SIMATIC
S7-200 应用示例

目　　录

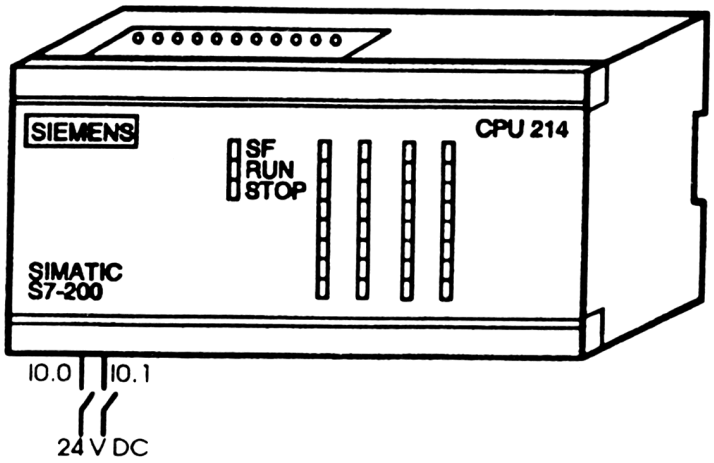
1 处理定时中断

概述

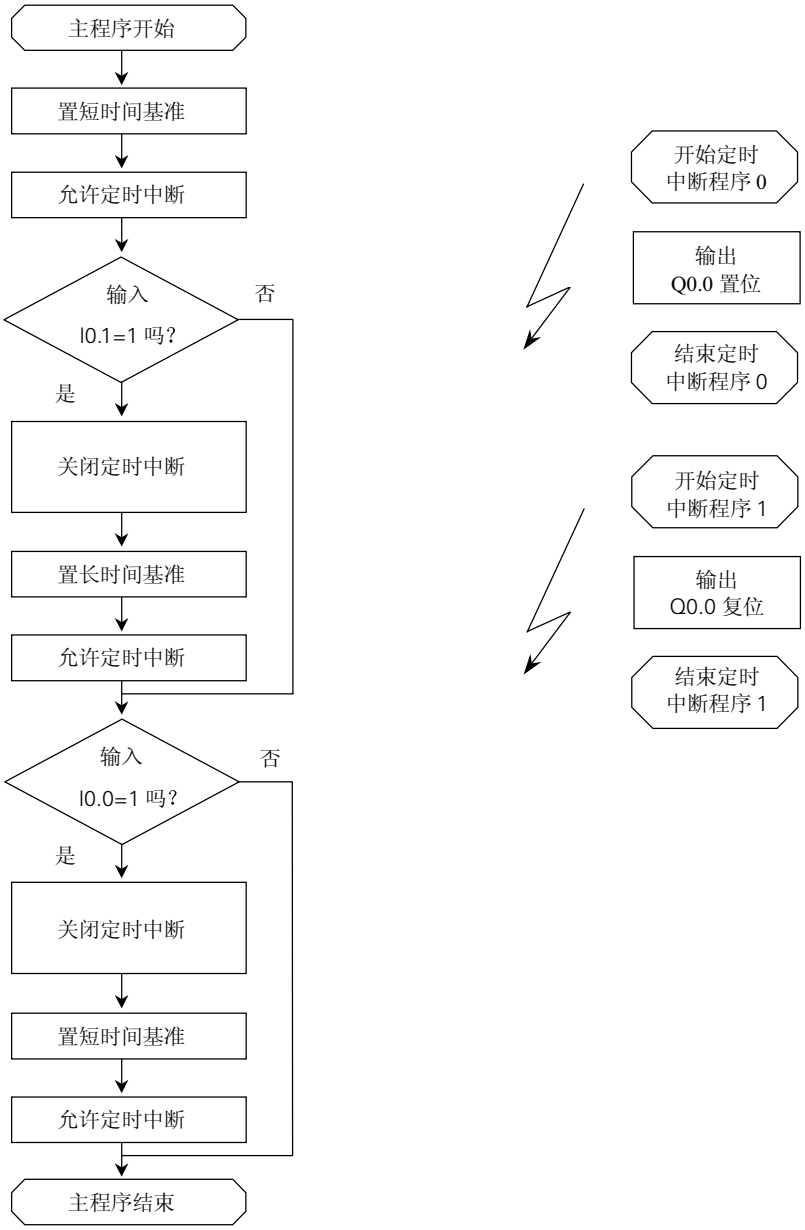
本例用定时中断来产生闪烁频率脉冲。当连在输入端 10.1 的开关接通时，闪烁频率减半；当连在输入端 10.1 的开关接通时，又恢复成原有的闪烁频率。

本例叙述由定时中断引起的一般性的处理以及改变其时间基准。

例图



程序框图


主程序开始
置短时间基准
允许定时中断
输入
I0.1=1 吗?
否
是
关闭定时中断
置长时间基准
允许定时中断
输入
I0.0=1 吗?
否
是
关闭定时中断
置短时间基准
允许定时中断
主程序结束
开始定时
中断程序 0
输出
Q0.0 置位
结束定时
中断程序 0
开始定时
中断程序 1
输出
Q0.0 复位
结束定时
中断程序 1

程序和注释

用特殊存储字节 SMB34 指定第一定时中断的时间基准，由此产生的定时中断称为中断事件 10。

用特殊存储字节 SMB35 指定第二定时中断的时间基准，由此产生的定时中断称为中断事件 11。仅 CPU214 支持第二定时中断。

这两种定时中断的时间基准的设定值只能以 1ms（毫秒）为单位增加，允许最小值是 5ms，最大值是 255ms。本例程序组成如下：

Main	主程序	初始化和指定时间基准
INT0	中断程序 1	对输出 Q0.0 置位（Q0.0=1）
INT1	中断程序 2	对输出 Q0.0 复位（Q0.0=0）

本程序长度为 51 个字。

```
标题：定时中断

//＊＊＊＊＊＊＊＊＊＊＊＊ 主 程 序 ＊＊＊＊＊＊＊＊＊＊

// 在主程序的第一部分指定起始时间基准。
// 为两个定时中断分别指定对应的中断处理程序。

LD      SM0.1           // 仅首次扫描处理。
MOVB    50，SMB34       // 设置定时中断 0 的时间基准为 50ms。
MOVB    100，SMB35      // 设置定时中断 1 的时间基准为 100ms。
ATCH    0，10           // 指定定时中断事件 10 调用中断程序 0。
ATCH    1，11           // 指定定时中断事件 11 调用中断程序 1。
ENI                     // 允许中断。
当输入 10.1 有上升沿（从 0 到 1）时，定时中断的时间基准加倍。
// 为了执行这一新的指令，必须断开中断事件与中断程序之间的联系，否则不承认新的时间基准。
// 用 DTCH 指令来切断两者之间的联系。
// 用指定了新的时间基准后，必须用 ATCH 指令来恢复中断事件与中断程序之间的联系。

LD      10，1           // 输入 10.1。
EU                      // 上升沿。
DTCH    10              // 切断定时中断事件 10 与中断程序 0 的联系。
DTCH    11              // 切断定时中断事件 11 与中断程序 1 的联系。
MOVB    100，SMB34      // 设置定时中断 0 的新的时间基准为 100ms。
MOVB    200，SMB35      // 设置定时中断 1 的新的时间基准为 200ms。
ATCH    0，10           // 恢复定时中断事件 10 调用中断程序 0。
ATCH    1，11           // 恢复定时中断事件 11 调用中断程序 1。

当输入    10.0 有上升沿时，恢复使用原频率。
```

```
LD      10.0                    // 输入 10.0
EU                              // 上升沿
DTCH    10                      // 切断定时中断事件 10 与中断程序 0 的联系。
DTCH    11                      // 切断定是中断事件 11 与中断程序 1 的联系。
MOVB    50，SMB34               // 设置定时中断 0 的时间基准为 50ms
MOVB    100，SMB35              // 设置定时中断 1 的时间基准为 100ms
ATCH    0，10                   // 恢复定时中断事件 10 与中断程序 0 的联系
ATCH    1，11                   // 恢复定时中断事件 11 与中断程序 1 的联系

MEND                            // 主程序结束。

//  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *

// 中断程序 0。
// 当调用中断程序 0 时，把输出 Q0.0 置位（Q0.0=1）。

INT     0                       // 中断程序 0
LD      SM0.0                   // 特殊存储器位 SM0.0 总是 1
S       Q0.0，1                 // 把输出 Q0.0 置位（Q0.0=1）
RNTI                            // 中断程序 0 结束

//  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *

// 中断程序 1
// 当调用中断程序 1 时，把输出 Q0.0 复位（Q0.0=0）。
// 因为调用中断程序 1 的时间基准是调用中断程序 0 的两倍。
// 所以输出端 Q0.0 输出的脉冲频率发生闪烁。

INT     1                       // 中断程序 1
LD      SM0.0                   // SM0.0 总是 1
R       Q0.0，1                 // 把输出 Q0.0 复位（Q0.0=0）
RETI                            // 中断程序 1 结束
```

请参考 SIMATIC STEP 7 编程参考手册 6.2 节“中断指令”，为您提供了更多的关于定时中断的信息。

 西门子 版权所有

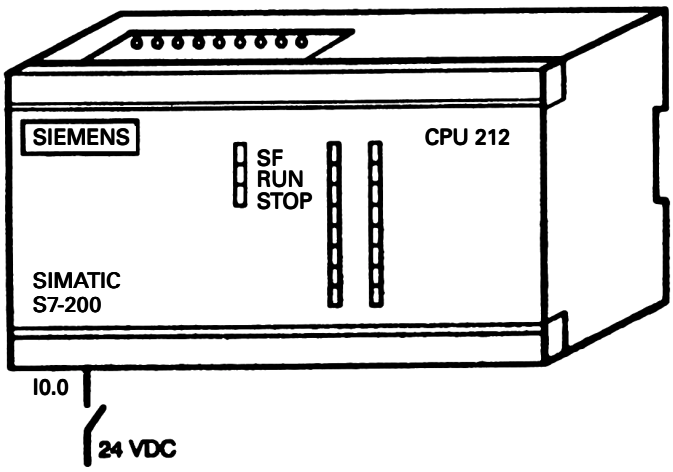
2 处理输入/输出中断

概述

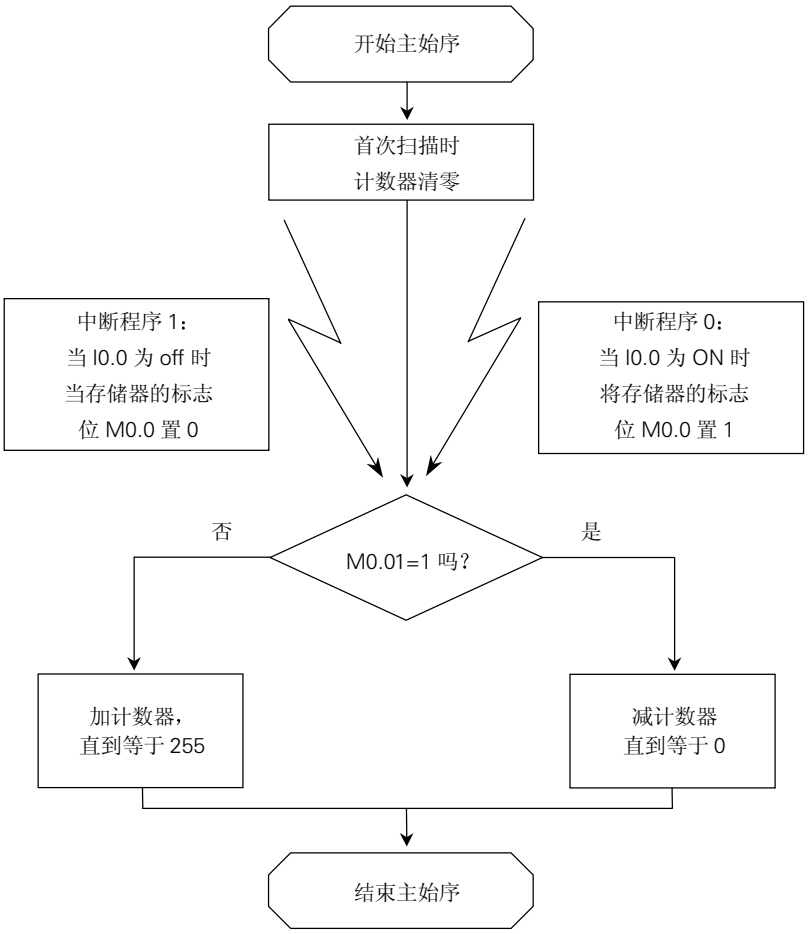
本程序适用于 SIMATIC S7-212 和 S7-214 的计数器，可以从 0 计到 255，这要取决于输入 10.0 的状态。如果将输入 10.0 置为 1，则程序减计数；如果将输入 10.0 置为 0，则程序加计数。

如果输入 10.0 的状态改变，则将立即激活输入/输出中断程序，中断程序 0 或 1 分别将存储器位 M0.0 置成 1 或 0。

例图



程序框图



程序和注解

本程序是一个输入/输出中断程序的范例，计数器从 0 计到 255。如果输入 10.0 为 0，则程序加计数；如果输入 10.0 为 1，则程序减计数。

本程序包括以下三个程序：

Main	（主程序）	初始化和计数
INT0	（中断程序 0）	输入 10.0 为 1 时，减计数。
INT1	（中断程序 1）	输入 10.0 为 0 时，加计数。

本程序长度为 32 个字。

 西门子 版权所有

```
// 标题：事件中断

// * * * * * * * * * 主程序 * * * * * * * * * *

// 主程序包括初始化程序和计数程序。
// 计数器的存储器标志位 M0.0 的 0 或 1 状态，决定计数方向为加或减计数。
// 当输入 10.0 由 0 变为 1 时，产生中断事件 0，激活中断程序 0（INT0）。
// 中断程序 0 将存储器位 M0.0 置成 1，导致主程序减计数。
// 当输入 10.0 由 1 变为 0 时，产生中断事件 1，激活中断程序 1（INT1）。
// 中断程序 1 将存储器位 M0.0 置成 0，导致主程序加计数。
// 主程序

LD      SM0.1           // 仅首次扫描时，SM0.1 才为 1，进行以下初始化
MOVB    +0，AC0         // 将计数累加器 AC0 清 0。
ENI                     // 允许中断。
ATCH    +0，0           // 输入 10.0 为上升沿时激活事件中断 0
ATCH    +1，1           // 输入 10.0 为上升沿时激活事件中断 1
LDN     M0.0            // 如果存储器的标志位 M0.0 为 0 状态
AB>=    16#FE，AC0      // 且计数累加器 AC0 的当前计数值小于或等于 254
A       SM0.5           // 且 0.5 秒脉冲
EU                      // 且上升沿
INCW    AC0             // 那么计算累加器 AC0 加 1

LD      M0.0            // 如果存储器的标志位 M0.0 为 1 状态
AB<=    16#1，AC0       // 且计数累加器 AC0 的当前计数值大于或等于 1
A       SM0.5           // 且 0.5 秒脉冲
EU                      // 且上升沿
DECW    AC0             // 那么计算器累加器 ACO 减 1

LD      SM0.0           // SM0.0 总是 1。
MOVB    AC0，QB0        // 在输出端 Q0.0 至 Q0.7 显示 AC0 的当前计数值。
MEND                    // 主程序结束。

// * * * * * * 中断程序 0 * * * * * *

// 事件中断程序 0 将存储器的标志位 M0.0 置成 1。
// 此情况下程序减计数。

INT     0               // 中断事件 0 减计数。
S       M0.0，1         // 将存储器的标志位 M0.0 置成 1。
RETI                    // 中断程序 0 结束。
```

```
// * * * * * * * 中断程序 1 * * * * * * *
// 事件中断程序 1 将存储器的标志位 M0.0 置成 0。
// 此情况下程序增计数。

INT     1                   // 中断事件 1 加计数。
R       M0.0，1             // 将存储器的标志位 M0.0 置成 0。
RETI                        // 中断程序 1 结束。
```

请参考 SIMATIC STEP 7 编程参考手册的 6.2 节“中断指令”，为您提供了更多的有关输入输出中断的信息。

 西门子 版权所有

3 模拟电位器

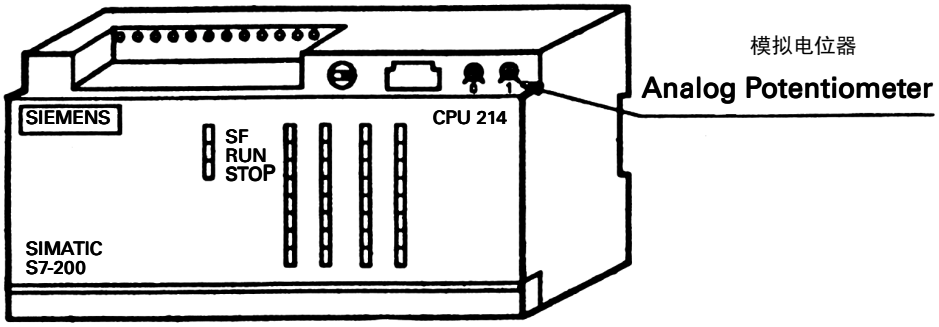
概述

本例包含了有关 SIMATIC S7-214 的模拟电位器（POT）的使用信息。电位器的位置转换为 0 至 255 之间的数字值，然后，存入两个特殊存储器字节 SMB28 和 SMB29 中，分别对应电位器 0 和电位器 1 的值。

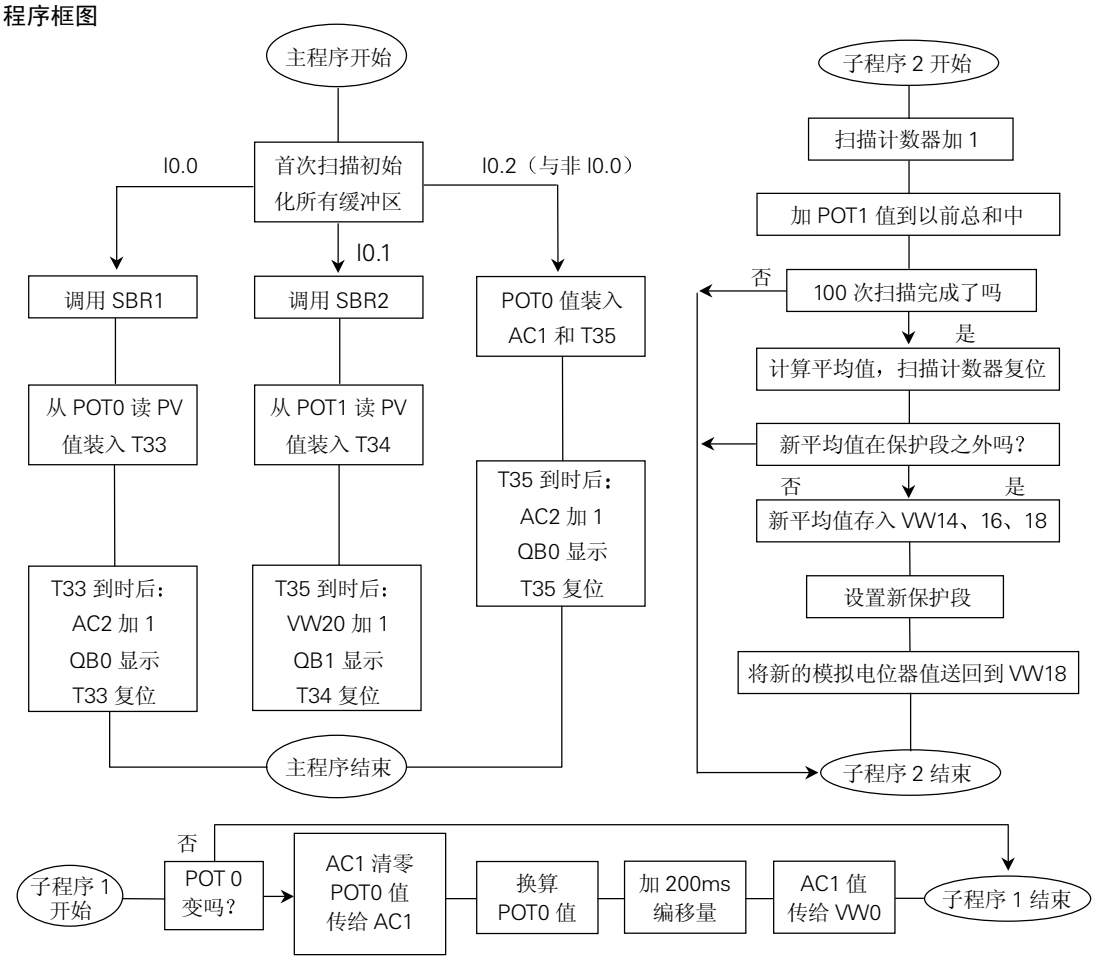
需要一把小螺丝刀用以调整电位器的位置。

本应用示例介绍了使用模拟电位器调整定时器设定值的三种方案。

例图



程序框图



程序和注释

方案 1 说明了用模拟电位器对定时器设定值进行细调的方法。首先通过程序中的偏移量（本例中为 200ms）对定时器进行粗调，然后再用电位器能把定时器的设定值精确地调整到满意的设置。每个定时器周期之后，执行子程序 1 中的指令，把 POT0 的值（在 SMB28 中）读到 AC1，除以 2，再加上 200ms 偏移量。返回主程序时，AC2 中的定时器循环计数值加 1，并拷贝到输出字节（QB0），以供显示。

在方案 2 中，对电位器 1（POT 1）的 100 次扫描值在 AC3 中累加后并取平均，再存入 VW12。如果该值低于低保护限值 VW14，或高于高保护限值 VW16（两者均在首次扫描时初始化），则将新值 VW12 拷贝到 VW14、VW16 和 VW18 中。然后再分别对 VW16 和 VW14 的值减、加 3ms，作为新限值，而 VW18 中的平均值被传回主程序作为定时器 T34 的设定值。返回主程序时，VW20 中的定时器循环计数值加 1，并拷贝到输出字节（QB1），以供显示。

在方案 3 中，把电位器 0（POT 0）的值直接作为定时器 T35 的设定值，AC2 中的定时器循环计数值加 1，并拷贝到输出字节（QB0），以供显示。

本程序长度为 110 个字。

```
// 标题：模拟电位器：

// * * * * * * * * * * * * * * 主程序 * * * * * * * * * * * * * *

// 这是 S7-214 的一个演示程序，介绍了使用模拟电位器调整定时器设定值的三种方案。
// 方案 1：对来自 POT 0 的值进行换算并加偏移量，以调整定时器的设定值，可以从 200ms 调到的 1.48s。
          每个定时器周期 QB0 加 1。
// 方案 2：从 POT 1 来的值经过滤波给定时器提供 0ms 到约 2.55s 的稳定的设定值。每个定时器周期
          QB1 加 1。
// 方案 3：把 POT 0 的值直接作为定时器设定值。每个定时器周期 QB0 加 1。

// 模拟电位器 POT 0 和 POT 1 的值可以分别从 SMB28 和 SMB29 中以一个字节读出。
// 每次扫描时，POT 的值会变化一点，方案 1 和 2 都能为定时器提供稳定的设定值。
// 方案 1 的设定值会改变 1 次或 2 次，但每个定时器周期只装载一次。
// 方案 2 的设定值非常稳定，每次扫描都装载。
// 方案 3 的设定值每次扫描都会改变。

// 主程序：
LD SM0.1                    // 首次扫描时清除工作缓冲区：
MOVD+0，AC0                 // AC0=0。
MOVD+0，AC3                 // AC3=0。
MOVW+0，VW10                // VW10=0。
MOVW+32000，VW14            // 低限工作区复位。
MOVW+0，VW16                // 高限工作区复位。

// 方案 1：
// 每次扫描时 POT 的值会改变一点。
// 下面的指令用来在每个定时器周期捕获一次换算后的值，并提供一个稳定的定时器设定值。
LD      10.0                // 如果输入 10.0 为 1 状态，则选方案 1。
TON     T33，VW0            // POT 0 的值经过运算后作为 T33 的设定值。
CALL    1                   // 调用子程序 1 对 POT 0 的值进行换算并加偏移量。

LD      T33                 // 若 T33 计时到，
INCW    AC2                 // 则 AC2 加 1，即定时器循环计数。
MOVB    AC2，QB0            // 把 AC2 的最低有效字节拷贝到输出字节 QB0，以供显示。
R       T33，1              // 定时器 T33 复位

// 方案 2：
LD      10.1                // 如果输入 10.1 为 1 状态，则选方案 2
CALL    2                   // 调用子程序 2，对 POT 1 的值进行滤波运算后存入 VW18。
TON     T34，VW18           // VW18 的值作为 T34 的设定值。
LD      T34                 // 若 T34 计时到，
INCW    VW20                // 则 VW20 加 1，即定时器循环计数。
MOVB    VB21，QB1           // 把 VW20 最低有效字节（VB21）拷贝到输出字节 QB1，以供显示。
R       T34，1              // 定时器 T34 复位
```

```
// 方案 3：
LD      10.2                  // 如果输入 10.2 为 1 状态。
AN      10.0                  // 且方案 1 不在运行（10.0=0），则选方案 3。
MOVW    0，AC1                // 清除累加器 1（AC1）
MOVB    SMB28，AC1            // 送 POT 0 的值到 AC1。
TON     T35，AC1              // POT 0 的值作为 T35 的设定值。
LD      T35                   // 若 T35 计时到，
INCW    AC2                   // 则 AC2 加 1，即定时器循环计数。
MOVB    AC2，QB0              // 把 AC2 最低有效字节拷贝到输出字节 QB0，以供显示。
R       T35，1                // 定时器 T35 复位。
MEND                          // 主程序结束

// 方案 1 的子程序
SBR     1                     // 子程序 1。
// 换算 POT 0 的值并加上偏移量后存在 VW0 中，再返回主程序。
LD      T33                   // 每个定时器周期检查 POT 0 的变化。
MOVW    0，AC1                // 清除累加器 1（AC1）。
MOVB    SMB28，AC1            // 送 POT 0 的值给 AC1。
DIV     2，AC1                // AC1 除 2，即把 POT 0 的输入范围从 0~255 换算成 0~127。
+1      20，AC1               // 加 200ms 偏移量。
MOVW    AC1，VW0              // 把 AC1 值拷贝到 VW0，以便能上程序员读取。
RET                           // 子程序 1 结束。

// 方案 2 的子程序
SER     2                     // 子程序 2。
// 对 POT 1 值采样 100 次，然后求平均值。
INCW    VW10                  // 扫描计数器加 1。
MOVB    SMB29，AC0            // 送 POT 1 的值到 AC0。
+1      AC0，AC3              // 再加到以前的总和中（即累加 POT1 的值，共累加 100 次）。

LDW=    VW10，100             // 100 次扫描之后。
DIV     100，AC3              // 求平均值。
MOVW    AC3，VW12             // 存平均值。
MOVW    0，VW10               // 扫描计数器复位。
MOVD    0,AC3                 // 工作内存复位。
AW<=    VW12，VW14            // 检查新的平均值是否在保护区之外。
OW>=    VW12，VW16            //
FILL    VW12，VW14，3         // 把新的平均值存入 VW14，VW16，VW18。
-1      +3，VW14              // 设置新的低保护限。
+1      +3，VW16              // 设置新的高保护限。
RET                           // POT 1 的滤波值存在 VW18 中，返回主程序
```

4 怎样使用高速计数器

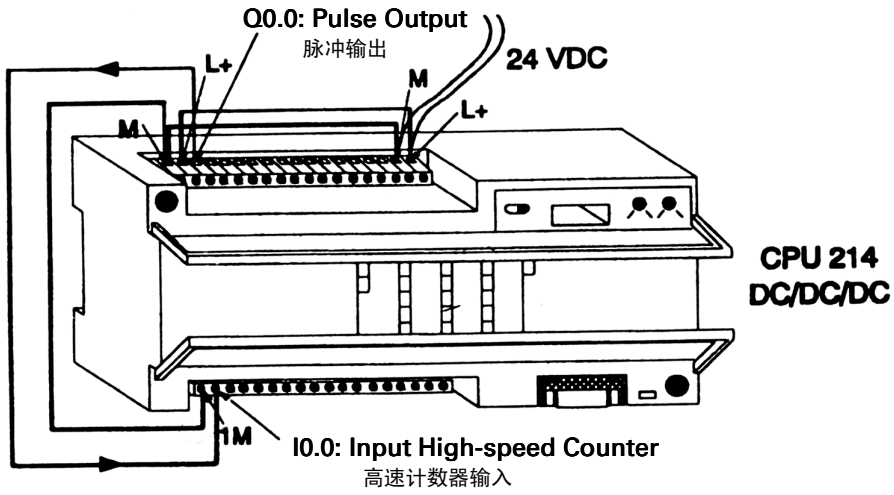
概述

本例叙述 SIMATIC S7-200 的高速计数器（HSC）的一种组态功能。对来自传感性（如编码器）信号的处理，高速计数器可采用多种不同的组态功能。

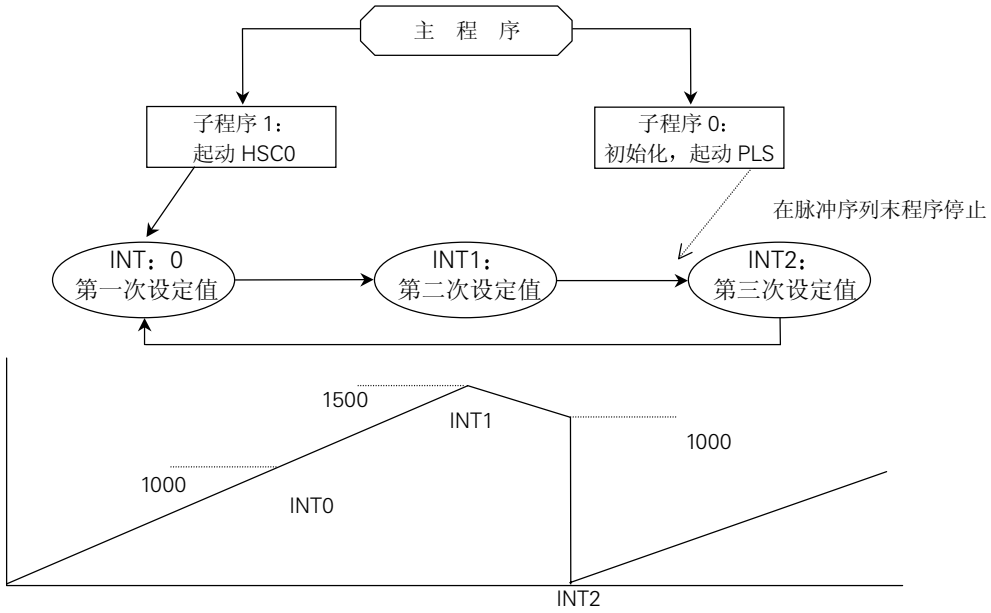
本例用脉冲输出（PLS）来为 HSC 产生高速计数信号，PLS 可以产生脉冲串和脉宽调制信号，例如用来控制伺服电机。既然利用脉冲输出，必须选用 CPU214DC/DC/DC。

下面这个例子，展示了用 HSC 和脉冲输出构成一个简单的反馈回答，怎样编制一个程序来实现反馈功能。

例图



程序框图



程序和注释

本例描述了 S7—200 DC/DC/DC 的高速计数器（HSC）的功能。HSC 计数速度比 PLC 扫描时间快得多，采用集成在 S7—212 中的 2kHz 的软件计数器进行计数。S7—214 除了有 2kHz 的计数器外，还有两个 7kHz 的硬件计数器。总的来说，每个高速计数器需要 10 个字节内存用来存控制位、当前值、设定值、状态位。

本程序长度为 91 个字。

```
// 主程序：
// 在主程序中，首先将输出 Q0.0 置，0，因为这是脉冲输出功能的需要。再初始化高速计。
// 数器 HSC0，然后调用子程序 0 和 1。
// HSC0 起动后具有下列特性：可更新 CV 和 PV 值，正向计数。
// 当脉冲输出数达到 SMD72 中规定的个数后，程序就终止。

// 主程序
LD      SM0.1           // 首次扫描标志（SM0.1=1）。
R       Q0.0，1         // 脉冲串输出 Q0.0 复位（Q0.0=0）。
MOVB    16#F8，SMB37    // 装载 HSC0 的控制位：
                        //       激活 HSC0，可更新 CV，可更新 PV，
                        //       可改变方向，正向计数。
                        // HSC 指令用这些控制位来组态 HSC。
MOVD    0，SMD38        // HSC0 当前值（CV）为 0。
MOVD    1000，SMD42     // HSC0 的第一次设定值（PV）为 1000。
HDEF    0，0            // HSC0 定为模式 0。
```

```
CALL    0                       // 调用子程序 0。
CALL    1                       // 调用子程序 1。

MEND                            // 主程序结束。

//* * * * * * * * * * * * * * * * * * * * * * * * * * * *

// 子程序 0：
// 子程序 0 初始化，并激活脉冲输出（PLS）。
// 在特殊存储字节 SMB67 中定义脉冲输出特性：脉冲串（PT0），时基，可更新数值，激活 PLS。
// SMW68 定义脉冲周期，其值为时基的倍数。
// 最后，在 SMD72 中指定需要产生的脉冲数。（SMD72）为内存双字，即 4 个字节）。

// 子程序 0
SBR     0                       // 子程序 0
MOVB    16#8D，SMB67            // 装载脉冲输出（PLS0）的控制位：PT0，时基 1ms，可更新，激活。
MOVW    1，SMW68                // 脉冲周期 1ms。
MOVD    30000，SMD72            // 产生 30000 个脉冲。
PLS     0                       // 起动脉冲输出（PLS 0），从输出端 Q0.0 输出脉冲。
RET                             // 子程序 0 结束。

//* * * * * * * * * * * * * * * * * * * * * * * * * * * *

// 子程序 1：
// 子程序 1 起动 HSC0，并把中断程序 0 分配给中断事件 12（HSC 0 的当前值 CV 等于设定值 PV）。
// 只要脉冲计数值（当前值 CV）达到设定值（PV），该事件就会发生。
// 最后，允许中断。

// 子程序 1
SBR     1                       // 子程序 1。
ATCH    0，12                   // 把中断程序 0 分配给中断事件 12（HSC 0  的 CV=PV）。
ENI                             // 允许中断。
HSC     0                       // 按主程序中对 HSC 0 的初始组态特性，起动 HSC0。
RET                             // 子程序 1 结束。

//* * * * * * * * * * * * * * * * * * * * * * * * * * * *

// 中断程序 0：
// 当 HSC 0 的计数脉冲达到第一，设定值 1000 时，调用中断程序 0。
// 输出端 Q0.1 置位（Q0.1=1）。
// 为 HSC0 设置新的设定值 1500（第二设定值）。
// 用中断程序 1 取代中断程序 0，分配给中断事件 12（HSC0 的 CV=PV）。
// 中断程序 0
```

```
INT     0                       // 中断程序 0。
S       Q0.1，1                 // 输出端 Q0.1 置位（Q0.1=1）。
MOVB    16#A0，SMB37            // 重置 HSC0 的控制位，仅更新设定值（PV）。
MOVD    1500，SMD42             // HSC0 的下一个设定值为 1500（第二设定值）。
ATCH    1，12                   // 用中断程序 1 取代中断程序 0，分配给中断事件 12。
HSC     0                       // 起动 HSC0，为其装载新的设定值。
RETI                            // 中断程序 0 结束。

//* * * * * * * * * * * * * * * * * * * * * * * * * * * *
// 中断程序 1：
// 当 HSC0 的计数脉冲达到第二设定值 1500 时，调用中断程序 1。
// 输出端 Q0.2 置位（Q0.2=1）。
// HSC0 改成减计数，并置新的设定值 1000（第三设定值）。
// 用中断程序 2 取代中断程序 1，分配给中断事件 12（HSC0 的 CV=PV）。

// 用中断程序 1
INT     1                       // 中断程序 1。
S       Q0.2，1                 // 输出端 Q0.2 置位（Q0.2=1）。
MOVB    16#B0，SMB37            // 重置 HSC 0 的控制位，更新设定值，并改成减计数（反向计数）。
MOVD    1000，SMD42             // HSC 0 的下一个设定值为 1000（第三设定值）。
ATCH    2，12                   // 用中断程序 2 取代中断程序 1，分配给中断事件 12。
HSC     0                       // 起动 HSC 0，为其装载新的设定值和方向。
RETI                            // 中断程序 1 结束。

//* * * * * * * * * * * * * * * * * * * * * * * * * * * *
// 中断程序 2：
// 当 HSC0 的计数脉冲达到第三设定值 1000 时，调用中断程序 2。
// 输出端 Q0.1 和 Q0.2 复位（Q0.1=0，Q0.2=0）。
// HSC0 的计数方向重新改为正向（增计数），并将当前计数值置为 0，而设定值 PV 保持不变（10000）。
// 重新把中断程序 0 分配给中断事件 12，程序再次起动 HSC 0 运行。
// 当脉冲数达到 SMD72 中规定的个数后，程序就终止。

// 中断程序 2
INT     2                       // 中程序 2。
R       Q0.1，2                 // 输出端 Q0.1 和 Q0.2 复位（Q0.1=Q0.2=0）。
MOVB    16#D8，SMB37            // 重置 HSC 0 的控制位，更新 CV，改为正向计数（增计数）。
MOVD    0，SMD38                // HSC0 的当前值复位（CV=0）。
ATCH    0，12                   // 把中断程序 0 分配给中断事件 12。
HSC     0                       // 重新起动 HSC 0。
RETI                            // 中断程序 2 结束。
```

请参考 SIMATIC STEP 7 编程参考手册 6.1 节“高速计数器指令”和 6.3 节“高速输出指令”，为您提供了更多的关于高速计数器和脉冲序列的信息。

5 自由通信口模式的简单应用

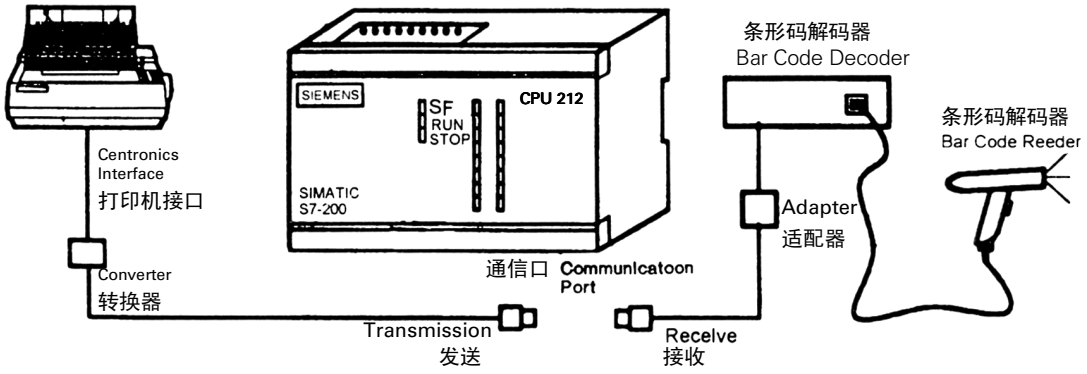
概述

自由通信口模式（Freeport Mode）的通信协议可自由定义，通信所需要的信息存放在特殊存储字节 SMB30 中，用户须作如下说明：

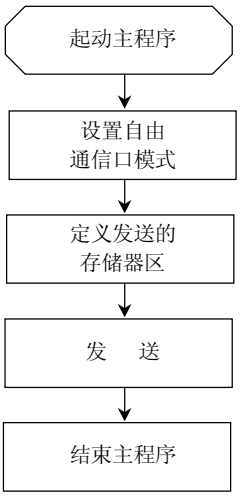
- 奇偶校验
- 每个字符的位数
- 波特率

自由通信口模式可以接收和发送数据。本例用一个仿真的打印机程序来描述数据发送，再用一个条形码阅读程序来说明数据接收。

例图



打印机程序框图



打印机程序和注解

此程序描述向打印机发送数据。为了简化此例，窗口下的终端程序可代替打印机作为接收器边接。打印机或终端的组态特性为 9600 波特，无奇偶校验，每字符 8 位。

本程序长度为 13 个字。

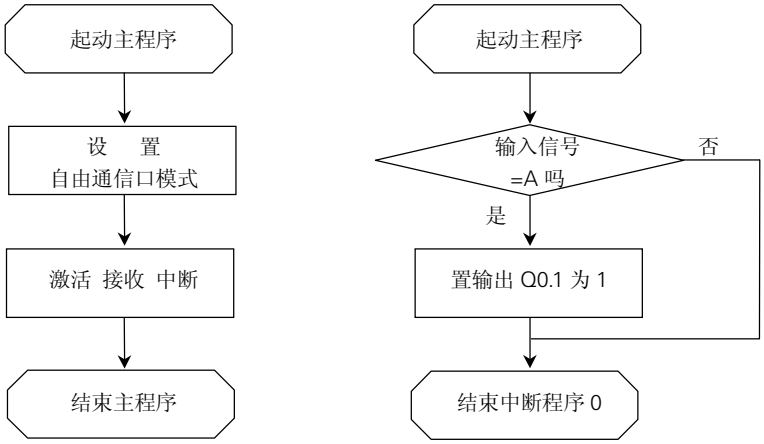
// 正确设置自由通信口模式对此应用很重要。
// 所需信息装载在特殊存储字节 SMB30 中。
// 这些输入数据可从操作手册中查询。
// 发送命令 XMT 包含了发送信息缓冲区的起始地址，该地址单元中只包含了发送信息的长度（以字节为单位）。

```
LD      SM0.1           // 第一次扫描（SM0.1=1）
MOVB    +9，SMB30       // 自由通信口模式；9600 波特，无奇偶校验，每字符 8 位。
MOVB    +1，VB100       // 信息长度为 1 个 ASCII 字符。
MOVB    16#41，VB101    // A 字符长度为 1 个字节，A=41H（十六进制）。
LD      10.1            // 输入 10.1 起动发送。
EU                      // 识别脉冲上升沿。
XMT     VB100，0        // 发送。
MEND                    // 主程序结束。
```

请参与 SIMATIC STEP 7 编程参考手册 2.6 节“特殊存储位”和 6 .4 节“发送指令”，为您提供了更多的关于自由通信口模式的通信接口的组态信息。

 西门子 版权所有

条形码阅读器程序框图



条形码阅读器程序和注解

该程序描述数据接收，条形码阅读器通过接口把读到的数据用自由通信口模式发给 SIMATIC S7-200。为简化此例，窗口下的终端程序可代替条形码阅读器作为发送器连接。

本程序长度为 15 个字。

```
// * * * * * * * * * * * * * * 主 程 序 * * * * * * * * * *

// 正确设置自由通信口模式对此应用很重要。
// 所需信息装载在特殊存储字节 SMB30 中。
// 这些输入数据可从操作手册中查询。
// 接收数据借助于中断实现，当数据进入自由编程接口，接收中断事件（8）。
// 就被触发了。
// 在此应用中，将中断程序 0（INT0）赋予接收中断事件（8）。

LD      SM0.1                   // 第一次扫描标志（SM0.1=1）。
MOVB    +9，SMB30               // 自由通信口模式：9600 波特，无奇偶校验，每字符 8 位。
ATCH    +0，8                   // 指定接收中断事件 8 调用中断程序 0。
ENI                             // 允许中断。
MEND                            // 结束主程序。
//* * * * * * * * * 中断程序 0 * * * * * * * *
// 在中断程序 0，把存放在特殊存储字节 SMB2 中的接收字符和大写字母 A 作比较。
// 如果符合，则置输出位 Q0.1 为 1。
INT                             // 接收中断程序 0。
LDB=    SMB2，16#41             // 字节 SMB2 中的接收字符和 A 比较。
S       Q0.1，1                 // 若字符为 A，则置 Q0.1 为 1。
RETI                            // 返回主程序。
```

请参考 SIMATIC STEP 7 编程参考手册 2. 6 节“特殊存储位”和 6.2 节“中断指令”，为您提供了更多的关于自由通信口模式接口的信息。

 西门子 版权所有

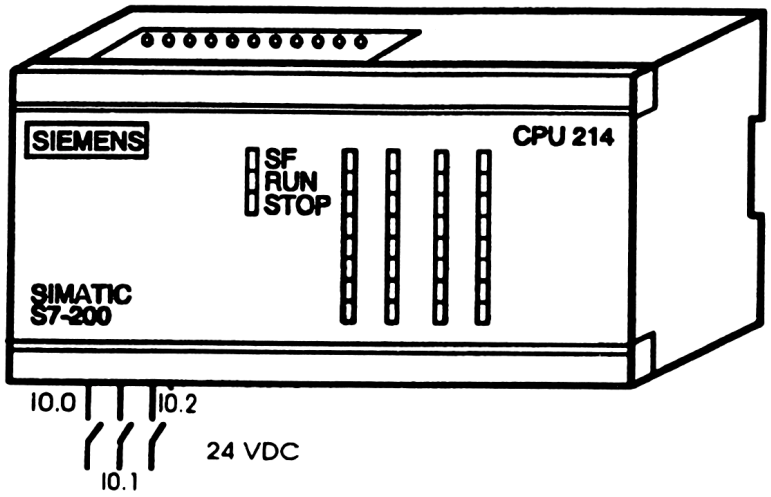
6 设置位或字节的几种方法

概述

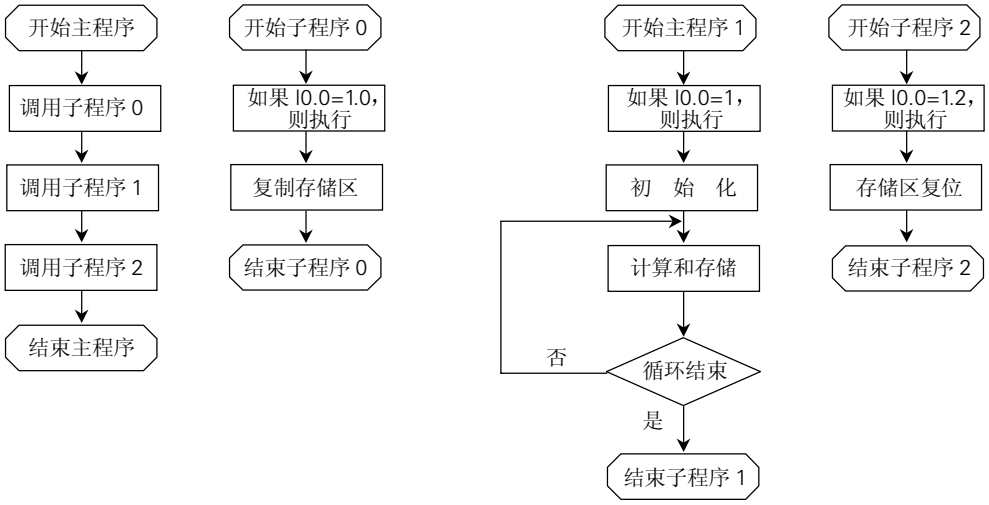
本程序描述了用一定值存入预定的存储区域或对预定的存储区域清零的几种方法。
本例采用了以下指令：

- FILL 指令。
- FOR……NEXT 循环指令。
- RESET 指令。

例图



程序框图



程序和注释

本例程序描述了用一定值存入预定的存储器位或字节，以及清除存储区内容的几种方法。采用指令如下：

FILL	设置一个字或几个字。
FOR……NEXT	FOR……NEXT 循环。
R	对一位或几位置 0。

本程序长度为 55 个字。

```
// 标题：FORNEXT
* * * * * * * 主 程 序 * * * * * * *
// 主程序包括对子程序 0、1、2、的调用。
// 主程序
LD      SM0.0                   // SM0.0 总是 1。
CALL    0                       // 调用子程序 0（FILL）。
CALL    1                       // 调用子程序 1（FOR……NEXT）。
CALL    2                       // 调用子程序 2（RESET）。
MEND                            // 主程序结束。
* * * * * * * 子 程 序 0 * * * * * * *
// 子程序 0 功能：如果输入 10.0=1，则把 VW200 中值复制到 VW204 至 VW216。
// 子程序 0
SBR     0                       // 子程序 0。
LD      10.0                    // 读输入 10.0，如果 10.0=1，则
MOVW    16#ABC3，VW200          // 将十六进制数 ABC3 写入 VW200。
FILL    VW200，VW204，7         // 把 VW200 中值复制到 VW204 至 VW216。
RET                             // 子程序 0 结束。
```

 西门子 版权所有

```
* * * * * * * 子 程 序 1 * * * * * * *
// 当输入 10.1=1 时，把几个初始数复制到变量存储区。
// 循环次数取决于 VW10 中的首次循环数和 VW0 中的最后一次循环数。
// 当前循环次数存储在内存字 VW20 中。
// 首次计数值（50）存储在累加器 AC0 中。
// 计数值缓冲区首址（&VB100）存储在累加器 AC1 中，AC1 为计数值缓冲区指针。
// 每循环一次的功能：
// AC0 的计数值存入 AC1 指针所指向的内存单元。
// AC1 指针加 2 个字节，指向下一个内存字。
// AC0 的计数值加 4。
// 直到最后一次循环。

// 字程序 1
SBR     1                   // 子程序 1。
LD      10.0                // 读输入 10.1，如果 10.1=1，则
MOVW    +10，VW0            // 把最后一次循环数存入 VW0。
MOVW    0，VW10             // 把第一次循环数存入 VW10。
MOVW    0，VW20             // 把当前循环数存入 VW20（计数器）。
MOVW    50，AC0             // 把开始计数值存入累加器 0（AC0）。
MOVD    &VB100，AC1         // 累加器 1（存储区指针）指向存储字节 VB100。
                            //
FOR     VW20，VW10，VW0     // 循环开始
MOVW    AC0，*AC1           // 把当前计数值存入当前存储地址中。
INCD    AC1                 // 存储区指针加一个字节。
INCD    AC1                 // 存储区指针加一个字节。
+1      4，AC0              // 给当前计数值加 4。
NEXT                        // 循环结束。
RET                         // 子程序 1 结束。

* * * * * * * 子 程 序 2 * * * * * * *
// 如果输入 10.2=1，则把存储器位 V100.0 至 V121.7 及 V204.0 至 V217.7 置 0

// 字程序 2
SBR     2                   // 子程序 2。
LD      10.2                // 读输入 10.2，如果 10.2=1，则
R       V100.0，176         // 把 V100.0 至 V121.7 置 0。
R       V204.0，112         // 把 V204.0 至 V217.7 置 0。
RET                         // 子程序 2 结束。
```

请参考 SIMATIC STEP 7 编程参考手册 4.7 节“FILL 指令”，5.6 节“FOR……NEXT 指令”和 3.5 节“RESET 指令”，为您提供了更多的关于对存储器位和字节置位和复位的信息。

 西门子 版权所有

7 处理脉宽调制

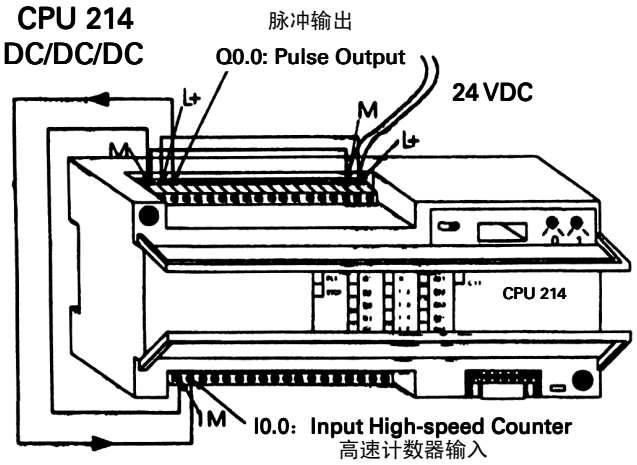
概述

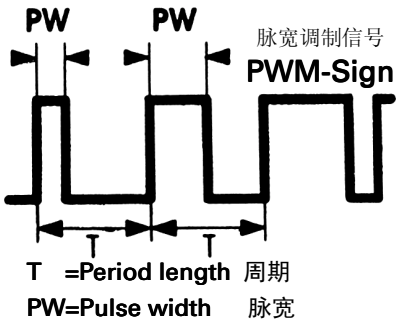
在 S7-200 系列中，CPU-214 的输出端 Q0.0 和 Q0.1 能够输出方波信号，而且方波信号的周期和脉宽均能独立调节，其中脉宽指的是在一个周期内，输出信号处于高电平的时间长度。

下面这个例子说明了脉宽调制（PWM）是如何工作的。输出端 Q0.0 输出方波信号，其脉宽每周期递增 0.5 秒，周期固定为 5 秒，并且脉宽的初始值为 0.5 秒。当脉宽达到设定的最大值 4.5 秒，脉宽改为每周期递减 0.5 秒，直到脉宽为零为止。以上过程周而复始。

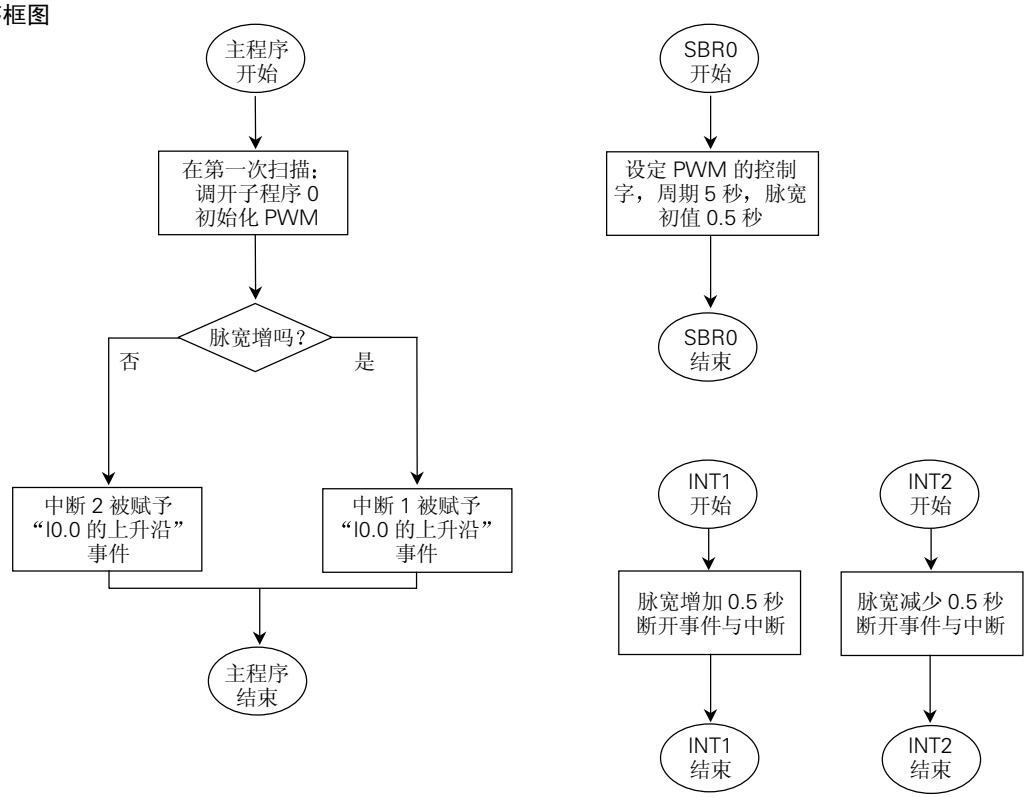
在这个例子中必须把输出端 Q0.0 与输入端 10.0 连接，这样程序才能控制 PWM。

例图





程序框图



程序和注解

特殊存储字节 SMB67 用来初始化输出端 Q0.0 的 PWM。这个控制字内含 PWM 允许位，修改周期和脉宽的允许位，以及时间基数选择位等，由子程序 0 来调整这个控制字节。通过 ENI 指令，使所有的中断成为全局允许，然后通过 PLS0 指令，使系统接受各设定值，并初始化“PTO/PWM 发生器”，从而在输出端 Q0.0 输出脉宽调制（PWM）信号。

另外，周期 5 秒是通过将数值 5000 置入特殊存储字 SMW68 来实现的，初始脉宽 0.5 秒则通过将 500 写入特殊存储字 SMW70 来实现的。

这个初始化过程是在程序的第一个扫描周期通过执行子程序 0 来实现，第一个扫描周期标志是 SM0.1=1。当一个 PWM 循环结束，即当前脉宽为 0 秒时，将再一次初始化 PWM。

辅助内存标记 M0.0 用来表明脉宽是增加，还是减少，初始化时将这个标记设为增加。输出端 Q0.0 与输入端 10.0 相连，这样输出信号民可送到输入端 10.0。当第一个方波脉冲输出时，利用 ATCH 指令，把中断程序 1（INT1）赋给中断事件 0（10.0 的上升沿）。

每个周期中断程序 1 将当前脉宽增加 0.5 秒，然后利用 DTCH 指令分离中断 INT1，使这个中断再次被屏蔽。如果在下次增加时，脉宽大于或等于周期，则将辅助内存标记位 M0.0 再次置 0。这样就把中断程序 2 赋予事件 0，并且脉宽也将每次递减 0.5 秒。当脉宽值减为零时，将再次执行，初始化程序（子程序 0）。

本程序长度为 63 个字。

```
// 标题：处理脉宽调制
* * * * * * * 主 程 序 * * * * * * *

LD      SM0.1                  // 在第一个扫描周期 SM0.1=1。
CALL    0                      // 调用子程序 0 来起动 PWM，即初始化 PWM。

LDW>=SMW70，VW0                // 如果脉宽大于等于（周期-脉宽），
R       M0.0，1                // 则将辅助内存标记位 M0.0 置 0。

LDW=    SMW70,0                // 如果脉宽为零。
CALL    0                      // 则调用子程序 0 来重新开始一个完整的 PWM。

LD      10. 0                  // 如果输入 10.1=1。
A       M0.0                   // 且辅助内存标记位 M0.0=1（脉宽增加），
ATCH    1，0                   // 则把 INT1 赋给事件 0（输入 10.0 的正向上升沿）。

LD      10.0                   // 如果输入 10.0=1。
AN      M0.0                   // 且辅助内存标记位 M0.0=0（脉宽减少），
ATCH    2，0                   // 则把 INT2 赋给事件 0（输入 10.0 的正向上升沿）。

MEND                           // 主程序结束。

* * * * * * * 主 程 序 0 * * * * * * *

SBR     0                      // 初始化脉宽调制。
S       M0.0，1                // 将增加脉宽的辅助内存标记位 M0.0 置 1。
MOVB    16#CB，SMB67           // 设定输出端 Q0.0 的 PTO/PWM 控制字节

// SM67.0：=1      ⇒允许接受新的周期。
// SM67.1：=1      ⇒允许接受新的脉宽。
// SM67.3：=1      ⇒时间基数为 1ms（基为 0，则时间基数为 1 μ s）。
// SM67.6：=1      ⇒选择 PWM 模式（若为 0，则 PTO 模式）。
// SM67.7：=1      ⇒允许高速输出功能。

MOVW    500，SMW70             // 指定初始脉宽（500ms）。
MOVW    500，SMW68             // 周期为 5s。
ENI                            // 允许全部中断。
PLS0                           // 对 PTO/PWM 生成器编程的指令。

MOVW    SMW68，VW0             // 将周期置入数据字 VW0。
-1      500，VW0               // 将（周期一脉宽）的值置入数据字 VW0。
RET                            // 子程序 0 结束并返回主程序。
```

```
* * * * * * * 中断服务程序 1 * * * * * * *

INT     1                   // 增加脉宽。
+1      500，SMW70          // 脉宽增加 500ms。
PLS     0                   // 对 PTO/PWM 生成器编程的指令。
DTCH    0                   // 将中断与事件 0 断开。
RETI                        // 中断服务程序 1 结束，并返回主程序。

* * * * * * * 中断服务程序 2 * * * * * * *

INT     2                   // 减少脉宽。
-1      500，SMW70          // 脉宽减少 500ms。
PLS     0                   // 对 PTO/PWM 生成器编程的指令。
DTCH    0                   // 将中断与事件 0 断开。
RETI                        // 中断服务程序 2 结束，并返回主程序。
```

请参考 SIMATIC STEP 7 编程参考手册 6.3 节“高速输出指令”，为您提供了更多的关于脉宽调制的信息。

8 怎样读和写 S7-214 的实时时钟

概述

这个程序示例涉及到关于实时时钟的两种特殊指令：读和写日期及时钟时间。为了进行这些操作，需要有如下结构的 8 字节缓冲区：

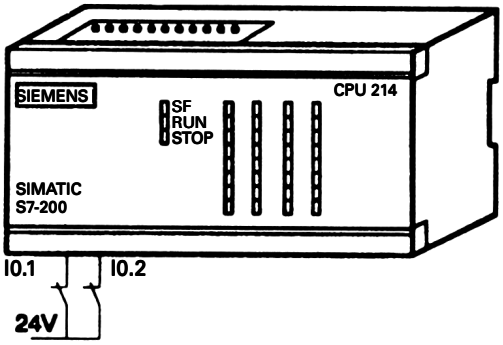
字节 0：年（00—99）　　字节 4：分（00—59）
字节 1：月（1—12）　　字节 5：秒（00—59）
字节 2：日（1—31）　　字节 6：未分配
字节 3：时（00—24）　　字节 7：星期（1—7=Su~Sa）

为了读或写方便，这些数据用 BCD 码存储。当操作开关 10.0 为 1 时，就将预定日期和时间写入实际时钟。为了显示当前的秒值，将其值拷贝到输出字节 QB0。当 10.1=1 时，则用 BCD 码显示；当 10.1=0 时，则用二进制码显示。

例图

用 BCD 码显示秒

I0.1 = ON： Second Display in BCD Format
I0.1 = OFF： Second Display Binary
用二进制显示秒

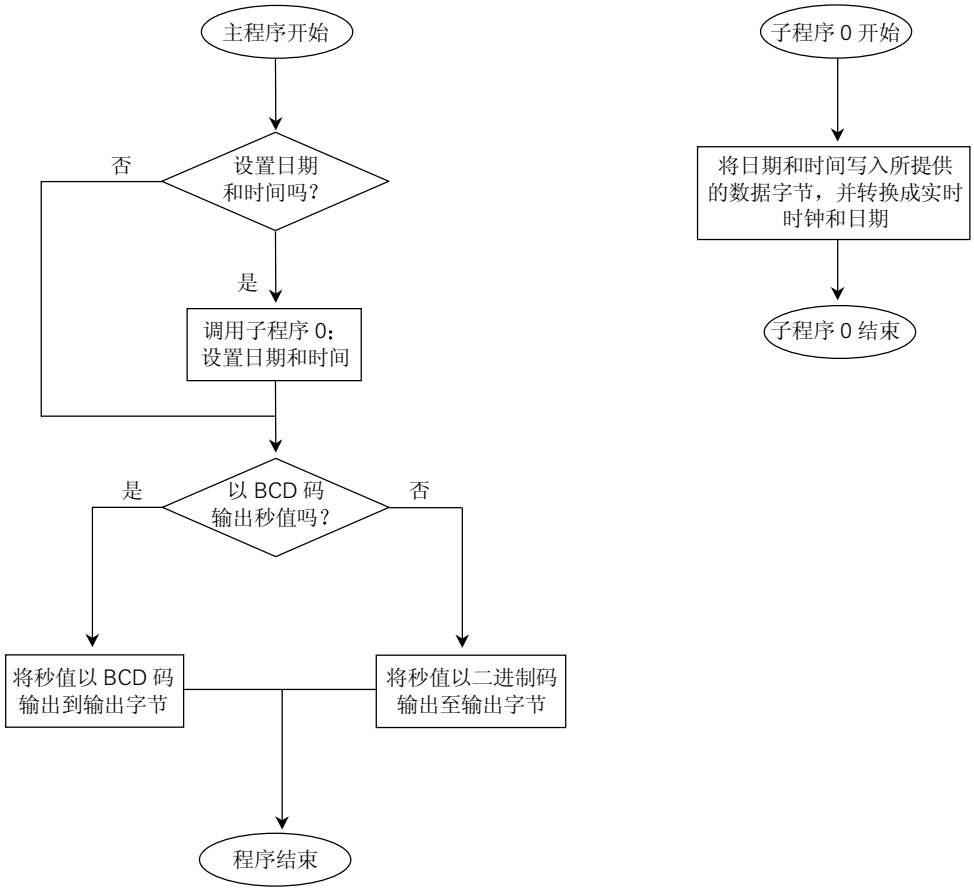


置日期

I0.0➡ Set Dete: 02/14/95 and
Clock Time:12:00:00
时间

95	VB400	Year	年
02	VB401	Month	月
14	VB402	Day	日
12	VB403	Hour	时
00	VB404	Minute	分
00	VB405	Second	秒
00	VB406		
03	VB407	Day of week	星期几

程序框图



程序和注释

通过按输入开关 10.0，可调用子程序 0。这个子程序按照要求的日期和时间，预先将其值置入 VB100 到 VB107 这 8 个字节，然后用 TODW 指令，将此设置传送给实时时钟。

每个周期都读出实时时钟的值，这些数据以 BCD 码形式（4 位代表 0 至 9 的数字）存储在 VB400 到 VB407 这 8 个字节中。如果输入 10.1 为 1，这些值就被直接拷贝到输出字节 QB0，以供显示。

如果输入开关 10.1 为 0，将数据定 VW404 拷贝到 VW204，再将包含分钟值的 VB204 清零。这一步是必须的，因为把秒值从 BCD 码形式转换成二进制码形式，只能按字来转换。现时的二进制码的秒值被传输到输出字节 QB0，以供显示。

本程序长度为 46 个字。

```
// 标题：实时时钟

* * * * * * * 主程序 * * * * * * * *

LD      10.0                    // 写实时时钟的开关，若按 10.0 开头。
EU                              // 上升沿，则，
CALL    0                       // 调用子程序 0，写实时时钟。
LD      SM0.0                   // 设置栈顶（SM0.0 总是 1）
TODR    VB400                   // 读出实时时钟数据并填入 8 字节缓冲区（VB400 至 VB407）。
LD      10.1                    // 以 BCD 码形式显示秒的开关，若 I10.1=1，则，
MOVB    VB405，QB0              // 将当前秒值拷贝到输出字节 QB0。
LDN     10.1                    // 以二进制码形式显示秒的开关，若 I10.1=0，则，
MOVW    VW404，VW204            // 将字 VW404 拷贝到字 VW204。
MOVB    0，VB204                // 清除高字节 VB204（分）。
BCDI    VW204                   // BCD 码转换为二进制码（秒）。
MOVB    VB205，QB0              // 当前秒值拷贝到输出字节 QB0。
MEND                            // 主程序结束。

* * * * * * * 子程序 0 * * * * * * * *

SBR     0                       // 设置日期和时间。
MOVB    16#95，VB100            // 年：95。
MOVB    16#02，VB101            // 月：2 月。
MOVB    16#14，VB102            // 日：14。
MOVB    16#12，VB103            // 时：12。
MOVB    16#0，VB104             // 分：00。
MOVB    16#0，VB105             // 秒：00。
MOVB    16#0，VB106             // 未分配。
MOVB    16#3，VB107             // 星期：星期二。
TODW    VB100                   // 写实时时钟。
RET                             // 子程序 0 结束。
```

请参考 SIMATIC STEP 7 编程参考手册 5. 7 节“实时时钟指令”和 5.5 节“转换指令”，为您提供了更多的关于实时时钟和数据转换的信息。

 西门子 版权所有

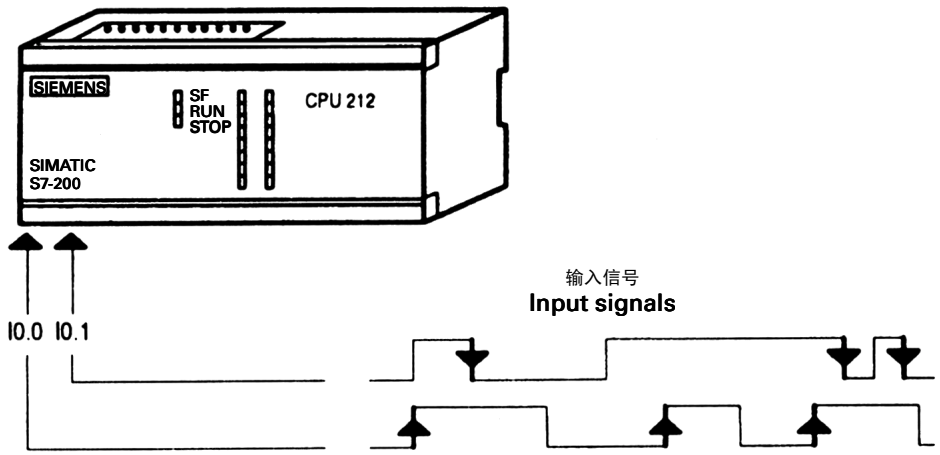
9 检测输入信号的边沿

概述

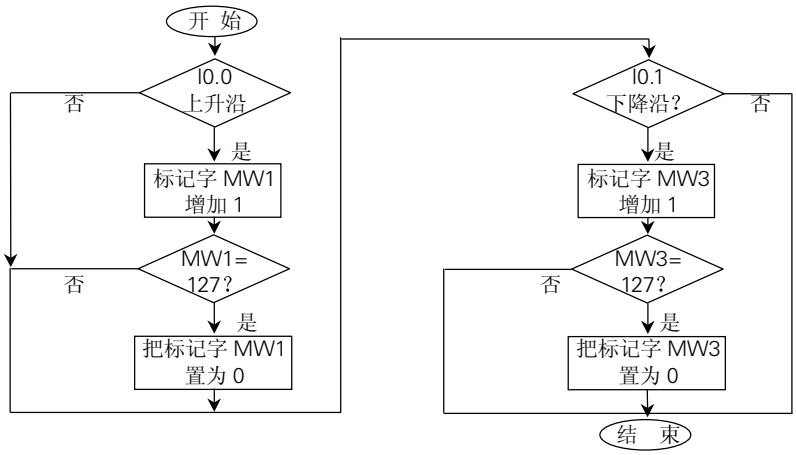
本例程序用来说明如何用 S7—200 的检测边沿指令来检测简单信号的变化。在这个过程中，用上升和下降来区分信号边沿，上升沿指信号由“0”变为“1”，下降沿指信号由“1”变为“0”。逻辑“1”表示输入上有电压，“0”表示输入上无电压。

程序用 2 个存储字分别累计输入 10.0 上升沿数目，以及输入 10.1 下降沿数目。

例图



程序框图



程序和注释

程序利用输入 10.0 和 EU（上升沿）指令来判定上升沿变化是否发生，也就是说，信号由“0”变为“1”。如果一个上升沿变化发生了，那么存储字 MW1 的值增加 1。ED（下降沿）指令用来计数输入 10.1 的下降沿，用存储字 MW3 来计数。如果某一个存储字计数达到 127，那么该存储字被重新置为 0。注意 MB2 是存储字 MW1 的低字节，MB1 为高字节。同样的，MB4 为存储字 MW3 的低字节，MB3 为高字节。

本程序长度为 27 个字。

```
// 标题：检测边沿

// 主程序
LD      SM0.1           // 仅在第一次扫描时 SM0.1=1。
MOVD    0，MD1          // 把存储器双字 MD1 置 0。

LD      10.0            // 输入信号 10.0。
EU                      // 上升沿。

+1      1，MW1          // 上升沿变化，使存储字 MW1 的值加 1。
LDW=    127，MW1        // 如果累计 127 个上升沿变化（或某个指定的数值）。
MOVW    0，MW1          // 则将存储字 MW1 置 0。

LD      10.1            // 输入信号 10.0。
ED                      // 下降沿。
+1      1，MW3          // 下降沿变化，使存储字 MW3 的值加 1。
LDW=    127，MW3        // 如果累计 127 个下降沿变化（或某个特定的数值）。
MOVW    0，MW3          // 则将存储字 MW3 置 0。

MEND                    // 主程序结束
```

