HCFa
— 禾川科技 —

ATC 功能库
使用手册

目录

1. 前言

本手册是对 ATC 库中包含的功能块的详细说明书。请对相关功能、操作方法等进行充分理解，正确使用功能模块。

此外，阅读后请将本手册妥善保管于易取处。

读者对象

禾川 ATC 库功能块的使用者，可以从本说明书中对该库中的功能块、操作方式进行充分理解，并掌握使用方法。

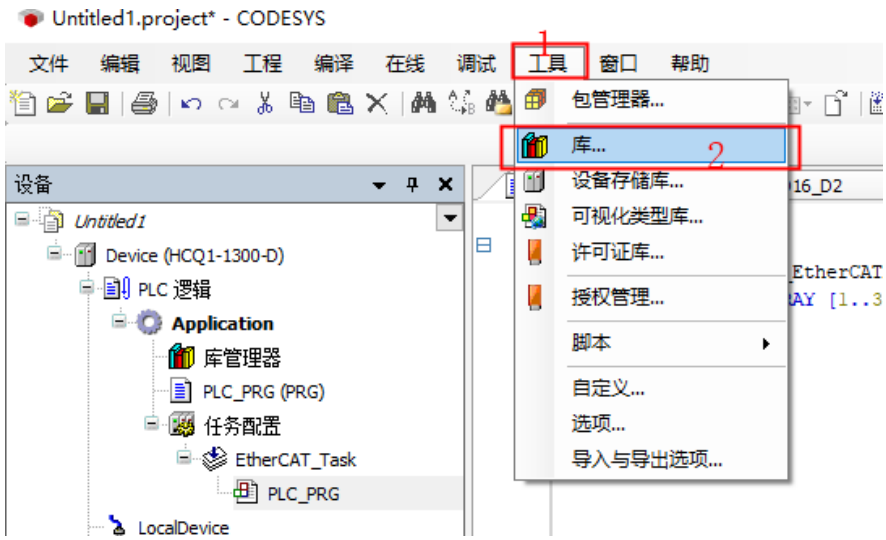
更新记录

版本	更新时间	更新内容
V0.9	20240320	同步 HCFA_ATCLib_1.15.14.compiled-library 库的功能说明
V0.10	20240402	同步 HCFA_ATCLib_1.15.16.compiled-library 库的功能说明

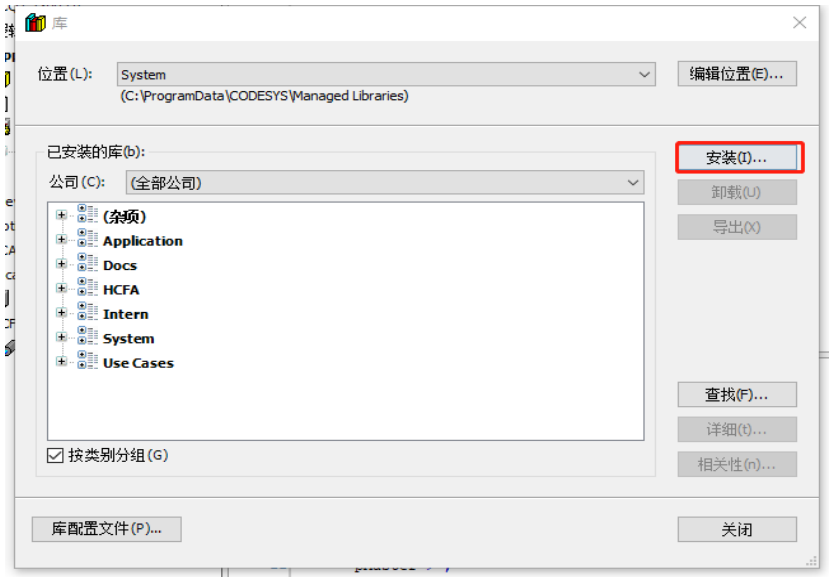
2. 库文件添加

本手册对 HCFA_ATCLib_1.8.5 库中所包含功能进行详细说明，使用库文件中所含功能之前，请按照如下步骤在 CODESYS 中安装相应库文件。

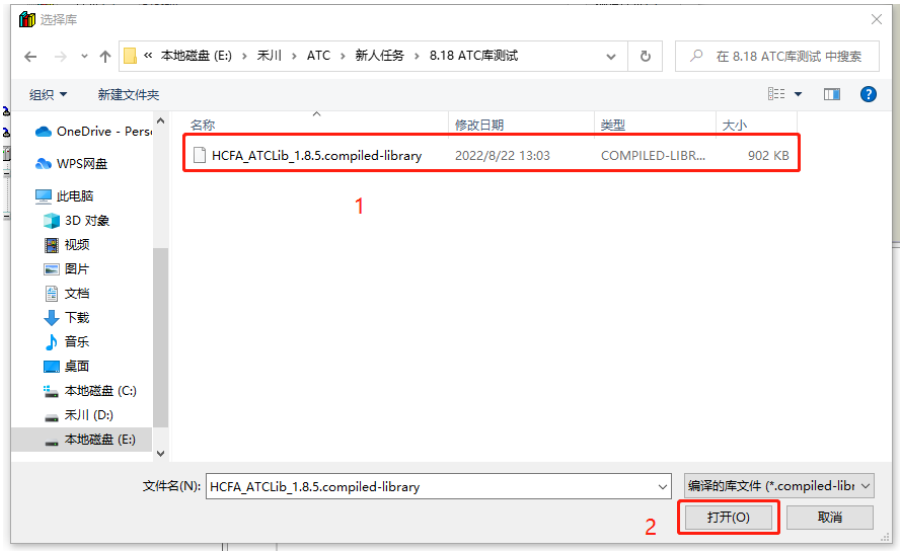
【1】在 CODESYS 中点击【工具】->【库】



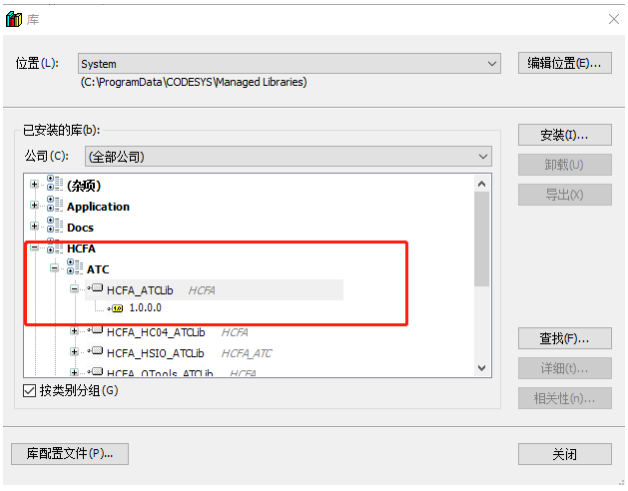
【2】在弹出的“库”窗口中，点击【安装】

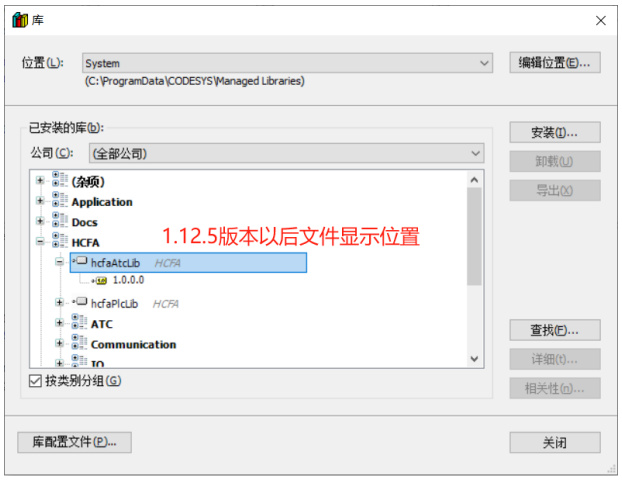


【3】在弹出的系统对话框中选择对应版本的库文件，此处为 HCFA_ATCLib_1.8.5,点击【打开】按钮安装库文件。



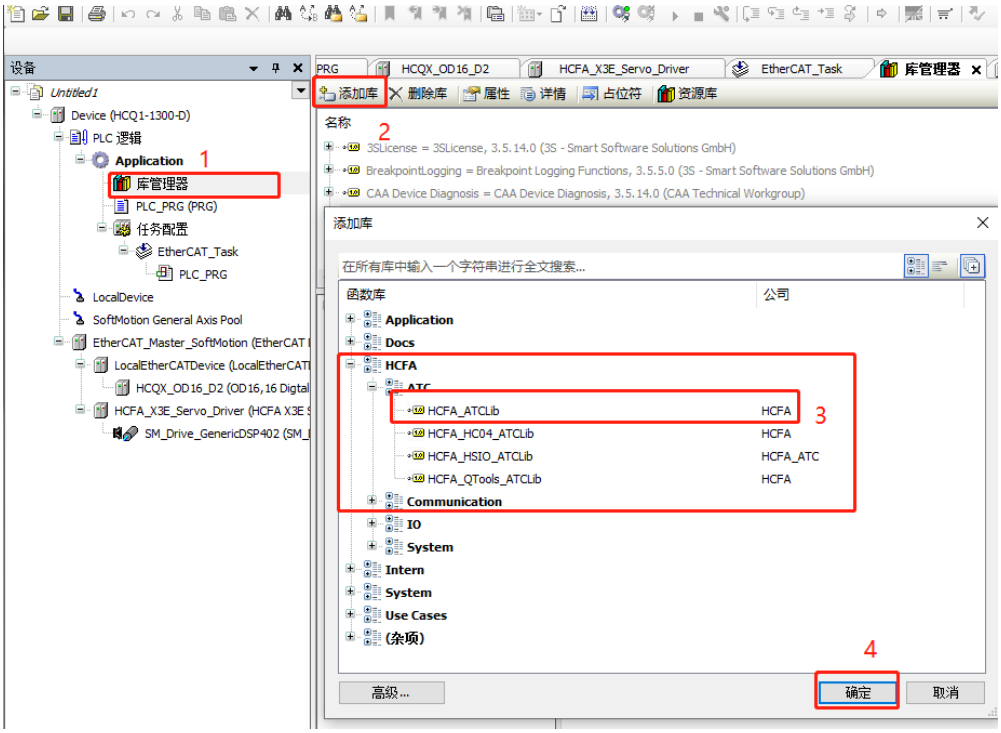
【4】安装成功后，在“库”窗口中【HCFA】->【ATC】可以看到 HCFA_ATCLib 的库，版本号为 1.0.0.0。在 1.12.5 版本以后的库，文件位置将变更为【HCFA】下，hcfaAtcLib。



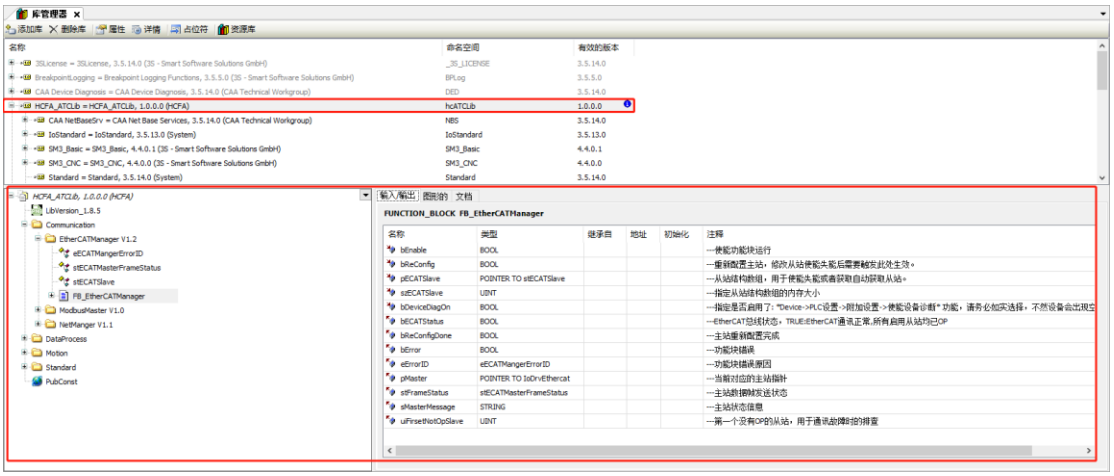


【5】在 CODESYS 界面，双击右侧设备树中的【库管理器】->【添加库】，在弹出的“添加库”窗口中，打开【显示高级库】选项，依次点击【HCFA】->【ATC】->【HCFA_ATCLib】选择刚刚安

装的库，点击确定完成添加。对于 1.12.5 版本以后的库，添加位置如【4】中图描述变为直接在【HCFA】路径下。



【6】完成添加后，可以看到库管理器中完成了对库 HCFA_ATCLib 1.0.0.0(1.12.5 以后名称变更为 hcfaAtcLib 1.0.0.0)的添加，点击可查看详细版本和各个功能块的简略说明。



3. Communication (通讯管理)

3.1. FB_EtherCATManager（FB）

用于实现 EtherCAT 主站网络管理，可以控制从站是否启用，并附带一些简易的诊断功能。

变量

名称	FB_EtherCATManager（通信管理）		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	


```
FB_EtherCATManager(
    bEnable:= ,
    bReConfig:= ,
    pECATSlave:=  ,
    szECATSlave:= ,
    bDeviceDiagOn:= ,
    bECATStatus=> ,
    bAllSlaveOp=> ,
    bReConfigDone=> ,
    bError=> ,
    eErrorID=> ,
    pMaster=> ,
    stFrameStatus=> ,
    sMasterMessage=> ,
    uiFirstNotOpSlave=> );
```

（1） 输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bEnable	使能	BOOL	TRUE、FALSE	FALSE	TRUE：启用功能块 FALSE：关闭功能块
bReConfig	重新配置主站	BOOL	TRUE、FALSE	FALSE	TRUE： 重新配置主站
pECATSlave	从站结构数组	POINT TO stECATSlave			用于使能失能或者自动获取从站
szECATSlave	内存大小	UINT		0	指定从站数组结构的内存大小
bDeviceDiagOn	诊断功能	BOOL	TRUE、FALSE	FALSE	指定是否开启诊断功能

（2） 输出变量

输出变量	名称	数据类型	有效范围	内容
bECATStatus	EtherCAT 总线状态	BOOL	TRUE、FALSE	TRUE:EtherCAT 通讯正常，DC 时

				钟同步成功
bAllSlaveOp	全部从站设备 op	BOOL	TRUE、FALSE	TRUE:全部从站设备进入 op 状态
bReConfigDone	主站重新配置完成	BOOL	TRUE、FALSE	TRUE:主站重新配置完成
bError	错误	BOOL	TRUE、FALSE	TRUE:功能块产生异常，已停止执行
ErrorID	错误代码	SMC_ERROR	0	功能块产生异常时输出对应的故障码
pMaster	主站	POINTER TO IoDrivEthercat		当前对应的主站指针
stFrameStatus	发送状态	stECATMaster FrameStatus		主站数据帧发送状态
sMasterMessage	信息	STRING		主站状态信息
uiFirsetNotOpSlave	异常从站 ID	UINT	0	第一个没有 OP 的从站，用于通讯故障时的排查

（3） 输出变量的转化时序

变量	变为 TRUE 时	变为 FALSE 时
bECATStatus	✧ ECAT 总线通讯正常，DC 时钟同步完成	✧ bError 变为 TRUE ✧ uiFirsetNotOpSlave 输出异常从站号
bAllSlaveOp	✧ 所有从站进入 op 状态	✧ 主站重新配置
bReConfigDone	✧ 主站重新配置完成	✧ 主站重新配置
bError	✧ 功能块执行中产生故障时	✧ 解除了异常时

数据类型

（4） 从站结构体

名称	数据类型	有效范围	初始值	内容
bDisableSlave	BOOL	TRUE、FALSE	FALSE	TRUE：失能此从站
pSlave	POINT TO stECATSlave			自动获取的从站指针
PAxis	POINT TO AXIS_REF_SM3			启用诊断状态下

要点说明

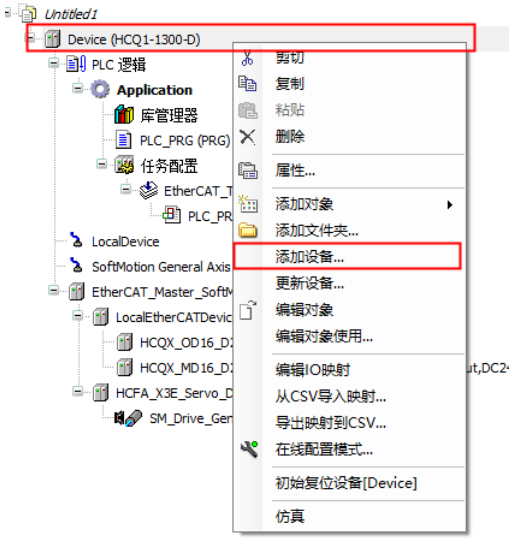
- 将 bEnable 设为 TRUE 后，FB_EtherCATManager 进入执行状态，此时其他的输入引脚被 FB_EtherCATManager 处理。
- 将 Enable 设为 FALSE 后，FB_EtherCATManager 将不会再执行任何功能块，此时修改 FB_EtherCATManager 其他的输入引脚将不会产生任何效果。
- bReConfig 引脚控制重新配置主站，对从站结构体完成使能失能的修改后，需要触发此引脚生效。

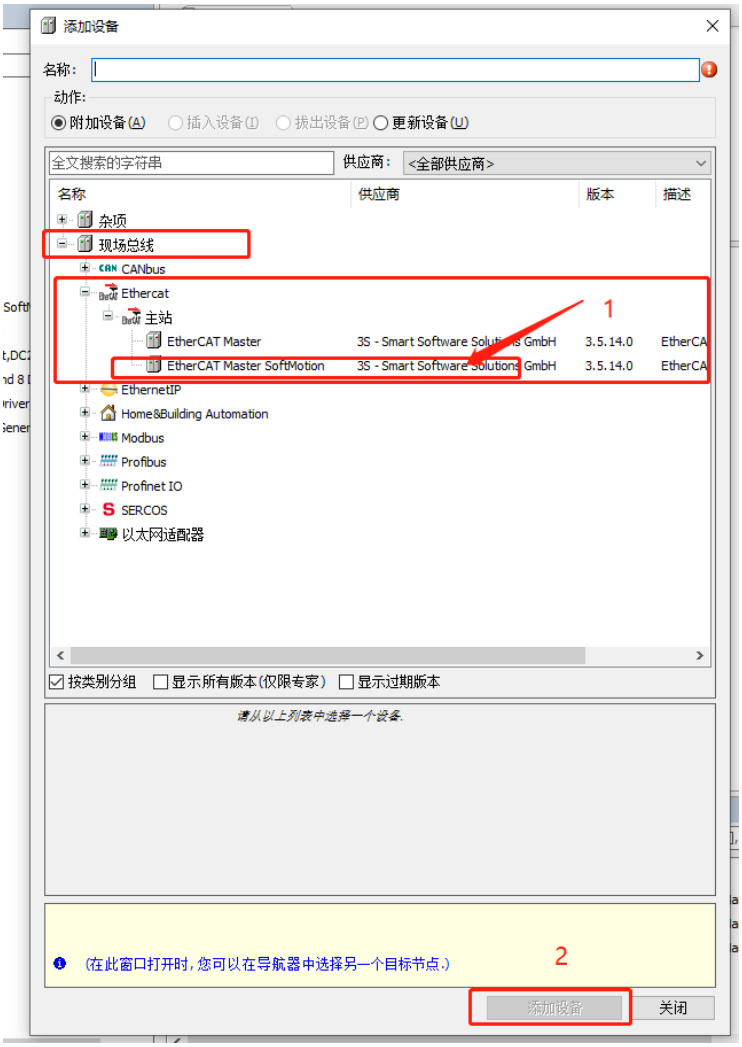
- pECATSlave（从站结构体数组首地址的指针），指向类型为 stECATSlave 的结构体数组，在触发功能块的 bEnable 引脚时，自动获取从站以及轴的指针。
- bDeviceDiagOn（诊断功能），指定是否启用了：*Device->PLC 设置->附加设置->使能设备诊断*功能，请务必如实选择。在开启诊断功能并且该引脚置 True 时，可以自动获取伺服下轴的指针，也可以对轴进行使能失能操作。

3.1.1. 使用举例（诊断模式）

【1】添加 EtherCat 主站。

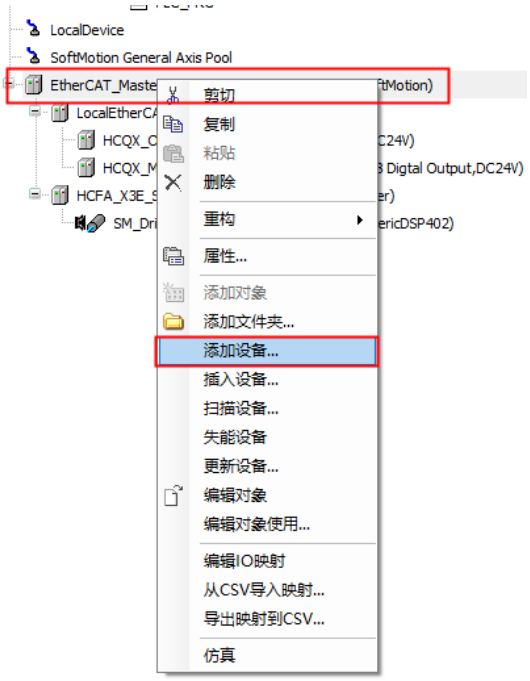
右键【Device】->【添加设备】，在弹出的“添加设备”窗口中，依次点击【现场总线】->【Ethercat】->【主站】->【EtherCAT Master SoftMotion】->【添加设备】完成主站的添加

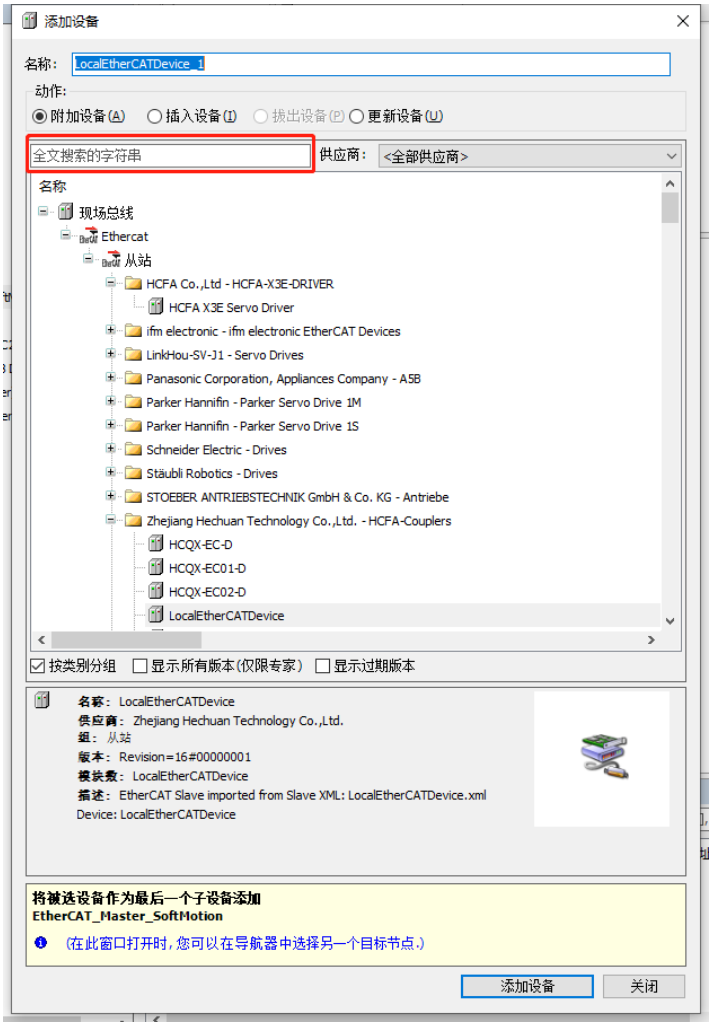




【2】添加 EtherCAT 从站。

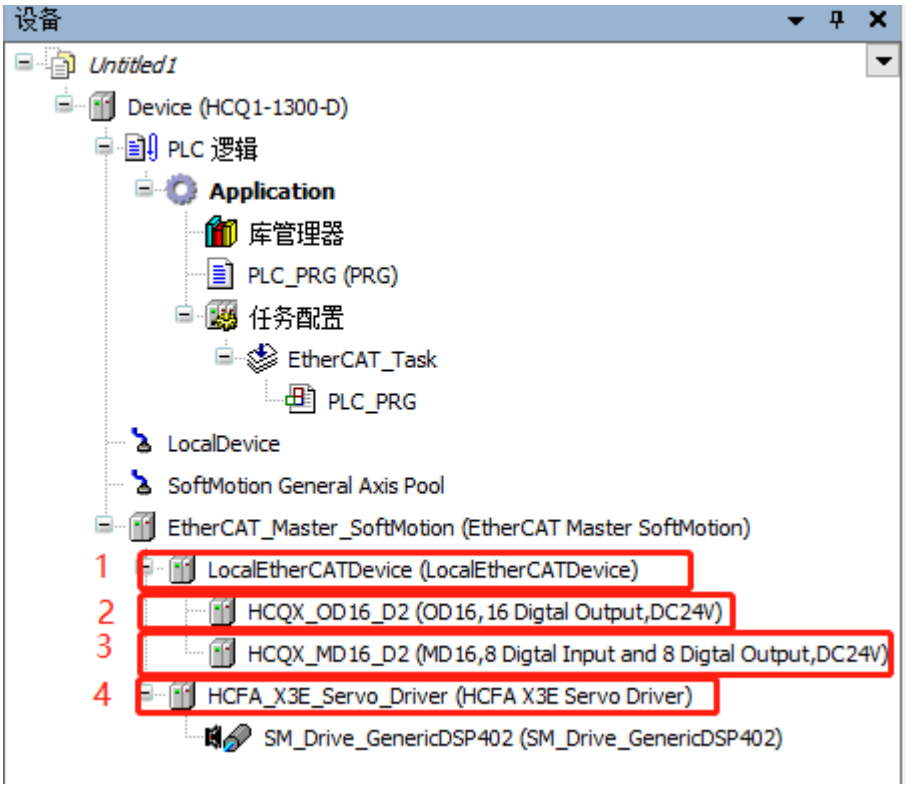
在左侧设备树右键【EtherCAT_Master_SoftMotion】->【添加设备】，在弹出的“添加设备”窗口中安装实际连接情况依次添加对应的设备，可以直接使用搜索框搜索对应设备完成添加。





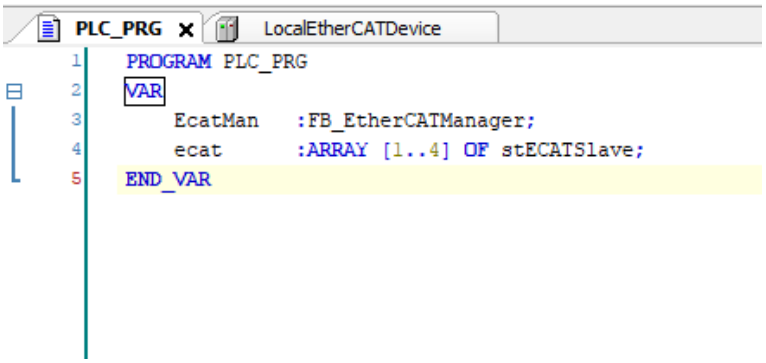
或者，登陆到 PLC 后，右键【EtherCAT_Master_SoftMotion】->【扫描设备】，待扫描完成后，电机【将全部设备复制到工程中】，即可完成对现有连设备的添加。

此处共有四个从站设备（注：LocalEtherCATDevice 也是从站设备）。



【3】声明功能块 FB_EtherCATManager 和从站结构体数组。

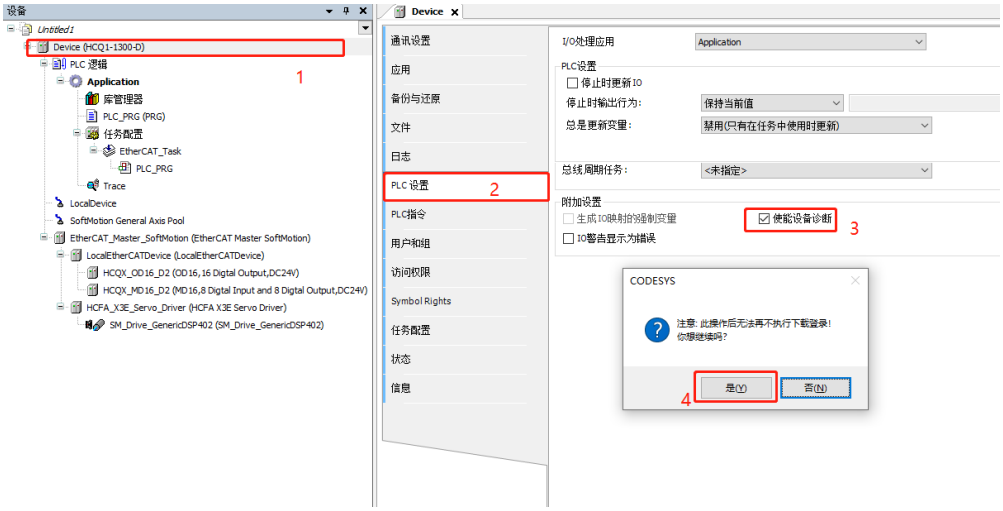
注：其中数值大小可以视实际情况而



【4】功能块调用，从站结构体数组赋值到对应引脚。

```
EcatMan(
    bEnable:= ,
    bReConfig:= ,
    pECATSlave:= ADR(ecat),
    szECATSlave:= SIZEOF(ecat),
    bDeviceDiagOn:= ,
    bECATStatus=> ,
    bAllSlaveOp=> ,
    bReConfigDone=> ,
    bError=> ,
    eErrorID=> ,
    pMaster=> ,
    stFrameStatus=> ,
    sMasterMessage=> ,
    uiFirstNotOpSlave=> );
```

【5】打开设备的诊断模式，在左侧设备树双击【Device】，选择【PLC 设置】选项卡，将“附加设置”，勾选【使能设备诊断】，在弹出的窗口中点击【确定】。

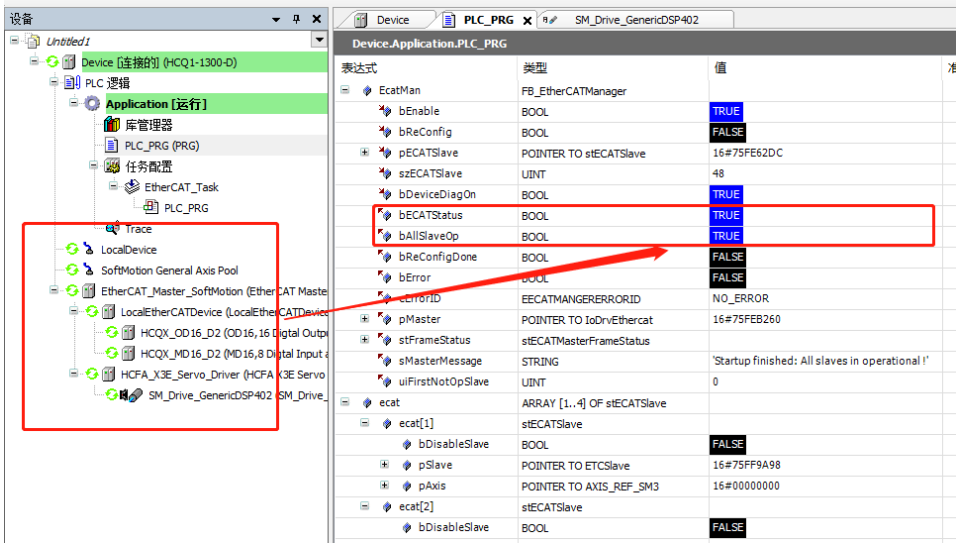


【6】登陆到 PLC，并启动程序

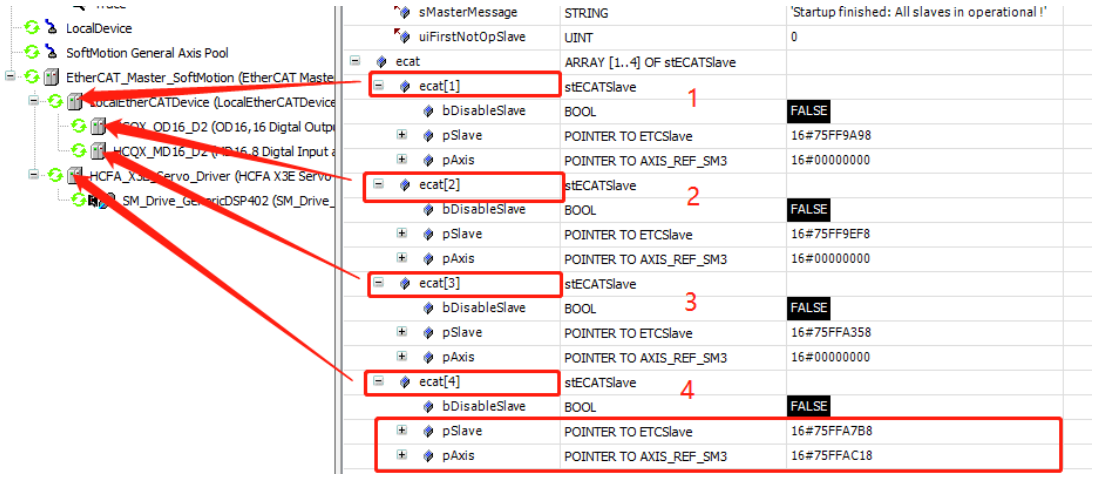
【7】诊断模式下，需要将 bDeviceDialog 与 bEnable 同时置 True，才可以正确开启诊断模式，并且获得从站的轴指针。若 bDeviceDialog 引脚滞后于 bEnable 引脚置 True，则实际功能块为非诊断模式，无法获取轴指针。

表达式	类型	值	准备值
EcatMan	FB_EtherCATManager		
bEnable	BOOL	FALSE	TRUE
bReConfig	BOOL	FALSE	
pECATSlave	POINTER TO stECATSlave	16#00000000	
szECATSlave	UINT	0	
bDeviceDiagOn	BOOL	FALSE	TRUE
bECATStatus	BOOL	FALSE	
bAllSlaveOp	BOOL	FALSE	
bReConfigDone	BOOL	FALSE	
bError	BOOL	FALSE	
eErrorID	EECATMANGERERRORID	NO_ERROR	
pMaster	POINTER TO IoDrvEthercat	16#00000000	
stFrameStatus	stECATMasterFrameStatus		
sMasterMessage	STRING	''	
uiFirstNotOpSlave	UINT	0	

在功能块使能并且 ECAT 通讯完成，且全部从站都为 TRUE 时，bECATStatus 和 bALLSlaveOp 引脚都输出 True。

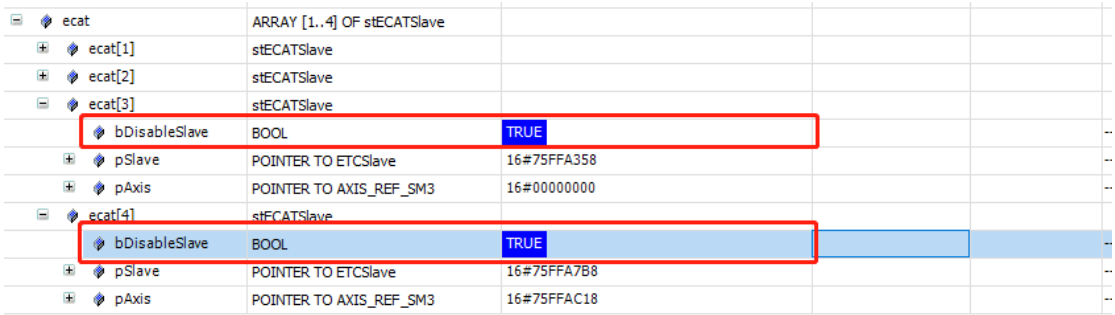


同时，在定义的结构体数组中，可以得到从站的指针和轴指针，数组按照顺序对应从站。

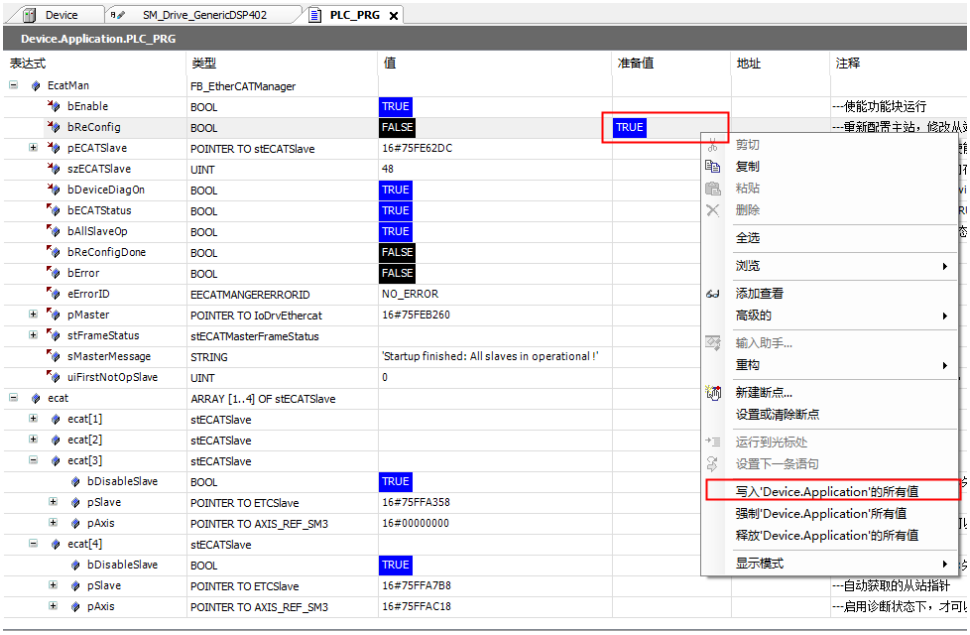


【8】失能从站

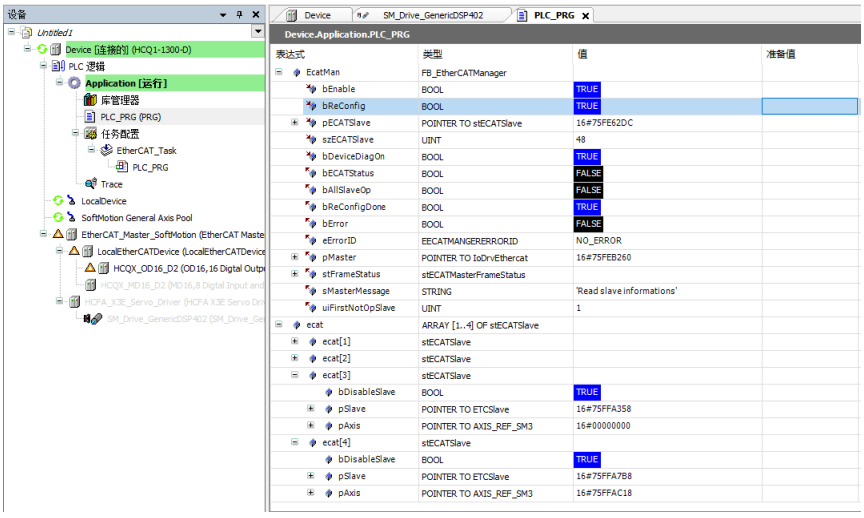
在对应的从站结构体中，通过 bDisableSlave 引脚使能和使能功能块，FALSE 为使能，TRUE 为失能，此处演示选择失能从站模块 HCQX_MD_D2 和从站伺服 HCFA_X3E_Servo_Driver，即此处的从站 3-4。



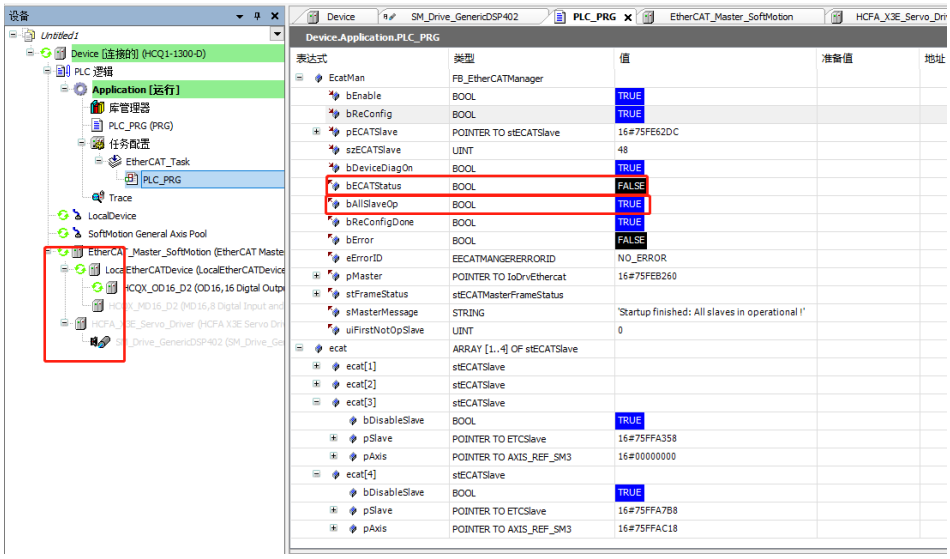
在完成从站结构体的引脚修改后，还需要通过触发 FB_EtherCATManger 的 bReConfig 引脚使得修改生效。



在触发重新配置主站引脚后，可以看到左侧设备树中从站全部重启，并且 3-4 号从站失能：



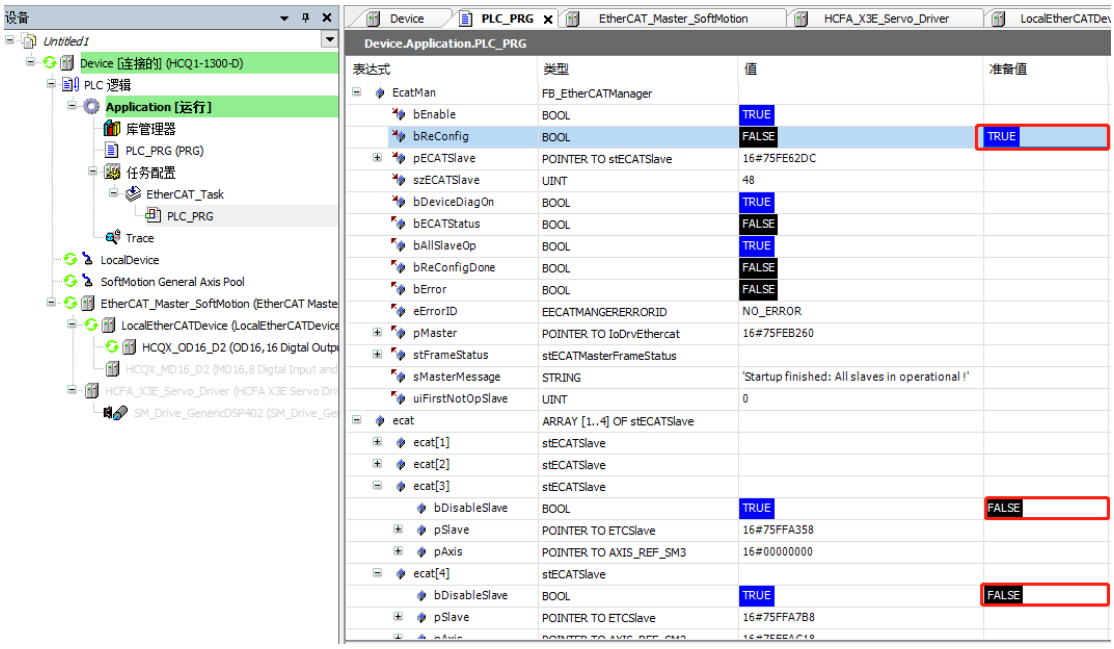
同时 FB_EtherCATManger 功能块的 bECATStatus 和 bALLSlaveOp 引脚输出变为为 Flase，待从站模块完成 ECAT 通讯后，bALLSlaveOp 引脚输出为 True，但由于从站伺服失能，且本地 local 设备为无 DC 设备，则 bECATStatus 仍旧输出为 FALSE。



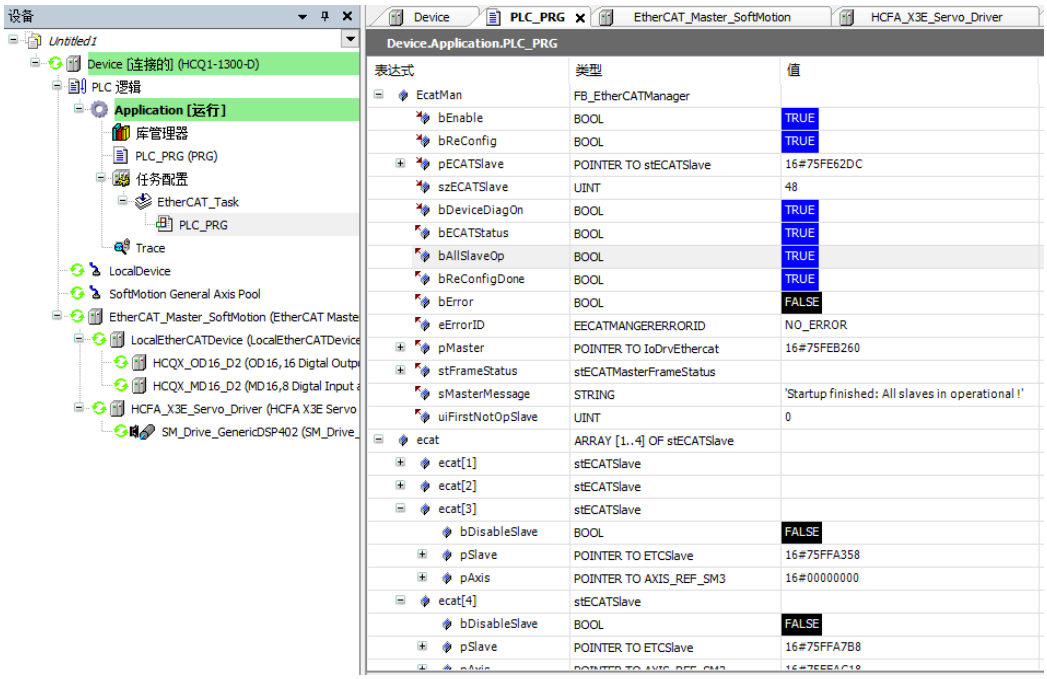
【8】使能从站

操作方式同失能从站相似，将对应的从站结构体中的 bDisableSlave 置为 FALSE，再次触发 FB_EtherCATManger 的 bReConfig 引脚使得修改生效，全部从站重启，同时对应从站恢复使能，待从站模块和从站伺服全都通讯完成时，FB_EtherCATManger 功能块的 bECATStatus 和 bALLSlaveOp 引脚输出变为为 True，但轴错误（SMC_DI_GENERAL_COMMUNICATION_ERROR）为通讯错误，不会恢复，需要另外调用功能块 SMC3_ReinitDrive 复位轴，此处举例：

```
17  FB_ReinitDrive(
18      Axis:= SM_Drive_GenericDSP402,
19      bExecute:= ,
20      bVirtual:= ,
21      bDone=> ,
22      bBusy=> ,
23      bError=> ,
24      nErrorID=> );
25
26  IF EcatMan.bECATStatus AND SM_Drive_GenericDSP402.nAxisState = 1 THEN
27      FB_ReinitDrive.bExecute:=TRUE;
28  ELSE
29      FB_ReinitDrive.bExecute:=FALSE;
30  END_IF
```



可以看到伺服从站全部重启，并且在通讯完成后自动触发 SMC3_ReinitDrive 功能块，将轴初始化。

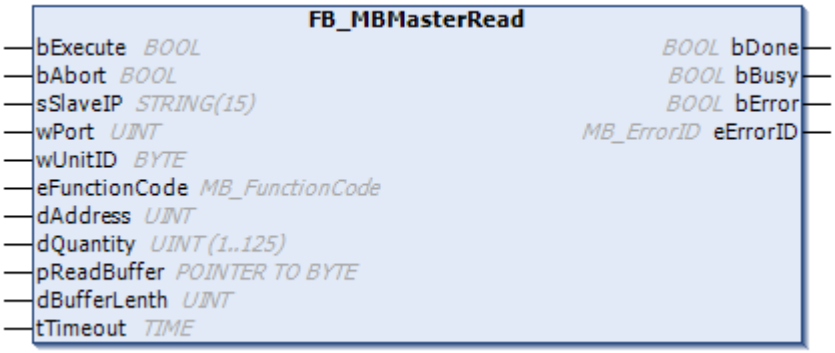


3.2. FB_MBMasterRead (FB)

ModbusTCP 主站读写功能块，无需设置设备树配置可直接通过功能块指定的 IP 和端口连接从站进行数据读操作。

变量

名称	FB_MBMasterRead（主站 MB 读功能块）		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	




```
FB_MBMasterRead (
    bExecute:= ,
    bAbort:= ,
    sSlaveIP:= ,
    wPort:= ,
    wUnitID:= ,
    eFunctionCode:= ,
    dAddress:= ,
    dQuantity:= ,
    pReadBuffer:= ,
    dBufferLenth:= ,
    tTimeout:= ,
    bDone=> ,
    bBusy=> ,
    bError=> ,
    eErrorID=> );
```

（1）输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bExecute	使能	BOOL	TRUE、FALSE	FALSE	TRUE：启用功能块 FALSE：关闭功能块
bAbort	终止	BOOL	TRUE、FALSE	FALSE	TRUE： 终止功能块运行
sSlaveIP	从站 IP 地址	STRING（15）			从站 IP
wPort	端口	UINT	0，正数	502	从站端口
wUnitID	站号	BYTE	0，正数		从站站号
eFunctionCode	功能码	MB_FunctionCode			Modbus 功能码
dAddress	起始地址	UINT			读取地址
dQuantity	读取数量	UINT（1..125）			读取数量，最大 125 个 word

pReadBuffer	缓冲区	POINTER TO BYTE			读取数据缓冲区
dBufferLenth	缓冲区大小	UINT			读取数据缓冲区大小单位 word
tTimeout	超时	TIME		T#50s0ms	超时时间

（2）输出变量

输出变量	名称	数据类型	有效范围	内容
bDone	完成	BOOL	TRUE、FALSE	TRUE:功能块执行完成
bBusy	运行中	BOOL	TRUE、FALSE	TRUE:功能块执行中，TCP 正常连接
bError	错误	BOOL	TRUE、FALSE	TRUE:功能块产生异常，已停止执行
eErrorID	错误代码	SMC_ERROR	0	功能块产生异常时输出对应的故障码

（3）输出变量的转化时序

变量	变为 TRUE 时	变为 FALSE 时
bDone	✧ 功能块执行完成时	✧ 功能块未完成
bBusy	✧ 功能块运行中且无错误	✧
bError	✧ 功能块产生异常	✧ 功能块无错误

要点说明

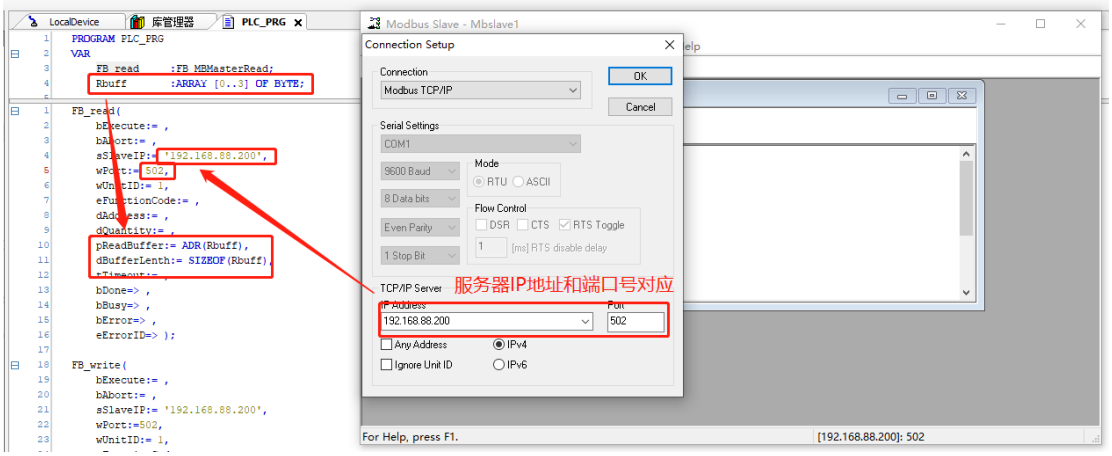
- 将 bExecute 设为 TRUE 后，FB_MBMasterRead 进入执行状态，此时其他的输入引脚被 FB_MBMasterRead 处理。
- 将 Enable 设为 FALSE 后，FB_MBMasterRead 将不会再执行任何功能块，此时修改 FB_MBMasterRead 其他的输入引脚将不会产生任何效果。
- bAbort 引脚用于终止功能块运行，bAbort 置 True 时，输出引脚全部变为 FALSE，eErrorID 信息也会清除，数据接收缓存区清空，需要将 bAbort 引脚重新置 FALSE 后功能块才可以再次被触发，否则无效。
- pReadBuffer 引脚为指向 BYTE 数组首地址的指针，应该根据具体想要读取的内存类型设置数组长度，例如读取保持型寄存器，每个寄存器需要两个 BYTE 的空间存放数据。
- 此处 pReadBuffer 读取数据缓存区在读取到数据后，数据存放位置为反向，例如数组 arr【0..1】BYTE 读取到寄存器的数据为 0002，实际上存放方式为 arr【0】：02，arr【1】：00。

3.2.1. 使用举例（Q1 做主站，客户端模式）

【1】主站 Q1 设置

Q1 主站声明并调用 FB_MBMasterRead 功能块，声明一个 BYTE 类型的数组 Rbuff 用作数据接收缓冲区，无需在设备树中作其他操作。

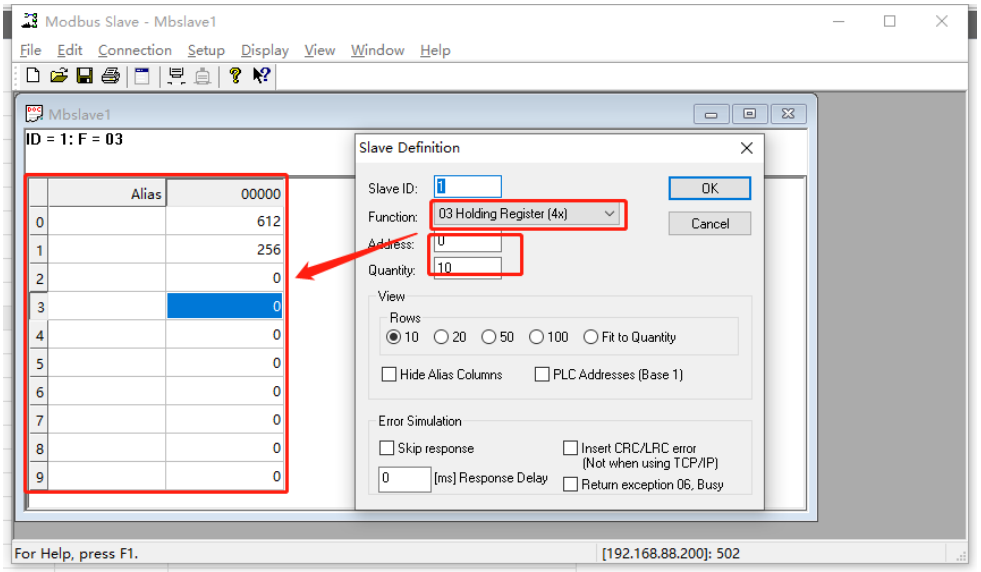
功能块引脚的具体设置如下所示



此处通过主站 Q1 的 Port2 连接上位电脑 ModbusTcp 服务端工具实现通讯。电脑网口 IP 地址为 192.168.88.200，端口号设置为 502。

【2】上位工具服务端设置

修改服务器端数据类型为保持型寄存器，起始地址为 0，长度为 10，即存在 0-9 共 10 个寄存器，方便进行修改操作。此处修改寄存器地址 0 的值为 612 即 16#0264，修改地址 1 寄存器为 256 即 16#0010。

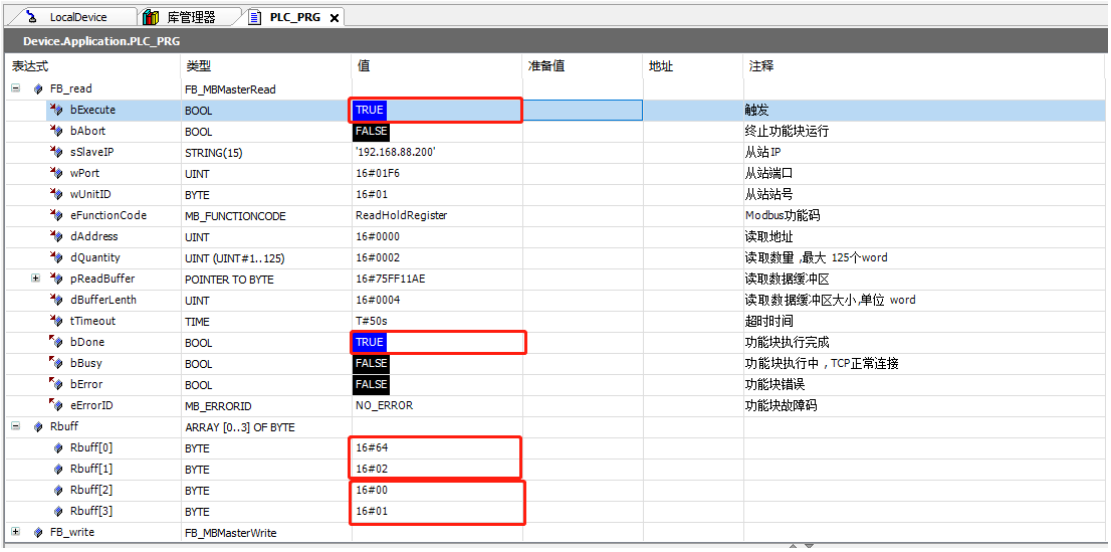


【3】读取数据

将程序下载登陆到主站 Q1 设备，设置功能码，读取寄存器目标其实地址，读取长度。此处演示读取地址为 0-1 两个寄存器数据。

表达式	类型	值	准备值	地址	注释
FB_read	FB_MBMasterRead				
bExecute	BOOL	FALSE			触发
bAbort	BOOL	FALSE			终止功能块运行
sSlaveIP	STRING(15)	'192.168.88.200'			从站IP
wPort	UINT	16#01F6			从站端口
wUnitID	BYTE	16#01			从站站号
eFunctionCode	MB_FUNCTIONCODE	ReadCoils	ReadHoldRegister		Modbus功能码
dAddress	UINT	16#0000	16#0000		读取地址
dQuantity	UINT (UINT#1..125)	16#0001	16#0002		读取数量 ,最大 125个word
pReadBuffer	POINTER TO BYTE	16#75FF11AC			读取数据缓冲区
dBufferLenth	UINT	16#0004			读取数据缓冲区大小,单位 word
tTimeout	TIME	T#50s			超时时间
bDone	BOOL	FALSE			功能块执行完成
bBusy	BOOL	FALSE			功能块执行中，TCP正常连接
bError	BOOL	FALSE			功能块错误
eErrorID	MB_ERRORID	NO_ERROR			功能块故障码
Rbuff	ARRAY [0..3] OF BYTE				
Rbuff[0]	BYTE	16#00			
Rbuff[1]	BYTE	16#00			
Rbuff[2]	BYTE	16#00			
Rbuff[3]	BYTE	16#00			
FB_write	FB_MBMasterWrite				

触发 bExecute 引脚，通讯完成后 bDone 引脚输出 TRUE，同时可以看到定义的数据缓冲区 Rbuff【0..3】中读取到相应数据，分别为 64 02， 00 01（数据高低位相反），与之前 Q1 从站中设置的寄存器 0-1 中的值 16#0264，16#0010 相同。



表达式	类型	值	准备值	地址	注释
FB_read	FB_MBMasterRead				
bExecute	BOOL	TRUE			触发
bAbort	BOOL	FALSE			终止功能块运行
sSlaveIP	STRING(15)	'192.168.88.200'			从站IP
wPort	UINT	16#01F6			从站端口
wUnitID	BYTE	16#01			从站站号
eFunctionCode	MB_FUNCTIONCODE	ReadHoldRegister			Modbus功能码
dAddress	UINT	16#0000			读取地址
dQuantity	UINT (UINT#1..125)	16#0002			读取数量 ,最大 125个word
pReadBuffer	POINTER TO BYTE	16#75FF11AE			读取数据缓冲区
dBufferLenth	UINT	16#0004			读取数据缓冲区大小,单位 word
tTimeout	TIME	T#50s			超时时间
bDone	BOOL	TRUE			功能块执行完成
bBusy	BOOL	FALSE			功能块执行中，TCP正常连接
bError	BOOL	FALSE			功能块错误
eErrorID	MB_ERRORID	NO_ERROR			功能块故障码
Rbuff	ARRAY [0..3] OF BYTE				
Rbuff[0]	BYTE	16#64			
Rbuff[1]	BYTE	16#02			
Rbuff[2]	BYTE	16#00			
Rbuff[3]	BYTE	16#01			
FB_write	FB_MBMasterWrite				

3. 3. FB_MBMasterWrite (FB)

ModbusTCP 主站写功能块，无需设置设备树配置可直接通过功能块指定的 IP 和端口连接从站进行数据写操作。

变量

名称	FB_MBMasterWrite（主站 MB 写功能块）		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	


```
FB_MBMasterWrite (
    bExecute:= ,
    bAbort:= ,
    sSlaveIP:= ,
    wPort:= ,
    wUnitID:= ,
    eFunctionCode:= ,
    dAddress:= ,
    dQuantity:= ,
    dValue:= ,
    pWriteBuffer:= ,
    dBufferLenth:= ,
    bDone=> ,
    bError=> ,
    bBusy=> ,
    ErrorID=> );
```

（1）输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bExecute	使能	BOOL	TRUE、FALSE	FALSE	TRUE：启用功能块 FALSE：关闭功能块
bAbort	终止	BOOL	TRUE、FALSE	FALSE	TRUE： 终止功能块运行
sSlaveIP	从站 IP 地址	STRING（15）			从站 IP
wPort	端口	UINT	0，正数	502	从站端口
wUnitID	站号	BYTE	0，正数		从站站号
eFunctionCode	功能码	MB_FunctionCode			Modbus 功能码

dAddress	起始地址	UINT			需要操作的寄存器起始地址
dQuantity	读取数量	UINT（1..125）			写入寄存器个数，当功能码为 16 或者 15 时使用，单位为 WORD
dValue	待写入的值	WORD			代表需要写入的单个寄存器的值
pWriteBuffer	缓冲区	POINTER TO BYTE		SM3_Basic.NULL	当功能码为 16 或者 15 时的写入数据缓冲区
dBufferLenth	缓冲区大小	UINT			写入数据缓冲区大小单位 word

（2）输出变量

输出变量	名称	数据类型	有效范围	内容
bDone	完成	BOOL	TRUE、FALSE	TRUE:功能块执行完成
bBusy	运行中	BOOL	TRUE、FALSE	TRUE:功能块执行中，TCP 正常连接
bError	错误	BOOL	TRUE、FALSE	TRUE:功能块产生异常，已停止执行
eErrorID	错误代码	SMC_ERROR	0	功能块产生异常时输出对应的故障码

（3）输出变量的转化时序

变量	变为 TRUE 时	变为 FALSE 时
bDone	✧ 功能块执行完成时	✧ 功能块未完成
bBusy	✧ 功能块运行中且无错误	✧
bError	✧ 功能块产生异常	✧ 功能块无错误

要点说明

- 将 bExecute 设为 TRUE 后，FB_MBMasterWrite 进入执行状态，此时其他的输入引脚被 FB_MBMasterWrite 处理。
- 将 Enable 设为 FALSE 后，FB_MBMasterWrite 将不会再执行任何功能块，此时修改 FB_MBMasterWrite 其他的输入引脚将不会产生任何效果。
- bAbort 引脚用于终止功能块运行，bAbort 置 True 时，输出引脚全部变为 FALSE，eErrorID 信息也会清除，数据接收缓存区清空，需要将 bAbort 引脚重新置 FALSE 后功能块才可以再次被触发，否则无效。
- pReadBuffer 引脚为指向 BYTE 数组首地址的指针，应该根据具体想要读取的内存类型设置数组长度，例如读取保持型寄存器，每个寄存器需要两个 BYTE 的空间存放数据。
- 此处 pReadBuffer 读取数据缓存区在读取到数据后，数据存放位置为反向，例如数组 arr【0..1】

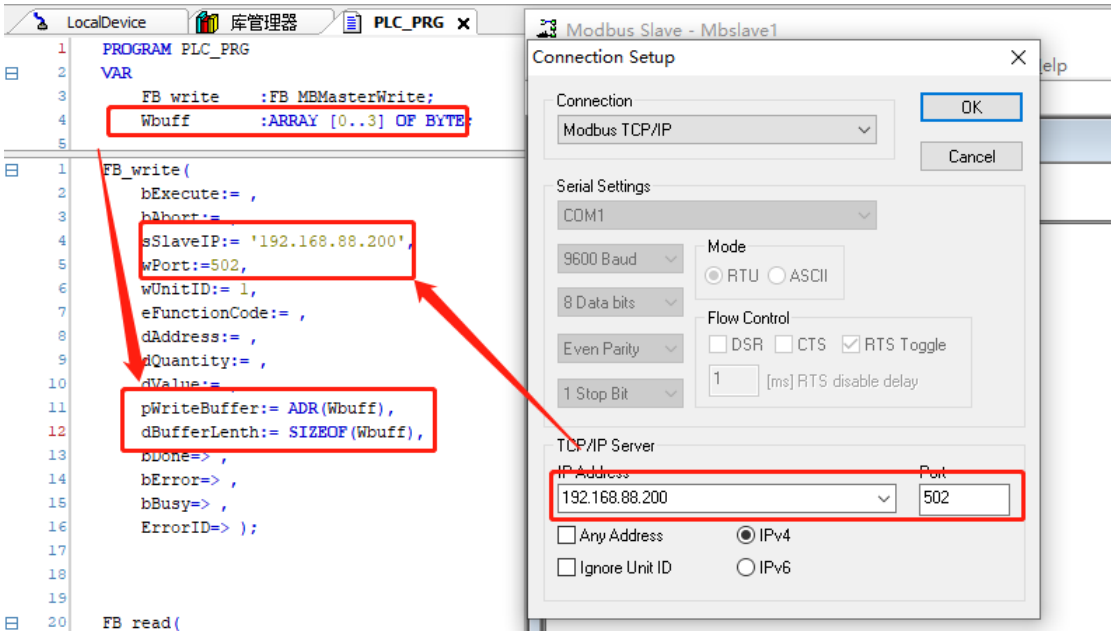
BYTE 读取到寄存器的数据为 0002，实际上存放方式为 arr【0】：02，arr【1】：00。

3.3.1. 使用举例（Q1 做主站，客户端模式）

【1】主站 Q1 设置

Q1 主站声明并调用 FB_MBMasterWrite 功能块，声明一个 BYTE 类型的数组 Wbuff 用作数据发送缓冲区，无需在设备树中作其他操作。

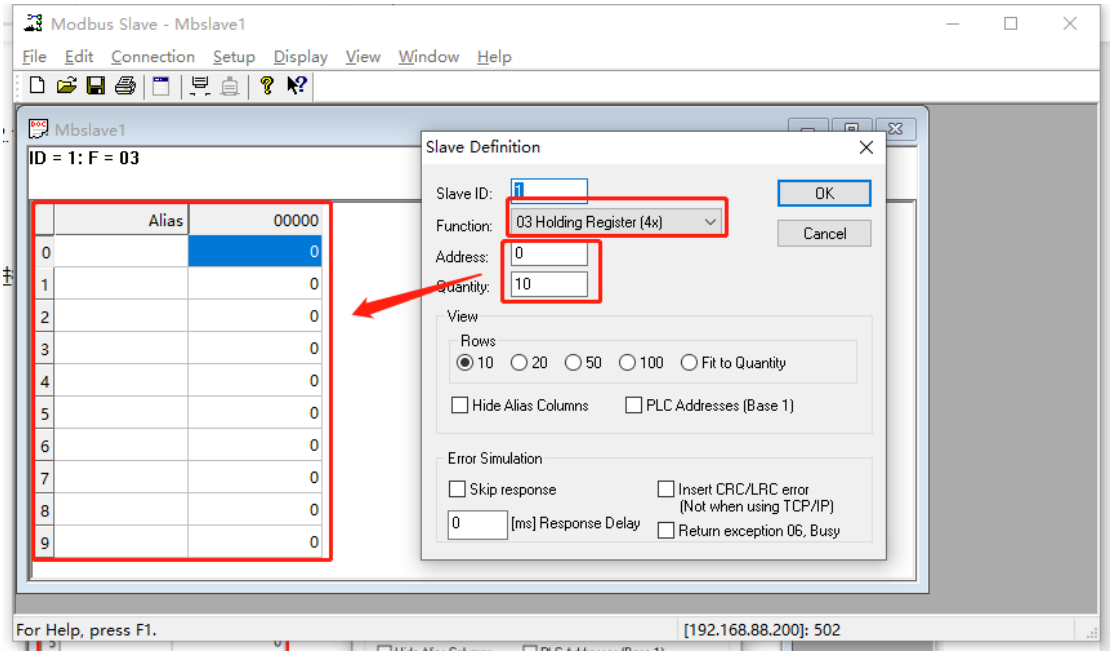
功能块引脚的具体设置如下所示



此处通过主站 Q1 的 Port2 连接上位电脑 ModbusTcp 服务端工具实现通讯。电脑网口 IP 地址为 192.168.88.200，端口号设置为 502。

【2】上位工具服务端设置

修改服务器端数据类型为保持型寄存器，起始地址为 0，长度为 10，即存在 0-9 共 10 个寄存器，方便进行修改操作。



【3】写单个寄存器

将程序下载登陆到主站 Q1 设备，设置功能码，目标寄存器地址。需要写入的值此处修改地址为 0 的寄存器数据为 16#1010，即十进制 4112。

表达式	类型	值	准备值	地址	注释
FB_write	FB_MBMasterWrite				
bExecute	BOOL	FALSE			开启ModbusTCP客户端
bAbort	BOOL	FALSE			终止功能块运行
sSlaveIP	STRING(15)	'192.168.88.200'			需要连接的服务器的IP地址
wPort	UINT	16#01F6			服务器开启侦听的端口号 默认 502
wUnitID	BYTE	16#01			从站节点号
eFunctionCode	MB_FUNCTIONCODE	ReadCoils	WriteSignleRegister		Modbus 功能码
dAddress	WORD	16#0000	16#0000		需要操作的寄存器或者线圈的起始地址
dQuantity	WORD (0..122)	16#0001			写入寄存器个数，当功能码为16或者15时使用 单位 word，
dValue	WORD	16#0000	16#1010		代表需要写入的单个寄存器的值
pWriteBuffer	POINTER TO BYTE	16#75FF11A8			当功能码为15或者16时的写入数据缓存区
dBufferLenth	UINT	16#0004			写入数据缓冲区大小 单位 word
bDone	BOOL	FALSE			TRUE 服务器连接成功
bError	BOOL	FALSE			TRUE：有错误产生
bBusy	BOOL	FALSE			功能块执行中
ErrorID	MB_ERRORID	NO_ERROR			报错代码
TCP_Client1	NBS.TCP_Client				
TCP_Read1	NBS.TCP_Read				
TCP_Write1	NBS.TCP_Write				
TCP_Write2	NBS.TCP_Write				
ipAddr1	NBS.IP_ADDR				
szCount	UDINT	16#00000000			

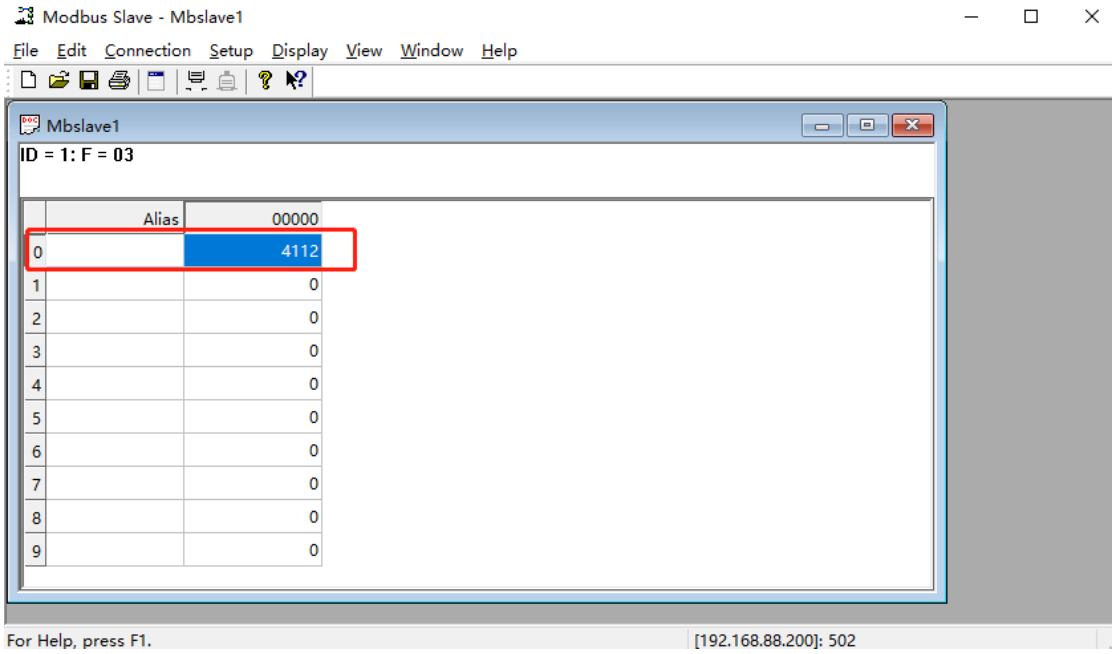
触发 bExecute 引脚，通讯完成后 bDone 引脚输出 TRUE，

LocalDevice | 库管理器 | PLC_PRG

Device.Application.PLC_PRG

表达式	类型	值	准备值	地址	注释
FB_write	FB_MBMasterWrite				
bExecute	BOOL	TRUE			开启ModbusTCP客户端
bAbort	BOOL	FALSE			终止功能块运行
sSlaveIP	STRING(15)	'192.168.88.200'			需要连接的服务器的IP地址
wPort	UINT	16#01F6			服务器开启侦听的端口号 默认 502
wUnitID	BYTE	16#01			从站节点号
eFunctionCode	MB_FUNCTIONCODE	WriteSignleRegister			Modbus 功能码
dAddress	WORD	16#0000			需要操作的寄存器或者线圈的起始地址
dQuantity	WORD (0..122)	16#0001			写入寄存器个数，当功能码为16或者15时使用 单位 word，
dValue	WORD	16#1010			代表需要写入的单个寄存器的值
pWriteBuffer	POINTER TO BYTE	16#75FF11A8			当功能码为15或者16时的写入数据缓存区
dBufferLenth	UINT	16#0004			写入数据缓冲区大小 单位 word
bDone	BOOL	TRUE			TRUE 服务器连接成功
bError	BOOL	FALSE			TRUE：有错误产生
bBusy	BOOL	FALSE			功能块执行中
ErrorID	MB_ERRORID	NO_ERROR			报错代码

同时上位机工具中，地址为 0 的寄存器值已经被修改为 4112。



【4】写多个寄存器

将程序下载登陆到主站 Q1 设备，设置功能码 16，目标寄存器起始地址，需要操作的寄存器个数，Wbuff 数据发送缓冲区数组中设置对应需要写入的值。

此处演示修改地址 1-2 两个寄存器，分别修改值为 16#0001，即十进制 1 和值 16#1000，即十进制 4096（注意缓冲区高低位相反）。

LocalDevice | 库管理器 | PLC_PRG

Device.Application.PLC_PRG

表达式	类型	值	准备值	地址	注释
FB_write	FB_MBMasterWrite				
bExecute	BOOL	FALSE			开启ModbusTCP客户端
bAbort	BOOL	FALSE			终止功能块运行
sSlaveIP	STRING(15)	'192.168.88.200'			需要连接的服务器的IP地址
wPort	UINT	16#01F6			服务器开启侦听的端口号 默认 502
wUnitID	BYTE	16#01			从站节点号
eFunctionCode	MB_FUNCTIONCODE	WriteSignleRegister	WriteMultipleRegister		Modbus 功能码
dAddress	WORD	16#0000	16#0001		需要操作的寄存器或者线圈的起始地址
dQuantity	WORD (0..122)	16#0001	16#0002		写入寄存器个数，当功能码为16或者15时使用 单位 word，
dValue	WORD	16#1010			代表需要写入的单个寄存器的值
pWriteBuffer	POINTER TO BYTE	16#75FF11A8			当功能码为15或者16时的写入数据缓存区
dBufferLenth	UINT	16#0004			写入数据缓冲区大小 单位 word
bDone	BOOL	FALSE			TRUE 服务器连接成功
bError	BOOL	FALSE			TRUE：有错误产生
bBusy	BOOL	FALSE			功能块执行中
ErrorID	MB_ERRORID	NO_ERROR			报错代码
TCP_Client1	NBS.TCP_Client				
TCP_Read1	NBS.TCP_Read				
TCP_Write1	NBS.TCP_Write				
TCP_Write2	NBS.TCP_Write				
ipAddr1	NBS.IP_ADDR				
szCount	UDINT	16#00000000			

16功能码，写多个寄存器

Wbuff	ARRAY [0..3] OF BYTE		
Wbuff[0]	BYTE	16#00	16#01
Wbuff[1]	BYTE	16#00	16#00
Wbuff[2]	BYTE	16#00	16#00
Wbuff[3]	BYTE	16#00	16#10

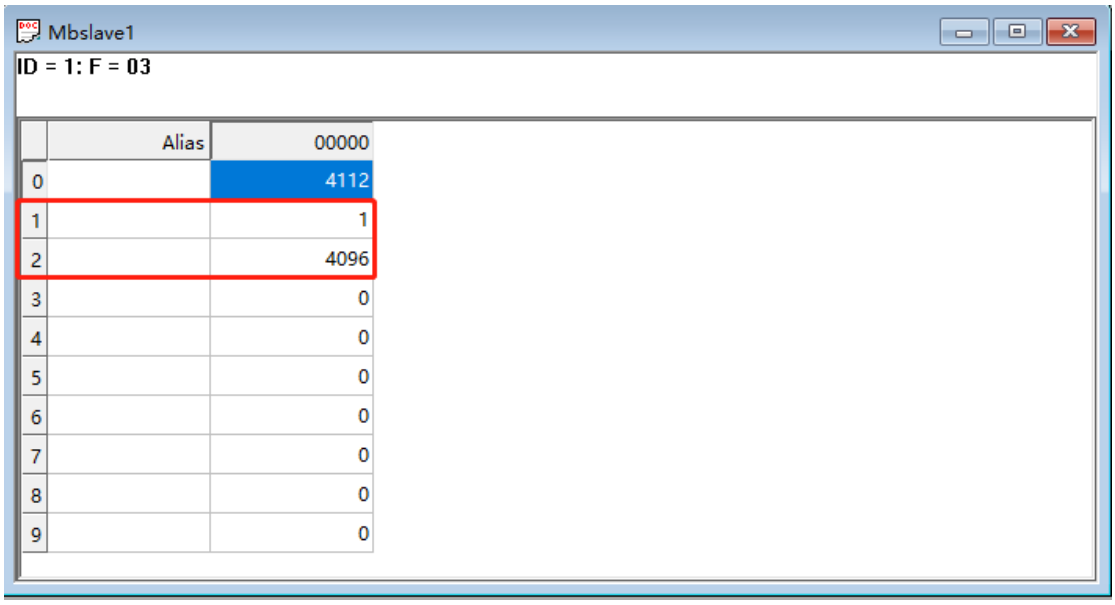
触发 bExecute 引脚，通讯完成后 bDone 引脚输出 TRUE

LocalDevice | 库管理器 | PLC_PRG

Device.Application.PLC_PRG

表达式	类型	值	准备值	地址	注释
FB_write	FB_MBMasterWrite				
bExecute	BOOL	TRUE			开启ModbusTCP客户端
bAbort	BOOL	FALSE			终止功能块运行
sSlaveIP	STRING(15)	'192.168.88.200'			需要连接的服务器的IP地址
wPort	UINT	16#01F6			服务器开启侦听的端口号 默认 502
wUnitID	BYTE	16#01			从站节点号
eFunctionCode	MB_FUNCTIONCODE	WriteMultipleRegister			Modbus 功能码
dAddress	WORD	16#0001			需要操作的寄存器或者线圈的起始地址
dQuantity	WORD (0..122)	16#0002			写入寄存器个数，当功能码为16或者15时使用 单位 word，
dValue	WORD	16#0000			代表需要写入的单个寄存器的值
pWriteBuffer	POINTER TO BYTE	16#75FF1814			当功能码为15或者16时的写入数据缓存区
dBufferLenth	UINT	16#0004			写入数据缓冲区大小 单位 word
bDone	BOOL	TRUE			TRUE 服务器连接成功
bError	BOOL	FALSE			TRUE：有错误产生
bBusy	BOOL	FALSE			功能块执行中
ErrorID	MB_ERRORID	NO_ERROR			报错代码
TCP_Client1	NBS.TCP_Client				
TCP_Read1	NBS.TCP_Read				
TCP_Write1	NBS.TCP_Write				
TCP_Write2	NBS.TCP_Write				
ipAddr1	NBS.IP_ADDR				

同时上位机工具中，寄存器 1-2 的值已经被修改为 1 和 4096。

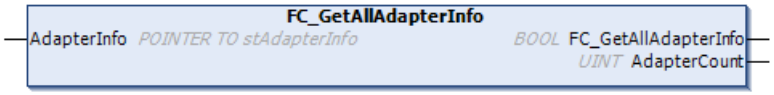


3.4. NetManger（功能组）

网口管理相关功能组，支持读取所有网口信息，以及设置对应信息等功能

3.4.1. FC_GetAllAdapterInfo (FUN)

变量

名称	FC_GetAllAdapterInfo（获取设备网口信息）		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	
		FC_GetAllAdapterInfo(AdapterInfo:= , AdapterCount=>);	

（1）输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
AdapterInfo	设备信息	POINTER TO stAdapterInfo			指向设备信息结构体的指针

（2）输出变量

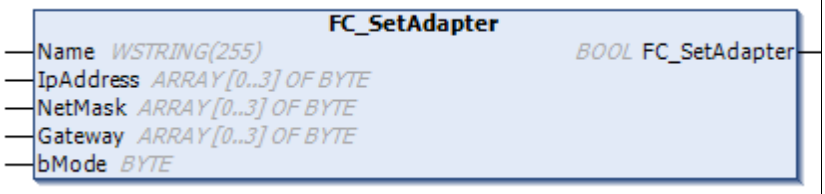
输出变量	名称	数据类型	有效范围	内容
AdapterCount	设备数量	UINT		当前连接设备数量

要点说明

- AdapterInfo 引脚是指向设备结构体数组的指针，调用函数并且程序处于运行状态时，函数自动读取当前设备所有网口信息。

3.4.2. FC_SetAdapter (FUN)

变量

名称	FC_SetAdapter（设置网口信息）		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	
		FC_SetAdapter(Name:= , IpAddress:= , NetMask:= , Gateway:= , bMode:=)	

（1）输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
Name	名称	WSTRING			网口名称
IpAddress	IP 地址	ARRAY [0..3] OF BYTE			IP 地址
NetMask	子网掩码	ARRAY [0..3] OF BYTE			子网掩码
Gateway	网关	ARRAY [0..3] OF BYTE			网关
bMode	写入模式	BYTE			需要修改的内容选择

要点说明

- bMode 已经用于控制是否对当前设备进行各个信息修改，bit0 代表是否写 IpAddress，bit1 代表是否写 NetMask，bit2 代表是否写 Getwag。
- 例：bMode 设置为 5，即 0101，代表修改 IP 和网关。

3.4.3. 使用举例

【1】声明变量和调用函数

对于函数的各个引脚都映射对应的变量。

```
PROGRAM PLC_PRG
VAR
    info    :ARRAY [0..4] OF  stAdapterInfo;
    num     :UINT;

    Name    :WSTRING;
    IP      :ARRAY [0..3] OF BYTE;
    Net     :ARRAY [0..3] OF BYTE;
    Getw    :ARRAY [0..3] OF BYTE;
    Mode    :BYTE;
END VAR
```

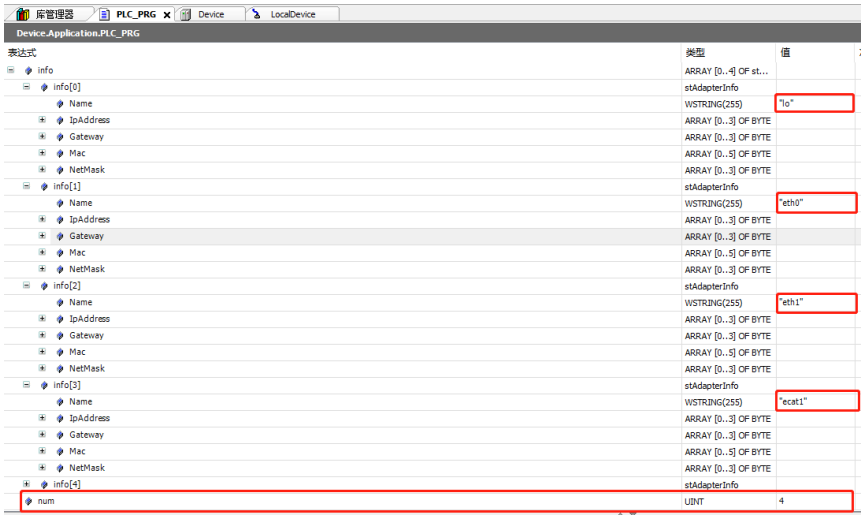
```
FC_GetAllAdapterInfo(AdapterInfo:= info, AdapterCount=> num);

FC_SetAdapter(
    Name:= Name,
    IpAddress:= IP,
    NetMask:= Net,
    Gateway:= Getw,
    bMode:= Mode);
```

【2】读取设备所有网口信息

登陆程序后，FC_GetAllAdapterInfo 函数会自动读取当前设备的所有网口信息，写入到 info 结构体数组中。

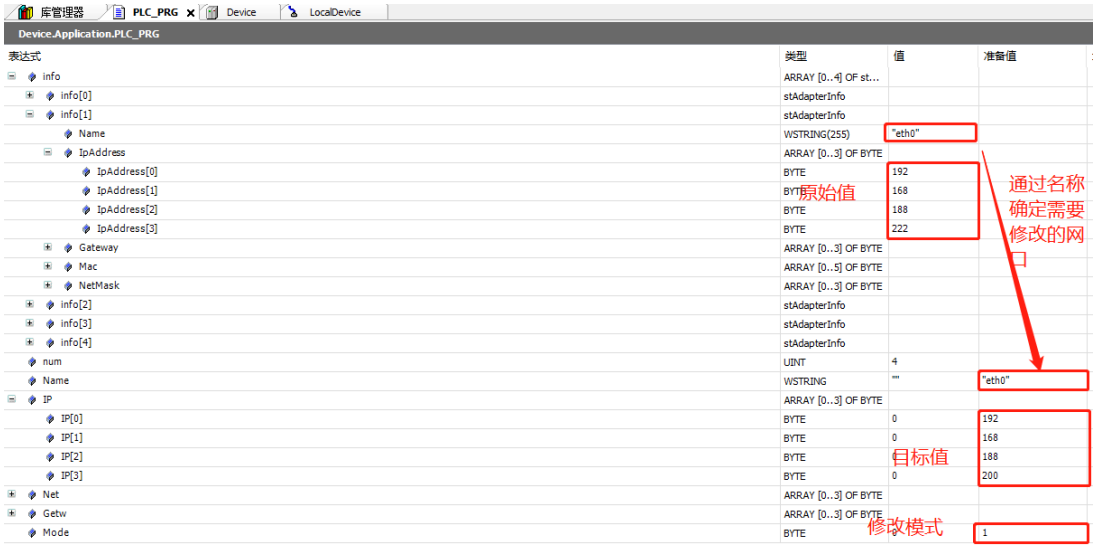
同时 AdapterCount 引脚输出当前读取到的设备网口数。



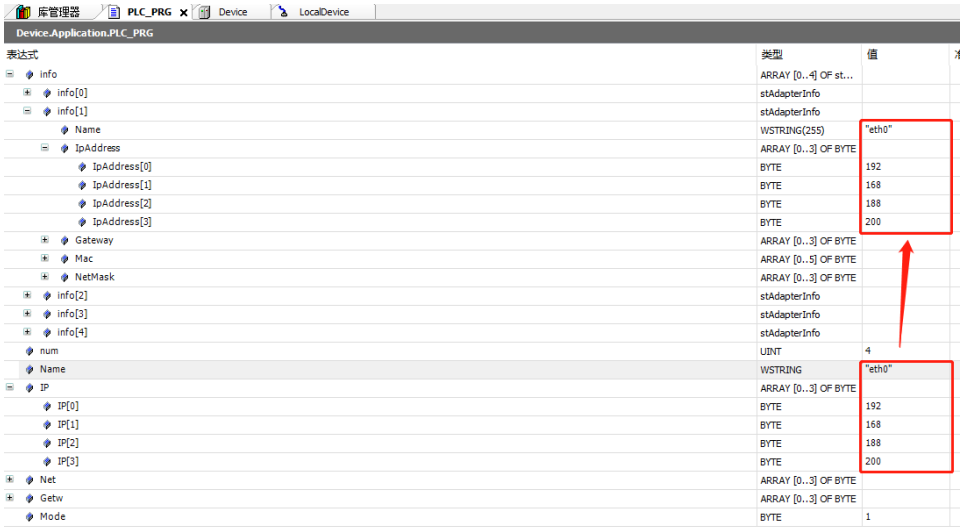
此处展示读取到的 Eth0 网口信息

【3】修改指定网口的指定信息

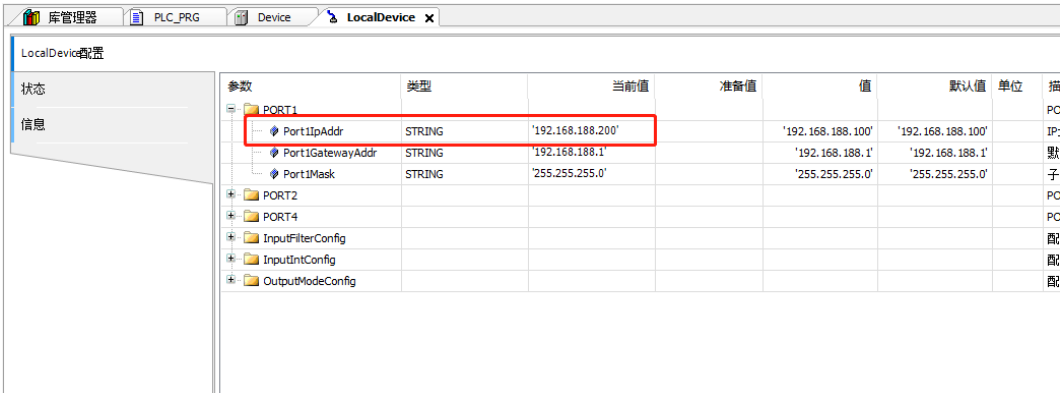
此处演示修改 Ethe0 网口的 IP 地址为 192.168.88.200。



右键写入数据后，可以看到设备网口 eth0 的 ip 地址引脚被修改完成



在 LocalDevice 中，可以看到设备的 Port1 网口 IP 地址已经变为 192.168.188.200。

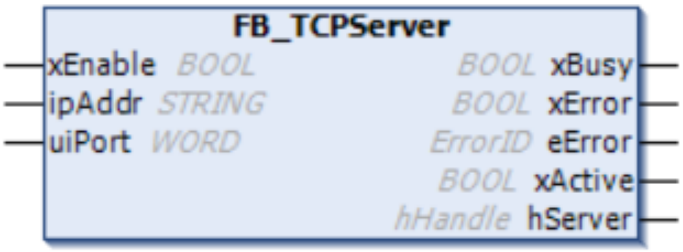


3.5. TCP（功能组）

基于 socket 套接字封装的 TCP 通讯功能块

3.5.1. FB_TCPServer (FB)

变量

名称	FB_TCPServer(创建服务功能块)	
图形表现		ST 表现
		FB_TCPServer (xEnable:= , ipAddr:= , uiPort:= , xBusy=> , xError=> , eError=> , xActive=> , hServer=>);

（1）输入变量

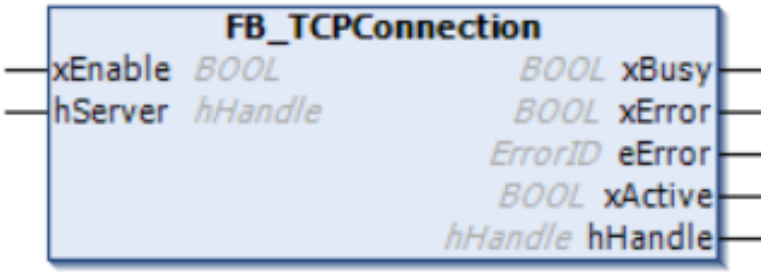
输入变量	名称	数据类型	有效范围	初始值	内容
xEnable	功能块使能	BOOL	TRUE、FALSE	FALSE	
ipAddr	服务器 IP 地址	STRING		0	
uiPort	服务器端口号	WORD		0	

（2）输出变量

输出变量	名称	数据类型	有效范围	内容
xBusy		BOOL	TRUE、FALSE	TRUE：功能块运行
xError		BOOL	TRUE、FALSE	
eError		ErrorID		报错代码
xActive		BOOL	TRUE、FALSE	TRUE:创建成功输出句柄 ，FALSE:当前句柄无效
hServer	句柄	hHandle		创建生成的接口句柄，给到连接功能块

