SIEMENS
Ingenuity for life

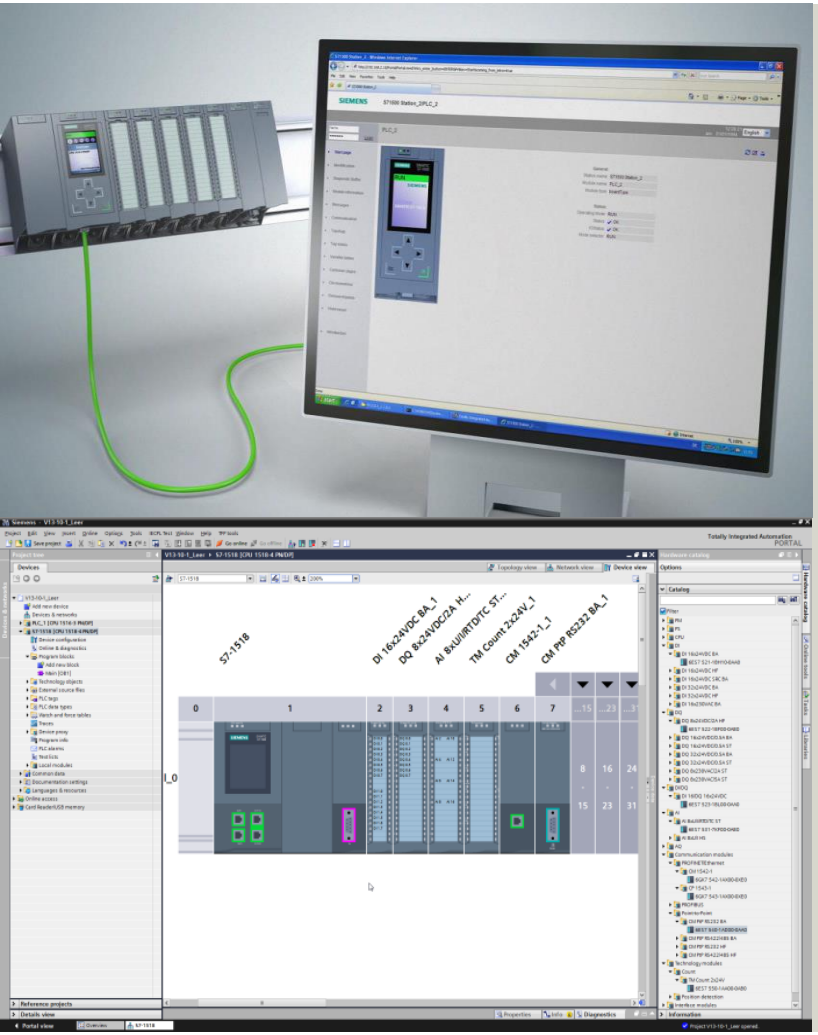
博图环境下通过用户程序实现硬件IO自由组态的基本方法

黄岑（Zane）

2018 西门子工业专家会议 | 苏州 9 月 18 日 - 20 日

博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life



一、技术背景

随着西门子S7-1200/1500系列PLC及TIA PORTAL博图软件应用的逐步推广，在软硬件标准化模块化得到深入应用的基础上，一些高端用户对PLC系统在柔性化应用方面有了更多的需求：

1.在有限的硬件资源前提下最大限度地去满足设备实际运行使用中的各种需求：

比如某一中小型锅炉控制系统，其基本控制功能涉及的控制设备（比如燃烧器，主辅循环泵，主回路阀门，温度压力流量传感器等）是确定不变的，但根据客户需求的不同有些辅助设备（补水泵，风机，补水阀，调节阀，泄压阀，辅助电源等）是否使用及其使用数量是有变化的，由于中小型供暖锅炉的控制系统一般据均采用小型PLC控制系统，硬件IO的扩展能力有限，控制系统的安装空间有限，所以锅炉制造厂家在系统标配的基础上，希望利用系统多余的IO点，能在一定程度上灵活应对上述设备需求的变化，简化控制系统和程序，实现标准化。

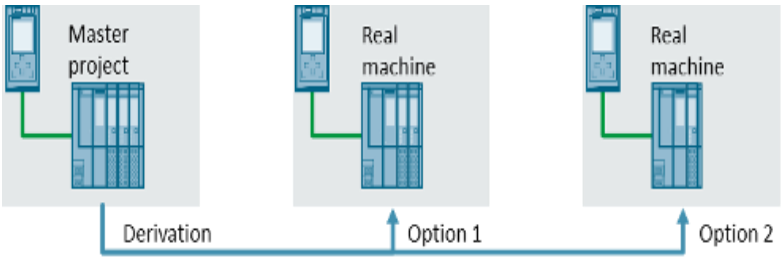
2.在一定规模的硬件资源下力求达到系统的最大灵活性：

比如某汽车总装车间的线边仓科的捡料防错系统，每当车型转换投产，线边仓库的部件库位位置及数量都会有较大的变化，捡料防错系统的指示灯传感器也会被相应地重新部署，这样库位与指示灯传感器对应关系会发生变化，不同型号的指示灯传感器（对应不同的IO数量）使用数量也会发生变化，重新组态库位与指示灯传感器的对应关系，以往需要修改并更新控制系统的程序，这需要专业的工程师才能做，那么如何能实现不修改控制系统程序，仅由普通生产管理人员通过人机界面以修改参数的形式重新组态库位呢？

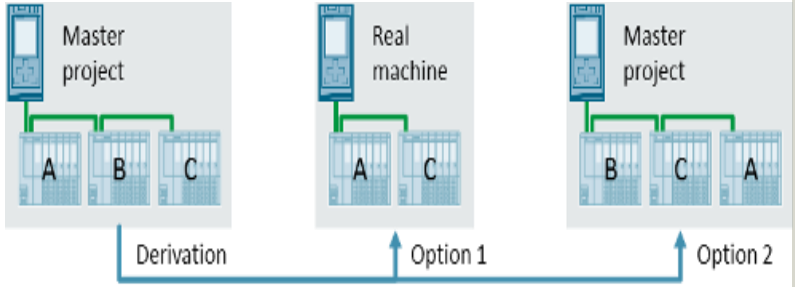
博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life

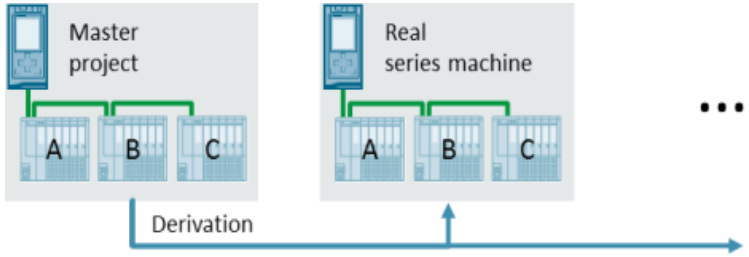
模块级组态控制



IO 系统的组态控制

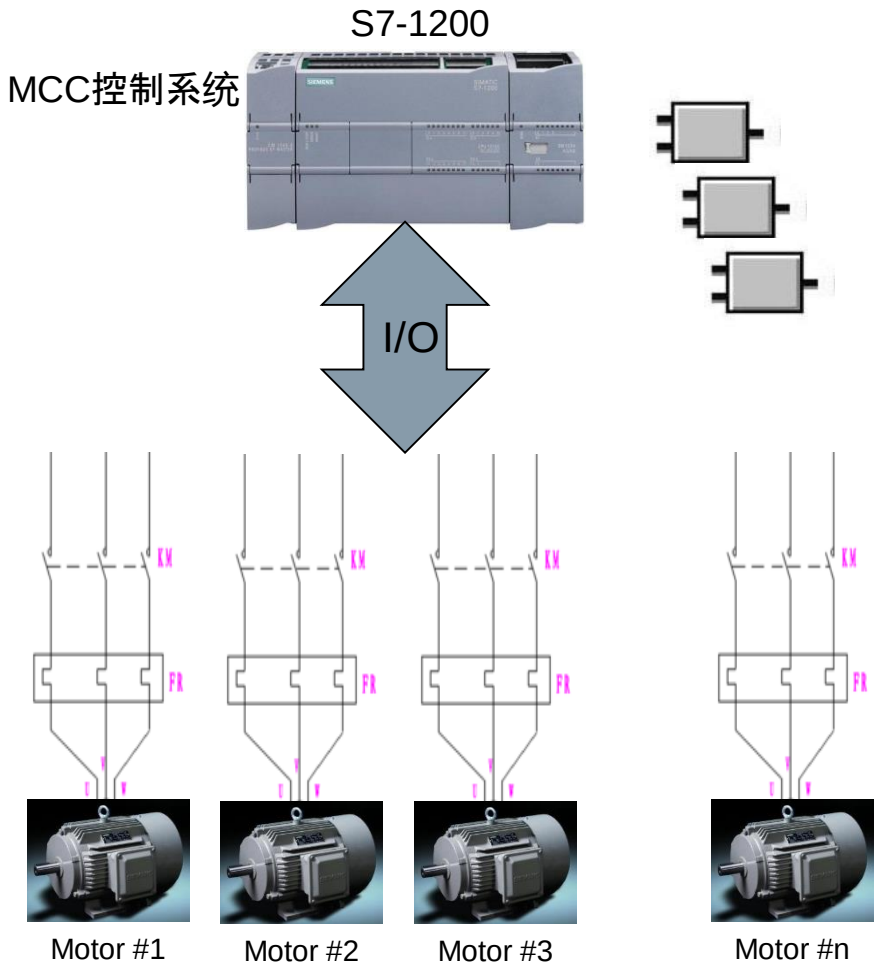


多用途 IO 系统



我们知道，西门子的S7系统早已实现了硬件配置用户程序再组态的功能，S7-300/400/1200/1500/均可由用户程序实现分站一级到模块级别硬件再配置功能，然而这是硬件层面的解决方案，并不能满足上述两个例子的需求，因此还要通过软件层面的配合来最终实现设计要求。在解决上述新的应用需求的实践过程中，作者提出了一种在博图环境下通过用户程序实现硬件IO可自由组态的基本编程方法，抛砖引玉，与大家共同探讨并开拓PLC柔性化系统程序设计的方法与思路。

