INVT Medium and Large-scale PLC Programming Manual

SHENZHEN INVT ELECTRIC CO., LTD.

Preface

Overview

Thank you for choosing our medium and large-scale PLC.

This manual contains the information necessary for using the medium and large-scale PLC. Please read this manual carefully before use to fully understand the functions and performance of the product, and complete system construction, which helps to give full play to the product's superior performance.

This manual is applicable to AX, TM, and TP series PLCs, but attention should be paid to the applicable scope of some function instructions.

Target Audience

This manual is intended for personnel with professional knowledge of electrical engineering (e.g., qualified electricians or personnel with equivalent knowledge).

Online Support

In addition to this manual, you can also obtain product information and technical support from our website.

Website: https://www.invt.com

If the product is ultimately used for military affairs or weapons manufacturing, please comply with the export control regulations in the *Foreign Trade Law of the People's Republic of China*, and complete related export formalities if required.

Revision History

The Company reserves the right to continuously improve the product without prior notice.

Number	Revision Description	Version	Release Date
1	First release.	V1.0	September 2024

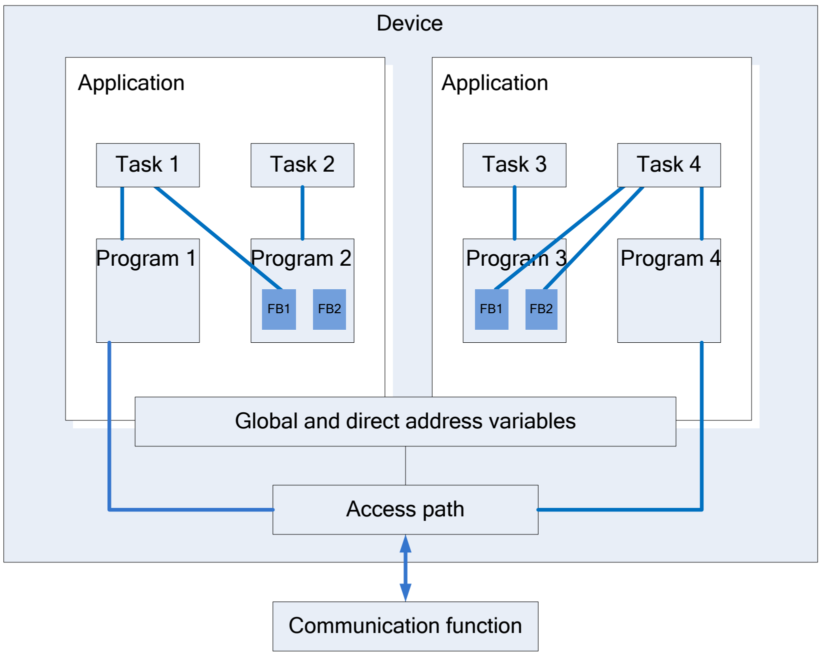
Contents

1 Program Structure and Execution

1.1 Program Structure

The software model is represented by a hierarchical structure and describes basic software elements and their relationships, with each layer implying many characteristics of the layers below it. These software elements include devices, applications, tasks, global variables, access paths, and application objects. Their internal structure is shown in Figure 1-1. The software model is consistent with that specified in the IEC 61131-3 standard.

Figure 1-1 Program Hierarchy



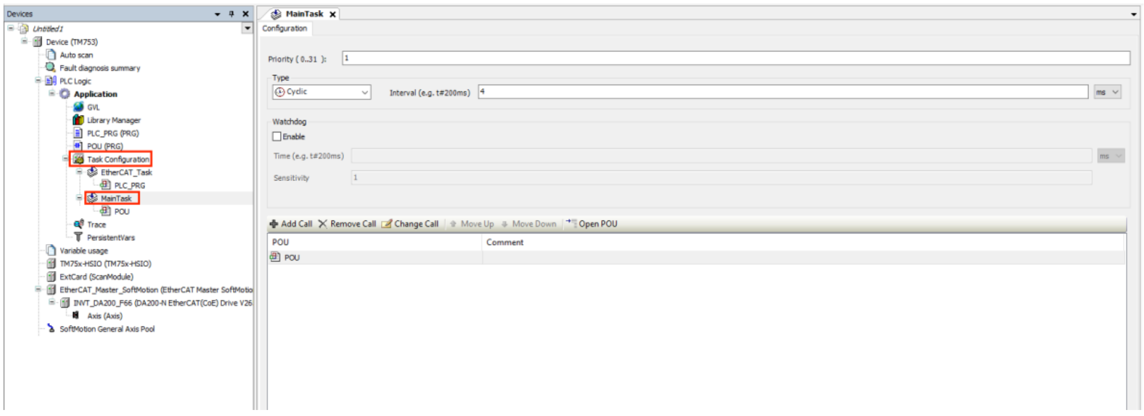
1.2 Task

A program can be written in different programming languages. A typical program consists of many interconnected function blocks that can exchange data with each other. The execution of different parts of a program is controlled by “tasks”. Once a “task” is configured, a series of programs or function blocks can be executed periodically or triggered by a specific event.

The “Task Manager” tab in the device tree can be used to control the execution of other subprograms within the project, in addition to the specific PLC_PRG program. A task is used to define the properties of a program organization unit at runtime. It is an execution control element with the calling ability. Multiple tasks can be created in a task configuration, and multiple program organization units can be called in a task. Once a task is set up, it can control the program’s cyclic execution or start execution through a specific event trigger.

In the task configuration, a task is defined with a name, priority, and startup type. The startup type can be defined by time (periodic, random) or by an internal or external trigger task time, for example, using a rising edge of a Boolean global variable or a specific event in the system. For each task, you can set a series of programs that are started by the task. If the task is executed in the current cycle, these programs will be processed within the duration of one cycle. The combination of priority and conditions will determine the task execution timing. The Task Configuration interface is shown in Figure 1-2.

Figure 1-2 Task Configuration Interface



Priority (0..31): 1

Type: Cyclic Interval (e.g. t#200ms): 4 ms

Watchdog

☐ Enable

Time (e.g. t#200ms): ms

Sensitivity: 1

POU	Comment
POU	

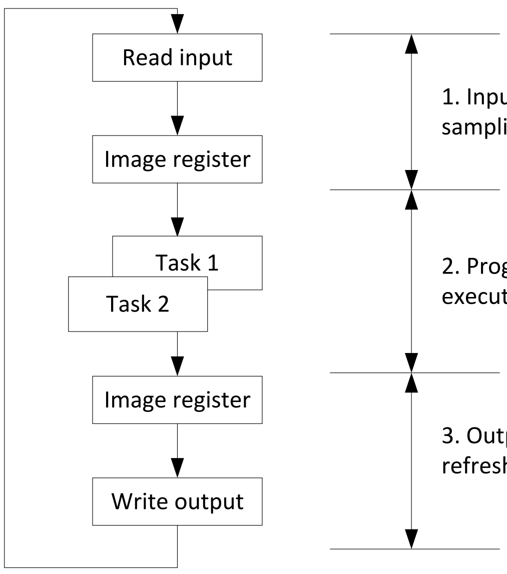
Programmers must follow the following rules:

- The maximum number of cyclic tasks is 100, the maximum number of freewheeling tasks is 100, and the maximum number of event-triggered tasks is 100.
- Depending on the target system, PLC_PRG may be executed in any case as a free program without being manually inserted into the Task Configuration.
- Processing and calling programs are executed in a top-down sequence within the Task Editor.

1.3 Program Execution Process

The figure below describes in detail the complete process of executing a program inside the PLC, which mainly consists of three parts: input sampling, program execution, and output refreshing.

Figure 1-3 Controller Execution Process



- Input sampling

At the beginning of each scan cycle, the PLC detects the status of the input device (switches, buttons, etc.) and writes the status into the input image register. During the program execution stage, the operating system reads data from the input image register to solve the program. It is important to note that input refreshing only occurs at the beginning of the scan cycle. During the scan, even if the output state changes, the input state will not change.

- Program execution

During the program execution stage of the scan cycle, the PLC reads the status and data from the input image register or the output image register, and performs logical and arithmetic operations according to the instructions. The results of the operations are stored in the corresponding cells in the output image register. During this stage, only the content in the input image register remains unchanged, while the content in other image registers will change with the program execution.

- Output refreshing

The output refreshing stage can also be called the write output stage. The PLC transmits the status and data from the output image register to the output point, and drives an external load through certain isolation and power amplification. In addition to completing the tasks of the above three stages in one scan cycle, the PLC also has to complete auxiliary tasks such as internal diagnosis, communication, public processing, and input/output services.

The PLC repeats the above process, and the time for each repetition is a work cycle (or scan cycle). It can be seen from the scanning method of the PLC that in order to quickly respond to changes in input and output data and complete control tasks, the scanning time is short and the controller's work cycle is generally controlled at the ms level. Therefore, it is necessary to develop a stable, reliable, and fast-response real-time system for the PLC operating system.

Since the PLC employs a cyclic work mode, the input signal is only refreshed at the beginning of each cycle and the output is output in a concentrated manner at the end of each work cycle, which inevitably causing a lag between the output signal and the input signal. When a signal is input from the input end and transmitted to the output end of the PLC, it takes some time to respond to the change of the input signal. The lag time is an important parameter that should be understood when a PLC control system is designed. Generally, the length of the lag time is related to the following factors.

1. Filtering time of the input circuit, which is determined by the time constant of the hardware RC filter circuit. The input lag time can be adjusted by changing the time constant.

Table 1-1 lists the technical parameters of the AX-EM-1600D digital input module, where the "Port filtering time" indicates the filtering time of the input module.

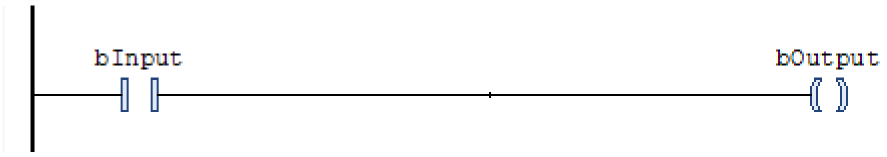
Table 1-1 Technical Parameters of the AX-EM-1600D Digital Input Module

Item	Specification
Input channel	16
Input connection method	18-pin terminal block
Input voltage class	24 V (up to 30 V)
Input current (typical)	4.7 mA
ON voltage	＞ 15 VDC
OFF voltage	＜ 5 VDC
Port filtering time	10 ms
Input resistance	5.4 kΩ
Input signal type	VDC input
Isolation method	Opto-coupler
Input dynamic display	The indicator is on when the input is valid.

2. Lag time of the output circuit, which is related to the mode of the output circuit. The lag time of the relay output mode is generally about 10 ms, while that of the transistor output mode is less than 1 ms.
3. Cyclic scan mode of the controller.
4. Arrangement of statements in the user program.

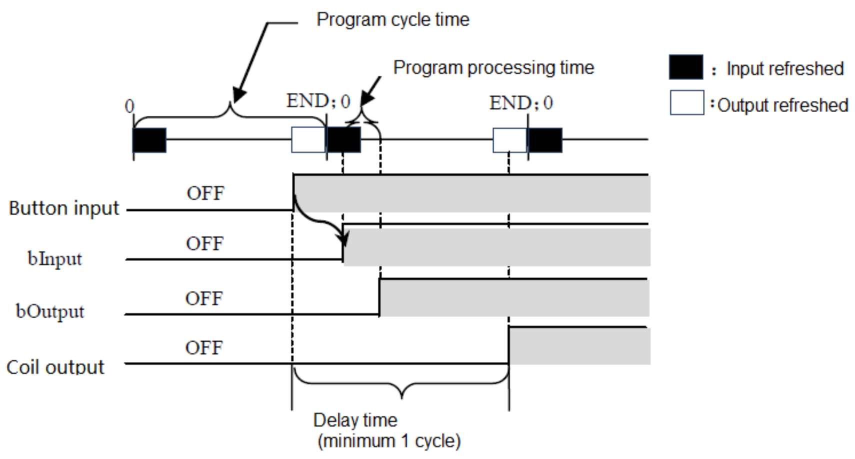
To allow readers to better understand the whole process, a simple ladder diagram program example is given below to show its input and output and how the lag is produced. The program logic is shown in Figure 1-4.

Figure 1-4 TM-series PLC Program



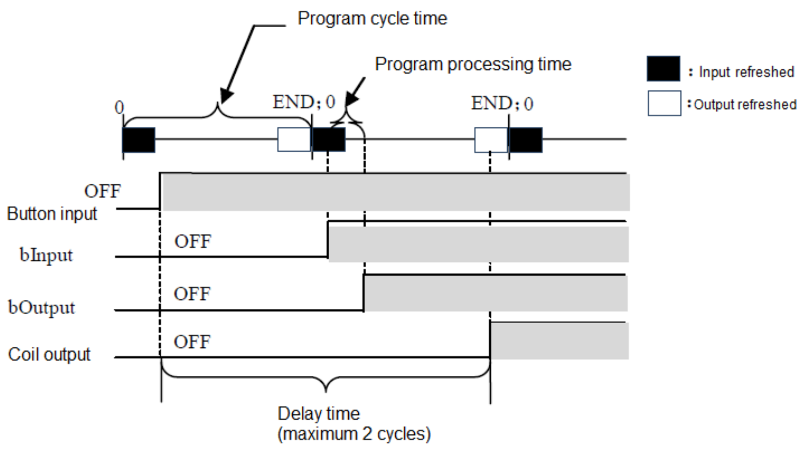
bInput has a hardware mapping relationship with an external input button, and when the button is pressed, bInput is ON. Meanwhile, bOutput has a hardware mapping relationship with the coil of an external relay, and when bOutput is ON, the coil is also energized. The relationship processed inside the PLC is shown in Figure 1-6. When the input button is pressed, bInput will not be set to ON immediately, because input sampling can only be executed by the program at the beginning of a work cycle. Since the button signal has passed the sampling stage, it will usually be executed at the beginning of the next work cycle. In the program shown in Figure 1-6, the state of bInput is assigned to bOutput. Since there are certain program operations during program running, it takes a certain amount of program processing time for bOutput to be set to ON. Since output refreshing occurs at the last stage of program processing, bOutput passes its value to actual hardware through the output refreshing function at the last stage of the cycle, and finally the coil can be energized. Figure 1-5 shows a relatively ideal state, in which the final output is only lagged by one cycle.

Figure 1-5 Fastest Output



While Figure 1-5 shows a relatively ideal state, we also need to consider a worse situation. When the input sampling of a cycle has just ended, the external input button is ON, Since the input signal can only be loaded into the input image register at the beginning of the next cycle, and the actual output can only be loaded into the output image register at the end of the next cycle, the whole process is shown in Figure 1-6. In this case, the output is lagged by about 2 cycles, which is the slowest output.

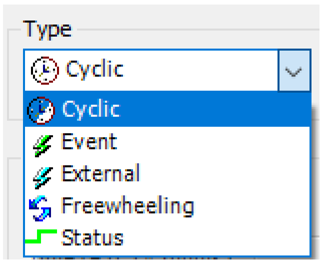
Figure 1-6 Slowest Output



1.4 Task Execution Type

There is an entry named “Task Configuration” at the very top of the task configuration tree, which includes the currently defined tasks, each represented by a task name. The call of POUs for specific tasks is not displayed in the task configuration tree. The execution type of each individual task can be edited and configured, including 4 types: Cyclic, Event, Freewheeling, and Status.

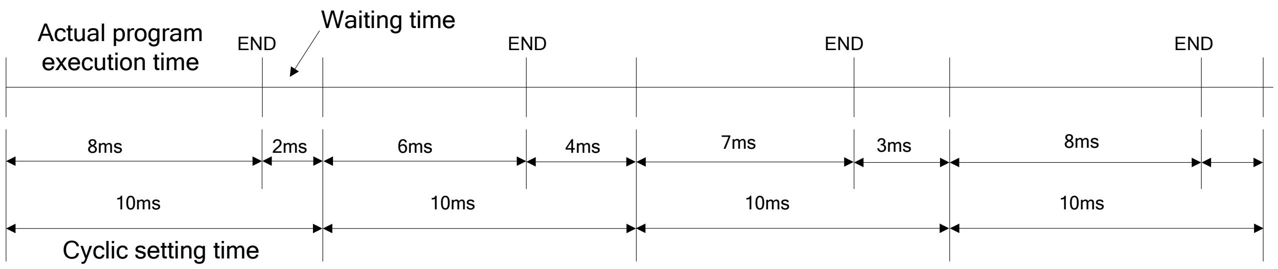
Figure 1-7Task Execution Types



1. Cyclic

The program processing time will vary depending on whether the instructions used in the program are executed or not, so the actual execution time varies in each scan cycle and may be longer or shorter. By using the Cyclic type, the program can be repeatedly executed while maintaining a certain cycle time. Even if the program execution time changes, a certain refresh interval can be maintained. Here, we recommend that you give priority to the Cyclic type. For example, if you set the corresponding task to the Cyclic type and the interval to 10 ms, the actual program execution timing is shown in Figure 1-8.

Figure 1-8 Cyclic Execution Timing

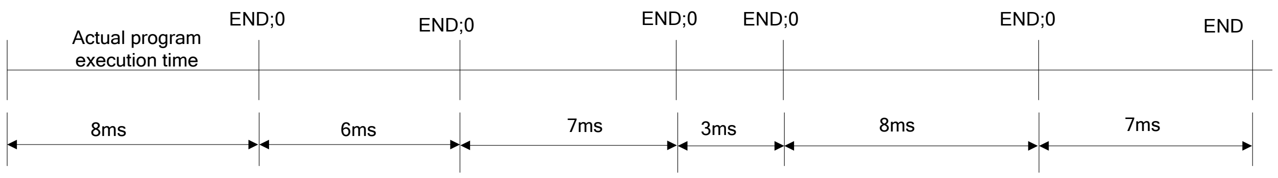


If the program is actually completed within the specified Cyclic setting time, the remaining time is used for waiting. If there are still lower-priority tasks in the application that have not been executed, the remaining waiting time is used to execute the these lower-priority tasks. The priority of tasks will be explained in detail later.

2. Freewheeling

The task will be processed as soon as the program starts running, and will be automatically restarted in the next cycle after one running cycle ends. This execution type is not affected by the program scan cycle. That is, it ensures that the next cycle starts only after the last instruction of the program is executed, otherwise the current cycle will not end. Figure 1-9 shows the freewheeling execution timing.

Figure 1-9 Freewheeling Execution Timing



Since there is no fixed task time for the freewheeling execution type, the execution time may be different each time. Therefore, the real-time performance of the program cannot be guaranteed, and this type is rarely used in actual applications.

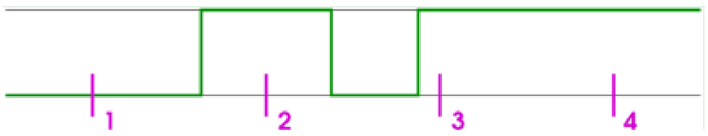
3. Event

If the variable in the event area receives a rising edge, the task starts.

4. Status

If the variable in the event area is TRUE, the task starts. The Status type is similar to the Event type except that the program will be executed as long as the trigger variable is TRUE, and will not be executed if it is FALSE. The Event type only collects the valid signal of the rising edge of the trigger variable. Figure 1-10 compares the Event type and the Status type. The green solid line represents the Boolean variable state selected by the two trigger types. The comparison results are listed in Table 1-2.

Figure 1-10 Task Input Trigger Signal



Different types of tasks at sampling points 1-4 (purple) show different responses. The trigger condition of the Status type is fulfilled when a specific event is TRUE, but an event-triggered task requires the event to change from FALSE to TRUE. If the task is scheduled to sample too slowly, the rising edge of the event may not be detected.

Table 1-2 Comparison of Event-triggered and Status-triggered Execution Results

Execution Point	**1**	**2**	**3**	**4**
Event	Not executed	Executed	Executed	Executed
Status	Not executed	Executed	Not executed	Not executed

1.5 Task Priority

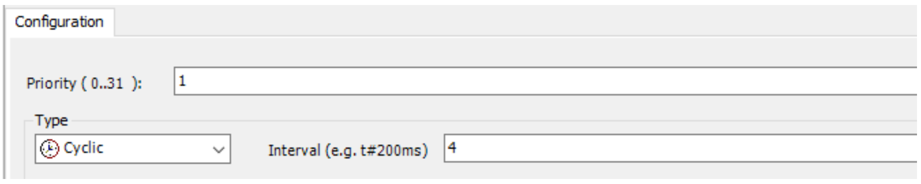
1. Task priority setting

You can set the priority of a task, with a total of 32 levels (a number between 0 and 31, with 0 representing the highest priority and 31 representing the lowest priority). When a program is being executed, high-priority tasks take precedence over low-priority tasks. A task with the highest priority 0 can interrupt the execution of lower-priority programs in the same resource, causing the execution of the lower-priority programs to be slowed down.

Note: When assigning task priority levels, do not assign tasks with the same priority. If there are other tasks that precede the task with the same priority, the results may be uncertain and unpredictable.

If the task execution type is “Cyclic”, the task will be executed cyclically according to the “interval”. The specific settings are shown in Figure 1-11.

Figure 1-11 Cyclic Configuration



Priority (0..31): 1

Cyclic

Interval (e.g. t#200ms) 4

Example: Assuming there are 3 different tasks, corresponding to three different priority levels, the specific allocation is as follows.

⬜: Task 1 has a priority level of 0 and a cycle time of 10 ms.

🟥: Task 2 has a priority level of 1 and a cycle time of 30 ms.

🟥: Task 3 has a priority level of 2 and a cycle time of 40 ms.

The timing relationship of each task inside the controller is shown in Figure 1-12, and explained as follows:

0-10 ms: Task 1 (with the highest priority) is executed first, and if the program is finished within the current cycle, the remaining time will be used to execute Task 2. However, if Task 2 has not been fully executed after 10 ms, it will be interrupted because Task 1 is executed every 10 ms and has the highest priority.

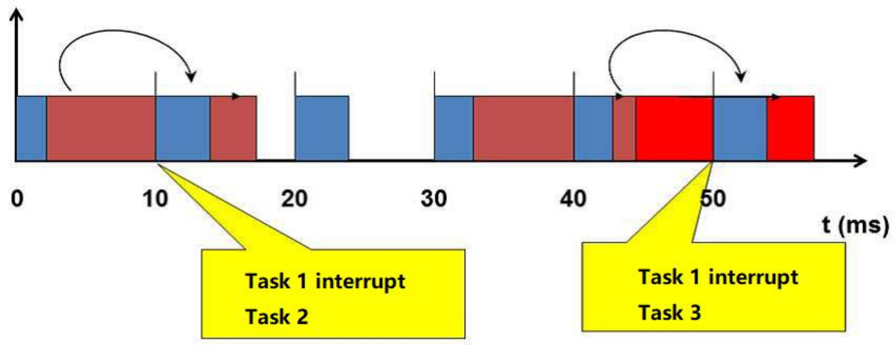
10-20 ms: Task 1 is executed first, and if there is any time left, the unfinished Task 2 in the previous cycle will be executed.

20-30 ms: Since Task 2 is executed every 30 ms and has been finished within 10-20 ms, there is no need to execute Task 2 at this time and only Task 1 with the highest priority is executed once.

30-40 ms: Similar as above.

40-50 ms: Task 3 appears at this time. Since Task 3 has the lowest priority, it can only be executed after ensuring that Task 2 has been thoroughly executed.

Figure 1-12 Task Priority Interrupt Execution Sequence



2. Task priority setting of AX, TM, and TP-series PLCs

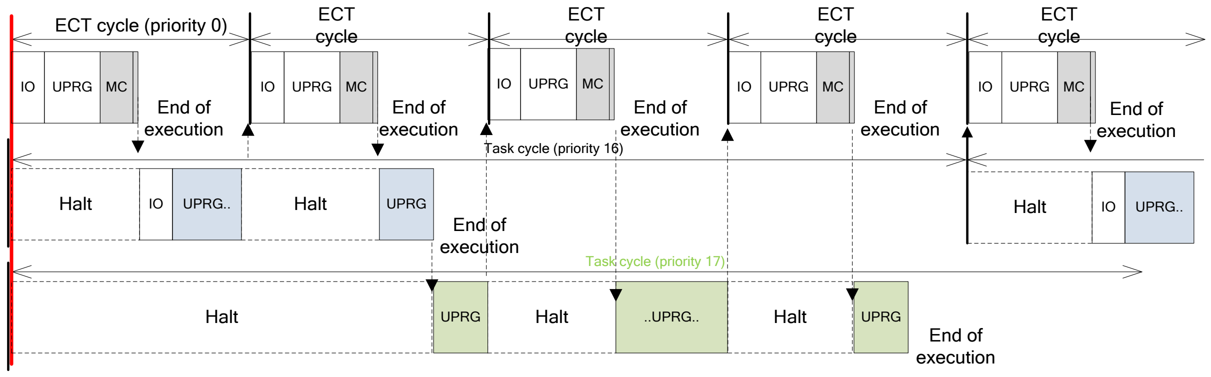
When the upper computer software of the AX, TM, and TP-series controllers creates a new standard project, a MainTask is created by default in the Task Configuration, with its priority being 1 by default. The priority of newly created tasks is also 1 by default, but to ensure that important tasks such as motion control are prioritized, the performance of the controllers can be given appropriate play in some applications that require high-performance motion control (MC). The following order is recommended for task priority setting (if there is only one task, the task priority can be set arbitrarily).

Table 1-3 Task Priority Setting

Task Type	Recommended Priority
RTC_Mod and other system parameter modules	31
ModbusTCP	15–30
ModbusRTU	15–30
High-speed I/O	1–15
Analog input and output modules	1–15
Temperature Module	1–15
EtherCAT	0

The smaller the priority setting value, the higher the priority. A high-priority POU can interrupt the execution of a low-priority POU, as shown in Figure 1-13, where ECT stands for EtherCAT.

Figure 1-13 POU Execution Sequence



It can be seen from Figure 1-13 that when the controller executes tasks, there is a time alignment point that cannot be observed by users, as shown on the left side of the above figure. Starting from this time point, the tasks are executed in an order of the highest priority → the next highest priority → the lowest priority.

A low-priority task may be interrupted by a high-priority task while it is being executed, and when the execution of the high-priority task is finished, the interrupted low-priority task will be returned to continue.

The EtherCAT task is a task with the highest priority, which is executed according to the EtherCAT cycle, and all POUs within the task are completely executed once before the tasks with lower priority are returned.

3. Requirements for execution cycle setting in Task Configuration

The upper computer software of medium and large-scale PLC systems execute the "tasks" of user programs in a multi-task mode, and each "task" is assigned a different execution cycle. Some global variables may need to be accessed and modified between different POUs, so the global variables need to be interactively synchronized, which is also performed at the "time alignment point" of the task. When the cycle of a cyclic type task is set, the cycle times of different types of cyclic tasks are integer multiples of each other.

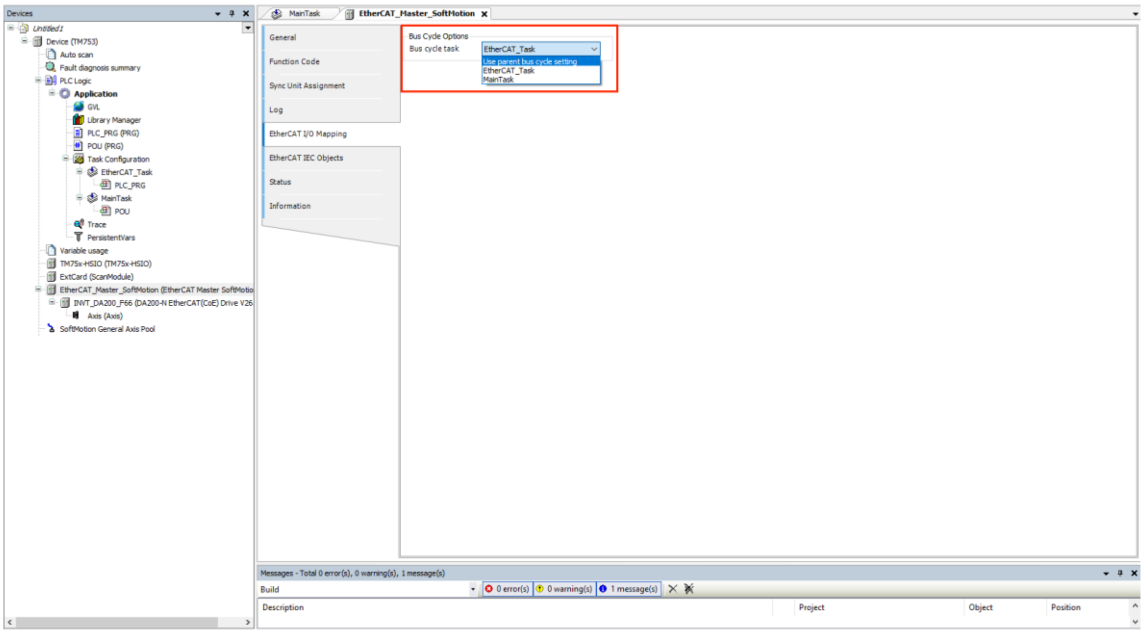
For example, the cycle time of the EtherCAT task is set to 4 ms or 8 ms, the cycle time of a normal cyclic task is set to 400 ms, and the cycle time of a lower-priority task is set to 100 ms or 200 ms. The cycle time of the EtherCAT task should not be set to 5 ms, 7 ms, 9 ms, etc., as it may easily cause an abnormal relationship other than an integer multiple of 2.

4. Considerations in configuration of sub-device bus cycle options

In the controller device "PLC Setting → Bus Cycle → Bus Cycle Task" option, the list of Bus Cycle Task Options provides the tasks defined in the Task Configuration of the current valid project (such as "MainTask" and "EtherCAT Master"). If you select one of the tasks as the bus cycle of the current project, or select the option <unspecified>, it means that the shortest task cycle time or the fastest execution cycle will be applied. You can switch to another setting, but be sure to understand the following:

Before modifying the <unspecified> setting, you should be aware of its impact. It is a default action defined by the device description. So, please check the description for this. By default, the task may be defined as having the shortest cycle time, but it may also have the longest cycle time. Therefore, when using expansion modules and EtherCAT modules, in order to improve the system operation stability (especially when using the EtherCAT_Master_SoftMotion module), select the tasks corresponding to each module in "EtherCAT I/O Mapping → Bus Cycle Option". The reference routine is shown in Figure 1-14.

Figure 1-14 Example of EtherCAT Bus Cycle Task Setting



Bus Cycle Options
Bus cycle task: EtherCAT_Task

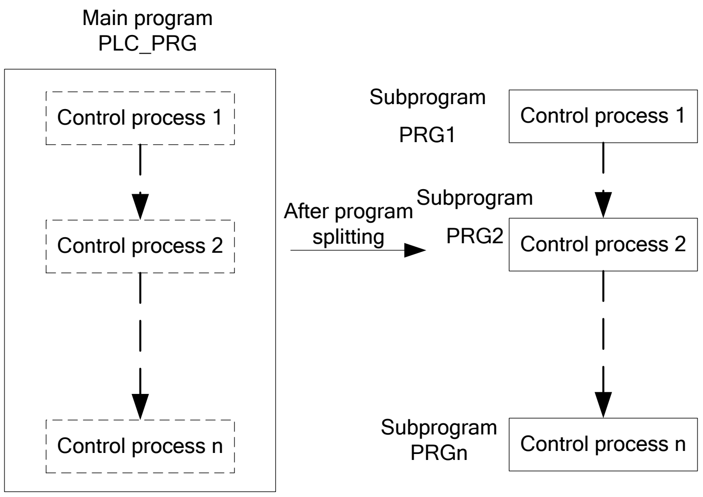
Messages - Total 0 error(s), 0 warning(s), 1 message(s)

0 error(s) | 0 warning(s) | 1 message(s)

1.6 Running of Multiple Subprograms

In actual projects, the program can usually be split into many subprograms by control flow or device object, based on which the designer can program each processing unit separately. As shown in Figure 1-15 below, the main program is split into multiple subprograms with different flows by control flow. The purpose of splitting is mainly to make the main program more organized and convenient for future debugging.

Figure 1-15 Splitting into Multiple Subprograms by Control Flow

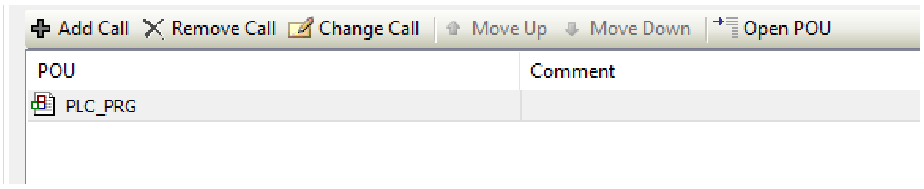


The right half of Figure 1-15 shows the subprograms PRG1, PRG2 ... PRGn classified by control flow, while the left half of Figure 1-15 shows the main program PLC_PRG, in which you can call subprograms PRG1 ... PRGn respectively. There are two methods to run multiple subprograms: the first method is to add subprograms in the Task Configuration; and the second method is to call subprograms in the main program, which is also common and flexible. The two methods are explained in detail below.

1. Add subprograms in the Task Configuration

Users can run multiple subprograms by adding subprograms in the Task Configuration page. Click “Add Call” to add subprograms in the sequence in which they are executed. As shown in Figure 1-16, after adding subprograms, the corresponding tasks will be executed cyclically in a top-down sequence specified by users, and the sequence can also be manually edited through the “Move Up” and “Move Down” functions.

Figure 1-16 Add Subprograms in the Task Configuration



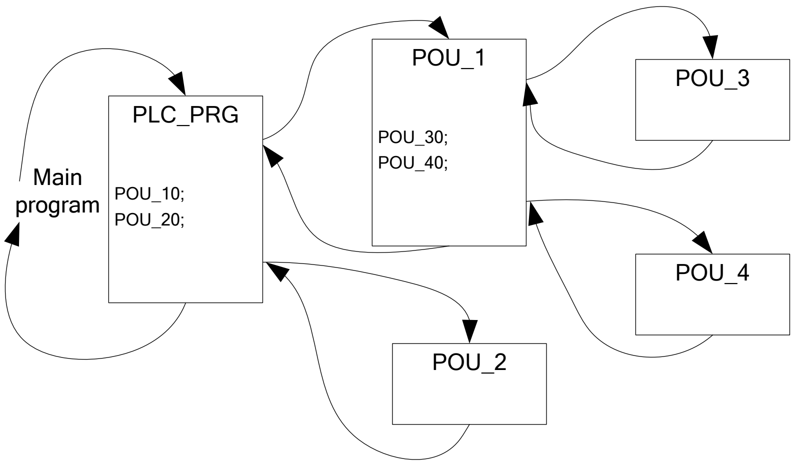
POU	Comment
PLC_PRG	

2. Calling subprograms in the main program PLC_PRG

PLC_PRG is defaulted as the main program by the system, which can be understood as a car's battery in a sense. When the car is produced, its various components are assembled, which is equivalent to the writing of subprograms; when the car is fully assembled, it is necessary to check whether it is usable. If you want to start the car, you must use the battery to start its various components, such as the engine and headlamps. The battery is equivalent to the entry point for starting the car. By calling subprograms in this way, the operability is enhanced and the program runs more flexibly. In addition, judgment statements can be added to the program, and nesting can be achieved.

PLC_PRG is a special POU that runs in the "freewheeling" mode by default. This POU is called every control cycle by default without any additional task configuration. Its configuration can also be found in the Task Configuration. Users can use it to call other subprograms, add necessary condition options when calling subprograms, or nest subprograms to make program calls more flexible. To implement the call relationship shown in Figure 1-17, you can write the following code in the main program PLC _PRG.

Figure 1-17 POU Calling Sequence



As shown in Figure 1-17, the main program is PLC_PRG, which uses the structured text programming language, and the program content is POU_1(); POU_2(). The main function of the above program is to call and execute the POU_1 and POU_2 subprograms respectively. Since POU_1 calls POU_3 and POU_4 respectively, the PLC actually executes the program in the following sequence:

A. The PLC first executes the subprogram POU_1.

B. Since POU_1 calls POU_3 and POU_4 in sequence, POU_3 is executed first.

C. POU_4 is executed, and POU_1 is finished.

D. POU_2 is finally executed to complete a full task cycle.

The above steps A to D are repeated as the execution sequence inside the PLC.

1.7 Single Axis Control

1.7.1 Programming Instructions for Single Axis Control

The motion control of the controller and the servo axis (such as DA200) is realized based on the EtherCAT bus network. Each EtherCAT bus cycle will perform an operation and issue a control instruction to control the servo. Different from the previous pulse control method, the EtherCAT bus is completely implemented by software. Attention should be paid to the following points during application:

- MC-related POUs should be configured to execute under the EtherCAT task. Most MC function blocks cannot run normally if they are placed in the POU of a low-priority Main task.
- The PDO configuration table needs to be configured with relevant data objects; otherwise, the servo will be unable to move due to the missing communication data object configuration, and no error alarm will be generated in this case, making it more difficult to troubleshoot.
- The controller can set the parameters of the servo by configuring SDO.
- An MC function block instance can only be used for the control of a unique servo axis; otherwise, an error may occur if it is used for the control of multiple servo axes.
- An MC function block must be used to monitor the running servo axis to avoid any error caused by program logic jump without MC function block monitoring. Such error is usually difficult to troubleshoot.
- Attention should be paid to the safe processing of debugging and it is required to ensure that the signal configuration is consistent with the actual application. If the servo system uses an incremental encoder, it needs to return to zero before normal operation. For movements within a limited range (such as a lead screw), limit and safety protection signals should be set.

1.7.2 Commonly Used MC Function Blocks for Single Axis Control

An MC function block (FB) is also called an MC instruction. In fact, the object instance of an MC function block is used in the user program, and the servo axis is controlled by the MC object instance, for example:

```
MC_Power1: MC_Power; // Declare instance MC_Power1
MC_Power1 (Axis=Axis1,);
```

Single axis control is generally used for positioning control, that is, the servo motor drives the external mechanism to move to the specified position. Sometimes, the servo is required to run at a specified speed or torque. In single axis control, the following MC function blocks are commonly used:

Table 1-4 Common MC Function Blocks for Single Axis Control

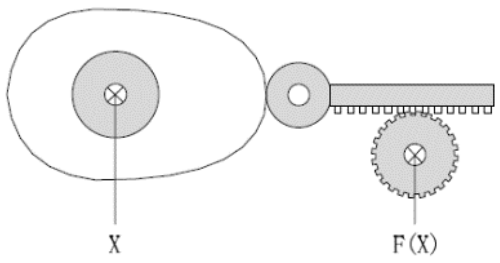
Control Action	MC Instructions To Be Used	Description
Servo enable	MC_Power	Run this instruction to enable the servo axis for subsequent operation control
Absolute position	MC_MoveAbsolute	Instruct the servo to move to the specified coordinate point
Relative position	MC_MoveRelative	Take the current position as reference and move to the specified distance
Servo jog	MC_Jog	Run the servo motor in a jog mode, which is often used for low-speed test runs to check the device or adjust the servo motor position
Relative superimposed position	MC_MoveAdditive	Based on the currently running instruction of the servo, move to the specified distance relatively
Speed control	MC_MoveVelocity	Instruct the servo to run at a specified speed

Control Action	MC Instructions To Be Used	Description
Servo halt	MC_Halt	Instruct the servo to halt running. If MC_Movexxx is triggered again, the servo can run again.
Emergency stop	MC_Stop	Instruct the servo to stop in an emergency. The servo can only run again after the stop instruction is reset and MC_Movexxx is triggered.
Alarm reset	MC_Reset	When the servo stops due to an alarm, run this instruction to reset it
Servo home	MC_Home	Instruct the servo to return to the home position. The home signal of the application system and the limit signals on both sides are connected to the DI port of the servo
Controller home	MC_Homing	The control system starts to return to the home position. The home signal of the application system and the limit signals on both sides are connected to the DI port of the controller

1.8 Cam Synchronization Control

An electronic cam (ECAM for short) is a software system that uses a constructed cam curve to simulate a mechanical cam to achieve the same relative motion between the camshaft and the master axis as in a mechanical cam system. Electronic cams can be used in various fields such as automobile manufacturing, metallurgy, mechanical processing, textiles, printing, and food packaging. An electronic cam curve is a function curve with the master axis pulse (active axis) input as X and the corresponding output of the servo motor (camshaft) as Y=F(X).

Figure 1-18 Electronic Cam



The electronic cam function of the PLC has the following features:

- Cam curves are easy to draw: Cams can be described by cam table, cam curve, or array, and multiple cam chart selections and dynamic switching during running are supported.
- Cam curves are easy to correct: The running cam table can be modified dynamically.
- Support one master and multiple slaves: One master axis can have multiple slave axes corresponding to it.
- Cam tappet: multiple cam tappets and multiple setting intervals are allowed.
- Cam clutch: The user program can make it enter and exit the cam running.
- Special functions: Virtual master axis, phase offset, and output superimposition are supported.

Note: The so-called “online cam curve modification” refers to the modification of the key point coordinates of the cam curve according to the needs of control characteristics during the execution of the program written by users. The key point coordinates are generally modified, but you can also modify the number of key points, the distance range of the master axis, etc.

The electronic cam of the PLC has three control elements:

1. Master axis: Reference axis for synchronous control.

2. Slave axis: a servo axis that follows the movement of the master axis according to the non-linear characteristics.
3. Cam table: Data table or cam curve describing the relative position, range, cyclicity of the master-slave axes.

Commonly used function blocks related to the electronic cam are listed in the following table:

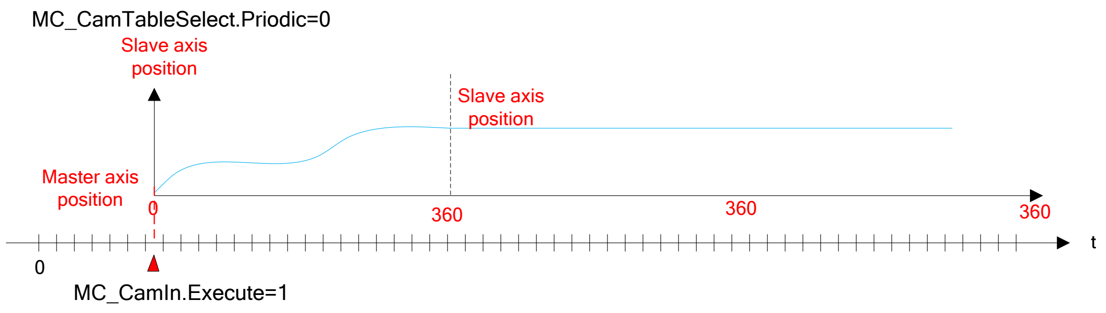
Table 1-5 Common Function Blocks of Electronic Cam

MC Instruction	Description
MC_CamTableSelect	Run this instruction to associate the relationship between the master axis, the slave axis, and the cam table.
MC_CamIn	Instruct the slave axis to enter cam operation
MC_CamOut	Instruct the slave axis to exit cam operation
MC_Phasing	Modify the phase of the master axis

1.8.1 Cyclic Mode of the Cam Table

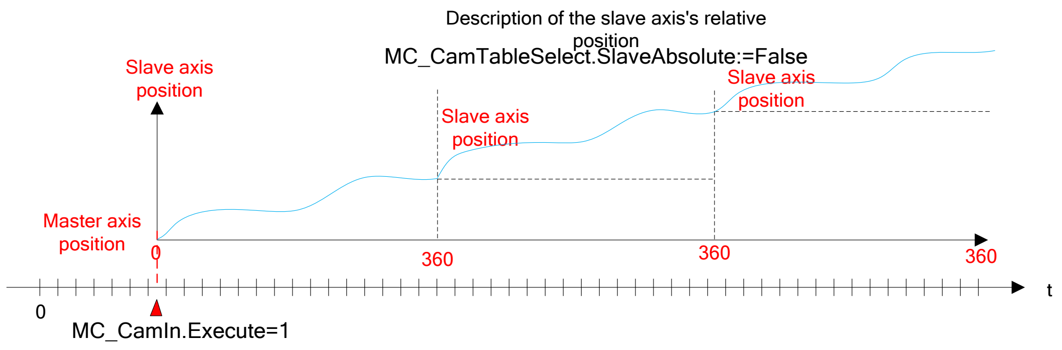
Single-cycle mode (Periodic:=0): After the cam table cycle is completed, the slave axis will exit the cam running state, as shown in Figure 1-19.

Figure 1-19 Single-Cycle Mode



Cyclic mode (Periodic:=1): After the cam table cycle is completed, the slave axis starts the next cam cycle until the user program instructions it to exit the cam running state, as shown in Figure 1-20.

Figure 1-20 Cyclic Mode



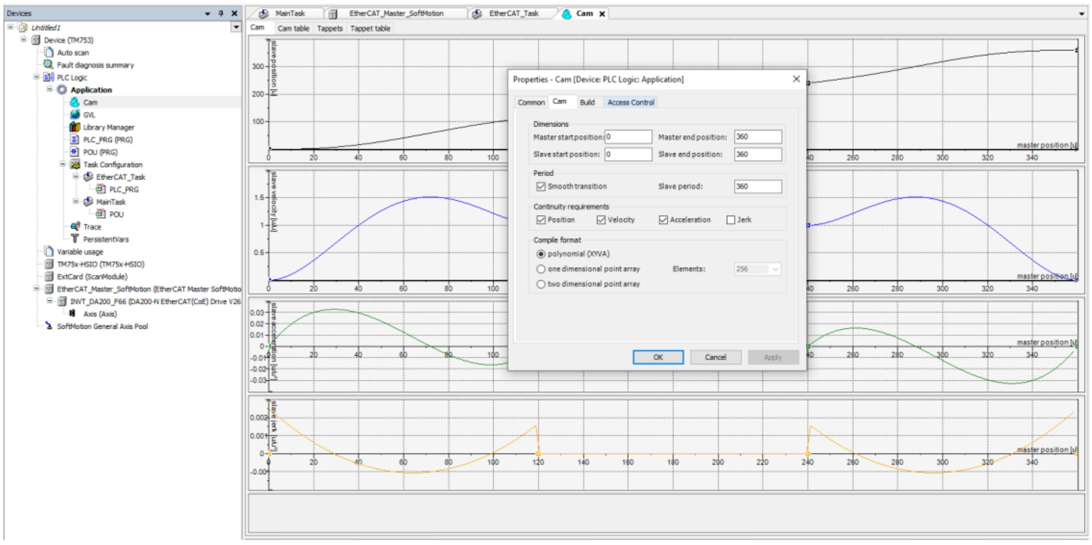
1.8.2 Input Method of the Cam Table

When a new cam table is created, the system will automatically generate the simplest cam curve, and you can edit it to form his or her own cam curve table.

You can increase or decrease the number of key points on the cam curve or change the coordinates of the key points.

The line pattern between two key points on the cam curve can be set to a straight line or a quintic polynomial, and the system will optimize each curve to minimize sudden changes in speed and acceleration.

Figure 1-21 Cam Curve



Properties - Cam [Device: PLC Logic: Application]

Common | Cam | Build | Access Control

Dimensions
Master start position: 0 Master end position: 360
Slave start position: 0 Slave end position: 360

Period
[✓] Smooth transition Slave period: 360

Continuity requirements
[✓] Position [✓] Velocity [✓] Acceleration [ ] Jerk

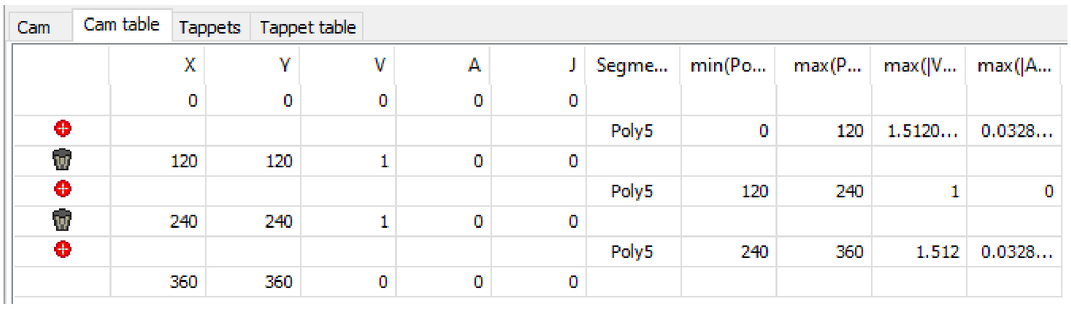
Compile format
[✓ polynomial (XYVA)]
one dimensional point array Elements: 256
two dimensional point array

OK Cancel Apply

1.8.3 Data Structure of the Cam Table

In Invtmatic Studio, for each cam table, there is a data structure and characteristic data describing the cam table. The figure below shows the data structure of the “CAM0” cam table. Please pay attention to the names of the variables in its structure.

Figure 1-22 Data Structure of the Cam Table



Cam | Cam table | Tappets | Tappet table

	X	Y	V	A	J	Segme...	min(Po...	max(P...	max(|V...	max(|A...
	0	0	0	0	0					
⊕						Poly5	0	120	1.5120...	0.0328...
🗑	120	120	1	0	0					
⊕						Poly5	120	240	1	0
🗑	240	240	1	0	0					
⊕						Poly5	240	360	1.512	0.0328...
	360	360	0	0	0					

There is a data structure inside Invtmatic Studio to describe the characteristics of the cam table. We can also manually write a cam table or modify the characteristic data of the cam through data structure access operations.

Note: When we declare the CAM0 cam table, the system automatically declares the CAM0 data structure of the global variable type by default, and also declares the CAM0_A[i] array at the same time.

For example, modify the number or coordinates of key points in the CAM0 cam table in the user program:

CAM0.nElements:=10; Modify the number of key points to 10

CAM0.xEnd:=300; Modify the end point of the master axis to 300

For example, modify the coordinates of two key points in the user program:

```
CAM0_A[2].dx:=10
CAM0_A[2].dy:=30
CAM0_A[2].dv:=1
CAM0_A[2].da:=0
CAM0_A[3].dx:=30
CAM0_A[3].dy:=50
CAM0_A[3].dv:=1
CAM0_A[3].da:=0
```

1.8.4 Reference and Switching of Cam Tables

The cam table is stored in an array inside the controller and can be referred to by a specific MC_CAM_REF variable type. For example, to declare:

```
Cam table q: MC_CAM_REF;
```

You can assign a value to this variable, or you can consider it as referring to a specific cam table:

```
Cam table q:=Cam0; // Refer to the required cam table
Cam table q: MC_CAM_REF; // Cam table pointer;
TableID: uint; // Cam table selection instruction, which can be set on the HMI;
CaseTableIDof
0: Cam table q:=Cam table A;
1: Cam table q:=Cam table B;
2: Cam table q:=Cam table C;
End_case
MC_CamTableSelect_0(// Cam relationship
Master:=Virtual master axis
Slave:=Cam slave axis
CamTable:=Cam table q
Execute:=bSelect, // Cam table selection is triggered at a rising edge
Periodic:=TRUE,
MasterAbsolute:=FALSE,
SlaveAbsolute:=FALSE);
```

The above routine uses the assignment operation of the MC_CAM_REF variable to realize the switching of multiple cam tables.

1.9 Programming Suggestions

In CODESYS, you can set the priority of a task, with a total of 32 levels (a number between 0 and 31, with 0 representing the highest priority and 31 representing the lowest priority). When a program is being executed, high-priority tasks take precedence over low-priority tasks. A task with the highest priority 0 can interrupt the execution of lower-priority programs in the same resource, causing the execution of the lower-priority programs to be slowed down. When assigning task priority levels, do not assign tasks with the same priority.

Note:

- For one task configuration, you can only set one priority, cycle type, and interval. If different execution characteristics are required, you need to add multiple task configurations.
- One task configuration can contain multiple POUs, which are executed in the sequence in which the POUs are added in the task.
- The task priority of EtherCAT bus communication is generally set to the highest priority 0, and the scan cycle is generally set to 1–4 ms. The smaller the set value, the higher the accuracy of motion control. When there are many axes, the scan cycle should be appropriately extended; otherwise, the CPU load rate will be high and axis loss may occur.

Task configuration – running status monitoring

After entering the online mode, you can use the system's built-in monitor to monitor task execution related parameters such as average, maximum, and minimum cycle time of a task. During the project development phase, this function can be used to test the maximum, minimum, and average cycle time of the program to determine the stability of the program and optimize the task cycle time set by the program.

In the task configuration, the following time setting relationship should be followed. This setting method can better optimize the task cycle and “watchdog” time to ensure the stability and real-time performance of the program.

“Watchdog” trigger time ＞ Cyclic time ＞ Maximum program cycle time

If the program cycle time is longer than the Cyclic time, the CPU will detect that the program has exceeded the count, which will affect the real-time performance of the program. If the program cycle time is longer than the watchdog trigger time, the CPU will detect a watchdog failure and stop program execution.

2 EtherCAT Operation Mechanism

2.1 EtherCAT Operation Principle

2.1.1 Introduction to the EtherCAT Protocol

EtherCAT (Ethernet for Control Automation Technology) is a technology that overcomes the inherent limitations of other Ethernet solutions and has the following key features:

Efficient data processing: Traditional Ethernet solutions require receiving data packets, decoding, and copying process data to each device, while EtherCAT slave devices can read data with corresponding addressing information as the message passes through its node, and insert input data as the message passes. This processing method results in a message delay of only a few nanoseconds.

Data transmission process: The frame sent from the master is transmitted and passes through all slaves to the last slave of the segment or branch. When the last device detects its open port, it returns the frame to the master. Since the sent and received Ethernet frame compresses a large amount of device data, the available data rate can reach over 90%, the 100Mb/s full-duplex feature is fully utilized, and the effective data rate can reach over 100 Mb/s.

Design of master and slave: The EtherCAT master uses a standard Ethernet media access controller (MAC) without the need for an additional communication processor, which means that any device controller with an integrated Ethernet interface can implement the EtherCAT master, regardless of the operating system or application environment. The EtherCAT slave uses an EtherCAT slave controller (ESC) to process data dynamically at a high speed. Network performance does not depend on the performance of the microprocessor used in the slave because all communications are completed in the ESC hardware. The process data interface (PDI) provides a dual-port random access memory (DPRAM) for the slave application layer to implement data exchange.

Precise synchronization: Precise synchronization is particularly important in distributed processes that require extensive synchronization actions, such as when multiple servo axes perform linked tasks simultaneously. Accurate calibration of distributed clocks is an effective solution to achieve synchronization. Compared to fully synchronous communication, distributed calibrated clocks are more tolerant to delay errors to some extent.

With these features, EtherCAT provides an efficient, flexible, and reliable industrial Ethernet solution suitable for various automation control applications.

2.1.2 Working Counter (WKC)

Each EtherCAT message ends with a 16-bit working counter (WKC). The WKC is a working counter used to record the number of read and write times for the EtherCAT slave device. The EtherCAT slave controller calculates the WKC in hardware, and the master checks the WKC in the sub-message after receiving the returned data. If it is not equal to the expected value, it means that the sub-message is not processed correctly. When a sub-message passes through a certain slave, the WKC will be increased by 1 if it is a single read or write operation. If it is a read/write operation, the WKC will be increased by 1 when the read operation is successful, by 2 when the write is successful, and by 3 when both are completed. The WKC is the accumulation of the processing results of each slave. The description of WKC increment is shown in Table 2-1.

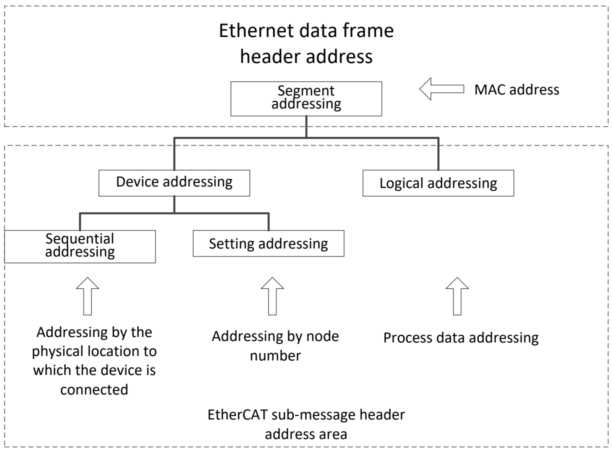
Table 2-1 WKC Increment

Instruction	Data Type	Increment
Read	Read failed	No change
	Read succeeded	+1
Write	Write failed	No change
	Write succeeded	+1
Read/write	Failed	No change
	Read succeeded	+1
	Write succeeded	+2
	Read/write succeeded	+3

2.1.3 Addressing Mode

EtherCAT communication is realized by the master sending EtherCAT data frames to read and write the internal storage area of the slave device. EtherCAT messages use multiple addressing modes to operate the internal storage area of the ESC for multiple communication services. EtherCAT addressing modes are shown in Figure 2-1. An EtherCAT segment is equivalent to an Ethernet device. The master first uses the MAC address of the Ethernet data frame header to address the segment, and then uses the 32-bit address in the EtherCAT sub-message header to address the device in the segment. There are two ways for in-segment addressing: device addressing and logical addressing. Device addressing performs read/write operations on a specific slave. Logical addressing is oriented towards process data and can realize multicast. The same sub-message can read/write multiple slave devices.