3.5.2. FB_TCPConnection (FB)

变量

名称	FB_TCPConnection (连接功能块)	
图形表现		ST 表现
		FB_TCPConnection(xEnable:= , hServer:= , xBusy=> , xError=> , eError=> , xActive=> , hHandle=>);

（1）输入变量

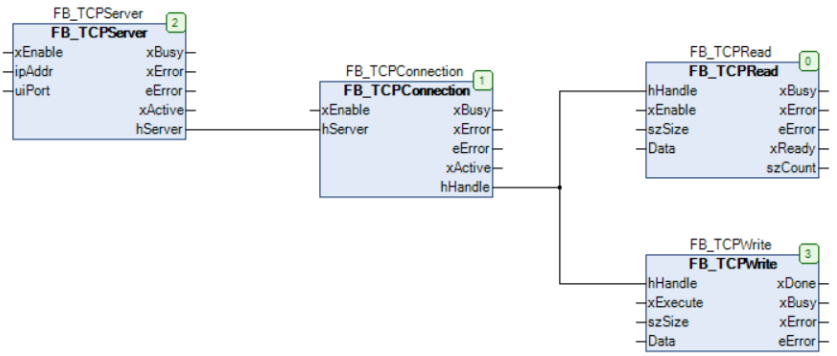
输入变量	名称	数据类型	有效范围	初始值	内容
xEnable	功能块使能	BOOL	TRUE、FALSE	FALSE	
hServer	句柄	hHandle			接受 FB_TCPServer 生成的句柄

（2）输出变量

输出变量	名称	数据类型	有效范围	内容
xBusy		BOOL	TRUE、FALSE	
xError		BOOL	TRUE、FALSE	
eError		ErrorID		报错代码
xActive		BOOL	TRUE、FALSE	TRUE:创建成功输出句柄 ，FALSE:当前句柄无效
hHandle		hHandle		连接句柄，给到读写功能块

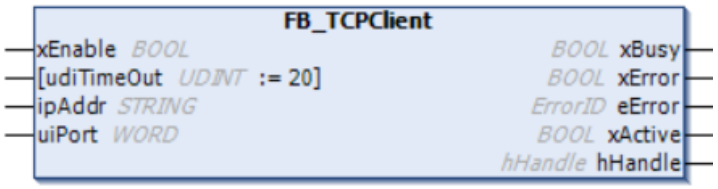
要点说明

- 句柄连接使用形式如下。



3.5.3. FB_TCPClient (FB)

变量

名称	FB_TCPClient (创建客户端功能块)	
图形表现		ST 表现
		FB_TCPClient (xEnable:= , udiTimeOut:=, ipAddr:=, uiPort:= , xBusy=> , xError=> , eError=> , xActive=> , hHandle=>);

（1）输入变量

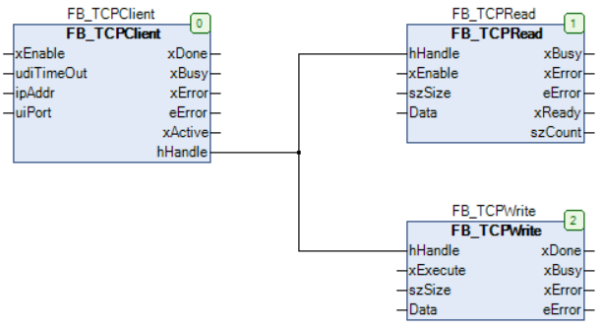
输入变量	名称	数据类型	有效范围	初始值	内容
xEnable	功能块使能	BOOL	TRUE、FALSE	FALSE	
udiTimeOut	句柄	UDINT		20	单位：ms 连接超时时间，受功能块所在任务周期影响
ipAddr	服务器 IP 地址	STRING		0	
uiPort	服务器端口号	WORD		0	

（2）输出变量

输出变量	名称	数据类型	有效范围	内容
xBusy		BOOL	TRUE、FALSE	
xError		BOOL	TRUE、FALSE	
eError		ErrorID		报错代码
xActive		BOOL	TRUE、FALSE	TRUE:创建成功输出句柄 ，FALSE:当前句柄无效
hHandle		hHandle		连接句柄，给到读写功能块

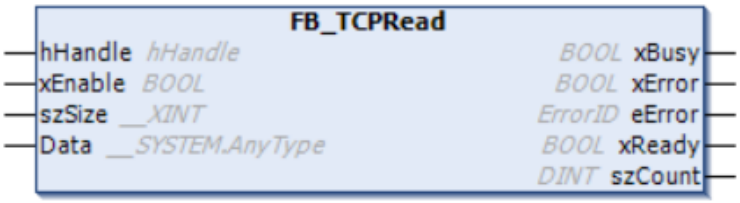
要点说明

- 句柄连接使用形式如下



3.5.4. FB_TCPRead (FB)

变量

名称	FB_TCPRead (读功能块)	
图形表现		ST 表现
		FB_TCPRead (hHandle:=, xEnable:= , szSize:=, Data:=, xBusy=> , xError=> , eError=> , xReady=> , szCount=>);

（1）输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
hHandle	句柄	hHandle			接受客户端或者连接功能块创建的句柄
xEnable	功能块使能	BOOL	TRUE、FALSE	FALSE	
szSize	接受数据长度	_XINT			在 32 位系统中等效 DINT,在 64 位系统中等效 LINT。不得大于接受数据缓存区大小
Data	接受数据缓存区	ANY			

（2）输出变量

输出变量	名称	数据类型	有效范围	内容
xBusy		BOOL	TRUE、FALSE	
xError		BOOL	TRUE、FALSE	
eError		ErrorID		报错代码
xReady		BOOL	TRUE、FALSE	接受到数据时有效，只有一个周期
szCount		DINT		接收到的数据长度，接受到数据时有效，只有一个周期

要点说明

- Data 引脚直接给入示例数据变量即可。

3.5.5. FB_TCPWrite (FB)

变量

名称	FB_TCPWrite (读功能块)	
图形表现		ST 表现
		FB_TCPWrite (hHandle:=, xExecute:= , szSize:= , Data:= , xDone=> , xBusy=> , xError=> , eError=>);

（1）输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
hHandle	句柄	hHandle			接受客户端或者连接功能块创建的句柄
xExecute	功能块使能	BOOL	TRUE、FALSE	FALSE	
szSize	发送数据长度	_XINT			在 32 位系统中等效 DINT,在 64 位系统中等效 LINT。不得大于发送数据缓存区大小
Data	发送数据缓存区	ANY			

（2）输出变量

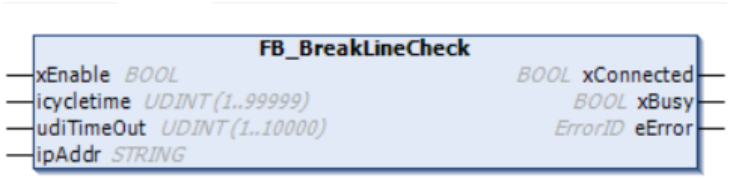
输出变量	名称	数据类型	有效范围	内容
xDone		BOOL	TRUE、FALSE	xExecute 没有 TRUE 时，指输出一个周期
xBusy		BOOL	TRUE、FALSE	
xError		BOOL	TRUE、FALSE	
eError		ErrorID		报错代码

要点说明

- Data 引脚直接给入示例数据变量即可。

3.5.6. FB_BreakLineCheck (FB)

变量

名称	FB_BreakLineCheck (断线检测功能块)	
图形表现		ST 表现
		FB_BreakLineCheck (xEnable:= , icycletime:= , udiTimeOut:= , ipAddr:= , xConnected=> , xBusy=> , eError=>);

（1）输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
xEnable	功能块使能	BOOL	TRUE、FALSE	FALSE	
icycletime	检测触发循环间隔次数	UDINT	1-99999	10	当前所在任务周期的循环次数
udiTimeOut	连接超时时间	UDINT	1-99999	10	超时时间
ipAddr	连接的 IP 地址	STRING		'192.168.88.100'	

（2）输出变量

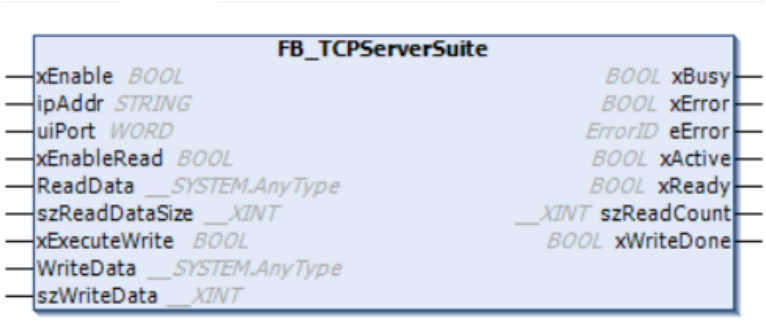
输入变量	名称	数据类型	有效范围	内容
xConnected	连接状态	BOOL	TRUE、FALSE	按照 icycletime 设置中的时间间隔输出状态
xBusy		BOOL	TRUE、FALSE	
eError		ErrorID		报错代码

要点说明

- 该功能块只能做断线检测，不能用于心跳检测功能。
- **icycletime** 引脚值是任务周期循环间隔的次数，任务周期时间是当前功能块所在任务周期的时间。
- 该功能块不建议放在低任务周期使用，不建议放在任务周期短的周期使用，不建议间隔时间太短，否则可能会引发负载率升高、影响通讯。

3.5.7. FB_TCPServerSuite (FB)

变量

名称	FB_TCPServerSuite (服务器封装功能块)	
图形表现		ST 表现
		FB_TCPServerSuite (xEnable:= , ipAddr:= , uiPort:= , xEnableRead:= , ReadData:= , szReadDataSize:= , xExecuteWrite:= , WriteData:= , szWriteData:= , xBusy=> , xError=> , eError=> , xActive=> , xReady=> , szReadCount=> , xWriteDone=>);

（1）输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
xEnable	功能块使能	BOOL	TRUE、FALSE	FALSE	
ipAddr	服务器 IP 地址	STRING		0	
uiPort	服务器端口号	WORD		0	
xEnableRead	功能块使能	BOOL	TRUE、FALSE	FALSE	
ReadData	接受数据缓存区	ANY			
szReadDataSize	接受数据长度	_XINT			在 32 位系统中等效 DINT,在 64 位系统中等效 LINT。不得大于接受数据缓存区大小
xExecuteWrite	功能块使能	BOOL	TRUE、FALSE	FALSE	
WriteData	发送数据缓存区	ANY			
szWriteData	发送数据长度	_XINT			在 32 位系统中等效 DINT,在 64 位系统中等效 LINT。不得大于发送数据缓存区大小

（2）输出变量

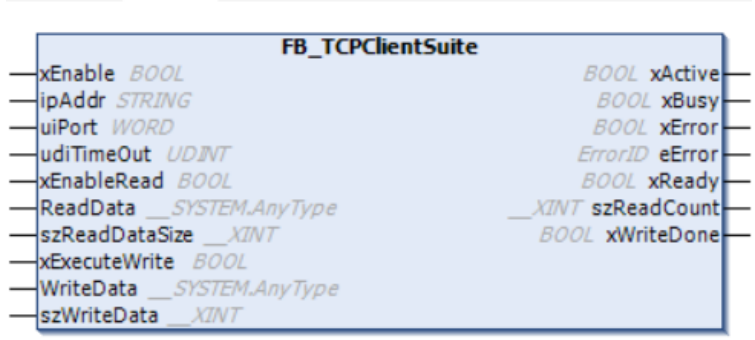
输出变量	名称	数据类型	有效范围	内容
xBusy		BOOL	TRUE、FALSE	
xError		BOOL	TRUE、FALSE	
eError		ErrorID		报错代码
xActive		BOOL	TRUE、FALSE	连接成功引脚
xReady	接收状态	BOOL	TRUE、FALSE	接受到数据时有效，只有一个周期
szReadCount	接收数据长度	DINT		接收到的数据长度，接受到数据时有效，只有一个周期
xWriteDone		BOOL	TRUE、FALSE	xExecute 没有 TRUE 时，指输出一个周期

要点说明

- FB_TCPServer、FB_TCPConnection、FB_TCPRead、FB_TCPWrite 整合在一起的功能块。

3.5.8. FB_TCPClientSuite (FB)

变量

名称	FB_TCPClientSuite (客户端封装功能块)	
图形表现		ST 表现
		FB_TCPClientSuite (xEnable:= , ipAddr:= , uiPort:= , udiTimeOut:= , xEnableRead:= , ReadData:= , szReadDataSize:= , xExecuteWrite:= , WriteData:= , szWriteData:= , xActive=> , xBusy=> , xError=> , eError=> , xReady=> , szReadCount=> , xWriteDone=>);

（1）输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
xEnable	功能块使能	BOOL	TRUE、FALSE	FALSE	
udiTimeOut	句柄	UDINT		20	单位：ms 连接超时时间，受功能块所在任务周期影响
ipAddr	服务器 IP 地址	STRING		0	
uiPort	服务器端口号	WORD		0	
xEnableRead	功能块使能	BOOL	TRUE、FALSE	FALSE	
ReadData	接受数据缓存区	ANY			
szReadDataSize	接受数据长度	_XINT			在 32 位系统中等效 DINT,在 64 位系统中等效 LINT。不得大于接受数据缓存区大小
xExecuteWrite	功能块使能	BOOL	TRUE、FALSE	FALSE	

WriteData	发送数据缓存区	ANY			
szWriteData	发送数据长度	_XINT			在 32 位系统中等效 DINT,在 64 位系统中等效 LINT。不得大于发送数据缓存区大小

（2）输出变量

输出变量	名称	数据类型	有效范围	内容
xActive		BOOL	TRUE、FALSE	连接成功引脚
xBusy		BOOL	TRUE、FALSE	
xError		BOOL	TRUE、FALSE	
eError		ErrorID		报错代码
xReady	接收状态	BOOL	TRUE、FALSE	接受到数据时有效，只有一个周期
szReadCount	接收数据长度	DINT		接收到的数据长度，接受到数据时有效，只有一个周期
xWriteDone		BOOL	TRUE、FALSE	xExecute 没有 TRUE 时，指输出一个周期

要点说明

- FB_TCPClient、FB_TCPRead、FB_TCPWrite 整合在一起的功能块。

4. DataProcess (数据处理)

4.1. StreamProcess （功能组）

流处理功能组，用于在连续的数据流（例如数组、字符串）中获取所需类型的数据，或者将对应类型的数据写入数据流中。 注意：本功能不会对数据流剩余长度做任何检查，使用时请保证数据流指针指向的数据长度足够取出或者写入对应的数据类型。

4.1.1. GET（获取数据）

名称	内容
Get_DINT (FUN)	将 BYTE 转换成 DINT
Get_INT (FUN)	将 BYTE 转换成 INT
Get_LREAL(FUN)	将 BYTE 转换成 LREAL
Get_REAL(FUN)	将 BYTE 转换成 REAL
Get_UDINT (FUN)	将 BYTE 转换成 UDINT
Get_UINT (FUN)	将 BYTE 转换成 UINT
Get_USINT (FUN)	将 BYTE 转换成 USINT

函数形式：

Get_XXXX(pData:=);

pData 是指向 BYTE 数组的首地址的指针。

4.1.2. SET（写入数据）

名称	内容
Set_DINT (FUN)	将 DINT 转换成 BYTE 再写入到 BYTE 数组中
Set_INT (FUN)	将 INT 转换成 BYTE 再写入到 BYTE 数组中
Set_LREAL(FUN)	将 LREAL 转换成 BYTE 再写入到 BYTE 数组中
Set_REAL(FUN)	将 REAL 转换成 BYTE 再写入到 BYTE 数组中
Set_UDINT (FUN)	将 UDINT 转换成 BYTE 再写入到 BYTE 数组中
Set_UINT (FUN)	将 UINT 转换成 BYTE 再写入到 BYTE 数组中
Set_USINT (FUN)	将 USINT 转换成 BYTE 再写入到 BYTE 数组中

函数形式：

Set_XXXX(in:= ,pData:=);

In 是需要写入的值。

pData 是指向 BYTE 数组的首地址的指针。

要点说明

- 整形数据转换只需遵从进制转换规则即可。
- 浮点型数据与 BYTE 之间的转换过程需要遵从浮点数规格化。在下方会举例说明。

4.1.3. 使用示例 1（整形数据转换）

【1】获取数据

声明一些整形数据变量用于存放函数输出的值，声明一个 4 字节的 BYTE 数组用于存放数据流。

PLC_PRG | 库管理器 | Device

```
PROGRAM PLC_PRG
VAR
    testbyte    :ARRAY [0..3] OF BYTE;

    getDint     :DINT;
    getInt      :INT;
    getUdint    :UDINT;
    getUint     :UINT;
    getUsint    :USINT;
```

```
getDint :=Get_DINT(pData:= ADR(testbyte));
getInt  :=Get_INT (pData:=ADR(testbyte));
getUdint:=Get_UDINT(pData:=ADR(testbyte));
getUint :=Get_UINT(pData:=ADR(testbyte));
getUsint:=Get_USINT(pData:=ADR(testbyte));
```

登陆程序后，为方便理解，这里使用二进制显示，将 BYTE 数组的值修改为 2#00010000 00000010 00001000 00011000（在数组中，地址滞后的为高位）。可以看到不同函数会根据函数所需空间大小获取数组中的值。

表达式	类型	值
testbyte	ARRAY [0..3] OF BYTE	
testbyte[0]	BYTE	2#00011000
testbyte[1]	BYTE	2#00001000
testbyte[2]	BYTE	2#00000010
testbyte[3]	BYTE	2#00010000
getDint	DINT	2#00010000000000100000100000011000
getInt	INT	2#0000100000011000
getUdint	UDINT	2#00010000000000100000100000011000
getUint	UINT	2#0000100000011000
getUsint	USINT	2#00011000

【2】写入数据

声明一些整形数据变量用于存放函数需要写入的值，声明一个 4 字节的 BYTE 数组用于存放数据流。

PLC_PRG | 库管理器 | Device

```
PROGRAM PLC_PRG
VAR
    testbyte    :ARRAY [0..3] OF BYTE;

    dintnum  :DINT;
    intnum   :INT;
    udintnum :UDINT;
    uintnum  :UINT;
    usintnum :USINT;
```

```
Set_DINT (in:= dintnum, pData:=ADR(testbyte));
Set_INT  (in:= intnum, pData:=ADR(testbyte));
Set_UDINT(in:= udintnum, pData:=ADR(testbyte));
Set_UINT (in:= uintnum, pData:=ADR(testbyte));
Set_USINT(in:= usintnum, pData:=ADR(testbyte));
```

由于对同一个数组写入数据会相互影响，此处将其余函数都注释掉，只演示写入 DINT 数据，DINT 为带符号位 32 位整形，修改 dintnum 的值为-158，即 2#11111111 11111111 11111111 01100010（补码）。

PLC_PRG | 库管理器 | Device

Device.Application.PLC_PRG

表达式	类型	值	准备值
testbyte	ARRAY [0..3] OF BYTE		
testbyte[0]	BYTE	0	
testbyte[1]	BYTE	0	
testbyte[2]	BYTE	0	
testbyte[3]	BYTE	0	
dintnum	DINT	0	-158
intnum	INT	0	
udintnum	UDINT	0	
uintnum	UINT	0	
usintnum	USINT	0	
getDint	DINT	0	
getInt	INT	0	
getUdint	UDINT	0	
getUint	UINT	0	
getUsint	USINT	0	
testreal	ARRAY [0..3] OF BYTE		
testlreal	ARRAY [0..7] OF BYTE		

可以看到 BYTE 数组 testbyte 的值已经被修改

PLC_PRG | 库管理器 | Device

Device.Application.PLC_PRG

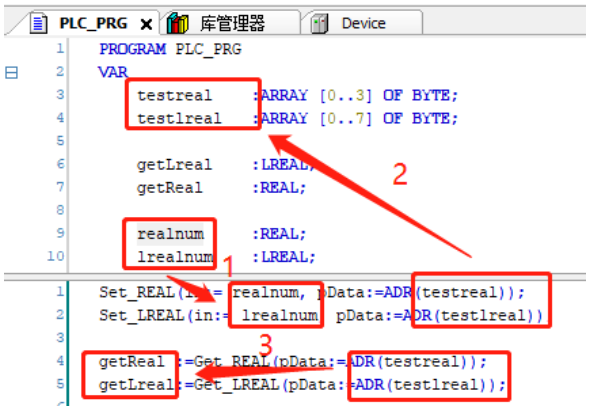
表达式	类型	值
testbyte	ARRAY [0..3] OF BYTE	
testbyte[0]	BYTE	2#01100010
testbyte[1]	BYTE	2#11111111
testbyte[2]	BYTE	2#11111111
testbyte[3]	BYTE	2#11111111
dintnum	DINT	2#11111111111111111111111101100010

4.1.4. 使用示例 2（浮点型数据转换）

为方便理解，浮点数举例先介绍写入数据，在此举例中将详细说明浮点数与 2 进制之间的转换方式。声明浮点型数据用于写函数的输入数组，将 BYTE 数组地址映射到对应函数输出引脚上。（确保 BYTE 数组空间大小足够）。

【1-2】此处先通过写函数将数值 realnum 和 lrealnum 写入到数组 testxxxx 中

【3】在通过读函数将数组 testxxxx 中的值读取到 getreal 和 getlreal 中



举例：将 real 数值 2.5 转换为二进制显示（32 位浮点数组成 ： 符号位 1 位 阶位 8 位 尾数 23 位）

（1）拆分整数和小数将十进制数 2.5 转换成 2 进制数，2#10.1

（2）将 10.1 左移化为 1.01*2^1。

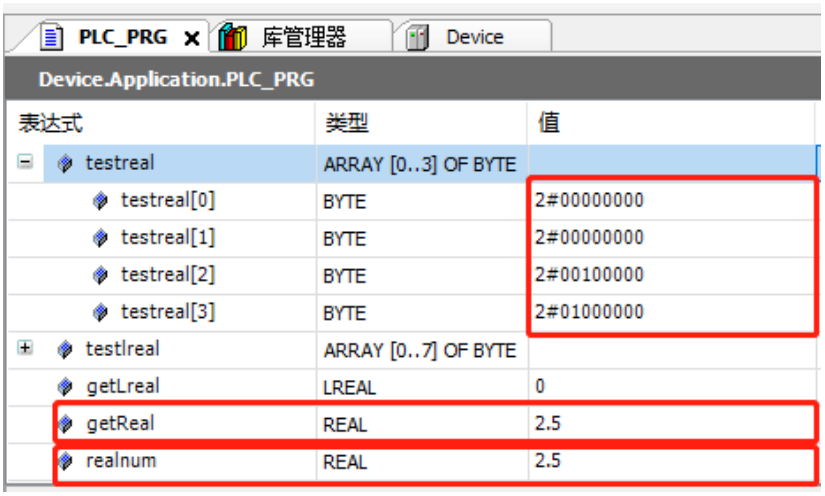
所以，对应可得：

●符号位（S）： 0 （正数）

●阶码（E）：应为 127 + （1） = 128，因此，二进制表示为：10000000。

●尾数部分（M）：补齐 23 位后为 01000000000000000000000。

最终 32 位浮点数用二进制表示为 2# 0 10000000 01000000000000000000000



可以看到数组中得到的值为 2#01000000 00100000 00000000 00000000（注意高低位相反），

与计算结果相同。getReal 得到的值为 2.5（反演过程不再举例，将上述过程反推一遍即可）。

举例：将 lreal 数值 300 转换为二进制显示（64 位浮点数组成 ： 符号位 1 位 阶位 11 位 尾数 52 位）

（3）拆分整数和小数将十进制数 300 转换成 2 进制数，2#00100101100

（4）将 00100101100 左移化为 1.001011 * 2^8。

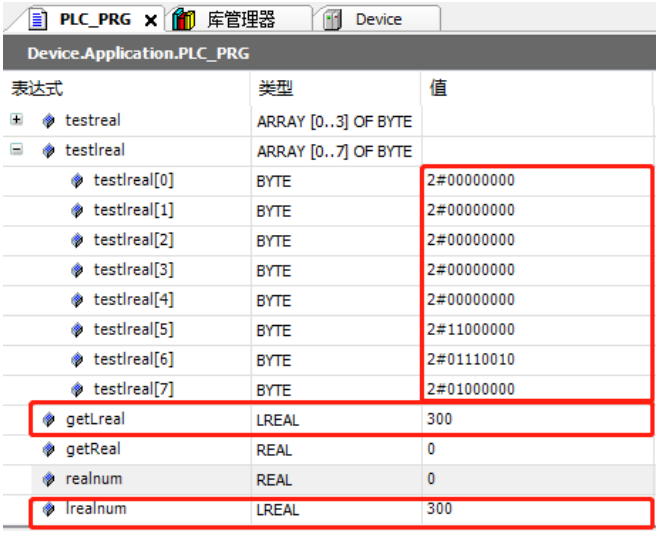
所以，对应可得：

●符号位（S）： 0 （正数）

●阶码（E）：应为 1023 + （8） = 1031，因此，二进制表示为：2#10000000111。

●尾数部分（M）：补齐 52 位后为 00101100000000.....。

最终 64 位浮点数用二进制表示为 2# 0 10000000111 00101100000000.....。

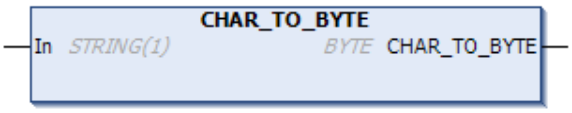


4.2. Type Convert （功能组）

类型转换功能组，包含各种数据类型转换。

4.2.1. CHAR_TO_BYTE (FUN)

变量

名称	CHAR_TO_BYTE（字符转数值）		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	
		CHAR_TO_BYTE(In:=);	

（1）输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
IN	输入字符串	STRING（1）			指向设备信息结构体的指针

（2）输出变量

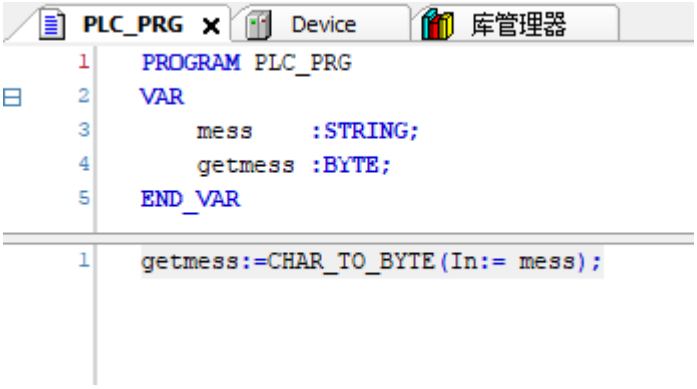
输出变量	名称	数据类型	有效范围	内容
CHAR_TO_BYTE	转换结果	BYTE		字符转成 BYTE 的值

要点说明

- 字符串与字节直接的转换关系请参考 ASCII 码表。

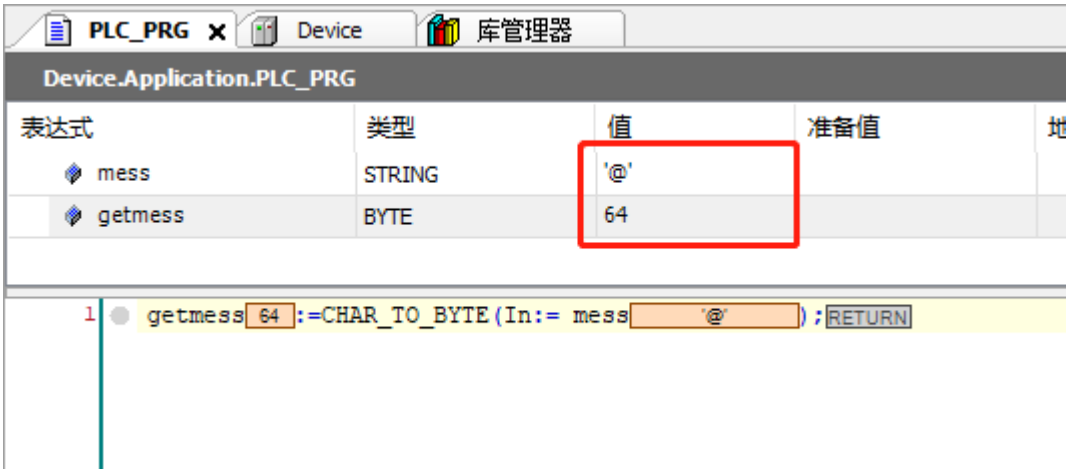
4.2.2. 使用举例

【1】声明如下两个变量，调用函数将 mess（STRING）转换成 BYTE，输出结果赋值给 getmess（BYTE）。



【2】下载并运行程序

修改 mess 的值为‘@’，可以看到 getmess 的值变为，64，与 ASCII 码表上的转换值对应。



4.3. Type Packing （功能组）

数据组合拆分功能组，可将 BYTE、BIT 等类型向其他类型组合转换。

4.3.1. 功能组（Packing）

名称	内容
BOOL_PACK_BYTE (FUN)	将 BOOL 组合转换成 BYTE
BYTE_PACK_DINT (FUN)	将 BYTE 组合转换成 DINT
BYTE_PACK_INT (FUN)	将 BYTE 组合转换成 INT
BYTE_PACK_LREAL (FUN)	将 BYTE 组合转换成 LREAL
BYTE_PACK_REAL (FUN)	将 BYTE 组合转换成 REAL
BYTE_PACK_UDINT (FUN)	将 BYTE 组合转换成 UDINT
BYTE_PACK_UINT (FUN)	将 BYTE 组合转换成 UINT

4.3.2. 功能组（UnPacking）

名称	内容
BYTE_UNPACK_BOOL (FUN)	将 BYTE 拆分成 BOOL
DINT_UNPACK_BYTE (FUN)	将 DINT 拆分成 BYTE
INT_UNPACK_BYTE (FUN)	将 INT 拆分成 BYTE
LREAL_UNPACK_BYTE (FUN)	将 LREAL 拆分成 BYTE
REAL_UNPACK_BYTE (FUN)	将 REAL 拆分成 BYTE
UDINT_UNPACK_BYTE (FUN)	将 UDINT 拆分成 BYTE
UINT_UNPACK_BYTE (FUN)	将 UINT 拆分成 BYTE

要点说明

- 本功能组针对不同类型的数据组合拆分功能，与 StreamProcess 功能组功能类似。

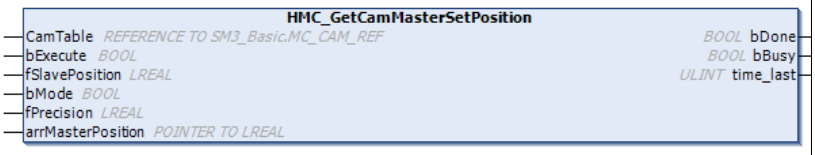
5. Motion (运动控制)

5. 1. GetCamPosition

5. 1. 1. HMC_GetCamMasterSetPosition (FB)

用于实现输入从轴位置算出主轴位置

变量

名称	HMC_GetCamMasterSetPosition		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	
		HMC_GetCamMasterSetPosition(CamTable:= , bExecute:= , fSlavePosition:= , bMode:= , fPrecision:= , arrMasterPosition:= , bDone=> , bBusy=> , time_last=>);	

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
CamTable	凸轮表	REFERENCE TO SM3_Basic.MC_CAM_REF			输入凸轮表，仅支持多项式模式
bExecute	使能	BOOL	TRUE、FALSE	FALSE	TRUE： 功能触发
fSlavePosition	从轴位置	LREAL			从轴位置
bMode	计算模式	BOOL	TRUE、FALSE	FALSE	TRUE：一个周期算一个解 FALSE：一个周期输出全部解
fPrecision	精度	LREAL		0.0001	主轴位置计算精度 单位：应用单元
arrMasterPosition	主轴位置	POINTER TO LREAL			用于存储输出的主轴位置，当有多个匹配的解时，输出多个解

输出变量

输出变量	名称	数据类型	有效范围	内容
bDone	完成	BOOL	TRUE、FALSE	TRUE:功能块执行完成
bBusy	运行中	BOOL	TRUE、FALSE	TRUE:功能块执行中
time_last	计算时间	ULINT		计算时间，单位：us

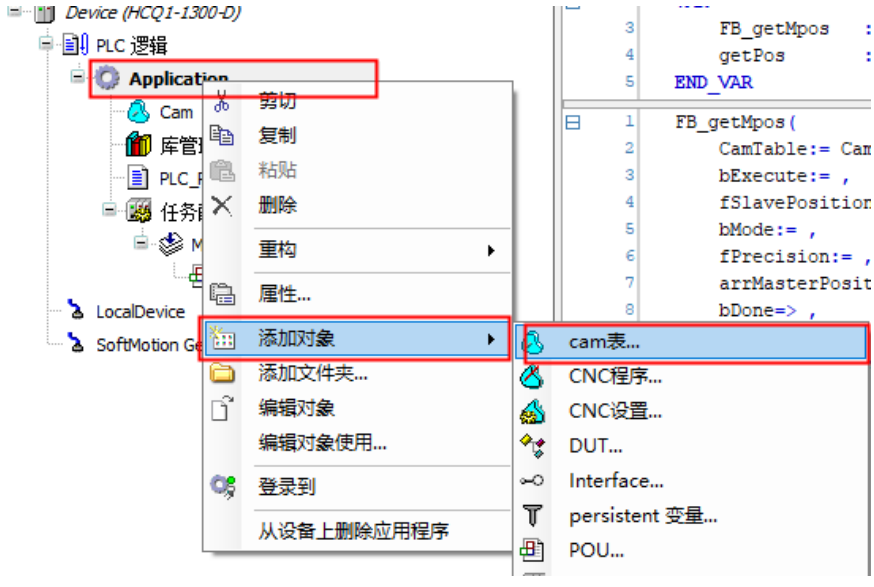
要点说明

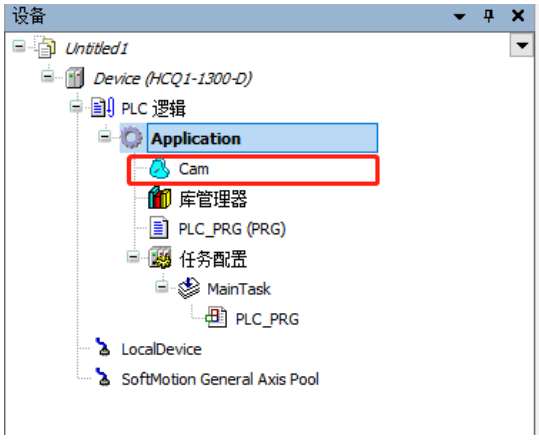
- 可以将计算方式设置为 TRUE 来降低对于任务周期的波动。但是计算点位会需要多个周期。
- arrMasterPosition 为指向 LREAL 数组首地址的指针，用于存放多个 LREAl 类型的数据（主轴位置）。

使用举例

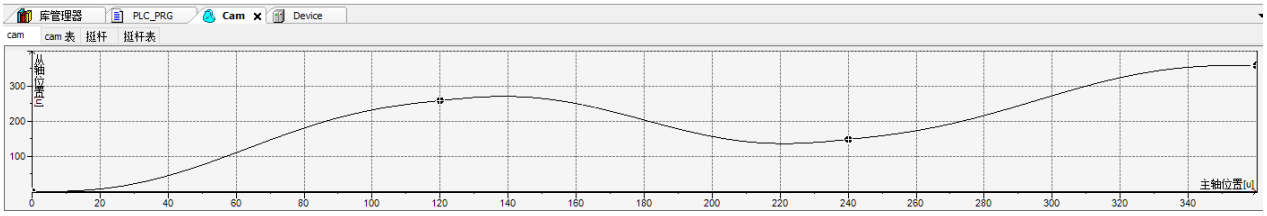
【1】新建 Cam 表（多项式）

右键【Application】->【添加对象】->【cam 表...】添加一个 Cam 表，此处命名为 Cam。

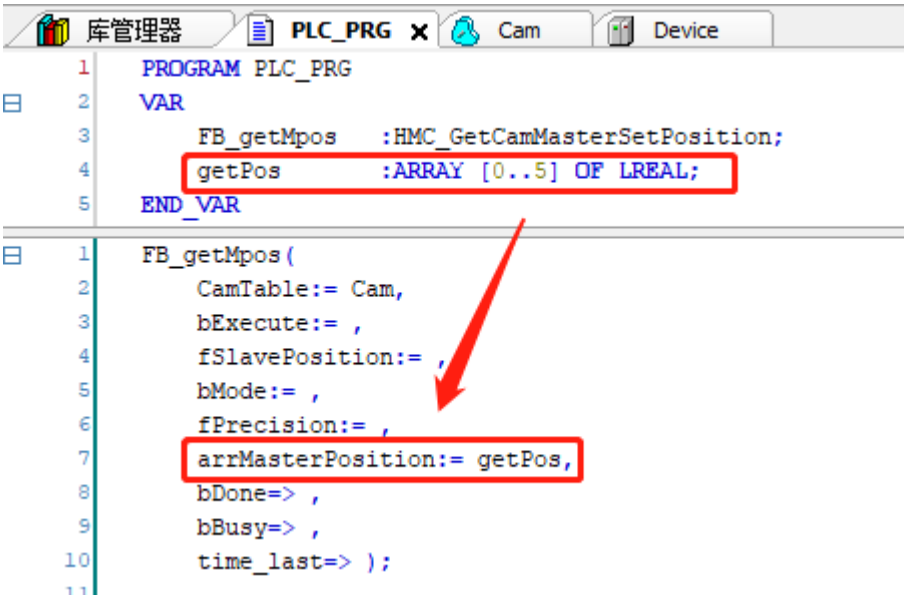




【2】修改 Cam 表中的主从轴位置对应关系，演示的 Cam 如下所示。（该表仅作演示本功能块计算使用。）

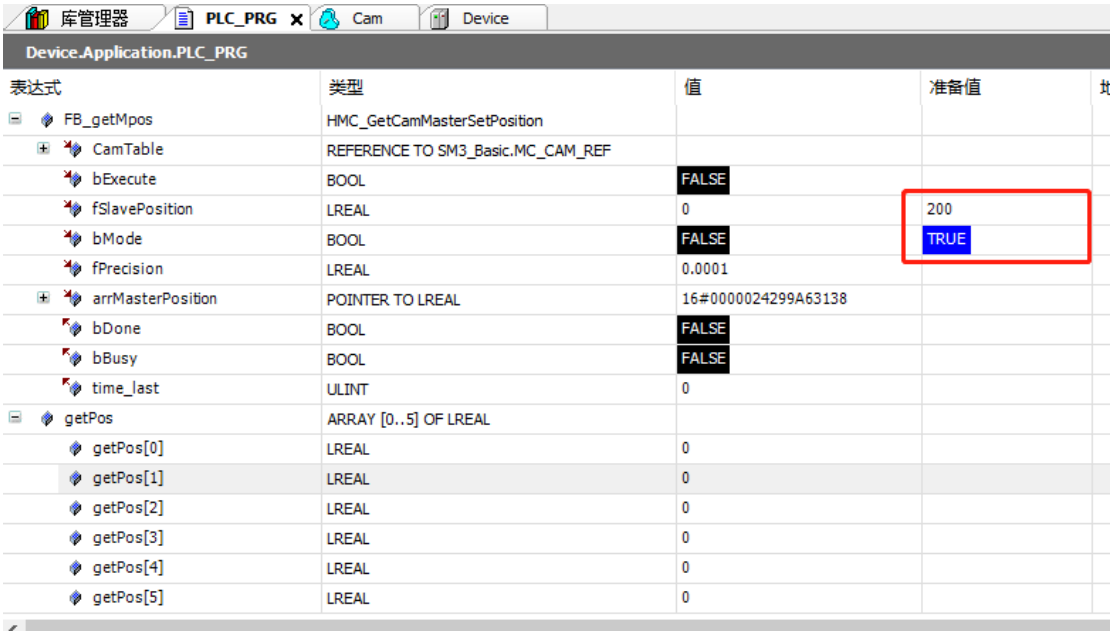


【3】声明并调用功能块，声明一个 LREAL 数组 getPos 用于接收功能块计算得到的主轴位置。



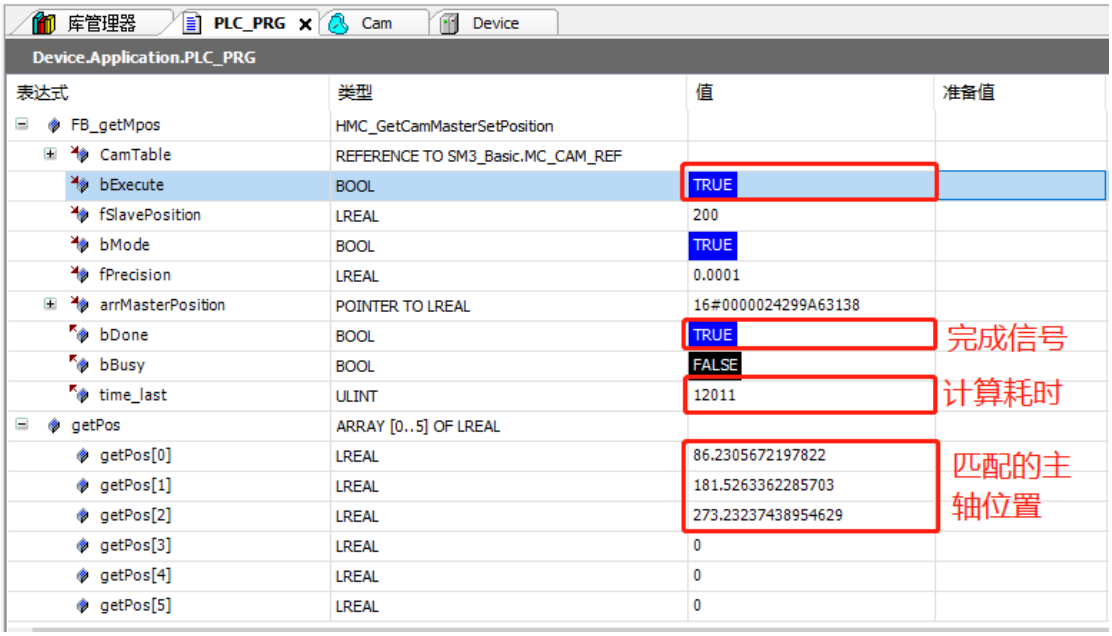
【4】使用功能块

举例演示计算从轴位置为 200 时，主轴的位置，计算方式为一个周期计算一次。



表达式	类型	值	准备值
FB_getMpos	HMC_GetCamMasterSetPosition		
CamTable	REFERENCE TO SM3_Basic.MC_CAM_REF		
bExecute	BOOL	FALSE	
fSlavePosition	LREAL	0	200
bMode	BOOL	FALSE	TRUE
fPrecision	LREAL	0.0001	
arrMasterPosition	POINTER TO LREAL	16#0000024299A63138	
bDone	BOOL	FALSE	
bBusy	BOOL	FALSE	
time_last	ULINT	0	
getPos	ARRAY [0..5] OF LREAL		
getPos[0]	LREAL	0	
getPos[1]	LREAL	0	
getPos[2]	LREAL	0	
getPos[3]	LREAL	0	
getPos[4]	LREAL	0	
getPos[5]	LREAL	0	

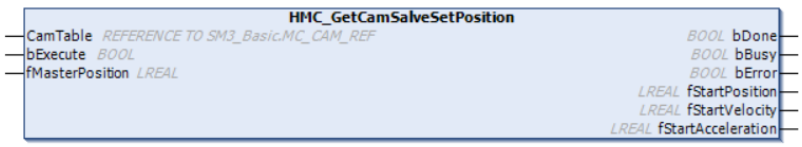
触发功能块后，功能块计算并输出匹配的所有的主轴位置，同时 bDone 引脚输出为 TRUE，time_last 引脚输出计算使用的时间。



5. 1. 2. HMC_GetCamSalveSetPosition (FB)

用于实现输入主轴位置算出从轴位置

变量

名称	HMC_GetCamMasterSetPosition		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	
		HMC_GetCamSalveSetPosition(CamTable:= , bExecute:= , fMasterPosition:= , bDone=> , bBusy=> , bError=> , fStartPosition=> , fStartVelocity=> , fStartAcceleration=>);	

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
CamTable	凸轮表	REFERENCE TO SM3_Basic.MC_CAM_REF			输入凸轮表，仅支持多项式模式
bExecute	使能	BOOL	TRUE、FALSE	FALSE	TRUE： 功能触发
fMasterPosition	主轴位置	LREAL			主轴位置

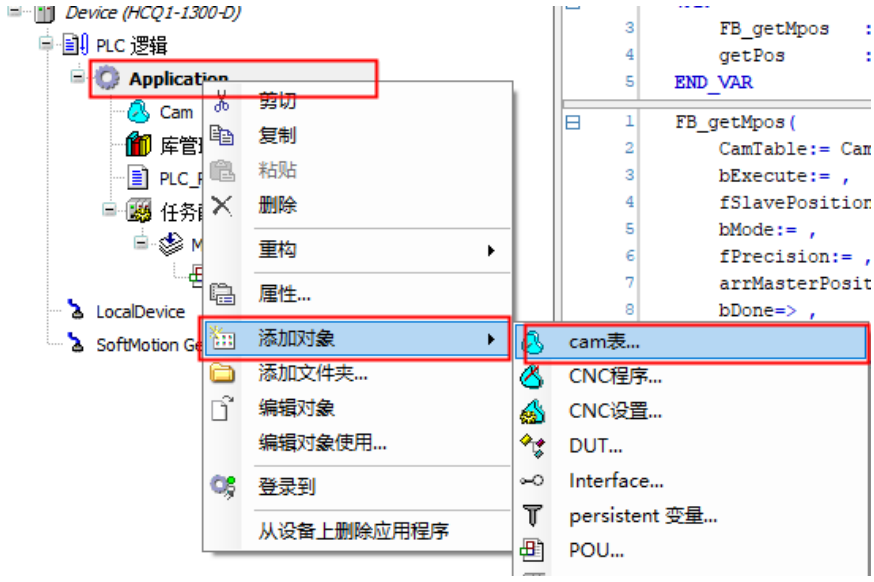
输出变量

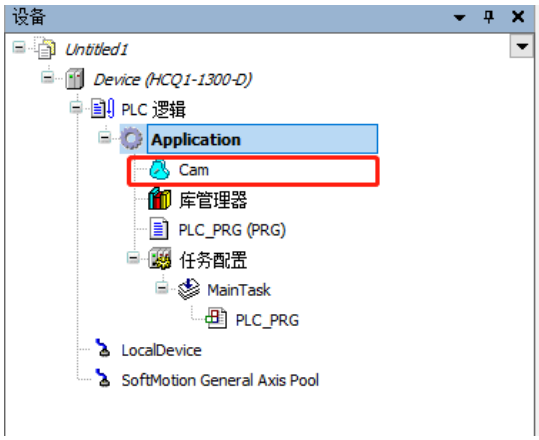
输出变量	名称	数据类型	有效范围	内容
bDone	完成	BOOL	TRUE、FALSE	TRUE:功能块执行完成
bBusy	运行中	BOOL	TRUE、FALSE	TRUE:功能块执行中
bError	错误	BOOL	TRUE、FALSE	TRUE:功能块报错
fStartPosition	从轴位置	LREAL		从轴位置
fStartVelocity	从轴速度	LREAL		从轴速度
fStartAcceleration	从轴加速度	LREAL		从轴加速度

使用举例

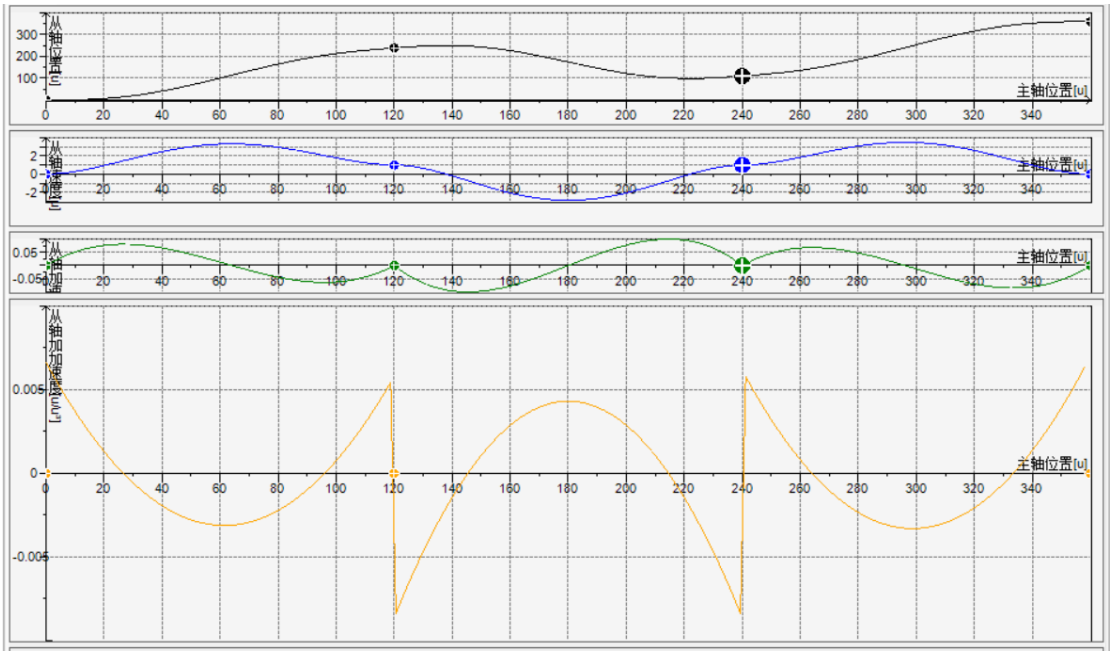
【1】新建 Cam 表（多项式）

右键【Application】->【添加对象】->【cam 表...】添加一个 Cam 表，此处命名为 Cam。



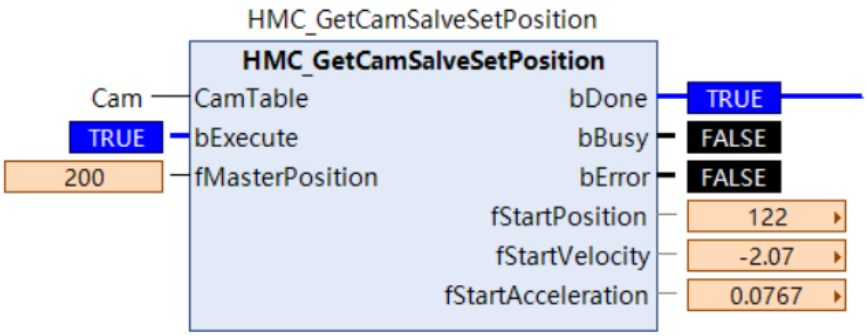


【2】修改 Cam 表中的主从轴位置对应关系，演示的 Cam 如下所示。（该表仅作演示本功能块计算使用。）



【3】声明并调用功能块，输入主轴位置计算得到从轴位置、速度、加速度。

举例演示计算主轴位置为 200 时，从轴参数。



5.2. GetVitualAxis

5.2.1. FB_CreatVitualAxis (FB)

用于创建一个虚拟轴

变量

名称	FB_CreatVitualAxis		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	


```
FB_CreatVitualAxis(
    bCreateEx:= ,
    bModulo:= ,
    fModuloPeriod:= ,
    bDone=> ,
    pAxis=> ,
    bError=> );
```

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bCreateEx	触发引脚	BOOL			升沿触发
bModulo	轴模态选择	BOOL	TRUE、FALSE	FALSE	TRUE:模态轴 FALSE:线性轴
fModuloPeriod	模态周期	REAL			模态轴周期

输出变量

输出变量	名称	数据类型	有效范围	内容
bDone	完成	BOOL	TRUE、FALSE	TRUE:功能块执行完成
pAxis	输出轴指针	POINTER TO AXIS_REF_VIRTUAL_SM3		成的虚轴指针
bError	错误	BOOL	TRUE、FALSE	TRUE:创建失败，请检查

要点说明

- 可以将计算方式设置为 TRUE 来降低对于任务周期的波动。但是计算点位会需要多个周期。
- arrMasterPosition 为指向 LREAL 数组首地址的指针，用于存放多个 LREAl 类型的数据（主轴位置）。

使用举例

声明功能块后，在程序中调用配置

```
(*生成虚轴,参数无误情况下，bCreateEx为TRUE后，bDone信号随即
给出，pAxis即为指向生成的虚轴的指针*)
FB_CreatVitualAxis(
    bCreateEx:=     TRUE,
    bModulo:=       bOn模态Off线性,
    fModuloPeriod:= f模态周期,
    bDone=> ,
    pAxis=> ,
    bError=> );
//---在所有轴控指令执行之前，调用m_Start()函数
FB_CreatVitualAxis.m_Start();
IF FB_CreatVitualAxis.bDone THEN    //---轴控功能块调用
    MC_Power(
        Axis:= FB_CreatVitualAxis.pAxis^,
        Enable:= TRUE,
        bRegulatorOn:= ,
        bDriveStart:= TRUE,
        Status=> ,
        bRegulatorRealState=> ,
        bDriveStartRealState=> ,
        Busy=> ,
        Error=> ,
        ErrorID=> );
    MC_Jog(
        Axis:= FB_CreatVitualAxis.pAxis^,
        JogForward:= bF,
        JogBackward:= bB,
        Velocity:= fVel,
        Acceleration:= fVel*10,
        Deceleration:= fVel*10,
        Jerk:= ,
        Busy=> ,
        CommandAborted=> ,
        Error=> ,
        ErrorId=> );
END_IF
//---在所有轴控指令之后调用m_End()函数;
FB_CreatVitualAxis.m_End();
```

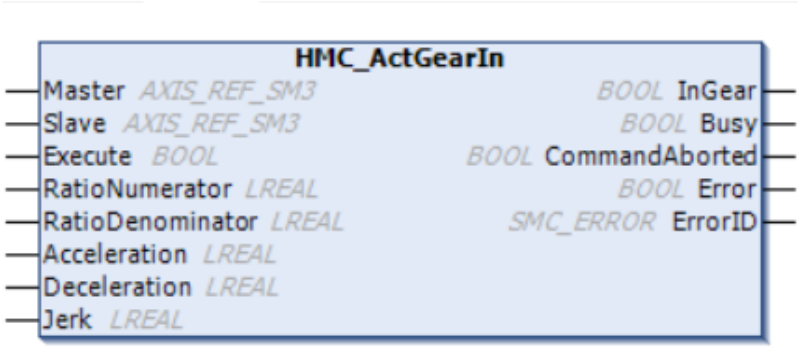
5.3. HMC_GearIn

5.3.1. HMC_ActGearIn (FB)

电子齿轮，耦合跟随主轴反馈位置

变量

名称	HMC_ActGearIn		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	




```
HMC_ActGearIn(
    Master:= ,
    Slave:= ,
    Execute:= ,
    RatioNumerator:= ,
    RatioDenominator:= ,
    Acceleration:= ,
    Deceleration:= ,
    Jerk:= ,
    InGear=> ,
    Busy=> ,
    CommandAborted=> ,
    Error=> ,
    ErrorID=> );
```

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bExecute	使能	BOOL	TRUE、FALSE	FALSE	TRUE： 功能触发
RatioNumerator	比例分子	LREAL			比例分子
RatioDenominator	比例分母	LREAL			比例分子母
Acceleration	加速度	LREAL			
Deceleration	减速度	LREAL			
Jerk	跃度	LREAL			

输出变量

输出变量	名称	数据类型	有效范围	内容
InGear	耦合	BOOL	TRUE、FALSE	TRUE:耦合完成
bBusy	运行中	BOOL	TRUE、FALSE	TRUE:功能块执行中
CommandAborted	打断	BOOL	TRUE、FALSE	TRUE:功能块被打断
Error	错误	BOOL	TRUE、FALSE	
ErrorID	错误代码	SMC_ERROR		

输入输出变量

输出变量	名称	数据类型	有效范围	内容
Master	主轴	AXIS_REF_SM3		主轴
Slave	从轴	AXIS_REF_SM3		从轴

要点说明

- 耦合跟随主轴反馈位置，使用方法与 MC_GearIn 相同。

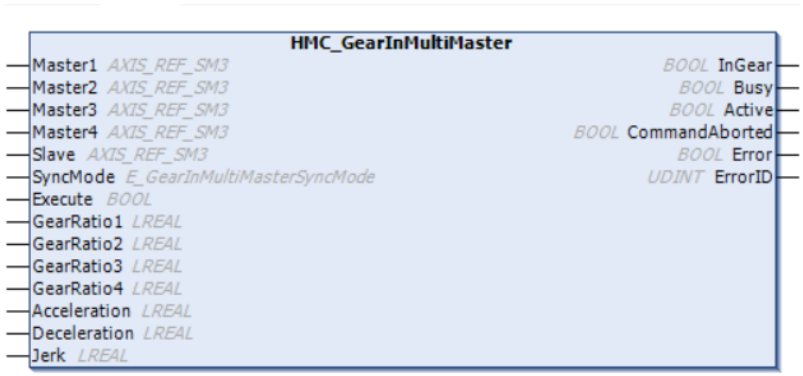
5.4. HMC_GearInMultiMaster

5.4.1. HMC_GearInMultiMaster (FB)

用于实现多主轴耦合

变量

名称	HMC_GetCamMasterSetPosition		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	




```
HMC_GearInMultiMaster(
    Master1:= ,
    Master2:= ,
    Master3:= ,
    Master4:= ,
    Slave:= ,
    SyncMode:= ,
    Execute:= ,
    GearRatio1:= ,
    GearRatio2:= ,
    GearRatio3:= ,
    GearRatio4:= ,
    Acceleration:= ,
    Deceleration:= ,
    Jerk:= ,
    InGear=> ,
    Busy=> ,
    Active=> ,
    CommandAborted=> ,
    Error=> ,
    ErrorID=> );
```

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
SyncMode	同步模式	E_GearInMultiMasterSyncMode			输入凸轮表，仅支持多项式模式
bExecute	使能	BOOL	TRUE、FALSE	FALSE	TRUE： 功能触发
GearRatio1	比例 1	LREAL			
GearRatio2	比例 2	LREAL			
GearRatio3	比例 3	LREAL			
GearRatio4	比例 4	LREAL			
Acceleration	加速度	LREAL			
Deceleration	减速度	LREAL			
Jerk	跃度	LREAL			

输出变量

输出变量	名称	数据类型	有效范围	内容
InGear	耦合同步	BOOL	TRUE、FALSE	TRUE:功能块耦合同步完成
bBusy	运行中	BOOL	TRUE、FALSE	TRUE:功能块执行中
Active	活动	BOOL	TRUE、FALSE	TRUE:功能块活动中
CommandAborted	打断	BOOL	TRUE、FALSE	TRUE:功能块被打断
Error	错误	BOOL	TRUE、FALSE	TRUE:功能块报错
ErrorID	错误 ID	UDINT		

输入输出变量

输出变量	名称	数据类型	有效范围	内容
Master1	主轴 1	AXIS_REF_SM3		主轴
Master2	主轴 2	AXIS_REF_SM3		主轴
Master3	主轴 3	AXIS_REF_SM3		主轴
Master4	主轴 4	AXIS_REF_SM3		主轴
Slave	从轴	AXIS_REF_SM3		从轴

要点说明

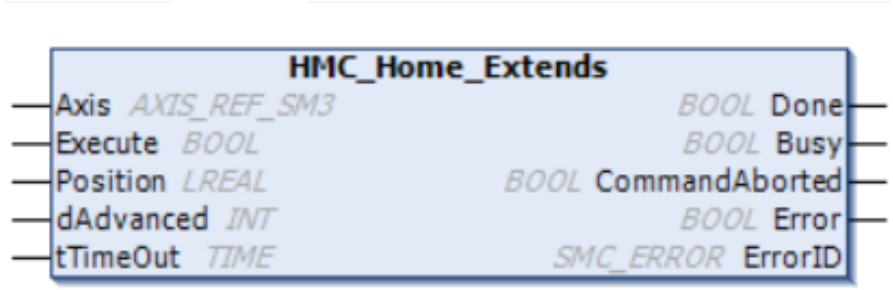
- E_GearInMultiMasterSyncMode.VELOSYNC，速度同步，耦合完成时从轴速度等于主轴速度*比例。
- E_GearInMultiMasterSyncMode.VELOSYNC，位置、速度同步，耦合完成时从轴速度（位置）等于主轴速度（位置）*比例。

5.5. HMC_Home

5.5.1. HMC_Home_Extends (FB)

禾川回原功能块，用于使用 MC_Home 出现回原问题时使用

变量

名称	HMC_Home_Extends		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	
		HMC_Home_Extends(Axis:= , Execute:= , Position:= , Done=> , Busy=> , CommandAborted=> , Error=> , ErrorID=> , dAdvanced:= , tTimeOut:=);	

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
Axis	轴	XIS_REF_SM3			
Execute	触发引脚	BOOL	TRUE、FALSE	FALSE	
Position	设置位置	LREAL		0	检测得到信号的绝对位置
dAdvanced	功能模式选择	INT	0；1；2		0:功能完全与 MC_Home 相同；1:超时后，尝试重启。2:超时后，先重置，再进行重启;
tTimeOut	超时时间	TIME		1S	回原动作超时判定

输出变量

输出变量	名称	数据类型	有效范围	内容
bDone	完成	BOOL	TRUE、FALSE	TRUE:功能块执行完成
Busy	运行信号	BOOL	TRUE、FALSE	TRUE:正在执行中
CommandAborted	中断信号	BOOL	TRUE、FALSE	TRUE:被其他命令终止
Error	错误	BOOL	TRUE、FALSE	TRUE:发生错误
ErrorID	错误 ID	SMC_ERROR	0	错误 ID

要点说明

- 与 MC_Home 功能块一样使用，填写“dAdvanced”“tTimeOut”相关参数即可。

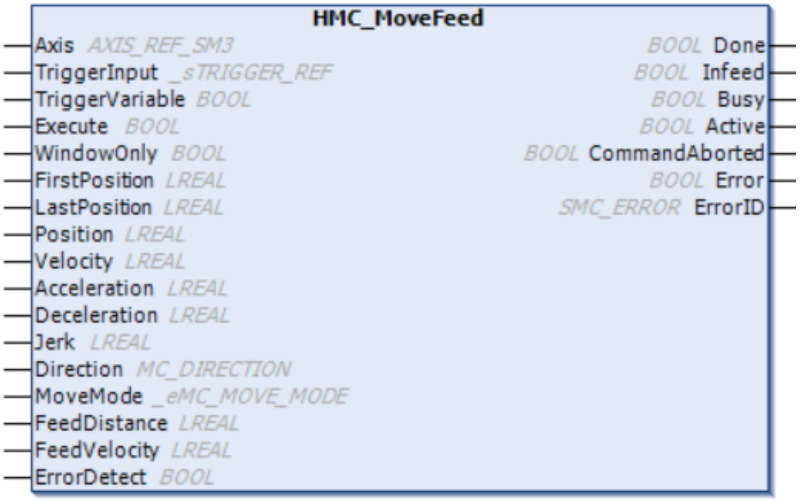
5.6. OMRONMotion

5.6.1. HMC_MoveFeed (FB)

指定自外部输入的中断输入发生位置起的移动距离，进行定位

变量

名称	HMC_MoveFeed		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	




```
HMC_MoveFeed(
    Axis:= ,
    TriggerInput:= ,
    TriggerVariable:= ,
    Execute:= ,
    WindowOnly:= ,
    FirstPosition:= ,
    LastPosition:= ,
    Position:= ,
    Velocity:= ,
    Acceleration:= ,
    Deceleration:= ,
    Jerk:= ,
    Direction:= ,
    MoveMode:= ,
    FeedDistance:= ,
    FeedVelocity:= ,
    ErrorDetect:= ,
    Done=> ,
    Infeed=> ,
    Busy=> ,
    Active=> ,
    CommandAborted=> ,
    Error=> ,
    ErrorID=> );
```

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bExecute	使能	BOOL	TRUE、FALSE	FALSE	TRUE： 功能触发
WindowOnly	窗口有效	BOOL			

FirstPosition	起始位置	LREAL			
LastPosition	结束位置	LREAL			
Position	目标位置	LREAL			
Velocity	目标速度	LREAL			
Acceleration	加速度	LREAL			
Deceleration	减速度	LREAL			
Jerk	跃度	LREAL			
Direction	方向选择	MC_DIRECTION			
MoveMode	移动方法选择	_eMC_MOVE_MODE			选择移动方法。 0：绝对值定位 1：相对值定位 2：速度控制
FeedDistance	标准距离	LREAL			指定中断输入后的移动距离。 沿着与中断输入前动作方向相同的方向使之按照标准距离动作时， 设为正数；反方向动作时，设为负数。
FeedVelocity	标准速度	LREAL			指定中断输入后的移动目标速度。
ErrorDetect	错误检测选择	BOOL			

输出变量

输出变量	名称	数据类型	有效范围	内容
Done	完成	BOOL	TRUE、FALSE	TRUE:功能块完成
Infeed	标准传送中	BOOL	TRUE、FALSE	接收锁定输入，标准传送中变为TRUE。
bBusy	运行中	BOOL	TRUE、FALSE	TRUE:功能块执行中
Active	活动	BOOL	TRUE、FALSE	TRUE:功能块活动中
CommandAborted	打断	BOOL	TRUE、FALSE	TRUE:功能块被打断
Error	错误	BOOL	TRUE、FALSE	TRUE:功能块报错
ErrorID	错误 ID	UDINT		

输入输出变量

输出变量	名称	数据类型	有效范围	内容
Axis	轴	AXIS_REF_SM3		
TiggerInput	触发条件	_sTRIGGER_REF		
TiggerVariable	触发变量	BOOL		在触发条件下指定控制器模式时，指定触发的输入变量

要点说明

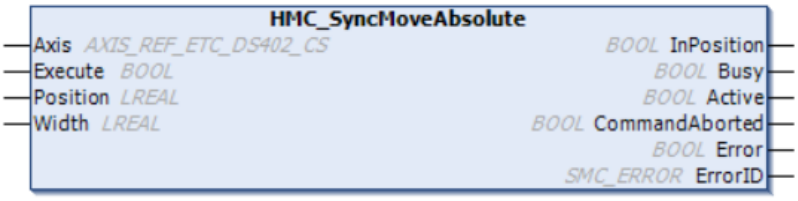
- _eMC_TRIGGER_MODE（0: _mcDrive，1: _mcController 指定触发模式。0：驱动器模式 1：控制器模式）
- LatchID 锁定 ID 选择：_eMC_TRIGGER_LATCHID（0: _mcLatch1，1: _mcLatch2， 0：锁定功能 1，1：锁定功能 2）
- PDO 映射：锁定功能(60B8Hex) 锁定状态(60B9Hex) 锁定位置 1(60BAHex) 锁定位置 2(60BCHex)
- 在 Execute(启动)的上升沿，根据 MoveMode(移动方法选择)的设定，按照绝对值移动、相对值移动或速度控制中的某一移动方法进行移动。
- 采用绝对值移动时，在 Position(目标位置)中设定目标位置；采用相对值移动时，在 Position(目标位置)中设定目标距离。无论何种移动方法，均以 Velocity(目标速度)进行移动动作。
- 移动过程中，在外部输入(中断输入)的上升沿进行相对定位动作。以 FeedVelocity(标准速度)，从反馈位置起，移动 FeedDistance(标准距离)指定的标准距离。
- 利用绝对值移动或相对值移动指令进行中断标准传送，在到达目标位置前未输入中断信号时，在当初的目标位置停止动作。无中断输入而停止动作时，通过 ErrorDetect(错误检测选择)，可指定有无异常输出。指定异常输出时，CommandAborted(执行中断)变为 TRUE，Busy(执行中)、Active(控制中)变为 FALSE。
- 使用中断屏蔽时，将 WindowOnly(窗口有效)设为 TRUE，指定 FirstPositon(起始位置)、LastPositon(终止位置)。通过反馈位置从 FirstPositon(起始位置)到 LastPosition(终止位置)之间发生的最初中断信号，执行中断标准定位。

5.6.2. HMC_SyncMoveAbsolute (FB)

按周期输出轴的指定目标位置

变量

名称	HMC_SyncMoveAbsolute		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	




```
HMC_SyncMoveAbsolute(
    Axis:= ,
    Execute:= ,
    Position:= ,
    Width:= ,
    InPosition=> ,
    Busy=> ,
    Active=> ,
    CommandAborted=> ,
    Error=> ,
    ErrorID=> );
```

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bExecute	使能	BOOL	TRUE、FALSE	FALSE	TRUE： 功能触发
Position		LREAL			
Width		LREAL			

输出变量

输出变量	名称	数据类型	有效范围	内容
InPosition	到位	BOOL	TRUE、FALSE	TRUE:功能块到位完成
bBusy	运行中	BOOL	TRUE、FALSE	TRUE:功能块执行中
Active	活动	BOOL	TRUE、FALSE	TRUE:功能块活动中
CommandAborted	打断	BOOL	TRUE、FALSE	TRUE:功能块被打断
Error	错误	BOOL	TRUE、FALSE	TRUE:功能块报错
ErrorID	错误 ID	SMC_ERROR		

输入输出变量

输出变量	名称	数据类型	有效范围	内容
Axis	轴	AXIS_REF_SM3		

要点说明

- 本指令按照任务周期、以周期同步位置模式(CSP)将用户程序给定的目标位置输出到伺服驱动器等,目标位置以绝对位置指定。
- 不更新 Position(目标位置)时，如果目标位置和反馈位置的幅度在轴参数[到位宽度]的范围内，则 InPosition(到位)变为 TRUE。

5.7. OverrideVel

5.7.1. HMC_Jog (FB)

通过改变速比值变速

变量

名称	HMC_Jog（可变速点动）		
支持的模式	CSP	CSV	
图形表现		ST 表现	


```
HMC_Jog(
    Axis:= ,
    JogForward:= ,
    JogBackward:= ,
    SpeedRatio:= ,
    Velocity:= ,
    Acceleration:= ,
    Deceleration:= ,
    Jerk:= ,
    Busy=> ,
    CommandAborted=> ,
    Error=> ,
    ErrorId=> );
```

输入输出变量

输入输出变量	名称	数据类型	内容
Axis	轴	AXIS_REF_SM3	指定轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
JogForward	正向点动	BOOL	TRUE、FALSE	FALSE	TRUE：轴向正方向移动 FALSE：轴停止向正方向移动
JogBackward	反向点动	BOOL	TRUE、FALSE	FALSE	TRUE：轴向反方向移动 FALSE：轴停止向反方向移动
SpeedRatio	速度比	LREAL	“0”～“1”	0	轴实际运动的速度占目标速度的比例
Velocity	目标速度	LREAL	正数或“0”	0	指定目标速度
Acceleration	加速度	LREAL	正数或“0”	0	加速度
Deceleration	减速度	LREAL	正数或“0”	0	减速度
Jerk	加加速度	LREAL	正数或“0”	0	指定加加速度

输出变量

输出变量	名称	数据类型	有效范围	内容
Busy	功能块执行中	BOOL	TRUE、FALSE	TRUE：功能块运行中
CommandAborted	功能块执行中断	BOOL	TRUE、FALSE	TRUE：功能块被中止
Error	错误	BOOL	TRUE、FALSE	TRUE：功能块产生异常，已停止执行
ErrorID	错误代码	SMC_ERROR	0	发生异常是，输出错误代码

要点说明

- 控制轴点动运行，正向点动由 JogForward 控制，设置为 TRUE 轴即会按照设定的速度、速度比、加速度进行正向点动。反向点动由 JogBackward 控制，设置为 TRUE 轴即会按照设定的速度、速度比、加速度进行反向点动。
- 速度比 SpeedRatio 在轴运动过程中更改也会即时生效，可以修改 SpeedRatio 来实时改变轴的当前运动速度占目标速度的比例。
- 点动运行时，如果 JogForward 或者 JogBackwar 从 TRUE 变为 FALSE，轴便会立即按照设定的减速度减速停止。
- 点动过程中轴处于 Continuous Motion 状态。
- 如果 JogForward 和 JogBackward 同时设置为 TRUE，轴将停止运动或不开始运动且不会报错。

5.7.2. HMC_Jogs (FB)

HMC_Jog 功能块的变形，直接通过改变速度值变速

变量

名称	HMC_Jogs（可变速点动）		
支持的模式	CSP	CSV	
图形表现		ST 表现	


```
HMC_Jogs(
    Axis:= ,
    JogForward:= ,
    JogBackward:= ,
    Velocity:= ,
    Acceleration:= ,
    Deceleration:= ,
    Jerk:= ,
    Busy=> ,
    CommandAborted=> ,
    Error=> ,
    ErrorId=> );
```

输入输出变量

输入输出变量	名称	数据类型	内容
Axis	轴	AXIS_REF_SM3	指定轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
JogForward	正向点动	BOOL	TRUE、FALSE	FALSE	TRUE：轴向正方向移动 FALSE：轴停止向正方向移动
JogBackward	反向点动	BOOL	TRUE、FALSE	FALSE	TRUE：轴向反方向移动 FALSE：轴停止向反方向移动
Velocity	目标速度	LREAL	正数或“0”	0	指定目标速度
Acceleration	加速度	LREAL	正数或“0”	0	加速度
Deceleration	减速度	LREAL	正数或“0”	0	减速度
Jerk	加加速度	LREAL	正数或“0”	0	指定加加速度

输出变量

输出变量	名称	数据类型	有效范围	内容
Busy	功能块执行中	BOOL	TRUE、FALSE	TRUE：功能块运行中
CommandAborted	功能块执行中断	BOOL	TRUE、FALSE	TRUE：功能块被中止
Error	错误	BOOL	TRUE、FALSE	TRUE：功能块产生异常，已停止执行
ErrorID	错误代码	SMC_ERROR	0	发生异常是，输出错误代码

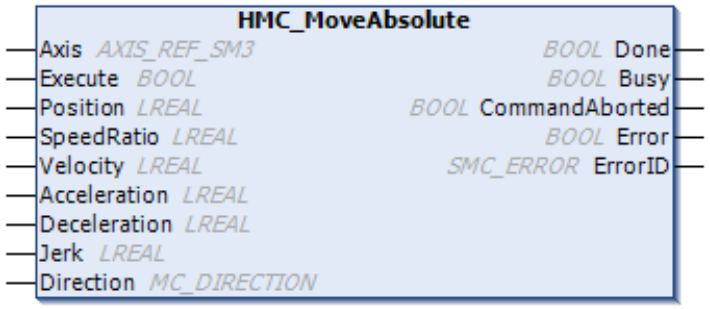
要点说明

- 在轴运动过程中更改也会即时生效，通过修改 Velocity 来实时改变轴的当前运动速度。
- 其他注意事项参考 HMC_jog 功能块描述

5.7.3. HMC_MoveAbsolute (FB)

变量

名称	HMC_MoveAbsolute(可变速绝对定位)		
支持的模式	CSP		
图形表现		ST 表现	




```
HMC_MoveAbsolute(
    Axis:= ,
    Execute:= ,
    Position:= ,
    SpeedRatio:= ,
    Velocity:= ,
    Acceleration:= ,
    Deceleration:= ,
    Jerk:= ,
    Direction:= ,
    Done=> ,
    Busy=> ,
    CommandAborted=> ,
    Error=> ,
    ErrorID=> );
```

输入输出变量

输入变量	名称	数据类型	内容
Axis	轴	AXIS_REF_SM3	指定轴，即 AXIS_REF_SM3 的一个实例

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
Execute	启动	BOOL	TRUE、FALSE	FALSE	TRUE：启用功能块
Position	目标位置	LREAL	负数、正数、“0”	0	指定绝对坐标的目标位置，单位为【指令单位】
Velocity	目标速度	LREAL	正数	0	指定目标速度
SpeedRatio	速度比	LREAL	“0”～“1”	0	轴实际运动的速度占目标速度的比例
Velocity	目标速度	LREAL	正数或“0”	0	指定目标速度
Acceleration	加速度	LREAL	正数或“0”	0	加速度
Deceleration	减速度	LREAL	正数或“0”	0	减速度
Jerk	加加速度	LREAL	正数或“0”	0	指定加加速度
Direction	方向选择	MC_Direction	Fastest，current，positive，shortest，negative	Shortest	参考 MC_Direction

输出变量

输出变量	名称	数据类型	有效范围	内容
Done	完成	BOOl	TRUE、FALSE	TRUE：功能块执行完成
Busy	功能块执行中	BOOL	TRUE、FALSE	TRUE：功能块运行中
CommandAborted	功能块执行中断	BOOL	TRUE、FALSE	TRUE：功能块被中止
Error	错误	BOOL	TRUE、FALSE	TRUE：功能块产生异常，已停止执行
ErrorID	错误代码	SMC_ERROR	0	发生异常是，输出错误代码

要点说明

- 本功能块为绝对定位功能块，Position 数据为轴的绝对位置，本功能块执行时轴的状态为 Discrete Motion。
- 速度比 SpeedRatio 在轴运动过程中更改也会即时生效，可以修改 SpeedRatio 来实时改变轴的当前运动速度占目标速度的比例。
- Execute 的上升沿为启动功能块，功能块执行中可以重新触发上升沿，每次上升沿都会重新载入功能块的输入参数，重新执行功能块。
- Acceleration 或 Deceleration 为零，功能块启动（Execute）会报错，轴的状态为 Standstill 。
- Direction(方向选择)，参考 MC_Direction。

5.8. RobotMove (功能组)

5.8.1. 插补模型及模型功能块

5.8.1.1. FB_KimTransl_None2 (无模型 2 轴模型)

无模型 2 轴类似于 2 轴龙门平面结构，轴 1 控制水平面横向运动(X 轴)，轴 2 控制水平面纵向运动(Y 轴)。

```
FB_KimTransl_None2(
        bWriteParameter:= ,        //写入模型以及轴配置参数
        outCartesianPos=> ,        //输出空间坐标
        bDone=> ,                  //模型参数配置完成信号
        bError=> ,                 //模型参数写入错误信号
        dOffsetA0:= ,              //A0轴的额外偏移量
        dOffsetA1:= ,              //A1轴的额外偏移量
        AxisX:= ,                  //水平X轴
        AxisY:= );                 //水平Y轴
```

5.8.1.2. FB_KimTransl_None3 (无模型 3 轴模型)

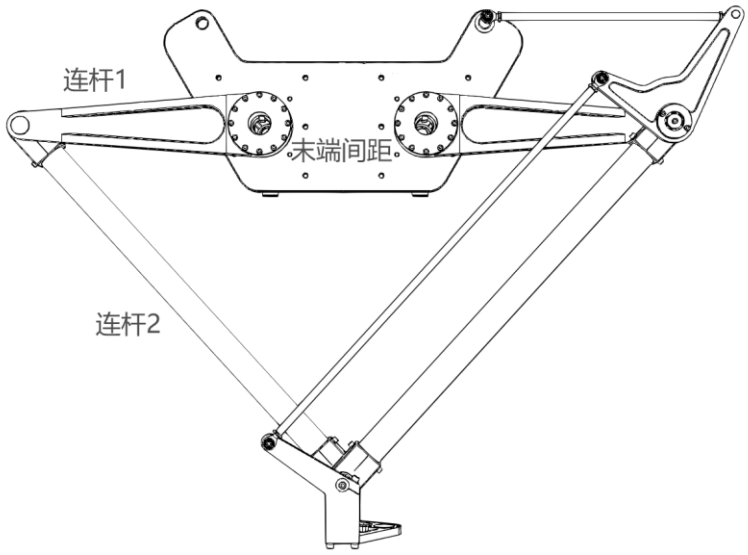
无模型 3 轴类似 3 轴龙门空间结构，在 2 轴的基础上增加一个轴 3 控制空间 Z 轴上的垂直运动。

```
FB_KimTransl_None3(
        bWriteParameter:= ,        //写入模型以及轴配置参数
        outCartesianPos=> ,        //输出空间坐标
        bDone=> ,                  //模型参数配置完成信号
        bError=> ,                 //模型参数写入错误信号
        dOffsetA0:= ,              //A0轴的额外偏移量
        dOffsetA1:= ,              //A1轴的额外偏移量
        dOffsetA2:= ,              //A2轴的额外偏移量
        AxisX:= ,                  //空间X轴
        AxisY:= ,                  //空间Y轴
        AxisZ:= );                 //空间Z轴
```

5.8.1.3. FB_KimTransl_Delta2 (2 轴 Delta 模型)

2 轴 Delta 机器人模型如下图所示，左右轴(模态)分别控制左右连杆 1(统一定义为连杆 1)的运动，需要在功能块中设定连杆 1 的长度(dArmLength1)，连杆 2 的长度(dArmLength2)，两个连杆 1 水平时的末端点距离(dDistance)，三个参数，默认两个连杆 1 处于水平位置时，机器人末端点所在的空间位置为(0,0)。

```
FB_KimTransl_Delta2(
        bWriteParameter:= ,        //写入模型以及轴配置参数
        outCartesianPos=> ,        //输出空间坐标
        bDone=> ,                  //模型参数配置完成信号
        bError=> ,                 //模型参数写入错误信号
        dArmLength1:= ,            //连杆1长度
        dArmLength2:= ,            //连杆2长度
        dDistance:= ,              //两个连杆1的末端距离
        dOffsetA0:= ,              //A0轴的额外偏移量
        dOffsetA1:= ,              //A1轴的额外偏移量
        Axis0:= ,                  //左轴，（在坐标系中，位置在X轴的负半轴的为左轴）
        Axis1:= );                 //右轴，（在坐标系中，位置应在X轴的正半轴为右轴）
```

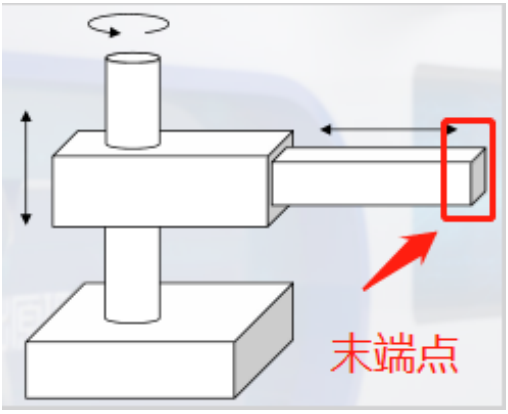


5.8.1.4. FB_KimTransl_Polar2_Z (3 轴 Polar 圆柱坐标模型)

轴 0 控制旋转轴，轴 1 控制水平伸缩轴，轴 2 控制空间 Z 轴的垂直移动。

Ps：末端点在空间上考研到达的点的集合是一个圆柱体，所以称为圆柱体坐标系模型。

```
FB_KimTransl_Polar2_Z(
        bWriteParameter:= ,        //写入模型以及轴配置参数
        outCartesianPos=> ,        //输出空间坐标
        bDone=> ,                  //模型参数配置完成信号
        bError=> ,                 //模型参数写入错误信号
        dOffsetA0:= ,              //A0轴（旋转C轴）的额外偏移量
        dOffsetA1:= ,              //A1轴（线性R轴）的额外偏移量
        dMaxR:= ,                  //A1轴（线性R轴）的最大距离
        Axis0:= ,                  //A0轴（旋转C轴）
        Axis1:= ,                  //A1轴（线性R轴）
        Axis2:= );                 //A2轴（升降Z轴）
```



5.8.1.5. FB_KimTransl_Scara2_Z_Tool (4 轴 Scara2 机器人模型)

4 轴 Scara 机器人模型如图所示，轴 Axis0（模态）控制大臂旋转，轴 Axis1（模态）控制小臂旋转，轴 Axis2（线性）控制辅助轴空间上下运动，轴 Axis3（模态）控制工具轴旋转，逆时针旋转为正方向。

需要在功能块中设定大臂的长度（dArmLength1），小臂的长度（dArmLength2），肘部姿态 bElbowlow（TRUE 为右手，FALSE 为左手）三个参数，仿真情况下默认大臂小臂都处于空间 x 轴正半轴上，即机械臂伸直，指向 x 轴正半轴。辅助轴未上下移动，工具轴旋转角度为 0，此时末端点位处于空间坐标（0,0,0），工具轴旋转角度为 0 度。

```
FB_KimTransl_Scara2_Z_Tool(
        bWriteParameter:=,          //写入模型以及轴配置参数
        outCartesianPos=>,          //输出空间坐标
        bDone=>,                    //模型参数配置完成信号
        bError=>,                   //模型参数写入错误信号
        dArmLength1:=,              //连杆1长度（大臂）
        dArmLength2:=,              //连杆2长度（小臂）
        dOffsetA0:=,                //A0轴的额外偏移量
        dOffsetA1:=,                //A1轴的额外偏移量
        bElbowLow:=,                //肘部高低设定，TRUE为低右手系，FALSE为高左手系
        Axis0:=,                    //大臂轴
        Axis1:=,                    //小臂轴
        Axis2:=,                    //辅助Z轴
        Axis3:=);                   //工具R轴
```

5.8.1.6. FB_KimTransl_Scara2_Z_Tool_ABS (4 轴 Scara2 机器人绝对值模型)

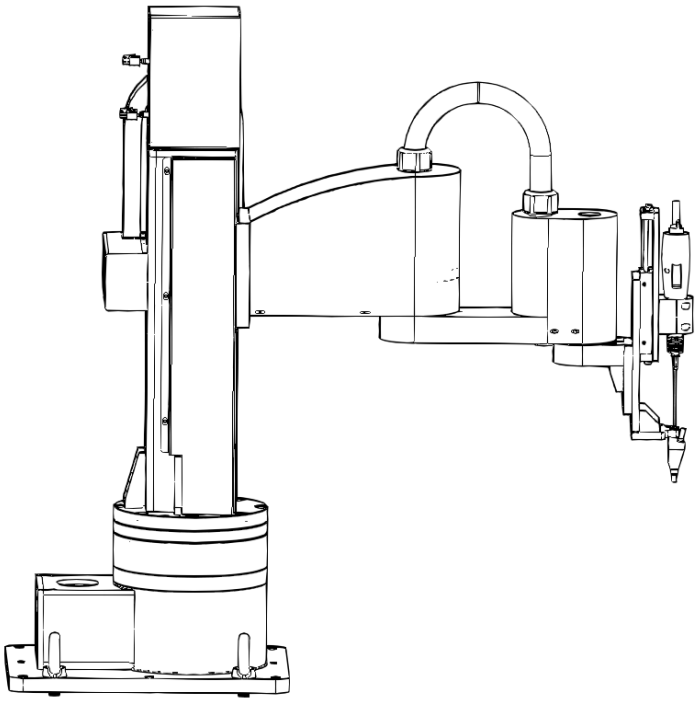
是 FB_KimTransl_Scara2_Z_Tool 模型的绝对值版本，伺服 0 点即空间坐标（L1+L1,0,0）姿态(0,0,0)。

```
FB_KimTransl_Scara2_Z_Tool_ABS(
        bWriteParameter:= ,        //写入模型以及轴配置参数
        outCartesianPos=> ,        //输出空间坐标
        bDone=> ,                  //模型参数配置完成信号
        bError=> ,                 //模型参数写入错误信号
        dArmLength1:= ,            //连杆1长度（大臂）
        dArmLength2:= ,            //连杆2长度（小臂）
        bElbowLow:= ,              //肘部高低设定，TRUE为低右手系，FALSE为高左手系
        Axis0:= ,                  //大臂轴
        Axis1:= ,                  //小臂轴
        Axis2:= ,                  //辅助Z轴
        Axis3:= );                 //工具R轴
```

5.8.1.7. FB_KimTransl_Scara3_Z (三关节 Scara 模型)

是 FB_KimTransl_Scara2_Z_Tool 模型的变种模型，参考如下图所示

```
FB_KimTransl_Scara3_Z(
        bWriteParameter:=,          //写入模型以及轴配置参数
        outCartesianPos=>,          //输出空间坐标
        bDone=>,                    //模型参数配置完成信号
        bError=>,                   //模型参数写入错误信号
        dArmLength1:=,              //连杆1长度（大臂）
        dArmLength2:=,              //连杆2长度（小臂）
        dArmLength3:=,              //连杆3长度
        bElbowLow:=,                //肘部高低设定，TRUE为低右手系，FALSE为高左手系
        Axis0:=,                    //大臂轴
        Axis1:=,                    //小臂轴
        Axis2:=,                    //辅助Z轴
        Axis3:= );                  //工具R轴
```

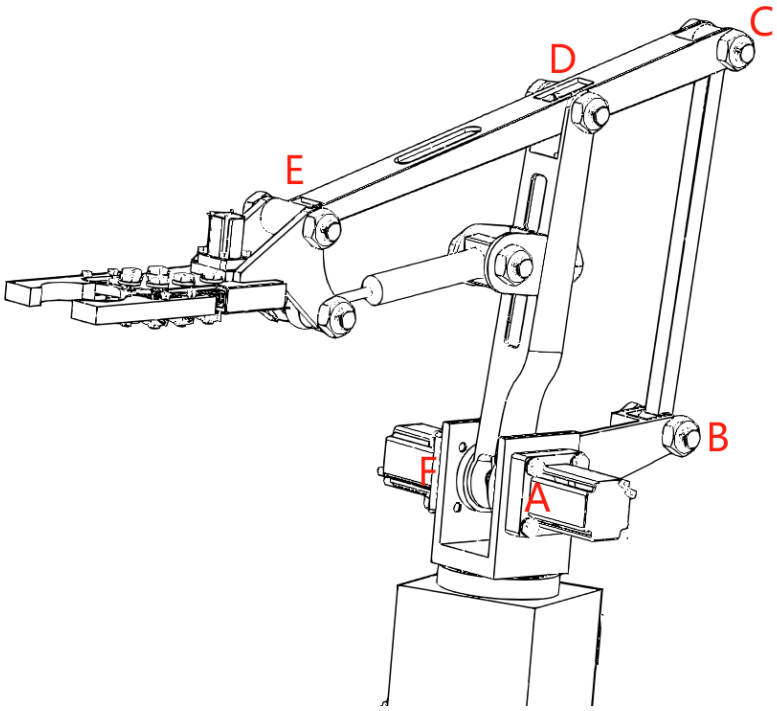


5.8.1.8. FB_KimTransl_SimilarScara2 (类 Scara 模型)

此模型是 Scara 模型的变种模型，参考如下图所示：

Ps：A、F 处是两部分由两个电机控制。FD 为大臂轴长度；EC 为小臂轴长度；CB 为中间支撑边长度；DC 为末端边长度；AB 为末端边上大臂轴到中间支撑边长度。

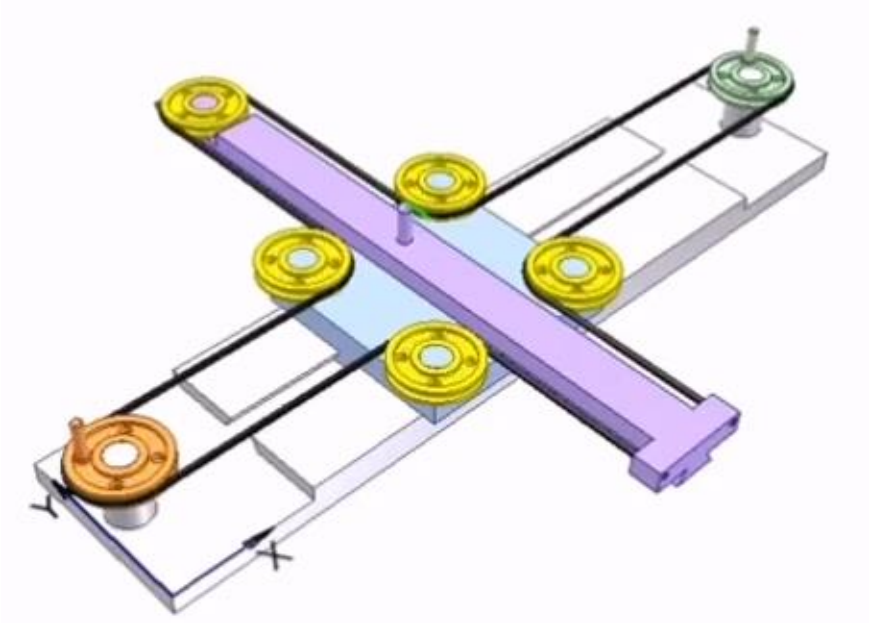
```
FB_KimTransl_SimilarScara2(
        bWriteParameter:=,          //写入模型以及轴配置参数
        outCartesianPos=>,          //输出空间坐标
        bDone=>,                    //模型参数配置完成信号
        bError=>,                   //模型参数写入错误信号
        dOffsetJ1:=,                //J1轴的额外偏移量
        dOffsetJ2:=,                //J2轴的额外偏移量
        dArmLength1:=,              //大臂轴长度
        dArmLength2:=,              //小臂轴长度
        dMidLength:=,               //中间支撑边长度
        dFinalLength:=,             //末端边长度
        dAxis1ToMidLength:=,        //末端边上大臂轴到中间支撑边长度
        AxisJ1:=,                   // J1轴
        AxisJ2:= );                 //J2轴
```



5.8.1.9. FB_KimTransl_Trapezoid2 (2 轴 T 型机械手)

此结构类似名称叫 CoreXY 结构或者 H_Bot 结构。

```
FB_KimTransl_Trapezoid2(
        bWriteParameter:= ,          //写入模型以及轴配置参数
        outCartesianPos=> ,          //输出空间坐标
        bDone=> ,                    //模型参数配置完成信号
        bError=> ,                   //模型参数写入错误信号
        dOffsetX:= ,                 //空间坐标X起始偏移量
        dOffsetY:= ,                 //空间坐标Y起始偏移量
        AxisA0:= ,                   //A0轴
        AxisA1:= );                  //A1轴
```



