和利时
HollySys

中国创造 世界品质
From China with Global Standard

LK大型可编程控制器

指令手册

北京和利时自动化驱动技术有限公司
Beijing HollySys Automation & Drive Co., Ltd.

版权申明

本手册内容，包括文字、图表、标志、标识、商标、产品型号、软件程序、版面设计等，均受《中华人民共和国著作权法》、《中华人民共和国商标法》、《中华人民共和国专利法》及与之适用的国际公约中有关著作权、商标权、专利权或其他财产所有权法律的保护，为北京和利时系统工程有限公司专属所有或持有。

本手册仅供商业用户阅读、查询，在未得到北京和利时系统工程有限公司特别授权的情况下，无论出于何种原因和目的，均不得用任何电子或机械方法，以任何形式复制和传递本手册的内容。否则本公司将依法追究法律责任。

我们已核对本手册中的内容、图表与所述指令相符，但误差难以避免，并不能保证完全一致。同时，我们会定期对手册的内容、图表进行检查、修改和维护，恕不另行通知。

本手册的说明、图表、简单程序及应用实例完全出于举例说明的目的，我们对其都进行了测试，但因为软件版本的更新和各种应用有许多未知的变化和要求，我们不承担根据本手册或本手册中的实例而构成的实际应用产生的责任。

北京和利时系统工程有限公司保留全部权利。
2007－2012 Copyright Hollysys
HollySys、和利时、LK

和利时 HollySys 的字样和徽标均为北京和利时系统工程有限公司的商标或注册商标。

Microsoft、Windows 和 WindowsNT 是微软公司在美国和/或其他国家分支机构的商标或注册商标。

手册中涉及到的其他商标或注册商标属于他们各自的拥有者。

联系方式

北京和利时自动化驱动技术有限公司
Beijing HollySys Automation & Drive Co., Ltd.
地址：北京市亦庄经济技术开发区地盛中路 2 号院
邮编：100176
电话：010-67888705/67885029/67883817/58981182
传真：010-67880141/58981100

杭州和利时大厦 PLC 服务中心
地址：杭州市下沙经济技术开发区 19 号路北 1 号（310018）
电话：0571-81633793
传真：0571-81633700

技术支持
和利时公司网址：http：//www.hollysys.com
技术支持邮箱：PLC@Hollysys.com
技术支持电话：400-811-1999
版本： 2013 年 12 月

前 言

LK 大型可编程控制器（PLC）是和利时公司在总结十五年的控制系统设计和几千套工程项目实施经验基础上推出的适用于中、高性能控制领域的产品，包括多种 CPU 模块、通信接口模块、I/O 模块及特殊功能模块等。同时，和利时公司还推出了功能强大的 PowerPro 编程软件及丰富的指令系统。

LK 大型可编程控制器（PLC）以先进的计算机技术、控制技术、通信技术和信号处理技术为基础，可为不同工业领域的用户提供个性化的解决方案，适用于逻辑控制、顺序控制、过程控制、传动控制和运动控制等领域。

包含内容

《LK 大型可编程控制器（PLC）指令手册》是和利时公司对其 LK 大型 PLC 包含的所有指令详细介绍的技术手册，主要包含以下信息：

- 指令系统概述
- 操作数
- 数据类型
- 指令详细介绍

读者

本指令手册配合软件手册与硬件手册使用，适用于有一定 PLC 背景知识、掌握了 PowerPro 软件使用方法的工程师、编程人员。

使用本手册，需对 LK 大型 PLC 有一定的了解。

适用范围

本手册仅适用于以下版本编程软件。由于 PowerPro V4.3.1B SP5 软件增加了扩展指令，因此本手册中部分新增扩展指令向下不兼容。

PowerPro V4.3.1B 中文版

PowerPro V4.3.1B 英文版

如何使用本手册

如果已经熟练掌握 PowerPro 编程软件，直接通过目录查找需要的指令。

如果刚刚开始学习 PowerPro 编程软件，建议阅读“第 1 章 LK 大型 PLC 指令概述”。

如果对 PLC 所使用的操作数与数据类型不是很了解，建议阅读第 2、3 章。

附录 包含指令速查表、IEC 标准指令表、指令关联冲突速查表。

相关手册

《LK 大型可编程控制器硬件手册》

《LK 大型可编程控制器编程手册》

目 录

第1章　指令系统概述

和利时公司 LK 大型 PLC 为用户提供了丰富的指令，这些指令均可通过 LK 大型 PLC 的编程软件 PowerPro 进行调用，操作简单，使用方便。

本手册将详细讲述 LK 大型 PLC 指令（简称 LK 指令）。

1.1　指令的定义与分类

在可编程控制器中，使 CPU 完成某种操作或实现某种功能的命令及多个命令的组合称为指令，指令的集合称为指令系统。指令系统是可编程控制器硬件和软件的桥梁，是可编程控制器程序设计的基础。

PowerPro 提供了丰富的指令，按照功能不同可分为转换指令、比较指令、类型转换指令、逻辑运算指令、模拟量应用等类型。为了便于理解和记忆，我们把这些类型分为两大类，一类是基本指令，包括全部 IEC 标准规定指令、高等数学运算指令等，另一类是扩展指令，包括模拟量应用指令、数据传送指令、通讯指令等。扩展指令都是通过功能块方式实现的，而且对应库的名字都是以“HS”开头的，具体见附录。

LK 指令在编程软件中有函数和功能块两种实现方式。函数和功能块都是 PowerPro 软件的程序组织单元，都是预先编好的、实现某种功能的程序，功能块输出可以是一个或多个结果，每一个功能块实例都有一个相关的标识符（即实例名称），函数则不需要标识符，而且只有一个输出结果（即函数的返回值），函数和功能块的具体概念可以参考《LK 大型可编程控制器编程手册》。“附录 LK 指令速查表”中注明了指令的实现方式，FUN 表明指令是以函数方式实现，FB 则表明指令是以功能块方式实现的。

提示：

- 以函数方式实现的指令，在使用的时候都无需声明。
- 以功能块方式实现的指令，在使用的时候都需声明实例名。
- 在 LK 大型 PLC 指令系统中有些指令需要先添加其所对应的库，才能被调用，还有部分指令没有封装在库中，可以直接被调用，在附录中列出各个指令所在库。
- 创建工程时，Standard.lib（标准库）是自动添加到库管理器之中，其它库在使用时需要用户手动添加。

1.2　指令库的定义与分类

编写 PLC 程序的过程中，经常会引用一些有库指令，如字符串处理指令、触发器指令、计数器指令、PID 控制器等等。把这些具有相关功能的指令集合起来进行存储，建立专门的指令库。

指令库是 LK 大型 PLC 指令代码的集合，所有的库都对应有库文件（库名.lib），调用某个库指令，必需载入相应的库文件。

按照库中指令代码功能不同将其分为基本指令库、扩展指令库两类：

- 基本指令库：指基本指令的集合。

- 扩展指令库：指扩展指令的集合，其库名是以“HS”开头。

按照库中指令执行代码所在位置的不同，指令库又可分为两类：

- 第一类：PowerPro 内部指令库。指令执行代码存在于库文件之中，可以使用 PowerPro 软件打开库文件，对指令的执行代码进行修改，用户也可以自己制作内部库。当程序下装到 PLC 之中，占用用户程序空间较大。
- 第二类：PowerPro 外部指令库。指令执行代码已经存在于 PLC 底层系统之中，用户无法修改此类库所包含的执行代码。当程序下装到 PLC 之中，占用用户程序空间较少。

提示：

- 使用 LK 指令时，必须添加相关的库文件（库名.lib）。
- PowerPro 内部指令库一经添加，即使不调用其中的指令，也会占用用户程序空间，因此在实际编程过程中，建议只添加需要的库。

1.2.1 基本指令库

标准指令库 Standard.lib

Standard.lib 属于 PowerPro 外部库，在工程建立时自动添加，无需用户再次添加，包含的指令如图 1-2-1 所示。

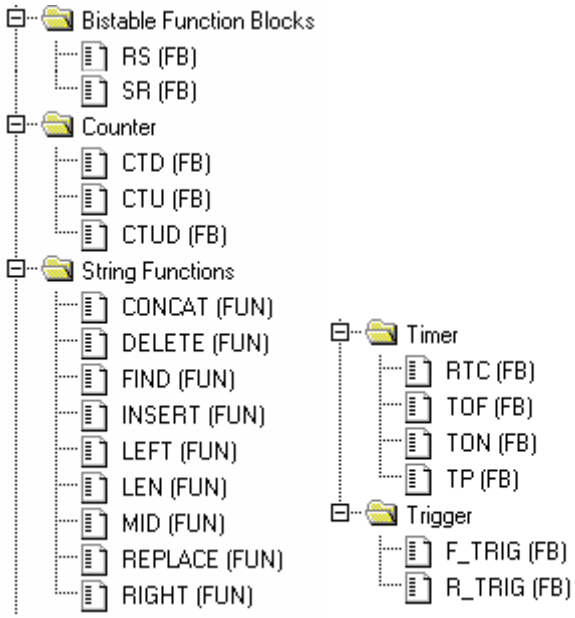


图 1-2-1

该库包含的 LK 指令的含义如表 1-2-1 所示。

表 1-2-1

双稳态指令（Bistable Function Blocks）	计数器（Counter）
RS（复位双稳态器） SR（置位双稳态器）	CTD（递减计数器） CTU（递增计数器） CTUD（递增递减计数器）
字符串指令（String Function）	**定时器（Timer）**

CONCAT（合并字符串指令） DELETE（删除字符串指令） FIND（查找字符串指令） INSERT（插入字符串指令） LEFT（左边取字符串指令） LEN（取字符串长度指令） MID（中间取字符串指令） REPLACE（替换字符串指令） RIGHT（右边取字符串指令）	RTC（实时时钟） TOF（断电延时定时器） TON（通电延时定时器） TP（普通定时器） **触发器（Trigger）** F_TRIG（下降沿检测触发器） R_TRIG（上升沿检测触发器）

应用指令库 Util.lib

Util.lib 属于 PowerPro 内部开放指令库，使用时需用户载入。Util.lib 包含的 LK 指令如图 1-2-2 所示。

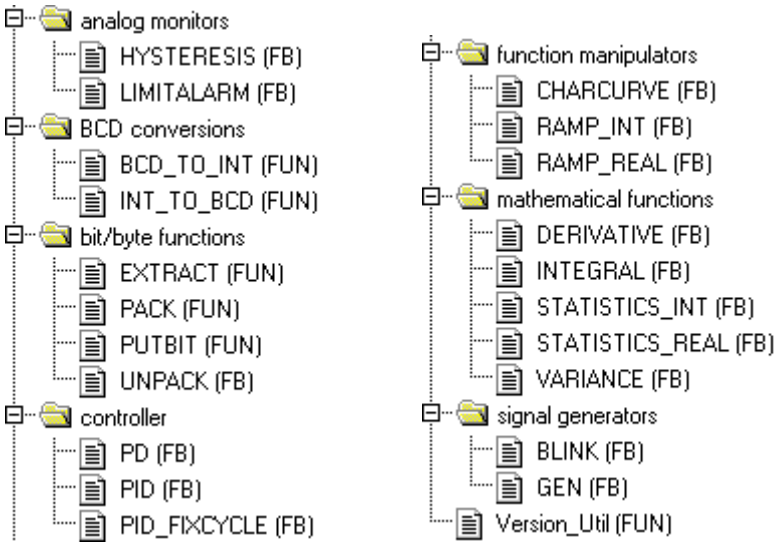


图 1-2-2 应用指令库 Util.lib

该库包含的 LK 指令的含义如表 1-2-2 所示。

表 1-2-2

模拟量应用指令	**BCD 码转换指令**
HYSTERESIS（滞后） LIMITALARM（上下限报警）	BCD_TO_INT（BCD 码转整型） INT_TO_BCD（整型转 BCD 码）
PID 控制器指令	**高等数学运算指令**
P（比例控制器） PD（比例微分控制器） PID（比例积分微分控制器） PID_FIXCYCLE （比例积分微分控制器，周期固定）	DERIVATIVE（微分） INTEGRAL（积分） STATISTICS_INT（整型统计） STATISTICS_REAL（实型统计） VARIANCE（平方偏差）
位转换指令	**函数操纵器指令**
EXTRACT（位提取） PACK（位整合） PUTBIT（位赋值） UNPACK（位拆分）	CHARCURVE（特征曲线） RAMP_INT（整型限速） RAMP_REAL（实型限速）
信号发生器指令	**库版本查看指令**
BLINK（脉冲信号发生器） GEN（典型周期信号发生器）	Version_Util（库版本查看）

系统指令库 SysLibCallBack.lib

SysLibCallback.lib 属于 PowerPro 外部库，其包含的 LK 指令分别如图 1-2-3 与图 1-2-4 所示。

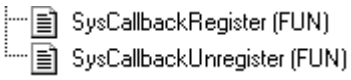


图 1-2-3 系统指令库 SysLibCallback.lib

该库中指令可实现以下功能：

事件调用（SysCallbackRegister）

解除事件调用（SysCallbackUnregistrer）

检查指令库 HS_Check.lib

HS_Check.lib 属于 PowerPro 内部开放库，其包含的 LK 指令分别如图 1-2-4 所示。

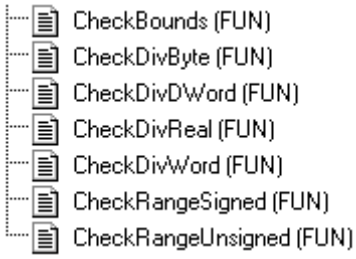


图 1-2-4 检查库指令

该库中指令可实现以下功能：

- 被除数为零的检查功能
- 边界检查功能

IEC 动作指令库 Iecsfc.lib

Iecsfc.lib 属于 PowerPro 内部开放库，其中只包含一个指令如图 1-2-5 所示。

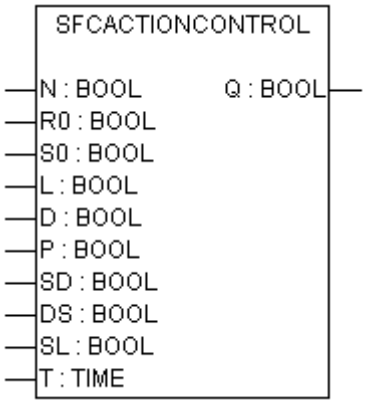


图 1-2-5 IEC 动作指令

该库可实现以下功能：

- SFCActionControl：SFC 语言中 IEC 步关联动作的控制

提示：
在 SFC 编程语言中使用 IEC 步，必须加入该库，否则编译提示错误，当 IEC 步有关联动作时，系统自动调用 SFCActionControl 指令，IEC 步的具体使用方法参见软件手册。

1.2.2 扩展指令库

该库包括模拟量应用库 HS_AnalogConvert.lib、数据传送库 HS_Move.lib、通讯指令库 HS_Communication.lib、SOE 指令库 HS_SOE.lib、预置输出指令库 HS_ScheduledTime.lib、实时时钟指令库 HS_RTC.lib、诊断指令库 HS_Diagnosis.Lib、修改 IP 地址功能库 HS_SetIPAddress.Lib、PID 调节控制器指令库 HS_PIDController.Lib、LK850 量程转换库 HS_LK850_Convert.Lib 和保持型定时器指令库 HS_Timer.Lib，共 11 个。其中，模拟量应用库 HS_AnalogConvert.lib 包含输入数据转工程量指令（HS_HEX_ENGIN）和工程量转输出数据（HS_ENGIN_HEX），数据传送库 HS_Move.lib 指令库包含数据传送指令（HS_MOVE），如图 1-2-6 所示。其他库的具体信息参见第五章相关内容。

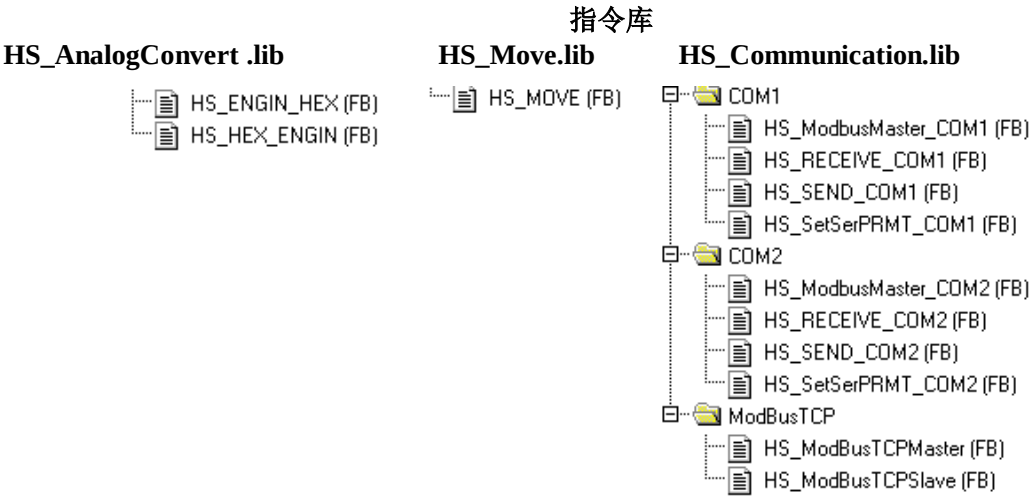


图 1-2-6 扩展指令

1.3 指令库的添加

使用库时，需要保证相应的库文件存在于如下目录：“PowerPro 安装目录\Library\”。

启动 PowerPro，选择“窗口/库管理器”，打开“库管理器”，点击右键，选择“添加库”，如图 1-3-1 所示。

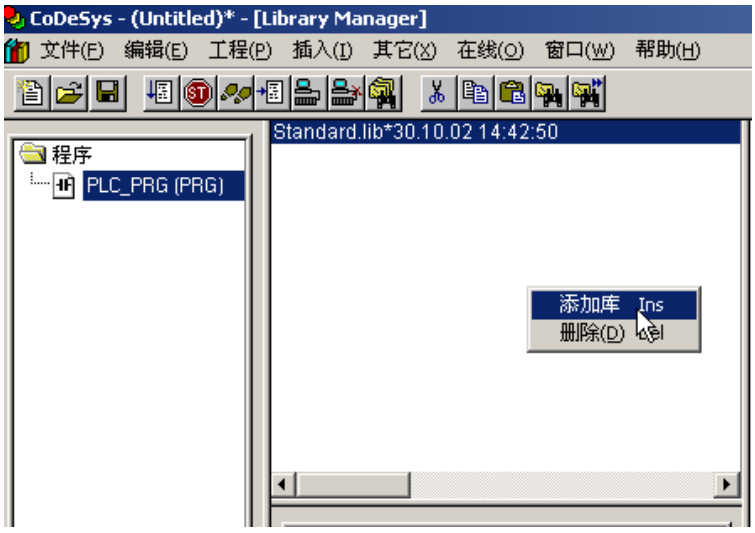


图 1-3-1 添加库

如图 1-3-2，选择需要的库文件，点击“打开”，不论哪种库只需要打开对应的*.lib 文件。

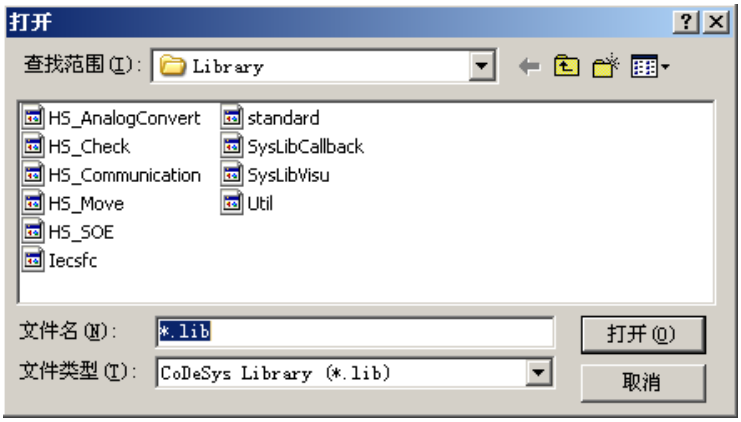


图 1-3-2 选择需要的库文件

如图 1-3-3，上面选择的库被添加到列表中，该库所包含的指令显示在图中鼠标位置。

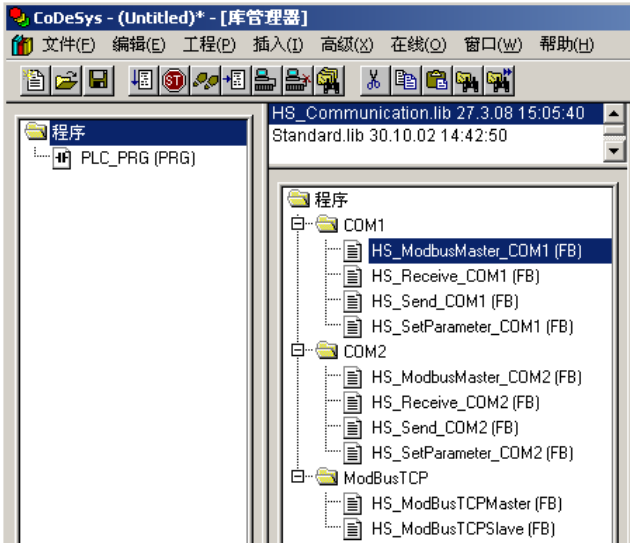


图 1-3-3 添加到列表的库

1.4 指令系统使用注意事项

- 上升沿使能，是指每当使能端由低电平变为高电平并一直保持高电平时，执行指令的相关功能。而且，如果指令的输入参数发生变化，只有使能端再次由低电平变为高电平并，才能生效。
- 高电平使能，是指当使能端保持高电平时，执行指令的相关功能，并且如果指令的输入参数发生变化，则可以实时生效。
- 在使用数学运算指令时，若输出数据定义的范围小于运算结果，则高位丢失。
- 若指令的实例名声明在 RETAIN（掉电保持）区，该指令的所有输入/输出变量都要占用 RETAIN 内存区，所以建议不要在 RETAIN 内存区声明太多的实例，以免 RETAIN 区空间不足。
- M 区的前 4000 个字节，即%MB0~%MB3999，具有掉电保持功能。M 区 4000 字节之后的地址，不具有掉电保持功能。
- RETAIN 区和 M 区的前 4000 个字节的数据在主从机之间切换的时候，具有掉电数据冗余功能。M 区前 4000 字节之外的数据可以通过 MOVE 到 RETAIN 区来实现掉电数据冗余(在上电流程里把要冗余的数据从 RETAIN 区 MOVE 回 M 区)。

- 在梯形图（LD）编程环境下，插入使能运算符与插入功能块是两种不同的指令调用方式。其不同在于，如果采用插入使能运算符调用指令，当使能端低电平时，相应的指令代码不会被扫描，如果采用插入功能块调用指令，不论使能端低电平或高电平时，相应指令代码都会将使能端作为一个输入值来扫描。

第2章 操作数

在可编程控制器中，指令系统是可编程控制器硬件和软件的桥梁，是可编程控制器程序设计的基础。

与计算机的操作指令类似，可编程控制器指令的基本形式也是由操作码和操作数组成。操作码表示 CPU 所要执行的操作类型和所要完成的操作功能。操作数表示 CPU 所要操作的对象和目的。常量、变量、地址和函数调用返回值都可以作为操作数。

2.1 常量

布尔常量

布尔常量只有两个：逻辑值 TRUE 和 FALSE（也可表示为 1 和 0），TRUE 等价于 1，FALSE 等价于 0。

时钟常量

时钟常量一般用来操作时钟，由“T#”（或“t#”）加上“时钟值”构成，时钟值的单位包括天（d）、小时（h）、分（m）、秒（s）和毫秒（ms），它们的正确顺序为 d、h、m、s、ms，如“T#12h38m16s”表示“12 小时 38 分 16 秒”。

下面是**正确**的时钟常量：

- T#18ms (*18 毫秒的一个时钟常量*)
- T#100s12ms (*100 秒 12 毫秒的一个时钟常量，高单位允许超限*)
- t#12h34m15s (*12 小时 34 分 15 秒的一个常量*)

下面是**错误**的时钟常量：

- t#5m68s (*低单位不允许超限*)
- 15ms (*没有 T# *)
- t#4ms13d (*顺序错误*)

日期常量

日期常量由“D#”(“d#”、“DATE#”或“date#”)加上“日期值”构成。例如：

- DATE#2007-1-06 (*日期常量 2007 年 1 月 6 日*)
- d#1980-09-22 (*日期常量 1980 年 9 月 22 日*)

时间常量

时间常量用于存储时间，由“TOD#”（“tod#”、“TIME_OF_DAY#”或“time_of_day#”）加上“时间值”构成。

时间值的格式为：小时:分钟:秒（可以用实数形式输入秒）。例如：

- TOD#00:00:00 (*时间常量为 0 点 0 时 0 分*)
- TIME_OF_DAY#15:36:30.123 (*时间常量为 15 点 36 分 30.123 秒*)

日期时间常量

日期常量和时间常量合并起来称为日期时间常量，由“DT#”(“dt#”、“DATE_AND_TIME#”或“date_and_time#”）加上“日期时间值”构成。例如：

- DT#1980-09-22-15:45:18 (*日期时间常量为 1980 年 9 月 22 日 15 点 45 分 18 秒*)
- date_and_time#2001-03-09-00:00:00 (*日期时间常量为 2001 年 3 月 9 日 0 点 0 分 0 秒*)

数字常量

数字常量的数值可以是二进制、十进制、八进制和十六进制。如果整数值不是十进制值，可以用“进制”加符号“#”放在整数值前面来表示。十进制的 10 至 15 在十六进制中表示为 A 至 F。在数字中可以使用下划线连字符。

数字常量的数据类型可以是 BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT。

默认情况下，不允许“较大”的数据类型作为“较小”的数据类型使用。比如，DWORD 类型的常量不能简单地当作 INT 类型使用，必须使用数据类型转换指令进行转换之后才可以使用。例如：

- 14 (*十进制数 14*)
- 2#1001_0011 (*二进制数 1001_0011*)
- 8#67 (*八进制数 67*)
- 16#AE (*十六进制数 AE *)

实数常量

实数常量用十进制小数和指数来表示，遵循标准的科学计数法格式。实数常量的数据类型是 REAL。例如：

- 7.4 (*实数 7.4*)
- 1.64e+009 (*实数 1.64e+009*)

字符串常量

字符串常量在两个单引号之间，可以包含空格和特殊字符。例如：

- ' Abby and Craig ' (*字符串 Abby and Craig *)
- ':-)' (*字符串:-)*)

在字符串中，当在字符中出现以$开始的复合字符时，解释为下面的形式：

表 2-1-1 以$开始的复合字符串的含义

字符串	代表含义
$$	$字符
$’	单引号
$L 或$l	输入行
$N 或$n	新行
$P 或$p	输入页
$R 或$r	换行
$T 或$t	Tab 键

2.2 变量

变量在 POU（Program Organization Unit）的变量表或者全局变量表中声明，使用中应注意以下几点：

- 变量名不能包含空格和特殊字符，不能多次声明，不能和关键字使用相同的名字。
- 变量名不区分大小写。（如：VAR1、Var1 和 var1 表示相同的变量）
- 变量名识别下划线。（如：“A_BCD”和“AB_CD”被认为是两个不同的变量名）
- 变量名中不能有连续的 2 个下划线。（如：“A_ _B”是错误的变量名）

系统标志符

系统标志符是隐含声明的变量，它在每个特定系统中是不同的。使用命令“插入/声明关键

字”打开输入辅助对话框，选择系统变量类别，这样可以找到系统中可用的系统标志符。

变量访问的语法

- 访问二维数组的元素：<字段名>[Index1, Index2]
- 访问结构变量：<结构名>.<变量名>
- 访问功能块和程序变量：<实例名>.<变量名>

访问变量的数据位

- 在整型变量中，可以访问变量的每个数据位。
- 数据位附加在变量的后面，变量与数据位之间用“圆点”分隔，数据位从 0 开始编号。
- 可以访问变量数据位的数据类型包括：SINT、INT、DINT、USINT、UINT、UDINT、BYTE、WORD 和 DWORD。
- 定义在 VAR_IN_OUT 数据区的变量，不能访问变量的数据位。

例如：

- a : INT; (*定义整型变量 a*)
- b : BOOL; (*定义布尔变量 b*)
- ...
- a.2 := b; (*将布尔变量 b 的值赋给整型变量 a 的第 2 位*)

出错处理：

- 如果数据位大于变量数据位的宽度（比如上例中若 a.16 := b），则会给出下列出错信息：

  Index '<n>' outside the valid range for variable '<var>'!

  因为整型变量的数据位的范围是 0－15，a.16 超出了范围。

- 如果变量类型不允许访问数据位（比如上例中若定义 a : REAL），则会给出下列出错信息：

  Invalid data type '<type>' for direct indexing

  因为实型变量不可以按位访问。

2.3 地址

地址格式

按照规定的地址格式显示内存中的地址。格式为：% 内存区范围 数据格式 地址。

内存区范围	数据格式
I 输入区（Input） Q 输出区（Output） M 中间存储区（Memory location）	X 单个位 B 字节(8 位) W 字（16 位） D 双字（32 位）

例如：

地址格式	对应地址
%QX7.5	输出区的地址 7，第 6 位
%IW4	输入区的地址 4，1 个字
%QB7	输出区的地址 7，1 个字节
%MD48	中间存储区的地址 48，双字

内存位置

在 PowerPro 中，内存地址按照字节排列，从 0 开始，其大小与 PLC 型号有关。例如 M 区

（中间存储区）地址定义如表 2-1-1。

表 2-3-1 M 区地址定义举例

地址	字节定义	字定义	双字定义
0	%MB0	%MW0	%MD0
1	%MB1		
2	%MB2	%MW1	
3	%MB3		
4	%MB4	%MW2	%MD1
5	%MB5		
6	%MB6	%MW3	
7	%MB7		
……	……	……	……
4n	%MB4n	%MW2n	%MDn
4n+1	%MB4n+1		
4n+2	%MB4n+2	%MW2n+1	
4n+3	%MB4n+3		

数据存储格式

PowerPro 软件中数据存储格式以 M 区为例，其中 MSB 表示最高有效位，LSB 最低有效位， 如下面所示：

- 字节

	MSB	LSB
	7	0
%MB300	% MB300	

- 字

	MSB		LSB
	15		0
%MW300	%MB600	%MB601	

- 双字

	MSB			LSB
	31			0
%MD150	%MB600	%MB601	%MB602	%MB603

2.4 函数返回值

在 ST 语言中，调用函数的返回值可以直接作为操作数使用。例如：

Result:= Fct(7) + 3; (*函数 Fct 的返回值加上 3，然后赋值给 Result *)。

第3章　数据类型

编程时可以使用标准数据类型和用户自定义数据类型。数据类型用标识符来表示，规定了数据占用内存空间的大小及存储于其中的数据种类。

标准数据类型包括布尔型数据、整型数据、实型数据、字符串型数据和时间型数据，可以使用 PowerPro 提供的数据类型转换指令进行相互转化。

用户自定义数据类型包括数组、指针、枚举和结构。

3.1　标准数据类型

布尔型数据类型

布尔型变量的标识符为 BOOL，其值为“TRUE”和“FALSE”（也可表示为“1”和“0”）。“TRUE”等价于“1”，“FALSE”等价于“0”。

整型数据类型

整型数据类型的标识符包括 BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT 和 UDINT 等。每一种不同数据类型的取值范围不同，整型数据类型的取值范围如表 3-1-1。

表 3-1-1　整数类型数据范围和存储空间

类型标识符	类型名称	数据下限	数据上限	存储空间
BYTE	字节型	0	255	8 Bit
WORD	字型	0	65535	16 Bit
DWORD	双字型	0	4294967295	32Bit
SINT	单整型	-128	127	8 Bit
USINT	无符号单整型	0	255	8 Bit
INT	整型	-32768	32767	16 Bit
UINT	无符号整型	0	65535	16 Bit
DINT	双整型	-2147483648	2147483647	32 Bit
UDINT	无符号双整型	0	4294967295	32 Bit

提示：
当较长的数据类型转换为较短的数据类型时，会丢失高位信息。

实型数据类型

实型数据类型也称为浮点型数据类型，用于表示有理数。实型数据类型的标识符为 REAL。REAL 型数据占用 32 位内存空间，即 4 个字节。

字符串型数据类型

字符串型数据可以包含任意多个字符，其标识符为 STRING。在声明时所输入的大小决定了存储变量所需要的内存空间，它是指字符串中字符的数量，可以用圆括号或方括号括起来。如

果没有给出大小说明，则缺省大小为 80 个字符。

例如：

str:STRING(35):='This is a String';　(*声明一个包含 35 个字符的字符串*)

注意：字符串变量的头尾必须加单引号。

时间型数据类型

时间型数据类型用于处理时间，其标识符包括 TIME（缩写为 T）、TIME_OF_DAY（缩写为 TOD）、DATE（缩写为 D）和 DATE_AND_TIME（缩写为 DT）。

TIME 表示一个时间值，单位为毫秒，初始值为 0。

TOD 表示当天的时间，单位为毫秒，初始值为凌晨 0 点 0 分。

DATE 表示当前日期，单位为秒。初始值是 1970 年 1 月 1 日。

DT 表示当前日期和时间，单位为秒。初始值是 1970 年 1 月 1 日凌晨 0 点 0 分。

3.2 自定义数据类型

3.2.1 数组

一维、二维和三维数组属于基本的数据类型。在 POU 的变量表或者全局变量表中，都可以声明数组。数组的标识符为 ARRAY。

声明数组的语法

<数组名>:ARRAY [<L1>..<U1>,<L2>..<U2>,<L3>..<U3>] OF <基本数据类型>;

L1、L2 和 L3 表示字段范围的最小值，U1、U2 和 U3 表示字段范围的最大值。字段范围必须是整数。例如：

Card_game: ARRAY [1..13, 1..4] OF INT;　(*定义一个整型的二维数组 Card_game*)

数组的初始化

在数组定义时，可以初始化数组中所有元素，也可以不进行初始化。

举例 1：数组的完全初始化

```
Arr1:ARRAY [1..5] OF BYTE:= 1,2,3,4,5;
Arr2:ARRAY [1..2,3..4] OF INT := 1,3(7) ;              （*即 1,7,7,7 的缩写形式*）
Arr3:ARRAY [1..2,2..3,3..4] OF INT := 2(0),4(4),2,3;   （*即 0,0,4,4,4,4,2,3 的缩写形式*）
```

举例 2：结构数组的初始化

```
TYPE STRUCT1:
STRUCT
    p1:int;
    p2:int;
    p3:dword;
END_STRUCT
END_ TYPE
ARRAY[1..3] OF STRUCT1:=(p1:=1,p2:=10,p3:= 3),(p1:=2,p2:=0,p3:=2),
                        (p1:=4,p2:=5,p3:=1);
```

举例 3：数组的部分初始化

Arr1:ARRAY [1..10] OF BYTE:= 1,2;

对于那些没有预先赋值的元素，按照基本数据类型的缺省初始值进行初始化。在此例中，元素[3]到[10]被初始化为 0。

数组的访问

在二维数组中访问数组元素，使用下列语法：

<Field_Name>[Index1,Index2]

例如：

数组的访问：

Card_game[9,2]

提示：

- 如果在工程中使用 CheckBounds 来定义函数，则可以自动进行数组越界错误检查。该函数名的关键字必须是 CheckBounds，CheckBounds 的程序如图 3-2-1 所示。

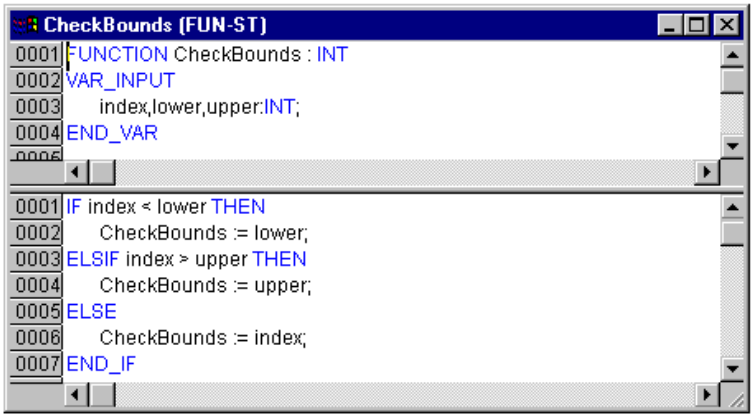


图 3-2-1 CheckBounds 的程序

- 若工程中没有上面所述的 CheckBounds 程序，在图 3-2-2 的例子中，A[B]应该为 A[10]，从而超出了数组 A 的最大上界值 7，程序编译时出错。
- 但因为工程中定义了上面的 CheckBounds 函数，所以执行图 3-2-2 的程序时，A[B]会被当作 A[7]来使用，将布尔量 TRUE 赋值给 A[7]，编译时不出错。

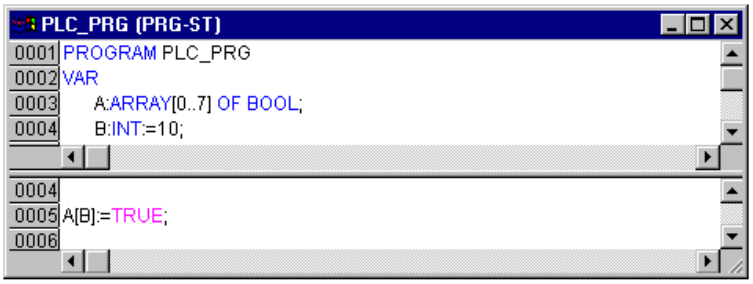


图 3-2-2 举例

注意：

- 数组不可定义得过大，否则编译将无法通过（LK 随机数据区的大小为 4M，即 4,194,304Bytes）。
- 将数组的边界值定义为变量时，要注意不能越界，即不可超出 LK 总数据区的大小（约为 6M），否则将引起不可预期的后果。

3.2.2 指针

当程序运行的时候，通过指针可以取得变量或功能块实例的地址。指针可以指向任何数据类型或功能块类型，包括用户自定义的数据类型。声明指针的语法：

<指针名>：POINTER TO <数据类型/功能块类型>。

通过取地址指令 ADR 可以将变量或指令实例的地址取出。通过在指针标识符的后面增加取地址内容指令“^”可以取得指针所指地址的数据。

例如：

程 序	含 义
pt:POINTER TO INT; Var_int1:INT := 5; Var_int2:INT; pt := ADR(Var_int1); Var_int2:= pt^;	(*定义一个整型数据的指针 pt*) (*定义整型变量 Var_int1，使其等于 5*) (*定义整型变量 Var_int2*) (*取出 Var_int1 变量的地址，将地址值赋给 pt*) (*将指针 pt 所指地址的值赋给 Var_int2，Var_int2=5*)

3.2.3 枚举

枚举是由一长串的数字常量组成的自定义数字类型，这些常量称为枚举值。

只要枚举值声明在 POU 内，在整个程序范围内都可以被识别。建议将枚举创建在 PowerPro 程序左下角选项卡中的  数据类型（Data types）里，通过在数据类型（Data types）上添加对象（Add Object）来创建一个新的枚举值。

枚举以关键字 TYPE 开始，以关键字 END_TYPE 结束。声明枚举的语法：

TYPE<标识符>:(<Enum_0>,<Enum_1>,…,<Enum_n>);

END_TYPE

注意

- 枚举值中可以包含标识符变量，初始化时，该标识符变量被初始化为该枚举中的一个值。可以把任何数字量分配给枚举值。如果枚举值没有被初始化，则从 0 开始递增进行初始化。定义的枚举值与其他标准数据类型兼容，就像使用 INT 数据类型一样对其进行操作。

举例：

程 序	含 义

TYPE TRAFFIC_SIGNAL: (Red, Yellow, Green:=10);
END_TYPE
 	
TRAFFIC_SIGNAL1: TRAFFIC_SIGNAL;
TRAFFIC_SIGNAL1:=0;
FOR i:= Red TO Green DO
i := i + 1;
END_FOR
	(*每个颜色的初始值是 Red=0，Yellow=1，Green=10*)
	
	(*交通信号的值是 Red*)

- 相同的枚举值不能使用两次。

举例：

TRAFFIC_SIGNAL: (red, yellow, green);
COLOR: (blue, white, red);
错误：red 不能同时被 TRAFFIC_SIGNAL 和 COLOR 使用。

3.2.4 结构

结构创建在 PowerPro 软件左下角数据类型选项卡窗口中，通过在数据类型上添加对象来创建一个新的结构。

结构以关键字 TYPE 和 STRUCT 开始，以关键字 END_STRUCT 和 END_TYPE 结束。

声明结构的语法

```
TYPE <结构名>:
STRUCT
<变量声明 1>
  :
<变量声明 n>
END_STRUCT
END_TYPE
```

<结构名>是一种可以在整个工程中被识别的数据类型，可以像标准数据类型一样引用，但不可以指定结构中变量的地址（即变量名之后不允许使用 AT 来指定该变量的地址）。

举例（定义名为 Polygonline 的结构）：

```
TYPE Polygonline:
STRUCT
  Start:ARRAY [1..2] OF INT;
  Point1:ARRAY [1..2] OF INT;
  Point2:ARRAY [1..2] OF INT;
  Point3:ARRAY [1..2] OF INT;
  Point4:ARRAY [1..2] OF INT;
End:ARRAY [1..2] OF INT;
END_STRUCT
END_TYPE
```

举例（初始化结构）：

```
Poly_1:polygonline:= (Start:=3,3,Point1 =5,2,Point2:=7,3,Point3:=8,5,
                      Point4:=5,7,End :=3,5);
```

结构成员的访问

<结构名>.<结构成员名>

举例：

如果结构名为“Week”，其中一个成员名为“Monday”，则可以用下面的形式访问：

Week.Monday

第4章　基本指令

基本指令是 PowerPro 指令系统中基本的指令，包括算术运算指令、赋值指令、逻辑运算指令、移位指令、选择指令、比较指令、数据类型转换指令、初等数学运算指令、地址运算指令等二十五个类型的指令，本章详细介绍各个指令的功能及使用方法，通过简单的指令举例来说明指令使用过程中应注意的问题。

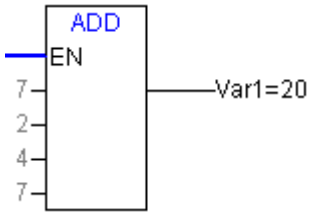
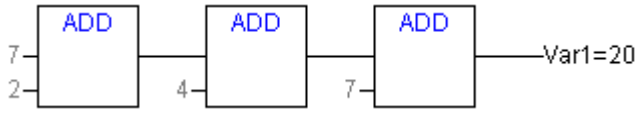
4.1 算术运算指令

算术运算指令主要是指通用的算术指令，包括加法、乘法、减法、除法和取余运算，这类运算的特点都是相对简单，因此在介绍此类指令时，仅简要标识了其在各类语言中的表示方法，使用时只要不违反常规（如除零，即除法运算中被除数为零）即可。

4.1.1 ADD——加法指令

- **功能**：两个（或者多个）变量或常量相加。
- **输入/输出数据类型**：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL、TIME。
- 指令使用举例

变量定义

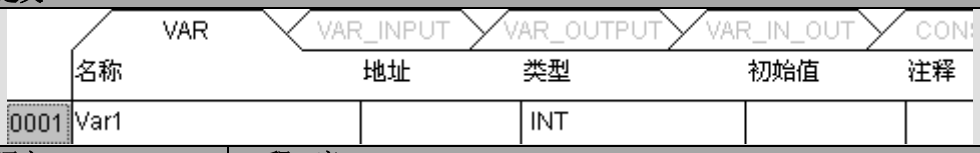
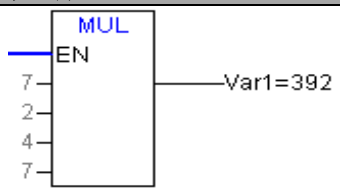
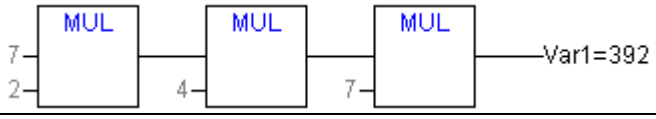
编程语言	程　序
梯形图（LD）	
结构化文本（ST）	Var1:=7+2+4+7;　（*结果 Var1 为 20*）
指令列表（IL）	LD　7 ADD　2,4,7 ST　Var1　（*结果 Var1 为 20*）
功能块（FBD）	

提示：

- TIME 型变量也可以使用加法功能，两个 TIME 变量相加得到一个新的时间。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。

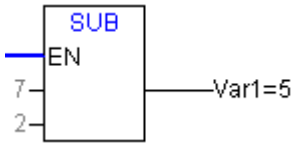
4.1.2 MUL——乘法指令

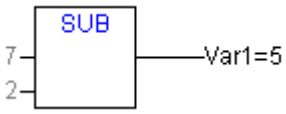
- **功能**：两个（或者多个）变量或常量相乘。
- **输入/输出数据类型**：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL。
- 指令使用举例

变量定义	
	
编程语言	**程　序**
梯形图（LD）	
结构化文本（ST）	Var1:=7*2*4*7;　（*结果 Var1 为 392*）
指令列表（IL）	LD　7 MUL　2,4,7 ST　Var1　（*结果 Var1 为 392*）
功能块（FBD）	

4.1.3 SUB——减法指令

- **功能**：两个变量或常量相减。
- **输入/输出数据类型**：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL、TOD。
- 指令使用举例

变量定义	
	
编程语言	**程　序**
梯形图（LD）	
结构化文本（ST）	Var1:=7-2;　（*结果 Var1 为 5*）
指令列表（IL）	LD　7 SUB　2 ST　Var1　（*结果 Var1 为 5*）

功能块（FBD）	

提示：

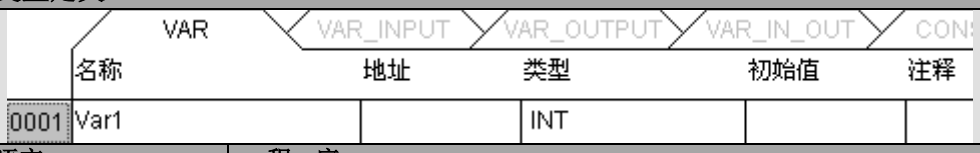
- TIME 型变量也可以使用减法功能，两个 TIME 变量相减得到一个新的时间。
  例如：t#1m35s - t#50s = t#45s ，但时间结果不能有负值。
- TOD 型变量也可以使用减法功能，两个 TOD 型相减得到一个新的 TIME 型数据，例如：TOD#45:40:30- TOD#22:30:20=T#1390m10s0ms ,但时间结果不能有负值。

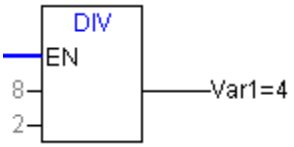
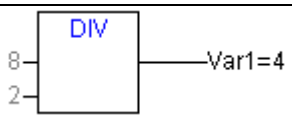
4.1.4 DIV——除法指令

- **功能**：变量或常量相除。
- **输入/输出数据类型**：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL。

✧ **指令使用举例**

变量定义



编程语言	**程　序**
梯形图（LD）	
结构化文本（ST）	Var1:=8/2;　　（*结果 Var1 为 4*）
指令列表（IL）	LD　8 DIV　2 ST　Var1　　（*结果 Var1 为 4*）
功能块（FBD）	
FBD	

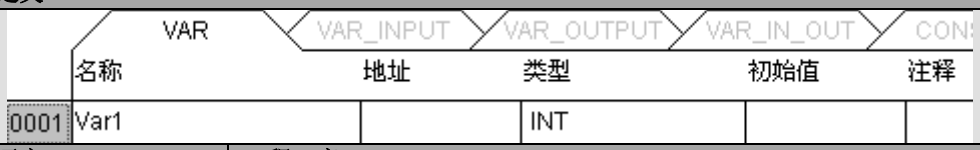
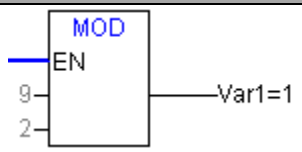
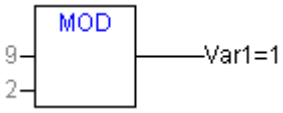
提示：

- 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，避免了除数为零的现象。

4.1.5 MOD——取余指令

- **功能**：变量或常量相除取余，结果为两数相除后的余数，是一个整数。
- **输入/输出数据类型**：BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT。

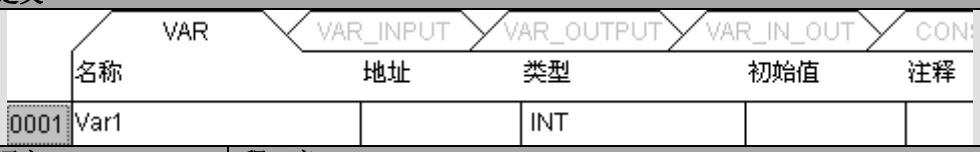
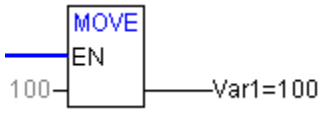
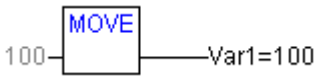
✧ 指令使用举例

变量定义	
	
编程语言	**程　序**
梯形图（LD）	
结构化文本（ST）	Var1:=9 MOD 2;　（*结果 Var1 为 1*）
指令列表（IL）	LD　9 MOD　2 ST　Var1　（*结果 Var1 为 1*）
功能块（FBD）	

4.2 赋值指令

- MOVE—赋值指令
- **功能**：将一个常量或者变量的值赋给另外一个变量。
- **输入/输出数据类型**： BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL、TIME、DT。

✧ 指令使用举例

变量定义	
	
编程语言	**程　序**
梯形图（LD）	
结构化文本（ST）	Var1:=100;　（*结果 Var1 为 100*）
指令列表（IL）	LD　100 MOVE ST　Var1　（*结果 Var1 为 100*）
功能块（FBD）	

4.3 逻辑运算指令

逻辑运算指令主要针对 BOOL、BYTE、WORD 和 DWORD 型的变量，对其进行逻辑与、或、非以及异或运算。对非 BOOL 型的变量，实际的运算过程是将其转换成 8 位、16 位或者 32 位二进制数，而后逐位进行逻辑运算。

4.3.1 AND——与指令

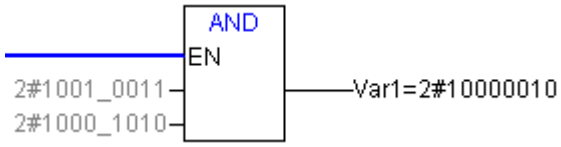
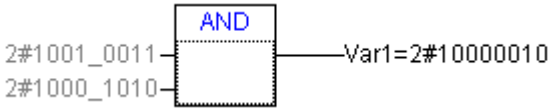
- **功能**：变量或常量的相与运算。
- **输入/输出数据类型**：BOOL、BYTE、WORD 和 DWORD。

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONS

名称	地址	类型	初始值	注释
0001 Var1		BYTE		

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	Var1:=2#1001_0011 AND 2#1000_1010; （*结果 Var1 为 2#10000・010*）
指令列表（IL）	LD　2#1001_0011 AND　2#1000_1010 ST　Var1　（*结果 Var1 为 2#10000010*）
功能块（FBD）	

4.3.2 OR——或指令

- **功能**：两个或多个变量或常量对应的二进制位进行“逻辑或”运算。
- **输入/输出数据类型**：BOOL、BYTE、WORD 和 DWORD。

表 4-3-1 列出了或运算的结果。

表 4-3-1　或运算

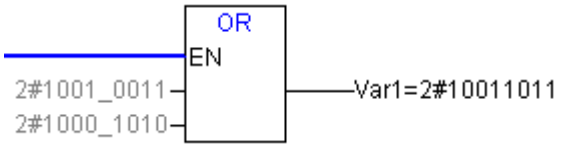
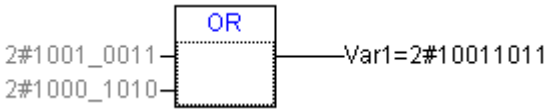
A 位	B 位	结果
0	0	0
0	1	1
1	0	1
1	1	1

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONS

名称	地址	类型	初始值	注释
0001 Var1		BYTE		

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	Var1:=2#1001_0011 OR 2#1000_1010 ; （*结果 Var1 为 2#10011011*）
指令列表（IL）	LD　2#1001_0011 OR　2#1000_1010 ST　Var1　（*结果 Var1 为 2#10011011*）
功能块（FBD）	

4.3.3　XOR——异或指令

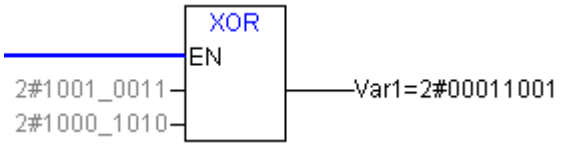
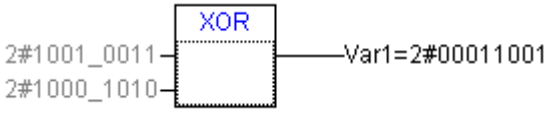
- **功能**：两个变量或常量对应二进制位进行“逻辑异或”运算。
- **输入/输出数据类型**：BOOL、BYTE、WORD 和 DWORD。

表 4-3-2 列出了异或运算的结果。

表 4-3-2　异或运算

A 位	B 位	结果
0	0	0
0	1	1
1	0	1
1	1	0

✧ **指令使用举例**

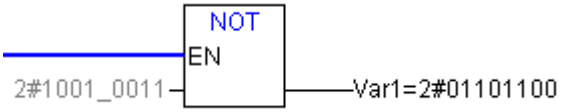
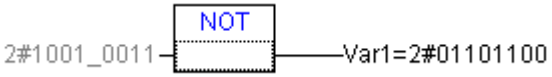
变量定义	
	
编程语言	程　序
梯形图（LD）	
结构化文本（ST）	Var1:=2#1001_0011 XOR 2#1000_1010 ; （*结果 Var1 为 2#00011001*）
指令列表（IL）	LD　2#1001_0011 XOR　2#1000_1010 ST　Var1　（*结果 Var1 为 2#00011001*）
功能块（FBD）	

4.3.4 NOT——取非指令

- **功能**：变量或常量的取非运算， 逐位取非。
- **输入/输出数据类型**：BOOL、BYTE、WORD 和 DWORD。

✧ **指令使用举例**

变量定义

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	Var1:= NOT 2#1001_0011 ; （*结果 Var1 为 2#01101100*）
指令列表（IL）	LD　2#1001_0011 NOT ST　Var1　（*结果 Var1 为 2#01101100*）
功能块（FBD）	

4.4 移位指令

4.4.1 SHL——左移指令

- **功能**：对操作数进行按位左移，左边移出的位被丢掉，右边移空位自动补 0。
- **输入/输出数据类型**：BYTE、INT、WORD、DWORD、SINT、UINT。

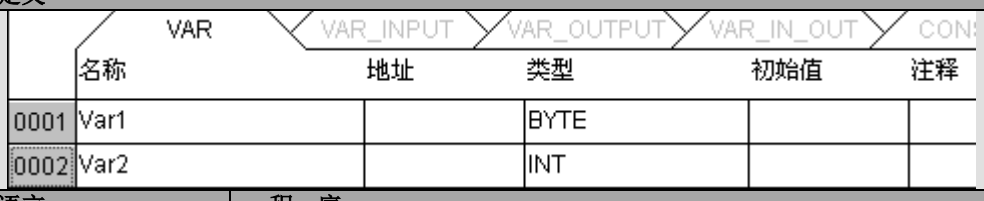
以左移 3 次为例，表 4-4-1 列出了左边的结果。

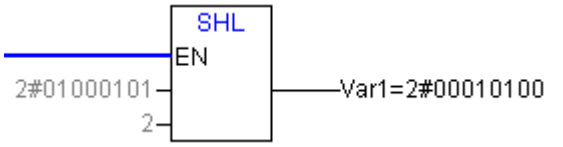
表 4-4-1 左移 3 次

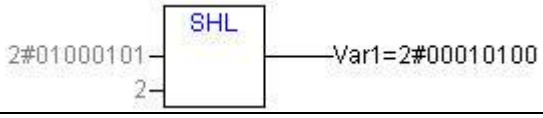
	B7	B6	B5	B4	B3	B2	B1	B0
原数	1	0	1	0	0	1	0	1
左移 1 次	0	1	0	0	1	0	1	0
左移 2 次	1	0	0	1	0	1	0	0
左移 3 次	0	0	1	0	1	0	0	0

✧ **指令使用举例**

变量定义



编程语言	程　序
梯形图（LD）	

结构化文本（ST）	Var1:=SHL(16#45,2); （*结果 Var1 为 16#14*） Var2:=SHL(16#45,2); （*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。
指令列表（IL）	LD 16#45 SHL 2 ST Var1 （*结果 Var1 为 16#14*）
功能块（FBD）	

4.4.2 SHR——右移指令

- **功能**：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。
- **输入/输出数据类型**：BYTE、INT、WORD、DWORD、SINT、UINT。

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONS

	名称	地址	类型	初始值	注释
0001	Var1		BYTE		
0002	Var2		INT		

编程语言	**程 序**
梯形图（LD）	
结构化文本（ST）	Var1:=SHR(16#45,2); （*结果 Var1 为 16#11*） Var2:=SHR(16#45,2); （*结果 Var2 为 16#0011*）
指令列表（IL）	LD 16#45 SHR 2 ST Var1 （*结果 Var1 为 16#11*）
功能块（FBD）	

4.4.3 ROL——循环左移指令

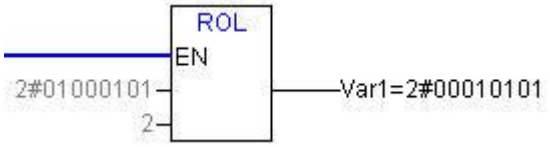
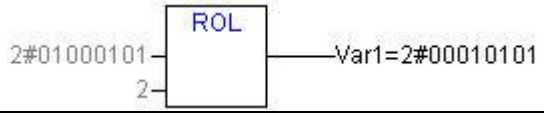
- **功能**：对操作数进行按位循环左移，左边移出的位直接补充到右边最低位。
- **输入/输出数据类型**：BYTE、INT、WORD、DWORD、SINT、UINT。

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONS

	名称	地址	类型	初始值	注释
0001	Var1		BYTE		
0002	Var2		INT		

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	Var1:=ROL(16#45,2);　（*结果 Var1 为 16#15*） Var2:= ROL (16#45,2);　（*结果 Var2 为 16#0114*） 注意：在循环左移过程中，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。
指令列表（IL）	LD　16#45 ROL　2 ST　Var1　（*结果 Var1 为 16#15*）
功能块（FBD）	

4.4.4　ROR——循环右移指令

- **功能**：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。
- **输入/输出数据类型**：BYTE、INT、WORD、DWORD、SINT、UINT。
- **指令使用举例**

变量定义					
	VAR	VAR_INPUT	VAR_OUTPUT	VAR_IN_OUT	CONS
	名称	地址	类型	初始值	注释
0001	Var1		BYTE		
0002	Var2		INT		

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	Var1:=ROR(16#45,2)　（*结果 Var1 为 16#51*） Var2:= ROR (16#45,2)　（*结果 Var2 为 16#4011*） **注意**：在循环右移过程中，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。
指令列表（IL）	LD　16#45 ROR　2 ST　Var1　（*结果 Var1 为 16#51*）
功能块（FBD）	

4.5 选择指令

所有的选择指令在执行时均可以带有变量。为了能够更加清楚地说明问题，以下各例只使用常量。被选择的输入数据类型存储长度应不大于输出类型存储长度。

4.5.1 SEL——二选一指令

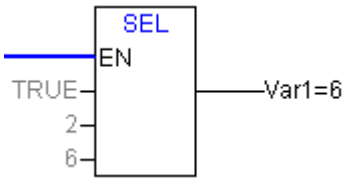
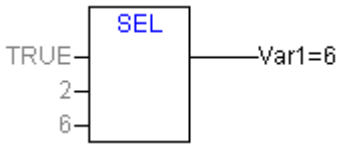
- **功能**：通过选择开关在两个输入数据中选择一个作为输出，选择开关为 FALSE 时输出为第一个输入数据，选择开关为 TRUE 时输出为第二个输入数据。
- **指令格式**：OUT := SEL(G, IN0, IN1)，其中 G 为选择开关，IN0 和 IN1 分别为第一个输入数据和第二个输入数据。
- **输入/输出数据类型**：G 必须是 BOOL 类型，IN0、IN1 可以是任意数据类型。

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONS

	名称	地址	类型	初始值	注释
0001	Var1		INT		
0002	Var2		INT		

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	Var1:=SEL(TRUE,3,4); (*结果 Var1 为 4 *)
指令列表（IL）	LD　TRUE SEL　2,6 ST　Var1　(*结果 Var1 为 6*) LD　FALSE SEL　2,6 ST　Var2　(*结果 Var2 为 2*)
功能块（FBD）	

4.5.2 MAX——取最大值指令

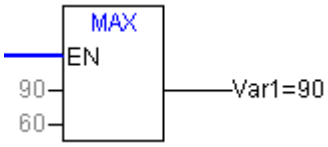
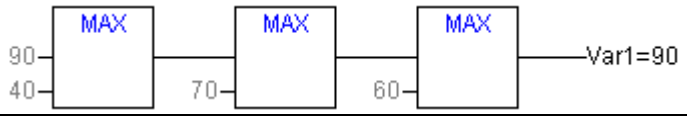
- **功能**：在多个输入数据中选择最大值作为输出。
- **指令格式**：OUT:=MAX(IN0, IN1)，其中 IN0 和 IN1 分别为第 1 个输入数据和第 2 个输入数据，OUT 是输出数据。
- **输入/输出数据类型**：IN0, IN1 和 OUT 可以是任意数据类型。

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONS

	名称	地址	类型	初始值	注释
0001	Var1		INT		
0002	Var2		INT		

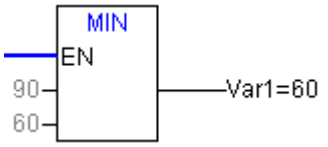
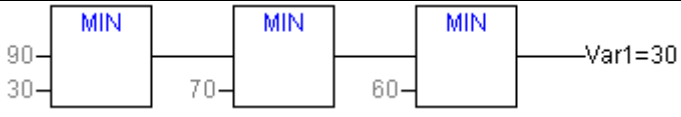
编程语言	程　序
梯形图（LD）	
结构化文本（ST）	Var1:=MAX(90,60);　　(*结果 Var1 为 90 *) Var2:=MAX(40,MAX(90,60));　　(*结果 Var2 为 90 *)
指令列表（IL）	LD　　90 MAX　　40 MAX　　70 MAX　　60 ST　　Var1　　（*结果 Var1 为 90*）
功能块（FBD）	

4.5.3　MIN——取最小值指令

- **功能**：在多个输入数据中选择最小值作为输出。
- **指令格式**：OUT:=MIN(IN0, IN1)，其中 IN0 和 IN1 分别为第 1 个输入数据和第 2 个输入数据，OUT 是输出数据。
- **输入/输出数据类型**：IN0， IN1 和 OUT 可以是任意数据类型。

✧ **指令使用举例**

变量定义

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	Var1:=MIN(90,30);　　(*结果 Var1 为 30 *) Var2:=MIN(MIN(90,30),60);　　(*结果 Var2 为 30 *)
指令列表（IL）	LD　　90 MIN　　30 MIN　　40 MIN　　70 ST　　Var1　　（*结果 Var1 为 30*）
功能块（FBD）	

4.5.4 LIMIT——极限值指令

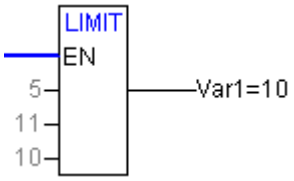
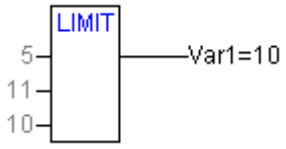
- **功能**：判断输入数据是否在最小值和最大值之间，若输入数据在二者之间，则直接把输入数据作为输出数据进行输出。若输入数据大于最大值，则把最大值作为输出值。若输入数据小于最小值，则把最小值作为输出值。
- **指令格式**： OUT := LIMIT(Min, IN, Max)
- **输入/输出数据类型**：IN 和 OUT 可以是任意数据类型。

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONS

	名称	地址	类型	初始值	注释
0001	Var1		INT		

编程语言	程　序
梯形图（LD）	 （*11 为输入数据，5 为最小值，10 为最大值*）
结构化文本（ST）	Var1:=LIMIT(30,90,80);　(*结果 Var1 为 80 *)
指令列表（IL）	LD　90 LIMIT　30，80 ST　Var1　(*结果 Var1 为 80*)
功能块（FBD）	 （*11 为输入数据，5 为最小值，10 为最大值*）

4.5.5 MUX——多选一指令

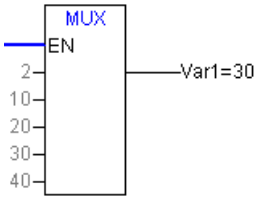
- **功能**：通过控制数在多个输入数据中选择一个作为输出。
- **指令格式**： OUT:=MUX(K,IN0,…,INn)，其中 K 为控制数，IN0,…,INn 为输入数据，OUT 为输出结果。控制数为 K 时选择第 INk 个输入数据作为输出。
- **输入/输出数据类型**：IN0，…， INn 和 OUT 可以是任意数据类型，K 必须是 BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT 或 UDINT。

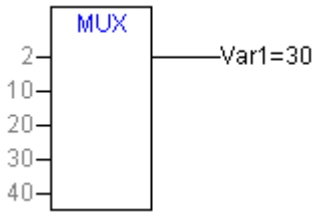
✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONS

	名称	地址	类型	初始值	注释
0001	Var1		INT		

编程语言	程　序
梯形图（LD）	 （*2 为控制数据，对应于 30，所以输出 30 *）
结构化文本（ST）	Var1:=MUX(0,30,40,50,60,70,80);（*结果 Var1 为 30*）

指令列表（IL）	LD 0 MUX 30，40，50，60，70，80 ST Var1 （*结果 Var1 为 30*）
功能块（FBD）	 （*2 为控制数据，对应于 30，所以输出 30*）

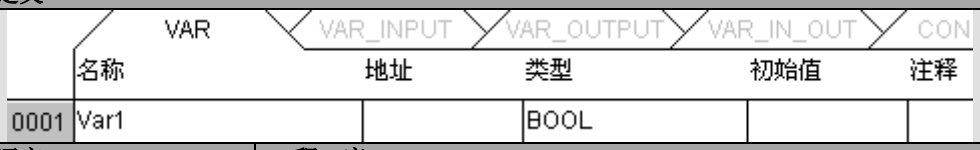
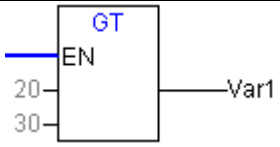
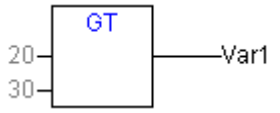
4.6 比较指令

所有的比较指令在执行时均可以带有变量。为了能够更加清楚地说明问题，以下各例只使用常量。

4.6.1 GT——大于指令

- **功能**：判断两个操作数的大小，当第一个数大于第二个数时输出 TRUE，否则输出为 FALSE。
- **输入数据类型**：BOOL、BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL、TIME、DATE、TOD、DT 和 STRING；
- **输出数据类型**：BOOL。

✧ **指令使用举例**

变量定义	
	
编程语言	**程 序**
梯形图（LD）	 （*结果 Var1 为 FALSE*）
结构化文本（ST）	Var1:=20>30;
指令列表（IL）	LD 20 GT 30 ST Var1 （*结果 Var1 为 FALSE*）
功能块（FBD）	 （*结果 Var1 为 FALSE*）

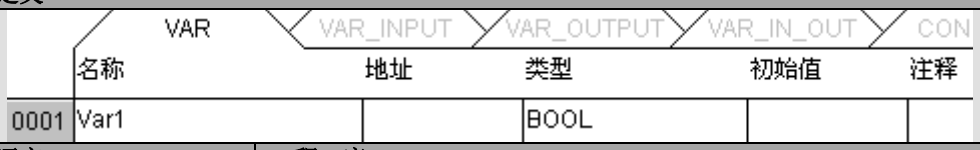
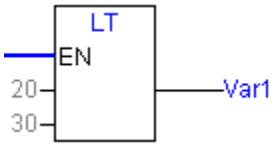
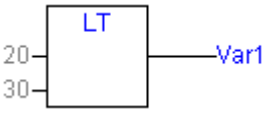
4.6.2 LT——小于指令

- **功能**：判断两个操作数的大小，当第一个数小于第二个数时返回 TRUE，否则结果为 FALSE。
- **输入数据类型**：BOOL、BYTE、WORD、DWORD、SINT、USINT、INT、UINT、

DINT、UDINT、REAL、TIME、DATE、TOD、DT 和 STRING；

- **输出数据类型**：BOOL。

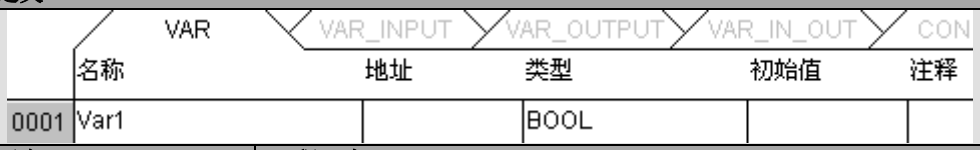
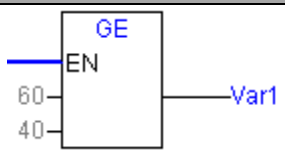
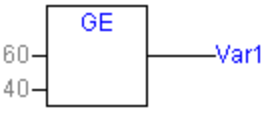
✧ **指令使用举例**

变量定义	
	
编程语言	**程　序**
梯形图（LD）	 （*结果 Var1 为 TRUE*）
结构化文本（ST）	VAR1:=20<30;　　（*结果 Var1 为 TRUE*）
指令列表（IL）	LD　20 LT　30 ST　Var1　　（*结果 Var1 为 TRUE*）
功能块（FBD）	 （*结果 Var1 为 TRUE*）

4.6.3　GE——大于等于指令

- **功能**：判断两个操作数的大小，当第一个数大于等于第二个数时返回 TRUE，否则结果为 FALSE。
- **输入数据类型**：BOOL、BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL、TIME、DATE、TOD、DT 和 STRING；
- **输出数据类型**：BOOL。

✧ **指令使用举例**

变量定义	
	
编程语言	**程　序**
梯形图（LD）	 （*结果 Var1 为 TRUE*）
结构化文本（ST）	VAR1:=60>=40;　　（*结果 Var1 为 TRUE*）
指令列表（IL）	LD　60 GE　40 ST　Var1　　（*结果 Var1 为 TRUE*）
功能块（FBD）	 （*结果 Var1 为 TRUE*）

4.6.4 LE——小于等于指令

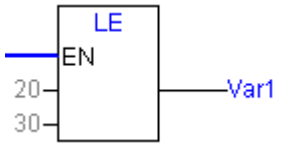
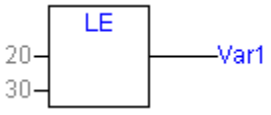
- **功能**：判断两个操作数的大小，当第一个数小于等于第二个数时返回 TRUE，否则结果为 FALSE。
- **输入数据类型**：BOOL、BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL、TIME、DATE、TOD、DT 和 STRING；
- **输出数据类型**：BOOL。

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CON

	名称	地址	类型	初始值	注释
0001	Var1		BOOL		

编程语言	程　序
梯形图（LD）	 （*结果 Var1 为 TRUE*）
结构化文本（ST）	VAR1:=20<=30;　　（*结果 Var1 为 TRUE*）
指令列表（IL）	LD　20 LE　30 ST　Var1　　（*结果 Var1 为 TRUE*）
功能块（FBD）	 （*结果 Var1 为 TRUE*）

4.6.5 EQ——等于指令

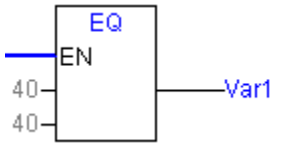
- **功能**：判断两个操作数是否相等，当第一个数等于第二个数时返回 TRUE，否则结果为 FALSE。
- **输入数据类型**：BOOL、BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL、TIME、DATE、TOD、DT 和 STRING；
- **输出数据类型**：BOOL。

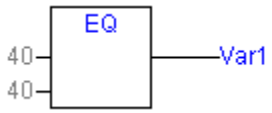
✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CON

	名称	地址	类型	初始值	注释
0001	Var1		BOOL		

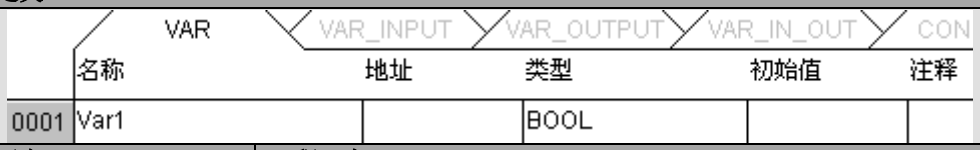
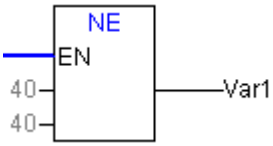
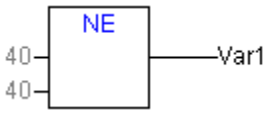
编程语言	程　序
梯形图（LD）	 （*结果 Var1 为 TRUE*）
结构化文本（ST）	VAR1:=40=40;　　（*结果 Var1 为 TRUE*）
指令列表（IL）	LD　40 EQ　40

	ST Var1 （*结果 Var1 为 TRUE*）
功能块（FBD）	 （*结果 Var1 为 TRUE*）

4.6.6 NE——不等于指令

- **功能**：判断两个操作数是否不相等，当第一个数不等于第二个数时返回 TRUE，否则结果为 FALSE。
- **输入数据类型**：BOOL、BYTE、WORD、DWORD、SINT、USINT、INT、UINT、DINT、UDINT、REAL、TIME、DATE、TOD、DT 和 STRING；
- **输出数据类型**：BOOL。

✧ **指令使用举例**

变量定义	
	
编程语言	**程 序**
梯形图（LD）	 （*结果 Var1 为 FALSE*）
结构化文本（ST）	VAR1:=40<>40; （*结果 Var1 为 FALSE*）
指令列表（IL）	LD 40 NE 40 ST Var1 （*结果 Var1 为 FALSE*）
功能块（FBD）	 （*结果 Var1 为 FALSE*）

4.7 数据类型转换指令

PowerPro 提供了 240 个数据类型转换指令，用于各种数据类型之间相互转换。

语法：<TYPE1>_TO_<TYPE2>

- 禁止将“较大的”数据类型隐含地转换为“较小的”数据类型使用，当从较大数据类型转为较小数据类型时，有可能丢失信息。
- 如果被转换的值超出目标数据类型的存储范围，则这个数的高字节将被忽略。例如将 INT 类型转换为 BYTE 类型，或者将 DINT 类型转换为 WORD 类型。
- <TYPE>_TO_STRING 的转换中，字符串是从左边开始生成的。如果定义的字符串长度小于<TYPE>的长度，右边部分会被截去。