Figure 2-1 EtherCAT Network Addressing Modes


Ethernet data frame header address
Segment addressing
MAC address
Device addressing
Logical addressing
Sequential addressing
Setting addressing
Addressing by the physical location to which the device is connected
Addressing by node number
Process data addressing
EtherCAT sub-message header address area

2.1.3.1 Segment Addressing

Depending on the connection type of the EtherCAT master and its segment, the following two modes can be used to address a segment.

1. Direct mode

An EtherCAT segment is directly connected to a standard Ethernet port of the master device, as shown in Figure 2-2. In this case, the master uses the broadcast MAC address and the EtherCAT data frame is shown in Figure 2-3.

Figure 2-2 EtherCAT Segment in Direct Mode

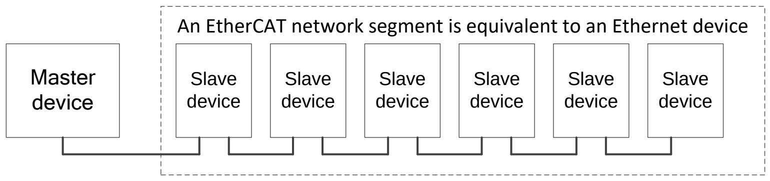


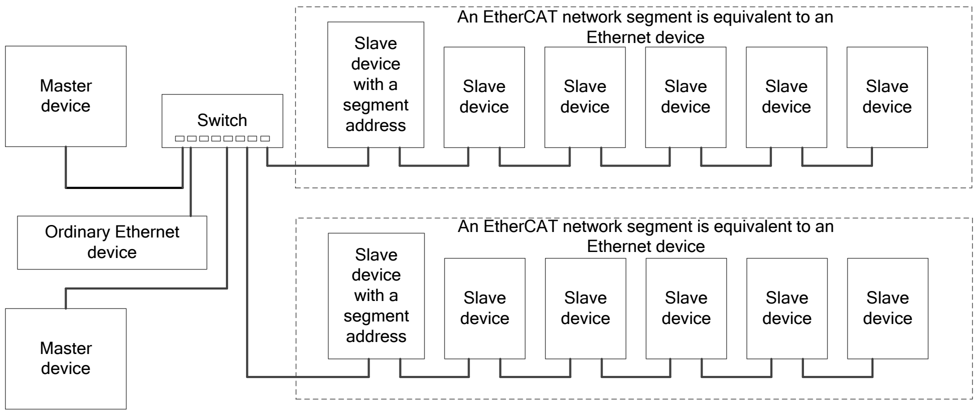
Figure 2-3 Addressed Content of EtherCAT Segment in Direct Mode

6 bytes	6 bytes	2 bytes	2 bytes	44 ~ 1498 bytes	4 bytes
Destination address: FF FF FF FF FF FF	Source address: FF FF FF FF FF FF	Frame type (0x88A4)	EtherCAT message header	EtherCAT data	PCS

2. Open Mode

An EtherCAT segment is connected to a standard Ethernet switch, as shown in Figure 2-4. In this case, the segment requires a MAC address, and the address in the EtherCAT data frame sent from the master is the MAC address of the segment it controls, as shown in Figure 2-5. The first slave device in the EtherCAT segment has an ISO/IEC 8802.3 MAC address, which represents the entire segment, and the slave is called a segment address slave, which can exchange the destination address area and source address area within the Ethernet. If EtherCAT data frame is sent over UDP, the device will also exchange the source and destination IP addresses and the source and destination UDP port numbers, making the response frame fully meet the UDP/IP protocol.

Figure 2-4 EtherCAT Segment in Open Mode



Master device
Switch
Ordinary Ethernet device
Master device
An EtherCAT network segment is equivalent to an Ethernet device
Slave device with a segment address
Slave device
Slave device
Slave device
Slave device
Slave device
An EtherCAT network segment is equivalent to an Ethernet device
Slave device with a segment address
Slave device
Slave device
Slave device
Slave device
Slave device

Figure 2-5 Addressed Content of EtherCAT Segment in Open Mode

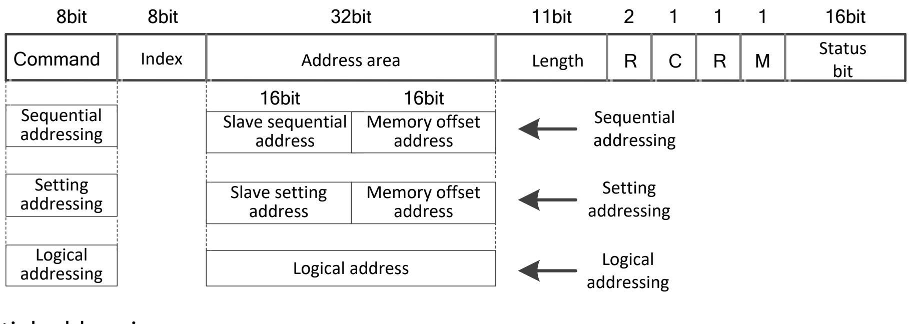
6 bytes	6 bytes	2 bytes	2 bytes	44 ~ 1498 bytes	4 bytes
Destination address: Segment MAC address	Source address: MAC address of master	Frame type (0x88A4)	EtherCAT message header	EtherCAT data	PCS

2.1.3.2 Device Addressing

During device addressing, the 32-bit address in the EtherCAT sub-message header is divided into a 16-bit slave device address and a 16-bit slave device internal physical storage space address, as shown in Figure 2-6. The 16-bit slave device address can address 65535 slave devices, each of which can have up to 64 local address spaces.

During device addressing, each message only addresses a unique slave device, but it has two different device addressing mechanisms (sequential addressing and set addressing).

Figure 2-6 EtherCAT Device Addressing Structure



1. Sequential addressing

For sequential addressing, the address of a slave is determined by its connection position in the segment, and a negative number is used to represent the position of each slave in the segment determined by the wiring sequence. When the sequential addressing sub-message passes through each slave device, its sequential address is increased by 1; and when the slave receives a message, the message with a sequential address of 0 is the message addressed to itself. Since this mechanism updates the device address as the message passes, it is also called “auto-increment addressing”.

As shown in Figure 2-7, there are 3 slave devices in the segment, and their sequentially addressed addresses are 0, -1, -2, and so on. When the master uses sequential addressing to access the slave, the address change of the sub-message is shown in Figure 2-8. The master sends 3 sub-messages to address 3 slaves, where the addresses are 0, -1, and -2 respectively, such as the data frame 1 in Figure 2-8. When the data frame reaches the slave ①, the slave ① checks that the address in the sub-message 1 is 0, thus knowing that the sub-message 1 is the message addressed to itself. After the data frame passes through the slave ①, all sequential addresses are increased by 1, called 1, 0, and -1, such as the data frame 2 in Figure 2-8. When it reaches the slave ②, the slave ② finds that the sequential address in the sub-message 2 is 0, which is its own message. Similarly, subsequent slaves are addressed in this way. As shown in Figure 2-7, in actual engineering applications, sequential addressing is mainly used in the startup phase, when the master configures addresses for each slave. Thereafter, the slave can be addressed using an address that is independent of its physical location. The sequential addressing mechanism can be used to set an address for the slave, as shown in Figure 2-8.

Figure 2-7 Sequentially Addressed Slave Address

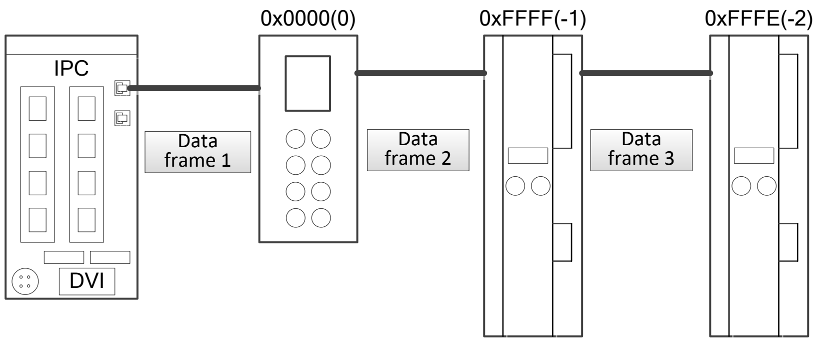


Figure 2-8 Change of Sub-message Address during Sequential Addressing

		Sub-message 1		Sub-message 2		Sub-message 3	
Data frame 1	… …	0	… …	0xFFFF (-1)	… …	0xFFFE (-2)	… …

The sequential address of the message sent by the master, that is, the address arriving at the slave ①

Data frame 2	… …	1	… …	0	… …	0xFFFF (-1)	… …

The sequential address of the message processed by the slave ①, that is, the address arriving at the slave ②

Data frame 3	… …	2	… …	1	… …	0	… …

The sequential address of the message processed by the master ②, that is, the address arriving at the slave ③

2. Set addressing

During set addressing, the addresses of slaves are independent of their consecutive sequence within the segment. As shown in Figure 2-9, the addresses can be configured by the master to slaves during the data link startup phase, or can be loaded by the configuration data of the slaves during the power-on initialization phase, and then the master uses sequential addressing to read the set address of each slave during the link startup phase. Its message structure is shown in Figure 2-10.

Figure 2-9 Slave Address in Set Addressing Mode

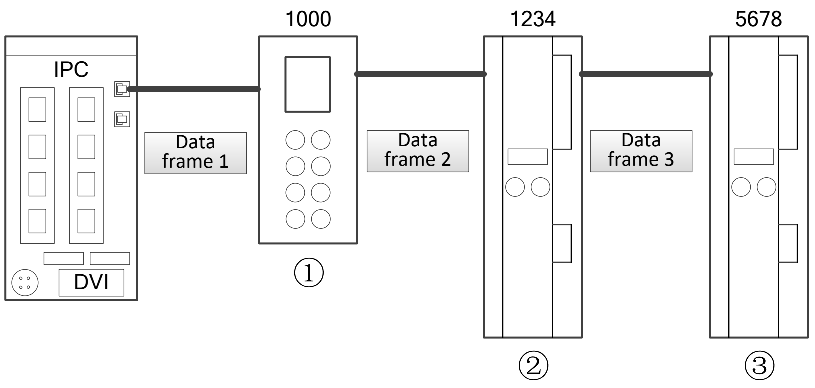


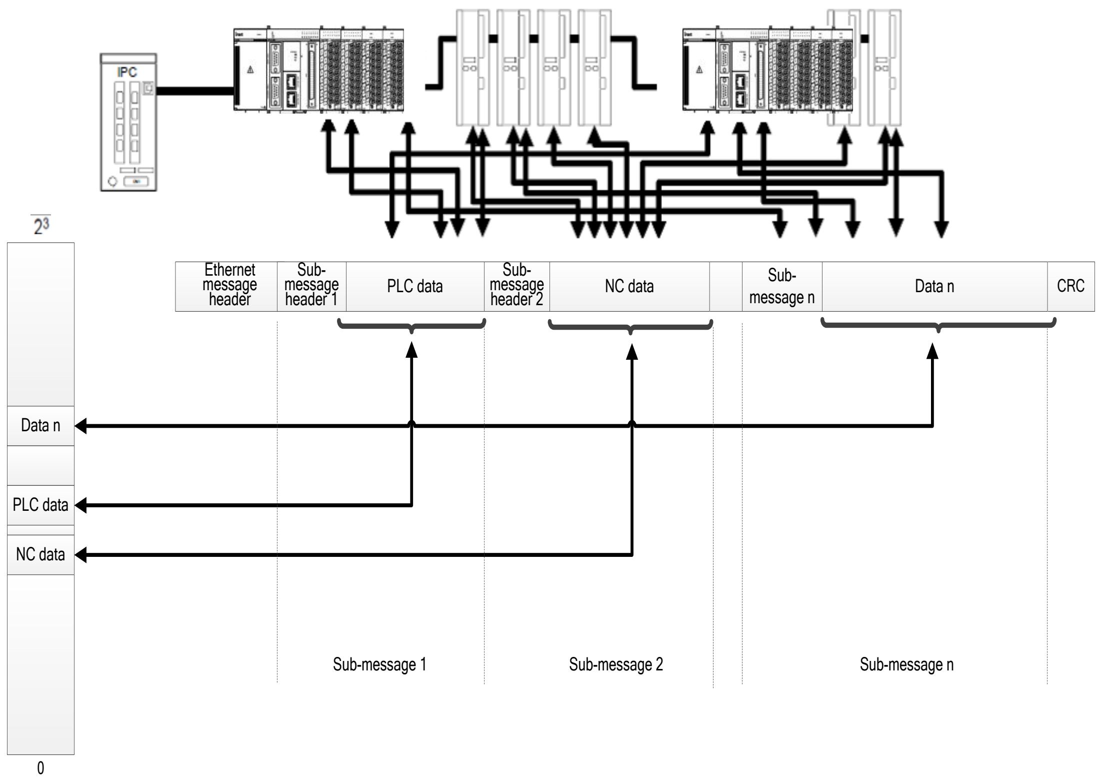
Figure 2-10 Message Structure in Set Addressing Mode

		Sub-message 1		Sub-message 2		Sub-message 3	
Data frame 1	… …	1000	… …	1234	… …	5678	… …

2.1.3.3 Logical Addressing

For logical addressing, the slave address is not defined separately, but using a section of the 4GB logical address space in the addressed segment. The 32-bit address area within the message is used as the logical address of the overall data to complete the logical addressing of the device. The logical addressing mode is implemented by the Fieldbus Memory Management Unit (FMMU). The FMMU function is located inside each ESC and maps the local physical storage address of a slave to the logical address in the segment. The schematic diagram is shown in Figure 2-11.

Figure 2-11 FMMU Operation Principle



When receiving an EtherCAT sub-message for logical addressing of data, the slave device will check for an FMMU unit address match. If so, it inserts the input type data into the corresponding position of the EtherCAT sub-message data area, and extracts the output type data from the corresponding position of the EtherCAT sub-message data area.

2.1.4 Distributed Clock

2.1.4.1 Distributed Clock Concept

Precise synchronization is particularly important for distributed processes that act simultaneously, for example, when several servo axes perform coordinated movements simultaneously. The distributed clock mechanism enables all slaves to be synchronized to a reference clock. The first distributed clock-capable slave connected to the master is used as the reference clock to synchronize the slave clocks of other devices and the master. In order to achieve precise clock synchronization control, data transmission delay and local clock offset must be measured and calculated, and the drift of the local clock must be compensated. The synchronous clock involves the following six concepts.

1. System time

System time is the system time used by the distributed clock. It starts from 0:00 on January 1, 2001, and is expressed as a 64-bit binary variable in nanoseconds (ns) and can be timed for up to 500 years. It can also be expressed as a 32-bit binary variable with a maximum of 4.2 s, which is usually used for communication and time stamping.

2. Reference clock and slave clock

The EtherCAT protocol defines the first distributed clock-capable slave connected to the master acts as the reference clock, and the clocks of other slaves are called slave clocks. The reference clock is used to synchronize the slave clocks of other slave devices and the master clock. The reference clock provides the EtherCAT system time.

3. Master clock

The EtherCAT master also has a timing function, which is called the master clock. The master clock can be synchronized as a slave clock in a distributed clock system. During the initialization phase, the master can send the master clock to the reference clock slave in the format of system time so that the distributed clock uses the system time for timing.

4. Local clock and its initial offset and clock drift

Each DC slave has a local clock, which runs independently and is timed using the local clock signal. When the system starts, there is a certain difference between the local clock of each slave and the reference clock, which is called initial clock offset. During operation, since the reference clock and the DC slave clock use their own clock sources, their timing cycles drift to a certain extent, which will cause the clocks to run asynchronously and the local clock to drift. Therefore, the initial clock offset and clock drift must be compensated.

5. Local system time

The local clock of each DC slave generates a local system time after compensation and synchronization. The distributed clock synchronization mechanism is to keep the local system time of each slave consistent. The reference clock is also the local system clock of the corresponding slave.

6. Transmission delay

There will be a certain delay when data frames are transmitted between slaves. It includes both internal device and physical connection delays. Therefore, when synchronizing slave clocks, the transmission delay between the reference clock and multiple slave clocks should be considered.

2.1.4.2 Clock Synchronization Process

Clock synchronization consists of the following three steps:

Step 1 Transmission delay measurement

When the distributed clock is initialized, the master will initialize the transmission delay for all slaves in all directions, calculate the deviation value between each slave clock and the reference clock, and write it to the slave clock.

Step 2 Reference clock offset compensation (system time)

The local clock of each slave will be compared with the system time, and then different comparison results will be written into different slaves so that all slaves will get the absolute system time.

Step 3 Reference clock drift compensation

Clock drift compensation and local time are used to periodically compensate for the error and fine-tune the local clock. The following diagrams illustrate two application cases of compensation operations: Figure 2-12 illustrates the case where the system time is less than the local clock of the slave, while Figure 2-13 illustrates the case where the system time is greater than the local time.

Figure 2-12 Clock Synchronization Process: System Time ＜ Local Time

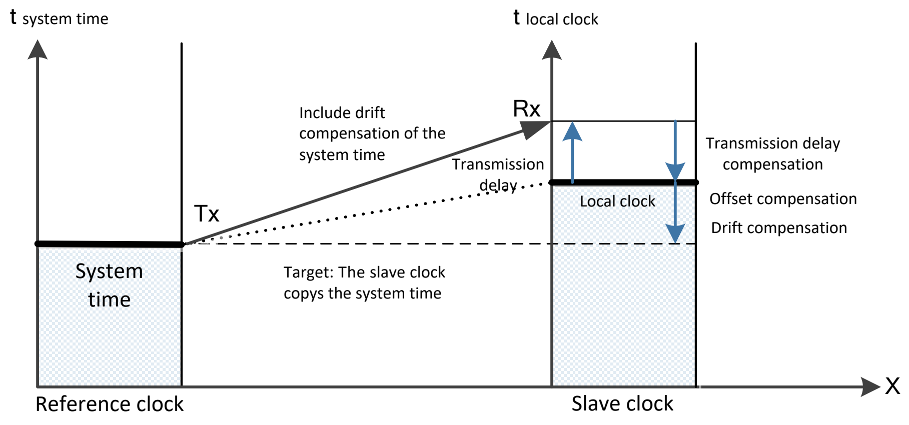
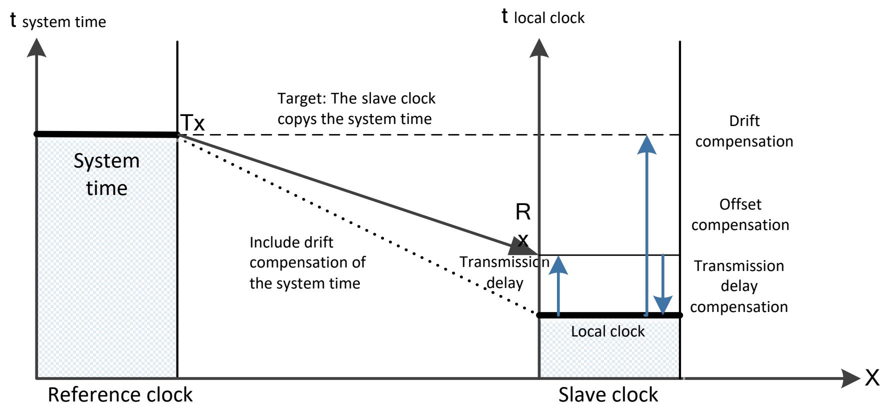
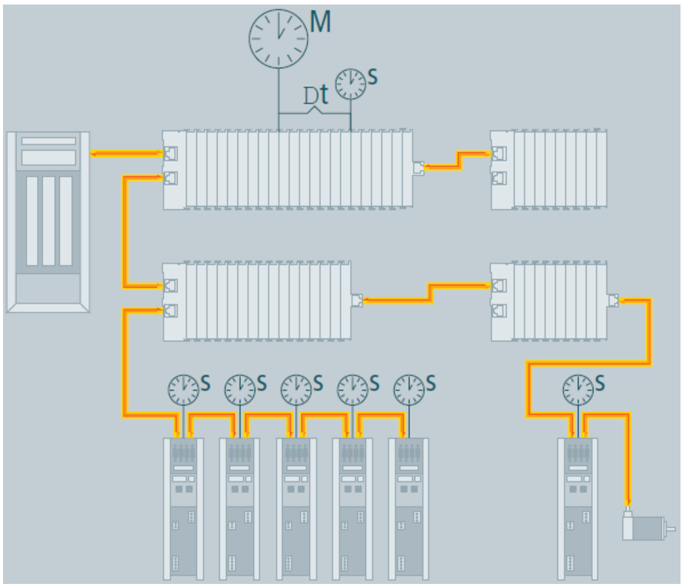


Figure 2-13 Clock Synchronization Process: System Time ＞ Local Time



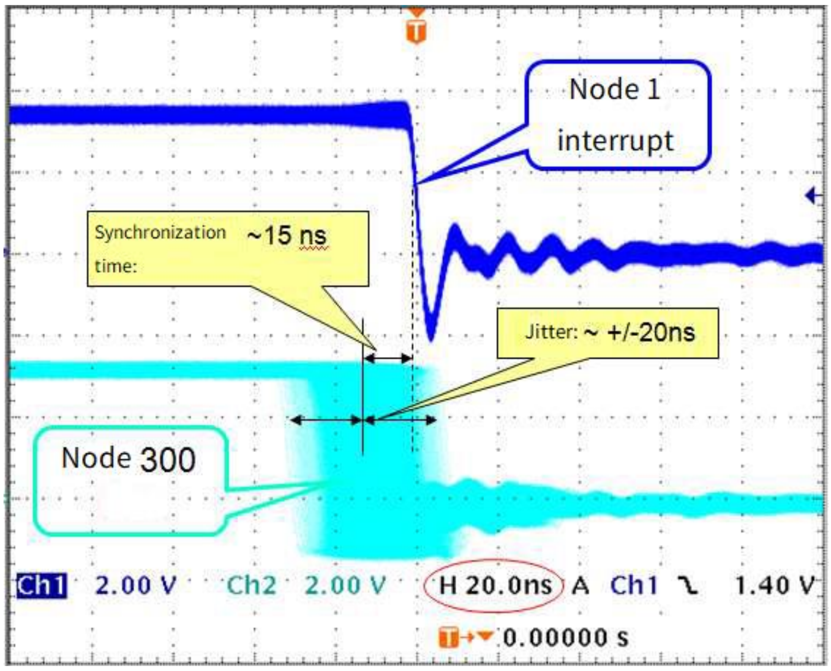
For EtherCAT, data exchange is based entirely on pure hardware mechanisms. Since a logical ring structure is used for communication (with the help of the physical layer of full-duplex Fast Ethernet), the master clock can simply and accurately determine the delay offset propagated by each slave clock, and vice versa. Distributed clocks are all adjusted based on this value, which means that a very precise, deterministic synchronization error time base of less than 1 μs can be used across the network. Its structure is shown in Figure 2-14.

Figure 2-14 Principle of Synchronous Clock



For example, the difference between two devices is 300 nodes, the cable length is 120 m, and the communication signal is captured using an oscilloscope. The result is shown in Figure 2-15.

Figure 2-15 Synchronous Clock Performance Test



This function is very important for motion control. It calculates the speed through the continuously detected position values. When the sampling time is very short, even a small instantaneous jitter in the position measurement will cause a large step change in the speed calculation. In EtherCAT, the introduction of time-stamped data types as a logical extension allows high-resolution system time to be added to the measured value, which is made possible by the huge bandwidth that Ethernet provides.

2.2 EtherCAT Communication Mode

In actual automation control systems, there are usually two forms of data exchange between applications: time-critical and non-time-critical.

The time-critical form indicates that a specific action must be completed within a certain time window. If communication cannot be completed within the required time window, control failure may occur.

Time-critical data is usually sent cyclically, which is called cyclic process data communication. Non-time-critical data can be sent acyclically, and acyclic mailbox data communication is adopted in EtherCAT.

2.2.1 Cyclic Process Data Communication

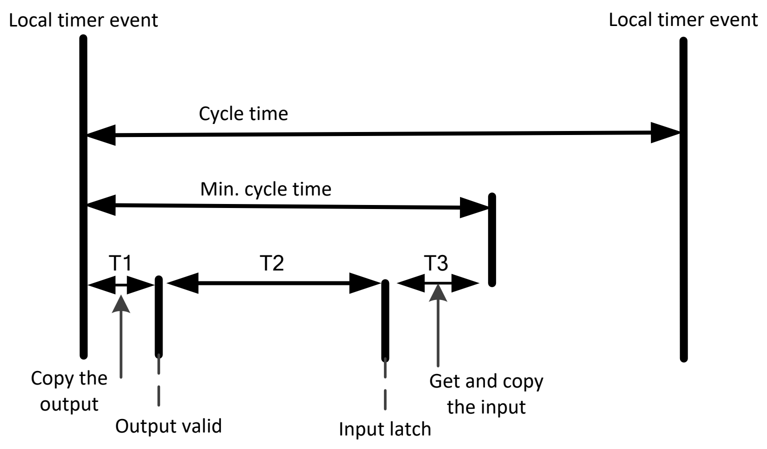
The master can use logical read, write, or read/write instructions to control multiple slaves at the same time. In the cyclic data communication mode, the master and the slave have multiple synchronous operation modes.

2.2.1.1 Slave Device Synchronization Mode

1. Freewheeling mode

In the freewheeling mode, the local control cycle is generated by a local timer interrupt. The cycle time can be set by the master and is an optional feature for the slave. The local cycle in the freewheeling mode is shown in Figure 2-16. Where T1 is the time it takes for the local microprocessor to copy data from the EtherCAT slave controller and calculate the output data; T2 is the output hardware delay; and T3 is the input latch offset time. These parameters reflect the time response performance of the slave.

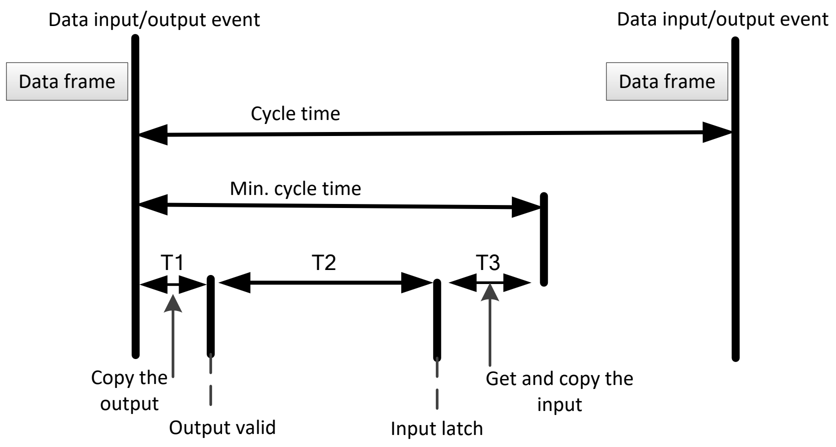
Figure 2-16 Local Cycle in Freewheeling Mode



2. Synchronization with data input and output events

The local cycle is triggered when a data input or output event occurs, as shown in Figure 2-17. The master can write the sending cycle of the process data frame to the slave, while the slave can check whether it supports this cycle time or optimize the cycle time locally. The slave can also choose to support this function. It is usually synchronized with data output events. If the slave only has input data, the data is synchronized with input events.

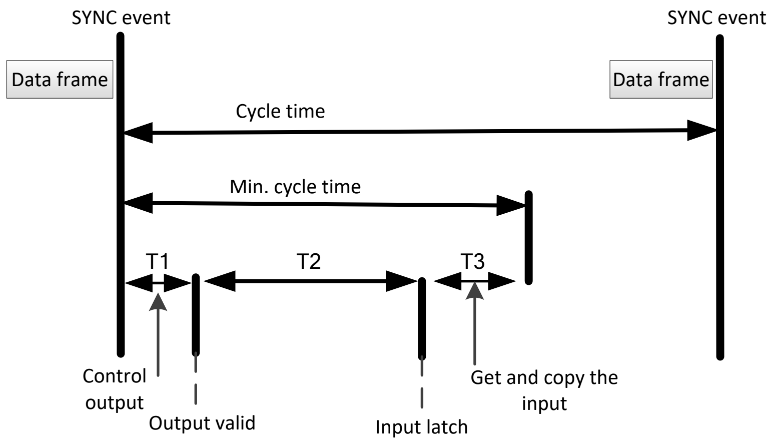
Figure 2-17 Local Cycle Synchronization with Data Input and Output Events



3. Synchronization with distributed clock synchronization events

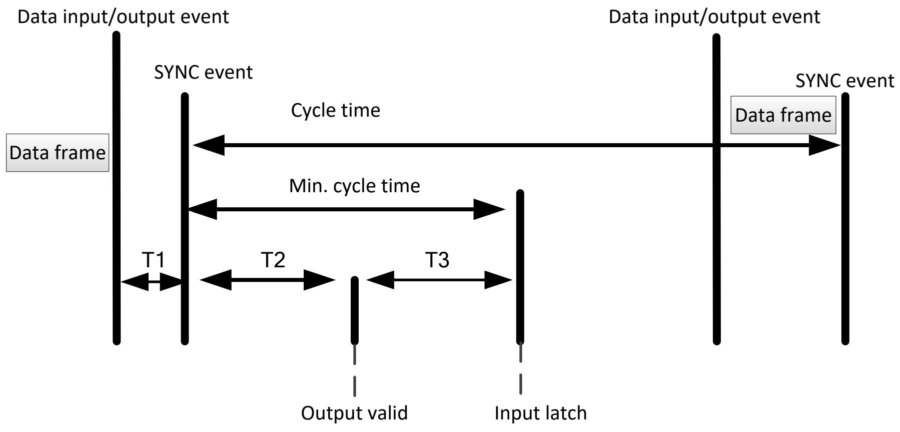
The local cycle is triggered by an SYNC event, as shown in Figure 2-18. The master must complete the transmission of the data frame before the SYNC event, so the master clock must also be synchronized with the reference clock.

Figure 2-18 Local Cycle Synchronization with SYNC Events



To further optimize slave synchronization performance, the master should copy the output information from the received process data frame when a data sending and receiving event occurs, and then wait for the SYNC signal to arrive before continuing local operations. As shown in Figure 2-19, the data frame must arrive T1 ahead of the SYNC signal. The slave has completed data exchange and control operation before the SYNC event, and can immediately perform output operations after receiving the SYNC signal, thereby further improving synchronization performance.

Figure 2-19 Optimized Local Cycle Synchronization with SYNC Events



2.2.1.2 Master Device Synchronization Mode

1. Cyclic mode

In the cyclic mode, the master sends process data frames cyclically. The master cycle is usually controlled by a local timer. The slave can run in the freewheeling mode or in synchronization with received data events. For the slave running in synchronization, the master should check the cycle time of the corresponding process data frame to ensure that it is greater than the minimum cycle time supported by the slave.

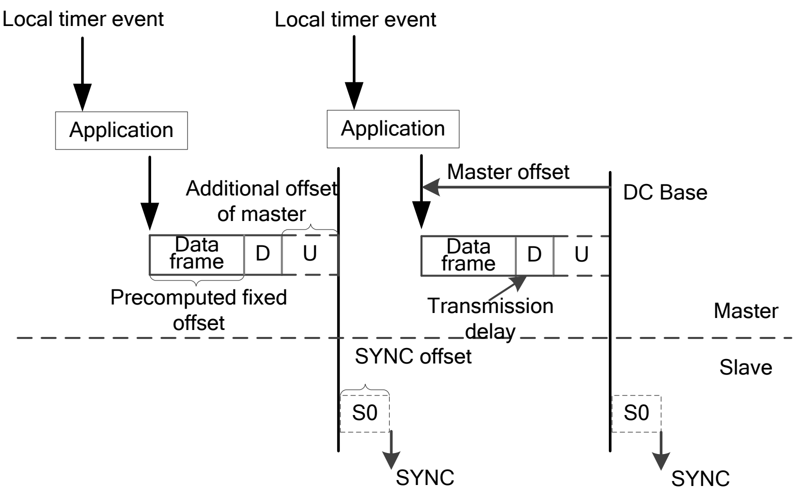
The master can send multiple cyclic process data frames with different cycle times in order to obtain the optimal bandwidth. For example, a shorter cycle is used to send motion control data and a longer cycle is used to send I/O data.

2. DC mode

The master runs in the DC mode similarly to the cyclic mode, except that the local cycle of the master should be synchronized with the reference clock. The master's local timer should be adjusted based on the ARMW message that publishes the reference clock. During operation, after the ARMW message used to dynamically compensate for clock drift is returned to the master, the master clock can be adjusted based on the reference clock time read back to be roughly synchronized with the latter.

In the DC mode, all DC-enabled slaves should be synchronized with the DC system time. The master should also synchronize other communication cycles with the DC reference clock time. The working principle of synchronizing the local cycle with the DC reference clock is shown in Figure 2-20.

Figure 2-20 Master DC Mode



The local operation of the master is started by a local timer. The local timer should have an advance over the DC reference clock timing, which is the sum of the following times.

- Control program execution time
- Data frame transmission time
- Data frame transmission delay D
- Additional offset U (related to the jitter of each slave delay time and the jitter of the control program execution time, used for adjusting the master cycle)

2.2.2 Acyclic Mailbox Data Communication

Acyclic data communication in the EtherCAT protocol is called mailbox data communication, which can be carried out in both directions - master-to-slave and slave-to-master. It supports full-duplex, bi-directional independent communication and multi-user protocols. Slave-to-slave communication is managed by the master acting as a router. The mailbox communication data header includes an address field so that the master can resend the mailbox data. Mailbox data communication is a standard way to implement parameter exchange and is required if cyclic process data communication needs to be configured or other acyclic services are needed.

The mailbox data message structure is shown in Figure 2-21. Usually the mailbox communication value corresponds to one slave, so the device addressing mode is used in the message. The data elements in its data header are explained in Table 2-2.

Figure 2-21 Mailbox Data Unit Structure

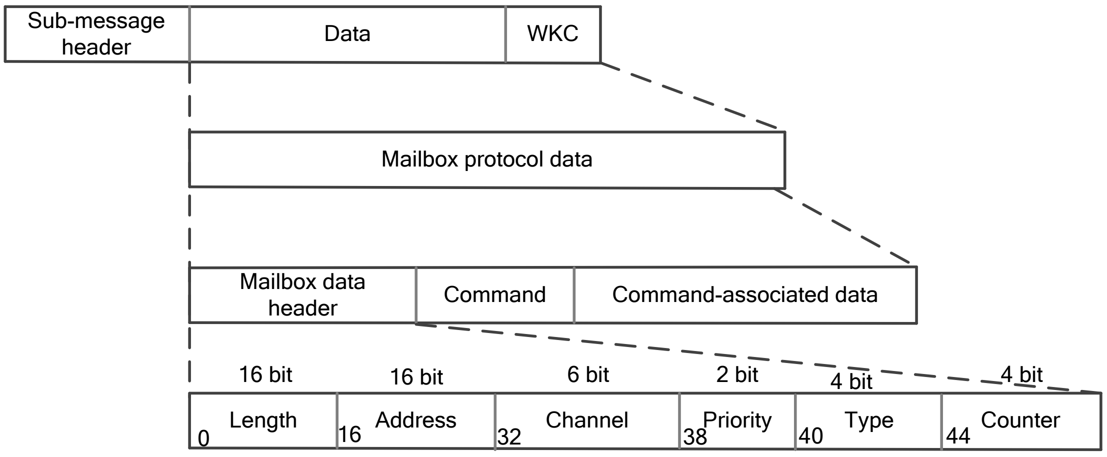


Table 2-2 Mailbox Data Header

Data Element	Number of Digits	Description
Length	16	The length of the followed mailbox service data
Address	16	The slave address of the data source in the case of master-to-slave communication The slave address of the data destination in the case of slave-to-slave communication
Channel	6	Reserved
Priority	2	Reserved
Type	4	Mailbox type, that is, the subsequent protocol type: 0: Mailbox communication error 2: EoE (Ethernet over EtherCAT) 3: CoE (CANopen over EtherCAT) 4: FoE (File Access over EtherCAT) 5: SoE (Sercos over EtherCAT) 15: VoE (Vendor Specific Profile over EtherCAT)
Counter Ctr	4	The sequential number used for duplicate detection, increasing by 1 for each new mailbox service (For compatibility with older versions, only 1–7 are used)

- Master-to-slave communication – mailbox write instruction

The master sends the data area write instruction to send the mailbox data to the slave. The master needs to check the working counter WKC in the slave mailbox instruction response message. If the working counter is 1, it means the write instruction succeeded. On the contrary, if the working counter does not increase, it is usually because the slave did not finish reading the previous instruction or did not respond within the specified time, and the master must resend the mailbox data write instruction.

- Slave-to-master communication – mailbox read instruction

When the slave has data to send to the master, it must first write the data into the input mailbox buffer cache and then the data is read by the master. If there is valid data waiting to be sent from the slave ESC input mailbox data area, the master will send the appropriate read instruction to read the slave data as soon as possible. The master has two ways to determine whether the slave has filled the mailbox data into the input data area: one is to use FMMU to cyclically read a flag bit, and the flag bits of multiple slaves can be read through logical addressing, but the disadvantage lies in that a FMMU unit is required for each slave; the other is to simply poll the data area ofthe ESC input mailbox. The working counter of the read instruction increases by 1, indicating that the slave has filled the new data into the input data area.

2.3 EtherCAT State Machine

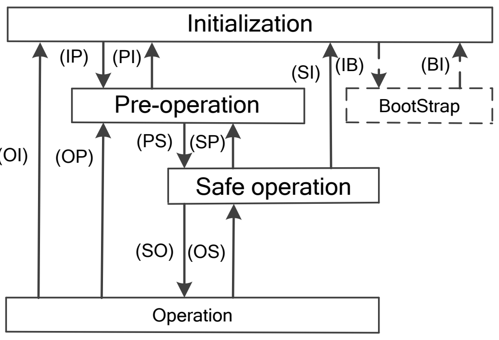
The EtherCAT State Machine (ESM) coordinates the states of the master and slave applications at initialization and runtime.

The EtherCAT device must support four states, in addition to an optional state.

- Init: Initial, abbreviated as I
- Pre-Op: Pre-operational, abbreviated as P
- Safe-Op: Safe-operational, abbreviated as S
- Op: Operational, abbreviated as O
- Boot-Strap: Boot state (optional), abbreviated as B

The transition relationship between the above states is shown in Figure 2-22. When the Init state transits to the Op state, the transition must be in the order of “Init → Pre-Op → Safe-Op → Op. Only when returning from the Op state can the state be skipped, and other states cannot be skipped. The Boot-Strap state is an optional state and can only transit to and from the Init state. All state changes are initiated by the master, which sends a state control instruction to the slave to request a new state. The slave responds to the instruction, executes the requested state transition, and writes the result to the slave state indication variable. If the requested state transition fails, the slave will raise an error flag.

Figure 2-22 EtherCAT State Transition Relationship



- Init: Initial

The Init state defines the initial communication relationship between the master and the slave at the application layer. At this time, the master and slave application layers cannot communicate directly, and the master uses the Init state to initialize some configuration registers of the ESC. If the master supports mailbox communication, the mailbox communication parameters are configured.

- Pre-OP: Pre-operational

In the Pre-Op state, mailbox communication is activated. The master and the slave can use mailbox communication to exchange initialization operations and parameters related to the application. Process data communication is not permitted in this state.

- Safe-Op: Safe-operational

In the Safe-Op state, the slave application reads input data but does not generate output signals. The device has no output and is in a “safe state”. At this time, mailbox communication is still possible.

- Op: Operational

In the Op state, the slave application reads data, the master application sends output data, and the slave device generates output signals. At this time, mailbox communication is still possible.

- Boot-Strap: Boot state (optional)

The function of the Boot-Strap state is to download the device firmware program. The master can use the mailbox communication of the FoE protocol to download a new firmware program to the slave.

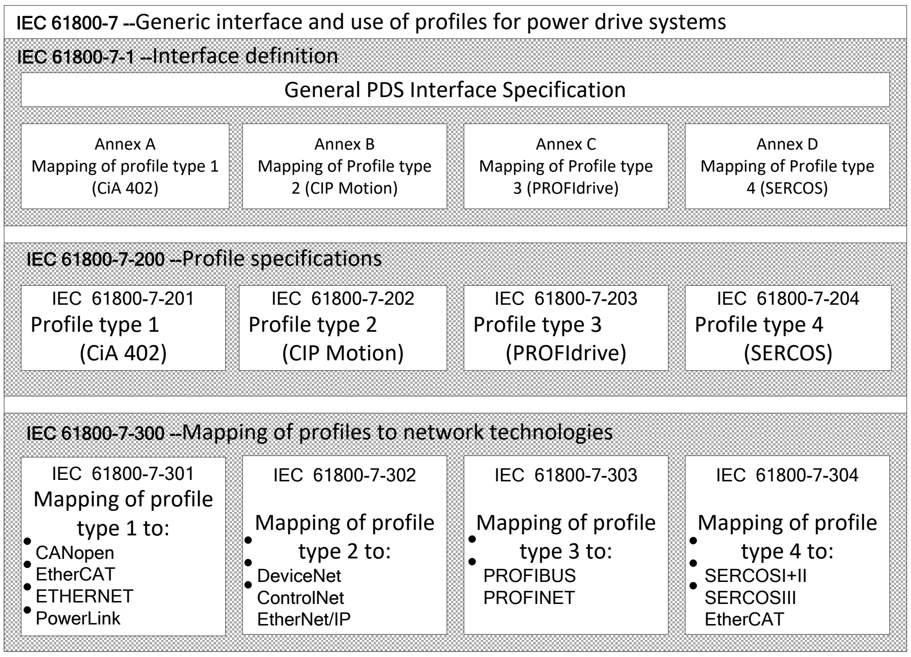
Table 2-3 EtherCAT State Machine Transition Summary

State and State Transition	Description
Init	There is no communication at the application layer, and the master can only read and write ESC registers
Init to Pre-OP (IP)	The master configures the slave address register If mailbox communication is supported, the mailbox channel parameters are configured; if distributed clocks are supported, DC related registers are configured The master writes the state control register to request the “Pre-Op” state
Pre-OP	Application layer mailbox data communication
Pre-Op to Safe-Op (PS)	The master uses the mailbox to initialize process data mapping The master configures the SM channel used for process data communication The master configures the FMMU The master writes the state control register to request the “Safe-Op” state
Safe-Op	The master sends valid output data The master writes the state control register to request the “Op” state
Op	All inputs and outputs are valid, and mailbox communication can still be used

2.4 EtherCAT Servo Drive Control Application Protocol

The IEC 61800 series of standards is a generic specification for adjustable speed electrical power drive systems. IEC 61800-7 defines the standard for the communication interface between the control system and the power drive system, including network communication technology and application profiles, as shown in Figure 2-23. As a network communication technology, EtherCAT supports the profile CiA402 in the CANopen protocol and the application layer of the SERCOS protocol, which are called CoE and SoE respectively.

Figure 2-23 IEC 61800-7 Architecture


IEC 61800-7 --Generic interface and use of profiles for power drive systems
IEC 61800-7-1 --Interface definition
General PDS Interface Specification
Annex A Mapping of profile type 1 (CiA 402)
Annex B Mapping of Profile type 2 (CIP Motion)
Annex C Mapping of Profile type 3 (PROFIdrive)
Annex D Mapping of Profile type 4 (SERCOS)
IEC 61800-7-200 --Profile specifications
IEC 61800-7-201 Profile type 1 (CiA 402)
IEC 61800-7-202 Profile type 2 (CIP Motion)
IEC 61800-7-203 Profile type 3 (PROFIdrive)
IEC 61800-7-204 Profile type 4 (SERCOS)
IEC 61800-7-300 --Mapping of profiles to network technologies
IEC 61800-7-301 Mapping of profile type 1 to: CANopen EtherCAT ETHERNET PowerLink
IEC 61800-7-302 Mapping of profile type 2 to: DeviceNet ControlNet EtherNet/IP
IEC 61800-7-303 Mapping of profile type 3 to: PROFIBUS PROFINET
IEC 61800-7-304 Mapping of profile type 4 to: SERCOSI+II SERCOSIII EtherCAT

2.4.1 EtherCAT-based CAN Application Protocol (CoE)

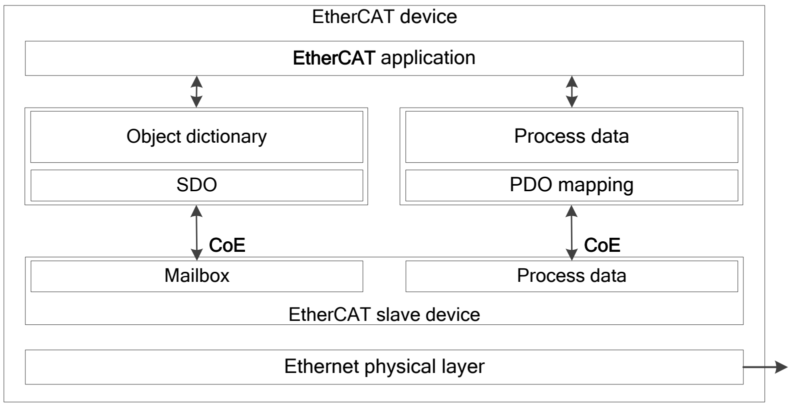
CANopen device and application profiles are used across a wide range of devices and applications, such as I/O components, drives, encoders, proportional valves, hydraulic controllers, as well as application profiles for the plastics or textile industries. EtherCAT can provide the same communication mechanism as the CANopen mechanism, including object dictionaries, PDOs (process data objects) and SDOs (service data objects), and even similar network management. EtherCAT can thus be implemented with minimum effort on devices equipped with CANopen, and large parts of the CANopen firmware can be reused. In addition, objects can be optionally extended to take advantage of the huge bandwidth resources provided by EtherCAT.

The EtherCAT protocol supports the CANopen protocol at the application layer and makes corresponding supplements, including the following main functions:

- Achieve network initialization by using mailbox communication to access CANopen object dictionaries and objects.
- Achieve network management by using CANopen application objects and optional time-driven PDO messages.
- Map process data by using object dictionaries and cyclically transmit instruction data and state data.

Figure 2-24 shows the CoE device structure whose communication modes mainly include cyclic process data communication and acyclic data communication. The differences between the two in practical applications will be explained below.

Figure 2-24 CoE Device Structure



2.4.1.1 CoE Object Dictionary

The CoE protocol fully complies with the CANopen protocol and has the same object dictionary definition, as shown in Table 2-4. Table 2-5 lists the CoE communication data objects, which extend the relevant communication objects 0x1C00–0x1C4F for EtherCAT communication to set the type of storage synchronization manager, communication parameters, and PDO data allocation.

Table 2-4 CoE Object Dictionary Definition

Index Number Range	Description
0x0000–0x0FFF	Data type description
0x1000–0x1FFF	Communication objects include: device type, identifier, PDO mapping, CANopen-compatible CANopen-specific data objects, and EtherCAT extended data objects reserved in EtherCAT
0x2000–0x5FFF	Manufacturer-defined objects
0x6000–0x9FFF	Profile-defined data objects
0xA000–0xFFFF	Reserved

Table 2-5 CoE Communication Data Objects

Index	Description
0x1000	Device type
0x1001	Error register

Index	Description
0x1008	Equipment manufacturer and equipment name
0x1009	Manufacturer hardware version
0x100A	Manufacturer software version
0x1018	Device identifier
0x1600–0x17FF	RxPDO mapping
0x1A00–0x1BFF	TxPDO mapping
0x1C00	Synchronization manager communication type
0x0x1C10–0x1C2F	Process data communication synchronization manager PDO assignment
0x0x1C30–0x1C4F	Synchronization management parameters

2.4.1.2 CoE Cyclic Process Data Communication (PDO)

In cyclic data communication, the process data can contain multiple PDO mapping data objects. The data objects 0x1C10–0x1C2F used by the CoE protocol define the corresponding PDO mapping channels. Table 2-6 shows the specific structure of the communication data in the EtherCAT protocol.

Table 2-6 CoE Communication Data Objects

Index	Object Type	Description	Type
0x1C10	Array	SM0 PDO assignment	Unsigned 16-bit integer
0x1C11	Array	SM1 PDO assignment	Unsigned 16-bit integer
0x1C12	Array	SM2 PDO assignment	Unsigned 16-bit integer
0x1C13	Array	SM3 PDO assignment	Unsigned 16-bit integer
…	…	…	…
0x1C2F	Array	SM31 PDO assignment	Unsigned 16-bit integer

An SM2 PDO assignment example (0x1C12) is given below. Table 2-7 lists examples of its values. For example, two data are mapped in PDO0. The first communication variable is the control word, and the corresponding mapped index and sub-index address are 0x6040:00; the second communication variable is the target position value, and the corresponding mapped index and sub-index address are 0x607A:00.

Table 2-7 Example of SM2 Channel PDO Assignment Object Data 0x1C12SM2

0X1C12 Sub-index	Value	PDO Data Object Mapping			
		Sub-index	Value	Number of Bytes	Description
0	3	-	-	1	Number of PDO mapping objects
1	PDO0 0x1600	0	2	1	Number of data mapping objects
		1	0x6040:00	2	Control word
		2	0x607A:00	4	Target position
1	PDO1 0x1601	0	2	1	Number of data mapping objects
		1	0x6071:00	2	Target torque
		2	0x6087:00	4	Target slope
1	PDO2 0x1602	0	2	1	Number of data mapping objects
		1	0x6073:00	2	Maximum current
		2	0x6075:00	4	Motor rated current

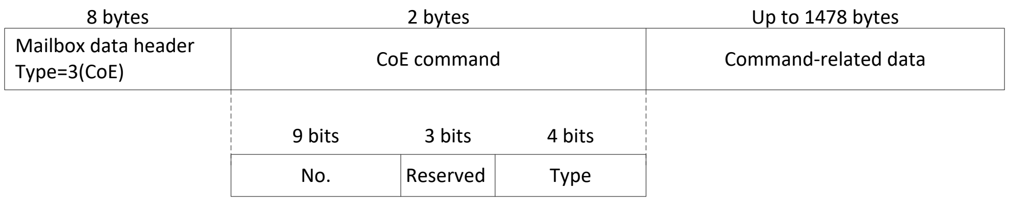
There are several ways for PDO mapping:

1. Simple devices do not require a mapping protocol: simple process data is used and read from the slave EEPROM.
2. Read PDO mapping: fixed process data mapping; read using SDO communication.
3. Optional PDO mapping: multiple fixed groups of PDOs are selected through the object 0x1C1X; read through SDO communication.
4. Variable PDO mapping: configured through CoE communication.

2.4.1.3 CoE Acyclic Process Data Communication (SDO)

The EtherCAT master realizes acyclic data communication by reading and writing mailbox data SM channels. The CoE protocol mailbox data structure is shown in Figure 2-25.

Figure 2-25 CoE Data Header



The number in Figure 2-25 is explained in detail in Table 2-8.

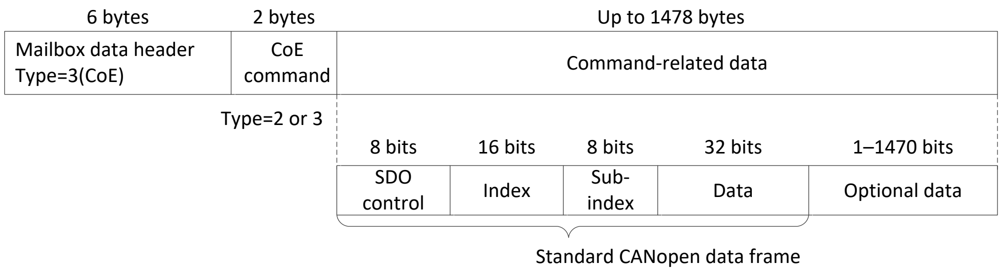
Table 2-8 Definitions of CoE Instructions

Number of CoE Instruction Field	Description
Number	Number when PDO is sent
Type	Message type: 0: Reserved 1: Emergency message 2: SDO request 3: SDO response 4: TxPDO 5: RxPDO 6: Remote transmission request of a TxPDO 7: Remote transmission request of a RxPDO 8: SDO message 9–15: Reserved

- SDO service

CoE communication service types 2 and 3 are SDO communication services, and the SDO data structure is shown in Figure 2-26.

Figure 2-26 SDO Data Frame Format



SDO is usually divided into the following three types according to the transmission method. Table 2-9 lists the specific content of the SDO data frame, and the results are shown in Figure 2-27

1. Fast transmission service: As with the standard CANopen protocol, only 8 bytes are used and up to 4 bytes of valid data can be transmitted.
2. Regular transmission service: More than 8 bytes can be used to transmit more than 4 bytes of valid data.

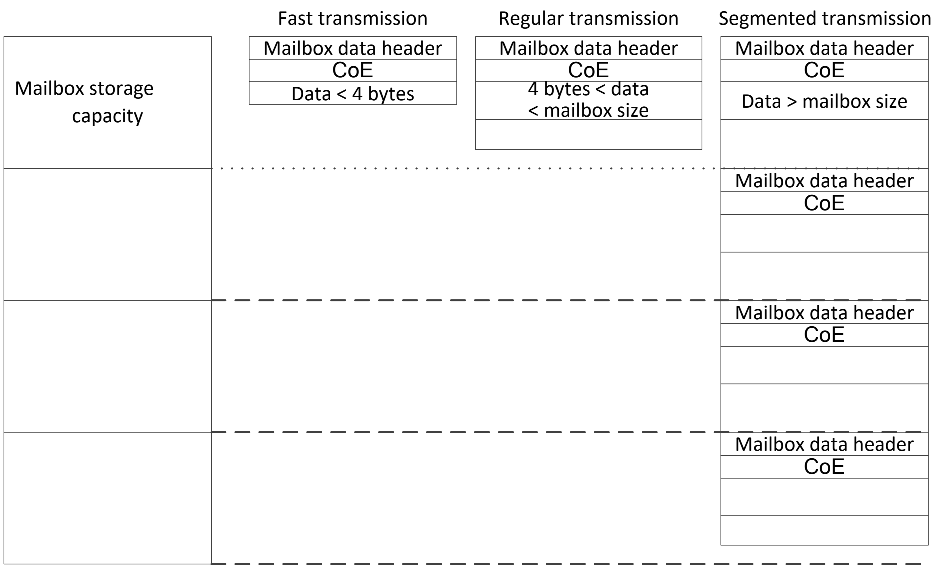
The maximum transmittable valid data depends on the storage area capacity managed by the mailbox SM.

3. Segmented transmission service: When the amount exceeds the mailbox capacity, the data is transmitted in segments.

Table 2-9 CoE Data Frame Content

SDO Control	Standard CANopen SDO Service
Index	Device object index
Sub-index	Sub-index
Data	Data in SDO
Data (optional)	There are 4 bytes of optional data that can be added to the data frame

Figure 2-27 SDO Transmission Type



If the data to be transmitted is larger than 4 bytes, the regular transmission service is used. In regular transmission, the 4 data bytes used in fast transmission represent the complete size of the data to be transmitted, and the extended data part is used to transmit the valid data. The maximum capacity of the valid data is the mailbox capacity minus 16.

2.4.2 Servo Drive Profiles According to IEC 61800-7-204 (SERCOS)

Serial Real-time Communication System (SERCOS) is recognized as a communication interface for high-performance real-time systems, especially for motion control applications. The profiles for its servo drive and communication technology fall within the scope of the IEC 61800-7-204 standard. The key points regarding the integration and compatibility of SERCOS and EtherCAT are listed below:

Mapping of SERCOS and EtherCAT (SoE): The mapping of the servo drive profiles of SERCOS to EtherCAT is defined in Part 304 of the IEC 61800-7-204 standard. SoE (SERCOS over EtherCAT) provides an EtherCAT mailbox-based access method for SERCOS servo drive parameters and functions.

Parameter access and service channel: The service channel for access to all parameters and functions in the drive is based on the EtherCAT mailbox. This approach ensures the compatibility of EtherCAT with the existing SERCOS protocol and enables access to the value, attributes, name, unit, and other information of IDN (SERCOS identifier).

Data transmission mechanism: SERCOS process data (AT and MDT format data) is transmitted through the EtherCAT device protocol mechanism, and its mapping method is similar to that of SERCOS. In this way, SERCOS data can be efficiently transmitted in the EtherCAT network.

State machine mapping: The EtherCAT slave state machine can be easily mapped to the states of the

SERCOS protocol. This state machine mapping makes the integration of SERCOS and EtherCAT smoother, ensuring compatibility and interoperability between the two.