请参考 SIMATIC STEP 7 编程参考手册 3.7 节“特殊触点指令”，为您提供了更多的关于检测信号边沿的信息。

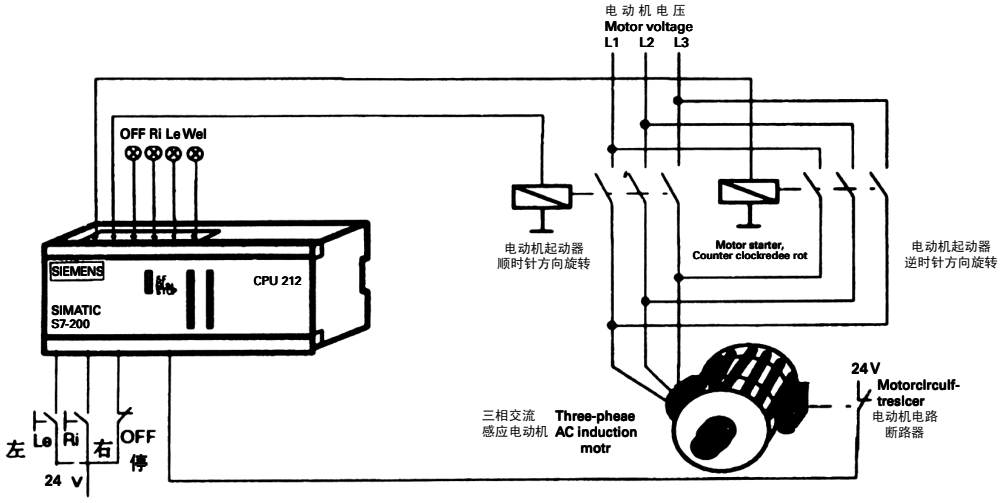
10 可逆电动机起动器电路——适用于改变三相交流感应电动机旋转方向

概述

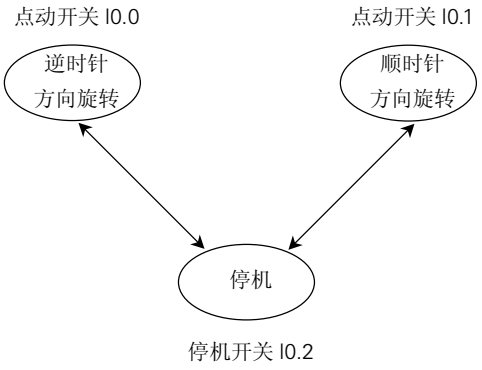
这个示例程序用于控制可双向运转的三相感应电动机。

当与输入点 10.0 相连的左转点动开关（Le）闭合时，电动机逆时针方向旋转，当与输入点 10.1 相连的右转点动开关（Ri）闭合时，电动机顺时针方向旋转。但这要有一个前题，即与输入点 10.3 相连的电动机电路断路器和与输入点 10.2 相连的停机开关（OFF）都没有动作。只有按下停机开关，并等待 5 秒钟之后，才可以改变电动机的旋转方向。这样做是为了让电动机有足够的时间刹车停转，然后再反向起动，如果需要电动机反转的话。如果与 10.0 和 10.1 相连的点动开关同时按下，电动机停转，并且不起动。

例图



程序框图



程序和注释

在程序起始部分，程序检查是否必须激活互锁电路。互锁电路防止电动机误起动，或者按错误方向起动。只有当所有点动开关都没有动作（位于起始状态）或者等待时间溢出时，互锁才清除，即 M2.0 被置成逻辑 0。

如果电动机断路器（输入点 10.3）没有动作，停机点动开关（输入点 10.2）也没有动作（这两个触点都是常闭触点）；并且状态位 M1.1 没有被设置成顺时针旋转标志，则使能位 M2.1 被置为逻辑 1。电动机才有可能逆时针旋转。代表逆时针旋转的状态位是 M1.0。用类似方法可得到顺时针方向旋转的起动条件。

当点动起动开关（Ie 和 Ri）这一动作，并且互锁位和状态位都没有被设置成相反的旋转方向时，电动机起动。即相关的输出位和状态位被置位，状态位的作用是使输出能够自保。电动机逆时针方向旋转起动器由输出点 Q0.0 控制。电动机顺时针方向旋转起动器由输出点 Q0.1 控制。

除此外，另有一组信号灯指示电动机当前的运行状态；逆时针方向旋转指示灯（Le）与输出点 Q0.4 相连；顺时针方向旋转指示灯（Ri）与输出点 Q0.3 相连；关电机指示灯（OFF）与输出点 Q0.2 相连。

当电动机被停机时，“ED”的下降沿将辅助存储位 M2.3 置为 1，进入停机模式。当 M2.3 被置位时，限制电动机再次起动的定时器开始计时，该定时器的预置时间是 5 秒（500×10ms），经过 5 秒钟后，内部存储器位 M2.3 被复位。在这段强制等待时间内与输出点 Q0.5 相连的信号灯（Wait）闪烁。如果状态位都没有被置位，则点亮与输出点 Q0.2 相连的停机状态指示灯（OFF）。

该程序的长度为 61 个字。

```
// 标题：可逆电动机起动器电路

// 互锁：

LD      10.1            // 如果既命令电动机右转（Ri）。
A       10.0            // 又命令电动机左转（Le）。
O       M2.3            // 或处于强制等待状态，则
S       M2.0，1         // 设置互锁（M2.0=1）。

// 解除互锁

LDN     10.1            // 如果既无左转命令（Le）。
AN      10.1            // 也无右转命令（Ri）。
AN      M2.3            // 并且等待时间溢出，则
R       M2.0，1         // 解除互锁（M2.0=0）。

// 逆时针方向旋转使能

LD      10.2            // 如果无停机命令（OFF），
A       10.3            // 且电路断路器未动作，
AN      M1.1            // 且顺时针方向旋转状态位未置位，
=       M2.1            // 则逆时针方向旋转使能位 M2.1=1。
```

```
// 顺时针方向旋转使能
LD      10.2            // 如果无停机命令（OFF），
A       10.3            // 且电路断路器未动作，
AN      M1.0            // 且逆时针方向旋转状态位未置位，
=       M2.2            // 则逆时针方向旋转使能位 M2.2=1。
// 逆时针方向旋转
LD      10.0            // 如果命令电动机左转（Le）。
O       M1.0            // 或逆时针方向状态位，
AN      M2.0            // 且无互锁，
A       M2.1            // 且逆时针方向旋转使能，则，
=       M1.0            // 置逆时针方向旋转状态位 M1.0=1。
=       Q0.0            // 置电动机起动器输出点 Q0.0=1。
=       Q0.4            // 点亮逆时针方向旋转信号灯（Le）。
// 顺时针方向旋转
LD      10.1            // 如果命令电动机右转（Ri）。
O       M1.1            // 或顺时针方向状态位，
AN      M2.0            // 且无互锁，
A       M2.2            // 且顺时针方向旋转使能，则，
=       M1.1            // 置顺时针方向旋转状态位 M1.1=1。
=       Q0.1            // 置电动机起动器输出点 Q0.1=1。
=       Q0.3            // 点亮顺时针方向旋转信号灯（Ri）。
// 检测边沿，关机过程
LDN     M1.0            // 如果既无逆时针方向旋转状态位，
AN      M1.1            // 也无顺时针方向旋转状态位，则，
=       Q0.2            // 点亮关机输出信号指示灯（OFF）。
LD      Q0.2            // 若关机时，
ED                      // 检测下降沿，则，
S       M2.3，1         // 将辅助存储器标志位（代表关机状态）置位（M2.3=1）。
LD      M2.3            // 若为关机状态，则，
MOVW    500，VW20       // 装载重新起动前必须等待的时间值（500×10ms=5 ms）。
TON     T33，VW20       // 起动重新起动要强制等待的定时器（T33）。
A       T33             //
R       M2.3，1         // 超过等待时间后，将辅助存储器标志位复位（M2.3=0）。
MOVW    0，T33          // 等待定时器清 0。
// 关机状态指示，等待
LD      M2.3            // 辅助存储器标志位（等待）。
A       SM0.5           // 指示灯以 1 秒闪烁。
=       Q0.5            // 点亮等待信号灯（Wait）。
MEND                    // 主程序结束。
```

 西门子 版权所有

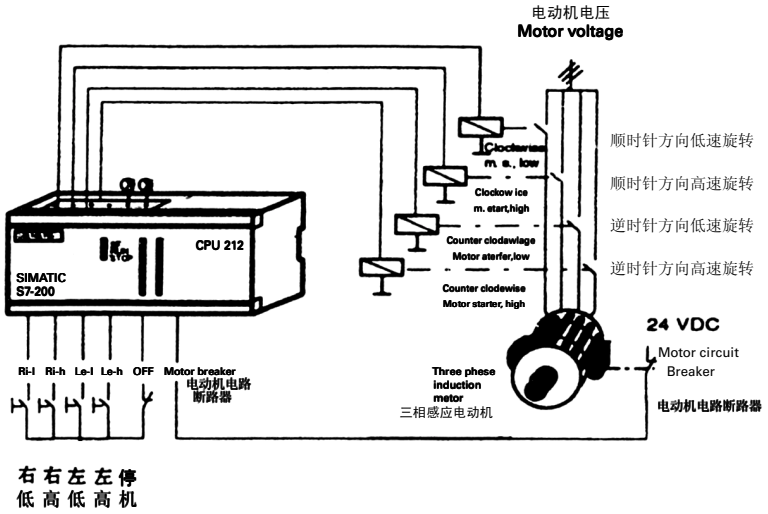
11 可逆电动机起动器——适用于旋转方向可选的变极调速的三相感应电动机

概述

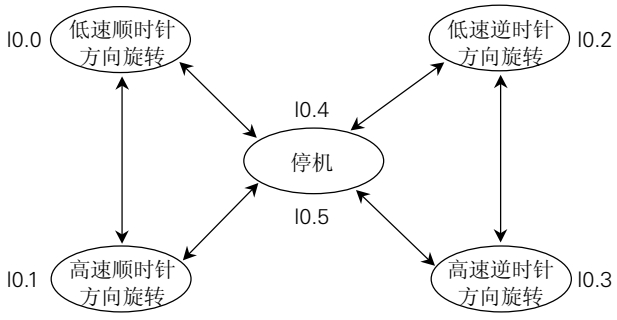
这个示例程序用来控制一台三相感应电动机，该电动机具有两个单独的绕组，对应不同的转速和旋转方向。该示例程序是对应用程序示例 10 的扩展。

当与各输入点相连的点动开关被按下时，电动机起动。不论电动机沿什么方向旋转，任何时候都可以改变电动机的转速，转速分高低两档。电动机需要 5 秒钟时间来刹车停机之后，才可以朝相反的方向起动。与输出点 Q0.5 相连的信号灯闪烁表示电动机正处于刹车状态。

例图



程序框图



程序和注释

这个示例程序用来控制一台具有两个单独绕组的三相感应电动机，该电动机可按不同的方向以不同的转速旋转。该示例程序是对应用示例 10 中的可逆起动器的扩展。在这个示例程序中，你除了可以选择电动机转向外，还可以选择电动机的转速。以下 4 只与输入点相连的点动开关用来起动电动机？

10.0——低速顺时针方向旋转（Ri—l）；
10.1——高速顺时针方向旋转（Ri—h）；
10.2——低速逆时针方向旋转（Le—l）；
10.3——高速逆时针方向旋转（Le—h）；

起动电动机之后，不论电动机沿哪个方向旋转，在任何时候都可以切换电动机的转速。如果需要改变电动机的旋转方向，必须先按下 OFF 点动开关来停机。电动机需要 5 秒钟时间刹车停机，之后才能反向起动。与输出点 Q0.5 相连的信号灯闪烁，表示电动机正处于刹车停机状态。

与输入点 10.4 相连的开关（OFF）用来关闭电动机。电动机电路断路器与输入点 10.5 相连。当电动机过载时，电动机电路断路触点断开，这也会导致电动机停机。当电动机停机时，点亮与输出点 Q0.4 相连的信号灯。

在程序扫描循环的起始处调用子程序 SBR1。该子程序的任务是检查输入信号。为了防止操作错误，首先要检查是否有两个或两个以上开关同时动作，然后，再检查在换向时是否已超过了限制等待时间。当禁止电动机起动器工作时，互锁存储器位 M1.0 被置为“1”。

互锁电路的作用是防止电动机误起动或朝错误方向起动。只有当所有的点动开关都没有动作（处于起始位置），且限时定时器溢出后，才解除互锁，即 M1.0 被置为逻辑“0”。

同时满足下述条件时，电动机才能以某种操作方式起动：停机开关（常闭触点，与 10.4 相连）没有动作；防止电动机过载的电动机断路器（常闭触点，与 10.5 相连）没有打开；并且电动机不是正在沿相反方向旋转（相反方向的状态位没有置位）。若改变转速也必须满足上述要求，否则命令无效。这就保证了两个电动机起动器不会同时起劲。也就是说，每个方向的高速运转起动器和低速运转起动器不同时工作。例如，当与输入点 10.3 相连的点动开关（Le—h）被按下，发出逆时针方向高速旋转命令时，负责锁住逆时针方向低速旋转的输出点 Q0.2 被复位。

只有当选择所期望的操作模式命令已发出，且互锁未生效时，才能锁定操作模式，即将相关的状态位 Q0.0，Q0.1，Q0.2，或 Q0.3 置位。

在该子程序结束处的边沿信号检测用来在电动机关机时刻起动限时等待定时器，以便使电动机有足够时间刹车。

退出子程序后，这些状态位的值被拷贝到输出点 Q0.0 至 Q0.3。这些输出点控制电动机起动器。输出点 Q0.4 和 Q0.5 分别用来指示“停机”状态和“限时等待”状态。

该程序的长度为 125 个字。

 西门子 版权所有

```
// 标题：双向变极调速三相感应电动机

// 输入

// 10.0    顺时针方向，低速，Ri—l。
// 10.1    顺时针方向，高速，Ri—h。
// 10.2    逆时针方向，低速，Le—l。
// 10.3    逆时针方向，高速，Le—h。
// 10.4    停机点动开关（带闭触点），OFF。
// 10.5    电机电路断路器（常闭触点）。

// 输出

// Q0.0    电动起动器，顺时针方向，低速。
// Q0.1    电动起动器，顺时针方向，高速。
// Q0.2    电动起动器，逆时针方向，低速。
// Q0.3    电动起动器，逆时针方向，高速。
// Q0.4    停机信号灯。
// Q0.5    限时等待信号灯。

LD      SM0.0           // 设置堆栈的栈顶（SM0.0 总是 1）。
CALL1                   // 调用子程序 1。

LD      10.0            // 若高速顺时针方向旋转命令（Ri—l）。
R       Q0.1，1         // 则复位高速顺时针方向旋转输出（Q0.1=0）。

LD      10.1            // 若高速顺时针方向旋转命令（Ri—h）。
R       Q0.1，1         // 则复位低速顺时针方向旋转输出（Q0.1=0）。
LD      10.2            // 若低速逆时针方向旋转命令（Le—l）。
R       Q0.3，1         // 则复位逆时针方向高速旋转输出（Q0.3=0）。

LD      10.3            // 若高速逆时针方向旋转命令（Le—h）。
R       Q0.2，1         // 则复位低速逆时针方向旋转输出（Q0.2=0）。

LDN     Q0.0
AN      Q0.1
AN      Q0.2
AN      Q0.3
=       Q0.4

// 强制等待状态显示

LD      M1，1           // 关机后设置的强制等待辅助存储器标志。
A       SM0.5           // 指示灯以 1 秒闪烁。
=       Q0.5            // 点亮强制等待信号灯。

MEND                    // 主程序结束

// 子程序 1

SBR1                    // 子程序 1 开始

// 互锁
```

```
LD      10.0            // 如果顺时针方向，低速旋转（Ri—l）。
O       10.1            // 或顺时针方向，高速旋转（Li—h）。
O       10.2            // 或逆时针方向，低速旋转（Le—l）。
O       10.3            // 或逆时针方向，高速旋转（Ri—h）。
AN      10.4            // 与停机点动开关（OFF）动作。

LD      10.1            // 或……
O       10.2
O       10.3
A       10.0
OLD
LD      10.2            // 或……
O       10.3
A       10.1
OLD
LD      10.2
A       10.3
OLD
LD      M1.1            // 或正在强制等待。
OLD
S       M1.0，1         // 则设置互锁状态标志 M1.0=1。
// 解除互锁

LD      10.4            // 如果所有点动开关都没有动作。
AN      10.0
AN      10.1
AN      10.2
AN      10.3
AN      M1.1
R       1.0，1          // 则解除互锁（M1.0=0）。

// 顺时针方向低速旋转使能

LDN     10.1            // 如果没有顺时针方向高速旋转命令（Ri—h）。
LD      10.4            // 没有停机命令（OFF）。
A       10.5            // 没有电动机电路断路器动作命令。
AN      Q0.2            // 逆时针低速旋转状态位未置位。
AN      Q0.3            // 逆时针高速旋转状态位未置位。
=       M2.0            // 则顺时针方向低速旋转使能（M2.0=1）。

// 顺时针方向高速旋转使能

LDN     10.0            // 如果没有顺时针方向低速旋转命令（Ri—l）。
LD      10.4            // 没有停机命令（OFF）。
A       10.5            // 没有电动机电路断路器动作命令。
AN      Q0.2            // 逆时针低速旋转状态位未置位。
AN      Q0.3            // 逆时针高速旋转状态位未置位。
=       M2.1            // 则顺时针方向高速旋转使能（M2.1=1）。
```

```
// 逆时针方向高速旋转使能
LDN     10.3        // 如果没有逆时针方向高速旋转命令（Le—h）。
LD      10.4        // 没有停机命令（OFF）。
A       10.5        // 没有电机电路断路器动作命令。
AN      Q0.0        // 顺时针低速旋转状态位未置位。
AN      Q0.1        // 逆时针高速旋转状态位未置位。
=       M2.2        // 则逆时针方向低速旋转使能（M2.2=1）。

// 逆时针方向高速旋转使能
LDN     10.2        // 如果没有逆时针方向低速旋转使能（Le—l）。
LD      10.4        // 没有停机命令（OFF）。
A       10.5        // 没有电机电路断路器动作命令。
AN      Q0.0        // 顺时针低速旋转状态位未置位。
AN      Q0.1        // 顺时针高速旋转状态位未置位。
=       M2.3        // 则逆时针方向高速旋转使能。
// 顺时针方向低速旋转
LD      10.0        // 如果顺时针方向低速旋转命令（Ri—l）。
O       Q0.0        // 或顺时针方向低速旋转状态。
AN      M1.0        // 且没有互锁。
A       M2.0        // 顺时针方向低速旋转使能。
=       Q0.0        // 则置顺时针方向低速旋转输出 Q0.0=1。
// 顺时针方向高速旋转
LD      10.1        // 如果顺时针方向高速旋转命令（Ri—h）。
O       Q0.1        // 或顺时针方向高速旋转状态。
AN      M1.0        // 且没有互锁。
A       M2.1        // 顺时针方向高速旋转。
=       Q0.1        // 则置顺时针方向高速旋转输出 Q0.1=1。
// 逆时针方向低速旋转
LD      10.2        // 如果逆时针方向低速旋转命令（Le—l）。
O       Q0.2        // 或逆时针低速旋转状态。
AN      M1.0        // 且没有互锁。
A       M2.2        // 逆时针方向低速旋转使能。
=       Q0.2        // 则置逆时针方向低速旋转输出 Q0.2=1。
// 逆时针方向高速旋转
LD      10.3        // 如果逆时针方向高速旋转命令（Le—h）。
O       Q0.3        // 或逆时针方向高速旋转状态。
AN      M1.0        // 且没有互锁。
A       M2.3        // 逆时针方向高速旋转使能。
=       Q0.3        // 则置逆时针方向高速旋转输出 Q0.3=1。

LD      Q0.4        // 关机过程的边沿检测。
EU                  // 上升沿。
```

```
S       M1.1，1          // 将关机过程的辅助存储器标志置位（M1.1=1。）
LD      M1.1
MOVW    500，VW20        // 装载重新起动前必须等待的时间值（500×10ms=5ms）。
TON     T33，VW20        // 起动重新起动要强制等待的定时器（T33）。
A       T33              //
R       M1.1，1          // 定时器溢出后将辅助存储器标志位复位（M1.1=0）。
MOVW    O，T33           // 清除强制等待定时器（T33）。
RET                      // 子程序 1 结束。
```

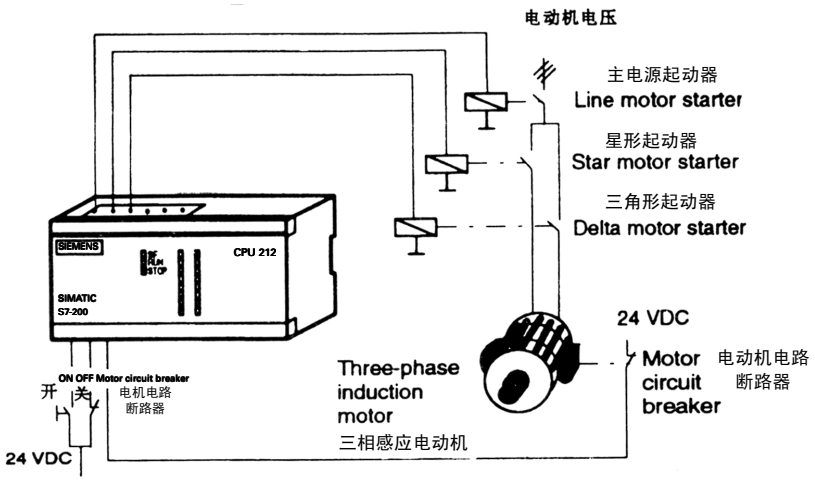
12 无反馈的电动机星形——三角形起动器

概述

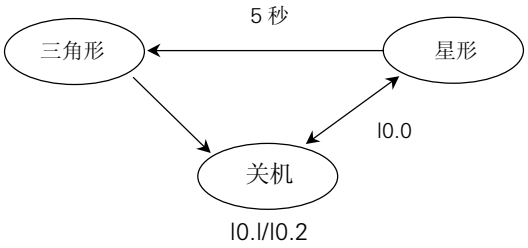
这个示例程序控制三相感应电动机的星形——三角形起动过程。当与输入点 10.0 相连的点动开关 ON（开机）接通时，电动机绕组星形连接运转。经过预置时间 5 秒钟后，电动机绕组切换为三角形连接。

当关机点动开关 OFF 或电动机电路断路器（分别与输入点 10.1 和 10.2 相连）动作时，电动机关机。当开机开关（ON）和停机开关（OFF）同时被按下时，电动机仍然处于停机状态。

例图



程序框图



程序和注释

在每个扫描周期的起始处程序都要检查是否必须将内部存储器标志位 M10.0 设置为互锁状态。当关机开关（10.1）和开机开关（10.0）同时动作时，M10.0 被设置成互锁状态。直到这两个开关都恢复为初始状态，互锁才解除。互锁的作用是防止误操作。

内部存储器标志位 M11.0 用于开机过程。当与输入点 10.0 相连的开机点动开关闭合，且主电源起动器尚未接通时，将 M11.0 置位。当电动机绕组正处于星形—三角形连接切换时，也就是主电源起动器（Q0.0）和星形起动器（Q0.1）同时接通时，也将 M11.0 置位。只有当电路断路器触点（10.2）和关机开关触点（10.1）都没有打开，且三角形起动器（Q0.2）没有工作时，M11. 0 才有可能被置位。

满足下述条件时输出 Q0.1 被置位，使星形起动器工作：用于开机过程的内部存储器标志位 M11.0 被置位；定时器 T37 没有溢出（预置时间为 5 秒）；且没有互锁标志（M10.0）。

用于开机过程的内部存储器标志位 M11.0 被置位时，只要没有互锁标志，限时定时器 T37 就开始计时（预置时间为 5 秒）。定时器 T37 的基准时间是 100ms，也就是说，当 T37 的预置值为 50 时，实际预置时间就是 5 秒。

控制主电源起动器的输出触点 Q0.0 闭合的条件是：接在输入点 I0.0 上的开机点动开关和控制星形起动器的输出点 Q0.1 都已经闭合，与输入点 I0.1 相连的人停机点动开关没有动作，且与输入点 I0.2 相连的电动机电路断路器没有断开，同时没有互锁标志。

当主电源起动器闭合，星形起动器切除后，控制三角形起动器的输出点 Q0.2 被置位。

该程序的长度为 40 个字。

```
// 标题：无反馈的电动机星形——三角形起动器

// I0.0    开机，点动开关（常开解点），ON。
// I0.1    停机，点动开关（常闭触点），OFF。
// I0.2    电动机电路断路器（常闭触点）。

// Q0.0    主电源起动器。
// Q0.1    星形连接起动器。
// Q0.2    三角形连接起动器。
// T37     5 秒定时器。

// 互锁和解除互锁

LDN     10.1            // 如果停机开关（OFF）动作。
A       10.0            // 且开机开关（ON）动作。
S       M10.0，1        // 则把内部存储器标志位 M10.0 置位（互锁）。
LD      10.1            // 如果停机开关（OFF）没有动作。
AN      10.0            // 且开机开关（ON）没有动作。
R       M10.0，1        // 则解除互锁（M10.0=0）。
```

 西门子 版权所有

```
// 开机过程
LD      I0.0            // 开机开关动作。
AN      Q0.0            // 主电源未接通。
LD      Q0.0            // 主电源接通。
A       Q0.1            // 电动机星形起动器。
OLD
LD      I0.2            // 电动机电路断路器未动作。
A       I0.1            // 停机开关未动作。
AN      Q0.2            // 电动机三角形起动器未动作。
ALD                     //
=       M11.0           // 内部存储器标志位。
// 接通电机星形起动器
LD      M11.0           // 内部存储器标志位。
AN      M10.0           // 没有互锁。
AN      T37             // 定时器未溢出。
=       Q0.1            // 星形起动器。
// 起动定时器
LDN     M10.0           // 未互锁。
A       M11.0           // 内部存储器标志位。
TON     M37，50         // 起动定时器（50×100ms=5s）。
// 接通主电源起动器
LD      10.1            // 停机开关没有动作。
A       10.2            // 电动机电路断路器没有动作。
AN      M10.0           // 未互锁。
LD      10.0            // 开机开关动作。
A       Q0.1            // 星形起动器。
O       Q0.0            // 主电源起动器。
ALD
=       Q0.0            // 主电源起动器。
// 接通电动机三角形起动器
LD      Q0.0            // 主电源起动器。
AN      Q0.1            // 电动机星形起动器。
=       Q0.2            // 电动机三角形起动器。
MEND                    // 主程序结束。
```

 西门子 版权所有

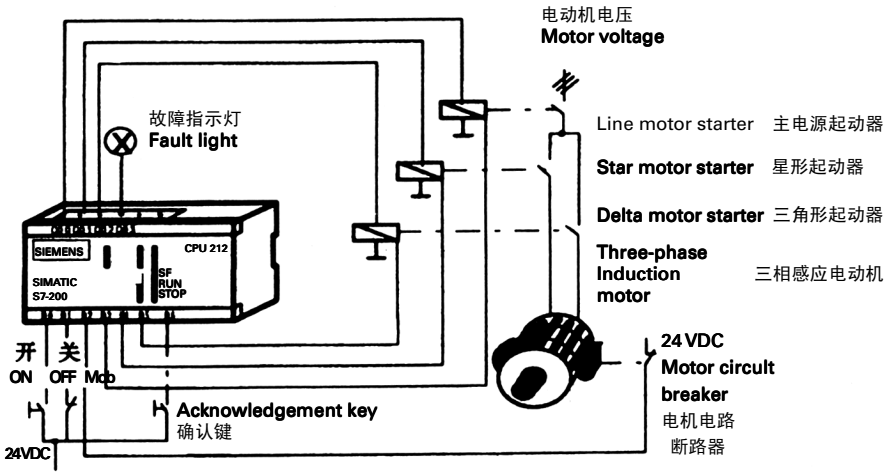
13 有反馈的电动机星形——三角形起动器

概述

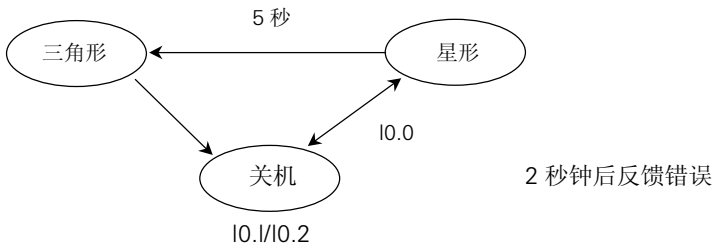
这个示例程序控制三相感应电动机的星形——三角形起动过程。它在应用示例 12 的基础上，增加了电动机起动器的实际电路状态的反馈。例如，一旦星形起动器出现故障，起动的反馈电路就有发现它。在 5 秒钟延时之后，SIMATIC S7-200 就不切换到三角形连接，这样就能避免可能造成的破坏。

与输入点 I0.0 相连的开机点动开关（ON）接通后，电动机绕组接成星形工作方式起动。与输出点 Q0.3 相连的信号灯指示各种可能出现的故障。

例图



程序框图



程序和注释

当输入点 I0.0 相连的开机开关（ON）动作后，电动机绕组接成星形工作方式起动。如果没有起动器故障信号，电动机绕组将在 5 秒钟后切换到三角形连接方式。故障信号由与输出点 Q0.3 相连的信号灯指示。当故障排除后，操作员按与输入点 I0.6 相连的确认键，即可消除故障信号。起动器反馈信号通过输入点 I0.3，I0.4 和 I0.5 引入。

当关机点动开关或电动机电路断路器（分别与输入点 I0.1 和 I0.2 连接）动作时，电动机关机。如果开机开关和关机开关同时动作，电动机仍然处于关机状态。

“互锁”、“解除互锁”、“开机过程”和“接通三角形起动器”都和应用示例 12 完全一样。因此这里不再详述。“接通星形起动器”、“起动定时器”和“接通主电源起动器”部分增加了一个条件：只有在无故障信号（Q0.3）出版时才动作。除此之外，为相关的起动器设置下述的存储器标志位：星形起动器（Q0.1），主电源起动器（Q0.0），以及起动定时器（T37）。

“起动器反馈”部分是新的。从原理上讲，反馈就是将输出信号和表示起动器实际状态的输入信号相比较。

输出信号的状态分别和下述反馈输入信号比较：主电源起动器的状态（I0.3），星形起动器的状态（I0.4），三角形起动器的状态（I0.5）。如果有差异就起动定时器 T38，T38 的预置时间为 2 秒。这段延迟时间对应起动器动作的最长时间。

如果 T38 溢出后，状态仍不同，故障指示输出点 Q0.3 被置位。这个故障信号可以用与输入点 I0.6 相连的反馈确认键复位。

该程序的长度为 70 个字。

```
// 标题：有反馈的电动机星形——三角形起动器
// I0.0    开机，点动开关，常开触点，ON。
// I0.1    关机，点动开关，常闭触点，OFF。
// I0.2    电动机电路断路器，常闭触点。
// I0.3    主电源起动器触点反馈。
// I0.4    星形起动器触点反馈。
// I0.5    三角形起动器触点反馈。
// I0.6    确认键。

// Q0.0    主电源起动器。
// Q0.1    星形起动器。
// Q0.2    三角形起动器。
// Q0.3    故障信号指示灯。

// T37     切换定时器（5 秒）。
// T38     故障信号定时器（2 秒）。
```

 西门子 版权所有

```
// 互锁和解除互锁
LDN     I0.1            // 关机开关动作。
A       I0.0            // 开机开关动作。
S       MI0.0，1        // 内部存储器标志位置位（互锁）。
LD      I0.1            // 关机开关没有动作。
AN      I0.0            // 开机开关没有动作。
R       m10.0，1        // 解除互锁。
// 开机过程
LD      I0.0            // 开机开关动作。
AN      Q0.0            // 主电源起动器未动作。
LD      Q0.0            // 主电源起动器。
A       Q0.1            // 星形起动器。
OLD
LD      I0.2            // 电动机电路断路器未动作。
A       I0.1            // 关机开关未动作。
AN      Q0.2            // 三角形起动器未动作。
ALD
=       M11.0           // 内部存储器标志位。
// 起动星形起动器
LD      M11.0           // 内部存储器标志位。
AN      M10.0           // 没有互锁。
AN      T37             // 切换定时器未溢出。
AN      Q0.3            // 无故障信号。
=       Q0.1            // 星形起动器。
// 起动切换定时器
LDN     M10.0           // 未互锁。
A       M11.0           // 内部存储器标志位。
AN      Q0.3            // 无故障信号。
TON     T37，50         // 起动定时器 T37（50×100ms=5s）。
// 接通主电源起动器
LD      I0.1            // 关机开关未动作。
A       I0.2            // 电动机电路断路器未动作。
AN      M10.0           // 没有互锁。
AN      Q0.3            // 无故障信号。
LD      I0.0            // 开机开关动作。
A       Q0.1            // 星形起动器。
O       Q0.0            // 主电源起动器。
```

```
ALD
=       Q0.0                    // 主电源起动器。
// 起动电动机三角形起动器
LD      Q0.0                    // 主电源起动器。
AN      Q0.1                    // 星形起动器。
=       Q0.2                    // 三角形起动器。
// 起动器反馈
LD      Q0.0                    // 主电源起动器。
AN      I0.3                    // 无主电源起动器反馈信号。
LDN     Q0.0                    // 主电源起动器未接通。
A       I0.3                    // 有主电源起动器反馈信号。
OLD
LD      Q0.1                    // 星形起动器。
AN      I0.4                    // 无星形起动器反馈信号。
OLD
LDN     Q0.1                    // 星形起动器。
A       I0.4                    // 有星形起动器反馈信号。
OLD                             //
LD      Q0.2                    // 三角形起动器。
AN      I0.5                    // 无三角形起动器反馈信号。
OLD
LDN     Q0.2                    // 三角形起动器未接通。
A       I0.5                    // 有三角形起动器反馈信号。
OLD
TON     T38，20                 // 故障信号的延迟时间（2s）。
LD      T38
S       Q0.3，1                 // 故障信号。
LD      I0.6                    // 确认键。
R       I0.3，1
MEND                            // 主程序结束。
```

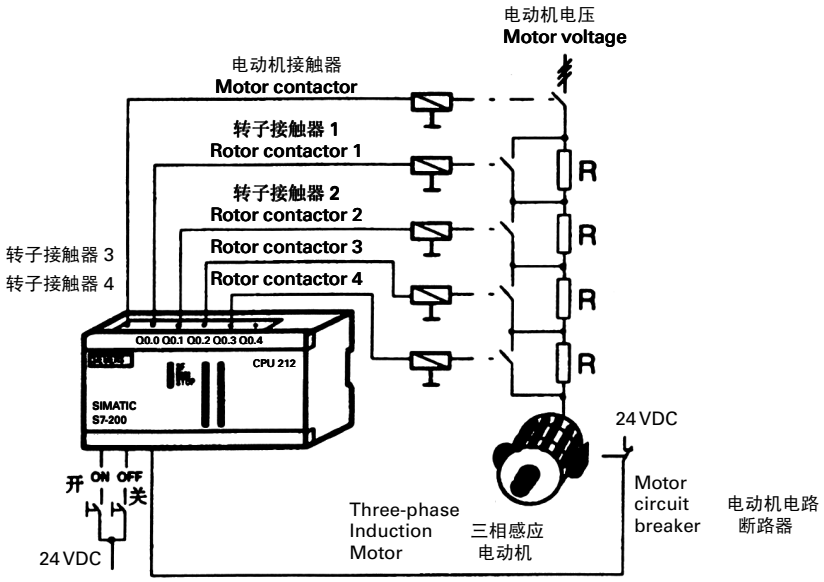
14 线绕转子

概述

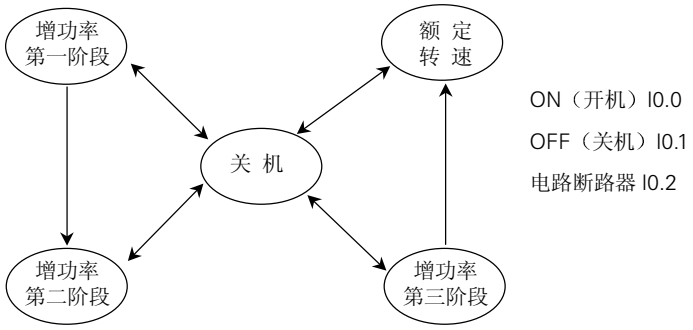
这个示例程序说明了 4 级线绕转子三相异步电动机的自动起动过程。电动机起动时转子为满电阻。经过一定时间后，第一个转子触点闭合并短接部分转子电阻。又经过一定时间后，后续触点逐步受到控制，而转子电阻每次都减小直至其完全短接，最终电动机以额定转速运行。

按接在输入端 I0.0 的点动开关 ON，即可开始平稳地启动电机。再按接在输入端 I0.1 的点动开关 OFF，即可停止电机。电机电路断路器接在输入端 I0.2，当电机过载时输入端 I0.2 打开，电机停止。

例图



程序框图



程序和注释

下述两种情况之一，可将中间结果内存标志位 M2.0 置位：一种是按接在输入端 I0.0 的点动开关 ON，并且 4 个转子接触器都未被激活；另一种是电机接触器已动作（Q0.0=1），这是为了锁定起动。这个中间结果内存标志位用来设置电机接触器内存标志位 Q0.0（运行电机），另外还必须同时满足 3 个条件：OFF 点动开关未动作，电机电路断路器未动作，无互锁。当 ON 和 OFF 点动开关同时动作时，将互锁内存标志位 M10.0 置位，直到这两个点动开关重新回到初始位置，才能将互锁内存标复位。

当控制电机接触器的输出 Q0.0 被置位后，第一个定时器 T37 开始计时，2 秒钟后，控制第一个转子接触器的输出 Q0.1 被置位。然后第二个定时器 T38 起动，又过 2 秒钟后，控制第二个转子接触器的输出 Q0.2 被置位。T39 和 T40 重复上述步骤，它们靠设置相应的输出 Q0.3 和 Q0.4 来分别起动转子接触器 3 和 4。这样，电机最终按额定转速旋转。当输入 I0.1 和 I0.2 不再有任何电压时，也就是 OFF 点动开关打开或电机电路断路器打开时，电机关闭。

本程序长度为 77 个字。

```
// 标题：线绕转子
// I0.0     ON 点动开关，常开触点，起动电机
// I0.1     OFF 点动开关，常闭触点，停止电机
// I0.2     电机电路断路器，常闭触点
// Q0.0     电机接触器
// Q0.1     转子接触器 1
// Q0.2     转子接触器 2
// Q0.3     转子接触器 3
// Q0.4     转子接触器 4

// T37      定时器，第一阶段
// T38      定时器，第二阶段
// T39      定时器，第三阶段
// T40      定时器，第四阶段

// ＊＊＊＊＊＊＊＊＊＊＊＊ 互锁和解除互锁 ＊＊＊＊＊＊＊＊＊＊＊＊

LDN      10.1                  // OFF 开关动作
A        10.0                  // ON 开关动作
S        M10.0，1              // 互锁（M10.0=1）
LD       10.1                  // OFF 开关不动作
AN       10.0                  // ON 开关不动作
R        M10.0，1              // 解除互锁(M10.0=0)
```

```
// ＊＊＊＊＊＊＊＊＊＊＊＊ 电机起动 ＊＊＊＊＊＊＊＊＊＊＊＊
LD      I0.0                    // ON 开关动作
AN      Q0.1                    // 转子接触器 1 未动作
AN      Q0.2                    // 转子接触器 2 未动作
AN      Q0.3                    // 转子接触器 3 未动作
AN      Q0.4                    // 转子接触器 4 未动作
LD      Q0.0
OLD
=       M2.0                    // 中间结果内存标志
LD      M2.0                    // 中间结果内存标志
A       I0.1                    // OFF 开关未动作
A       I0.2                    // 电机电路断路器未动作
AN      M10.0                   // 无互锁
=       Q0.0                    // 电机运行（起动电机接触器）
//＊＊＊＊＊＊＊＊＊＊＊＊ 起动定时器 T37 ＊＊＊＊＊＊＊＊＊＊＊＊
LD      Q0.0                    // 电机运行
AN      Q0.1                    // 转子接触器 1 未动作
AN      Q0.2                    // 转子接触器 2 未动作
AN      Q0.3                    // 转子接触器 3 未动作
AN      Q0.4                    // 转子接触器 4 未动作
TON     T37，20                 // 起动 T37（20×100ms=2s）
//＊＊＊＊＊＊＊＊＊＊＊＊ 起动转子接触器 1 ＊＊＊＊＊＊＊＊＊＊＊
LD      T37                     //T37 溢出
O       Q0.1                    // 锁定转子接触器 1
LD      Q0.0                    // 电机运行
AN      Q0.2                    // 转子接触器 2 未动作
AN      Q0.3                    // 转子接触器 3 未动作
AN      Q0.4                    // 转子接触器 4 未动作
ALD
=       Q0.1                    // 起动转子接触器 1
//＊＊＊＊＊＊＊＊＊＊＊＊ 起动定时器 T38 ＊＊＊＊＊＊＊＊＊＊＊
LD      Q0.0                    // 电机运行
AN      Q0.2                    // 转子接触器 2 未动作
AN      Q0.3                    // 转子接触器 3 未动作
AN      Q0.4                    // 转子接触器 4 未动作
A       Q0.1                    // 转子接触器 1 已动作
TON     T38，20                 // 起动 T38（20×100ms=2s）
//＊＊＊＊＊＊＊＊＊＊＊＊ 起动转子接触器 2 ＊＊＊＊＊＊＊＊＊＊＊
```

```
LD      T38                 //T38 溢出
O       Q0.2                // 锁定转子接触器 2
LD      Q0.0                // 电机运行
AN      Q0.3                // 转子接触器 3 未动作
AN      Q0.4                // 转子接触器 4 未动作
ALD
=       Q0.2                // 起动转子接触器 2

//＊＊＊＊＊＊＊＊＊＊＊ 起动定时器 T39＊＊＊＊＊＊＊＊＊＊＊

LD      Q0.0                // 电机运行
A       Q0.2                // 转子接触器 2 已动作
AN      Q0.3                // 转子接触器 3 未动作
AN      Q0.4                // 转子接触器 4 未动作
TON     T39，20             // 起动 T39（20×100ms=2s）

//＊＊＊＊＊＊＊＊＊＊＊ 起动转子接触器 3＊＊＊＊＊＊＊＊＊＊＊

LD      T39                 // T39 溢出
O       Q0.3                // 锁定转子接触器 3
LD      Q0.0                // 电机运行
AN      Q0.4                // 转子接触器 4 未动作
ALD
=       Q0.3                // 起动转子接触器 3

//＊＊＊＊＊＊＊＊＊＊＊ 起动定时器 T40＊＊＊＊＊＊＊＊＊＊＊

LD      Q0.0                // 电机运行
A       Q0.3                // 转子接触器 3 已动作
AN      Q0.4                // 转子接触器 4 未动作
TON     T40，20             // 起动 T40（20+100ms=2s）

//＊＊＊＊＊＊＊＊＊＊＊ 起动转子接触器 4＊＊＊＊＊＊＊＊＊＊＊

LD      T40                 // T40 溢出
O       Q0.4                // 锁定转子接触器 4
LD      Q0.0                // 电机运行
ALD
=       Q0.4                // 起动转子接触器 4
MEND                        // 主程序结束
```

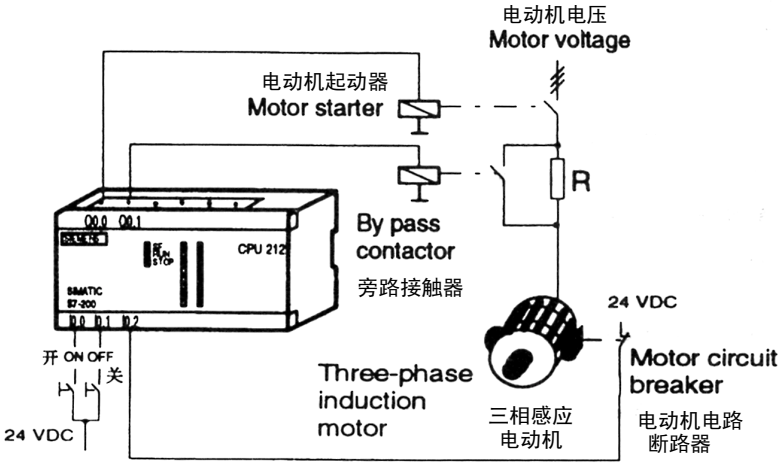
15 定子电阻起动电路

概述

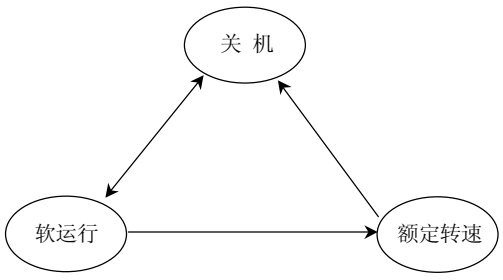
这个示例程序说明了带短路软起动开关的鼠笼转子的三相感应电动机的自动起动过程。通过这种短路软起动，电动机减速起动，并在一定时间段后达到额定转速。

通过按接在输入端 I0.0 的点动开关 ON 来实现电机软起动。如果按接在输入端 I0.1 的点动开关，则停止电机。电机电路断路器接在输入端 I0.2，当电机过载时电机电路断路器打开，电机停止。

例图



程序框图



ON（开机）I0.0

OFF（关机）I0.1

电动机电路断路器 I0.2

程序和注释

如果接在输入端 I0.0 的 ON 点动开关（常开触点）和接在输入端 I0.1 的 OFF 点动开关（常闭触点）同时动作，则设置内存标志位 M1.0 以互锁。直至两个点动开关又回到初始状态，才取消互锁。

接在输出端 Q0.0 的电机起动器动作的条件（与逻辑）如下：按下 ON 点动开关，无互锁（M1.0），电机电路断路器（I0.2）常闭触点，未动作，OFF 点动开关（I0.1）未动作。另外，再通过对 Q0.0 作或逻辑运算完成起动锁定。现在，电机以减速起动，因为起动电阻还未被短接。

如果电机已起动（Q0.0），并且用于旁路接触器的输出 Q0.1 还未被置位，那么计时器 T37 开始计时。在设定的 5 秒钟后，如果电机仍处于起动状态（Q0.0），则起动接在输出端 Q0.1 的旁路接触器。另外再通过对 Q0.1 作或逻辑运算完成旁路锁定。

本程序长度为 28 个字。

```
// 标题：KUSA 开关
// I0.0     ON 点动开关，常开触点，起动电机
// I0.1     OFF 点动开关，常闭触点，停止电机
// I0.2     电机电路断路器，常闭触点
// Q0.0     电机起动器
// Q0.1     旁路接触器
// T37      用于起动计时的定时器（5 秒）
//＊＊＊＊＊＊＊＊＊＊＊＊ 互锁 ＊＊＊＊＊＊＊＊＊＊＊
LDN      I0.1                  // OFF 点动开关动作
A        I0.0                  // ON 点动开关动作
S        M1.0，1               // 互锁内存标志置位（M1.0=1）
LD       I0.1                  // OFF 点动开关未动作
AN       I0.0                  // ON 点动开关未动作
R        M1.0，1               // 互锁内存标志复位（M1.0=0）
//＊＊＊＊＊＊＊＊＊＊＊＊ 电机起动 ＊＊＊＊＊＊＊＊＊＊＊
LD       I0.0                  // ON  点动开关动作
O        Q0.0                  // 锁定电机起动器
LD       I0.1                  // OFF 点动开关未动作
A        I0.2                  // 电机电路断路器未动作
AN       M1.0                  // 互锁内存标志未置位
ALD
=        Q0.0                  // 电机起动器
//＊＊＊＊＊＊＊＊＊＊＊＊ 起动定时器 T37＊＊＊＊＊＊＊＊＊＊＊
LD       Q0.0                  // 电机运行
AN       Q0.1                  // 旁路接触器
TON      T37，50               // 起动 T38（50×100ms=5s）
//＊＊＊＊＊＊＊＊＊＊＊＊ 起动旁路接触器 ＊＊＊＊＊＊＊＊＊＊＊
LD       T37                   //T37 溢出
O        Q0.1                  // 锁定旁路接触器
LD       Q0.0                  // 电机运行
ALD
=        Q0.1                  // 旁路接触器
MEND                           // 主程序结束
```

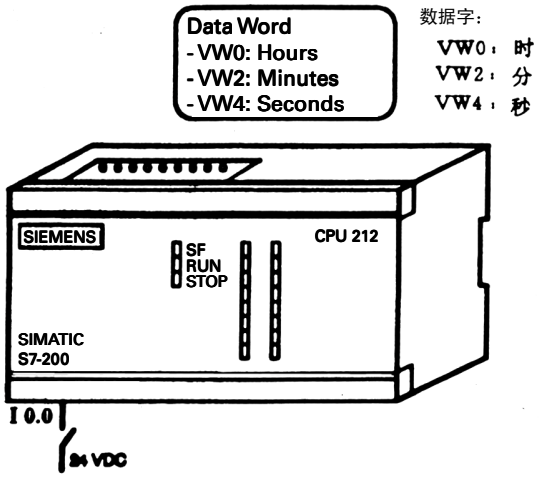
16 怎样追踪一台设备运行了多长时间

概述

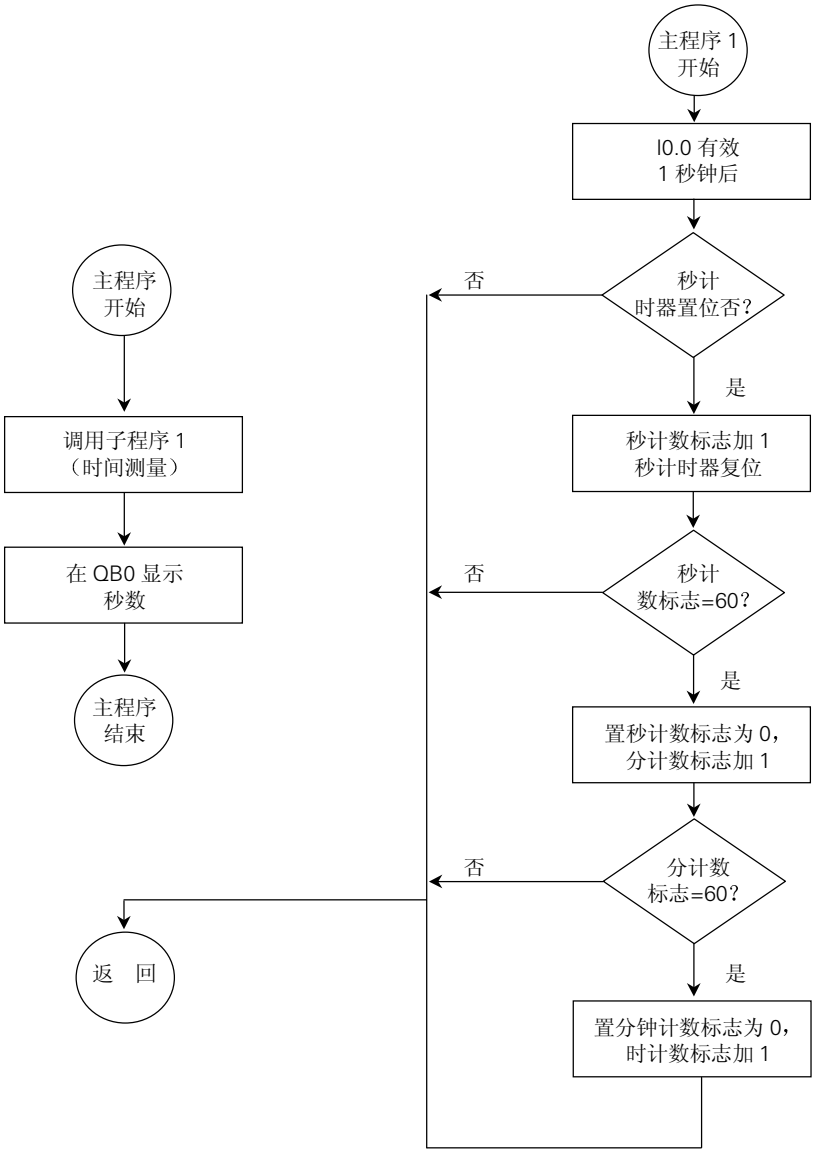
本例程序的目的是记录一台设备（制动器、开关等）运行的时间，以下前提必须满足：当设备运行时，必须给输入 I0.0 提供 24V 信号；当设备不工作时不提供电压。

当提供输入信号时，开始测量时间。如果没有输入信号，那么就中断时间的测量，直到重新提供输入信号为止。测量到的小时数存在字 VWD 中，分钟数存在字 VW2 中，秒数存在 VW4 中。

例图



程序框图


主程序 1
开始
I0.0 有效
1 秒钟后
秒计
时器置位否?
否
是
秒计数标志加 1
秒计时器复位
秒计
数标志=60?
否
是
置秒计数标志为 0，
分计数标志加 1
分计数
标志=60?
否
是
置分钟计数标志为 0，
时计数标志加 1
主程序
开始
调用子程序 1
（时间测量）
在 QB0 显示
秒数
主程序
结束
返 回

程序和注释

程序第一个扫描周期调用子程序 1。在子程序 1 中，设定计时器 T5 为 1 秒的运行时间，当达到 1 秒时，计时器位“T5”被置 1，同时，秒计数标志 VW4 增加 1，并将计时器位“T5”复位。因此，计时器能在下一周期立即重新启动。

当秒计数标志达到 60 时，分钟计数标志 VW2 增加 1，秒计数标志 VW4 被置为 0。

当分钟计数标志达到 60 时，小时计数标志 VW0 增加 1，分钟计数标志被置为 0。子程序结束。

主程序结束的最后一行用二进制来显示当前的秒数，用输出端的 LED 显示。

本程序长度为 35 个字。

```
// 标题：计时器

//  ＊＊＊＊＊＊＊＊＊＊＊ 主程序 ＊＊＊＊＊＊＊＊＊＊＊

// 主程序
LD      SM0.0                   // SM0.0 总是 1
CALL1                           // 调用测量子程序 1
MOVB    VB5，QB0                // 在输出 QB0 的 LED 上显示秒数
MEND                            // 主程序结束

//  ＊＊＊＊＊＊＊＊＊＊＊ 子程序 1＊＊＊＊＊＊＊＊＊＊＊

SBR1                            // 测量子程序开始。
LD      I0.0                    // 测量输入 I0.0 的运行时间。如果 I0.0=1，则
TONR    T5，10                  // 设定计时器时间为 1s（100ms*10=1s）。
LD      T5                      // 如果达到 1 秒钟，则
INCW    VW4                     // 秒计数标志加 1
R       T5，1                   // 秒计时器复位。

LDW>=VW4，60                    // 若计到 60 秒，则
MOVW    0，VW4                  // 秒计数标志置为 0，
INCW    VW2                     // 分计数标志加 1。

LDW>=VW2，60                    // 若计到 60 分，则
MOVW    0，VW2                  // 分计数标志置为 0，
INCW    VW0                     // 时计数标志加 1。

RET                             // 测量子程序结束
```

请参考 SIMATIC STEP 7 编程参考手册 4.1 节“时间指令”，为您提供了更多的关于计时器的信息。

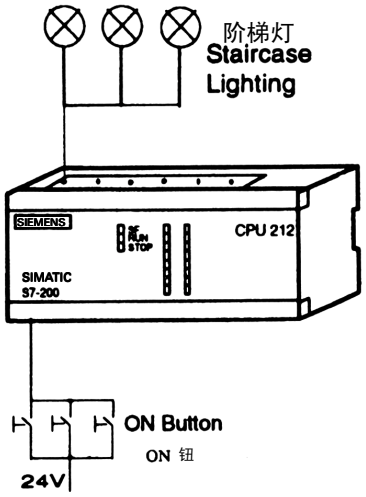
 西门子 版权所有

17 阶梯灯的定时点亮

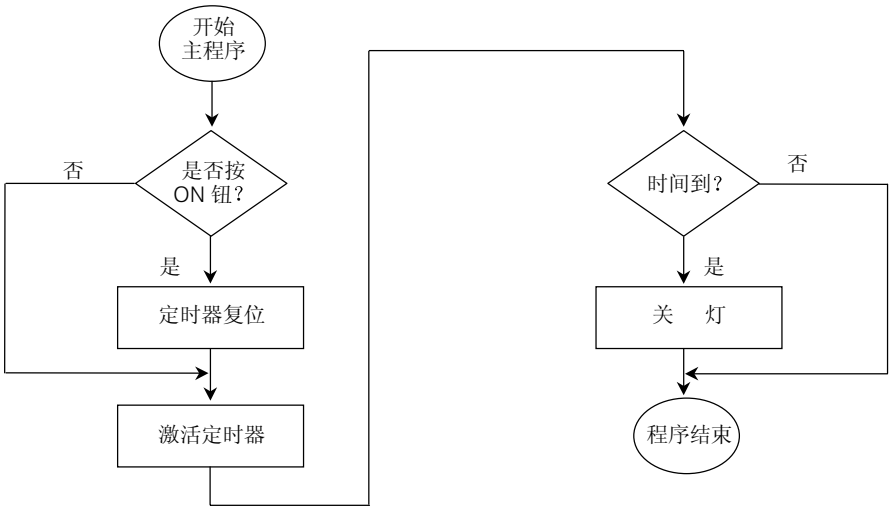
概述

本例程序是用来点亮阶梯灯。不同层上的 ON 按钮都被接到控制输入端 I0.0。当按下 I0.0 的 ON 按钮，则输出端 Q0.0 的灯发光 30 秒，如果在这段时间内又一次按 ON 按钮，则时间间隔又得从头开始。这样可确保在最后一次按 ON 按钮，在 30 秒内灯光不会熄灭。

例图



程序框图



程序和注释

如果按 ON 按钮，使输入 I0.0 的 ON 信号有效（I0.0=1），则定时器位 T37 复位（T37=0）。因而定时器 T37 从头开始计时。与此同时，输出 Q0.0 被置位（Q0.0=1，灯亮）。当计足 30 秒，定时器位 T37 置位（T37=1），那将又一次使输出 Q0.0 为 off（Q0.0=0，灯灭）。

本程序的长度为 17 个字。

```
// 题目：定时

LD      I0.0            // 若一个按钮被按下（I0.0=1），则
R       T37，1          // 定时器 T37 复位（T37=0）
S       Q0.0，1         // 开灯（Q0.0=1）
LD      SM0.0           // 特殊存储标记位 SM0.0 总是 1。
TON     T37，300        // 让定时器计时 30 秒（100ms×300=30s）

LD      T37             // 若 30 秒时间到，则
R       Q0.0，1         // 关灯（Q0.0=0）
MEND                    // 结束
```

18 步执行顺序（事件鼓定时器）

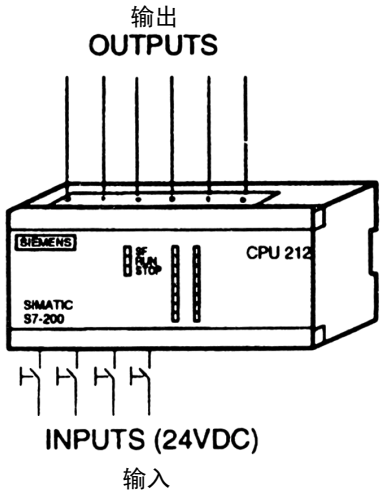
概述

本程序实现了一个按事件步顺序执行的例子。每步均包含一系列的动作，一步紧跟一步，并且只有所有前提条件均满足时，才能执行。如下所示

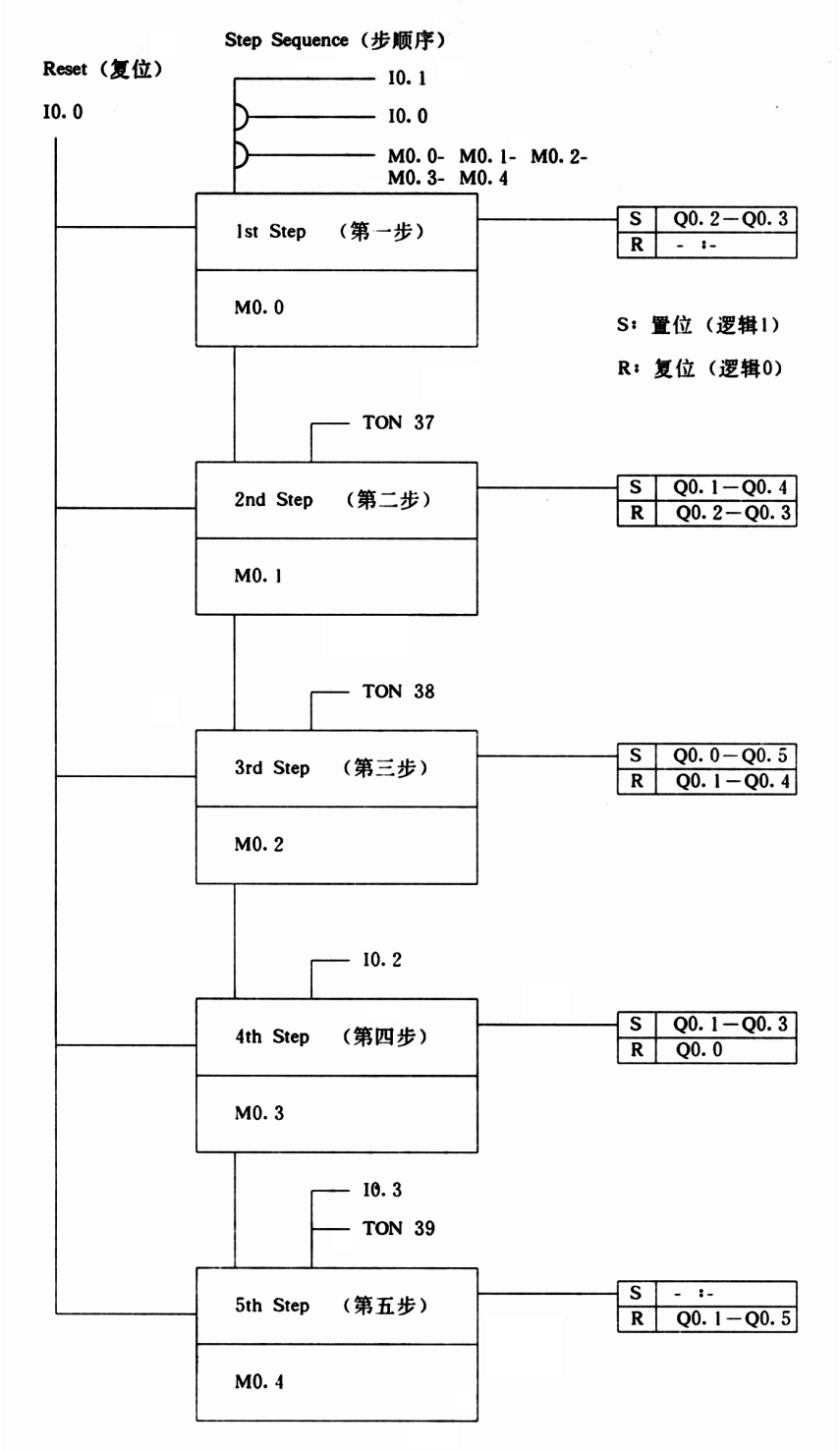
	前提条件	实际输出		
第 1 步：	I0.1 已被置为 1	Q0.2	Q0.3	
第 2 步：	间隔 5 秒（T37 定时器）	Q0.1	Q0.4	
第 3 步：	间隔 5 秒（T38 定时器）	Q0.0	Q0.5	
第 4 步：	I0.2 已被置为 1	Q0.1	Q0.3	Q0.5
第 5 步：	间隔 5 秒（T39 定时器） 并且 I0.3 已被置为 1	Q0.3		

复位步执行顺序（I0.0 已被置为 1） 无

例图



程序框图


Step Sequence（步顺序）
Reset（复位）
I0.0
I0.1
I0.0
M0.0- M0.1- M0.2-
M0.3- M0.4
1st Step （第一步）
M0.0
S Q0.2—Q0.3
R -
S：置位（逻辑1）
R：复位（逻辑0）
TON 37
2nd Step （第二步）
M0.1
S Q0.1—Q0.4
R Q0.2—Q0.3
TON 38
3rd Step （第三步）
M0.2
S Q0.0—Q0.5
R Q0.1—Q0.4
I0.2
4th Step （第四步）
M0.3
S Q0.1—Q0.3
R Q0.0
I0.3
TON 39
5th Step （第五步）
M0.4
S -
R Q0.1—Q0.5

程序和注释

本示例程序由五个能连续地执行的步组成。每步实质上就是对某些输出置位和复位。在某一步可执行以前，必须满足一些必要的前提条件。例如，已按下某一开关，或满足了要求的等待时间。另外，还可以随时按开关 I0.0 来复位步执行顺序。这些前提条件与输出结果已在程序框图中描述过了。

本程序长度为 116 个字。

```
// 标题：步执行顺序
//＊＊＊＊＊＊＊＊ 第 1 步 ＊＊＊＊＊＊＊＊

LD      I0.1                    // 起动条件，若输入 I0.1=1
AN      I0.0                    // 且未复位（I0.0=0）
AN      M0.0                    // 且无步执行
AN      M0.1                    //
AN      M0.2                    //
AN      M0.3                    //
AN      M0.4                    //
S       M0.0，1                 // 则将第 1 步标志位 M0.0 置 1。
LD      M0.0                    // 若为第 1 步，则
S       Q0.2，2                 // 设置输出（Q0.2=1，Q0.3=1）
TON     T37，50                 // 设与第 2 步之间的时间间隔为 5s（100ms×50）

//＊＊＊＊＊＊＊＊ 第 2 步 ＊＊＊＊＊＊＊＊

LD      T37                     // 若第 1 个定时器的时间间隔（5s）结束（T37=1）
A       M0.0                    // 且第 1 步已执行完（M0.0=1），则
R       M0.0，1                 // 将第 1 步标志位 M0.0 置 0，
S       M0.1，1                 // 将第 2 步标志位 M0.1 置 1。
LD      M0.1                    // 若为第 2 步，则
S       Q0.1，1                 // 设置输出，即 Q0.1=1。
S       Q0.4，1                 // 设置输出，即 Q0.4=1。
R       Q0.2，2                 // 将输出 Q0.2 和 Q0.3 置 0。
TON     T38，50                 // 设与第 3 步之间的时间间隔为 5 秒（100ms×50）

//＊＊＊＊＊＊＊＊ 第 3 步 ＊＊＊＊＊＊＊＊

LD      T38                     // 若第 2 个定时器时间间隔（5s）结束（T38=1）
A       M0.1                    // 且第 2 步已执行完（M0.1=1），则
R       M0.1，1                 // 将第 2 步标志位 M0.1 置 0，
S       M0.2，1                 // 将第 3 步标志位 M0.2 置 1。
LD      M0.2                    // 若为第 3 步，则
S       Q0.0，1                 // 设置输出，即 Q0.0=1
S       Q0.5，1                 // 设置输出，即 Q0.5=1
R       Q0.1，1                 // 将输出 Q0.1 和 Q0.4 置 0。
R       Q0.4，1                 //
```

```
//＊＊＊＊＊＊＊＊ 第 4 步 ＊＊＊＊＊＊＊＊

LD      I0.2                    // 起动条件，若输入 I0.2=1，
A       M0.2                    // 且第 3 步已执行完（M0.2=1,）则
R       M0.2，1                 // 将第 3 步标志位 M0.2 置 0，
S       M0.3，1                 // 将第 4 步标志位 M0.3 置 1，
LD      M0.3                    // 若为第 4 步，则
S       Q0.1，1                 // 设置输出，即 Q0.1=1
S       Q0.3，1                 // 设置输出，即 Q0.3=1
R       Q0.0，1                 // 将输出 Q0.0 置 0。
TON     T39，50                 // 设与第 5 步之间的时间间隔为 5s（100ms×50）

//＊＊＊＊＊＊＊＊ 第 5 步 ＊＊＊＊＊＊＊＊

LD      I0.3                    // 起动条件，若输入 I0.3=1，
A       T39                     // 且第三个定时器时间间隔（5s）结束（T39=1）
A       M0.3                    // 且第 4 步已执行完，则
R       M0.3，1                 // 将第 4 步标志位 M0.3 置 0。
S       M0.4，1                 // 将第 5 步标志位 M0.3 置 1。
LD      M0.4                    // 若为第 5 步，则
R       Q0.1，1                 // 将输出 Q0.1 和 Q0.5 置 0。
R       Q0.5，1                 //

//＊＊＊＊＊＊＊＊ 复位步执行顺序 ＊＊＊＊＊＊＊＊

LD      I0.0                    // 起动条件，若输入 I0.0=1，则
R       M0.0，5                 // 将所有 5 步的标志位 M0.0 至 M0.4 置 0
R       Q0.0，6                 // 将所有 6 个输出 Q0.0 至 Q0.5 置 0

MEND                            // 结束
```

请参考 SIMATIC STEP 7 编程参考手册 4.1 节“定时器指令”，为您提供了更多的关于定时器的信息。

19 S7—212 用自由通信口模式和并行打印机连接

概述

本例描述了 S7-212CPU 和外部设备（例如打印机）的连接方法。

该例中 SIMATIC PLC 自由通信口模式（Freeport Mode）向打印机发送信息。

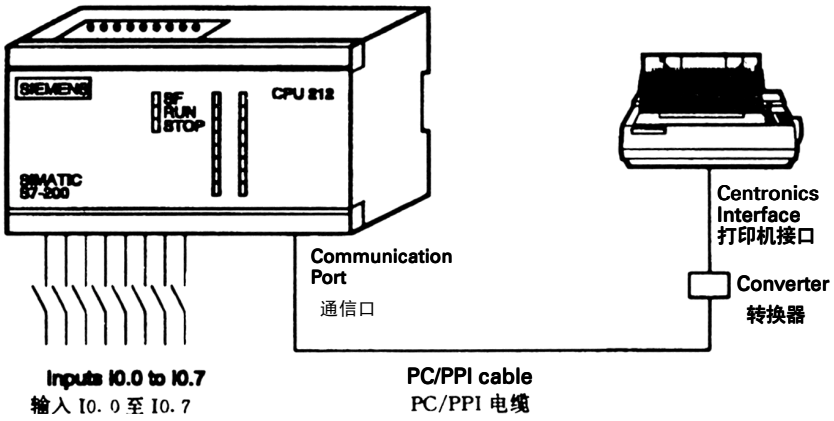
程序包含以下功能：

输入 I0.0 为 1 时，打印文字“SIMATIC S7—200”；

输入 I0.1 到 I0.7 为 1 时，打印句子“INPUT 0.×IS SET!”（其中×分别为 1，2，……，7）。

假定打印机用并行接口连接，并假定发送波特率为 9600 波特。

例图



硬件要求

为能正确地应用此例，你需要

1 台　Simatic S7-212 或 S7-214

1 条　PC/PPI　电缆

1 只　9 孔阴性插座到 25 针阳性插座的转换器

1 台串行到并行的转换器。

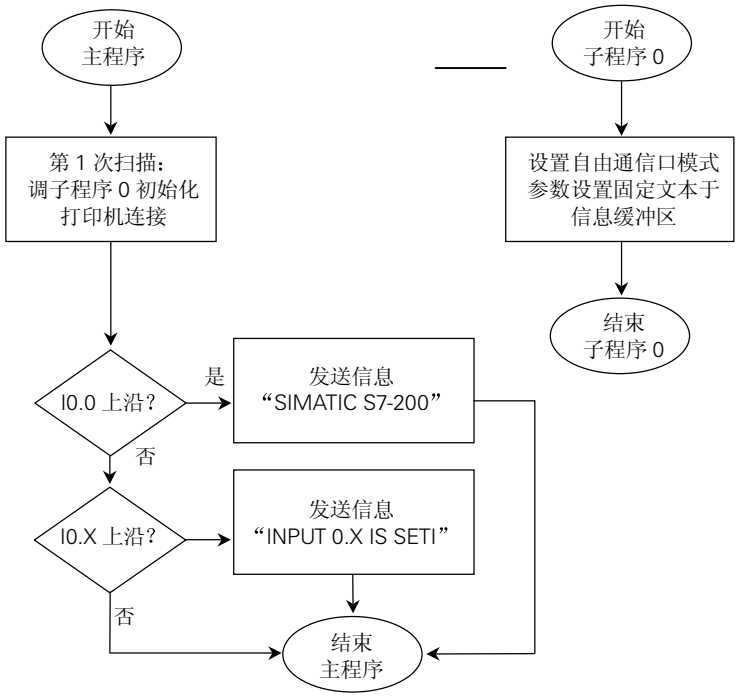
1 台线 2 和线 3 互换的空调制解调式的适配器（如果需要）

1 台并行打印机

可能会出现一个问题：

因为 SIMATIC S7-200 和打印机都作为数据通信设备（DCE），所以两台设备的数据传输方向有可能会相同，也就是说，两者的数据接收线接在了一起，或发送线也接在了一起（线 2 和线 3）。这个问题可以通过转换器的正确设置或使用合适的线路适配器（空调制解调式的适配器）来解决。

