第 5 章

边 沿 触 发

边沿触发指上升沿触发和下降沿触发，捕捉的是 BOOL 型变量从高电平到低电平或从低电平到高电平变化的瞬间。由于 PLC 是采用循环扫描的工作方式，扫描周期一般是毫秒级，因此当操作人员按下按钮，或者接近开关、光电开关等传感器检测到工件，虽然这些操作时间很短，但 PLC 已经执行了成千上万次指令。有时只需要指令执行一次，例如使用光电开关进行产品计数，光电开关检测到一件产品，只需要计一次数即可。由于各种客观因素的影响，光电开关检测到一件产品，PLC 的计数指令可能已经执行了很多次，计数结果显然是不准确的。如果只捕捉到信号的变化，即光电开关捕捉的信号从无到有这一瞬间，才计一次数，也就是执行一次指令，则就能实现正确的计数，这就是边沿触发，捕捉的是电平的变化。

注意：边沿触发捕捉的是 BOOL 型变量的变化，其他类型的变量不存在边沿触发的概念。

5.1 基本概念

5.1.1 上升沿

上升沿(R_TRIG)指 BOOL 型变量从低电平变化为高电平的那一瞬间，梯形图中的实现如图 5-1 所示。

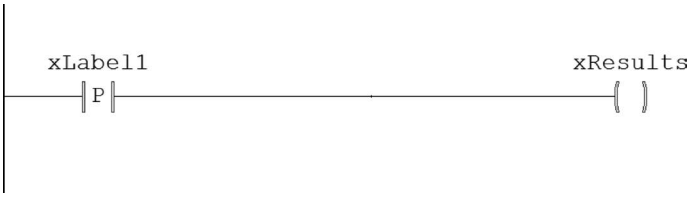

图 5-1 上升沿梯形图

图 5-1 中的梯形图与图 4-1 中的最大不同在于：只在变量 xLabel1 从 FALSE 变为 TRUE 的瞬间，变量 xResults 才为 TRUE，并持续一个扫描周期。假设变量 xLabel1 为 PLC 的输入点，接入按钮的常开点；变量 xResults 为 PLC 的输出点，接指示灯，只有在按下

按钮的瞬间，指示灯才会亮。图 4-1 中的梯形图，只要按下按钮，指示灯就一直亮。所以，上升沿的实质是捕捉 BOOL 型变量从低电平到高电平的变化，或者从无到有的变化，取的是“变化”；而常开常闭点，取的是“结果”。

IEC 61131-3 标准制定了专门的功能块 R_TRIG(R 是英文 RISE 的首字母，意为上升，TRIG 意为触发)来实现上升沿。如果调用功能块来实现图 5-1 中的上升沿触发，则如图 5-2 所示。

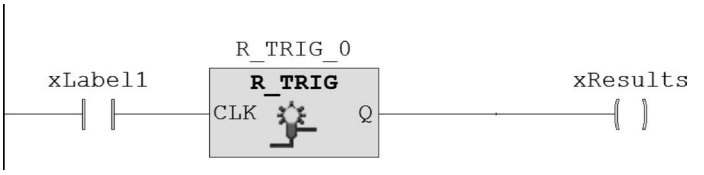

图 5-2　功能块实现上升沿

图 5-2 中的梯形图与图 5-1 中的梯形图是完全等价的，它采用功能块实现了上升沿触发。当功能块 R_TRIG 检测到输入引脚 CLK 上的变量 xLabel1 发生从低电平到高电平的变化时，输出引脚 Q 变为高电平，并持续一个扫描周期。因此，变量 xResults 也变为高电平，并持续一个扫描周期。功能块 R_TRIG 的作用就是捕捉它的输入引脚 CLK 的上升沿。

上升沿的实质是把变量当前扫描周期的值与上一个扫描周期的值进行比较，如果上一个扫描周期的值为 FALSE，而本次扫描周期的值为 TRUE，则捕捉到一次上升沿变化。因此，要想捕捉到上升沿，必须存储上一个扫描周期的值。回顾 2.6.3 节，功能块和函数的区别就是功能块有存储空间，因此实现上升沿必须调用功能块。使用功能块必须实例化，也就是要定义功能块型变量。图 5-2 中的 R_TRIG_0 就是定义的功能块型变量。功能块型变量在 CODESYS 中与 BOOL 型、INT 型、结构体等一样，是一种数据类型。而在西门子博途中，称为背景数据块。实现上升沿的代码如下：

```
VAR
    R_TRIG_0 : R_TRIG;                    //上升沿功能块实例化
END_VAR
R_TRIG_0(CLK := xLabel1,Q => xResults);
```