博图环境下通过用户程序实现硬件IO自由组态的基本方法



为了便于说明问题，我们把目标控制系统简化成一个简单的单纯的MCC控制系统，该MCC控制系统满足如下已知条件：

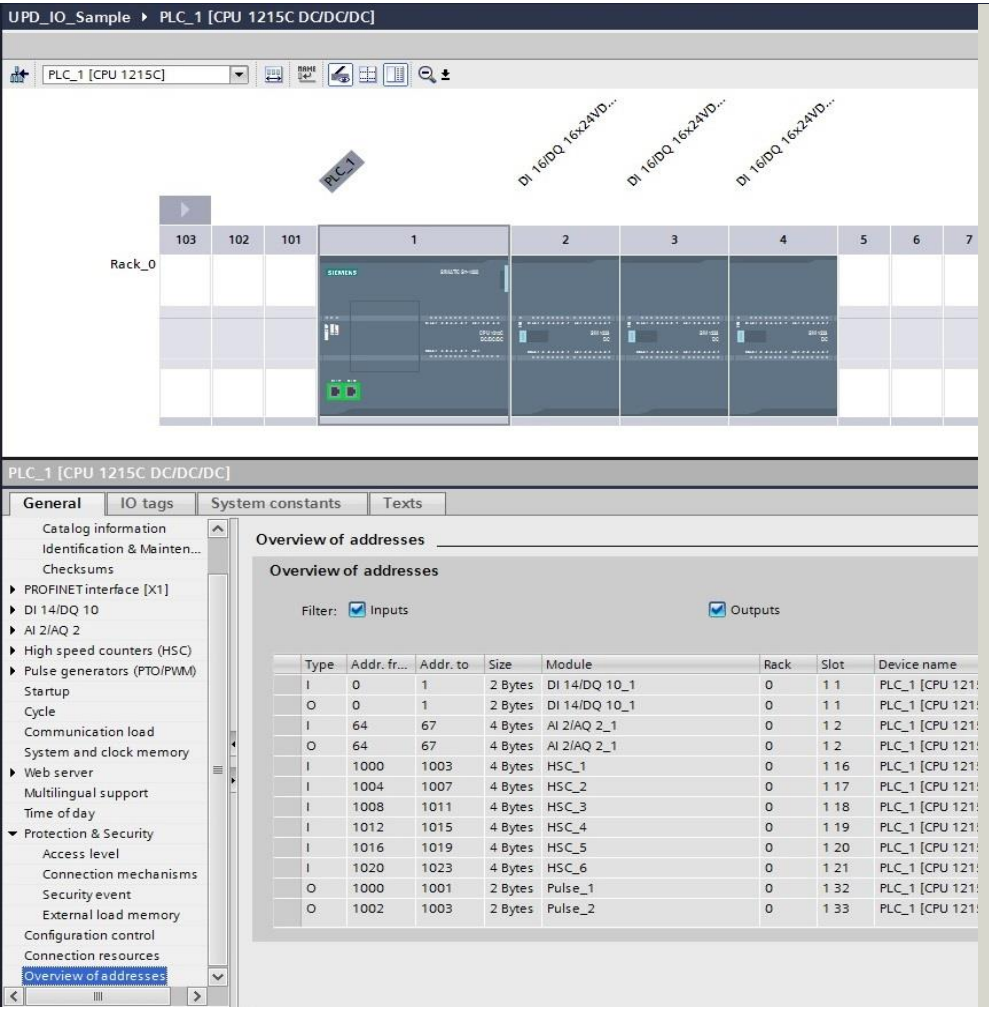
1. 满足若干个直接启停电机的控制的硬件IO的需求；
2. 每个电机包括**启动、停止、故障复位，故障反馈**4个输入信号；**运行输出，故障指示**2个输出信号；
3. 电机控制功能块是已经经过测试并封装好的FB（功能块）；

系统控制程序实现要求：

1. 实现在设计范围内对任意一台电机的可靠控制；
2. 电机控制程序对应的硬件IO点可通过用户程序实现自由定义，使控制系统能按实际硬件IO接线及定义正确地完成控制任务；
3. 必须使用且不能修改已封装的FB块；
4. 不允许修改控制程序及重新下载控制程序（包括数据块）的方式，只能通过在线修改参数的方式来实现控制对象硬件IO的定义；
5. 实现方法力求简单易行，不占用过多的系统资源；
6. 博图V14版本编程环境下，S7-1200/1500程序通用

博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life



二、方案分析与实现：

图1所示系统是一个基于S7-1200PLC的小型MCC控制系统示例，由以下元器件组成：

CPU1215C DC/DC/DC ×1

EM223 16DI DC/16DO DC × 3

是一个以开关量信号为主的简单控制系统，其IO点满足一个小型MCC控制系统的点数需求。

博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life

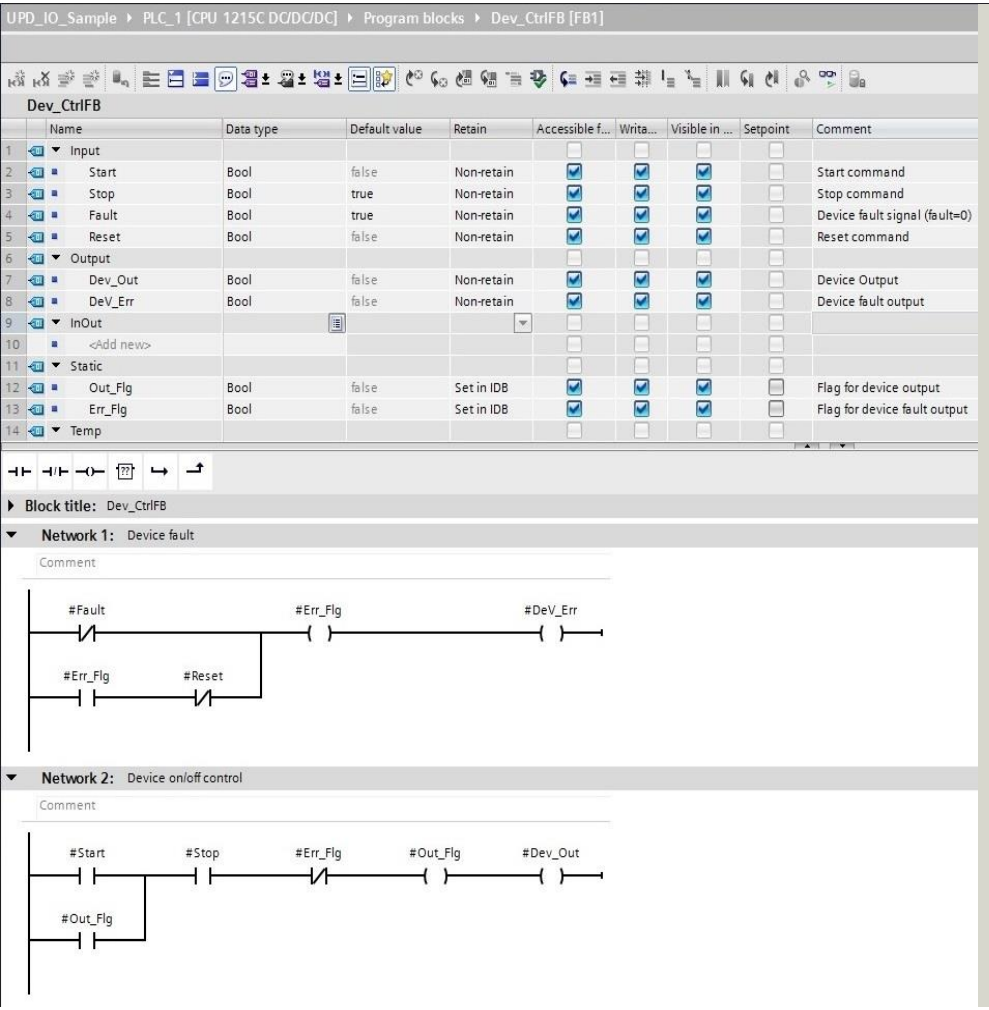
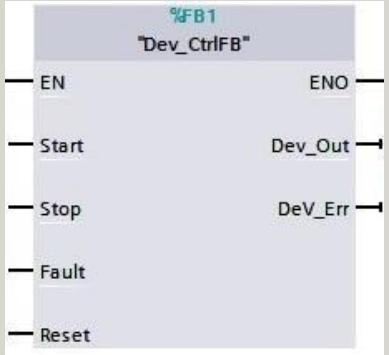


图2是一个常见的启停控制带故障保护逻辑的电机控制功能块，有4个bool输入变量，2个bool输出变量，并且封装成一个标准的功能块FB1，我们以此功能块为例代表已有的成熟的控制功能，希望在不对该功能块做任何改变的基础上，能够实现控制器硬件IO由用户程序自由定义，并且控制系统能按实际硬件IO接线及定义基于该功能块正确地完成MCC控制系统的控制任务。



