fischertechnik

GROLLMUS

KNOWLEDGE | SOFTWARE | AUTOMATION

Bending machine 24V

Structured programming

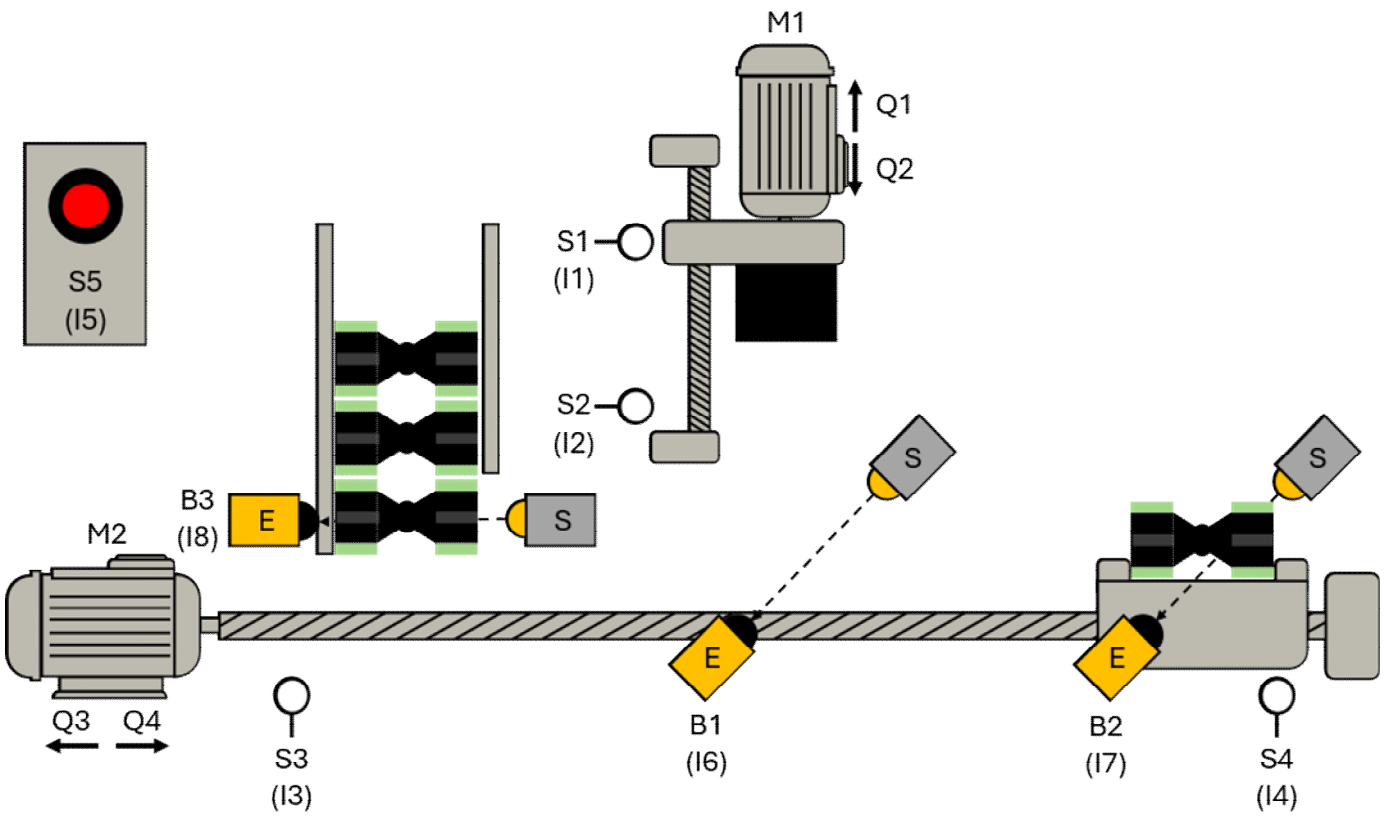


Table of contents

5 Structured programming

5.1 Introduction

Structured programming in PLC systems is used to organize complex programs by dividing them into smaller, clear building blocks. This results in improved code readability, maintainability, and reusability. The user program can be structured according to technological or functional aspects.

In a PLC program, building blocks such as functions (FC) and function modules (FB) are used to structure program parts.

The building blocks should communicate with each other through their building block interfaces, rather than directly accessing global variables. The parameter transfer is carried out via inputs and outputs as well as InOut parameters.

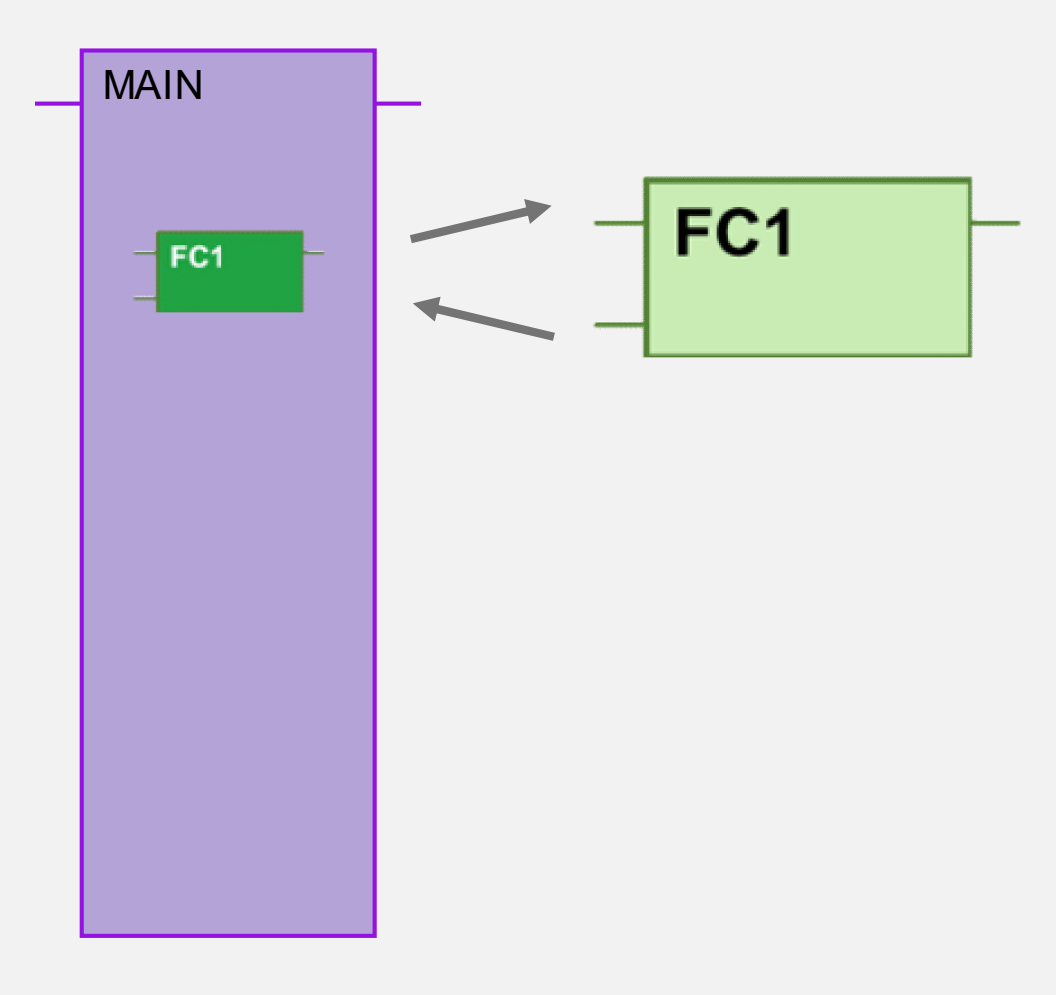
In order to execute the code modules in the control program, they must be called.

5.2 Function

Functions (FCs) are code building blocks without memory. They do not have a data store in which values of building block parameters could be stored. Therefore, all interface parameters must be switched when a function is called. In order to store data permanently, global data modules must first be created.

Functions are ideal for tasks that do not require memory over several cycles, such as mathematical calculations or logical links.

A function contains a program that is executed whenever the function is called by another piece of code.



Pictrue 1 Example: Calling a function from MAIN

A function can also be called several times at different places within a program.