数据类型转换指令列表

表 4-7-1 列出了所有的数据类型转换指令，本小节讲述数据类型转换指令。

表 4-7-1 数据类型转换指令表

BOOL_TO_<TYPE>	BYTE_TO_<TYPE>	DATE_TO_<TYPE>	DINT_TO_<TYPE>
BOOL_TO_BYTE	BYTE_TO_BOOL	DATE_TO_BOOL	DINT_TO_BOOL
BOOL_TO_DATE	BYTE_TO_DATE	DATE_TO_BYTE	DINT_TO_BYTE
BOOL_TO_DINT	BYTE_TO_DINT	DATE_TO_DINT	DINT_TO_DATE
BOOL_TO_DT	BYTE_TO_DT	DATE_TO_DT	DINT_TO_DT
BOOL_TO_DWORD	BYTE_TO_DWORD	DATE_TO_DWORD	DINT_TO_DWORD
BOOL_TO_INT	BYTE_TO_INT	DATE_TO_INT	DINT_TO_INT
BOOL_TO_REAL	BYTE_TO_REAL	DATE_TO_REAL	DINT_TO_REAL
BOOL_TO_SINT	BYTE_TO_SINT	DATE_TO_SINT	DINT_TO_SINT
BOOL_TO_STRING	BYTE_TO_STRING	DATE_TO_STRING	DINT_TO_STRING
BOOL_TO_TIME	BYTE_TO_TIME	DATE_TO_TIME	DINT_TO_TIME
BOOL_TO_TOD	BYTE_TO_TOD	DATE_TO_TOD	DINT_TO_TOD
BOOL_TO_UDINT	BYTE_TO_UDINT	DATE_TO_UDINT	DINT_TO_UDINT
BOOL_TO_UINT	BYTE_TO_UINT	DATE_TO_UINT	DINT_TO_UINT
BOOL_TO_USINT	BYTE_TO_USINT	DATE_TO_USINT	DINT_TO_USINT
BOOL_TO_WORD	BYTE_TO_WORD	DATE_TO_WORD	DINT_TO_WORD
DT_TO_<TYPE>	**DWORD_TO_<TYPE>**	**INT_TO_<TYPE>**	**WORD_TO_<TYPE>**
DT_TO_BOOL	DWORD_TO_BOOL	INT_TO_BOOL	WORD_TO_BOOL
DT_TO_BYTE	DWORD_TO_BYTE	INT_TO_BYTE	WORD_TO_BYTE
DT_TO_DATE	DWORD_TO_DATE	INT_TO_DATE	WORD_TO_DATE
DT_TO_DINT	DWORD_TO_DINT	INT_TO_DINT	WORD_TO_DINT
DT_TO_DWORD	DWORD_TO_DT	INT_TO_DT	WORD_TO_DT
DT_TO_INT	DWORD_TO_INT	INT_TO_DWORD	WORD_TO_DWORD
DT_TO_REAL	DWORD_TO_REAL	INT_TO_REAL	WORD_TO_INT
DT_TO_SINT	DWORD_TO_SINT	INT_TO_SINT	WORD_TO_REAL
DT_TO_STRING	DWORD_TO_STRING	INT_TO_STRING	WORD_TO_SINT
DT_TO_TIME	DWORD_TO_TIME	INT_TO_TIME	WORD_TO_STRING
DT_TO_TOD	DWORD_TO_TOD	INT_TO_TOD	WORD_TO_TIME
DT_TO_UDINT	DWORD_TO_UDINT	INT_TO_UDINT	WORD_TO_TOD
DT_TO_UINT	DWORD_TO_UINT	INT_TO_UINT	WORD_TO_UDINT
DT_TO_USINT	DWORD_TO_USINT	INT_TO_USINT	WORD_TO_UINT
DT_TO_WORD	DWORD_TO_WORD	INT_TO_WORD	WORD_TO_USINT
REAL_TO_<TYPE>	**SINT_TO_<TYPE>**	**STRING_TO_<TYPE>**	**TIME_TO_<TYPE>**
REAL_TO_BOOL	SINT_TO_BOOL	STRING_TO_BOOL	TIME_TO_BOOL
REAL_TO_BYTE	SINT_TO_BYTE	STRING_TO_BYTE	TIME_TO_BYTE
REAL_TO_DATE	SINT_TO_DATE	STRING_TO_DATE	TIME_TO_DATE
REAL_TO_DINT	SINT_TO_DINT	STRING_TO_DINT	TIME_TO_DINT
REAL_TO_DT	SINT_TO_DT	STRING_TO_DT	TIME_TO_DT
REAL_TO_DWORD	SINT_TO_DWORD	STRING_TO_DWORD	TIME_TO_DWORD
REAL_TO_INT	SINT_TO_INT	STRING_TO_INT	TIME_TO_INT
REAL_TO_SINT	SINT_TO_REAL	STRING_TO_REAL	TIME_TO_REAL
REAL_TO_STRING	SINT_TO_STRING	STRING_TO_SINT	TIME_TO_SINT
REAL_TO_TIME	SINT_TO_TIME	STRING_TO_TIME	TIME_TO_STRING
REAL_TO_TOD	SINT_TO_TOD	STRING_TO_TOD	TIME_TO_TOD
REAL_TO_UDINT	SINT_TO_UDINT	STRING_TO_UDINT	TIME_TO_UDINT
REAL_TO_UINT	SINT_TO_UINT	STRING_TO_UINT	TIME_TO_UINT
REAL_TO_USINT	SINT_TO_USINT	STRING_TO_USINT	TIME_TO_USINT
REAL_TO_WORD	SINT_TO_WORD	STRING_TO_WORD	TIME_TO_WORD
TOD_TO_<TYPE>	**UDINT_TO_<TYPE>**	**UINT_TO_<TYPE>**	**USINT_TO_<TYPE>**
TOD_TO_BOOL	UDINT_TO_BOOL	UINT_TO_BOOL	USINT_TO_BOOL
TOD_TO_BYTE	UDINT_TO_BYTE	UINT_TO_BYTE	USINT_TO_BYTE
TOD_TO_DATE	UDINT_TO_DATE	UINT_TO_DATE	USINT_TO_DATE
TOD_TO_DINT	UDINT_TO_DINT	UINT_TO_DINT	USINT_TO_DINT
TOD_TO_DT	UDINT_TO_DT	UINT_TO_DT	USINT_TO_DT
TOD_TO_DWORD	UDINT_TO_DWORD	UINT_TO_DWORD	USINT_TO_DWORD
TOD_TO_INT	UDINT_TO_INT	UINT_TO_INT	USINT_TO_INT
TOD_TO_REAL	UDINT_TO_REAL	UINT_TO_REAL	USINT_TO_REAL
TOD_TO_SINT	UDINT_TO_SINT	UINT_TO_SINT	USINT_TO_SINT
TOD_TO_STRING	UDINT_TO_STRING	UINT_TO_STRING	USINT_TO_STRING
TOD_TO_TIME	UDINT_TO_TIME	UINT_TO_TIME	USINT_TO_TIME
TOD_TO_UDINT	UDINT_TO_TOD	UINT_TO_TOD	USINT_TO_TOD
TOD_TO_UINT	UDINT_TO_UINT	UINT_TO_UDINT	USINT_TO_UDINT
TOD_TO_USINT	UDINT_TO_USINT	UINT_TO_USINT	USINT_TO_UINT
TOD_TO_WORD	UDINT_TO_WORD	UINT_TO_WORD	USINT_TO_WORD

4.7.1 BOOL_TO_<TYPE>——布尔类型转换指令

- **功能**：把布尔数据类型转换为其它数据类型。
- **输入/输出数据类型**（参见表 4-7-1）：

输出为数字类型时，如果输入是 TRUE，则输出为 1，如果输入是 FALSE，则输出为 0；输出为字符串类型时，如果输入是 TRUE，则输出字符串'TRUE'，如果输入是 FALSE，则输出为字符串'FALSE'。

✧ **指令使用举例**

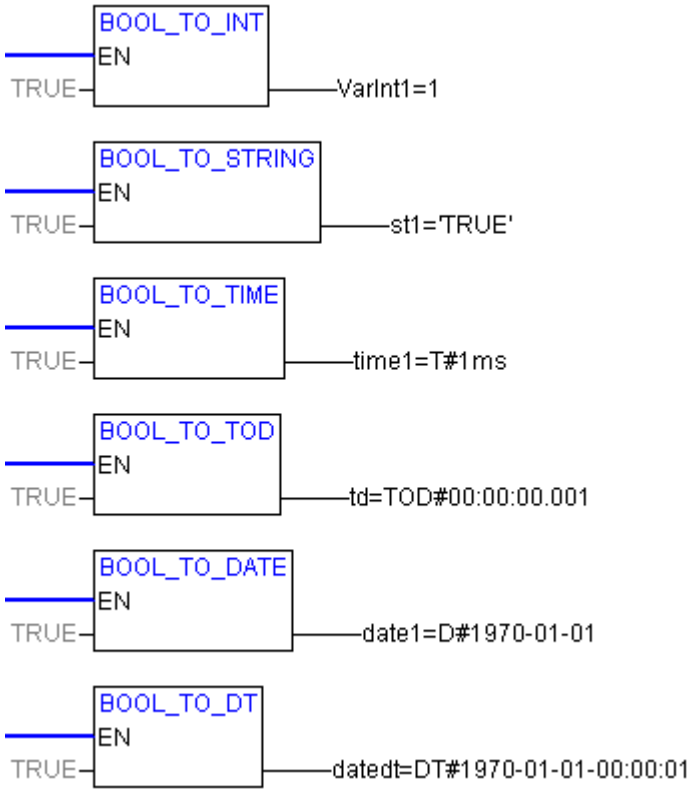
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	VarInt1		INT		
0002	st1		STRING		
0003	time1		TIME		
0004	td		TOD		
0005	date1		DATE		
0006	datedt		DT		

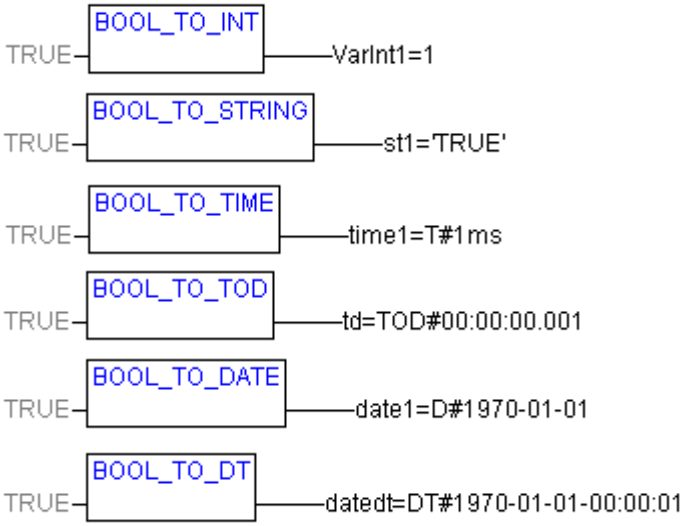
编程语言 | **程 序（部 分）**

梯形图（LD）



结构化文本（ST）

```
VarInt1:=BOOL_TO_INT(TRUE);     (*结果为 1*)
st1:=BOOL_TO_STRING(TRUE);      (*结果为'TRUE'*)
time1:=BOOL_TO_TIME(TRUE);      (*结果为 T#1ms*)
td:=BOOL_TO_TOD(TRUE);     (*结果为 TOD#00:00:00.001*)
date1:=BOOL_TO_DATE(TRUE);      (*结果为 D#1970-01-01*)
datedt :=BOOL_TO_DT(TRUE);     (*结果为 DT#1970-01-01-00:00:01*)
```

指令列表（IL）	LD TRUE BOOL_TO_INT ST VarInt1 (*结果为 1*) LD TRUE BOOL_TO_STRING ST st1 (*结果为'TRUE'*) LD TRUE BOOL_TO_TIME ST time1 (*结果为 T#1ms*) LD TRUE BOOL_TO_TOD ST td (*结果为 TOD#00:00:00.001*) LD TRUE BOOL_TO_DATE ST date1 (*结果为 D#1970-01-01*) LD TRUE BOOL_TO_DT ST datedt (*结果为 DT#1970-01-01-00:00:01*)
功能块（FBD）	

4.7.2 BYTE_TO_<TYPE>——字节类型转换指令

- **功能**：把字节类型转换为其他数据类型。
- **输入/输出数据类型**（参见表 4-7-1）：

当 BYTE_TO_BOOL 时，输入不等于 0 时输出为 TRUE，当输入等于 0 时输出为 FALSE；
当 BYTE_TO_TIME、BYTE_TO_TOD 时，输入将以毫秒值进行转换；
当 BYTE_TO_DATE、BYTE_TO_DT 时，输入将以秒值进行转换。

✧ 指令使用举例

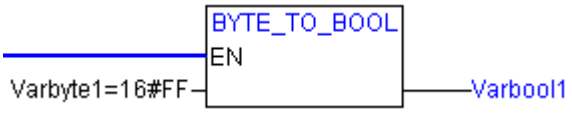
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

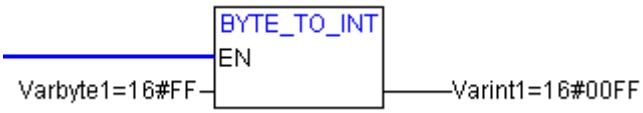
	名称	地址	类型	初始值	注释
0001	Varbool1		BOOL		
0002	Varbyte1		BYTE		
0003	Varint1		INT		
0004	Vartime1		TIME		
0005	Vardt1		DT		
0006	Varreal1		REAL		
0007	Varstring1		STRING		

编程语言 | **程 序（部 分）**

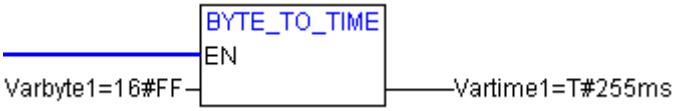
梯形图（LD）



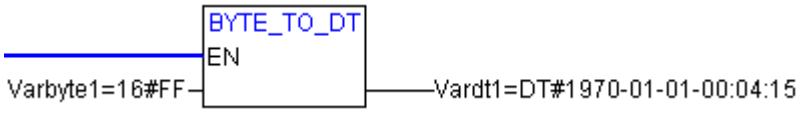
(*结果为 TRUE *)



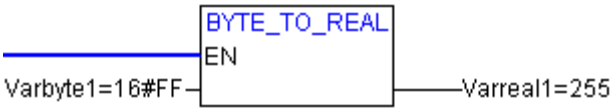
(*结果为 16# 00FF *)



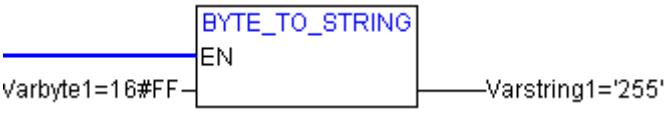
(*结果为 T#255ms *)



(*结果为 DT#1977-01-01-00:04:05 *)



(*结果为 255 *)



(*结果为字符串'255'*)

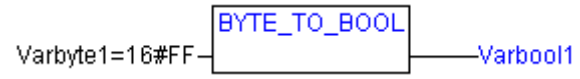
结构化文本（ST）

```
Varbyte1:=16#FF                                  (*Varbyte1 取值*)
Varbool1:=BYTE_TO_BOOL(Varbyte1);                (*结果为 TRUE *)
Varint1:=BYTE_TO_INT(Varbyte1);                  (*结果为 16# FF *)
Vartime1:=BYTE_TO_TIME(Varbyte1);                (*结果为 T#255ms *)
Vardt1:=BYTE_TO_DT(Varbyte1);
(*结果为 DT#1977-01-01-00:04:05 *)
Varreal1:=BYTE_TO_REAL(Varbyte1);                (*结果为 255 *)
Varstring1:=BYTE_TO_STRING(Varbyte1); (*结果为字符串'255'*)
```

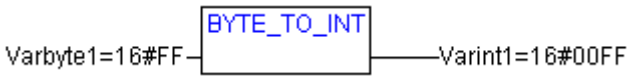
指令列表（IL）

```
LD      16#FF
ST      Varbyte1                    (*Varbyte1 取值*)
LD      Varbyte1
BYTE_TO_BOOL
ST      Varbool1                    (*结果为 TRUE *)
LD      Varbyte1
BYTE_TO_INT
ST      Varint1                     (*结果为 16# FF *)
LD      Varbyte1
BYTE_TO_TIME
ST      Vartime1                    (*结果为 T#255ms *)
LD      Varbyte1
BYTE_TO_DT
ST      Vardt1          (*结果为 DT#1970-01-01-00:04:05 *)
LD      Varbyte1
BYTE_TO_REAL
ST      Varreal1                    (*结果为 255 *)
LD      Varbyte1
BYTE_TO_STRING
ST      Varstring1                  (*结果为字符串'255'*)
```

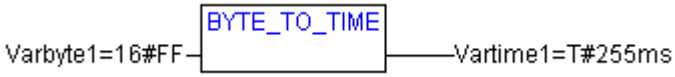
功能块（FBD）



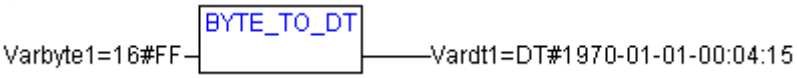
(*结果为 TRUE *)



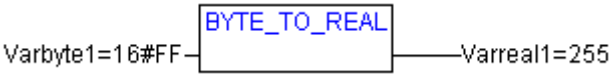
(*结果为 16# 00FF *)



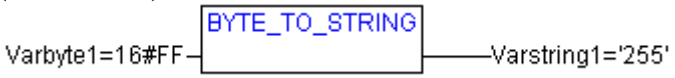
(*结果为 T#255ms *)



(*结果为 DT#1977-01-01-00:04:05 *)



(*结果为 255 *)



(*结果为字符串'255'*)

4.7.3 WORD_TO_<TYPE>——字类型转换指令

- **功能**：把字类型转换为其它数据类型。
- **输入/输出数据类型**（参见表 4-7-1）：

当 WORD_TO_BOOL 时，输入不等于 0 时输出为 TRUE，当输入等于 0 时输出为 FALSE；
当 WORD_TO_TIME、WORD_TO_TOD 时，输入将以毫秒值进行转换；
当 WORD_TO_DATE、WORD_TO_DT 时，输入将以秒值进行转换。

✧ 指令使用举例

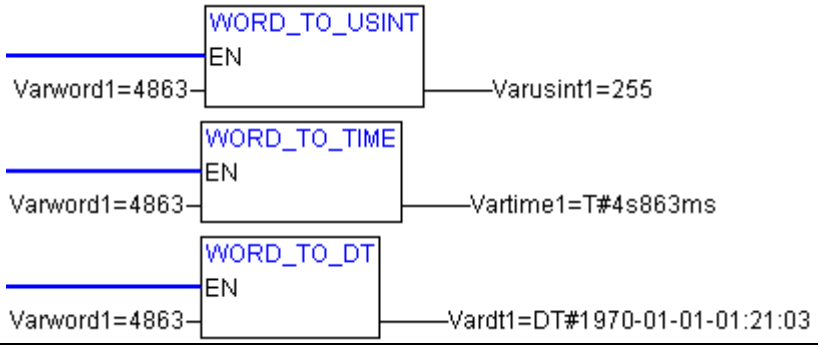
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	Varusint1		USINT		
0002	Varword1		WORD		
0003	Vartime1		TIME		
0004	Vardt1		DT		

编程语言 | **程 序（部 分）**

梯形图（LD）



结构化文本（ST）

```
Varword1:=4863;                                    (*Varword1 取值*)
Varusint1:=WORD_TO_USINT(Varword1);                (*结果 255 *)
```

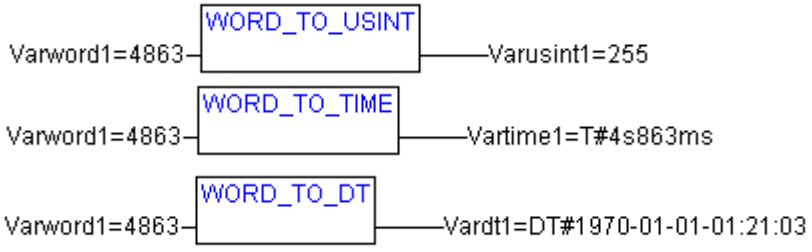
说明：如果将整数 4863（十六进制为 16#12FF）保存为 USINT 型变量，则会丢失高位数据，只显示低位数据 255（十六进制为 16#FF）。

```
Vartime1:=WORD_TO_TIME(Varword1);          (*结果 T#4s863ms*)
Vardt1:=WORD_TO_DT(Varword1);
(*结果 DT#1970-01-01-01:21:03 *)
```

指令列表（IL）

```
LD      4863
ST      Varword1                           (*Varword1 取值*)
LD      Varword1
WORD_TO_USINT
ST      Varusint1                          (*结果 255 *)
LD      Varword1
WORD_TO_TIME
ST      Vartime1                           (*结果 T#4s863ms*)
LD      Varword1
WORD_TO_DT
ST      Vardt1                 (*结果 DT#1970-01-01-01:21:03 *)
```

功能块（FBD）



4.7.4 DWORD_TO_<TYPE>——双字类型转换指令

- **功能**：把双字类型转换为其它数据类型。
- **输入/输出数据类型**（参见表 4-7-1）：

当 DWORD_TO_BOOL 时，输入不等于 0 时输出为 TRUE，当输入等于 0 时输出为 FALSE；
当 DWORD_TO_TIME、DWORD_TO_TOD 时，输入将以毫秒值进行转换；
当 DWORD_TO_DATE、DWORD_TO_DT 时，输入将以秒值进行转换。

✧ **指令使用举例**

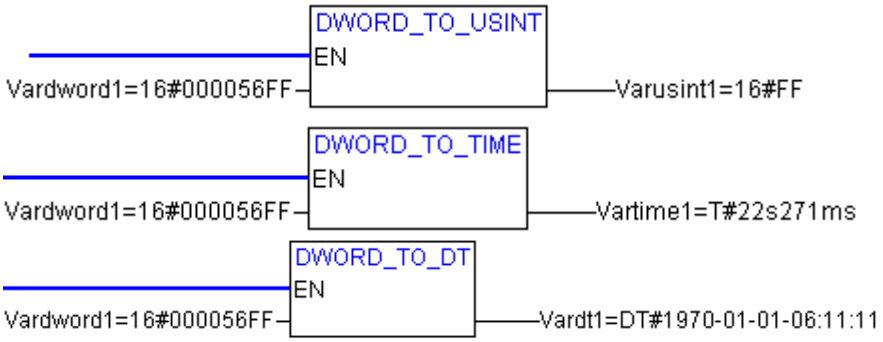
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	Varusint1		USINT		
0002	Vardword1		DWORD		
0003	Vartime1		TIME		
0004	Vardt1		DT		

编程语言 | **程 序（部 分）**

梯形图（LD）



结构化文本（ST）

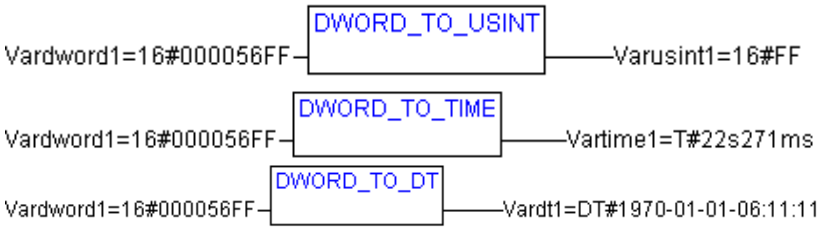
```
Varword1:=16#56FF;                          (*Varword1 取值*)
Varusint1:=DWORD_TO_USINT(Vardword1);       (*结果 255 *)
```

说明：如果将整数 16#56FF（十进制为 22271）保存为 USINT 型变量，则会丢失高位数据，只显示低位数据 255（十六进制为 16#FF）。

```
Vartime1:=DWORD_TO_TIME(Vardword1);     (*结果 T#22s271ms*)
Vardt1:=DWORD_TO_DT(Vardword1);
(*结果 DT#1970-01-01-06:11:11 *)
```

指令列表（IL）

```
LD      16#56FF
ST      Vardword1                   (*Vardword1 取值*)
LD      Vardword1
DWORD_TO_USINT
ST      Varusint1                   (*结果 255 *)
LD      Vardword1
DWORD_TO_TIME
ST      Vartime1                    (*结果 T#22s271ms*)
LD      Vardword1
DWORD_TO_DT
ST      Vardt1              (*结果 DT#1970-01-01-06:11:11 *)
```

功 能 块（FBD）	

4.7.5 SINT_TO_<TYPE>——单整型转换指令

- **功能**：把单整型转换为其它数据类型。
- **输入/输出数据类型**（参见表 4-7-1）：

当 SINT_TO_BOOL 时，输入不等于 0 时输出为 TRUE，当输入等于 0 时输出为 FALSE；
当 SINT_TO_TIME、SINT _TO_TOD 时，输入将以毫秒值进行转换；
当 SINT_TO_DATE、SINT _TO_DT 时，输入将以秒值进行转换。

✧ **指令使用举例**

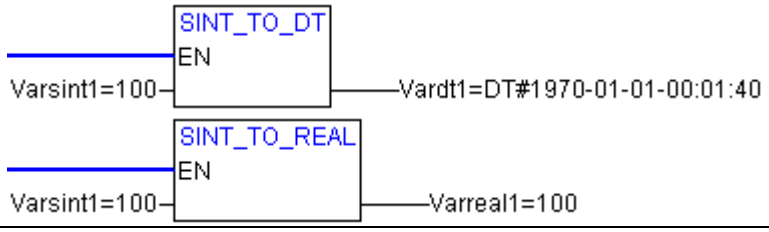
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	Varsint1		SINT		
0002	Vardt1		DT		
0003	Varreal1		REAL		

编程语言 | **程 序（部 分）**

梯形图（LD）



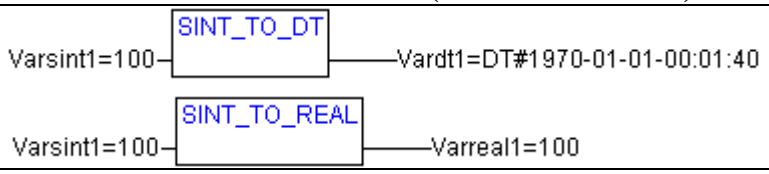
结构化文本（ST）

```
Varsint1:=100;                          (*Varsint1 取值*)
Vardt1:=SINT_TO_DT(Varsint1);           (*结果 DT#1970-01-01-00:01:40 *)
Varreal1:=SINT_TO_REAL(Varsint1);       (*结果为 100 .0*)
```

指令列表（IL）

```
LD      100
ST      Varsint1                        (*Varsint1 取值*)
LD      Varsint1
SINT_TO_DT
ST      Vardt1                          (*结果 DT#1970-01-01-00:01:40 *)
LD      Varsint1
SINT_TO_REAL
ST      Varreal1                        (*结果 Varrea1 为 100.0*)
```

功能块（FBD）



4.7.6 USINT_TO_<TYPE>——无符号单整型转换指令

- **功能**：把无符号单整型转换为其它数据类型。
- **输入/输出数据类型**（参见表 4-7-1）：

当 USINT_TO_BOOL 时，输入不等于 0 时输出为 TRUE，当输入等于 0 时输出为 FALSE；
当 USINT_TO_TIME、USINT _TO_TOD 时，输入将以毫秒值进行转换；
当 USINT_TO_DATE、USINT _TO_DT 时，输入将以秒值进行转换。

✧ **指令使用举例**

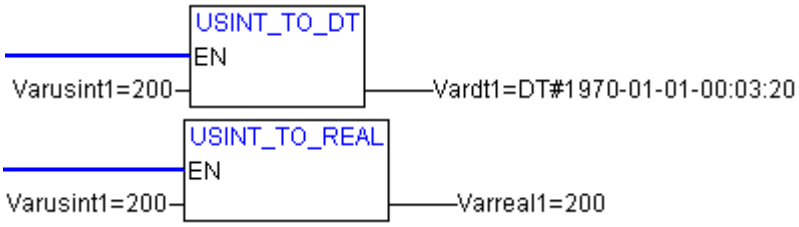
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	Varusint1		USINT		
0002	Vardt1		DT		
0003	Varreal1		REAL		

编程语言 | **程 序（部 分）**

梯形图（LD）



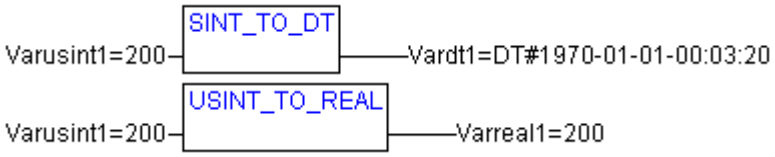
结构化文本（ST）

```
Varusint1:=200;                                    (*Varusint1 取值*)
Vardt1:=USINT_TO_DT(Varusint1);(*结果 DT#1970-01-01-00:03:20 *)
Varreal1:=USINT_TO_REAL(Varusint1);          (*结果为 200 .0*)
```

指令列表（IL）

```
LD      200
ST      Varusint1                    (*Varusint1 取值*)
LD      Varusint1
USINT_TO_DT
ST      Vardt1                       (*结果 DT#1970-01-01-00:03:20 *)
LD      Varusint1
USINT_TO_REAL
ST      Varreal1                     (*结果 Varreal1 为 200.0*)
```

功能块（FBD）



4.7.7 INT_TO_<TYPE>——整数类型转换指令

- **功能**：把整型数据类型转换为其它数据类型。
- **输入/输出数据类型**（参见表 4-7-1）：

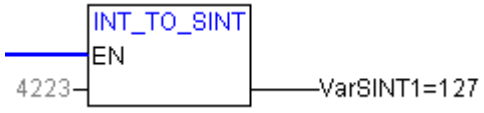
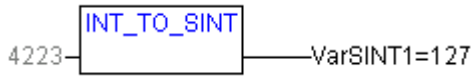
当 INT_TO_BOOL 时，输入不等于 0 时输出为 TRUE，当输入等于 0 时输出为 FALSE；
当 INT_TO_TIME、INT _TO_TOD 时，输入将以毫秒值进行转换；
当 INT_TO_DATE、INT _TO_DT 时，输入将以秒值进行转换。

✧ 指令使用举例

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONS

	名称	地址	类型	初始值	注释
0001	VarSINT1		SINT		
0002	VarREAL1		REAL		

编程语言	程 序（部 分）
梯形图（LD）	
结构化文本（ST）	VarSINT1 := INT_TO_SINT(4223); (*结果 VarSINT1 为 127 *) 说明：如果将整数 4223（十六进制为 16#107F）保存为 SINT 型变量，则会丢失高位数据，只显示低位数据 127（十六进制为 16#7F）。
指令列表（IL）	LD 2 INT_TO_REAL ST VarREAL1 (*结果 VarREAL1 为 2.0*)
功能块（FBD）	

4.7.8 UINT_TO_<TYPE>——无符号整数类型转换指令

➢ **功能**：无符号整数类型转换为其它数据类型。

➢ **输入/输出数据类型**（参见表 4-7-1）：

当 UINT_TO_BOOL 时，输入不等于 0 时输出为 TRUE，当输入等于 0 时输出为 FALSE；

当 UINT_TO_TIME、UINT _TO_TOD 时，输入将以毫秒值进行转换；

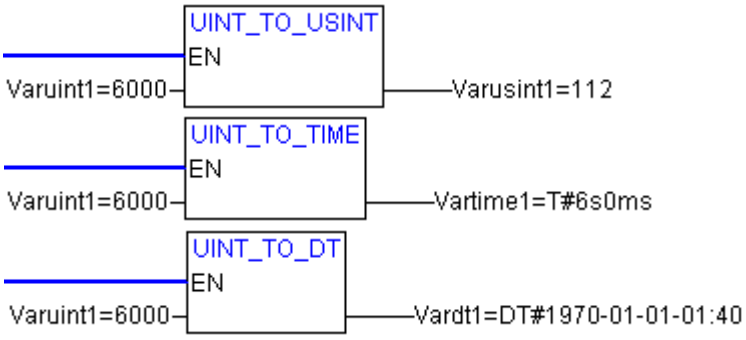
当 UINT_TO_DATE、UINT _TO_DT 时，输入将以秒值进行转换。

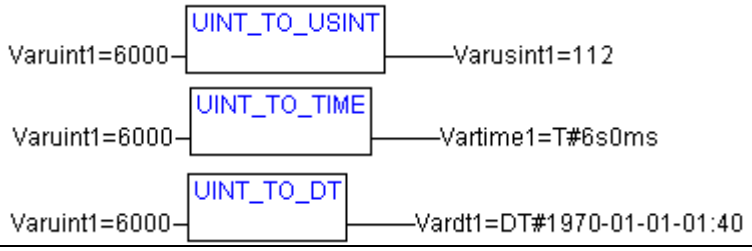
✧ 指令使用举例

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	Varuint1		UINT		
0002	Varusint1		USINT		
0003	Vartime1		TIME		
0004	Vardt1		DT		

编程语言	程 序（部 分）
梯形图（LD）	

结构化文本（ST）	Varuint1:=6000; Varusint1:=UINT_TO_USINT(Varuint1); 说明：如果将整数 6000（十六进制为 16#1770）保存为 SINT 型变量，则会丢失高位数据，只显示低位数据 112（十六进制为 16#70）。 Vartime1:=UINT_TO_TIME(Varuint1); Vardt1:=UINT_TO_DT(Varuint1);
指令列表（IL）	LD 6000 ST Varuint1 (*Varuint1 取值*) LD Varuint1 UINT_TO_USINT ST Varusint1 (*结果 112*) LD Varuint1 UINT_TO_TIME ST Vartime1 (*结果 T#6s0ms*) LD Varuint1 UINT_TO_DT ST Vardt1 (*结果 DT#1970-01-01-01:40:00 *)
功能块（FBD）	

4.7.9 DINT_TO_<TYPE>——双整数类型转换指令

- **功能**：双整数类型类型转换为其它数据类型。
- **输入/输出数据类型**（参见表 4-7-1）：

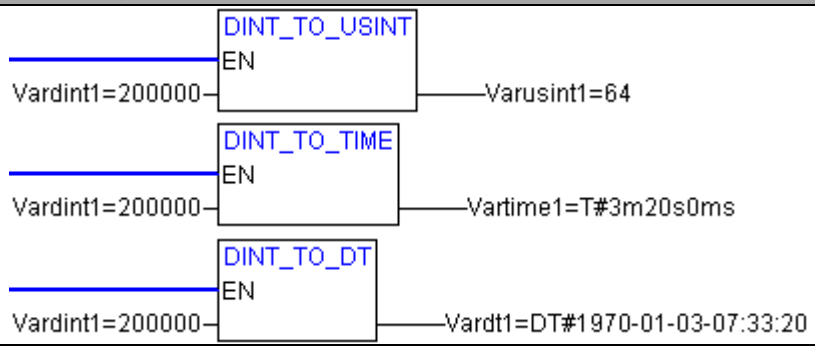
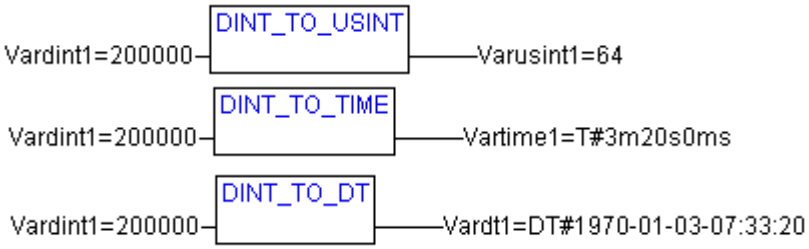
当 DINT_TO_BOOL 时，输入不等于 0 时输出为 TRUE，当输入等于 0 时输出为 FALSE；
当 DINT_TO_TIME、DINT _TO_TOD 时，输入将以毫秒值进行转换；
当 DINT_TO_DATE、DINT _TO_DT 时，输入将以秒值进行转换。

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	Vardint1		DINT		
0002	Varusint1		USINT		
0003	Vartime1		TIME		
0004	Vardt1		DT		

编程语言	程　序（部 分）
梯形图（LD）	
结构化文本（ST）	Vardint1:=200000; Varusint1:=DINT_TO_USINT(Vardint1);　　(*结果 64*) 说明：如果将整数 200000（十六进制为 16#30D40）保存为 USINT 型变量，则会丢失高位数据，只显示低位数据 64（十六进制为 16#40）。 Vartime1:=DINT_TO_TIME(Vardint1);　(*结果 T#3m20s0ms*) Vardt1:=DINT_TO_DT(Vardint1);　　(*结果 DT#1970-01-03-07:33:20 *)
指令列表（IL）	LD　200000 ST　Vardint1　　(*Vardint1 取值*) LD　Vardint1 DINT_TO_USINT ST　Varusint1　　(*结果 64*) LD　Vardint1 DINT_TO_TIME ST　Vartime1　　(*结果 T#3m20s0ms*) LD　Vardint1 DINT_TO_DT ST　Vardt1　　(*结果 DT#1970-01-03-07:33:20 *)
功能块（FBD）	

4.7.10　UDINT_TO_<TYPE>——无符号长整数类型转换指令

- **功能**：无符号双整数类型转换为其它数据类型。
- **输入/输出数据类型**（参见表 4-7-1）：

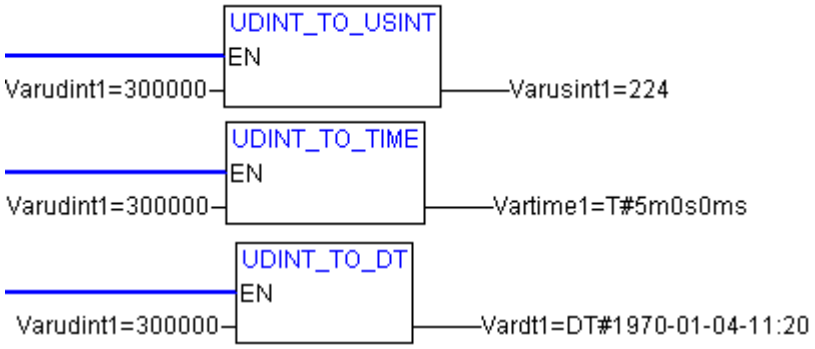
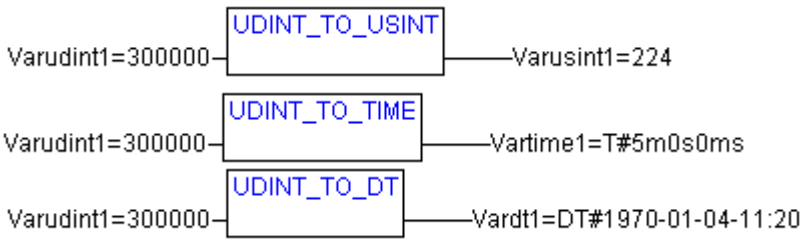
当 UDINT_TO_BOOL 时，输入不等于 0 时输出为 TRUE，当输入等于 0 时输出为 FALSE；
当 UDINT_TO_TIME、UDINT _TO_TOD 时，输入将以毫秒值进行转换；
当 UDINT_TO_DATE、UDINT _TO_DT 时，输入将以秒值进行转换。

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	Varudint1		UDINT		
0002	Varusint1		USINT		
0003	Vartime1		TIME		
0004	Vardt1		DT		

编程语言	程　序（部 分）
梯形图（LD）	
结构化文本（ST）	Varudint1:=300000; Varusint1:= UDINT_TO_USINT(Varudint1);　　(*结果 224*) 说明：如果将整数 300000（十六进制为 16#493E0）保存为 USINT 型变量，则会丢失高位数据，只显示低位数据 224（十六进制为 16#E0）。 Vartime1:=UDINT_TO_TIME(Varudint1);　　(*结果 T#5m0s0ms*) Vardt1:=UDINT_TO_DT(Varudint1); (*结果 DT#1970-01-04-11:20:00 *)
指令列表（IL）	LD　　300000 ST　　Varudint1　　(*Varudint1 取值*) LD　　Varudint1 UDINT_TO_USINT ST　　Varusint1　　(*结果 224*) LD　　Varudint1 UDINT_TO_TIME ST　　Vartime1　　(*结果 T#5m0s0ms*) LD　　Varudint1 UDINT_TO_DT ST　　Vardt1　　(*结果 DT#1970-01-04-11:20:00 *)
功能块（FBD）	

4.7.11　REAL_TO_<TYPE>——实数类型转换指令

- **功能**：把浮点数转换为其它类型数据。把浮点数转换为其它类型数据时，先将值四舍五入成整数值，然后转成新的变量类型。
- **输入/输出数据类型**（参见表 4-7-1）：

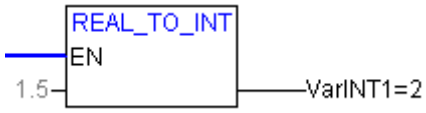
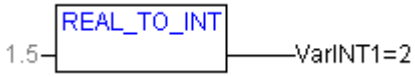
当 REAL_TO_BOOL 时，输入不等于 0 时输出为 TRUE，当输入等于 0 时输出为 FALSE；
当 REAL_TO_TIME、REAL_TO_TOD 时，输入将以毫秒值进行转换；
当 REAL_TO_DATE、REAL_TO_DT 时，输入将以秒值进行转换。

✧ 指令使用举例

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	Varint1		INT		
0002	Varint2		INT		
0003	Varint3		INT		
0004	Varint4		INT		

编程语言	程 序（部 分）
梯形图（LD）	
结构化文本（ST）	VarINT1:= REAL_TO_INT(1.5); (*结果 VarINT1 为 2*) VarINT2:= REAL_TO_INT(1.4); (*结果 VarINT2 为 1*) VarINT3:= REAL_TO_INT(-1.5); (*结果 VarINT3 为-2*) VarINT4:= REAL_TO_INT(-1.4); (*结果 VarINT4 为-1*)
指令列表（IL）	LD 2.7 REAL_TO_INT ST VarINT1 (*结果 VarINT1 为 3*)
功能块（FBD）	

4.7.12 TIME_TO_<TYPE>——时间类型转换指令

- **功能**：把时间型数据转换为其它类型数据，时间在内部以毫秒为单位存储成 DWORD 类型（对于 TIME_OF_DAY 变量从凌晨 00：00 开始）。
- **输入/输出数据类型**（参见表 4-7-1）：

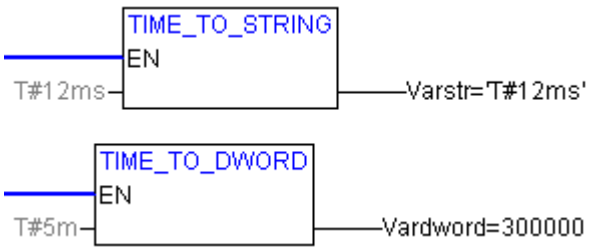
当 TIME_TO_BOOL 时，输入不等于 0 时输出为 TRUE；
当输入等于 0 时输出为 FALSE。

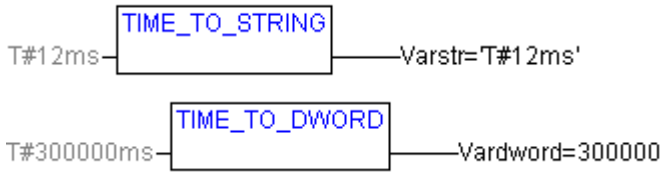
✧ 指令使用举例

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CON

	名称	地址	类型	初始值	注释
0001	Varstr		STRING		
0002	Vardword		DWORD		

编程语言	程 序（部 分）
梯形图（LD）	

结构化文本（ST）	Varstr:=TIME_TO_STRING(T#12ms);　(*结果为‘T#12ms’*) Vardword:=TIME_TO_DWORD(T#5m);　(*结果为 300000*)
指令列表（IL）	LD　T#12ms TIME_TO_STRING ST　Varstr　(*结果为‘T#12ms’*) LD　T#300000ms TIME_TO_DWORD ST　Vardword　(*结果为 300000*)
功能块（FBD）	

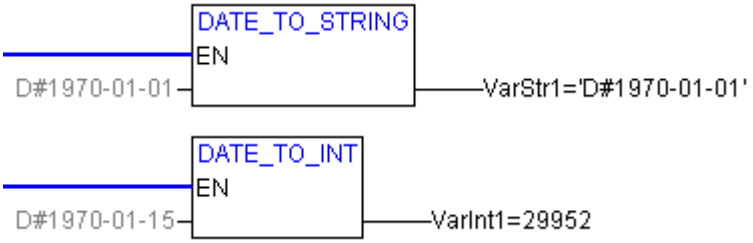
4.7.13　DATE_TO_<TYPE>——日期类型转换指令

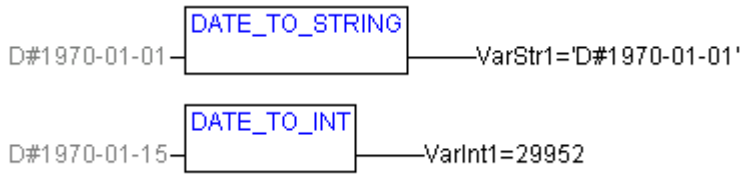
- **功能**：把日期型数据转换为其它类型数据，日期在内部以秒为单位存储，时间从 1970 年 1 月 1 日开始。
- **输入/输出数据类型**（参见表 4-7-1）：

当 DATE_TO_BOOL 时，输入不等于 0 时输出为 TRUE；
当输入等于 0 时输出为 FALSE。

✧ **指令使用举例**

变量定义

编程语言	**程　序（部 分）**
梯形图（LD）	
结构化文本（ST）	VarStr1:=DATE_TO_STRING(D#1970-01-01); (*结果为'D#1970-01-01' *) VarInt1:=DATE_TO_INT(D#1970-01-15);　(*结果为 29952*) 说明：将 D#1970-01-15（十进制 14×24×3600=1209600=16#127500）保存为 INT 型变量，则会丢失高位数据，只显示低位数据 16#7500,转换十进制数为 29952。
指令列表（IL）	LD　D#1970-01-01 DATE_TO_STRING ST　VarStr1　(*结果为‘TRUE’*) LD　D#1970-01-15 DATE_TO_INT ST　VarInt1　(*结果为 29952*)

功能块（FBD）	

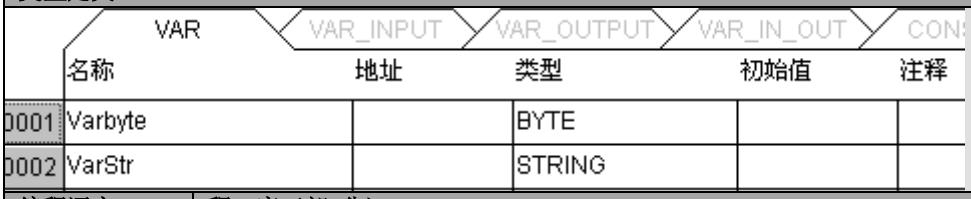
4.7.14 DT_TO_<TYPE>——日期时间类型转换指令

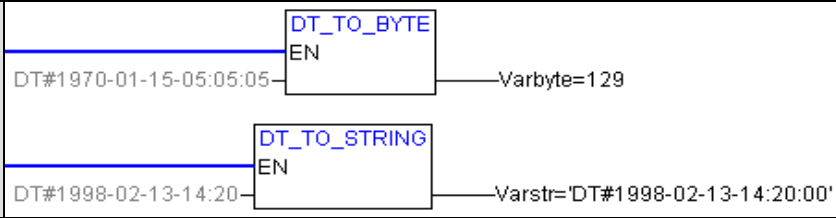
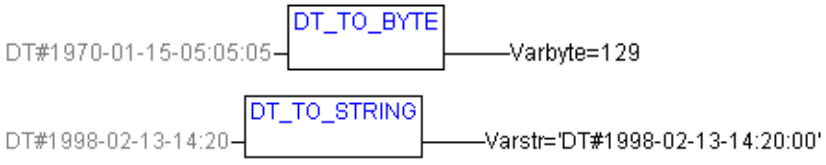
- **功能**：把日期时间型数据转换为其它类型数据，日期在内部以秒为单位存储，时间从1970 年 1 月 1 日开始。
- **输入/输出数据类型**（参见表 4-7-1）：

当 DT_TO_BOOL 时，输入不等于 0 时输出为 TRUE，当输入等于 0 时输出为 FALSE。

✧ **指令使用举例**

变量定义



编程语言	程 序（部 分）
梯形图（LD）	
结构化文本（ST）	Varbyte:=DT_TO_BYTE(DT#1970-01-15-05:05:05); (*结果 Varbyte 为 129*) 说明：将 DT#1970-01-15-05:05:05 转换成秒数，值为(((14*24+5)*60+5)*60+5)=1227905=16#12BC81，保存为 BYTE 型变量，则会丢失高 24 位数据，只显示低 8 位数据，16#81= 129。 Varstr:=DT_TO_STRING(DT#1998-02-13-14:20); (*结果 Varstr 为‘DT#1998-02-13-14:20’)
指令列表（IL）	LD DT#1970-01-15-05:05:05 DT_TO_BYTE ST Varbyte (*结果 Varbyte 为 129*) LD DT#1998-02-13-14:20 DT_TO_STRING ST Varstr (*结果 Varstr 为‘DT#1998-02-13-14:20’)
功能块（FBD）	

4.7.15 TOD_TO_<TYPE>——时间类型转换指令

- **功能**：把时间型数据转换为其它类型数据，日期在内部以毫秒为单位进行转化。
- **输入/输出数据类型**（参见表 4-7-1）：

当 TOD_TO_BOOL 时，输入不等于 0 时输出为 TRUE，当输入等于 0 时输出为 FALSE。

✧ **指令使用举例**

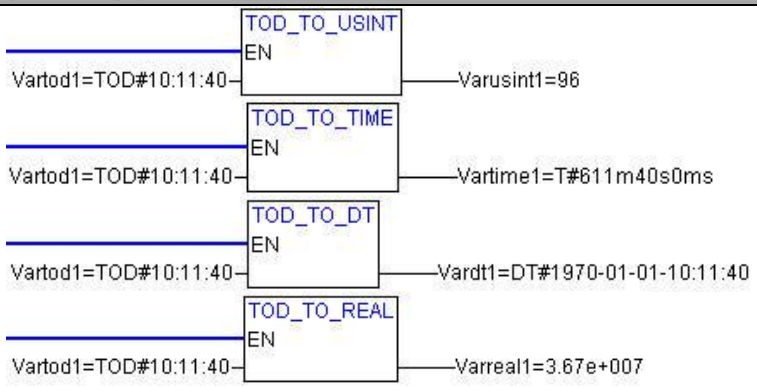
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	Vartod1		TOD		
0002	varbool1		BOOL		
0003	Varusint1		USINT		
0004	Vartime1		TIME		
0005	Vardt1		DT		
0006	Varreal1		REAL		

编程语言 | **程 序（部 分）**

梯形图（LD）

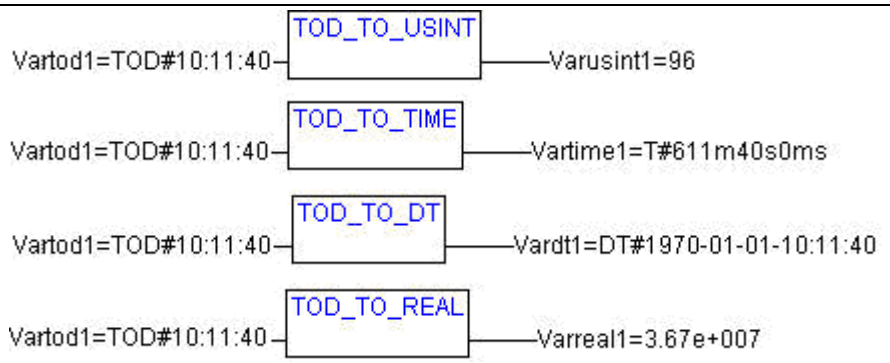


结构化文本（ST）

```
Vartod1:=TOD#10:11:40;                    (*Vartod1 取值*)
Varusint1:=TOD_TO_USINT(Vartod1);         (*结果为 96*)
Vartime1:=TOD_TO_TIME(Vartod1);   (*结果为 T#611m40s0ms*)
Vardt1:=TOD_TO_DT(Vartod1);
(*结果为 DT#1970-01-01-10:11:40*)
Varreal1:=TOD_TO_REAL(Vartod1);        (*结果为 3.67e+007*)
```

指令列表（IL）

```
LD      TOD#10:11:40
ST      Vartod1                          (*Vartod1 取值*)
LD      Vartod1
TOD_TO_USINT
ST      Varusint1                        (*结果为 96*)
LD      Vartod1
TOD_TO_TIME
ST      Vartime1                  (*结果为 T#611m40s0ms*)
LD      Vartod1
TOD_TO_DT
ST      Vardt1           (*结果为 DT#1970-01-01-10:11:40*)
LD      Vartod1
TOD_TO_REAL
ST      Varreal1                         (*结果为 3.67e+007*)
```

功能块（FBD）	

4.7.16 STRING_TO_<TYPE>——字符类型转换指令

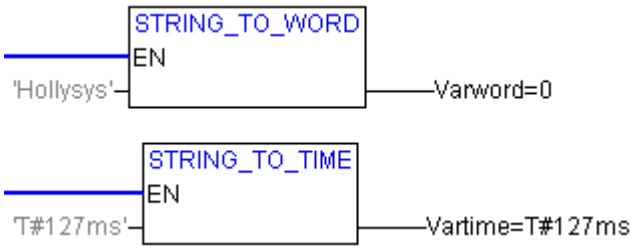
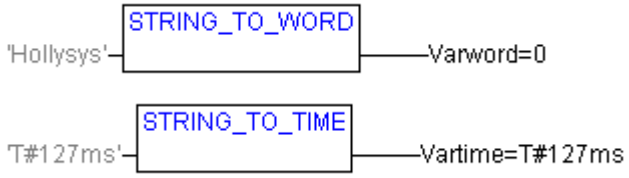
- **功能**：把字符串转换为其它类型数据，字符串型变量必须包含一个有效的目标变量值，否则转换结果为 0。
- **输入/输出数据类型**：（参见表 4-7-1）

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONS

	名称	地址	类型	初始值	注释
0001	Varword		WORD		
0002	Vartime		TIME		

编程语言	程　序（部 分）
梯形图（LD）	
结构化文本（ST）	Varword:=STRING_TO_WORD('Hollysys'); (*结果为 0*) Vartime:=STRING_TO_TIME('T#127ms'); (*结果为 T#127ms*)
指令列表（IL）	LD 'Hollysys' STRING_TO_WORD ST Varword (*结果 Varword 为 0*) LD 'T#127ms' STRING_TO_TIME ST Vartime (*结果 Vartime 为 T#127ms*)
功能块（FBD）	

4.7.17 TRUNC——截短转换指令

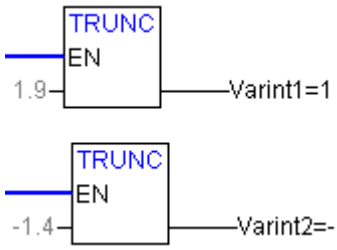
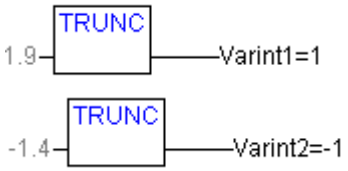
- **功能**：该指令将数据截去小数部分，只保留整数部分。
- **输入/输出数据类型**：输入为 REAL 型，输出为 INT、WORD、DWORD 型。

✧ 指令使用举例

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONS

	名称	地址	类型	初始值	注释
0001	Varint1		INT		
0002	Varint2		INT		

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	Varint1:=TRUNC(1.9);　　(*结果 Varint1 为 1*) Varint2:=TRUNC(-1.4);　　(*结果 Varint2 为-1*)
指令列表（IL）	LD　　1.9 TRUNC ST　　Varint1　　(*结果 Varint1 为 1*) LD　　-1.4 TRUNC ST　　Varint2　　(*结果 Varint2 为-1*)
功能块（FBD）	

提示：

- 当从较大数据类型转为较小数据类型时，有可能丢失信息。
- 本指令只是截取整数部分，如果想四舍五入取整，可以使用 REAL_TO_INT 指令。

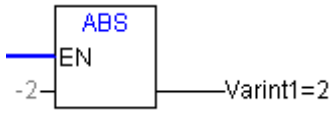
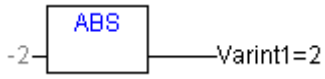
4.8 初等数学运算指令

4.8.1 ABS——绝对值指令

- **功能：**把输入数据的绝对值赋予输出变量。
- **输入/输出数据类型：**

输入数据类型	输出数据类型
INT	INT、WORD、DWORD、DINT、UINT、REAL
REAL	REAL
BYTE	INT、BYTE、WORD、DWORD、DINT、UINT、REAL
WORD	WORD、DWORD、DINT、REAL
DWORD	DWORD、DINT、REAL
SINT	INT、BYTE、WORD、DWORD、DINT、UINT、REAL
USINT	INT、BYTE、WORD、DWORD、DINT、UINT、REAL
UINT	WORD、DWORD、DINT、UINT、REAL
DINT	DWORD、DINT、REAL
UDINT	DWORD、DINT、UDINT、REAL

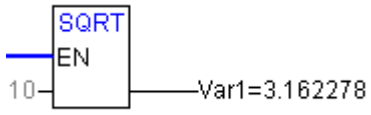
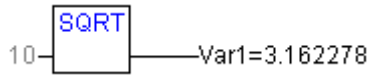
✧ 指令使用举例

变量定义	
	
编程语言	**程　序**
梯形图（LD）	
结构化文本（ST）	Varint1:=ABS(-2);　　(*结果 Varint1 为 2*)
指令列表（IL）	LD　-2 ABS ST　Varint1　　(*结果 Varint1 为 2*)
功能块（FBD）	

4.8.2 SQRT——平方根指令

- **功能**：对输入数据求平方根，输入数据为非负数。
- **输入数据类型**：BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT;
- **输出数据**：REAL。

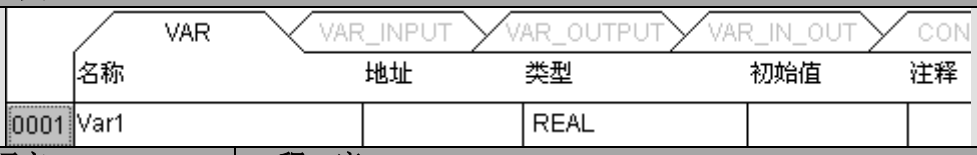
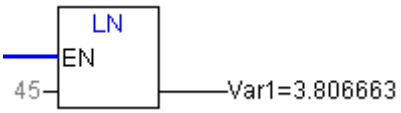
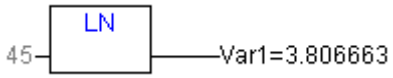
✧ 指令使用举例

变量定义	
	
编程语言	**程　序**
梯形图（LD）	
结构化文本（ST）	Var1:=SQRT(10); (*结果 Var1 为 3.162278*)
指令列表（IL）	LD　10 SQRT ST　Var1　　(*结果 Var1 为 3.162278*)
功能块（FBD）	

4.8.3 LN——自然对数指令

- **功能**：对输入数据求自然对数，输入数据必须为正数。
- **输入数据**：BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT;
- **输出数据**：REAL。

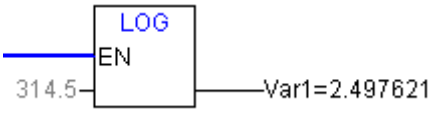
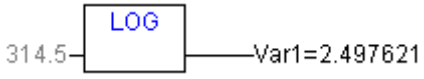
✧ 指令使用举例

变量定义	
	
编程语言	**程　序**
梯形图（LD）	
结构化文本（ST）	Var1:=LN(45);　(*结果 Var1 为 3.806663*)
指令列表（IL）	LD　45 LN ST　Var1　(*结果 Var1 为 3.806663*)
功能块（FBD）	

4.8.4　LOG——常用对数指令

- **功能**：对输入数据求以 10 为底的对数，输入数据必须为正数。
- **输入数据**：BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT；
- **输出数据**：REAL。

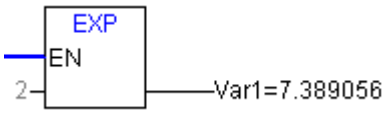
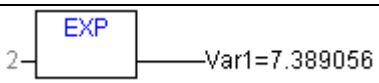
✧ 指令使用举例

变量定义	
	
编程语言	**程　序**
梯形图（LD）	
结构化文本（ST）	Var1:=LOG(314.5);　(*结果 Var1 为 2.497621 *)
指令列表（IL）	LD　314.5 LOG ST　Var1　(*结果 Var1 为 2.497621 *)
功能块（FBD）	

4.8.5　EXP——指数指令

- **功能**：以输入数据为指数的幂计算，即 $y = e^x$，其中 x 为输入，y 为输出。
- **输入数据**：BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT；
- **输出数据**：REAL。

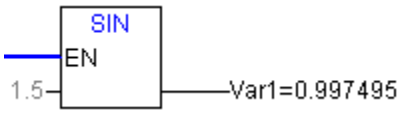
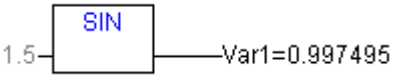
✧ 指令使用举例

变量定义	
	
编程语言	**程　序**
梯形图（LD）	
结构化文本（ST）	Var1:=EXP(2);　　（*结果 Var1 为 7.389056*）
指令列表（IL）	LD　2 EXP ST　Var1　　（*结果 Var1 为 7.389056*）
功能块（FBD）	

4.8.6　SIN——正弦指令

- **功能**：求输入数据的正弦值，输入数据以弧度表示。$弧度(rad) = 角度 * \frac{\pi}{180^\circ}$
- **输入数据**：BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT;
- **输出数据**：REAL。

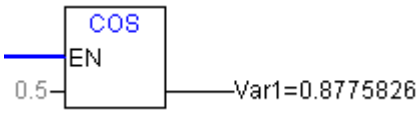
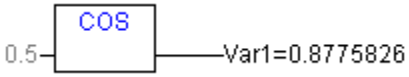
✧ 指令使用举例

变量定义	
	
编程语言	**程　序**
梯形图（LD）	
结构化文本（ST）	Var1:=SIN(1.5);　　(*结果 Var1 为 0.997495 *)
指令列表（IL）	LD　1.5 SIN ST　Var1　　(*结果 Var1 为 0.997495 *)
功能块（FBD）	

4.8.7　COS——余弦指令

- **功能**：求输入数据的余弦值，输入数据以弧度表示。（弧度公式参见 SIN 指令）
- **输入数据**：BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT;
- **输出数据**：REAL。

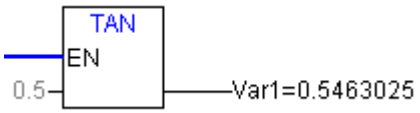
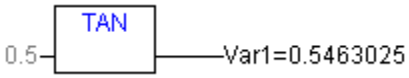
✧ 指令使用举例

<table>
<tr><td colspan="2">变量定义
 
</td></tr>
<tr><td>编程语言</td><td>程　序</td></tr>
<tr><td>梯形图（LD）</td><td> 
</td></tr>
<tr><td>结构化文本（ST）</td><td>Var1:=COS(0.5);　　(*结果 Var1 为 0.8775826 *)</td></tr>
<tr><td>指令列表（IL）</td><td>LD　0.5
COS
ST　Var1　　(*结果 Var1 为 0.8775826 *)</td></tr>
<tr><td>功能块（FBD）</td><td> 
</td></tr>
</table>

4.8.8　TAN——正切指令

- **功能**：求输入数据的正切值，输入数据以弧度表示。（弧度公式参见 SIN 指令）
- **输入数据**：BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT；
- **输出数据**：REAL。

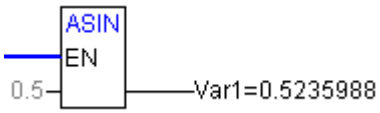
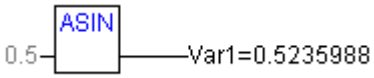
✧ 指令使用举例

<table>
<tr><td colspan="2">变量定义
 
</td></tr>
<tr><td>编程语言</td><td>程　序</td></tr>
<tr><td>梯形图（LD）</td><td> 
</td></tr>
<tr><td>结构化文本（ST）</td><td>Var1:=TAN(0.5);　　(*结果 Var1 为 0.5463025 *)</td></tr>
<tr><td>指令列表（IL）</td><td>LD　0.5
TAN
ST　Var1　　(*结果 Var1 为 0.5463025 *)</td></tr>
<tr><td>功能块（FBD）</td><td> 
</td></tr>
</table>

4.8.9　ASIN——反正弦指令

- **功能**：求输入数据的反正弦值。
- **输入数据**：BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT；
- **输出数据**：REAL，（以弧度表示）。

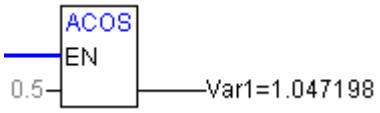
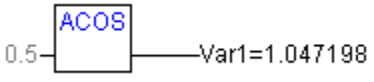
✧ 指令使用举例

变量定义	
	
编程语言	**程　序**
梯形图（LD）	
结构化文本（ST）	Var1:=ASIN(0.5);　(*结果 Var1 为 0.5235988 *)
指令列表（IL）	LD　0.5 ASIN ST　Var1　(*结果 Var1 为 0.5235988 *)
功能块（FBD）	

4.8.10 ACOS——反余弦指令

- **功能**：求输入数据的反余弦值。
- **输入数据**：BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT;
- **输出数据**：REAL（输出数据以弧度表示）。

✧ 指令使用举例

变量定义	
	
编程语言	**程　序**
梯形图（LD）	
结构化文本（ST）	Var1:=ACOS(0.5);　(*结果 Var1 为 1.047198 *)
指令列表（IL）	LD　0.5 ACOS ST　Var1　(*结果 Var1 为 1.047198 *)
功能块（FBD）	

4.8.11 ATAN——反正切指令

- **功能**：求输入数据的反正切值。
- **输入数据**：BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT;
- **输出数据**：REAL 型，（输出数据以弧度表示）。

✧ 指令使用举例

变量定义					
	VAR	VAR_INPUT	VAR_OUTPUT	VAR_IN_OUT	CON
	名称	地址	类型	初始值	注释
0001	Var1		REAL		

编程语言	程　序
梯形图（LD）	ATAN EN 0.5 —— Var1=0.4636476
结构化文本（ST）	Var1:=ATAN(0.5);　(*结果 Var1 为 0.4636476 *)
指令列表（IL）	LD　0.5 ATAN ST　Var1　(*结果 Var1 为 0.4636476 *)
功能块（FBD）	ATAN 0.5 —— Var1=0.4636476

4.8.12 EXPT——幂指令

- **功能**：对输入数据求幂，输入数据 1 为幂底数，输入数据 2 为幂指数。
- **输入数据**：BYTE、WORD、DWORD、INT、DINT、REAL、SINT、USINT、UINT、UDINT;
- **输出数据**：REAL。

✧ 指令使用举例

变量定义					
	VAR	VAR_INPUT	VAR_OUTPUT	VAR_IN_OUT	CON
	名称	地址	类型	初始值	注释
0001	Var1		REAL		

编程语言	程　序
梯形图（LD）	EXPT EN 7 2 —— Var1=49
结构化文本（ST）	Var1:=EXPT(7,2);　(*结果 Var1 为 49 *)
指令列表（IL）	LD　7 EXPT　2 ST　Var1　(*结果 Var1 为 49 *)
功能块（FBD）	EXPT 7 2 —— Var1=49

4.9 地址运算指令

4.9.1 ADR——取地址指令

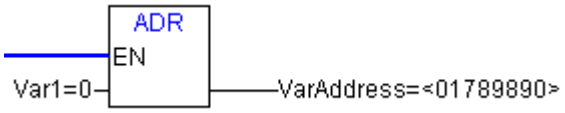
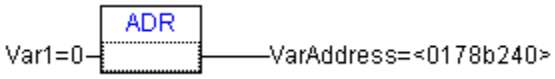
- **功能**：取得输入变量的内存地址，并输出。该地址可以在程序内当作指针使用，也可以作为指针传送给函数。

✧ 指令使用举例

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONS

	名称	地址	类型	初始值	注释
0001	Var1		BYTE		
0002	VarAddress		POINTER TO BYTE		

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	VarAddress:= ADR(Var1);
指令列表（IL）	LD　Var1 ADR ST　VarAddress
功能块（FBD）	

4.9.2 ^——取地址内容指令

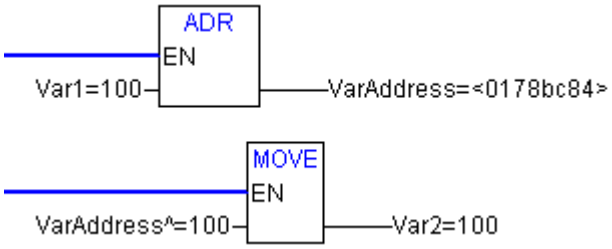
➢ **功能**：在指针变量后增加“^”符号，以取得该指针所指地址的数据。

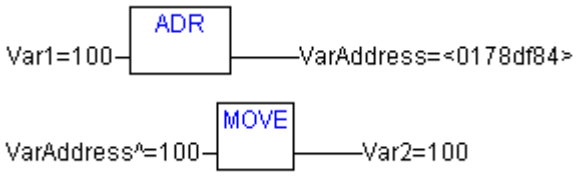
✧ 指令使用举例

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CON

	名称	地址	类型	初始值	注释
0001	Var1		BYTE		
0002	Var2		BYTE		
0003	VarAddress		POINTER TO BYTE		

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	VarAddress:= ADR(Var1); Var2:= VarAddress^;　(*结果 Var2 为 100 *)
指令列表（IL）	LD　Var1 ADR ST　VarAddress LD　VarAddress^ ST　Var2

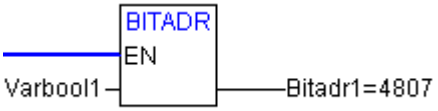
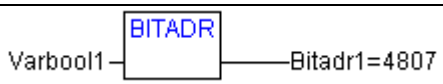
功能块（FBD）	

4.9.3 BITADR——位地址指令

➢ **功能**：取得 BOOL 量的位地址，下例中 MX300.7 的地址为 300*16+7=4807。

✧ **指令使用举例**

变量定义

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	Bitadr1:=BITADR(Varbool1)；
指令列表（IL）	LD　Varbool1 BITADR ST　Bitadr1
功能块（FBD）	

4.9.4 INDEXOF——索引指令

➢ **功能**：在 POU 中执行索引指令，可以寻找 POU 的索引号，其输入为 POU 的名称，输出为 INT 的数据。

✧ **指令使用举例**

变量定义

编程语言	程　序
梯形图（LD）	 （POU2 可以是程序、功能块、函数）
结构化文本（ST）	Var1:=INDEXOF(POU2);　（*结果 Var1 为 38*）
指令列表（IL）	LD　POU2 INDEXOF ST　Var1　（*结果 Var1 为 38*）

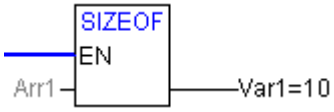
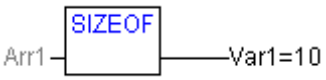
功能块（FBD）	 （*结果 Var1 为 38*）

4.9.5 SIZEOF——数据类型大小指令

- **功能**：取得数据类型所需字节数。

✧ **指令使用举例**

变量定义

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	Var1:=SIZEOF(arr1); （*结果 Var1 为 10*）
指令列表（IL）	LD　　arr1 SIZEOF ST　　Var1　　（*结果 Var1 为 10*）
功能块（FBD）	

4.10 调用指令

- CAL—调用指令
- **功能**：调用功能块或程序。在 IL 语言中使用 CAL 运算来调用功能块或者程序。被调用功能块/程序的输入变量位于该功能块/程序名称右侧的括号内。

✧ **指令使用举例**

在程序中调用名为 Inst 的功能块，其输入变量分别为 Par1=0，Par2=TRUE。IL 中调用如下：

CAL Inst(Par1:=0, Par2:=TRUE)

4.11 初始化操作指令

- INI—初始化操作指令
- **功能**：用于初始化在程序中使用的功能块程序的内部保持型变量。

✧ **指令使用举例**

语法: <bool-Variable> := INI(<FB-instance, TRUE|FALSE)

在程序中调用名称为 FB-instance 的功能块，其输入变量分别为 Par1=FB-instance，Par2=TRUE|FALSE，输出为初始化完成标志。

主程序 PLC_PRG 变量定义：

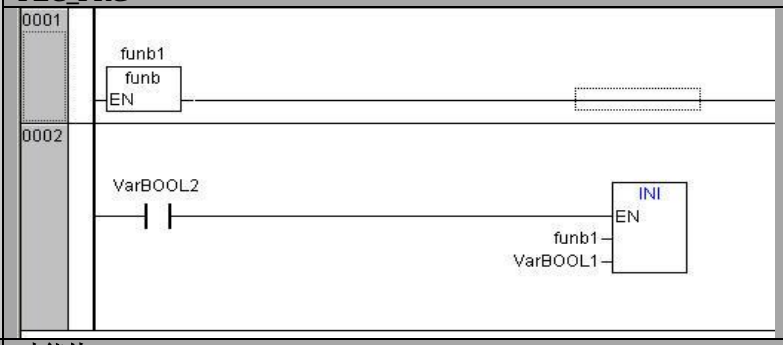
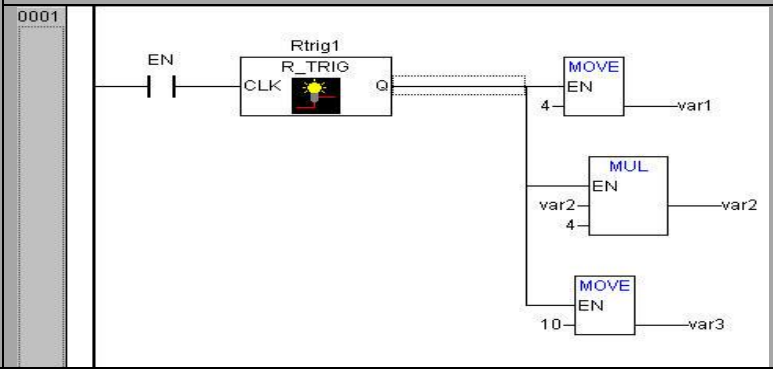
VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	VarBOOL2		BOOL		
0002	VarBOOL1		BOOL		
0003	funb1		funb		

funb（FB）变量定义：

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT | RETAIN

	名称	地址	类型	初始值	注释
0001	Rtrig		R_TRIG		
0002	var1		INT	1	
0003	var2		INT	2	
0004	var3		INT	3	

编程语言	程　序
LD	**PLC_PRG** 
	功能块 funb 

程序说明：

- 程序运行第一个扫描周期将变量 var1、var2、var3 的值改为 4、8、10，强制变量 VarBOOL2 接通，INI 指令执行，将三个保持型变量恢复为初始值 1、2、3。

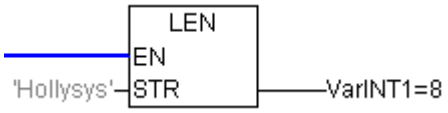
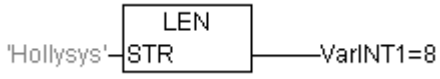
4.12　字符串处理指令（Standard.lib）