博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life

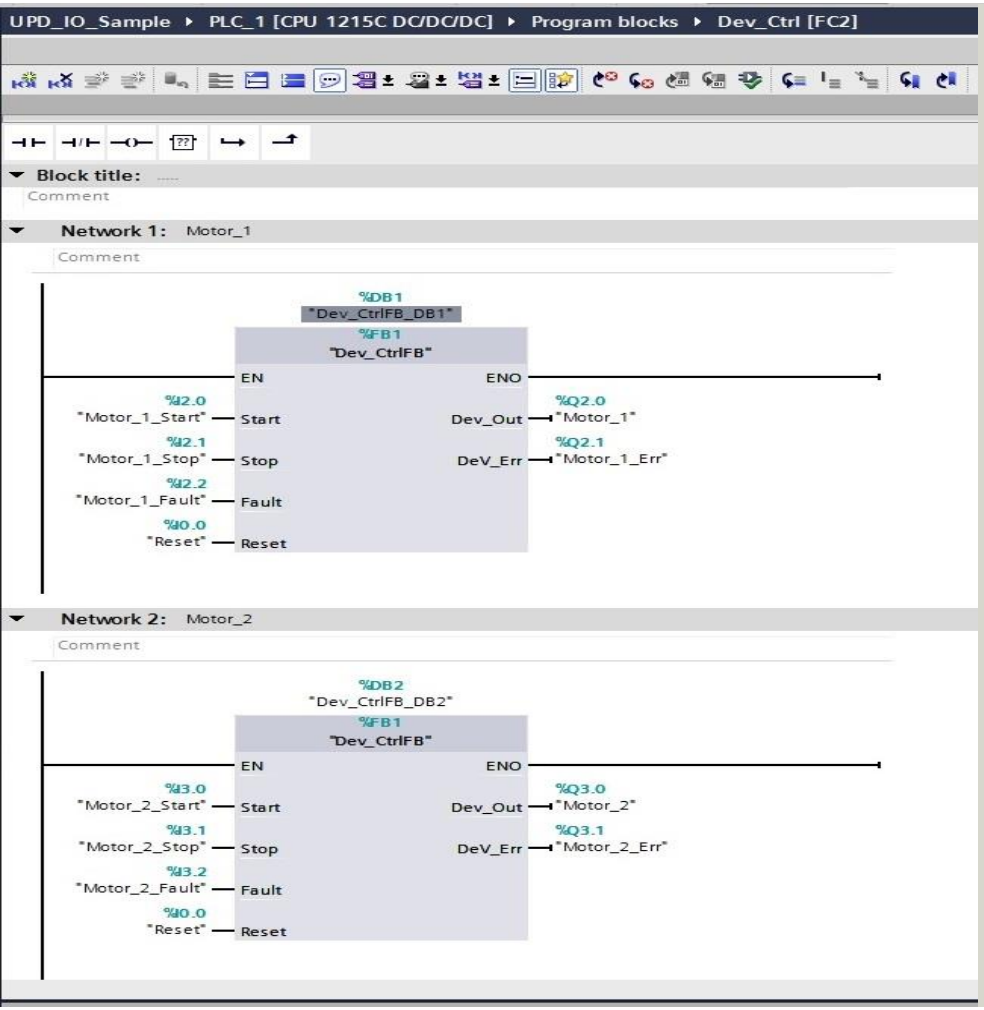


图3为示例中电机直接控制功能块在博图中的传统调用方式，功能块多次调用，生成多个对应的背景数据块，每次调用其输入输出参数均为实参赋值，显式调用，多重背景调用亦是如此。如果某一个电机控制块的对应的硬件IO点需要修改，则需要修改程序并重新下载，系统才能按新的硬件IO对应方式进行工作。

如何可以不需要修改并重新下载程序，仅仅通过修改运行程序的数据，就能改变控制功能块与硬件IO对应关系呢？

显然只有通过间址寻址的方式才能实现硬件IO变址访问的要求，博图V14版本以上只有SCL编程语言的PEEK_BOOL / POKE_BOOL指令可以比较方便的实现对硬件IO位地址的间址寻址，即间址读写访问。

博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life

UPD_IO_Sample ▸ PLC_1 [CPU 1215C DC/DC/DC] ▸ PLC data types ▸ IO_Addr_UDT

IO_Addr_UDT

	Name	Data type	Default value	Accessible f...	Writa...	Visible in ...	Setpoint	Comment
1	Area	Byte	16#0	☑	☑	☑	☐	IO can be defined as Input=16#81 output=16#82
2	Byte_Offset	DInt	0	☑	☑	☑	☐	Address to read from or to be written, Only the 16 least significant bits are used.
3	Bit_Offset	Int	0	☑	☑	☑	☐	Bit to be read from or to be written
4	Value	Bool	false	☑	☑	☑	☐	IO value, to be written for input, to be read for output
5	Status	Byte	16#0	☑	☑	☑	☐	IO status Deactive=16#00 Active=16#01 Fault=16#FF

图4为通过PEEK_BOOL / POKE_BOOL 指令实现硬件IO位地址间址寻址的硬件IO地址说明表，采用UDT形式，以便后续程序中多次调用，变量具体说明如下：

Area: 16#81 代表输入 (Input)
16#82 代表输出 (Output)

Byte_Offset: 字节地址 >= 0

Bit_Offset: 位地址 0 ~ 7位

Value: IO位变量的值，输入为写操作，输出为读操作

Status: 16#00 代表IO未被使用 Deactivated
16#01 代表IO被使用 Activated
16#FF 代表IO说明错误 Fault

博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life

UPD_IO_Sample ▸ PLC_1 [CPU 1215C DC/DC/DC] ▸ Program blocks ▸ IO_Mapping [FC1000

IO_Mapping

	Name	Data type	Default value	Comment
1	Input			
2	Output			
3	InOut			
4	IO_Addr	"IO_Addr_UDT"		IO Address
5	Temp			
6	Constant			

IF... CASE... OF... FOR... TO DO.. WHILE.. DO... (*...*) REGION

Block title: IO_Mapping

Network 1: IO Mapping

Comment

```
IF      #IO_Addr.Area=16#00 THEN                                  // IO not defined
        #IO_Addr.Value := FALSE;                                  // Clear IO value
        #IO_Addr.Status := 16#00;                                 // IO status inactive
        RETURN;                                                   // Exit Function
ELSIF   (#IO_Addr.Area = 16#81) AND
        (#IO_Addr.Byte_Offset >= 0) AND
        (#IO_Addr.Bit_Offset >= 0) AND
        (#IO_Addr.Bit_Offset <= 7) THEN                           // IO defined as input and IO address valid
         #IO_Addr.Value := PEEK_BOOL(area := #IO_Addr.Area,       // Read input
                                     dbNumber := 0,
                                     byteOffset := #IO_Addr.Byte_Offset,
                                     bitOffset := #IO_Addr.Bit_Offset);
         #IO_Addr.Status := 16#01;                                // IO status acitve
         RETURN;                                                  // Exit Function
ELSIF    (#IO_Addr.Area=16#82) AND
         (#IO_Addr.Byte_Offset >= 0) AND
         (#IO_Addr.Bit_Offset >= 0) AND
         (#IO_Addr.Bit_Offset <= 7) THEN                          // IO defined as output and IO address valid
          POKE_BOOL(area := #IO_Addr.Area,                        // Write output
                    dbNumber := 0,
                    byteOffset := #IO_Addr.Byte_Offset,
                    bitOffset := #IO_Addr.Bit_Offset,
                    value := #IO_Addr.Value);
          #IO_Addr.Status := 16#01;                               // IO status active
          RETURN;                                                 // Exit Function
ELSE
          #IO_Addr.Value := FALSE;                                // Clear IO value
          #IO_Addr.Status := 16#FF;                               // IO status mapping fault
END_IF;
```

图5为IO变址映射功能FC1000，SCL语言编写，输入输出通用，根据硬件IO地址说明表中指定的硬件IO地址读取输入点的状态或刷新输出点的状态，每个硬件IO位调用一次.

博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life

UPD_IO_Sample ▸ PLC_1 [CPU 1215C DC/DC/DC] ▸ PLC data types ▸ Obj_IO_UDT_1

Obj_IO_UDT_1

	Name	Data type	Default value	Accessible f...	Writa...	Visible in ...	Setpoint	Comment
1	Start	"IO_Addr_UDT"		☑	☑	☑	☐	
2	Area	Byte	16#0	☑	☑	☑	☐	IO can be defined as Input=16#81 output=16#82
3	Byte_Offset	DInt	0	☑	☑	☑	☐	Address to read from or to be written, Only the 16 leas
4	Bit_Offset	Int	0	☑	☑	☑	☐	Bit to be read from or to be written
5	Value	Bool	false	☑	☑	☑	☐	IO value, to be written for input, to be read for output
6	Status	Byte	16#0	☑	☑	☑	☐	IO status Deactive=16#00 Active=16#01 Fault=16#F
7	Stop	"IO_Addr_UDT"		☑	☑	☑	☐	
8	Fault	"IO_Addr_UDT"		☑	☑	☑	☐	
9	Reset	"IO_Addr_UDT"		☑	☑	☑	☐	
10	Dev_out	"IO_Addr_UDT"		☑	☑	☑	☐	
11	Dev_Err	"IO_Addr_UDT"		☑	☑	☑	☐	
12	<Add new>			☐	☐	☐	☐	