从这段代码不难看出，功能块调用的格式如下：

实例名(VAR_IN := <变量或表达式>,VAR_OUT => <变量或表达式>);

功能块调用就是分别使用运算符“:=”和运算符“=>”来获取功能块的输入和输出，它们之间用“,”隔开，调用结束，以“;”结束。所以调用功能块的实质是一系列赋值操作。调用功能块的目的就是对于给定的输入值获取其输出值。如果要在其他语句中使用功能块的输入和输出，调用格式为“实例名.输入/输出”。因此，图 5-1 中的梯形图也可以用如下代码表示。

```
VAR
  R_TRIG_0 : R_TRIG;
END_VAR
R_TRIG_0(CLK := xLabel1);
```

```
IF R_TRIG_0.Q THEN
    xResults := TRUE;                          //检测到上升沿时,输出一个扫描周期
ELSE
    xResults := FALSE;
END_IF
```

这段代码使用 IF…ELSE…END_IF 语句实现图 5-1 中的梯形图程序。从代码可以看出,对于功能块的调用,ST 语言还是很灵活的,既可以在功能块的输入引脚和输出引脚上直接填写变量,也可以使用“.”引用输入引脚和输出引脚的值,再通过赋值语句传递相应的变量。这段代码中,IF…ELSE…END_IF 语句的判断条件 R_TRIG_0.Q,就是使用实例名 R_TRIG_0 来调用功能块的输出引脚 Q。对于不需要的引脚,在调用的时候也可以省略。ST 语言在实现上升沿的时候要比梯形图略显烦琐,如果不想手动输入,SoMachine 还提供了输入助手功能。

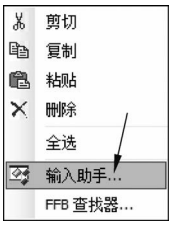

图 5-3　SoMachine 中调用输入助手

首先,在需要输入代码的空白处右击,弹出的快捷菜单如图 5-3 所示。

然后,选择“输入助手”命令,弹出“输入助手”对话框,如图 5-4 所示。

图 5-4　“输入助手”对话框

选择“功能块”，右侧会出现调用功能块选项，单击 Standard 前面的“＋”，展开 Standard 列表，在展开的列表中选择 Trigger 前面的“＋”，展开 Trigger 列表，如图 5-5 所示。