4.12.1　LEN——取字符串长度指令

- **功能**：计算字符串的长度。
- **输入数据**：STRING；

- **输出数据**：INT。

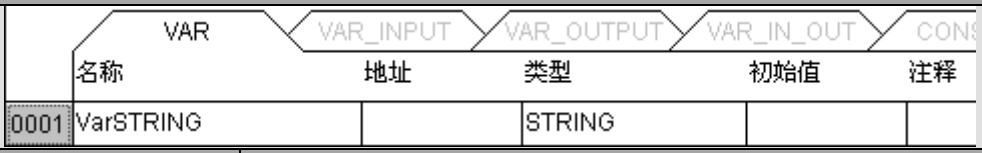
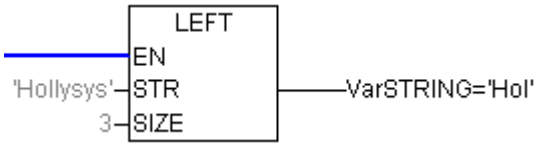
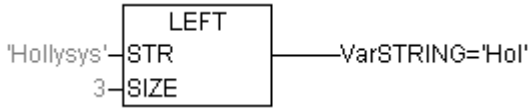
✧ **指令使用举例**

变量定义	
	
编程语言	**程　序**
LD	
ST	Var1INT1 := LEN (‘Hollysys’); (*结果 Var1INT1 为 8*)
IL	LD　‘Hollysys’ LEN ST　VarINT1　(*结果 Var1INT1 为 8*)
FBD	

4.12.2　LEFT——左边取字符串指令

- **功能**：从字符串左边取字符串。
- **指令格式**：LEFT（STR,SIZE），
- **输入数据**：STR 是 STRING 类型，为输入字符串；SIZE 是 INT 型，为从输入字符串左边开始获取的字符个数，
- **输出数据**：STRING。

✧ **指令使用举例**

变量定义	
	
编程语言	**程　序**
LD	
ST	VarSTRING := LEFT (‘Hollysys’,3); (*结果为' Hol '*)
IL	LD　' Hollysys ' LEFT　3 ST　VarSTRING　(*结果为' Hol '*)
FBD	

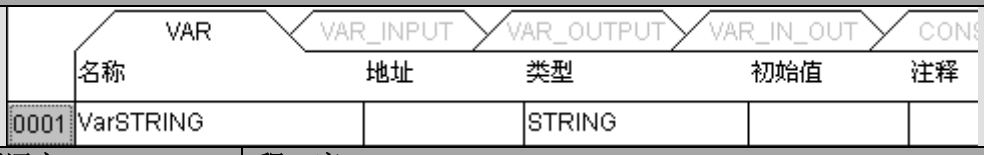
4.12.3　RIGHT——右边取字符串指令

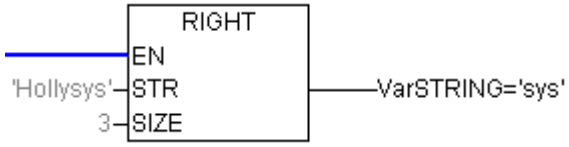
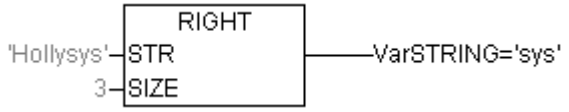
- **功能**：从字符串右边取字符串。
- **指令格式**：RIGHT（STR,SIZE）；

- **输入数据**：STR 是 STRING 类型，为输入字符串，SIZE 是 INT 型，为从字符串右边开始获取的字符个数；
- **输出数据**：STRING。

✧ **指令使用举例**

变量定义



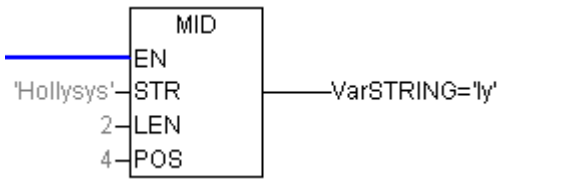
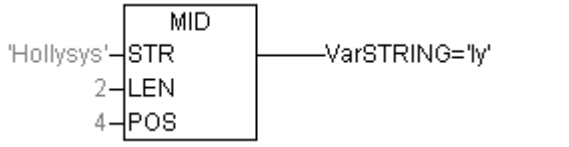
编程语言	程　序
LD	
ST	VarSTRING := RIGHT(‘Hollysys’,3);　(*结果为' sys '*)
IL	LD　' Hollysys ' RIGHT 3 ST　VarSTRING　(*结果为' sys '*)
FBD	

4.12.4　MID——中间取字符串指令

- **功能**：从字符串中间取字符串。
- **指令格式**：MID(STR,LEN,POS)，
- **输入数据**：STR 是 STRING 类型，为输入字符串；LEN 和 POS 是 INT 型，该指令从 POS 开始从左往右获取 LEN 个字符；
- **输出数据**：STRING。

✧ **指令使用举例**

变量定义

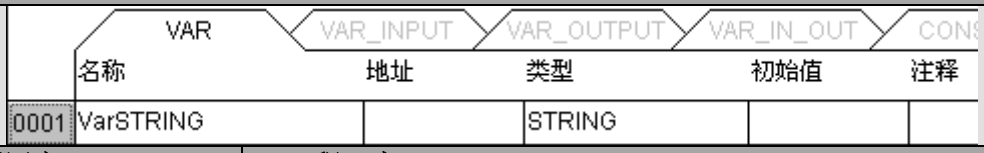
编程语言	程　序
LD	
ST	VarSTRING:= MID(‘Hollysys’,2,4);　(*结果为‘ly’ *)
IL	LD　‘Hollysys’ RIGHT　2,4 ST　VarSTRING　(*结果为‘ly’ *)
FBD	

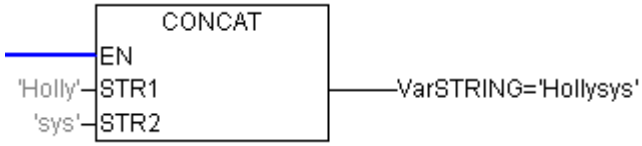
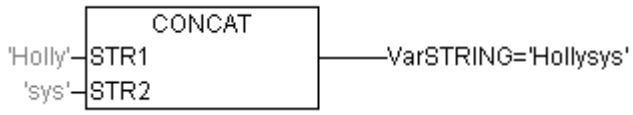
4.12.5 CONCAT——合并字符串指令

- **功能**：把两个字符串按前后顺序结合成一个字符串，
- **输入数据**：STRING；
- **输出数据**：STRING。

✧ **指令使用举例**

变量定义



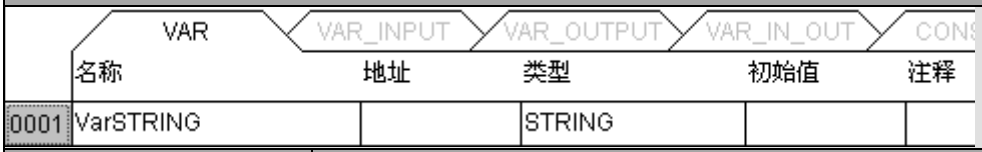
编程语言	程　序
LD	
ST	VarSTRING := CONCAT ('Holly', 'sys'); (*结果为'Hollysys'*)
IL	LD 'Holly' CONCAT ‘sys’ ST VarSTRING (*结果为'Hollysys'*)
FBD	

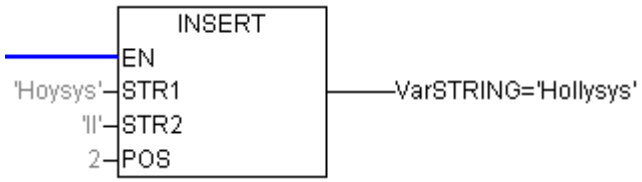
4.12.6 INSERT——插入字符串指令

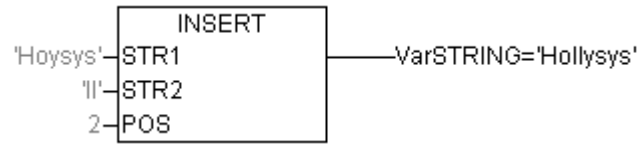
- **功能**：把一个字符串插入到另一个字符串中。
- **指令格式**：INSERT(STR1,STR2,POS)。
- **输入数据**：STR1 和 STR2 是 STRING 类型，POS 是 INT 型，该指令把 STR2 插入到 STR1 的 POS 位置之后。
- **输出数据**：STRING。

✧ **指令使用举例**

变量定义



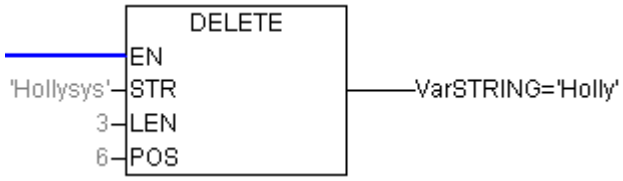
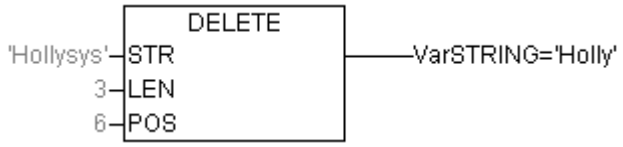
编程语言	程　序
LD	
ST	VarSTRING := INSERT ('Hoysys', ‘ll’,2); (*结果为 'Hollysys'*)
IL	LD 'Hoysys' INSERT ‘ll’,2 ST VarSTRING (*结果为 'Hollysys'*)

FBD	

4.12.7 DELETE——删除字符指令

- **功能**：从字符串中删除字符。
- **指令格式**：DELETE(STR,LEN,POS)。
- **输入数据**：STR 是 STRING 类型，为输入字符串；LEN 和 POS 是 INT 型，该指令从输入字符的位置 POS 处开始从左往右删除 LEN 个字符。
- **输出数据**：STRING。

✧ **指令使用举例**

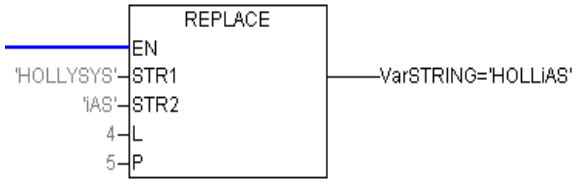
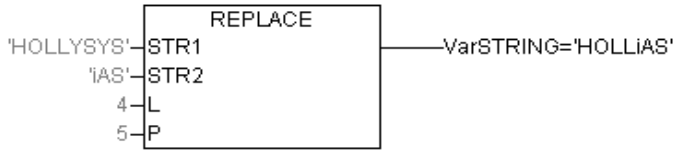
变量定义					
	VAR / VAR_INPUT / VAR_OUTPUT / VAR_IN_OUT / CONS				
	名称	地址	类型	初始值	注释
	0001 VarSTRING		STRING		
编程语言	**程　序**				
LD					
ST	VarSTRING := DELETE ('Hollysys',3,6);　(*结果为'Holly'*)				
IL	LD　'Hollysys' DELETE　3,6 ST　VarSTRING　(*结果为'Holly'*)				
FBD					

4.12.8 REPLACE——替换字符串指令

- **功能**：用一个字符串替代另一字符串中的部分内容。
- **指令格式**：REPLACE(STR1,STR2,L,P)。
- **输入数据**：STR1 和 STR2 是 STRING 类型，为输入字符串；L 和 P 是 INT 型，该指令用 STR2 代替 STR1 中从 P 位置开始的 L 个字符。
- **输出数据**：STRING。

✧ **指令使用举例**

变量定义					
	VAR / VAR_INPUT / VAR_OUTPUT / VAR_IN_OUT / CONS				
	名称	地址	类型	初始值	注释
	0001 VarSTRING		STRING		

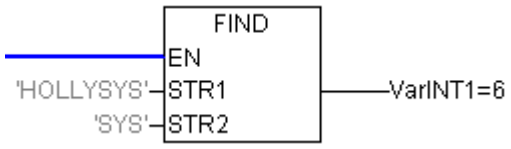
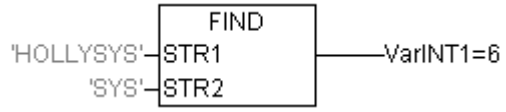
编程语言	程　序
LD	
ST	VarSTRING:= REPLACE ('HOLLYSYS', 'iAS',4,5); (*结果为'HOLLiAS'*)
IL	LD　　'HOLLYSYS' REPLACE　'iAS', 4,5 ST　　VarSTRING　　(*结果为'HOLLiAS'*)
FBD	

4.12.9　FIND——查找字符串指令

- **功能**：在一个字符串中查找与另一字符串完全相同的内容。
- **指令格式**：FIND(STR1,STR2)。
- **输入数据**：STR1 和 STR2 都是 STRING 类型，为输入字符串，返回数据类型为 INT 型。指令功能为在第一个字符串 STR1 中查找与字符串 STR2 完全相同的部分，返回该相同部分在字符串 STR1 的起始位置。若没有完全相同的部分，输出结果为 0。

✧ **指令使用举例**

变量定义					
	VAR	VAR_INPUT	VAR_OUTPUT	VAR_IN_OUT	CON
	名称	地址	类型	初始值	注释
0001	VarINT1		INT		

编程语言	程　序
LD	
ST	VarINT1 := FIND ('HOLLYSYS', 'SYS'); (*结果为 6*)
IL	LD　　'HOLLYSYS' FIND　　'SYS' ST　　VarINT1　　(*结果为 6*)
FBD	

4.13　库版本信息检查指令（Util.lib）

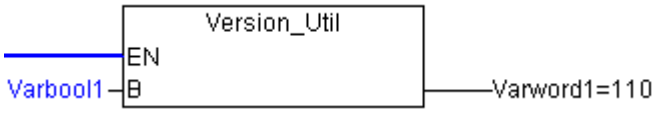
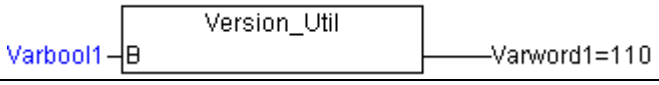
- Version_Util——库版本信息检查指令
- **功能**：读取当前软件的库版本信息码。
- **输入类型**：BOOL；
- **输出类型**：WORD；

✧ 指令使用举例

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	Varbool1		BOOL		
0002	Varword1		WORD		

编程语言	程　序
LD	
ST	Varword1:=Version_Util(Varbool1);
IL	LD　Varbool1 Version_Util ST　Varword1
FBD	

4.14 检查指令（HS_Check.lib）

4.14.1 CheckBounds——数组边界检查指令

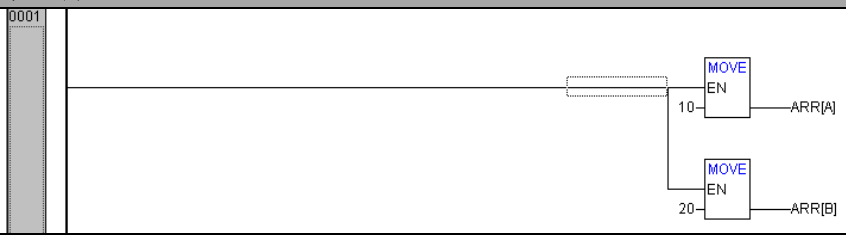
- **功能**：数组边界检查。
- **输入类型**：Index, lower, upper: DINT；
- **输出类型**：CheckBounds : DINT

✧ 指令使用举例

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	Arr		ARRAY[1..7] OF BYT		
0002	A		INT	-2	
0003	B		INT	10	

编程语言	程　序
LD	
ST	ARR[A]:=10; ARR[B]:=20;
IL	LD　10 ST　Arr[A] LD　20 ST　Arr[B]

FBD	

程序说明：

添加 HS_Check.lib 库后，无需调用 CheckBounds 指令，就可以检测数组边界，程序中 A 和 B 初始值分别为-2 和 10，超过 1 至 7 范围，所以 Arr[A]和 Arr[B]自动对应数组 Arr[1]和 Arr[7]，也就是 10 赋值给 Arr[1]而 20 赋值给 Arr[7]。

4.14.2 CheckDivByte——字节型除数为零检查指令

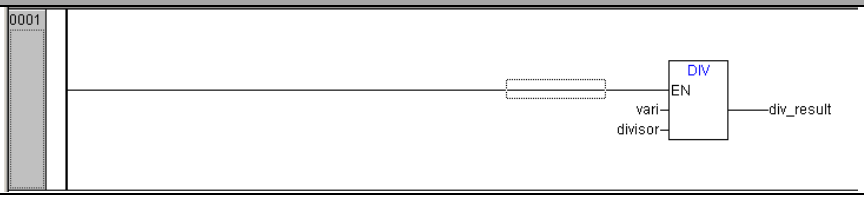
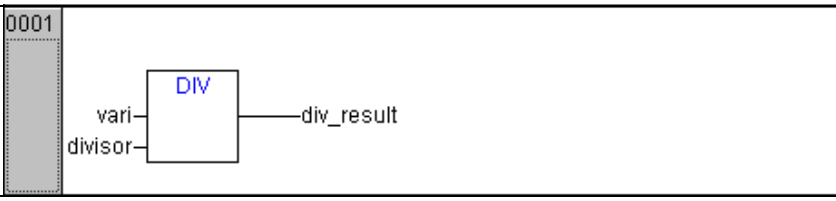
- **功能**：用于字节型除法运算中的除数为零检查，当除数为零时，按照除数等于 1 进行运算。
- **输入类型**：divisor: BYTE；
- **输出类型**：CheckDivByte: BYTE。

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	divisor		BYTE	0	
0002	vari		BYTE	100	
0003	div_result		BYTE	10	

编程语言	程　序
LD	
ST	Div_result:=vari/divsor;
IL	LD　vari DIV　divisor ST　div_result
FBD	

程序说明：

添加 HS_Check.lib 库后，无需调用 CheckDivByte 指令，就可以检测 0 除数，程序中 divisor 为 0，当 vari 除以 divisor 时就相当于除以 1。

4.14.3 CheckDivWord——字型除数为零检查指令

- **功能**：用于字型除法运算中的除数为零检查，当除数为零时，按照除数等于 1 进行运算。
- **输入类型**：divisor: WORD；
- **输出类型**：CheckDivWord: WORD。

✧ **指令使用举例**

见“4.15.2 CheckDivByte”指令使用举例，只是把变量类型改为 WORD 型。

4.14.4 CheckDivDWord——双字型除数为零检查指令

- **功能**：用于双字型除法运算中的除数为零检查，当除数为零时，按照除数等于 1 进行运算。
- **输入类型**：divisor: DWORD；
- **输出类型**：CheckDivDword: DWORD。

✧ **指令使用举例**

见“4.15.2 CheckDivByte”指令使用举例，只是把变量类型改为 DWORD 型。

4.14.5 CheckDivReal——实型除数为零检查指令

- **功能**：用于实数型除法运算中的除数为零检查，当除数为零时，按照除数等于 1 进行运算。
- **输入类型**：divisor: REAL；
- **输出类型**：CheckDivReal: REAL。

✧ **指令使用举例**

见 4.15.2CheckDivByte 指令使用举例，只是把变量类型改为 REAL 型。

4.14.6 CheckRangeSigned——整型边界检查指令

- **功能**：用于检查整型的 subrange 类型变量的范围越界。如果所赋值小于 subrange 类型变量的下边界，则该变量值为所定义的边界最小值；如果所赋值大于 subrange 类型变量的上边界，则该变量值为所定义的边界最大值。
- **Subrange 类型**：是某种基本数据类型取值范围的子集，它可以在数据类型中定义见下图，也可以在程序的变量定义中直接定义，见指令使用举例中变量定义。

在数据类型中声明 Subrange 类型

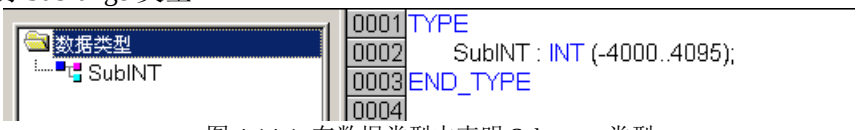


图 4-14-1 在数据类型中声明 Subrange 类型

✧ 指令使用举例

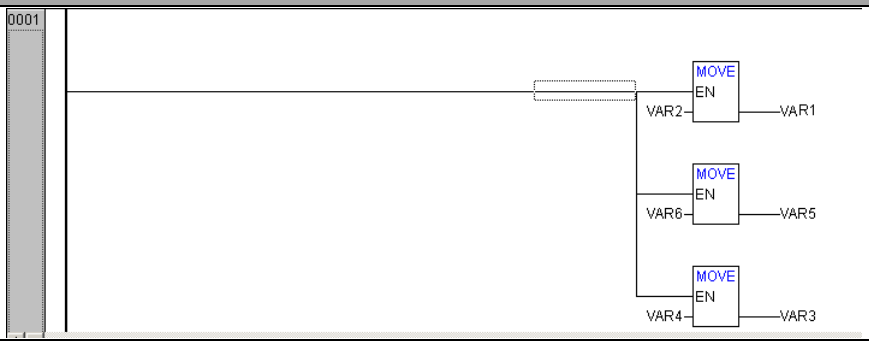
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT | R

	名称	地址	类型	初始值	注释
0001	VAR1		INT(-100..100)		
0002	VAR2		INT	500	
0003	VAR3		subint		
0004	VAR4		INT	5000	
0005	VAR5		WORD(100..400)		
0006	VAR6		WORD	500	

编程语言 | **程 序**

LD



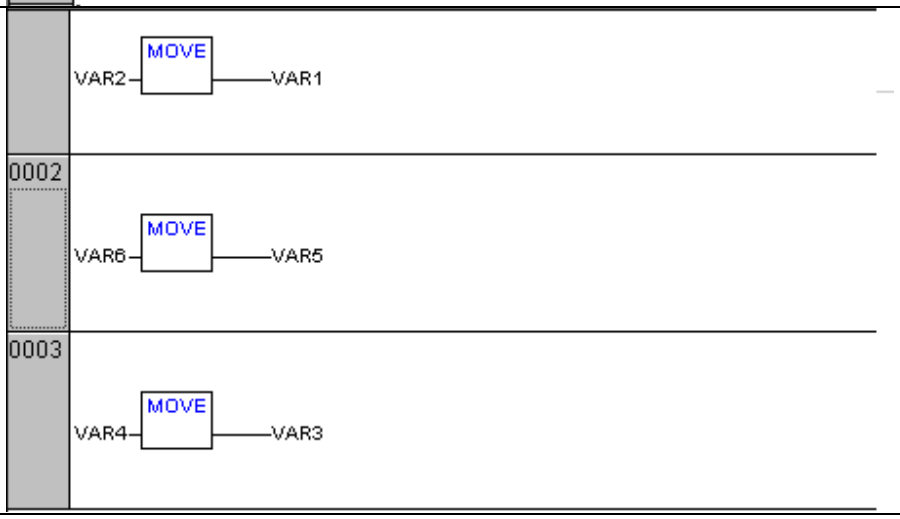
ST

```
VAR1:=VAR2;
VAR5:=VAR6;
VAR3:=VAR4
```

IL

```
0001   LD    VAR2
0002   ST    VAR1
0003   LD    VAR6
0004   ST    VAR5
0005   LD    VAR4
0006   ST    VAR3
```

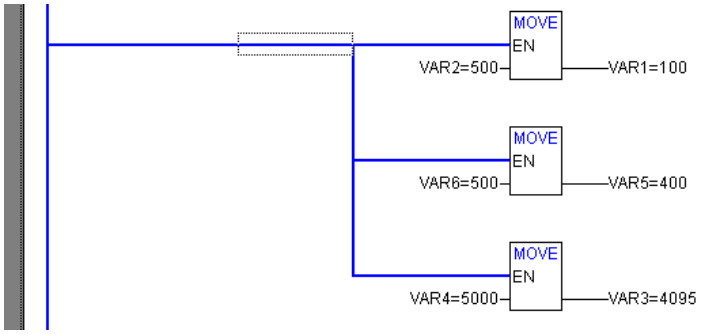
FBD



程序说明：

添加 HS_Check.lib 库后，无需调用 CheckRangeSigned 指令，在程序中定义了变量 VAR3 为 SubINT 类型，SubINT 类型的取值范围为-4000~4095，在程序中将 VAR4=5000，i 赋值给 VAR3，由于 VAR4 的值 5000 已经超过了 4095 上限，所以运行后，VAR3 的值取上边界值。同样道理

VAR1 和 VAR5 的值也是取上边界值，运行结果如下：



4.14.7 CheckRangeUnsigned——无符号整型边界检查指令

- **功能**：用于检查无符号整型的 subrange 类型变量的范围越界。如果所赋值小于 subrange 类型变量的下边界，则该变量值为所定义的边界最小值；如果所赋值大于 subrange 类型变量的上边界，则该变量值为所定义的边界最大值。

✧ **指令使用举例**

参照 4.14.6CheckRangesigned 指令使用举例，只是把变量类型定义成无符号整型。

4.15 BCD 码转换指令（Util.lib）

BCD 码的一个字节包含 0 到 99 之间的整数。每个十进制位对应 4 位，十位数存储在 4-7 位，个位数存储在 0-3 位。BCD 码格式和 16 进制表达方式很相似，差别在于 BCD 字节值是 0-99，而 16 进制是 0-FF。

BCD 码用 4 位二进制数表示一个十进制数位，整个十进制数用一串 BCD 码来表示。例如，十进制数 59 表示成 BCD 码为 0101 1001，但表示成 2 进制为 2#111011。51 转换成 BCD 码，5 的二进制是 0101，1 的二进制是 0001，那么 51 转换 BCD 码为 0101 0001。

4.15.1 BCD_TO_INT——BCD 码转整型指令

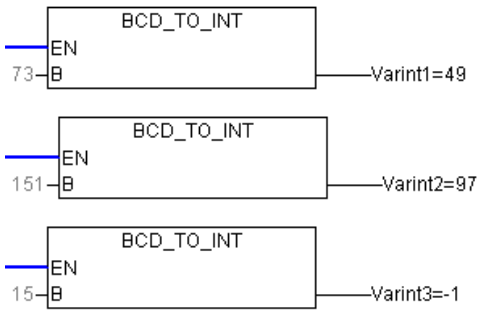
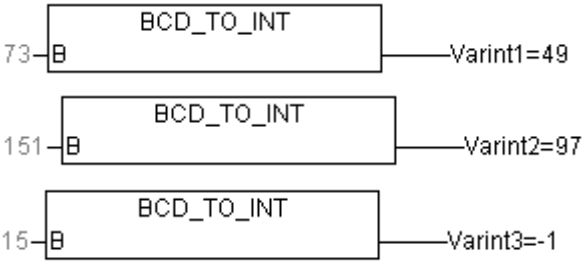
- **功能**：该指令将 BCD 码转为 INT 值。
- **输入 B**：BYTE 型，输入 BCD 码的二进制形式（或者该二进制对应的十进制和十六进制）。比如 BCD 码 49，表示为 2#100_1001（或者对应 10#73、16#49），则此处输入 2#100_1001（或者 10#73，16#49）；
- **输出**：INT 型，该 BCD 码所代表的实际值，如果输入的字节不是 BCD 码，输出是 -1。

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	Varint1		INT		
0002	Varint2		INT		
0003	Varint3		INT		

编程语言	程　序
LD	
ST	Varint1:=BCD_TO_INT(73);　(*结果为 49 *) Varint2:=BCD_TO_INT(151);　(*结果为 97*) Varint3:=BCD_TO_INT(15);　(*输出-1，因为不是 BCD 码格式*)
IL	LD　73 BCD_TO_INT ST　Varint1　(*结果为 49 *)
	LD　151 BCD_TO_INT ST　Varint2　(*结果为 97*)
	LD　15 BCD_TO_INT ST　Varint3　(*输出-1，因为不是 BCD 码格式*)
FBD	

4.15.2　INT_TO_BCD——整型转 BCD 码指令

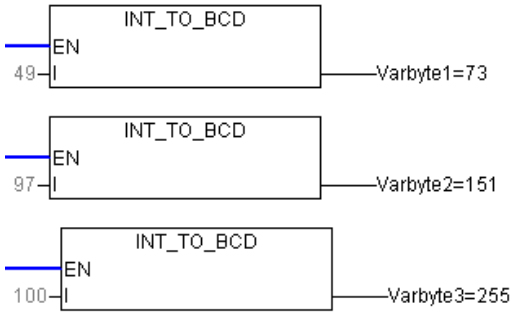
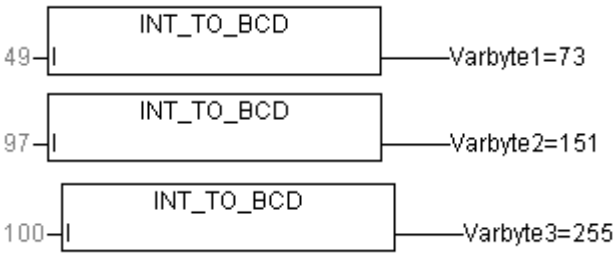
- **功能**：将整数值转换成 BCD 码，当整数值不能转换成 BCD 码字节时，输出数值 255。
- **输入 I**：INT 型，如果整数值为 49，则此处输入整型数据 49。
- **输出**：BYTE 型，转换完的 BCD 码的值，比如将 49 转换为 BCD 码为 2#100_1001，则此处输出 2#100_1001（或者该二进制对应的 10#73、16#49）。

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	Varbyte1		BYTE		
0002	Varbyte2		BYTE		
0003	Varbyte3		BYTE		

编程语言	程　序
LD	
ST	Varbyte1:=INT_TO_BCD(49);　　(*结果为 73*) Varbyte2:= INT_TO_BCD(97);　　(*结果为 151*) Varbyte3:= INT_TO_BCD(100);　　(*错误！输出：255*)
IL	LD　　49 INT_TO_BCD ST　　Varbyte1　　(*结果为 73*)
	LD　　97 INT_TO_BCD ST　　Varbyte2　　(*结果为 151*)
	LD　　100 INT_TO_BCD ST　　Varbyte3　　(*错误！输出：255*)
FBD	

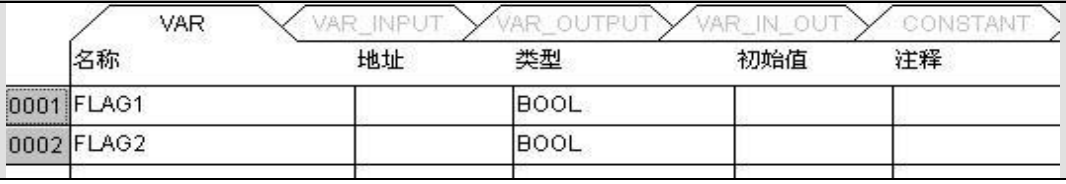
4.16　位/字节操作指令（Util.lib）

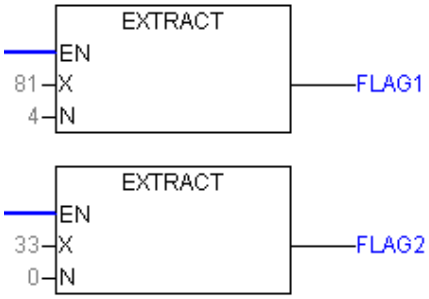
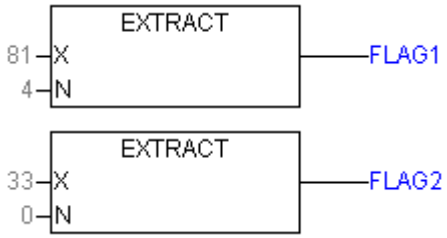
4.16.1　EXTRACT——位提取指令

- **功能**：提取输入变量 X 二进制数的第 N 位（N=0,1...）并输出该位数值。
- **输入变量**：X 是 DWORD 类型，N 是 BYTE 型；
- **输出变量**：BOOL。

✧ **指令使用举例**

变量定义



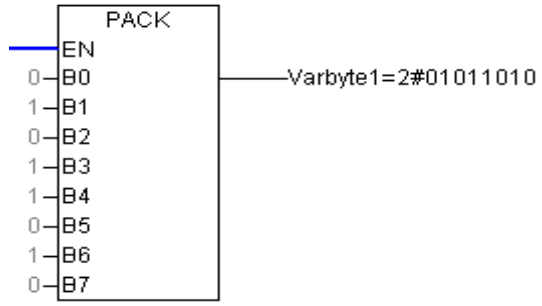
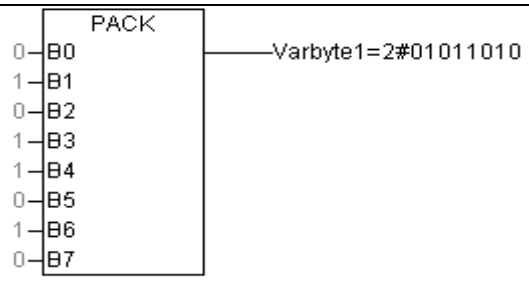
编程语言	程　序
LD	 (*结果 FLAG1=TRUE，FLAG2=TRUE *)
ST	FLAG1:=EXTRACT(X:=81, N:=4); (*结果：TRUE，因为 81 的二进制数是 1010001，所以第四位是 1*) FLAG2:=EXTRACT(X:=33, N:=0); (*结果：TRUE，因为 33 的二进制数是 100001，所以第 0 位是 1*)
IL	LD　81 EXTRACT　4 ST　FLAG1 (*结果：TRUE，因为 81 的二进制数是 1010001，所以第四位是 1*)
	LD　33 EXTRACT　0 ST　FLAG2 (*结果：TRUE，因为 33 的二进制数是 100001，所以第 0 位是 1*)
FBD	 (*结果 FLAG1=TRUE *) (*结果 FLAG2=TRUE *)

4.16.2　PACK——位整合指令

- **功能**：把输入位 B0、B1、……、B7 合成为一个字节，与这个指令相对应的指令是 UNPACK。
- **输入数据**：B0、B1、……、B7 均为 BOOL 类型；
- **输出数据**：BYTE。

✧ **指令使用举例**

变量定义					
	VAR	VAR_INPUT	VAR_OUTPUT	VAR_IN_OUT	CONSTANT
	名称	地址	类型	初始值	注释
0001	Varbyte1		BYTE		
编程语言	**程　序**				

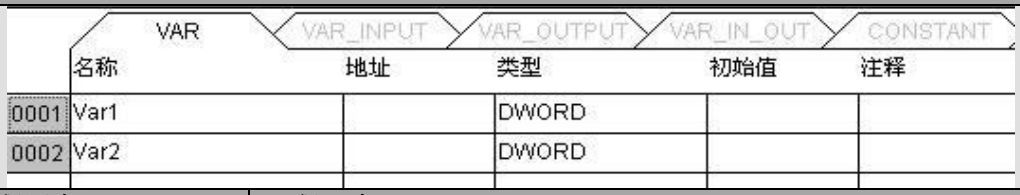
LD	
ST	Varbyte1:= PACK(0,1,0,1,1,0,1,0); (*结果为 2#01011010*)
IL	LD FALSE PACK TRUE,FALSE,TRUE,TRUE,FALSE,TRUE,FALSE ST Varbyte1 (*结果为 2#01011010*)
FBD	

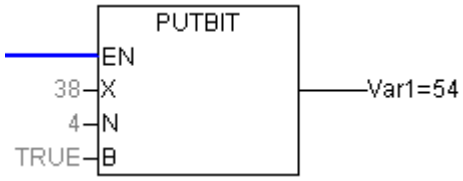
4.16.3 PUTBIT——位赋值指令

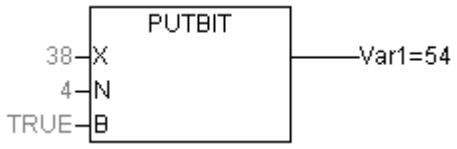
- **功能**：将输入变量 X 值的第 N 位（N=0,1...）赋值为 B，并输出 X 转变后的值。
- **输入变量**：X 为 DWORD 型、N 为 BYTE 型和 B 为 BOOL 型；
- **输出变量**：DWORD。

✧ **指令使用举例**

变量定义



编程语言	程　序
LD	
ST	Var2:=38; (*二进制 100110*) Var1:=PUTBIT(Var2,4,TRUE); (*结果：54 = 2#110110*)
IL	LD 38 (*二进制 100110*) PUTBIT 4,TRUE ST Var1 (*结果：54 = 2#110110*)

FBD	

4.16.4 UNPACK——位拆分

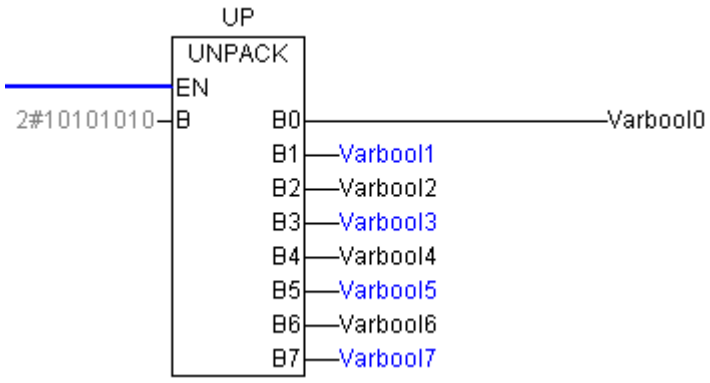
- **功能**：将字节型的输入 B 拆分转换成 8 个 BOOL 类型的输出变量 B0,…,B7，与 PACK 相反。
- **输入数据**：B：BYTE；
- **输出数据**：B0、B1、……、B7： BOOL。

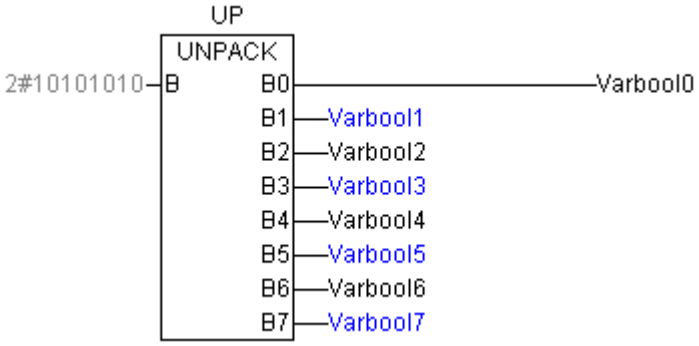
✧ **指令使用举例**

变量定义

VAR

	名称	地址	类型	初始值	注释
0001	UP		UNPACK		
0002	Varbool0		BOOL		
0003	Varbool1		BOOL		
0004	Varbool2		BOOL		
0005	Varbool3		BOOL		
0006	Varbool4		BOOL		
0007	Varbool5		BOOL		
0008	Varbool6		BOOL		
0009	Varbool7		BOOL		

编程语言	程　序
LD	
ST	UP(B:=2#10101010); Varbool0:=UP.B0; Varbool1:=UP.B1; Varbool2:=UP.B2; Varbool3:=UP.B3; Varbool4:=UP.B4; Varbool5:=UP.B5; Varbool6:=UP.B6; Varbool7:=UP.B7;
IL	CAL　UP(B := 2#10101010) LD　UP.B1

	ST Varbool1 LD UP.B2 ST Varbool2 LD UP.B3 ST Varbool3 LD UP.B4 ST Varbool4 LD UP.B5 ST Varbool5 LD UP.B6 ST Varbool6 LD UP.B7 ST Varbool7 LD UP.B0 ST Varbool0
FBD	

4.17 高等数学运算指令（Util.lib）

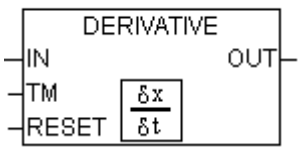
4.17.1 DERIVATIVE——微分

- **功能**：该指令对连续输入的变量进行微分运算。为了获得最好结果，DERIVATIVE 指令只对最新的四个输入值进行微分，减小因输入参数不精确产生的误差。
- **微分迭代公式**：

$$OUT = \frac{3*[IN(k) - IN(k-3)] + IN(k-1) - IN(k-2)}{3*TM(k-2) + 4*TM(k-1) + 3*TM(k)}$$

k-3、k-2、k-1、k 为连续四次输入值的标记。

- **指令示例**：

梯形图语言示例


- **参数说明**：

输入参数	数据类型	功能描述	参数值说明
IN	REAL	连续输入的变量	
TM	BOOL	微分时间	毫秒

RESET	BOOL	复位信号	值是 TRUE 时，重新启动指令
输出参数	**数据类型**	**功能描述**	**参数值说明**
OUT	REAL	微分结果输出	

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	DERIVATIVEInst		DERIVATIVE		
0002	Varreal1		REAL		
0003	Varint1		INT		
0004	VarBOOL1		BOOL		

编程语言	**程　序**
LD	DERIVATIVEInst DERIVATIVE EN Varint1 — IN　OUT — Varreal1 100 — TM VarBOOL1 — RESET
ST	DERIVATIVEInst (IN:=Varint1,TM:=100,RESET:=VarBOOL1); Varreal1:=DERIVATIVEInst.OUT;
IL	CAL　DERIVATIVEInst(IN := Varint1, TM := 100, RESET := VarBOOL1) LD　DERIVATIVEInst.OUT ST　Varreal1
FBD	DERIVATIVEInst DERIVATIVE Varint1 — IN　OUT — Varreal1 100 — TM VarBOOL1 — RESET

输入输出对应关系如下图所示：

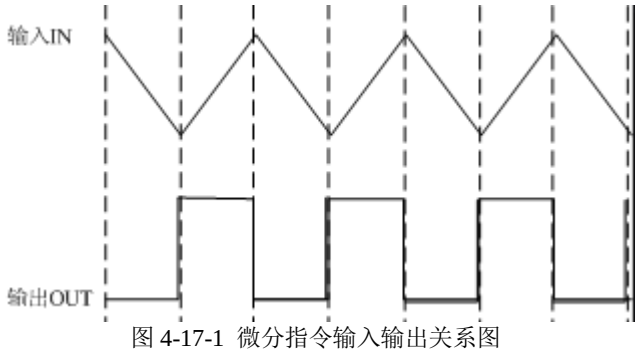


图 4-17-1 微分指令输入输出关系图

4.17.2 INTEGRAL——积分

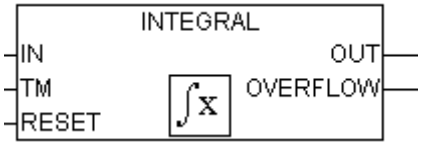
- **功能**：该指令对连续输入的变量进行进行积分运算。
- **积分迭代公式**：$A(k) = A(k-1) + TM * IN(k-1)$

$B(k) = B(k-1) + TM * IN(k)$

$OUT(k) = \frac{A(k) + B(k)}{2}$

k-1 、k 为连续两次输入值的标记。

➢ 指令示例：

梯形图语言示例


➢ 参数说明：

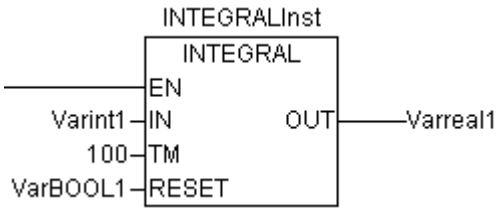
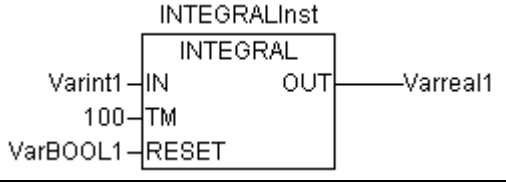
输入参数	数据类型	功能描述	参数值说明
IN	REAL	连续输入的变量	
TM	BOOL	积分时间	
RESET	BOOL	复位信号	其值是 TRUE 时，重新启动指令
输出参数	**数据类型**	**功能描述**	**参数值说明**
OUT	REAL	积分结果输出	
OVERFLOW	BOOL	溢出标志	其值是 TRUE 时，说明计算溢出

✧ 指令使用举例

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	INTEGRALInst		INTEGRAL		
0002	Varreal1		REAL		
0003	Varint1		REAL		
0004	VarBOOL1		BOOL		

编程语言	程　序
LD	
ST	INTEGRALInst(IN := Varint1, TM := 100, RESET := VarBOOL1); Varreal1:=INTEGRALInst.OUT;
IL	CAL INTEGRALInst(IN := Varint1, TM := 100, RESET := VarBOOL1) LD　　INTEGRALInst.OUT ST　　Varreal1
FBD	

输入输出对应关系如下图所示。

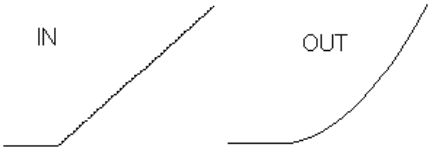
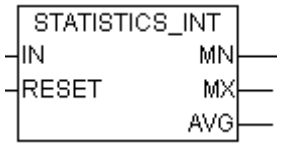


图 4-17-2 积分指令的输入输出关系

4.17.3 STATISTICS_INT——整型统计

- **功能**：周期统计输入端整型数据的最小值、最大值和平均值。平均值的计算方法是：当功能块使能运行后，把每个 CPU 任务周期的 IN 值累加起来，除以运行后所经历的周期数 COUNTER，结果四舍五入。
- **指令示例**：

梯形图语言示例


- **参数说明**：

输入参数	数据类型	功能描述	参数值说明
IN	INT	输入	
RESET	BOOL	初始化	其值是 TRUE 时，重新初始化： MN=IN; MX=IN; AVG=0; SUM=0; COUNTER=0
输出参数	**数据类型**	**功能描述**	**参数值说明**
MN	INT	最小值	
MX	INT	最大值	
AVG	INT	平均值	每周期计算 1 次，AVG=SUM/COUNTER
SUM（内部参数）	DINT	输入值之和	SUM 是每周期的 IN 值之和，即 SUM=SUM+IN
COUNTER（内部参数）	DINT	周期数	功能块使能运行后所经历的 CPU 任务周期数

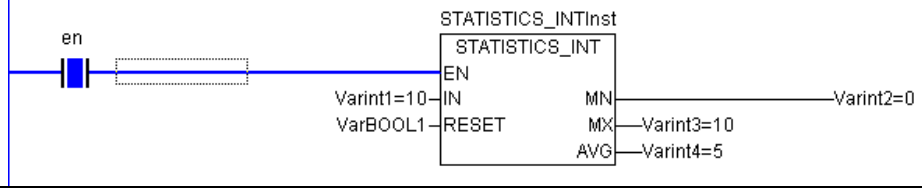
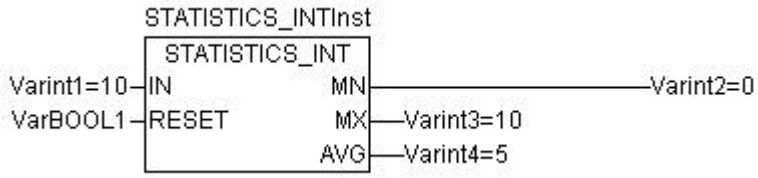
✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	STATISTICS_INTInst		STATISTICS_INT		
0002	VarBOOL1		BOOL		
0003	Varint1		INT		
0004	Varint2		INT		
0005	Varint3		INT		
0006	Varint4		INT		

编程语言	程　序

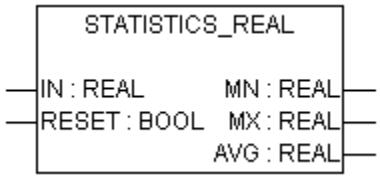
LD	(* 由于STATISTICS_INT没有EN使能端，所以需用"使能运算符"的方法调用。 *) AVG=SUM/COUNTER，举例说明如下： 假设运行后的第1个周期时IN=0，那么AVG=0；第2个周期时输入IN=10，那么AVG=(0+10)/2=5; 第3个周期我们没有改变IN的值，那么AVG=(0+10+10)/3=7; 第4个周期我们也没有改变IN值，那么AVG=(0+10+10+10)/4=8; 以后每个周期的AVG值以此类推。 
ST	STATISTICS_INTInst(IN := Varint1, RESET := VarBOOL1); Varint3:=STATISTICS_INTInst.MX; Varint4:=STATISTICS_INTInst.AVG; Varint2:=STATISTICS_INTInst.MN;
IL	CAL STATISTICS_INTInst(IN := Varint1, RESET := VarBOOL1) LD STATISTICS_INTInst.MX ST Varint3 LD STATISTICS_INTInst.AVG ST Varint4 LD STATISTICS_INTInst.MN ST Varint2
FBD	

程序说明：

- 由于 STATISTICS_INT 没有 EN 使能端，所以在 LD 编程语言中，需用“使能运算符”的方法调用。
- 平均值 AVG 是瞬时值，每个周期的 AVG 按照公式 AVG=SUM/COUNTER 计算得到，其中 SUM=SUM+IN。

4.17.4 STATISTICS_REAL——实型统计

- **功能**：周期统计输入实型数据的最小值、最大值和平均值。平均值的计算方法是：当功能块使能运行后，把每个 CPU 任务周期的 IN 值累加起来，除以运行后所经历的周期数 COUNTER。
- **指令示例**：

梯形图语言示例


- **参数说明**：

输入参数	数据类型	功能描述	参数值说明
IN	REAL	输入	

RESET	BOOL	初始化	其值是 TRUE 时，重新初始化： MN=IN; MX=IN; AVG=0; SUM=0; COUNTER=0
输出参数	**数据类型**	**功能描述**	**参数值说明**
MN	REAL	最小值	
MX	REAL	最大值	
AVG	REAL	平均值	每周期计算 1 次，AVG=SUM/COUNTER
SUM（内部参数）	REAL	输入值之和	SUM 是每周期的 IN 值之和，即 SUM=SUM+IN
COUNTER（内部参数）	DINT	周期数	功能块使能运行后所经历的 CPU 任务周期数

✧ **指令使用举例**

变量定义

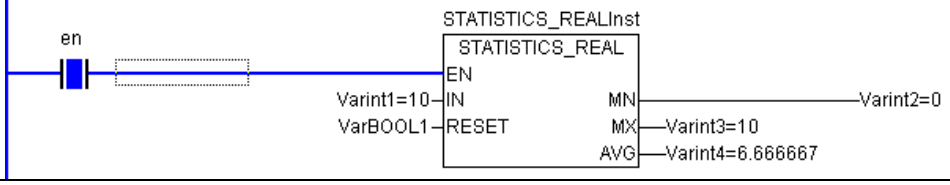
VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	STATISTICS_REALInst		STATISTICS_REAL		
0002	VarBOOL1		BOOL		
0003	Varreal1		REAL		
0004	Varreal2		REAL		
0005	Varreal3		REAL		
0006	Varreal4		REAL		

编程语言 | **程　序**

LD

```
(* 由于STATISTICS_REAL没有EN使能端，所以需用"使能运算符"的方法调用。 *)
AVG=SUM/COUNTER，举例说明如下：
假设运行后的第1个周期时IN=0，那么AVG=0；第2个周期时输入IN=10，那么AVG=(0+10)/2=5;
第3个周期我们没有改变IN的值，那么AVG=(0+10+10)/3=6.666667;
第4个周期我们也没有改变IN值，那么AVG=(0+10+10+10)/4=7.5;
以后每个周期的AVG值以此类推。
```



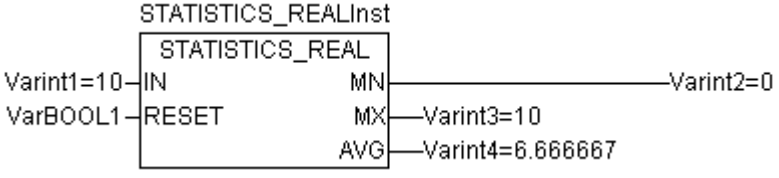
ST

```
STATISTICS_REALInst(IN :=Varreal1,RESET:=VarBOOL1);
Varreal3:=STATISTICS_REALInst.MX;
Varreal4:=STATISTICS_REALInst.AVG;
Varreal2:=STATISTICS_REALInst.MN;
```

IL

```
CAL STATISTICS_REALInst(IN := Varreal1, RESET := VarBOOL1)
LD      STATISTICS_REALInst.MX
ST      Varreal3
LD      STATISTICS_REALInst.AVG
ST      Varreal4
LD      STATISTICS_REALInst.MN
ST      Varreal2
```

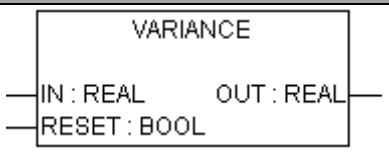
FBD



提示：
该指令与 STATISTIC_INT 功能相同，只是输入和输出为 REAL 型。

4.17.5 VARIANCE——平方偏差

- **功能**：该指令计算变量输入值的平方偏差（瞬时值，每个任务周期计算 1 次）。标准偏差可以由平方偏差的平方根得到。
- **指令示例**：

梯形图语言示例


- **参数说明**：

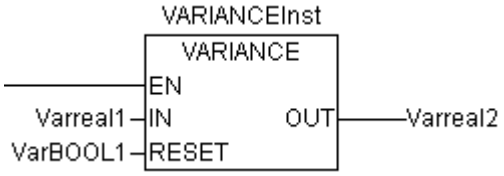
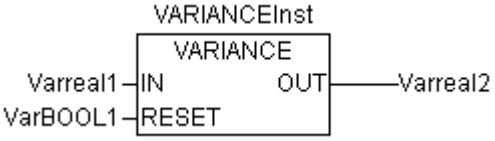
输入参数	数据类型	功能描述	参数值说明
IN	REAL	输入	
RESET	BOOL	复位	其值是 TRUE 时，指令复位
输出参数	**数据类型**	**功能描述**	**参数值说明**
OUT	REAL	平方偏差	

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	VARIANCEInst		VARIANCE		
0002	VarBOOL1		BOOL		
0003	Varreal1		REAL		
0004	Varreal2		REAL		

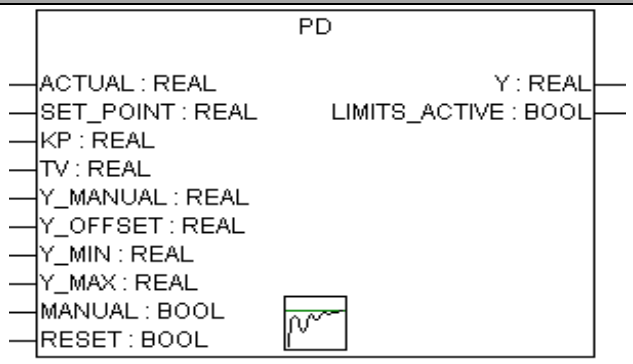
编程语言	程　序
LD	
ST	VARIANCEInst(IN := Varreal1, RESET := VarBOOL1); Varreal2:=VARIANCEInst.OUT;
IL	CAL VARIANCEInst(IN := Varreal1, RESET := VarBOOL1) LD　　VARIANCEInst.OUT ST　　Varreal2
FBD	

4.18 控制器指令（Util.lib）

4.18.1 PD——比例微分控制器

- **功能**：该指令为比例微分控制器。
- **控制方程**：$\Delta = \mathrm{SET}\_POINT - ACTUAL$

  $Y = KP * (\Delta + TV * \frac{\sigma\Delta}{\sigma t}) + Y\_OFFSET$ 该指令会自动计算 $\frac{\sigma\Delta}{\sigma t}$，用户无需考虑。
- **指令示例**：

梯形图语言示例


- **参数说明**：

输入参数	数据类型	功能描述	参数值说明
ACTUAL	REAL	测量值	
SET_POINT	REAL	设定值	
KP	REAL	比例系数	
TV	DWORD	微分时间	单位为秒（s）
Y_MANUAL	REAL	手动值	MANUAL=TRUE 时，Y= Y_MANUAL
Y_OFFSET	REAL	输出值的偏移量	
Y_MIN	REAL	输出值的最小值	
Y_MAX	REAL	输出值的最大值	
MANUAL	BOOL	手自动选择	TRUE 时为手动调节，FALSE 时为自动调节
RESET	BOOL	重置	TRUE 时重置该控制器，正常运行时应置 FALSE
输出参数	**数据类型**	**功能描述**	**参数值说明**
Y	REAL	输出值	
LIMITS_ACTIVE	BOOL	输出超限标志	输出值超限时等于 TRUE，即超出（Y_MIN，Y_MAX）

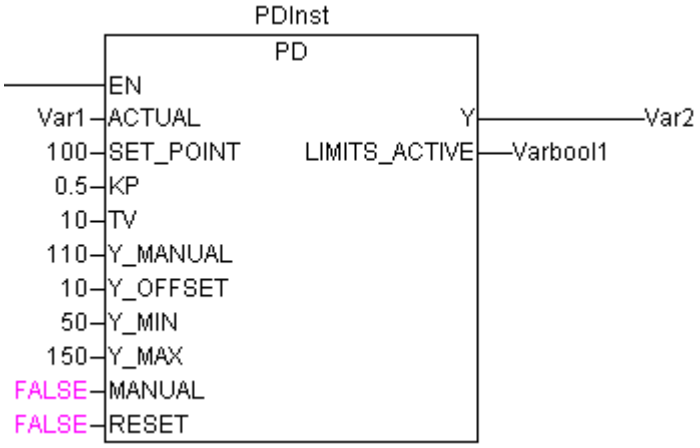
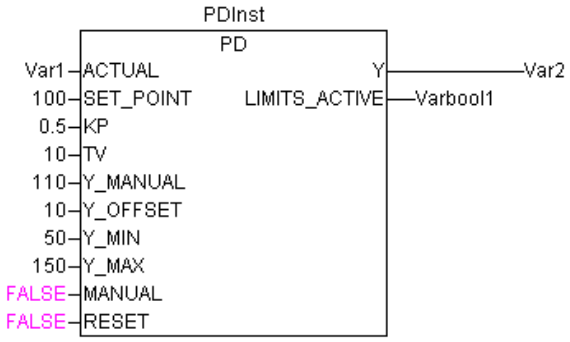
- ✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	PDInst		PD		
0002	Var1		REAL		
0003	Var2		REAL		
0004	Varbool1		BOOL		

编程语言	程　序

LD	
ST	PDInst(ACTUAL := Var1, SET_POINT := 100, KP := 0.5, TV := 10, Y_MANUAL := 110, Y_OFFSET := 10, Y_MIN := 50, Y_MAX := 150, MANUAL := FALSE, RESET := FALSE); Varbool1:=PDInst.LIMITS_ACTIVE; Var2:=PDInst.Y;
IL	CAL PDInst(ACTUAL := Var1, SET_POINT := 100, KP := 0.5, TV := 10, Y_MANUAL := 110, Y_OFFSET := 10, Y_MIN := 50, Y_MAX := 150, MANUAL := FALSE, RESET := FALSE) LD PDInst.LIMITS_ACTIVE ST Varbool1 LD PDInst.Y ST Var2
FBD	

调整波形如下图所示。

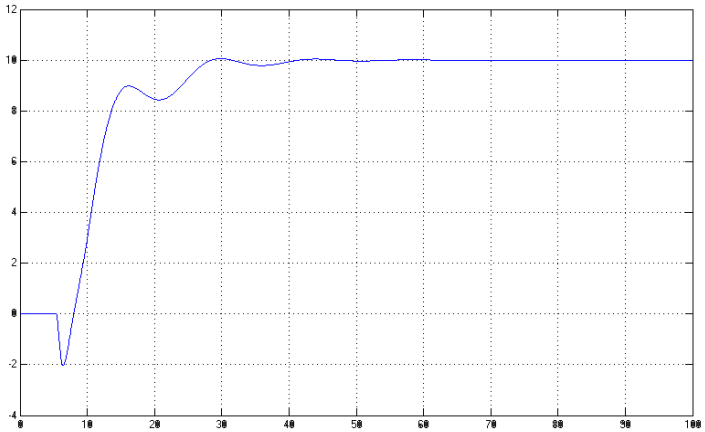


图 4-18-1 PD 调整的波形

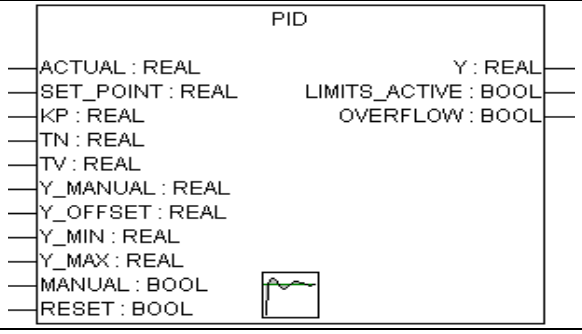
4.18.2 PID——比例积分微分控制器

- **功能**：该指令为比例积分微分控制器。当 TV=0 时，PID 控制器作为 PI 控制器。
- **控制方程**：$Y = KP*(\Delta + \frac{1}{TN}*\int \Delta(t)dt + TV*\frac{\sigma\Delta}{\sigma t}) + Y\_OFFSET$

  $\Delta = SET\_POINT - ACTUAL$

  该指令会自动计算$\int \Delta(t)dt$ 与$\frac{\sigma\Delta}{\sigma t}$，用户无需考虑。

- **指令示例**：

梯形图语言示例


- **参数说明**：

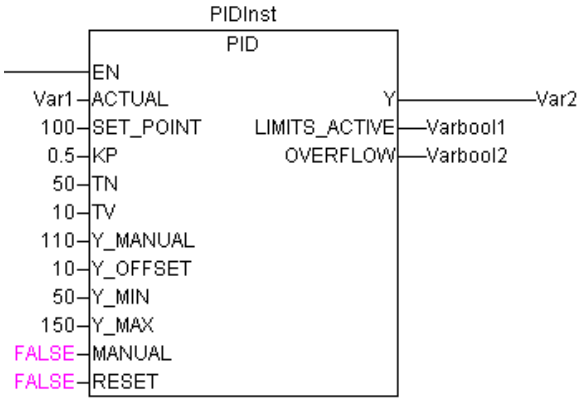
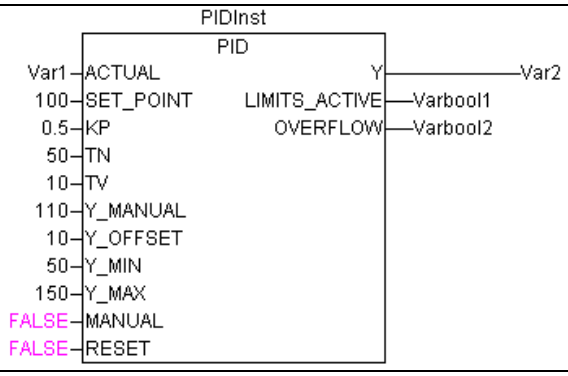
输入参数	数据类型	功能描述	参数值说明
ACTUAL	REAL	测量值	
SET_POINT	REAL	设定值	
KP	REAL	比例系数	
TN	DWORD	积分时间	单位为秒（s）
TV	DWORD	微分时间	单位为秒（s）
Y_MANUAL	REAL	手动值	MANUAL=TRUE 时，Y= Y_MANUAL
Y_OFFSET	REAL	输出值的偏移量	
Y_MIN	REAL	输出值的最小值	
Y_MAX	REAL	输出值的最大值	
MANUAL	BOOL	手自动选择	值为 TRUE 时为手动调节，值为 FALSE 时为自动调节
RESET	BOOL	重置	值为 TRUE 时重置该控制器，正常运行时应置 FALSE
输出参数	**数据类型**	**功能描述**	**参数值说明**
Y	REAL	输出值	
LIMITS_ACTIVE	BOOL	输出超限标志	输出值超限时等于 TRUE，即超出（Y_MIN，Y_MAX）
OVERFLOW	BOOL	输出溢出标志	积分溢出时等于 TRUE

- ✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	PIDInst		PID		
0002	Var1		REAL		
0003	Var2		REAL		
0004	Varbool1		BOOL		
0005	Varbool2		BOOL		

编程语言	程　序
LD	
ST	PIDInst(ACTUAL := Var1, SET_POINT := 100, KP := 0.5, TN := 50, TV := 10, Y_MANUAL := 110, Y_OFFSET := 10, Y_MIN := 50, Y_MAX := 150, MANUAL := FALSE, RESET := FALSE); Varbool1:=PIDInst.LIMITS_ACTIVE; Varbool2:=PIDInst.OVERFLOW; Var2:=PIDInst.Y;
IL	CAL PIDInst(ACTUAL := Var1, SET_POINT := 100, KP := 0.5, TN := 50, TV := 10, Y_MANUAL := 110, Y_OFFSET := 10, Y_MIN := 50, Y_MAX := 150, MANUAL := FALSE, RESET := FALSE) LD　　PIDInst.LIMITS_ACTIVE ST　　Varbool1 LD　　PIDInst.OVERFLOW ST　　Varbool2 LD　　PIDInst.Y ST　　Var2
FBD	

调整波形如下图所示。

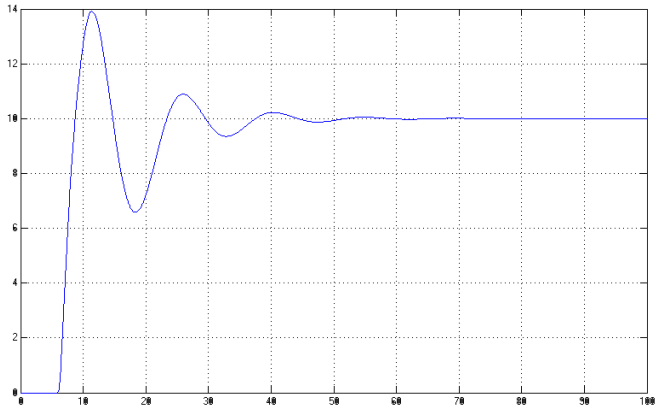


图 4-18-2 PID 调整的波形

4.18.3 PID_FIXCYCLE——比例积分微分控制器

- **功能**：该指令为比例积分微分控制器。与前面介绍过的 PID 控制器对比，就是多了一个采样周期的参数 CYCLE，CYCLE 是一个 REAL 型的输入参数，是用来设定积分和微分的时间步长，单位是秒。控制方程和参数说明详见前面 PID 指令。

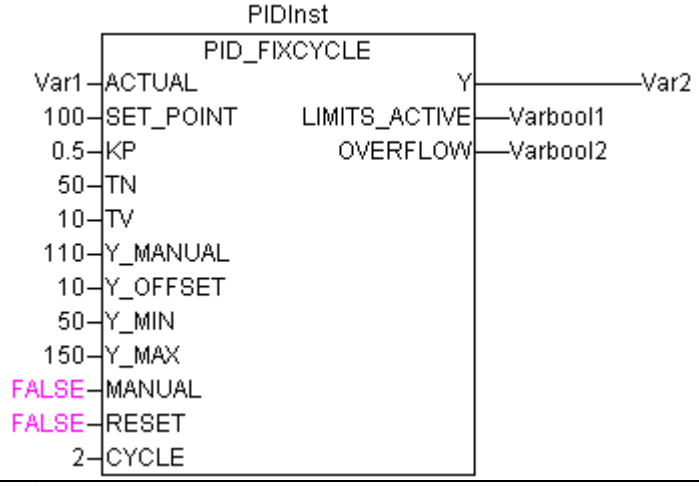
✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	Var1		REAL		
0002	Var2		REAL		
0003	Varbool1		BOOL		
0004	Varbool2		BOOL		
0005	PIDInst		PID_FIXCYCLE		

编程语言	程　序
LD	PIDInst PID_FIXCYCLE EN Var1 — ACTUAL　　Y — Var2 100 — SET_POINT　　LIMITS_ACTIVE — Varbool1 0.5 — KP　　OVERFLOW — Varbool2 50 — TN 10 — TV 110 — Y_MANUAL 10 — Y_OFFSET 50 — Y_MIN 150 — Y_MAX FALSE — MANUAL FALSE — RESET 2 — CYCLE
ST	PIDInst(ACTUAL := Var1, SET_POINT := 100, KP := 0.5, TN := 50, TV := 10, Y_MANUAL := 110, Y_OFFSET := 10, Y_MIN := 50, Y_MAX := 150, MANUAL := FALSE, RESET := FALSE, CYCLE := 2); Varbool1:=PIDInst.LIMITS_ACTIVE;

	Varbool2:=PIDInst.OVERFLOW; Var2:= PIDInst.Y;
IL	CAL PIDInst(ACTUAL := Var1, SET_POINT := 100, KP := 0.5, TN := 50, TV := 10, Y_MANUAL := 110, Y_OFFSET := 10, Y_MIN := 50, Y_MAX := 150, MANUAL := FALSE, RESET := FALSE, CYCLE := 2) LD PIDInst.LIMITS_ACTIVE ST Varbool1 LD PIDInst.OVERFLOW ST Varbool2 LD PIDInst.Y ST Var2
FBD	

调整波形如前面 PID 指令所示。

4.19 信号发生器指令（Util.lib）

4.19.1 BLINK——脉冲信号发生器

- **功能**：该指令产生脉冲信号。脉冲信号发生器开始工作，输出高电平时间为 TIMEHIGH，输出低电平时间为 TIMELOW，周期循环输出。
- **参数说明**：

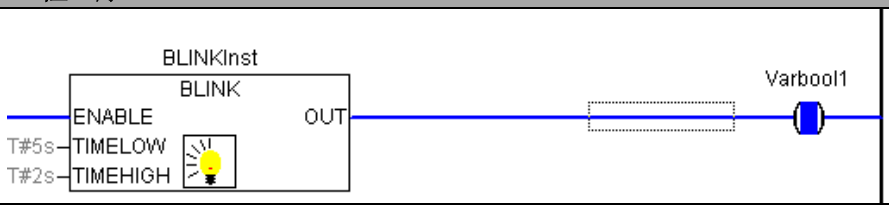
输入参数	数据类型	功能描述	参数值说明
ENABLE	BOOL	使能	TRUE 时，指令开始工作
TIMELOW	TIME	输出低电平时间	
TIMEHIGH	TIME	输出高电平时间	
输出参数	**数据类型**	**功能描述**	**参数值说明**
OUT	BOOL	脉冲信号输出值	

- ✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	Varbool1		BOOL		
0002	PIDInst		PID_FIXCYCLE		
0003	BLINKInst		BLINK		

编程语言	程　序
LD	
ST	BLINKInst(TIMELOW := T#5s,TIMEHIGH := T#2s); Varbool1:=BLINKInst.OUT;
IL	CAL BLINKInst(TIMELOW := T#5s,TIMEHIGH := T#2s) LD　　BLINKInst.OUT ST　　Varbool1
FBD	

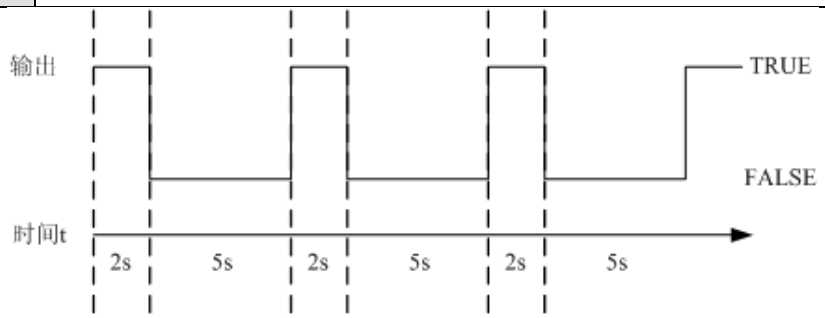


图 4-19-1 BLINK 输出波形

4.19.2　GEN——典型周期信号发生器

- **功能**：该指令用于生成典型的周期信号，如：三角波、零起点三角波、上升锯齿波、下降锯齿波、方波、正弦波和余弦波共七种。
- **参数说明**：

输入参数	数据类型	功能描述	参数值说明
MODE	GEN_MODE	指定要产生的信号类型	直接在 MODE 处输入 TRIANGLE、TRIANGLE_POS、SAWTOOTH_RISE、SAWTOOTH_FALL、RECTANGLE、SINUS、COSINUS，则产生对应的波形
		TRIANGLE	三角波
		TRIANGLE_POS	零起点三角波
		SAWTOOTH_RISE	上升锯齿波
		SAWTOOTH_FALL	下降锯齿波
		RECTANGLE	方波
		SINUS	正弦波
		COSINU	余弦波
BASE	BOOL	循环方式选择	当 BASE 为 TRUE 时，信号发生器与定义的循环周期有关。当 BASE 为 FALSE 时，信号发生器与特定的发生的个数有关
PERIOD	TIME	循环周期	
CYCLES	INT	发生的个数	

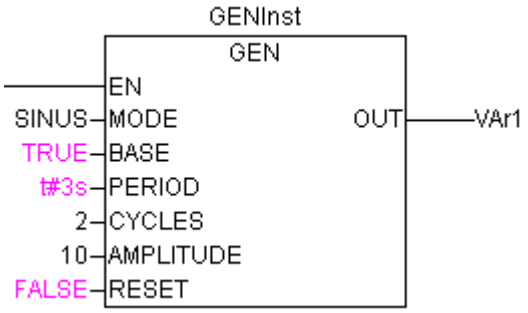
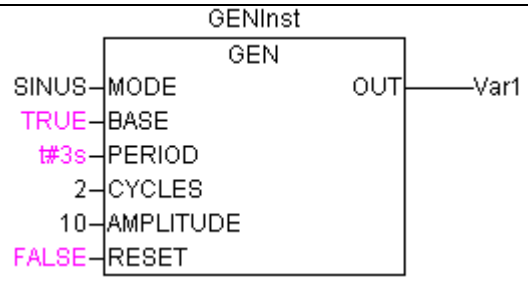
AMPLITUDE	INT	信号的振幅	
RESET	BOOL	初始化	当 RESET=TRUE 时，信号发生器被重新设置为 0
输出参数	**数据类型**	**功能描述**	**参数值说明**
OUT	INT	波形信号输出值	

✧ **指令使用举例**

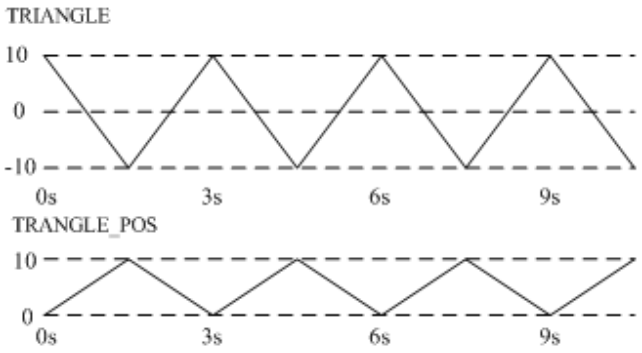
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	GENInst		GEN		
0002	Var1		INT		

编程语言	程　序
LD	
ST	GENInst(MODE:=SINUS, BASE:=TRUE, PERIOD:=T#3s, CYCLES:=2, AMPLITUDE:=10, RESET:=FALSE); Var1:=GENInst.OUT;
IL	CAL　GENInst(MODE:=SINUS, BASE:=TRUE, PERIOD:=T#3s, CYCLES:=2, AMPLITUDE:=10, RESET:=FALSE) LD　　GENInst.OUT ST　　Var1
FBD	