UPD_IO_Sample ▸ PLC_1 [CPU 1215C DC/DC/DC] ▸ PLC data types ▸ Obj_IO_UDT

Obj_IO_UDT

	Name	Data type	Default value	Accessible f...	Writa...	Visible in ...	Setpoint	Comment
1	Obj_IO	Array[0..5] of "IO_Addr_UDT"		☑	☑	☑	☐	
2	Obj_IO[0]	"IO_Addr_UDT"		☑	☑	☑	☐	
3	Area	Byte	16#0	☑	☑	☑	☐	IO can be defined as Input=16#81 output=16#82
4	Byte_Offset	DInt	0	☑	☑	☑	☐	Address to read from or to be written, Only the 16 leas
5	Bit_Offset	Int	0	☑	☑	☑	☐	Bit to be read from or to be written
6	Value	Bool	false	☑	☑	☑	☐	IO value, to be written for input, to be read for output
7	Status	Byte	16#0	☑	☑	☑	☐	IO status Deactive=16#00 Active=16#01 Fault=16#F
8	Obj_IO[1]	"IO_Addr_UDT"		☑	☑	☑	☐	
9	Obj_IO[2]	"IO_Addr_UDT"		☑	☑	☑	☐	
10	Obj_IO[3]	"IO_Addr_UDT"		☑	☑	☑	☐	
11	Obj_IO[4]	"IO_Addr_UDT"		☑	☑	☑	☐	
12	Obj_IO[5]	"IO_Addr_UDT"		☑	☑	☑	☐	

我们引入控制对象的概念，示例中每一个电机就是一个控制对象，每个电机控制对象包括了4个位输入，2个位输出，1个背景数据块，通过控制对象硬件IO表（UDT）将每个控制对象的硬件IO关联到一起。可以有以下两种定义方式：

图6所示直接根据控制对象硬件IO的名称定义控制对象硬件IO表，这种方式后续再封装程序的可读性强一些，与控制功能块的输入输出管脚定义是一致的，即控制对象硬件IO表与控制功能块输入输出是明确**指定关联关系**的。然而一个控制系统的控制对象往往是多样化的，其硬件IO的数量和类型也会不同，这样做的结果是不同的控制对象，就要定义一个对应的控制对象硬件IO表，导致编程过于繁琐，另外也不利于控制功能块多次调用后实现对硬件IO表的访问的编程；

图7所示采用数组形式定义控制对象硬件IO表，只要数组变量的数量满足需求，可以事先不用确定数组变量关联的硬件IO其类型究竟是输入还是输出，满足程序标准化柔性化的要求。

博图环境下通过用户程序实现硬件IO自由组态的基本方法

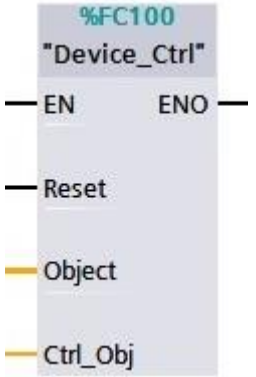
SIEMENS
Ingenuity for life

UPD_IO_Sample ▸ PLC_1 [CPU 1215C DC/DC/DC] ▸ Program blocks ▸ Device_Ctrl [FC100]

Device_Ctrl

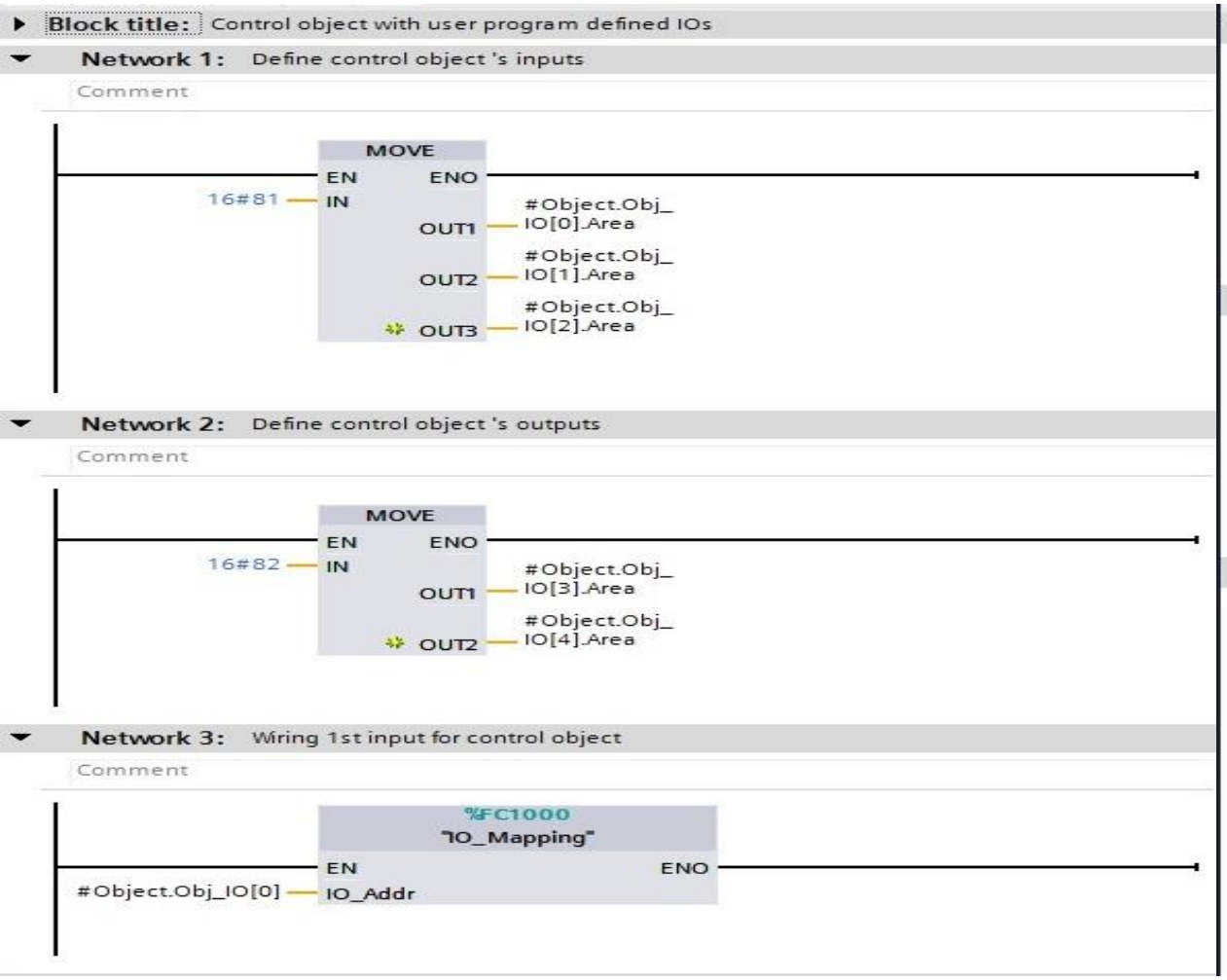
	Name	Data type	Default value	Comment
1	Input			
2	Reset	Bool		
3	Output			
4	<Add new>			
5	InOut			
6	Object	"Obj_IO_UDT"		User program defined IOs for control object
7	Obj_IO	Array[0..5] of "IO_A...		
8	Obj_IO[0]	"IO_Addr_UDT"		
9	Obj_IO[1]	"IO_Addr_UDT"		
10	Obj_IO[2]	"IO_Addr_UDT"		
11	Obj_IO[3]	"IO_Addr_UDT"		
12	Obj_IO[4]	"IO_Addr_UDT"		
13	Obj_IO[5]	"IO_Addr_UDT"		
14	Ctrl_Obj	"Dev_CtrlFB"		Control object itself
15	Input			
16	Start	Bool		Start command
17	Stop	Bool		Stop command
18	Fault	Bool		Device fault signal (fault=0)
19	Reset	Bool		Reset command
20	Output			
21	Dev_Out	Bool		Device Output
22	DeV_Err	Bool		Device fault output
23	InOut			
24	Static			
25	Out_Flg	Bool		Flag for device output
26	Err_Flg	Bool		Flag for device fault output
27	Temp			
28	<Add new>			
29	Constant			
30	<Add new>			
31	Return			
32	Device_Ctrl	Void		

图9～12为基于原控制功能块FB1基础上再进行封装的硬件IO可定义的控制功能FC100，实际上就是通过IO变址映射功能FC1000将一个未指定的控制对象硬件IO表与原控制程序FB1进行了关联，FC100的每一次调用就代表了一个硬件IO可自由定义的控制对象。



