任务循环。 ·当功能块的某个实例在完成之前收到新的 Execute 时，则对于以前的操作，功能块不返回任何反馈，比如 Done。
	错误处理： 所有功能块都有 2 个输出，可以报告在执行功能块期间检测到的错误。 ·检测到错误时，Error = TRUE。 ·ErrID 在 Error = TRUE 时返回检测到的错误 ID。

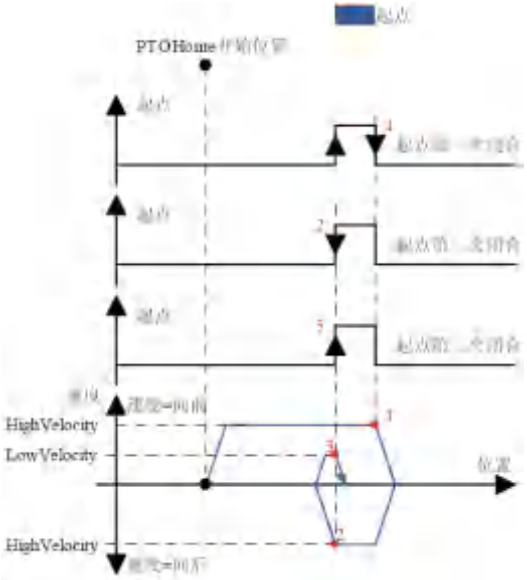
9.3.3 PTO-PTOHOME

操作符	PTOHOME
功能说明	该功能块用于将轴设置到参考位置
图形	
管脚定义	输入： PTO_REF_IN：PTO_REF类型， 具体PTO轴的参考。在PTOSimple的PTO_REF输出引脚上生成该变量，或直接连接在PTOSimple的PTO_REF输出引脚上，或连接在其它PTO功能块的PTO_REF_OUT引脚上(如果该功能块的PTO_REF_IN已经确定时)。 Execute：BOOL类型；如果开始执行PTOHome功能块后变为FALSE，则在PTOHome执行完成时，复位PTOHome功能块的所有输出。 HighVelocity：DWORD类型，最大初始回归搜索速度值。 LowVelocity：DWORD类型，最大最终回归接近速度值。 两种速度值的关系：0<LowVelocityjHighVelocityj最大频率。 HomingType：PTO_HOMING_TYPE类型，定义回归动作类型，共有0~ 5六种类型，细节请参考回归类型介绍部分。 DIRECTION：PTO_DIRECTION类型，PTOHome执行时移动的方向，0=正向，1=反向，2=保持上一次的运行方向。 输出： PTO_REF_OUT：PTO_REF类型；具体PTO轴的参考输出，可以连接到其它PTO功能块的PTO_REF_IN上或关联一个变量。 Done：BOOL类型。TRUE = 表示PTOHome功能块完成。代表PTOHome功能块执行正常结束。 Busy：BOOL类型。TRUE = 表示PTOHome功能块正在执行中。 CommandAborted：BOOL类型。TRUE = 表示PTOHome功能块因为另一个移动命令而中止。PTOHome功能块执行结束。 Error：BOOL类型。TRUE = 表示检测到一个错误。PTOHome功能块执行结束。 ErrorID：PTOPWM_ERR_TYPE类型。当Error为TRUE时，ErrorID存放检测到的错误的类型。共有0~ 7八种错误类型。
变量声明	VAR PTO1Home: PTOHome; PTO1Ref: PTO_REF; PTO1HomingExe: BOOL; PTO1HV: DWORD; PTO1LV: DWORD; HomType: PTO_HOMING_TYPE; HomDirection: PTO_DIRECTION; PTO1HomeDone: BOOL; PTO1HomeBusy: BOOL; PTO1HomeCommandAborted: BOOL; PTO1HomeErr: BOOL; PTO1HomeErrID: PTOPWM_ERR_TYPE; END_VAR
CFC语言示例	
ST语言示例	
LD语言示例	

时序图	参考具体的回归类型介绍部分。
使用限制	参考具体的回归类型介绍部分。
注意事项	不同的回归类型，要求对起点输入及限位Prox输入进行配置，否则功能块会有错误输出。 不同的回归类型，对输入的信号类型有不同要求，请参考具体的回归类型介绍部分。

六种寻原点方式
1. 短凸轮(Short Cam)
描述
PTOHome功能块的一种运行模式，用于将轴设置到参考位置。
短凸轮回归模式仅使用一个输入点：起点输入(凸轮)，PTO0使用I9，PTO1使用I11。

下图说明短凸轮回归模式，HomingType=0(PTO_SHORT_CAM,短凸轮) Direction=0(PTO_POSITIVE)，正向即向前：



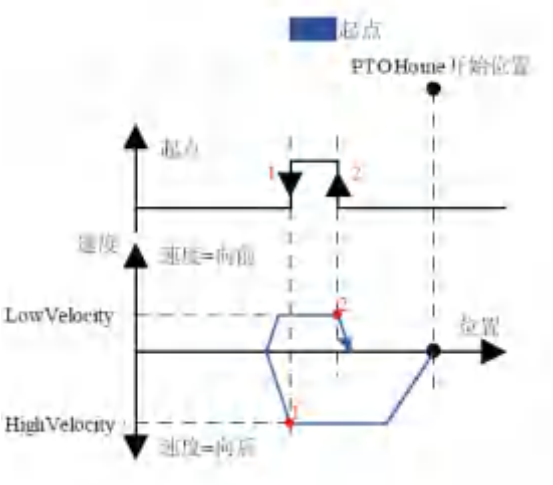
PTOHome相关参数：
HighVelocity最大初始回归搜索速度的值
LowVelocity最大最终回归接近速度的值

详细说明：在起点低电平期间启动，以HighVelocity速度向前运行；第一次遇到起点上升沿，不改变速度及方向；第一次遇到起点下降沿，反向，以HighVelocity速度向后运行；第二次遇到起点下降沿，再次反向，以LowVelocity速度向前运行；第三次遇到起点上升沿后停止。

报错情况：
1、PTOHome在Execute不可以在起点的高电平期间内执行，否则PTOHome产生错误输出。
2、速度值关系：0<LowVelocity<=HighVelocity 最大频率，两个速度均不能为0，否则PTOHome产生错误输出。
3、正向运行时超过PTO配置窗口中的软件上限时，PTOHome产生错误输出。
4、PTO配置窗口中的启用了辅助输入中的PROX，相应的输入高电平时PTOSimple产生错误输出。
5、如果在PTO配置窗口中禁用起点Aux，编译时会报错，错误信息"XXX使用未授权"。
以上错误会引起PTOSimple功能块产生错误输出，需要在PTOSimple功能块ResetError处复位消除错误。

注意：
1、PTOHome中的LowVelocity值不宜过大，其大小决定回归的定位精度。
2、PTOHome中的Direction会影响PTOHome的动作行为。
3、PTOHome开始执行的位置、PTOHome的Direction及起点的位置的关系决定是否能正确寻到原点。

下图说明短凸轮回归模式，HomingType=0(PTO_SHORT_CAM,短凸轮) Direction=1(PTO_NEGATIVE)，反向即向后：



PTOHome相关参数：
HighVelocity最大初始回归搜索速度的值
LowVelocity最大最终回归接近速度的值

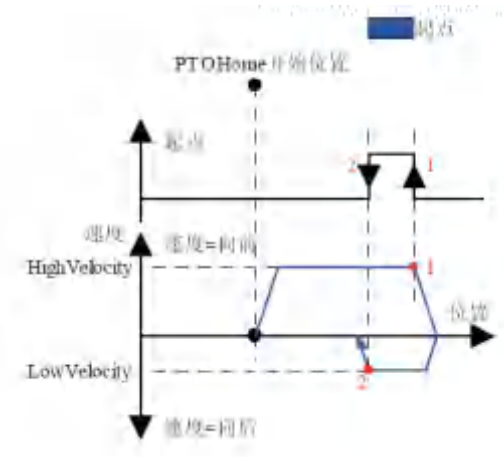
详细说明：在起点低电平期间启动，以HighVelocity速度向后运行；第一次遇到起点上升沿，不改变速度及方向；第一次遇到起点下降沿反向，以LowVelocity速度向前运行；第二次遇到起点上升沿后停止。

报错情况：
1、PTOHome在Execute不可以在起点的高电平期间内执行，否则PTOHome产生错误输出。
2、速度值关系：0<LowVelocity<=HighVelocity 最大频率，两个速度均不能为0，否则PTOHome产生错误输出。
3、正向运行时超过PTO配置窗口中的软件上限时，PTOHome产生错误输出。
4、PTO配置窗口中的启用了辅助输入中的PROX，相应的输入高电平时PTOSimple产生错误输出。
以上错误会引起PTOSimple功能块产生错误输出，需要在PTOSimple功能块ResetError处复位消除错误。

注意：
1、PTOHome中的LowVelocity值不宜过大，其大小决定回归的定位精度。
2、PTOHome中的Direction会影响PTOHome的动作行为。
3、PTOHome开始执行的位置、PTOHome的Direction及起点的位置的关系决定是否能正确寻到原点。

六种寻原点方式
2. 正向长凸轮(Long Cam Positive)
描述
PTOHome功能块的一种运行模式，用于将轴设置到参考位置。
正向长凸轮回归模式仅使用一个输入点：起点输入(凸轮)，PTO0使用I9，PTO1使用I11。

下图说明正向长凸轮回归模式，HomingType=1(PTO_LONG_CAM_POS,正向长凸轮) Direction=0(PTO_POSITIVE)，



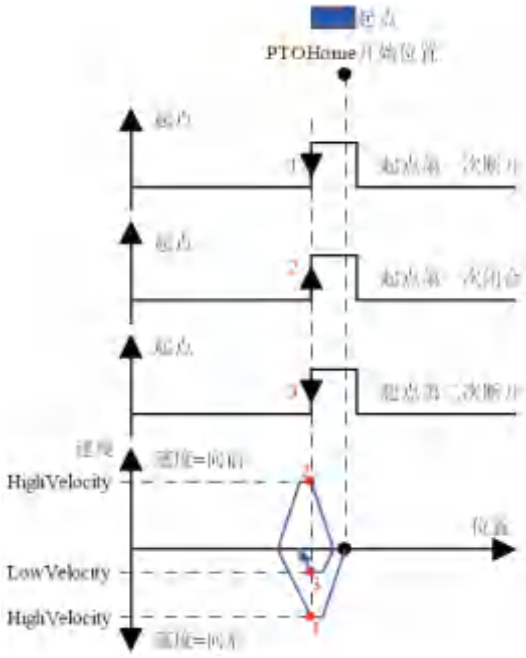
PTOHome相关参数：
HighVelocity最大初始回归搜索速度的值
LowVelocity最大最终回归接近速度的值

详细说明：起点为低电平时启动；以HighVelocity速度向前运行；第一次遇到起点上升沿反向，以LowVelocity速度向后运行；第一次遇到起点下降沿停止。

报错情况：
1、PTOHome在Execute不可以在起点的高电平期间内执行，否则PTOHome产生错误输出。
2、速度值关系：0<LowVelocity<=HighVelocity 最大频率，两个速度均不能为0，否则PTOHome产生错误输出。
3、正向运行时超过PTO配置窗口中的软件上限时，PTOHome产生错误输出。
4、PTO配置窗口中的启用了辅助输入中的PROX，相应的输入高电平时PTOSimple产生错误输出。
以上错误会引起PTOSimple功能块产生错误输出，需要在PTOSimple功能块ResetError处复位消除错误。

注意：
1、PTOHome中的LowVelocity值不宜过大，其大小决定回归的定位精度。
2、PTOHome中的Direction会影响PTOHome的动作行为。
3、PTOHome开始执行的位置、PTOHome的Direction及起点的位置的关系决定是否能正确寻到原点。

下图说明正向长凸轮回归模式，HomingType=1(PTO_LONG_CAM_POS,正向长凸轮) Direction=1(PTO_NEGATIVE)，



PTOHome相关参数：
HighVelocity最大初始回归搜索速度的值
LowVelocity最大最终回归接近速度的值

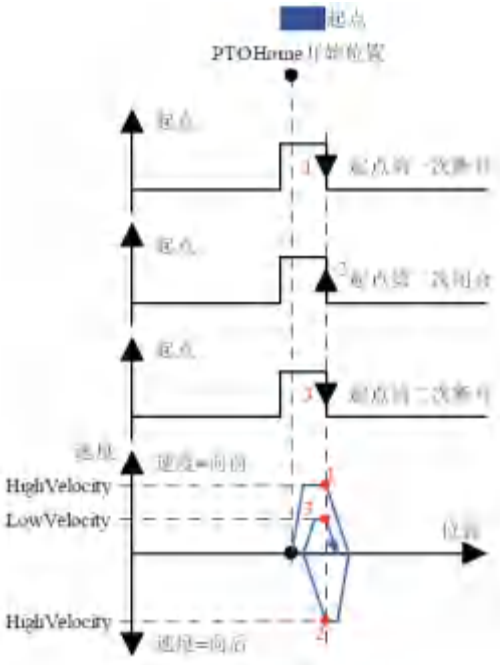
详细说明：起点为高电平时启动；以HighVelocity速度向后运行；第一次遇到起点下降沿反向，以HighVelocity速度向前运行；第一次遇到起点上升沿反向，以LowVelocity速度向后运行；第二次遇到起点下降沿停止。

报错情况：
1、PTOHome在Execute不可以在起点的低电平期间内执行，否则PTOHome产生错误输出。
2、速度值关系：0<LowVelocity<=HighVelocity 最大频率，两个速度均不能为0，否则PTOHome产生错误输出。
3、正向运行时超过PTO配置窗口中的软件下限时，PTOHome产生错误输出。
4、PTO配置窗口中的启用了辅助输入中的PROX，相应的输入高电平时PTOSimple产生错误输出。
以上错误会引起PTOSimple功能块产生错误输出，需要在PTOSimple功能块ResetError处复位消除错误。

注意：
1、PTOHome中的LowVelocity值不宜过大，其大小决定回归的定位精度。
2、PTOHome中的Direction会影响PTOHome的动作行为。
3、PTOHome开始执行的位置、PTOHome的Direction及起点的位置的关系决定是否能正确寻到原点。

六种寻原点方式
3. 负向长凸轮(Long Cam Negative
描述
PTOHome功能块的一种运行模式，用于将轴设置到参考位置。
负向长凸轮回归模式仅使用一个输入点：起点输入(凸轮)，PTO0使用I9，PTO1使用I11。

下图说明负向长凸轮回归模式，HomingType=2(PTO_LONG_CAM_NEG,负向长凸轮) Direction=0(PTO_POSITIVE)，
正向即向前：



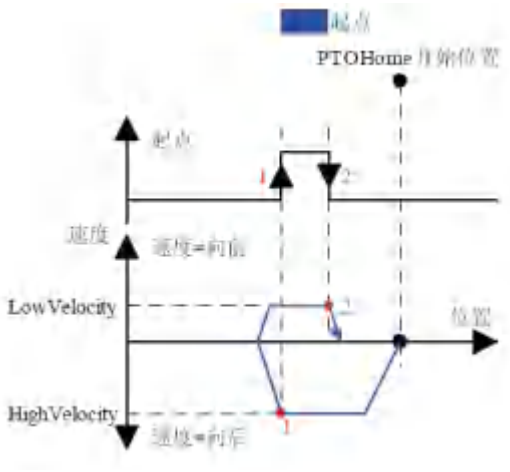
PTOHome相关参数：
HighVelocity最大初始回归搜索速度的值
LowVelocity最大最终回归接近速度的值

详细说明：起点为高电平时启动；以HighVelocity速度向前运行；第一次遇到起点下降沿反向，以HighVelocity速度向后运行；第二次遇到起点上升沿反向，以LowVelocity速度向前运行；第三次遇到起点下降沿停止。

报错情况：
1、PTOHome在Execute不可以在起点的低电平期间内执行，否则PTOHome产生错误输出。
2、速度值关系：0<LowVelocity<=HighVelocity 最大频率，两个速度均不能为0，否则PTOHome产生错误输出。
3、正向运行时超过PTO配置窗口中的软件上限时，PTOHome产生错误输出。
4、PTO配置窗口中的启用了辅助输入中的PROX，相应的输入高电平时PTOSimple产生错误输出。
以上错误会引起PTOSimple功能块产生错误输出，需要在PTOSimple功能块ResetError处复位消除错误。

注意：
1、PTOHome中的LowVelocity值不宜过大，其大小决定回归的定位精度。
2、PTOHome中的Direction会影响PTOHome的动作行为。
3、PTOHome开始执行的位置、PTOHome的Direction及起点的位置的关系决定是否能正确寻到原点。

下图说明负向长凸轮回归模式，HomingType=2(PTO_LONG_CAM_NEG,负向长凸轮) Direction=1(PTO_NEGATIVE)，
反向即向后：



PTOHome相关参数：
HighVelocity最大初始回归搜索速度的值
LowVelocity最大最终回归接近速度的值

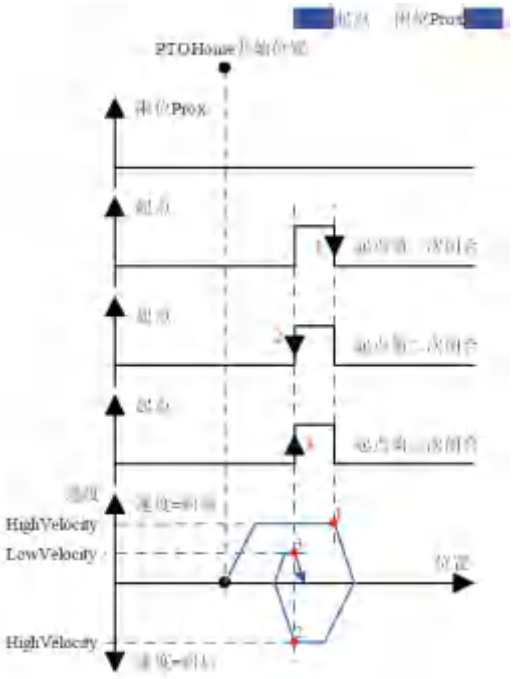
详细说明：起点为低电平时启动；以HighVelocity速度向后运行；第一次遇到起点上升沿反向，以LowVelocity速度向前运行；第一次遇到起点下降沿停止。

报错情况：
1、PTOHome在Execute不可以在起点的高电平期间内执行，否则PTOHome产生错误输出。
2、速度值关系：0<LowVelocity<=HighVelocity 最大频率，两个速度均不能为0，否则PTOHome产生错误输出。
3、正向运行时超过PTO配置窗口中的软件下限时，PTOHome产生错误输出。
4、PTO配置窗口中的启用了辅助输入中的PROX，相应的输入高电平时PTOSimple产生错误输出。
以上错误会引起PTOSimple功能块产生错误输出，需要在PTOSimple功能块ResetError处复位消除错误。

注意：
1、PTOHome中的LowVelocity值不宜过大，其大小决定回归的定位精度。
2、PTOHome中的Direction会影响PTOHome的动作行为。
3、PTOHome开始执行的位置、PTOHome的Direction及起点的位置的关系决定是否能正确寻到原点。

六种寻原点方式
4. 带正向限位的短凸轮(PTO_SHORT_CAM_POS)
描述
PTOHome功能块的一种运行模式，用于将轴设置到参考位置。
带正向限位短凸轮回归模式使用两个输入点：起点输入(凸轮)及限位输入Prox，PTO0使用I9及I8，PTO1使用I11及I10。

下图说明带正向限位的短凸轮回归模式，HomingType=3(PTO_SHORT_CAM_POS) Direction=0(PTO_POSITIVE)，正向即向前：



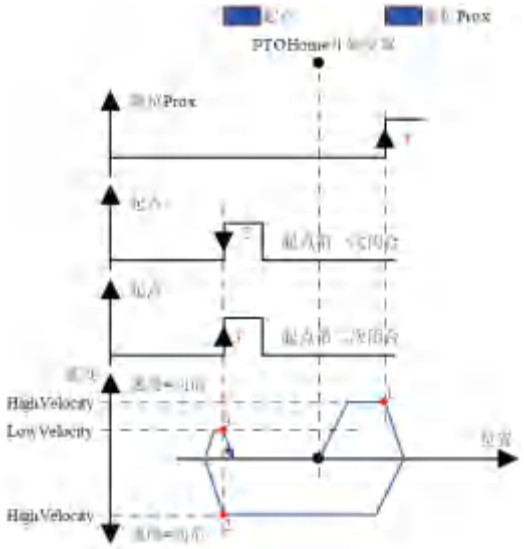
PTOHome相关参数：
HighVelocity最大初始回归搜索速度的值
LowVelocity最大最终回归接近速度的值

详细说明：起点为低电平，限位为低电平时启动；以HighVelocity速度向前运行，第一次遇到起点下降沿反向，以HighVelocity速度向后运行，第二次遇到到起点下降沿反向，以LowVelocity速度向前运行，第三次遇到起点上升沿停止。

报错情况：
1、PTOHome在Execute不可以在起点的高电平期间内执行，否则PTOHome产生错误输出。
2、速度值关系：0<LowVelocity<=HighVelocity 最大频率，两个速度均不能为0，否则PTOHome产生错误输出。
3、正向运行时超过PTO配置窗口中的软件上限或下限时，PTOHome产生错误输出。
以上错误会引起PTOSimple功能块产生错误输出，需要在PTOSimple功能块ResetError处复位消除错误。

注意：
1、PTOHome中的LowVelocity值不宜过大，其大小决定回归的定位精度。
2、PTOHome中的Direction会影响PTOHome的动作行为。
3、PTOHome开始执行的位置、PTOHome的Direction、起点的位置及限位Prox的位置的关系决定是否能正确寻到原点。

下图说明带正向限位的短凸轮回归模式，HomingType=3(PTO_SHORT_CAM_POS) Direction=0(PTO_POSITIVE)，正向即向前：



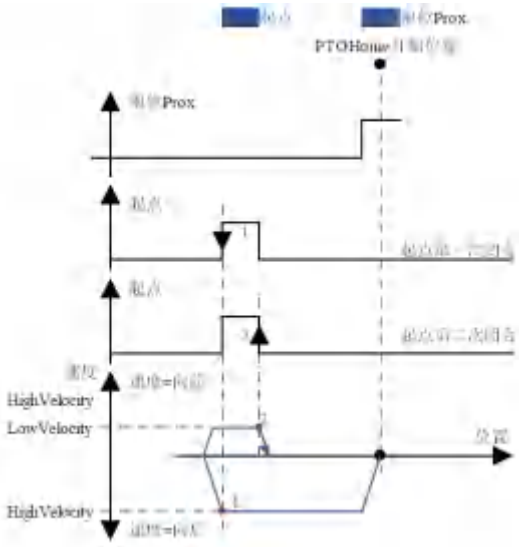
PTOHome相关参数：
HighVelocity最大初始回归搜索速度的值
LowVelocity最大最终回归接近速度的值

详细说明：起点为低电平，限位为低电平时启动；以HighVelocity速度向前运行；遇到限位上升沿反向以HighVelocity速度向后运行；第一次遇到起点下降沿反向，以LowVelocity速度向前运行，第二次遇到起点上升沿停止。

报错情况：
1、PTOHome在Execute不可以在起点的高电平期间内执行，否则PTOHome产生错误输出。
2、速度值关系：0<LowVelocity<=HighVelocity 最大频率，两个速度均不能为0，否则PTOHome产生错误输出。
3、正向运行时超过PTO配置窗口中的软件上限或下限时，PTOHome产生错误输出。
以上错误会引起PTOSimple功能块产生错误输出，需要在PTOSimple功能块ResetError处复位消除错误。

注意：
1、PTOHome中的LowVelocity值不宜过大，其大小决定回归的定位精度。
2、PTOHome中的Direction会影响PTOHome的动作行为。
3、PTOHome开始执行的位置、PTOHome的Direction、起点的位置及限位Prox的位置的关系决定是否能正确寻到原点。

下图说明带正向限位的短凸轮回归模式，HomingType=3(PTO_SHORT_CAM_POS) Direction=0(PTO_NEGATIVE)，反向即向后：



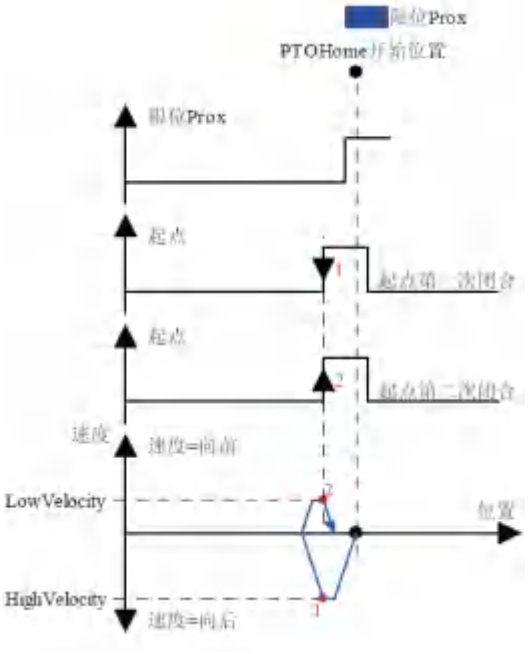
PTOHome相关参数：
HighVelocity最大初始回归搜索速度的值
LowVelocity最大最终回归接近速度的值

详细说明：起点为低电平，限位为高电平时启动；以HighVelocity速度向后运行，第一次遇到起点下降沿反向，以LowVelocity速度向前运行，第二次遇到起点上升沿停止。

报错情况：
1、PTOHome在Execute不可以在起点的高电平期间内执行，否则PTOHome产生错误输出。
2、速度值关系：0<LowVelocity<=HighVelocity 最大频率，两个速度均不能为0，否则PTOHome产生错误输出。
3、正向运行时超过PTO配置窗口中的软件上限或下限时，PTOHome产生错误输出。
以上错误会引起PTOSimple功能块产生错误输出，需要在PTOSimple功能块ResetError处复位消除错误。

注意：
1、PTOHome中的LowVelocity值不宜过大，其大小决定回归的定位精度。
2、PTOHome中的Direction会影响PTOHome的动作行为。
3、PTOHome开始执行的位置、PTOHome的Direction、起点的位置及限位Prox的位置的关系决定是否能正确寻到原点。

下图说明带正向限位的短凸轮回归模式，HomingType=3(PTO_SHORT_CAM_POS) Direction=1(PTO_NEGATIVE)，反向即向后：



PTOHome相关参数：
HighVelocity最大初始回归搜索速度的值
LowVelocity最大最终回归接近速度的值

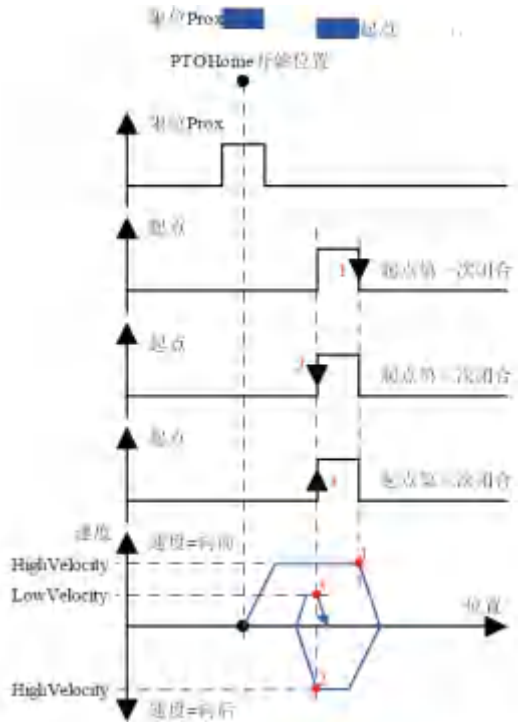
详细说明：起点为低电平，限位为高电平时启动；以HighVelocity速度向后运行，第一次遇到起点下降沿反向，以LowVelocity速度向前运行，遇到起点上升沿停止。

报错情况：
1、PTOHome在Execute不可以在起点的低电平期间内执行，否则PTOHome产生错误输出。
2、PTOHome在Execute不可以在限位的低电平期间内执行，否则PTOHome产生错误输出。
3、速度值关系：0<LowVelocity<=HighVelocity 最大频率，两个速度均不能为0，否则PTOHome产生错误输出。
4、正向运行时超过PTO配置窗口中的软件上限或下限时，PTOHome产生错误输出。
以上错误会引起PTOSimple功能块产生错误输出，需要在PTOSimple功能块ResetError处复位消除错误。

注意：
1、PTOHome中的LowVelocity值不宜过大，其大小决定回归的定位精度。
2、PTOHome中的Direction会影响PTOHome的动作行为。
3、PTOHome开始执行的位置、PTOHome的Direction、起点的位置及限位Prox的位置的关系决定是否能正确寻到原点。

六种寻原点方式
5. 带反向限位的短凸轮(PTO_SHORT_CAM_NEG)
描述
PTOHome功能块的一种运行模式，用于将轴设置到参考位置。
带反向限位短凸轮回归模式使用两个输入点：起点输入(凸轮)及限位输入Prox，PTO0使用I9及I8，PTO1使用I11及I10。

下图说明带反向限位的短凸轮回归模式，HomingType=4(PTO_SHORT_CAM_NEG) Direction=0(PTO_POSITIVE)，正向即向前：



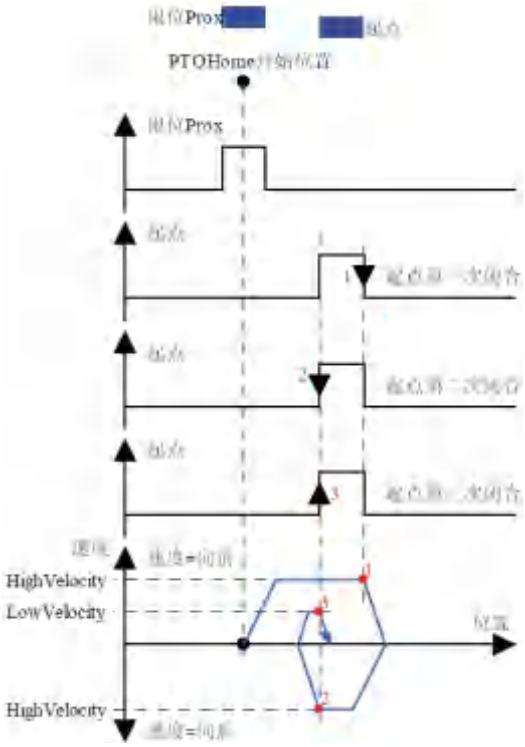
PTOHome相关参数：
HighVelocity最大初始回归搜索速度的值
LowVelocity最大最终回归接近速度的值

详细说明：起点为低电平，限位为高电平时启动；以HighVelocity速度向前运行；遇到起点下降沿反向，以HighVelocity速度向后运行；再次遇到起点下降沿反向，以LowVelocity速度向前运行；遇到起点上升沿停止。

报错情况：
1、PTOHome在Execute不可以在起点的高电平期间内执行，否则PTOHome产生错误输出。
2、PTOHome在Execute不可以在限位的低电平期间内执行，否则PTOHome产生错误输出。
3、速度值关系：0<LowVelocity<=HighVelocity 最大频率，两个速度均不能为0，否则PTOHome产生错误输出。
4、正向运行时超过PTO配置窗口中的软件上限或下限时，PTOHome产生错误输出。
以上错误会引起PTOSimple功能块产生错误输出，需要在PTOSimple功能块ResetError处复位消除错误。

注意：
1、PTOHome中的LowVelocity值不宜过大，其大小决定回归的定位精度。
2、PTOHome中的Direction会影响PTOHome的动作行为。
3、PTOHome开始执行的位置、PTOHome的Direction、起点的位置及限位Prox的位置的关系决定是否能正确寻到原点。

下图说明带反向限位的短凸轮回归模式，HomingType=4(PTO_SHORT_CAM_NEG) Direction=1(PTO_NEGATIVE)，反向即向后：



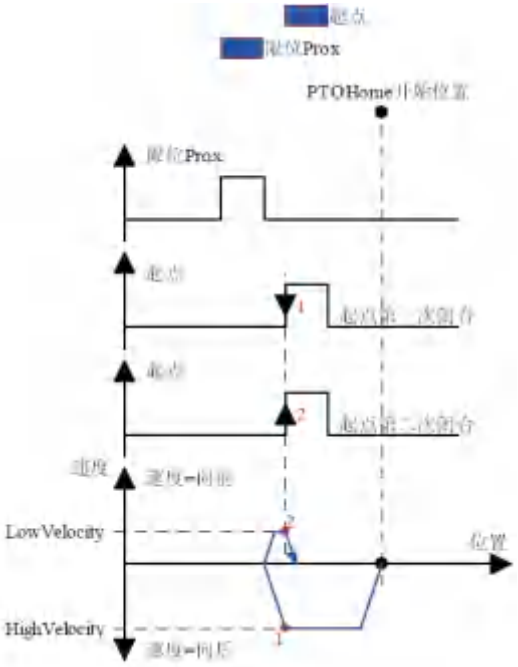
PTOHome相关参数：
HighVelocity最大初始回归搜索速度的值
LowVelocity最大最终回归接近速度的值

详细说明：起点为低电平，限位为高电平时启动；以HighVelocity速度向前运行；遇到起点下降沿反向，以HighVelocity速度向后运行；再次遇到起点下降沿反向，以LowVelocity速度向前运行；遇到起点上升沿停止。

报错情况：
1、PTOHome在Execute不可以在起点的高电平期间内执行，否则PTOHome产生错误输出。
2、速度值关系：0<LowVelocity<=HighVelocity 最大频率，两个速度均不能为0，否则PTOHome产生错误输出。
3、正向运行时超过PTO配置窗口中的软件上限或下限时，PTOHome产生错误输出。
以上错误会引起PTOSimple功能块产生错误输出，需要在PTOSimple功能块ResetError处复位消除错误。

注意：
1、PTOHome中的LowVelocity值不宜过大，其大小决定回归的定位精度。
2、PTOHome中的Direction会影响PTOHome的动作行为。
3、PTOHome开始执行的位置、PTOHome的Direction、起点的位置及限位Prox的位置的关系决定是否能正确寻到原点。

下图说明带反向限位的短凸轮回归模式，HomingType=4(PTO_SHORT_CAM_NEG) Direction=1(PTO_POSITIVE)，正向即向前：



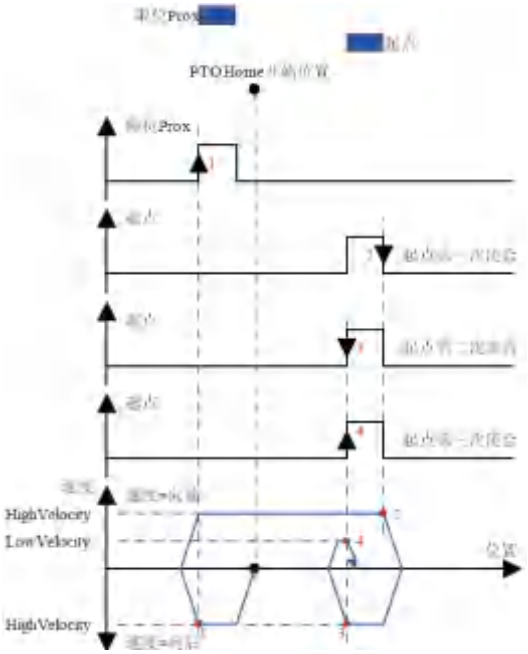
PTOHome相关参数：
HighVelocity最大初始回归搜索速度的值
LowVelocity最大最终回归接近速度的值

详细说明：起点为低电平，限位为低电平时启动；以HighVelocity速度向后运行；遇到起点下降沿反向,以LowVelocity速度向前运行；遇到起点上升沿停止。

报错情况：
1、PTOHome在Execute不可以在起点的高电平期间内执行，否则PTOHome产生错误输出。
2、速度值关系：0<LowVelocity<=HighVelocity 最大频率，两个速度均不能为0，否则PTOHome产生错误输出。
3、正向运行时超过PTO配置窗口中的软件上限或下限时，PTOHome产生错误输出。
以上错误会引起PTOSimple功能块产生错误输出，需要在PTOSimple功能块ResetError处复位消除错误。

注意：
1、PTOHome中的LowVelocity值不宜过大，其大小决定回归的定位精度。
2、PTOHome中的Direction会影响PTOHome的动作行为。
3、PTOHome开始执行的位置、PTOHome的Direction、起点的位置及限位Prox的位置的关系决定是否能正确寻到原点。

下图说明带反向限位的短凸轮回归模式，HomingType=4(PTO_SHORT_CAM_NEG) Direction=1(PTO_NEGATIVE)，反向即向后：



PTOHome相关参数：
HighVelocity最大初始回归搜索速度的值
LowVelocity最大最终回归接近速度的值

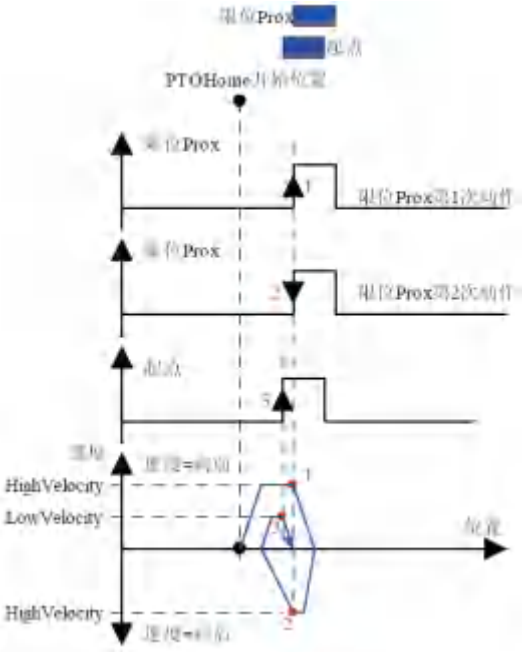
详细说明：起点为低电平，限位为低电平时启动；以HighVelocity速度向后运行；遇到限位上升沿反向，以HighVelocity速度向前运行；第一次遇到起点下降沿反向，以HighVelocity速度向后运行；第二次遇到起点下降沿反向，以LowVelocity速度向前运行；第三次遇到起点上升沿停止。

报错情况：
1、PTOHome在Execute不可以在起点的高电平期间内执行，否则PTOHome产生错误输出。
2、速度值关系：0<LowVelocity<=HighVelocity 最大频率，两个速度均不能为0，否则PTOHome产生错误输出。
3、正向运行时超过PTO配置窗口中的软件上限或下限时，PTOHome产生错误输出。
以上错误会引起PTOSimple功能块产生错误输出，需要在PTOSimple功能块ResetError处复位消除错误。

注意：
1、PTOHome中的LowVelocity值不宜过大，其大小决定回归的定位精度。
2、PTOHome中的Direction会影响PTOHome的动作行为。
3、PTOHome开始执行的位置、PTOHome的Direction、起点的位置及限位Prox的位置的关系决定是否能正确寻到原点。

六种寻原点方式
6. 带标记的短凸轮(PTO_SHORT_CAM_MARKER)
描述
PTOHome功能块的一种运行模式，用于将轴设置到参考位置。
带标记的短凸轮回归模式使用两个输入点：起点输入(凸轮)及限位输入Prox，PTO0使用I9及I8，PTO1使用I11及I10。

下图说明带标记的短凸轮回归模式，HomingType=5(PTO_SHORT_CAM_MARKER) Direction=0(PTO_POSITIVE)，正向即向前：



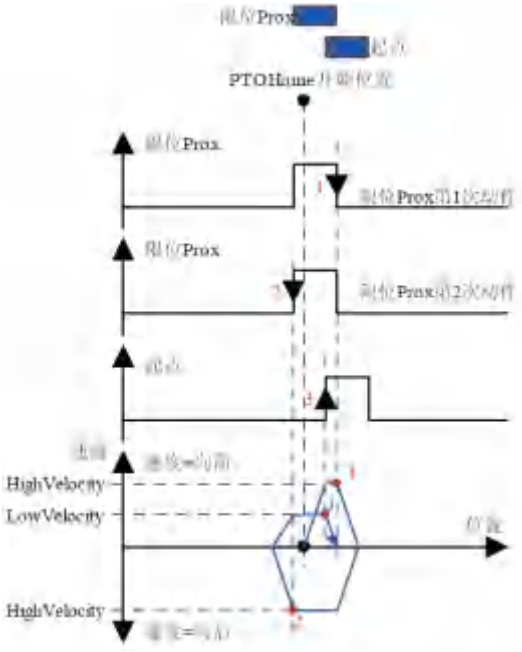
PTOHome相关参数：
HighVelocity最大初始回归搜索速度的值
LowVelocity最大最终回归接近速度的值

详细说明：起点为低电平，限位为低电平时启动；以HighVelocity速度向前运行；第一次遇到限位上升沿反向，以HighVelocity速度向后运行；第二次遇到限位下降沿反向，以LowVelocity速度向前运行；遇到起点上升沿停止。

报错情况：
1、速度值关系：0<LowVelocity<=HighVelocity 最大频率，两个速度均不能为0，否则PTOHome产生错误输出。
2、正向运行时超过PTO配置窗口中的软件上限或下限时，PTOHome产生错误输出。
以上错误会引起PTOSimple功能块产生错误输出，需要在PTOSimple功能块ResetError处复位消除错误。

注意：
1、PTOHome中的LowVelocity值不宜过大，其大小决定回归的定位精度。
2、PTOHome中的Direction会影响PTOHome的动作行为。
3、PTOHome开始执行的位置、PTOHome的Direction、起点的位置及限位Prox的位置的关系决定是否能正确寻到原点。

下图说明带标记的短凸轮回归模式，HomingType=5(PTO_SHORT_CAM_MARKER) Direction=0(PTO_POSITIVE)，正向即向前：



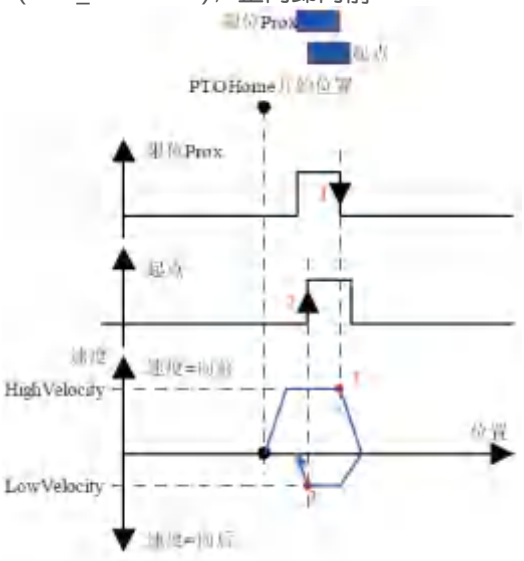
PTOHome相关参数：
HighVelocity最大初始回归搜索速度的值
LowVelocity最大最终回归接近速度的值

详细说明：起点为低电平，限位为高电平或起点为高电平，限位为低电平或起点为高电平，限位为高电平时启动；以HighVelocity速度向前运行；第一次遇到限位下降沿反向，以HighVelocity速度向后运行；第二次遇到限位下降沿反向，以LowVelocity速度向前运行；遇到起点上升沿停止。

报错情况：
1、速度值关系：0<LowVelocity<=HighVelocity 最大频率，两个速度均不能为0，否则PTOHome产生错误输出。
2、正向运行时超过PTO配置窗口中的软件上限或下限时，PTOHome产生错误输出。
以上错误会引起PTOSimple功能块产生错误输出，需要在PTOSimple功能块ResetError处复位消除错误。

注意：
1、PTOHome中的LowVelocity值不宜过大，其大小决定回归的定位精度。
2、PTOHome中的Direction会影响PTOHome的动作行为。
3、PTOHome开始执行的位置、PTOHome的Direction、起点的位置及限位Prox的位置的关系决定是否能正确寻到原点。

下图说明带标记的短凸轮回归模式，HomingType=5(PTO_SHORT_CAM_MARKER) Direction=0(PTO_POSITIVE)，正向即向前：



PTOHome相关参数：
HighVelocity最大初始回归搜索速度的值
LowVelocity最大最终回归接近速度的值

详细说明：起点及限位为任意高低电平组合均可启动；以HighVelocity速度向前运行；遇到限位下降沿反向，以LowVelocity速度向前运行；遇到起点上升沿停止。

报错情况：
1、速度值关系：0<LowVelocity<=HighVelocity 最大频率，两个速度均不能为0，否则PTOHome产生错误输出。
2、正向运行时超过PTO配置窗口中的软件上限或下限时，PTOHome产生错误输出。
以上错误会引起PTOSimple功能块产生错误输出，需要在PTOSimple功能块ResetError处复位消除错误。

注意：
1、PTOHome中的LowVelocity值不宜过大，其大小决定回归的定位精度。
2、PTOHome中的Direction会影响PTOHome的动作行为。
3、PTOHome开始执行的位置、PTOHome的Direction、起点的位置及限位Prox的位置的关系决定是否能正确寻到原点。

9.3.3 PTO-PTOMoveRelative
功能块名称：PTOMoveRelative
功能描述：相对定位，目标位置直接由其距离轴的当前位置的脉冲数指定。
此功能块指示相对于当前位置移动一定的距离。
移动配置文件取决于指定的速度、减速度和加速度值。
功能块图形：

下表介绍输入变量：

输入	类型	注释
PTO_REF_IN	PTO_REF	对 PTO 轴的参考。 连接到 PTOSimple 的 PTO_REF，或管理或运动输出引脚功能块的 PTO_REF_OUT。
Execute	BOOL	在上升沿启动功能块的执行。 如果为 FALSE，则在其执行终结时，复位功能块的输出。
Velocity	DWORD	最大速度 (Hz)（不一定达到）。 范围：1 到最大频率
Distance	DWORD	以脉冲数表示的运动距离。 范围：1...4294967295
Acceleration	DWORD	以赫兹/毫秒或毫秒（根据配置）表示的加速度。 范围（赫兹/毫秒）：1...加速度最大值 范围（毫秒）：最大加速度...最大软件限制
Deceleration	DWORD	以赫兹/毫秒或毫秒（根据配置）表示的减速度。 范围（赫兹/毫秒）：1...)减速度最大值 范围（毫秒）：最大减速度...最大软件限制
MODE	PTO_CMD_MODE	此枚举提供 FB 工作模式： 00：PTO_CMD_ABORTABLE，立即开始新的移动。 01：PTO_CMD_BUFFERED，缓冲新的移动。
Direction	PTO_DIRECTION	移动的方向。 00：PTO_POSITIVE，根据配置方向为正。 01：PTO_NEGATIVE，根据配置方向为负。 02：PTO_CURRENT，保持最后的方向。

注意：加速和减速斜坡不能超过 2,147,483,647 个脉冲。

输出变量:

输出	类型	注释
PTO_REF_OUT	PTO_REF	对 PTO 轴的参考。 连接到管理和运动功能块的 PTO_REF_IN 输入引脚。
Done	BOOL	TRUE = 表示命令完成。 功能块执行结束。
Busy	BOOL	TRUE = 表示命令正在执行中。
Active	BOOL	此输出在功能块控制相应轴运动时进行设置。
CommandAborted	BOOL	TRUE = 表示该命令因为另一个移动命令而中止。 功能块执行结束。
Error	BOOL	TRUE = 表示检测到一个错误。 功能块执行结束。
ErrID	PTOPWM_ERR_TYPE	当 Error 为 TRUE 时：检测到的错误的类型。 00: NO_ERROR,未检测到错误 01: PTO_UNKNOW_REF,，未知轴参考或配置不当的轴。 02: PTO_UNKNOW_PARAMETER,，未知参数类型 03: PTO_INVALID_PARAMETER,，用于所请求移动的无效参数值或不正确的参数值组合 04: PTO_COM_ERROR,检测到 PTO 接口通讯错误 05:PTO_AXIS_ERROR，检测到轴错误（例如状态机无效）。 06: PTO_CMD_ERROR,，缓冲区已满 07: PTO_LIMIT_FAULT， 回归模式中出现近似错误。

9.3.3 PTO-PTOMoveAbsolute
功能块名称：PTOMoveAbsolute
功能描述：绝对定位，目标位置由相对于先前设定的原点的脉冲数的坐标指定。
此功能块指示相对于先前设定的位置移动一定的距离。
移动配置文件取决于指定的速度、减速度和加速度值。
功能块图形：

管脚定义说明：
输入变量：

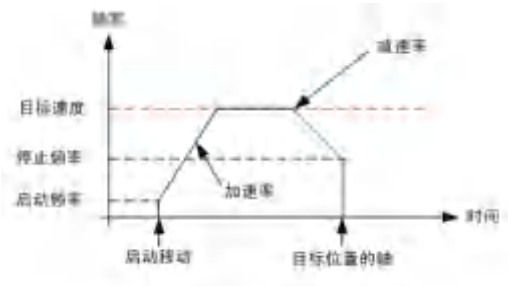
输入	类型	注释
PTO_REF_IN	PTO_REF	对 PTO 轴的参考。 连接到 PTOSimple 的 PTO_REF，或管理或运动输出引脚功能块的 PTO_REF_OUT。
Execute	BOOL	在上升沿启动功能块的执行。 如果为 FALSE，则在其执行终结时，复位功能块的输出。
Velocity	DWORD	最大速度 (Hz)（不一定达到）。 范围：1 到最大频率
Position	DWORD	以脉冲数表示的运动距离。 范围：1...2,147,483,647
Acceleration	DWORD	以赫兹/毫秒或毫秒（根据配置）表示的加速度。 范围（赫兹/毫秒）：1...加速度最大值 范围（毫秒）：最大加速度...最大软件限制
Deceleration	DWORD	以赫兹/毫秒或毫秒（根据配置）表示的减速度。 范围（赫兹/毫秒）：1...减速度最大值 范围（毫秒）：最大减速度...最大软件限制
Mode	PTO_CMD_MODE	此枚举提供 FB 工作模式： 00：PTO_CMD_ABORTABLE，立即开始新的移动。 01：PTO_CMD_BUFFERED，缓冲新的移动。
Deceleration	PTO_DIRECTION	移动的方向： 00：PTO_POSITIVE，根据配置方向为正。 01：PTO_NEGATIVE，根据配置方向为负。 02：PTO_CURRENT，保持最后的方向。

注意：加速和减速斜坡不能超过 2,147,483,647 个脉冲。

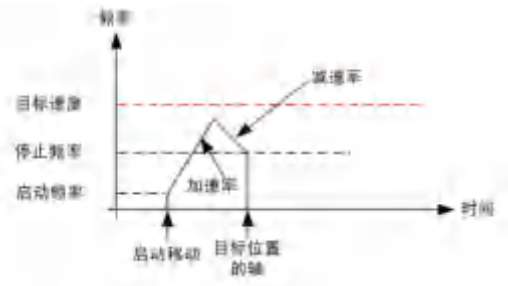
输出变量:

输出	类型	注释
PTO_REF_OUT	PTO_REF	对 PTO 轴的参考。 连接到管理和运动功能块的 PTO_REF_IN 输入引脚。
Done	BOOL	TRUE = 表示命令完成。 功能块执行结束。
Busy	BOOL	TRUE = 表示命令正在执行中。
Active	BOOL	此输出在功能块控制相应轴运动时进行设置。
CommandAborted	BOOL	TRUE = 表示该命令因为另一个移动命令而中止。 功能块执行结束。
Error	BOOL	TRUE = 表示检测到一个错误。 功能块执行结束。
ErrID	PTOPWM_ERR_TYPE	当 Error 为 TRUE 时：检测到的错误的类型。 00: NO_ERROR , 未检测到错误 01: PTO_UNKNOW_REF, 未知轴参考或配置不当的轴。 02: PTO_UNKNOW_PARAMETER, 未知参数类型 03: PTO_INVALID_PARAMETER, 用于所请求移动的无效参数值或不正确的参数值组合 04: PTO_COM_ERROR , 检测到 PTO 接口通讯错误 05: PTO_AXIS_ERROR, 检测到轴错误（例如状态机无效）。 06: PTO_CMD_ERROR, 缓冲区已满 07: PTO_LIMIT_FAULT, 回归模式中出现近似错误。

相对定位与绝对定位功能描述图:
位置控制功能块根据定义的速度、减速度和加速度值将轴移动一定距离。开始和停止频率通过调整参数定义。 距离只能有正值（方向由方向输入控制）



正常情况（在到达目标位置时达到设置的目标速度）



特殊情况（在到达目标位置之前无法达到设置的目标速度）

9.3.3 PTO-PTOMoveVelocity
功能块名称：PTOMoveVelocity
功能描述：
此功能块指示以指定速度连续移动。
按照指定的加速度和减速度值达到此速度。
功能块图形：

管脚定义说明：
输入变量：

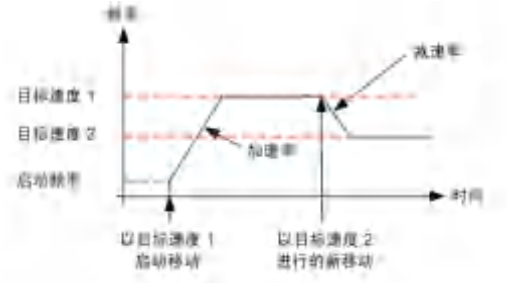
输入	类型	注释
PTO_REF_IN	PTO_REF	对 PTO 轴的参考。 连接到 PTOSimple 的 PTO_REF，或管理或运动输出引脚功能块的 PTO_REF_OUT。
Execute	BOOL	在上升沿启动功能块的执行。 如果为 FALSE，则在其执行终结时，复位功能块的输出。
Velocity	DWORD	以赫兹表示的目标速度。 范围：1 到最大频率
Acceleration	DWORD	以赫兹/毫秒或毫秒（根据配置）表示的加速度。 范围（毫秒）：最大加速度...最大软件限制
Deceleration	DWORD	以赫兹/毫秒或毫秒（根据配置）表示的减速度。 范围（毫秒）：最大减速度...最大软件限制
Mode	PTO_CMD_MODE	此枚举提供 FB 工作模式： 00：PTO_CMD_ABORTABLE，立即开始新的移动。 01：PTO_CMD_BUFFERED，缓冲新的移动。
Direction	PTO_DIRECTION	移动的方向。 00：PTO_POSITIVE，根据配置方向为正。 01：PTO_NEGATIVE，根据配置方向为负。 02：PTO_CURRENT，保持最后的方向。

注意： 加速和减速斜坡不能超过 2,147,483,647 个脉冲。

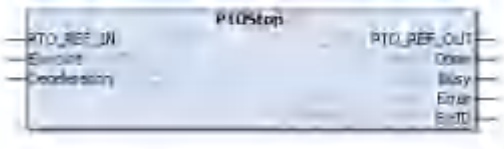
输出变量:

输出	类型	注释
PTO_REF_OUT	PTO_REF	对 PTO 轴的参考。 连接到管理和运动功能块的 PTO_REF_IN 输入引脚。
InVelocity	BOOL	TRUE = 表示已达到目标速度。 移动正在进行，功能块执行已完成。
Busy	BOOL	TRUE = 表示命令正在执行中。
CommandAborted	BOOL	TRUE = 表示该命令因为另一个移动命令而中止。 功能块执行结束。
Error	BOOL	TRUE = 表示检测到一个错误。 功能块执行结束。
ErrID	PTOPWM_ERR_TYPE	当 Error 为 TRUE 时：检测到的错误的类型。 00: NO_ERROR , 未检测到错误 01: PTO_UNKNOW_REF, 未知轴参考或配置不当的轴。 02: PTO_UNKNOW_PARAMETER, 未知参数类型 03: PTO_INVALID_PARAMETER, 用于所请求移动的无效参数值或不正确的参数值组合 04: PTO_COM_ERROR , 检测到 PTO 接口通讯错误 05: PTO_AXIS_ERROR, 检测到轴错误（例如状态机无效）。 06: PTO_CMD_ERROR, 缓冲区已满 07: PTO_LIMIT_FAULT, 回归模式中出现近似错误

功能描述图:



9.3.3 PTO-PTOStop
功能块名称：PTOStop
功能描述：
此功能块指示受控停止轴（减速到停止），并中止正在进行的所有运动。轴完全停止后，只要 Execute 输入仍为TRUE 或检测到轴错误并且未复位，就不会允许新的运动。
功能块图形：



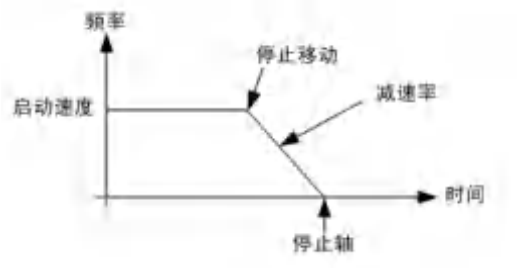
管脚定义说明：
输入变量：

输入	类型	注释
PTO_REF_IN	PTO_REF	对 PTO 轴的参考。 连接到 PTOSimple 的 PTO_REF，或管理或运动输出引脚功能块的 PTO_REF_OUT。
Execute	BOOL	在上升沿上启动功能块的执行。 当 Execute 为 True 时，任何运动命令都会被拒绝。 如果为 FALSE，则在其执行终结时，复位功能块的输出。
Deceleration	DWORD	以赫兹/毫秒或毫秒（根据配置）表示的减速度。 范围（赫兹/毫秒）：1...减速度最大值 范围（毫秒）：减速度最大值...100000

输出变量：

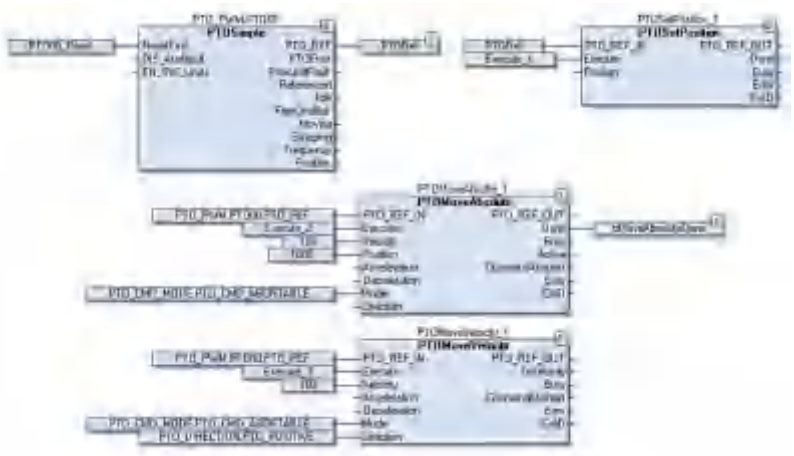
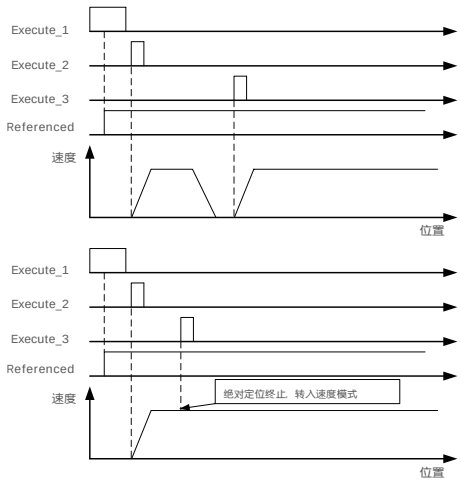
输出	类型	注释
PTO_REF_OUT	PTO_REF	对 PTO 轴的参考。 连接到管理和运动功能块的 PTO_REF_IN 输入引脚。
Done	BOOL	TRUE = 表示命令完成。 功能块执行结束。
Busy	BOOL	TRUE = 表示命令正在执行中。
Error	BOOL	TRUE = 表示检测到一个错误。 功能块执行结束。
ErrID	PTOPWM_ERR_TYPE	当 Error 为 TRUE 时：检测到的错误的类型。 00: NO_ERROR , 未检测到错误 01: PTO_UNKNOW_REF, 未知轴参考或配置不当的轴。 02: PTO_UNKNOW_PARAMETER, 未知参数类型 03: PTO_INVALID_PARAMETER, 用于所请求移动的无效参数值或不正确的参数值组合 04: PTO_COM_ERROR , 检测到 PTO 接口通讯错误 05: PTO_AXIS_ERROR, 检测到轴错误（例如状态机无效）。 06: PTO_CMD_ERROR, 缓冲区已满 07: PTO_LIMIT_FAULT, 回归模式中出现近似错误