程序框图



程序和注解

此打印程序向并行打印机发送信息。

主程序检查 S7-200 模式开关，如果模式开关为 RUN 模式，则切换到自由通信口模式。

根据输入把相应的信息传送到打印机，主程序定义了这些内存变量。

以下的任务由子程序 0 完成：

子程序 0 包括设置自由通信口模式的参数和相应于不同输入的打印输出文本。

若输入 I0.0=1，则打印“SIMATIC S7-200!”。
若输入 I0.1=1，则打印“INPUT 0.1 IS SET!”。
若输入 I0.2=1，则打印“INPUT 0.2 IS SET!”。
⋮
若输入 I0.7=1，则打印“INPUT 0.7 IS SET!”。

程序结构如下：

MAIN（主程序）——初始化和输入请求
SBRO（子程序）——打印设置

本程序长度为 118 个字。

 西门子 版权所有

```
//标题：打印机

//＊＊＊＊＊＊＊＊ 主程序 ＊＊＊＊＊＊＊＊

LD      SM0.1                   // 第一次扫描标志：（SM0.1=1）。
CALL    0                       // 调用子程序 0
LD      SM0.7                   // 若在 TERM 模式，则设置 PPI（点到点接口）协议。
=       SM30.0                  // 若在 RUN 模式，则设置 Freeport(自由通信口)协议

LD      I0.0                    // 起动打印输入 I0.0
EU                              // 识别脉冲上升沿
XMT     VB80，0                 // 发送 ASCII 码，并打印（VB80 中存放所发送的 ASCII 码个数）。

LD      I0.0                    // 起动打印输入 I0.0
EU                              // 识别脉冲上升沿
MOVB    16#31，VB109            // 把 1 的 ASCII 码 31 存入 VB109
XMT     VB100，0                // 发送 ASCII 码，并打印（VB100 中存放所发送的 ASCII 码个数）

LD      I0.0                    // 起动打印输入 I0.2
EU                              // 识别脉冲上升沿
MOVB    16#32，VB109            // 把 2 的 ASCII 码#32 存入 VB109
XMT     VB100，0                // 发送

LD      I0.3                    // 起动打印输入 I0.3

EU
MOVB    16#33，VB109            // 把 3 的 ASCII 码 33 存入 VB109。
XMT     VB100，0                //
LD      I0.4                    // 起动打印输入 I0.4
EU
MOVB    16#34，VB109            // 把 4 的 ASCII 码 34 存入 VB109。
XMT     VB100，0                //

LD      I0.5                    // 起动打印输入 I0.5。
EU
MOVB    16#35，VB109            // 把 5 的 ASCII 码 35 存入 VB109。
XMT     VB100，0                //

LD      I0.6                    // 起动打印输入 I0.6。
EU
MOVB    16#36，VB109            // 把 7 的 ASCII 码 37 存入 VB109。
XMT     VB100，0                //

LD      I0.7                    // 起动打印输入 I0.7。
EU
MOVB    16#37，VB109            // 把 7 的 ASCII 码 37 存入 VB109。
XMT     VB100，0                //

MEND                            // 主程序结束。
```

```
//＊＊＊＊＊＊＊＊ 子程序 0 ＊＊＊＊＊＊＊＊

// 子程序 0

SBR     0                   // 设置打印信息。
MOVB    +9，SMB30           // 9600 波特，无奇偶校验，每字符 8 位

MOVB    +16，VB80           // 信息长度为 16 个 ASCII 码字符：SIMATIC S7-200
MOVW    16#5349，VW81       // 字符 SI
MOVW    16#4D41，VW83       // 字符 MA
MOVW    16#5449，VW85       // 字符 TI
MOVW    16#4320，VW87       // 字符 C␣
MOVW    16#5337，VW89       // 字符 S7
MOVW    16#2D32，VW91       // 字符-2
MOVW    16#3030，VW93       // 字符 00
MOVW    16#0D0A，VW95       //
MOVB    +20，VB100          // 信息长度为 20 个 ASCII 码字符：INPUT 0.x IS SET!

MOVW    16#494E，VW101      // 字符 IN
MOVW    16#5055，VW103      // 字符 PU
MOVW    16#420，VW105       // 字符 T␣ 。
MOVW    16#302E，VW107      // 字符 0。
MOVB    16#20，VW110        // 由主程序装裁 VB109（十六进制 31，32，……37）。
MOVW    16#4953，VW111      // 字符 IS。
MOVW    16#2053，VW113      // 字符␣ S。
MOVW    16#4554，VW115      //
MOVW    16#2021，VW117      // 字符 ET。
MOVW    16#0D0A，VW119      // 字符␣ !。

RET                         子程序 0 结束。
```

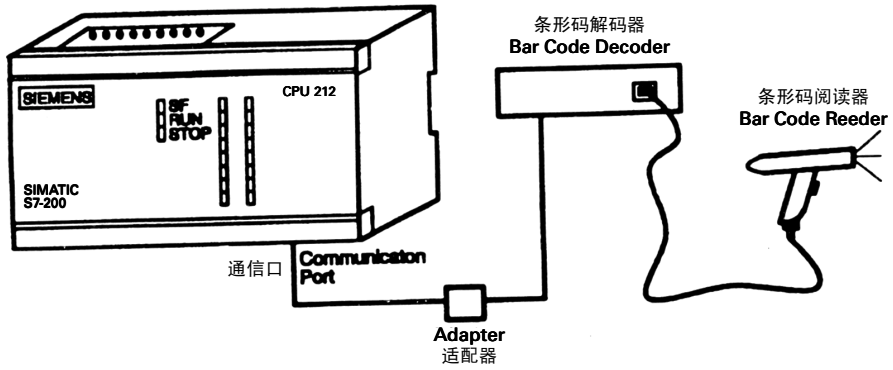
请参考 SIMATIC STEP 7 编程参考手册 2.6 节“特殊存储位”和 6.4 节“发送指令”，为您提供了更多的关于自由通信口模式和发送方面的信息。

20 通过自由通信口模式接受条形码阅读器的信息

概述

本例说明如何将 SIMATIC S7-212 或 S7-214 与条形码阅读器配合使用。
读入条形码的信息并经解码器翻译后，再通过自由通信口模式（Freeport Mode）把信息传入 SIMATIC。在 S7-212 或 214 的内存中有两个缓冲区，用来存储条形码信息，这两个缓冲区轮流地存储每次新读入的条形码。
通常这些数据可供程序调用。但本例中仅仅将信息存入接收缓冲区，可以用 S7-200 程序包来查看。

例图



硬件要求

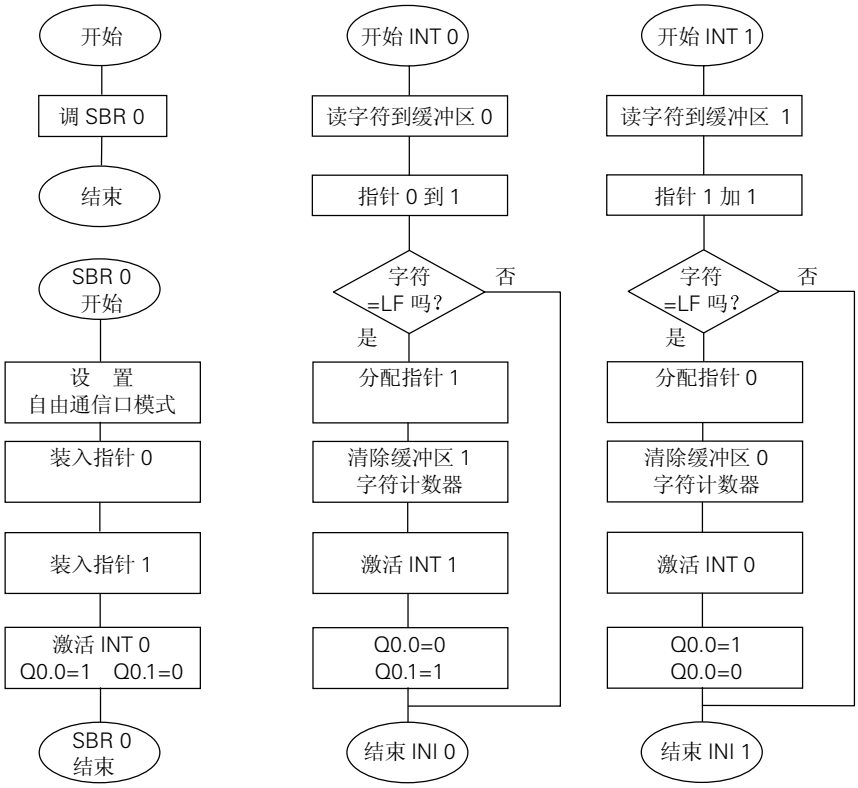
为能正常进应用此例，你需要：

1 台　SIMATIC S7-214 或 S7-212
1 条　PC/PPI　电缆
1 台　合适的适配器　（依据条形码解码器的接口类型，如 9 针阳性转换到 25 针阴性的插座，线 2 和线 3 互换的空调制解调器）
1 台条形码阅读器
1 台条形码解码器　（有时读码器与解码器是合一的）

可能会出现一个问题：

因为 SIMATIC S7-200 和条形码阅读器都作为数据通信设备（DCE），所以两台设备的数据传输方向有可能会相同，也就是说，二者的数据接收线接在一起，发送线也接在了一起（线 2 和 3）。这个问题可以通过转换器的正确设置或使用合适的线路适配器（空调制解调式的适配器）来解决。

程序框图



程序和注释

该程序从条形码阅读器接收信息再存入两个缓冲区。

从条形码解码器传出的信息是 ASCII 码形式，所接收的条形码存在 SIMATIC 内存中。这些数据可被程序利用，但本例中仅仅将信息存入接收缓冲区，可以用 SIMATIC S7-200 程序包来查看。

程序结构：

MAIN（主程序）：被始化程序
SBRO（子程序 0）：接收条形码
INT0（中断程序 0）：缓冲区 0 接收
INT1（中断程序 1），缓冲区 1 接收

本程序长度为 73 个字。

```
标题：条形码
//＊＊＊＊＊＊＊＊ 主程序 ＊＊＊＊＊＊＊＊
// 主程序
// 主程序的基本任务是初始化协议模式。
// 若开关在 RUN 位置，则特殊存储标志位 SM0.7 被设置为 1，可以采用的通信口模式。准确的自由通
// 信口模式协议是通过特殊存储标志字节 SM30 来设定（参考 Step 7 编程参考手册 2.6 节）。若开关在
//TERM 位置，则 SM0.7 为 0，传输协议将是点到点接口协议（PPI），因此，条形码阅读器将不能向 PLC
// 发送信息。这是因为条形码阅读器不支持 PPI 协议。

LD      SM0.1                   // 第一次扫描标志位 SM0.1=1
CALL    0                       // 调子程序 0
LD      SM0.7                   // 若在 TERM 模式，则设置 PPI（点到点接口）协议。
=       SM30.0                  // 若在 RUN 模式，则设置 Freeport（自由通信口）协议。
MEND                            // 主程序结束
//＊＊＊＊＊＊＊＊ 子程序 0 ＊＊＊＊＊＊＊＊
// 子程序 0

// 若开关在 RUN 位置，则置成自由通信口模式，SIMATIC 从条形码解码器得到信息。选择了自由通信
// 口模式协议，并且定义了两个指针。缓冲区 0 首地址（VB100）装入指针
// VD50，缓冲区 1 首地址（VB200）装入指针 VD60。
// 用 VW54 和 VW64 作字符计数器。激活中断程序 0 并允许中断。
// 读入的条形码将存到缓冲区 0 或 1 中。
// 子程序 0
SBR     0                       // 准备接收条形码
MOVB    +4，SMB30               //9600 波特，无奇偶校验，每字符 8 位
MOVD    &VB100，VD50            // 指针指向缓冲区 0
MOVD    &VB200，VD60            // 指针指向缓冲区 1
MOVD    VD50，VD56              //VD56 也指向缓冲区 0
MOVW    +4，VW54                // 清除缓冲区 0 的字符计数器
ATCH    +0，8                   // 中断程序 0 处理缓冲区 0 的接收
MOVB    +1，QB0                 // 设 Q0.0 为 1，Q0.1 为 0
ENI                             // 允许中断
RET                             // 结束子程序 0

//＊＊＊＊＊＊＊＊ 中断程序 0 ＊＊＊＊＊＊＊＊

// 中断 0
// 若缓冲区 0 有效，则中断 0 中断程序，执行下面的程序：
// 指针（VD56 内容）加 1，指向缓冲区的下一个位置，并且字符计数器也加 1。
// 若字符是 LF，则转向缓冲区 1 接收。
// 允许接收中断 1，且设置输出 Q0.0 为 0、Q0.1 为 1。
```

```
// 中断程序 0
INT     0                   // 缓冲区 0 接收
MOVB    SMB2，*VD56         // 字符装入缓冲区 0
INCD    VD56                // 指针加 1，指向缓冲区的下一个位置。
INCW    VW54                // 字符计数器加 1
LDB=    SMB2，10            // 若字符是 LF，则
MOVD    VD60，VD66          // 使指针 VD66 指向缓冲区 1，
MOVW    +0，VW64            // 清除缓冲区 1 的字符计数器，
ATCH    +1，8               // 中断程序 1 处理缓冲区 1 的接收
MOVB    +2，QB0             // 设置 Q0.0 为 0，Q0.1 为 1。
RET1                        // 中断程序 0 结束

//＊＊＊＊＊＊＊＊ 中断程序 1 ＊＊＊＊＊＊＊＊

// 中断 1
// 若缓冲区 1 有效，则中断 1 中断程序，执行下面的程序：
// 指针（VD66 内容）加 1，指向缓冲区的下一个位置，并且字符计数器也加 1。
// 若字符是 LF，则转向缓冲区 0 接收，允许接收中断 0。
// 且设置输出 Q0.0 为 1、Q0.1 为 0。

// 中断程序 1

INT     1                   // 缓冲区 1 接收
MOVB    SMB2，*VD56         // 字符装入缓冲区 1
INCD    VD66                // 指针加 1，指向缓冲区的下一个位置。
INCW    VW64                // 字符计数器加 1
LDB=    SMB2，10            // 若字符是 LF，则
MOVD    VD50，VD56          // 使指针 VD56 指向缓冲区 0，
MOVW    +0，VW54            // 清除缓冲区 0 的字符计数器，
ATCH    +0，8               // 中断程序 0 处理缓冲区 0 的接收
MOVB    +2，QB0             // 设置 Q0.0 为 1，Q0.1 为 0。
RET1                        // 中断程序 1 结束
```

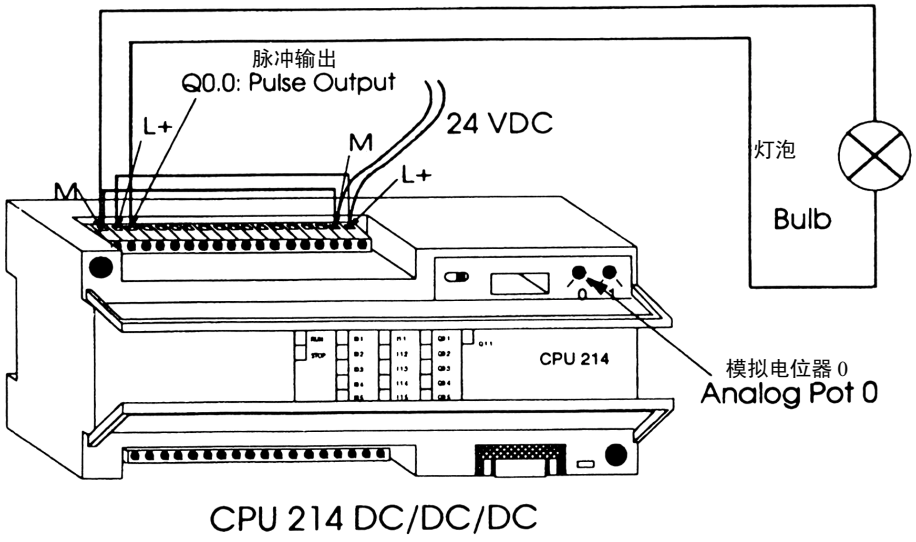
请参考 SIMATIC STEP 7 编程参考手册 2.6 节“特殊存储位”和 6.2 节“中断指令”，为您提供了更多的关于自由通信口模式和中断程序的信息。

21 灯炮亮度控制

概述

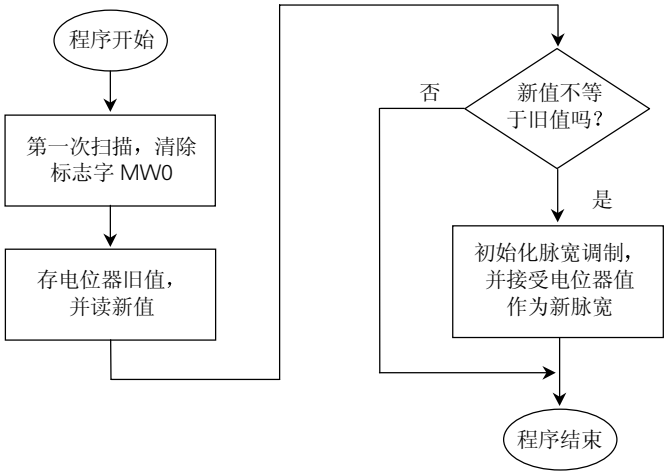
这个应和解释了一个使用 S7-200 的集成高速脉冲输出指令来控制灯炮（24V/1W）亮度的例子。模拟电位器 0 的设置值影响输出端 Q0.0 方波信号的脉冲宽度，也就是灯泡的亮度。调整电位器时需要一把（2.5mm）螺丝刀。

例图



CPU 214 DC/DC/DC

程序框图



程序和注释

在程序的每次扫描中，模拟电位器 0 的值，通过特殊存储字节 SMB28 被拷贝到内存字 MW0 的低字节 MB1。电位器的值除以 8 作为脉宽，脉宽和脉冲周期的比率大致决定了灯炮的亮度（相对于最大亮度）。除以 8 会带来这样一个额外的好处，即丢弃了 SMB28 所存值的 3 个最低有效位，从而使程序更稳定。如果电位器值变化了，那么将重新初始化输出端 Q0.0 的脉宽调制，借此电位器的新值将被变换成脉宽的毫秒值。

例：SMB28=80 （电位器 0 的值）
80/8=10
10/25（=脉宽/周期）=40％（电压时间比）=40％最大亮度

本程序的长度为 30 个字

```
// 标题：用电位器调灯泡亮度

LD      SM0.1               // 第一次扫描 SM0.1=1
MOVW    0，MW0              // 标志字 MW0 清零

LD      SM0.0               // 设置栈顶（SM0.0 总是 1）
MOVW    MW0，WM2            // 保存电位器的旧值

MOVB    SMB28，MB1          // 在 MW0 的低字节（MB1）保存模拟电位器新值
SRW     MW0，3              // 将电位器新值除 8
LDW=    MW0，MW2            // 判电位器新值（MW0）等于旧值（MW2）吗?
NOT                         // MW0≠MW2
MOVB    16#CB，SMW67        // 设置 PWM 的控制字节（激活 PWM，时基 1 毫秒，可更新脉宽
                               和周期）

MOVW    25，SMW68           // 设置脉冲周期为 25 毫秒。
MOVW    MW0，SMW70          // 根据电位器值设置脉宽。
PLS     0                   // 在输出端 Q0.0 输出脉冲

MEND                        // 主程序结束
```

请参考 SIMATIC STEP 7 编程参考手册 6.3 节“高速输出指令”，为您提供了更多的关于脉冲序列的信息；再参考 2.6 节“特殊存储位”和 2.12 节“CPU 214 的附加特性”，为您提供了更多的关于使用模拟电位器的详细数据。

 西门子 版权所有

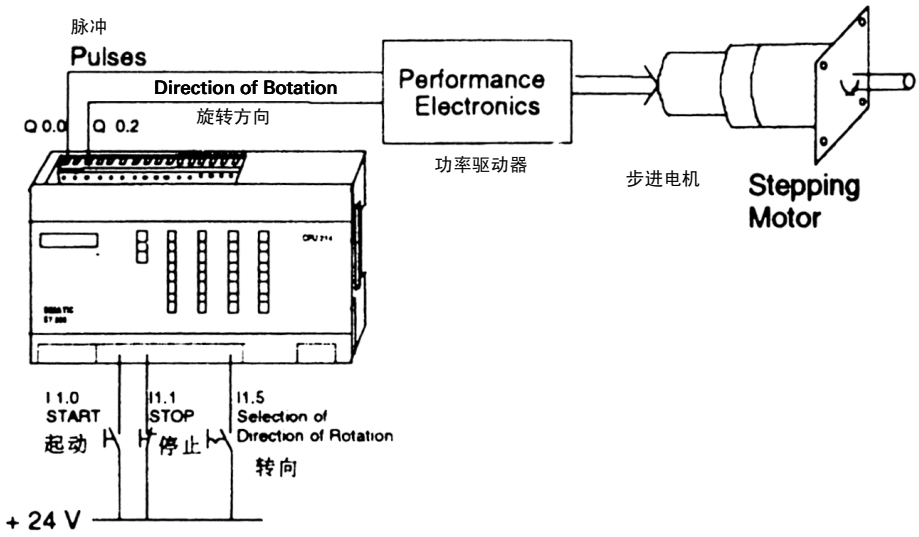
22 用集成脉冲输出触发步进电机驱动器

概述

CPU 214 有两个脉冲输出，可以用来产生控制步进电机驱动器的脉冲。功率驱动器将控制脉冲按照某种模式转换成步进电机线圈的电流，产生旋转磁场，使得转子只能按固定的步数（步角 a）来改变它的位置。连续的脉冲序列产生与其对应的同频率（同步机）步序列。如果控制频率足够高，步进电机的转动可看作一个连续的转动。

本例叙述用 Q0.0 的输出脉冲触发步进电机驱动器。当输入端 I1.0 发出“START”（起动）信号后，控制器将输出固定数目的方波脉冲，使步进电机按对应的步数转动。当输入端 I1.1 发出“STOP”（停止）信号后，步进电机停止转动。接在输入端 I1.5 的方向开关位置决定电机正转或反转。

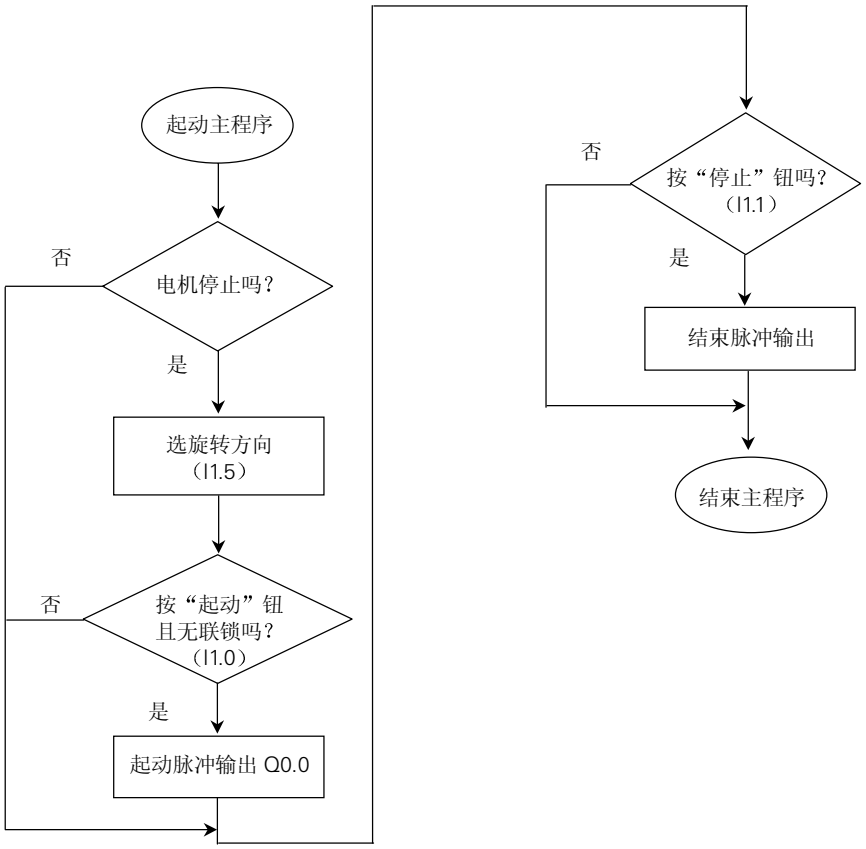
例图



硬件要求

数量	设备	制造厂/定货号
1	SIMATIC S7-200 CPU—214	Siemens/6ES7 214—1AC00—0XB0
1	PC/PPI 电缆	Siemens/6ES7 901—3BF00—0XA0
1	编程设备或 PC	
1	带有标准的功率驱动器和相关连接电缆的步进电机	
1	用于传输控制信号到功率驱动器的电缆	
1	开关	
2	按钮	

程序框图



程序和注释

一、初始化

在程序的第一个扫描周期（SM0.1=1），为两种脉冲输出功能（PTO 和 PTW）选择参数，本例从中选择了 PTO，并规定了脉冲周期和脉冲数。

二、选择转动方向

用接在输入端 I1.5 的开关来选择转动方向。如果 I1.5=1，将输出 Q0.2 置成高电位，那么电机逆时针转动。如果 I1.5=0，将输出 Q0.2 置成低电位，那么电机顺时针转动。为保护电机避免漏步，电机转动方向的改变只能在电机处于停止状态（M0.1=0）时进行。

三、起动电机

起动电机的 3 个条件如下：

1. 按“START”（起动）按钮，在输入端 I1.0 产生脉冲上升沿（从 0 升到 1）；
2. 无联锁，即联锁标志 M0.2=0；
3. 电机处于停止状态，即操作标志 M0.1=0。

 西门子 版权所有

如果同时具备上述 3 个条件，则将 M0.1 置位（M0.1=1），控制器执行 PLS0 指令，在输出端 Q0.0 输出脉冲，其它必须预先具备的条件，已经在首次扫描（SM0.1=1）设置，主要是脉冲输出功能的基本数据。例如，时基、周期和脉冲数。这些数据置于相应的属于 PTO/PWM 的特殊存储字 SMW68，SMW70 和 SMD72。

四、停止电位

停止电机的 2 个条件如下：

1. 按“STOP”（停止）按钮，在输入端 I1.1 产生脉冲上升沿（从 0 升到 1）；
2. 电机处于转状态，即操作标志 M0.1=1。

如果同时具备上述 2 个条件，则将标志 M0.1 复位（M0.1=0），并中断输出端 Q0.0 的脉冲输出。这与执行 PLS0 指令有关，它将脉宽调制（PWM）输出的脉冲宽度减为 0（所需的基本设置已在第一个扫描周期中定义了），因而输出信号被抑制。

在完整的脉冲序列输出后，中断程序 0 将标志 M0.1 复位（M0.1=0），从而使电机能够重新起动。为更清晰起见，这部分程序不包括在本例程序流程图中。

五、联锁

为保护人员和设备的安全，在按“STOP”（停止）按钮（I1.1）之后，必须规定驱动器联锁（或称阻塞），将联锁标志 M0.2 置位（M0.2=1），立即关断驱动器。只有在 M0.2 复位（M0.2=0）后，才能重新起动电机。当“STOP”按钮松开后，为防止电机的意外起动，只有在“START”按钮（I1.0）和“STOP”按钮（I1.1）都松开后，才能将 M0.2 复位（M0.2=0），如要再次起动电机，则必须再发出一个起动信号。

六、程序清单

本程序长度为 64 个字。

```
// 标题：驱动器功能测试
LD      SM0.1              // 仅首次扫描周期 SM0.1 置位（SM0.1=1）
MOVW    500，SMW68         // 输出脉冲周期为 500μs
MOVW    0，SMW70           // 脉宽为 0（脉宽调制）
MOVD    40000，SMD72       // 输出 40000 个脉冲
ATCH    0，19              // 把中断程序 0 分配给中断事件 19（PLS 0 脉冲输出结束）
ENI                        // 允许中断
// 设置转动方向
LDN     M0.1               // 若电机处于停止状态，
A       I1.5               // 且转向开关置于 1，
S       Q0.2，1            // 则逆时针转动（Q0.2=1）。
LDN     M0.1               // 若电机处于停止状态，
AN      I1.5               // 且转向开关置于 0，
R       Q0.2，1            // 则顺时针转动（Q0.2=0）。
```

```
// 联锁和解除联锁
LD      I1.1                    // 若按“STOP”（停止）按钮，
S       M0.2，1                 // 则联锁有效（M0.2=1）。
LDN     I1.0                    // 若“START”（起动）按钮松开，
AN      I1.1                    // 且“STOP”（停止）按钮松开，
R       M0.2，1                 // 则解除联锁（M0.2=0）。

// 起动电机
LD      I1.0                    // 若按“START”（起动）按钮，
EU                              // 上升沿，
AN      M0.2                    // 且无联锁，
AN      M0.1                    // 且电机停止，则

MOVB    16#85，SMB67            // 置脉冲输出功能的控制位，
PLS     0                       // 起动脉冲输出（Q0.0），
S       M0.1，1                 // 电机运行标志 M0.1 置位（M0.1=1）。

// 停止电机
LD      I1.1                    // 若按“STOP”（停止）按钮，
EU                              // 上升沿，
A       M0.1                    // 且电机正在转动，则
R       M0.1，1                 // 电机运行标志 M0.1 复位（M0.1=0）
MOVB    16#CB，SMB67            // 置脉冲输出功能的控制位，PWM 的脉宽为 0，
PLS     0                       // 输出端 Q0.0 无脉冲。
MEND                            // 主程序结束。

// * * * * * * * * * * * * * * * * * * * * * * * *
INT0                            // 中断程序 0。
R       M0.1，1                 // 电机运行标志 M0.1 复位（M0.1=0）
RETI                            // 中断程序 0 结束
```

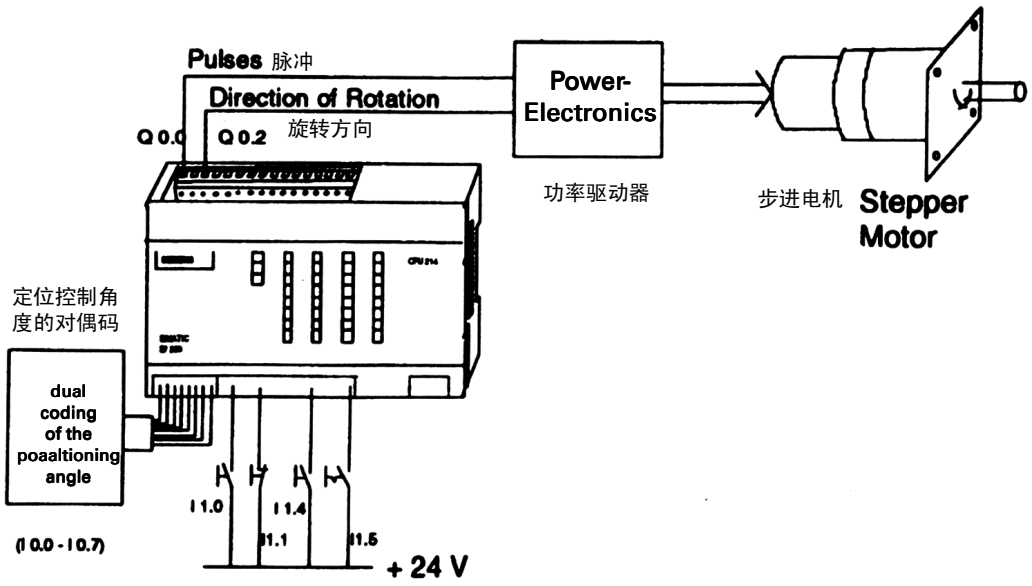
请参考 SIMATIC STEP 7 编程参考手册 6.3 节“高速输出指令”和 6.2 节“中断指令”，为您提供了更多的关于脉冲序列和中断程序的信息。

23 集成脉冲输出通过步进电机进行定位控制

概述

关于定位控制（Positioning），调节（Regulated）和控制（Controlled）操作之间存在一些区别。步进电机不需要连续的位置控制，而在控制操作中得到应用。在以下的程序例子中，借助于 CPU214 所产生的集成脉冲输出，通过步进电机来实现相对的位置控制。虽然这种类型的定位控制不需要参考点，本例还是初略地描述了确定参考点的简单步骤。因为实际上它总是相对一根轴确定一个固定的参考点，因此，用户借助于一个输入字节的对偶码（Duul coding）给 CPU 指定定位角度。用户程序根据该码计算出所需的定位步数，再由 CPU 输出相关个数的控制脉冲。

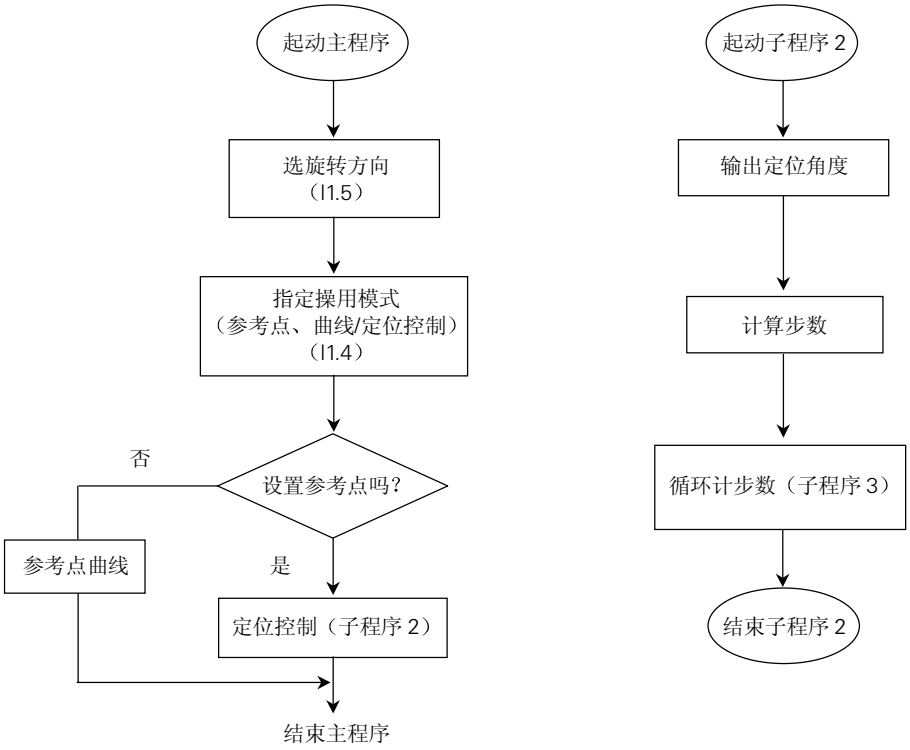
例图



硬件要求

数量	设备	制造厂/定货号
1	SIMATIC S7-200 CPU—214	Siemens/6ES7 214—1AC00—0XB0
1	PC/PPI 电缆	Siemens/6ES7 901—3BF00—0XA0
1	编程器或 PC	
1	带有标准的功率驱动器和相关连接电缆的步进电机	
1	用于传输控制信号到功率驱动器的电缆	
9	开关	
3	按钮	

程序框图



程序和注释

一、初始化

在程序的第一个扫描周期（SM0.1=1），初始化重要参数。选择旋转方向和解除联锁，类似于应用示例 22。

二、设置和取消参考点

如果还没有确定参考点，那么参考点曲线（Reference Point Curve）应从按“START”（起动）按钮（I1.0）开始。CPU 有可能输出最大数量的控制脉冲。在所需的参考点，按“设置/取消参考点”开关（I1.4）后，首先调用停止电机的子程序。然后，将参考点标志位 M0.3 置成 1，再把新的操作模式“定位控制激活”显示在输出端 Q1.0。

如果 I1.4 的开关已被激活，而且“定位控制”也被激活（M0.3=1），则切换到“参考点曲线”操作模式。在子程序 1 中，将 M0.3 置成 0，并取消“定位控制激活”的显示（Q1.0=0）。此外，控制还为输出最大数量的控制脉冲做准备。当两次激活 I1.4 开关，便在两个模式之间切换。如果此信号产生，同时电机在运转，那么电机就自动停止。

实际上，一个与驱动器连接的参考点开关将代替手动操作切换开关的使用，所以，参考点标志能解决模式切换。

三、定位控制

如果确定了一个参考点（M0.3=1），而且没有联锁（请参考应用示例 22），那么就执行相对的定位控制。在子程序 2 中，控制器从输入字节 IBO 读出对偶码方式的定位角度后，再存入字节 MB11。与此角度有关的脉冲数，根据下面的公式计算：

$$N = \frac{\varphi}{360^\circ} \times S$$

n=控制脉冲数

φ =旋转角度（以度为单位）

S=每转所需的步数

该示例程序所使用的步进电机采用半步操作方式（S=1000）。在子程序 3 中循环计算步数。如果说现在按“START”按钮（I1.0），CPU 将从输出端 Q0. 0 输出所计算的控制脉冲个数，而且电机将根据相应的步数来转动，并在内部将“电机转动”的标志位 M0.1 置成 1。

在完整的脉冲输出之后，执行中断程序 0（请参考应用示例 22），此程序将 M0.1 置成 0，以便能够再次起动电机。象应用示例 22 那样，为更清晰起见，这一步并没有包含有程序流程图中。

四、停止电机

类似于应用示例 22，按“STOP”（停止）按钮（I1.1），可在任何时候停止电机。执行子程序 0 中与此有关的指令。

本程序长度为 147 个字。

```
// 输入：    I0.0—I0.7      ——以度为单位的定位角（对偶码）
//           I1.0             “START”开关，起动电机
//           I1.1             “STOP”开关，停止电机
//           I1.4             “设置/取消参考点”开关
//           I1.5             选择旋转方向的开关
// 输出：    Q0.0           ——脉冲输出
//           Q0.2             旋转方向信号（Q0.2=1 左转，Q0.2=0 右转）
//           Q1.0             操作模式的显示
// 标志位：  M0.1           ——电机运转标志位
//           M0.2             联锁标志位
//           M0.3             参考点标志位
//           MD8，MD12        辅助标志位

// 标题：用脉冲输出进行定位控制

// 主程序
LD        SM0. 1         // 仅首次扫描周期 SM0.1 才为 1。
R         M0.0，128      // MD0 至 MD12 复位
ATCH      0，19          // 把中断程序 0 分配给中断事件 19（脉冲串终止）
ENI                      // 允许中断
```

```
// 脉冲输出功能的初始化
MOVW    500，SMW68          // 脉冲周期 T=500μs
MOVW    0，SMW70            // 脉冲宽度为 0（脉宽调制）
MOVD    429496700，SMD72    // 为参考点设定的最大脉冲数

// 设置逆时针旋转
LDN     M0.1                // 若电机停止
A       I1.5                // 且旋转方向开关=1，
S       Q0.2，1             // 则逆时针旋转（Q0.2=1）

// 设置顺时针旋转
LDN     M0.1                // 若电机停止
AN      I1.5                // 且旋转方向开关=0，
R       Q0.2，1             // 则顺时针旋转（Q0.2=0）

// 联锁
LD      I1.1                // 若按“STOP”（停止）按钮
S       M0.2，1             // 则激活联锁（M0.2=1）

// 解除联锁
LDN     I1.1                // 若“START”（起动）按钮松开
AN      I1.0                // 且“STOP”（停止）按钮松开
R       M0.2，1             //则解除联锁（M0.2=0）

// 确定操作模式（参考点定位控制）

LD      I1.4                // 若按“设置/取消参考点”按钮
EU                          // 上升沿
CALL    1                   // 则调用子程序 1

// 起动电机
LD      I1.0                // 若按“START”（起动）按钮
EU                          // 上升沿
AN      M0.1                // 且电机停止
AN      M0.2                // 且无联锁
AD>=    SMD72，1            // 且步数>=1，则

MOVB    16#85，SMB67        // 置脉冲输出功能（PTO）的控制位
PLS     0                   // 起动脉冲输出（Q0.0）
S       M0.1，1             // “电机运行”标志位置位（M0.1=1）

// 定位控制
LD      M0.3                // 若已激活“定位控制”操作模式
AN      M0.1                // 且电机停止
CALL2                       // 则调用子程序 2。
```

```
// 停止电机

LD          I1.1                  // 若按“STOP”（停止）按钮
EU                                // 上升沿
A           M0.1                  // 且电机运行，则
CALL0                             // 调用子程序 0
MEND                              // 主程序结束

//* * * * * * * * * * * * * * * * * * * * * * * *

//子程序 1

SBR 0                             // 子程序 0，“停止电机”
MOVB        16#CB，SMB67          // 激活脉宽调制
PLS         0                     // 停止输出脉冲到 Q0.0
R           M0.1，1               // “电机运行”标志位复位（M0.1=0）
RET                               // 子程序 0 结束

SBR1                              // 子程序 1，“确定操作模式”
LD          M0.1                  // 若电机运行
CALL 0                            // 则调用子程序 0，停止电机

//申请“参考点曲线”
LD          M0.3                  // 若已激活“定位控制”，则
R           M0.3，1               // 参考点标志位；复位（M0.3=0）
R           Q1.0，1               // 取消“定位控制激活”信息（Q1.0=0）
MOVD        429496700，SMD72      // 为新的“参考点曲线”设定最大脉冲数。
CRET                              //条件返回到主程序。

//申请“定位控制”
LDN         M0.3                  // 若未设置参考点（M0.3=0），则
S           M0.3，1               // 参考点标志位置位（M0.3=1）
S           Q1.0，1               // 输出“定位控制激活”信息（Q1.0=1）
RET                               // 子程序 1 结束

//* * * * * * * * * * * * * * * * * * * * * * * *
// 子程序 2

SBR2                              // 子程序 2，“定位控制”
MOVB        IB0，MB11             // 把定位角度从 IB0 拷到 MD8 的最低有效字节 MB11。
R           M8.0，24              // MB8 至 MB10 清零
DIV         9，MD8                // 角度/9=q1+r1
MOVW        MW8，MW14             // 把 r1 存入 MD12
MUL         25，MD8               // q1×25→MD8
MUL         25，MD12
DIV         9，MD12               // r1×25/ 9=q2+r2
```

```
CALL 3                          // 在子程序 3 中循环步数
MOVW    0，MW12                 // 删除 r2
+D      MD12，MD8               // 把步数写入 MD8
MOVD    MD8，SMD72              // 把步数传到 SMD72
RET                             // 子程序 2 结束

//＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊
//子程序 3
SBR3                            // 子程序 3，“循环步数”
LDW>=   MW12，5                 // 如果 r2>5/9，则
INCW    MW14                    // 步数增加 1。
RET                             // 子程序 3 结束

//＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊

//中断程序 0，“脉冲输出终止”

INT0                            // 中断程序 0
R       M0. 1，1                // “电机运行“标志位复位（M0.1=0）
RET1                            // 中断程序 0 结束
```

请参考 SIMATIC STEP7 编程参考手册 6.3 节“高速输出指令”和 6.2 节“中断指令”，为您提供了更多的关于脉冲序列和中断程序的信息。

24 怎样利用 S7-214 DC/DC/DC 脉冲输出演奏音乐

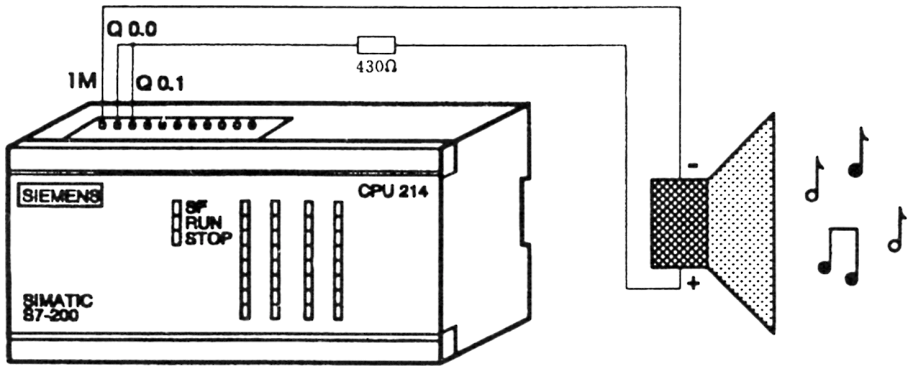
概述

这个应用例子展示怎样利用 S7-214 DC/DC/DC PLC 的脉冲输出功能演奏音乐。为了使音调能持续 0.125 秒，25 个音符周期时间用与之对应的脉冲数存放在音符表中。因为是同时演奏 2 个音符，所以另有两个乐曲表格为通道 0 和通道 1 存储乐曲信息。

两个脉冲通道都被设置成脉冲序列输出（PTO），当演奏每个通道的第一个音符时，就请求第 2 个音符，这样就构成了深度为 1 的队列（一个在进程中，一个在队列中）。

中断子程序附着于 PTO 完成事件。第一个音符演奏完，中断程序调用下一个音符。这个过程继续下去，直到乐曲结束。

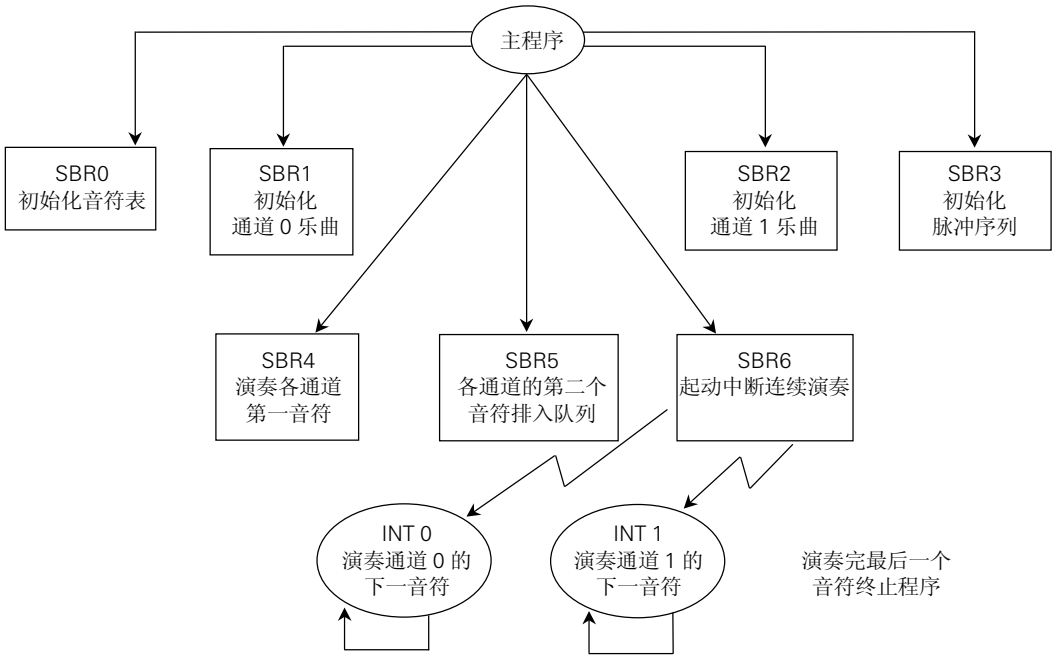
例图



硬件要求

SIMATIC S7-214 DC/DC/DC
电源：115VAC/24 VDC，0.9A（通常 300mA 至 400mA 就可以）
扬声器，430 Ω 电阻

程序框图



程序和注释

此脉冲输出程序长度为 778 个字。

```
// 标题：WHISTLE（号笛）

// 此程序演奏乐曲

// 注释：
// 由于脉冲输出运用方法所限，本应用仅适用于 SIMATIC S7-214 DC/DC/DC。
// SIMATIC S7-214 的输出端 Q0.0 和 Q0.1 串联一只电阻（约 430 Ω，1/2W）与扬声器的一端相连。
// 扬声器的另一端与公共输出端（1M）相连。
// SIMATIC S7-214 的电源输入端（1L+）接+24V
// 各自的地线接在公共输出端（1M）

// 所用内存单元：
//          V4—V103          音符表
//          V500—V503        指向音符表的指针
//          V504—V507        指向通道 0 乐曲表的指针
//          V508—V511        临时的工作寄存器
//          V554—V557        指向通道 1 乐曲表的指针
//          V600—V743        通道 0 乐曲表
//          V800—V1059       通道 1 乐曲表
```

```
// 程序说明：
// 为了使音调能持续 0.125 秒，从“A”（440Hz）开始的 25 个半音阶音符的音符周期时间用与之对应
   的脉冲数存放在音符表中。另外有一表存放通道 0 的乐曲信息，第三个表存放通道 1 的乐曲信息。
// 每个乐曲信息由两个字节音符组成，第一个字节是音符的参考号码（1~25），第二个字节是这个音符
   的时间单位数目（以 0.125 秒为一个时间单位）。
// 两个脉冲通道都被设置为脉冲序列输出（PTO）。首先演奏每个通道的第一个音符，紧接着就请求第
// 二个音符。这样就构成了深度为 1 的队列（1 个在进程中，一个在队列中）。
// 中断程序附着于 PTO 完成事件。第一个音符演奏完，中断程序调进下一个音符。
// 这个过程继续下去，总是保持深度为 1 的队列，直到乐曲结束。

// 程序结构：
// MAIN      初始化程序
// SBR 0     初始化音符表
// SBR 1     初始化通道 0 的乐曲表
// SBR 2     初始化通道 1 的乐曲表
// SBR 3     初始化脉冲序列
// SBR 4     演奏各通道的第一个音符
// SBR 5     各通道的第二个音符排入队列
// SBR 6     起动中断程序连续地演奏
// INT 0     演奏通道 0 的下一个音符
// INT 1     演奏通道 1 的下一个音符

//＊＊＊＊＊＊＊＊＊＊＊ 子程序 0 ＊＊＊＊＊＊＊＊＊＊

// 主程序只在第一个扫描周期中执行，如果主菜单关闭，主程序也结束运行。
// 此程序除了调用子程序外什么也不做。

LD      SM0.1           // 第一次扫描 SM0.1=1
CALL    0               // 初始化音符表
CALL    1               // 初始化通道 0 乐曲表
CALL    2               // 初始化通道 1 乐曲表
CALL    3               // 初始化脉冲序列
CALL    4               // 演奏各通道第一个音符
CALL    5               // 把各通道第二个音符排入队列
CALL    6               // 起动中断程序连续地演奏
MEND                    // 主程序结束

//＊＊＊＊＊＊＊＊＊＊＊ 子程序 0 ＊＊＊＊＊＊＊＊＊＊

// 子程序 0 初始化程序所使用的音符。
// 用 MOVD 指令把用十六进制数表示的音符存于 S7-214 的内存中。前 4 个字符码表征音符的频率，后
// 4 个字符码表示持续音调 0.125 秒所需的脉冲数。
// 例如指令行
// MOVD  16#08E00037，VD4
// 含意是：
// 把音符频率 08E0（16 进制）存进内存 VW4，把脉冲数 0037（十六进制）存进内存 VW6
```

```
SBR     0                    // 子程序 0，初始化音符表。
MOVD    16#08E00037，VD4
MOVD    16#0850003B，VD8
MOVD    16#07E4003E，VD12
MOVD    16#07620042，VD16
MOVD    16#07030046，VD20

MOVD    16#0691004A，VD24
MOVD    16#063D0043E，VD28
MOVD    16#05EB0053，VD32
MOVD    16#058A0058，VD36
MOVD    16#0542005D，VD40
MOVD    16#04ED0063，VD44
MOVD    16#04AE0068，VD48
MOVD    16#04610070，VD52
MOVD    16#04280075，VD56
MOVD    16#03F2007C，VD60
MOVD    16#03B20084，VD64
MOVD    16#0382008B，VD68
MOVD    16#03490095，VD72
MOVD    16#031E009D，VD76
MOVD    16#02F600A5，VD80
MOVD    16#02C500B0，VD84
MOVD    16#02A100BA，VD88
MOVD    16#027700C6，VD92
MOVD    16#025600D1，VD96
MOVD    16#023100DF，VD100
RET

//＊＊＊＊＊＊＊＊＊＊＊ 子程序 1 ＊＊＊＊＊＊＊＊＊＊

// 由于乐曲的演奏需双重的音符，所以必须定义两个不同的通道（0 和 1）。
// 子程序 1 初始化通道 0 的乐曲。
// 每个 MOVD 指令包含 4 个字节，每一个字节是音符的参考号码（1~25），第二个字母是 0.125 秒时间单位的数目；第 3 个字节是下一个音符的参考号码，第 4 个字节是下一个音符的 0.125 秒时间单位的数目。
// 例如指令行
// MOVD  16#05040104，VD600
// 含意是：
// 第一个音符的号码是 5，时间单位数是 4；下一个音符的号码是 1，时间单位数也是 4。
SBR     1                    // 子程序 1，初始化通道 0 的乐曲表。
MOVD    16#05040104，VD600
MOVD    16#03040504，VD604
MOVD    16#03040104，VD608
```

```
MOVD    16#03040504，VD612
MOVD    16#03040304，VD616
MOVD    16#05040604，VD620
MOVD    16#05040304，VD624
MOVD    16#05040602，VD628
MOVD    16#07020804，VD632
MOVD    16#01040304，VD636

MOVD    16#05040304，VD640
MOVD    16#01040304，VD644
MOVD    16#05040304，VD648
MOVD    16#03040504，VD652
MOVD    16#06040504，VD656
MOVD    16#03040504，VD660
MOVD    16#06080608，VD664
MOVD    16#08080A08，VD668
MOVD    16#08080604，VD672
MOVD    16#08040A04，VD676
MOVD    16#0B040D04，VD680
MOVD    16#0F041204，VD684
MOVD    16#11040104，VD688
MOVD    16#03040504，VD692
MOVD    16#03040104，VD696
MOVD    16#03040504，VD700
MOVD    16#03040304，VD704
MOVD    16#05040604，VD708
MOVD    16#05040304，VD712
MOVD    16#05040608，VD716
MOVD    16#03040504，VD720
MOVD    16#06080304，VD724
MOVD    16#05040608，VD728
MOVD    16#03040504，VD732
MOVD    16#06020102，VD736
MOVD    16#0602FFFF，VD740
RET

//＊＊＊＊＊＊＊＊＊＊＊ 子程序 2 ＊＊＊＊＊＊＊＊＊＊

// 由于乐曲的演奏需双重的音符，所以必须定义两个不同的通道（0 和 1）。
// 子程序 2 初始化，通道 1 的乐曲。
// 每个 MOVD 指令包含 4 个字节，第 1 个字节是音符的参考号码（1~25），第 2 个字节是 0.125 秒时
// 间单位数目；第 3 个字节是下一个音符的参考号码（1~25），第 4 个字节是下一音符的 0.125 秒时间
// 单位的数目。
// 请参考 SBR1 的例子。
```

```
SBR     2                    // 子程序 2，初始化通道 1 的乐曲表。
MOVD    16#0D040D02，VD800
MOVD    16#0B020A02，VD804
MOVD    16#0B020D06，VD808
MOVD    16#0D010F01，VD812
MOVD    16#0D020B02，VD816
MOVD    16#0A020B02，VD820
MOVD    16#0D060D02，VD824
MOVD    16#0D030B01，VD828
MOVD    16#0A020802，VD832

MOVD    16#0D020B02，VD836
MOVD    16#0A020802，VD840
MOVD    16#0D020B02，VD844
MOVD    16#0A020802，VD848
MOVD    16#0D060D01，VD852
MOVD    16#0F010D02，VD856
MOVD    16#0B020A02，VD860
MOVD    16#0B020D06，VD864
MOVD    16#0D010F01，VD868
MOVD    16#0D020B02，VD872
MOVD    16#0A020B02，VD876
MOVD    16#0D060D02，VD880
MOVD    16#0D020B02，VD884
MOVD    16#0A020802，VD888
MOVD    16#0D020B02，VD892
MOVD    16#0A020802，VD896
MOVD    16#0D020B02，VD900
MOVD    16#0A020802，VD904
MOVD    16#06060602，VD908
MOVD    16#012021002，VD912
MOVD    16#0F021002，VD916
MOVD    16#012060602，VD920
MOVD    16#012021002，VD924
MOVD    16#0F021002，VD928
MOVD    16#012060602，VD932
MOVD    16#012021002，VD936
MOVD    16#0E021002，VD940
MOVD    16#012021202，VD944
MOVD    16#010020E02，VD948
MOVD    16#0D030F01，VD952
MOVD    16#0D020F02，VD956
MOVD    16#0D010F01，VD960
MOVD    16#0D040D01，VD964
```

```
MOVD    16#0F010D02，VD968
MOVD    16#0B020A02，VD972
MOVD    16#0B020D06，VD976
MOVD    16#0D010F01，VD980
MOVD    16#0D020B02，VD984
MOVD    16#0A020B02，VD988
MOVD    16#0D060D02，VD992
MOVD    16#0D020B02，VD996
MOVD    16#0A020802，VD1000
MOVD    16#0D020B02，VD1004
MOVD    16#0A020802，VD1008
MOVD    16#0D020B02，VD1012
MOVD    16#0A020802，VD1016
MOVD    16#06040F04，VD1020
MOVD    16#0D020B02，VD1024

MOVD    16#0A020802，VD1028
MOVD    16#06040F04，VD1032
MOVD    16#0D020B02，VD1036
MOVD    16#0A20802，VD1040
MOVD    16#06040F04，VD1044
MOVD    16#0D020D02，VD1048
MOVD    16#0F021102，VD1052
MOVD    16#1206FFFF，VD1056
RET

//＊＊＊＊＊＊＊＊＊＊＊ 子程序 3 ＊＊＊＊＊＊＊＊＊＊

// 子程序 3 初始化脉冲序列输出，特殊标志字节 SMB67 定义输出端 Q0.0 输出的方波特性，SMB77 定
// 义输出端 Q0.1 输出的方波特性。
// 16 进制（16#）8D 的含意为：
// 设置使脉冲序列输出有效的控制字
// 选择脉冲序列输出（PTO）操作，每拍（tick）毫秒。
// 可变更脉冲计数值和周期时间值。
SBR     3                       // 初始化脉冲序列
MOVB    16#85，SMB67            // 设置脉冲输出 0（PLS 0）的控制字。
MOVB    16#85，SMB77            // 设置脉冲输出 1（PLS 1）的控制字。
RET

//＊＊＊＊＊＊＊＊＊＊＊ 子程序 4 ＊＊＊＊＊＊＊＊＊＊

// 子程序 4 演奏每个通道的第一个音符，通道 0 的音符参考号码和与之匹配的时间单位数被装入脉冲输
// 出 0（PLS 0），同样的过程在通道 1 中进行（PLS 1），因此能同时演奏两个音符。

SBR     4                       // 子程序 4，演奏每个通道的第一个音符。
MOVD    &VB4，VD500             // 设置指向音符表的指针。
MOVD    &VB600，VD504           // 设置指向通道 0 的乐曲表的指针
```

```
MOVB    * VD504，VB503          // 取出通常 0 的第一个音符
SLW     VW502，2                // 指向音符表（乘以 4）
MOVW    * VD500，SMW68          // 置通道 0 输出脉冲的周期时间
INCD    VD500                   // 指针指向每单位时间的计数
INCD    VD500
INCD    VD504                   // 指针指向时间单位数
MOVD    +0，VD508               // 清除内存 V508
MOVB    * VD504，VB511          // 取出音符的时间单位数
MUL     * VD500，VD508          // 计算音符的总计数值。
MOVD    VD508，SMD72            // 置音符的计数值。
MOVD    &VB4，VD500             // 指针指向音符表
MOVD    &VB800，VD554           // 指针指向通道 1 的乐曲表。
MOVB    * VD554，VB503          // 取出通道 1 的第一个音符。
SLW     VW502，2                // 指向音符表（乘以 4）
MOVW    * VD500，SMW78          // 置通道 1 输出脉冲的周期时间
INCD    VD500                   // 指针指向每单位时间的计数
INCD    VD500
INCD    VD554                   // 指针指向时间单位数。
MOVD    +0，VD508               // 清除内存 V508。
MOVB    * VD554，VB511          // 取出音符的时间单位数
MUL     * VD500，VD508          // 计算音符的总计数值。
MOVD    VD508，SMD82            // 置音符的计数值

PLS     0                       // 演奏每个通道的第一个音符
PLS     1

RET                             // 子程序 4 结束

//＊＊＊＊＊＊＊＊＊＊＊ 子程序 5 ＊＊＊＊＊＊＊＊＊＊

// 子程序 5 把每个通道的第二个音符排入队列，通道 0 的音符参考号码和与之匹配的时间单位数被装入
// 脉冲输出 0（PLS 0），同样的过程在通道 1 进行（PLS 1），因此，能同时演奏两个音符。

SBR     5                       // 子程序 5
MOVD    &VB4，VD500             // 指针指向音符表
INCD    VD504                   // 指针指向通道 0 的第二个音符
MOVB    * VD504，VB503          // 取出通常 0 的第二个音符
SLW     VW502，2                // 指向音符表（乘以 4）
MOVW    * VD500，SMW68          // 置通道 0 输出脉冲的周期时间
INCD    VD500                   // 指针指向每单位时间的计数
IBCD    VD500
INCD    VD504                   // 指针指向时间单位数
MOVD    +0，VD508               // 清除内存 V508
MOVB    * VD504，VB511          // 取出音符的时间单位数
```

```
MUL     * VD500，VD508          // 计算音符的总计数值。
MOVD    VD508，SMD72            // 置音符的计数值。
MOVD    &VB4，VD500             // 指针指向音符表
INCD    VD554                   // 指针指向通道 1 的第 2 个音符
MOVB    * VD554，VB503          // 取出通道 1 的第 2 个音符。
SLW     VW502，2                // 指向音符表（乘以 4）
MOVW    * VD500，SMW78          // 置通道 1 输出脉冲的周期时间
INCD    VD500                   // 指针指向每单位时间的计数
INCD    VD500
INCD    VD554                   // 指针指向时间单位数。
MOVD    +0，VD508               // 清除内存 V508。
MOVB    * VD554，VB511          // 取出音符的时间单位数
MUL     * VD500，VD508          // 计算音符的总计数值。
MOVD    VD508，SMD82            // 置音符的计数值

PLS     0                       // 把每个通道的第二个音符排入队列
PLS     1

RET                             // 子程序 5 结束

//* * * * * * * * * * * 子程序 6 * * * * * * * * * *

// 子程序 6 设置的中断对于连续地演奏乐曲是很重要的。
// 此程序附上了中断，中断 0 是脉冲输出 0（PLS 0）的脉冲计数中断事件 19，中断 1 是脉冲输出 1
// （PLS 1）的脉冲计数中断事件 20，演奏完一个音符后中断发生。

SBR     6                       // 子程序 6
ATCH    +0，19                  // 把中断程序 0 附着于 PLS 0 完成的事件（事件 19）
ATCH    +1，20                  // 把中断程序 0 附着于 PLS 1 完成的事件（事件 20）
ENI                             // 允许中断
RET                             // 子程序 6 结束

//* * * * * * * * * * * 中断程序 0 * * * * * * * * * *

// 演奏完通道 0 的一个音符后，就激活中断程序 0，下一个音符的参考号码和计算出的时间单位数将被
// 传给通道 0 的脉冲输出（PLS 0）
// 当乐曲的最后一个音符奏完时，将自动禁止中断。因此，程序终止。
// 中断程序 0

INT     0                       // 演奏通道 0 的下一个音符
MOVD    &VB4，VD500             // 指针指向音符表
INCD    VD504                   // 指针指向通道 0 的下一个音符
MOVB    * VD504，VB503          // 取出通常 0 的下一个音符
LDB=    VB503，16#FF            // 如果是结束音符，则终止程序。
DTCH    +19
CRETI
```

```
LD      SM0.0
SLW     VW502，2              // 指向音符表（乘以 4）
MOVW    * VD500，SMW68        // 置通道 0 输出脉冲的周期时间
INCD    VD500                 // 指针指向每单位时间的计数
INCD    VD500
INCD    VD504                 // 指针指向时间单位数
MOVD    +0，VD508             // 清除内存 V508
MOVB    * VD504，VB511        // 取出音符的时间单位数
MUL     * VD500，VD508        // 计算音符的总计数值。
MOVD    VD508，SMD72          // 置音符的计数值
PLS     0                     // 请求通道 0 的下一个音符
RETI                          // 中断程序 0 结束

//＊＊＊＊＊＊＊＊＊＊＊＊ 中断程序 1 ＊＊＊＊＊＊＊＊＊＊

// 演奏完通道 1 的一个音符后，就激活中断程序 1，下一个音符的参考号码和计算出的时间单位数将被
// 传给通道 1 的脉冲输出（PLS 1）
// 当乐曲的最后一个音符奏完时，就自动禁止中断。因此，程序终止。

// 中断程序 1

INT     1                     // 演奏通道 1 的下一个音符
MOVD    &VB4，VD500           // 指针指向音符表
INCD    VD504                 // 指针指向通道 1 的下一个音符
MOVB    * VD554，VB503        // 取出通常 1 的下一个音符
LDB=    VB503，16#FF          // 如果是结束音符，则终止程序。
DTCH    +20
CRETI
LD      SM0.0
SLW     VW502，2              // 指向音符表（乘以 4）
MOVW    * VD500，SMW78        // 置通道 1 输出脉冲的周期时间
INCD    VD500                 // 指针指向每单位时间的计数
INCD    VD500
INCD    VD554                 // 指针指向时间单位数
MOVD    +0，VD508             // 清除内存 V508
MOVB    * VD554，VB511        // 取出音符的时间单位数
MUL     * VD500，VD508        // 计算音符的总计数值。
MOVD    VD508，SMD82          // 置音符的计数值
PLS     1                     // 请求通道 1 的下一个音符
RETI                          // 中断程序 1 结束
```

请参考 SIMATIC STEP 7 编程参考手册 6.3 节“高速输出指令”，为您提供了更多的关于脉冲输出功能的信息。

 西门子 版权所有

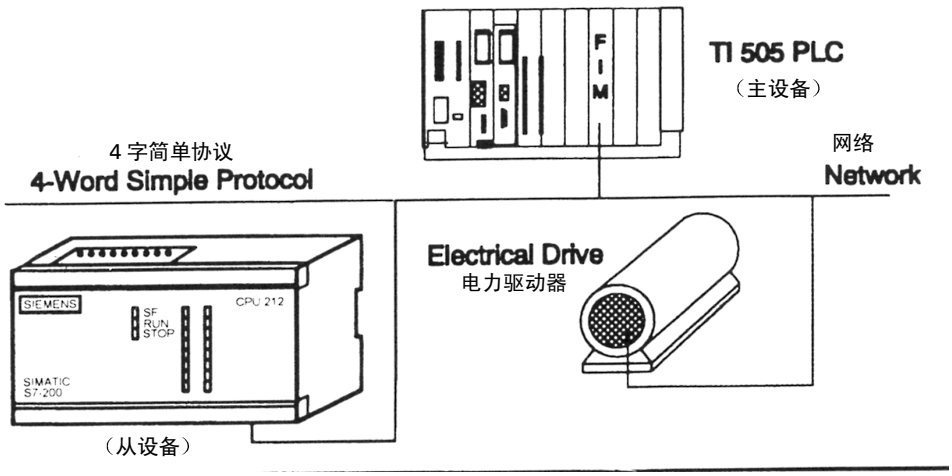
25 TI505 系统通过现场接口模板（FIM）连接 SIMATIC S7-212

概述

本例描述了如何将 SIMATIC S7-212（或 S7-214）与 SIMATIC TI505 可编程逻辑控制系统连接起来。主设备（TI505）通过现场接口模板（FIM）向从设备（S7-212）发送信息。数据传输的协议为 4 字（4-Word）简单协议。这样 TI505 可控制新型 SIMATIC PLC 及其它支持此协议的设备（例如某些 SE & A 驱动器）。

通过自由端模式（Freeport Mode），S7-212 接收来自主设置的信息，以及向主设备发送信息。由几个中断程序完成从设备的数据处理。

例图



硬件要求

SIMATIC S7-212 或 S7-214
有两个阳性接口的 RS485 电缆线
SIMATIC TI505（如 TI545 或 TI555）
用于 SIMATIC TI505 的现场接口模板

程序结构

MAIN	程序初始化
SBR 0	接收 FIM 请求的设置
INT 0	静止线定时器到时
INT 1	寻找静止线
INT 2	接收起始字符
INT 3	接收地址字符