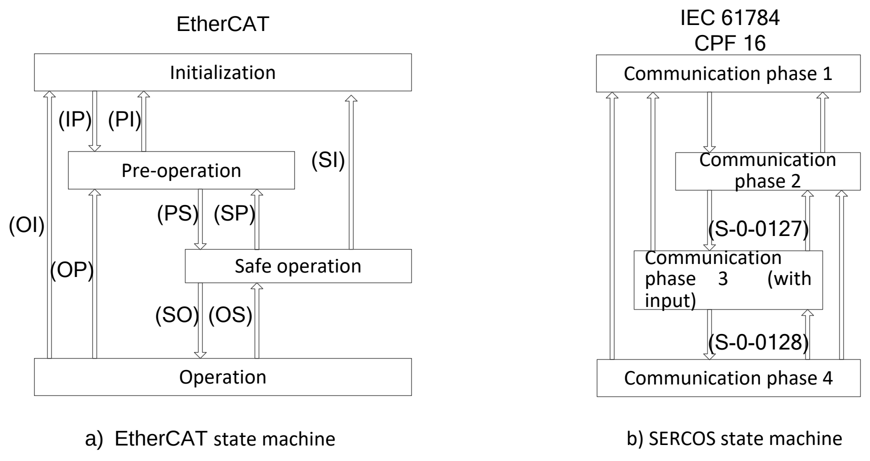
Scalability and data length limitation: While ensuring compatibility, EtherCAT also focuses on scalability related to data length limitations. This scalability ensures that EtherCAT can flexibly respond to different data requirements when processing complex applications.

2.4.2.1 SoE State Machine

A comparison between the communication phases of the SERCOS protocol and the EtherCAT State Machine is shown in Figure 2-28. The SoE protocol has the following features:

1. SERCOS protocol communication phases 0 and 1 are overwritten by the EtherCAT Init state.
2. The communication phase 2 corresponds to the Op state, allowing the use of mailbox communication to implement service channels and manipulate IDN parameters.
3. The communication phase 3 corresponds to the Safe-Op state, where cyclic data transmission begins. At this time, only input data is valid, and output data is ignored. Meanwhile, clock synchronization can be achieved.
4. The communication phase 4 corresponds to the Op state, where all inputs and outputs are valid.
5. The phase switching process instructions S-0-0127 (communication phase 3 switching check) and S-0-0128 (communication phase 4 switching check) of the SERCOS protocol are not used and are replaced by PS and SO state transitions respectively.
6. The SERCOS protocol only allows a high-level communication phase to switch down to the communication phase 0, while EtherCAT allows any state to switch down, as shown in Figure 2-28 a). For example, transition from the Op state to the Safe-Op state, or from the Safe-Op state to the Pre-Op state. SoE should also support this transition, as shown in Figure 2-28 b). If the slave does not support it, an error bit shall be set in the EtherCAT AL State Register.

Figure 2-28 SoE State Machine



2.4.2.2 IDN Inheritance

The SoE protocol inherits the DIN parameter definition of the SERCOS protocol. Each IDN parameter has a unique 16-bit IDN, which corresponds to a unique data block that stores all information about the parameter. The data block consists of 7 elements, as listed in Table 2-10. IDN parameters are divided into two parts: standard data and product data. Each part is divided into 8 parameter groups, which are represented by different IDNs, as listed in Table 2-11.

Table 2-10 IDN Data Block Structure

Number	Name
Element 1	IDN
Element 2	Name

Number	Name
Element 3	Attribute
Element 4	Unit
Element 5	Minimum allowable value
Element 6	Maximum allowable value
Element 7	Data value

Table 2-11 IDN Number Definition

Bit	15	14–12	11–0
Meaning	Classification	Parameter group	Parameter number
Value	0: Standard data S; 1: Product data P	0–7: 8 parameter groups	0000–4095

When EtherCAT is used as the communication network, some IDNs in the SERCOS protocol for communication interface control are deleted, as listed in Table 2-12. And the definitions of some IDNs are modified, as listed in Table 2-13.

Table 2-12 Deleted IDNs

IDN	Description
S-0-0003	Shortest AT transmission starting time
S-0-0004	Transmit/receive transition time
S-0-0005	Minimum feedback processing time
S-0-0009	Position of data record in MDT
S-0-0010	Length of MDT
S-0-0088	Receive to receive recovery time
S-0-0090	Instruction value proceeding time
S-0-0127	CP3 transition check
S-0-0128	CP4 transition check

Table 2-13 Modified IDNs

IDN	Original Description	Updated Description
S-0-0006	AT transmission starting time	Time offset in which an application writes AT data to the ESC storage area after a synchronization signal within the slave.
S-0-0014	Interface status	Mapping of slave DL state and AL state code
S-0-0028	MST error counter	Mapping of slave RX error counter to loss counter
S-0-0089	MDT transmission starting time	Time offset of obtaining new MDT data from the ESC storage area after a synchronization signal within the slave

2.4.2.3 SoE Cyclic Process Data

The output process data (MDT data content) and input process data (AT data content) are configured by S-0-0015, S-0-0016, and S-0-0024. Process data does not include service channel data and only includes cyclic process data. The output process data includes servo control words and instruction data, while the input process data includes status words and feedback data. S-0-0015 sets the type of cyclic process data, as listed in Table 2-14, and the definitions of parameters S-0-0016 and S-0-0024 are listed in Table 2-15. The master writes these three parameters through mailbox communication in the "Pre-Op" phase to configure the content of cyclic process data.

Table 2-14 Definition of Parameter S-0-0015

S-0-0015	Instruction Data	Feedback Data
0: Standard type 0	N/A	No feedback data
1: Standard type 1	Torque instruction S-0-0080 (2	No feedback data

S-0-0015	Instruction Data	Feedback Data
	bytes)	
2: Standard type 2	Speed instruction S-0-0036 (4 bytes)	Speed feedback S-0-0053 (4 bytes)
3: Standard type 3	Speed instruction S-0-0036 (4 bytes)	Position feedback S-0-0051 (4 bytes)
4: Standard type 4	Position instruction S-0-0047 (4 bytes)	Speed feedback S-0-0053 (4 bytes)
5: Standard type 5	Position instruction S-0-0047 (4 bytes) Speed instruction S-0-0036 (4 bytes)	Position feedback S-0-0051 (4 bytes) Or speed feedback S-0-0053 (4 bytes) + Position feedback S-0-0051 (4 bytes)
6: Standard type 6	Speed instruction S-0-0036 (4 bytes)	No feedback data
7: Customized	S-0-0024 configuration	S-0-0016 configuration

Table 2-15 Definitions of Parameters S-0-0016 and S-0-0024

Data Word	S-0-0024 Definition	S-0-0016 Definition
0	Maximum length of output data (Word)	Maximum length of input data (Word)
1	Actual length of output data (Word)	Actual length of input data (Word)
2	First IDN of instruction data mapping	First IDN of feedback data mapping
3	Second IDN of instruction data mapping	Second IDN of feedback data mapping
…	…	…

2.4.2.4 SoE Acyclic Service Channel

The EtherCAT SoE Service Channel (SSC) is implemented by the EtherCAT mailbox communication function and used for acyclic data exchange, such as reading/writing IDNs and their elements. The SoE data header format is shown in Figure 2-29.

Figure 2-29 SoE Data Header Format

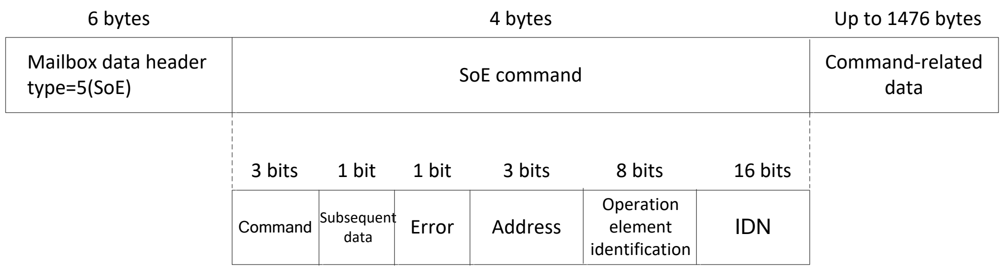


Table 2-16 Description of SoE Data Instructions

Data Area	Description
Instruction	i.e. the instruction type: 0x01: Read request 0x02: Read response 0x03: Write request 0x04: Write response 0x05: Notification

Data Area	Description
	0x06: Slave information 0x07: Reserved
Subsequent data	Subsequent data signal: 0x00: No subsequent data frame 0x01: The transmission is not completed and there are subsequent data frames
Error	Error signal: 0x00: No error 0x01: An error occurred, and the data area has a 2-byte error ID
Address	Specific address of the slave device
Operation element identification	Element selection for single element operation, defined by bit, with each bit corresponding to one element; number of elements for addressing constructs
IDN	IDN number of the parameter, or the remaining segments during the segment operation

Commonly used SSC operations include SSC read operations, SSC write operations, and SSC process instructions.

SSC read operation: The master initiates the SSC read operation and writes the SSC request to the slave. After receiving the read operation request, the slave responds with the requested IDN number and data value. The master can read multiple elements at the same time, so the slave should answer multiple elements. If the slave only supports single element operation, it should respond with the first element requested.

SSC write operation: This operation is used to download data from the master to the slave, which should answer with the result of the write operation. Segment operation consists of one or more segmented write operations and an SSC write response service.

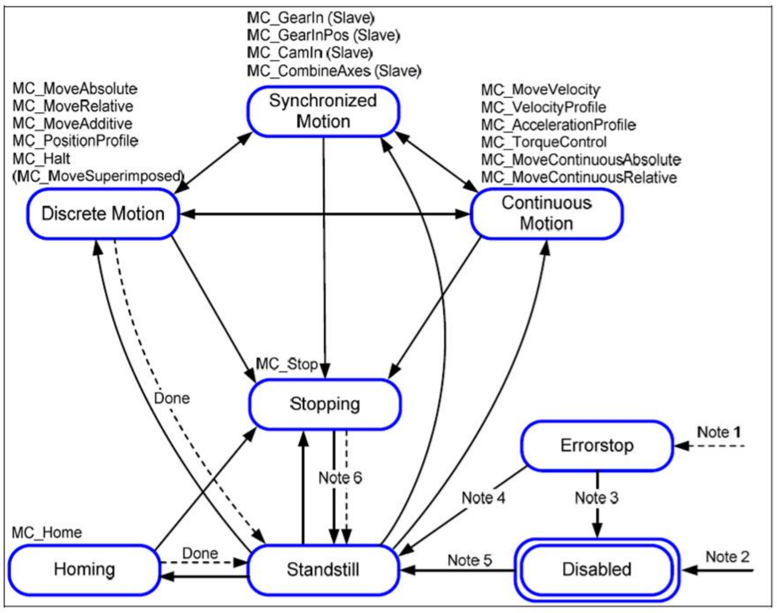
SSC process instruction: It is a special acyclic data. Each process instruction has a unique IDN and specified data elements, which are used to start certain specific functions or processes of the servo device. It usually takes a while to execute these functions or processes. The process instruction only triggers the start of the process, so after that, the service channel it occupies will become immediately available for the transfer of other acyclic data or process instructions. There is no need to wait until the triggered functions or processes to complete their execution.

3 Axis State Mechanism

3.1 Axis State Transition

Axis state transition is designed based on the PLCopen state machine diagram. The specific transition is shown in Figure 3-1.

Figure 3-1 Axis State Transition



Note 1:	From any state. An error in the axis occurred.
Note 2:	From any state. MC_Power.Enable = FALSE and there is no error in the axis.
Note 3:	MC_Reset AND MC_Power.Status = FALSE
Note 4:	MC_Reset AND MC_Power.Status = TRUE AND MC_Power.Enable = TRUE
Note 5:	MC_Power.Enable = TRUE AND MC_Power.Status = TRUE
Note 6:	MC_Stop.Done = TRUE AND MC_Stop.Execute = FALSE

- When the axis is standstill, it can transit to various operational states.
- It can transit to the standstill state from multiple states.
- Discrete motion, synchronized motion, and continuous motion states can be switched directly with each other.
- If an alarm occurs on the servo axis (Errorstop), the MC_Reset and MC_Power instructions must be run first to put the axis into the standstill state before the axis can run again.
- If the MC instruction is not used to instruct the axis to move according to the above transition diagram, the axis will not respond and an error alarm message will be generated from the MC function block.

4 Basics of Programming

4.1 Variable

Variables are to-be-processed abstract data stored in the memory. They are names used to identify PLC input/output and storage areas within the PLC, and can replace physical addresses in programs. Data values stored in the variables can be changed at any time as needed. During program execution, the value of a variable can change.

Before using a variable, you must declare it and specify its type and name. A variable has a name, type, and value. The data type of a variable determines the size and type of memory it represents. A variable name is an identifier in the program source code.

4.1.1 Variable Declaration

Variable declaration is to specify the name, type, and initial value of a variable. Variable declaration is very important. Undeclared variables cannot pass compilation and therefore cannot be used in the program. Users can declare variables in the Program Organization Unit (POU), Global Variable List (GVL), and Auto-Declare dialog box. In CoDeSys applications, variable declaration is divided into two categories: normal variable declaration and direct variable declaration.

- Normal variable declaration

It is the most commonly used variable declaration, which does not need to be associated with hardware peripherals or communications and is only used for internal logic of the project. Normal variable declaration must comply with the following rules:

    <Identifier>:<data type>{:=<initial value>};

The part in {} is optional, such as: nTest:BOOL;, nTest:BOOL:=TRUE;

- Direct variable declaration

In CoDeSys applications, this declaration is required when you need to map variables with the I/O modules of the PLC or communicate with external devices over the network. You can use the keyword AT to directly link a variable to a specific address. Direct variable declaration must comply with the following rules:

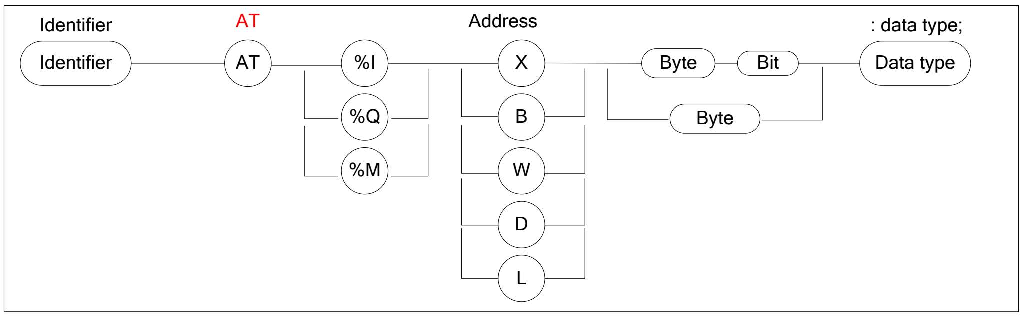
    AT<address>:

    <ATidentifier>AT<address>:<data type>{:=<initialization value>};

The part in {} is optional.

Direct variable declaration starts with "%", followed by the position prefix symbol and the size prefix symbol. If there is a grade, the grade is represented by an integer and a decimal point symbol ".", such as %IX0.0 and %QW0. The specific format of direct variable declaration is shown in Figure 4-1.

Figure 4-1 Direct Variable Declaration



I: input unit; Q: output unit; M: memory cell. The size prefixes are defined as shown in Table 4-1.

Table 4-1 Definitions of Size Prefixes

Prefix Symbol	Definition	Conventional Data Type
X	Bit	BOOL
B	Byte	BYTE
W	Word	WORD
D	Double word	DWORD
L	Long word	LWORD
*	Internal variables without specified positions are automatically allocated by the system.	

[Example 4.1] A variable of double word type Var1 is defined in the program. If you need to fetch part of the data in the variable and convert it into a variable of Boolean, byte, or word type, what is its starting address and how to convert it?

```
VAR
Var1                    AT%ID48                  :DWORD;
END_VAR
```

%I indicates that this variable belongs to the input unit, and its specific address is %ID48. Table 4-2 lists that when CoDeSysV3.x is addressing, the system will make allocation according to the size of the data type (X: bit, B: byte, W: word, D: dword).

In the address memory map, the word addresses %IW96 and %IW97 are combined to correspond to %ID48, because the byte starting address after 48*2 (bytes) is 96. Similarly, the four byte variables of byte addresses %IB192, %IB193, %IB194, and %IB195 correspond to %ID48 when combined, because the corresponding byte starting address after 48*4 (bytes) is exactly 192.

Table 4-2 Memory Map

%IX	96.0–96.7	96.8–192.15	97.0–97.7	97.8–97.15
%IB	192	193	194	195
%IW	96		97	
%ID	48			

[Example 4.2] Based on [Example 4.1], it is easy to understand the following address mapping relationship.

%MX12.0: the first digit of %MB12.

%IW4: the input word unit 4 (byte units 8 and 9).

%Q*: output in a specific location.

%IX1.3: the third bit of the first byte unit of input.

4.1.2 Data Type

Whether you are declaring a variable or a constant, you must use a data type. The standardization of data types is an important sign of openness of programming languages. In CoDeSys, data types fully comply with the standards defined by IEC 61131-3. CoDeSys divides data types into standard data types, extended data types of the IEC1131-3 standard, and custom data types. The data type determines how much storage space it will occupy and what type of value it will store.

CoDeSys standard data types are divided into five categories: Boolean type, integer type, real number type, string type, and time data type.

Table 4-3 lists the standard data types supported by CoDeSys.

Table 4-3 Standard Data Types

Data Category	Data Type	Keyword	Number of Digits	Value Range
Boolean	Boolean	BOOL	1	FALSE (0) or TRUE (1)
Integer	Byte	BYTE	8	0–255
	Word	WORD	16	0–65535
	Double word	DWORD	32	0–4294967295
	Long word	LWORD	64	0–(2^{64}-1)
	Short integer	SINT	8	-128–127
	Unsigned short integer	USINT	8	0–255
	Integer	INT	16	-32768–32767
	Unsigned integer	UINT	16	0–65535
	Double integer	DINT	32	-2147483648–2147483647
	Unsigned double integer	UDINT	32	0–4294967295
	Long integer	LINT	64	-2^{63}–(2^{63}-1)
Real number	Real number	REAL	32	1.175494351e-38–3.402823466e+38
	Long real number	LREAL	64	2.2250738585072014e-308–1.7976931348623158e+308
String	String	STRING	8*N	-
Time data	Storage time	TIME	32	T#0ms–T#71582m47s295ms
		TINE_OF_DAY		T0D#0:0:0–T0D#1993:02:47.295
		DATE		D#1970-1-1–D#2106-02-06
		DATE_AND_TIME		DT#1970-1-1-0:0:0–DT#2106-02-06-06:28:15

4.1.2.1 Boolean

Boolean variables are used to represent TRUE/FALSE values. A Boolean variable has only two states: TRUE or FALSE. In CoDeSys, it can also be represented by 0 or 1.

Type	Memory Usage
BOOL	8 digits

[Example 4.3] Assign the AND logic result of the door opening signal and the material gripping signal to the Boolean variable bReady. The structured text language code is as follows.

```
VAR
bReady,bDoors_Open,bGrip:BOOL;
END_VAR
bReady:=(bDoors_Open and bGrip);
```

In CoDeSys, variables of the same type can be declared uniformly and separated by ",".

[Example 4.4] Assign the decimal number 211 to the variable bReady. The structured text language code is as follows.

```
VAR
bReady:BOOL;
END_VAR
```

```
bReady:=211;
```

Assigning integer data to Boolean data is obviously incorrect. After program compilation, the compiler will return an error message “C0032: Cannot convert “USINT” to “BOOL”.

Boolean variables are the most commonly used variable type. Therefore, it is crucial to learn how to use them correctly since they are often used in process control statements (such as IF, CASE, and loop statements).

Note: If the lowest bit in the memory is set (e.g. 2#00000001), the BOOL type variable is “TRUE”. If the lowest bit in the memory is not set, the BOOL variable is FALSE, for example 2#00000000. All other values cannot be converted correctly and are displayed as (***INVALID:16#xy*** during online monitoring). Similar problems may occur, for example, if overlapped memory ranges are used in a PLC program.

For example, if you define a Boolean array, A:ARRAY[0..7]OFBOOL, the total memory it occupies in the system is not an 8-bit byte but eight 8-bit bytes.

4.1.2.2 Integer

The integer type represents whole numbers without decimal points. In CoDeSys, integer is the largest standard category with the most members. There is no need to memorize the keywords of each type. As long as you understand the rules, it is very easy to remember them. The following briefly explains the rules of integer prefixes.

- U_ represents an unsigned data type, and U is the abbreviation of Unsigned.
- S_ represents the short data type, and S is the abbreviation of Short.
- D_ represents the double data type, and D is the abbreviation of Double.
- L_ indicates the long data type, and L is the abbreviation of Long.

For example, UINT represents unsigned integer data, USINT represents unsigned short integer data, and LINT represents long integer data.

[Example 4.5] Example of integer data.

```
VAR
nValue1:USINT;
nValue2:LINT;
nValue3:WORD;
END_VAR

nValue1:=4;
nValue3:=16;
nValue2:=nValue1+nValue3;
```

The final output result of nValue2 after program running is 20.

The difference between unsigned data and signed data lies in the highest bit.

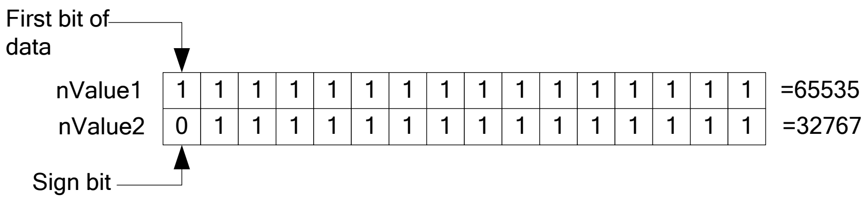
For unsigned data, all storage space is used to store data without a sign bit. For example, an UINT type variable uses all the 16 bits to store data, that is, the data range is 0–65535.

For signed data, the highest bit is used as the sign bit. For example, if the highest bit is used as a sign bit for an INT type variable, and the remaining 15 bits are used for data storage, the data range is -32768–32767. Therefore, the range of positive numbers that can be stored in a signed integer variable is half that in an unsigned integer variable.

Here are two examples of unsigned and signed variables:

```
nValue1:UINT;
nValue2:INT;
```

Figure 4-2 Unsigned and Signed Data Structures



4.1.2.3 Real Number

Real numbers, also called floating-point numbers, are mainly used to process numerical data containing decimals. The real number type includes two data types: REAL and LREAL. REAL real numbers occupy 32 bits of storage space, while LREAL long real numbers occupy 64 bits of storage space. In CoDeSys, there are two representations for real and long real constants.

1. Decimal form

It consists of numbers and a decimal point. 0.123, 123.1, and 0.0 are all decimal numbers.

2. Exponential form

For example, 123e3 or 123E3 both represent 123×10^3. However, it should be noted that there must be a number before the letter e (or E), and the exponent after e must be an integer. For example, e3, 2.1e3.5, .e3, e, etc. are all ungrammatical exponential forms.

A floating-point number can have multiple exponential representations, for example, 123.456 can be represented as 123.456e0, 12.3456e1, 1.23456e2, and so on. Here, 1.23456e2 is called the “normalized exponential form”. That is, in the decimal part before the letter e (or E), there should be one (and only one) non-zero digit to the left of the decimal point.

[Example 4.6] Assign 12.3 to the rRealVar1 variable. The structured text language code is as follows:

```
VAR
rRealVar1:REAL;
END_VAR
RealVar1:=1.23e1;
```

In Example 4.6, 1.23e1 means 12.3. Of course, you can also use the expression RealVar1:=12.3 to meet the requirement in the above example.

At this time, if the requirement is changed to assigning 0.123 to the rRealVar1 variable, according to the rules mentioned above, you only need to change the expression to:

```
RealVar1:=1.23e-1;
or,
RealVar1:=0.123;
```

Note: Support for the data type LREAL depends on the target device.

During compilation, whether the 64-bit LREAL type is converted to REAL (with possible information loss) or remains unchanged requires reference to the corresponding documentation of different hardware products. If a REAL or LREAL data type is converted to an SINT, USINT, INT, UINT, DINT, UDINT, LINT or ULINT data type and the value of the real data type is outside the range of the integer, the result will be uncertain and the value depends on the target system.

This situation may generate an exception. In order to get target-independent codes, all range outliers should be processed by the application If REAL and LREAL data are within the range of the integer, conversion between them can be performed on all systems.

4.1.2.4 String

A string is a sequence of characters. String constants use single quotes as their prefix and suffix. You can also enter spaces and special characters such as ampersands. These characters are treated like all other characters. In CoDeSys, a string type variable can contain any string of characters, enclosed in single quotes. For example, ‘Hello’, ‘Howareyou’, ‘CoDeSys’ and ‘why?’ are all constant strings. The declared size determines the storage space required to store the variable. The storage space here refers to the number of characters in the string, enclosed in parentheses or square brackets. The specific operation and declaration methods are as follows.

- If the string size is not specified when the variable is defined, the system will allocate 80 characters to the variable by default, and the actual storage space occupied in the system = [80+1] bytes.

For example, Str1:STRING:=‘a’ is defined in the variable declaration. Although the actual initial value of the Str1 variable contains only one character, no brackets are used in the declaration to limit the string size. Therefore, the memory space occupied by Str1 in the system is 80+1 bytes.

- If the size is defined, the actual storage space occupied in the system = [(the defined string size) + 1] bytes. In CoDeSys, there is generally no limit on the length of a string, but string functions can only process strings with a length between 1 and 255 characters. For example, to define two strings, the statements are as follows:

```
Str1:STRING[10]:= ‘a’;
Str2:STRING:= ‘a’;
```

The above two statements are similar except that there is an additional storage space limit [10] in the first statement. Figure 4-3 shows the difference between these two statements in the program memory. On the left, since Str1 is limited to 10 bytes, the actual byte size occupied in the program is 10+1 or 11 bytes. The default allocation for Str2 is 80 characters, and the actual size is 80+1 or 81 characters.

Figure 4-3 String Storage Mode

Address	Str1
1	a
2	
3	
4	
5	
...	
10	
11	

Address	Str2
1	a
2	
3	
4	
5	
...	
80	
81	

Generally speaking, the default size of 80 characters can satisfy most applications. However, if the application contains a large amount of string data, but the actual character data in each string is very small, this will cause a large waste of data storage area. Limiting the size can save a lot of storage space for other variables. If a variable is initialized with a string and the string is too long for the variable’s data type, the string will be truncated accordingly from right to left.

When a string is represented in a program, single quotes ‘XXX’ are required to distinguish it from normal variables.

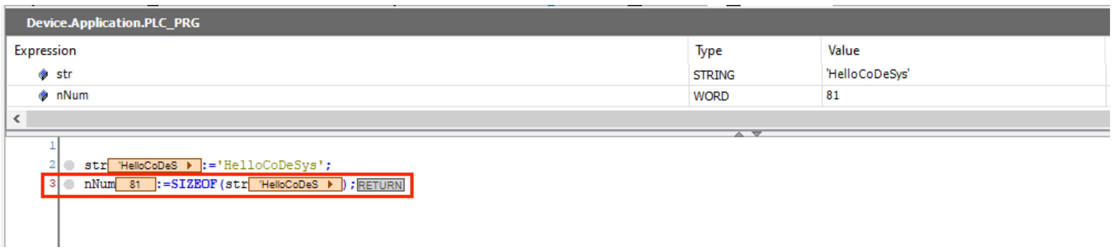
[Example 4.7] Assign the string ‘HelloCoDeSys’ to the str variable.

```
VAR
str:STRING;
nNum:WORD;
END_VAR
str:= ‘HelloCoDeSys’;
```

nNum:=SIZEOF(str); (*Use the SIZEOF instruction to view the storage space usage*)

The result of program execution is shown in Figure 4-4. The actual number of characters in ‘HelloCoDeSys’ is 13 and occupies a storage space of 14 bytes. However, the output result of the SIZEOF instruction is 81 bytes. This is because the string size is not specified and the system automatically allocates 80 characters to the str variable.

Figure 4-4 String Instance Running Results



Device.Application.PLC_PRG

Expression	Type	Value
str	STRING	'HelloCoDeSys'
nNum	WORD	81

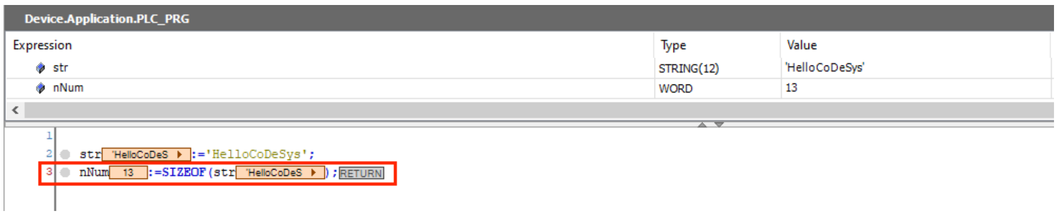
```
str[HelloCoDeS]:='HelloCoDeSys';
nNum[81]:=SIZEOF(str[HelloCoDeS]);RETURN
```

[Example 4.8] Assign the string ‘HelloCoDeSys’ to the str variable, which is defined as 12 characters in size.

```
VAR
str:STRING[12];
nNum:WORD;
END_VAR
str:= ‘HelloCoDeSys’ ;
nNum:=SIZEOF(str);
```

The actual result of program execution is shown in Figure 4-5.

Figure 4-5 String Instance Running Results



Device.Application.PLC_PRG

Expression	Type	Value
str	STRING(12)	'HelloCoDeSys'
nNum	WORD	13

```
str[HelloCoDeS]:='HelloCoDeSys';
nNum[13]:=SIZEOF(str[HelloCoDeS]);RETURN
```

It can be seen from the running results that str only shows ‘HelloCoDeSy’, missing an ‘s’, which means that the redundant part has been automatically truncated by the system. The storage space occupied by the string is 13 bytes.

4.1.2.5 Time Data

Time data types include TIME, TIME_OF_DAY/TOD, DATE, and DATE_AND_TIME/DT. The system processes this data internally in a similar way to the double word (DWORD) type.

1. **TIME:** time, accurate to millisecond (ms), and ranging from 0 to 71582m47s295ms. The syntax format is as follows.

t#<time declaration>

A TIME constant always consists of a start character T or t (or TIME or time) and a numeric identifier #. Then, the actual time declaration follows, including day (d flag), hour (h flag), minute (m flag), second (s flag), and millisecond (ms flag). It should be noted hat the time items must be set according to the order of time length units (i.e., d before h, h before m, m before s, s before ms), but not all time length units need to be included.

Examples of correct use of time constants in ST language assignment statements are as follows:

```
TIME1:=T#14ms;
TIME1:=T#100S12ms;
(*The value of the highest unit can exceed its limit*)
TIME1:=t#12h34m15s;
```

[Example 4.9] Definition and use of time type variables.

```
VAR
tTime:TIME;
END_VAR
tTime:=T#3d19h27m41s1ms;
```

Note: Time can overflow, for example, the hour can exceed 24 h. If T#3d29h27m41s1ms is written during value assignment, the system will automatically correct the final output result to T#4d5h27m41s1ms.

The following time constant assignment is incorrect.

```
tTime:=15ms; (*T# is missing*)
tTime:=t#4ms13d; (*wrong order*)
```

2. **TIME_OF_DAY/TOD:** Time of day, accurate to millisecond (ms), and ranging from 0:0:0 to 1193:02:47.295. The time of day declaration uses the format of "<hour:minute:second>". The syntax format is as follows.

tod#<time declaration>

In addition to "tod#", you can also use "TOD#", "time_of_day", and "TIME_OF_DAY".

[Example 4.10] Definition and use of time of day type variables.

```
VAR
tTime_OF_DAY:TIME_OF_DAY;
END_VAR
tTime_OF_DAY:=TOD#21:32:23.123;
```

The time expressed by the above statement is 21h:32m:23s:123ms.

3. **DATE:** Date, accurate to day (d), and ranging from 1970-01-01 to 2106-02-06. Date declaration uses the format "<year-month-day>". The syntax format is as follows.

   dt#<date declaration>

In addition to "d#", you can also use "D#", "date", and "DATE". These constants can be used to enter dates. When declaring a DATE constant, you can enter the start character d, D, DATE, or date followed by a # sign. Then, you can enter any date in the format YYY-MM-DD.

[Example 4.11] Definition and use of date type variables.

```
VAR
tDate:DATE;
END_VAR
tDate:=D#2014-03-09;
```

The time expressed by the above statement is March 9, 2014.

4. **DATE_AND_TIME/DT:** Date and time, accurate to second (s), and ranging from 1970-01-01-00:00 to 2106-02-06-06:28:15. The declaration of date and time uses the format of "<year-month-day-hour:minute:second>" and the syntax is as follows.

   dt#<date and time declaration>

In addition to "dt#", you can also use "DT#", "date_and_time", and "DATE_AND_TIME".

[Example 4.12] Definition and use of date and time type variables.

```
VAR
tDT:DATE_AND_TIME;
END_VAR
tDT:=DT#2014-03-09-16:22:31.223;
```

The time expressed by the above statement is 16h:22m:31s:223ms on March 9, 2014.

4.1.3 Variable Type

Table 4-4 Types of Variables

Keyword of Variable Type	Variable Attribute	External Read/Write	Internal Read/Write
VAR	Local variable	-	R/W
VAR_INPUT	Input variable, provided externally	R/W	R
VAR_OUTPUT	Output variable, provided by internal variables to external devices	W	R/W
VAR_IN_OUT	Input-output variable	R/W	R/W
VAR_GLOBAL	Global variable, which can be used in all configurations and resources	R/W	R/W
VAR_TEMP	Temporary variable, which are used by programs and function blocks for internal storage	-	R
VAR_STAT	Static variable	-	-
VAR_EXTERNAL	External variable, which can be modified within the program, but must be provided by global variables	R/W	R/W

VAR, VAR_INPUT, VAR_OUTPUT, and VAR_IN_OUT are the most commonly used variable types in program organization units (POUs). VAR_GLOBAL is also used extensively in actual engineering projects.

4.1.4 Persistent Variable

Table 4-5 List of Variable Attributes

Keyword of Additional Variable Attribute	Additional Variable Attribute
RETAIN	Retain variable, used for power failure retention
PERSISTENT	Persistent variable
VAR RETAIN PERSISTENT VAR PERSISTENT RETAIN	With the same function, both are persistent variables used for power failure retention
CONSTANT	Constant

PERSISTENT

Currently, only a few PLCs still retain independent memory areas for storing PERSISTENT type data. In CoDeSysV3.x, the original power failure retention function is canceled and replaced with VARRETAINPERSISTENT or VARPERSISTENTRETAIN, which are exactly the same in function.

The declaration format of a PERSISTENT type variable is as follows:

```
VARGLOBALPERSISTENTRETAIN
<identifier>:<data type>;
END_VAR
```

Memory storage location: Like RETAIN variables, RETAINPERSISTENT and PERSISTENTRETAIN variables are also stored in a separate memory area.

Resetting of persistent variables:

Retain variables are identified with the keyword "RETAIN". These variables always retain their values, even after an abnormal or normal shutdown of the controller or when the "warm reset" instruction is executed. When the program is re-run, the stored values undergo further processing. A specific example is that a pie counter on a production line restarts counting after a power failure. In this case, all other variables are reinitialized rather than using their initialization values or standard initialization values. In contrast to persistent variables, retain variables are reinitialized when the program executes a new download.

Persistent variables are identified by the keyword **"PERSISTENTRETAIN"**. Unlike retain variables, these variables continue to retain their values after a re-download or after executing the instruction “cold reset” or “original reset”. Table 4-6 shows which online instructions will reset persistent variables when executed.

Table 4-6 List of Online Instruction Behaviors for Persistent Variables

Online Instruction	**VAR**	**VAR RETAIN**	**VAR PERSISTENT RETAIN**
Warm reset	-	X	X
Cold reset	-	-	X
Original reset	-	-	-
Download	-	-	X
Online change	X	X	X
Re-download	-	X	X
Note: “X” = Retention value, “-” = Initial value.			

5 Programming Language

5.1 Overview

Different engineering applications have different optimal programming methods, and each programming language has its own characteristics. You can choose the appropriate programming language according to the needs of the actual engineering application. The following briefly introduces CoDeSys's 6 languages with different characteristics.

1. Structured Text (ST): Its advantages lie in that it can realize complex operation control and requires high skills of programmers, while its disadvantages lie in that the code needs to be converted into machine language during compilation, which will lead to long compilation time, slow execution speed, and poor intuitiveness and ease of operation.
2. Ladder Diagram (LD): It corresponds to the electrical diagram. Its advantage lies in its intuitiveness, which is easy for electrical technicians to learn and master, while its disadvantage lies in that the program description is often not clear enough when dealing with complex control system programming. Ladder Diagram is the most widely used PLC programming language in the domestic industrial automation field.
3. Function Block (FBD): With function blocks as design units, we can start from the control function. Its advantages lie in that it facilitates the analysis and understanding of control schemes, is intuitive and easy to master, and has good operability. When dealing with complex control systems, It can still be described clearly in graphical form. Its disadvantages lie in that each function block takes up program storage space and prolongs the program execution cycle.
4. Instruction List (IL): Its advantages lie in that it is easy to remember and master, has a corresponding relationship with the ladder diagram (LD), is convenient for mutual conversion and program check, and is not limited by the screen size during programming and debugging, and the input elements are not restricted, while its disadvantage lies in that, like the ladder diagram, the program description of complex systems is not clear enough.
5. Sequential Function Chart (SFC): The completed functions are represented by the main line. Its advantages lie in that the operation process is clear and easy to understand; for large programs, the design can be divided into different tasks and a more flexible program structure can be used to save program design time and debugging time; and since only active steps are scanned, the program execution time can be shortened.
6. Continuous Function Chart (CFC): Actually, it is another form of Function Block Diagram (FBD). The operation order of operation blocks can be customized throughout the program, making it easy to implement numerous large-scale process operations that are difficult to subdivide. It is widely used in the continuous control industry.

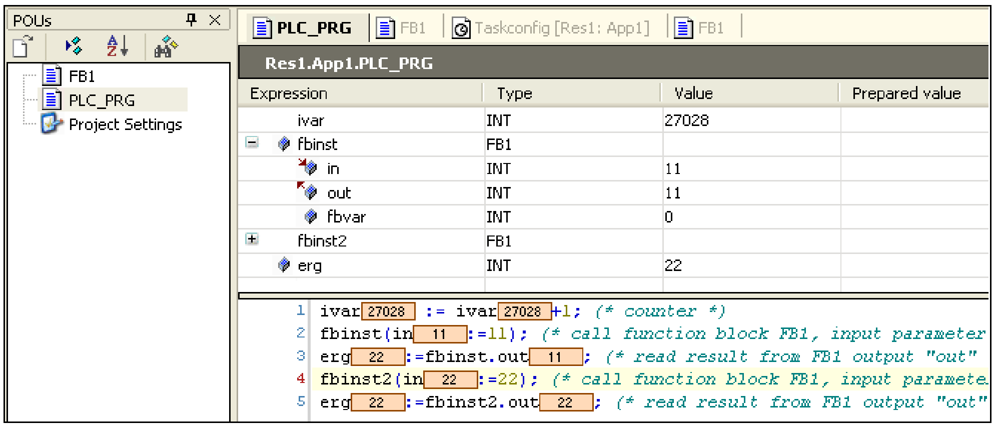
5.2 Structured Text (ST)

5.2.1 Introduction to the Structured Text Programming Language

Structured Text (ST) is a high-level text language that can be used to describe functions, function blocks, and program behaviors, and can also describe the behaviors of steps, actions, and transitions in Sequential Function Charts.

Structured Text Programming Language is a high-level language, similar to Pascal, which is developed specifically for industrial control applications and is the most commonly used language in CoDeSys. For personnel who are familiar with high-level computer language development, the structured text language is easy to learn and use. It can implement functions such as selection, iteration, and jump statements. In addition, the structured text language is easy to read and understand, especially when annotated with meaningful identifiers and comments. In complex control systems, structured text can greatly reduce the amount of code and make complex system problems simple. Its disadvantage lies in unintuitive debugging and relatively slow compilation speed. The view of structured text is shown in Figure 5-1.

Figure 5-1 Structured Text



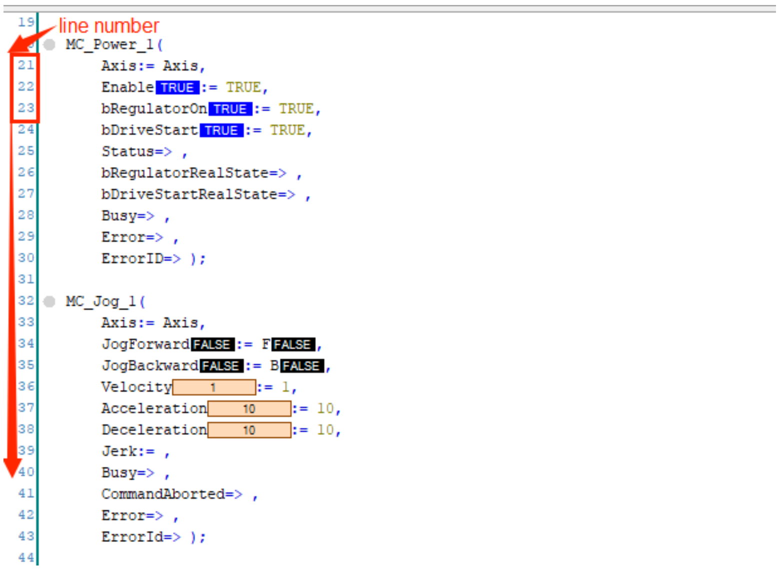
Expression	Type	Value	Prepared value
ivar	INT	27028	
fbinst	FB1		
in	INT	11	
out	INT	11	
fbvar	INT	0	
fbinst2	FB1		
erg	INT	22	

```
ivar 27028 := ivar 27028 +1; (* counter *)
fbinst(in 11 :=11); (* call function block FB1, input parameter
erg 22 :=fbinst.out 11 ; (* read result from FB1 output "out"
fbinst2(in 22 :=22); (* call function block FB1, input paramete
erg 22 :=fbinst2.out 22 ; (* read result from FB1 output "out"
```

5.2.2 Program Execution Sequence

The execution sequence of the program using structured text is based on the "line number" from top to bottom, as shown in Figure 5-2. At the beginning of each cycle, the program lines with smaller line numbers are executed first.

Figure 5-2 Structured Text Program Execution Sequence



```
19
20  MC_Power_1(
21      Axis:= Axis,
22      Enable TRUE := TRUE,
23      bRegulatorOn TRUE := TRUE,
24      bDriveStart TRUE := TRUE,
25      Status=> ,
26      bRegulatorRealState=> ,
27      bDriveStartRealState=> ,
28      Busy=> ,
29      Error=> ,
30      ErrorID=> );
31
32  MC_Jog_1(
33      Axis:= Axis,
34      JogForward FALSE := F FALSE ,
35      JogBackward FALSE := B FALSE ,
36      Velocity 1 := 1,
37      Acceleration 10 := 10,
38      Deceleration 10 := 10,
39      Jerk:= ,
40      Busy=> ,
41      CommandAborted=> ,
42      Error=> ,
43      ErrorId=> );
44
```

5.2.3 Expression Execution Sequence

An expression includes operators and operands. The operands are calculated according to the rules specified by the operators to obtain the results and return them. Operands can be variables, constants, register addresses, functions, etc.

[Example 5.1] Expression examples.

a+b+c;

3.14*R*R;

ABS(-10)+var1;

If there are several operators in an expression, the operators are executed in the conventional order of precedence: operators with higher precedence are executed before those with lower precedence sequentially. If there are operators with the same precedence in an expression, they are executed from left to right in the order they are written. The operator precedence is shown in Table 5-1.

Table 5-1 Operator Precedence

Operator	Symbol	Priority
Parentheses	()	Highest
Function call	Function name(Parameter list)	↑
Exponentiation	EXPT	
Negation	NOT	
Multiplication	*	
Division	/	
Modulo	MOD	
Addition	+	
Subtraction	-	
Comparison	<, >, <=, >=	
Equal	=	
Not equal	<>	
And	AND	
Exclusive or	XOR	
Or	OR	Lowest

5.2.4 Instruction Statement

There are five main types of structured text statements, namely assignment statements, function and function block control statements, selection statements, iteration (loop) statements, and jump statements. Table 5-2 lists all the statements used in structured text.

Table 5-2 Structured Text Statements

Instruction Type	Instruction Statement	Example
Assignment statement	:=	bFan:=TRUE;
Function and function block control statement	Function block/function call name ();	-
Selection statement	IF	IF<Boolean expression>THEN <statement content>;END_IF
	CASE	CASE<condition variable>OF <value 1>:<statement content 1>; ... <value n>:<statement content n>;ELSE <ELSE statement content>; END_CASE;
Iteration statement	FOR	FOR<variable>:=<initial value>TO<target value>{BY<step length>}DO <statement content> END_FOR;
	WHILE	WHILE<Boolean expression> <statement content>;END_WHILE;
	REPEAT	REPEAT <statement content>UNTIL <Boolean expression>

Instruction Type	Instruction Statement	Example
		END_REPEAT;
Jump statement	EXIT	EXIT;
	CONTINUE	CONTINUE**;**
	JMP	<identifier>: ... JMP<identifier>;
Return statement	RETURN	RETURN;
NULL statement	;	-

1. Assignment statement

The assignment statement is one of the most commonly used statements in structured text. Its function is to assign the value generated by the expression on its right side to the operand (variable or address) on the left side. It is represented by ":=".

The specific format is as follows:

```
<variable>:=<expression>;
```

[Example 5.2] Assign values to two Boolean variables: bFan is set to TRUE and bHeater is set to FALSE.

```
VAR
    bFan: BOOL; bHeater:BOOL;
END_VAR
    bFan:=TRUE;
    bHeater:=FALSE;
```

The above functions are achieved by using the ":=" assignment statement.

You need to pay attention to the matching of data types when using it. If the data types on both sides of the assignment operator are different, the data type conversion function should be called. For example, rVar1 is of Real type, and iVar1 is of Int type. When iVar1 is assigned to rVar1, the conversion function INT_TO_REAL should be called.

The statement format is as follows.

```
rVar1:=INT_TO_REAL(iVar1);
```

There can be multiple statements in one line, for example, arrData[1]:=3;arrData[2]:=12; these two instructions can be written in one line.

[Example 5.3] There can be multiple data in one line.

```
arrData1[i]:=iDataInLine1; arrData2[j]:=iDataInLine2;
```

When a function is called, the function return value is assigned as the value of the expression, which should be the most recently evaluated result.

[Example 5.4] The return value of the function call is used as the value of the expression.

```
Str1:=INSERT(IN1:=' CoDe' ,IN2:=' Sys' ,P:=2);
```

2. Function and function block control statement

A. Function control statement

The function block control statement is used to call a function. After the function is called, the return value is directly assigned to the variable as the value of the expression. For example, in the statement rVar1:=SIN(rData1);, the sine function SIN is called and the return value is assigned to the variable rVar1. The statement format is as follows.

```
Variable:=function name (parameter list);
```

[Example 5.5] Example of a function control statement.

rResult:=ADD(rData1,rData2);// Use the ADD function to assign the result of rData1 plus rData2 to the variable rResult.

B. Function block control statement

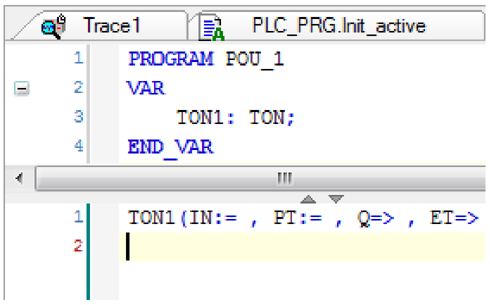
Function block control statements are used for function blocks. Function block calls are implemented by instantiating function block names. For example, Timer is the instance name of the TON function block. The specific format is as follows.

Function block instance name: (function block parameter);

If you need to call a function block in the ST programming language, you can directly enter the instance name of the function block and assign values or variables separated with commas to each parameter of the function block in the subsequent brackets. The function block call ends with a semicolon.

For example, call the TON timer function block in the ST programming language. Assuming its instance name is TON1, the specific implementation is shown in Figure 5-3.

Figure 5-3 Function Block Call in Structured Text



```
PROGRAM POU_1
VAR
    TON1: TON;
END_VAR
```

```
TON1(IN:= , PT:= , Q=> , ET=>
```

A selection statement selects an expression based on specified conditions to determine which statement it consists of to be executed. It can be broadly divided into two categories: IF and CASE.

3. Selection statement

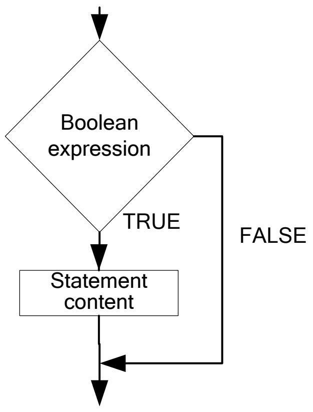
A. IF statement

The IF statement is used to implement a single-branch selection structure. Its basic format is as follows.

```
IF<Boolean expression>THEN
    <statement content>;
END_IF
```

If the above format is used, the statement content will be executed only when <Boolean expression> is TRUE; otherwise, the <statement content> of the IF statement will not be executed. The statement content can be a single statement or a null statement, or multiple statements can be listed in parallel. The statement expression execution process is shown in Figure 5-4.

Figure 5-4 Execution Process of Simple IF Statement



[Example 5.6] Use the PLC to determine whether the current temperature exceeds 60°C. If so, always turn on the fan for heat dissipation. The implementation code is as follows.

```
VAR
        nTemp:BYTE;          (*Current temperature state signal*)
        bFan:BOOL;           (*Fan switch control signal*)
END_VAR
        nTemp:=80;
        IF nTemp>60 THEN bFan:=TRUE;
END_IF
```

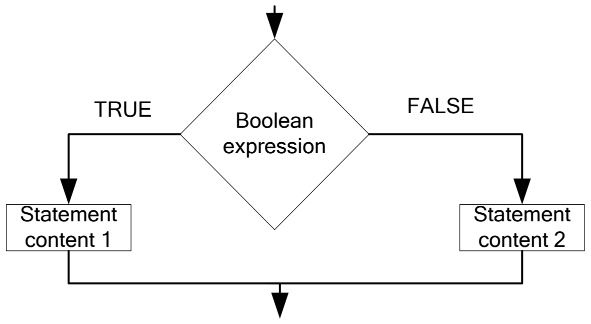
B. IF…ELSE statement

Use the IF...ELSE statement to implement the double-branch selection mechanism. Its basic format is as follows:

```
IF <Boolean expression> THEN
<statement content 1>;
ELSE
<statement content 2>;
END_IF
```

As shown in the above expression, the value in <Boolean expression> is first determined: If it is TRUE, <statement content 1> is executed; if it is FALSE, <statement content 2> is executed. The program execution process is shown in Figure 5-5.

Figure 5-5 Execution Process of IF ELSE Statement



[Example 5.7] Use the PLC to determine that when the temperature is less than 20℃, turn on the heating device; otherwise (temperature ≥ 20℃), disconnect the heating device.

```
IF nTemp<20 THEN
        bHeating:=TRUE;
ELSE
        bHeating:=FALSE;
END_IF
VAR
        nTemp:BYTE;              (*Current temperature state signal*)
        bHeating:BOOL;           (*Heater switch control signal*)
END_VAR
```

When there is more than one conditional expression in the program, a nested IF...ELSE statement is required, that is, a multi-branch selection structure. Its basic format is as follows.

```
IF<Boolean expression1>THEN
```

```
            IF<Boolean expression2>THEN
                <statement content1>;
                ELSE
                <statement content2>;
                END_IF
        ELSE
            <statement content3>;
        END_IF
```

As shown above, another IF...ELSE statement is placed in IF...ELSE to achieve nesting. The following example illustrates the use of nesting.

The above expression first determines the value in <Boolean expression 1>: If it is TRUE, it continues to determine the value of <Boolean expression 2>; if it is FALSE, it executes <statement content 3> and returns to <Boolean expression 2> for determination. If <Boolean expression 2> is TRUE, it executes <statement content 1>; otherwise, it executes <statement content 2>.

[Example 5.8] When the device enters the automatic mode, if the actual temperature is ＞ 50℃, the fan will be turned on and the heater will be turned off. When the temperature is ≤ 50℃, the fan will be turned off and the heater will be turned on. In manual mode, the heater and fan will not work.

```
        VAR
            bAutoMode: BOOL;            (*Manual/automatic mode state signal*)
            nTemp:BYTE;             (*Current temperature state signal*)
            bFan:BOOL;              (*Fan switch control signal*)
            bHeating:BOOL;          (*Heater switch control signal*)
        END_VAR
        IF bAutoMode=TRUE THEN IF
          nTemp>50 THEN
            bFan:=TRUE;
            bHeating:=FALSE;
        ELSE
            bFan:= FALSE;
            bHeating:= TRUE;
        END_IF
        ELSE
            bFan:= FALSE;
            bHeating:=FALSE;
        END_IF
```

C. IF…ELSIF…ELSE statement

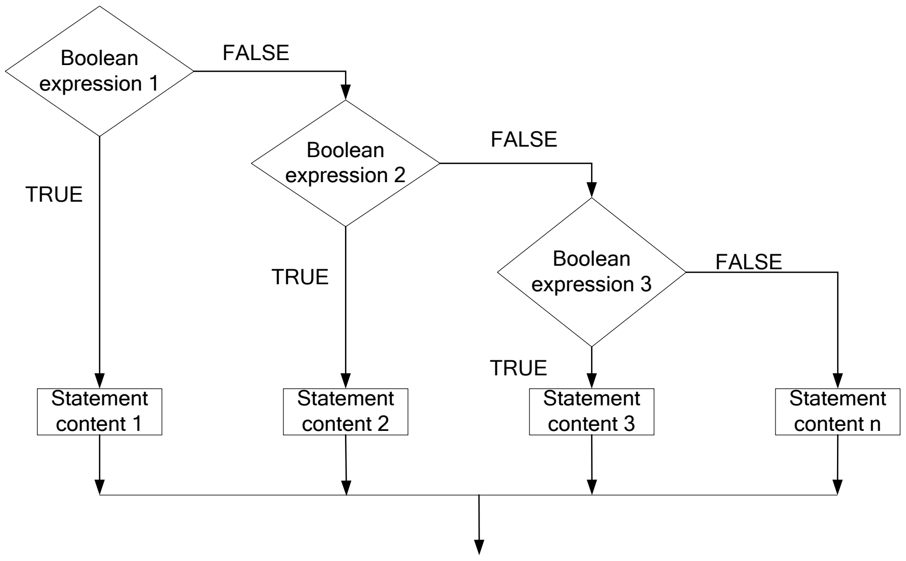
In addition, the multi-branch selection structure can also be presented in the following forms. Its specific format is as follows:

```
        IF <Boolean expression 1> THEN
            <statement content 1>;
        ELSIF <Boolean expression 2> THEN
            <statement content 2>;
        ELSIF <Boolean expression 3> THEN
            <statement content 3>;
```

```
    . . .
    ELSE
        <statement content n>;
    END_IF
```

If the expression <Boolean expression 1> is TRUE, only the instruction <statement content 1> is executed, and no other instructions are executed. Otherwise, determination is started from the expression <Boolean expression 2> until one of the Boolean expressions is TRUE, and then the statement content corresponding to this Boolean expression is executed. If the value of the Boolean expression is not TRUE, only the instruction <statement content n> is executed. The program execution process is shown in Figure 5-6.

Figure 5-6 Execution Process of IF...ELSIF...ELSE Statement



D. CASE statement

A CASE statement is a multi-branch selection statement that enables the program to select a branch from multiple branches for execution based on the value of an expression. Its basic format is as follows.

```
    CASE<condition variable>OF
        <value 1>:<statement content 1>;
        <value 2>:<statement content 2>;
        <value 3,value 4,value 5>:<statement content3>;
        <value6..value10>:<statement content4>;
    ...
        <valuen>:<statement contentn>;
        ELSE
        <ELSEstatement content>;
    END_CASE;
```

The CASE statement is executed in the following mode:

- If the value of the <condition variable> is <value i>, then the instruction <statement content i> is executed.
- If the <condition variable> does not have any specified value, the instruction <ELSE statement content> is executed.
- If several values of the condition variable require the same instruction to be executed, the values can be written one after the other, separated by commas. In this way, the common instruction is executed, as shown in the fourth line of the above program.

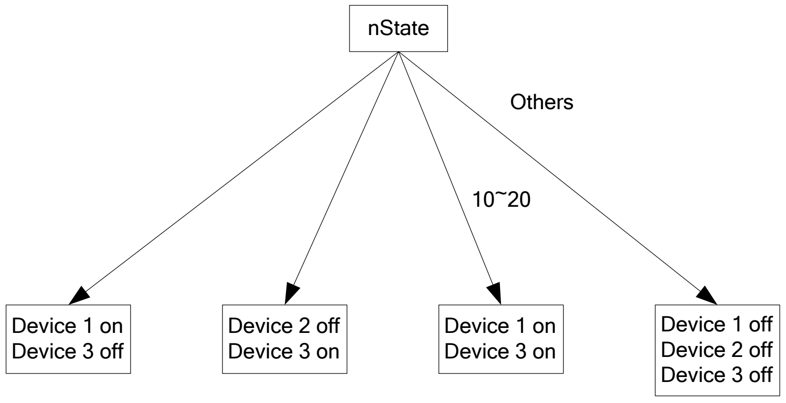
- If the condition variable needs to execute the same instruction within a certain range, you can write the initial and final values, separated by two dots. In this way, the common instruction is executed, as shown in the fifth line of the above program.