功能描述图：



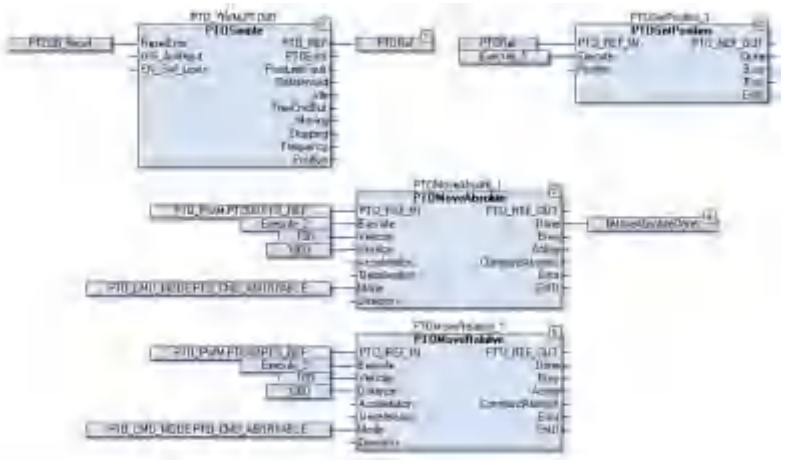
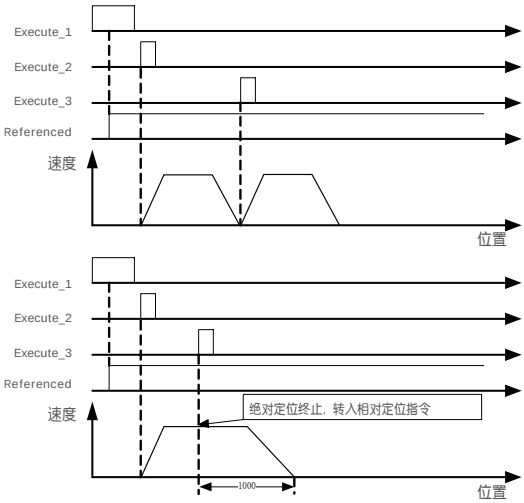
9.3.3 PTO-Abort模式(PTOAbsolute to PTOMoveVelocity)

操作符	PTOAbsolute to PTOMoveVelocity in AbortMode		
功能说明	退出模式下，绝对位置指令向速度指令转换。		
用到的功能块1			
用到的功能块2			
用到的功能块3			
用到的功能块4			
功能块管脚定义	参见PTOSimple章节 参见PTOSetPosition章节 参见PTOMoveAbsolute章节 参见PTOMoveVelocity章节		
声明	VAR_GLOBAL		
	PTOMoveAbsolte_1	: PTOMoveAbsolute;	
	PTOMoveVelocity_1	: PTOMoveVelocity;	
	PTOSetPosition_1	: PTOSetPosition;	
	PTORef	: PTO_REF;	
	Execute_1	:BOOL;	//1#功能块启动命令
	Execute_2	:BOOL;	//2#功能块启动命令
	Execute_3	:BOOL;	//3#功能块启动命令
	PTO00_Reset	:BOOL;	//复位PTO错误
	END_VAR		

示例	
时序图	
使用限制	1. PTOMoveAbsolute执行前，必须执行PTOSetPosition或PTOHome指令，使标定原点。 2. 绝对位置指令执行过程中，切换速度指令脉冲输出方向必须与绝对位置指令脉冲输出方向相同，否则将产生PTOError。此错误只能通过复位PTOSimple的ResetError或重启PLC消除。
注意事项	

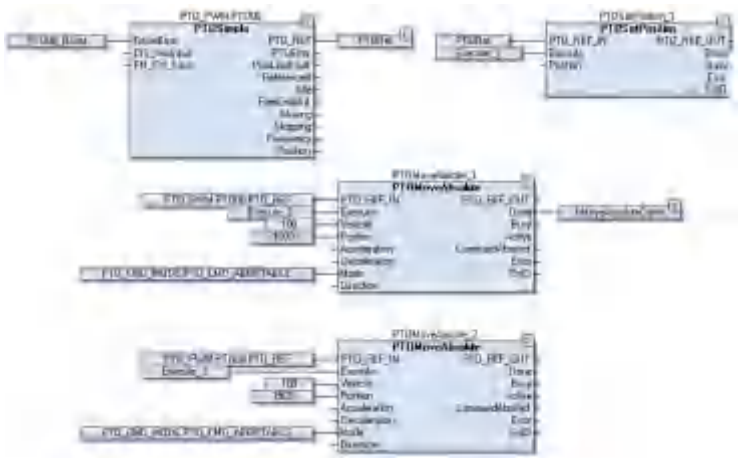
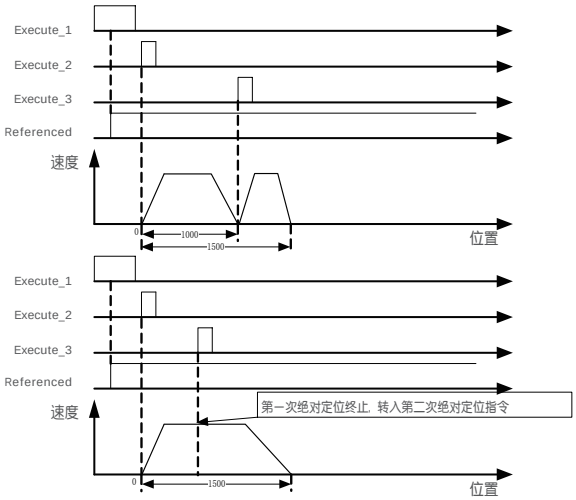
9.3.3 PTO-Abort模式(PTOAbsolute to PTOMoveRelative)

操作符	PTOAbsolute to PTOMoveRelative in AbortMode		
功能说明	退出模式下，绝对位置指令向相对位置指令转换。		
用到的功能块1			
用到的功能块2			
用到的功能块3			
用到的功能块4			
功能块管脚定义	参见PTOSimple章节 参见PTOSetPosition章节 参见PTOMoveAbsolute章节 参见PTOMoveRelative章节		
声明	VAR_GLOBAL		
	PTOMoveAbsolte_1	: PTOMoveAbsolute;	
	PTOMoveRelative_1	: PTOMoveRelative;	
	PTOSetPosition_1	: PTOSetPosition;	
	PTORef	: PTO_REF;	
	Execute_1	:BOOL;	//1#功能块启动命令
	Execute_2	:BOOL;	//2#功能块启动命令
	Execute_3	:BOOL;	//3#功能块启动命令
	PTO00_Reset	:BOOL;	//复位PTO错误
	END_VAR		

示例	
时序图	
使用限制	1. PTOMoveAbsolute执行前，必须执行PTOSetPosition或PTOHome指令，使标定原点。 2. 绝对位置指令执行过程中，切换相对位置指令脉冲输出方向必须与绝对位置指令脉冲输出方向相同，否则将产生PTOError，此错误只能通过复位PTOSimple的ResetError或重启PLC消除。
注意事项	

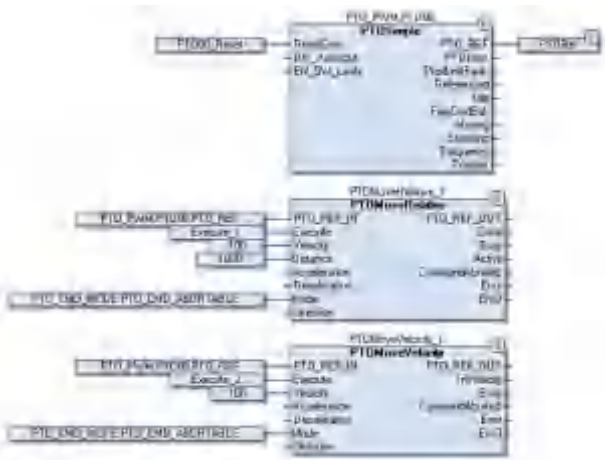
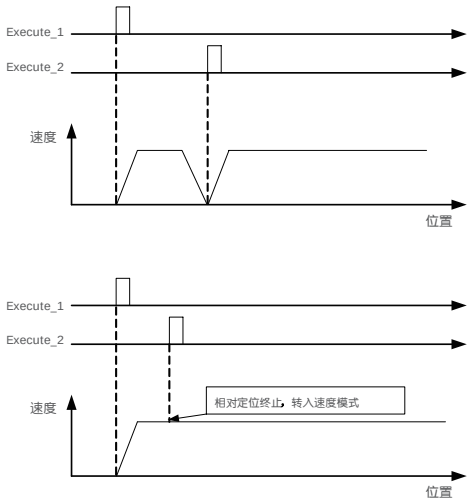
9.3.3 PTO-Abort模式(PTOAbsolute to PTOAbsolute)

操作符	PTOAbsolute to PTOAbsolute in AbortMode		
功能说明	退出模式下，绝对位置指令向绝对位置指令转换。		
用到的功能块1			
用到的功能块2			
用到的功能块3			
功能块管脚定义	参见PTOSimple章节 参见PTOSetPosition章节 参见PTOMoveAbsolute章节		
声明	VAR_GLOBAL		
	PTOMoveAbsolte_1	: PTOMoveAbsolute;	
	PTOMoveAbsolte_2	: PTOMoveAbsolute;	
	PTOSetPosition_1	: PTOSetPosition;	
	PTORef	: PTO_REF;	
	Execute_1	:BOOL;	//1#功能块启动命令
	Execute_2	:BOOL;	//2#功能块启动命令
	Execute_3	:BOOL;	//3#功能块启动命令
	PTO00_Reset	:BOOL;	//复位PTO错误
	END_VAR		

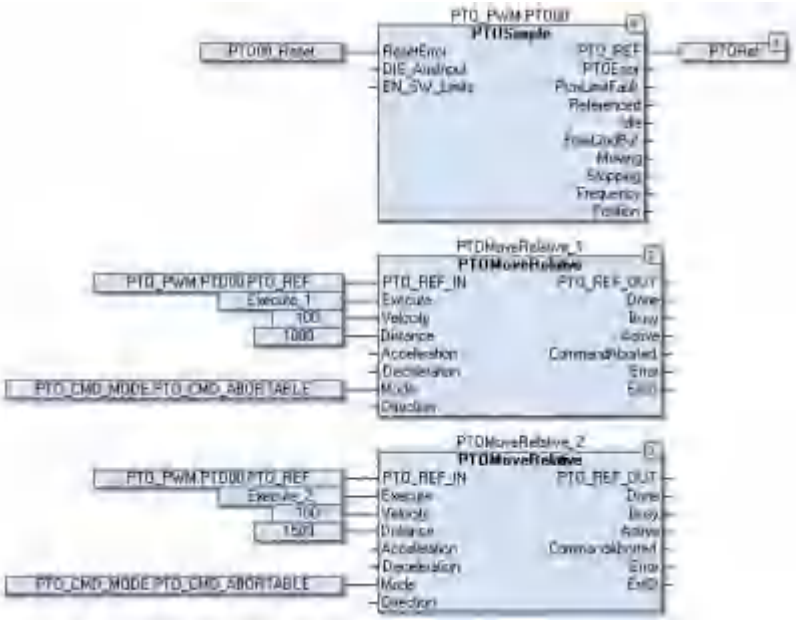
示例	
时序图	
使用限制	1. PTOMoveAbsolute执行前，必须执行PTOSetPosition或PTOHome指令，使标定原点。 2. 首次绝对位置指令执行过程中，切换绝对位置指令脉冲输出方向必须与前次绝对位置指令脉冲输出方向相同，否则将产生PTOError。此错误只能通过复位PTOSimple的ResetError或重启PLC消除。
注意事项	

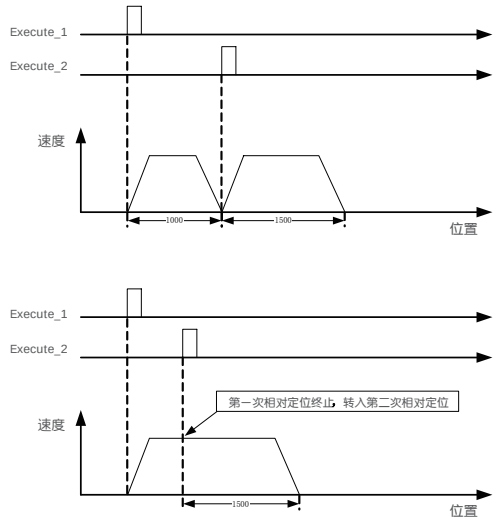
9.3.3 PTO-Abort模式(PTOMoveRelative to PTOMoveVelocity)

操作符	PTOMoveRelative to PTOMoveVelocity in AbortMode		
功能说明	退出模式下，相对位置指令向速度指令转换。		
用到的功能块1			
用到的功能块2			
用到的功能块3			
功能块管脚定义	参见PTOSimple章节 参见PTOMoveRelative章节 参见PTOMoveVelocity章节		
声明	VAR_GLOBAL		
	bMoveAbsoluteDone: BOOL;		
	PTOMoveRelative_1	: PTOMoveRelative;	
	PTOMoveVelocity_1	: PTOMoveVelocity;	
	PTORef	: PTO_REF;	
	Execute_1	:BOOL;	//1#功能块启动命令
	Execute_2	:BOOL;	//2#功能块启动命令
	PTO00_Reset	:BOOL;	//复位PTO错误
	END_VAR		

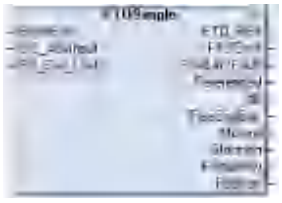
示例	
时序图	
使用限制	1. 相对位置指令执行过程中，切换速度指令脉冲输出方向必须与相对位置指令脉冲输出方向相同，否则将产生PTOError，此错误只能通过复位PTOSimple的ResetError或重启PLC消除。
注意事项	

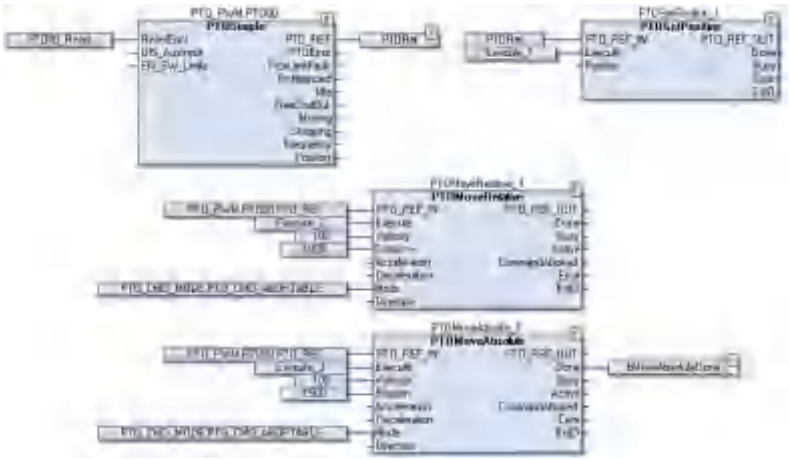
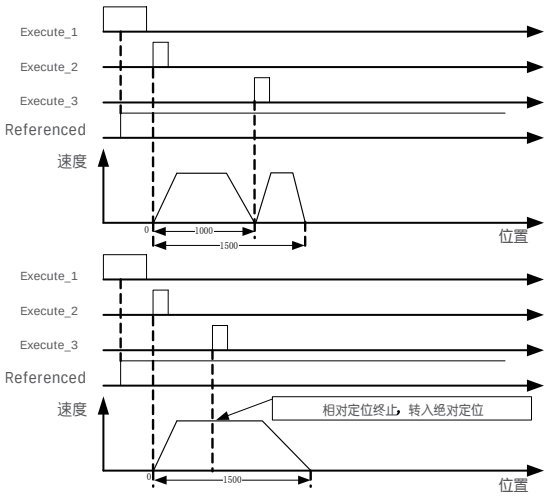
9.3.3 PTO-Abort模式(PTOMoveRelative to PTOMoveRelative)

操作符	PTOMoveRelative to PTOMoveRelative in AbortMode		
功能说明	退出模式下，相对位置指令向相对位置指令转换。		
用到的功能块1			
用到的功能块2			
功能块管脚定义	参见PTOSimple章节 参见PTOMoveRelative章节		
声明	VAR_GLOBAL		
	bMoveAbsoluteDone: BOOL;		
	PTOMoveRelative_1	: PTOMoveRelative;	
	PTOMoveRelative_2	: PTOMoveRelative;	
	PTORef	: PTO_REF;	
	Execute_1	:BOOL;	//1#功能块启动命令
	Execute_2	:BOOL;	//2#功能块启动命令
	PTO00_Reset	:BOOL;	//复位PTO错误
	END_VAR		
示例			

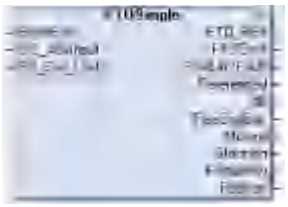
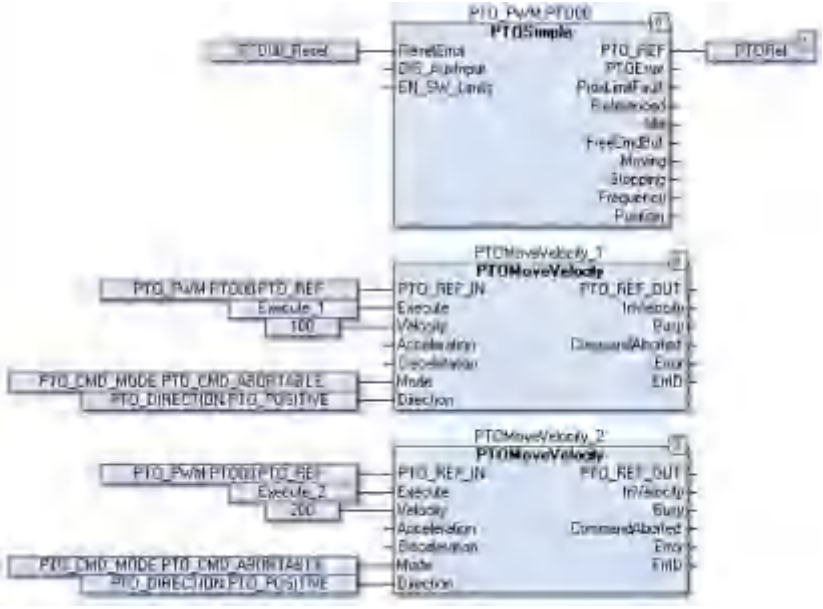
时序图	
使用限制	1. 第一次相对位置指令执行过程中，第二次切换相对位置指令脉冲输出方向必须与第一次相对位置指令脉冲输出方向相同，否则将产生PTOError，此错误只能通过复位PTOSimple的ResetError或重启PLC消除。
注意事项	

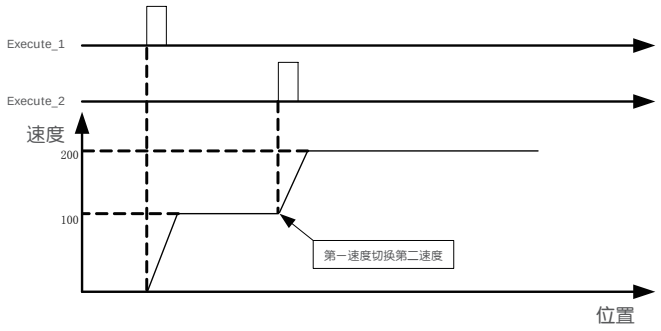
9.3.3 PTO-Abort模式(PTOMoveRelative to PTOMoveAbsolute)

操作符	PTOMoveRelative to PTOMoveAbsolute in AbortMode		
功能说明	退出模式下，相对位置指令向绝对位置指令转换。		
用到的功能块1			
用到的功能块2			
用到的功能块3			
用到的功能块4			
功能块管脚定义	参见PTOSimple章节 参见PTOSetPosition章节 参见PTOMoveAbsolute章节 参见PTOMoveRelative章节		
声明	VAR_GLOBAL		
	PTOMoveAbsolte_1	: PTOMoveAbsolute;	
	PTOMoveRelative_1	: PTOMoveRelative;	
	PTOSetPosition_1	: PTOSetPosition;	
	PTORef	: PTO_REF;	
	Execute_1	:BOOL;	//1#功能块启动命令
	Execute_2	:BOOL;	//2#功能块启动命令
	Execute_3	:BOOL;	//3#功能块启动命令
	PTO00_Reset	:BOOL;	//复位PTO错误
	END_VAR		

示例	
时序图	
使用限制	1. PTOMoveAbsolute执行前，必须执行PTOSetPosition或PTOHome指令，使标定原点。 2. 相对位置指令执行过程中，切换绝对位置指令脉冲输出方向必须与相对位置指令脉冲输出方向相同，否则将产生PTOError，此错误只能通过复位PTOSimple的ResetError或重启PLC消除。
注意事项	

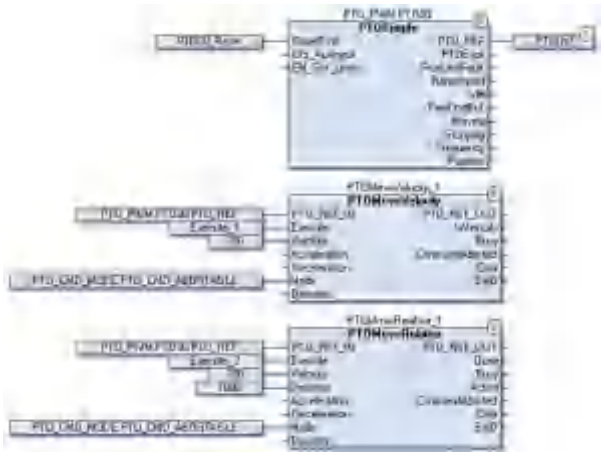
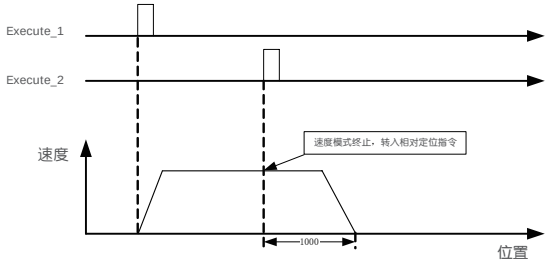
9.3.3 PTO-Abort模式(PTOMoveVelocity to PTOMoveVelocity)

操作符	PTOMoveVelocity to PTOMoveVelocity in AbortMode		
功能说明	退出模式下，速度指令向速度指令转换。		
用到的功能块1			
用到的功能块2			
功能块管脚定义	参见PTOSimple章节 参见PTOMoveVelocity章节		
声明	VAR_GLOBAL		
	bMoveAbsoluteDone: BOOL;		
	PTOMoveVelocity_1	: PTOMoveVelocity;	
	PTOMoveVelocity_2	: PTOMoveVelocity;	
	PTORef	: PTO_REF;	
	Execute_1	:BOOL;	//1#功能块启动命令
	Execute_2	:BOOL;	//2#功能块启动命令
	PTO00_Reset	:BOOL;	//复位PTO错误
	END_VAR		
示例			

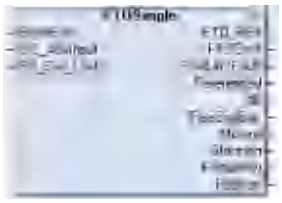
时序图	
使用限制	1. 第一次速度指令执行过程中，第二次切换速度指令脉冲输出方向必须与第一次速度指令脉冲输出方向相同，否则将产生PTOError，此错误只能通过复位PTOSimple的ResetError或重启PLC消除。
注意事项	

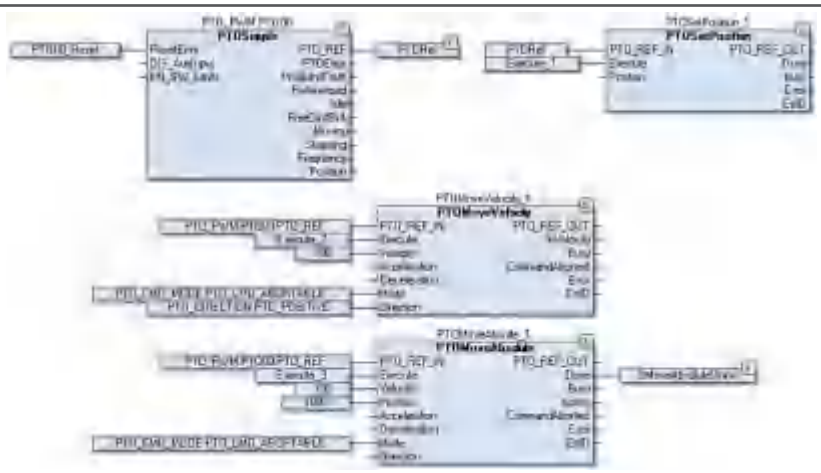
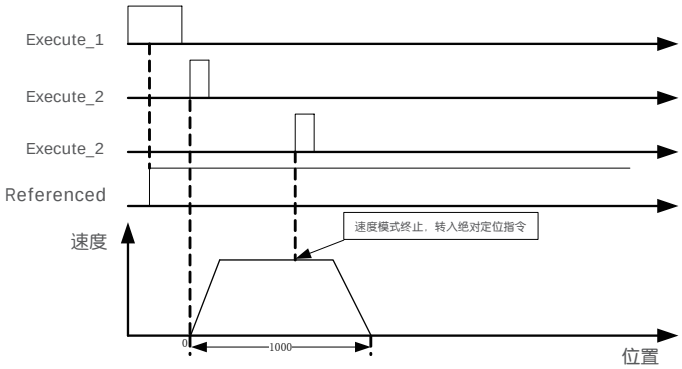
9.3.3 PTO-Abort模式(PTOMoveVelocity to PTOMoveRelative)

操作符	PTOMoveVelocity to PTOMoveRelative in AbortMode		
功能说明	退出模式下，速度指令向相对位置指令转换。		
用到的功能块1			
用到的功能块2			
用到的功能块3			
功能块管脚定义	参见PTOSimple章节 参见PTOMoveRelative章节 参见PTOMoveVelocity章节		
声明	VAR_GLOBAL		
	bMoveAbsoluteDone: BOOL;		
	PTOMoveRelative_1	: PTOMoveRelative;	
	PTOMoveVelocity_1	: PTOMoveVelocity;	
	PTORef	: PTO_REF;	
	Execute_1	:BOOL;	//1#功能块启动命令
	Execute_2	:BOOL;	//2#功能块启动命令
	PTO00_Reset	:BOOL;	//复位PTO错误
	END_VAR		

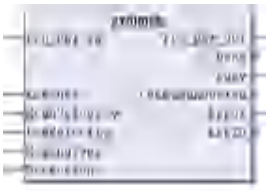
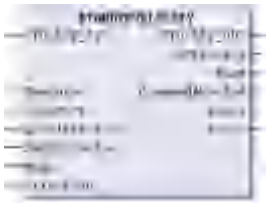
示例	
时序图	
使用限制	1. 速度指令执行过程中，切换相对位置指令脉冲输出方向必须与速度指令脉冲输出方向相同，否则将产生PTOError，此错误只能通过复位PTOSimple的ResetError或重启PLC消除。
注意事项	

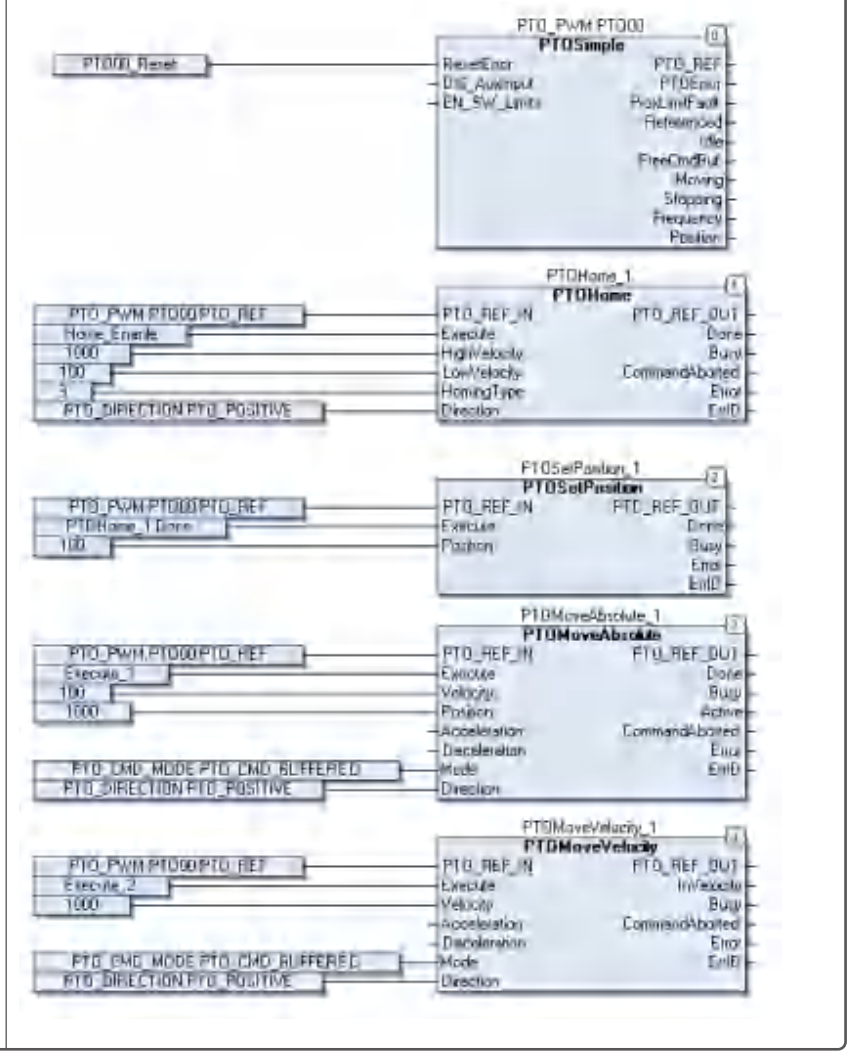
9.3.3 PTO-Abort模式(PTOMoveVelocity to PTOAbsolute)

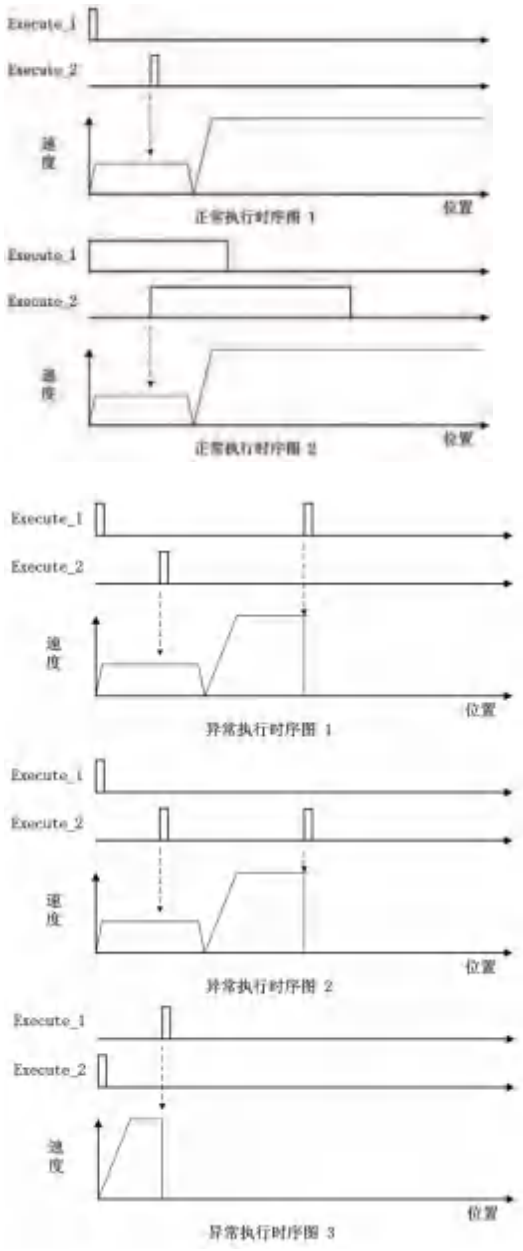
操作符	PTOMoveVelocity to PTOAbsolute in AbortMode		
功能说明	退出模式下，速度指令向绝对位置指令转换。		
用到的功能块1			
用到的功能块2			
用到的功能块3			
用到的功能块4			
功能块管脚定义	参见PTOSimple章节 参见PTOSetPosition章节 参见PTOMoveAbsolute章节 参见PTOMoveVelocity章节		
声明	VAR_GLOBAL		
	PTOMoveAbsolte_1	: PTOMoveAbsolute;	
	PTOMoveVelocity_1	: PTOMoveVelocity;	
	PTOSetPosition_1	: PTOSetPosition;	
	PTORef	: PTO_REF;	
	Execute_1	:BOOL;	//1#功能块启动命令
	Execute_2	:BOOL;	//2#功能块启动命令
	Execute_3	:BOOL;	//3#功能块启动命令
	PTO00_Reset	:BOOL;	//复位PTO错误
	END_VAR		

示例	
时序图	
使用限制	1. PTOMoveAbsolute执行前，必须执行PTOSetPosition或PTOHome指令，使标定原点。 2. 速度指令执行过程中，切换绝对位置指令脉冲输出方向必须与速度指令脉冲输出方向相同，否则将产生PTOError，此错误只能通过复位PTOSimple的ResetError或重启PLC消除。
注意事项	

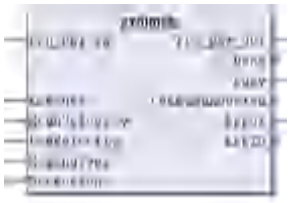
9.3.3 PTO-Buffer模式(PTOMoveAbsolute and PTOMoveVelocity)

操作符	PTOMoveAbsolute and PTOMoveVelocity BufferMode		
功能说明	绝对位移与速度模式之间采用缓存运行模式		
用到的功能块1			
用到的功能块2			
用到的功能块3			
用到的功能块4			
用到的功能块5			
功能块管脚定义	参见PTOSIMPLE章节 参见PTOHome章节 参见PTOSetPosition章节 参见PTOMoveAbsolute章节 参见 PTOMoveVelocity章节		
	VAR_GLOBAL		
	PTOMoveVelocity_1	: PTOMoveVelocity;	
	PTOMoveAbsolute_1	: PTOMoveAbsolute;	
	PTOHome_1	: PTOHome;	
	PTOSetPosition_1	: PTOSetPosition;	
	Execute_1	:BOOL;	//1#功能块启动命令
	Execute_2	:BOOL;	//2#功能块启动命令
	PTO00_Reset	:BOOL;	//复位PTO错误
	Home_Enanle	:BOOL;	//寻原点启动
	END_VAR		

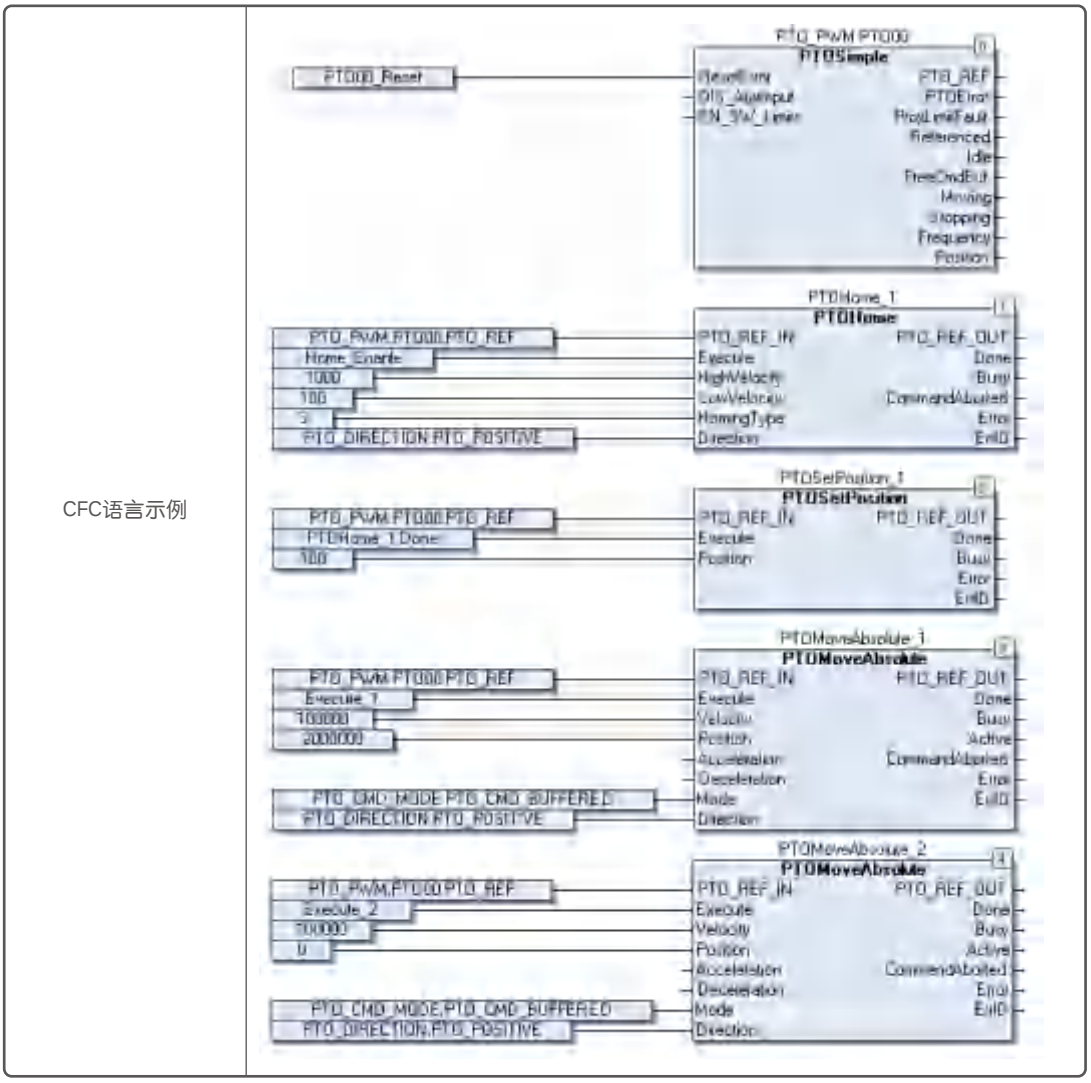
CFC语言示例	 PTO_PWM.PTO00 PTO_REF PTOSimple PTO00_Reset ResetError DIS_Axis.Input EN_SW_Limits PTO_REF PTOError PosLimitFault Referenced Idle FreeCmdBuf Moving Stopping Frequency Position PTOHome_1 PTOHome PTO_PWM.PTO00.PTO_REF Home_Enable 1000 100 3 PTO_DIRECTION.PTO_POSITIVE PTO_REF_IN Execute HighVelocity LowVelocity HomingType Direction PTO_REF_OUT Done Busy CommandAborted Error ErrID PTOSetPosition_1 PTOSetPosition PTO_PWM.PTO00.PTO_REF PTOHome_1.Done 100 PTO_REF_IN Execute Position PTO_REF_OUT Done Busy Error ErrID PTOMoveAbsolute_1 PTO_PWM.PTO00.PTO_REF Execute_1 100 1000 PTO_CMD_MODE.PTO_CMD_BUFFERED PTO_DIRECTION.PTO_POSITIVE PTO_REF_IN Execute Velocity Position Acceleration Deceleration Mode Direction PTO_REF_OUT Done Busy Active CommandAborted Error ErrID PTOMoveVelocity_1 PTO_PWM.PTO00.PTO_REF Execute_2 1000 PTO_CMD_MODE.PTO_CMD_BUFFERED PTO_DIRECTION.PTO_POSITIVE PTO_REF_IN Execute Velocity Acceleration Deceleration Mode Direction PTO_REF_OUT InVelocity Busy CommandAborted Error ErrID

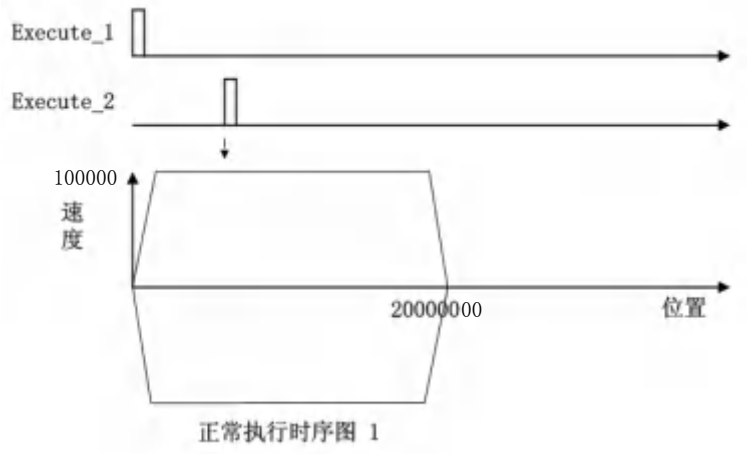
时序图	
使用限制	速度模式下不能使用缓存模式。如果在速度模式下使用缓存模式则PTO出错，PTO停止发送脉冲 PTO命令是依照先进先出的原则，所以功能块的执行时可能由于扫描周期或在堆栈的位置会延迟几个扫描周期 使用绝对位移功能块，没有回原点之前，如果PTOSIMPLE报错误，请检查极限限位开关，这个限位接常开信号；可以回原点，但是不能运行绝对位移功能块
注意事项	在PTO的配置中一定要启用AUX和PROX，否则不能回原点 在PTO_PWM.PTO00.FreeCmdBuf为True时才能把第二条PTO指令发送到PTO的缓存中，否则PTO报错，PTO停止发送脉冲。

9.3.3 PTO-Buffer模式(PTOMoveAbsolute and PTOMoveAbsolute)

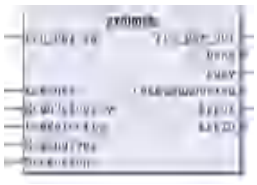
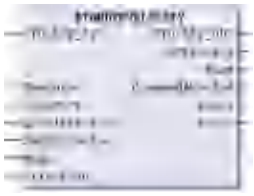
操作符	PTOMoveAbsolute and PTOMoveAbsolute BufferMode		
功能说明	绝对位移与绝对位移之间采用缓存运行模式		
用到的功能块1			
用到的功能块2			
用到的功能块3			
用到的功能块4			
功能块管脚定义	参见PTOSIMPLE章节 参见PTOHome章节 参见PTOSetPosition章节 参见PTOMoveAbsolute章节		
	VAR_GLOBAL		
	PTOMoveAbsolute_1	: PTOMoveRelative;	
	PTOMoveAbsolute_2	: PTOMoveAbsolute;	
	PTOHome_1	: PTOHome;	
	PTOSetPosition_1	: PTOSetPosition;	
	Execute_1	:BOOL;	//1#功能块启动命令
	Execute_2	:BOOL;	//2#功能块启动命令
	PTO00_Reset	:BOOL;	//复位PTO错误
	Home_Enanle	:BOOL;	//寻原点启动
	END_VAR		

CFC语言示例

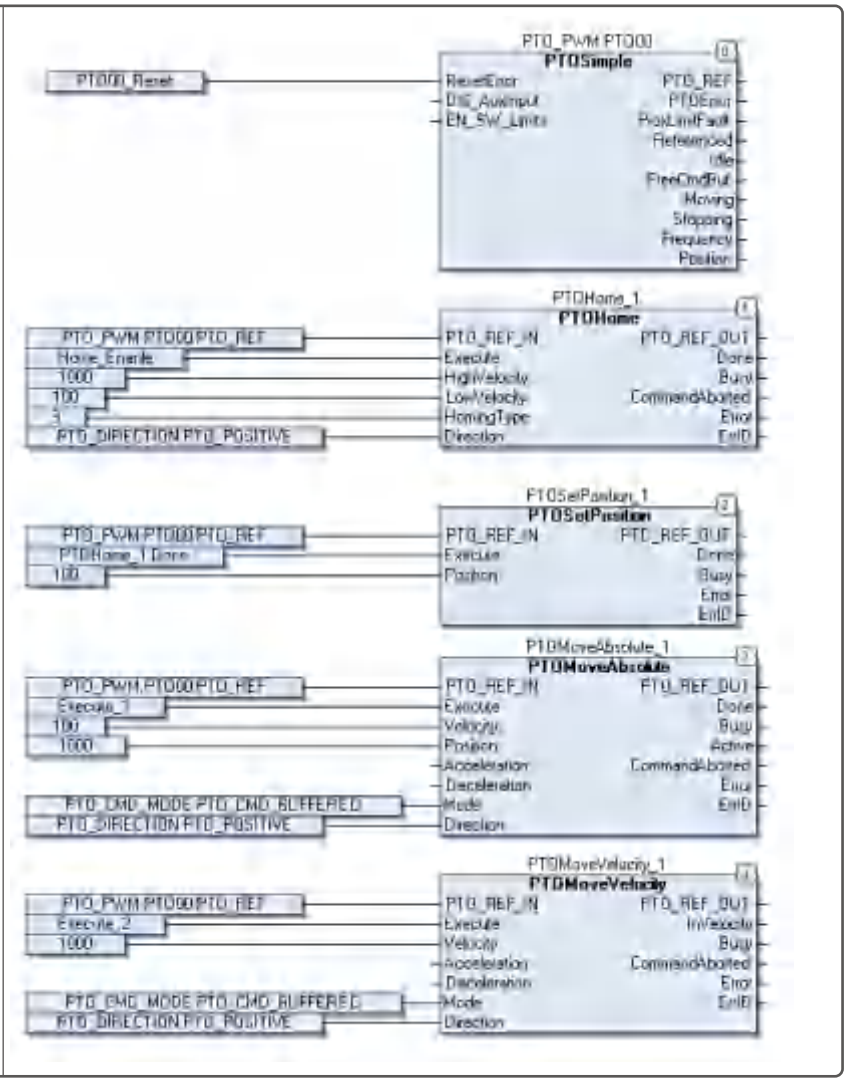

PTO_PWM.PTO00
PTOSimple
PTO00_Reset
PTO_REF
PTOHome_1
PTOHome
PTO_PWM.PTO00.PTO_REF
Home_Enable
1000
100
0
PTO_DIRECTION.PTO_POSITIVE
PTO_REF_IN
PTO_REF_OUT
Execute
Done
Busy
Error
HomingType
Direction
ErrID
PTOSetPosition_1
PTOSetPosition
PTOHome_1.Done
Position
PTOMoveAbsolute_1
PTOMoveAbsolute
Execute_1
100000
2000000
Velocity
Position
Acceleration
Deceleration
Mode
PTO_CMD_MODE.PTO_CMD_BUFFERED
PTOMoveAbsolute_2
Execute_2
10000
0
CommandAborted
Active

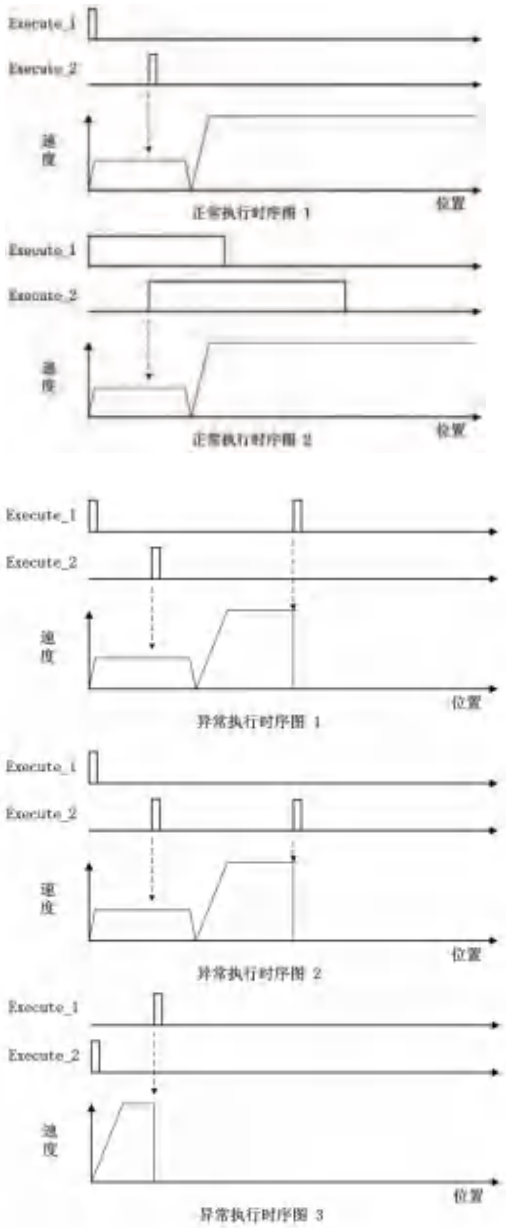
时序图	 正常执行时序图 1
使用限制	PTO命令是依照先进先出的原则，所以功能块的执行时可能由于扫描周期或在堆栈的位置会延迟几个扫描周期 使用绝对位移功能块，没有回原点之前，如果PTOSIMPLE报错误，请检查极限限位开关，这个限位接常开信号；可以回原点，但是不能运行绝对位移功能块
注意事项	在PTO的配置中一定要启用AUX和PROX，否则不能回原点 在PTO_PWM.PTO00.FreeCmdBuf为True时才能把第二条PTO指令发送到PTO的缓存中，否则PTO报错，PTO停止发送脉冲。 如果绝对位置的目标值没有改变，在PTO_PWM.PTO00.FreeCmdBuf为True时频繁、快速的触发绝对位置功能块，PTO会出错，只能重启PLC

9.3.3 PTO-Buffer模式(PTOMoveAbsolute and PTOMoveVelocity)

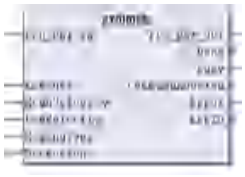
操作符	PTOMoveAbsolute and PTOMoveVelocity BufferMode		
功能说明	绝对位移与速度模式之间采用缓存运行模式		
用到的功能块1			
用到的功能块2			
用到的功能块3			
用到的功能块4			
用到的功能块5			
功能块管脚定义	参见PTOSIMPLE章节 参见PTOHome章节 参见PTOSetPosition章节 参见PTOMoveAbsolute章节 参见 PTOMoveVelocity章节		
	VAR_GLOBAL		
	PTOMoveVelocity_1	: PTOMoveVelocity;	
	PTOMoveAbsolute_1	: PTOMoveAbsolute;	
	PTOHome_1	: PTOHome;	
	PTOSetPosition_1	: PTOSetPosition;	
	Execute_1	:BOOL;	//1#功能块启动命令
	Execute_2	:BOOL;	//2#功能块启动命令
	PTO00_Reset	:BOOL;	//复位PTO错误
	Home_Enanle	:BOOL;	//寻原点启动
	END_VAR		

CFC语言示例

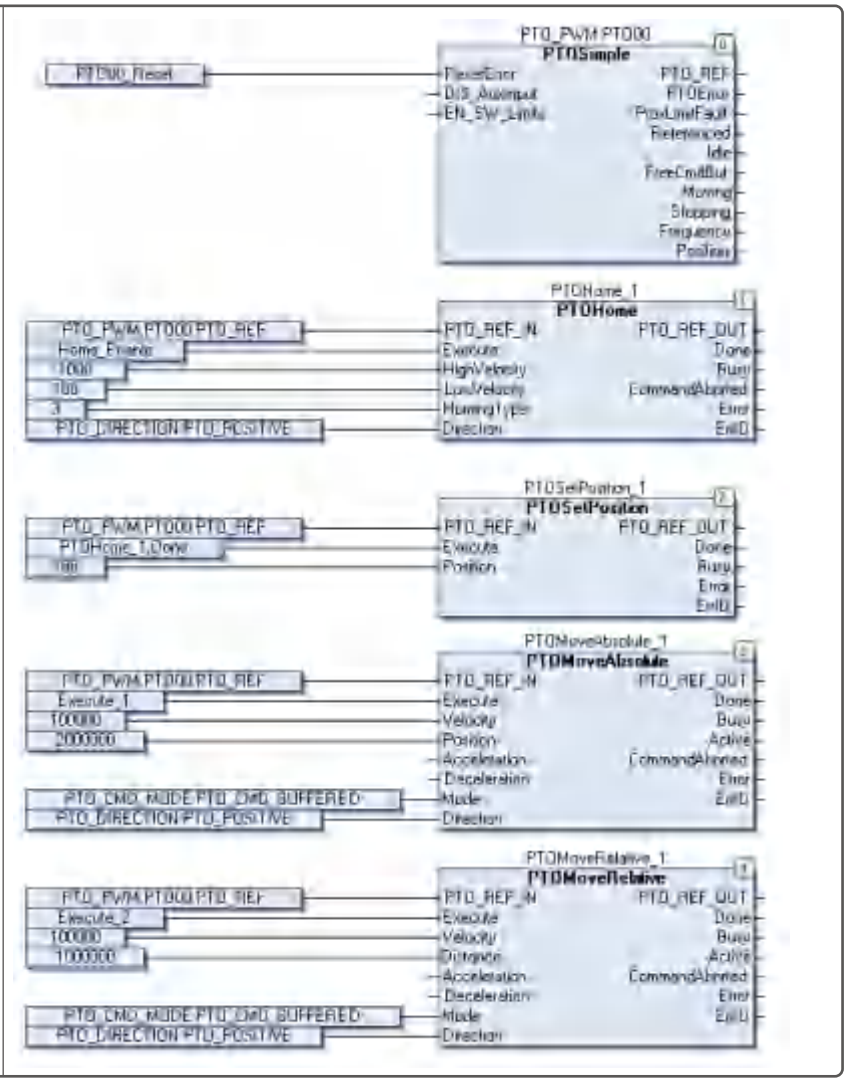

PTO_PWM.PTO00
PTOSimple
PTO00_Reset
ResetError
Dis_Auxinput
EN_SW_Limits
PTO_REF
PTOError
PosLimitFault
ReferenceSet
Idle
FreeCmdRut
Moving
Stopping
Frequency
Position
PTOHome_1
PTOHome
PTO_PWM.PTO00.PTO_REF
Home_Enable
1000
100
3
PTO_DIRECTION.PTO_POSITIVE
PTO_REF_IN
Execute
HighVelocity
LowVelocity
HomingType
Direction
PTO_REF_OUT
Done
Busy
CommandAborted
Error
ErrID
PTOSetPosition_1
PTOSetPosition
PTO_PWM.PTO00.PTO_REF
PTOHome_1.Done
100
PTO_REF_IN
Execute
Position
PTO_REF_OUT
Done
Busy
Error
ErrID
PTOMoveAbsolute_1
PTOMoveAbsolute
PTO_PWM.PTO00.PTO_REF
Execute_1
100
1000
PTO_CMD_MODE.PTO_CMD_BUFFERED
PTO_DIRECTION.PTO_POSITIVE
PTO_REF_IN
Execute
Velocity
Position
Acceleration
Deceleration
Mode
Direction
PTO_REF_OUT
Done
Busy
Active
CommandAborted
Error
ErrID
PTOMoveVelocity_1
PTOMoveVelocity
PTO_PWM.PTO00.PTO_REF
Execute_2
1000
PTO_CMD_MODE.PTO_CMD_BUFFERED
PTO_DIRECTION.PTO_POSITIVE
PTO_REF_IN
Execute
Velocity
Acceleration
Deceleration
Mode
Direction
PTO_REF_OUT
InVelocity
Busy
CommandAborted
Error
ErrID

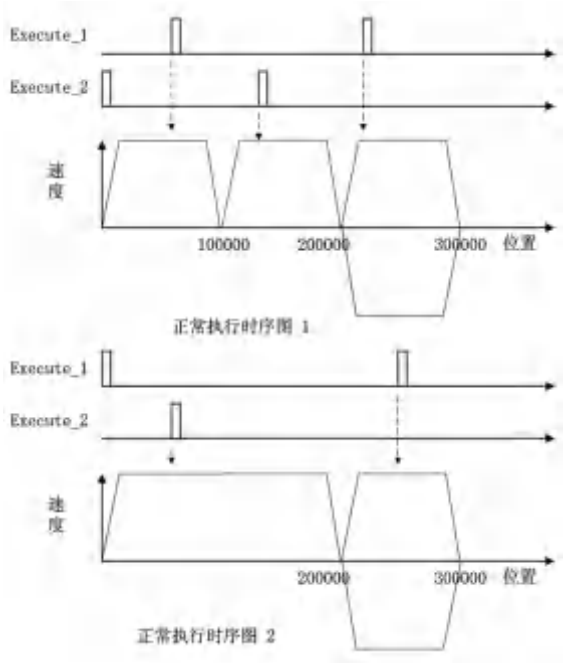
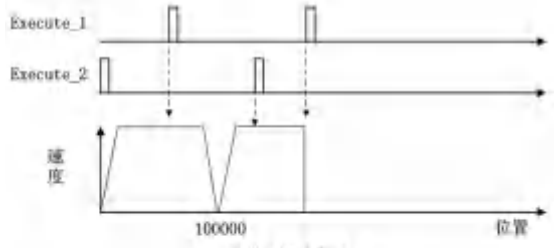
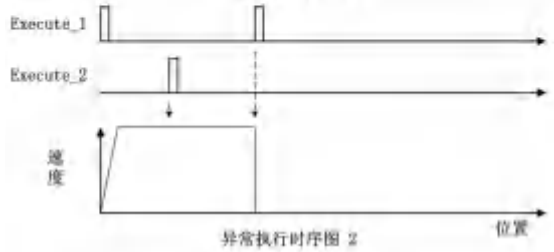
时序图	
使用限制	速度模式下不能使用缓存模式。如果在速度模式下使用缓存模式则PTO出错，PTO停止发送脉冲 PTO命令是依照先进先出的原则，所以功能块的执行时可能由于扫描周期或在堆栈的位置会延迟几个扫描周期 使用绝对位移功能块，没有回原点之前，如果PTOSIMPLE报错误，请检查极限限位开关，这个限位接常开信号；可以回原点，但是不能运行绝对位移功能块
注意事项	在PTO的配置中一定要启用AUX和PROX，否则不能回原点 在PTO_PWM.PTO00.FreeCmdBuf为True时才能把第二条PTO指令发送到PTO的缓存中，否则PTO报错，PTO停止发送脉冲。

