北岛李工的工控博客

https://www.founderchip.com/

之前我曾写过一个斜坡发生器函数（参考文章：西门子SCL编程实例——斜坡发生器（基于定时器）），它能控制输出值经过一个平滑的上升或下降的变化过程而达到目标值。这个函数只实现一次变化，不能连续变化。前段时间有小伙伴询问如何让斜坡发生器实现连续的变化输出？比如电机转速从0 r/min平滑变化到100 r/min，然后再到 500 r/min等等。今天这篇文章，我给出我的解决方案。

在原来函数的基础上，我编写了更新版的斜坡发生器，命名为：FB5028_RampGeneratorEx。

为了实现斜坡的连续变化，我定义了一个可变数组（见下文的输入/输出变量），你可以在这个数组中随意添加斜坡节点信息。这些信息包括：启动信号（何时开始输出）、目标值和斜坡时间，我将其定义为用户自定义数据类型：typeRampData，如下图所示：

typeRampData

	名称	数据类型	默认值	设定值	注释
1	start	Bool	false		开始启动斜坡发生器，上升沿有效
2	targetValue	Real	0.0		目标值
3	rampTime	DInt	10		斜坡时间，单位：秒

在博途开发环境下新建函数块：FB5028_RampGeneratorEx，其输入、输出及输入/输出变量的定义如下图所示：

FB5028_RampGeneratorEx

	名称	数据类型	默认值	保持	设定值	注释
1	Input					
2	enable	Bool	false	非保持		使能
3	initValue	Real	0.0	非保持		初始值
4	Output					
5	value	Real	0.0	非保持		输出值
6	stepActive	Bool	false	非保持		步激活
7	cycleCounter	DInt	0	非保持		循环计数
8	singleStepDone	Bool	false	非保持		单步完成
9	complete	Bool	false	非保持		全部完成
10	InOut					
11	arrayRampData	Array[*] of "typeRampData"				不定长数组，存放斜坡信息
12	arrayRampData[*]	"typeRampData"				不定长数组，存放斜坡信息
13	Static					
14	statValueDiffer	Real	0.0	非保持		差值
15	statValueDifferUnit	Real	0.0	非保持		单位差值

其中：

输入变量

名称	数据类型	描述
enable	布尔值	使能（TRUE=使能，FALSE=复位）
initValue	实数	初始值

输出变量

名称	数据类型	描述
value	实数	输出值
stepActive	布尔型	步激活
cycleCounter	双整数	步循环计数
singleStepDone	布尔型	单步完成
complete	布尔型	全部完成

输入/输出变量

名称	数据类型	描述
arrayRampData	typeRampData	斜坡函数信息数组

静态变量及临时变量定义如下图所示：

FB5028_RampGeneratorEx

	名称	数据类型	默认值	保持	设定值	注释
11	Static					
12	statValueDiffer	Real	0.0	非保持		差值
13	statValueDifferUnit	Real	0.0	非保持		单位差值
14	IEC_Timer_TCON_Inst...	TON_TIME		非保持		延时接通定时器
15	statTimeReached	Bool	false	非保持		定时器时间到达
16	satTimeElapse	Time	T#0ms	非保持		定时器流逝时间
17	statEnableTimer	Bool	false	非保持		使能定时器
18	statStepTotalCounter	DInt	0	非保持		总步数
19	statStepCounter	DInt	0	非保持		步计数器
20	statValueOutput	Real	0.0	非保持		输出值
21	statUBound	DInt	0	非保持		数组上限
22	statLBound	DInt	0	非保持		数组下限
23	statArrayCount	DInt	0	非保持		数组元素总数
24	statCycleCounter	DInt	0	非保持		循环计数器
25	statStart	Bool	false	非保持		启动斜坡发生器
26	statStartRisingEdge	Bool	false	非保持		启动信号上升沿
27	statStartRisingEdgeHF	Bool	false	非保持		启动信号上升沿辅助变量
28	statSingleStepDone	Bool	false	非保持		单步完成
29	statSingleStepDoneRis...	Bool	false	非保持		单步完成上升沿
30	statSingleStepDoneRis...	Bool	false	非保持		单步完成上升沿辅助变量
31	statStepActive	Bool	false	非保持		步激活
32	statRampTime	DInt	0	非保持		斜坡时间
33	statMiniCycleFinished	Bool	false	非保持		微步(一次定时器运行)完成
34	statComplete	Bool	false	非保持		完全完成
35	Temp					
36	tmpRealStepCounters	Real				临时变量
37	tmpStartValue	Real				初始值
38	tmpTargetValue	Real				目标值
39	i	DInt				循环计数
40	Constant					
41	<新增>					