[Example 5.9] When the current state is 1 or 5, the device 1 is running and the device 3 is stopped; when the state is 2, the device 2 is stopped and the device 3 is running; if the current state is between 10 and 20, both devices 1 and 3 are running. In other cases, devices 1, 2, and 3 are required to stop. The specific implementation code is as follows:

```
VAR
    nDevice1,nDevice2,nDevice3:BOOL;      (*Device 1..3 switch control signal*)
    nState:BYTE;          (*Current state signal*)
END_VAR

CASE nState OF 1,
5:
    nDevice1:=TRUE;
    nDevice3:=FALSE;
2:
    nDevice2:=FALSE;
    nDevice 3:=TRUE;
10..20:
    nDevice1:=TRUE;
    nDevice 3:=TRUE;
ELSE
    nDevice1:=FALSE;
    nDevice2:=FALSE;
    nDevice3:=FALSE;
END_CASE;
```

The CASE statement execution process is shown in Figure 5-7. When nState is 1 or 5, the device 1 is on and the device 3 is off; when nState is 2, the device 2 is off and the device 3 is on; when nState is 10-20, the device 1 is off and the device 3 is on; in other cases, devices 1 , 2, and 3 are all off.

Figure 5-7 Execution Process of CASE Statement



4. Iteration statement

Iteration statements are mainly used for repeatedly executing programs. In CoDeSys, common iterative statements include FOR, REPEAT, and WHILE statements, which are explained in detail below.

A. FOR loop

The FOR loop statement is used to calculate an initialization sequence. When a certain condition is TRUE, the nested statements are repeatedly executed and an iterative expression sequence is calculated. If it is FALSE, the loop is terminated. Its specific format is as follows.

```
FOR<variable>:=<initial value>TO<target value>{BY<step size>}DO
    <statement content>
END_FOR;
```

The execution sequence of the FOR loop is as follows:

- Calculate whether the <variable> is within the range of the <initial value> and the <target value>.
- When the <variable> is less than the <target value>, the <statement content> is executed.
- When the <variable> is greater than the <target value>, the <statement content> is not executed.
- Each time the <statement content> is executed, the value of the <variable> is always increased by the specified step size. The step size can be any integer value. If the step size is not specified, it defaults to 1. When the <variable> is greater than the <target value>, exit the loop.

In a sense, the principle of the FOR loop is like a copier. The number of copies to be made is preset on the copier, which is the condition of the loop. When the condition is met, that is, the actual number of copies is equal to the set number of copies, copying stops.

The FOR loop is the most commonly used loop statement. It embodies a function of repeating a specified number of times, but due to different code writing methods, other loop functions can also be implemented. The following example demonstrates how to use the FOR loop.

[Example 5.10] Use the FOR loop to calculate 2 to the 5th power.

```
VAR
    Counter:BYTE;      (*Loop counter)
    Var1:WORD;         (*Output result*)
END_VAR
FOR Counter:=1 TO 5 BY 1 DO
    Var1:=Var1*2;
END_FOR;
```

Assuming that the initial value of Var1 is 1, the value of Var1 is 32 after the loop ends.

Note: If the <target value> is equal to the limit value of the <variable>, an infinite loop will be entered. Assume that the type of the counting variable Counter in [Example 5.10] is SINT (-128 to 127). When the <target value> is set to 127, the controller will enter an infinite loop. Therefore, a limit value cannot be set for the <target value>.

B. WHILE loop

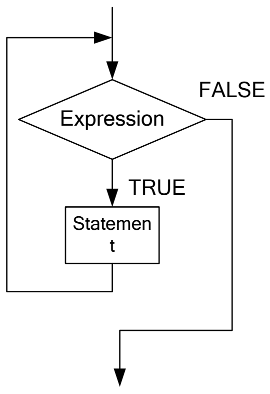
The WHILE loop is used in a similar way to the FOR loop. The difference between the two is that the end condition of the WHILE loop can be any logical expression. That is, you can specify a condition, and when the condition is met, the loop is executed. Its specific format is as follows.

```
WHILE <Boolean expression>
    <statement content>;
END_WHILE;
```

The execution sequence of the WHILE loop is as follows:

- Calculate the return value of the <Boolean expression>.
- When the value of the <Boolean expression> is TRUE, the <statement content> is executed repeatedly.
- When the initial value of the <Boolean expression> is FALSE, the instruction <statement content> is not executed and jumps to the end of the WHILE statement. The execution process is shown in Figure 5-8.

Figure 5-8 Execution Process of WHILE Statement



The WHILE statement is like controlling a motor in a project: when the “Start” button is pressed (when the Boolean expression is TRUE), the motor keeps rotating; when the stop button is pressed (when the Boolean expression is FALSE), the motor stops immediately. The following example demonstrates how to use the WHILE loop.

Note: If the value of the <Boolean expression> is always TRUE, an infinite loop will be entered, which should be avoided. The generation of an infinite loop can be avoided by changing the condition of the loop instruction. For example: Use an incrementing and decrementing counter to avoid an infinite loop.

[Example 5.11] As long as the counter is not zero, the program inside the loop body is always executed.

```
VAR
    Counter: BYTE;          (*Counter*)
    Var1:WORD;
END_VAR
WHILE Counter<>0 DO
    Var1 := Var1*2;
    Counter := Counter-1;
END_WHILE
```

In a sense, the WHILE loop is more powerful than the FOR loop because the WHILE loop does not need to know the number of loops before executing the loop. Therefore, in some cases, it is sufficient to use only these two loops. However, if the number of loops is known, the FOR loop is better because it avoids infinite loops.

C. REPEAT loop

A REPEAT loop differs from a WHILE loop because it checks the end condition only after the instruction is executed. This means that the loop will be executed at least once, regardless of the end condition.

Its specific format is as follows.

```
REPEAT
    <statement content>
UNTIL
    <Boolean expression>
END_REPEAT;
```

The execution sequence of the REPEAT loop is as follows:

- When the value of the <Boolean expression> is FALSE, the <statement content> is executed.
- When the value of the <Boolean expression> is TRUE, the execution of the <statement content> stops.
- After the first execution of the <statement content>, if the value of the <Boolean expression> is TRUE,

the <statement content> is executed only once.

Note: If the value of the <Boolean expression> is always TRUE, an infinite loop will be entered, which should be avoided. The generation of an infinite loop can be avoided by changing the condition of the loop instruction. For example: Use an incrementing and decrementing counter to avoid an infinite loop.

The following example demonstrates how to use the REPEAT loop.

[Example 5.12] Example of a REPEAT loop. The REPEAT loop stops when the counter reaches 0.

```
VAR
    Counter: BYTE;
END_VAR

REPEAT
    Counter := Counter+1;
UNTIL
    Counter=0
END_REPEAT
```

The result of this example is that each program cycle enters the REPEAT loop, and the Counter is BYTE (0–255), that is, 256 auto-increment operations are performed in each cycle.

As mentioned above, "This means that the loop will be executed at least once, regardless of the end condition", so every time the REPEAT statement is entered, the Counter is first 1, and the Counter:=Counter+1 instruction is executed 256 times in each cycle until the Counter variable is accumulated to overflow to 0, and then the loop is exited. It is incremented until it overflows, and so on.

5. Jump statement

A. EXIT statement

If the EXIT instruction is used in the FOR, WHILE, and REPEAT loops, the inner loop stops immediately regardless of the end condition. Its specific format is as follows.

```
EXIT;
```

[Example 5.13] Use the EXIT instruction to avoid division by zero when an iterative statement is used.

```
FOR Counter:=1 TO 5 BY 1 DO INT1:= INT1/2;
IF INT1=0 THEN
    EXIT;                (*Avoid division by zero*)
END_IF
    Var1:=Var1/INT1;
END_FOR
```

When INT1 is equal to 0, the FOR loop ends.

B. CONTINUE Statement

This instruction is an extended instruction of the IEC 61131-3 standard. The CONTINUE instruction can be used in three loops: FOR, WHILE, and REPEAT.

The CONTINUE statement interrupts the current loop, ignoring the code following it and starting a new loop directly. When multiple loops are nested, the CONTINUE statement can only cause the loop statement that directly contains it to start a new loop. Its specific format is as follows.

```
CONTINUE;
```

[Example 5.14] Use the CONTINUE instruction to avoid division by zero when an iterative statement is used.

```
VAR
```

```
        Counter: BYTE;          (*Loop counter*)
        INT1,Var1: INT;         (*Intermediate variable*)
        Erg: INT;               (*Output result*)
    END_VAR
    FOR Counter:=1 TO 5 BY 1 DO
    INT1:= INT1/2;
    IF INT1=0 THEN
        CONTINUE;               (*Avoid division by zero*)
    END_IF
        Var1:=Var1/INT1;        (*Executed only when INT1 is not equal to 0*)
    END_FOR;
    Erg:=Var1;
```

C. JMP statement

A jump statement can be used to unconditionally jump to the code line marked with a jump identifier. Its specific format is as follows.

```
<identifier>:
.
JMP <identifier>;
```

The <identifier> can be any identifier and is placed at the beginning of a program line. The JMP instruction is followed by the jump destination, which is a predefined identifier. When the JMP instruction is executed, it will jump to the program line corresponding to the identifier.

Note: It is necessary to avoid creating an infinite loop, and you can use the IF condition to control the jump instruction.

[Example 5.15] Use the JMP statement to loop the counter in the range of 0..10.

```
VAR
    nCounter: BYTE;
END_VAR
Label1:nCounter:=0;
Label2:nCounter:=nCounter+1;
IF nCounter<10 THEN
    JMP Label2;
ELSE
    JMP Label1;
END_IF
```

Label1 and Label2 in the above example are labels rather than variables, so variable declaration is not required in the program.

Use the IF statement to determine whether the counter is within the range of 0-10. If it is within the range, the statement JMPLabel2 is executed, and the program will jump to Label2 in the next cycle and execute the program nCounter:=nCounter+1 to increase the counter by 1. Otherwise, it will jump to Label1 and execute nCounter:=0 to clear the counter.

The function in this example can also be implemented using a FOR, WHILE, or REPEAT loop. In general, you should avoid using the JMP instruction because it will reduce the readability and reliability of your code.

6. RETURN instruction

The RETURN instruction is used to exit a program organization unit (POU). Its specific format is as follows.

```
        RETURN;
```

[Example 5.16] Use the IF statement for determination. When the condition is met, end the execution of this program immediately.

```
        VAR
                nCounter: BYTE;
                bSwitch: BOOL;          (*switching signal*)
        END_VAR
        IF bSwitch=TRUE THEN
        RETURN;
        END_IF;
        nCounter:= nCounter +1;
```

When bSwitch is FALSE, nCounter is always auto-incremented by 1. When bSwitch is TRUE, nCounter keeps the value of the previous cycle and exits the program organization unit (POU) immediately.

7. NULL statement

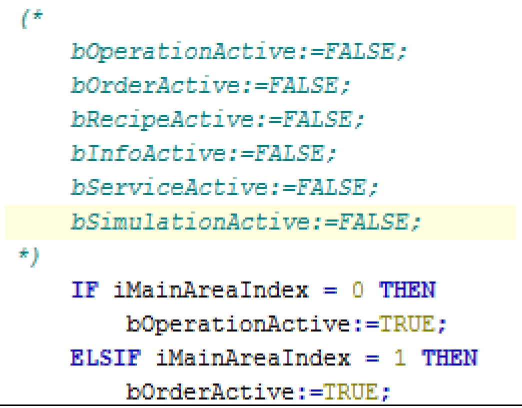
A null statement (;) means that nothing is executed.

8. Annotation

Annotation is a very important part of a program, which makes the program more readable without affecting its execution. You can add an annotation anywhere in the declaration or execution section of the ST editor. In the ST language, there are two ways of annotation.

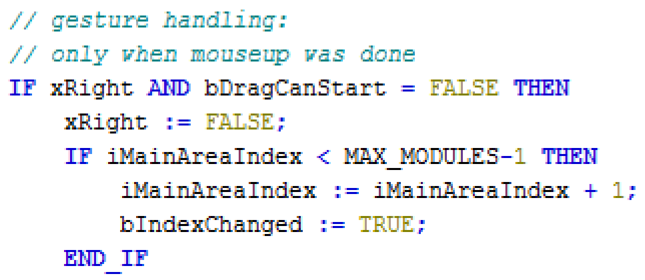
1. A multi-line annotation starts with (* and ends with *). This annotation method allows multi-line annotations, as shown in Figure 5-9.
2. A single-line annotation starts with "//" and continues to the end of the line, as shown in Figure 5-10. Please note that CoDeSysV2 does not support this annotation method currently.

Figure 5-9 Structured Text Language Annotation (Multi-line Annotation)



```
(*
    bOperationActive:=FALSE;
    bOrderActive:=FALSE;
    bRecipeActive:=FALSE;
    bInfoActive:=FALSE;
    bServiceActive:=FALSE;
    bSimulationActive:=FALSE;
*)
    IF iMainAreaIndex = 0 THEN
        bOperationActive:=TRUE;
    ELSIF iMainAreaIndex = 1 THEN
        bOrderActive:=TRUE;
```

Figure 5-10 Structured Text Language Annotation (Single-line Annotation)



```
// gesture handling:
// only when mouseup was done
IF xRight AND bDragCanStart = FALSE THEN
    xRight := FALSE;
    IF iMainAreaIndex < MAX_MODULES-1 THEN
        iMainAreaIndex := iMainAreaIndex + 1;
        bIndexChanged := TRUE;
    END_IF
```

5.2.5 Application Examples

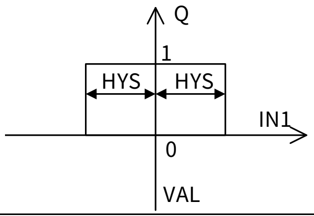
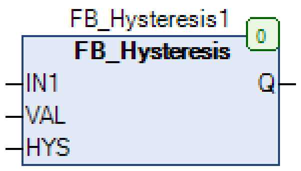
[Example 5.17] Hysteresis function block FB_Hystersis.

1. Control requirements

This function block has three input signals, namely the current real-time value input signal, the comparison setting value input signal, and the deviation value input signal. In addition, an output value is required. When the output is TRUE, it switches to FALSE only when the input signal IN1 is less than VAL-HYS. When the output signal is FALSE, the output switches to TRUE only when the input signal IN1 is greater than VAL+HYS.

The input/output variables of the function block FB_Hystersis are defined as follows.

```
FUNCTION_BLOCK FB_Hysteresis
VAR_INPUT
        IN1:REAL;               // Input signal
        VAL:REAL;               // Comparison signal
        HYS:REAL;               // Hysteresis deviation signal
END_VAR
VAR_OUTPUT
        Q:BOOL;
END_VAR
```

Figure 5-11 Hysteresis Process	Figure 5-12 Function Block Diagram
	 FB_Hysteresis1 0 FB_Hysteresis IN1 VAL HYS Q

2. Function block programming

The program used by the function block body to judge the input signal is as follows.

```
IF Q THEN
  IF IN1<(VAL-HYS) THEN
        Q:=FALSE;               // IN1 decreases
  END_IF
  ELSIF IN1>(VAL+HYS) THEN
        Q:=TRUE;                // IN1 increases
END_IF
```

3. Function block application

The FB_Hysteresis function block can be used for bit signal control, where IN1 is connected to the process variable rActuallyValue, VAL is linked to the process setting value rSetValue, and rTolerance is the required control deviation. The program declaration is as follows.

```
PROGRAM POU
VAR
        fbHysteresis:FB_Hysteresis;   // fbHysteresis is an instance of the FB_Hysteresis function block
        rActuallyValue:REAL;          // Actual measurement value
        rSetValue:REAL;               // Process setting value
        rTolerance:REAL;              // Deviation setting value
        bOutput AT%QX0.0:BOOL;        // Bit signal output
END_VAR
```

The program body is as follows.

```
fbHysteresis(IN1:=rActuallyValue , VAL:=rSetValue , HYS:=rTolerance , Q=>bOutput);
```

The above program can also be expressed by the following program, and the result is the same.

```
fbHysteresis(IN1:=rActuallyValue , VAL:=rSetValue , HYS:=rTolerance); bOutput:=fbHysteresis.Q;
```

Figure 5-13 Program Execution Results of Hysteresis Function Block

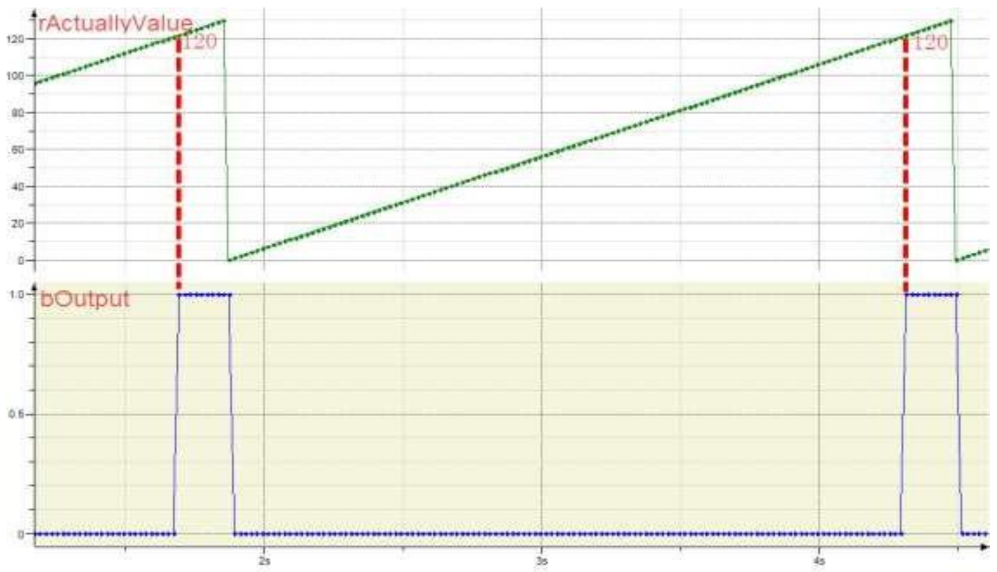


Figure 5-13 shows the results of actual program execution. In the program, rSetValue is set to 100 and rTolerance is set to 20. When the value of rActuallyValue increases from 0 to 120, the bOutput signal is set to TRUE. Then, when rActuallyValue drops to 0, bOutput also becomes FALSE. In theory, when it drops to 80, bOutput will become FALSE.

[Example 5.18] Time delay function block FB_Delay.

The function block FB_Delay is a time delay function block, which is different from the hysteresis function block FB_Hystersis. The time that the output signal lags behind the input signal is called time delay. The controlled objects in the production process are often described by a first-order filter plus a time delay. Here we only introduce the time delay function block, and will not go into detail about the first-order filter.

The transfer function of time delay is as follows:

$$Y(s)={}^{-s\tau}X(s)$$

Assuming the sampling cycle is Ts, after discretization, we get:

$$Y(k)=X(k-N)$$

Where X is the input signal of time delay; Y is the output signal of time delay. Assuming that the sampling cycle used for discretization is Ts, the ratio of the time delay τ to the sampling cycle Ts is the lag number N.

- Variable declaration of the function block FB_Delay

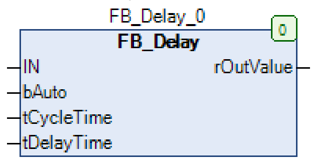
The program uses an array to store input signals, and the array stores sampling data at different times, that is, the first cell stores the sampling value at the time 1×Ts, and the i-th cell stores the sampling value at the time i×Ts. The integer value of the ratio of the time delay τ to the sampling cycle Ts is N (represented by N after the decimal part of N is removed). Therefore, if the input signal is stored in the Nth cell at a certain moment, the output signal after the time delay should be output from the first memory cell.

```
FUNCTION_BLOCK FB_Delay
VAR_INPUT
        IN:REAL;                // Input signal
        bAuto: BOOL;            // Automatic/manual flag signal
        tCycleTime:TIME;        // Sampling cycle
        tDelayTime:TIME;        // Time delay
END_VAR
VAR_OUTPUT
        rOutValue:REAL;         // Output after time delay processing
```

```
END_VAR
VAR
    N:INT;                      // Lag number
    arrValue:ARRAY[0..2047] OF REAL;      // First-in-first-out array stack
    i:INT;                      // Array subscript, used for input
    j:INT;                      // Array subscript, used for output
    fbTrig:R_TRIG;              // Convert the automatic signal into a pulse
    fbTon:TON;
END_VAR
```

After filling in the above input and output parameters, call the function block diagram through the graphical programming language. The schematic effect diagram is shown in Figure 5-14.

Figure 5-14 FB_Delay Function Block Diagram



- Program body of the Function block FB_Delay

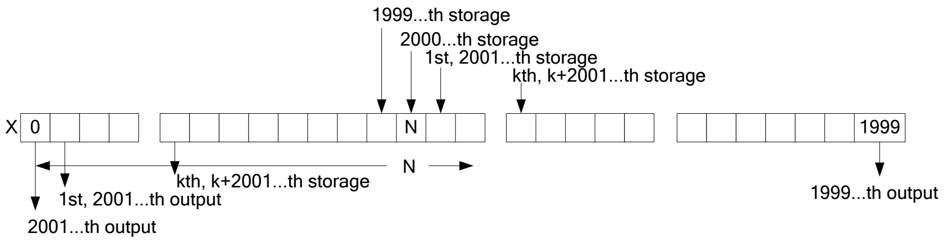
```
N:=TIME_TO_INT(tDelayTime)/TIME_TO_INT(tCycleTime);
fbTrig(CLK:= bAuto);
IF fbTrig.Q THEN
    i:=N;
    j:=0;
END_IF
fbTon(IN:= NOT fbTon.Q , PT:=tCycleTime);
IF fbTon.Q AND bAuto THEN
    i:=(i+1)MOD 2000;
    arrValue[i]:=i;
    j:=(j+1)MOD 2000;
    rOutValue:=arrValue[j];
END_IF
```

The function block body uses two subscript windows to manage the access and output of input and output signals. The input signal data is stored at the i-th subscript address of the array X, and the initial value is equal to the lag number. The output signal is at the j-subscript address of the array X, and the initial output value is equal to 0. The modulo method is used to determine the storage and output address each time, and after each operation, the original address is increased by 1. It is ensured that the next time the operation is executed, the input of this time and the input signals of the previous N times are stored as the output of this time.

The number of array memory cells determines the size of the time delay and is related to the sampling cycle. The larger the time delay is and the smaller the sampling cycle is, the more memory cells are required. Generally, the lag number N can be made larger than the total number of memory cells according to the size of the application.

In this example, the lag number N is required to be less than 2000 (the array length is 2048). In addition, the array memory cell starts from 0, and the actual application starts from the address 0. Figure 5-15 shows the relationship between the input and output windows.

Figure 5-15 Relationship between Input and Output Windows



Notes on using the function block FB_Delay:

- The lag number N is related to the time delay and the sampling cycle. The signal of switching from the running state to the auto state is used as the pulse signal for setting the initial value in the program.
- This function block can be combined with the first-order filter link to simulate the actual production process and conduct control system simulation research.

[Example 5.19] Calculate maximum, minimum, and average values.

In some industrial controls, it is often necessary to calculate the average, maximum, and minimum values of several measured values. The following uses the structured text programming language to implement such an application.

1. Control requirements

It is required to measure the temperatures of 32 points in a kiln. The maximum, minimum, and average temperature values of these 32 points need to be calculated.

2. Programming

The maximum, minimum, accumulated total, and average values are defined in the program respectively. The specific variable definitions are as follows.

```
PROGRAM PLC_PRG
VAR
    rMaxValue:REAL;     // Maximum
    rMinValue:REAL;     // Minimum
    rSumValue:LREAL;    // Accumulated total
    rAvgValue:REAL;     // Average
    arrInputBuffer AT%IW100 :ARRAY[1..32] OF REAL;  // Input source data
    i:INT;
END_VAR
```

The program body is as follows, using the FOR...DO statement to scan all input channels, calculate the average, maximum, and minimum values, and also calculate the total value.

```
rSumValue:=0;
FOR i:=1 TO 32 BY 1 DO
rSumValue:=REAL_TO_LREAL(arrInputBuffer[i])+rSumValue;
    IF arrInputBuffer[i]> rMaxValue THEN
        rMaxValue:=arrInputBuffer[i];
    END_IF
    IF arrInputBuffer[i]< rMinValue THEN
        rMinValue:=arrInputBuffer[i];
    END_IF
END_FOR;
rAvgValue:=rSumValue/32;
```

5.3 Ladder Diagram (LD) and Function Block (FBD)

5.3.1 Introduction to Ladder Diagram and Function Block Diagram Programming Languages

Two graphical programming languages are defined in the IEC 61131-3 standard: namely Ladder Diagram (LD) and Function Block Diagram (FBD). The LD programming language uses a series of rungs to form a ladder diagram to represent the relationship between variables in the industrial control logic system. The FBD programming language uses a series of function blocks to represent the main body of a program organization unit.

- Ladder Diagram (LD)

The ladder diagram originated in the United States and was originally based on a graphical representation of relay logic for programming programmable logic controllers (PLCs). It is one of the most widely used graphical programming languages for PLC programming.

The basic structure of a ladder diagram is as follows:

1. **Power rail:** The left power rail is nominally the start point of power flow; while the right power rail is the end point of power flow. The power flows from left to right along the horizontal rungs, providing power through various contacts, functions, function blocks, coils, etc.
2. **Contact and coil:** A contact represents the state of a Boolean variable (such as the state of a switch); while a coil represents the state of an actual device (such as the startup state of a motor). Each contact and coil corresponds to a memory cell in the PLC memory.
3. **Function and function block:** It corresponds to the functions or function blocks in the standard library of IEC1131-3 or defined by users.

Ladder Diagram logic solution: According to the state and logical relationship of each contact in the ladder diagram, the state of the programming element corresponding to each coil in the diagram is found. This process is called the logic solution of the ladder diagram.

Soft relay: In the ladder diagram, some programming elements use the names of traditional relays, such as coils and contacts, but they are actually memory cells (soft relays). Each soft relay corresponds to a memory cell of the image register in the PLC memory.

TRUE/ON state**:** If the memory cell is “TRUE”, the coil of the corresponding soft relay is “energized”, the normally open contact is engaged, and the normally closed contact is disengaged.

FALSE/OFF state**:** If the memory cell is "FALSE", the state of the coil and contact of the corresponding soft relay is opposite to the above.

- Function Block Diagram (FBD)

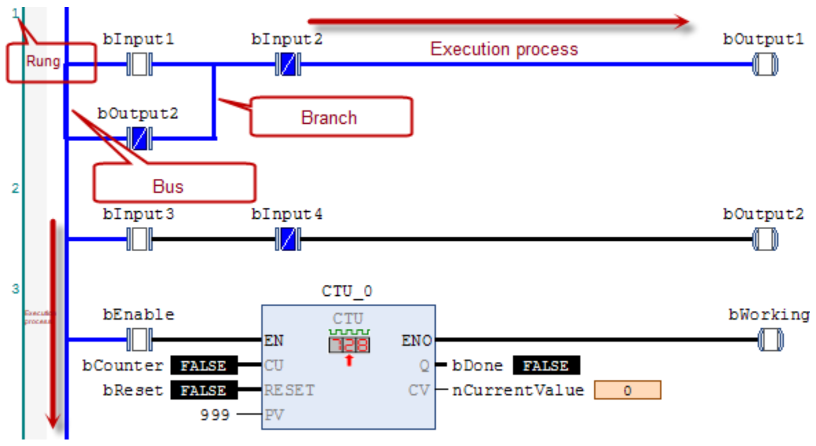
Function block diagrams are used to describe functions, function blocks, and program behaviors, and can also describe the behaviors of steps, actions, and transitions in Sequential Function Charts. A function block diagram is very similar to a signal flow diagram in an electrical diagram. In a program, it can be seen as the flow of information between two process elements. Function block diagrams are widely used in the field of process control.

Function blocks are represented by rectangular blocks. Each function block has at least one input terminal on the left side and at least one output terminal on the right side. The type name of a function block is usually written inside the block, but the instance name of a function block is usually written at the top of the block. The input and output names of a function block are written in the corresponding places of the input and output points in the block.

5.3.2 Program Execution Sequence

The execution sequences of ladder diagram and function block diagram are similar, both executed from left to right and from top to bottom, as shown in Figure 5-16.

Figure 5-16 Program Execution Sequence



Bus: The ladder diagram uses a network structure which is bounded by the left bus. When analyzing the logical relationship of the ladder diagram, in order to learn from the analysis method of the relay circuit diagram, we can imagine that there is a DC power supply voltage between the left and right bus (left bus and right bus), positive on the left and negative on the right, and there is an “energy flow” from left to right between the bus. The right bus is not displayed.

Rung: It is the smallest unit in the ladder diagram network structure. A logic-related network starting from the input condition to a coil is called a rung. In the editor, rungs are arranged vertically. In CoDeSys, each rung is represented by a label on the left, contains input and output instructions, and is composed of logical or arithmetic expressions, programs, and jump, return, or function block call instructions. To insert a rung, you can use the insert instruction or drag it from the Toolbox. Elements contained in a rung can be copied or moved by dragging and dropping them in the editor. When the ladder diagram is executed, it starts from the rung with the smallest label, determines the state of each element from left to right and the states of the link elements on the right, and executes one by one to the right. The execution results are output by the execution control element. Then it proceed to the next rung. Figure 5-16 shows the execution process of a ladder diagram.

Energy flow: The bold blue line on the left side of Figure 5-16 is the energy flow, which can be understood as an imaginary "conceptual current" or "power flow" flowing from left to right. This direction is consistent with the order of logical operations when the user program is executed. Energy flow can only flow from left to right. Using the concept of energy flow can help us better understand and analyze ladder diagrams.

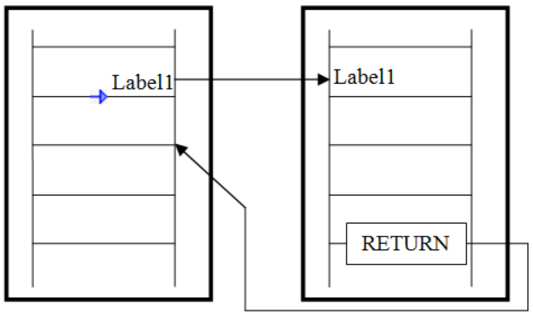
Branch: When a branch appears in a ladder diagram, the state of each graphical element is analyzed in the same order from top to bottom and from left to right. The states of the link elements on the right side of the vertical link elements are determined according to the above-mentioned relevant regulations, so as to execute the evaluation process one by one from left to right and from top to bottom. In ladder diagrams, evaluation without feedback paths is not very clear. All external input values associated with these contacts must be evaluated before each rung.

5.3.3 Execution Control

Jump and Return

When the jump condition is met, the program jumps to the rung marked with Label and starts execution until this part of the program runs to RETURN, then returns to the original rung and continues execution. Its structure diagram is shown in Figure 5-17.

Figure 5-17 Jump Instruction Execution Process

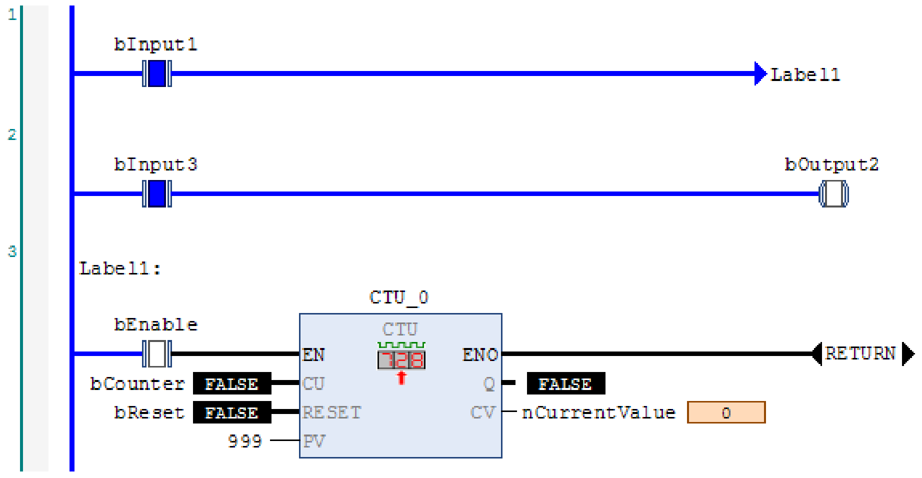


When the program is executed to Label1 on the left side of Figure 5-17, the program starts to execute the jump and jumps directly to the right side of Figure 5-17 to find the program segment marked with Label1, and then starts executing the following program until the program runs to RETURN. At this time, the jump program is completed and returns to the main program loop on the left side of the figure.

The jump and return instructions using ladder diagrams in CoDeSys are shown in Figure 5-18.

[Example 5.20] Example of program execution using a jump instruction.

Figure 5-18 Execution of a Jump Instruction



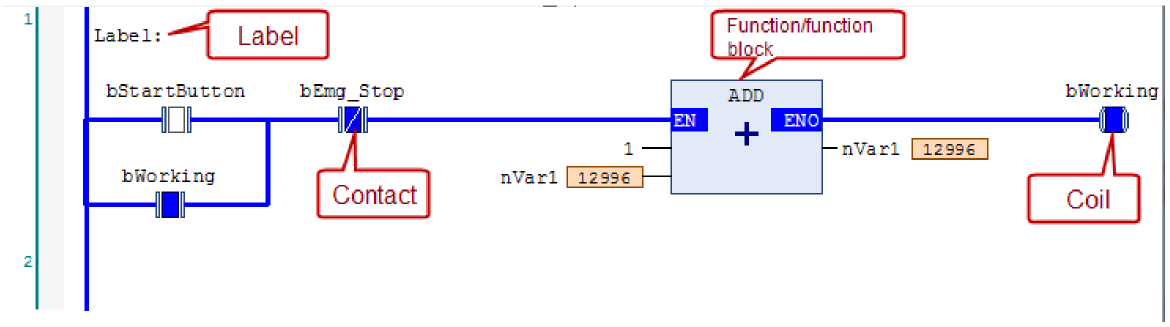
As shown in Figure 5-18, when bInput1 is set to TRUE, the main program executes the jump statement. According to Label1, the program jumps to the Label1 program segment in Rung 3. It is not difficult to see from the figure that although bInput3 in Rung 2 is set to ON, bOutput2 will never be set to TRUE because the program directly skips the statement. bOutput2 will be TRUE only if bInput1 is FALSE and bInput3 is TRUE.

5.3.4 Link Element

The ladder diagram language in IEC1131-3 reasonably absorbs and draws lessons from the ladder diagram languages f various PLC manufacturers, and uses the basically consistent graphic symbols with those of various PLC manufacturers. The view of the ladder diagram editor is shown in Figure 5-19. The main graphic symbols in IEC 61131-3 include the following.

- Basic connection: power rails, link elements
- Contacts: normally open contacts, normally closed contacts, positive transition-sensing contacts, negative transition-sensing contacts
- Coils: general coils, negated coils, set (latch) coils, reset (unlatch) coils, holding coils, set holding coils, reset holding coils, positive transition-sensing coils, negative transition-sensing coils
- Functions and function blocks: standard functions and function blocks as well as user-defined function blocks

Figure 5-19 Ladder Diagram Editor

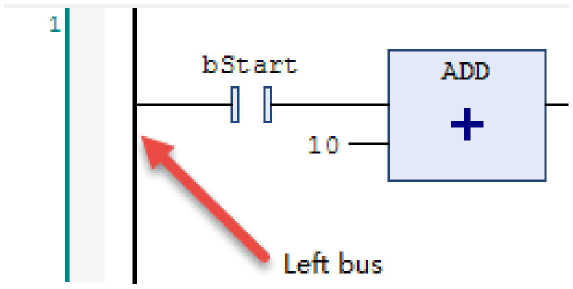


5.3.4.1 Line Element

1. Power rail (bus)

The graphic element of a power rail in a ladder diagram is also called bus. Its graphic representation is located on the left side of the ladder diagram, and it can also be called the left power bus. The left bus graph is shown in Figure 5-20.

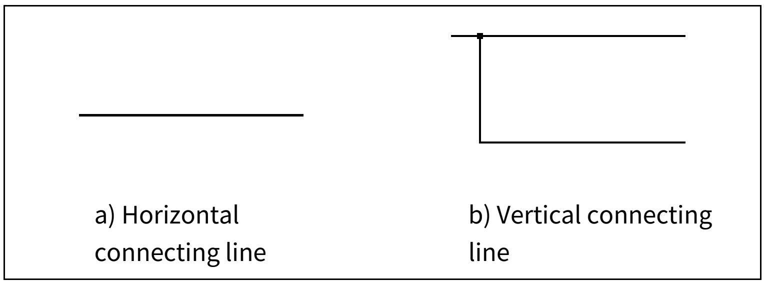
Figure 5-20 Left Bus



2. Connecting line

In a ladder diagram, each graphic symbol is connected by a connecting line. The graphic symbols of connecting lines include horizontal lines and vertical lines, which are the most basic elements of a ladder diagram. The horizontal and vertical connecting lines are shown in Figure 5-21.

Figure 5-21 Connecting Line



3. Transmission rules for link elements

The state of a link element is transmitted from left to right, realizing the flow of energy. The state transmission follows the rules below: The state of the link element connected to the left power rail is TRUE at any time, which indicates that the left power rail is the start point of the energy flow. The right power rail is analogous to zero potential in an electrical diagram.

A horizontal link element shall be indicated by a horizontal line. A horizontal link element transmits the state of the element on its immediate left to the element on its immediate right.

A vertical link element is always connected to one or more horizontal link elements, that is, the vertical link element shall consist of a vertical line intersecting with one or more horizontal link elements on each side. The state of the vertical link element is represented by the state or operation of each horizontal link element on its left side.

Therefore, the state of the vertical link shall be:

FALSE if the states of all the attached horizontal link elements to its left are FALSE;

TRUE if the state of one or more of the attached horizontal link elements to its left is TRUE.

The state of the vertical link element shall be transmitted to all of the attached horizontal link elements on its right, but shall not be transmitted to any of the attached horizontal links on its left.

[Example 5.21] Examples of link elements and their state transmission.

Figure 5-22 Examples of Link Elements and Their States

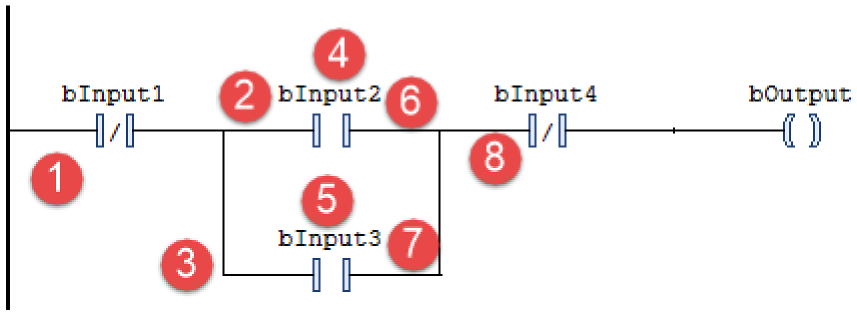


Figure 5-22 shows examples of link elements and their states. The link element 1 is connected to the left power rail in a TRUE state. The link element 2 is connected to the link element 1 and its state is transmitted from the link element 1, so its state is TRUE. The link element 3 is a vertical link element and connected to the horizontal link element 1 in a TRUE state.

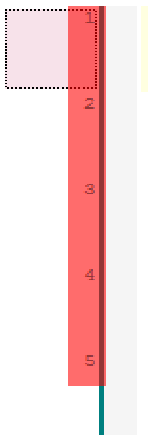
The link elements 2 and 3 transmit the states of link elements 4 and 5 respectively. Since the variables bInput2 and bInput3 corresponding to graphic elements 4 and 5 are normally open contacts, the states of link elements 6 and 7 become FALSE after being transmitted by the graphic elements; the states of all the link elements on the left side of the link element 8 are FALSE.

The input and output data types of a link element must be the same. In the standard, the data types of graphic elements such as contacts and coils are not limited to the Boolean type. Therefore, the input and output data types of a link element must be the same to ensure correct state transmission.

5.3.4.2 Rung

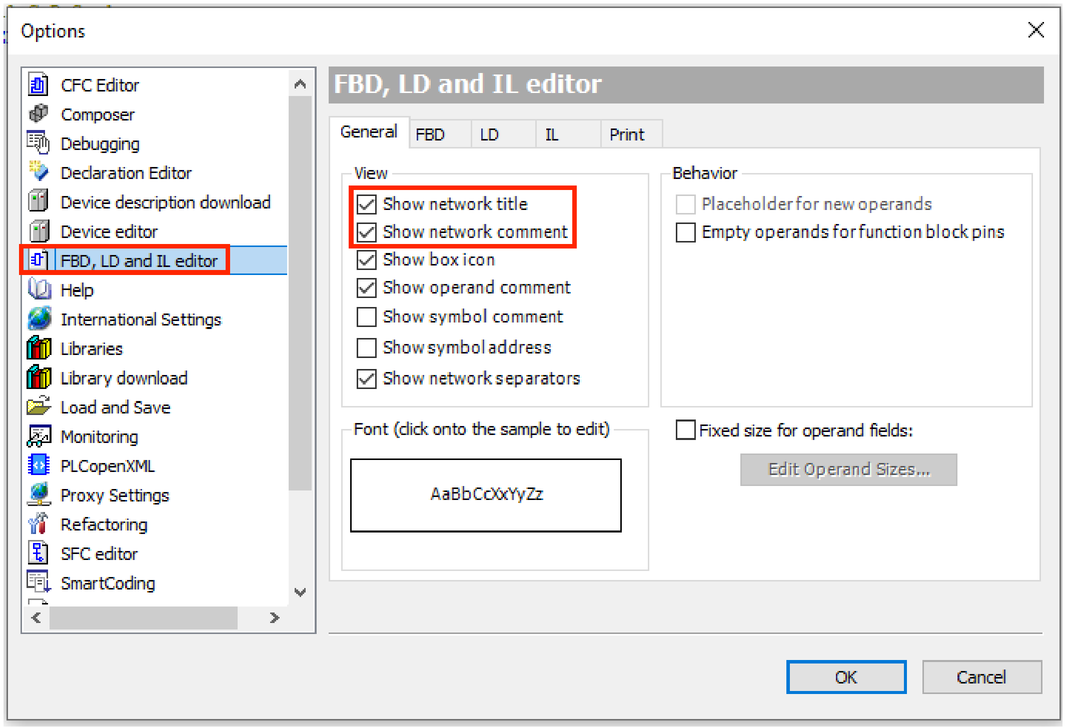
Rungs are the basic entities of LDs and FBDs. In the LD/FBD editor, rungs are arranged in numerical order. Each rung starts with a label on the left and has a structure consisting of logical or arithmetic expressions, programs, functions, and function block call, jump, or return instructions. The schematic diagram of rungs is shown by the red shaded part in Figure 5-23. The rungs are arranged in sequence by serial number.

Figure 5-23 Rung View



Rung annotation: A rung can also be assigned a title, annotation, and label. The title and annotation areas can be enabled or disabled via the “Options” → “FBD, LD, and IL Editor” dialog box, as shown in Figure 5-24.

Figure 5-24 Rung Title, Annotation, and Label Functions



Options

FBD, LD and IL editor

General | FBD | LD | IL | Print

View
- [✓] Show network title
- [✓] Show network comment
- [✓] Show box icon
- [✓] Show operand comment
- [ ] Show symbol comment
- [ ] Show symbol address
- [✓] Show network separators

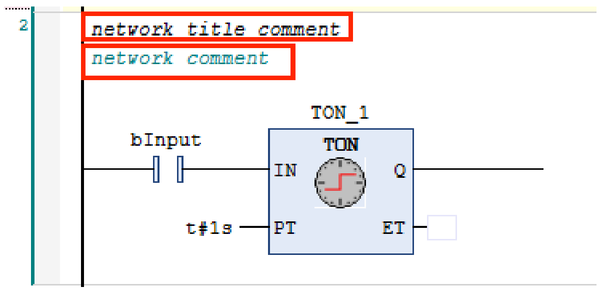
Behavior
- [ ] Placeholder for new operands
- [ ] Empty operands for function block pins

Font (click onto the sample to edit)
AaBbCcXxYyZz

- [ ] Fixed size for operand fields:

If the above option is activated, you can open an editable field for the title by clicking below the upper border of the rung with the mouse. If you want to enter an annotation, you need to open the corresponding editable field below the title field. Annotations can be made in multiple lines. You can start a new line by pressing the Enter key, and terminate the input of annotation text by pressing [Ctrl]+[Enter]. Figure 5-25 shows how to add a rung title and annotation.

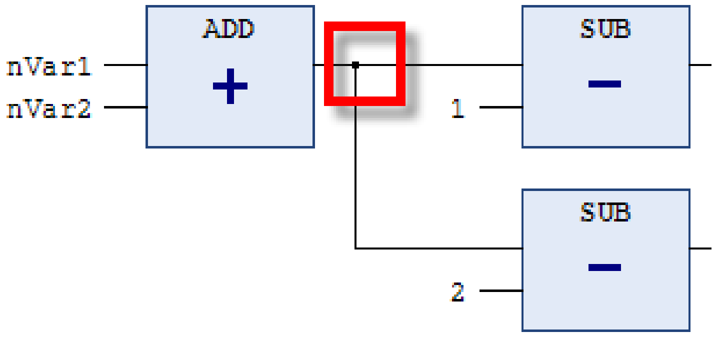
Figure 5-25 Rung Title and Annotation



Rung title: You can switch to the “Annotation State” via “Switch Rung Annotation State”. Then, the rung will be displayed for annotation and will not be executed.

Rung branch: You can create a “sub-rung” by inserting “ Branch ”in the toolbox, as shown in Figure 5-26, in which the branch function is used.

Figure 5-26 Create Sub-rungs through the Branch Function



5.3.4.3 Label

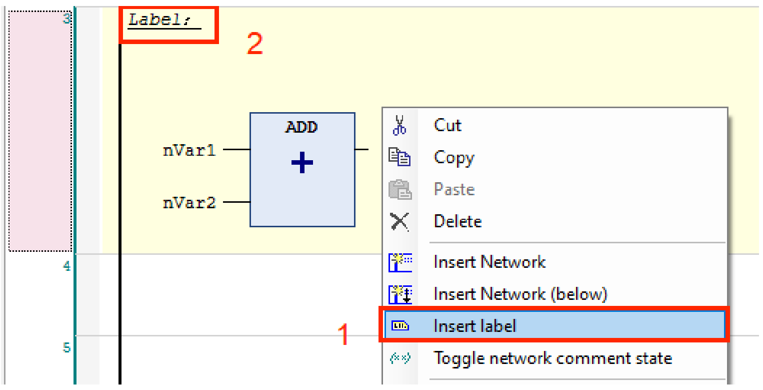
A label is an optional identifier and its address can be determined when a jump is defined. It can contain any characters.

In the rung area, each FBD, LD, or IL rung has a text entry field to define a label. A label is an optional identifier for a rung that can be addressed when a jump is defined, and it can contain any sequence of characters.

Use in the FBD

If you make a right-click in a blank space in the rung area and select "Insert Label", as indicated by 1 in Figure 5-27, Label: will pop up in 2 and you can edit it.

Figure 5-27 Add a Rung Label



5.3.4.4 Contact

1. Contact type

A contact is a graphic element which transmits a state to the horizontal link element on its right side in a ladder diagram. The contact in the ladder diagram follows the contact terminology in an electrical diagram and is used to indicate the state change of a Boolean variable.

Contacts can be divided into normally open contacts (NOs) and normally closed contacts (NCs). Normally open contacts are disengaged under normal operating conditions and their state is FALSE. Normally closed contacts are engaged under normal operating conditions and their state is TRUE. Table 5-3 lists commonly used graphic symbols of contacts in CoDeSys ladder diagrams and their descriptions.

Table 5-3 Graphic Symbols and Descriptions of Contact Elements

Type	Graphic Symbol	Description
Normally Open Contact	-| |-	If the current Boolean variable value corresponding to the contact is TRUE, the contact is engaged; if the state of the link element on the left side of the contact is TRUE, the state TRUE is transmitted to the right side of the contact, making the state of the link element on the right side TRUE. Conversely, when the Boolean variable value is FALSE, the state of the right link element is FALSE.
Normally Closed Contact	-|/|-	If the current Boolean variable value corresponding to the contact is FALSE, the normally closed contact is engaged. If the state of the link element on the left side of the contact is TRUE, the state TRUE is transmitted to the right side of the contact, making the state of the link element on the right side TRUE. Conversely, when the Boolean variable value is TRUE, the contact is disengaged and the state of the right link element is FALSE.
Insert Right Contact	→ -| |-	Multiple contacts can be connected in series by inserting contacts on the right side. When the multiple contacts in series are all engaged,

Type	Graphic Symbol	Description
		the last contact can transmit the TRUE state.
Insert Normally Open Contact in Parallel		Multiple contacts can be connected in parallel, and normally open contacts can be inserted in parallel on the lower side of the contacts. Only one of two parallel contacts needs to be TRUE for the parallel line to transmit the TRUE state.
Insert Normally Closed Contact in Parallel		Multiple contacts can be connected in parallel, and normally closed contacts can be inserted in parallel on the lower side of the contacts. A normally closed contact is defaulted to engaged. If the current Boolean variable value corresponding to the contact is FALSE and the state of the link element on the left is TRUE, the right side of the parallel contact transmits the TRUE state.
Insert Upper Normally Open Contact in Parallel		Multiple contacts can be connected in parallel, and normally open contacts can be inserted in parallel on the upper side of the contacts. Only one of two parallel contacts needs to be TRUE for the parallel line to transmit the TRUE state.

2. State transmission rules

Based on the state of a contact and the state of the link element on the left side of the contact, the state of the graphic symbol on the right side can be determined according to the following rules.

When the state of the graphic element on the left side of the contact is TRUE, its state can be transmitted to the graphic element on the right side of the contact according to the following principles:

- If the state of the contact is TRUE, the state of the graphic element on its right side is TRUE.
- If the state of the contact is FALSE, the state of the graphic element on its right side is FALSE.

When the state of the graphic element on the left side of the contact is FALSE, no matter what the state of the contact is, its state cannot be transmitted to the graphic element on its right side, that is, the state of the graphic element on its right side is FALSE.

When the graphic symbol on the left side of the contact changes from FALSE→TRUE, its associated variables also change from FALSE→TRUE, and the state of the graphic symbol on the right side of the contact changes from FALSE→TRUE, remains TRUE for one cycle, and then becomes FALSE, which is called rising edge triggering.

When the graphic symbol on the left side of the contact changes from TRUE→FALSE, its associated variables also change from TRUE→FALSE, and the state of the graphic symbol on the right side of the contact changes from TRUE→FALSE, remains FALSE for one cycle, and then becomes TRUE, which is falling edge triggering.

5.3.4.5 Coil

1. Coil type

A coil is a graphic element in a ladder diagram. The coil in the ladder diagram follows the coil terminology in an electrical diagram and is used to indicate the state change of a Boolean variable.

According to different characteristics of coils, they can be divided into momentary coils and latched coils, and latched coils are further divided into set coils and reset coils. Table 5-4 lists commonly used graphic symbols of coils in CoDeSys ladder diagrams and their descriptions.

Table 5-4 Graphic Symbols and Descriptions of Coil Elements

Type	Graphic Symbol	Description
Coil		The state of the left link element is transmitted to the associated Boolean variable and the right link element. If the state of the link element on the

Type	Graphic Symbol	Description
		left side of the coil is TRUE, the Boolean variable of the coil is TRUE; otherwise, it is FALSE.
Set Coil	(S)	There is an S in the coil. When the state of the left link element is TRUE, the Boolean variable of the coil is set and remains set until it is reset by the reset coil.
Reset Coil	(R)	There is an R in the coil. When the state of the left link element is TRUE, the Boolean variable of the coil is reset and remains reset until it is set by the set coil.

2. Coil state transmission rules

A coil is a graphic element in a ladder diagram that transmits the state of the horizontal or vertical link element on its left side to the horizontal link element on its right side without modification. During the transmission process, the states of the left associated variables and direct addresses are stored in appropriate Boolean variables. Conversely, a negated coil is a graphic element in a ladder diagram that first inverts the state of the horizontal or vertical link element on its left side and then transmits it to the horizontal link element on its right side.

A set/reset coil maintains the state of the horizontal link element on its left side for one evaluation cycle at the moment when the state changes from FALSE to TRUE or from TRUE to FALSE, and transmits the state of the horizontal link element on its left side to the horizontal link element on its right side at other times.

A rising edge/falling edge jump coil maintains its associated variable for one evaluation cycle at the moment when the state of the horizontal link element on its left side changes from FALSE to TRUE or from TRUE to FALSE, and transmits the state of the horizontal link element on its left side to the horizontal link element on its right side at other times.

There is no rule on the right side that only one element can be linked, so you can expand elements on the right side to simplify the program. For example, other coils can be connected in parallel on the right side, as shown in [Example 5.22].

[Example 5.22] Transmission of coil state.

Figure 5-28 Transmission of Coil State

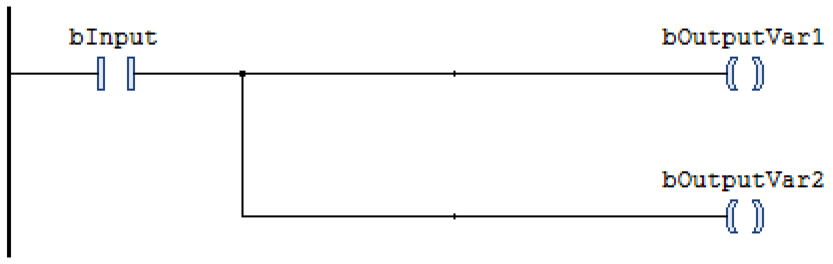


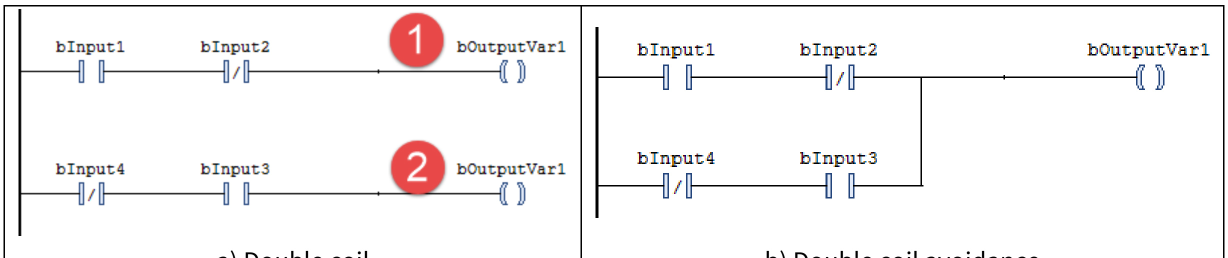
Figure 5-28 shows the coil state transmission process. In the figure, when the contact bInput is closed, the state of the link element on its right side is TRUE, and it is connected to the coils bOutputVar1 and bOutputVar2 after passing through the horizontal and vertical link elements respectively, and also sets their states to TRUE.

3. Double-coil

The so-called double-coil means that the same coil is used twice or more in the user program. This phenomenon is called double coil output. In Figure 5-29 a), there are two coils with the output variable "bOutputVar1". In the same scan cycle, the logical operation results of the two coils may be exactly opposite, that is, one coil of the variable bOutputVar1 may be “powered on” while the other may be “powered off”. For the control of the variable bOutputVar1, what really works is the state of the last coil of the variable bOutputVar1.

In addition to affecting the external load, the on/off state of the coil of the variable bOutputVar1 may also affect the state of other variables in the program through its contact. Therefore, double coil output should be avoided as much as possible, and the parallel connection method as shown in Figure 5-29 b) should be used to solve the double coil problem.

Figure 5-29 Double Coil Example



a) Double coil b) Double coil avoidance

As long as it can be ensured that only the logical operation corresponding to one of the coils is executed in the same scan cycle, such double coil output is allowed. The following 3 situations allow double coil output.

- In two program segments with opposite judgment conditions (such as automatic program and manual program), double coil output is allowed, that is, the coil of the same variable can appear once in each of the two program segments. In fact, the PLC only executes one coil output instruction of the double coil element in the program segment being processed.
- In two subprograms with opposite calling conditions (such as automatic program and manual program), double coil output is allowed. That is, the coil of the same variable can appear once in each of the two subprograms. The instructions in a subprogram are only executed when the subprogram is called, and are not executed if the subprogram is not called.
- To avoid double coil output, the set/reset instruction can be used multiple times for the same variable.

5.3.4.6 Function and Function Block Calls

If you want to call a function or function block, you must use an operation block, which can represent all POUs, including function blocks, functions, and even programs. Function blocks include timers, counters, etc., and can be inserted into FBD and LD rungs. Operation blocks can have arbitrary inputs and outputs. For detailed description of graphic symbols of functions and function blocks, see Table 5-5.