9.3.3 PTO-Buffer模式(PTOMoveAbsolute and PTOMoveRelative)

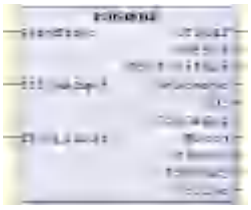
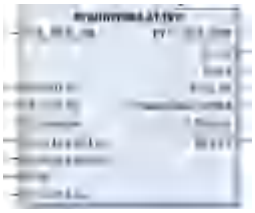
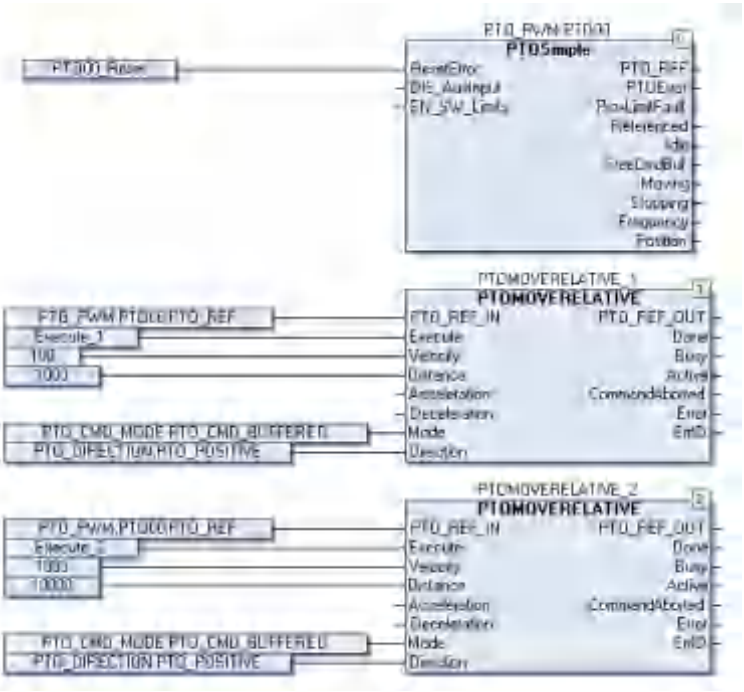
操作符	PTOMoveAbsolute and PTOMoveRelative BufferMode		
功能说明	绝对位移与相对位移之间采用缓存运行模式		
用到的功能块1			
用到的功能块2			
用到的功能块3			
用到的功能块4			
用到的功能块5			
功能块管脚定义	参见PTOSIMPLE章节 参见PTOHome章节 参见PTOSetPosition章节 参见PTOMoveAbsolute章节 参见PTOMoveRelative章节		
	VAR_GLOBAL		
	PTOMoveRelative_1	: PTOMoveRelative;	
	PTOMoveAbsolute_1	: PTOMoveAbsolute;	
	PTOHome_1	: PTOHome;	
	PTOSetPosition_1	: PTOSetPosition;	
	Execute_1	:BOOL;	//1#功能块启动命令
	Execute_2	:BOOL;	//2#功能块启动命令
	PTO00_Reset	:BOOL;	//复位PTO错误
	Home_Enanle	:BOOL;	//寻原点启动
	END_VAR		

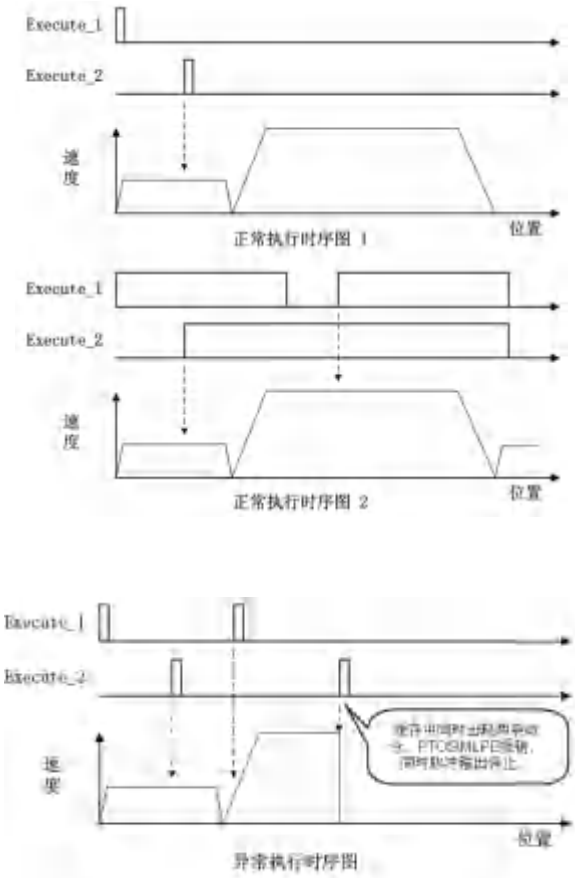
CFC语言示例


PTO_PWM.PTO00
PTOSimple
PTO00_Reset
PTO_REF
PTOError
PosLimitFault
Referenced
Idle
FreeCmdBuf
Moving
Stopping
Frequency
PosTeach
PTOHome_1
PTOHome
PTO_PWM.PTO00.PTO_REF
Home_Enable
1000
100
3
PTO_DIRECTION.PTO_POSITIVE
PTO_REF_IN
Execute
HighVelocity
LowVelocity
HomingType
Direction
PTO_REF_OUT
Done
Busy
CommandAborted
Error
ErrID
PTOSetPosition_1
PTOSetPosition
PTO_PWM.PTO00.PTO_REF
PTOHome_1.Done
100
Position
PTOMoveAbsolute_1
PTOMoveAbsolute
PTO_PWM.PTO00.PTO_REF
Execute_1
100000
2000000
Velocity
Acceleration
Deceleration
PTO_CMD_MODE.PTO_CMD_BUFFERED
Mode
PTOMoveRelative_1
PTOMoveRelative
PTO_PWM.PTO00.PTO_REF
Execute_2
100000
1000000
Distance

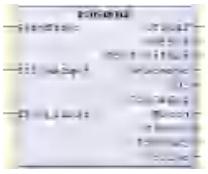
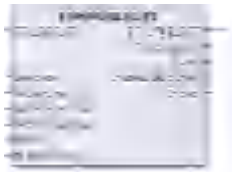
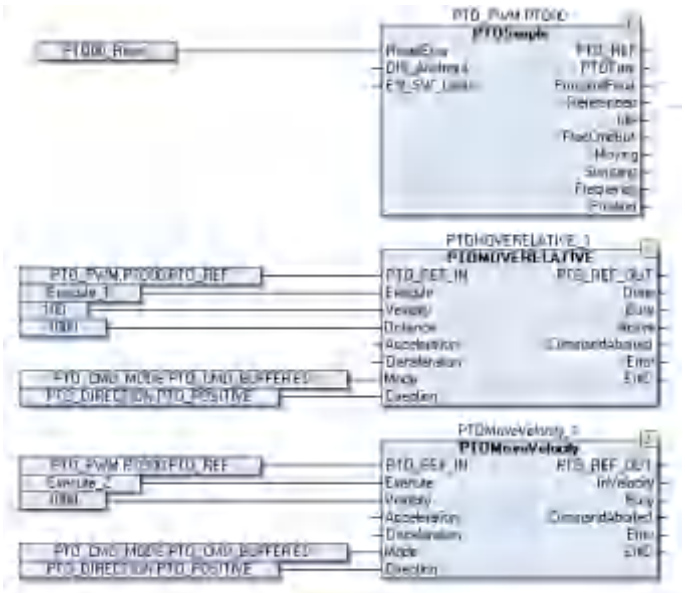
时序图	 正常执行时序图 1 正常执行时序图 2  异常执行时序图 1  异常执行时序图 2
使用限制	PTO命令是依照先进先出的原则，所以功能块的执行时可能由于扫描周期或在堆栈的位置会延迟几个扫描周期 使用绝对位移功能块，没有回原点之前，如果PTOSIMPLE报错误，请检查极限限位开关，这个限位接常开信号；可以回原点，但是不能运行绝对位移功能块
注意事项	在PTO的配置中一定要启用AUX和PROX，否则不能回原点 在PTO_PWM.PTO00.FreeCmdBuf为True时才能把第二条PTO指令发送到PTO的缓存中，否则PTO报错，PTO停止发送脉冲。 如果绝对位置的目标值没有改变，在PTO_PWM.PTO00.FreeCmdBuf为True时频繁、快速的触发绝对位置功能块，PTO会出错，只能重启PLC

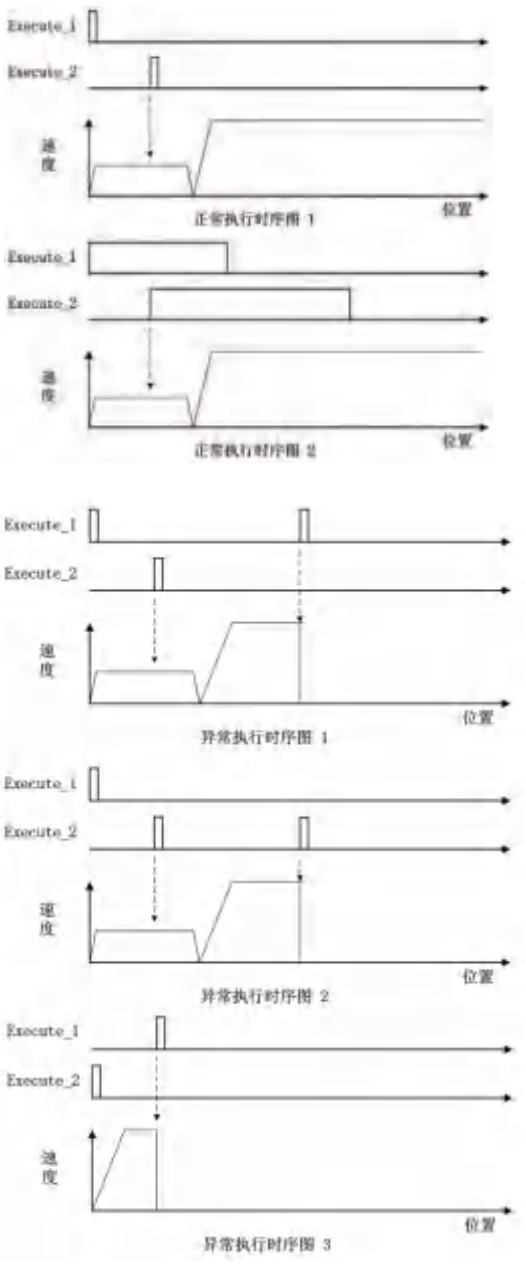
9.3.3 PTO-Buffer模式(PTOMoveRelative and PTOMoveRelative)

操作符	PTOMoveRelative and PTOMoveRelative BufferMode		
功能说明	多个相对位移的点到点之间采用缓存运行模式		
用到的功能块1			
用到的功能块2			
功能块管脚定义	参见PTOSIMPLE章节 参见PTOMoveRelative章节		
	VAR_GLOBAL		
	PTOMOVERELATIVE_1	: PTOMOVERELATIVE;	
	PTOMOVERELATIVE_2	: PTOMOVERELATIVE;	
	Execute_1	:BOOL;	//1#功能块启动命令
	Execute_2	:BOOL;	//2#功能块启动命令
	PTO00_Reset	:BOOL;	//复位PTO错误
	END_VAR		
CFC语言示例			

时序图	
使用限制	PTO缓存堆栈中只能为空或一个，如果超过一个则PTOSIMPLE功能块的“PTOError”引脚变为“TRUE”，对应的PTO通道停止发送脉冲，只有通过复位PTOSIMPLE功能块或PLC重启PTO通道才能再发送脉冲。 PTO命令是依照先进先出的原则，所以功能块的执行时可能由于扫描周期或在堆栈的位置会延迟几个扫描周期
注意事项	

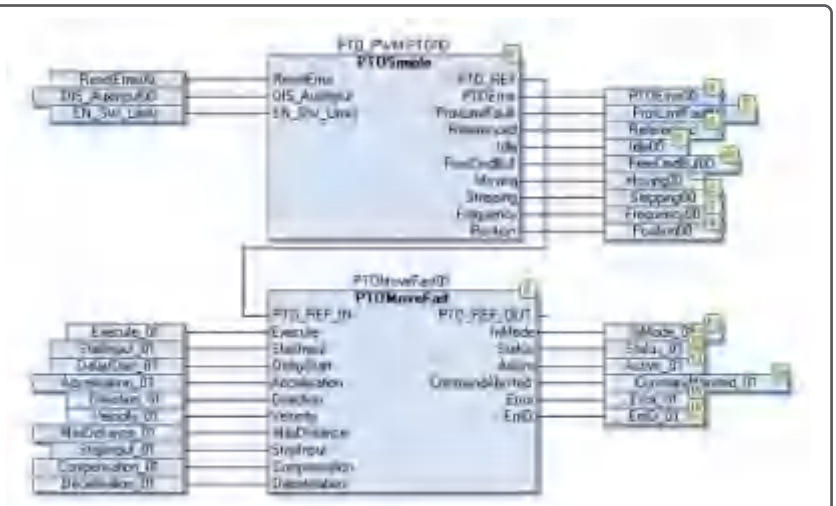
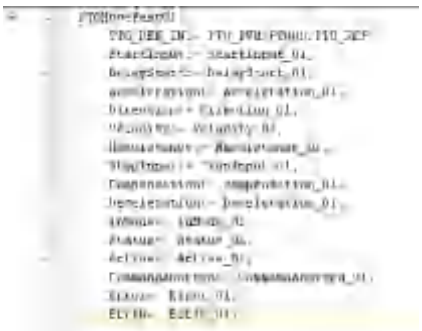
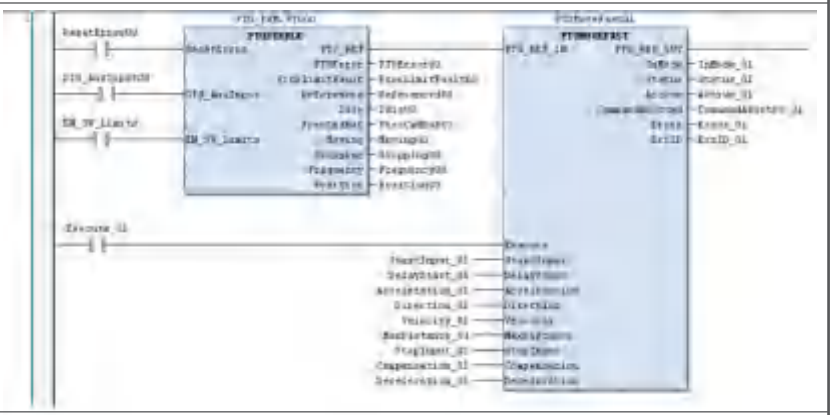
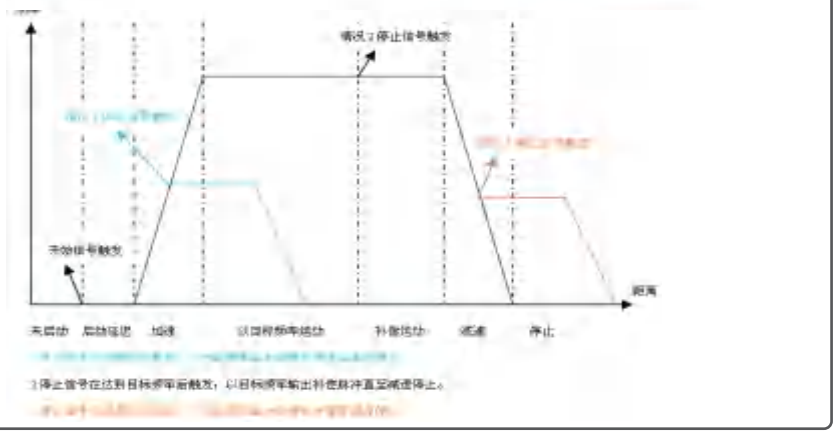
9.3.3 PTO-Buffer模式(PTOMoveRelative and PTOMoveVelocity)

操作符	PTOMoveRelative and PTOMoveVelocity BufferMode		
功能说明	相对位移与速度模式之间采用缓存运行模式		
用到的功能块1			
用到的功能块2			
用到的功能块3			
功能块管脚定义	参见PTOSIMPLE章节 参见PTOMoveRelative章节 参见 PTOMoveVelocity章节		
	VAR_GLOBAL		
	PTOMOVERELATIVE_1	: PTOMOVERELATIVE;	
	PTOMoveVelocity_1	: PTOMoveVelocity;	
	Execute_1	:BOOL;	//1#功能块启动命令
	Execute_2	:BOOL;	//2#功能块启动命令
	PTO00_Reset	:BOOL;	//复位PTO错误
	END_VAR		
CFC语言示例			

时序图	
使用限制	速度模式下不能使用缓存模式。如果在速度模式下使用缓存模式则PTO出错，PTO停止发送脉冲通过复位PTOSIMPLE功能块。 PTO命令是依照先进先出的原则，所以功能块的执行时可能由于扫描周期或在堆栈的位置会延迟几个扫描周期
注意事项	

9.4.1 PTOMovefast

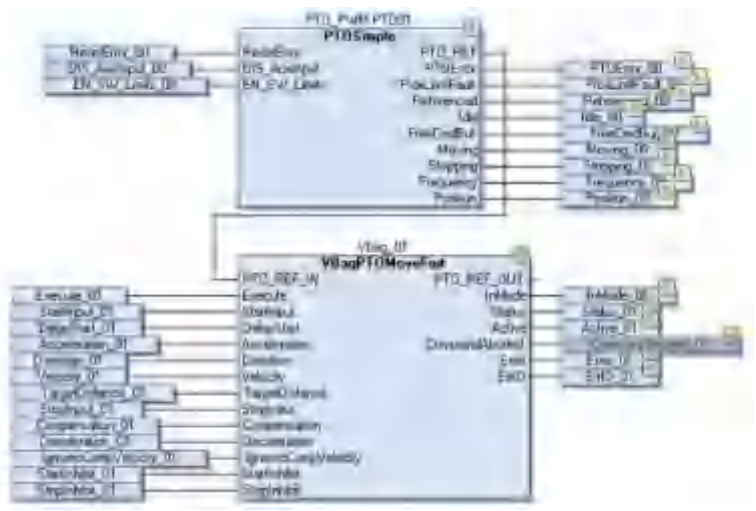
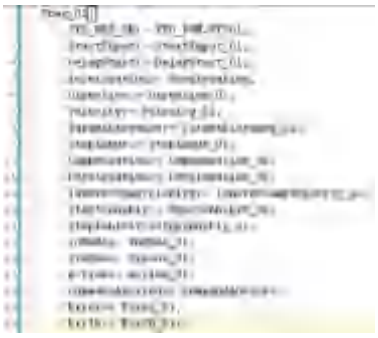
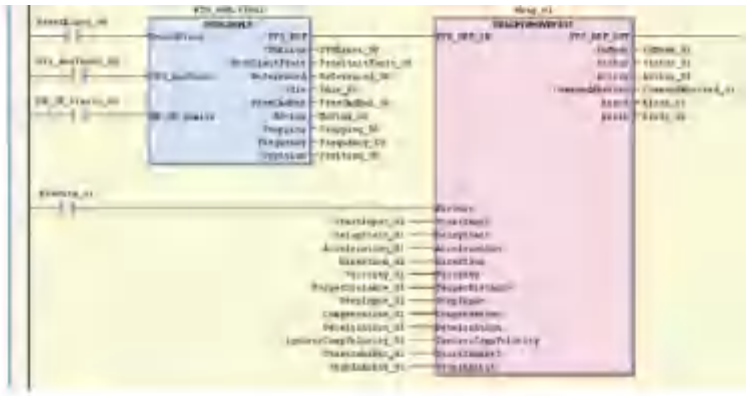
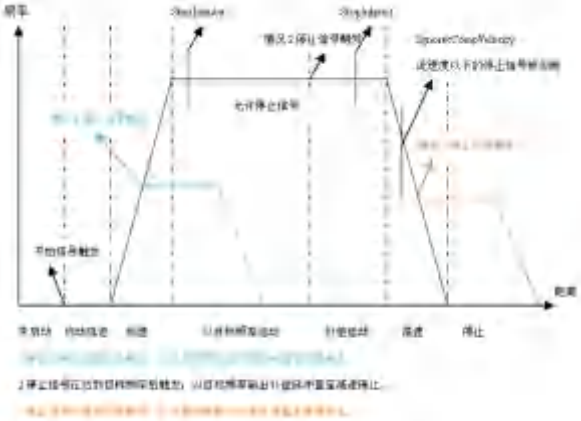
操作符	PTOMovefast
功能说明	此功能块由外部数字量信号触发，恒定频率持续输出脉冲。 适用于需要快速启停及精确位置补偿的应用。 功能块由应用库PTO/PWM.Library中Motion提供。 该功能块由外部物理输入触发启动（启动信号可配置成IN_IMM类型，此时由Execute引脚触发启动）和停止。启动信号触发后频率由0开始根据设定的加速度增加到目标频率。停止信号触发后，功能块开始输出位置补偿值，并根据设定的减速度，在输出完位置补偿值时减速到0。功能块的执行状态可通过输出管脚Status,Active,Error的状态来判断。
图形	
管脚定义	输入 类型 注释 PTO_REF_IN PTO_REF 对PTO的参考输入。连接到PTOSimple的PTO_REF,或管理/运动功能块的输出引脚PTO_REF_OUT。 Execute BOOL 功能块使能输入，有上升沿触发。 StartInput FAST_PTO_TRIGGER 外部启动输入，触发运动执行。也可配制成内部输入（IN_IMM），此时由Execute引脚启动。 DelayStart DWORD 当外部启动输入信号为True时，延迟运动的执行。范围：0...1999ms Acceleration DWORD 以Hz/ms或ms为单位的加速度值。（单位由PTO配置决定） Direction PTO_DIRECTION 运动方向。 Velocity DWORD 以Hz为单位的目标速度。范围：1Hz...配置的最大频率 MaxDistance DWORD 运动执行的最大保护距离，到达此距离时，功能块会输出 StopInput FAST_PTO_TRIGGER 只能配置成外部输入，外部停止输入为True时，开始输出位置补偿脉冲。 Compensation DWORD 位置补偿脉冲值。 Deceleration DWORD 以Hz/ms或ms为单位的减速度值。（单位由PTO配置决定） 输出 类型 注释 PTO_REF_OUT PTO_REF 对PTO的参考输出。连接到管理/运动功能块的输入引脚PTO_REF_IN。 InMode BOOL TRUE = 功能块已经使能，可由外部输入触发执行。 Status BYTE 提供运动状态。0.运动停止； 1.处于启动延时阶段；2.处于加速阶段；3.以目标频率运动；4.输出补偿频率；5.处于减速阶段；6.达到最大距离减速停止。 Active BOOL TRUE = 功能块已由外部输入触发，正在执行。 CommendAborted BOOL TRUE = 由于其他运动功能块正在执行，本功能块的执行请求被忽略。 Error BOOL TRUE = 检测到错误，功能块执行结束。 ErrID PTOPMW_ERR_TYPE 当Error为TRUE时检测到的错误类型。
错误代码	枚举器 / 值（16进制） / 说明 NO_ERROR / 00 / 未检测到错误。 PTO_UNKNOW_REF / 01 / 未知轴参考或配置不当的轴。（例如PTO_REF_IN引脚未连接到PTOSimple的PTO_REF,或管理/运动功能块的输出引脚PTO_REF_OUT） PTO_UNKNOW_PARAMETER / 02 / 未知参数类型。对于此功能块无效。 PTO_INVALID_PARAMETER / 03 / 用于所请求运动的无效参数值或不正确的参数值组合。（例如加减速配置不正确，外部启停信号配置不正确） PTO_COM_ERROR / 04 / 检测到 PTO 接口通讯错误。（例如功能块运行中再次触发Execute） PTO_AXIS_ERROR / 05 / 检测到轴错误（例如在已报错的情况未复位，再次执行一个PTO功能块）。 PTO_CMD_ERROR / 06 / 缓冲区已满。（StartInput是IN_IMM时，重复执行Execute时会产生） PTO_LIMIT_FAULT / 07 / 回归模式中出现近似错误。（例如PTO的Position值超出软件限位）
变量声明	PROGRAM POU_1 VAR ResetError00: BOOL; DIS_AuxInput00: BOOL; EN_SW_Limits: BOOL; PTOError00: BOOL; ProxLimitFault00: BOOL; Referenced00: BOOL; Idle00: BOOL; FreeCmdBuf00: BOOL; Moving00: BOOL; Stopping00: BOOL; Frequency00: DWORD; Position00: DINT; Execute_01: BOOL; StartInput_01: FAST_PTO_TRIGGER; DelayStart_01: DWORD; Acceleration_01: DWORD; Direction_01: PTO_DIRECTION; Velocity_01: DWORD; MaxDistance_01: DWORD; StopInput_01: FAST_PTO_TRIGGER; Compensation_01: DWORD; Deceleration_01: DWORD; InMode_01: BOOL; Status_01: BYTE; Active_01: BOOL; CommandAborted_01: BOOL; Error_01: BOOL; ErrID_01: PTOPWM_ERR_TYPE; PTOMoveFast01: PTOMOVEFAST; END_VAR

CFC语言示例	
ST语言示例	
LD语言示例	
时序图	

使用限制	·StartInput配置成外部输入时（StopInput必须配置成外部输入），必须使用以下步骤配置StartInput和StopInput管脚： 程序中每个功能块必须配置两个数字量输入点（I0...I3）用于启动和停止运动。这两个输入点仅服务于此功能块且必须在任务配置中配置成外部事件模式，并在I/O表中将这两个输入点的事件选项配置成上升沿，下降沿或者上升下降沿。 ·运动执行过程中不能通过修改velocity引脚的值来改变当前速度。 ·功能块在Execute上升沿触发后永远被使能。
注意事项	·当功能块在执行过程中,不得改变PTO_REF_IN的输入。 ·在不同的任务中,不能使用相同的功能块实例。 ·在同一个应用(Application)中，如果两个以上的管理类型或者运动类型的功能块被同时调用，该功能块将会在执行结束之后输出一个COM_ERROR错误类型。（意思指请求队列已满） ·功能块需要分配两个数字量输入点（I0...I3）用于启动停止才能编译通过。 ·FAST_PTO_TRIGGER类型的输入值（StartInput可以配置成IN_IMM）只能为IN_I0,IN_I1,IN_I2,IN_I3，分别对应于数字量输入点I0,I1,I2,I3。 ·StartInput管脚可以配置成IN_IMM类型，当配置成次类型时，功能块由Execute上升沿触发。
	输入变量管理： ·该功能块在 Execute 置为TRUE的上升沿后使能，在StartInput的上升沿被启动。 ·在功能块执行过程中输入变量无法进行任何修改。 ·按照 IEC 61131-3 标准，如果功能块有任何变量输入缺失（即断开或未连接），则使用上一次调用功能块实例的值。在此情况下，第一次调用时将应用初始配置值。因此，功能块最好始终带有特定于其输入的已知值，这样有助于消除调试程序的麻烦。对于 HSC 和 PTO 功能块，最好只使用一次实例，且该实例必须位于主任务中。 ·MaxDistance管脚为最大运动距离，起软件极限保护作用。 ·Compensation管脚为StopInput信号触发后需要输出的位置脉冲数，它和已发完脉冲数的总和应小于Maxdistance定义的脉冲数。
	错误处理： 所有功能块都有 2 个输出，可以报告在执行功能块期间检测到的错误。 · 检测到错误时，Error = TRUE。 · ErrID 在 Error = TRUE 时返回检测到的错误 ID。

9.4.1 VbagPTOMovefast

操作符	VbagPTOMovefast
功能说明	此功能块由外部数字量信号触发，恒定频率持续输出脉冲，适用于包装机应用。 由应用库Packaging.Library中Motion提供。 该功能块由外部物理输入触发启动（启动可配置成IN_IMM，并由Execute引脚启动）和停止。启动信号触发后频率由0开始根据设定的加速度增加到目标频率。当停止信号处于StartInhibit和StopInhibit之间可被执行，停止信号触发后，功能块开始输出补偿值，并根据设定的减速度，在输出完补偿值时减速到0。当IgnorecCompVelocity的值大于停止信号触发时的当前频率值，此停止信号被忽略。功能块的执行状态可通过输出管脚Status,Active,Error的状态来判断。
图形	
管脚定义	输入 类型 注释 PTO_REF_IN PTO_REF 对PTO的参考。连接到PTOSimple的PTO_REF,或管理或运动输出引脚功能块的PTO_REF_OUT。 Execute BOOL 在上升沿后功能块可被外部输入触发执行。 StartInput FAST_PTO_TRIGGER 外部物理输入，触发功能块执行。也可配制成内部输入（IN_IMM），此时由Execute引脚启动。 DelayStart DWORD 当外部开始信号为True时，延迟功能块的启动。范围：0...1999ms Acceleration DWORD 以Hz/ms或ms为单位的加速度值。（单位由PTO配置决定） Direction PTO_DIRECTION 运动方向。 Velocity DWORD 以Hz为单位的目标速度。范围：1Hz...配置的最大频率 TargetDistance DWORD 本次运动执行的最大距离。 StopInput FAST_PTO_TRIGGER 外部输入为True时，开始输出补偿脉冲。 Compensation DWORD 补偿脉冲。 Deceleration DWORD 以Hz/ms或ms为单位的减速度值。（单位由PTO配置决定） IgnoreCompVelocity DWORD 除加速阶段外，当停止信号输入时，如果当前频率小于此值，则停止信号被忽略，运动将按原定曲线完成。 StartInhibit DWORD 允许执行停止信号的起始点，发生于此点之后的停止信号可以被执行。必须小于StopInhibit值。 StopInhibit DWORD 允许执行停止信号的终止点，发生于此点之前的停止信号可以被执行。必须大于StartInhibit值。 输出 类型 注释 PTO_REF_OUT PTO_RE 对PTO的参考。连接到管理或运动输出引脚功能块的PTO_REF_IN输入引脚。 InMode BOOL TRUE = 功能块已经使能，可由外部输入触发执行。 Status BYTE 提供运动状态。0.运动停止；1.处于启动延时阶段；2.处于加速阶段；3.以目标频率运动；4.输出补偿频率；5.处于减速阶段；6.达到最大距离减速停止。 Active BOOL TRUE = 功能块已由外部输入触发，正在执行。 CommendAborted BOOL TRUE = 由于其他运动功能块已触发，本功能块的执行请求被忽略。 Error BOOL TRUE = 检测到错误，功能块执行结束。 ErrID PTOPMW_ERR_TYPE 当Error为TRUE时检测到的错误类型。
错误代码	枚举器 / 值（16进制） / 说明 NO_ERROR 00 未检测到错误。 PTO_UNKNOW_REF 01 未知轴参考或配置不当的轴。（例如PTO_REF_IN引脚未连接到PTO-Simple的PTO_REF,或管理/运动功能块的输出引脚PTO_REF_OUT） PTO_UNKNOW_PARAMETER 02 未知参数类型。对于此功能块无效。 PTO_INVALID_PARAMETER 03 用于所请求运动的无效参数值或不正确的参数值组合。（例如加减速配置不正确，外部启停信号配置不正确） PTO_COM_ERROR 04 检测到 PTO 接口通讯错误。（例如功能块运行中再次触发Execute） PTO_AXIS_ERROR 05 检测到轴错误（例如在已报错的情况未复位，再次执行一个PTO功能块）。 PTO_CMD_ERROR 06 缓冲区已满。（StartInput是IN_IMM时，重复执行Execute时会产生） PTO_LIMIT_FAULT 07 回归模式中出现近似错误。（例如PTO的Position值超出软件限位）
变量声明	VAR ResetError_00: BOOL; DIS_AuxInput_00: BOOL; EN_SW_Limits_00: BOOL; PTOError_00: BOOL; ProxLimitFault_00: BOOL; Referenced_00: BOOL; Idle_00: BOOL; FreeCmdBuf_00: BOOL; Moving_00: BOOL; Stopping_00: BOOL; Frequency_00: DWORD; Position_00: DINT; Vbag_01: VBAGPTOMOVEFAST; Execute_01: BOOL; StartInput_01: FAST_PTO_TRIGGER; DelayStart_01: DWORD; Acceleration_01: DWORD; Direction_01: PTO_DIRECTION; Velocity_01: DWORD; TargetDistance_01: DWORD; StopInput_01: FAST_PTO_TRIGGER; Compensation_01: DWORD; Deceleration_01: DWORD; IgnorecCompVelocity_01: DWORD; StartInhibit_01: DWORD; StopInhibit_01: DWORD; InMode_01: BOOL; Status_01: BYTE; Active_01: BOOL; CommandAborted_01: BOOL; Error_01: BOOL; ErrID_01: PTOPWM_ERR_TYPE; END_VAR

CFC语言示例	
ST语言示例	
LD语言示例	
时序图	

使用限制	·StartInput配置成外部输入时（StopInput必须配置成外部输入），必须使用以下步骤配置StartInput和StopInput管脚： 程序中每个功能块必须配置两个数字量输入点（I0...I3）用于启动和停止功能块。这两个输入点仅服务于此功能块且必须在任务配置中配置成外部事件模式，并在I/O表中将这两个输入点的事件选项配置成上升沿，下降沿或者上升下降沿。 ·运动执行过程中不能通过修改velocity引脚的值来改变当前速度。 ·功能块在Execute上升沿触发后永远被使能。
注意事项	·当功能块在执行过程中,不得改变PTO_REF_IN的输入。 ·在不同的任务中,不能使用相同的功能块实例。 ·在同一个应用(Application)中，如果有太多的管理类型或者运动类型的功能块被同时调用，该功能块将会在执行结束之后输出一个COM_ERROR错误类型。（意思指请求队列已满） ·功能块需要分配一个数字量输入点（I0...I3）用于StopInput才能编译通过。 ·StartInput可以使用IN_IMM类型，此时功能块由Execute引脚的上升沿触发。 ·FAST_PTO_TRIGGER类型的输入值只能为IN_I0,IN_I1,IN_I2,IN_I3（StartInput管脚可以是IN_IMM），分别对应于数字量输入点I0,I1,I2,I3。
	输入变量管理： ·该功能块在 Execute 置为TRUE的上升沿后可被启动，在StartInput的上升沿被启动。 ·此时无需对输入变量进行任何进一步的修改。 ·按照 IEC 61131-3 标准，如果功能块有任何变量输入缺失（即断开或未连接），则使用上一次调用功能块实例的值。在此情 况下，第一次调用时将应用初始配置值。因此，功能块最好始终带有特定于其输入的已知值，这样有助于消除调试程序的麻烦。对于 HSC 和 PTO 功能块，最好只使用一次实例，且该实例必须位于主任务中。 ·MaxDistance管脚为最大运动距离，起软件极限保护作用。 ·Compensation管脚为StopInput信号触发后需要输出的脉冲数，它和已发完脉冲数的总和应小于Maxdistance定义的脉冲数。 ·IgnorecCompVelocity管脚定义可执行停止信号的最小速度，当前速度小于此管脚的速度时，停止信号被忽略，功能块将自动执行完剩余的补偿脉冲。 ·StartInhibit和StopInhibit定义了Stop信号可被执行的区间，StartInhibit必须大于0且小于StopInhibit值，StopInhibit必须大于StartInhibit且小于MaxDistance，否则会报错，错误代码为PTO_INVALID_PARAMETER。
	错误处理： 所有功能块都有 2 个输出，可以报告在执行功能块期间检测到的错误。 ·检测到错误时，Error = TRUE。 ·ErrID 在 Error = TRUE 时返回检测到的错误 ID。

附录

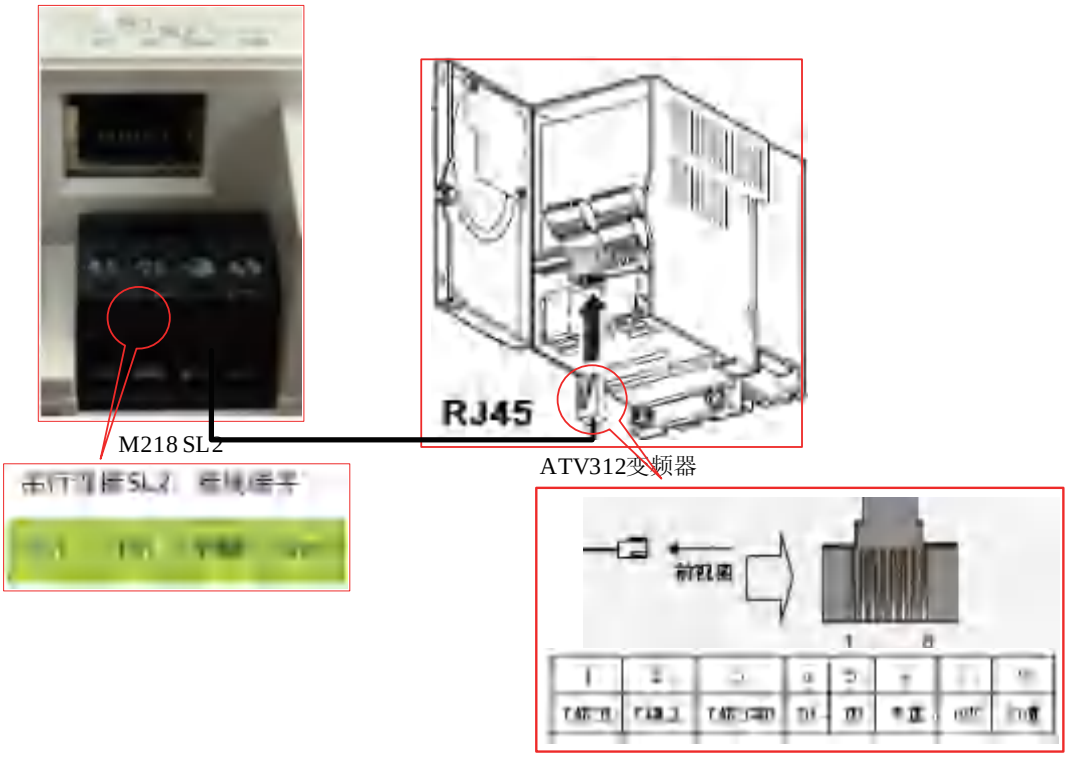
附1：串口Modbus通讯示例

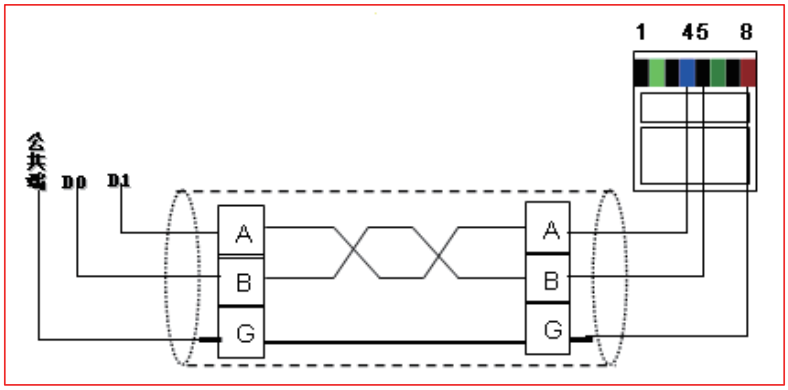
内容简介

PLC通过Modbus监控变频器的运行是工业中较常见的应用,本文以施耐德M218PLC与ATV312变频器为例, 简要介绍PLC与变频器之间Modbus串行通信的过程, 包括硬件接线、变频器参数设置、硬软件组态、上电调试等，实现在PLC上远程控制ATV312变频器的故障初始化，启动/停止，正转/反转，频率给定等。

硬件连接

M218 SL2的螺钉端子与双绞线的裸露端连接，双绞线的RJ45端接到ATV312变频器，系统的硬件构架和连接如下：





接线图

图2-1

附1：串口Modbus通讯示例

备注：M218控制器内集成了2个RS485串口，1个端口（SL1）位RJ45口，另一个端口（SL2）为端子排接口，SL1可用来与HMI通讯，SL2可用来与变频器等通讯。

ATV312变频器设置

⑴、控制和给定通道
如果采用Modbus通信控制启停及速度给定，步骤如下：

参数路径	参数说明	值	功能描述
CTL-LAC	功能访问等级	L3	访问高级功能与混合控制模式的管理
CTL-FR1	配置给定 1	ndb	通过 Modbus总线给定
CTL-CHCF	控制模式设置	SIN	如果 LAC=L3 可访问此参数：SIN - 组合，控制和频率给定由同一种方式设定；SEP - 分离，控制和频率给定由不同的方式设定

⑵、通信参数

此设置和PLC中的设置保持一致，步骤如下：

参数路径	参数说明	值	功能描述
COnADD	从站地址	1	范围 1 - 247
COntbr	通信速率	19.2	4.8-4800bps;9.6-9600bps;19.219200
COntfo	通信格式	8E1	8O1:8个数据位, 奇校验,1 个停止位 8E1:8个数据位, 偶校验,1 个停止位 8n1:8个数据位, 无校验,1 个停止位 8n2:8个数据位, 无校验,2 个停止位

注意：设置完毕后，关闭变频器电源再重新上电，参数设置才能生效。

PLC编程

本例中的PLC组态和编程全部采用SoMachine软件平台完成。SoMachine是施耐德电气支持OEM PLC（M258、M238、M218）、HMI、Motion、VSD的通用编程、调试和运行的软件平台。

PLC编程主要包括硬件组态和软件编程两个部分

(1)、组态CPU

步骤	动作&示例
1	打开 SoMachine→创建新机器→使用空项目启动→将项目另存为(例：test)→配置→选择合适的 M218 型号

图2-2

2	进入“程序”界面，进行编程及“PLC 内嵌功能”等的组态

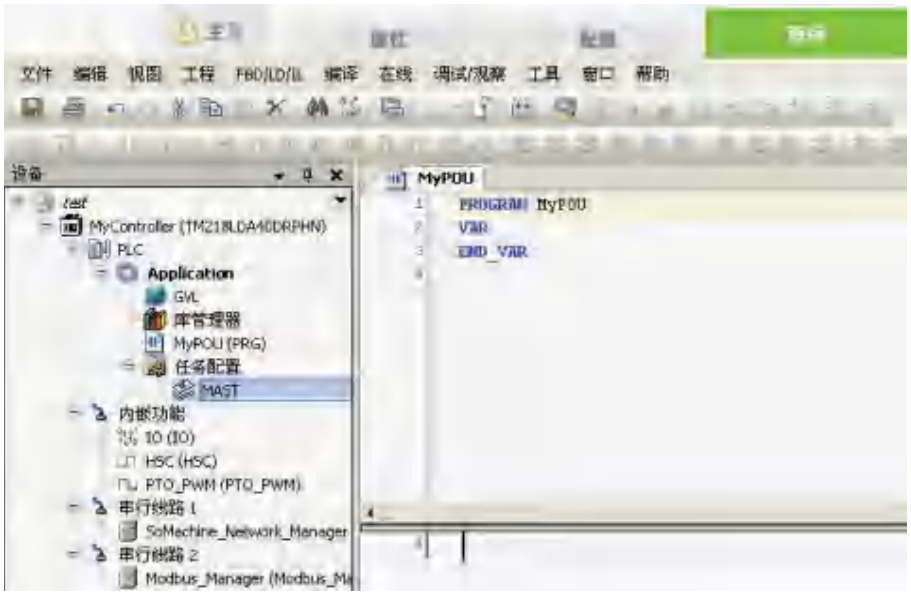
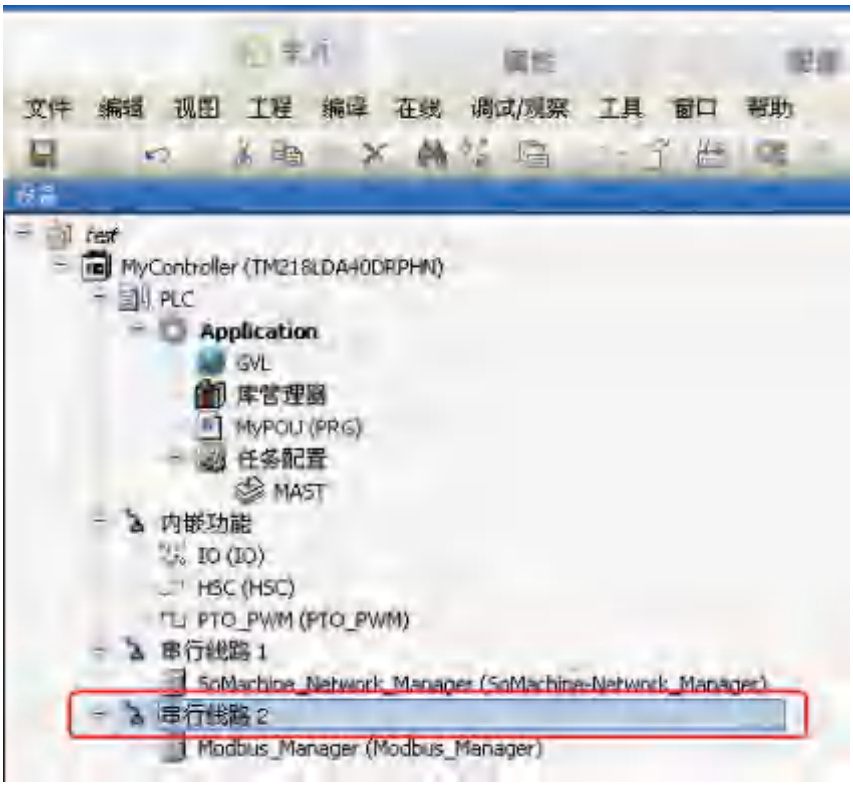


图2-3

⑵、组态Modbus 串口

步骤	动作&示例
1	选中“串行线路 2（SL2）”。“串行线路 2”的缺省通讯协议是“Modbus_Manager”，无需更改  图2-4
2	双击“串行线路 2”，物理参数保证与 ATV312 变频器中的通信参数设置保持一致，如下图所示：  图2-5
3	双击“Modbus_Manager”，配置 Modbus 通讯协议参数，如下图所示：

ATV312 Modbus变量说明

在M218 PLC中编写程序，将变频器的内部变量用功能块Read_var/Write_var映射到本地寄存器，通过对本地寄存器进行读写，来完成对变频器的监控。

⑴、ATV312 Modbus内部字

在此仅列出本例中使用的ATV31内部寄存器以及对应功能,见下表:

类型	地址	代码	说明
读出变量	3201	ETA	DRIVECOM状态字
	3202	RFR	电机输出频率
写入变量	8501	CMD	DRIVECOM控制字
	8502	LFR	通过总线写入变频器的给定频率

对ATV312变频器实现Modbus通信控制的状态字和控制字说明如下（更详细的内容见ATV312 Modbus用户手册）：

位	状态字 ETA(W3201)	控制字 CMD(W8501)
Bit0	准备接通	接通
Bit1	接通	为0时激活
Bit2	操作允许	为0时激活
Bit3	故障	允许操作
Bit4	电压无效	0
Bit5	为0时激活	0
Bit6	接通禁止	0
Bit7	报警	故障复位
Bit8	0	0
Bit9	强制本地模式，为0时激活	0
Bit10	达到给定值	0
Bit11	超过给定值	正转/反转 0为正转
Bit12	0	按停车类型制动
Bit13	0	注入制动
Bit14	STOP 键停止	为1时激活
Bit15	旋转方向	0

⑵ATV312 DRIVERCOM流程

Modbus通讯控制变频器时要遵循DRIVERCOM流程，如下图

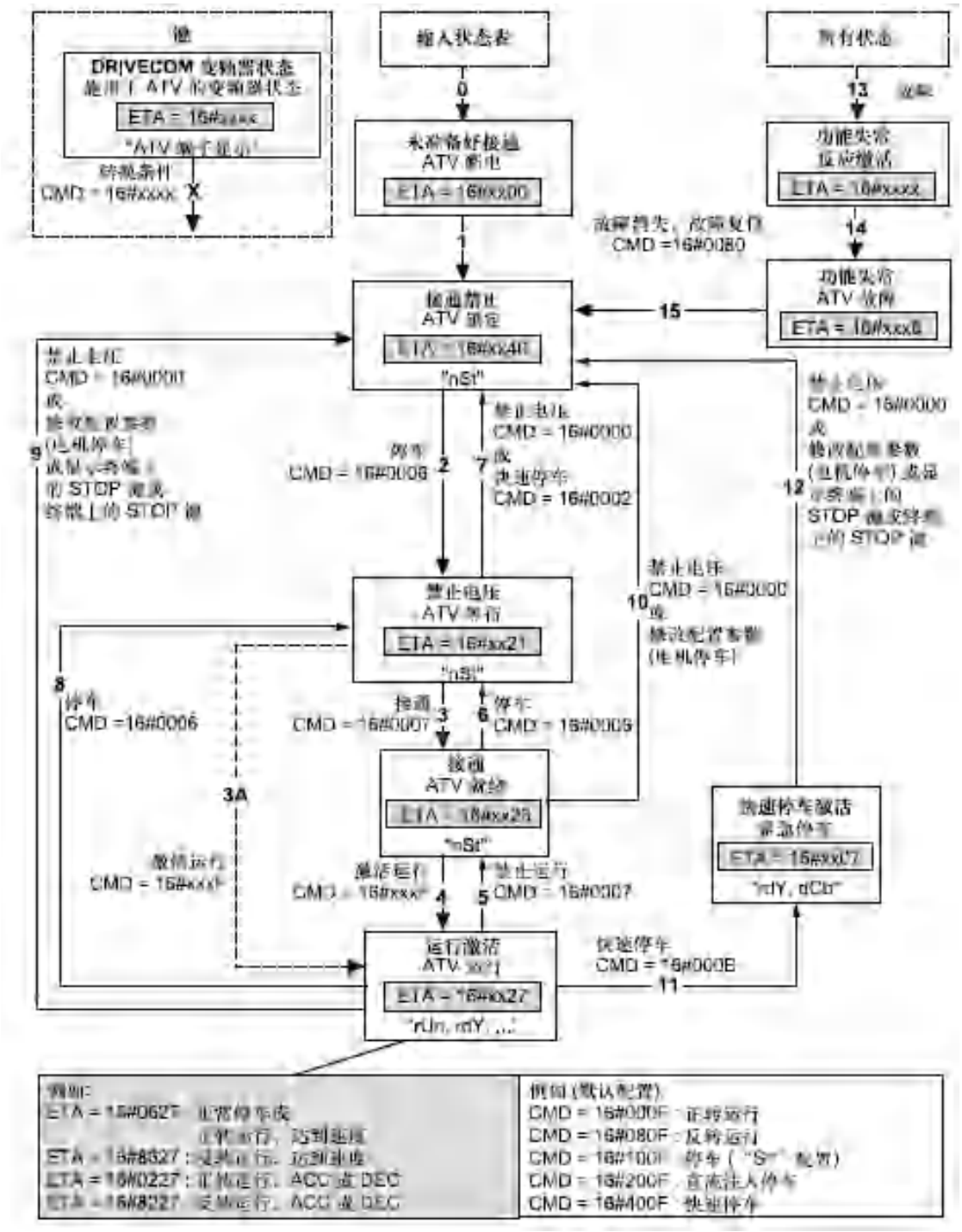


图2-7

编程

⑴Modbus设备的读写可分时读写，也可按依次顺续执行方式编写，例程如下所示（其中Drive-Com_ATV312是自定义
功能块，按DriveCom流程控制ATV312）：

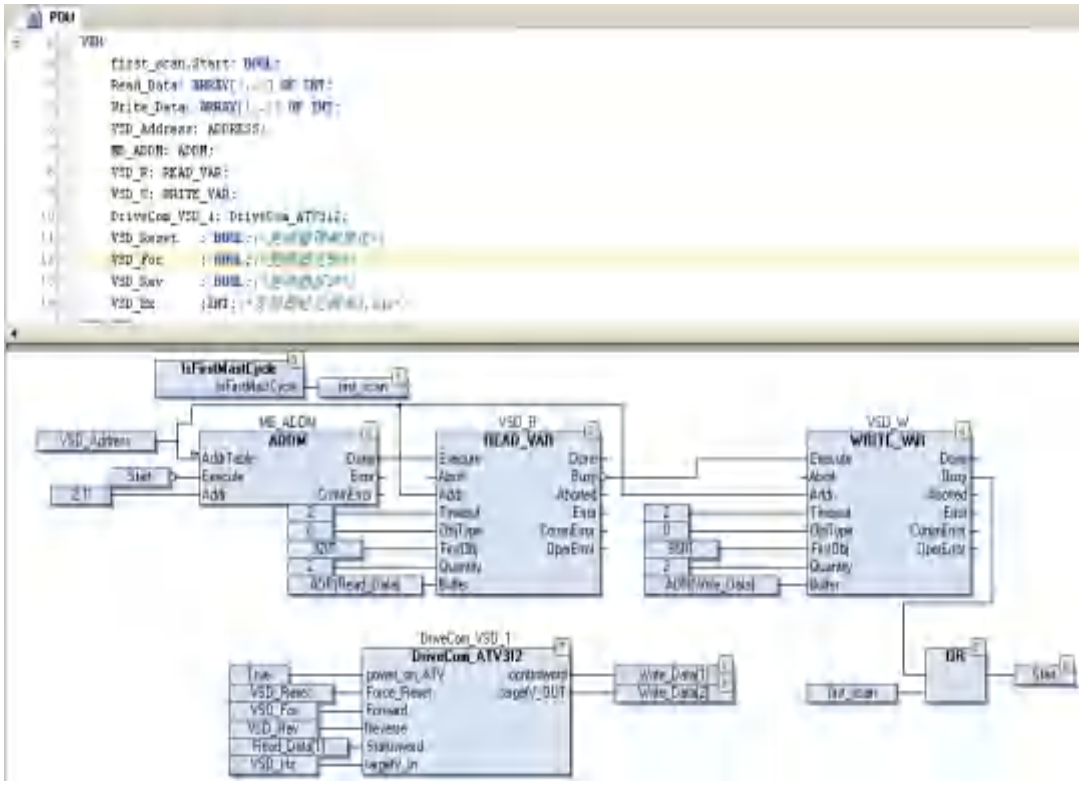


图2-8

自定义功能块DriveCom_ATV312程序代码如下：

图2-9

⑵例程中使用到的功能块简介

①IsFirstMastCycle：系统功能，PLC启动之后的第一个 MAST 任务循环期间为TRUE。功能块的详细说明，参见本手册9章 M218系统库指南。

②ADDM：管理通讯地址表。本例中，Addr 参数中写入“2.1”，其中2 表示PLC通讯端口2，1表示从站设备地址是1。 假如用户使用端口1，从站地址是4，那么可以写成“1.4”。功能块的详细说明，参见本手册8.4.1 通讯库中的ADDM章节。

③ READ_VAR：连续读功能块，从目标设备的寄存器区读取多个数据到功能块的读取数据接收缓冲区（本例为数组Read_Data）。功能块的详细说明，参见本手册8.4.2 通讯库中的READ_VAR章节。

④ WRITE_VAR：连续写功能块，将功能块的写入数据缓冲区（本例为数组Write_Data）中的数据发送到目标设备的寄存器区。功能块的详细说明，参见本手册8.4.3 通讯库中的WRITE_VAR章节。

⑤ ADR：取地址指令。由于“Read_Var”和“Write_Var”功能块的管脚“Buffer”是指针变量，所以用ADR 功能块来取数组的首地址来指向该“Buffer”指针。功能块的详细说明，参见本手册7.6.1操作符说明中的取地址章节。

⑥ DriveCom_ATV312:自定义功能块。该功能块按照ATV312变频器在使用modbus通讯控制时的状态机的时序，控制ATV312变频器启动停止和运行。

附2：自由口通讯

内容简介

M218 PLC提供了多种通讯方式可与施耐德产品或第三方设备进行配合,除Modbus协议外，也可以支持ASCII 协议，这样在理论上使得M218 PLC能和其他任何支持串行通讯的设备建立通讯。

本文以施耐德M218 PLC与某单片机为例, 简要介绍PLC与单片机之间以ASCII方式通讯的过程, 包括硬件接线、参数设置、硬软件组态等。

硬件连接

M218 SL2的螺钉端子与双绞线的一端连接，双绞线的另一端接到单片机，系统的硬件构架和连接如下：

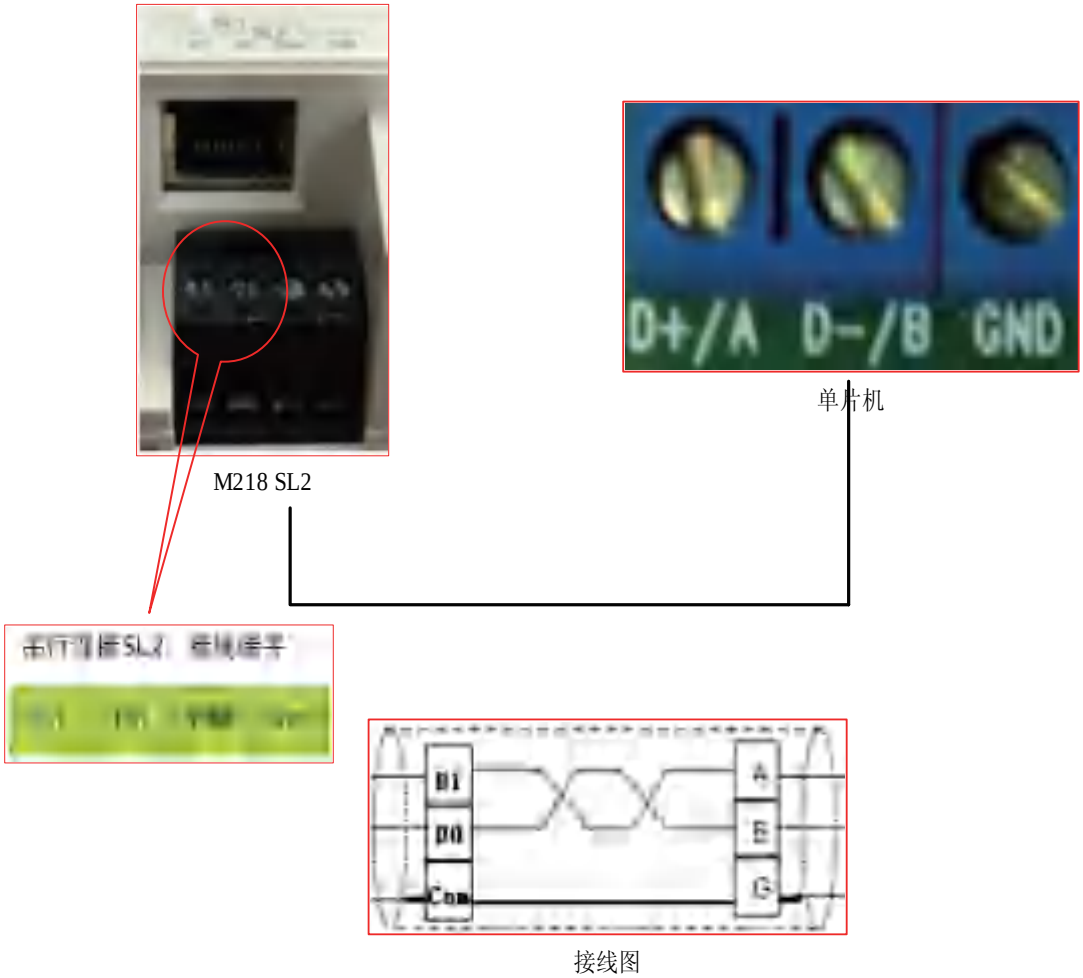


图3-1

备注：M218控制器内集成了2个RS485串口，1个端口（SL1）位RJ45口，另一个端口（SL2）为端子排接口，SL1可用来与HMI通讯，SL2可用来与变频器等通讯。

单片机ASCII 协议说明

⑴上位机取数据用的命令

当上位机发出如下命令时，则表示上位机要取得机组的运行数据：

字节数	1	2	3	4	7	9	11	12	16	17
名称	STX	上位机局号	机组局号	命令	首地址	字长	ETX	和数校验	CR	LF
数据	02H	' 2 '	‘a’	‘R’	‘aa’	‘aa’	03H	‘aaad	0DH	0AH

上述''表示为文字列（ASC II 码），a表示为ASC II 码的文字列（1个文字）。

上位机局号：2固定不变

机组局号：1~ 6（本文为1，表示机组1）

注：和数校验的计算为各CHECKSUM的计算方法，其位数为4位，计算时不包括STX在内。

其中校验和的计算=上位机局号+机组局号+命令+首地址+字长+03H

⑵单片机的正确反映

字节数	1	2	3	4	7	9	11	13	. . .
名称	STX	反应 1	反应 2	命令	首地址	字长	读出数据	读出数据	. . .
数据	02H	' 1 '	‘1’	‘W’	‘aa’	‘aa’	‘aa’	‘aa’	‘aa’

字节数	NX	NX+1	NX+5	NX+6
名称	ETX	和数校验	CR	LF
数据	03H	‘aaad	0DH	0AH

⑶错误反应：则无回应

⑷例：读机组1，F200开始的64个字节

字节数	1	2	3	4	5,6,7,8	9,10	11	12,13,14,15	16	17
名称	STX	上位机局号	机组局号	命令	首地址	字长	ETX	和数校验	CR	LF
数据	02H	32H	31H	52H	46H,32H,30H,30H	34H,30H	03H	30H,31H,46H,34H	0DH	0AH