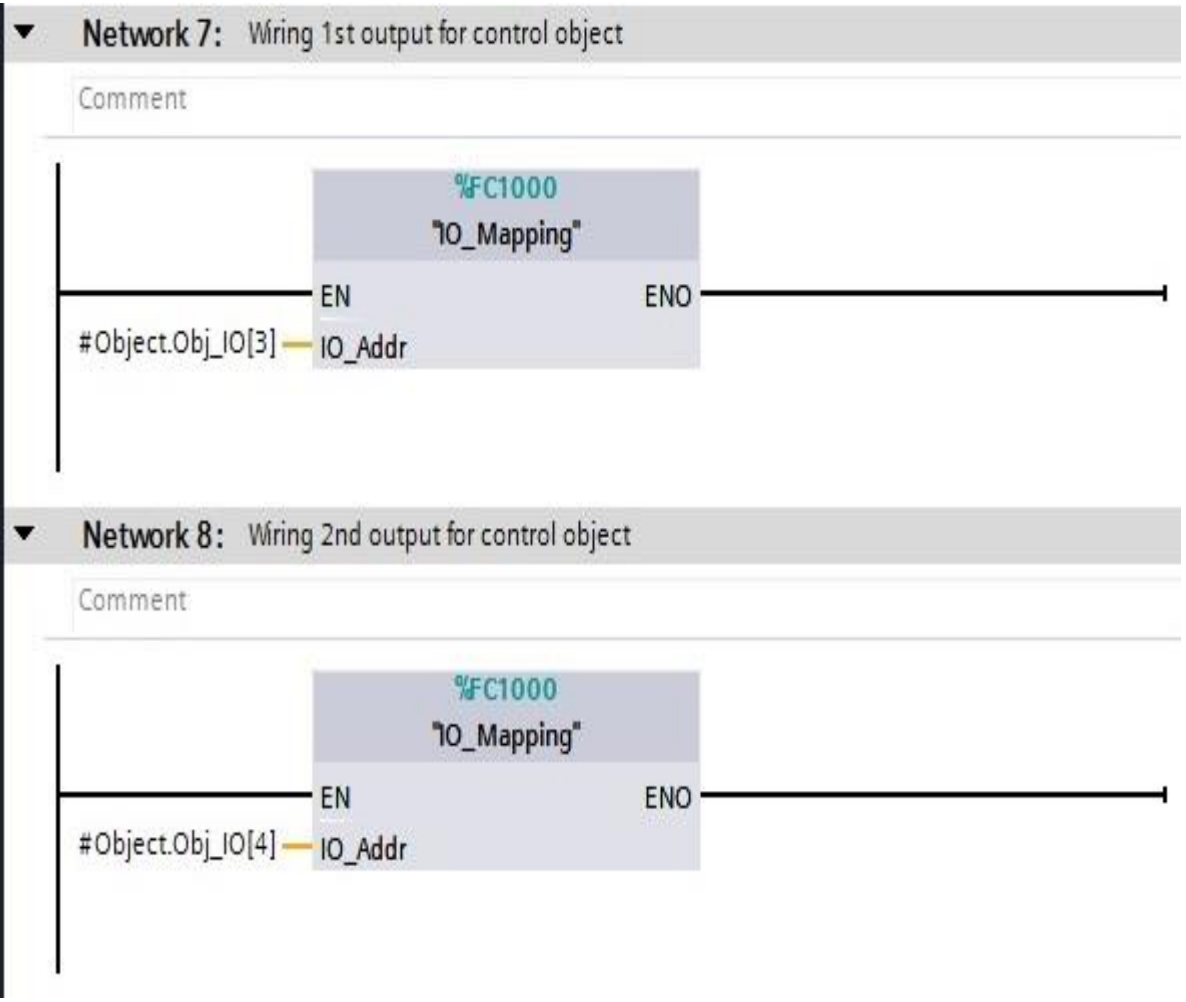
博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life

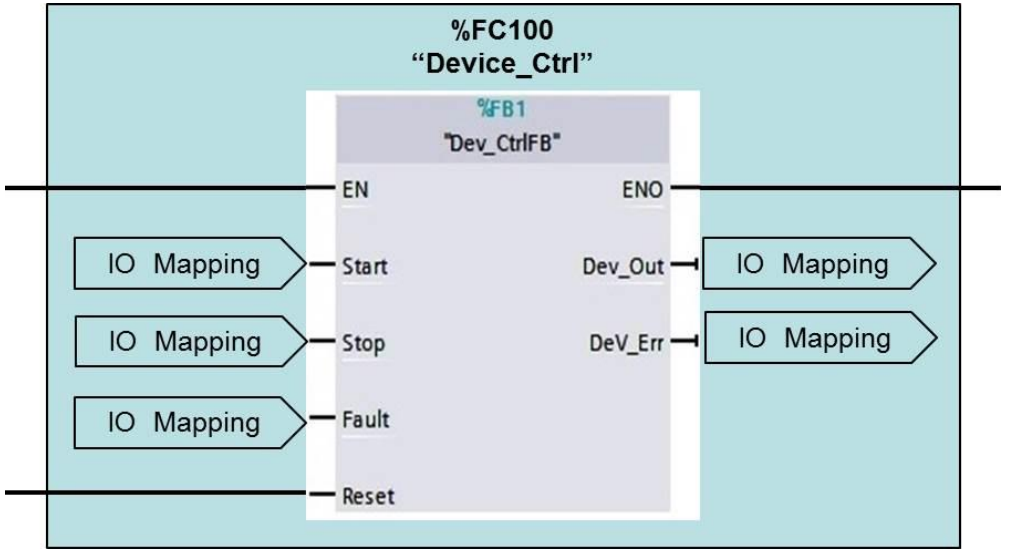


博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life



封装好的硬件IO可定义的控制功能块FC100



当然硬件IO表中的元素是否一定要与原控制程序FB1关联，也是可以自由选择的，本例中考虑多个控制对象的故障复位命令信号可能会共用同一个变量，因此FB1的Reset参数就未与控制对象硬件IO定义表关联，而是单独直连FC100的同名输入变量了。

博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life

UPD_IO_Sample ▸ PLC_1 [CPU 1215C DC/DC/DC] ▸ Program blocks ▸ Ctrl_Object [FB100]

Ctrl_Object

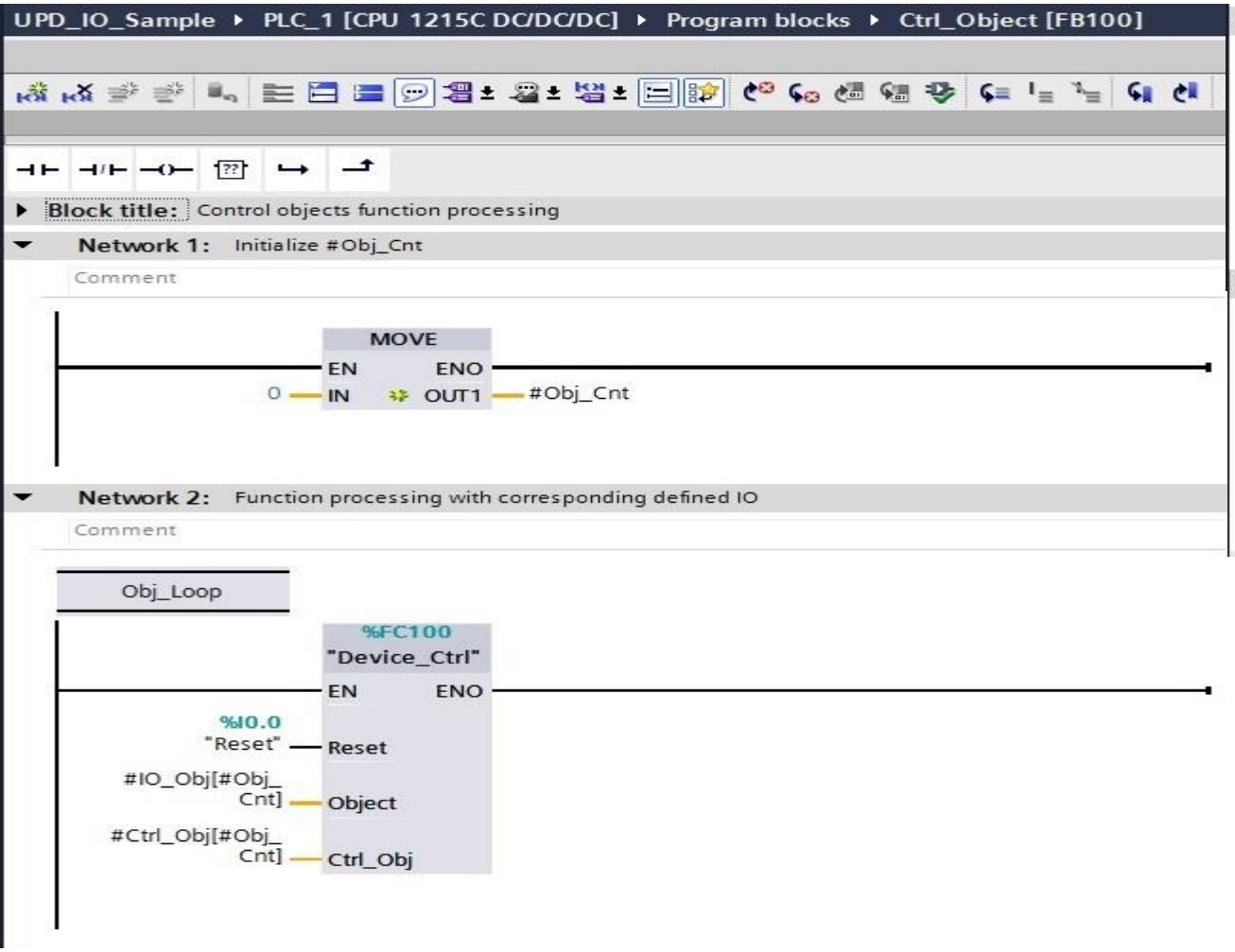
	Name	Data type	Default value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint	Comment
1	Input								
2	<Add new>								
3	Output								
4	<Add new>								
5	InOut								
6	<Add new>								
7	Static								
8	Obj_No_Set	Int	9	Set in IDB	☑	☑	☑	☐	Number set of control objects
9	IO_Obj	Array[0..9] of "Obj_IO_UDT"		Set in IDB	☑	☑	☑	☐	Control objects with defined IO
10	IO_Obj[0]	"Obj_IO_UDT"		Set in IDB	☑	☑	☑	☐	Control objects with defined IO
11	IO_Obj[1]	"Obj_IO_UDT"		Set in IDB	☑	☑	☑	☐	Control objects with defined IO
12	IO_Obj[2]	"Obj_IO_UDT"		Set in IDB	☑	☑	☑	☐	Control objects with defined IO
13	IO_Obj[3]	"Obj_IO_UDT"		Set in IDB	☑	☑	☑	☐	Control objects with defined IO
14	IO_Obj[4]	"Obj_IO_UDT"		Set in IDB	☑	☑	☑	☐	Control objects with defined IO
15	IO_Obj[5]	"Obj_IO_UDT"		Set in IDB	☑	☑	☑	☐	Control objects with defined IO
16	IO_Obj[6]	"Obj_IO_UDT"		Set in IDB	☑	☑	☑	☐	Control objects with defined IO
17	IO_Obj[7]	"Obj_IO_UDT"		Set in IDB	☑	☑	☑	☐	Control objects with defined IO
18	IO_Obj[8]	"Obj_IO_UDT"		Set in IDB	☑	☑	☑	☐	Control objects with defined IO
19	IO_Obj[9]	"Obj_IO_UDT"		Set in IDB	☑	☑	☑	☐	Control objects with defined IO
20	Ctrl_Obj	Array[0..9] of "Dev_CtrlFB"			☐	☐	☐	☐	Function of control objects
21	Ctrl_Obj[0]	"Dev_CtrlFB"			☐	☐	☐	☐	Function of control objects
22	Ctrl_Obj[1]	"Dev_CtrlFB"			☐	☐	☐	☐	Function of control objects
23	Ctrl_Obj[2]	"Dev_CtrlFB"			☐	☐	☐	☐	Function of control objects
24	Ctrl_Obj[3]	"Dev_CtrlFB"			☐	☐	☐	☐	Function of control objects
25	Ctrl_Obj[4]	"Dev_CtrlFB"			☐	☐	☐	☐	Function of control objects
26	Ctrl_Obj[5]	"Dev_CtrlFB"			☐	☐	☐	☐	Function of control objects
27	Ctrl_Obj[6]	"Dev_CtrlFB"			☐	☐	☐	☐	Function of control objects
28	Ctrl_Obj[7]	"Dev_CtrlFB"			☐	☐	☐	☐	Function of control objects
29	Ctrl_Obj[8]	"Dev_CtrlFB"			☐	☐	☐	☐	Function of control objects
30	Ctrl_Obj[9]	"Dev_CtrlFB"			☐	☐	☐	☐	Function of control objects
31	Temp								
32	Obj_Cnt	Int			☐	☐	☐	☐	Loop counter for control objects functio...
33	Constant								
34	<Add new>								

