

0628 【万泉河】优雅的 PLC 程序一定是用 EXCEL 写出来的

我有一些认知盲区。

我一直以为，许多能谈得上 PLC 编程高手的同行，EXCEL 的灵活运用肯定不在话下，至少不会比办公室文员的小姑娘们水平差。

所以，以往我写文章，甚至写书，到接近 EXCEL 的时候就点到为止，不再多说了。

怕被人指责内容太水，没有技术含量。

当然，同时也怕被其他行业的人看到，笑话咱们搞高大上的自动化程序，竟然连 EXCEL 的技巧都还掰扯不清。

所以，我在《PLC 标准化编程原理与方法》书中，明确把 EXCEL 技能列在技能需求第二项，重要程度 8，难度系数 2。

因为 EXCEL 技能现在相当于通用技能，而不是局限于 PLC 行业之内。所以要提升技能，或者获取答案的方法非常多，即便有解决不了的问题，网上随便搜索一下，分分钟可以搞定。

所以，我给标准化学员讲课时也从来不会涉及这个方面，只需要论证出程序的整体结构，告诉他们一句话，这部分可以用 EXCEL 生成，后面的操练就他们自己搞了。而其实我给的资料里面，有时候会夹杂了我做项目时的 EXCEL 中间文件，他们或许可以借用我的 EXCEL 文件当模板，自己做项目的时候可以用到。

但单独的培训交接，是绝对不会有的。

所以，缺省我会认为，跟我学习的学员一定都具备了 EXCEL 的基本功。当然每个人不一样，如果他自己知道自己这方面有欠缺，自然会私下去补上。不需要在我这儿添麻烦。

所以，我做的 80 工位双联开关的例子，把程序脚本列在那里，在我看来，就足够优雅了，就自个儿去得意洋洋了。

比如 SMART200 的例子里面备注了一下：

后面 79 个工位调用用 STL 编写

```
//CALL L31_工位控制, GW01_SIG, LAMP, GW01_SAV  
CALL L31_工位控制, GW02_SIG, LAMP, GW02_SAV  
CALL L31_工位控制, GW03_SIG, LAMP, GW03_SAV  
CALL L31_工位控制, GW04_SIG, LAMP, GW04_SAV  
CALL L31_工位控制, GW05_SIG, LAMP, GW05_SAV  
CALL L31_工位控制, GW06_SIG, LAMP, GW06_SAV  
CALL L31_工位控制, GW07_SIG, LAMP, GW07_SAV
```

```
CALL L31_工位控制, GW08_SIG, LAMP, GW08_SAV  
CALL L31_工位控制, GW09_SIG, LAMP, GW09_SAV  
CALL L31_工位控制, GW10_SIG, LAMP, GW10_SAV
```

.....

而至于怎么写出来的， 这么简单的有规律的程序脚本， EXCEL 分分钟可以搞定的呀！

昨天发表了文章《0627 【万泉河】程序算法的本质》，其中提到了我 3 年前写过的文章《201907 【万泉河】 PLC 编程中的循环语法使用》。

原本想得很好呢， 咱技术进步一步一个脚印， 以前一篇文章的结论为基础， 发展理论到最新的认知。 没想到， 在群里吵成一团， 一大票人对前一篇文章的观点压根不认同。 你不管怎么表达， 他都一口认定用循环总是最省事的， 能节省编程工作量。

我就奇怪了， 你们只看到循环那几句指令省事了， 可前处理后处理的把 IO 映射到数组中的程序怎么写的， 怎么生成的， 那些不是工作量吗？

纠结了很久， 才搞明白， 很多人写程序原来都是从来不使用 EXCEL 的， 甚至许多 EXCEL 的技巧压根不晓得。 难怪行业中总有那么多文章教程和视频在宣讲调用模拟量循环调用， 当成提高编程效率的不二法门， 也难怪有人反而倒打一耙把我反对用循环， 提倡减少用循环的文章当成流毒了。 感情是你们只会用程序软件的编辑器写程序， 从来不会用 EXCEL 写程序的呀！