INT 4 接收数据字符
INT 5 接收检验和
INT 6 信息定时器到时
INT 7 发送信息并设置定时器
INT 8 发送完成或终止。

程序和注释

本程序长度为 181 个字。

```
// 标题：FIM
// 本例说明了 TI505 系统通过现场接口模板（FIM）连接 S7-200。
// SIMATIC S7-212 或 S7-214 作为 SIMATIC TI505 的从设备而工作。
// 为设置 FIM 各开关，须作如下调整：
// 开关 1：FIM 的序号
// 开关 2：协议模式为 4 字简单协议 0
// 开关 3 所含 8 个 Dip 开关的设置如下：

//      Dip    开关 1   闭
//      Dip    开关 2   闭
//      Dip    开关 3   闭
//      Dip    开关 4   闭
//      Dip    开关 5   闭
//      Dip    开关 6   闭
//      Dip    开关 7   闭
//      Dip    开关 8   闭

// 4 字简单协议有如下结构：
//      02     起始     1 字节
//      AA     地址     1 字节
//      MM     信息     8 字节
//      …      …
//      MM                信息
//      CC 检验和         1 字节

// 程序结构:
//      MAIN     程序初始化
//      SBR 0    接收 FIM 请求的设置
//      INT 0    静止线定时器到时
//      INT 1    寻找静止线
//      INT 2    接收起始字符
//      INT 3    接收地址字符
//      INT 4    接收数据字符
//      INT 5    接收检验和
//      INT 6    信息定时器到时
//      INT 7    发送信息并设置定时器
//      INT 8    发送完居或终止。
```

```
//* * * * * * * * * * * 主程序 * * * * * * * * * *

// 主程序调用设置予程序 0，并设置协议模式。
// 如果 SIMATIC S7-212 的模式开头处于 TERM 位置，则使用 PPI（点到点接口）协议。
// 如果开关处于 RUN 位置，则设置自由端模式。
// 第一个输入字节（IB 0）被复制到 V 存储器发送缓冲区，而 V 存储器接收缓冲区的第一个字节被复制
// 到输出 QB0。
// 主程序

LD      SM0.1           // 第 1 次扫描 SM0.1=1
CALL    0               // 调用设置子程序 0
LD      SM0.7           // 当 TERM 模式时，则设置 PPI 协议
=       SM30.0          // 当 RUN 模式时，则设置自由端协议
LD      SM0.0           // SM0.0 总是 1
MOVB    VB100，QB0      // 将接收到的第一个字节的第 1 个字节复制到输出 QB0
MOVB    1B0，VB203      // 将输入 IB0 复制到第 1 个发送字的第一个字节
MEND                    // 主程序结束

//* * * * * * * * * * * 主程序 * * * * * * * * * *

// 子程序 0 是为接收 FIM 请求的设置程序，它设置自由端模式（9600 波特，每字符 8 位，偶校验）
// 子程序 0
SBR     0               // 为接收 FIM 请求而设置
MOVB    16#44，SMB30    // 9600 波特，每字符 8 位，偶校验
MOVD    &VB100，VD50    // 将接收缓冲区的指针赋给 VD50
MOVB    +11，VB200      // 赋发送字符数目（11 存入 VB200）
MOVB    +2，VB201       // 赋 STX（起始）字符（2 存入 VB201）
MVOB    +2，VB202       // 赋地址字符（2 存入 VB202）
MOVD    &VB201，VD60    // 将发送缓冲区的指针赋给 VD60
ENI                     // 允许中断
MOVB    +5，SMB34       // 设置静止线定时器 5ms（定时器 0）
ATCH    0，10           // 起动静止线定时器（定时器 0 的中断事件 10 调用中断程序 0）
ATCH    1，8            // 调用中断程序 1 寻找静止线（接收中断事件 8 调用中断程序 1）
RET

//* * * * * * * * * * * 中断程序 0 * * * * * * * * * *

// 若静止线定时器到时了，则中断程序 0 进行中断处理。
// 激活 STX（起始）程序。静止线是指两条信息之间的间隔时间。

// 中断程序 0
INT     0               // 静止线定时器到时了
DTCH    10              // 停止静止线定时器中断（事件 10）
MOVD    VD50，VD56      // 将接收缓冲区的指针赋给 VD56
MOVW    +8，VW54        // 准备接收 8 个数据字符
ATCH    2，8            // 激活接收 STX 程序（接收中断事件 8 调用中断程序 2）
RETI                    // 中断程序 0 结束
```

```
//＊＊＊＊＊＊＊＊＊＊＊ 中断程序 1 ＊＊＊＊＊＊＊＊＊＊

// 当中断发生时，中断程序 1 寻找静止线
// 中断程序 1

INT     1                   // 寻找静止线
ATCH    0，10               // 重新起动静止线定时器（定时器中断事件 10 调用中断程序 0）
RETI                        // 中断程序 1 结束

//＊＊＊＊＊＊＊＊＊＊＊ 中断程序 2 ＊＊＊＊＊＊＊＊＊＊

// 当接收到一个 STX（起始）字符时，中断程序 2 中断进程

// 中断程序 2

INT     2                   // 接收 STX 字符
LDB=    SMB2，2             // 是 STX 字符吗？
MOVW    +2，AC0             // 起始检验和
MOVB    +20，SMB34          // 设置信息定时器为 20ms（定时器 0）
ATCH    3，8                // 调用中断程序 3 接收地址（接收中断事件 8 调用中断程序 3）
ATCH    6，10               // 若信息定时器时间到，则调用中断程序 6
CRETI
NOT                         // 栈顶值取反
ATCH    0，10               // 重新起动静止线定时器（定时器中断事件 10 调用中断程序 0）
ATCH    1，8                // 调用中断程序 1 寻找静止线（接收中断事件 8 调用中断程序 1）
RETI                        // 中断程序 2 结束

//＊＊＊＊＊＊＊＊＊＊＊ 中断程序 3 ＊＊＊＊＊＊＊＊＊＊

// 当接收到一个地址字符时，激活中断程序 3

// 中断程序 3
INT     3                   // 接收地址字符
LDB=    SMB2，2             // 是本站信息吗？
XORM    SMW1，AC0           // 计算检验和
ATCH    4，8                // 调用中断程序 4 接收数据
GRETI
NOT                         // 栈顶值取反
MOVB    +5，SMB34           // 设置静止线定时器为 5ms
ATCH    0，10               // 设置起动静止线定时器
ATCH    1，8                // 调用中断程序 1 寻找静止线
RETI                        // 中断程序 3
```

```
//＊＊＊＊＊＊＊＊＊＊＊ 中断程序 4 ＊＊＊＊＊＊＊＊＊＊

// 当接收到数据字符时，激活中断程序 4

// 中断程序 4
INT     4                   // 接收数据字符
XORW    SMW1，AC0           // 计算检验和
MOVB    SMB2，* VD56        // 存储接收数据
INCD    SMB2，* VD56        // 数据指针加 1
DECW    VW54                // 每接收一个字符，字符计数器减 1
LD      SM1.0               // 若接收完所有数据字，则取检验和
ATCH    5，8                // 调用中断程序 5 接收检验和
RETI                        // 中断程序 4 结束

//＊＊＊＊＊＊＊＊＊＊＊ 中断程序 5 ＊＊＊＊＊＊＊＊＊＊

// 当接收到检验和，激活中断程序 5

// 中断程序 5
INT     5                   // 接收检验和
DTCH    8                   // 停止接收
XORW    SMW1，AC0           // 计算检验和
ANDW    16#FF，AC0          // 分离出检验和（AC0）的最低有效（LS）字节
LDN     SM1.0               // 若零标志位 SM1.0=0，则放弃信息。
MOVB    5，SMB34            // 设置静止线定时器为 5ms
ATCH    0，10               // 起动静止线定时器
ATCH    1，8                // 调用中断程序 1 寻找静止线
CRETI
LD      SM1.0               // 若零标志位 SM1.0=1，则检验和有效。
MOVB    +3，SMB34           // 设置发送延时定时器为 3ms
ATCH    7，10               // 发送延时时间到，调用 INT 7。
MOVB    +2，AC0             // 初始化 AC0
MOVW    +9，AC1             // 赋检验和计算结果为字符个数
MOVD    VD60，VD66          // 将发送缓冲区的指针赋给 VD66

LBL     0                   // 检查和循环
LD      SM0.0               // 标志位 SM0.0 总是 1
XORW    * VD66，AC0         // 计算发送检验和
INCD    VD66                // 指向下一个字符
DECW    AC1                 // 字符计数器减 1
LDN     SM1.0               // 若零标志位 SM1.0=0，则继续检验和。
JMP     0                   // 转向检验和循环

LD      SM1.0               // 若零标志位 SM1.0=1，则存储检验和
INCD    VD66                // 指向检验和字符
MOVB    AC0，* VD66         // 存储检验和
RETI                        // 中断程序 5 结束
```

```
//＊＊＊＊＊＊＊＊＊＊＊ 中断程序 6 ＊＊＊＊＊＊＊＊＊＊

// 当信息定时器到时了，中断程序 6 中断进程

// 中断程序 6
INT     6                   // 信息定时器到时
MOVD    VD50，VD56          // 将接收缓冲区的指针赋给 VD56
MOVW    +8，VW54            // 准备接收 8 个数据字
MOVB    +5，SMB3            // 设置静止线定时器为 5ms
ATCH    1，8                // 起动寻找静止线的程序
ATCH    0，10               // 起动静止线定时器
RETI                        // 中断程序 6 结束

//＊＊＊＊＊＊＊＊＊＊＊ 中断程序 7 ＊＊＊＊＊＊＊＊＊＊

// 中断程序 7 设置发送延时定时器并发送信息

// 中断程序 7
INT     7                   // 发送延时定时器
MOVB    +15，SMB34          // 设置发送定时器为 15ms
ATCH    8，10               // 起发送信息定时器
ATCH    8，9                // 激活发送信息中断
XMT     VB200，0            // 发送信息
RETI                        // 中断程序 7 结束

//＊＊＊＊＊＊＊＊＊＊＊ 中断程序 8 ＊＊＊＊＊＊＊＊＊＊

// 中断程序 8 设置发送延时定时器并发送信息

// 中断程序 8
INT     8                   // 发送信息
DTCH    9                   //禁止发送中断
MOVB    +15，SMB34          // 设置静止线定时器为 5ms
ATCH    8，9                // 起动静止线定时器
ATCH    1，8                // 调用中断程序 1 寻找静止线
RETI                        // 中断程序 8 结束
```

请参考 SIMATIC STEP 7 编程参考手册 6.2 节“中断指令”，为您提供了更多的关于中断程序的信息。

 西门子 版权所有

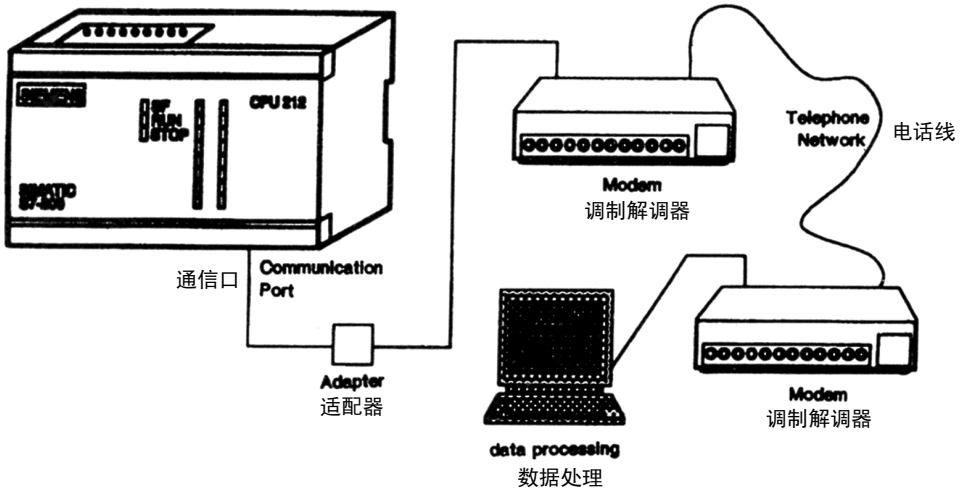
26 SIMATIC S7-212 通过自由通信口模式控制海叶斯（Hayes）调制解调器的应用

概述

这一应用描述了如何通过 SIMATIC 调制解调器来使用与标准海叶斯（Hayes）兼容的调制解调器，以及如何发送信息串。与 S7-200 相连的调制解调器，通过自由通信口模式（Freeport Mode）拨号叫另一台 S7-212 或 S7-214。由于海叶斯调制解调器只能支持 7 个奇偶数据位，因而不能使用 PPI 模式，所以只能通过自由通信口模式来发送信息。

S7-200 也可用作通信从设备。

例图



硬件要求

为了实现该程序的功能，你需要：

1 台　SIMATIC S7-212 或 S7-214

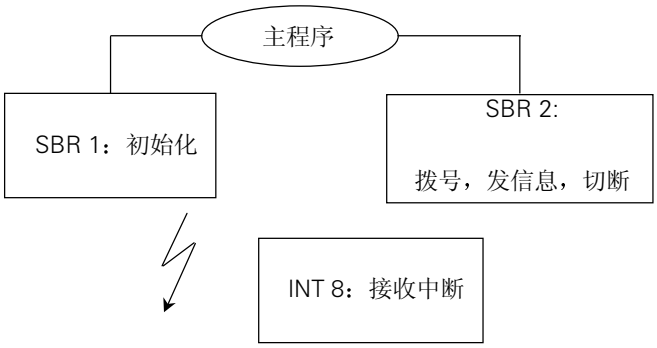
1 条　PC \ PPI　电缆

1 台　合适的接口（取决于调制解调器的输出，多数情况下是用 9 针阳性转换到 25 针阴性的插座，发送线和接收线互换的空调解调器）

1 台　海叶斯（Hayes）调制解调器

海叶期调制解调器应该有 9600 波特率，但用稍慢一点的调制解调器也可以。

程序框图



程序和注释

本程序长度为 189 个字。

```
// 该应用将对通过调制解调器呼叫主系统进行初始化。

// 如果有信息要发送，SIMATIC S7-212 或 S7 212-214 发送信息给海叶斯调制解调器，数据包括所要呼
// 叫的电话号码和所要发出的信息，信息存储在 PLC 的存储器中。

// 程序结构：
//       MAIN      对程序进行初始化
//       SBR 2     拨号、发送和切断
//       SBR 1     对信息和自由通信口模式进行初始化
//       INT 8     捕捉并处理来自调制解调器的响应

//* * * * * * * * * * * 主程序 * * * * * * * * * * *

// 主程序
LD      SM0.1                 // 如果第一次扫描，则
CALL    1                     // 对信息进行初始化
LDB=    +0,MB0                // 如果要发信息，则
NOT
CALL    2                     // 调用子程序进行发送。
LD      SM0.1                 // 如果第一次扫描，则
MOVB    16#1，MB0             // 对发送的信息进行初始化
MEND                          // 主程序结束
```

```
// * * * * * * * * * * * 子程序 2 * * * * * * * * * * *
// 子程序 2 拨号，发送信息，然后切断。
// 子程序 2
SBR     2                   // 子程序 2
LDB=    16#0，MB0           // 如果命令有效，则
NOT
TON     T37，600            // 激活定时器 T37（100ms×600=60s），
LD      737                 // 如果定时器 T37 到时，则
MOVB    16#FF，MB3          // 显示错误
STOP                        // 异常结束

LD      SM0.0               // SM0.0 总是 1
TON     T38，15             // 运行定时器 T38（15×100ms=1.5s），可用于所有情况。

// 状态 1——拨号叫调制解调器

LD      M0.0                // 如果处于拨号状态，则
XMT     VB1010，0           // 拨号叫调制解调器
MOVB    16#2，MB0           // 转到等待状态
CRET                        // 异常结束

// 状态 2——等待连接和发送信息

LD      M0.1                // 如果处于等待连接状态
A       M2.1                // 且得到连接响应
A       T38                 // 且等待定时器 T38 到时，则
XMT     VB130.0             // 发信息
MOVB    16#4，MB0           // 转到等待状态
CRET                        // 条件返回

// 状态 3——等待发送信息

LD      M0.2                // 如果正在等待发送结束
A       SM4.5               // 且发送已结束，则
MOVB    16#8，MB0           // 转到挂起（hang up）状态

// 状态 4 到 8：挂起电话
// 挂起电话有以下 5 种状态（状态 4 至状态 8）：
//      4）     等待 1.5 秒
//      5）     发换码（Eacape）序列（+++）
//      6）     等待 1.5 秒
//      7）     发挂起命令
//      8）     等待离线
```

```
// 状态 4——第一次挂起暂停
LD      M0.3                    // 如果是第一次等待，则
MOVB    16#10，MB0              // 显示发送状态，
R       T38，1                  // 复位等待定时器 T38
MOVW    +0，VW1008              // 清除“+”计数器
CRET                            // 跳过剩余部分
// 状态 5——发送换码(Escape)序列,这必须是发送 3 条信息的序列,只发送含 3 个字符的一条信        //
息，它将不起作用。
LD      M0.4                    // 如果是发换码序列状态
AW=     +3，VW1008              // 且计数器=3，则
MOVB    16#20，MB0              // 转到第二等待状态
R       T38，1                  // 复位等待定时器 T38
CRET                            // 返回
LD      M0.4                    // 如果是发送状态，且
A       T38                     // 等待定时器 T38 到时，则
XMT     VB1000，0               // 发送换码序列
INCW    VW1008                  // “+”计数器加 1
CRET                            // 返回
// 状态 6——第二次挂起暂停（pause）
LD      M0.5                    // 如果是第二等待状态，且
A       SM4.5                   // 发送完毕，则
MOVB    15#40，MB0              // 显示挂起状态
R       T38，1                  // 复位等待定时器 T38
CRET                            // 返回
// 状态 7——发送挂起命令
LD      M0.6                    // 如果是挂起状态
A       T38                     // 且等待定时器 T38 到时，则
XMT     VB1002，0               // 发送挂起命令
MOVB    16#80，MB0              // 转到第三等待状态。
// 状态 8——等待来自调制解调器的 ACK
LD      M0.7                    // 如果是第三等待状态，且
A       T38                     // 等待定时器 T38 到时，则
MOVB    16#0，MB0               // 发送无效
MOVB    +0，MB2                 // 清除错误响应
RET                             // 子程序 2 结束
```

```
//  ********** 子程序 1 **********

// 子程序 1 执行信息和自由通信口模式的初始化
// 存储单元分配：
//          V1000-V1001          在线换码字符序列
//          V1002-V1007          挂起命令
//          V1008-V1009          换码字符序列计数器
//          V1010-V1???          拨号命令和电话号码

// 子程序 1

SBR      1                       // 子程序 1
LD       SM0.0                   // SM0.0 总是 1
MOVW     16#143，VW1000          // 在线换码序列字符（+）
MOVB     16#5，VB1002            // 设置挂起命令
MOVD     16#41544830，VD1003
MOVB     16#D，VB1007            // 回车

MOVB     16#9，VB1010            // 设置拨号指令
MOVD     16#41544454，VD1011     // 用按钮拨号
MOVD     16#32363137，VD1015     // 2617：一个调制解调器电话号码
MOVB     16#0D，VB1019           // 回车，发送信息
MOVB     +20，VB130              // 设置一条发送信息：S7-200 PHONE HOME
MOVD     16#53372D32，VD131      // 字符 S7-2
MOVW     16#3030，VW135          // 字符 00
MOVW     16#2020，VW137          // 字符␣␣
MOVD     16#50484F4E，VD139      // 字符 PHON
MOVB     16#45，VB143            // 字符 E
MOVB     16#20，VB144            // 字符␣
MOVD     16#484F4D45，VD145      // 字符 HOME
MOVW     16#D0A，VW149
MOVB     16#9，SMB30             // 设置自由通信口：9600 波特，每字符 8 位，无奇偶校
MOVB     +0，MB0                 // 显示空状态
MOVB     +0，MB2                 // 清除错误响应
ATCH     +8，8                   // 指定接收中断事件 8 调用中断程序 8
ENI                              // 允许用户中断
R        T37，1                  // 复位定时器 T37
REN                              // 子程序 1 结束

//  ********** 中断程序 8 **********

// 这个中断服务程序捕捉来自调制解调器的响应并且处理它们，代码的含义：

//          0                    // OK（好）
//          1                    // 连接
```

```
//          2                   // 振铃
//          3                   // 无载体
//          4                   // 错误
//          5                   // 连接 1200
//          6                   // 无拨号按钮
//          7                   // 忙（占线）
//          8                   // 无回答
//          10                  // 连接 2400
//          11                  // 连接 4800
//          12                  // 连接 9600

// 如果选中选项 X0，则只有代码 0，1，2，3，4。
// 代码 0 和 1 是可接受的，代码 2、3、4 是错误的。

// 实际被接收的框架是代码加 CR

// 中断程序 8

INT         8                   // 中断程序 8

LDB=        16#D，SMB2          // 如果 CR
CRET1                           // 则异常结束

LDB=        16#30，SMB2         // 如果“0”（OK）
MOVB        16#1，MB2           // 则置“OK”标志
CRETI                           // 返回

LDB=        16#31，SMB2         // 如果“1”（连接），则
MOVB        16#2，MB2           // 置连接状态
R           T38，1              // 复位等待定时器 T38
CRETI                           // 返回
MOVB        16#4，MB2           // 否则，显示出错标志
RETI                            // 中断子程序 8 结束
```

请参考 SIMATIC STEP 7 程序参考手册 6.4 节“发送指令”，为您提供了更多的关于发送指令 XMT 的信息。

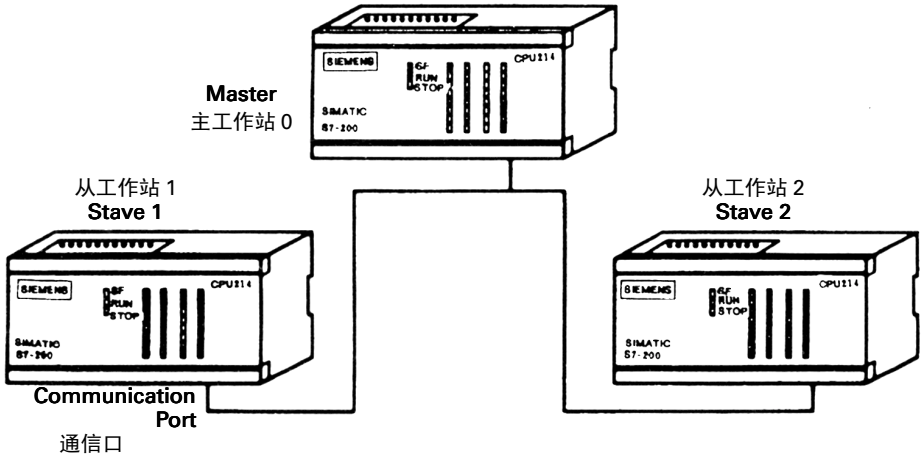
27 几台 SIMATIC S7-200PLC 使用自由通信口模式连接在一个远程 I/O 网络上

概述

在这个例子中连接了三台 SIMATIC S7-214CPU。工作站 0 被称为主工作站（Master），与工作站 1 和 2 相连，而工作站 1 和 2 被称为从工作站（Slave）。主工作站轮流发送四个字节的输出数据到每个从工作站。随之每个从工作站响应产生四个字节的输入数据。自由通信口模式（Freeport Mode）被用来进行数据传输。

配备 2 个存储缓冲区，一个用作远程输入，另一个用作远程输出。发送的输出数据可从发送缓冲区获取，该数据是从输出缓冲区移到发送缓冲区的两个字长度的值。发送后，主工作站接收从工作站的响应，并且将数据存储在接收缓冲区。

例图



硬件要求

如要实现该程序的功能，你需要：

2 台以上 SIMATIC S7-212 或 S7-214

1 根 9 芯电缆连接线

如果使用 2 台以上 PLC（多于 1 台从工作台），则另需一台网络连接器（Siemens-MLFB：6ES7-972-0BB00-0XA0 或 6ES5-762-2AA21）。

主工作站程序结构

Main	主程序
SBR0	选择 PPI 通信或 Freeport(自由通信口)通信
INT0	接收定时器中断程序
INT1	发送定时器中断程序
INT10	在发送完输出数据后的发送中断程序

INT11	接收信息第一个字符的中断程序
INT12	接收输入数据的中断程序
INT13	接收 FCS 字符的中断程序
INT14	静止线接收器中断程序

主工作站程序和注解

主工作台用于远程 I/O 的程序长度为 191 个字。

```
// 此程序用于连接两台、三台或更多台的 SIMATIC S7-214 或 SIMATIC
// S7-212，构成一个通信网络

// UART 初始化如下：奇校验
                        每字符 8 位
                        38，400 波特（S7-214）
                        19，200 波特（S7-212）

// 如果你仅仅使用 SIMATIC S7-214PLC，则应用将会以 38，400 波特运行。
// 如果你仅仅使用 SIMATIC S7-212，则应用将会自动地以 19，200 波特运行。
// 如果你同时使用 SIMATIC S7-214 和 S7-212 PLC，则程序需要更改。
// 有一点注意：只需将 SMB30 字节的设置值从 16#C0 改为 16#C4，以及从 16#C1 改为 16#C5，则应
// 用将会以 19，200 波特运行。
// 程序功能：
// 每条信息必须由一个检查序列字符所验证，该字符由此信息的数据字节进行异或运算形成。
// 远端 I/O 主工作站为工作站 0，并且可以连接 1 台、2 台或 3 台从工作站。如果只连接了一台从工作
// 站，则它必须为工作站 1。
// 如果连接了 2 台从工作站，则它们必须是工作站 1 和 2。如果连接了 3 台从工作站，则它们必须是工
// 作站 1、2 和 3。所连接的工作站的数目必须作为一个参数提供给主工作站存入内存 V0，对于从工作
// 站，将工作站的地址存入内存 V0。
// 主工作站必须轮流发送四个字节的输出数据到每个从工作站，随之每个从工作站必须响应产生四个字
// 节的输入数据。配备两个 V 存储缓冲区，一个用作远程输入，另一个用作远程输出。主工作站中的缓
// 冲区大小如下面所示：
//      工作站 1          工作站 2          工作站 3
//      输入缓冲区        输入缓冲区        输入缓冲区
//      VB500 字节 0      VB504 字节 0      VB508 字节 0
//      VB501 字节 1      VB505 字节 1      VB509 字节 1
//      VB502 字节 2      VB506 字节 2      VB510 字节 2
//      VB503 字节 3      VB507 字节 3      VB511 字节 3

//      工作站 1          工作站 2          工作站 3
//      输出缓冲区        输出缓冲区        输出缓冲区
//      VB540 字节 0      VB544 字节 0      VB548 字节 0
//      VB541 字节 1      VB545 字节 1      VB549 字节 1
//      VB542 字节 2      VB546 字节 2      VB550 字节 2
//      VB543 字节 3      VB547 字节 3      VB551 字节 3
```

```
// 发送的输出数据可从发送缓冲区获取，该数据是从输出缓冲区移到发送援冲区的两个字长度的值。
// 发送后，主工作站接收从工作站的响应，并且将数据存储在接收缓冲区内。发送和接收缓冲区的格式
// 和下面所示。其中 VB 607 是在产生发送检查和时所使用的存储单元。

// 发送缓冲区                发送缓冲区
// VB600 长度                VB608 字节 0
// VB601 地址                VB609 字节 1
// VB602 字节 0              VB610 字节 2
// VB603 字节 1              VB611 字节 3
// VB604 字节 2
// VB605 字节 3
// VB606FCS
// VB607xx

// 信息格式如下所示：
// 地址          B0    B1    B2    B3    FCS

//  * * * * * * * * * * * 主程序 * * * * * * * * * *

// 主菜单初始化程序
LD        SM0.0                  // SM0.0 总是 1
CALL      0                      // 调用子程序 0
// 用户程序由此开始
// 用户程序由此结束
MEND                             // 主程序结束

//  * * * * * * * * * * * 子程序 0 * * * * * * * * * *

// 子程序 0 初始化自由通信口模式，并且处理批针
// 子程序 0
SBR       0                      // 子程序 0
LDN       SM0.7                  // 当开关处于 TERM 位置时，则
MOVB      16#C0，SMB30           // 使自由通信口模式无效
DTCH      +8                     // 使接收中断无效
DTCH      +9                     // 使发送中断无效
DTCH      +10                    // 定时器中断无效
RET

LDN       SM30.0                 // 如果不是自由通信口模式（即 PPI 模式），则使
MOVB      16#C1，SMB30           // 自由通信口模式有效：奇校验，每个字符 8 位，38.4K 波特
ENI                              // 允许中断
MOVB      VB1，SMB34             // 设置定时器
CRET
```

```
LD      SM0.0                   // SM0.0 总是 1
R       17.0，1                 // 复位远程 I/O 指示器（I7.0=0）
MOVD    &VB540，VD630           // 指针指向输出数据缓冲区
MOVD    &VB500，VD634           // 指针指向输入数据缓冲区
MOVB    +6，VB600               // 发送缓冲区长度（6 个字节）
MOVB    +1，VB601               // 工作站地址=1
MOVD    * VD630，VD602          // 置入发送的数据
MOVW    VW602，AC0              // 计算 FCS
XORW    VW604，AC0
MOVB    AC0，VB606
XORW    AC0，VW606              // 存储 FCS
ATCH    +1，10                  // 使发送定时器有效（定时器中断事件 10 调中断程序 0）
ATCH    +10，9                  // 使发送中断有效（发送中断事件 9 调中断程序 10）
XMT     VB600，0                // 发送数据
LBL     0                       // 跳转（JMP）标识符 0
LDN     17.0                    // 如果远程 I/O 更新还没有完成
JMP     0                       // 则等待它完成
LD      SM0.0
INCW    AC1
MOVW    AC1，AC2
SWAP    AC1
MOVW    AC1，VW544
SWAP    AC1
SRW     AC2，4
SWAP    AC2
MOVW    AC2，VW540
MOVB    VB500，QB0
MOVW    SMW22，VW10
EET                             // 子程序 0 结束

//  * * * * * * * * * * * 中断程序 0 * * * * * * * * * * *

// 如果发送或接收数据，或重新起动定时器，则中断程序 0，将禁止接收中断和定时器中断。

// 中断程序 0
INT     0                       // 当接收定时器到时，调中断程序 0
LD      SM0.0                   // SM0.0 总是 1
DTCH    +8，                    // 使接收中断无效
DTCH    +10                     // 使定时器中断无效
LDB>=   VB601，VB0              // 如果这是网络中的最后一个从工作站，则
=       17.0                    // 指示远程 I/O 循环结束
CRETI
```

```
LD      SM0.0
INCW    VW600               // 否则，工作站地址加 1
+D,     +4，VD630           // 增大指针，指向下个工作站的输出数据缓冲区
+D      +4，VD634           // 增大指针，指向下个工作站的输出数据缓冲区
MOVD    * VD630，VD602      // 置入发送的数据
MOVW    VW620，AC0          // 计算 FCS
XORW    VW604，AC0
MOVB    AC0，VB606
XORW    AC0，VW606          // 存储 FCS
ATCH    +1，10              // 使发送定时器有效（定时器中断事件 10 调中断程序 1）
ATCH    +10，9              // 使发送中断有效（发送中断事件 9 调中断程序 10）
XMT     VB600，0            // 发送数据
RETI                        // 中断程序 0 结束

//  * * * * * * * * * * * 中断程序 1 * * * * * * * * * *

// 中断程序 1 个处理 XMT（发送）定时器到时
INT     1                   // 当发送信息定时器到时，调用中断程序 1
LD      SM0.0               // SM0.0 总是 1
DTCH    +10                 // 停止定时器
DTCH    +9                  // 停止发送器
STOP                        // 引起程序模式的转换
RETI                        // 中断程序 1 结束

//  * * * * * * * * * * * 中断程序 10 * * * * * * * * * *

// 中断程序 10 中断程序，以接收数据
// 中断程序 10
INT     10                  // 发送器中断（发送中断事件 9 调中断程序 10）
LD      SM0.0               // SM0.0 总是 1
DTCH    +9                  // 使发送中断无效
ATCH    +0，10              // 起动接收定时器
ATCH    +11，8              // 使接收数据有效（接收中断事件 8 调中断程序 11）
RETI                        // 中断程序 10 结束

//  * * * * * * * * * * * 中断程序 11 * * * * * * * * * *

// 如果接收到信息的第一个字符，那么中断程序 11 中断程序

// 中断程序 11
INT     11                  // 接收信息的第一个字符
LDN     SM3.0               // 如果没有奇偶校验错误
AB=     SMB2，VB601         // 且有正确的工作站响应，则
MOVB=   +0，AC0             // 初始化检查和寄存器
MOVW    +4，AC1             // 置入接收字符总数
```

```
MOVD    &VB608，VD638       //VD638 指针指向接收缓冲区
ATCH    +12，8              // 使接收数据有效（接收中断事件 8 调中断程序 12）
CRETI

LD      SM0.0               // 否则，错误的工作站响应或有错误
ATCH    +0，10              // 使接收定时器有效
ATCH    +14，8              // 使静止线接收器有效
RETI                        // 中断程序 11 结束

//  ＊＊＊＊＊＊＊＊＊＊＊ 中断程序 12 ＊＊＊＊＊＊＊＊＊＊

// 如果接收到数据，那么中断程序 12 中断程序
// 中断程序 12
INT     12                  // 接收数据
LDN     SM3.0               // 如果没有奇偶，校验错误，则
MOVB    SMB2，*VD638        // 将数据存储于接收寄存器中

INCD    VD638               // 指向下一个接收缓冲区的地址
XORW    SMW1，AC0           // 计算正在运行的检查和
DECW    AC1                 // 接收的字符数减 1
LD      SM1.0               // 如果接收到四个字符，则
ATCH    +13，8              // 使接收 FCS 有效
CRETI

LD      SM3.0               // 如果有奇偶校验错误，则
ATCH    +0，10              // 使接收定时器有效
ATCH    +14，8              // 使静止线接收器有效
RETI                        // 中断程序 12 结束

//  ＊＊＊＊＊＊＊＊＊＊＊ 中断程序 13 ＊＊＊＊＊＊＊＊＊＊

// 中断程序 13
INT     13                  // 接收 FCS 字符
LD      SM0.0               // SM0.0 总是 1
ATCH    +0，10              // 使接收定时器有效
LD      SM3.0               // 如果有奇偶校验错误
AB=     SMB2，AC0           // 或者如果 FCS 不匹配
CRETI
LD      SM0.0               // 否则
MOVD    VD608，*VD634       // 存储接收到的数据
RETI                        // 中断程序 13 结束
```

```
//  ＊＊＊＊＊＊＊＊＊＊＊ 中断程序 14 ＊＊＊＊＊＊＊＊＊＊
// 中断程序 14 重新激发接收定时器
// 中断程序 14
INT     14                  // 静止线接收器
LD      SM0.0               // SM0.0 总是 1
ATCH    +0，10              // 重新激发接收定时器
RETI                        // 中断程序 14 结束
```

请参考 SIMATIC STEP7 编程参考手册，为您提供了下述更多的信息：

SWAP 指令可见于 4.6 节“MOVE 指令”；
Shift 指令可见于 5.1 节“Shift 指令”；
中断程序可见于 6.2 节“中断指令”；
XMT 可见于 6.4 节“发送指令”。

从工作站程序结构

Main	主程序
SBR 0	激活用于初始化工作站的程序
SBR 1	使通信无效
INT 0	静止线定时器
INT 1	发送定时器中断程序
INT 2	信息定时器
INT 10	发送输入数据后的发送中断
INT 11	接收信息的首个字符
INT 12	接收输出数据
INT 13	接收 FCS 字符
INT 14	静止线接收器

从工作站程序和注释

从工作站用于远程 I/O 的程序长度为 148 个字。

```
// 此程序连接两台、三台或更多台 SIMATIC S7-214 或 SIMATIC S7-212，构成通信网络
// UART 初始化如下：奇校验
                    每字符 8 位
                    38，400 波特（S7-214）
                    19，200 波特（S7-212）
// 如果你仅仅使用 SIMATIC S7-214，则应用将会以 38，400 波特运行。
// 如果你仅仅使用 SIMATIC S7-212，则应用将会自动地以 19，200 波特运行。
// 如果你同时使用 SIMATIC S7-214 和 S7-212PLC，则程序需要更改。
// 有一点注意：只需将字节 SMB 30 设置值从 16#C0 改为 16#C4，以及从 16#C1 改为 16#C5，则应用
// 将会以 19，200 波特运行。
```

```
// 程序功能：
// 每条信息必须由一个检查序列字符所验证，该字符由此信息中的数据字节进行异或运算形成。
// 远端 I/O 主工作站为工作站 0，并且可以连接一台、两台或三台从工作站。
// 如果只连接了一台从工作台，则它必须为工作站 1。如果连接了 2 台从工作站，则它们必须为工作站
1 和 2。
// 如果连接了 3 台工作站，则它们必须为工作站 1，2 和 3。所连接的从工作站的数目必须作为一个参
// 数提供给主工作站存入内存 V0，对于从工作站，将工作站地址存入 V0。
// 每个工作站都有一个如下所示的输入和输出缓冲区：

//                输入缓冲区
//        IB0     字节 0
//        IB1     字节 1
//        IB2     字节 2
//        IB3     字节 3

//                输出缓冲区
//        QB0      字节 0
//        QB1      字节 1
//        QB2      字节 2
//        QB3      字节 3

// 当主工作站发送一条信息时，则从工作站由主工作站存储在接收缓冲区中的输出数据所赋址；并且响
// 应主工作站的发送，发送存于发送缓冲区中的输入数据。
// 这些缓冲区如下所示，其中 VB 607 是在产生发送检查和时所使用的存储单元。

//        发送缓冲区              发送缓冲区
//        VB600       长度        VB608     字节 0
//        VB601       地址        VB609     字节 1
//        VB602       字节 0      VB610     字节 2
//        VB603       字节 1      VB611     字节 3
//        VB604       字节 2
//        VB605       字节 3
//        VB606       字节 FCS
//        VB607       xx

//信息格式如下所示：

        地址          B0   B1   B2   B3   FCS

//  * * * * * * * * * * * 主程序 * * * * * * * * * *

// 主菜单初始化程序

// 主程序
LD      SM0.7                   // 如果开关在 RUN 位置
A       SM0.1                   // 且为首次扫描，则
```

```
CALL    0                       // 起动通信
LD      SM0.7                   // 当开关被移到 RUN 位置
EU                              // 上升沿
CALL    0                       // 如果开关在 TERM 位置，则
LD      SM0.7                   // 起动的通信
CALL    1                       // 使通信无效
MEND                            // 主程序结束

//   * * * * * * * * * * * 子程序 0 * * * * * * * * * *

// 子程序 0 设置自由通信口模式，并使静止线定时器和静止线接收器有效

// 子程序 0
SBR     0                       // 初始化从工作站
LD      SM0.0                   // SM0.0 总是 1
MOVB    16#C1，SMB30            // 使自由通信口模式有效：奇校验，每字符 8 位，38.4K 波特
ENI                             // 允许中断
MOVB    VB1，SMB34              // 设置定时器为 5ms
ATCH    +0，10                  // 使静止线定时器有效（定时器中断事件 10 调中断程序 0）
ATCH    +14，8                  // 使静止线接收器有效（接收中断事件 8 调中断程序 14）
RET                             // 子程序 0 结束

//* * * * * * * * * * * 子程序 1 * * * * * * * * * *

// 子程序 1 使自由通信口模式通信无效，且禁止接收、发送和定时器中断

// 子程序 1

SBR     1                       // 使通信无效
LD      SM0.0                   // SM0.0 总是 1
MOVB    16#C0，SMB30            // 使自由通信口模式无效
DTCH    +8                      // 使接收器无效
DTCH    +9                      // 使发送器无效
DTCH    +10                     // 使定时器无效
RET                             // 子程序 1 结束

//* * * * * * * * * * * 中断程序 0 * * * * * * * * * *

// 中断程序 0 使接收输入数据有效

// 中断程序 0
INT     0                       // 当静止线定时器到时，调中断程序 0
LD      SM0.0                   // SM0.0 总是 1
ATCH    +11，8                  // 使接收输入数据有效
RET                             // 中断程序 0 结束
```

```
//＊＊＊＊＊＊＊＊＊＊＊ 中断程序 1 ＊＊＊＊＊＊＊＊＊＊
// 如果发送或接收数据，或重新激发定时器，则中断程序 1 将禁止定时器中断和发送中断
// 中断程序 1
INT     1                   // 当发送信息定时器到时，调中断程序 1
LD      SM0.0               // SM0.0 总是 1
DTCH    +10                 // 停止定时器
DTCH    +9                  // 停止发送器
STOP                        // 引起程序模式的转换
RETI                        // 中断程序 1 结束
//   ＊＊＊＊＊＊＊＊＊＊＊ 中断程序 2 ＊＊＊＊＊＊＊＊＊＊
// 当接收到信息的首个字符时，中断程序 2 中断程序
// 中断程序 2
INT     2                   // 当信息定时器到时，调中断程序 2
LD      SM0.0               // SM0.0 总是 1
ATCH    +0，10              // 使静止线定时器有效
ATCH    +14，8              // 使静止线接收器有效
RETI                        // 中断程序 2 结束
//   ＊＊＊＊＊＊＊＊＊＊＊ 中断程序 10 ＊＊＊＊＊＊＊＊＊＊
// 如果发送信息，那么中断程序 10 中断程序。发送中断无效，静止线定时器开始运行，且使静止线接
收器有效。
// 中断程序 10
INT     10                  // 发送器中断
LD      SM0.0               // SM0.0 总是 1
DTCH    +9                  // 使发送器中断无效
ATCH    +0，10              // 起动静止线定时器
ATCH    +14，8              // 使静止线接收器有效
RETI
//   ＊＊＊＊＊＊＊＊＊＊＊ 中断程序 11 ＊＊＊＊＊＊＊＊＊＊
// 当接收到信息的首个字符时，中断程序 11 有效。设置信息定时器，可以接收数据。如果有错误，即
// 使静止线定时器和静止线接收器有效。
// 中断程序 11
INT     11                  // 接收信息的首个字符
LDN     SM3.0               // 如果无奇偶校验错误
AB=     SMB2，VB0           // 且是本工作站的信息，则
MOVB    +0，AC0             // 初始化检查和寄存器
MOVW    +4，AC1             // 置入接收字符数
```

```
MOVD    &VB 608，VD638      //VD638 指针指向接收缓冲区
ATCH    +2，10              // 使信息定时器有效
ATCH    12，8               // 使接收数据有效
CRETI

LD      SM0.0               // 否则，如不同工作站或有错误，则
ATCH    +0，10              // 使静止线定时器有效
ATCH    +14，8              // 使静止线接受器有效
RETI                        // 中断程序 11 结束

//＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊＊
//＊＊＊＊＊＊＊＊＊＊＊ 中断程序 3 ＊＊＊＊＊＊＊＊＊＊

// 如果接收到数据且无错误，那么中断程序 12 中断程序。
// 将信息存储于缓冲区。结构检查序列 FCS 可被另一个中断所接收。
// 如果有错误发生，则激活静止线定时器和静止线接收器。

// 中断程序 12
INT     12                  // 接收数据
LDN     SM3.0               // 如果无奇偶校验错误则
MOVB    SMB2，*VD638        // 将数据存储在接收寄存器中
INCD    VD638               // 指向下一个接收缓冲区地址
XORW    SMW1，AC0           // 计算正在运行的检查和
DECW    AC1                 // 接收字符计数器减 1
LD      SM1.0               // 如果接收到四个字符，则
ATCH    +13，8              // 使接收 FCS 有效
CRETI

LD      SM3.0               // 如果有奇偶，校检错误，则
ATCH    +0，10              // 使静止线定时器有效
ATCH    +14，8              // 使静止线接收器有效
RETI                        // 中断程序 12 结束

//＊＊＊＊＊＊＊＊＊＊＊ 中断程序 13 ＊＊＊＊＊＊＊＊＊＊

// 中断程序 13 接收结构检查序列字符（FCS），无定时器和无接收中断被激活。当计算 FCS 后，激活发
// 送器定时中断和发送器中断。如果有错误，则激活静止线定时器和静止线接收器。

// 中断程序 13
INT     13                  // 接收 FCS 字符
LD      SM0.0               //SM0.0 总是 1
DTCH    +8                  // 使接收器无效
DTCH    +10                 // 使定时器无效
LDN     SM3.0               // 如果无奇偶校验错误
AB=     SMB2，AC0           // 且 FCS 匹配，则
MOVD    VD608，QD0          // 存储输出数据
```

```
MOVB    +6，VB600           // 置入发送缓冲区的长度
MOVB    VB0，VB601          // 将工作站地址置入发送缓冲区
MOVD    1D0，VD602          // 将输入数据置入发送缓冲区
MOVW    VW602，AC0          // 计算 FCS
XORW    VW604，AC0
MOVB    AC0，VB606
XORW    AC0，VW606          // 存储 FCS
ATCH    +1，10              // 使发送器定时器有效
ATCH    +10，9              // 使发送器中断有效
XMT     VB 600，0           // 发送数据
CRETI
LD      SM0.0               // 否则，如果有奇偶校验错误，则
ATCH    +0，10              // 激活静止线定时器
ATCH    +14，8              // 激活静止线接收器

RETI                        // 中断程序 13 结束

//＊＊＊＊＊＊＊＊＊＊＊ 中断程序 14 ＊＊＊＊＊＊＊＊＊＊

// 中断程序 14 激活定时器中断
INT     14                  // 静止线接收器
LD      SM0.0               // SM0.0 总是 1
ATCH    +0，10              // 重新激活静止线定时器
RETI                        // 中断程序 14 结束
```

请参考 SIMATIC STEP-7 编程参考手册，为您提供了下述更多的信息：

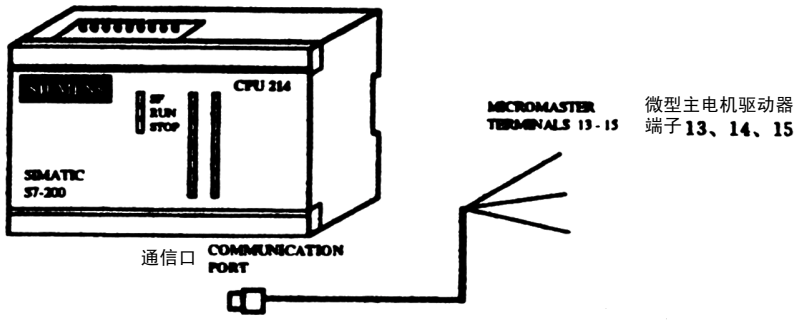
SWAP 指令可见于 4.6 “MOVE 指令”;
Shift 指令可见于 5.1 节 “Shift 指令”;
中断程序可见于 6.2 节 “中断指令”;
发送可见于 6.4 “发送指令”。

28 S7-214 与 SIMOVERT 电机驱动器之间的自由通信口通信接口

概述

S7-214 通过与一台 SIMOVERT 微型主电机驱动器通信来起动，停止电机，以及改变输出到电机的频率。通信是通过 S7-200 自由通信口模式进行，使用 USS5 字协议。输入仿真器用来初始化发给电机驱动器的命令。

例图



硬件要求

这个程序假定使用者已正确地将电机和微型主电机驱动器接好线，并且所有的电机和微型主电机驱动器的参数已通过人工设定了。必须把微型主电机驱动器设置在遥控方式(P910=1)，波特率 19.2kb(P 92=7),地址 1(P91=1)。

要求用一根带 9 针阳性插头的通信电缆接在 S7-214 的 1，3，8 端上，电缆另一端是无插头的，以便接到微型主电机驱动器的 13，14，15 端子上(1 接 15，3 接 13，8 接 14)。

程序结构

MAIN	主程序：监视用于自由通信口/PPI 通信切换的 RUN/TERM 开关，寻找输入信号上升沿作业电机命令。
SBR0	设置自由通信口通信：首次扫描时设置自由通信口模式的参数。
SBR1	RUN 子程序：设定电机恒速运转。
SBR2	RAMP 子程序：设定电机变速动转。
SBR3	增加频率倍率的子程序：增加微型主电机驱动器的输出频率。
SBR4	降低频率倍率的子程序：降低微型主电机驱动器的输出频率。
SBR5	STOP 子程序：停止电机。
SBR6	计算输出信息的 BCC。
SBR7	发送信息，初始化发送定时器。
INT0	发送结束的中断处理程序，打开接收器。
INT1	发送超时的中断处理程序，再试发送，最多试发 3 次。

INT2 接收字符的中断程序，结束后进行有效性校验。
INT3 接收超时的中断处理程序，再试接收最多试收 3 次。

程序和注释

S7-214 通过与一台 SIMOVERT 微型主电机驱动器通信来起动、停止电机，以及改变输出到电机的频率，通信是通过 S7-200 自由通信口，模式进行，使用 USS5 字协议。输入仿真器用来初始化发给电机驱动器的命令，程序监视 RUM/TERM 开关，并选择相应的协议来设置自由通信口模式的控制字节（SMB 30）。程序监视如下电机命令的输入点：

10.0	上升沿	使电机以上次命令的恒定频率运转。
10.2	上升沿	使电机以上次命令的频率开始变频运转。 频率可用 10.6 和 10.7 升高或降低
10.4	上升沿	停止电机。
10.5	电平	以 1x 或 2x 倍率改变频率。10.5：0→1x，1→2x。
10.6	上长沿	以 1x 或 2x 频率增量（此例中为 50）增加电机频率。
10.7	上升沿	以 1x 或 2x 频率增量（此例中为 50）降低电机频率。
11.0	电平	电机旋转方向：I1.0：0→CW，1→CCW

程序检测并报告通信错误。首先对微型主电机驱动器的发送要计时，如果失败，允许再试发送，最多可试发送 3 次。然后，对来自微型主电机驱动器的接收亦要计时，在退出发送接收操作之前可重试多达 3 次。对来自微型主电机驱动器的响应信息要进行有效性校验（STX，LEN，ADR 和 BCC），任何被检测到的错误都要由 QBO 显示，直到下一次操作结束，QBO 的显示含义如下：

0 无错误
1 非法的响应（除去坏的 BCC）。
2 坏的 BCC。
3 发送超时。
4 接收超时。

尽管这个示例程序只与一台微型主电机驱动器通信，可把它扩展用于另外的输入点，选择多站通信线路上的某一台微型主电机驱动器的地址，向它发送命令。另外，这个程序的基本通信结构还可用来发送别的信息给微型主电机驱动器，如监视电流，转矩等。

本程序长度为 342 个字。

```
// 这个程序是 S7-200 自由通信口通信接口与 SIMOVERT 微型主电机驱动器通信的示例
// 两者间通信用 USS5 字协议。
// S7-200 作为主设备，使用 USS 协议地址 0，而微型主电机驱动器则是地址 1。
// 为了正确地应用此例，你需要：
// 1 台带输入仿真器的 SIMATIC S7-212 或 S7-214。
// 1 根 RS485 电缆，其一端为阳型插头，另一端无插头，以接到微型主电机驱动器上。
// 1 台微型主电机驱动器和电机
// 这个例子使用 Herb Taylor，SE&A 的示例单元（驱动器和电机）
// 你要确保正确地设置微型主电机驱动器的参数，并保证将安置于遥控方式
```

 西门子 版权所有

```
// （P910=1），波特率设置正确（P092），从地址设置正确（P091）。这个例子的波特率为 19.2kb，从地址为 1。

// 微型主电机驱动器使用的 USS 串行通信 5 字协议有如下 14 字节结构：
//        02          开始信息（STX）。
//        12          微型主电机驱动器的信息从 AA 到 BCC 长度为 12 字节。
//        AA          设备号地址（本例中驱动器为 1）
//        PKE         参数控制高字节
//        PKE         参数控制低字节
//        IND         陈列序引字的高字节
//        IND         陈列序引字的低字节
//        PWE         参数值和错误代码的高字节。
//        PWE         参数值和错误代码的低字节。
//        PZD1        控制/状态字高字节
//        PZD1        控制/状态字低字节
//        PZD2        主设定 2 低点和返回值的高字节
//        PZD2        主设定点和返回值的低字节
//        BCC         块校验和

// * * * * * * * 注意例中对于开/关/速度、改变只使用数据字：PZD1 和 PZD2。
// 程序结构：
//        MAIN        主程序。
//        SBR0        设定自由通信口通信。
//        SBR1        RUN 子程序。
//        SBR2        RAMP 子程序。
//        SBR3        增加频率倍率的子程序。
//        SBR4        降低频率倍率的子程序。
//        SBR5        STOP 子程序。
//        SBR6        计算输出信息的 BCC。
//        SBR7        发送消息，初始化发送定时器。
//        INT0        发送结束的中断处理程序，打开接收器。
//        INT1        发送超时的中断处理程序。
//        INT2        接收字符中断程序。
//        INT3        接收超时中断处理程序。

// 内存单元分配：
//        VB99        发送信息的长度
//        VB100-VB113 发送缓冲区。
//        VB114-VB127 接以缓冲区。
//        VW200       缺省频率倍率，初始值=5461，频率=（5461/16384）× P094。
//        VW202       频率倍率增/减量，初始值=50。
//        VW204       发送再试次数，初始值为 3 次，每发一次减 1。
//        VW206       接收再试次数，初始值为 3 次，每收一次减 1。
```

```
//          VW208           信息中接收字符数，USS 10=14。
//          VB210           最后一次试操作的状态（也在输出 QB0 显示），其值含义如下：
//                          0   操作正确。
//                          1   来自微型主电机驱动器的非法响应（除去坏的 BCC）。
//                          2   坏的 BCC。
//                          3   发送超时。
//                          4   接收超时。
//          VD211           接收缓冲区地址指针
//          VW215           累积接收信息为 BCC，存于最低字节。
//          VW217-VB218     暂时存储区

// 输入点的功能：
//          10.0    RUN 使电机以当前频率倍率的值运转，方向由输入 L1.0 确定，仅在“STOP”后才影
                    响操作。
//                  上升沿操作
//          10.2    RAMP 使电机以当前频率倍率的值运转，方向由输入 L1.0 确定，仅在“STOP”后才
                    影响操作。
//                  上升沿操作。
//          10.4    STOP 停止电机，允许后续的 RUN/RAMP 命令。
//                  上升沿操作。
//          10.5    指定倍率（1x 或 2x），频率倍率的增/减量，
//                  电平操作：0-1x，1-2x。
//          10.6    增加频率倍率，其值为 VW202x 倍率（10.5）
                    若电机处于 RAMP，则速度立即升高。
//                  上升沿操作。
//          10.7    减少频率倍率，其值为 VW202x 倍率（10.5）
                    若电机处于 RAMP，则速率立即下降。
//                  上升沿操作。
//          I1.0    指定电机旋转方向。电平操作：0→CW， 1→ CCW。

// 首次扫描执行的程序
LD      SN0.1               // 首次扫描时 SM0.1=1
CALL    0                   // 调子程序 0
// 用 RUN/TERM 开关选择自由通信口/PPI 通信。
LD      SM0.7               // RUN/TERM 开关位置：1→RUN， 0→0TERM

// 检查命令
LD      10.4                // 若 STOP（停止电机）命令，且
EU                          // 上升沿，则
CALL    5                   // 往微型主电机驱动器发送信息
S       M0.0，1             // 使后续的 RUN 或 RAMP 命令有效（M0.1=1）。
R       M0.1，1             // M.01=0

LD      10.0                // 若 RUN（运行电机）命令，且
```

```
EU                              // 上升沿
A       M0.0                    // M0.0=1，则
CALL    1                       // 往微型主电机驱动器发送信息
R       M0.0，02                // 使后续的 RUN 命令无效(M0.0=0，M0.1=0)
LD      10.2                    // 若 RAMP 命令，且
EU                              // 上升沿
A       M0.0                    // M0.0=1，则
R       M0.0，1                 // 使后续 RUN 命令无效(M0.0=0)
CALL    2                       // 往微型主电机驱动器发信息
S       M0.1，1                 // 显示现在状态(M0.1=1)

LD      10.6                    // 若增加频率，且
EU                              // 上升沿,则
CALL    3                       // ++速度。

LD      10.7                    // 若减少频率，且
EU                              // 上升沿，则
CALL    4                       // 一速度。

MEND                            // 主程序结束。

// * * * * * * * * * * * * 子程序 0 * * * * * * * * * * *

//初始化子程序——仅在第一次扫描时执行
SBR     0                       // 初始化 XMT 缓冲区，设置通信参数

MOVB    16#44，SMB30            // 19.2kb，每字符 8 位，偶校验
MOVB    16#0E，VB99             // XMT 长度
MOVB    16#02，VB100            // STX
MOVB    16#0C，VB101            // LEN
MOVB    16#01，VB102            // ADR（微型主电机驱动器地址为 1）
FILL    0，VW103，5             // 清除所有 5 个数据字（VW103 至 VW111）以后还可改变

MOVW    16#1500，VW200          // 1/3 最大的频率倍率

MOVW    16#32，VW202            // 频率倍率以 50 增/减

MOVD    16#00030003，VD204      // 设定发送和接收重试次数为 3

MOVB    0，VB210                // 清除操作状态指示
MOVB    0，QB0                  // QB0=0

MOVB    0，VB219                // S7-200 地址

S       M0.0，1                 // 允许 RUN 或 RAMP 或 RAMP（方向改变）

ENI                             // 允许用户中断

RET                             // 子程序 0 结束
```

```
//＊＊＊＊＊＊＊＊＊＊＊＊ 子程序 1 ＊＊＊＊＊＊＊＊＊＊

//运行电机子程序，旋转方向取决于输入 I1.0 的状态
SBR     1                       // 运行电机

MOVB    16#05，VB109            // 为 CW 设置 STW
MOVB    16#7F，VB110            //
MOVW    VW202，VW111            // 设定频率

LD      I1.0                    // 设定旋转方向，若 I1.0=0，则为 CW；若 I1.0=1，则为 CCW
=       V109.3                  //
NOT
=       V109.4

LD      SM0.0                   // SM0.0 总是 1

CALL    6                       // 计算 BCC
CALL    7                       // 初始化。XMT 及 XMT 定时器

RET                             // 子程序 1 结束

//＊＊＊＊＊＊＊＊＊＊＊＊ 子程序 2＊＊＊＊＊＊＊＊＊＊

// RAMP 电机的处理子程序，电机运转时，根据输入 10. 2 的上升沿脉冲，对电机频率进行+/-RAMP 修
// 正，电机旋转方向取决于输入 I1.0 的状态。
// 子程序 2
SBR     2                       // 运行电机

MOVB    16#04，1VB217           // 设置命令字节
LD      V109.3                  // 保存以前的电机旋转方向
=       V217.3                  //
LD      V109.4                  //
=       V217.4                  //
LD      SM0.0                   // SM0，0 总是 1
MOVB    VB217，VB109            // 为 RUN 设置 STW
MOVB    16#7F，VB110            //
MOVW    VW200，VW111            // 设置频率

LD      0.1                     // 检查状态，看是否允许改变电机旋转方向
JMP     0                       // 只有在 STOP（停止）后才允许，否则不允许
                                // 保存以前的电机旋转方向
LD      I1.0                    // 根据输入 I1.0 的状态，设置电机旋转方向
=       V109.3                  //

NOT

=       V109.4
```

```
LBL     0
LD      SM0.0
CALL    6                       // 计算 BCC
CALL    7                       // 初始化 XMT 及 XMT 定时器

RET                             // 子程序 2 结束

// * * * * * * * * * * * 子程序 3 * * * * * * * * * *

// 增加电机频率的处理子程序。频率增量存于 VW202。若 10.5 为 1，则倍增；若上溢出，则设为 32767。
// 子程序 3
SBR     3                       // 增加频率

+1      VW202，VW200            // +因子（增加）
LD      10.5                    // 倍增吗?
+1      VW202，VW200
LDW>=VW200，16384               // 判是否上溢出（大于最大值 16384）?
MOVW    16384，VW200            // 若上溢出，则设为最大值（16384）

LD      M0.1                    // 若 M0.1=1，则
CALL    2                       // 发送信息，增加频率

RET                             // 子程序 3 结束

// * * * * * * * * * * * 子程序 4 * * * * * * * * * *

// 降低电机频率子程序，频率减量存于 VW202，若 10.5 为 1（ON），则倍减；若下溢出，则为 0
// 子程序 4
SBR     4                       // 降低频率

-1      VW202，VW200            // -因子（降低）
LD      10.5                    // 倍减?
-1      VW202，VW200
LD      SM1.2                   // 判是否下溢出（小于 0）?
MOVW    0，VW200                // 若下溢出，则设为 0

LD      M0.1                    // 若 M0.1=1，则
CALL    2                       // 发送信息，降低频率

RET                             // 子程序 4 结束

// * * * * * * * * * * * 子程序 5 * * * * * * * * * *

// 停止电机的子程序
SBR     5                       // 停止电机

MOVB    16#0C，VB109            // 为 STOP 设置 STW
MOVB    16#7E，VB110            //
```

```
CALL    6                       // 计算 BCC
CALL    7                       // 初始化 XMT 及 XMT 定时器

RET                             // 子程序 5 结束

// * * * * * * * * * * * 子程序 6 * * * * * * * * * *

// 用 XOR（异或）计算信息块的检查和，并存入缓冲区。
SBR     6                       // 计算 USS5 字的 BCC

// 十六进制信息为：02，0C，ADR，BYTE1，…，BYTE10，BCC
// 本程序入口时，AC1 指向信息 LEN 字节（0C）的位置。
// 本程序出口时，AC1 指向 BCC 字节位置。
// 将 BCC 存入信息缓冲区，
// AC2 含有 BCC。

MOVD    &VB101，AC1             // 设置缓冲区地址指针
MOVD    16#0E，AC2              // 2xOR12，信息的头两字节

FOR     AC3，1，11              // 计算剩余 11 个字符的 BBC。
XORW    *AC1，AC2
INCD    AC1                     // 地址指针加 1
NEXT

INCD    AC1                     // 指针加 1，指向 BCC 位置
MOVB    AC2，*AC1               // 把 BCC 存到信息缓冲区中。
RET                             // 子程序 6 结束。

// 初始化 XMT 及 XMT 定时器
SBR     7                       // 初始化 XMT，捕捉 XMT 中断

XMT     VB99，0                 // 发送
ATCH    0，9                    // 捕捉 XMT（发送）中断（事件 9），并调用中断程序 0（INT 0）

MOVB    255，SMB34              // 设置 XMT 定时器定 255ms，实际只用约 7ms（19.2kb）
ATCH    1,10                    // 捕捉 XMT 定时器中断（事件 10），并调用中断程序 1（INT 1）
RET                             // 子程序 7 结束
// XMT（发送）中断处理，关掉 XMT 定时器，起动接收从设备响应
INT     0                       // 中断程序 0，XMT 中断处理

DTCH    10                      // 退出 XMT 定时器
DTCH    9                       // 中止 XMT 事件

MOVW    3，VW204                // 刷新 XMT 重试次数（3 次）
MOVW    14，VW208               // 响应信息中接收的字符数（14 个）
MOVW    0，VW215                // 清除 BCC 累加器
MOVD    &VB114，VD211           // 设置接收缓冲区指针
ATCH    2，8                    // 捕捉 RCV（接收）中断，并调用中断程序 2（INT2）
ATCH    3，10                   // 捕捉接收定时器中断（事件 10），并调用中断程序 3（INT3）

RETI                            // 中断程序 0 结束。
```

```
// 中断程序 1
// 若 XMT（发送）定时器时间到，这段程序取得控制权，XMT 操作会重试，直到可重试次数减到 0
INT     1                   // 定时器中断 0 处理—发送
DTCH    9                   // 停止 XMT（发送）
DTCH    10                  // 退出定时器
DECW    VW204               // 重试次数减 1，若为 0，则
LD      SM1.0               // SM1.0=1
MOVB    3，VB210            //
VOVB    3，QB0              // 用 QB0 指示发送超时
MOVW    3，VW204            // 刷新发送重试计数（3 次）
S       M0.01，1            // 使 RUN，RAMP 有效（M0.0=1）
CRET1                       // 条件返回
                            // 重试
XMT     VB99，0             // 发送
ATCH    0，9                // 捕捉 XMT（发送）中断（事件 9），并调用中断程序 0（INT0）
MOVB    255，SMB34          // 设置 XMT（发送）定时器为 255ms，实际只用约 7ms(19.2kb)
ATCH    1，10               // 捕捉定时器中断（事件 10），并调用中断程序 1（INT1）
RET1                        // 中断程序 1 结束
// 中断程序 2
// 这段处理程序计算接收字符数，并校验错误
// 若检测到错误，则再试接收，直至可重试次数减到 0
INT     2                   // 接收字符处理
MOVB    SMB2，AC0           // 得到接收字符
XORW    AC0，VW215          // 累积 BCC，即用 XOR（异或）计算 BCC
MOVB    AC0，* VD211        // 把接收到的字符送入缓冲区
INCD    VD211               // 缓冲区指针加 1
DECW    VW208               // 有待接收的字符总数减 1
LDN     SM1.0               // 检验是否结束
CRETI
NOT
DTCH    10                  // 退出接收定时器
DTCH    8                   // 关掉接收
AB=     0，VB216            // 检验已算好的 BCC 是否为 0
NOT
MOVB    2，VB210            // 坏的 BCC 操作码
MOVB    2，QB0
JMP     0
                            // BCC 正确，检验信息的其它部分
```

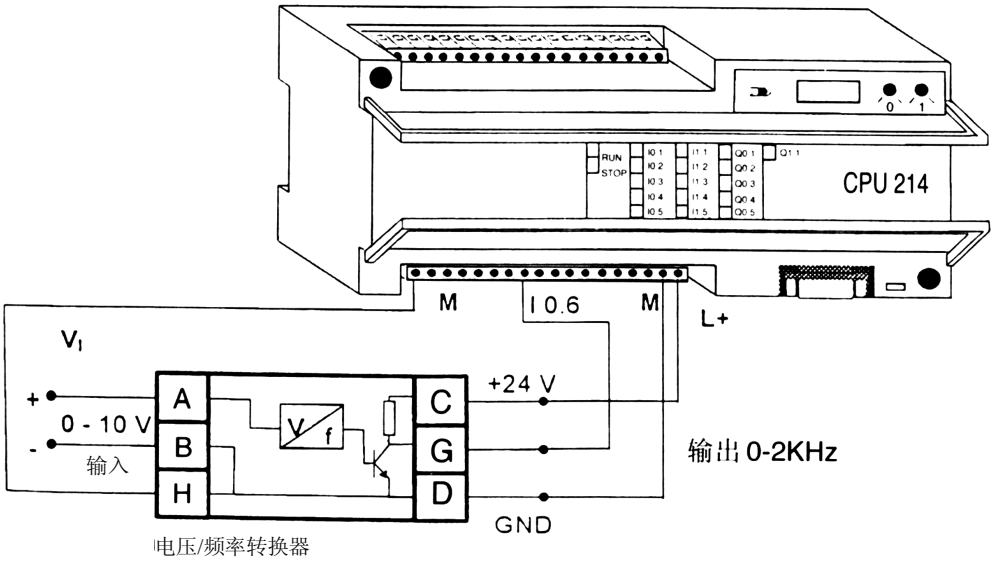
```
LDB=    VB114，16#02          // STX 第一个字符吗？
AB=     VB115，16#0C          // 长度=12 吗？
AB=     VB116，VB102          // 将信息发往同一从设备吗？
//＊＊＊＊

// 任何其它校验都会因相应响应而经过这一段

//＊＊＊＊
MOVB    0，VB210              // 操作正确
MOVB    0，QB0
JMP     0

LD      SM0.0
MOVB    1，VB210              // 信息中有不对的地方
MOVB    1，QB0

LBL     0
MOVW    3，VW206              // 刷新接收可重试次数（3 次）

RET1                          // 中断程序 2 结束

// 中断程序 3
// 若响应的接收时间已到，这段程序取得控制权，再试发送信息，再试一次接收。
// 在超时时操作重试，直至可重试次数减为 0
INT     3                     // 定时器中断 0 处理——接收

DTCH    8                     // 关掉接收中断
DTCH    10                    // 退出接收定时器

DECQ    VW206                 // 检查可重试次数，重试次数减 1，若为 0，则
LD      SM1.0                 // SM1.0=1
MOVB    4，VB210              //
MOVB    4，QB0                // 用 QB0 指示接收超时
MOVW    3，VW206              // 刷新接收重试次数
S       M0.0，1               // 使 RUM，RAMP 有效
CRETI                         // 条件返回

NOT
MOVD    &VB114，VD211         // 设置接收缓冲区的指针
MOVW    0，VW215              // 清除 BCC 累加器
                              // 再重试发送
XMT     VB99，0               // 发送
ATCH    VB0，9                // 捕捉 XMT（发送）中断，并调用中断程序 0

MOVB    255，SMB34            // 设定 XMT（发送）定时器为 255ms
ATCH    1，10                 // 捕捉定时器中断（事件 10），并调用中断程序 1（INT 1）

RET1                          // 中断程序 3 结
```

29 用 S7-200 CPU 214 的高速计数器 HSC 累计来自模拟量/频率转换器（A/F）的脉冲来计算模拟电压值

概述

本例说明了如何利用 CPU 214 的高速计数器 HSC 及频率转换器来计算模拟电压。首先频率转换器将输入电压（0~10V）转换为矩形脉冲信号（0~2000Hz），再将此信号送入 CPU 214 高速记数器的输入端并累计脉冲数。当预置的间隔时间到后，通过累计脉冲数，计算出被测模拟电压值。

例图



电压/频率转换器

硬件要求

需使用如下设备：

1 台 CPU 214	AC/DC/RLY 或 DC/DC/DC
1 台电压/频率转换器	SFW01(Trankner Company)

（地址：Engineering Office Trankner，Industrial Area North PF38，09618 Brand-Erbisdorf，Germany）

技术参数：	供电电压	24V DC
	输　　入	0~10V DC
	输　　出	方波，GND~24V
	测量范围	0~10V⇒0~2000Hz
	比　　率	200Hz/V（线性）

程序结构

主程序	在第一个扫描周期调用子程序 SBR0
SBR0	高速计数器和定时中断的初始化
INT0	对高速计数器求值的定时中断程序

程序和注释

主程序在第一个扫描周期调用初始化程序 SBR0，仅在第一个扫描周期标志位 SM0.1 = 1

由子程序 SBR0 实现初始化。首先，把高速记数器 HSC1 的控制字节 SMB47 置为 16 进制‘FC’，其含义是：正方向计数，可更新预置值（PV），可更新当前值（CV），激活 HSC1。

然后，用指令‘HDEF’把高速计数器 HSC1 置成工作模式 0，即没有复位或起始输入，也没有外部的方向选择。当前值 SMD48 复位为 0，预置值 SMD52 置为 FFFF（16 进制）。定时中断 0 间隔时间 SMB34 置为 100ms，中断程序 0 分配给定时中断 0（中断事件 10），并允许中断。用指令 HSC1 启动高速计数器。

每 100ms 调用一次中断程序 0，读出高速计数器的数值后将其置零。通过 HSC1 计数值及变换关系（0~2KHz 对应于 0~10V）来求被测的模拟电压值。本例中，计数值仅除以 2，然后置入输出字节 QB0，以便通过 LED 来显示被测的模拟电压值。显示值与 10 倍真实电压值相对应。例如，计数值为 200 除以 2 是 100，那么，被测的模拟电压值就是 10.0V。因为计数器 100ms 内共有 200 个计数脉冲，这正与 2000Hz⇒10V 相对应。假设计数值为 104，则实际电压值应为 5.2V。

注意：定时中断间隔时间可在 5~255ms 的范围内变化，然而，通过设立一个标志，可根据需要来延长高速计数器的求值和复位时间，这样就有更长的扫描间隔，以便提高精确度，同时也会带来更长的更新时间。例如，定时中断设为 100ms，每调用一次，标志增加 1，仅当标志满 10 时，才对高速计数器求值和复位。也就是说，10V 电压可接收的最大脉冲为 2000，这样，求值精确到 5/1000V，即精确度是上例的 10 倍，但同时速度也减慢了 10 倍。

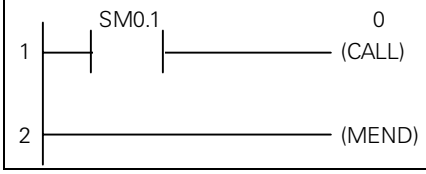
程序长度是 45 个字。

 西门子 版权所有

LAD(S7-MicroDOS)	STL(IEC)

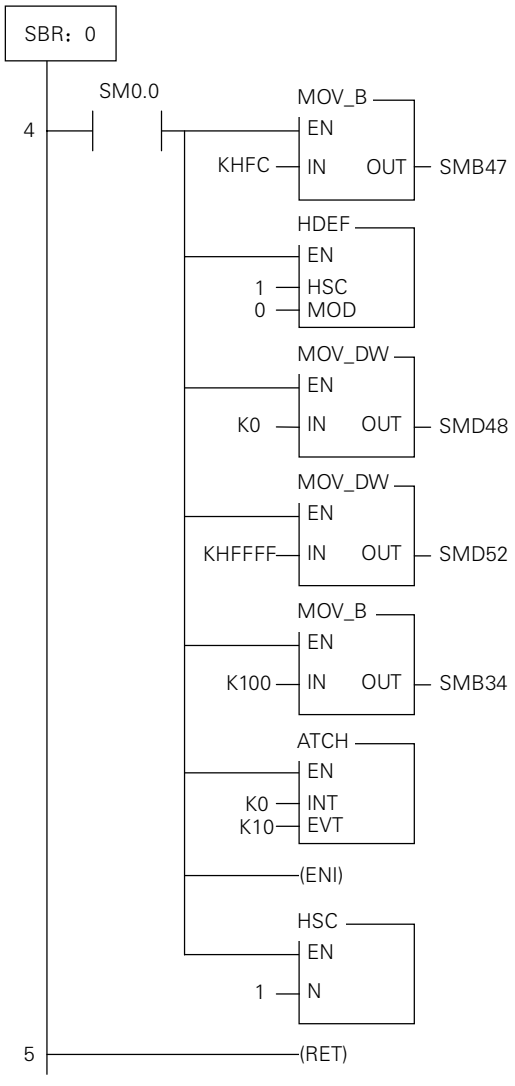
主程序

// 标题：通过 A/F 转换器将 DI 用作 AI



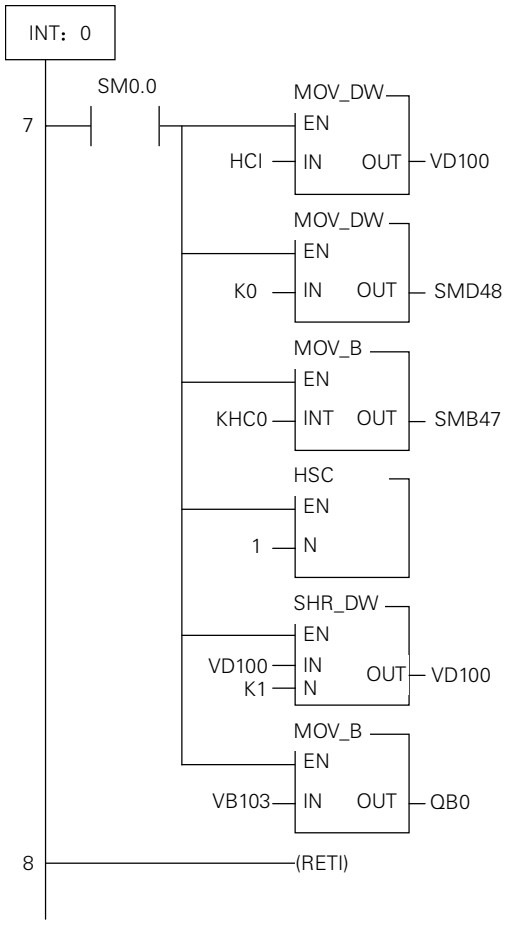

```
LD      SM0.1          // 仅首次扫描时 SM0.1=1
CALL    0              // 调用子程序 0
MEND                   // 结束主程序
```