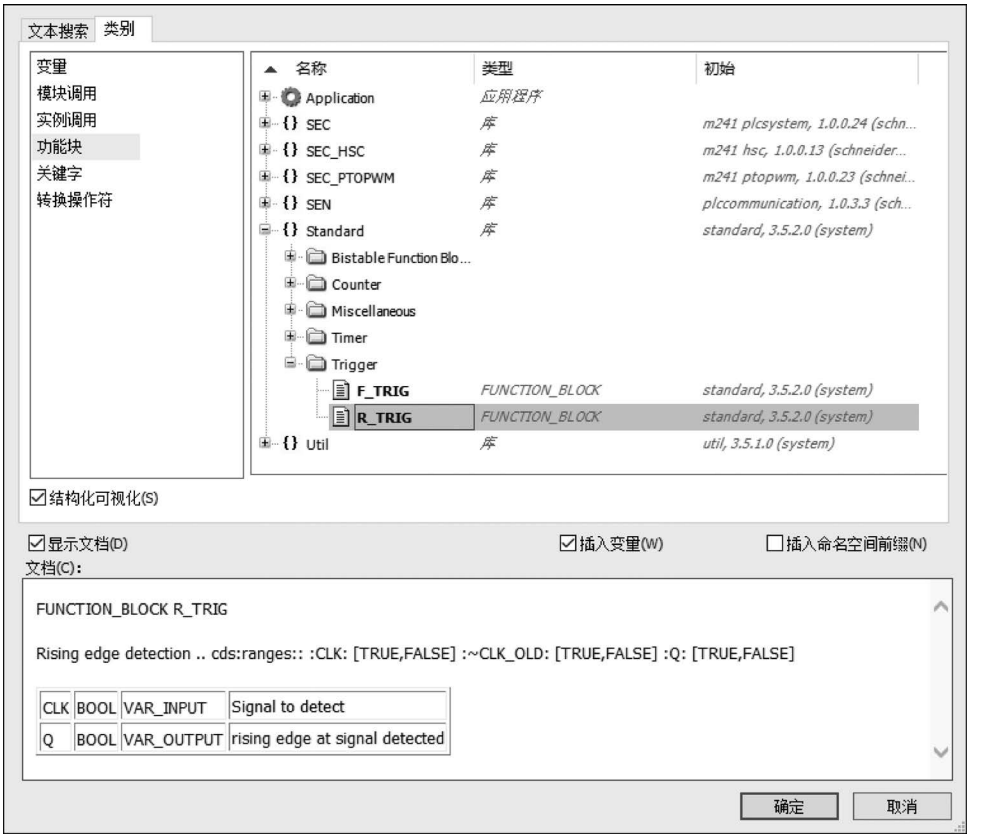

图 5-5　SoMachine 中输入助手对话框中的边沿触发

选择“R_TGIG”，弹出“自动声明”对话框，即调用 R_TRIG 的对话框，如图 5-6 所示。自动声明是指为调用的 R_TRIG 功能块输入/输出引脚分配变量。

图 5-6　SoMachine 中定义 R_TRIG 对话框

在图 5-6 中的“名称(N):”文本框中填写实例名,从“范围(S):”“标志(F):”“对象(O):”“初始化(J):”“地址(A):”等描述不难看出,CODESYS 中把功能块的实例名看作一种数据类型。因此,在 CODESYS 中,功能块的实例名简称功能块型变量。对于功能块型变量的命名,可以按照 3.4.4 节介绍的变量命名原则来命名,也可以使用简单的方式命名为“R1”。单击“确定”按钮,系统就自动生成功能块 R_TRIG 的调用格式,如图 5-7 所示。

```
75
76  R1(CLK:= , Q=> );
77
```

图 5-7　输入助手自动建立的代码

在 SoMachine 中,所有的功能块和函数调用都可以采用这种方式辅助输入。

5.1.2　下降沿

下降沿(F_TRIG)是指 BOOL 型变量从高电平变化为低电平的那一瞬间,使用功能块 F_TRIG(F 是英文单词 FALL 的首字母,意为下降)实现,下降沿在梯形图中的实现方法如图 5-8 所示。

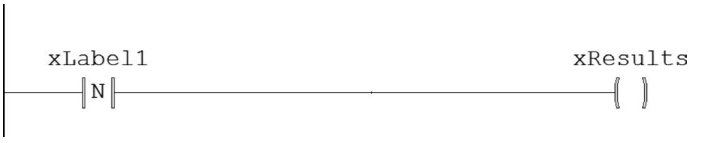

图 5-8　下降沿梯形图

用 ST 语言实现下降沿,代码如下:

```
VAR
    F_TRIG_0 : F_TRIG;                  //下降沿功能块实例化
END_VAR
F_TRIG_0();                             //调用功能块
F_TRIG_0.CLK := xLabel1;
xResults     := F_TRIG_0.Q ;            //检测到下降沿时,输出一个扫描周期
```

调用下降沿功能块的代码中采用了另一种方式,不是在调用功能块时将变量赋值给输入/输出,而是直接用赋值表达式实现。可见,功能块的调用是非常灵活的。采用这种方法时,应尽量把赋值语句和调用功能块的语句放在一起,不然调试时非常麻烦。

通过对边沿触发功能块的学习,可以发现,ST 语言中调用功能块是非常灵活的。调用功能块在 ST 语言中应用非常广泛,类似梯形图中的各种指令。希望读者能够在学习边沿触发的过程中掌握调用功能块的方法,后续章节还会继续介绍如何调用功能块。

5.1.3　西门子博途中的边沿触发

西门子博途也提供了 R_TRIG 和 F_TRIG 功能块实现边沿触发。R_TRIG 和 F_TRIG 功能块是 IEC 61131-3 中制定的标准功能块,凡是符合 IEC 61131-3 标准的 PLC,都可以使用这两个功能块来实现边沿触发。由于西门子博途中功能块的调用,涉及背景数据块,与 CODESYS 平台的 PLC 略有不同。下面介绍西门子博途中边沿触发的实现方法。