PLC编程

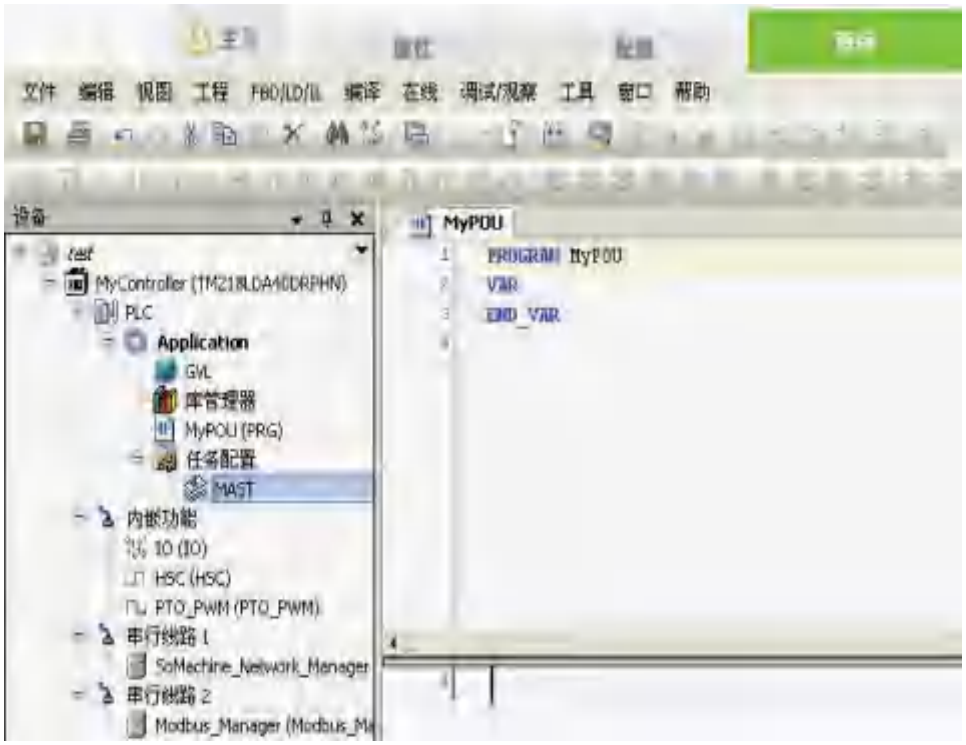
本例中的PLC组态和编程全部采用SoMachine软件平台完成。SoMachine是施耐德电气支持OEM PLC（M258、M238、M218）、HMI、Motion、VSD的通用编程、调试和运行的软件平台。

PLC编程主要包括硬件组态和软件编程两个部分

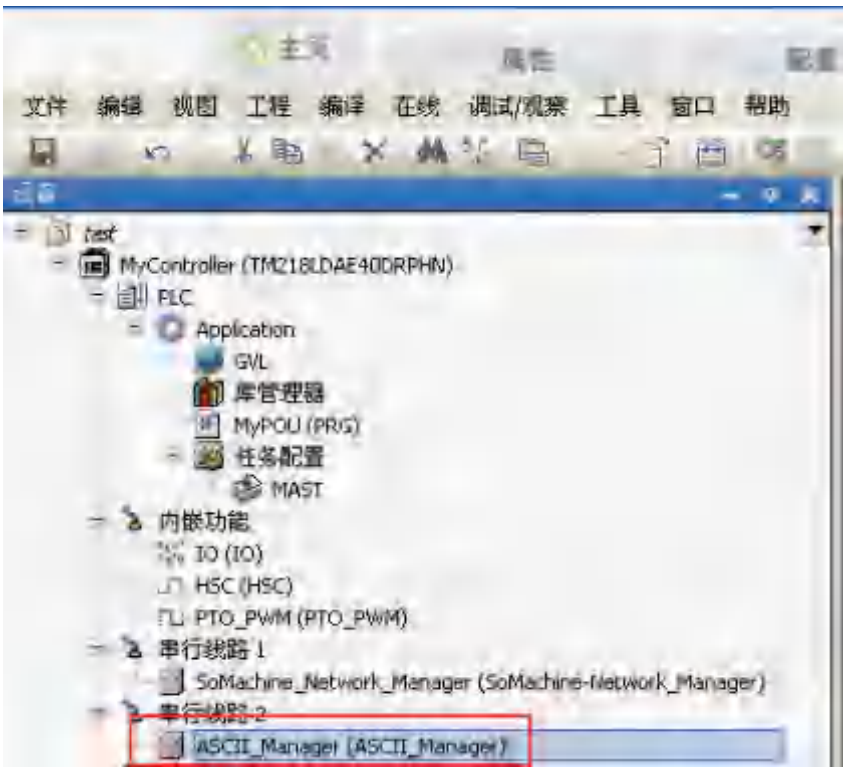
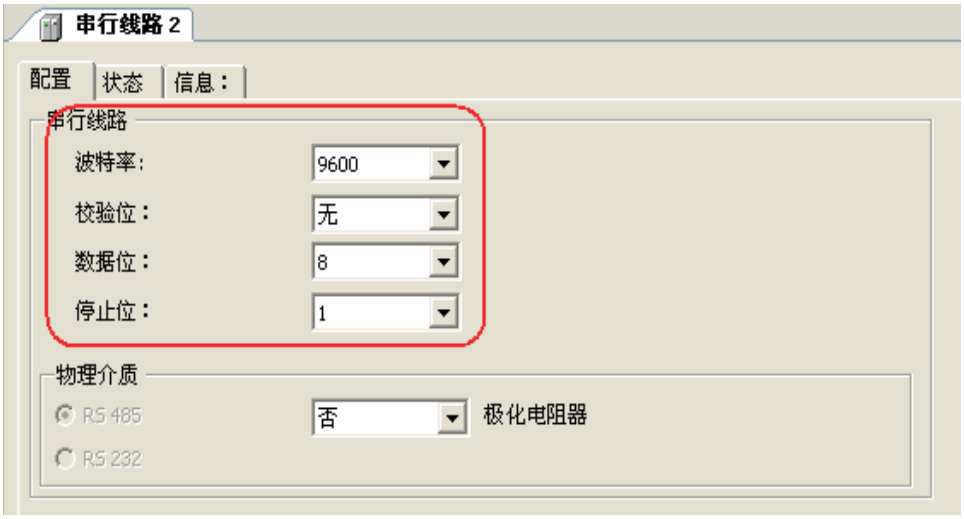
⑴、组态CPU

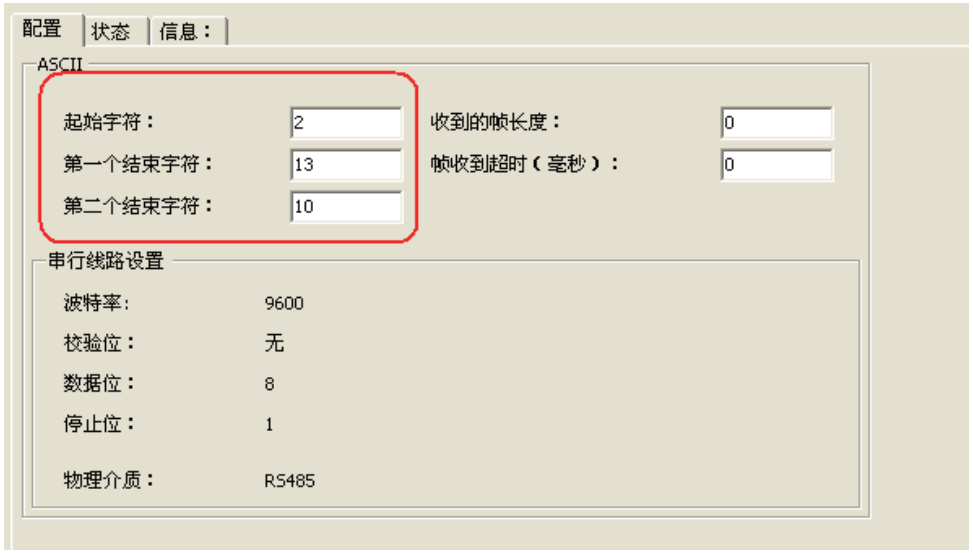
步骤	动作&示例
1	打开 SoMachine→创建新机器→使用空项目启动→将项目另存为(例：test)→配置→选择合适的 M218 型号

图3-2

2	进入“程序”界面，进行编程及“PLC 内嵌功能”等的组态  图3-3

⑵、组态PLC串口

步骤	动作&示例
1	选中“串行线路 2（SL2）”。“串行线路 2”的缺省通讯协议是“Modbus_Manager”，需删除，然后添加为“ASCII_Manager”  图3-4
2	双击“串行线路 2”，物理参数保证与单片机中的通信参数设置保持一致，如下图所示：  图3-5

3	双击“ ASCII_Manager”， 配置数据帧格式参数，如下图所示：  图3-6 **备注**：收发数据帧时，有三种方式判断结束：1、起始字符和结束符；2、收到的帧数据长度；3、帧收到超时（本文使用方式 1），说明如下表所示：

参数	**描述**
起始字符	如果为 0，则帧中不使用起始字符。否则，将在接收模式下使用相应的 ASCII 字符以检测帧的开头。在发送模式下，此字符将添加到用户帧的开头。
第一个结束字符	如果为 0，则帧中不使用第一个结束字符。否则，将在接收模式下使用相应的 ASCII 字符以检测帧的结尾。在发送模式下，此字符将添加到用户帧的结尾。
第二个结束字符	如果为 0，则帧中不使用第二个结束字符。否则，将在接收模式下使用相应的 ASCII 字符以检测帧的结尾。在发送模式下，此字符将添加到用户帧的结尾。

编程

⑴ ASCII通讯需要使用SEND_RECV_MSG功能块，例程如下所示：

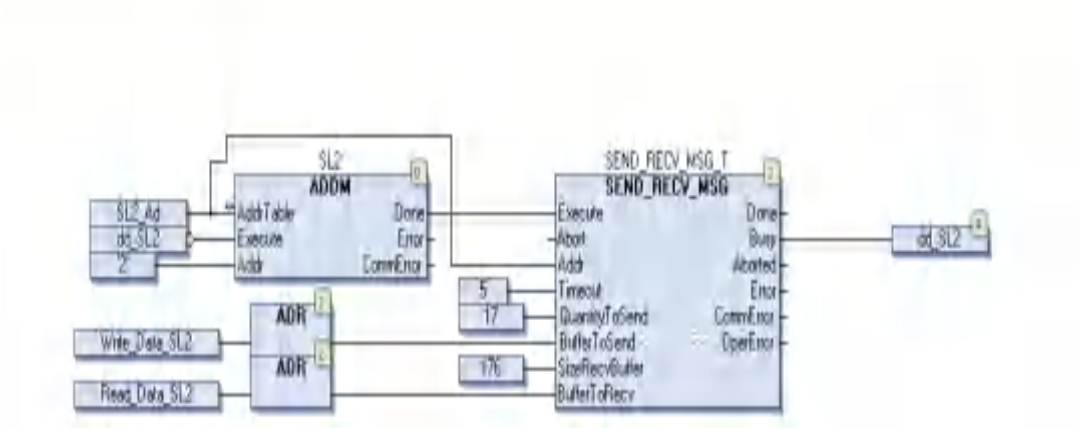


图3-7

⑵例程中使用到的功能块简介

①ADDM：管理通讯地址表。本例中，Addr 参数中写入'2'， 表示本PLC串行通讯端口2，功能块的详细说明，参见本手册8.4.1 通讯库中的ADDM章节。

② SEND_RECV_MSG：发送和/或接收用户定义的消息。将功能块的发送缓冲区（本例为数组Write_Data_SL2）中的多个数据发送到目标设备，并接收多个数据到功能块的接收缓冲区（本例为数组Read_Data_SL2）。功能块的详细说明，参见本手册8.4.6 通讯库中的SEND_RECV_MSG章节。

③ ADR：取地址指令。由于“SEND_RECV_MSG R”功能块的管脚“BufferToSend”和“BufferToRecv”是指针变量，所以用ADR 功能块来取数组的首地址来指向相应指针。功能块的详细说明，参见本手册7.6.1操作符说明中的取地址章节。

④ dd_SL2 变量必须在第一个循环后设置为 TRUE（由用户在线设置或由应用程序设置），才能启动连续发送/接收消息。

⑤ SizeRecvBuffer 数量设置为143 可参考本文前面通讯协议的说明。

附3：以太网通讯示例

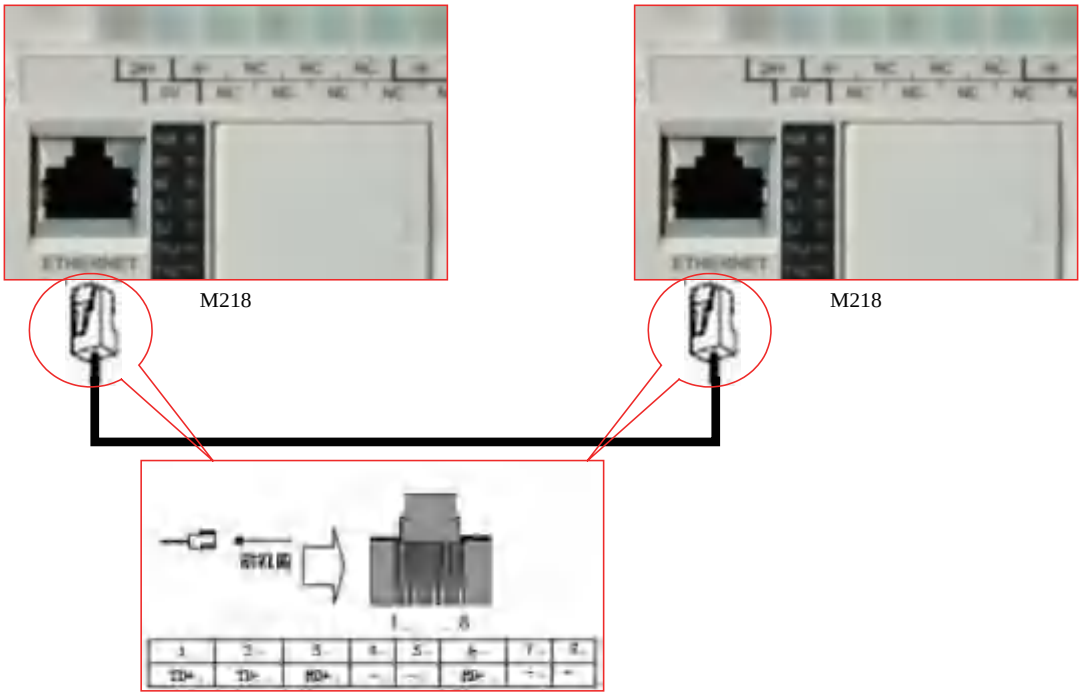
内容简介:

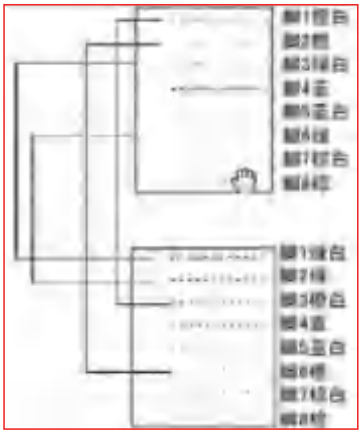
M218 PLC中TM218LDAE24DRHN/TM218LDAE40DRPHN两款PLC，本体集成了以太网通讯口，能自适应数据传输率（10/100M）和工作模式（半/全双工），并支持端口自动翻转。支持ModbusTCP/IP通讯协议（可做ModbusTCP服务器/客户端），该以太网口可用于与其它支持ModbusTCP/IP协议的设备之间的数据通讯。

本文以两台M218 PLC为例, 简要介绍M218PLC与M218PLC之间Modbus以太网通信的过程, 包括硬件接线、参数设置、硬软件组态等，实现一台PLC对另一台PLC的数据读写。

硬件连接:

两台M218 PLC 间的连接网线可采用直通线也可采用交叉线，系统的硬件构架和连接如下（本文以交叉网线为例）：





交叉网线接法

图4-1

PLC编程

本例中的PLC组态和编程全部采用SoMachine软件平台完成。SoMachine是施耐德电气支持OEM PLC（M258、M238、M218）、HMI、Motion、VSD的通用编程、调试和运行的软件平台。

PLC编程主要包括硬件组态和软件编程两个部分

⑴、组态CPU（客户端）

步骤	动作&示例
1	打开 SoMachine→创建新机器→使用空项目启动→将项目另存为(例：test)→配置→选择合适的 M218 型号  图4-2

2 进入“程序”界面，进行编程及“PLC 内嵌功能”等的组态

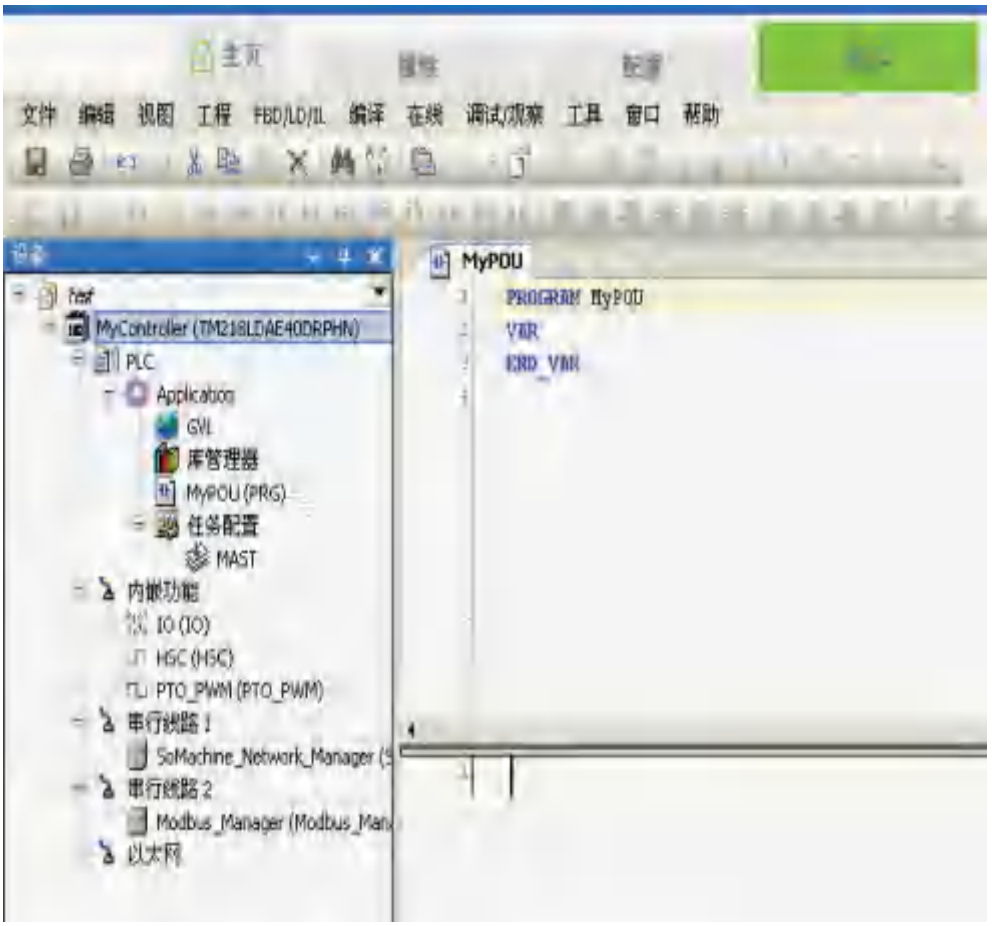
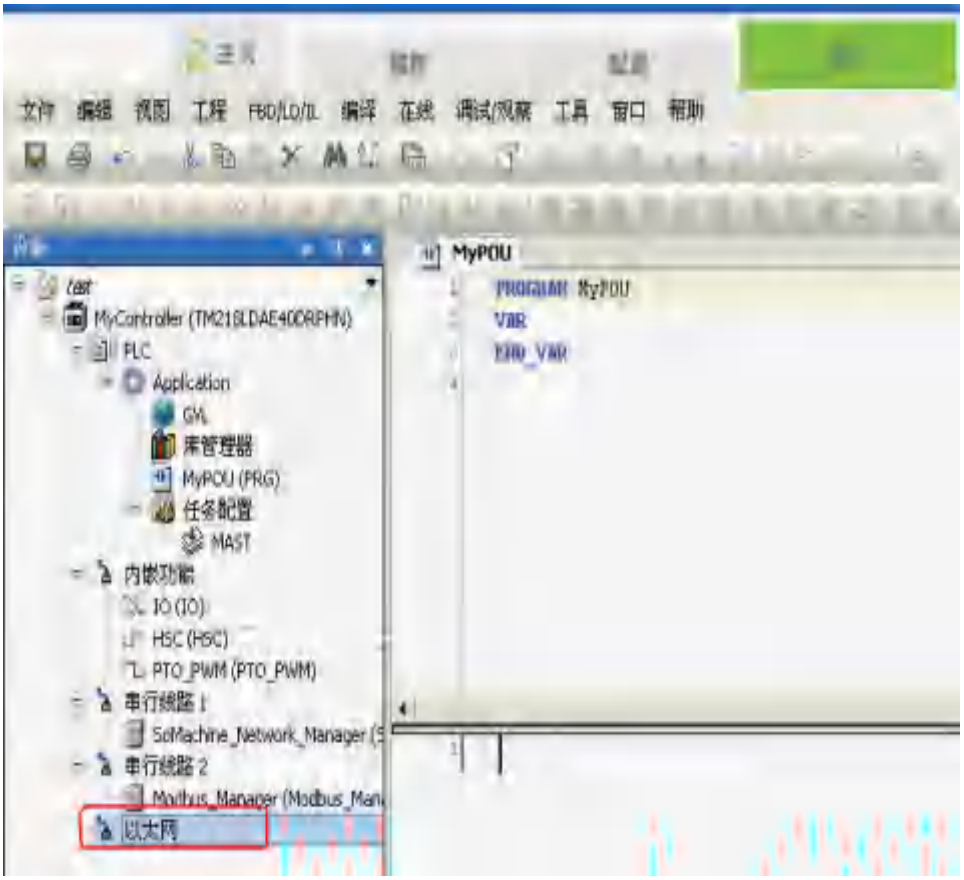


图4-3

⑵、组态以太网端口（客户端）

步骤	动作&示例
1	单击，选中“以太网”。  图4-4

2 双击“以太网”，设置 IP 地址，如下图所示：

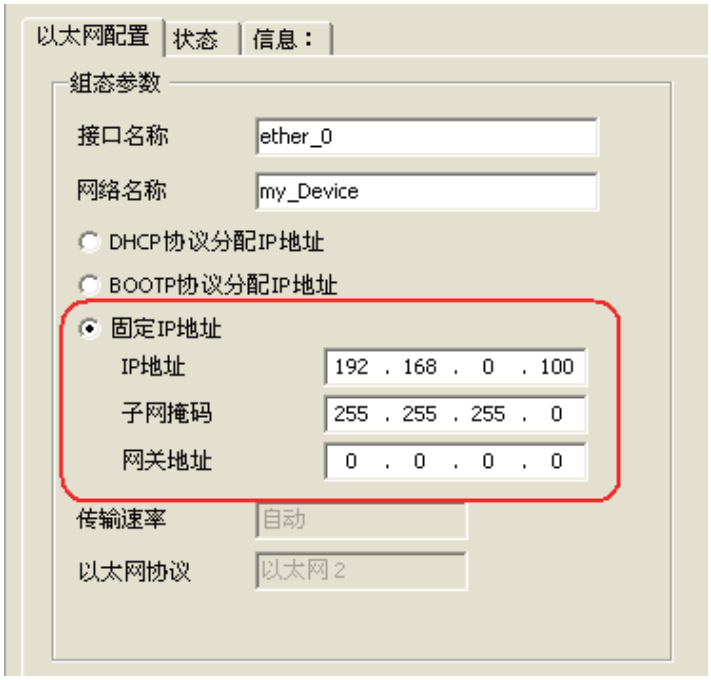
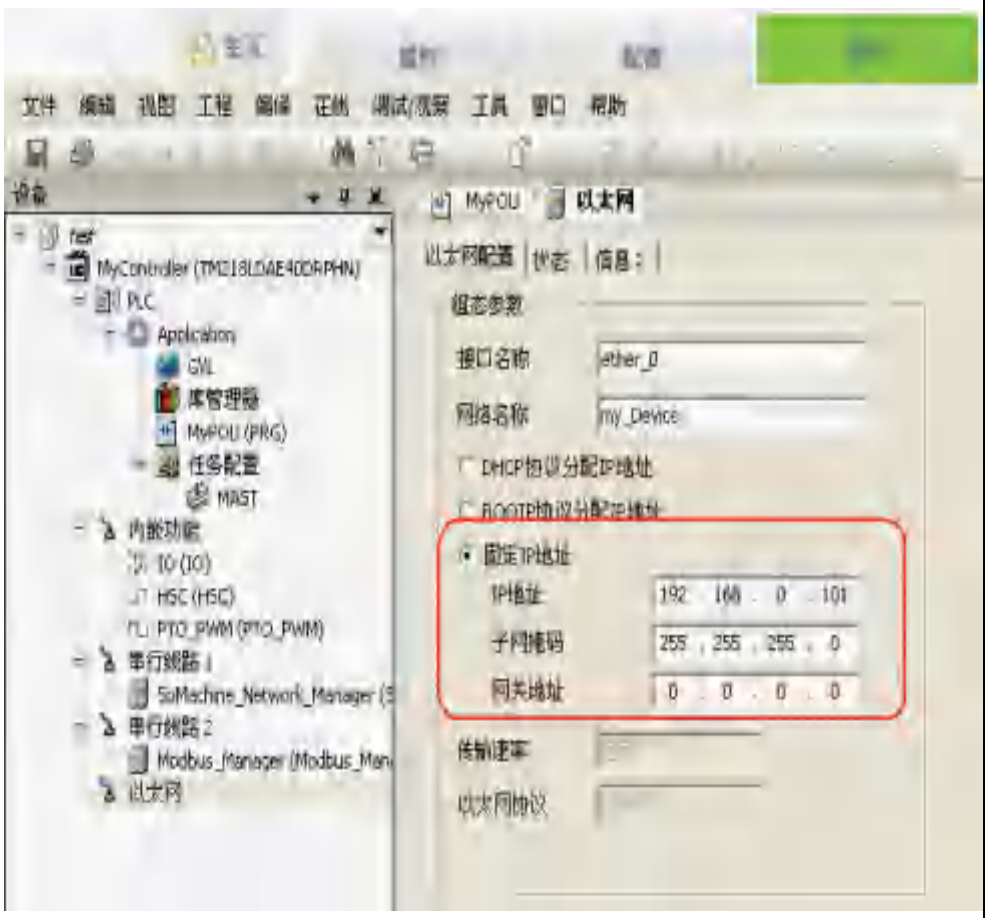


图4-5

⑶、服务器PLC的组态CPU及以太网端口过程相同，只需将IP地址设为同一网段不同地址即可

步骤	动作&示例
	双击“以太网”，设置 IP 地址，如下图所示：  图4-6

编程

⑴ModbusTCP 服务器M218 PLC只需要配置一下IP地址即可,在客户端M218 PLC侧编程。例程如下所示（因M218支持Modbus功能代码23,可使用全双工功能块WRITE_READ_VAR）：

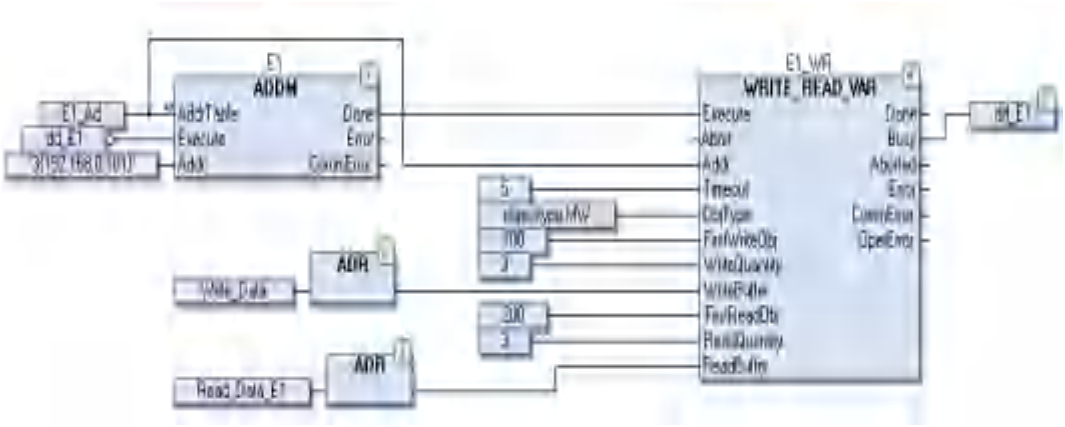


图4-7

⑵例程中使用到的功能块等简介

①ADDM：管理通讯地址表。本例中，Addr 参数中写入'3{192.168.0.101}'，其中3 表示本PLC以太网端口，192.168.0.101 表示ModbusTCP服务器的IP地址。Modbus TCP 标准从站使用 Modbus 地址 255（UnitId 默认值），但是，Modbus TCP 设备的值可能不同，在这种情况下，请添加 UnitId 值（如下图连接第三方触摸屏的例子，它的UnitId为1）。功能块的详细说明，参见本手册8.4.1 通讯库中的ADDM章节。

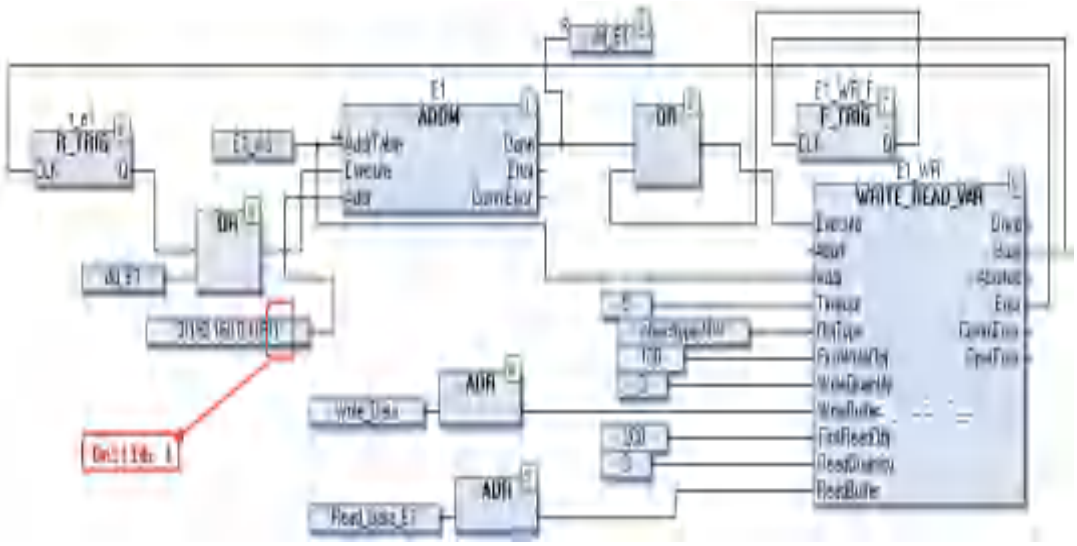


图4-8

② WRITE_READ_VAR：写入读取功能块。将功能块的写入数据发送缓冲区（本例为数组Write_Data）中的多个数据发送到目标设备的寄存器区，并从目标设备的寄存器区读取多个数据到功能块的读取数据接收缓冲区（本例为数组Read_Data_E1）。读取和写入操作在同一个事务中完成，功能块的详细说明，参见本手册8.4.4 通讯库中的WRITE_READ_VAR章节。

③ ADR：取地址指令。由于“WRITE_READ_VAR”功能块的管脚“WriteBuffer”和“ReadBuffer”是指针变量，所以用ADR 功能块来取数组的首地址来指向相应指针。功能块的详细说明，参见本手册7.6.1操作符说明中的取地址章节。

④ dd_E1 变量必须在第一个循环后设置为 TRUE（由用户在线设置或由应用程序设置），才能启动连续读/写

⑤M218服务器/客户端的最大访问连接：4 Server/4 Client。以太网通讯中，M218可作为客户端去问一个或多个服务器，同时也可作为服务器被一个或多个客户端访问。但访问的服务器和被访问的客户端总和不能超过4个。

如果客户端尝试打开的连接超过了最大连接数4，则会关闭最先打开的空闲的连接，若所有的连接都忙，则新连接会被拒绝。

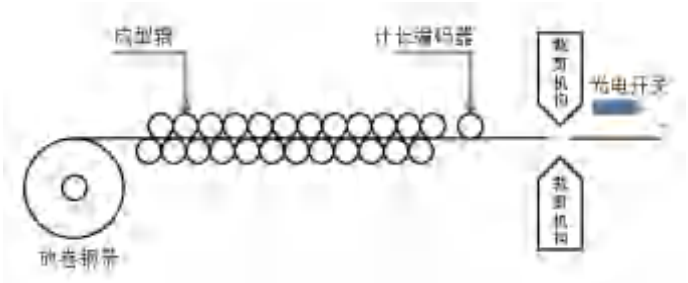
附4：M218高速计数器示例

内容简介

本文介绍如何通过M218的高速计数器功能实现冷弯成型机的定长裁剪控制。

1.冷弯成型机控制系统描述：

冷弯成型机最基本的控制要求是将滚压成型完成的材料送到指定位置，然后进行压膜或者裁切动作。材料的输送电机由变频器控制。在靠近压膜或者裁剪机构的滚轮上安装有用于计长的旋转编码器，PLC实时检测该编码器的脉冲信号并换算成长度数值。当机器启动时，PLC将实际检测的长度数值与设定数值进行比较，控制变频器进行多段速定位。即当长度到达阀值0时，变频器切换到低速；当长度到达阀值1时变频器输出0速。



(图1) 冷弯机控制系统说明

裁剪机构上检测开关的上升沿可以用于捕捉，当裁剪机构动作时的编码器值，通过该值自动修正阀值1的设定值；同时，该检测开关的下降沿用于将编码器的当前值复位成预设值，重新计数。

变频器多段速设置，当阀值0和阀值1的反馈输出都为FALSE的时候，变频器以高速运行，频率50Hz；当阀值0输出TRUE时，频率切换到第二段速低速5Hz；当阀值1输出TRUE时，频率切换到第三段速0Hz。

2.编码器选型：

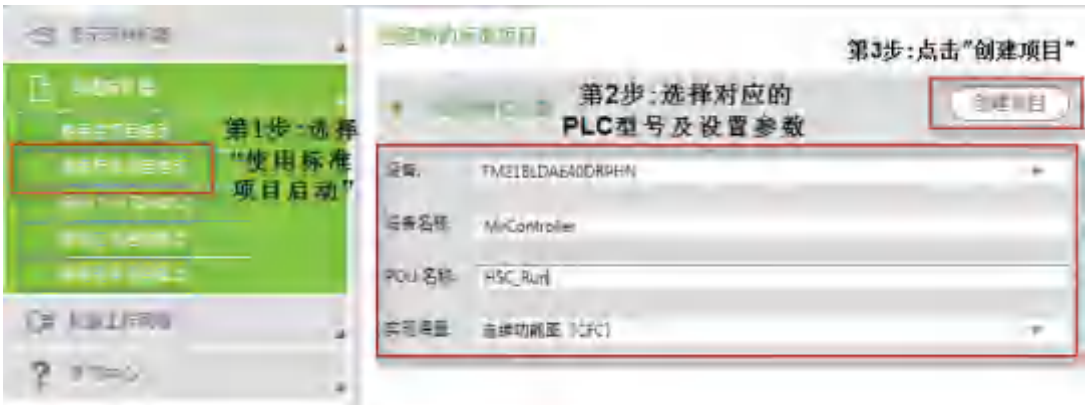
安装编码器的滚轮周长是250mm，冷弯机的设计速度是15米/分钟，即滚轮的最大速度是1转/秒。设计采用的编码器脉冲输入是2000脉冲/转，即脉冲输入信号最大为2KHz，小于M218高速计数器的最大采样输入100KHz。

根据M218的输入端的电源及接线信号选择24V的PNP或者NPN的编码器。

3.软件配置及编程：

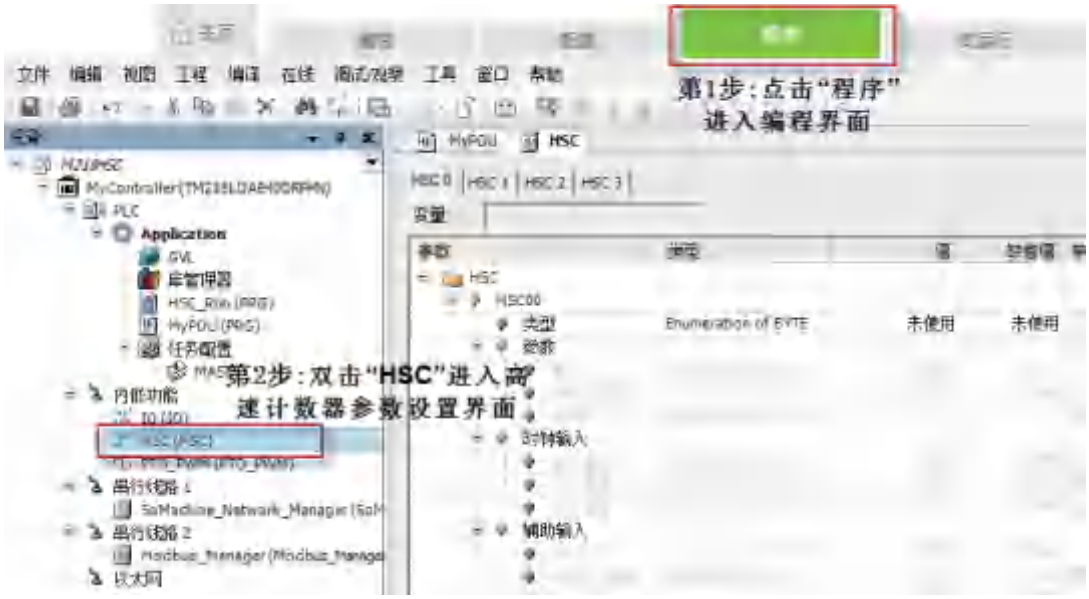
3.1 创建新项目：

打开Somachine软件，选择“创建新机器”–>“使用标准项目启动”–>选择对应的PLC及设置参数–>点击“创建项目”–>将项目保存为“M218HSC.project”。



3.2配置Somachine软件的HSC参数：

3.2.1 项目创建完毕后，进入“程序”页面：



3.2.2 配置高速计数器参数：

类型：高速计数器的类型选项有2种，一种是“简化”型，一种是“主要”型。该示例中使用的是AB相的计数器，因此选择“主要”类型；

参数-模式：默认是“一次性”模式，这里选择“自由大型计数”模式；

参数-预设/模数：设为0，该参数可以通过程序在线修改；

时钟输入-输入模式：选择“正常积分×4”；

时钟输入-A过滤值/B过滤值：该参数保留默认值；

辅助输入-CAP：选择“已启用”；

辅助输入-CAP过滤器/CAP沿：该参数保留默认值；

阀值-阀值0/阀值1：选择“已启用”；

阀值-阀值0值/阀值0事件：该参数保留默认值，阀值可以通过程序修改；

阀值-阀值1值/阀值1事件：该参数保留默认值，阀值可以通过程序修改；

反射输出-反射输出0（反射输出0的条件是 值<Threshold 0）：选择“否”，保留默认值；

反射输出-反射输出0（反射输出0的条件是 Threshold 0<值<Threshold 1）：选择“是”；

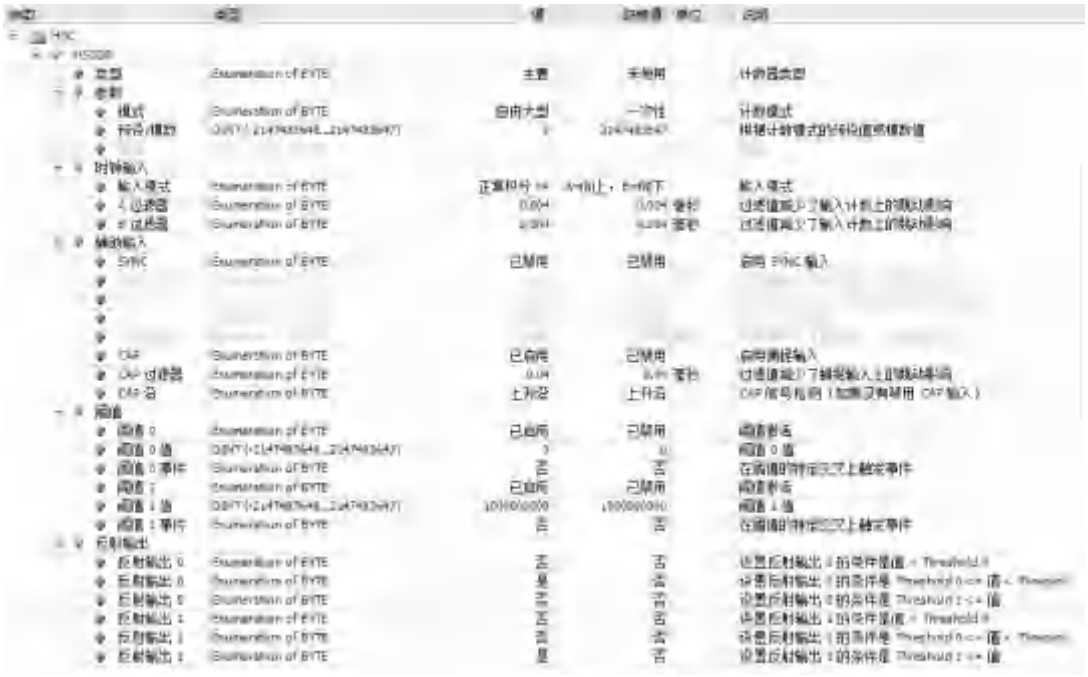
反射输出-反射输出0（反射输出0的条件是 值>=Threshold 1）：选择“否”，保留默认值；；

反射输出-反射输出1（反射输出0的条件是 值<Threshold 0）：选择“否”，保留默认值；

反射输出-反射输出1（反射输出0的条件是 Threshold 0<值<Threshold 1）：选择“否”，保留默认值；

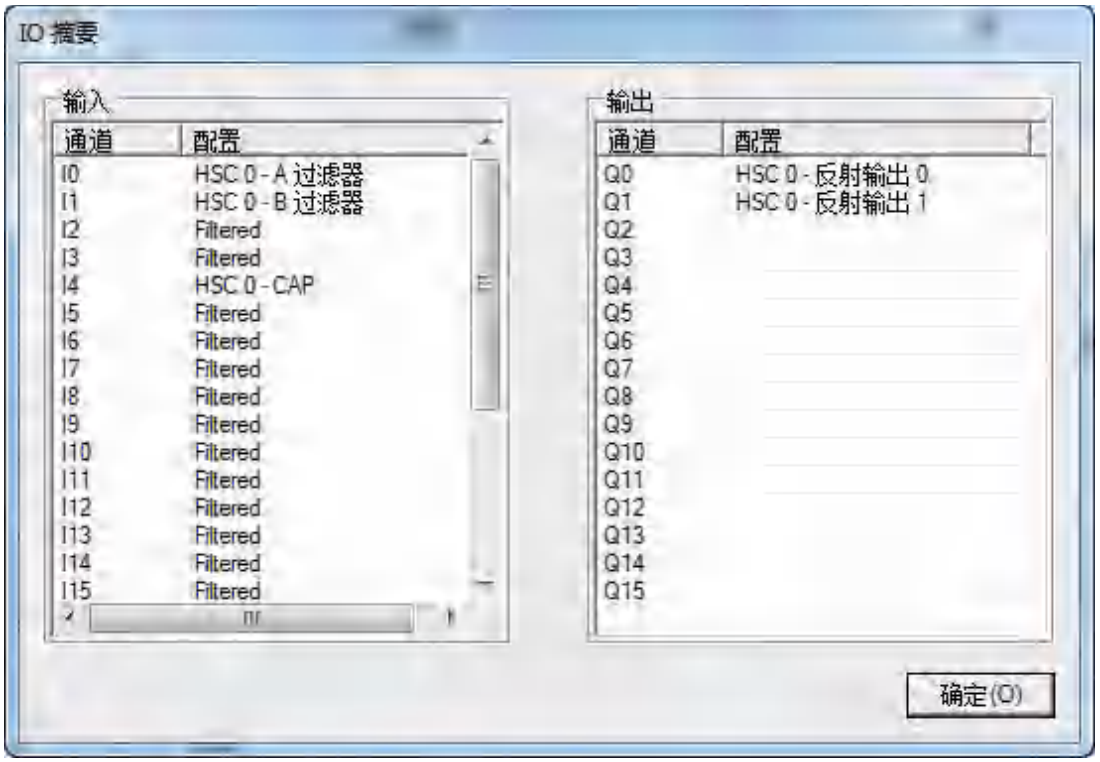
反射输出-反射输出1（反射输出0的条件是 值>=Threshold 1）：选择“否”，选择“是”；

如下图：



3.2.3 查看IO分配：

点击该窗口右下角的“IO和摘要”，Somachine将自动显示该配置对应的物理IO配置，如下图：



从上图可以看出系统的IO分配如下：

I0/I1：编码器的A相和B相信号输入；

I4（%IX0.4）：高速计数器0的Cap信号输入，当信号有效时，捕捉编码器的当前值，光电开关信号接到该端子上；

Q0(%QX0.0)：配置的反射输出0信号输出，当信号输出时，控制变频器切换到低速；

Q1(%QX0.1)：配置的反射输出1信号输出，当信号输出时，控制变频器切换到0速或者停止；

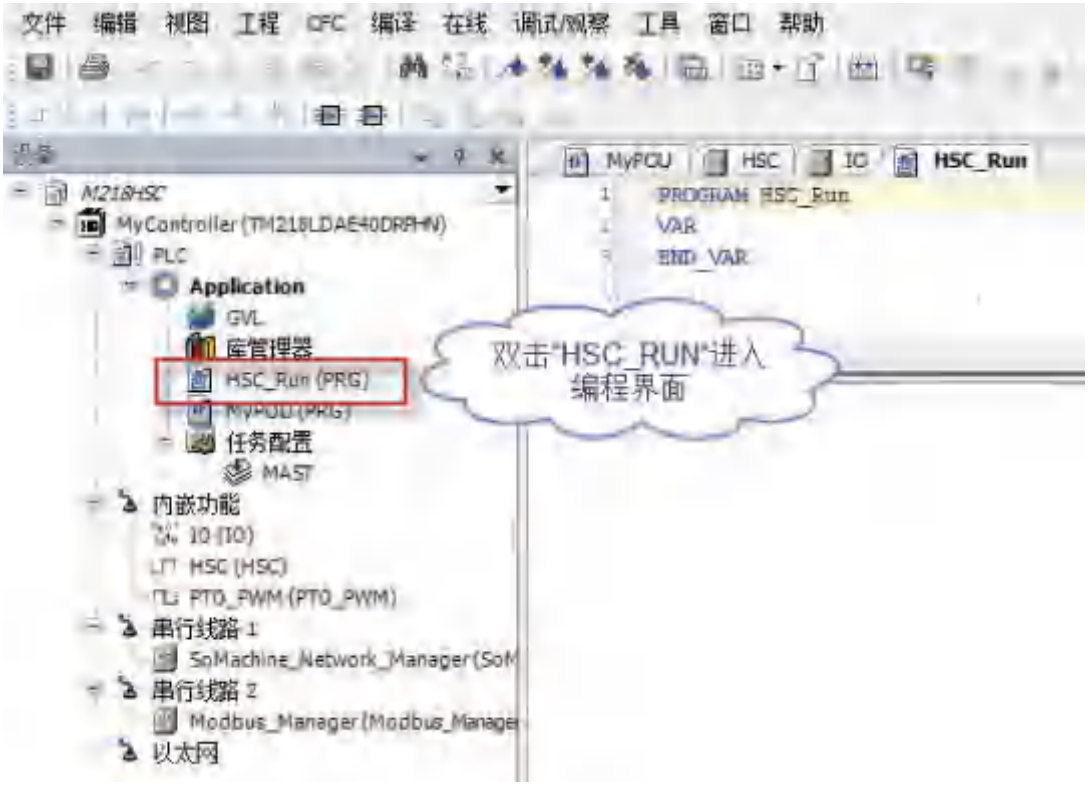
同时根据程序需要配置：

Q4(%QX0.4)：变频器启停信号；

Q5(%QX0.5)：切刀上下启停信号；

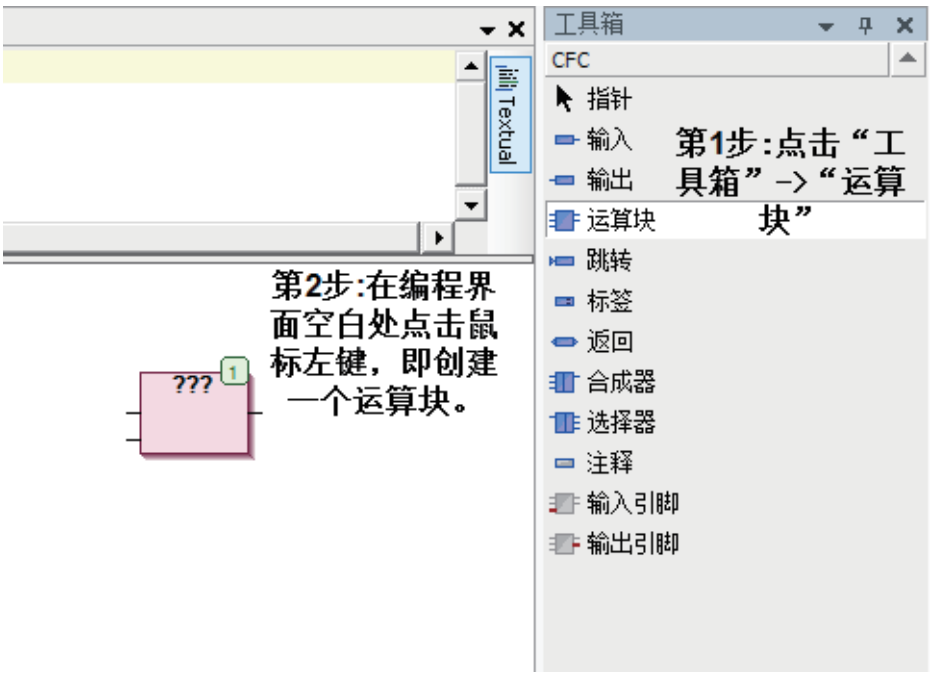
3.3程序编程

3.3.1 双击“HSC_RUN”，进入程序编程界面；

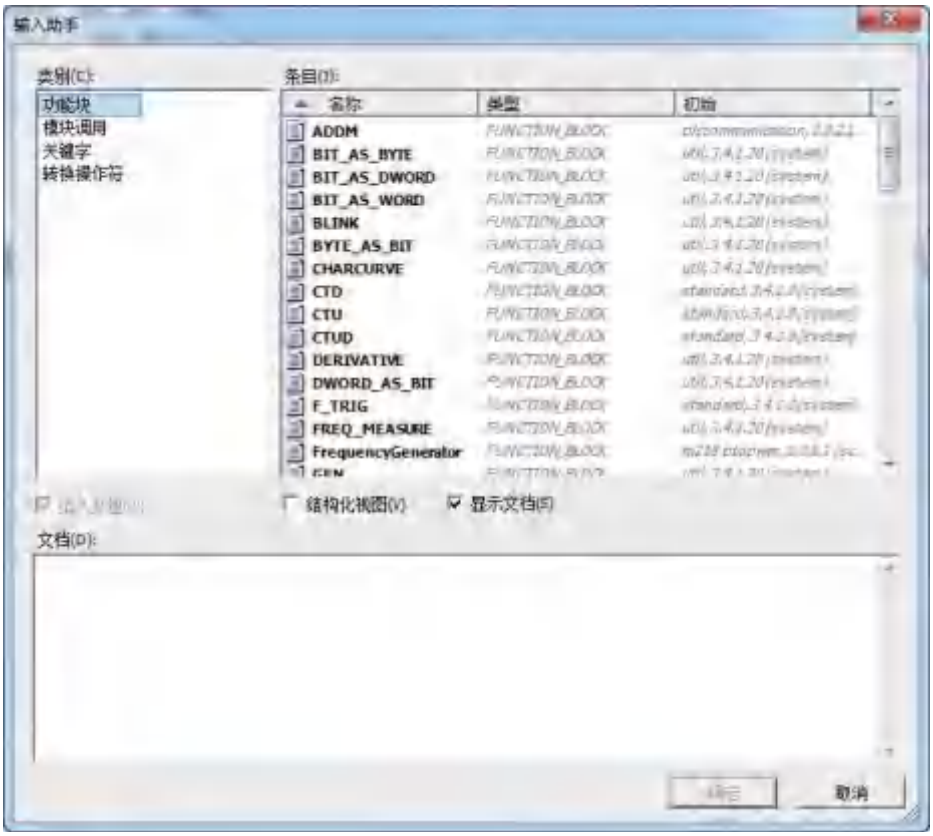


3.3.2 选择功能块并创建功能块示例：

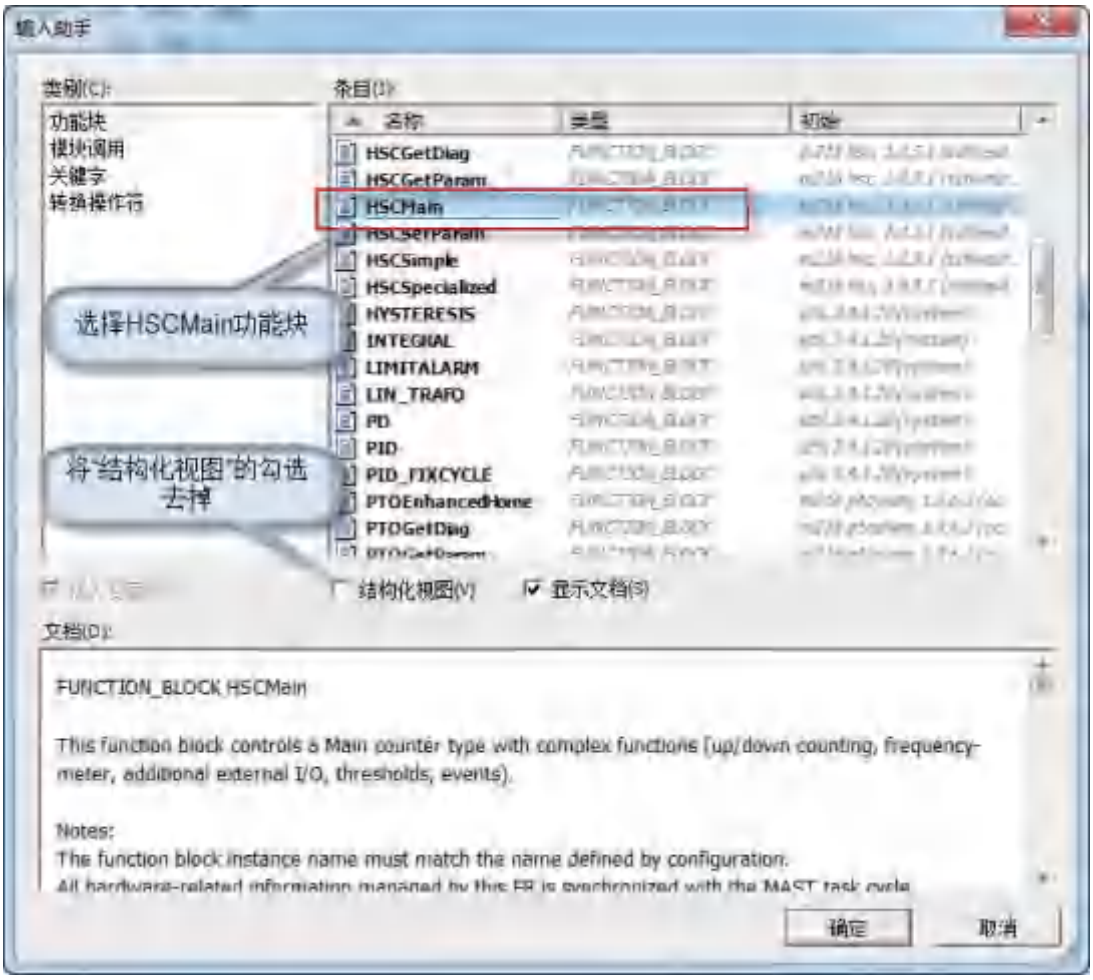
3.3.2.1 点击“工具箱”–>点击“运算块”并在编程空白地方按下鼠标左键



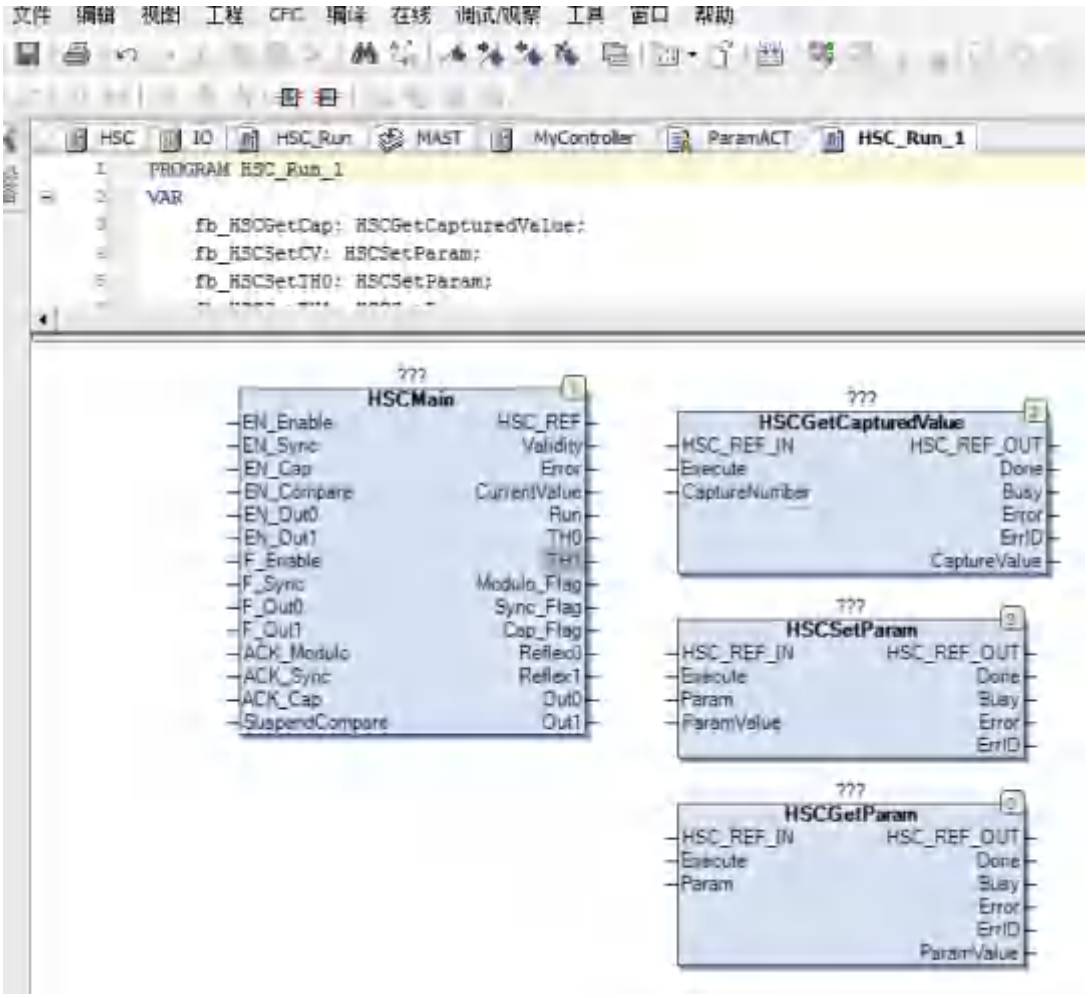
3.3.2.2 点击运算块的问号，并按键盘的“F2”功能键，打开输入助手菜单：



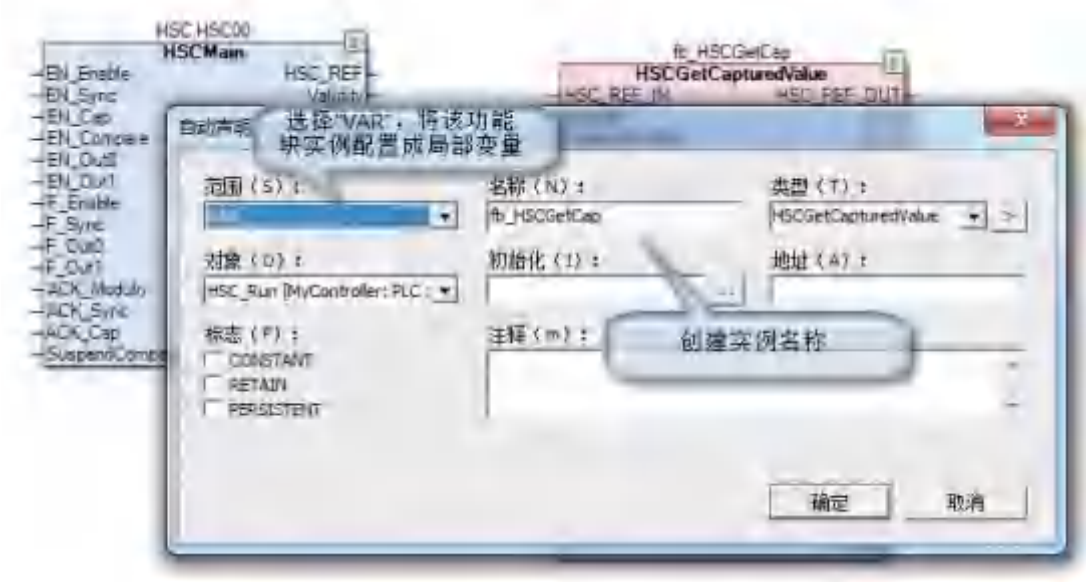
在“功能块”的右边“条目”中选择需要的功能块，这里我们选择“HSCMain”功能块。