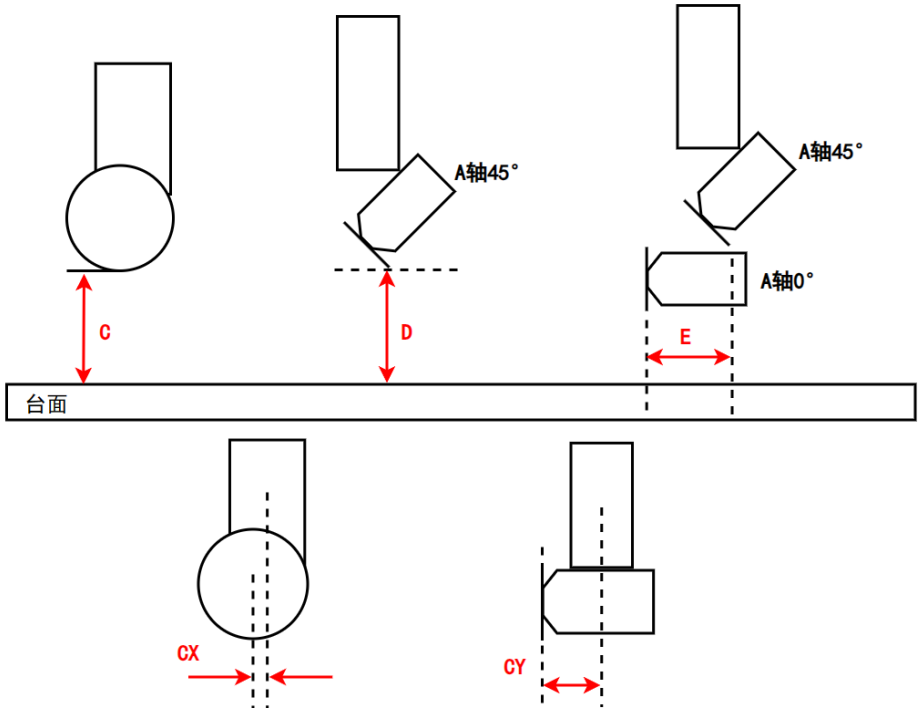
5.8.1.10. FB_KimTransl_GantryCutter2 (二维龙门加切线)

```
FB_KimTransl_GantryCutter2(
        bWriteParameter:= ,          //写入模型以及轴配置参数
        outCartesianPos=> ,          //输出空间坐标
        bDone=> ,                    //模型参数配置完成信号
        bError=> ,                   //模型参数写入错误信号
        dOffsetX:= ,                 //A0轴的额外偏移量
        dOffsetY:= ,                 //A1轴的额外偏移量
        dOffsetR:= ,                 //A3轴的额外偏移量
        AxisX:= ,                    //X轴
        AxisY:= ,                    //Y轴
        AxisR:= );                   //R轴
```

5. 8. 1. 11. FB_KimTransl_GantryCutter3 (三维龙门加切线)

```
FB_KimTransl_GantryCutter3(
        bWriteParameter:= ,        //写入模型以及轴配置参数
        outCartesianPos=> ,        //输出空间坐标
        bDone=> ,                  //模型参数配置完成信号
        bError=> ,                 //模型参数写入错误信号
        dOffsetX:= ,               //A0轴的额外偏移量
        dOffsetY:= ,               //A1轴的额外偏移量
        dOffsetZ:= ,               //A2轴的额外偏移量
        dOffsetR:= ,               //A3轴的额外偏移量
        AxisX:= ,                  //X轴
        AxisY:= ,                  //Y轴
        AxisZ:= ,                  //Z轴
        AxisR:= );                 //R轴
```

5. 8. 1. 12. FB_KimTransl_Axis4 (4 轴桥切机)



直切 Z 平台差：测量距离如图 C，此时 Z 轴在零位。

斜切 Z 平台差：测量距离如图 D，此时 Z 轴在零位、A 轴 45°

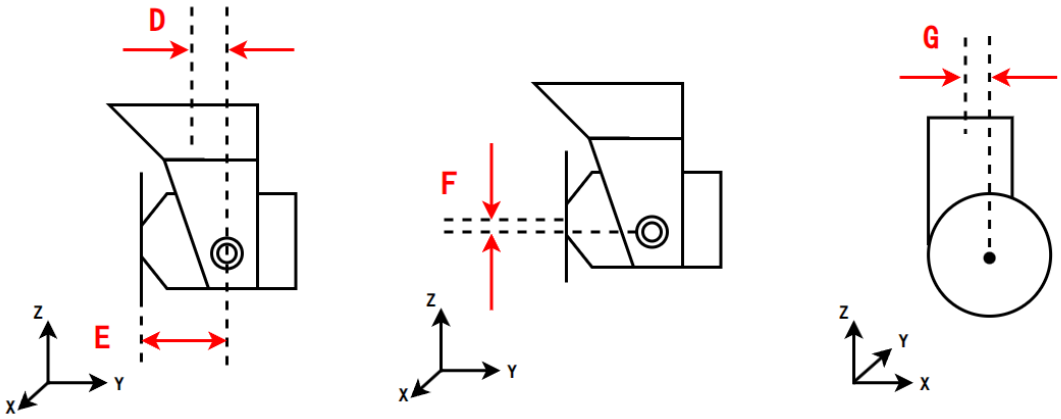
斜切 Y 差：测量距离如图 E，A 轴 45°和 0°时在 Y 轴上的投影间距

C 轴心偏差 X：测量距离如图 CX

C 轴心偏差 Y：测量距离如图 CY

```
FB_KimTransl_Axis4(
        bWriteParameter:=,          //写入模型以及轴配置参数
        outCartesianPos=>,          //输出空间坐标
        bDone=>,                    //模型参数配置完成信号
        bError=>,                   //模型参数写入错误信号
        fZ1:=,                      //直切Z平台差
        fZ2:=,                      //斜切Z平台差
        frotateA_Y:=,               //斜切Y差
        frotateC_X:=,               //C轴心偏差X
        frotateC_Y:=,               //C轴心偏差Y
        Apos:=,                     //A轴角度
        dOffsetX:=,                 //X轴的额外偏移量
        dOffsetY:=,                 //Y轴的额外偏移量
        dOffsetZ:=,                 //Z轴的额外偏移量
        AxisX:=,                    //X轴
        AxisY:=,                    //Y轴
        AxisZ:=,                    //Z轴
        AxisC:=);                   //C轴
```

5. 8. 1. 13. FB_KimTransl_Axis5 (五轴桥切机)



A 轴旋转中心与 C 轴旋转中心偏移：测量距离如图 D

锯片内侧和 A 轴旋转中心偏移：测量距离如图 E

锯片中心和 A 轴旋转中心高度差：测量距离如图 F

锯片中心和 C 轴旋转中心偏移：测量距离如图 G

锯片厚度：根据具体刀的不同输入合适的厚度数值

```
FB_KimTransl_Axis5(
        bWriteParameter:= ,          //写入模型以及轴配置参数
        outCartesianPos=> ,          //输出空间坐标
        bDone=> ,                    //模型参数配置完成信号
        bError=> ,                   //模型参数写入错误信号
        farm1:= ,                    //A轴旋转中心与C旋转中心偏移
        farm2:= ,                    //锯片内侧和A轴旋转中心偏移
        fsawW:= ,                    //锯片厚度
        fsawR:= ,                    //锯片半径R
        fsawZ:= ,                    //锯片中心和A轴旋转中心高度差
        fsawC:= ,                    //锯片中心和C旋转中心偏移
        dOffsetX:= ,                 //X轴的额外偏移量
        dOffsetY:= ,                 //Y轴的额外偏移量
        dOffsetZ:= ,                 //Z轴的额外偏移量
        AxisX:= ,                    //X轴
        AxisY:= ,                    //Y轴
        AxisZ:= ,                    //Z轴
        AxisA:= ,                    //A轴
        AxisC:= );                   //C轴
```

5.8.2. 运动控制功能块

5.8.2.1. HMC_RobotHandWheel (手摇轮空间 Jog 功能块)

```
HMC_RobotHandWheel(
        bEnable:=,                  // 使能
        pRobotKimTransl:=,          //机器人模型，必须使用，如果无需模型请使指定无模型即可
        HandWhellPosX:=,            //手轮X位置
        HandWhellPosY:=,            //手轮Y位置
        HandWhellPosZ:=,            //手轮Z位置
        bBusy=>,                    //插补运行中信号，TRUE为运行中，FALSE为插补运行完毕
        bError=>,                   //运行错误信号，插补计算错误和轴异常都会导致功能块报错
        outCartesianPos=> );        //根据当前轴位置以及运动学模型换算后的空间坐标位置
```

5.8.2.2. HMC_RobotJog (插补点动功能块)

```
HMC_RobotJog(
        fVelocity:=,                //点动速度
        fAcceleration:=,            //点动加速度
        fDeceleration:=,            //点动减速度
        xJogForward:=,              //X轴正方向点动
        xJogBackward:=,             //X轴反方向点动
        yJogForward:=,              //轴正方向点动
        yJogBackward:=,             //轴反方向点动
        zJogForward:=,              //Z轴正方向点动
        zJogBackward:=,             //Z轴负方向点动
        pRobotKimTransl:=,          //机器人模型，必须使用，如果无需模型请指定无模型。
        bBusy=>,                    //插补运行中信号，TRUE为运行中，FALSE为插补运行完毕
        bError=>,                   //运行错误信号，插补计算错误和轴异常都会导致功能块报错
        outCartesianPos=> );        //根据当前轴位置以及运动学模型换算后得到的框架坐标(X,Y,Z)
```

5.8.2.3. HMC_RobotMove (运动控制功能块)

轨迹指令参数组，最大支持 100 条指令

```
HMC_RobotMove(
        bExecute:=,                 //上升沿触发信号
        bPause:=,                   //插补暂停信号，为TRUE后停止插补，恢复FALSE后继续运行
        bAbort:=,                   //插补终止信号，为TRUE后终止插补运行并重置输出
        fOverride:=,                //插补器当前运行的指令的速度比例，调整全局插补速度，必须填写参数
        bSmooth:=,                  //启用多段轨迹平滑功能，需要在功能块执行前选择
        fSmooth_R:=,                //多段轨迹平滑半径
        fSmooth_Override:=,         //平滑功能速度缩放比例，值应当小于1
        pRobotKimTransl:=,          //机器人模型，必须使用，如果无需模型请指定无模型
        NumberOfCommand:=,          //轨迹指令使用个数
        stMoveCommand:=,            //运动指令参数数组，最大支持100条指令
        bDone=>,                    //插补完成信号
        bPaused=>,                  //暂停完成信号
        bBusy=>,                    //插补运行中信号，TRUE为运行中，FALSE为插补运行完毕
        bError=>,                   //运行错误信号，插补计算错误和轴异常都会导致功能块报错
        eErrorID=>,                 //故障代码
        dCommandCount=>,            //输出运动指令个数，当功能块将运动指令全部执行完毕后置为0
        dRunCount=>,                //当前运行到了第几条指令
        outCartesianPos=> );        //根据当前轴位置以及运动学模型换算后得到的空间坐标位置
```

5.8.2.4. HMC_RobotMove_max1000 ()

轨迹指令参数组，最大支持 1000 条指令。

```
HMC_RobotMove_max1000(
        bExecute:= ,            //上升沿触发信号
        bPause:= ,              //插补暂停信号，为TRUE后停止插补，恢复FALSE后继续运行
        bAbort:= ,              //插补终止信号，为TRUE后终止插补运行并重置输出
        fOverride:= ,           //插补器当前运行的指令的速度比例，调整全局插补速度，必须填写参数
        bSmooth:= ,             //启用多段轨迹平滑功能，需要在功能块执行前选择
        fSmooth_R:= ,           //多段轨迹平滑半径
        fSmooth_Override:=,     //平滑功能速度缩放比例，值应当小于1
        pRobotKimTransl:=,      //机器人模型，必须使用，如果无需模型请指定无模型
        NumberOfCommand:=,      //轨迹指令使用个数
        stMoveCommand:= ,       //运动指令参数数组，最大支持1000条指令
        bDone=> ,               //插补完成信号
        bPaused=> ,             //暂停完成信号
        bBusy=> ,               //插补运行中信号，TRUE为运行中，FALSE为插补运行完毕
        bError=> ,              //运行错误信号，插补计算错误和轴异常都会导致功能块报错
        eErrorID=> ,            //故障代码
        dCommandCount=> ,       //输出运动指令个数，当功能块将运动指令全部执行完毕后置为0
        dRunCount=> ,           //当前运行到了第几条指令
        outCartesianPos=> );    //根据当前轴位置以及运动学模型换算后得到的空间坐标位置
```

5.8.3. 运动指令参数 stMoveParameter

5.8.3.1. 直线插补模式

直线插补就是控制机器的末端点由初始位置以直线轨迹运动到目标点，只需选择 bMoveType 模式为 FALSE

即可，需要使用的参数如下：

```
stCommand[1].fAcceleration :=;      //加速度
stCommand[1].fDeceleration :=;      //减速度
stCommand[1].fVelocity    :=;       //速度
stCommand[1].stTargetPos.X :=;      //末端点设定位置空间坐标X
stCommand[1].stTargetPos.Y :=;      //末端点设定位置空间坐标Y
stCommand[1].stTargetPos.Z :=;      //末端点设定位置空间坐标Z
stCommand[1].stTargetPos.A :=;      //末端点工具轴设定旋转角度A
```

5.8.3.2. 圆弧插补模式-半径模式

确定起始点到目标点位置的距离，设定半径，由此可得到一个确定的等腰三角形（腰即为半径），以等腰三角形的顶点作为圆心，选择劣弧(圆心角小于等于 180°对应的弧)作为运动轨迹，需要使用的参数如下：

```
stCommand[1].bMoveType    :=TRUE;     //插补模式选择（TRUE为圆弧插补，FALSE为直线插补）
stCommand[1].fAcceleration :=;        //加速度
stCommand[1].fDeceleration :=;        //减速度
stCommand[1].fVelocity    :=;         //速度
stCommand[1].stArcParameter.eArcMode:=1;        //圆弧模式，圆弧插补时生效（1为半径模式）
stCommand[1].stArcParameter.bDirection:=;       //圆弧方向（TRUE为逆时针，FALSE为顺时针）
stCommand[1].stArcParameter.fRadius:= ;         //设置半径，半径模式启用,默认在XY平面插补
stCommand[1].stTargetPos.X :=;        //末端点设定位置空间坐标X
stCommand[1].stTargetPos.Y :=;        //末端点设定位置空间坐标Y
stCommand[1].stTargetPos.Z :=;        //末端点设定位置空间坐标Z
stCommand[1].stTargetPos.A :=;        //末端点工具轴设定旋转角度A
```

5.8.3.3. 圆弧插补模式-圆心模式

确定起始点，目标点位置和圆心位置三点坐标，运动轨迹由圆心坐标定义，圆心的空间坐标需要在起始点和目标点之间的垂直平分线上，如果不是这样，那么圆心坐标将自动校正，偏差值应不大于 10%，同时可以选择圆弧运动方向 bDirection，自由选择顺时针或者逆时针旋转(劣弧：圆心角小于等于 180°对应的弧)，需要使用的参数如下：

```
stCommand[1].bMoveType    :=TRUE;     //插补模式选择（TRUE为圆弧插补，FALSE为直线插补）
stCommand[1].fAcceleration :=;        //加速度
stCommand[1].fDeceleration :=;        //减速度
stCommand[1].fVelocity    :=;         //速度
stCommand[1].stArcParameter.eArcMode:=0;        //圆弧模式，圆弧插补时生效（0为圆心模式）
stCommand[1].stArcParameter.bDirection:=;       //圆弧方向（TRUE为劣弧，FALSE为优弧）
stCommand[1].stArcParameter.stMidPoint.X:=;     //圆心空间坐标X，圆心模式启用
stCommand[1].stArcParameter.stMidPoint.Y:=;     //圆心空间坐标Y，圆心模式启用
stCommand[1].stArcParameter.stMidPoint.Z:=;     //圆心空间坐标Z，圆心模式启用
stCommand[1].stTargetPos.X :=;                  //末端点设定位置空间坐标X
stCommand[1].stTargetPos.Y :=;                  //末端点设定位置空间坐标Y
stCommand[1].stTargetPos.Z :=;                  //末端点设定位置空间坐标Z
stCommand[1].stTargetPos.A :=;                  //末端点工具轴设定旋转角度A
```

5.8.3.4. 圆弧插补模式-过渡点模式

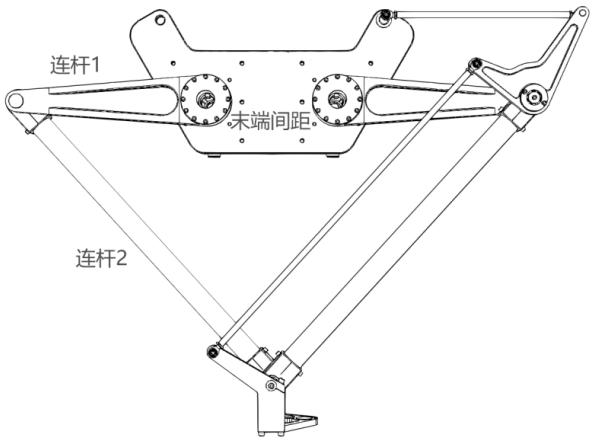
确定起始点，目标点位置和过渡点位置三点坐标，需要注意的是，三点位置不能一线，因为这样会导致无法在三维空间中确定唯一的插补平面。由三点确定的三角形的外心作为圆心（三边垂直平分线的交点）确定三角形的外接圆，可以选择圆弧运动方向 bDirection，自由选择顺时针或者逆时针旋转(劣弧：圆心角小于等于 180°对应的弧)，需要使用的参数如下：

```
stCommand[1].bMoveType    :=TRUE;     //插补模式选择（TRUE为圆弧插补，FALSE为直线插补）
stCommand[1].fAcceleration :=;        //加速度
stCommand[1].fDeceleration :=;        //减速度
stCommand[1].fVelocity    :=;         //速度
stCommand[1].stArcParameter.eArcMode:=2;                    //圆弧模式，圆弧插补时生效（2为过渡点模式）
stCommand[1].stArcParameter.bDirection:=;                   //圆弧方向（TRUE为劣弧，FALSE为优弧）
stCommand[1].stArcParameter.stAcorssPoint.X:=;              //过渡点空间坐标X，过渡点模式启用
stCommand[1].stArcParameter.stAcorssPoint.Y:=;              //过渡点空间坐标Y，过渡点模式启用
stCommand[1].stArcParameter.stAcorssPoint.Z:=;              //过渡点空间坐标Z，过渡点模式启用
stCommand[1].stTargetPos.X :=;        //末端点设定位置空间坐标X
stCommand[1].stTargetPos.Y :=;        //末端点设定位置空间坐标Y
stCommand[1].stTargetPos.Z :=;        //末端点设定位置空间坐标Z
stCommand[1].stTargetPos.A :=;        //末端点工具轴设定旋转角度A
```

5.8.4. 使用流程举例

5.8.4.1. 2 轴 Delta 模型功能块举例

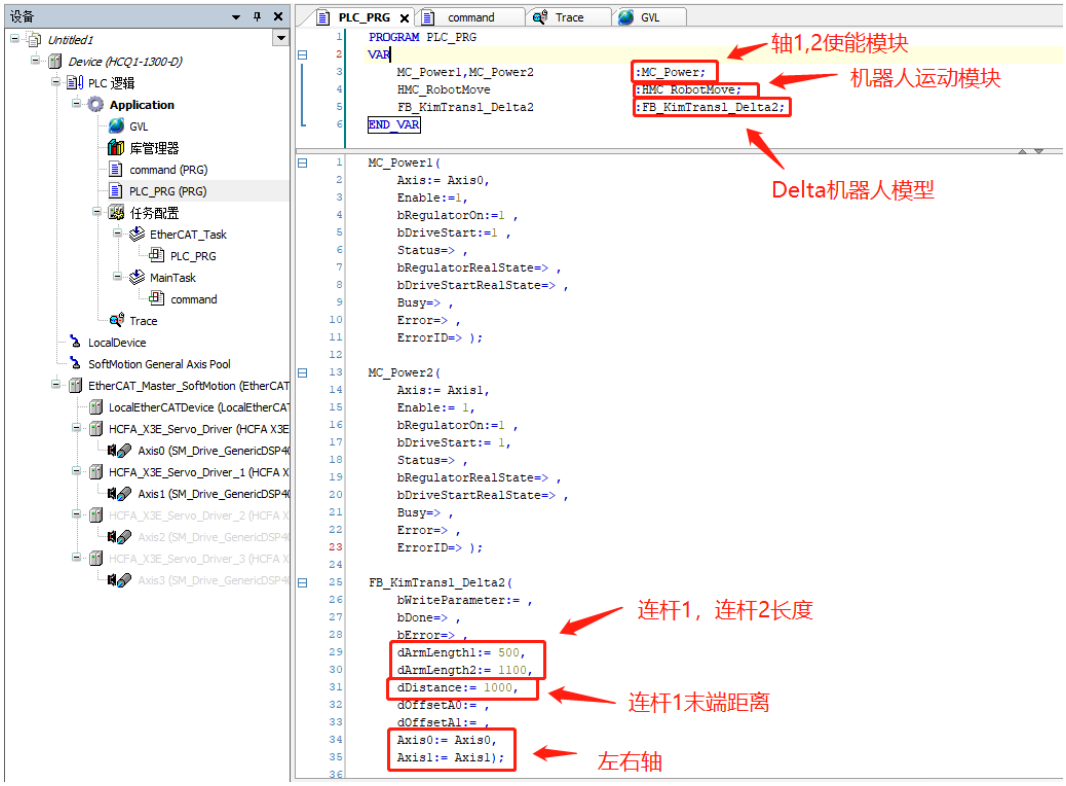
例：现有一个 Delta 机器人（如下图），连杆 1 长度为 500，连杆 2 长度为 1100，两个连杆 1 的起始端为圆形相切，因此末端距离取 2 倍连杆 1 的长度即可，水平位置为原点，两个轴的正方向都设定为向内，即左轴顺时针为正，右轴逆时针为正。此时，功能块自动转换得到的机器人末端点位置为（0,0），需要将末端向上抬升 20，需要使用直线插补运动控制指令，设定坐标（0,20）：



Delta 机器人模型功能块和运动功能块的声明和调用

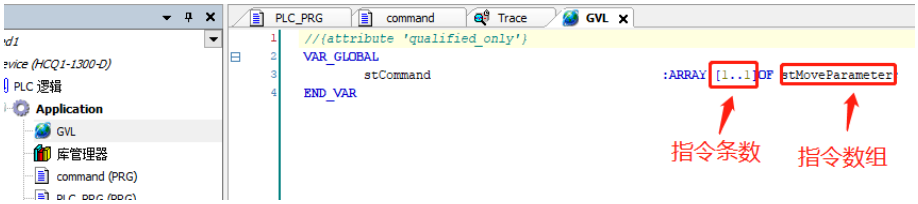
在 PLC_RPG 中声明和调用 HMC_RobotMove 和 FB_KimTransl_Delta2 功能块，同时需要给 Axis0 和 AXis1 使能，此处为了举例方便直接将使能数值写为 1，即自动使能。

FB_KimTransl_Delta 的参数部分：连杆 1 长度为 500，连杆 2 长度为 1100，dDistance 为连杆连杆 1 末端之间的说距离，由于连杆初始端为圆形相切，因此此处 dDistance 为连杆 1 长度*2 即可，即 500*2=1000；

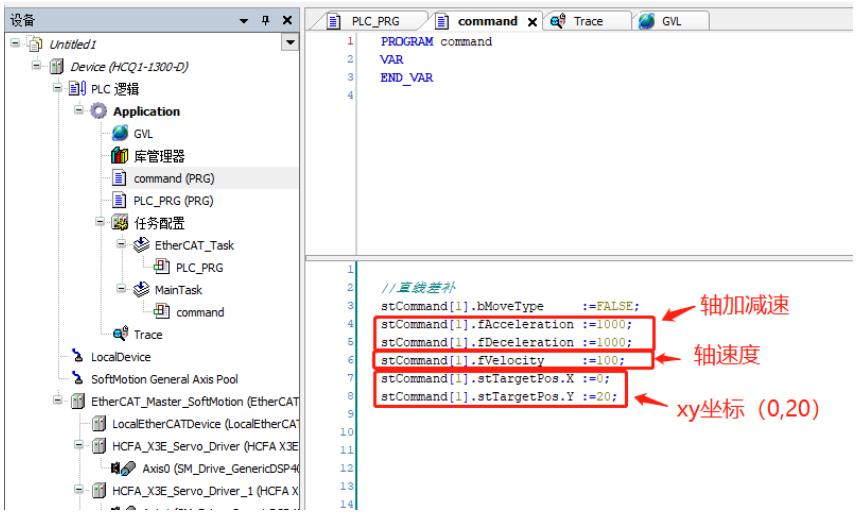



```
HMC_RobotMove(
    bExecute:= ,
    bPause:= ,
    bAbort:= ,
    fOverride:= 1,  速度比例，必须填写
    bSmooth:= ,
    fSmooth_R:= ,
    SmoothPathMode:= ,
    fSmooth_Override:= ,
    pRobotKimTransl:= ADR(FB_KimTransl_Delta2),  机器人模型地址
    NumberOfCommand:= NumberCommand,  使用指令个数
    stMoveCommand:= stCommand,  运动指令数组名
    bDone=> ,
    bPaused=> ,
    bBusy=> ,
    bError=> ,
    eErrorID=> ,
    dCommandCount=> ,
    dRunCount=> ,
    outCartesianPos=> );
```

在 GVL 中声明运动指令数组

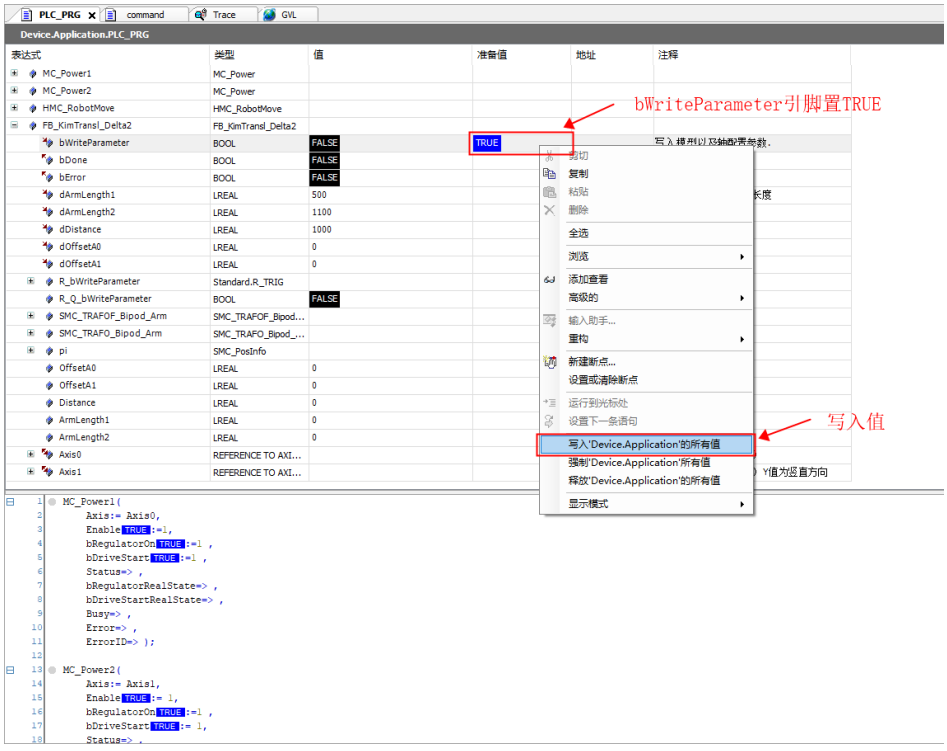


并在 POUcommand 中填写参数，因为需要将末端向上抬升 20，使用直线插补运动控制指令，设定坐标（0,20）：



实现运动控制

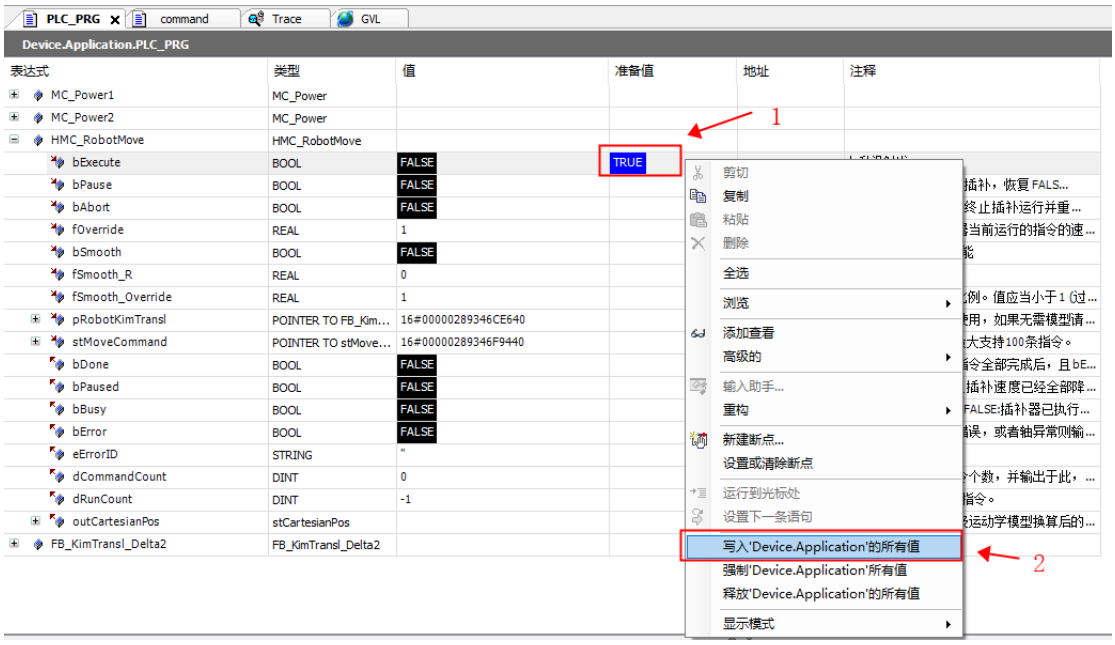
触发 FB_KimTransl_Delta2 功能块的 bWriteParameter 引脚，完成模型以及轴配置参数写入。



在参数正确，且轴未出错的情况下，模型以及轴参数配置完成，bDone 引脚，模型参数配置完成信号输出TRUE。

表达式	类型	值	准备值	地址	注释
MC_Power1	MC_Power				
MC_Power2	MC_Power				
HMC_RobotMove	HMC_RobotMove				
FB_KimTransl_Delta2	FB_KimTransl_Delta2				
bWriteParameter	BOOL	TRUE			写入模型以及轴配置参数.
bDone	BOOL	TRUE			模型参数配置完成
bError	BOOL	FALSE			模型写入参数有误
dArmLength1	LREAL	500			与电机连接的连杆 1的长度
dArmLength2	LREAL	1100			连杆 2的长度
dDistance	LREAL	1000			两个电机的距离
dOffsetA0	LREAL	0			A0轴的额外偏移量
dOffsetA1	LREAL	0			A1轴的额外偏移量
R_bWriteParameter	Standard.R_TRIG				
R_Q_bWriteParameter	BOOL	TRUE			
SMC_TRAFOF_Bipod_Arm	SMC_TRAFOF_Bipod...				
SMC_TRAFO_Bipod_Arm	SMC_TRAFO_Bipod_...				
pi	SMC_PosInfo				
OffsetA0	LREAL	0			
OffsetA1	LREAL	0			
Distance	LREAL	1000			
ArmLength1	LREAL	500			
ArmLength2	LREAL	1100			
Axis0	REFERENCE TO AXI...				机械左轴位置（负X值）
Axis1	REFERENCE TO AXI...				机械右轴位置（正X值）Y值为竖直方向

触发 HMC_RobotMove 功能块的 bExecute 引脚，开启机器人运动。



表达式	类型	值	准备值	地址	注释
MC_Power1	MC_Power				
MC_Power2	MC_Power				
HMC_RobotMove	HMC_RobotMove				
bExecute	BOOL	FALSE	TRUE		
bPause	BOOL	FALSE			
bAbort	BOOL	FALSE			
fOverride	REAL	1			
bSmooth	BOOL	FALSE			
fSmooth_R	REAL	0			
fSmooth_Override	REAL	1			
pRobotKimTransl	POINTER TO FB_Kim...	16#00000289346CE640			
stMoveCommand	POINTER TO stMove...	16#00000289346F9440			
bDone	BOOL	FALSE			
bPaused	BOOL	FALSE			
bBusy	BOOL	FALSE			
bError	BOOL	FALSE			
eErrorID	STRING	''			
dCommandCount	DINT	0			
dRunCount	DINT	-1			
outCartesianPos	stCartesianPos				
FB_KimTransl_Delta2	FB_KimTransl_Delta2				

程序在运行完全部的数组指令的后（此处为 1 条指令），bDone 引脚输出 TRUE 信号，即完成了运动控制，即机器人末端点已经到达指定位置。

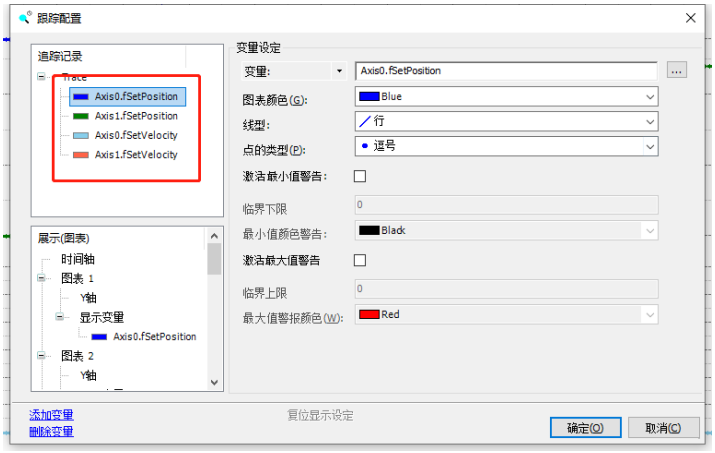
PLC_PRG | command | Trace | GVL

Device.Application.PLC_PRG

表达式	类型	值	准备值	地址	注释
MC_Power1	MC_Power				
MC_Power2	MC_Power				
HMC_RobotMove	HMC_RobotMove				
bExecute	BOOL	TRUE			上升沿触发
bPause	BOOL	FALSE			暂停插补，为TRUE..插补，恢复FALS...
bAbort	BOOL	FALSE			终止插补，为TRUE..终止插补运行并重...
fOverride	REAL	1			速度比例，为插补器当前运行的指令的速...
bSmooth	BOOL	FALSE			启用多段轨迹平滑功能
fSmooth_R	REAL	0			多段轨迹平滑半径
fSmooth_Override	REAL	1			平滑过度速度缩小比例。值应当小于1 (过...
pRobotKimTransl	POINTER TO FB_Kim...	16#00000289346CE640			机器人模型，必须使用，如果无需模型请...
stMoveCommand	POINTER TO stMove...	16#00000289346F9440			轨迹指令参数组，最大支持100条指令。
bDone	BOOL	TRUE			当插补池内所有的指令全部完成后，且bE...
bPaused	BOOL	FALSE			当bPause为TRUE，且插补速度已经全部降...
bBusy	BOOL	FALSE			TRUE:插补器运行中 FALSE:插补器已执行...
bError	BOOL	FALSE			若插补中发现计算错误，或者轴异常则输...
eErrorID	STRING	''			故障代码
dCommandCount	DINT	0			触发执行后检查指令个数，并输出于此，...
dRunCount	DINT	1			当前运行到了第几个指令。
outCartesianPos	stCartesianPos				根据当前轴位置以及运动学模型换算后的...
FB_KimTransl_Delta2	FB_KimTransl_Delta2				

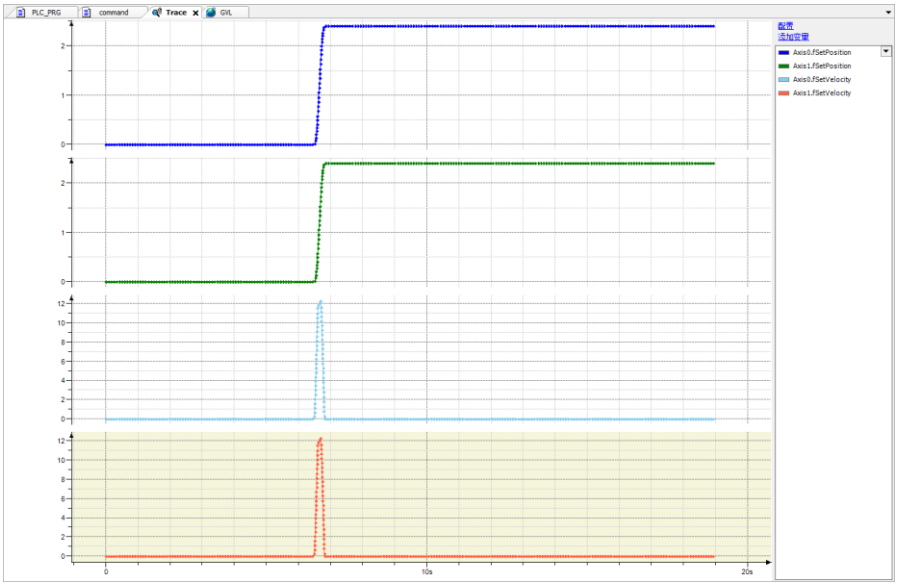
bDone引脚输出为TRUE

为了方便查看具体的轴运动过程，在 trace 中添加追踪 Axis0 和 Axis1 两轴的设定位置和设定速度。



观察重新操作一遍，观察 trace 中轴的设定位置和速度。由 trace 图可知，左右两轴都正向旋转 2.4 度，即

实现 Delta 机器人末端抬升，换算后可得抬升距离为 20。



5.8.4.2. 4 轴 Scara 模型举例

例：现有 4 轴 Scara 机器人如下图，假设此时，机械臂为右手姿态，大小臂垂直状态，辅助轴未上下移动，即末端点空间坐标 xyz（500,500,0）工具轴旋转角度为 0，大臂长度 500，小臂长度 500，需要操控末端点：

（1）以直线插补模式从（500，500，0）运动到（800，0，0）。

（2）辅助轴下降到-50，即空间坐标（800，0，-50）。

（3）收回辅助轴，即空间坐标（800，0，0）。

（4）以半径模式，半径为 800 移动到（0,800，0）。

（5）放下辅助轴到-50，即空间坐标（0,800，0），同时工具轴正向旋转 90 度。

（6）收回辅助轴，即空间坐标（0，800，0），同时工具轴逆向旋转 90 度。

（7）圆心模式下以（0，0，0）为圆心，逆时针移动到（800，0，0）。

Scara 机器人模型功能块和运动功能块的声明和调用

在 PLC_RPG 中声明和调用 HMC_RobotMove 和 FB_KimTransl_Scara2_Z_Tool 功能块，同时需要给 Axis0-AXis3 共 4 个轴使能，此处为了举例方便直接将使能数值写为 1，即自动使能。

FB_KimTransl_Scara2_Z_Tool 的参数部分：大臂长度为 500，小臂长度为 500，手臂姿态选择置 TRUE，即右手姿态。

```
FB_KimTransl_Scara2_Z_Tool(
    bWriteParameter:= ,
    bDone=> ,
    bError=> ,
    dArmLength1:= 500,
    dArmLength2:= 500,
    dOffsetA0:= ,
    dOffsetA1:= ,
    bElbowLow:= TRUE,
    Axis0:= Axis0,
    Axis1:= Axis1,
    Axis2:= Axis2,
    Axis3:= Axis3);

HMC_RobotMove(
    bExecute:= ,
    bPause:= ,
    bAbort:= ,
    fOverride:= 1,
    bSmooth:= ,
    fSmooth_R:= ,
    SmoothPathMode:= ,
    fSmooth_Override:= ,
    pRobotKimTransl:= ADR(FB_KimTransl_Scara2_Z_Tool),
    NumberOfCommand:= NumberCommand,
    stMoveCommand:= stCommand,
    bDone=> ,
    bPaused=> ,
    bBusy=> ,
    bError=> ,
    eErrorID=> ,
    dCommandCount=> ,
    dRunCount=> ,
    outCartesianPos=> );
```

大小臂长度

手臂姿态选择

速度比例，必须填写

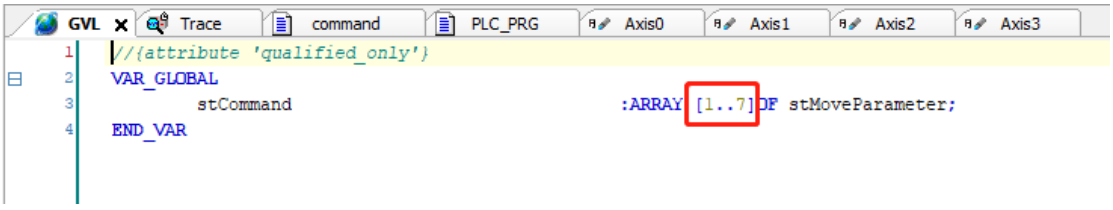
机器人模型地址

指令使用个数

运动指令数组名

在 GVL 中声明运动指令数组

根据示例要求，在 GVl 中声明 stMoveParameter 数组，使用 7 条运动控制指令。



并在 POUcommand 中填写参数，按照示例要求，共 7 条运动控制指令：

```
//1.直线差补移动到（800,0,0）
stCommand[1].bMoveType     :=FALSE;
stCommand[1].fAcceleration :=300;
stCommand[1].fDeceleration :=300;
stCommand[1].fVelocity     :=100;
stCommand[1].stTargetPos.X :=800;
stCommand[1].stTargetPos.Y :=0;
stCommand[1].stTargetPos.Z :=0;
stCommand[1].stTargetPos.A :=0;

//2.将辅助轴下降50
stCommand[2].bMoveType     :=FALSE;
stCommand[2].fAcceleration :=300;
stCommand[2].fDeceleration :=300;
stCommand[2].fVelocity     :=100;
stCommand[2].stTargetPos.X :=800;
stCommand[2].stTargetPos.Y :=0;
stCommand[2].stTargetPos.Z :=-50;
stCommand[2].stTargetPos.A :=0;

//3.收回辅助轴
stCommand[3].bMoveType     :=FALSE;
stCommand[3].fAcceleration :=300;
stCommand[3].fDeceleration :=300;
stCommand[3].fVelocity     :=100;
stCommand[3].stTargetPos.X :=800;
stCommand[3].stTargetPos.Y :=0;
stCommand[3].stTargetPos.Z :=0;
stCommand[3].stTargetPos.A :=0;

//4.半径模式下，以800位半径，运动到（0,800,0）
stCommand[4].bMoveType     :=TRUE;
stCommand[4].fAcceleration :=300;
stCommand[4].fDeceleration :=300;
stCommand[4].fVelocity     :=100;
stCommand[4].stArcParameter.eArcMode:=1;
stCommand[4].stArcParameter.fRadius:=800;
stCommand[4].stTargetPos.X :=0;
stCommand[4].stTargetPos.Y :=800;
stCommand[4].stTargetPos.Z :=0;
stCommand[4].stTargetPos.A :=0;

//5.将辅助轴下降50,同时工具轴正向旋转90度
stCommand[5].bMoveType     :=FALSE;
stCommand[5].fAcceleration :=300;
stCommand[5].fDeceleration :=300;
stCommand[5].fVelocity     :=100;
stCommand[5].stTargetPos.X :=0;
stCommand[5].stTargetPos.Y :=800;
stCommand[5].stTargetPos.Z :=-50;
stCommand[5].stTargetPos.A :=90;

//6.收回辅助轴,同时工具轴逆向旋转90度
stCommand[6].bMoveType     :=FALSE;
stCommand[6].fAcceleration :=300;
stCommand[6].fDeceleration :=300;
stCommand[6].fVelocity     :=100;
stCommand[6].stTargetPos.X :=0;
stCommand[6].stTargetPos.Y :=800;
stCommand[6].stTargetPos.Z :=0;
stCommand[6].stTargetPos.A :=-90;

//7.圆心模式下，以（0,0,0）为圆心，顺时针运动到（800,0,0）
stCommand[7].bMoveType     :=TRUE;
stCommand[7].fAcceleration :=300;
stCommand[7].fDeceleration :=300;
stCommand[7].fVelocity     :=100;
stCommand[7].stArcParameter.eArcMode:=0;
stCommand[7].stArcParameter.bDirection:=FALSE;
stCommand[7].stArcParameter.stMidPoint.X:=0;
stCommand[7].stArcParameter.stMidPoint.Y:=0;
stCommand[7].stArcParameter.stMidPoint.Z:=0;
stCommand[7].stTargetPos.X :=800;
stCommand[7].stTargetPos.Y :=0;
stCommand[7].stTargetPos.Z :=0;
stCommand[7].stTargetPos.A :=0;
```

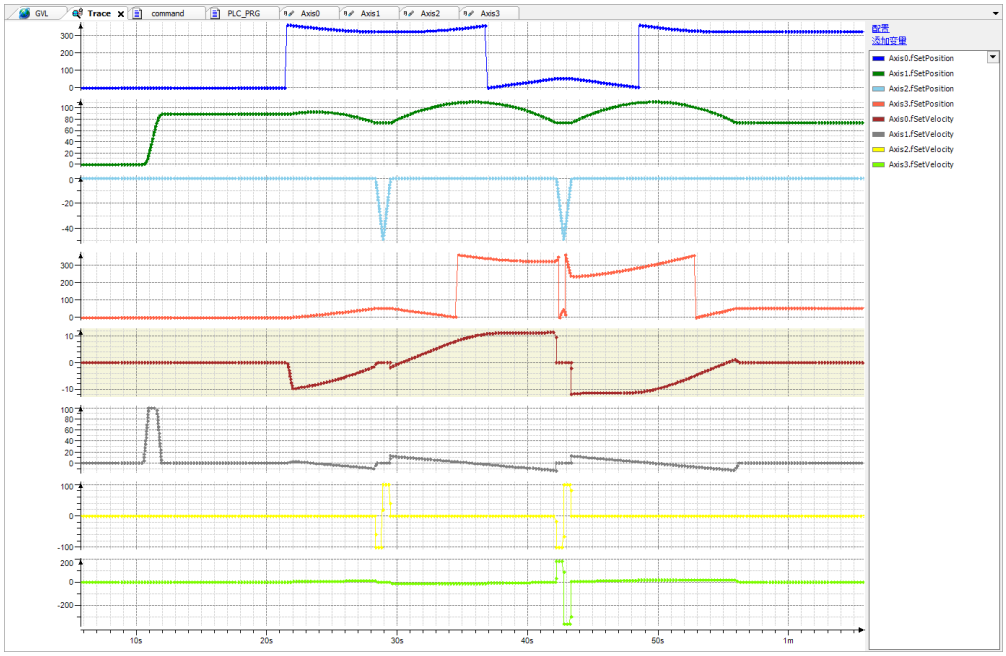
实现运动控制

示例仿真程序中，为了调整机器人大小臂为垂直状态，即将末端点定位到（500，500,0）的空间位置，利用 MC_MoveAdditive 功能块将 Axis1 正向旋转 90 度。

PS：需要注意的是在实际运用中，使用不同的模型功能块，功能块会自动读取当前轴位置，在各自的模型下自动换算来获取末端点的空间坐标（x,y,z），而在仿真中，由于功能块内部默认所有的轴都处于初始位置 0，因此需要先对单独的轴进行一些定位操作，才能模仿实际上的使用情景，在仿真中，请先完成对各个轴的单独定位，再进行机器人功能块的模型参数写入。

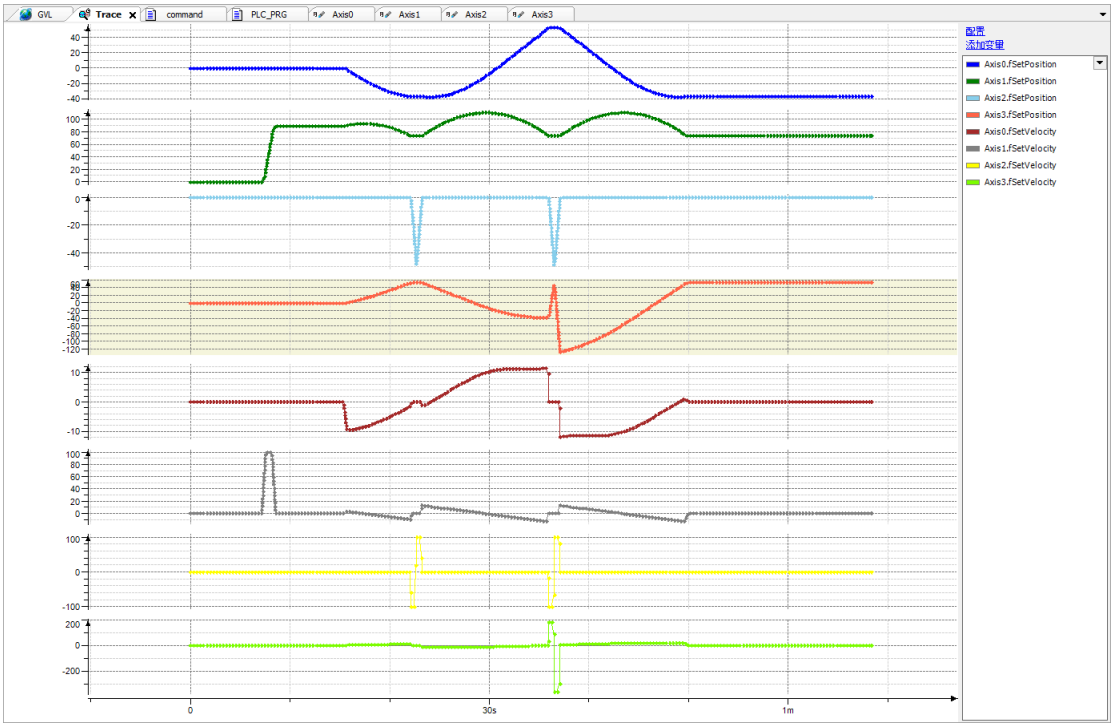
```
MC_MoveAdditive(
    Axis:= Axis1,
    Execute:= ,
    Distance:= 90,
    Velocity:= 100,
    Acceleration:= 300,
    Deceleration:= 300,
    Jerk:= ,
    Done=> ,
    Busy=> ,
    CommandAborted=> ,
    Error=> ,
    ErrorID=> );
```

依次触发 MC_MoveAdditive 的 Execute 引脚，FB_KimTransl_Scara2_Z_Tool 的 bWriteParameter 引脚和 HMC_RobotMove 的 bExecute 引脚，启动机器人按照数组中的 7 条运动指令开始运动，在 trace 中追踪 Axis0-Axis3 四个轴的位置和速度，具体表现如图：



需要注意的是，trace 图中轴 Axis0 和轴 Axis3 存在的“阶跃”并不是飞车。这个“阶跃”是由于轴 Axis0 和轴 Axis3 存在 0 度逆向旋转的运动方式产生的。(例如在模态轴状态下，原本 0 度，逆向旋转到-10 度，由于在 codesys 中模态轴没有负值，因此会显示成 350 度)

为了感官上更为合理，这里将 Axis0 和 Axis3 改为线性轴再观察一次 trace 图，可以看到 4 轴运动都较为平滑：



点动功能块 HMC_RobotJog 使用举例

此处点动功能块使用举例采用的是 4 轴 Scara 机器人模型，点动功能块 HMC_RobotJog 的作用是操控模型的多轴运动实现控制特定模型的末端点在空间 X，Y，Z 轴上的单轴正负方向的直线运动。

此处举例在轴末端点处于（500,500,0）的位置时，控制末端点进行末端点在 X 轴上向正方向运动到极限位置(即大小臂伸直）。理论上最终末端点的空间位置为（（√3）*500，500，0），即轴 Axis0 正向旋转 30 度，轴 Axis1 旋转 0 度，即大小臂完全伸直。

在 PLC_RPG 中声明和调用 HMC_RobotJog 功能块：

```
PROGRAM PLC_PRG
VAR
    MC_Power1,MC_Power2,MC_Power3,MC_Power4          :MC_Power;
    HMC_RobotMove                                    :HMC_RobotMove;
    MC_MoveAdditive                                  :MC_MoveAdditive;
    HMC_RobotJog                                     :HMC_RobotJog;
    FB_KimTransl_Scara2_Z_Tool                       :FB_KimTransl_Scara2_Z_Tool;

    HMC_RobotJog(
    fVelocity:= 100,
    fAcceleration:= 1000,
    fDeceleration:= 1000,
    xJogForward:= ,
    xJogBackward:= ,
    yJogForward:= ,
    yJogBackward:= ,
    zJogForward:= ,
    zJogBackward:= ,
    pRobotKimTransl:= ADR(FB_KimTransl_Scara2_Z_Tool),
    bBusy=> ,
    bError=> ,
    outCartesianPos=> );
```

速度设置

机器人模型

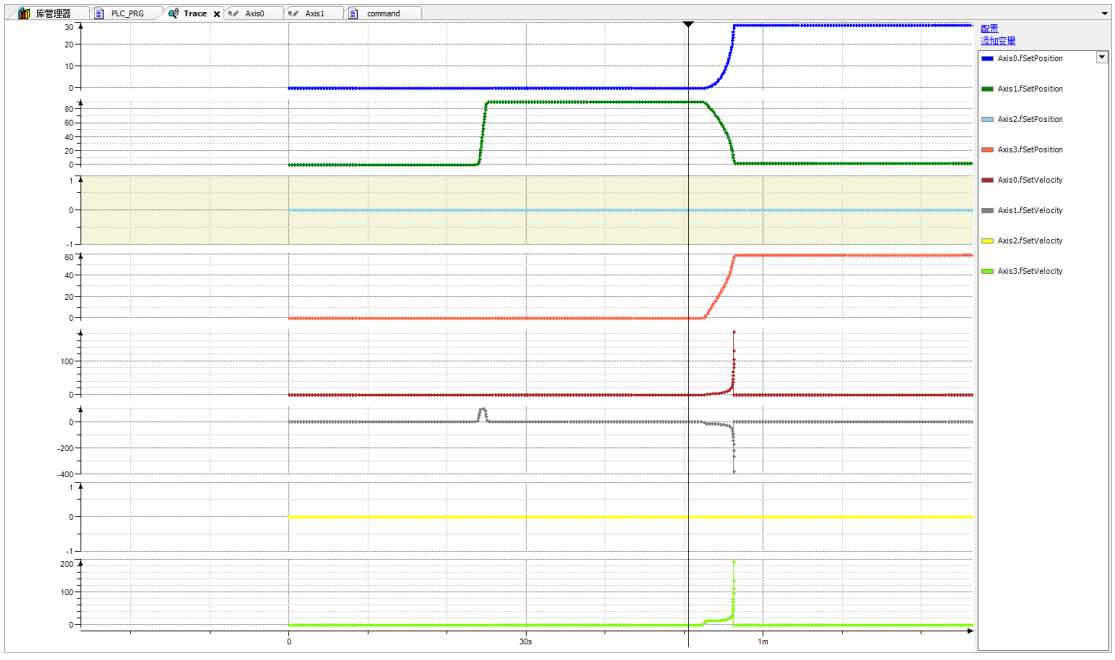
与上文使用指令控制机器人模型相同，需要先使用 MC_MoveAdditive 给你看将 Axis 正向选择 90 度，即将末端的的位置定位为（500，500，0），再利用功能块写入模型参数，具体操作步骤如下：

（1）触发 MC_MoveAdditive 功能块的 Execute 引脚，让轴 Axis 正向选择 90 度。

（2）触发 FB_KimTransl_Scara2_Z_Tool 功能块的 bWriteParameter 引脚，完成模型参数的写入。

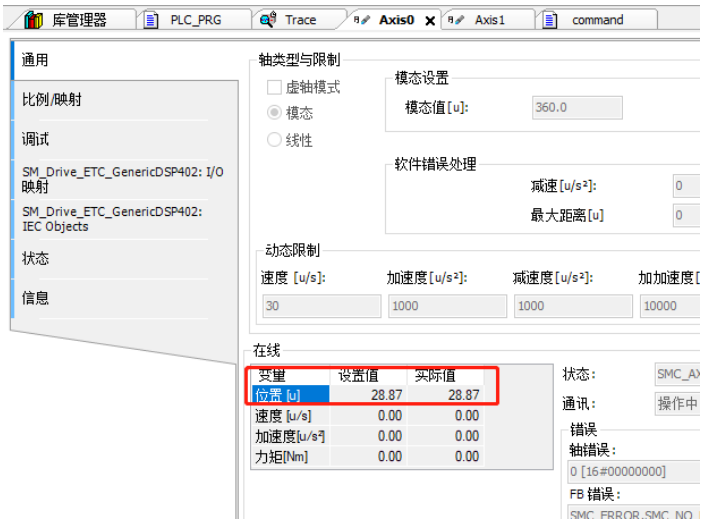
（3）触发 HMC_RobotJog 的 xJogForWard 引脚，控制末端点就行 X 轴正方形的直线电动运动。

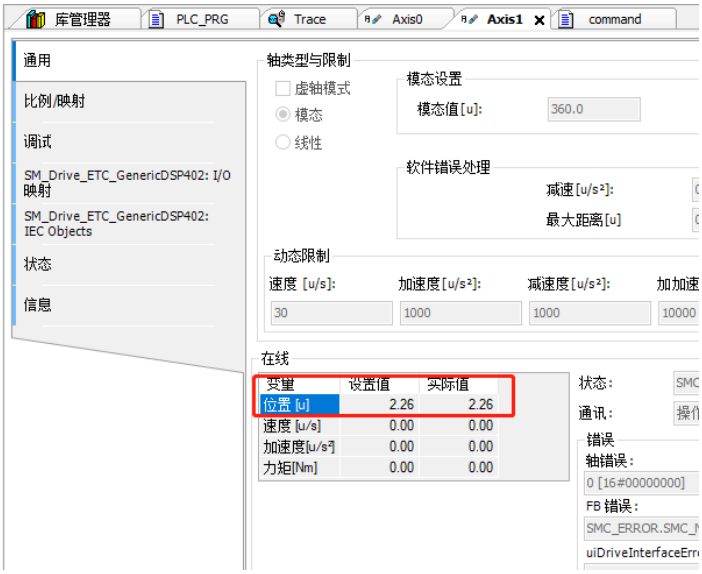
Trace 图如下所示：



可以看到，轴 Axis0 最终旋转角度为 28.87 度，轴 Axis1 的最终旋转角度为 2.26 度，实际使用也是如此，

无法到达理论状态是正常的



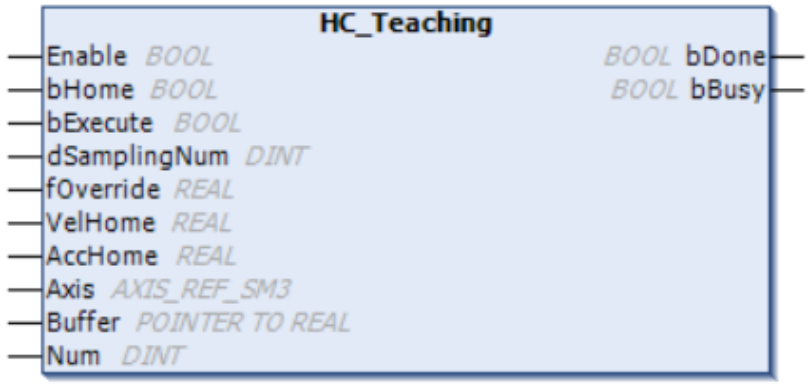


5.9. Teaching

5.9.1. HC_teaching (FB)

开启示教功能后记录轴的位置并复现出来，类似于机器人拖动示教。

变量

名称	HC_teaching（示教功能块）		
支持的模式			
图形表现		ST 表现	
		HC_Teaching(Enable:= , bHome:= , bExecute:= , dSamplingNum:= , fOverride:= , VelHome:= , AccHome:= , bDone=> , bBusy=> , Axis:= , Buffer:= , Num:=);	

输入输出变量

输出变量	名称	数据类型	有效范围	内容
Axis	轴	AXIS_REF_SM3		
Buffer	数据缓存区	ARRAY [*] OF REAL		用于记录示教位置信息，添加 PERSISTENT 变量
Num	数组元素个数	DINT		用于记录数组元素个数，添加 PERSISTENT 变量

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
Enable	开始示教	BOOL			为 True 开始记录轴的位置
bHome	回原	BOOL			回到记录初始位置
bExecute	使能	BOOL			复现示教过程
dSamplingNum	采样周期个数	DINT		1	采样周期个数
fOverride	示教速度倍速	Real		1	示教速度倍速
VelHome	回原速度	Real			回原速度
AccHome	回原加速度	Real			回原加速度

输出变量

输出变量	名称	数据类型	有效范围	内容
bDone	完成	BOOL		
bBusy	执行中	BOOL		

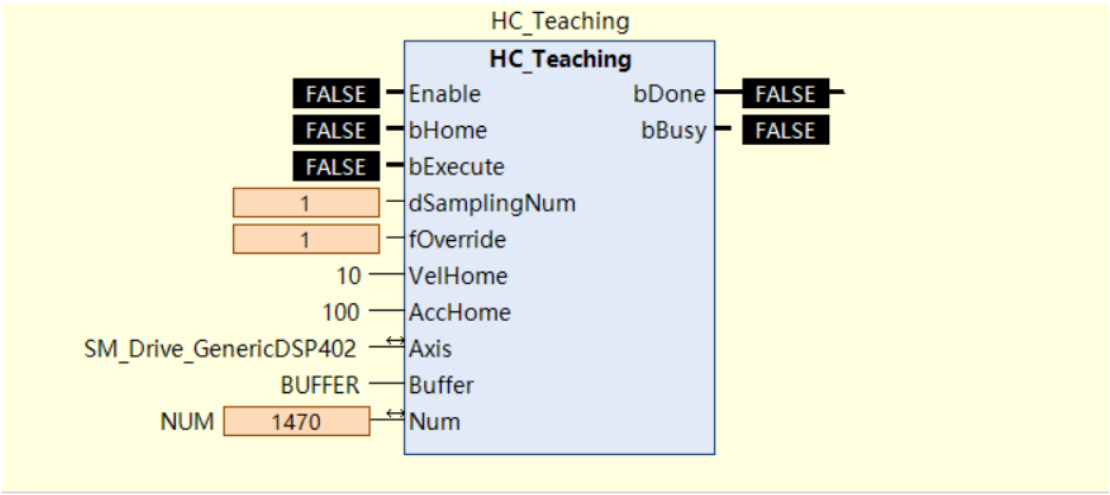
使用举例

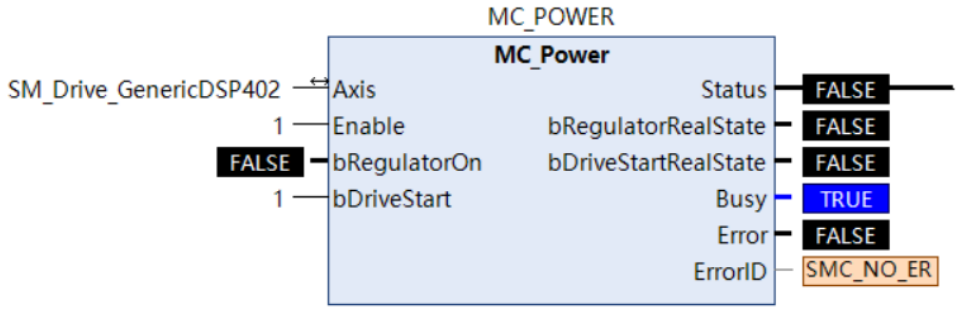
【1】声明如下变量，（BUFFER、NUM 不需要掉电保持也可以声明成普通变量）

```
PROGRAM PLC_PRG
VAR
    HC_Teaching:HC_Teaching;
    MC_POWER:MC_Power;;
END_VAR
VAR RETAIN PERSISTENT
    BUFFER   :ARRAY [0..6000] OF REAL;
    NUM      :DINT;
END_VAR
```

【2】下载并运行程序

伺服 power_off 状态下触发 Enable，开始示教。给定缓存区存满或 Enable 置 False，示教结束，将位置数据保持在缓存区，记入数据个数。





Device.Application.PersistentVars

表达式	类型	值	准备值
BUFFER[41]	REAL	0.005493164	
BUFFER[42]	REAL	0.005493164	
BUFFER[43]	REAL	0.002746582	
BUFFER[44]	REAL	0.002746582	
BUFFER[45]	REAL	0.005493164	
BUFFER[46]	REAL	0.002746582	
BUFFER[47]	REAL	0.005493164	
BUFFER[48]	REAL	0.002746582	
BUFFER[49]	REAL	0.005493164	
BUFFER[50]	REAL	0.005493164	
BUFFER[51]	REAL	0.002746582	
BUFFER[52]	REAL	0.005493164	
BUFFER[53]	REAL	0.005493164	
BUFFER[54]	REAL	0.002746582	
BUFFER[55]	REAL	0.005493164	
BUFFER[56]	REAL	0.005493164	
BUFFER[57]	REAL	0.002746582	
BUFFER[58]	REAL	0.005493164	
BUFFER[59]	REAL	0.002746582	
BUFFER[60]	REAL	0.002746582	
BUFFER[61]	REAL	0.002746582	
BUFFER[62]	REAL	0.005493164	
BUFFER[63]	REAL	0.005493164	
BUFFER[64]	REAL	0.005493164	
BUFFER[65]	REAL	0.005493164	
BUFFER[66]	REAL	0.005493164	
BUFFER[67]	REAL	0.005493164	
BUFFER[68]	REAL	0.002746582	
BUFFER[69]	REAL	0.005493164	

【3】执行回原

伺服使能，触发 bHome，轴回到初始位置。

【4】复现示教过程

要点说明

- 若轴为模态轴，dSamplingNum 数值过大，复现示教容易造成电机旋转方向不对。例如：采样周期 100，轴从 240 正向跑到 80，此时判断正向要走 200，反向只要 160，实际伺服会反向跑到 80。

5.10. TransformCam

5.10.1. HMC_TransformCam (FB)

用于将不同曲线转换为 Cam 表。

名称	HMC_TransformCam

图形表现

ST 表现

```
HMC_TransformCam(
    Execute:= ,
    MC_CAM_REF:= ,
    nElements:= ,
    Busy=> ,
    Done=> ,
    Error=> ,
    ERROR_ID=> ,
    arrCamType:= ,
    MC_CAM_REF_OUT:= );
```

输入输出变量

输出变量	名称	数据类型	有效范围	内容
MC_CAM_REF_OUT	转换表	MC_CAM_REF		

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bExecute	使能	BOOL			
MC_CAM_REF	Cam	MC_CAM_REF			
nElements	元素个数	INT		3601	
arrCamType	曲线类型数组	ARRAY [*] OF Camtype			

输出变量

输出变量	名称	数据类型	有效范围	内容
Done	完成	BOOL		
Busy	执行中	BOOL		
Error	错误	BOOL		
ErrorID	错误 ID	WSTRING		

要点说明

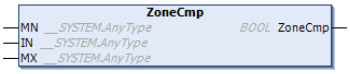
- CamType(ENUM)：Polynomial5=0,5 次曲线；Line=1,直线；Sintype=2,三角函数;Polynomial3=3,三次曲线;parabola=4,抛物线;Exponential=5,指数曲线。
- MC_CAM_REF 为原 Cam 表，使用其 XYVA 参数，将其转换为 nElements 条，类型为 arrCamType 数组内类型的曲线，输出到 MC_CAM_REF_OUT。

6. OmronUtils (欧姆龙指令功能)

6.1. 比较指令

6.1.1. ZoneCmp (区域比较)

判定比较数据是否在指定的上限值与下限值之间。

指令	名称	FB/FUN	图形表现	ST 表现
ZoneCmp	区域比较	FUN		Out:=ZoneCmp(MN, In, MX);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
MN	下限值	输入	下限值	遵从数据类型	—	0
In	比较数据		要比较的数值			(*)
MX	上限值		上限值			0
Out	比较结果	输出	比较结果	遵从数据类型	—	—

*省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
MN						○	○	○	○	○	○	○	○	○	○	○	○	○	○	
																(*)				
In						○	○	○	○	○	○	○	○	○	○	○	○	○	○	
																(*)				
MX						○	○	○	○	○	○	○	○	○	○	○	○	○	○	
																(*)				
Out	○																			

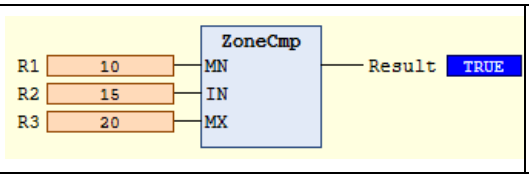
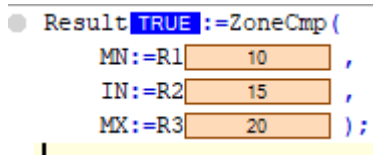
功能

判定比较数据 “In” 是否在上限值 “MX” 和下限值 “MN” 之间。

“MX” ≥ “In” ≥ “MN” 时，“Out” 的值为 TRUE，否则为 FALSE。

数据类型	大小关系
TIME	值较大者判断为大。
DATE、TOD、DT	对于日期和时刻，较后者判断为大。

“MN”=INT#10、“In”=INT#20、“MX”=INT#30 时的示例如下所示。变量 Result 的值为 TRUE。

	FBD	ST
定义变量	VAR R1:REAL; R2:REAL; R3:REAL; Result:BOOL; END_VAR	
程序		
运行结果		

参考

比较 TIME 型、DT 型、TOD 型的大小时，请符合要比较的值的精度。有“TruncTime 指令”、“TruncDt 指令”、“TruncTod 指令”，以符合值的精度。

要点说明

- “In”、“MX”、“MN” 的数据类型不同时，将类型扩展为包括所有数据类型的有效范围在内的数据类 型后，再进行比较。
- “In”、“MX”、“MN” 为实数时，如果含有除不尽的除法结果等，由于误差的原因，处理结果可能会 出现意外。
- 带符号整数型（SINT，INT，DINT，LINT）和无符号整数型（USINT，UINT，UDINT，ULINT）无法比较。
- TIME 型、DATE 型、TOD 型、DT 型仅可在相同的数据类型之间进行比较。如果指定了不同的数据类型， 则编连时会出现异常。

- + ∞之间或 - ∞之间判断为相等。
- “In” 的值为非数时，“Out” 的值为 FALSE。
- 通过梯形图程序使用本指令时，如果本指令的电路前段发生异常，则 “Out” 的值为 FALSE。
- 以下情况时会发生异常，“Out” 的值为 FELSE。
- “MN” 的值大于 “MX” 的值时。
- “MX”、“MN” 中任意一个为非数时

6.1.2. TableCmp (表格比较)

将比较数据与比较表指定的多个定义区间进行比较。

指令	名称	FB/FUN	图形表现	ST 表现
TableCmp	表格比较	FUN		Out:=TableCmp(In, Table, Size,AryOut);

变量

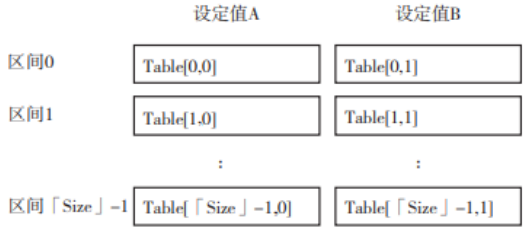
	名称	输入/输出	内容	有效范围	单位	初始值
In	比较数据	输入	要比较的数值	遵从数据类型	—	(*)
Table[ ] 2 维数组	比较表		以各定义区间为元素的 2 维数组			
Size	比较规格		与 “In” 比较的 Table[ ]的元素数			1
AryOut[ ]	单独比较结果数组	输入输出	Table[ ] 的各元素比较结果 TRUE ： 一致 FALSE：不一致	遵从数据类型	—	—
Out	比较结果	输出	TRUE ： Table[ ] 的所有元素与 “In” 一致 FALSE：不一致的元素至少有 1 个	遵从数据类型	—	—

*省略输入参数时，初始值不适用。编连时会发生异常。

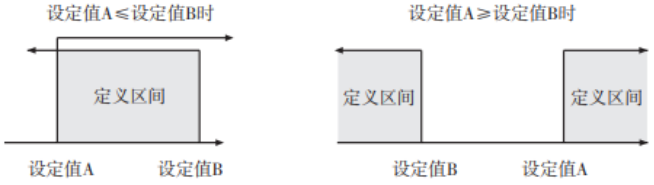
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○	○	○					
Table[ ]2 维数组	带有与 “In” 相同数据类型的元素的 2 维数组																			
Size							○													
AryOut[ ] 数组	○																			
Out	○																			

功能

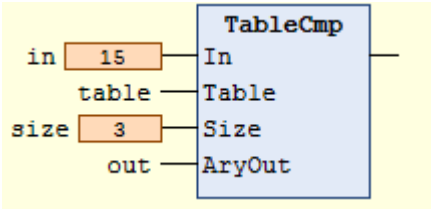
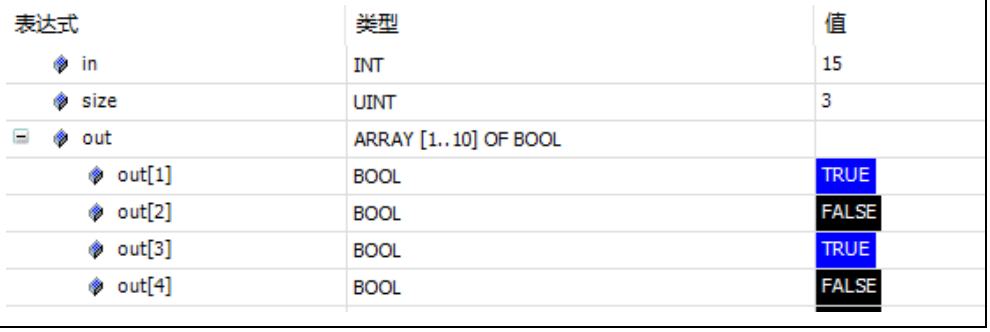
与比较数据 “In” 和比较表 Table[ ] 指定的 “Size” 组的定义区间进行比较。Table[ ]为 2 维数组，第 1 维为定义区间的编号，第 2 维的 0 号元素表示定义区间的设定值 A，1 号元素表示定义区间的设定值 B。



设定值 A 和设定值 B 的定义区间指定方法如下图所示。设定值 A 和设定值 B 的值包含在定义区间内。



“In” 和表 [ ] 的比较结果保存在单独比较结果数组 AryOut[ ] 中。如果 “In” 在编入 i 号的定义区间内，则 AryOut[i] 的值为 TRUE ；如果超出，则 AryOut[i] 的值为 FALSE。如果 AryOut[ ] 的 “Size” 个元素均为 TRUE，则比较结果 “Out” 的值为 TRUE，否则为 FALSE。

	FBD	ST
定义变量	VAR in:INT; size:UINT; out :ARRAY [1..10] OF BOOL; table: ARRAY [1..10,0..1] OF INT; END_VAR	
程序		TableCmp(In:=in , Table:=Table[1,0] , Size:=size , AryOut:=out);
运行结果		

表达式	类型	值
in	INT	15
size	UINT	3
out	ARRAY [1..10] OF BOOL	
out[1]	BOOL	TRUE
out[2]	BOOL	FALSE
out[3]	BOOL	TRUE
out[4]	BOOL	FALSE

table	ARRAY [1..10, 0..1] OF INT	
table[1, 0]	INT	10
table[1, 1]	INT	20
table[2, 0]	INT	20
table[2, 1]	INT	10
table[3, 0]	INT	4
table[3, 1]	INT	20

要点说明

- 请将 “In” 和 Table[ ] 的元素的数据类型设为相同。否则，编连时会发生异常。
- 请务必使 Table[ ] 为 2 维数组。
- Table[ ] 的第 2 维的数组的大小大于等于 3 时，将忽略第 2 维的 2 号之后的元素。
- AryOut[ ]的数组的大小大于等于“Size”时，将比较结果保存在 AryOut[0]～AryOut[“Size”-1]中。其他 的数组元素不变。
- 带符号整数型（SINT，INT，DINT，LINT）和无符号整数型（USINT，UINT，UDINT，ULINT）无法比较。
- 比较对象为实数时，如果含有除不尽的除法结果等，由于误差的原因，处理结果可能会出现意外。
- “Size” 的值为 0 时，“Out” 的值为 FALSE，AryOut[ ] 不变。
- 通过梯形图程序使用本指令时，如果本指令的电路前段发生异常，则 “Out” 的值为 FALSE。
- 以下情况时会发生异常。“Out” 为 FALSE。
- Table[ ] 的第 2 维的数组大小为 1 时
- “Size” 的值超出 AryOut[ ] 的数组大小时。
- “Size” 的值超出 Table[ ] 的第 1 维的数组大小时。

6.1.3. AryCmpNE (排列批量比较)

比较 2 个数组的各元素，判定是否不同。

指令	名称	FB/FUN	图形表现	ST 表现
AryCmpNE	数组整体比较 NE	FUN		AryCmpNE(In1, In2, Size, AryOut);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In[ ],In2[ ] 数组	比较数据	输入	以要比较的数值为元素的数组	遵从数据类型	—	(*)
Size	比较元素数		要比较的元素数			1
AryOut[ ] 数组	比较结果数组	输入输出	比较结果数组	遵从数据类型	—	—
Out	返回值	输出	始终为 TRUE	仅 TRUE	—	—

*省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○					
Table[ ] 2 维数组	与 In1[ ]相同数据类型的数组																			
Size							○													
AryOut[ ] 数组	○																			
Out	○																			

功能

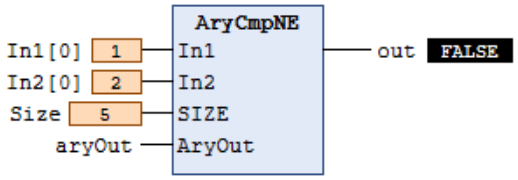
2 个数组 In1[0] ～ In1[「Size」-1]、In2[0] ～ In2[“Size”-1] 的相同元素编号之间进行比较。将比较结果保存在比较结果数组 AryOut[0] ～ AryOut[“Size”-1] 对应的元素编号中。

待比较数组 In1 和 In2 初始化如下所示，“Size”=5，示例如下。

	FBD	ST
定义变量		

```
PROGRAM PLC_PRG
VAR
    In1: ARRAY[0..9] OF BYTE := [1, 2, 5, 6, 8, 5(0)];
    In2: ARRAY[0..9] OF BYTE := [2, 2, 5, 7, 8, 5(0)];
    Size        :UINT:=5;
    aryOut      :ARRAY [0..9] OF BOOL;
    out         :BOOL;
END_VAR
```

程序




```
out FALSE :=AryCmpNE(
      In1:= In1[0] 1,
      In2:= In2[0] 2,
      SIZE:= Size 5,
      AryOut:= aryOut);RETURN
```

运行结果

Device.Application.PLC_PRG

表达式	类型	值
In1	ARRAY [0..9] ...	
In2	ARRAY [0..9] ...	
Size	UINT	5
aryOut	ARRAY [0..9] ...	
aryOut[0]	BOOL	TRUE
aryOut[1]	BOOL	FALSE
aryOut[2]	BOOL	FALSE
aryOut[3]	BOOL	TRUE
aryOut[4]	BOOL	FALSE
aryOut[5]	BOOL	FALSE
aryOut[6]	BOOL	FALSE
aryOut[7]	BOOL	FALSE
aryOut[8]	BOOL	FALSE
aryOut[9]	BOOL	FALSE
out	BOOL	FALSE

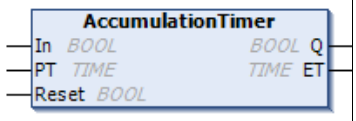
要点说明

- 请将 In1[ ] 和 In2[ ] 的数据类型设为相同。
- 请使 AryOut[ ] 数组的大小大于等于 “Size”。
- In1[]、In2[ ] 为实数时，如果含有除不尽的除法结果等，由于误差的原因，处理结果可能会出现意外。
- “Size” 的值为 0 时，“Out” 的值为 TRUE，AryOut[ ] 不变。
- 在 ST 程序中使用本指令时，不使用返回值 “Out”。
- 以下情况时会发生异常。ENO 为 FALSE，AryOut[ ] 不变。
- In1[ ] 和 In2[ ] 的数据类型不同时。
- In1[ ]、In2[ ]、AryOut[ ] 中任一数组大小小于 “Size” 时。

6.2. 定时器指令

6.2.1. AccumulatioTimer (累积定时器)

累计定时器输入为 TRUE 的时间的计时器。

指令	名称	FB/FUN	图形表现	ST 表现
Accumulation Timer	累计定时器	FB		AccumulationTimer(In, PT, Reset, Q, ET);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	定时器输入	输入	TRUE：定时器工作 FALSE：定时器停止	遵从数据类型	—	FALSE
PT	设定时间		计时的最大值	(*)	ms	0
Reset	复位		TRUE：定时器工作 FALSE：定时器停止	遵从数据类型	—	FALSE
Q	定时器输出	输出	TRUE： “ET” 到达 “PT” FALSE： “ET” 未到达 “PT”	遵从数据类型	—	—
ET	累计时间		累计时间	(*)	ms	

* T#0ms ～ T#106751d_23h_47m_16s_854.775807ms

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○																			
PT																○				
Reset	○																			
Q	○																			
ET																○				

功能

累计定时器输入 “In” 为 TRUE 的时间的定时器。设定时间单位为 ns，计时精度为 100ns。

复位“Reset”为 FALSE 时，“In”从 FALSE 变为 TRUE 后，定时器启动，累计时间“ET”与时间同时增

加。

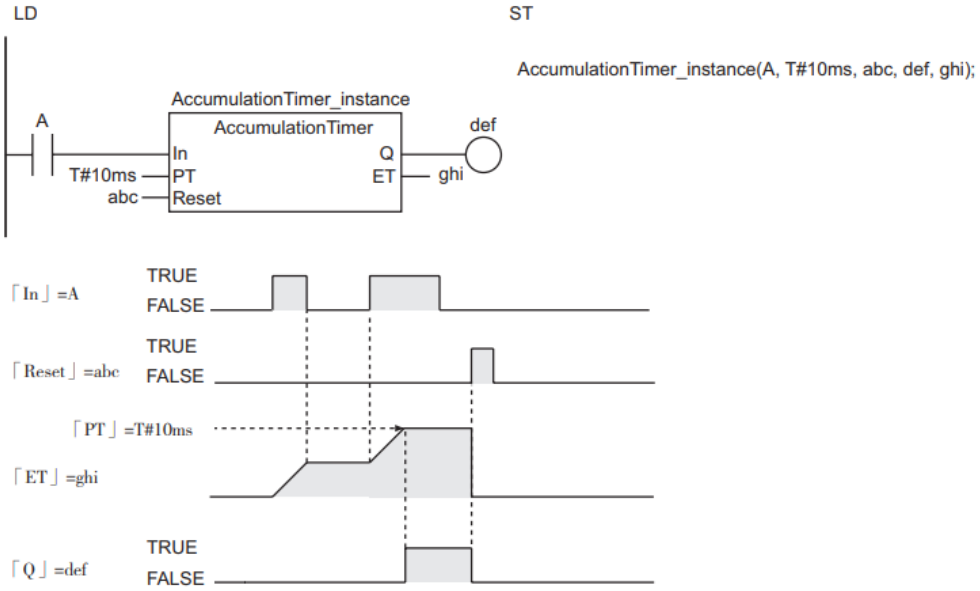
“In” 变为 FALSE 时，定时器停止。但保持该时刻的 “ET” 值。

“In” 再次变为 TRUE 时，定时器再次启动。“ET” 从保持值开始增加。

“ET” 到达设定时间 “PT” 时，定时器输出 “Q” 变为 TRUE。此时，“ET” 停止增加。

“Reset” 变为 TRUE 时，定时器复位。“ET” 的值变为 0，“Q” 变为 FALSE。

“PT” =T#10ms 时的示例和时序图如下所示。变量 A 为 TRUE 的累计时间到达 10ms 时，变量 abc 的值变为 TRUE。

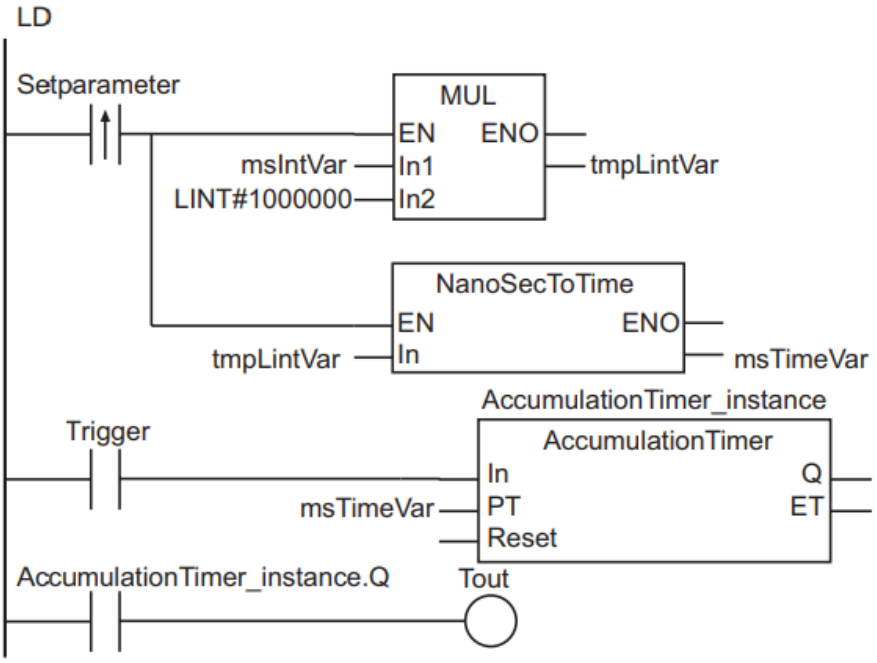


参考

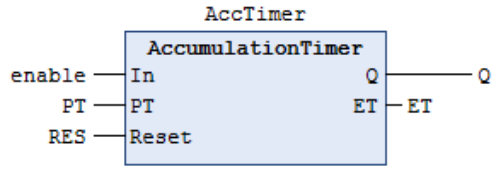
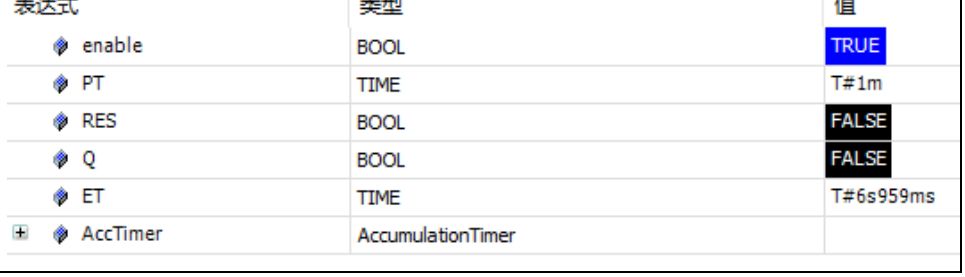
“In” 变为 FALSE 时，如果要将定时器输出和经过时间复位，请使用 “TON 指令 (P.2-124)”。

连接不支持 TIME 型的触摸屏等时，需先将以整数型表示的设定时间转换为 TIME 型后，再输入至本指令。

从整数型转换为 TIME 型时，请使用 “NanoSecToTime 指令 (P.2-617)”。从 TIME 型转换为整数型时，请使用 “TimeToNanoSec 指令(P.2-615)”。上述指令的时间单位均为纳秒。INT 型变量 msIntVar 的设定时间以 ms 为单位时的用户程序如下所示。



使用举例如下所示

	FBD	ST
定义变量	VAR enable:BOOL; PT:TIME; RES:BOOL; Q:BOOL; ET:TIME; AccTimer: AccumulationTimer; END_VAR	
程序		AccTimer(In:=enable , PT:=PT , Reset:=RES , Q=>Q , ET=>ET);
运行结果		

表达式	类型	值
enable	BOOL	TRUE
PT	TIME	T#1m
RES	BOOL	FALSE
Q	BOOL	FALSE
ET	TIME	T#6s959ms
AccTimer	AccumulationTimer	

要点说明

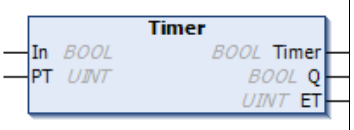
- “ET”和“Q”在执行本指令时更新。因此，严格来说，“Q”变为 TRUE 的条件不是定时器工作的累计时间等于 “PT” 的时刻。而是定时器工作的累计时间到达 “PT” 后，首次执行本指令的时刻。因此， 会发生最大 1 个任务周期的延迟。
- “PT”、“ET” 以 ns 为单位进行设定，计时精度为 100ns。
- “In” 和 “Reset” 均变为 TRUE 时，“Reset” 优先。即 “ET” 的值变为 0，“Q” 变为 FALSE。
- 如果开始运行时 “In” 的值已为 TRUE，则从该时刻开始计时。
- 为 “PT” 设定了 T#0ms 或负数时，“In” 的值变为 TRUE 后，“Q” 变为 TRUE。
- “ET” 的值到达 “PT” 的值之前，可变更 “PT” 的值。此时的动作如下所示。

定时器的状态	“Q”的值	变更后的“PT”值	动作
计时结束后	TRUE	—	“Q” 的值保持为 TRUE。 “ET” 的值也不变（保持变更前的 “PT” 的值）。
计时中	FALSE	“PT” ≥ “ET”	“In” 的值变为 TRUE 时，继续计时。“ET”的值达到变更后 的“PT” 的值时，“Q” 的值变为 TRUE，“ET” 停止增加。
		“PT” < “ET”	“In” 的值变为 TRUE 时，“Q” 的值立即变为 TRUE。 “ET” 也立即停止增加。

- 本指令存在于主站控制区域，通过主站控制执行复位时，动作如下所示。
- 定时器停止。“ET” 和 “Q” 保持该时刻的值。
- 通过主站控制解除复位时，“ET” 从保持值重新开始增加。
- 将 “Q” 连接至后段的 Out 指令时，“Q” 的值即使为 TRUE，Out 指令中也会输入 FALSE。
- “Reset” 为有效。
- 因执行 JMP 系统指令(JMP 指令等) 而未执行本指令时，不更新 “ET”的值。但该期间仍将继续计时。因 此，之后执行本指令时，“ET” 将更新为正确的值。
- 通过梯形图程序使用本指令时，如果本指令的电路前段发生异常，则 “Q” 的值为 FALSE。

6. 2. 2. Timer (100ms 定时器)

在启动经过设定时间后，输出 TRUE 的定时器。设定时间单位和计时精度均为 100ms。

指令	名称	FB/FUN	图形表现	ST 表现
Timer	100ms 定时器	FUN		Out:=Timer (In, PT,TimerDat, Q, ET);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	定时器输入	输入	TRUE：定时器工作 FALSE：定时器停止	遵从数据类型	—	FALSE
PT	设定时间		从定时器启动到“Q”变为 TRUE 的时间		ms	(*)
Out	返回值	输出	TRUE ：定时器输出 ON FALSE：定时器输出 OFF	遵从数据类型	—	—
Q	定时器输出		与“Out”意义相同			
ET	累计时间		剩余时间		ms	

*省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○																			
PT							○													
Reset	○																			
Q	○																			
ET							○													

功能

在启动经过设定时间后，输出 TRUE 的定时器。设定时间单位和计时单位均为 100ms。

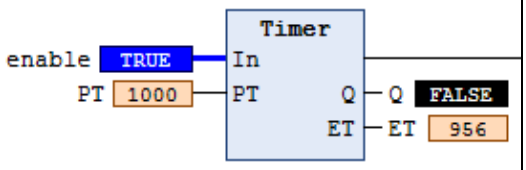
定时器输入 “In” 的值变为 FALSE 时，定时器复位。为剩余时间 “ET” 设定设定时间 “PT”，定时器输出 “Q” 的值变为 FALSE。

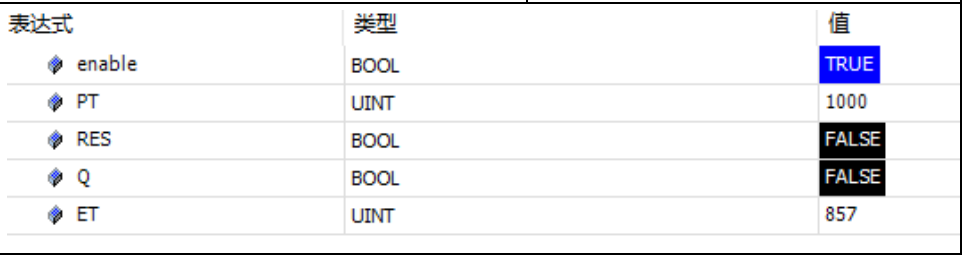
“In” 从 TRUE 变为 FALSE 时，定时器启动。“ET” 的值与时间同时减少。

“ET” 的值到达 0 时，定时器输出 “Q” 变为 TRUE。此时，停止减少 “ET” 的值。

定时器启动后，即使在 “ET” 到达 0 之前，“In” 变为 FALSE 时，定时器也将复位。

“PT”= UINT#10 时的示例和时序图如下所示，变量 enable 变为 TRUE 的 1000ms（1s）后，变量 Q 的值变为 TRUE。

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">
<pre>VAR
    enable:BOOL;
    PT:UINT;
    RES:BOOL;
    Q:BOOL;
    ET:UINT;
END_VAR</pre>
</td></tr>
<tr><td>程序</td><td>


</td><td>
<pre>Timer(
  In:=enable ,
  PT:=PT ,
  Q=>Q ,
  ET=>ET);</pre>
</td></tr>
<tr><td>运行结果</td><td colspan="2">


</td></tr>
</table>

参考

要进行更准确的计时，请使用 “TON 指令 (P.2-124)”，以 100ns 为单位进行计时。TON 指令在执行指令时以 100ns 为单位进行计时，因此可比 Timer 指令更准确地进行计时。但 Timer 指令的指令执行时间较短。

要点说明

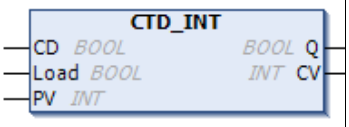
- 在记述本指令的 POU 的开头进行计时。因此，在相同 POU 内的无论何处执行本指令，“ET”的值均相 同。
- “Q”在执行本指令时更新。因此，严格来说，“Q”变为 TRUE 条件不是定时器启动后的经过时间等于 “PT”的时刻。“Q”变为 TRUE 条件为在定时器启动后的经过时间达到“PT”后，首次执行本指令的 时刻。因此，会发生最大 1 个任务周期的延迟。
- “TimerDat” 为输入输出变量，无需传输值。请确保结构体 _sTimer 所需的存储区域，传输至本指令。
- 请勿变更 “TimerDat” 的内容。