1. 如何调用

在西门子博途(以下简称博途)中调用各种功能块非常方便,它提供了指令窗口,可以把指令直接拖曳到编辑区,如图 5-9 所示。

直接把图 5-9 中的 R_TRIG 拖曳到代码编辑区,会自动弹出“调用选项”对话框,如图 5-10 所示。

图 5-9　博途中的指令窗口

图 5-10　博途中的分配背景数据块

选择“单个实例”,名称为“R_TRIG_DB”,这在博途中称为分配背景数据块,实质是分配一块存储空间。“R_TRIG_DB”就是为这块存储空间取的名字,类似于 CODESYS 中的功能块型变量,也就是实例名。可根据需求更改名称,只需要符合博途的命名规范即可,具体可参考博途编程手册。此处更改名称为“R1”,单击“确定”按钮,在代码编辑区会自动生成上升沿代码格式,如图 5-11 所示。

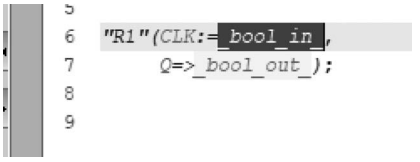

图 5-11　博途自动生成的上升沿代码格式

从图 5-11 可以看出,系统已经自动生成调用格式,只需要添加变量即可,并且系统自动为 R1 添加了变量标识符。博途中的背景数据块的实质也是变量。默认单个实例是全局变量；填入变量后,代码如下：

```
"R1"(CLK := #xLabel1);
#xResults := "R1".Q;
```

当然,也可以使用 IF…ELSE…END_IF 语句,实现代码如下：

```
"R1"(CLK := #xLabel1);
IF  "R1".Q THEN
  #xResults := TRUE;
```

```
ELSE
  #xResults := FALSE;
END_IF
```

2. 如何避免产生大量背景数据块

边沿触发是 PLC 编程中经常需要使用的，一个项目中可能需要很多边沿触发。在博途中，每调用一次边沿触发就会建立一个背景数据块，因此会产生大量的背景数据块，使得程序结构混乱，影响程序的可读性，并且严重浪费 PLC 的系统资源。那么如何避免这个问题呢？博途提供了多重背景的解决方案，可以利用 FB 本身的背景数据块，保存边沿检测需要的数据。

下面介绍如何使用多重背景。在图 5-10 的“调用选项”对话框中选择“多重实例”，需要注意的是，博途默认是选择“单个实例”，如图 5-12 所示。

图 5-12　定义多重实例

图 5-12 中，“接口参数中的名称”栏中的内容就是定义的实例名，可以使用默认值，也可以根据需要更改。这里更改为“R1”，单击“确定”按钮，系统就在当前 FB 的 STATIC 中即静态变量中，自动建立一个名称为“R1”，数据类型为 R_TRIG 的变量，如图 5-13 所示。

类似地，需要多少个变量就拖曳多少个上升沿进来。注意一定要选择“多重实例”，如图 5-14 所示。

从图 5-14 可以看出，系统同样为 R1 自动添加了变量标识符，由于 R1 是在 FB 内建立的，所以是局部变量。这样充分利用了 FB 的存储空间存储上升沿功能块所需要的数据，减少了背景数据块。在博途中，所有的功能块调用都可以按照这种方式操作，以减少背景数据块的数量。但是此种方法只适用在 FB 中调用边沿触发的情况，在 FC 中调用边沿触发是无法使用多重实例的。如果需要在 FC 中调用，可以使用参数实例来解决，例如在博途中新建 FC 块 FC1，然后调用 R_TRIG 功能块，出现“调用选项”对话框时，选择“参数实例”，如图 5-15 所示。