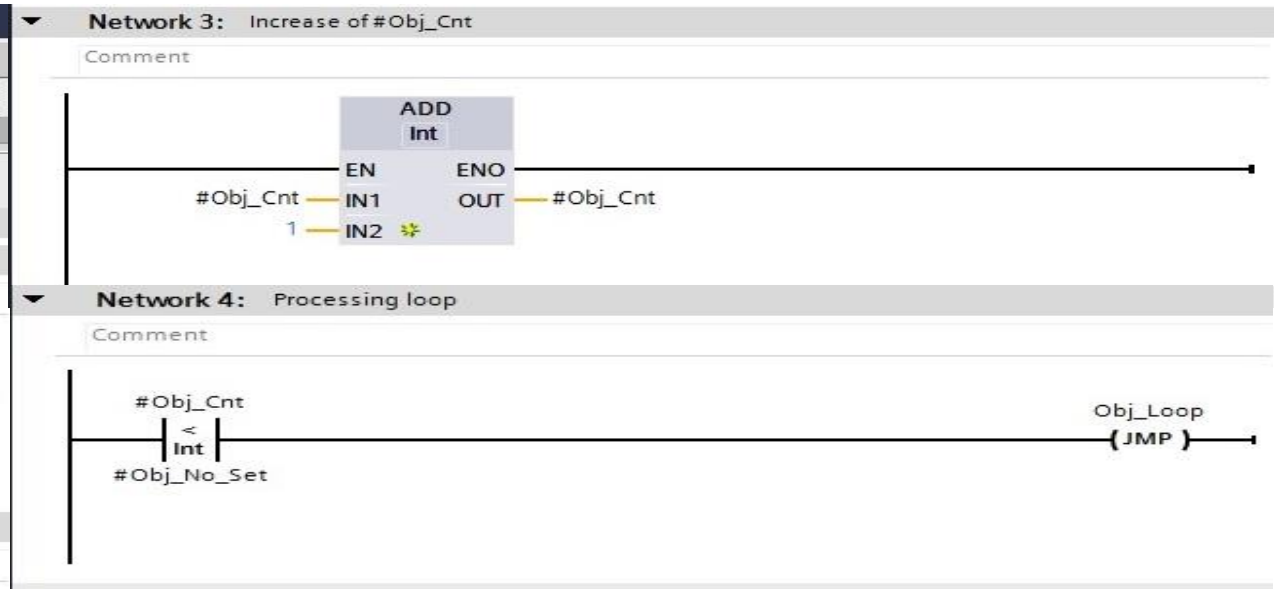
图13～14是一个具有10个控制对象的小型MCC控制功能块FB100的完整示例程序，在保留原有传统控制功能块FB1的控制功能的基础上，又实现了任意一个控制对象的硬件IO点可自由定义的功能，且硬件IO点的关联与激活亦可由用户灵活决定。

由于底层的硬件IO地址说明表采用了结构数据，中间层控制对象硬件IO表及顶层控制对象功能调用均采用了数组结构形式，因此整体控制程序结构变得极为简单，循环调用即可，数组结构也使得每个控制对象关联的硬件IO地址说明表很容易通过间址寻址的方式实现读写访问，甚至是不同的控制对象且每个控制对象关联着不同数量的硬件IO点。

博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life





由此可见，在博图环境下通过上述的编程思路和方法，硬件IO可定义MCC控制功能块的程序代码得到了极大的精简，且控制对象的数量多少以及每个控制对象关联硬件IO点的多少均与程序代码量无关。

博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life

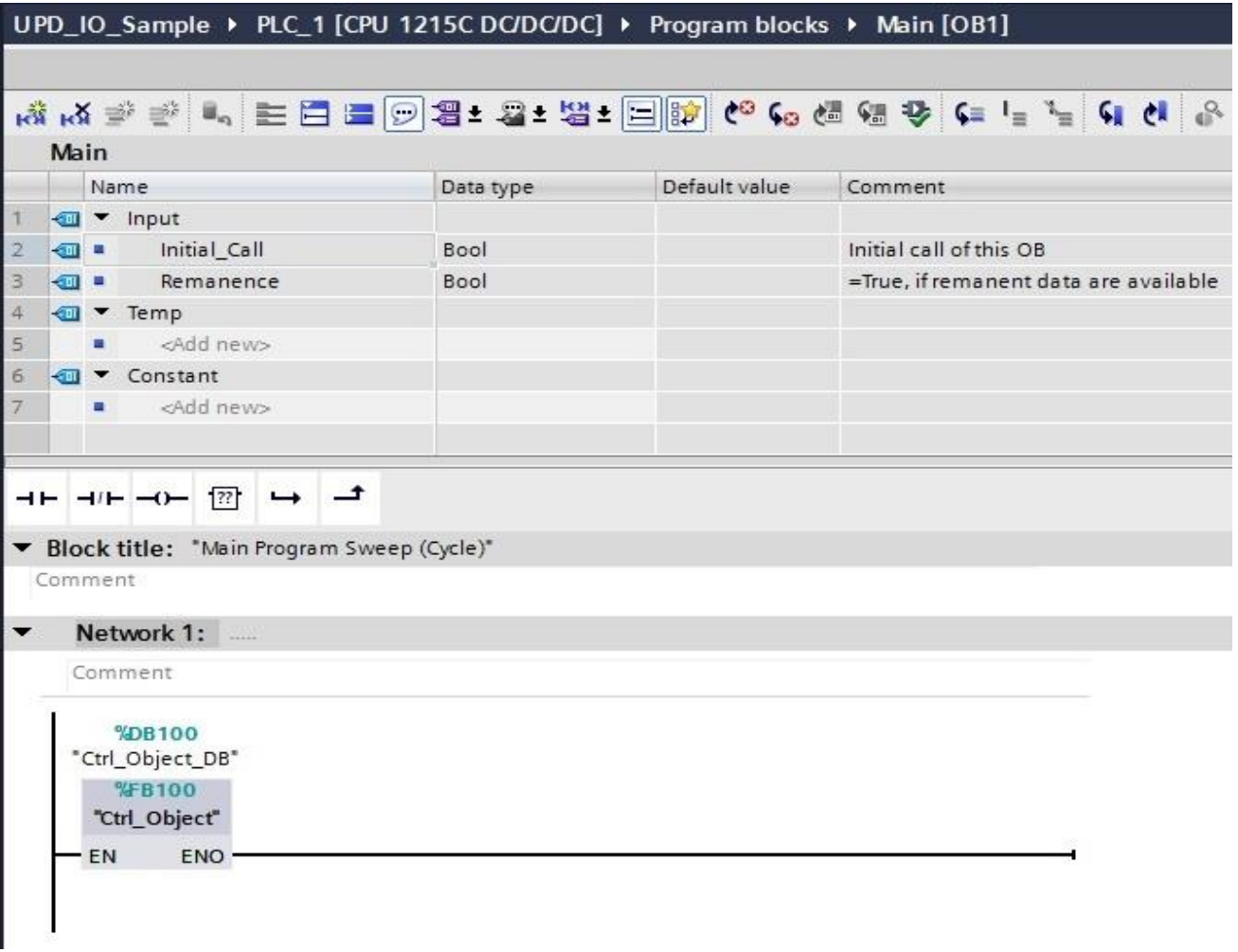


图15为硬件IO可定义MCC控制功能块在OB1的调用，我们可以通过变量修改的方式修改DB100中IO_Obj数组变量关联的硬件IO点，示例中激活了3个控制对象关联的硬件IO点，分别位于S7-1200不同的IO扩展模块上，地址定义如图16～18所示：