子程序 0




```
SBR0                   // 启动子程序 0
LD      SM0.0          // SM0.0 总是 1
MOVB    16#FC，SMB47   // 设置 HSC1 控制字节：上升沿复位，上升沿起动，1x 计数速率，正向计数，可改变方向，可更新 PV（预置值），可更新 CV（当前值），激活 HSC1。
HDEF    1，0           // HSC1 工作于模式 0
MOVD    0，SMD48       // HSC1 当前值复位
MOVD    16#FFFF，      // 设置 HSCl 预置值（本
        SMD52          // 例未用）
MOVB    100，SMB34     // 设置定时中断 0 间隔时间为 100ms
ATCH    0，10          // 指定定时中断事件 10 调用中断程序 0
ENI                    // 允许所有中断
HSC     1              // 启动高速计数器 HSC1
RET                    // 结束子程序 0
```

// 中断程序 0



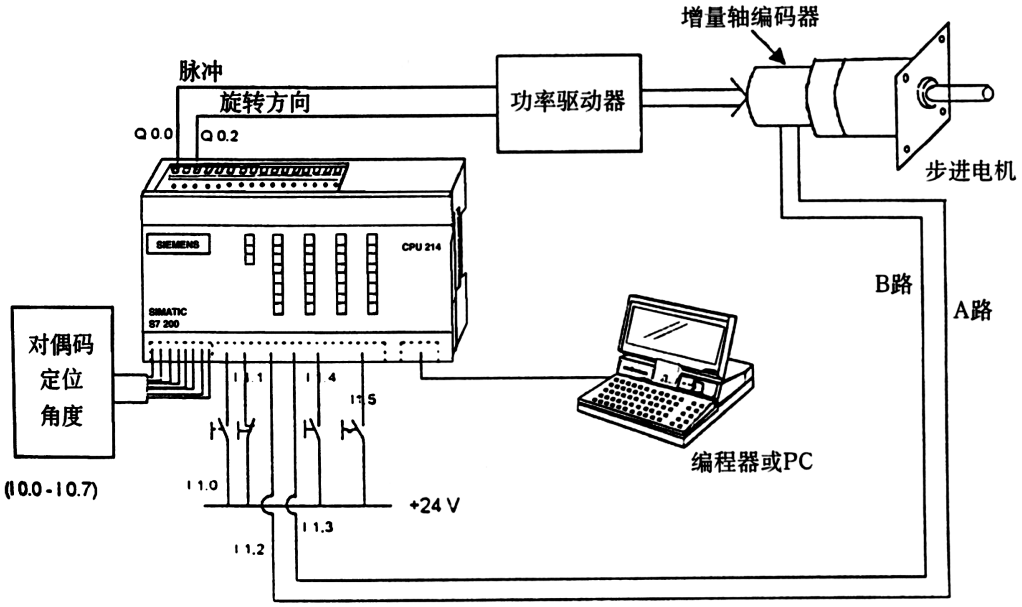

```
INT   0              // 启动中断程序 0
LD    SM0.0          // SM0.0 总是 1
MOVD  HC1，VD100     // HSC1 的计数值存入 VD100
MOVD  0，SMD48       // HSC1 当前值复位
MOVB  16#C0，SMB47   // 重新设置 HSC1 控制字节：上升沿复位，上升沿起动，4x 计数速率，反向计数，不改变方向，不更新 PV，可更新 CV，激活 HSC1。
HSC   1              // 启动高速计数器 HSC1
SRD   VD100，1       // HSC1 的计数值除以 2（即 VD100 右移 1 位）
MOVB  VB103，QB0     // 在输出端 Q0.0 至 Q0.7 显示 10 倍被测电压值（0~100V）
RETI                 // 结束中断程序 0
```

30 用 S7-200 CPU 214 DC/DC/DC 进行定位控制，并具有位置监视和位置校正

概述

本例是建立在例 23 基础上的，所不同的是，相对位置由增量传感器进行位置监视。为了求出传感器信号，将该信号作为 CPU 214 中的最大可处理 7kHz 信号的高速计数器的输入，这样，就可检测出位置误差。例如，当起--停频率超出时，通过步数丢失可以检测到位置错误。一旦检测出位置误差，就以较低频率进行位置校正。

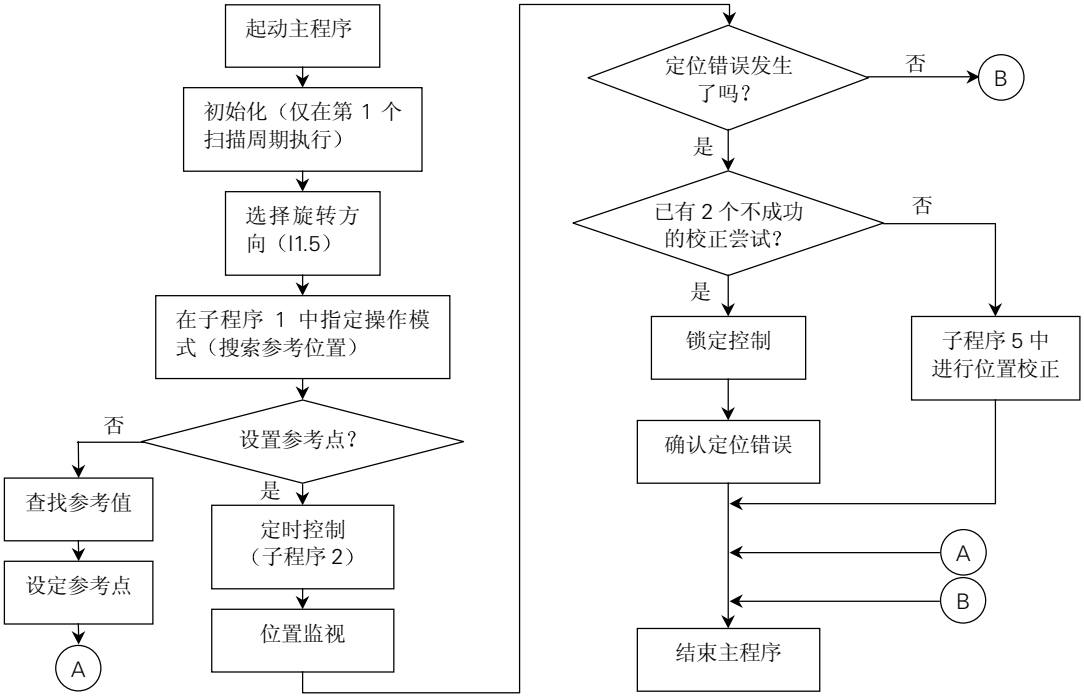
例图



硬件要求

数 量	设备	制造厂/定货号
1	SIMATIC S7-200 CPU 214 DC/DC/DC	Siemens/6ES7 214-1AC00-0XB0
1	PC/PPI 电缆	Siemens/6ES7 901-3BF00-0XA0
1	编程器或 PC	
1	带有相应功率驱动器和相关连接电缆的步进电机	
1	为控制信号提供电源的电缆	
1	24V 增量传感器	
1	轴连接器	
1	传感器信号电缆	
9	开关	
3	按钮	

程序框图



程序和注解

一、初始化

和例 22 及例 23 类似，在程序的第一个扫描周期（SM0.1=1）设置重要的参数。此外，高速计数器 HSC2 由外部复位并初始化为 A/B 计数器。HSC2 对检测定位的增量轴编码器信号计数。传感器的 A 路和 B 路信号分别作为 CPU 输入端 I1.2 和 I1.3 的输入。

旋转方向的选择、按钮锁定、操作模式的选择及定位的过程都和例 23 相同（请参考此例概述）。与例 23 不同的是，由增量传感器进行定位监视，在输出脉冲结束之后，等待 T1 时间，以便使连接电机和传感器的轴连接器的扭转振动消失。

二、实际值和设定值的比较

T1 到时后，子程序 4 对实际值和设定值进行比较。如果轴的位置在设定位置的±2 步范围内，定位就是正确的。如果实际位置在此目标范围之外，当超过起停频率时，那就会造成电机失步这种情况的发生，此时，一个相应的警告信号就会则 Q1.1 输出。

三、位置的较正

若定位错误被检测出来，则起动第二等待定时器 T2。此后，根据设定值和实际值之间的差值计算出校正的步数。当校正时，电机频率低于起停频率，以防新的步数丢失。

四、校正取消

如果在两次校正尝试之后还不能达到设定位置，为安全起见，控制将被锁定（M0.2=1）。只有按下确认按钮 I1.4 之后，控制才被打开，然后，进行另一个参考点的检测。

五、信号清单：

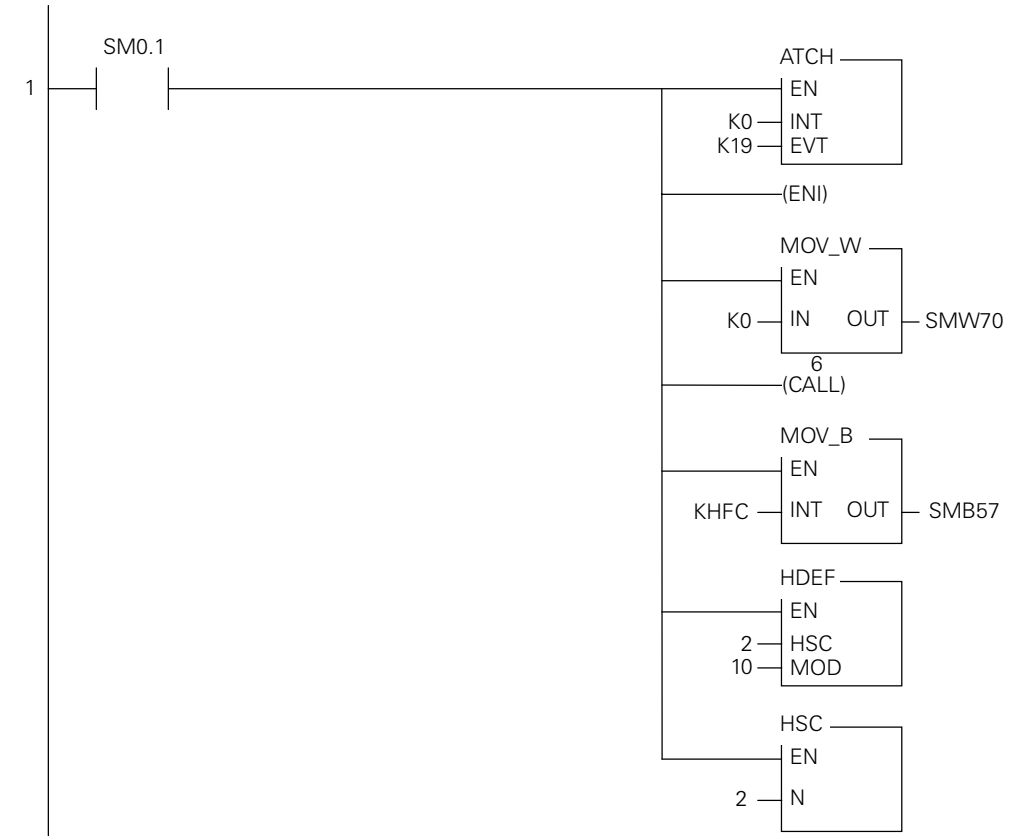
输入：	I0.0——I0.7	以度为单位的定位角（对偶码）
	I1.0	“电机启动”按钮“START”
	I1.1	“电机停止”按钮“STOP”
	I1.2	传感器信号，A 路
	I1.3	传感器信号，B 路
	I1.4	“设置/取消参考点”按钮（确认开关）
	I1.5	选择旋转方向的开关
输出：	Q0.0	脉冲输出
	Q0.2	旋转方向信号
	Q1.0	操作模式的显示
	Q1.1	定位错误的显示
标志位：	M0.1	电机运转标志位
	M0.2	锁定标志位
	M0.3	参考点标志位
	M0.4	完成第一次定位标志
	M1.1	T1 等待时间到标志位
	MD8，MD12	计算步数时的辅助内存单元
	M20.0	脉冲输出结束标志位
	MW25	错误定位计数器
精度：	AC0	允许偏差的下限
	AC1	允许偏差的上限
	AC2	设定值
	AC3	辅助寄存器

本程序的长度为 310 个字。

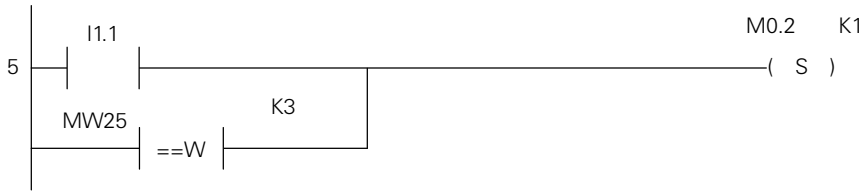
LAD(S7-MicroDOS)	STL(IEC)

// 主程序

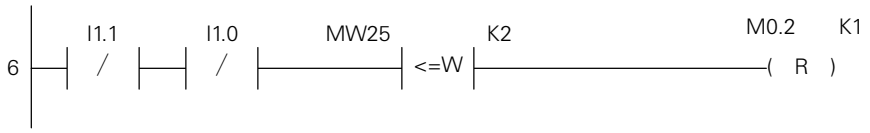
// 初始化




```
LD      SM0.1       // 仅首次扫描时 SM0.1 才为 1
ATCH    0，19       // 把中断程序 0 分配给中断事件 19（PLS0 脉冲串终止）
ENI                 // 允许中断
MOVW    0，SMW70    // 脉冲宽度=0（脉宽调制）
CALL    6           // 在子程序 6 中进行初始化

// 高速记数器 HSC2
MOVB    16#FC，SMB57  // 置 HSC2 的控制字节
HDEF    2，10         // 置 HSC2 模式 10：外部复位，A/B 计数器
HSC     2             // 激活 HSC2
```

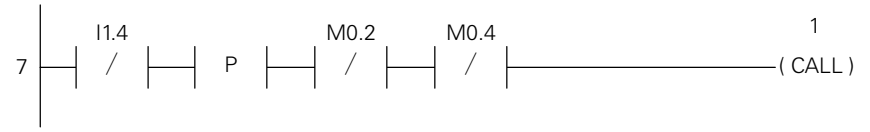
// 定位速度		
		
LDW=	MW25，0	// 若没有错误定位
MOVW	200，SMW68	// 则高速定位（T=200 μs）
// 设置逆时针旋转		
		
LDN	M0.1	// 若电机停止（M0.1=0）
A	I1.5	// 且按下旋转方向开关（I1.5=1）
S	Q0.2，1	// 则逆时针旋转（Q0.2=0）
// 设置顺时针旋转		
		
LDN	M0.1	// 若电机停止（M0.1=0）
AN	I1.5	// 且未按旋转方向开关（I1.5=0）
R	Q0.2，1	// 则顺时针旋转（Q0.2=0）
// 锁定		
		
LD	I1.1	// 若按“电机停止（stop）”钮
OW=	MW25，3	// 或有 3 个错误定位
S	M0.2，1	// 则激活锁定（M0.2=1）

// 解除锁定



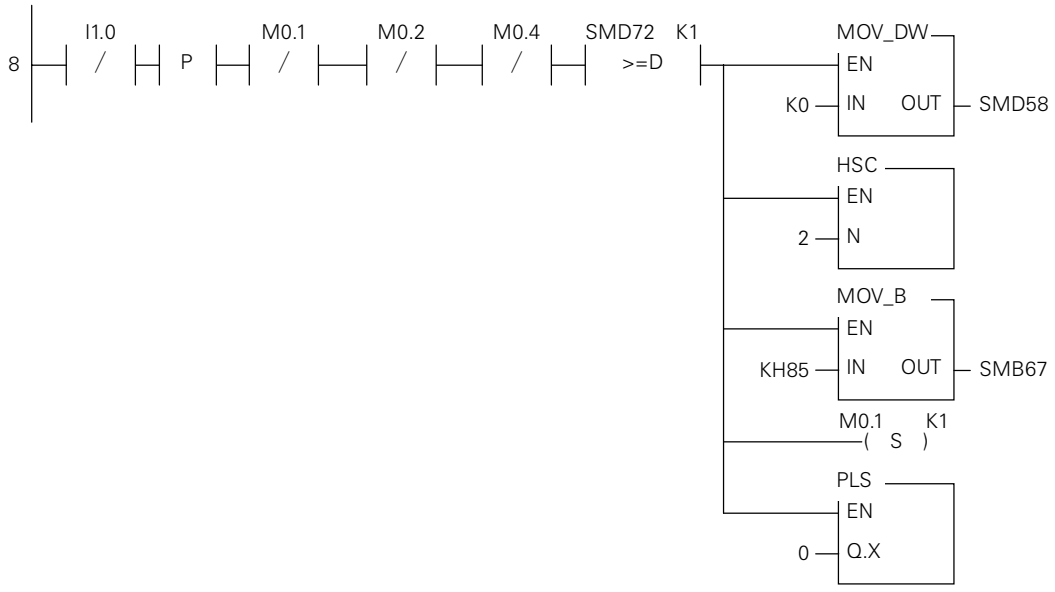
LDN	I1.1	// 若未按电机停止钮“STOP”(I1.1=0)
AN	I1.0	// 且未按电机起动钮“START”(I1.0=0)
AW<=	MW25，2	// 且<2 个错误定位
R	M0.2，1	// 则解除锁定

// 指定操作模式（检索参考/定位）



LD	I1.4	// 若按下“设置/删除参考点”按钮（I1.4=1）
EU		// 且上向沿
AN	M0.2	// 且无锁定（M0.2=0）
AN	M0.4	// 且无定位（M0.4=0）
CALL	1	// 则调用子程序 1 指定操作模式

// 起动电机

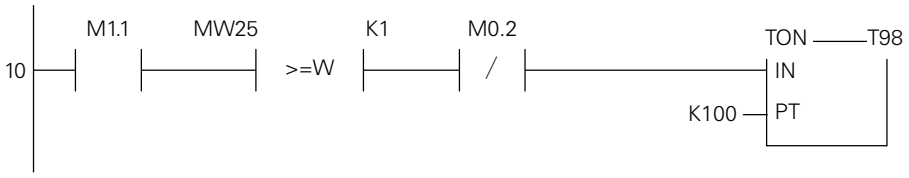


LD	I1.0	// 若按“电机起动”按钮“START”（I1.0=1）
EU		// 且上升沿
AN	M0.1	// 且电机在停止状态（M0.1=0）
AN	M0.2	// 且无联锁（M0.2=0）
AN	M0.4	// 且无定位控制（M0.4=0）
AD>=	SMD72，1	// 且步数>=1，则
MOVD	0，SMD58	// 置 HSC2 的起始值为 0
HSC	2	// 起动 HSC2
MOVB	16#85，SMB67	// 激活脉冲输出功能 PTO0（即置 PTO0 的控制位）
S	M0.1，1	// “电机运转”标志置位（M0.1=1）
PLS	0	// 起动输出端 Q0.0 输出脉冲

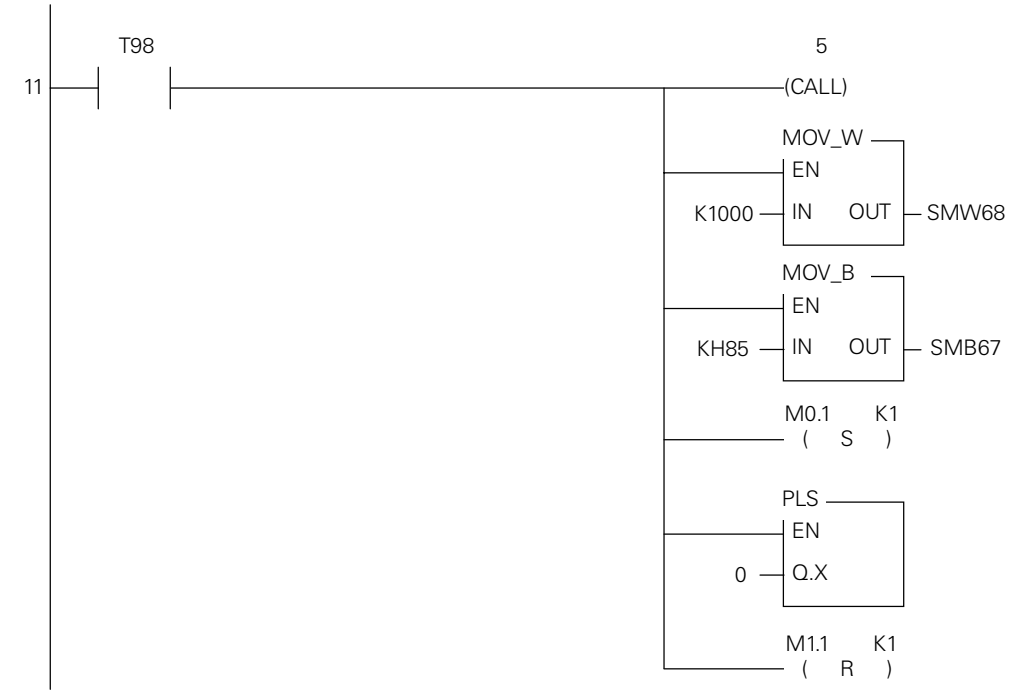
// 定位

LD	M0.3	// 若“定位”操作模式（M0.3=1）
AN	M0.4	// 且尚未定位（M0.4=0）
CALL	2	// 则调用子程序 2 定位

// 位置校正



LD	M1.1	// 若 T1 到时（M1.1=1）
AW>=	MW25，1	// 且检测出错误定位（MW25≥1）
AN	M0.2	// 且未激活锁定（M0.2=0）
TON	T98，100	// 则起动等待定时器 T2（1s）



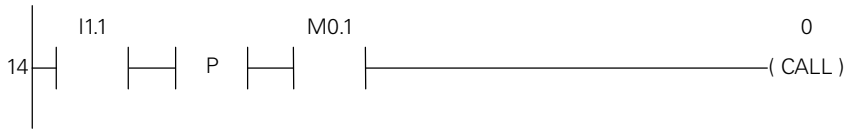
LD	T98	// 若 T2 到时（T98=1），则
CALL	5	// 调用子程序 5 计算校正步数
MOVW	1000，SMW68	// 用 1kHz 进行位置校正
MOVB	16#85，SMB67	// 激活 PTO0（即置 PTO0 的控制位）
S	M0.1，1	// 设置“电机运转”标志（M0.1=1）
PLS	0	// 起动 Q0.0 输出脉冲
R	M1.1，1	//T1 复位（M1.1=0）

// 位置监视

LD	M20.0	// 若脉冲输出结束（M20.0=1）
AN	M0.2	// 且未激活锁定（M0.2=0）
TON	T97，50	// 则起动等待定时器 T1（500ms）

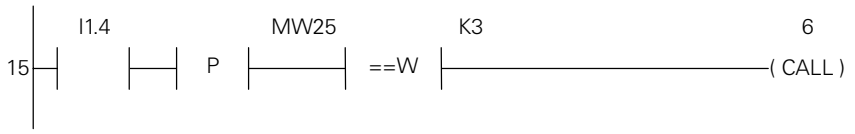

```
LD      T97     // 若 T1 到时（T97=1），则
S       M1.1，1 // T1 标志置位（M1.1=1）
CALL    4       // 在子程序 4 中调用位置监视
R       M20.0，1// 脉冲输出结束标志位复位（M20.0=0）
```

// 电机停止




```
LD      I1.1    // 若按下“电机停止”钮“STOP”（I1.1=1）
EU              // 且上升沿
A       M0.1    // 且电机在运转（M0.1=1）
CALL    0       // 则调用子程序 0 停止电机
```

// 3 次定位失败之后的错误确认

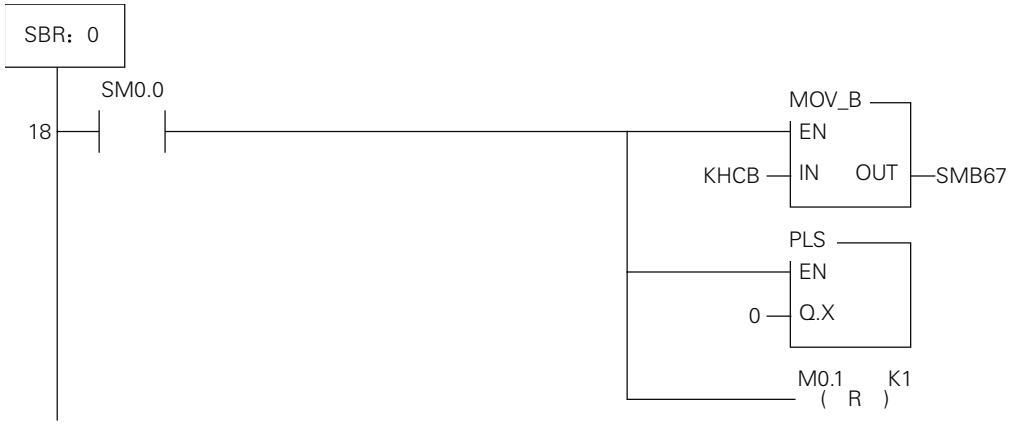



```
LD      I1.4    // 若按下确认钮（I1.4=1）
EU              // 且上升沿
AW=     MW25，3 // 且有 3 次定位失败
CALL    6       // 则调用子程序 6 返回初始状态
```

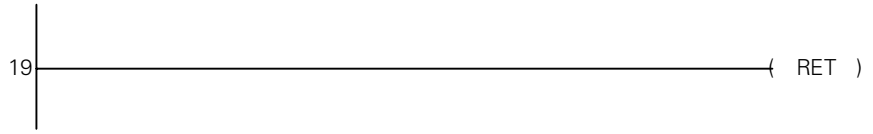

```
MEND            // 主程序结束
```

子程序

```
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
// 子程序 0“停止电机”
```




```
SBR     0          // 子程序 0
LD      SM0.0      // SM0.0 总是 1
MOVB    16#CB，SMB67  // 激活脉冲宽度调制（即置 PTO0 的控制位）
PLS     0          // Q0.0 停止输出脉冲
R       M0.1，1    // 对“电机运转”标志复位（M0.1=0）
```

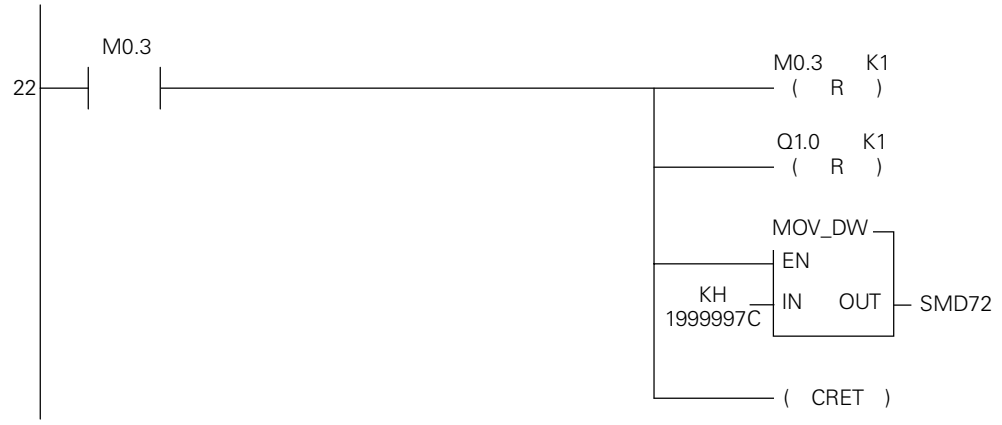



```
RET                // 子程序 0 结束
```

```
// 子程序 1“指定操作模式”
```

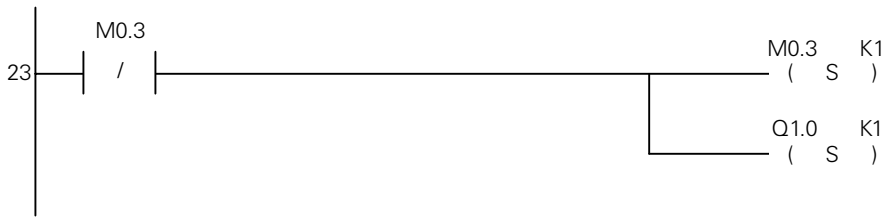

```
SBR     1          // 子程序 1
LD      M0.1       // 若电机运转（M0.1=1）
CALL    0          // 则调用子程序 0 停止电机
```

```
// 开始检索参考点
```

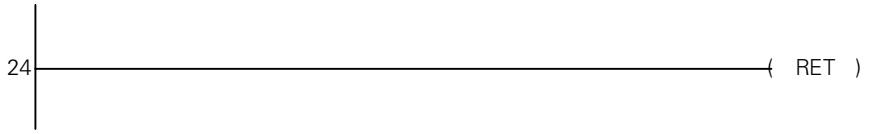



```
LD      M0.3              // 若“定位”标志激活（0.3=1），则
R       M0.3，1           // 参考点标志复位（M0.3=0）
R       Q1.0，1           // 删除“定位激活”信号（Q1.0=0）
MOVD    16#1999997C，SMD72 // 为新的参考点设置最大脉冲数
CRET                      // 条件返回
```

```
// 请求定位控制
```

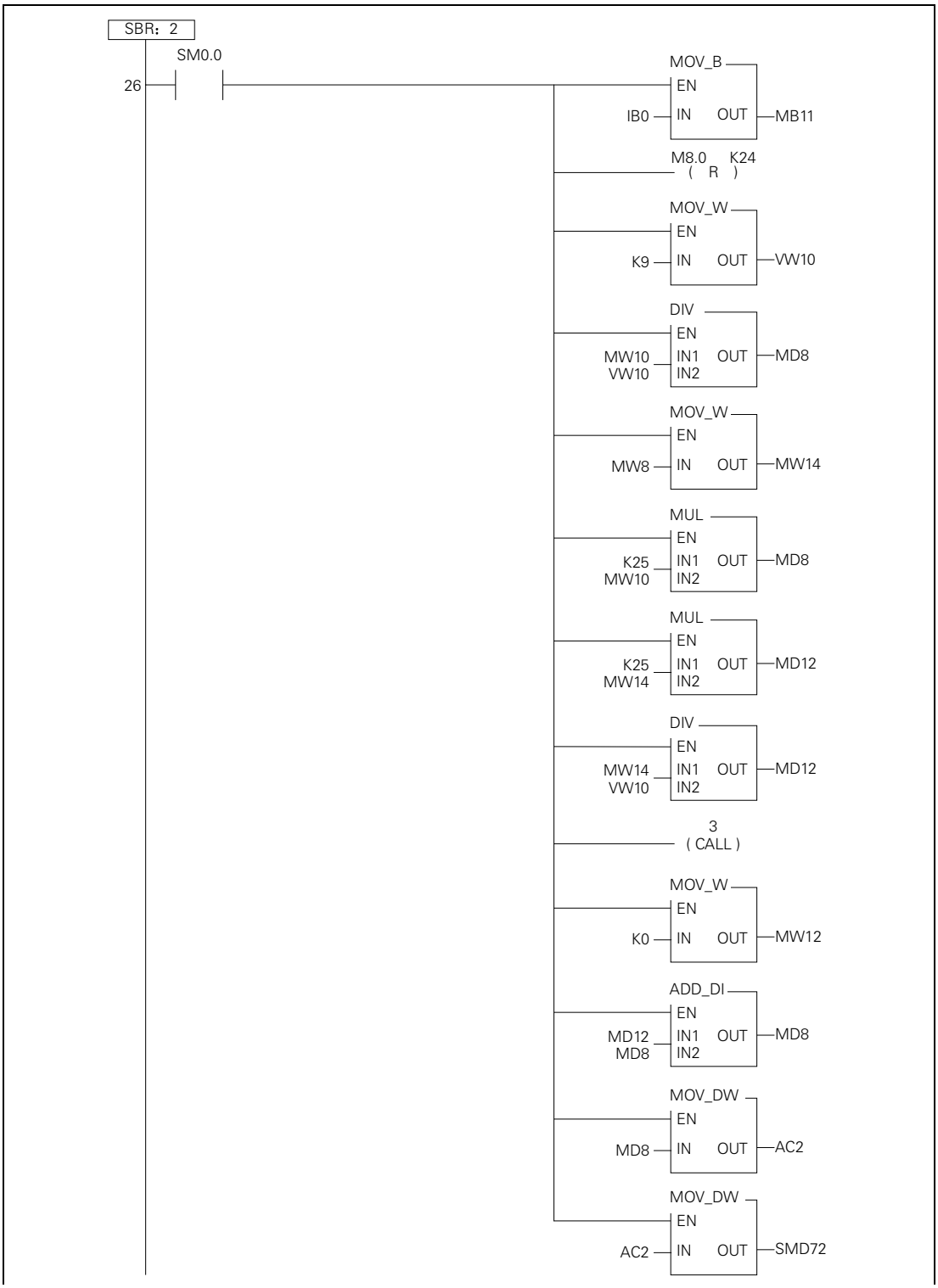



```
LDN     M0.3              // 若未设置参考点（M0.3=0），则
S       M0.3，1           // 参考点标志位置位（M0.3=1）
S       Q1.0，1           // 输出“定位激活”信号（Q1.0=1）
```

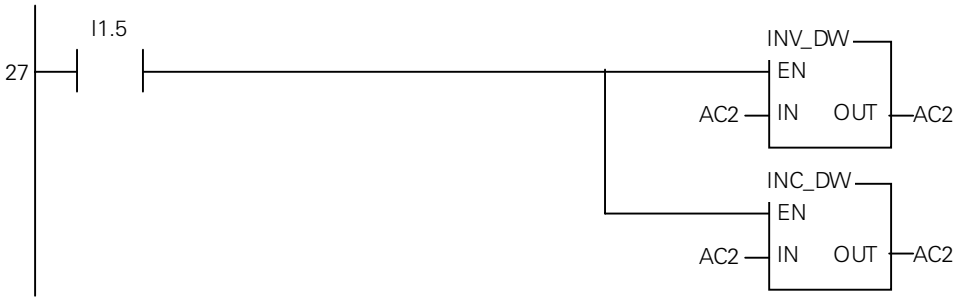



```
RET                       // 子程序 1 结束
```

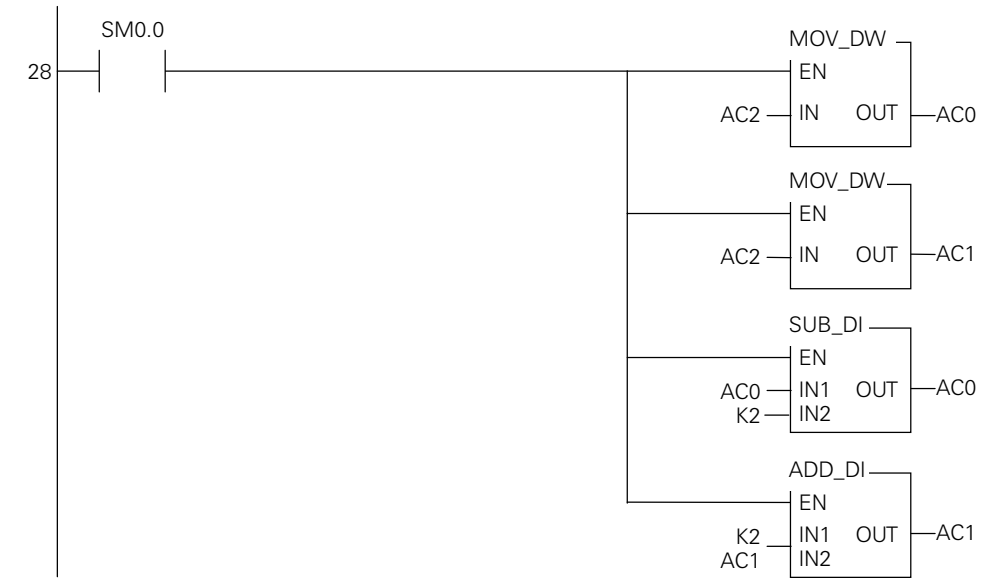
```
// 子程序 2“计算步数，允许偏差极限”
```


SBR：2
SM0.0
26
MOV_B
EN
IB0
IN
OUT
MB11
M8.0
K24
(R)
MOV_W
EN
K9
IN
OUT
VW10
DIV
EN
MW10
VW10
IN1
IN2
OUT
MD8
MOV_W
EN
MW8
IN
OUT
MW14
MUL
EN
K25
MW10
IN1
IN2
OUT
MD8
MUL
EN
K25
MW14
IN1
IN2
OUT
MD12
DIV
EN
MW14
VW10
IN1
IN2
OUT
MD12
3
(CALL)
MOV_W
EN
K0
IN
OUT
MW12
ADD_DI
EN
MD12
MD8
IN1
IN2
OUT
MD8
MOV_DW
EN
MD8
IN
OUT
AC2
MOV_DW
EN
AC2
IN
OUT
SMD72

```
SBR2                    // 子程序 2
LD      SM0.0           // SM0.0 总为 1
MOVB    IB0，MB11       // 把预设定位角从输入字节 IB0 复制到 MD8 的最低有效字节 MB11
R       M8.0，24        // MB8 至 MB10 清零
MOVW    K9，VW10        // 把 9 置入 VW10
DIV     VW10，MD8       // 角度/9=q1+r1（q1=商，r1=余数）
MOVW    MW8，MW14       // 把 r1（余数）存入 MD12
MUL     25，MD8         // q1×25=MD8
MUL     25，MD12        // r1×25=MD12
DIV     VW10，MD12      // r1×25/9=q2+r2（q2=商，r2=余数）
CALL    3               // 在子程序 3 中 4 舍 5 入步数
MOVW    0，MW12         // 删除 r2
+D      MD12，MD8       // 把步数写入 MD8（MD12+MD8=MD8）
MOVD    MD8，AC2        // 步数=设定值（把步数 MD8 存入累加寄存器 AC2）
MOVD    AC2，SMD72      // 把步数存入 SMD72
```



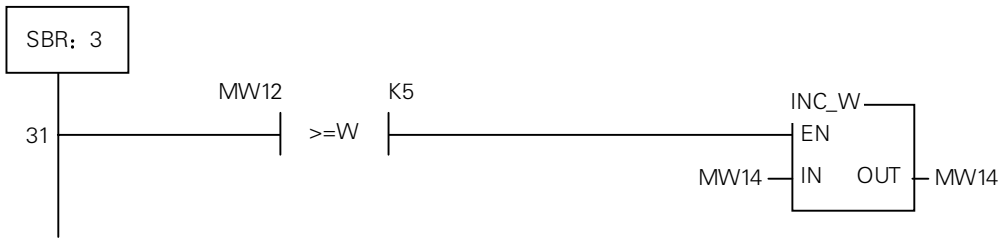

```
LD      I1.5            // 若按下逆时针旋转钮（I1.5=1），则
INVD    AC2             // AC2 取反
INCD    AC2             // AC2+1=AC2
```



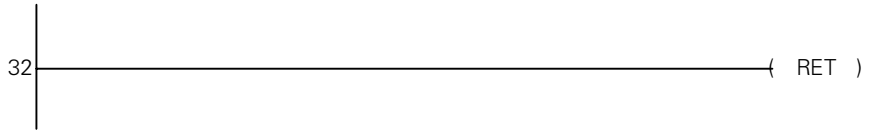

```
LD      SM0.0       // SM0.0 总为 1
MOVD    AC2，AC0    // AC2 存入 AC0
MOVD    AC2，AC1    // AC2 存入 AC1
-D      2，AC0      // 最低限值（AC2-2=AC0）
+D      2，AC1      // 最高限值（AC2+2=AC1）
```


```
RET                 // 子程序 2 结束
```

```
// 子程序 3 “4 舍 5 入步数”
```

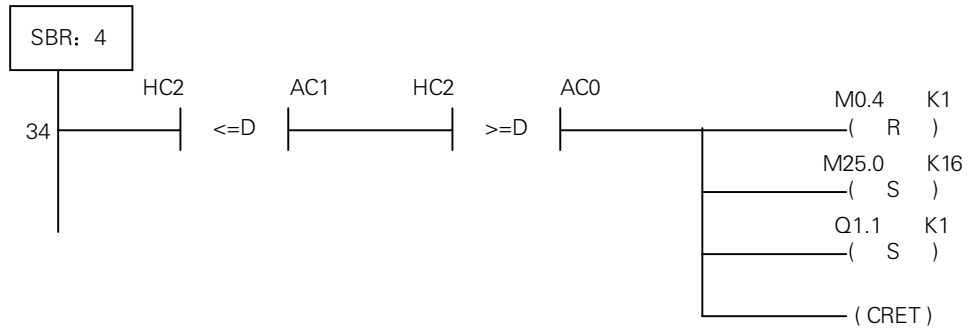



```
SBR     3           // 子程序 3
LDW>=   MW12，5     // 如果 r2>=5/9
INCW    MW14        // 则步数增加 1
```

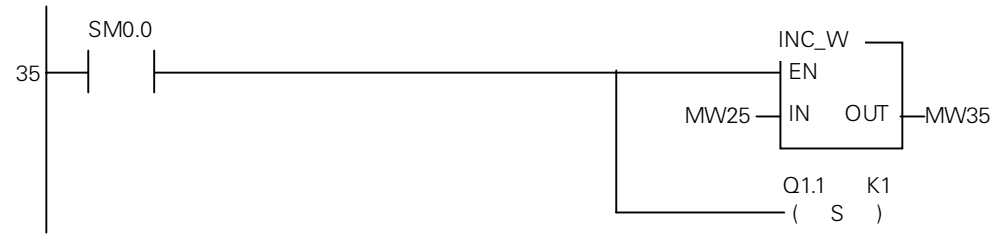



```
RET                                    // 子程序 3 结束
```

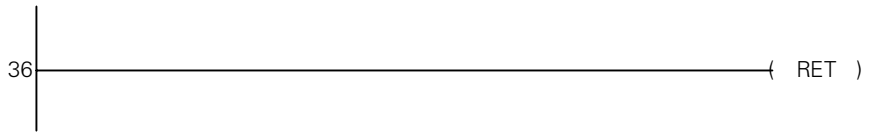
// 子程序 4“位置监视”




```
SBR4                                   // 子程序 4
LDD<=       HC2，AC1
AD>=        HC2，AC0                   // 若当前值在限值范围内（即 ACO≤HC2≤AC1），则
R           M0.4，1                    // 第 1 个位置标志复位（M0.4=0）
R           M25.0，16                  // 错误定位计数器复位
R           Q1.1，1                    // 删除错误定位显示（Q1.1=0）
CRET                                   // 条件返回，否则
```



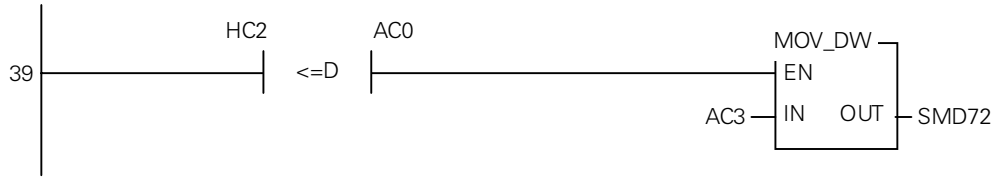

```
LD          SM0.0                      // 错误定位
INCW        MW25                       // 错误定位计数器加 1
S           Q1.1，1                    // 显示错误定位
```



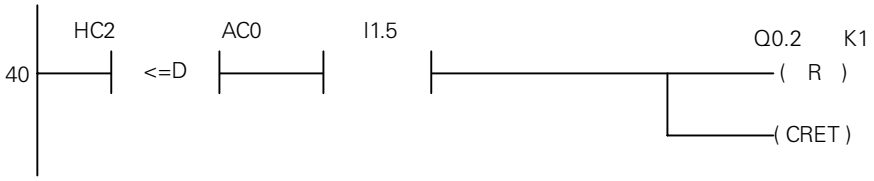

```
RET                                    // 子程序 4 结束
```

// 子程序 5“计算校正步数”

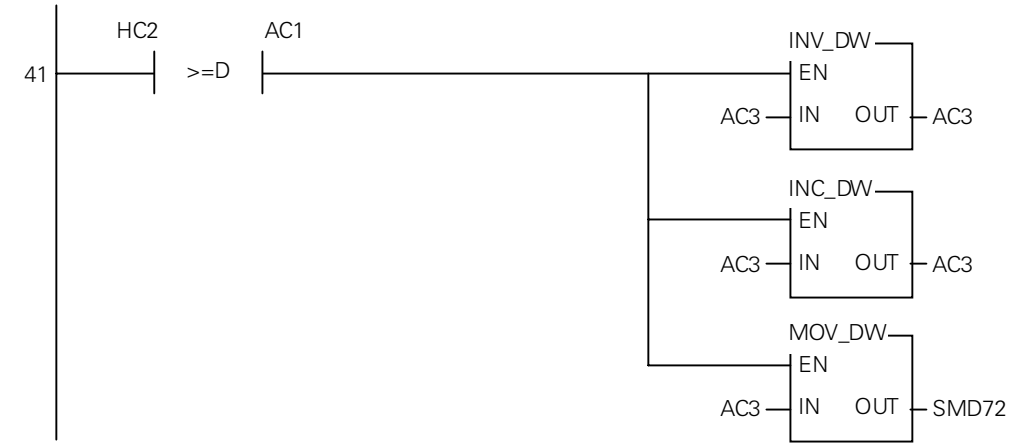
SBR	5	// 子程序 5
LD	SM0.0	// SM0.0 总为 1
MOVD	AC2，AC3	// AC2 存入 AC3
-D	HC2，AC3	// 设定值-实际值=AC3



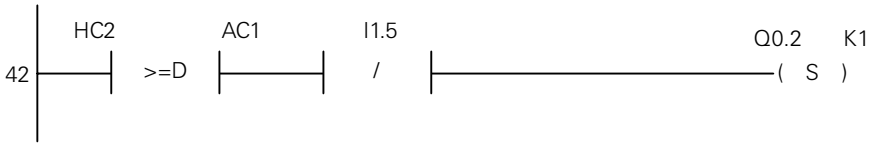
LDD<=	HC2，AC0	// 若实际值<设定值 AC0
MOVD	AC3，SMD72	// 则步数=设定值－实际值



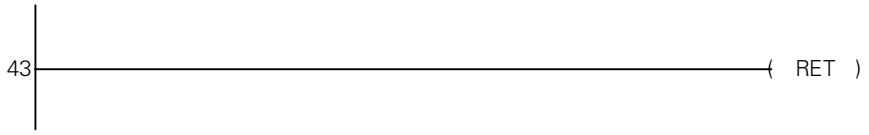
LDD<=	HC2，AC0	// 若实际值<设定值 AC0
A	I1.5	// 且逆时针方向旋转（I1.5=1）
R	Q0.2，1	// 则顺时针方向校正
CRET		// 条件返回



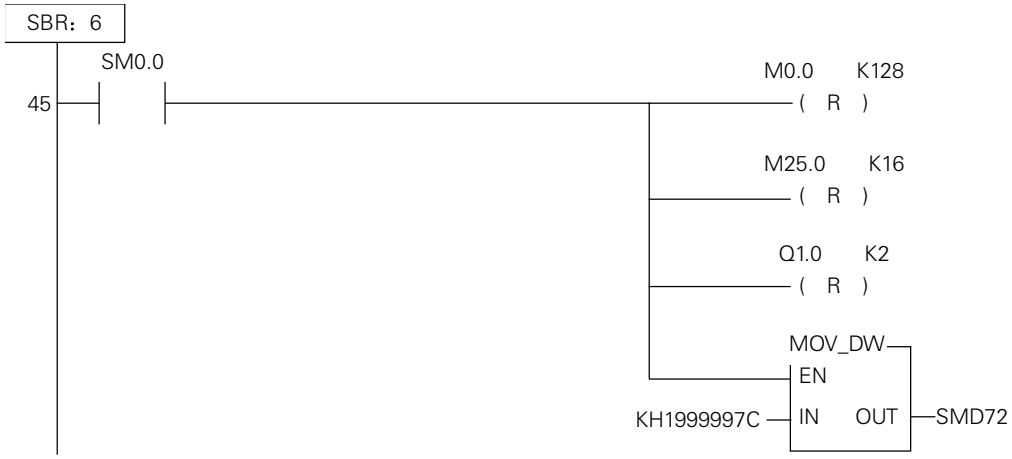
```
LDD>=   HC2，AC1        // 若实际值>设定值 AC1，则
INVD    AC3             // AC3 取反
INCD    AC3             // AC3+1=AC3
MOVD    AC3，SMD72      // 步数=实际值－设定值
```



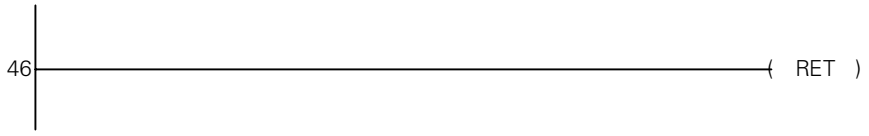

```
LDD>=   HC2，AC1        // 若实际值>设定值 AC1
AN      I1.5            // 且顺时针方向旋转时（方向开关 I1.5=0）
S       Q0.2，1         // 则逆时针方向校正
```




```
RET                     // 子程序 5 结束

// 子程序 6“程序开始和错误确认之后的初始化”
```



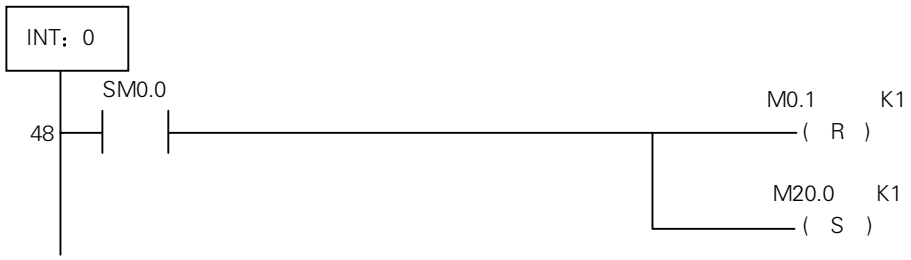

```
SBR     6               // 子程序 6
LD      SM0.0           // SM0.0 总为 1
R       M0.0，128       // M0.0 至 M15.7 复位
R       M25.0，16       // 错误定位计数器复位
R       Q1.0，2         // 操作模式和错误定位显示复位（Q1.0=0，Q1.1=0）
MOVD    16#1999997C，SMD72  // 搜索参考点的脉冲计数（PTO0）
```



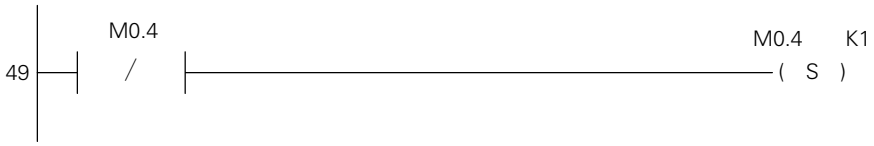

```
RET                     // 子程序 6 结束
```

中断程序

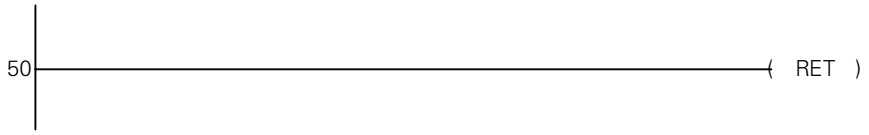
```
//*********************************************
// 中断 0“脉冲输出终止”
```




```
INT     0       // 中断程序 0
LD      SM0.0   // SM0.0 总是 1
R       M0.1，1 // “电机运转”标志复位（M0.1=0）
S       M20.0，1// “脉冲输出结束”标志置位（M20.0=1）
```




```
LDN     M0.4    // 第 1 次定位控制之后
S       M0.4，1 // 设置相应标志信号（M0.4=1）
```




```
RETI            // 中断程序 0 结束
```

31 用定器产生断开延迟、脉冲和扩展脉冲

概述

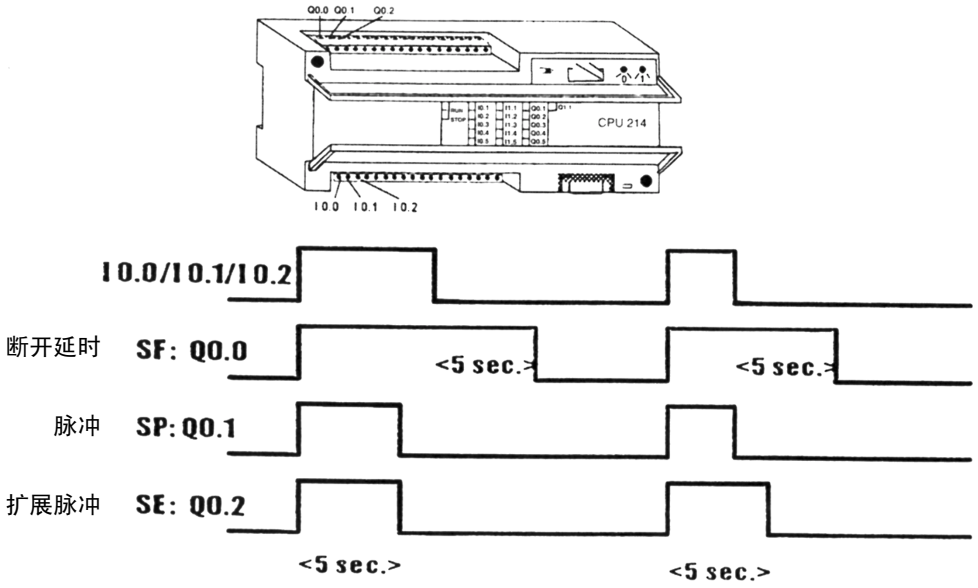
本例说明了利用 S7-200 的集成“接通延迟”（ON-Delayed）定时器，能够方便地产生断开延迟（OFF-Delay）、脉冲（Pulse）及扩展脉冲（Extended Pulse）。

为了在输出端 Q0.0 得到断开延迟信号，Q0.0 端的输出信号的置位时间要比 I0.0 端的输入信号长一段定时器的时间。

为了在输出端 Q0.1 得到脉冲信号，I0.1 端的输入信号被置位之后，信号会在输出端 Q0.1 停留一段定时器的时间；但是，如果输入 I0.1 被复位，那么输出端 Q0.1 脉冲信号也将被复位。

为了在输出端 Q0.2 得到扩展脉冲信号，一旦输入 I0.2 已经置位，无论输入 I0.2 是否复位，那么在预置定时器时间内 Q0.2 端输出信号将一直处于置位状态。

例图



程序和注释

下列程序分为 3 部分，每部分都相互独立，用来实现断开延迟（OFF-Delay）、脉冲（Pulse）和扩展脉冲（Extended Pulse）。

一、断开延迟（OFF-Delay）

当接通输入 I0.0 时，输出 Q0.0 被置位。如果输入 I0.0 被复位（下降沿），则启动定时器 T33，运行 5 秒钟后，定时器 T33 置位，同时使标志位 M0.0 和输出 Q0.0 复位。

二、脉冲（Pulse）

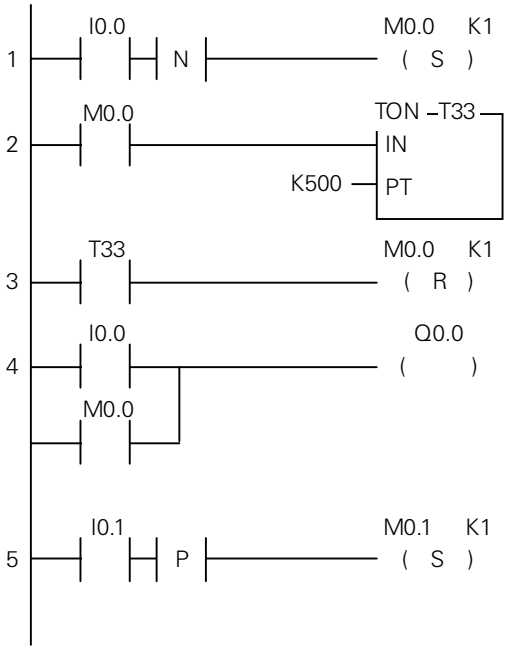
当接通输入 I0.1 时，输出 Q0.1 和标志位 M0.1 被置位。通过对标志位 M0.1 置位使定时器 T34 启动，运行 5 秒钟后或输入 I0.1 复位，就立即使输出 Q0.1 复位。

三、扩展脉冲（Extended Pulse）

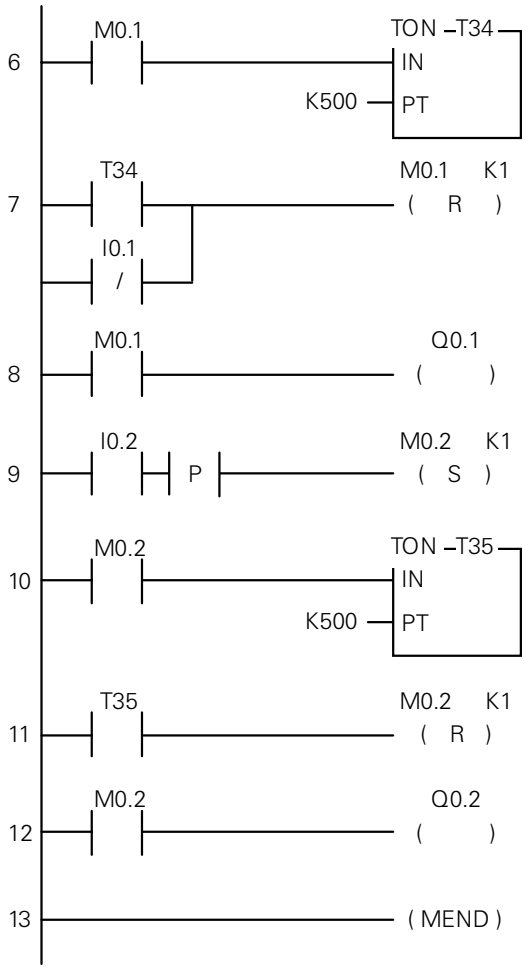
当接通输入 I0.2 时，输出 Q0.2 和标志位 M0.2 被置位。通过对标志位 M0.2 置位，使定时器 T35 启动，运行 5 秒钟后，立即使输出 Q0.2 复位。

LAD(S7-MicroDOS)	STL(IEC)

// 主程序




```
// “断开延迟”SF
LD      I0.0            // 若在 I0.0 的下降沿
ED                      //
S       M0.0，1         // 则为 SF 设置标志位（M0.0=1）

LD      M0.0            // 若标志位已被激活（M0.0=1）

TON     T33，500        // 则启动定时器 T33
LD      T33             // 若 5 秒定时到
R       M0.0，1         // 则对标志位复位（M0.0=0）

LD      I0.0            // 若有输入 I0.0（I0.0=1）
O       M0.0            // 或延迟已激活（M0.0=1）
=       Q0.0            则对输出 Q0.0 置位（Q0.0=1）

// 脉冲 SP
LD      I0.1            // 若在 I0.1 的上升沿
EU                      //
S       M0.1，1         // 则为 SP 设置标置位（M0.1=1）
```




```
LD      M0.1          // 若标志位已激活（M0.1=1）
TON     T34，500      // 则启动定时器 T34

LD      T34           // 若 5 秒定时到
ON      I0.1          // 或无输入（I0.1=0）
R       M0.1，1       // 则对标志位复位（M0.1=0）

LD      M0.1          // 将标志位 M0.1 复制到输出
                         端 Q0.1
=       Q0.1          //


// “扩展脉冲”SE
LD      I0.2          // 若在 I0.2 的上升沿
EU                    //
S       M0.2，1       // 则为 SE 设置标志位
                         （M0.2=1）

LD      M0.2          // 若标志位已激活
                         （M0.2=1）
TON     T35，500      // 则启动 5 秒定时器 T35
LD      T35           // 若 5 秒定时到

R       M0.2，1       // 则对标志位复位
                         （M0.2=0）
LD      M0.2          // 将标志位 M0.2 复制到输出端
=       Q0.2          // Q0.2

MEND                  // 主程序结束
```

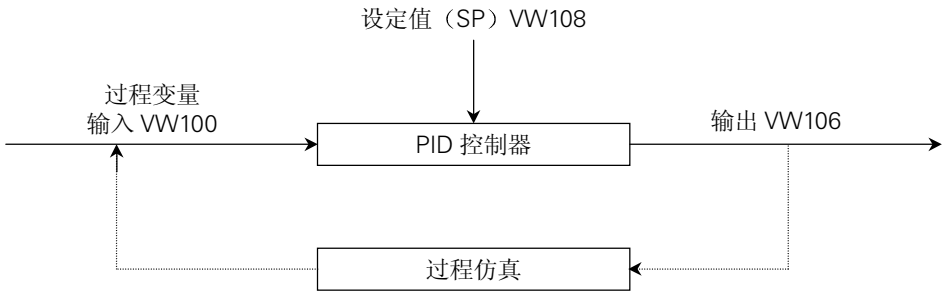
 西门子 版权所有

32 用 S7-200 实现 PID 控制

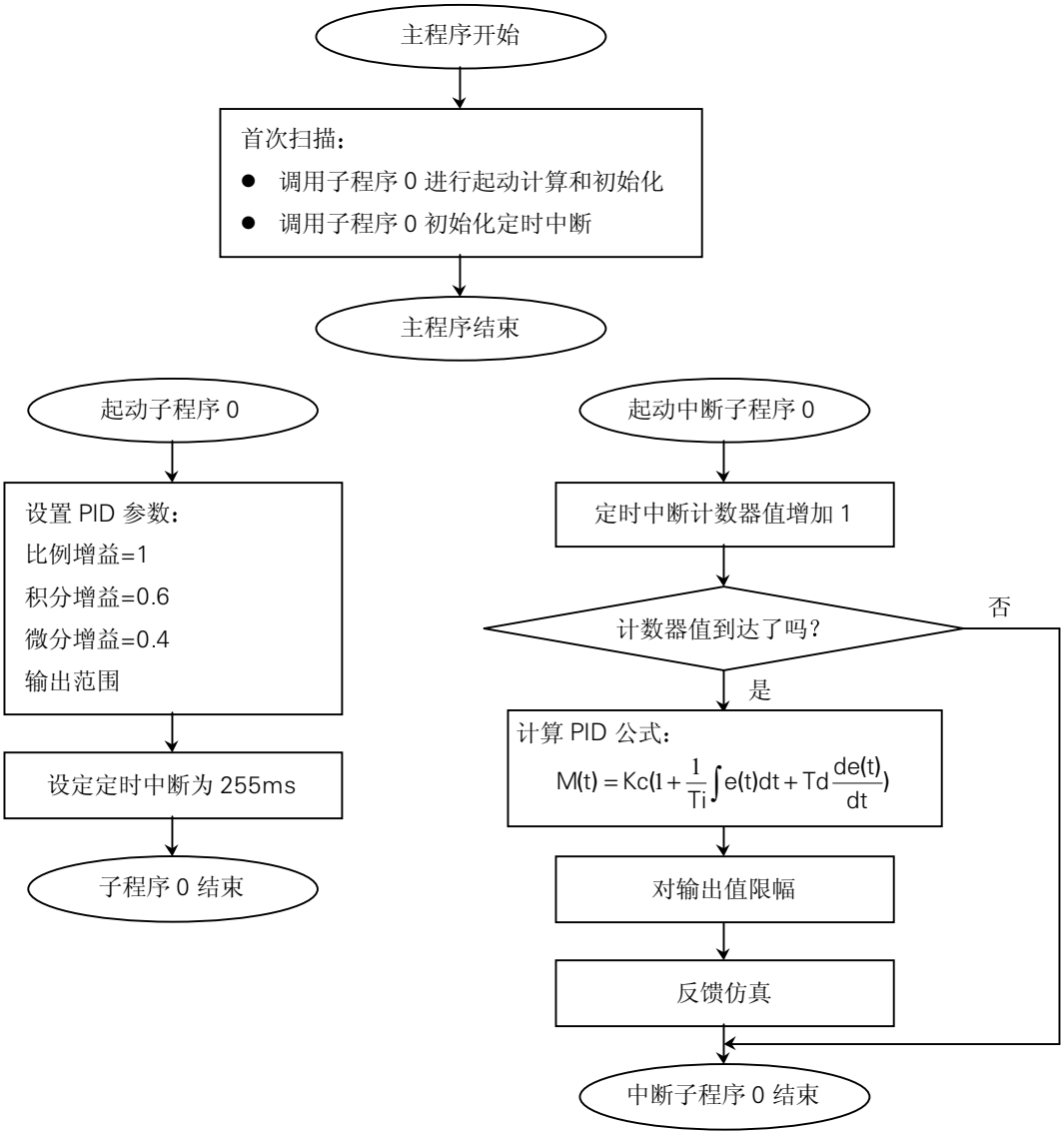
概述

本例描述了用 S7-200 实现 PID 控制功能。这个程序是一个带过程仿真的独立执行的 PID 例子，它很容易修改后与模拟模块 EM235 一起使用。

例图



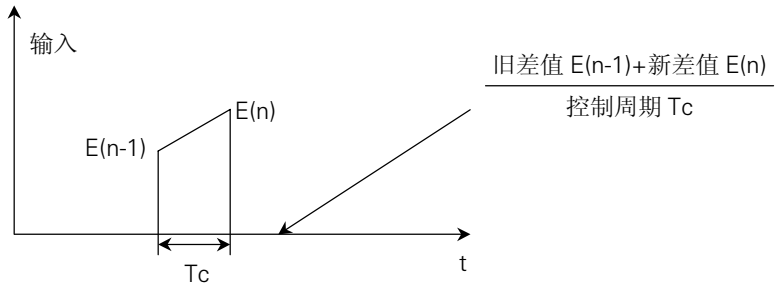
程序结构图


主程序开始
首次扫描：
调用子程序 0 进行起动计算和初始化
调用子程序 0 初始化定时中断
主程序结束
起动子程序 0
设置 PID 参数：
比例增益=1
积分增益=0.6
微分增益=0.4
输出范围
设定定时中断为 255ms
子程序 0 结束
起动中断子程序 0
定时中断计数器值增加 1
计数器值到达了吗？
否
是
计算 PID 公式：
M(t) = Kc(1 + 1/Ti ∫e(t)dt + Td de(t)/dt)
对输出值限幅
反馈仿真
中断子程序 0 结束

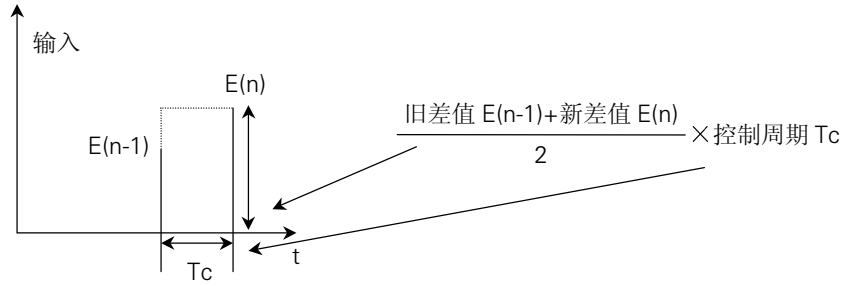
程序及注解

初始化部分将 PID 的所有值复位，并定义了计算 PID 控制器的控制周期 Tc。

计算 PID 过程中，出现了某些数字方面的问题，以及控制周期 Tc 的计算。由于扫描时间的限制，除法运算通过移位来实现（1024 近似为 1000），而未调用专门的除法子程序。微分和积分是另外 2 个比较灵敏的数学运算，采用如下公式：
微分运算：



积分运算：

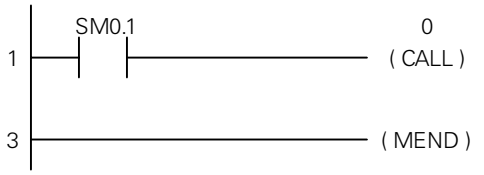


处理过程中的反馈输入变量来自仿真
程序全长 370 个字

LAD(S7-MicroDOS)	STL(IEC)

```
// 主程序
//VW100=过程变量（PV）输入值
//如果模拟模块已准备好，它将是一个模拟输入字，并假设该输入的最大范围是
//-2047~+2047
//VW102=新控制差值（运行期间计算）E(n)
//VW104=为输出值而设的计算缓冲器
//VW106=输出值
//VW108=设定值（在 CPU 214 中，此值能够从模拟调节装置 POTS 中调入）
//VW110=预置定时中断个数，如果控制周期 Tc 大于 255ms，则在此内存字中置数；
//计算 Tc 的公式：
//                    VW110×VB113=Tc（单位：ms）
//VW112=未用
//VW113=定时中断 0 的时间（单位：ms）
//注意：若 VW110 是 0，则控制周期 Tc=VW113(ms)；
//        若 VW110 大于 0，则控制周期 Tc=VW110×VB113(ms)
//
//VW114=控制周期 Tc（VW110 与 VB113 之积）
//VW116=未用
//VW118=比例增益。1…1000=0.0078~7.8，
//          注意：例如，比例增益为 1，可由 VW118=128 得到
//          注意：比例增益用于 P、I、D 部分，并与这 3 部分的和相乘。
//VW120=积分增益（1…1000=0.001~1）
//VW122=微分增益（1…1000=0.001~1）
//VW124=最大输出值（<=+2047），这是允许的最大输出值
//VW126=最小输出值（>-2047），这是允许的最小输出值
//VW128=输出上溢出，此值由启动时，VW124 值加 1 得到
//VW130=输出下溢出，此值由启动时，VW126 值减 1 得到
//VW132=旧的控制差值（缓冲值，启动时置 0）E（n-1）
//VW134=积分寄存器（缓冲值，启动时置 0）
//VW136=积分运算缓冲器
//VW138=积分值
//VW140=微分运算缓冲器
//VW142=微分值
//VW144=微分因子计算缓冲器
//VW146=微分因子，启动时，由微分增益/控制周期[1/ms]得到
//          （此值实际用于控制器运算）
//VW148 至
//VW154=用于起动运算的运算缓冲器
//VW156=定时中断计数器
//VW200 至
//VD214=用于反馈仿真
```

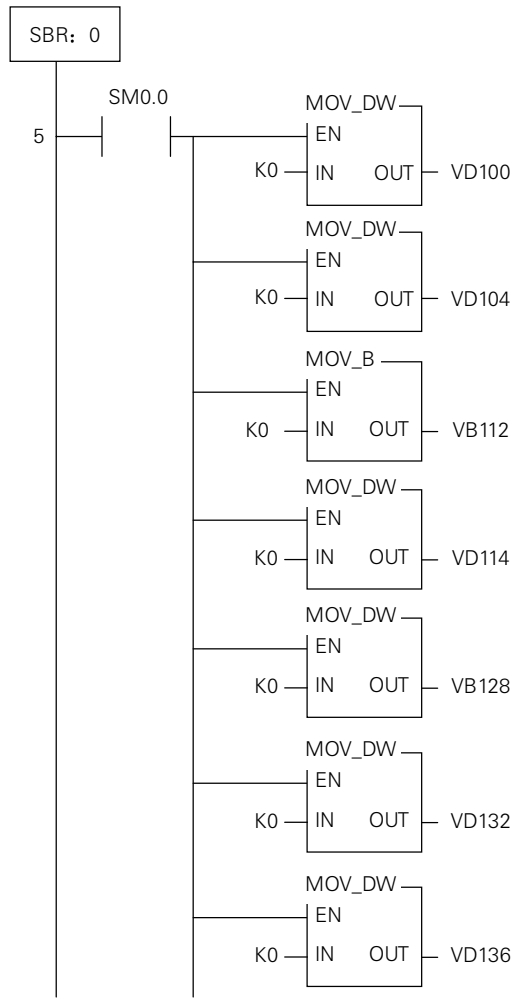
 西门子 版权所有



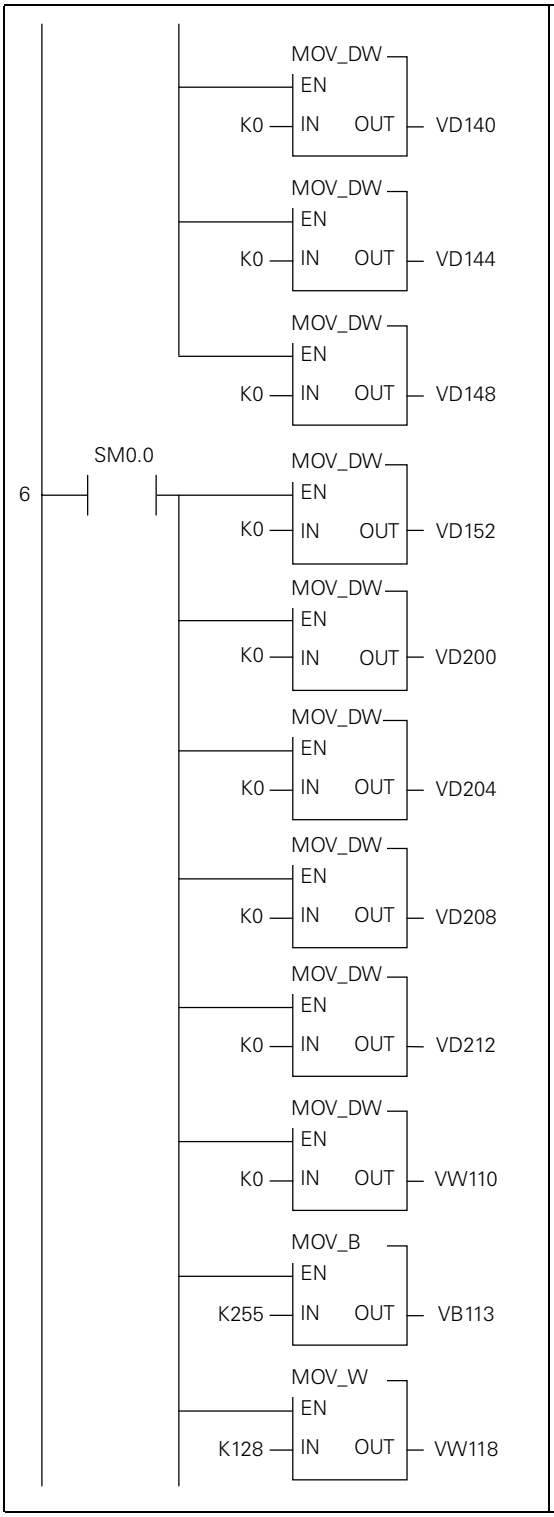

```
LD      SM0.1     // 仅首次扫描时 SM0.1 才为 1
CALL    0         // 调用子程序 0，进行起动运
                     算，初始化，设置定时中断
// 用户程序码段
MEND              // 主程序结束
```