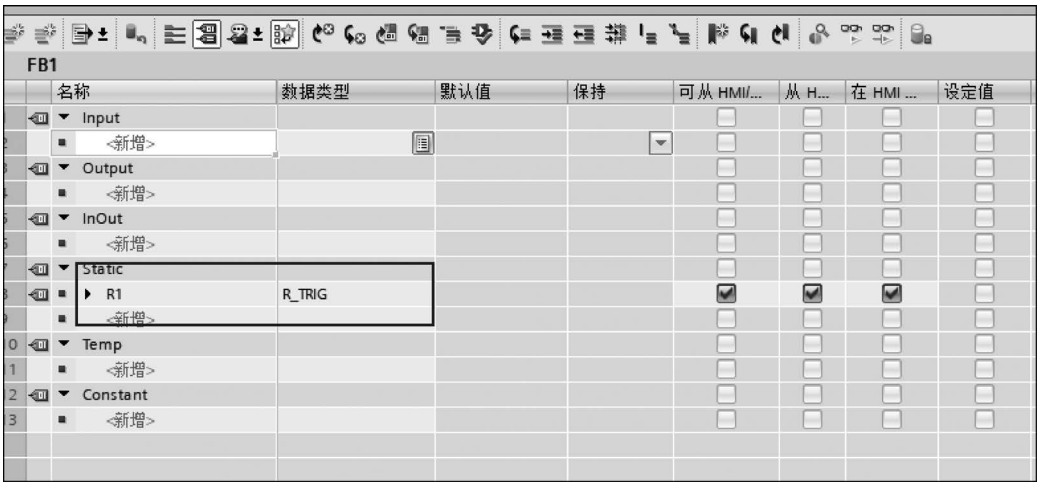

图 5-13　定义的多重背景

对比图 5-12 和图 5-15 可以看出，在 FC 中调用功能块时是没有“多重实例”选项的。接口参数中的名称更改为“R1”，然后单击“确定”按钮。系统就自动在当前 FC 的 InOut 即输入/输出变量中建立一个名称为“R1”，数据类型为 R_TRIG 的变量，如图 5-16 所示。

图 5-14　FB 中调用多个上升沿触发功能块

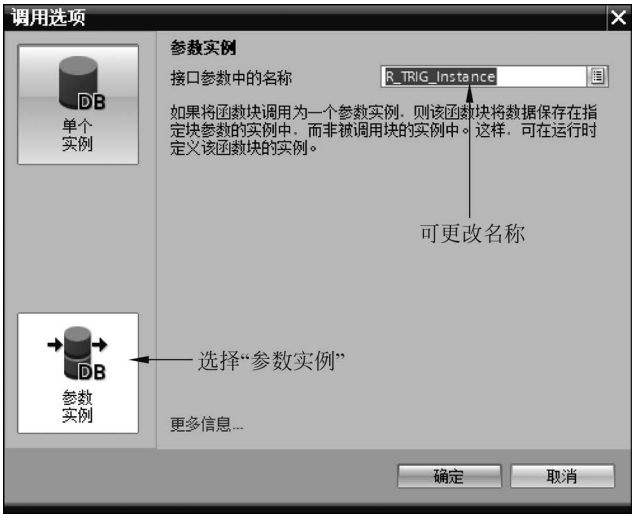

图 5-15　FC 中选择参数实例

与多重背景一样，需要多少个变量就建立多少个实例，注意一定要选择“参数实例”，如图 5-17 所示。

由于 FC 并没有存储空间，所以无法实现边沿触发，参数实例的目的是利用其他 FB 的存储空间。新建 FB 块，命名为“FB1”，然后在 FB1 中调用 FC1，如图 5-18 所示。