注意：

变量**IEC_Timer_TCON_Instance**是定时器指令TON在多重背景数据块下自动生成的静态变量。

FB5028_RampGeneratorEx的代码如下图所示（注释部分详细介绍了代码功能）：

```
(*
Copyrights@Founderchip
=============================================================
功能说明：可连续变化的斜坡函数发生器。
          用户通过外部数组提供斜坡发生器所需要的信息，比如目标值和时间，
          给出启动信号后，会从初始值逐渐上升到目标值；
          例如：
          第一次：
            启动值（startValue） = 初始值
            目标值（targetValue）= 数组第一个元素的目标值
            斜坡时间（rampTime） = 数组第一个元素的斜坡时间
          第二次：
            启动值（startValue） = 数组第一个元素的目标值
            目标值（targetValue）= 数组第二个元素的目标值
            斜坡时间（rampTime） = 数组第二个元素的斜坡时间
          依次类推
依赖：
        外部数组的数据类型为用户自定义类型：typeRampData,包括三个元素：
        ------------------------------------------------
        start           布尔型          启动信号
        targetValue     实数            目标值
        rampTime        双整数          斜坡时间（单位：秒，默认值=10）
        ------------------------------------------------

输入：
    enable          :       使能（TRUE=使能，FALSE=复位）
    initValue       :       初始值
输出：
    value           :       输出值
    stepActive      :       步激活
    cycleCounter    :       步循环计数
    singleStepDone  :       单步完成
    complete        :       全部完成
输入/输出：
    arrayRampData   :       斜坡函数信息数组
作者：
    北岛李工
    2023-8-31
-------------------------------------------------------------
修改日志：

2023-8-31 v1.0版本（首发）          北岛李工
=============================================================
*)

//若未使能
IF NOT #enable THEN
    //复位定时器及静态变量
    #statStepCounter := 0;
    #statCycleCounter := 0;
    #statValueOutput := 0;
    #statEnableTimer := FALSE;
    #statComplete := FALSE;
    #statSingleStepDone := FALSE;
    GOTO _output;   //跳转到输出
END_IF;
```

```
REGION preparation
    // 准备工作
    //获取数组上限
    #statUBound := UPPER_BOUND(ARR := #arrayRampData, DIM := 1);
    //获取数组下限
    #statLBound := LOWER_BOUND(ARR := #arrayRampData, DIM := 1);
    //计算数组元素个数
    #statArrayCount := #statUBound - #statLBound + 1;
    //单步完成（一次斜坡处理称为一步，比如从初始值到数组中的第一个斜坡值）
    //判断上升沿信号
    #statSingleStepDoneRisingEdge := #statSingleStepDone AND NOT #statSingleStepDoneRisingEdgeHF;
    #statSingleStepDoneRisingEdgeHF := #statSingleStepDone;
    IF #statSingleStepDoneRisingEdge THEN
        //复位步激活
        #statStepActive := FALSE;
        //循环计数加1
        #statCycleCounter += 1;
        IF #statCycleCounter = #statArrayCount THEN
            #statComplete := TRUE;
            //重新开始
            #statCycleCounter := 0;
        END_IF;
        #statStepCounter := 0;
    END_IF;
    //计算索引值
    #i := #statLBound + #statCycleCounter;
    //获取启动信号
    #statStart := #arrayRampData[#i].start;
    //获取斜坡时间
    #statRampTime := #arrayRampData[#i].rampTime;
    //获取起始值和目标值
    CASE #statCycleCounter OF
        0:
            #tmpStartValue := #initValue;
            #tmpTargetValue := #arrayRampData[#i].targetValue;
        ELSE
            #tmpStartValue := #arrayRampData[#i - 1].targetValue;

            #tmpTargetValue := #arrayRampData[#i].targetValue;
    END_CASE;
END_REGION

REGION stepInitialize
    //初始化数据
    IF #statStepCounter = 0 THEN
        //计算数值差值
        #statValueDiffer := #tmpTargetValue - #tmpStartValue;
        //计算需要的时间周期
        #statStepTotalCounter := #statRampTime * 10;
        #tmpRealStepCounters := DINT_TO_REAL(#statStepTotalCounter);
        //计算单位值
        #statValueDifferUnit := #statValueDiffer / #tmpRealStepCounters;
    END_IF;
END_REGION
```

```
REGION stepActive
    //上升沿信号判断
    #statStartRisingEdge := #statStart AND NOT #statStartRisingEdgeHF;
    #statStartRisingEdgeHF := #statStart;
    //上升沿
    IF #statStartRisingEdge THEN
        #statStepActive := TRUE;
        #statSingleStepDone := FALSE;
        #statComplete := FALSE;
    END_IF;
    //延时接通定时器
    //时基=100ms
    //用于斜坡时间的计时
    #statEnableTimer := #statStepActive AND NOT #statMiniCycleFinished;
    #IEC_Timer_TCON_Instance(IN := #statEnableTimer,
                             PT := T#100ms,
                             Q => #statTimeReached,
                             ET => #satTimeElapse);
    //为了重新启动定时器
    IF #statMiniCycleFinished THEN
        #statMiniCycleFinished := FALSE;
    END_IF;
    //计时时间到
    IF #statTimeReached THEN
        IF #statStepCounter < #statStepTotalCounter THEN
            #statStepCounter += 1;//计数值加1
            #tmpRealStepCounters := DINT_TO_REAL(#statStepCounter);
            #statValueOutput := #tmpStartValue + #statValueDifferUnit * #tmpRealStepCounters;
            #statMiniCycleFinished := true;
        ELSE
            //单步完成
            #statSingleStepDone := TRUE;
            //复位
            #statStepCounter := 0;
        END_IF;
    END_IF;
END_REGION

_output:
REGION output
    //输出
    #complete := #statComplete;
    #value := #statValueOutput;
    #stepActive := #statStepActive;
    #singleStepDone := #statSingleStepDone;
    #cycleCounter := #statCycleCounter;
END_REGION
```