// 子程序

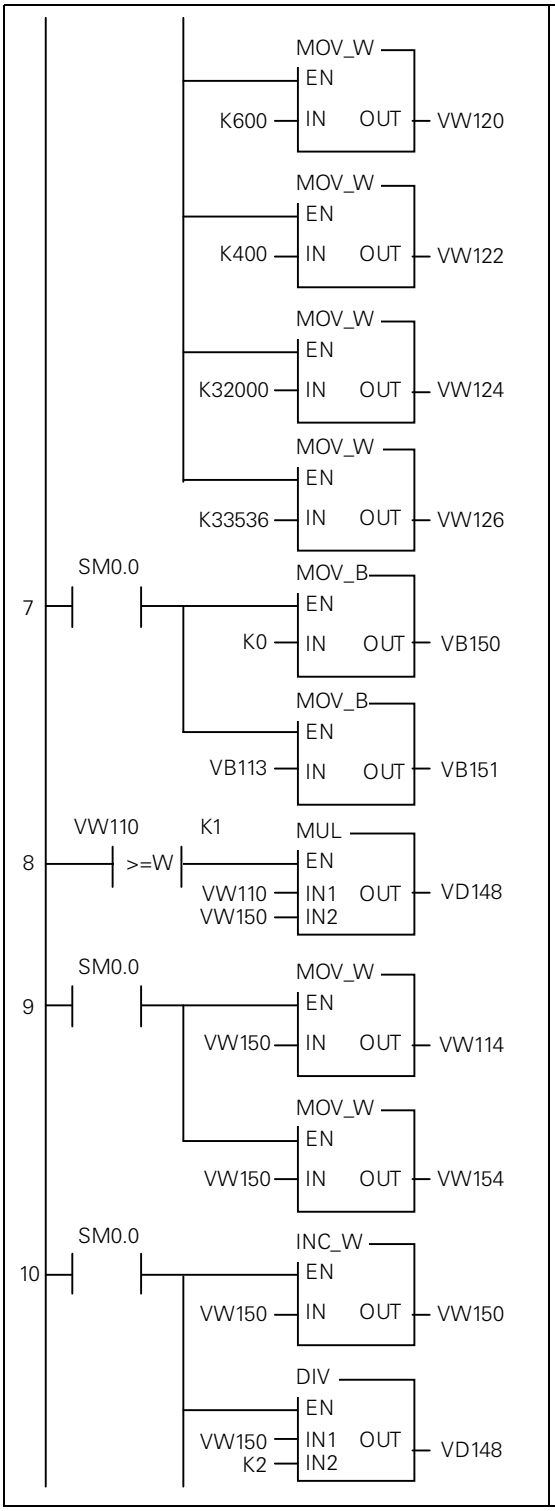
```
//*********************************************
// 功能：起动子程序
//*********************************************
```



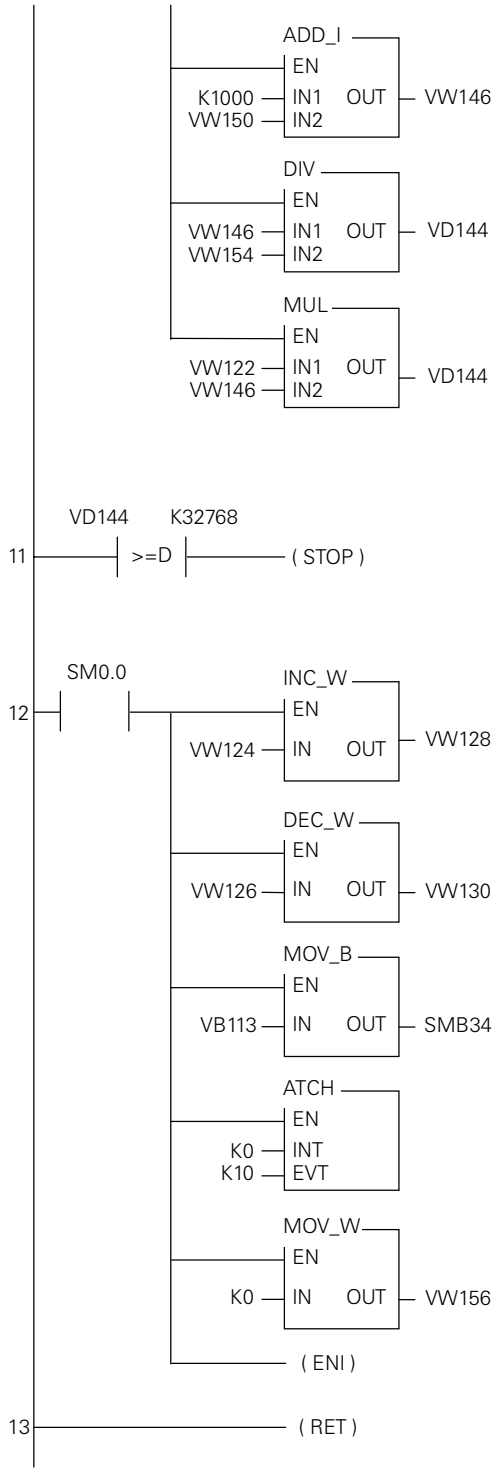

```
SBR     0         // 子程序 0
// 对各变量进行正确复位
LD      SM0.0     // SM0.0 总是为 1
MOVD    0，VD100
MOVD    0，VD104
MOVB    0，VB112
MOVD    0，VD114
MOVD    0，VB128
MOVD    0，VD132
MOVD    0，VD136
```




```
MOVD    0，VD140

MOVD    0，VD144

MOVD    0，VD148

LD      SM0.0
MOVD    0，VD152

// 对仿真变量复位
  MOVD  0，VD200

MOVD    0，VD204

MOVD    0，VD208

MOVD    0，VD212

// 设置 PID 控制器参数
  MOVW  0，VW110      // 预置定时中断个数为 0

MOVB    255，VB113    // 定时中断 0 的时间为
                      255ms

MOVW    128，VW118    // 比例增益=128/128=1
```



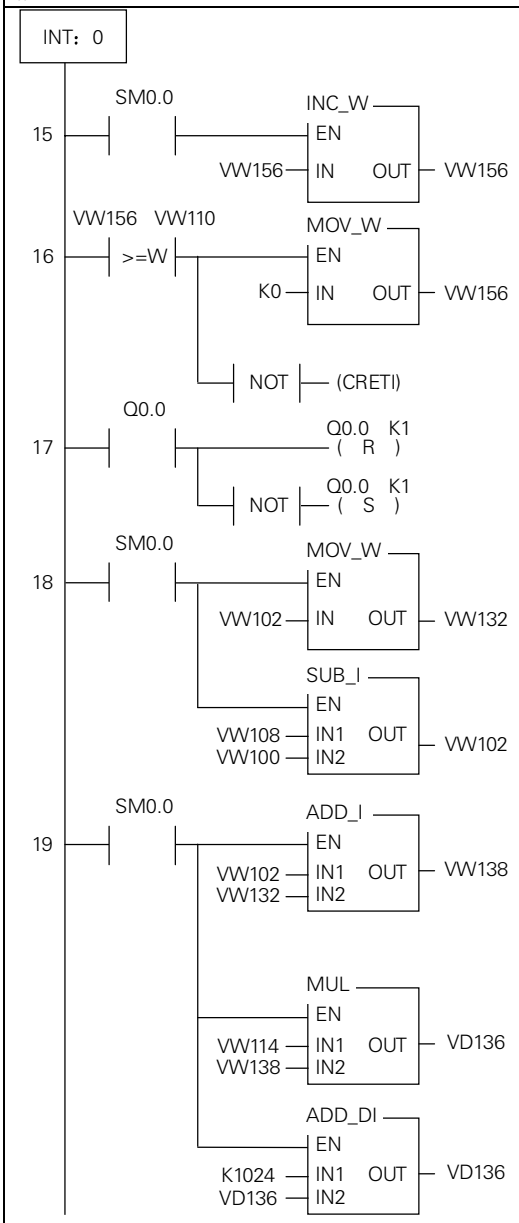

```
MOVW    600，VW120      // 积分增益=600/1024=0.6
MOVW    400，VW122      // 微分增益=400/1024=0.4
MOVW    32000，VW124    // 最大输出值是+32000
MOVW    -32000，VW126   // 最小输出值是-32000
LD      SM0.0           // SM0.0 总为 1
MOVB    0，VB150        // 运算缓冲器 VB150 清零
MOVB    VB113，VB151    // 将定时中断时间移入运算缓冲器 VB151
LDW>=   VW110，1        // 若预置定时中断个数≥1
MUL     VW110，VD148    // 则与定时中断时间相乘，求得控制周期 Tc=VD148，这是一个小于 32767 的正值(15 位为 1)
//
LD      SM0.0           // SM0.0 总为 1
MOVW    VW150，VW114    // 将控制周期 Tc(ms)存入 VW114
                        // 为微分部分而进行参数计算
                        // KD（1…1000）/Tc [1/s] 移入 VW146
                        // Tc[ms]仍留在 VW150 中
MOVW    VW150，VW154    // 将 T 控制周期 Tc 存入缓冲器 VW154
LD      SM0.0           // SM0.0 总为 1
INCW    VW150           // Tc+1，这是为了移位
DIV     2，VD148        // (Tc+1)÷2
```



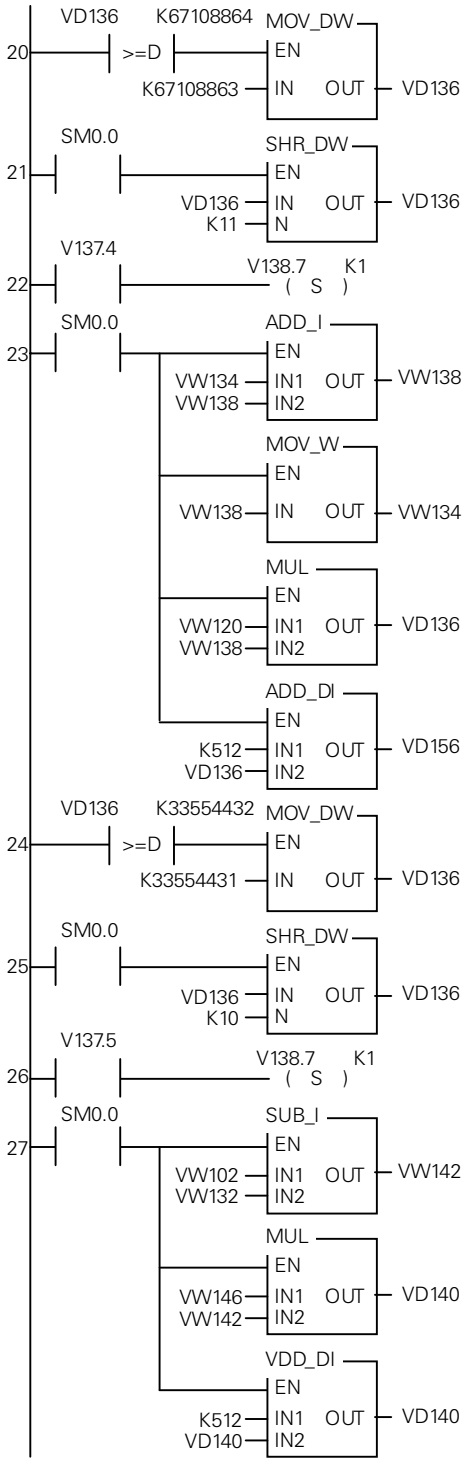

```
MOVW  1000，VW146      // 将 1000 置入微分因子缓冲器 VW146
+I    VW150，VW146     // (Tc+1)÷2 的低端字加 1000，这是为了移位
DIV   VW154，VD144     // 被 Tc 除
MUL   VW122，VD144     // 与微分增益相乘(1…1000)
LDD>= VD144，32768     // 检查是否上溢出，即计算值是否大于或等于 2 的 15 次方
                       // （16 位为 1）
STOP                   // 若有溢出，则停止‘STOP’

LD    SM0.0            // SM0.0 总为 1
MOVW  VW124，VW128     // 将最大输出值置入缓冲器 VW128
INCW  VW128            // 加 1 以得到输出上溢出值
MOVW  VW126，VW130     // 将最小输出值移入缓冲器 VW130
DECW  VW130            // 减 1 以得到输出下溢出值

//* * * * * * * * * * * * * * * * * * * * * * * *
// 初始化定时中断
MOVB  VB113，SMB34     // 设置定时中断 0 的时间
ATCH  0.10             // 指定定时中断 0（中断事件 10）调用中断程序 0
MOVW  0，VW156         // 定时中断计数器复位

ENI                    // 允许中断

RET                    // 子程序 0 结束
```

// 中断程序

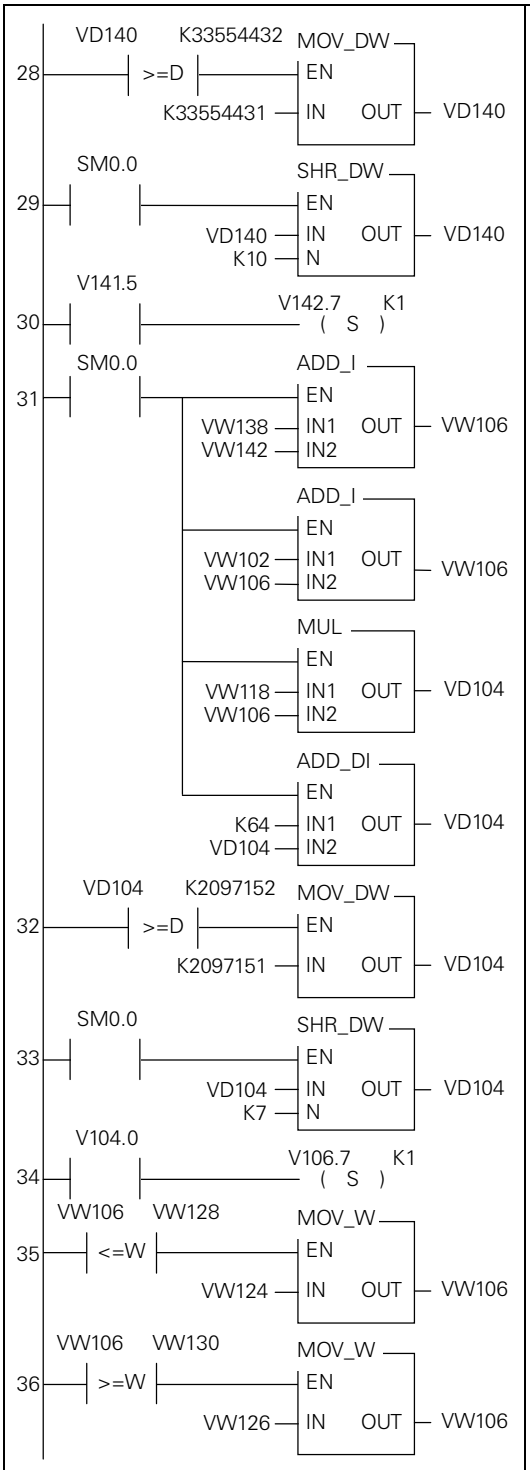
```
//* * * * * * * * * * * * * * * * * * * * * * * *
// 功能：由定时中断 0 调用的 PID 程序
//* * * * * * * * * * * * ** ** * * * * * * * *
```



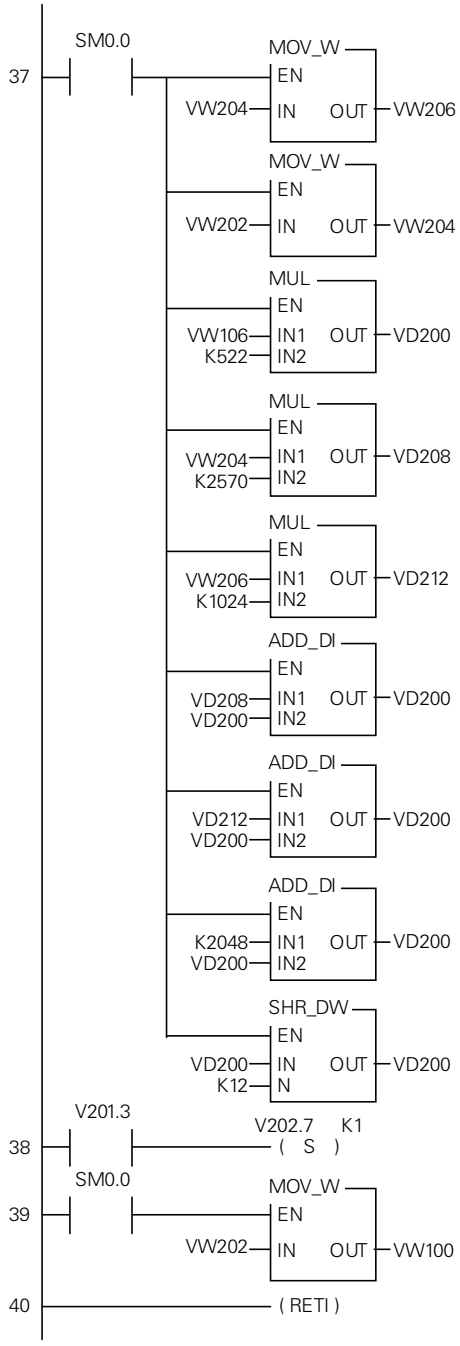

```
INT      0                  // 中断程序 0
LD       SM0.0              // SM0.0 总为 1
INCW     VW156              // 定时中断计数器加 1
LDW>=    VW156，VW110       // 若定时中断计数器数值≥预置定时中断个数
MOVW     0，VW156           // 则定时中断计数器复位；
NOT                         //
CRETI                       // 否则返回
// 下面 4 条指令是用来检查中断功能的，即在控制
// 器的扫描时间内输出端 Q0.0 的指示灯 LED 闪烁
LD       Q0.0
R        Q0.0，1
NOT
S        Q0.0，1
LD       SM0.0              // SM0.0 总为 1
MOVW     VW102，VW132       // 保存旧的控制差值 E(n-1)
MOVW     VW108，VW102       // 设定值 SP 存入 VW102
-I       VW100，VW102       // 设定值 SP-过程变量 PV=新的控制差值 E(n)
                            // E(n)存入 VW102
//* * * * * * * * * * * * * * * * * * * * * * * *
// 积分部分
LD       SM0.0              // SM0.0 总为 1
MOVW     VW102，VW138       // 将新的控制差值 E(n)置入缓冲器 VW138
                  // 注：除以 2 由随后的移位实现
MUL      VW114，VD136       // 与控制周期 Tc 相乘（因为控制周期 Tc 单位是
                            // 毫秒，乘积也许会是双字值!），即[E(n-1)+ E(n)]×Tc=VD136
+D       1024，VD136        // 为更好的移位而加 1024
```




```
LDD>= VD136，67108864   // 检查是否上溢出，即计算值是否大于或等于 2 的 26 次方
                        // （27 位为 1）
MOVD 67108863，VD136    // 若有上溢出，则取最大值 67108863
LD   SM0.0              // SM0.0 总为 1
SRD  VD136，11          // 除以 1024*2，即求得 [E(n-1)+E(n)] × Tc ÷ 2=VD136
LD   V137.4             // 注意符号位
S    V138.7，1
LD   SM0.0              // SM0.0 总为 1
+I   VW134，VW138       // 求积分部分累加值

MOVW VW138，VW134       // 将新的积分部分值存入缓冲器 VW134

MUL  VW120，VD136       // 与积分增益相乘（0…1000）

+D   512，VD136         // 为更好的移位而加 512

LDD  >=VD136，33554432
                        // 检查是否上溢出，即计算值是否大于或等于 2 的 25 次方
                        // （26 位为 1）
MOVD 33554431，VD136    // 若上溢出，则取最大值 33554431
LD   SM0.0              // SM0.0 总为 1
SRD  VD136，10          // 被 1024 除（正确值应是 1000）
LD   V137.5             // 注意符号位
S    V138.7，1          // VW138=积分值

// * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
// 微分部分
LD   SM0.0              // SM0.0 总为 1
MOVW VW102，VW142       // 把新的控制差值 E(n)存入 VW142
-I   VW132，VW142       // E(n)-E(n-1)=VW142
MUL  VW146，VD140       // 与微分因子相乘

+D   512，VD140         // 为更好的移位而加 512
```




```
LDD>=   VD140，33554432   // 检查是否上溢出，即计算值是否大于或等于 2 的 25 次方
                          // （26 位为 1）
MOVD    33554431，VD140   // 若上溢出，则取最大值 33554431
LD      SM0.0             // SM0.0 总为 1
SRD     VD140，10         // 被 1024 除（正确值应是 1000）
LD      V141.5            // 注意符号位
S       V142.7，1         // VW142=微分值
// ************************
// 比例部分及总和
LD      SM0.0             // SM0.0 总为 1
MOVW    VW138，VW106      // 积分结果送入输出缓冲器
+I      VW142，VW106      // 将微分和积分结果相加
+I      VW102，VW106      // 再加比例部分（新的控制差值 E(n)）
MUL     VW118，VD104      // 与比例增益相乘，比例增益 1 … 1000 =0.0078~7.8
+D      64，VD104         // 为更好的移位而加 64
LDD>=   VD104，2097152    // 检查是否上溢出。即计算值是否大于或等于 2 的 21 次方
                          // （22 位为 1）
MOVD    2097151，VD104    // 若上溢出，则取最大值 2097151
LD      SM0.0             // SM0.0 总为 1
SRD     VD104，7          // 被 128 除

LD      V104.0            // 注意符号位
S       V106.7，1         //VW106=控制器输出
// ********************************
// 对输出值限幅
LDW>=   VW106，VW128      // 若控制器输出值上溢出，即 VW106>=VW128
MOVW    VW124，VW106      // 则将输出置为最大允许输出，即 VW106=VW124
LDW<=   VW106，VW130      // 若控制器输出值下溢出，即 VW106≤ VW130
MOVW    VW126，VW106      // 则将输出置为最小允许输出，即 VW106=VW126
// 控制器中断程序结束，在过程仿真求反馈输入期间此程序休息
```



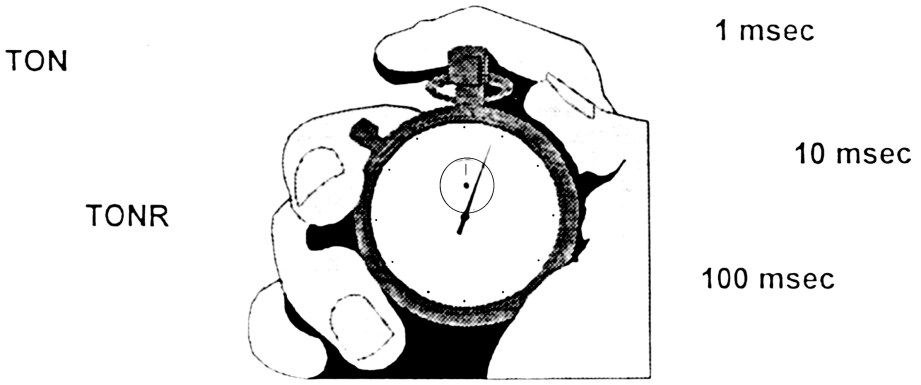

```
// 反馈输入通道
LD      SM0.0                   // SM0.0 总为 1
MOVW    VW204，VW206            // 保存旧值，VW204 存入 VW206
MOVW    VW202，VW204            // 保存旧值，VW202 存入 VW204
MOVW    VW106，VW202            // 控制器输出值 VW106 存入 VW202
MUL     522，VD200
MOVW    VW204，VW210
MUL     2570，VD208
MOVW    VW206，VW214
MUL     1024，VD212
+D      VD208，VD200
+D      VD212，VD200
+D      2048，VD200             // 为更好的移位而加 2048
SRD     VD200，12               // 被 4096 除
LD      V201.3                  // 注意符号位
S       V202.7，1
LD      SM0.0                   // SM0.0 总为 1
MOVW    VW202，VW100            // 作为新输入值反馈回去，即过程仿真结果作为过程变量（PV）输入 VW100
RETI                            // 中断程序 0 结束
```

33 S7-200 的定时器处理

概述

在程序中用定时器来控制时间。SIMATIC S7-200 系列可编程控制器设置了两种类型的定时器：接通延迟（On-Delay）定时 器（TON），“保持接通延迟”（Retentive On-Delay）定时器（TONR）。它们都可工作在三种精度下，即 1msec、10msec 和 100msec。
本例说明了每种定时器的操作及使用方法，重点在于不同精度下，定时器的操作方法的区别。

例图



说明

一、概述
S7-200 定时器由一个单独的使能输入端（IN）来控制，由于定时器是可使能的，因此，能够保留过去了的时间值。定时器还有一个预置时间值（PT），当前值更新时，它与当前值比较，定时器位（T 位）置位/复位（set/reset）就取决于当前值与预置值的比较结果。若当前值大于或等于预置时间值，定时器位接通（ON）；否则，定时器位断开（OFF）。当前值达到最大值时，计时停止。

当定时器复位时，它的当前值置 0，且定时器位断开。定时器在 Reset 复位指令作用下才复位（这是 TONR 定时器复位的唯一途径）。将定时器的当前值写入 0，并不能对定时器复位；同理，对定时器位写入 0，也不能使当前值复位。

两种类型的定时器（TON 和 TONR）对使能输入的响应不同。当使能输入有效时，它们开始计时；无效时，都断开。但当使能输入无效时，TON 定时器会自动复位而 TONR 则不复位。因此，TON 定时器适用于单个间隔的计时，而 TONR 定时器适用于对一些定时间隔的累加。

TON 和 TONR 定时器都可在 1msec、10msec 和 100msec 三种精度下工作，最大值分别为 32.767 秒，327.67 秒和 3276.7 秒。精度根据定时器编号来决定，如下表所示。定时器的当前计数值应与计时单位相乘，如 10msec 精度下计数值为 50，表示的时间值是 500ms。

	1msec	10msec	100msec
TON	CPU 212：T32 CPU 214：T32，T96	CPU 212：T33-T36 CPU 214：T33-T36， T97-T100	CPU 212：T37-T63 CPU 214：T37-T63， T101-T127
TONR	CPU 212：T0 CPU 214：T0，T64	CPU 212：T1-T4 CPU 214：T1-T4 T65-T68	CPU 212：T5-T31 CPU 214：T5-T31， T69-T95

二、1ms 精度的 TON/TONR 定时器

为了保证 1ms 定时器的精度，保持系统时间基准的系统中断程序每毫秒对它更新一次，通过增加 1 来更新当前值，除非它已在最大值。

保持 1ms 系统时间基准的系统中断程序独立于使能和使不能定时器。于是在当前 1ms 间隔内的某一点上，一个给定的 1ms 定时器将被使能，所以，时间间隔能短于 1ms。为此，用户应当预设一个时间值，该值比要求的最小时间间隔大 1。例如，用一个 1ms 定时器保证一个至少 56ms 的时间间隔，预设时间值应当设为 57。

由于一个激活的 1ms 定时器的当前值是在系统程序中被更新的，而且更新是自动的。一旦使能，执行 1ms 定时器 TON/TONR 控制指令就可控制定时器的开/关状态。

系统为了保证紧急中断程序的执行时间不受影响，只支持较小数目的 1ms 定时器。因为 1ms 定时器在一个中断程序内更新，所以这些定时器的当前值和定时器位（T 位）可能在扫描间隔的任何地方被更新，如果扫描时间超过 1ms 的话，那么每次扫描将被更新多次。因此，通过用户主程序的执行，这些值不能保持常数。

一旦 1ms 定时器使能位复位，则将关断它，并把当前值置为 0 且清除 T 位。

三、10ms 精度的 TON/TONR 定时器

为了保证更多的 10ms 定时器的精度，所有使能 10ms 定时器在每次扫描的开始更新，并在当前值上加 10ms 累加值。这个累加值也在扫描开始时决定，并且由 10ms 间隔数组成，该数自从最后一次更新（上次扫描的开始）就被确定，这个累加值用在所有使能 10ms 定时器。

10ms 间隔累加的处理独立于使能和使不能定时器，于是，在当前 10ms 间隔内的某一点上，一个给定的 10ms 定时器将被使能，所以，时间间隔能短于 10ms。为此，用户应当预设一个时间值，该值比要求的最小时间间隔大 1。例如，用一个 10ms 定时器保证一个至少 140ms 的时间间隙，预设时间值应当设为 15。

由于一个激活的 10ms 定时器的当前值是在扫描开始被更新的，而且更新是自动的，一旦使能，执行 10ms 定时器 TON/TONR 的控制指令就可控制定时器的开/关状态。与 1ms 定时器不一样，10ms 定时器的当前值每次扫描只被更新一次，通过用户主程序的执行，这些值将保持常数。

一旦 10ms 定时器使能位复位，则将关断它，并把当前值置为 0 且清除 T 位。

 西门子 版权所有

四、100ms 精度的 TON/TONR 定时器

100ms 定时器数目是最多的，相应的，它们的精度也最低。当执行了使能定时器指令，通过把 100ms 累加值加到当前值来更新所有使能 100ms 定时器。这些累加值与 10ms 累加值一样，也在扫描开始时决定，并且由 100ms 间隔数组成，该数自从最后一次更新（上次扫描的开始）就被确定。这个累加值用在所有使能 100ms 定时器上，使它们更新。

100ms 间隔累加的处理独立于使能和使不能定时器，于是，在当前 100ms 间隔内的某一点上，一个给定的 100ms 定时器将被使能，所以，时间间隔能短于 100ms。为此，用户应当预设一个时间值，该值比要求的最小时间间隔大 1。例如，用一个 100ms 定时器保证一个至少 2100ms 的时间间隔，预设时间值应当设为 22。

100ms 定时器的更新不是自动的，只有当定时器指令执行时，当前值才更新。因此，如果一个 100ms 定时器使能，但每次扫描不执行定时器指令，则该定时器当前值不能更新且它会丢失时间值。同样，在一次扫描中，如果相同的 100ms 定时器指令执行很多次，100ms 累加值将被多次加到定时器当前值上，于是，它就多得到时间值。因此，100ms 定时器只能用在一次扫描只执行一次的情况下。

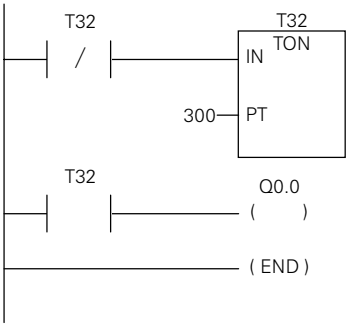
一旦 100ms 定时器使能位复位，则将关断它，并把当前值置为 0 且清除 T 位。

五、举例

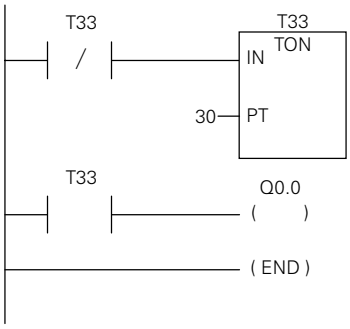
在不同的时刻更新 1ms、10ms 和 100ms 定时器所产生的效果，决定了你怎样使用定时器。例如，在下段程序中分析定时器的操作。

自动再触发一次有效的定时器

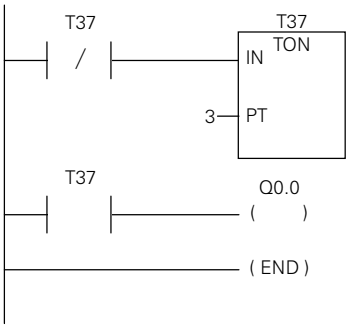
用 1ms 定时器



用 10ms 定时器



用 100ms 定时器



在使用 1ms 定时器的情况中，无论常开触点 T32 执行前或常闭触点 T32 执行后，每当更新定时器当前值，一次扫描将接通 Q0.0。

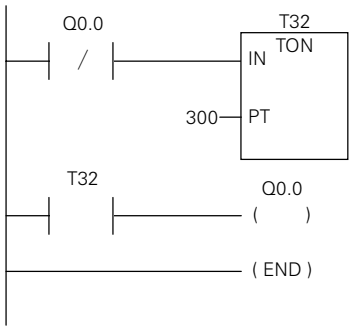
在使用 10ms 定时器的情况中，Q0.0 永远不被接通，因为定时器位 T33 接通是从扫描的顶部到定时器指令执行。一旦定时器指令执行，定时器当前值和它的 T 位将被置为 0。当常开触点 T33 执行时，T33 被关断，Q0.0 跟着被关断。

在使用 100ms 定时器的情况中，当某一次扫描定时器当前值达到预设值时，Q0.0 总接通。

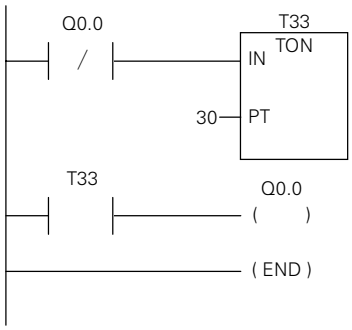
对三个定时器来说，通过简单的梯形图修改，将产生相同的效果，修改见下面。

自动再触发一次有效的定时器（固定）

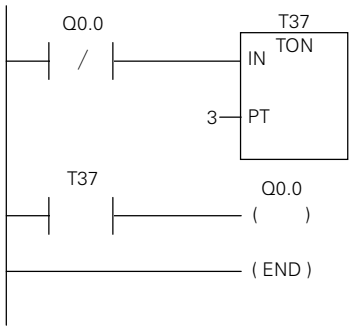
用 1ms 定时器



用 10ms 定时器



用 100ms 定时器



用常闭触点 Q0.0 代替定时器位（T 位）作为定时器的使能输入，在一次扫描后定时器达到预定值时可保证接通输出 Q0.0。

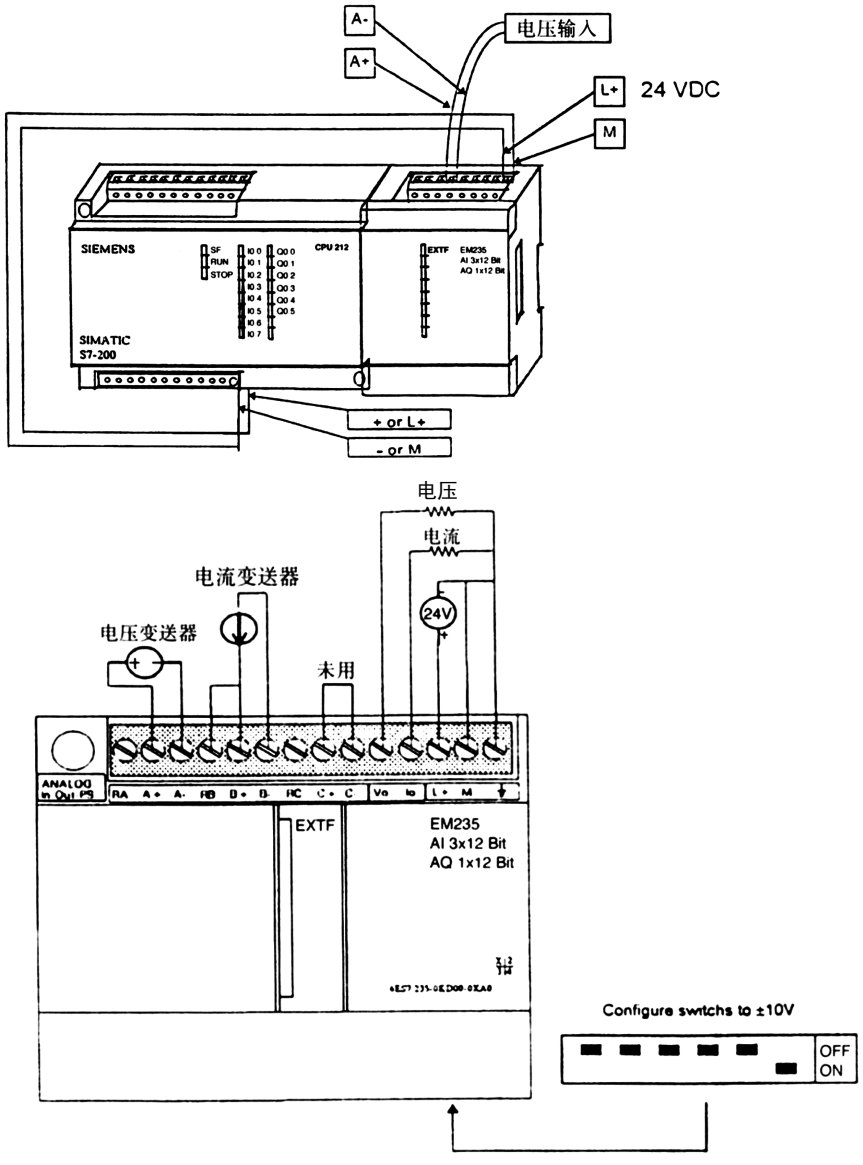
 西门子 版权所有

34 模拟量输入的处理

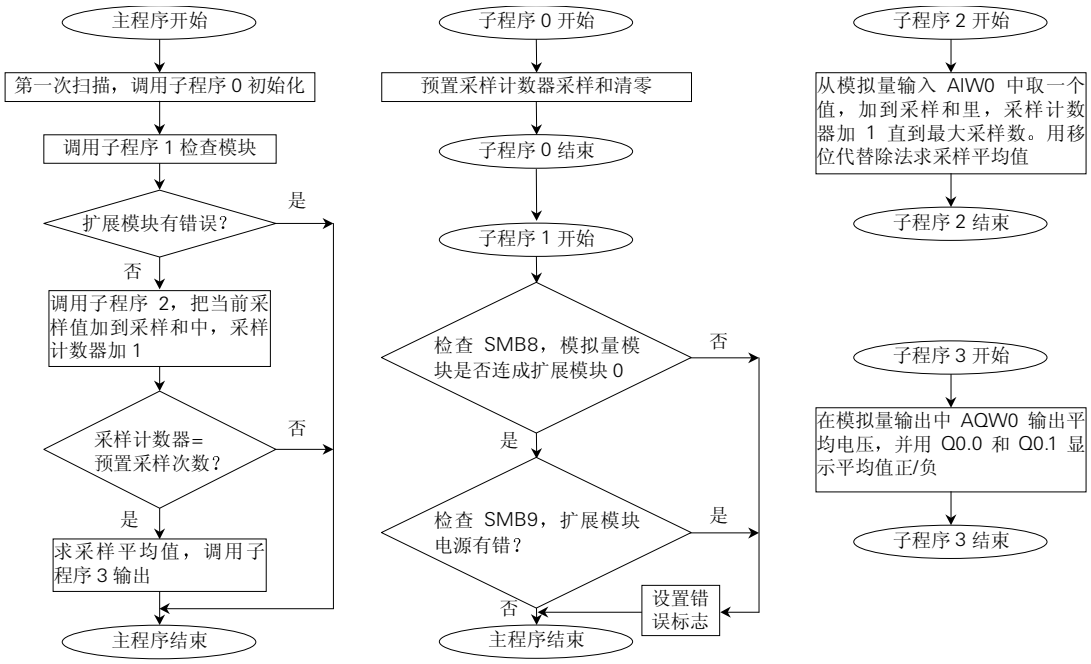
概述

本示例描述了模拟量模块 EM235 3AI/1AQ 与 CPU-212 或 CPU-214 一起使用的一种探讨。本例中模拟量输入值是给定采样次数的采样平均值，然后试验决定怎样设置输出。EM235 配置成±10V。

例图



程序结构



程序和注解

本程序描述了模拟量模块 EM235（3AI/1AQ）的功能，从 AIWO 中取输入值，为了增加稳定性而求多次采样值的平均值，再依据计算出的平均值在 AQW0 中输出模拟电压。

模拟量模块经过测试可提供模块错误信息。如果第一个扩展模块不是模拟量模块，Q1.0 接通。另外模拟量模块检查到的错误是电源出错，则将 CPU 上 Q1.1 接通。模拟量模块上有 EXTF 字样。

本程序中所用除法是简单的移位除法（用采样次数的 2 的方次）。因为移位只花费较短的扫描时间，该数能从 2 变化到 32768。

输入字是 12 位长。如果采样次数大于 16（2 的 4 次方），那么和的长度将大于一个字（16 位）。于是需要用双字（32 位）存贮采样和。为把输入值加到采样和中，你应当把它转成双字。

当输入数为负值时，最高有效字增添 1；若为正值，最高有效字增添 0 来校正输入值。

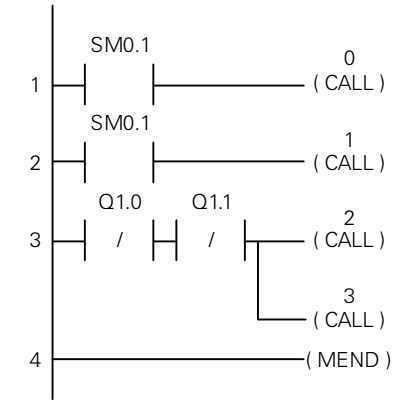
本程序长度为 118 个字。

 西门子 版权所有

LAD（S7-MicroDOS）	STL（IEC）

// 主程序

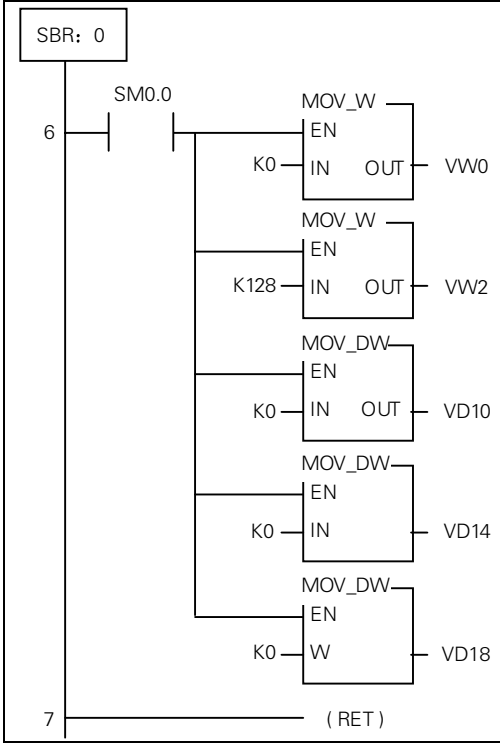
// 主程序




```
LD      SM0.1   // 仅首次扫描时 SM0.1=1
CALL    0       // 初始化
LD      SM0.0   //
CALL    1       // 检查模块
LDN     Q1.0    // 若模块正常（即标志位 Q1.0=0）
AN      Q1.1    // 且 Q1.1=0
CALL    2       // 则采样模拟量输入
CALL    3       // 输出所要电压
MEND            // 主程序结束
```

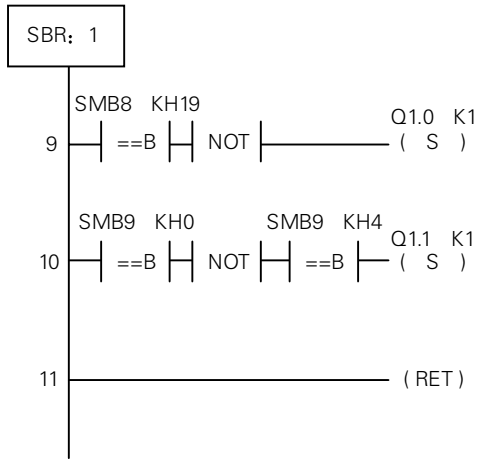
// 子程序

```
//**********************************************
// 功能：初始化
//**********************************************
```



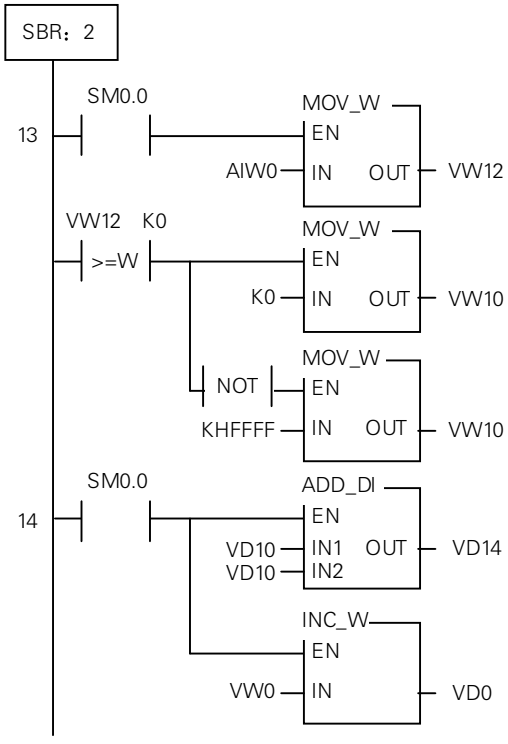

```
SBR     0       // 子程序 0
LD      SM0.0   // SM0.0 总为 1

MOVW    0，VW0  // 计数器清零

MOVW    128，VW2 // 预置采样次数

MOVD    0，VD10 // 当前采样值清零

MOVD    0，VD14 // 当前采样和清零

MOVD    0，VD18 // 平均值清零

RET             子程序 0 结束
```

```
//  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
// 功能：检查模块是否正常工作
//  * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

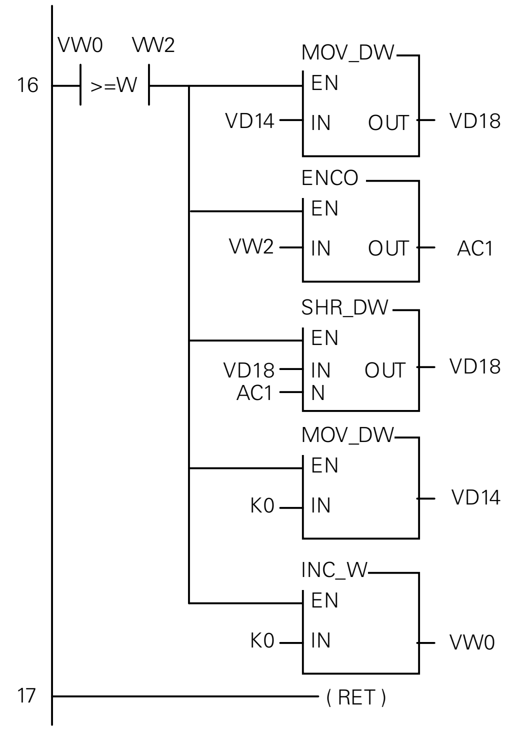



```
SBR    1              // 子程序 1
LDB=   SMB8，16#19    // 检查第一个扩展模块是否存在，
NOT                   //
S      Q1.0，1        // 若不存在，则置 Q1.0=1；
LDB=   SMB9，16#00    // 检查第一个扩展模块是否有错误，
NOT                   //
AB=    SMB9，16#04    // 检查电源是否正常，
S      Q1.1，1        // 若有错，则置 Q1.1=1。
RET
```

```
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
// 功能：采样模拟量，从 AIW0 取模块量输入值
//* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```




```
SBR    2              // 子程序 2

LD     SM0.0          // SM0.0 总为 1
MOVW   AIW0，VW12     // 在 VW12 中放置模拟量输入值

LDW>=  VW12，0        // 检查输入信号

MOVW   0，VW10        // 把输入值转换成双字

NOT                   // 即 VD10=模拟量输入值（当前采样值）
MOVW   16#FFFF，VW10

LD     SM0.0          // SM0.0 总为 1
+D     VD10，VD14     // 把当前采样值加到采样和中

INCW   VW0            // 采样计数器值加 1
```

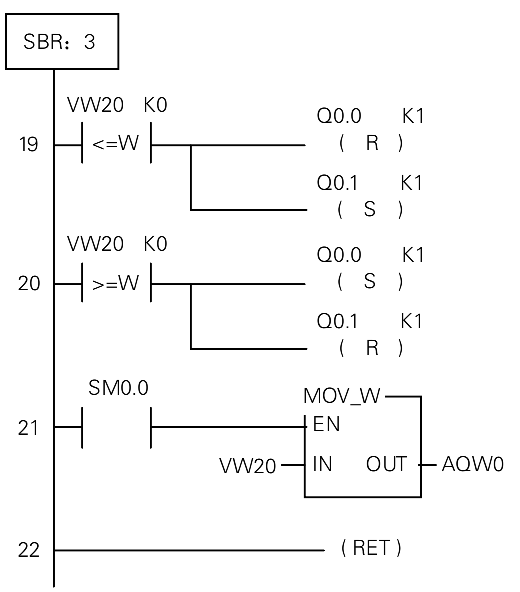



```
LDW>=VW0，VW2        // 若达到采样次数
MOVD  VD14，VD18     // 则把采样和 VD14 复制到
                        VD18 中
ENCO  VW2，AC1       // 计算移位数
SRD   VD18，AC1      // 用移位实现除法，即求采
                        样平均值
MOVD  0，VD14        // 重新初始化，即将采样和
                        清零
MOVW  0，VW0         // 采样计数器清零
RET                  // 子程序 2 结束
                     // 模拟量输入平均值=VW20
```

//＊＊＊

// 功能：基于平均输入值设定输出值

//＊＊＊




```
SBR   3              // 子程序 3
LDW<=VW20，－160     // 如果平均值为负
R     Q0.0，1        // 则关断 Q0.0
S     Q0.1，1        // 且接通 Q0.1，用来显示当
                        前平均值为负；
LDW>=VW20，160       // 如果平均值为正
S     Q0.0，1        // 则接通 Q0.0
R     Q0.1，1        // 且关断 Q0.1，用来显示当
                        前值为正。
LD    SM0.0          // SM0.0 总为 1
MOVW  VW20，AQW0     // 在模块 AQW0 输出平均
                        值
RET                  // 子程序 3 结束
```

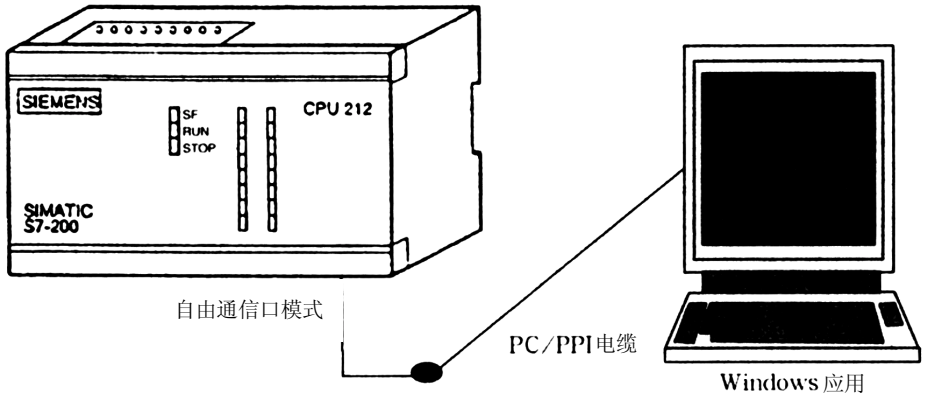
 西门子 版权所有

35 S7-200 与 PC 之间的连接：从 Windows 应用程序中读数据

概述

本示例讲述了怎样用第三部分软件，由 Windows 应用程序，从 SIMATIC S7-200 系列 CPU 中读数据。本例模仿一个简单的‘泵站’系统，把数据发送到 Microsoft Excel 中不同的位置。

例图

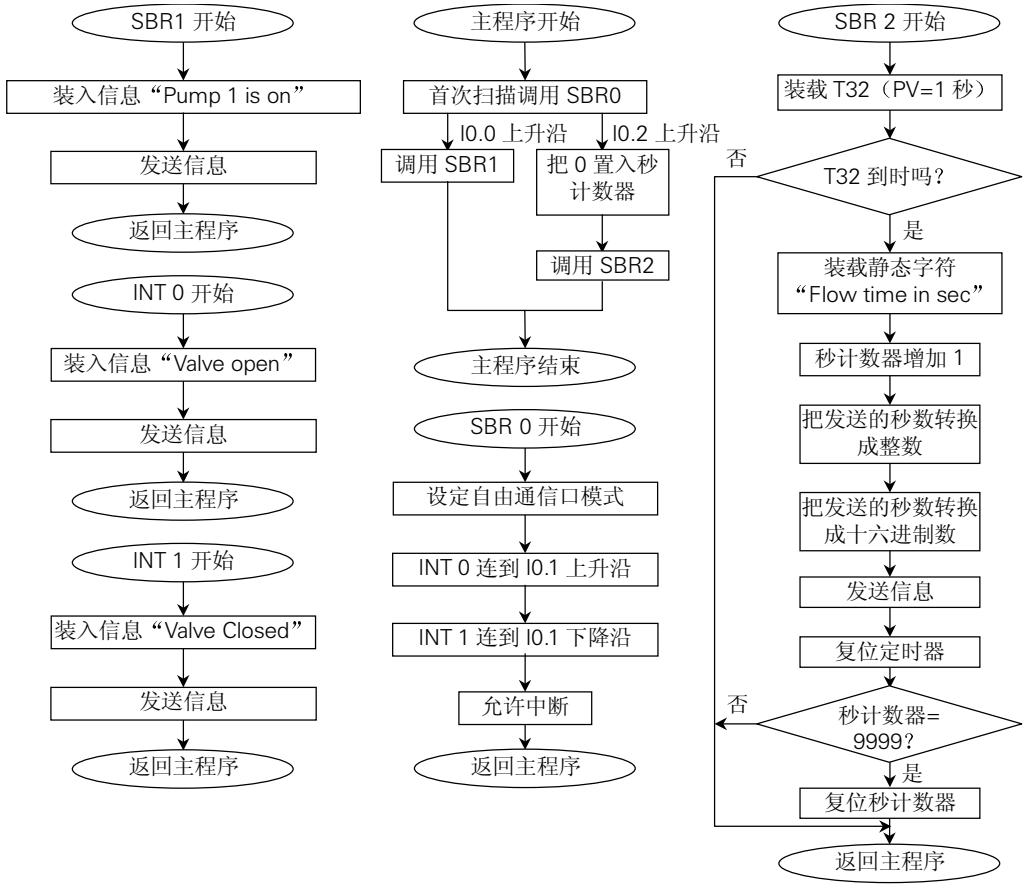


硬件和软件要求

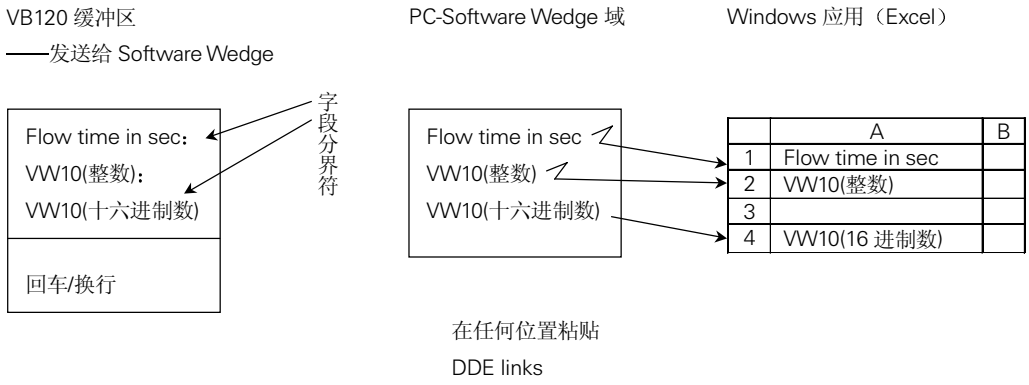
硬件：
SIMATIC CPU 214 或 212

软件：
Software Wedge for Windows：
Professional Edition Part # SW20WP 或其它支持 DDE Link 的基于 Windows 的兼容软件

程序结构：


SBR1 开始
装入信息“Pump 1 is on”
发送信息
返回主程序
INT 0 开始
装入信息“Valve open”
发送信息
返回主程序
INT 1 开始
装入信息“Valve Closed”
发送信息
返回主程序
主程序开始
首次扫描调用 SBR0
I0.0 上升沿
I0.2 上升沿
调用 SBR1
把 0 置入秒计数器
调用 SBR2
主程序结束
SBR 0 开始
设定自由通信口模式
INT 0 连到 I0.1 上升沿
INT 1 连到 I0.1 下降沿
允许中断
返回主程序
SBR 2 开始
装载 T32（PV=1 秒）
否
T32 到时吗？
是
装载静态字符“Flow time in sec”
秒计数器增加 1
把发送的秒数转换成整数
把发送的秒数转换成十六进制数
发送信息
复位定时器
否
秒计数器=9999？
是
复位秒计数器
返回主程序

程序和注释


VB120 缓冲区
——发送给 Software Wedge
Flow time in sec：
VW10(整数)：
VW10(十六进制数)
回车/换行
字段分界符
PC-Software Wedge 域
Flow time in sec
VW10(整数)
VW10(十六进制数)
在任何位置粘贴
DDE links
Windows 应用（Excel）
A
B
1 Flow time in sec
2 VW10(整数)
3
4 VW10(16 进制数)

SIMATIC CPU 214 能与基于 Windows 的程序，如 SoftwareWedge for Windows 之类的软件相联系。所以，来自 CPU 214 的信息能显示在任何 Windows 应用程序中，同时信息也能从 Windows 应用程序写到 CPU 214 中。

目前，SoftwareWedge 不允许发送来自不同输入的信息，在不同的时间显示和更新屏幕的不同部分。然而，从 CPU 214 发送来的各种信息，可以显示在不同位置，每个部分必须显示在 SoftwareWedge 自己的区域里，每个区域被发送来的某个字段分界符分隔开。这些字符可以是用户任意要求的。此外，每次发送结束，必须有一个或多个“结束”字符，它也可由用户指定。

装载完 SoftwareWedge 软件包后，选择 DDE 服务器方式，指定 DDE 应用名、题目及适用的项目，接着把通信口设定为 9600 波特，没有奇偶校验，每个字符 8 位，1 个停止位。记住所设定的通信口是好的。最后，要输入的记录结构必须定义。在下面程序中，从收到任一字符作为记录的开始，收到一个回车和换行作为记录的结束，选择多个数据字段，用 3 作为字段的最大数目，用“:”（ASCII 码为 58）号作为字段分界符。最后，在 Windows 应用中，用拷贝/粘贴联接命令把不同数据字段粘贴在屏幕上所要求的部分。

选择：在它进入另一个 Windows 应用前 SoftwareWedge 提供了取消变量格式的自动转换。

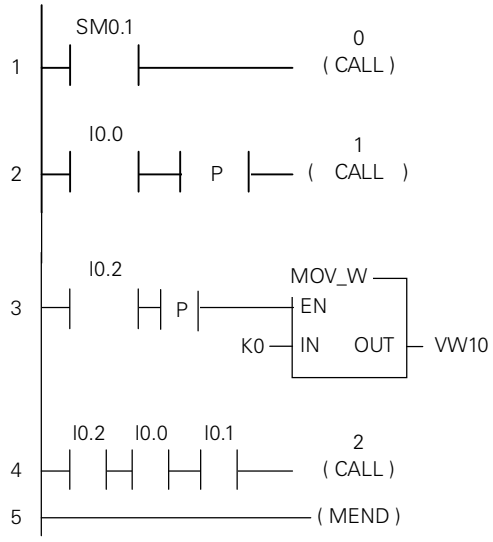
本程序长度为 158 个字

LAD（S7-MicroDOS）	STL（IEC）

主程序

```
// 标题：Windows 界面，并在 Windows 里读
// 该程序描述了利用适当的软件，S7-200 系列 CPU 可以发送信息给任何 Windows
// 应用产品（见上面）
// 本例给出了一个简单的‘泵站’系统。假定 I0.0 为起动“主泵”的开关，I0.1 为开或
// 关紧急阀的开关，I0.2 为开或关主阀的开关（主阀控制液体流出）
// 操作员坐在计算机可以看到三个状态变化中的任一个信息。I0.0 控制一个静态信息
// “Pump1 is on”的出现。I0.1 控制着信息“Valve open”或
// “Valve closed”交替出现。I0.2 控制显示液体流出的时间，这个信息在 I0.2
// 接通时每秒变化一次。该程序不允许同时显示所有信息，但是可作为每个开关变化的
// 状态。然而，对程序作某些修改，那就可以同时显示所有信息。这就要求所有数据随
// 时发送（即使没有变化），并改用新的发送模式。
// 本程序字段分界符是（:），发送结束符是一个回车及换行符。
// 该程序试图写一系列同类型的信息给 Windows 应用程序。一条静态信息
// “Pump is on”，一条切换信息“Valve is ope/closed”，以及一条不断地变化的
// 整数或十六进制数“Flow time（in sec）# # # # ”全包括在一起，以便给用户一个多种多样的信
  息变化。
```

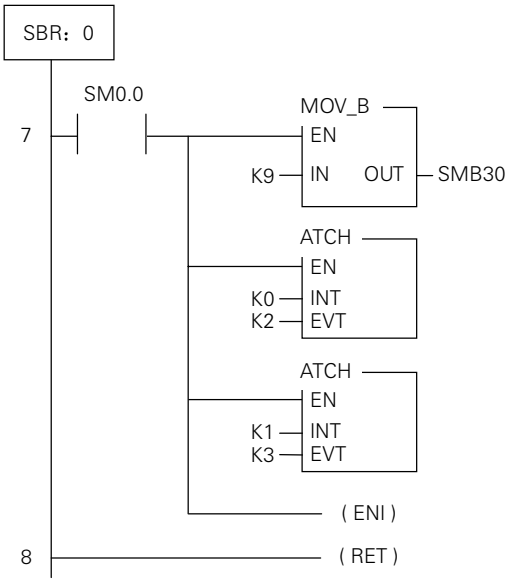
// 下面列出本程序用到的变量	
//VW10	主计数存贮器，用来显示液体流出时间（秒）。
//VW20	第二计数存贮器——来自 VW10 的拷贝，用在 SBR2 中进行 IBCD 转换
//	而且不擦除计数器里的值。
//VB80	存贮数值 14，即发送缓冲区中待发送的 ASCII 码（十六进制）字母数。用作 XMT 发送命令的前提。
//VD81—VW93	信息：“Pump 1 is on”
//VB100	依据紧急阀的状态存贮 12 或 14，即发送缓冲区中待发送的 ASCII 码（十六进制）字母数。
//VD101—VD109	
// 或 VD101—VD113	信息：“Valve open”或“Valve closed”
//VB120	存贮数值 28，即发送绘冲区中待发送的 ASCII 码（十六进制）字母数。
//VD121—VD133	信息“Flow time in sec”
//VB137	以十六进制形式存贮“:”，作为字段分界符
//VB138—VB141	存贮秒计数器的 ASCII 码，它在 Windows 中作为整数显示
//VB142	以十六进制形式存贮“:”，作为下一个字段分界符
//VB143—VB146	存贮秒计数器的 ASCII 码，它在 Windows 中作为十六进制显示
//VW147	回车换行—表示发送信息结束




```
LD      SM0.1   // 首次扫描位 SM0.1=1
CALL    0       // 调用 SBR 0
LD      I0.0    // 若主泵开关 I0.0=1
EU              // 且上升沿
CALL    1       // 则调用 SBR1

LD      I0.2    // 若主阀开关 I0.2=1
EU              // 且上升沿
MOVW    0，VW10 // 则定时器置 0
LD      I0.2    // 若主阀开关 I0.2=1
A       I0.0    // 且泵运转（I0.0=1）
A       I0.1    // 且紧急阀打开（I0.1=1）
CALL 2          // 则调用 SBR2
MEND            // 主程序结束
```

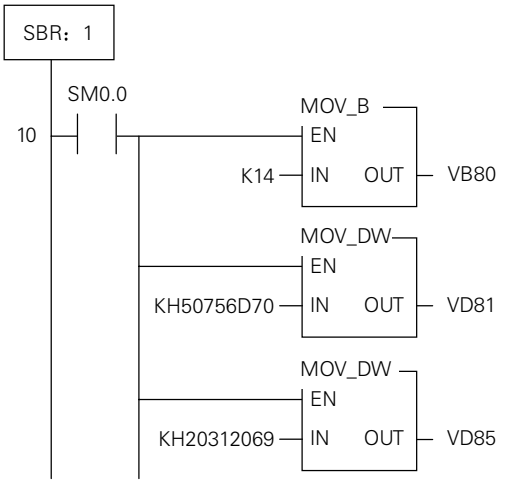
 西门子 版权所有

子程序

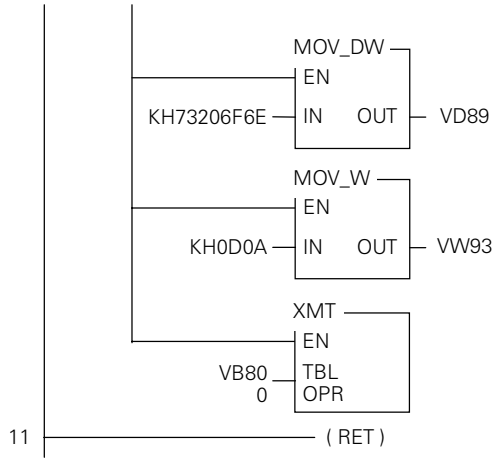



```
SBR     0           // 初始化子程序 0

LD      SM0.0       // SM0.0 总是 1
MOVB    9，SMB30    // 自由通信口模式：9600 波特，无奇偶校验，每个字符 8 位
ATCH    0，2        // 指定中断事件 2（I0.1 上升沿）
                    调用中断程序 0
ATCH    1，3        // 指定中断事件 3（I0.1 下降沿）
                    调用中断程序 1
ENI                 // 允许中断
RET                 // 返回主程序
```

// 每当主泵开关 I0.0 上升沿时，将“Pump 1 is on”装入，从 VB81 开始的缓冲区，并在
//VB80 中置发送字节数 14，通过自由通信口模式发送

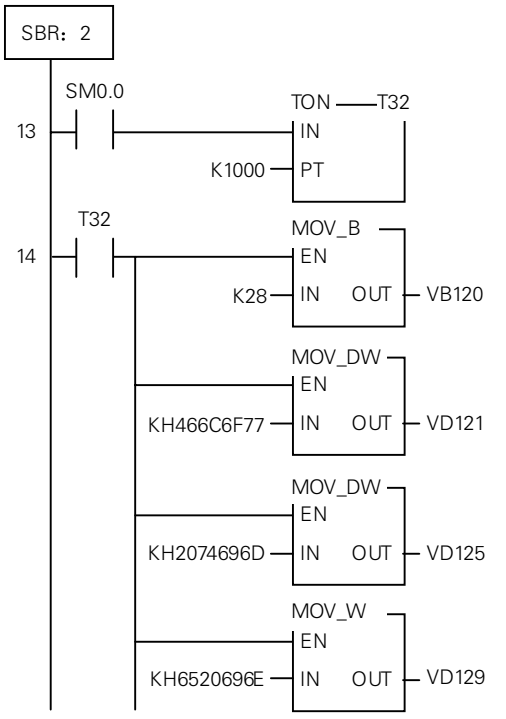



```
SBR     1           // 写“Pump 1 is on”

LD      SM0.0       // SM0.0 总是 1

MOVB    14，VB80    // 置发送信息长度（14 个字节）

MOVD    16#50756D70，VD81
                    // 十六进制信息— “Pump”

MOVD    16#20312069，VD85
                    // “1 i”
```

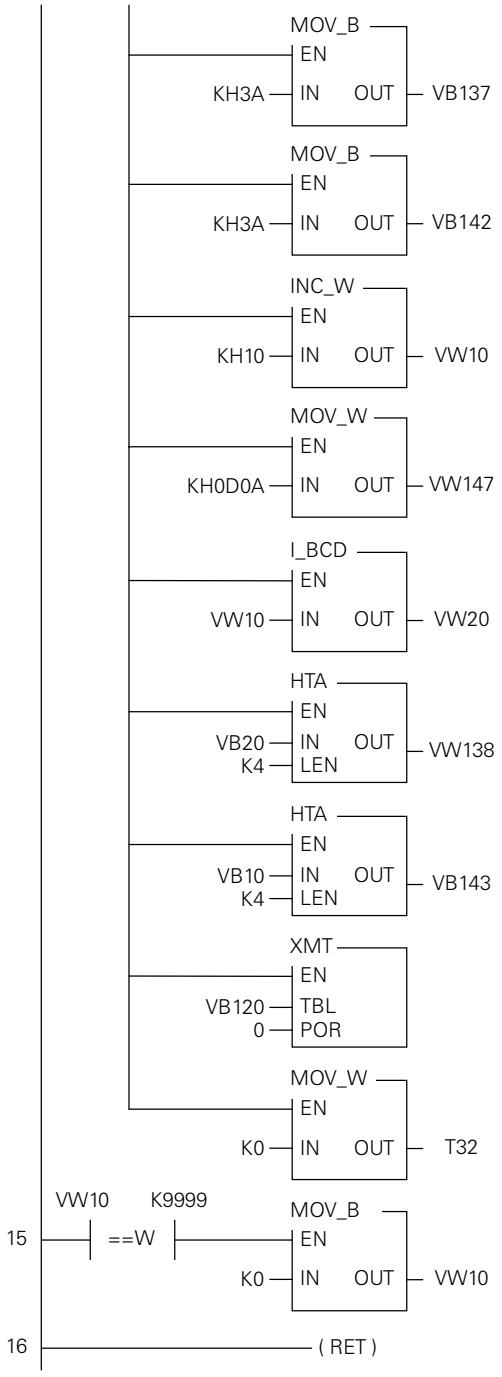



```
MOVD    16#73206F6E，VD89
                    // “s on”
MOVW    16#0D0A，VW93
                    // 回车换行表示信息结束
XMT     VB80，0     // 发送（VB80 中已存放所
                       发送信息的字节数）
RET                 // 返回主程序
```

```
// 当 I0.2 接通时，给 T32（TON，1ms 精度的定时器）装入预设值 1000（1 秒）。当 T32 到时后
//VW10 增加 1，并将新值半入 VW20。VW20 由整数转换成 BCD 码，再转换成 ASCII 码，
// 并存入发送缓冲区。VW10 也被移到适当的缓冲器。
// 十六进制数作为实际值转换成 ASCII 码供发送。在子程序结束时，进行测试，
// 如果值超过 9999，复位 VW10。这是因为 BCDI 命令只允许转换一个字（或 4 位址六进制数）。
// 如果需要十六进制数，则任何数都能被转换。从 VB120 开始的发送缓冲区每秒都更新
// （由 T32 定时），且读“Flow time in sec（整数）（十六进制数）”
```