Users can insert function blocks and programs along with contacts and coils. In the network, they must have one input and one output with Boolean values and can be used like contacts at the same position, that is, on the left side of the LD network.

Table 5-5 Graphic Symbols and Descriptions of Function and Function Block

Type	Graphic Symbol	Description
Insert Operation Block		Insert a function or function block, and select the function or function block you want to use with the mouse according to the pop-up dialog box . It is suitable for users who are not familiar with functions and function blocks.
Insert Null Operation Block		Insert a rectangular block directly and directly enter the name of the function or function block at the "???" position. It is suitable for users who are familiar with functions and function blocks.
Insert Operation Block with EN/ENO		Only when EN is TRUE will the function or function block be executed and allowed to transmit the state downstream. It is suitable for users who are not familiar with functions and function blocks.
Insert Null		Insert a rectangular block with EN/ENO and directly enter the name of

Type	Graphic Symbol	Description
Operation Block with EN/ENO		the function or function block at the "???" position. Only when EN is TRUE will the function or function block be executed and allowed to transmit the state downstream. It is suitable for users who are familiar with functions and function blocks.

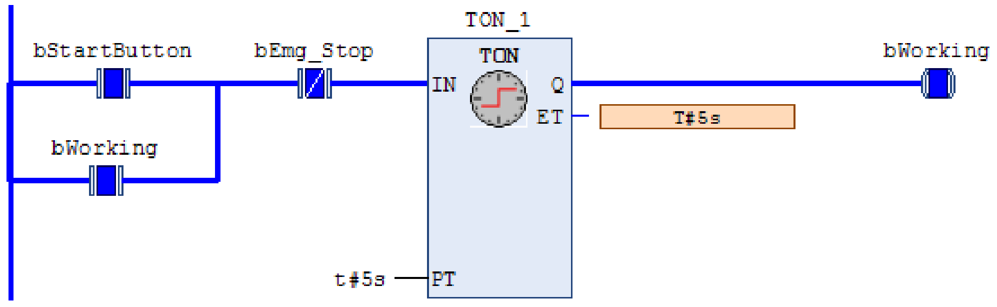
The ladder diagram programming language supports calling functions and function blocks. When calling functions and function blocks, please note the following:

1. In a ladder diagram, functions and function blocks are represented by a rectangular box. A function can have multiple input parameters but only one return parameter. A function block can have multiple input parameters and multiple output parameters.
2. The inputs are listed on the left side of the rectangle box while the outputs are listed on the right side of the rectangle box.
3. The names of functions and function blocks are displayed in the upper middle part of the box. Function blocks need to be instantiated, and the instance names are listed in the upper middle part outside the box. The instance name of a function block is used as its unique identifier in the project.
4. To ensure that energy can flow through a function or function block, each called function or function block should have at least one input and output parameter. To execute a connected function block, at least one Boolean input must be connected to the vertical left power rail via a horizontal rung.
5. When calling a function block, you can directly fill in the actual parameter value at the external connecting line of the function block of the internal formal parameter variable name.

[Example 5.23] Actual parameter setting for a function block call.

In Figure 5-30, the TON delayed ON function block is called, where TON_1 is the instance name of the instantiated function block TON. The input formal parameter PT of the function block is set to t#5s. Q and ET are output formal parameters. When output formal parameters are not needed, such as ET in the example, the variable can be left unconnected.

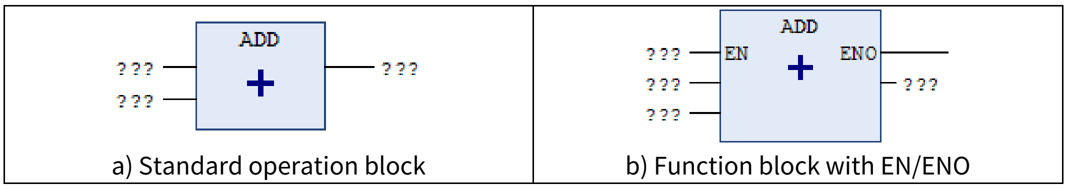
Figure 5-30 Actual Parameter Setting for a Function Block Call



It can be seen that the output parameter Q of the function block TON is connected to the coil bWorking. It means that when the contact bStartButton is TRUE and bEmg_Stop is FALSE for more than 5 s, bWorking is TRUE. When bEmg_Stop is off, namely TRUE, bWorking is FALSE.

If there are no dedicated input and output parameters for EN and ENO, the functions and function blocks are automatically executed and their states are transmitted downstream. In [Example 5.23], a function block with EN and ENO is called. In the Toolbox, you can choose to insert a standard operation block " Box ", or a function block " Box with EN/ENO " with EN/ENO. You can drag and drop to copy or move it in the editor. Figure 5-31 a) and b) are diagrams comparing the standard operation block and the operation block with EN/ENO.

Figure 5-31 Comparison of Two Types of Operation Blocks in the FBD



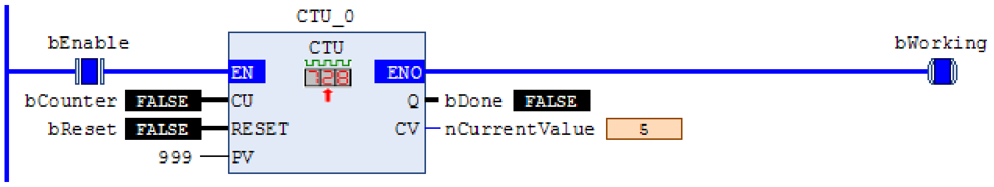
a) Standard operation block b) Function block with EN/ENO

In Figure 5-31 a), as long as the front-end conditions are met, the function block will be executed directly, while in b), the function block will be executed only when EN is TRUE. Otherwise, even if all the front-end conditions are met, the function block will not be executed by the program. If the input signal of EN in b) is set to the constant "TRUE", the effects of a) and b) are exactly the same.

[Example 5.24] Call a function block with EN and ENO.

Figure 5-32 shows a function block with EN and ENO. the Boolean input bEnable is used to start the counter function block CTU_0, and bWorking is used as the state variable signal that the function block is enabled.

Figure 5-32 Call of a Function Block with EN and ENO



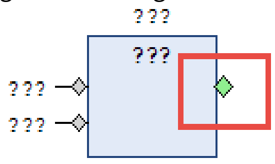
It can be seen that when bCounter has a rising edge trigger signal, the formal parameter output variable CV is incremented by 1.

- When EN is FALSE, the operation defined by the function block body is not executed and the value of ENO is also FALSE accordingly.
- When the value of ENO is TRUE, it means that the function block is being executed.

5.3.4.7 Assignment

The assignment function can be understood as the assignment of inputs/outputs to operation blocks. In the Toolbox, you can choose to insert the " VAR Assignment " tool and drag it to the editable field of the program. Then, a small gray diamond pattern will appear at the input and output interface corresponding to the operation block in the editable field. Readers can directly drag it to the interface. After insertion, the text string "???" can be replaced with the name of the variable to be assigned, or you can use the ... button to call the "Input Assistant". At this time, the assignment of the input/output interface variables of the operation block has been completed. The assignment view is shown in Figure 5-33.

Figure 5-33 Assignment View



5.3.4.8 Jump Execution

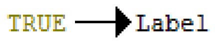
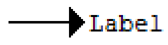
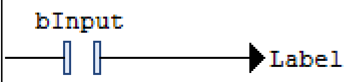
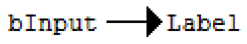
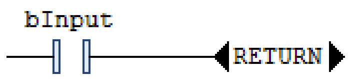
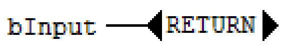
Jump execution control element: A jump execution control element is represented by a Boolean signal line terminated in a double arrowhead. The signal line for a jump condition originates at a Boolean variable, at a Boolean output of a function or function block, or on the power flow line of a ladder diagram.

Jumps are divided into conditional jumps and unconditional jumps.

When a jump signal originates at a Boolean variable or at a Boolean output of a function or function block, the jump is a conditional jump. A jump occurs only when program control executes to the jump signal line of the designated network label and the Boolean value is TRUE.

If the jump signal line originates on the left power rail line of a ladder diagram, the jump is unconditional. In the function block diagram programming language, if a jump occurs when the Boolean constant is 1, the jump is also unconditional. The graphic symbols of jump control elements are listed in Table 5-6.

Table 5-6 Graphic Symbols of Jump Control Elements

Execution Control Type		Graphic Symbol of Execution Control Element	Description
Unconditional Jump	LD language		Unconditional jump to Label directly
	FBD language		
Conditional Jump	LD language		When bInput is 1, conditional jump to Label
	FBD language		
Conditional Return	LD language		When bInput is 1, the conditional jump returns
	FBD language		

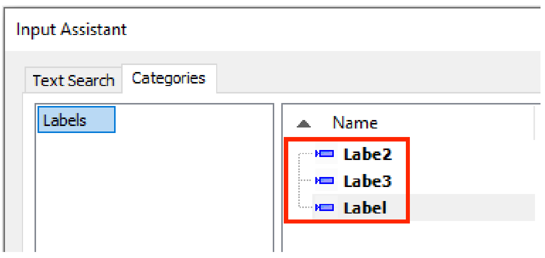
Jump target: In a program organization unit, the jump target is a label within the program organization unit where the jump occurs. It indicates that after the jump occurs, the program will start execution from this target.

Return: Return is divided into two types: conditional return and unconditional return.

The conditional return is applicable to functions and function blocks. When the Boolean input of the conditional return is TRUE, program execution will return to the called entity. When the Boolean input is FALSE, program execution will continue in the normal manner, and an unconditional return is reached by the physical end of the function or function block. As shown in Table 5-6, connecting the RETURN statement directly to the left rail indicates an unconditional return.

Configuration of jump execution: Insert "→" in the Toolbox, and after inserting → representing a jump, replace the automatically entered "???" with the label of the jump target. You can directly enter the label of the target or click the browse key "→I ..." to use the Input Assistant to select one, as shown in Figure 5-34. The system will automatically filter the available labels for users to choose.

Figure 5-34 Jump Input Assistant



[Example 5.25] Jump statement example.

In cylinder control, the extension signal of the cylinder solenoid valve is bExtrent. If the feedback signal bExtrented_Sensor1 of the extension sensor is not received within 5 s after the extension signal bExtrent is sent, it jumps to the alarm program, and the variable declaration and program are as follows.

```
PROGRAM PLC_PRG
```

```
VAR
        bExtrent:BOOL;
        bExtrented_Sensor1:BOOL;
        fb_TON:ton;
END_VAR
```

Figure 5-35 Jump Statement Example Program

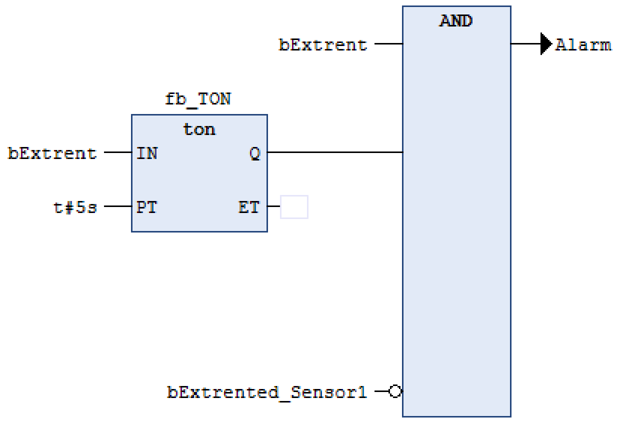


Figure 5-35 shows an example program for the jump statement. Finally, when the output signal Q of the fb_TON function block and the bExtrent signal are met at the same time, the output signal Alarm of the AND logic is set to TRUE.

5.3.5 Application Examples

[Example 5.25] Flashing signal light.

Control requirements

Use timers and logical functions to construct a flashing signal light system. This circuit output can turn the signal light on/off at a certain cycle.

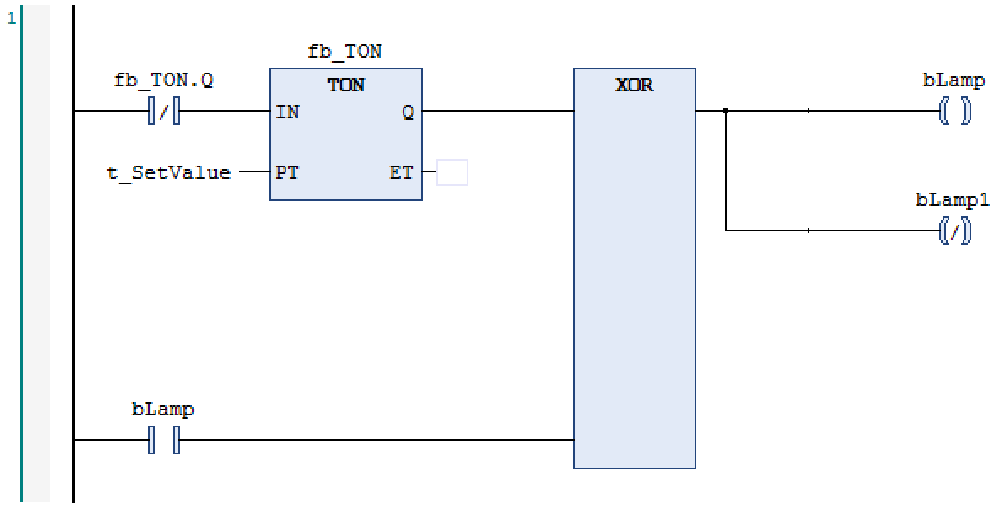
Programming

The program realizes the control requirements of the flashing signal light system by switching bLamp and bLamp1 on/off alternately. The program is implemented by using the ladder diagram shown in Figure 5-36.

Users can use t_SetValue to set the ON/OFF switching time, such as 500 ms. The specific variable definition is as follows.

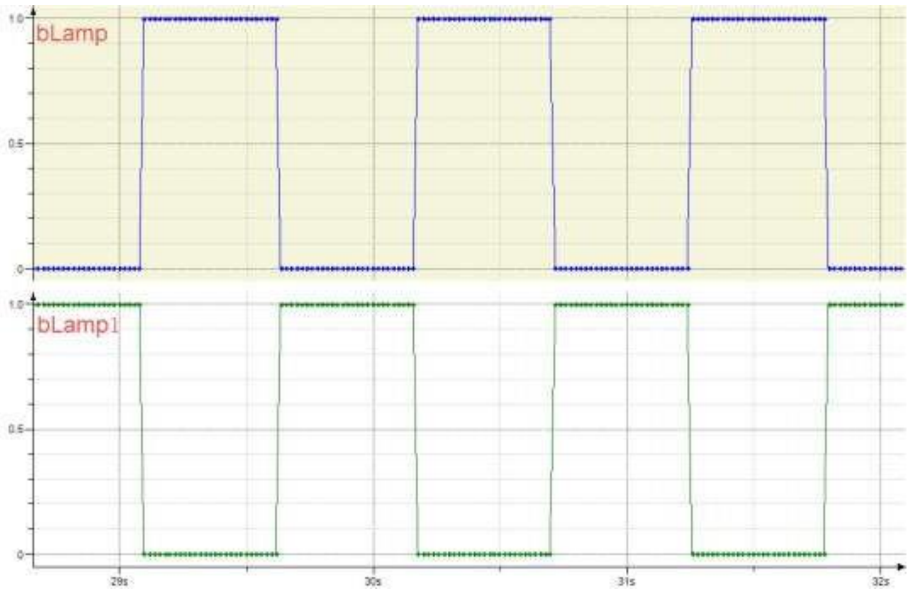
```
PROGRAM
PLC_PRG VAR
        fb_TON:ton;                     //TimeDelay
        t_SetValue:TIME:=t#500ms;       //SetTime
        bLamp AT%QX0.0:BOOL;            //Output0
        bLamp1 AT%QX0.1:BOOL;           //Output1
END_VAR
```

Figure 5-36 Ladder Diagram Program for a Flashing Signal Light System



The output effect is shown in Figure 5-37. The output curves of bLamp and bLamp1 are exactly opposite, and the time for their state switching is exactly 1 s.

Figure 5-37 Output Curves of Flashing Signal Lights



[Example 5.26] pH control system.

Control requirements

pH control is often required in wastewater treatment or fermentation processes. Since the controlled objects of the pH control system have nonlinearity and time delay behaviors, nonlinearity and time delay compensation control schemes are commonly used. However, the following control strategy can also be used in a simple control scheme: when the measured pH value exceeds the set acidity value, wait for a certain period of time and then add alkaline liquid for a certain period of time. When the pH exceeds the set value, the contact PHH is closed. Conversely, when it is less than the set value, the alkali addition valve is bValves1. The control scheme is “Look and Adjust”.

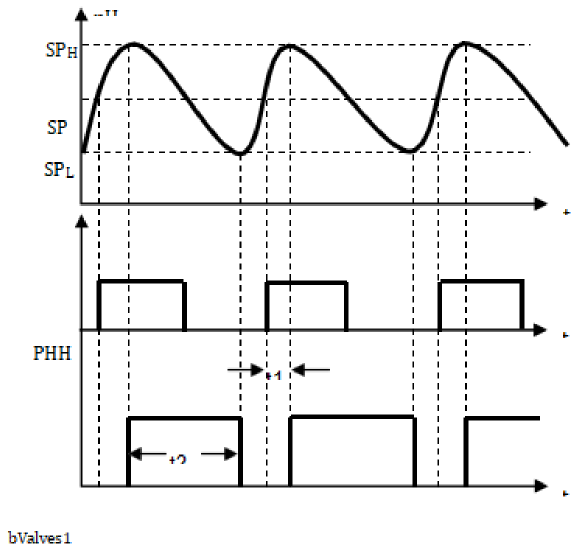
When the pH is controlled in the linear region, it can be assumed that the change in pH during the control process is linear, that is, when alkali or acid is added for neutralization, the change in pH is linear. Generally, when the difference between the set upper limit SPH and the set lower limit SPL is small, a linear relationship is established.

Assuming the time required for the pH value to change from SPL to SPH during the fermentation process is t, and the time required for the pH value to change from SPH to SPL after adding alkali is t2, the time delay can be set to t1=t/2, and the time for the alkali addition control valve to open is t2.

The actual set value for pH control SP = (SPH + SPL) / 2. Reducing the difference between SPH and SPL is beneficial to improving control accuracy.

The startup condition of the alkali addition control valve bValves1 is the expiration of the set time of the timer t1; therefore, t1.Q is used as the startup condition in the program. The stop condition of the alkali addition control valve bValves1 is the expiration of the set time of t2; therefore, t2.Q is used as the stop condition in the program.

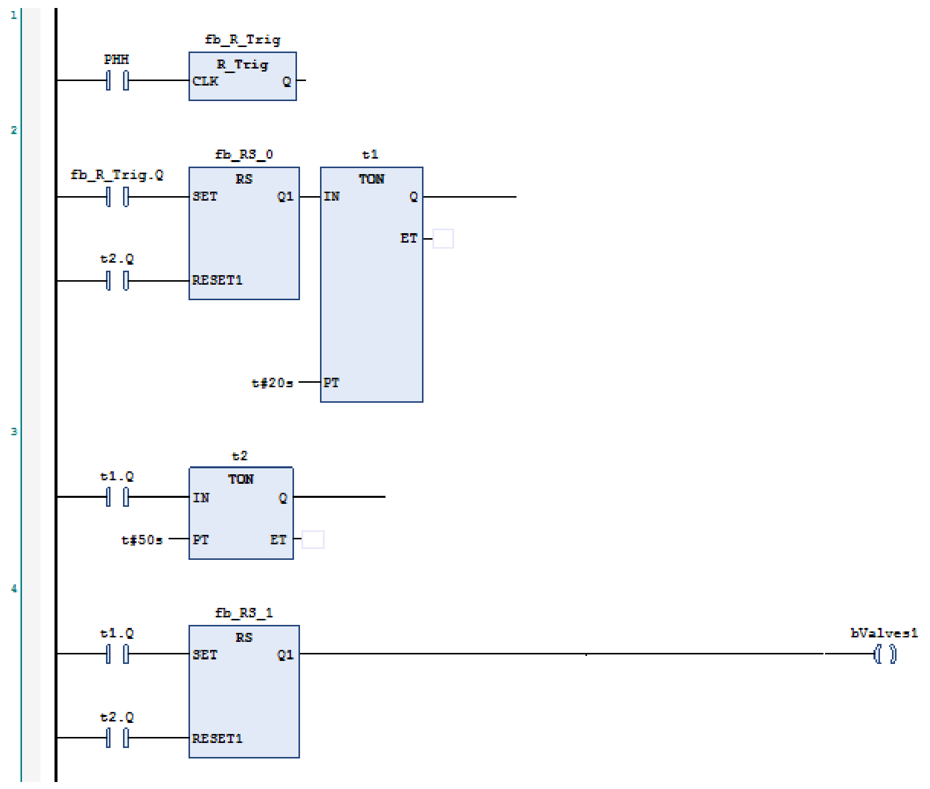
The startup condition of the timer t1 is that pH reaches the set value SP; therefore, the rising edge of the contact PHH is used to trigger the fb_Trigger function block, and its signal is temporarily stored by the RS function block. The startup condition of the timer t2 is the expiration of the set time of the timer t1.



Programming

According to the above control requirements, the pH value control program is written using the ladder diagram programming language, and its variable declaration and program are shown in the figure. Two timers are used in the program.

```
PROGRAM PLC_PRG
VAR
        t1,t2:ton;              // Timers t1, t2
        PHH:BOOL;               // Set value exceeding signal
        bValves1 AT%QX0.0 :BOOL;          // Alkali addition control valve
        fb_R_Trig:R_Trig;
        fb_RS_0,fb_RS_1:RS;
END_VAR
```



5.4 Instruction List (IL)

Instruction List (IL) is a low-level programming language defined in the IEC 61131-3 standard and resembles assembly. It is easy to learn and simple to implement, and can be downloaded directly to the PLC. However, IL lacks effective tools for solving large and complex control problems, so it is rarely used in these scenarios. Nevertheless, as a basic programming language, IL occupies an important position in PLC programming due to its versatility and simplicity.

5.4.1 Introduction to the Instruction List Programming Language

An instruction list (IL) is composed of a sequence of instructions. Each instruction begins in a new line and contains an operator and operands immediately following the operator. The operands are variables and constants defined in the IEC 61131-3 standard.

The instruction list is a line-oriented language, similar to the assembly language. An instruction is a command that can be executed by the PLC. It must be described strictly in lines, and blank lines are allowed as null instructions.

Basic format:

1. Instruction format**:** an operator, the instruction for executing a specific operation; an operand, the variable or constant that the instruction acts on; a label, optional, the instruction is preceded by a label and followed by a colon; annotation, optionally added after the operand.
2. Multiple operands**:** Some operators require several operands, separated by commas.

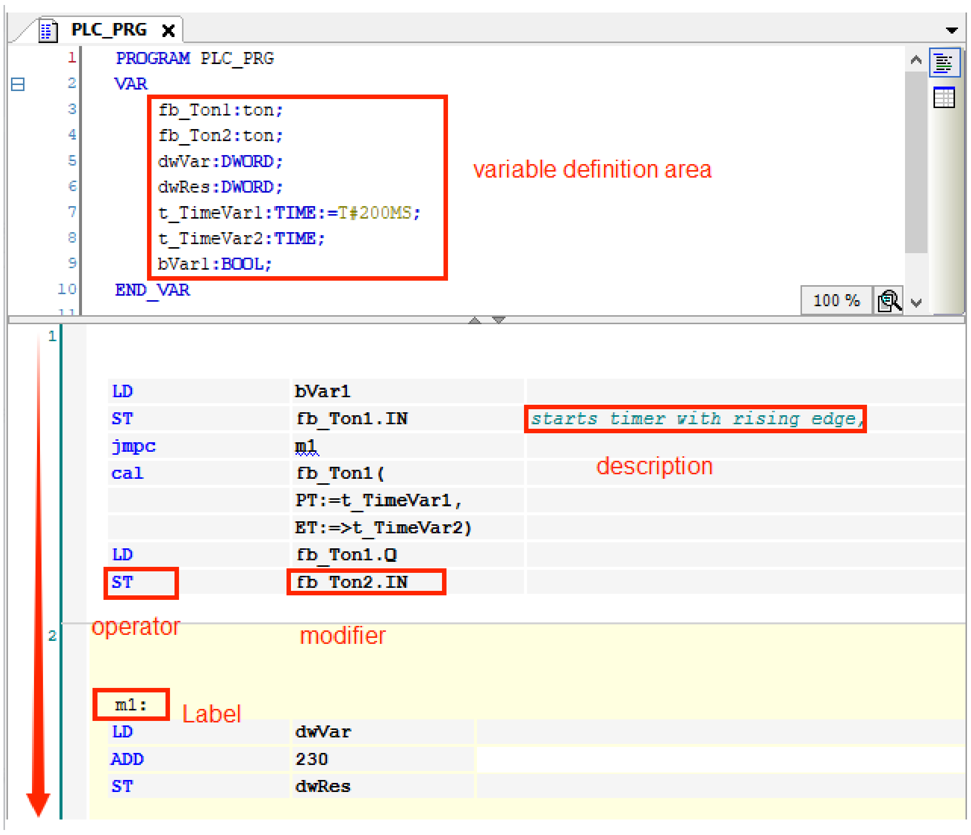
The instruction list programming language has the following characteristics:

- Easy to learn: The instructions are simple to operate and easy to master, suitable for programming small and simple control systems.
- Powerful operators: Operators are used to manipulate variables of all basic data types and call functions and function blocks.
- Direct interpretation and execution: The instruction list programming language can be directly interpreted and executed inside the PLC, which is suitable for most PLC manufacturers.

- Error detection: Most programs written in the instruction list programming language cannot detect errors until they are run.
- Language conversion: Programs written in the instruction list programming language are difficult to convert to other programming languages, while programs written in other programming languages are easy to convert to the instruction list programming language.

Program execution sequence

The instruction list programming language is executed from top to bottom, as shown in the figure below.



```
PROGRAM PLC_PRG
VAR
    fb_Ton1:ton;
    fb_Ton2:ton;
    dwVar:DWORD;
    dwRes:DWORD;
    t_TimeVar1:TIME:=T#200MS;
    t_TimeVar2:TIME;
    bVar1:BOOL;
END_VAR
```

```
LD      bVar1
ST      fb_Ton1.IN         starts timer with rising edge
jmpc    m1
cal     fb_Ton1(
        PT:=t_TimeVar1,
        ET:=>t_TimeVar2)
LD      fb_Ton1.Q
ST      fb_Ton2.IN

m1:
LD      dwVar
ADD     230
ST      dwRes
```

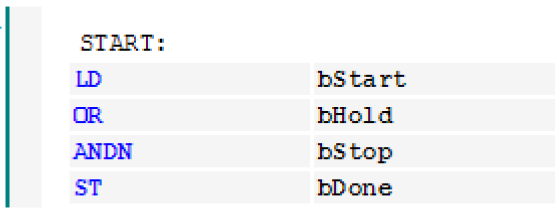
Programming example in the IL editor

Instruction format

In the instruction list programming language, instructions have the following format.

Label: Operator/Function Operand Annotation

[Example 5.27] Use the instruction list to realize the start, operation, and stop control of a motor.



```
START:
LD      bStart
OR      bHold
ANDN    bStop
ST      bDone
```

The program in [Example 5.27] is used to perform start, operation, and stop control on the motor of a device. In the program, the label is START, and the first line of instruction stores the result of the variable bStart in the accumulator. The second line of instruction is used to perform a logical OR operation on the result of the first line of instruction and the bHold output hold signal, and the result is still overwritten in the accumulator. The third line of instruction is used to perform a logical AND operation on the negated result of the second line of instruction and the stop signal bStop, and the result is still stored in the accumulator. The fourth line of instruction is used to output the result in the current accumulator to the variable bDone.

5.4.2 Link Element

An instruction list is composed of a sequence of instructions. Each instruction begins on a new line and contains an operator with optional modifiers, and, if necessary for the particular operation, one or more operands separated by commas. Table 5-7 lists the operators and modifiers.

Table 5-7 Semantics of Operators and Modifiers

Operator	Modifier	Meaning	Example
LD	N	Load the (negated) operand into the accumulator	LD iVar
ST	N	Store the (negated) value in the accumulator into the operand variable	ST iErg
S	-	Set the operand (Boolean) to TRUE when the value in the accumulator is TRUE	S bVar1
R	-	Set the operand (Boolean) to TRUE when the value in the accumulator is FALSE	R bVar1
AND	N,(	Bitwise AND operation of the value in the accumulator and the (negated) operand	AND bVar2
OR	N,(	Bitwise OR operation of the value in the accumulator and the (negated) operand	OR xVar
XOR	N,(	Bitwise XOR operation of the value in the accumulator and the (negated) operand	XOR N,(bVar1,bVar2)
NOT	-	Bitwise negation of the value in the accumulator	-
ADD	(	Add the value in the accumulator to the operand and copy the result to the accumulator	ADD (iVar1,iVar2)
SUB	(	Subtract the operand from the value in the accumulator and copy the result to the accumulator	SUB iVar2
MUL	(	Multiply the value in the accumulator by the operand and copy the result to the accumulator	MUL iVar2
DIV	(	Divide the value in the accumulator by the operand and copy the result to the accumulator	DIV 44
GT	(	Check if the value in the accumulator is greater than the operand and copy the result (Boolean) to the accumulator; >	GT 23
GE	(	Check if the value in the accumulator is greater than or equal to the operand and copy the result (Boolean) to the accumulator; >=	GE iVar2
EQ	(	Check if the value in the accumulator is equal to the operand and copy the result (Boolean) to the accumulator; =	EQ iVar2
NE	(	Check if the value in the accumulator is not equal to the operand and copy the result (Boolean) to the accumulator; <>	NE iVar1
LE	(	Check if the value in the accumulator is less than or equal to the operand and copy the result (Boolean) to the accumulator; <=	LE 5
LT	(	Check if the value in the accumulator is less than the operand, copy the result (Boolean) to the accumulator, and jump unconditionally (conditionally) to the label; <	LT cVar1
JMP	CN	Unconditional (conditional) jump to the label	JMPN next
CAL	CN	(Conditionally) call a program or function block (when the value in the accumulator is positive)	CAL prog1
RET		Return from the current POU and jump to the called POU□□	RET
RET	C	Conditional: Return from the current POU and jump to the called POU only if the value in the accumulator is TRUE	RETC

Operator	Modifier	Meaning	Example
RET	CN	Conditional: Return from the current POU and jump to the called POU only if the value in the accumulator is FALSE	RETCN
)	-	Evaluate the delayed operand	-

Note:

- The accumulator always stores the current value, which is generated when there is a subsequent operation. The operand of CAL should be the instance name of a called function block.
- The result of the NOT operation is the bit negation of the current result. The modifier N indicates a negation operation. The RET operator does not require an operand.
- The modifier C indicates that the instruction is executed only if the result of the current operation is Boolean TRUE (or when the Boolean value of the operator is FALSE in combination with the "N" modifier).
- An operator can have more than one modifier at the same time, or have only one or none. For example, the JMP operator can have three formats: JMP, JMPC, and JMPN.

The left parenthesis "(" indicates that the operation of the operator is deferred until a right parenthesis ")" is encountered. Therefore, this operator can be used to implement program block operations and master control operations in traditional PLCs.

5.4.2.1 Operand

Operands can represent variables or symbolic variables directly. For example:

LDA: It indicates setting the current value equal to the value corresponding to the symbolic variable A.

AND%IX1.3: It indicates that the current result is ANDed with the third bit of the input unit 1, and the result is used as the current value.

JMPABC: It indicates that when the current calculated value is the Boolean value 1, execution starts from the position labeled ABC.

RET: It is an operator without operands. When this instruction is executed, the program will return to the instruction after the original breakpoint. Breakpoints are caused by function calls, function block calls, or interrupt subprograms.

5.4.2.2 Instruction

The instruction list programming language defined in the IEC61131-3 standard summarizes the traditional instruction list programming language by taking its strengths and overcoming its weaknesses, uses functions and function blocks, and employs the overload properties of data types, etc., making the programming language simpler and more flexible and the instructions easier. Its main advantages are as follows:

Function and function block calls:

Standard library calls: Timer and counter function block instructions can be directly called in the instruction list programming language through the standard library, making complex function implementation easier.

Simplified programming: By calling predefined functions and function blocks, you can reduce programming workload and improve code readability and maintainability.

Overload properties of data types:

Simplified operation: The overload properties of data types allow the same operation to be performed on variables of different data types, which simplifies the operation process and makes the code more concise and intuitive.

Program block combination:

Utilization of parentheses: You can use parentheses to easily combine program blocks together and realize the functions of instructions such as master control, making the implementation of complex logic more intuitive.

Edge detection:

Differentiation function: Signal are differentiated by using edge detection, which simplifies the instruction set and makes the edge detection of input signals easier.

Data transmission instructions:

Assignment function MOVE: Data transmission instructions can be directly implemented using the assignment function MOVE, making data transmission operations more direct and clear.

5.4.2.3 Operator

Before introducing the operator, we need to introduce a concept, namely the accumulator, which is particularly important in the instruction list programming language.

The instruction list programming language provides an accumulator to store the current result. Unlike the accumulator used in a traditional PLC, the number of storage bits of this standard accumulator is variable, that is, the standard instruction list programming language provides a virtual accumulator with a variable number of storage bits, and the number of storage bits depends on the operands and data types being processed. Similarly, the data type of the virtual accumulator may also be changed to adapt to the data type of the operand of the latest operation result.

During the execution of instructions, the data storage method is as follows:

Operation result: = current operation result operation operand

Therefore, under the operation defined by the operator, the current operation result and the operand undergo the operation defined by the operator. The operation result is used as a new operation to store the result back into the accumulator of the current operation result.

5.4.2.4 Modifier

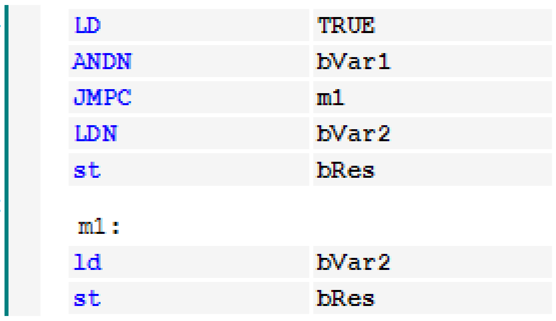
There are three modifiers, namely C, N, and N,(, as shown in Table 5-8. The modifier itself cannot be constructed independently and needs to be combined with the preceding operator to form a complete statement.

Table 5-8 Modifier Instructions

Modifier	Use	Function
C	Use in combination with JMP, CAL, and RET	This instruction is executed only when the result of the preceding expression is TRUE
N	Use in combination with JMPC, CALC, and RETC	This instruction is executed only when the result of the preceding expression is FALSE
N,(	Miscellaneous	Negate the operand (rather than the value in the accumulator)

The modifier C indicates that the instruction is executed only if the result of the current operation is TRUE (or when the Boolean value of the operator is FALSE in combination with the “N” modifier). The logic of the modifier N is exactly opposite to that of C.

[Example 5.28] Modifier example.



```
LD      TRUE
ANDN    bVar1
JMPC    m1
LDN     bVar2
st      bRes

m1:
ld      bVar2
st      bRes
```

First, TRUE is loaded into the accumulator, and then the value of the variable bVar1 is negated and ANDed with the value in the accumulator. At this time, the ANDN instruction is used. If AND is used, it means that the AND operation is performed directly. If the result is TRUE, the program jumps to m1. Otherwise, the variable bVar2 is negated and loaded into the accumulator and output. This instruction uses LDN and the modifier N, which also means negation.

5.4.3 Operation Instructions

The instruction list programming language includes 9 categories of instructions, which are described below.

5.4.3.1 Data Access Instructions

Data access instructions indicate operations that read data from data storage units. Standard instructions use LD and LDN instructions to represent access and access

negation instructions. The programming language format is as follows:

LD operand // Store the content in the data storage unit specified by the operand as the current result.

LDN operand // Negate the content in the data storage unit specified by the operand and then store it as the current result.

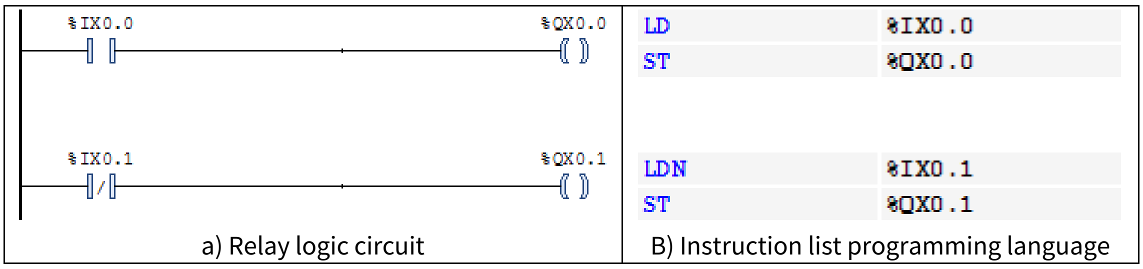
LD is short for Load, while LDN is short for LoadNot.

The operation object of a data access instruction, that is, the operation object of LD or LDN, is an operand. It is a read operation on the content in the data storage unit corresponding to the operand. The read data is stored in the operation result accumulator, which is also called the current value.

The LD instruction is used to read the data of a normally open contact, while the LDN instruction is used to read the data of a normally closed contact.

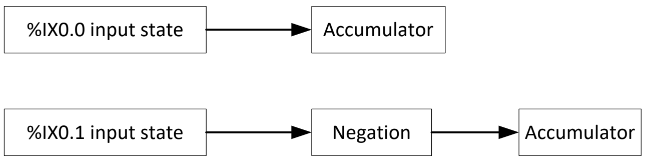
Similar to a relay logic circuit, for normally open contacts, that is, the movable contacts, the LD access instruction is used. For example, the LD%IX0.0 instruction executes the operation of accessing the contact state of the operand address %IX0.0. From the register point of view, the operation process is to transmit the input state of the address %IX0.0 to the operation result accumulator. Figure 5-38 a) and b) show instruction examples of a relay logic circuit and the instruction list programming language.

Figure 5-38 Examples and Operation Processes of LD and LDN Instructions



```
LD      %IX0.0
ST      %QX0.0

LDN     %IX0.1
ST      %QX0.1
```

Figure 5-39 Operation Processes of LD and LDN Instructions



The figure shows the execution process of data access. For normally closed contacts, that is, the movable contacts, the LDN logical negation instruction is used. For example, the LDN%IX0.1 instruction executes the operation of accessing the contact state of the operand address %IX0.1. From the register point of view, the operation process is to negate the state of the input state register of the address %IX0.1 and then transmit the negated result to the operation result accumulator.

Table 5-9 Examples of LD and LDN Instructions

Instruction	Description	Data Type of the Accumulator
LD FALSE	The current value is equal to FALSE	Boolean
LD TRUE	The current value is equal to TRUE	Boolean
LD 3.14	The current value is equal to 3.14	Real number
LD 100	The current value is equal to 100	Integer
LD T#0.5s	The current value is equal to the time constant 0.5s	Time data
LD START	The current value is equal to the state of the variable START	Depend on the type of the variable START

5.4.3.2 Output Instructions

Output instructions are used to transmit the content in the operation result accumulator to the output state register. Standard instructions use ST and STN

instructions to represent access and access negation instructions. The programming language format is as follows:

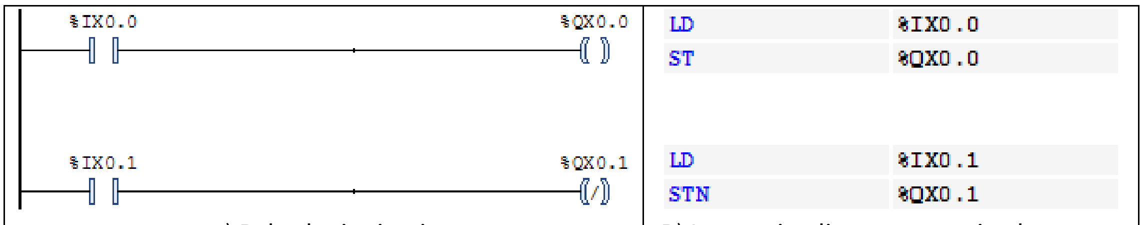
ST operand // Store the current result in the data storage unit specified by the operand.

STN operand // After negating the current result, store it in the data storage unit specified by the operand.

It should be noted that after executing the ST or STN instruction, the current operation result is still retained in the data storage unit of the operation result accumulator. ST means Store and STN is short for StoreNot.

Similar to a relay logic circuit, the ST instruction is used for coils. For example, the ST%QX0.0 instruction executes the operation of output to the %QX0.0 coil: from the register point of view, the operation process is to transmit the state of the operation result accumulator to the output address of %QX0.0. Figure 5-40 a) and b) show instruction examples of a relay logic circuit and the instruction list programming language.

Figure 5-40 Examples and Operation Processes of ST and STN Instructions

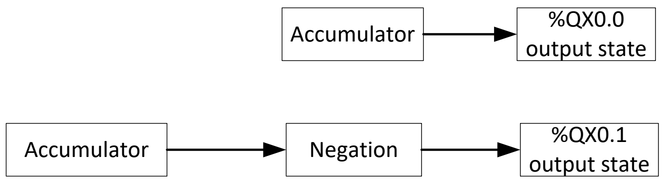


```
LD      %IX0.0
ST      %QX0.0

LD      %IX0.1
STN     %QX0.1
```

a) Relay logic circuit B) Instruction list programming language

The figure shows the execution process of data access. Use the STN%QX0.1 instruction to execute the operation of output to the %QX0.1 de-excitation coil; the operation process is to negate the state of the operation result accumulator and then transmit the negated result to the address of the output %QX0.1.

Figure 5-41 Operation Processes of ST and STN instructions



5.4.3.3 Set and Reset Instructions

The standard instruction set uses S and R instructions to represent set and reset instructions. The programming language format is as follows:

S operand // When the current result is FALSE, set the content in the data storage unit corresponding to the operand to TRUE and retain it.

R operand // When the current result is TRUE, set the content in the data storage unit corresponding to the operand to FALSE and retain it.

This type of instruction has memory properties. After the S operand is executed, the content in the data storage unit corresponding to the operand is set to TRUE and is memorized and retained until the R operand instruction is executed. The execution of the R operand instruction sets the content in the data storage unit corresponding to the operand to FALSE. Likewise, the content in the data storage unit is retained until the S operand instruction is executed and its content is set to TRUE.

The S and R instructions can be implemented by calling the SR and RS function blocks. Compared with function blocks, the difference lies in that the execution sequence of S and R instructions is determined according to their positions in the program. Therefore, the precedence determination is different from RS and SR. In addition, function blocks must first set the S and R terminals before executing the call instruction.

S stands for Set, while R stands for Reset. Examples of S and R instructions are shown in Table 5-10.

Table 5-10 Examples of S and R Instructions

Instruction			Description
SET:	LD	TRUE	The current value is equal to TRUE
	S	START	The current value is equal to TRUE, and the value of the variable START is set to TRUE and retained
	LD	FALSE	The current value is equal to FALSE
	S	STOP	The current value is equal to FALSE, and the value of the variable STOP is set to FALSE and retained
RESET:	LD	TRUE	The current value is equal to TRUE
	R	STOP	The current value is equal to TRUE, and the value of the variable STOP is set to FALSE and retained.

The S instruction is a conditional STC output instruction, while the R instruction is a conditional STCN output instruction. Therefore, when the current result memory is TRUE, the S operand instruction executes the operation of setting the output operand to a set position. Similarly, the R operand instruction executes the operation of setting the output operand to a reset position, that is, the operation of set negation.

5.4.3.4 Logical Operation Instructions

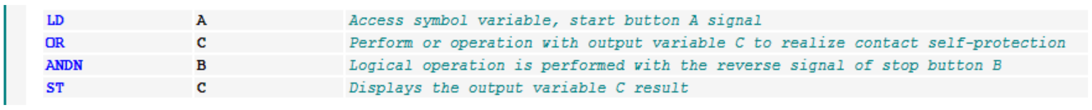
Standard logical operations include: AND(N), OR(N), XOR(N) and NOT. The programming language format is as follows:

Logical operator operand or logical operator N operand

Logical operator: The operand is used to perform a specified logical operation on the content in the current result memory and the content in the data storage unit corresponding to the operand, and the operation result is stored in the current result memory as the new current result.

Logical operator N: The operand is used to perform a specified logical operation on the content in the current result memory and the negated result of the content in the data storage unit corresponding to the operand, and the operation result is stored in the current result memory as the current result.

[Example 5.29] Motor control program example.



```
LD      A       Access symbol variable, start button A signal
OR      C       Perform or operation with output variable C to realize contact self-protection
ANDN    B       Logical operation is performed with the reverse signal of stop button B
ST      C       Displays the output variable C result
```

This example is a typical motor control program. The input variables A and B as well as the output variable C are all symbolic variables, and their actual addresses must be assigned in the declaration part.

5.4.3.5 Arithmetic Operation Instructions

This type of instruction includes ADD, SUB, MUL, DIV, and MOD. The programming language format is as follows:

ADD operand // The content in the data storage unit corresponding to the operand is added to the current result, and the operation result is stored in the current result memory.

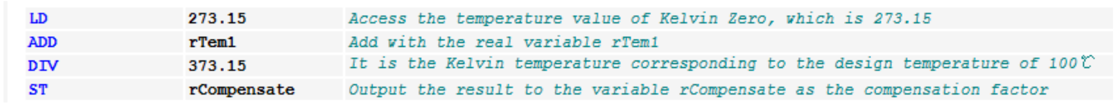
SUB operand // The content in the data storage unit corresponding to the operand is subtracted from the current result, and the operation result is stored in the current result memory.

MUL operand // The current result is multiplied by the content in the data storage unit corresponding to the operand, and the operation result is stored in the current result memory.

DIV operand // The current result is divided by the content in the data storage unit corresponding to the operand, and the operation result (quotient) is stored in the current result memory.

MOD operand // The current result is modulo the content in the data storage unit corresponding to the operand, and the operation result is stored in the current result memory.

[Example 5.30] Operation example of the temperature compensation coefficient.



```
LD      273.15          Access the temperature value of Kelvin Zero, which is 273.15
ADD     rTem1           Add with the real variable rTem1
DIV     373.15          It is the Kelvin temperature corresponding to the design temperature of 100℃
ST      rCompensate     Output the result to the variable rCompensate as the compensation factor
```

[Example 5.30] is used to perform temperature compensation on the gas flow, where rTem1 is the actual temperature in °C. The program reads 273.15 in the first line; the second line adds the actual temperature value rTem1 to 273.15 and uses it as the current value. The third line divides this current value by

the designed temperature value, and the result is stored in the current value memory; the fourth line stores the operation result as the temperature compensation value in rCompensate. It can be seen that the ADD and DIV operations in the program are both operations of real number data types.

5.4.3.6 Comparison Operation Instructions

Comparison instructions include: GT (>), GE (≥), EQ (=), NE (≠), LE (<), and LT (≤). The programming language format is as follows:

GT operand // The current operand > the content in the data storage unit corresponding to the operand, and the operation result TRUE is sent to the current result register.

GE operand // The current operand ⩾ the content in the data storage unit corresponding to the operand, and the operation result TRUE is sent to the current result register.

EQ operand // The current operand = the content in the data storage unit corresponding to the operand, and the operation result TRUE is sent to the current result register.

NE operand // The current operand ≠ the content in the data storage unit corresponding to the operand, and the operation result TRUE is sent to the current result register.

LE operand // The current operand < the content in the data storage unit corresponding to the operand, and the operation result TRUE is sent to the current result register.

LT operand // The current operand ⩽ the content in the data storage unit corresponding to the operand, and the operation result TRUE is sent to the current result register.

This type of instruction is used to compare the current result with the content in the data storage unit corresponding to the operand. When the comparison condition specified by the operator is met, the current result is set to TRUE; otherwise, it is set to FALSE. The comparison instruction changes the data type of the current result memory to a Boolean data type.

Note:

- This instruction directly stores the comparison result in the data storage unit, and the user can execute subsequent programs according to the state of the storage unit.
- Comparison operation instructions are suitable for comparing variables of different data types and are not limited to single-bit comparisons. Therefore, their application scope can be expanded.

[Example 5.31] Example of a comparison operation instruction.

```
LD      rRealVar   Access real variables rRealVar
GT      50.0       The variable rRealVar is greater than 50
ST      bRed       If it is greater than 50, the rRealVar value exceeds the limit and is bRed to be TRUE
STN     bGreen     If it is not greater than or equal to 50, the green bGreen is set to TRUE
```

In [Example 5.31], the variable rRealVar is a process measurement value. When its value is greater than 50, it means that the measurement value is out of limit, and the red alarm bRed is TRUE. Otherwise, bGreen is TRUE.

5.4.3.7 Jump and Return Instructions

The jump instruction is JMP and the return instruction is RET. The programming language format for each of them is as follows:

```
JMP Label    // Jump to the label position and then continue execution
RET          // Return to the breakpoint at the time of jump and then continue execution
```

The operand of the jump instruction is a label rather than the address of the data storage unit corresponding to the operand.

The return instruction has no operands and is used to call a function, function block, or program to return.

JMP is short for Jump. When this instruction is executed, if the current result is TRUE, the jump condition is met, and the program is interrupted at this point and jumps to the program line where the label is located to continue execution. It is used in conjunction with the RET instruction to implement the execution of subprograms. It can be accompanied by a modifier C or N, indicating execution or negation based on the current result memory content.

RET is short for Return. After this instruction is executed, the program returns and starts execution from the first instruction after the power failure. It can be accompanied by a modifier C or N, indicating execution or negation based on the current result memory content.

Note:

- The jump instruction jumps from the master program to a subprogram. A subprogram cannot jump to the master program using a jump instruction, but can only return using a return instruction. A subprogram starts with a label and ends with a RET instruction.
- The label in the program is unique.

[Example 5.32] Example of a jump instruction

```
LD      AUTO       Access real variables rRealVar
JMPC    AUTOPRO    If AUTO is TRUE, the program jumps to the AUTOPRO subroutine
JMP     MANPRO     If AUTO is FALSE, the program jumps to the MANPRO subroutine
```

[Example 5.32] is used for switching control between automatic and manual programs. When the AUTO switch is switched to the automatic position, AUTO is TRUE and the program will execute the jump instruction JMPC. Therefore, the program jumps to the AUTOPRO subprogram and executes the associated programs under automatic conditions. When the jump condition is not met, the JMP instruction is executed, so the program jumps to the MANPRO subprogram and executes the associated programs under manual conditions.

It should be noted here that AUTOPRO and MANPRO are subprogram labels rather than program names.

5.4.3.8 Call Instructions

The standard call instruction of IEC61131-3 is the CAL instruction. The programming language format is as follows.

CAL operand // Call the function, function block, or program represented by the operand

By executing this instruction, functions, function blocks, and programs can be called to simplify the program structure and make the program description clear. The general call format is as follows.

CAL is short for Call, which means calling. The operand of the CAL instruction is a function name or a function block instance name. Parameters in the instance name are separated by commas.

5.4.3.9 Parentheses Instructions

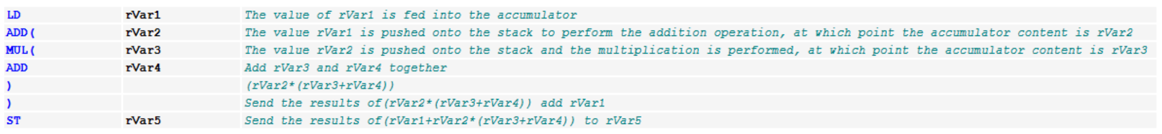
The IEC61131-3 standard uses parentheses to modify instructions, that is, to perform precedent operations.

The left parenthesis "(" is used to push the current accumulator content into the stack and store the operation instruction of the operator. At this time, the other content of the stack is moved down one layer. The right parenthesis ")" is used to pop the content in the top layer of the stack and perform the corresponding operation on the current accumulator content. The operation result is placed in the current accumulator. At this time, the content of the stack is moved up one layer. Therefore, the left parenthesis is called an operation delay, and its instantaneous result does not affect the current accumulator.

Table 5-11 Expressive Properties of Parentheses

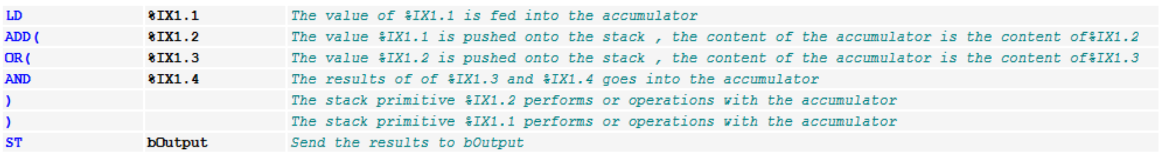
No.	Description/Example	
1	Parenthesized expression starting with an explicit operator	AND(LD %IX0.1 OR %IX0.2)
2	Parenthesized expression (short form)	AND(LD %IX0.1 OR %IX0.2)

[Example 5.33] Modify an arithmetic operation with parentheses.



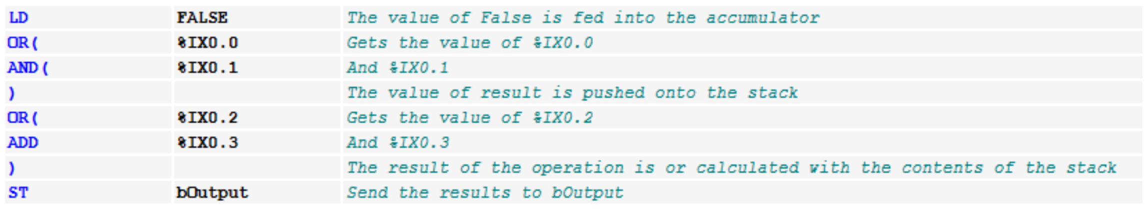
```
LD      rVar1   The value of rVar1 is fed into the accumulator
ADD(    rVar2   The value rVar1 is pushed onto the stack to perform the addition operation, at which point the accumulator content is rVar2
MUL(    rVar3   The value rVar2 is pushed onto the stack and the multiplication is performed, at which point the accumulator content is rVar3
ADD     rVar4   Add rVar3 and rVar4 together
)               (rVar2*(rVar3+rVar4))
)               Send the results of(rVar2*(rVar3+rVar4)) add rVar1
ST      rVar5   Send the results of(rVar1+rVar2*(rVar3+rVar4)) to rVar5
```

In [Example 5.33], the final implemented algorithm is rVar1+rVar2*(rVar3+rVar4). During the entire operation, the data type must remain consistent. In addition, the data type is transmitted. The operation starts from the innermost parentheses and moves outwards layer by layer until the outermost parentheses are reached.



```
LD      %IX1.1    The value of %IX1.1 is fed into the accumulator
ADD(    %IX1.2    The value %IX1.1 is pushed onto the stack , the content of the accumulator is the content of%IX1.2
OR(     %IX1.3    The value %IX1.2 is pushed onto the stack , the content of the accumulator is the content of%IX1.3
AND     %IX1.4    The results of of %IX1.3 and %IX1.4 goes into the accumulator
)                 The stack primitive %IX1.2 performs or operations with the accumulator
)                 The stack primitive %IX1.1 performs or operations with the accumulator
ST      bOutput   Send the results to bOutput
```

[Example 5.34] Modify a logical operations with parentheses.



```
LD      FALSE     The value of False is fed into the accumulator
OR(     %IX0.0    Gets the value of %IX0.0
AND(    %IX0.1    And %IX0.1
)                 The value of result is pushed onto the stack
OR(     %IX0.2    Gets the value of %IX0.2
ADD     %IX0.3    And %IX0.3
)                 The result of the operation is or calculated with the contents of the stack
ST      bOutput   Send the results to bOutput
```

[Example 5.35] Application of parentheses in parallel connection of program blocks.

In [Example 5.35], the two instructions starting with OR are two program blocks, which are programs that connect two contacts in series. Finally, after the OR operation, the operation result is stored in the bOutput variable.

In mathematical operations, parentheses have a similar function to brackets, that is, the operations outside the brackets are deferred.

```
LD          rVar1       The value of rVar1 is fed into the accumulator
ADD         rVar2       Add rVar2,send the results into the accumulator
MUL(        rVar3       The value rVar3 is pushed onto the stack
SUB         rVar4       rVar3-rVar4
)                       Send the results to the accumulator
```

[Example 5.36] Delay function of parentheses.

In [Example 5.36], the operation result is (rVar1+rVar2)*(rVar3-rVar4).

The relationship between the accumulator and the stack is illustrated by the following example.

```
LD          rVar1       The value of rVar1 is fed into the accumulator
ADD(        rVar2       The value rVar2 is pushed onto the stack
MUL(        rVar3       The value rVar3 is pushed onto the stack
SUB         rVar4       rVar3-rVar4
)                       rVar2*(rVar3-rVar4)
)                       rVar1+(rVar2*(rVar3-rVar4))
```

[Example 5.37] Relationship between the accumulator and the stack.

In [Example 5.37], the data in the stack and the current accumulator data are shown in Table 5-12.