根据 MODE 输入不同，产生的波形如图 4-20-2 所示。



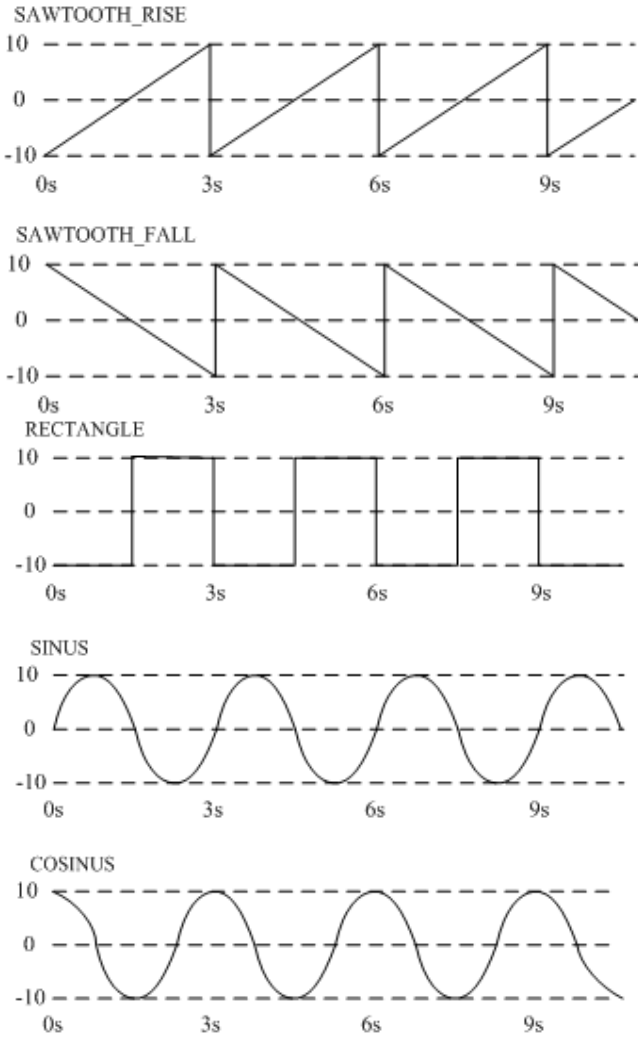


图 4-19-2 GEN 不同 MODE 下的波形输出

4.20 函数操纵器指令（Util.lib）

4.20.1 CHARCURVE——特征曲线

- **功能**：输入的 POINT 类型数组 P[0..N-1]在 XY 坐标图上定义了一条曲线，输入值 IN 为坐标图上的 X 轴上的点，输出值 OUT 为坐标图上该曲线所对应的 Y 轴的值。
- **参数说明：**

输入参数	数据类型	功能描述	参数值说明
IN	INT	输入坐标图上 X 轴的值	
N	BYTE	指定义曲线所使用数组中的点数	
P	ARRAY[0..n] OF POINT	（2≤n≤11）	用来在 XY 坐标上定义曲线
输出参数	**数据类型**	**功能描述**	**参数值说明**
OUT	INT	输出值	坐标图上曲线对应的 Y 轴的值
ERR	BYTE	显示错误类型	ERR=1：数组中的点 P[0]..P[N-1]中的 X 值有错误。 ERR=2：输入值 IN 不在 P[0].X 和 P[N-1].X 之间，即超出了数组

			定义曲线的 X 轴的范围。此时 OUT 输出 IN 包含在限制值 P[0].Y 和 P[N-1].Y 之间所对应的数据。 ERR=4：输入 N 小于 2，或者大于 11。
P	BYTE	N 输出值	

✧ **指令使用举例**

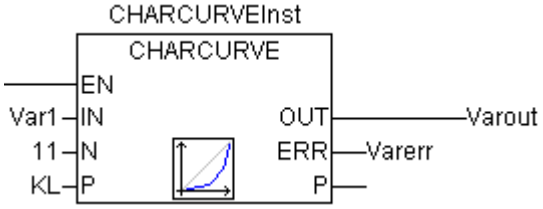
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT | RETAIN

	名称	地址	类型	初始值	注释
0001	CHARCURVEInst		CHARCURVE		
0002	Var1		INT		
0003	Varout		INT		
0004	Varerr		BYTE		
0005	KL		ARRAY[0..10] OF POINT	(X:=0,Y:=0),(X:=250,Y:=50),(X:=500,Y:=150),(X:=750,Y:=400),7((X:=1000,Y:=1000));	

编程语言 | **程　序**

LD



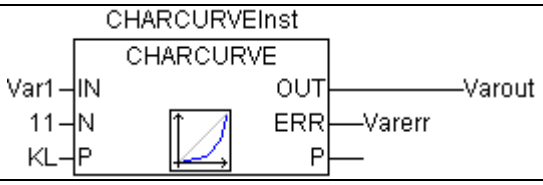
ST

```
CHARCURVEInst(IN := Var1, N := 11, P := KL);
Varerr:=CHARCURVEInst.ERR;
Varout:=CHARCURVEInst.OUT;
```

IL

```
CAL     CHARCURVEInst(IN := Var1, N := 11, P := KL)
LD      CHARCURVEInst.ERR
ST      Varerr
LD      CHARCURVEInst.OUT
ST      Varout
```

FBD



程序说明：

当指令执行时，根据输入值的变化，对应的输出值如图 4-21-1，坐标曲线由数组 KL 确定，输入 IN 为 X 轴上的值，输出 OUT 为该曲线对应的 Y 轴上的值。

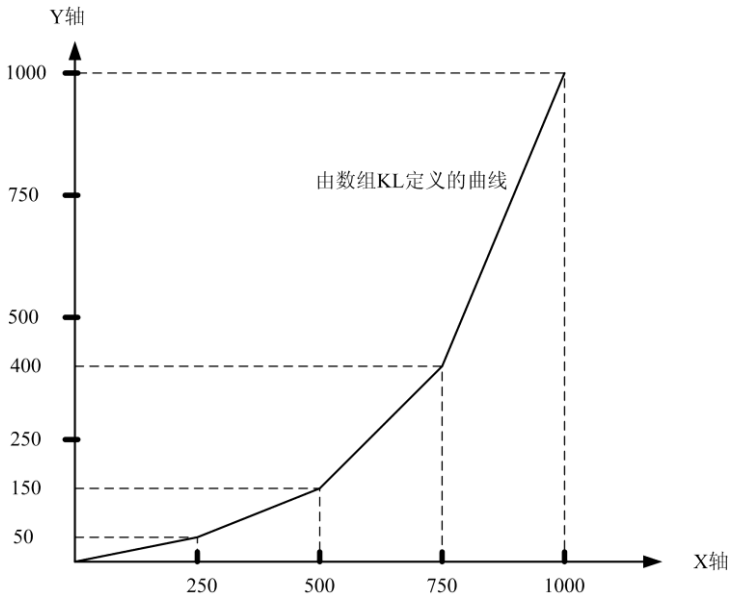


图 4-20-1 特征曲线的输出

4.20.2 RAMP_INT——整型限速

- **功能**：限制整型输入函数的升降速度。
- **参数说明**：

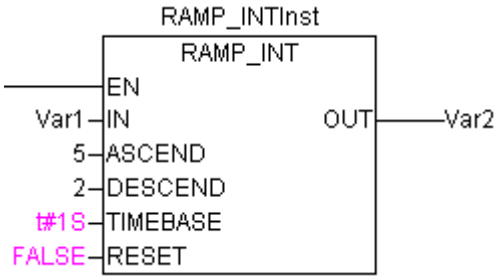
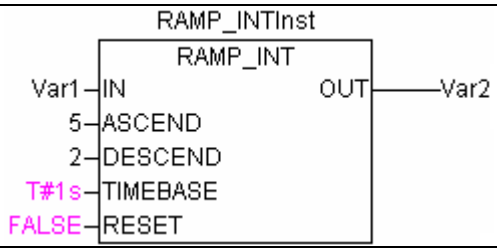
输入参数	数据类型	功能描述	参数值说明
IN	INT	目标值	若当前设定值大于先前值，则按照设定的时间和上升速率进行加法运算，由 OUT 即时输出，若当前设定值小于先前值，则按照设定的时间和下降速率进行减法运算，由 OUT 即时输出
ASCEND	INT	上升的增量	在规定时间（TIMEBASE）内上升的数量
DESCEND	INT	下升的增量	在规定时间（TIMEBASE）内下降的数量
TIMEBASE	TIME	时间基数	规定上升或者下降的速率
RESET	BOOL	初始化	设置为 TRUE 时，RAMP_INT 被重新初始化
输出参数	**数据类型**	**功能描述**	**参数值说明**
OUT	INT	即时数据输出	

- **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	RAMP_INTInst		RAMP_INT		
0002	Var1		INT		
0003	Var2		INT		

编程语言	程　序
LD	
ST	RAMP_INTInst(IN := Var1, ASCEND := 5, DESCEND := 2, TIMEBASE := T#1000ms, RESET := FALSE); Var2:=RAMP_INTInst.OUT;
IL	CAL　RAMP_INTInst(IN := Var1, ASCEND := 5, DESCEND := 2, TIMEBASE := T#1000ms, RESET := FALSE) LD　RAMP_INTInst.OUT ST　Var2
FBD	

程序说明：

当指令执行时，根据输入值 IN 的变化，对应的输出值如图 4-21-2 所示。

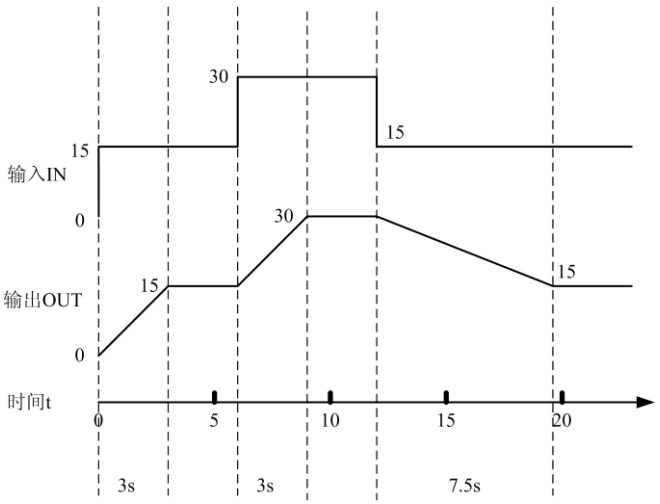


图 4-20-2　整型限速的输入输出关系

4.20.3　RAMP_REAL——实型限速

- **功能**：限制实型输入函数的升降速度。
- **参数说明：**

输入参数	数据类型	功能描述	参数值说明
IN	REAL	目标值	若当前设定值大于先前值，则按照设定的时间和上升速率进行加法运算，由 OUT 即时输出，若当前设定值小于

			先前值，则按照设定的时间和下降速率进行减法运算，由 OUT 即时输出
ASCEND	REAL	上升的增量	在规定时间（TIMEBASE）内上升的数量
DESCEND	REAL	下升的增量	在规定时间（TIMEBASE）内下降的数量
TIMEBASE	TIME	时间基数	规定上升或者下降的速率
RESET	BOOL	初始化	设置为 TRUE 时，RAMP_INT 被重新初始化
输出参数	**数据类型**	**功能描述**	**参数值说明**
OUT	REAL	即时数据输出	

指令使用举例及输出图请参见 RAMP_INT 指令所述。

4.21 模拟量处理指令（Util.lib）

4.21.1 HYSTERESIS——滞后

- **功能**：该指令的输入包括三个 INT 类型的数值 IN、HIGH 和 LOW。如果 IN 小于下限值 LOW，OUT 为 TRUE，保持至 IN 大于上限值 HIGH。此时，OUT 由 TRUE 变为 FALSE，保持至 IN 小于下限值 LOW。OUT 由 FALSE 变为 TRUE，保持至 IN 大于上限值 HIGH，OUT 变为 FALSE，如此循环。
- **参数说明：**

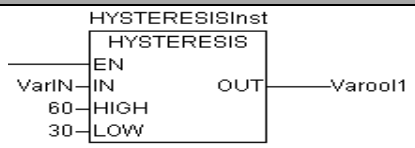
输入参数	**数据类型**	**功能描述**	**参数值说明**
IN	INT	输入值	
HIGH	INT	上限值	
LOW	INT	下限值	
输出参数	**数据类型**	**功能描述**	**参数值说明**
OUT	BOOL	输出值	低下限值后为 TRUE，直到上限值后为恢复 FALSE

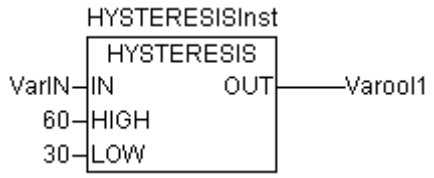
✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	HYSTERESISInst		HYSTERESIS		
0002	Varool1		BOOL		
0003	VarIN		INT		

编程语言	**程　序**
LD	
ST	HYSTERESISInst(IN := VarIN, HIGH := 60, LOW := 30); Varool1:=HYSTERESISInst.OUT;
IL	CAL　HYSTERESISInst(IN := VarIN, HIGH := 60, LOW := 30) LD　HYSTERESISInst.OUT ST　Varool1

FBD	

程序说明：

当指令执行时，根据输入值的变化，对应的输出值如图 4-22-1 所示。

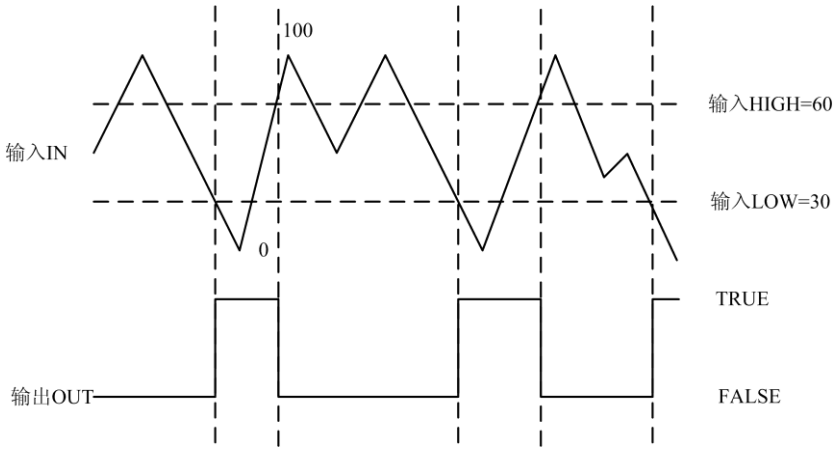


图 4-21-1 滞后指令的输入输出关系

4.21.2 LIMITALARM——上下限报警

- **功能**：如果 IN 超出上限 HIGH，则 O 为 TRUE，U 和 IL 为 FALSE。如果 IN 低于下限 LOW，则 U 为 TRUE，O 和 IL 为 FALSE。如果 IN 在下限 LOW 和上限 HIGH 之间，则 IL 为 TRUE，O 和 U 为 FALSE。
- **参数说明：**

输入参数	数据类型	功能描述	参数值说明
IN	INT	输入值	
HIGH	INT	上限值	
LOW	INT	下限值	
输出参数	**数据类型**	**功能描述**	**参数值说明**
O	BOOL	输出值	
U	BOOL	输出值	
IL	BOOL	输出值	

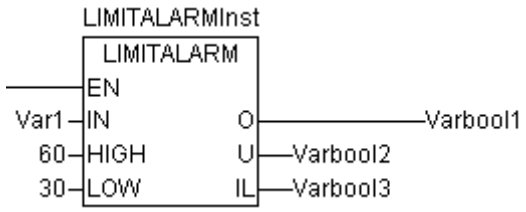
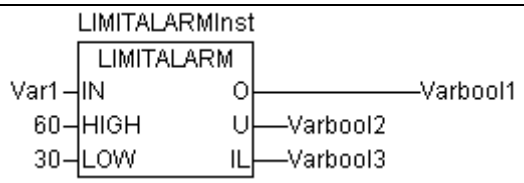
✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	LIMITALARMInst		LIMITALARM		
0002	Var1		INT		
0003	Varbool1		BOOL		
0004	Varbool2		BOOL		
0005	Varbool3		BOOL		

编程语言	程　序

LD	
ST	LIMITALARMInst(IN := Var1, HIGH := 60, LOW := 30); Varbool2:=LIMITALARMInst.U; Varbool3:= LIMITALARMInst.IL; Varbool1:= LIMITALARMInst.O;
IL	CAL LIMITALARMInst(IN := Var1, HIGH := 60, LOW := 30) LD LIMITALARMInst.U ST Varbool2 LD LIMITALARMInst.IL ST Varbool3 LD LIMITALARMInst.O ST Varbool1
FBD	

上例中，当指令执行时，根据输入值的变化，对应的输出值如下图 4-22-2 所示。

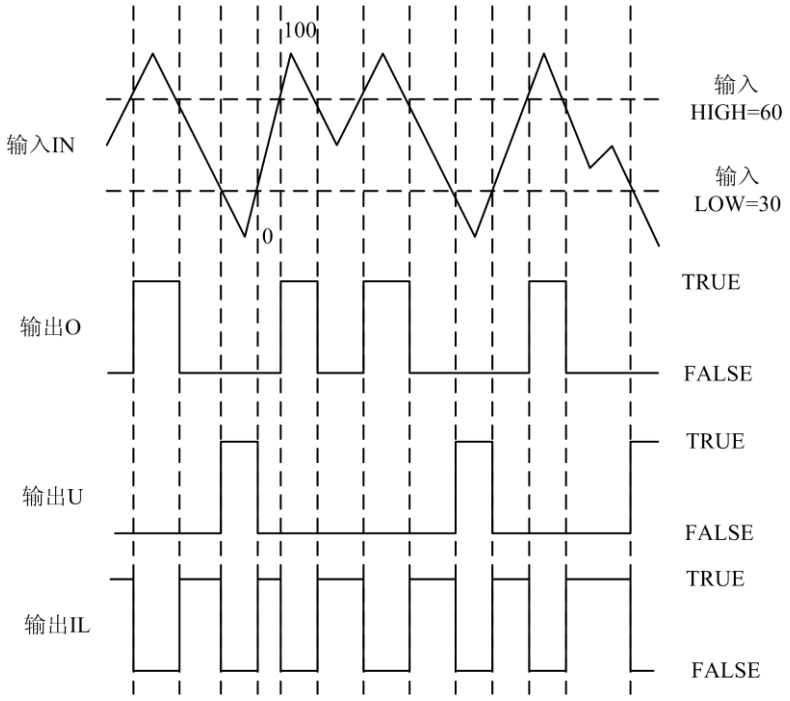


图 4-21-2 上下限报警指令的输入输出关系

4.22 双稳态指令（Standard.lib）

4.22.1 SR——置位优先双稳态器

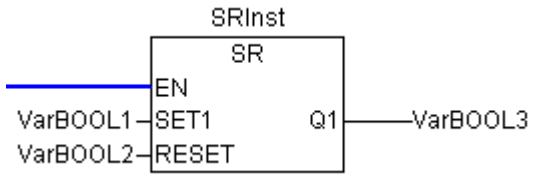
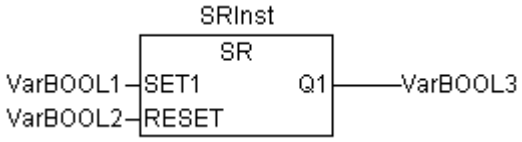
- **功能**：置位双稳态触发器，置位优先。
- **逻辑关系**：Q1=(NOT RESET AND Q1) OR SET1
  其中 SET1 为置位信号，RESET 为复位信号。
- **输入/输出数据类型**： BOOL。

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	SRInst		SR		
0002	VarBool1		BOOL		
0003	VarBool2		BOOL		
0004	VarBool3		BOOL		

编程语言	程　序
LD	 说明：VarBOOL3＝（NOT VarBOOL2 AND VarBOOL3）OR VarBOOL1
ST	SRInst(SET1:= VarBOOL1 , RESET:=VarBOOL2); VarBOOL3 := SRInst.Q1 ;
IL	CALSRInst(SET1:= VarBOOL1, RESET:= VarBOOL2) LD　　SRInst.Q1 ST　　VarBOOL3
FBD	

✧ **指令的真值表**

指令	SET1	RESET	输出
SR	0	0	保持原状态
	1	0	1
	0	1	0
	1	1	1

4.22.2 RS——复位优先双稳态器

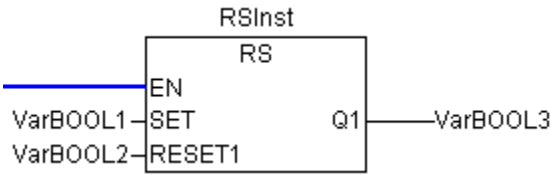
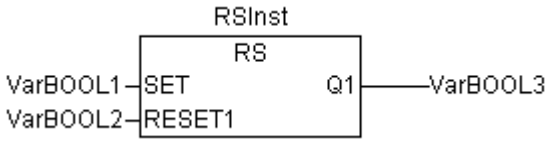
- **功能**：复位双稳态触发器，复位优先。
- **逻辑关系**：Q1=NOT RESET1 AND (Q1 OR SET)
  其中 SET 为置位信号，RESET1 为复位信号。
- **输入/输出数据类型**： BOOL

✧ 指令使用举例

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	RSInst		RS		
0002	VarBool1		BOOL		
0003	VarBool2		BOOL		
0004	VarBool3		BOOL		

编程语言	程　序
LD	 说明：VarBOOL3＝NOT VarBOOL2 AND (VarBOOL3 OR VarBOOL1)
ST	RSInst(SET:= VarBOOL1 , RESET1:=VarBOOL2); VarBOOL3 := RSInst.Q1 ;
IL	CAL RSInst(SET := VarBOOL1, RESET1 := VarBOOL2) LD RSInst.Q1 ST VarBOOL3
FBD	

指令的真值表：

指令	SET	RESET1	输出
RS	0	0	保持原状态
	1	0	1
	0	1	0
	1	1	0

4.23 触发器指令（Standard.lib）

触发器包含上升沿检测触发器 R_TRIG 和下降沿检测触发器 F_TRIG，分别用于检测信号上升沿和下降沿。

4.23.1 R_TRIG——上升沿检测触发器

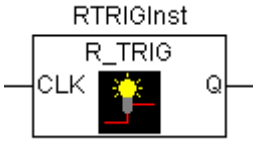
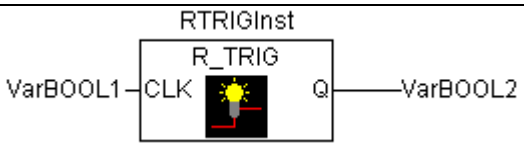
- **功能**：用于检测上升沿。
- **逻辑关系**：Q: = CLK AND NOT M;
  M: = CLK;
  M——中间变量，初始值为 TRUE；
  当 CLK 检测到上升沿，则 Q 输出一标准脉冲。
- **输入/输出数据类型**： CLK、Q 类型为 BOOL。

✧ 指令使用举例

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	RTRIGInst		R_TRIG		
0002	VarBOOL1		BOOL		
0003	VarBOOL2		BOOL		

编程语言	程　序
LD	
ST	RTRIGInst(CLK:= VarBOOL1); VarBOOL2 := RTRIGInst.Q;
IL	CAL　RTRIGInst(CLK := VarBOOL1) LD　RTRIGInst.Q ST　VarBOOL2
FBD	

4.23.2　F_TRIG——下降沿检测触发器

- **功能**：用于检测下降沿。
- **逻辑关系**：　Q := NOT CLK AND NOT M;

  M: = NOT CLK;

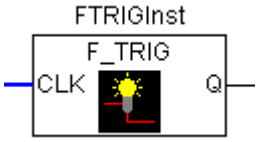
  M——中间变量，初始值为 FALSE；
  当 CLK 检测到下降沿时，Q 输出一标准脉冲。
- **输入/输出数据类型**：　BOOL。

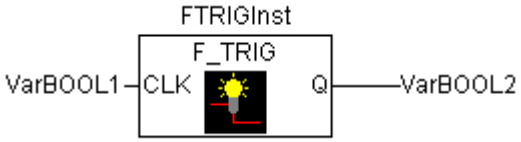
✧ 指令使用举例

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	FTRIGInst		F_TRIG		
0002	VarBOOL1		BOOL		
0003	VarBOOL2		BOOL		

编程语言	程　序
LD	
ST	FTRIGInst(CLK:= VarBOOL1); VarBOOL2 := FTRIGInst.Q;
IL	CAL　FTRIGInst(CLK := VarBOOL1) LD　FTRIGInst.Q

	ST　　VarBOOL2
FBD	

4.24 计数器（Standard.lib）

计数器包括递增计数器 CTU、递减计数器 CTD 和增减计数器 CTUD。其中在仿真模式下 PV 的值必须小于 32768。

4.24.1 CTU——递增计数器

- **功能**：实现对上升沿脉冲的递增计数。
- **参数说明：**

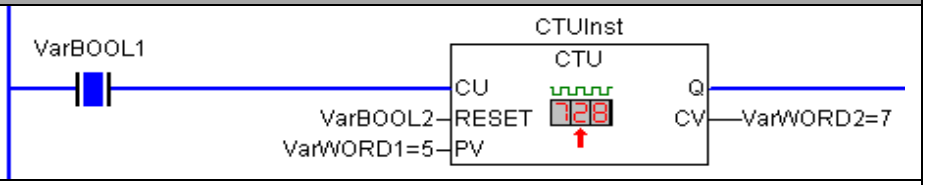
输入参数	数据类型	功能描述	参数值说明
CU	BOOL	计数输入	当 CU 有从 FALSE 到 TRUE 的上升沿，CV 加 1
RESET	BOOL	初始化	设置为 TRUE 时，CTU 被重新初始化
PV	WORD	计数器设定值	0-65535
输出参数	**数据类型**	**功能描述**	**参数值说明**
Q	BOOL	计数标志输出	当 CV 大于或等于上限 PV 时，Q 为 TRUE
CV	WORD	当前计数值	CV 超过 PV 后，仍会保持计数，直到 65535，如果这时再来上升沿脉冲 CV 保持在 65535 不变。

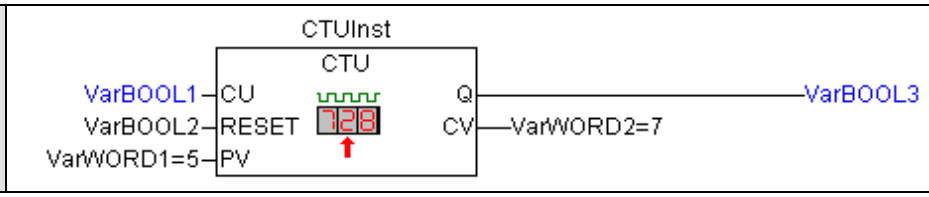
- **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	CTUInst		CTU		
0002	VarWORD1		WORD		
0003	VarWORD2		WORD		
0004	VarBOOL1		BOOL		
0005	VarBOOL2		BOOL		
0006	VarBOOL3		BOOL		

编程语言	程　序
LD	
ST	CTUInst(CU:= VarBOOL1, RESET:=VarBOOL2 , PV:= VarWORD1); VarBOOL3 := CTUInst.Q; VarWORD2 := CTUInst.CV;
IL	CAL　CTUInst(CU := VarBOOL1, RESET := VarBOOL2, PV :=VarWORD1) LD　　CTUInst.Q ST　　VarBOOL3 LD　　CTUInst.CV ST　　VarWORD2

FBD	

4.24.2 CTD——递减计数器

- **功能**：用户设定计数器初始值，当输入为一上升沿信号时，实现计数器递减计数。
- **参数说明**：

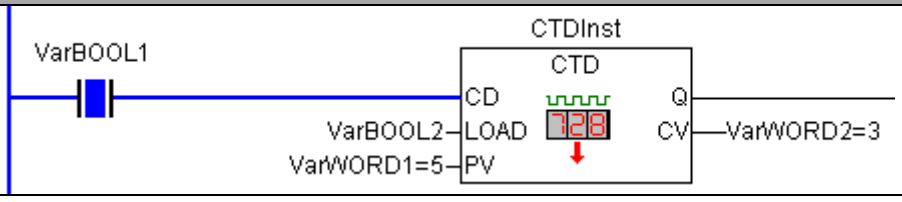
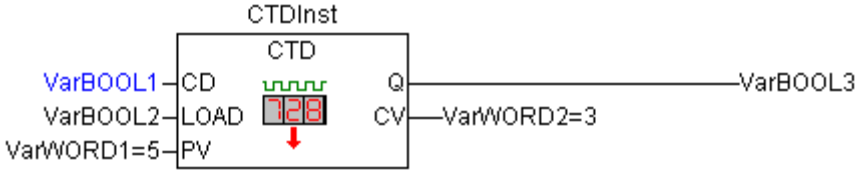
输入参数	数据类型	功能描述	参数值说明
CD	BOOL	计数输入	当 CD 有从 FALSE 到 TRUE 的上升沿，若 CV 大于 0，CV 减 1（CV 的值不小于 0）
LOAD	BOOL	初始化	LOAD 为 TRUE 时，计数变量 CV 装载为 PV
PV	WORD	计数器设定值	0-65535
输出参数	**数据类型**	**功能描述**	**参数值说明**
Q	BOOL	计数标志输出	当 CV 等于 0 时，Q 为 TRUE
CV	WORD	当前计数值	

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	CTDInst		CTD		
0002	VarWORD1		WORD		
0003	VarWORD2		WORD		
0004	VarBOOL1		BOOL		
0005	VarBOOL2		BOOL		
0006	VarBOOL3		BOOL		

编程语言	程　序
LD	
ST	CTDInst(CD:= VarBOOL1, LOAD:=VarBOOL2 , PV:= VarWORD1); VarBOOL3 := CTDInst.Q; VarWORD2 := CTDInst.CV;
IL	CAL　CTDInst(CD := VarBOOL1, LOAD := VarBOOL2, PV :=VarWORD1) LD　CTDInst.Q ST　VarBOOL3 LD　CTDInst.CV ST　VarWORD2
FBD	

4.24.3 CTUD——递增递减计数器

- **功能**：既可实现成单纯的递增、递减计数操作，又可实现继时的递增递减计数或递减递增计数功能。
- **参数说明：**

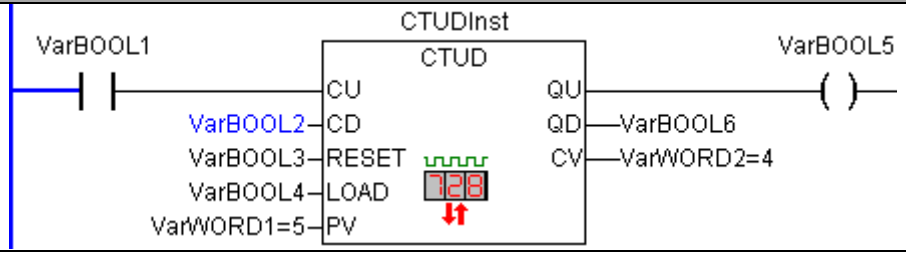
输入参数	数据类型	功能描述	参数值说明
CU	BOOL	计数输入	当 CU 有从 FALSE 到 TRUE 的上升沿，CV 加 1
CD	BOOL	计数输入	当 CD 有从 FALSE 到 TRUE 的上升沿，若 CV 大于 0，CV 减 1（CV 的值不小于 0）
RESET	BOOL	复位输入	RESET 为 TRUE 时，计数变量 CV 初始化为 0
LOAD	BOOL	初始化	LOAD 为 TRUE 时，计数变量 CV 装载为 PV
PV	WORD	计数器设定值	0-65535
输出参数	**数据类型**	**功能描述**	**参数值说明**
QU	BOOL	计数标志输出	当 CV 等于 PV 时，QU 为 TRUE
QD	BOOL	计数标志输出	当 CV 等于 0 时，QD 为 TRUE
CV	WORD	当前计数值	

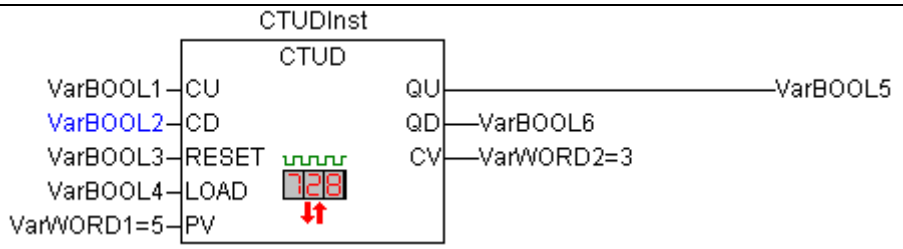
- ✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	CTUDInst		CTUD		
0002	VarBOOL1		BOOL		
0003	VarBOOL2		BOOL		
0004	VarBOOL3		BOOL		
0005	VarBOOL4		BOOL		
0006	VarBOOL5		BOOL		
0007	VarBOOL6		BOOL		
0008	VarWORD1		WORD		
0009	VarWORD2		WORD		

编程语言	程　序
LD	
ST	CTUDInst(CU:=VarBOOL1,CD:=VarBOOL2,RESET:=VarBOOL3, LOAD:=VarBOOL4,PV:=VarWORD1) VarBOOL5 := CTUDInst.QU VarBOOL6 := CTUDInst.QD VarWORD2 := CTUDInst.CV
IL	CAL　CTUDInst(CU:=VarBOOL1,CD:=VarBOOL2, RESET:=VarBOOL3,LOAD:=VarBOOL4,PV:=VarWORD1) LD　CTUDInst.QU

	ST VarBOOL5 LD CTUDInst.QD ST VarBOOL6 LD CTUDInst.CV ST VarWORD2
FBD	

4.25 定时器（Standard.lib）

定时器包括普通定时器 TP、通电延时定时器 TON、断电延时定时器 TOF 和实时时钟 RTC，下面分别描述。

4.25.1 TP——普通定时器

- **功能**：普通定时器，也称脉冲定时器。通电后，定时器输出端 Q 立即置 1，只有计时到设定时间后 Q 端才断开。
- **参数说明：**

输入参数	数据类型	功能描述	参数值说明
IN	BOOL	定时器初始化信号	当 IN 变成 TRUE 时，Q 置 1，ET 以毫秒开始计时直到 ET 等于 PT，然后 Q 置 0，ET 保持常数
PT	TIME	定时时间值	
输出参数	**数据类型**	**功能描述**	**参数值说明**
Q	BOOL	定时器输出	若 IN 为 FALSE，则 Q 为 FALSE，ET 为 0。当 IN 为 TRUE 时，定时器开始工作，Q 为 TRUE，在 ET 小于等于 PT 前，IN 无效，ET 等于 PT 时，Q 为 FALSE
ET	TIME	当前时间值	当 IN 变成 TRUE 时，ET 以毫秒计时直到 ET 等于 PT，然后 ET 保持常数。计时完毕后，当 IN 为 FALSE 时，ET 等于 0

- TP 时间顺序图

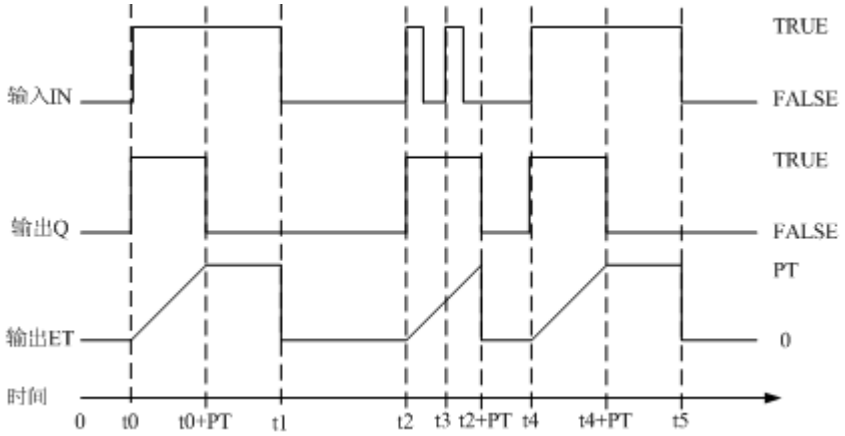


图 4-25-1 TP 时序图

✧ 指令使用举例

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	TPInst		TP		
0002	in		BOOL		
0003	pt		TIME		
0004	et		TIME		
0005	q		BOOL		

编程语言	程　序
LD	(*当IN为TRUE时，定时器开始工作，Q为TRUE，在ET小于等于PT前，IN无效，ET等于PT时，Q为FALSE*) TPInst in — IN TP Q — q pt=T#1m0s0ms — PT ET — et=T#1m0s0ms

ST	TPInst(IN:= in,PT:=pt);
	q:=TPInst.Q;
	et:= TPInst.ET;

IL	CAL　TPInst(IN:= in,PT:=pt) LD　TPInst.Q ST　q LD　TPInst.ET ST　et
FBD	TPInst TP in — IN　Q — q pt=T#1m0s0ms — PT　ET — et=T#1m0s0ms

4.25.2 TON——通电延时定时器

- **功能**：通电延时定时器。通电后，定时器开始工作，计时到设定的时间后，定时器输出 Q 才置 1。
- **参数说明：**

输入参数	数据类型	功能描述	参数值说明
IN	BOOL	定时器初始化信号	当 IN 变成 TRUE 时，ET 以毫秒计时直到 ET 等于 PT，然后 Q 置 1，ET 保持常数
PT	TIME	定时时间值	
输出参数	**数据类型**	**功能描述**	**参数值说明**
Q	BOOL	定时器输出	当 IN 为 FALSE 时，Q 为 FALSE，ET 为 0。当 IN 为 TRUE 时，定时器开始工作，ET 等于 PT 时，Q 为 TRUE
ET	TIME	当前时间值	当 IN 变成 TRUE 时，ET 以毫秒计时直到 ET 等于 PT，然后 ET 保持常数。无论何时，当 IN 为 FALSE 时，ET 等于 0

➢ TON 时间顺序图

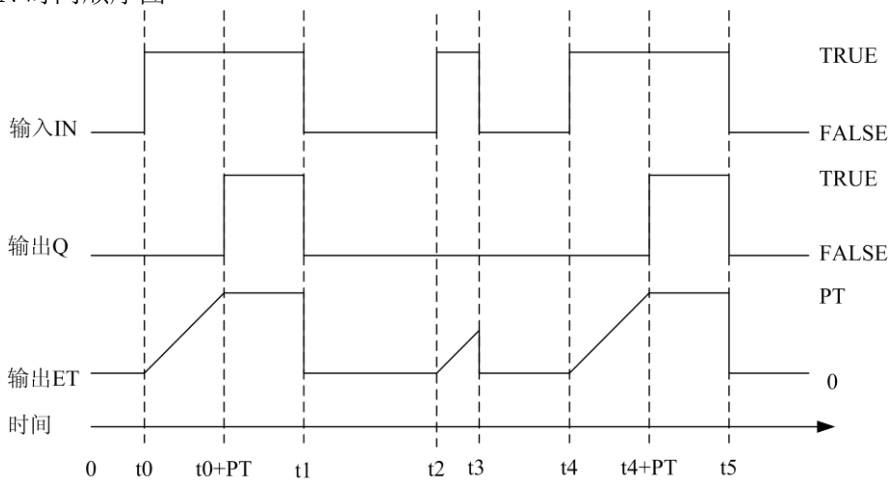


图 4-25-2 TON 时序图

✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	TONInst		TON		
0002	in		BOOL		
0003	pt		TIME		
0004	Q		BOOL		
0005	et		TIME		

编程语言	程　序
LD	(*IN=FALSE时，Q为FALSE，ET为0; IN=TRUR时，ET开始计时直到ET等于PT时，Q为TRUE*) TONInst in — TON (IN, PT, Q, ET) — q pt=T#1m0s0ms　et=T#1m0s0ms

ST	TONInst(IN:=in,PT:= pt);
	q:=TONInst.Q;
	et:= TONInst.ET;

IL	CAL　TONInst(IN:= in,PT:=pt) LD　TONInst.Q ST　q LD　TONInst.ET ST　et
FBD	TONInst in — IN　Q — q pt=T#1m0s0ms — PT　ET — et=T#1m0s0ms

4.25.3 TOF——断电延时定时器

- **功能**：断电延时定时器。通电后，定时器输出端 Q 立即置 1；断电后，定时器开始计时，Q 端保持 1，直到计时到设定的时间后，Q 端才断开。
- **参数说明**：

输入参数	数据类型	功能描述	参数值说明
IN	BOOL	定时器初始化信号	当 IN 由 TRUE 变成 FALSE 时，ET 以毫秒计时直到 ET 等于 PT，然后 Q 置 0，ET 保持常数
PT	TIME	定时时间值	
输出参数	**数据类型**	**功能描述**	**参数值说明**
Q	BOOL	定时器输出	当 IN 为 FALSE 且 ET 等于 PT 时，Q 为 FALSE。否则，Q 为 TRUE
ET	TIME	当前时间值	当 IN 为 FALSE 时，ET 以毫秒计数直到 ET 等于 PT，然后 ET 保持常数。无论何时，当 IN 为 TRUE 时，ET 等于 0，Q 为 TRUE

- TOF 时间顺序图

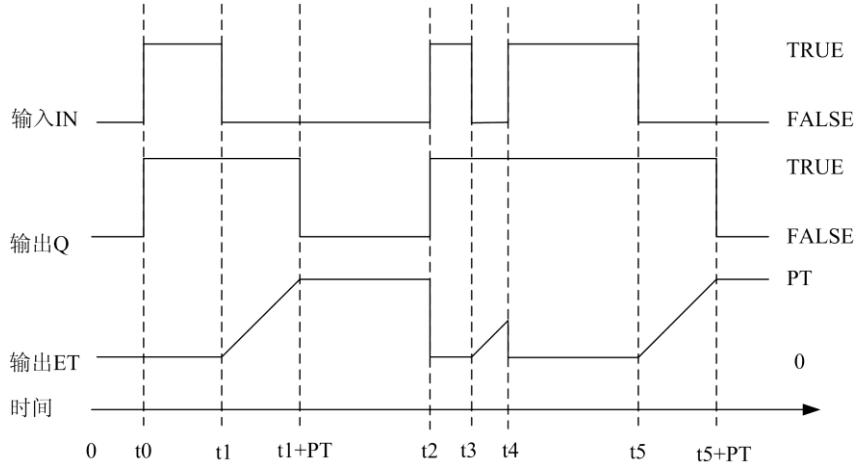


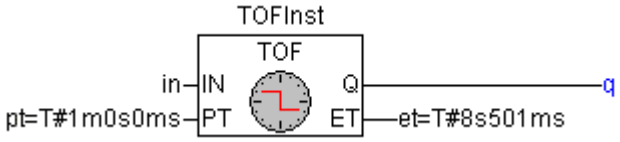
图 4-25-3 TOF 时序图

- **指令使用举例**

变量定义

	名称	地址	类型	初始值	注释
0001	TOFInst		TOF		
0002	in		BOOL		
0003	pt		TIME		
0004	Q		BOOL		
0005	et		TIME		

编程语言	程　序
LD	(*当IN由TRUE变成FALSE时，ET以毫秒计时直到ET等于PT，然后Q置0，ET保持常数*) a　tof　TOF　IN　PT　Q　ET　q　pt=T#1m0s0ms　et=T#6s101ms
ST	TOFInst(IN:=in,PT:= pt); q:=TOFInst.Q;

	et:= TOFInst.ET;
IL	CAL TOFInst(IN:= in,PT:=pt) LD TOFInst.Q ST q LD TOFInst.ET ST et
FBD	

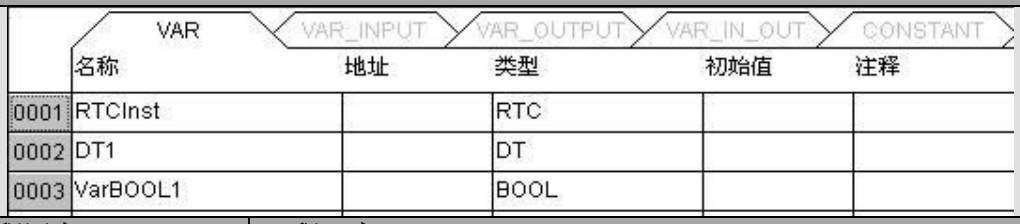
4.25.4 RTC——实时时钟

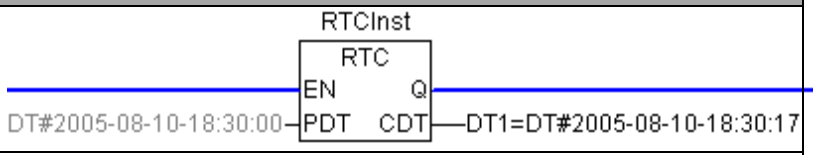
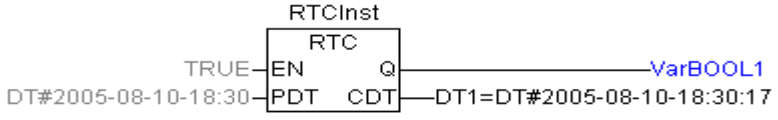
- **功能**：在给定时间启动，返回当前日期和时间。
- **参数说明**：

输入参数	数据类型	功能描述	参数值说明
EN	BOOL	使能信号	启动该指令
PDT	DT	时间基值	
输出参数	**数据类型**	**功能描述**	**参数值说明**
Q	BOOL	定时器输出	EN 为 FALSE 时，Q 为 FALSE，EN 为 TRUE 时，Q 为 TRUE
CDT	DT	当前时间值	EN 为 FALSE 时，CDT 为 1970-01-01-00:00:00，EN 为 TRUE 时，CDT 从 PDT 中的时间开始计时

✧ **指令使用举例**

变量定义



编程语言	程　序
LD	
ST	RTCInst(PDT:=DT#2005-08-10-18:30:31); DT1:=RTCInst.CDT; VarBOOL1:=RTCInst.Q;
IL	CAL RTCInst(PDT := DT#2005-08-10-18:30:31) LD RTCInst.CDT ST DT1 LD RTCInst.Q ST VarBOOL1
FBD	

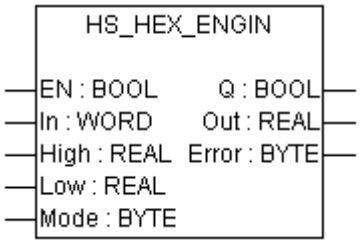
第5章 扩展指令

5.1 模拟量应用指令 HS_AnalogConvert.lib

5.1.1 HS_HEX_ENGIN—普通 AI 模块测量数据转工程量

- **功能**：将普通 AI 模块测量的现场信号(16 位二进制数)转化为可供系统计算的工程量；
- **指令示例**：

梯形图语言示例



- **参数说明**：

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能输入	当 EN=FALSE 时，停止转换，Q=FALSE。 当 EN=TRUE 时，开始转换，Q=TRUE， 将十六进制数对应于工程量的量程范围 Low-High。	FALSE
In	WORD	现场信号	普通 AI 模块测量的现场信号(16 位二进制数)	0
High	REAL	量程上限	工程量的量程上限	0
Low	REAL	量程下限	工程量的量程下限	0
Mode	BYTE	转换模式	从 16 进制数据到工程量数据的转换计算公式是根据不同的转换模式来确定的，详见转换模式说明。	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	使能输出	当 EN=FALSE 时，停止转换，Q=FALSE。 当 EN=TRUE 时，开始转换，Q=TRUE， 将十六进制数对应于工程量的量程范围 LOW-HIGH。	
Out	REAL	输出的工程量值	各量程意义下的工程量值	
Error	BYTE	错误信息	=0：转换模式正确，且工程量没有超越量程。 =1：转换模式错误，或工程量超越量程，此时工程量的值以量程的上限或下限代替。	

测量数据转换为工程量数据的转换模式说明：

转换模式	转换数据	量程下限	量程上限	输入信号类型
MODE=0	工程量	Low	High	输入电流信号：4~20mA
	十进制数	0	63243	
	十六进制数	0	F70B	
MODE=1	工程量	Low	High	输入电流信号：0~20mA
	十进制数	0	63688	
	十六进制数	0	F8C8	
MODE=2	工程量	Low	High	输入电压信号：0~5V 或 0~10V

	十进制数	0	63936	
	十六进制数	0	F9C0	
MODE=3	工程量	Low	High	输入电压信号：-10~10V
	十进制数	33568	31967	
	十六进制数	8320	7CDF	
MODE=4	工程量	Low	High	输入电压信号：-12~32mV 或-12~78mV
	十进制数	0	65535	
	十六进制数	0	FFFF	

✧ **指令使用举例**

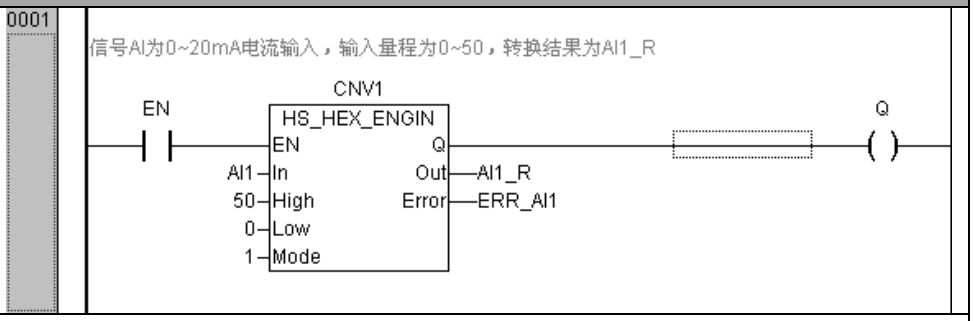
变量定义

VAR

	名称	地址	类型	初始值	注释
0001	EN		BOOL		
0002	CNV1		HS_HEX_ENGIN		
0003	AI1	%MW4	WORD		
0004	AI1_R		REAL		
0005	ERR_AI1		BYTE		
0006	Q		BOOL		

编程语言 | **程　序**

梯形图（LD）



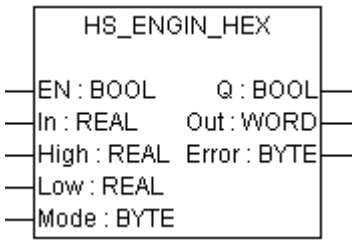
结构化文本（ST）

```
CNV1(EN:=EN,In:=AI1, High:=50, Low:=0, Mode:=1);
Q:=CNV1.Q;
AI1_R:=CNV1.Out;
ERR_AI1:=CNV1.Error;
```

5.1.2　HS_ENGIN_HEX—工程量转普通 AO 模块输出数据

- **功能**：将工程量计算结果转换为十六位二进制数据，供现场信号操作。
- **指令示例**：

梯形图语言示例



➢ 参数说明：

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能输入	当 EN=FALSE 时，停止转换，Q=FALSE。 当 EN=TRUE 时，开始转换，Q=TRUE，工程量的量程范围为 LOW-HIGH。	FALSE
In	REAL	需转换的工程量		0
High	REAL	量程上限	工程量的量程上限	0
Low	REAL	量程下限	工程量的量程下限	0
Mode	BYTE	转换模式	从工程量数据到输出量的转换计算公式是根据不同的转换模式来确定的，详见转换模式说明。	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	使能输出	初始值为 FALSE。 当 EN=FALSE 时，停止转换，Q=FALSE。 当 EN=TRUE 时，开始转换，Q=TRUE，将工程量的量程范围 LOW-HIGH。	
Out	WORD	输出的 16 位二进制数据		
Error	BOOL	错误信息	=0：转换模式正确，且工程量没有超越量程； =1：转换模式错误，或工程量超越量程，此时输出量以量程的上限或下限代替。	

从工程量数据到输出量的转换模式说明：

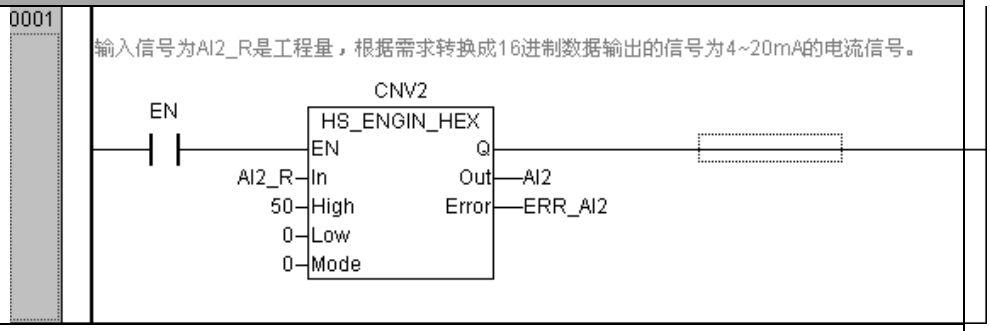
转换模式	转换数据	量程下限	量程上限	输出信号类型
MODE=0	工程量	Low	High	输出电流信号：4~20mA
	十进制数	0	65535	
	十六进制数	0	FFFF	
MODE=1	工程量	Low	High	输出电流信号：0~20mA
	十进制数	0	62414	
	十六进制数	0	F3CE	
MODE=2	工程量	Low	High	输出电压信号：0~5V 或 0~10V
	十进制数	0	63936	
	十六进制数	0	F9C0	
MODE=3	工程量	Low	High	输出电压信号：-10~10V
	十进制数	33568	31968	
	十六进制数	8320	7CE0	

✧ 指令使用举例

变量定义

VAR

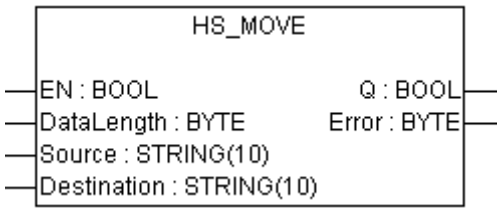
	名称	地址	类型	初始值	注释
0001	EN		BOOL		
0002	CNV2		HS_ENGIN_HEX		
0003	AI2		WORD		
0004	AI2_R		REAL		
0005	ERR_AI2		BYTE		

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	CNV2(EN:=EN,In:= AI2_R,High:=50,Low:=0,Mode:=0); AI2:=CNV2.Out; ERR_AI2:=CNV2.Error;

5.2 数据传送指令 HS_Move.lib

5.2.1 HS_MOVE——数据传送指令

- **功能**：将 M 区、I 区或 Q 区的数据从一个位置批量传送到另一个位置。支持按字节（BYTE）、字（WORD）或双字（DWORD）批量传送。一次传送数据个数的最大值为 255 个双字。
- **指令示例：**

梯形图语言示例


- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能输入	当 EN=FALSE 时，停止传送。 当 EN=TRUE 时，开始传送， 从数据源地址 SOURCE 开始，将长度为 LENGTH 的数据传送到从目的地址 DESTINATION 开始的连续地址中。	FALSE
DataLength	BYTE	传送数据的长度	0~255	0
Source	STRING(10)	数据源地址的起始地址	注意：该变量为字符串变量，其头尾需加单引号，并且，地址字符串必须以%开头，其中的字母必须大写，如‘%MB100’。	‘’
Destination	STRING(10)	数据目的地址的起始地址	注意：该变量为字符串变量，其头尾需加单引号，并且，地址字符串必须以%开头，其中的字母必须大写，如‘%MB200’。	‘’
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	使能输出	当 EN=FALSE 时，Q=FALSE。 当 EN=TRUE 时，Q=TRUE。	
Error	BYTE	错误信息	=0：数据传送成功 =1：数据传送出错	

数据源地址的首地址和目的地址的首地址使用字符串的形式描述，如 SOURCE:=‘%MB100’符合语法规则。功能块会解析字符串，判断出源地址和目的地址所在的数据区，以及应该按何种方式传送数据。

地址中字母 B、W 或 D 表示以何种方式传送数据，B 表示按字节（BYTE）传送，W 表示按字（WORD）传送，D 表示按双字（DWORD）传送。源地址和目的地址所描述的传送方式必须一致。例如，如果源地址是‘%MB’，表示按字节（BYTE）方式传送，目的地址不能是‘%MW’或‘%MD’。

✧ **指令使用举例**

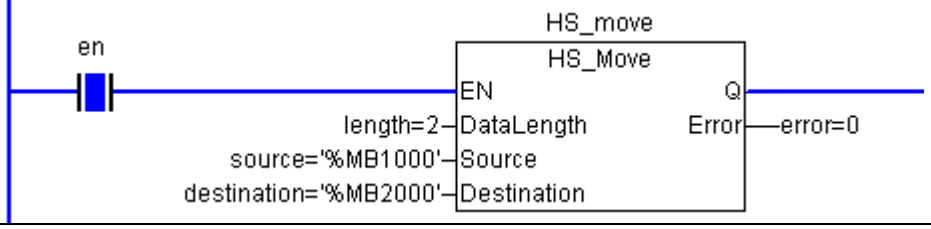
变量定义

VAR

	名称	地址	类型	初始值
0001	HS_Move		HS_Move	
0002	length		BYTE	2
0003	error		BYTE	
0004	Q		BOOL	
0005	en		BOOL	
0006	Source		STRING	'%MB1000'
0007	destination		STRING	'%MB2000'

编程语言 | **程　序**

梯形图（LD）



结构化文本（ST）

```
HS_Move(EN:=en,DataLength:=length,Source:=source,
    Destination:=destination,
    Error=>error);
```

提示：

可能出错的情况是：

- 输入无效的字符串。
- 源地址和目的地址所描述的传送方式不一致。
- 无效的数据区或者数据地址（注意：若地址字符串中的字母小写，也视为无效）。
- 地址越界。
- 源地址和目的地址相同。如果此时地址有效，则认为传送成功。

5.3 通讯指令 HS_Communication.Lib

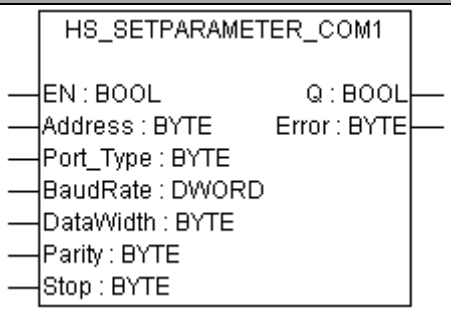
5.3.1 COM1 口通讯指令

- **注意**：在 PowerPro V4.3.1B 中，对 COM1 口通讯指令进行了完善和更新，如果用户的主控型号低于 LK202-B01, LK205-B01, LK207-A04, LK209-B01, LK210-B06 这几类，请到 Backup Library 中调用适合您主控的 HS_Communication.lib 库版本文件，具体操作方法请参见附录 3。

5.3.1.1 HS_SetParameter_COM1——COM1 口通讯参数设置

- **功能**：提供设置 COM1 串口通讯参数、选择通讯协议的用户接口。如果使能 Modbus 协议，默认支持 Modbus 从站功能，即这时可以响应 Modbus 主站的数据请求。
- **指令示例**：

梯形图语言示例



- **参数说明**：

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：上升沿有效	0
Address	BYTE	本站地址	范围 1-247	1
Port_Type	BYTE	串口类型	.0 协议选择 =1：自由协议 =0：Modbus 协议	0
BaudRate	DWORD	通讯的波特率（bps）	1200，2400，4800，9600，19200，38400，57600，115200。	115200
DataWidth	BYTE	数据位	=5：5 位数据位（Modbus 协议不支持） =6：6 位数据位（Modbus 协议不支持） =7：7 位数据位（Modbus 协议不支持） =8：8 位数据位（Modbus 协议只能选用该位数） 其它，非法	8
Parity	BYTE	校验方式	=0：无校验 =1：奇校验 =2：偶校验	0
Stop	BYTE	停止位位数	=1：1 位停止位 =2：2 位停止位 其它，非法	1
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	设置完成	0：未完成 1：已完成	
Error	BYTE	错误信息	=0：正确 =1：初始化串口错 =2：站地址（Address）错 =3：串口类型（Port_Type）错 =4：波特率（BautRate）错	

			=5：数据位（DataWidth）错 =6：校验方式（Parity）错 =7：停止位（Stop）错 =8：调用多个功能块	

提示：

- COM1 口通讯功能块**不可声明为 RETAIN 区变量**，不支持掉电保护，不支持冗余；
- COM1 口通讯**不支持多实例应用**，即在一个工程中每个 COM1 口通讯指令只能被调用一次；
- 对于输入参数如果不做任何设置，引脚为空(无变量定义)，则启用默认设定值；
- 由于 **EN** 是上升沿有效，正常时只设置一次，所以调试中如果本指令的输入参数发生改变需要进行全下载，或者在线下载（增量下载）后将 **EN** 置位，形成上升沿，以使新的参数生效。
- **EN** 是上升沿有效，正常时只设置一次，不可持续不断设置上升沿。

✧ **指令使用举例**

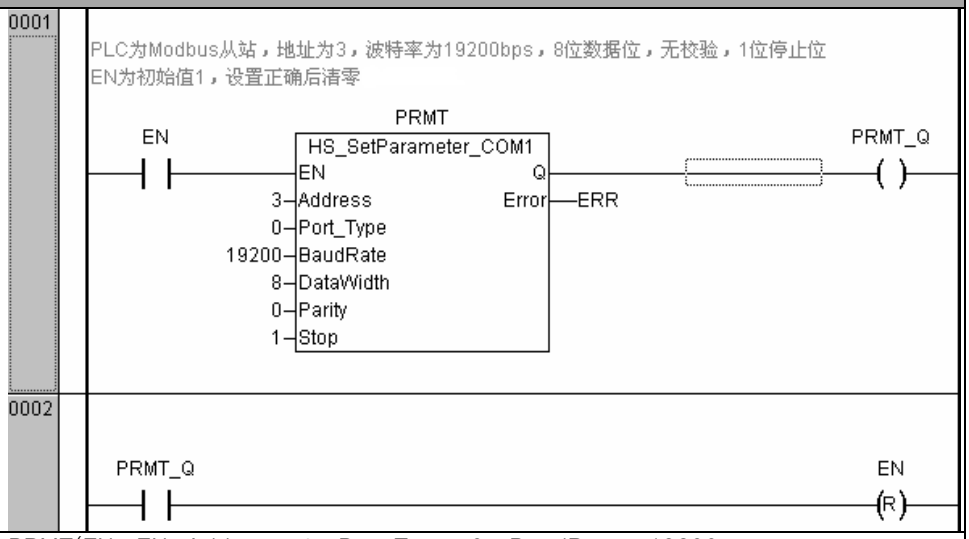
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	PRMT		HS_SetParameter_		
0002	EN		BOOL	1	使能
0003	PRMT_Q		BOOL		
0004	ERR		BYTE		

编程语言 | **程　序**

梯形图（LD）



结构化文本（ST）

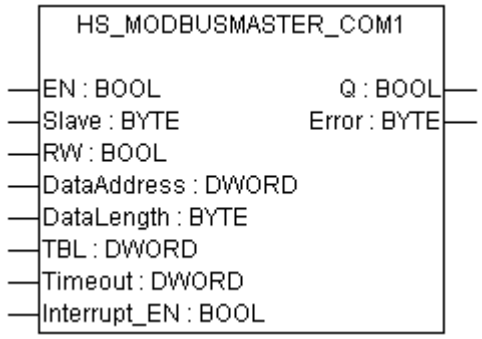
```
PRMT(EN:=EN, Address:=4 , Port_Type:=0 , BaudRate:= 19200,
    DataWidth:=8, Parity:=0, Stop:=1,
    Q=>PRMT_Q, Error=>ERR );
IF(PRMT_Q=1) THEN
    EN:=0;
END_IF
```

程序说明：

- 上电或者全下装后，EN 上升沿有效，设置完毕后，Q 等于 TRUE，设置正常时只设置一次。
- 设置 PLC 为 Modbus 3 号从站，波特率 19200，8 位数据位，无校验，1 位停止位。

5.3.1.2 HS_ModbusMaster_COM1——COM1 Modbus 主站通讯

- **功能**：提供实现 COM1 串口 Modbus 协议的主站数据读写操作的用户接口。
- **指令示例**：

梯形图语言示例


- **参数说明**：

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：上升沿使能，且高电平有效	0
Slave	BYTE	从站地址	Modbus 从站地址，1-247	1
RW	BOOL	读/写选择	0：读取数据 1：写入数据	0
DataAddress	DWORD	从站存放数据的地址	详见表 5-3-1	1
DataLength	BYTE	数据长度	1-100．对于开入/开出为所需要传输的总比特数。对于模入/模出为所要传输的总通道数	1
TBL	DWORD	主站存放数据的首字地址	诸如 200，表示存放地址为%MW200 开始的一段空间。如果是读指令，读回的数据值存放到这个数据区中；例如 3 号从站 3050 地址的数据为 1000，则%MW200 存放 1000。如果是写指令，要写出的数据值放到这个数据区中；例如向 3 号从站的 3050 地址写入 500，则%MW200 存放 500。	0
Timeout	DWORD	超时时间设置(ms)	对于 LK205/LK207，最小时间单位为 30ms。 对于 LK209/LK210，最小时间单位为 150ms。	150
Interrupt_EN	BOOL	中断使能	0：禁止中断 1：中断使能 目前不支持中断	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	完成标志	0：未完成 1：已完成	
Error	BYTE	错误信息	=0：正确 =1：超时时间(Timeout)设定过小 =2：站地址(Slave)错 =3：从站存放数据的地址(DataAddress)错 =4：数据长度(DataLength)错 =5：主站存放数据的地址(TBL)越界 =6：功能码错误 =7：获取用户空间指针失败 =8：超时 =9：CRC 校验错 =10：应答异常错 =11：未选择 Modbus 协议 =12：调用多个功能块	

表 5-3-1　DataAddress 参数详细说明

参数说明	
最高位	低五位
0：开出	Modbus 原始地址 (大于等于 5000 的部分属于 M 区)
1：开入	
3：模入	
4：模出	
从站的地址采用标准 Modbus RTU 驱动的地址填写方式，即采用六位数（十进制），第六位只能选择 0、1、3、4，分别表示 Modbus 不同的数据类型，低五位表示寄存器地址（地址范围 0-65535）。 **例如：000001，表示 Modbus 地址为 0001 的开出点** **403050，表示 Modbus 地址为 3050 的模出点**	

M 区地址与 Modbus 原始地址的对应关系

DataAddress 最高位	M 区地址	Modbus 原始地址	举例	备注
0：开出 1：开入	%MXA.B	5000+A*16+B	%MX50.7 对应于 5000+50*16+7=5807	A 为字号； B 为位号，范围 0~15
3：模入 4：模出	%MWA	5000+A	%MW100 对应于 5000+100=5100	

Modbus 功能描述

功能码	名称	作用（对主站而言）
01	读取开出状态	取得一组开关量输出的当前状态
02	读取开入状态	取得一组开关量输入的当前状态
03	读取模出状态	取得一组模拟量输出的当前状态
04	读取模入状态	取得一组模拟量输入的当前状态
05	强制单路开出	强制设定某个开关量输出的值
06	强制单路模出	强制设定某个模拟量输出的值
15	强制多路开出	强制设定从站几个开关量输出的值
16	强制多路模出	强制设定从站几个模拟量输出的值

提示：

- COM1 口通讯功能块**不可声明为 RETAIN 区变量**，不支持掉电保护，不支持冗余；
- COM1 口通讯**不支持多实例应用**，即在一个工程中每个 COM1 口通讯指令只能被调用一次；
- 调用 HS_ModbusMaster_COM1 主站指令之前应先调用并使能 HS_SetParameter_COM1 进行串口基本参数设置，激活 Modbus 协议。
- 当 HS_SetParameter_COM1 设置选择 Modbus 协议，且 HS_ModbusMaster_COM1 模块使能有效时，CPU 模块将作为主站发出数据请求，并对从站的数据响应进行处理。
- 若 HS_ModbusMaster_COM1 模块使能无效时，必须重新调用 HS_SetParameter_COM1 设置，才能使 CPU 模块作为 Modbus 从站响应其它 Modbus 主站的数据请求；否则不能响应其它 Modbus 主站的数据请求。
- Modbus 主站指令自动进行超时判断和 CRC 校验检测。
- Modbus 主站支持的功能码包括 1、2、3、4、5、6、15、16。
- 对于读从站，主站接收到从站的应答后，将接收到的数据放在 M 区偏移首地址 TBL 的数据区中；对于强制从站，主站所要强制的数据放在 M 区偏移首地址 TBL 的数据区中。存放数据的格式与内容依照 Modbus RTU 协议的规则定义。
- 数据交换过程中，需要 **EN** 端维持高电平。
- 一次上升沿只进行一次读或者写操作，若需要多次或者连续读写，则需要在上次操作完成后重复给予上升沿。
- 对于输入参数如果不做任何设置，引脚为空(无变量定义)，则启用默认设定值。

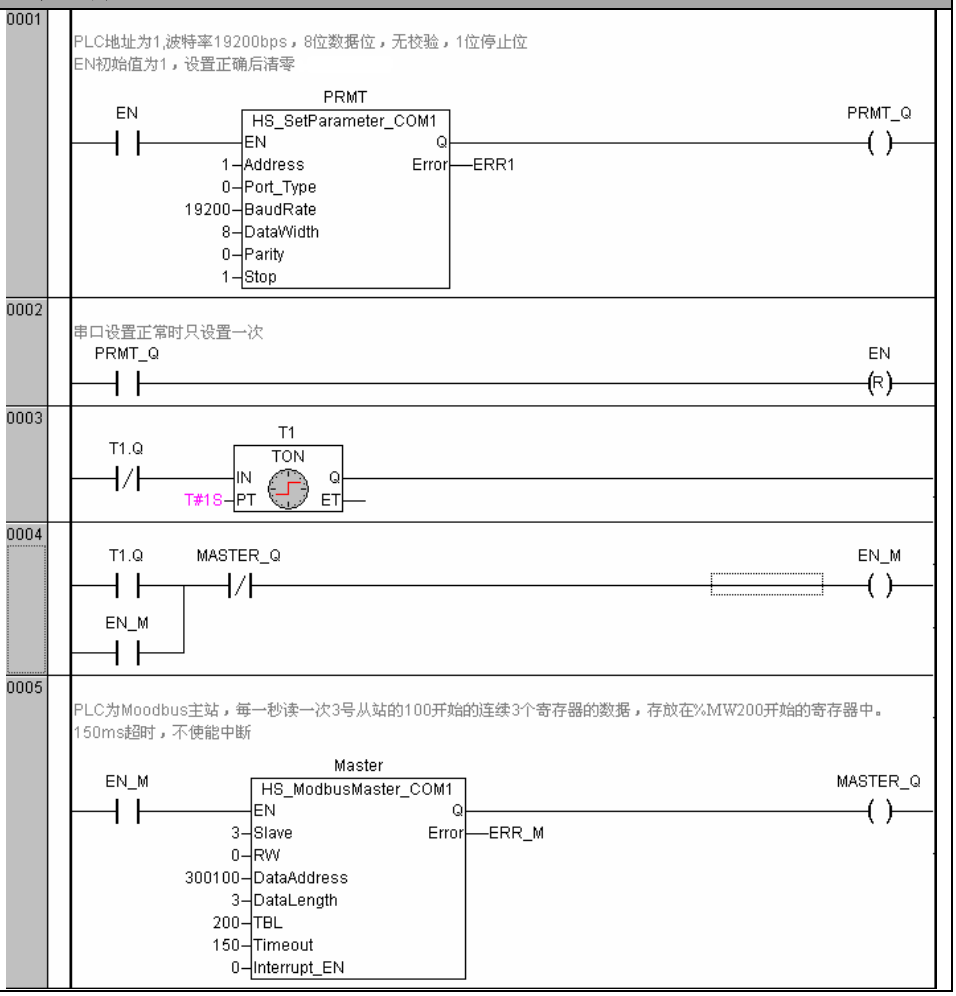
✧ 指令使用举例（一）

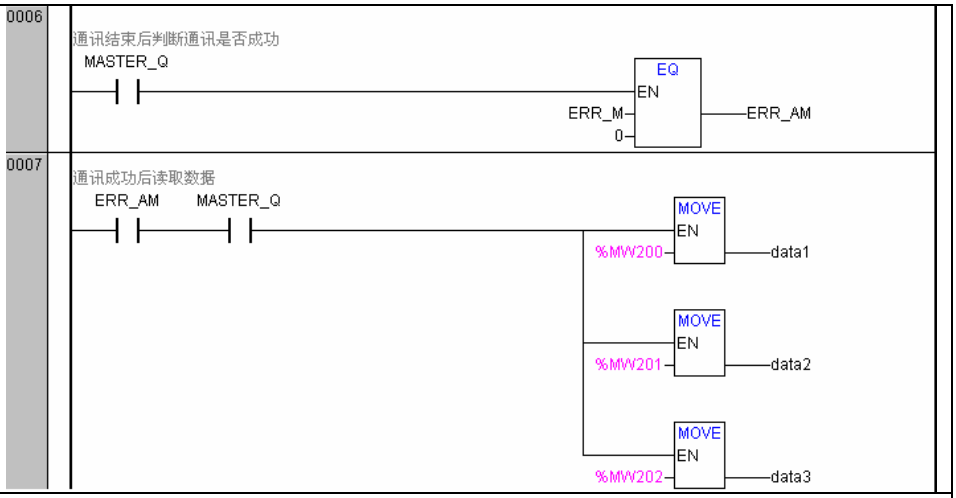
变量定义

	VAR 名称	地址	类型	初始值	注释
0001	PRMT		HS_SetParameter_		
0002	en		BOOL	1	
0003	PRMT_Q		BOOL		
0004	ERR1		BYTE		
0005	T1		TON		
0006	EN_M		BOOL		
0007	Master_Q		BOOL		通讯正确收到数据
0008	Master		HS_ModbusMaster_		
0009	ERR_M		BYTE		
0010	ERR_AM		BYTE		
0011	data1		WORD		
0012	data2		WORD		
0013	data3		WORD		

编程语言 | **程 序**

梯形图（LD）



<table>
<tr><td></td><td> 
</td></tr>
<tr><td>结构化文本
（ST）</td><td>(*设置串口，设置正常时只设置一次*)
PRMT (EN:=EN ,Address:=1,Port_Type:=0 ,BaudRate:= 19200,
DataWidth:= 8,Parity:=0 ,Stop:= 1,
Q=>PRMT_Q ,Error=>ERR1);
IF (PRMT_Q=1) THEN
EN:=0;
END_IF
T1 (IN:=NOT T1.Q , PT:=t#1s);
IF ((T1.Q =1OR EN_M=1) AND Master_Q=0) THEN
EN_M:=1;
ELSE
EN_M:=0;
END_IF
(*PLC 作为 Modbus 主站，每秒读一次从站的 100 开始的连续 3 个寄存器的数据，并存放在%MW200 开始的寄存器中，150ms 超时，不使能中断*)
Master(EN:= EN_M, Slave:=3 , RW:=0 , DataAddress:= 300100,
DataLength:=3 , TBL:= 200, Timeout:=150 , Interrupt_EN:=0 ,
Q=> Master_Q, Error=>ERR_M);
IF (Master_Q=1 AND ERR_M=0) THEN (*通讯成功后读取数据*)
data1:=%MW200;
data2:=%MW201;
data3:=%MW202;
END_IF</td></tr>
</table>

程序说明：

- 在一个工程中每个 COM1 口通讯指令只能被调用一次；
- 首先设置 PLC 的地址为 1，波特率 19200，8 位数据位，无校验，1 位停止位。
- 每秒使能一次 PLC Moodbus 主站功能，读取 3 号从站的 100 开始的连续 3 个寄存器的数据，存放在%MW200 开始的寄存器中。
- 超过 150ms 从站无应答，判定为超时，不使能中断。