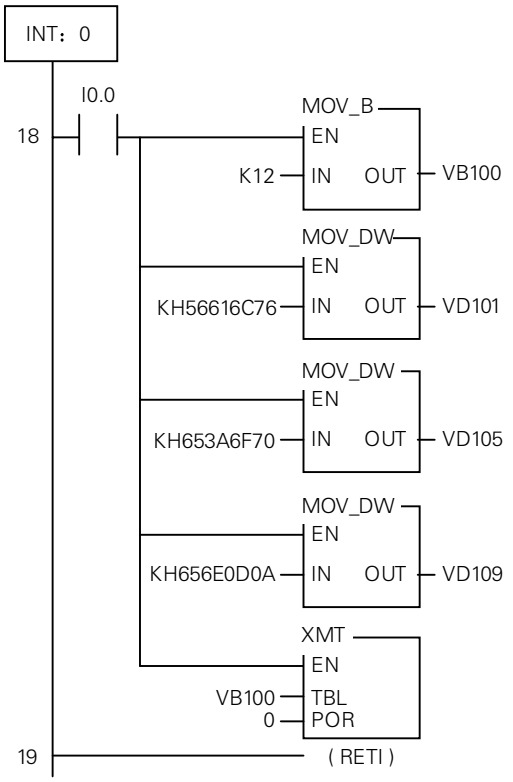

```
SBR     2           // 子程序 2
LD      SM0.0       // SM0.0 总为 1
TON     T32，1000   // 启动定时器 T32 (1000ms)
LD      T32         // 一秒以后
MOVB    28，VB120   // 置发送信息长度（28 个字
                       节）
MOVD    16#466C6F77，VD121
                    // “Flow”
MOVD    16#2074696D，VD125
                    // “tim”
MOVD    16#6520696E，VD129
                    // “e in”
```




```
MOVD    16#20736563VD133
                        // “sec”
MOVB    16#3A，VB137
                        // “:”字段分界符
MOVB    16#3A，VB142
                        // “:”字符分界符
INCWVW10                // VW10 增加 1
MOVW    16#0D0A，VW147
                        // 回车换行表示信息结束
MOVW    VW10，VW20
                        // 把 VW10 拷贝到 VW20
IBCD    VW20            // 把 VW20 转换成 BCD 码
HTA     VB20，VB138，4
                        // 把 BCD 转换成 ASCII（作
                           数值显示），装入发送缓
                           冲区。
HTA     VB10，VB143，4
                        // 把十六进制转换成 ASCII
                           （作十六进制显示），装
                           入发送缓冲区
XMT     VB120，0        // 发送（VB120 中已存放所
                           发送信息的字节数）
MOVW    0，T32          // 复位定时器
LDW=    VW10，9999
                        // 若流出时间超过 9999 秒
MOVW    0，VW10         // 则复位秒计算器
RET                     // 返回主程序
```

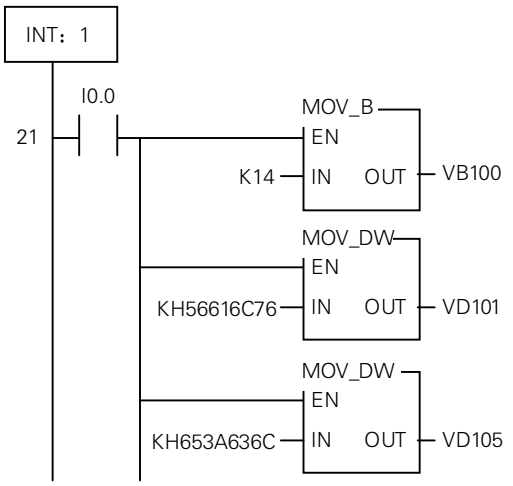
中断程序

// 若为 I0.1 上升沿，（即接通），则调用中断程序 0 发送信息“Valve open”。
// 因此，只有阀转换状态时才更新阀信息。

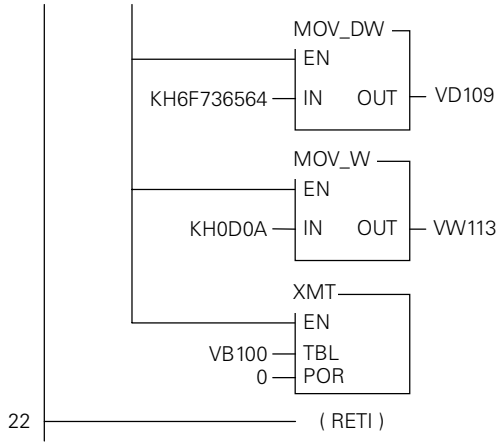



```
INT   0                    // 中断程序 0

LD    SM0.0                // SM0.0 总为 1
MOVB  12，VB100            // 置发送信息长度（12 个字节）

MOVD  16#56616C76，VD101
                           // “Valv”

MOVD  16#653A6F70，VD105
                           // “e:op”，“：”是字段分界符

MOVD  16#656E0D0A，VD109
                           // “en”，回车换行表示信息结束

XMT   VB100，0             // 发送（VB100 中已存放所发送信息的字节数）

RETI                       // 返回主程序
```

// 若为 I0.0 下降沿（即断开），则调用中断程序 1 发送“Valve closed”。只有阀转换
// 状态时信息才再一次更新。




```
INT   1                    // 中断程序 1

LD    SM0.0                // SM0.0 总为 1
MOVB  14，VB100            // 置发送信息长度（14 个字节）

MOVD  16#56616C76，VD101
                           // “Valv”

MOVD  16#653A636C，VD105
                           // “e cl”
```




```
MOVD    16#6F736564，VD109
                    // “osed”

MOVW    16#0D0A，VW113
                    // 回车换行表示信息结束

XMT  VB100，0       // 发送（VB100 中已存放所
                       发送信息的字节数）

RETI                // 返回主程序
```

 西门子 版权所有

36 用 PT100 电阻温度传感器测量温度并监视温度

概述

本例讨论如下内容：用模拟量扩展模块 EM235 测量温度和监视指定温限，在该模拟量模块的一个输入通道上连接 PT100 温度传感器。

为了把 PT100 的随温度变化的电阻转换成电压，模拟量输出作为恒电流源而使用，即输出 12.5mA 恒电流供给 PT100 传感器。在这个电路中，产生了 5mV/°C 的线性输入电压。EM235 把这个电压转换成数字量，程序周期性地读这些数字量，并将所读的这些数，利用下面的公式计算出温度°C。

$$T[°C]=\frac{\text{温度数字量}-\text{0°C 偏置量}}{\text{1°C 数字量}}$$

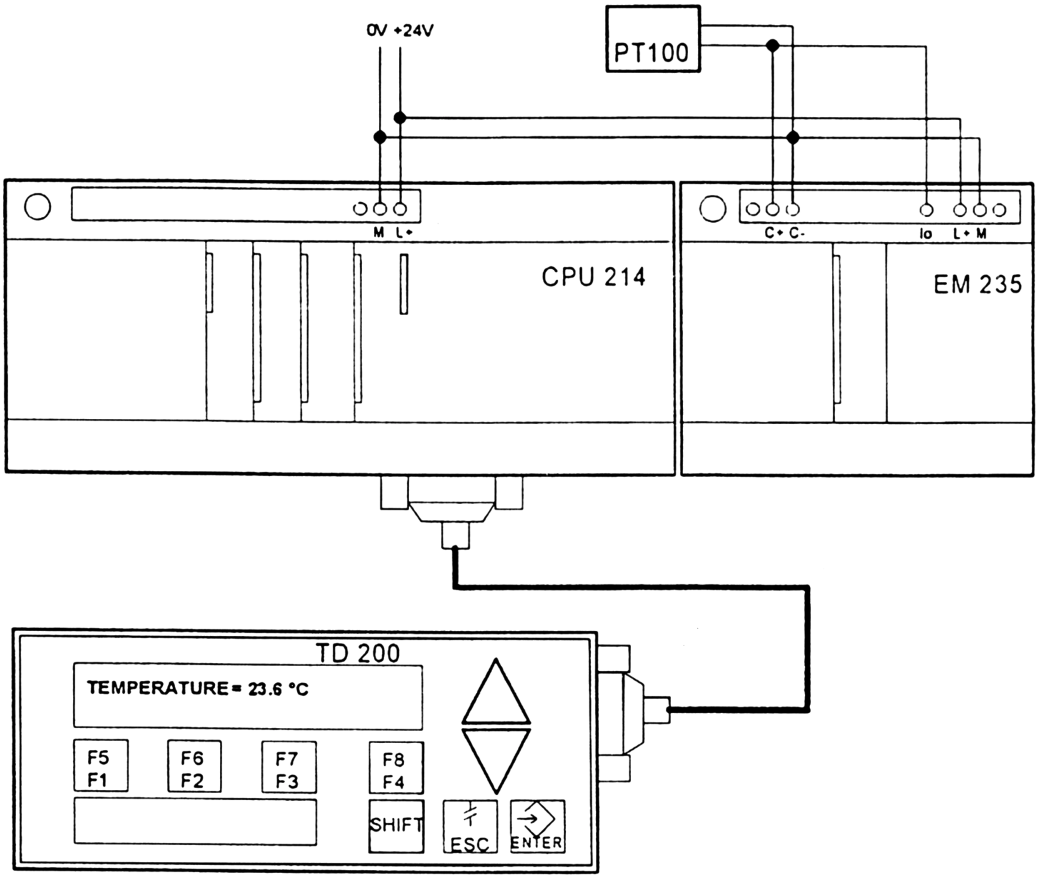
温度数字量=存贮在 AIWx（x=0，2，4）中的值

0°C 偏置量=在 0°C 测量出的数字量，该值为 4000

1°C 数字量=温度每升高 1°C 的数字量，本例中，为 16

程序计算出带一位十进制小数点的温度值，并把该值写入信息 1 所属的变量单元中，信息 1 为“Temperature=xxx.x°C”，再用 TD200 显示这些信息。在程序的初始化段中，用户可以输入高限和低限温度。如果测量温度超出选范围，那么 TD200 温度监视器的第二行显示警告信息。如果测量温度超过温度高限，那么 TD200 的第二行显示信息 2，即“Temperature＞xxx.x°C”；如果测量温度低于温度低限，那么 TD200 的第二行显示信息 3，即“Temperature＜xxx.x°C”。

例图

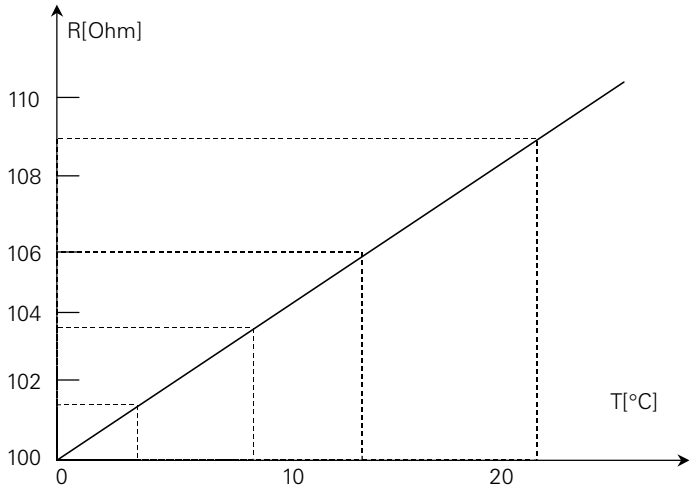

0V +24V
PT100
M L+
CPU 214
C+ C-
Io L+ M
EM 235
TD 200
TEMPERATURE = 23.6 °C
F5
F1
F6
F2
F7
F3
F8
F4
SHIFT
ESC
ENTER

 西门子 版权所有

硬件描述

温度传感器：
PT100 是铂电阻温度传感器，它适用于测量－60°C 到+400°C 之间的温度。

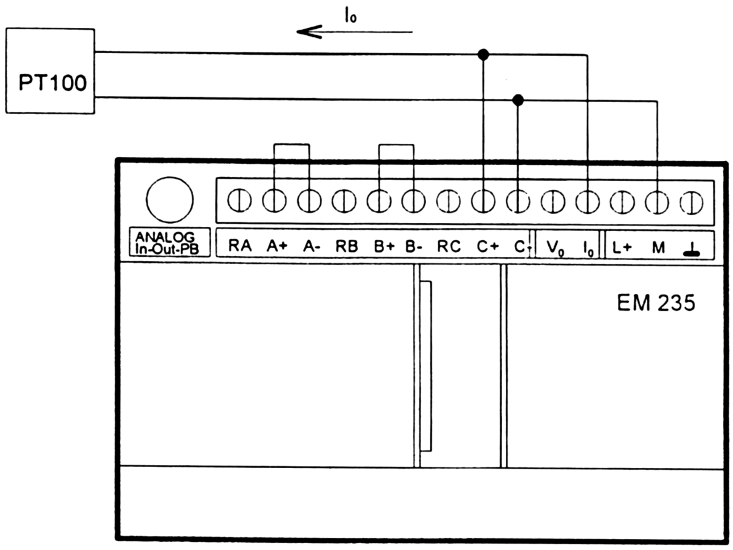
计算 PT100 所需电流
PT100 在 0°C 时电阻为 100Ω，随着温度的变化电阻呈线性变化，大约是每摄氏度 0.4Ω。



为了产生 5mV/°C 的电压系数，需要提供 12.5mA 电流。由于模拟量输出精度为 10μA/数，为了得到 12.5mA 输出电流所需的输出数必须为 1250。因为 AQW 数据字向右移 4 位，因此输出数必须乘以 16。这样，为了初始化模拟量输出 Io 为 12.5mA 电流，在 AQW0 中必须设置 20000 输出数。

等式为：（32000/20mA*12.5mA=20000）

EM235 电路：



设置 EM235 模块上的开关，0~10V 范围可如下选择：

开关号：

1	3	5	7	9	11
ON	OFF	OFF	OFF	ON	OFF

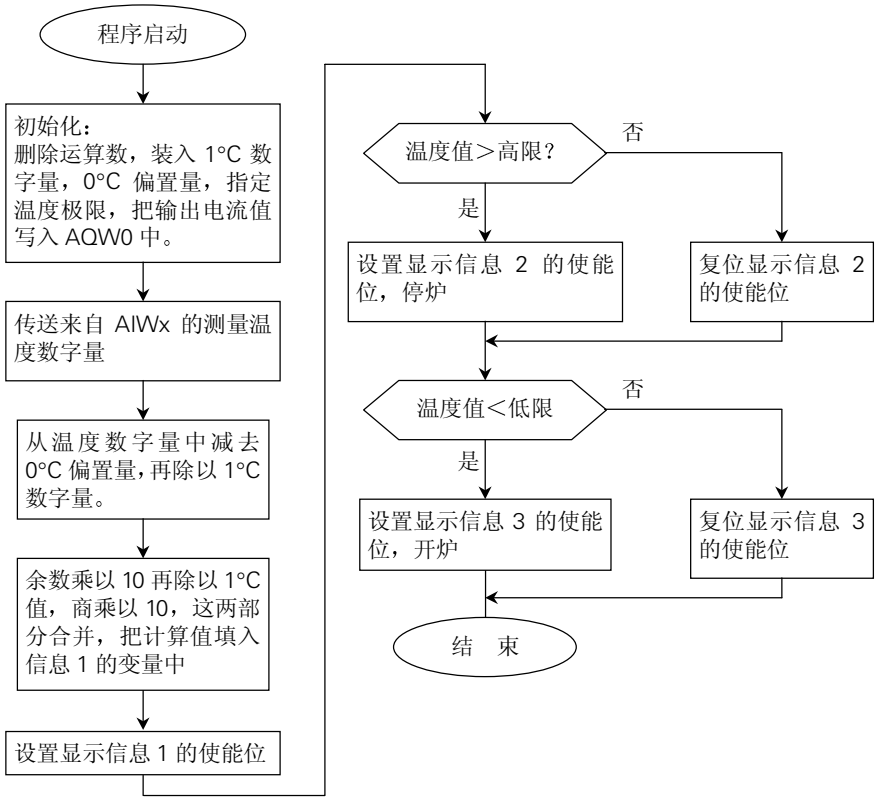
依据用在 EM235 上的通道数，本程序用到的 AI 字相应的地址为：

AIW0：输入通道 1　　AIW2：输入通道 2

AIW4：输入通道 3　　AQW0：输出通道 1

未使用的 EM235 输入端被短路。

程序结构

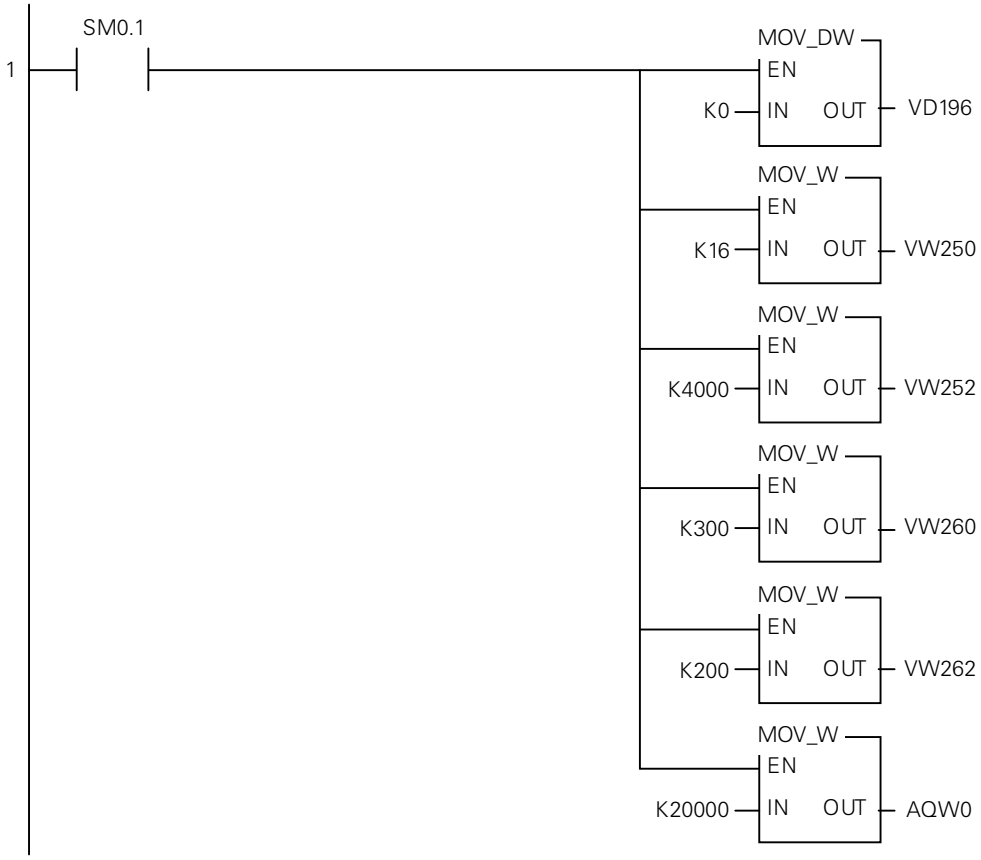

程序启动
初始化：
删除运算数，装入 1°C 数字量，0°C 偏置量，指定温度极限，把输出电流值写入 AQW0 中。
传送来自 AIWx 的测量温度数字量
从温度数字量中减去 0°C 偏置量，再除以 1°C 数字量。
余数乘以 10 再除以 1°C 值，商乘以 10，这两部分合并，把计算值填入信息 1 的变量中
设置显示信息 1 的使能位
温度值＞高限?
否
是
设置显示信息 2 的使能位，停炉
复位显示信息 2 的使能位
温度值＜低限
否
是
设置显示信息 3 的使能位，开炉
复位显示信息 3 的使能位
结 束

程序和注释

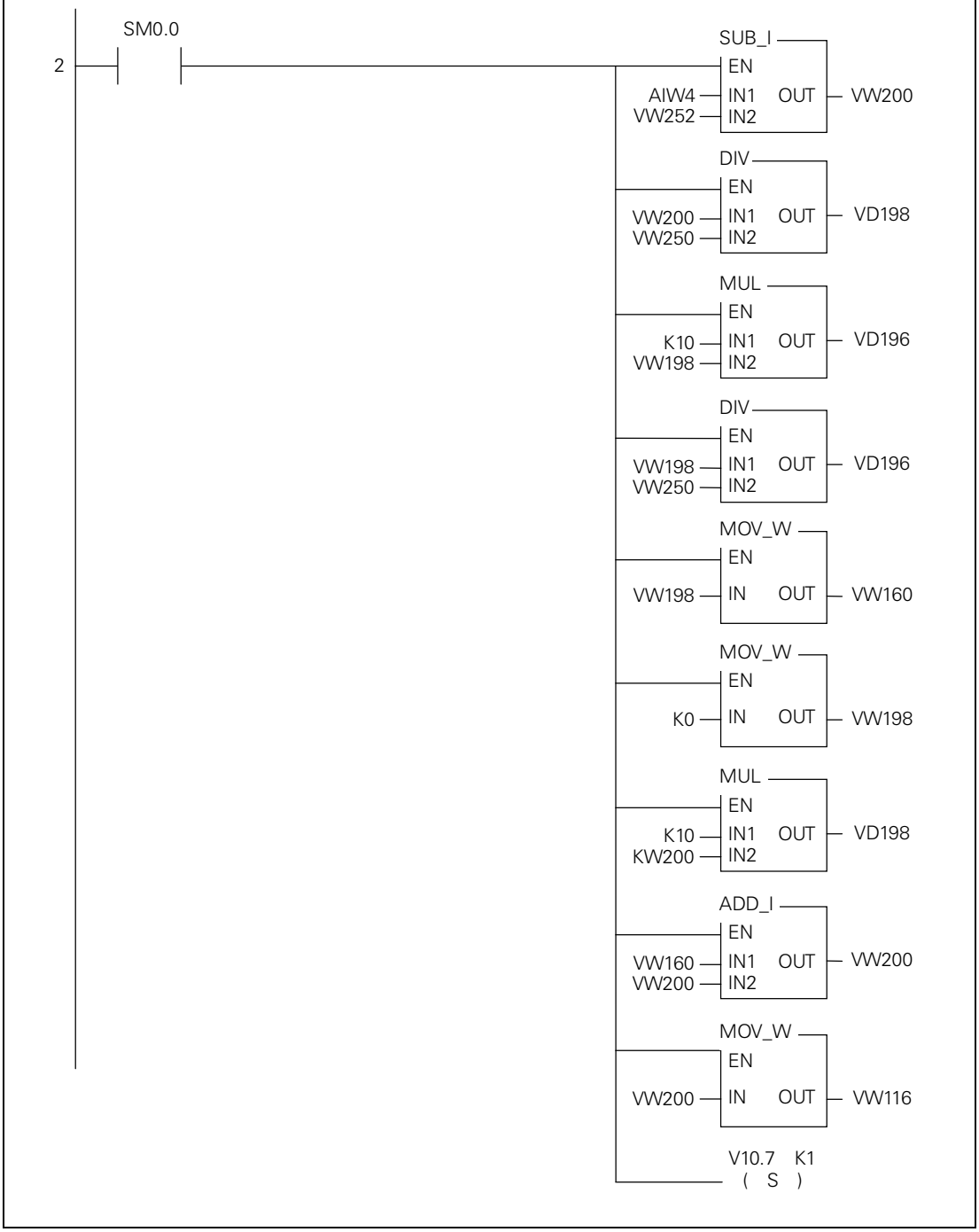
LAD(S7-MicroDOS)	STL(IEC)

主程序

// 题目：用 PT100 RTD 测量温度，并用 TD200 显示温度




```
LD      SM0.1          // 在第一个扫描周期 SM0.1=1，
MOVD    0，VD196       // 清除 VW196 和 VW198
MOVW    16，VW2502     // 在 VW250 中装入 1°C 数字量=16
MOVW    4000，VW252    // 0°C 偏置量=4000
MOVW    300，VW260     // 温度高限为 30°C
MOVW    200，VW262     // 温度低限为 20°C
MOVW    20000，AQW0    // Io 输出数=20000
```


2
SM0.0
SUB_I
EN
AIW4
VW252
IN1
IN2
OUT
VW200
DIV
EN
VW200
VW250
IN1
IN2
OUT
VD198
MUL
EN
K10
VW198
IN1
IN2
OUT
VD196
DIV
EN
VW198
VW250
IN1
IN2
OUT
VD196
MOV_W
EN
VW198
IN
OUT
VW160
MOV_W
EN
K0
IN
OUT
VW198
MUL
EN
K10
KW200
IN1
IN2
OUT
VD198
ADD_I
EN
VW160
VW200
IN1
IN2
OUT
VW200
MOV_W
EN
VW200
IN
OUT
VW116
V10.7 K1
(S)

```
LD      SM0.0          // SM0.0 总为 1
MOVW    AIW4，VW200    // 把测量温度数字量装入 VW200
-I      VW252，VW200   // 减去 0°C 偏置量
DIV     VW250，VD198   // 除以 1°C 数字量
MUL     10，VD196      // 余数乘以 10
DIV     VW250，VD196   // 10×余数/16=一位小数点的数
MOVW    VW198，VW160   // 保存一位小数点的数，即温度小数值×10
MOVW    0，VW198       // 删除 VW198 中的值
MUL     10，VD198      // 温度整数值乘以 10
+I      VW160，VW200   // 温度整数值×10+温度小数值×10
MOVW    VW200，VW116   // 传送结果到 VW116 以供显示
S       V10.7，1       // 显示信息 1 的使能位 V10.7=1
```


```
LDW>=   VW200，VW260   // 如果温度超过高限
=       V10.6          // 则显示信息 2 的使能位 V10.6=1
R       Q0.0，1        // 停炉，即输出端 Q0.0=0
MOW     VW260，VW136   //VW136 里的高限供信息 2 显示
```

```
LDW<=   VW200，VW262        // 如果温度低于低限
=       V10.5               // 则显示信息 3 的使能位 V10.5=1
S       Q0.0，1             // 开炉，即输出端 Q0.0=1
MOVW    VW262，VW156        //VW156 里的低限供信息 3 显示
5 ├──────────────────────────────────────────( MEND )
MEND                        // 主程序结束
```

```
数据块 DB1（V 存贮器）
//TD200 的参数块，存测量温度的信息，及温度极限的警告信息

B0      “TD”                //TD200 的标识符
B2      16#10               // 英语，尽快更新
B3      16#00               // 显示 20 个字符
B4      3                   // 信息数=3
B5      0                   // MB0 标志作 F 键
B6      0
B7      100                 //VB100=信息的开始
B8      0
B9      10                  //VB10=信息使能位的地址

B100    “Temperature=”      //Text1=信息 1 的 12 个字符
B112    “  ”                // 两个空格
B114    16#00               // MSB 字节的格式----不处理，不认可
B115    16#11               // LSB 字节的格式-----带 1 位十进制小数点的字
B116    16#00               // 填入 MSB 字的值
B117    16#00               // 填入 LSB 字的值
B118    16#DF               //Text2=2 个字符
B119    “C”

B120    “Temperature>”      //Text1=信息 2 的 12 个字符
B132    “  ”                // 两个空格
B134    16#00               // MSB 字节的格式----不处理，不认可
B135    16#11               // LSB 字节的格式-----带 1 位十进制小数点的字
B136    16#00               // 填入 MSB 字的值
B137    16#00               // 填入 LSB 字的值
B138    16#DF               //Text2=2 个字符
B139    “C”
```

```
B140    “Temperature<”          //Text1=信息 3 的 12 个字符
B152    “  ”                    // 两个空格
B154    16#00                   // MSB 字节的格式---不处理，不认可
B155    16#11                   // LSB 字节的格式----带 1 位十进制小数点的字
B156    16#00                   // 填入 MSB 字的值
B157    16#00                   // 填入 LSB 字的值
B158    16#DF                   //Text2=2 个字符
B159    “C”
```

37 S7-214 做从 Modbus RTU

概述

本示例包括一组子程序和中断程序，通过 S7-214 自由端口功能来产生从 Modbus RTU。它支持下列 Modbus 功能：

1. 读输出（线圈）
2. 读输入（触点）
3. 读保持寄存器（V 存贮器）
4. 读输入寄存器
5. 写单路输出
6. 写单路保持寄存器
15. 写多路输出
16. 写多路保持寄存器

程序结构

Modbus 协议驱动器包括一组子程序和中断程序，用来初始化和处理 Modbus 的请求。有两个程序必须加到用户主程序中。一个程序用来在第一次扫描时初始化 Modbus，另一个程序检查一个 M 位（M31.7）和处理已有的 Modbus 请求。第二个程序放在靠近用户主程序结束的地方（就在 MEND 之前），以便数据只在扫描结束时发生改变。

这些子程序和中断程序是：

SBR	50	初始化 Modbus RTU 驱动器
SBR	51	处理 Modbus 请求和传送响应
SBR	52	处理 Modbus 功能 1 和 2
SBR	53	处理 Modbus 功能 3 和 4
SBR	54	处理 Modbus 功能 5
SBR	55	处理 Modbus 功能 6
SBR	56	处理 Modbus 功能 15
SBR	56	处理 Modbus 功能 16
SBR	61	产生错误响应 2
SBR	62	初始化 CRC 表
SBR	63	计算 CRC
INT	120	静止线定时器到时处理
INT	121	在等待静止线定时器到时期间收到字符的处理
INT	122	接收请求的首字符（地址）
INT	123	接收请求的其余字符
INT	124	静止线定时器到时后结束请求
INT	125	发送完后复位静止线寻找

程序和注解

本程序允许一个和多个 S7-214 连接到主 Modbus。它利用 S7-214 的自由通信口功能来执行 Modbus RTU 协议。Modbus RTU 协议是一个主从协议。这就意味着一个网络配置包括一个主设备（一台主机）和一个或多个从设备。每个从设备有不同的地址。主机给一个从机发送请求，然后等待从机的响应。从机将回答此请求已收到或出现错误。如果请求没有正确接收到，则有象奇偶错或 CRC（校验和）错这一类的传送错误。此时从机将不响应，主机必须在等待一个适当的时间后重新发送请求。

Modbus RTU 协议是一个二进制协议。信息的开始由当前波特率下 3.5 字节时间的线上静止时间来指示。信息的结束也由线上相同的静止时间来指示。因为静止线时间是字符时间的若干倍，故它将根据波特率而改变。以下讲述的程序其静止线时间对应于 9600 波特率。如果波特率改变，静止线时间也必须改变，详见 SBR 50。

Modbus RTU 协议传送 8 位二进制字符，每个字符也包括一个起始位，一个或两个停止位（S7-214 提供一个停止位），一个可选择的奇偶校验位。以下讲述的程序设定 S7-214 为 9600 波特，偶校验。它可以通过在 SBR 50 里修改通信口设置而改变。

Modbus RTU 协议使用 CRC（循环冗余检验）来进行出错校验。本示例在检查收到的信息和传送响应时用一张 CRC 值表来加速 CRC 计算。这个 CRC 表在 Modbus 驱动器初始化期间在 V 存贮器高端产生，需要约 700ms 时间。它仅在第一次扫描时出现。

支持 Modbus 功能 1，2，3，4，5，6，15 和 16 的子程序已提供。如果一个特殊主机不使用所有这些功能，那么它们可被移去以提供更多的空间给用户程序。移去支持子程序和子程序的调用部分，就移去了该功能。调用部分都在 SBR 51 里。

通过子程序支持的功能有：

1、读单路/多路线圈（输出）状态
—返回任意数目输出（Q）的开/关状态。最大输出数由用户指定（见后面）

2、读单路/多路输入状态
—返回任意数目输入（I）的开/关状态。最大输入数由用户指定（见后面）

3、读单路/多路保持寄存器
—返回 V 存贮器的内容。在 Modbus 中保持寄存器为字。该区域起始为 V0，区域长度（字）由用户指定（见后面）

4、读单路/多路输入寄存器
—读模拟量输入。该功能实际上只返回从保持寄存器分出的 V 存贮器的内容。用户必须用程序命令读 AI 字，并在需要时把它们移到 V 存贮器中去。在该命令中用到的 V 存贮器区间由用户指定（见后面）。

5、强制单路线圈（输出）
—写到输出（Q）映象寄存器，输出实际上不是强制的，只是被写入。

6、写单路保持寄存器
—写一个字到 V 存贮器

15、强制多路线圈（输出）
—写多路输出（Qs）。开始输出必须以字节边界开始（即 Q0.0 或 Q2.0），而且所写的输出数必须是 8 的倍数。这在 Modbus 下不要求，但此处这样做是简化执行。输出实际上不是强制的，只是被写到输出映象寄存器里。

 西门子 版权所有

16、写多路保持寄存器
—写多个字到 V 存贮器中，在一次请求中最多能写 60 个字。

下面存贮器地址用来配置 Modbus 驱动器。这些地址在 SBR50 中被初始化，并可由用户修改，通过 Modbus 驱动器给主机，改变可用存贮器空间。
VW 3290 通过 Modbus 可达到的最大输入/输出数。它影响功能 1，2，5 和 15 的范围，缺省值为 64（I0.0 到 I7.7，Q0.0 到 Q7.7）。

VW 3294 通过 Modbus 功能 4 可达到的最大输入寄存器数，缺省值为 16。
VW 3296 当响应 Modbus 功能 4 时，AI 读到的 V 存贮器地址，缺省值为 VB2000
VW4095 Modbus 从机地址，缺省值为地址 1

下列 V 存贮器地址供 Modbus 驱动器使用，不能被用户更改

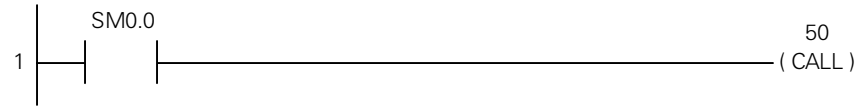
M31.7	已收到 Modbus 请求的标志
VB3300—VB3559	通信缓冲区
VB3560—VB3573	Modbus 的临时存贮器
VB3580—VB4091	CRC 数据表

子程序 50—63 保留给 Modbus 驱动器。
中断程序 120—127 保留给 Modbus 驱动器。
标签 254 和 255 保留给 Modbus 驱动器。

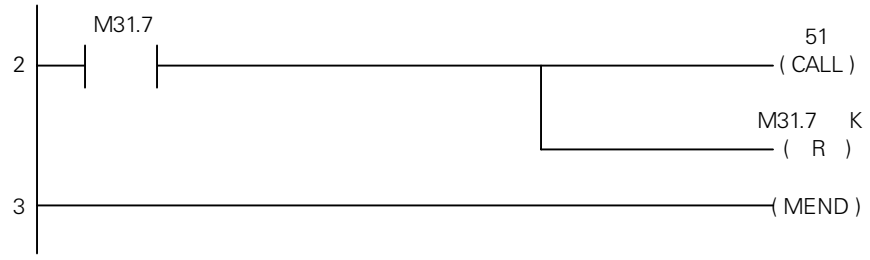
LAD(S7-MicroDOS)	STL(IEC)

主程序

```
// 如果是第一次扫描，那么就调用子程序 50 初始化 Modbut 通信。
// 该段程序应当放在梯形图程序顶部附近。
```



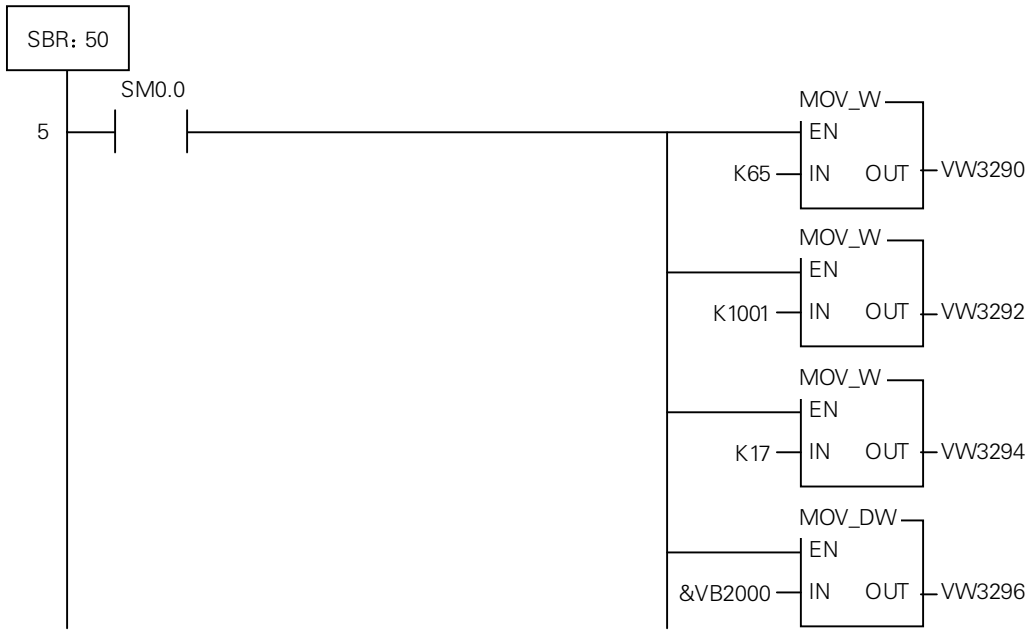

```
LD      SM0.1     // 如果是首次扫描 SM0.1=1
CALL    50        // 则初始化 Modebus 通信
// 检查是否已收到 Modbus 请求，如果已收到信息，那么由接受中断
// 子程序将 M31.7 置 1。
// 该段程序应当放在梯形图程序底部附近
```



```
LD      M31.7          // 如果有 Modbus 请求出现
CALL    51             // 则调用处理程序
R       M31.7，1       // 清除请求标志（M31.7=0）
MEND                   // 主程序结束
```

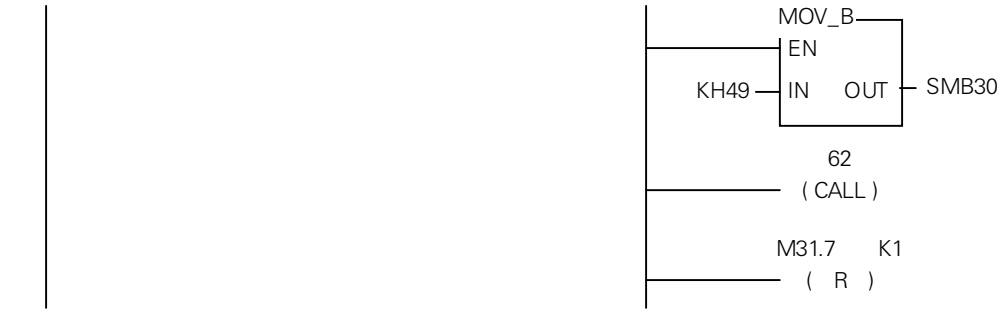
子程序

```
// 子程序 50
// 初始化 Modbus 驱动器通信口 0
// 注意：由于要初始化 CRC 表，这段初始化子程序需要大约 690ms 执行时间。
// 不要报警！
```

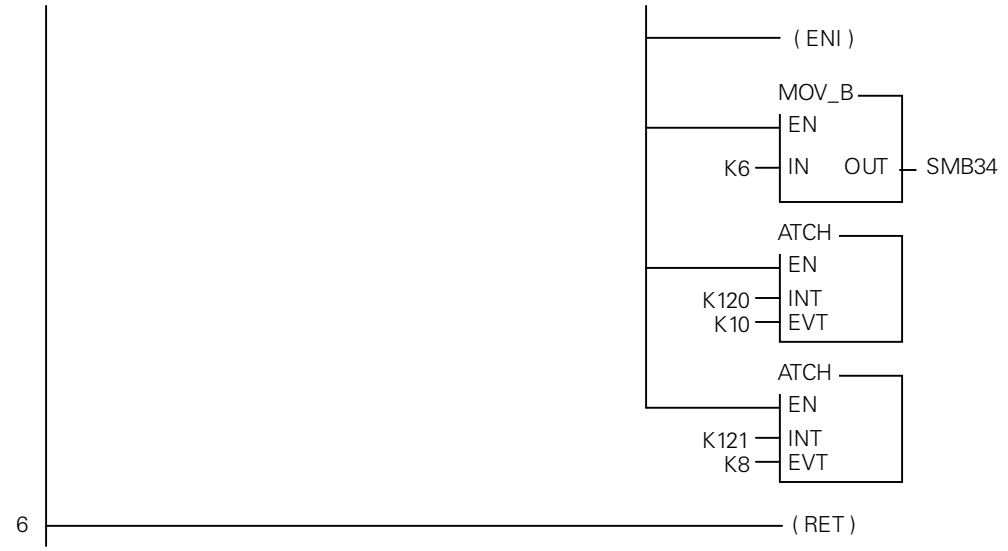



```
// 初始化不同数据类型的存贮器极限
// 注意：存贮的最大值必须比实际极限大 1
// 例如：允许 32 个输出，那就设定 33。

SBR     50             // 字程序 50
LD      SM0.0          // SM0.0 总为 1
MOVW    65，VW 3290    // 最大 IR=64 位
MOVW    1001，VW3292   // 最大 V 字=2000 字节（1000 字）
MOVW    17，VW3294     // 最大 AI 字=16 字
MOVD    &VB2000，VD3296 // V 存贮器的 AI 字首址
```


```
// 初始化 Modbus 地址
MOVB    1，VB4095      // Modbus 地址=1
```

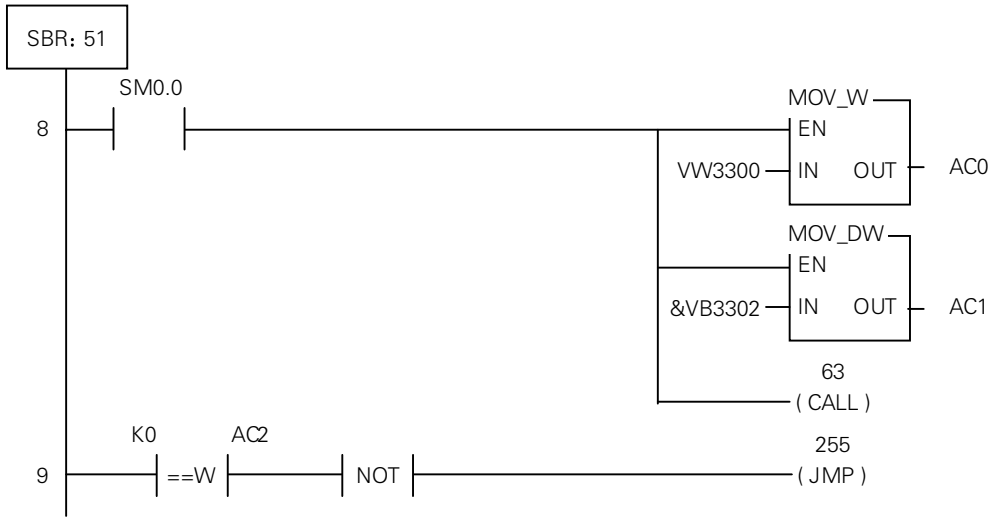



```
// 初始化通信端口，见 Step-7 编程参考手册对其它端口的配置（即波特率，奇偶）。
// RTU Modbus 要求用 8 位数据，可以修改波特率和奇偶校验。
// 注意：波特率设定为 38.4K，功能不会正常。
MOVB      16#49，SMB30                  // 9600 波特，每个符 8 位，偶校验

// 初始化 CRC 数据表，清除信息到达标志

CALL      62                            // 初始化 CRC 表
R         M31.7，1                      // 信息到达标志复位（M31.7=0）
```



```
// Modbus 信息由一条至少 3.5 字节时间的静止线刻划，当波特率为 9600 时是 4ms，所以静止
// 线时间为 6ms 以确保至少 5ms（4ms 静止线时间+1ms 收到一个字符的时间）。
// 注意：不同波特率的静止线定时器时间必须跟着更改。
//          300 波特          166ms
//          600               84
//          1200              43
//          2400              22
//          4800              12
//          9600              6
//          19.2k             5

ENI                           // 允许中断
MOB      6，SMB34             // 设定静止线定时器＞5ms
ATCH     120，10              // 开始寻找静止线，若静止线定时器到时，则调用中断程序 INT120
ATCH     121，8               // 在寻找静止线期间，如果收到一个字符，那就执行中断程序 INT121
RET                           // 返回
```

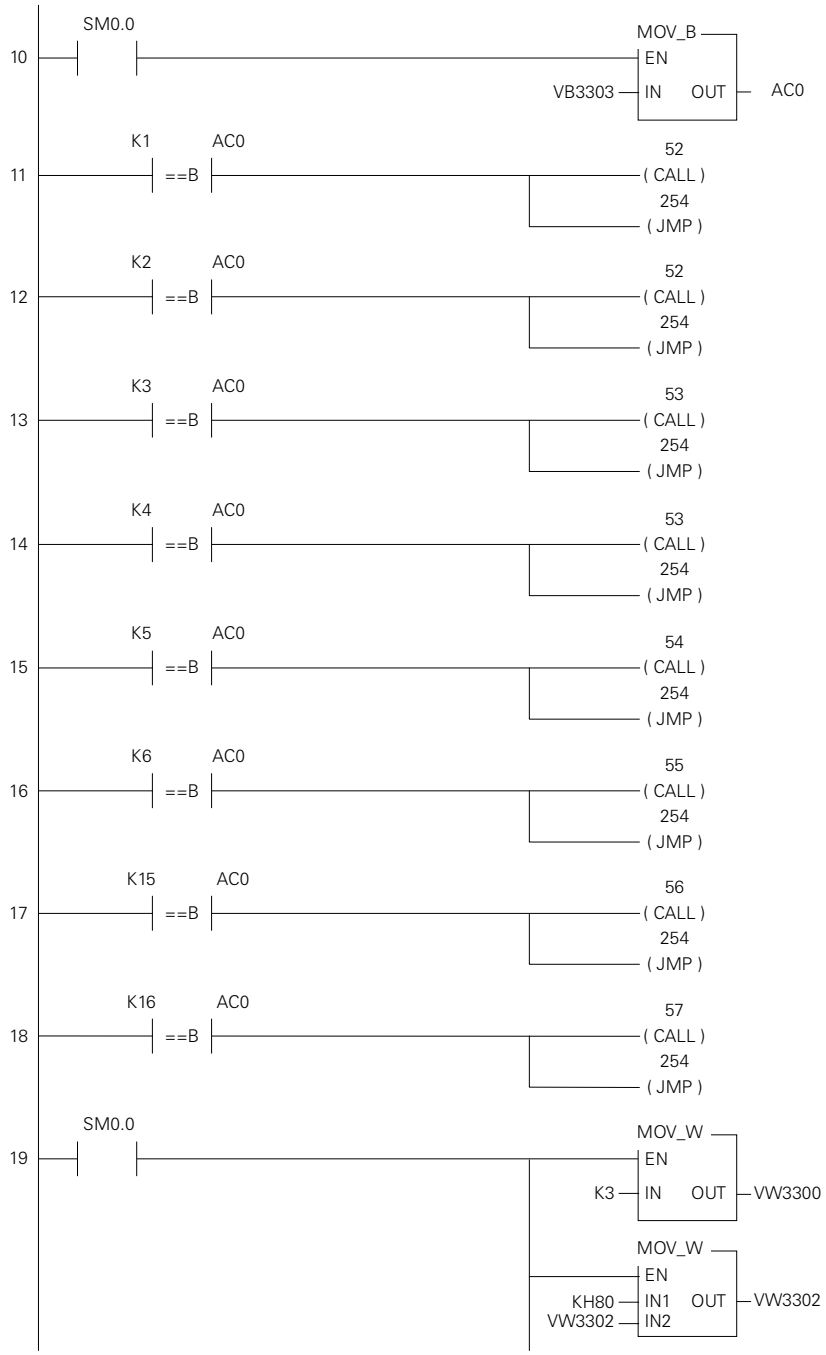
```
// 子程序 51
// 该子程序在正常的梯形图扫描期内处理 Modbus 请求。
// 计算收到信息的 CRC。由于收到信息中的 CRC 包含在计算之内，若没有接收错误，那么计算结果总是 0。
```

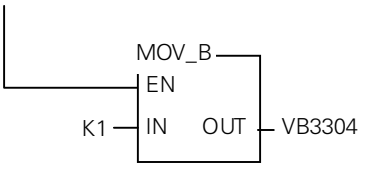



```
SBR      51               // 子程序 51
LD       SM 0.0           // SM0.0 总为 1
MOVW     VW3300，AC0      // 取缓冲区长度存入 AC0
MOVD     &VB3302，AC1     // 取缓冲区地址存入 AC1，用于 CRC 检查
CALL 63                   // 计算 CRC
LDW=     0，AC2           // 如果（计算 CRC！=0）
NOT                       // 那就表示有接收错误
JMP      255              // 跳转到 LBL 255
```

 西门子 版权所有

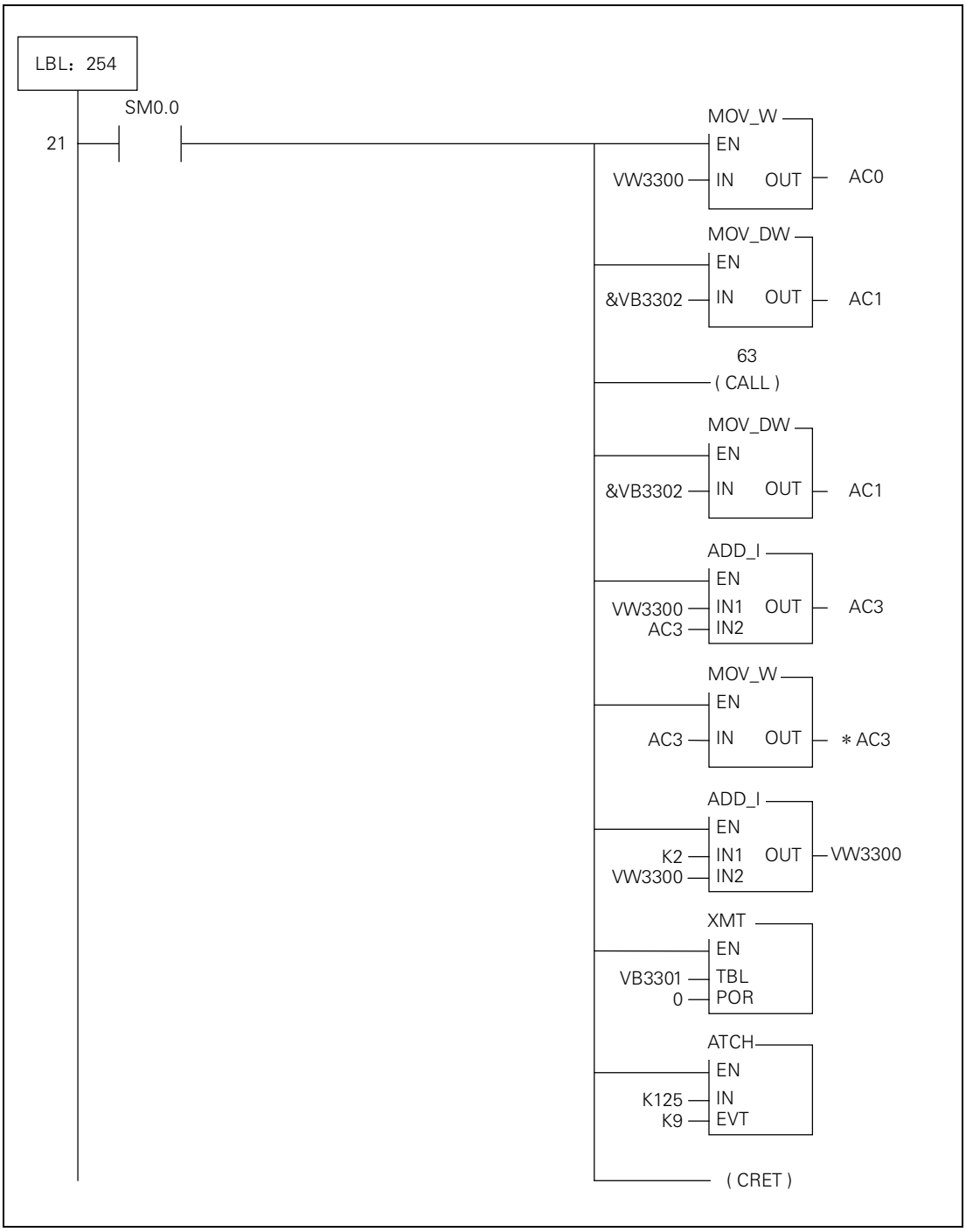
// 信息正常，决定哪个 Modbus 功能被请求。
// 在调用返回后总是执行紧随调用之后的跳步指令。
// 因为返回前子程序总设定 TOS 为 1。



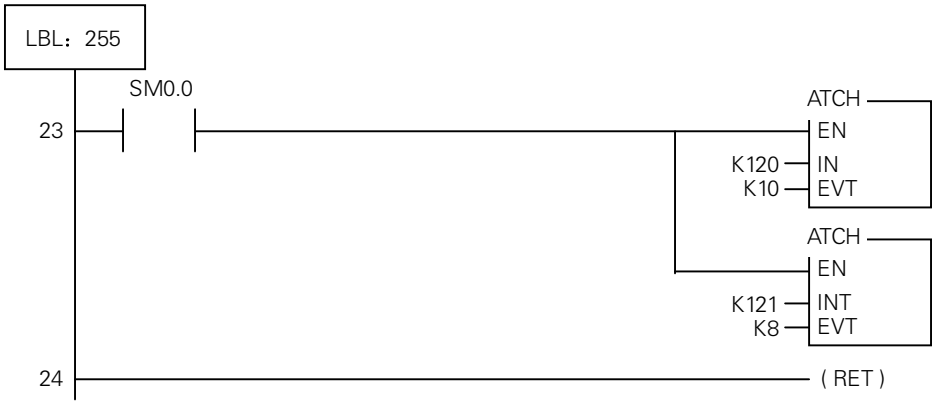



```
LD      SM0.0           // SM0.0 总为 1
MOVB    VB3303，AC0     // 从请求缓冲区中取功能号存入 AC0

LDB=    1，AC0          // 是功能 1?
CALL 52                 // 是，执行功能 1
JMP     254             // 然后跳到结束（LBL  254）

LDB=    2，AC0          // 是功能 2?
CALL 52                 // 是，执行功能 2
JMP     254             // 然后跳到结束

LDB=    3，AC0          // 是功能 3?
CALL 53                 // 是，执行功能 3
JMP     254             // 然后跳到结束

LDB=4，AC0              // 是功能 4?
CALL    53              // 是，执行功能 4
JMP     254             // 然后跳到结束

LDB=    5，AC0          // 是功能 5?
CALL 54                 // 是，执行功能 5
JMP     254             // 然后跳到结束

LDB=    6，AC0          // 是功能 6?
CALL    55              // 是，执行功能 6
JMP     254             // 然后跳到结束

LDB=    15，AC0         // 是功能 15?
CALL    56              // 是，执行功能 15
JMP     254             // 然后跳到结束

LDB=    16，AC0         // 是功能 16?
CALL    57              // 是，执行功能 16
JMP     254             // 然后跳到结束

LD      SM0.0           // 如果上面都不成立
MOVW    3，VW3300       // 装入错误响应长度
ORW     16#0080，VW3302 // 把功能 MS 位置 1（VB3303.7=1），表示出错
MOVB    1，VB3304       // 装入“功能不支持”代码（VB3304=1）
```

// 功能服务，首先计算响应的 CRC，然后发送响应。跟请求一样，把响应的长度也放在缓冲区的第一个字中。该长度不包括 CRC 的 2 个字节长度，因此在调用发送指令之前必须将长度增加 2。

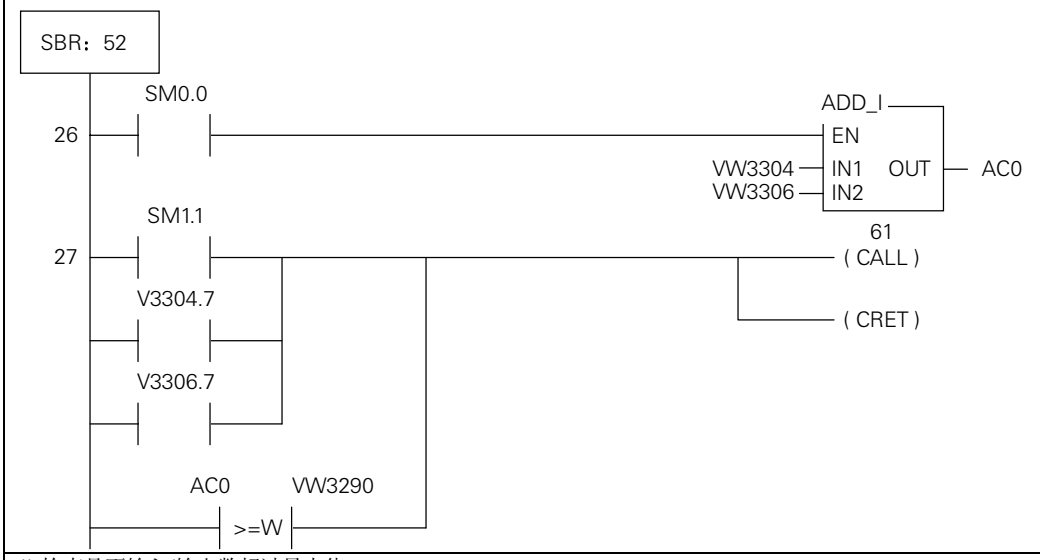

LBL：254
SM0.0
21
MOV_W
EN
VW3300
IN
OUT
AC0
MOV_DW
EN
&VB3302
IN
OUT
AC1
63
(CALL)
MOV_DW
EN
&VB3302
IN
OUT
AC1
ADD_I
EN
VW3300
IN1
OUT
AC3
AC3
IN2
MOV_W
EN
AC3
IN
OUT
* AC3
ADD_I
EN
K2
IN1
OUT
VW3300
VW3300
IN2
XMT
EN
VB3301
TBL
0
POR
ATCH
EN
K125
IN
K9
EVT
(CRET)

```
LBL     254             // 标签 254（跳转入口）
LD      SM0.0           // SM0.0 总为 1
MOVW    VW3300，AC0     // 取信息长度存入 AC0
MOVD    &VB3302，AC1    // 取缓冲区起始地址存入 AC1，用于 CRC 检验
CALL    63              // 计算 CRC

MOVD    &VB3302，AC3    // 取缓冲区起始地址
+I   VW3300，AC3        // 指向缓冲区尾
MOVW    AC2，*AC3       // 把 CRC 放到缓冲区
+I   2，VW3300          // 加 CRC 的两字节

XMT     VB3301，0       // 发送响应
ATCH    125，9          // 当发送完成时执行中断程序 INT125
CRET                    // 返回

// CRC 或长度出错处理
// 在下列情况下，如 CRC 出错，或者没有收到足够字节的处理信息，除了通信复位外。
// 寻找下一段信息，且使主机到时。
```

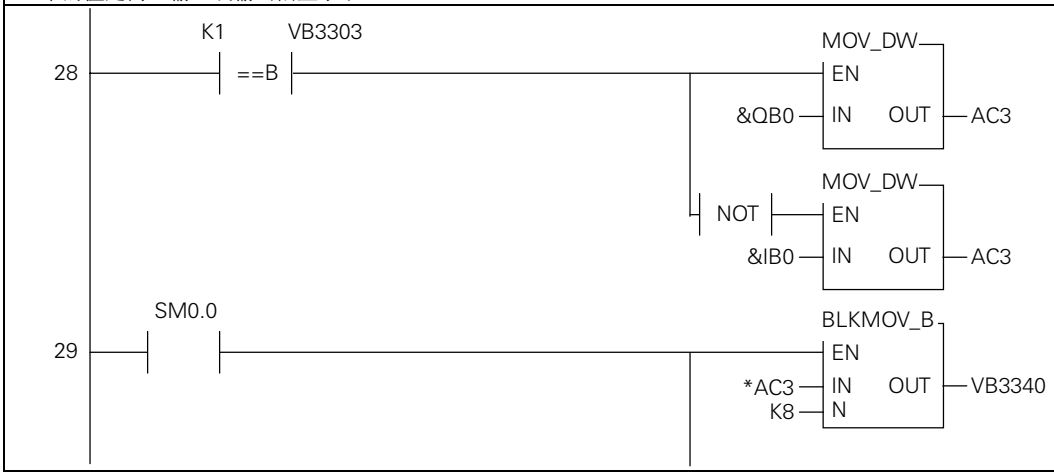


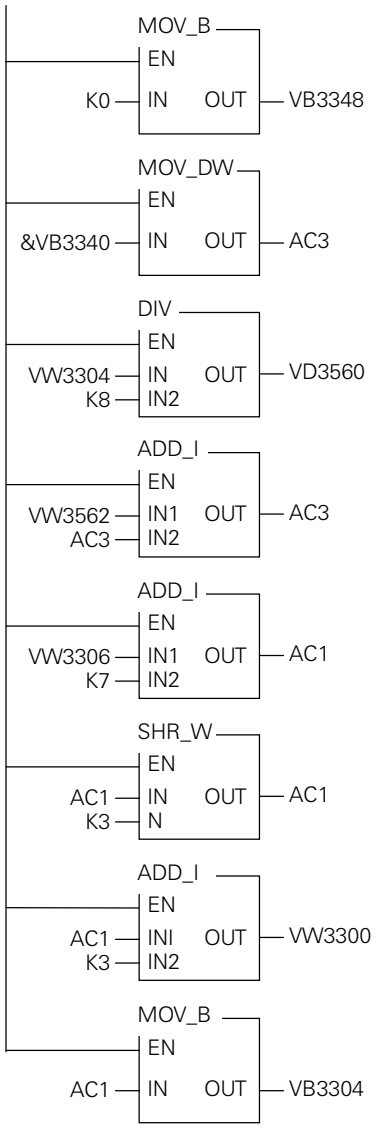

```
LBL     255             // 标签 255（跳转入口）
LD      SM0.0           // SM0.0 总为 1
ATCH120，10             // 开始寻找静止线，此期间
ATCH121，8              // 如果收到字符，那就执行中断程序 INT121
RET                     // 返回
```

```
// 子程序 52
// 该子程序支持 Modbus 功能 1 和 2，用来读一路或多路输入或输出状态。
// 响应位按每字节 8 位打包。第一个输入/输出请求位于第一个数据字节的 LS 位。如果请求的输入/输出数不能被 8
   除尽，则最后数据字节的 MS 位将被填充成 0，但不是当前完成。
// 请求的格式是：
// addr   01   start_bit（MSB,LSB）bit_count（MSB，LSB）
// start-bit 是第一个请求的输入/输出位（基于 0）
// bit-count 是请求的输入/输出数
// 响应的格式是：
// addr   01   byte_count data……
```




```
// 检查是否输入/输出数超过最大值
SBR      52          // 子程序 52
LD       SM0.0       // SM0.0 总为 1
MOVW     VW3304，AC0 // 取 start_bit 值
+I       VW3306，AC0 // ……加上 bit_count
LD       SM1.1       // 如果（超过最大值）或
O        V3304.7     //  （start_bit＜0）或
O        V3306.7     //  （bit_count＜0）或
OW＞=    AC0，VW3290 //  （last_bit  number＞max_IR）
CALL 61              // 则建立错误响应
CRET                 // 返回
// 决定工作区，然后在通信缓冲区内将输入或输出拷贝到该响应未用到的可擦除区，并将该区的下一字节置为 0。
   这考虑到下节使用的移位方法正常工作。
// 注意：设定一个映象寄存器为 8 字节。如果映象寄存器长度大于 8 字节，块移动指令常数必须改变。
// 取 start_bit 数，决定要拷贝的第一个字节的地址。除完后，VW3560 中的值是余数（移位计数），VW3562
// 中的值是商（输入或输出偏置字节）。
```

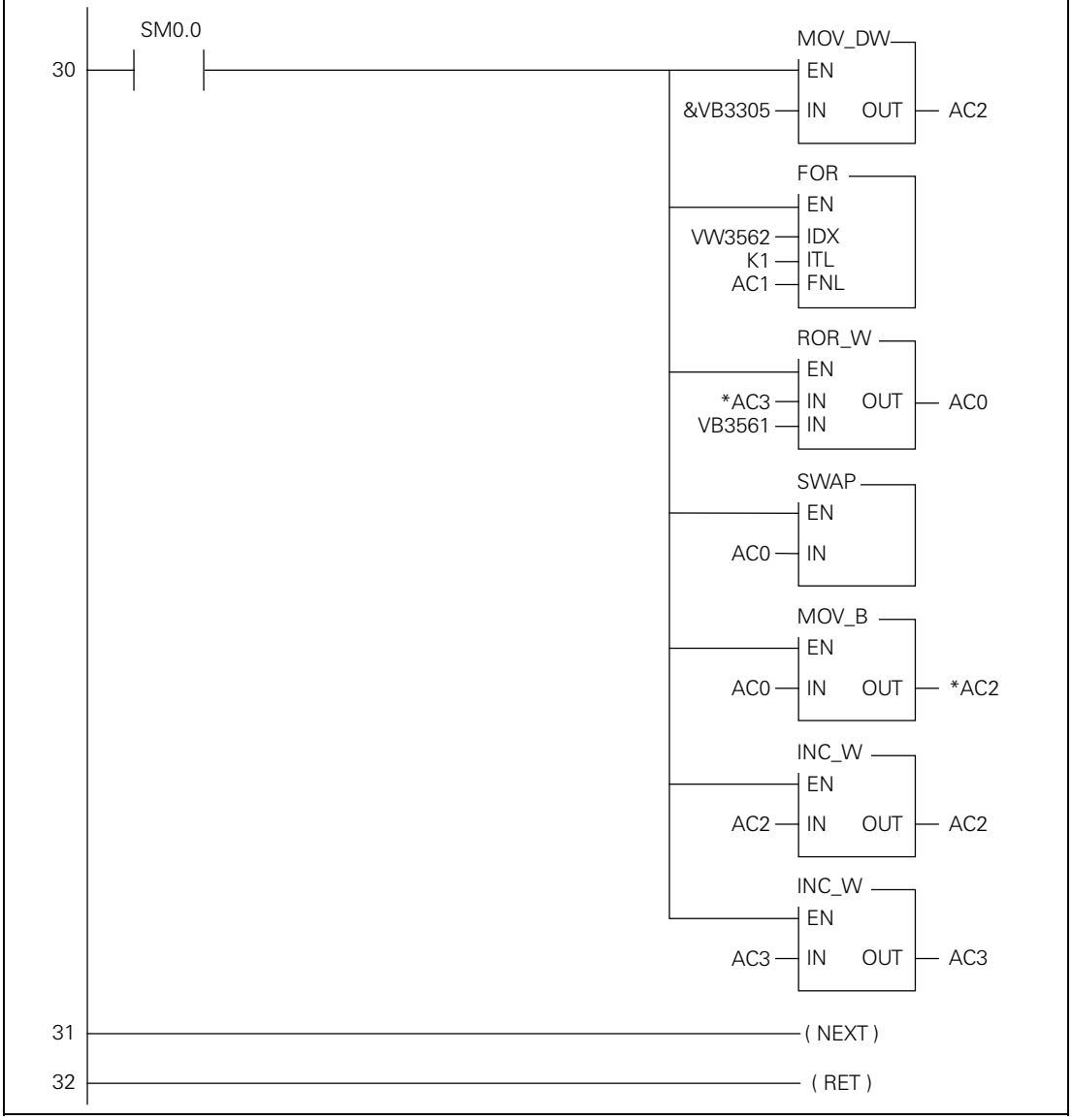





```
LDB=    1，VB3303           // 如果是功能 1（输出）
MOVD    &QB0，AC3           // 那么指向输出
NOT                         // 否则
MOVD    &IB0，AC3           // 指向输入
LD      SM 0.0              // SM0.0 总为 1
BMB     *AC3，VB3340，8     // 拷贝输入或输出到可擦写区（VB3340~VB3347）
MOVB    0，VB3348           // 把可擦写区的“下面”字节（VB3348）置 0
MOVD    &VB3340，AC3        // 源指针=可擦写区地址
MOVW    VW3340，VW3562      // 取 start_bit 值
DIV     8，VD3560           // start_bit/8
+I   VW3562，AC3            // 源指针加上偏置值

// 取 bit_count，决定在响应中需要送回的字节数。
```

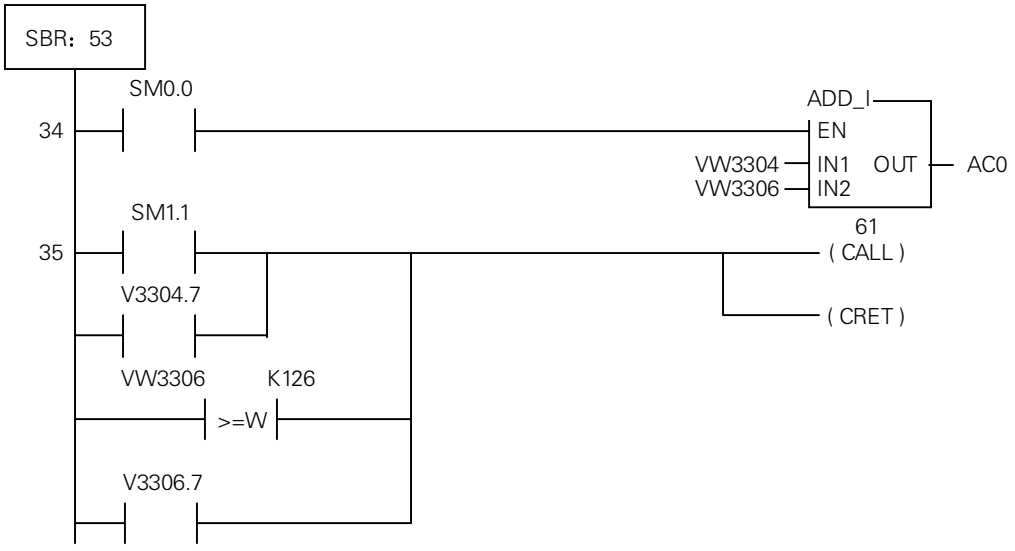
```
MOVW   VW3360，AC1          // 取 bit_count 存入 AC1
+I   7，AC1                 // AC1+7
SRW   AC1，3                // 除以 8（位/字节）

// 在响应缓冲区里装入响应缓冲区容量和 byte_count
MOVW   AC1，VW3300          // 取 byte_count
+I   3，VW3300              // 加头部 3 个字节
MOVB AC1，VB3304            // 装入字节计数器
```

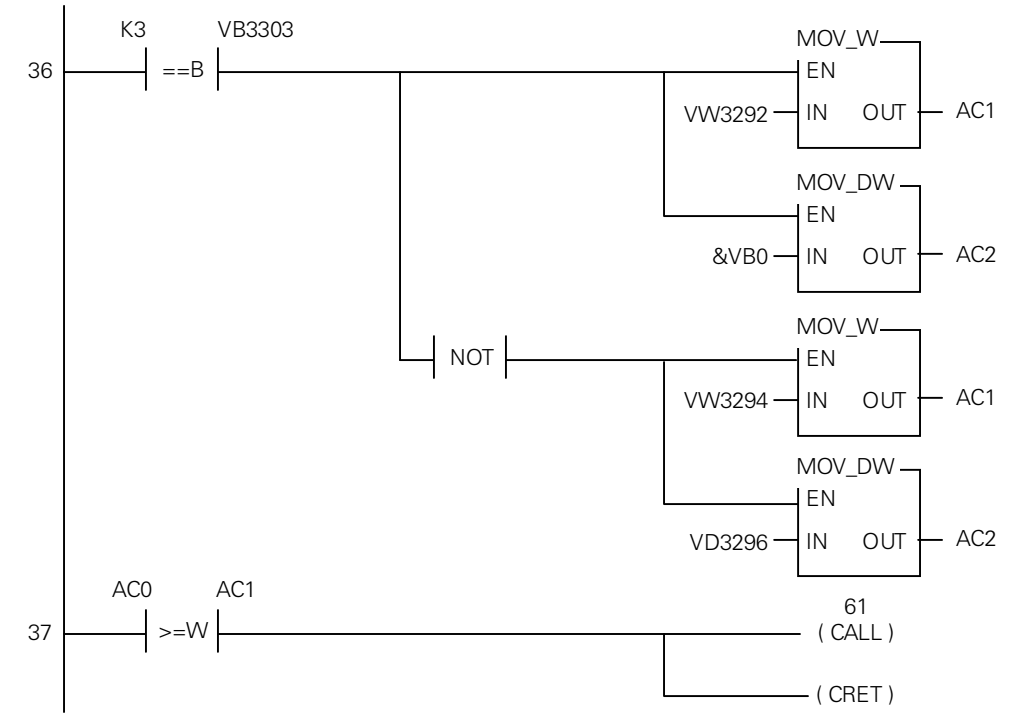
```
// 从映象寄存器中取出数据，移位后把它放在响应缓冲区里。

// 使用的方法是取出映象寄存器中一的个字，字中 MS 字节是我们需要的字节。为了移位计数而旋转字，
// 把“下一个”IR 字节的 LS 位移到我们所需要的字节的 MS 位。
// 当移位完成后，我们用交换指令 SWAP 取得的 LS 字节，然后把该字节存入响应缓冲区
```



```
LD      SM0.0          // SM0.0 总为 1
MOVD    &VB3305，AC2   // 装入响应缓冲区指针
FOR     VW3562，1，AC1 // 循环开始
MOVW    *AC3，AC0      // 取输出字
RRW     AC0，VB3561    // 右移
SWAP    AC0            // 把字节移到 LS 字节
MOVB    AC0，*AC2      // 存储字节
INCWAC2                // 缓冲区指针增加 1
INCWAC3                // 映象寄存器指针增加 1
NEXT                   // 循环结束
RET                    // 返回
```