Table 5-12 Changes in Stack Data and Current Accumulator Data

Instruction	1	2	3	4	5	6
Current accumulator	rVar1	rVar1	rVar1	rVar1	rVar1	rVar1+ rVar2*(rVar3- rVar4)
Stack 1	-	rVar2	rVar2	rVar2	rVar2*(rVar3-rVar4)	-
Stack 2	-	-	rVar3	rVar3 - rVar4	-	-

Finally, the operation result of [Example 5.37] is rVar1+rVar2*(rVar3-rVar4).

Therefore, it is easier to implement more complex operation relationships using the structured text or ladder diagram language. Jumping from instructions within parentheses can sometimes produce unpredictable results, so you need to be careful when doing so.

5.4.4 Function and Function Block

5.4.4.1 Function Call

In the instruction list programming language, function calls are relatively simple.

Function Call Method

Enter the function name in the operator field, and use the first input parameter as the operand of LD. If there are more parameters, enter the next one in the same line as the function name, and add subsequent parameters separated by commas to this line or the following line. The function return value will be stored in the accumulator. It should be noted that according to the IEC standard, there can only be one return value. The programming language format and example of a function call are shown in Table 5-13.

Table 5-13 Programming Format and Example of a Function Call

Type	Programming Format		Example	
Single parameter	LD Function name ST	Parameter Return value	LD COS ST	0.5 // Read the radian 0.5 // Call the COS function Var1 // Store the operation result 0.87758 in the variable Var1
Dual-parameter	LD Function name ST	Parameter 1 Parameter 2 Return value	LD ADD ST	Var1 // Read the value of the variable Var1 Var2 // Add to the value of the variable Var2 Var3 // Store the return value of the operation result in Var3回
Multi-parameter	LD Function name ST	Parameter 1 Parameter 2,…,parameter n Return value	LD SEL ST	Var1 // Read the value of the variable Var1 IN0,IN1// Select IN0 or IN1 as the return value according to the value of Var1 Var2 // Store the return value of the operation result in Var2回

The programming language format of a function call with a non-formal parameter list is as follows.

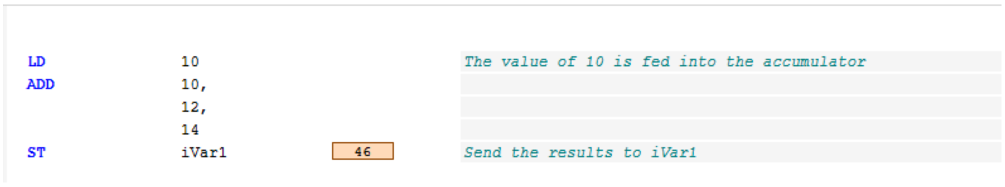
Function name non-formal parameter, non-formal parameter, ..., non-formal parameter,

The programming language format of a function call with a formal parameter list is as follows.

Function name (first formal parameter := actual parameter, ..., last formal parameter := actual parameter)

Examples of function calls

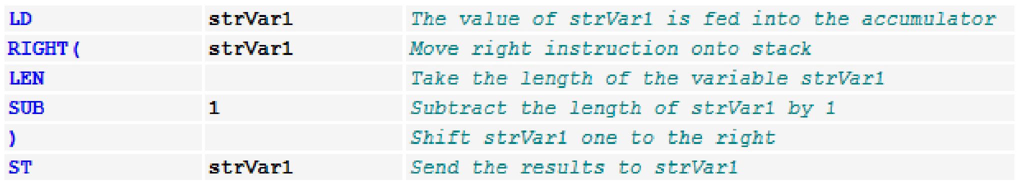
The following two examples of function calls will help you have a clearer understanding.



```
LD      10              The value of 10 is fed into the accumulator
ADD     10,
        12,
        14
ST      iVar1   46      Send the results to iVar1
```

[Example 5.38] Example of a non-formal parameter function call.

In [Example 5.38], the ADD function is used to directly implement the addition operation of multiple values. Therefore, compared with the addition operation of a traditional PLC, the program is simplified. It should be noted that some traditional PLC products only allow one operand, for example, ADD10. CoDeSys, however, can be directly superimposed.



```
LD          strVar1     The value of strVar1 is fed into the accumulator
RIGHT(      strVar1     Move right instruction onto stack
LEN                     Take the length of the variable strVar1
SUB         1           Subtract the length of strVar1 by 1
)                       Shift strVar1 one to the right
ST          strVar1     Send the results to strVar1
```

[Example 5.39] Example of a formal parameter function call.

Call the RIGHT function. The first parameter of this function is LEN, which is the data length of the variable strVar1. The second parameter is sub1, which is the data length in the current accumulator minus 1 and represents the number of bits to shift right, that is, 1 bit at a time. If the above program is converted into a structured text, it becomes as follows:

strVar1 := RIGHT(strVar1, (LEN(strVar1) - 1));

5.4.4.2 Function Block Call

Function Block and Program Call Method

In non-formal parameter programming languages, there are two methods to call a function block.

1. The programming language format of a function block call with a formal parameter list is as follows.

   CAL function block instance name (formal parameter list)

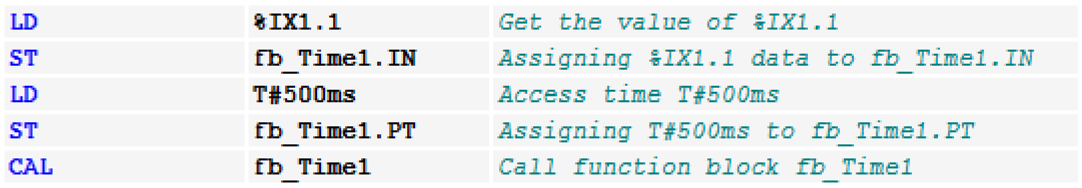
2. The programming language format of a function block call with parameter reading/storage is as follows.

   CAL function block instance name

Function block and program call example

The following uses the TON function block as an example to illustrate how to call a function block.

[Example 5.40] The program for calling a function block with parameter reading/storage is as follows.



```
LD     %IX1.1          Get the value of %IX1.1
ST     fb_Time1.IN     Assigning %IX1.1 data to fb_Time1.IN
LD     T#500ms         Access time T#500ms
ST     fb_Time1.PT     Assigning T#500ms to fb_Time1.PT
CAL    fb_Time1        Call function block fb_Time1
```

5.4.5 Application Examples

[Example 5.41] Weighing display example.

In actual industrial production, many devices are equipped with weighing and screening instruments. When the actual weight of the product does not meet the set value, the product will be considered defective and a rejection signal will be triggered to reject the product. Such instruments have weighing-related programs.

Control requirements

The weighing instrument stores the gross weight data of the material in the PLC register, uses the weighing function to subtract the tare weight from the gross weight, and finally outputs the net weight as a REAL type variable.

Assumptions: gross weight variable: rGrossWeight; tare weight variable: rTareWeigh; net weight variable: rActuallyWeight.

In order to control the execution of the weighing signal, it is necessary to set the manual trigger signal as the weighing instruction and use the Boolean variable bStart as the start instruction.

The data display types of gross weight rGrossWeight, tare weight rTareWeight, and net weight rActuallyWeight are all REAL data types.

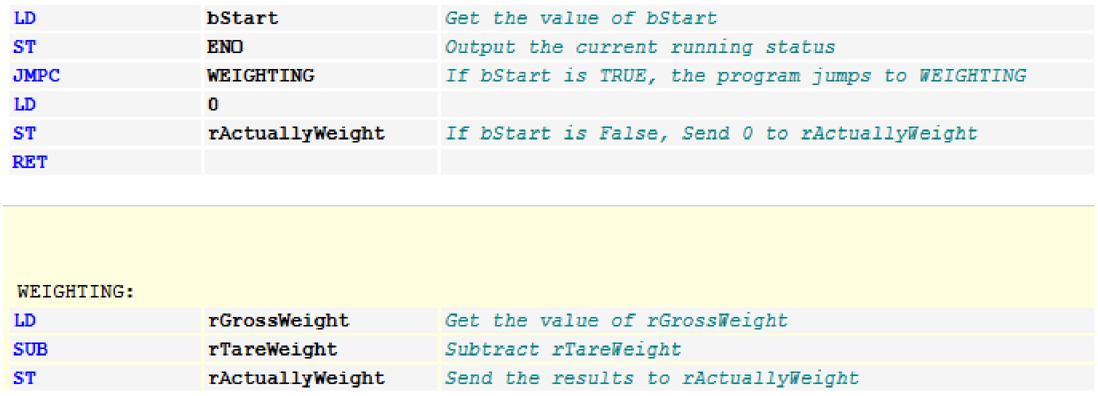
Programming

Write the function block FB_Weight, with the function block declaration area shown below.

```
FUNCTION_BLOCK FB_WEIGHT
VAR_INPUT
        bStart: BOOL;
        rGrossWeight:REAL;
        rTareWeight:REAL;
END_VAR VAR_OUTPUT
        ENO:BOOL;
        rActuallyWeight:REAL;
END_VAR
VAR
```

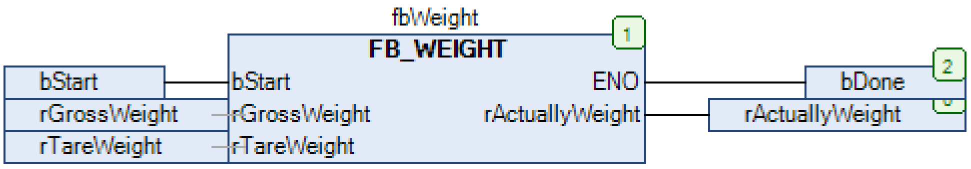
END_VAR

The logical program of the function block is as follows:



```
LD          bStart              Get the value of bStart
ST          ENO                 Output the current running status
JMPC        WEIGHTING           If bStart is TRUE, the program jumps to WEIGHTING
LD          0
ST          rActuallyWeight     If bStart is False, Send 0 to rActuallyWeight
RET

WEIGHTING:
LD          rGrossWeight        Get the value of rGrossWeight
SUB         rTareWeight         Subtract rTareWeight
ST          rActuallyWeight     Send the results to rActuallyWeight
```

After the program of the function block is completed, the actual test is performed by adding a new program, calling the function block FB_Weight in the program, and filling in the corresponding input and output parameters. The final program is shown in the figure below.



Weighing program example

For example, the gross weight is 5 (g) and the tare weight is set to 1 (g). Only when bStart is triggered and becomes TRUE, the final net weight will be 4 (g); otherwise, it will always be 0.

[Example 5.42] Example of a loop operation.

Control requirements

Create a program that calculates the cumulative sum and factorial of numbers from 1 to 10. You can use the JMPC jump instruction in the program.

Programming

The variable declaration is as follows:

```
PROGRAM PLC_PRG
VAR
        diSum,diProduct:DINT;
        i:BYTE;
END_VAR
```

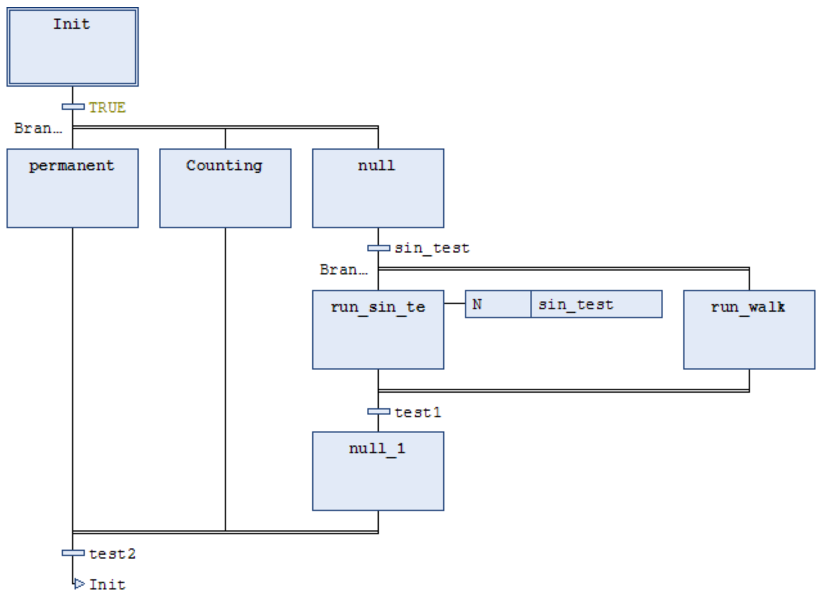
```
LD          1               Get initial value
ST          i               i:=1
ST          diProduct       diProduct:=1
LD          0               Get initial value
ST          diSum           diSum:=0;


LOOP:
LD          diSum           Get the value of diSum
ADD         i               i+diSum
ST          diSum           diSum:=i+diSum
LD          diProduct       Get the value of diProduct
MUL         i               diProduct*i
ST          diProduct       diProduct:=diProduct*i
LD          i               Get the value of i
ADD         1               i+1
ST          1               i:=i+1
LE          10              If the value is less than 10, the current value is set to 1
JMPC        LOOP            If the value is 1, jump to LOOP
RET                         return
```

The above program can simply perform cumulative sum and factorial operations. The operation result is 55 in diSum and 3628800 in the variable diProduct. It should be noted that when the operation result is larger than the allowable range of the data type set for the variable, the result will be set to 0. For example, if the cumulative sum and factorial of 1 to 50 are calculated and the factorial result exceeds the allowable range of a long long integer, the result is set to 0. To solve this problem, you can change the data type of the variable to a REAL type or DOUBLE PRECISION REAL type.

5.5 Sequential Function Chart (SFC)

The SFC programming language is designed to meet the needs of sequential logic control. During programming, the process of sequential action flow is divided into steps and transition conditions. The function flow sequence of the control system is allocated according to the transition conditions, and the action is performed step by step in sequence, as shown in the figure. Each step indicates a control function task and is represented by a box. The box contains the ladder logic used to complete the corresponding control function task. This programming language makes the program structure clear and easy to read and maintain, which can greatly reduce the programming workload and shorten the programming and debugging time. It is used in situations where the system is large in scale and the program relationships are complex. It features taking the function as the main line, performing allocation in the sequence of function flow, clear organization, and ease of understanding the user program.

Figure 5-42 Sequential Function Chart Programming Language

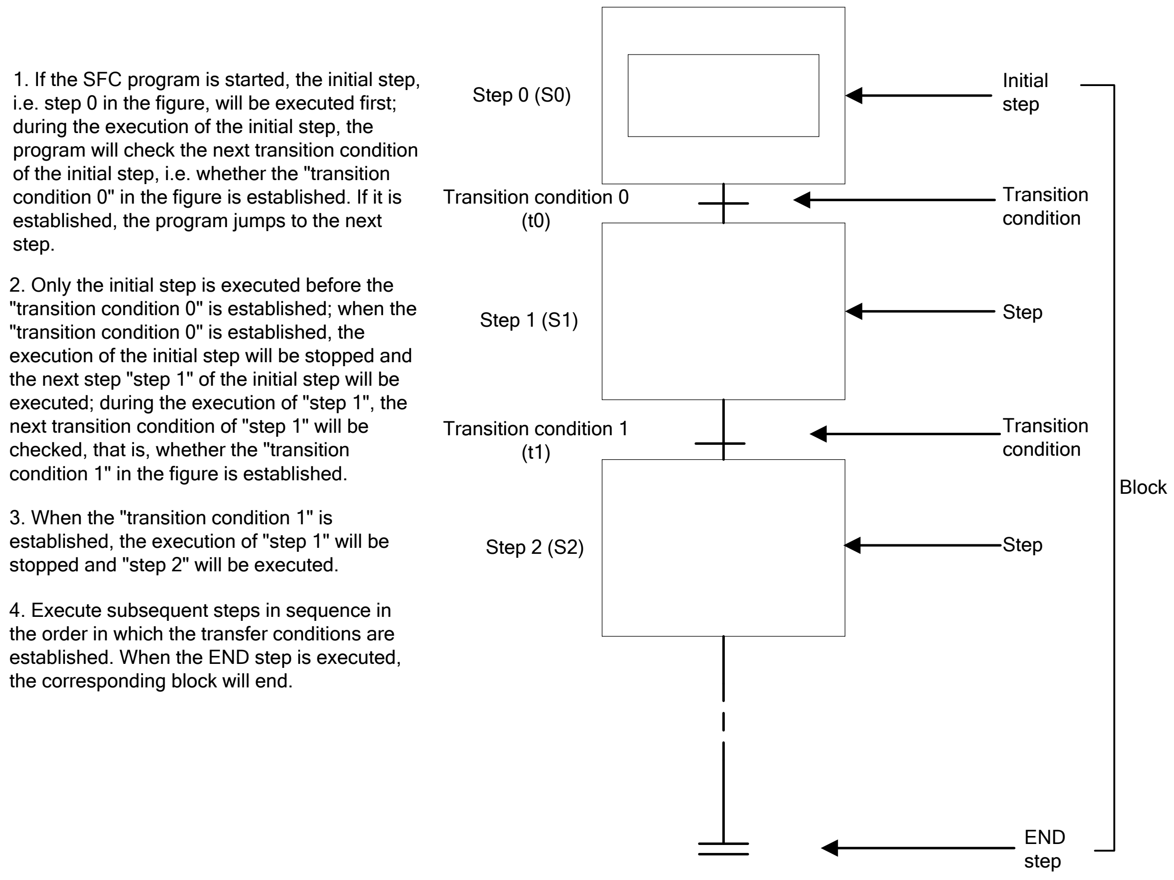


5.5.1 Introduction to the Sequential Function Chart Programming Language

Basic structure

A SFC program starts from the initial step. When the transition condition is met, the next step of the transition condition is executed in sequence, and the series of actions is ended by the END step. The whole process is shown in Figure 5-43.

Figure 5-43 Basic Process of an SFC Program



1. If the SFC program is started, the initial step, i.e. step 0 in the figure, will be executed first; during the execution of the initial step, the program will check the next transition condition of the initial step, i.e. whether the "transition condition 0" in the figure is established. If it is established, the program jumps to the next step.

2. Only the initial step is executed before the "transition condition 0" is established; when the "transition condition 0" is established, the execution of the initial step will be stopped and the next step "step 1" of the initial step will be executed; during the execution of "step 1", the next transition condition of "step 1" will be checked, that is, whether the "transition condition 1" in the figure is established.

3. When the "transition condition 1" is established, the execution of "step 1" will be stopped and "step 2" will be executed.

4. Execute subsequent steps in sequence in the order in which the transfer conditions are established. When the END step is executed, the corresponding block will end.

Program features

1. SFC advantages

- Best choice for sequential control

The automatic operation sequence can be converted into a graphic description as it is, so it is easy to program and understand the program.

- Comprehensible structured program

Graphics can be used for hierarchical and modular programming, so it is easy to test run and maintain. As shown in the figure, the left side is an equipment operation flowchart. Through the SFC programming language, the flowchart on the left can be directly converted into a program, and the programmer only needs to add a corresponding logic to each action and add appropriate transition conditions when the process jumps.

Figure 5-44 Equipment Operation Flowchart

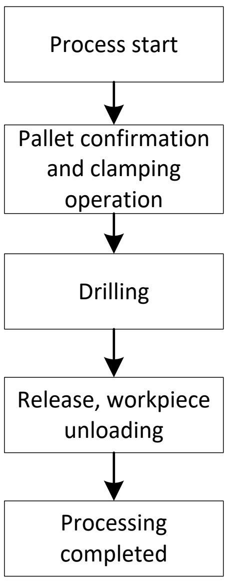
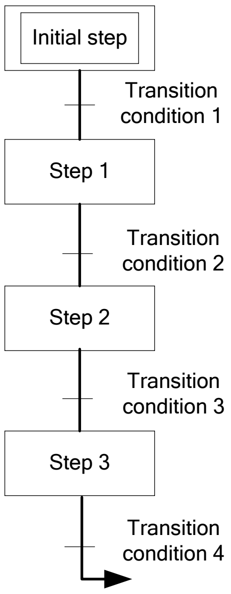

Process start
Pallet confirmation and clamping operation
Drilling
Release, workpiece unloading
Processing completed

Figure 5-45 SFC Diagram



- No interlocking required between processes (steps)

Since the CPU only operates on the action steps, the forward and backward action logics do not need to be interlocked. During SFC programming, you do not need to consider interlocking, because if the conditions of the previous and next steps are not met, the program will not execute other steps. Therefore, there is no need to consider too much about contact interlocking.

- Same coil shared in multiple processes (steps)

Since the CPU only operates on the action steps, even if the same coil exists in the steps that are not in motion, it will not be processed as a double coil. (If the coil is the same as that in the master sequential control program, it will be processed as a double coil.)

- Action state monitoring with graphics

When mechanical equipment stops due to a fault, the current step in which it is stopped can be displayed on the monitor so that the cause of the shutdown can be found quickly, which is convenient for troubleshooting. In addition, if there are annotations attached to each step, it will be clear at a glance why the action stops.

- Standardized design

The program is created graphically according to the control flow, so no matter who writes it, it will be almost the same with no individual differences, thus achieving standardization of design drawings.

- Coordinated programming by multiple persons

The control content can be divided into multiple parts, which are written by different people and then combined into one.

- Operation processing by step

Since the CPU only operates on the action steps, the scan time can be shortened through good programming methods.

- Easy system design and maintenance

Since the control of the entire system, the individuals, and the machine corresponds to the SFC program and steps on a one-to-one basis, even personnel with little experience in sequential control programs can design and maintain the system. In addition, other programmers also use this format to design programs that are more readable than other programming languages.

- In addition, by effectively using the functions of SFC, the cycle time of mechanical operation can be shortened.

2. SFC disadvantages

- Inapplicable control content

Programs such as emergency stop, continuous monitoring, and receiving data from a computer that use interrupt processing are not sequentially controlled and are therefore not suitable for writing SFC programs. (If you write a ladder diagram program in the master program to control such content, it will be easier to summarize and grasp it.)

- Prejudice due to unfamiliarity

Due to unfamiliarity, there may be prejudice that it cannot be used in extremely complex controls. (Structuring and modularization through SFC can organize the content to be controlled, so only ladder diagram programming is required).

5.5.2 SFC Structure

In the "Toolbox" of the SFC programming language, you can add SFC tools. An SFC consists of the 6 major parts listed in Table 5-14, among which steps and transition conditions are the basic elements of SFC. The various basic elements can be integrated for form several basic structures. Any complex or simple SFC structure is composed of these basic elements, as shown in Figure 5-46.

Figure 5-46 Basic Structure of an SFC

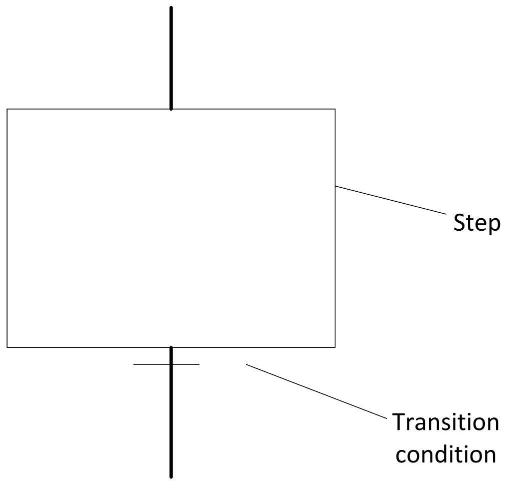


Table 5-14 SFC Toolbox

Type	Graphic Symbol	Description
Step		SFC consists of a series of steps that are interconnected via directed links.
Transition		An action is a collection of instructions implemented in other languages, such as a collection of statements implemented in IL or in ST, or a collection of networks implemented in LD, in FBD, or in SFC
Action	a	An action instruction can add an entry action and an exit action to a step.
Jump		The switching between steps is a transition. The step transition is performed only when the step transition condition is TRUE.
Macro		Add a macro
Branch		Add parallel branches

5.5.2.1 Step

Definition of a step

A step represents a major function in the entire industrial process. It can be a specific time or stage, or an action performed by several devices. The step belongs to the execution body of an SFC, and all the logical codes for implementing the execution are included in it. A transition condition determines the state of the step. When the transition condition of the previous step is met, this step is activated and the activated step will enter the execution state.

During activation, this step is scanned repeatedly until the transition condition of this step is met, the step activation is released, execution exits to the next step, and the next step is activated.

Each step in an SFC is represented by a box, which contains the "step name" and the up and down transition relationships represented by connecting lines. The step name can be edited directly at the current location, and must be unique within the POU where it is located, which requires special attention when SFC action programming is used.

Step configuration

There are two types of steps: initial step and normal step. The following will introduce these two different types of steps one by one.

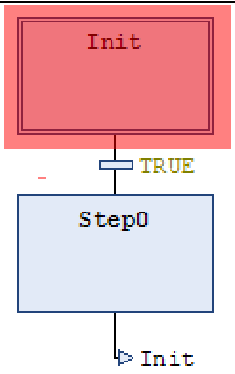
1. Initial step

The initial step is a step indicating the start of each block. You can select the corresponding "step" by right-clicking to select "Initial Step" or pressing the "⧈" button in the shortcut menu to set the initial step. As shown in Figure 5-47 a), the view of the initial step is slightly different from that of the normal step. The initial step is represented by a double rectangular box (surrounded by a double-sided line), which can be set by right-clicking, as shown in Figure 5-47 b).

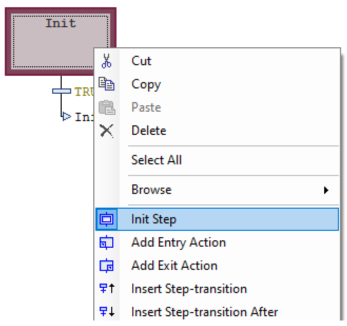
Table 5-15 Step Editing

Type	Graphic Symbol	Description
Initial Step	⧈	It is used to set the step currently selected in the SFC editor as the initial step

Figure 5-47 Initial Step Setting



a) Initial step view

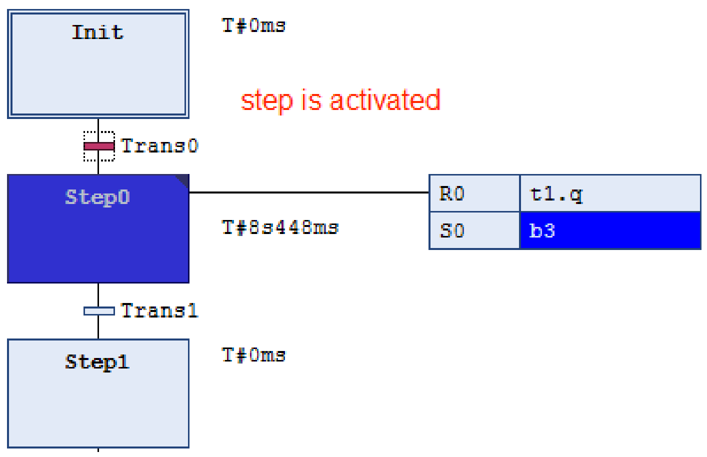


b) Initial step setting

2. Normal step

The "normal step" currently being executed is called an "active step". In online mode, the "active step" is filled in blue.

Each step consists of an action and a flag that indicates whether the step is active. If a single-step action is being executed, the step will be displayed as a blue frame, as shown in the figure below.



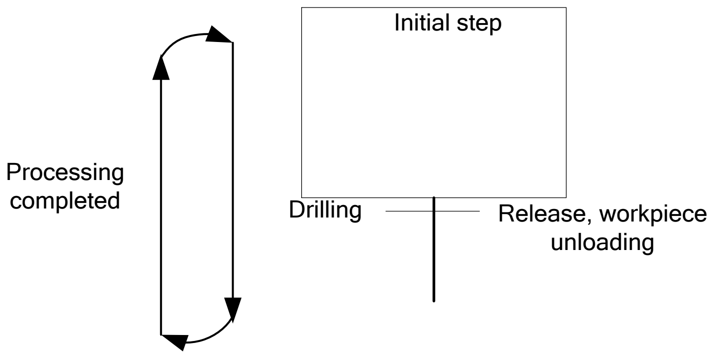
For a normal step, all actions of the active step in a control cycle will be executed. Therefore, when the transition condition is TRUE after the step is activated, the step after it is activated. The currently active step will be executed again in the next cycle.

5.5.2.2 Action

Definition of an action

As introduced at the beginning of this section, the most basic structure of the SFC execution process is an coordination of steps and transition conditions. Whenever a step is activated, it will be executed until the transition condition is met before moving to the next step. The next step is activated to start a new execution action, and it will not stop until its transition condition is met. The steps are executed in sequence, as shown in the figure below.

Figure 5-48 SFC Execution Steps



An action is a specific operation to be performed, such as opening a valve, starting or stopping a motor, and moving a workpiece or product. In each step, multiple actions can be executed, and the transition condition is also an execution judgment. Therefore, when an SFC operation process structure is established, a very important part is to determine the steps and configure their actions.

In addition, the following labels will be generated:

- Whenever a step is created, it is automatically assigned a structure label.
- Whenever an action is created, it is automatically assigned a structure label.
- Whenever a transition condition is created, it is automatically assigned a BOOL label. The data of these labels can be referenced in programming.

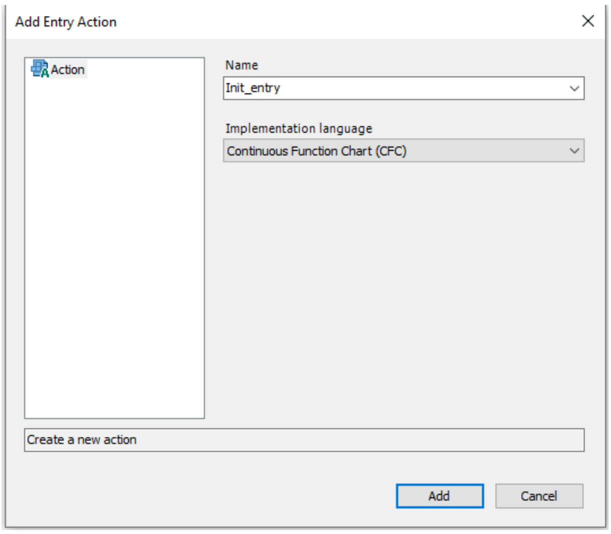
Each step can define multiple (or single) actions which include a detailed description of the execution of this step. The action can be written in LD, FBD, ST, SFC, or other languages. Users can edit entry and exit actions. The elements for editing actions are listed in Table 5-16

Table 5-16 Action

Type	Graphic Symbol	Description
Add Entry Action		Action performed before step activation
Add Exit Action		Action to be executed in the next cycle after the step is executed ("step exit")

Once you select "Add Action", the system will automatically pop up a prompt box, as shown in Figure 5-49. You can select the desired programming language to write action programs.

Figure 5-49 Programming Languages Supported by Step Actions



Name: Init_entry

Implementation language: Continuous Function Chart (CFC)

Create a new action

Qualifier

Qualifiers are used to configure how an action will be associated with an IEC step. They are inserted into the qualifier field of an action element. These qualifiers are processed by the SFC Action Control function block of the IecSfc.library and can be automatically included in a project through the SFC plug-in IecSfc.library. SFC qualifiers are listed in Table 5-17.

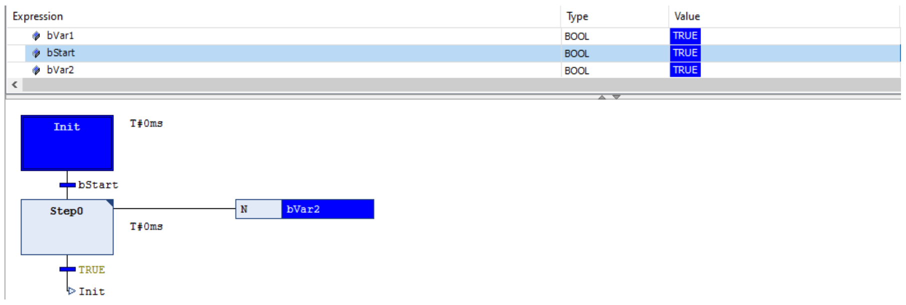
Table 5-17 Qualifiers

Qualifier	Name	Description
N	Non-stored	The action is active as long as the step is active
S0	Set (Stored)	The action is activated when the step is activated and remains active even when the step is deactivated until it is reset
R0	Overriding Reset	The action is deactivated
L	Time Limited	The action is activated when the step is activated and remains active until the step becomes inactive or the set time expires
D	Time Delayed	The delay timer starts after activation. If the step is still active after the delay, the action is active until the step becomes inactive
P	Pulse	When the step is activated/deactivated, the action is executed only once
SD	Stored and time Delayed	After the delay, the action is activated and remain active until it is reset
DS	Delayed and Stored	If the step is still active after a specific time delay, the action is activated and remains active until it is reset
SL	Stored and time limited	The action is activated when the step is activated and remains active for a certain time before being reset

When the qualifier L, D, SD, DS, or SL is used, a time value is required in the format of the TIME type.

[Example 5.43] Application example of the qualifier N.

Figure 5-50 Action Qualifier N



Expression	Type	Value
bVar1	BOOL	TRUE
bStart	BOOL	TRUE
bVar2	BOOL	TRUE

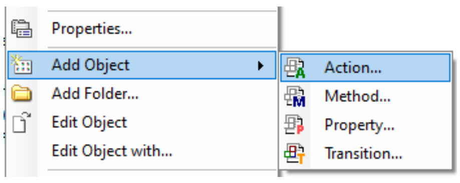
As described in Table 5-17, the qualifier "N" plays the following role: As long as the corresponding step is activated, the corresponding associated variable is also activated. As shown in the figure, bVar2 is set to ON every time Step0 is executed; otherwise, it is set to OFF. This qualifier can be used to monitor the step execution state.

The qualifiers L, D, SD, DS, and SL require a time value in the format of a time constant, i.e. T#(value)(unit). For example, the time value of 5 s is expressed as: T#5S.

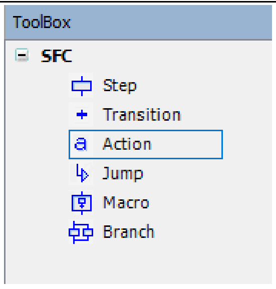
Action configuration

You can find the POU of the SFC programming language in the device tree, right-click and select "Add Object", and then select an action, as shown in Figure 5-51 a), or you can directly use the Toolbar and press the "a Action " button to add an action. The Toolbar is shown in Figure 5-51 b).

Figure 5-51 Add Actions 1



a) Add actions from the device tree

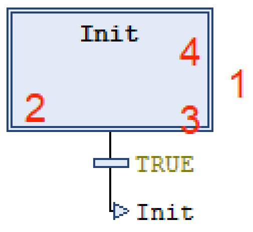


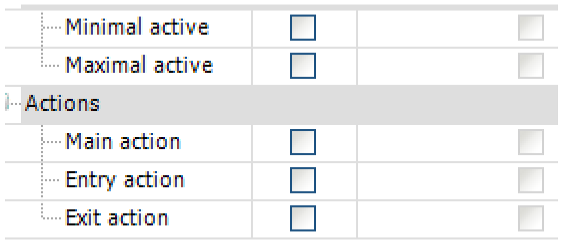
b) Add actions from the Toolbox

If you use the second method, select an "Action" in the Toolbar drag it to the top of the step. Then, four gray boxes will be displayed, as shown in Figure 5-52. You can drag the action into the corresponding box. After dragging, the corresponding settings will be added to the "Step Properties" corresponding to the step, as shown on the right side of Figure 5-52.

The "1" in Figure 5-52 is the action of the IEC standard step. The CoDeSys control platform extends the IEC standard actions and adds three additional actions: "step entry", "step exit", and "step active". The corresponding three extended actions are "2", "3", and "4" respectively.

Figure 5-52 Add Actions 2



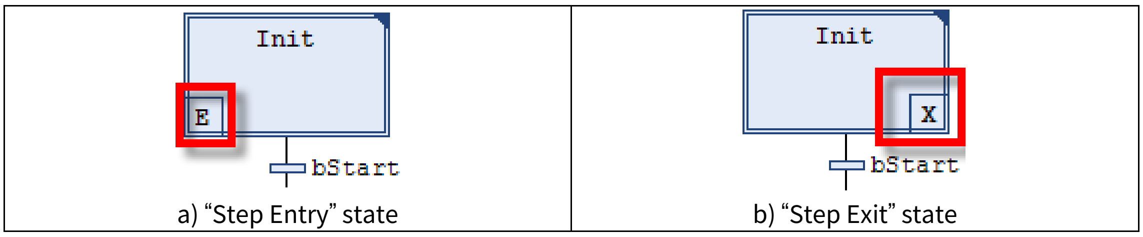


The specific step actions corresponding to "1", "2", "3", and "4" shown in Figure 5-52 are explained as follows.

1. Step entry

It refers to an action performed before the step is activated. The step actions will be executed as long as the step is activated by the program before the "step active" actions. The action is associated with the step via an entry in the "Step Entry" field of the "Step Properties". It is indicated by an "E" in the lower left corner of the step, as shown in Figure 5-53 a).

Figure 5-53 "Step Entry" and "Step Exit" states



a) "Step Entry" state b) "Step Exit" state

2. Step exit

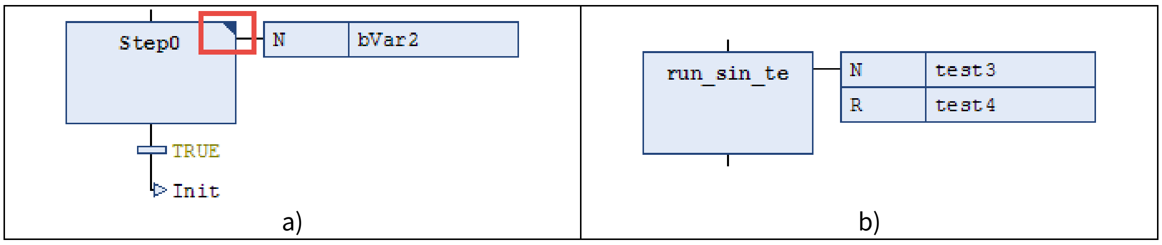
This action will be executed in the next cycle after the step is executed. When the step is invalid, it will be executed once. The execution will not be in the same cycle but at the beginning of the next execution cycle. The action is associated with the step via an exit in the "Step Exit" field of the "Step Properties". It is indicated by an "X" in the lower right corner of the step, as shown in Figure 5-53 b).

3. Step action

When a step is activated, the step actions are executed and possible entries have been completed. After the "Step Entry" of the step is executed, the step actions will be executed when the step is activated. However, unlike an IEC step action, the actions will not be further executed when they are invalid and they cannot be assigned qualifications.

The actions are associated with the step via an entry in the "Step Active" field of the "Step Properties". It is indicated by a small triangle in the upper right corner of the step, as shown in Figure 5-54 a).

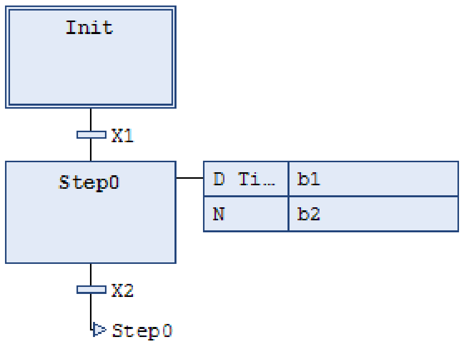
Figure 5-54 "Step Active" State



a) b)

```
PROGRAM PLC_PRG
VAR
        b1,b2,b3: BOOL;
        X1, X2: BOOL;
        Time1:TIME:=T#5S;
END_VAR
```

Figure 5-55 “Step Activated” State



As shown in the figure, when the transition condition variable X1 is TRUE, the program will execute the step Step0. At the same time, the corresponding step activation state variable b2 is TRUE. Since the qualifier corresponding to b1 is D, and the specific time is defined in the declaration area of the program as the variable Time, namely 5s, after Step0 is executed for 5s, the b1 variable is set from FALSE to TRUE.

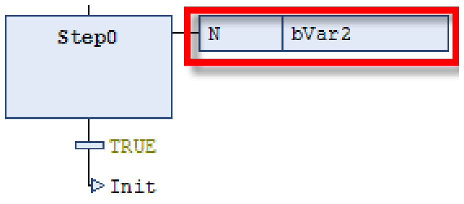
4. Step association action

Step association actions include Insert Action Association and Insert Action Association After, as shown in Table 5-19.When the current step is used as an IEC standard step, first click on the step, such as Step0, and then select “SFC” → “Insert Action Association” to associate the IEC step action with the step. A step can be associated with one or more actions.The position of the new action is determined by the current cursor position and the instruction used. The action must be available in the project and must be inserted with a unique action name, as shown in Figure 5-56.

Table 5-18 Step Association Actions

Type	Graphic Symbol	Description
Insert Action Association	↑	Associate an action with the step
Insert Action Association After	↓	Associate a further action with step after an existing one.

Figure 5-56 Add an Associated Action to the IEC Standard Step



The IEC standard step is executed at least twice: the first time when it is activated and the second time in the next cycle when it is deactivated.

Since multiple actions can be assigned to a step, these actions are executed in sequence from top to bottom. For example, the action Action_AS1 is associated to the step AS1, and a step action and an IEC action with the qualifier N are added respectively. In both cases, assuming that the transition conditions have been met, it takes 2 cycles to reach the initial step again. Assuming that a variable iCounter is incremented in Action_AS1, after the step Init is activated again, the value of iCounter in the step action example is 1. In contrast, the value of iCounter for the IEC action with the qualifier N is 2.

5.5.2.3 Transition

The switching between steps is simply called transition. The value of a transition condition must be TRUE or FALSE, so it can be a Boolean variable, Boolean address, or Boolean constant. A step transition can only be performed if the step transition condition is TRUE. That is, after the action of the previous step is executed, if there is an exit action, the exit action is executed once; if there is an entry action in the next step, the entry action of the next step is executed once, and then all the actions of the active step are executed according to the control cycle.

The program organization unit written in a sequential function chart contains a series of steps, which are connected via directed links (transition conditions). The operations associated with step transition in a sequential function chart are shown in Table 5-19.

Table 5-19 Transition Operations in the SFC

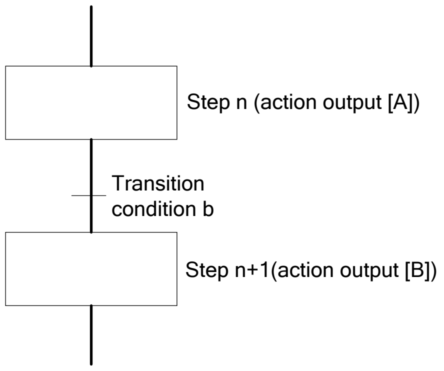
Type	Graphic Symbol	Description
Insert Step-Transition	⊤↑	Insert a transition condition before a step
Insert Step-Transition After	⊤↓	Insert a transition condition after a step

Generally speaking, there are different transition modes. The following are several transition modes that are commonly used in sequential function charts. They will be introduced one by one below.

1. Serial transition

Serial transition refers to transition to the next to-be-executed step in a serial connection when the transition condition is met.

Figure 5-57 Serial Branch Transition



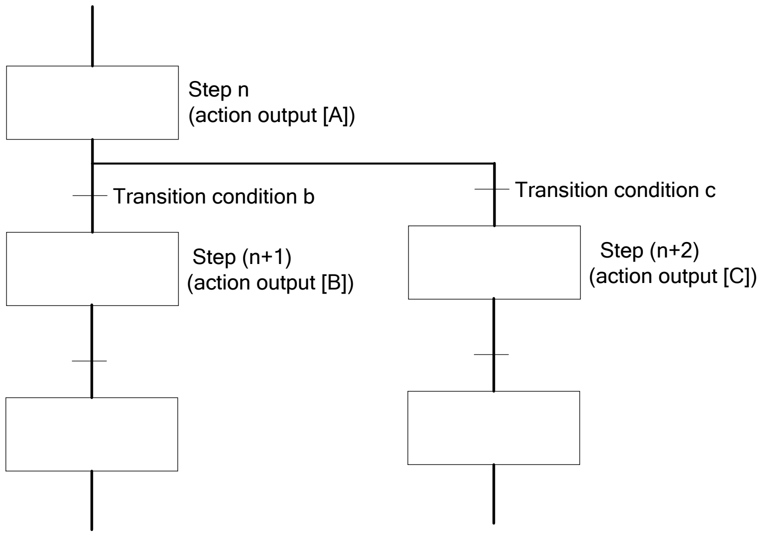
As shown in the figure, when the action output [A] of the step n is executed, if the transition condition b is met, the action output [A] is not executed and the action output [B] of the step (n+1) is executed.

2. Alternative transition

Alternative transition refers to executing only the step whose transition condition is met first among multiple steps connected in parallel.

A. Alternative branch transition

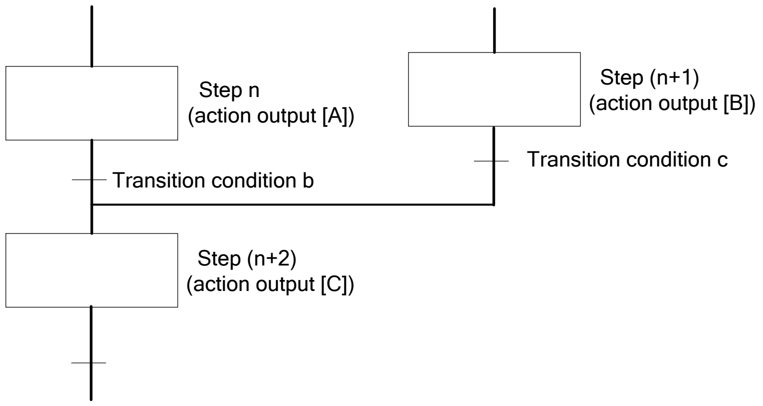
Figure 5-58 Alternative Branch Transition



- When the action output [A] of the step n is executed, the step (step (n+1) or step (n+2)) whose transition condition is met first among transition conditions b or c is selected, and the action output ([B] or [C]) of that step is executed.
- When the transition conditions are met at the same time, the transition condition on the left takes precedence. The action output [A] of the step n is not executed.
- Once selected, the steps in the selected sequence are executed sequentially until a merge is performed.

B. Alternative merge transition

Figure 5-59 Alternative Merge Transition



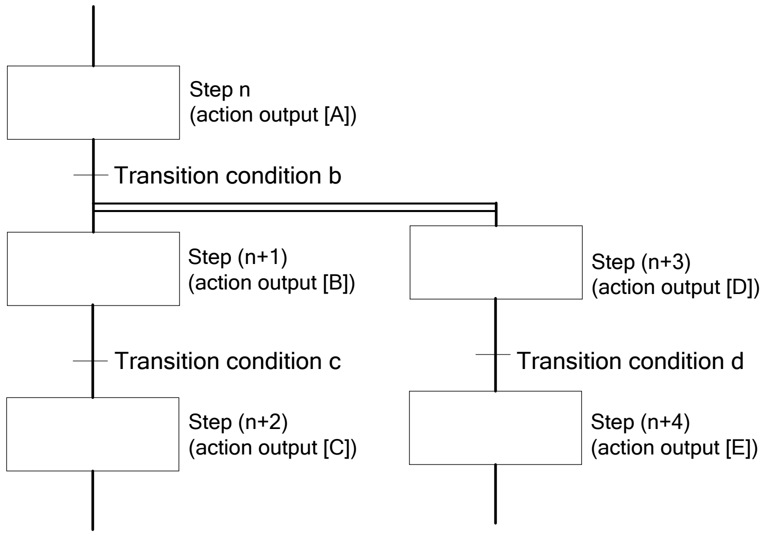
If the transition condition (b or c) of the execution sequence in the branch is met, the action output ([A] or [B]) of the step is not executed, and the action output [C] of the step (n+2) is executed.

3. Parallel transition

Parallel transition refers to executing multiple steps connected in parallel at the same time when the transition condition is met.

A. Parallel branch transition

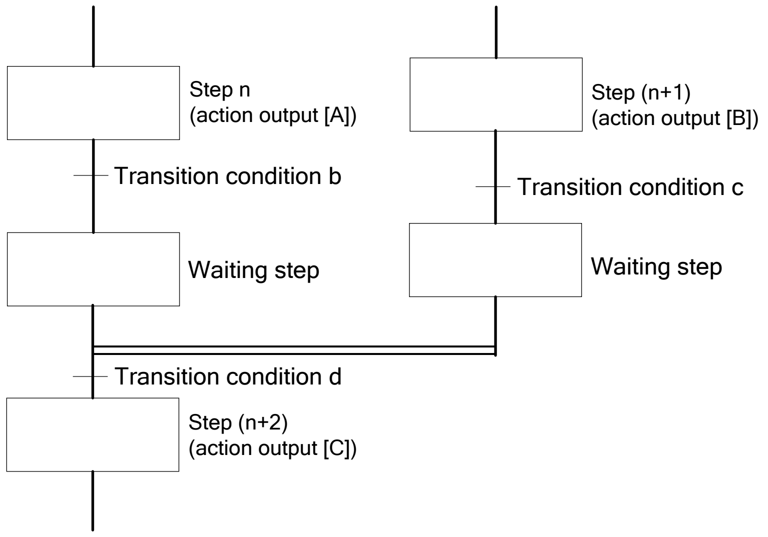
Figure 5-60 Parallel Branch Transition



- When the action output [A] of the step n is executed, if the transition condition b is met, the action output [B] of the step (n+1) and the action output [D] of the step (n+3) are executed simultaneously.
- When the transition condition c is met, the program transits to the step (n+2), and when the transition condition d is met, the program transits to the step (n+4).

B. Parallel merge transition

Figure 5-61 Parallel Merge Transition 1



- When the action output [A] of the step n and the action output [B] of the step (n+1) are executed, if the transition conditions b and c are met, the action output [A] of the step n and the action output [B] of the step (n+1) are not executed and the program transits to the waiting step.
- The waiting step is used to synchronize the steps executed in parallel. By transiting all the steps executed in parallel to the waiting step, the transition condition d is checked. If the transition condition d is met, the action output [C] of the step (n+2) is executed.
- The waiting step is regarded as a virtual step, and it does not matter even if there is no action output ladder diagram.

5.5.2.4 Jump

A jump refers to transition to a specified step in the same POU when a transition condition is met. It is indicated by a vertical line and horizontal arrow and the jump target name, as shown in ↳▷ Step0.

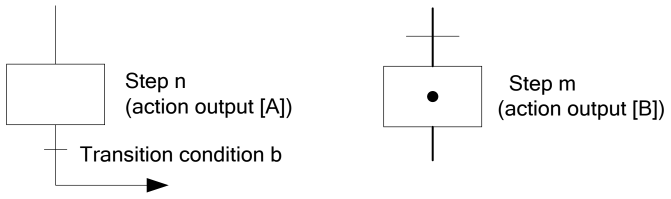
A jump defines the step to be executed when the subsequent transition is TRUE. According to the program execution sequence, the program cannot be cross-executed or executed upward, so a jump is needed. A jump can only be used at the end of a branch. When the last transition is selected, it can be inserted through the “Insert Jump” instruction, and the executable operations of the jump are shown in the figure below.

Table 5-20 Jump Elements in an SFC

Type	Graphic Symbol	Description
Insert Jump	⤷↑	Add a jump before a step.
Insert Jump After	⤷↓	Add a jump after a step.

The jump target can be given by an associated text string, which can be edited inline. It can be a step name or a label for a parallel branch, as shown in the figure below.

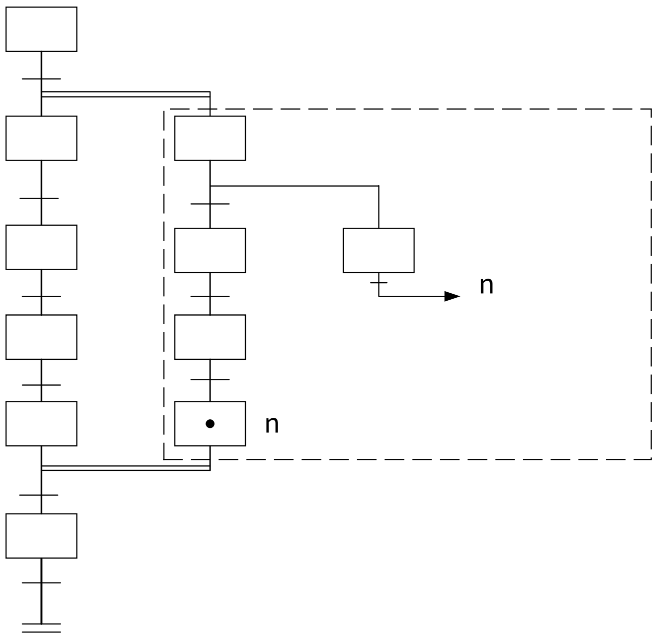
Figure 5-62 Parallel Merge Transition 2



When the action output [A] of the step n is executed, if the transition condition b is met, the action output [A] is not executed and the action output [B] of the step m is executed.

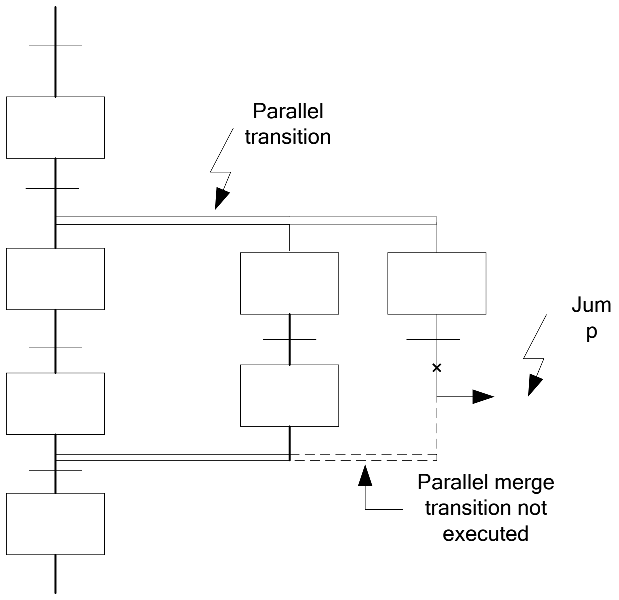
When a jump is executed within a parallel transition, it can only be executed in each vertical direction of the branch. For example, a jump in the vertical direction from the branch to the merge, as shown in Figure 5-63.

Figure 5-63 Parallel Merge Transition 3



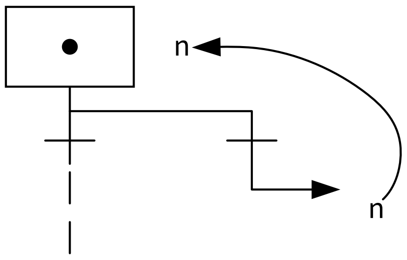
The jump programs shown in Figure 5-64 cannot be created: jumps to other vertical ladder diagrams within a branch, jumps to the outside of a parallel branch, and jumps from the outside of a parallel branch to the inside of the parallel branch. For example, a jump to the outside of a parallel branch (It cannot be specified).

Figure 5-64 Parallel Merge Transition 4



For example, when the transition condition shown in Figure 5-65 is met, a jump to the current step should not be specified.If a jump to the current step is specified, it will not operate normally.

Figure 5-65 Parallel Merge Transition 5



Jump creation

Find "Jump" in the Toolbox to insert a jump, as shown in Figure 5-66 a). Then, you only need to enter the jump target name, as shown in Figure 5-66 b). The jump target is Step0, so you just need to write Step0.

Figure 5-66 Jump Creation

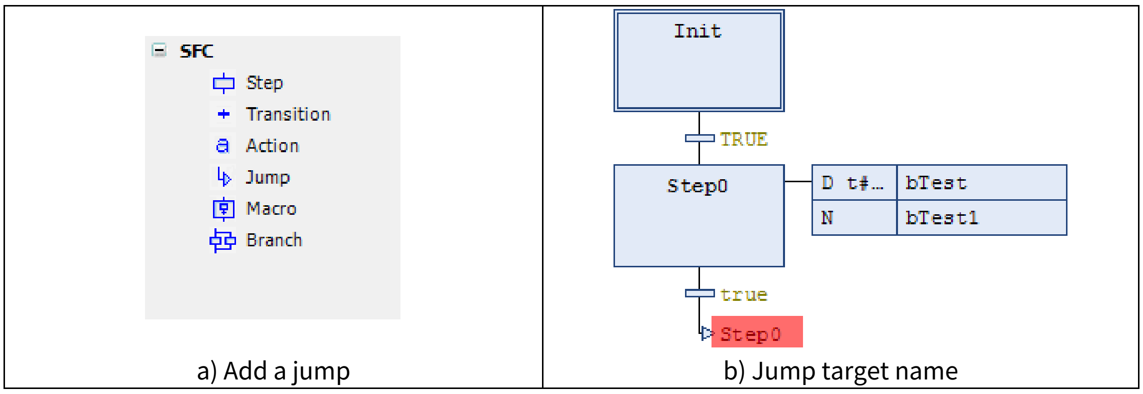
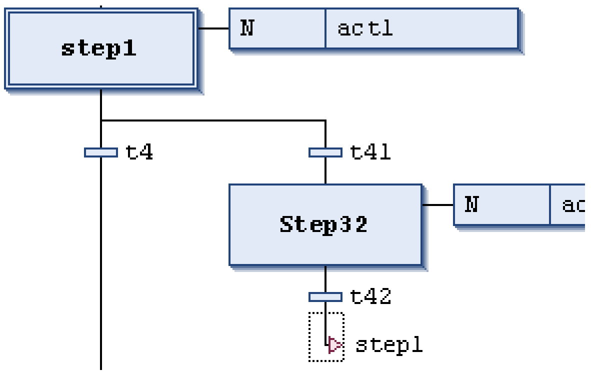


Figure 5-67 shows a typical application of the jump instruction. When the jump instruction t42 condition is met, it will automatically jump to step1 according to the program instruction and re-enter the program loop.