3.3.2.3 重复上述操作，再选出HSCSetParam/HSCGetParam/ HSCGetCapturedValue,如下图：



3.3.2.4 点击功能块的问号，输入该功能块的实例名称并回车；Somachine将提示是否创建该实例，点击“确定”确认创建；注意：HSCMain功能块的示例必须为HSC.HSC00；



3.3.2.4 点击功能块的问号，输入该功能块的实例名称并回车；Somachine将提示是否创建该实例，点击“确定”确认创建；注意：HSCMaifb_HSCGetCap: HSCGetCapturedValue功能块的实例;

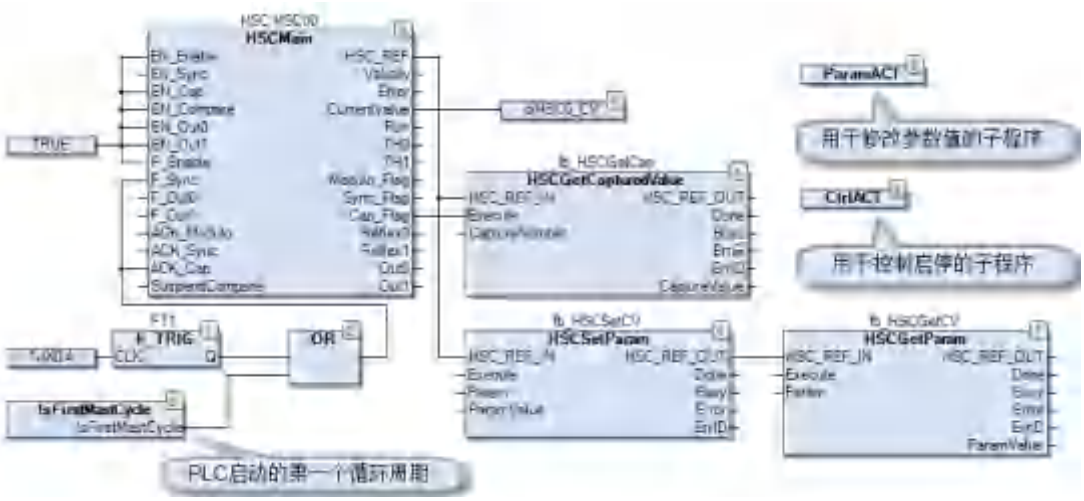
fb_HSCSetCV: HSCSetParam功能块的实例;

fb_HSCSetTH0: HSCSetParam功能块的实例;

fb_HSCSetTH1: HSCSetParam功能块的实例;

3.3.3 定义各个功能块的需要用到的管脚名称，并连接相应的管脚；

n功能块的示例必须为HSC.HSC00；



diHSC0_CV:DINT型变量，用于记录当前的计数器数值；

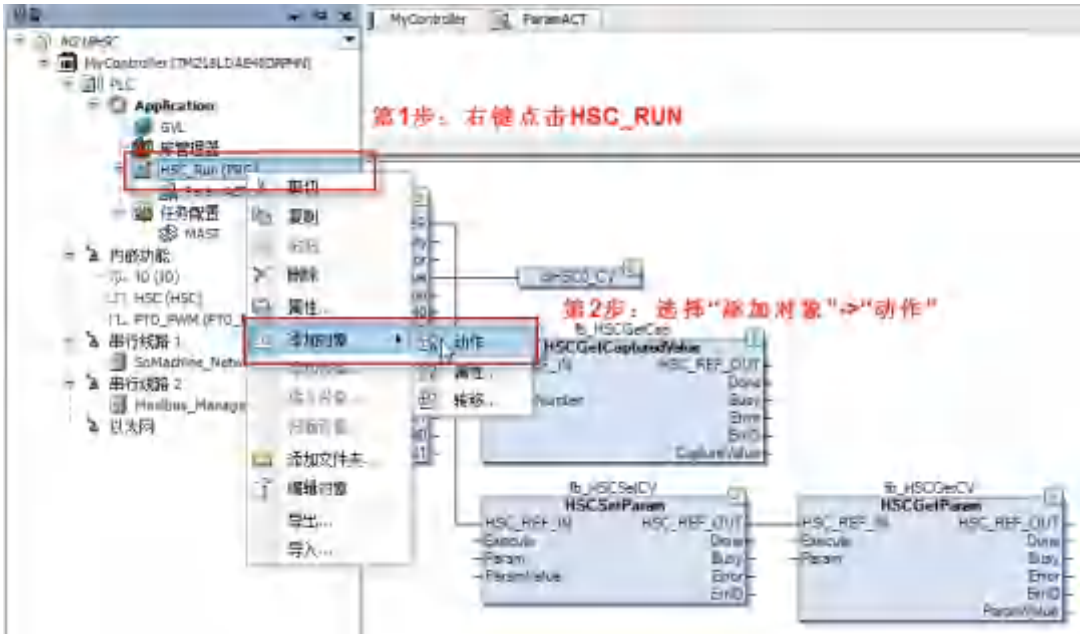
ParamACT：动作名称，相当于子程序调用；

预设值和阀值可以根据需要在线修改，当设置的数值变化时，执行阀值修改；

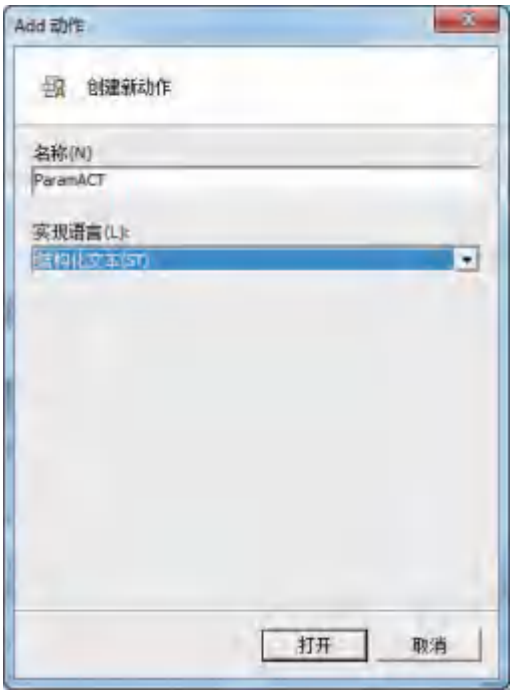
注意：在修改阀值时必须保证阀值0<阀值1，否则阀值1将保持原值；

3.3.4 添加HSC参数设置动作（ParamACT）：

3.3.4.1 创建动作ParamACT：



在打开的"创建新动作"对话框中填入名称"ParamACT"，并选择"ST"语言，如下图：



3.3.4.2 添加动作程序：

采用ST语言编写读写高速计数器预设值、阀值0和阀值1的程序。读写功能块的“Param”管脚对应的是枚举型变量，变量对应值如下表：

枚举变量名	值	说明
HSC_PRESET	0	获取或者设置HSC的预设值
HSC_THRESHOLD0	5	获取或者设置HSC的阀值0值
HSC_THRESHOLD1	6	获取或者设置HSC的阀值1值

首先添加变量至变量定义单元：

```
PROGRAM HSC_Run
VAR
        fb_HSCGetCap: HSCGetCapturedValue;
        fb_HSCSetCV: HSCSetParam;
        fb_HSCGetCV: HSCGetParam
        diHSC0_CV: DINT;
        diSet_PV: DINT; (*HSC预设值设定值*)
        diSet_TH0: DINT;        (*HSC阀值0设定值*)
        diSet_TH1: DINT;        (*HSC阀值1设定值*)
        diHSC0_PV: DINT;        (*HSC预设值显示值*)
        diHSC0_TH0: DINT; (*HSC阀值0显示值*)
        diHSC0_TH1: DINT; (*HSC阀值1显示值*)
        diDiffCheck:DINT; (*捕捉编码器值与设定值的差值*)
diSetLength:DINT; (*设定的长度值*)
bAutoStart: BOOL; (*整机启停信号*)
bSystemError: BOOL; (*系统故障报警*)
iStepRead: INT; (*读步计数器*)
        iStepSet: INT; (*写步计数器 *)
END_VAR
```

将下列程序添加到动作“ParamACT”中：

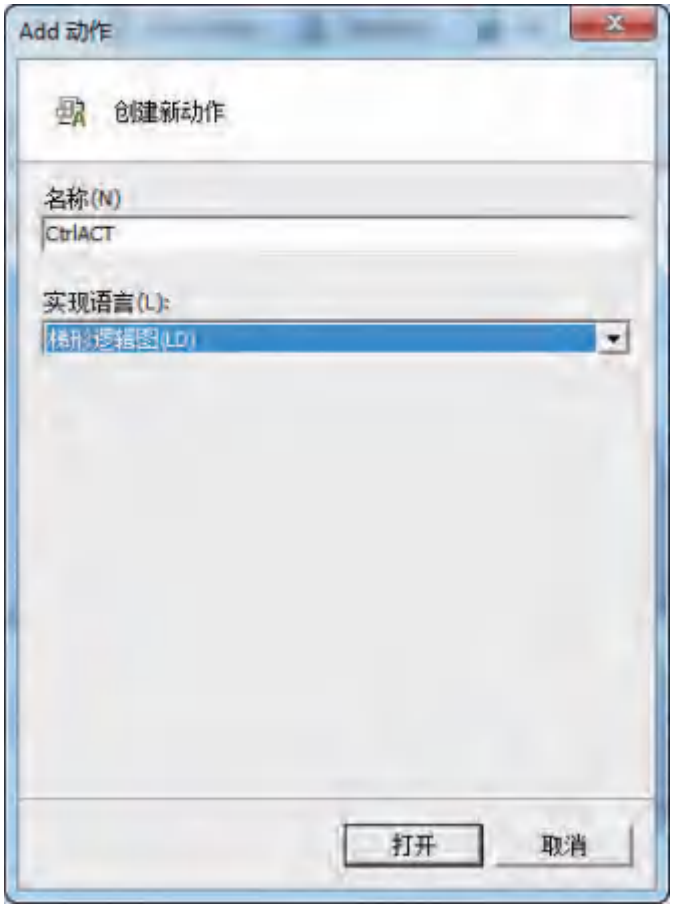
```
(*****程序开始*****)
(*******调用HSCGetParam功能块,读取预设值/阀值0/阀值1********)
CASE iStepRead OF
0:
fb_HSCGetCV.Param:=0;(*0=HSC_PRESET 获取HSC的预设值*)
fb_HSCGetCV.Execute:=TRUE;
IF fb_HSCGetCV.Done OR fb_HSCGetCV.Error THEN
        fb_HSCGetCV.Execute:=FALSE;
```

```
diHSC0_PV:=fb_HSCGetCV.ParamValue;
        iStepRead:=iStepRead+10;
END_IF
10:
fb_HSCGetCV.Param:=5;(*5=HSC_THRESHOLD0  获取HSC的阈值0值*)
fb_HSCGetCV.Execute:=TRUE;
IF fb_HSCGetCV.Done OR fb_HSCGetCV.Error THEN
        fb_HSCGetCV.Execute:=FALSE;
        diHSC0_TH0:=fb_HSCGetCV.ParamValue;
        iStepRead:=iStepRead+10;
END_IF
20:
fb_HSCGetCV.Param:=6;(*6=HSC_THRESHOLD1  获取HSC的阈值1值*)
fb_HSCGetCV.Execute:=TRUE;
IF fb_HSCGetCV.Done OR fb_HSCGetCV.Error THEN
        fb_HSCGetCV.Execute:=FALSE;
        diHSC0_TH1:=fb_HSCGetCV.ParamValue;
        iStepRead:=0;
END_IF
END_CASE
(**********调用HSCSetParam功能块,设置预设值/阀值0/阀值1*********)
CASE iStepSet OF
0:
IF diSet_PV<>diHSC0_PV THEN
        fb_HSCSetCV.Param:=0;(*0=HSC_PRESET 设置HSC的预设值*)
        fb_HSCSetCV.ParamValue:=diSet_PV;
        fb_HSCSetCV.Execute:=TRUE;
        IF fb_HSCSetCV.Done OR fb_HSCSetCV.Error THEN
                fb_HSCSetCV.Execute:=FALSE;
                iStepSet:=iStepSet+10;
        END_IF
ELSE
        iStepSet:=iStepSet+10;
END_IF
10:
IF diSet_TH0<>diHSC0_TH0 THEN
```

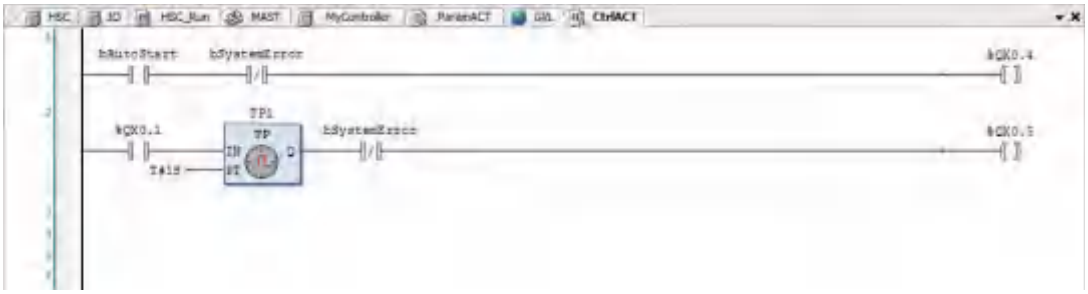
```
fb_HSCSetCV.Param:=5;(*5=HSC_THRESHOLD0  设置HSC的阈值0值*)
        fb_HSCSetCV.ParamValue:=diSet_TH0;
        fb_HSCSetCV.Execute:=TRUE;
        IF fb_HSCSetCV.Done OR fb_HSCSetCV.Error THEN
            fb_HSCSetCV.Execute:=FALSE;
            iStepSet:=iStepSet+10;
        END_IF
ELSE
        iStepSet:=iStepSet+10;
END_IF
20:
IF diSet_TH1<>diHSC0_TH1 -diDiffCheck THEN
        fb_HSCSetCV.Param:=6;(*6=HSC_THRESHOLD1  设置HSC的阈值1值*)
        fb_HSCSetCV.ParamValue:=diSet_TH1- diDiffCheck;
        fb_HSCSetCV.Execute:=TRUE;
        IF fb_HSCSetCV.Done OR fb_HSCSetCV.Error THEN
            fb_HSCSetCV.Execute:=FALSE;
            iStepSet:=0;
        END_IF
ELSE
        iStepSet:=0;
END_IF
END_CASE
(**********数值计算*********)
IF fb_HSCGetCap.Done THEN
        diTempDiff:=fb_HSCGetCap.CaptureValue-diSetLength;
        IF diTempDiff<>0 THEN
            diDiffCheck:=diTempDiff;
        END_IF
END_IF
(**********程序结束*********)
```

3.3.5 添加IO控制动作（CtrlACT）：

.3.5.1 与创建ParamACT动作一样，创建CtrlACT动作。但CtrlACT采用LD（梯形图语言）编程，如下图：



3.3.5.2 添加动作程序：



a.bAutoStart启动后，变频器即启动，当bAutoStart复位后，变频器即停止；

b.当阀值1输出时(%QX0.1=TRUE)，变频器输出0Hz，控制切刀液压气缸（%QX0.5）启动；这里加了个TP延时，即切刀气缸通1秒后，自动断开；

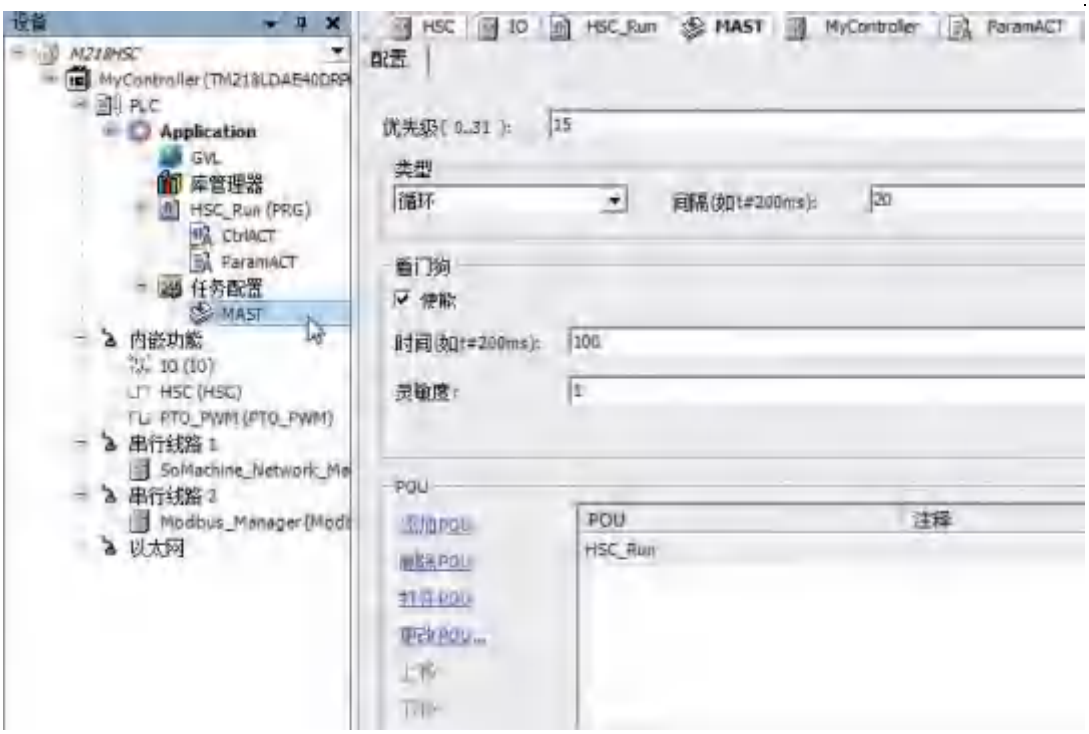
c.当切刀气缸复位，同时PLC检测到光电开关信号的下降沿，将复位计数器的值至预设值，则阀值1信号复位。变频器重新以高速启动。

3.3.6 将HSC_RUN程序添加到运行任务中，如下图：

a.双击“MAST”打开配置窗口；

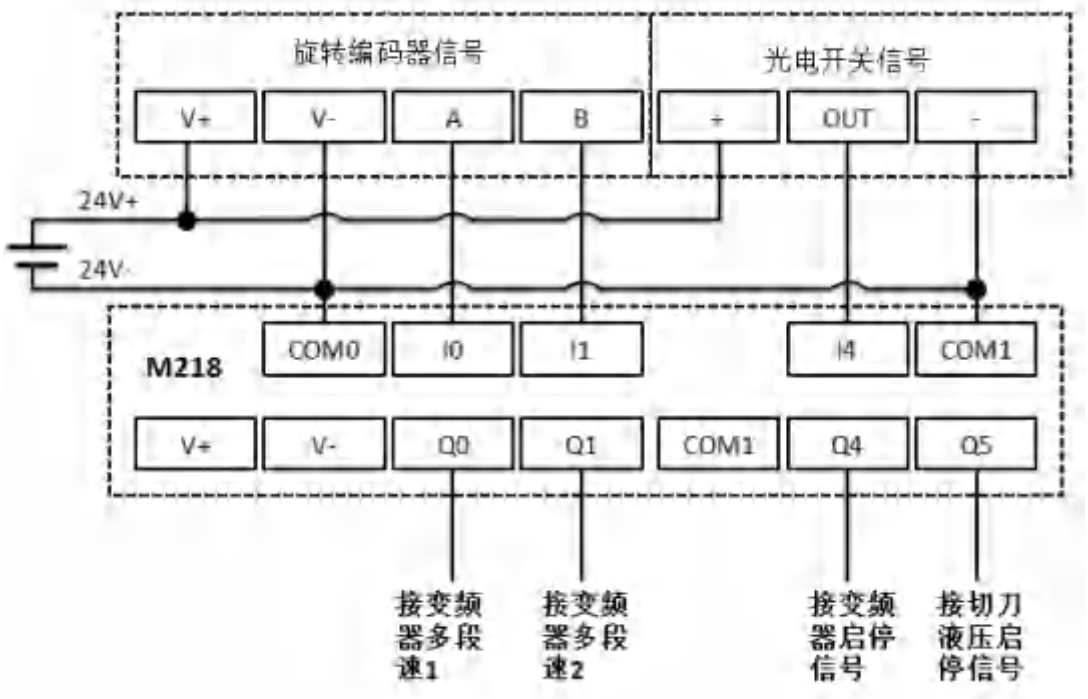
b.选择“添加POU”；

c.在弹出的窗口中，选择“HSC_RUN”，添加完成；

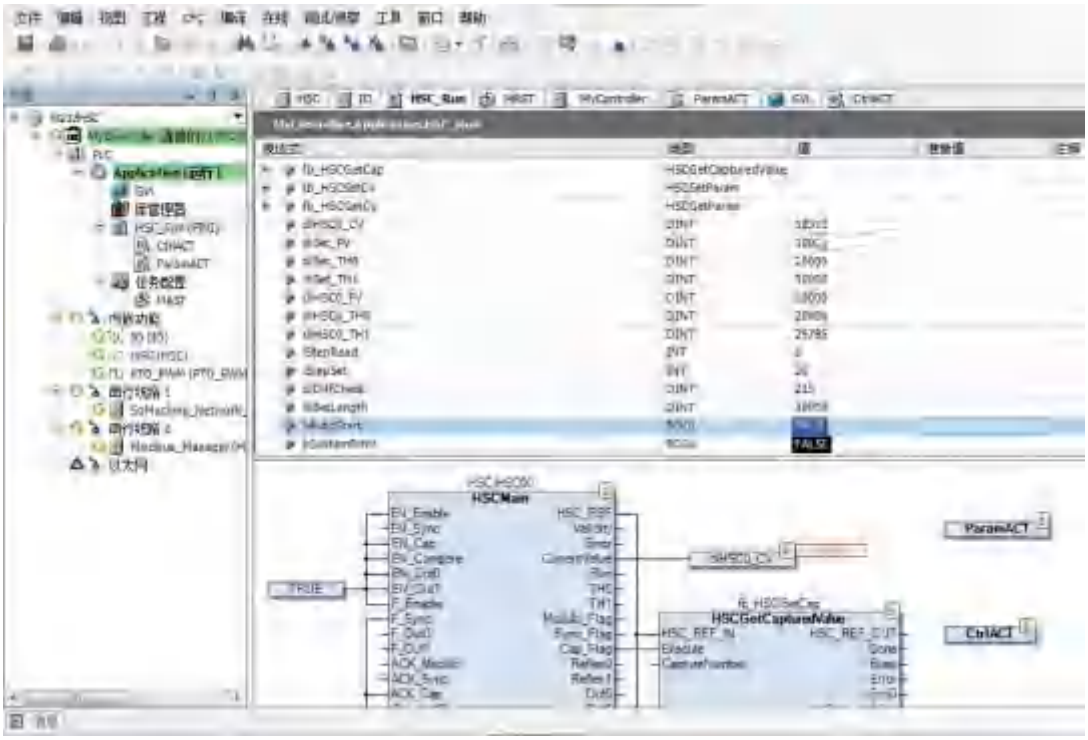


4.硬件接线：

PLC的硬件接线图如下图：

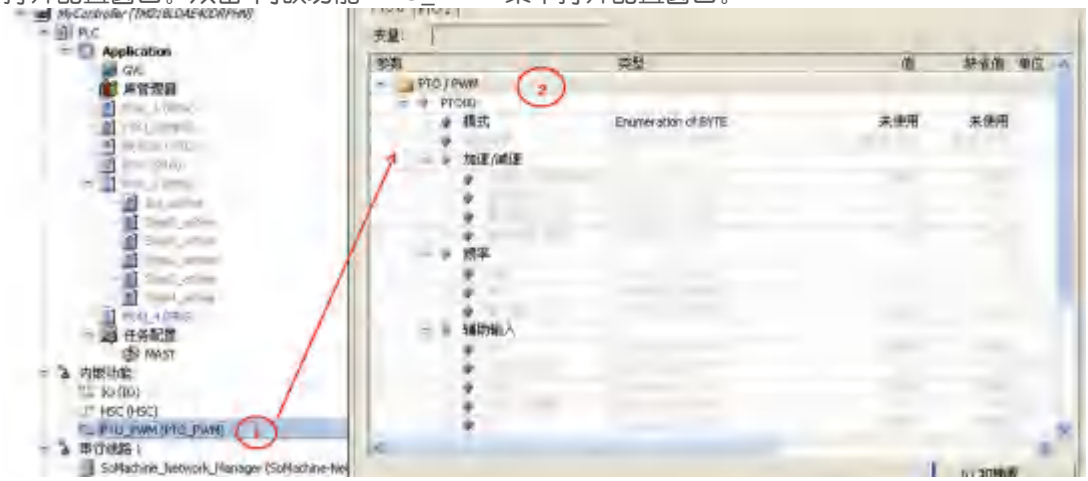


5.仿真调试：



PWM示例

步骤1. PWM配置
打开配置窗口。双击“内嵌功能-PTO_PWM”菜单打开配置窗口。

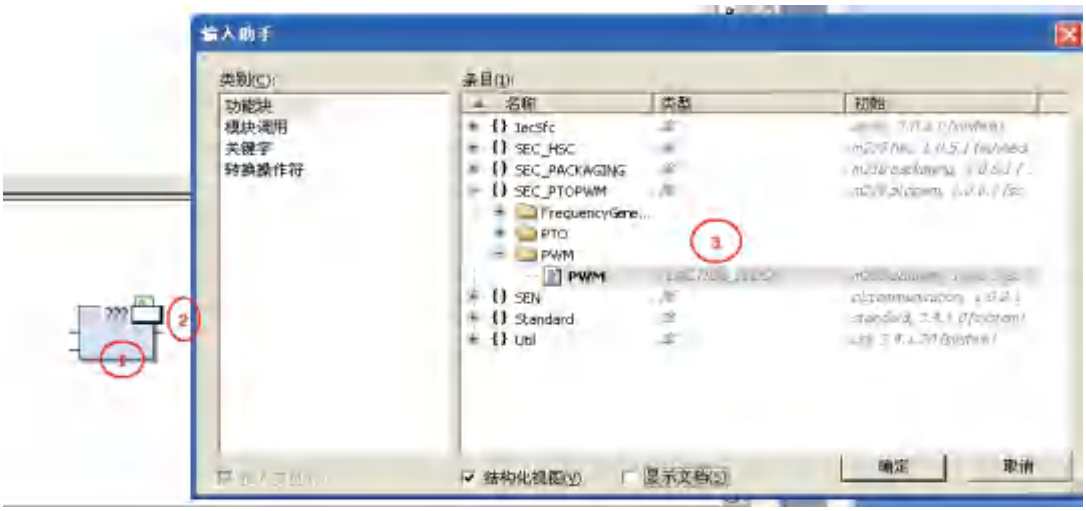


双击“内嵌功能-PTO_PWM”后会弹出左边的配置窗口。默认情况下PTO/PWM是“未使用”状态。
单击“未使用”有下拉菜单供选择。

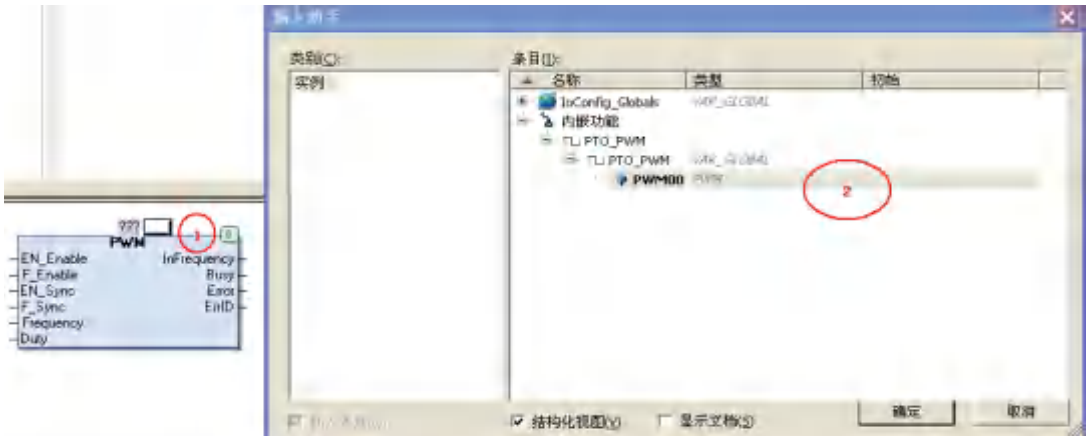
选择“PWM”功能。选中此功能后可用的参数如下：

参数		值	设备	说明
模式		PWM	-	选择的模式为 PWM。
辅助输入	EN	禁用* 启用	-	启用要用于启用功能的 IN_EN 物理输入。
	EN 过滤器	0.04* 0.4 1.2 4	毫秒	定义 IN_EN 过滤器值的值。
	SYNC	禁用* 启用	-	启用要用于同步的 IN_SYNC 输入。
	SYNC 过滤器	0.04* 0.4 1.2 4	毫秒	定义 IN_SYNC 过滤器值的值。
	SYNC 沿	上升沿* 下降沿	-	定义进行同步的 IN_SYNC 跳变沿。
说明		* 参数缺省值		

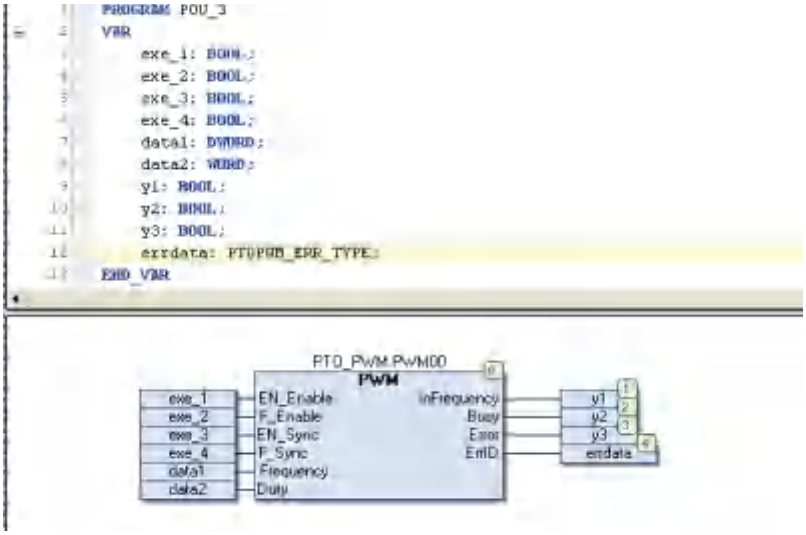
步骤2.程序编制
本例以CFC语言为例。
新建一个新的POU,语言选择CFC。



点取一个空的功能块，点击“？？？”旁边的按钮打开输入助手，在“SEC_PTOPWM”下选择“PWM”。点击“确认”。

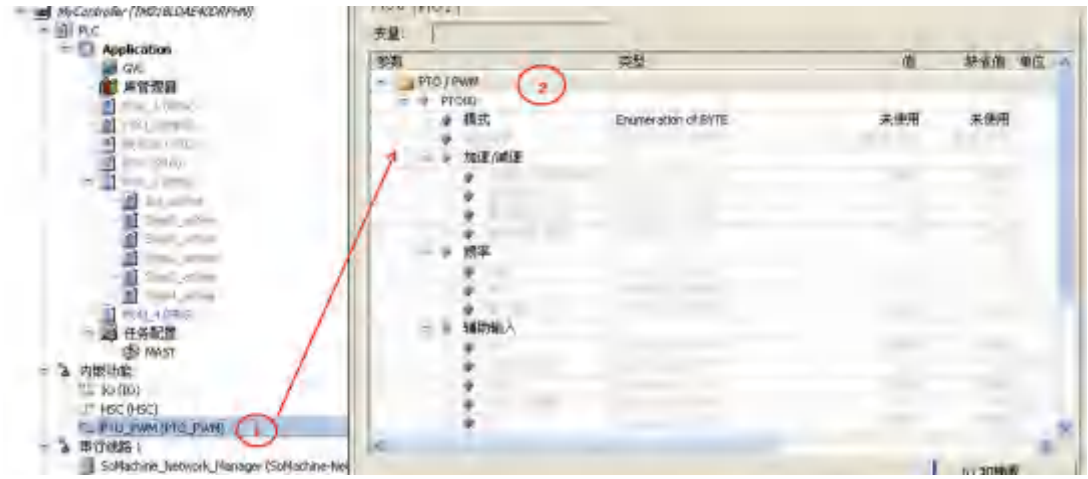


在PWM功能块上方“？？？”输入通道号“PWM00”。在对应管脚输入标签标量。具体管脚定义参考PWM功能块说明。编辑好的画面如下：



把编好的POU添加到“MAST”任务里。下载后就可以执行相应动作。

FG频率发生器示例
步骤1 FG配置
打开配置窗口。双击“内嵌功能-PTO_PWM”菜单打开配置窗口

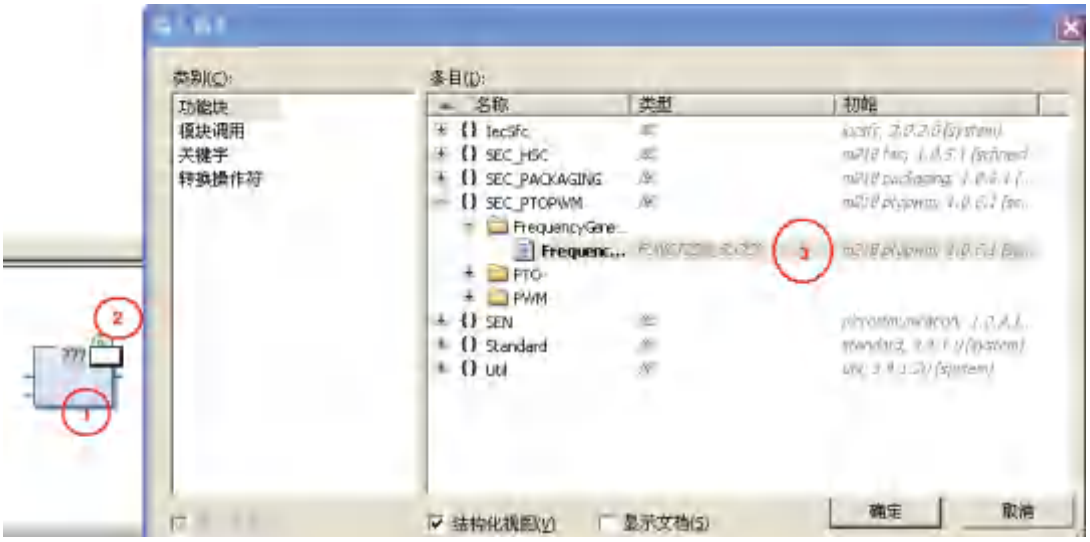


双击“内嵌功能-PTO_PWM”后会弹出左边的配置窗口。默认情况下PTO/PWM是“未使用”状态。单击“未使用”有下拉菜单供选择。

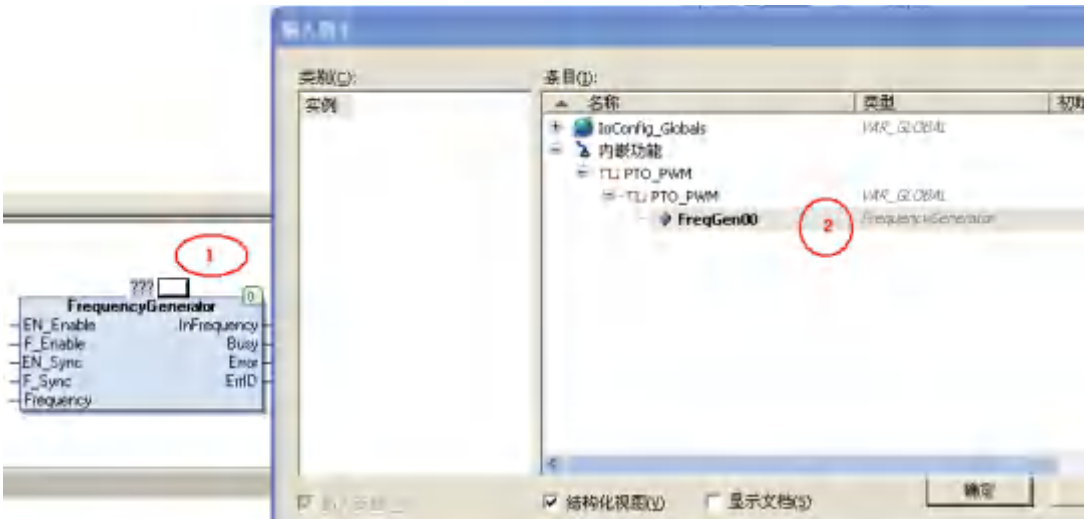
选择“频率发生器”。可使用的参数如下：

参数		值	设备	说明
模式		PWM	-	选择的模式为 PWM。
辅助输入	EN	禁用* 启用	-	启用要用于启用功能的 IN_EN 物理输入。
	EN 过滤器	0.04* 0.4 1.2 4	毫秒	定义 IN_EN 过滤器值的值。
	SYNC	禁用* 启用	-	启用要用于同步的 IN_SYNC 输入。
	SYNC 过滤器	0.04* 0.4 1.2 4	毫秒	定义 IN_SYNC 过滤器值的值。
	SYNC 沿	上升沿* 下降沿	-	定义进行同步的 IN_SYNC 跳变沿。
说明	* 参数缺省值			

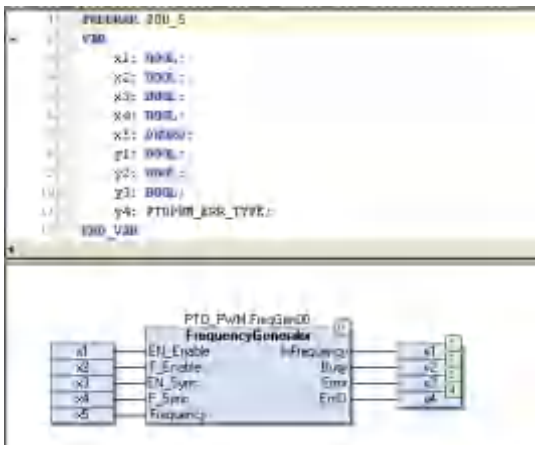
步骤2 程序编制
本例以CFC语言为例。



选择一个空功能块，点击“？？？”边上的按钮弹出输入助手，点击“SEC_PTOPWM”菜单，选择子菜单“FrequencyGenerator”。点击“确定”。 在FrequencyGenerator功能块上方“？？？”输入通道号“FreGen00”。如下图示：



在对应管脚输入标签标量。具体管脚定义参考PWM功能块说明。
给对应管脚添加标签变量。编辑好换面如下：



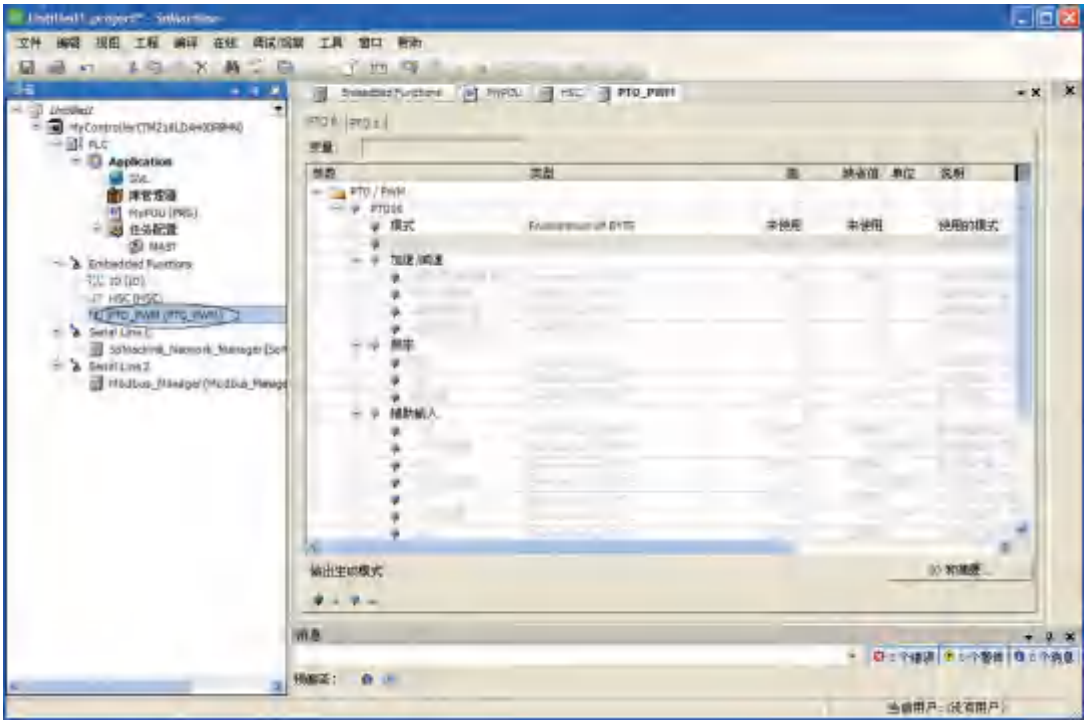
示例说明:
通过M218实现脉冲输出控制Lexium23伺服,
案例中所使用M218型号为TM218LDA40DRPHN, LXM23信号为LXM23AU04M3X

一.Lexium23与M218的PTO接线

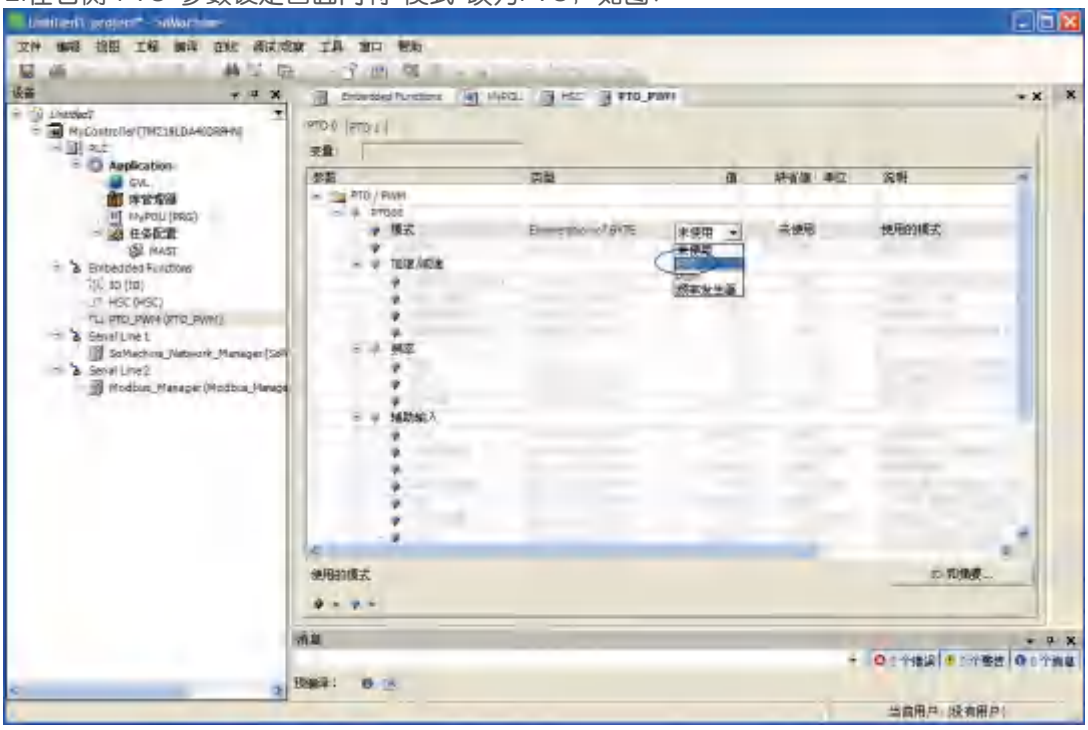
M218		Lexium23	
TR0	----------	Pin 43	/Pulse
TR1	----------	Pin 36	/Sign
V-	----------	Pin 45	Com-
Q5	----------	Pin 9	SON
Q6	----------	Pin 33	ARST

二.Lexium23的参数设置
P1-00 : 01
P1-01 : 00

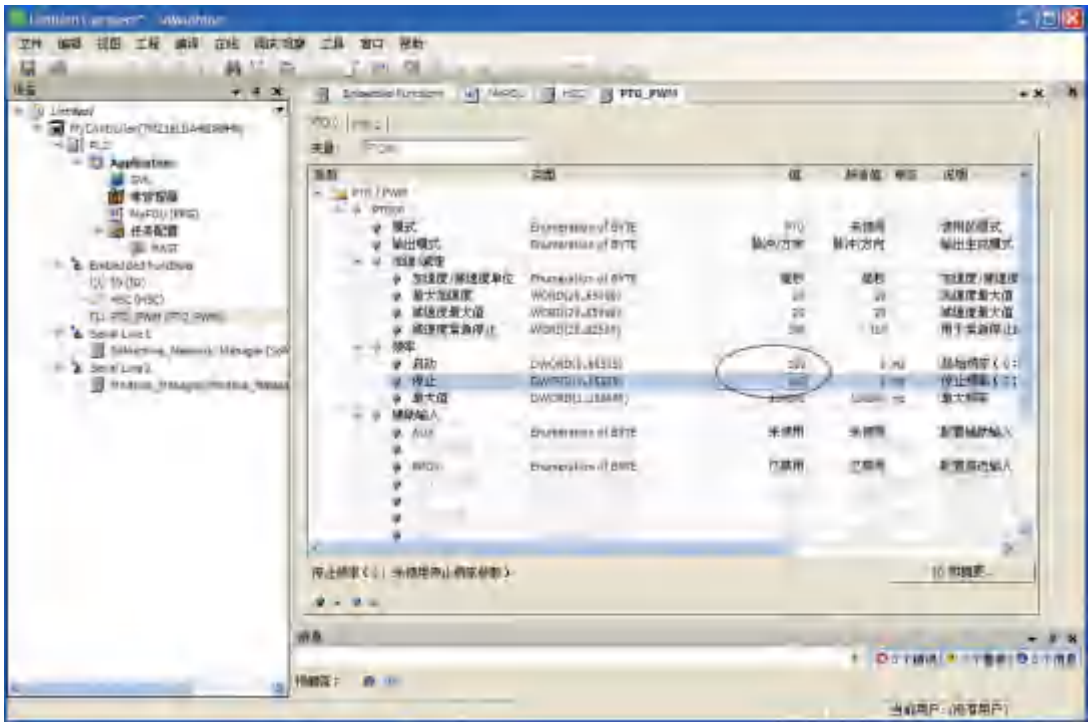
三.M218的PTO配置
1.如图, 在SoMachine左侧导航栏内选择“嵌入功能”下的PTO选项:



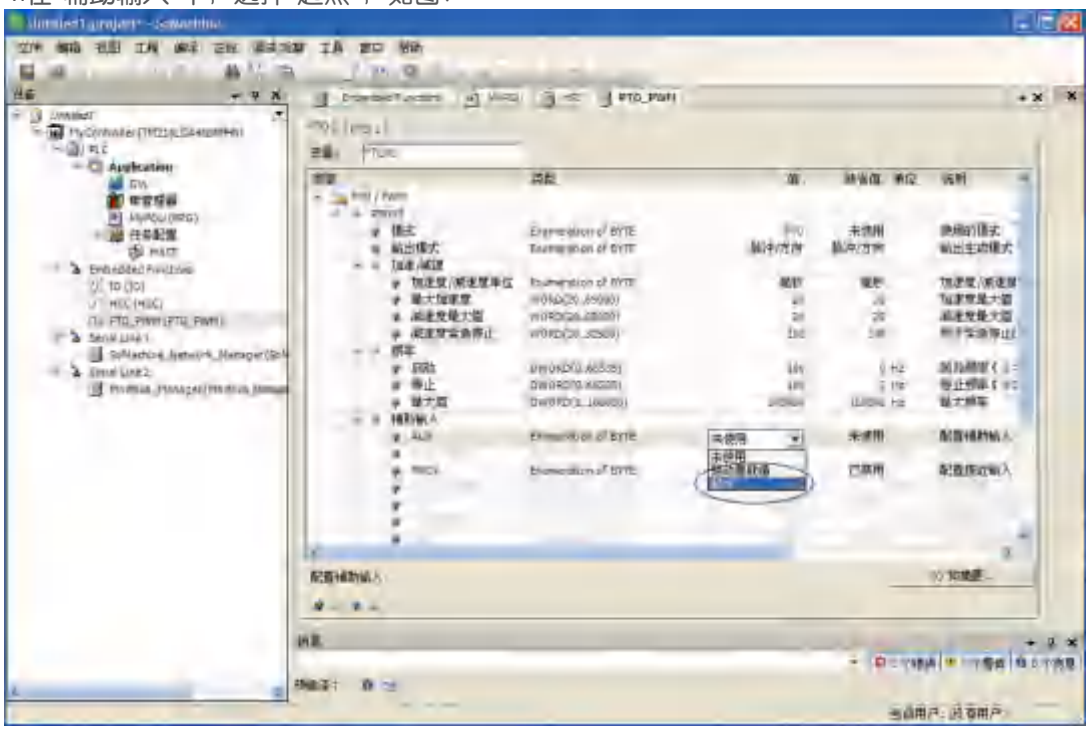
2.在右侧“PTO”参数设定画面内将“模式”改为PTO，如图：



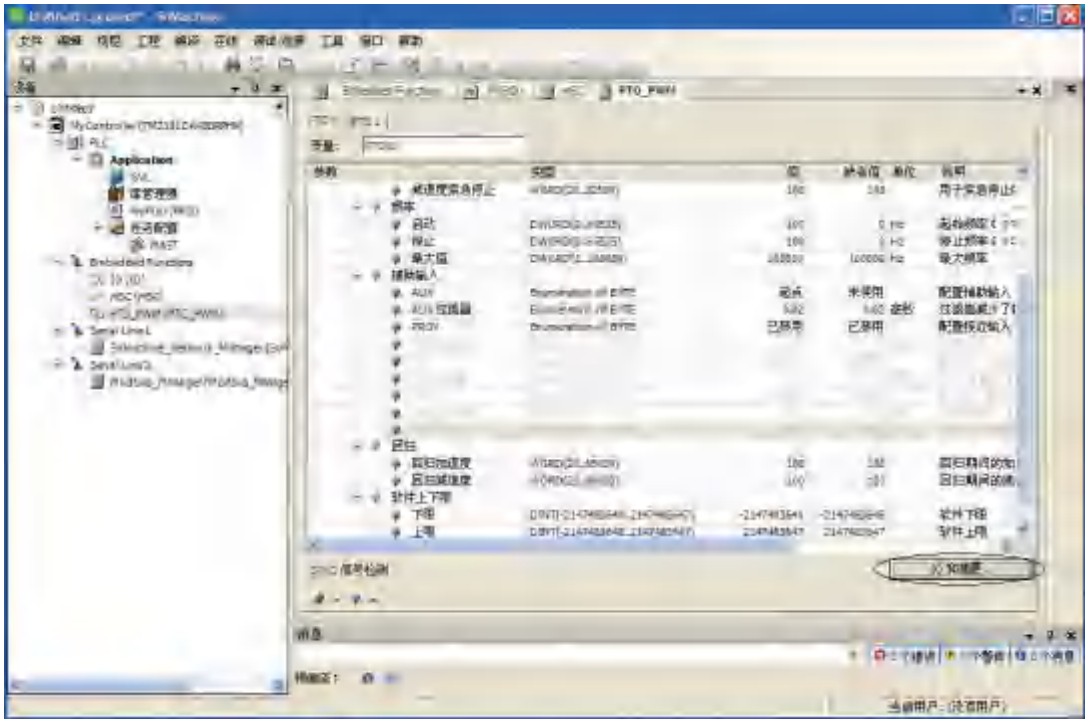
3.将“频率”中的“启动”，“停止”都设为100，如图：



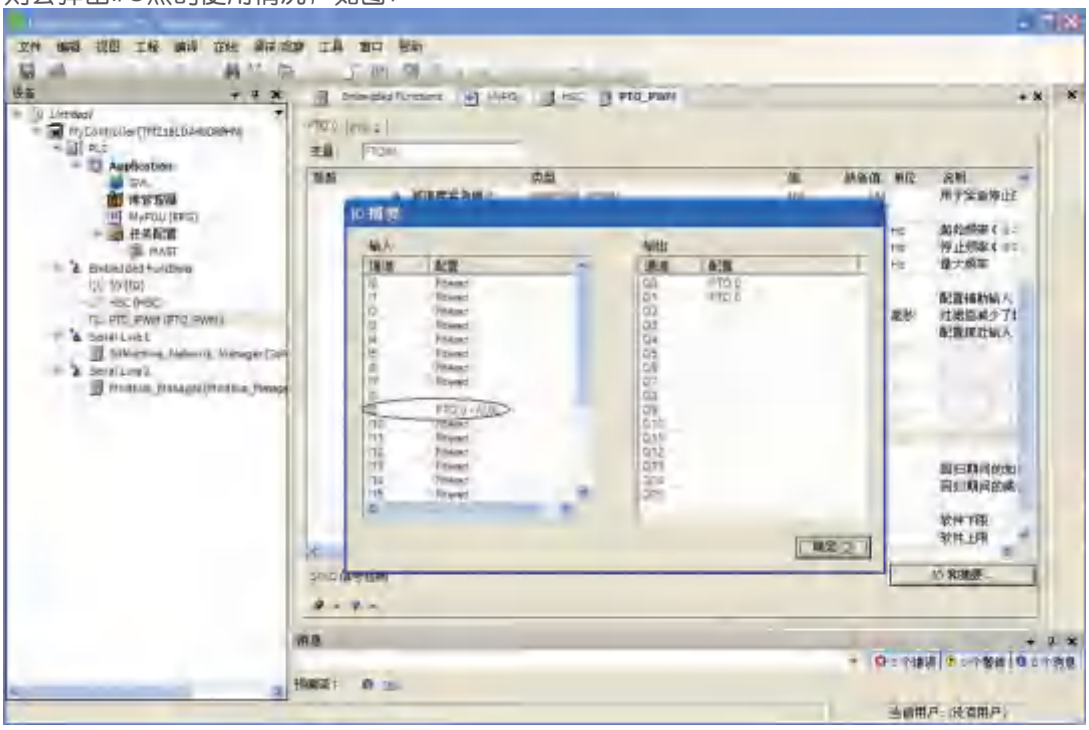
4.在“辅助输入”中，选择“起点”，如图：



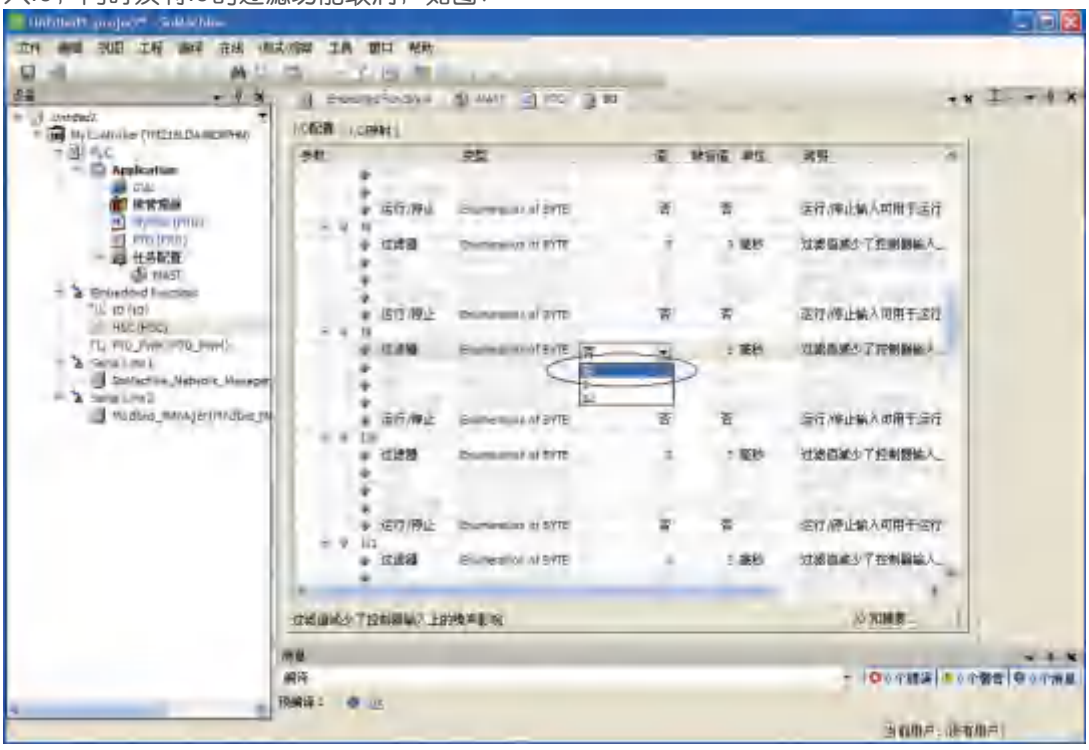
5.点击右下角“IO 和摘要”，如图：



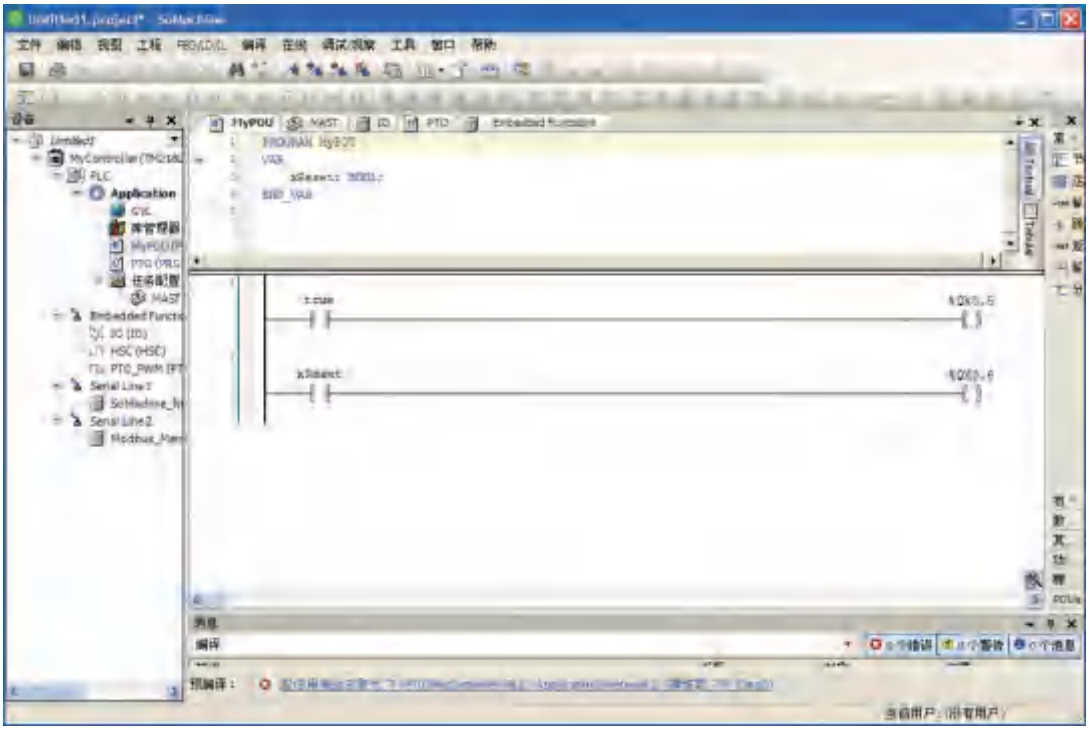
则会弹出I/O点的使用情况，如图：



选择了寻原点功能，系统自动为I9配置了外部的原点输入信号，将外部的原点信号接入I9，同时须将I9的过滤功能取消，如图：



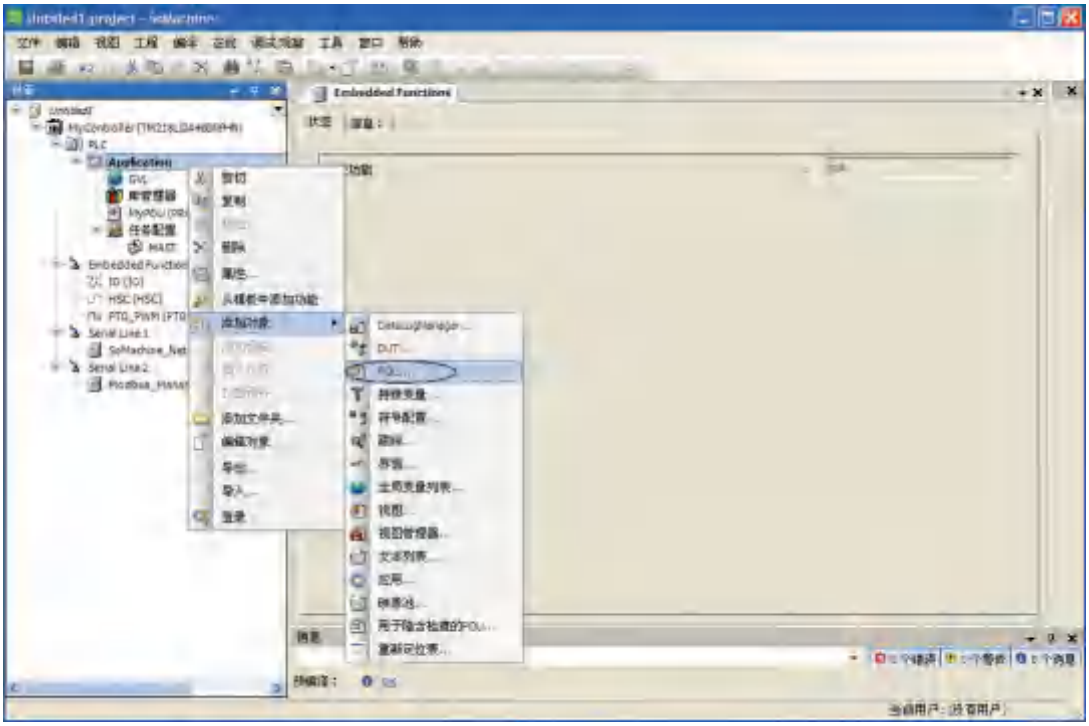
四．编程
1．逻辑控制
在左侧导航栏内的Application中双击MyPou，建立程序控制伺服启动，及报警复位，如图：