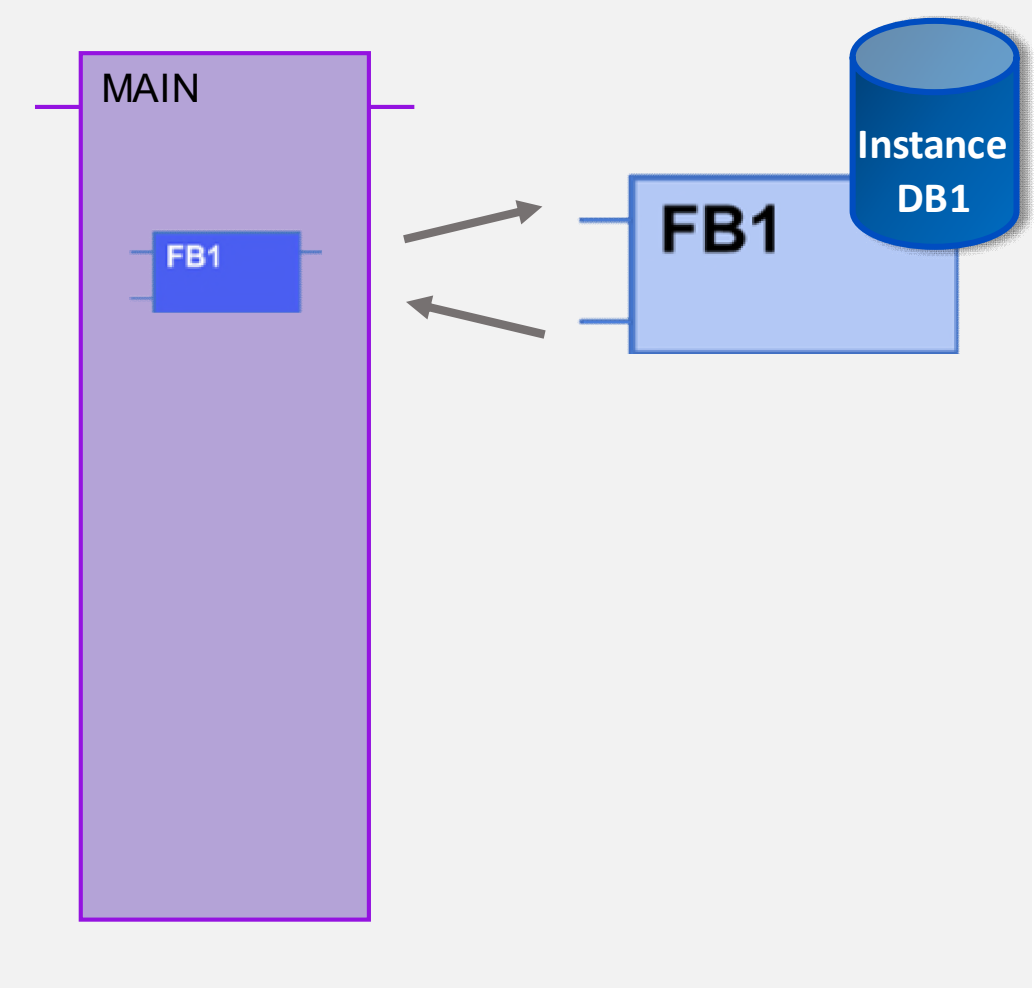
5.3 Function

Function modules (FBs) are code modules that permanently store their input variables, output variables, pass-through variables and also the static variables in instance data modules so that they are also available after the module has been edited. That's why they are also called building blocks with memory.

Function modules are used for tasks that cannot be realized with functions:

- Whenever times and counters are needed in the building blocks or
- if information needs to be stored in the program (e.g. state of the step chain).

Function modules are always executed when a function module is called by another code module.



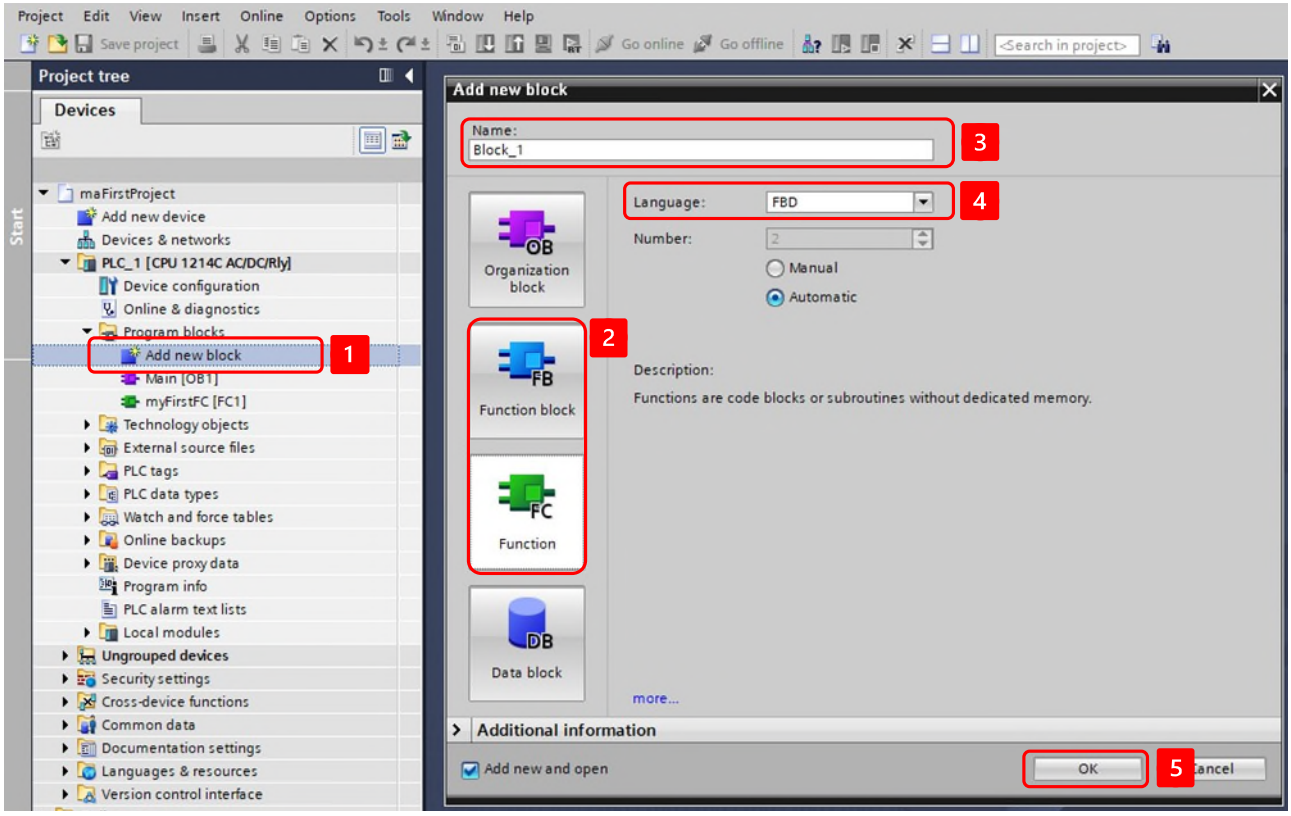
Picture 2 Example: Calling a function module from MAIN

A function module can also be called several times at different places within a program.

A call to a function module is called an instance. Each instance of a function block is assigned a memory area that contains the data that the function module works with.

5.4 Add a new building block

In the TIA Portal, the modules are managed in the project navigation below the PLC in the "Program Modules" folder.
Double-clicking on the "Add New Block" command within the "Program Blocks" folder opens the "Add New Block" dialog, which can be used to create a new block.



Picture 3 Adding a new building block

Here you have to select the module type (2), name (3), and the desired programming language (4).

5.5 Call for modules

In order to execute the code modules in the control program, they must be called. The code module responsible for cyclic program processing is usually referred to as "MAIN". This is started by the operating system and forms the interface to the operating system. The CPU processes the program code that is located in the "MAIN". Within the "MAIN", the program parts structured in functions and function modules can be called.

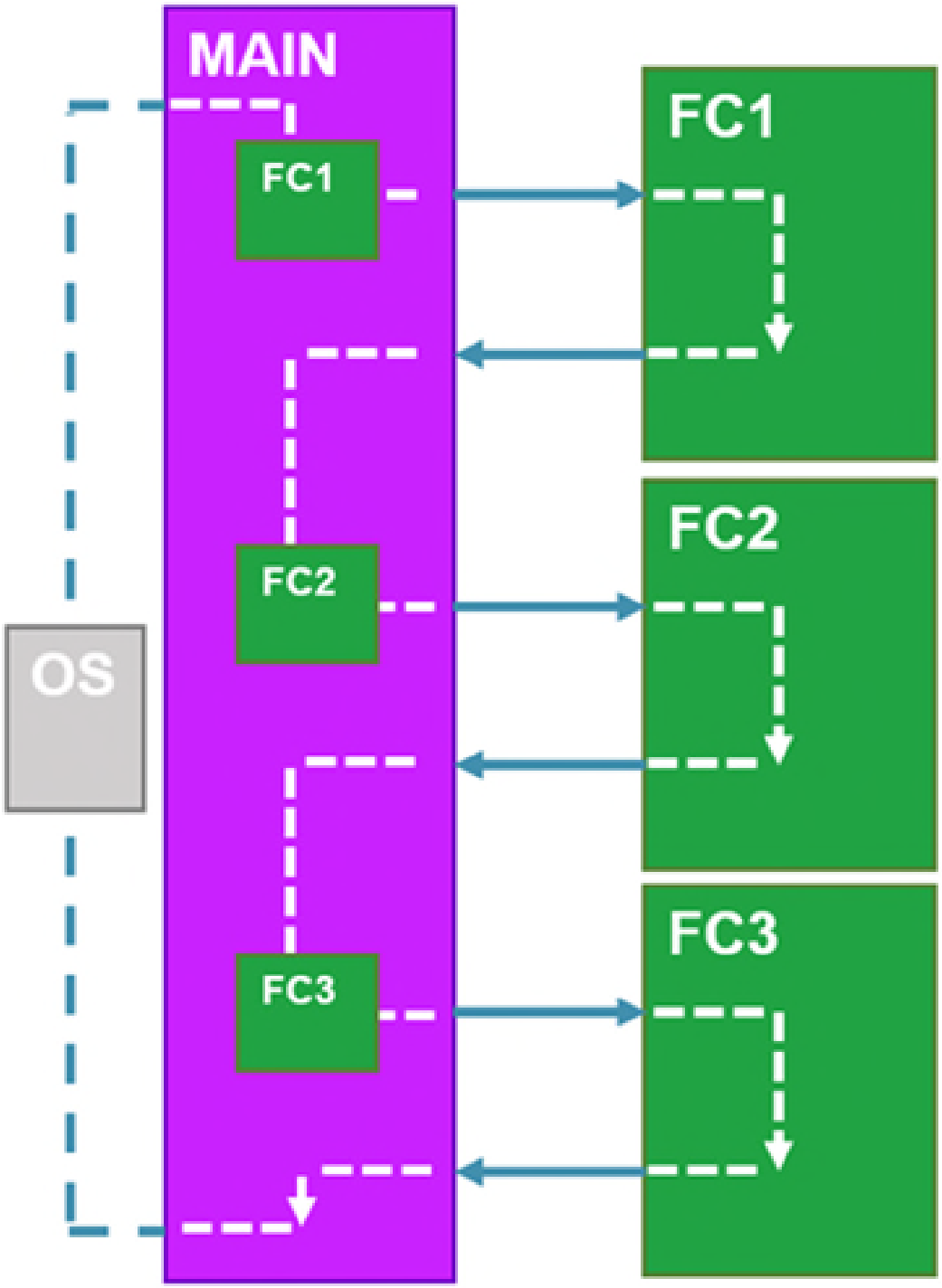
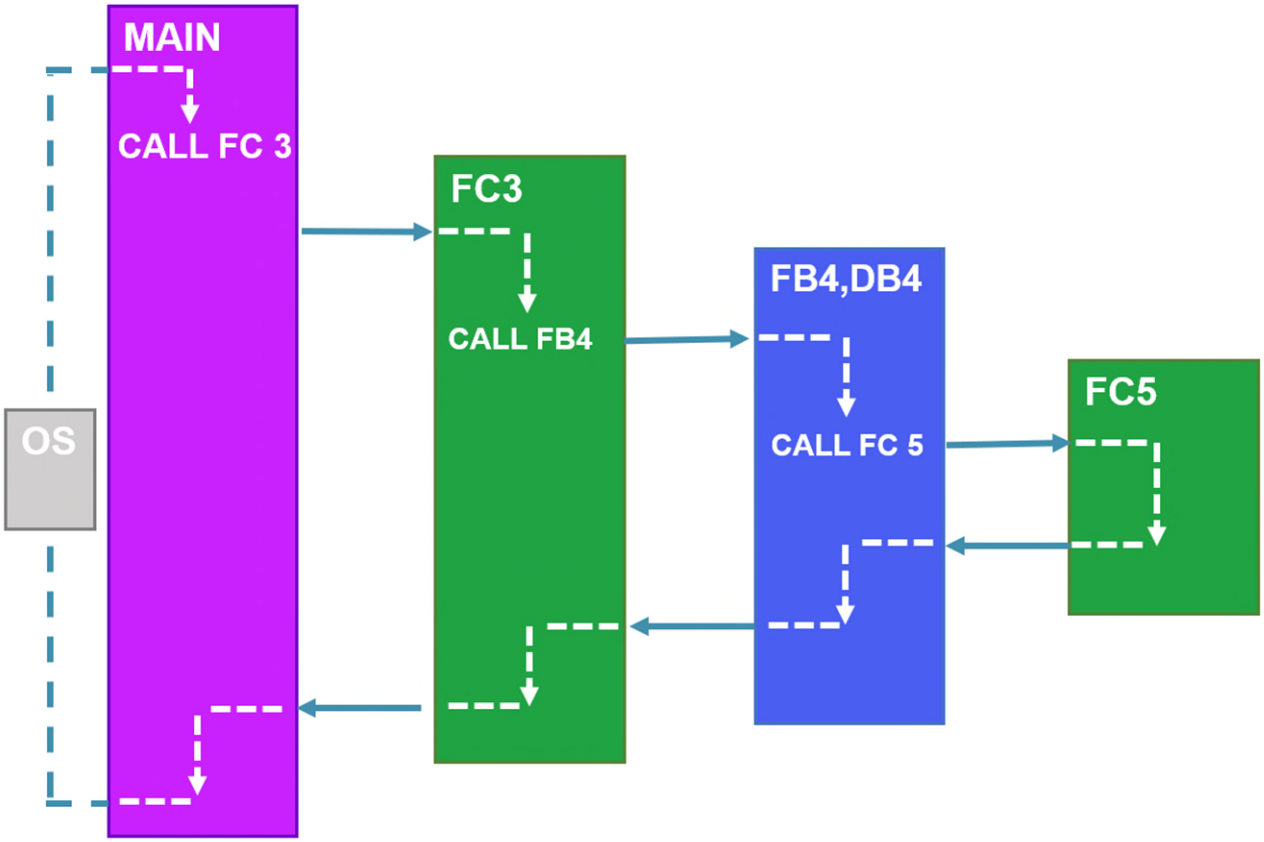