- 如果开始运行时 “In” 的值已为 TRUE，则从该时刻开始计时。
- 变更了 “PT” 的值后，将在之后使定时器复位时反映。计时过程中不会反映。 • 本指令存在于主站控制区域，通过主站控制执行复位时，复位定时器。“ET”中设定“PT”的值，“Q” 的值变为 FALSE。
- 因执行 JMP 系统指令(JMP 指令等) 而未执行本指令时，不更新 “ET”的值。但该期间仍将继续计时。因 此，之后执行本指令时，“ET” 将更新为正确的值。
- 通过梯形图程序使用本指令时，如果本指令的电路前段发生异常，则“Q”和“Out”的值变为 FALSE。

6.3. 计数器指令

6.3.1. CTD_** (减法计数器组)

每次输入计数器输入信号时进行减法运算的计数器。预设值、计数值的数据类型为 INT、DINT、LINT、UDINT、ULINT 中的任意一种。

指令	名称	FB/FUN	图形表现	ST 表现
CTD_**	减法计数器组	FB		CTD_** (CD, Load, PV,Q, CV); ** 为 INT、 DINT、LINT、UDINT、ULINT 中的任意一个

变量

	名称	输入/输出	内容	有效范围	单位	初始值
CD	计数器输入	输入	计数器输入	遵从数据类型	—	FALSE
Load	加载信号		TRUE：向 “CV” 加载 “PV”			
PV	预设值		计数器的预设值	遵从数据类型		0
Q	计数器输出	输出	TRUE ：计数器输出 ON FALSE：计数器输出 OFF	遵从数据类型	—	—
CV	计数值		计数器的当前值	遵从数据类型		

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CD	○																			
Load	○																			
PV								○	○			○	○							
Q	○																			
CV	与“PV”相同的数据类型																			

功能

减法计数器。预设值、计数值的数据类型为 INT、DINT、LINT、UDINT、ULINT 中的任意一种。

指令名称因 “PV” 和 “CV” 的数据类型而异。例如，两者均为 LINT 型时，指令名称为 CTD_LINT。

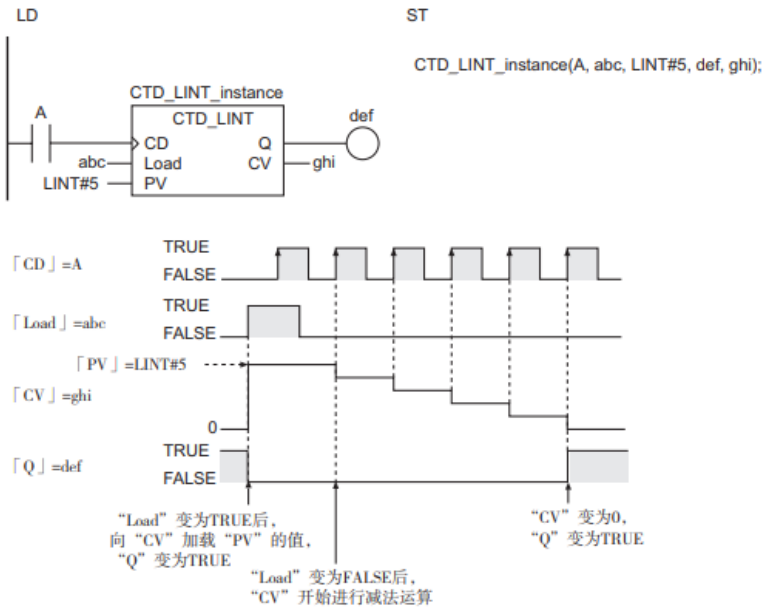
加载信号 “Load” 设置为 TRUE 时，将预设值 “PV” 的值加载到计数值 “CV” 中，计数器输出 “Q” 变为 FALSE。

计数器输入信号 “CD” 处于上升沿时，使 “CV” 进行减法运算。“CV” 的值小于 0 时，“Q” 的值变为 TRUE。

“CV” 的值小于 0 时，即使 “CD” 处于上升沿，“CV” 也不会变化。

“Load” 为 TRUE 期间，忽略 “CD”。“CV” 不进行减法运算。

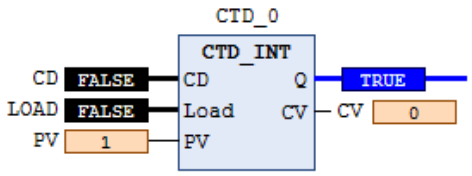
CTD_LINT 指令下，“PV”=LINT#5 时的示例和时序图如下所示。



<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">

```
VAR
    CD:BOOL;
    LOAD:BOOL;
    PV:INT;
    Q:BOOL;
    CV:INT;
    CTD_0:CTD_INT;
END_VAR
```

</td></tr>
<tr><td>程序</td><td>



</td><td>

```
CTD_0(
    CD:=CD ,
    LOAD:=LOAD ,
    PV:=PV ,
    Q=>Q ,
    CV=>CV );
```

</td></tr>
<tr><td>运行结果</td><td colspan="2">

表达式	类型	值
CD	BOOL	FALSE
LOAD	BOOL	FALSE
PV	INT	1
Q	BOOL	TRUE
CV	INT	0

</td></tr>
</table>

参考

如果需要计数器每次输入计数器输入信号时进行加法运算， 请使用 “CTU_** ”。

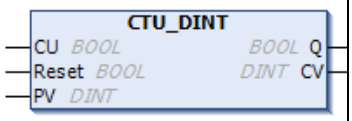
如果需要计数器同时进行加法运算和减法运算，请使用 “CTUD 指令 (P.2-152)”。

要点说明

- 倒计数结束后要使计数器再次启动时，请先将 “Load” 的值设为 TRUE 后再设为 FALSE。
- 请将 “PV” 和 “CV” 的数据类型设为相同。
- 为 “PV” 设定了负数时，在 “Load” 的值变为 TRUE 时，将 “PV” 的值加载到 “CV” 中。“CV” 的 值小于 0，因此 “Q” 的值会立即变为 TRUE。之后，即使 “CD” 变化，“CV” 也不进行减法运算。
- “CD” 的值为 FALSE 的状态下发生电源断开或由程序控制动作模式后，重新开始执行本指令时，如果 “CD” 的值变为 TRUE，则 “CV” 进行 1 次减法运算。
- 通过梯形图程序使用本指令时，如果本指令的电路前段发生异常，则 “Q” 的值变为 FALSE。

6.3.2. CTU_** (加法计数器组)

每次输入计数器输入信号时进行加法运算的计数器。预设值、计数值的数据类型为 INT、DINT、LINT、UDINT、ULINT 中的任意一种。

指令	名称	FB/FUN	图形表现	ST 表现
CTU_**	加法计数器组	FB		CTU_** (CU, Reset, PV, Q, CV); ** 为 INT、DINT、LINT、UDINT、ULINT 中的任意一个

变量

	名称	输入/输出	内容	有效范围	单位	初始值
CU	计数器输入	输入	计数器输入	遵从数据类型	—	FALSE
Reset	复位信号		TRUE：向 “CV” 复位为 0			
PV	预设值		计数器的预设值	遵从数据类型		0
Q	计数器输出	输出	TRUE ：计数器输出 ON FALSE：计数器输出 OFF	遵从数据类型	—	—
CV	计数值	输入	计数器的当前值	遵从数据类型		

*省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	○																			
Reset	○																			
PV								○	○			○	○							
Q	○																			
CV	与 “PV” 相同的数据类型																			

功能

加法计数器。预设值、计数值的数据类型为 DINT、LINT、UDINT、ULINT 中的任意一种。 指令名称因 “PV” 和 “CV” 的数据类型而异。例如，两者均为 LINT 型时，指令名称为 CTU_LINT。

复位信号 “Reset” 设置为 TRUE 时，计数值 “CV” 的值变为 0，计数器输出 “Q” 变为 FALSE。

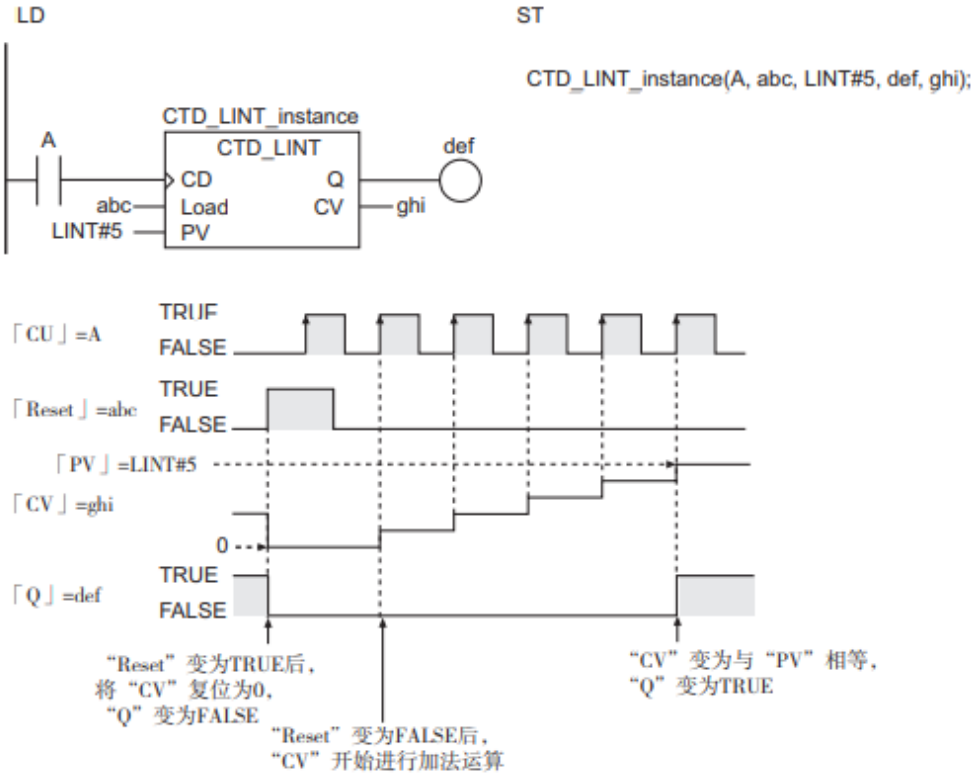
计数器输入信号 “CU” 处于上升沿时，使 “CV” 进行加法运算。“CV” 的值大于预设值 “PV” 的

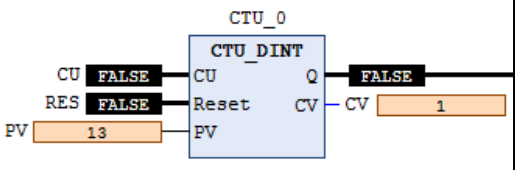
值 时，“Q” 的值变为 TRUE。

“CV” 的值大于 “PV” 的值时，即使输入更大的 “CU” 的值，“CV” 也不会变化。

“Rreset” 为 TRUE 期间，忽略 “CU”。“CV” 不进行加法运算。

CTU_LINT 指令下，“PV”=LINT#5 时的示例和时序图如下所示。



	FBD	ST
定义变量	VAR CU:BOOL; RES:BOOL; PV:DINT; Q:BOOL; CV:DINT; CTU_0:CTU_DINT; END VAR	
程序		CTU_0(CU:=CU , Reset:=RES , PV:=PV , Q=>Q , CV=>CV);

运行结果	表达式	类型	值
	CU	BOOL	FALSE
	RES	BOOL	FALSE
	PV	DINT	13
	Q	BOOL	TRUE
	CV	DINT	13

参考

如果需要计数器每次输入计数器输入信号时进行减法运算， 请使用 “CTD_** 指令”。

如果需要计数器同时进行加法运算和减法运算，请使用 “CTUD_** 指令”。

要点说明

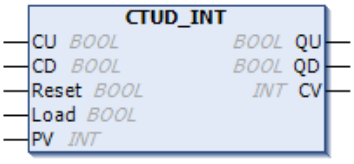
- 计数结束后要使计数器再次启动，请先将 “Reset” 的值设为 TRUE 后再设为 FALSE。
- 为“PV”设定了负数时，在“Reset”的值变为 TRUE 时， “CV”的值变为 0。“CV”的值大于 “PV” 的值，因此 “Q” 的值会立即变为 TRUE。之后，即使 “CU” 变化，“CV” 也不进行加法运算。
- 请将 “PV” 和 “CV” 的数据类型设为相同。
- “Reset” 的值为 FALSE 的状态下， “PV” 的值如有变更，则其动作如下所示。

指令	含义
大于该时刻的“CV”	继续计数
小于该时刻的“CV”	计数结束。“Q”的值变为 TRUE。“CV”的值报错该时刻不变。

- “CU” 的值为 FALSE 状态下发生电源断开或由程序控制动作模式后，重新开始执行本指令时，如果 “CU” 的值变为 TRUE，则 “CV” 进行 1 次加法运算。
- 通过梯形图程序使用本指令时，如果本指令的电路前段发生异常，则 “Q” 的值变为 FALSE。

6.3.3. CTUD_** (可逆计数器组)

根据加法计数器输入和减法计数器输入进行加减法运算的计数器。预设值、计数值的数据类型为 INT、DINT、LINT、UDINT、ULINT 中的任意一种。

指令	名称	FB/FUN	图形表现	ST 表现
CTUD_**	可逆计数器组	FB		CTUD_** (CU, CD, Reset, Load, PV, QU, QD, CV); * 为 INT、DINT、LINT、UDINT、ULINT 中的任意一个

变量

	名称	输入/输出	内容	有效范围	单位	初始值
CU	加法计数器输入	输入	加法计数器输入	遵从数据类型	—	FALSE
CD	减法计数器输入		减法计数器输入			
Reset	复位信号		TRUE：将 “CV” 复位为 0。			0
Load	加载信号		TRUE：向 “CV” 加载 “PV”。			
PV	预设值		加法计数器的计数结束值 减法计数器的初始值	遵从数据类型		
QU	加法计数器输出	输出	TRUE：加法计数器输出 ON FALSE：加法计数器输出 OFF	遵从数据类型	—	—
QD	减法计数器输出		TRUE：减法计数器输出 ON FALSE：减法计数器输出 OFF			
CV	计数值		计数器的当前值	遵从数据类型		

*省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
CU	○																			
CD	○																			
Reset	○																			
Load	○																			
PV								○	○			○	○							
QU	○																			
QD	○																			
CV	与 “PV” 相同的数据类型																			

功能

根据加法计数器输入信号和减法计数器输入信号进行加减法运算的计数器。兼具加法计数器和减法计数器两者的功能。

预设值、计数值的数据类型为 DINT、LINT、UDINT、ULINT 中的任意一种。指令名称因 “PV” 和 “CV” 的数据类型而异。例如，两者均为 LINT 型时，指令名称为 CTUD_LINT。

加法计数器的功能

复位信号 “Reset” 设置为 TRUE 时，计数值 “CV” 的值变为 0，加法计数器输出 “QU” 变为 FALSE。加法计数器输入信号“CU”处于上升沿时，使“CV”进行加法运算。“CV”的值大于预设值“PV”的值时，“QU” 的值变为 TRUE。

“CV” 的值大于 “PV” 的值时，即使输入更大的 “CU” 的值，“CV” 也不会变化。

减法计数器的功能

加载信号 “Load” 设置为 TRUE 时，将预设值 “PV” 的值加载到计数值 “CV” 中，减法计数器输出“QD” 变为 FALSE。减法计数器输入信号 “CD” 处于上升沿时，使 “CV” 进行减法运算。

“CV” 的值小于 0 时，“QD” 的值变为 TRUE。

“CV” 的值小于 0 时，即使输入更大的 “CD” 的值，“CV” 也不会变化。

加法计数器、减法计数器的共同功能

“Load” 或 “Reset” 为 TRUE 期间，忽略 “CU” 和 “CD”。“CV” 不进行加减法运算。

“CU” 和 “CD” 同时处于上升沿时，“CV” 不会变化。

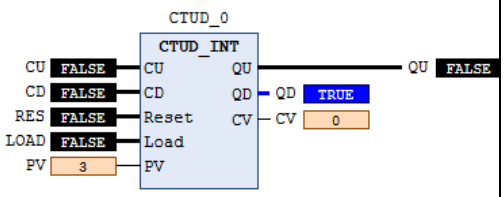
“Reset” 和 “Load” 均为 TRUE 时，“Reset” 优先，“CV” 的值变为 0。

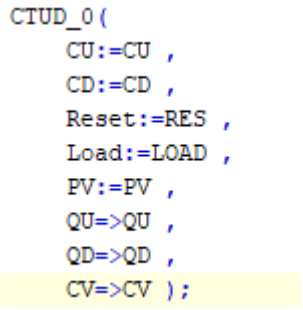
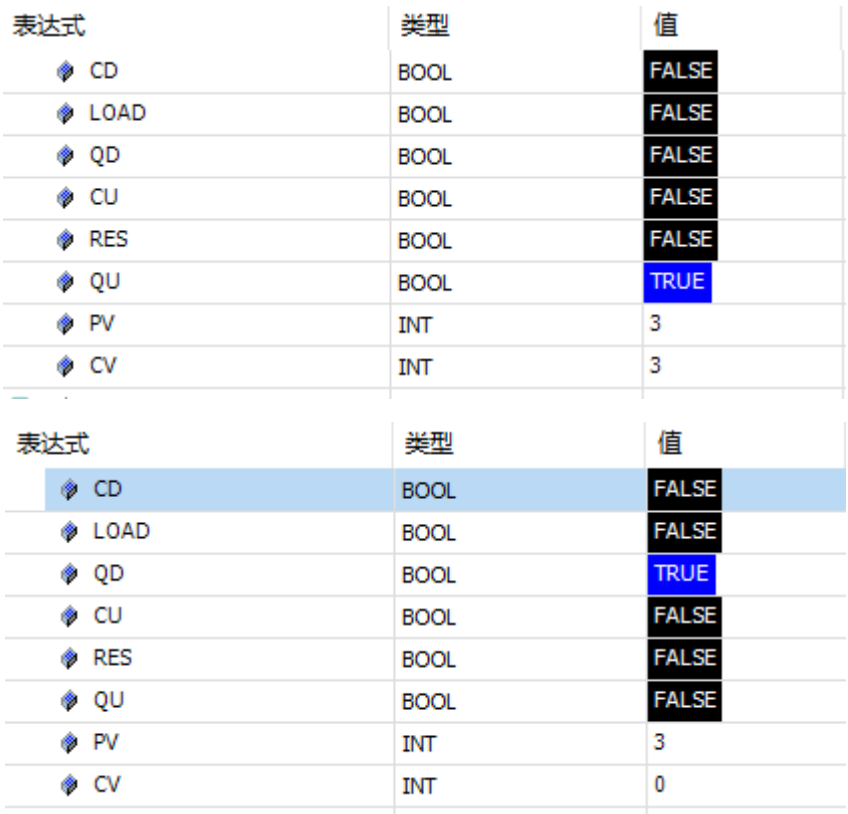
“Reset” 设置为 TRUE 时，“CV” 的值变为 0，因此 “QD” 的值变为 TRUE。

“Load” 设置为 TRUE 时，“CV” 的值与 “PV” 相等，因此 “QU” 的值将变为 TRUE。

“Reset”、“Load”、“CV”、“QU”、“QD” 之间的关系如下所示。“PV” 的值设置为大于 0。

	FBD	ST
定义变量		VAR CD:BOOL; LOAD:BOOL; CU:BOOL; RES:BOOL; PV:INT; QU:BOOL; QD:BOOL; CV:INT; CTUD_0:CTUD_INT; END_VAR

<table>
<tr>
<td>程序</td>
<td> 


</td>
</tr>
<tr>
<td>运行结果</td>
<td> 
</td>
</tr>
</table>

参考

如果需要计数器每次输入计数器输入信号时进行减法运算， 请使用 “CTD_** 指令”。

如果需要计数器同时进行加法运算和减法运算，请使用 “CTUD_** 指令”。

要点说明

- 计数结束后要使计数器再次启动，请先将 “Reset” 的值设为 TRUE 后再设为 FALSE。
- 为“PV”设定了负数时，在“Reset”的值变为 TRUE 时， “CV”的值变为 0。“CV”的值大于 “PV” 的值，因此 “Q” 的值会立即变为 TRUE。之后，即使 “CU” 变化，“CV” 也不进行加法运算。
- 请将 “PV” 和 “CV” 的数据类型设为相同。
- “Reset” 的值为 FALSE 的状态下， “PV” 的值如有变更，则其动作如下所示。

指令	含义
大于该时刻的“CV”	继续计数
小于该时刻的“CV”	计数结束。“Q”的值变为 TRUE。“CV”的值报错该时刻不变。

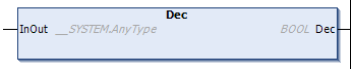
- “CU” 的值为 FALSE 状态下发生电源断开或由程序控制动作模式后，重新开始执行本指令时，如果 “CU” 的值变为 TRUE，则 “CV” 进行 1 次加法运算。
- 通过梯形图程序使用本指令时，如果本指令的电路前段发生异常，则 “Q” 的值变为 FALSE。

6.4. 算术指令

6.4.1. Inc/Dec (增量/减量)

Inc ：对整数值进行增量。

Dec：对整数值进行减量。

指令	名称	FB/FUN	图形表现	ST 表现
Inc	增量	FUN		Inc(InOut);
Dec	减量	FUN		Dec(InOut);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Inout	对象数据	输入	对象数据	遵从数据类型	—	—
Out	返回值		始终为 TRUE	仅 TRUE	—	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
InOut						○	○	○	○	○	○	○	○							
Out	○																			

功能

- Inc：对对象数据 “InOut” 进行增量。最终，超过 “InOut” 的最大值时，恢复到最小值。
- Dec：对对象数据 “InOut” 进行减量。最终，超过 “InOut” 的最小值时，恢复到最大值。

6.4.2. AryAddV (排列要素加法)

数组的各元素加上相同数值。

指令	名称	FB/FUN	图形表现	ST 表现
AryAddV	数组元素加法	FUN		AryAddV(In1, In2, Size, AryOut);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In1[ ]数组	加法数组	输入	加法数组	遵从数据类型	—	(*)
In2	加数		加数			
Size	元素数		待加 In1[ ] 的元素数			0
Aryout[ ] 数组	相加结果数组		相加结果数组			—
Out	返回值	输出	始终为 TRUE	仅 TRUE	—	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[ ] 数组						○	○	○	○	○	○	○	○	○	○					
In2	与 In1[ ] 的源相同的数据类型																			
Size							○													
AryOut[ ] 数组	与 In1[ ] 的源相同的数据类型																			
Out	○																			

功能

加法数组 In1[ ]的从 In1[0]起的“Size”个各元素加上加数“In2”的值，并输出至相加结果数组 AryOut[ ]。

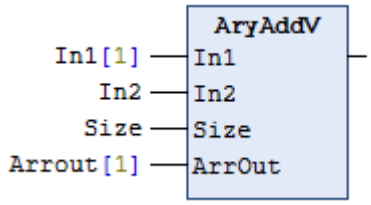
“In2”=INT#11、“Size”=UINT#3 时的示例如下所示。

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">

```
VAR
    In1: ARRAY [1..10] OF INT := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    In2     :INT;
    Size    :UINT;
    Arrout  :ARRAY [1..10] OF INT;

END_VAR
```

</td></tr>
<tr><td>程序</td><td>



</td><td>

```
AryAddV(
    In1:=In1[1]   1  ,
    In2:=In2      3  ,
    Size:=Size    10 ,
    ArrOut:=Arrout[1]  4  );
```

</td></tr>
<tr><td>运行结果</td><td colspan="2">

表达式	类型	值	准备值	地址	注释
In1	ARRAY [1..10] OF INT				
In2	INT	3			
Size	UINT	10			
Arrout	ARRAY [1..10] OF INT				
Arrout[1]	INT	4			
Arrout[2]	INT	5			
Arrout[3]	INT	6			
Arrout[4]	INT	7			
Arrout[5]	INT	8			
Arrout[6]	INT	9			
Arrout[7]	INT	10			
Arrout[8]	INT	11			
Arrout[9]	INT	12			
Arrout[10]	INT	13			

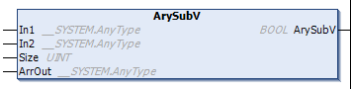
</td></tr>
</table>

要点说明

- 请将 In1[ ]、“In2”、AryOut[ ] 的数据类型设为相同。
- 相加结果在 AryOut[ ] 的有效范围外时，AryOut[ ] 的元素为错误值。此时，不会发生异常。也不会破坏其元素相邻的存储区域。
- “Size” 的值为 0 时，AryOut[ ] 的值不变。
- 在 ST 程序中使用本指令时，不使用返回值 “Out”。
- 以下情况时会发生异常。ENO 为 FALSE，AryOut[ ] 不变。In1[ ]、“In2”、AryOut[ ] 的数据类型不同时。“Size” 的值超过 In1[ ] 或 AryOut[ ] 的数组区域时。

6.4.3. ArySubV (排列要素减法)

数组的各元素减去相同数值。

指令	名称	FB/FUN	图形表现	ST 表现
ArySubV	数组元素减法	FUN		ArySubV(In1, In2, Size, AryOut);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In1[ ]数组	被减数组	输入	被减数组	遵从数据类型	—	(*)
In2	减数		减数			
Size	元素数		元素数			0
Aryout[ ]数组	相减结果数组		相减结果数组			—
Out	返回值	输出	始终为 TRUE	仅 TRUE	—	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[ ] 数组						○	○	○	○	○	○	○	○	○	○					
In2	与 In1[ ] 的源相同的数据类型																			
Size							○													
AryOut[ ] 数组	与 In1[ ] 的源相同的数据类型																			
Out	○																			

功能

被减数组 In1[ ]的从 In1[0]起的“Size”个各元素减去减数“In2”的值，并输出至相减结果数组 AryOut[ ]。

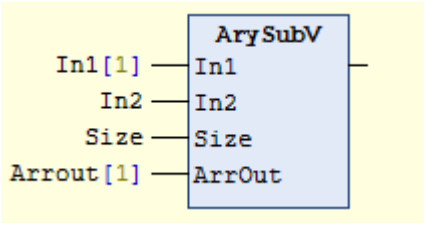
“In2”=INT#11、“Size”=UINT#3 时的示例如下所示。

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">

```
VAR
    In1: ARRAY [1..10] OF INT := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
    In2     :INT;
    Size    :UINT;
    Arrout  :ARRAY [1..10] OF INT;

END_VAR
```

</td></tr>
<tr><td>程序</td><td>



</td><td>

```
ArySubV(
    In1:=In1[1] ,
    In2:=In2 ,
    Size:=Size ,
    ArrOut:=Arrout[1] );
```

</td></tr>
<tr><td>运行结果</td><td colspan="2">

表达式	类型	值	准备值	地址	注释
In1	ARRAY [1..10] OF INT				
In2	INT	3			
Size	UINT	10			
Arrout	ARRAY [1..10] OF INT				
Arrout[1]	INT	-2			
Arrout[2]	INT	-1			
Arrout[3]	INT	0			
Arrout[4]	INT	1			
Arrout[5]	INT	2			
Arrout[6]	INT	3			
Arrout[7]	INT	4			
Arrout[8]	INT	5			
Arrout[9]	INT	6			
Arrout[10]	INT	7			

</td></tr>
</table>

要点说明

- 请将 In1[ ]、“In2”、AryOut[ ] 的数据类型设为相同。
- 相减结果在 AryOut[ ] 的有效范围外时，AryOut[ ] 的元素为错误值。此时，不会发生异常。也不会破坏其元素相邻的存储区域。
- “Size” 的值为 0 时，AryOut[ ] 的值不变。
- 在 ST 程序中使用本指令时，不使用返回值 “Out”。
- 以下情况时会发生异常。ENO 为 FALSE，AryOut[ ] 不变。In1[ ]、“In2”、AryOut[ ] 的数据类型不同时。“Size” 的值超过 In1[ ] 或 AryOut[ ] 的数组区域时。

6.4.4. AryMean (排列要素的平均值运算)

计算数组元素的平均值

指令	名称	FB/FUN	图形表现	ST 表现
AryMean	数组元素的平均值计算	FUN		AryMean(In, Size，Out);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In1[ ]数组	运算对象数组	输入	运算对象数组	遵从数据类型	—	(*)
Size	运算对象的元素数		In[ ]的元素数			0
Out	运算结果		运算结果	遵从数据类型	—	—

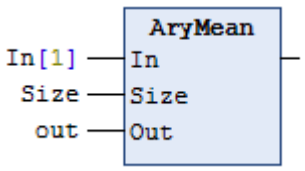
* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[ ] 数组						○	○	○	○	○	○	○	○	○	○	○				
Size							○													
Out						○	○	○	○	○	○	○	○	○	○	○				

功能

计算运算对象数组 In[ ] 的 In[0] 之后的 “Size” 个元素的平均值。

“Size”=UINT#5 时的示例如下所示。

	FBD	ST
定义变量	VAR In :ARRAY [1..10] OF INT := [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] Size :UINT; out :INT; END_VAR	
程序		AryMean(In:=In[1] , Size:=Size , Out:=out) ;

运行结果

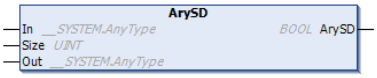
表达式	类型	值	准备值	地址	注释
In	ARRAY [1..10] OF INT				
In[1]	INT	1			
In[2]	INT	2			
In[3]	INT	3			
In[4]	INT	4			
In[5]	INT	5			
In[6]	INT	6			
In[7]	INT	7			
In[8]	INT	8			
In[9]	INT	9			
In[10]	INT	10			
Size	UINT	10			
out	INT	5			

要点说明

- In[ ]、“Out” 为整数型时，舍去平均值的小数点后的数字。
- In[ ] 和 “Out” 的数据类型不同时，请将 “Out” 的有效范围设为包含 In[ ] 的有效范围。
- 运算结果超过 “Out” 的有效范围时，“Out” 的值为错误值。此时，不会发生异常。
- 运算过程的中间值超过 In[ ] 的有效范围时，“Out” 的值为错误值。此时，不会发生异常。
- “Size” 的值为 0 时，“Out” 的值为 0。
- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。“Size” 的值超过 In[ ] 的数组区域时。

6.4.5. ArySD (排列要素的标准差)

计算数组元素的标准偏差

指令	名称	FB/FUN	图形表现	ST 表现
ArySD	数组元素的标准偏差	FUN		ArySD(In, Size，Out);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In1[ ]数组	运算对象数组	输入	运算对象数组	遵从数据类型	—	(*)
Size	元素数		In[ ]的元素数			0
Out	标准偏差		运算结果	遵从数据类型	—	—

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[ ] 数组														○	○					
Size							○													
Out														○	○					

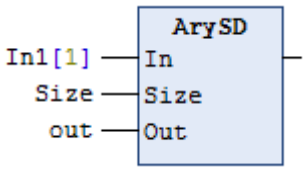
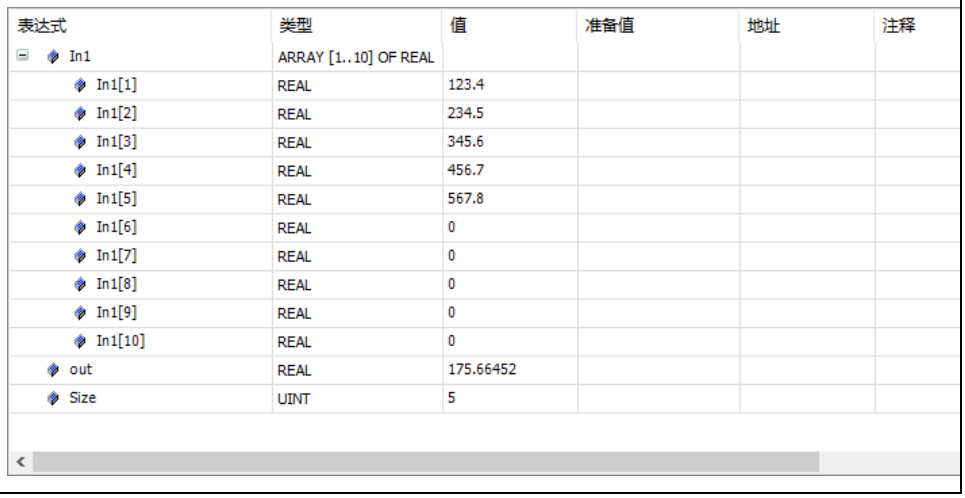
功能

计算运算对象数组 In[ ] 的 In[0] 起的 “Size” 个元素的标准偏差。

$$\text{标准偏差} = \sqrt{\frac{\sum_{i} (In[i]-InM)^2}{\lceil Size \rfloor - 1}}$$

i ：In[ ]的下标 0～“Size”-1
InM：In[0]～In[“Size”-1]的平均值

“Size”=UINT#5 时的示例如下所示。

	FBD	ST
定义变量	VAR In1 :ARRAY [1..10] OF REAL ; out :REAL; Size :UINT; END_VAR	
程序		ArySD(In:=In[1] , Size:=Size , Out:=out);
运行结果		

表达式	类型	值	准备值	地址	注释
In1	ARRAY [1..10] OF REAL				
In1[1]	REAL	123.4			
In1[2]	REAL	234.5			
In1[3]	REAL	345.6			
In1[4]	REAL	456.7			
In1[5]	REAL	567.8			
In1[6]	REAL	0			
In1[7]	REAL	0			
In1[8]	REAL	0			
In1[9]	REAL	0			
In1[10]	REAL	0			
out	REAL	175.66452			
Size	UINT	5			

原理分析	“Size”=UINT#5 In[0]=aryIn[1] 123.4 In[1]=aryIn[2] 234.5 In[2]=aryIn[3] 345.6 In[3]=aryIn[4] 456.7 In[4]=aryIn[5] 567.8 标准差计算 → “Out”=rOut 175.6645

要点说明

- In[]、“Out” 为整数型时，舍去平均值的小数点后的数字。
- In[] 和 “Out” 的数据类型不同时，请将 “Out” 的有效范围设为包含 In[] 的有效范围。
- 运算结果超过 “Out” 的有效范围时，“Out” 的值为错误值。此时，不会发生异常。
- 运算过程的中间值超过 In[] 的有效范围时，“Out” 的值为错误值。此时，不会发生异常。
- “Size” 的值为 0 时，“Out” 的值为 0。
- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。“Size” 的值超过 In[] 的数组区域时。

6.4.6. ModReal (实数余数)

计算对实数进行除法运算后的余数。

指令	名称	FB/FUN	图形表现	ST 表现
ModReal	实数余数	FUN		ModReal(In1, In2，Out);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In1	被除数	输入	被除数	遵从数据类型	—	(*)
In2	除数		除数			
Out	余数		余数	遵从数据类型	—	—

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[] 数组														○	○					
Size														○	○					
Out														○	○					

功能

计算被除数 “In1” 除以除数 “In2” 后的余数。

本指令的运算通过下式进行。

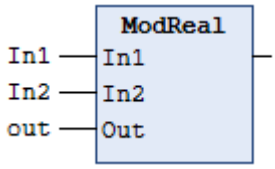
“Out” = “In1” -(“In1” / “In2”)* “In2” 括弧内的除法舍去小数点后的数字。

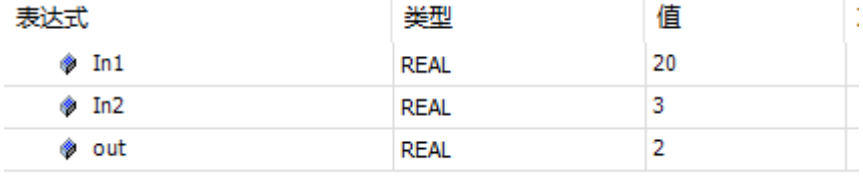
因此，“In1”、“In2” 及 “Out” 的值的示例如下所示。

“In1” 的值	“In2” 的值	“Out” 的值
9.9	3.14	0.48
9.9	-3.14	0.48
-9.9	3.14	-0.48
-9.9	-3.14	-0.48

“In1” =REAL#20、“In2” =REAL#3 时的示例如下所示。变量 abc 的值为 REAL#2。

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">
<pre>VAR
    In1   :REAL;
    In2   :REAL;
    out   :REAL;

END_VAR</pre>
</td></tr>
<tr><td>程序</td><td>


</td><td>
<pre>ModReal(
    In1:=In1 ,
    In2:=In2 ,
    Out:=out);</pre>
</td></tr>
<tr><td>运行结果</td><td colspan="2">


</td></tr>
</table>

参考

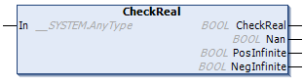
请使用 “CheckReal 指令” 检测 “Out” 的值是否为 + ∞、- ∞、非数。

要点说明

- 请将 “In1”、“In2”、“Out” 的属性设置为相同类型使用。

6.4.7. CheckReal (实数检查)

判定实数是否无限大或非数。

指令	名称	FB/FUN	图形表现	ST 表现
CheckReal	实数检查	FUN		CheckReal(In, Nan，PosInfinite, NegInfinite);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	实数	输入	实数	遵从数据类型	—	(*)
Out	返回值	输出	始终为 TRUE	仅为 TRUE		
Nan	非数判定结果		TRUE ： 非数 FALSE：不是非数	遵从数据类型	—	—
PosInfinite	正无穷大判定结果		TRUE ： 正无穷大 FALSE：不是正无穷大			
NegInfinite	负无穷大判定结果		TRUE ： 负无穷大 FALSE：不是负无穷大			

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out	○																			
Nan	○																			
PosInfinite	○																			
NegInfinite	○																			

功能

判定实数 “In” 是否为非数、正无穷大、负无穷大，将结果分别输出至 “Nan”、“PosInfinite”、“NegInfinite”。

“In”=10#2.44 时使用举例如下

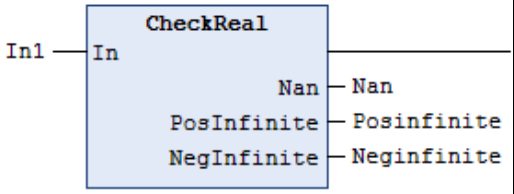
<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">

```
VAR

    In1      :REAL;
    Nan      :BOOL;
    Posinfinite:BOOL;
    Neginfinite:BOOL;

END_VAR
```

</td></tr>
<tr><td>程序</td><td>



</td><td>

```
CheckReal(
    In:=In1 ,
    Nan=>Nan ,
    PosInfinite=>Posinfinite ,
    NegInfinite=>Neginfinite );
```

</td></tr>
<tr><td>运行结果</td><td colspan="2">

表达式	类型	值
In1	REAL	2.44
Nan	BOOL	FALSE
Posinfinite	BOOL	FALSE
Neginfinite	BOOL	FALSE

</td></tr>
</table>

参考

检测使用实数的运算指令的结果是否为非数、正无穷大、负无穷大时，请使用本指令。

要点说明

- 在 ST 程序中使用本指令时，不使用返回值 “Out”。

6.5. 位串运算指令

6.5.1. AryAnd/AryOr/AryXor/AryXorN

对数组间各元素的布尔、位串的每 1 位进行运算。

AryAnd ： 逻辑积

AryOr ： 逻辑和

AryXor ： 异或

AryXorN ： 同或

指令	名称	FB/FUN	图形表现	ST 表现
AryAnd	数组逻辑积	FUN		AryAnd(In1, In2, Size, AryOut);
AryOr	数组逻辑和	FUN		AryOr(In1, In2, Size, AryOut);
AryXor	数组异或	FUN		AryXor(In1, In2, Size, AryOut);
AryXorN	数组同或	FUN		AryXorN(In1, In2, Size, AryOut);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In1[ ],In2[ ]数组	运算对象数组	输入	运算对象数组	遵从数据类型	—	(*)
Size	元素数		运算对象元素数			0
AryOut[ ]数组	运算结果数组		运算结果数组	遵从数据类型	—	—
Out	返回值	输出	始终为 TRUE	仅 TRUE	—	—

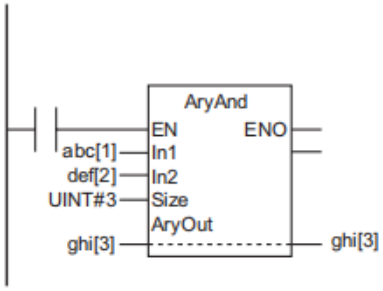
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1[ ]数组	○	○	○	○	○															
Out	与 In1[ ]相同的数据类型																			
Size							○													
AryOut[ ]	与 In1[ ]相同的数据类型																			
Out	○																			

功能

对运算对象数组 In1[ ]、In2[ ] 开头的 “Size” 个元素，对每个对应元素的对应位进行逻辑运算。运算结果保存在 AryOut[ ] 的对应元素中。因此，In1[ ]、In2[ ] 及 AryOut[ ] 的数据类型必须相同。

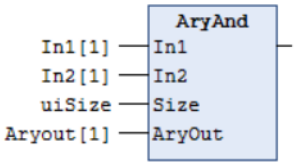
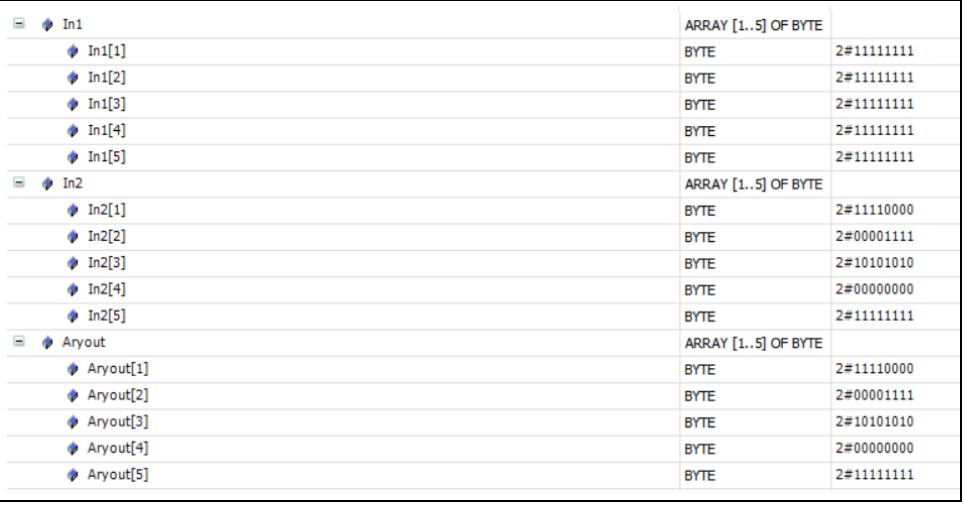
AryAnd 指令下，“Size”=UINT#3 时的示例如下所示。

LD



ST

```
AryAnd(abc[1], def[2], UINT#3, ghi[3]);
```

	FBD	ST
定义变量	VAR In1 :ARRAY[1..5] OF BYTE; In2 :ARRAY[1..5] OF BYTE; Aryout :ARRAY[1..5] OF BYTE; uiSize :UINT:=5; END_VAR	
程序		AryAnd(In1:=In1[1] , In2:=In2[1] , Size:=uiSize , AryOut:=Aryout[1]);
运行结果		

In1	ARRAY [1..5] OF BYTE	
In1[1]	BYTE	2#11111111
In1[2]	BYTE	2#11111111
In1[3]	BYTE	2#11111111
In1[4]	BYTE	2#11111111
In1[5]	BYTE	2#11111111
In2	ARRAY [1..5] OF BYTE	
In2[1]	BYTE	2#11110000
In2[2]	BYTE	2#00001111
In2[3]	BYTE	2#10101010
In2[4]	BYTE	2#00000000
In2[5]	BYTE	2#11111111
Aryout	ARRAY [1..5] OF BYTE	
Aryout[1]	BYTE	2#11110000
Aryout[2]	BYTE	2#00001111
Aryout[3]	BYTE	2#10101010
Aryout[4]	BYTE	2#00000000
Aryout[5]	BYTE	2#11111111

要点说明

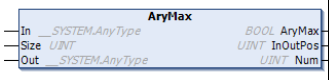
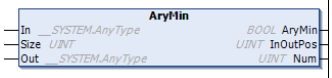
- 请将 In1[ ]、In2[ ]、AryOut[ ] 的数据类型全部设为相同。
- 请将 AryOut[ ] 的数组元素数设为 “Size” 个以上。
- “Size” 的值为 0 时，AryOut[ ] 的值不变。
- 在 ST 程序中使用本指令时，不使用返回值 “Out”。
- 以下情况时会发生异常。ENO 为 FALSE，AryOut[ ] 不变。
- In1[ ]、In2[ ]、AryOut[ ] 的数据类型不同时。
- “Size” 的值超过 In1[ ]、In2[ ]、AryOut[ ] 中任一元素数时。

6.6. 选择指令

6.6.1. AryMax/AryMin (排列变量的最大/小值检索)

AryMax：检索 1 维数组元素的最大值。

AryMin ：检索 1 维数组元素的最小值。

指令	名称	FB/FUN	图形表现	ST 表现
AryMax	数组变量的最大值检索	FUN		Out:=AryMax(In、Size、InOutPos、Num);
AryMin	数组变量的最小值检索	FUN		Out:=AryMin(In、Size、InOutPos、Num);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In[ ]数组	检索对象数组	输入	检索对象数组	遵从数据类型	—	(*)
Size	检索对象的元素数		在 In[ ]的元素中，作为检索对象的元素数量			0
InOutPos	检索元素编号	输出	检索值的数值元素编号		—	—
Out	检索结果	输入	检索结果		—	—
Num	检索值的数量	输出	检索值的数量			

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ]数组						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Size							○													
InOutPos							○													
Out						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Out							○													

功能

针对从检索对象数组 In[ ]的 In[0]开始的“Size”个的数组元素进行检索。分别将检索值代入“Out”、检索元素编号代入“InOutPos”、检索值的数量代入“Num”。“Num”大于 1 时，“InOutPos”的值变为检索值中最低位的元素编号。

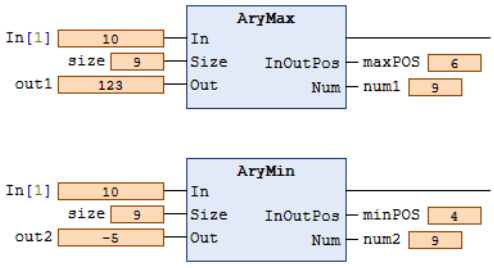
整数、实数以外的数据类型的值的大小关系的判断如下表所示。

指令	名称
TIME	值较大者判断为大。
DATE、TOD、Dt	对于日期和时刻，较后者判断为大。
STRING	依据字符的 ASCII 码大小来判断。

AryMax：检索最大值。

AryMin：检索最小值。

AryMax 指令下，“Size”=UINT#6 时的示例如下所示。发送至 In[ ]的输入参数为 abc[2]，因此 abc[2] 以后变为检索对象。

	FBD	ST
定义变量	VAR In :ARRAY [1..10] OF REAL; size :UINT; out1 :REAL; out2 :REAL; maxPOS :UINT; minPOS :UINT; num1 :UINT; num2 :UINT; END_VAR	
程序		AryMax(In:=In[1] , Size:=size , Out:=out1 , InOutPos=>maxPOS , Num=>num1); AryMin(In:=In[1] , Size:=size , Out:=out2 , InOutPos=>minPOS , Num=>num2);

运行结果

表达式	类型	值	准备值	地址	注释
In	ARRAY [1..10] OF REAL				
In[1]	REAL	10			
In[2]	REAL	23			
In[3]	REAL	12			
In[4]	REAL	3			
In[5]	REAL	-5			
In[6]	REAL	3			
In[7]	REAL	123			
In[8]	REAL	-2			
In[9]	REAL	1			
In[10]	REAL	3			
size	UINT	9			
out1	REAL	123			
out2	REAL	-5			
maxPOS	UINT	6			
minPOS	UINT	4			
num1	UINT	9			
num2	UINT	9			

参考

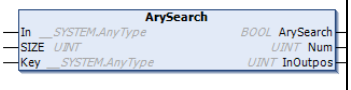
比较 TIME 型、DT 型、TOD 型的大小时，请符合待比较的值的精度。有 “TruncTime 指令”、“TruncDt 指令”、 “TruncTod 指令”，以符合值的精度。

要点说明

- In[ ] 和 “Out” 的数据类型不同时，请将 “Out” 的有效范围设为包含 In[ ] 的有效范围。
- In[ ] 为实数时，因误差的影响，可能无法获取期望的结果。
- 请务必使 In[ ] 为 1 维数组。
- “Size” 的值为 0 时，“Out”、“Num” 的值变为 0。此外，“InOutPos” 的值不变。
- In[ ] 为 STRING 型、“Size” 的值为 0 时，“Out” 仅为 NULL 字符。
- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。“Size” 的值超过有效范围时。“Size” 超出 In[ ] 的数组区域时。In[ ] 非 1 维数组时。In[ ] 为 STRING 型，且未以 NULL 字符结尾时。In[ ] 为 STRING 型且字符数多于 “Out” 的大小时。

6. 6. 2. ArySearch (排列检索)

在 1 维数组中检索指定值。

指令	名称	FB/FUN	图形表现	ST 表现
ArySearch	数组检索	FUN		Out:=ArySearch(In、Size、Key、InOutPos、Num);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In[ ]数组	检索对象数组	输入	检索对象数组	遵从数据类型	—	(*)
Size	检索对象的元素数		在 In[ ]的元素中，作为检索对象的元素数量			0
Key	检索关键词		检索值		—	—
InOutPos	检索元素编号	输出	检索值的数值元素编号		—	—
Out	检索结果		检索结果			
Num	检索值的数量		检索值的数量			

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ]数组	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	也可指定枚举体的数组																			
Size							○													
Key	与 In[ ]的元素相同的数据类型																			
InOutPos							○													
Out	○																			
Num							○													

功能

针对从 1 维检索对象数组 In[ ]的 In[0]开始的“Size”个的数组元素，检索与检索关键词“Key”同值的元素。检索结果 “Out”、检索元素编号 “InOutPos”、检索值的数量 “Num” 的值如下表所示。

与“Key”同值的元素的存在	“Out”	“InOutPos”	“Num”
存在	TRUE	与 “Key” 同值的元素中，最低位 元素的元素编号	与“Key”同值的元素数量
不存在	FALSE	不变	0

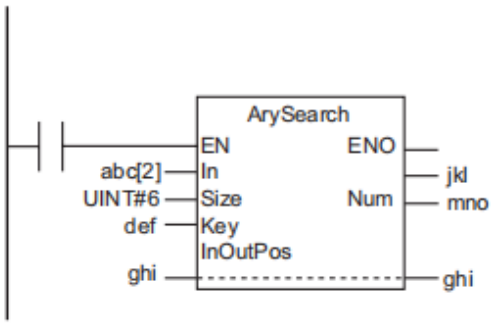
整数、实数以外的数据类型的值的大小关系的判断如下表所示。

指令	名称
TIME	值较大者判断为大。
DATE、TOD、Dt	对于日期和时刻，较后者判断为大。

“Size”=UINT#6 时的示例如下所示。

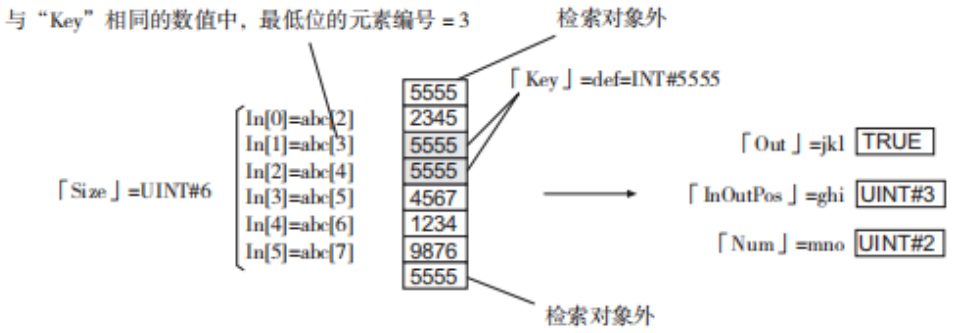
发送至 In[ ] 的输入参数为 abc[2]，因此 abc[2] 以后变为检索对象。

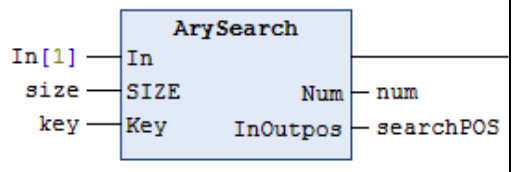
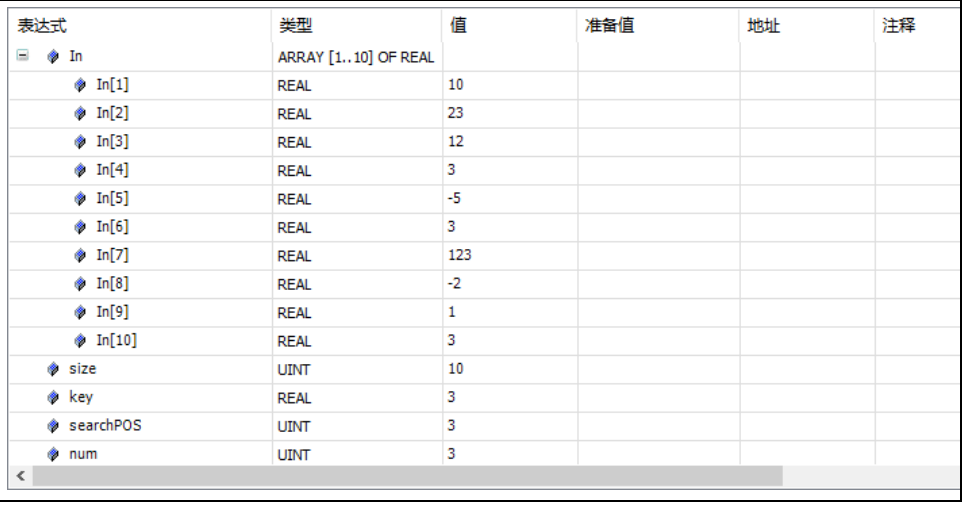
LD



ST

```
jkl:=ArySearch(abc[2], UINT#6, def, ghi, mno);
```



	FBD	ST
定义变量	VAR In :ARRAY [1..10] OF REAL; size :UINT; key :REAL; searchPOS :UINT; num :UINT; END_VAR	
程序		ArySearch(In:=In[1] , SIZE:= size, Key:=key , Num=>num , InOutpos=>searchPOS);
运行结果		

表达式	类型	值	准备值	地址	注释
In	ARRAY [1..10] OF REAL				
In[1]	REAL	10			
In[2]	REAL	23			
In[3]	REAL	12			
In[4]	REAL	3			
In[5]	REAL	-5			
In[6]	REAL	3			
In[7]	REAL	123			
In[8]	REAL	-2			
In[9]	REAL	1			
In[10]	REAL	3			
size	UINT	10			
key	REAL	3			
searchPOS	UINT	3			
num	UINT	3			

参考

比较 TIME 型、DT 型、TOD 型的大小时，请符合待比较的值的精度。有 “TruncTime 指令”、“TruncDt 指令”、 “TruncTod 指令”，以符合值的精度。

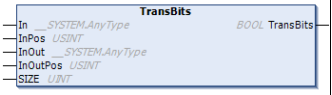
要点说明

- 请务必使 In[ ] 为 1 维数组。
- 请确保 “Key” 与 In[ ] 的元素为相同数据类型。
- “Size” 的值为 0 时，“Out”、“Num” 的值变为 0。此外，“InOutPos” 的值不变。
- 请务必将传输至 “Key” 的输入参数设为变量。如果传输常数，编连时会发生异常。
- “Key” 为枚举体时，无法直接传输枚举元素。如果直接传输枚举元素，编连时会发生异常。
- 以下情况时会发生异常。ENO 变为 FALSE，“Out”、“Num”、“InOutPos” 不变。
- “Size” 超出 In[ ] 的数组区域时。
- In[ ] 或 “Key” 为 STRING 型，且未以 NULL 字符结尾时。
- In[ ] 非 1 维数组时。

6.7. 数据传输指令

6.7.1. TransBits (多位传输)

传送位串内的多位

指令	名称	FB/FUN	图形表现	ST 表现
TransBits	多位传输	FUN		TransBits(In、InPos、InOut、InOutPos、Size);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	传送源	输入	传送源	遵从数据类型	—	(*1)
InPos	传送源的位位置		“In”中的传送位位置	(*2)		0
InOutPos	传送目标的位位置		“Out”中的传送目标位位置	(*3)	—	
Size	传送位数		传送位数	(*4)		1
InOut	传送目标		传送目标	遵从数据类型	—	—
Out	返回值	输出	始终为 TRUE	仅 TRUE	—	—

*1 省略输入参数时，初始值不适用。编连时会发生异常。

*2 0 ～ “In” 的位数 - 1

*3 0 ～ “InOut” 的位数 - 1

*4 0 ～ “In” 的位数

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
InPos						○														
InOutPos						○														
Size						○														
InOut		○	○	○	○															
Out	○																			

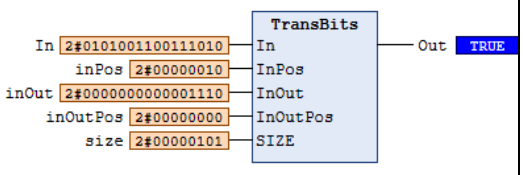
功能

从传送源 “In” 的位位置 “InPos”，将 “Size” 位传送至传送目标 “InOut” 的位位置 “InOutPos”。

“In”=2#0101001100111010，“InPos”=USINT#2、“InOutPos”=USINT#0、“Size”=USINT#5 时的示例如下所示。

	FBD	ST
定义变量		
程序		

```
PROGRAM PLC_PRG
VAR
    check       :BOOL;
    In          :WORD:=2#0101001100111010;
    inPos       :USINT:=2;
    inOutPos    :USINT;
    size        :USINT:=5;
    inOut       :WORD;
    Out         :BOOL;
END_VAR
```




```
IF check FALSE THEN
    Out TRUE :=TransBits(
    In:= In 2#0101001100111010,
    InPos:= inPos 2#00000010,
    InOut:= inOut 2#0000000000001110,
    InOutPos:= inOutPos 2#00000000,
    SIZE:= size 2#00000101 );
    check FALSE :=FALSE;
END_IF RETURN
```

运行结果	表达式 / 类型 / 值 check / BOOL / FALSE In / WORD / 2#0101001100111010 inPos / USINT / 2#00000010 inOutPos / USINT / 2#00000000 size / USINT / 2#00000101 inOut / WORD / 2#0000000000001110 Out / BOOL / TRUE

参考

传送源与传送目标的数据区域允许重叠。

要点说明

- 请勿指定传送源及传送目标的位位置超出 “In” 及 “InOut” 的最高位。此时变为异常，不进行动作。
- “Size” 的值为 0 时，不进行传送。
- “InOut” 内与传送无关的位不变。
- 在 ST 程序中使用本指令时，不使用返回值 “Out”。
- 以下情况时会发生异常。ENO 变为 FALSE，“InOut” 不变。
- “InPos” 的值超过有效范围时。
- “InOutPos” 的值超过有效范围时。
- “Size” 的值超过有效范围时。
- “InPos” 和 “Size” 的指定超过 “In” 的位数时。
- “InOutPos” 和 “Size” 的指定超过 “InOut” 的位数时。

6.7.2. SetBlock (模块设定)

将变量和常数的值传送至多个数组元素。

指令	名称	FB/FUN	图形表现	ST 表现
SetBlocks	块设定	FUN		SetBlock(In、AryOut、Size);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	传送源	输入	传送源	遵从数据类型	—	(*)
Size	传送元素数量		传送数组元素数量			1
AryOut [ ]数组	传送目标数组		传送目标数组	遵从数据类型	—	—
Out	返回值	输出	始终为 TRUE	仅 TRUE	—	—

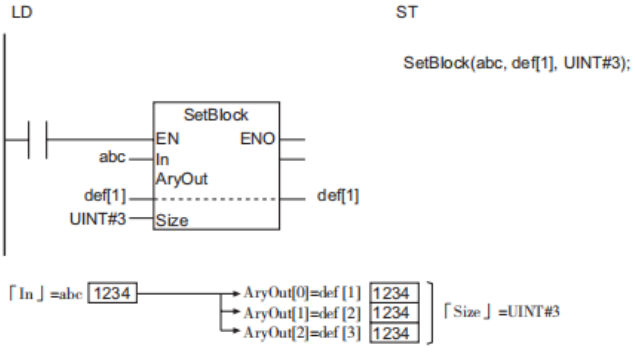
*省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	也可指定枚举体、整个结构体、结构体的 1 个结构要素																			
Size						○														
Aryout[ ]数组	将与“In”相同的数据类型作为元素的数组																			
Out	○																			

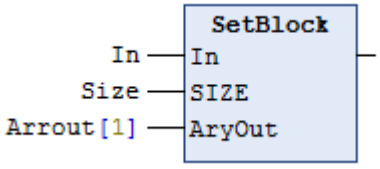
功能

将传送源 “In” 的值传送至从传送目标数组 AryOut[ ] 的 AryOut[0] 开始的 “Size” 个的位置。

“Size”=UINT#3 时的示例如下所示。



	FBD	ST
定义变量		VAR In:REAL; Size:USINT; Arrout:ARRAY [1..10] OF REAL; END_VAR

<table>
<tr><td>程序</td><td> 
</td><td>

```
SetBlock(
    In:= In,
    SIZE:= Size,
    AryOut:=Arrout[1] );
```

</td></tr>
<tr><td>运行结果</td><td>

表达式	类型	值	准备值	地址	注释
In	REAL	4			
Size	USINT	6			
Arrout	ARRAY [1..10] OF REAL				
Arrout[1]	REAL	4			
Arrout[2]	REAL	4			
Arrout[3]	REAL	4			
Arrout[4]	REAL	4			
Arrout[5]	REAL	4			
Arrout[6]	REAL	4			
Arrout[7]	REAL	0			
Arrout[8]	REAL	0			
Arrout[9]	REAL	0			
Arrout[10]	REAL	0			

</td></tr>
</table>

要点说明

- 请将 “In” 和 AryOut[ ] 的数据类型设为相同。如果不同，则编连时会发生异常。
- “In” 和 AryOut[ ] 为 STRING 型时，请使各区域大小保持一致。
- “Size” 的值为 0 时，“Out” 的值为 TRUE，AryOut[ ] 不变。
- 在 ST 程序中使用本指令时，不使用返回值 “Out”。
- 以下情况时会发生异常。ENO 为 FALSE，AryOut[ ] 不变。
- “Size” 的值超过 AryOut[ ] 的数组区域时。

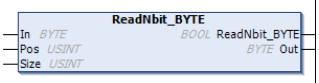
6.7.3. ReadNbit_**** (读取位串内的多位)

ReadNbit_BYTE ：读取 BYTE 数据中的 N 位。

ReadNbit_WORD ：读取 WORD 数据中的 N 位。

ReadNbit_DWORD：读取 DWORD 数据中的 N 位。

ReadNbit_LWORD ：读取 LWORD 数据中的 N 位。

指令	名称	FB/FUN	图形表现	ST 表现
ReadNbit_***	位串读取	FUN		ReadNbit_****(In:= , Pos:= , Size:= , Out=>) ****：为 BYTE、WORD、DWORD、LWORD 中的其中一个。

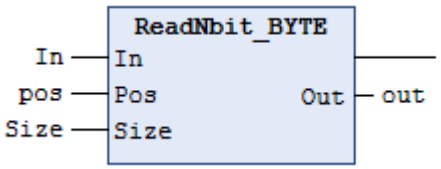
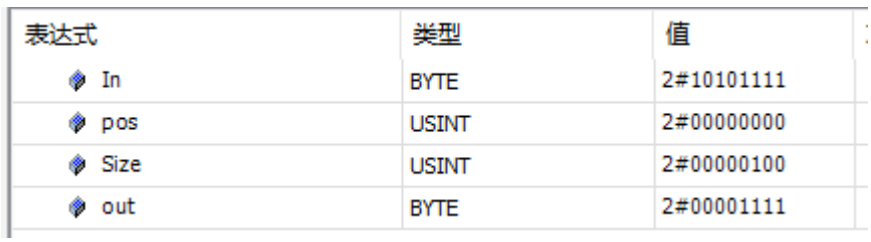
变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	读取源	输入	读取源	遵从数据类型	—	—
Pos	读取起始位		读取起始位			0
Size	读取位数 N		读取位数 N	遵从数据类型	—	1
Out	读取结果	输出	读取的位串结果	遵从数据类型	—	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Pos						○														
Size						○														
Out		○	○	○	○															

功能

将读取源“In”数据从“Pos”开始的“Size”位数据读取拷贝到“Out”中。

	FBD	ST
定义变量	VAR In:BYTE; pos:USINT; Size:USINT; out:BYTE; END_VAR	
程序		ReadNbit_BYTE(In:=In , Pos:=pos , Size:=Size , Out=>out);
运行结果		

表达式	类型	值
In	BYTE	2#10101111
pos	USINT	2#00000000
Size	USINT	2#00000100
out	BYTE	2#00001111

要点说明

- 请将 “In” 和 Out 的数据类型设为相同。如果不同，则编连时会发生异常。
- “Size” 的值为 0 时，“Out” 的值为 TRUE，Out 不变。

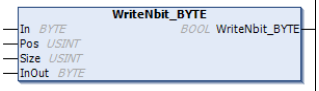
6.7.4. WriteNbit_**** (写位串内的多位)

WriteNbit_BYTE ：在 BYTE 数据中写入 N 位。

WriteNbit_WORD ：在 WORD 数据中写入 N 位。

WriteNbit_DWORD：在 DWORD 数据中写入 N 位。

WriteNbit_LWORD ：在 LWORD 数据中写入 N 位。

指令	名称	FB/FUN	图形表现	ST 表现
WriteNbit_***	位串写入	FUN		WriteNbit_****(In:= , Pos:= , Size:= , Out=>) ****：为 BYTE、WORD、DWORD、LWORD 中的其中一个。

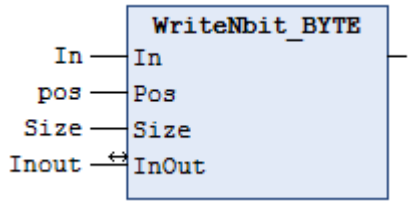
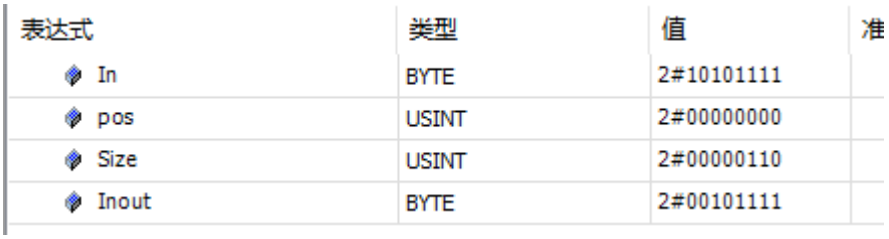
变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	数据源	输入	待拷贝的数据源	遵从数据类型	—	—
Pos	写入起始位		待写入的起始位置			0
Size	写入位数		拷贝位数	遵从数据类型	—	1
InOut	待写入数据	输入输出	待写入的数据	遵从数据类型	—	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Pos						○														
Size						○														
Out		○	○	○	○															

功能

将数据源“In”的从第 0 位数据开始，拷贝“Size”位数据到“InOut”的第“Pos”位到“Pos+Size”位。

	FBD	ST
定义变量	VAR In:BYTE; pos:USINT; Size:USINT; Inout:BYTE; END_VAR	
程序		WriteNbit_BYTE(In:=In , Pos:=pos , Size:=Size , InOut:=Inout);
运行结果		

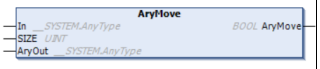
表达式	类型	值	准
In	BYTE	2#10101111	
pos	USINT	2#00000000	
Size	USINT	2#00000110	
Inout	BYTE	2#00101111	

要点说明

- 请将 “In” 和 InOut 的数据类型设为相同。如果不同，则编连时会发生异常。
- “Size” 的值为 0 时，“Out” 的值为 TRUE，Out 不变。
- 数据尺寸不为零 位置不超过数据长度 尺寸与写入的长度不超过数据长度。

6.7.5. AryMove (排列传输)

传送多个数组元素。传送源和传送目标的数据类型可不同。

指令	名称	FB/FUN	图形表现	ST 表现
AryMove	数组的传送	FUN		AryMove(In、AryOut、Size);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In[ ]数组	传输对象数组	输入	传输数组	遵从数据类型	—	(*)
Size	传输元素数量		传送元素数量			1
AryOut[ ]数组	传输结果数组		传送结果数组	遵从数据类型	—	—
Out	返回值	输出	始终为 TRUE	仅 TRUE	—	—

* 省略输入参数时，初始值不适用。编连时会发生异常。

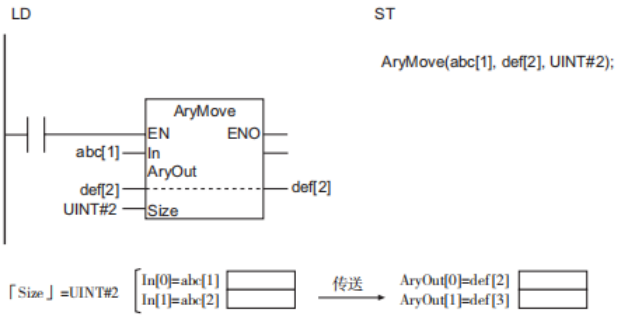
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	也可指定以枚举体为元素的数组、以结构体为元素的数组																			
Size							○													
AryOut	与 In [ ]类型相同的数据																			
Out	○																			

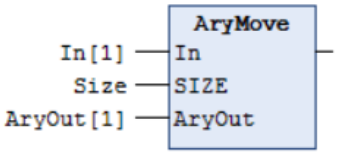
功能

将从传送对象数组 In[ ] 的 In[0] 开始的 “Size” 个的元素传送至从传送结果数组 AryOut[ ] 的 AryOut[0] 开

始的元素。In[ ] 与 AryOut[ ] 的数据类型允许不同。

“Size”=UINT#2 时的示例如下所示。



	FBD	ST
定义变量	VAR In :ARRAY[1..5] OF BOOL ; Size :UINT; AryOut :ARRAY [1..5] OF BOOL; END_VAR	
程序		AryMove(In:=In[1] , SIZE:=Size , AryOut:=AryOut[1]);

运行结果

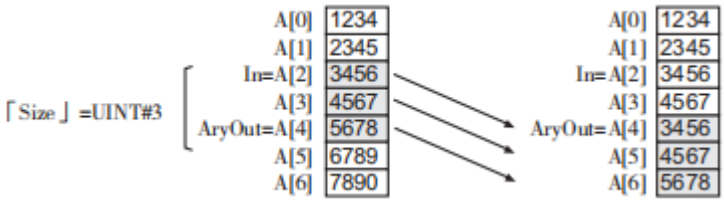
In	ARRAY [1..5] OF BOOL	
In[1]	BOOL	TRUE
In[2]	BOOL	FALSE
In[3]	BOOL	TRUE
In[4]	BOOL	FALSE
In[5]	BOOL	TRUE
Size	UINT	5
AryOut	ARRAY [1..5] OF BOOL	
AryOut[1]	BOOL	TRUE
AryOut[2]	BOOL	FALSE
AryOut[3]	BOOL	TRUE
AryOut[4]	BOOL	FALSE
AryOut[5]	BOOL	TRUE

参考

• In[ ] 和 AryOut[ ] 的数据类型相同时，可高速处理 MemCopy 指令。

• 也可给 In[ ] 和 AryOut[ ] 指定相同的数组。此时，传送源元素与传送目标元素允许重叠。

In[0]=A[2]、AryOut[0]=A[4]、“Size”=UINT#3 时的示例如下所示。



要点说明

- In[ ]和 AryOut[ ]的数据类型不同时，请设为无论下述哪一种数据类型群，AryOut[ ]的有效范围均包含 In[ ] 的有效范围的组合。BYTE、WORD、DWORD、LWORD、USINT、UINT、UDINT、ULINT、SINT、INT、DINT、LINT、REAL、LREAL
- In[ ] 为将结构体作为元素的数组时，请将 In[ ] 和 AryOut[ ] 的数据类型设为相同。
- “Size” 的值为 0 时，“Out” 的值为 TRUE，AryOut[ ] 不变。
- 在 ST 程序中使用本指令时，不使用返回值 “Out”。
- 以下情况时会发生异常。ENO 为 FALSE，AryOut[ ] 不变。
- “Size” 的值超过 In[ ] 或 AryOut[ ] 的大小时。
- In[ ]、AryOut[ ] 为 STRING 型的数组，且任一传送元素均未以 NULL 字符结尾时。
- In[ ]、AryOut[ ] 为 STRING 型的数组，且传送元素的字符串长度超过 AryOut[ ] 元素的大小时。

6.7.6. Clear (初始化)

初始化变量。

指令	名称	FB/FUN	图形表现	ST 表现
Clear	初始化	FUN		Out:=Clear(InOut);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
InOut	初始化对象	输入输出	初始化对象	遵从数据类型	—	—
Out	返回值	输出	始终为 TRUE	仅 TRUE		

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	也可指定以枚举体为元素的数组、以结构体为元素的数组																			
Out	○																			

功能

使初始化对象 “InOut” 的值初始化。

设定变量的初始值属性时，初始化为该值。未设定初始值属性时，变为各数据类型默认的初始值。

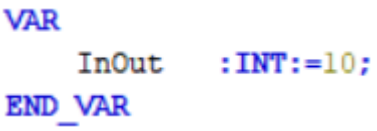
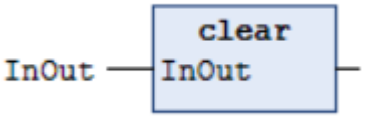
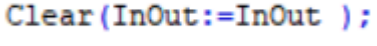
各数据类型默认的初始值如下表所示。

数据类型	默认值
BOOL	FALSE
BYTE、WORD、DWORD、LWORD	16#0
USINT、UINT、UDINT、ULINT、SINT、INT、DINT、LINT、REAL、LREAL	0
TIME	T#0ms
DATE	D#1970-1-1
TOD	TOD#0:0:0
DT	DT#1970-1-1-0:0:0
STRING	“

“InOut” 为整个数组、数组的 1 个元素、整个结构体、结构体的 1 个结构要素时，按下表所示进行处理。

“InOut”	处理内容
整个数组	使数组的所有元素初始化。
数组的 1 个元素	仅使该元素初始化。
整个结构体	使结构体的所有结构要素初始化。
结构体的 1 个结构要素	仅使该结构要素初始化。

示例如下所示。使变量 abc 的值初始化。例如，abc=INT#10 时，abc 的值变为 INT#0。

	FBD	ST
定义变量		
程序		
运行结果		

参考

“InOut” 为作为堆栈使用的数组时，请在执行本指令的同时，将管理堆栈保存数量的变量值清零。

通过本指令使凸轮数据变量初始化时，数值并非为通过 MC_SaveCamTable 指令进行保存的值，而是归零。

要点说明

- 在 ST 程序中使用本指令时，不使用返回值 “Out”。
- 使枚举体的变量初始化时，请设定初始值属性。未设定初始值属性时，枚举体的变量值归零。

6.8. 移位指令

6.8.1. AryShiftReg (移位寄存器)

将数组元素组成的整个位串左移 1 位，并将输入值插入最低位。

指令	名称	FB/FUN	图形表现	ST 表现
AryShiftReg	移位寄存器	FB		AryShiftReg_instance(Shift、Reset、In、InOut、Size、P_CY=>);

变量

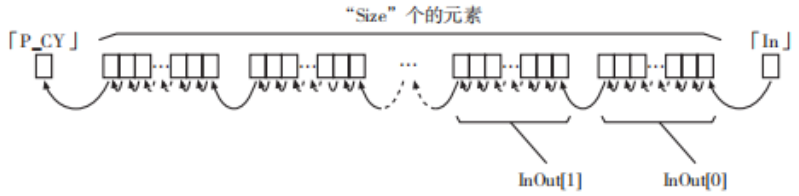
	名称	输入/输出	内容	有效范围	单位	初始值
Shift	移位	输入	FALSE→TRUE 是执行移位	遵从数据类型	—	FALSE
Reset	复位		TRUE：执行复位			
In	输入值		插入 InOut[ ]的最低位的值			
Size	位串数组的元素数量		用于 InOut[ ]内移位寄存器的元素数量		—	1
InOut[ ]数组	位串数组		位串数组	遵从数据类型		—
P_CY	进位标志	输出	进位标准中保存的值	遵从数据类型	—	FALSE

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Shift	○																			
Reset	○																			
In	○																			
Size							○													
InOut[ ]数组	○	○	○	○	○															
P_CY	○																			

功能

“Shift”为 FALSE→TRUE 时，将从位串数组 InOut[ ]的 InOut[0]开始的“Size”个的数组元素向左(高位方向)移 1 位。

将输入值 “In” 插入最低位。将溢出位串数组的最高位输出至进位（CY）标志 “P_CY”。



“Reset” 为 TRUE 时，将 FALSE 设定至从 InOut[0] 开始的 “Size” 个的元素的所有位与进位（CY）标志。

InOut[ ] 为 BYTE 型数组、InOut[0] = 2#10101010，InOut[1] = 2#10101010，“Size” =UINT#2,"In" = TRUE 时触发一次 Shift 的示例如下所示。

InOut[0] = 2#10101010，InOut[1] = 2#10101010 , "P_CY" = FALSE。

→ InOut[0] = 2#01010101，InOut[1] = 2#01010101,"P_CY"=TRUE。

	FBD	ST
定义变量	PROGRAM PLC_PRG VAR Shift :BOOL; Reset :BOOL; In :BOOL; Size :UINT; InOut :ARRAY [0..1] OF BYTE; P_CY :BOOL; AryShiftReg :HCFA_OmronUitls.AryShiftReg; END_VAR	
程序	AryShiftReg AryShiftReg Shift TRUE → Shift Reset FALSE → Reset In TRUE → In InOut[0] 2#01010101 → InOut Size 2#0000000000000010 → Size P_CY ← P_CY TRUE	AryShiftReg(Shift TRUE := Shift TRUE, Reset FALSE := Reset FALSE, In TRUE := In TRUE, InOut:= InOut[0] 2#01010101, Size 2#0000000000000010 := Size 2#0000000000000010 P_CY TRUE => P_CY TRUE);
运行结果	表达式 / 类型 / 值 Shift / BOOL / TRUE Reset / BOOL / FALSE In / BOOL / TRUE Size / UINT / 2#0000000000000010 InOut / ARRAY [0..1] OF BYTE InOut[0] / BYTE / 2#01010101 InOut[1] / BYTE / 2#01010101 P_CY / BOOL / TRUE	

相关的系统定义变量

变量名称	名称	数据类型	内容
P_CY	进位（CY）标志	BOOl	进位标准中保存的值

要点说明

- “Reset” 为 TRUE 时，即使 “Shift” 由 FALSE →TRUE，也不执行移位。
- “Size” 的值为 0 时，InOut[ ] 不变。
- 以下情况时会发生异常。ENO 为 FALSE，InOut[ ] 不变。
- “Size” 的值超过 InOut[ ] 的数组区域时。

6. 8. 2. AryShiftRegLR (左右移位寄存器)

将数组元素组成的整个位串左移 1 位，并将输入值插入最低位。

指令	名称	FB/FUN	图形表现	ST 表现
AryShiftRegLR	左右移位寄存器	FB		AryShiftReg_instance(ShiftL、 ShiftR、 Reset、 In、 InOut、 Size， P_CY=>);

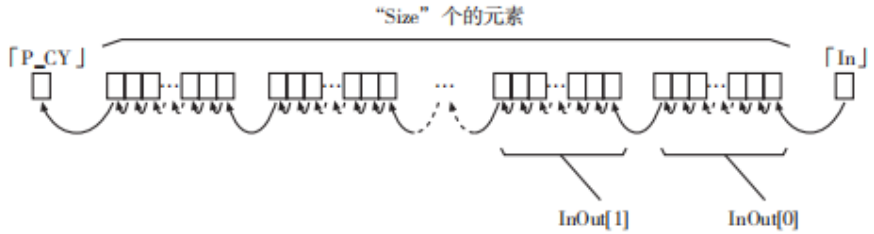
变量

	名称	输入/输出	内容	有效范围	单位	初始值
ShiftL	左移位	输入	FALSE→TRUE 是执行左移位	遵从数据类型	—	FALSE
ShiftR	右移位		FALSE→TRUE 是执行右移位			
Reset	复位		TRUE：执行复位			
In	输入值		插入 InOut[ ]的最低位的值			
Size	位串数组的元素数量		用于 InOut[ ]内移位寄存器的元素数量		—	1
InOut[ ]数组	位串数组		位串数组	遵从数据类型		—
P_CY	进位标志	输出	进位标准中保存的值	遵从数据类型	—	FALSE

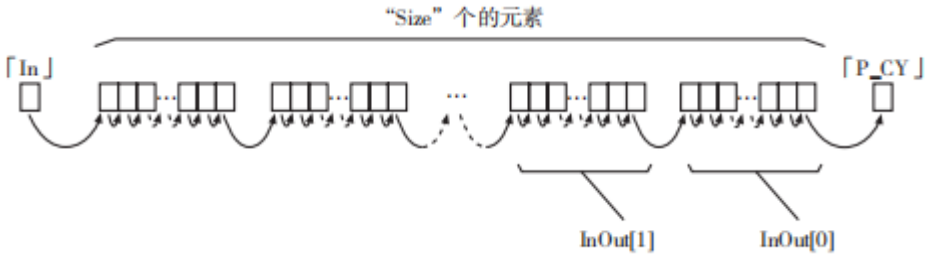
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOI	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
ShiftL	○																			
ShiftR	○																			
Reset	○																			
In	○																			
Size							○													
InOut[ ]数组	○	○	○	○	○															
P_CY	○																			

功能

“ShiftL”由 FALSE→TRUE 时，将从位串数组 InOut[ ]的 InOut[0] 开始的“Size”个的数组元素向左移 1 位。将输入值 “In” 插入最低位。将溢出位串数组的最高位输出至进位 (CY) 标志“P_CY”。



“ShiftR” 由 FALSE TRUE 时，向右移 1 位，将 “In” 插入最高位。将溢出位串数组的最低位输出至进位(CY)标志“P_CY”。



“Reset” 为 TRUE 时，将 FALSE 设定至 InOut[0] 以后的 “Size” 个的元素的所有位与 “P_CY”。

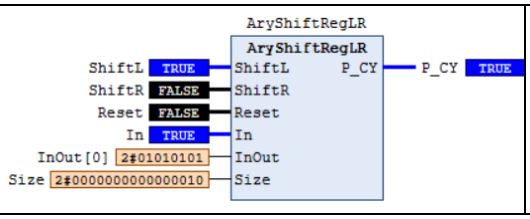
InOut[ ] 为 BYTE 型数组、InOut[0]=2#10101010，InOut[1]=2#10101010，“Size” =UINT#2,"In"=TRUE 时，SfittL 由 FALSE→TRUE 的示例如下所示。

InOut[0] = 2#10101010，InOut[1] = 2#10101010 , "P_CY" = FALSE。

→ InOut[0] = 2#01010101，InOut[1] = 2#01010101,"P_CY"=TRUE。

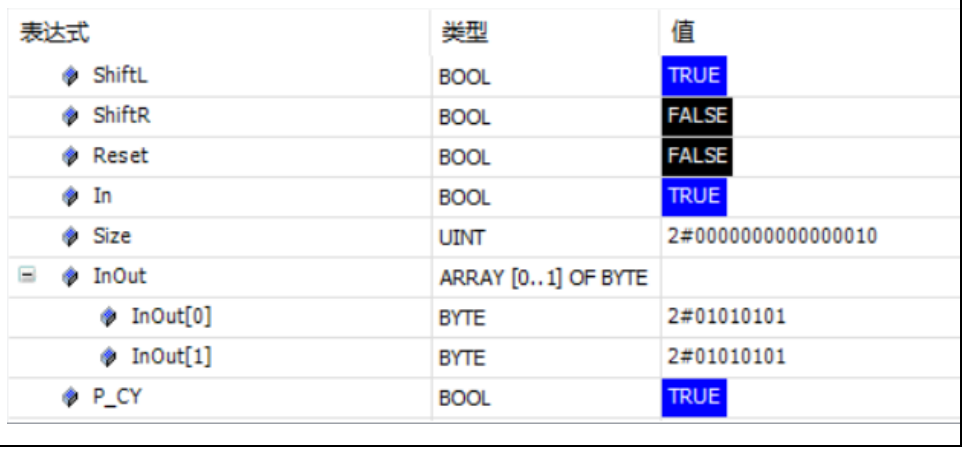
<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">

```
PROGRAM PLC_PRG
VAR
    ShiftL   :BOOL;
    ShiftR   :BOOL;
    Reset    :BOOL;
    In       :BOOL;
    Size     :UINT;
    InOut    :ARRAY [0..1] OF BYTE;
    P_CY     :BOOL;
    AryShiftRegLR   :HCFA_OmronUitls.AryShiftRegLR;
END_VAR
```

</td></tr>
<tr><td>程序</td><td>


</td><td>

```
AryShiftRegLR(
    ShiftL := ShiftL,
    ShiftR := ShiftR,
    Reset := Reset,
    In := In,
    InOut:= InOut[0],
    Size := Size,
    P_CY => P_CY );
```

</td></tr>
<tr><td>运行结果</td><td colspan="2">


</td></tr>
</table>

相关的系统定义变量

变量名称	名称	数据类型	内容
P_CY	进位（CY）标志	BOOI	进位标准中保存的值

要点说明

- “Reset” 为 TRUE 时，即使 “Shift” 由 FALSE →TRUE，也不执行移位。
- “ShiftL” 及 “ShiftR” 同时由 FALSE →TRUE 时，不执行移位。
- “Shift”由 FALSE→TRUE，移位动作正常进行或“Reset”变为 TRUE，复位动作正常进行时，ENO 变为 TRUE。

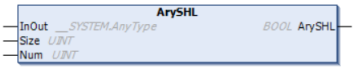
- “Size” 的值为 0 时，InOut[ ] 不变。
- 以下情况时会发生异常。ENO 为 FALSE，InOut[ ] 不变。
- “Size” 的值超过 InOut[ ] 的数组区域时。

6.8.3. ArySHL/ArySHR (排列左/右移位 N 个要素)

对多个数组元素进行移位。

ArySHL：向左（ 高位方向 ）移位。

ArySHR：向右（ 低位方向 ）移位。

指令	名称	FB/FUN	图形表现	ST 表现
ArySHL	数组的 N 元素左移位	FUN		ArySHL(InOut、Size、Num);
ArySHR	数组右移 N 个元素	FUN		ArySHR(InOut、Size、Num);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Size	移位对象的元素数量	输入	移位对象的元素数量	遵从数据类型	—	1
Num	移位元素数量		移位元素数量			
InOut[ ]数组	移位对象数组		移位对象数组			—
Out	返回值	输出	始终为 TRUE	仅为 TRUE		—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							○													
Num							○													
InOut[ ]数组	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	也可将结构体作为元素的数组																			
Out	○																			

功能

针对移位对象数组 InOut[ ] 的高位 “Size” 个的元素，进行移位元素数量 “Num” 个的移位。通过移

位删除溢出值。此外，将 InOut[ ]的数据类型的初始值保存至备用元素。给 InOut[ ]设定初始值属性时，初始化为该值。未设定初始值属性时，变为各数据类型默认的初始值。InOut[ ] 为将结构体作为元素的数组时，使各元素结构体的所有结构要素初始化。

各数据类型默认的初始值如下表所示。

数据类型	默认值
BOOL	FALSE
BYTE、WORD、DWORD、LWORD	16#0
USINT、UINT、UDINT、ULINT、SINT、INT、DINT、LINT、REAL、LREAL	0
TIME	T#0ms
DATE	D#1970-1-1
TOD	TOD#0:0:0
DT	DT#1970-1-1-0:0:0
STRING	“

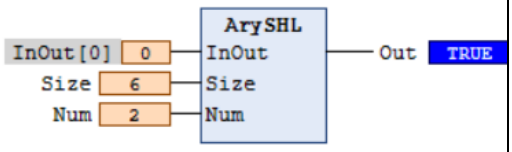
ArySH：向左（ 数组的高位方向 ）移位。

ArySHR：向右（ 数组的低位方向 ）移位。

ArySHL 指令下，“Size”=UINT#6、“Num”=UINT#2 时的示例如下所示。

	FBD	ST
定义变量		
程序		

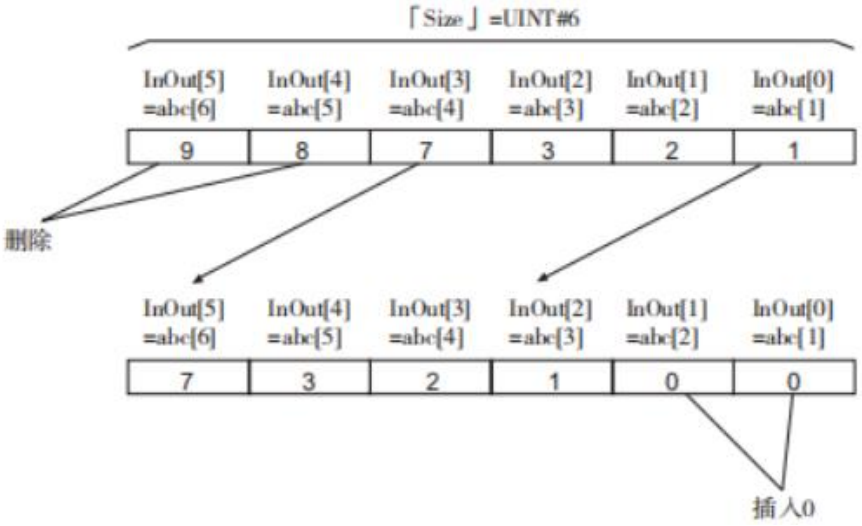
```
PROGRAM PLC_PRG
VAR
    Num     :UINT;
    Size    :UINT;
    InOut   :ARRAY [0..5] OF BYTE;
    Out     :BOOL;
    check   :BOOL;
END_VAR
```




```
IF check FALSE THEN
    out TRUE :=ArySHL(
        InOut:= InOut[0] 0,
        Size:= Size 6,
        Num:= Num 2 );
    check FALSE :=FALSE;
END_IF RETURN
```

运行结果

表达式	类型	值
Num	UINT	2
Size	UINT	6
InOut	ARRAY [0..5] OF BYTE	
InOut[0]	BYTE	0
InOut[1]	BYTE	0
InOut[2]	BYTE	1
InOut[3]	BYTE	2
InOut[4]	BYTE	3
InOut[5]	BYTE	7
Out	BOOL	TRUE



参考

InOut[ ] 为 BOOL 型时，与将“Size”位的位串移“Num”位相同。

要点说明

- “Num” 的值为 0 时，不进行移位动作。
- “Num” 的值大于 “Size” 时，从 InOut[0] 开始，使 InOut[“Size”-1] 的所有值初始化。
- 在 ST 程序中使用本指令时，不使用返回值 “Out”。
- 以下情况时会发生异常。ENO 为 FALSE，InOut[ ] 不变。
- “Size” 的值超过 InOut[ ] 的数组区域时。

6.9. 数据转换指令

6.9.1. Swap (字节交换)

更换 16 位值的高位字节和低位字节。

指令	名称	FB/FUN	图形表现	ST 表现
Swap	字节交换	FUN		Out:=Swap(In);

变量

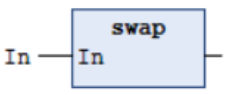
	名称	输入/输出	内容	有效范围	单位	初始值
In	转换对象	输入	转换对象	遵从数据类型	—	0
Out	转换结果	输出	转换结果		—	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Shift			○																	
Reset			○																	

功能

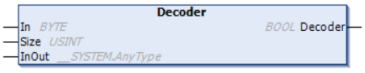
更换转换对象 “In” 的高位字节和低位字节，代入转换结果 “Out”。

“In”=WORD#16#1234 时的示例如下所示。

	FBD	ST
定义变量	VAR In :WORD:=16#1234; Out :WORD; END_VAR	
程序		Out:= SWAP(In:= In);
运行结果	In WORD 16#1234 Out WORD 16#3412	

6.9.2. Decoder (位解码器)

将最大 256 位组成的数组元素指定的 1 位设置为 TRUE，其它位设置为 FALSE。

指令	名称	FB/FUN	图形表现	ST 表现
Decoder	位译码器	FUN		Decoder(In, Size, InOut);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	转换位位置	输入	待转换的位位置	遵从数据类型	—	0
Size	转换位数		待转换的位数	0~8	位	1
InOut[ ]数组	转换对象数组		转换对象数组	遵从数据类型	—	—
Out	返回值	输出	始终为 TRUE	仅 TRUE	—	—

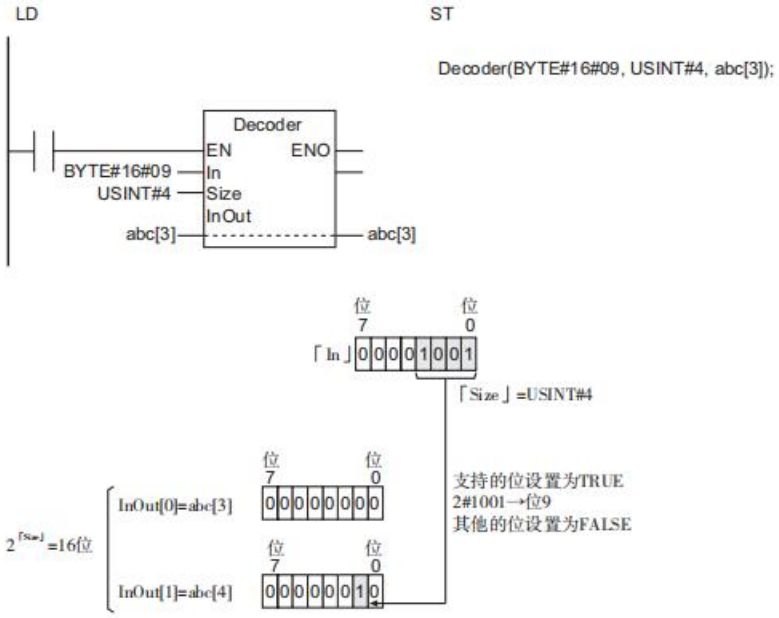
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○																		
Size						○														
InOut[ ]数组	○	○	○	○	○															
Out	○																			