✧ 指令使用举例（二）

部分变量定义

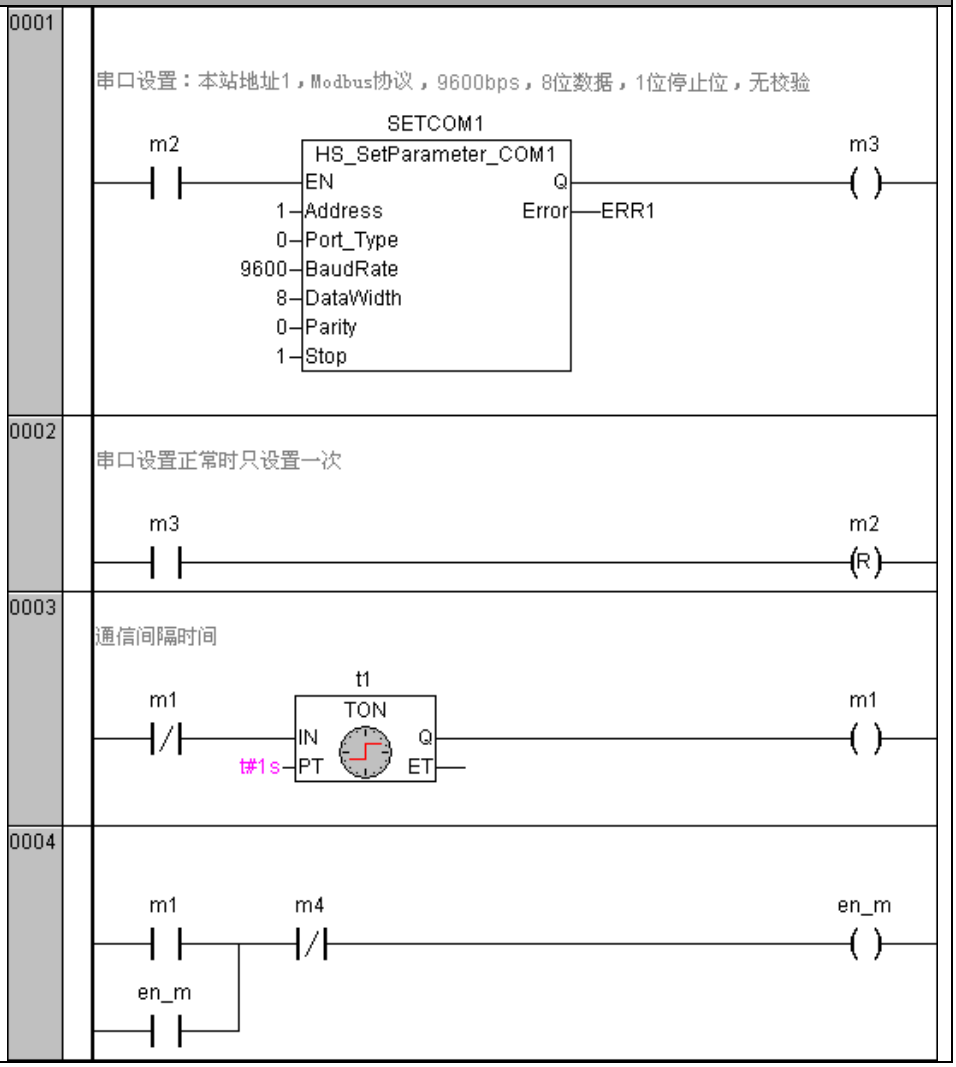
VAR

	Name	Address	Type	Initial	Comment
0001	SETCOM1		HS_SetParameter_		
0002	t1		TON		
0003	m1		BOOL		
0004	m2		BOOL	TRUE	
0005	m3		BOOL		
0006	Master		HS_ModbusMaster_		
0007	addr		BYTE	3	从站地址
0008	m4		BOOL		通讯结束
0009	m100		BOOL		通讯正确
0010	m11		BOOL		3号正确
0011	m22		BOOL		4号正确

限于篇幅，其余变量定义省略。

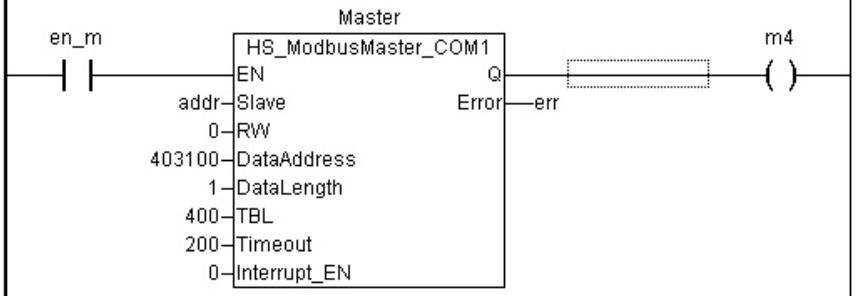
编程语言 **程 序**

梯形图（LD）



0005

向从站发命令

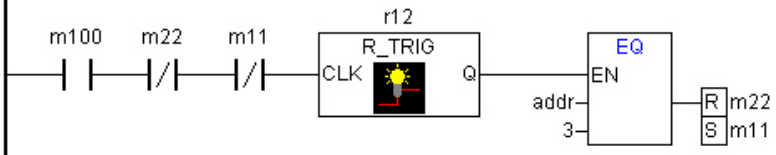


0006

通讯结束后判断是否成功

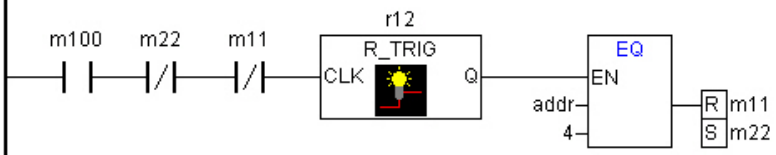
0007

与3号从站通讯是否成功



0008

与4号从站通讯是否成功

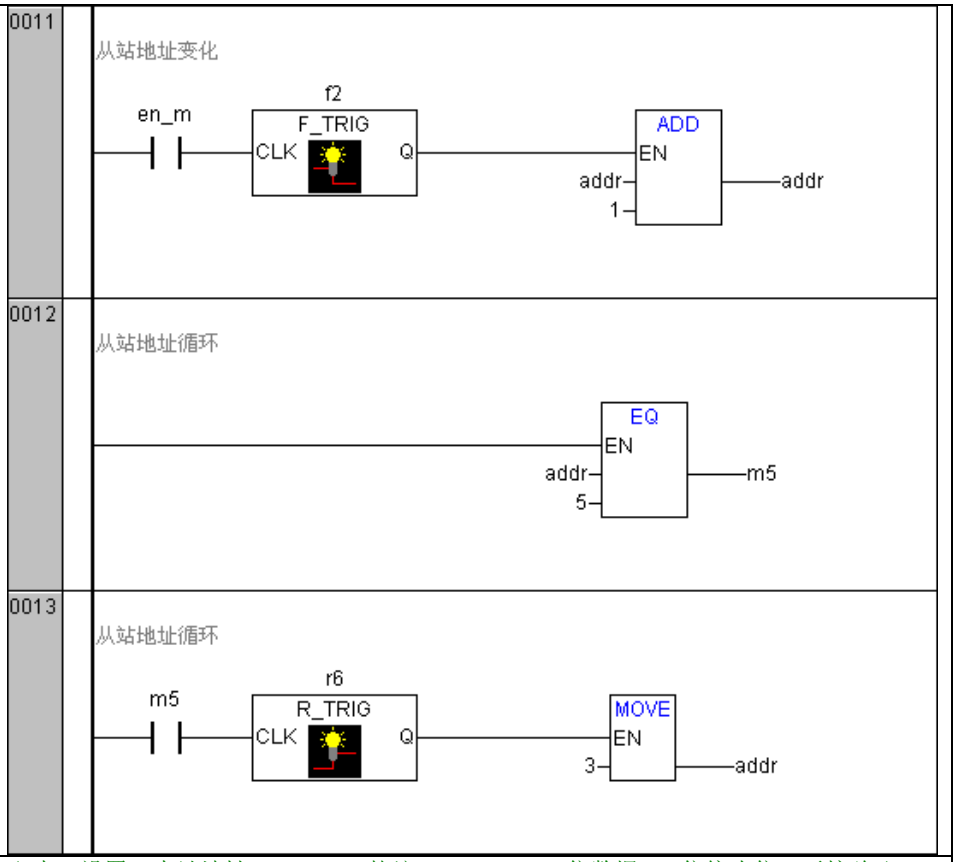


0009

数据对号入座

0010

数据对号入座



结构化文本（ST）

```
(*串口设置：本站地址 1，Modbus 协议，9600bps，8 位数据，1 位停止位，无校验*)
SETCOM1( EN:=m2, Address:=1, Port_Type:=0, BaudRate:=9600,
    DataWidth:=8, Parity:=0, Stop:=1, Q=>m3, Error=>ERR1 );
IF m3=TRUE THEN       (*串口设置正常时只设置一次*)
    m2:=FALSE;
END_IF
t1 (IN:= NOT m1, PT:=t#1s, Q=>m1)    (*通信间隔时间*)
IF ((m1 OR en_m) AND (NOT m4))=TRUE THEN
    en_m:=TRUE;
ELSE
    en_m:=FALSE;
END_IF
(*向从站发命令*)
Master(EN:=en_m, Slave:=addr, RW:=0, DataAddress:=403100,
    DataLength:=1, TBL:=400, Timeout:=200, Interrupt_EN:=0,
    Q=>m4, Error=>err );
(*通讯结束后判断是否成功*)
IF (m4=TRUE AND err=0) THEN
    m100:=TRUE;
END_IF
r12 (CLK:=(m100 AND (NOT m22) AND (NOT m11)) );
IF r12.Q =TRUE THEN
  IF addr=3 THEN      (*与 3 号从站通讯是否成功*)
    m22:=FALSE;
    m11:=TRUE;
  END_IF
  IF addr=4 THEN      (*与 4 号从站通讯是否成功*)
    m11:=FALSE;
    m22:=TRUE;
  END_IF
END_IF
```

```
r2 (CLK:=m11 );
IF r2.Q=TRUE THEN
    %MW1000:=%MW400;      (*数据对号入座*)
END_IF
r4 (CLK:=m22);
IF r4.Q=TRUE THEN
    %MW1200:=%MW400;      (*数据对号入座*)
END_IF
f2 (CLK:=en_m);
IF f2.Q=TRUE THEN
addr:=addr+1;      (*从站地址变化*)
END_IF
IF addr=5 THEN
m5:=TRUE;          (*从站地址循环*)
END_IF
r6 (CLK:=m5);
IF r6.Q=TRUE THEN
addr:=3;           (*从站地址循环*)
END_IF
```

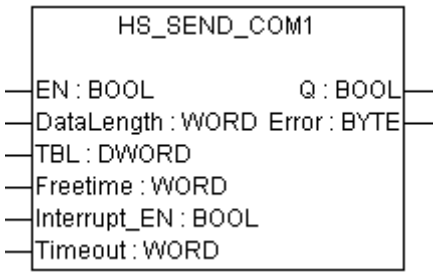
程序说明：

- 从站地址 addr 的初始值为 3，并且在 3 和 4 之间循环；
- 在一个工程中每个 COM1 口通讯指令只能被调用一次；
- 本例通过 HS_ModbusMaster_COM1 功能块的从站地址的循环变化，来实现只调用一次 HS_ModbusMaster_COM1 功能块的前提下，与多个从站进行通讯的目的。

5.3.1.3 HS_SEND_COM1——COM1 自由协议通讯数据发送

- **功能**：提供 COM1 口自由协议通讯数据发送端的参数配置和显示发送错误信息的用户接口。
- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：上升沿使能，高电平有效	0
DataLength	WORD	发送字节数	发送的字节数，1-1024 个字节。	255
TBL	DWORD	数据存放地址	指向 M 区数据存放的首字节地址。诸如 200，表示存放地址为%MB200 开始的一段空间。	0
Freetime	WORD	空闲线时间 (ms)	发送数据时保证与前一帧数据间传输总线上规定的空闲时间。 目前不支持空闲线	0

Interrupt_EN	BOOL	中断使能输入	0：发送完数据后不产生中断 1：发送完数据后产生中断 目前不支持中断	0
Timeout	WORD	超时设置 (ms)	从启动发送过程开始计时，发送完整个数据包后进行超时判断，若超过设置的时间未完成发送则将相应的状态位置 1。 其值为 0 时不判断超时。	0
输出	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	发送完成标志	0：发送未完成 1：发送已完成	
Error	BYTE	错误信息	=0：正确 =1：数据长度（DataLength）错 =2：数据存放地址（TBL）越界 =3：获取用户空间指针失败 =4：超时 =5：未选择自由协议 =6：调用多个功能块	

提示：

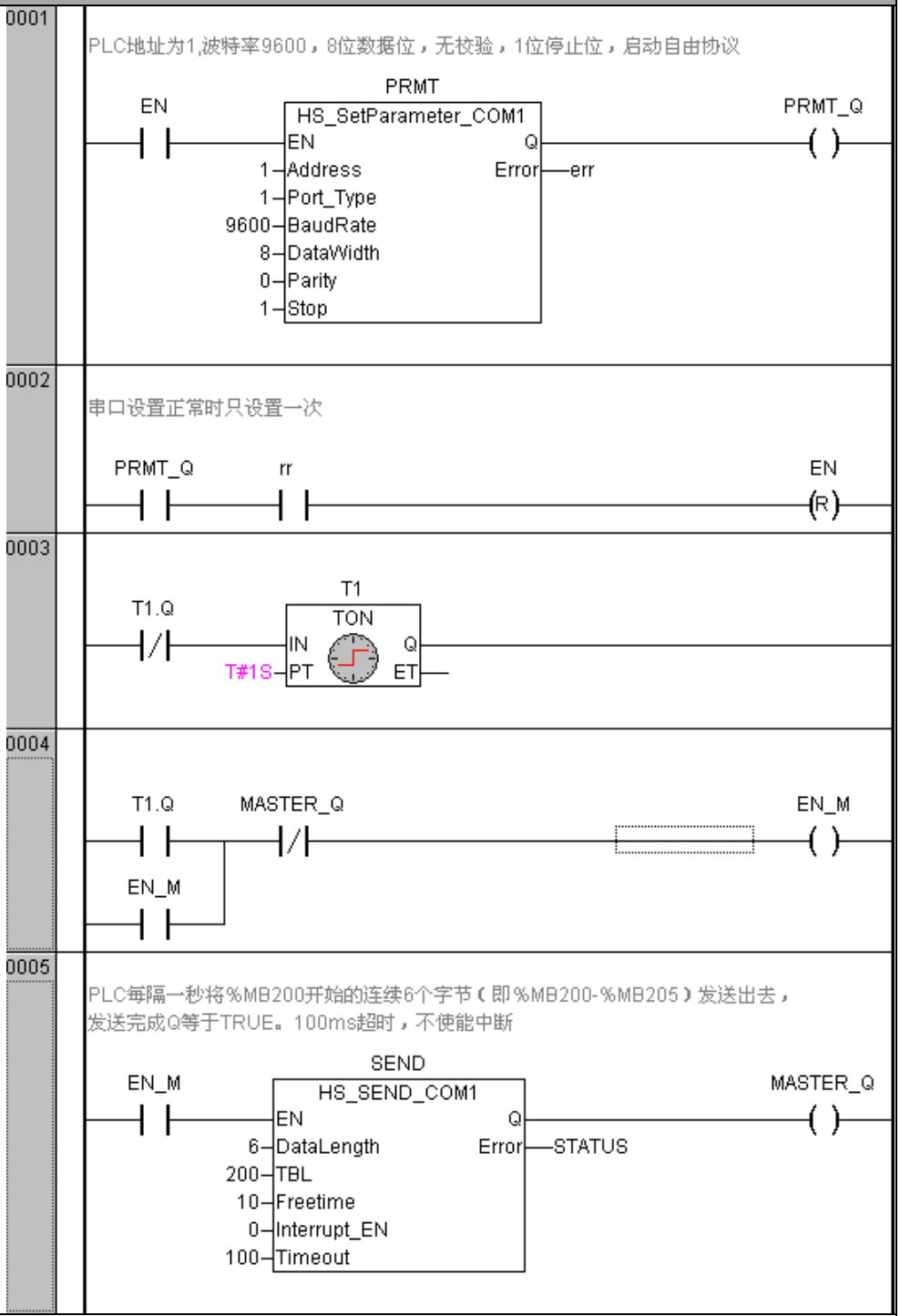
- COM1 口通讯功能块**不可声明为 RETAIN 区变量**，不支持掉电保护，不支持冗余；
- COM1 口通讯**不支持多实例应用**，即在一个工程中每个 COM1 口通讯指令只能被调用一次；
- 调用 HS_SEND_COM1 主站指令之前应先调用并使能 HS_SetParameter_COM1 进行串口基本参数设置，激活自由协议。
- 数据发送过程中，需要 **EN** 维持高电平。
- 对于输入参数如果不做任何设置，引脚为空(无变量定义)，则启用默认设定值。

✧ **指令使用举例**

变量定义

VAR

	名称	地址	类型	初始值	注释
0001	PRMT		HS_SetParameter_		
0002	EN		BOOL	1	使能
0003	ERR		BYTE		
0004	PRMT_Q		BOOL		
0005	Master		HS_ModbusMaster_		
0006	EN_M		BOOL		
0007	MASTER_Q		BOOL		
0008	ERR_M		BYTE		
0009	T1		TON		
0010	SEND		HS_SEND_COM1		
0011	STATUS		BYTE		
0012	rr		BOOL		

编程语言	程　序
梯形图（LD）	

结构化文本（ST）	(*设置串口，设置正常时只设置一次*)
	PRMT(EN:=EN, Address:=1, Port_Type:=1, BaudRate:= 9600,
	DataWidth:= 8, Parity:=0 , Stop:= 1,
	Q=>PRMT_Q , Error=> err);
	IF(PRMT_Q=1 AND rr=1) THEN
	EN:=0;
	END_IF
	T1 (IN:=NOT T1.Q , PT:=t#1s);
	IF ((T1.Q=1 OR EN_M=1) AND Master_Q=0) THEN
	EN_M:=1;

```
ELSE
    EN_M:=0;
END_IF
(*PLC 每隔一秒将%MB200 开始的连续 6 个字节发送出去，发送完成 Q 等于 TRUE，100ms 超时，不使能中断*)
SEND( EN:= EN_M, DataLength:= 6, TBL:= 200,
      Freetime:=10, Interrupt_EN:= 0, Timeout:= 100,
      Q=>Master_Q , Error=>STATUS );
```

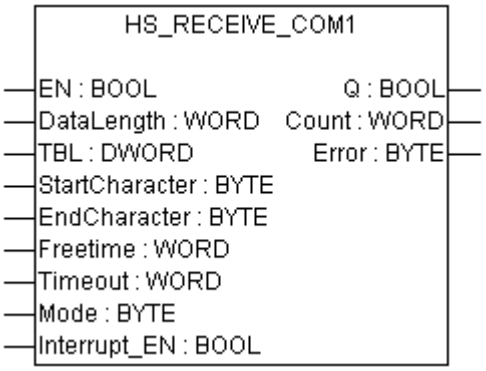
程序说明：

- 首先设置 PLC 的地址为 1，波特率 9600，8 位数据位，无校验，1 位停止位，启动自由协议。
- PLC 每隔一秒将%MB200 开始的连续 6 个字节（即%MB200-%MB205）发送出去，发送完成 Q 等于 TRUE。
- 不使能中断。
- 在一个工程中每个 COM1 口通讯指令只能被调用一次。

5.3.1.4 HS_ RECEIVE _COM1——COM1 自由协议通讯数据接收

- **功能**：提供 COM1 口自由协议通讯数据接收端的参数配置和显示接收错误信息的用户接口。
- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：上升沿使能，高电平有效	0
DataLength	WORD	数据长度	接收的字节数，0-1024 个字节。 该参数会与其它参数配合使用，详见表 5-3-2	255
TBL	DWORD	数据存放地址	指向 M 区数据存放的首字节地址， 诸如 200，表示存放地址为%MB200 开始的一段空间。	0
StartCharacter	BYTE	开始字符	当启用开始字符时，只有接收到开始字符才表征一帧数据的开始，否则被丢弃	0
EndCharacter	BYTE	结束字符	当启用结束字符时，只有接收到结束字符才表征一帧数据的结束，否则继续接收直到缓冲区满（1024 字节）	0
Freetime	WORD	空闲线时间(ms)	在规定的时间溢出后接收的第一个字符，才认为是一个完整的数据帧的开始。 最小时间单位为系统调度时间。 目前不支持空闲线	0
Timeout	WORD	超时设置	从启动接收过程开始计算，在规定的时间内没	0

		(ms)	有再接收到完整的数据包，中止接收过程。最小时间单位为系统调度时间。 其值为 0 时不判断超时。	
Mode	BYTE	控制模式参数设置	详见表 5-3-3	0
Interrupt_EN	BOOL	中断使能输入	0：接收完数据后不产生中断 1：接收完数据后产生中断 目前不支持中断	0
输出	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	接收完成标志	0：接收未完成 1：接收已完成	
Count	WORD	实际接收的字节数	指示本次接收实际接收的字节数	
Error	BYTE	错误信息	=0：正确 =1：数据长度（DataLength）错 =2：数据存放地址（TBL）越界 =3：缺少开始条件 =4：缺少结束条件 =5：超时时间（Timeout）设定过小 =6：获取用户空间指针失败 =7：超时 =8：未选择自由协议 =9：调用多个功能块	

表 5-3-2　数据长度与开始字符、结束字符的配合使用

DataLength	StartCharacter	EndCharacter	字符的接收过程
0	使能	使能	接收到开始字符才表征一帧数据的开始，否则数据被丢弃；接收到结束字符才表征一帧数据的结束，否则继续接收直到缓冲区满（1024 字节）。
0	禁止	使能	接收到结束字符才表征一帧数据的结束，否则继续接收直到缓冲区满（1024 字节）。
0	使能	禁止	Error 报错，缺少结束字符。
0	禁止	禁止	Error 报错，缺少开始字符和结束字符。
不为 0	使能	禁止	收到开始字符后继续接收，并依次排放，直到接满 DataLength 个字节（包括开始字符）后结束本次接收过程。
不为 0	使能	使能	收到开始字符后继续接收，直到接满 DataLength 个字节（包括开始字符）或收到结束字符，结束本次接收过程。
不为 0	禁止	使能	指令有效即开始接收，直到接满 DataLength 个字节（包括开始字符）或收到结束字符，结束本次接收过程。
不为 0	禁止	禁止	指令有效即开始接收，直到接满 DataLength 个字节，结束本次接收过程。

表 5-3-3　Mode 参数详细说明

Mode（BYTE）						
7　6	5	4	3	2	1	0
未定义	接收完数据中断	单字节接收中断	空闲时间控制	结束字符控制	开始字符控制	超时控制
超时控制						
0：禁止超时控制 1：启动超时控制						
开始字符控制						
0：禁止开始字符控制 1：启动开始字符控制						

结束字符控制
0：禁止结束字符控制 1：启动结束字符控制
空闲时间控制
0：禁止空闲时间控制 1：启动空闲时间控制
单字节接收中断
0：禁止单字节接收中断功能 1：启动单字节接收中断功能 目前不支持中断
接收完数据中断
0：禁止接收完数据中断功能 1：启动接收完数据中断功能 目前不支持中断

提示：

- COM1 口通讯功能块**不可声明为 RETAIN 区变量**，不支持掉电保护，不支持冗余；
- COM1 口通讯**不支持多实例应用**，即在一个工程中每个 COM1 口通讯指令只能被调用一次；
- 调用 HS_RECEIVE_COM1 主站指令之前应先调用并使能 HS_SetParameter_COM1 进行串口基本参数设置，激活自由协议。
- 数据接收过程中，需要 **EN** 维持高电平。
- 一次上升沿只进行一次读或者写操作，若需要多次或者连续读写，则需要在上次操作完成后重复给予上升沿。
- 对于输入参数如果不做任何设置，引脚为空(无变量定义)，则启用默认设定值。

✧ **指令使用举例**

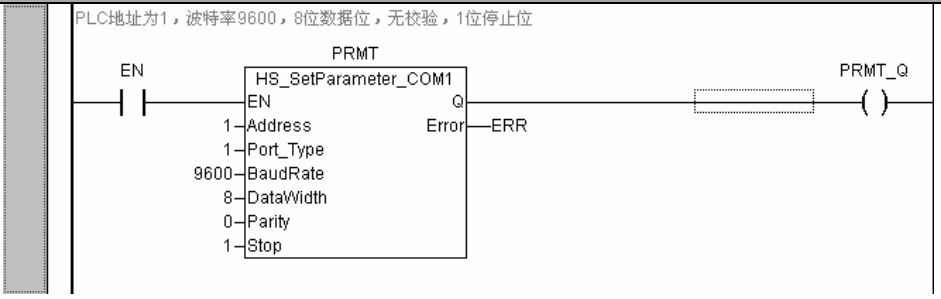
变量定义

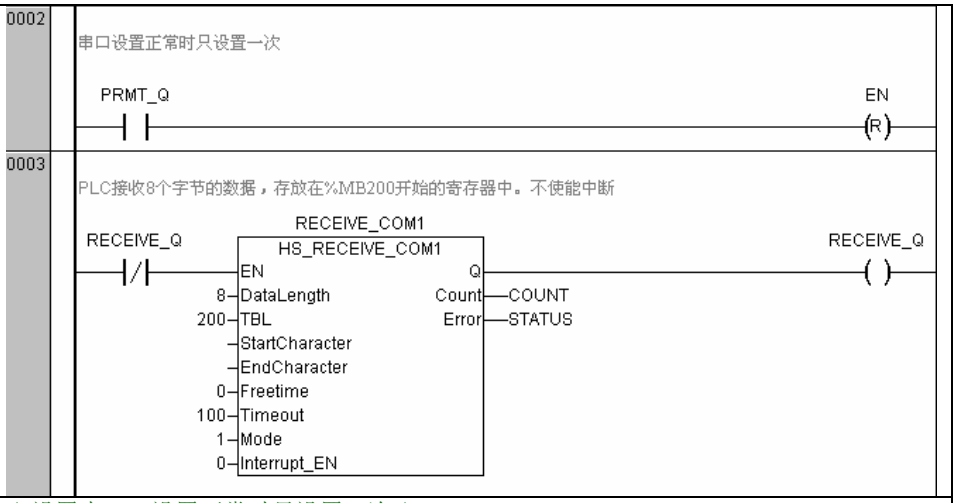
VAR

	名称	地址	类型	初始值	注释
0001	PRMT		HS_SetParameter_		
0002	EN		BOOL	1	使能
0003	ERR		BYTE		
0004	PRMT_Q		BOOL		
0005	RECEIVE_Q		BOOL		
0006	RECEIVE_COM1		HS_RECEIVE_COM		
0007	STATUS		BYTE		
0008	COUNT		WORD		

编程语言 | **程　序**

梯形图（LD）



<table>
<tr><td></td><td> 
</td></tr>
<tr><td>结构化文本
（ST）</td><td>(*设置串口，设置正常时只设置一次*)
PRMT(EN:=EN, Address:=1, Port_Type:=1, BaudRate:= 9600,
DataWidth:= 8, Parity:=0 , Stop:= 1,
Q=>PRMT_Q , Error=> err);
IF(PRMT_Q=1) THEN
EN:=0;
END_IF
(*PLC 接收 8 个字节的数据，存放在%MB200 开始的寄存器中，不使能中断*)
RECEIVE_COM1(EN:=NOT RECEIVE_Q , DataLength:= 8, TBL:= 200,
Freetime:=0 ,Timeout:= 100,Mode:= 1, Interrupt_EN:=0 ,
Q=>RECEIVE_Q ,Count=>COUNT ,Error=>STATUS);</td></tr>
</table>

程序说明：

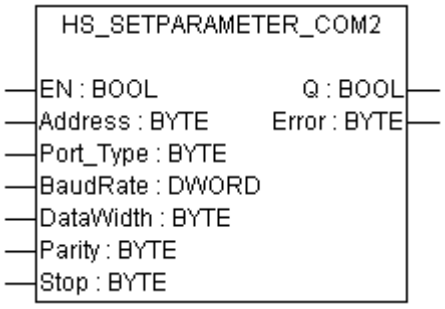
- 设置 PLC 的地址为 1，波特率 9600，8 位数据位，无校验，1 位停止位，启动自由协议。
- PLC 每隔一个扫描周期接收 8 个字节数据保存到%MB200-%MB207 中。
- 不使能中断。
- 在一个工程中每个 COM1 口通讯指令只能被调用一次。

5.3.2 COM2 口通讯指令

- **注意：**在 PowerPro V4.3.1B 中，对 COM2 口通讯指令进行了完善和更新，如果用户的主控型号低于 LK202-B01, LK205-B01, LK207-A04, LK209-B01, LK210-B06 这几类，请到 Backup Library 中调用适合您主控的 HS_Communication.lib 库版本文件，具体操作方法请参见附录 3。

5.3.2.1 HS_SetParameter_COM2——COM2 通讯参数设置

- **功能：**提供设置 COM2 串口通讯参数、选择通讯协议的用户接口。如果使能 Modbus 协议，默认支持 Modbus 从站功能，即此时可以响应 Modbus 主的数据请求。
- **指令示例：**

梯形图语言示例


- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：上升沿有效	0
Address	BYTE	本站地址	范围 1-247	1
Port_Type	BYTE	串口类型	.0 RS232 端口配置 =1：使能 RS232 端口 =0：使能 RS485 端口 .1 端口协议选择 =1：自由协议 =0： Modbus 协议	0
BaudRate	DWORD	通讯的波特率（bps）	1200，2400，4800，9600，19200，38400，57600，115200	115200
DataWidth	BYTE	数据位	=5：5 位数据位（Modbus 协议不支持） =6：6 位数据位（Modbus 协议不支持） =7：7 位数据位（Modbus 协议不支持） =8：8 位数据位（Modbus 协议只能使用该位数） 其它，非法	8
Parity	BYTE	校验方式	=0：无校验 =1：奇校验 =2：偶校验	0
Stop	BYTE	停止位位数	=1：1 位停止位 =2：2 位停止位 其它，非法	1
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	设置完成	0：未完成 1：已完成	
Error	BYTE	错误信息	=0：正确	

			=1：初始化串口错 =2：站地址（Address）错 =3：串口类型（Port_Type）错 =4：波特率（BaudRate）错 =5：数据位（DataWidth）错 =6：校验方式（Parity）错 =7：停止位（Stop）错 =8：串口通信参数设置失败 =9：调用多个功能块	

提示：

- COM2 口通讯功能块**不可声明为 RETAIN 区变量**，不支持掉电保护，不支持冗余；
- COM2 口通讯**不支持多实例应用**，即在一个工程中每个 COM2 口通讯指令只能被调用一次；
- 对于输入参数如果不做任何设置，引脚为空(无变量定义)，则启用默认设定值。
- 由于 **EN** 是上升沿有效，正常时只设置一次，所以调试中如果本指令的输入参数发生改变需要进行全下载，或者在线下载（增量下载）后将 **EN** 置位，形成上升沿，以使新的参数生效。
- **EN** 是上升沿有效，正常时只设置一次，不可持续不断设置上升沿。

✧ **指令使用举例**

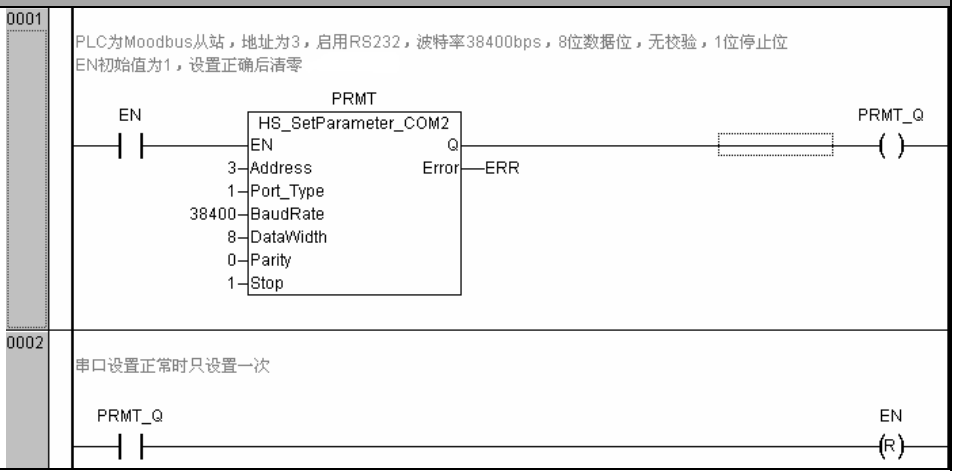
变量定义

VAR

	名称	地址	类型	初始值	注释
0001	PRMT		HS_SetParameter_		
0002	EN		BOOL	1	使能
0003	PRMT_Q		BOOL		
0004	ERR		BYTE		

编程语言 | **程　序**

梯形图（LD）



结构化文本（ST）

```
PRMT(EN:=EN, Address:=3 , Port_Type:=1 , BaudRate:= 38400,
    DataWidth:=8, Parity:=0, Stop:=1,
    Q=>PRMT_Q, Error=>ERR );
IF(PRMT_Q=1) THEN
    EN:=0;
END_IF
```

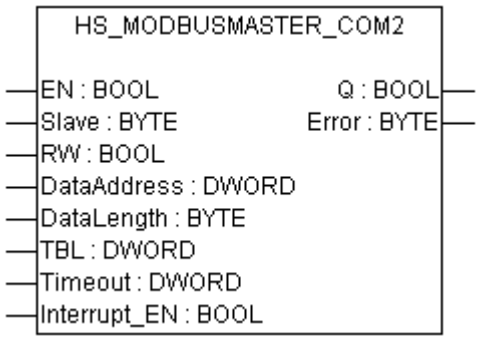
程序说明：

- 上电或者全下装后，EN 上升沿有效，设置完毕后，Q 等于 TRUE，设置正常时只设置一次。

- 设置 PLC 为 Modbus 3 号从站，启用 RS232，波特率 38400，8 位数据位，无校验，1 位停止位。

5.3.2.2 HS_ModbusMaster_COM2——COM2 Modbus 主站通讯

- **功能**：提供实现 COM2 串口 Modbus 协议主站数据读写操作的用户接口。
- **指令示例**：

梯形图语言示例


- **参数说明**：

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：上升沿使能，高电平有效	0
Slave	BYTE	从站地址	Modbus 从站地址，1-247	1
RW	BOOL	读/写选择	0：读取数据　　1：写入数据	0
DataAddress	DWORD	从站存放数据地址	地址范围 0-65535，详见表 5-3-4	1
DataLength	BYTE	数据长度	1-100.对于开入/开出为所需要传输的总比特数;模入/模出为所要传输的总通道数。	1
TBL	DWORD	主站存放数据的字地址	诸如 200，表示存放地址为%MW200 开始的一段空间。如果是读指令，读回的数据值存放到这个数据区中；例如 3 号从站 3050 地址的数据为 1000，则%MW200 存放 1000。如果是写指令，要写出的数据值放到这个数据区中；例如向 3 号从站的 3050 地址写入 500，则%MW200 存放 500。	0
Timeout	DWORD	超时时间(ms)	对于 LK205/LK207，最小时间单位为 30ms。 对于 LK209/LK210，最小时间单位为 150ms。	100
Interrupt_EN	BOOL	中断使能	0：禁止中断 1：中断使能 目前不支持中断	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	完成标志	0：未完成 1：已完成	
Error	BYTE	错误信息	=0：正确 =1：超时时间（Timeout）设定过小 =2：站地址（Slave）错 =3：从站存放数据的地址（DataAddress）错 =4：数据长度（DataLength）错 =5：主站存放数据的地址（TBL）越界 =6：功能码错误 =7：获取用户空间指针失败	

			=8：超时 =9：CRC 校验错 =10：应答异常错 =11：未选择 Modbus 协议 =12：调用多个功能块	

表 5-3-4 DataAddress 参数详细说明

参数说明	
最高位	低五位
0：开出	Modbus 原始地址 (大于等于 5000 的部分属于 M 区)
1：开入	
3：模入	
4：模出	
从站的地址采用标准 Modbus RTU 驱动的地址填写方式，即采用六位数（十进制），第六位只能选择 0、1、3、4，分别表示 Modbus 不同的数据类型，低五位表示寄存器地址（地址范围 0-65535）。 **例如：000001，表示 Modbus 地址为 0001 的开出点** **403050，表示 Modbus 地址为 3050 的模出点**	

M 区地址与 Modbus 原始地址的对应关系

Modbus 功能码	M 区地址	Modbus 原始地址	举例	备注
0：开出	%MXA.B	5000+A*16+B	%MX50.7 对应于 5000+50*16+7=5807	A 为字号；B 为位号，范围 0~15
1：开入				
3：模入	%MWA	5000+A	%MW100 对应于 5000+100=5100	
4：模出				

提示：

- COM2 口通讯功能块**不可声明为 RETAIN 区变量**，不支持掉电保护，不支持冗余；
- COM2 口通讯**不支持多实例应用**，即在一个工程中每个 COM2 口通讯指令只能被调用一次；
- 调用 HS_ModbusMaster_COM2 主站指令之前应先调用并使能 HS_SetParameter_COM2 进行串口基本参数设置，激活 Modbus 协议。
- 当 HS_SetParameter_COM2 设置选择 Modbus 协议，且 HS_ModbusMaster_COM2 模块使能有效时，CPU 模块将作为主站发出数据请求，并对从站的数据响应进行处理。
- 若 HS_ModbusMaster_COM2 模块使能无效时，必须重新调用 HS_SetParameter_COM2 设置，才能使 CPU 模块作为 Modbus 从站响应其它 Modbus 主站的数据请求；否则不能响应其它 Modbus 主站的数据请求。
- Modbus 主站指令自动进行超时判断和 CRC 校验检测。
- Modbus 主站支持的功能码包括 1、2、3、4、5、6、15、16。
- 对于读从站，主站接收到从站的应答后，将接收到的数据放在 M 区偏移首地址 TBL 的数据区中；对于强制从站，主站所要强制的数据放在 M 区偏移首地址 TBL 的数据区中。存放数据的格式与内容依照 Modbus RTU 协议的规则定义。
- 数据交换过程中，需要 **EN** 端维持高电平。
- 一次上升沿只进行一次读或者写操作，若需要多次或者连续读写，则需要在上次操作完成后重复给予上升沿。
- 对于输入参数如果不做任何设置，引脚为空(无变量定义)，则启用默认设定值。

✧ 指令使用举例

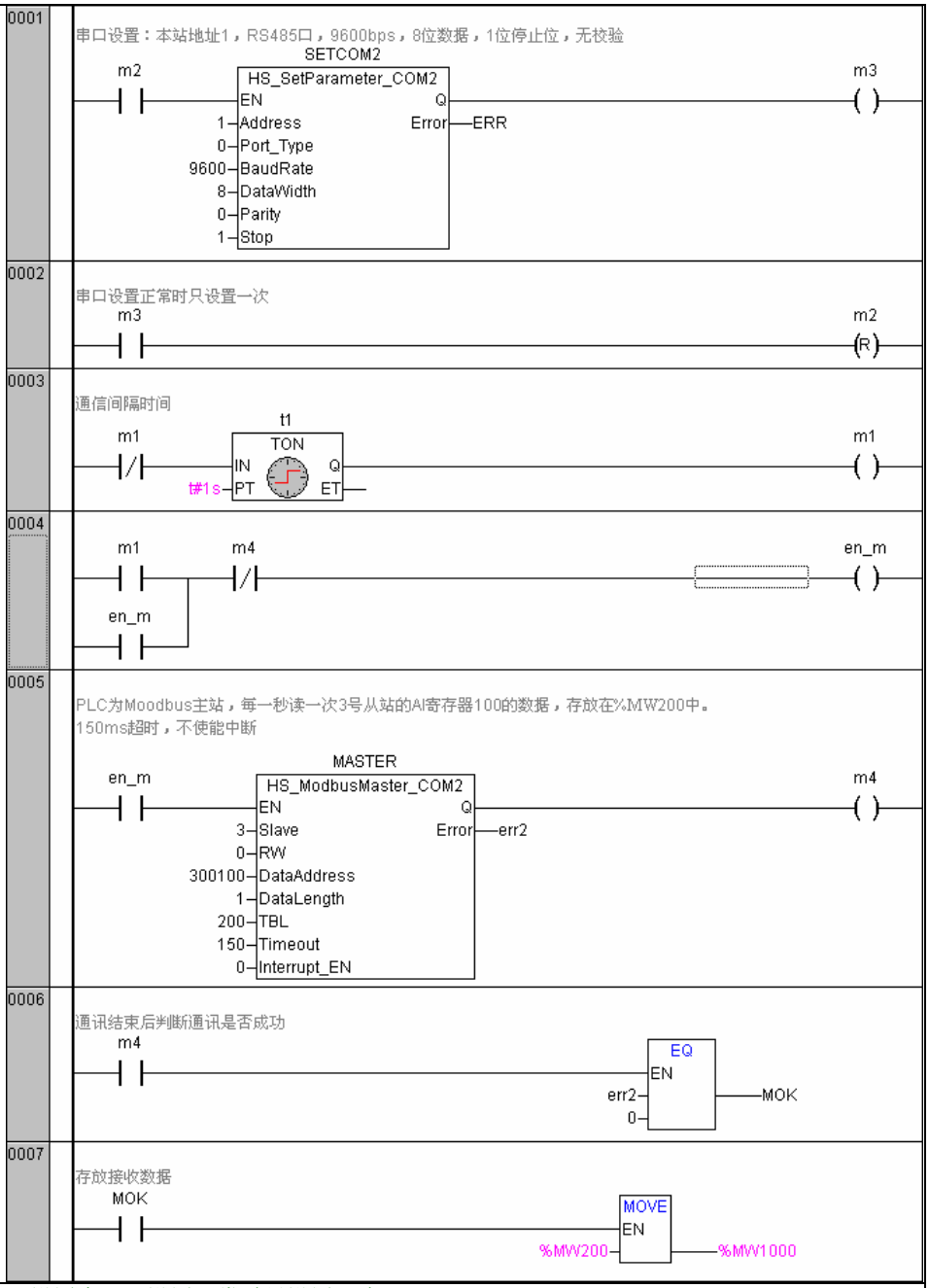
变量定义

VAR

	名称	地址	类型	初始值	注释
0001	t1		TON		
0002	m1		BOOL		
0003	m2		BOOL	TRUE	
0004	m3		BOOL		
0005	err		BYTE		
0006	m4		BOOL		通讯正确收到数据
0007	MASTER		HS_ModbusMaster_		
0008	en_m		BOOL		
0009	err2		BYTE		
0010	SETCOM2		HS_SetParameter_		
0011	MOK		BOOL		

编程语言	程　序

梯形图（LD）



结构化文本（ST）

```
(*设置串口，设置正常时只设置一次*)
PRMT ( EN:=m2 ,Address:=1,Port_Type:=0 ,BaudRate:= 9600,
    DataWidth:= 8,Parity:=0 ,Stop:= 1,
    Q=>m3 ,Error=>ERR );
IF (m3=1) THEN
    M2:=0;
END_IF
T1 (IN:= NOT m1, PT:=t#1s,
     Q=>m1);
IF ((m1=1 OR EN_M=1) AND m4=0) THEN
    EN_M:=1;
ELSE
    EN_M:=0;
END_IF
(*PLC 作为 Modbus 主站，每秒读一次 3 号从站 AI 寄存器 100 的数据，并存放在%MW200 中，
150ms 超时，不使能中断*)
```

```
Master(EN:= EN_M, Slave:=3, RW:=0 , DataAddress:= 300100,
    DataLength:=1 , TBL:= 200, Timeout:=150 , Interrupt_EN:=0 ,
    Q=> m4, Error=>err2 );
IF (m4=1 AND err2=0) THEN    (*通讯成功后读取数据*)
    %MW1000:=%MW200;
END_IF
```

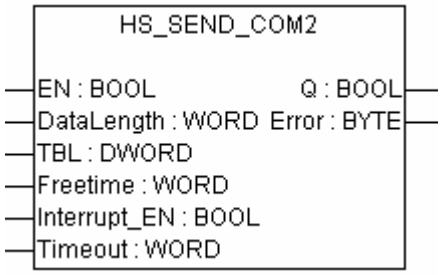
程序说明：

- 首先设置 PLC 的地址为 1，启用 RS485，波特率 9600，8 位数据位，无校验，1 位停止位。
- 每秒使能一次 PLC Moodbus 主站功能，读取 3 号从站的 100 开始的连续 3 个寄存器的数据，存放在%MW200 开始的寄存器中。
- 超过 150ms 从站无应答，判定为超时，不使能中断。
- 在一个工程中每个 COM2 口通讯指令只能被调用一次，如果希望与多个从站进行通讯，可以通过在线修改 HS_ModbusMaster_COM2 功能块的从站地址来实现，或者请参考 5.3.1.2 节的**程序使用举例（二）**中的方法。

5.3.2.3 HS_SEND_COM2——COM2 自由协议通讯数据发送

- **功能**：提供 COM2 口自由协议通讯数据发送端的参数配置和显示发送错误信息的用户接口。
- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：上升沿使能，高电平有效	0
DataLength	WORD	发送字节数	发送的字节数，1-1024 个字节。	255
TBL	DWORD	数据存放地址	指向 M 区数据存放的首字节地址，如 200，表示存放地址为%MB200 开始的一段空间。	0
Freetime	WORD	空闲线时间(ms)	发送数据时保证与前一帧数据间传输总线上规定的空闲时间。 目前不支持空闲线	0
Interrupt_EN	BOOL	中断使能输入	0：发送完数据后不产生中断 1：发送完数据后产生中断 目前不支持中断	0
Timeout	WORD	超时设置(ms)	从启动发送过程开始计时，发送完整个数据包后进行超时判断，若超过设置的时间未完成发送则将相应的状态位置 1。 其值为 0 时不判断超时。	0
输出	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	发送完成标志	0：发送未完成 1：发送已完成	
Error	BYTE	错误信息	=0：正确	

			=1：数据长度（DataLength）错 =2：数据存放地址（TBL）越界 =3：获取用户空间指针失败 =4：超时 =5：未选择自由协议 =6：调用多个功能块	

提示：

- COM2 口通讯功能块**不可声明为 RETAIN 区变量**，不支持掉电保护，不支持冗余；
- COM2 口通讯**不支持多实例应用**，即在一个工程中每个 COM2 口通讯指令只能被调用一次；
- 调用 HS_SEND_COM2 主站指令之前应先调用并使能 HS_SetParameter_COM2 进行串口基本参数设置，激活自由协议；
- 数据发送过程中，需要 **EN** 维持高电平；
- 一次上升沿只进行一次读或者写操作，若需要多次或者连续读写，则需要在上次操作完成后重复给予上升沿。
- 对于输入参数如果不做任何设置，引脚为空(无变量定义)，则启用默认设定值。

✧ **指令使用举例**

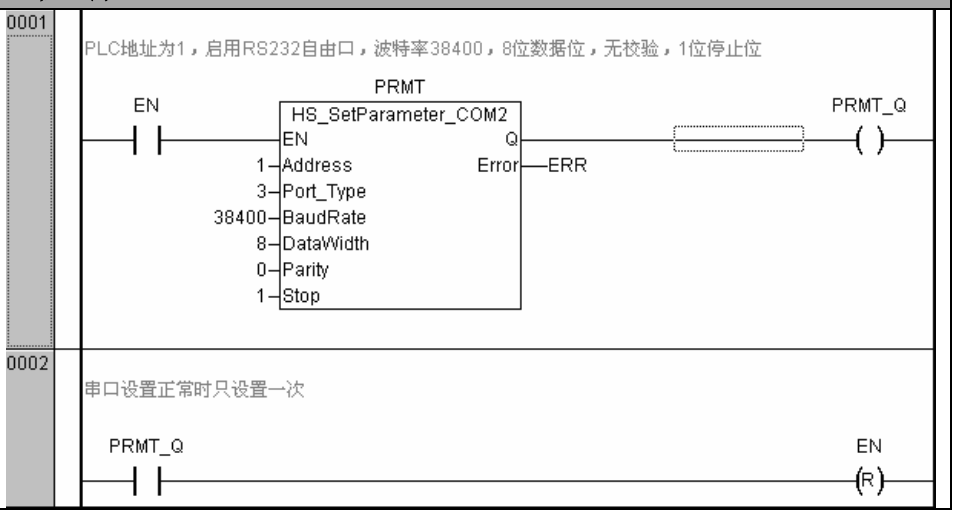
变量定义

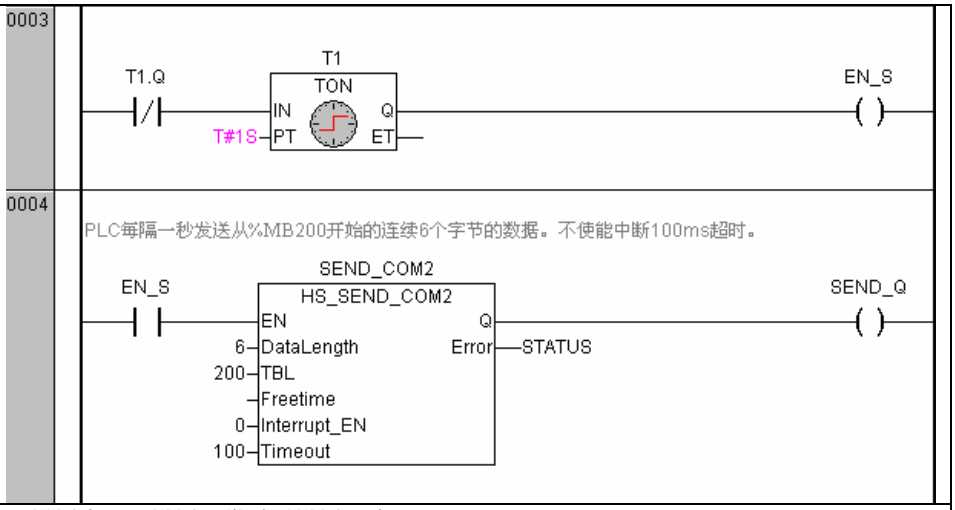
VAR

	名称	地址	类型	初始值	注释
0001	PRMT		HS_SetParameter_		
0002	EN		BOOL	1	使能
0003	ERR		BYTE		
0004	PRMT_Q		BOOL		
0005	T1		TON		
0006	EN_S		BOOL		
0007	SEND_COM2		HS_SEND_COM2		
0008	SEND_Q		BOOL		
0009	STATUS		BYTE		

编程语言 | **程　序**

梯形图（LD）



<table>
<tr><td></td><td> 
</td></tr>
<tr><td>结构化文本（ST）</td><td>(*设置串口，设置正常时只设置一次*)
PRMT(EN:=EN, Address:=1, Port_Type:=3, BaudRate:= 38400,
DataWidth:= 8, Parity:=0 , Stop:= 1,
Q=>PRMT_Q , Error=>err);
IF PRMT_Q=1 THEN
EN:=0;
END_IF
T1 (IN:=NOT T1.Q , PT:=t#1s,
Q=>EN_S);
(*PLC 每隔一秒将%MB200 开始的连续 6 个字节发送出去，发送完成 Q 等于 TRUE，100ms 超时，不使能中断*)
SEND_COM2(EN:= EN_S, DataLength:= 6, TBL:= 200,
Interrupt_EN:= 0, Timeout:= 100,
Q=>SEND_Q , Error=>STATUS);</td></tr>
</table>

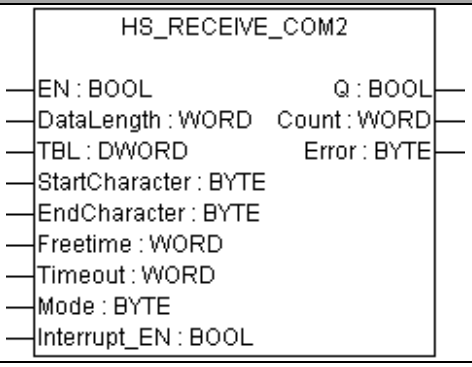
程序说明：

- 首先设置 PLC 的地址为 1，启用 RS232，波特率 38400，8 位数据位，无校验，1 位停止位，不使能中断。
- PLC 每隔一秒将%MB200 开始的连续 6 个字节（即%MB200-%MB205）发送出去，发送完成 Q 等于 TRUE。
- 在一个工程中每个 COM1 口通讯指令只能被调用一次。

5.3.2.4 HS_ RECEIVE _COM2——COM2 自由协议通讯数据接收

- **功能**：提供 COM2 口自由协议通讯数据接收端的参数配置和显示接收错误信息的用户接口。
- **指令示例：**

梯形图语言示例



➢ **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：上升沿使能，高电平有效	0
DataLength	WORD	数据长度	接收的字节数，0-1024 个字节。 该参数会与其它参数配合使用，详见表 5-3-5	255
TBL	DWORD	M 区首地址	指向 M 区数据存放的首地址，诸如 200，表示存放地址为%MB200 开始的一段空间	0
StartCharacter	BYTE	开始字符	当启用开始字符时，只有接收到开始字符才表征一帧数据的开始，否则被丢弃	0
EndCharacter	BYTE	结束字符	当启用结束字符时，只有接收到结束字符才表征一帧数据的结束，否则继续接收直到缓冲区满（1024 字节）	0
Freetime	WORD	空闲线时间(ms)	在规定的时间溢出后接收的第一个字符，才认为是一个完整的数据帧的开始。最小时间单位为系统调度时间。 目前不支持空闲线时间	0
Timeout	WORD	帧超时时间(ms)	从启动接收过程开始计算，在规定的时间内没有再接收到完整的数据包，中止接收过程。最小时间单位为系统调度时间。 其值为 0 时不判断超时。	0
Mode	BYTE	控制模式参数设置	详见表 5-3-6	0
Interrupt_EN	BOOL	中断使能输入	0：接收完数据后不产生中断 1：接收完数据后产生中断 目前不支持中断	0
输出	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	接收完成标志	0：接收未完成 1：接收已完成	
Count	WORD	实际接收的字节数	指示本次接收实际接收的字节数	
Error	BYTE	错误信息	=0：正确 =1：数据长度（DataLength）错 =2：数据存放地址（TBL）越界 =3：缺少开始条件 =4：缺少结束条件 =5：超时时间（Timeout）设定过小 =6：获取用户空间指针失败 =7：超时 =8：未选择自由协议 =9：调用多个功能块	

表 5-3-5　数据长度与开始字符、结束字符的配合使用

DataLength	StartCharacter	EndCharacter	字符的接收过程
0	使能	使能	接收到开始字符才表征一帧数据的开始，否则数据被丢弃；接收到结束字符才表征一帧数据的结束，否则继续接收直到缓冲区满（1024 字节）。
0	禁止	使能	接收到结束字符才表征一帧数据的结束，否则继续接收直到缓冲区满（1024 字节）。
0	使能	禁止	Error 报错，缺少结束字符。
0	禁止	禁止	Error 报错，缺少开始字符和结束字符。
不为 0	使能	禁止	收到开始字符后继续接收，并依次排放，直到接满 DataLength 个字节（包括开始字符）后结束本次接收过程。
不为 0	使能	使能	收到开始字符后继续接收，直到接满 DataLength 个字节（包括开始字符）或收到结束字符，结束本次

			接收过程。
不为 0	禁止	使能	指令有效即开始接收，直到接满 DataLength 个字节（包括开始字符）或收到结束字符，结束本次接收过程。
不为 0	禁止	禁止	指令有效即开始接收，直到接满 DataLength 个字节，结束本次接收过程。

表 5-3-6　Mode 参数详细说明

Mode（BYTE）							
7	6	5	4	3	2	1	0
未定义		接收完数据中断	单字节接收中断	空闲时间控制	结束字符控制	开始字符控制	超时控制

超时控制

0：禁止超时控制
1：启动超时控制

开始字符控制

0：禁止开始字符控制
1：启动开始字符控制

结束字符控制

0：禁止结束字符控制
1：启动结束字符控制

空闲时间控制

0：　禁止空闲时间控制
1：　启动空闲时间控制

单字节接收中断

0：禁止单字节接收中断功能
1：启动单字节接收中断功能

接收完数据中断

0：禁止接收完数据中断功能
1：启动接收完数据中断功能

提示：

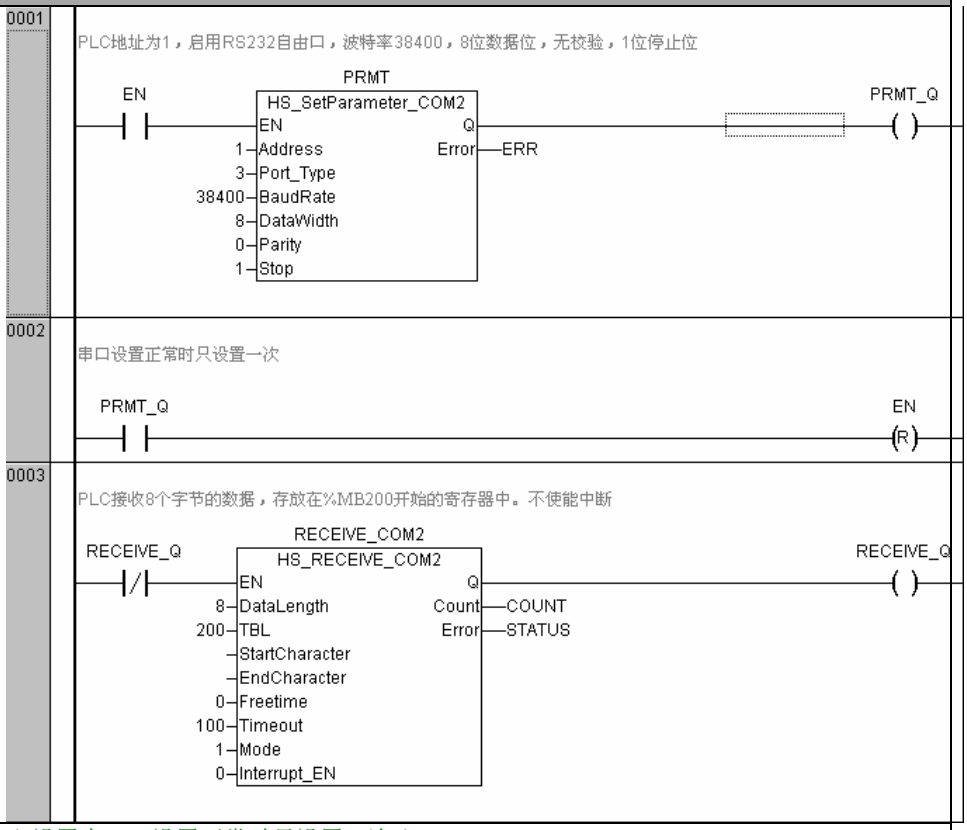
- COM2 口通讯功能块**不可声明为 RETAIN 区变量**，不支持掉电保护，不支持冗余。
- COM2 口通讯**不支持多实例应用**，即在一个工程中每个 COM2 口通讯指令只能被调用一次。
- 调用 HS_RECEIVE_COM2 主站指令之前应先调用并使能 HS_SetParameter_COM2 进行串口基本参数设置，激活自由协议。
- 对于输入参数如果不做任何设置，引脚为空(无变量定义)，则启用默认设定值。

✧ 指令使用举例

变量定义

VAR

	名称	地址	类型	初始值	注释
0001	PRMT		HS_SetParameter_		
0002	EN		BOOL	1	使能
0003	ERR		BYTE		
0004	PRMT_Q		BOOL		
0005	RECEIVE_Q		BOOL		
0006	RECEIVE_COM2		HS_RECEIVE_CO		
0007	STATUS		BYTE		
0008	COUNT		WORD		

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	(*设置串口，设置正常时只设置一次*) PRMT(EN:=EN, Address:=1, Port_Type:=3, BaudRate:= 38400, DataWidth:= 8, Parity:=0 , Stop:= 1, Q=>PRMT_Q , Error=> err); IF (PRMT_Q=1) THEN EN:=0; END_IF (*PLC 接收 8 个字节的数据，存放在%MB200 开始的寄存器中，不使能中断*) RECEIVE_COM1(EN:=NOT RECEIVE_Q , DataLength:= 8, TBL:= 200, Freetime:=0 ,Timeout:= 100,Mode:= 1, Interrupt_EN:=0 , Q=>RECEIVE_Q ,Count=>COUNT ,Error=>STATUS);

程序说明：

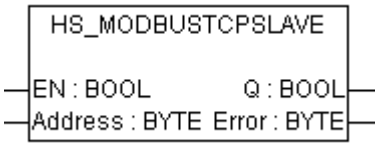
- 设置 PLC 的地址为 1，启用 RS232，波特率 38400，8 位数据位，无校验，1 位停止位。
- PLC 每隔一个扫描周期接收 8 个字节数据分别保存到%MB200-%MB207 中。
- 不使能中断。
- 在一个工程中每个 COM1 口通讯指令只能被调用一次。

5.3.3 以太网通讯指令

5.3.3.1 HS_ModBusTCPSlave——ModBus TCP 从站通讯

- **功能**：提供 Modbus TCP 从站协议，以便响应 Modbus 主站的数据请求。
- **指令示例**：

梯形图语言示例



- **参数说明**：

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：上升沿使能，高电平有效	0
Address	BYTE	本站地址	范围 1-247	1
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	设置完成	0：未完成 1：已完成	0
Error	BYTE	错误信息	为 0：正确 不为 0：有错误 =1：站地址有错误 =2：网络通讯失败	0

- ✧ **指令使用举例**

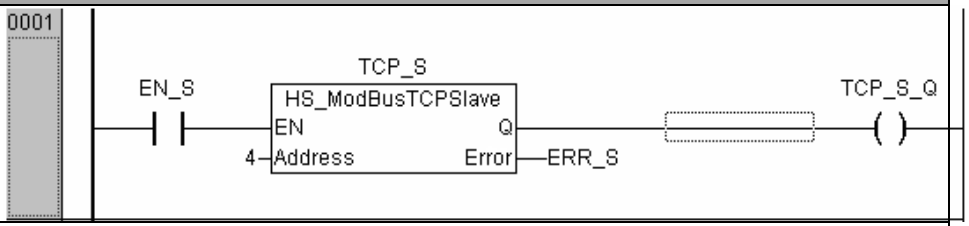
变量定义

VAR

	名称	地址	类型	初始值	注释
0001	EN_S		BOOL	TRUE	初值为1
0002	ERR_S		BYTE		
0003	TCP_S_Q		BOOL		
0004	TCP_S		HS_ModBusTCPSla		

编程语言 | **程 序**

梯形图（LD）



结构化文本（ST）

```
TCP_S (EN:=EN_S, Address:=4,
    Q=>TCP_S_Q, Error=>ERR_S);
```

程序说明：

- 上电或全下装后，EN 上升沿有效，设置完毕后，Q 等于 TRUE。
- 设置 PLC 为 ModBus TCP 4 号从站。

提示：

- HS_ModBusTCPSlave 功能块，对象名**不可声明为 RETAIN 型变量**，不支持掉电保护类型。
- 对于输入参数如果不做任何设置，引脚为空(无变量定义)，则启用默认设定值。
- 由于 **EN** 是上升沿有效，所以调试中如果本指令的输入参数发生改变需要进行全下载，或者在线下载（增量下载）后将 **EN** 置位，形成上升沿，以使新的参数生效。
- 从站支持最多与 32 个主站进行通讯。
- 可支持多链路。

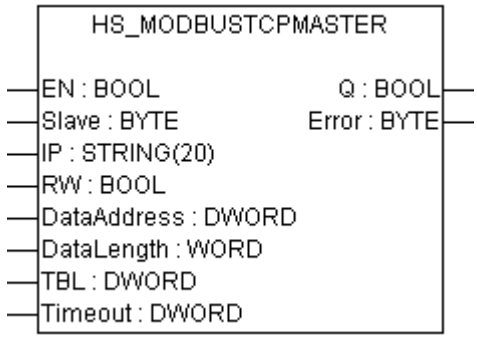
5.3.3.2 HS_ModBusTCPMaster——ModBus TCP 主站通讯

- **功能**：提供实现 ModBus TCP 协议的主站数据读写操作的用户接口。

当 HS_ModBusTCPMaster 模块使能有效时，CPU 模块将作为主站发出数据请求，并对从站的数据响应进行处理。

- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：上升沿使能，高电平有效	0
Slave	BYTE	从站地址	Modbus 从站地址，1-247	1
IP	String	从站 IP 地址	注意：该变量为字符串变量，其头尾需加单引号，例如 ’129.0.0.10’	空
RW	BOOL	读/写选择	0：读取数据 1：写入数据	0
DataAddress	DWORD	从站存放数据的地址	详见表 5-3-7	1
DataLength	WORD	数据长度	对于不同的功能码，ModBus/TCP 协议对长度有不同的要求： 功能码 1，长度 1-1968，单位：位 功能码 2，长度 1-1968，单位：位 功能码 3，长度 1-123，单位：字 功能码 4，长度 1-123，单位：字 功能码 5，长度 1，单位：位 功能码 6，长度 1，单位：字 功能码 15，长度 1-800，单位：位 功能码 16，长度 1-100，单位：字	1
TBL	DWORD	主站存放数据的地址	诸如 200，表示存放地址为%MW200 开始的一段空间。如果是读功能块，读回的数据值存放到这个数据区中；例如 3 号从站 3050 地址的大小为	0

			1000，则%MW200 存放 1000。如果是写功能块，要写出的数据值放到这个数据区中。例如向 3 号从站的 3050 地址写入 500，则%MW200 存放 500。	
Timeout	DWORD	超时时间 (ms)	对于 LK205/LK207，最小时间单位为 30ms。 对于 LK209/LK210，最小时间单位为 150ms。	150
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	完成标志	0：未完成 1：已完成	0
Error	BYTE	错误信息	为 0：数据接收正确 不为 0：数据接收错误 =1：超时时间设定过小 =2：从站地址（Slave）错 =3：IP 地址错 =4：从站存放数据的地址（DataAddress）错 =5：数据长度（DataLength）错 =6：主站存放数据的地址（TBL）越界 =7：功能码错 =8：获取用户空间指针失败 =9：网络通信失败 =10：应答异常错 =11：超时	0

表 5-3-7 DataAddress 参数详细说明

参数说明	
最高位	低五位
0：开出	Modbus 原始地址(大于等于 5000 的部分属于 M 区)
1：开入	
3：模入	
4：模出	
从站的地址采用标准 Modbus RTU 驱动的地址填写方式，即采用六位数（十进制），第六位只能选择 0、1、3、4，分别表示 Modbus 不同的数据类型，低五位表示寄存器地址（地址范围 0-65535）。 **例如：000001，表示 Modbus 地址为 0001 的开出点** **403050，表示 Modbus 地址为 3050 的模出点**	

M 区地址与 Modbus 原始地址的对应关系

Modbus 功能码	M 区地址	Modbus 原始地址	举例	备注
0：开出 1：开入	%MXA.B	5000+A*16+B	%MX50.7 对应于 5000+50*16+7=5807	A 为字号； B 为位号，范围 0~15
3：模入 4：模出	%MWA	5000+A	%MW100 对应于 5000+100=5100	

提示：

- HS_ModBusTCPMaster 功能块**不可声明为 RETAIN 型变量**，不支持掉电保护。
- Modbus 主站指令自动进行超时判断。
- Modbus 主站支持的功能码包括 1、2、3、4、5、6、15、16。
- 对于输入参数如果不做任何设置，引脚为空(无变量定义)，则启用默认设定值。
- 可支持多链路。

✧ 指令使用举例

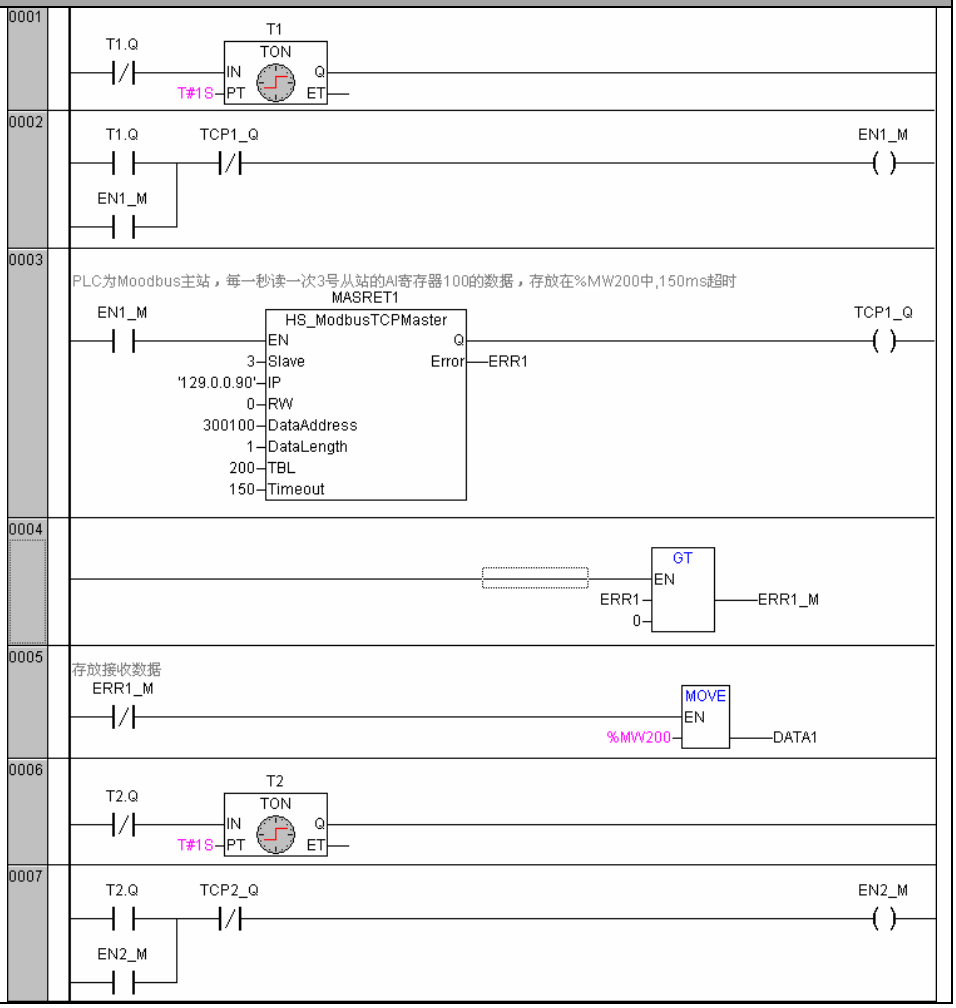
变量定义

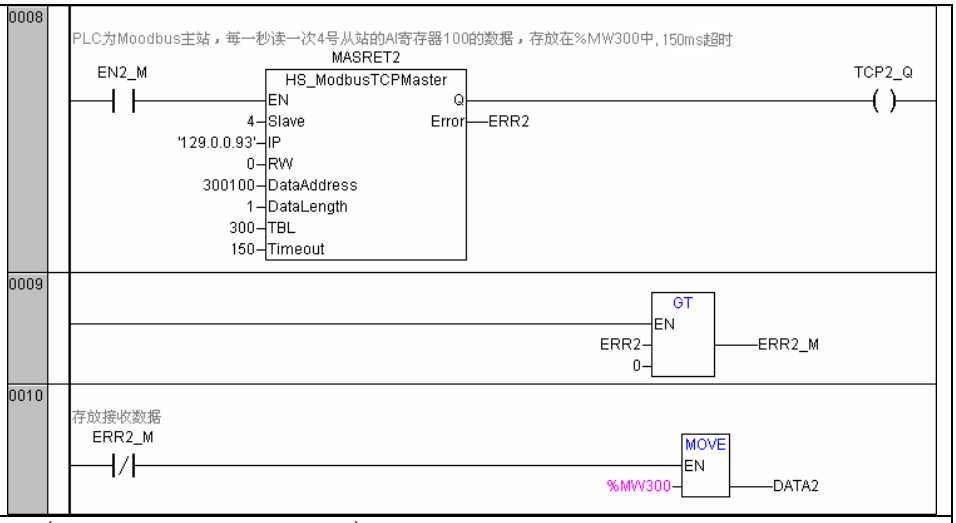
VAR

	名称	地址	类型	初始值	注释
0001	MASRET1		HS_ModbusTCPMa		
0002	T1		TON		
0003	EN1_M		BOOL		
0004	TCP1_Q		BOOL		
0005	ERR1		BYTE		
0006	ERR1_M		BOOL		
0007	DATA1		WORD		
0008	T2		TON		
0009	TCP2_Q		BOOL		
0010	EN2_M		BOOL		
0011	MASRET2		HS_ModbusTCPMa		
0012	ERR2		BYTE		
0013	ERR2_M		BOOL		
0014	DATA2		WORD		

编程语言 | **程 序**

梯形图（LD）



<table>
<tr><td></td><td> 
</td></tr>
<tr><td>结构化文本
（ST）</td><td>T1 (IN:= NOT T1.Q , PT:=T#1S);
IF ((T1.Q=TRUE OR EN1_M=TRUE) AND TCP1_Q=FALSE) THEN
EN1_M:=TRUE;
END_IF
(*PLC 为 Modbus 主站，每一秒读一次 3 号从站的 AI 寄存器 100 的数据，存放在%MW200 中，150ms 超时*)
MASTER1 (EN:=EN1_M, Slave:=3, IP:='129.0.0.90' , RW:=0,
DataAddress:=300100, DataLength:=1, TBL:=200, Timeout:=150,
Q=>TCP1_Q, Error=>ERR1);
IF ERR1=0 THEN
DATA1:=%MW200; (*通讯成功，则存放接收数据*)
END_IF
T2 (IN:= NOT T2.Q , PT:=T#1S);
IF ((T2.Q=TRUE OR EN2_M=TRUE) AND TCP2_Q=FALSE) THEN
EN2_M:=TRUE;
END_IF
(*PLC 为 Modbus 主站，每一秒读一次 4 号从站的 AI 寄存器 100 的数据，存放在%MW300 中，50ms 超时*)
MASTER2 (EN:=EN2_M, Slave:=4, IP:='129.0.0.93' , RW:=0,
DataAddress:=300100, DataLength:=1, TBL:=300, Timeout:=150,
Q=>TCP2_Q, Error=>ERR2);
IF ERR2=0 THEN
DATA2:=%MW300; (*通讯成功，则存放接收数据*)
END_IF</td></tr>
</table>

程序说明：

- 本例中主站每秒发出读取命令，收到返回信息（Q=1）后，一次通讯结束。
- 本例中读取 IP 地址为 128.0.0.90 的从站的 100 寄存器的数据，存放在%MW200 中。超过 150ms 从站无应答，判定为超时，不使能中断。
- 每秒读取一次 IP 地址为 128.0.0.93 的从站的 100 寄存器的数据存放在%MW300。超过 50ms 从站无应答判定为超时，不使能中断。

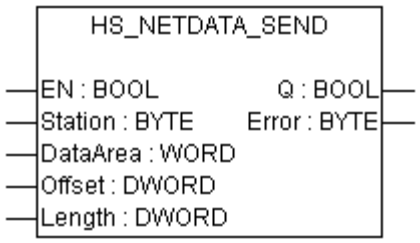
5.3.4 站间引用通讯指令

- “站”的概念：即 PLC 上拨码开关表示的站号，范围 10~99。
- 注意：站间引用通讯指令是 PowerPro V4.3.1B 版本软件中新增的指令，它适用于 LK202-B01, LK205-B01, LK207-A04, LK209-B01, LK210-B06 以及更高型号的主控，如果用户的主控型号低于以上几类，请到 Backup Library 中调用适合您主控的 HS_Communication.lib 库版本文件，具体操作方法请参见附录 3。

5.3.4.1 HS_NetData_Send——站间引用通讯数据发送

- **功能**：提供站间引用发送端参数配置和显示发送错误信息的用户接口。
- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	配置使能端	0：无效 1：上升沿使能，高电平有效	0
Station	BYTE	接收站站号	范围：10-99	0
DataArea	WORD	数据区	=0：表示 M 区； =1：表示 I 区； =2：表示 Q 区； 其他，非法	0
Offset	DWORD	数据区的偏移字地址	当 DataArea=0 时，Offset 最大为(2M/2-Length)； 当 DataArea=1 或 DataArea=2 时，Offset 最大为(20K/2-Length)	0
Length	DWORD	数据大小	最大为 700 字	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	配置完成标志	0：未完成 1：已完成	
Error	BYTE	错误信息	=0，数据发送正确 =1，接收站站号(Station)有误 =2，发送站数据区(DataArea)有误 =3，数据区偏移地址(Offset)有误 =4，发送数据大小(Length)超出最大限制 =6，网络通讯失败 =9，组站超过 32 个	