Figure 5-67 Typical Application of the SFC Jump Instruction



5.5.2.5 Macro

Just like the definition of a macro in other software, the main function of a macro in SFC programs is to avoid a lot of repetitive work. You only need to define a macro in advance and then call it in the program. Common operations on a macro are listed in Table 5-21.

Table 5-21 Macro Elements in an SFC

Type	Graphic Symbol	Description
Insert Macro	⊡↑	Insert a macro
Add Macro	⊡↓	Add a macro
Enter Macro	▷	Open the Macro Editor view
Exit Macro	◁	Return to the SFC standard view

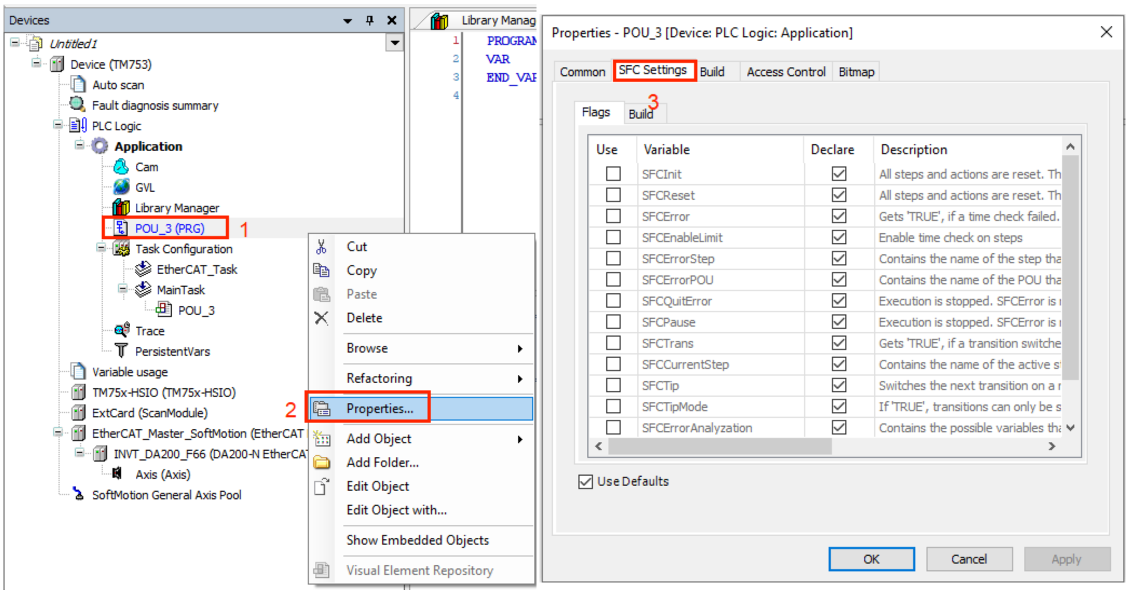
Implicit variables

Each SFC step and IEC action provides implicitly generated variables for runtime monitoring of the step and IEC action. It is also possible to define the variables to monitor and control SFC execution (timeout, restart, spike mode). The types of these implicit variables are defined in the library IecSFC.library.

This library is automatically added when an SFC object is added.

In the SFC programming language, some implicit variables can be called externally. In normal conditions, these variables are not displayed. To use these variables, you need to set the SFC properties. Right-click the “Properties” of the POU of the SFC language, click the “SFC Settings” option in the pop-up Properties dialog box, and check the variables you need to use, as shown in Figure 5-68.

Figure 5-68 Implicit Properties of an SFC



Use	Variable	Declare	Description
☐	SFCInit	☑	All steps and actions are reset. Th
☐	SFCReset	☑	All steps and actions are reset. Th
☐	SFCError	☑	Gets 'TRUE', if a time check failed.
☐	SFCEnableLimit	☑	Enable time check on steps
☐	SFCErrorStep	☑	Contains the name of the step tha
☐	SFCErrorPOU	☑	Contains the name of the POU tha
☐	SFCQuitError	☑	Execution is stopped. SFCError is r
☐	SFCPause	☑	Execution is stopped. SFCError is r
☐	SFCTrans	☑	Gets 'TRUE', if a transition switche
☐	SFCCurrentStep	☑	Contains the name of the active s
☐	SFCTip	☑	Switches the next transition on a r
☐	SFCTipMode	☑	If 'TRUE', transitions can only be s
☐	SFCErrorAnalyzation	☑	Contains the possible variables tha

In order to access these flags and make them work, they must be declared and activated. You can set them in the "SFC Settings" dialog box. It is a child dialog box of the "Object Properties" dialog box. If you want to use this variable, you must check the "Enable" box in front of the variable. The specific usage of the variable is also explained in its description.

5.6 Continuous Function Chart (CFC)

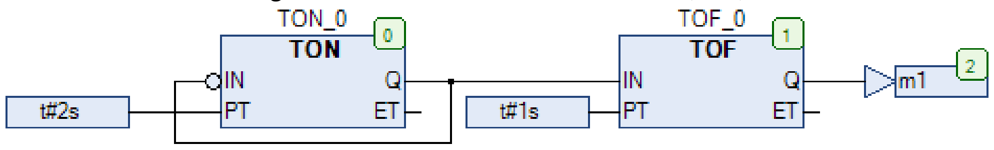
5.6.1 Continuous Function Chart Programming Language Structure

5.6.1.1 Introduction

Continuous Function Chart (CFC) is actually another form of FBD. In the whole program, the sequence of operation blocks can be customized to facilitate the implementation of process operations. It is used to describe the top-layer structure of resources and the allocation of tasks to programs and function blocks.

The main difference between a continuous function chart and a function block chart lies in resource and task allocation. Each function is described by a task name, as shown in the figure below. If a function block within a program is executed under the same task as its parent program, the task association is implicit. The Continuous Function Chart (CFC) is shown in Figure 5-69.

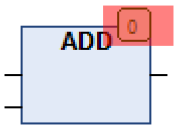
Figure 5-69 Continuous Function Chart (CFC)



5.6.1.2 Execution Sequence

The number in the upper right corner of the element in the CFC language shows the execution sequence of the element in the CFC in online mode. The execution process starts from the element numbered 0. In each PLC operation cycle, the element numbered 0 is always the first to be executed. When the element is moved manually, its number remains the same. When a new element is added, the system automatically assigns a number according to the topological sequence (from left to right, from top to bottom), as indicated by the red part in Figure 5-70.

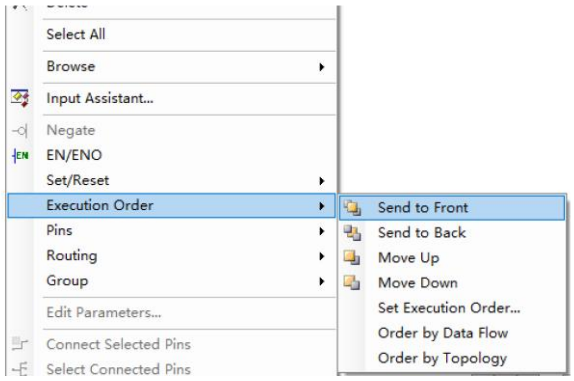
Figure 5-70 Sequence Number in the CFC Programming Language



The numbers in the upper right corner of the operation block, output, jump, return, and label elements in the CFC language show the execution sequence of the elements in CFC in online mode. The execution process starts from the element numbered 0. Considering that the execution sequence will affect the results,it can be changed under certain circumstances. The execution sequence of the element can be changed by using the sub-menu instructions in the “Execution Sequence” under the “CFC” menu.

The execution sequence includes the following instructions: Send to Front, Send to Back, Move Up, Move Down, Set Execution Sequence, Order by Data Flow, Order by Topology, as show in Figure 5-71.

Figure 5-71 CFC Sequence Arrangement



1. Send to Front

Move the selected element to the beginning of the execution sequence. If multiple elements are selected to execute this instruction, the original internal sequence of selected elements remains unchanged, and the internal sequence of unselected elements also remains unchanged.

2. Send to Back

Move all selected elements to the end of the execution sequence. The internal sequence of selected elements remains unchanged, and the internal sequence of unselected elements also remains unchanged. For specific operations, please refer to the above-mentioned “Send to Front” function.

3. Move Up

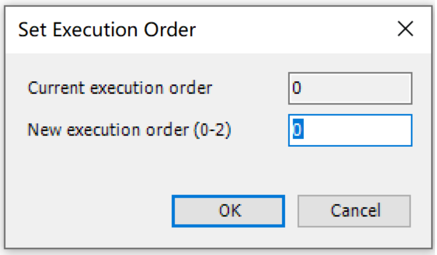
Move all selected elements (except for the element which has been already at the beginning of the execution sequence) one place forwards in the execution sequence. For example, if you select the element No. 3 in Figure 7 and execute the “Move Up” instruction, the result is that the execution sequence of elements No. 2 and No. 3 is swapped, and the rest elements remain unchanged.

4. Move Down

Move all selected elements (except for the element which has been already at the end of the execution sequence) one place backwards in the execution sequence. For specific operations, please refer to above-mentioned “Move Up” function.

5. Set Execution Sequence

This instruction can renumber the selected elements and adjust the their execution sequence. Once the “Set Execution Sequence” instruction is executed, the “Set Execution Sequence” dialog box will be opened. The current element number is displayed in the “Current Execution Sequence” field. You can enter the desired element number in the “New Execution Sequence” field. The possible values are displayed in brackets, as shown in the figure below.


Set Execution Order

Current execution order 0

New execution order (0-2) 0

6. Order by Data Flow

The "Order By Data Flow" instruction means the execution sequence is determined by the data flow of the elements rather than by their positions (topology). Once the "Order By Data Flow" instruction is executed, the CFC editor will perform the following operations.

Step 1 Order the elements topographically.

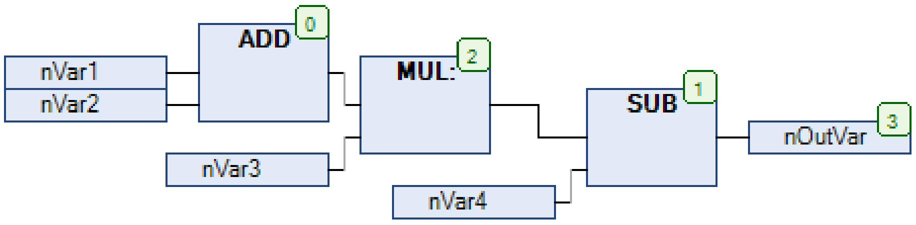
Step 2 Create a new sequential processing list.

Step 3 Based on the known values of the inputs, calculate which of the elements not yet numbered can be processed next.

The advantage of the "Order By Data Flow" instruction is that after an algorithm is executed, the algorithm block connected to its output pin will be executed immediately, which is not always so in the case of "Order by Topology". The execution result of the "Order by Topology" instruction may be different from that of the "Order By Data Flow" instruction.

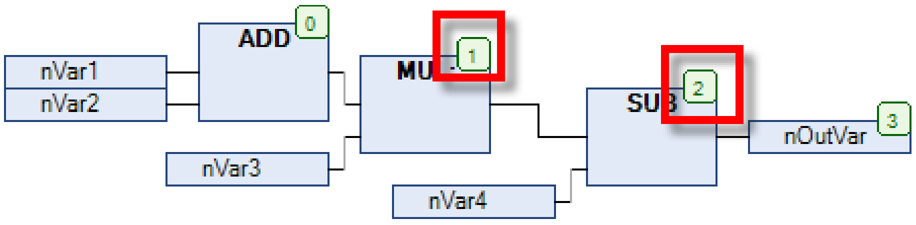
[Example 5.44] The figures below show how to view the results using the "Order By Data Flow" instruction after the element labels are disrupted.

Figure 5-72 View Before Using the "Order By Data Flow" Instruction



After selecting all elements and executing the "Order By Data Flow" instruction, the result is as shown in Figure 5-73.

Figure 5-73 View After Using the "Order By Data Flow" Instruction



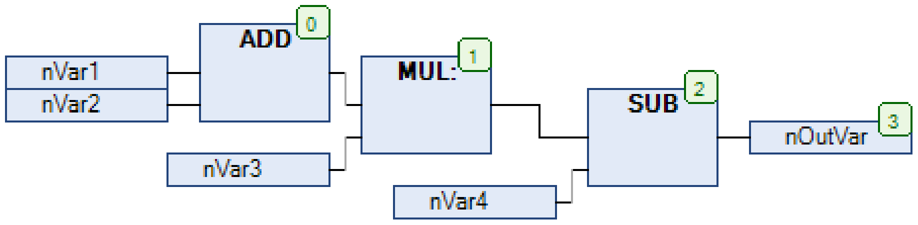
The element numbers are re-arranged in the order of data flow, and the execution sequence of the functions MUL and SUB has also changed.

7. Order by Topology

The "Order by Topology" instruction means that the execution sequence is determined by the topological order of the elements rather than by the data flow. Once the "Order by Topology" instruction is executed, the elements are executed from left to right and from top to bottom. The element numbers, indicating the position of an element within the execution sequence, increase from left to right and from top to bottom. In this case, the position of the connecting line is not relevant, only the location of the element is important.

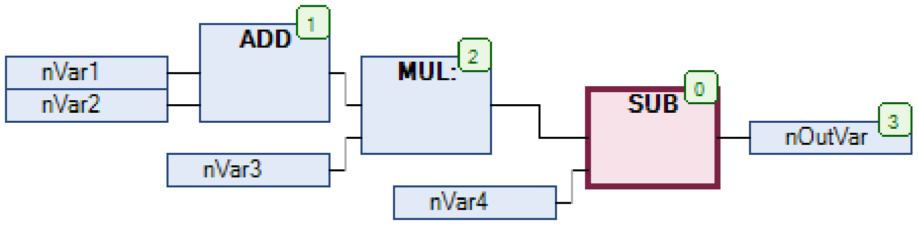
[Example 5.45] Figure 5-74 shows disrupted element labels. Use the "Order by Topology" instruction to view the results.

Figure 5-74 View Before Using the “Order by Topology” Instruction



Select the SUB function, right-click and execute the “Order by Topology” instruction. The result is shown in Figure 5-75.

Figure 5-75 View After Using the “Order by Topology” Instruction

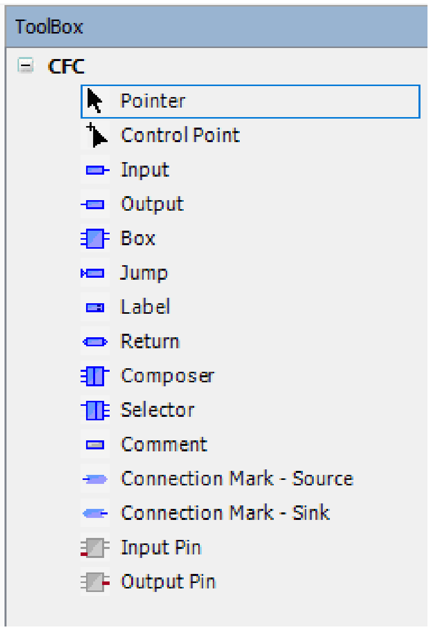


The execution sequence follows the rule below: the elements are executed from left to right and from top to bottom, and the element numbers, indicating the position of an element within the execution sequence, increase from left to right and from top to bottom.

5.6.2 Link Element

The CFC elements include Block, Input, Output, Jump, Label, Return, and Comment.

Figure 5-76 CFC Toolbox



5.6.2.1 Pointer

The pointer is at the top of the Toolbox list by default. As long as this entry is selected, the cursor has the shape of an arrow and you can select elements in the editor window for positioning and editing.

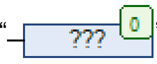
5.6.2.2 Input and Output

Input

You can insert the “▭-” symbol in the CFC Toolbox list to add the Input function. The graphic after insertion is “ ??? ”.

You can select the text offered by "???" and replace it with a variable or constant. You can also use the Input Assistant to select a valid identifier.

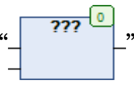
Output

You can insert the "" symbol in the CFC Toolbox list to add the Output function. The graphic after insertion is "

".

You can select the text offered by "???" and replace it with a variable or constant. You can also use the Input Assistant to select a valid identifier.

5.6.2.3 Block

You can insert the "" symbol in the CFC Toolbox list to add the Block function. The graphic after insertion is "

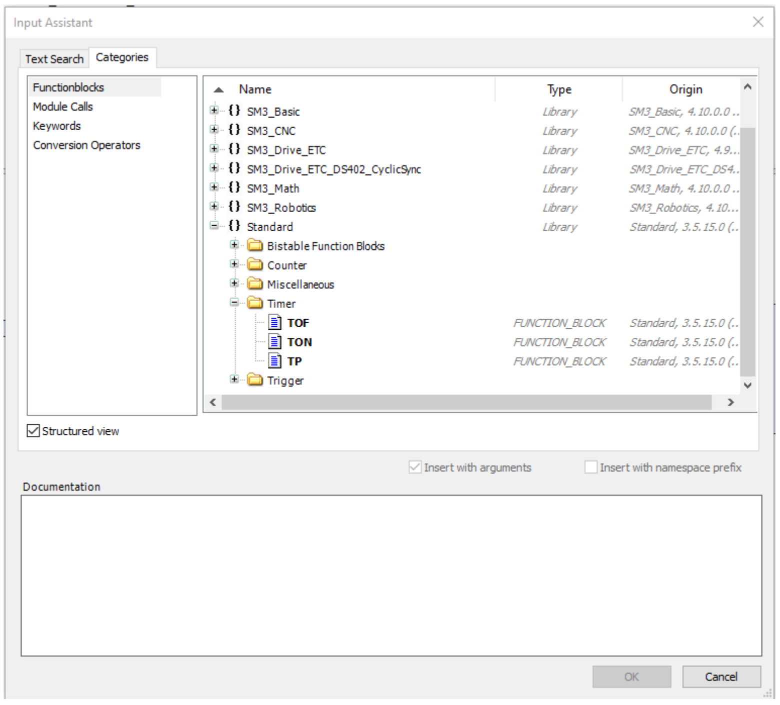
".

You can use a block to represent operators, functions, function blocks, and programs. You can select the text offered by "???" and replace it with an operator, function, function block, or program name after adding the Block function. Alternatively, you can use the Input Assistant to select one of the available objects.

[Example 5.46] Call the timer function block in the CFC programming language through the Block function.

Create a new POU, use the CFC programming language to add "Block", click "???", and enter "F2" to pop up the Input Assistant. Find and select the desired timer function block, as shown in Figure 5-77.

Figure 5-77 CFC Input Assistant Tool


Input Assistant

Text Search | Categories

Functionblocks
Module Calls
Keywords
Conversion Operators

Name	Type	Origin
{} SM3_Basic	Library	SM3_Basic, 4.10.0.0 ..
{} SM3_CNC	Library	SM3_CNC, 4.10.0.0 (..
{} SM3_Drive_ETC	Library	SM3_Drive_ETC, 4.9...
{} SM3_Drive_ETC_DS402_CyclicSync	Library	SM3_Drive_ETC_DS4..
{} SM3_Math	Library	SM3_Math, 4.10.0.0 ..
{} SM3_Robotics	Library	SM3_Robotics, 4.10...
{} Standard	Library	Standard, 3.5.15.0 (..
Bistable Function Blocks		
Counter		
Miscellaneous		
Timer		
TOF	FUNCTION_BLOCK	Standard, 3.5.15.0 (..
TON	FUNCTION_BLOCK	Standard, 3.5.15.0 (..
TP	FUNCTION_BLOCK	Standard, 3.5.15.0 (..
Trigger		

☑ Structured view

☑ Insert with arguments ☐ Insert with namespace prefix

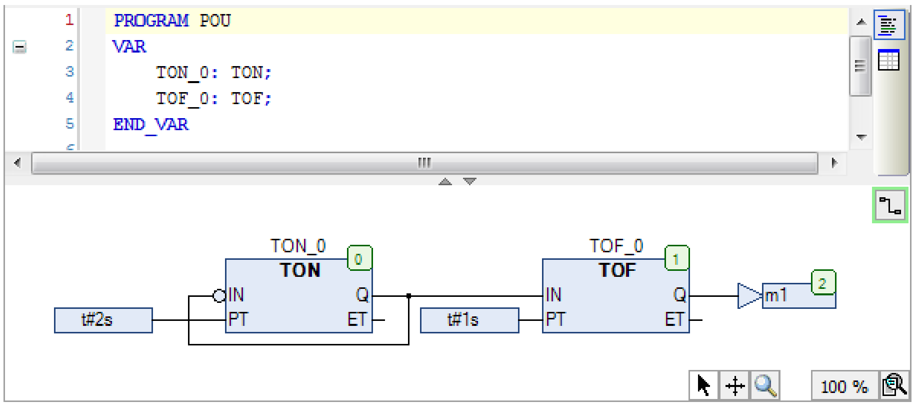
Documentation

OK Cancel

If you need to call a function block in the CFC programming language, you can directly enter the instance name of the function block and assign values or variables separated with commas to each parameter of the function block in the subsequent brackets. The function block call ends with a semicolon.

For example, call the TON timer function block in the CFC programming language. Assuming its instance name is TON1, the specific implementation is shown in Figure 5-78.

Figure 5-78 Function Block Call in the CFC Programming Language



```
PROGRAM POU
VAR
    TON_0: TON;
    TOF_0: TOF;
END_VAR
```

If you insert a function block, another “???” will be displayed above the block. You need to replace “???” with the name of the function block instance. In this example, the instance names are TON_0 and TOF_0.

If you replace an existing block with another (by modifying the entered name) and the new one has a different minimum or maximum number of input or output pins, the pins will be adapted correspondingly. If pins are to be removed, the lowest one will be removed first.

5.6.2.4 Jump and Label

The jump of a CFC program consists of two parts: jump instruction and label, which will be explained in detail below.

Jump

You can insert the “” symbol in the CFC Toolbox list to add the Jump function. The graphic after insertion is “??? 0”.

You can use the jump element to indicate at which position the execution of the program should continue. This position is defined by a “label” (see below). After inserting a new jump, you need to replace the text offered by “???” with the label name.

Label

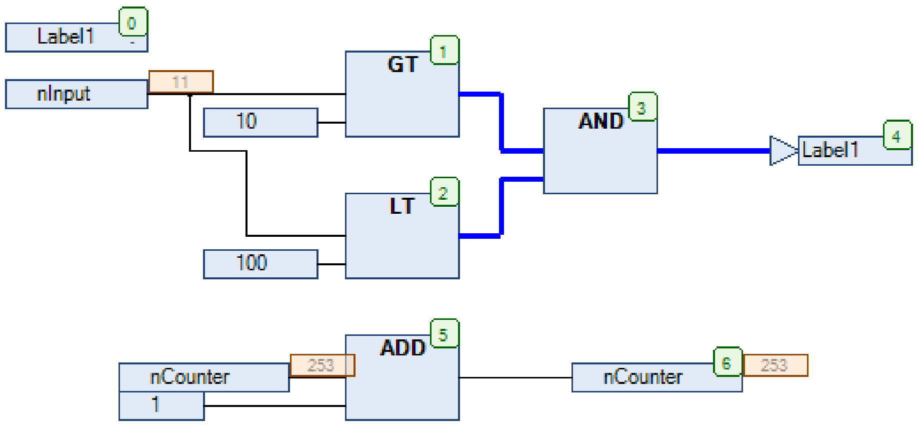
You can insert the “” symbol in the CFC Toolbox list to add the Label function. The graphic after insertion is “??? 0”.

A “label” marks the position to which the program can jump. In online mode, if a jump is activated, you can enter the label corresponding to the jump.

A label name is not a variable, so it does not need to be defined in the program declaration area. [Example 5.49] illustrates how to correctly use the jump instruction and label.

[Example 5.47] Examples of CFC jump instruction and label functions.

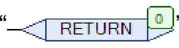
Figure 5-79 Example of CFC Jump Function



After the program starts, when the input value nInput is greater than 10 and less than 100, the program executes the jump function and goes to the label Label1. Since the execution sequence number of Label1 is 0, the execution sequence in this program is: 4→0→1→2→3→4, and performed in a loop.

Since the program has an auto-increment function, but the execution sequence numbers are 5 and 6, when the jump instruction is executed, the auto-increment function will not be executed by the program; otherwise, nCounter will be auto-incremented.

5.6.2.5 Return

You can insert the "" symbol in the CFC Toolbox list to add the Return function. The graphic after insertion is "
".

You need to pay special attention to the execution sequence number. When the condition is met, the program will be returned directly.

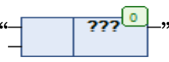
In online mode, a return element with the RETURN name is automatically inserted in the first column and after the last element in the editor. In a branch, it is automatically jumped to the place before execution leaves the POU.

The RETURN instruction is used to exit a program organization unit (POU).

Note: In online mode, a RETURN element is automatically inserted after the last element in the editor. In single-step debugging, it will automatically jump to the RETURN before leaving the POU.

5.6.2.6 Composer

You can use a composer to handle an input of a structure type operation block. The composer will display the structure components and thus make them accessible in the CFC for the programmer.

You can insert the "" symbol in the CFC Toolbox list to add the Composer function. The graphic after insertion is "
" .

The usage method of the composer is as follows: first add a composer to the editor, replace "???" with the name of the concerned structure, and then connect the output pin of the composer and the input pin of the operation block.

[Example 5.48] Process a function block instance fubblo1 with the CFC program CFC_PROG, which has an input variable struvar of structure type. By using the composer element, the structure type variable can be accessed:

Definition of the structure stru1:

```
TYPE stru1:
STRUCT
ivar:INT;
strvar:STRING:='hallo';
END_STRUCT
END_TYPE
```

Declaration and implementation of the function block fublo1:

```
FUNCTION_BLOCK fublo1
VAR_INPUT
struvar:STRU1;
END_VAR
VAR_OUTPUT
fbout_i:INT;
```

```
fbout_str:STRING;
END_VAR
VAR
fbvar:STRING:='world';
END_VAR
fbout_i:=struvar.ivar+2;
fbout_str:=CONCAT (struvar.strvar,fbvar);
```

Declaration and implementation of the program CFC_PROG:

```
PROGRAM PLC_PRG
VAR
intvar: INT; stringvar: STRING;
fbinst: fublo1;
erg1: INT;
erg2: STRING;
END_VAR
```

In the program, as shown in Figure 5-80, ‘1’ is a composer, and ‘2’ is stru1 containing a structure input variable. The operation of the structure type input block is implemented.

Figure 5-80 CFC Composer Example

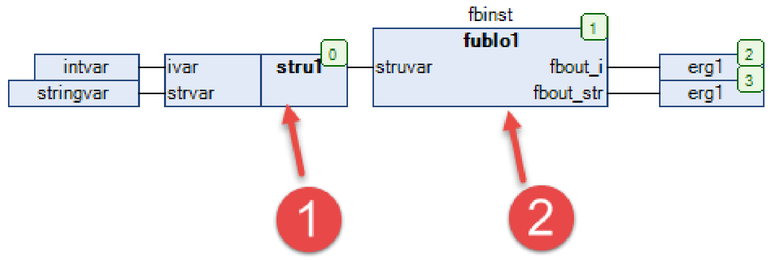
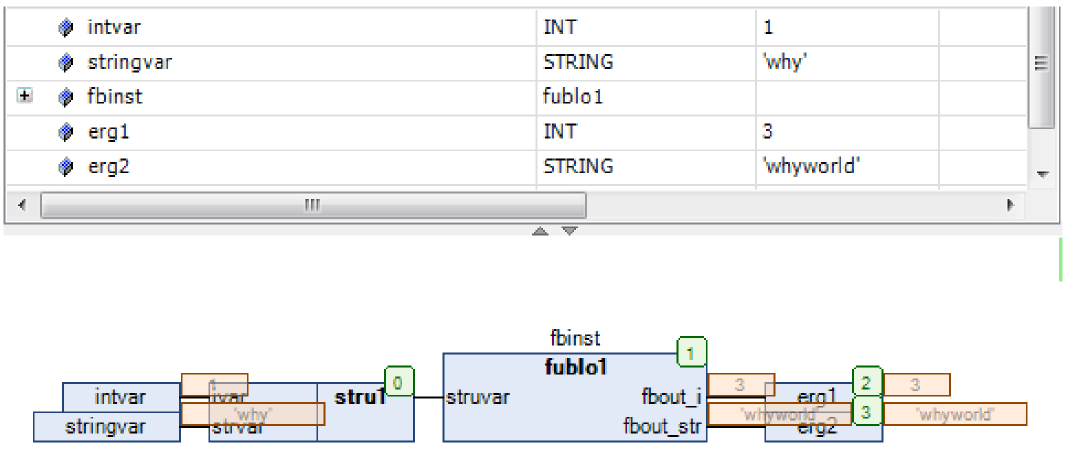


Figure 5-81 Operation Result of the CFC Composer Example



intvar	INT	1
stringvar	STRING	'why'
fbinst	fublo1	
erg1	INT	3
erg2	STRING	'whyworld'

5.6.2.7 Selector

You can use a selector to handle an output of a structure type operation block. The selector will display the structure components and thus make them accessible in the CFC for the programmer.

You can insert the “” symbol in the CFC Toolbox list to add the Selector function. The graphic after insertion is “???”.

The usage method of the selector is as follows: first add a selector to the editor, replace "???" with the name of the concerned structure, and then connect the output pin of the selector and the output pin of the operation block.

[Example 5.49] Process a function block instance fubblo2 with the CFC program CFC_PROG, which has an output variable fbout of stru1 structure type. By using the selector element, the structure type variable can be accessed:

Definition of the structure stru1:

```
TYPE stru1:
STRUCT
ivar:INT;
strvar:STRING:='hallo';
END_STRUCT
END_TYPE
```

Declaration and implementation of the function block fublo1:

```
FUNCTION_BLOCK fublo2
VAR_INPUT
fbin : INT;
fbin2:STRING;
END_VAR
VAR_OUTPUT
fbout : stru1;
END_VAR
VAR
fbvar:INT:=2;
fbin3:STRING:='Hallo';
END_VAR
```

Declaration and implementation of the program PLC_PRG_1:

```
PROGRAM PLC_PRG_1 VAR
intvar: INT;
stringvar: STRING;
fbinst: fublo2;
erg1: INT;
erg2: STRING;
fbinst2: fublo2;
END_VAR
```

In the program, as shown in Figure 5-82, ‘1’ is a function block with an output variable fbout of stru1 structure type, and ‘2’ is a selector.

Figure 5-82 CFC Selector Example

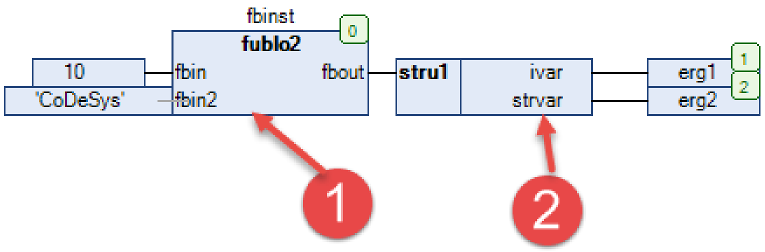
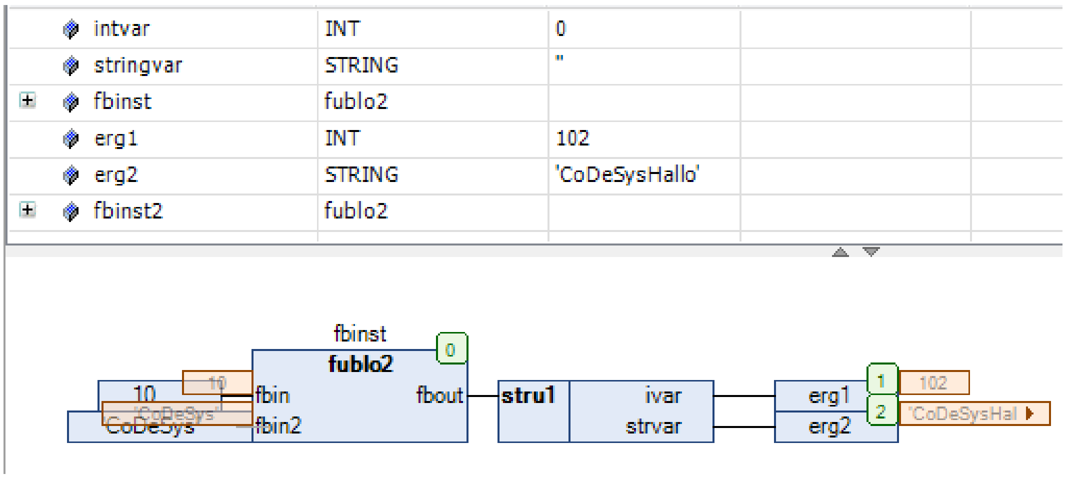


Figure 5-83 Operation Result of the CFC Selector Example



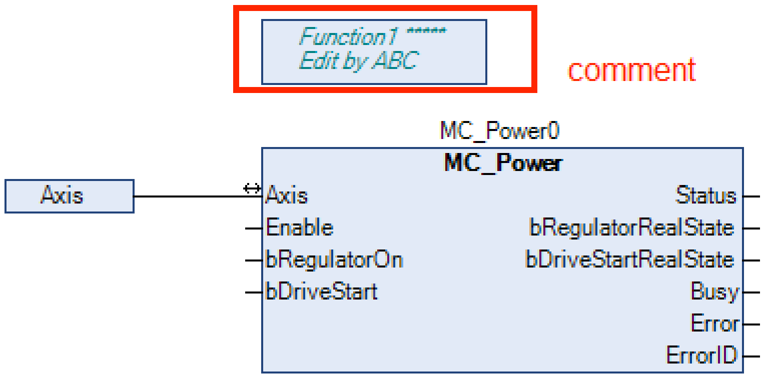
	intvar	INT	0
	stringvar	STRING	''
⊞	fbinst	fublo2	
	erg1	INT	102
	erg2	STRING	'CoDeSysHallo'
⊞	fbinst2	fublo2	

5.6.2.8 Comment

You can insert the “▭” symbol in the CFC Toolbox list to add the Comment function.

The graphic after insertion is “ *<Enter your comment here...>* ”.

You can use this element to add any comments to the chart in the CFC program. Select the placeholder text and replace it with any desired text. To obtain a new line within the comment, press <ctrl>+<enter>. The CFC Comment view is as shown below.



5.6.2.9 Input and Output Pins

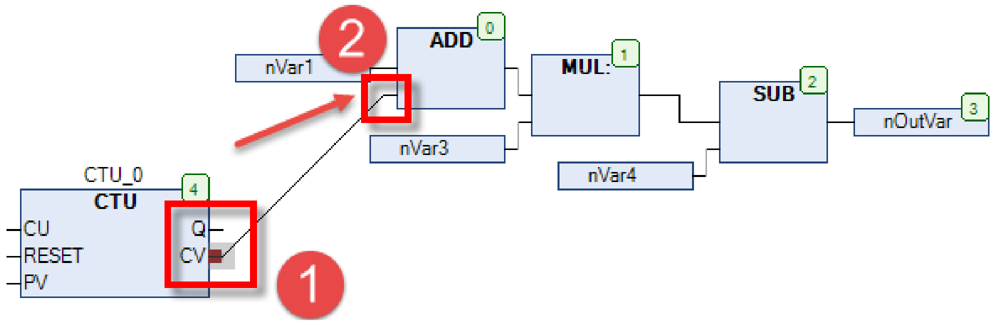
Depending on the block type, you can add an input pin (or output pin). For this purpose, select “Input Pin” (or “Output Pin”) in the Toolbox list, then drag and drop it onto the block in the CFC editor. At this time, an input pin (or output pin) will be added to the block.

5.6.3 CFC Configuration

1. Add a connection in the CFC program

When adding a connection, first activate the pin of the connection block. After activation, you will see a red filled square at the pin. Select the square with the left mouse button, as indicated by ‘1’ in Figure 5-84, hold down the mouse and draw a line to the other block to be connected, as indicated by ‘2’ in Figure 5-84, and then release the mouse. At this time, the connection between the two blocks is completed.

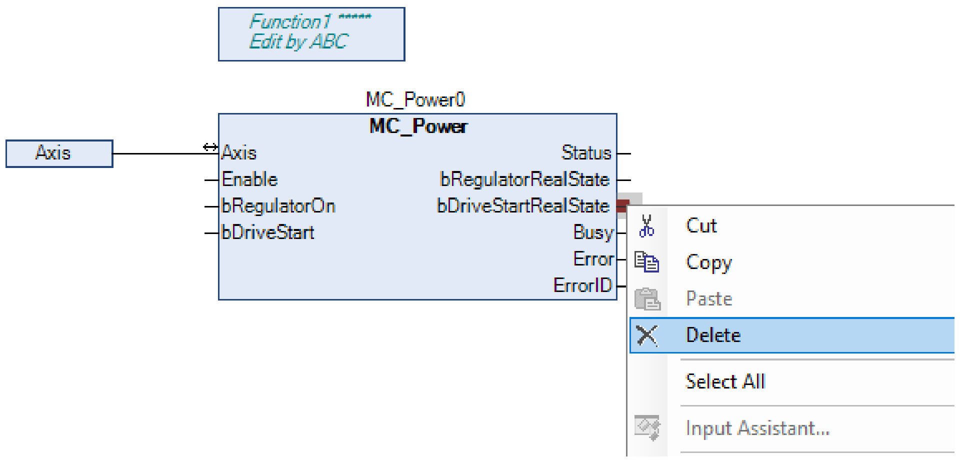
Figure 5-84 Add a Connection in the CFC Program



2. Delete a connection in the CFC program

When deleting a connection, first activate the pin of the connection block. After activation, you will see a red filled square at the pin. Right-click the square and select “Delete” in the menu bar that appears, as indicated by the framed part in Figure 5-85. You can also select the “ ✕ ” button in the shortcut menu bar to delete the connecting line in the program.

Figure 5-85 Delete a Connection in the CFC Program



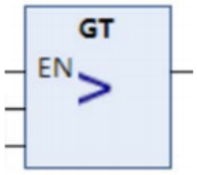
6 Basic Instructions

6.1 Comparison Instructions

6.1.1 Greater Than (GT)

Evaluate two input values: When the first input value is greater than the second input value, TRUE is output; otherwise, FALSE is output.

Example in FBD:

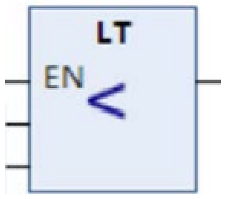


Note: When the data types of the two input variables are inconsistent, a compilation error will be reported.

6.1.2 Less Than (LT)

Evaluate two input values: When the first input value is less than the second input value, TRUE will be output; otherwise, FALSE will be output.

Example in FBD:

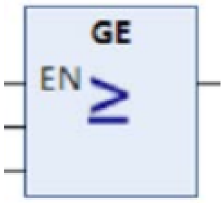


Note: When the data types of the two input variables are inconsistent, a compilation error will be reported.

6.1.3 Greater Than Or Equal To (GE)

Evaluate two input values: When the first input value is less than the second input value, TRUE will be output; otherwise, FALSE will be output.

Example in FBD:

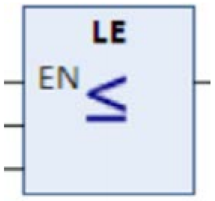


Note: When the data types of the two input variables are inconsistent, a compilation error will be reported.

6.1.4 Less Than Or Equal To (LE)

Evaluate two input values: When the first input value is less than or equal to the second input value, TRUE is output; otherwise, FALSE is output.

Example in FBD:

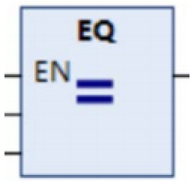


Note: When the data types of the two input variables are inconsistent, a compilation error will be reported.

6.1.5 Equal To (EQ)

Evaluate two input values: When the first input value is equal to the second input value, TRUE is output; otherwise, FALSE is output.

Example in FBD:

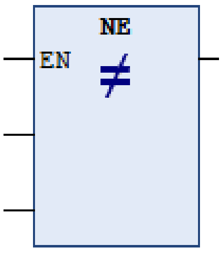


Note: When the data types of the two input variables are inconsistent, a compilation error will be reported.

6.1.6 Not Equal To (NE)

Evaluate two input values: When the first input value is not equal to the second input value, TRUE is output; otherwise, FALSE is output.

Example in FBD:



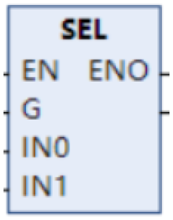
Note: When the data types of the two input variables are inconsistent, a compilation error will be reported.

6.2 Selection Instructions

6.2.1 Binary Selection (SEL)

When G=FLASE, IN0 is output; when G=TRUE, IN1 is output.

Example in FBD:



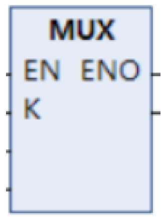
Note: When G is TRUE, CODESYS does not evaluate the expression before IN0. When G is FALSE, CODESYS does not evaluate the expression before IN1.

6.2.2 Multiplexer (MUX)

Select the k-th value from a group of values. The first value is K=0. If K is greater than the other input values, CODESYS transmits the last value

(INn).

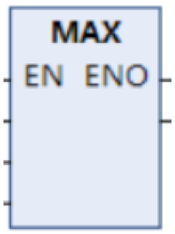
Example in FBD:

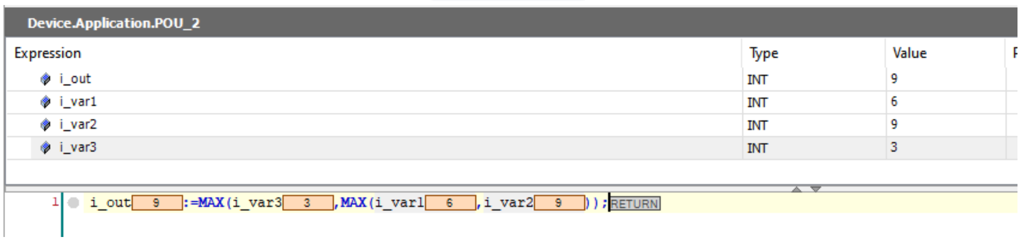


6.2.3 Maximum (MAX)

Take the maximum of the two input values and output the maximum value from the right side.

Example in FBD:





Device.Application.POU_2

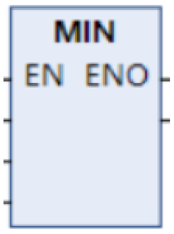
Expression	Type	Value
i_out	INT	9
i_var1	INT	6
i_var2	INT	9
i_var3	INT	3

```
i_out 9 :=MAX(i_var3 3 ,MAX(i_var1 6 ,i_var2 9 ));RETURN
```

6.2.4 Minimum (MIN)

Take the minimum of the two input values and output the minimum value from the right side.

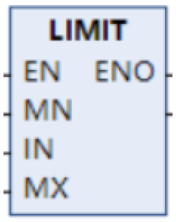
Program example:



6.2.5 Limit (LIMIT)

MX is the upper and MN the lower limit for the result. If the value IN exceeds the upper limit MX, LIMIT will return MX; if IN falls below the lower limit MN, the result will be MN. When the value IN is within the range of MN and MX, the result is the input value of IN.

Example in FBD:



Note: The MX and MN data types must be the same.

6.3 Counter Instructions

6.3.1 Counter Up (CTU)

This counter function block counts up.

Input:

CU:BOOL; if a rising edge is detected, CV is increased by 1

RESET: BOOL; if TRUE, CV is reset to 0

PV:WORD; the upper limit of CV count

Output:

Q:BOOL; TRUE if CV=PV

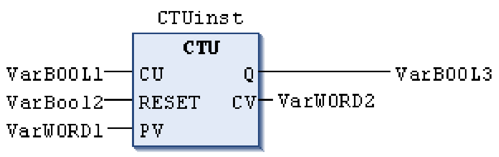
CV:WORD; continuously increased by 1 until CV

If the value of RESET is TRUE, CV is reset to 0. If a rising edge is detected on CU from FALSE to TRUE, CV is increased by 1. If CV is greater than or equal to PV, Q is TRUE.

Declaration example:

CTUInst:CTU;

Example in FBD:



Example in ST:

CTUInst(CU:=VarBOOL1,RESET:=VarBOOL2,PV:=VarWORD1);

VarBOOL3:=CTUInst.Q;

VarWORD2:=CTUInst.CV;

6.3.2 Count Down (CTD)

This counter function block counts down.

Input:

CD:BOOL; if a rising edge is detected, CV is decreased by 1

LOAD:BOOL; if TRUE, CV is set to PV

PV:WORD; the initial value when CV starts to decrease

Output:

Q:BOOL; TRUE if CV=0

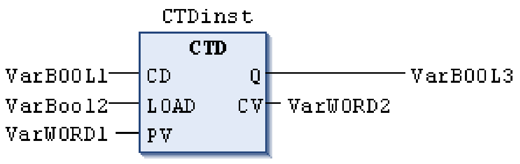
CV:WORD; continuously decreased by 1 until PV=0

If the value of LOAD is TRUE, CV is initialized to PV. If a rising edge is detected on CD from FALSE to TRUE and CV is greater than 0, CV is decreased by 1 (that is, CV cannot be less than 0). If CV is equal to 0, Q is TRUE.

Declaration example:

CTDInst:CTD;

Example in FBD:



Example in ST:

```
CTDInst(CD:=VarBOOL1,LOAD:=VarBOOL2,PV:=VarWORD1);
VarBOOL3:=CTDInst.Q;
VarWORD2:=CTDInst.CV;
```

6.3.3 Counter Up/Down (CTUD)

This counter function block counts up/down.

Input:

CU:BOOL; if a rising edge is detected, CV is increased by 1

CD:BOOL; if a rising edge is detected, CV is decreased by 1

RESET: BOOL; if TRUE, CV is reset to 0

LOAD:BOOL; if TRUE, CV is set to PV

PV:WORD; the upper limit value when CV starts to increase, or the initial value when CV decreases

Output:

QU:BOOL; TRUE if CV = PV

QD: BOOL; TRUE if CV=0

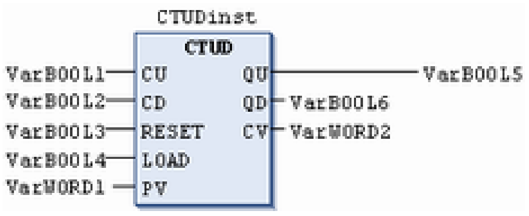
CV:WORD; continuously decreased by 1 until PV=0

If a rising edge is detected on CU from FALSE to TRUE, CV is increased by 1. If a rising edge is detected on CD from FALSE to TRUE and CV is greater than 0, CV is decreased by 1. If CV is greater than or equal to PV, QU is TRUE. If CV is equal to 0, QD is TRUE.

Declaration example:

CTUDInst:CUTD;

Example in FBD:



Example in ST:

CTUDInst(CU:=VarBOOL1,CD:=VarBOOL2,RESET:=VarBOOL3,LOAD:=VarBOOL4,PV:=VarWORD1);

VarBOOL5:=CTUDInst.QU;

VarBOOL6:=CTUDInst.QD;

VarWORD2:=CTUDInst.CV;

6.4 Timer Instructions

6.4.1 Pulse Timer (TP)

This timer function block creates a pulse.

Input:

IN:BOOL; if a rising edge is detected, ET starts timing

PT:TIME; the upper limit value of ET timing

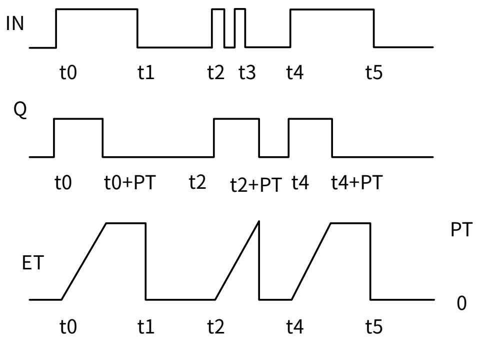
Output:

Q:BOOL; when ET is timing, its value is TRUE

ET:TIME; the current state of time

If the value IN is FALSE, Q is FALSE and ET=0. If the value IN is TRUE, the time in ET starts counting in milliseconds until ET=PT. After ET=PT, it will remain constant. If IN is TRUE and ET is less than or equal to PT, Q is TRUE; otherwise, Q is FALSE.

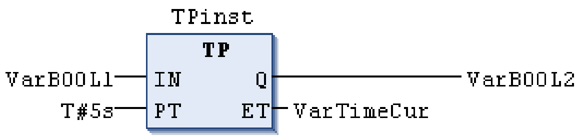
During the time period defined by PT, Q is TRUE. The time sequence diagram of TP is as follows:



Declaration example:

TPInst:TP;

Example in FBD:



Example in ST:

TPInst(IN:=VarBOOL1,PT:=T#5s);

VarBOOL2:=TPInst.Q;

6.4.2 On-delay Timer (TON)

This timer function block realizes an on-delay timing.

Input:

IN:BOOL; if a rising edge is detected, ET starts timing

PT:TIME; the upper limit value of ET timing (i.e. delay time)

Output:

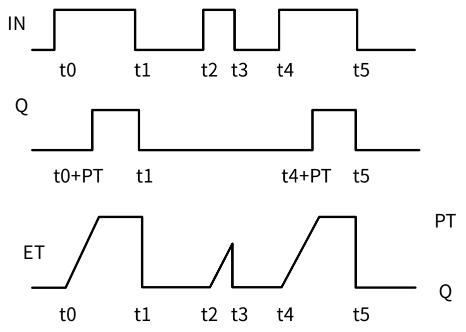
Q: BOOL; if the ET timing reaches PT, a rising edge is output

ET:TIME; the current state of time

TP(IN,PT,Q,ET): IN and PT are input variables of BOOL type and TIME type respectively. Q and ET are output variables of BOOL type and TIME type respectively. If the value IN is FALSE, Q is FALSE and ET=0.

If the value IN is TRUE, the time in ET starts counting in milliseconds until ET=PT. After ET=PT, it will remain constant. If IN is TRUE and ET=PT, Q is TRUE. Otherwise, Q is FALSE. Therefore, when the delay (the time defined by PT) elapses, a rising edge will be detected on Q.

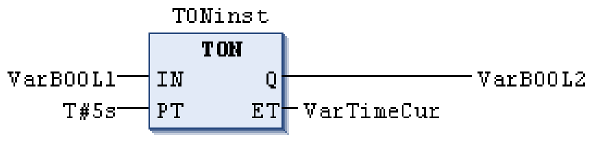
The time sequence diagram of TON is as follows:



Declaration example:

TONInst:TON;

Example in FBD:



Example in ST:

TONInst(IN:=VarBOOL1,PT:=T#5s);

6.4.3 Off-delay Timer (TOF)

This timer function block realizes an off-delay timing.

Input:

IN:BOOL; if a falling edge is detected, ET starts timing

PT:TIME; the upper limit value of ET timing (i.e. delay time)

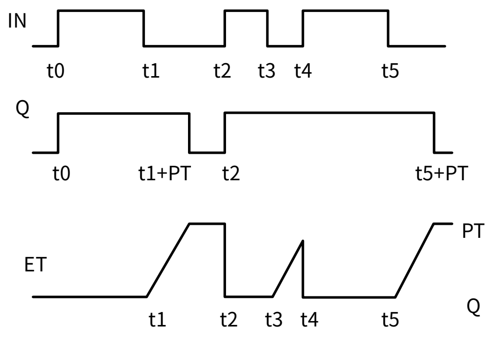
Output:

Q: BOOL; if the ET timing reaches PT, a falling edge is output

ET:TIME; the current state of time

TOF(IN,PT,Q,ET): if IN is TRUE, Q is TRUE. If the value IN is FALSE, the time in ET starts counting in milliseconds until ET=PT. After ET=PT, it will remain constant. If IN is FALSE and ET=PT, Q is FALSE; otherwise, Q is TRUE. Therefore, when the delay elapses, a falling edge will be detected on Q.

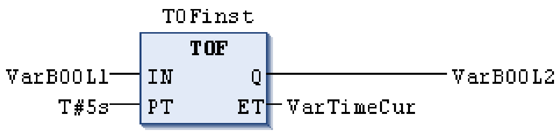
The time sequence diagram of TOF is as follows:



Declaration example:

TOFInst:TOF;

Example in FBD:



Example in ST:

TOFInst(IN:=VarBOOL1,PT:=T#5s);

VarBOOL2:=TOFInst.Q;

6.4.4 Real-time Clock (RTC)

This clock function block starts timing from the set time.

Input:

EN:BOOL; if a rising edge is detected, CDT starts timing

PDT:DATE_AND_TIME; the date and time when the timing starts

Output:

Q: BOOL; when CDT starts timing, the output is TRUE

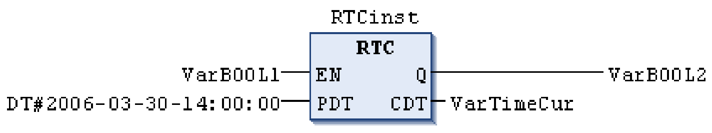
CDT:DATE_AND_TIME; the current date and time of the timer

VarBOOL2:=RTC(EN,PDT,Q,CDT): when EN is FALSE, the output variable Q is FALSE and CDT is DT#1970-01-01-00:00:00. Once EN becomes TRUE (a rising edge is detected), as long as EN remains TRUE, CDT is incremented in seconds with PDT as the initial value. When EN is reset to FALSE, CDT is reset to the initial value DT#1970-01-01-00:00:00.

Declaration example:

RTCInst:RTC;

Example in FBD:



Example in ST:

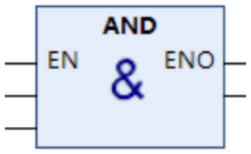
RTCInst(EN:=VarBOOL1,PDT:=DT#2006-03-30-14:00:00,Q=>VarBOOL2,CDT=>VarTimeCur);

6.5 Bit and Word Logic Instructions

6.5.1 AND Instruction

When the two input bits on the left side are non-zero, the output bit on the right side also outputs 1; otherwise, it outputs 0.

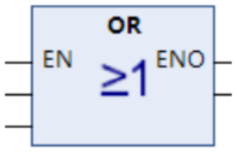
Example in FBD:



6.5.2 OR Instruction

When at least one of the two input bits on the left side is non-zero, the value of the output bit on the right side is 1; otherwise, it is 0.

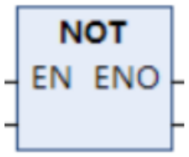
Example in FBD:



6.5.3 NOT Instruction

When the input bit is 0, the output bit on the right side outputs 1, and when the input bit on the left side is 1, the output bit on the right side outputs 0.

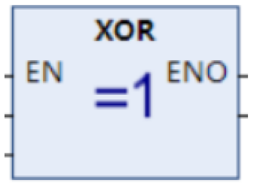
Example in FBD:



6.5.4 XOR Instruction

When one of the two input bits on the left side is 1 and the other is 0, the output is 1; when the two input values are both 1 or 0, the output is 0.

Example in FBD:



6.5.5 Set Dominant (SR)

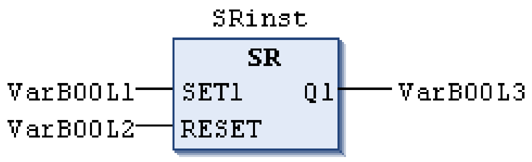
This bistable function block realizes a prior set. Q1=SR(SET1,RESET): Q1=(NOTRESETANDQ1)ORSET1.

The input variables SET1 and RESET and the output variable Q1 are all of BOOL type.

Declaration example:

SRInst:SR;

Example in FBD:



Example in ST:

SRInst(SET1:=VarBOOL1,RESET:=VarBOOL2);

VarBOOL3:=SRInst.Q1;

6.5.6 Reset Dominant (RS)

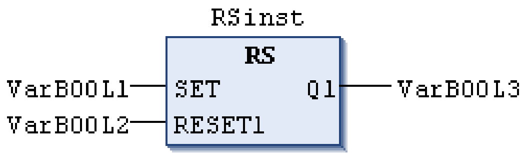
This bistable function block realizes a prior reset. Q1=RS(SET,RESET1): Q1=NOTRESET1AND(Q1ORSET).

The input variables SET1 and RESET and the output variable Q1 are all of BOOL type.

Declaration example:

RSInst:RS;

Example in FBD:



Example in ST:

RSInst(SET:=VarBOOL1,RESET1:=VarBOOL2);

VarBOOL3:=RSInst.Q1;

6.5.7 Rising Edge Detector (R_TRIG)

This edge detection function block detects a rising edge.