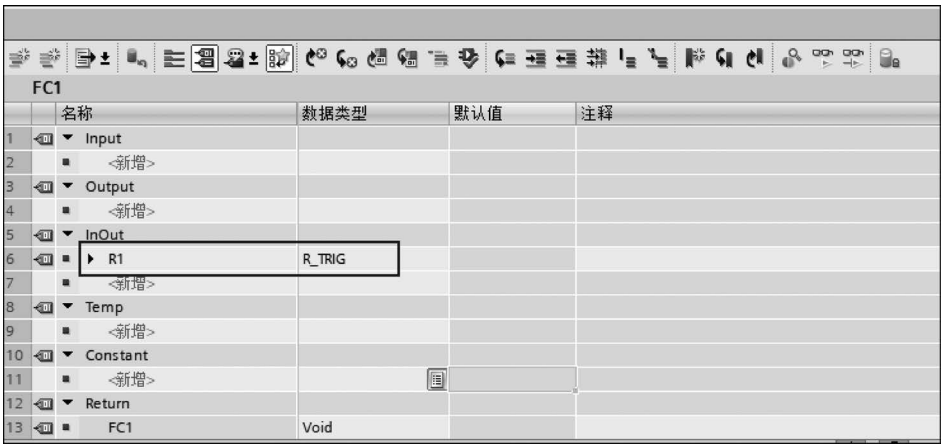

图 5-16　FC 中建立的参数实例

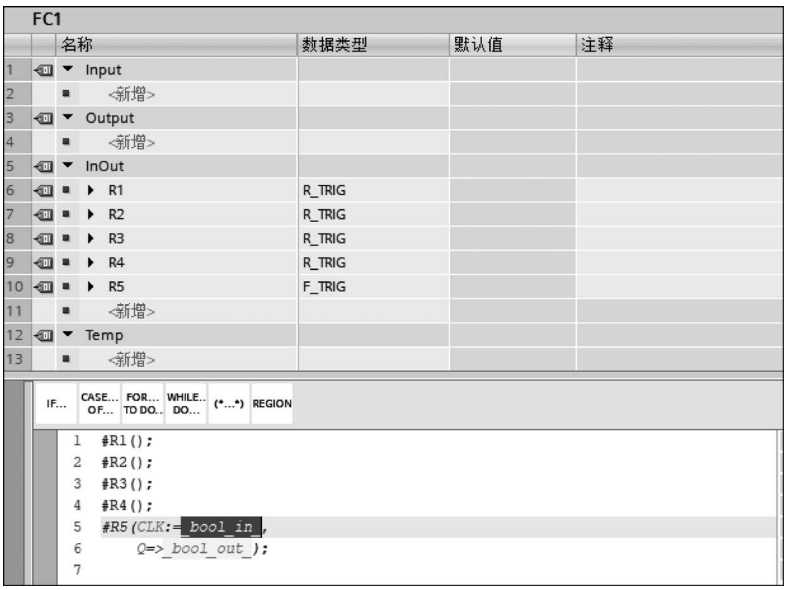

图 5-17　FC 中建立多个参数实例

从图 5-18 可以看出，只需要在调用 FC 的 FB 中建立 R_TRIG 类型的静态变量即可，这就是参数实例的意义，如图 5-19 所示。

可以看出，参数实例的目的是利用其他 FB 的存储空间，为没有存储空间的 FC 服务。参数实例可以看作是一个接口，用来建立 FB 和 FC 之间的联系。

注意：博途中的 FB 就是功能块，FC 就是函数。本例中调用的 FB 和 FC 就是用户自定义功能块和自定义函数。

图 5-18　在 FB 中调用 FC

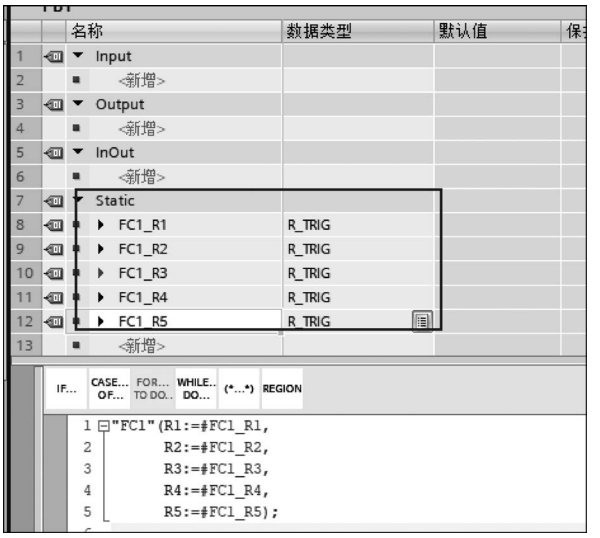

图 5-19　为调用的 FC 分配变量

5.2　边沿触发与逻辑运算的综合应用

边沿触发在 PLC 编程中应用广泛，下面结合前面章节介绍的逻辑运算语句和 IF 语句，并结合具体的例子讲解它们的具体应用。

5.2.1　启动保持停止

启动保持停止梯形图是 PLC 梯形图的基础，是梯形图原理最好的体现，在 PLC 编程中应用广泛。由于现场存在各种不可预知的干扰因素，为了最大限度地减少误操作，或者出于程序编写的便利性，可以使用边沿触发。例如，启动信号来自其他功能块的输出，功能块的输出一直是高电平信号，因此启动信号需要使用上升沿触发，如图 5-20 所示。

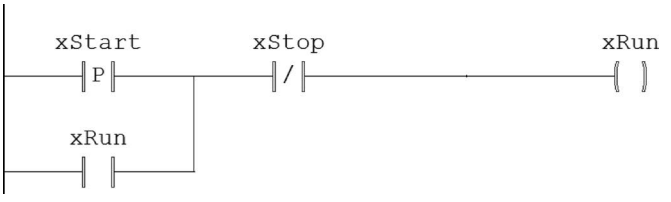

图 5-20　上升沿触发的启动保持停止梯形图

图 5-20 中的梯形图是采用上升沿触发的启动保持停止梯形图。需要再次强调，为了设备安全，停止信号应该接到按钮的常闭点上，因此程序中应当使用常开触点。为了演示 NOT 逻辑，图 5-20 的梯形图使用了常闭触点，即停止信号接到停止按钮的常开点上。