为了测试代码，我定义了全局数据块DB_test，它包含了斜坡信息数据及相关变量，如下图所示：

DB_test

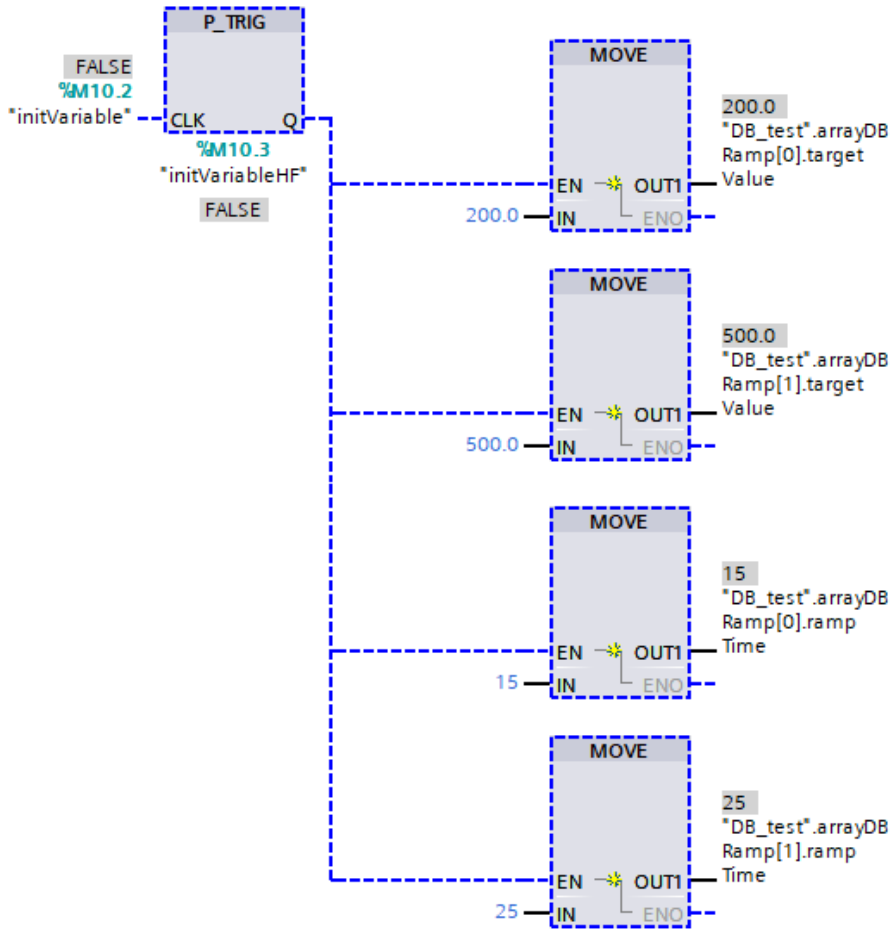
	名称	数据类型	起始值	保持	监控	注释
1	Static					
2	arrayDBRamp	Array[0..1] of "typeRampData"				
3	arrayDBRamp[0]	"typeRampData"				
4	arrayDBRamp[1]	"typeRampData"				
5	cycleCounter	DInt	0			
6	singleStepDone	Bool	false			
7	stepActive	Bool	false			