好吧， 那我就从最水货的 EXCEL 技巧出发， 逐步演示用 EXCEL 生成 PLC 程序的方法吧！ 最终也做个 80 个模拟量调用的示例程序， 会与 80 工位双联开关程序合并到一个系列中。 当然， 前提基础仍然是所有 PLC 平台均兼容的方法， 所以不必在意我文章中演示是用的什么 PLC 平台。

把上述的 STL 程序生成的第一个实例的程序复制到 EXCEL 中， 然后拖拽单元格右下角的小黑点拖到 80 行， 并选择填充序列：

A1		fx	CALL L31_工位控制, GW01_SIG, LAMP, GW01_SAV
	A		B
1	CALL L31_工位控制, GW01_SIG, LAMP, GW01_SAV		
2	CALL L31_工位控制, GW01_SIG, LAMP, GW02_SAV		
3	CALL L31_工位控制, GW01_SIG, LAMP, GW03_SAV		
4	CALL L31_工位控制, GW01_SIG, LAMP, GW04_SAV		
5	CALL L31_工位控制, GW01_SIG, LAMP, GW05_SAV		
6	CALL L31_工位控制, GW01_SIG, LAMP, GW06_SAV		
7	CALL L31_工位控制, GW01_SIG, LAMP, GW07_SAV		
8	CALL L31_工位控制, GW01_SIG, LAMP, GW08_SAV		
9	CALL L31_工位控制, GW01_SIG, LAMP, GW09_SAV		
10	CALL L31_工位控制, GW01_SIG, LAMP, GW10_SAV		
11	CALL L31_工位控制, GW01_SIG, LAMP, GW11_SAV		
12	CALL L31_工位控制, GW01_SIG, LAMP, GW12_SAV		
13	CALL L31_工位控制, GW01_SIG, LAMP, GW13_SAV		
14	CALL L31_工位控制, GW01_SIG, LAMP, GW14_SAV		
15	CALL L31_工位控制, GW01_SIG, LAMP, GW15_SAV		
16	CALL L31_工位控制, GW01_SIG, LAMP, GW16_SAV		
17	CALL L31_工位控制, GW01_SIG, LAMP, GW17_SAV		
18			

会发现， 程序中 2 个数字序列， 但 EXCEL 只给文本中的最后一个数字生成序列。 这怎么办？

方法是文本复制到 AB 两列， 然后各自删掉头和尾， 保证数字分到了两个列。 然后这时候对这两个单元格同时拖拽 80 行， 即生成了 80 行调用程序。

A1		fx	CALL L31_工位控制, GW01_SIG,
	A		B
1	CALL L31_工位控制, GW01_SIG,		LAMP, GW01_SAV
2	CALL L31_工位控制, GW02_SIG,		LAMP, GW02_SAV
3	CALL L31_工位控制, GW03_SIG,		LAMP, GW03_SAV
4	CALL L31_工位控制, GW04_SIG,		LAMP, GW04_SAV
5	CALL L31_工位控制, GW05_SIG,		LAMP, GW05_SAV
6	CALL L31_工位控制, GW06_SIG,		LAMP, GW06_SAV
7	CALL L31_工位控制, GW07_SIG,		LAMP, GW07_SAV
8	CALL L31_工位控制, GW08_SIG,		LAMP, GW08_SAV
9	CALL L31_工位控制, GW09_SIG,		LAMP, GW09_SAV
10	CALL L31_工位控制, GW10_SIG,		LAMP, GW10_SAV
11	CALL L31_工位控制, GW11_SIG,		LAMP, GW11_SAV
12	CALL L31_工位控制, GW12_SIG,		LAMP, GW12_SAV
13	CALL L31_工位控制, GW13_SIG,		LAMP, GW13_SAV