Motor_0	Start = I8.0	Dev_Out = Q8.0
	Stop = I8.1	Dev_Err = Q8.1
	Fault = I8.2	
Motor_1	Start = I12.0	Dev_Out = Q12.0
	Stop = I12.1	Dev_Err = Q12.1
	Fault = I12.2	
Motor_2	Start = I16.0	Dev_Out = Q16.0
	Stop = I16.1	Dev_Err = Q16.0
	Fault = I16.2	

博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life

UPD_IO_Sample ▸ PLC_1 [CPU 1215C DC/DC/DC] ▸ Program blocks ▸ Ctrl_Object_DB [DB100]

Keep actual values | Snapshot | Copy snapshots to start values | Load start va

Ctrl_Object_DB

	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...
1	Input						
2	Output						
3	InOut						
4	Static						
5	Obj_No_Set	Int	9				
6	IO_Obj	Array[0..9] of "Obj_IO_UDT"					
7	IO_Obj[0]	"Obj_IO_UDT"					
8	Obj_IO	Array[0..5] of "IO_Addr_UDT"					
9	Obj_IO[0]	"IO_Addr_UDT"					
10	Area	Byte	16#0				
11	Byte_Offset	DInt	8				
12	Bit_Offset	Int	0				
13	Value	Bool	false				
14	Status	Byte	16#0				
15	Obj_IO[1]	"IO_Addr_UDT"					
16	Area	Byte	16#0				
17	Byte_Offset	DInt	8				
18	Bit_Offset	Int	1				
19	Value	Bool	false				
20	Status	Byte	16#0				
21	Obj_IO[2]	"IO_Addr_UDT"					
22	Area	Byte	16#0				
23	Byte_Offset	DInt	8				
24	Bit_Offset	Int	2				
25	Value	Bool	false				
26	Status	Byte	16#0				
27	Obj_IO[3]	"IO_Addr_UDT"					
28	Area	Byte	16#0				
29	Byte_Offset	DInt	8				
30	Bit_Offset	Int	0				
31	Value	Bool	false				
32	Status	Byte	16#0				
33	Obj_IO[4]	"IO_Addr_UDT"					
34	Area	Byte	16#0				
35	Byte_Offset	DInt	8				
36	Bit_Offset	Int	1				
37	Value	Bool	false				
38	Status	Byte	16#0				
39	Obj_IO[5]	"IO_Addr_UDT"					

Motor_0 Start I8.0
Motor_0 Stop I8.1
Motor_0 Fault I8.2
Motor_0 Dev_Out Q8.0
Motor_0 Dev_Err Q8.1

UPD_IO_Sample ▸ PLC_1 [CPU 1215C DC/DC/DC] ▸ Program blocks ▸ Ctrl_Object_DB [DB100]

Keep actual values | Snapshot | Copy snapshots to start values | Load start values

Ctrl_Object_DB

	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setp
1	Input							
2	Output							
3	InOut							
4	Static							
5	Obj_No_Set	Int	9					
6	IO_Obj	Array[0..9] of "Obj_IO_UDT"						
7	IO_Obj[0]	"Obj_IO_UDT"						
8	IO_Obj[1]	"Obj_IO_UDT"						
9	IO_Obj[2]	"Obj_IO_UDT"						
10	Obj_IO	Array[0..5] of "IO_Addr_UDT"						
11	Obj_IO[0]	"IO_Addr_UDT"						
12	Area	Byte	16#0					
13	Byte_Offset	DInt	16					
14	Bit_Offset	Int	0					
15	Value	Bool	false					
16	Status	Byte	16#0					
17	Obj_IO[1]	"IO_Addr_UDT"						
18	Area	Byte	16#0					
19	Byte_Offset	DInt	16					
20	Bit_Offset	Int	1					
21	Value	Bool	false					
22	Status	Byte	16#0					
23	Obj_IO[2]	"IO_Addr_UDT"						
24	Area	Byte	16#0					
25	Byte_Offset	DInt	16					
26	Bit_Offset	Int	2					
27	Value	Bool	false					
28	Status	Byte	16#0					
29	Obj_IO[3]	"IO_Addr_UDT"						
30	Area	Byte	16#0					
31	Byte_Offset	DInt	16					
32	Bit_Offset	Int	0					
33	Value	Bool	false					
34	Status	Byte	16#0					
35	Obj_IO[4]	"IO_Addr_UDT"						
36	Area	Byte	16#0					
37	Byte_Offset	DInt	16					
38	Bit_Offset	Int	1					
39	Value	Bool	false					
40	Status	Byte	16#0					
41	Obj_IO[5]	"IO_Addr_UDT"						

Motor_3 Start I16.0
Motor_3 Stop I16.1
Motor_3 Fault I16.2
Motor_3 Dev_out Q16.0
Motor_3 Dev_Err Q16.1

UPD_IO_Sample ▸ PLC_1 [CPU 1215C DC/DC/DC] ▸ Program blocks ▸ Ctrl_Object_DB [DB100]

Keep actual values | Snapshot | Copy snapshots to start values | Load start values as actual values

Ctrl_Object_DB

	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint	Comment
1	Input								
2	Output								
3	InOut								
4	Static								
5	Obj_No_Set	Int	9						Number set of control objects
6	IO_Obj	Array[0..9] of "Obj_IO_UDT"							Control objects with defined IO
7	IO_Obj[0]	"Obj_IO_UDT"							Control objects with defined IO
8	IO_Obj[1]	"Obj_IO_UDT"							Control objects with defined IO
9	Obj_IO	Array[0..5] of "IO_Addr_UDT"							
10	Obj_IO[0]	"IO_Addr_UDT"							
11	Area	Byte	16#0						IO can be defined as Input=16#81 output=16#82
12	Byte_Offset	DInt	12						Address to read from or to be written, Only the 16 least significant bits are used.
13	Bit_Offset	Int	0						Bit to be read from or to be written
14	Value	Bool	false						IO value, to be written for input, to be read for output
15	Status	Byte	16#0						IO status Deactive=16#00 Active=16#01 Fault=16#FF
16	Obj_IO[1]	"IO_Addr_UDT"							
17	Area	Byte	16#0						IO can be defined as Input=16#81 output=16#82
18	Byte_Offset	DInt	12						Address to read from or to be written, Only the 16 least significant bits are used.
19	Bit_Offset	Int	1						Bit to be read from or to be written
20	Value	Bool	false						IO value, to be written for input, to be read for output
21	Status	Byte	16#0						IO status Deactive=16#00 Active=16#01 Fault=16#FF
22	Obj_IO[2]	"IO_Addr_UDT"							
23	Area	Byte	16#0						IO can be defined as Input=16#81 output=16#82
24	Byte_Offset	DInt	12						Address to read from or to be written, Only the 16 least significant bits are used.
25	Bit_Offset	Int	2						Bit to be read from or to be written
26	Value	Bool	false						IO value, to be written for input, to be read for output
27	Status	Byte	16#0						IO status Deactive=16#00 Active=16#01 Fault=16#FF
28	Obj_IO[3]	"IO_Addr_UDT"							
29	Area	Byte	16#0						IO can be defined as Input=16#81 output=16#82
30	Byte_Offset	DInt	12						Address to read from or to be written, Only the 16 least significant bits are used.
31	Bit_Offset	Int	0						Bit to be read from or to be written
32	Value	Bool	false						IO value, to be written for input, to be read for output
33	Status	Byte	16#0						IO status Deactive=16#00 Active=16#01 Fault=16#FF
34	Obj_IO[4]	"IO_Addr_UDT"							
35	Area	Byte	16#0						IO can be defined as Input=16#81 output=16#82
36	Byte_Offset	DInt	12						Address to read from or to be written, Only the 16 least significant bits are used.
37	Bit_Offset	Int	1						Bit to be read from or to be written
38	Value	Bool	false						IO value, to be written for input, to be read for output
39	Status	Byte	16#0						IO status Deactive=16#00 Active=16#01 Fault=16#FF
40	Obj_IO[5]	"IO_Addr_UDT"							
41	IO_Obj[2]	"Obj_IO_UDT"							Control objects with defined IO

Motor_1 Start I12.0
Motor_1 Stop I12.1
Motor_1 Fault I12.2
Motor_1 Dev_Out Q12.0
Motor_1 Dev_Err Q12.1

博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life

Siemens - D:\Projects\UPD_IO_Sample_V15\UPD_IO_Sample_V15

Project Edit View Insert Online Options Tools Window Help

Project tree

Devices

UPD_IO_Sample_V15 ▸ PLC_1 [CPU 1215C DC/DC/DC] ▸ PLC tags ▸ UPD_IO [16]

UPD_IO

	Name	Data type	Address	Retain	Acces...	Writa...	Visibl...	Monitor value	Comme
1	Motor_0_Start	Bool	%I8.0					FALSE	
2	Motor_0_Stop	Bool	%I8.1					TRUE	
3	Motor_0_Fault	Bool	%I8.2					TRUE	
4	Motor_0_Out	Bool	%Q8.0					TRUE	
5	Motor_0_Err	Bool	%Q8.1					FALSE	
6									
7	Motor_1_Start	Bool	%I12.0					FALSE	
8	Motor_1_Stop	Bool	%I12.1					TRUE	
9	Motor_1_Fault	Bool	%I12.2					FALSE	
10	Motor_1_Out	Bool	%Q12.0					FALSE	
11	Motor_1_Err	Bool	%Q12.1					TRUE	
12									
13	Motor_2_Start	Bool	%I16.0					FALSE	
14	Motor_2_Stop	Bool	%I16.1					TRUE	
15	Motor_2_Fault	Bool	%I16.2					TRUE	
16	Motor_2_Out	Bool	%Q16.0					FALSE	
17	Motor_2_Err	Bool	%Q16.1					FALSE	
18									
19	Reset	Bool	%I0.0					FALSE	
20	<Add new>								

Siemens
PLC_1 [CPU 1215C DCDCDC]
SIEMENS
RUN/STOP
ERROR
MAINT
RUN
STOP
MRES
X1 192.168.0.1
<无项目>

Dev_CtrlFB [FB1]
Ctrl_Object_DB [DB100]
Dev_CtrlFB_DB1 [DB1]
Dev_CtrlFB_DB2 [DB2]
Technology objects
External source files
PLC tags

Details view

Name	Offset	Data type
Obj_No_Set		Int
IO_Obj		Array[0..9] ...
Ctrl_Obj		Array[0..9] ...