该梯形图使用 ST 语言实现的代码如下：

```
VAR
    xStart   : BOOL;
    xStop    : BOOL;
    xRun     : BOOL;
    R_TRIG_0 : R_TRIG;
END_VAR
R_TRIG_0(CLK := xStart);
xRun := (R_TRIG_0.Q  OR  xRun ) AND  (NOT xStop);
```

与 4.5.1 节中的代码相比，本例用 R_TRIG_0.Q 代替了启动信号，即变量 xStart，而 R_TRIG_0.Q 正是变量 xStart 的上升沿。

当然，也可以使用 IF…ELSE…END_IF 语句实现，代码如下：

```
R_TRIG_0(CLK := xStart);
IF ( (R_TRIG_0.Q  OR  xRun ) AND  (NOT xStop) ) THEN
    xRun := TRUE;                          //启动
ELSE
    xRun := FALSE;                         //停止
END_IF
```

5.2.2　单按钮启停

4.5.3 节用逻辑关系实现了单按钮启停功能。从程序代码可以看出，利用逻辑关系实现单按钮启停功能，十分烦琐。单按钮启停功能也可以使用 XOR 运算实现。

回顾前面介绍的异或运算，是比较两个逻辑变量的值，如果值相同，运算结果为 FALSE；如果值不同，则运算结果为 TRUE。因此，比较输入和输出的值就可以实现单按钮输出功能。代码如下：

```
VAR
    xIN   : BOOL;                  //输入按钮
    xOUT  : BOOL;                  //输出继电器
    R1    : R_TRIG;
END_VAR
R1(CLK := xIN);
xOUT := R1.Q XOR xOUT;
```

以上是使用逻辑表达式实现的单按钮启停，使用 IF…ELSE…END_IF 语句实现的代码如下：

```
R1(CLK := xIN);
IF (R1.Q XOR xOUT ) THEN
    xOUT := TRUE;
ELSE
    xOUT := FALSE;
END_IF
```

变量 xIN 分配的地址接入按钮，变量 xOUT 分配的地址接入继电器，按下按钮，变量 xIN 的值为 TRUE，变量 xOUT 的值为 FALSE。根据异或的运算规则，输出变量 xOUT

的值为 TRUE，继电器吸合；再次按下按钮，变量 xIN 的值为 TRUE，变量 xOUT 的值此时为 TRUE，根据异或的运算规则，此时输出变量 xOUT 的值为 FALSE，继电器释放。如此反复，就实现了单按钮启停功能。

此程序中取的是变量 xIN 的上升沿，读者可以思考并自行模拟如果不采用上升沿会发生什么结果。

如果不采用上升沿，按下按钮时，变量 xOUT 变为 TRUE，由于按下按钮的时间，远远大于 PLC 的扫描周期，会进行多次 XOR 运算。第一次运算时，变量 xIN 的值为 TRUE，变量 xOUT 的值为 FALSE，经过 XOR 运算后，变量 xOUT 的值变为 TRUE；然后会进行第二次运算，此时变量 xIN 的值为 TRUE，而变量 xOUT 的值也为 TRUE，运算结果为 FALSE；然后进行第三次运算，此时变量 xIN 的值为 TRUE，变量 xOUT 的值为 FALSE，运算结果为 FALSE，因此输出变量 xOUT 的值会在 FALSE 和 TRUE 之间不停地切换。如果取变量 xIN 的上升沿，只在第一次运算的时候 R1.Q 的值为 TRUE，以后每次运算中，R1.Q 为 FALSE，XOR 运算的结果一直为 TRUE，因为按下按钮后只有一次上升沿触发。

5.2.3 逻辑运算实现边沿触发

边沿触发的原理是比较变量在相邻两个扫描周期的值。如果上一个扫描周期的值为 TRUE，而本次扫描周期的值为 FALSE，那么就捕捉到此变量的一次下降沿；如果上一个扫描周期的值为 FALSE，而本次扫描周期的值为 TRUE，则捕捉到此变量的一次上升沿，这也是功能块 F_TRIG 和 R_TRIG 的工作原理。所以，要想捕捉到边沿信号，必须存储变量在每个扫描周期的值，至少要存储相邻两个扫描周期的值，这也是为什么边沿触发功能块需要实例名，并分配背景数据块的原因。