直接选中， 复制内容到程序中， 直接可用。

当然， 其实这样复制的内容中有表格分隔符， 也可以另外生成一个 C 列， 里面的公式填入：  
=A1&B1, 同样拖拽到 80 行。 生成了 80 行结果。 或者在前面两列已经生成 80 行的情况下， 直接双击黑点， 也可以。

由此得到了完整的 80 行程序调用。

然而，这里 80 个工位编号完整整齐的从 01 递增到 80，是我为了例子生成便捷，刻意安排的。而实际的工程项目中，位号通常不连续。比如：GW1001 GW1002 GW1003  
GW1004 GW1005 GW1006 GW1007 GW1008 GW1009 GW1010 GW2001 GW2002  
GW2003 GW2004 GW2005 GW2006 GW3001 GW3002 GW3003 GW3004 GW3005  
GW3006.....总计 80 个。

首先把上述的位号数据复制到 A 列，这里是一行数据，可以先复制到一个行中，然后选择性粘贴，转置，把行排列的数据转置成了列。

程序调用的位号部分修改到 AAAA，即：

CALL L31\_工位控制, AAAA\_SIG, LAMP, AAAA\_SAV  
复制到 B 列所有行。

C2 中填入公式：

=SUBSTITUTE(B2,"AAAA",A2)

意思为把 B 列中的 AAAA 字符的部分替换为 A2 单元格的内容。

即得到了目标的程序，拖拽到底，则生成了所有程序：

	A	B	C
1	位号	程序模板	替换位号
2	GW1001	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW1001_SIG, LAMP, GW1001_SAV
3	GW1002	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW1002_SIG, LAMP, GW1002_SAV
4	GW1003	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW1003_SIG, LAMP, GW1003_SAV
5	GW1004	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW1004_SIG, LAMP, GW1004_SAV
6	GW1005	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW1005_SIG, LAMP, GW1005_SAV
7	GW1006	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW1006_SIG, LAMP, GW1006_SAV
8	GW1007	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW1007_SIG, LAMP, GW1007_SAV
9	GW1008	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW1008_SIG, LAMP, GW1008_SAV
10	GW1009	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW1009_SIG, LAMP, GW1009_SAV
11	GW1010	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW1010_SIG, LAMP, GW1010_SAV
12	GW2001	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW2001_SIG, LAMP, GW2001_SAV
13	GW2002	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW2002_SIG, LAMP, GW2002_SAV
14	GW2003	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW2003_SIG, LAMP, GW2003_SAV
15	GW2004	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW2004_SIG, LAMP, GW2004_SAV
16	GW2005	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW2005_SIG, LAMP, GW2005_SAV
17	GW2006	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW2006_SIG, LAMP, GW2006_SAV
18	GW3001	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW3001_SIG, LAMP, GW3001_SAV
19	GW3002	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW3002_SIG, LAMP, GW3002_SAV
20	GW3003	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW3003_SIG, LAMP, GW3003_SAV
21	GW3004	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW3004_SIG, LAMP, GW3004_SAV
22	GW3005	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW3005_SIG, LAMP, GW3005_SAV
23	GW3006	CALL L31_工位控制, AAAA_SIG, LAMP, AAAA_SAV	CALL L31_工位控制, GW3006_SIG, LAMP, GW3006_SAV