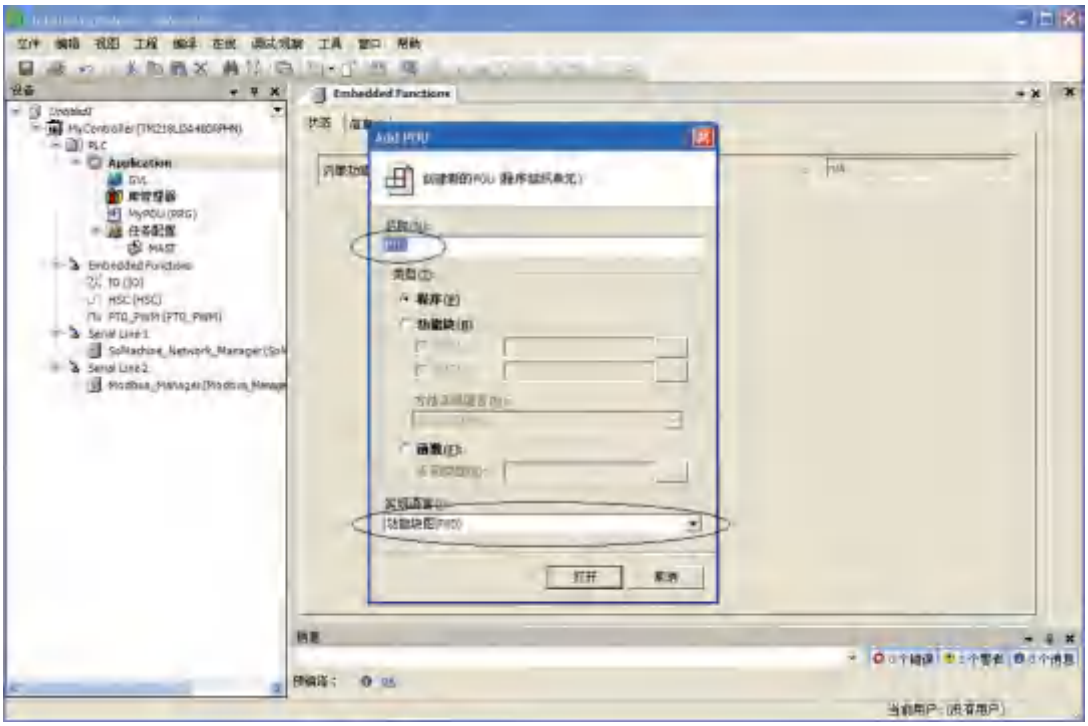
A - 将%QX0.5常置为TRUE，即控制器上电即让伺服处于Servo On的状态
B - 使用变量xReset进行伺服的复位

2．建立新POU

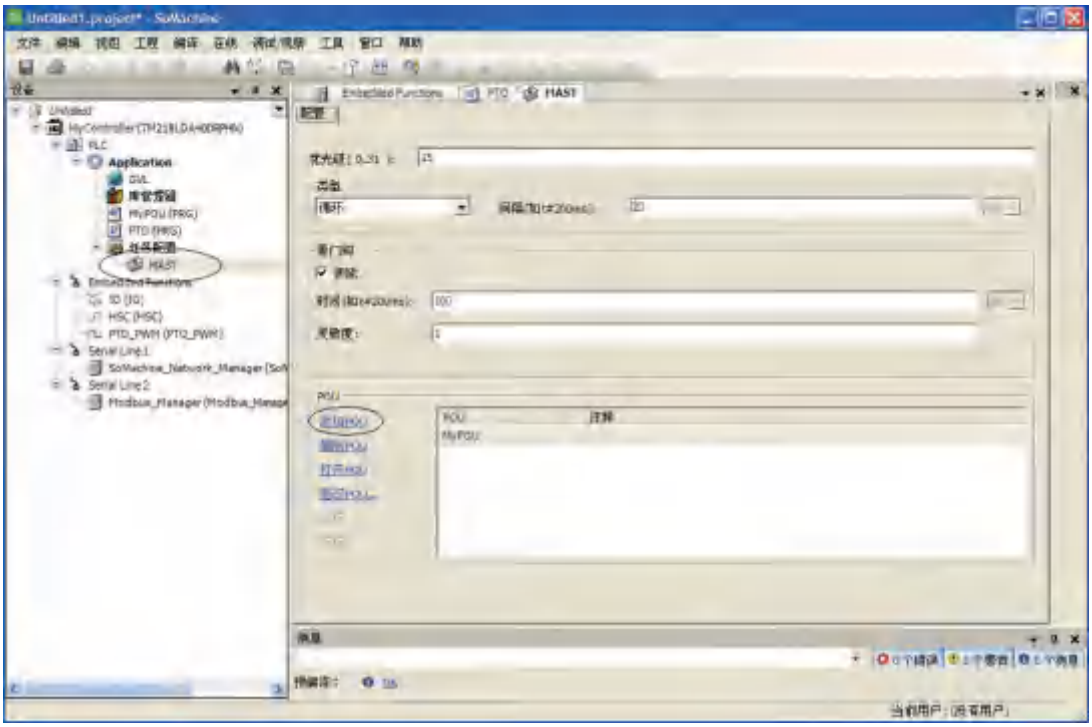
1）. 右击Application，在“添加对象”中点击POU选项，如图：



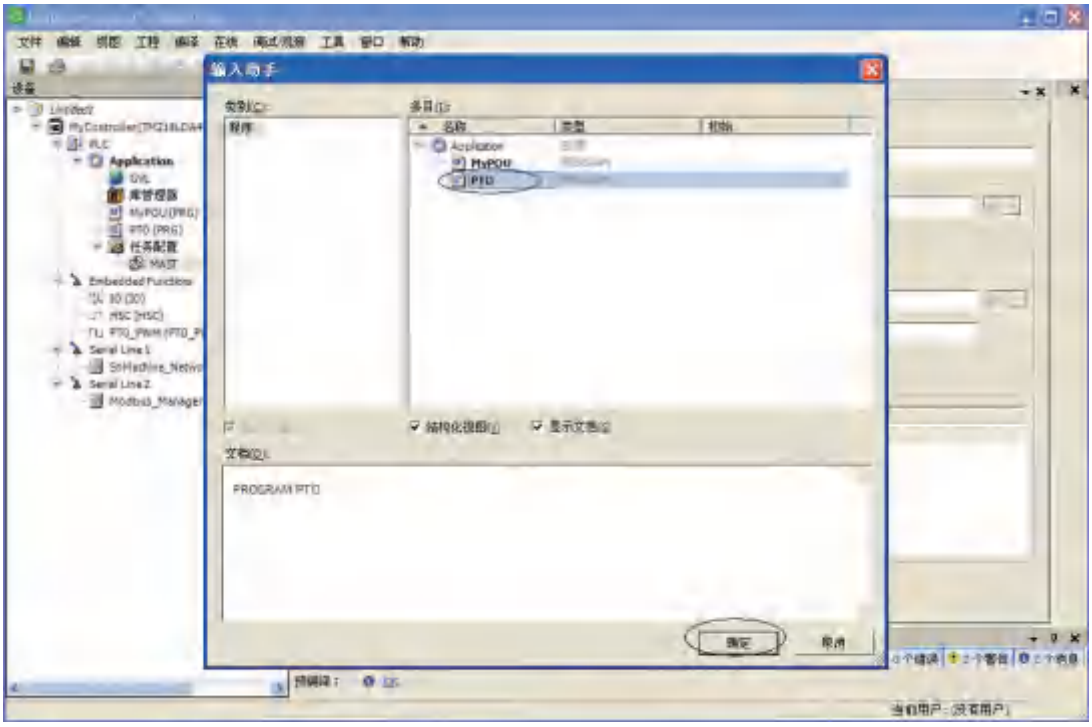
在弹出的对话框中，填写POU的名称，在此改为PTO；在“实现语言”中选择FBD，如图：

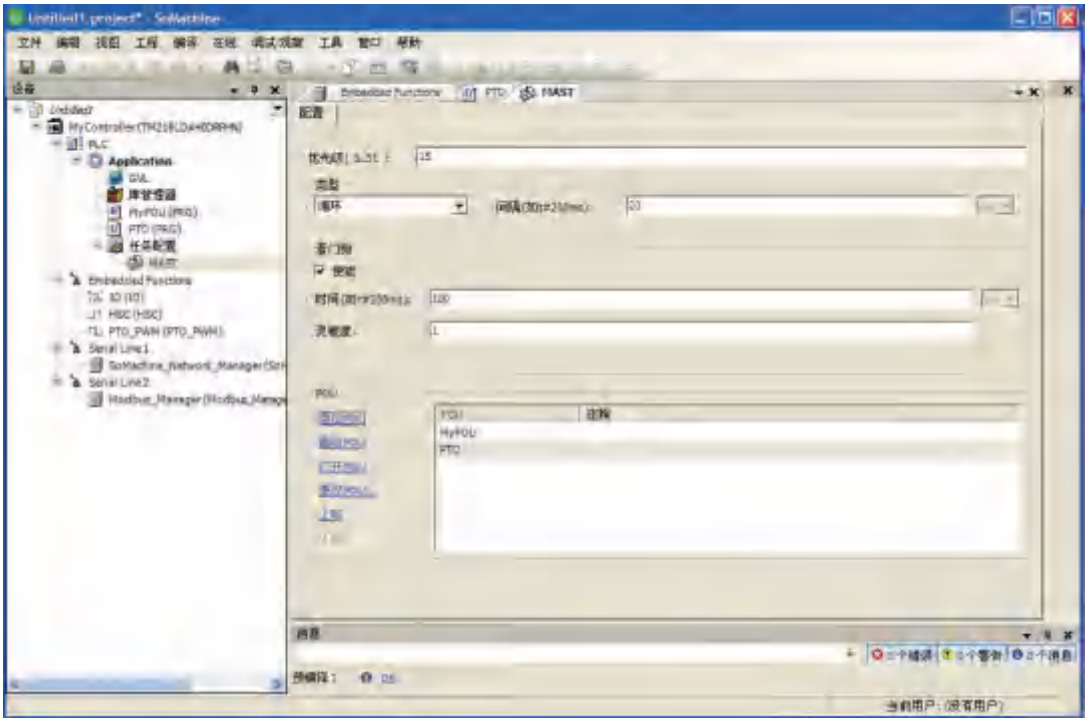


2）. 双击右侧导航栏中“任务配置”下的MAST，在右侧的MAST任务配置中点击“添加POU”如图：



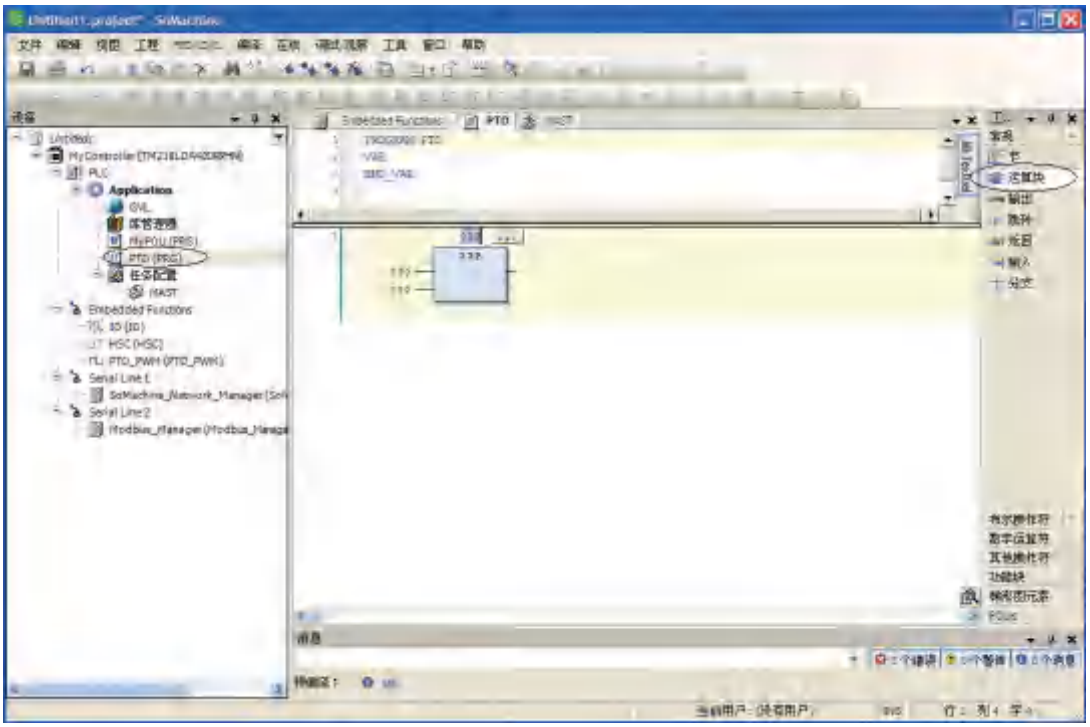
选择刚才建立的PTO，并点击“确认”，将其添加入扫描任务中，如图：



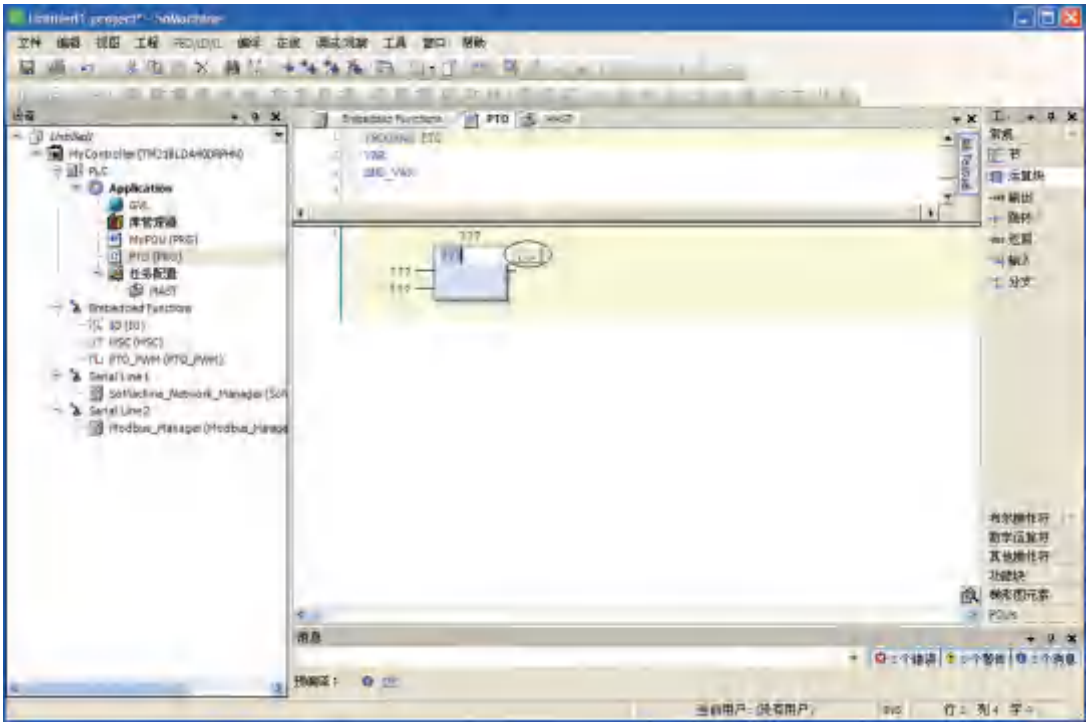


3．功能块PTOSimple的应用

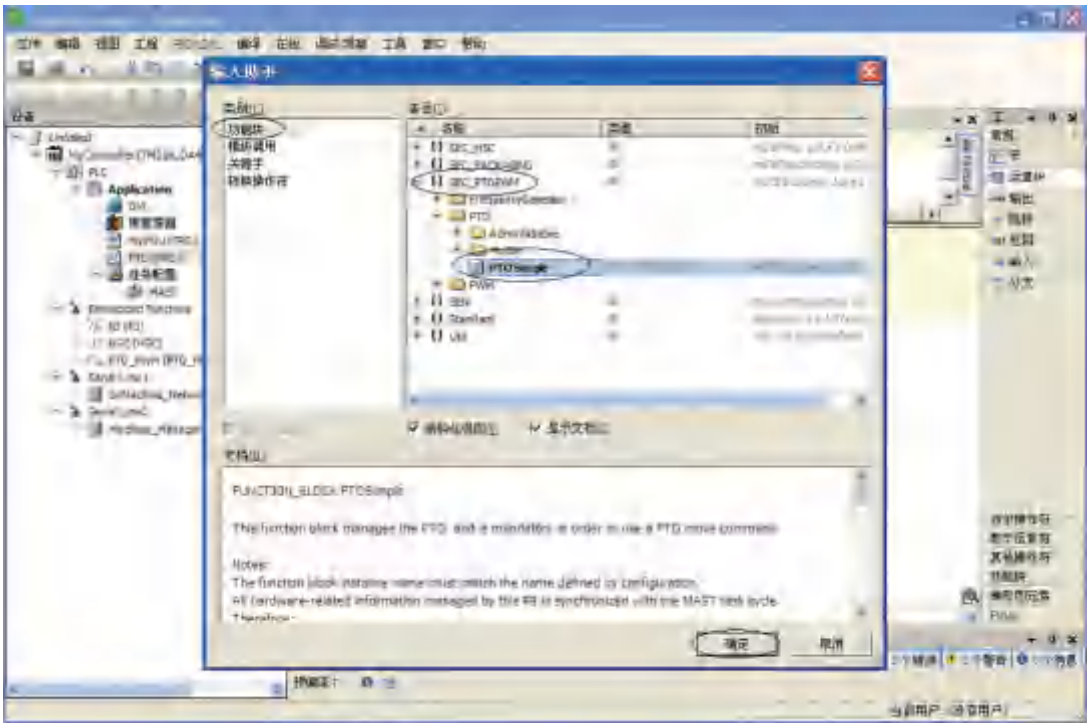
1）.双击导航栏中的PTO程序块，然后在右侧的工具栏中选择“运算块”，将其拖入程序编辑区域中，如图：



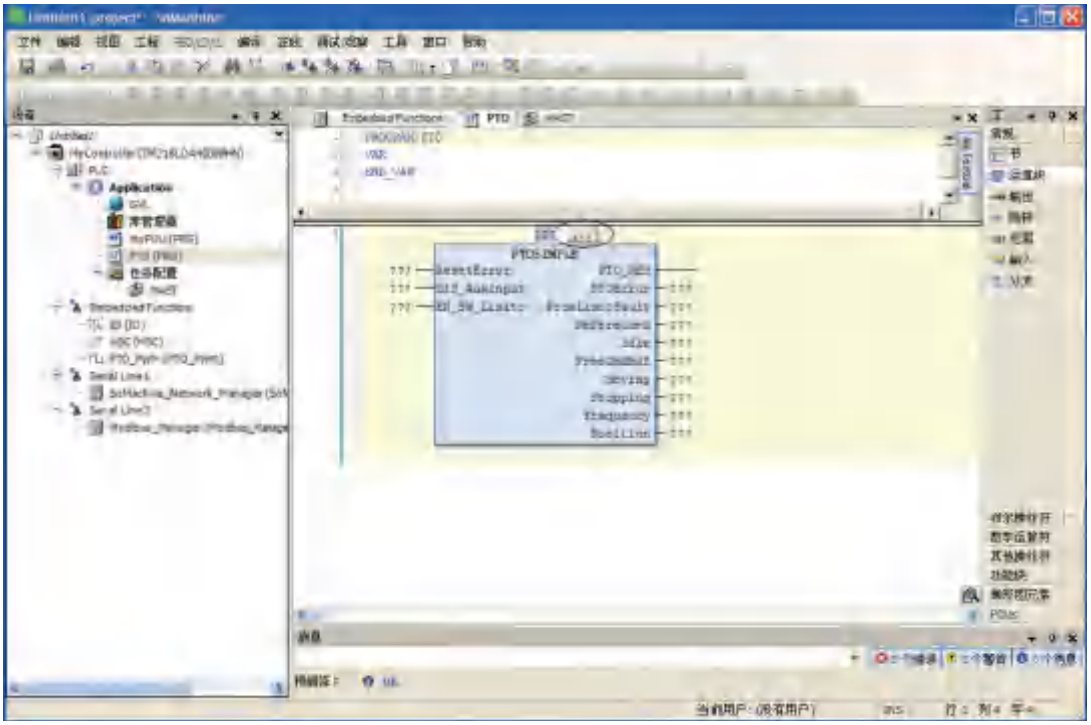
2）.点击空功能块名称，可以使用“输入助手”，点击进入功能块选择帮助，如图：



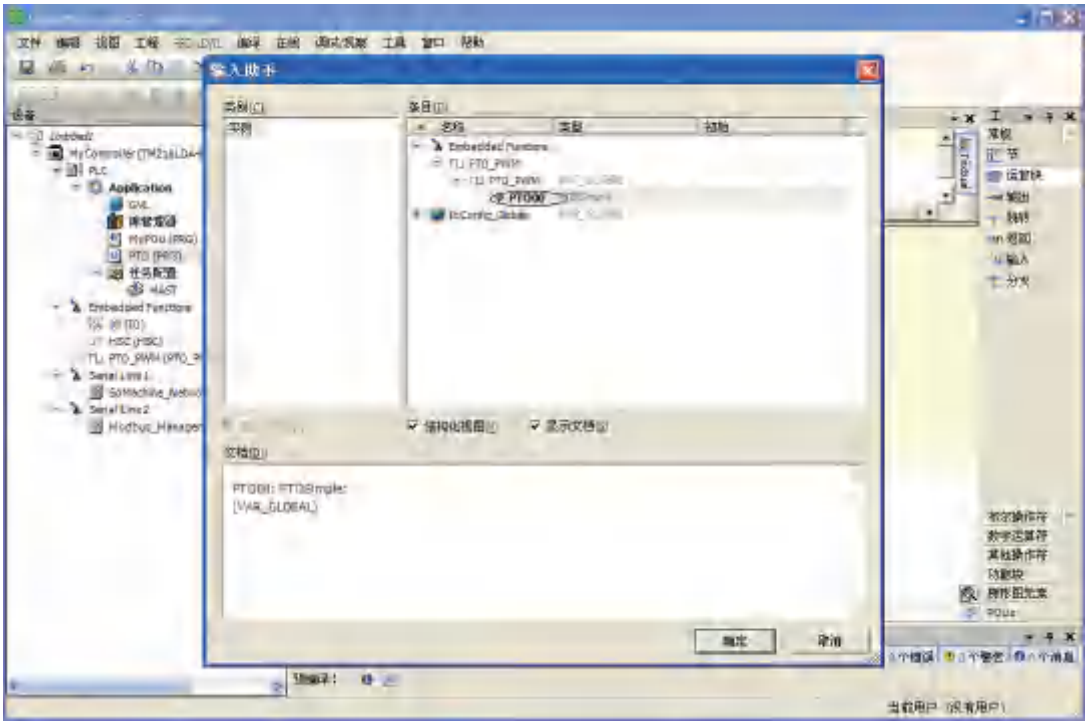
在弹出画面中，选择“功能块”，在右侧的条目中选择SEC_PTOPWM下PTO文件夹中的PTOSimple功能块，点击“确定”，如图：

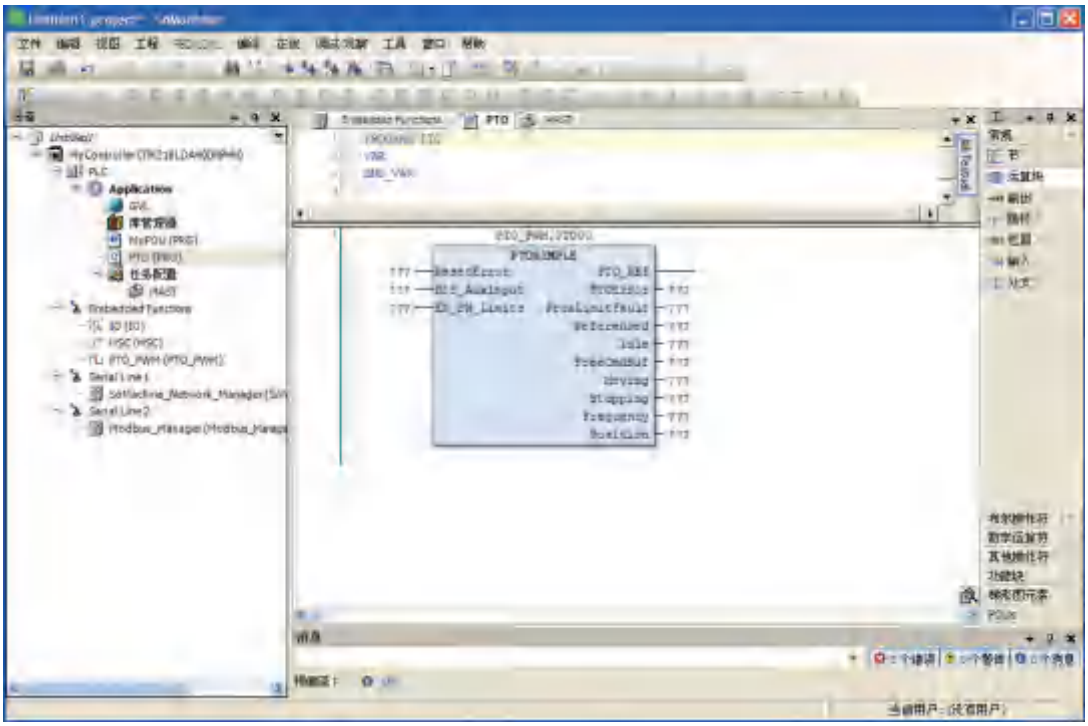


3）.随后点击功能块的声明对话框，同样进入输入助手，如图：

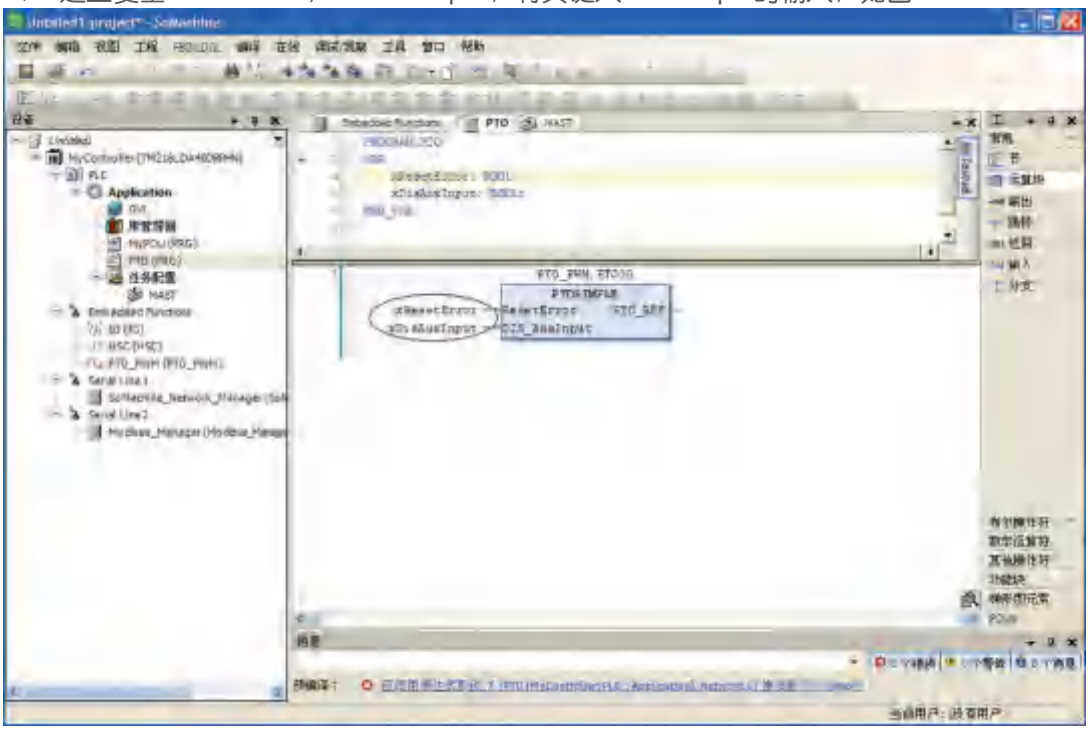


在“输入助手”中，选择Embedded Functions，并在其中找到PTO00，通过此操作建立功能块与设备硬件的连接，如图：





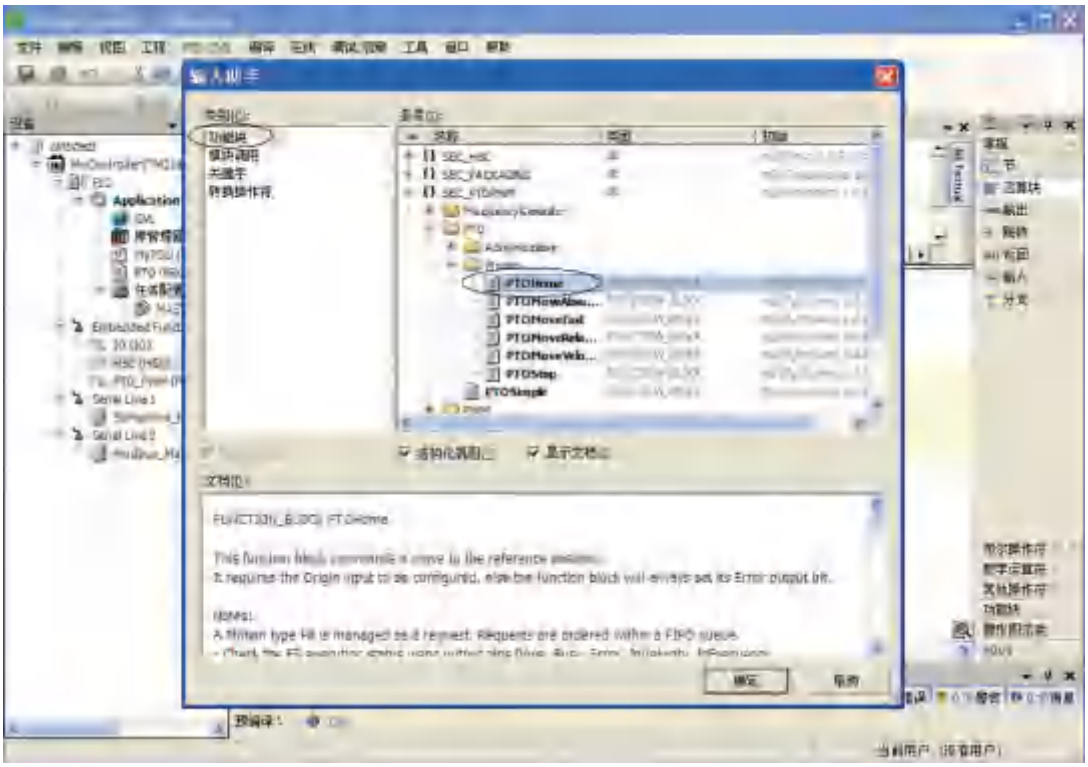
4）. 建立变量xResetError，xDisAuxInput，将其键入PTOSimple的输入，如图：



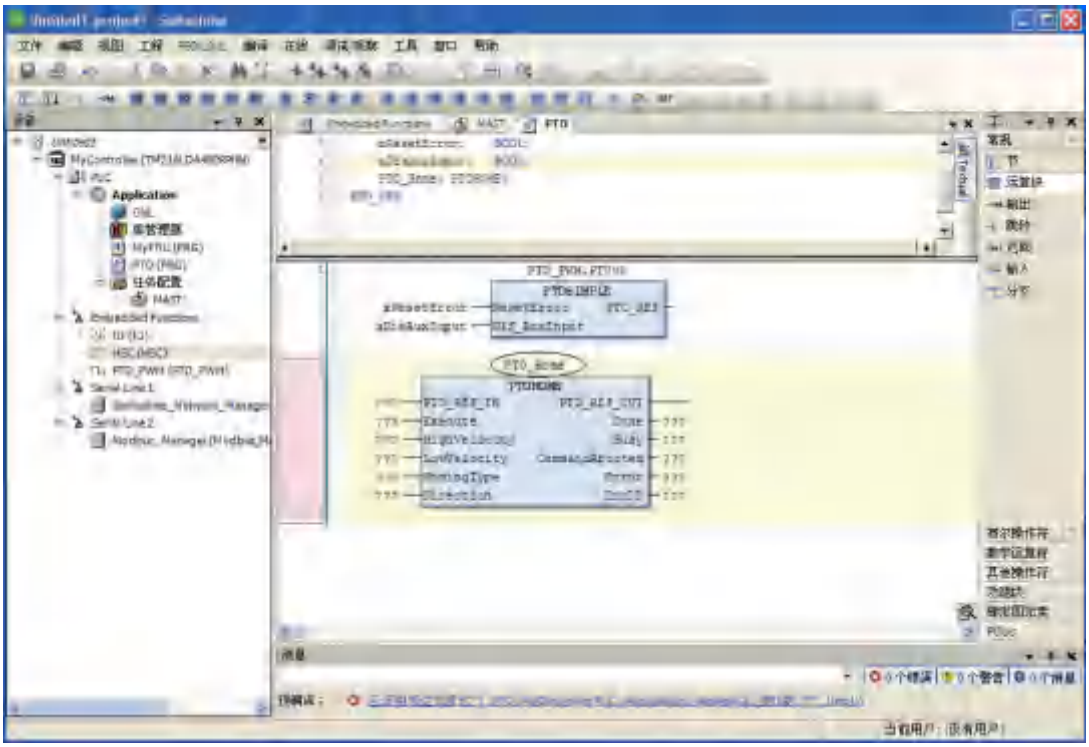
A - xResetError可对使用该通道的PTO功能块的错误进行复位
B - xDisAusInput用于屏蔽辅助输入点

4．PTO的寻零控制

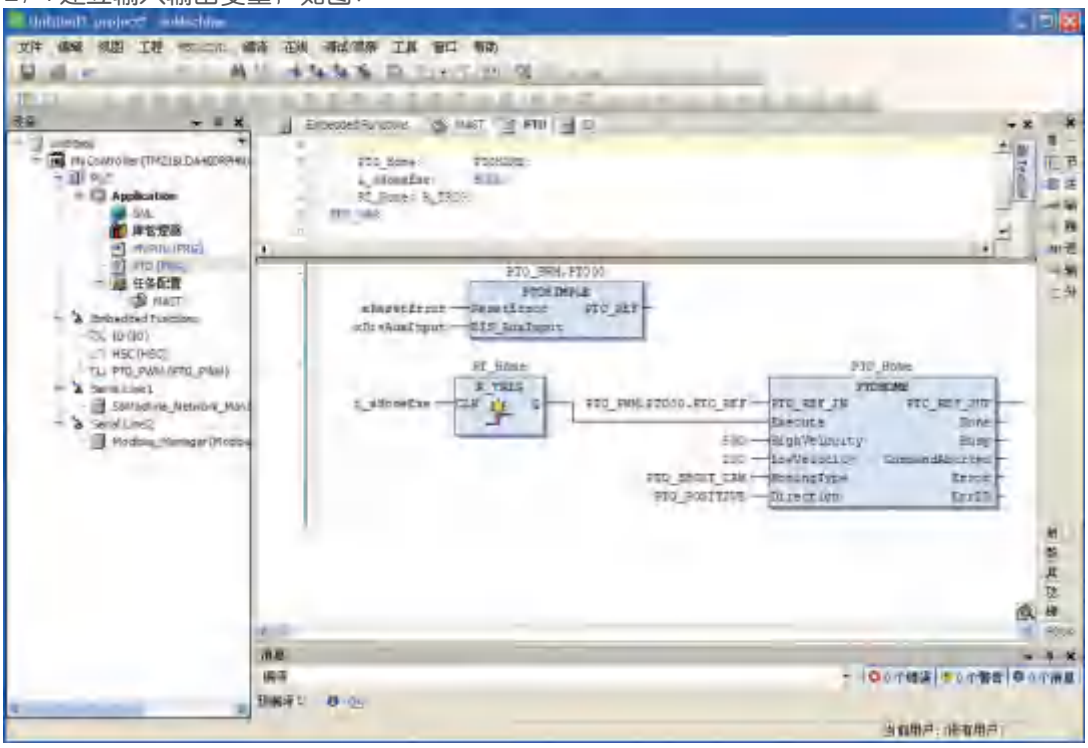
1）. 如上所述先建立PTOSimple功能块，然后在其下添加新的功能块，在库中选择PTOHome，如图所示：



对功能块进行变量声明，如图：



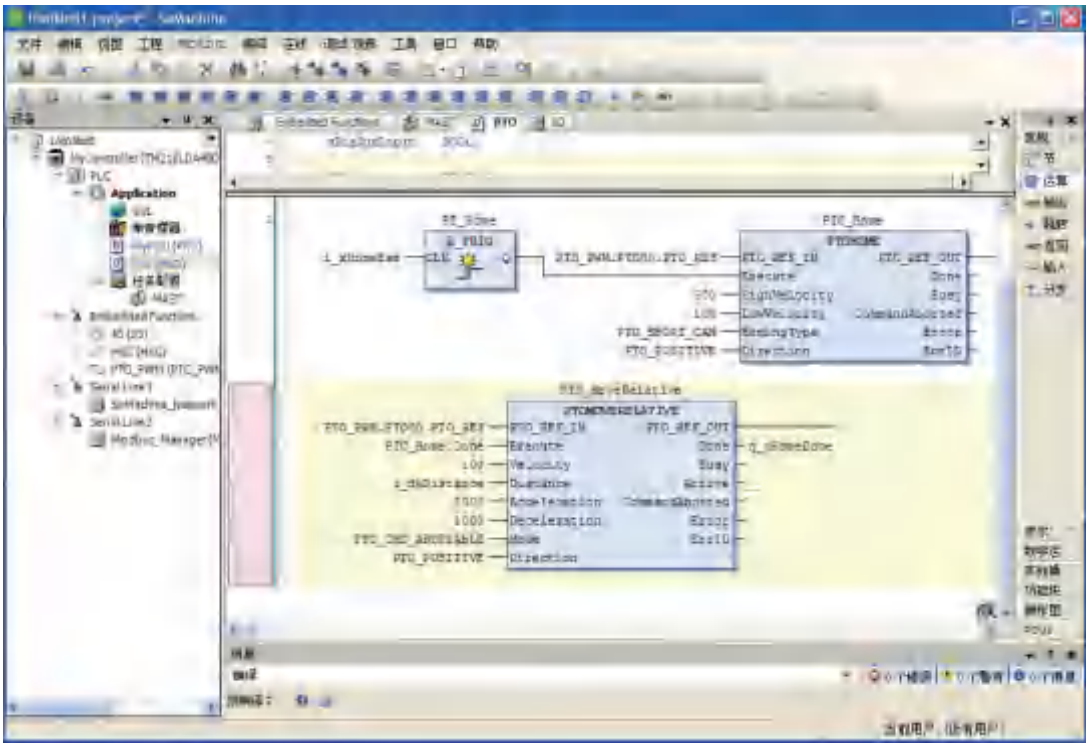
2）. 建立输入输出变量，如图：



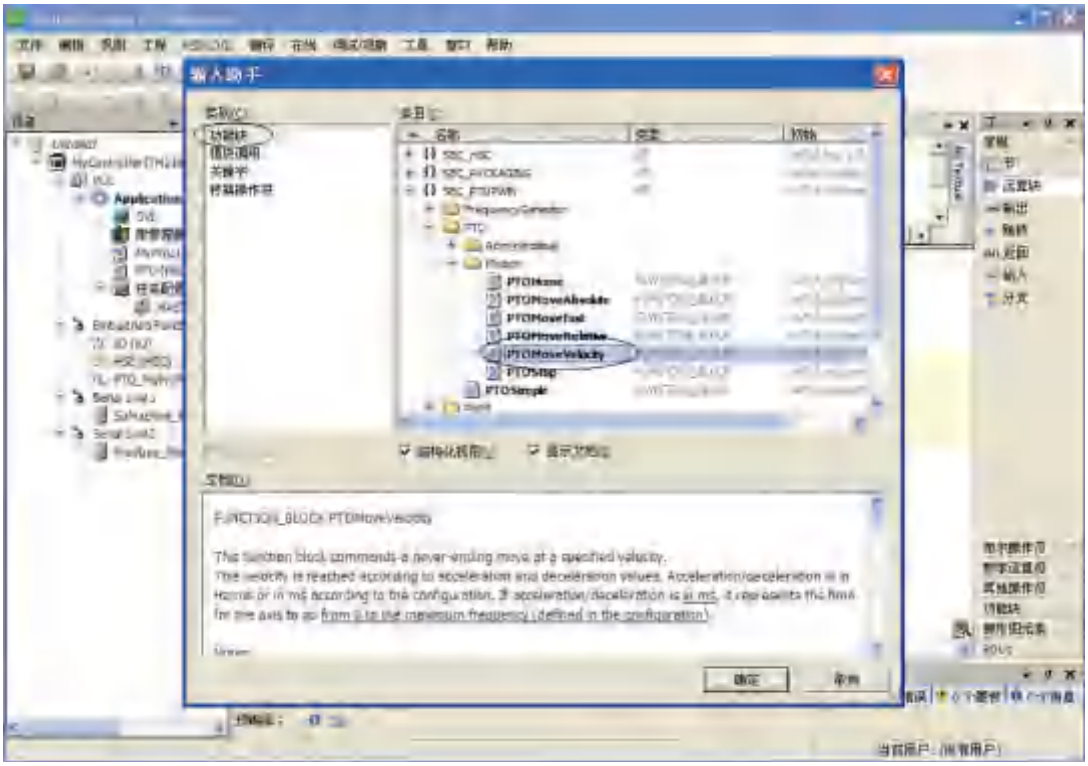
A - PTO_Home的输入PTO_REF_IN需连接对应通道的PTOSimple功能块的输出PTO_REF_OUT
B - 变量i_xHomeExe作为触发寻零动作的变量，功能块只接收变量的上升沿信号，因此可添加一个上升沿触发功能块
C - HomeType处需填写所需进行寻零的方式，具体见寻零模式介绍，此处我们填写短凸轮模式，即PTO_SHORT_CAM

3），在实际应用中，很多时候机械参考点不一定就是设备运行的零点，往往需要电机到达机械参考点后在做一段位
置偏移，以此作为实际运行的原点。对于此种应用，可在原程序上进行如下修改：
a) 添加一位移功能块
b）将PTOHome的Done输出信号作为位移功能块的触发变量
c）将位移功能块的Done输出信号作为整个寻零过程的完成标志位
d）如果需要，可在将位移功能块的Done信号连接到PTOSetPosition的触发信号，进行位置置位

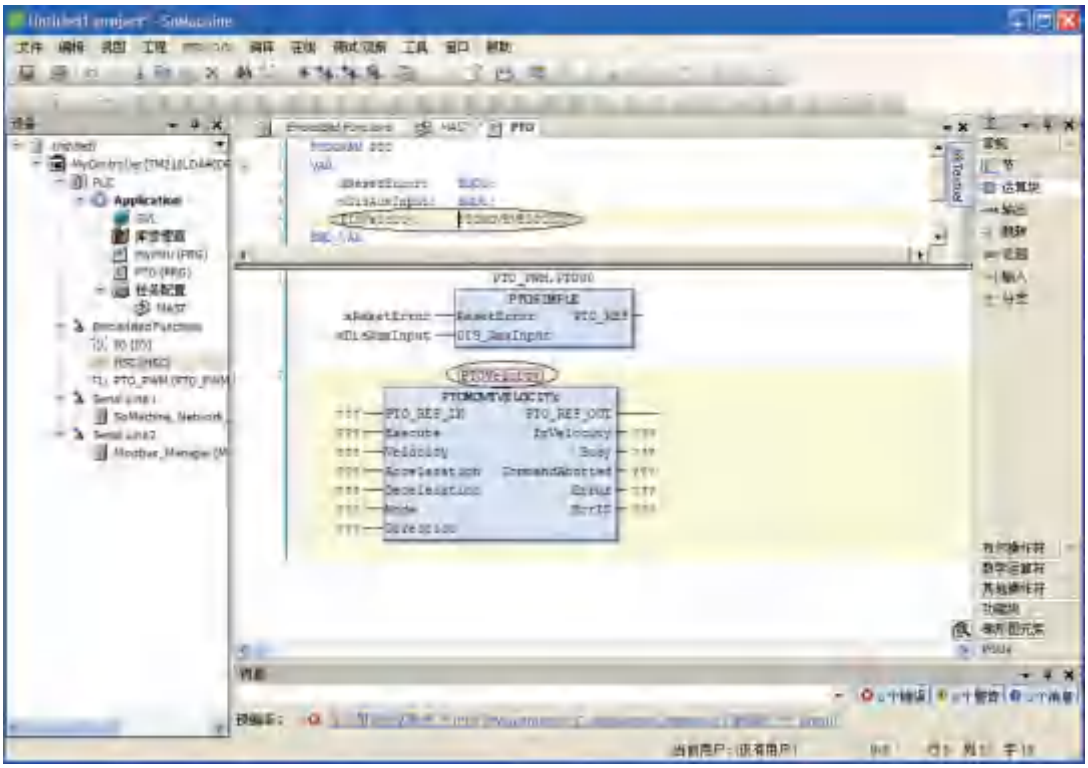
具体见下图：



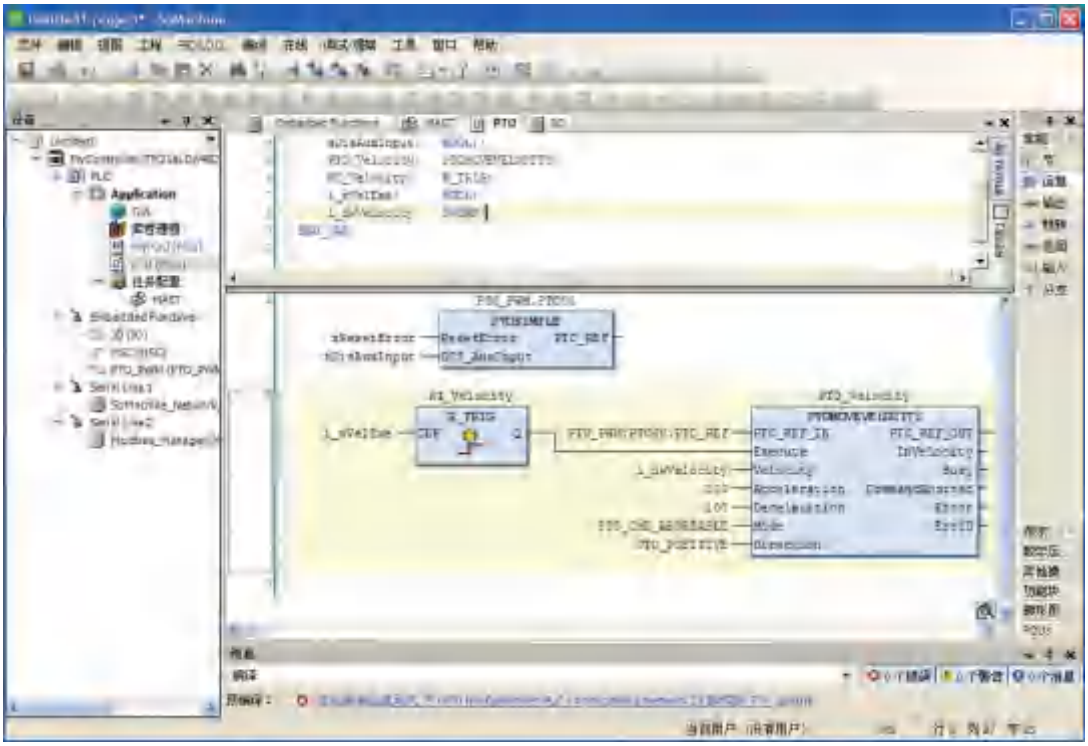
5．PTO的速度控制
1）. 如上所述先建立PTOSimple功能块，然后在其下添加新的功能块，选择PTOMoveVelcity，如图所示：



对功能块进行变量声明，如图：

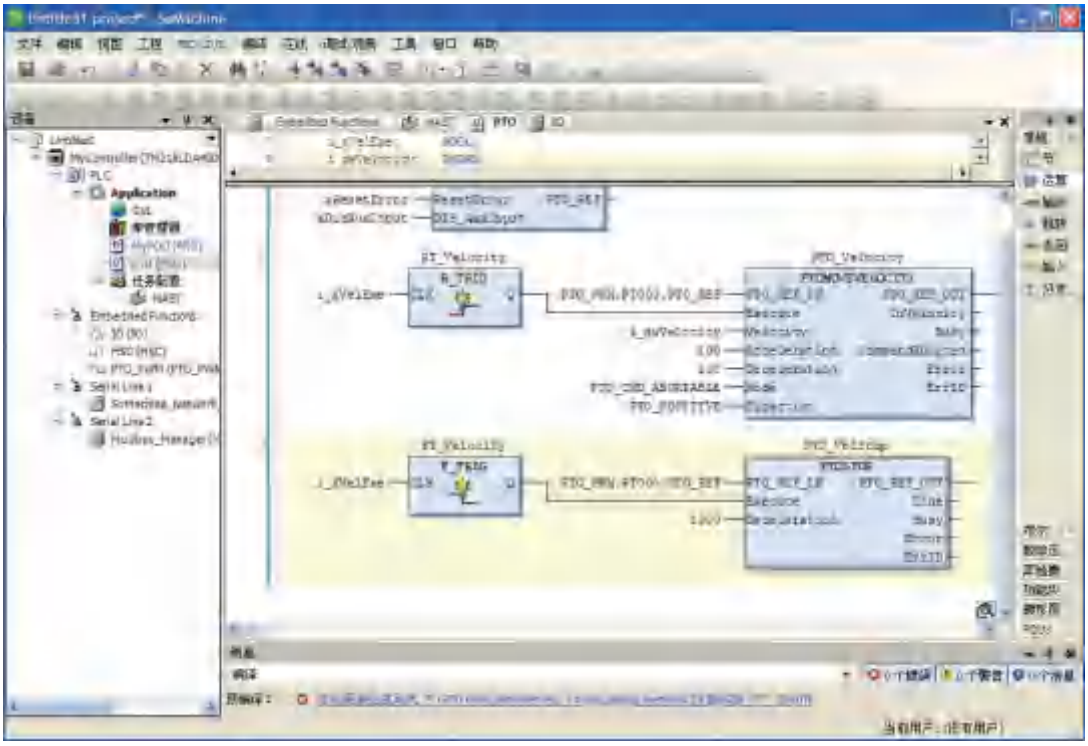


2）. 建立输入输出变量，如图：



A - i_xVelExe为功能块执行变量；
B - i_dwVelocity为速度给定变量，通过改变此变量的值，同时配合触发i_xExe可改变PTO的脉冲输出速度，以此改变伺服电机的转速；

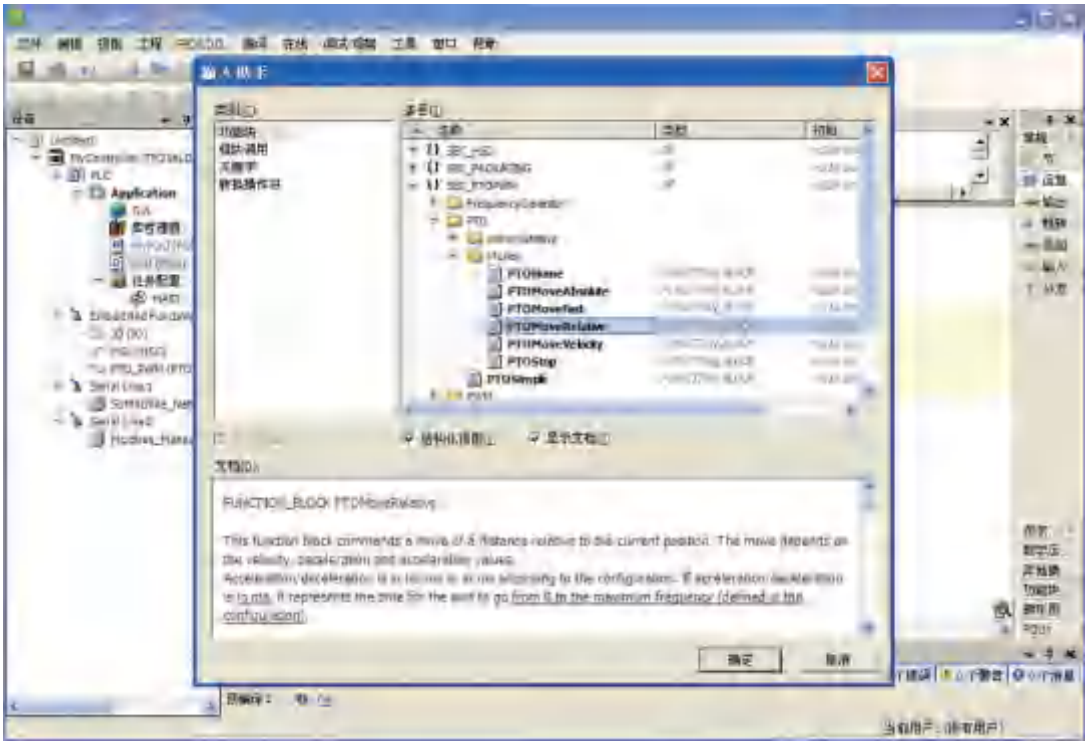
3）.但如果在执行速度控制时我们想进行停机控制，则必须使用PTOStop功能块，而不能只简单地将速度降为零，见下图：