提示：

- 站间引用支持双网冗余和双机冗余，保证主从机数据的一致性；
- 站间引用只支持整字传输(即传输一段大小为字整数倍的内存区的数据)；
- 向一个站发送数据就需要组一个发送功能块，向 n 个站发送 n 个独立的数据块就需要组 n 个发送功能块；
- 相同的两站之间同一时刻只能有一个数据块的发送接收；
- 一个站最多能同时向 32 个站发送数据；
- 站间引用最大数据通讯量限制在每次 700 字之内；
- 发送站只有主机可以发送数据，主从机可以实现无扰切换，切换后新的主机继续重新发送数据；
- 站间引用支持 IP 地址修改的所有网段，只要修改后，两个站仍然处于同一网段内，就可以继续通讯；
- 站间引用不受 IEC 任务周期的影响，发送端按照 250ms 的周期发送数据，接收端按照 50ms 的周期接收数据。即 EN 端上升沿触发一次，使能配置有效后，保持高电平即可，正常通讯过程不受 IEC 任务调度影响。如果修改了配置，需要 EN 端重新上升沿使能，才能使新配置有效。

✧ 指令使用举例

部分变量定义

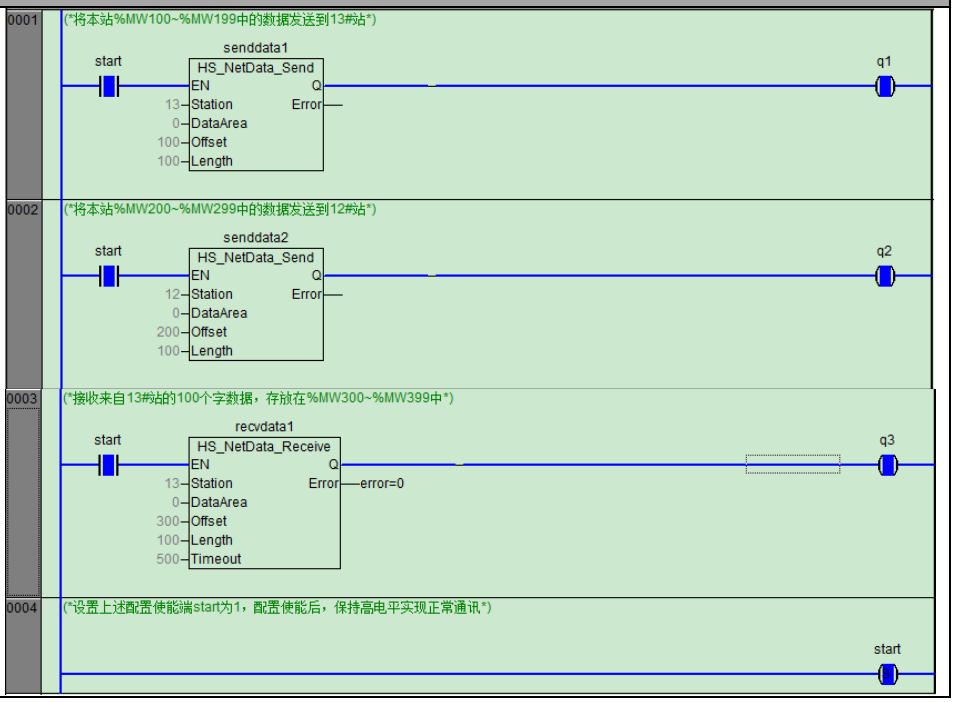
VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

	名称	地址	类型	初始值	注释
0001	senddata1		HS_NetData_Send		
0002	start		BOOL		
0003	q1		BOOL		
0004	senddata2		HS_NetData_Send		
0005	q2		BOOL		
0006	recvdata1		HS_NetData_Recei		
0007	error		BYTE		
0008	q3		BOOL		

限于篇幅，其余变量定义省略。

编程语言：梯形图（LD）

程　序



结构化文本（ST）	(*将本站%MW100~%MW199 中的数据发送到 13#站*) senddata1(EN:=start,Station:=13,DataArea:=0,Offset:=100, Length:=100,Q=>q1); (*将本站%MW200~%MW299 中的数据发送到 12#站*) senddata2(EN:=start,Station:=12,DataArea:=0,Offset:=200, Length:=100,Q=>q2); (*接收来自 13#站的 100 个字数据，存放在%MW300~%MW399 中*) recvdata1(EN:=start,Station:=13,DataArea:=0,Offset:=300, Length:=100,Timeout:=500,Q=>q3,Error=>error); (*设置上述配置使能端 start 为 1，配置使能后，保持高电平实现正常通讯*) start:=1;

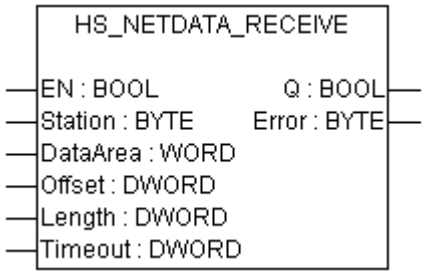
程序说明：

- 本站同时向 13#和 12#两个站发送数据（一个站最多能同时向 32 个站发送数据）；
- 本站同时接收来自 13#站的数据，具体见下面 HS_NetData_Receive 模块；
- 相同的两站之间同一时刻只能有一个数据块的发送接收（程序里没有在相同两站之间进行多个数据块的发送接收）。

5.3.4.2 HS_NetData_Receive——站间引用通讯数据接收

- **功能**：提供站间引用接收端参数配置和显示接收错误信息的用户接口。
- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：上升沿使能，高电平有效	0
Station	BYTE	发送站站号	范围：10-99	0
DataArea	WORD	数据区	=0：表示 M 区； 其他，非法 **注意**：接收的数据只能放在 M 区	0
Offset	DWORD	数据区的偏移字地址	Offset 最大为(2M/2-Length)；	0
Length	DWORD	数据大小	最大为 700 字	0
Timeout	DWORD	超时时间(ms)	最小值为 500ms，若小于 500ms,Error 会报错	500
输出	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	配置完成标志	0：未完成 1：已完成	
Error	BYTE	错误信息	=0，数据接收正确 =1，发送站站号(Station)有误 =2，接收站数据区(DataArea)有误 =3，数据区偏移地址(Offset)有误	

			=4，接收数据大小(Length)超出最大限制 =5，超时时间(Timeout)设置有误 =6，网络通讯失败,接收数据超时 =7，接收数据大小大于发送数据大小 =8，接收数据大小小于发送数据大小 =9，组站超过 32 个	

提示：

- 站间引用支持双网冗余和双机冗余，保证主从机数据的一致性；
- 站间引用只支持整字传输(即传输一段大小为字整数倍的内存区的数据)；
- 要接收另一个站发来的数据就需要组一个接收功能块，要接收 n 个站发来的 n 个独立的数据块就需要组 n 个接收功能块；
- 相同的两站之间同一时刻只能有一个数据块的发送接收；
- 一个站最多能同时接收来自 32 个站的数据。
- 站间引用最大数据通讯量限制在每次 700 字之内；
- 接收站的主从机均可以接收到发送站发来的数据(双机冗余)；
- 站间引用支持 IP 地址修改的所有网段，只要修改后，两个站仍处同一网段内，就可继续通讯；
- 站间引用不受 IEC 任务周期的影响，发送端按照 250ms 的周期发送数据，接收端按照 50ms 的周期接收数据。即 EN 端上升沿触发一次，使能配置有效后，保持高电平即可，正常通讯过程不受 IEC 任务调度影响。如果修改了配置，需要 EN 端重新上升沿使能，才能使新配置有效。

✧ 指令使用举例

部分变量定义

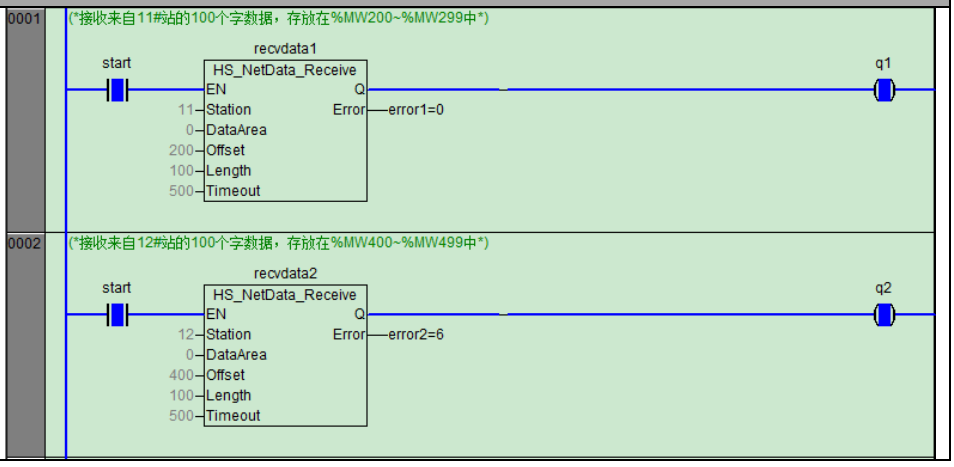
VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT

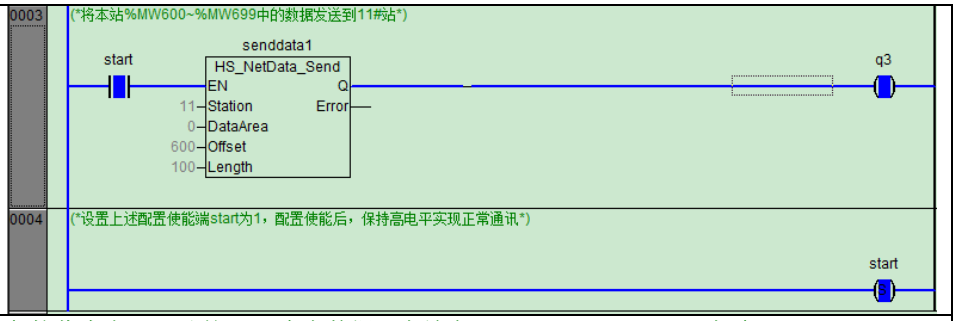
	名称	地址	类型	初始值	注释
0001	recvdata1		HS_NetData_Recei		
0002	q1		BOOL		
0003	start		BOOL		
0004	recvdata2		HS_NetData_Recei		
0005	q2		BOOL		
0006	senddata1		HS_NetData_Send		
0007	error1		BYTE		
0008	error2		BYTE		
0009	q3		BOOL		

限于篇幅，其余变量定义省略。

程语言 | **程　序**

梯形图（LD）



	
结构化文本（ST）	(*接收来自 11#站的 100 个字数据，存放在%MW200~%MW299 中*) recvdata1(EN:=start,Station:=11,DataArea:=0,Offset:=200, Length:=100,Timeout:=500,Q=>q1,Error=>error1); (*接收来自 12#站的 100 个字数据，存放在%MW400~%MW499 中*) recvdata2(EN:=start,Station:=12,DataArea:=0,Offset:=400, Length:=100,Timeout:=500,Q=>q2,Error=>error2); (*将本站%MW600~%MW699 的数据发送到 11#站*) senddata1(EN:=start,Station:=11,DataArea:=0,Offset:=600, Length:=100,Q=>q3); (*设置上述配置使能端 start 为 1，配置使能后，保持高电平实现正常通讯*) start:=1;

程序说明：

- 某站 PLC 可同时作为站间引用的发送站和接收站。
- 向 11#站发送数据的同时，也接收来自 11#站、12#站的数据（一个站最多能同时向 32 个站发送数据，并且最多能同时接收来自 32 个站的数据）。
- 程序中 recvdata2 的输出端 Error 上报错误码 6，说明本站接收来自 12#站的数据超时。可能原因有：12#站 PLC 不存在，或者与 12#通讯网络出现问题，或者网络环境较差，通讯超时。
- 相同的两站之间同一时刻只能有一个数据块的发送接收（程序里没有在相同两站之间进行多个数据块的发送接收）。

5.4 SOE 指令库 HS_SOE.Lib

SOE（Sequence Of Event）：事故顺序记录，根据 DI 输入变化记录时序来判断监控开关动作发生时序，主要用于事故后的事故分析。因其时间精度高，而应用于重要现场开关量信号的监测。

SOE 模块：是一种带时间戳（Time Stamp）的 DI 采集模块。

SOE 包：每次向缓冲区中写入若干 SOE 事件，这若干个 SOE 事件的集合叫做一个 SOE 包。

包号：每个在 SOE 缓冲区中的 SOE 包对应的号码叫做包号。

- **说明：**用户在 PowerProV4 中添加库 HS_SOE.lib 后，可以通过两种指令对控制器中存储的 SOE 数据进行读取，分别是 SOE 存入 M 区指令 HS_DP_SOE_M 和 DP SOE 读取指令 HS_DP_SOE_Read。

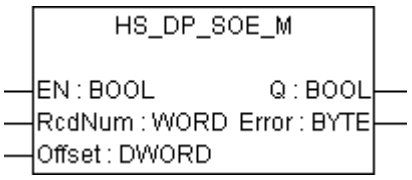
5.4.1 DP SOE 指令

5.4.1.1 HS_DP_SOE_M——SOE 存入 M 区指令

- **注意：**该指令是 PowerProV4.3.0 版本软件中新增的指令，它适用于 LK202-A01,

LK205-A01，LK207-A03，LK209-A01，LK210-B05 以及更高型号的主控，如果用户的主控型号低于以上几类，请到 Backup Library 中调用适合您主控的 HS_SOE.Lib 库版本文件，具体操作方法请参见附录 3。

- **功能**：配置 M 区中的 SOE 缓冲区以及是否将 SOE 数据存入 M 区中 SOE 缓冲区。
- **指令示例：**

梯形图语言示例


- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能标志	0-不使能，这时 SOE 存入 M 区功能不再有效; 1-上升沿使能，根据 RcdNum 和 Offset 参数设置 SOE 缓冲区,若设置成功(Error 输出为 0)则 SOE 存入 M 区功能有效，若设置不成功(Error 输出不为 0)则 SOE 存入 M 区功能是否有效与设置前相同。	0
RcdNum	WORD	缓冲区能容纳的 SOE 记录的最大数目	该参数的有效范围为 1-100; 假设该参数的值为 k，且 1<＝k<=100，那么缓冲区的大小为(16*k+8)字节。	14
Offset	DWORD	SOE 缓冲区在 M 区中的起始偏移字节地址	设置 M 区中 SOE 缓冲区的起始字节偏移地址，最小值为 4000，假设缓冲区的大小为 n 字节，那么缓冲区的起始地址必须不大于(2097152-n)并且能被 4 整除（2097152 为 2 的 21 次幂）。	4000
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	配置完成标志	0：配置未完成 1：配置完成	
Error	BYTE	错误信息	=0：设置成功 =1：最大数目 RcdNum 无效 =2：偏移地址 Offset 无效 =4：其它错误	

- **SOE 数据在 M 区的存储格式**

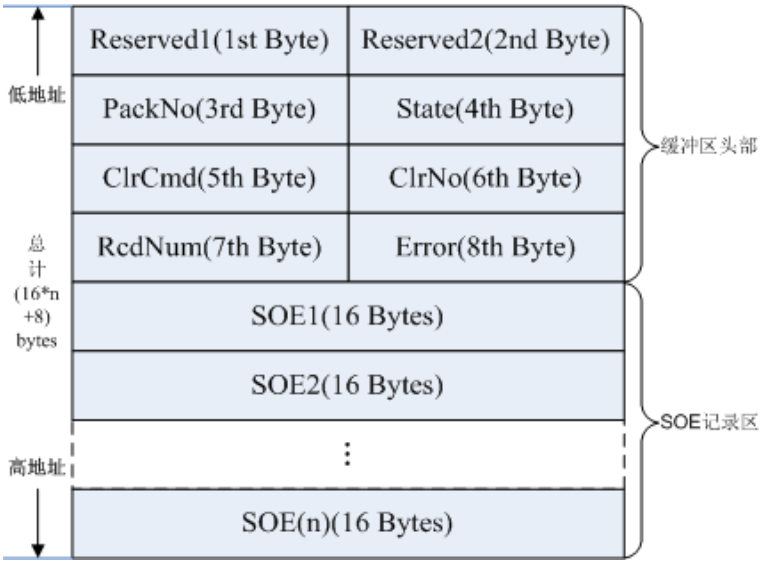


图 5-4-1 SOE 数据在 M 区的存储格式

缓冲区由缓冲区头部和后面的 SOE 记录区两部分构成。缓冲区头部包含 8 个字节，SOE 记录区最多可以存放 n 条 SOE 记录，每条 SOE 记录所占空间为 16 bytes，所以该缓冲区的大小为：(16 n +8)(bytes)。缓冲区头部的 8 个字节含义如下：

Reserve1，Reserve2：保留字节 1，保留字节 2。

PackNo：缓冲区中当前 SOE 记录对应的包号，从 1 到 255 循环使用。

State：控制器运行状态(0：为主机或单机状态；1：为从机状态)。

ClrCmd：清空命令，用户需先发送清空命令来清除 M 区中 SOE 缓冲区的 SOE 包，这样新的 SOE 包才可以被写入缓冲区。

ClrNo：清空包号。只有当 ClrCmd 等于 1，并且 ClrNo 等于 PackNo 时，清空操作才会执行。

RcdNum：当前缓冲区中 SOE 记录的条数。

Error：错误状态指示。=0：表明没有错误；
=2：表明 ClrNo 不等于 PackNo；
=4：表明其它类型错误。

每条 SOE 记录在 M 区的存储空间为 16 bytes，按照 4 字节对齐存储，具体格式如下：

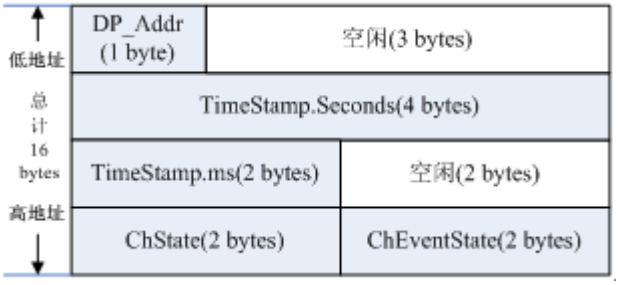


图 5-4-2 每条 SOE 数据的具体格式

图中各变量的含义如下：

DP_Addr：发生 SOE 的设备地址。

TimeStamp.Seconds：时戳信息中 SOE 事件发生时的秒时间。

TimeStamp.ms：时戳信息中 SOE 事件发生时的毫秒时间。

ChState：0~15 通道当时状态，0 = OFF，1 = ON。

ChEventState：0~15 通道产生事件的标志，=0：表示该通道无跳变事件，
=1：表示该通道发生通道跳变事件。

✧ 读取方式说明

用户通过 HS_DP_SOE_M 功能块使能 SOE 存入 M 区的 SOE 缓冲区功能后（该功能块使能后则 M 区开始存入 SOE 数据），用户可以使用 ModbusTCP、Modbus 串口、以及读写标签变量方式读取。具体步骤如下：

1. 判断 ClrCmd 字节是否为 0，如果不为 0，表明 RTS 程序中处理 SOE 缓冲区操作正在进行，等待 T 毫秒（T>=50ms），再次进入步骤 1；如果 ClrCmd 字节为 0，进入步骤 2。

2. 通过 State 字节读取控制器的主从状态（0，主机；1，从机），进入步骤 3。

3. 判断 Error 字节，如果 Error 等于 0，表明程序正确执行，进入 4；如果 Error 等于 2，表明当 ClrCmd 等于 1 时，PackNo 与 ClrNo 不相等，进入 5。如果 Error 等于 4 或其他未定义值，可以记录错误值和错误时间，进入步骤 5。

4. 通过 RcdNum 字节读取 SOE 记录的数目，如果 RcdNum 等于 0，表明没有新的 SOE 事件生成；如果 RcdNum 不为 0，表明 SOE 记录区中存放了 RcdNum 条 SOE 记录，SOE 记录存放的首地址 =（SOE 缓冲区的首地址 + 缓冲区头部的大小）=（SOE 缓冲区的首地址 + 8），用户可以对其进行读取，并按照其存储格式对其进行解释。此次读取 SOE 操作完成。进入步骤 5。

5. 如需读取新的 SOE 数据，则先读取缓冲区中的 PackNo 字节，将其值赋给 ClrNo 字节，然后将 ClrCmd 字节赋值 1，再次进入步骤 1；如果不需读取新的 SOE 数据，则直接退出。

✧ 指令使用举例

部分变量定义

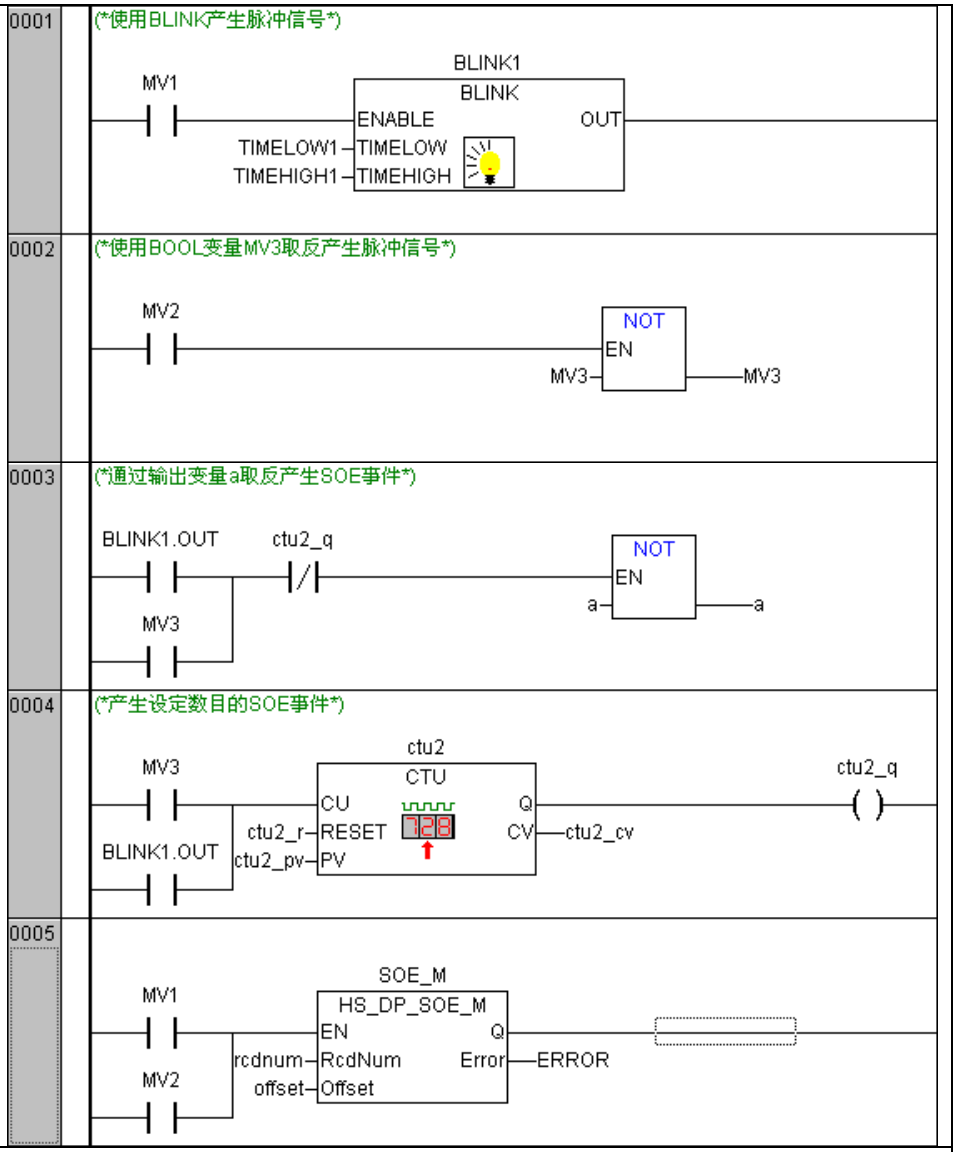
VAR

	Name	Address	Type	Initial	Comment
0001	SOE_M		HS_DP_SOE_M		
0002	en		BOOL		
0003	RcdNum		WORD	100	SOE记录的最大数目
0004	Offset		DWORD	4000	SOE缓冲区在M区中的起始偏移字节地址
0005	error		BYTE		
0006	SOE_M1_PackNo	%MB4002	BYTE		包号
0007	SOE_M1_State	%MB4003	BYTE		控制器运行状态
0008	SOE_M1_ClrCmd	%MB4004	BYTE		清空命令
0009	SOE_M1_ClrNo	%MB4005	BYTE		清空包号
0010	SOE_M1_RcdNum	%MB4006	BYTE		当前Soe条数
0011	SOE_M1_error	%MB4007	BYTE		错误标志位
0012	SOE_M1_event1	%MB4008	ARRAY[0..15] OF BY		第1条SOE记录
0013	SOE_M1_event2	%MB4024	ARRAY[0..15] OF BY		第2条SOE记录
0014	SOE_M1_event3	%MB4040	ARRAY[0..15] OF BY		第3条SOE记录
0015	a	%QW2	WORD		

限于篇幅，其余变量定义省略。

编程语言	程　序

梯形图（LD）



结构化文本（ST）

```
(*使用 BLINK 产生脉冲信号*)
BLINK1(ENABLE:= MV1, TIMELOW:= timelow1, TIMEHIGH:=timehigh1 );
IF MV2=TRUE THEN
    MV3:=NOT MV3;    (*使用 BOOL 变量 MV3 取反产生脉冲信号*)
END_IF
IF ((BLINK1.OUT=1 OR MV3=1) AND ctu2_q=0) THEN
    a:=NOT a;        (*通过输出变量 a 取反产生 SOE 事件，a 的地址为%QW2*)
END_IF
(*产生设定数目的 SOE 事件*)
ctu2(CU:=(MV3 OR BLINK1.OUT) ,RESET:=ctu2_r ,PV:= ctu2_pv ,
    Q=>ctu2_q ,CV=>ctu2_cv );
SOE_M( EN:=(MV1 OR MV2) , RcdNum:= rcdnum, Offset:= offset,
    Error=>error );
```

程序说明：

- 配置 M 区中的 SOE 缓冲区，起始偏移地址为 4000，缓冲区所能容纳 SOE 记录的最大数目是 100。
- 定义了 SOE_M1_ClrCmd，SOE_M1_PackNo 等变量来发送清空命令，并且读取 M 区中 SOE 缓冲区的当前情况。
- SOE 读取信号，必须与硬件 LK630 配合使用，PLC 硬件模块配置如下图所示：

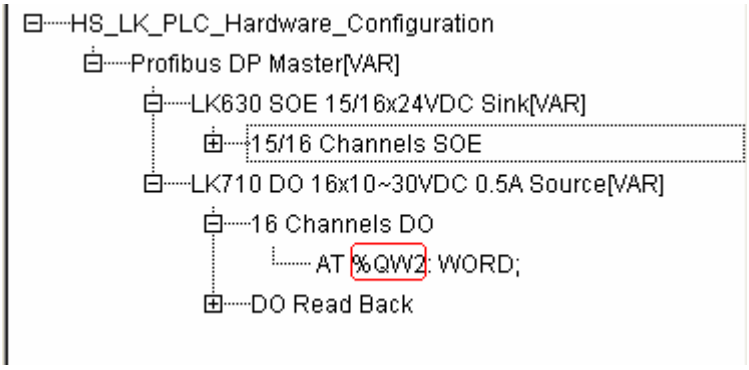
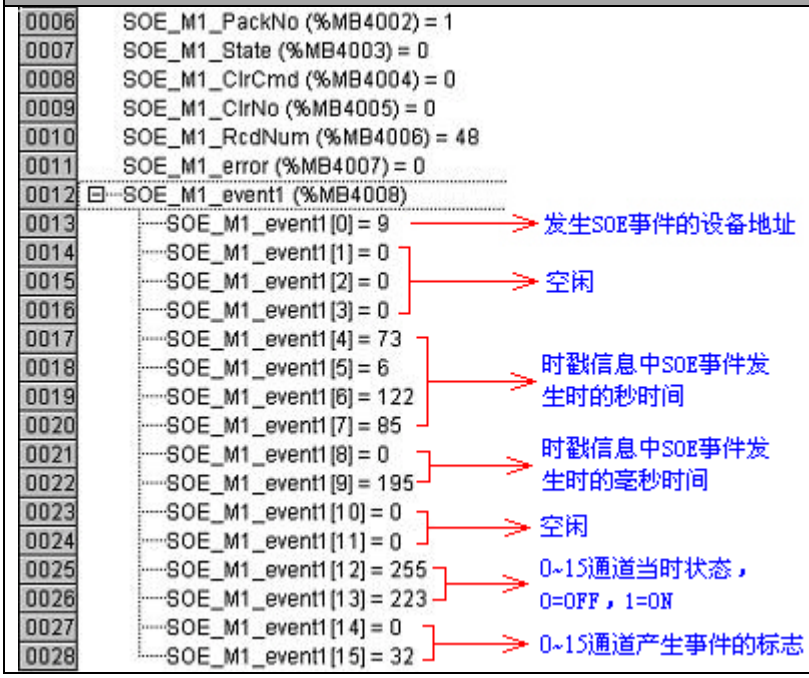


图 5-4-3 PLC 硬件模块配置图

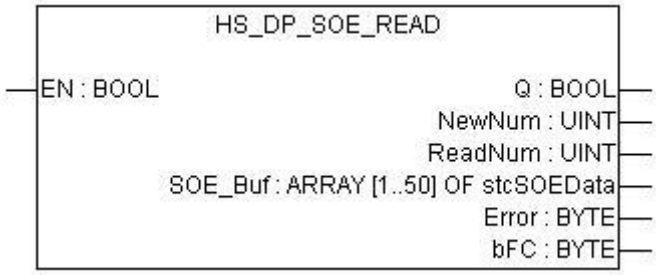
部分变量运行结果说明



5.4.1.2 HS_DP_SOE_Read——DP SOE 读取指令

- **功能**：HS_DP_SOE_Read 功能块提供了读取 SOE 事件的用户接口。
- **说明**：符合《LK PLC 通用 HMI 软件设计指南》所述的与 LK PLC 通讯规则的 HMI 软件均可通过该功能块读取 SOE 事件。
- **指令示例**：

梯形图语言示例



➢ **参数说明：**

⚠**注意：**除了参数 EN 和 Q 外，其余参数最好不要定义变量。

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能标志	0-禁止　　1-使能	0
BufState	BYTE	Buffer 状态	控制器填写完 SOE 数据之后，将该位置 1，若控制器在填写缓冲区的时候发现该位为 1，则不向缓冲区中填写新的数据。操作员站读取完 SOE 数据之后将该位清零。 0-Buffer 非满　　1-Buffer 满	
ClrCmd	BYTE	清空命令	上位机填写，表示要清空缓冲区数据 0-保持当前数据　　1-有清空命令	0
ClrNo	BYTE	需清空的 SOE 包号	需清空的 SOE 包号 1～255 由于网络阻塞会导致上位机对同一包 SOE 数据发送多个清除命令，如果对清除命令没有包号的判断，这样会导致尚未发送的 SOE 数据丢失。	1
PackNo	BYTE	SOE 包号	SOE 包号 1～255 循环计数	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	输出完成标志	0：输出未完成 1：输出完成	
SOE_Buf	stcSOEData	存放若干个 SOE 事件的缓冲区	存放 SOE 事件的缓冲区，最大可以存放 50 个 SOE 事件。	
NewNum	WORD	新产生的 SOE 事件的个数	该输出用户不可见。 控制器中保存新 SOE 事件个数，1～6000（20 个 SOE 事件*30 个模块*10）。	
ReadNum	WORD	实际读取的 SOE 事件的个数	实际读取的 SOE 事件的个数，也即存放在 Buffer 中的 SOE 事件的个数，1～50	
Error	BYTE	错误信息	该输出用户不可见。 =0：数据读取正确 非 0，在读取 SOE 事件过程中存在错误： =1：SOE 缓冲区不存在 =2：清空包号错误 =3：SOE_Buf[]缓冲区中的数据没有被及时的读走。	

表 5-4-1 stc SOEData 结构中各成员说明

成员	数据类型	功能描述	参数说明	默认值
DP_Addr	BYTE	发生 SOE 的设备地址		
TimeStamp	stcTimeStamp	时戳信息	参见表 5-4-2，时戳数据结构	
ChState	WORD	0……15　通道当时状态	0 = OFF 1 = ON	
ChEventState	WORD	0……15　通道产生事件的标志	0 =　该通道无跳变事件 1 =　该通道发生通道跳变事件	

表 5-4-2 stcTimeStamp 结构中各成员说明

成员	数据类型	功能描述	参数说明	默认值
Seconds	DWORD	SOE 事件发生时的秒时间		
Ms	WORD	SOE 事件发生时的毫秒		

		时间		

表 5-4-3 功能块输出的数据结构如下：

Q		
NewNum		
ReadNum		
SOEBuf[1]	DP_Addr	
	TimeStamp	Second
		MilliSecond
	ChState	
	ChEventState	
……		
SOE_Buf[50]		

提示：

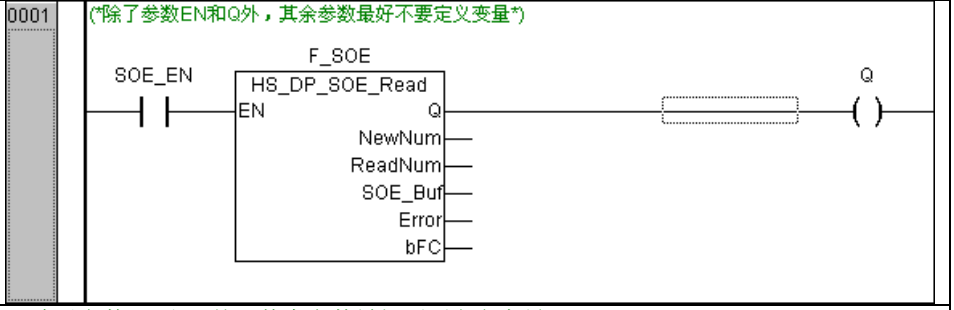
- **SOE 数据的读取**：以 FacView 为例读取 SOE 事件数据。需要在 PowerPro 算法中添加库 HS_SOE.lib，调用功能块 HS_DP_SOE_Read，FacView 通过变量表获取 SOE 数据。
- **再次强调：** 除了参数 EN 和 Q 外，其余参数最好不要定义变量。
- 通过调用 HS_DP_SOE_Read 功能块，可将若干 SOE 事件作为一包数据放入到 SOE_Buf 缓冲区中，(并对该数据包分配一个包号 PackNo，包号从 1 到 255 循环计数)，以便上位机(如 FacView)取出 SOE_Buf 中的数据进一步处理。
- 首次调用该功能块时，由于 SOE_Buf 缓冲区是空的，因此不需要清空命令。以后每次调用该功能块时，需要用户发送清空命令(ClrCmd=1，ClrNo=x，由上位机发送)来清除 SOE_Buf 中的 SOE 包。这样，新的 SOE 包才可以写入到 SOE_Buf 中。
- **注意**：清空包号 ClrNo 和 SOE_Buf 中记录的包号 PackNo 一致时，才能成功清空 SOE_Buf 中的 SOE 包。
- 为减轻网络负荷，每次最多只读取 50 条记录。

✧ **指令使用举例**

变量定义

VAR

	名称	地址	类型	初始值	注释
0001	SOE_EN		BOOL		
0002	F_SOE		HS_DP_SOE_Read		
0003	Q		BOOL		

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	(*除了参数 EN 和 Q 外，其余参数最好不要定义变量*) F_SOE(EN:= SOE_EN, Q=>Q);

程序说明：

- 读取 SOE 信号，读取完成则 Q=1。
- SOE 读取信号，必须与硬件 LK630 配合使用。
- DP SOE 功能块的输入输出端数据显示如图 5-4-4 所示：

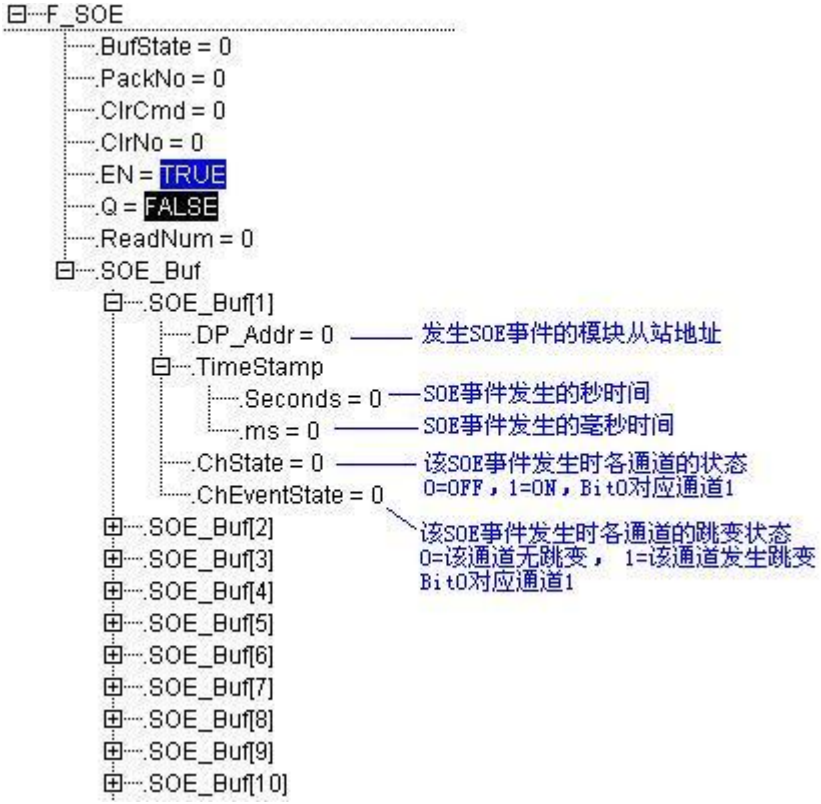


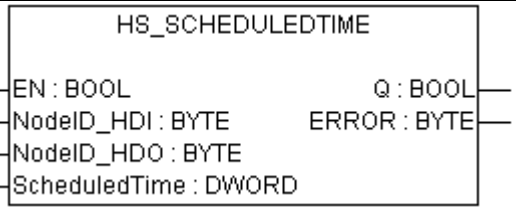
图 5-4-4 SOE 功能块的输入输出端数据

5.5 预置输出指令 HS_ScheduledTime.Lib

5.5.1 HS_ScheduledTime——预置输出指令

- **功能**：用于实现在延迟某个时间后才对通道进行输出，需要延迟的时间称为预置时间。预置时间通过组态设定，最长为 16.7ms。目前 LK750 模块支持预置输出功能。取时间参数的时间戳功能则由 LK650、651、652 模块支持(Time Stamp)。其中时间戳是用来记录输入信号状态变化时刻的一个相对时间值，称为协调系统时间(CST)。预置输出时间可以达到微妙级，对于时间精度要求高的可以使用此功能块。
- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：上升沿有效	0
NodeID_HDI	BYTE	输入模块节点号	合法的模块节点号 2-9	0
NodeID_HDO	BYTE	输出模块节点号	合法的模块节点号 2-9	0
ScheduledTime	DWORD	预置间隔时间	单位微秒 us(1 至 16 777 215 us) （注：考虑到预置时间的特性，建议用户所组的时间大于 IEC 任务循环周期加上主循环周期 10000us，即 Scheduled Time>IEC 任务循环周期+10000us,否则可能会得到非预期的结果）	1000000
输出参数	**数据类型**	**功能描述**	**参数值说明**	**默认值**
Q	BOOL	设置完成	0：未完成 1：已完成	0
ERROR	BYTE	错误信息	0：　正确 非 0：有错误 . 0=1，输入模块节点号有错误 . 1=1，输出模块节点号有错误 . 2=1，间隔时间输入有错误 . 3=1，数据区指针寻址出错	0

✧ **指令使用举例**

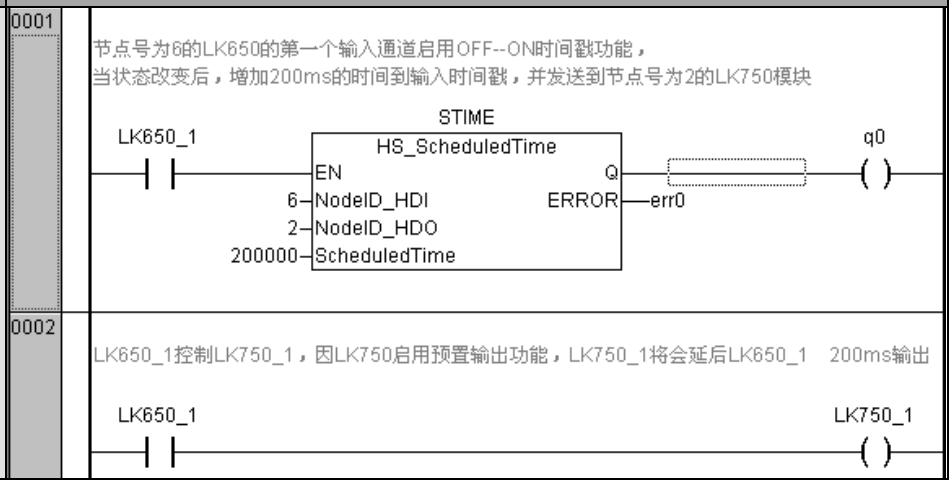
变量定义

VAR

	名称	地址	类型	初始值	注释
0001	STIME		HS_ScheduledTime		
0002	Err0		BYTE		
0003	Q0		BOOL		
0004	Name		BOOL		
0005	LK650_1	%IX8.0	BOOL		LK650的第一个输入通道
0006	LK750_1	%QX0.0	BOOL		LK750的第一个输出通道

编程语言：梯形图（**LD**）

程　序



结构化文本（ST）	STIME (EN:=LK650_1, NodeID_HDI:=6, NodeID_HDO:=2, ScheduledTime:=200000, Q=>q0, ERROR=>err0); IF LK650_1=TRUE THEN LK750_1:=TRUE END_IF

程序说明：

- 预置时间间隔的 ScheduledTime 项为 DWROD 格式，其最大值为 1677215（单位 us）；
- 输入和输出模块的合法地址均为 2~14；
- ERROR 项为错误信息，上层可根据其值判断错误种类。
- 该功能块的功能：从某输入模块中取出时间戳，与预置时间相加，并将值输出到相应输出模块中。同时还实现对相应输入、输出模块的类型的识别及报错功能。
- 输入模块上除标记时间戳点（只有一点）外其余点不具有该功能。

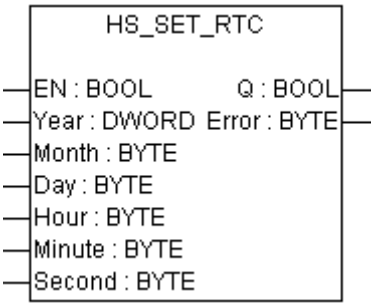
注：以上程序流程为，LK650 指定通道发生变化，在此刻时间戳开始计时，当到达预置时间时，LK750 启用预置的输出。

5.6 实时时钟指令 HS_RTC.Lib

5.6.1 HS_Set_RTC——设置实时时钟

- **功能**：将用户规定的时间设置到 PLC 实时时钟中，冗余系统中，设置实时时钟功能块只针对主机起作用。
- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：上升沿有效	0
Year	DWORD	年	范围 2000-2037	2000
Month	BYTE	月	1-12	1
Day	BYTE	日	1-31	1
Hour	BYTE	时	0-23	0
Minute	BYTE	分	0-59	0
Second	BYTE	秒	0-59	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	**默认值**
Q	BOOL	设置完成	0：未完成 1：已完成	
Error	BYTE	参数设置错误	=0:设置 RTC 时间没有错误 非 0，表示设置 RTC 时间时出现错误 . 0=1：设置年超出规定范围	

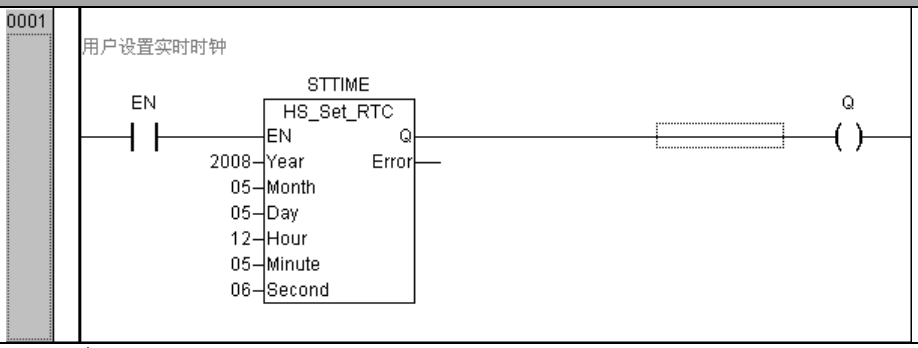
			.1=1：设置月不合理 .2=1：设置日不合理 .3=1：设置时不合理 .4=1：设置分不合理 .5=1：设置秒不合理 .6=1：底层 RTC 没有响应 .7=1：底层 RTC 设置时间不符合预期	

✧ **指令使用举例**

变量定义

VAR

	名称	地址	类型	初始值	注释
0001	EN		BOOL		使能
0002	STTIME		HS_Set_RTC		设置时间
0003	Q		BOOL		设置时间完成

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	STTIME (EN:=EN, Year:=2008, Month:=05, Day:=05, Hour:=12, Minute:=05, Second:=06, Q=>Q);

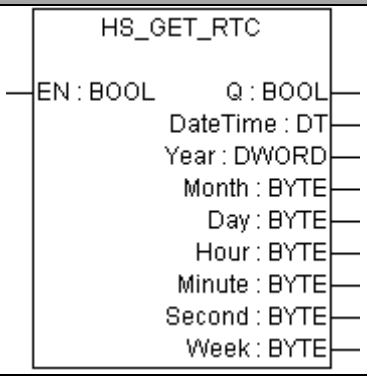
程序说明：

- 当使能 EN 置位时，用户规定的日期、时间、星期等信息将被设置到实时时钟中，设置标志 Q 置 1，设置日期时间不可以超出规定范围。
- 如果 Error=1，是 2 的 0 次方，表示设置年超出规定范围；Error=2，是 2 的 1 次方，表示设置月不合理；Error=8，是 2 的 3 次方，表示设置时不合理，以此类推！

5.6.2 HS_Get_RTC——读取实时时钟

- **功能**：该功能块读取 PLC 硬件实时时钟中的时间。
- **指令示例：**

梯形图语言示例



➢ 参数说明：

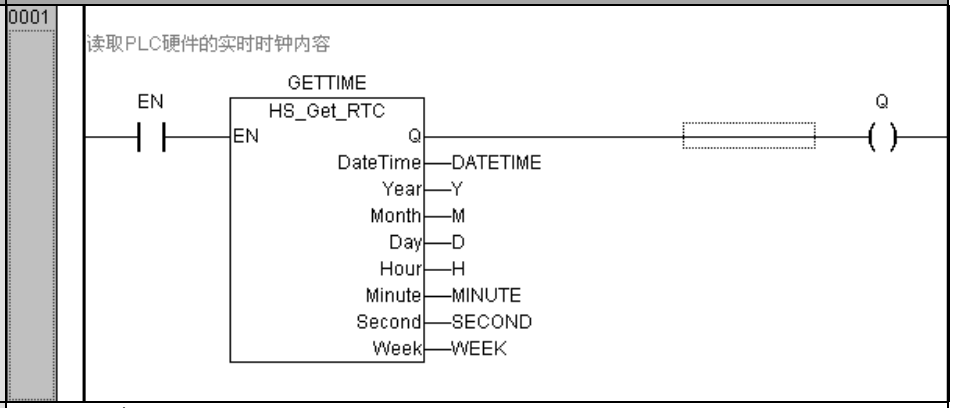
输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：有效	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	**默认值**
Q	BOOL	是否读出数据	0：未读出数据。 1：正确取出数据。	
DateTime	DT	日期/时间	DT#2000-01-01-00:00:00~ DT#2037-12-31-23:59:59	
Year	DWORD	年	2000-2037	
Month	BYTE	月	1-12	
Day	BYTE	日	1-31	
Hour	BYTE	时	0-23	
Minute	BYTE	分	0-59	
Second	BYTE	秒	0-59	
Week	BYTE	星期	1-7	

✧ 指令使用举例

变量定义

VAR

	名称	地址	类型	初始值	注释
0001	EN		BOOL		使能
0002	GETTIME		HS_Get_RTC		读取时间
0003	Q		BOOL		读取时间完成
0004	DATETIME		DT		日期/时间
0005	Y		DWORD		年
0006	M		BYTE		月
0007	D		BYTE		日
0008	H		BYTE		时
0009	MINUTE		BYTE		分
0010	SECOND		BYTE		秒
0011	WEEK		BYTE		星期

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	GETTIME (EN:=EN, Q=>Q, DateTime=>DATETime, Year:=Y, Month:=M, Day:=D, Hour:=H, Minute:=MINUTE, Second:=SECOND, Week:=WEEK);

程序说明：

- 当使能 EN 置位时，既会分别输出硬件实时时钟的日期、时间、星期等信息，也会以 DT 型数据格式整体地输出以上信息，读取标志 Q 置 1。

5.7 诊断指令库 HS_Diagnosis.Lib

注意：诊断指令库是 PowerPro V4.3.0B 版本软件最新增加的指令库，它融合了 CPU 诊断和本地总线模块诊断以及 DP 诊断。该库适用于 LK202-A01, LK205-A01, LK207-A03, LK209-A01, LK210-B05 以及更高型号的主控，如果用户的主控型号低于以上几类，建议用户**不要调用**此库，否则工程将不能下装。

用户按其需要，调用该库中对应文件夹内的诊断模块，可详细的读取系统的诊断信息，帮助了解系统的当前状态，便于用户及时调整系统状态。

本节的 5.7.1 ~5.7.10 小节主要介绍 CPU 诊断的内容，存放于 HS_Diagnosis.Lib 库中的 CPU_Diag 文件夹内；5.7.11 小节介绍本地总线模块诊断，存放于 LocalBus Diag 文件夹内；5.7.12~5.7.14 小节主要介绍 DP 诊断，存放于 DP Diag 文件夹内；如下图所示：

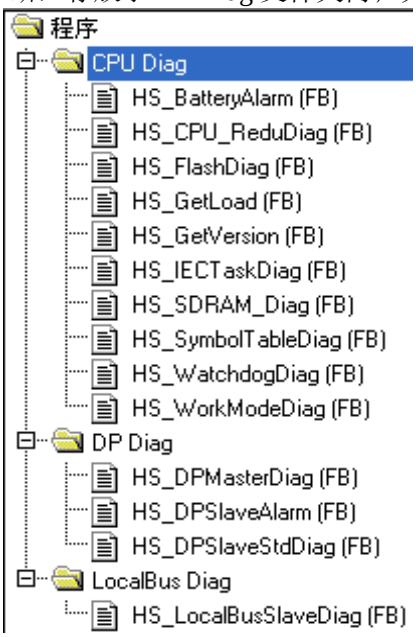
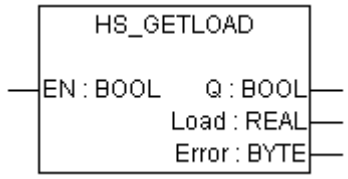


图 5-7-1 诊断指令库

5.7.1 HS_GetLoad——CPU 负荷诊断指令

- **功能：**读取 CPU 负荷当前值。
- **指令示例：**

梯形图语言示例


- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值

EN	BOOL	使能	0：无效 1：有效	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	读取完成	0：未完成 1：已完成	
Load	REAL	读取负荷	CPU 负荷率，单位%	
Error	BYTE	读取负荷错误	0： 取负荷量成功 1： 读取负荷量不成功	

✧ **指令使用举例**

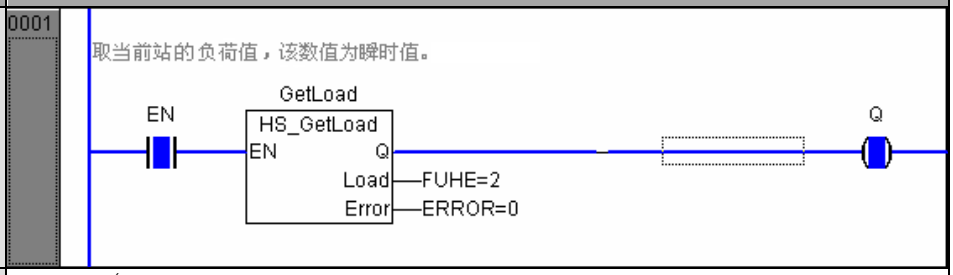
变量定义

VAR

	名称	地址	类型	初始值	注释
0001	EN		BOOL		使能
0002	GetLoad		HS_GetLoad		诊断CPU负荷功能块
0003	FUHE		REAL		CPU负荷实时值
0004	ERROR		BYTE		故障报警标识
0005	Q		BOOL		完成标识

编程语言 | **程 序**

梯形图（LD）



结构化文本（ST）

```
GetLoad (EN:=EN,
    Q=>Q, Load=>FUHE, Error=>ERROR);
```

注意：

- 调用该功能块的任务的任务周期应大于或等于 1 秒，如下图所示，否则读取的负荷值将保持不变。

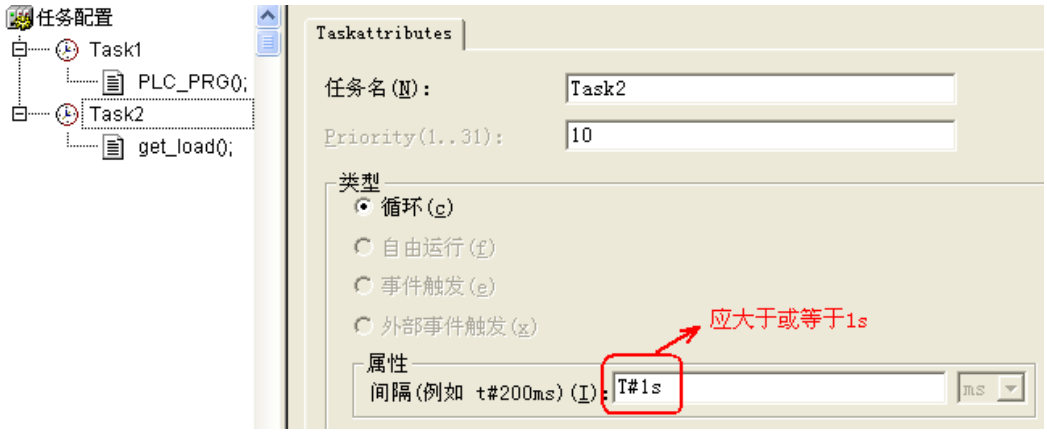
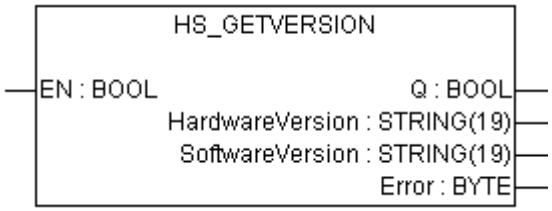


图 5-7-2 任务配置图

5.7.2 HS_GetVersion——版本号诊断指令

- **功能**：诊断硬件和软件版本号。
- **指令示例**：

梯形图语言示例



- **参数说明**：

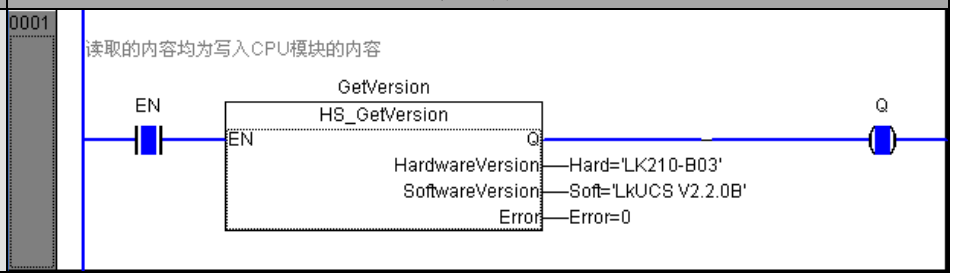
输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：有效	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	是否读取完成	0：未完成 1：已完成	
HardwareVersion	STRING	读取硬件版本号		
SoftwareVersion	STRING	读取软件版本号		
Error	BYTE	错误信息	0：读取版本型号成功 非 0，表示读取版本型号不成功 .0=1：表示读取硬件版本号不成功 .1=1：表示读取软件版本号不成功	

- **指令使用举例**

变量定义

VAR

	名称	地址	类型	初始值	注释
0001	EN		BOOL		使能
0002	GetVersion		HS_GetVersion		诊断版本号功能块
0003	Hard		STRING		硬件类型号
0004	Soft		STRING		软件版本号
0005	Error		BYTE		错误
0006	Q		BOOL		读取完成标识

编程语言	程　序
梯形图（LD）	

结构化文本（ST）	GetVersion (EN:=EN, Q=>Q, HardwareVersion=>Hard, SoftwareVersion=>Soft, Error=>Error);

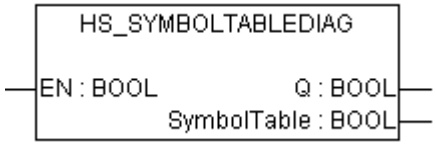
程序说明：

- 当使能 EN 置位时，可以读取硬件类型号和软件版本号；
- 其中软件版本号为系统内部的版本号；
- 读取完成，标志 Q 置 1，读取时出现错误置位 Error 相应标志位。

5.7.3 HS_SymbolTableDiag——符号表诊断指令

- **功能**：诊断符号表是否正确。当使能 EN 置位时，可以诊断符号表是否正确，判断完成 Q 置 1。
- **指令示例**：

梯形图语言示例



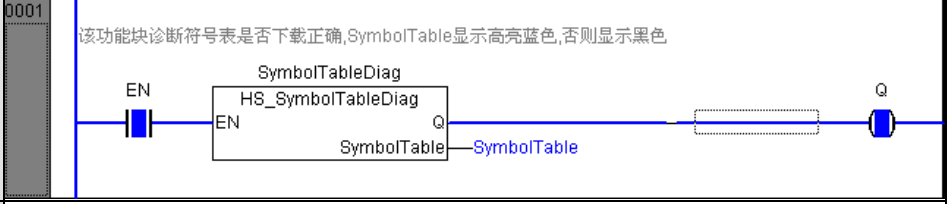
- **参数说明**：

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：有效	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	读取完成	0：未完成 1：已完成	
SymbolTable	BOOL	下装的符号表是否正确	0：不正确或未下装符号表 1：下装的符号表正确	

✧ **指令使用举例**

变量定义

	VAR 名称	地址	类型	初始值	注释
0001	EN		BOOL		使能
0002	SymbolTableDiag		HS_SymbolTableDi		诊断符号表功能块
0003	SymbolTable		BOOL		符号表是否正确
0004	Q		BOOL		诊断完成标识

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	SymbolTableDiag(EN:=EN, Q=>Q, SymbolTable=>SymbolTable);

程序说明：

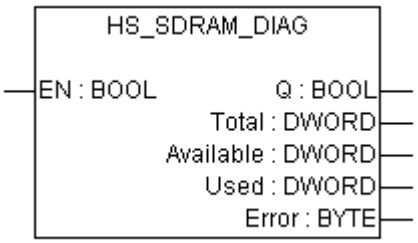
- 下载的符号表正确时，SymbolTable 项为 1（或 Ture）否则为 0（或 False）

> **提示：**
> - 当符号表下载不正确时，可能会导致下层和上位机不能正常通讯，（使用 Modbus 协议通讯的除外）。

5.7.4 HS_SDRAM_Diag——SDRAM 实际使用量诊断指令

- **功能**：读取 SDRAM 的实际使用情况。其中，SDRAM 是指控制器的内存空间，目前用户使用的空间都是固定分配好的，用户组态程序的大小对 SDRAM 的空间影响不大，并且目前 SDRAM 的剩余空间还比较大，所以用户可不必特别关心。此功能块主要起诊断错误的作用，当发现 SDRAM 剩余量越来越小，或者所剩无几时，说明有内存泄露等情况，有错误产生。
- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	是否读取完成	0：无效 1：有效	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	读取完成	0：未完成 1：已完成	
Total	DWORD	读取 SDRAM 的总量（KB）		
Used	DWORD	读取 SDRAM 的实际使用量(KB)		
Available	DWORD	读取 SDRAM 的空闲量（KB）		
Error	BYTE	读取 SDRAM 实际使用量错误	=0：读取 SDRAM 成功 =1：读取 SDRAM 不成功	

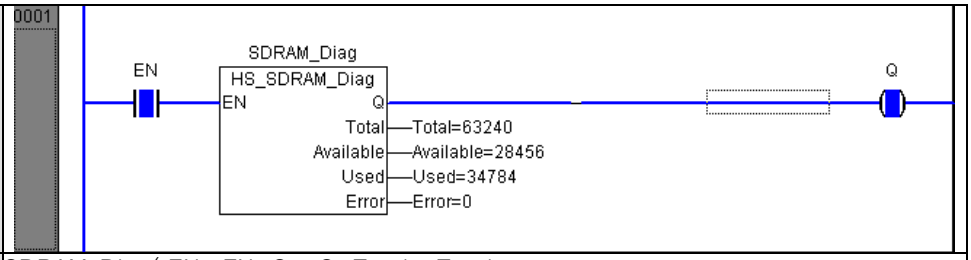
- **指令使用举例**

变量定义

VAR

	名称	地址	类型	初始值	注释
0001	SDRAM_Diag		HS_SDRAM_Diag		诊断SDRAM实际使用量功能块
0002	EN		BOOL		使能
0003	Q		BOOL		诊断完毕标识
0004	Total		DWORD		读取SDRAM的总量（KB）
0005	Used		DWORD		读取SDRAM的实际使用量（KB）
0006	Available		DWORD		读取SDRAM的空闲量（KB）
0007	Error		BYTE		读取错误

编程语言	程　序

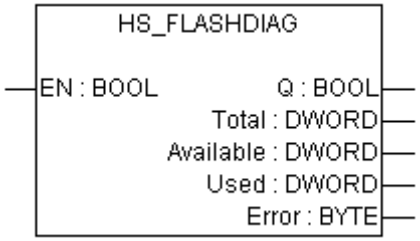
梯形图（LD）	
结构化文本（ST）	SDRAM_Diag(EN:=EN, Q=>Q, Total=>Total, Available=>Available, Used=>Used, Error=>Error);

程序说明：

- 程序使能后，功能块自动读取 SDRAM 的容量信息，读取完毕，标志 Q 置 1，如果读取错误则 Error 置 1 。

5.7.5 HS_FlashDiag——Flash 使用量诊断指令

- **功能**：用于读取当前 Flash 的使用情况。其中，Flash 指的是控制器的 Flash 存储器，用户程序和系统程序存储在 Flash 中，控制器上电后执行 Flash 中的系统程序，系统程序将用户程序调入 SDRAM（内存）中运行从而获得最快的运算性能。
- **指令示例：**

梯形图语言示例


- **参数说明：**

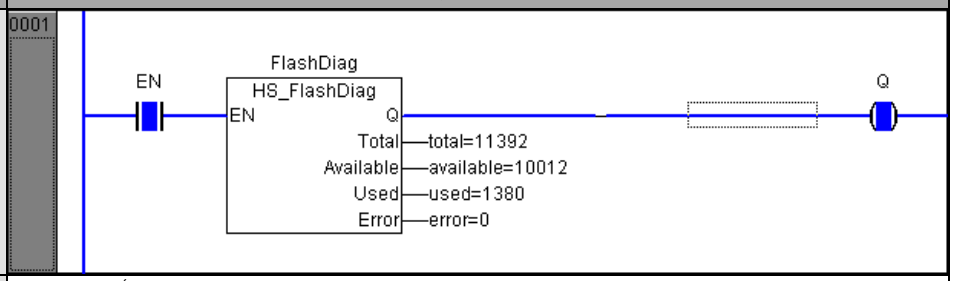
输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	是否读取完成	0：无效 1：有效	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	读取完成	0：未完成 1：已完成	
Total	DWORD	读取 flash 的总量（KB）		
Used	DWORD	读取 flash 的实际使用量（KB）		
Available	DWORD	读取 flash 的空闲量（KB）		
Error	BYTE	错误信息	=0：读取 flash 成功 =1：读取 flash 不成功	

- **指令使用举例**

变量定义

VAR

	名称	地址	类型	初始值	注释
0001	FlashDiag		HS_FlashDiag		诊断flash使用量功能块
0002	EN		BOOL		使能
0003	Q		BOOL		诊断完成标识
0004	Total		DWORD		读取flash总量（KB）
0005	Used		DWORD		读取flash使用量（KB）
0006	Available		DWORD		读取flash空闲量（KB）
0007	Error		BYTE		错误

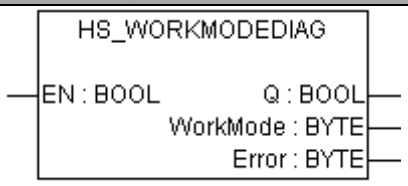
编程语言	程　序
梯形图（LD）	
结构化文本（ST）	FlashDiag(EN:=EN, Q=>Q, Total=>total, Available=>available, Used=>used, Error=>error);

程序说明：

- 当使能 EN 置位时，将读取 flash 的使用总量、空闲量和实际使用量，读取标志 Q 置 1，如果读取不成功则 Error 置位相应的标志位。

5.7.6 HS_WorkModeDiag——工作模式诊断指令

- **功能**：用于诊断 CPU 目前的工作模式。
- **指令示例：**

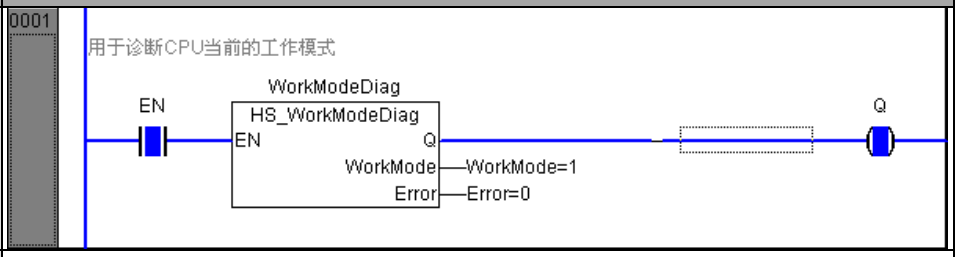
梯形图语言示例


- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	是否读取完成	0：无效 1：有效	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	读取完成	0：未完成 1：已完成	
WorkMode	BYTE	读取工作模式	工作模式（PRG 无法读取） =0：REM =1：RUN	
Error	BYTE	错误信息	=0：读取工作模式成功 =1：读取工作模式不成功	

✧ **指令使用举例**

变量定义					
VAR	名称	地址	类型	初始值	注释
0001	EN		BOOL		使能
0002	WorkModeDiag		HS_WorkModeDiag		工作模式诊断功能块
0003	Q		BOOL		诊断完成标识
0004	WorkMode		BYTE		工作模式
0005	Error		BYTE		错误

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	WorkModeDiag(EN:=EN, Q=>Q, WorkMode=>WorkMode, Error=>Error);

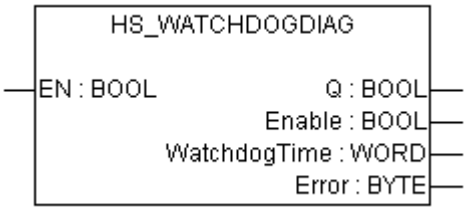
程序说明：

- 当使能 EN 置位时，实时读取用户工作模式，读取完成，标志 Q 置 1，读取错误 Error 置 1；
- CPU 处于 PRG 模式时，无法读取工作模式。

5.7.7 HS_ WatchdogDiag——看门狗诊断指令

- **功能**：用于诊断看门狗是否正常，正常时系统已对其进行了固有设置。
- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	是否读取完成	0：无效 1：有效	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	是否读取完成	0：未完成。 1：已完成。	
Enable	BOOL	看门狗是	0：未被设置	

		否被设置	1：已被设置	
WatchdogTime	DWORD	读取看门狗时间（ms）		
Error	BYTE	错误信息	0：读取看门狗成功 非 0，表示读取看门狗不成功情况，具体如下所示: .0=1：置位表示读取看门狗是否被设置不成功 .1=1：置位表示读取看门狗时间不成功	

✧ **指令使用举例**

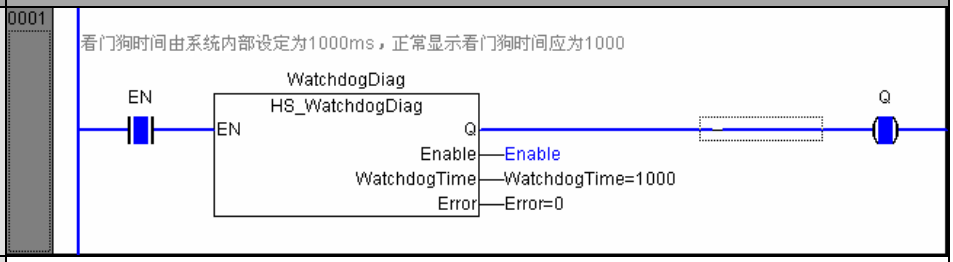
变量定义

VAR

	名称	地址	类型	初始值	注释
0001	WatchdogDiag		HS_WatchdogDiag		诊断"看门狗"功能块
0002	EN		BOOL		使能
0003	Enable		BOOL		看门狗是否被设置
0004	WatchdogTime		DWORD		读取看门狗时间（ms）
0005	Error		BYTE		错误
0006	Q		BOOL		诊断完成标识

编程语言 | **程　序**

梯形图（LD）



结构化文本（ST）

```
WatchdogDiag (EN:=EN, Q=>Q, Enable=>Enable,
    WatchdogTime=>WatchdogTime, Error=>Error);
```

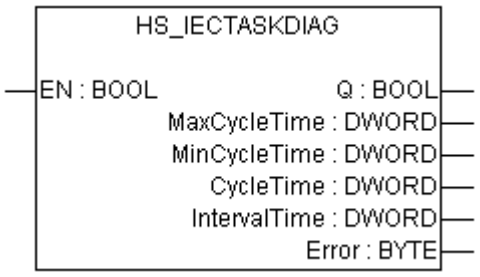
程序说明：

- 当使能 EN 置位时，读取看门狗是否被设置以及看门狗时间，读取标志 Q 置 1，读取错误时，Error 相应位会置位；
- CPU 正常运行时 WatchdogTime 数值不变，恒为 1000ms（该数值为系统内部固有设置）；

5.7.8 HS_IECTaskDiag——任务运行状况诊断指令

- **功能**：读取调用该功能块的任务的运行状况，包括最大执行时间、最小执行时间、本次实际执行时间和本次间隔时间。
- **指令示例：**

梯形图语言示例



➢ **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：有效	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	是否读取完成	0：未完成。 1：已完成。	
MaxCycleTime	DWORD	读取最大执行时间(ms)	初始化为 0；实测用户任务调度周期，保存历史周期中最大值；首周期不可用；精确到毫秒。	
MinCycleTime	DWORD	读取最小执行时间(ms)	初始化为(DWORD)(~0)；实测用户任务调度周期，保存历史周期中最小值；首周期不可用；精确到毫秒。	
CycleTime	DWORD	读取本次实际执行时间(ms)	实测用户任务执行所耗时间；精确到毫秒。	
IntervalTime	DWORD	读取本次间隔时间(ms)	实测用户任务调度间隔；首周期为用户在工程里配置的理论值；若无配置，则默认 10ms(LK205/207)或 50ms(LK209/210)；精确到毫秒。	
Error	BYTE	错误信息	0：获取正运行的任务成功 非 0，表示读取任务运行状况错误，具体 bit 位如下所示 .0=1：表示获取任务失败 .1=1：表示任务队列为空	

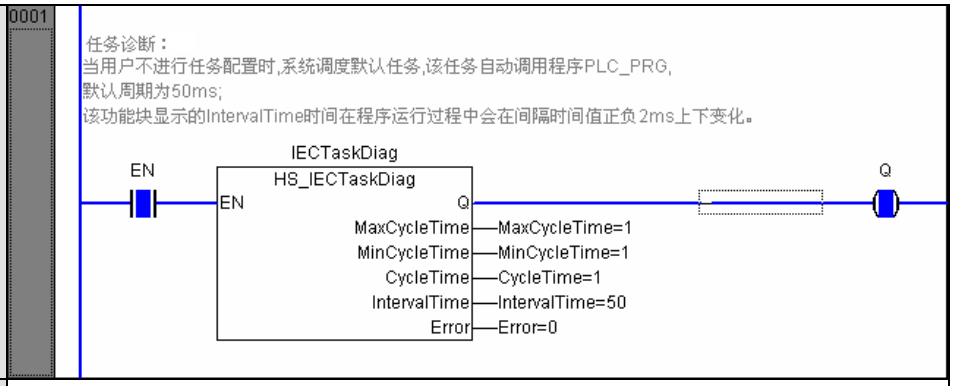
✧ **指令使用举例**

变量定义

VAR

	名称	地址	类型	初始值	注释
0001	IECTaskDiag		HS_IECTaskDiag		诊断任务运行状况功能块
0002	EN		BOOL		使能
0003	Q		BOOL		诊断完成标识
0004	MaxCycleTime		DWORD		读取最大执行时间(ms)
0005	MinCycleTime		DWORD		读取最小执行时间(ms)
0006	CycleTime		DWORD		读取本次实际执行时间（ms）
0007	IntervalTime		DWORD		读取本次间隔时间(ms)
0008	Error		BYTE		错误

编程语言	程　序

梯形图（LD）	
结构化文本（ST）	IECTaskDiag (EN:=EN, Q=>Q, MaxCycleTime=>MaxCycleTime, MinCycleTime=>MinCycleTime, CycleTime=>CycleTime, IntervalTime=>IntervalTime, Error=>Error);

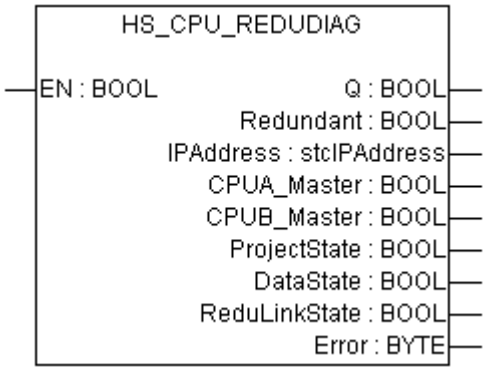
程序说明：

- 此功能块为高电平触发，当使能 EN 置位时，读取任务的运行状况，设置标志 Q 置 1；
- 显示的 IntervalTime 时间在程序运行过程中会在间隔时间正负 2ms 上下变化；
- 当无法读取信息时，置位 Error 相应的标志位。