然后模拟量转换程序的调用。

模拟量程序的特点是，输入的参数很多，每一个模拟量的标定数据上下限，物理单位等都不一样，来自工艺统计的位号表，如：

A	B	E	F	G	H	I
序号	符号	类型	单位	量程下限	量程上限	注释
19	AI_V019	AI	pa	0	100	DPT-R5
20	AI_V020	AI	pa	0	500	DPT-F5
21	AI_V021	AI	°C	-5	55	THT-R6-T
22	AI_V022	AI	%	0	100	THT-R6-RH
23	AI_V023	AI	pa	0	100	DPT-R6
24	AI_V024	AI	pa	0	500	DPT-F6
25	AI_V025	AI	°C	-5	55	THT-R7-T
26	AI_V026	AI	%	0	100	THT-R7-RH
27	AI_V027	AI	pa	0	100	DPT-R7
28	AI_V028	AI	pa	0	500	DPT-F7
29	AI_V029	AI	°C	-5	55	THT-R8-T
30	AI_V030	AI	%	0	100	THT-R8-RH
31	AI_V031	AI	pa	0	100	DPT-R8
32	AI_V032	AI	pa	0	500	DPT-F8
33	AI_V033	AI	°C	-5	55	THT-R9-T
34	AI_V034	AI	%	0	100	THT-R9-RH
35	AI_V035	AI	pa	0	100	DPT-R9
36	AI_V036	AI	pa	0	500	DPT-F9

工艺表中还必然另外存在一些数据信息列，我们不关心的，只需要隐藏即可，留下的内容都需要生成到程序中。包括字符类型的注释和单位部分，我们也不愿意亲自手工二次录入，在 PLC 支持的情况下，可以直接做到 FB 的管脚上，最终不仅仅程序中直观可见，字符数据还可以传到上位机中，上位组态时也不必再包含这部分录入的工作量了。

那么，程序模板会是：

```
"//#AAAA(IN_INT:=""AAAA",
      HI_LIM:=CCCC,LO_LIM:=BBBB,
      INSTANCE:='DDDD',unit:='EEEE');"
```

其中除了信号名称 AAAA 需要替换之外，后面的 BBBB,CCCC,DDDD,EEEE 也分别替换为表格内的内容。

把模板所在的单元格起名字定义为 AI\_1500，替换语法设置为：

```
=SUBSTITUTE(SUBSTITUTE(SUBSTITUTE(SUBSTITUTE(SUBSTITUTE(SUBSTITUTE(AI_1500,"AAAA",B
2),"BBBB",G2),"CCCC",H2),"DDDD",I2),"EEEE",F2),CHAR(10)," ")
```

最终生成了程序：

```
"AI_V019"(IN_INT:="AI_V019",      HI_LIM:=100,LO_LIM:=0,      INSTANCE:='DPT-R5',unit:='pa',
QOUT=>"HMI".AI.AI_V019);
"AI_V020"(IN_INT:="AI_V020",      HI_LIM:=500,LO_LIM:=0,      INSTANCE:='DPT-F5',unit:='pa',
QOUT=>"HMI".AI.AI_V020);
```