功能

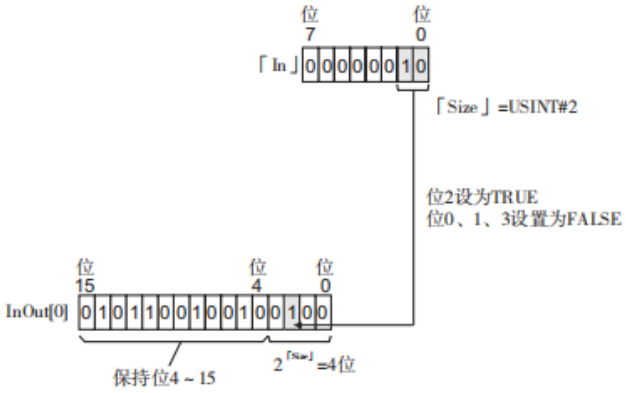
将转换对象数组 InOut[ ]从 InOut[0]开始 2“Size”位大小的数组元素的指定位设为 TRUE。其他位设为 FALSE。值为 TRUE 的位位置由转换位位置“In”的低位“Size”位指定。

传输至 InOut[ ] 的输入输出参数务必也像 array[3] 那样加上元素编号后再指定。

以 “In”=BYTE#16#09，“Size”=USINT#4，InOut[ ] 为 BYTE 型数组为例进行说明。“In”的低位“Size”位的值 16#9 转换为 10 进制后为 9。因此，从 InOut[ ] 的最低位起的第 9 位为 TRUE，其他位为 FALSE。InOut[ ] 是 BYTE 型数组，从最低位起的第 9 位是 InOut[1] 的位 1。因此，InOut[1] 的位 1 为 TRUE，InOut[0]的所有位和 InOut[1] 的位 1 以外的位为 FALSE。



InOut[ ] 的元素的位数大于以 “Size” 表示的位数时，保持剩余位的值。以 “In” =BYTE#16#02，“Size” =USINT#2，InOut[ ]为 WORD 型数组为例进行说明。“Size” =USINT#2，因此 InOut[0] 的低位 4 位设置值。保持 InOut[0] 的剩余位 4 ～ 15 的值。



	FBD	ST
定义变量	VAR In :BYTE; Size :USINT; InOut :ARRAY[1..5] OF BYTE; END_VAR	
程序	Decoder In — In Size — Size InOut[1] — InOut	Decoder(In:=In , Size:=Size , InOut:=InOut[1]);

运行结果	In	BYTE	2#00001001
	Size	USINT	2#00000110
	InOut	ARRAY [1..5] OF BYTE	
	InOut[1]	BYTE	2#00000000
	InOut[2]	BYTE	2#00000010
	InOut[3]	BYTE	2#00000000
	InOut[4]	BYTE	2#00000000
	InOut[5]	BYTE	2#00000000

参考

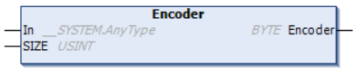
在最多由 256 位组成的数组元素中，计算 TRUE 的位位置时，请使用 “Encoder 指令 (P.2-394)”。

要点说明

- “Size” 的值为 0 时，InOut[ ] 的所有位为 FALSE。
- 在 ST 程序中使用本指令时，不使用返回值 “Out”。
- 以下情况时会发生异常。ENO 为 FALSE，InOut[ ] 不变。
- “Size” 超过有效范围时。
- 2 “Size” 的值超过 InOut[ ] 的数组元素的位数时。
- InOut[ ] 不是 BOOL 或位串数据类型的数组时。
- 向 InOut[ ] 传输了无下标的数组时。

6.9.3. Encoder (位编码器)

计算最大 256 位组成的数组元素中，值为 TRUE 的位位置。

指令	名称	FB/FUN	图形表现	ST 表现
Encoder	位编码器	FUN		Out:=Encoder(In, Size);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In[ ]数组	转换对象数组	输入	转换对象数组	遵从数据类型	—	(*)
Size	检查位数		待检查的位数	0~8	位	1
Out	检查结果	输出	检查结果	遵从数据类型	—	—

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ]数组	○	○	○	○	○															
Size						○														
Out		○																		

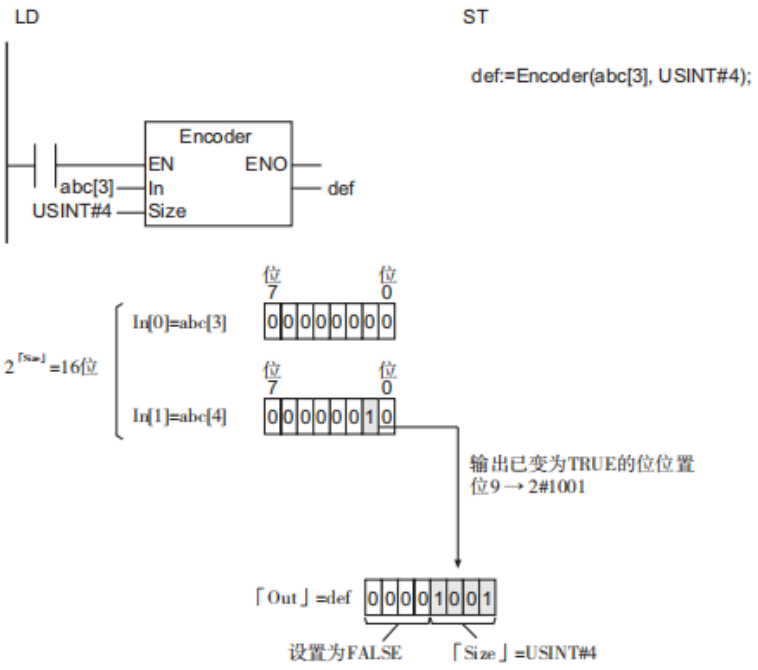
功能

计算转换对象数组 In[ ] 在任意范围中值为 TRUE 的位位置。在以 In[0] 开始的 2“Size” 位的范围中计算位位置。以 2 进制表示此范围中值为 TRUE 的位位置，在转换结果 “Out” 的低位 “Size” 位中保存。“Out” 的剩余位中保存 FALSE。

如果指定范围内有多个值为 TRUE 的位，则在这些位中计算最高位的位位置。传输至 In[ ] 的输入参数务必也像 array[3] 那样加上元素编号后再指定。表示 “Size”=USINT#4，In[ ] 为 BYTE 型数组时的示例。

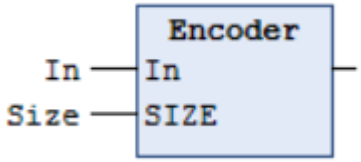
“Size”=USINT#4，因此从 In[0] 开始的 24 =16 位是计算 TRUE 的位位置的对象范围。下图中，该范围的第 9 位有值为 TRUE 的位。“Size”=USINT#4，因此在“Out”的低位 4 位中保存已计算的 9=2#1001。

“Out”的高位 4 位中保存 FALSE。



<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">

```
VAR
    In      :ARRAY [1..5] OF BYTE;
    Size    :USINT;
    Out     :BYTE;
END_VAR
```

</td></tr>
<tr><td>程序</td><td> 
</td><td>

```
Out:=Encoder(In:=In[1] ,
             SIZE:=Size );
```

</td></tr>
<tr><td>运行结果</td><td colspan="2">
<table>
<tr><td>In</td><td>ARRAY [1..5] OF BYTE</td><td></td></tr>
<tr><td>In[1]</td><td>BYTE</td><td>2#00000000</td></tr>
<tr><td>In[2]</td><td>BYTE</td><td>2#00000000</td></tr>
<tr><td>In[3]</td><td>BYTE</td><td>2#00000101</td></tr>
<tr><td>In[4]</td><td>BYTE</td><td>2#00000001</td></tr>
<tr><td>In[5]</td><td>BYTE</td><td>2#00000000</td></tr>
<tr><td>Size</td><td>USINT</td><td>2#00000101</td></tr>
<tr><td>Out</td><td>BYTE</td><td>2#00011000</td></tr>
</table>
</td></tr>
</table>

参考

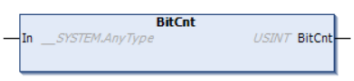
在最多由 256 位组成的数组元素中，将 1 位设置为 TRUE，其他位设置为 FALSE 时，请使用 “Decoder 指令”。

要点说明

- “Size” 的值为 0 时，“Out” 的所有位为 FALSE。
- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- “Size” 超过有效范围时。
- 2“Size” 的值超过 In[] 的数组元素的位数时。
- In[] 的位中，“Size” 指定的所有的值为 FALSE 时。
- In[] 不是 BOOL 或位串数据类型的数组时。
- 向 In[] 传输了无下标的数组时。

6.9.4. BitCnt (位计数器)

对位串内的值为 TRUE 的位的总数进行计数。

指令	名称	FB/FUN	图形表现	ST 表现
BitCnt	位计数	FUN		Out:=BitCnt(In);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	计数对象	输入	对值为 TRUE 的位进行计数的对象	遵从数据类型	—	(*)
Out	计数结果	输出	值为 TRUE 的位数	0~“In”的位数	—	—

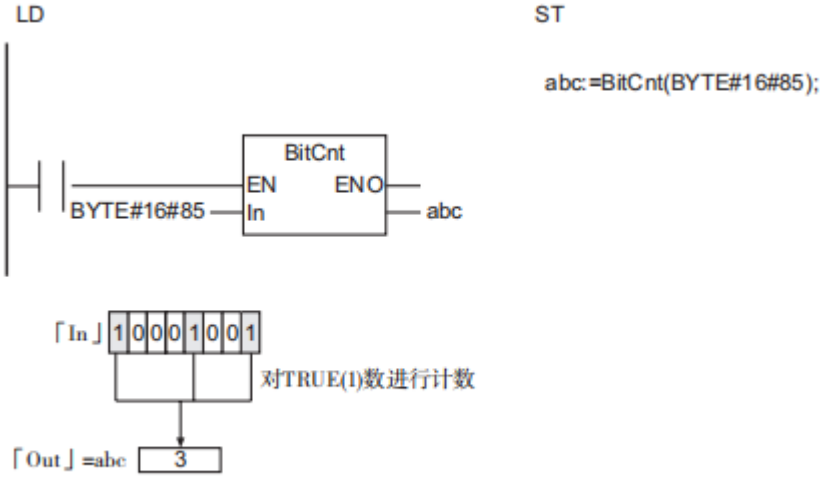
* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Out						○														

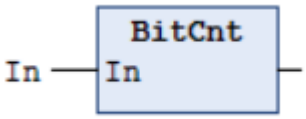
功能

对计数对象 “In” 中值为 TRUE 的位的总数进行计数。

“In” 为 BYTE 型，值为 BYTE#16#85 时的示例如下所示。



	FBD	ST
定义变量	VAR In :BYTE; Out :USINT; END_VAR	

程序		Out:=BitCnt(In:=In);
运行结果		

6.9.5. LineToColm (位行 TO 位列转换)

分解位串，输出至数组元素指定的位位置。

指令	名称	FB/FUN	图形表现	ST 表现
LineToColm	位行→位串转换	FUN		LineToColm(In, InOut, Size, Pos);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	转换对象	输入	转换对象	遵从数据类型	—	(*)
Size	转换结果的元素数		转换结果的元素数	0~“In”的位数		1
Pos	转换位位置		待转换的位位置	0~InOut[ ]的位数-1		0
InOut[ ]数组	转换结果数组		转换结果	遵从数据类型	—	—
Out	返回值	输出	始终为 TRUE	仅 TRUE	—	—

* 省略输入参数时，初始值不适用。编连时会发生异常。

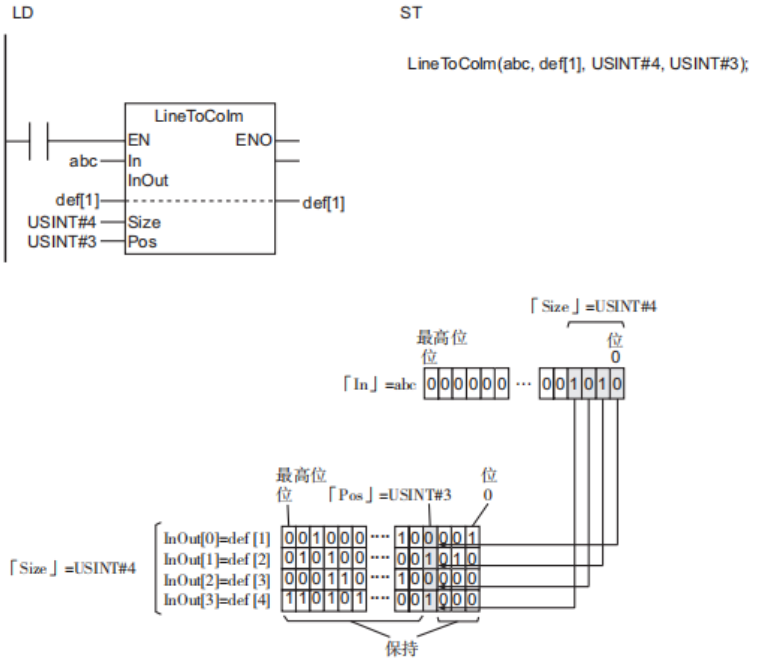
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
Size						○														
Pos						○														
InOut[ ]数组		○	○	○	○															
Out	○																			

功能

分解位串，输出至数组元素指定的位位置。

首先，从转换对象 “In” 的最低位提取 “Size” 位，并分解成每个位。然后，将其保存在转换结果数组 InOut[ ] 从 InOut[0]开始的各元素的第 “Pos” 位中。保存的数组元素有“Size” 个。未保存值的位，进行值的保持。

“Pos”=USINT#3、“Size”=USINT#4 时的示例如下所示。



	FBD	ST
定义变量	VAR In :BYTE; Size :USINT; Pos :USINT; InOut :ARRAY[1..5] OF BYTE; END_VAR	
程序	LineToColm In — In Size — Size Pos — Pos InOut[1] — InOut	LineToColm(In:=In , Size:=Size , Pos:=Pos , InOut:=InOut[1]);
运行结果	In BYTE 2#00011 Size USINT 2#00000 Pos USINT 2#00000 InOut ARRAY [1..5] OF BYTE InOut[1] BYTE 2#00100 InOut[2] BYTE 2#00100 InOut[3] BYTE 2#00100 InOut[4] BYTE 2#00100 InOut[5] BYTE 2#00100	

要点说明

- “Size” 的值为 0 时，“Out” 的所有位为 FALSE。
- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- “Size” 超过有效范围时。
- “Pos” 的值超过有效范围时。
- “Size” 的值超过 InOut[ ] 的数组区域时。
- InOut[ ] 的不是位串数据类型的数组时。
- 向 InOut[ ] 传输了无下标的数组时。

6.9.6. Gray (格雷码转换)

将格雷码转换为角度。

指令	名称	FB/FUN	图形表现	ST 表现
Gray	格雷码转换	FUN		Out:=Gray(In, Resolution, ERC, ZPC);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	转换对象	输入	转换对象的格雷码	遵从数据类型	—	0
Resolution	分辨率		分辨率	枚举体 _eGRY_RESOLUTION		_R256
ERC	编码器余数补偿值		编码器余数补偿值	0~“Resolution”的分辨率		0
ZPC	原点补偿值		原点补偿值			
Out	转换结果	输出	转换结果	(*)	°	—

* 0 ～ 3.599999999999999e+2。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In			○																	
Resolution	枚举体_eGRY_RESOLUTION																			
ERC							○													
ZPC							○													
Out															○					

功能

将格雷码表现的旋转编码器的输出值 “In” 转换为角度值。转换结果 “Out” 的单位为 °。

“Resolution” 的数据类型为枚举体 _eGRY_RESOLUTION。枚举元素的含义如下所示。

枚举元素	含义
_R256	256
_R1B	1 位（2）
_R2B	2 位（4）
_R3B	3 位（8）
_R4B	4 位（16）
_R5B	5 位（32）
_R6B	6 位（64）
_R7B	7 位（128）
_R8B	8 位（256）
_R9B	9 位（512）
_R10B	10 位（1024）
_R11B	10 位（2048）
_R12B	12 位（4096）
_R13B	13 位（28192）
_R14B	14 位（16384）
_R15B	15 位（32768）
_R360	360
_R720	720
_R1024	1024

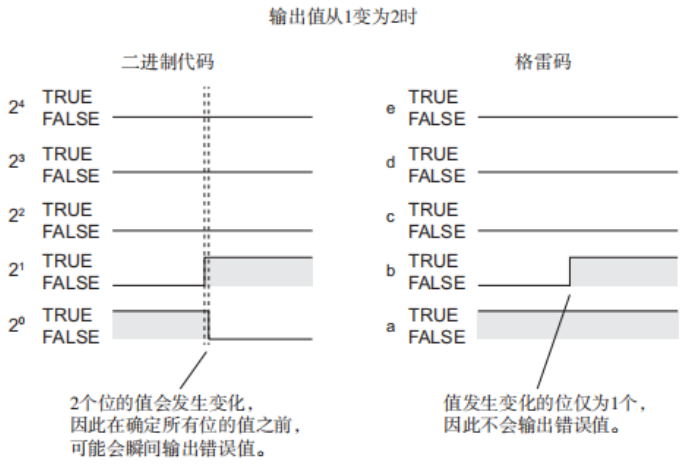
格雷码

格雷码是一种名为交替二进制代码的 2 进制数。其特征在于，像 0 和 1、1 和 2 那样，差为 1 的数值的代码必定有 1 个位的值不同。格雷码用于绝对值编码器的输出等用途。

以下列出了 4 位的二进制代码和格雷码。

10 进制	二进制代码				格雷码			
	2^3	2^2	2^1	2^0	d	c	b	a
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

如果使用格雷码，编码器的输出值增加或减少 1 时，仅有 1 个位的值发生变化，因此可防止瞬间输出错误的值。因格雷码和二进制代码的差异导致的编码器输出值变化的差异如下所示。



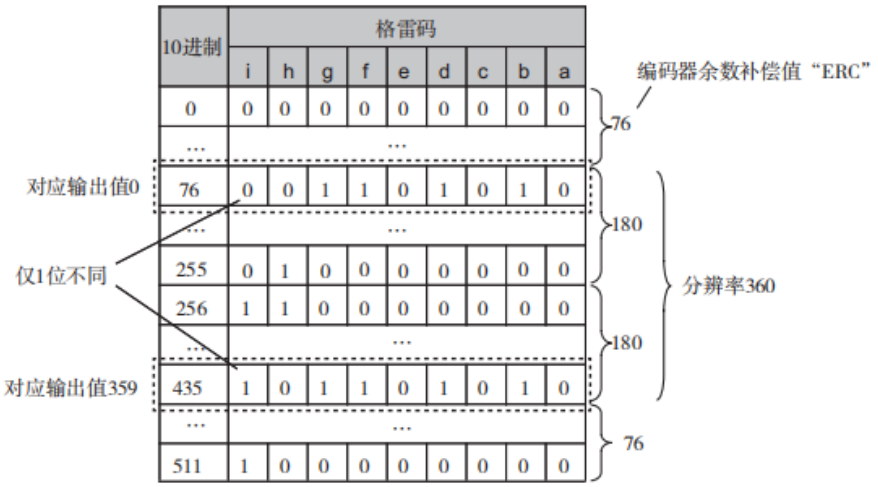
	FBD	ST
定义变量	VAR In :WORD; Resolution :_eGRY_RESOLUTION; ERC :UINT; ZPC :UINT; Out :LREAL; END_VAR	
程序	Gray In — In Resolution — Resolution ERC — ERC ZPC — ZPC	Out:=Gray(In:=In , Resolution:=Resolution ERC:=ERC, ZPC:=ZPC);
运行结果	In WORD 16#01A9 Resolution _EGRY_RESOLUTION _R10B ERC UINT 16#0000 ZPC UINT 16#0151 Out LREAL 348.75	

编码器余数补偿值“ERC”

ERC” 为指定格雷码范围的补偿值，编码器的分辨率不是 2 的乘幂时，以使相对于编码器输出最大值和最小值的格雷码的差异仅有 1 位。

例如，使用分辨率为 360 的绝对值编码器。使用 9 位格雷码。9 位可表现的范围为 0 ～ 511。此时，自 0～511 的中心向前后各 180 的范围，即在 76 ～ 435 的范围内使用格雷码。因此，表示输出值为 0

时的格雷码为 001101010(10 进制为 76)，表示输出值为 359 时的格雷码为 101101010(10 进制为 435)，两者相差仅 1 位。此时，编码器余数补偿值“ERC”的值为 76。

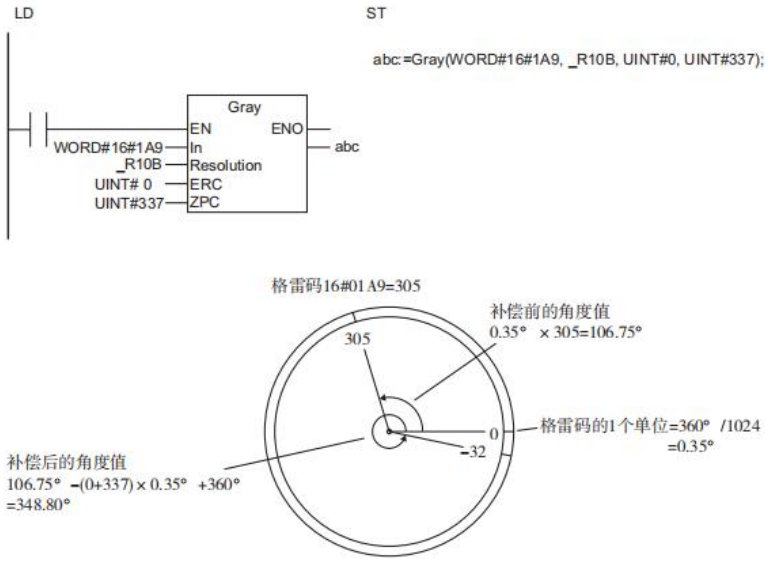


原点补偿值“ZPC”

移动旋转编码器的原点角度时，需设定 “ZPC”。例如，将分辨率为 256 的旋转编码器的原点移动 90° 时，“ZPC” 的值设为 256×(90/360)=64。

记述示例

“In”=WORD#16#1A9，“Resolution”=_R10B，“ERC”=UINT#0，“ZPC”=UINT#337 时的示例如下所示。首先，分辨率为 10 位，因此格雷码的 1 个单位为 360°/1024=0.35°。格雷码 16#01A9 相应的 10 进制数的值为 305。因此，补偿前的角度值为 0.35°×305=106.75°。“ERC”的值为 0，“ZPC”的值为 337。因此，对其补偿后的角度值为 106.75°-(0+337)×0.35°=-11.20°。“Out” 的值的范围为大于 0，因此为 -11.20°+360°=348.80°。“Out” 的值为 LREAL#348.8。

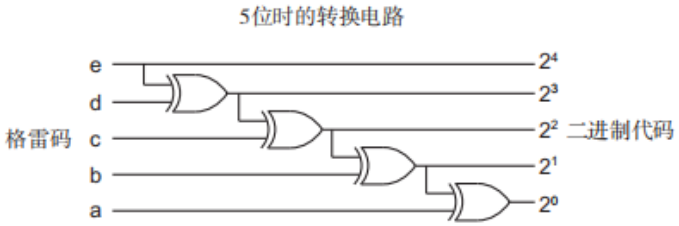


参考

请参阅所使用的旋转编码器的使用说明书等后再确定为 “Resolution” 和 “ERC” 指定的值。

从格雷码到二进制代码的转换

可通过以下处理进行从格雷码到二进制代码的转换。图中的逻辑符号表示异或。

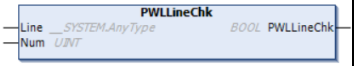


使用注意事项

- 以下情况时会发生异常。“ENO” 变为 FALSE，“Out” 不变。
- “Resolution” 的值超过有效范围时。
- “ERC” 的值大于 “Resolution” 指定的分辨率时。
- “ZPC” 的值大于 “Resolution” 指定的分辨率时。
- 将 “In” 转换为位串后的值小于 “ERC” 的值时。
- 通过 “ERC” 补偿位串后的值大于 “Resolution” 指定的分辨率时。

6.9.7. PWLLineChk (折线数据检查)

判定折线近似转换（无折线数据检查）指令使用的折线数据是否按 X 坐标升序排列。

指令	名称	FB/FUN	图形表现	ST 表现
PWLLineChk	折线数据检查	FUN		Out:=PWLLineChk(Line,Num);

变量

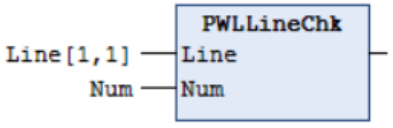
	名称	输入/输出	内容	有效范围	单位	初始值
Line[ ]数组	折线数据数组	输入	折线数据数组	遵从数据类型	—	(*)
Num	折线数据数		折线数据数			1
Out	判定结果	输出	判定结果	遵从数据类型	—	—

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○	○	○					
Size							○													
Out	○																			

功能

判定折线近似转换（无折线数据检查）PWLApproxNoLineChk 指令使用的折线数据数组 Line[ ] 的元素是否按 X 坐标升序排列。如果按升序排列，判定结果 “Out” 的值为 TRUE，否则 “Out” 的值为 FALSE。

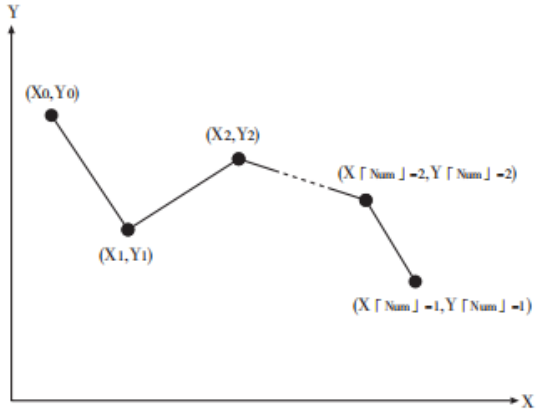
	FBD	ST
定义变量	VAR Line :ARRAY[1..5,1..2] OF REAL; Num :UINT:=5; Out :BOOL; END_VAR	
程序		Out:=PWLLineChk(Line:=Line[1,1] , Num:=Num);

运行结果

Line	ARRAY [1..2, 1..5] ...	
Line[1, 1]	REAL	1
Line[1, 2]	REAL	1.1
Line[1, 3]	REAL	2
Line[1, 4]	REAL	-1.1
Line[1, 5]	REAL	3
Line[2, 1]	REAL	2
Line[2, 2]	REAL	4
Line[2, 3]	REAL	0
Line[2, 4]	REAL	5
Line[2, 5]	REAL	-3
Num	UINT	5
Out	BOOL	TRUE

折线数据 Line[ ]的元素和折线数据“Num”

Line[ ] 为 2 维，请将第 1 维的元素数设为 2。如图所示，请将折线数据的各点坐标值（X0 ,Y0)、(X1 ,Y1)……作为 Line[ ] 的各元素。折线数据数 “Num” 为用于折线近似计算的 Line[ ] 的元素数的 1/2。



Line[ ]为2维数组时

Line[0,0]	X0
Line[0,1]	Y0
Line[1,0]	X1
Line[1,1]	Y1
Line[2,0]	X2
Line[2,1]	Y2
:	:
Line[⌈Num⌋-1,0]	X⌈Num⌋-1
Line[⌈Num⌋-1,1]	Y⌈Num⌋-1

Line[ ]为3维时

Line[0,0,0]	X0
Line[0,0,1]	Y0
Line[0,1,0]	X1
Line[0,1,1]	Y1
Line[0,2,0]	X2
Line[0,2,1]	Y2
:	:
Line[0,⌈Num⌋-1,0]	X⌈Num⌋-1
Line[0,⌈Num⌋-1,1]	Y⌈Num⌋-1

记述示例

判定元素数为 4 的折线数据数组 abc[ ] 是否按 X 坐标升序排列的示例如下所示。“Num”=UINT#4，abc[ ]的元素值如下所示。

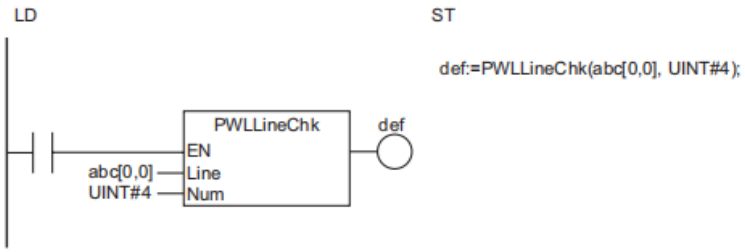
abc[0.0]=X0 =LREAL#1.0、abc[0,1]=Y0 =LREAL#5.0、

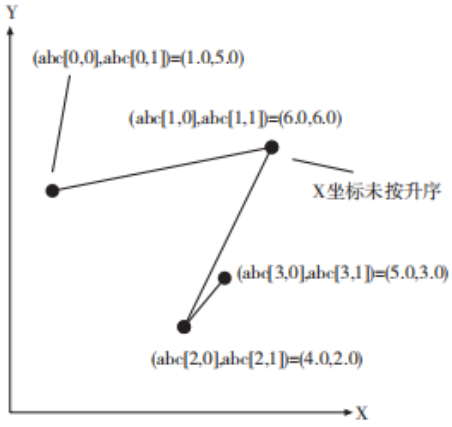
abc[1.0]=X1 =LREAL#6.0、abc[1,1]=Y1 =LREAL#6.0、

abc[2.0]=X2 =LREAL#4.0、abc[2,1]=Y2 =LREAL#2.0、

abc[3.0]=X3 =LREAL#5.0、abc[3,1]=Y3 =LREAL#3.0

X 坐标未按升序排列，因此 “Out” 的值为 FALSE。



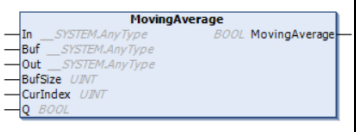


要点说明

- Line[ ] 为 2 维，请将第 1 维的元素数设为 2。
- 以下情况时会发生异常。“Out” 为 FALSE。
- “Num” 的值超过 Line[ ] 的数组区域时。
- Line[ ] 为实数型，且元素的值为非数、正无穷大、负无穷大时。
- Line[ ] 不是支持的数据类型时。

6. 9. 8. MovingAverage (移动平均)

计算出移动平均值

指令	名称	FB/FUN	图形表现	ST 表现
Moving Average	移动平均	FUN		MovingAverage(In, Buf, Out, BufSize, CurIndex, Q);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	输入值	输入	用于平均值计算的数值	遵从数据类型	—	(*)
BufSize	最大元素数		用于平均值计算的最大元素数			1
CurIndex	输入值保存位置	输入输出	保存“In”的 Buf[ ]的位置	遵从数据类型	—	—
Buf[ ]数组	输入值保存数组	输入	保存“In”的数组			
Q	计算完成标志	输入输出	TRUE ：保存至 Buf [ ] 的数值数大于“BufSize” FALSE：保存至 Buf [ ] 的数值数为小于“BufSize”			
Out	运算结果	输入	运算结果	遵从数据类型	—	—

* 省略输入参数时，初始值不适用。编连时会发生异常。

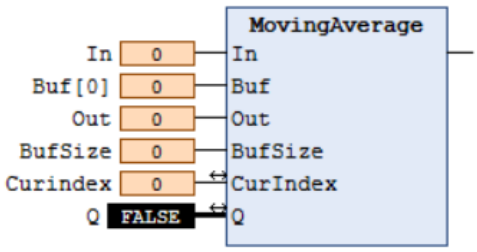
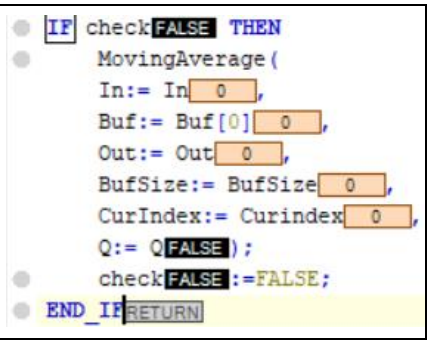
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○	○	○					
BufSize							○													
CurIndex							○													
Buf[ ]数组	将与“In”相同的数据类型作为元素的数组																			
Q	○																			
Out						○	○	○	○	○	○	○	○	○	○					

功能

每次执行本指令时，均将输入值 “In” 保存在输入值保存数组 Buf[ ] 中。再将已保存的值的平均值保存在运算结果 “Out”中。用于平均值计算的最大元素数由 “BufSize” 指定。

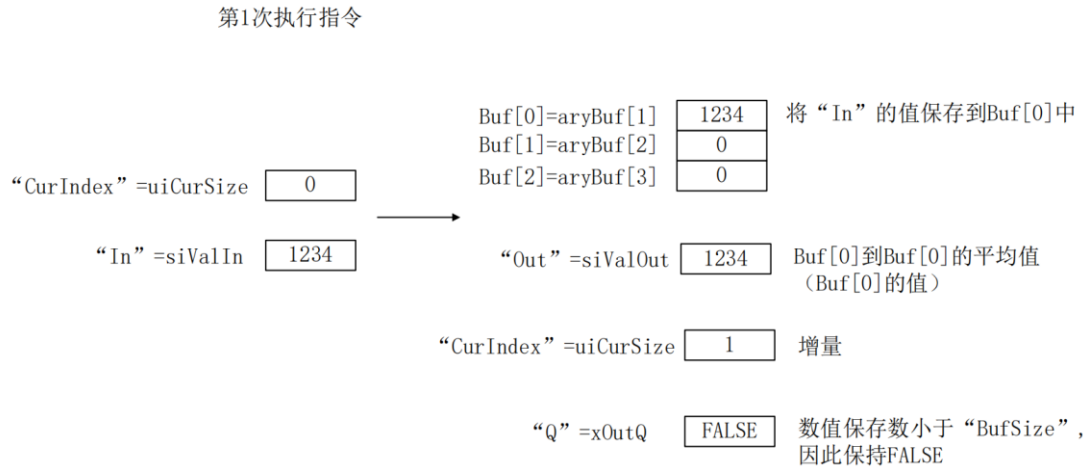
	FBD	ST
定义变量		

```
PROGRAM PLC_PRG
VAR
    In      :UINT;
    BufSize :UINT;
    Curindex:UINT;
    Buf     :ARRAY [0..3] OF UINT;
    Q       :BOOL;
    Out     :UINT;
    check   :BOOL;
END_VAR
```

程序		

初始值输入

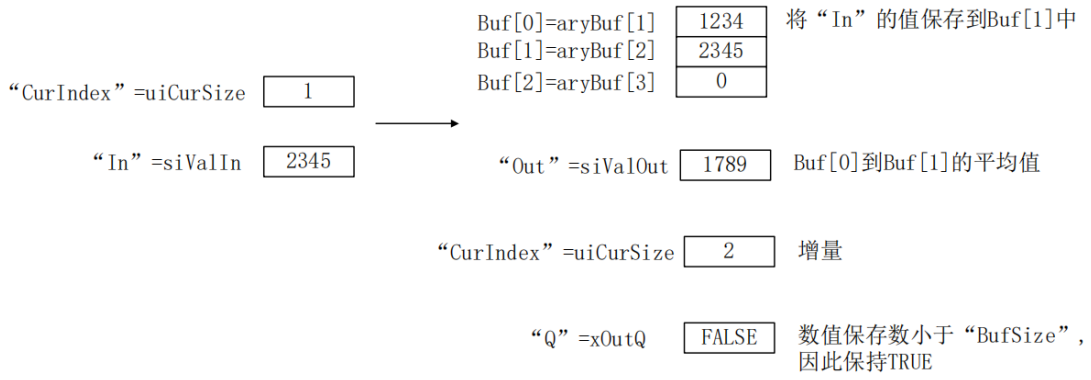
将输入值保存位置“CurIndex”设定为 0，然后执行本指令。将输入值保存排列 Buf[] 的 Buf[0] 开始到 Buf[“BufSize”-1] 清零后，初始输入值“In”的值将保存到 Buf[0] 中。计算结束标志“Q”的值为 FALSE。这表示保存到 Buf[] 的数值数还未达到“BufSize”。“Q”的值为 FALSE 时，计算平均值时使用 Buf[0] 开始到“CurIndex”+1 个的数值。运算结果保存到“Out”中。然后，“CurIndex”的值增加。



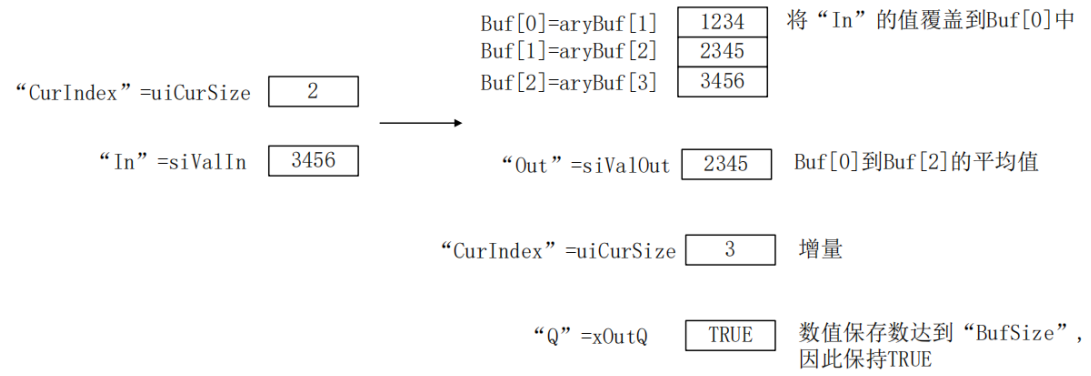
第“BufSize”次为止的数值输入

每次执行本指令，“In”的值都将保存到 Buf[“CurIndex”] 中。计算平均值时使用 Buf[0] 开始到“CurIndex”+1 个的数值，运算结果保存到“Out”中。执行次数达到“BufSize”后，“Q”的值变为 TRUE。

第2次执行指令



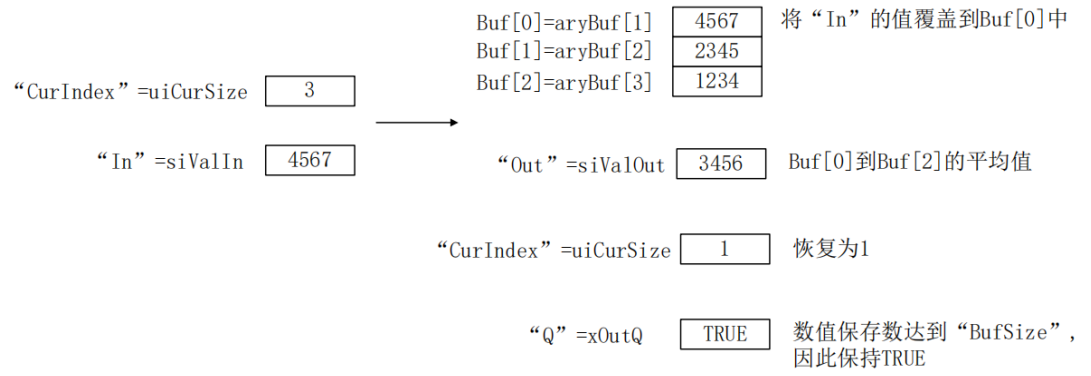
第3次执行指令



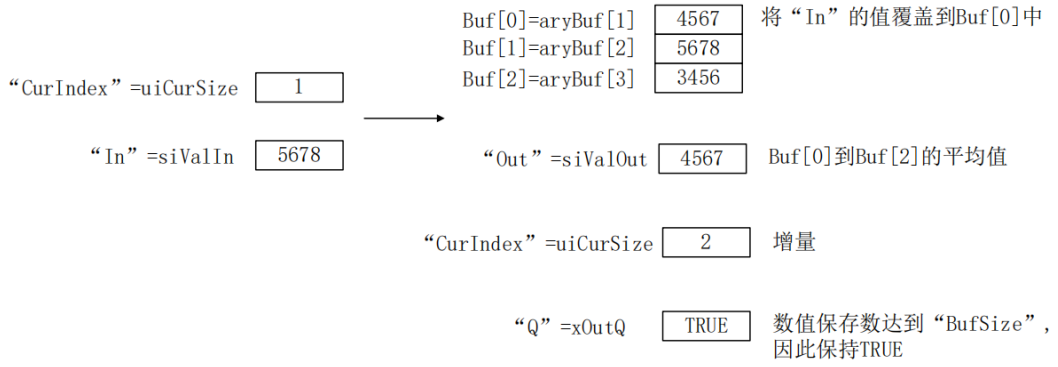
第“BufSize”次后的数值输入

每次执行本指令，“In”的值都将在 Buf[0] 到 Buf[“BufSize”-1] 之间周期性覆盖。计算平均值时使用当时 Buf[0]到 Buf[“BufSize”-1] 的值，运算结果保存到“Out”中。“CurIndex”的值达到“BufSize”后恢复为 1，然后再增加。“Q”的值保持 TRUE。

第4次执行指令

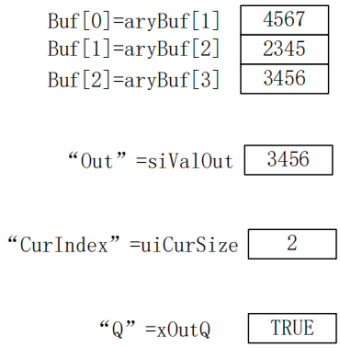


第5次执行指令

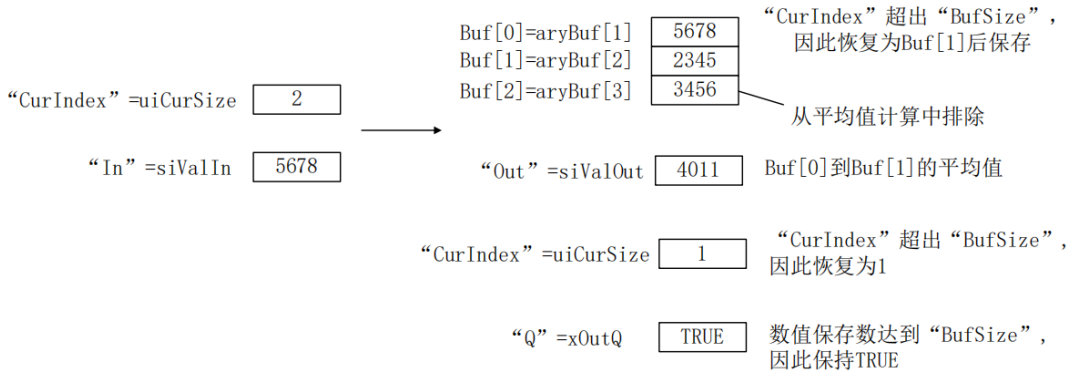


保存值的初始化

若将“CurSize”的值设为 0 后再执行本指令，Buf[0] 到 Buf[“BufSize”-1] 的值会变为 0，然后再将当时“In” 的值保存到 Buf[0] 中。“CurIndex”的值变为 1，“Q”的值变为 FALSE。“BufSize”值的变更若变更“BufSize”的值后再执行本指令，将按照变更后的“BufSize”和当时的“CurIndex”值动作。



变更为“BufSize”=2后执行指令

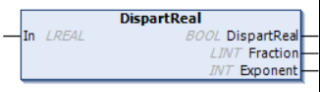


要点说明

- “In”和“Out”以及 Buf[] 要素的数据类型应统一。，否则编译时将会发生异常
- 即使运算结果超出“Out”的有效范围，也不会发生异常，“Out”中保存非法值。
- “BufSize”的值为 0 时，“Out”、“CurIndex”的值为 0,“Q”的值变为 TRUE。
- 变更“BufSize”的值后，“CurIndex”将保持当前值。
- “BufSize”的值超出 Buf[] 的大小时，函数返回值为 FALSE，“Out”无变化。

6.9.9. DispartReal (实数的尾数、指数分离)

将实数分离为带符号尾数部分和指数部分。

指令	名称	FB/FUN	图形表现	ST 表现
DispartReal	实数的尾数、指数分离	FUN		Out:=DispartReal(In,Fraction, Exponent);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	实数	输入	待分离的实数	遵从数据类型	—	(*1)
Out	返回值	输出	始终为 TRUE	仅 TRUE	—	—
Fraction	带符号的尾数部分		带符号的尾数部分	(*2)		
Exponent	指数部分		指数部分	(*3)		

*1 省略输入参数时，初始值不适用。编连时会发生异常。

*2 有效范围因 “In” “Fraction” 的数据类型的组合而异。详情请参阅功能说明。

*3 “In” 为 REAL 型时，有效范围为 -44 ～ 32 ； “In” 为 LREAL 型时，有效范围为 -322 ～ 294。

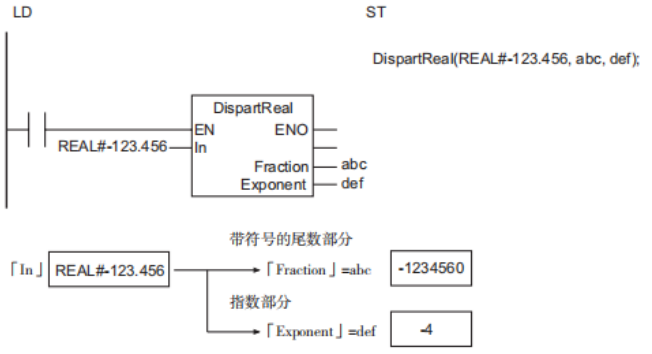
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In															○					
Out	○																			
Fraction													○							
Exponent											○									

功能

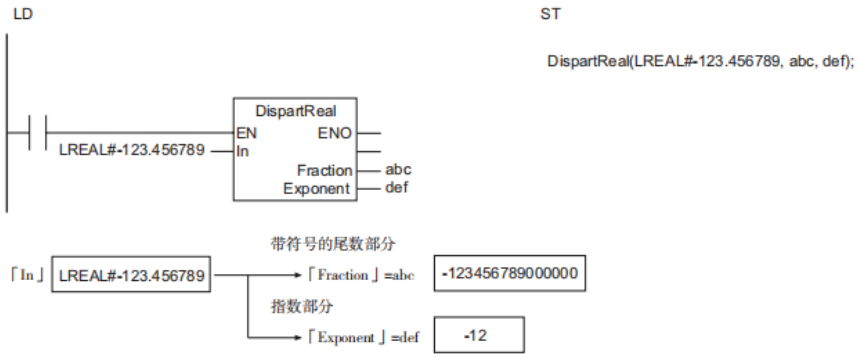
将实数 “In” 分离为带符号尾数部分 “Fraction” 和指数部分 “Exponent”。

“In” 为 REAL 型时，“Fraction” 为 7 位整数。“In” 为 LREAL 型时，“Fraction” 为 15 位整数。

“In” 为 REAL 型，值为 REAL#-123.456 时的示例如下所示。



“In” 为 LREAL 型，值为 LREAL#-123.456789 时的示例如下所示。



	FBD	ST
定义变量	VAR In :LREAL; Fraction :LINT; Exponent :INT; END_VAR	
程序	In — DispartReal: In, Fraction — Fraction, Exponent — Exponent	DispartReal(In:=In , Fraction=>Fraction , Exponent=>Exponent);
运行结果	In LREAL -123.456789 Fraction LINT -123456789000000 Exponent INT -12	

参考

组合带符号的尾数部分和指数部分来获取实数时，请使用 “UniteReal 指令 (P.2-427)”。

要点说明

- 转换为整数时，可能因 “In” 的值而产生误差。
- “In”的有效位数大于“Fraction”的有效位数时，进行五舍五入，以将其控制在“Fraction”的有效范围内。五舍五入是指如下所示的处理。

小数部分的值	处理	例
小于 0.5	舍去	1.49→1 -1.48→-1
大于等于 0.5	进位	1.51→2 -1.51→-2

- 以下情况时会发生异常。ENO 变为 FALSE，“Fraction” “Exponent”不变。
- “In” 的值为非数或无穷大时。

6. 9. 10. UnitReal (将尾数、指数结合为实数)

将带符号的尾数部分和指数部分组合成实数。

指令	名称	FB/FUN	图形表现	ST 表现
UniteReal	尾数、指数组合成实数	FUN		Out:=UniteReal(Fraction, Exponent);

变量

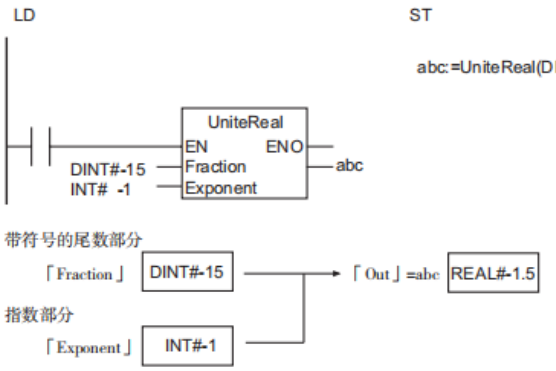
	名称	输入/输出	内容	有效范围	单位	初始值
Fraction	带符号的尾数部分	输入	带符号的尾数部分	遵从数据类型	—	(*)
Exponent	指数部分		指数部分			0
Out	实数	输出	实数	遵从数据类型	—	—

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Fraction													○							
Exponent											○									
Out															○					

功能

组合带符号尾数部分 “Fraction” 和指数部分 “Exponent”，获取实数 “Out”。“Fraction”=DINT#-15，“Exponent”=INT#-1 时的示例如下所示。



	FBD	ST
定义变量	VAR Fraction :LINT; Exponent :INT; Out :LREAL; END_VAR	
程序	Fraction → Fraction, Exponent → Exponent, UniteReal → Out	Out:=UniteReal(Fraction:=Fraction , Exponent:=Exponent);
运行结果	Fraction LINT -15 Exponent INT -1 Out LREAL -1.5	

参考

将实数分离为带符号尾数部分和指数部分时，请使用 “DispartReal 指令”。

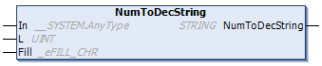
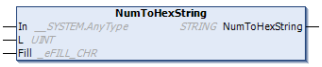
要点说明

- 将整数转换为实数时，可能 “Fraction” 和 “Exponent” 的值而产生误差。
- 组合结果超过“Out”的有效范围时，如果“Exponent”为正数，则“Out”的值为与“Fraction”具有相同符号的无穷大。如果 “Exponent” 为负数，则 “Out” 的值为 0。

6.9.11. NumToDecString/NumToHexString (固定长度 10/16 进制字符串转换)

NumToDecString：将整数转换为固定长度的 10 进制字符串格式。

NumToHexString：将整数转换为固定长度的 16 进制字符串格式。

指令	名称	FB/FUN	图形表现	ST 表现
NumToDecString	固定长度 10 进制字符串转换	FUN		Out:=NumToDecString(In, L, Fill);
NumToHexString	固定长度 16 进制字符串转换	FUN		Out:=NumToHexString(In, L, Fill);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	整数	输入	整数	遵从数据类型	—	(*)
L	字符数		“Out”的字符数	0~1985		1
Fill	添加字符		添加字符	_BLANK _ZERO	—	_BLANK
Out	字符串	输出	字符串	遵从数据类型	—	—

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In						○	○	○	○	○	○	○	○							
L							○													
Fill	枚举体_eFILL_CHR 枚举元素参阅功能说明																			
Out																				○

功能

NumToDecString：将整数“In”转换为 UTF-8 半角英数字的 10 进制格式的字符串。“In”为负数时，在开头加上'-'(减号)。

NumToHexString：将整数 “In” 转换为 UTF-8 半角英数字的 16 进制格式的字符串。“In” 为负数时，以 2 的补数形式表现(位取反+1)。

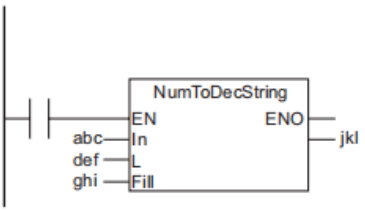
上述指令均将字符串 “Out” 的字符数设为 “L”。字符数不足时，将添加字符 “Fill” 指定的字符添加在高位上。转换结果的字符数多于 “L” 时，从转换结果的最低位仅将 “L” 字符代入 “Out”。“Out” 的结尾带 NULL 字符。字符数中不包含 NULL 字符。

“Fill” 的数据类型为枚举体 _eFILL_CHR。枚举元素的含义如下所示。

枚举元素	含义
_BLANK	‘ ‘（半角空白字符）
_ZERO	‘0’

NumToDecString 指令的几个示例如下所示。

LD



ST

```
jkl:=NumToDecString(abc, def, ghi);
```

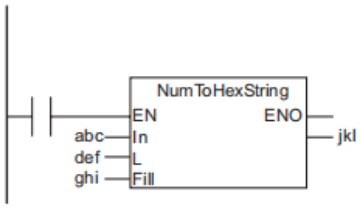
「In」=abc=INT#128、「L」=def=UINT#8、「Fill」=ghi=_BLANK
「Out」=jkl[ ][ ][ ][ ][ ]1 2 8

「In」=abc=INT#-128、「L」=def=UINT#8、「Fill」=ghi=_BLANK
「Out」=jkl[ ][ ][ ][ ]- 1 2 8

「In」=abc=INT#-128、「L」=def=UINT#8、「Fill」=ghi=_ZERO
「Out」=jkl- 0 0 0 0 1 2 8

NumToHexString 指令的几个示例如下所示。

LD



ST

```
jkl:=NumToHexString(abc, def, ghi);
```

「In」=abc=INT#128、「L」=def=UINT#8、「Fill」=ghi=_BLANK
「Out」=jkl[ ][ ][ ][ ][ ][ ]8 0

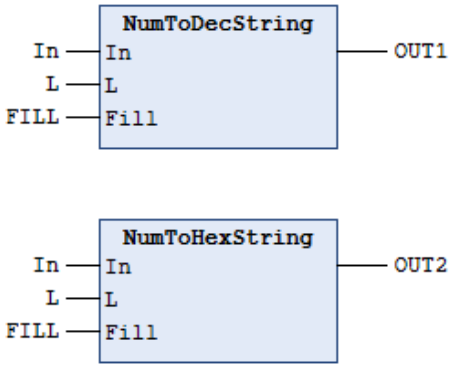
「In」=abc=INT#128、「L」=def=UINT#8、「Fill」=ghi=_ZERO
「Out」=jkl0 0 0 0 0 0 8 0

「In」=abc=INT#-128、「L」=def=UINT#8、「Fill」=ghi=_BLANK
「Out」=jkl F F F F F F 8 0

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">

```
VAR
    In:INT;
    L:UINT;
    FILL:_eFILL_CHR;
    OUT:STRING;
END_VAR
```

</td></tr>
<tr><td>程序</td><td>



</td><td>

```
OUT1:=NumToDecString(
    In:=In ,
    L:=L ,
    Fill:=FILL );

OUT2:=NumToHexString(
    In:=In ,
    L:=L ,
    Fill:=FILL );
```

</td></tr>
<tr><td>运行结果</td><td colspan="2">

表达式	类型	值
In	INT	12345
L	UINT	6
FILL	_EFILL_CHR	_ZERO
OUT1	STRING	'012345'
OUT2	STRING	'003039'

</td></tr>
</table>

要点说明

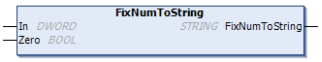
- “L” 的值为 0 时，“Out” 的值不变。
- 转换结果的字符数多于 “L” 时，从转换结果的低位将 “L” 位保存在 “Out” 中。示例如下所示。

指令	“In”的值	“L”的值	“Out”的值
NumToDecString	128	2	28
NumToHexString			80

- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- “L” 的值超过有效范围时。
- “Fill” 的值超过有效范围时。
- 转换结果超过 “Out” 的范围时。

6.9.12. FixNumToString (固定小数点数 TO 字符串转换)

将带符号的固定小数点数转换为 10 进制字符串格式。

指令	名称	FB/FUN	图形表现	ST 表现
FixNumToString	固定小数点数→字符串转换	FUN		Out:=FixNumToString(In, Zero);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	固定小数点数	输入	带符号的固定小数点数	遵从数据类型	—	0
Zero	零显示		小数点后的位数不到 3 时的显示 TRUE ：补 “0” FALSE：不补 “0”			TRUE
Out	10 进制字符串	输出	10 进制字符串	遵从数据类型	—	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In				○																
Zero	○																			
Out																				○

功能

将带符号的固定小数点数 “In” 转换为 10 进制字符串。转换步骤如下所示。

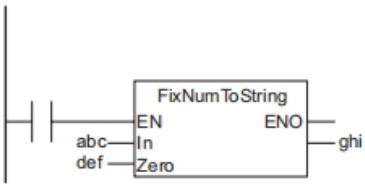
1. 将 16 进制格式的 “In” 转换为 10 进制格式。

2. 将该值除以 1,000。

“In” 的小数点后位数不到 3 时，零显示 “Zero” 指定 “Out” 的小数点后第 3 位前是否补 “0”。

如果“Zero” 的值为 TRUE，则补'0'。“Out”的结尾带 NULL 字符。几个示例如下所示。

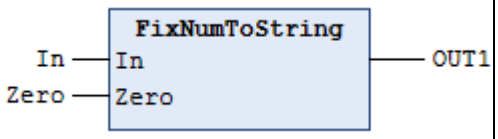
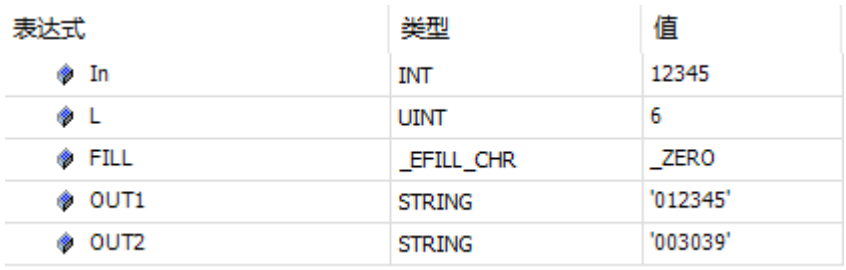
LD



ST

```
ghi:=FixNumToString(abc, def);
```

"In" =abc	"Out" =ghi	
	"Zero" =def=TRUE	"Zero" =def=FALSE
16#0001462C (10#83500)	'83.500'	'83.5'
16#00051AA4 (10#334500)	'334.500'	'334.5'
16#0003BEFC (10#245500)	'245.500'	'245.5'

	FBD	ST
定义变量	VAR In:INT; L:UINT; FILL:_eFILL_CHR; OUT:STRING; END_VAR	
程序		OUT1:=FixNumToString(In:=In , Zero:=Zero);
运行结果		

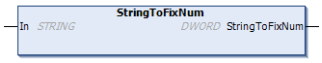
表达式	类型	值
In	INT	12345
L	UINT	6
FILL	_EFILL_CHR	_ZERO
OUT1	STRING	'012345'
OUT2	STRING	'003039'

要点说明

- "In" 的小数点后 4 位之后舍去。
- 忽略 "In" 字符串中的下划线 (16#5F)。
- 以下情况时会发生异常。ENO 变为 FALSE，"Out" 不变。
- "In" 未以 NULL 字符结尾时。
- "In" 的内容为无法进行数值转换的字符时。
- "In" 的内容有小数点，但无小数部分时。
- 转换结果超过 "Out" 的有效范围时。

6.9.13. StringToFixNum (字符串 TO 固定小数点数转换)

将 10 进制字符串转换为带符号的固定小数点数形式。

指令	名称	FB/FUN	图形表现	ST 表现
StringToFixNum	字符串→固定小数点数转换	FUN		Out:=StringToFixNum(In);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	10 进制字符串	输入	10 进制字符串	遵从数据类型	—	‘’
Out	固定小数点数	输出	带符号的固定小数点数	遵从数据类型	—	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out				○																

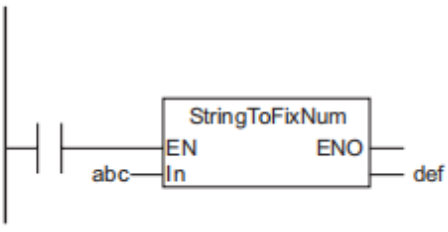
功能

将 10 进制字符串 “In” 转换为固定小数点数。转换步骤如下所示。

1. 将 “In” 表现的数值乘以 1,000。
2. 该值的小数点后舍去。
3. 将该值转换为 32 位的 16 进制格式 (DWORD 型)。

几个值的示例如下所示。

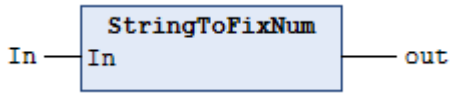
LD



ST

```
def:=StringToFixNum(abc);
```

“In”=abc	“Out”=def
‘83.5’	16#0001462C （10#83500）
‘334.5’	16#00051AA4 （10#334500）
‘245.5’	16#0003BEFC （10#245500）

	FBD	ST
定义变量	VAR In:STRING; out:DWORD; END_VAR	
程序		out:=StringToFixNum(In:=In);
运行结果		

要点说明

- “In” 的小数点后 4 位之后舍去。
- 忽略 “In” 字符串中的下划线 (16#5F)。
- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- “In” 未以 NULL 字符结尾时。
- “In” 的内容为无法进行数值转换的字符时。
- “In” 的内容有小数点，但无小数部分时。
- 转换结果超过 “Out” 的有效范围时

6.9.14. DtToString (日期时间 TO 字符串转换)

将日期时间转换为字符串格式。

指令	名称	FB/FUN	图形表现	ST 表现
DtToString	日期时间 →字符串转换	FUN		Out:=DtToString(In);

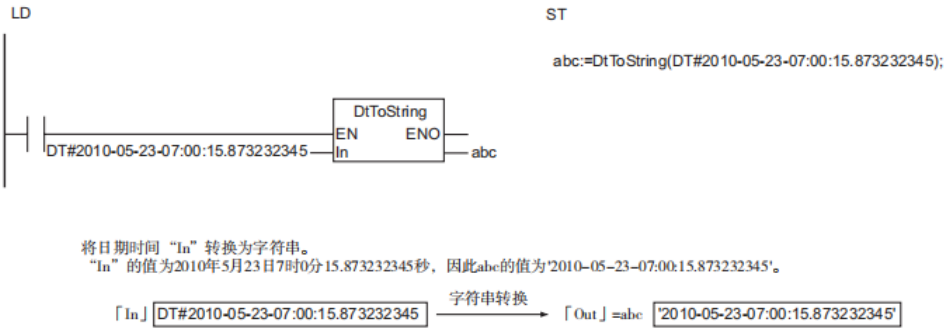
变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	日期时间	输入	日期时间	遵从数据类型	年月日时分秒	DT#1970-1-1-0:0:0
Out	字符串	输出	字符串	30 字节（29 个半角英数字字符+结尾 NULL 字符）	—	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			○	
Out																				○

功能

将日期时间 “In”转换为字符串。字符串“Out”的结尾带 NULL 字符。“In”的值为 2010 年 5 月 23 日 7 时 0 分 15.873232345 秒时的示例如下所示。变量 abc 的值为'2010-05-23-07:00:15.873232345'。



	FBD	ST
定义变量	VAR In:DATE_AND_TIME; out:STRING; END_VAR	
程序	In — DtToString (In) — out	out:=DtToString(In:=In);

运行结果	表达式	类型	值
	In	DATE_AND_TIME	DT#1970-1-1-0:0:0
	out	STRING	'1970-01-01-00:00:00'

要点说明

- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- 转换结果超过 “Out” 的有效范围时。

6.9.15. DateToString (日期 TO 字符串转换)

将日期转换为字符串格式。

指令	名称	FB/FUN	图形表现	ST 表现
DateToString	日期 →字符串转换	FUN		Out:=DateToString(In);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	日期	输入	日期	遵从数据类型	年月日	D#1970-1-1
Out	字符串	输出	字符串	11 字节（10 个半角英数字字符+结尾 NULL 字符）	—	—

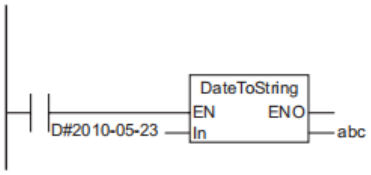
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	○			
Out																				○

功能

将日期 “In” 转换为字符串。字符串 “Out” 的结尾带 NULL 字符。

“In”的值为 2010 年 5 月 23 日时的示例如下所示。变量 abc 的值为 '2010-05-23'。

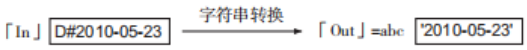
LD

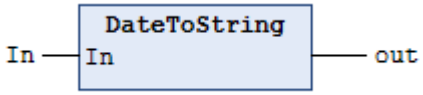


ST

```
abc:=DateToString(D#2010-05-23);
```

将日期“In”转换为字符串。
“In”的值为2010年5月23日，因此abc的值为'2010-05-23'。



	FBD	ST
定义变量	VAR In:DATE; out:STRING; END_VAR	
程序		out:=DateToString(In:=In);
运行结果	表达式 / 类型 / 值 In / DATE / D#1970-1-1 out / STRING / '1970-01-01'	

要点说明

- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- 转换结果超过 “Out” 的有效范围时。

6. 9. 16. StringToAry (字符串 TO 排列转换)

将字符串转换为 BYTE 型数组。

指令	名称	FB/FUN	图形表现	ST 表现
StringToAry	字符串→数组转换	FUN		StringToAry(In:= , AryOut:=);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	字符串	输入	字符串	遵从数据类型	—	‘’
AryOut[ ]数组	BYTE 型数组	输入输出	BYTE 型数组	遵从数据类型	—	—
Out	转换后的字节数	输出	转换后的字节数	0~1985	字节	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
AryOut[ ]数组		○																		
Out							○													

功能

将字符串“In”的字符码视为数值，按字符保存在 BYTE 型数组 AryOut[ ]中。“Out”中保存转换后的字节数。

“In”='XYZ' 时的示例如下所示。

	FBD	ST
定义变量	PROGRAM PLC_PRG VAR In :STRING(1985) :='XYZ'; AryOut :ARRAY [0..4] OF BYTE; Out :UINT; check :BOOL; END_VAR	

程序	In 'XYZ' — In
	AryOut[0] 16#58 — AryOut
	StringToAry
	Out — 16#0003
		IF check FALSE THEN
		Out 16#0003 :=StringToAry(
		In:= In 'XYZ',
		AryOut:= AryOut[0] 16#58);
		check FALSE :=FALSE;
		END_IF RETURN

运行结果	表达式 / 类型 / 值 In / STRING(1985) / 'XYZ' AryOut / ARRAY [0..4] OF BYTE / AryOut[0] / BYTE / 16#58 AryOut[1] / BYTE / 16#59 AryOut[2] / BYTE / 16#5A AryOut[3] / BYTE / 16#00 AryOut[4] / BYTE / 16#00 Out / UINT / 16#0003	

要点说明

- “In” 结尾的 NULL 字符不保存在 AryOut[ ] 中。
- “In” 为仅有 NULL 字符的字符串时，“Out” 的值变为 0，AryOut[ ] 不变。
- “In"的字节数超出 AryOut[ ]的元素数时，超出部分无法存储。

6.9.17. AryToString (排列 TO 字符串转换)

将 BYTE 型数组转换为字符串。

指令	名称	FB/FUN	图形表现	ST 表现
AryToString	数组→字符串转换	FUN		AryToString(In:= , Size:= , Out=>);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In[ ]数组	BYTE 型数组	输入	BYTE 型数组,元素数最大位 1985	遵从数据类型	—	(*)
Size	转换元素数		待转换的 In[ ]的元素数	0~1985		1
Out	字符串	输出	字符串	遵从数据类型	—	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ]数组		○																		
Size							○													
Out																				○

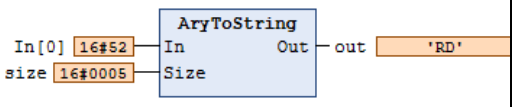
功能

将 BYTE 型的数组 In[0] 之后的各元素的值视为字符码，保存在字符串 “Out” 中。“Out” 的结尾带 NULL 字符。为 “Size” 指定待转换的 In[ ] 的元素数。In[0] ～ In[“Size”-1] 中含有 NULL 字符时，仅该处前的内容保存在 “Out” 中。

“Size”=UINT#5，时的示例如下所示。只转换 NULL 字符之前的数据到 Out 中，即将 In[0]=16#52,In[1]=16#44。转换为字符串，即‘RD’。

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">

```
PROGRAM PLC_PRG
VAR
    In: ARRAY[0..4] OF byte := [16#52, 16#44, 16#0, 16#52, 16#44];
    size    :UINT:=5;
    out     :STRING;
    check   :BOOL;
END_VAR
```

</td></tr>
<tr><td>程序</td><td> 
</td><td>

```
IF check FALSE THEN
    AryToString(
        In:= In[0] 16#52,
        Size:= size 16#0005,
        Out=> out 'RD' );
    check FALSE :=FALSE;
END_IF RETURN
```

</td></tr>
<tr><td>运行结果</td><td colspan="2">
<table>
<tr><th>表达式</th><th>类型</th><th>值</th></tr>
<tr><td>In</td><td>ARRAY [0..4] OF BYTE</td><td></td></tr>
<tr><td>In[0]</td><td>BYTE</td><td>16#52</td></tr>
<tr><td>In[1]</td><td>BYTE</td><td>16#44</td></tr>
<tr><td>In[2]</td><td>BYTE</td><td>16#00</td></tr>
<tr><td>In[3]</td><td>BYTE</td><td>16#52</td></tr>
<tr><td>In[4]</td><td>BYTE</td><td>16#44</td></tr>
<tr><td>size</td><td>UINT</td><td>16#0005</td></tr>
<tr><td>out</td><td>STRING</td><td>'RD'</td></tr>
</table>
</td></tr>
</table>

要点说明

- “Size” 的值为 0 时，“Out” 为仅含有 NULL 字符的字符串。
- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- “Size” 的值超过 In[ ] 的数组区域时。
- 转换结果超过 “Out” 的有效范围时。

6.9.18. RoundUp (实数舍入)

小数点后第 1 位进位。

指令	名称	FB/FUN	图形表现	ST 表现
RoundUp	实数进位	FUN		RoundUp(In，Out);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	转换对象	输入	转换对象	遵从数据类型	—	(*)
Out	转换结果		转换结果	遵从数据类型	—	—

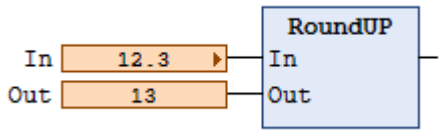
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In														○	○					
Out												○	○							

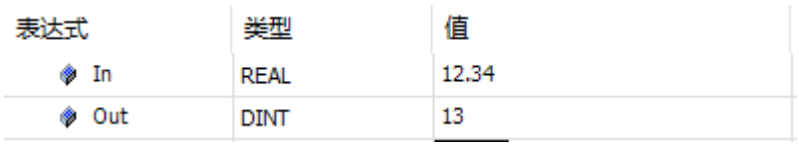
功能

对实数 “In” 的小数点后第 1 位进行进位处理，使其成为整数。

Real 类型的“In” =12.34 时，示例如下。

	FBD	ST
定义变量	PROGRAM PLC_PRG VAR In :REAL:=12.34; Out :DINT; check :BOOL; END_VAR	

程序		IF check FALSE THEN
		RoundUP(
		In:=In 12.3 ,
		Out:=Out 13);
		check FALSE :=FALSE;
		END_IF RETURN

运行结果		

要点说明

- 转换结果超过 “Out” 的有效范围时，“Out” 的值为错误值。

6.9.19. TodToString (时刻 TO 字符串转换)

将时刻转换为字符串格式。

指令	名称	FB/FUN	图形表现	ST 表现
TodToString	时刻→字符串转换	FUN		Out:=TodToString(In);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	时刻	输入	时刻	遵从数据类型	时分秒	TOD#0:0:0
Out	字符串	输出	字符串	13 字节（12 个半角英数字字符+结尾 NULL 字符）	—	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																		○		
Out																				○

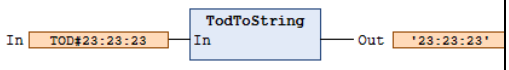
功能

将时刻 “In” 转换为字符串。“Out” 的结尾带 NULL 字符。

输入变量“In” = 23:23:23 时，示例如下。

	FBD	ST
定义变量		
程序		
运行结果		

```
PROGRAM PLC_PRG
VAR
    In          :TOD;
    Out         :STRING;
END_VAR
```




```
out := TodToString( In:= In );
```

表达式	类型	值
In	TIME_OF_DAY	TOD#23:23:23
Out	STRING	'23:23:23'

要点说明

- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- 转换结果超过 “Out” 的有效范围时。

6.9.20. StringToDt (字符串 TO 日期时间转换)

将字符串转换为日期格式。

指令	名称	FB/FUN	图形表现	ST 表现
StringToDt	字符串→日期转换	FUN		StringToDt(In:= , Out=>);

变量

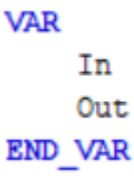
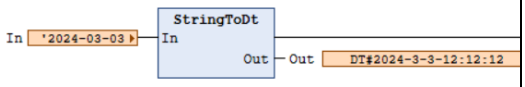
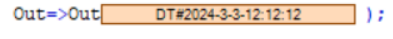
	名称	输入/输出	内容	有效范围	单位	初始值
In	字符串	输入	字符串	遵从数据类型		
Out	日期	输出	日期时间	遵从数据类型	年月日时分秒	DT#1970-1-1-0:0:0

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out																			○	

功能

将字符串 “In” 转换为字符串日期。

输入变量“In” =’2024-03-03-12:12:12’ 时，示例如下。

	FBD	ST
定义变量		
程序		
运行结果		

表达式	类型	值
In	STRING	'2024-03-03-12:12:12'
Out	DATE_AND_TIME	DT#2024-3-3-12:12:12

6.9.21. AryToWstring (排列 TO 字符串转换)

将 BYTE 型数组转换为字符串。

指令	名称	FB/FUN	图形表现	ST 表现
AryToWstring	数组→字符串转换	FUN		AryToWstring(In:= , Size:= , Out=>);

变量

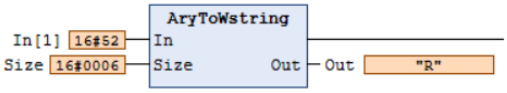
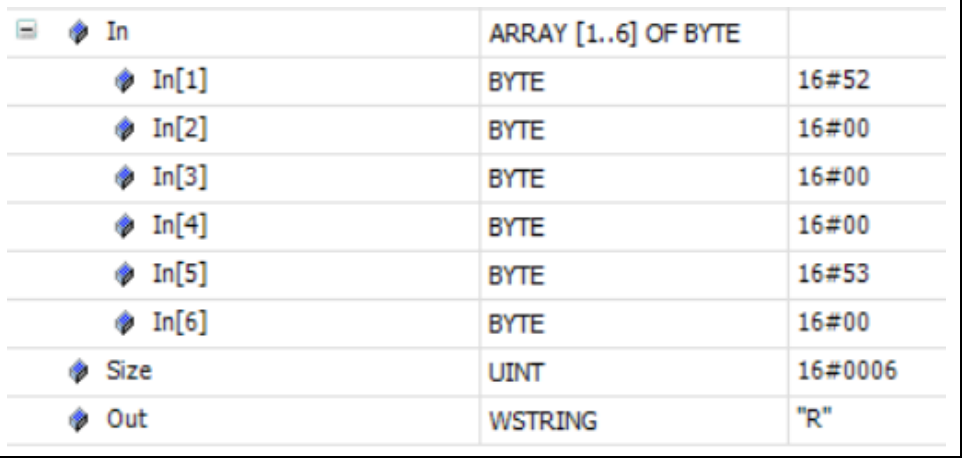
	名称	输入/输出	内容	有效范围	单位	初始值
In[ ]数组	BYTE 型数组	输入	BYTE 型数组,元素数最大位 1985	遵从数据类型	—	(*)
Size	转换元素数		待转换的 In[ ]的元素数	0~1985		1
Out	字符串	输出	字符串	遵从数据类型	—	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	WSTRING
In[ ]数组		○																		
Size							○													
Out																				○

功能

将 BYTE 型的数组 In[0]之后的两个元素的值视为一个字符码，保存在字符串“Out”中。“Out”的结尾带 NULL 字符。“Size”指定待转换的 In[ ]的元素数。In[0]～In[“Size”-1]中含有连续两个 NULL 字符时，仅该处前的内容保存在“Out”中。

“Size”=UINT#6，时的示例如下所示。只转换 NULL 字符之前的数据到 Out 中，即将 In[0]=16#52,In[1]=16#00。转换为字符串，即‘R’。

	FBD	ST
定义变量	VAR In :ARRAY[1..6] OF BYTE :=[16#52,16#00,16#0,16#0,16#44,16#00]; Size :UINT :=6; Out :WSTRING; END_VAR	
程序		AryToWstring(In:= In[1] 16#52, Size:= Size 16#0006, Out=> Out "R");
运行结果		

要点说明

- “Size” 的值为 0 时，“Out” 为仅含有 NULL 字符的字符串。
- 一个 Wstring 转换需要两个 Byte。
- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- “Size” 的值超过 In[ ] 的数组区域时。
- 转换结果超过 “Out” 的有效范围时。

6.9.22. WstringToAry (字符串 TO 排列转换)

将字符串转换为 BYTE 型数组。

指令	名称	FB/FUN	图形表现	ST 表现
WstringToAry	字符串→数组转换	FUN		WstringToAry(In:= , AryOut:=);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	字符串	输入	字符串	遵从数据类型	—	‘’
AryOut[ ]数组	BYTE 型数组	输入输出	BYTE 型数组	遵从数据类型	—	—
Out	转换后的字节数	输出	转换后的字节数	0~1985	字节	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	WSTRING
In																				○
AryOut[ ]数组		○																		
Out							○													

功能

将字符串“In”的字符码视为数值，按字符保存在 BYTE 型数组 AryOut[ ]中。“Out”中保存转换后的字节数。

“In” = "ABC" 时的示例如下所示。

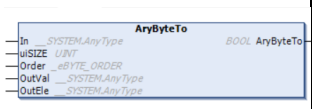
	FBD	ST
定义变量	VAR AryOut :ARRAY[1..6] OF BYTE ; Out :UINT; In :WSTRING :="ABC"; END_VAR	
程序	In "ABC" AryOut[1] 16#41 WstringToAry In AryOut Out 16#0003	Out 16#0003 :=WstringToAry(In:= In "ABC" AryOut:= AryOut[1] 16#41);
运行结果	AryOut ARRAY [1..6] OF BYTE AryOut[1] BYTE 16#41 AryOut[2] BYTE 16#00 AryOut[3] BYTE 16#42 AryOut[4] BYTE 16#00 AryOut[5] BYTE 16#43 AryOut[6] BYTE 16#00 Out UINT 16#0003 In WSTRING "ABC"	

要点说明

- “In” 结尾的 NULL 字符不保存在 AryOut[ ] 中。
- “In” 为仅有 NULL 字符的字符串时，“Out” 的值变为 0，AryOut[ ] 不变。
- “In"的字节数超出 AryOut[ ]的元素数时，超出部分无法存储。
- 一个 Wstring 转换需要两个 Byte。

6.9.23. AryByteTo (从字节排列转换)

结合 BYTE 型排列要素，并保存到变量中。

指令	名称	FB/FUN	图形表现	ST 表现
AryByteTo	从字节排列转换	FUN		AryByteTo(In:= , uiSIZE:= , Order:= , OutVal:= , OutEle:=);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In[ ]排列	转换对象排列		转换对象的排列	遵从数据类型		(*)
Size	转换要素数		要转换的 In[] 的要素数	遵从数据类型		
Order	转换顺序		转换的顺序	_LOW_HIGH, _HIGH_LOW		_LOW_HIGH
OutVal	结果数组		结果数组	遵从数据类型		
OutEle	结果数组元素		结果数组开始元素	遵从数据类型		

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ]排列		○																		
Size							○													
Order	列举型 _eBYTE_ORDER 列举值参考功能说明																			
OutVal	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
OutEle	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

功能

将转换对象排列 In[]的起始“Size”个要素，根据数组“OutVal”的某位元素“OutEle”的数据类型大小计算，然后保存到“OutVal”中。

结合 In[] 要素时的顺序，在转换顺序“Order”中指定。“Order”的数据类型为列举型 _eBYTE_ORDER。

列举值的含义如下所示

列举值	含义
_LOW_HIGH	先低位字节，后高位字节
_HIGH_LOW	先高位字节，后低位字节

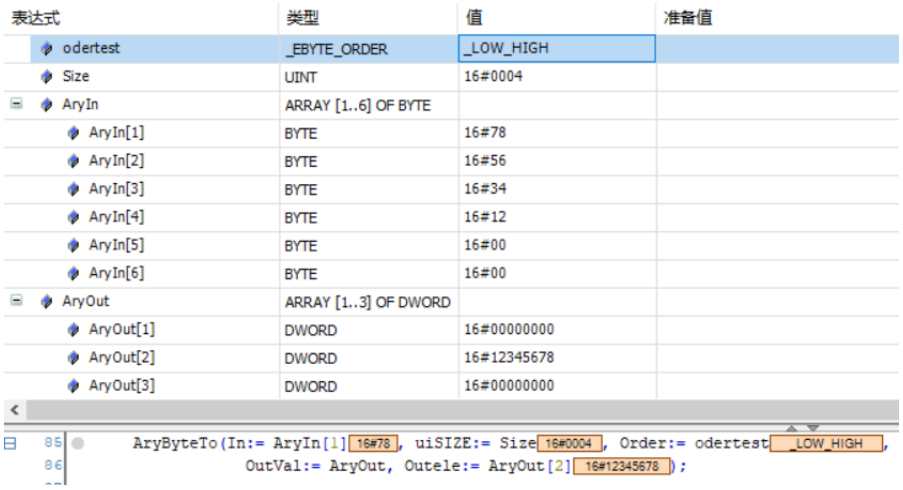
“OutEle”的数据类型为 2 字节以上时，处理步骤如下：

1.根据“Order”的值结合 In[0]和 In[1]，创建 1 个字(2 字节)的数据。若“Order”的值为_LOW_HIGH，则高位字节为 In[1]、低位字节为 In[0]。若“Order”的值为_HIGH_LOW，则高位字节为 In[0]、低位字节为 In[1]。

2.以相同的方法结合 In[2]和 In[3]之后的值，创建多个 1 个字的数据。

3.直到使用了“Size”个元素数据，将数据依次保存在“OutVal”中。

“OutVal”为 DWORD 型数组、Size=UINT#4、“In”为 BYTE 数组的第 1 个元素、Qrder=_LOW_HIGH、“OutEle”为“OutVal”数组第 1 个元素时，示例如下：



“OutVal”为 DWORD 型数组、Size=UINT#4、“In”为 BYTE 数组的第 3 个元素、Qrder=_HIGH_LOW、“OutEle”为“OutVal”数组第 2 个元素时，示例如下：

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_HIGH_LOW	
Size	UINT	16#0004	
AryIn	ARRAY [1..6] OF BYTE		
AryIn[1]	BYTE	16#78	
AryIn[2]	BYTE	16#56	
AryIn[3]	BYTE	16#34	
AryIn[4]	BYTE	16#12	
AryIn[5]	BYTE	16#91	
AryIn[6]	BYTE	16#92	
AryOut	ARRAY [1..3] OF DWORD		
AryOut[1]	DWORD	16#00000000	
AryOut[2]	DWORD	16#91923412	
AryOut[3]	DWORD	16#00000000	

```
85    AryByteTo(In:= AryIn[3] 16#34, uiSIZE:= Size 16#0004, Order:= odertest _HIGH_LOW,
86              OutVal:= AryOut, Outele:= AryOut[2] 16#91923412);
```

“OutEle”的数据类型为 1 字节时，示例如下：

“OutVal”为 SINT 型数组、Size =UINT#3、“In”为 BYTE 数组的第 1 个元素、Qrder =_LOW_HIGH、

“OutEle”为“OutVal”数组第 1 个元素时，示例如下：

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_LOW_HIGH	
Size	UINT	16#0003	
AryIn	ARRAY [1..6] OF BYTE		
AryIn[1]	BYTE	16#11	
AryIn[2]	BYTE	16#22	
AryIn[3]	BYTE	16#33	
AryIn[4]	BYTE	16#00	
AryIn[5]	BYTE	16#00	
AryIn[6]	BYTE	16#00	
AryOut	ARRAY [1..4] OF SINT		
AryOut[1]	SINT	16#11	
AryOut[2]	SINT	16#22	
AryOut[3]	SINT	16#33	
AryOut[4]	SINT	16#00	

```
85    AryByteTo(In:= AryIn[1] 16#11, uiSIZE:= Size 16#0003, Order:= odertest _LOW_HIGH,
86              OutVal:= AryOut, Outele:= AryOut[1] 16#11);
```

其他条件与上述描述相同，Qrder =_HIGH_LOW，示例如下：

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_HIGH_LOW	
Size	UINT	16#0003	
AryIn	ARRAY [1..6] OF BYTE		
AryIn[1]	BYTE	16#11	
AryIn[2]	BYTE	16#22	
AryIn[3]	BYTE	16#33	
AryIn[4]	BYTE	16#00	
AryIn[5]	BYTE	16#00	
AryIn[6]	BYTE	16#00	
AryOut	ARRAY [1..4] OF SINT		
AryOut[1]	SINT	16#22	
AryOut[2]	SINT	16#11	
AryOut[3]	SINT	16#00	
AryOut[4]	SINT	16#33	

```
85    AryByteTo(In:= AryIn[1] 16#11, uiSIZE:= Size 16#0003, Order:= odertest _HIGH_LOW,
86              OutVal:= AryOut, Outele:= AryOut[1] 16#22);
```

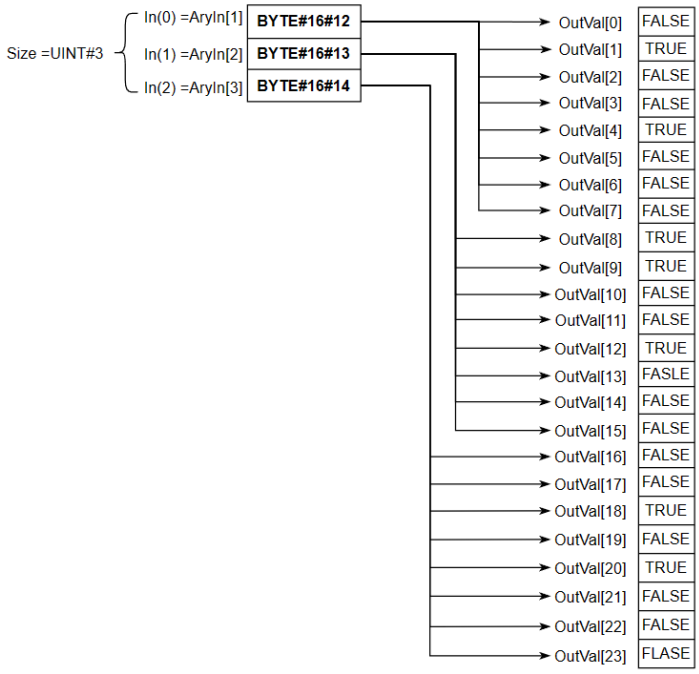
“OutEle”的数据类型为 BOOL 时，示例如下：

“OutVal”为 BOOL 型数组、Size =UINT#3、 “In”为 BYTE 数组的第 1 个元素、Qrder =_LOW_HIGH、

“OutEle”为“OutVal”数组第 1 个元素时，示例如下：

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_LOW_HIGH	
Size	UINT	16#0003	
AryIn	ARRAY [1..4] OF BYTE		
AryIn[1]	BYTE	16#12	
AryIn[2]	BYTE	16#13	
AryIn[3]	BYTE	16#14	
AryIn[4]	BYTE	16#00	
AryOut	ARRAY [1..32] OF BOOL		
AryOut[1]	BOOL	FALSE	
AryOut[2]	BOOL	TRUE	
AryOut[3]	BOOL	FALSE	
AryOut[4]	BOOL	FALSE	
AryOut[5]	BOOL	TRUE	
AryOut[6]	BOOL	FALSE	
AryOut[7]	BOOL	FALSE	
AryOut[8]	BOOL	FALSE	
AryOut[9]	BOOL	TRUE	
AryOut[10]	BOOL	TRUE	
AryOut[11]	BOOL	FALSE	

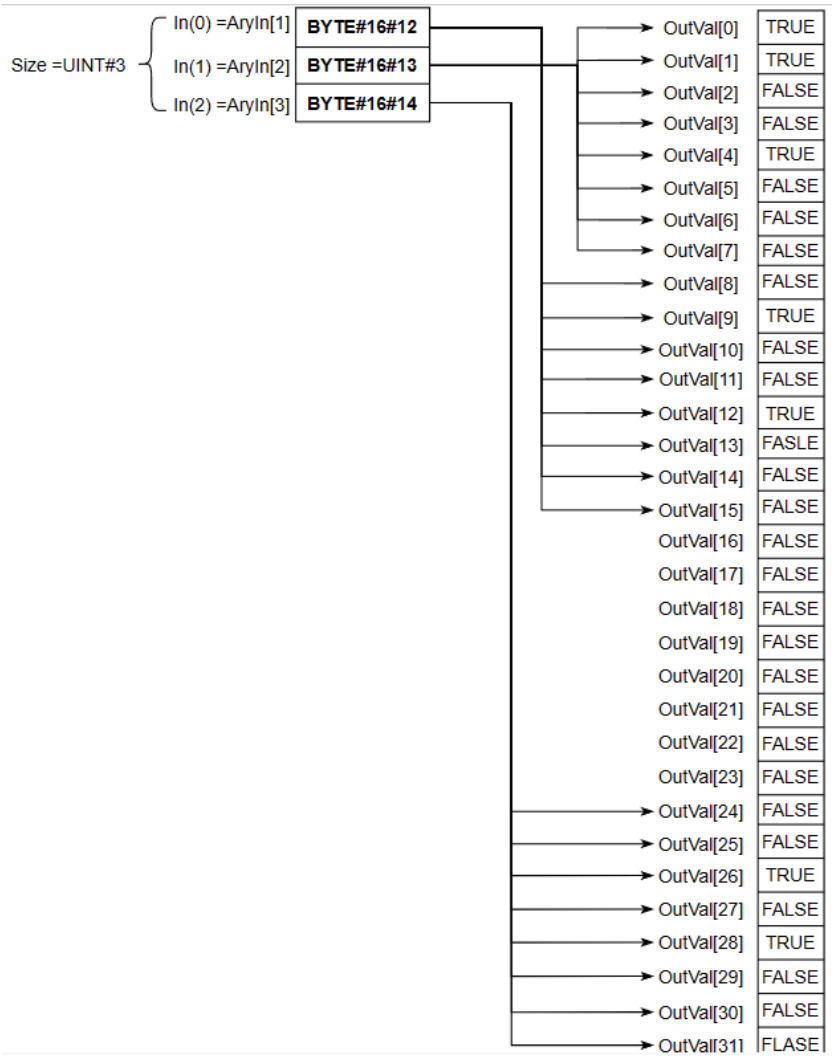
```
85      AryByteTo(In:= AryIn[1] 16#12 , uiSIZE:= Size 16#0003 , Order:= odertest _LOW_HIGH ,
86                OutVal:= AryOut, Outele:= AryOut[1] FALSE );
```



其他条件与上述描述相同，Qrder =_HIGH_LOW，示例如下：

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_HIGH_LOW	
Size	UINT	16#0003	
AryIn	ARRAY [1..4] OF BYTE		
AryIn[1]	BYTE	16#12	
AryIn[2]	BYTE	16#13	
AryIn[3]	BYTE	16#14	
AryIn[4]	BYTE	16#00	
AryOut	ARRAY [1..32] OF BOOL		
AryOut[1]	BOOL	TRUE	
AryOut[2]	BOOL	TRUE	
AryOut[3]	BOOL	FALSE	
AryOut[4]	BOOL	FALSE	
AryOut[5]	BOOL	TRUE	
AryOut[6]	BOOL	FALSE	
AryOut[7]	BOOL	FALSE	
AryOut[8]	BOOL	FALSE	
AryOut[9]	BOOL	FALSE	
AryOut[10]	BOOL	TRUE	
AryOut[11]	BOOL	FALSE	

```
85    AryByteTo(In:= AryIn[1] 16#12 , uiSIZE:= Size 16#0003 , Order:= odertest _HIGH_LOW ,
86              OutVal:= AryOut, Outele:= AryOut[1] TRUE );
```

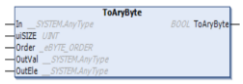


要点说明

- “In”输入数组格式必须是 Byte 数组元素
- “OutVal”填写数组即可，“OutEle”填写起始操作的数组元素
- 输出数组内存不够转换时，函数返回状态 FALSE，不做任何处理。

6.9.24. ToAryByte (转换为字节排列)

将变量以 1 字节未单位分割，并保存到 BYTE 信排列中。

指令	名称	FB/FUN	图形表现	ST 表现
ToAryByte	转换为字节排列	FUN		ToAryByte(In:= , uiSIZE:= , Order:= , OutVal:= , OutEle:=);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In[ ]排列	转换对象排列		转换对象的排列	遵从数据类型		(*)
Size	转换要素数		要转换的 In[] 的要素数	遵从数据类型		
Order	转换顺序		转换的顺序	_LOW_HIGH, _HIGH_LOW		_LOW_HIGH
OutVal	结果数组		BYTE 数组	遵从数据类型		
OutEle	结果数组元素		结果数组开始元素	遵从数据类型		

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ]排列	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Size							○													
Order	列举型 _eBYTE_ORDER 列举值参考功能说明																			
OutVal		○																		
OutEle		○																		