5.7.9 HS_CPU_ReduDiag——冗余状态诊断指令

- **功能**：读取冗余系统的状态信息进行，并对其状态进行诊断。
- **指令示例：**

梯形图语言示例



- **参数说明：**

（注：A 机为插在冗余背板右边 CPU 插槽的控制器，B 机则为左边 CPU 插槽的控制器）

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：有效	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	是否读取完成	0：未完成。 1：已完成。	
Redundant	BOOL	判断是否为冗余系统	0：为单机 1：为冗余	
IPAddress	stcIPAddress	读取 IP 地址	具体内容参见表 5-7-1	
CPUA_Master	BOOL	A 机是否为主机	1：为主机 0：为从机	
CPUB_Master	BOOL	B 机是否为主机	1：为主机 0：为从机	
ProjectState	BOOL	工程状态	0：新工程（只有主机有工程从机没有工程）	

			1：两边都具有工程	
DataState	BOOL	数据状态	0：主从数据不同步 1：主从数据同步	
ReduLinkState	BOOL	冗余链路状态	0：冗余链路未通 1：冗余链路已通	
Error	BYTE	读取错误	=0：读取冗余状况成功 非 0，表示读取冗余运行状况时不正常情况，具体如下所示 .0=1：单机状况下为从机状态 .1=1：读取 ETHERNET2 主从 IP 地址不匹配 .2=1：读取 ETHERNET1 主从 IP 地址不匹配 .3=1：有两个主机状态 .4=1：有两个从机状态	

表 5-7-1　IP 地址数据结构 stcIPAdress 中具体内容：

成员	数据类型	功能描述
CPUA_ETHERNET2_IP	STRING	A 机 ETHERNET2 网的 IP 地址
CPUA_ETHERNET1_IP	STRING	A 机 ETHERNET1 网的 IP 地址
CPUB_ETHERNET2_IP	STRING	B 机 ETHERNET2 网的 IP 地址
CPUB_ETHERNET1_IP	STRING	B 机 ETHERNET1 网的 IP 地址

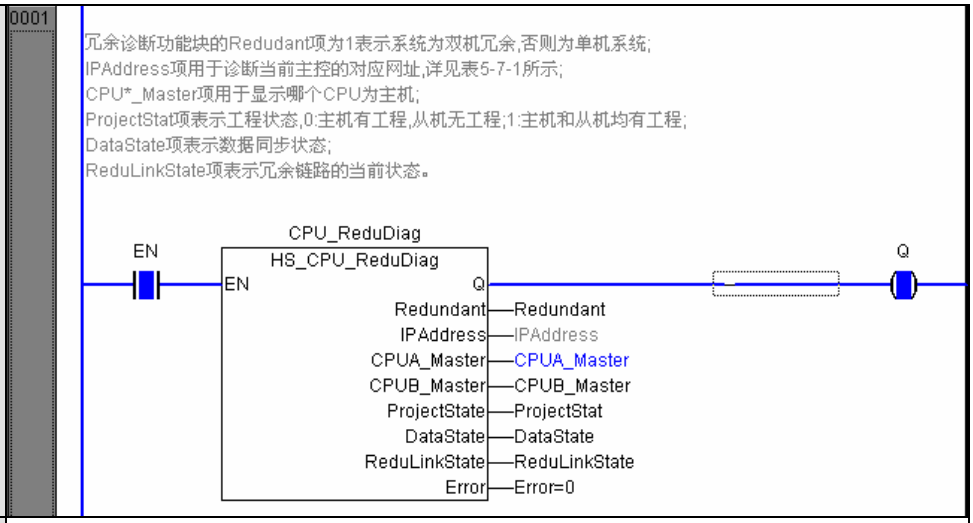
✧ **指令使用举例**

变量定义

VAR

	名称	地址	类型	初始值	注释
0001	CPU_ReduDiag		HS_CPU_ReduDiag		诊断冗余运行状态指令
0002	EN		BOOL		使能
0003	Q		BOOL		诊断完成标识
0004	Redundant		BOOL		判断是否为冗余
0005	IPAddress		stcIPAddress		读取IP地址
0006	CPUA_Master		BOOL		A机是否为主机
0007	CPUB_Master		BOOL		B机是否为主机
0008	ProjectStat		BOOL		工程状态
0009	DataState		BOOL		数据状态
0010	ReduLinkState		BOOL		冗余链路状态
0011	Error		BYTE		错误

编程语言 | **程　序**

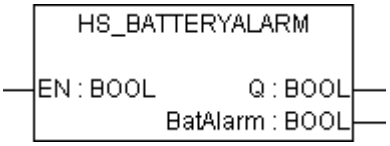
梯形图（LD）	
结构化文本（ST）	CPU_ReduDiag (EN:=EN, Q=>Q, Redundant=>Redundant, IPAddress=>IPAddress, CPUA_Master=>CPUA_Master, CPUB_Master=>CPUB_Master, ProjectState=>ProjectState, DataState=>DateState, ReduLinkState=> ReduLinkState, Error=>Error);

程序说明：

- 此功能块为高电平触发，当使能 EN 置位时，读取运行时的信息，包括判断单双机、IP 地址、主从信息、工程状况、数据状况、冗余链路状态，读取完毕标志 Q 置 1，读取错误时置位 Error 相应的标志位；

5.7.10 HS_BatteryAlarm——电池电量状况诊断指令

- **功能**：诊断 CPU 电池电量是否充足，不充足则发出警告。
- **功能块特征**：此功能块为高电平触发，当使能 EN 置位时，读取 CPU 电池电量状况，设置标志 Q 置 1。若电量充足，BatAlarm 端为 0；若电池电量低于额定值的 90%或者电池没有安装，则 BatAlarm 端为 1 作为向用户发出警告，这时用户需立即更换电池。
- **指令示例：**

梯形图语言示例


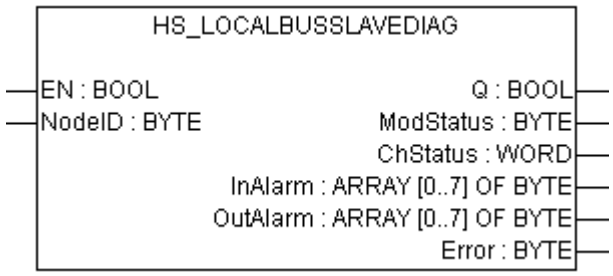
- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：有效	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	读取完成	0：未完成 1：已完成	0
BatAlarm	BOOL	电池是否需要更换	0：电池电量充足 1：没电池或者电池电量不足，需更换电池	0

5.7.11 HS_LocalBusSlaveDiag——本地总线诊断指令

- **功能**：对本地总线模块进行诊断。
- **指令示例**：

梯形图语言示例



- **参数说明**：

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：有效	0
NodeID	BYTE	模块节点号	2～15	2
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	获得诊断信息完成	0：未完成 1：完成	
Error	BYTE	错误信息	=0：无错误 =1：模块地址不正确 =2：该地址未配置模块 =3：该模块未上电 =4：该模块配置不正确	
ModStatus	BYTE	状态信息	.0=1：通道信息无效（预留） .1=1：模块内部故障（预留） .2=1：通道错误（预留） .3=1：无现场电源 .4=1：模块参数错误（预留） .7 .6 .5 = 001，开入模块 .7 .6 .5= 010，开出模块 .7 .6 .5= 100，模拟量模块 .7 .6 .5 = 011，通讯模块	
ChStatus	WORD	通道状态	=0：正常 .0=1，通道 0 有错 .1=1，通道 1 有错 …… .15=1，通道 15 有错	
InAlarm[8]	BYTE ARRAY	输入通道数据报警	输入通道 1-8 诊断数据：（Lk850 模块有效） =0：正常 .0=1，输入超上限报警 .1=1，输入超下限报警 .2=1，输入过量程报警 .3=1，输入欠量程报警 .4=1，断线报警 bit 5~7，预留	
OutAlarm[8]	BYTE ARRAY	输出通道数据报警	输出通道 1-8 诊断数据：（Lk850 模块有效） =0：正常 .0=1，输出过载 bit 1~7，预留	

- ✧ **指令使用举例**

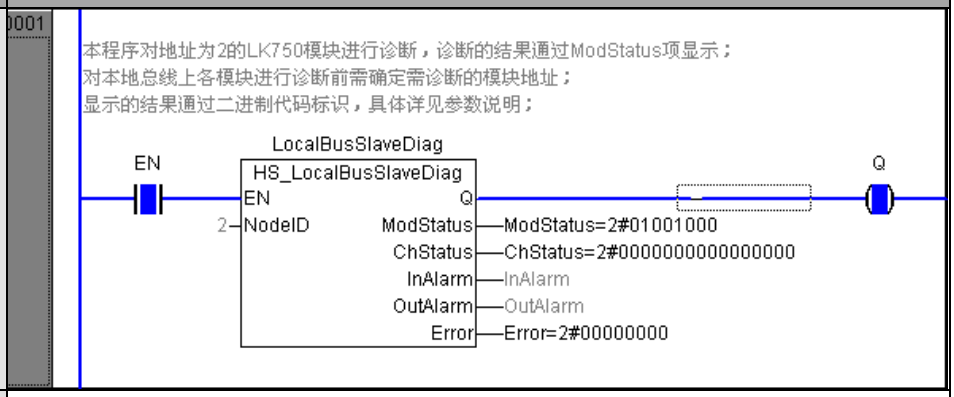
变量定义

VAR

	名称	地址	类型	初始值	注释
0001	EN		BOOL		使能
0002	LocalBusSlaveDiag		HS_LocalBusSlavel		本地总线诊断功能块
0003	Q		BOOL		诊断完成标识
0004	ModStatus		BYTE		状态信息
0005	ChStatus		WORD		通道状态
0006	InAlarm		ARRAY [0..7] OF BYT		输入通道数据报警
0007	OutAlarm		ARRAY [0..7] OF BYT		输出通道数据报警
0008	Error		BYTE		错误

编程语言 | **程　序**

梯形图（LD）



结构化文本（ST）

```
LocalBusSlaveDiag (EN:=EN, NodeID:=2, Q=>Q,
    ModStatus=>ModStatus, ChStatus=> ChStatus,
    InAlarm=>InAlarm, OutAlarm=>OutAlarm, Error=>Error);
```

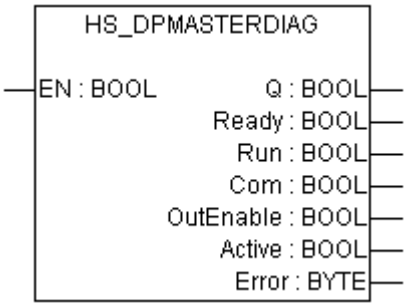
程序说明：

- 选择需诊断的模块，并将该模块所在槽位的地址添入 NodeID 项，从而对该模块进行诊断；
- ChStatus 项仅用于诊断 LK850 模块的通道状况；

5.7.12 HS_DPMasterDiag——DP 主站诊断指令

- **功能**：对 DP 主站（ LK CPU 里内置）括对主站的初始化是否完成，主站是否完成接收下载数据，以及是否与从站建立正常的数据交换等状态。
- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：有效	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	获得诊断信息完成	0：未完成 1：完成	
Ready	BOOL	是否已准备好	0：主站初始化未完成 1：主站初始化完成	
Run	BOOL	是否已运行	0：主站未接收完下装数据 1：主站接收下装数据完成，开始与从站通讯	
Com	BOOL	是否与从站建立数据交换	0：还未与从站建立数据交换 1：至少与一个从站建立了数据交换	
OutEnable	BOOL	输出使能	0：输出不使能 1：输出使能	
Active	BOOL	活动状态	0：热备 DP 主站 1：活动 DP 主站	
Error	BYTE	错误信息	=0：无错误信息 =1：没有检测到 DP 主站	

✧ **指令使用举例**

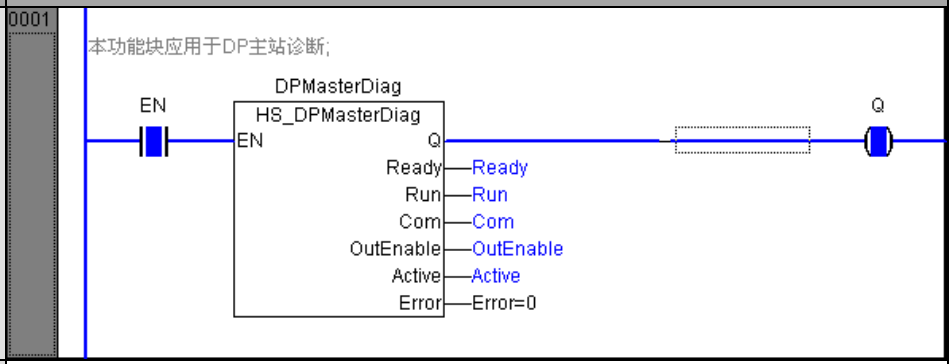
变量定义

VAR

	名称	地址	类型	初始值	注释
0001	DPMasterDiag		HS_DPMasterDiag		DP主站诊断功能块
0002	EN		BOOL		使能
0003	Q		BOOL		诊断完成标识
0004	Ready		BOOL		是否已准备好
0005	Run		BOOL		是否已运行
0006	Com		BOOL		是否与从站建立数据交换
0007	OutEnable		BOOL		输出使能
0008	Active		BOOL		活动状态
0009	Error		BYTE		错误信息

编程语言 | **程　序**

梯形图（LD）



结构化文本（ST）

```
DPMasterDiag (EN:=EN, Q=>Q, Ready=>Ready, Run=> Run,
    Com=> Com, OutEnable=> OutEnable, Active=>ctive, Error=> Error);
```

程序说明：

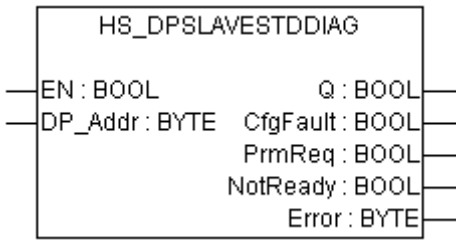
- 该功能块采用高电平触发方式，使能后高电平触发诊断。

- Ready项标识DP主站是否初始化完成，Run项标识主站是否接收下载数据完毕并开始与从站进行通讯，Com项标识主从站是否通讯正常，OutEnable项标识输出使能，Active项则标识主站的活动状态。

5.7.13 HS_DPSlaveStdDiag——DP从站标准诊断指令

- **功能**：诊断从站模块与主站的通讯状态，主要用来获取未与主站建立通讯的从站的诊断信息，辅助解决数据通讯不正常的问题。
- **注意**：该功能块的主要用途是内部开发调试，用户使用时可能会引起疑惑或者误解，所以**不推荐**用户使用该功能块。DP从站诊断目前推荐用户使用HS_DPSlaveAlarm功能块。
- **指令示例**：

梯形图语言示例



- **参数说明**：

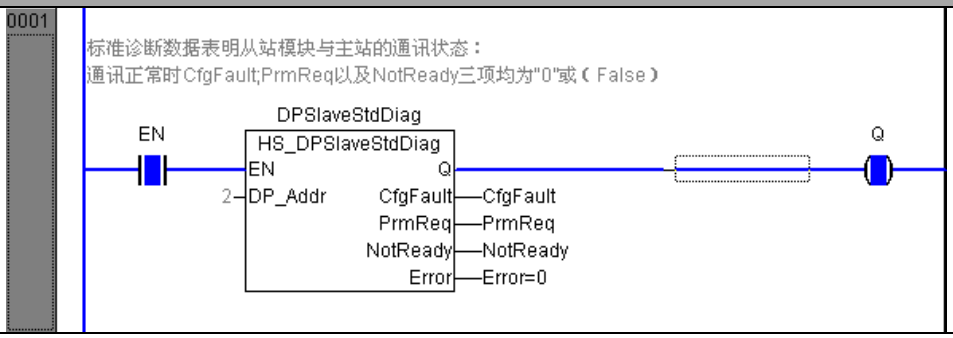
输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：有效	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	获得诊断信息完成	0：未完成 1：完成	
CfgFault	BOOL	配置数据是否错误	0：配置数据正确 1：配置数据错误	
PrmReq	BOOL	是否重新请求参数数据	0：参数数据正确 1：参数数据有误，请求主站重新参数化	
NotReady	BOOL	数据是否准备好	0：从站数据已经准备好 1：从站数据还未准备好	
Error	BYTE	错误信息	=0：无错误信息 =1：模块地址不正确 =2：该设备未配置 =3：无诊断数据	

- **指令使用举例**

变量定义

VAR

	名称	地址	类型	初始值	注释
0001	EN		BOOL		使能
0002	DPSlaveStdDiag		HS_DPSlaveStdDia		DP从站标准诊断功能块
0003	Q		BOOL		诊断完成标识
0004	CfgFault		BOOL		配置数据是否错误
0005	PrmReq		BOOL		是否重新请求参数数据
0006	NotReady		BOOL		数据是否准备好
0007	Error		BYTE		错误

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	DPSlaveStdDiag (EN:=EN, Q=>Q, CfgFault=> CfgFault, PrmReq=> PrmReq, NotReady=> NotReady, Error=>Error);

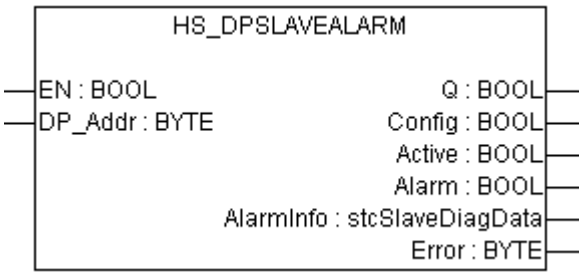
程序说明：

- 从站数据通讯正常的情况下可不调用该功能块；
- 程序中，输入项 DP_Addr 要求添入需诊断的模块地址；

5.7.14 HS_DPSlaveAlarm——DP 从站用户扩展诊断指令

- **功能**：对 DP 从站模块的进行状态诊断。**推荐**用户使用该功能块来进行 DP 从站诊断，不推荐使用 HS_DPSlaveStdDiag 功能块。
- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：有效	0
DP_Addr	BYTE	站地址	需要诊断的从站地址	2
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	获得诊断信	0：未完成	

		息完成	1：完成	
Config	BOOL	是否配置	0：未配置 1：配置	
Active	BOOL	是否为活动站	0：从站离线 1：从站在线，数据交换正常	
Alarm	BOOL	是否有用户扩展诊断	0：无扩展诊断数据 1：有扩展诊断数据	
AlarmInfo	stcSlaveDiagData	诊断信息	该结构中输出诊断信息	
Error	BYTE	错误信息	=0：无错误信息 =1：模块地址不正确 =2：该设备未配置	

用户定义的诊断信息数据结构 stcSlaveDiagData 中各成员说明:

成员	数据类型	功能描述	参数说明	默认值
DevDiag	stcGeneralDiag	设备诊断数据	从站设备的设备相关诊断数据包括：设备诊断数据的长度及数据	
IdentDiag	stcGeneralDiag	识别号（模块）诊断数据*	从站识别号诊断数据包括：识别号诊断数据的长度及数据	
ChDiag	stcChDiag	通道诊断数据	通道相关诊断数据包括：当前发生诊断的通道的个数及诊断数据	

通用诊断数据结构 stcGeneralDiag 中，详细内容如下。

成员	数据类型	功能描述	参数说明	默认值
Length	BYTE	诊断数据长度	设备诊断信息的字节数	
Data[10]	BYTE ARRAY	诊断信息	设备诊断信息	

其中设备相关诊断数据区中的数据含义如下（目前设备相关诊断域只有 1 字节）

位	含义	值
Bit3 ~ Bit1	板级故障	0 —— 板级正常 1 —— 保留 2 —— 开关量现场掉电 其他 —— 保留
Bit 0	通道故障	0 —— 通道正常 1 —— 通道故障

识别号*（模块）诊断相关说明如下：

- 对于紧凑型设备（compact device），一个设备只有一个可组态模块，识别号为 0，如 LK610 模块；
- 对于模块型设备（modular device），一个设备可以有多个可组态模块，每一个模块有一个识别号(Module_reference)与之对应，如 LK511 模块；

常用设备与识别号的对应关系如下：

所属设备类型	设备	模块	识别号
紧凑型设备	LK410	8 Channels AI	0
	LK411	8 Channels AI	0
	LK412	6 Channels AI	0
	LK414	8 Channels AI	0
	LK415	6 Channels AI	0
	LK430	6 Channels RTD	0
	LK440	8 Channels AI(TC)	0
	LK441	8 Channels AI(TC)	0
	LK510	4 Channels AO	0

	LK610	16 Channels DI	0
	LK611	16 Channels DI	0
	LK612	16 Channels DI	0
	LK613	16 Channels DI	0
	LK614	16 Channels DI	0
	LK615	16 Channels DI	0
	LK630	15/16 Channels SOE	0
	LK711	8 Channels DO	0
	LK712	8 Channels DO	0
	LK720	8 Channels RELAY DO	0
	LK721	8 Channels RELAY DO	0
模块型设备	LK511	4 Channels AO	0
		Read Back	1
	LK620	2 Channels Counter	0
		4 Channels DO	1
	LK710	16 Channels DO	0
		DO Read Back	1
	LK810	4 Channels AI	0
		2 Channels AO	1

例如诊断域的第一字节 IdentDiag .Data[1]=0x02 则表示识别号为 1 的模块有诊断数据。

IdentDiag .Data[1]	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
	0	0	0	0	0	0	1	0

注：目前识别号诊断域只有 1 字节，其余字节为今后预留用。

通道诊断数据结构 stcChDiag 中，详细内容如下。

成员	数据类型	功能描述	参数说明	默认值
ChNum	BYTE	通道个数	诊断数据中包含的通道个数	
Module[40]	stcModData 结构数组	模块中通道诊断数据	记录每个模块中各个通道的通道诊断数据，每个通道没有新的诊断数据时一直保持旧诊断数据	

stcModData 的结构

成员	数据类型	功能描述	参数说明	默认值
ModuleNo	BYTE	模块号		
Channel[16]	stcEveryChDiag 结构数组	模块通道号	诊断信息发生变化的模块通道号	

stcEveryChDiag 的数据结构：

成员	数据类型	功能描述	参数说明	默认值
ChNo	BYTE	模块通道号	诊断信息发生变化的模块通道号	
Error	BYTE	信息类型	=0：通道故障恢复 =1：短路 =2：欠量程 =3：过量程 =4：过载 =5：温度超高 =6：断线 =7：超上限 =8：超下限 =9：故障 =10～15：保留 =16：开关触点故障 =17：通道现场掉电 =18：通道输出故障	

输出各项数据结构如下：

Q	

Config				
Active				
Alarm				
AlarmInfo	DevDiag	Length		
		Data[1]		
		……		
		Data[10]		
	IdentDiag	Length		
		Data[1]		
		……		
		Data[10]		
	ChDiag	ChNum		
		Module[1]	ModNo	
			Channel[1]	ChNo
				Error
			……	
			Channel[16]	Ch_No
				Error
		……		
		Module[30]		

✧ **指令使用举例**

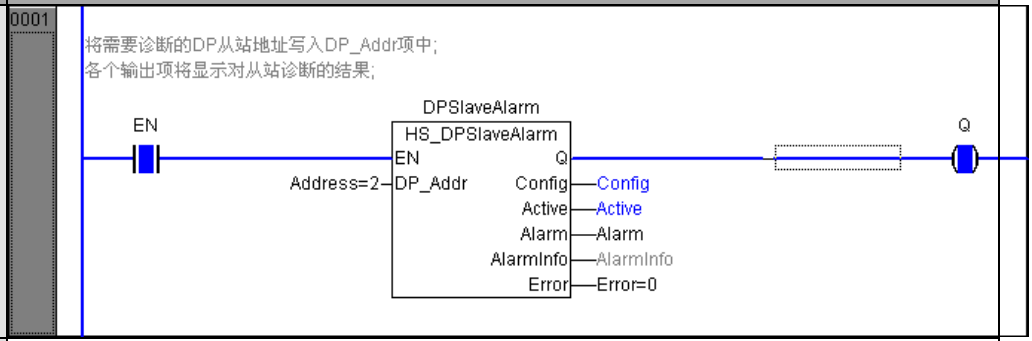
变量定义

VAR

	名称	地址	类型	初始值	注释
0001	DPSlaveAlarm		HS_DPSlaveAlarm		DP从站用户扩展诊断功能块
0002	Address		BYTE		从站地址
0003	EN		BOOL		使能
0004	Q		BOOL		诊断完成标识
0005	Config		BOOL		是否配置
0006	Active		BOOL		是否为活动站
0007	Alarm		BOOL		是否有用户扩展诊断
0008	AlarmInfo		stcSlaveDiagData		诊断信息
0009	Error		BYTE		错误

编程语言 | **程　序**

梯形图（LD）



结构化文本（ST）

```
DPSlaveAlarm (EN:=EN, DP_Addr:=Address, Q=>Q, Config=>Config,
    Active=>Active, Alarm=>Alarm, AlarmInfo=>AlarmInfo, Error=>Error);
```

程序说明：

- 从站硬件用户扩展诊断信息分为三个部分：设备相关诊断区、识别号（模块）相关诊断信息区和通道相关诊断信息区；

- 这些诊断数据以块结构存在，每个块结构中包含 1 个“头(Head)”字节，用以指明不同的诊断信息。设备相关诊断区、识别号（模块）相关诊断信息区在外部扩展诊断数据中各有一个，分别用于记录设备的总体诊断信息和有诊断信息的模块；
- 通道相关诊断信息区记录有诊断信息的通道的信息。
- AlarmInfo 项为 stcSlaveDiagData 型数据结构，具体各个诊断说明都对应于该项的各个子项，分别对 DP 从站的各个通道进行状态诊断具体如图 5.7.1 所示。
- Alarm 的状态根据诊断结果的不同而变化，包括发生故障，也包括恢复正常，以上情况都属于状态变化，只要有变化，Alarm 就跳变一次，具体情况在 Alarm_info 中反应。Alarm_info 保持最近一次的诊断信息的状态。
- PowerPro 系统运行时 AlarmInfo 内各项的显示情况如下(变量声明区显示)：

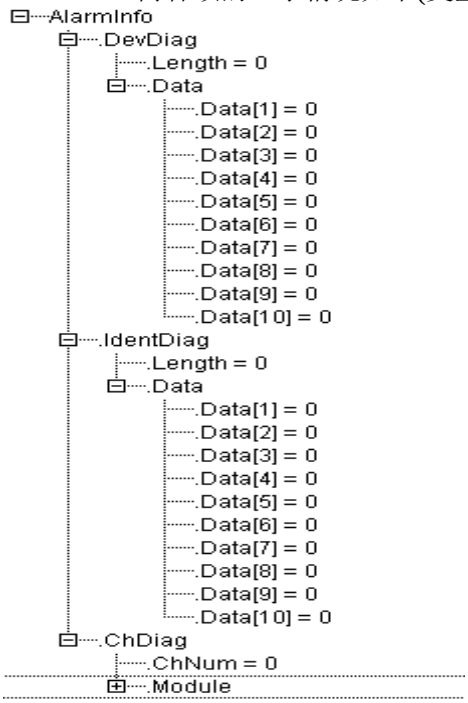


图 5-7-3 诊断信息(变量声明区显示)

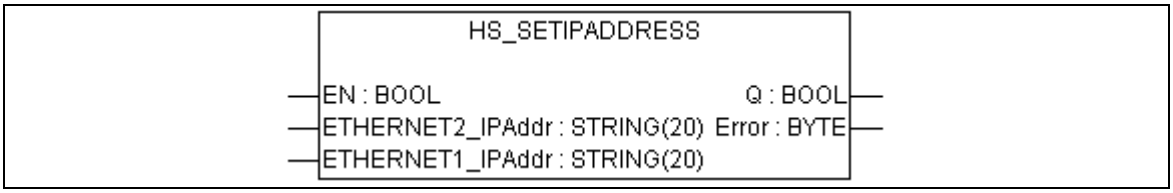
5.8 修改 IP 地址功能库 HS_SetIPAddress.Lib

- **注意：**修改 IP 地址功能库是 PowerPro V4.3.0B 版本软件中增加的指令库，目前只包含了一个功能块即修改 IP 地址指令的功能块。该库适用于 LK202-A01, LK205-A01, LK207-A03, LK209-A01, LK210-B05 以及更高的主控型号。如果用户的主控型号低于以上几类，建议用户**不要调用**此库，否则工程将不能下装。

5.8.1 HS_SetIPAddress——修改 IP 地址指令

- **功能**：对四段制 IP 地址的前三位地址进行修改，可对单 CPU 系统、双机冗余 CPU 系统的 EHTERNET2, ETHERNET1 的 IP 地址参数进行前三位字段地址的设定。
- **指令示例**：

梯形图语言示例



➢ **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：上升沿有效	0
ETHERNET2_IPAddr	String	ETHERNET2 网的 IP 地址	IP 地址前三段，如 ’128.0.0’ 注意：该变量为字符串变量，其头尾需加单引号。	‘128.0.0’
ETHERNET1_IPAddr	String	ETHERNET1 网的 IP 地址	IP 地址前三段，如’129.0.0’ 注意：该变量为字符串变量，其头尾需加单引号。	‘129.0.0’
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	设置完成	0：未完成 1：已完成	0
Error	BYTE	错误信息	=0：正确 不为 0：有错 =1，EHTERNET2，ETHERNET1 的 IP 地址前三段相同或和已有 IP 冲突 =2，取站号地址失败或站号不合法 =3，EHTERNET2 或 ETHERNET1IP 地址不合法 =4，停止网络服务失败 =5，执行 IP 设置命令失败 =6，写 IP 配置文件失败 =7，读 IP 配置文件失败 =8，写 IP 配置文件不正确 =9，获取网络配置不成功 =10，IP 设置成功但和输入不一致 =11，重新启动网络服务失败 =255，仅作为提示，LK 与 FacView 通讯只支持 128，129 网段	0

➢ **功能说明：**

1，只能任意修改 IP 地址中的前面三段，最后一段由 CPU 拨码开关确定，只支持有限的 A,B,C 类地址，即第一段为 1~126 或 128~223（不考虑主机地址再次划分为子网的情况）；

2，主从机都需要分别设置 IP，且从机需要设置与主机相同的网段（从机地址最后一段由主站最后一段加 128），否则不能正常通讯；

3，可以恢复为默认 IP 地址，操作为：

将站号（即 CPU 上的拨码开关）设为预留站号 9，复位控制器或断电再上电，迅速将钥匙开关拨到 PRG 位置；

注：操作完请将站号（即 CPU 上的拨码开关）和钥匙开关设为正确位置。

4，修改 IP 地址后不支持与 FacView 的通讯；如果想继续支持与 FacView 的通讯，需要将两个网络地址分别设为 128.0.0 和 129.0.0，只设一个也不支持。

5，只需设置 IP 地址，子网掩码和广播地址将自动修改；

6，用户需建立单独工程只组该功能块，用于修改 IP 地址，不要组其他功能块；

7，IP 地址设置成功后立即生效，且重启后保持。如果重启后保持修改 IP 失败将恢复成默认 IP；

8，设置 IP 可能会影响正在进行的通讯，重新建立连接后一切通讯正常；

9，ETHERNET1 和 ETHERNET2 的 IP 冲突，不能修改；

10，允许用户只修改一个 IP，另一个输入为空，但不允许两个都空；

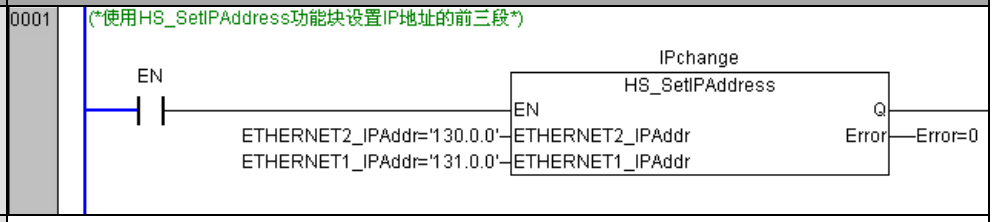
11，在修改过程中如果有错，将恢复成修改前的 IP，如果恢复不成功将恢复成默认 IP。

✧ 指令使用举例

变量定义

VAR

	名称	地址	类型	初始值	注释
0001	EN		BOOL		使能
0002	Q		BOOL		设置完成标识
0003	Error		BYTE		错误
0004	HS_SetIPAddress		HS_SetIPAddress		修改IP地址指令
0005	ETHERNET2_IPAddr		STRING		ETHERNET2的IP地址
0006	ETHERNET1_IPAddr		STRING		ETHERNET1的IP地址

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	IPchange (EN:=EN, ETHERNET2_IPAddr:= ETHERNET2_IPAddr, ETHERNET1_IPAddr:= ETHERNET1_IPAddr, Q=>Q, Error=>Error);

程序说明：

- 使能后，IP 地址进行了变更，由于当前登录的 IP 地址为原地址，因此系统提示通讯中断；
- 重新按照设定后的 IP 地址登录，用双机运行诊断模块可诊断出该控制器对应的网段地址。

5.9 PID 调节控制器指令库 HS_PIDController.Lib

5.9.1 HS_PID——PID 调节控制器功能块

- **功能**：对现场输入的过程值与设定值比较，对其进行 PID 调节控制，将调节的指令输出给现场设备从而达到及时响应的调节任务，同时对输出值进行限幅操作，以保证输出在许可的范围内操作，并对超限数值及时报警。
- **术语与缩略语**：

PID (Proportional-Integral-Differential):	比例-积分-微分
Manual / Auto:	手动/自动
Direct Action / Reverse Action:	正作用/反作用
Dead Band:	死区
EN (Enable):	使能
Ts (Sampling Time):	采样时间，运算周期
SP (Set Point):	设定点，设定值
PV (Process Variable):	过程变量，过程值
EV (Error Variable):	偏差变量，偏差值
MV (Manipulated Variable):	操纵变量，控制量，HS_PID 的输出

- **原始公式**：

HS_PID 功能模块的数学依据是采用不完全微分 PID 的传递函数公式，如下：

$$\frac{U(s)}{E(s)}=K_p\left(1+\frac{1}{T_i s}+\frac{T_d s}{1+T_d s}\right)=K_p\left[1+\frac{1}{T_i s}+(T_d s)\left(\frac{1}{1+T_d s}\right)\right] \quad (1)$$

其中：

$U(s)$：PID 运算的控制量

$E(s)$：偏差量

K_p：比例增益

T_i：积分时间

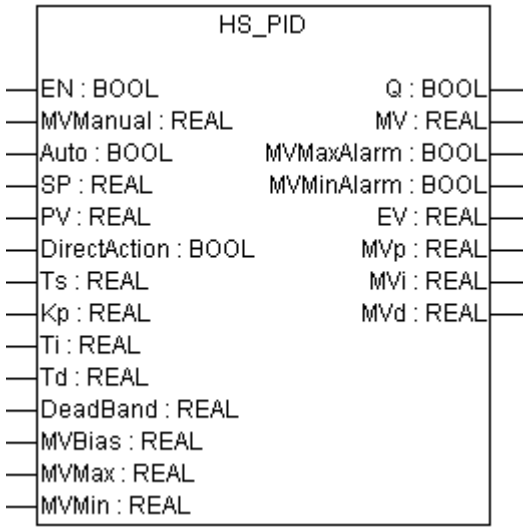
T_d：微分时间

不完全微分：在微分部分$(T_d s)$之后串联一个一阶惯性环节$\left(\frac{1}{T_d s+1}\right)$进行滤波处理，可以有效防止信号突变时导致微分项剧烈变化所引起的扰动。

注：为了不增加额外的参数，同时又保留一定的微分滤波处理功能，这里，微分滤波环节的惯性时间常数只单一地用微分时间T_d来代替。

- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	每个调度周期更新： FALSE：无效-模块调度周期保持低电平 TRUE：上升沿有效-模块调度周期保持高电平	FALSE
MVManual	REAL	手动输入值	Auto 为 FALSE 时，输出控制量 MV 取该值	0
Auto	BOOL	“自动”模式选择	FALSE：手动 TRUE：自动	FALSE
SP	REAL	设定值	目标设定值	0
PV	REAL	过程值	闭环反馈值	0
DerectAction	BOOL	“正作用”方式选择	FALSE：反作用（EV=SP-PV） TRUE：正作用（EV=PV-SP）	FALSE
Ts	REAL	运算周期（S）	Ts>0 并且可设置为用户任务调度周期的整数倍；	0.05

Kp	REAL	比例增益		1
Ti	REAL	积分时间（S）	Ti>=0,如果积分时间为 0，表示没有积分环节	1
Td	REAL	微分时间（S）	Td>=0,如果微分时间为 0，表示没有微分环节	0
DeadBand	REAL	偏差死区限	与偏差（正作用 PV-SP，反作用 SP-PV）比较；该值可衡量用户对被控对象的参数的精确度要求。如果设为 0，表示用户希望被控对象参数达到主控可识别的浮点精度。 1）DeadBand>=0 2）若 DeadBand=0，表示没有死区 3）偏差绝对值大于该值时有效，否则偏差为 0	0
MVBias	REAL	前馈控制量	与被控对象有关。用户工程人员可根据控制系统特性选用。	0.0
MVMax	REAL	输出值上限	限制输出 MV 上限；与所用执行器的量纲和范围有关	+100
MVMin	REAL	输出值下限	限制输出 MV 下限；与所用执行器的量纲和范围有关	-100
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	使能状态标志	每个调度周期根据输入使能 EN 更新： FALSE：无效-模块调度周期保持低电平 TRUE：有效-模块调度周期保持高电平	
MV	REAL	控制量输出	MV= MVp+ MVi +MVd	
MVMaxAlarm	BOOL	输出值超上限报警	FALSE：未超上限 TRUE：已超上限	
MVMinAlarm	BOOL	输出值超下限报警	FALSE：未超下限 TRUE：已超下限	
EV	REAL	设定值与过程值的偏差	偏差值（正作用 PV-SP，反作用 SP-PV） 若有死区，EV 则为经死区处理后的偏差值	
MVp	REAL	比例分量		
MVi	REAL	积分分量		
MVd	REAL	微分分量		

HS_PID 功能块的逻辑结构：

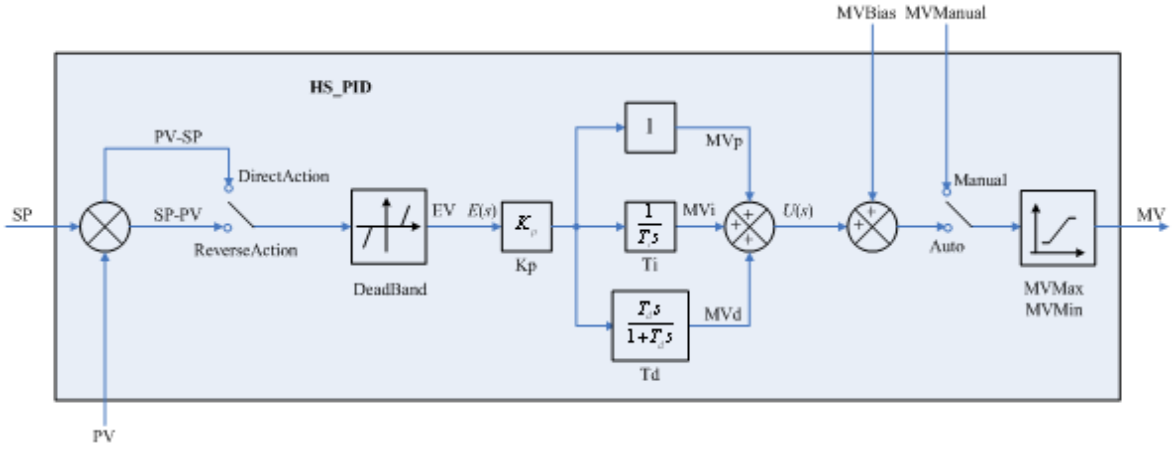


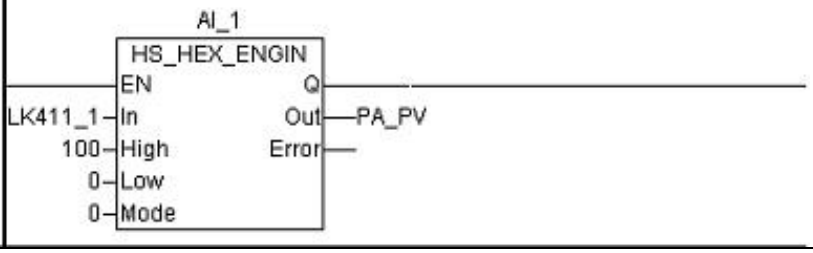
图 5.9.1-1PID 逻辑结构图

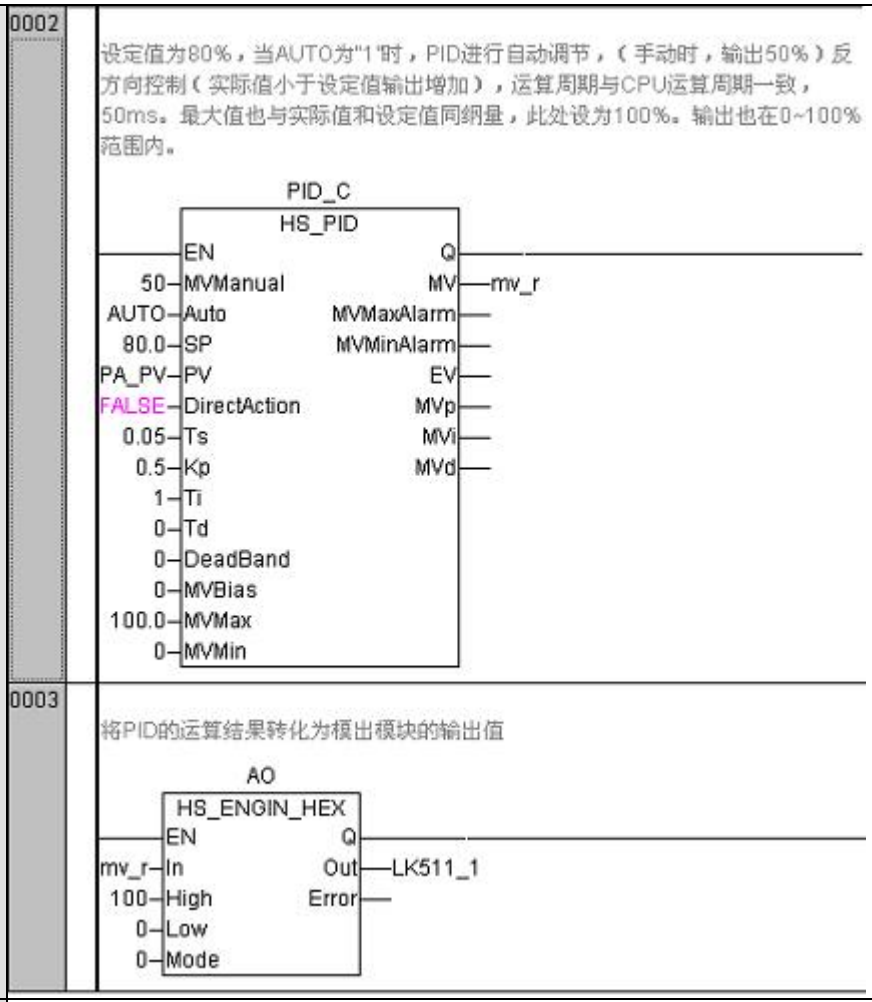
- **指令使用举例**

变量定义

VAR

	名称	地址	类型	初始值	注释
0001	AI_1		HS_HEX_ENGIN		
0002	LK411_1	%IW2	WORD		压力反馈值
0003	PA_PV		REAL		压力实际值%
0004	PID_C		HS_PID		
0005	mv_r		REAL		
0006	AO		HS_ENGIN_HEX		
0007	LK511_1	%QW0	WORD		
0008	AUTO		BOOL		

编程语言	程　序
梯形图（LD）	0001 将模入模块采集的数据转化为0~100%的工程量值，以作为与设定值同纲量的实际值 

<table>
<tr><td></td><td> 
</td></tr>
<tr><td>结构化文本（ST）</td><td>(*将模入模块采集的数据转化为 0~100%的工程量值*)
AI_1 (EN:=TRUE, In:=LK411_1, High:=100, Low:=0, Mode:=0,
Out=>PA_PV);
(*设定值为 80%，当 AUTO 为 1 时，PID 进行调节*)
PID_C (EN:= TRUE, MVManual:=50, Auto:=AUTO, SP:=80, PV:=PA_PV,
DirectAction:=False, Ts:=0.05, Kp:=0.5, Ti:=1, Td:=0, DeadBand:=0,
MVBias:=0, MVMax:=100.0, MVMin:=0, MV=>mv_r);
(*将 PID 的运算结果转化为模出模块的输出值*)
AO (EN:=1, In:=mv_r, High:=100, Low:=0, Mode:=0,
Out=>LK511_1);</td></tr>
</table>

功能说明：

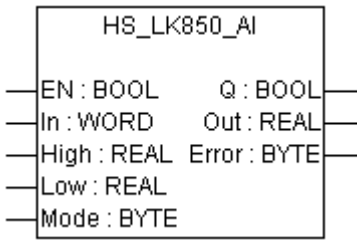
- 根据实际现场需要为 PID 控制器各项设置参数，并逐步调节各项参数达到控制目标；
- 提供微分滤波处理功能，即：在 PID 的微分部分串联一个一阶惯性环节，对微分运算结果实现低通滤波和平滑处理；
- 本功能块具有偏差死区功能；
- 本功能块具有积分饱和处理功能，即如果输出达到上限，则不再进行正向积分，只进行负向积分；如果输出达到下限，则不再进行负向积分，只进行正向积分；
- 本功能块具有输出限幅功能；
- 提供“手动/自动”无扰切换功能；
- 用于周期性的控制任务，允许多次调用，理论上无调用次数限制，用户可根据变量区使用情况、主控负荷情况和实际工程需求来决定调用次数；
- 允许用户设置采样周期大小（以用户任务调度周期为基数）。

5.10 LK850 量程转换指令库 HS_LK850_Convert.Lib

5.10.1 HS_LK850_AI——LK850 模拟输入量程转换功能块

- **功能**：提供 LK850 模块 AI 通道可选量程下对应数字量到工程量的转换功能。
- **指令示例：**

梯形图语言示例

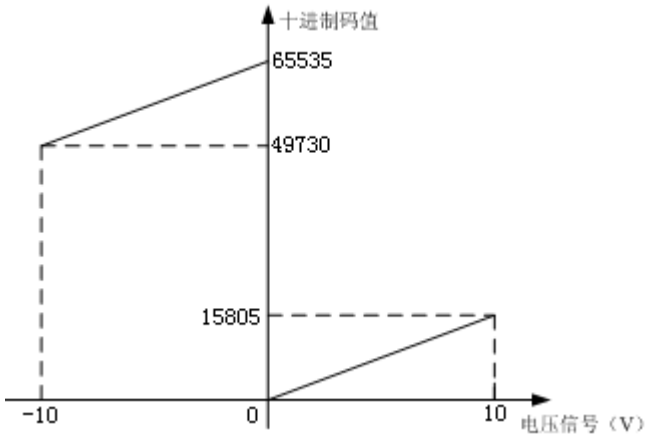


- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能输入	当 EN=FALSE 时，停止转换，Q=FALSE。 当 EN=TRUE 时，开始转换，Q=TRUE， 将十进制数对应于工程量的量程范围 Low-High。	FALSE
In	WORD	输入码值	0～65535	0
High	REAL	量程上限	初始值为 0	0
Low	REAL	量程下限	初始值为 0	0
Mode	BYTE	模式选择	=0：输入电流信号 4～20.58mA =1：输入电流信号 0～20.58mA =2：输入电压信号 0～+5.125V =3：输入电压信号 0～+10.25V =4：输入电压信号 -10.25～+10.25V	4
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	使能输出	当 EN=FALSE 时，停止转换，Q=FALSE。 当 EN=TRUE 时，开始转换，Q=TRUE， 将十进制数对应于工程量的量程范围 Low-High。	FALSE
Out	REAL	输出工程量值	各量程意义下的工程量值	0
Error	BYTE	错误类型	=0：当前模式量程转换无错 =1：量程下限超过上限 =2：模式输入错误 =3：输入码值超出当前模式范围	0

测量数据转换为工程量数据的转换模式说明：

转换模式	转换数据	量程下限	量程上限	输入信号类型
MODE=0	工程量	Low	High	电流信号：4~20mA
	十进制数	1523	7838	
	十六进制数	05F3	1E9E	
MODE=1	工程量	Low	High	电流信号：0~20mA
	十进制数	0	7838	
	十六进制数	0	1E9E	
MODE=2	工程量	Low	High	电压信号：0~5V
	十进制数	0	7903	
	十六进制数	0	1EDF	
MODE=3	工程量	Low	High	电压信号：0~10V
	十进制数	0	15805	

	十六进制数	0	3DBD	
MODE=4	工程量	Low	High	电压信号：-10~10V
	十进制数	49730	15805	
	十六进制数	C242	3DBD	
	具体转换关系如下图所示：  			

✧ **指令使用举例**

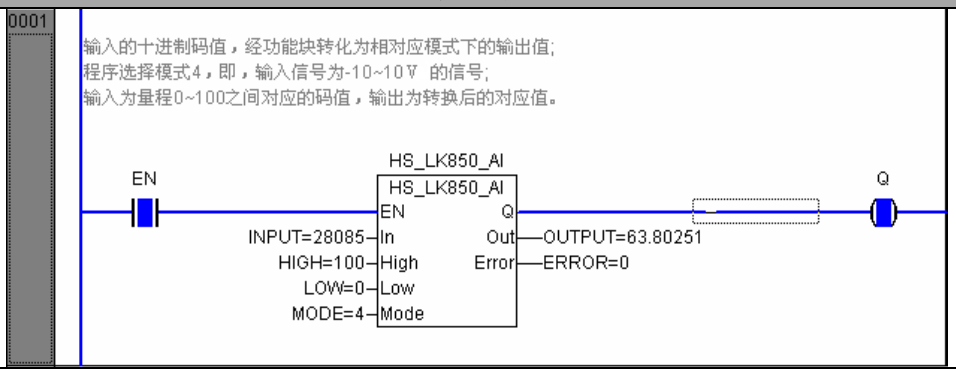
变量定义

VAR

	名称	地址	类型	初始值	注释
0001	HS_LK850_AI		HS_LK850_AI		LK850模块输入转换功能块
0002	INPUT		WORD		输入码值
0003	Q		BOOL		转换完成标识
0004	HIGH		REAL		工程量量程上限
0005	LOW		REAL		工程量量程下限
0006	MODE		BYTE		模式
0007	EN		BOOL		使能
0008	OUTPUT		REAL		输出值
0009	ERROR		BYTE		错误标识

编程语言 | **程　序**

梯形图（LD）



结构化文本（ST）

```
HS_LK850_AI (EN:=EN, In:=INPUT, High:=HIGH, Low:=LOW,
    Mode:=MODE, Q=>Q, Out=>OUTPUT, Error=>ERROR);
```

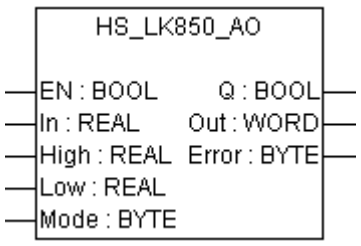
程序说明：

- 输入 INPUT 为十进制显示的现场输入值，HIGH 和 LOW 分别为现场信号对应的量程上下限；

- MODE 标识该现场信号模式为“4”即信号大小为-10~10V；
- OUTPUT 显示经转换后的工程量，该值为 0~100 的数据，该数据将被用于数据读取和控制。

5.10.2 HS_LK850_AO——LK850 模拟输出量程转换指令

- **功能**：提供 LK850 模块 AO 通道的量程转换功能。
- **指令示例：**

梯形图语言示例


- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能输入	当 EN=FALSE 时，停止转换，Q=FALSE。 当 EN=TRUE 时，开始转换，Q=TRUE， 将工程量的量程范围 Low-High 对应于十进制数。	FALSE
In	REAL	输入工程量值	各量程意义下的工程量值	0 0
High	REAL	量程上限	初始值为 0	0
Low	REAL	量程下限	初始值为 0	0
Mode	BYTE	模式选择	=0：输出电流信号 4～20.58mA =1：输出电流信号 0～20.58mA =2：输出电压信号 0～+5.125V =3：输出电压信号 0～+10.25V =4：输出电压信号 -10.25～+10.25V	4
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	使能输出	当 EN=FALSE 时，停止转换，Q=FALSE。 当 EN=TRUE 时，开始转换，Q=TRUE， 将工程量的量程范围 Low-High 对应于十进制数。	FALSE
Out	WORD	输出码值	0～65535	0
Error	BYTE	错误类型	=0：当前模式量程转换无错 =1：量程下限超过上限 =2：模式输入错误 =3：输入工程量超出量程范围	0

从工程量数据到输出量的转换模式说明：

转换模式	转换数据	量程下限	量程上限	输出信号类型
MODE=0	工程量	Low	High	电流信号：4~20mA
	十进制数	38527	61567	
	十六进制数	967F	F07F	
MODE=1	工程量	Low	High	电流信号：0~20mA
	十进制数	32768	63007	
	十六进制数	8000	F61F	
MODE=2	工程量	Low	High	电压信号：0~5V
	十进制数	32768	47527	

	十六进制数	8000	B9A7	
MODE=3	工程量	Low	High	电压信号：0~10V
	十进制数	32768	62287	
	十六进制数	8000	F34F	
MODE=4	工程量	Low	High	电压信号：-10~10V
	十进制数	3248	62287	
	十六进制数	0CB0	F34F	

✧ **指令使用举例**

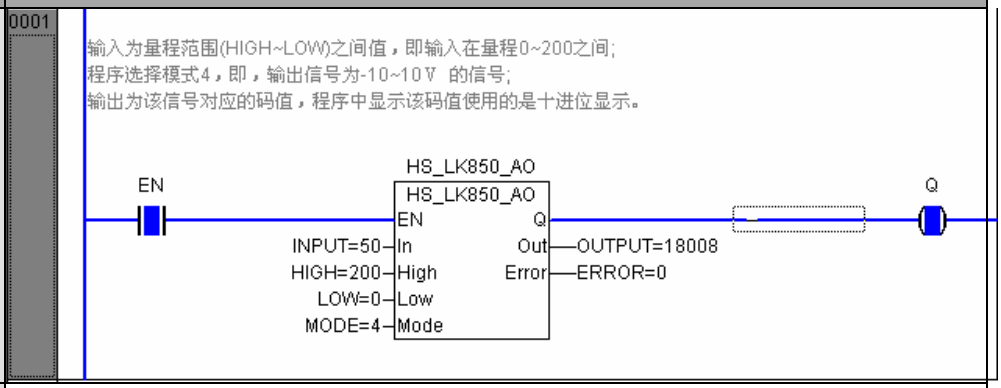
变量定义

VAR

	名称	地址	类型	初始值	注释
0001	INPUT		WORD		输入码值
0002	Q		BOOL		转换完成标识
0003	HIGH		REAL		工程量量程上限
0004	LOW		REAL		工程量量程下限
0005	MODE		BYTE		模式
0006	EN		BOOL		使能
0007	OUTPUT		REAL		输出值
0008	ERROR		BYTE		错误标识
0009	HS_LK850_AO		HS_LK850_AO		LK850模拟输出量程转换功能块

编程语言 | **程　序**

梯形图（LD）



结构化文本（ST）

```
HS_LK850_AO (EN:=EN, In:=INPUT, High:=HIGH, Low:=LOW,
    Mode:=MODE, Q=>Q, Out=>OUTPUT, Error=>ERROR);
```

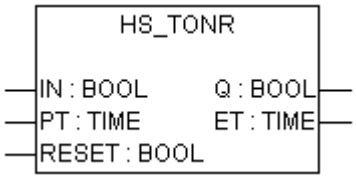
程序说明：

- INPUT 为量程 0~200 的数值，信号的大小为模式 4 即-10~10V；

5.11 时间相关指令库 HS_Timer.Lib

5.11.1 HS_TONR——保持型通电延时定时器

- **注意：**该指令是 PowerPro V4.3.0B 版本软件中增加的指令，它适用于 LK202-A01, LK205-A01, LK207-A03, LK209-A01, LK210-B05 以及更高型号的主控，如果用户的主控型号低于以上几类，建议用户**不要调用** HS_Timer.Lib 库，否则工程将不能下装。
- **功能：**保持型通电延时定时器，可以累计输入信号的接通时间。
- **指令示例：**

梯形图语言示例


- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
IN	BOOL	计时触发端	当 IN 变成 TRUE 时，ET 以毫秒计时直到 ET 等于 PT。如有外部触发导致在延时时间未达设定时间值前 IN 变成 FALSE，此时 ET 保持计时停止前的时间值，等到 IN 再次触发变成 TRUE 时，ET 继续计时，直到 ET 等于 PT，输出 Q 置位	0
PT	TIME	定时时间值	设定的延时时间值，请按照 TIME 数据类型的格式要求来设定，单位可以是 d，h，m，s 和 ms，最小单位为 ms	0
RESET	BOOL	复位端	TRUE：复位所有的输出和中间变量，结束定时 FALSE：允许执行延时定时功能	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	定时器溢出标志	当 Q 为 FALSE 时，说明 ET 未达设定值 PT。当 ET 等于 PT 时，Q 为 TRUE。计时开始时，无论 IN 为 TRUE 还是 FALSE，都不影响 Q 的输出，Q 的输出只取决于 ET 是否等于 PT 以及 RESET 端状态	0
ET	TIME	当前时间值	当 IN 变成 TRUE 时，ET 以毫秒计时直到 ET 等于 PT。计时开始时，无论 IN 为 TRUE 还是 FALSE，只要 Q 为 FALSE，ET 会一直输出当前时间值，直到 ET 等于 PT	0

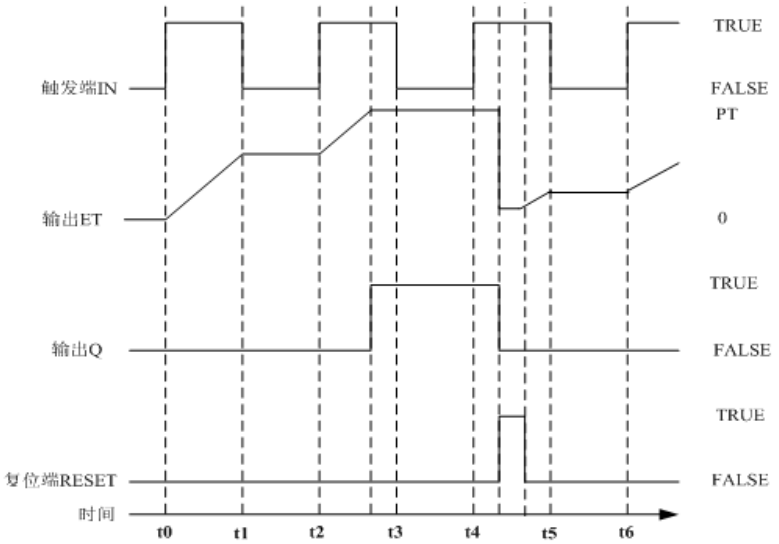


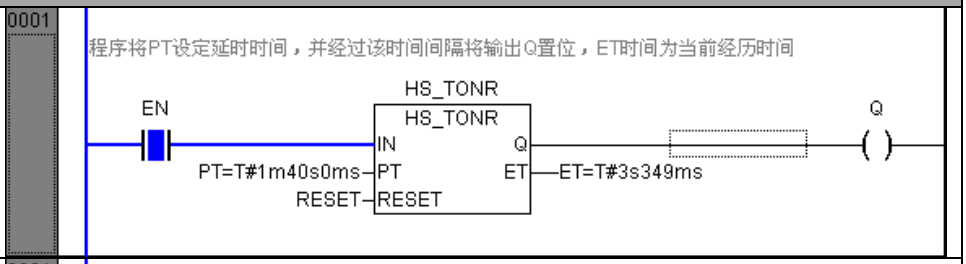
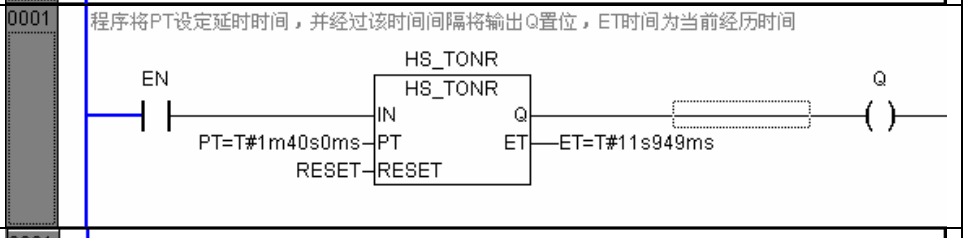
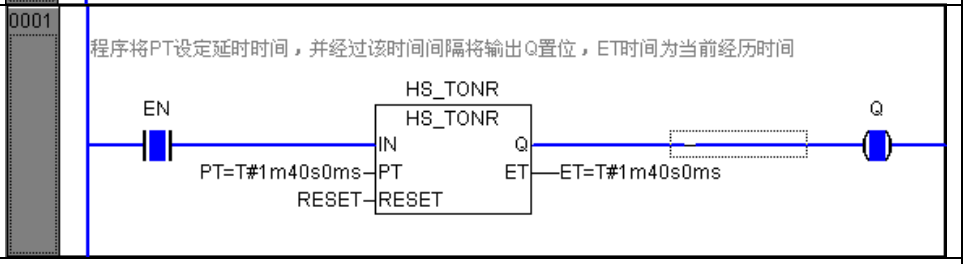
图 5-11-1 HS_TONR 时序图

✧ **指令使用举例**

变量定义

VAR

	名称	地址	类型	初始值	注释
0001	EN		BOOL		
0002	HS_TONR		HS_TONR		保持型通电延时定时器
0003	RESET		BOOL		复位端
0004	PT		TIME		设定的延时时间值
0005	ET		TIME		当前时间值
0006	Q		BOOL		完成标识

编程语言	程　序
梯形图（LD）（运行过程中）	
梯形图（LD）（EN 为 FALSE 时，ET 保持）	
梯形图（LD）（定时结束）	
结构化文本（ST）	HS_TONR (IN:=EN, PT:=PT, RESET:=RESET, Q=>Q, ET:=ET);

程序说明：

- 当 ET 值与 PT 值相等时，定时器输出 Q 为 1；
- 重新设定 PT 值时需先置 RESET 为 TRUE，而后再将其置位为 FALSE。经过此项操作后 ET 值为 T#0ms。

5.11.2 HS_GetCPUCounter——获取运行时间指令

- 注意：该指令是 PowerPro V4.3.1B 软件版本中新增的指令，它适用于 LK202-B01，LK205-B01，LK207-A04 ,LK209-B01，LK210-B06 以及更高型号的主控，如果用户的主控型号低于以上几类，请到 Backup Library 中调用适合您主控的 HS_Timer.Lib 库版本文件，具体操作方法请参见附录 3。
- 功能：获取 PLC 上电之后的运行时间，支持精度为 ms。
- 说明：该指令属于风电定制功能。

5.11.3 HS_GetTime——获取系统时间指令

- 注意：该指令是 PowerPro V4.3.1B 软件版本中新增的指令，它适用于 LK202-B01，LK205-B01，LK207-A04 ,LK209-B01，LK210-B06 以及更高型号的主控，如果用户的主控型号低于以上几类，请到 Backup Library 中调用适合您主控的 HS_Timer.Lib 库版本文件，具体操作方法请参见附录 3。
- 功能：获取 PLC 当前的系统时间，支持精度为 ms。
- 说明：该指令属于风电定制功能。

5.11.4 HS_SetLocalTime——设置系统时间指令

- 注意：该指令是 PowerPro V4.3.1B 软件版本中新增的指令，它适用于 LK202-B01，LK205-B01，LK207-A04 ,LK209-B01，LK210-B06 以及更高型号的主控，如果用户的主控型号低于以上几类，请到 Backup Library 中调用适合您主控的 HS_Timer.Lib 库版本文件，具体操作方法请参见附录 3。
- 功能：设置 PLC 当前的系统时间，支持精度为 ms。
- 说明：该指令属于风电定制功能。

5.11.5 HS_SetTime2RTCTime——设置 RTC 时间指令

- 注意：该指令是 PowerPro V4.3.1B 软件版本中新增的指令，它适用于 LK202-B01，LK205-B01，LK207-A04 ,LK209-B01，LK210-B06 以及更高型号的主控，如果用户的主控型号低于以上几类，请到 Backup Library 中调用适合您主控的 HS_Timer.Lib 库版本文件，具体操作方法请参见附录 3。
- 功能：将 PLC 当前的系统时间写入 RTC，精确到秒。
- 说明：该指令属于风电定制功能。

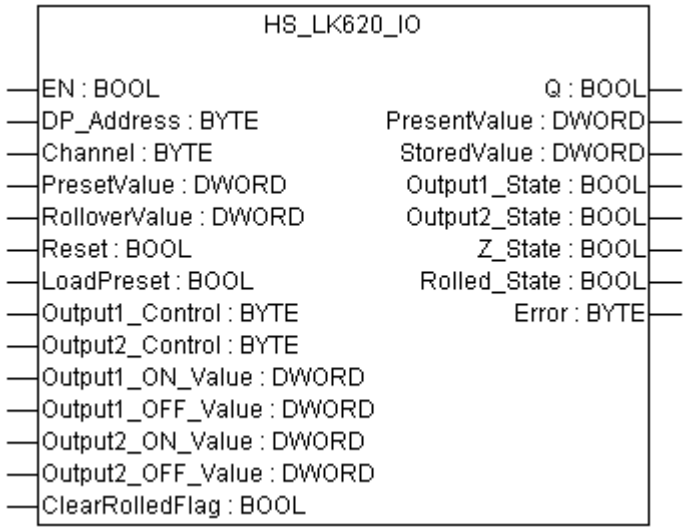
5.12 LK620 输入输出指令库

注意：LK620 输入输出指令库（HS_LK620_IO.Lib）是 PowerPro V4.3.1B 版本软件中增加的指令库，它适用于 LK202-B01, LK205-B01, LK207-A04, LK209-B01, LK210-B06 以及更高型号的主控，如果用户的主控型号低于以上几类，建议用户不要调用此库，否则工程将不能下装。

5.12.1 HS_LK620_IO——LK620 输入输出指令

- **功能**：提供设置 LK620 模块的参数以及读取 LK620 模块的运算状态的用户接口。
- **指令示例**：

梯形图语言示例



- **参数说明**：

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能输入	0：无效 1：有效	0
DP_Address	BYTE	站地址	选择某站地址的 LK620 模块	2
Channel	BYTE	通道号	=1，通道 1（计数器 1） =2，通道 2（计数器 2）	1
PresetValue	DWORD	预置值	计数器从该值开始，此值必须小于翻转值，否则会发生计数错误，预置值的取值范围 0~4,294,967,295。	0
RolloverValue	DWORD	翻转值	计数模式下，作为计数的上限，取值范围为 1~4,294,967,295。在计数过程中，当计数值＝（翻转值－1）时，计数器回到 0，重新开始计数。	0
Reset	BOOL	复位计数器	0：无效 0→1：上升沿，计数器复位并从 0 开始计数 1：两个输出通道输出 OFF，不按照组态设置输出	0
LoadPreset	BOOL	预置值控制位	0：无效 1：上升沿有效，计数器载入预置值并从预置值开始计数	0
Output1_Control	BYTE	输出控制	用于控制计数器各输出点的状态，输出 1 当前输出值修改： =0x00，根据计数结果输出 =0x02，强制修改 OUT1 输出值为 OFF =0x03，强制修改 OUT1 输出值为 ON	0x00

Output2_Control	BYTE	输出控制	用于控制计数器各输出点的状态，输出 2 当前输出值修改： =0x00，根据计数结果输出 =0x02，强制修改 OUT1 输出值为 OFF =0x03，强制修改 OUT1 输出值为 ON	0x00
Output1_ON_Value	DWORD	输出 ON 触发值	输出 1 输出 ON 触发值(0~4,294,967,295)	0
Output1_OFF_Value	DWORD	输出 OFF 触发值	输出 1 输出 OFF 触发值(0~4,294,967,295)	0
Output2_ON_Value	DWORD	输出 ON 触发值	输出 2 输出 ON 触发值(0~4,294,967,295)	0
Output2_OFF_Value	DWORD	输出 OFF 触发值	输出 2 输出 OFF 触发值(0~4,294,967,295)	0
ClearRolledFlag	BOOL	翻转标志清除位	清除翻转标志位	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	输出完成标志	0：输出未完成 1：输出完成	
PresentValue	DWORD	当前计数值	存放所选计数器的当前计数值	
StoredValue	DWORD	存储计数值	所选计数器的当前存储计数值	
Output1_State	BOOL	输出 1 状态	1：ON（通道闭合），即当前计数值达到 Output1_ON_Value 0：OFF（通道断开，停止输出），即当前计数达到 Output1_OFF_Value	
Output2_State	BOOL	输出 2 状态	1：ON（通道闭合），即当前计数值达到 Output2_ON_Value 0：OFF（通道断开，停止输出），即当前计数达到 Output2_OFF_Value	
Z_State	BOOL	Z 状态	0：低电平 1：高电平	
Rolled_State	BOOL	翻转标志	计数器是否达到翻转值并翻转 0：未翻转 1：已翻转	
Error	BYTE	错误信息	=0， 无错误 .0=1，此号站配置的不是 620 模块 .1=1，输入从站地址不合法 .2=1，通道值错误 .3=1，预设值错误 .4=1，输出控制位错误 .5=1，输出触发值错误	

提示：

- 预置值必须小于翻转值，否则会发生计数错误。
- 当前计数值达到翻转值时，计数器翻转回 0，从 0 重新开始计数，并不是从预置值开始.
- 若 Reset 和 LoadPreset 同时被写入上升沿，计数器载入预置值并从预置值开始计数。
- 输出 ON 触发值应小于翻转值；若设定的输出 ON 触发值≥翻转值，则计数值不可能达到输出 ON 触发值，输出通道不会输出 ON。
- 输出 OFF 触发值应小于翻转值；若设定的输出 OFF 触发值≥翻转值，则计数值不可能达到输出 OFF 触发值，输出通道不会输出 OFF。
- 当输出 OFF 触发值＝输出 ON 触发值时，则输出 OFF。
- 测频模式和计数模式相关用法参考《LK620 24VDC 双通道计数模块使用说明书》。
- 请结合《LK620 24VDC 双通道计数模块使用说明书》来使用本功能块。

各输入输出参数与 LK620 模块数据的对应关系，如下图所示：

图 5-12-1 输入输出参数与 LK620 模块数据的对应关系

✧ **指令使用举例**

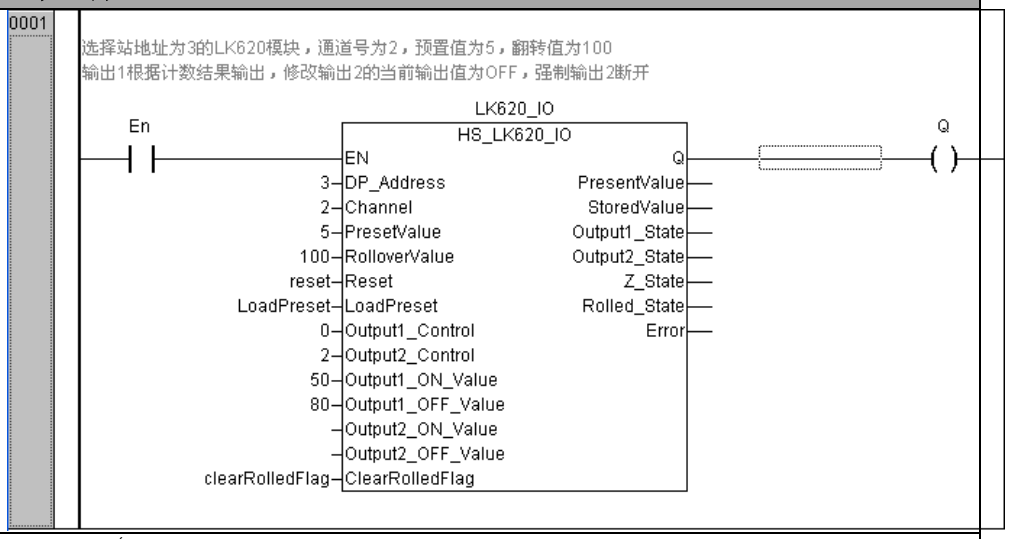
变量定义

VAR

	Name	Address	Type	Initial	Comment
0001	En		BOOL		
0002	LK620_IO		HS_LK620_IO		
0003	reset		BOOL		
0004	LoadPreset		BOOL		
0005	clearRolledFlag		BOOL		
0006	Q		BOOL		

编程语言 | **程　序**

梯形图（LD）



结构化文本（ST）

```
LK620_IO(EN:=En, DP_Address:=3, Channel:=2 , PresetValue:= 5,
      RolloverValue:=100 ,Reset:=reset, LoadPreset:= LoadPreset,
     Output1_Control:=0, Output2_Control:=2,
     Output1_ON_Value:= 50, Output1_OFF_Value:= 80,
     ClearRolledFlag:= clearRolledFlag, Q=>Q, );
```

程序说明：

- **选择站地址为 3 的 LK620 模块，通道号为 2（选择计数器 2），预置值为 5，翻转值为 100。**
- 输出 1 根据计数结果输出，修改输出 2 的当前输出值为 OFF，强制输出 2 断开。
- 预置值必须小于翻转值，否则会发生计数错误。

5.13 文件操作指令库（HS_File.lib）

注意：文件指令库（HS_File.lib）是 **PowerPro V4.3.2B** 版本软件中增加的指令库，它目前只适用于 LK211 型号的主控，不适合于其它型号的主控。如果用户使用其它型号主控，建议用户不要调用此库，否则工程将不能下装。

注意：HS_File_Transfer 主要用于读写二进制文件，它是对 HS_File_Open、HS_File_Read、HS_File_Write、HS_File_Close 功能的封装。HS_File_Transfer 简化了文件操作，推荐用户使用 HS_File_Transfer。

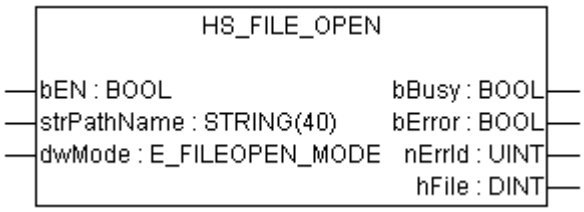
注意：HS_File_Transfer_Config 主要用于读写文本文件，它也是对 HS_File_Open、HS_File_Read、HS_File_Write、HS_File_Close 功能的封装。读写文本文件会对控制器性能产生较大影响，推荐用户直接读写二进制文件，一般不要读写文本文件。

注意：如果要使用文件功能，则 LK211 必须插入专用高速 SDHC 存储卡，SDHC 卡最低容量是 4G。文件会保存到 SDHC 卡中，可以通过 FTP 连接 LK211 进行文件传输。LK211 中存取文件的路径是：\mnt\sd。