理解了边沿触发的原理，完全可以通过编写程序来实现边沿触发，程序代码如下：

```
VAR
    xLabel_Last : BOOL;                          //变量 xLabel 在上一个扫描周期的值
    xLabel      : BOOL;                          //用于捕捉边沿信号的变量
    xLabel_R    : BOOL;                          //上升沿信号
    xLabel_F    : BOOL;                          //下降沿信号
END_VAR
IF (xLabel_Last = FALSE) AND (xLabel) THEN       //和上一个扫描周期的值比较
    xLabel_R := TRUE;                            //上升沿
ELSE
    xLabel_R := FALSE;
END_IF;
IF (xLabel_Last) AND (xLabel = FALSE) THEN       //和上一个扫描周期的值比较
    xLabel_F := TRUE;                            //下降沿
ELSE
    xLabel_F := FALSE;
END_IF
xLabel_Last := xLabel;                           //保存当前扫描周期的值
```

第一个 IF…ELSE…END_IF 语句中，变量 xLabel 在上一个扫描周期的值为 FALSE，而在本次扫描周期的值为 TRUE，捕捉到它的上升沿，变量 xLabel_R 在一个扫描周期内值

为 TRUE。第二个 IF…ELSE…END_IF 语句中，变量 xLabel 在上一个扫描周期的值为 TRUE，而在本次扫描周期的值为 FALSE，捕捉到它的下降沿，变量 xLabel_F 在一个扫描周期内值为 TRUE。

注意：利用逻辑语句实现边沿触发依赖 PLC 的循环扫描原理。在程序的最后，执行语句“xLabel_Last:= xLabel;”保存当前扫描周期内变量 xLabel 的值。在下一个扫描周期，变量 Label_Last 的值，就是变量 xLabel 在上一个扫描周期的值。如果把语句“xLabel_Last:= xLabel;”放到第一个 IF…ELSE…END_IF 语句的前面，则无论如何也无法实现边沿触发功能。

5.2.2 节的单按钮启停程序也可以使用下面的代码实现。

```
xOUT   := (xIN AND NOT xLabel) XOR xOUT;
xLabel := xIN;
```

以上代码利用上升沿的逻辑运算式代替了上升沿功能块 R_TRIG。

5.3　注意事项

边沿触发在 PLC 中的应用非常广泛。前文提到过，IEC 61131-3 标准是推荐标准，并不是强制标准，因此各家 PLC 在符合标准的前提下，都有自己的特色。在三菱 GX Works3 中，既有标准的边沿触发功能块 R_TRIG 和 F_TRIG，其用法与 CODESYS 平台 PLC 一样；也有三菱 PLC 独有的边沿触发功能块——PLS 功能块和 PLF 功能块，如图 5-21 所示。

图 5-21　三菱 GX Works3 中的 PLS 功能块和 PLF 功能块

图 5-21 中是 PLS 和 PLF 在梯形图中的表示形式，PLS 是上升沿触发，PLF 是下降沿触发。从图中可以看出，它们都没有分配实例名，所以是函数。熟悉三菱 PLC 的读者，对这两个功能块应该不陌生。当输入引脚 EN 捕捉到上升沿，PLS 功能块的输出引脚 d 会输出一个扫描周期的高电平信号；当输入引脚 EN 捕捉到下降沿，PLF 功能块的输出引脚 d 会输出一个扫描周期的高电平信号。可见，它们的用法与 R_TRIG 和 F_TRIG 类似。在三菱 GX Works3 中，使用这两个功能块实现边沿触发的 ST 语言代码如图 5-22 所示。

```
PLS( xLabel , xResults1);
PLF( xLabel , xResults2);
```

图 5-22　GX Works3 中边沿触发的 ST 语言代码实现

从图 5-22 可以看出，在三菱 GX Works3 中，这两个功能块用函数调用的形式实现。在变量 xLabel 的上升沿，变量 xResults1 的值变为 TRUE，并持续一个扫描周期；在变量

xLabel 的下降沿，变量 xResults2 的值变为 TRUE，并持续一个扫描周期。

PLS 和 PLF 两个功能块是三菱 PLC 独有的，主要目的是兼容自家的产品。为了和自己既有的 PLC 兼容，许多 PLC 都有类似的指令。它们在 ST 语言中的使用方法，无非是调用函数和调用功能块，但是这些 PLC 特有的指令无法进行跨平台移植。所以笔者建议，尽量使用标准的 R_TRIG 和 F_TRIG 功能块来实现边沿触发。