Figure 4 Call to the building block in the MAIN

Functions and function modules structure the program, making it more readable and maintainable.
All called modules are processed one after the other.

The CPU's operating system calls the " MAIN " again after the program cycle, executing all the commands programmed in it again.

A module can be processed by calling it from the "MAIN", for example. Alternatively, it can also be called from a FB or FC, which in turn are called in the "MAIN".

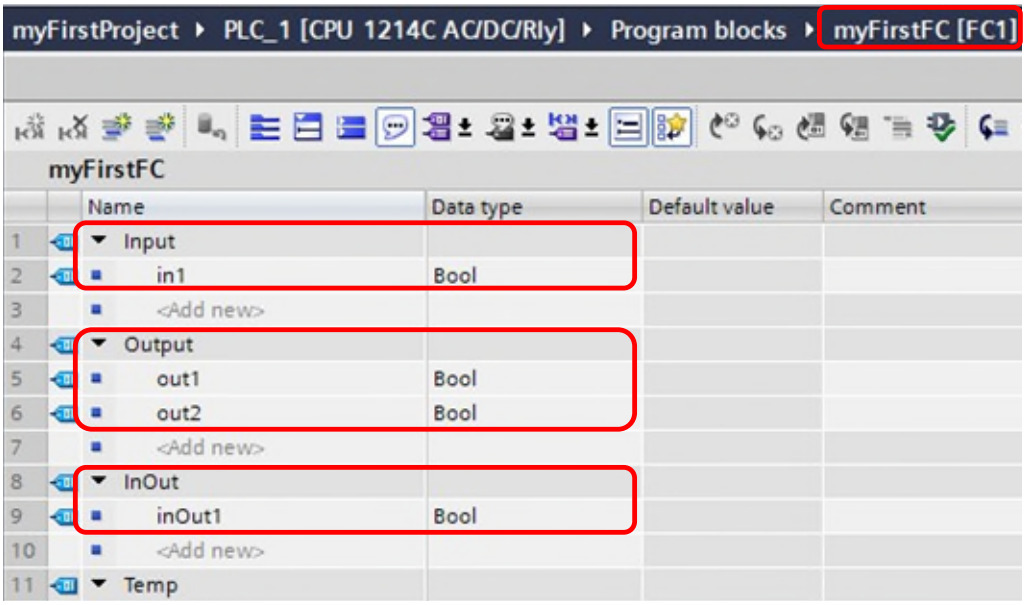


Picture 5 Call to the building blocks

5.6 Parameter transfer

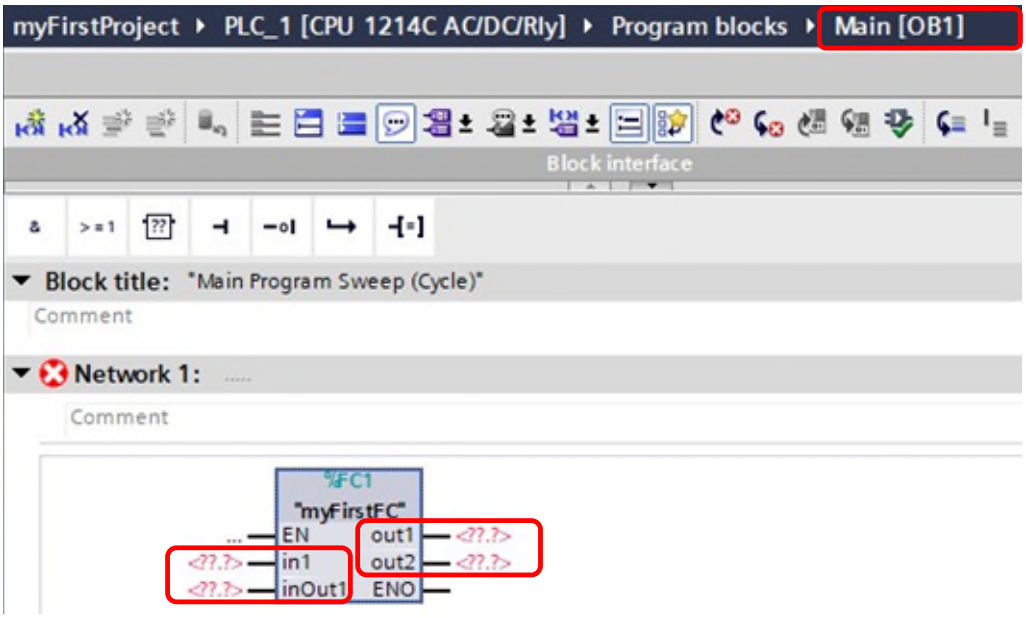
When calling functions and function modules, parameters (variables and variable values) can be passed. The program code within the code module then works with the passed values of the formal parameters and can return results via initial parameters or the function value.

The data exchange takes place via the module interface. These are declared in the module interface of the function (FC) or the function module (FB) and can be used locally in the module.



Picture 6 Formal Parameters in the Brick Interface

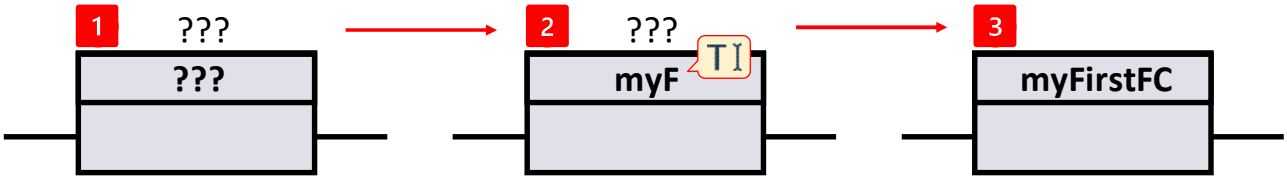
When the module is called, these formal parameters are connected to current parameters (global variables of the PLC).