在OB1中编写代码测试如下图所示：

块接口

程序段 1： ramp test

注释

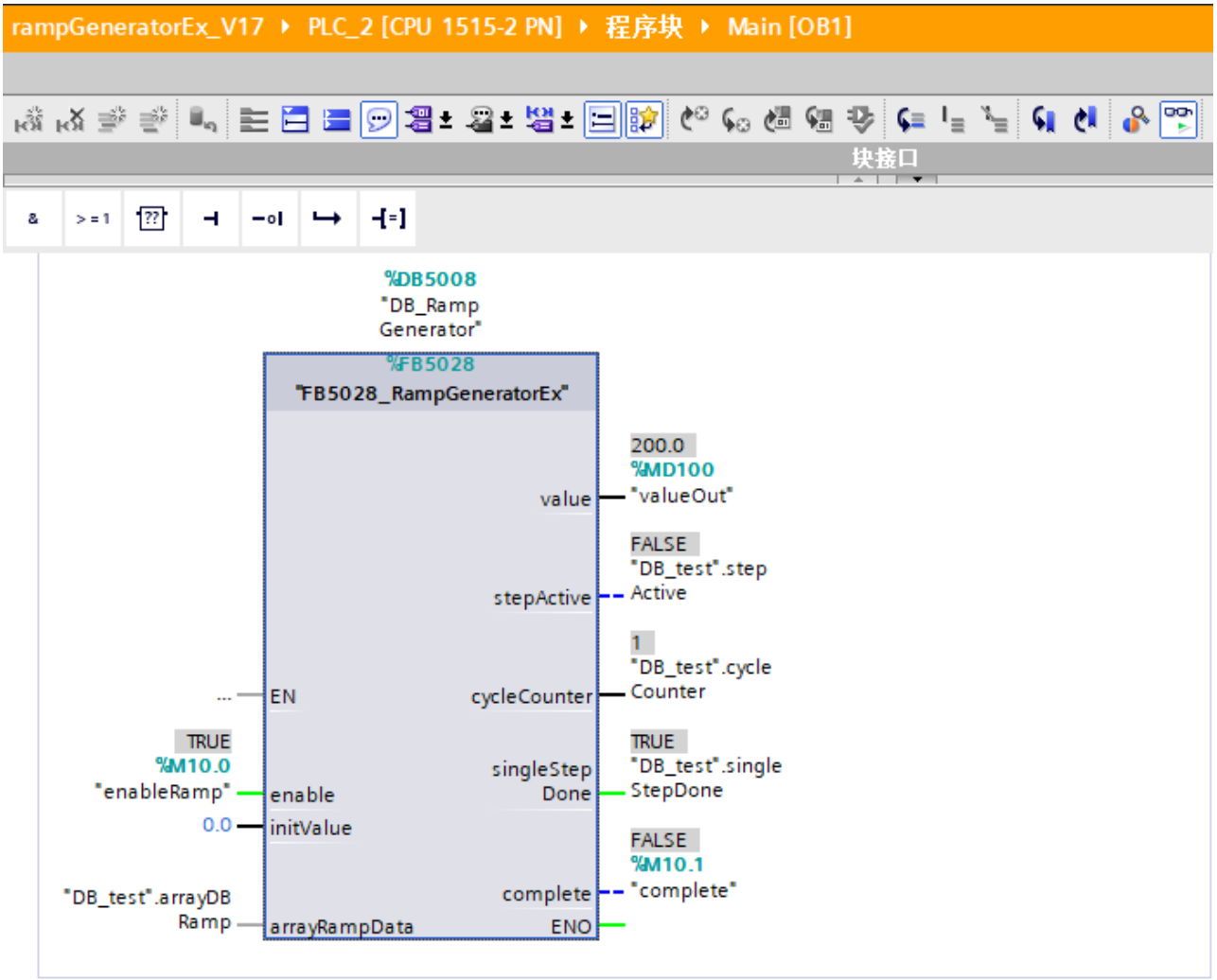


程序段 2： 开始启动斜坡发生器，上升沿有效

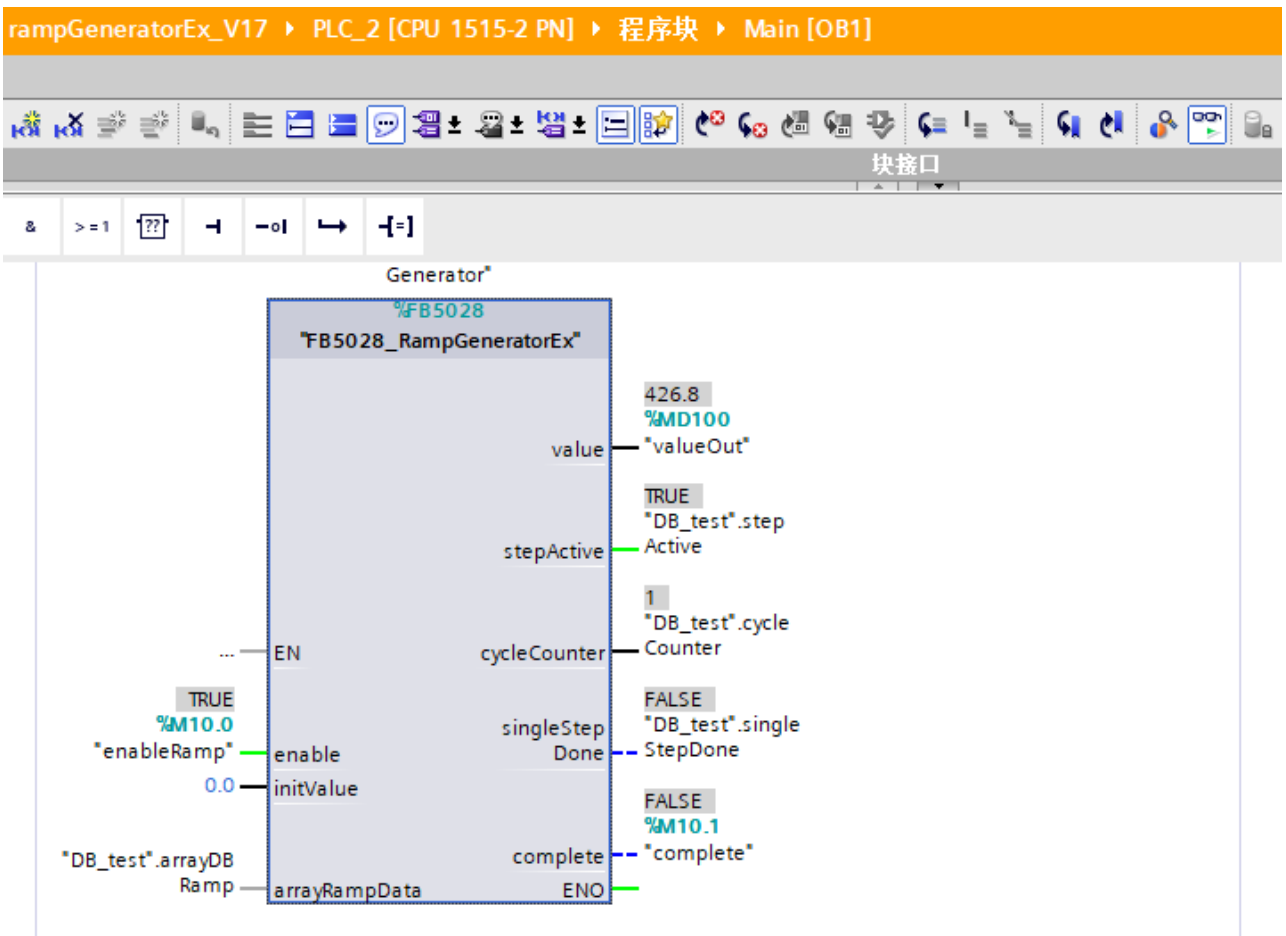