功能

将转换对象排列 In[]的起始“Size”个要素，根据一字节为单位分割，将结果保存到“OutEle”元素开始的 BYTE 数组“OutVal”中。

将 In[]的值以一字节为单位转换时的顺序，在转换顺序 “Order” 中指定。“Order”的数据类型为列举型 _eBYTE_ORDER。列举值的含义如下所示

列举值	含义
_LOW_HIGH	先低位字节，后高位字节
_HIGH_LOW	先高位字节，后低位字节

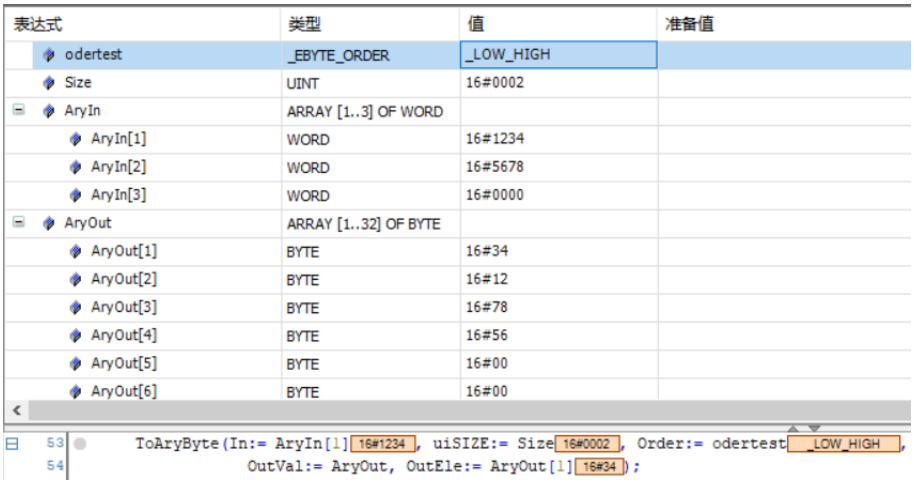
“In”的元素数据类型为 2 字节以上时，处理步骤如下：

1.讲“In”的值以 2 字节为分割单位，划分出来的两字节在以一字节为分割单位生成成高低位字节。

2. 若“Order”的值为_LOW_HIGH，以先低字节再高字节的顺序方式保存到“OutEle”元素开始的“OutVal”中。若“Order”的值为_HIGH_LOW，以先高字节再低字节的顺序方式保存到“OutEle”元素开始的“OutVal”中。

3.直到使用了“Size”个元素数据，将数据依次保存在“OutVal”中。

“In”为 WORD 型数组的第 1 个元素、Size =UINT#2、Qrder =_LOW_HIGH、“OutEle”为“OutVal”数组第 1 个元素时，示例如下：



“In”为 WORD 型数组的第 2 个元素、Size =UINT#2、Qrder = _HIGH_LOW、“OutEle”为“OutVal”数组第 2 个元素时，示例如下：

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_HIGH_LOW	
Size	UINT	16#0002	
AryIn	ARRAY [1..3] OF WORD		
AryIn[1]	WORD	16#1010	
AryIn[2]	WORD	16#2345	
AryIn[3]	WORD	16#6789	
AryOut	ARRAY [1..32] OF BYTE		
AryOut[1]	BYTE	16#00	
AryOut[2]	BYTE	16#23	
AryOut[3]	BYTE	16#45	
AryOut[4]	BYTE	16#67	
AryOut[5]	BYTE	16#89	
AryOut[6]	BYTE	16#00	

```
53    ToAryByte(In:= AryIn[2] 16#2345 , uiSIZE:= Size 16#0002 , Order:= odertest _HIGH_LOW ,
54              OutVal:= AryOut, OutEle:= AryOut[2] 16#23 );
```

“OutEle”的数据类型为 1 字节时，示例如下：

“In”为 SINT 型数组的第 1 个元素、Size =UINT#3、Qrder =_LOW_HIGH、“OutEle”为“OutVal”数组第 1 个元素时，示例如下：

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_LOW_HIGH	
Size	UINT	16#0003	
AryIn	ARRAY [1..4] OF SINT		
AryIn[1]	SINT	16#11	
AryIn[2]	SINT	16#22	
AryIn[3]	SINT	16#33	
AryIn[4]	SINT	16#00	
AryOut	ARRAY [1..32] OF BYTE		
AryOut[1]	BYTE	16#11	
AryOut[2]	BYTE	16#22	
AryOut[3]	BYTE	16#33	
AryOut[4]	BYTE	16#00	
AryOut[5]	BYTE	16#00	

```
53    ToAryByte(In:= AryIn[1] 16#11 , uiSIZE:= Size 16#0003 , Order:= odertest _LOW_HIGH ,
54              OutVal:= AryOut, OutEle:= AryOut[1] 16#11 );
```

其他条件与上述描述相同，Qrder =_HIGH_LOW，示例如下：

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_HIGH_LOW	
Size	UINT	16#0003	
AryIn	ARRAY [1..4] OF SINT		
AryIn[1]	SINT	16#11	
AryIn[2]	SINT	16#22	
AryIn[3]	SINT	16#33	
AryIn[4]	SINT	16#00	
AryOut	ARRAY [1..32] OF BYTE		
AryOut[1]	BYTE	16#22	
AryOut[2]	BYTE	16#11	
AryOut[3]	BYTE	16#00	
AryOut[4]	BYTE	16#33	
AryOut[5]	BYTE	16#00	

```
53    ToAryByte(In:= AryIn[1] 16#11 , uiSIZE:= Size 16#0003 , Order:= odertest _HIGH_LOW ,
54              OutVal:= AryOut, OutEle:= AryOut[1] 16#22 );
```

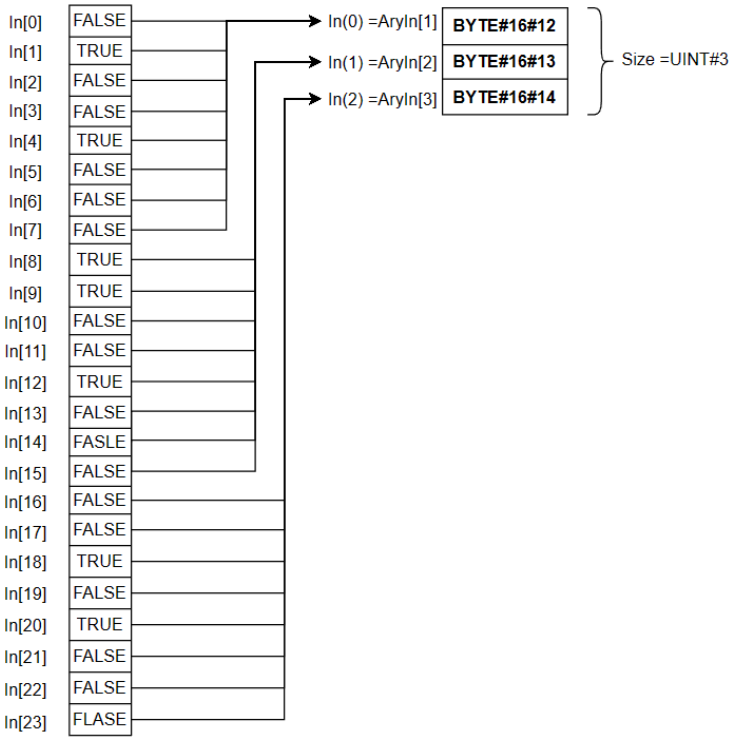
“OutEle”的数据类型为 BOOL 时，示例如下：

“OutVal”为 BOOL 型数组、Size =UINT#3、 “In”为 BYTE 数组的第 1 个元素、Qrder =_LOW_HIGH、

“OutEle”为“OutVal”数组第 1 个元素时，示例如下：

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_LOW_HIGH	
Size	UINT	16#0003	
AryOut	ARRAY [1..5] OF BYTE		
AryOut[1]	BYTE	16#12	
AryOut[2]	BYTE	16#13	
AryOut[3]	BYTE	16#14	
AryOut[4]	BYTE	16#00	
AryOut[5]	BYTE	16#00	
AryIn	ARRAY [1..32] OF BOOL		
AryIn[1]	BOOL	FALSE	
AryIn[2]	BOOL	TRUE	
AryIn[3]	BOOL	FALSE	
AryIn[4]	BOOL	FALSE	
AryIn[5]	BOOL	TRUE	
AryIn[6]	BOOL	FALSE	
AryIn[7]	BOOL	FALSE	
AryIn[8]	BOOL	FALSE	
AryIn[9]	BOOL	TRUE	
AryIn[10]	BOOL	TRUE	
AryIn[11]	BOOL	FALSE	

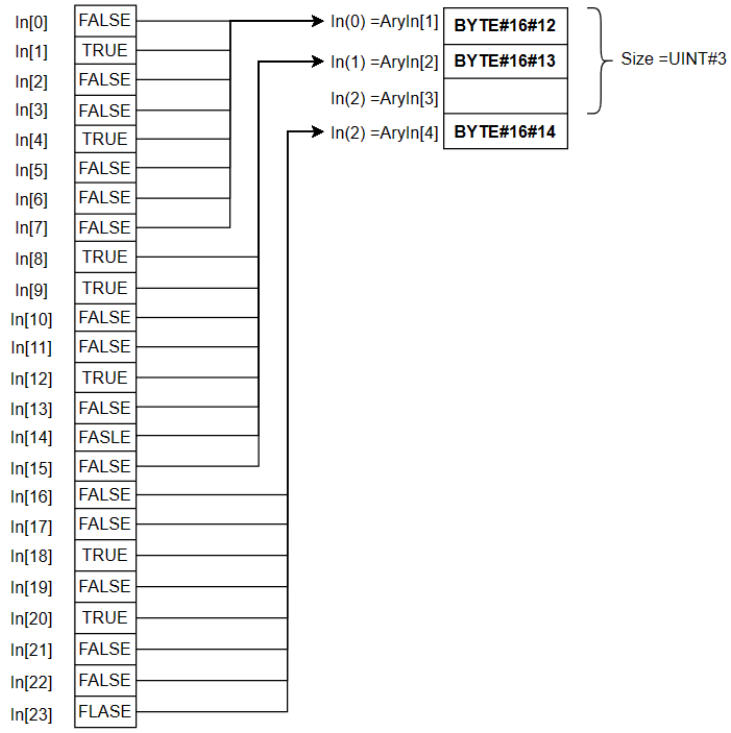
```
53    ToAryByte(In:= AryIn[1] FALSE , uiSIZE:= Size 16#0003 , Order:= odertest _LOW_HIGH ,
54              OutVal:= AryOut, OutEle:= AryOut[1] 16#12 );
```



其他条件与上述描述相同，Qrder =_HIGH_LOW，示例如下：

表达式	类型	值	准备值
odertest	_EBYTE_ORDER	_HIGH_LOW	
Size	UINT	16#0003	
AryOut	ARRAY [1..5] OF BYTE		
AryOut[1]	BYTE	16#13	
AryOut[2]	BYTE	16#12	
AryOut[3]	BYTE	16#00	
AryOut[4]	BYTE	16#14	
AryOut[5]	BYTE	16#00	
AryIn	ARRAY [1..32] OF BOOL		
AryIn[1]	BOOL	FALSE	
AryIn[2]	BOOL	TRUE	
AryIn[3]	BOOL	FALSE	
AryIn[4]	BOOL	FALSE	
AryIn[5]	BOOL	TRUE	
AryIn[6]	BOOL	FALSE	
AryIn[7]	BOOL	FALSE	
AryIn[8]	BOOL	FALSE	
AryIn[9]	BOOL	TRUE	
AryIn[10]	BOOL	TRUE	
AryIn[11]	BOOL	FALSE	

```
53    ToAryByte(In:= AryIn[1]FALSE, uiSIZE:= Size16#0003, Order:= odertest_HIGH_LOW,
54                  OutVal:= AryOut, OutEle:= AryOut[1]16#13);
```



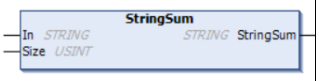
要点说明

- “In”输入起始操作的数组元素
- “OutVal”填写 BYTE 数组，“OutEle”填写起始操作数组元素
- 输出数组内存不够转换时，函数返回状态 FALSE，不做任何处理。

6.10. FSC 指令

6.10.1. StringSum (SUM 值计算)

计算字符串的和值。

指令	名称	FB/FUN	图形表现	ST 表现
StringSum	和值计算	FUN		Out:=StringSum(In, Size);

变量

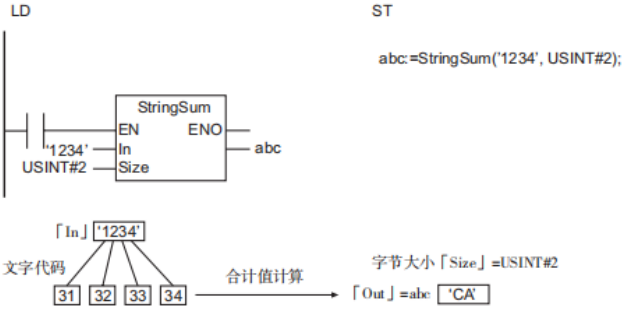
	名称	输入/输出	内容	有效范围	单位	初始值
In	对象字符串	输入	对象字符串	遵从数据类型	—	‘’
Size	字节大小		和值的字节大小	1,2	字节	1
Out	和值	输出	和值	“Size”显示的字节数	字节	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Size						○														
Out																				○

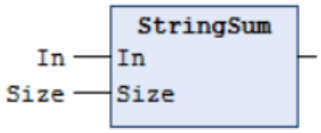
功能

计算对象字符串“In”的和值(各字符的文字代码的总和)。和值“Out”为字节大小“Size”指定的字节数。“Out” 以 16 进制的字符串表现，结尾保存 NULL 字符。

“In”='1234'，“Size”=USINT#2 时的示例如下所示。



上例中，如果 “Size”=USINT#1，则 “Out” 的值为'A'。

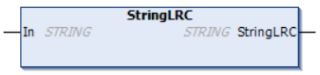
	FBD	ST
定义变量	VAR In :STRING; Size :USINT; Out :STRING; END_VAR	
程序		Out:=StringSum(In:=In , Size:=Size);
运行结果		

要点说明

- “In” 的文字代码的总和超过 “Size” 可表现的位数时，舍去高位。
- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- “Size” 的值超过有效范围时。
- “In” 未以 NULL 字符结尾时。
- “In” 的字节数为 0(仅 NULL 字符) 时。
- 处理结果的字符串大小超过 “Out” 的大小时。

6. 10. 2. StringLRC (LRC 值计算<字符串>)

计算字符串的 LRC 值 (水平奇偶校验)

指令	名称	FB/FUN	图形表现	ST 表现
StringLRC	LRC 值计算（字符串）	FUN		Out:=StringLRC(In);

变量

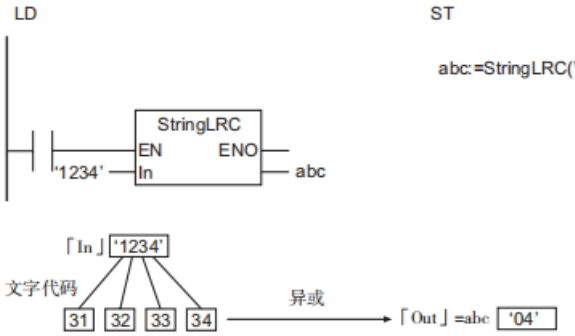
	名称	输入/输出	内容	有效范围	单位	初始值
In	对象字符串	输入	对象字符串	遵从数据类型	—	‘’
Out	LRC 值	输出	LRC 值	最大 3 字节（2 个半角英数字字符+结尾 NULL 字符）	—	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out																				○

功能

计算对象字符串 “In” 的 LRC 值（水平奇偶校验）。LRC 值是指 “In” 的各字符的文字代码的异或。LRC 值 “Out” 以 16 进制的字符串表现，结尾保存 NULL 字符。

“In”='1234' 时的示例如下所示。



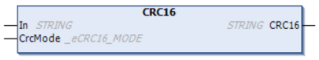
	FBD	ST
定义变量	VAR In :STRING; Out :STRING; END_VAR	
程序	In — StringLRC In	Out:=StringLRC(In:=In);
运行结果	In STRING '1234' Out STRING '04'	

要点说明

- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- “In” 未以 NULL 字符结尾时。
- “In” 的字节数为 0（仅 NULL 字符）时。
- “Out” 的字节数超过有效范围时。

6.10.3. CRC16 (CRC16 通用功能块<字符串>)

计算字符串的 CRC16 值（ 循环冗余校验 ）

指令	名称	FB/FUN	图形表现	ST 表现
CRC16	CRC 值计算（字符串）	FUN		Out:=CRC16(In，CrcMode);

变量

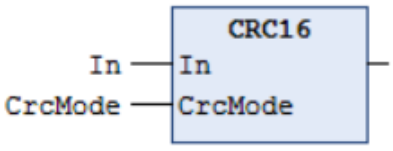
	名称	输入/输出	内容	有效范围	单位	初始值
In	对象字符串	输入	对象字符串	遵从数据类型	—	''
CrcMode	CRC 模型	输入	_eCRC16_MODE	—	—	_CRC16_CCITT
Out	CRC 值	输出	CRC 值	最大 5 字节（4 个半角英数字字符+结尾 NULL 字符）	—	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
CrcMode	枚举体_eCRC16_MODE																			
Out																				○

功能

计算对象字符串 “In” 的 CRC 值（ 循环冗余校验)。CRC 值是指 “In” 的各字符通过 CRC 模型进行校验后得到的校验码。CRC 值“Out”以 16 进制的字符串表现，结尾保存 NULL 字符。

“In”= ‘12’，“CrcMode” =—CRC16_MODBUS 时，示例如下。

	FBD	ST
定义变量	VAR In :STRING; CrcMode :_eCRC16_MODE; Out :STRING; END_VAR	
程序		Out:=CRC16(In:=In , CrcMode:=CrcMode);

运行结果	In	STRING	'12'
	CrcMode	_ECRC16_MODE	_CRC16_MODBUS
	Out	STRING	'F595'

枚举元素	含义
_CRC16_CCITT	CCITT 校验
_CRC16_CCITT_FALSE	CCITT_FALSE 校验
_CRC16_XMODEM	XMODEM 校验
_CRC16_X25	X25 校验
_CRC16_MODBUS	MODBUS 校验
_CRC16_IBM	IBM 校验
_CRC16_MAXIM	MAXIM 校验
_CRC16_USB	USB 校验

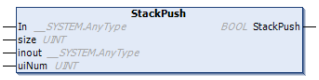
要点说明

- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- “In” 未以 NULL 字符结尾时。
- “In” 的字节数为 0(仅 NULL 字符) 时。

6. 11. 堆叠/表格指令

6. 11. 1. StackPush (保存堆叠数据)

将值保存至堆栈中。

指令	名称	FB/FUN	图形表现	ST 表现
StackPush	堆栈数据保存	FUN		StackPush(In, InOut, Size, Num);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	输入值	输入	输入至堆栈的值、结构体、结构体的 1 个结构要素	遵从数据类型	—	‘’
Size	堆栈的元素数		堆栈的数组元素数			1
InOut[ ]数组	堆栈数组		构成堆栈的数组	遵从数据类型	—	—
uiNum	堆栈的保存元素数	输入输出	保存在堆栈中个元素数			
Out	返回值	输出	始终为 TRUE	仅 TRUE	—	—

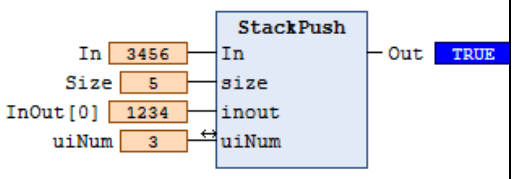
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	也可指定枚举体、整个结构体、结构体的 1 个结构要素																			
Size							○													
InOut[ ]数组	将与 “In” 相同的数据类型作为元素的数组																			
uiNum							○													
Out	○																			

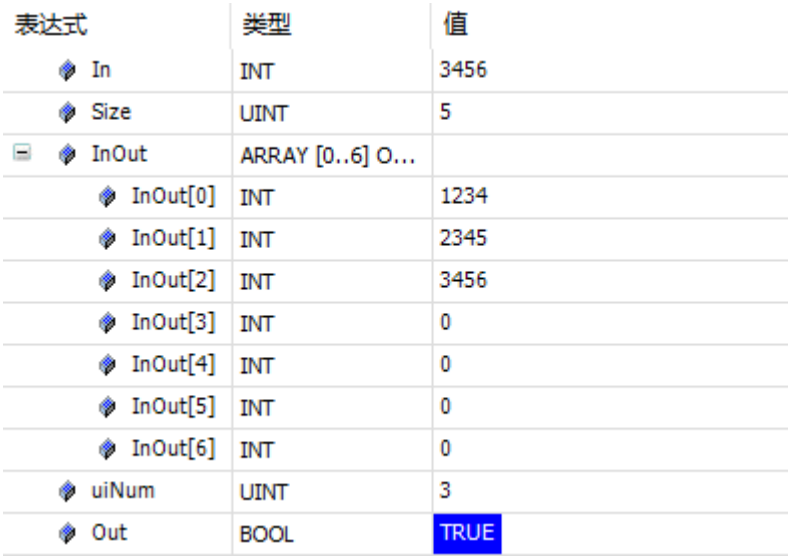
功能

视为当前已将保存元素数 “Num” 个元素保存至堆栈数组 InOut[ ]。将输入值 “In” 覆盖下一元素 InOut[“Num”]。然后，对 “Num” 进行增量。在堆栈的元素数 “Size” 中指定 InOut[ ] 中用作堆栈的元素数。

“Size”=UINT#5，“Num”=UINT#2 时的示例如下所示。

	FBD	ST
定义变量	PROGRAM PLC_PRG VAR In :INT; Size :UINT :=5; InOut :ARRAY[0..6] OF INT := [1234, 2345, 5(0)]; uiNum :UINT :=2; Out :BOOL; check :BOOL; END_VAR	

程序		IF check FALSE THEN
		Out TRUE :=StackPush(
		In:= In 3456 ,
		size:= Size 5 ,
		inout:= InOut[0] 1234 ,
		uiNum:=uiNum 3);
		check FALSE :=FALSE;
		END_IF RETURN

运行结果	

参考

抽取堆栈的最低位或最高位的值时，请使用 “StackFIFO/StackLIFO 指令”。

要点说明

- 请将 “In” 和 InOut[ ] 的元素的数据类型设为相同。
- 将数组中的元素传输至 InOut[ ] 时，该元素之后为处理对象。
- “Size” 的值为 0 时，InOut[ ]、“Num” 的值不变。
- 请务必将传输至 “In” 的输入参数设为变量。如果传输常数，编连时会发生异常。
- “In” 为枚举体时，无法直接传输枚举元素。如果直接传输枚举元素，编连时会发生异常。
- 在 ST 程序中使用本指令时，不使用返回值 “Out”。
- 以下情况时会发生异常。ENO 为 FALSE，InOut[ ] 不变。
- “In” 和 InOut[ ] 的元素的数据类型不同时。
- “Size” 的值不为 0，“Num” ≥ “Size” 时。
- “Size” 的值超过 InOut[ ] 的数组区域时。
- “In” 为 STRING 型，且未以 NULL 字符结尾时。
- “In”、InOut[ ] 为 STRING 型，“In” 的字节数超过 InOut[ ] 的大小时。

6. 11. 2. StackFIFO/StackLIFO (先入先出/后入先出)

StackFIFO：提取堆栈最低位的值。

StackLIFO：提取堆栈最高位的值。

指令	名称	FB/FUN	图形表现	ST 表现
StackFIFO	先入先出	FUN		StackFIFO(InOut, OutVal, Size, Num);
StackLIFO	后入先出	FUN		StackLIFO(InOut, OutVal, Size, Num);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Size	堆栈的元素数	输入	堆栈的数值元素	遵从数据类型	—	1
InOut[ ]数组	堆栈数组		构成堆栈的数值			—
OutVal	输出值		从堆栈输出的值、整个结构体	遵从数据类型	—	—
Num	堆栈的报错元素数	输入输出	保存在堆栈中的元素数			
Out	返回值	输出	始终为 TRUE	仅 TRUE	—	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							○													
InOut[ ]数组	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	也可指定以枚举体为元素的数组、以结构体为元素的数组																			
OutVal	与 InOut[ ] 的元素相同的数据类型																			
Num							○													
Out	○																			

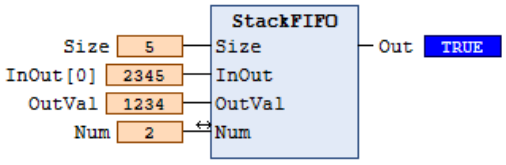
功能

视为当前已将保存元素数“Num”个元素保存至堆栈数组 InOut[ ]。从该处提取值，代入输出值“OutVal”。在堆栈的元素数 “Size” 中指定 InOut[ ] 中用作堆栈的元素数。

StackFIFO：提取堆栈最低位的值。将 InOut[0] 的值代入 “OutVal”。然后，将 InOut[1] 之后的“Num”-1 个元素向堆栈数组的低位方向各移 1 位。最后，对 “Num” 进行减量。

“Size”=UINT#5，“Num”=UINT#3 时的示例如下所示。

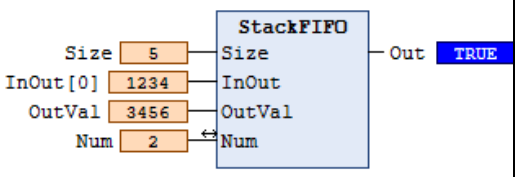
	FBD	ST
定义变量	PROGRAM PLC_PRG VAR Size :UINT :=5; InOut :ARRAY[0..6] OF INT := [1234, 2345,3456,4(0)]; OutVal :INT; Num :UINT :=3; Out :BOOL; check :BOOL; END_VAR	

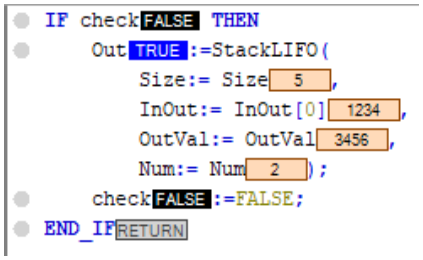
程序		IF check FALSE THEN
		Out TRUE :=StackFIFO(
		Size:= Size 5 ,
		InOut:= InOut[0] 2345 ,
		OutVal:= OutVal 1234 ,
		Num:= Num 2);
		check FALSE :=FALSE;
		END_IF RETURN

运行结果	表达式 / 类型 / 值 Size / UINT / 5 InOut / ARRAY [0..6] O... InOut[0] / INT / 2345 InOut[1] / INT / 3456 InOut[2] / INT / 0 InOut[3] / INT / 0 InOut[4] / INT / 0 InOut[5] / INT / 0 InOut[6] / INT / 0 OutVal / INT / 1234 Num / UINT / 2 Out / BOOL / TRUE	

StackLIFO：提取堆栈最高位的值。将 InOut[“Num”-1] 的值代入 “OutVal”。对 “Num” 进行减量。“Size”=UINT#5，“Num”=UINT#3 时的示例如下所示。

	FBD	ST
定义变量	PROGRAM PLC_PRG VAR Size :UINT :=5; InOut :ARRAY[0..6] OF INT := [1234, 2345,3456,4(0)]; OutVal :INT; Num :UINT :=3; Out :BOOL; check :BOOL; END_VAR	

<table>
<tr>
<td>程序</td>
<td> 


</td>
</tr>
<tr>
<td>运行结果</td>
<td>
<table>
<tr><th>表达式</th><th>类型</th><th>值</th></tr>
<tr><td>Size</td><td>UINT</td><td>5</td></tr>
<tr><td>InOut</td><td>ARRAY [0..6] O...</td><td></td></tr>
<tr><td>InOut[0]</td><td>INT</td><td>1234</td></tr>
<tr><td>InOut[1]</td><td>INT</td><td>2345</td></tr>
<tr><td>InOut[2]</td><td>INT</td><td>0</td></tr>
<tr><td>InOut[3]</td><td>INT</td><td>0</td></tr>
<tr><td>InOut[4]</td><td>INT</td><td>0</td></tr>
<tr><td>InOut[5]</td><td>INT</td><td>0</td></tr>
<tr><td>InOut[6]</td><td>INT</td><td>0</td></tr>
<tr><td>OutVal</td><td>INT</td><td>3456</td></tr>
<tr><td>Num</td><td>UINT</td><td>2</td></tr>
<tr><td>Out</td><td>BOOL</td><td>TRUE</td></tr>
</table>
</td>
</tr>
</table>

要点说明

- 请将 “In” 和 InOut[ ] 的元素的数据类型设为相同。
- 将数组中的元素传输至 InOut[ ] 时，该元素之后为处理对象。
- “Size” 的值为 0 时，InOut[ ]、“Num” 的值不变。
- 在 ST 程序中使用本指令时，不使用返回值 “Out”。
- 以下情况时会发生异常。ENO 为 FALSE，InOut[ ] 不变。
- InOut[ ] 的元素和 “OutVal” 的数据类型不同时。
- “Num” 和 “Size” 的值不为 0，“Num” ＞ “Size” 时。
- “Size” 的值超过 InOut[ ] 的数组区域时。
- InOut[ ] 为 STRING 型数组，所有元素均未以 NULL 字符结尾时。
- InOut[ ] 为 STRING 型数组，元素的字节数超过 “OutVal” 的大小时。

6.11.3. StackIns (插入堆叠数据)

将值插入堆栈的任意位置

指令	名称	FB/FUN	图形表现	ST 表现
StackIns	堆栈数据插入	FUN		StackIns(In, InOut, Size, Num, Offset);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	插入值	输入	插入堆栈的值、整个结构体、结构体的 1 个结构要素	遵从数据类型	—	(*)
Size	堆栈的元素数		独占的数值元素数			1
OffSet	偏置位置		插入“In”的堆栈的偏置位置			0
InOut[ ]数组	堆栈数组		构成堆栈的数组	遵从数据类型	—	—
Num	堆栈的保存元素数	输入输出	保存在堆栈中的元素	遵从数据类型	—	—
Out	返回值	输出	始终为 TRUE	仅 TRUE	—	—

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	也可指定枚举体、整个结构体、结构体的 1 个结构要素																			
Size							○													
OffSet							○													
InOut[ ]数组	将与 “In” 相同的数据类型作为元素的数组																			
Num							○													
Out	○																			

功能

视为当前已将保存元素数“Num”个元素保存至堆栈数组 InOut[ ]。将插入值“In”插入由偏置位置“Offset” 确定的位置 InOut[“Offset”]。将更高位的元素即 InOut[“Offset”] ～ InOut[“Num”-

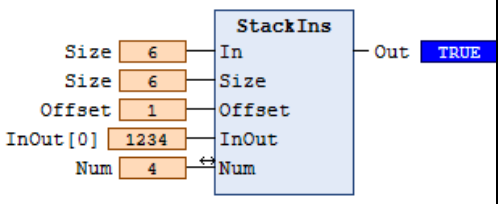
1] 向堆栈数组的高位方向各移 1 位。然后，对“Num”进行增量。在堆栈的元素数“Size”中指定 InOut[ ] 中用作堆栈的元素数。

“Size”=UINT#6，“Num”=UINT#3，“Offset”=UINT#1 时的示例如下所示。

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">

```
PROGRAM PLC_PRG
VAR
    In      :INT;
    Size    :UINT :=6;
    Offset  :UINT :=1;
    InOut   :ARRAY[0..6] OF INT := [1234, 3456,4567,4(0)];
    Num     :UINT :=3;
    Out     :BOOL;
    check   :BOOL;
END_VAR
```

</td></tr>
<tr><td>程序</td><td>



</td><td>

```
IF check FALSE THEN
    Out TRUE :=StackIns(
        In:= In 2345,
        Size:= Size 6,
        Offset:= Offset 1,
        InOut:= InOut[0] 1234,
        Num:= Num 4);
        check FALSE :=FALSE;
END_IF RETURN
```

</td></tr>
<tr><td>运行结果</td><td colspan="2">

表达式	类型	值
In	INT	2345
Size	UINT	6
Offset	UINT	1
InOut	ARRAY [0..6] O...	
InOut[0]	INT	1234
InOut[1]	INT	2345
InOut[2]	INT	3456
InOut[3]	INT	4567
InOut[4]	INT	0
InOut[5]	INT	0
InOut[6]	INT	0
Num	UINT	4
Out	BOOL	TRUE

</td></tr>
</table>

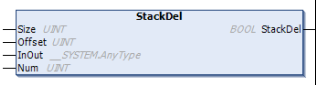
要点说明

- 请将 “In” 和 InOut[ ] 的数据类型设为相同。
- 将数组中的元素传输至 InOut[ ] 时，该元素之后为处理对象。
- “Size” 的值为 0 时，InOut[ ]、“Num” 的值不变。
- 请务必将传输至 “In” 的输入参数设为变量。如果传输常数，编连时会发生异常。

- “In” 为枚举体时，无法直接传输枚举元素。如果直接传输枚举元素，编连时会发生异常。
- 在 ST 程序中使用本指令时，不使用返回值 “Out”。
- 以下情况时会发生异常。ENO 为 FALSE，InOut[ ] 不变。
- “In” 和 InOut[ ] 的元素的数据类型不同时。
- “Size” 的值不为 0，不满足 “Size” > “Num” ≥ “Offset” 时。
- “Size” 的值超过 InOut[ ] 的数组区域时。
- “In” 为 STRING 型，且未以 NULL 字符结尾时。
- “In”、InOut[ ] 为 STRING 型，“In” 的字节数超过 InOut[ ] 的大小时。

6.11.4. StackDel (删除堆叠数据)

删除堆栈任意位置的值

指令	名称	FB/FUN	图形表现	ST 表现
StackDel	堆栈数据删除	FUN		StackDel(InOut, Size, Num, Offset);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Size	堆栈的元素数	输入	堆栈的数值元素数	遵从数据类型	—	1
OffSet	偏置位置		待删除的堆栈的偏置位置			0
InOut[ ]数组	堆栈数组		构成堆栈的数组	遵从数据类型	—	—
Num	堆栈的保存元素数	输入输出	保存在堆栈中的元素	遵从数据类型	—	—
Out	返回值	输出	始终为 TRUE	仅 TRUE	—	—

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							○													
OffSet							○													
InOut[ ]数组	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	也可指定以枚举体为元素的数组、以结构体为元素的数组																			
Num							○													
Out	○																			

功能

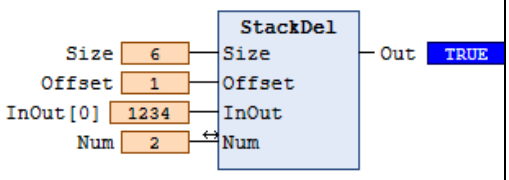
视为当前已将保存元素数 “Num” 个元素保存至堆栈数组 InOut[ ]。删除由偏置位置 “Offset” 确定的位置 InOut[“Offset”] 的值。将更高位的元素即 InOut[“Offset”+1] ～ InOut[“Num”-1] 向堆栈数组的低位方向各移 1 位。然后，对 “Num” 进行减量。在堆栈的元素数 “Size” 中指定 InOut[ ] 中用作堆栈的元素数。

“Size”=UINT#6，“Num”=UINT#3，“Offset”=UINT#1 时的示例如下所示（InOut 数组初始值见示例）。

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">

```
PROGRAM PLC_PRG
VAR
    Size    :UINT :=6;
    Offset  :UINT :=1;
    InOut   :ARRAY[0..6] OF INT := [1234, 2345,3456,4(0)];
    Num     :UINT :=3;
    Out     :BOOL;
    check   :BOOL;
END_VAR
```

</td></tr>
<tr><td>程序</td><td>



</td><td>

```
IF check FALSE THEN
    Out TRUE :=StackDel(
        Size:= Size 6,
        Offset:= Offset 1,
        InOut:= InOut[0] 1234,
        Num:= Num 2);
    check FALSE :=FALSE;
END_IF RETURN
```

</td></tr>
<tr><td>运行结果</td><td colspan="2">

表达式	类型	值
Size	UINT	6
Offset	UINT	1
InOut	ARRAY [0..6] O...	
InOut[0]	INT	1234
InOut[1]	INT	3456
InOut[2]	INT	0
InOut[3]	INT	0
InOut[4]	INT	0
InOut[5]	INT	0
InOut[6]	INT	0
Num	UINT	2
Out	BOOL	TRUE

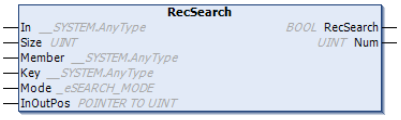
</td></tr>
</table>

要点说明

- 将数组中的元素传输至 InOut[ ] 时，该元素之后为处理对象。
- “Size” 或 “Num” 的值为 0 时，InOut[ ]、“Num” 的值不变。
- 在 ST 程序中使用本指令时，不使用返回值 “Out”。
- 以下情况时会发生异常。ENO 为 FALSE，InOut[ ] 不变。
- “Num” 和 “Size” 的值不为 0，不满足 “Size” ≥ “Num”>“Offset” 时。

6.11.5. RecSearch (记录检索)

在以结构体为元素的数组中，按指定方法检索与检索关键词一致的要素。

指令	名称	FB/FUN	图形表现	ST 表现
RecSearch	记录检索	FUN		Out:=RecSearch(In, Size, Member, Key, Mode, InOutPos, Num);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In[ ]数组	检索对象数组	输入	作为检索对象的以结构体为元素的数组	—	—	(*)
Size	检索对象的元素数		检索对象的数组元素数	遵从数据类型		1
Member	检索对象结构要素		In[ ]的结构体的检索对象结构要素			(*)
Key	检索关键词		检索值			
Mode	检索方法		检索方法	_LINEAR, _BIN_ASC, _BIN_DESC		_LINEAR
InOutPos[ ]数组	一致元素的元素编号	输入输出	一致元素的元素编号	遵从数据类型	—	—
Out	检索结果	输出	TRUE ：有一致的元素 FALSE：无一致的元素	遵从数据类型	—	—
Num	一致元素的个数		一致元素的个数		—	—

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ]数组	指定以结构体为元素的数组																			
Size							○													
Member						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	与 In[ ] 的检索对象结构要素相同的数据类型																			
Key	与 “Member” 相同的数据类型																			
Mode	枚举体 _eSEARCH_MODE 枚举元素参阅功能说明																			
InOutPos[ ] 数组							○													
Out	○																			
Num							○													

功能

在以结构体为元素的数组 In[ ] 的 “Size” 个元素即 In[0] ～ In[“Size”-1] 中，检索结构体的检索对象结构要素“Member” 的值与检索关键词 “Key” 一致的内容。将 In[ ] 的任一元素的检索对象结构要素作为自变量传输至“Member”。如果存在一致的元素，则检索结果 “Out” 的值变为 TRUE。将一致元素的元素编号、一致元素数分别代入 InOutPos[0]、“Num”。有 2 个以上一致元素时，将 In[ ]中最低位的一致元素的元素编号代入 InOutPos[0]。如果不存在一致元素，则 “Out” 的值变为 FALSE，InOutPos[0] 和 “Num” 变为 0。传输至 In[ ] 的输入参数务必也像 array[3] 那样加上元素编号后再指定。

检索方法 “Mode” 的数据类型为枚举体 _eSEARCH_MODE。枚举元素的含义如下所示。

枚举元素	含义
_LINEAR	线性检索
_BIN_ASC	升序二分检索
_BIN_DESC	降序二分检索

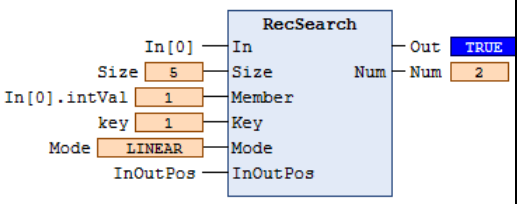
线性检索时，从 In[ ] 的首个元素起依次进行检索。

“Size”=UINT#5，“Key”=INT#1，“Mode”=_LINEAR 的示例如下所示。示例中查找到 In[0]和 In[3]的 Int 元素值与“Key”相同。

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">

```
PROGRAM PLC_PRG
VAR
    In    :ARRAY[0..4] OF str:= [(intVal := 1, realVal := 2.3),
                                 (intVal := 5, realVal := 536),
                                 (intVal := 2, realVal := 5.6),
                                 (intVal := 1, realVal := 6.3),
                                 (intVal := 8, realVal := 55)];
    Size     :UINT :=5;
    key      :INT:=1;
    Mode     :HCFA_OmronUitls._eSEARCH_MODE;
    InOutPos:ARRAY [0..4] OF UINT;
    Num      :UINT;
    Out      :BOOL;
    check    :BOOL;
END_VAR
```

</td></tr>
<tr><td>程序</td><td>



</td><td>

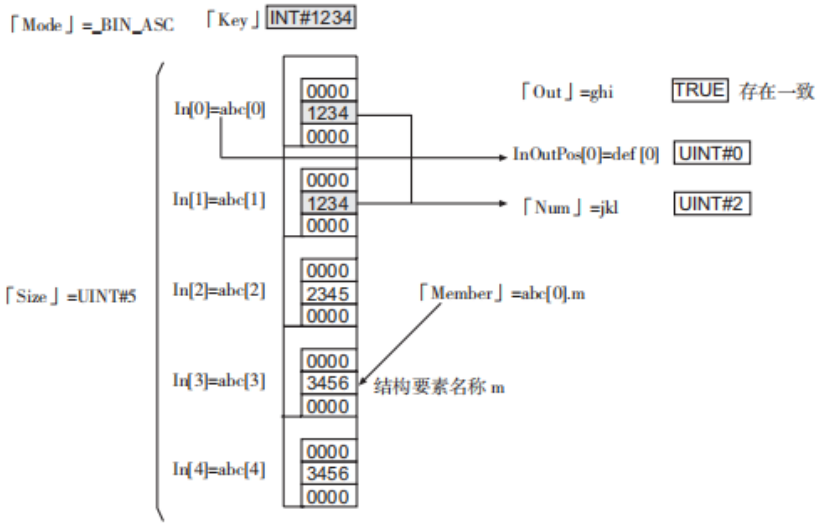
```
IF checkFALSE THEN
    OutTRUE :=RecSearch(
        In:= In[0],
        Size:= Size5,
        Member:= In[0].intVal1,
        Key:= key1,
        Mode:= Mode_LINEAR,
        InOutPos:= InOutPos,
        Num=> Num2 );
        checkFALSE :=FALSE;
END_IF
```

</td></tr>
</table>

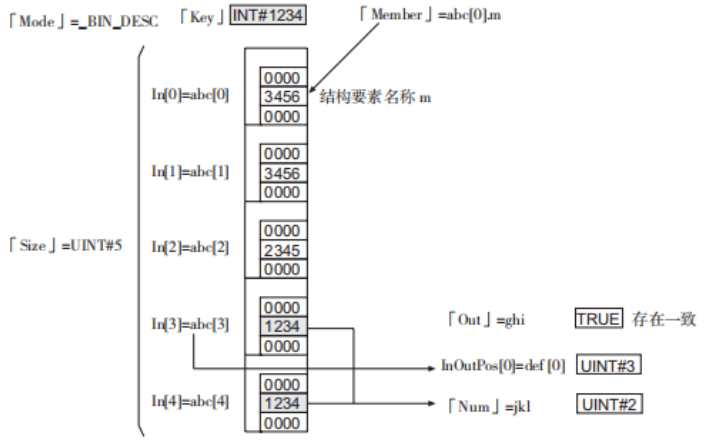
运行结果

表达式	类型	值
In	ARRAY [0..4] O...	
In[0]	str	
intVal	INT	1
realVal	REAL	2.3
In[1]	str	
intVal	INT	5
realVal	REAL	536
In[2]	str	
intVal	INT	2
realVal	REAL	5.6
In[3]	str	
intVal	INT	1
realVal	REAL	6.3
In[4]	str	
intVal	INT	8
realVal	REAL	55
Size	UINT	5
key	INT	1
Mode	_ESEARCH_MODE	_LINEAR
InOutPos	ARRAY [0..4] O...	
InOutPos[0]	UINT	0
InOutPos[1]	UINT	3
InOutPos[2]	UINT	0
InOutPos[3]	UINT	0
InOutPos[4]	UINT	0
Num	UINT	2
Out	BOOL	TRUE

升序二分检索时，执行本指令前需事先将传输至 In[ ] 的输入参数的数组元素进行升序排列。再通过执行本指令进行二分检索。对与上述相同的示例进行升序二分检索后，数组元素的顺序和处理结果如下所示。



降序二分检索时，执行本指令前需事先将传输至 In[ ] 的输入参数的数组元素进行降序排列。再通过执行本指令进行二分检索。对与上述相同的示例进行降序二分检索后，数组元素的顺序和处理结果如下所示。

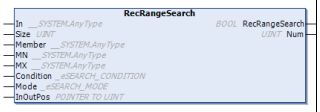


要点说明

- 将数组中的元素传输至 In[ ] 时，该元素之后为处理对象。
- “Member” 为实数时，会因数值而产生误差，因此结果可能会出现意外。
- “Key” 为实数时，“Key” 请勿指定为非数。
- “Size” 的值为 0 时，“Out” 的值为 FALSE，“Num” 的值为 0。InOutPos[ ] 不变。
- “Mode”的值为_BIN_ASC(或_BIN_DESC)时，如果 In[ ]的元素未按升序(或降序)排列，则无法获得正确
- 结果。执行本指令前，请将元素升序（ 或降序 ）排列。
- 以下情况时会发生异常。ENO 变为 FALSE，“Out”、InOutPos[ ]、“Num” 不变。
- “Size” 的值超过 In[ ] 的数组区域时。
- “Member” 不是 In[ ] 的结构要素时。
- 不是 “Member” 支持的数据类型时。
- “Key” 和 “Member” 的数据类型不同时。
- In[ ] 不是以结构体为元素的数组时。
- “Member”、“Key” 为 STRING 型，且未以 NULL 字符结尾时。

6. 11. 6. RecRangeSearch (指定范围记录检索)

在以结构体为元素的数组中，按指定方法检索与检索条件的范围一致的要素。

指令	名称	FB/FUN	图形表现	ST 表现
RecRange Search	范围指定记录检索	FUN		Out:=RecRangeSearch(In, Size, Member, MN, MX, Condition, Mode, InOutPos, Num);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In[ ]数组	检索对象数组	输入	作为检索对象的以结构体为元素的数组	—	—	(*)
Size	检索对象的元素数		检索对象的数组元素数	遵从数据类型		1
Member	检索对象结构要素		In[ ]的结构体的检索对象结构要素			(*)
MN	检索条件下限值		检索条件上限值			
MX	检索条件上限值		检索条件下限值			
Condition	检索条件		检索条件	_EQ_BOTH, _EQ_MIN, _EQ_MAX, _NE_BOTH		_EQ_BOTH
Mode	检索方法		检索方法	_LINEAR, _BIN_ASC, _BIN_DESC		_LINEAR
InOutPos[ ] 数组	一致元素的元素编号	输入输出	一致元素的元素编号	遵从数据类型	—	—
Out	检索结果	输出	TRUE ：有一致的元素 FALSE：无一致的元素	遵从数据类型	—	—
Num	一致元素的个数		一致元素的个数		—	—

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ]数组	指定以结构体为元素的数组																			
Size							○													
Member						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○

	与 In[ ] 的检索对象结构要素相同的数据类型																		
MN	与 “Member” 相同的数据类型																		
MX	与 “Member” 相同的数据类型																		
Condition	枚举体_eSEARCH_CONDITION-枚举体元素参阅功能说明																		
Mode	枚举体 _eSEARCH_MODE 枚举元素参阅功能说明																		
InOutPos[ ] 数组							○												
Out	○																		
Num							○												

功能

在以结构体为元素的数组 In[ ] 的 “Size” 个元素即 In[0] ～ In[“Size”-1] 中，检索结构体的检索对象结构要素“Member” 的值与检索条件一致的内容。检索条件和检索方法由 “Condition” 和 “Mode” 指定。详情将于下文阐述。将 In[ ] 的任一元素的检索对象结构要素作为自变量传输至 “Member”。如果存在与条件一致的元素，则检索结果 “Out” 的值变为 TRUE。将一致元素的元素编号、一致元素数分别代入 InOutPos[0]、“Num” 中。有 2 个以上一致元素时，将 In[ ] 中最低位的一致元素的元素编号代入 InOutPos[0]。如果不存在一致元素，则 “Out” 的值变为 FALSE，InOutPos[0] 和 “Num” 变为 0。传输至 In[ ] 的输入参数务必也像 array[3] 那样加上元素编号后再指定。

检索条件 “Condition” 的数据类型为枚举体 _eSEARCH_CONDITION。枚举元素的含义如下所示。

枚举元素	含义
_EQ_BOTH	“MN” ≤ “Member” ≤ “MX”
_EQ_MIN	“MN” ≤ “Member” ＜ “MX”
_EQ_MAX	“MN” ＜ “Member” ≤ “MX”
_NE_BOTH	“MN” ＜ “Member” ＜ “MX”

检索方法 “Mode” 的数据类型为枚举体 _eSEARCH_MODE。枚举元素的含义如下所示。

枚举元素	含义
_LINEAR	线性检索
_BIN_ASC	升序二分检索
_BIN_DESC	降序二分检索

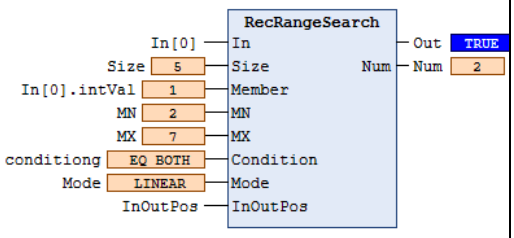
线性检索时，从 In[ ] 的首个元素起依次进行检索。

“Size”=UINT#5，“MN”=INT#2，“MX”=INT#7，“Condition”=_EQ_BOTH，“Mode”=_LINEAR 时的示例如下所示。

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">

```
PROGRAM PLC_PRG
VAR
    In    :ARRAY[0..4] OF str:= [(intVal := 1, realVal := 2.3),
                                 (intVal := 5, realVal := 536),
                                 (intVal := 2, realVal := 5.6),
                                 (intVal := 1, realVal := 6.3),
                                 (intVal := 8, realVal := 55)];
    Size          :UINT :=5;
    MN            :INT  :=2;
    MX            :INT  :=7;
    conditiong    :HCFA_OmronUitls._eSEARCH_CONDITION;
    Mode          :HCFA_OmronUitls._eSEARCH_MODE;
    InOutPos      :ARRAY [0..4] OF UINT;
    Num           :UINT;
    Out           :BOOL;
    check         :BOOL;
END_VAR
```

</td></tr>
<tr><td>程序</td><td>



</td><td>

```
IF check FALSE THEN
    Out TRUE :=RecRangeSearch(
        In:= In[0],
        Size:= Size 5 ,
        Member:= In[0].intVal 1 ,
        MN:= MN 2 ,
        MX:= MX 7 ,
        Condition:= conditiong _EQ_BOTH ,
        Mode:= Mode _LINEAR ,
        InOutPos:= InOutPos,
        Num=> Num 2 );
        check FALSE :=FALSE;
END_IF
```

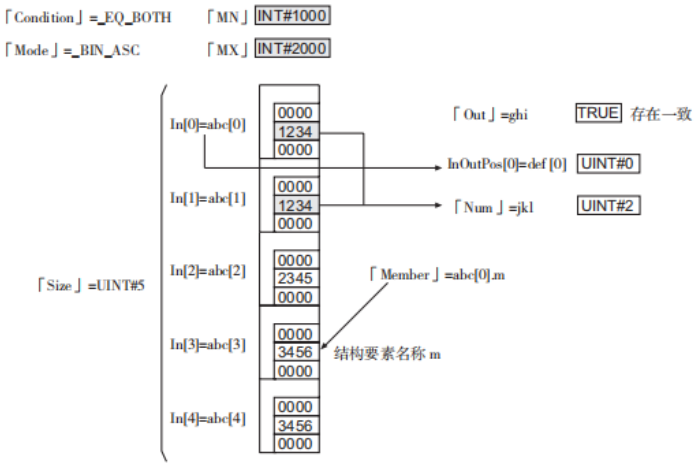
</td></tr>
<tr><td>运行结果</td><td colspan="2">

表达式	类型	值
In	ARRAY [0..4] O...	
Size	UINT	5
MN	INT	2
MX	INT	7
conditiong	_ESEARCH_CO...	_EQ_BOTH
Mode	_ESEARCH_MODE	_LINEAR
InOutPos	ARRAY [0..4] O...	
InOutPos[0]	UINT	1
InOutPos[1]	UINT	2
InOutPos[2]	UINT	0
InOutPos[3]	UINT	0
InOutPos[4]	UINT	0
Num	UINT	2
Out	BOOL	TRUE

</td></tr>
</table>

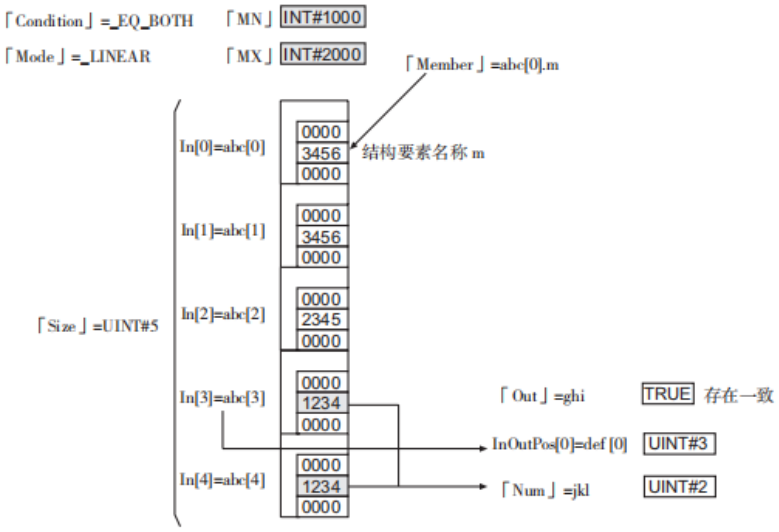
升序二分检索时，执行本指令前需事先将传输至 In[ ] 的输入参数的数组元素进行升序排列。再通过执行本指令进行二分检索。

对与上述相同的示例进行升序二分检索后，数组元素的顺序和处理结果如下所示。



降序二分检索时，执行本指令前需事先将传输至 In[ ] 的输入参数的数组元素进行降序排列。再通过执行本指令进行二分检索。

对与上述相同的示例进行降序二分检索后，数组元素的顺序和处理结果如下所示。



要点说明

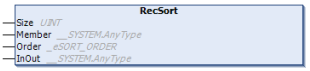
- 请将 “Member”、“MN”、“MX” 的数据类型与 In[ ] 的检索对象结构要素的数据类型设为一致。
- 将数组中的元素传输至 In[ ] 时，该元素之后为处理对象。
- “Member” 为实数时，会因数值而产生误差，因此结果可能会出现意外。
- “MN”、“MX” 为实数时，请勿指定非数。
- “Size” 的值为 0 时，“Out” 的值为 FALSE，“Num” 的值为 0。InOutPos[ ] 不变。
- “Mode”的值为_BIN_ASC(或_BIN_DESC)时，如果 In[ ]的元素未按升序(或降序)排列，则无法

获得正确

- 结果。执行本指令前，请将元素升序（ 或降序 ）排列。
- 以下情况时会发生异常。ENO 变为 FALSE， “Out”、InOutPos[ ]、 “Num” 不变。
- In[ ] 的检索对象结构要素 “Member”、 “MN”、 “MX” 的数据类型不同时。
- “MN” ＞ “MX” 时。
- “Mode” 的值超过有效范围时。
- “Member” 不是 In[ ] 的结构要素时。
- 不是 “Member” 支持的数据类型时。
- In[ ] 不是以结构体为元素的数组时。
- “Member”、 “MN”、 “MX” 为 STRING 型，且未以 NULL 字符结尾时。

6. 11. 7. RecSort (记录排序)

将以结构体为元素的数组的要素进行分类

指令	名称	FB/FUN	图形表现	ST 表现
RecSort	记录排序	FB		RecSort_instance(Execute, InOut, Size, Member, Order, Done, Busy, Error);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Size	排序对象的元素数	输入	排序对象的数值元素数	遵从数据类型	—	1
Member	排序对象结构要素		In[ ] 的结构体的排序对象结构要素			(*)
Order	排序顺序		排序顺序	_ASC, _DESC		_ASC
InOut[ ]数组	排序对象数组		作为排序对象的以结构体为元素的数组	—	—	—

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Size							○													
Member						○	○	○	○	○	○	○	○	○	○	○	○	○	○	
	与 InOut[ ] 的检索对象结构要素相同的数据类型																			
Order	枚举体 _eSORT_ORDER 枚举元素参阅功能说明																			
InOut[ ]数组	指定以结构体为元素的数组																			

功能

“Execute” 的值为 TRUE 时，按照结构体的排序对象结构要素 “Member” 的值，将以结构体为元素的数组 InOut[ ] 的 “Size” 个元素即 InOut[0] ～ InOut[「Size」-1] 进行排序。排序顺序由 “Order” 指定。详情将于下文阐述。将 In[ ] 的任一元素的排序对象结构要素作为自变量传输至 “Member”。传输至 InOut[ ] 的输入输出参数务必也像 array[3] 那样加上元素编号后再指定。

排序顺序 “Order” 的数据类型为枚举体 _eSORT_ORDER。枚举元素的含义如下所示。

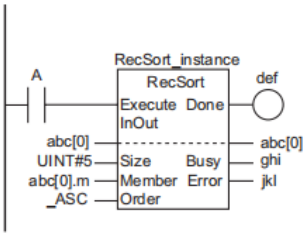
枚举元素	含义
_ASC	升序
_DESC	降序

整数、实数以外的数据类型的值的大小关系的判断如下表所示。

数据类型	含义
TIME	值较大者判断为大
DATE、TOD、DT	对于日期和时刻，较后者判断为大

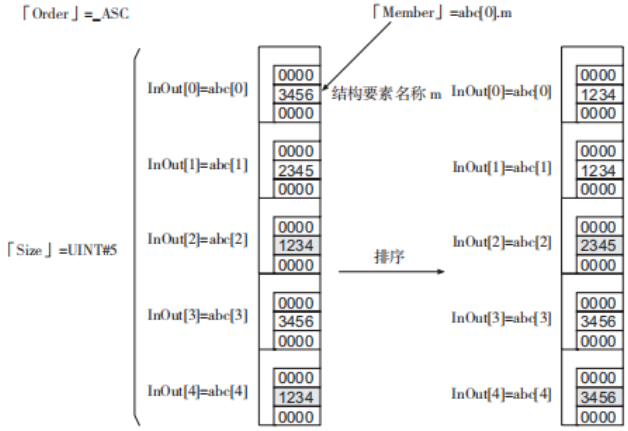
“Size”=UINT#5，“Order”=_ASC 时的示例如下所示。

LD



ST

```
RecSort_instance(A, abc[0], UINT#5, abc[0].m, _ASC, def, ghi, jkl);
```



	FBD	ST
定义变量	VAR size:UINT; order:_eSORT_ORDER; Inout:ARRAY [1..10] OF DUT; RecSort_0:RecSort; END_VAR	
程序	RecSort_0 RecSort size — Size Inout[1].int1 — Member order — Order Inout[1] — InOut	RecSort_0(Size:=size , Member:=Inout[1].int1 , Order:=order , InOut:=Inout[1]);

运行结果

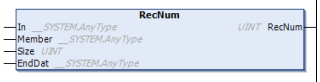
表达式	类型	值	准备值	地址	注
size	UINT	10			
order	_ESORT_ORDER	_ASC			
Inout	ARRAY [1..10] OF DUT				
Inout[1]	DUT				
real1	REAL	0			
int1	INT	0			
Inout[2]	DUT				
Inout[3]	DUT				
Inout[4]	DUT				
real1	REAL	0			
int1	INT	123			
Inout[5]	DUT				
real1	REAL	234			
int1	INT	222			
Inout[6]	DUT				
Inout[7]	DUT				
Inout[8]	DUT				
Inout[9]	DUT				
Inout[10]	DUT				
real1	REAL	0			
int1	INT	2342			

要点说明

- 本指令一旦执行，即使 “Execute” 的值为 FALSE 或执行时间超过任务周期，仍将继续处理直至最后。
- 请通过 “Done” 的值是否变为 TRUE 确认处理是否正常结束。
- “Execute”、“Done”、“Busy”、“Error” 的时序图请参阅 “ 本章说明 (P.2-2)”。
- “Member” 为实数时，会因数值而产生误差，因此结果可能会出现意外。
- 将数组中的元素传输至 InOut[ ] 时，该元素之后为处理对象。
- “Size” 的值为 0 时，“Done” 的值为 TRUE，InOut[ ] 不变。
- 以下情况时会发生异常。“Error” 变为 TRUE。
- “Order” 的值超过有效范围时。
- “Size” 的值超过 InOut[ ] 的数组区域时。
- “Member” 不是 InOut[ ] 的结构要素时。
- 不是 “Member” 支持的数据类型时。
- InOut[ ] 不是以结构体为元素的数组时。

6. 11. 8. RecNum (获取记录数)

计算以结构体为元素的数组到结尾数据为止的记录数。

指令	名称	FB/FUN	图形表现	ST 表现
RecNum	记录数获取	FUN		Out:=RecNum(In, Member, Size，EndDat);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In[ ]数组	对象数组	输入	作为处理对象的以结构体为元素的数组	—	—	(*)
Member	对象结构要素		In[ ] 的结构体的处理对象结构要素	遵从数据类型		
Size	元素个数		元素个数			
EndDat	结尾数据		结尾数据			
Out	记录数	输出	记录数	遵从数据类型	—	—

* 省略输入参数时，初始值不适用。编连时会发生异常。

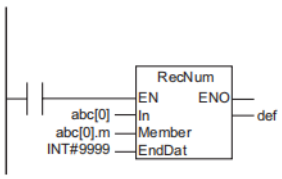
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ]数组	指定以结构体为元素的数组																			
Member	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	枚举体也可指定																			
	与 In[ ] 的处理对象结构要素相同的数据类型																			
Size							○													
EndDat	与 “Member” 相同的数据类型																			
Out							○													

功能

从以结构体为元素的数组 In[ ] 的开头起，检索对象结构要素 “Member” 的值与结尾数据 “EndDat” 一致的元素。最终，将与 “EndDat” 一致的元素前的元素数（记录数）代入 “Out”。将 In[ ] 的任一元素的对象结构要素作为自变量传输至 “Member”。传输至 In[ ] 的输入参数务必也像 array[3] 那样加上元素编号后再指定。

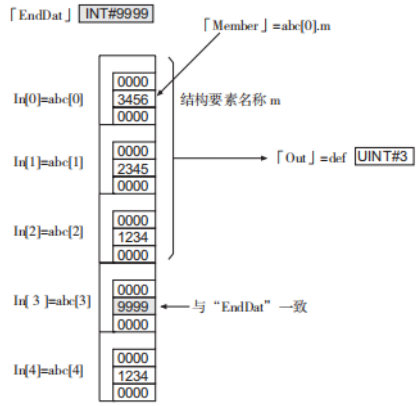
“EndDat”=INT#9999 时的示例如下所示。

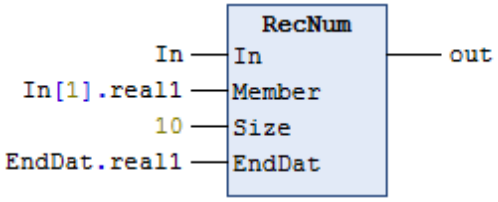
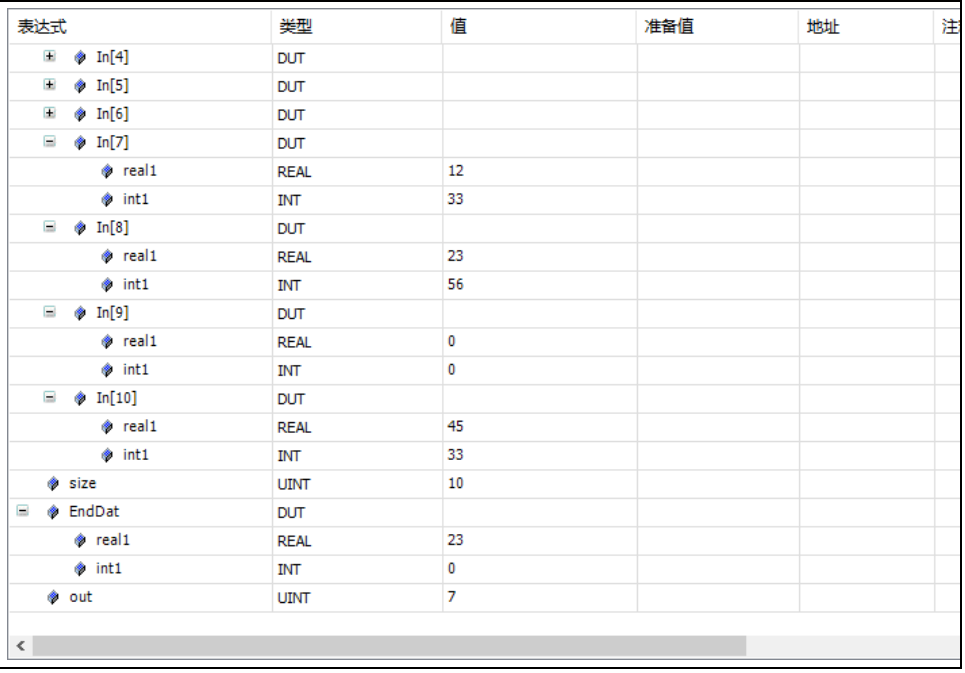
LD



ST

```
def:=RecNum(abc[0], abc[0].m, INT#9999);
```



	FBD	ST
定义变量	VAR In:ARRAY [1..10] OF DUT; size:UINT; EndDat:DUT; out:UINT; END_VAR	
程序		out:=RecNum(In:=In[1] , Member:=In[1].real1 , Size:=size , EndDat:=EndDat.real1);
运行结果		

表达式	类型	值	准备值	地址	注
In[4]	DUT				
In[5]	DUT				
In[6]	DUT				
In[7]	DUT				
real1	REAL	12			
int1	INT	33			
In[8]	DUT				
real1	REAL	23			
int1	INT	56			
In[9]	DUT				
real1	REAL	0			
int1	INT	0			
In[10]	DUT				
real1	REAL	45			
int1	INT	33			
size	UINT	10			
EndDat	DUT				
real1	REAL	23			
int1	INT	0			
out	UINT	7			

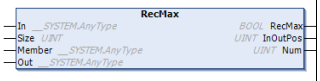
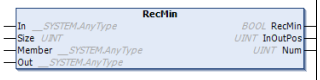
要点说明

- In[ ] 的结构要素中无与 “EndDat” 一致的结构要素时，将 In[ ] 的所有元素数代入 “Out”。
- “Member” 为实数时，会因数值而产生误差，因此结果可能会出现意外。
- “EndDat” 为实数时，“EndDat” 请勿指定为非数。
- 将数组中的元素传输至 In[ ] 时，该元素之后为处理对象。
- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- “Member” 不是 In[ ] 的结构要素时。
- “Member” 或 “EndDat” 为 STRING 型，且未以 NULL 字符结尾时。
- 不是 “Member” 支持的数据类型时。
- “Member” 和 “EndDat” 的数据类型不同时。
- In[ ] 不是以结构体为元素的数组时。

6.11.9. RecMax/RecMin (记录最大值检索/记录最小值检索)

RecMax：在以结构体为元素的数组中，检索指定的结构要素的最大值。

RecMin：在以结构体为元素的数组中，检索指定的结构要素的最小值。

指令	名称	FB/FUN	图形表现	ST 表现
RecMax	记录最大值检索	FUN		RecMax(In, Size,Member, Out，InOutPos, Num);
RecMin	记录最小值检索	FUN		RecMin(In, Size, Member, Out，InOutPos, Num);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In[ ]数组	对象数组	输入	作为处理对象的以结构体为元素的数组	—	—	(*)
Size	检索对象的元素数		检索对象的元素数	遵从数据类型		1
Member	对象结构要素		In[ ] 的结构体的处理对象结构要素			(*)
Out	记录数		检索结果			—
InOutPos[ ] 数组	检索结果的元素编号	输出	检索结果的元素编号	遵从数据类型	—	—
Num	检索个数	输出	检索个数			0

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In[ ]数组	指定以结构体为元素的数组																			
Size							○													
Member	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	与 In[ ] 的处理对象结构要素相同的数据类型																			
InOutPos[ ]数组							○													
Out						○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
Num							○													

功能

在以结构体为元素的数组 In[ ] 的 “Size” 个元素即 In[0] ～ In[“Size”-1] 中，检索结构体的检索对象结构要素“Member” 的值。将 In[ ] 的任一元素的检索对象结构要素作为自变量传输至 “Member”。将检索结果的元素编号、检索元素数分别代入 InOutPos[0]、“Num”。有 2 个以上检索结果时，将 In[ ]中最低位的检索结果的元素编号代入 InOutPos[0]。传输至 In[ ] 的输入参数务必也像 array[3] 那样加上元素编号后再指定。

整数、实数以外的数据类型的值的大小关系的判断如下表所示。

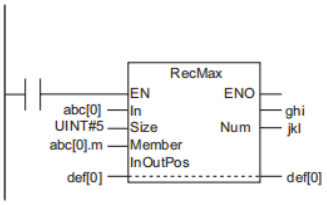
数据类型	含义
TIME	值较大者判断为大
DATE、TOD、DT	对于日期和时刻，较后者判断为大
STRING	比较首字符的 ASCII 码大小

RecMax：检索最大值。检索结果 “Out” 中代入检索对象结构要素的最大值。

RecMin：检索最小值。检索结果 “Out” 中代入检索对象结构要素的最小值。

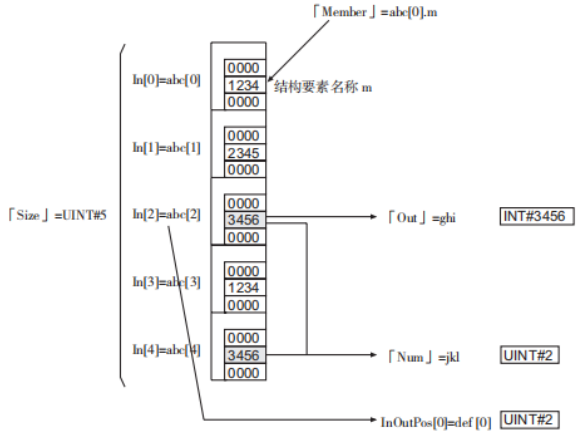
RecMax 指令下，“Size”=UINT#5 时的示例如下所示。

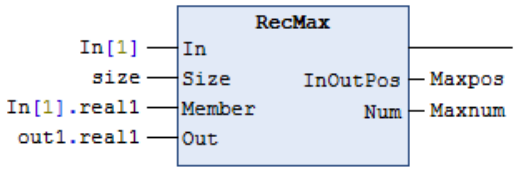
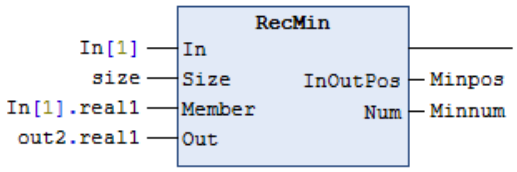
LD



ST

```
ghi:=RecMax(abc[0], UINT#5, abc[0].m, def[0], jkl);
```



	FBD	ST
定义变量		VAR size :UINT; In :ARRAY [1..10] OF DUT; out1 :UINT; out2 :UINT; Maxpos :UINT; Minpos :UINT; Maxnum :UINT; Minnum :UINT; END_VAR
程序	 	RecMax(In:=In[1] , Size:=size , Member:=In[1].real1 , Out:=out1.real1 , InOutPos=>Maxpos , Num=>Maxnum); RecMin(In:=In[1] , Size:=size , Member:=In[1].real1 , Out:=out2.real1 , InOutPos=>Minpos , Num=>Minnum);

运行结果

表达式	类型	值	准备值	地址	注
In[2]	DUT				
In[3]	DUT				
In[4]	DUT				
In[5]	DUT				
In[6]	DUT				
In[7]	DUT				
In[8]	DUT				
real1	REAL	1			
int1	INT	0			
In[9]	DUT				
real1	REAL	3456			
int1	INT	0			
In[10]	DUT				
out1	DUT				
real1	REAL	3456			
int1	INT	0			
out2	DUT				
real1	REAL	1			
int1	INT	0			
Maxpos	UINT	8			
Minpos	UINT	7			
Maxnum	UINT	10			
Minnum	UINT	10			

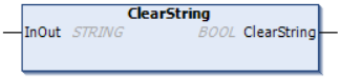
要点说明

- “Member” 和 “Out” 的数据类型不同时，请从下列数据类型中选择，以使 “Out” 的有效范围包含
- “Member” 的有效范围。（INT，UINT，UDINT，ULINT，SINT，INT，DINT，LINT，REAL，LREAL ）
- “Member” 为实数时，会因数值而产生误差，因此结果可能会出现意外。
- 将数组中的元素传输至 In[ ] 时，该元素之后为处理对象。
- “In” 为枚举体时，请务必将传输至 “In” 的输入参数设为变量。如果传输常数，编连时会发生异常。
- “Size” 的值为 0 时，“Out”、“Num” 的值变为 0。InOutPos[ ] 的值不变。
- 以下情况时会发生异常。ENO 变为 FALSE，“Out”、InOutPos[ ]、“Num” 不变。
- “Size” 的值超过 In[ ] 的数组区域时。
- “Member” 不是 In[ ] 的结构要素时。
- InOutPos[ ] 的数组大小小于 In[ ] 的维数时。
- 向 In[ ] 传输了无下标的数组时。
- 不是 “Member” 支持的数据类型时。
- “Member” 为 STRING 型，且未以 NULL 字符结尾时。

6.12. 字符串指令

6.12.1. ClearString (字符串清除)

清除字符串

指令	名称	FB/FUN	图形表现	ST 表现
ClearString	字符串 • 清除	FUN		ClearString(InOut);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
InOut	清除字符串	输入输出	待清除的字符串	遵从数据类型	—	—
Out	返回值	输出	始终为 TRUE	仅 TRUE		

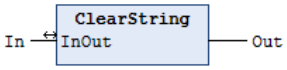
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out	○																			

功能

将清除字符串 “InOut” 清除。在 “InOut” 的所有区域中保存 NULL 字符。示例如下所示。

STRING 型变量 In 的默认值为‘abcd’,在调用函数后被清空。

	FBD	ST
定义变量		PROGRAM PLC_PRG VAR check :BOOL; In :STRING :='abcd'; Out :BOOL; END_VAR

程序		IF check FALSE THEN
		Out FALSE :=ClearString(InOut:= In 'abcd');
		END_IF RETURN
		
		IF check TRUE THEN
		Out FALSE :=ClearString(InOut:= In '');
		END_IF RETURN

运行结果		

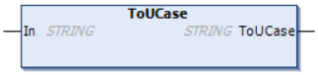
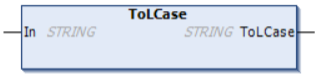
要点说明

- 在 ST 程序中使用本指令时，不使用返回值 “Out”。

6.12.2. ToUCase/ToLCase (字符串大/小写字母转换)

ToUCase：将字符串中的半角字母均转换为大写字母

ToLCase：将字符串中的半角字母均转换为小写字母

指令	名称	FB/FUN	图形表现	ST 表现
ToUCase	字符串•大写字母转换	FUN		Out:=ToUCase(In);
ToLCase	字符串•小写字母转换	FUN		Out:=ToLCase(In);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	转换对象	输入	转换的对象字符串	遵从数据类型	—	‘’
Out	转换结果	输出	转换后的字符串	遵从数据类型		—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out																				○

功能

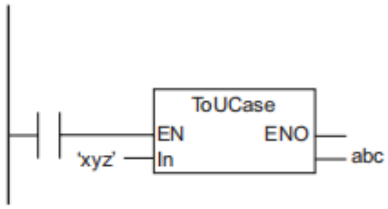
ToUCase：将转换对象字符串 “In” 的半角字母均转换为大写字母。

ToLCase：将转换对象字符串 “In” 的半角字母均转换为小写字母。

所有指令在输出时，字符串结尾均含 NULL 字符。非半角字母的字符不受影响。

ToUCase 指令下，“In”='xyz' 时的示例如下所示。变量 abc 的值为 'XYZ'。

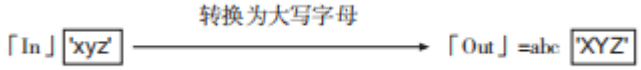
LD

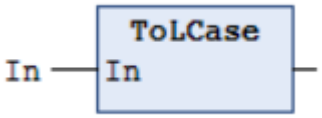


ST

```
abc:=ToUCase('xyz');
```

将“In”的半角字母均转换为大写字母。



	FBD	ST
定义变量	VAR In :STRING; Out :STRING; END_VAR	
程序		Out:=ToLCase(In:=In);
运行结果		

要点说明

- 全角字母不属于转换对象。
- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- “In” 未以 NULL 字符结尾时。
- “In” 的文字代码异常时。
- 转换结果超过 “Out” 的大小时。

6.12.3. TrimL/TrimR (字符串左/右侧调整)

TrimL：删除字符串开头的空格

TrimR：删除字符串末尾的空格

指令	名称	FB/FUN	图形表现	ST 表现
TrinL	字符串•左侧调整	FUN		Out:=TrinL(In);

TrinR	字符串•右侧调整	FUN		Out:=TrinR(In);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	删除对象	输入	删除的对象字符	遵从数据类型	—	‘’
Out	删除结果	输出	删除后的字符串	遵从数据类型		—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out																				○

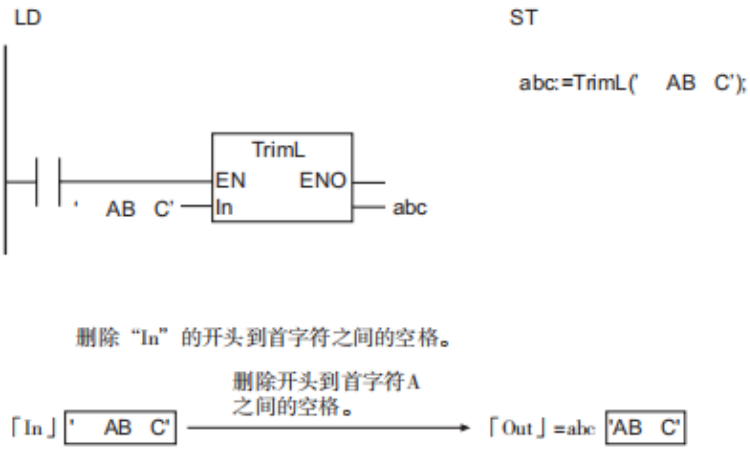
功能

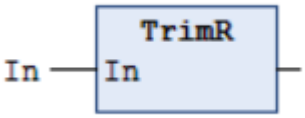
TrimL：将删除对象字符串 “In” 的开头到首字符之间的空格删除。开头无空格时，不进行任何处理。

TrimR：将删除对象字符串 “In” 的尾字符到末尾之间的空格删除。末尾无空格时，不进行任何处理。

所有指令在输出时，字符串结尾均含 NULL 字符。ASCII 码的空格（16#20）与中文全角空格(16#E38080)均视为空格。

TrimL 指令下，“In”=' AB C' 时的示例如下所示。变量 abc 的值为 'AB C'。



	FBD	ST
定义变量	VAR In :STRING; Out :STRING; END_VAR	
程序		Out:=TrimR(In:=In);
运行结果		

In	STRING	'AABB '
Out	STRING	'AABB'

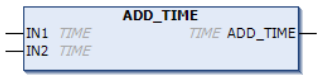
要点说明

- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- “In” 未以 NULL 字符结尾时。
- “In” 的文字代码异常时。
- 转换结果超过 “Out” 的大小时。

6.13. 时间/时刻指令

6.13.1. ADD_TIME (时间相加)

对时间和时间进行加法运算

指令	名称	FB/FUN	图形表现	ST 表现
ADD_TIME	时间加法	FUN		Out:=ADD_TIME(In1, In2);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In1	被加法时间	输入	被加法时间	遵从数据类型	ms	T#0S
In2	加法时间		加法时间			
Out	加法结果时间	输出	加法结果时间	遵从数据类型	ms	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																○				
In2																○				
Out																○				

功能

对时间 “In1” 和 “In2” 进行加法运算。加法运算结果 “Out” 也是时间。

“In1”=T#1d2h3m4s、“In2”=T#5d6h7m8s 时的示例如下所示。

	FBD	ST
定义变量	PROGRAM PLC_PRG VAR time1 :TIME; time2 :TIME; out :TIME; check :BOOL; END_VAR	

程序	time1 T#1d2h3m4s
	time2 T#5d6h7m8s
	ADD_TIME IN1 IN2
	Out T#6d8h10m12s
		IF check FALSE THEN
		Out T#6d8h10m12s :=ADD_TIME(
		IN1:=time1 T#1d2h3m4s ,
		IN2:=time2 T#5d6h7m8s);
		check FALSE :=FALSE;
		END_IF

运行结果	表达式 / 类型 / 值 time1 / TIME / T#1d2h3m4s time2 / TIME / T#5d6h7m8s out / TIME / T#6d8h10m12s	

6. 13. 2. ADD_TOD_TIME (时刻和时间的加法)

将时刻加上时间

指令	名称	FB/FUN	图形表现	ST 表现
ADD_TOD_TIME	时刻和时间的加法	FUN	ADD_TOD_TIME IN1 TIME_OF_DAY TIME_OF_DAY ADD_TOD_TIME IN2 TIME	Out:=ADD_TOD_TIME(In1, In2);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In1	被加法时刻	输入	被加法时刻	遵从数据类型	时分秒	TOD#0:0:0
In2	加法时间		加法时间		ms	T#0ms
Out	加法结果时刻	输出	加法结果时刻	遵从数据类型	时分秒	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																		○		
In2																○				
Out																		○		

功能

对时刻 “In1” 加上时间 “In2” 。加法运算结果 “Out” 为时刻。

	FBD	ST
定义变量	PROGRAM PLC_PRG VAR In1 :TOD; In2 :TIME; out :TOD; END_VAR	
程序	In1 TOD#10:10:10 — IN1 In2 T#50ms — IN2 ADD_TOD_TIME Out — TOD#10:10:10.050	out TOD#10:10:10.050 :=ADD_TOD_TIME(IN1:= In1 TOD#10:10:10 , IN2:=In2 T#50ms);
运行结果	表达式 / 类型 / 值 In1 / TIME_OF_DAY / TOD#10:10:10 In2 / TIME / T#50ms out / TIME_OF_DAY / TOD#10:10:10.050	

6.13.3. ADD_DT_TIME (日期时刻和时间的加法)

将时刻加上时间

指令	名称	FB/FUN	图形表现	ST 表现
ADD_DT_TIME	日期时刻和时间的加法	FUN		Out:=ADD_DT_TIME(In1, In2);

变量

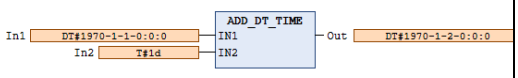
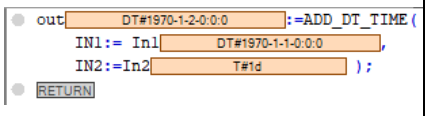
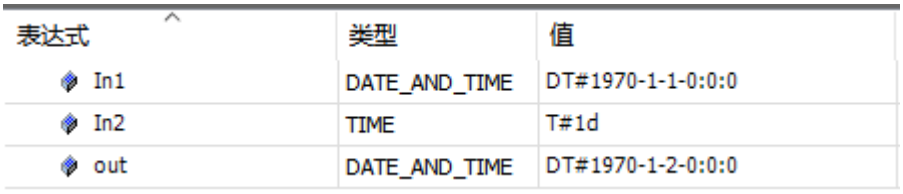
	名称	输入/输出	内容	有效范围	单位	初始值
In1	被加法日期时刻	输入	被加法日期时刻	遵从数据类型	年月日时分秒	TDT#1970-1-1-0:0:0
In2	加法时间		加法时间		ns	T#0s
Out	加法结果日期时刻	输出	加法结果日期时刻	遵从数据类型	年月日时分秒	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																			○	
In2																○				
Out																			○	

功能

将日期时刻 “In1” 加上时间 “In2”。加法运算结果 “Out” 为日期时刻。闰年也加进去。

“In1”=DT#1970-1-1-0:0:0、“In2”=T#1d 时的示例如下所示。

	FBD	ST
定义变量	PROGRAM PLC_PRG VAR In1 :DT; In2 :TIME; out :DT; END_VAR	
程序		
运行结果		

6.13.4. SUB_TIME (时间相减)

将时间减去时间。

指令	名称	FB/FUN	图形表现	ST 表现
SUB_TIME	时间减法	FUN		Out:=SUB_TIME(In1, In2);

变量

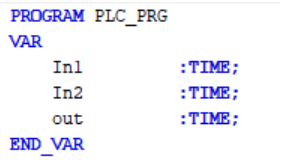
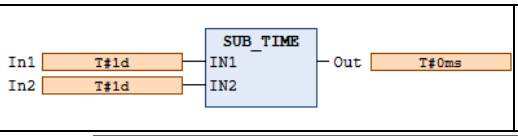
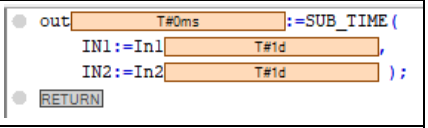
	名称	输入/输出	内容	有效范围	单位	初始值
In1	被减法时间	输入	被减法时间	遵从数据类型	ms	T#0ms
In2	减法时间		减法时间			
Out	减法结果时间	输出	减法结果时间	遵从数据类型	ms	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	LISINT	LUINT	LUDINT	LULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																○				
In2																○				
Out																○				

功能

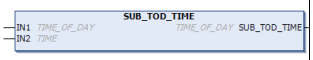
将时间 “In1” 减去时间 “In2”。减法运算结果 “Out” 也是时间。

“In1” = “In2” =T#1d 时的示例如下所示。

	FBD	ST
定义变量		
程序		
运行结果		

6.13.5. SUB_TOD_TIME (时刻和时间的减法)

将时刻减去时间

指令	名称	FB/FUN	图形表现	ST 表现
SUB_TOD_TIME	时刻和时间的减法	FUN		Out:=SUB_TOD_TIME(In1, In2);

变量

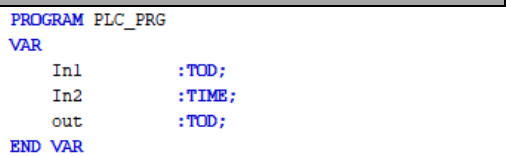
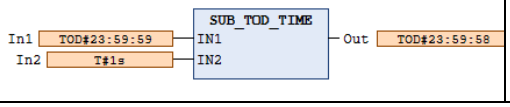
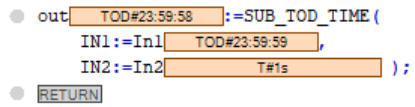
	名称	输入/输出	内容	有效范围	单位	初始值
In1	被减法时刻	输入	被减法时刻	遵从数据类型	时分秒	TOD#0:0:0
In2	减法时间		减法时间		ms	T#0s
Out	减法结果时刻	输出	减法结果时刻	遵从数据类型	时分秒	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																		○		
In2																○				
Out																		○		

功能

将时刻 “In1” 减去时间 “In2”。减法运算结果 “Out” 为时刻。

“In1”=TOD#23:59:59、“In2”=T#1s 时的示例如下所示。

	FBD	ST
定义变量		
程序		
运行结果		

6.13.6. SUB_TOD_TOD (时刻减法)

将时刻减去时刻

指令	名称	FB/FUN	图形表现	ST 表现
SUB_TOD_TOD	时刻减法	FUN		Out:=SUB_TOD_TOD(In1, In2);

变量

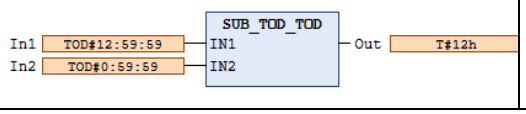
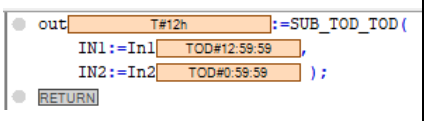
	名称	输入/输出	内容	有效范围	单位	初始值
In1	被减法时刻	输入	被减法时刻	遵从数据类型	时分秒	TOD#0:0:0
In2	减法时刻		减法时刻			
Out	减法结果时间	输出	减法结果时间	遵从数据类型	ms	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																		○		
In2																		○		
Out																○				

功能

将时刻 “In1” 减去时刻 “In2”。减法运算结果 “Out” 为时间。

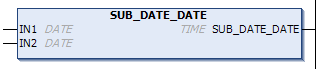
"In1" = TOD#12:59:59,"In2" = TOD#0:59:59 时示例结果如下。

	FBD	ST
定义变量	PROGRAM PLC_PRG VAR In1 :TOD; In2 :TOD; out :TIME; END_VAR	
程序		
运行结果		

表达式	类型	值
out	TIME	T#12h
In2	TIME_OF_DAY	TOD#0:59:59
In1	TIME_OF_DAY	TOD#12:59:59

6.13.7. SUB_DATE_DATE (日期减法)

将日期减去日期。

指令	名称	FB/FUN	图形表现	ST 表现
SUB_DATE_DATE	日期减法	FUN		Out:=SUB_DATE_DATE(In1,In2);

变量

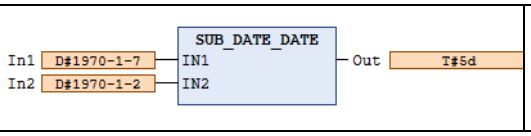
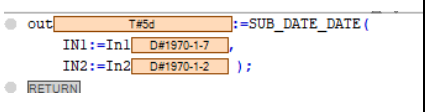
	名称	输入/输出	内容	有效范围	单位	初始值
In1	被减法日期	输入	被减法日期	遵从数据类型	年月日	D#1970-1-1
In2	减法日期		减法日期			
Out	减法结果时间	输出	减法结果时间	遵从数据类型	ms	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	LISINT	LUINT	LUDINT	LULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																	○			
In2																	○			
Out																○				

功能

将日期 “In1” 减去日期 “In2”。减法运算结果 “Out” 为时间。

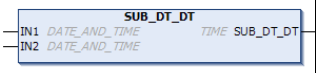
“In1”=D#1970-1-7、“In2”=D#1970-1-2 时的示例如下所示。

	FBD	ST
定义变量	PROGRAM PLC_PRG VAR In1 :DATE; In2 :DATE; out :TIME; END_VAR	
程序		
运行结果		

表达式	类型	值
In1	DATE	D#1970-1-7
In2	DATE	D#1970-1-2
out	TIME	T#5d

6.13.8. SUB_DT_DT (日期时刻相减)

将日期时刻减去日期时刻

指令	名称	FB/FUN	图形表现	ST 表现
SUB_DT_DT	日期时刻减法	FUN		Out:=SUB_DT_DT(In1, In2);

变量

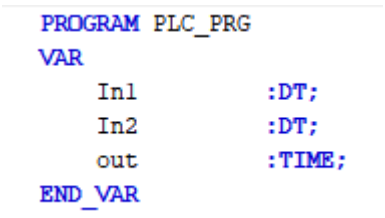
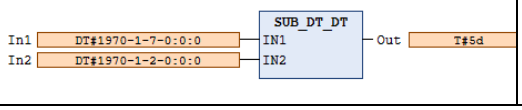
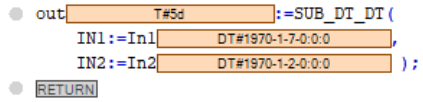
	名称	输入/输出	内容	有效范围	单位	初始值
In1	被减法日期时刻	输入	被减法日期时刻	遵从数据类型	年月日时分秒	DT#1970-1-1-0:0:0
In2	减法日期时刻		减法日期时刻			
Out	减法结果时间	输出	减法结果时间	遵从数据类型	ns	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																			○	
In2																			○	
Out																○				

功能

将日期时刻 “In1” 减去日期时刻 “In2”。减法运算结果 “Out” 为时间。

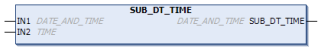
“In1”=DT#1970-1-7-0:0:0、“In2”=DT#1970-1-2-0:0:0 时的示例如下所示。

	FBD	ST
定义变量		
程序		

运行结果	表达式 / 类型 / 值 In1 / DATE_AND_TIME / DT#1970-1-7-0:0:0 In2 / DATE_AND_TIME / DT#1970-1-2-0:0:0 out / TIME / T#5d

6. 13. 9. SUB_DT_TIME (日期时刻和时间相减)

将日期时刻减去时间

指令	名称	FB/FUN	图形表现	ST 表现
SUB_DT_TIME	日期时刻和时间的减法	FUN		Out:=SUB_DT_TIME(In1, In2);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In1	被减法日期时刻	输入	被减法日期时刻	遵从数据类型	年月日时分秒	DT#1970-1-1-0:0:0
In2	减法时间		减法时间		ms	T#0s
Out	减法结果日期时刻	输出	减法结果日期时刻	遵从数据类型	年月日时分秒	—

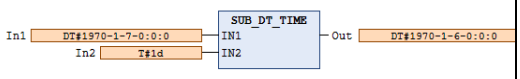
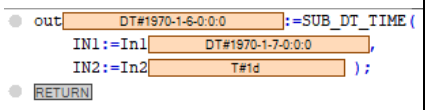
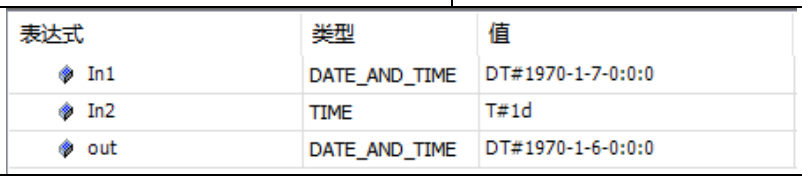
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																			○	
In2																○				
Out																			○	

功能

将日期时刻 “In1” 减去时间 “In2”。减法运算结果 “Out” 为日期时刻。闰年也加进去。

“In1”=DT#1970-1-7-0:0:0、“In2”=T#1d 时的示例如下所示。

	FBD	ST
定义变量	PROGRAM PLC_PRG VAR In1 :DT; In2 :TIME; out :DT; END_VAR	

程序		
运行结果		

表达式	类型	值
In1	DATE_AND_TIME	DT#1970-1-7-0:0:0
In2	TIME	T#1d
out	DATE_AND_TIME	DT#1970-1-6-0:0:0

6. 13. 10. MULTIME (时间乘法)

按指定乘数对时间进行乘法运算

指令	名称	FB/ FUN	图形表现	ST 表现
MULTIME	时间乘法	FUN		Out:=MULTIME(In1, In2);