Picture 7 Handover of the Akualparamter at the call

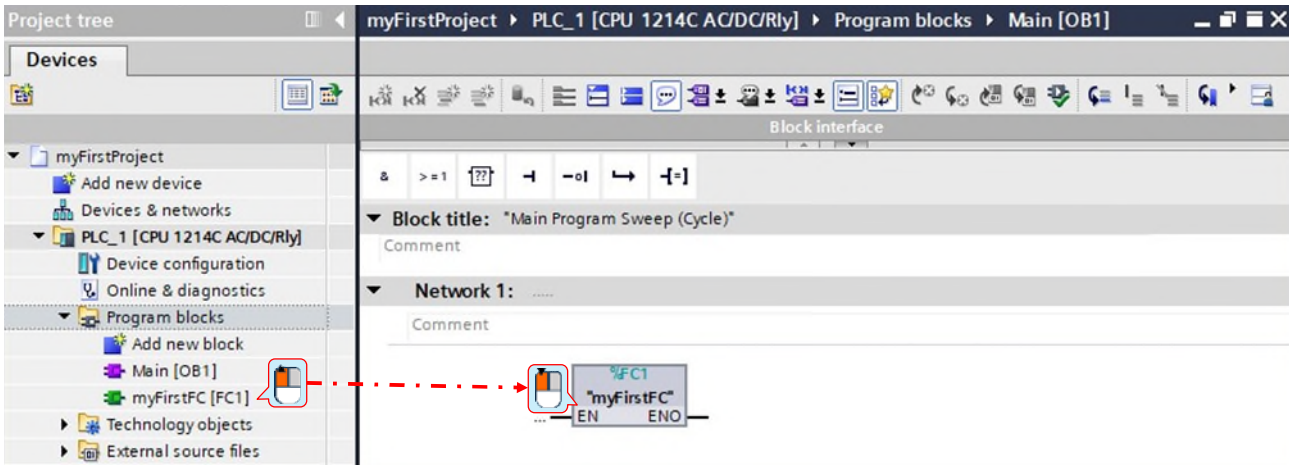
5.7 Calling a Function (FC) in FUP

In order for the function to be processed, it must be called by the program. The call can be made by inserting an empty box (TIA shortcut "F8"). After we have inserted the empty box (1), we replace (2) the placeholders ("???") in the empty box with the symbolic block name, so the empty box is replaced by the corresponding block call.



Picture 8: Block call from empty box

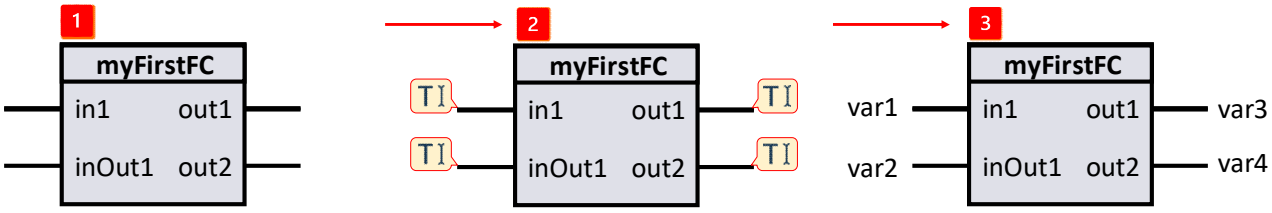
In the TIA Portal, the desired module can also be called up by drag & drop, by dragging it from the project navigation to the desired location:



Picture 9 Building block call in the TIA Portal

Parameter transfer

If the called module has interface parameters, these are displayed. In the case of functions, the formal parameters must be supplied with current parameters.

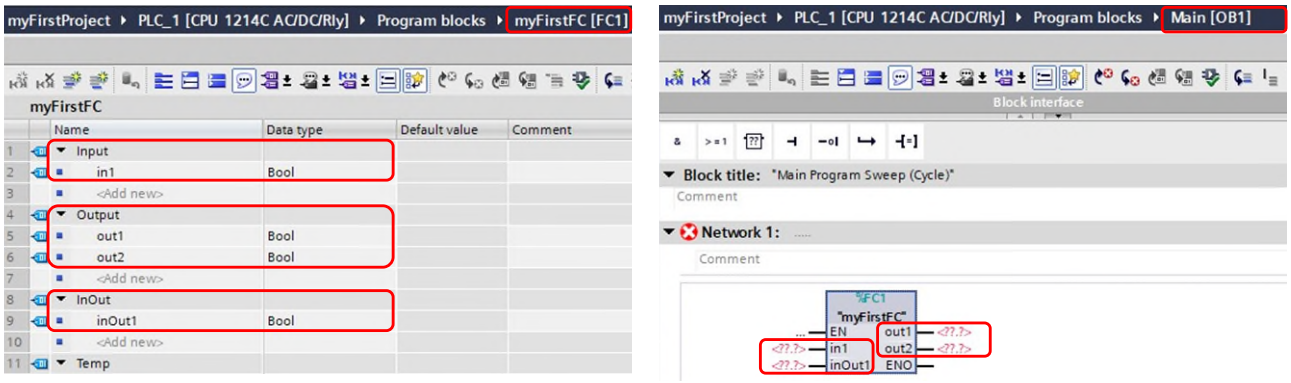


Picture 10 Function with Component Interface

Parameters of the type "Input" and "InOut" are displayed to the left of the module by means of a connecting leg. Parameters of the type "Output" must be connected to the right of the device.

Example

After creating the "myFirstFC" function, it was called in the "Main" (OB1). The parameter transfer has not yet taken place; the formal parameters to be combined were initially marked with placeholders "<??.? >".



Picture 11 Module interface in the TIA portal

5.8 Calling a function module (FB) in FUP

In order for the function module to be processed, it must be called by the program. The call can be made by inserting an empty box (TIA shortcut "F8"). After we have inserted the blank box (1) and replaced the placeholders in the blank box with the name of the function module, we replace the placeholder above the blank box with the instance name (2).

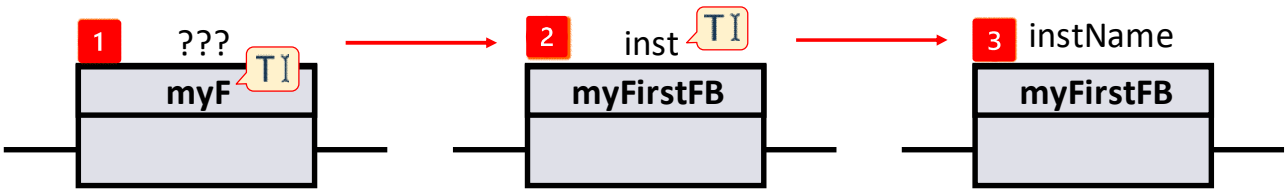
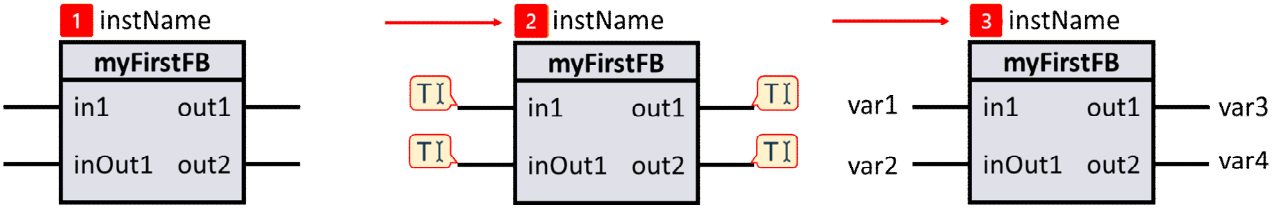


Bild 12 Funktionsbausteinaufruf aus Leerbox

Alternatively, you can call up the building block by dragging and dropping as shown under Calling a Function.

Parameter transfer

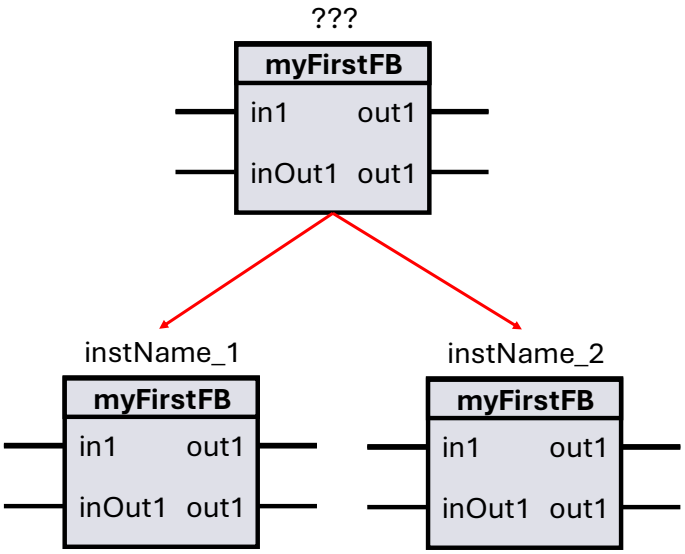
If the called function module has formal parameters in the module interface, these are displayed. For function blocks, parameter passing is not always necessary because the instance already has a private memory area assigned.



Picture 13 Function module with parameter transfer

Instantiating Function Module Multiple Times

If a function module (FB) is called (instantiated) twice in the user program, two separate instances are created. In which they can store their data throughout the cycle.



Picture 14 Instantiation of two FBs

Call options

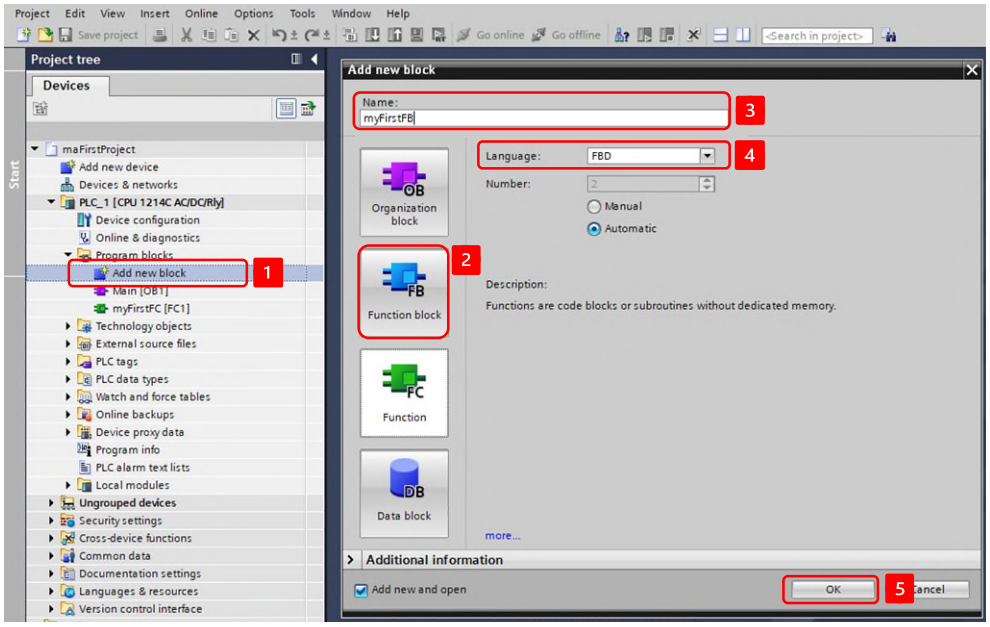
Depending on the type of calling module, the instances can be located directly in the module interface (= multi-instance in the TIA Portal) or stored as global instances (= single instance in the TIA Portal).