注释

程序段 3： 开始启动斜坡发生器，上升沿有效

注释



启动二次斜坡发生测试示例如下图：

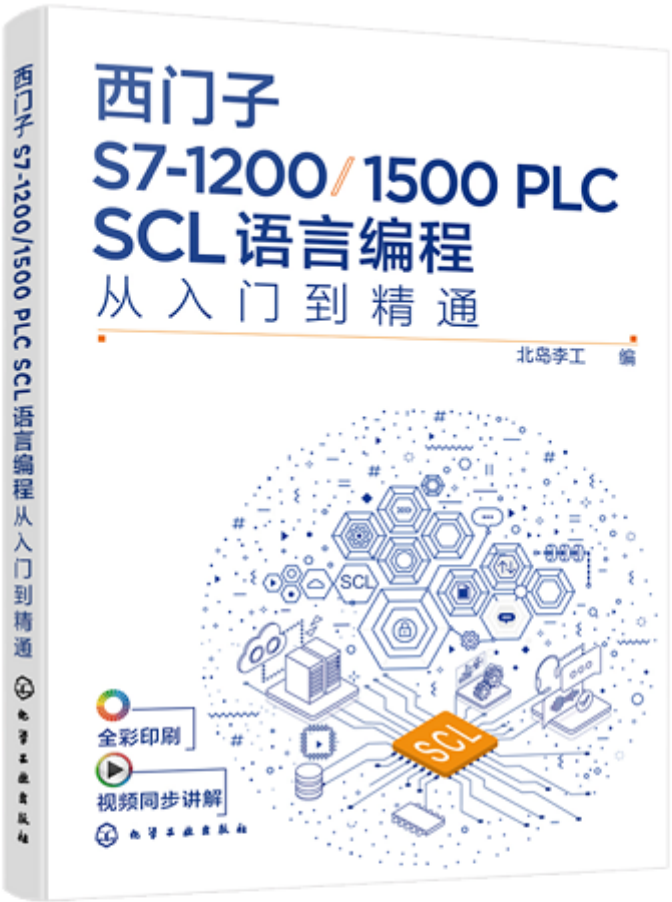


该函数块的代码我已经测试通过，如果你有任何问题，欢迎留言。

下面是西门子SCL编程的文章归档链接：

》》西门子SCL编程入门到精通文章归档《《

我的书《西门子S7-1200/1500 PLC SCL语言编程 ——从入门到精通》从硬件到软件，比较详细的介绍了SCL语言的编程，感兴趣的话可以扫描下面的二维码查看：



识别图中小程序码购买