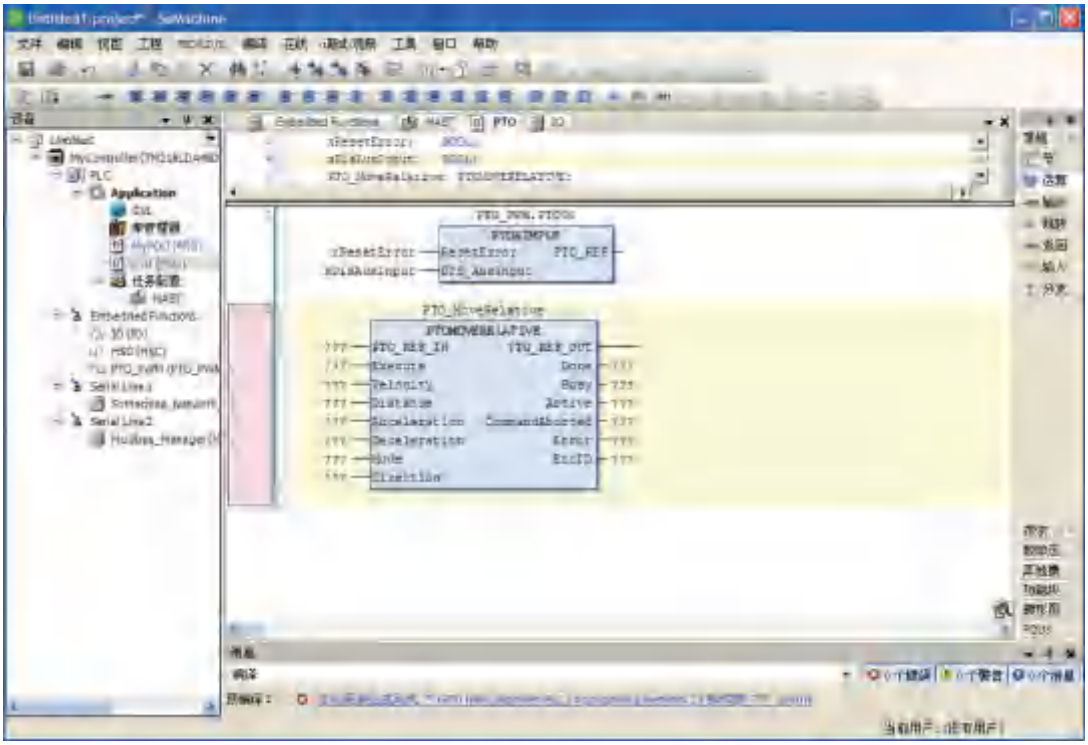


如图所示，我们可以取变量i_xVelExe的下降沿作为触发停止功能块的变量，这样就可以实现通过一个变量的0和1来实现控制伺服进行速度控制

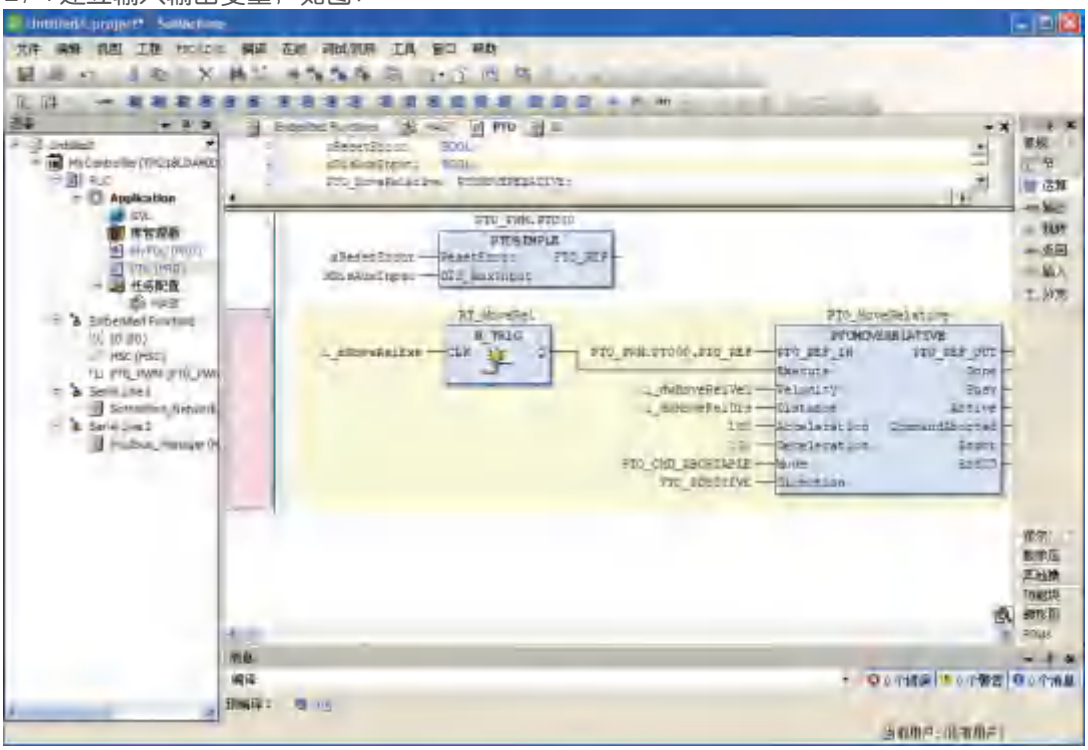
6．PTO的相对位置控制

1）. 如上步骤冲PTO库中选取PTOMoveRelative，建立功能块，如图：





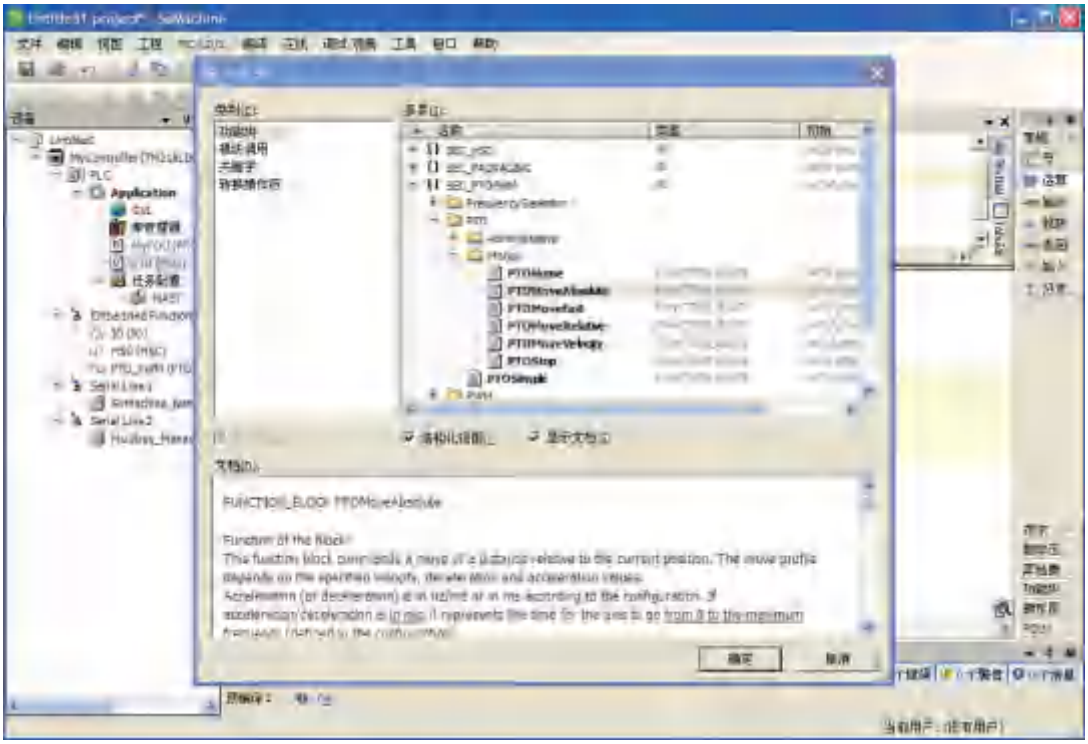
2）. 建立输入输出变量，如图：

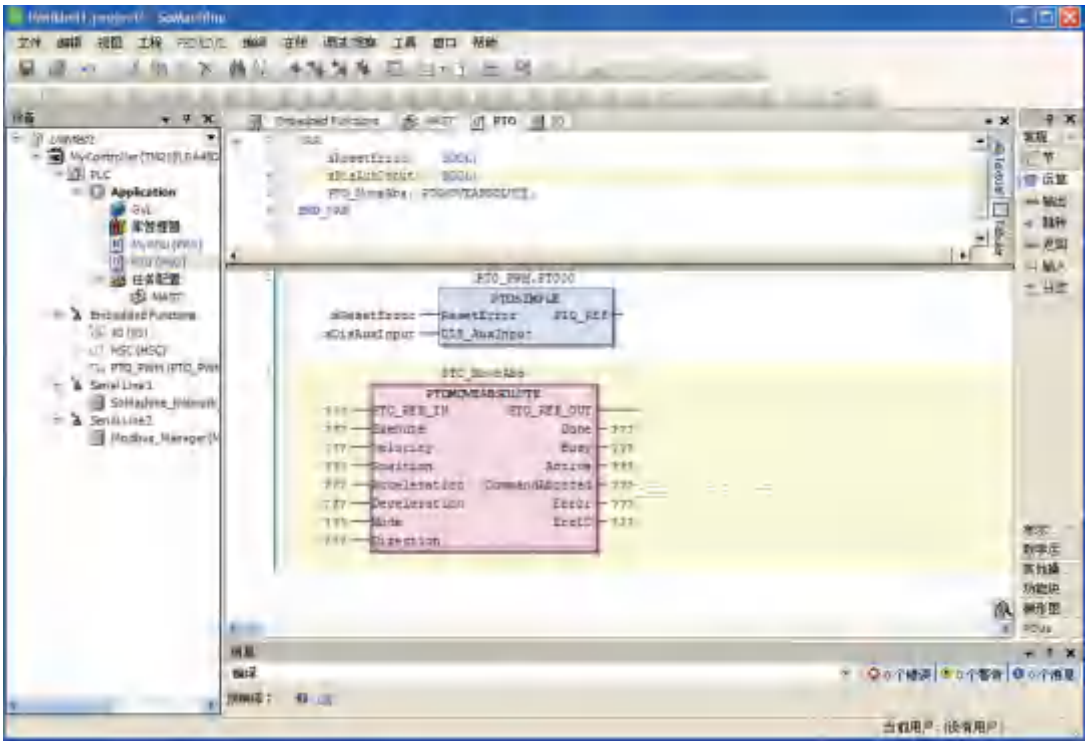


A - 变量i_xMoveRelExe为功能块触发变量
B - 变量i_dwMoveRelVel用于设定电机执行相对位移时的速度
C - 变量i_dwMoveRelDis用于设定电机执行的相对位移的实际距离

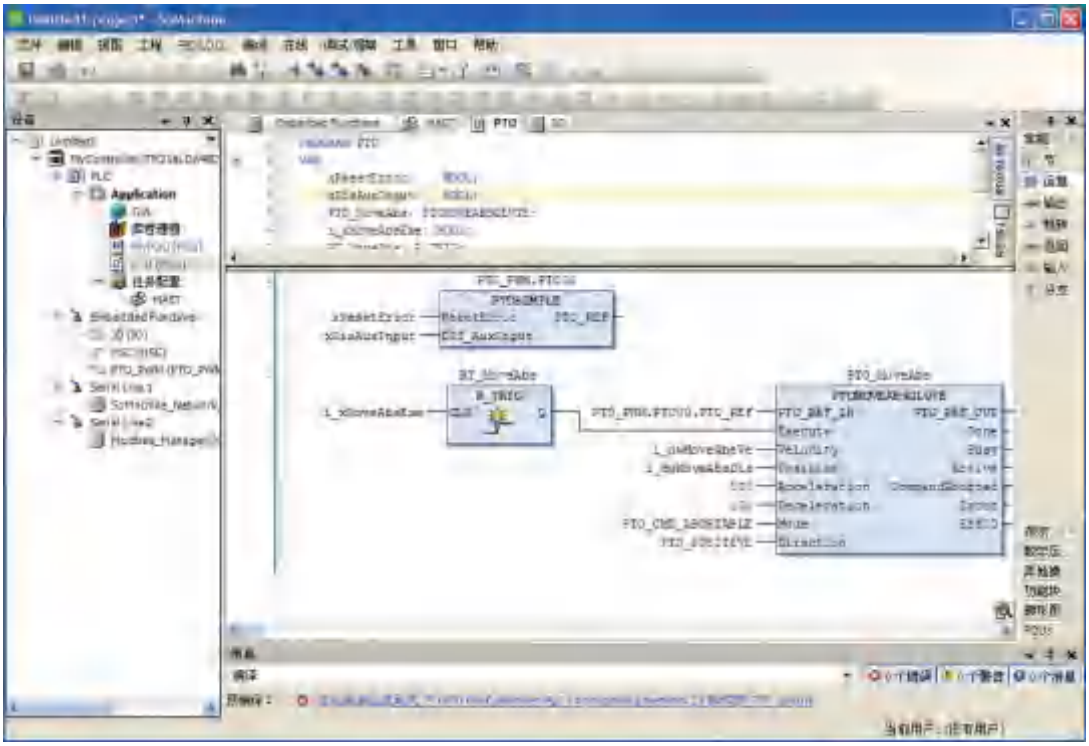
7．PTO的绝对位置控制

1）. 如上步骤冲PTO库中选取PTOMoveAbsolute，建立功能块，如图：



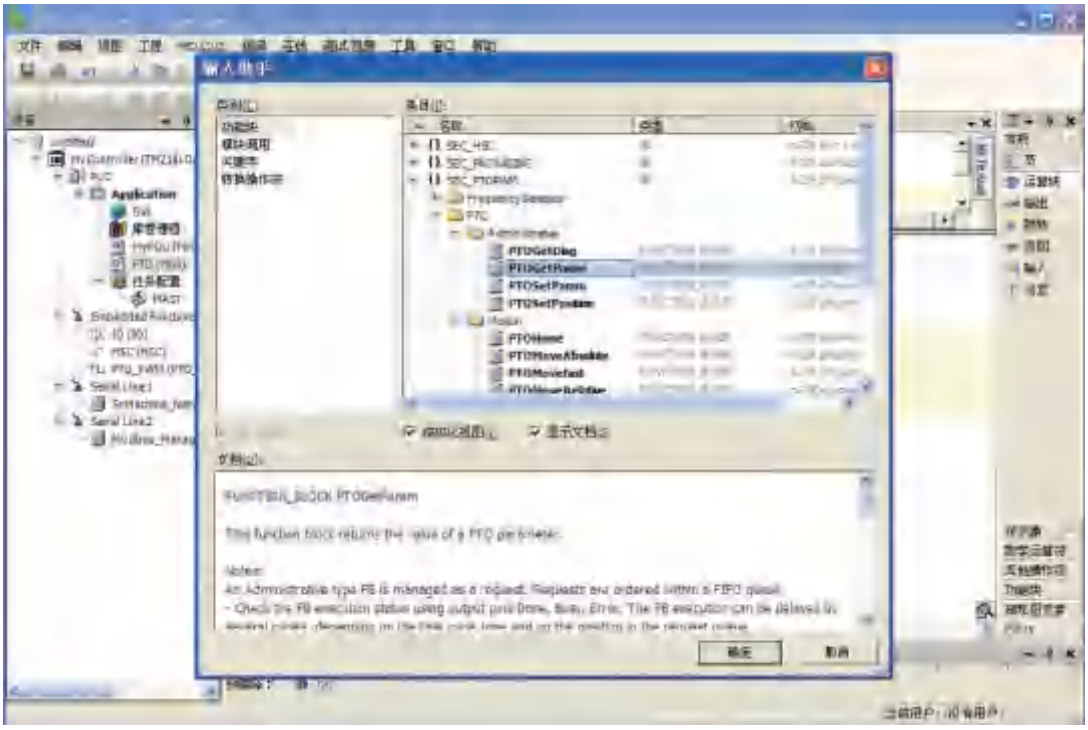


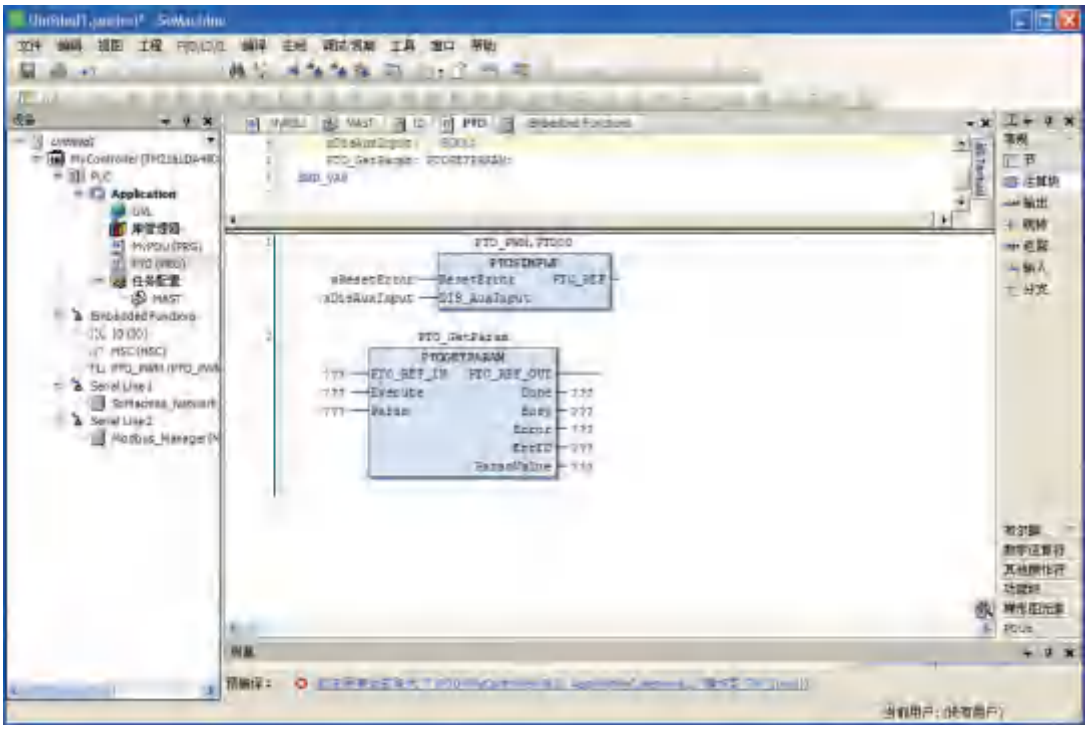
2）. 建立输入输出变量，如图：



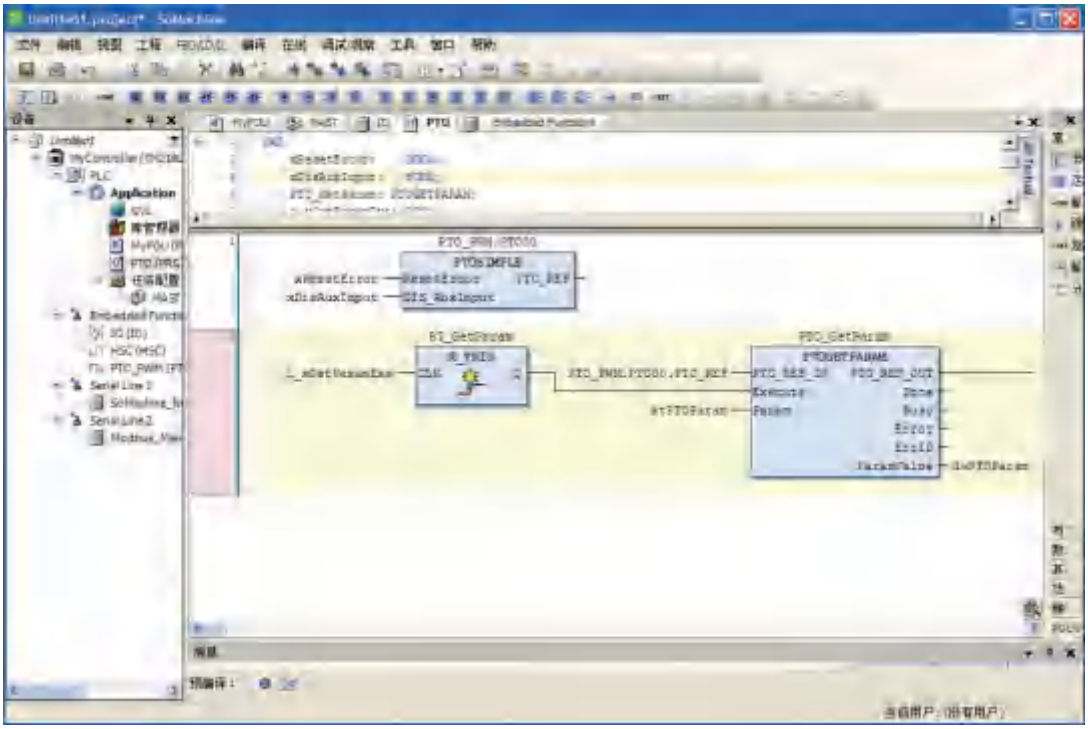
A - 变量i_xMoveAbsExe为功能块触发变量
B - 变量i_dwMoveAbsVel用于设定电机执行相对位移时的速度
C - 变量i_dwMoveAbsDis用于设定电机执行的相对位移的实际距离

8．PTO参数读取功能块
1）. 从PTO库中选取PTOGetParam功能块，如图：





2）. 建立输入输出变量，如图：



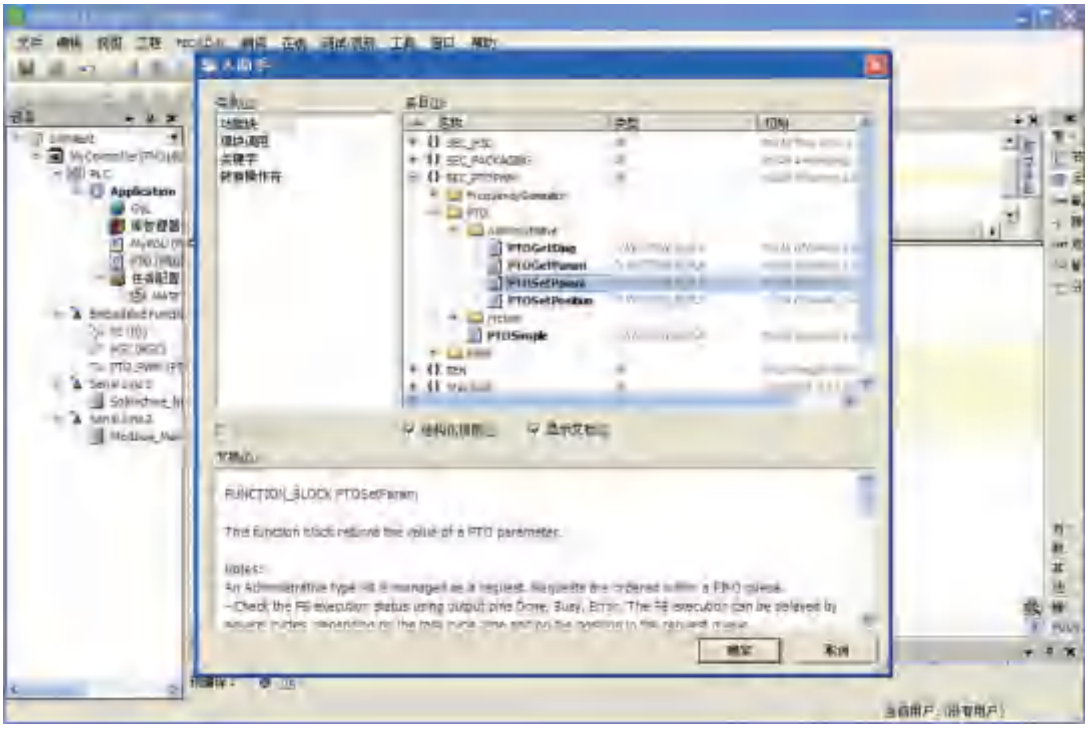
A - 变量i_xGetParamExe为功能块触发变量
B - 变量stPTOParam为PTO_PARAMETER_TYPE类型
C - 变量dwPTOParam为具体读取的数值

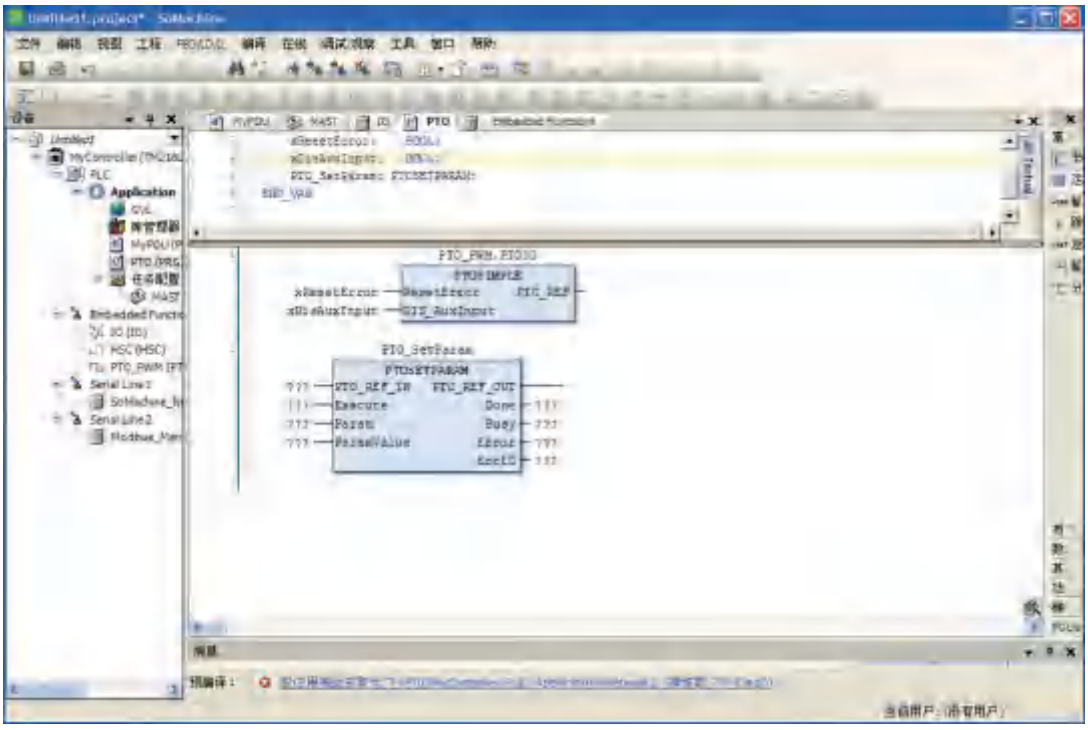
枚举变量PTO_PARAMETER_TYPE具体类型如下：

PTO_START_FREQUENCY	00	PTO启动速度
PTO_STOP_FREQUENCY	01	PTO停止速度
PTO_EMY_DEC	02	PTO紧急停止速度

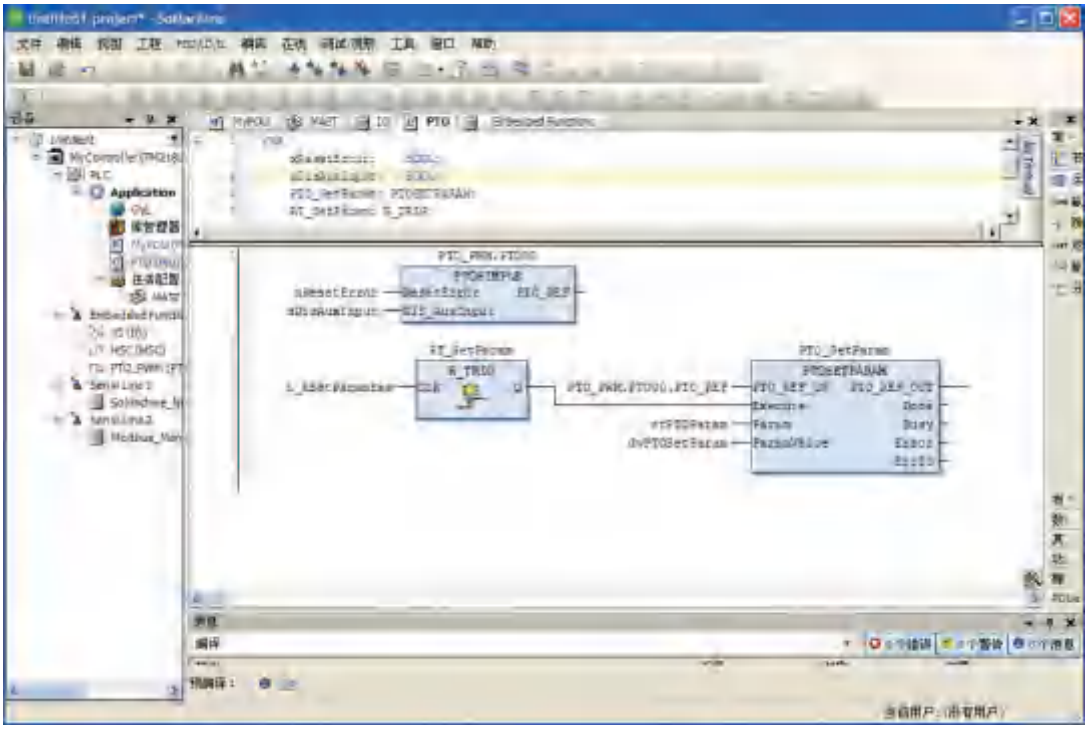
9．PTO参数调整功能块

1）. 从PTO库中选取PTOSetParam功能块，如图：



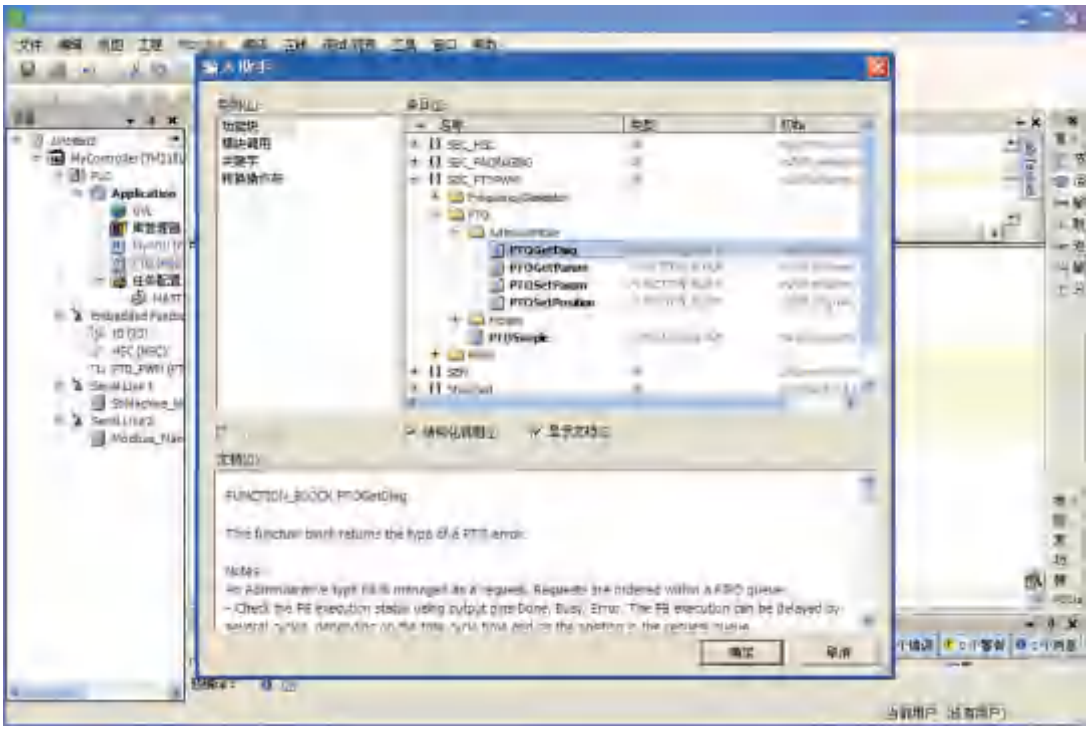


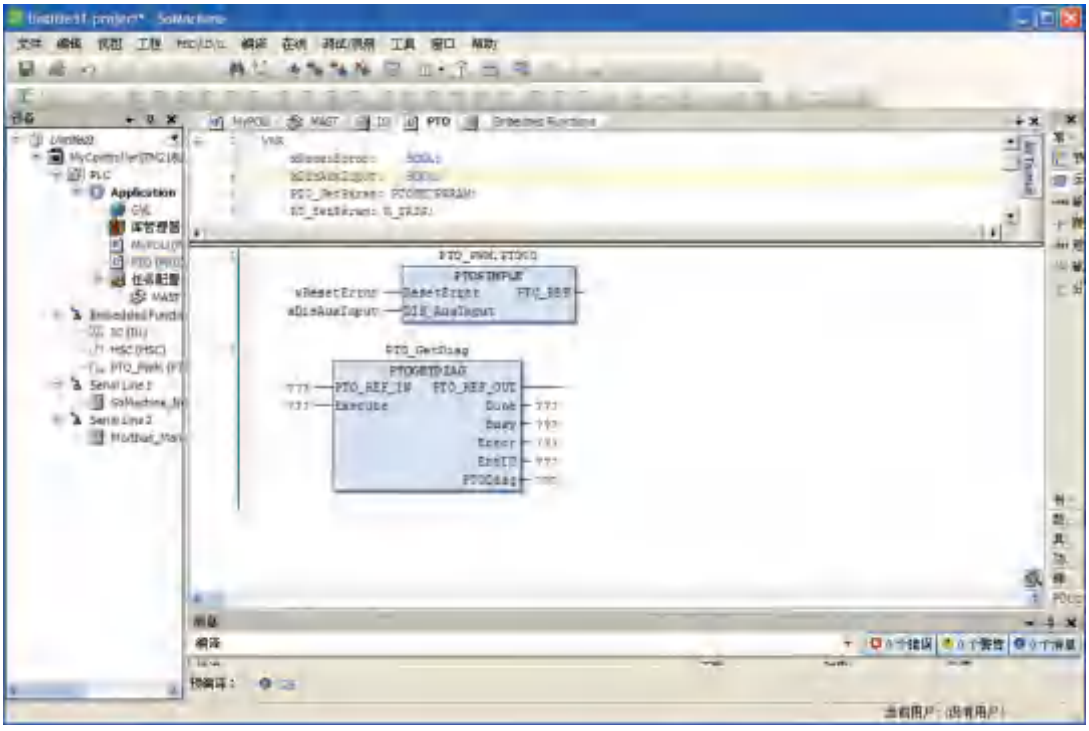
2）. 建立输入输出变量，如图：



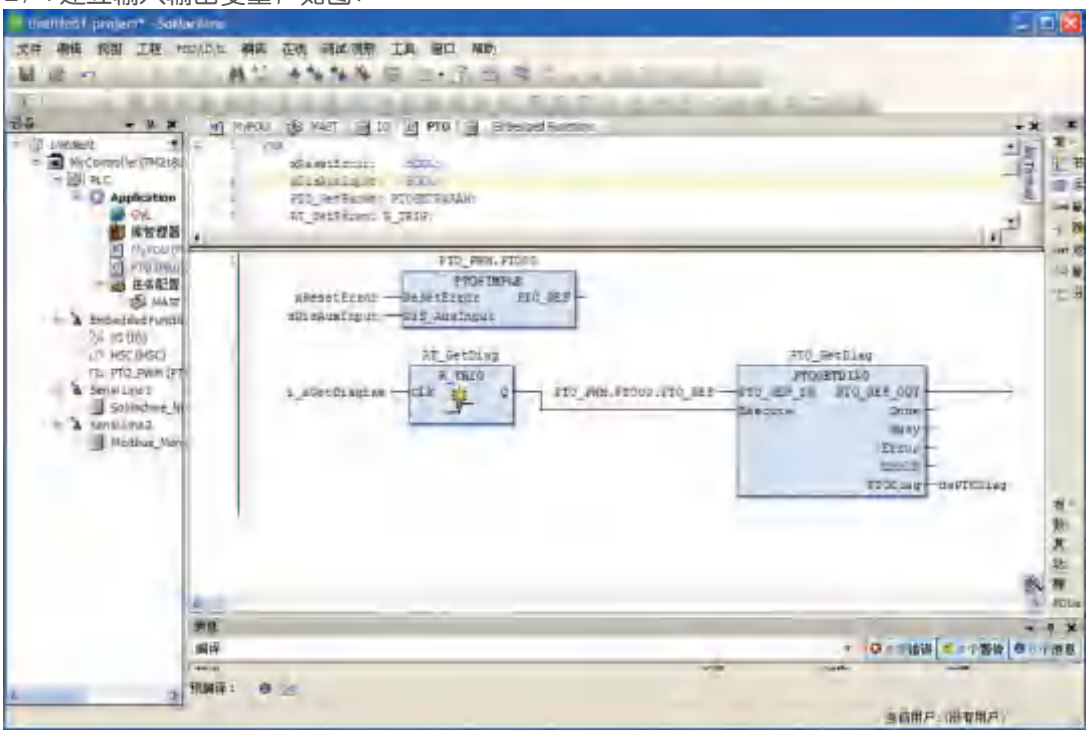
A - 变量i_xSetParamExe为功能块触发变量
B - 变量stPTOParam为PTO_PARAMETER_TYPE类型
C - 变量dwPTOParam为具体读取的数值
其中变量stPTOParam的类型与功能块PTOGETPARAM中使用的相同

10．PTO诊断功能块
1）. 从PTO库中选取PTOGetDiag功能块，如图：





2）. 建立输入输出变量，如图：



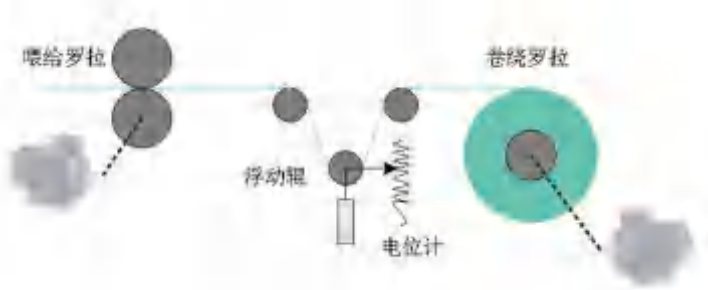
A - i_xGetDiagExe为功能块触发变量
B - 变量dwPTODiag为诊断变量，其具体含义如下：

位	定义		位	定义
0...3	未使用		18	检测到软件上限
4	检测到内部错误		19	检测到软件下限
5,6	未使用		20	没有为FastPTO分配触发引脚
7	检测到配置错误		21	检测到回归错误
8	未使用		22	频率无效
9	检测到近似限制错误		23	加速度无效
10	命令缓冲区已满		24	减速度无效
11	检测到缓冲区速度错误		25	命令被拒绝
12	轴未被引用		26	距离无效
13	回归近似被禁用		27	位置无效
14	FastPTO停止例外		28	回归模式无效
15	FastPTO重新配置		29	方向无效
16	FastPTO过量		30	反向
17	驱动器未就绪（辅助输入DriveReady为False）		31	检测到配置文件错误

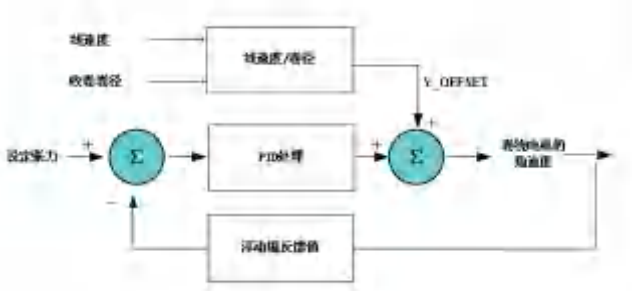
PID示例

概述

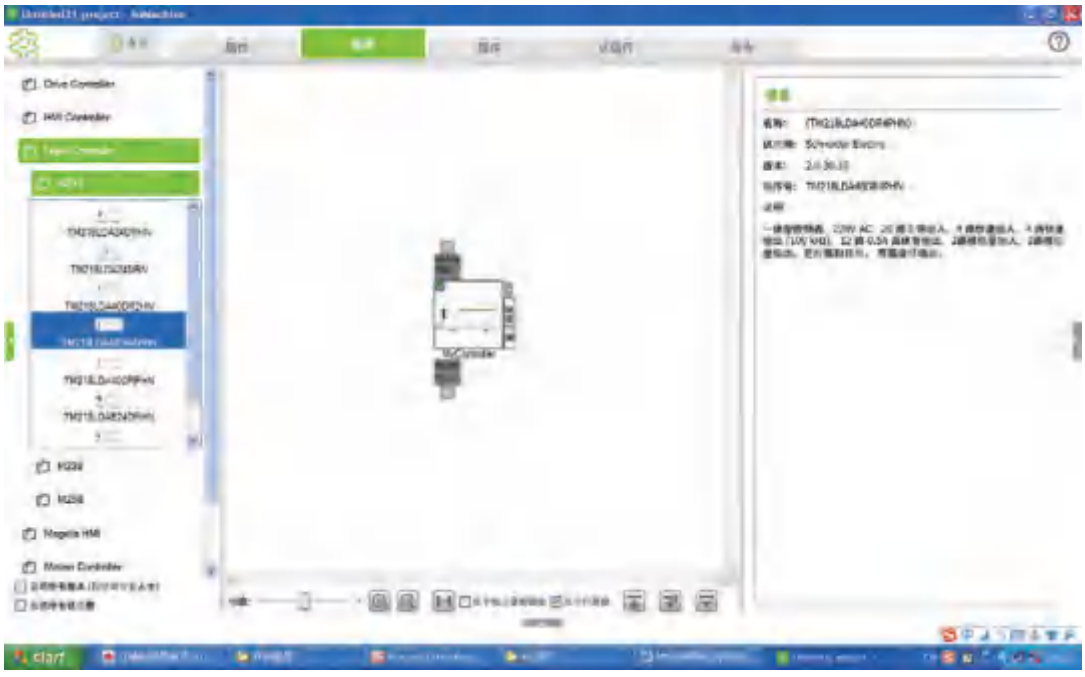
本例描述了用Somachine中的PID调节器实现PID速度控制功能，控制要求如下：通过设定的材料张力值与浮动辊所在位置所对应的张力值进行PID调节保持在不同的线速度及不同的卷径时实现卷绕恒张力控制。



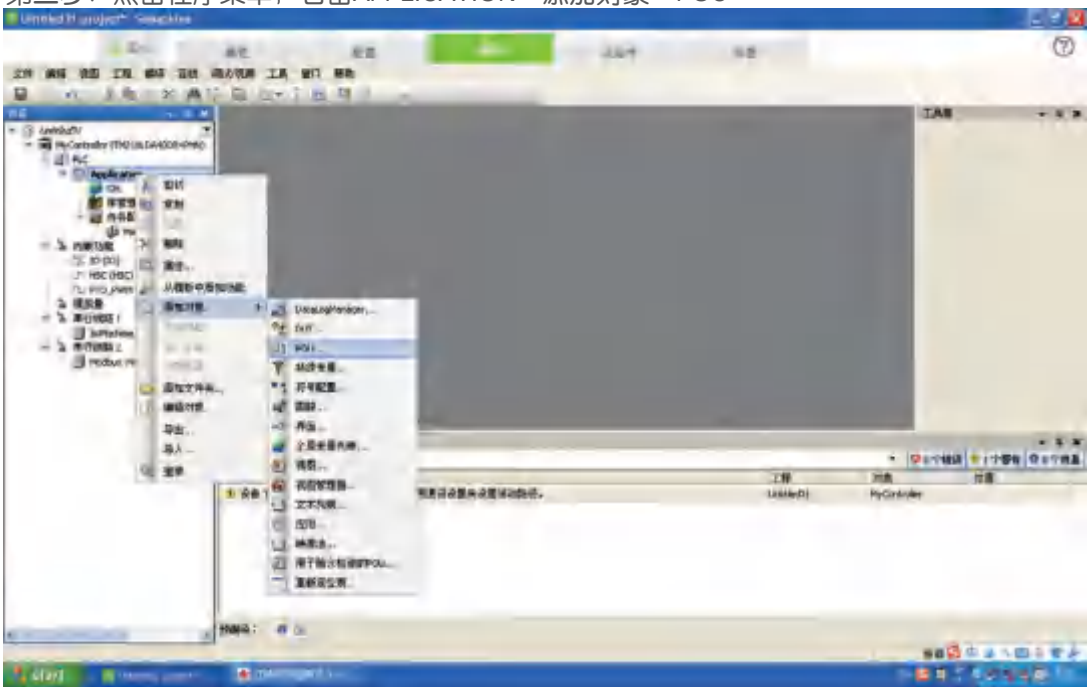
在本例中材料的线速度由喂给罗拉给定，保持恒线速度，卷绕罗拉的角速度由两部分组成通过卷径和运行线速度所产生的角速度+浮动辊PID计算出的角速度误差补偿（用来消除加减速及卷径计算所带来的误差），这里的电位计可以通过浮动辊的角度或高度计算得到实际的张力值。控制框图如下：



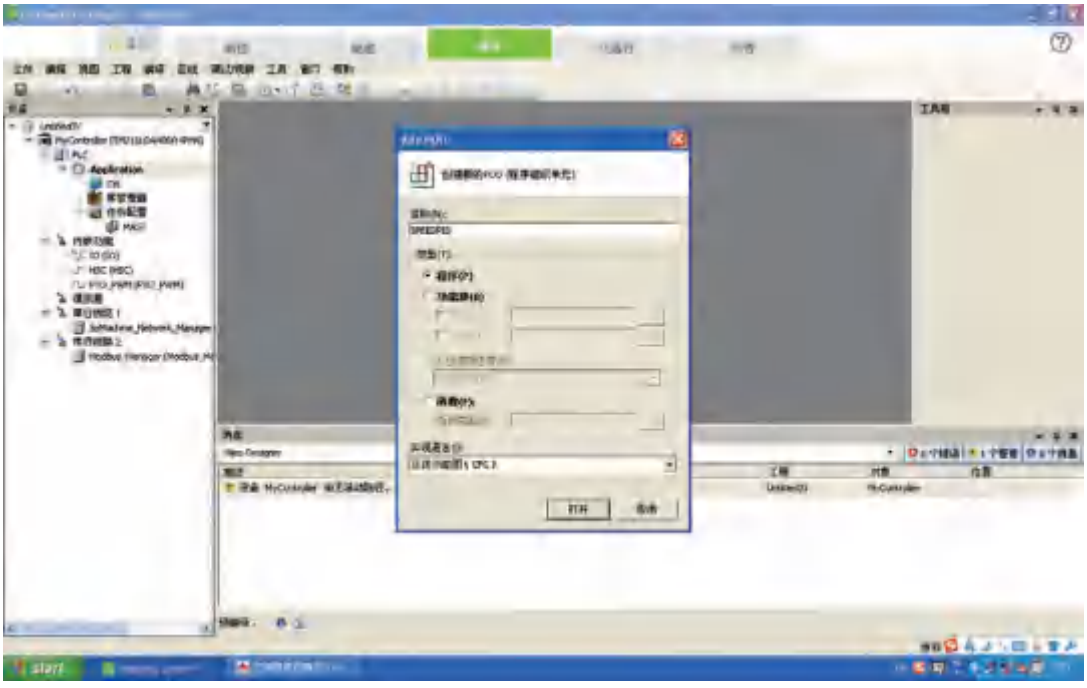
实际例程如下：
1首先在Somachine中建立相应的硬件控制器项目；



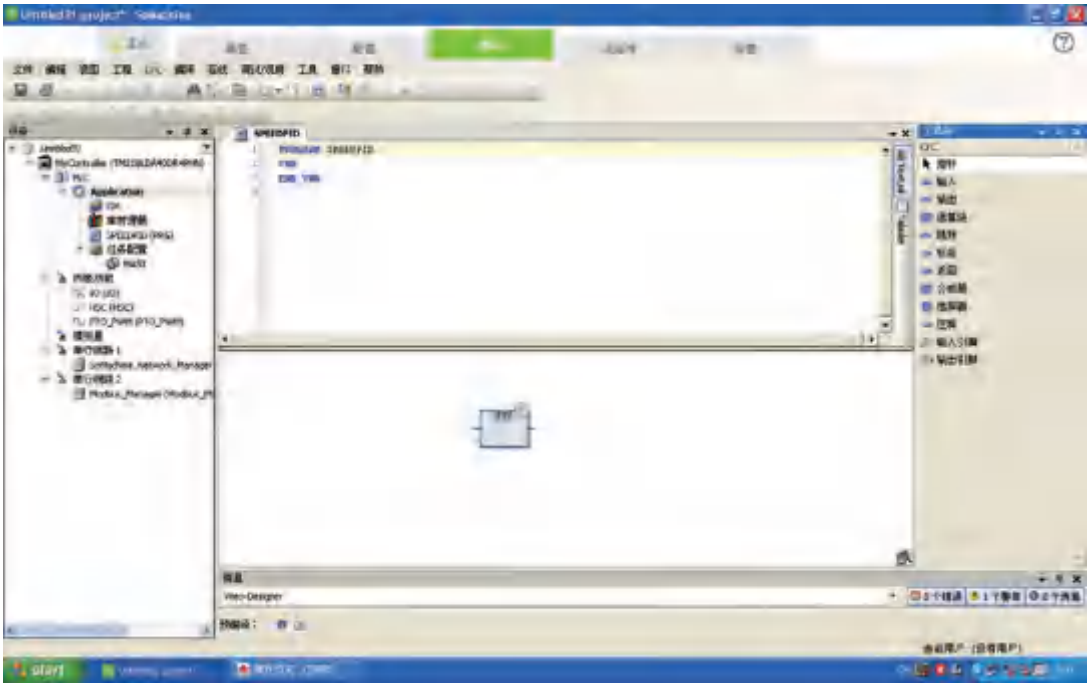
第二步：点击程序菜单，右击APPLICATION→添加对象→POU



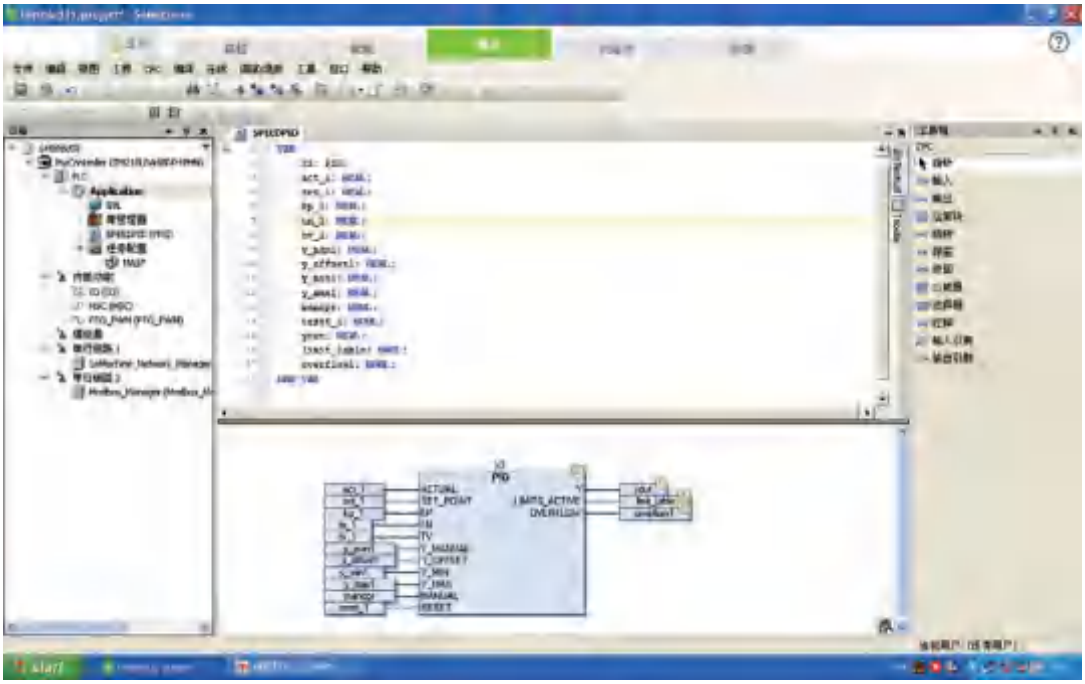
第三步：将POU重命名为SPEEDPID（也可以不改）,选择程序，在实现语言下拉菜单中选择熟悉编程语言指令如CFC,然后点击确定按钮



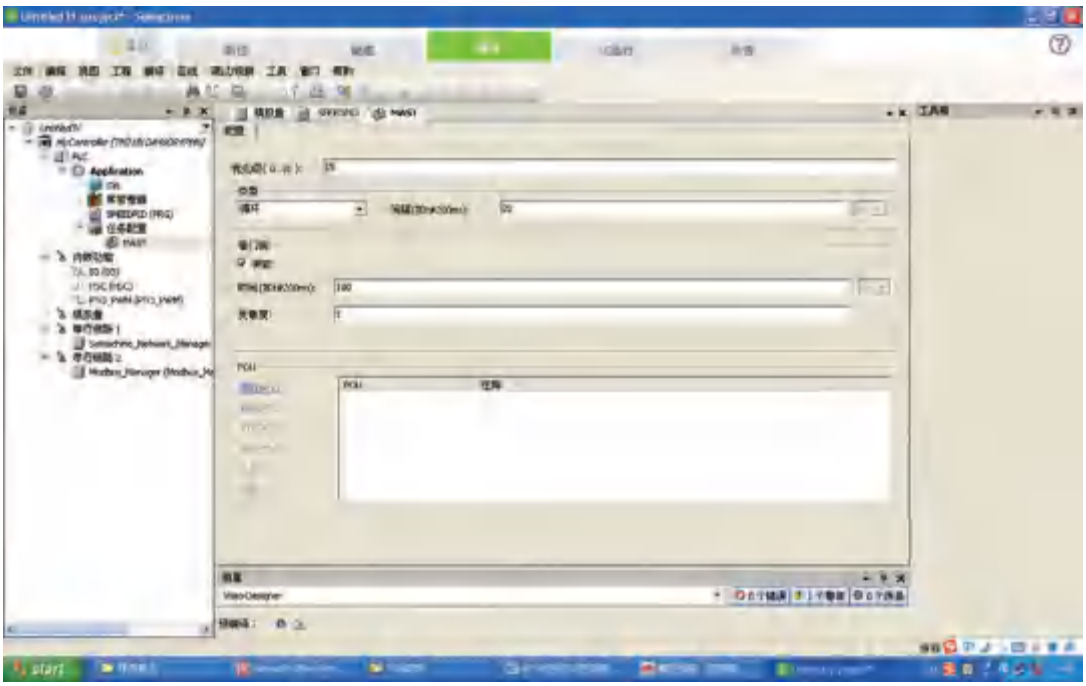
第四步：在图示的编程窗口中点击运算器添加功能块，并在？？？处点击后输入PID后点enter



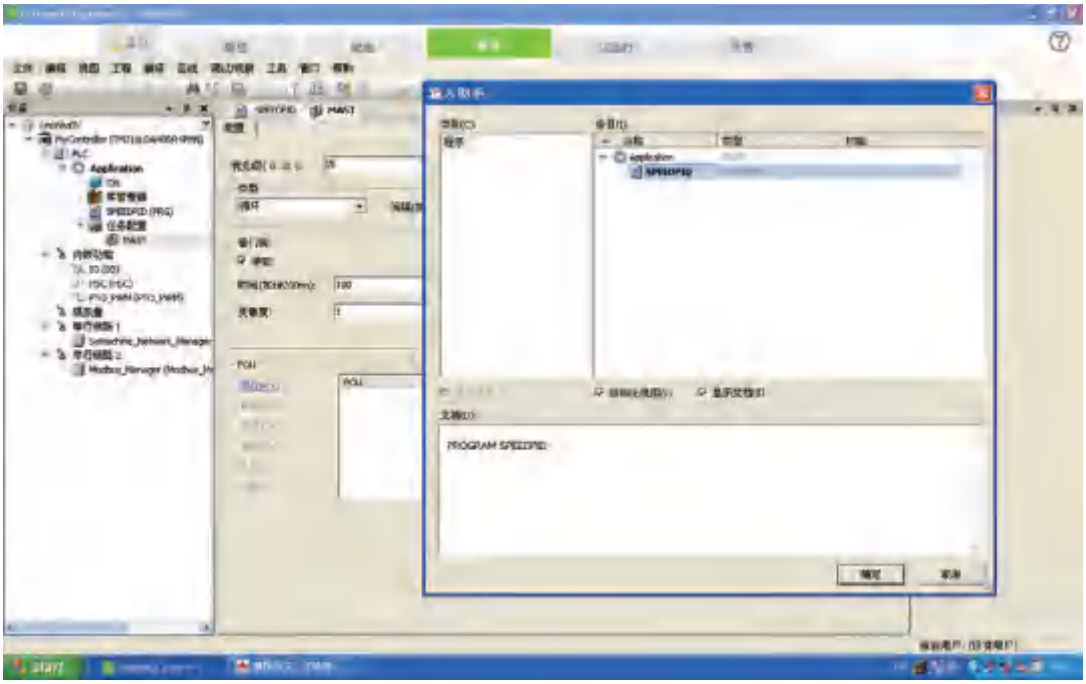
第五步：命名相应的参数和功能块名称完成后，下图所示相应的变量会出现在编程窗口的变量定义窗口中。



第六步：点击任务配置下的MAST任务菜单设置相应的任务参数，需要尤其注意好扫描周期的设置，会影响相应的积分和微分效果，尤其在PID_FIXCYCLE功能块时。设置的时间越短，PID处理的周期越快。这样对要求实时响应快的应用场合有好处。但是对于响应有较大滞后的场合会产生超调。并且过短的扫描时间会导致自由运行的任务周期变长，影响其他功能的执行速度。



第七步：然后在该任务中将相应的程序段中添加到当前任务中；右击添加POU,在输入助手选择相应的程序段SPEEDPID。



第八步：程序编译仿真或下载；然后进行PID控制所涉及到的参数进行设定一定的值

act_1: REAL;由对应浮动辊的电位计所产生的张力，通过模拟量转化出来的实际反馈张力；在此我们主要讨论PID控制，不再表述如何将模拟量转化成实际张力；

set_1: REAL;该参数由用户输入工艺要求的张力；

kp_1: REAL;PID的比例增益，该参数对控制要求响应速度有较大的影响，数值大时可以加快调节速度，但是会产生超调震荡，过小会导致响应滞后长时间出现偏差；

tn_1: REAL;PID的积分时间，单位为秒，该参数越大，积分效果越明显，但是较小的该参数会引起调节器的震荡。

tv_1: REAL;PID的微分时间，对于响应比较及时的控制调节器，建议取消该项调节，将该参数值输入为零即可，因为微分调节会对突然间的干扰起到放大作用，引起震荡，对于响应比较滞后的调节控制可以引入该项调节能够预见调节趋势，达到快速调节的目的；

y_man1: REAL;手动操作时调节器的输出值；

y_offset1: REAL;该参数比较重要，由用户设定操作的参考值，在该实例中即由线速度与卷径计算得来的，这样可以在接近目标值的附近进行PID微调，可以较快的实现无大幅超调达到目标控制值。同时该值可以随卷径的变化或线速度的变化进行实时调整。

y_min1: REAL;此参数对于PID的控制也会起到一定的调节作用,主要用来防止当误差较大持续时间较长时或PID参数不太合适时设置出现调节器输出值太小或太大而导致的误动作或对实施设备造成的损坏等情况。具有一定保护作用；一旦到达极限值就会停止响应的积分工作从而防止积分项超调引起震荡。

y_max1: REAL;功能同Y_MIN；

manopr: BOOL;手动操作使能位；

reset_1: BOOL;此参数具有复位功能，功能包括有修改KP，TN，TV参数生效、复位积分项为零、停止PID调节功能，例如当机器停止时可以将该为置为TRUE,机器运行启动后再复位该位将PID投入运行调节目标控制值；

yout: REAL;PID调节器的输出；

limit_lable: BOOL;当限位出现报警指示标志；可以此位来提醒调试人员或现场操作人员是否PID参数设置是否合适或反馈传感器是否故障等信息；一般设备或参数设置正常时不会出现此警示标志现象；

overflow_1: BOOL;用来指示积分项浮点数调节超范围。一旦出现此现象必须使用复位端子复位才能再次使用该功能块；

第九步：PID参数赋值计算

假如 设定张力=10KG,

线速度=50M/MIN,

收卷周长=100MM,

当前实际张力=8KG,

KP=0.1,

TN=100S,

TV=0（PI控制），

PID的执行周期为20MS

则Y_OFFSET=线速度/周长=50/0.1=500RPM,

在第一个扫描周期，比例项输出=KP*E=0.1*(10-8)=0.2

积分项输出=KP*E*T/TN=0.1*(10-8)*0.02/100=0.00004

YOUT=500+0.2+0.00004=500.20004

第二个扫描周期，假定实际张力=8.1kg

比例项输出=KP*E=0.1*(10-8.1)=0.19

积分项输出=KP*E*T/TN=0.1*(10-8.1)*0.02/100+0.00004≈0.000078

YOUT=500+0.19+0.000078=500.190078

如果实际的张力达到目标值时间较长，并且没有超调的话，可以适当的增加KP如KP=0.15,0.2,0.25,0.3，当然也可以减少积分时间如TN=90,70，80,60,50，当然设定KP的时候也要因情况而定，比如当设定值10KG对于5000，当前值8KG则为4000，偏差值为1000，此时的KP就要小一点如0.0001，或者在输出值后再除以1000.