Example

For example, if the FB contains the user program for a calculation of switching operations, each call represents an instance that only uses states at the runtime of this call. A separate instance is required for each call.
Thus, all data (information) that belongs to this calculation is available in this assigned instance.
Before an instance can be created, the assigned FB must already exist.
The variables of the respective instance can be observed in it.

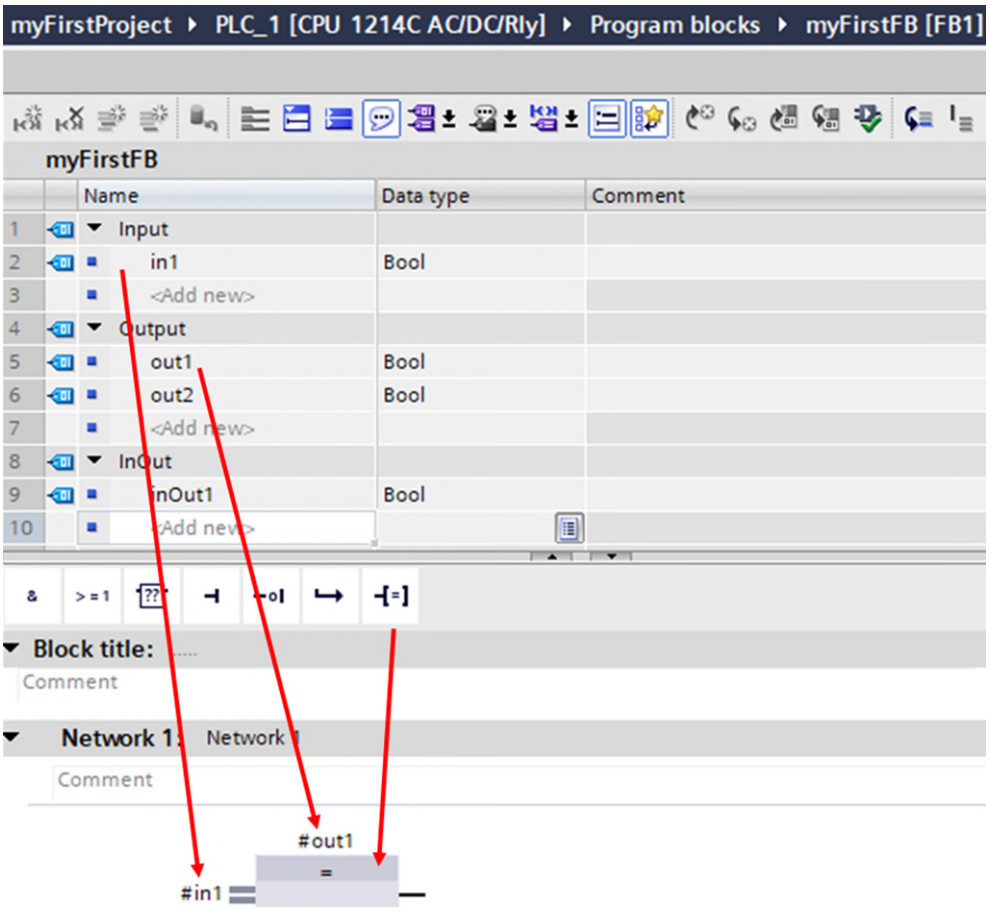
5.8.1 Procedure Call with Single Instance

The procedure for calling the function module "myFirstFB" twice is now shown step by step in the TIA portal:
Creating the function module "myFirstFB":



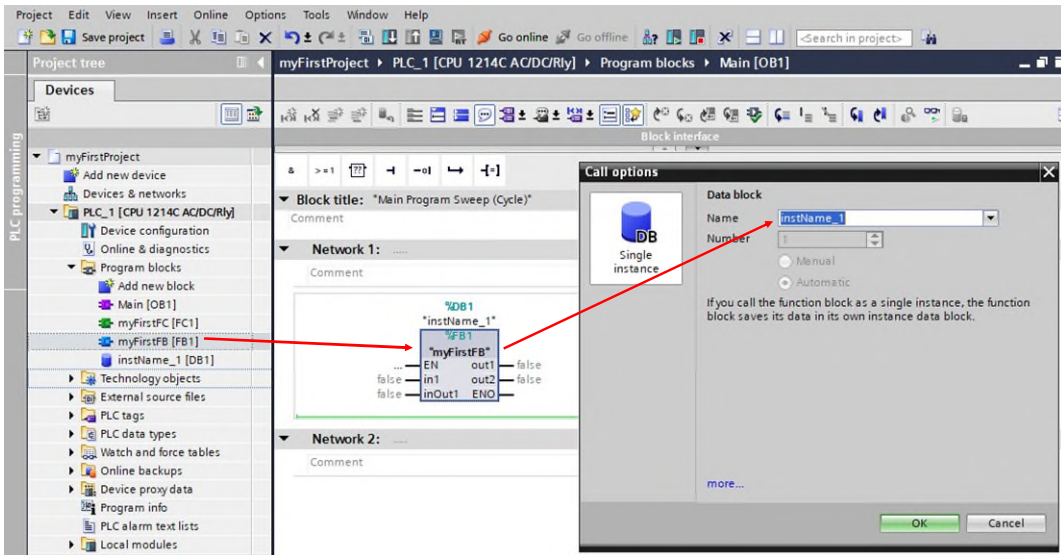
Picture 15 Adding a new building block

Declaration of the module interface and interconnection of the variables in the instruction part of the module:



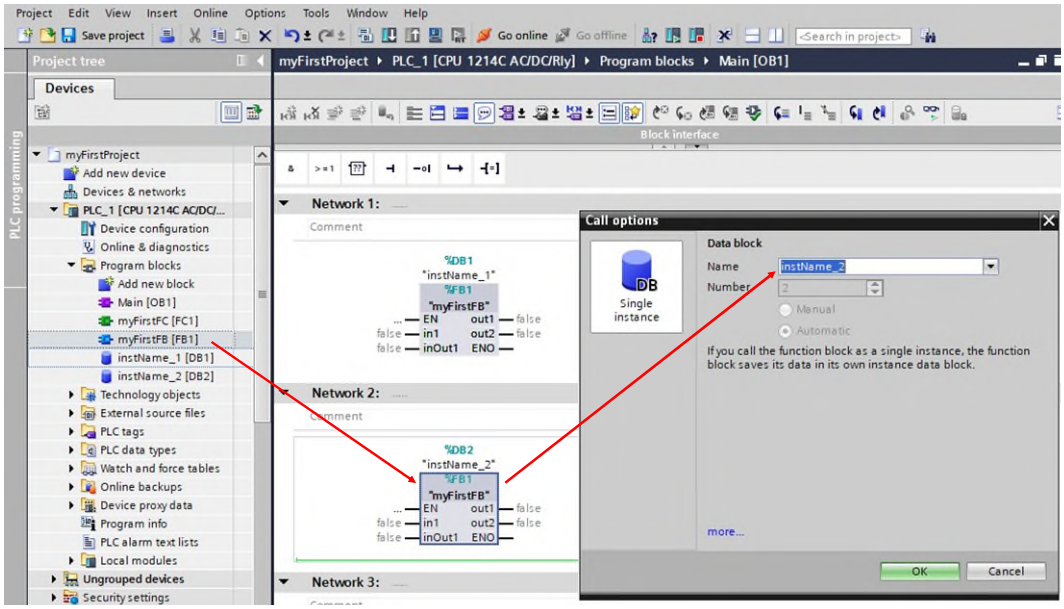
Picture 16 FB with module interface in the TIA Portal

3. First call of "myFirstFB" in the first network of the OB "MAIN" by drag & drop and declaration of the first instance as a single instance:



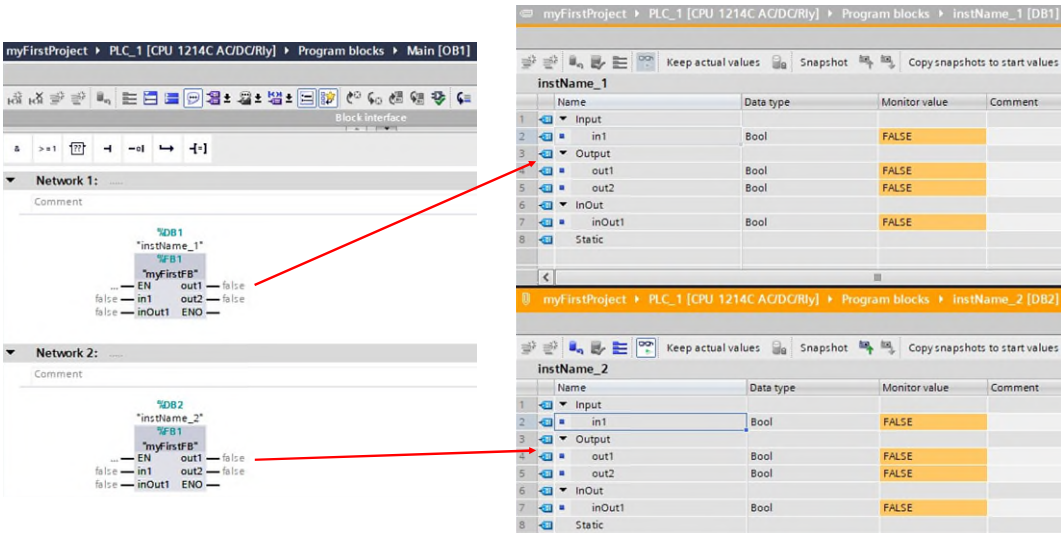
Picture 17 Instancing of the first block call

4. Second call to "myFirstFB" in the second network of the OB "MAIN" by drag & drop and declaration of the second instance as a single instance:



Picture 18 Instantiation of the second block call

5. Both instance data modules can be observed:

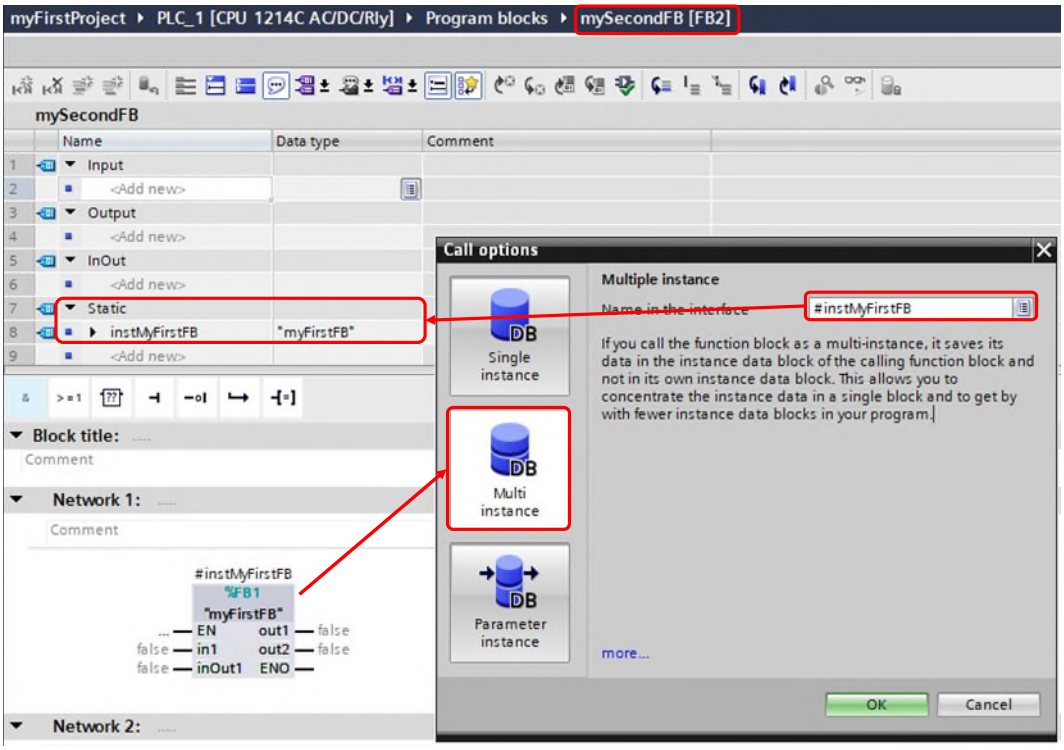


Picture 19 Current values in the instances

Instance data modules, as well as the possibility to observe and control them will be described in detail in the next chapter (Data modules).

5.8.2 Call option as multi-instance (TIA-Portal)

If a function module is called in another function module, the multi-instance can also be selected in the call options.
By using multi-instances, the number of instance data modules can be reduced. When creating subroutines, the use of multi-instances is often mandatory. If, for example, you want to implement a runtime counter in a control module for a motor, each motor instance needs its own IEC counter instance.
Multi-instances are placed in the building block interface of the calling building block.



Picture 20 Call Options in the TIA Portal

5.8.3 Textuelle Deklaration als Multiinstanz (CODESYS / Beckhoff)

The textual declaration of the instances is carried out according to the following scheme.
Syntax:

Instance Name (Variable Name) : Block Name (Data Type);

Example:

```
//Declaration of instances
VAR
  instName_1 : myFirstFB;   //Instance 1
  instName_2 : myFirstFB;   //Instance 2
END_VAR
```

Picture 21 Declaration of Instances

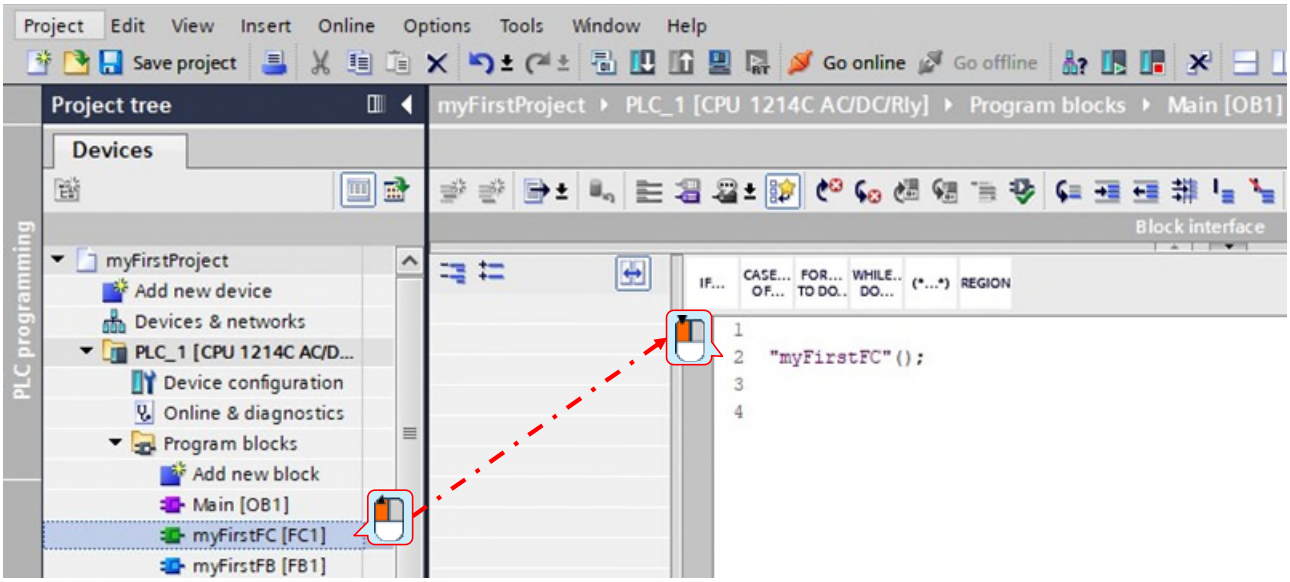
5.9 Calling a function (FC) in ST/SCL

In order for the function to be processed, we want to call it up in the program. A function call with no return value in ST/SCL is made by the function name, followed by "()" and a semicolon. In the parentheses, the parameter is passed, and the semicolon concludes the statement.

```
myFirstFC();
```

Picture 22 Function call ST / SCL

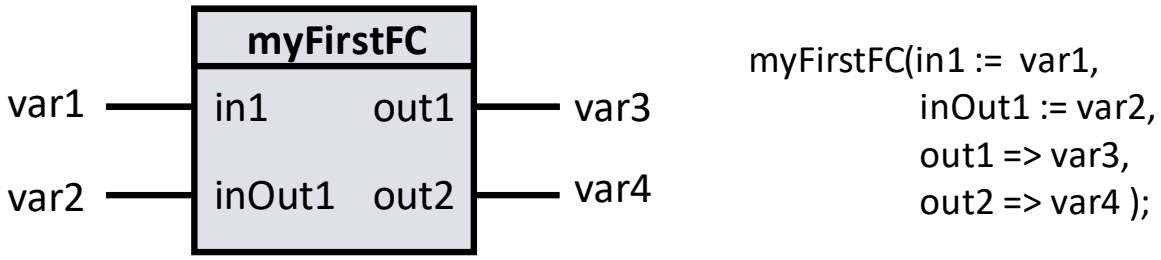
In the TIA Portal, the desired module can also be called up by drag & drop, by dragging it from the project navigation to the desired location:



Picture 23 Building block call in the TIA Portal

Parameter transfer

If the called module has interface parameters, these are displayed. In the case of functions, the formal parameters must be supplied with current parameters.
The parameter is passed in round brackets, parameters are separated from each other with "."

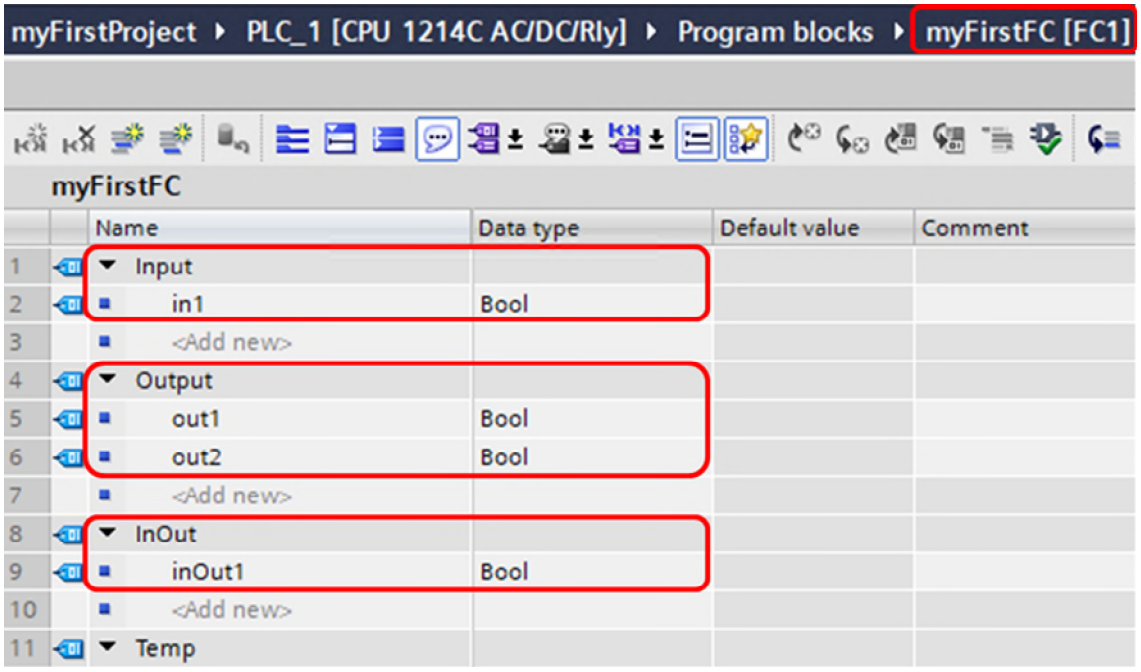


Picture 24 Syntax function call, with parameter passing in SCL

Parameters of the type "Input" and "InOut" are assigned using ":=". Parameters of type "Output" must be connected with "=>".