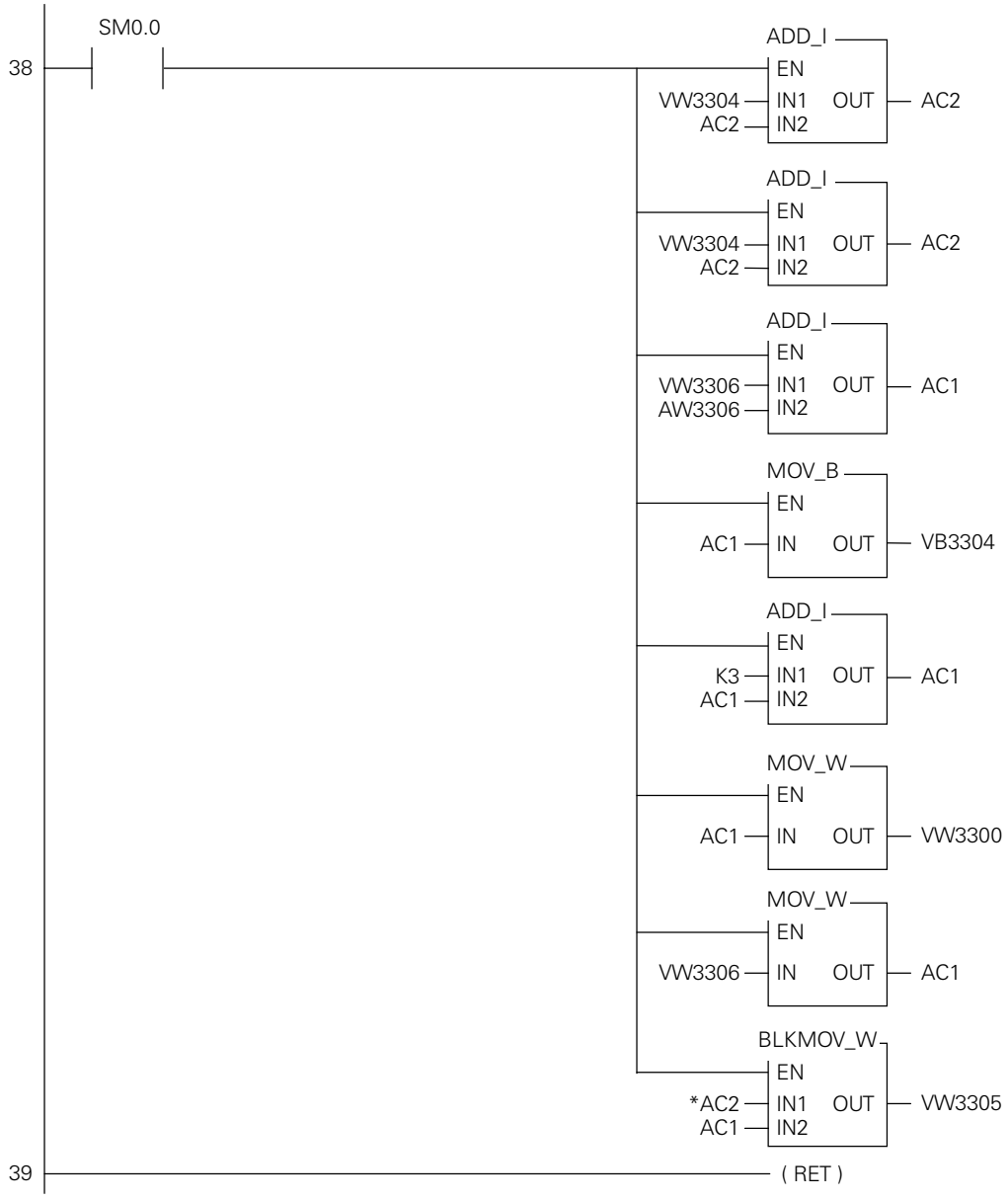
```
// 子程序 53
// 该子程序支持 Modbus 功能 3（读输出/保持寄存器）和功能 4（读输入寄存器）。
// 输出/保持寄存器被看作该 PLC 中 V 存贮器。输入寄存器被看作模拟量输入。
// 注意：
//      模拟量输入在该子程序中不直接读，因为 PLC 不允许直接读模拟量。
//      模拟量输入从 V 存储器指定的 VD3296 开始读。这个位置应该在子程序初始化时设定。用户
//      有责任把模拟量移到 V 存储器里。
// 请求的格式是：addr  03  start_word（MSB，LSB）  word_count（MSB，LSB）
// start_word 是请求的第一个字（基于 0）
// word_count 是请求字的数目。
// 响应的格式是：
//      addr  03  byte_count  data……
```






```
// 检查是否超出存储器极限
SBR       53                    // 子程序 53
LD        SM0.0                 // SM0.0 总为 1
MOVW      VW3304，AC0           // 取 start_word 值
+I        VW3306，AC0           // …加上 word_count

LD        SM1.1                 // 如果（超出）或者
O         V3304.7               // （start_word＜0）或者
OW＞=     VW3306，126           // （word_count＞125）或者
O         V3306.7               // （word_count＜0）
CALL 61                         // 则建立错误响应
CRET                            // 返回

LDB=3，VB3303                   // 如果是功能 3（V 存储器）
MOVW      VW3292，AC1           // 则取 V 存储器最大容量
MOVD      &VB0，AC2             // 源指针=V 存储器
NOT                             // 否则
MOVW      VW3294，AC1           // 取最大 AI 字容量
MOVD      VD3296，AC2           // 源指针=AI 存储器

LDW＞=    AC0，AC1              // 如果 start_word+word_count＞=存储器容量
CALL 61                         // 则建立错误响应
CRET                            // 返回
```

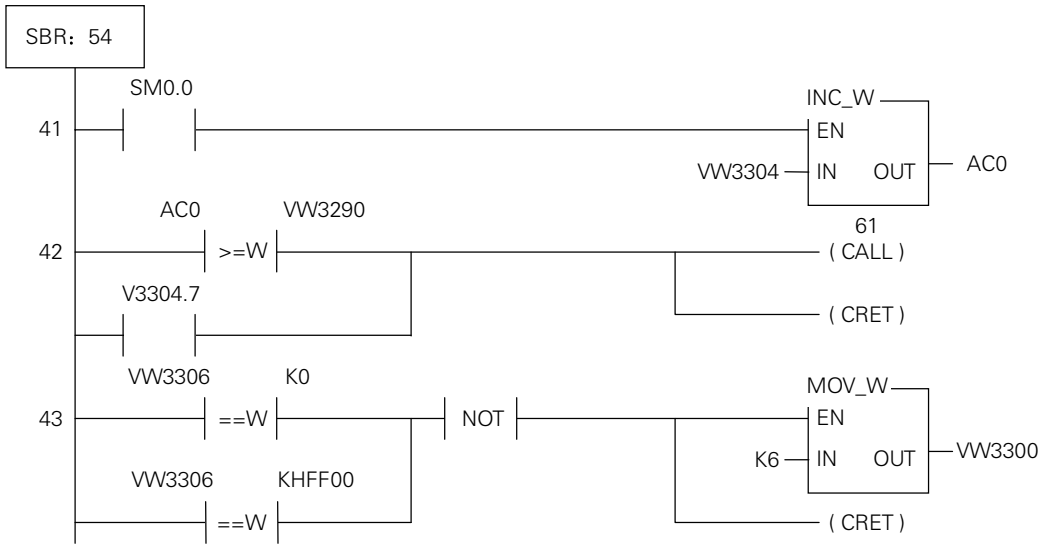
// 请求正确，取 start_word，加倍它作为字节偏置量加到源指针上，指向响应缓冲区然后把数据移入。




```
LD      SM0.0           // SM0.0 总为 1
+I      VW3304，AC2     // 把 start_word 加到源指针
+I   VW3304，AC2        // 把 start_word 乘以 2 加到源指针

MOVW    VW3306，AC1     // 取字数
+I      VW3306，AC1     // 字数乘以 2，用作字节计数
MOVB    AC1，VB3304     // 存储响应字节数
+I      3，AC1          // 加头部 3 个字节
MOVW    AC1，VW3300     // ……存储响应缓冲区容量
```

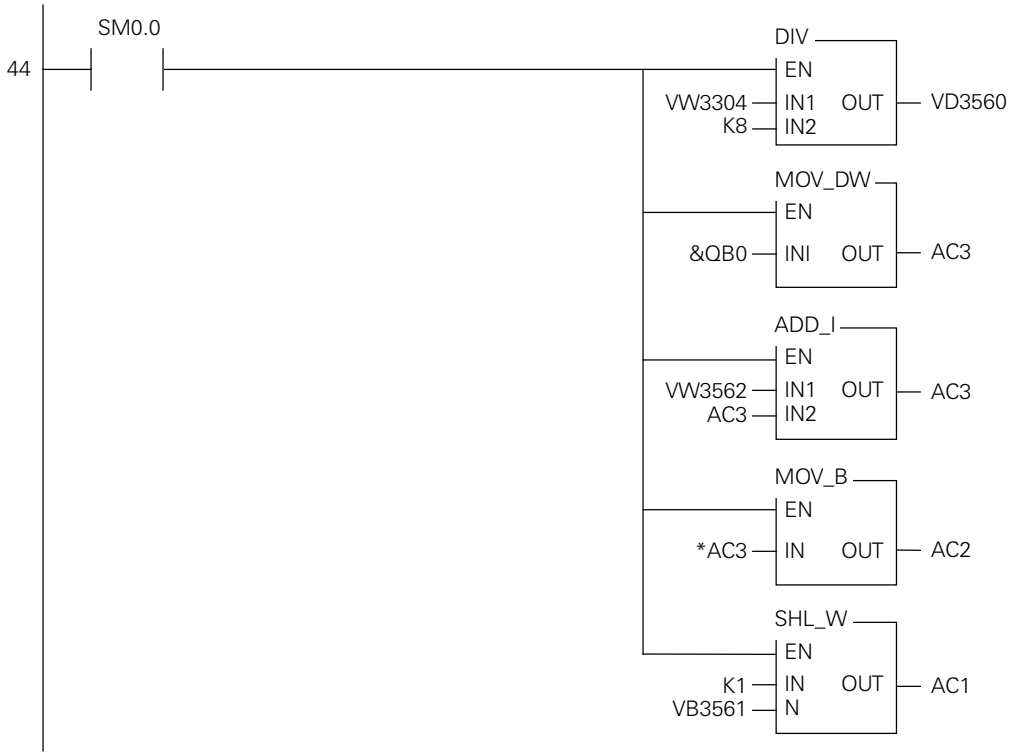
```
MOVW    VW3306，AC1          // 重新取出字计数
BMW*AC2，VW3305，AC1         // 从源区拷贝数据到缓冲区
RET                          // 返回
```

```
// 子程序 54
// 该子程序支持 Modbus 功能 5，用来强制单路输出开或关。
// 请求格式是：
// addr  05  output_bit（MSB，LSB）  data（FF00 或 0000）
// 数据 FF00 接通输出，数据 0000 关断输出，其它数不产生任何动作
// 响应信息是重传请求信息。
```

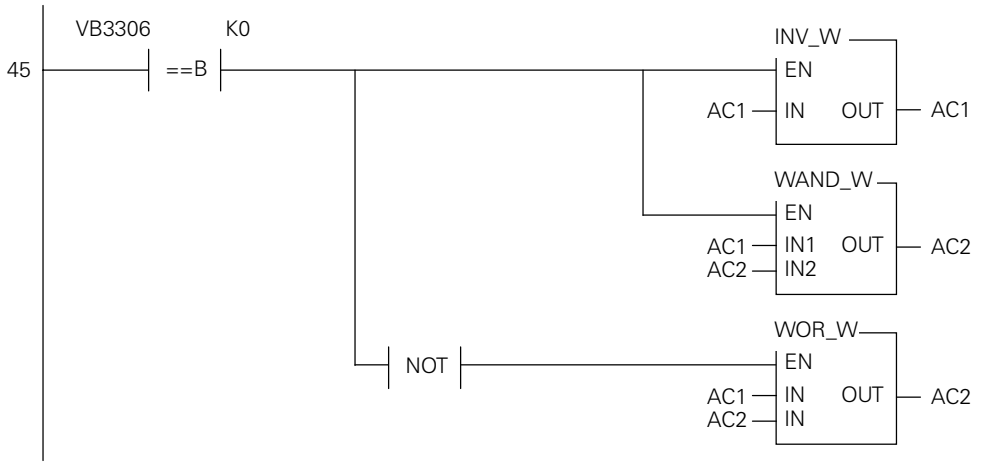



```
SBR     54                   // 子程序 54
LD      SM0.0                // SM0.0 总为 1
MOVW    VW3304，AC0          // 取 output_bit
INCW    AC0                  // …检查它

LDW>=   AC0，VW3290          // 若（output_bit＞max_IR_number）或者
O       V3304，7             // （output_bit＜0）
CALL    61                   // 则建立响应错误
CRET                         // 返回

// 检查数据是否是 FF00 或 0000，如果不是上述任一个，则不产生任何动作，只把请求返回给主机。
LDW=    VW3306，0            // 如果（数据！=0）或
OW=     VW3306，16#FF00      // （数据！=FF00）
NOT                          //
MOVW    6，VW3300            // 则设定响应长度
CRET                         // 返回
```

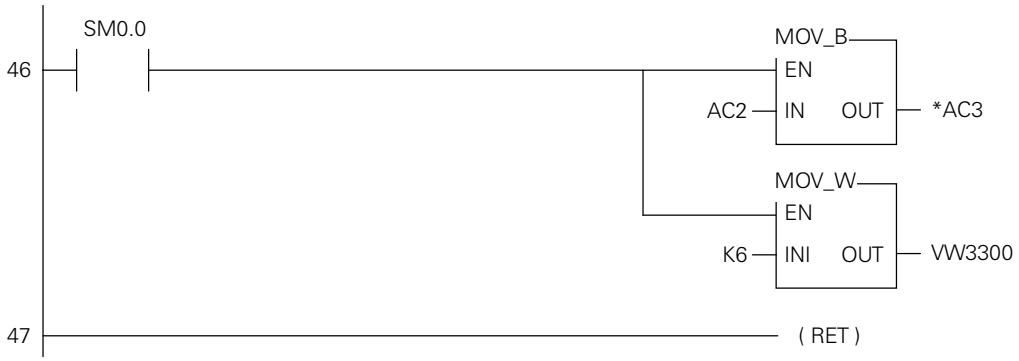
```
// 决定输出哪个字节和修改字节的哪个位。
// 该工作通过将输出位除以 8 来完成。商作为字节偏置量，余数作为位数。
// 除完后，在 VW3560 中的值是余数（位数），在 VW3562 中的值是商（输出偏置量）
```




```
LD      SM0.0           // SM0.0 总为 1
MOVW    VW3304，VW3562  // 取 output_bit
DIV     8，VD3560       // output_bit 除以 8

MOVD    &QB0，AC3       // 指向输出
+I      VW3562，AC3     // 加偏置，指向正确字节
MOVB    * AC3，AC2      // 取输出字节

MOVW    1，AC1          // 建立标志
SLW     AC1，VB3561     // …对正确的位
```

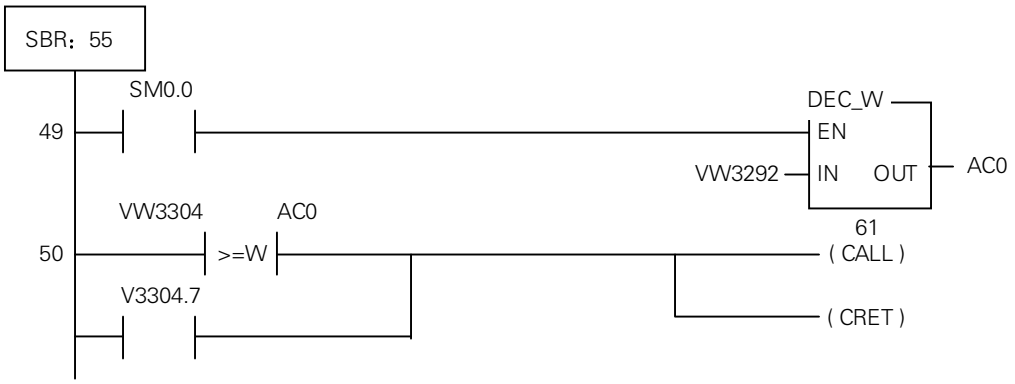



```
LDB=    VB3306，0       // 若数据== 0
INVW    AC1             // 清除标志
ANDW    AC1，AC2        // 清除位
NOT                     // 否则
ORW     AC1，AC2        // 置位
```



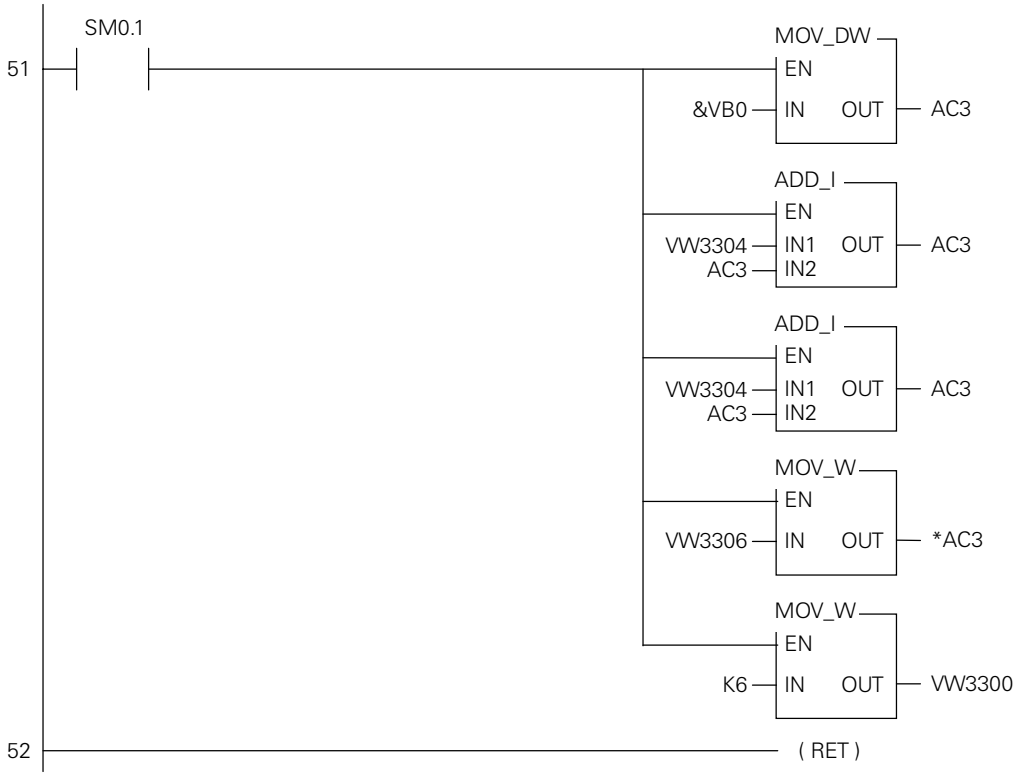

```
LD      SM0.0           // SM0.0 总为 1
MOVB    AC2，*AC3       // 恢复输出字节
MOVW    6，VW3300       // 设定响应长度
RET                     // 返回
```

```
// 子程序 55
// 该子程序支持 Modbus 功能 6，用来在 PLC 中写单路保持寄存器。在执行中保持寄存器被看作 V 存储器
// 请求的格式是：
// addr  06  start_word（MSB,LSB）  data（MSB，LSB）
// start_word 是第一个请求字（基于 0）
// 写到 PLC 的数据是字
// 响应的格式跟请求的格式一样：
// addr  06  start_word（MSB，LSB）  data（MSB，LSB）

// 检查是否超出存储器极限
```



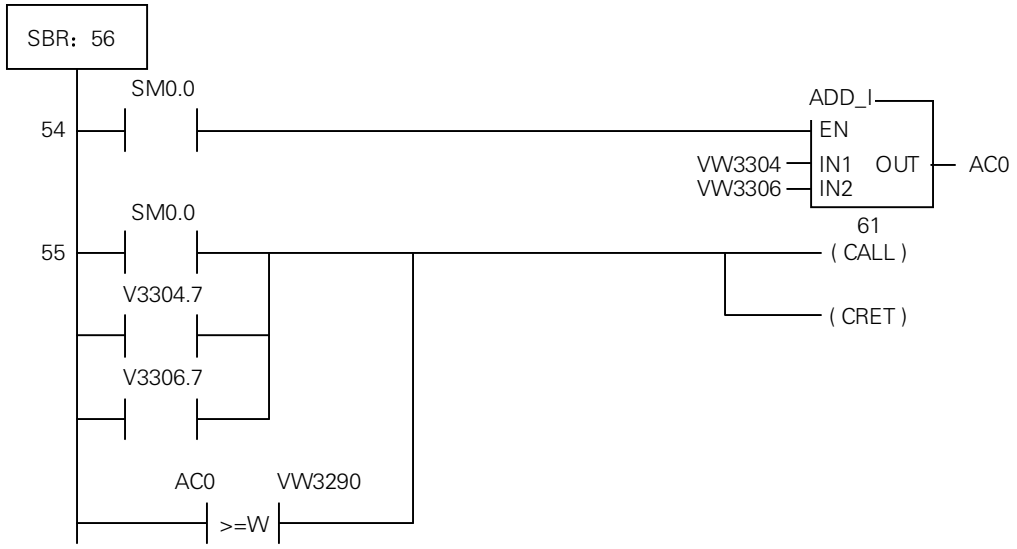

```
SBR      55                  // 子程序 55
LD       SM0，0              // SM0.0 总为 1
MOVW     VW3292，AC0         // 存储器容量 1
DECW     AC0                 // ……使之正好是存储器容量

LDW>=    VW3304，AC0         // 若（start_word>=存储器容量）或者
O        V3304，7            // （start_word<0）
CALL 61                      // 则建立错误响应
CRET                         // 返回
```

```
// 请求正确，取 start_word，加倍它作为字节偏置量加到源指针上。从请求中把数据移到 V 存储器
```



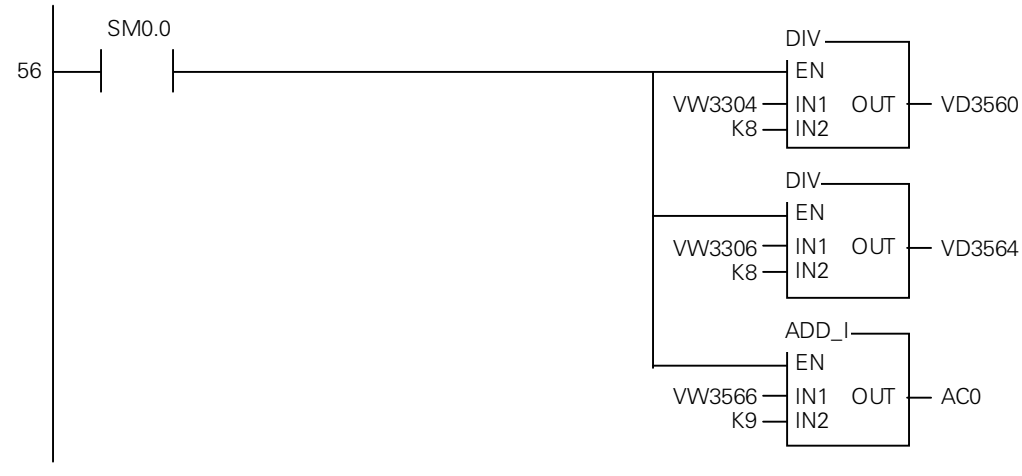

```
LD      SM0，0          // SM0.0 总为 1
MOVD    &VB0，AC3       // 源指针=V 存储器
+I      VW3304，AC3     // 加偏置量
+I      VW3304，AC3     // ……字节偏置量加倍
MOVW    VW3306，*AC3    // 把数据存到 V 存储器中
MOVW    6，VW3300       // 设定响应长度
RET                     // 返回
```

```
// 子程序 56
// 该子程序支持 Modbus 功能 15，用来强制多路线圈。
// 输出由 8 位组成一个字节。数据的第一个字节 LS 位被写到 Start_bit，后续位被写到紧接着的位置
// 注意：
// 如果起始位和位数不是 8 的倍数，那么执行将出错。对 STL 或 LAD 来说处理任意数目起始显得太复杂。
// 请求的格式是：
// addr  0F  start_bit（MSB，LSB）  bit_count（MSB，LSB）  byte_count data…
// start_bit 指第一个被写的输出，且是第一个数据字节的 LS 位
// bit_count 指被写的输出数
// byte_count 指数据字节数
// 响应的格式是：
// addr  0F  start_bit（MSB，LSB）  bit_count（MSB，LSB）
```



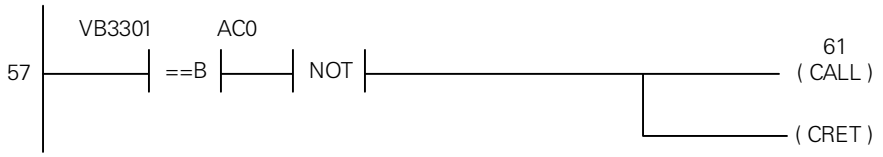

```
// 检查是否输出数超过最大值
SBR       56                    // 子程序 56
LD        SM0，0                // SM0.0 总为 1
MOVW      VW3304，AC0           // 取 start_bit 值
+I        VW3306，AC0           // ……加上 bit_count

LD        SM1.1                 // 如果（超过）或者
O         V3304.7               // （start_bit＜0）或
O         V3306.7               // （bit_count＜0）或
OW>=      AC0，VW3290           // （last_address＞max_IR）
CALL      61                    // 则建立错误响应
CRET                            // 返回
```

```
// 决定送入输出 IR 的字节和位偏置量，以及被写的全部和部分字节。
// 除完后，在 VW3560 里的值是余数（移位数），在 VW3562 里的值是商（输出字节偏置量）
// 除完后，在 VW3564 里的值是余数，在 VW3566 里的值是要写的全部字节数。
```



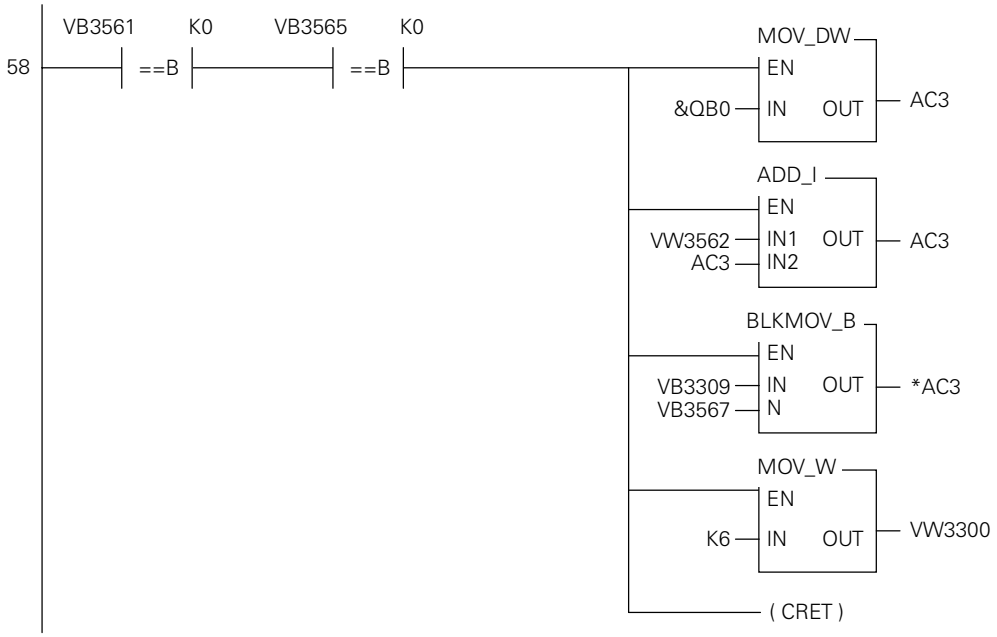
```
LD      SM0.0              // SM0.0 总为 1
MOVW    VW3304，VW3562     // 取 start_bit 值
DIV  8，VD3560             // start_bit 除以 8

MOVW    VW3306，VW3566     // 取 bit_count
DIV  8，VD3564             // bit_count 除以 8
MOVW    VW3566，AC0        // 取（bit_count 除以 8）
+I      9，AC0             // 加头部 9 个字节
```

// 确认主机发送足够数据来请求。缓冲区长度等于（bit_count/8）+9 字节的头部



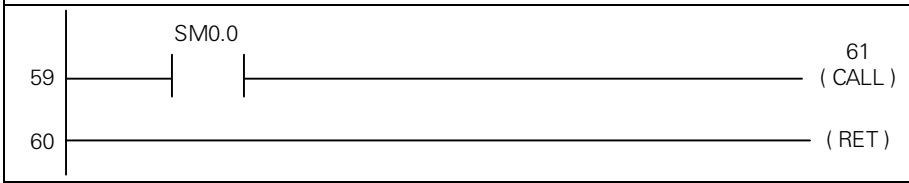

```
LDB=    VB3301，AC0        // 如果（计算长度不等于收到的长度）
NOT                        //
CALL    61                 // 则表示出错
CRET                       // 返回
```

// 决定我们是否建立了一个“好”（nice）帧，这里 start_bit 是字节的第一位，bit_count 是整个字节数。如果不是这种情况，则返回错误给主机。



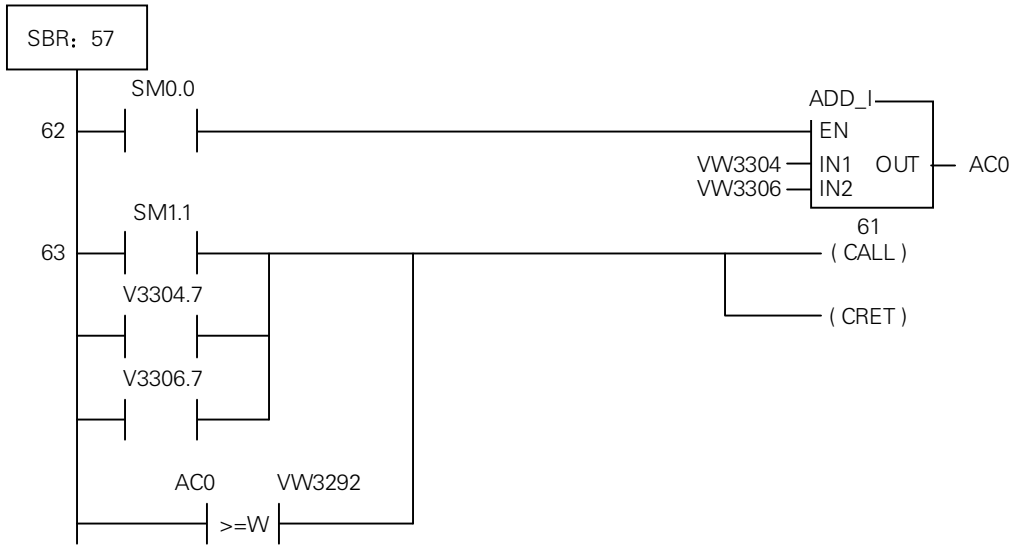

```
LDB=    VB3561，0          // 若（start_bit 是字节的 0 位），
AB=     VB3565，0          // 且（整个字节数= =TRUE）
MOVD    & QB0，AC3         // 则指向输出
+I      VW3562，AC3        // 加上偏置量给起始输出
BMB     VB3309，*AC3，VB3567 // 拷贝请求数据给输出
MOVW    6，VW3300          // 设定响应长度
CRET                       // 返回
```

// 这不是一个“好”（nice）帧，因此退出请求




```
LD          SM0，0                  // 否则，
CALL 61                             // 表示出错
RET                                 // 返回
```

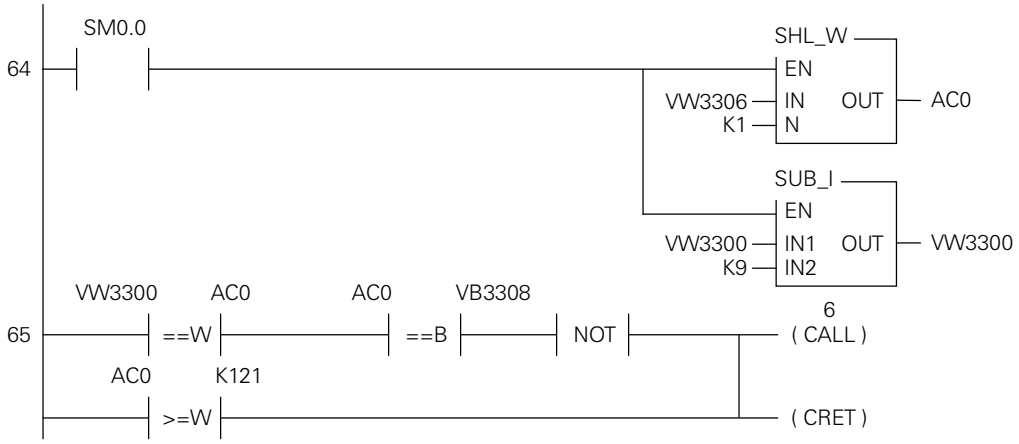
```
// 子程序 57
// 该子程序支持 Modbus 功能 16，用来写最多 60 个保持寄存器给 PLC，在执行中保持寄存器被看作 V 存贮器。
// 注意：在 Modbus 说明书里最大只能有 60 个寄存器（120 个字节）
// 请求的格式是：
   addr   10   start_word（MSB,LSB）   word_count（MSB，LSB）   byte_count   data
// 响应的格式是：
// addr   10   start_bit（MSB，LSB）   word_count（MSB，LSB）
```



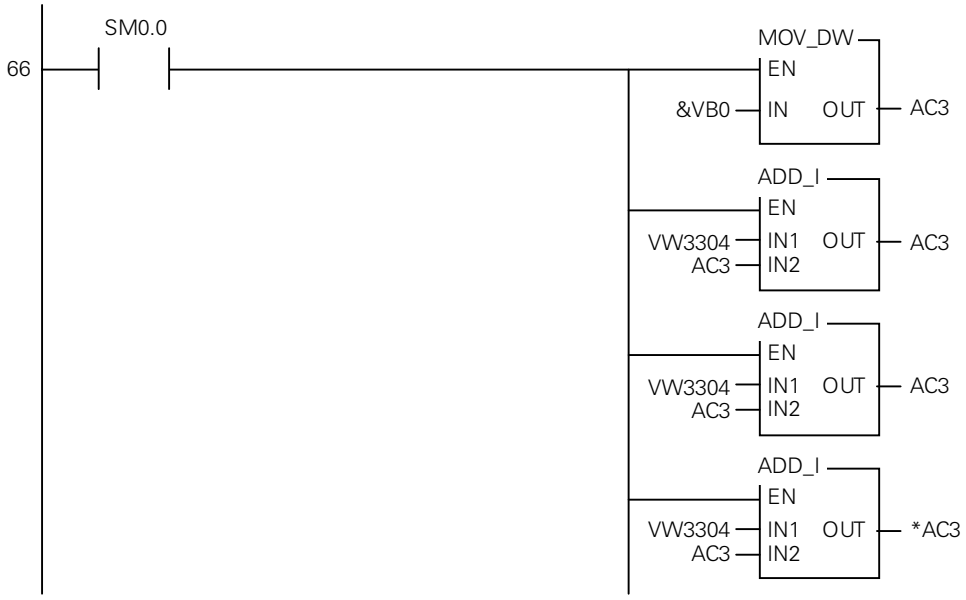
// 通过 word_count 和 start_word 计算出最后的请求字地址。如果最后的地址超过最大的 V 存贮器地址，那么返回错误。

```
SBR         57                      // 子程序 57
LD          SM0，0                  // SM0.0 总为 1
MOVW        VW3304，AC0             // 取 start_word
+I          VW3306，AC0             // ……加上 word_count

LD          SM1，1                  // 如果（超出）或
O           V3304.7                 // （start_word＜0）或
O           V3306.7                 // （word_count＜0）或
OW>=        AC0，VW3292             // （最后地址>=存贮容量）
CALL        61                      // 则建立错误响应
CRET                                // 返回
```

```
// 比较收到的字节数和信息中的 word_count，确定收到足够的数据字节。收到的字节数应当等于（word_count×
   2）+9 个头部字节。同时也验证字数不超过功能允许的最大值 60（120 个字节）。
// 另外还验证 word_count 乘以 2 等于 byte_count。所有这些加倍校验有点多余，但必须完成。
```

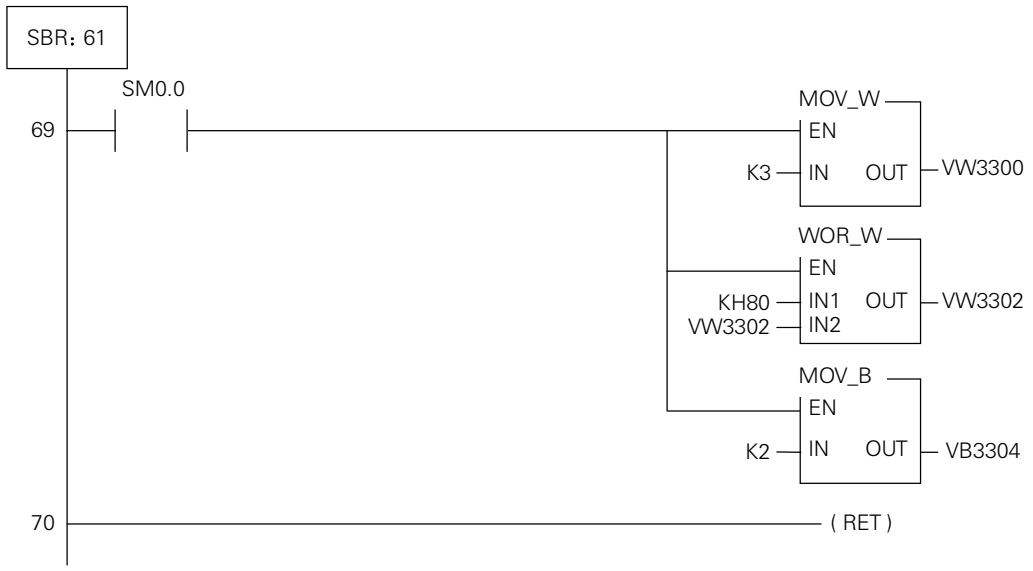



```
LD      SM0，0          // SM0.0 总为 1
MOVW    VW3306，AC0     // 取字计数
SLW     AC0，1          // word_count 乘以 2
－I     9，VW3300       // 收到字节数减去 9

LDW=    VW3300，AC0     // 如果（word_count 乘以 2！=收到字节数减 9）或
AB=     AC0，VB3308     // （word_count 乘以 2！=byte_count）或
NOT                     //
OW>=    AC0，121        // （计数>60 字+头部）
CALL    61              // 则建立出错响应
CRET                    // 返回
// 请求正确，取 start_word，加倍它作为字节偏置量加到原指针上。从请求中把数据移到 V 存储器。
```



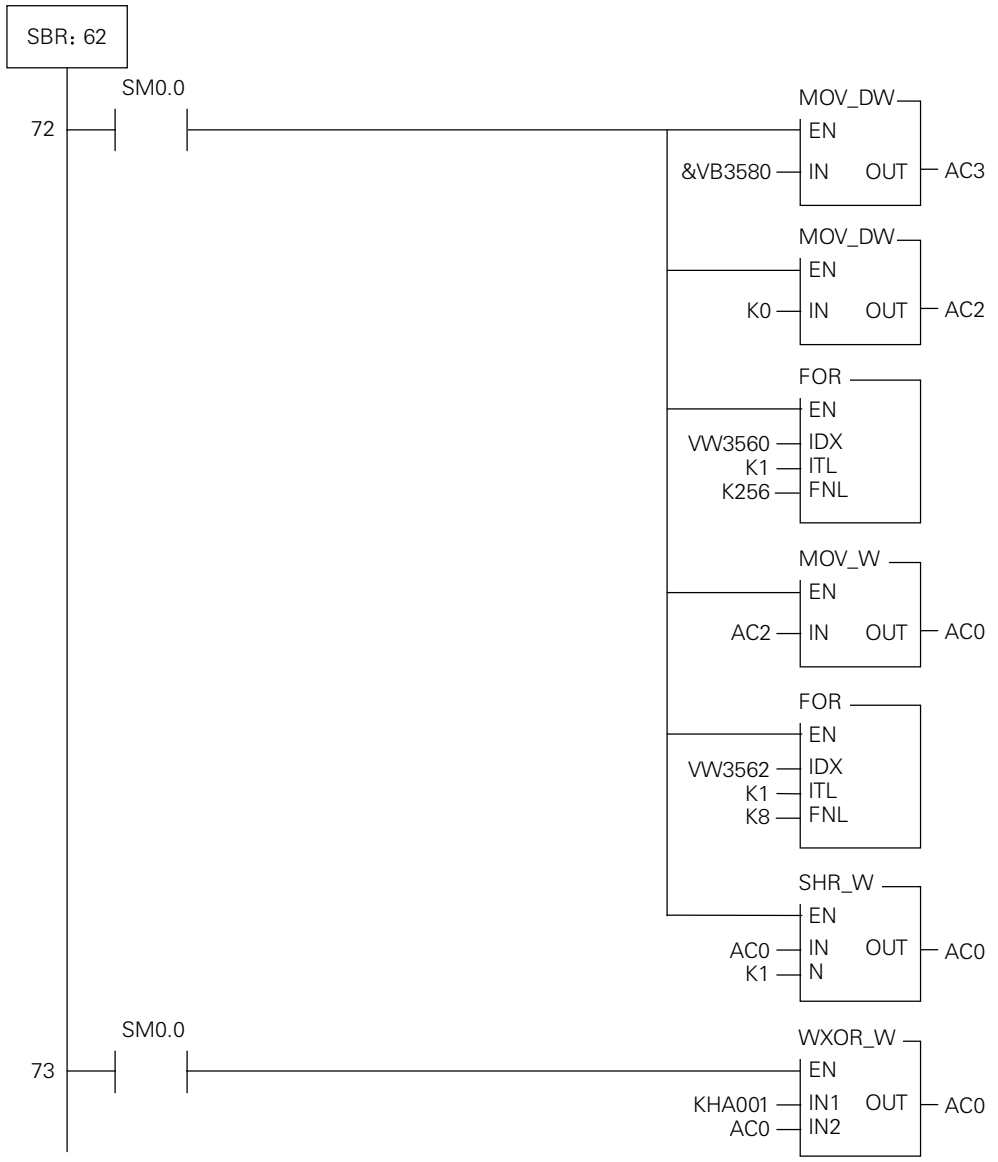

```
LD      SM0，0              // SM0.0 总为 1
MOVD    &VB0，AC3           // 源指针=V 存储器
+I      VW3304，AC3         // 加偏置量
+I      VW3304，AC3         // …2 倍字节偏移量

BMW     VW3309，*AC3，VB3307 // 存储数据到 V 存储器
MOVW    6，VW3300           // 设定响应长度
RET                         // 返回
```

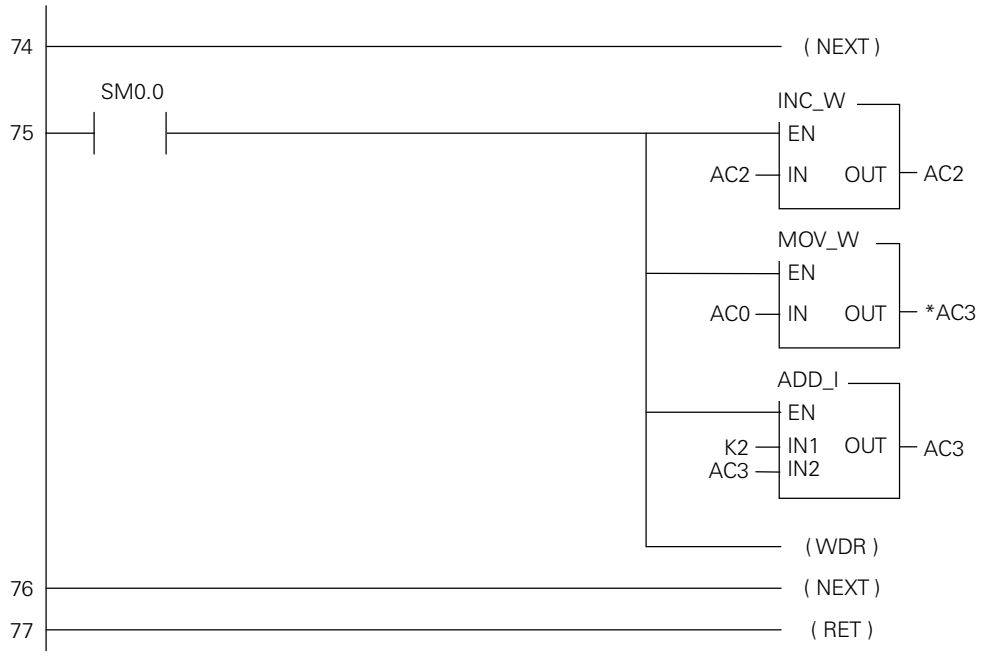
```
// 子程序 61
// 该子程序将建立响应缓冲区，并装入 Modbus 异常码 2。
```




```
SBR     61              // 子程序 61
LD      SM0.0           // SM0.0 总为 1
MOVW    3，VW3300       // 装入错误响应长度
ORW     16#0080，VW3302 // 把功能 MS 位置 1，表示出错
MOVB    2，VB3304       // 装入异常码 2
RET                     // 返回
```

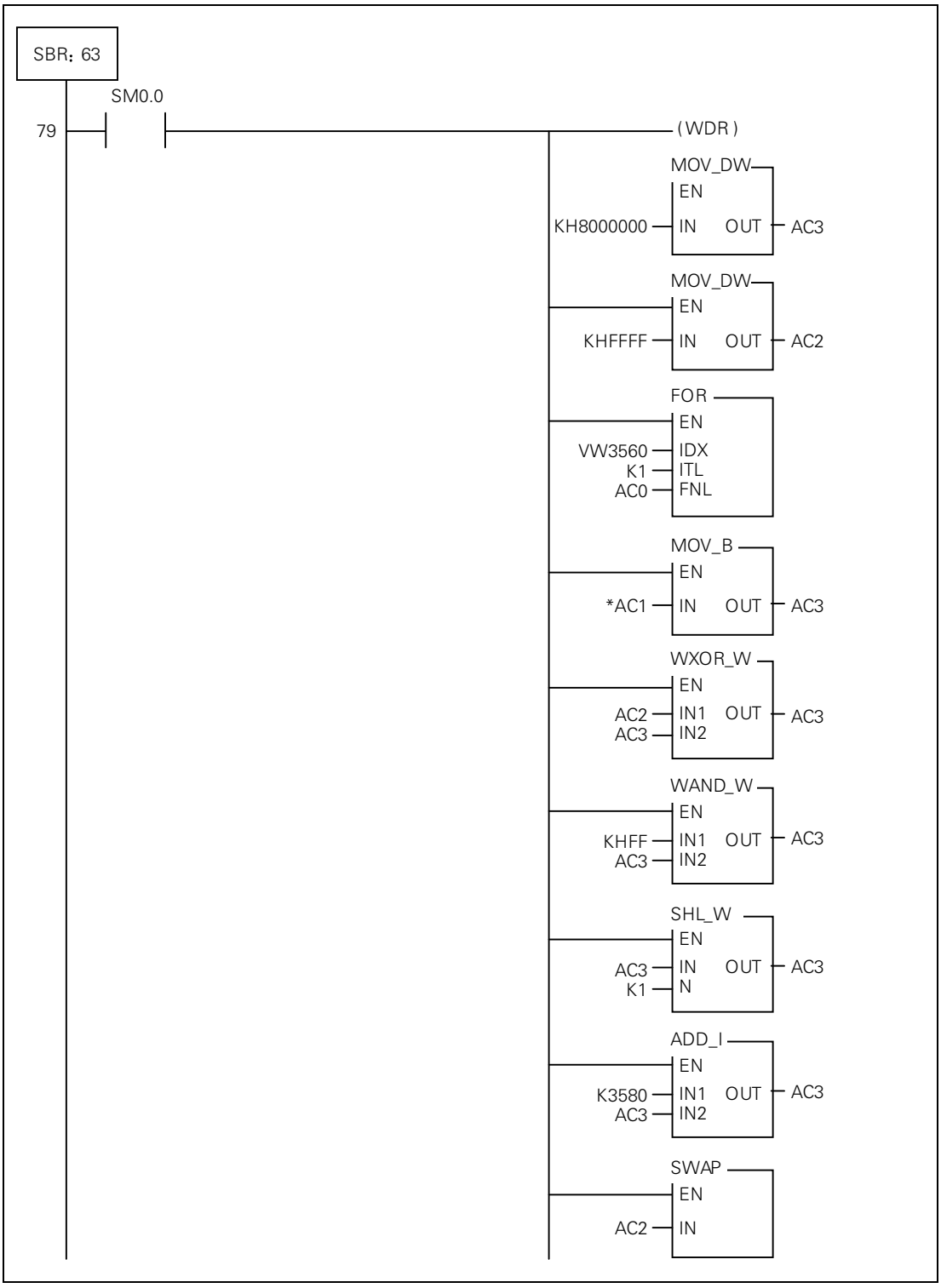
```
// 子程序 62
// 该子程序产生 CRC 表，用来计算信息的 CRC 值。
// 多项式为：X16+X15+X2+1
```

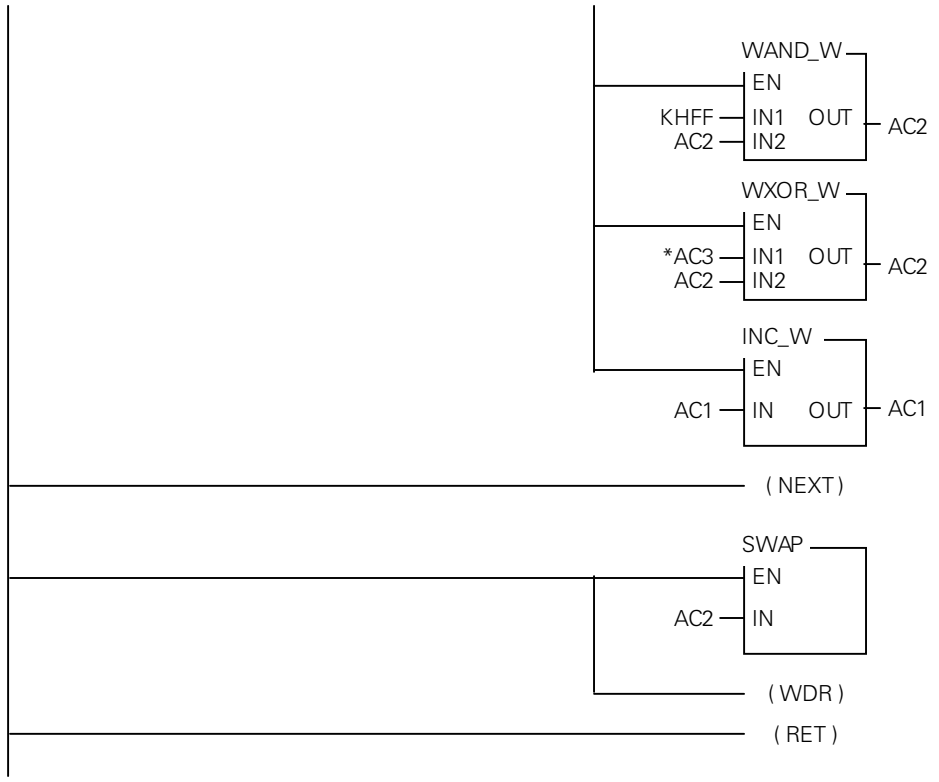





```
SBR     62              // 子程序 62
LD      SM0.0           // SM0.0 总为 1
MOVD    &VB3580，AC3    // 装入表指针
MOVD    0，AC2          // 装入索引（index）

FOR     VW3560，1，256  // 外循环开始
MOVW    AC2，AC0        // 值=索引（index）
FOR     VW3562，1，8    // 内循环开始（由于每字节 8 位，所以循环 8 次）
SRW     AC0，1          // 移出 LS 位
LD      SM1.1           // 如果移出位是 1
XORW    16#A001，AC0    // 则执行异或
NEXT                    // 内循环结束

LD      SM0.0           // SM0.0 总为 1
INCW    AC2             // 索引（index）加 1
MOVW    AC0，*AC3       // 存储表字
+I      2，AC3          // 表指针加 2
WDR                     // 复位看门狗
NEXT                    // 外循环结束
RET                     // 返回
```

```
// 子程序 63
// 该子程序用快速表查找方法计算出信息的 CRC 值

// 输入：   AC0          信息长度
//          AC1          信息指针
// 输出：   AC2          CRC 值的 LS 字
```

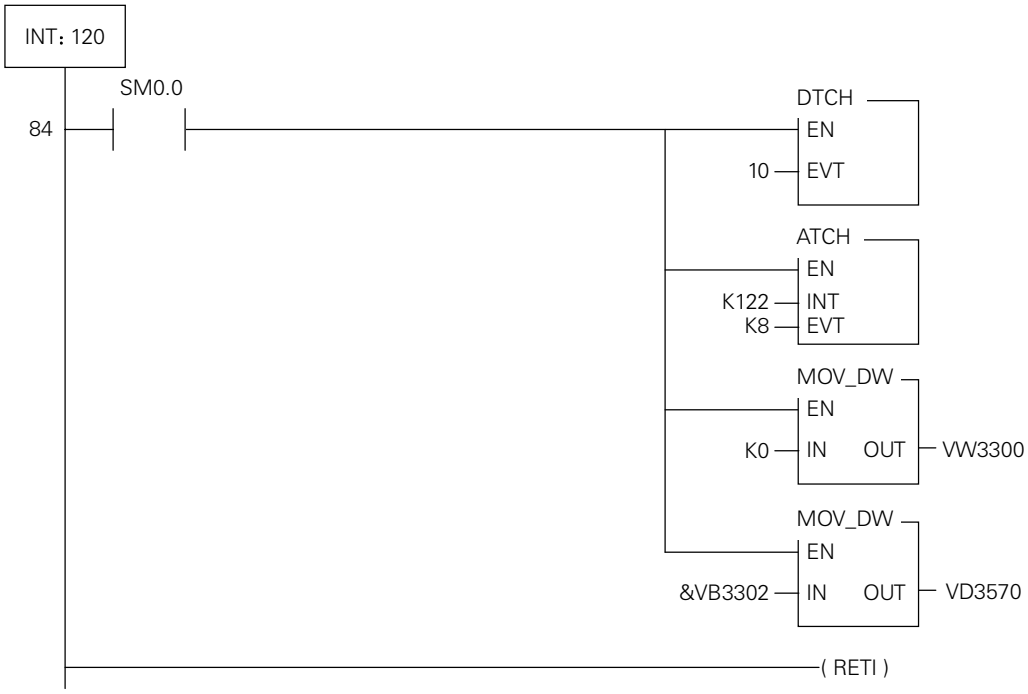

SBR: 63
SM0.0
79
(WDR)
MOV_DW
EN
KH8000000
IN
OUT
AC3
MOV_DW
EN
KHFFFF
IN
OUT
AC2
FOR
EN
VW3560
IDX
K1
ITL
AC0
FNL
MOV_B
EN
*AC1
IN
OUT
AC3
WXOR_W
EN
AC2
IN1
OUT
AC3
AC3
IN2
WAND_W
EN
KHFF
IN1
OUT
AC3
AC3
IN2
SHL_W
EN
AC3
IN
OUT
AC3
K1
N
ADD_I
EN
K3580
IN1
OUT
AC3
AC3
IN2
SWAP
EN
AC2
IN




```
SBR     63                  // 子程序 63
LD      SM0.0               // SM0.0 总为 1
WDR                         // 复位看门狗
MOVD    16#08000000，AC3    // 清除临时寄存器
MOVD    16#0000FFFF，AC2    // 初始化 CRC 值为 0xFFFF

FOR     VW3560，1，AC0      // 循环开始
MOVB    *AC1，AC3           // 取信息字节
XORW    AC2，AC3            // 数据与 CRC 异或
ANDW    16#00FF，AC3        // 只保留 LS 字节
SLW     AC3，1              // 把索引（index）置换成字表
+I      3580，AC3           // 加表起始地址（VB3580）
SWAP    AC2                 // 交换 CRC 字节
ANDW    16#00FF，AC2        // 只保留 LS 字节
XORW    *AC3，AC2           // 表值与 CRC 异或
INCW    AC1                 // 指向下一段信息字节
NEXT                        // 循环结束

SWAP    AC2                 // 返回前交换 CRC 字节
WDR                         // 复位看门狗
RET                         // 返回
```

中断程序

// INT 120
// 当静止线定时器到时，该中断程序执行。起动接收请求首字符的中断程序 INT122，因为下一字符应当是我们要接收信息的开始。

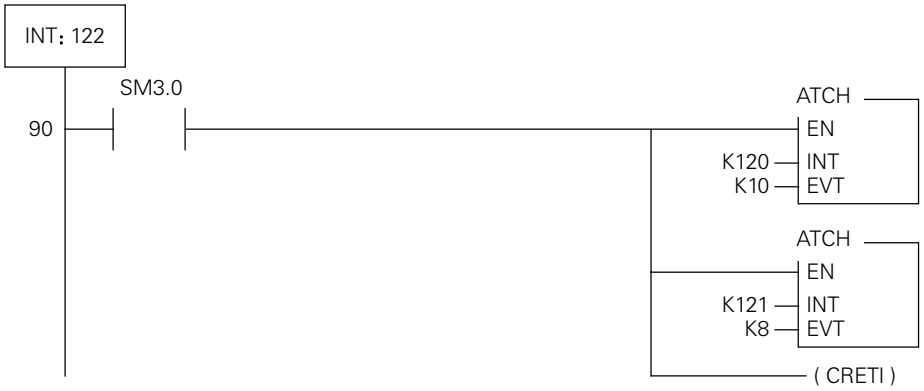



```
INT     120              // 中断程序 120
LD      SM0.0            // SM0.0 总为 1
DTCH    10               // 关断静止线定时器
ATCH    122，8           // 起动接收请求首字符的中断程序 INT122
MOVW    0，VW3300        // 把接收计数器清零
MOVD    &VB3302，VD3570  // 设定接收缓冲区指针
RETI                     // 返回
```

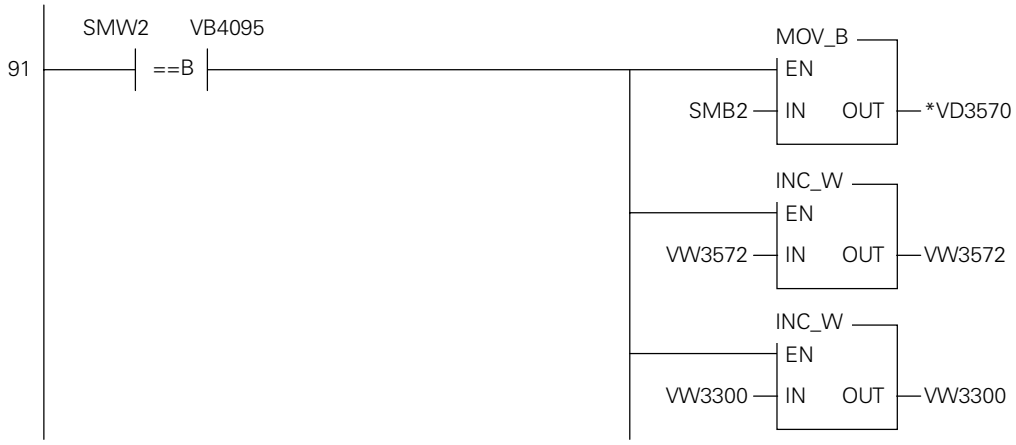
// INT 121
// 该中断程序在寻找静止线时处理串行口中断，如果在寻找静止线期间收到字符，则重新起动静止线定时器。

```
INT   121                    // 中断程序 121
LD        SM0.0              // SM0.0 总为 1
ATCH 120，10                 // 重新使能静止线定时器
RETI                         // 返回
```

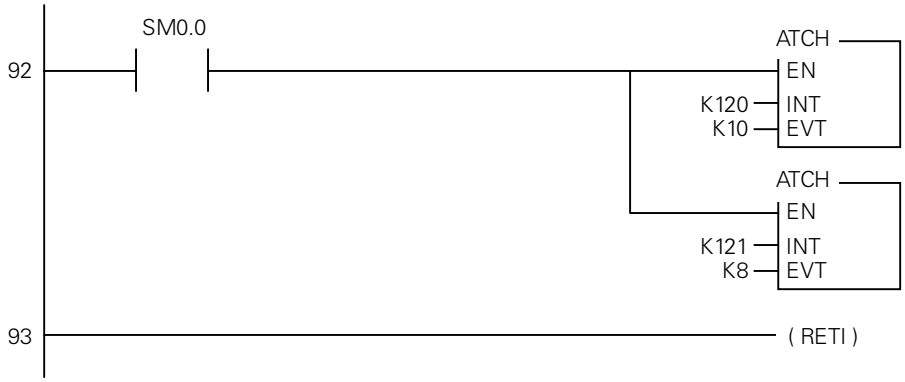
```
// INT   122
// 在静止线定时器到时后，将接收信息的第一个字节。信息的第一个字节是地址。检查是否是请求的这个地址，若是，则起动中断程序 INT123 接收整个信息。如果不是这个地址的信息，则返回并寻找静止线时间。
// 注意：
// 在该执行中当加上额外代码时不支持广播地址。如果需要这个特性，该中断程序必须修改成能适应所有功能的处理器，因为广播地址不被所有功能支持。
```




```
INT   122                    // 中断程序 122
LD        SM3.0              // 若（奇偶校验错），则
ATCH 120，10                 // 开始寻找静止线，此期间
ATCH 121，8                  // 若收到一个字符，那就执行中断程序 LNT121
CRETI                        // 返回
```

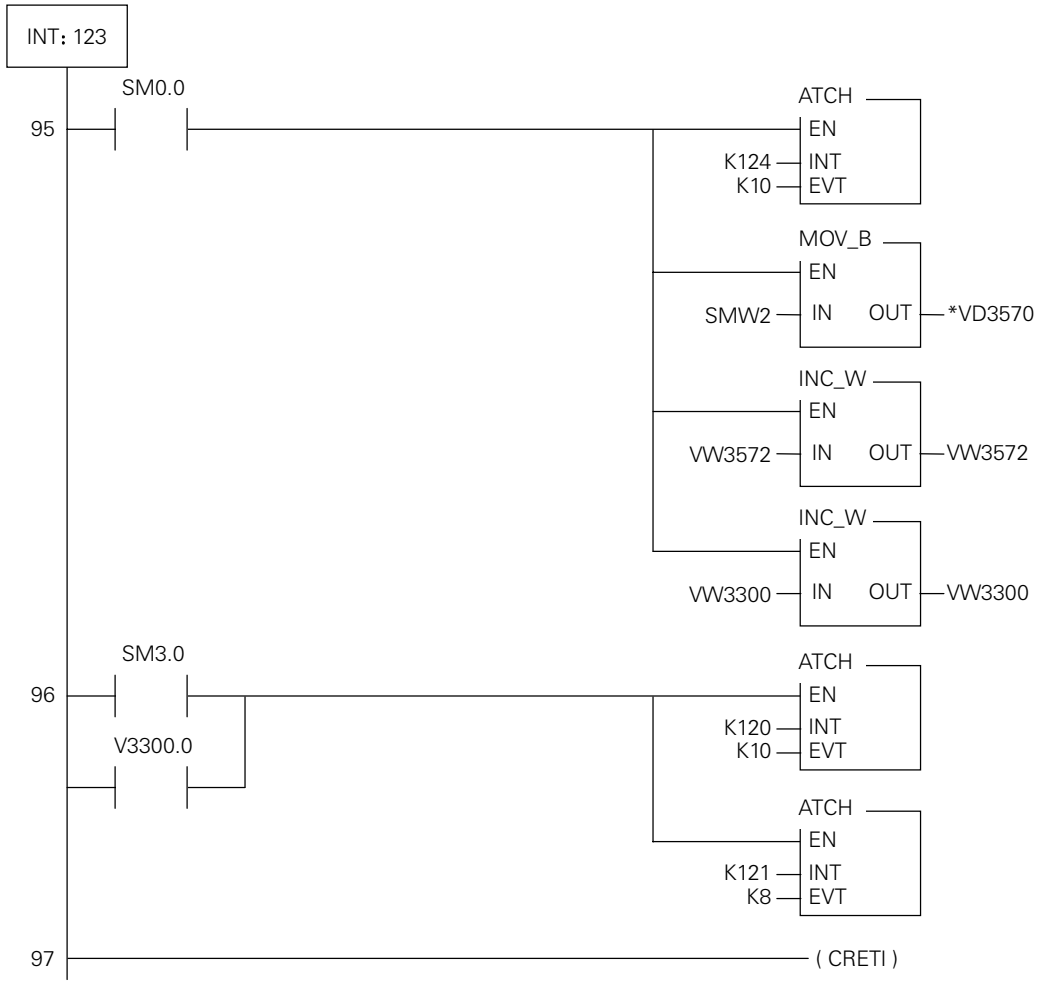



```
                                    // 否则（奇偶校验正确）
LDB=    SMB2，VB4095                // 若（地址=我的地址），则
MOVB    SMB2，*VD3570               // 把地址写入缓冲区
INCW    VW3572                      // 缓冲区指针加 1
INCW    VW3300                      // 接收字节计数器加 1
ATCH    123，8                      // 接收其余信息
ATCH    124，10                     // 若时间到，则结束接收
CRETI                               // 返回
```



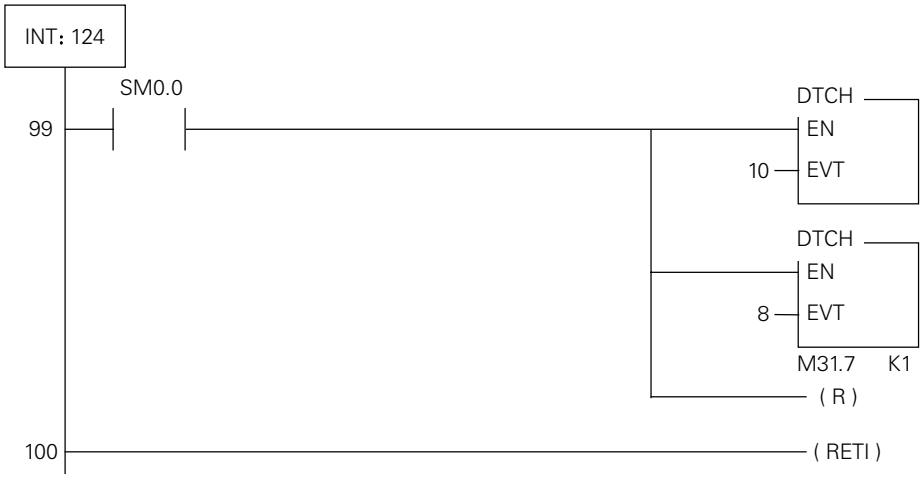

```
LD      SM0.0                       // 若（地址！=我的地址），则
ATCH120，10                         // 重新使能静止线定时器，此期间
ATCH121，8                          // 若收到一个字符，那就执行中断程序 INT121
RETI                                // 返回
```

```
// INT  123
// 这是主要字符中断服务程序。接收字符、校验奇偶错，接着存入通信缓冲区。每接收 1 个字符将字节计数增加 1。
   若收到的字节数大于 256，则接收失败。
```



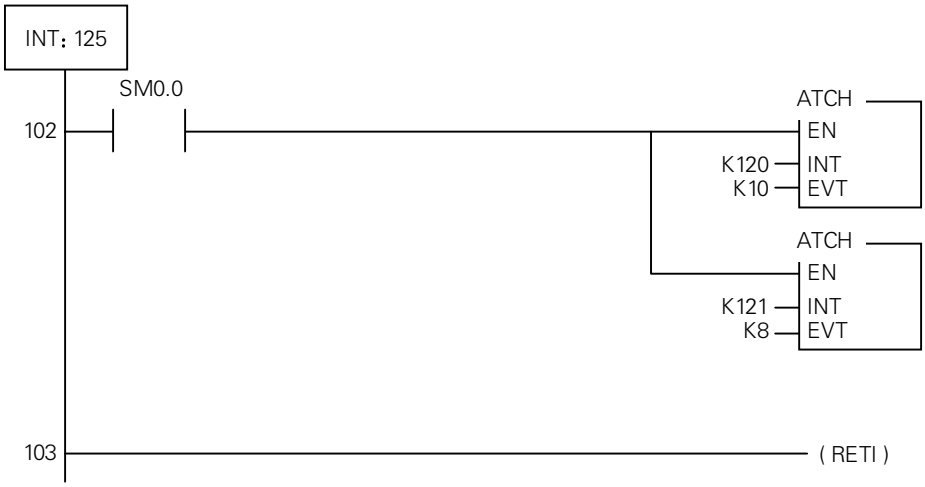

```
INT     123              // 中断程序 123
LD      SM.0.0           // SM0.0 总为 1
ATCH    124，10          // 下一个静止线定时器时间到信息完成
MOVB    SMB2，*VD3570    // 在缓冲区里存贮收到的信息
INCW    VW3572           // 缓冲区指针加 1
INCW    VW3300           // 接收字节计数器加 1

LD      SM3.0            // 若（奇偶校验错）或
O       V3300.0          // （缓冲区溢出）
ATCH    120，10          // 则开始寻找静止线，此期间
ATCH    121，8           // 若收到一个字符，那就执行中断程序 INT121
RETI                     // 返回
```

```
// INT 124
// 当信息已收到，且我们已找到另一个静止线而意味着请求完成时，则这个中断程序执行。它关断静止线定时器，
   关断端口中断，且设置一个处理信息的反馈标志。
```




```
INT   124                  // 中断程序 124
LD    SM0.0                // SM0.0 总为 1
DTCH  10                   // 关断静止线定时器
DTCH  8                    // 关断端口中断
S     M31.7，1             // 设置处理信息的反馈标志（M31.7=1）
RETI                       // 返回
```

```
// INT   125
// 当响应已完成，发送回主机时该中断程序执行。复位中断系统，重新寻找静止线，开始新的请求。
```




```
INT   125                  // 中断程序 125
LD    SM0.0                // SM0.0 总为 1
ATCH  120，10              // 开始寻找静止线，此期间
ATCH  121，8               // 若收到一个字符，那就执行中断程序 INT121
RETI                       // 返回
```

后　　记

SIMATIC S7-200 应用示例的目的是为了从编程技术的角度，就如何用这种控制器解决一些问题，仅供 S7-200 的用户在使用时参考。这些应用示例指令决无覆盖该设备的所有细节、差异或各种可能的意图。您可以无偿使用 S7-200 应用示例。

西门子（SIEMENS）保留下述权利：任何时候，西门子可能会改动这里的技术参数。在做出任何改进时，西门子不必通知用户，也不承担任何责任。

该应用示例不是为了免除用户应承担的责任，即用户在应用、安装、操用以及维护购买的设备时，必须要具备的全面技能。如果在这份出版物中包含的内容与图纸或附件中的内容不一致，则以图纸或附件为准。

由于使用该应用示例而造成的设备损坏或人员伤亡，西门子不承担任何责任，包括法律责任。

版权所有。任何形式的复制、发行、包括摘录，都必须经西门子（SIEMENS）授权认可。
欢迎您使用 S7-200 应用示例。