当使用微分控制时要特别当心，设定的积分时间要求尽量的小，防止当有扰动误差时引起PID控制不稳定现象。

例如设定TV=0.001s,假设上面两个扫描周期张力变化0.1，

则微分项输出=KP*TV*△E/T=0.1*0.001*0.1/0.02=0.0005

当微分时间为10S时微分项输出= KP*TV*△E/T=0.1*10*0.1/0.02=5，即当张力出现0.1KG的波动时就会产生较大调节量，如果张力波动较大时，就会产生较大的速度波动引起震荡。

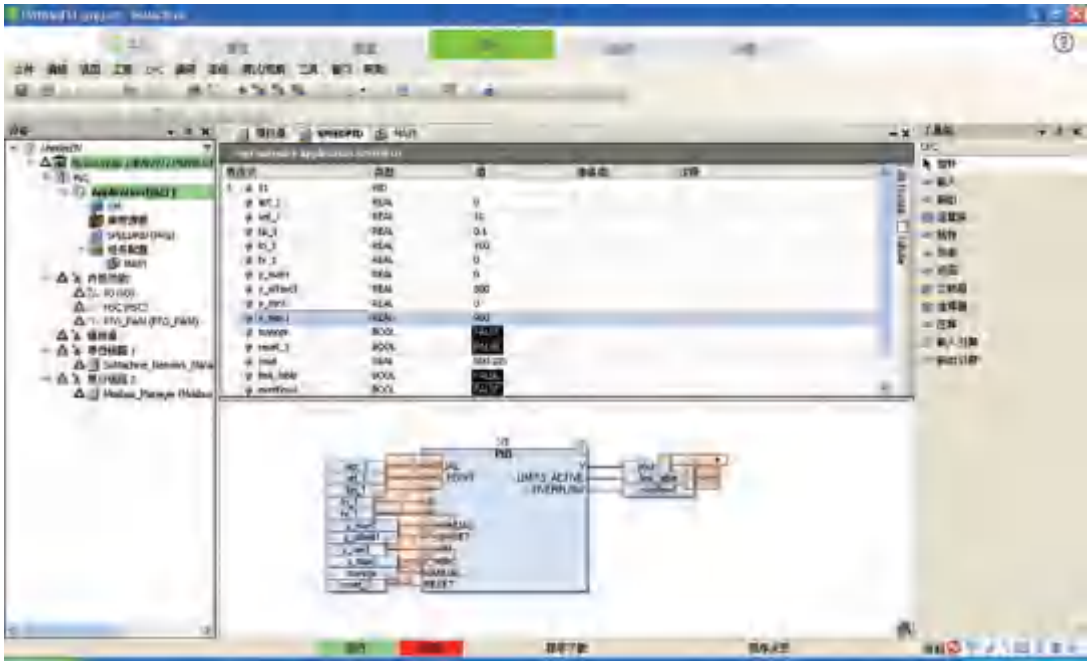
针对Y_MIN,Y_MAX参数如何设定可以参考此例

如最大线速度为80m/min,最小的收卷卷径周长为100mm,最大收卷卷径周长为600mm

则可以设定Y_MAX=80/100*110=880RPM≈900

Y_MIN=80/600*90=12RPM≈0

仿真程序如下：



RTC实现示例
概述
本章提供演示如何获取或设置 Real Time Clock 的实现示例。

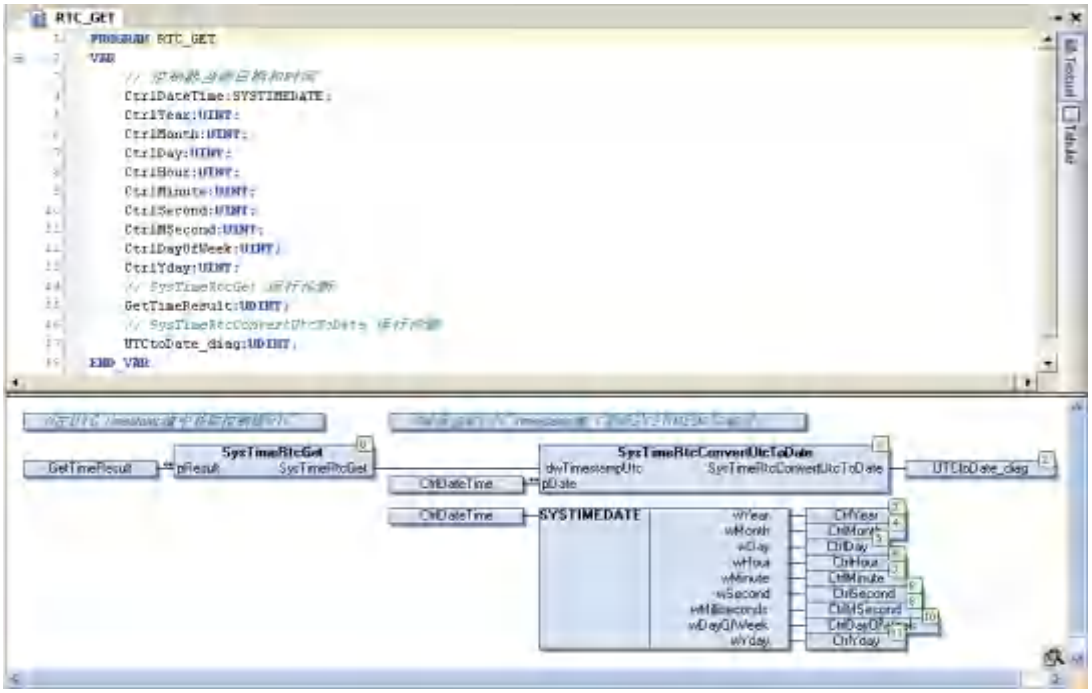
本章包含了哪些内容
本章包含了以下主题：
1.获取控制器日期和时间
2.设置控制器日期和时间
3.获取/ 设置 RTC SoMachine 项目示例

获取控制器日期和时间
概述
借助此程序示例，可以获取控制器日期和时间。

程序
变量声明:

```
VAR
// 控制器当前日期和时间
CtrlDateTime:SYSTIMEDATE;
CtrlYear:UINT;
CtrlMonth:UINT;
CtrlDay:UINT;
CtrlHour:UINT;
CtrlMinute:UINT;
CtrlSecond:UINT;
CtrlMSecond:UINT;
CtrlDayOfWeek:UINT;
CtrlYday:UINT;
// SysTimeRtcGet 运行诊断
GetTimeResult:UDINT;
// SysTimeRtcConvertUtcToDate 运行诊断
UTCtoDate_diag:UDINT;
END_VAR
```

POU程序:



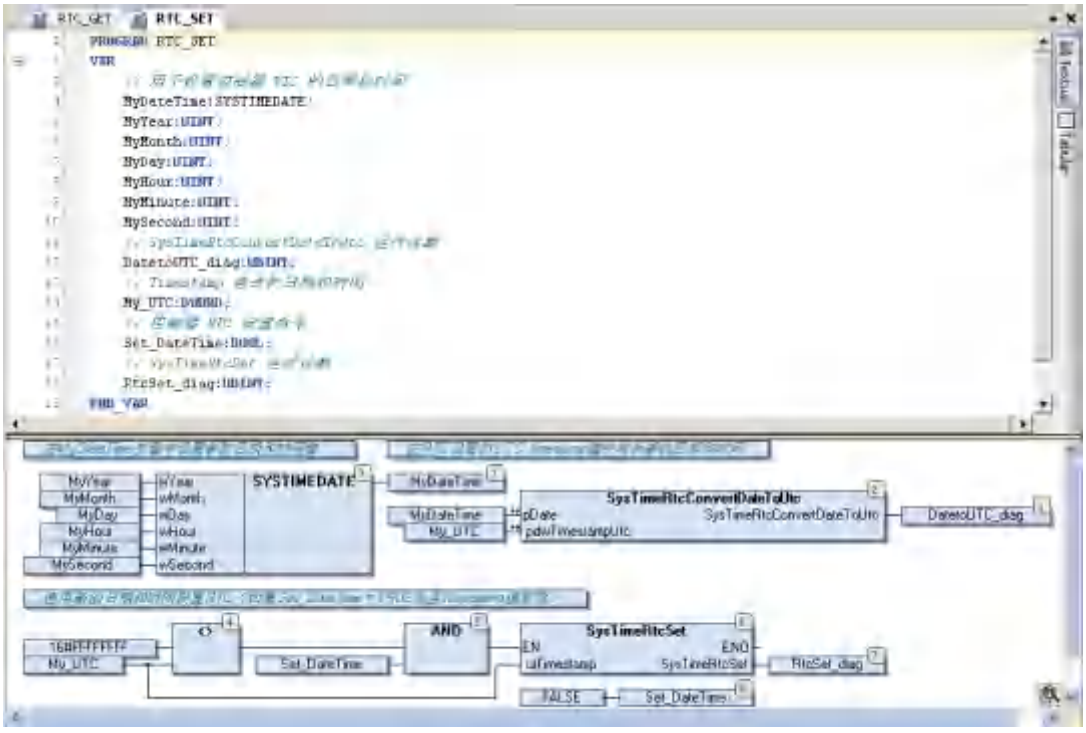
设置控制器日期和时间

概述
借助此程序示例，可以通过用户日期和时间设置控制器 Real Time Clock。

程序
变量声明:

```
VAR
// 用于设置控制器 RTC 的日期和时间
MyDateTime:SYSTIMEDATE;
MyYear:UINT;
MyMonth:UINT;
MyDay:UINT;
MyHour:UINT;
MyMinute:UINT;
MySecond:UINT;
// SysTimeRtcConvertDateToUtc 运行诊断
DatetoUTC_diag:UDINT;
// Timestamp 格式的日期和时间
My_UTC:DWORD;
// 控制器 RTC 设置命令
Set_DateTime:BOOL;
// SysTimeRtcSet 运行诊断
RtcSet_diag:UDINT;
END_VAR
```

POU程序:



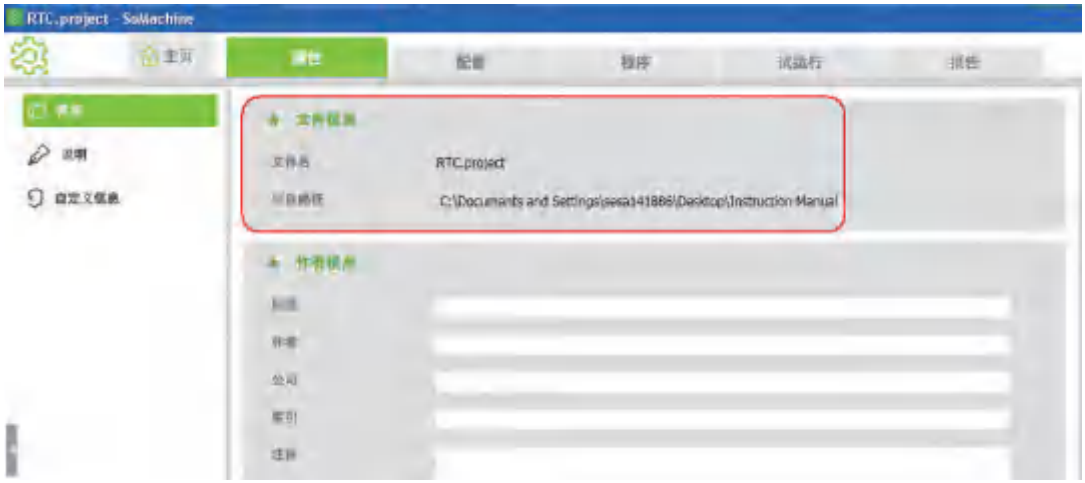
获取/ 设置 RTC SoMachine 项目示例

概述
借助此程序示例，可以按照SoMachine操作步骤实现RTC的读取和设置。

1．RTC SoMachine 项目RTC读取示例步骤（以M258为例）：

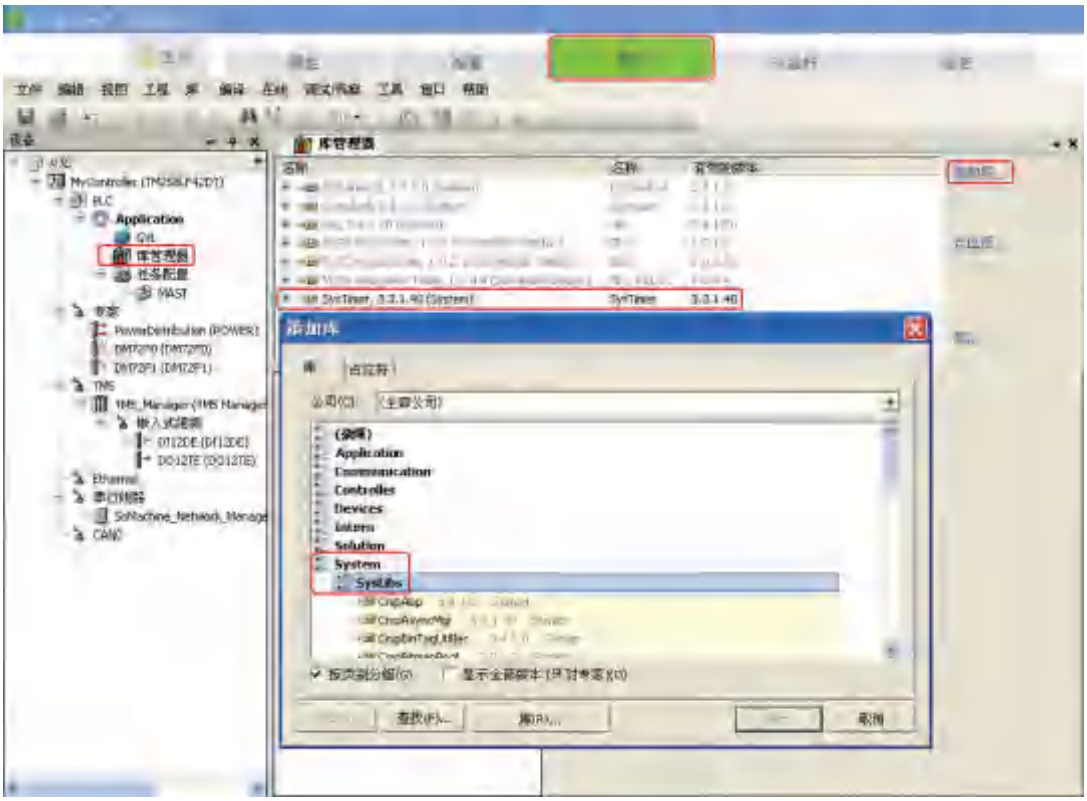
步骤1:使用空项目启动创建一个项目工程

步骤2：创建一个文件名为RTC的项目工程

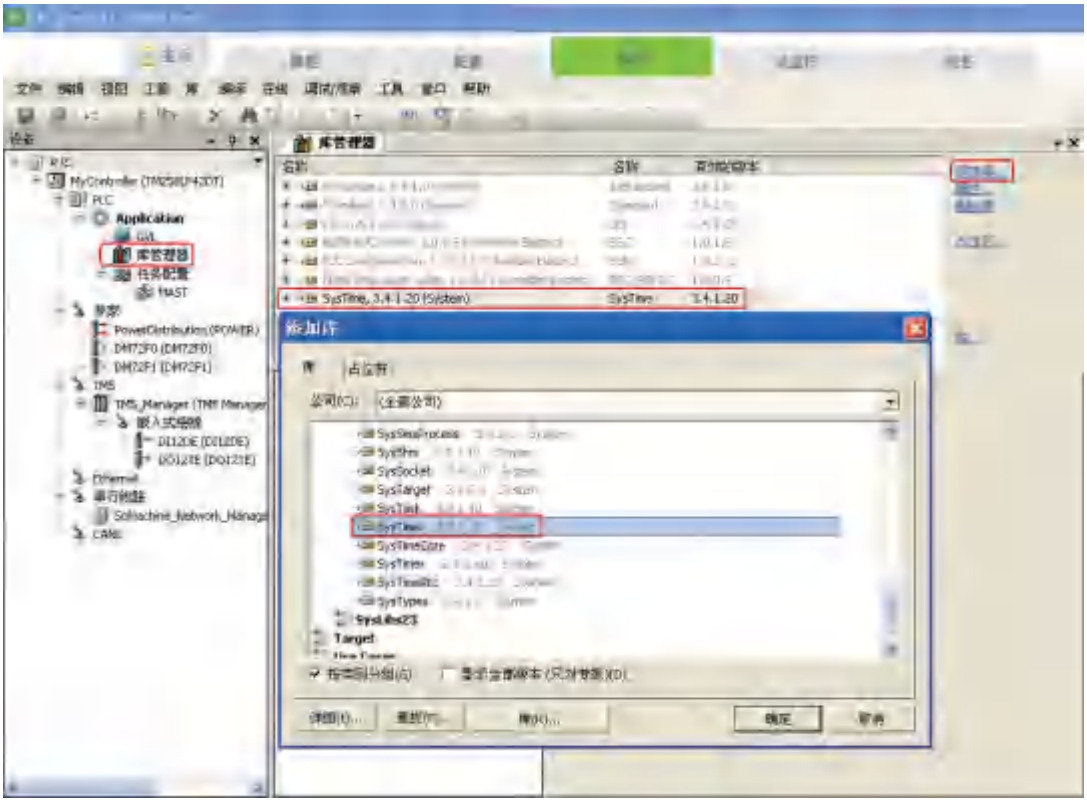


步骤3：在配置中添加一个M258LF42DT

步骤4：在程序的库管理器中添加一个SysTime库

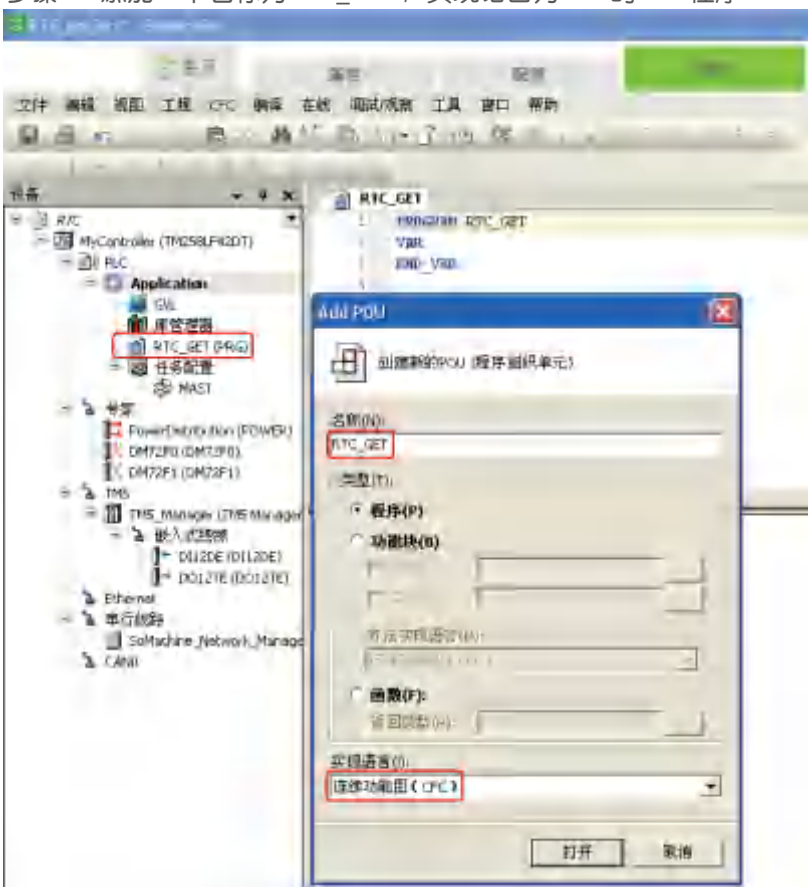


步骤5：在库管理器的添加库中找到System - SsyLibs - SysTime库，单击确定添加该库

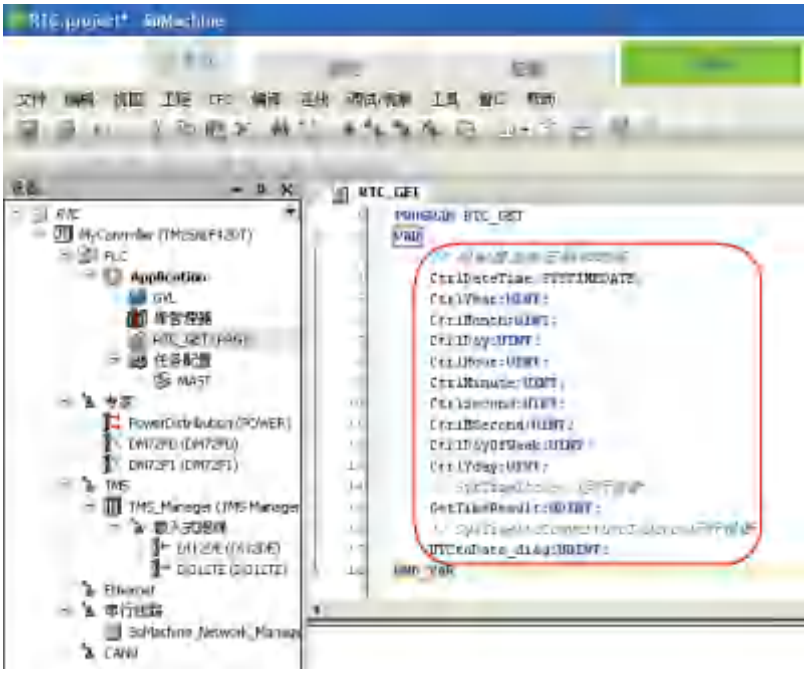


步骤6：右键点击Application，在添加对象中添加一个POU程序

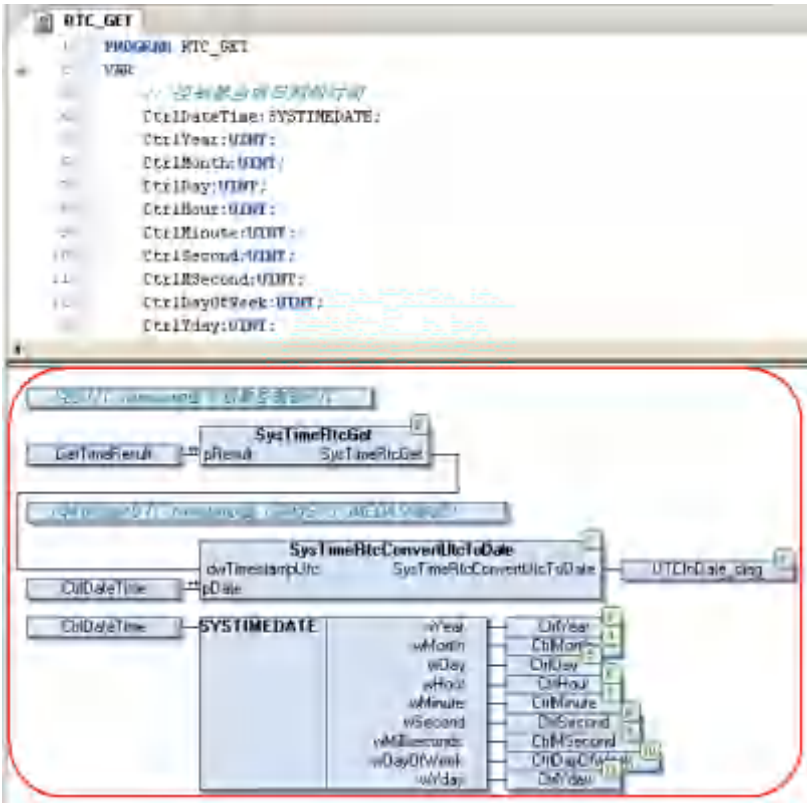
步骤7：添加一个名称为RTC_GET，实现语言为CFC的POU程序



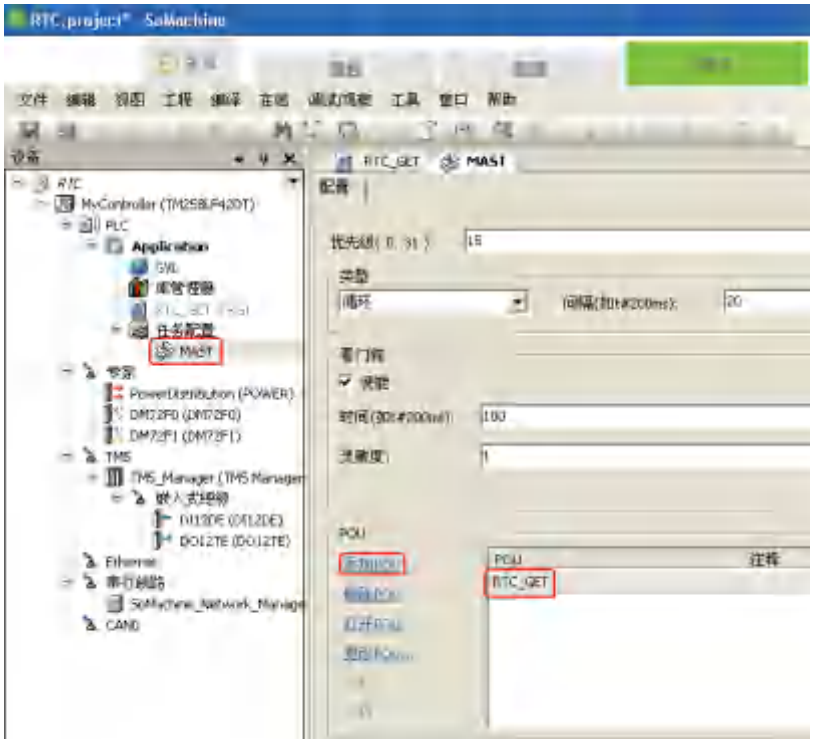
步骤8：在程序RTC_GET中添加变量



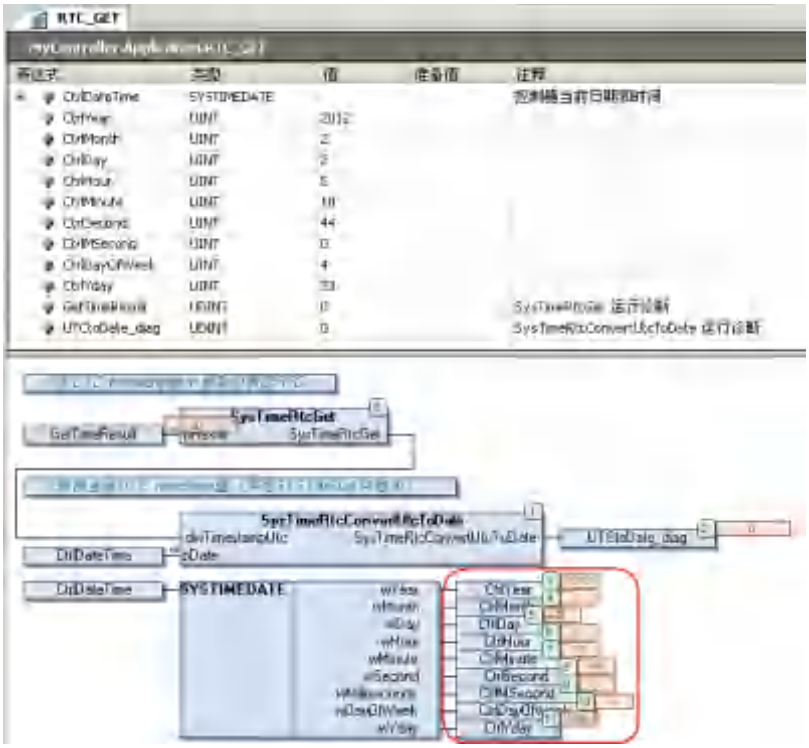
步骤9：在程序RTC_GET中添加程序



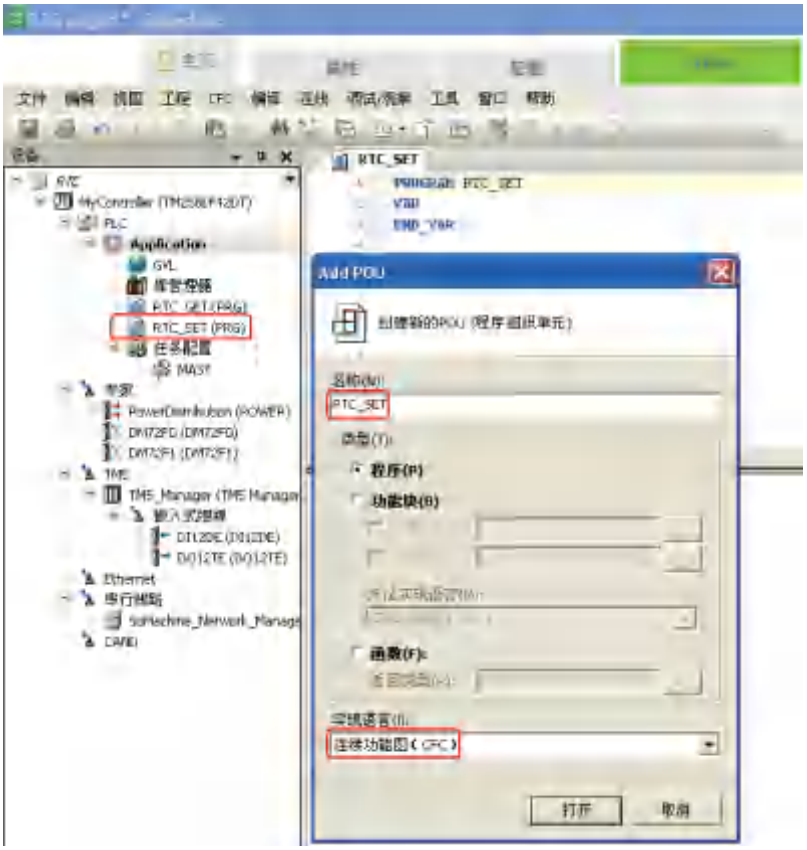
步骤10：在任务配置的MAST任务中添加RTC_GET程序POU



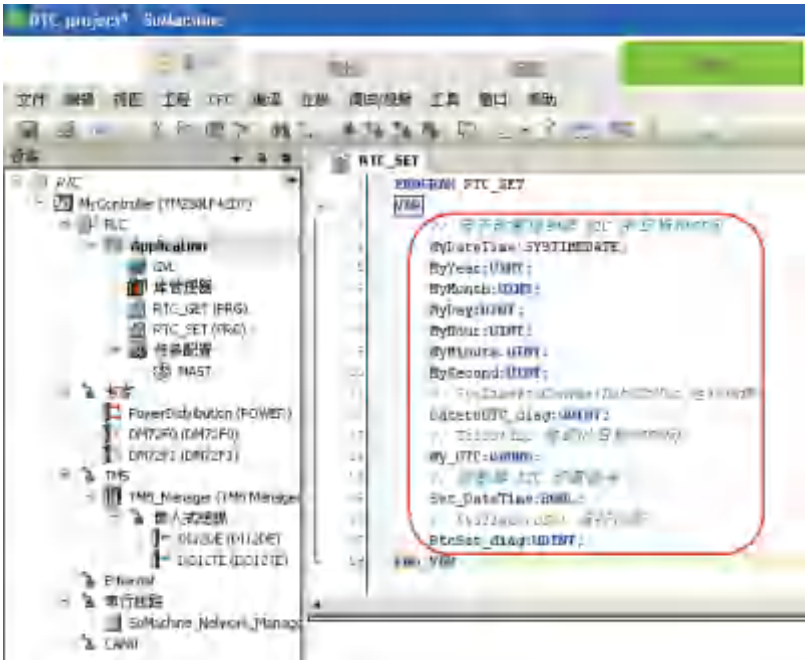
步骤11：编译下载运行程序



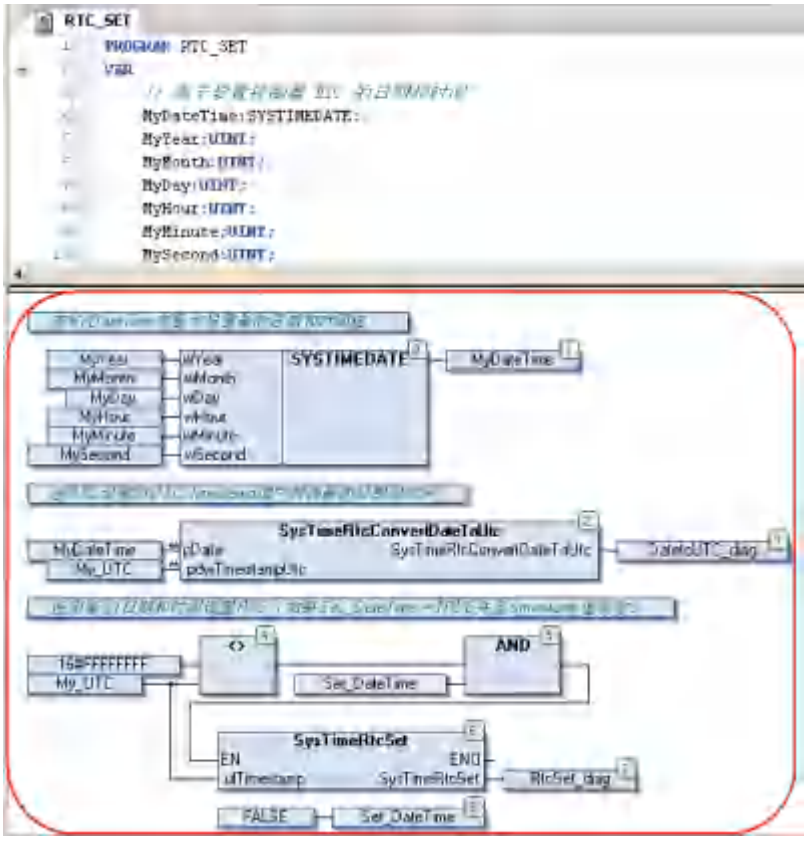
2．RTC SoMachine 项目RTC设置示例步骤（以M258为例）：
步骤1~ 步骤6：同上
步骤7：添加一个名称为RTC_SET，实现语言为CFC的POU程序



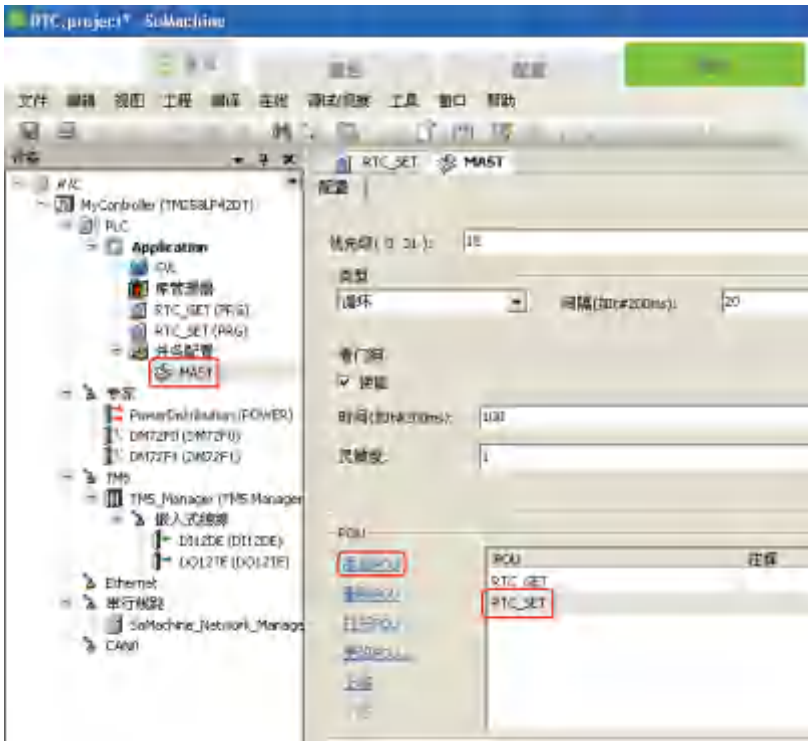
步骤8：在程序RTC_GET中添加变量



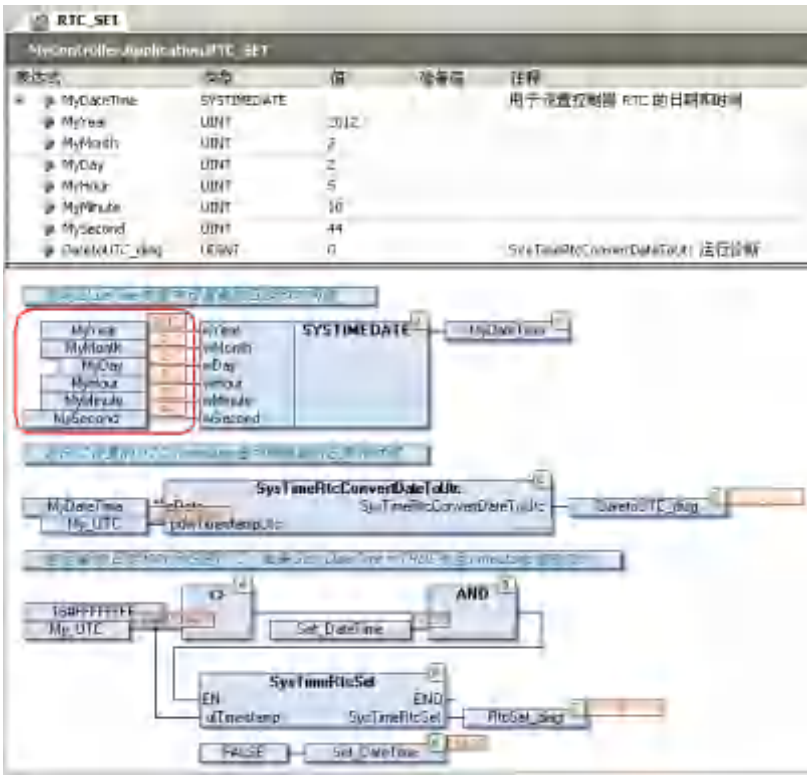
步骤9：在程序RTC_GET中添加程序



步骤10：在任务配置的MAST任务中添加RTC_SET程序POU



步骤11：编译下载运行程序



ST语言高级语言指引
1、ST 编程语言的各种元素
使用结构化文本 (ST) 的编程语言，可以执行多种操作，例如调用功能块、和赋值、有条件地执行指令和重复任务。(ST) 的编程语言由各种元素组成，具体如下。
表达式：表达式是由操作符和操作数组成的结构，在执行表达式时会返回值。
操作数：操作数表示变量，数值，地址，功能块等。
操作符：操作符是执行运算过程中所用的符号。
指令：指令用于将表达式返回的值赋给实际参数，并构造和控制表达式。

5.2.2. 表达式
计算表达式时将根据操作符的优先级所定义的顺序将操作符应用于操作数表。首先执行表达式中具有最高优先级的操作符，接着执行具有次优先级的操作符，依此类推，直到完成整个计算过程。优先级相同的操作符将根据它们在表达式中的书写顺序从左至右执行。可使用括号更改此顺序。
例如为如果 A、B、 C 和 D 的值分别为 1、 2、 3 和 4，并按以下方式计算：A+B-C*D，结果则为 -9。(A+B-C)*D，结果则为 0。
如果操作符包含两个操作数，则先执行左边的操作数，例如在表达式SIN(A)*COS(B)中，先计算表达式 SIN(A)，后计算 COS(B)，然后计算它们的乘积。

5.2.3. 操作数
操作数可以是：地址，数值，变量，多元素变量，多元素变量的元素，功能调用， 功能块输出等。处理操作数的指令中的数据类型必须相同。如果需要处理不同类型的操作数，则必须预先执行类型转换。
在下面的示例中，整数变量 i1 在与实数变量 r4 相加之前会先转换为实数变量。
r3 := r4 + INT_TO_REAL(i1) ;

5.2.4. 操作符
操作符是一种符号，它表示：
要执行的算术运算，或要执行的逻辑运算，功能编辑调用。
操作符是泛型的，即它们自动适应操作数的数据类型。
例如：实数变量r1、r2、r3执行以下操作r1:=r2+r3;这里操作符+为实数加法；而整数变量i1、i2、i3执
行i1:=i2+i3;
在这里加法会自动适应为整数加法。

5.2.5. ST 语言的操作符
1、括号（）
优先级：1（最高）
适用的操作数：表达式
描述：括号用于改变操作符的执行顺序。例如A、B、C、D分别赋值为1、2、3、4，A+B-C*D结果为
-9，而
(A+B-C)*D结果为0。

2、功能调用FUNCTIONNAME（实际参数）
优先级：2
适用的操作数：所有数据类型
描述：功能调用用于执行该功能。

3、取反-
优先级：3
适用的操作数：INT、DINT、REAL
描述：取反时操作数的数值会反转。IN1为4，则执行OUT1:=-IN1;OUT1为-4。

4、反码NOT
优先级：3
适用的操作数：BOOL、BYTE、WORD、DWORD
描述：执行反码操作时，操作数会被逐位反转。IN1为011100，则执行OUT1:=NOT IN1;OUT1为
100011。

5、幂**
优先级：4
适用的操作数：底数为REAL，指数为INT、DINT、UINT、UDINT、REAL
描述：将第一个操作数为底数，第二个操作数为指数，进行就幂运算。例如，IN1为5.0，IN2为4.0，则
OUT1：
=IN1**IN2；则OUT1为625.0。

6、乘法*
优先级：5
适用的操作数：INT、DINT、UINT、UDINT、REAL
描述：将第一个操作数值乘上第二个操作数值。

7、除法/
优先级：5
适用的操作数：INT、DINT、UINT、UDINT、REAL
描述：将第一个操作数值除以第二个操作数值。

8、模数MOD
优先级：5
适用的操作数：INT、DINT、UINT、UDINT
描述：将第一个操作数值除以第二个操作数值，除法的余数显示为结果。例如INT1为7，IN2为2，则
OUT1：=IN1
MOD IN2；则OUT1为1。

9、加法+
优先级：6
适用的操作数：INT、DINT、UINT、UDINT、REAL、TIME
描述：将第一个操作数值加上第二个操作数值。

10、减法-
优先级：6
适用的操作数：INT、DINT、UINT、UDINT、REAL、TIME
描述：将第一个操作数值减去第二个操作数值。

11、小于比较<
优先级：7
适用的操作数：BOOL、BYTE、INT、DINT、UINT、UDINT、REAL、TIME、WORD、DWORD、STRING、DT、
TOD、DATE
描述：将第一个操作数值与第二个进行比较，如果第一个操作数值小于第二个操作数值则结果为1，否则为0。

12、大于比较<
优先级：7
适用的操作数：BOOL、BYTE、INT、DINT、UINT、UDINT、REAL、TIME、WORD、DWORD、STRING、DT、
TOD、DATE

描述：将第一个操作数值与第二个进行比较，如果第一个操作数值大于第二个操作数值则结果为1，否则为0。

13、小于或等于比较<=
优先级：7
适用的操作数：BOOL、BYTE、INT、DINT、UINT、UDINT、REAL、TIME、WORD、DWORD、STRING、DT、
TOD、DATE
描述：将第一个操作数值与第二个进行比较，如果第一个操作数值小于或者等于第二个操作数值则结果为1，否则为0。

14、大于或者等于比较>=
优先级：7
适用的操作数：BOOL、BYTE、INT、DINT、UINT、UDINT、REAL、TIME、WORD、DWORD、STRING、DT、
TOD、DATE
描述：将第一个操作数值与第二个进行比较，如果第一个操作数值大于或者等于第二个操作数值则结果为1，否则为
0。

15、等于=
优先级：8
适用的操作数：BOOL、BYTE、INT、DINT、UINT、UDINT、REAL、TIME、WORD、DWORD、STRING、DT、
TOD、DATE
描述：将第一个操作数值与第二个进行比较，如果第一个操作数值等于第二个操作数值则结果为1，否则为0。

16、不等于<>
优先级：8
适用的操作数：BOOL、BYTE、INT、DINT、UINT、UDINT、REAL、TIME、WORD、DWORD、STRING、DT、
TOD、DATE
描述：将第一个操作数值与第二个进行比较，如果第一个操作数值不等于第二个操作数值则结果为1，否则为0。

17、逻辑与AND
优先级：9
适用的操作数：BOOL、BYTE、WORD、DWORD
描述：操作数之间的与关联，如果操作数类型为BYTE、WOD、DWORD，则此关联是逐位进行的。例如：IN1为
1100，IN2为1010，则OUT1：=IN1 AND IN2；结果OUT1为1000。

18、逻辑异或XOR
优先级：10
适用的操作数：BOOL、BYTE、WORD、DWORD
描述：操作数之间的异或关联，如果操作数类型为BYTE、WORD、DWORD，则此关联是逐位进行的。如果两个以
上操作数进行异或运算时，当状态为1的操作数个数不是为偶数时结果为1，否则为0。

19、逻辑或OR
优先级：11
适用的操作数：BOOL、BYTE、WORD、DWORD
描述：描述：操作数之间的或关联，如果操作数类型为BYTE、WOD、DWORD，则此关联是逐位进行的。例如：
IN1为1100，IN2为1010，则OUT1：=IN1 OR IN2；结果OUT1为1110。

5.2.6. ST 指令
赋值指令
描述：执行赋值时，变量的当前值会替换为表达式的计算结果。赋值表达式的结构为：左边是变量名称，之后是赋
值操作符 :=，然后是要求值的表达式。两个变量（分别位于赋值操作符的左侧和右侧）的数据类型必须相同。数组
是个特例，可对长度不同的两个数组执行赋值操作。
例如：A：=B；

选择指令 IF...THEN...END_IF
描述：IF 指令只有确定其相关布尔表达式的值为 1（真）时，才会执行指令或一组指令。如果条件为 0（假），将不会执行该指令或指令组。THEN 指令标识条件的结尾和指令的开头。END_IF 指令标记指令的结尾。
注意： 可以嵌套任何数量的 IF...THEN...END_IF 指令，以生成复杂的选择指令。
示例 IF...THEN...END_IF
该条件可以使用布尔变量表达。
如果 FLAG 为 1，将执行指令；如果 FLAG 为 0，则不会执行。

```
IF FLAG THEN
C:=SIN(A) * COS(B) ;
B:=C - A ;
END_IF ;
```

选择指令 ELSE
描述： ELSE 指令始终出现在 IF...THEN、ELSIF...THEN 或 CASE 指令后面。
如果 ELSE 指令出现在 IF 或 ELSIF 指令后面，则仅当 IF 和 ELSIF 指令的关联布尔表达式为 0（假）时，才会执行该指令或指令组。如果 IF 或 ELSIF 指令的条件为 1（真），则不会执行该指令或指令组。
如果 ELSE 指令出现在 CASE 后面，则仅当所有标签都不包含选择器的值时，才会执行该指令或指令组。如果某个标识包含选择器的值，则不会执行该指令或指令组。
注意： 可以嵌套任何数量的 IF...THEN...ELSE...END_IF 指令，以生成复杂的选择指令。
示例 ELSE

```
IF A>B THEN
C:=SIN(A) * COS(B) ;
B:=C - A ;
ELSE
C:=A + B ;
B:=C * A ;
END_IF ;
```

选择指令 ELSIF...THEN
描述：ELSE 指令始终出现在 IF...THEN 指令后面。ELSIF 指令确定仅当 IF 指令的关联布尔表达式的值为 0（假）并且 ELSIF 指令的关联布尔表达式的值为 1（真）时，才会执行指令或指令组。如果 IF 指令的条件为 1（真）或者 ELSIF 指令的条件为 0（假），则不会执行该命令或命令组。THEN 指令标识 ELSIF 条件的结尾和指令的开头。
注意： 可以嵌套任何数量的 IF...THEN...ELSIF...THEN...END_IF 指令，以生成复杂的选择指令。
示例 ELSIF...THEN

```
IF A>B THEN
C:=SIN(A) * COS(B) ;
B:=SUB(C,A) ;
ELSIF A=B THEN
C:=ADD(A,B) ;
B:=MUL(C,A) ;
END_IF ;
```

选择指令 CASE...OF...END_CASE
描述：CASE 指令包含一个 INT 数据类型的表达式（选择器）和一个指令组列表。每组都具有一个包含一个或多个整数（INT、DINT、UINT 或 UDINT）或整数值范围的标签。将执行的指令为其标签中包含选择器计算出的值的第一组指令。否则，将不执行任何标签对应的指令。
OF 指令指示标签的开头。所有标签都不包含选择器的值时，才会在 CASE 指令内执行 ELSE 指令。 END_CASE 指令标记指令的结尾。
举例：

```
CASE INT1 OF
1, 5: BOOL1 := TRUE;
 BOOL3 := FALSE;
2: BOOL2 := FALSE;
 BOOL3 := TRUE;
10..20: BOOL1 := TRUE;
 BOOL3:= TRUE;
ELSE
 BOOL1 := NOT BOOL1;
 BOOL2 := BOOL1 OR BOOL2;
END_CASE;
```

循环指令 FOR...TO...BY...DO...END_FOR
描述 FOR 指令用于在发生次数可预先确定的情况下。否则可使用 WHILE 或 REPEAT。
FOR 指令会重复执行指令序列，直到遇到 END_FOR 指令为止。发生次数由起始值、结束值和控制变量决定。
控制变量、起始值和结束值必须具有相同的数据类型（DINT 或INT）。控制变量、起始值和结束值可由重复指令进行更改。这是对 IEC 61131-3 的补充。FOR 指令以控制变量值为步幅递增起始值，直到达到结束值。增量值的缺省值为 1。如果要使用其他值，则可以指定显式增量值（变量或常量）。每个新的循环之前都要检查控制变量值。如果它位于起始值和结束值的范围之外，则将离开循环。
首次运行循环之前，会进行检查以确定从初始值开始的控制变量递增是否是朝着结束值的方向。如果不是（例如，起始值 j 结束值并且增量为负值），则不会对循环进行处理。
控制变量值不是在循环外定义的。DO 指令标识重复定义的结尾和指令的开头。可以使用 EXIT 提前终止循环。END_FOR 指令标记指令的结尾。
举例：

```
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的值为32。

重复指令 WHILE...DO...END_WHILE
说明： WHILE 指令可使一个指令序列重复执行，直到其相关布尔表达式为 0（假）。
如果从一开始该表达式就为假，则根本不会执行该指令组。
DO 指令标识重复定义的结尾和指令的开头。可以使用 EXIT 提前终止循环。
END_WHILE 指令标记指令的结尾。
下列情况下不应使用 WHILE，因为它可能导致无限循环，从而造成程序崩溃：
• WHILE 不能用于过程之间的同步，例如，不能用作具有外部定义的结束条件的"等待循环"。
• WHILE 不能用在算法中，因为无法确保完成循环结束条件或执行 EXIT 指令。
示例 WHILE...DO...END_WHILE

```
x:=1;
WHILE x<100
        DO x:=X+1;
END_WHILE
```

重复指令 REPEAT...UNTIL...END_REPEAT
描述： REPEAT 指令可使一个指令序列重复执行（至少执行一次），直到相关布尔条件为 1（真）。UNTIL 指令标记结束条件。可以使用 EXIT 提前终止循环。END_REPEAT 指令标记指令的结尾。
下列情况下不应使用 REPEAT，因为它可能导致无限循环，从而造成程序崩溃：
• REPEAT 不能用于过程之间的同步，例如，不能用作具有外部定义的结束条件的"等待循环"。
• REPEAT 不能用在算法中，因为无法确保完成循环结束条件或执行 EXIT 指令。
示例 REPEAT...UNTIL...END_REPEAT

```
x := -1
REPEAT
x := x + 2
UNTIL x >= 101
END_REPEAT ;
```

退出指令 EXIT
描述：EXIT 指令用于在满足结束条件前终止重复指令（FOR、WHILE 或 REPEAT）。如果 EXIT 指令位于嵌套的重复指令内，则会离开最里面的循环（EXIT 所在的循环）。接下来，将执行循环结尾（END_FOR、END_WHILE 或 END_REPEAT）后的第一个指令。
示例 EXIT
如果 FLAG 的值为 0，执行指令后 SUM 将为 15。
如果 FLAG 的值为 1，执行指令后 SUM 将为 6。

```
SUM : = 0 ;
FOR I := 1 TO 3 DO
FOR J := 1 TO 2 DO
IF FLAG=1 THEN EXIT;
END_IF ;
SUM := SUM + J ;
END_FOR ;
SUM := SUM + I ;
END_FOR
```

施耐德电气(中国)有限公司

施耐德电气(中国)有限公司	北京市朝阳区望京东路6号施耐德电气大厦	邮编：100102	电话：(010) 84346699	传真：(010) 84501130
上海分公司	上海市普陀区云岭东路89号长风国际大厦5-14楼	邮编：200062	电话：(021) 60656699	传真：(021) 60656688
张江办事处	上海龙东大道3000号9号楼	邮编：201213	电话：(021) 61598888	
广州分公司	广州市珠江新城临江大道3号发展中心大厦25层	邮编：510623	电话：(020) 85185188	传真：(020) 85185195
武汉分公司	武汉市汉口建设大道568号新世界国贸大厦I座37层01、02、03、05单元	邮编：430022	电话：(027) 68850668	传真：(027) 68850488
天津办事处	天津市河西区围堤道125号天信大厦22层2205-07室	邮编：300074	电话：(022) 28408408	传真：(022) 28408410
天津分公司	天津市河东区十一经路78号万隆太平洋大厦1401-1404室	邮编：300171	电话：(022) 84180888	传真：(022) 84180222
济南办事处	济南市顺河街176号齐鲁银行大厦31层	邮编：250001	电话：(0531) 81678100	传真：(0531) 86121628
青岛办事处	青岛崂山区秦岭路18号青岛国展财富中心二号楼四层413室	邮编：266061	电话：(0532) 85793001	传真：(0532) 85793002
石家庄办事处	石家庄市中山东路303号世贸皇冠酒店办公楼12层1201室	邮编：050011	电话：(0311) 86698713	传真：(0311) 86698723
沈阳办事处	沈河区青年大街219号华新国际大厦16层F/G/H/I座	邮编：110016	电话：(024) 23964339	传真：(024) 23964296/97
哈尔滨办事处	哈尔滨市南岗区红军街15号奥威斯发展大厦21层J座	邮编：150001	电话：(0451) 53009797	传真：(0451) 53009640
长春办事处	长春解放大路 2677号长春光大银行大厦1211-12室	邮编：130061	电话：(0431) 88400302/03	传真：(0431) 88400301
大连办事处	大连沙河口区五一路267号17号楼201-I室	邮编：116023	电话：(0411) 84769100	传真：(0411) 84769511
西安办事处	中国陕西省西安市高新区科技二路72号西岳阁201室	邮编：710075	电话：(029) 65692599	传真：(029) 65692555
太原办事处	太原市府西街268号力鸿大厦B区1003室	邮编：030002	电话：(0351) 4937186	传真：(0351) 4937029
乌鲁木齐办事处	乌鲁木齐市新华北路165号广汇中天广场21层TUVW号	邮编：830001	电话：(0991) 6766838	传真：(0991) 6766830
南京办事处	南京市中山路268号汇杰广场2001-2005室	邮编：210008	电话：(025) 83198399	传真：(025) 83198321
苏州办事处	苏州市工业园区苏华路2号国际大厦1711-1712室	邮编：215021	电话：(0512) 68622550	传真：(0512) 68622620
无锡办事处	无锡市太湖广场永和路28号无锡工商综合大楼17层	邮编：214021	电话：(0510) 81009780	传真：(0510) 81009760
南通办事处	江苏省南通市工农路111号华辰大厦A座1103室	邮编：226000	电话：(0513) 85228138	传真：(0513) 85228134
常州办事处	常州市局前街2号常州椿庭楼宾馆1216室	邮编：213000	电话：(0519) 8130710	传真：(0519) 8130711
合肥办事处	合肥市长江东路1104号古井假日酒店913房间	邮编：230001	电话：(0551) 4291993	传真：(0551) 2206956
杭州办事处	杭州市滨江区江南大道588号恒鑫大厦10楼	邮编：310053	电话：(0571) 89825800	传真：(0571) 85825801
南昌办事处	江西省南昌市红谷滩赣江北大道1号中航国际广场1001-1002室	邮编：330043	电话：(0791) 2075750	传真：(0791) 2075751
福州办事处	福州市仓山区建新镇闽江大道169号水乡温泉住宅区二期29号楼101单元	邮编：350000	电话：(0591) 87114853	传真：(0591) 87112046
洛阳办事处	洛阳市涧西区凯旋西路88号华阳广场国际大饭店609室	邮编：471003	电话：(0379) 65588678	传真：(0379) 65588679
厦门办事处	厦门市思明区厦禾路189号银行中心2502-03B室	邮编：361003	电话：(0592) 2386700	传真：(0592) 2386701
宁波办事处	宁波市江东北路1号宁波中信国际大酒店833室	邮编：315040	电话：(0574) 87706808	传真：(0574) 87717043
温州办事处	温州市车站大道高联大厦写字楼9层B2号	邮编：325000	电话：(0577) 86072225/6/7/9	传真：(0577) 86072228
成都办事处	成都市科华北路62号力宝大厦22楼1.2.3.5单元	邮编：610041	电话：(028) 66853777	传真：(028) 66853778
重庆办事处	重庆市渝中区邹容路68号重庆大都会商厦12楼1211-12室	邮编：400010	电话：(023) 63839700	传真：(023) 63839707
佛山办事处	佛山市祖庙路33号百花广场26层2622-2623室	邮编：528000	电话：(0757) 83990312/0029/1312	传真：(0757) 83991312
昆明办事处	昆明市三市街6号柏联广场10楼07-08单元	邮编：650021	电话：(0871) 3647549	传真：(0871) 3647552
长沙办事处	长沙市劳动西路215号湖南佳程酒店14层01, 10, 11室	邮编：410011	电话：(0731) 85112588	传真：(0731) 85159730
郑州办事处	郑州市金水路115号中州皇冠假日酒店C座西翼2层	邮编：450003	电话：(0371) 6593 9211	传真：(0371) 6593 9213
泰州办事处	江苏省泰州市青年南路39号新永泰大酒店8512房间	邮编：225300	电话：(0523) 86397849	传真：(0523) 86397847
中山办事处	中山市东区兴政路1号中环广场3座1103室	邮编：528403	电话：(0760) 8235971	传真：(0760) 8235979
鞍山办事处	鞍山市铁东区南胜利路21号万科写字楼2009室	邮编：114001	电话：(0412) 5575511/5522	传真：(0412) 5573311
烟台办事处	烟台市南大街9号金都大厦2516室	邮编：264001	电话：(0535) 3393899	传真：(0535) 3393998
扬中办事处	扬中市前进北路52号扬中宾馆2018号房间	邮编：212000	电话：(0511) 88398528	传真：(0511) 88398538
南宁办事处	南宁市青秀区民族大道111号广西发展大厦10层	邮编：530000	电话：(0771) 5519761/9762	传真：(0771) 5519760
东莞办事处	东莞市南城区体育路2号鸿禧中心A406单元	邮编：523070	电话：(0769) 22413010	传真：(0769) 22413160
深圳办事处	深圳市罗湖区深南东路5047号深圳发展银行大厦17层H-I室	邮编：518001	电话：(0755) 25841022	传真：(0755) 82080250
贵阳办事处	贵阳市中华南路49号贵航大厦1204室	邮编：550003	电话：(0851) 5887006	传真：(0851) 5887009
海口办事处	海南省海口市文华路18号的海南文华大酒店的第六层 607室	邮编：570305	电话：(0898) 6859 7287	传真：(0898) 6859 7295
施耐德(香港)有限公司	香港鲗鱼涌英皇道979号太古坊和域大厦13楼东翼		电话：(00852) 25650621	传真：(00852) 28111029
施耐德电气中国研修学院	北京市朝阳区望京东路6号施耐德电气大厦	邮编：100102	电话：(010) 84346699	传真：(010) 84501130

施耐德电气中国版权所有

客户关爱中心热线：**400 810 1315**

施耐德电气中国
Schneider Electric China
www.schneider-electric.cn

北京市朝阳区望京东路6号
施耐德电气大厦
邮编: 100102
电话: (010) 8434 6699
传真: (010) 8450 1130

Schneider Electric Building, No. 6,
East WangJing Rd., Chaoyang District
Beijing 100102 P.R.C.
Tel: (010) 8434 6699
Fax: (010) 8450 1130

由于标准和材料的变更，文中所述特性和本资料中的图像只有经过我们的业务部门确认以后，才对我们有约束。

本手册采用生态纸印刷

SCDOC 1715

2012.05