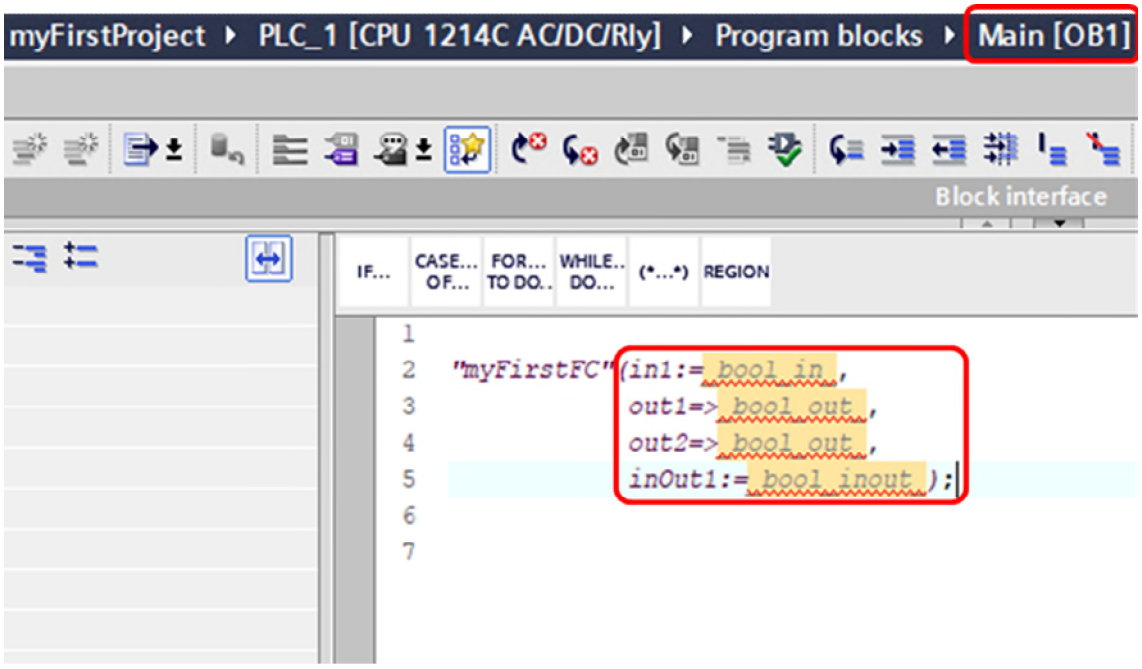
Example

In this figure, the following interface parameters have been declared in the "myFirstFC" function.



Picture 25 Component interface of a function in the TIA Portal

After creating the "myFirstFC" function, it was called in the "MAIN" (OB1). The parameter transfer has not yet been carried out, and the formal parameters to be connected are assigned placeholders for the time being. These placeholders provide information about the data type and parameter type (Input, Output, InOut).



Picture 26 Module interface in the TIA portal

5.10 Calling a function module (FB) in ST / SCL

To work through the function module, we call it up in the program. The main difference between calling a function module (FB) is that an instance is assigned to it. The call is similar to a function (FC), but instead of the building block name, the name of the previously declared instance is used, followed by "()" and a semicolon. In the parentheses, the parameter is passed, and the semicolon concludes the statement.

```
instMyFirstFB();
```

Picture 27 Syntax instance, without parameter passing in SCL

Parameter transfer

If the called instance has interface parameters, these are displayed. For functional building blocks, parameter passing is not always necessary because the instance already has a private memory area associated with it.

```
instMyFirstFB(in1 :=  var1,
              inOut1 := var2,
              out1 => var3,
              out2 => var4 );
```

Picture 28 Syntax instance, with parameter passing in SCL

Call options

Depending on the type of calling module, the instances can be located directly in the module interface (= multi-instance in the TIA Portal) or stored as global instances (= single instance in the TIA Portal).

Instantiating Function Module Multiple Times

If a FB is called twice in the user program, there are two instances. Each instance is assigned its own memory area, in which the instance can store its data throughout the cycle.

Example

For example, if the FB contains the user program for a calculation of switching operations, each call represents an instance that only uses states at the runtime of this call. A separate instance is required for each call.
Thus, all data (information) that belongs to this calculation is available in this assigned instance.
Before an instance can be created, the assigned FB must already exist.
The variables of the respective instance can be observed in it.

5.10.1 Procedure Call with Single Instance

The procedure for calling the function module "myFirstFB" twice is now explained step by step in the TIA Portal.

1. Creating the function module "myFirstFB":

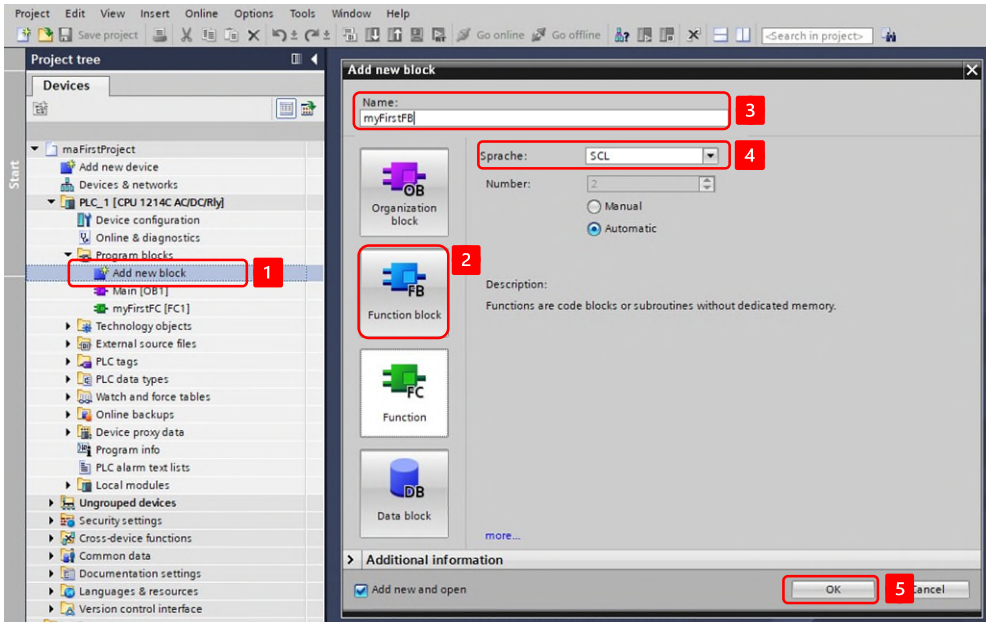
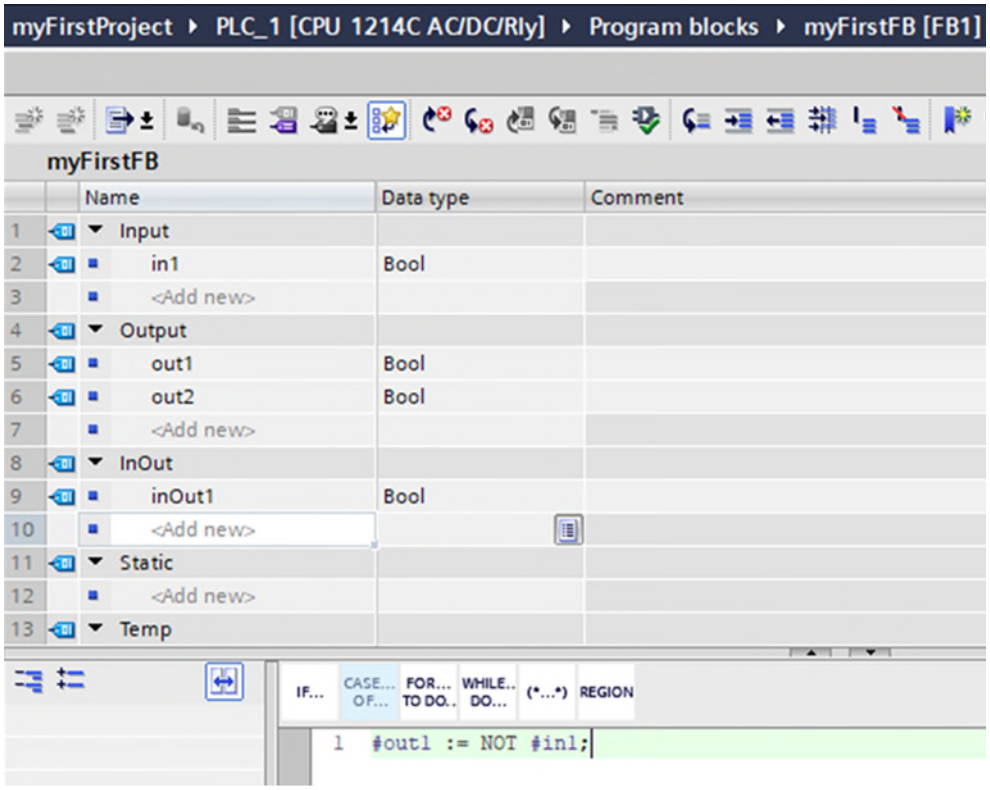