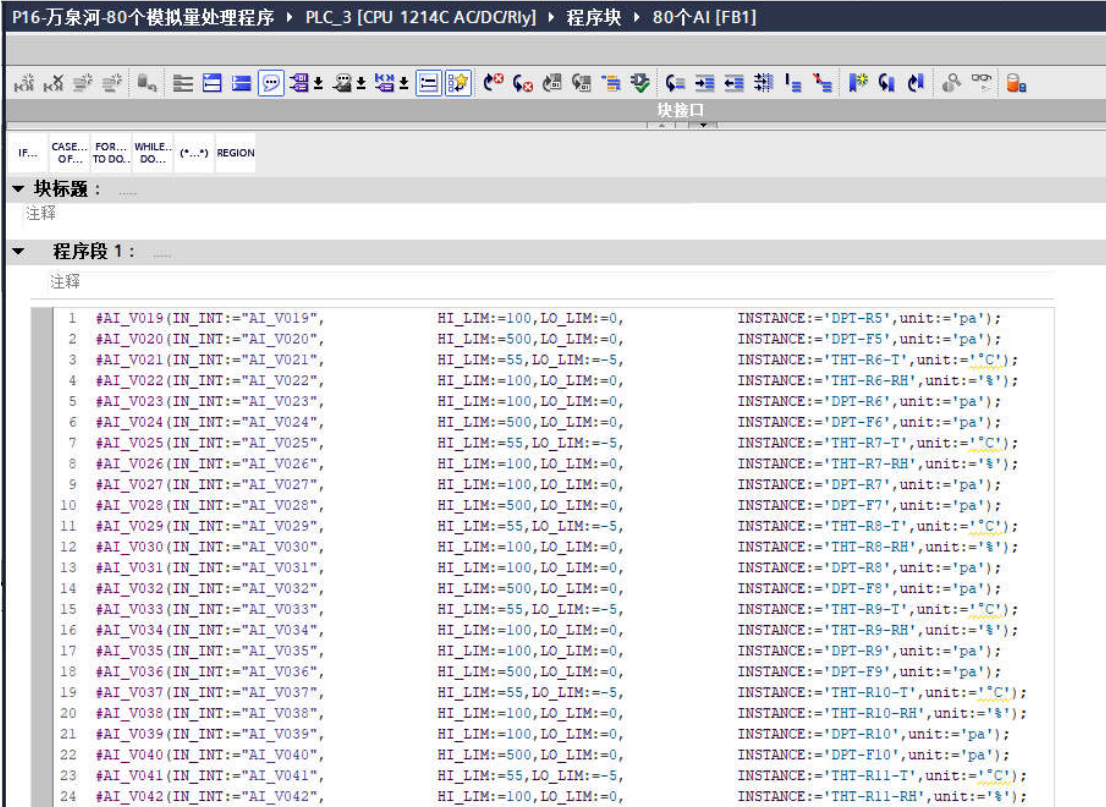


```
"AI_V021"(IN_INT:="AI_V021", HI_LIM:=55,LO_LIM:=-5, INSTANCE:='THT-R6-T',unit:='°C',  
QOUT=>"HMI".AI.AI_V021);
```

```
"AI_V022"(IN_INT:="AI_V022", HI_LIM:=100,LO_LIM:=0, INSTANCE:='THT-R6-RH',unit:='%',  
QOUT=>"HMI".AI.AI_V022);
```

```
"AI_V023"(IN_INT:="AI_V023", HI_LIM:=100,LO_LIM:=0, INSTANCE:='DPT-R6',unit:='pa',  
QOUT=>"HMI".AI.AI_V023);
```