变量

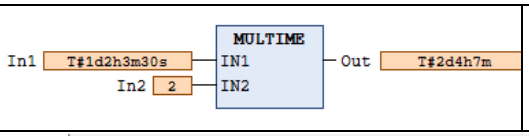
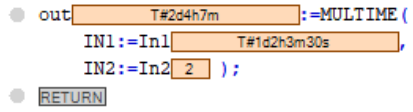
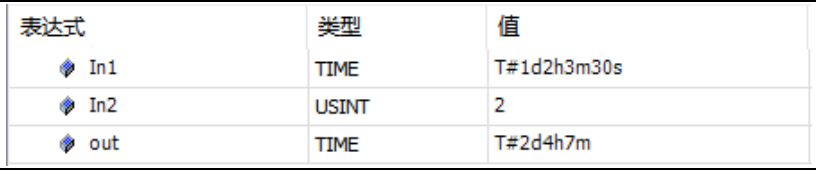
	名称	输入/ 输出	内容	有效范围	单位	初始值
In1	被乘法时间	输入	被乘法时间	遵从数据类型	ms	T#0s
In2	乘数		乘数		—	(*)
Out	乘法结果时间	输出	乘法结果时间	遵从数据类型	ms	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																○				
In2						○	○	○	○	○	○	○	○	○	○					
Out																○				

功能

将日期时刻 “In1” 减去时间 “In2”。减法运算结果 “Out” 为日期时刻。闰年也加进去。

“In1”=T#1d2h3m30s、“In2”=T#1d 时的示例如下所示。

	FBD	ST
定义变量	PROGRAM PLC_PRG VAR In1 :TIME; In2 :USINT; out :TIME; END_VAR	
程序		
运行结果		

表达式	类型	值
In1	TIME	T#1d2h3m30s
In2	USINT	2
out	TIME	T#2d4h7m

6. 13. 11. DIVTIME (时间除法)

按指定除数对时间进行除法运算

指令	名称	FB/ FUN	图形表现	ST 表现
MULTIME	时间乘法	FUN		Out:=MULTIME(In1, In2);

变量

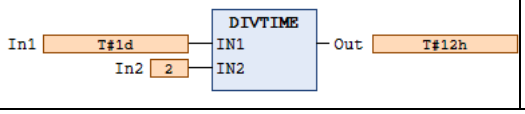
	名称	输入/ 输出	内容	有效范围	单位	初始值
In1	被除法时间	输入	被除法时间	遵从数据类型	ms	T#0s
In2	除数		除数		—	(*)
Out	除法结果时间	输出	除法结果时间	遵从数据类型	ms	—

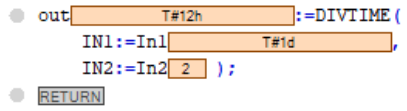
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																○				
In2						○	○	○	○	○	○	○	○	○	○					
Out																○				

功能

将时间 “In1” 除以除数 “In2”。除法运算结果 “Out” 为时间。

“In1”=T#1d、“In2”=INT#2 时的示例如下所示。

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">
<pre>
PROGRAM PLC_PRG
VAR
    In1   :TIME;
    In2   :USINT;
    out   :TIME;
END_VAR
</pre>
</td></tr>
<tr><td>程序</td><td>


</td><td>


</td></tr>
<tr><td>运行结果</td><td colspan="2">


</td></tr>
</table>

6. 13. 12. CONCAT_DATE_TOD (日期和时刻结合)

组合日期和时刻。

指令	名称	FB/FUN	图形表现	ST 表现
CONCAT_DATE_TOD	日期和时刻的组合	FUN		Out:=CONCAT_DATE_TOD(In1, In2);

变量

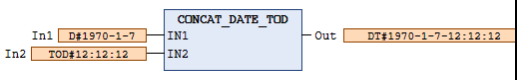
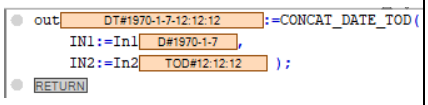
	名称	输入/输出	内容	有效范围	单位	初始值
In1	日期	输入	日期	遵从数据类型	年月日	D#1970-1-1
In2	时刻		时刻		时分秒	TOD#0:0:0
Out	组合结果日期时刻	输出	组合结果日期时刻	遵从数据类型	年月日时分秒	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In1																	○			
In2																		○		
Out																			○	

功能

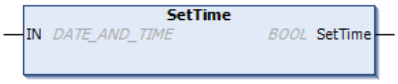
组合日期 “In1” 和时刻 “In2”。组合结果 “Out” 为日期时刻。

“In1”=D#1970-1-7，“In2” = TOD#12:12:12 时，示例如下所示。

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2"><pre>PROGRAM PLC_PRG
VAR
    In1   :DATE;
    In2   :TOD;
    out   :DT;
END_VAR</pre></td></tr>
<tr><td>程序</td><td> 
</td><td> 
</td></tr>
<tr><td>运行结果</td><td colspan="2"> 
</td></tr>
</table>

6. 13. 13. SetTime (时钟修正)

设置 UTC 时间

指令	名称	FB/FUN	图形表现	ST 表现
SetTime	时钟补偿	FUN		SetTime(In);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	UTC 时刻数据	输入	用于补偿系统时刻的当前日期时刻	遵从数据类型	年月日时分秒	DT#1970-1-1-0:0:0
Out	返回值	输出	时钟为 TRUE	仅 TRUE	—	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			○	
Out	○																			

要点说明

- 将日期时刻 “In” 的值设置为系统 UTC 时刻。
- 建议放在 EtherCat_Task 任务中使用，可能会造成堵塞

6. 13. 14. GetTime (获取时刻)

读取当前 UTC 时刻。

指令	名称	FB/FUN	图形表现	ST 表现
GetTime	时刻获取	FUN	GetTime DATE_AND_TIME GetTime	Out:=GetTime();

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Out	当前时刻	输出	当前时刻	遵从数据类型	年月日时分秒	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Out																			◯	

功能

读取当前时刻。当前时刻为已设定时间区的标准时间，而非 GMT(格林尼治标准时间)。

利用指令 Settime 将系统时间设置为“In” = DT#2022-12-27-16:38:10，再利用指令 Gettime 将系统时间读取到“recTime”中

	FBD	ST
定义变量	PROGRAM PLC_PRG VAR In1 :DATE; In2 :TOD; out :DT; END_VAR	

程序	In1 DT#2022-12-27-16:38:10 — SetTime IN — Out TRUE
	GetTime — recTime DT#2022-12-27-16:40:32
		IF set FALSE THEN
		out TRUE :=SetTime(IN:= In1 DT#2022-12-27-16:38:10);
		set FALSE :=FALSE;
		END_IF
		IF Out TRUE THEN
		recTime DT#2022-12-27-16:40:50 :=GetTime();
		END_IF
		RETURN

运行结果	表达式 / 类型 / 值 In1 / DATE_AND_TIME / DT#2022-12-27-16:38:10 Out / BOOL / TRUE recTime / DATE_AND_TIME / DT#2022-12-27-16:39:48 set / BOOL / FALSE	

6. 13. 15. DtToSec (日期时刻 TO 秒转换)

将日期时刻转换为 1970 年 1 月 1 日 0 时 0 分 0 秒起的秒数

指令	名称	FB/FUN	图形表现	ST 表现
DtToSec	日期时刻→秒转换	FUN		Out:=DtToSec(In);

变量

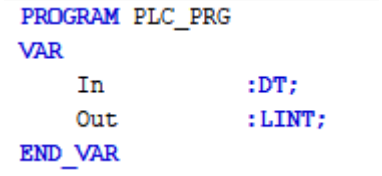
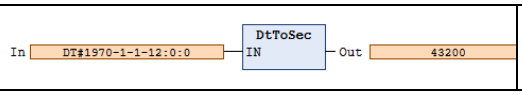
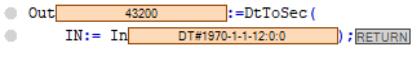
	名称	输入/输出	内容	有效范围	单位	初始值
In	日期时刻	输入	日期时刻	遵从数据类型	年月日时分秒	DT#1970-1-1-0:0:0
Out	秒	输出	1970 年 1 月 1 日 0 时 0 分 0 秒起的秒数	0~18446744073	秒	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			○	
Out													○							

功能

将日期时刻“In”转换为 1970 年 1 月 1 日 0 时 0 分 0 秒起的秒数。转换后的数值单位为秒。舍去小于 1 秒的值。

“In" = DT#1970-1-1-12:0:0 时，示例如下所示。

	FBD	ST
定义变量		
程序		

运行结果	表达式 / 类型 / 值 In / DATE_AND_TIME / DT#1970-1-1-12:0:0 Out / LINT / 43200

6. 13. 16. DateToSec (日期 TO 秒转换)

将日期转换为 1970 年 1 月 1 日 0 时 0 分 0 秒起的秒数

指令	名称	FB/FUN	图形表现	ST 表现
DateToSec	日期→秒转换	FUN		Out:=DateToSec(In);

变量

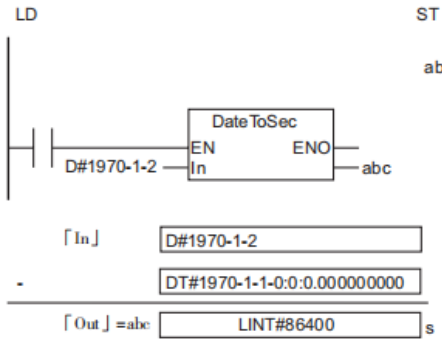
	名称	输入/输出	内容	有效范围	单位	初始值
In	日期	输入	日期	遵从数据类型	年月日	D#1970-1-1
Out	秒	输出	1970 年 1 月 1 日 0 时 0 分 0 秒起的秒数	0~18446744073	秒	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	○			
Out													○							

功能

将日期“In”的 0 时 0 分 0 秒的时刻转换为 1970 年 1 月 1 日 0 时 0 分 0 秒起的秒数。转换后的数值单位为秒。

“In”=D#1970-1-2 时的示例如下所示。



	FBD	ST
定义变量	VAR In :DATE; Out :LINT; END_VAR	
程序	In — dateToSec IN	Out:=DateToSec(IN:=In);
运行结果	In DATE D#1970-1-2 Out LINT 86400	

6. 13. 17. TodToSec (时刻 TO 秒转换)

将时刻转换为 0 时 0 分 0 秒起的秒数。

指令	名称	FB/FUN	图形表现	ST 表现
TodToSec	时刻→秒转换	FUN	TodToSec IN TIME_OF_DAY LINT TodToSec	Out:=TodToSec(In);

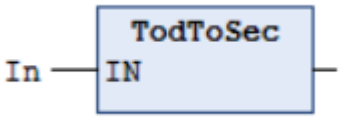
变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	时刻	输入	时刻	遵从数据类型	时分秒	TOD#0:0:0
Out	秒	输出	0 时 0 分 0 秒起的秒数	0~86399	秒	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																		○		
Out													○							

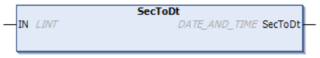
功能

将时刻 “In” 转换为 0 时 0 分 0 秒起的秒数。转换后的数值单位为秒。舍去小于 1 秒的值。

	FBD	ST
定义变量	VAR In :TOD; Out :LINT; END_VAR	
程序		Out:=TodToSec(IN:=In);
运行结果	In TIME_OF_DAY TOD#23:59:59 Out LINT 86399	

6. 13. 18. SecToDt (秒 TO 日期时刻转换)

将 1970 年 1 月 1 日 0 时 0 分 0 秒起的秒数转换为日期时刻。

指令	名称	FB/FUN	图形表现	ST 表现
SecToDt	秒→日期时刻转换	FUN		Out:=SecToDt(In);

变量

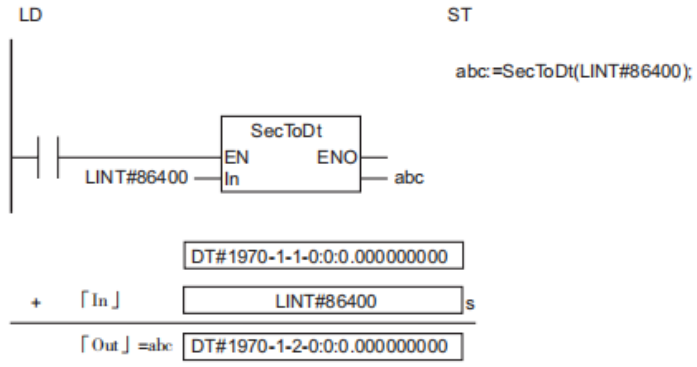
	名称	输入/输出	内容	有效范围	单位	初始值
In	秒	输入	1970 年 1 月 1 日 0 时 0 分 0 秒起的秒数	0~18446744073	秒	0
Out	日期时刻	输出	日期时刻	遵从数据类型	年月日时分秒	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													○							
Out																			○	

功能

将 1970 年 1 月 1 日 0 时 0 分 0 秒起的秒数 “In” 转换为日期时刻。

“In”=LINT#86400 时的示例如下所示。



	FBD	ST
定义变量	VAR In :LINT; Out :DT; END_VAR	
程序	In — SecToDt / IN	Out:=SecToDt(IN:=In);
运行结果	In LINT 86400 Out DATE_AND_TIME DT#1970-1-2-0:0:0	

6. 13. 19. SecToDate (秒 TO 日期转换)

将 1970 年 1 月 1 日 0 时 0 分 0 秒起的秒数转换为日期。

指令	名称	FB/FUN	图形表现	ST 表现
SecToDate	秒→日期转换	FUN	SecToDate IN LINT DATE SecToDate	Out:=SecToDate(In);

变量

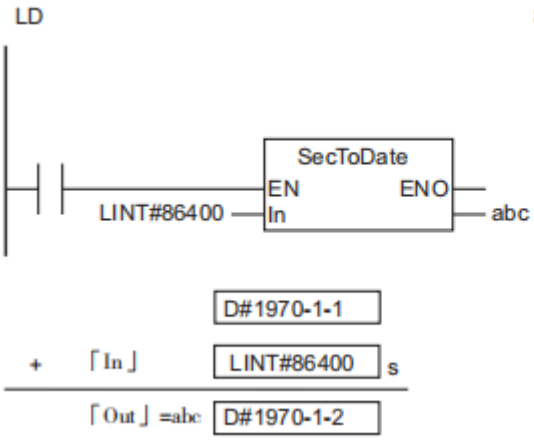
	名称	输入/输出	内容	有效范围	单位	初始值
In	秒	输入	1970 年 1 月 1 日 0 时 0 分 0 秒起的秒数	0~18446744073	秒	0
Out	日期	输出	日期	遵从数据类型	年月日	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													○							
Out																	○			

功能

将 1970 年 1 月 1 日 0 时 0 分 0 秒起的秒数 “In” 转换为日期。 舍去小于 1 日的值。

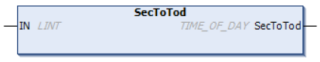
“In”=LINT#86400 时的示例如下所示。



	FBD	ST
定义变量	VAR In :LINT; Out :DATE; END_VAR	
程序	In — SecToDate / IN	Out:=SecToDate(IN:=In);
运行结果	In LINT 86400 Out DATE D#1970-1-2	

6. 13. 20. SecToTod (秒 TO 时刻转换)

将 0 时 0 分 0 秒起的秒数转换为时刻。

指令	名称	FB/FUN	图形表现	ST 表现
SecToTod	秒→时刻转换	FUN		Out:=SecToTod(In);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	秒	输入	0 时 0 分 0 秒起的秒数	遵从数据类型（*）	秒	0
Out	时刻	输出	时刻	遵从数据类型	时分秒	—

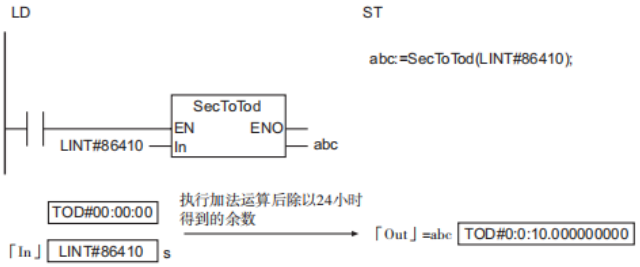
* 不含负数。

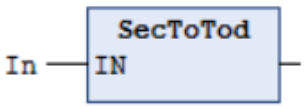
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													○							
Out																		○		

功能

将 0 时 0 分 0 秒起的秒数 “In” 转换为时刻。“In” 的值为 24 小时以上时，将 “In” 除以 24 小时得到的余数转换为时刻。

“In”=LINT#86410 时的示例如下所示。



	FBD	ST
定义变量	VAR In :LINT; Out :TOD; END_VAR	
程序		Out:=SecToTod(IN:=In);
运行结果	In LINT 86410 Out TIME_OF_DAY TOD#0:0:10	

要点说明

- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- “In” 的值超过有效范围时。

6. 13. 21. TimeToNanoSec (时间 TO 纳秒转换)

将时间转换为纳秒数。

指令	名称	FB/FUN	图形表现	ST 表现
TimeToNanoSec	时间→纳秒转换	FUN		Out:=TimeToNanoSec(In);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	时间	输入	时间	遵从数据类型		T#0s
Out	纳秒	输出	纳秒	(*)	ns	—

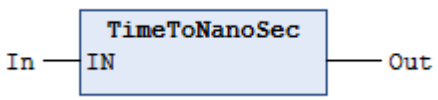
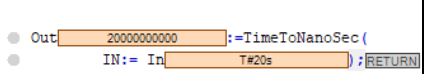
* -2,147,483,648 ～ 2,147,483,647

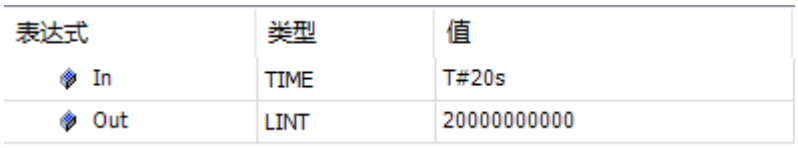
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																○				
Out													○							

功能

将时间 “In” 转换为纳秒数。

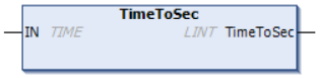
“In”=T#20s 如下所示。

	FBD	ST
定义变量	PROGRAM PLC_PRG VAR In :TIME; Out :LINT; END_VAR	
程序		

运行结果	

6. 13. 22. TimeToSec (时间 TO 秒转换)

将时间转换为秒数

指令	名称	FB/FUN	图形表现	ST 表现
TimeToSec	时间→秒转换	FUN		Out:=TimeToSec(In);

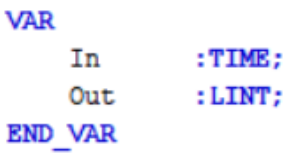
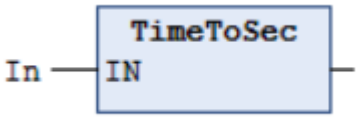
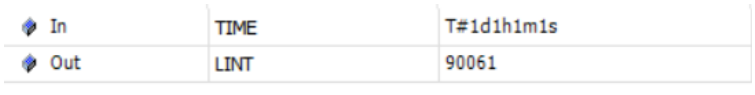
变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	时间	输入	时间	遵从数据类型		T#0s
Out	秒数	输出	秒数	9223372036~9223372036	秒	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																○				
Out													○							

功能

将时间 “In” 转换为秒数。舍去小于 1 秒的值。

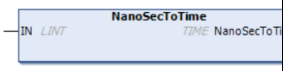
	FBD	ST
定义变量		
程序		Out:=TimeToSec(IN:=In);
运行结果		

要点说明

- “In” 的单位为纳秒，“Out” 的单位为秒。

6. 13. 23. NanoSecToTime (纳秒 TO 时间转换)

将纳秒数转换为时间

指令	名称	FB/FUN	图形表现	ST 表现
NanoSecToTime	纳秒→时间转换	FUN		Out:=NanoSecToTime(In);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	秒数	输入	秒数	(*)	ns	0
Out	时间	输出	时间	遵从数据类型	ns	—

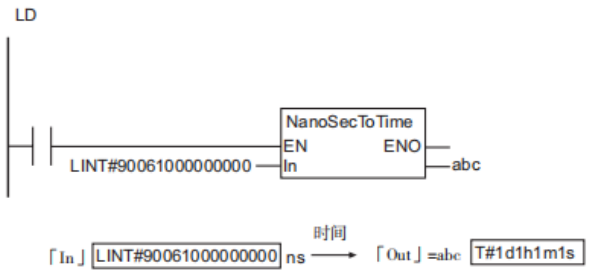
* -9223372036854775808 ～ 9223372036854775807

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													○							
Out																○				

功能

将纳秒数 “In” 转换为时间。

“In”=LINT#90061000000000 时的示例如下所示。

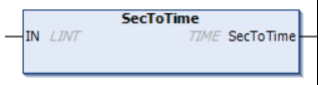


ST

```
abc:=NanoSecToTime(LINT#90061000000000);
```

6. 13. 24. SecToTime (秒 TO 时间转换)

将秒数转换为时间

指令	名称	FB/FUN	图形表现	ST 表现
SecToTime	秒→时间转换	FUN		Out:=SecToTime(In);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	秒数	输入	秒数	0~4294967	s	0
Out	时间	输出	时间	遵从数据类型	s	—

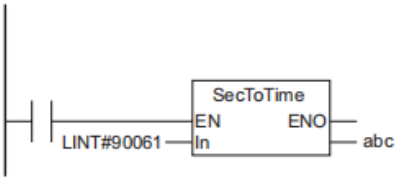
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													○							
Out																○				

功能

将秒数 “In” 转换为时间。

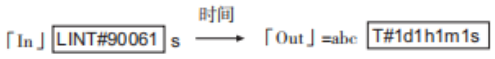
“In”=LINT#90061 时的示例如下所示。

LD



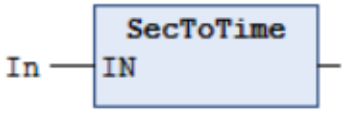
ST

```
abc:=SecToTime(LINT#90061);
```



0

	FBD	ST
定义变量	VAR In :LINT; Out :TIME; END_VAR	

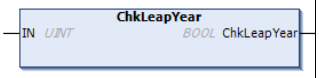
程序		Out:=SecToTime(IN:=In);
运行结果	In LINT 90061 Out TIME T#1d1h1m1s	

要点说明

- “In” 的单位为秒。
- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- “In” 的值超过有效范围时。

6. 13. 25. ChkLeapYear (闰年判别)

判断知道的年份是否为闰年

指令	名称	FB/FUN	图形表现	ST 表现
ChkLeapYear	闰年判别	FUN		Out:=ChkLeapYear(In);

变量

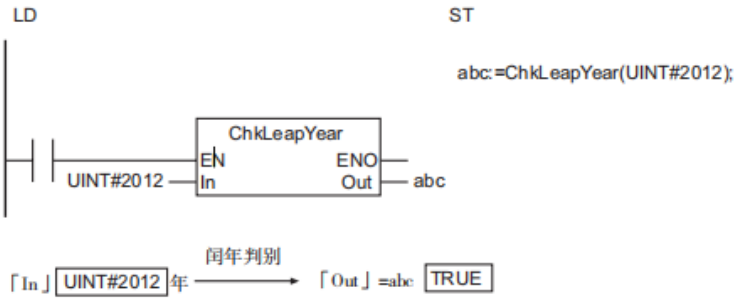
	名称	输入/输出	内容	有效范围	单位	初始值
In	公历年	输入	公历年	1970~2554	年	1970
Out	判定结果	输出	TRUE ： 是闰年 FALSE：非闰年	遵从数据类型	—	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In							○													
Out	○																			

功能

判断公历年 “In” 是否为闰年。如果是闰年，则判定结果 “Out” 的值为 TRUE。否则为 FALSE。

“In”=UINT#2012 时的示例如下所示。



	FBD	ST
定义变量	VAR In :UINT; Out :BOOL; END_VAR	
程序	In — ChkLeapYear / IN	Out:=ChkLeapYear(IN:=In);
运行结果	In UINT 2012 Out BOOL TRUE	

使用注意事项

- “In” 的值超过有效范围时，不会发生异常，“Out” 的值为错误值。

6. 13. 26. GetDaysOfMonth (月的天数获取)

获取指定月的天数。

指令	名称	FB/FUN	图形表现	ST 表现
GetDaysOfMonth	月的天数获取	FUN	GetDaysOfMonth Year UINT Month USINT USINT GetDaysOfMonth	Out:=GetDaysOfMonth(Year, Month);

变量

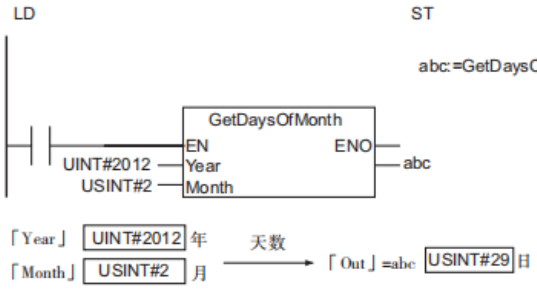
	名称	输入/输出	内容	有效范围	单位	初始值
Year	年	输入	公历年	1970~2554	年	1970
Month	月		月	1~12	月	1
Out	天数	输出	天数	28~31	日	—

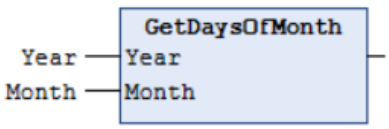
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Year							○													
Month						○														
Out						○														

功能

获取 “Year” 表示年的天数、“Month” 表示月的天数。

“Year”=UINT#2012、“Month”=USINT#2 时的示例如下所示。



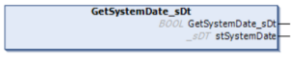
	FBD	ST
定义变量	VAR Year :UINT; Month :USINT; Days :USINT; END_VAR	
程序		Days:=GetDaysOfMonth(Year:=Year , Month:=Month);
运行结果		

要点说明

- “Year” 的值超过有效范围时，不会发生异常，“Out” 的值为错误值。
- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- “Month” 的值超过有效范围时。

6. 13. 27. GetSystemDate_sDt (_sDT 格式时间获取)

获取系统 Local 时间

指令	名称	FB/FUN	图形表现	ST 表现
GetSystemDate_sDt	获取系统 Local 时间	FUN		GetSystemDate_sDt(stSystemDate=>);

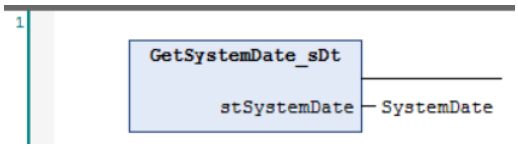
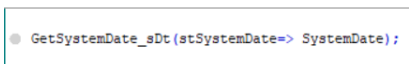
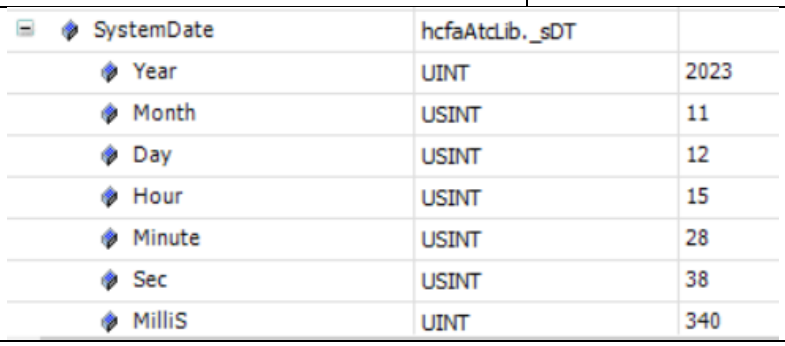
变量

	名称	输入/输出	内容	有效范围	单位	初始值
stSystemDate	日期时刻	输出	日期时刻，_sDT 格式	遵从数据类型	年月日时分秒	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
stSystemDate																			○	

功能

读取当前时刻。当前时刻为已设定时间区的标准时间。

	FBD	ST
定义变量	VAR SystemDate :hcfaAtcLib._sDT; END_VAR	
程序		
运行结果		

SystemDate	hcfaAtcLib._sDT	
Year	UINT	2023
Month	USINT	11
Day	USINT	12
Hour	USINT	15
Minute	USINT	28
Sec	USINT	38
MilliS	UINT	340

6.13.28. DaysToMonth (天数 TO 月转换)

根据从 1 月 1 日起的天数，计算出该日为哪月

指令	名称	FB/FUN	图形表现	ST 表现
DaysTo Month	天数→月转换	FUN		Out:=DaysToMonth(Year,Days);

变量

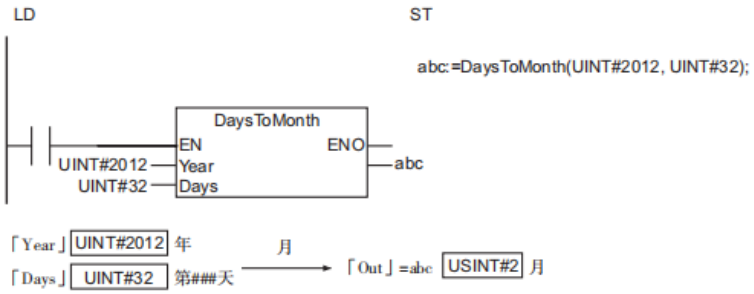
	名称	输入/输出	内容	有效范围	单位	初始值
Year	年	输入	公历年	1970~2554	年	1970
Days	天数		1 月 1 日起的天数	1~365“Year”为闰年时则为 1~365	日	1
Out	月	输出	月	1~12	月	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Year							○													
Days							○													
Out						○														

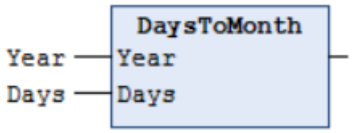
功能

根据从年 “Year” 的 1 月 1 日起的天数 “Days”，计算出该日为哪月。

“Year”=UINT#2012，“Days”=UINT#32 时的示例如下所示。



	FBD	ST
定义变量	VAR Year :UINT; Days :UINT; Out :USINT; END_VAR	

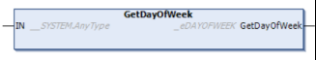
程序		Out:=DaysToMonth(Year:=Year , Days:=Days);
运行结果		

要点说明

- “Year” 的值超过有效范围时，不会发生异常，“Out” 的值为错误值。
- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- “Days” 的值超过有效范围时。

6. 13. 29. GetDayOfWeek (星期获取)

获取指定年月日的星期

指令	名称	FB/FUN	图形表现	ST 表现
GetDayOfWeek	星期获取	FUN		Out:=GetDayOfWeek(In);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	年月日	输入	年月日	遵从数据类型	年月日	(*)
Out	星期	输出	星期	_MON,_TUE, _WED,_THU, _FEI,_SAT, _SUN	星期	—

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	○		○	
Out	枚举体 _eDAYOFWEEK 枚举元素参阅功能说明																			

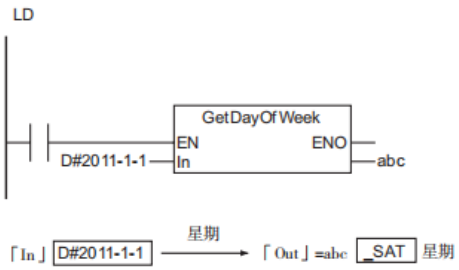
功能

获取年月日 “In” 表示年月日的星期。

“Out” 的数据类型为枚举体 _eDAYOFWEEK。枚举元素的含义如下所示。

枚举元素	含义
_MON	星期一
_TUE	星期二
_WED	星期三
_THU	星期四
_FRI	星期五
_SAT	星期六
_SUN	星期天

“In”=D#2011-1-1 时的示例如下所示。



ST

```
abc:=GetDayOfWeek(D#2011-1-1);
```

	FBD	ST
定义变量	VAR In :DATE; Out :_eDAYOFWEEK; END_VAR	
程序	In — GetDayOfWeek IN —	Out:=GetDayOfWeek(IN:=In);
运行结果	In DATE_AND_TIME DT#2011-1-1-0:0:0 Out _EDAYOFWEEK _SAT	

6. 13. 30. GetWeekOfYear (周获取)

计算指定年月日为当年的第几周。

指令	名称	FB/FUN	图形表现	ST 表现
GetWeekOfYear	周获取	FUN	GetWeekOfYear IN __SYSTEM.AnyType USINT GetWeekOfYear	Out:=GetWeekOfYear(In);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	年月日	输入	年月日	遵从数据类型	年月日	(*)
Out	周	输出	当年的第几周	1~54	周	—

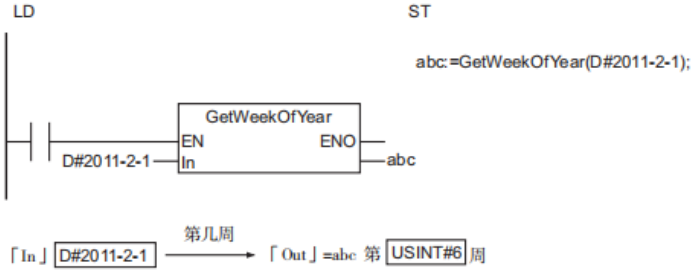
* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																	○		○	
Out						○														

功能

计算年月日 “In” 表示年月日为当年的第几周。以星期一到星期天为 1 周，在星期天变为星期一时对周数进行正计数。1 月 1 日必定是第 1 周。例如，假设 1 月 1 日为星期四，则从 1 月 1 日(星期四)到 1 月 4 日(星期天)为第 1 周；1 月 5 日(星期一)到 1 月 11 日(星期天)为第 2 周。

“In”=D#2011-2-1 时的示例如下所示。



	FBD	ST
定义变量	VAR In :DATE; Out :USINT; END_VAR	
程序	In — GetWeekOfYear / IN	Out:=GetWeekOfYear(IN:=In);
运行结果	In DATE D#2011-2-1 Out USINT 4	

6. 13. 31. DtToDateStruct (时刻分解)

将日期时间分解为 “ 年 ”、“ 月 ”、“ 日 ”、“ 时 ”、“ 分 ”、“ 秒 ”。

指令	名称	FB/FUN	图形表现	ST 表现
DtToDateStruct	时刻分解	FUN		Out:=DtToDateStruct(In, DateStruct);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	日期时刻	输入	日期时刻	遵从数据类型	年月日时分秒	DT#1970-1-1-0:0:0
Out	返回值	输出	始终为 TRUE	仅 TRUE	—	—
DateStruct	日期时刻		分解为“年”、“月”、“日”、“时”、“分”、“秒”的日期时刻	—		

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			○	
Out	○																			
DateStruct	结构体_sDT 详情参阅功能说明																			

功能

将日期时刻“In”分解为“年”、“月”、“日”、“时”、“分”、“秒”。

已分解日期时刻 “DateStruct” 的数据类型为结构体 _sDT。规格如下所示。

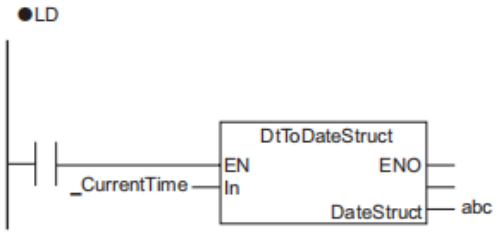
变量	名称	内容	数据类型	有效范围	单位	初始值
DateStruct	日期时刻	分解为“年”、“月”、“日”、“时”、“分”、“秒”的日期时刻	_sDt	—	—	—
Year	年	年	UINT	1970~2554	年	—
Month	月	月	USINT	1~12	月	
Day	日	日	USINT	1~31	日	
Hour	时	时	USINT	0~23	时	
Min	分	分	USINT	0~59	分	
Sec	秒	秒	USINT	0~59	秒	

“In”=DT#1970-1-1-12:12:12 时的示例如下所示。

	FBD	ST
定义变量	VAR In :DATE_AND_TIME; Out :_sDT; END_VAR	
程序	In — DtToDateStruct IN	Out:=DtToDateStruct(IN:=In);
运行结果	In DATE_AND_TIME DT#1970-1-1-12:12:12 Out _sDT Year UINT 1970 Month USINT 1 Day USINT 1 Hour USINT 12 Minute USINT 12 Sec USINT 12	

参考

组合分解的“ 年”、“月”、“日 ”、“ 时 ”、“ 分”、“ 秒”设为日期时间时，请使用“DateStructToDt 指令”。求得当前时刻的示例如下所示。



要点说明

- 在 ST 程序中使用本指令时，不使用返回值 “Out”。

6. 13. 32. DateStructToDt (时刻组合)

组合分解为 “ 年 ”、“ 月 ”、“ 日 ”、“ 时 ”、“ 分 ”、“ 秒 ” 的日期时刻

指令	名称	FB/ FUN	图形表现	ST 表现
DateStructToDt	时刻组合	FUN		Out:=DateStructToDt(In);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	日期时刻	输入	分解为“年”、“月”、“日”、“时”、“分”、“秒”的日期时刻	—	—	—
Out	日期时刻	输出	日期时刻	遵从数据类型	年月日时分秒	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In	结构体_sDT 详情参阅功能说明																			
Out																			○	

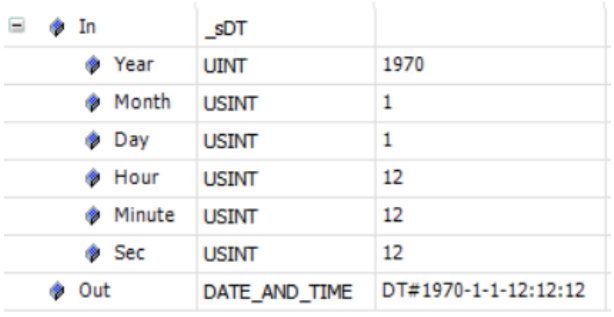
功能

组合分解为“年”、“月”、“日”、“时”、“分”、“秒”的日期时刻 “In”。

“In” 的数据类型为结构体 _sDT。规格如下所示。

变量	名称	内容	数据类型	有效范围	单位	初始值
DateStruct	日期时刻	分解为“年”、“月”、“日”、“时”、“分”、“秒”的日期时刻	_sDt	—	—	—
Year	年	年	UINT	1970~2554	年	—
Month	月	月	USINT	1~12	月	
Day	日	日	USINT	1~31	日	
Hour	时	时	USINT	0~23	时	
Min	分	分	USINT	0~59	分	
Sec	秒	秒	USINT	0~59	秒	

	FBD	ST
定义变量	VAR In :_sDT; Out :DATE_AND_TIME; END_VAR	
程序	In — DateStructToDt IN —	Out:=DateStructToDt(IN:=In);

运行结果	

In	_sDT	
Year	UINT	1970
Month	USINT	1
Day	USINT	1
Hour	USINT	12
Minute	USINT	12
Sec	USINT	12
Out	DATE_AND_TIME	DT#1970-1-1-12:12:12

要点说明

- 以下情况时会发生异常。ENO 变为 FALSE，“Out” 不变。
- “In” 的任一结构要素的值超过有效范围时。
- 处理结果超过 “Out” 的有效范围时。

6. 13. 33. TruncTime (时间舍去)

舍去 LTIME 型变量中小于指定单位的值

指令	名称	FB/FUN	图形表现	ST 表现
TruncTime	时间舍去	FUN		Out:=TruncTime(In, Accuracy);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	对象时间	输入	时间舍去的对象	遵从数据类型	ns	T#0s
Accuracy	舍去单位		不舍去的最小时间单位	(1，2，4，8)	—	0
Out	舍去后时间	输出	舍去后的时间	遵从数据类型	ns	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	LTIME	DATE	TOD	DT	STRING
In																○				
Accuracy		○																		
Out																○				

功能

舍去对象时间 “In” 中小于舍去单位 “Accuracy” 的值。舍去后的值保存在舍去后时间 “Out” 中。

“Accuracy” 的数据类型为 BYTE。其值的含义如下所示。

Accuracy	含义
1	秒
2	毫秒
4	微秒
8	纳秒

	FBD	ST
定义变量	VAR TIME_IN :LTIME; TIME_OUT :LTIME; Accuracy :BYTE; END_VAR	
程序	TIME_IN, Accuracy → TruncTime (In, Accuracy)	TIME_OUT:=TruncTime(In:=TIME_IN, Accuracy:=Accuracy);
运行结果	TIME_IN LTIME LTIME#12d20h38m31s111ms111us111ns TIME_OUT LTIME LTIME#12d20h38m31s Accuracy BYTE 1	

6. 13. 34. TruncDt (日期时刻舍去)

舍去 DT 型变量中小于指定单位的值

指令	名称	FB/FUN	图形表现	ST 表现
TruncDt	日期时刻舍去	FUN	TruncDt In DATE_AND_TIME BOOL TruncDt Accuracy _eSUBSEC DATE_AND_TIME Out	Out:=TruncDt(In, Accuracy);

变量

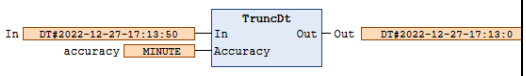
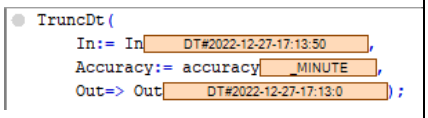
	名称	输入/输出	内容	有效范围	单位	初始值
In	对象时间	输入	日期时刻舍去的对象	遵从数据类型	年月日时分秒	DT#1970-1-1-0:0:0
Accuracy	舍去单位		不舍去的最小时间单位	_MILLISEC, _SEC, _MINUTE, _HOUR	—	_MILLISEC
Out	舍去后日期时刻	输出	舍去后的日期时刻	遵从数据类型	年月日时分秒	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																			○	
Accuracy	枚举体 _eSUBSEC 枚举元素参阅功能说明																			
Out																			○	

功能

舍去对象时间 “In” 中小于舍去单位 “Accuracy” 的值。舍去后的值保存在舍去后时间 “Out” 中。

“In” = DT#2022-12-27-17:13:50，"accuracy" = _MINUTE 时，示例如下所示。

	FBD	ST
定义变量	PROGRAM PLC_PRG VAR In :DT; accuracy :HCFA_OmronUitls._eSUBSEC; Out :DT; END_VAR	
程序		
运行结果		

“Accuracy” 的数据类型为枚举体 _eSUBSEC。枚举元素的含义如下所示。

枚举元素	含义
_MILLISEC	毫秒
_SEC	秒
_MINUTE	分
_HOUR	时

6. 13. 35. TruncTod (时刻舍去)

舍去 TOD 型变量中小于指定单位的值

指令	名称	FB/FUN	图形表现	ST 表现
TruncTod	时刻舍去	FUN		Out:=TruncTod(In, Accuracy);

变量

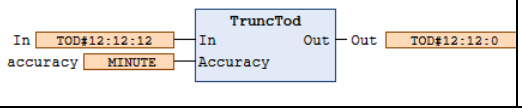
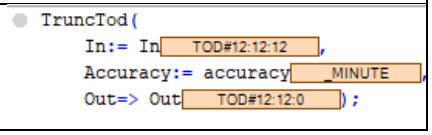
	名称	输入/输出	内容	有效范围	单位	初始值
In	对象时刻	输入	时刻舍去的对象	遵从数据类型	时分秒	TOD#0:0:0
Accuracy	舍去单位		不舍去的最小时间单位	_MILLISEC, _SEC, _MINUTE, _HOUR	—	_MILLISEC
Out	舍去后时刻	输出	舍去后的时刻	遵从数据类型	时分秒	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																		○		
Accuracy	枚举体 _eSUBSEC 枚举元素参阅功能说明																			
Out																		○		

功能

舍去对象时刻 “In” 中小于舍去单位 “Accuracy” 的值。舍去后的值保存在舍去后时刻 “Out” 中。

“In” = TOD#12:12:12，"accuracy" = _MINUTE 时，示例如下所示。

	FBD	ST
定义变量		
程序		

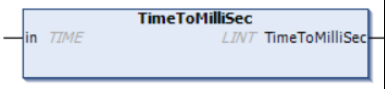
运行结果	表达式	类型	值
	In	TIME_OF_DAY	TOD#12:12:12
	accuracy	_ESUBSEC	_MINUTE
	Out	TIME_OF_DAY	TOD#12:12:0

“Accuracy” 的数据类型为枚举体 _eSUBSEC。枚举元素的含义如下所示。

枚举元素	含义
_MILLISEC	毫秒
_SEC	秒
_MINUTE	分
_HOUR	时

6. 13. 36. TimeToMilliSec (时间 TO 毫秒转换)

将时间转换为毫秒数

指令	名称	FB/FUN	图形表现	ST 表现
TimeToMilliSec	时间→毫秒转换	FUN		Out:=TimeToMilliSec(in:=);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	时间	输入	时间	遵从数据类型		T#0s
Out	毫秒数	输出	毫秒数	9223372036~9223372036	毫秒	—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																○				
Out													○							

功能

	FBD	ST
定义变量	VAR Intime :TIME; Out :LINT; END_VAR	
程序	Intime → TimeToMilliSec (in) → Out	Out :=TimeToMilliSec(in:= Intime);
运行结果	Intime TIME T#5m36s Out LINT 336000	

6. 13. 37. MilliSecToTime (毫秒 TO 时间转换)

将毫秒数转换为时间

指令	名称	FB/FUN	图形表现	ST 表现
MilliSecToTime	毫秒→时间转换	FUN	MilliSecToTime (in LINT → TIME MilliSecToTime)	Out:= MilliSecToTime (in:=);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	毫秒数	输入	毫秒数	9223372036~9223372036	毫秒	
Out	时间	输出	时间	遵从数据类型		T#0s

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In													○							
Out																○				

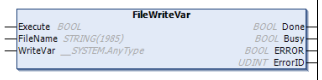
功能

	FBD	ST
定义变量	VAR Intime :LINT; Out :TIME; END_VAR	
程序	Intime — in MilliSecToTime — Out	Out :=MilliSecToTime(in:= Intime);
运行结果	Intime LINT 34567 Out TIME T#34s567ms	

6.14. SD 存储卡指令

6.14.1. FileWriteVar (变量文件写入)

以二进制格式将 1 个变量值写入 SD 存储卡内的指定文件。

指令	名称	FB/FUN	图形表现	ST 表现
FileWriteVar	变量文件写入	FB		FileWriteVar_instance(Execute, FileName, WriteVar, Done, Busy, Error, ErrorID);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Execute	功能块触发	输入	功能块触发	遵从数据类型	—	FALSE
FileName	指定文件名		写入文件名	最大 66 字节（65 个半角英数字字符+结尾 NULL 字符）	—	‘’
WriteVar	指定变量		写入变量			(*)
Done	完成	输出	完成	遵从数据类型	—	FALSE
Busy	运行中		运行中		—	
Error	错误		错误		—	
ErrorID	错误代码		错误代码		—	0

*省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileName																				○
WriteVar	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	也可指定枚举体、整个数组、数组的 1 个元素、整个结构体、结构体的 1 个结构要素																			
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

功能

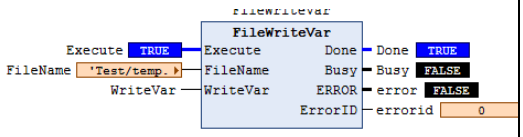
以二进制格式将 1 个变量 “WriteVar” 值写入 SD 存储卡内 “FileName” 指定的文件中。也可给 “WriteVar” 指定枚举体、整个数组、数组的 1 个元素、整个结构体、结构体的 1 个结构要素。SD 存储卡内不存在与 “FileName” 同名的文件时，新建文件。“FileName” 包含目录且 SD 存储卡内不存在该目录时，按不同的目录分别新建。

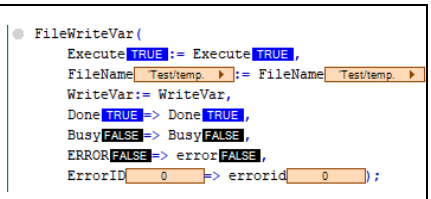
注：仅在指定目录的最底层不存在时，按不同的目录分别新建。

示例如下所示。将整个数组变量 WriteVar[ ]写入名为 ''Test/temp.bin'' 的文件。将 WriteVar[ ]设定为元素数量 5 的 BYTE 型数组变量。

	FBD	ST
定义变量		

```
PROGRAM PLC_PRG
VAR
    FileWriteVar        :FileWriteVar;
    Execute             :BOOL;
    FileName            :STRING :='Test/temp.bin';
    WriteVar            :ARRAY [0..4] OF BYTE;
    Done                :BOOL;
    Busy                :BOOL;
    error               :BOOL;
    errorid             :UDINT;
END_VAR
```

<table>
<tr><td>程序</td><td> 

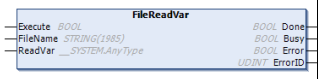

</td></tr>
<tr><td>运行结果</td><td>
<table>
<tr><th>表达式</th><th>类型</th><th>值</th></tr>
<tr><td>FileWriteVar</td><td>FileWriteVar</td><td></td></tr>
<tr><td>WriteVar</td><td>ARRAY [0..4] O...</td><td></td></tr>
<tr><td>WriteVar[0]</td><td>BYTE</td><td>1</td></tr>
<tr><td>WriteVar[1]</td><td>BYTE</td><td>5</td></tr>
<tr><td>WriteVar[2]</td><td>BYTE</td><td>3</td></tr>
<tr><td>WriteVar[3]</td><td>BYTE</td><td>7</td></tr>
<tr><td>WriteVar[4]</td><td>BYTE</td><td>22</td></tr>
<tr><td>errorid</td><td>UDINT</td><td>0</td></tr>
<tr><td>Busy</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>error</td><td>BOOL</td><td>FALSE</td></tr>
<tr><td>FileName</td><td>STRING</td><td>'Test/temp.bin'</td></tr>
<tr><td>Execute</td><td>BOOL</td><td>TRUE</td></tr>
<tr><td>Done</td><td>BOOL</td><td>TRUE</td></tr>
</table>
</td></tr>
</table>

要点说明

- 即使“Execute”的值变为 FALSE 或指令执行时间超过任务周期，本指令也将一直执行到最后。处理是否正常
- 结束，可通过“Done”的值是否变为 TRUE 来确认。“WriteVar”为整个结构体时，根据具体结构，可能在各成员之间插入调整用区域。
- 以下情况下将发生异常。“Error”为 TRUE。
- SD 存储卡不是可使用状态时。
- SD 存储卡为写保护中。
- SD 存储卡的剩余空间不足时。
- “FileName”的值为非法的文件名时。
- 超出可创建的文件数、目录数时。
- 已存在与“FileName”同名的文件且正在访问时。
- 已存在与“FileName”同名的文件且“OverWrite”的值为 FALSE 时。
- 已存在与“FileName”同名的文件且该文件为只读时。
- 正在访问 SD 存储卡时，因发生异常而无法访问时。

6.14.2. FileReadVar (变量文件读取)

以二进制格式读取 SD 存储卡内指定文件的值，并写入变量。

指令	名称	FB/FUN	图形表现	ST 表现
FileReadVar	变量文件读取	FB		FileReadVar_instance(Execute, FileName, ReadVar, Done, Busy, Error, ErrorID);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Execute	功能块触发	输入	功能块触发	遵从数据类型	—	FALSE
FileName	指定文件名		读取文件名	最大 66 字节（65 个半角英数字字符+结尾 NULL 字符）	—	‘’
ReadVar	指定变量		读取变量			(*)
Done	完成	输出	完成	遵从数据类型	—	FALSE
Busy	运行中		运行中		—	
Error	错误		错误		—	
ErrorID	错误代码		错误代码		—	0

*省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileName																				○
ReadVar	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	也可指定枚举体、整个数组、数组的 1 个元素、整个结构体、结构体的 1 个结构要素																			
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

功能

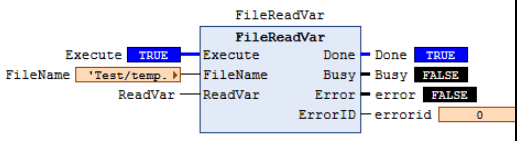
以二进制格式读取 SD 存储卡内 “FileName” 指定的文件内部值。将读取的值代入读取对象变量

“ReadVar”。也可给 “ReadVar” 指定枚举体、整个数组、数组的 1 个元素、整个结构体、结构体的 1 个结构要素。

示例如下所示，读取名为 ''Test/temp.bin'' 的文件内容。并写入数组变量 ReadVar[ ]。将 ReadVar[ ]设定为元素数量 5 的 BYTE 型数组变量。

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">

```
PROGRAM PLC_PRG
VAR
    FileReadVar         :FileReadVar;
    Execute             :BOOL;
    FileName            :STRING :='Test/temp.bin';
    ReadVar             :ARRAY [0..4] OF BYTE;
    Done                :BOOL;
    Busy                :BOOL;
    error               :BOOL;
    errorid             :UDINT;
END_VAR
```

</td></tr>
<tr><td>程序</td><td>


</td><td>

```
FileReadVar(
    Execute TRUE := Execute TRUE,
    FileName 'Test/temp. := FileName 'Test/temp.
    ReadVar:= ReadVar,
    Done TRUE => Done TRUE,
    Busy FALSE => Busy FALSE,
    Error FALSE => error FALSE,
    ErrorID 0 => errorid 0 );

RETURN
```

</td></tr>
<tr><td>运行结果</td><td colspan="2">

表达式	类型	值
FileReadVar	FileReadVar	
Execute	BOOL	TRUE
FileName	STRING	'Test/temp.bin'
ReadVar	ARRAY [0..4] O...	
ReadVar[0]	BYTE	1
ReadVar[1]	BYTE	5
ReadVar[2]	BYTE	3
ReadVar[3]	BYTE	7
ReadVar[4]	BYTE	22
Done	BOOL	TRUE
Busy	BOOL	FALSE
error	BOOL	FALSE
errorid	UDINT	0

</td></tr>
</table>

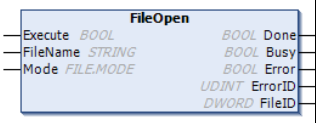
要点说明

- 即使“Execute”的值变为 FALSE 或指令执行时间超过任务周期，本指令也将一直执行到最后。处理是否正常

- 结束，可通过“Done”的值是否变为 TRUE 来确认。
- 指定文件的大小大于“ReadVar”的大小时，不会发生异常，将只读取与“ReadVar”的大小对应的数据。
- 指定文件的大小小于“ReadVar”的大小时，不会发生异常，将只读取与指定文件的大小对应的数据。 “ReadVar”
- 的剩余区域将保持执行本指令前的值。
- “ReadVar”为整个结构体时，根据具体结构，可能在各成员之间插入调整用区域。
- 以下情况下将发生异常。“Error”为 TRUE。
- SD 存储卡不是可使用状态时。
- “FileName”的值为非法的文件名时。
- “FileName”中指定的文件不存在时。
- 正在访问“FileName”中指定的文件时。
- 正在访问 SD 存储卡时，因发生异常而无法访问时。

6.14.3. FileOpen (文件打开)

打开 SD 存储卡内的指定文件

指令	名称	FB/FUN	图形表现	ST 表现
FileOpen	文件打开	FB		FileOpen_instance(Execute, FileName, Mode, Done, Busy, Error, ErrorID, FileID);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Execute	功能块触发	输入	功能块触发	遵从数据类型	—	FALSE
FileName	指定文件名		打开文件名	最大 66 字节（65 个半角英数字字符+结尾 NULL 字符）	—	‘’
Mode	打开方式		枚举体 HCFA_OmronUitls.file.MODE	MWRITE MREAD MRDWR MAPPD		(*)
Done	完成	输出	完成	遵从数据类型	—	FALSE

Busy	运行中		运行中			
Error	错误		错误			
ErrorID	错误代码		错误代码			0
FileID	文件 ID		文件句柄			0

*省略输入参数时，初始值不适用。编连时会发生异常。

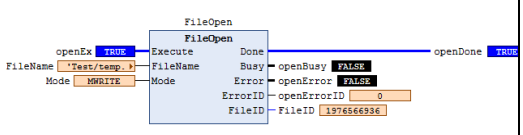
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileName																				○
Mode	枚举体 HCFA_OmronUitls.FILE.MODE 枚举元素参阅功能说明																			
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												
FileID				○																

功能

在“Mode”指定的模式下打开 SD 存储卡内 “FileName” 指定的文件。打开文件后，输出文件 ID“FileID”。

通过 FileRead 指令、FileWrite 指令等指定文件时使用 “FileID”。

打开名称为'Test/temp.txt'的文件，输出文件句柄到“FileID”中，示例如下所示。

	FBD	ST
定义变量		VAR FileOpen :FileOpen; openEx :BOOL; FileName :STRING :='Test/temp.txt'; Mode :HCFA_OmronUitls.file.MODE; openDone :BOOL; openBusy :BOOL; openError :BOOL; openErrorID :UDINT; FileID :DWORD;
程序		FileOpen(Execute TRUE := openEx TRUE, FileName 'Test/temp.' := FileName 'Test/temp.', Mode MWRITE := Mode MWRITE, Done TRUE => openDone TRUE, Busy FALSE => openBusy FALSE, Error FALSE => openError FALSE, ErrorID 0 => openErrorID 0, FileID 1976566936 => FileID 1976566936);

运行结果

表达式	类型	值
FileOpen	FileOpen	
openEx	BOOL	TRUE
FileName	STRING	'Test/temp.txt'
Mode	MODE	MWRITE
openDone	BOOL	TRUE
openBusy	BOOL	FALSE
openError	BOOL	FALSE
openErrorID	UDINT	0
FileID	DWORD	1976566936

“Mode” 的数据类型为枚举体 _eFOPEN_MODE。枚举元素的含义如下所示。

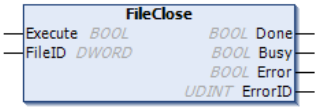
枚举元素	含义
MWRITE	写访问，文件将被覆盖或创建读取权限
MREAD	文件只会被打开用于读取读写访问权限
MRDWR	文件将被覆盖或创建
MAPPD	文件将以 WRiTE 模式打开，但写入的数据将被追加到文件的末尾

要点说明

- 即使“Execute”的值变为 FALSE 或指令执行时间超过任务周期，本指令也将一直执行到最后。处理是否正常结束，可通过“Done”的值是否变为 TRUE 来确认。
- 本指令需要在 FileSeek 指令、FileRead 指令、FileWrite 指令、FileGets 指令、FilePuts 指令前执行。
- 对于用本指令打开的文件，使用后请务必执行 FileClose 指令，关闭文件。
- 本指令完成时，在“FileID”中保存值。即“Done”的值从 FALSE 变为 TRUE 时。
- 在文件打开的状态下，拔出了 SD 存储卡时，文件将保持打开状态。但再次装入 SD 存储卡时，将无法对文
- 件进行读写。若要对文件进行读写，请重新打开文件。
- 以下情况下将发生异常。“Error”为 TRUE。
- SD 存储卡不是可使用状态时。
- SD 存储卡为写保护中。
- “Mode”的值为 AM_READ、AM_APPEND 或 AM_READ_PLUS，且“FileName”中指定的文件不存在时。
- “FileName”的值为非法的文件名时。
- 超出可创建的文件数、目录数时。
- 正在访问“FileName”中指定的文件时。
- “FileName”中指定的文件为只读时。 正在访问 SD 存储卡时，因发生异常而无法访问时。

6. 14. 4. FileClose (文件关闭)

关闭 SD 存储卡内的指定文件

指令	名称	FB/FUN	图形表现	ST 表现
FileCose	文件关闭	FB		FileClose_instance(Execute, FileID, Done, Busy, Error, ErrorID);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Execute	功能块触发	输入	功能块触发	遵从数据类型	—	FALSE
FileID	文件 ID		文件句柄			0
Done	完成	输出	完成			FALSE
Busy	运行中		运行中			
Error	错误		错误			
ErrorID	错误代码		错误代码			0

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileID				○																
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

功能

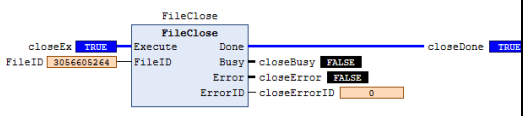
关闭 SD 存储卡内 “FileID” 指定的文件。

示例如下所示。关闭将变量 FileID 的值设定为文件 ID 的文件。

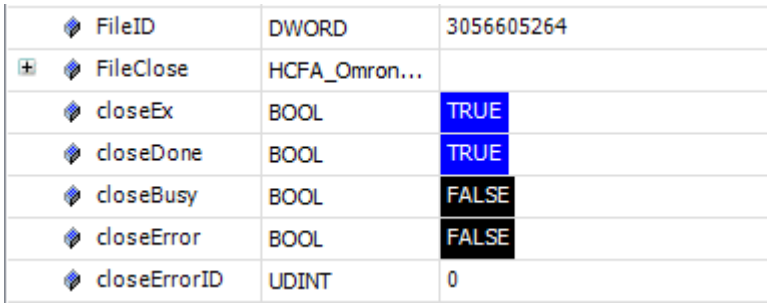
<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">

```
PROGRAM PLC_PRG
VAR
    FileOpen        :HCFA_OmronUitls.FileOpen;
    openEx          :BOOL;
    FileName        :STRING:='Test/Temp.txt';
    Mode            :HCFA_OmronUitls.FILE.MODE;
    openDone        :BOOL;
    openBusy        :BOOL;
    openError       :BOOL;
    openErrorID     :UDINT;
    FileID          :DWORD;

    FileClose       :HCFA_OmronUitls.FileClose;
    closeEx         :BOOL;
    closeDone       :BOOL;
    closeBusy       :BOOL;
    closeError      :BOOL;
    closeErrorID    :UDINT;
END_VAR
```

</td></tr>
<tr><td>程序</td><td> 
</td><td>

```
FileClose(
    Execute TRUE := closeEx TRUE,
    FileID 3056605264 := FileID 3056605264,
    Done TRUE => closeDone TRUE,
    Busy FALSE => closeBusy FALSE,
    Error FALSE => closeError FALSE,
    ErrorID 0 => closeErrorID 0 );RETURN
```

</td></tr>
<tr><td>运行结果</td><td colspan="2"> 
</td></tr>
</table>

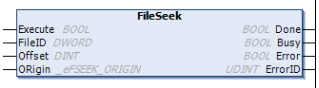
要点说明

- 禁止对同一文件连续多次执行本指令，否则会造成 PLC 运行异常，甚至死机。
- 即使“Execute”的值变为 FALSE 或指令执行时间超过任务周期，本指令也将一直执行到最后。处理是否正常
- 结束，可通过“Done”的值是否变为 TRUE 来确认。
- “FileID”需要事先执行 FileOpen 指令来获取。
- 对于用 FileOpen 指令打开的文件，使用后请务必执行本指令，关闭文件，否则会导致非预期的结果。
- 在文件打开的状态下，拔出了 SD 存储卡时，文件将保持打开状态。但再次装入 SD 存储卡时，将无法对文件
- 进行读写。若要对文件进行读写，请重新打开文件。
- 以下情况下将发生异常。“Error”为 TRUE。

- SD 存储卡不是可使用状态时。
- 正在访问“FileID”中指定的文件时。
- “FileID”中指定的文件不存在时。
- 正在访问 SD 存储卡时，因发生异常而无法访问时。

6.14.5. FileSeek (文件查找)

打开 SD 存储卡内的指定文件，设定文件位置指示器

指令	名称	FB/FUN	图形表现	ST 表现
FileSeek	文件查找	FB		FileSeek(Execute,FileID, Offset, Origin, Done, Busy, Error, ErrorID);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Execute	功能块触发	输入	功能块触发	遵从数据类型	—	FALSE
FileID	文件 ID		设定文件位置指示器的文件 ID		—	0
Offset	偏置	输出	从“Origin”起的偏置位置		字节	
Origin	基准位置		文件位置指示器的基准位置	_SEEK_SET, _SEEK_CUR , _SEEK_END	—	_SEEK_SET
Done	完成		完成	遵从数据类型	—	FALSE
Busy	运行中		运行中			
Error	错误		错误			
ErrorID	错误代码		错误代码			0

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileID				○																
Offset												○								
ORigin	枚举体_eFSEEK_ORIGIN 枚举元素参阅功能说明																			

Done	○																		
Busy	○																		
Error	○																		
ErrorID								○											

功能

为 SD 存储卡内文件 ID“FileID”指定的文件设定文件位置指示器。所谓文件位置指示器，是指执行 FileRead 指令及 FileWrite 指令等之后，开始读取、写入的文件内的位置。例如，希望从文件的起始部分执行读取时，通过 FileSeek 指令将文件位置指示器设定为文件的起始部分，然后执行 FileRead 指令。以基准位置 “Origin” 中添加偏置 “Offset” 的位置为文件位置指示器设定位置。

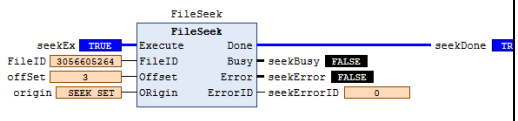
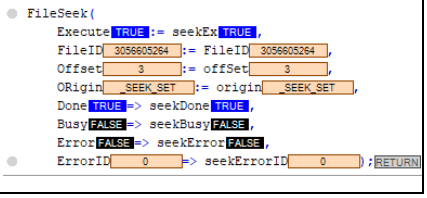
“Origin” 的数据类型为枚举体 _eFSEEK_ORIGIN。枚举元素的含义如下所示。

枚举元素	含义
_SEEK_SET	文件的起始部分
_SEEK_CUR	当前文件位置指示器的位置
_SEEK_END	文件的末尾

示例如下所示。将文件位置指示器设定至从文件起始部分起 3 字节的位置。

	FBD	ST
定义变量		

```
PROGRAM PLC_PRG
VAR
    FileID          :DWORD;
    FileSeek        :HCFA_OmronUitls.FileSeek;
    seekEx          :BOOL;
    offSet          :DINT;
    origin          :HCFA_OmronUitls._eFSEEK_ORIGIN;
    seekDone        :BOOL;
    seekBusy        :BOOL;
    seekError       :BOOL;
    seekErrorID     :UDINT;
```

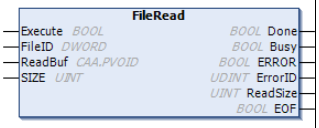
程序		


```
FileSeek(
    Execute TRUE := seekEx TRUE,
    FileID 3056605264 := FileID 3056605264,
    Offset 3 := offSet 3,
    ORigin SEEK_SET := origin SEEK_SET,
    Done TRUE => seekDone TRUE,
    Busy FALSE => seekBusy FALSE,
    Error FALSE => seekError FALSE,
    ErrorID 0 => seekErrorID 0);RETURN
```

运行结果	表达式 / 类型 / 值 / 准 FileID — DWORD — 3056605264 FileSeek — HCFA_Omron... — seekEx — BOOL — TRUE offSet — DINT — 3 origin — _EFSEEK_ORI... — _SEEK_SET seekDone — BOOL — TRUE seekBusy — BOOL — FALSE seekError — BOOL — FALSE seekErrorID — UDINT — 0

6. 14. 6. FileRead (文件读取)

读取 SD 存储卡内指定文件的数据

指令	名称	FB/ FUN	图形表现	ST 表现
FileRead	文件读取	FB		FileRead(Execute, FileID, ReadBuf, Size, Done, Busy, Error, ErrorID, ReadSize, EOF);

变量

	名称	输入/ 输出	内容	有效范围	单位	初始值
Execute	功能块触发	输入	功能块触发	遵从数据类型	—	FALSE
FileID	文件 ID		设定文件位置指示器的文件 ID	遵从数据类型	—	0
Size	读取元素数量		待读取的元素数量			1
ReadBuf	读取缓冲区		读取数据的写入对象	遵从数据类型	—	—
ReadSize	实际读取的元素数量	输出	实际读取的元素数量	遵从数据类型	—	—
EOF	文件结尾		判定是否到达文件结尾 TRUE ： 到达 FALSE：未到达			
Done	完成		完成	遵从数据类型	—	FALSE

Busy	运行中		运行中			
Error	错误		错误			
ErrorID	错误代码		错误代码			

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileID				○																
Size							○													
ReadBuf	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	也可知道一枚举体为元素的数值、以结构体为元素的数值																			
ReadSize							○													
EOF	○																			
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

功能

从 SD 存储卡内文件 ID“FileID” 指定的文件、文件位置指示器的某个位置起读取数据，并保存至读取缓存 ReadBuf[ ]。事先通过 FileSeek 指令，将文件位置指示器设定至任意位置。待读取的数据量为 (ReadBuf[ ] 数据类型的容量)×“Size”。即 ReadBuf[ ]“Size” 个的元素量。ReadBuf[ ] 允许为将枚举体设定为元素的数组、将结构体设定为元素的数组。

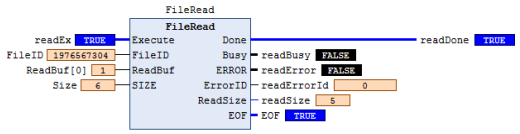
将实际读取的元素数量保存至 “ReadSize”。通常情况下，“Size” 的值与 “ReadSize” 的保持一致。如果从文件位置指示器的某个位置至文件结尾的数据量小于 “Size”，则不会发生异常，将至文件结尾的数据保存至 ReadBuf[ ]。此时，“ReadSize” 的值小于 “Size” 的值。此外，读取至文件结尾时，文件结尾 “EOF” 的值变为 TRUE。否则，“EOF” 的值变为 FALSE。

示例如下所示。读取名为 ''Test/temp.txt'' 的文件（先通过 FileOpen 指令以读访问模式打开文件获取文件句柄）。将 ReadBuf[ ]设定为元素数量 6 的 BYTE 型数组变量，尝试读取 6 个 BYTE 的数据。读取到上

文使用 FileWrite 指令写入到"Test/temp.txt"文件中的 byte 数组。同时读取到文件末尾，EOF 输出 TRUE。

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">

```
FileRead        :FileRead;
readEx          :BOOL;
ReadBuf         :ARRAY [0..5] OF BYTE;
Size            :UINT:=6;
readDone        :BOOL;
readBusy        :BOOL;
readError       :BOOL;
readErrorId     :UDINT;
readSize        :UINT;
EOF             :BOOL;
```

</td></tr>
<tr><td>程序</td><td> 
</td><td>

```
FileRead(
    Execute TRUE := readEx TRUE,
    FileID 1976567304 := FileID 1976567304,
    ReadBuf:= ReadBuf[0] 1,
    SIZE 6 := Size 6,
    Done TRUE => readDone TRUE,
    Busy FALSE => readBusy FALSE,
    ERROR FALSE => readError FALSE,
    ErrorID 0 => readErrorId 0,
    ReadSize 5 =>readSize 5,
    EOF TRUE => EOF TRUE);
```

</td></tr>
<tr><td>运行结果</td><td colspan="2">

FileRead	FileRead	
readEx	BOOL	TRUE
ReadBuf	ARRAY [0..5] O...	
ReadBuf[0]	BYTE	1
ReadBuf[1]	BYTE	5
ReadBuf[2]	BYTE	8
ReadBuf[3]	BYTE	35
ReadBuf[4]	BYTE	44
ReadBuf[5]	BYTE	0
Size	UINT	6
readDone	BOOL	TRUE
readBusy	BOOL	FALSE
readError	BOOL	FALSE
readErrorId	UDINT	0

</td></tr>
</table>

要点说明

- 即使“Execute”的值变为 FALSE 或指令执行时间超过任务周期，本指令也将一直执行到最后。处理是否正常结束，可通过“Done”的值是否变为 TRUE 来确认。
- “Size”的值超出 WriteBuf[] 的数组区域时，程序执行发生异常，“Error”变为 TRUE。
- “FileID”需要事先执行 FileOpen 指令来获取。
- WriteBuf[] 为以结构体为要素的数组时，根据具体结构，可能在各成员之间插入调整区域。
- 以下情况下将发生异常。“Error”为 TRUE。
- SD 存储卡不是可使用状态时。

- 正在访问“FileID”中指定的文件时。
- “FileID”中指定的文件不存在时。
- 正在访问 SD 存储卡时，因发生异常而无法访问时。
- “FileID”中指定的文件不是以可写入的模式打开时。

6.14.7. FileWrite (文件写入)

写 SD 存储卡内指定文件的数据

指令	名称	FB/FUN	图形表现	ST 表现
FileWrite	文件写入	FB		FileWrite(Execute, FileID, WriteBuf, Size, Done, Busy, Error, ErrorID, WriteSize);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Execute	功能块触发	输入	功能块触发	遵从数据类型	—	FALSE
FileID	文件 ID	输入	待写入文件的 ID	遵从数据类型	—	0
WriteBuf[ ] 数值	写入缓冲		待写入的数据			(*)
Size	写入元素数量		待写入的元素数量			1
Done	完成	输出	完成	遵从数据类型	—	FALSE
Busy	运行中		运行中			
Error	错误		错误			
ErrorID	错误代码		错误代码			
WriteSize	实际写入的元素数量		实际写入的元素数量			0

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileID				○																
WriteBuf[ ]数值	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○	○
	也可知道一枚举体为元素的数值、以结构体为元素的数值																			
Size							○													
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												
WriteSize							○													

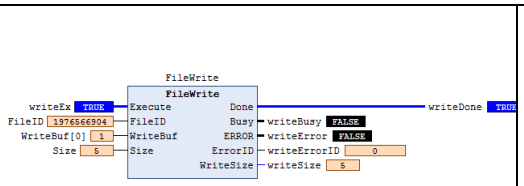
功能

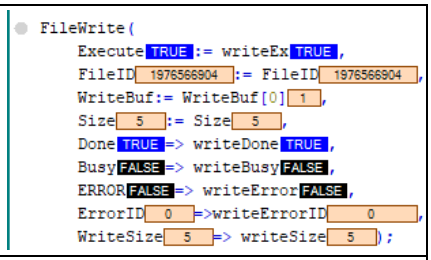
将数据写入 SD 存储卡内文件 ID“FileID” 指定的文件、文件位置指示器的某个位置。事先通过 FileSeek 指令，将文件位置指示器设定至任意位置。待写入的数据为写入缓存 WriteBuf[ ] 的内容。待写入的数据量为（WriteBuf[ ] 数据类型的容量 ）“Size”。即 WriteBuf[ ]“Size” 个的元素量。WriteBuf[ ] 允许为将枚举体设定为元素的数组、将结构体设定为元素的数组。将实际写入的数据容量输出至“WriteSize”。

示例如下所示。将整个数组变量 WriteBuf[ ]写入名为 ''Test/temp.txt'' 的文件（先通过 FileOpen 指令以写访问模式打开文件获取文件句柄）。将 WriteBuf[ ]设定为元素数量 5 的 BYTE 型数组变量。

	FBD	ST
定义变量		

```
PROGRAM PLC_PRG
VAR
    FileID          :DWORD;
    FileWrite       :HCFA_OmronUitls.FileWrite;
    writeEx         :BOOL;
    writeBuff       :ARRAY [0..4] OF BYTE;
    writeDone       :BOOL;
    writeBusy       :BOOL;
    writeError      :BOOL;
    writeErrorID    :BOOL;
    writeSize       :UINT;
```

<table>
<tr><td>程序</td><td> 


</td></tr>
<tr><td>运行结果</td><td>

FileWrite	FileWrite	
writeEx	BOOL	TRUE
WriteBuf	ARRAY [0..4] O...	
WriteBuf[0]	BYTE	1
WriteBuf[1]	BYTE	5
WriteBuf[2]	BYTE	8
WriteBuf[3]	BYTE	35
WriteBuf[4]	BYTE	44
writeDone	BOOL	TRUE
writeBusy	BOOL	FALSE
writeError	BOOL	FALSE
writeErrorID	UDINT	0
writeSize	UINT	5

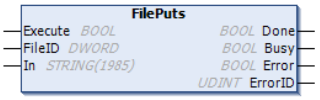
</td></tr>
</table>

要点说明

- 即使“Execute”的值变为 FALSE 或指令执行时间超过任务周期，本指令也将一直执行到最后。处理是否正
- 常结束，可通过“Done”的值是否变为 TRUE 来确认。
- “Size”的值超出 WriteBuf[] 的数组区域时，程序执行发生异常，“Error”变为 TRUE。
- “FileID”需要事先执行 FileOpen 指令来获取。
- WriteBuf[] 为以结构体为要素的数组时，根据具体结构，可能在各成员之间插入调整区域。
- 以下情况下将发生异常。“Error”为 TRUE。
- SD 存储卡不是可使用状态时。
- 正在访问“FileID”中指定的文件时。
- “FileID”中指定的文件不存在时。
- 正在访问 SD 存储卡时，因发生异常而无法访问时。
- “FileID”中指定的文件不是以可写入的模式打开时。

6.14.8. FilePuts (字符串写入)

将字符串写入 SD 存储卡内的指定文件

指令	名称	FB/FUN	图形表现	ST 表现
FilePuts	字符串写入	FB		FilePuts(Execute, FileID, In, Done, Busy, Error, ErrorID);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Execute	功能块触发	输入	功能块触发	遵从数据类型	—	FALSE
FileID	文件 ID		待写入文件的 ID			0
In	写入字符串		待写入的字符串			‘’
Done	完成	输出	完成	遵从数据类型	—	FALSE
Busy	运行中		运行中			
Error	错误		错误			
ErrorID	错误代码		错误代码			

* 省略输入参数时，初始值不适用。编连时会发生异常。

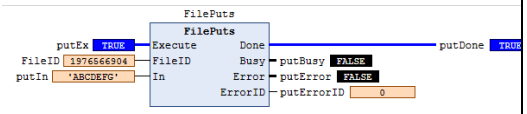
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileID				○																
In																				○
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

功能

针对文件 ID“FileID”中指定的 SD 存储卡的文件，将字符串“In”写入到文件位置指示符所在的位置。

文件位置指示符应事先通过 FileSeek 指令设定为任意的位置。

示例如下所示。往名为 ''Test/String.txt'' 的文件中写入“putIn”。（先通过 FileOpen 指令以写访问模式打开文件获取文件句柄）。“putIn” = ‘ABCDEFG’ 时，示例如下所示。

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">FilePuts :FilePuts;
putEx :BOOL;
putIn :STRING;
putDone :BOOL;
putBusy :BOOL;
putError :BOOL;
putErrorID :UDINT;</td></tr>
<tr><td>程序</td><td> 
</td><td>FilePuts(
Execute TRUE := putEx TRUE,
FileID 1976566904 := FileID 1976566904,
In 'ABCDEFG' := putIn 'ABCDEFG',
Done TRUE => putDone TRUE,
Busy FALSE => putBusy FALSE,
Error FALSE => putError FALSE,
ErrorID 0 => putErrorID 0);</td></tr>
<tr><td>运行结果</td><td colspan="2"> 
</td></tr>
</table>

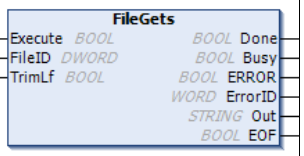
要点说明

- 即使“Execute”的值变为 FALSE 或指令执行时间超过任务周期，本指令也将一直执行到最后。处理是否正常结束，可通过“Done”的值是否变为 TRUE 来确认。
- “Size”的值超出 WriteBuf[] 的数组区域时，程序执行发生异常，“Error”变为 TRUE。
- “FileID”需要事先执行 FileOpen 指令来获取。
- WriteBuf[] 为以结构体为要素的数组时，根据具体结构，可能在各成员之间插入调整区域。
- 以下情况下将发生异常。“Error”为 TRUE。
- SD 存储卡不是可使用状态时。
- 正在访问“FileID”中指定的文件时。
- “FileID”中指定的文件不存在时。

- 正在访问 SD 存储卡时，因发生异常而无法访问时。
- “FileID”中指定的文件不是以可写入的模式打开时。

6.14.9. FileGets (字符串读取)

从 SD 存储卡内的指定文件读取 1 行字符串。

指令	名称	FB/FUN	图形表现	ST 表现
FileGets	字符串读取	FB		FileGets_instance(Execute, FileID, TrimLF, Done, Busy, Error, ErrorID, Out, EOF);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Execute	功能块触发	输入	功能块触发	遵从数据类型	—	FALSE
FileID	文件 ID		待写入文件的 ID			0
TrimLF	换行代码删除指定		待读取字符串的换行代码删指定 TRUE ： 删除 FALSE：不删除			FALSE
Out	读取字符串		读取的字符串	遵从数据类型	—	—
EOF	文件结尾	输出	判定是否到达文件结尾 TRUE ： 到达 FALSE：未到达			FALSE
Done	完成		完成			
Busy	运行中		运行中			
Error	错误		错误			
ErrorID	错误代码		错误代码			—

* 省略输入参数时，初始值不适用。编连时会发生异常。

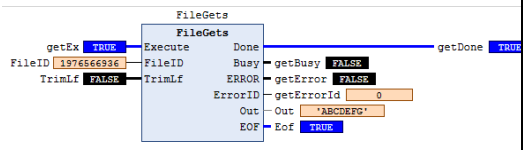
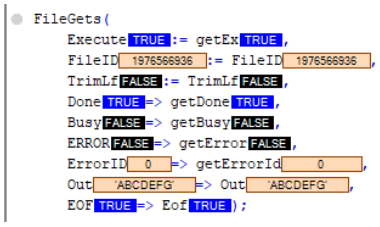
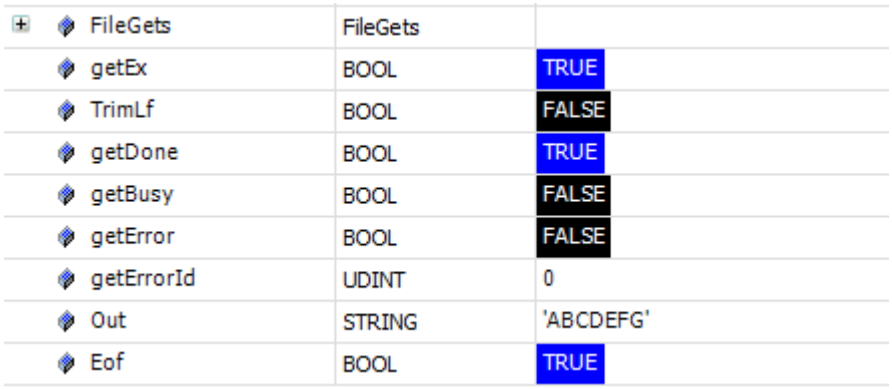
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileID				○																
TrimLF	○																			
Out																				○
EOF	○																			
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

功能

从 SD 存储卡内文件 ID“FileID” 指定的文件、文件位置指示器的某个位置起读取 1 行字符串。事先通过 FileSeek 指令，将文件位置指示器设定至任意位置。通过换行代码识别行与行的间隔。将读取的字符串写入读取字符串 “Out”。从 CR、LF、CR+LF 的 3 种类型中自动判别换行代码。换行代码删除指定“TrimLF” 的值变为 TRUE 时 删除字符串中的换行代码，然后写入 “Out”。此外，读取至文件结尾时，文件结尾 “EOF” 的值变为 TRUE。否则，“EOF” 的值变为 FALSE。

示例如下所示。读取名为 ''Test/String.txt'' 的文件，输出到“Out”中。（先通过 FileOpen 指令以读访问模式打开文件获取文件句柄）。

	FBD	ST
定义变量		FileGets :FileGets; getEx :BOOL; TrimLf :BOOL; getDone :BOOL; getBusy :BOOL; getError :BOOL; getErrorId :UDINT; Out :STRING; Eof :BOOL;

程序		
运行结果		

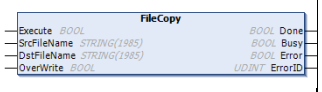
FileGets	FileGets	
getEx	BOOL	TRUE
TrimLf	BOOL	FALSE
getDone	BOOL	TRUE
getBusy	BOOL	FALSE
getError	BOOL	FALSE
getErrorId	UDINT	0
Out	STRING	'ABCDEFG'
Eof	BOOL	TRUE

要点说明

- 即使“Execute”的值变为 FALSE 或指令执行时间超过任务周期，本指令也将一直执行到最后。处理是否正常
- 结束，可通过“Done”的值是否变为 TRUE 来确认。1 行字符串的长度超出 1985 字节时，将把第 1985 个字节及其前面的字符串和末尾 NULL 字符保存到“Out”中。
- “FileID”需要事先执行 FileOpen 指令来获取。
- 以下情况下将发生异常。“Error”为 TRUE。
- SD 存储卡不是可使用状态时。
- 正在访问“FileID”中指定的文件时。
- “FileID”中指定的文件不存在时。
- 正在访问 SD 存储卡时，因发生异常而无法访问时。
- “FileID”中指定的文件不是以可写入的模式打开时。

6. 14. 10. FileCopy (文件复制)

复制 SD 存储卡内的指定文件。

指令	名称	FB/FUN	图形表现	ST 表现
FileCopy	文件复制	FB		FileCopy(Execute, SrcFileName, DstFileName, OverWrite, Done, Busy, Error, ErrorID);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Execute	功能块触发	输入	功能块触发	遵从数据类型	—	FALSE
SrcFile Name	复制源文件名		复制源文件名	最大 66 字节 (65 个半角英数字字符+结尾 NULL 字符)	—	‘’
In	写入字符串		待写入的字符串			
OverWrite	允许覆盖		TRUE ： 允许覆盖 FALSE：禁止覆盖	遵从数据类型		FALSE
Done	完成		完成			FALSE
Busy	运行中		运行中			
Error	错误		错误			
ErrorID	错误代码		错误代码			—

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
SrcFileName																				○
In																				○
OverWrite	○																			
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

功能

将 SD 存储卡内复制源文件名 “SrcFileName” 指定的文件复制至复制对象文件名 “DstFileName”。

SD 存储卡内已存在与 “DstFileName” 同名的文件时，根据允许覆盖 “OverWrite” 的值，执行下述处理。

“OverWrite”的值	处理
TRUE（允许覆盖）	覆盖至该文件
FALSE（禁止覆盖）	不覆盖至该文件时会发生异常

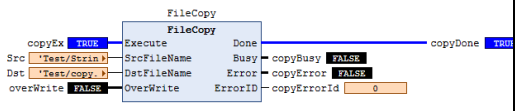
示例如下所示。拷贝名为 ''Test/String.txt'' 的文件，新文件命名为 ''Test/copy.txt''

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">

```
PROGRAM PLC_PRG
VAR
    FileCopy        :FileCopy;
    copyEx          :BOOL;
    Src             :STRING:='Test/String.txt';
    Dst             :STRING:='Test/copy.txt';
    overWrite       :BOOL;
    copyDone        :BOOL;
    copyBusy        :BOOL;
    copyError       :BOOL;
    copyErrorId     :UDINT;

END_VAR
```

</td></tr>
<tr><td>程序</td><td>



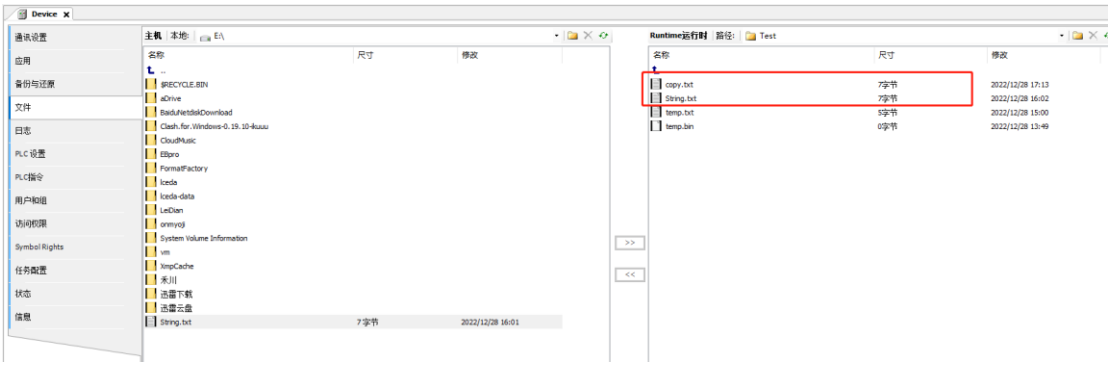
</td><td>

```
FileCopy(
    Execute TRUE := copyEx TRUE,
    SrcFileName 'Test/Strin' := Src 'Test/Strin',
    DstFileName 'Test/copy.' := Dst 'Test/copy.',
    OverWrite FALSE := overWrite FALSE,
    Done TRUE => copyDone TRUE,
    Busy FALSE => copyBusy FALSE,
    Error FALSE => copyError FALSE,
    ErrorID 0 => copyErrorId 0);
```

</td></tr>
<tr><td>运行结果</td><td colspan="2">

表达式	类型	值
FileCopy	FileCopy	
copyEx	BOOL	TRUE
Src	STRING	'Test/String.txt'
Dst	STRING	'Test/copy.txt'
overWrite	BOOL	FALSE
copyDone	BOOL	TRUE
copyBusy	BOOL	FALSE
copyError	BOOL	FALSE
copyErrorId	UDINT	0

</td></tr>
</table>



要点说明

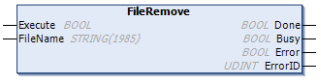
- 即使“Execute”的值变为 FALSE 或指令执行时间超过任务周期，本指令也将一直执行到最后。处理是否正常 结束，可通过“Done”的值是否变为 TRUE 来确认。
- 在文件打开的状态下，拔出了 SD 存储卡时，文件将保持打开状态。但再次装入 SD 存储卡时，

将无法对文件

- 进行读写。若要对文件进行读写，请重新打开文件。
- 以下情况下将发生异常。“Error”为 TRUE。
- SD 存储卡不是可使用状态时。
- SD 存储卡为写保护中。
- SD 存储卡的剩余空间不足时。
- “SrcFileName”中指定的文件不存在时。
- “SrcFileName”的值为非法的文件名时。
- “DstFileName”的值为非法的文件名时。
- 超出可创建的文件数、目录数时。
- 已存在与“DstFileName”同名的文件且正在访问时。
- 已存在与“DstFileName”同名的文件且“OverWrite”的值为 FALSE 时。
- 已存在与“DstFileName”同名的文件且该文件为只读时。
- 正在访问 SD 存储卡时，因发生异常而无法访问时。

6. 14. 11. FileRemove (文件删除)

删除 SD 存储卡内的指定文件。

指令	名称	FB/FUN	图形表现	ST 表现
FileRemove	文件删除	FB		FileRemove(Execute, FileName, Done, Busy, Error, ErrorID);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Execute	功能块触发	输入	功能块触发	遵从数据类型	—	FALSE
FileName	指定文件名		带删除的文件名	最大 66 字节 (65 个半角英数字字符+结尾 NULL 字符)	—	‘’
Done	完成	输出	完成	遵从数据类型	—	FALSE
Busy	运行中		运行中			
Error	错误		错误			
ErrorID	错误代码		错误代码			

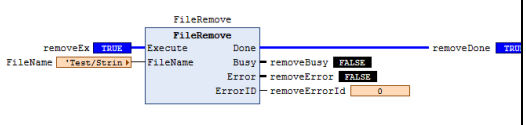
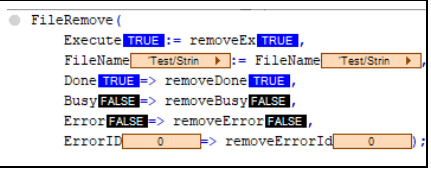
* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileName																				○
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

功能

删除 SD 存储卡内指定文件名 “FileName” 指定的文件。

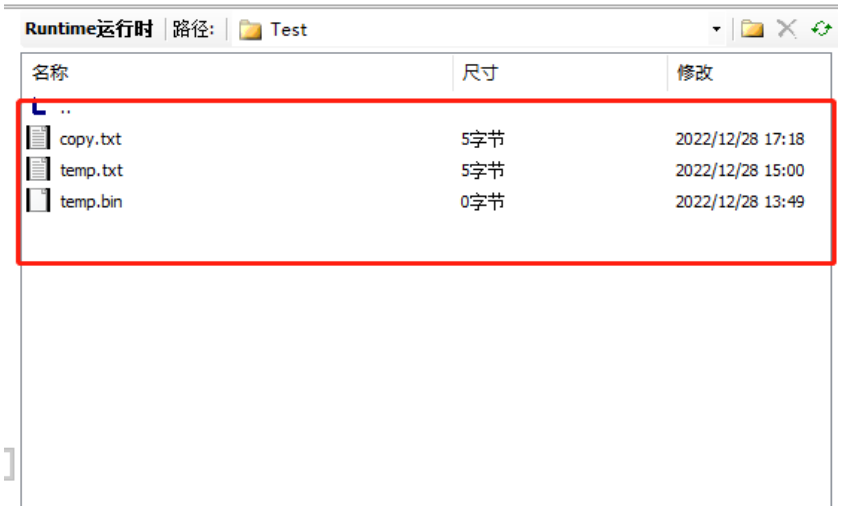
示例如下所示，PLC 中有名为 ''Test/String.txt'' 的文件，现调用 FIleRemove 指令删除该文件。

	FBD	ST
定义变量	PROGRAM PLC_PRG VAR FileRemove :FileRemove; removeEx :BOOL; FileName :STRING:='Test/String.txt'; removeDone :BOOL; removeBusy :BOOL; removeError :BOOL; removeErrorId :UDINT; END_VAR	
程序		

运行结果

表达式	类型	值
FileRemove	FileRemove	
removeEx	BOOL	TRUE
FileName	STRING	'Test/String.txt'
removeDone	BOOL	TRUE
removeBusy	BOOL	FALSE
removeError	BOOL	FALSE
removeErrorId	UDINT	0

名为 ''Test/String.txt'' 的文件已经被移除。

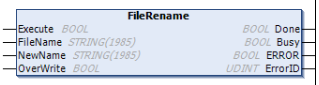


要点说明

- 即使“Execute”的值变为 FALSE 或指令执行时间超过任务周期，本指令也将一直执行到最后。处理是否正常结束，可通过“Done”的值是否变为 TRUE 来确认。
- 在文件打开的状态下，拔出了 SD 存储卡时，文件将保持打开状态。但再次装入 SD 存储卡时，将无法对文件
- 进行读写。若要对文件进行读写，请重新打开文件。
- 以下情况下将发生异常。“Error”为 TRUE。
- SD 存储卡不是可使用状态时。
- 正在访问“FileID”中指定的文件时。
- “FileID”中指定的文件不存在时。
- 正在访问 SD 存储卡时，因发生异常而无法访问时。
- “FileID”中指定的文件不是以可写入的模式打开时。

6. 14. 12. FileRename (文件名变更)

变更 SD 存储卡内指定文件和目录的名称。

指令	名称	FB/FUN	图形表现	ST 表现
FileRename	文件名变更	FB		FileRename(Execute, FileName, NewName, OverWrite, Done, Busy, Error, ErrorID);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Execute	功能块触发	输入	功能块触发	遵从数据类型	—	FALSE
FileName	原文件名		原文件名	最大 66 字节 (65 个半角英数字字符+结尾 NULL 字符)	—	‘’
NewName	变更后的文件名		变更后的文件名			
OverWrite	允许覆盖		TRUE ： 允许覆盖 FALSE：禁止覆盖	遵从数据类型		FLASE
Done	完成	输出	完成	遵从数据类型	—	FALSE
Busy	运行中		运行中			
Error	错误		错误			
ErrorID	错误代码		错误代码			

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
FileName																				○
NewName																				○
OverWrite	○																			
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

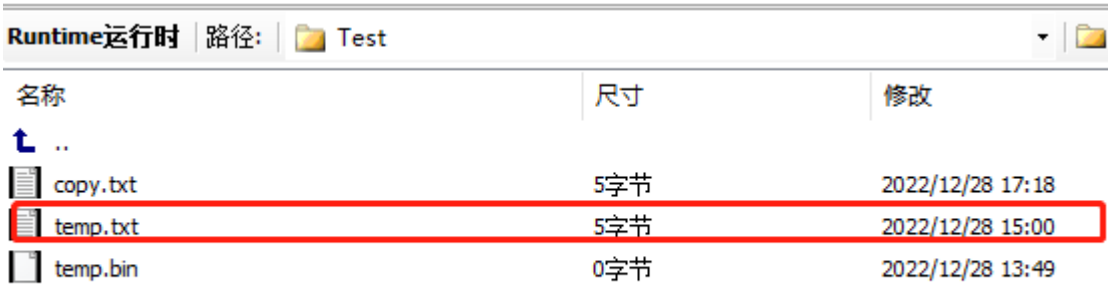
功能

将 SD 存储卡内原文件名“FileName”指定的文件及目录的名称变更为变更后的文件名“NewName”。

SD 存储卡内已存在与 “NewName” 同名的文件及目录时，根据允许覆盖 “OverWrite” 的值，执行下述处理。

“OverWrite”的值	处理
TRUE（允许覆盖）	覆盖至该文件
FALSE（禁止覆盖）	不覆盖至该文件时会发生异常

示例如下所示，文件夹中有名为 ‘Test/temp.txt’ 的文件，现调用 FileRename 指令修改文件名为 ‘Test/New.txt'.