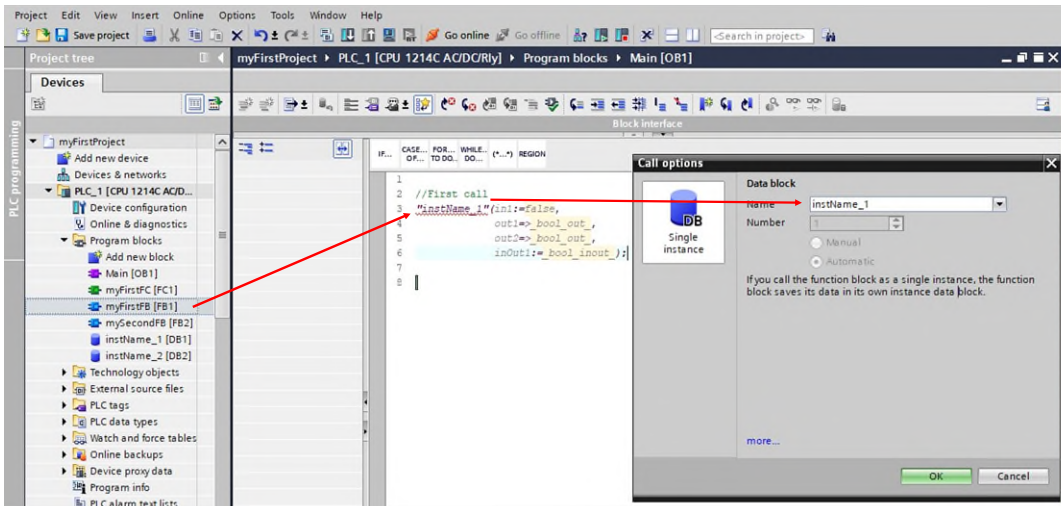
Bild 29 Neuen Baustein hinzufügen

2. Declaration of the module interface and interconnection of the variables in the instruction part of the module:



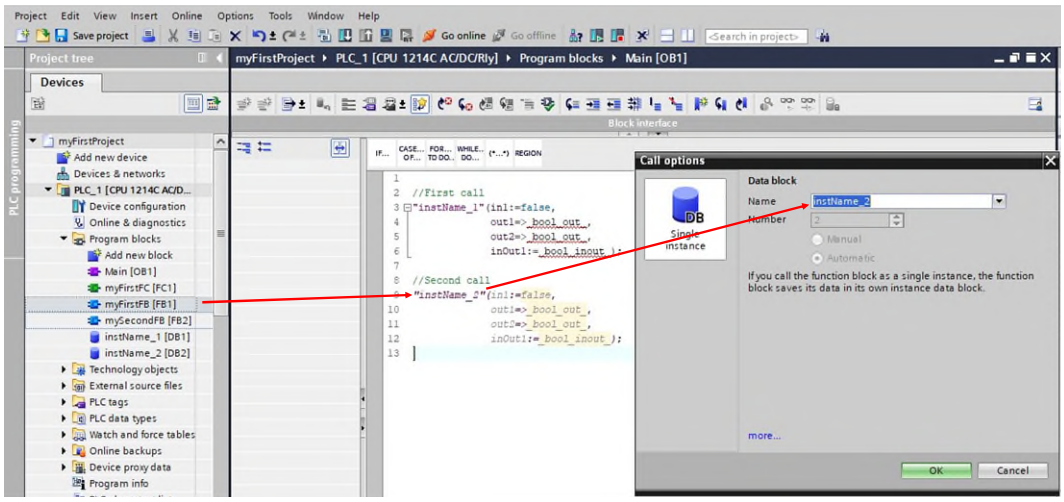
Picture 30 FB with module interface in the TIA Portal

3. First call of "myFirstFB" in the OB "MAIN" by drag & drop and declaration of the first instance as a single instance:



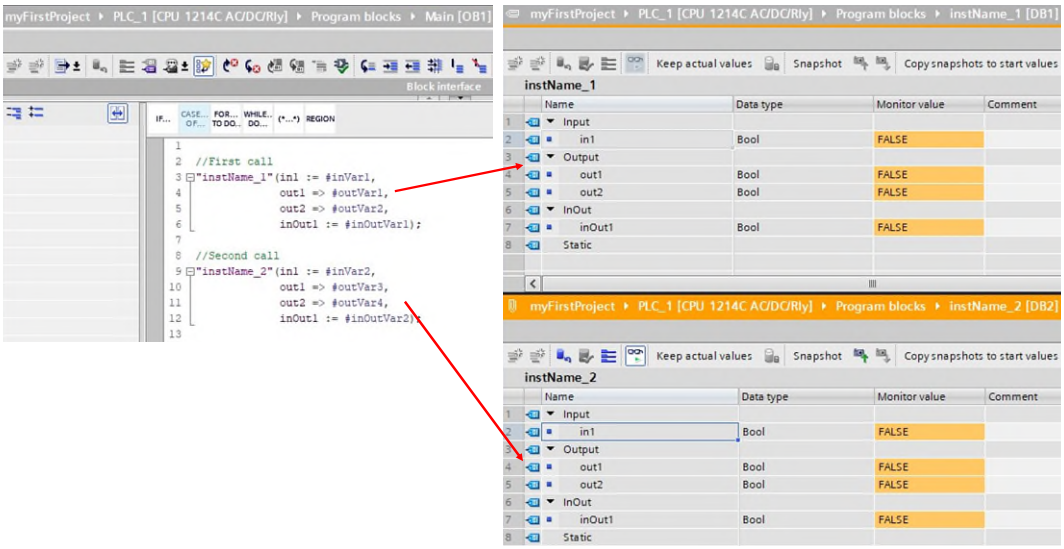
Picture 31 Instantiation of the first block call

4. Second call of "myFirstFB" in the OB "MAIN" by means of drag & drop and declaration of the second instance as a single instance:



Picture 32 Instantiation of the second block call

5. Both instance data modules can be observed:

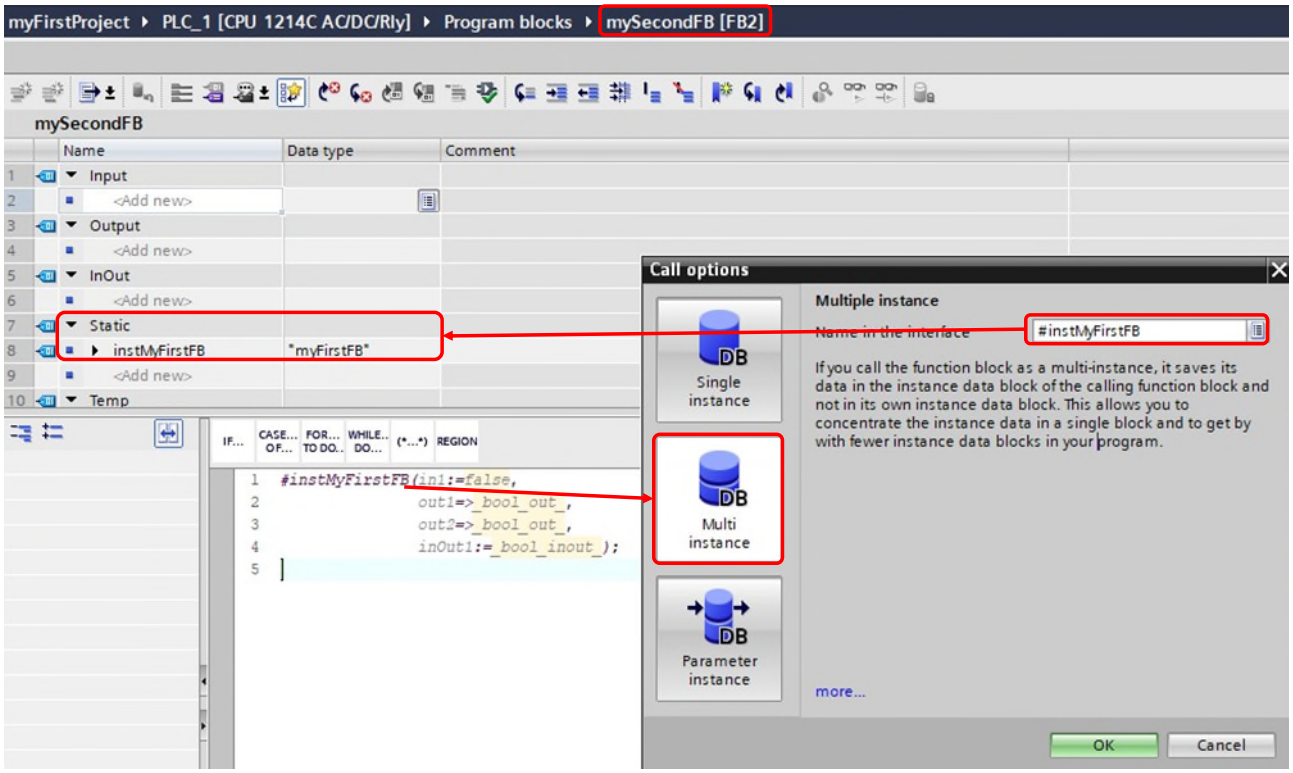


Picture 33 Current values in the instances

Instance data modules, as well as the possibility to observe and control them will be described in detail in the next chapter (Data modules).

5.10.2 Call option as multi-instance (TIA-Portal)

If a function module is called in another function module, the multi-instance can also be selected in the call options.
By using multi-instances, the number of instance data modules can be reduced. When creating subroutines, the use of multi-instances is often mandatory. If, for example, you want to implement a runtime counter in a control module for a motor, each motor instance needs its own IEC counter instance.
Multi-instances are placed in the building block interface of the calling building block.



Picture 34 Call options in the TIA Portal

5.10.3 Textual declaration as a multi-instance (CODESYS / Beckhoff)

The textual declaration of the instances is carried out according to the following scheme.

Syntax:

Instance Name (Variable Name) : Block Name (Data Type);

Example:

```
//Declaration in the function block interface
VAR
  instMyFirstFB : myFirstFB;   //declaration of the instance
END_VAR


//Implementation in the program
instMyFirstFB(in1:=       ,
              inOut1:=    ,
              out1=>      ,
              out2=>      );
```

Picture 35 Declaration and Call of the Instance

Instantiating Function Module Multiple Times

If a FB is called twice in the user program, there are two instances. Each instance is assigned its own memory area, in which the instance can store its data throughout the cycle.

```
//Declaration in the function block interface
VAR
   instName_1 : myFirstFB;   //Instance 1
   instName_2 : myFirstFB;   //Instance 2
END_VAR



//Implementation in the program
//Call 1
instName_1(in1:=          ,
           inOut1:=       ,
           out1=>         ,
           out2=>         );

//Call 2
instName_2(in1:=          ,
           inOut1:=       ,
           out1=>         ,
           out2=>         );
```

Picture 36 Instancing of two FBs