5.13.1 HS_File_Open——打开文件指令

- **功能**：打开要操作的文件
- **指令示例**：

梯形图语言示例



- **参数说明**：

输入参数	数据类型	功能描述	参数值说明	默认值
bEN	BOOL	上升沿触发文件打开操作	上升沿触发开始，保持高电平有效	0
strPathName	STRING[40]	打开文件的路径	待操作文件的路径	Null
dwMode	E_FILEOPEN_MODE	文件的打开模式	详见后面	FOPEN_MODENONE
输出参数	**数据类型**	**功能描述**	**参数值说明**	
bBusy	BOOL	是否正在执行命令	打开文件是否执行过程中	0
bError	BOOL	执行结果是否有错	打开文件过程中是否出错	0
nErrId	UINT	错误的 ID 号	0x20 ： 文件名不合理； 0x40 ： 文件打开模式不合理； 0x80 ： 文件打开失败。	0
hFile	DINT	打开文件的句柄	-	-1

提示：

- 文件名不能包括特殊字符，特殊字符包括*、\、$、[、]、+、-、&、%、#、!、~、遇见特殊字符会报错误，文件名中的空格会自动删除，文件名中不会包含空格
- bEN 使能上升沿有效，打开文件过程中 bEN 需要保持高电平

dwMode 数据类型选择如下表所示：

dwMode	C 语言	含义
FOPEN_MODENONE :=0		初始值
FOPEN_MODEREAD_OR_FOPEN_MODETEXT:= 1	r	只读文本文件
FOPEN_MODEREAD_OR_FOPEN_MODEBINARY:= 2	rb	只读二进制文件
FOPEN_MODEWRITE_OR_FOPEN_MODETEXT:= 3	w	只写文本文件
FOPEN_MODEWRITE_OR_FOPEN_MODEBINARY:= 4	wb	只写二进制文件
FOPEN_MODEAPPEND_OR_FOPEN_MODETEXT:= 5,	a	追加写文本文件
FOPEN_MODEAPPEND_OR_FOPEN_MODEBINARY:= 6	ab	追加写二进制文件
FOPEN_MODEREAD_OR_FOPEN_MODEPLUS_OR_FOPEN_MODETEXT:= 7	r+	读文本文件
FOPEN_MODEREAD_OR_FOPEN_MODEPLUS_OR_FOPEN_MODEBINARY:= 8	rb+	读二进制文件
FOPEN_MODEWRITE_OR_FOPEN_MODEPLUS_OR_FOPEN_MODETEXT:=9	w+	写文本文件
FOPEN_MODEWRITE_OR_FOPEN_MODEPLUS_OR_FOPEN_MODEBINARY:= 10	wb+	写二进制文件
FOPEN_MODEAPPEND_OR_FOPEN_MODEPLUS_OR_FOPEN_MODETEXT:= 11,	a+	追加可读写文本文件
FOPEN_MODEAPPEND_OR_FOPEN_MODEPLUS_OR_FOPEN_MODEBINARY:= 12	ab+	追加可读写二进制文件

✧ **指令使用举例**

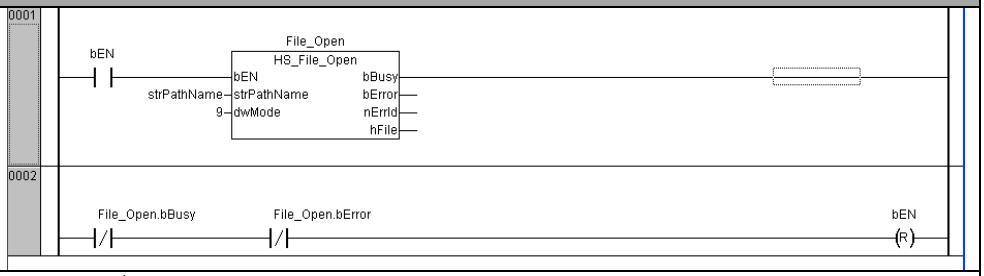
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT | RETAIN | INFO

	名称	地址	类型	初始值	注释
0001	File_Open		HS_File_Open		
0002	bEN		BOOL		
0003	strPathName		STRING(80)	file_data	

编程语言 | **程 序**

梯形图（LD）



结构化文本（ST）

```
File_Open(
      bEN:=bEN ,
      strPathName:=strPathName ,
      dwMode:=9 ,
      bBusy=> ,
      bError=> ,
      nErrId=> ,
      hFile=> );
IF File_Open.bBusy=0 THEN
      IF File_Open.bError THEN
            bEN:=0;
      END_IF
END_IF
```

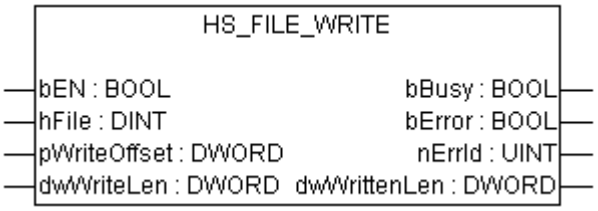
程序说明：

手动触发 bEN，开始打开文件 file_data，打开结束后 bEN 复位

5.13.2 HS_File_Write——写入文件指令

- **功能**：向打开的文件中写入数据。
- **指令示例**：

梯形图语言示例



- **参数说明**：

输入参数	数据类型	功能描述	参数值说明	默认值
bEN	BOOL	上升沿触发文件写入操作	上升沿触发开始，保持高电平有效	0
hFile	DINT	写入文件的句柄	获取打开的文件句柄	0
pWriteOffset	DWORD	写入内容的首地址	向文件写入的数据起始地址，例如:200，代表起始地址为%MB200	0
dwWriteLen	DWORD	写入数据的长度	写入的连续字节长度，如 100，代表 100 个字节	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
bBusy	BOOL	是否正在执行命令	写入文件是否执行过程中	0
bError	BOOL	执行结果是否有错	写入文件过程中是否出错	0
nErrId	UINT	错误的 ID 号	0x01：文件句柄错误； 0x03： 每次写入数据的最大长度超过 2*1024*1024； 0x05：所写数据超出了 M 区的范围； 0x07： 获取写数据区地址失败； 0x09： 写入数据出错。	0
dwWrittenLen	DWORD	实际写入的长度	向文件中实际写入的数据字节长度	0

提示：

- 文件名不能包括特殊字符，特殊字符包括*、\、$、[、]、+、-、&、%、#、!、~、遇见特殊字符会报错误，文件名中的空格会自动删除，文件名中不会包含空格
- bEN 使能上升沿有效，写入文件过程中 bEN 需要保持高电平
- hFile 必须是已打开的要写入文件的句柄

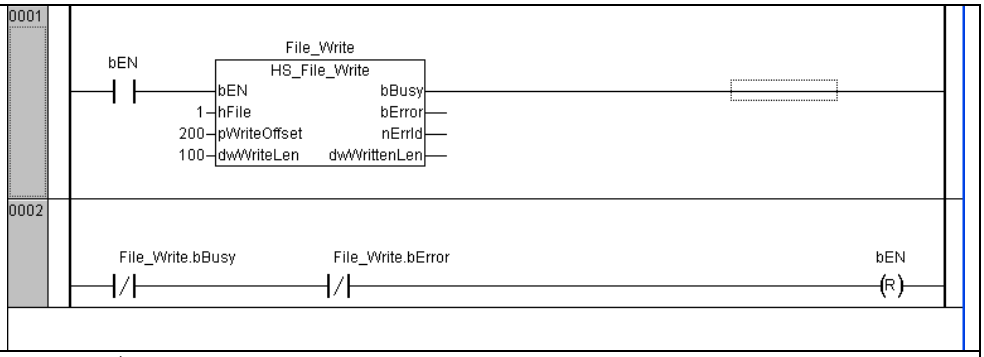
- **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT | RETAIN | INFO

	名称	地址	类型	初始值	注释
0001	File_Write		HS_File_Write		
0002	bEN		BOOL		

编程语言 程 序

梯形图（LD）	

结构化文本（ST）	File_Write(
	bEN:=bEN，
	hFile:=1，
	pWriteOffset:=200，
	dwWriteLen:=100，
	bBusy=>，
	bError=>，
	nErrId=>，
	dwWrittenLen=>);
	IF File_Write.bBusy=0 THEN
	IF File_Write.bError=0 THEN
	bEN:=0;
	END_IF
	END_IF

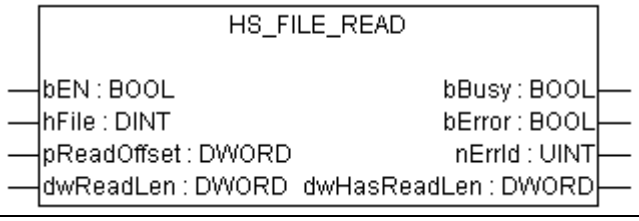
程序说明：

- **手动触发 bEN，开始将%MB200 起始，长度为 100 字节(%MB200~%MB299)的数据写入到文件 file_data 中，写文件结束后 bEN 复位**

5.13.3 HS_File_Read——读取文件指令

- **功能**：从打开的文件中数据到 PLC 的 M 区中。
- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
bEN	BOOL	上升沿触发文件读取操作	上升沿触发开始，保持高电平有效	0
hFile	DINT	读取文件的句柄	获取打开的文件句柄	0
pReadOffset	DWORD	写入读取内容的首地址	读取文件到 PLC 的 M 区数据起始地址，例如:200，代表起始地址为%MB200	0
dwReadLen	DWORD	读取数据的长度	读取的连续字节长度，如 100，代表 100 个字节	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
bBusy	BOOL	是否正在执行	读取文件是否执行过程中	0

		命令		
bError	BOOL	执行结果是否有错	读取文件过程中是否出错	0
nErrId	UINT	错误的 ID 号	0x01： 文件句柄错误； 0x02： 每次读取数据的最大长度超过 2*1024*1024； 0x04： 读取的数据超出了 M 区的范围； 0x08： 获取读数据区地址失败； 0x10： 读取数据出错。	0
dwHasReadLen	DWORD	实际读取的长度		0

提示：

- 文件名不能包括特殊字符，特殊字符包括*、\、$、[、]、+、-、&、%、#、!、~、遇见特殊字符会报错误，文件名中的空格会自动删除，文件名中不会包含空格
- bEN 使能上升沿有效，读取文件过程中 bEN 需要保持高电平
- hFile 必须是已打开的要读取文件的句柄

✧ **指令使用举例**

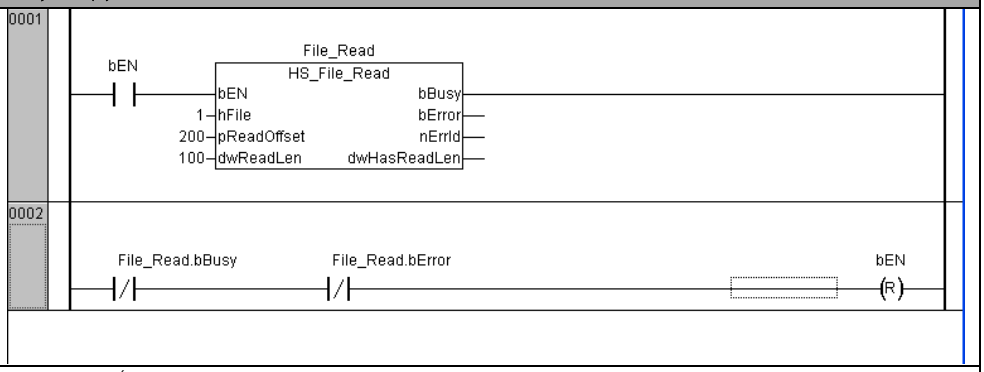
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT | RETAIN | INFO

	名称	地址	类型	初始值	注释
0001	File_Read		HS_File_Read		
0002	bEN		BOOL		

编程语言 | **程 序**

梯形图（LD）



结构化文本（ST）

```
File_Read(
      bEN:=bEN ,
      hFile:=1 ,
      pReadOffset:=200 ,
      dwReadLen:= 100 ,
      bBusy=> ,
      bError=> ,
      nErrId=> ,
      dwHasReadLen=> );
IF File_Read.bBusy=0 THEN
      IF File_Read.bError=0 THEN
            bEN:=0;
      END_IF
END_IF
```

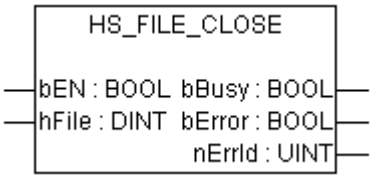
程序说明：

手动触发 bEN，开始将文件 file_data 中的数据读取，存放在%MB200 起始，长度为 100 字节(%MB200~%MB299)的 M 区中

5.13.4 HS_File_Close——关闭文件指令

- **功能**：将操作完毕的文件关闭。
- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
bEN	BOOL	上升沿触发，文件关闭操作	上升沿触发开始，保持高电平有效	0
hFile	DINT	关闭文件的句柄	获取打开的文件句柄	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
bBusy	BOOL	是否正在执行命令	关闭文件是否执行过程中	0
bError	BOOL	执行结果是否有错	关闭文件过程中是否出错	0
nErrId	UINT	错误的 ID 号	0x01： 文件名句柄有错； 0x06： 关闭文件出错。	0

提示：

- 文件名不能包括特殊字符，特殊字符包括*、\、$、[、]、+、-、&、%、#、!、~、遇见特殊字符会报错误，文件名中的空格会自动删除，文件名中不会包含空格
- bEN 使能上升沿有效，关闭文件过程中 bEN 需要保持高电平
- hFile 必须是已打开的要读取文件的句柄

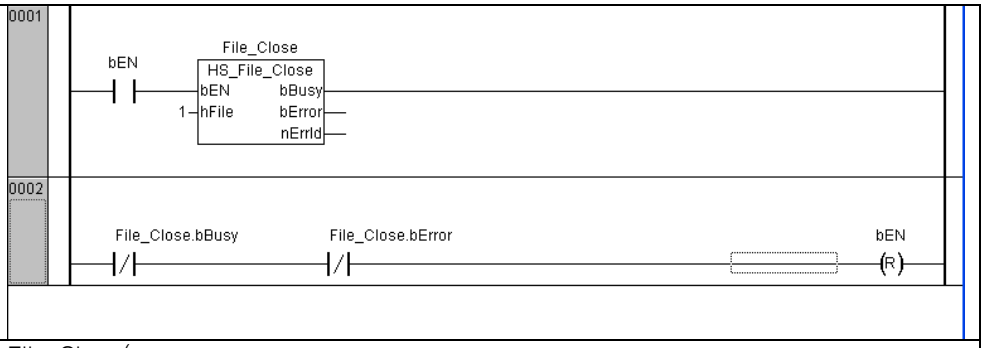
✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT | RETAIN

	名称	地址	类型	初始值	注释
0001	File_Close		HS_File_Close		
0002	bEN		BOOL		

编程语言	程　序

梯形图（LD）	

结构化文本（ST）	File_Close(
	bEN:=bEN ,
	hFile:=1 ,
	bBusy=> ,
	bError=> ,
	nErrId=>);
	IF File_Close.bBusy THEN
	IF File_Close.bError THEN
	bEN:=0;
	END_IF
	END_IF

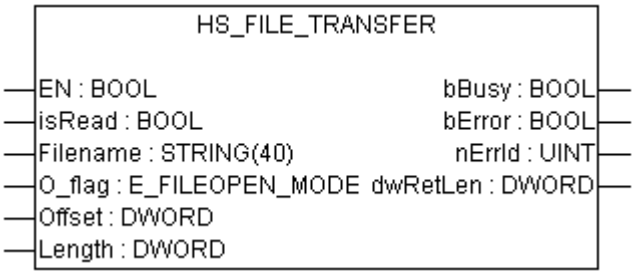
程序说明：

- **手动触发 bEN，开始关闭文件**
-

5.13.5 HS_File_Transfer——读写二进制文件指令

- **功能**：将 PLC 中数据写入到文件中，或将文件中的数据读取到 PLC 中
- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	上升沿触发文件打开操作	上升沿触发，读/写过程保持高电平	0
isRead	BOOL	True,读取功能; False,写入功能	选择读取或写入	0
Filename	STRING[40]	打开文件的路径	需要操作的文件名称	Null
O_flag	E_FILEOPEN_MODE	文件的打开模式	详见后面	0
Offset	DWORD	读取/写入内容的首地址	M 区存放文件数据的首地址，%MB，例如%MB200	0
Length	DWORD	读取/写入数据的长度	数据字节长度，例如 100，代表 100 个字节	0

输出参数	数据类型	功能描述	参数值说明	
bBusy	BOOL	是否正在执行命令	关闭文件是否执行过程中	0
bError	BOOL	执行结果是否有错	关闭文件过程中是否出错	0
nErrId	UINT	错误的 ID 号	0x01： 文件句柄错误； 0x02： 每次读取数据的最大长度超过 2*1024*1024； 0x03： 每次写入数据的最大长度超过 2*1024*1024； 0x04： 读取的数据超出了 M 区的范围； 0x05：所写数据超出了 M 区的范围； 0x06： 关闭文件出错。0x07： 获取写数据区地址失败； 0x08： 获取读数据区地址失败； 0x09： 写入数据出错。 0x10： 读取数据出错。 0x20 ： 文件名不合理； 0x40 ： 文件打开模式不合理； 0x80 ： 文件打开失败。	0
dwRetLen	DWORD	实际读取/写入的长度	实际读取/写入的字节长度	0

提示：

- 文件名不能包括特殊字符，特殊字符包括*、\、$、[、]、+、-、&、%、#、!、~、遇见特殊字符会报错误，文件名中的空格会自动删除，文件名中不会包含空格
- EN 使能上升沿有效，操作文件过程中 EN 需要保持高电平

O_flag 数据类型选择如下表所示：

O_flag	C 语言	含义
FOPEN_MODENONE :=0		初始值
FOPEN_MODEREAD_OR_FOPEN_MODETEXT:= 1	r	只读文本文件
FOPEN_MODEREAD_OR_FOPEN_MODEBINARY:= 2	rb	只读二进制文件
FOPEN_MODEWRITE_OR_FOPEN_MODETEXT:= 3	w	只写文本文件
FOPEN_MODEWRITE_OR_FOPEN_MODEBINARY:= 4	wb	只写二进制文件
FOPEN_MODEAPPEND_OR_FOPEN_MODETEXT:= 5,	a	追加写文本文件
FOPEN_MODEAPPEND_OR_FOPEN_MODEBINARY:= 6	ab	追加写二进制文件
FOPEN_MODEREAD_OR_FOPEN_MODEPLUS_OR_FOPEN_MODETEXT:= 7	r+	读文本文件
FOPEN_MODEREAD_OR_FOPEN_MODEPLUS_OR_FOPEN_MODEBINARY:= 8	rb+	读二进制文件
FOPEN_MODEWRITE_OR_FOPEN_MODEPLUS_OR_FOPEN_MODETEXT:=9	w+	写文本文件
FOPEN_MODEWRITE_OR_FOPEN_MODEPLUS_OR_FOPEN_MODEBINARY:= 10	wb+	写二进制文件
FOPEN_MODEAPPEND_OR_FOPEN_MODEPLUS_OR_FOPEN_MODETEXT:= 11,	a+	追加可读写文本文件
FOPEN_MODEAPPEND_OR_FOPEN_MODEPLUS_OR_FOPEN_MODEBINARY:= 12	ab+	追加可读写二进制文件

✧ 指令使用举例

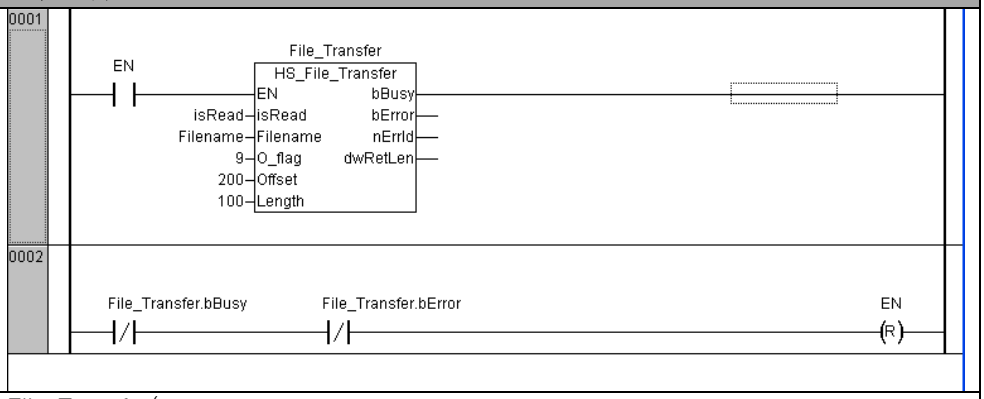
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT | RETAIN | IN

	名称	地址	类型	初始值	注释
0001	File_Transfer		HS_File_Transfer		
0002	EN		BOOL		
0003	isRead		BOOL		
0004	Filename		STRING	'file_data'	

编程语言 | **程 序**

梯形图（LD）



结构化文本（ST）

```
File_Transfer(
      EN:=EN ,
      isRead:=isRead ,
      Filename:=Filename ,
      O_flag:=9 ,
      Offset:=200 ,
      Length:=100 ,
      bBusy=> ,
      bError=> ,
      nErrId=> ,
      dwRetLen=> );
IF File_Transfer.bBusy=0 THEN
      IF File_Transfer.bError=0 THEN
            EN:=0;
      END_IF
END_IF
```

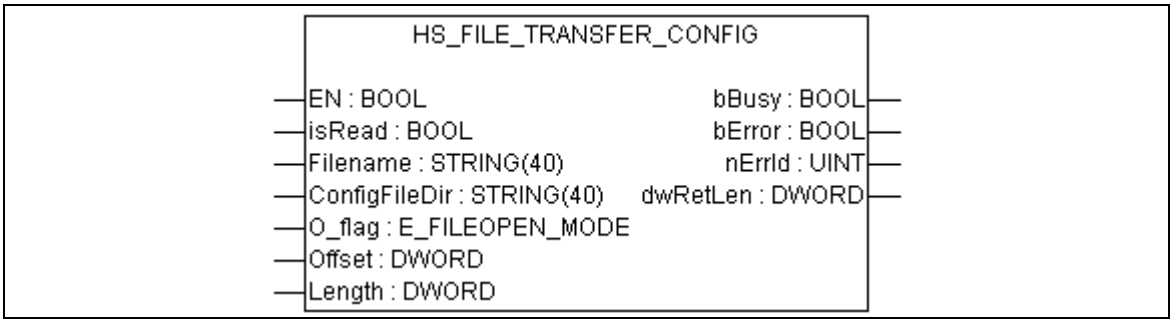
程序说明：

手动触发 EN，开始写文件

5.13.6 HS_File_Transfer_Config——读写文本文件指令

- **功能**：将 PLC 中数据按照配置文件格式，写入到文件中
- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	上升沿触发文件打开操作	上升沿触发，读/写过程保持高电平	0
isRead	BOOL	True,读取功能; False,写入功能	选择读取或写入	0
Filename	STRING[40]	打开文件的路径	需要操作的文件名称	Null
ConfigFileDir	STRING[40]	配置文件的路径	配置文件	Null
O_flag	E_FILEOPEN_MODE	文件的打开模式	详见下面	0
Offset	DWORD	读取/写入内容的首地址	M 区存放文件数据的首地址，%MB，例如%MB200	0
Length	DWORD	读取/写入数据的长度	数据字节长度，例如 100，代表 100 个字节	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
bBusy	BOOL	是否正在执行命令	关闭文件是否执行过程中	0
bError	BOOL	执行结果是否有错	关闭文件过程中是否出错	0
nErrId	UINT	错误的 ID 号	0x01： 文件句柄错误； 0x02： 每次读取数据的最大长度超过 2*1024*1024； 0x03： 每次写入数据的最大长度超过 2*1024*1024； 0x04： 读取的数据超出了 M 区的范围； 0x05：所写数据超出了 M 区的范围； 0x06： 关闭文件出错。0x07： 获取写数据区地址失败； 0x08： 获取读数据区地址失败； 0x09： 写入数据出错。 0x10： 读取数据出错。 0x20 ： 文件名不合理； 0x40 ： 文件打开模式不合理； 0x80 ： 文件打开失败。	0
dwRetLen	DWORD	实际读取/写入的长度	实际读取/写入的字节长度	0

提示：

- 文件名不能包括特殊字符，特殊字符包括*、\、$、[、]、+、-、&、%、#、!、~、遇见特殊字符会报错误，文件名中的空格会自动删除，文件名中不会包含空格
- EN 使能上升沿有效，操作文件过程中 EN 需要保持高电平

O_flag 数据类型选择如下表所示：

O_flag	C 语言	含义
FOPEN_MODENONE :=0		初始值
FOPEN_MODEREAD_OR_FOPEN_MODETEXT:= 1	r	只读文本文件
FOPEN_MODEREAD_OR_FOPEN_MODEBINARY:= 2	rb	只读二进制文件
FOPEN_MODEWRITE_OR_FOPEN_MODETEXT:= 3	w	只写文本文件
FOPEN_MODEWRITE_OR_FOPEN_MODEBINARY:= 4	wb	只写二进制文件
FOPEN_MODEAPPEND_OR_FOPEN_MODETEXT:= 5,	a	追加写文本文件
FOPEN_MODEAPPEND_OR_FOPEN_MODEBINARY:= 6	ab	追加写二进制文件
FOPEN_MODEREAD_OR_FOPEN_MODEPLUS_OR_FOPEN_MODETEXT:= 7	r+	读文本文件
FOPEN_MODEREAD_OR_FOPEN_MODEPLUS_OR_FOPEN_MODEBINARY:= 8	rb+	读二进制文件
FOPEN_MODEWRITE_OR_FOPEN_MODEPLUS_OR_FOPEN_MODETEXT:=9	w+	写文本文件
FOPEN_MODEWRITE_OR_FOPEN_MODEPLUS_OR_FOPEN_MODEBINARY:= 10	wb+	写二进制文件
FOPEN_MODEAPPEND_OR_FOPEN_MODEPLUS_OR_FOPEN_MODETEXT:= 11,	a+	追加可读写文本文件
FOPEN_MODEAPPEND_OR_FOPEN_MODEPLUS_OR_FOPEN_MODEBINARY:= 12	ab+	追加可读写二进制文件

✧ **指令使用举例**

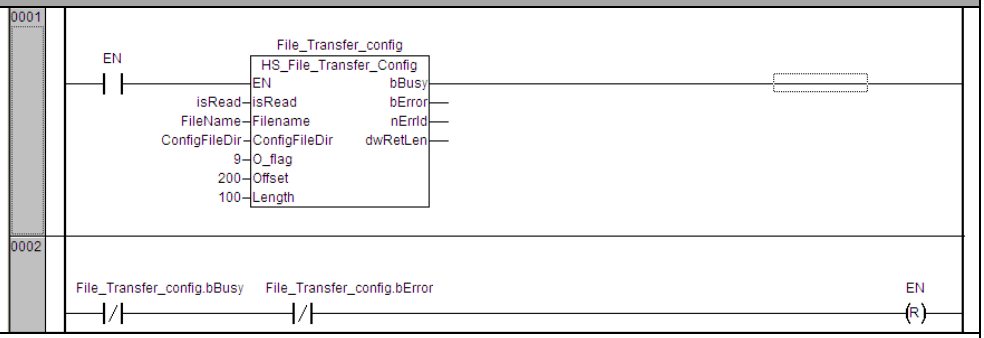
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT | RETAIN | INFO

	名称	地址	类型	初始值	注释
0001	File_Transfer_Config		HS_File_Transfer_C		
0002	EN		BOOL		
0003	isRead		BOOL		
0004	Filename		STRING		
0005	ConfigFileDir		STRING		

编程语言 | **程 序**

梯形图（LD）



结构化文本（ST）

```
File_Transfer_Config(
        EN:=EN ,
        isRead:=isRead ,
        Filename:=Filename ,
        ConfigFileDir:=ConfigFileDir ,
        O_flag:=9 ,
        Offset:=200 ,
        Length:=100 ,
        bBusy=> ,
        bError=> ,
        nErrId=> ,
        dwRetLen=> );
IF File_Transfer_Config.bBusy=0 THEN
        IF File_Transfer_Config.bError=0 THEN
                EN:=0;
        END_IF
END_IF
```

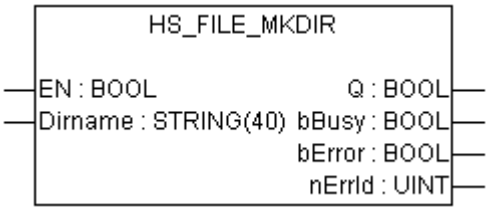
程序说明：

手动触发 EN，开始写文件

5.13.7 HS_File_MkDir——新建文件夹指令

- **功能**：在 LK211 的 SD 卡上新建文件夹
- **指令示例**：

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	上升沿触发新建文件夹操作	上升沿触发，新建过程保持高电平	0
Dirname	STRING[40]	创建的文件夹名	需要创建的文件夹名	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	0：功能块正在执行或失败； 1：功能块执行正确。	0：功能块正在执行或失败； 1：功能块执行正确。	
bBusy	BOOL	是否正在执行命令	新建文件夹是否执行过程中	0
bError	BOOL	执行结果是否有错	新建文件夹过程中是否出错	0
nErrId	UINT	错误的 ID 号	0x01：超时时间必须大于 1000ms； 0x02： 文件名不合理； 0x04：路径创建失败。	0

提示：

- 文件名不能包括特殊字符，特殊字符包括*、\、$、[、]、+、-、&、%、#、!、~、遇见特殊字符会报错误，文件名中的空格会自动删除，文件名中不会包含空格
- EN 使能上升沿有效，创建文件夹过程中 EN 需要保持高电平

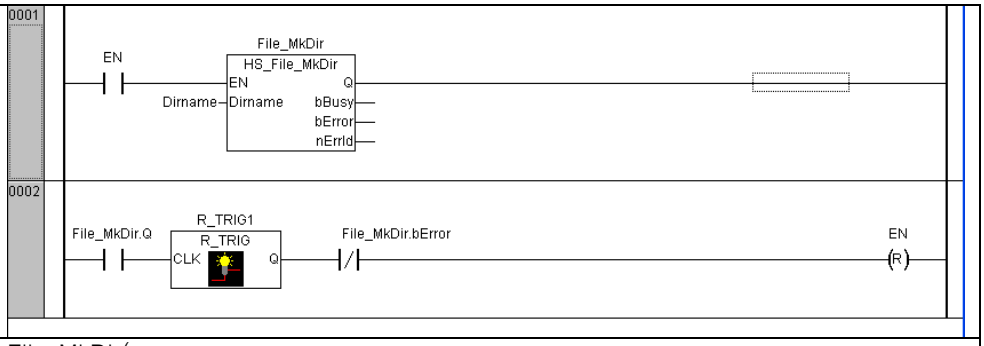
✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT | RETAIN | INFO

	名称	地址	类型	初始值	注释
0001	File_MkDir		HS_File_MkDir		
0002	EN		BOOL		
0003	Dirname		STRING		
0004	R_TRIG1		R_TRIG		

编程语言	程　序

梯形图（LD）	

结构化文本（ST）	File_MkDir(
	EN:=EN ,
	Dirname:=Dirname ,
	Q=> ,
	bBusy=> ,
	bError=> ,
	nErrId=>);
	R_TRIG1(CLK:=File_MkDir.Q , Q=>);
	IF R_TRIG1.Q=1 THEN
	IF File_MkDir.bError=0 THEN
	EN:=0;
	END_IF
	END_IF

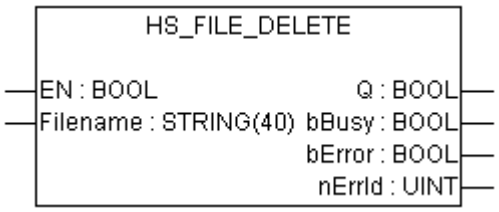
程序说明：

手动触发 EN，开始创建文件夹

5.13.8 HS_File_Delete——删除文件或文件夹指令

- **功能**：将 LK211 的 SD 卡上文件或文件夹强制删除
- **指令示例：**

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	上升沿触发操作	上升沿触发，删除过程保持高电平	0
Dirname	STRING[40]	打开文件的路径	需要删除的文件名	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	0：功能块正在执行或失败； 1：功能块执行正确。	0：功能块正在执行或失败； 1：功能块执行正确。	
bBusy	BOOL	是否正在执行命令	删除操作是否执行过程中	0
bError	BOOL	执行结果是否	删除操作过程中是否出错	0

		有错		
nErrId	UINT	错误的 ID 号	0x01 ： 超时时间必须大于 1000ms； 0x02 ： 路径名不合理； 0x04 ： 路径删除失败。	0

提示：

- 文件名不能包括特殊字符，特殊字符包括*、\、$、[、]、+、-、&、%、#、!、~、遇见特殊字符会报错误，文件名中的空格会自动删除，文件名中不会包含空格
- EN 使能上升沿有效，删除文件过程中 EN 需要保持高电平

✧ **指令使用举例**

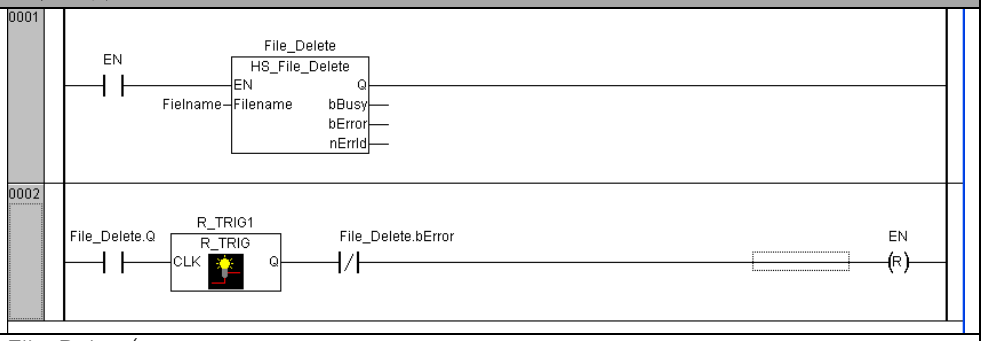
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT | RETAIN | INFO

	名称	地址	类型	初始值	注释
0001	File_Delete		HS_File_Delete		
0002	EN		BOOL		
0003	Fielname		STRING		
0004	R_TRIG1		R_TRIG		

编程语言 | **程 序**

梯形图（LD）



结构化文本（ST）

```
File_Delete(
      EN:=EN ,
      Filename:=Fielname ,
      Q=> ,
      bBusy=> ,
      bError=> ,
      nErrId=> );
R_TRIG1(CLK:=File_Delete.Q , Q=> );
IF R_TRIG1.Q:=1 THEN
      IF File_Delete.bError=0 THEN
            EN:=0;
      END_IF
END_IF
```

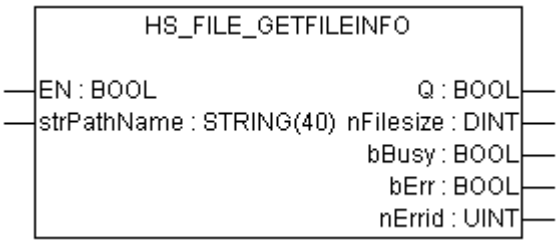
程序说明：

手动触发 EN，开始删除文件夹

5.13.9 HS_File_GetFileInfo——获取文件信息指令

- **功能**：获取 LK211 的 SD 卡上文件的文件信息
- **指令示例：**

梯形图语言示例



➢ **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	上升沿触发文件打开操作	上升沿触发，获取信息过程保持高电平	0
strPathName	STRING[40]	打开文件的路径	需要获取信息的文件名	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	输出是否成功 1：成功； 0：失败。	输出是否成功 1：成功； 0：失败。	
nFilesize	DINT	文件大小，字节为单位	获取文件大小	
bBusy	BOOL	是否正在执行命令	获取文件信息是否执行过程中	0
bErr	BOOL	是否出错	获取文件信息过程中是否出错	0
nErrId	UINT	错误的 ID 号	0x01： 超时时间必须大于 1000ms； 0x02： 路径名不合理； 0x04：无法获得文件信息。	0

提示：

- 文件名不能包括特殊字符，特殊字符包括*、\、$、[、]、+、-、&、%、#、!、~、遇见特殊字符会报错误，文件名中的空格会自动删除，文件名中不会包含空格
- EN 使能上升沿有效，获取文件信息过程中 EN 需要保持高电平

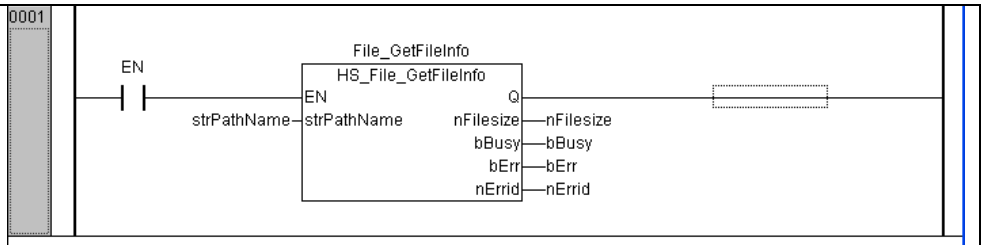
✧ **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT | RETAIN | INFO

	名称	地址	类型	初始值	注释
0001	File_GetFileInfo		HS_File_GetFileInfo		
0002	EN		BOOL		
0003	strPathName		STRING(80)	'file_data.txt'	
0004	nFilesize		DINT		
0005	bBusy		BOOL		
0006	bErr		BOOL	0	
0007	nErrid		UINT	0	

编程语言 | **程 序**

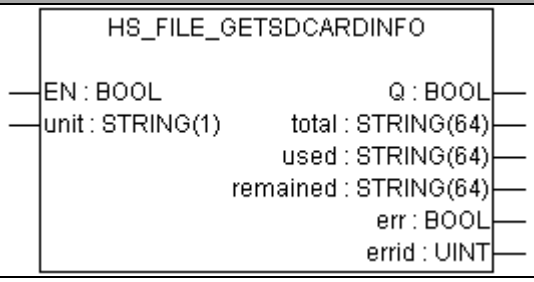
梯形图（LD）	
结构化文本（ST）	File_GetFileInfo(EN:=EN , strPathName:=strPathName , Q=> , nFilesize=> nFilesize, bBusy=>bBusy , bErr=>bErr , nErrid=>nErrid);

程序说明：

手动触发 EN，开始获取文件信息

5.13.10 HS_File_GetSDCardInfo——获取控制器 SD 卡信息指令

- **功能**：获取 LK211 的 SD 卡的信息
- **指令示例：**

梯形图语言示例


- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	上升沿触发文件打开操作	上升沿触发，获取信息过程保持高电平	0
unit	STRING[1]	输出显示单位	显示 SD 卡容量的大小	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	输出是否成功 1：成功； 0：失败。	输出是否成功 1：成功； 0：失败。	
total	STRING[64]	SD 卡总容量		
used	STRING[64]	SD卡已经使用的容量		0
remained	STRING[64]	SD卡剩余容量		0
err	BOOL	是否出错	0：正常； 1：出错	0
errid	UINT	错误码	0x01：SD 卡无法加载； 0x02：无法获得 SD 卡信息。	

提示：

- 文件名不能包括特殊字符，特殊字符包括*、\、$、[、]、+、-、&、%、#、!、~、遇见特殊字符会报错误，文件名中的空格会自动删除，文件名中不会包含空格
- EN 使能上升沿有效，获取 SD 信息过程中 EN 需要保持高电平

✧ **指令使用举例**

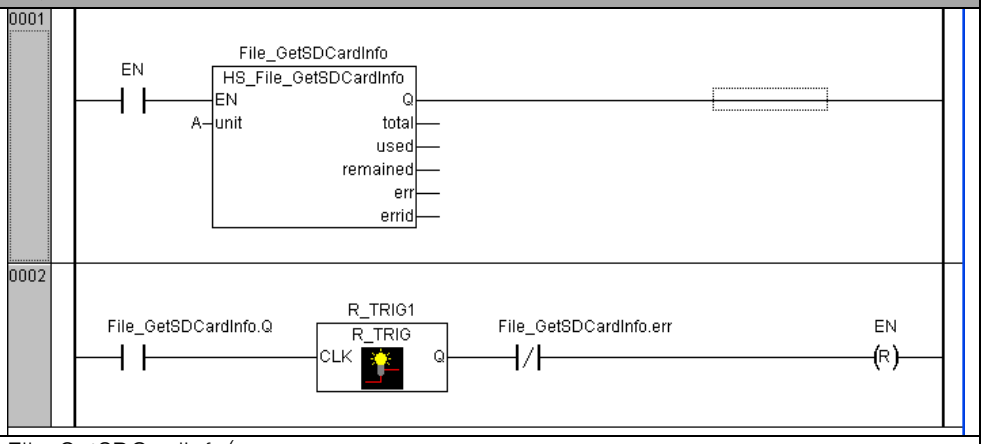
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT | RETAIN | INFO

	名称	地址	类型	初始值	注释
0001	File_GetSDCardInfo		HS_File_GetSDCar		
0002	EN		BOOL		
0003	A		STRING		
0004	R_TRIG1		R_TRIG		

编程语言 | **程　序**

梯形图（LD）



结构化文本（ST）

```
File_GetSDCardInfo(
        EN:=EN ,
        unit:='A' ,
        Q=> ,
        total=> ,
        used=> ,
        remained=> ,
        err=> ,
        errid=> );
R_TRIG1(CLK:=File_GetSDCardInfo.Q , Q=> );
IF R_TRIG1.Q=1 THEN
        IF File_GetSDCardInfo.err=0 THEN
                EN:=0;
        END_IF
END_IF
```

程序说明：

手动触发 EN，开始获取 SD 卡信息信息

➢

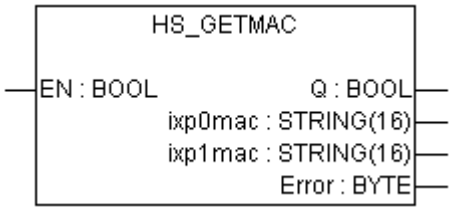
5.14 获取控制器网络 MAC 指令库（HS_GetMAC.lib）

注意：获取控制器网络 MAC 指令库（HS_GetMAC.lib）是 PowerPro V4.3.2B 版本软件中增加的指令库，它适用于 LK211 和 LKUCS V2.3.5 版本及更高型号的主控，如果用户使用其它型号主控，建议用户不要调用此库，否则工程将不能下装。

5.14.1 HS_GetMAC——获取控制器网络 MAC 指令

- **功能**：获取控制器网络 MAC
- **指令示例**：

梯形图语言示例



- **参数说明**：

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	上升沿触发	上升沿触发有效	0
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	执行完毕	执行完毕后置 1	0
ixp0mac	STRING(16)		128 网段网络 MAC	0
ixp1mac	STRING(16)		129 网段网络 MAC	0
Error	BYTE	打开文件的句柄	0x01：系统错误	-1

- **指令使用举例**

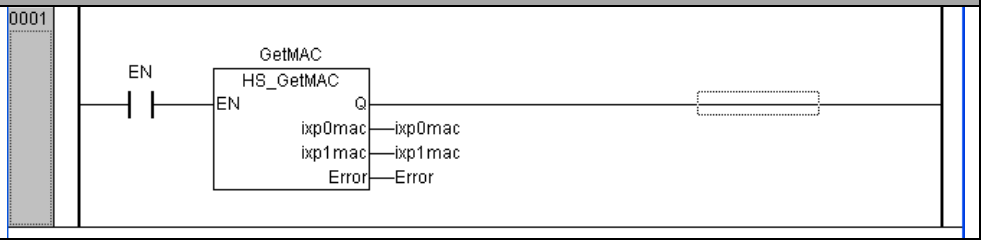
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT | RETAIN

	名称	地址	类型	初始值	注释
0001	GetMAC		HS_GetMAC		
0002	EN		BOOL		
0003	ixp0mac		STRING(16)		
0004	ixp1mac		STRING(16)		
0005	Error		BYTE		

编程语言 | **程　序**

梯形图（LD）



结构化文本（ST）	GetMAC(EN:=EN， Q=>， ixp0mac=>ixp0mac， ixp1mac=>ixp0mac， Error=>Error);

程序说明：

手动触发 EN，可以得到控制器网络 MAC

5.15 冗余主从切换指令库（MaterSlaveSwitch.lib）

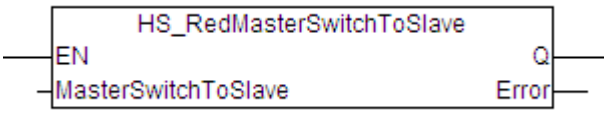
注意：此库是针对 LKUCS V2.3.5B 版本设计，如果用户使用其它型号主控，建议用户不要调用此库，否则工程将不能下装。

- **若主控是单机运行，则指令不会起实际作用。**

5.15.1 HS_RedMasterSwitchToSlave——冗余主从切换指令

- 功能：冗余运行的主控主从状态切换
- 指令示例：

梯形图语言示例



- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	上升沿触发	上升沿触发，高电平有效	FALSE
MasterSwitchToSlave	BOOL	是否允许切换	TRUE 允许切换，FALSE 不允许切换	FALSE
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	执行完毕	执行完毕后置 TRUE	FALSE
Error	BYTE	是否有错	错误信息： 0：正确 1：单机系统 2：主从同步尚未完成 3：从机的钥匙开关在 PRG 状态 4：从机网络有问题	0

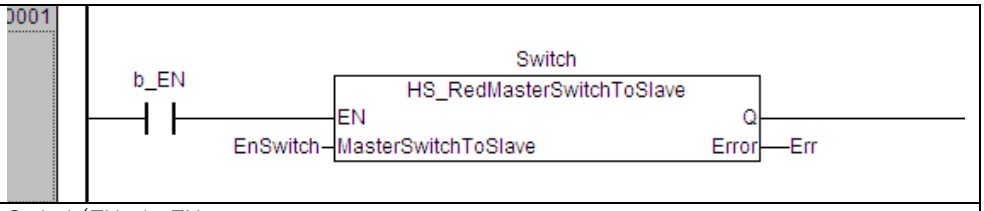
- **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT | RETAIN

	名称	地址	类型	初始值	注释
0001	b_EN		BOOL		
0002	Switch		HS_RedMasterSwit		
0003	EnSwitch		BOOL		
0004	Err		BYTE		

编程语言	程　序

梯形图（LD）	
结构化文本（ST）	Switch(EN:=b_EN, MasterSwitchToSlave:=EN_Switch, Q=>, Error=>Err);

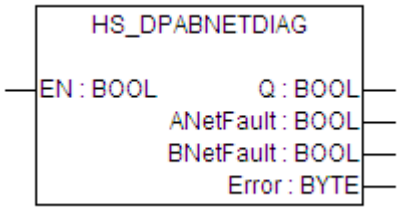
程序说明：手动执行主从切换

5.16 DP 双网诊断指令库（HS_DPABNetDiag.lib）

注意：目前 DP 双网诊断指令库只适用与 LK210-B10 模块。

5.16.1 HS_DPABNetDiag——DP 双网诊断指令

- 功能：检测 DP 双网通信是否正常
- 指令示例：

梯形图语言示例


- **参数说明：**

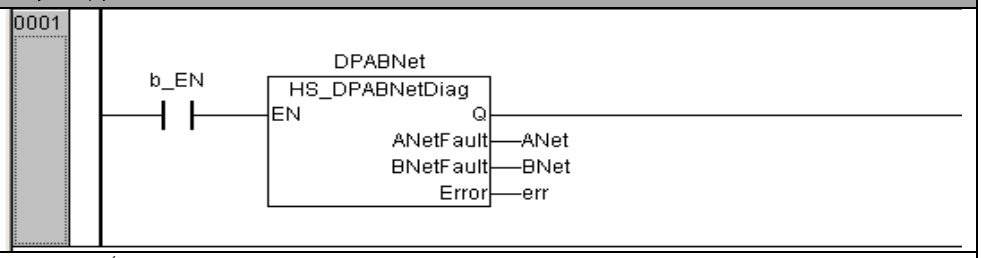
输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	上升沿触发	上升沿触发，高电平有效	FALSE
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	执行完毕	执行完毕后置 TRUE	FALSE
ANetFault	BOOL	A 网是否正常	FALSE：正常；TRUE：故障	FALSE
ANetFault	BOOL	B 网是否正常	FALSE：正常；TRUE：故障	FALSE
Error	BYTE	是否有模块离线	错误信息： 0：正确 1：有模块离线	0

- **指令使用举例**

变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT | RETAIN | INFO

	名称	地址	类型	初始值	注释
0001	DPABNet		HS_DPABNetDiag		
0002	b_EN		BOOL	TRUE	
0003	ANet		BOOL		
0004	BNet		BOOL		
0005	err		BYTE		

编程语言	程　序
梯形图（LD）	
结构化文本（ST）	DPABNet(EN:=b_EN, Q=>, ANetFault=>ANet, BnetFault=>Bnet, Error=>err);

程序说明：

- 手动触发 b_EN，可以查看 A、B 网运行状况和是否有无模块离线。

注意：

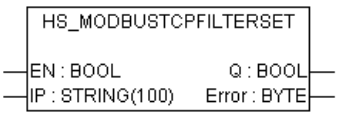
- 一旦有模块处于离线状态，则 A,B 网的诊断信息有可能误报。

5.17 IP 过滤指令库 (HS_ModbusTCPFilterSet.lib)

注意：此库是针对 LKUCS V2.3.6B 版本设计，如果用户使用其它型号主控，建议用户不要调用此库，否则工程将不能下装。

5.17.1 HS_ModbusTCPFilterSet——IP 过滤指令

- **功能**：本功能块用于过滤用户不允许的 ModBusTCP 主站请求。
- **指令示例：**

梯形图语言示例


- **参数说明：**

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：上升沿使能，高电平有效	0
IP	STRING	允许的 IP 地址	IP 与 IP 之间请以“;” 隔开，最大支持 5 个	无
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	设置完成	0：未完成	0

			1：已完成	
Error	BYTE	错误信息	为 0：正确 不为 0：有错误 =1：功能块重复调用 =2：IP 地址错 =3：输入的 IP 地址超过 5 个	0

✧ **指令使用举例**

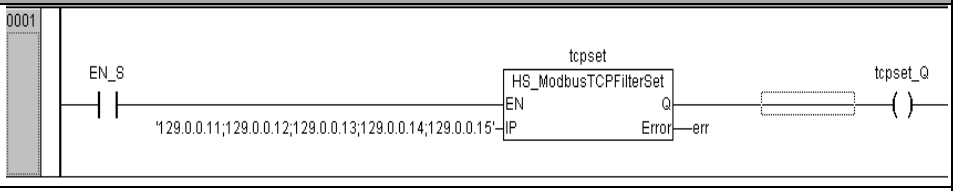
变量定义

VAR | VAR_INPUT | VAR_OUTPUT | VAR_IN_OUT | CONSTANT | RETAIN | INFO

	名称	地址	类型	初始值	注释
0001	tcpset		HS_ModbusTCPFilt		
0002	err		BYTE		
0003	EN_S		BOOL	1	
0004	tcpset_Q		BOOL		

编程语言 | **程　序**

梯形图（LD）



结构化文本（ST）

```
tcpset(EN:=EN_S,IP:='129.0.0.11;129.0.0.12;129.0.0.13;129.0.0.14;129.0.0.15',
    Q=>tcpset_Q, Error=>err);
```

程序说明：

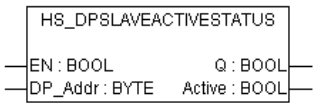
- 上电或全下装后，EN 上升沿使能高电平维持有效，请将本功能块放在 HS_ModbusTCPSlave 功能块前面一起配合使用且需设置一次即可，高电平期间若有不允许的 ModbusTcp 主站请求，会限制其访问。功能块低电平期间取消过滤。设置完毕后，Q 等于 TRUE。

5.18 从站离线状态指令库(HS_DPSlaveActiveStatus.lib)

注意：此库必须配合库 HS_Diagnosis.lib 使用，请使用时，务必添加上 HS_Diagnosis.lib 库，否则工程将会编译出错。

5.18.1 HS_DPSlaveActiveStatus——从站离线状态指令

- **功能**：本功能块用于 LK 主控判断从站模块是否在线。
- **指令示例**：

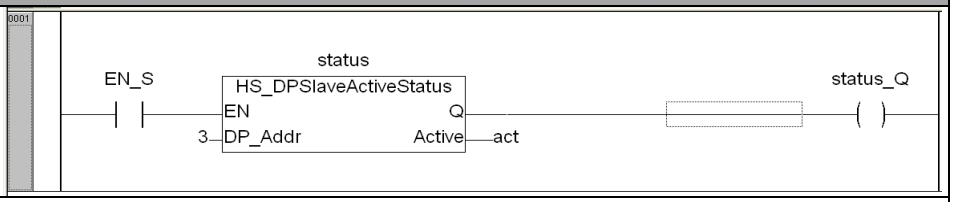
梯形图语言示例


- **参数说明**：

输入参数	数据类型	功能描述	参数值说明	默认值
EN	BOOL	使能	0：无效 1：高电平有效	0
DP_Addr	BYTE	从站地址	需要检测的从站地址	无
输出参数	**数据类型**	**功能描述**	**参数值说明**	
Q	BOOL	诊断执行结束	0：未完成 1：已完成	0
Active	BOOL	从站状态	为 TRUE： 从站上线且数据交换正常 为 FALSE：从站离线或者数据没有准备好	0

- **指令使用举例**

变量定义

编程语言	程 序
梯形图（LD）	
结构化文本（ST）	status(EN:=EN_S,DP_Addr:='3,Q=>status_Q, Active=>act);

程序说明：

- 本功能块可以用于判断功能块是否离线，如果 Active 引脚为 FALSE 时即表示从站离线或者从站数据和主控并没有进行正常的数据交换。

附录

➢ 1、LK 指令速查表

基 本 指 令			
指令类型	**指令说明**	**指令名**	**所在库**
算术运算指令	加法指令	ADD（FUN）	无
	乘法指令	MUL（FUN）	
	减法指令	SUB（FUN）	
	除法指令	DIV（FUN）	
	取余指令	MOD（FUN）	
赋值指令	赋值指令	MOVE（FUN）	无
逻辑运算指令	与指令	AND（FUN）	无
	或指令	OR（FUN）	
	异或指令	XOR（FUN）	
	取非指令	NOT（FUN）	
移位指令	左移指令	SHL（FUN）	无
	右移指令	SHR（FUN）	
	循环左移指令	ROL（FUN）	
	循环右移指令	ROR（FUN）	
选择指令	二选一指令	SEL（FUN）	无
	取最大值指令	MAX（FUN）	
	取最小值指令	MIN（FUN）	
	极限值指令	LIMIT（FUN）	
	多选一指令	MUX（FUN）	
比较指令	大于指令	GT（FUN）	无
	小于指令	LT（FUN）	
	大于等于指令	GE（FUN）	
	小于等于指令	LE（FUN）	
	等于指令	EQ（FUN）	
	不等于指令	NE（FUN）	
数据类型转换指令	布尔类型转换指令	BOOL_TO_<TYPE>(FUN)	无
	字节类型转换指令	BYTE_TO_<TYPE>(FUN)	
	日期类型转换指令	DATE_TO_<TYPE>(FUN)	
	长整型转换指令	DINT_TO_<TYPE>(FUN)	
	日期时间类型转换指令	DT_TO_<TYPE>(FUN)	
	双字类型转换指令	DWORD_TO_<TYPE>(FUN)	
	整数类型转换指令	INT_TO_<TYPE>(FUN)	

指令类型	指令说明	指令名	所在库
数据类型转换指令	字类型转换指令	WORD_TO_<TYPE>(FUN)	无
	实数类型转换指令	REAL_TO_<TYPE>(FUN)	
	短整型转换指令	SINT_TO_<TYPE>(FUN)	
	字符类型转换指令	STRING_TO_<TYPE>(FUN)	
	时钟类型转换指令	TIME_TO_<TYPE>(FUN)	
	时间类型转换指令	TOD_TO_<TYPE>(FUN)	
	无符号长整型转换指令	UDINT_TO_<TYPE>(FUN)	
	无符号整型转换指令	UINT_TO_<TYPE>(FUN)	
	无符号短整型转换指令	USINT_TO_<TYPE>(FUN)	
	截短转换指令	TRUNC(FUN)	
初等数学运算指令	绝对值指令	ABS(FUN)	无
	平方根指令	SQRT(FUN)	
	自然对数指令	LN(FUN)	
	常用对数指令	LOG(FUN)	
	指数指令	EXP(FUN)	
	正弦指令	SIN(FUN)	
	余弦指令	COS(FUN)	
	正切指令	TAN(FUN)	
	反正弦指令	ASIN(FUN)	
	反余弦指令	ACOS(FUN)	
	反正切指令	ATAN(FUN)	
	幂指令	EXPT(FUN)	
地址运算指令	取地址指令	ADR(FUN)	无
	取地址内容指令	^(FUN)	
	位地址指令	BITADR(FUN)	
	索引指令	INDEXOF（FUN）	
	数据类型大小指令	SIZEOF（FUN）	
调用指令	调用运算指令	CAL(FUN)	无
初始化操作指令	初始化操作指令	INI(FUN)	无
字符串指令	结合字符串指令	CONCAT(FUN)	Standard.lib
	删除字符指令	DELETE(FUN)	
	查找字符串指令	FIND(FUN)	
	插入字符串指令	INSERT(FUN)	
	左边取字符串指令	LEFT(FUN)	
	字符串长度指令	LEN(FUN)	
	中间取字符串指令	MID(FUN)	
	替换字符串指令	REPLACE(FUN)	
	右边取字符串指令	RIGHT(FUN)	
库版本信息指令	读取库版本查看	Version_Util(FUN)	Util.lib
软件版本信息指令	读取软件版本信息	SyslibGetVersion2300(FUN)	SysLibC16x.lib

指令类型	指令说明	指令名	所在库
事件使用指令	事件调用	SysCallbackRegister(FUN)	SysLibCallback.lib
	解除事件调用	SysCallbackUnregistrer(FUN)	
检查指令	数组边界检查	CheckBounds(FUN)	HS_Check.lib
	字节型除数为零检查	CheckDivByte(FUN)	
	字型除数为零检查	CheckDivWord(FUN)	
	双字形型除数为零检查	CheckDivDword(FUN)	
	实型除数为零检查	CheckDivReal(FUN)	
	整型边界检查	CheckRangeSigned(FUN)	
	无符号整型边界检查	CheckRangeUnsigned(FUN)	
BCD 转换指令	BCD 码转整型指令	BCD_TO_INT(FUN)	Util.lib
	整型转 BCD 码指令	INT_TO_BCD(FUN)	
位/字节操作指令	位提取指令	EXTRACT(FUN)	Util.lib
	位整合指令	PACK(FUN)	
	位赋值指令	PUTBIT(FUN)	
	位拆分指令	UNPACK(FB)	
高等数学运算指令	微分	DERIVATIVE(FB)	Util.lib
	积分	INTEGRAL(FB)	
	整型统计	STATISTIC_INT(FB)	
	实型统计	STATISTIC_REAL(FB)	
	平方偏差	VARIANCE(FB)	
控制器指令	比例控制器	P(FB)	Util.lib
	比例微分控制器	PD(FB)	
	比例积分微分控制器	PID(FB)	
	比例积分微分控制器	PID_FIXCYCLE(FB)	
信号发生器指令	脉冲信号发生器	BLINK(FB)	Util.lib
	典型周期信号发生器	GEN(FB)	
SFC 动作控制指令	SFC 动作控制	SFCActionControl(FB)	Iecsfc.lib
函数操纵器指令	特征曲线	CHARCURVE(FB)	Util.lib
	整型限速	RAMP_INT(FB)	
	实型限速	RAMP_REAL(FB)	
模拟量处理指令	滞后	HYSTERESIS(FB)	Util.lib
	上下限报警	LIMITALARM(FB)	
双稳态指令	置位优先双稳态器	SR(FB)	Standard.lib
	复位优先双稳态器	RS(FB)	
触发器	上升沿检测触发器	R_TRIG(FB)	Standard.lib
	下降沿检测触发器	F_TRIG(FB)	
计数器	递增计数器	CTU(FB)	Standard.lib
	递减计数器	CTD(FB)	
	递增递减计数器	CTUD(FB)	
定时器	普通定时器	TP(FB)	Standard.lib
	通电延时定时器	TON(FB)	
	断电延时定时器	TOF(FB)	
	实时时钟	RTC(FB)	

扩 展 指 令			
指令类型	**指令说明**	**指令名**	**所在库**
模拟量应用指令	测量数据转工程量	HS_HEX_ENGIN	HS_AnalogConvert.Lib
	工程量转输出数据	HS_ ENGIN _ HEX	
数据传送指令	数据传送	HS_MOVE	HS_Move.lib
COM1 口通讯指令	COM1 通讯参数设置	HS_SetParameter_COM1	HS_Comunication.lib
	COM1 Modbus 主站通讯	HS_ModbusMaster_COM1	
	COM1 自由协议通讯数据发送	HS_SEND_COM1	
	COM2 自由协议通讯数据接收	HS_RECEIVE_COM1	
COM2 口通讯指令	COM2 通讯参数设置	HS_SetParameter_COM2	
	COM2 Modbus 主站通讯	HS_ModbusMaster_COM2	
	COM2 自由协议通讯数据发送	HS_SEND_COM2	
	COM2 自由协议通讯数据接收	HS_RECEIVE_COM2	
以太网通讯指令	ModBus TCP 从站	HS_ModBusTCPSlave	
	ModBus TCP 主站通讯	HS_ModBusTCPMaster	
站间引用通讯	站间引用通讯数据发送	HS_NetData_Send	
	站间引用通讯数据接收	HS_NetData_Receive	
DP SOE 指令	SOE 存入 M 区	HS_DP_SOE_M	HS_SOE.lib
	DP SOE 读取	HS_DP_SOE_Read	
预置输出指令	预置输出指令	HS_ScheduledTime	HS_ScheduledTime.lib
实时时钟指令	设置实时时钟	HS_Set_RTC	HS_RTC.lib
	读取实时时钟	HS_Get_RTC	
诊断指令	CPU 负荷诊断	HS_GetLoad	HS_Diagnosis.Lib
	版本号诊断	HS_GetVersion	
	符号表诊断	HS_SymbolTableDiag	
	SDRAM 实际使用量诊断	HS_SDRAM_Diag	
	Flash 使用量诊断	HS_FlashDiag	
	工作模式诊断	HS_WorkModeDiag	
	看门狗诊断	HS_DIAG_WatchdogDiag	
	任务运行状况诊断	HS_IECTaskDiag	
	双机运行状态诊断	HS_CPU_ReduDiag	
	电池电量状况诊断	HS_BatterAlarm	
	本地总线诊断	HS_LocalBusSlaveDiag	
	DP 主站诊断	HS_DPMasterDiag	
	DP 从站标准诊断	HS_DPSlaveStdDiag	

	DP 从站用户扩展诊断	HS_DPSlaveAlarm	
修改 IP 地址指令	修改 IP 地址	HS_SetIPAddress	HS_SetIPAddress.Lib
PID 调节控制器指令	PID 调节控制器	HS_PID	HS_PIDController.Lib
LK850 量程转换	LK850 模拟输入量程转换	HS_LK850_AI	HS_LK850_Convert.Lib
	LK850 模拟输出量程转换	HS_LK850_AO	
保持型定时器指令	保持型通电延时定时器	HS_TONR	HS_Timer.Lib

➢ 2、IEC 标准指令表

LK 大型 PLC 编程软件 PowerPro 遵循国际电工技术委员会（IEC）的 IEC61131-3 标准，在这个标准中明确规定了作为可编程控制器必须具有的指令，即标准指令。标准指令包括类型转换指令、数字运算指令、位移位指令、比较指令、类型转换指令、定时器、计数器等，所有 IEC61131-3 标准规定指令，见下表。

IEC 标 准 指 令		
指令类型	**指令说明**	**指令名**
算术运算指令	加法指令	ADD
	乘法指令	MUL
	减法指令	SUB
	除法指令	DIV
	取余指令	MOD
逻辑指令	与指令	AND
	或指令	OR
	异或指令	XOR
	取非指令	NOT
移位指令	左移指令	SHL
	右移指令	SHR
	循环左移指令	ROL
	循环右移指令	ROR
选择指令	二选一指令	SEL
	取最大值指令	MAX
	取最小值指令	MIN
	极限值指令	LIMIT
	多选一指令	MUX
比较指令	大于指令	GT
	小于指令	LT
	大于等于指令	GE
	小于等于指令	LE
	等于指令	EQ

	不等于指令	NE
类型转换指令	布尔类型转换指令	BOOL_TO_<TYPE>
	字节类型转换指令	BYTE_TO_<TYPE>
	日期类型转换指令	DATE_TO_<TYPE>
	长整型转换指令	DINT_TO_<TYPE>
	日期时间类型转换指令	DT_TO_<TYPE>
	双字类型转换指令	DWORD_TO_<TYPE>
	整数类型转换指令	INT_TO_<TYPE>
	字类型转换指令	WORD_TO_<TYPE>
	实数类型转换指令	REAL_TO_<TYPE>
	短整型转换指令	SINT_TO_<TYPE>
	字符类型转换指令	STRING_TO_<TYPE>
	时钟类型转换指令	TIME_TO_<TYPE>
	时间类型转换指令	TOD_TO_<TYPE>
	无符号长整型转换指令	UDINT_TO_<TYPE>
	无符号整型转换指令	UINT_TO_<TYPE>
	无符号短整型转换指令	USINT_TO_<TYPE>
	截短转换指令	TRUNC
初等数学运算指令	绝对值指令	ABS
	平方根指令	SQRT
	自然对数指令	LN
	常用对数指令	LOG
	指数指令	EXP
	正弦指令	SIN
	余弦指令	COS
	正切指令	TAN
	反正弦指令	ASIN
	反余弦指令	ACOS
	反正切指令	ATAN
	幂指令	EXPT
地址运算指令	取地址指令	ADR
	取地址内容指令	^
	位地址指令	BITADR
	索引指令	INDEXOF
	数据类型大小指令	SIZEOF
调用指令	调用运算指令	CAL
初始化操作指令	初始化操作指令	INI
字符串指令	结合字符串指令	CONCAT
	删除字符指令	DELETE
	查找字符串指令	FIND
	插入字符串指令	INSERT

	左边取字符串指令	LEFT
	字符串长度指令	LEN
	中间取字符串指令	MID
	替换字符串指令	REPLACE
	右边取字符串指令	RIGHT
BCD 转换指令	BCD 码转整型指令	BCD_TO_INT
	整型转 BCD 码指令	INT_TO_BCD
双稳态指令	置位优先双稳态器	SR
	复位优先双稳态器	RS
触发器	上升沿检测触发器	R_TRIG
	下降沿检测触发器	F_TRIG
计数器	递增计数器	CTU
	递减计数器	CTD
	递增递减计数器	CTUD
定时器	普通定时器	TP
	通电延时定时器	TON
	断电延时定时器	TOF
	实时时钟	RTC

➢ 3、主控型号的高低顺序及其与 PowerPro V4 的适用关系

LK 的各类主控型号的高低顺序：

- ✧ LK202 的型号由低到高：LK202-A01、LK202-B01；
- ✧ LK205 的型号由低到高：LK205-A01、LK205-B01；
- ✧ LK207 的型号由低到高：LK207-A02、LK207-A03、LK205-A04；
- ✧ LK209 的型号由低到高：LK209-A01、LK209-B01；
- ✧ LK210 的型号由低到高：LK210-A02、LK210-B02、LK210-B03、LK210-B04、LK210-B05、LK210-B06。

以下从库文件和 Target 双重考虑的 PowerPro V4 与主控型号适用关系：

1) **PowerPro V4.0.0B**：适用于 LK210-A02、LK210-B02、LK210-B03 型主控。

2) **PowerPro V4.1.0B**：适用于 LK210-A02、LK210-B02、LK210-B03、LK210-B04 型主控。当用于 LK210-A02、LK210-B02、LK210-B03 型主控时，请将 PowerPro 库文件的默认路径设置成“安装路径\Backup Library\V4.0.0B”，如图 1 所示；并且添加该路径下的库文件，如图 2 所示。

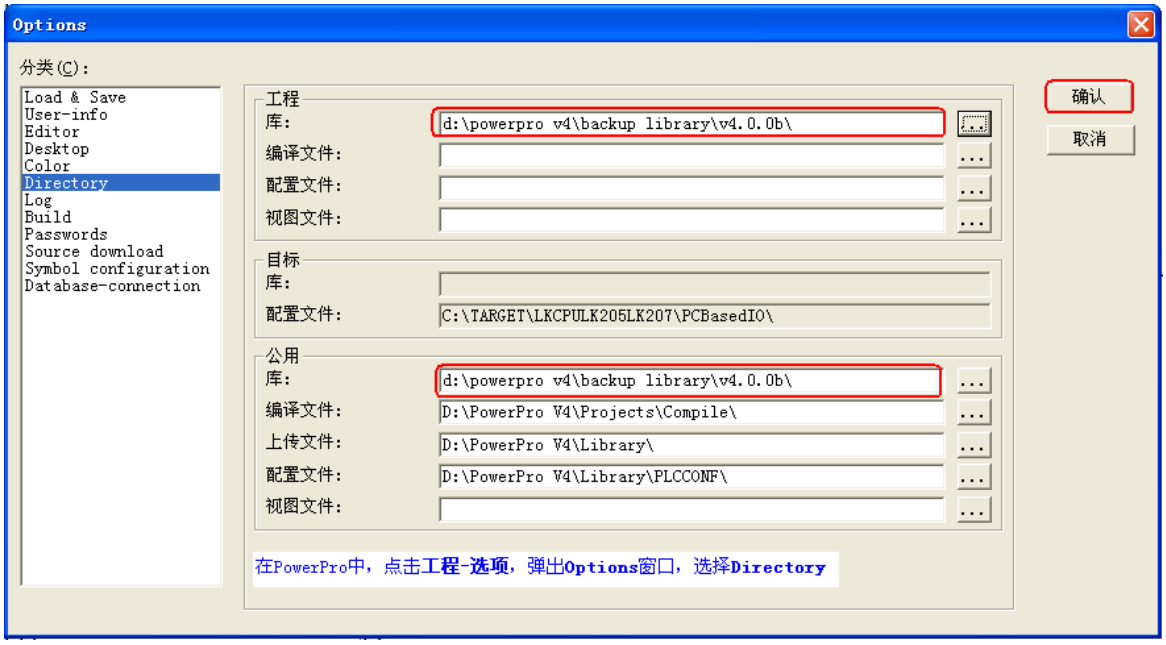


图 1. 设置默认的库路径为 V4.0.0B

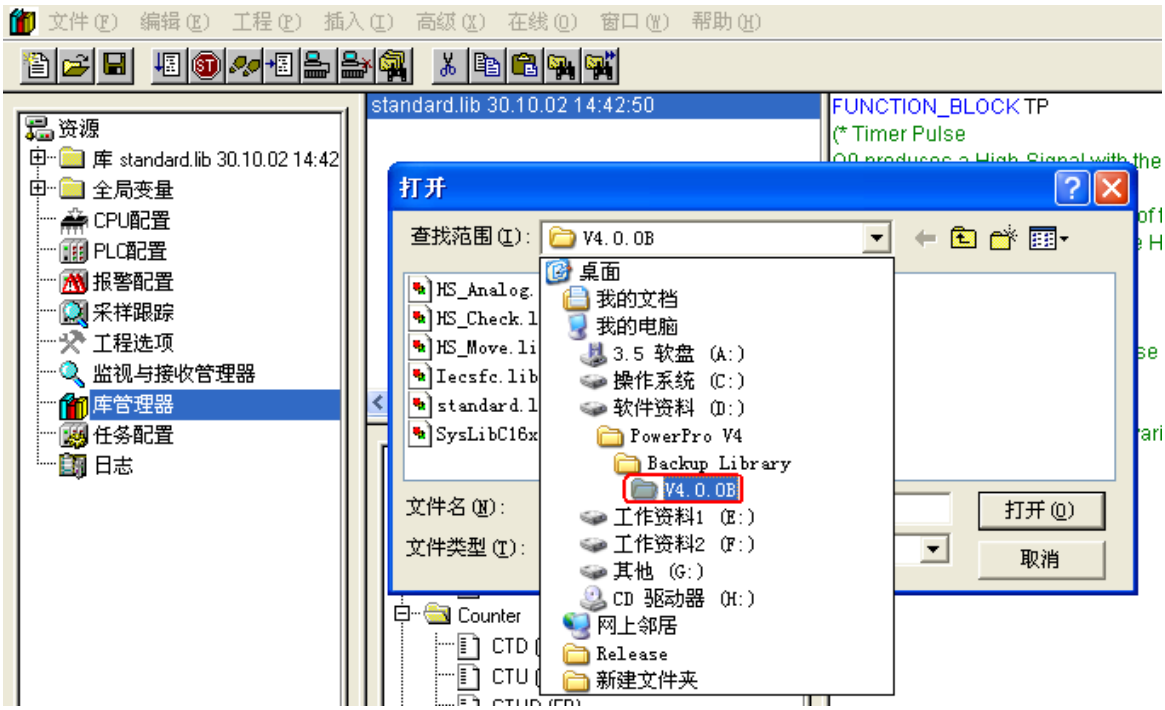


图 2 添加“安装路径\Backup Library\V4.0.0B”目录下的库文件

3) **PowerPro V4.2.0B**：适用于 LK207-A02 型主控。

4) **PowerPro V4.3.0B**：适用于 LK202-A01、LK205-A01、LK207-A02、LK207-A03、LK209-A01、LK210-B05 型主控。当用于 LK207-A02 型主控时，请将 PowerPro 库文件的默认路径设置成“安装路径\Backup Library\V4.2.0B”；并且添加该路径下的库文件。

5) **PowerPro V4.3.1B**：适用于 LK202-A01、LK202-B01、LK205-A01、LK205-B01、LK207-A02、LK207-A03、 LK207-A04、LK209-A01、LK209-B01、LK210-B05、LK210-B06 型主控。当用于 LK202-A01、LK205-A01、LK207-A03、 LK209-A01、LK210-B05 型主控时，请将 PowerPro 库文件的默认路径设置成“安装路径\Backup Library\V4.3.0B”，并且添加该路径下的库文件；当用于 LK207-A02 型主控时，请将，请将 PowerPro 库文件的默认路径设置成“安装路径\Backup Library\V4.2.0B”，并且添加该路径下的库文件。

请根据所使用主控的型号，来选择合适的库文件版本。

➢ 4、LK 的标识符与 Modbus 地址对应关系表

标识符	类型	地址范围	Modbus 地址	映射公式	X(寄存器类型)选择
%QW	WORD	QW0，QW1，…	X0001，X0002，… X65535	QWm：m+1	只读，X 选 3 读写，X 选 4
%IW	WORD	IW0，IW1，…	X0001，X0002，… X65535	IWm：m+1	只读，X 选 3
%MW	WORD	MW0，MW1，…	X5001，X5002，… X65535	MWm：m+5000+1	只读，X 选 3 读写，X 选 4
%MD	DWORD、REAL	MD0，MD1，…	X5001，X5002，… X65534	MDm：m*2+5000+1	只读，X 选 8 读写，X 选 9
%QX (LK710)	BOOL	QX0.0,…QX0.15, QX1.0,…QX1.15，…	X0001，…X0016 X0017,… X0023, …X65535	QXm.n：m*16+n+1	只读，X 选 1 读写，X 选 0
%MX	BOOL	MX0.0,…MX0.15 MX1.0,…MX1.15 …	X5001，…X5016 X5017,… X5023, …X65535	MXm.n：m*16+n+5000+1	只读，X 选 1 读写，X 选 0