博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life

三、应用体会：

1. 博图的创新平台，使得通过用户程序实现硬件IO自由组态成为可能，可保留原有传统的成熟应用的控制功能块，无需对原有功能块进行改造，不需要复杂的编程，在梯形图下即可很容易地实现（示例中的硬件IO变址映射功能FC1000亦可在梯形图环境下编程，只是在使用PEEK/POKE指令时需插入SCL语言程序段），并且实现S7-1200/1500通用化。
2. 示例中是针对一个功能块FB来实现其输入输出参数硬件IO地址用户程序可组态的功能的，然而在实际编程中发现FB的多重背景应用是可以数组化的（含V14版本以上），但参数实例只能是单个的数据块定义而无法数组化定义的，这给编程的灵活性还是带来了一定的影响，因此多控制对象应用必须通过FB的多重背景才能实现，期望博图软件在这方面能够更有进一步的创新，以方便用户程序开发。

   反之，这也让我们看到了另一种可能性，即用FC加全局DB替代FB（这种方式编程在S7-300/400/1200/1500的软件开发中不常见），利用全局数据比较容易实现数组化的特点，同样在原有程序基础上实现硬件IO地址用户程序可组态，也许这样编程方法会更加的灵活简单。
3. 有一种情况我在前面应用背景的描述中没有提及，即IO点如果损毁，那么我们不用修改程序就可以替换IO点了。的确是可以这么做，而且很容易实现，这是硬件IO用户程序可组态的编程方法实现带来的“副作用”，当然这是有益的“副作用”，但是当初的确也没有以“损毁IO无编程替换”为目的来做这项软件开发的，没有引入控制对象的概念，如果这么做反而是有难度的了。

博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life

4. 本论文讲述了在博图环境下通过用户程序实现硬件IO自由组态的基本方法，还是原理性质的，其真正要应用到实际的项目中去，仍旧有很多的实际的技术问题需要去解决的，比如：

1) 在尝试使用精简型面板为上述系统开发一个简易的硬件IO地址参数设置界面时，发现在优化数据块访问即全符号名访问的设置下，HMI还是只能实现一维的数组变量的变址访问（图21所示），其结果如图20所示，IO地址参数设置界面的形式就只能与具体的控制对象绑定了；因此在目前博图的技术能力下，要实现“去控制对象化”的通用标准化的硬件参数设置界面功能，还是需要通过PLC及HMI两者紧密的编程配合来实现。

同时，面对数量众多的硬件IO点的地址表，通过HMI逐一手工输入的方式并不可取，费时费力枯燥易错，HMI只能适合少量人工输入及修改；因此，还是要有更简便高效的方案来实现（比如通过计算机EXCEL文件编辑硬件IO地址表，把它导出为格式文件，然后拷贝到PLC的MMC卡，再由PLC程序读取格式文件的数据到内部数据块的方式）。

UPD_IO_Sample ▸ HMI_1 [KTP700 Basic PN] ▸ HMI tags

HMI tags

Name	Tag table	Data type	Connection	PLC name	PLC tag	Address	Access mode	Acquisiti...	Source comment
Area_Err	Default tag table	Byte	HMI_Connection_1	PLC_1	<Multiplex ta...	Ctrl_Object_DB.IO_Obj[Object].Obj_IO[4].Area	<symbolic access>	1 s	IO can be defined as Input=16#81 output=16#82
Area_Fault	Default tag table	Byte	HMI_Connection_1	PLC_1	<Multiplex ta...	Ctrl_Object_DB.IO_Obj[Object].Obj_IO[2].Area	<symbolic access>	1 s	IO can be defined as Input=16#81 output=16#82
Area_Out	Default tag table	Byte	HMI_Connection_1	PLC_1	<Multiplex ta...	Ctrl_Object_DB.IO_Obj[Object].Obj_IO[3].Area	<symbolic access>	1 s	IO can be defined as Input=16#81 output=16#82
Area_Start	Default tag table	Byte	HMI_Connectio...	PLC_1	<Multiple...	Ctrl_Object_DB.IO_Obj[Object].Obj_IO[0].Area	<symbolic access>	1 s	IO can be defined as Input=16#81 output=16#82
Area_Stop	Default tag table	Byte	HMI_Connection_1	PLC_1	<Multiplex ta...	Ctrl_Object_DB.IO_Obj[Object].Obj_IO[1].Area	<symbolic access>	1 s	IO can be defined as Input=16#81 output=16#82
Bit_Addr_Err	Default tag table	Int	HMI_Connection_1	PLC_1	<Multiplex ta...	Ctrl_Object_DB.IO_Obj[Object].Obj_IO[4].Bit_Offset	<symbolic access>	1 s	Bit to be read from or to be written
Bit_Addr_Fault	Default tag table	Int	HMI_Connection_1	PLC_1	<Multiplex ta...	Ctrl_Object_DB.IO_Obj[Object].Obj_IO[2].Bit_Offset	<symbolic access>	1 s	Bit to be read from or to be written
Bit_Addr_Out	Default tag table	Int	HMI_Connection_1	PLC_1	<Multiplex ta...	Ctrl_Object_DB.IO_Obj[Object].Obj_IO[3].Bit_Offset	<symbolic access>	1 s	Bit to be read from or to be written
Bit_Addr_Start	Default tag table	Int	HMI_Connection_1	PLC_1	<Multiplex ta...	Ctrl_Object_DB.IO_Obj[Object].Obj_IO[0].Bit_Offset	<symbolic access>	1 s	Bit to be read from or to be written
Bit_Addr_Stop	Default tag table	Int	HMI_Connection_1	PLC_1	<Multiplex ta...	Ctrl_Object_DB.IO_Obj[Object].Obj_IO[1].Bit_Offset	<symbolic access>	1 s	Bit to be read from or to be written

博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life

2) 在实际应用中，PLC硬件的组态是变化的，实际可寻址的IO物理地址范围也是变化的，因此，实现硬件IO地址寻址范围的合法性判断，从而避免程序的寻址错误也是很有必要的。

3) 本文阐述的硬件IO用户程序可组态的编程实现方法，使得程序的硬件IO寻址由显式寻址改为隐式寻址，因此通过程序代码来检查某个硬件IO是否被重复使用的可能性几乎没有，尤其输出类型的硬件IO点是需要严格限制重复访问的（不仅仅是输出多线圈问题，还有多控制对象使用同一输出点的安全性问题），因此硬件IO点地址参数设置查重也是一个很重要的技术点，似乎每设定一个硬件IO点的地址就要比对所有已有的硬件IO地址的方法是不可取的，这个运算量不是一个普通PLC能够胜任的。

4) 在实际应用中控制对象IO定义动态修改所引发的控制系统安全隐患也必须引起足够的重视，并且要有可靠的对策；输入点的再定义会导致控制信号产生额外的信号变化，输出点的再定义会导致被替换输出点的失控以及替换输出点意外输出或切断，因此本实现方法实际并不适合IO定义的动态修改，比较安全的做法是在设备整体停运，主动力电源被有效切断的前提下，IO再定义功能才能被激活使用，并且应配合HMI的权限管理功能。

5) PLC如何永久保存已经设置好的硬件IO点的地址参数，在经历了PLC长时间断电，断电重启，数据区总清复位后，系统如何能够依旧正常地工作。西门子S7系统事实上也已经提供了很好的解决方案，示例中如图16～18所示的背景数据块DB100中Ctrl_Object_DB.IO_Obj的数组结构是存储硬件IO点的地址参数的数据区，我们可以通过WRIT_DBL指令将该数组结构的内容拷贝到CPU具有“仅存储在装载内存中”属性的数据块中，起到永久保存设置参数的作用；等到系统重新上电初始化时，我们通过OB100触发READ_DBL指令，将备份的数据又拷贝到CPU工作内存中的DB100 Ctrl_Object_DB.IO_Obj的数组结构中去，恢复因断电或总清而被清除的设置参数。

博图环境下通过用户程序实现硬件IO自由组态的基本方法

SIEMENS
Ingenuity for life

四、结束语：

博图下通过用户程序实现硬件IO自由组态的基本方法，为各类自动化控制系统实现软硬件层面的模块化，标准化，柔性化设计提供了又一种切实可行的方案，也体现了博图平台与S7-1200/1500系列PLC的卓越性能，相信随着博图平台的不断完善，一定能助力西门子自动化用户实现更多更好的创新应用，创造出更大的价值！

五、例程下载：

UPD_IO_Sample_20180909_1519.zap14

https://yunpan.360.cn/surl_ydip3pLdETk （提取码：27ad）

SIEMENS
Ingenuity for life

谢 谢！