把程序直接复制到 PLC 软件中，即可。



```
P16-万泉河-80个模拟量处理程序 ▸ PLC_3 [CPU 1214C AC/DCRly] ▸ 程序块 ▸ 80个AI [FB1]  
块接口  
IF... CASE... FOR... WHILE... DO... DO... (*...*) REGION  
块标题 :  
注释  
程序段 1 :  
注释  
1 #AI_V019(IN_INT:="AI_V019", HI_LIM:=100,LO_LIM:=0, INSTANCE:='DPT-R5',unit:='pa');  
2 #AI_V020(IN_INT:="AI_V020", HI_LIM:=500,LO_LIM:=0, INSTANCE:='DPT-F5',unit:='pa');  
3 #AI_V021(IN_INT:="AI_V021", HI_LIM:=55,LO_LIM:=-5, INSTANCE:='THT-R6-T',unit:='°C');  
4 #AI_V022(IN_INT:="AI_V022", HI_LIM:=100,LO_LIM:=0, INSTANCE:='THT-R6-RH',unit:='%');  
5 #AI_V023(IN_INT:="AI_V023", HI_LIM:=100,LO_LIM:=0, INSTANCE:='DPT-R6',unit:='pa');  
6 #AI_V024(IN_INT:="AI_V024", HI_LIM:=500,LO_LIM:=0, INSTANCE:='DPT-F6',unit:='pa');  
7 #AI_V025(IN_INT:="AI_V025", HI_LIM:=55,LO_LIM:=-5, INSTANCE:='THT-R7-T',unit:='°C');  
8 #AI_V026(IN_INT:="AI_V026", HI_LIM:=100,LO_LIM:=0, INSTANCE:='THT-R7-RH',unit:='%');  
9 #AI_V027(IN_INT:="AI_V027", HI_LIM:=100,LO_LIM:=0, INSTANCE:='DPT-R7',unit:='pa');  
10 #AI_V028(IN_INT:="AI_V028", HI_LIM:=500,LO_LIM:=0, INSTANCE:='DPT-F7',unit:='pa');  
11 #AI_V029(IN_INT:="AI_V029", HI_LIM:=55,LO_LIM:=-5, INSTANCE:='THT-R8-T',unit:='°C');  
12 #AI_V030(IN_INT:="AI_V030", HI_LIM:=100,LO_LIM:=0, INSTANCE:='THT-R8-RH',unit:='%');  
13 #AI_V031(IN_INT:="AI_V031", HI_LIM:=100,LO_LIM:=0, INSTANCE:='DPT-R8',unit:='pa');  
14 #AI_V032(IN_INT:="AI_V032", HI_LIM:=500,LO_LIM:=0, INSTANCE:='DPT-F8',unit:='pa');  
15 #AI_V033(IN_INT:="AI_V033", HI_LIM:=55,LO_LIM:=-5, INSTANCE:='THT-R9-T',unit:='°C');  
16 #AI_V034(IN_INT:="AI_V034", HI_LIM:=100,LO_LIM:=0, INSTANCE:='THT-R9-RH',unit:='%');  
17 #AI_V035(IN_INT:="AI_V035", HI_LIM:=100,LO_LIM:=0, INSTANCE:='DPT-R9',unit:='pa');  
18 #AI_V036(IN_INT:="AI_V036", HI_LIM:=500,LO_LIM:=0, INSTANCE:='DPT-F9',unit:='pa');  
19 #AI_V037(IN_INT:="AI_V037", HI_LIM:=55,LO_LIM:=-5, INSTANCE:='THT-R10-T',unit:='°C');  
20 #AI_V038(IN_INT:="AI_V038", HI_LIM:=100,LO_LIM:=0, INSTANCE:='THT-R10-RH',unit:='%');  
21 #AI_V039(IN_INT:="AI_V039", HI_LIM:=100,LO_LIM:=0, INSTANCE:='DPT-R10',unit:='pa');  
22 #AI_V040(IN_INT:="AI_V040", HI_LIM:=500,LO_LIM:=0, INSTANCE:='DPT-F10',unit:='pa');  
23 #AI_V041(IN_INT:="AI_V041", HI_LIM:=55,LO_LIM:=-5, INSTANCE:='THT-R11-T',unit:='°C');  
24 #AI_V042(IN_INT:="AI_V042", HI_LIM:=100,LO_LIM:=0, INSTANCE:='THT-R11-RH',unit:='%');
```

这里篇幅有限，只复制了其中的前几行。而实际项目中别说 80 个模拟量了，就是 800 个，8000 个，只要工艺来的数据表格规范完整，这些工作量都是秒成的。

比如如果有 8000 个的天量数据，项目所控制的 PLC CPU 至少也几十个。那么只需要在数据表中表明 CPU 的标识，程序生成后按标识复制到相应的 CPU 中即可。啥循环都不需要做。

所有的宣传用循环语法处理模拟量的文章或者视频，只在意了调用部分，而参数的输入部分的工作量都忽略不计了。然而那才是工作量最大的，最令人厌烦的。

模块参数的给定，物理通道的给定等等，最方便的方式恰恰是通过 FB 调用的实例化时给定，因为可以在一行程序语句里面一次性完成。如果只为了循环调用的爽一下，留给数据整理部分的工作量反而增加了，而且分散到整个程序的多个角落去了。查错，维护都成了问题。

所有坚持使用循环语法调用模拟量程序的网友们，不妨尝试把我的程序改进到你们的模式，看看效率能不能再提高些。

本文中提及的数据表格，以及生成的例子程序，我会统一打包在一起，方便下载和借用。

也会归档上传到“80 工位双联开关”的群文件中。

当然这种程序方法也完全可以照猫画虎迁移到其它品牌平台，有兴趣者欢迎钻研一下，完成后署名上传，业界同行会记住你的名字你的贡献的。