	FBD	ST
定义变量		
程序		

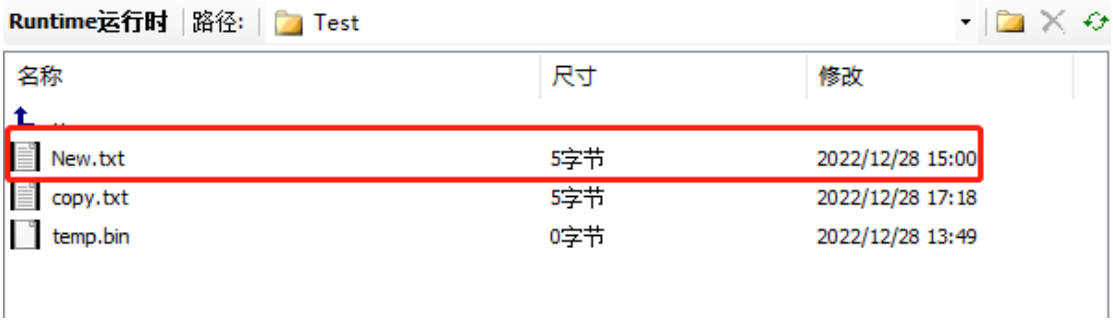
```
PROGRAM PLC_PRG
VAR
    FileRename      :FileRename;
    renameEx        :BOOL;
    FileName        :STRING:='Test/temp.txt';
    NewName         :STRING:='Test/New.txt';
    overWrite       :BOOL;
    renameDone      :BOOL;
    renameBusy      :BOOL;
    renameError     :BOOL;
    renameErrorId   :UDINT;
END_VAR
```

```
FileRename(
    Execute:= renameEx,
    FileName:= FileName,
    NewName:= NewName,
    OverWrite:= overWrite,
    Done=> renameDone,
    Busy=> renameBusy,
    ERROR=>renameError,
    ErrorID=> renameErrorId);
RETURN
```

运行结果

表达式	类型	值
FileRename	FileRename	
renameEx	BOOL	TRUE
FileName	STRING	'Test/temp.txt'
NewName	STRING	'Test/New.txt'
overWrite	BOOL	FALSE
renameDone	BOOL	TRUE
renameBusy	BOOL	FALSE
renameError	BOOL	FALSE
renameErrorId	UDINT	0

PLC 中，文件已经被改名为’ New.txt ‘。

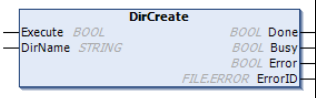


要点说明

- 即使“Execute”的值变为 FALSE 或指令执行时间超过任务周期，本指令也将一直执行到最后。处理是否正常结束，可通过“Done”的值是否变为 TRUE 来确认。
- “FileName”和“NewName”的目录不同时，文件将移动到“NewName”指定的目录下。
- 在文件打开的状态下，拔出了 SD 存储卡时，文件将保持打开状态。但再次装入 SD 存储卡时，将无法对文件
- 进行读写。若要对文件进行读写，请重新打开文件。
- 以下情况下将发生异常。“Error”为 TRUE。
- SD 存储卡不是可使用状态时。
- SD 存储卡为写保护中。
- “FileName”中指定的文件不存在时。
- “FileName”或“NewName”的值为非法的文件名时。
- 正在访问“FileName”中指定的文件时。
- 超出可创建的文件数、目录数时。
- 已存在与“NewName”同名的文件且“OverWrite”的值为 FALSE 时。
- 已存在与“DstFileName”同名的文件且“OverWrite”的值为 TRUE 且该文件为只读时。
- 正在访问 SD 存储卡时，因发生异常而无法访问时。

6. 14. 13. DirCreate (目录创建)

在 SD 存储卡内创建指定名称的目录。

指令	名称	FB/FUN	图形表现	ST 表现
DirCreate	目录创建	FB		DirCreate(Execute, DirName, Done, Busy, Error, ErrorID);

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Execute	功能块触发	输入	功能块触发	遵从数据类型	—	FALSE
DirName	创建目录名		待创建的目录名	最大 66 字节 (65 个半角英数字字符+结尾 NULL 字符)	—	‘’
Done	完成	输出	完成	遵从数据类型	—	FALSE
Busy	运行中		运行中			
Error	错误		错误			
ErrorID	错误代码		错误代码			—

* 省略输入参数时，初始值不适用。编连时会发生异常。

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
DirName																				○
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

功能

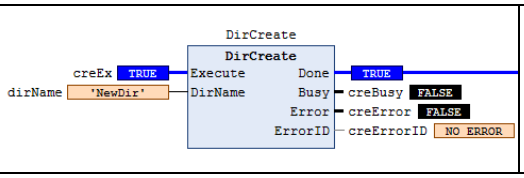
在 SD 存储卡内创建名称由创建目录名 “DirName” 指定的目录。

示例如下，调用指令 DirCreate 在 PLC 根目录下创建新目录‘NewDIr’。

<table>
<tr><th></th><th>FBD</th><th>ST</th></tr>
<tr><td>定义变量</td><td colspan="2">

```
PROGRAM PLC_PRG
VAR
    DirCreate       :DirCreate;
    creEx           :BOOL;
    dirName         :STRING :='NewDir';
    creDone         :BOOL;
    creBusy         :BOOL;
    creError        :BOOL;
    creErrorID      :HCFA_OmronUitls.FILE.ERROR;
```

</td></tr>
<tr><td>程序</td><td>



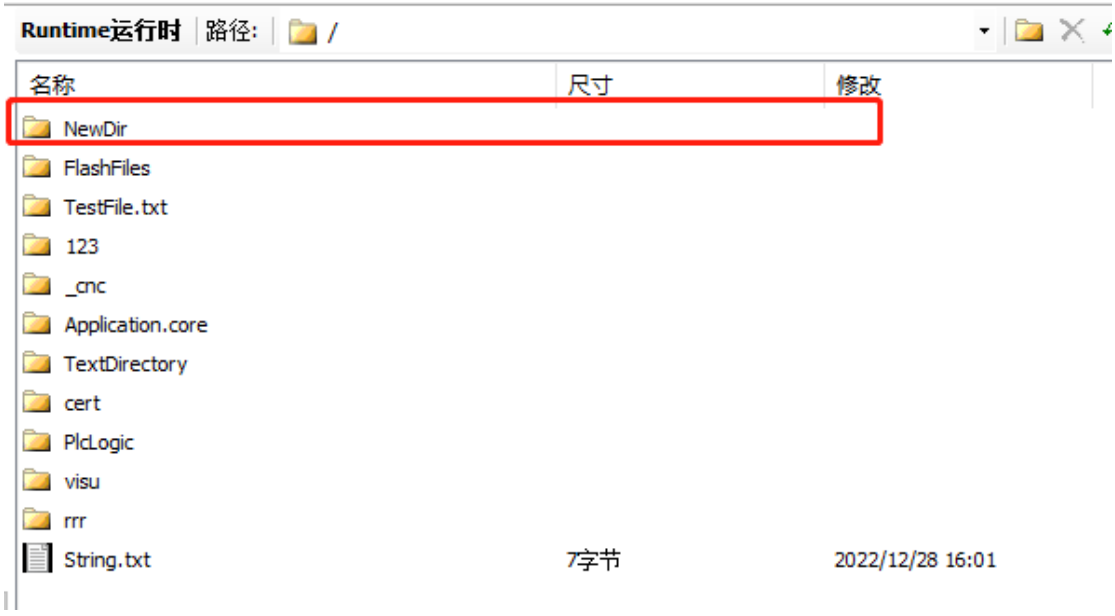
</td><td>

```
DirCreate(
    Execute TRUE := creEx TRUE,
    DirName 'NewDir' := dirName 'NewDir',
    Done TRUE => creDone TRUE,
    Busy FALSE => creBusy FALSE,
    Error FALSE => creError FALSE,
    ErrorID NO_ERROR => creErrorID NO_ERROR );
```

</td></tr>
<tr><td>运行结果</td><td colspan="2">

表达式	类型	值
DirCreate	DirCreate	
creEx	BOOL	TRUE
dirName	STRING	'NewDir'
creDone	BOOL	TRUE
creBusy	BOOL	FALSE
creError	BOOL	FALSE
creErrorID	ERROR	NO_ERROR

</td></tr>
</table>



要点说明

- 即使“Execute”的值变为 FALSE 或指令执行时间超过任务周期，本指令也将一直执行到最后。处理是否正常结束，可通过“Done”的值是否变为 TRUE 来确认。
- 在文件打开的状态下，拔出了 SD 存储卡时，文件将保持打开状态。但再次装入 SD 存储卡时，

将无法对文件

- 进行读写。若要对文件进行读写，请重新打开文件。
- 以下情况下将发生异常。“Error”为 TRUE。
- SD 存储卡不是可使用状态时。
- SD 存储卡为写保护中。
- SD 存储卡的剩余空间不足时。
- 超出可创建的目录数时。
- “DirName”中指定的目录已存在时。
- “DirName”的值为非法的目录名时。
- 正在访问 SD 存储卡时，因发生异常而无法访问时。

6. 14. 14. DirRemove (目录删除)

删除 SD 存储卡内的指定目录。

指令	名称	FB/FUN	图形表现	ST 表现
DirRemove	目录删除	FB		DirRemove(Execute, DirName, All, Done, Busy, Error, ErrorID)

变量

	名称	输入/输出	内容	有效范围	单位	初始值
Execute	功能块触发	输入	功能块触发	遵从数据类型	—	FALSE
DirName	删除目录名		待删除的目录名	最大 66 字节 (65 个半角英数字字符+结尾 NULL 字符)	—	‘’
ALL	所有指定		目录内存在文件/子目录时的指定 TRUE ：连同文件/子目录一起删除 FALSE：不删除	遵从数据类型		FASLE
Done	完成	输出	完成	遵从数据类型	—	FALSE
Busy	运行中		运行中			
Error	错误		错误			
ErrorID	错误代码		错误代码			—

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
Execute	○																			
DirName																				○
All	○																			
Done	○																			
Busy	○																			
Error	○																			
ErrorID								○												

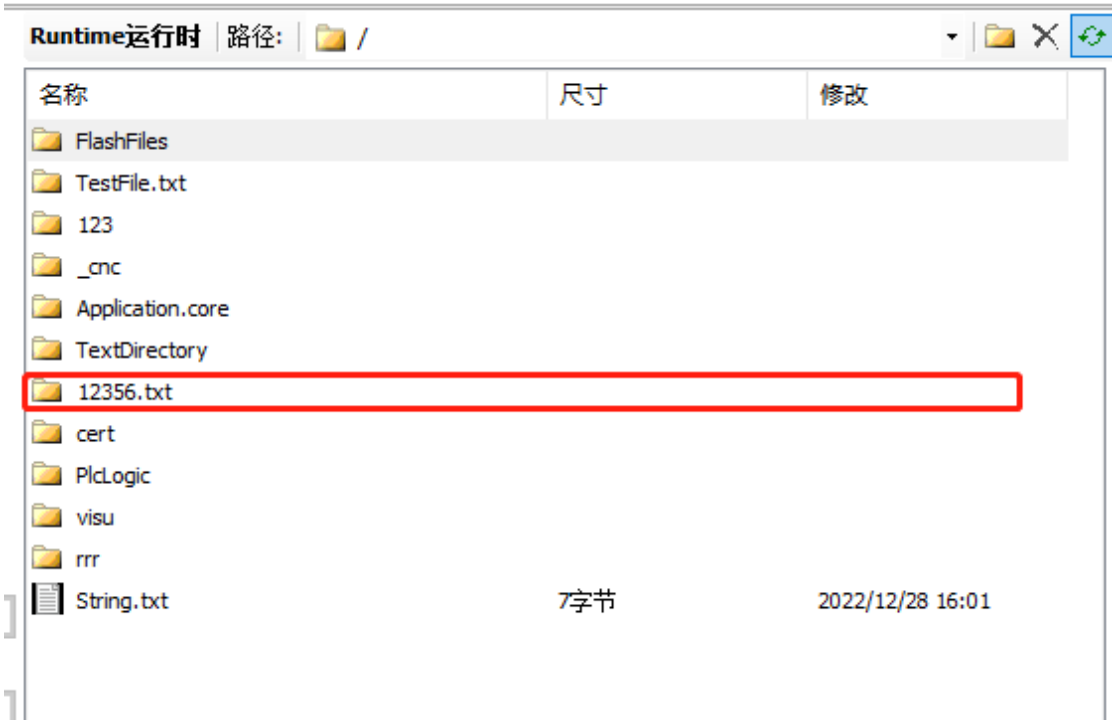
功能

删除 SD 存储卡内删除目录名 “DirName” 指定的目录。

指定目录内存在文件及子目录时，根据所有指定 “All” 的值，执行下述处理。

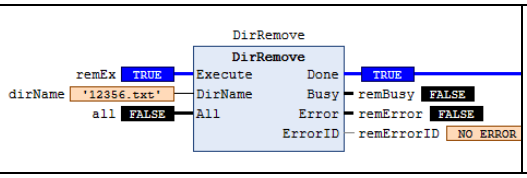
“ALL”的值	处理
TRUE	连同文件及子目录一起，删除指定的目录。
FALSE	不删除指定的目录，从而导致异常。。

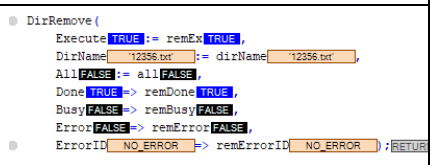
示例如下所示，PLC 根目录有名为‘12356.txt’的文件夹，现调用 DirRemove 指令删除该文件夹。

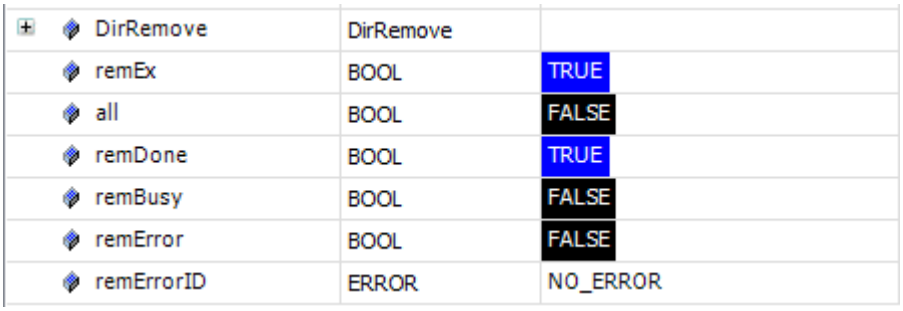


	FBD	ST
定义变量		
程序		
运行结果		

```
DirRemove       :DirRemove;
remEx           :BOOL;
all             :BOOL;
remDone         :BOOL;
remBusy         :BOOL;
remError        :BOOL;
remErrorID      :HCFA_OmronUitls.FILE.ERROR;
```







PLC 中，文件夹‘12356.txt’已经被移除。

要点说明

- 即使“Execute”的值变为 FALSE 或指令执行时间超过任务周期，本指令也将一直执行到最后。处理是否正常结束，可通过“Done”的值是否变为 TRUE 来确认。
- 在文件打开的状态下，拔出了 SD 存储卡时，文件将保持打开状态。但再次装入 SD 存储卡时，将无法对文件
- 进行读写。若要对文件进行读写，请重新打开文件。
- 以下情况下将发生异常。“Error”为 TRUE。

- SD 存储卡不是可使用状态时。
- SD 存储卡为写保护中。
- “All”的值为 TRUE 且“DirName”中指定的目录正在通过其他指令访问时。
- “All”的值为 FALSE 且“DirName”中指定的目录中存在文件或目录时。
- “DirName”中指定的目录为只读时。
- “DirName”中指定的文件不存在时。
- 正在访问 SD 存储卡时，因发生异常而无法访问时。

6.15. 16 进制字符转换指令

6.15.1. HexStringToNum_ (16 进制字符串 TO 整数)

FC_ByteToStrHex：用 Byte 表示的十六进制数值数转换单个 16 进制字符

FC_StrHexToByte：单个 16 进制字符转换为十进制数，用 Byte 表示

HexStringToNum_DINT：16 进制字符串转换为 DINT 型整数

HexStringToNum_INT：16 进制字符串转换为 INT 型整数

HexStringToNum_LINT：16 进制字符串转换为 LINT 型整数

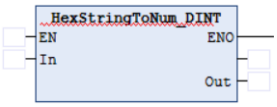
HexStringToNum_SINT：16 进制字符串转换为 SINT 型整数

HexStringToNum_UDINT：16 进制字符串转换为 UDINT 型整数

HexStringToNum_UINT：16 进制字符串转换为 UINT 型整数

HexStringToNum_ULINT：16 进制字符串转换为 ULINT 型整数

HexStringToNum_USINT：16 进制字符串转换为 USINT 型整数

指令	名称	FB/FUN	图形表现	ST 表现
HexStringTo Num_***	16 进制字符 TO 整数	FUN		HexStringToNum_DINT(In:= , Out=>);

变量

	名称	输入/输出	内容	有效范围	单位	初始值

In	16 进制字符串	输入		遵从数据类型	—	—
Out	整数	输输出			—	—

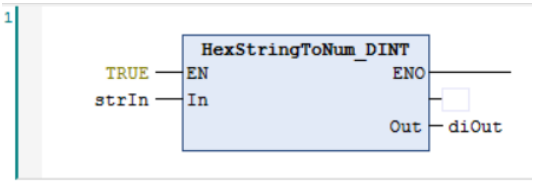
	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOl	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In																				○
Out						○	○	○	○	○	○	○	○							

功能

将 16 进制格式的字符串“In”转换为整数，指令名称根据具体数据类型变化。以 HexStringToNum_DINT

指令举例如下图：

LD:



ST:

```
strIn       :STRING;
diOut       :DINT;

HexStringToNum_DINT(In:=strIn , Out=>diOut );
```

strIn:=

					1	1	1

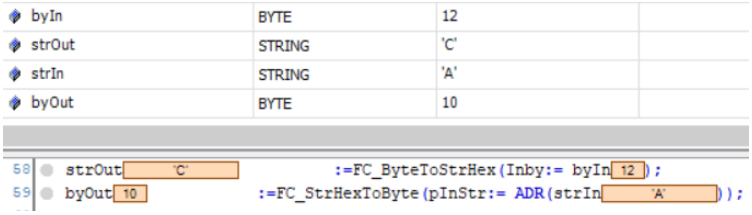
diOut:=DINT#273

strIn:=

				-	1	1	1

diOut:=DINT#-273

FC_ByteToStrHex 函数、FC_StrHexToByte 函数举例如下：



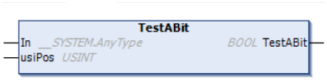
要点说明

- 转换结果超出“OUT”有效范围，不会显示异常，显示数值将会是无效数值。
- 输入非 16 进制表示的错误字符，不会显示异常，显示数值将会是无效数值。

6.16. 时序输入输出指令

6.16.1. TestABit (位测试)

输出位列中指定位的值。

指令	名称	FB/FUN	图形表现	ST 表现
TestABit	位测试	FUN		Out :=TestABit(In:= ,usiPos:=);

变量

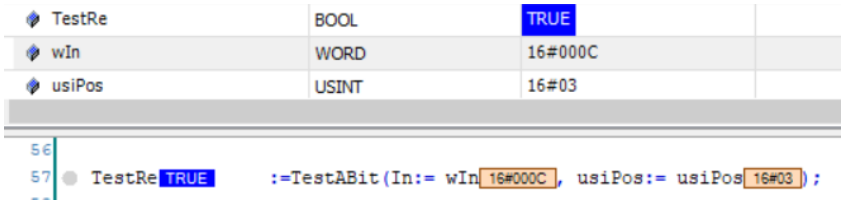
	名称	输入/输出	内容	有效范围	单位	初始值
In	位列	输入	位列	遵从数据类型		(*)
usiPos	位位置		指定位的位置	位数，从 0 位开始		0
Out	返回值		TestABit 的返回值	遵从数据类型		

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
usiPos						○														
Out	○																			

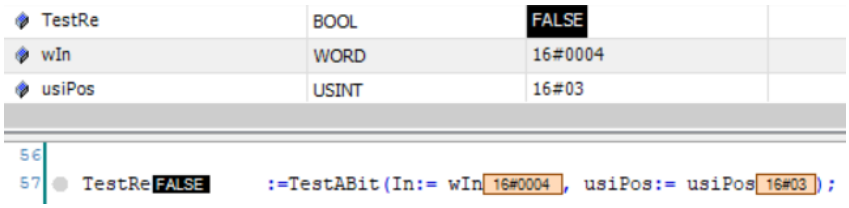
功能

将位列“In”的“Pos”位置上的值代，通过 TestABit 返回 TRUE/FALSE.

“In”输入 WORD 型数据，数据值 16#C(2#1100)，“usiPos”读取位 3，返回值 TRUE。



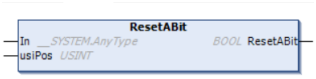
“In”输入 WORD 型数据，16#4(2#0100) ，“usiPos”读取位 3，返回值 FALSE。



6. 16. 2. SetABit/ResetABit (1 位设置/复位)

SetABit：将位列数据的指定位设置为 TRUE

ResetABit：将位列数据的指定位设置为 FALSE

指令	名称	FB/FUN	图形表现	ST 表现
SetABit	1 位设置	FUN		Out := SetABit(In:= , usiPos:=);
ResetABit	1 位复位	FUN		Out := ResetABit(In:= , usiPos:=);

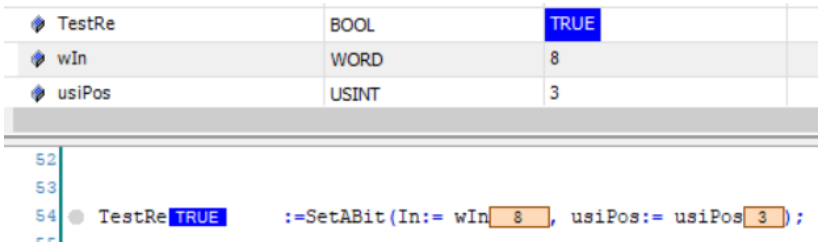
变量

	名称	输入/输出	内容	有效范围	单位	初始值
In	位列	输入	位列	遵从数据类型		(*)
usiPos	位位置		指定位的位置	位数，从 0 位开始		0
Out	返回值		TestABit 的返回值	遵从数据类型		

	布尔	位串				整数								实数		时刻、持续时间、日期、字符串				
	BOOL	BYTE	WORD	DWORD	LWORD	USINT	UINT	UDINT	ULINT	SINT	INT	DINT	LINT	REAL	LREAL	TIME	DATE	TOD	DT	STRING
In		○	○	○	○															
usiPos						○														
Out	○																			

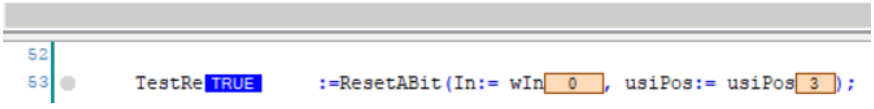
功能

通过 SetABit 函数，将 wIn 第 3 位设置为 TRUE，结果如下图：



通过 ResetABit 函数，将 wIn 第 3 位设置为 FALSE，结果如下图

TestRe	BOOL	TRUE
wIn	WORD	0
usiPos	USINT	3



要点说明

- 运行完成时，函数返回值为 TRUE

7. Standard (标准库)

7.1. CheckDevice（功能组）

用于检查控制器是否为浙江禾川科技产品。以下几种功能类似，可根据自身习惯选择使用功能块或者是函数。

7.1.1. 功能块 FB_CheckPAC (FB)

名称	FB_CheckPAC (PAC 检测)		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	
		FB_CheckPAC(eResult=>);	

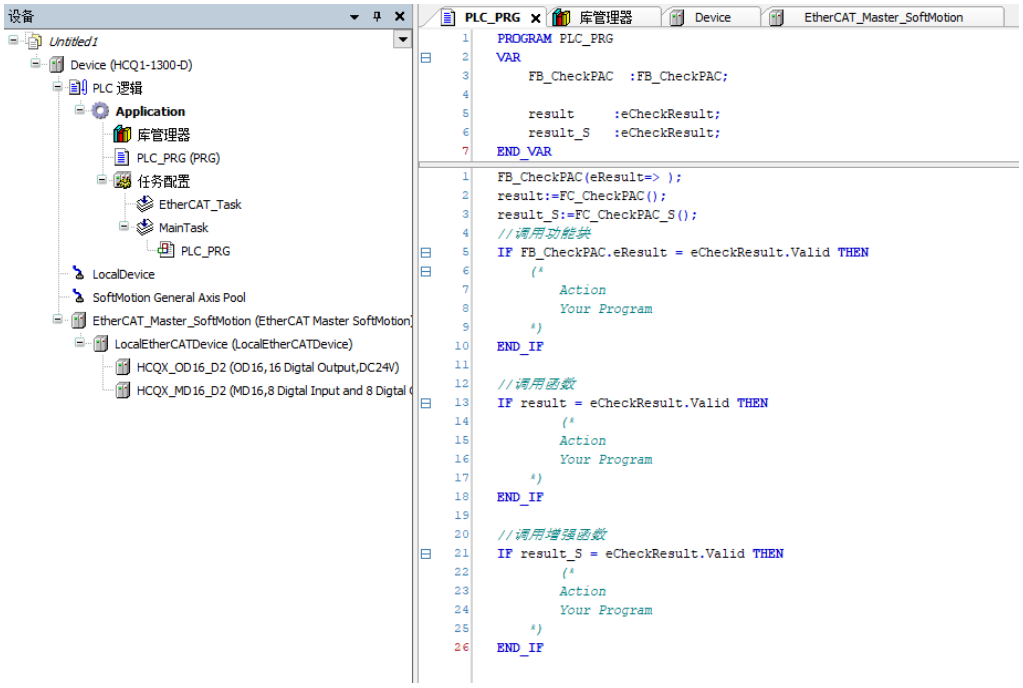
7.1.2. 函数形式

名称	内容
FC_CheckPAC (FUN)	检查禾川控制器标准函数
FC_CheckPAC_S (FUN)	检查禾川控制器增强函数,如果为非禾川控制器，会启用故障机制。

7.1.3. eCheckResult (ENUM)

名称	类型	值	内容
NotCalled	INT	0	未被调用
Checking	INT	1	检查中
Valid	INT	2	控制器有效
Invalid	INT	3	控制器无效

7.1.4. 使用举例



三个方法效果相同，请根据习惯选用

7.2. LockMachine（功能组）

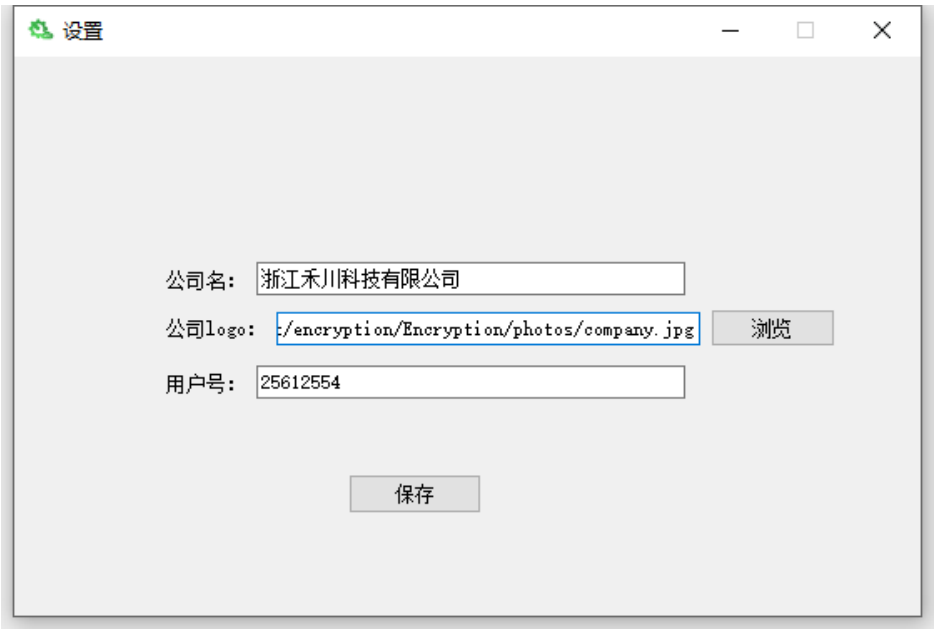
需要禾川加密软件配合 CODESYS 对 PLC 机器进行解锁。

7.2.1. 主要功能介绍

7.2.1.1. 界面介绍

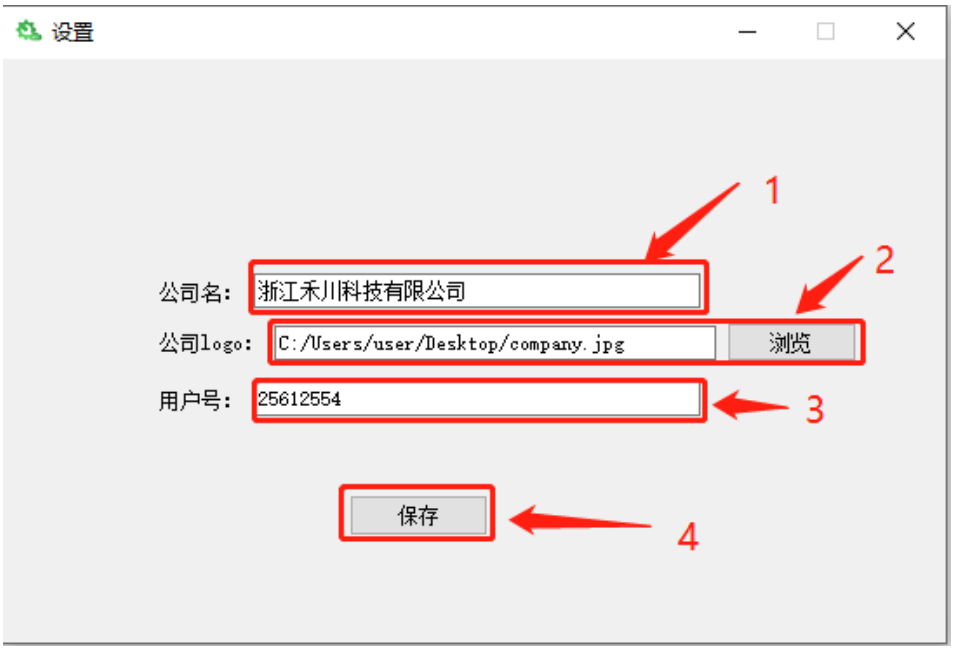
主界面

设置界面

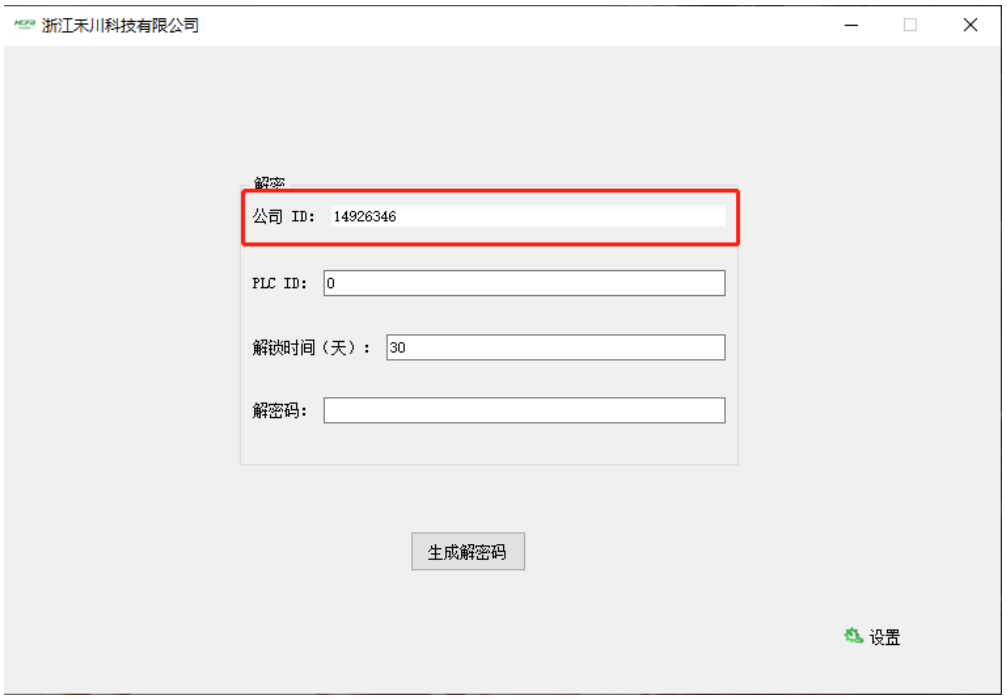


7.2.1.2. 软件初始化设置

初次使用时，推荐修改软件名称和图标，并且需要输入用户号信息，此处以浙江禾川科技有限公司为例，点击【设置】在打开的设置界面，输入公司名、公司 logo 图片和用户号共三项信息，点击【保存】即可完成修改，随后主界面会随机生成公司 ID。



修改完成后主界面如图所示：



7.2.1.3. 获取解密码和解锁 PLC

在主界面的“PLC ID”文本框内输入 CODESYS 连接 PLC 后随机生成的 PLC ID，设置解锁天数。

此处举例 CODESYS 软件中功能块生成的 PLC ID 为 55604，需要解锁 30 天。

点击【生成解密码】按钮，在“解密码”栏就会生成一个解密码，此处生成的是 233492974.

保存解密码，只需解密码输入到 CODESYS 的 FB_LockMachine 功能块的 KEY 中，触发 bCheckKey 的上升沿信号，即可自动识别解密码是否正确，如果正确就会将 PLC 解锁 30 天。30 天后 PLC 自动锁定，届时需要重新解锁。如下图

7.2.2. CODESYS 端配合使用说明

7.2.2.1. 功能块介绍

FB_LockMachine

用于判断 PLC 是否锁定和解锁 PLC 的功能块，下面对该功能块做个简单介绍

ST 表现：

```
FB_LockMachine(
        bCheckKey:=Execute ,        →触发信号
        Key:= KEY,                  →KEY 用于输入解锁软件生成的解密码
        bKeyCorrect=> ,             →解密码正确信号
        bKeyWrong=> ,               →解密码错误信号
        MachineValid=> ,            →机器可用状态信号
        bUnLocked=> ,               →机器永久解锁信号
        PLCID=>PLCID ,              →连接 PLC 后随机生成 ID 码，输入到解锁软件中用于生成解密码
        dRemainingDay=>lastDay);    →机器剩余可用天数
```

FB_getRTCDate

FB_getRTCDate(

Enable:= , →触发信号

dwDate=> , →日期

wYear=> , →年

wMonth:= , →月

wDay:= , →日

wHour:= , →时

wMinute:= , →分

wSecond:= , →秒

bError=>); →报错信号

FB_setRTCDate

用于设定 PLC 系统时间的功能块，下面对该功能块做个简单介绍

ST 表现：

FB_setRTCDate(

Execute:= , →触发信号

wYear:= , →年

wMonth:= , →月

wDay:= , →日

wHour:= , →时

wMinute:= , →分

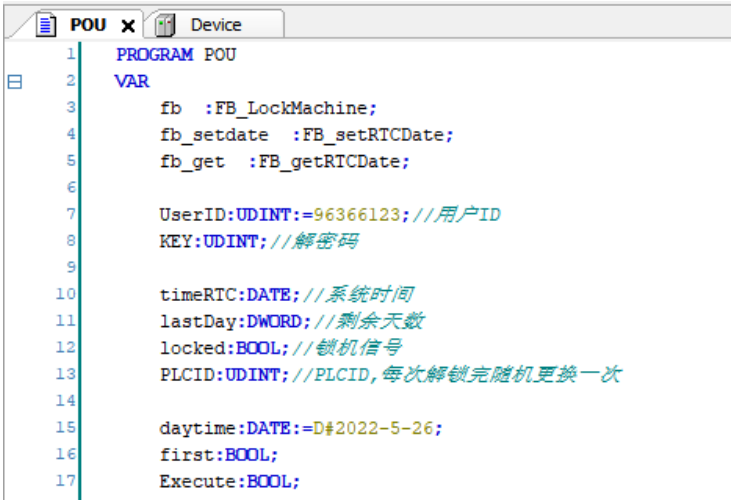
wSecond:= , →秒

Done=> ,); →时间设置完成信号

7. 2. 2. 2. 添加 CODESYS 工程

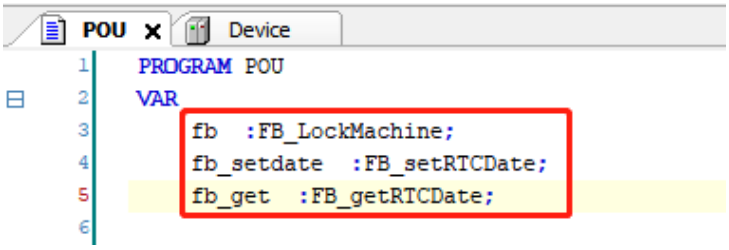
新建 CODESYS 工程文件，在 POU 中调用和声明 FB_LockMachine，FB_getRTCDate 和 FB_setRTCDate

共三个功能块，为了方便调试程序，这里给出示例程序中声明的一些变量供作参考：

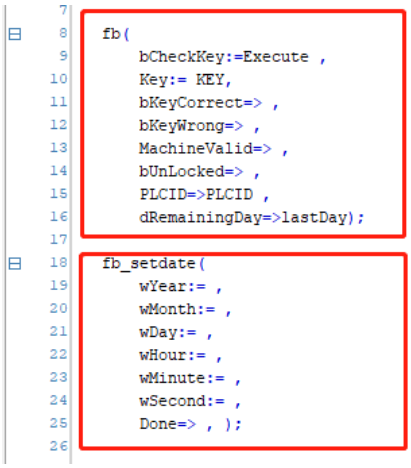


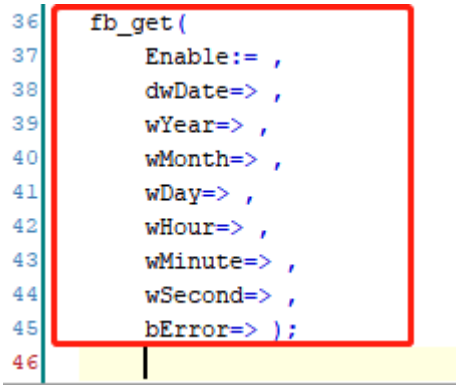
声明功能块

示例工程声明和调用如下图所示



调用功能块



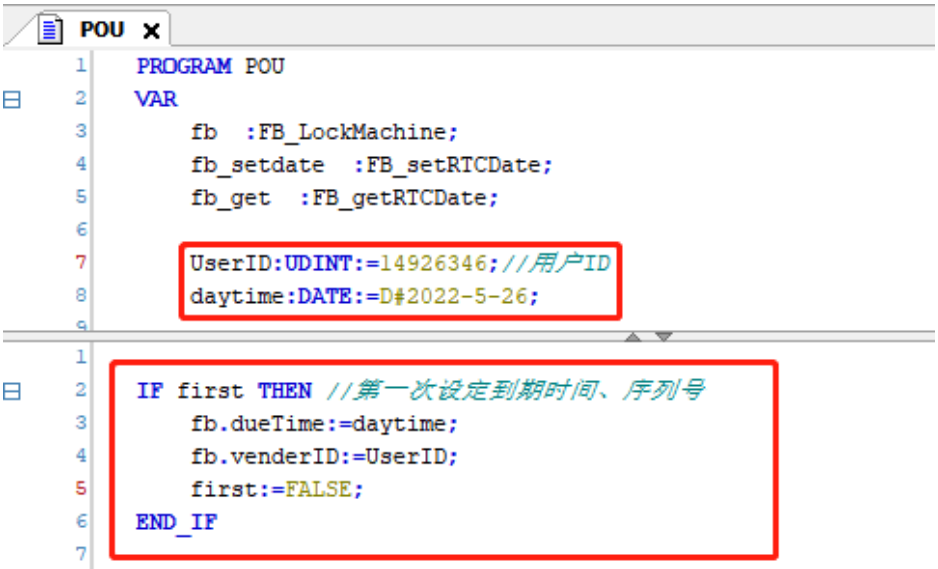


首次使用时，需要添加到期时间和公司 ID 的设定函数

首次使用需要向功能块中写入 daytime 和 UserID 两个数据。

示例程序，在 POU 中定义两个变量 UserID（UDINT）和 daytime（DATE），利用这两个变量将需要设定

的数值传入到功能块的 fb.venderID 和 fb.dueTime 两个引脚中，具体程序写法可参考下图。

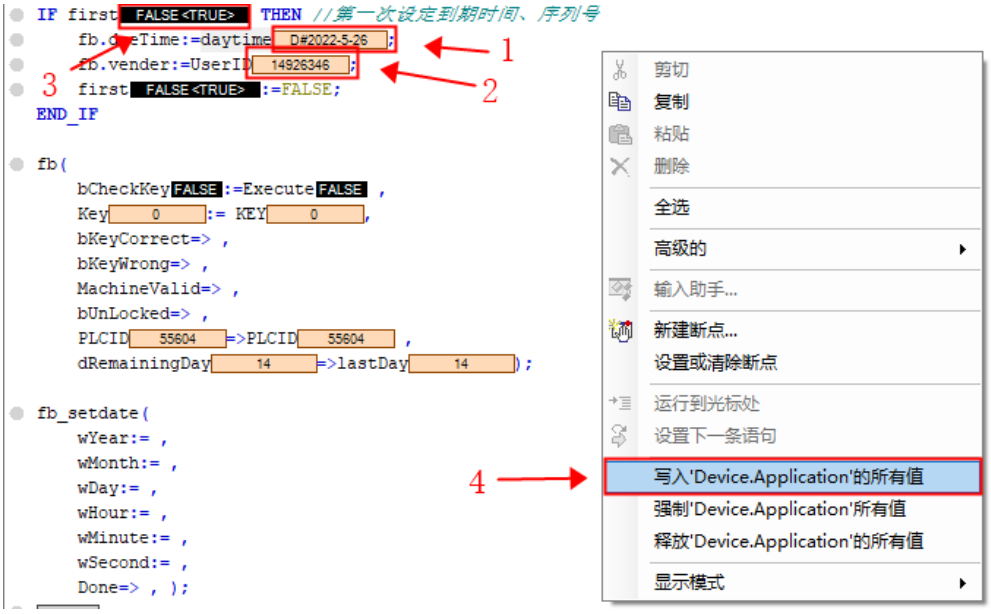


点击登陆，将程序下载到 PLC 中，运行程序。

7.2.2.3. 详细使用方法说明

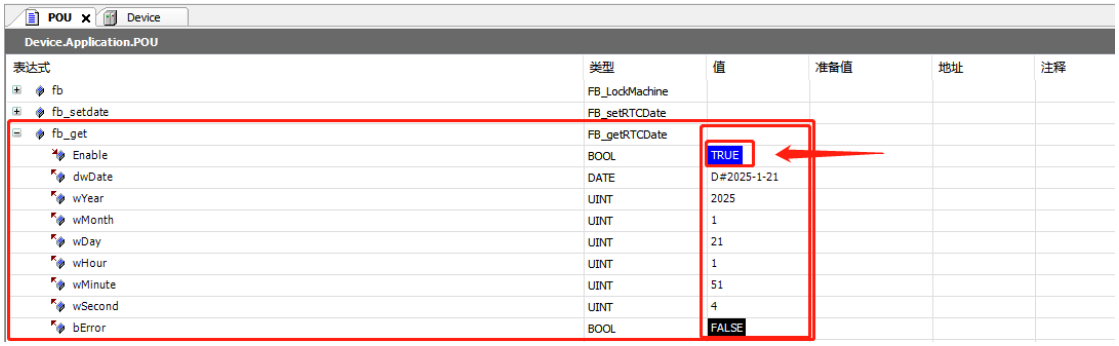
首次使用

首次使用需要使用到 IF 语句写入“公司 ID”（公司 ID 由解锁软件主界面中随机生成的）和设置“第一次到期时间”，此处举例到期时间为：D#2022-5-26，公司 ID：14926346。在示例程序中，只要写入这两项信息后触发 first 即可完成写入。



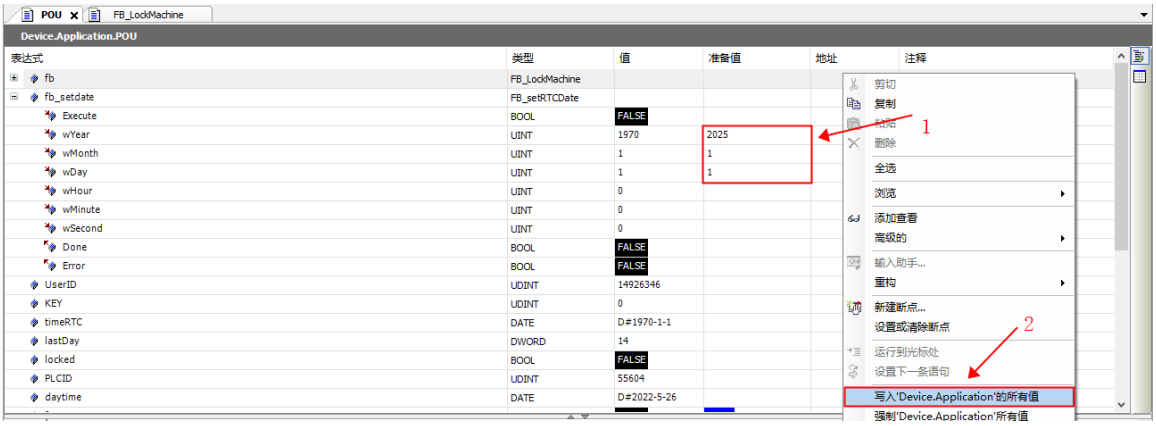
利用 FB_getRTCDate 功能块读取 PLC 系统时间

在 FB_getRTCDate 功能块中，触发 Enable 引脚，即可读取 PLC 的系统时间，示例程序中，读取到的系统时间为 D#2025-1-21。

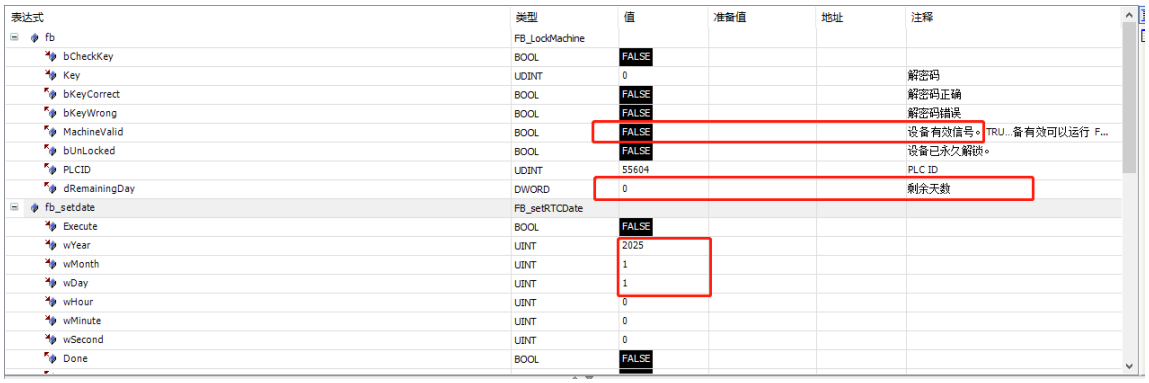


利用 FB_setRTCDate 功能块修改 PLC 系统时间

此处举例修改当前系统时间为 2025-1-1。

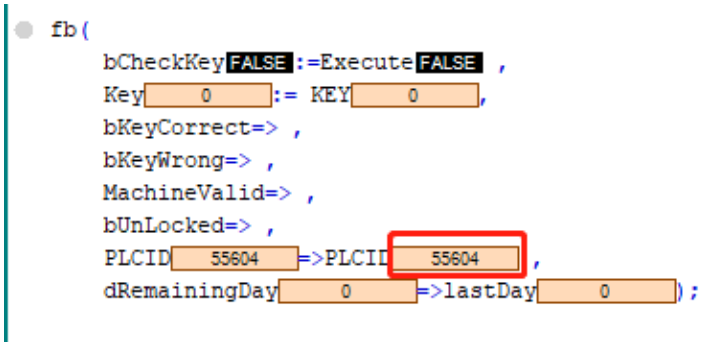


修改完成后可以看到，FB_LockMachine 功能块的设备有效信号识别为 FALSE，剩余天数为 0，PLC 已经被锁定，无法使用。



利用 FB_LockMachine 解锁 PLC

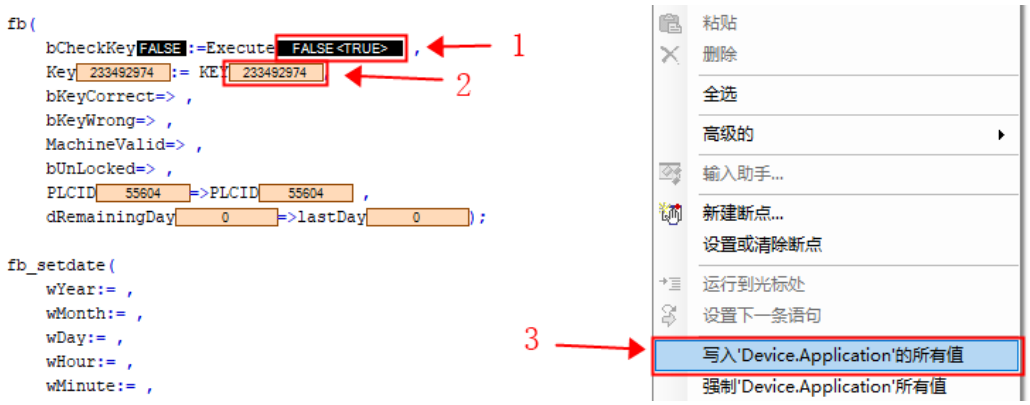
连接 PLC，运行程序后，功能块自动生成 PLC ID，如下图所示：



在解锁软件中输入 PLC ID 码生成解密码：

在上文第 8.2.1.3 章中已经举例过，对 PLC ID 为 55604 的 PLC 机器生成了一个 30 天的解密码

233492974。我们将该密码输入到功能块的 KEY 一项中，触发功能块的 bCheckKEY 引脚的上升沿信号，完成激活。



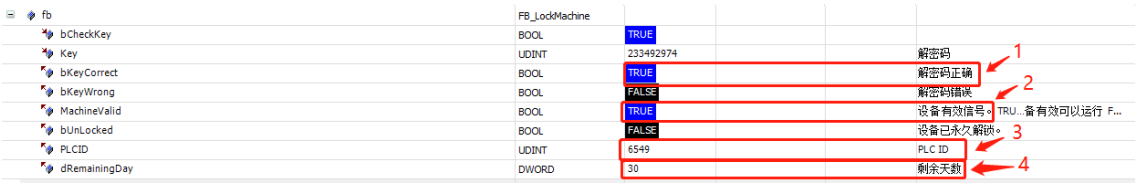
可以看到功能块有 4 处改变

【1】提示解密码正确信号为 TRUE。（解密码错误时无法激活解密码正确信号为 FALSE，解密码错误信号为 TRUE）

【2】设备有效信号为 TRUE，设备解锁。

【3】PLC ID 更新。（每次解锁后都会更新一个新的随机 PLC ID）

【4】剩余天数更新。（此处激活 30 天，剩余时间更新为 30 天）



至此，就完成了一次 PLC 的解锁，当 PLC 到期之后，重复操作一次即可。

7.2.3. 常见问题

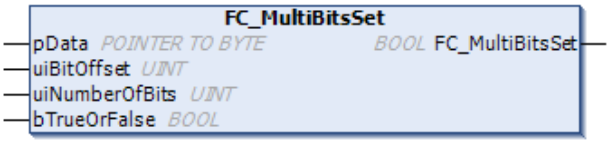
7.2.3.1. 系统时间设置误操作导致锁机

请谨慎使用 FB_setRTCDate 功能块来设置系统时间，需要注意输入的日期不可向前修改，例如读取到的系统时间为 2022-5-29，则修改系统时间为 2022-5-28 会导致 PLC 锁机，需要重新激活机器。

7.3. FC_MultiBitsSet （FUN）

连续 bit 状态设置，用于给多个 bit 设置为 TRUE 或者 FALSE.

变量

名称	FC_MultiBitsSet		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	
		FC_MultiBitsSet(pData:= , uiBitOffset:= , uiNumberOfBits:= , bTrueOrFalse:=);	

（1）输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
pData	起始地址	POINTER TO BYTE			起始地址
uiBitOffset	偏移量	UINT		0	Bit 偏移量，指定从第几个 Bit 开始操作
uiNumberOfBits	操作数	UINT		1	需要操作的 Bit 数量
bTrueOrFalse	目标值	BOOL	TRUE、FALSE	FALSE	每个 Bit 写入 TRUE 还是 FALSE

（2）输出变量

输出变量	名称	数据类型	有效范围	内容
FC_MultiBitsSet	完成	BOOL	TRUE、FALSE	TRUE:功能块执行完成

要点说明

- pData 是指针类型，对实际物理地址进行取值时，不要使用类似%QX10.3，而是用%QB10，因为%QX10.3 指向的地址实际还是%QX10.0。容易产生迷惑
- 功能块运行模式调用即开始实时修改，设定 bTrueOrFalse 引脚的值，即为当前设定地址与数量的 Bit 的值。

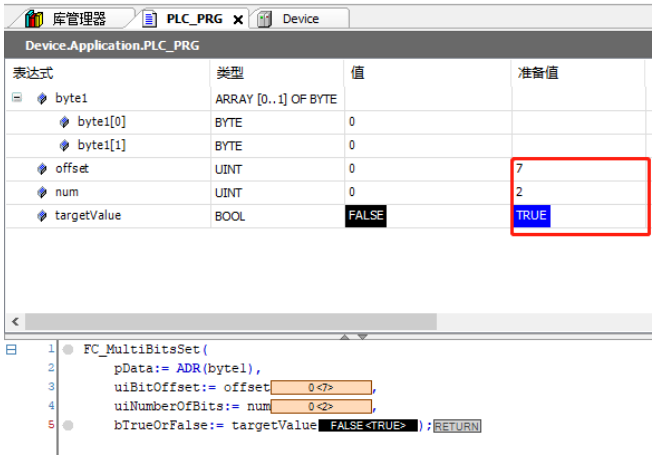
7.3.1. 使用举例 1 （对虚拟地址进行修改）

【1】定义一个 BYTE 数组和相应的 UINT 变量、BOOL 变量，调用函数

```
PROGRAM PLC_PRG
VAR
    bytel     :ARRAY [0..1] OF BYTE;
    offset    :UINT;
    num       :UINT;
    targetValue :BOOL;
END_VAR
```

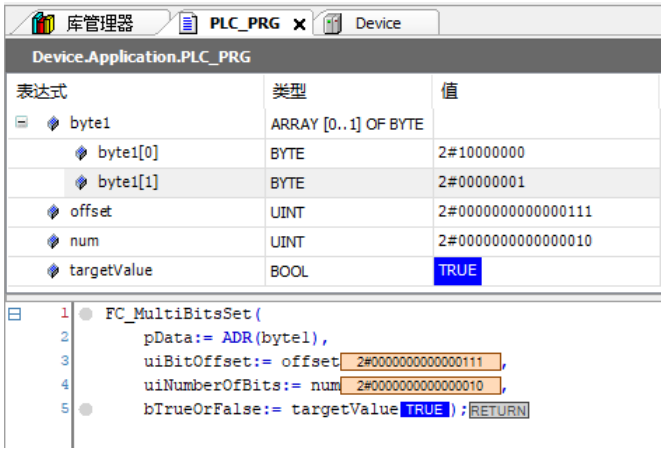
```
FC_MultiBitsSet(
    pData:= ADR(bytel),
    uiBitOffset:= offset,
    uiNumberOfBits:= num,
    bTrueOrFalse:= targetValue);
```

【2】举例对起始地址的 Bit7 开始对 2 个 BIT 写入 TRUE。



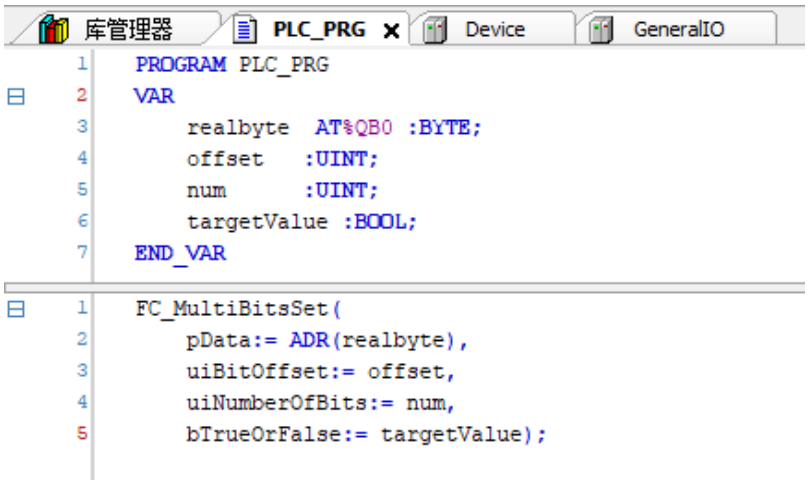
写入效果如图，为方便查看这里将数据显示模式切换为二进制，可以看到 Bit7 和 Bit8 已经被修改为 TRUE

（BYTE 中左边为高位）。

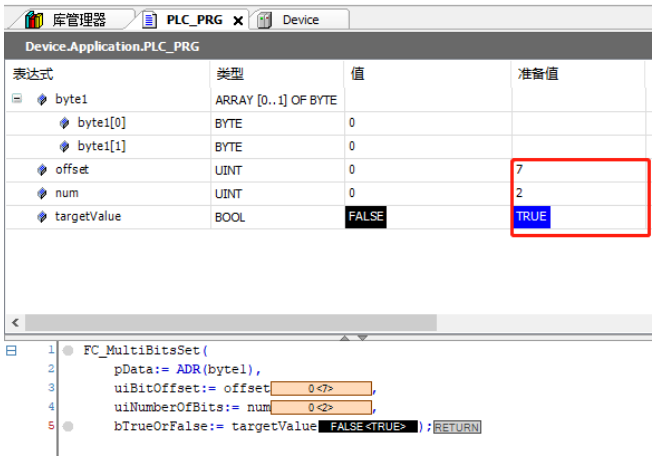


7.3.2. 使用举例 2（对物理地址进行修改）

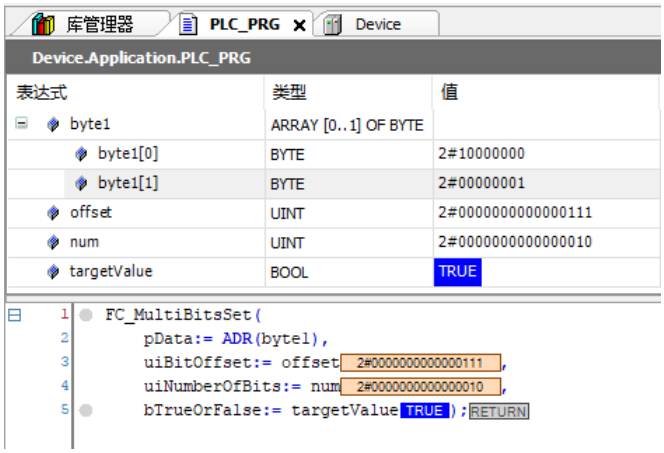
【1】定义一个 BYTE 数组（QB0 为禾川 Q1 控制器本体 IO 输出的起始地址）和相应的 UINT 变量、BOOL 变量，调用函数。



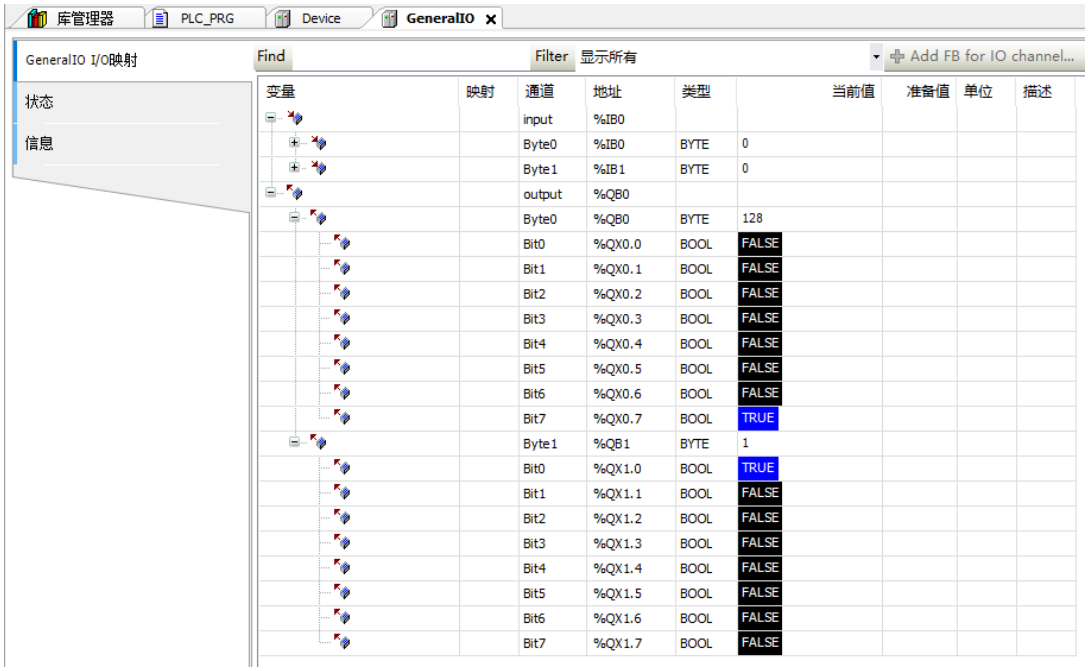
【2】举例对起始地址的 Bit7 开始对 2 个 BIT 写入 TRUE。



写入效果如图，为方便查看这里将数据显示模式切换为二进制，可以看到 Bit7 和 Bit8 已经被修改为 TRUE（BYTE 中左边为高位）。



在 GeneralIO 中，Q1 的输出端子 QX0.7 和 QX1.0 引脚被修改为 TRUE，即 Bit7 和 Bit8 被修改为 TRUE。

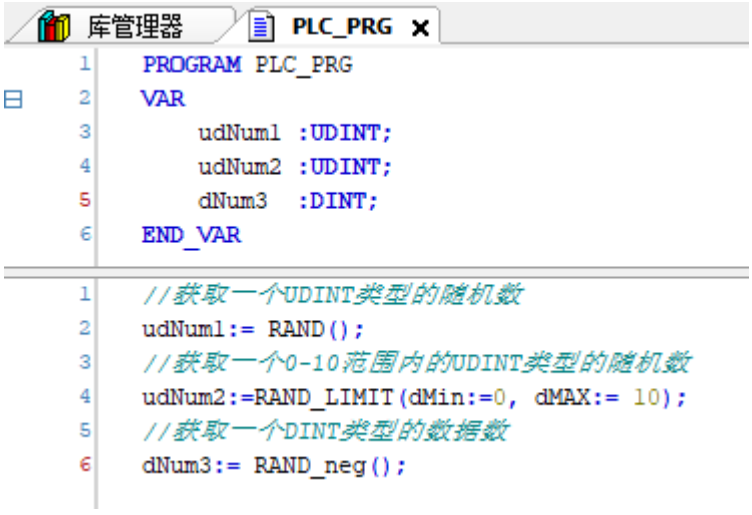


7.4. RAND （功能组）

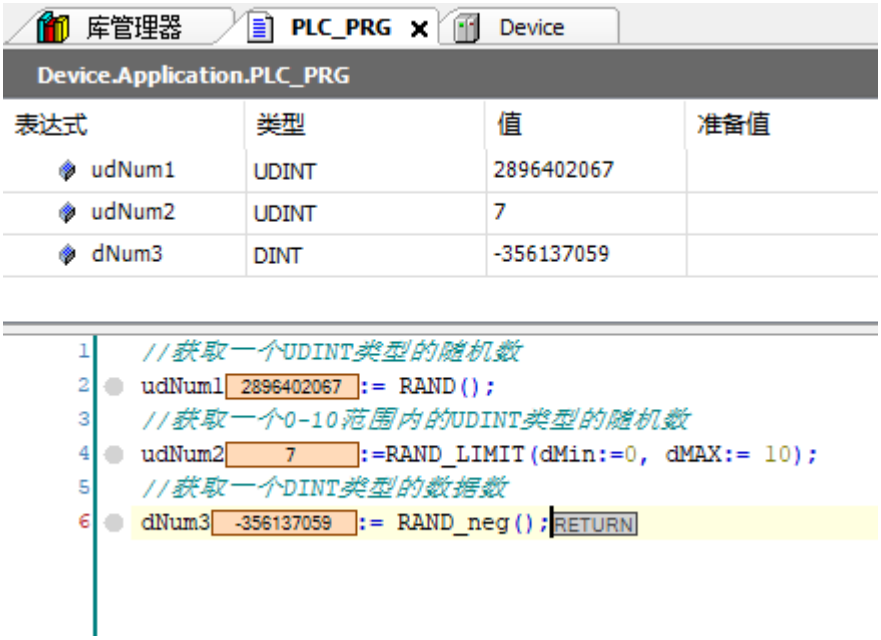
随机数生成功能，包含三种函数，具体参看每个函数介绍

名称	内容
RAND (FUN)	生成随机数
RAND_LIMIT (FUN)	生成带有正反极限的随机数
RAND_neg （FUN）	生成带符号的随机数

7.4.1. 使用举例



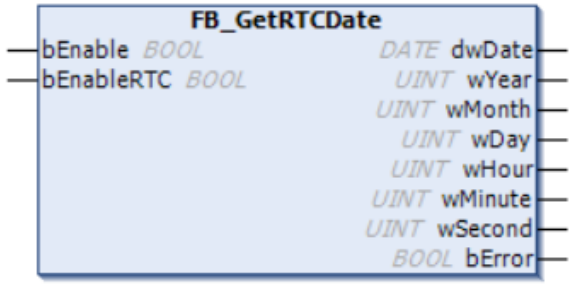
可以看到函数一直在输出对应类型的随机数。



7.5. RTCTime（功能组）

PLCrtc 时钟读取与修改功能组

7.5.1. 读取功能块 FB_GetRTCDate (FB)

名称	FB_GetRTCDate (FB)（获取 RTC 时间）		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	
		FB_GetRTCDate(bEnable:= , bEnableRTC:= , dwDate=> , wYear=> , wMonth=> , wDay=> , wHour=> , wMinute=> , wSecond=> , bError=>);	

输入变量

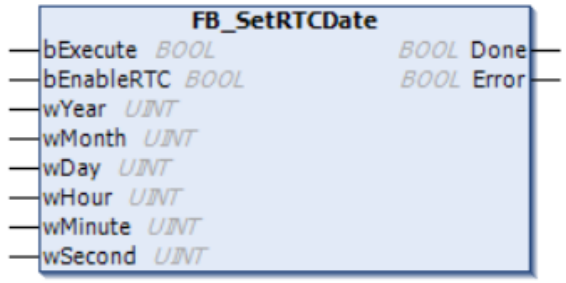
输入变量	名称	数据类型	有效范围	初始值	内容
bEnable	使能	BOOL	TRUE、FALSE	FALSE	TRUE：启用功能块 FALSE：关闭功能块
bEnableRTC	时区选择引脚	BOOL	TRUE、FALSE	TRUE	TRUE：读取 Local 时间 FALSE：读取 UTC 时间

输出变量

输出变量	名称	数据类型	有效范围	初始值	内容
dwDate	日期	DATE			日期
wYear	年	UINT		1970	年
wMonth	月	UINT		1	月
wDay	日	UINT		1	日
wHour	时	UINT			时
wMinute	分	UINT			分
wSecond	秒	UINT			秒
bError	错误	BOOL	TRUE、 FALSE		TRUE:功能块产生异常，已停止执行

7.5.2. 修改功能块 FB_SetRTCDate (FB)

名称	FB_SetRTCDate (FB)（获取 RTC 时间）		
支持的模式	CSP	CSV	CST
图形表现		ST 表现	




```
FB_SetRTCDate(
    bExecute:= ,
    bEnableRTC:= ,
    wYear:= ,
    wMonth:= ,
    wDay:= ,
    wHour:= ,
    wMinute:= ,
    wSecond:= ,
    Done=> ,
    Error=> );
```

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
Execute	使能	BOOL	TRUE、FALSE	FALSE	TRUE：启用功能块 FALSE：关闭功能块

bEnableRTC	时区选择引脚	BOOL	TRUE、FALSE	TRUE	TRUE：读取 Local 时间 FALSE：读取 UTC 时间
wYear	年	UINT		1970	年
wMonth	月	UINT		1	月
wDay	日	UINT		1	日
wHour	时	UINT			时
wMinute	分	UINT			分
wSecond	秒	UINT			秒

输出变量

输出变量	名称	数据类型	有效范围	内容
bDone	完成	BOOL	TRUE、FALSE	TRUE:功能块执行完成
bError	错误	BOOL	TRUE、FALSE	TRUE:功能块产生异常，已停止执行

7.5.3. 使用举例

【1】声明并调用功能块 FB_GetRTCDate、FB_SetRTCDate

```
VAR
    FB_GetRTCDate       :FB_GetRTCDate;
    FB_SetRTCDate   :FB_SetRTCDate;
END_VAR
```

连接 PLC 并连接程序，使能 FB_SetRTCDate 功能块，写入成功后，通过 FB_GetRTCDate 读取现在 PLC 中的 UTC 时间。

表达式	类型	值	准备值
FB_GetRTCDate	FB_GetRTCDate		
bEnable	BOOL	TRUE	
bEnableRTC	BOOL	TRUE	
dwDate	DATE	D#2024-3-17	
wYear	UINT	2024	
wMonth	UINT	3	
wDay	UINT	17	
wHour	UINT	21	
wMinute	UINT	16	
wSecond	UINT	21	
bError	BOOL	FALSE	

表达式	类型	值	准备值
FB_GetRTCDate	FB_GetRTCDate		
bEnable	BOOL	TRUE	
bEnableRTC	BOOL	FALSE	
dwDate	DATE	D#2024-3-17	
wYear	UINT	2024	
wMonth	UINT	3	
wDay	UINT	17	
wHour	UINT	13	
wMinute	UINT	17	
wSecond	UINT	15	
bError	BOOL	FALSE	

使能 FB_SetRTCDate 功能块，写入成功后，通过 FB_GetRTCDate 读取现在 PLC 中的 rtc 时间。

表达式	类型	值	准备值
FB_SetRTCDate	FB_SetRTCDate		
bExecute	BOOL	TRUE	
bEnableRTC	BOOL	TRUE	
wYear	UINT	2024	
wMonth	UINT	3	
wDay	UINT	17	
wHour	UINT	21	
wMinute	UINT	21	
wSecond	UINT	0	
Done	BOOL	TRUE	
Error	BOOL	FALSE	
FB_GetRTCDate	FB_GetRTCDate		
bEnable	BOOL	TRUE	
bEnableRTC	BOOL	TRUE	
dwDate	DATE	D#2024-3-17	
wYear	UINT	2024	
wMonth	UINT	3	
wDay	UINT	17	
wHour	UINT	21	
wMinute	UINT	21	
wSecond	UINT	0	
bError	BOOL	FALSE	

表达式	类型	值	准备值
FB_SetRTCDate	FB_SetRTCDate		
bExecute	BOOL	TRUE	
bEnableRTC	BOOL	FALSE	
wYear	UINT	2024	
wMonth	UINT	3	
wDay	UINT	17	
wHour	UINT	12	
wMinute	UINT	21	
wSecond	UINT	0	
Done	BOOL	TRUE	
Error	BOOL	FALSE	
FB_GetRTCDate	FB_GetRTCDate		
bEnable	BOOL	TRUE	
bEnableRTC	BOOL	TRUE	
dwDate	DATE	D#2024-3-17	
wYear	UINT	2024	
wMonth	UINT	3	
wDay	UINT	17	
wHour	UINT	20	
wMinute	UINT	21	
wSecond	UINT	0	
bError	BOOL	FALSE	

要点说明

- 不建议把 FB_SetRTCDate、FB_GetRTCDate 功能块放在 EtherCatTask 等运动控制任务中使用，在设置时间时会产生阻塞，从而导致报错。

7.6. 滤波指令 (功能组)

7.6.1. ArithmeticAverageFilter (算术平均滤波)

名称	FB_ArithmeticAverageFilter (算术平均滤波)	
图形表现		ST 表现
		FB_ArithmeticAverageFilter(bEnable:= , fSample:= , uiSampleNum:= , uiSampleCycle:= , bBusy=> , bValid=> , bError=> , eError=> , fValue=>);

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bEnable	功能块使能	BOOL	TRUE、FALSE	FALSE	True 功能块执行， false 不执行功能块
fSample	输入采样值	REAL		0	输入采样值
uiSampleNum	输入采样值	UINT	1-1000		输入采样值
uiSampleCycle	输入采样周期	UINT	1-1000	0	输入采样周期

输出变量

输出变量	名称	数据类型	有效范围	内容
bBusy	指令正在执行	BOOL	TRUE、FALSE	TRUE: 功能块运行
bValid	输出有效	BOOL	TRUE、FALSE	True，指令执行有效
bError	错误	BOOL	TRUE、FALSE	True，异常产生
eErrorID	错误 ID	WSTRING		
fValue	滤波输出有效值	REAL		滤波输出有效值

要点说明

- 当 bEnable 电平使能后，给定扫描周期个数 uiSampleCycle，给定采样次数 uiSampleNum，输入采样变量 fValue。连续采样 uiSampleNum 次，进行算术平均运算。
- uiSampleNum 值越大时：信号平滑度较高，但灵敏度较低。
- uiSampleNum 值越小时：信号平滑度较低，但灵敏度较高。
- uiSampleNum 取值的选取：一般流量，uiSampleNum=12；压力：uiSampleNum=4。
- 滤波功能块中输入参数 uiSampleCycle 为采样周期数，若 uiSampleCycle=1 时表示每个周期均进行采样操作；若 uiSampleCycle=3 表示每 3 个周期进行采样一次。

- 采样数据为存放于数组的数据，需要用户自行处理数据，将数组中的数据逐个取出来

7.6.2. DebounceFilter (消抖滤波)

名称	FB_DebounceFilter(消抖滤波)
图形表现	ST 表现


```
FB_DebounceFilter(
    bEnable:= ,
    fSample:= ,
    uiUpLimit:= ,
    fReference:= ,
    uiSampleCycle:= ,
    bBusy=> ,
    bValid=> ,
    bError=> ,
    ErrorID=> ,
    fVaule=> );
```

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bEnable	功能块使能	BOOL	TRUE、FALSE	FALSE	True 功能块执行，false 不执行功能块
fSample	输入采样值	REAL			输入待滤波值
uiUpLimit	设置滤波计数器上限	REAL	1-65535		设置滤波计数器上限
fReference	输入参考值	REAL			输入参考值
uiSampleCycle	输入参考值	UINT	1-1000		输入采样周期

输出变量

输出变量	名称	数据类型	有效范围	内容
bBusy	指令正在执行	BOOL	TRUE、FALSE	TRUE，功能块执行中
bValid	输出有效	BOOL	TRUE、FALSE	True，指令执行有效
bError	错误	BOOL	TRUE、FALSE	True，异常产生
eErrorID	错误 ID	WSTRING		
fValue	滤波输出有效值	LREAL		滤波输出有效值

要点说明

- 当 bEnable 电平使能后，给定扫描周期个数 uiSampleCycle、采样参考值 fReference、设置一个滤波上限值 uiUpLimit 后，输入采样变量 fSample。将每次采样值与当前有效值比较：
- 采样值 = 当前有效值，则当前计数值清零
- 采样值 <> 当前有效值，则计数值 +1，并判断计数值是否溢出（计数值 > 上限 uiUpLimit）
- 计数值溢出，则将本次采样值替换当前滤波值输出，并清零计数值

- 对于变化缓慢的被测参数有较好的滤波效果，变化较快的参数效果不明显

7.6.3. FirstOrderLagFilter (一阶滞后滤波)

名称	FB_FirstOrderLagFilter (一阶滞后滤波)
图形表现	ST 表现


```
FB_FirstOrderLagFilter(
    bEnable:= ,
    fSample:= ,
    fCoefficient:= ,
    fReference:= ,
    uiSampleCycle:= ,
    bBusy=> ,
    bValid=> ,
    bError=> ,
    ErrorID=> ,
    fVaule=> );
```

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bEnable	功能块使能	BOOL	TRUE、FALSE	FALSE	True 功能块执行，false 不执行功能块
fSample	输入采样值	REAL		0	输入待滤波值
fCoefficient	输入一阶低通滤波系数 a=0 ~1	REAL	0-1		输入一阶低通滤波系数
fReference	输入有效值	REAL		0	输入有效值
uiSampleCycle	输入采样周期	UINT	1-1000	0	输入采样周期

输出变量

输出变量	名称	数据类型	有效范围	内容
bBusy	指令正在执行	BOOL	TRUE、FALSE	TRUE，功能块执行中
bValid	输出有效	BOOL	TRUE、FALSE	True，指令执行有效
bError	错误	BOOL	TRUE、FALSE	True，异常产生
eErrorID	错误 ID	WSTRING		
fValue	滤波输出有效值	LREAL		滤波输出有效值

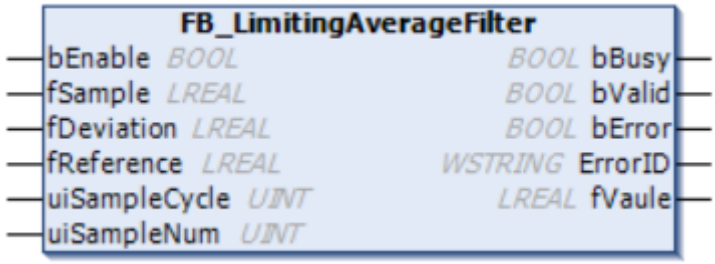
要点说明

- 当 bEnable 电平使能后，给定扫描周期个数 uiSampleCycle，采样参考值 fReference，输入一阶低通滤波系数 fCoefficient（取 0~1），输入采样变量 fSample。
- 计算公式: 本次滤波结果 =fCoefficient* 本次采样值 +（1-fCoefficient）* 上次滤波结果

- 使用波动频率较高、周期性干扰的场合，但是相位滞后、灵敏度低，不能消除滤波频率高于采样频率 1/2 的干扰信号。

7. 6. 4. LimitingAverageFilter (限幅平均滤波)

名称	FB_LimitingAverageFilter (限幅平均滤波)
图形表现	ST 表现




```
FB_LimitingAverageFilter(
    bEnable:= ,
    fSample:= ,
    fDeviation:= ,
    fReference:= ,
    uiSampleCycle:= ,
    uiSampleNum:= ,
    bBusy=> ,
    bValid=> ,
    bError=> ,
    ErrorID=> ,
    fVaule=> );
```

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bEnable	功能块使能	BOOL	TRUE、FALSE	FALSE	True 功能块执行， false 不执行功能块
fSample	输入采样值	REAL		0	输入待滤波值
fDeviation	输入两次采样允许的最大偏差值	REAL		0	输入两次采样允许的最大偏差值
fReference	输入有效值	REAL		0	输入有效值
uiSampleCycle	输入采样周期	UINT	1-1000	0	输入采样周期
uiSampleNum	输入采样次数	UINT	1-1000	0	输入采样次数

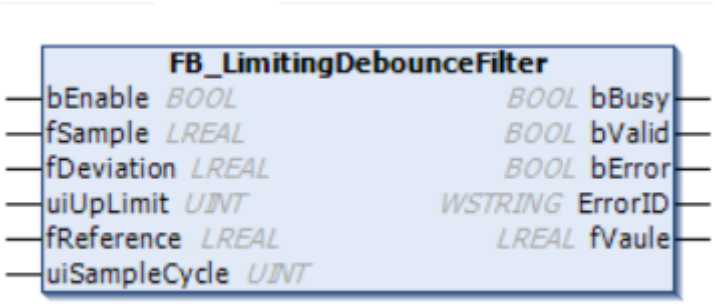
输出变量

输出变量	名称	数据类型	有效范围	内容
bBusy	指令正在执行	BOOL	TRUE、FALSE	TRUE，功能块执行中
bValid	输出有效	BOOL	TRUE、FALSE	True，指令执行有效
bError	错误	BOOL	TRUE、FALSE	True，异常产生
eErrorID	错误 ID	WSTRING		
fValue	滤波输出有效值	LREAL		滤波输出有效值

要点说明

- bEnable 电平使能后，给定扫描周期个数 uiSampleCycle，给定采样个数 uiSampleNum、采样参考有效值 fReference、两次采样允许的最大偏差值 fDeviation 后，输入采样变量 fSample。
- 每次采样到的新数据先进行限幅处理，再送入队列进行递推平均滤波处理，相当于结合了限幅滤波法和递推平均滤波法，可以消除由偶发干扰带来的采样值偏差。

7. 6. 5. LimitingDebounceFilter (限幅消抖滤波)

名称	FB_LimitingDebounceFilter (限幅消抖滤波)	
图形表现		ST 表现
		FB_LimitingDebounceFilter(bEnable:= , fSample:= , fDeviation:= , uiUpLimit:= , fReference:= , uiSampleCycle:= , bBusy=> , bValid=> , bError=> , ErrorID=> , fVaule=>);

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bEnable	功能块使能	BOOL	TRUE、FALSE	FALSE	True 功能块执行， false 不执行功能块
fSample	输入采样值	REAL		0	输入待滤波值
fDeviation	输入两次采样允许的最大偏差值	REAL		0	输入两次采样允许的最大偏差值
uiUpLimit	设置滤波计数器上限	UINT	1-65535	0	设置滤波计数器上限
fReference	输入参考值	REAL		0	输入参考值
uiSampleCycle	输入采样周期	UINT	1-1000	0	输入采样周期

输出变量

输出变量	名称	数据类型	有效范围	内容
bBusy	指令正在执行	BOOL	TRUE、FALSE	TRUE，功能块执行中
bValid	输出有效	BOOL	TRUE、FALSE	True，指令执行有效
bError	错误	BOOL	TRUE、FALSE	True，异常产生
eErrorID	错误 ID	WSTRING		
fValue	滤波输出有效值	LREAL		滤波输出有效值

要点说明

- bEnable 电平使能后，给定扫描周期个数 uiSampleCycle，设置滤波计数器上限 uiUpLimit，采样参考值 fReference、两次采样允许的最大偏差值 fDeviation 后，输入采样变量 fSample。
- 本次值与上次值之差 <= fDeviation，则本次值有效作为消抖滤波的采样值
- 本次值与上次值之差 > fDeviation，则本次值无效，放弃本次值，用上次值代替本次值作为消抖滤波的采样值
- 先经历过限幅后再作为消抖滤波的采样值与当前有效值比较
- 采样值 = 当前有效值，则当前计数值清零
- 采样值 <> 当前有效值，则计数值 +1，并判断计数值是否溢出(计数值 > 上限 uiUpLimit），如果计数值溢出，则将本次采样值替换当前滤波值输出，并清零计数值。

7.6.6. LimitingFilter (限幅滤波)

名称	FB_LimitingFilter (限幅滤波)
图形表现	ST 表现


```
FB_LimitingFilter(
        bEnable:= ,
        fSample:= ,
        fDeviation:= ,
        fReference:= ,
        uiSampleCycle
:= ,
        bBusy=> ,
        bValid=> ,
        bError=> ,
        eError=> ,
        fValue=> );
```

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bEnable	功能块使能	BOOL	TRUE、FALSE	FALSE	True 功能块执行，
fSample	输入采样值	REAL		0	输入采样值
fDeviation	输入两次采样允许的最大偏差值	REAL		0	输入两次采样允许的最大偏差值
fReference	输入参考值	REAL		0	输入采样参考值
uiSampleCycle	输入采样周期	UINT	1-1000	0	输入采样周期

输出变量

输入变量	名称	数据类型	有效范围	内容
bBusy	指令正在执行	BOOL	TRUE、FALSE	TRUE，功能块执行中
bValid	输出有效	BOOL	TRUE、FALSE	True，指令执行有效
bError	错误	BOOL	TRUE、FALSE	True，异常产生
eErrorID	错误 ID	WSTRING		
fValue	滤波输出有效值	REAL		滤波输出有效值

要点说明

- bEnable 电平使能后，给定扫描周期个数 uiSampleCycle，采样参考值 fReference 与两次采样允许的最大偏差值 fDeviation 后，输入采样值 fSample。每次采样到新值时判断：
- 本次值与上次值之差 <= fDeviation，则本次值有效作为滤波值输出
- 本次值与上次值之差 > fDeviation，则本次值无效，放弃本次值，用上次值代替本次值作为滤波值输出
- 可以克服偶发引起的干扰，对周期性的干扰效果较差

7.6.7. MedianAverageFilter (中位值平均滤波)

名称	FB_MedianAverageFilter (中位值平均滤波)
图形表现	ST 表现


```
FB_MedianAverageFil
ter(
      bEnable:= ,
      fSample:= ,
      uiSampleNum:
= ,
      uiSampleCycle:
= ,
      bBusy=> ,
      bValid=> ,
      bError=> ,
      eError=> ,
      fValue=> );
```

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bEnable	功能块使能	BOOL	TRUE、FALSE	FALSE	True 功能块执行，
fSample	输入采样值	REAL		0	输入采样值
uiSampleNum	输入采样个数	UINT	3-3000	0	输入采样个数
uiSampleCycle	输入采样周期	UINT	1-1000	0	输入采样周期

输出变量

输出变量	名称	数据类型	有效范围	内容
bBusy	指令正在执行	BOOL	TRUE、FALSE	TRUE，功能块执行中
bValid	输出有效	BOOL	TRUE、FALSE	True，指令执行有效
bError	错误	BOOL	TRUE、FALSE	True，异常产生
eErrorID	错误 ID	WSTRING		
fValue	滤波输出有效值	REAL		滤波输出有效值

要点说明

- bEnable 电平使能后，给定扫描周期个数 uiSampleCycle，给定采样个数 uiSampleNum，输入采样变量 fSample。连续采样 uiSampleNum 次，将该组数据进行排序，去掉最大值和最小值后取平均值，相当于“中位值滤波法’ 和“算术平均值滤波法”。 uiSampleNum 值通常选取 3~14。
- 可以克服偶发引起的干扰，对于周期干扰也有一定克服作用，适用于高频振荡的系统。但是计算速度较慢

7. 6. 8. MedianFilter (中位值滤波)

名称	FB_MedianFilter (中位值滤波)
图形表现	ST 表现


```
FB_MedianFilter(
    bEnable:= ,
    fSample:= ,
    uiSampleNum:= ,
    uiSampleCycle:= ,
    bBusy=> ,
    bValid=> ,
    bError=> ,
    eError=> ,
    fValue=> );
```

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bEnable	功能块使能	BOOL	TRUE、FALSE	FALSE	True 功能块执行，
fSample	输入采样值	REAL		0	输入采样值
uiSampleNum	输入采样个数	UINT	1-1000	0	输入采样个数
uiSampleCycle	输入采样周期	UINT	1-1000	0	输入采样周期

输出变量

输出变量	名称	数据类型	有效范围	内容
bBusy	指令正在执行	BOOL	TRUE、FALSE	TRUE，功能块执行中
bValid	输出有效	BOOL	TRUE、FALSE	True，指令执行有效
bError	错误	BOOL	TRUE、FALSE	True，异常产生
eErrorID	错误 ID	WSTRING		
fValue	滤波输出有效值	REAL		滤波输出有效值

要点说明

- bEnable 电平使能后，给定扫描周期个数 uiSampleCycle，给定采样个数 uiSampleNum，输入采样变量 fValue。连续采样 uiSampleNum 次，把 uiSampleNum 次采样值按大小排列，取中间值为本次滤波值。
- 可以克服偶发引起的干扰，但对于快速变化的参数效果不好。

7.6.9. RecursiveAverageFilter (递推平均滤波)

名称	FB_RecursiveAverageFilter (递推平均滤波)
图形表现	ST 表现


```
FB_RecursiveAverageFi
lter(
        bEnable:= ,
        fSample:= ,
        uiSampleNum:= ,
        uiSampleCycle:=
,
        bBusy=> ,
        bValid=> ,
        bError=> ,
        eError=> ,
        fValue=> );
```

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bEnable	功能块使能	BOOL	TRUE、FALSE	FALSE	True 功能块执行，
fSample	输入采样值	REAL		0	输入采样值
uiSampleNum	输入队列长度	UINT	1-3000	0	输入队列长度
uiSampleCycle	输入采样周期	UINT	1-1000	0	输入采样周期

输出变量

输出变量	名称	数据类型	有效范围	内容
bBusy	指令正在执行	BOOL	TRUE、FALSE	TRUE，功能块执行中
bValid	输出有效	BOOL	TRUE、FALSE	True，指令执行有效
bError	错误	BOOL	TRUE、FALSE	True，异常产生
eErrorID	错误 ID	WSTRING		
fValue	滤波输出有效值	REAL		滤波输出有效值

要点说明

- bEnable 电平使能后，给定扫描周期个数 uiSampleCycle，给定队列长度 uiQueueCount，输入采样变量 fSample。把连续 uiQueueCount 个采样看成一个队列，队列的长度固定为 uiQueueCount。每次把采样到一个新数据放入队尾，并扔掉原来队首的一个数据（先进先出原则），再把队列中的 N 个数据进行算术平均运算，获得新的滤波结果。
- uiQueueCount 值越大时：信号平滑度较高，但灵敏度较低
- uiQueueCount 值越小时：信号平滑度较低，但灵敏度较高
- uiQueueCount 取值的一般选取：流量:12；压力:4；温度:4；
- 对于高频振荡，有周期干扰的系统有比较好的效果，但灵敏度低，对于偶发干扰克服能力较差

7.6.10. WeightRecursiveAverageFilter (加权递推平均滤波)

名称	FB_WeightRecursiveAverageFilter (加权递推平均滤波)
图形表现	ST 表现


```
FB_WeightRecursiveAverageF
ilter(
    Enable:= ,
    fSample:= ,
    pWeighted:= ,
    uiQueueCount:= ,
    uiSampleCycle:= ,
    Busy=> ,
    Valid=> ,
    Error=> ,
    ErrorID=> ,
    fVaule=> );
```

输入变量

输入变量	名称	数据类型	有效范围	初始值	内容
bEnable	功能块使能	BOOL	TRUE、FALSE	FALSE	True 功能块执行，
fSample	输入采样值	REAL		0	输入待滤波值
pfWeighted	输入加权系数，存放于数组中	POINTER TO REAL		0	输入加权系数，存放于数组中，不能都等于 0
uiQueueCount	输入队列长度	UINT	1-3000	0	输入有效值
uiSampleCycle	输入采样周期	UINT	1-1000		输入采样周期

输出变量

输出变量	名称	数据类型	有效范围	内容
bBusy	指令正在执行	BOOL	TRUE、FALSE	TRUE，功能块执行中
bValid	输出有效	BOOL	TRUE、FALSE	True，指令执行有效
bError	错误	BOOL	TRUE、FALSE	True，异常产生
eErrorID	错误 ID	WSTRING		
fValue	滤波输出有效值	REAL		滤波输出有效值

要点说明

- bEnable 电平使能后，给定扫描周期个数 uiSampleCycle，给定队列长度 uiQueueCount，输入采样变量 fSample 和相应的加权系数数组的地址 pfWeighted。
- 不同时刻的数据加以不同的权。通常，越接近现时刻的数据，权取越大给予新采样值的权系数越大，则灵敏度越高，但信号平滑度越低。
- 对于采样周期较短，或有较大滞后周期时间常数的对象

7.7. PID 自整定功能块

注： 该功能块详细说明篇幅较长，有独立的完善说明，请阅读参靠说明书：

ATC 温控 PID 使用说明

HCFA

创新 诚信 服务

HCFA

HCFA_ATC