Input:

CLK: BOOL; the Boolean input signal is used to detect a rising edge

Output:

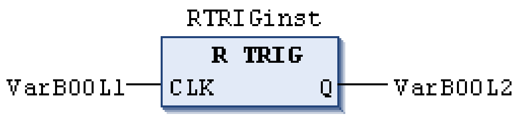
Q:BOOL; if CLK detects a rising edge, the output is TRUE

When CLK changes from “FALSE” to “TRUE”, the rising edge detector starts, the output Q changes from “TRUE” to “FALSE” and remains “FALSE” for one operation cycle of the PLC; if CLK continues to remain “TRUE” or “FALSE”, the output Q remains “FALSE”.

Declaration example:

RTRIGInst:R_TRIG;

Example in FBD:



Example in ST:

RTRIGInst(CLK:=VarBOOL1);

VarBOOL2:=RTRIGInst.Q;

6.5.8 Falling Edge Detector (F_TRIG)

This edge detection function block detects a falling edge.

Input:

CLK: BOOL; the Boolean input signal is used to detect a falling edge

Output:

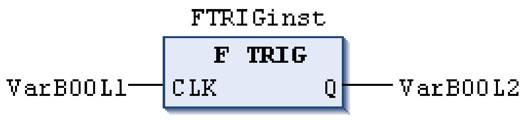
Q:BOOL; if CLK detects a falling edge, the output is TRUE

When CLK changes from "TRUE" to "FALSE", the falling edge detector starts, the output Q changes from "TRUE" to "FALSE" and remains "FALSE" for one operation cycle of the PLC; if CLK continues to remain "TRUE" or "FALSE", the output Q remains "FALSE".

Declaration example:

```
FTRIGInst:F_TRIG;
```

Example in FBD:



Example in ST:

```
FTRIGInst(CLK:=VarBOOL1);
VarBOOL2:=FTRIGInst.Q;
```

6.6 Bit/Byte Functions

6.6.1 EXTRACT

The input variable X of this function is of DWORD type and N is of BYTE type. The output variable is of BOOL type, and the output is the value of the Nth bit of the input variable X, where N starts from the 0th bit.

Example in ST:

```
FLAG:=EXTRACT(X:=81,N:=4);
(*Result:TRUE,because81isbinary1010001,sothe4thbitis1*)
FLAG:=EXTRACT(X:=33,N:=0);
(*Result:TRUE,because33isbinary100001,sothebit'0'is1*)
```

6.6.2 PACK

PACK is used to pack 8 BOOL type input variables B0, B1, ..., B7 into 1 BYTE type data.

The function block UNPACK is closely related to this function.

6.6.3 PUTBIT

The input variables X, N, and B of this function are of DWORD type, BYTE type, and BOOL type respectively.

PUTBIT is used to set the Nth bit of X to the value B, where N starts from the 0th bit.

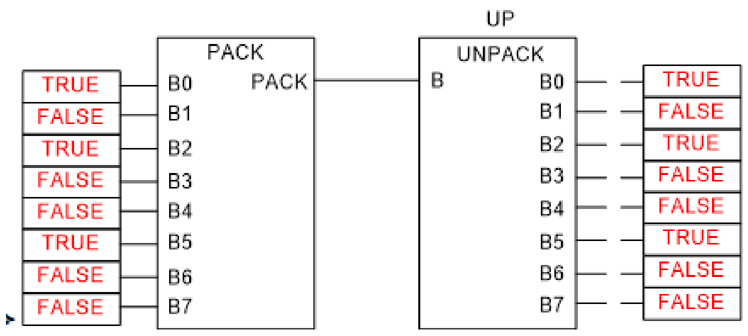
Example in ST:

```
var1:=38;(*binary100110*)
var2:=PUTBIT(A,4,TRUE);(*Result:54=2#110110*)
var3:=PUTBIT(A,1,FALSE);(*Result:36=2#100100*)
```

6.6.4 UNPACK

UNPACK is used to split the BYTE type input variable B into 8 BOOL type output variables B0, B1, ..., B7. It functions oppositely to PACK.

Example in FBD:

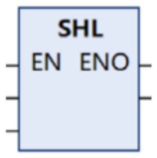


6.7 Bit Shift Instructions

6.7.1 Bitwise Left-shift (SHL)

Shift the input value bit by bit to the left. The bits shifted out on the left are not processed and the bits on the right are automatically filled with 0.

Example in FBD:

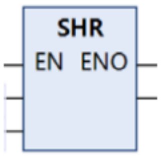


Note: The data can only be of Integer type. If it is of floating point type, an error will be reported.

6.7.2 Bitwise Right-shift (SHR)

Shift the input value bit by bit to the right. The bits shifted out on the right are not processed and the bits on the left are automatically filled with 0.

Example in FBD:

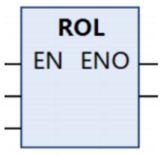


Note: The data can only be of Integer type. If it is of floating point type, an error will be reported.

6.7.3 Bitwise Left-rotation (ROL)

Rotate the input value bit by bit to the left, and the bits rotated out on the left are directly added to the least significant bit on the right.

Example in FBD:

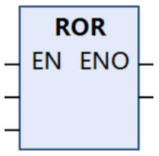


Note: The instruction supports the Integer data type.

6.7.4 Bitwise Right-rotation (ROR)

Rotate the input value bit by bit to the right, and the bits rotated out on the right are directly added to the most significant bit on the left.

Example in FBD:



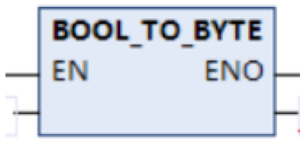
Note: The instruction supports the Integer data type.

6.8 Data Type Conversion Instructions

6.8.1 BOOL_TO_<TYPE>

Convert a Boolean type variable to a variable of any other type.

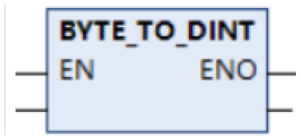
Example in FBD:



6.8.2 BYTE_TO_<TYPE>

Convert a Byte type variable to a variable of any other type.

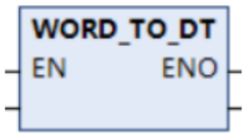
Example in FBD:



6.8.3 WORD_TO_<TYPE>

Convert a Word type variable to a variable of any other type.

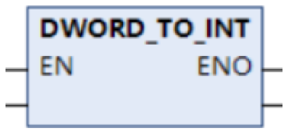
Example in FBD:



6.8.4 DWORD_TO_<TYPE>

Convert a Double-word type variable to a variable of any other type.

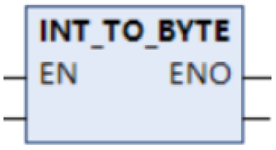
Example in FBD:



6.8.5 INT_TO_<TYPE>

Convert an Integer type variable to a variable of any other type.

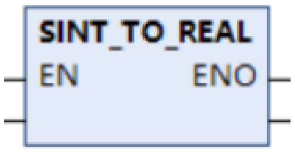
Example in FBD:



6.8.6 SINT_TO_<TYPE>

Convert a Short-integer type variable to a variable of any other type.

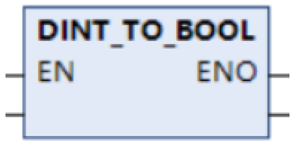
Example in FBD:



6.8.7 DINT_TO_<TYPE>

Convert a Long-integer type variable to a variable of any other type.

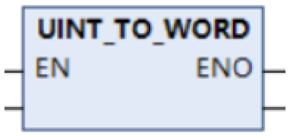
Example in FBD:



6.8.8 UDINT_TO_<TYPE>

Convert an unsigned Long-integer type variable to a variable of any other type.

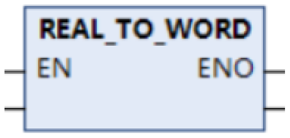
Example in FBD:



6.8.9 REAL_TO_<TYPE>

Convert a Real number type variable to a variable of any other type.

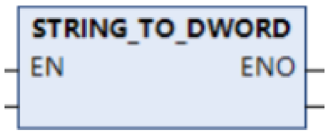
Example in FBD:



6.8.10 STRING_TO_<TYPE>

Convert a Character type variable to a variable of any other type.

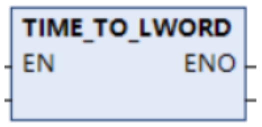
Example in FBD:



6.8.11 TIME_TO_<TYPE>

Convert a Clock type variable to a variable of any other type.

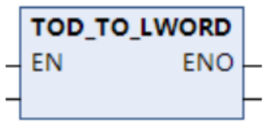
Example in FBD:



6.8.12 TOD_TO_<TYPE>

Convert a Time type variable to a variable of any other type.

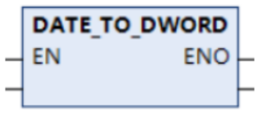
Example in FBD:



6.8.13 DATE_TO_<TYPE>

Convert a Date type variable to a variable of any other type.

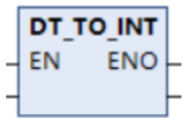
Example in FBD:



6.8.14 DT_TO_<TYPE>

Convert a DateTime type variable to a variable of any other type.

Example in FBD:

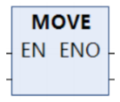


6.9 Data Processing Instructions

6.9.1 MOVE

This operator is used to assign the value of one variable to another variable of the same type.

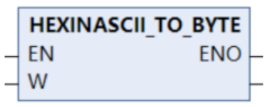
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
MOVE	Assignment	FC		a2:=MOVE(a1);

6.9.2 HEXinASCII_TO_BYTE

When this instruction is triggered, the HEXinASCII data in the source data is converted into Byte type data.

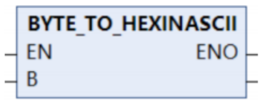
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
HEXinASCII_TO_BYTE	HEXinASCII to BYTE	FC		HEXinASCII_TO_BYTE(W:=);

6.9.3 BYTE_TO_HEXinASCII

When this instruction is triggered, the Byte type data in the source data is converted into HEXinASCII type data.

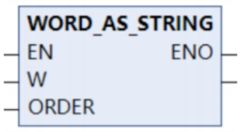
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
BYTE_TO_HEXinASCII	BYTE to HEXinASCII	FC		BYTE_TO_HEXinASCII(B:=);

6.9.4 WORD_AS_STRING

When this instruction is triggered, the WORD type data in the source data is converted into STRING type data.

Instruction format:

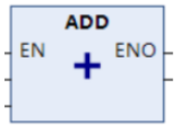
Instruction	Name	FB/FC	LD Representation	ST Representation
WORD_AS_STRING	WORD to STRING	FC		WORD_AS_STRING (W:=,ORDER:=);

6.10 Arithmetic Instructions

6.10.1 ADD

Add the two inputs on the left side and output the result on the right side.

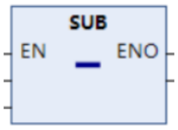
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
ADD	Addition	FC		a1:=a2+a3;

6.10.2 SUB

Subtract one input from the other on the left side and output the result on the right side.

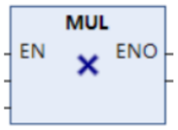
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
SUB	Subtraction	FC		a1:=a2-a3;

6.10.3 MUL

Multiply the two inputs on the left side and output the result on the right side.

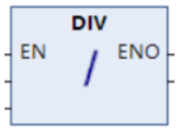
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
MUL	Multiplication	FC		a1:=a2*a3;

6.10.4 DIV

Divide one input by the other on the left side and output the quotient on the right side.

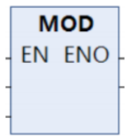
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
DIV	Division	FC		a1:=a2/a3;

6.10.5 MOD

Perform the modulo division of one input by the other on the left side and output the non-negative integer remainder on the right side.

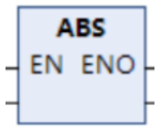
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
MOV	Modulo division	FC		a1:=a2 MOD a3;

6.10.6 ABS

Take the absolute value of the input data and assign it to the output variable.

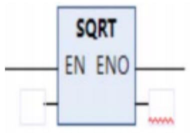
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
ABS	Absolute value	FC		q:=ABS();

6.10.7 SQRT

Compute the square root of the input value and output the result.

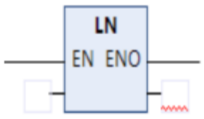
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
SQRT	Square root	FC		q:=SQRT();

6.10.8 LN

Compute the natural logarithm of the input value and output the result.

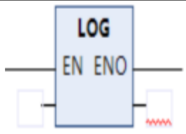
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
LN	Natural logarithm e	FC		q:=LN();

6.10.9 LOG

Compute the logarithm of the input value in base 10 and output the result.

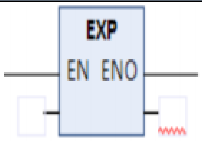
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
LOG	Logarithm in base 10	FC		q:=LOG();

6.10.10 EXP

Compute the exponential function of the input value and output the result.

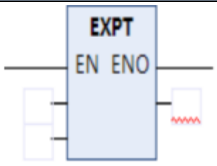
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
EXP	Exponential function	FC		q:=EXP();

6.10.11 EXPT

Raise the input variable 1 to the power of the input variable 2 and output the result.

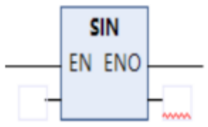
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
EXPT	Exponentiation	FC		q:=EXPT();

6.10.12 SIN

Compute the sine of the input value and output the result.

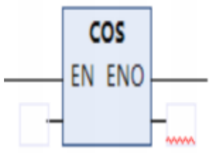
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
SIN	Sine	FC		q:=SIN();

6.10.13 COS

Compute the cosine of the input value and output the result.

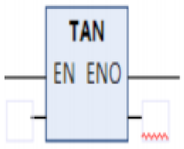
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
COS	Cosine	FC		q:=COS();

6.10.14 TAN

Compute the tangent of the input value and output the result.

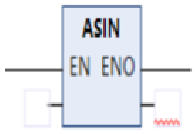
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
TAN	Tangent	FC		q:=TAN();

6.10.15 ASIN

Compute the arc sine of the input value and output the result.

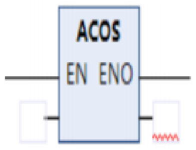
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
ASIN	Arc sine	FC		q:=ASIN();

6.10.16 ACOS

Compute the arc cosine of the input value and output the result in radians.

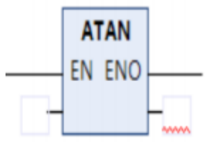
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
ACOS	Arc cosine	FC		q:=ACOS();

6.10.17 ATAN

Compute the arc tangent of the input value and output the result in radians.

Instruction format:

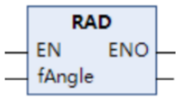
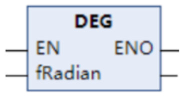
Instruction	Name	FB/FC	LD Representation	ST Representation
ATAN	Arc tangent	FC		q:=ATAN();

6.10.18 RAD/DEG

RAD: Convert floating point degrees into radians. The calculation formula is [Radians = Degrees × л/180].

DEG: Convert floating point radians into degrees. The calculation formula is [Degrees = Radians × л/180].

Instruction format:

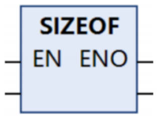
Instruction	Name	FB/FC	LD Representation	ST Representation
RAD	Degrees to radians	FC		q:=RAD();
DEG	Radians to degrees	FC		q:=DEG();

6.10.19 SIZEOF

The input value is used to define the number of bytes required by a “variable”. The SIZEOF operator always returns an unsigned value. The type of the returned variable adapts to

the detected size of the “variable”.

Instruction format:

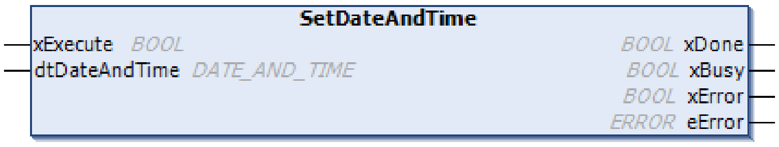
Instruction	Name	FB/FC	LD Representation	ST Representation
SIZEOF	Size of data type	FC		SIZEOF();

6.11 Date and Time Instructions

6.11.1 SetDateAndTime

Set the time zone, date, and time of the current system.

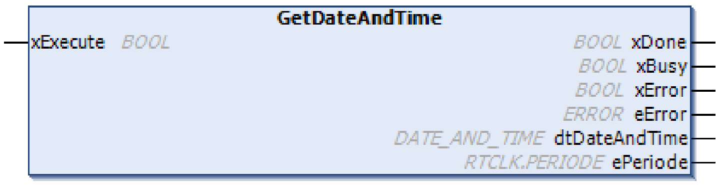
Instruction format:



6.11.2 GetDateAndTime

Get the time zone, date, and time of the current system.

Instruction format:

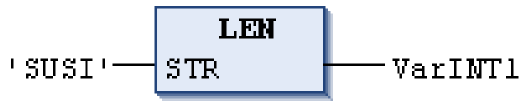


6.12 String Function Instructions

6.12.1 LEN

This function is used to get the length of a character string. The input variable STR is of the STRING type, and the return value is of the INT type.

Example in FBD:



Example in ST:

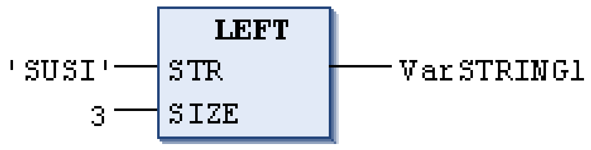
```
VarINT1:=LEN('SUSI');
```

6.12.2 LEFT

This function is used to get certain characters from the left of a source character string. The input variable STR is of the STRING type, the input variable SIZE is of the INT type, and the return value is of the STRING type.

LEFT (STR, SIZE) is used to get the characters with the length specified by SIZE, starting from the left of the character string STR.

Example in FBD:



Example in ST:

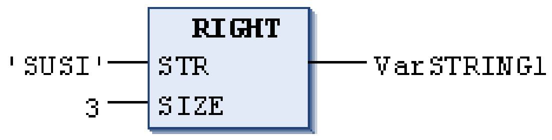
```
VarSTRING1:=LEFT('SUSI',3);
```

6.12.3 RIGHT

This function is used to obtain certain characters from the right of a source character string. The input variable STR is of the STRING type, the input variable SIZE is of the INT type, and the return value is of the STRING type.

RIGHT (STR, SIZE) is used to get the characters with the length specified by SIZE, starting from the right of the character string STR.

Example in FBD:



Example in ST:

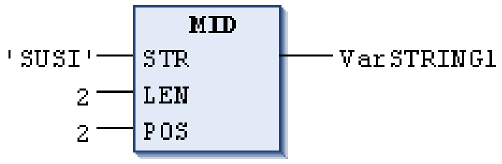
```
VarSTRING1 := RIGHT ('SUSI',3);
```

6.12.4 MID

This function is used to get certain characters from a source character string. The input variable STR is of the STRING type, the input variables LEN and POS are of the INT type, and the return value is of the STRING type.

MID (STR, LEN, POS) is used to obtain the characters with the length specified by LEN, starting from the character with the position specified by POS of the character string STR.

Example in FBD:



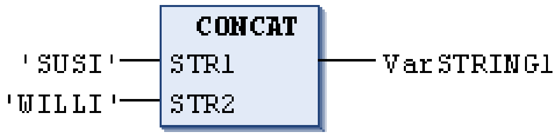
Example in ST:

VarSTRING1:=MID('SUSI',2,2);

6.12.5 CONCAT

This function is used to concatenate two character strings. The input variables STR1 and STR2, and the return value are of the STRING type.

Example in FBD:



Example in ST:

VarSTRING1:=CONCAT('SUSI','WILLI');

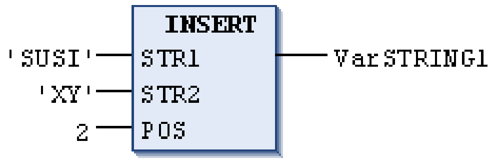
6.12.6 INSERT

This function is used to insert another character string at a specified position into a source character string.

The input variables STR1 and STR2 are of the STRING type, the input variable POS is of the INT type, and the return value is of the STRING type.

INSERT(STR1,STR2,POS) is used to insert the character string STR2 next to the position specified by POS into the character string STR1.

Example in FBD:



Example in ST:

VarSTRING1:=INSERT('SUSI','XY',2);

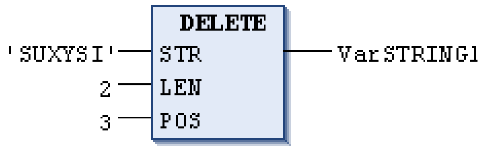
6.12.7 DELETE

This function is used to delete specified characters from a specified position of a source character string.

The input variable STR is of the STRING type, the input variables LEN and POS are of the INT type, and the return value is of the STRING type.

DELETE (STR, L, POS) is used to delete certain characters from the character string STR, where L specifies the length of characters to be deleted and POS specifies the character deletion start position.

Example in FBD:



Example in ST:

Var1:=DELETE('SUXYSI',2,3);

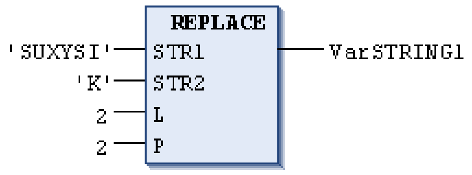
6.12.8 REPLACE

This function is used to replace certain characters at a specified position of a source character string with another given character string.

The input variables STR1 and STR2 are of the STRING type, the input variables L and P are of the INT type, and the return value is of the STRING type.

REPLACE(STR1,STR2,L,P) is used to replace certain characters with the character string STR2 for the character string STR1, where L specifies the length of characters to be replaced and P specifies the character replacement start position.

Example in FBD:



Example in ST:

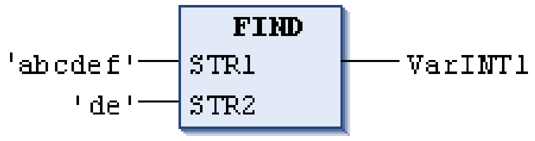
```
VarSTRING1:=REPLACE(’ SUXYSI’ ,’ K’ ,2,2);
```

6.12.9 FIND

This function is used to search a character string for certain characters. The input variables STR1 and STR2 are of the STRING type, and the return value is of the INT type.

FIND(STR1,STR2)) is used to find where STR2 occurs in STR1 for the first time. If STR2 is not found in STR1, the message is displayed: "OUT:=0".

Example in FBD:



Example in ST:

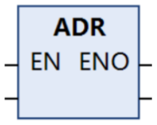
```
arINT1:=FIND('abcdef','de');
```

6.13 Address Operation Instructions

6.13.1 ADR/^

ADR: Get the memory address of the input variable and assign the result to the output variable. This operator is an extension of the IEC61131-3 standard^: Get the address content of the input variable and assign the result to the output variable.

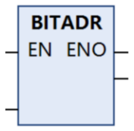
Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
ADR	Get address	FC		ADR();
^	Get address content	FC	∧	^

6.13.2 BITADR

Get the memory address of a BOOL type variable and assign the result to the output variable.

Instruction format:

Instruction	Name	FB/FC	LD Representation	ST Representation
BITADR	Bit address	FC		BITADR();

6.14 File Operation Instructions

6.14.1 Overview

This library is mainly used for importing and exporting files/folders between an SD card and the local AX7X, as well as deleting and writing files/folders.

The default SD card path is /home/root/temp/. If this path cannot be used, please try another version of the SD card path: /home/root/sdcard/. The default local PLC path is /home/CODESYS/PlcLogic/.

Note: Since the PLC memory is small, it is recommended to use it reasonably.

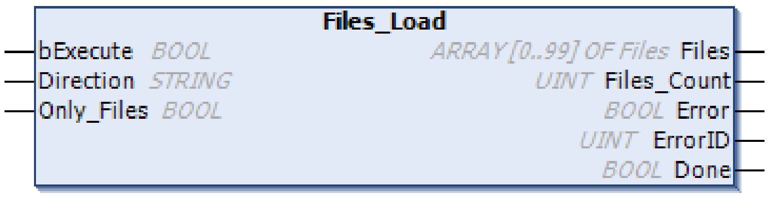
6.14.2 Input and Output

Description of file structure:

In order to facilitate the storage and operation of file information, a custom file structure is created. This structure differs from the FILE_DIR_ENTRY structure in that a Direction (file path) parameter is added.

Name	Data Type	Comment
Name	STRING	File name (including the file extension)
Direction	STRING	File path
isDirectory	BOOL	Folder flag, TRUE: folder, FALSE: file
Size	CAA.SIZE	File memory size, unit: Byte
LastModification	DATE_AND_TIME	Last modification time of the file

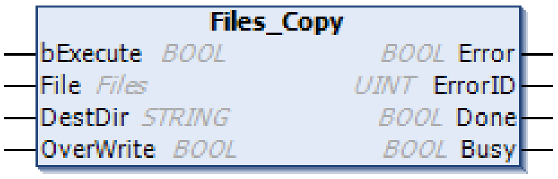
6.14.3 Load Files (files_load)



Category	Name	Data Type	Initial Value	Comment
Input	bExecute	BOOL	-	Rising edge trigger
	Direction	STRING	'/home/CODESYS /PlcLogic/_cnc/'	Target path
	Only_Files	BOOL	-	TRUE: Only load files FALSE: Load both files and folders
Output	Files	ARRAY[0…20] OF Files	-	List of loaded files, maximum 20
	Files_Count	UINT	-	Number of loaded files/folders
	Error	BOOL	-	Alarm flag

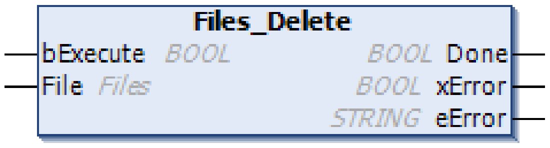
Category	Name	Data Type	Initial Value	Comment
	ErrorID	UINT	-	Alarm code
	Done	BOOL	-	Load completed flag

6.14.4 Copy Files (Files_Copy)



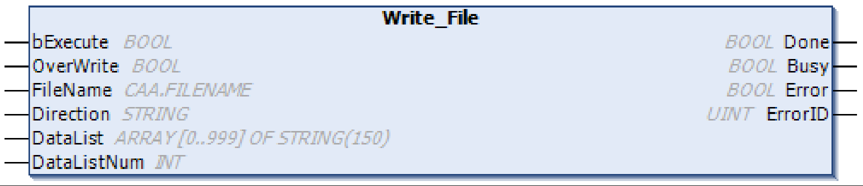
Category	Name	Data Type	Initial value	Comment
Input	bExecute	BOOL	-	Rising edge trigger
	File	Files	-	Files to be copied
	DestDir	STRING	'/home/CODESYS /PlcLogic/_cnc/'	Default path: local PLC: _cnc folder, SD card: /home/root/temp/
	OverWrite	BOOL	-	Overwrite existing files TRUE: Overwrite, FALSE: Do not overwrite
Output	Error	BOOL	-	Alarm flag
	ErrorID	UINT	-	Alarm code
	Done	BOOL	-	Copy completed flag
	Busy	BOOL	-	Copying

6.14.5 Delete Files (Delete_File)



Category	Name	Data Type	Initial value	Comment
Input	bExecute	BOOL	-	Rising edge trigger
	File	Files	-	Files to be deleted
Output	Done	BOOL	-	Delete completed flag
	xError	BOOL	-	Alarm flag
	eError	STRING	-	Alarm code

6.14.6 Write Files (Write_File)



Category	Name	Data Type	Initial value	Comment
Input	bExecute	BOOL	-	Rising edge trigger
	OverWrite	FALSE	FALSE	TRUE: Overwriting FALSE: The file has not been written
Input	FileName	CAA.FILENAME	‘xx.cnc’	Name of the file to be written (including the file extension)

Category	Name	Data Type	Initial value	Comment
	Direction	STRING	'/home/CODESYS/PlcLogic/_cnc/'	File path
	DataList	ARRAY[0..999] OF STRING(150)	-	List of data to be written row by row, with a maximum of 1000 data entries stored at one time
	DataListNum	INT	-	Number of entries in the data list
Output	Done	BOOL	-	Write completed flag
	Busy	BOOL	-	Writing flag
	Error	BOOL	-	Alarm flag
	ErrorID	UINT	-	Alarm code

Use of the function block

It is implemented as follows in the ST language:

```
VAR
    Load:files_load;
    Load_Execite:BOOL;
    Load_Direction:STRING:=' /home/CODESYS/PlcLogic/_cnc/' ;
    Load_Only_File:BOOL;
    files:ARRAY[0..19] OF files;
    files_Cunt:UINT;
    Load_Error:BOOL;
    Load_ErrorID:UINT;
    Load_Done:BOOL;

    Copy:Files_Copy;
    Copy_Execute:BOOL;
    File_Index:UINT:=0;
    Copy_File:Files;
    Copy_DestDir:STRING:=' /home/CODESYS/PlcLogic/_cnc/112/' ;
    Copy_OverWrite:BOOL;
    Copy_Error:BOOL;
    Copy_ErrorID:UINT;
    Copy_Done:BOOL;
    Copy_Busy:BOOL;


    Delete:Files_Delete;
    Delete_Execute:BOOL;
    Delete_File:Files;
    Delete_Done:BOOL;
    Delete_Error:BOOL;
    Delete_ErrorID:UINT;

    Write:Write_File;
    Write_Execute:BOOL;
    Write_OverWrite:BOOL;
```

```
          Write_FileName:STRING;
          Write_Direction:STRING:=’  /home/CODESYS/PlcLogic/_cnc/’  ;
          Write_DataList:ARRAY [0..999] OF STRING(150);
          Write_DataListNum:UINT;
          Write_Done:BOOL;
          Write_Busy:BOOL;
          Write_Error:BOOL;
          Write_ErrorID:UINT;
END_VAR

Load(
          bExecute:= Load_Execite,
          Direction:= Load_Direction,
          Only_Files:= Load_Only_File,
          Files=> files,
          Files_Count=> files_Cunt,
          Error=> Load_Error ,
          ErrorID=>Load_ErrorID ,
          Done=> Load_Done);
IF Load_Done THEN
          Load_Execite:=FALSE;
END_IF

Copy_File:=files[File_Index];
Copy(
          bExecute:= Copy_Execute,
          File:= Copy_File,
          DestDir:= Copy_DestDir,
          OverWrite:= Copy_OverWrite,
          Error=> Copy_Error,
          ErrorID=> Copy_ErrorID,
          Done=> Copy_Done,
          Busy=> Copy_Busy);
IF Copy_Done THEN
          Copy_Execute:=FALSE;
END_IF

//Delete_File.Direction:=Copy_DestDir;
Delete(
          bExecute:= Delete_Execute,
          File:= Delete_File,
          Done=> Delete_Done,
          Error=>Delete_Error ,
		ErrorID=> Delete_ErrorID);
	IF Delete_Done THEN
		Delete_Execute:=FALSE;
	END_IF

	Write_DataListNum:=3;
	Write_DataList[0]:= ‘00’ ;
	Write_DataList[1]:= ‘111’ ;
	Write_DataList[2]:= ‘2222’ ;
	Write_FileName:=files[File_Index].Name;
	Write(
		bExecute:= Write_Execute,
		OverWrite:= Write_OverWrite,
		FileName:= Write_FileName,
		Direction:= Write_Direction,
		DataList:= Write_DataList,
		DataListNum:=Write_DataListNum,
		Done=> Write_Done,
		Busy=> Write_Busy,
		Error=> Write_Error,
		ErrorID=> Write_ErrorID);
	IF Write_Done THEN
		Write_Execute:=FALSE;
	END_IF
```

Error ID	Error Type	Solution
16#0000	No alarm	-
16#0001	The target direction path does not exist	Re-confirm the target path
16#0002	Failed to get the file name	Confirm the name of the file to be loaded
16#0003	Timeout when closing the target path	Confirm that other functions do not operate on the path when closing it
16#0004	Failed to copy the file/folder	Confirm whether the file and target path exist
16#0005	Failed to delete the file/folder	Confirm whether the file and target path exist
16#0006	Failed to open the file	Confirm that the file path and name are correct
16#0007	Failed to write the file	Confirm the amount of data to be written to avoid a timeout
16#0008	Failed to close the file	Confirm that no other operations are being performed on the file when the file is closed.
16#0009	The copied file exceeds 100 kB	Determine if the size of the file to be copied exceeds 100 kB

6.15 Regulators

6.15.1 PD

This function block is used to regulate proportions and differentials.

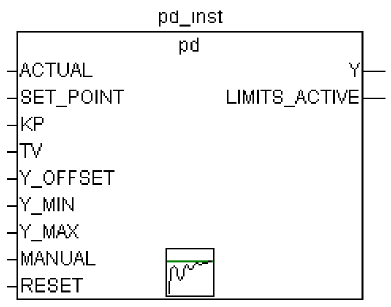
Input variables:

Variable	Data Type	Description
ACTUAL	REAL	Actual value of the control variable.
SET_POINT	REAL	Description value and instruction value.
KP	REAL	Proportional coefficient used to represent the proportional gain of the P-part.
TV	REAL	Differential time used to represent the time calculated in seconds of the D-part. For example, "0.5" indicates 500 s.
Y_MANUAL	REAL	Used to define the output value Y when MANUAL=TRUE.
Y_OFFSET	REAL	Offset value of the operation value Y.
Y_MIN,Y_MAX	REAL	Lower limit and upper limit of the operation value Y. If Y reaches a limit value, LIMITS_ACTIVE is set to TRUE and Y is kept within the formulated range. This function block works only when Y_MIN < Y_MAX.
MANUAL	BOOL	If it is TRUE, manual operating is activated, and the output value is defined through Y_MANUAL.
RESET	BOOL	Setting the value to TRUE will reset the controller. During re-initialization, Y is equal to Y_OFFSET.

Output variables:

Variable	Data Type	Description
Y	REAL	Operation value, defined by the function block (see the following).
LIMITS_ACTIVE	BOOL	When the value is TRUE, Y reaches the given limit value (Y_MIN or Y_MAX).

Example in FBD:



Y_OFFSET, Y_MIN, and Y_MAX are used to convert numbers in specified ranges.

MANUAL can be used to enable or disable manual operating. RESET is used to reset the controller.

During normal operating (MANUAL = RESET = LIMITS_ACTIVE = FALSE), the controller calculates the deviation value SET_POINT-ACTUAL and stores the time-related derivatives de/dt as internal variables.

The output value Y can be obtained by using the following:

$$\mathrm{Y} = \mathrm{KP} \cdot \left(\Delta + \mathrm{TV}\frac{\delta\Delta}{\delta t}\right) + \mathrm{Y\_OFFSET}$$

Where Δ=SET_POINT-ACTUAL

Therefore, except for the P-part and the present deviation (D-part) of the controller, all the others have an impact on the calculation output.

In addition, Y is restricted to the range defined by Y_MIN and Y_MAX. If Y reaches a limit value, LIMITS_ACTIVE is set to TRUE. If there is no calculation limit value, Y_MIN and Y_MAX must be set to 0.

Once MANUAL=TRUE, Y is written into Y_MANUAL.

A P adjustment can be achieved by setting TV=0.

6.15.2 PID

This function block is used to regulate proportions, integrals, and differentials.

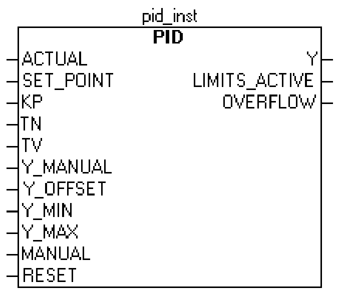
Input variables:

Variable	Data Type	Description
ACTUAL	REAL	Actual value of the control variable
SET_POINT	REAL	Expected value, instruction variable
KP	REAL	Proportional coefficient. The value cannot be 0 for the unity gain in the P-part; otherwise, the function block does not perform any calculations.
TN	REAL	Reset time. The unit gain in the i part is fixed to seconds. For example, "0.5" is 500 milliseconds, the value must be greater than 0; otherwise, the function block does not perform any calculations. A smaller TN value obtains a greater integral part, including the variable value. A greater TN value obtains a smaller integral part
TV	REAL	When the differential functions, the unit gain in the D-part is fixed to seconds. For example, "0.5" is 500 milliseconds
Y_MANUAL	REAL	The output value is Y when MANUAL = TRUE
Y_OFFSET	REAL	Offset operation variable Y
Y_MIN Y_MAX	REAL	A smaller resp value indicates a higher upper limit of the operation variable Y. If Y exceeds a limit value, LIMITS_ACTIVE is set to TRUE and Y is kept within the formulated range. Only when Y_MIN < Y_MAX, the control takes effect.
MANUAL	BOOL	If it is TRUE, manual operating is activated, and the operation variable is defined through Y_MANUAL.
RESET	BOOL	During initialization in which Y is equal to Y_OFFSET, setting the value to TRUE will reset the controller.

Output variables:

Variable	Data Type	Description
Y	REAL	Operation variable value, defined by the function block (see the following).
LIMITS_ACTIVE	BOOL	The value TRUE indicates that Y is out of the range defined by Y_MIN and Y_MAX.
OVERFLOW	BOOL	The value TRUE indicates overflow (see the following).

Example in FBD:



Y_OFFSET, Y_MIN, and Y_MAX are used to convert numbers in specified ranges.

MANUAL can be used to enable or disable manual operating. RESET is used to reset the controller.

During normal operating (MANUAL = RESET = LIMITS_ACTIVE = FALSE), the controller calculates the deviation value SET_POINT-ACTUAL and stores the time-related derivatives de/dt as internal variables.

The output value Y can be obtained by using the following:

$$\mathrm{Y} = \mathrm{KP} \cdot \left(\Delta + \frac{1}{TN}\int e\mathrm{dt} + \mathrm{TV}\frac{\delta\Delta}{\delta t}\right) + \mathrm{Y\_OFFSET}$$

Where Δ=SET_POINT-ACTUAL

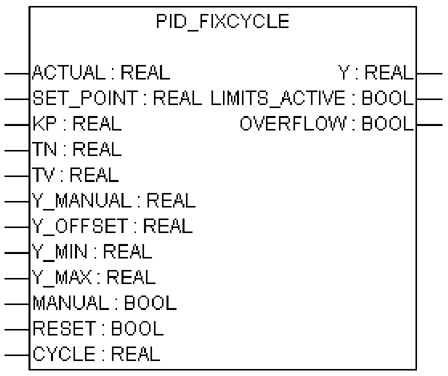
Therefore, except for the P-part and the present deviation (D-part) of the controller, all the others have an impact on the calculation output.

The PID controller can be easily converted into a PI controller by setting TV=0.

Incorrect controller parameter settings may cause overflow if the incorrect integral part becomes larger. Therefore, for safety purpose, the output can call OVERFLOW, in which the value is TRUE. This happens only when the control system is unstable due to incorrect parameter settings. At the same time, the controller is suspended and can be reactivated only through re-initialization.

6.15.3 PID_FIXCYCLE

Example in FBD:



The function of this function module is the same as that of the PID controller. The difference is that its cycle time is set by CYCLE (seconds) instead of being automatically measured by an internal function.

6.16 BCD Conversion Instructions

6.16.1 BCD_TO_INT

This function is used to convert one byte in BCD format into an INT value. The input variable is of BYTE type and the output variable is of INT type.

When the byte to be converted is not in BCD format, the output is -1.

Example in ST:

```
i:=BCD_TO_INT(73);    (* Result is 49 *)
k:=BCD_TO_INT(151); (* Result is 97 *)
l:=BCD_TO_INT(15);    (* Output -1, because it is not in BCD format *)
```

6.16.2 INT_TO_BCD

This function is used to convert an INT value into a byte in BCD format. The input variable is of INT type and the output variable is of BYTE type.

When the INT value cannot be converted to a byte in BCD format, the output is 255.

Example in ST:

```
i:=INT_TO_BCD(49);    (* Result is 73 *)
k:=BCD_TO_INT(97);    (* Result is 151 *)
l:=BCD_TO_INT(100); (* Error! Output: 255 *)
```

6.17 System Instructions

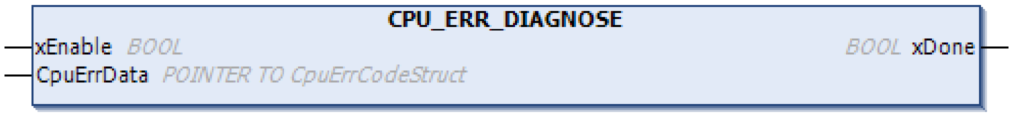
6.17.1 PLC Fault Diagnosis Instructions

These fault diagnosis instructions are applicable to TM and TP series PLCs. For error IDs, please refer to section 10.2 PLC Error Code Table (for TM and TP series PLCs) to look for the error description.

6.17.1.1 CPU_ERR_DIAGNOSE

This function block is used to read/write CPU fault information.

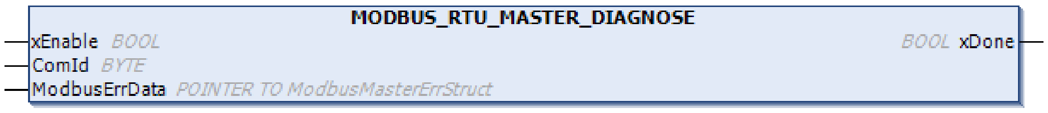
Example in FBD:



6.17.1.2 MODBUS_RTU_MASTER_DIAGNOSE

This function block is used to read/write Modbus_RTU_Master fault information.

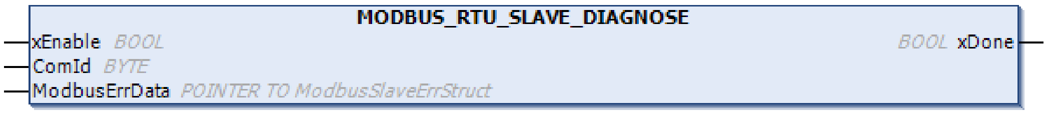
Example in FBD:


MODBUS_RTU_MASTER_DIAGNOSE
xEnable BOOL
ComId BYTE
ModbusErrData POINTER TO ModbusMasterErrStruct
BOOL xDone

6.17.1.3 MODBUS_RTU_SLAVE_DIAGNOSE

This function block is used to read/write Modbus_RTU_Slave fault information.

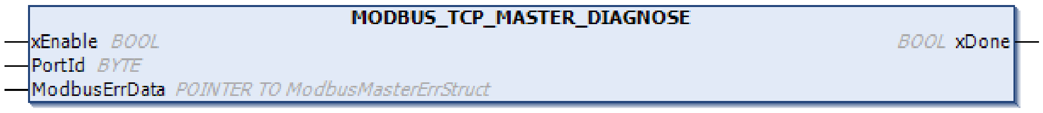
Example in FBD:


MODBUS_RTU_SLAVE_DIAGNOSE
xEnable BOOL
ComId BYTE
ModbusErrData POINTER TO ModbusSlaveErrStruct
BOOL xDone

6.17.1.4 MODBUS_TCP_MASTER_DIAGNOSE

This function block is used to read/write Modbus_TCP_Master fault information.

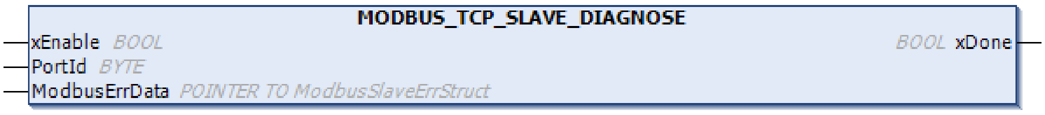
Example in FBD:


MODBUS_TCP_MASTER_DIAGNOSE
xEnable BOOL
PortId BYTE
ModbusErrData POINTER TO ModbusMasterErrStruct
BOOL xDone

6.17.1.5 MODBUS_TCP_SLAVE_DIAGNOSE

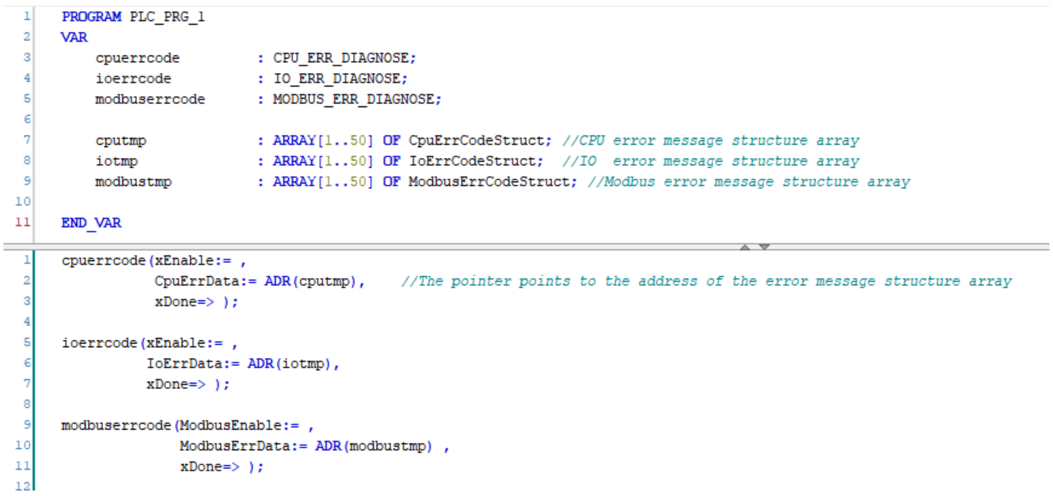
This function block is used to read/write Modbus_TCP_Slave fault information.

Example in FBD:


MODBUS_TCP_SLAVE_DIAGNOSE
xEnable BOOL
PortId BYTE
ModbusErrData POINTER TO ModbusSlaveErrStruct
BOOL xDone

Example in ST:

The use routine of the function block is as follows, and you can choose which function block to use as needed. When calling a function block instance, you should point the structure pointer to the corresponding error information structure array address, which stores the corresponding error diagnosis information; when multiple errors are diagnosed, the array can store multiple errors. The size of the array depends on your needs and can be defined by you, but it must be larger than the number of errors diagnosed.



```
PROGRAM PLC_PRG_1
VAR
    cpuerrcode          : CPU_ERR_DIAGNOSE;
    ioerrcode           : IO_ERR_DIAGNOSE;
    modbuserrcode       : MODBUS_ERR_DIAGNOSE;

    cputmp              : ARRAY[1..50] OF CpuErrCodeStruct; //CPU error message structure array
    iotmp               : ARRAY[1..50] OF IoErrCodeStruct;  //IO  error message structure array
    modbustmp           : ARRAY[1..50] OF ModbusErrCodeStruct; //Modbus error message structure array

END_VAR
```

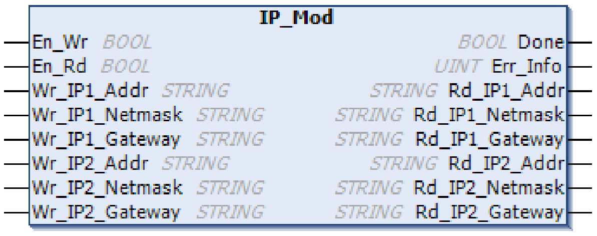
```
cpuerrcode(xEnable:= ,
           CpuErrData:= ADR(cputmp),    //The pointer points to the address of the error message structure array
           xDone=> );

ioerrcode(xEnable:= ,
          IoErrData:= ADR(iotmp),
          xDone=> );

modbuserrcode(ModbusEnable:= ,
              ModbusErrData:= ADR(modbustmp) ,
              xDone=> );
```

6.17.2 IP and Time Instructions of the TM Controller

6.17.2.1 IP_Mod (only applicable to the TM series PLC)

This function block is used to read/write network parameter information, including IP addresses, subnet masks, and gateway addresses.

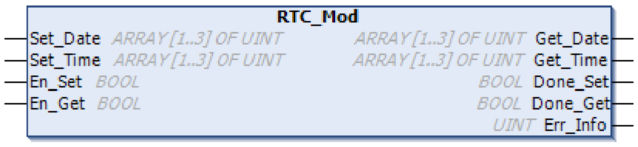
Example in FBD:



6.17.2.2 RTC_Mod (only applicable to the TM series PLC)

This function block is used to read/write the controller time.

Example in FBD:

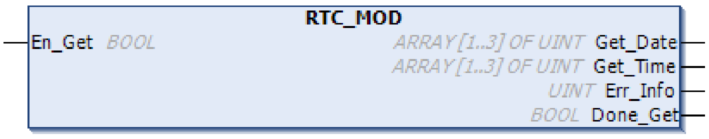

RTC_Mod
Set_Date ARRAY[1..3] OF UINT
Set_Time ARRAY[1..3] OF UINT
En_Set BOOL
En_Get BOOL
ARRAY[1..3] OF UINT Get_Date
ARRAY[1..3] OF UINT Get_Time
BOOL Done_Set
BOOL Done_Get
UINT Err_Info

6.17.3 IP and Time Instructions of the TP Controller

6.17.3.1 RTC_Mod (only applicable to the TP series PLC)

This function block is used to read the controller time.

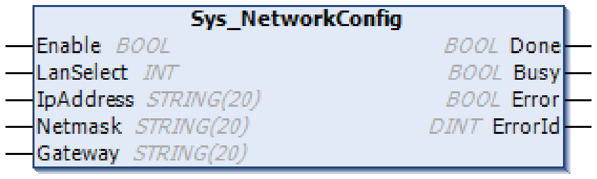
Example in FBD:



6.17.3.2 Sys_NetworkConfig (only applicable to the TP series PLC)

This function block is used to set network parameter information, including IP addresses, subnet masks, and gateway addresses.

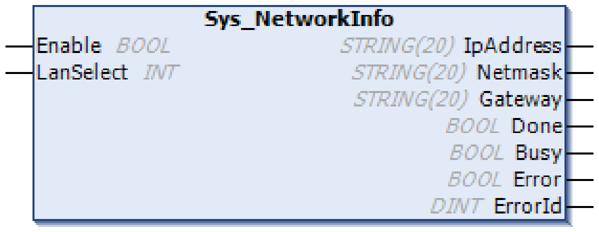
Example in FBD:



6.17.3.3 Sys_NetworkInfo (only applicable to the TP series PLC)

This function block is used to read network parameter information, including IP addresses, subnet masks, and gateway addresses.

Example in FBD:



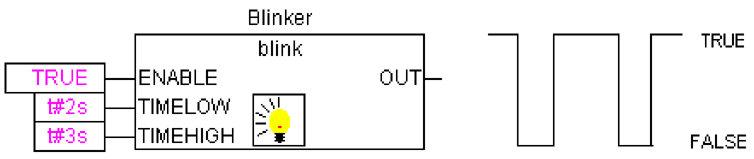
6.18 Signal Generator

6.18.1 BLINK

This function block is used to generate a pulse signal. The input variable ENABLE is of BOOL type, and TIMELOW and TIMEHIGH are of TIME type. The output variable OUT is of BOOL type.

If the value of ENABLE is TRUE, BLINK is enabled. OUT is TRUE during the time period set in TIMEHIGH, and OUT is FALSE during the time period set in TIMELOW.

Example in CFC:



6.18.2 FREQ_MEASURE

This function block is used to measure the (average) frequency value (Hz) of a Boolean input signal. You can specify the measurement cycle. One cycle refers to the interval between two rising edges of the signal.

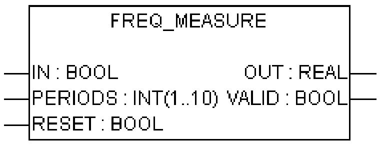
Input variables:

Variable	Data Type	Description
IN	BOOL	Input signal
PERIODS	INT	Cycle number, the time interval between two rising edges, through which the average frequency of the input signal is calculated, possible values: 1–10
RESET	BOOL	Reset all parameters to 0

Output variables:

Variable	Data Type	Description
OUT	REAL	Result frequency (Hz)
VALID	BOOL	FALSE until the first measurement cycle is completed, or if the cycle > 3*OUT (indicating an input error)

Example in FBD:



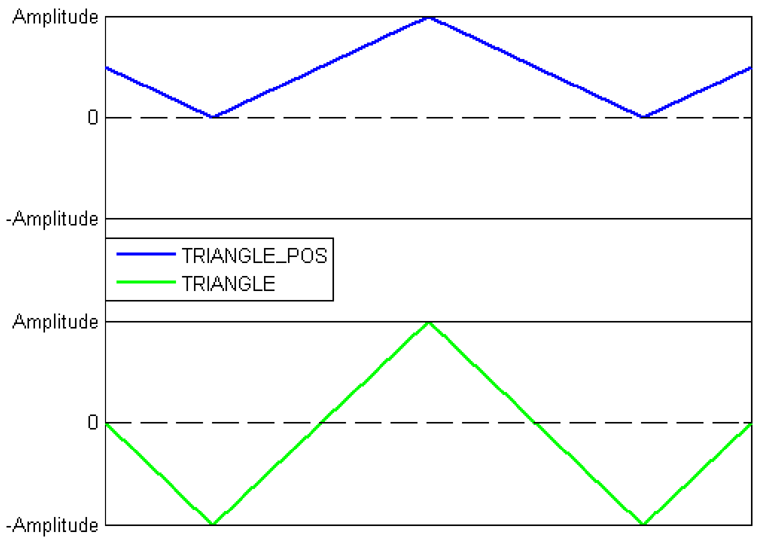
6.18.3 GEN

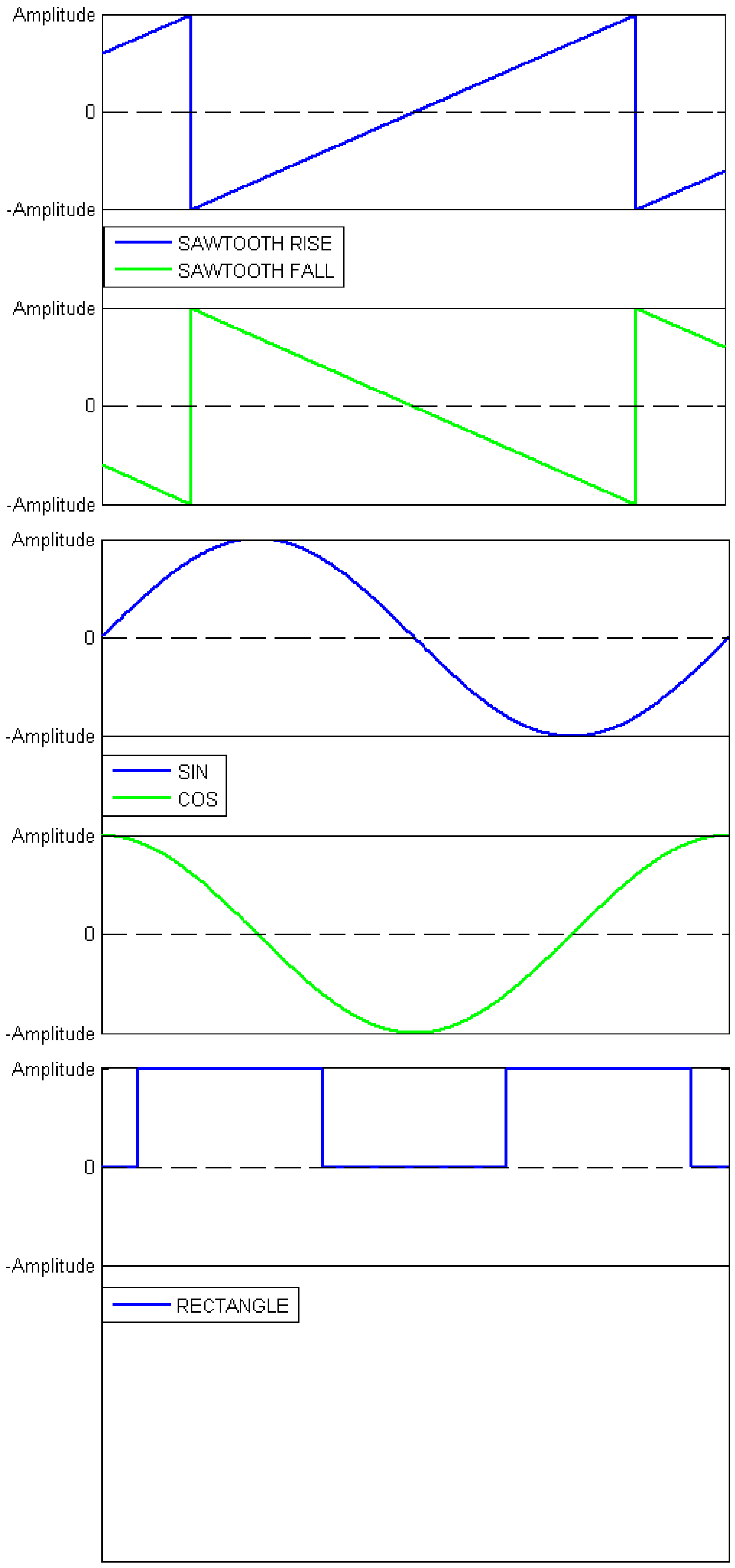
This function block is used to generate a standard oscillation cycle.

The input variable MODE can be predefined as the GEN_MODE type; BASE as the BOOL type; PERIOD as the TIME type; CYCLES and AMPLITUDE as the INT type; and RESET as the BOOL type.

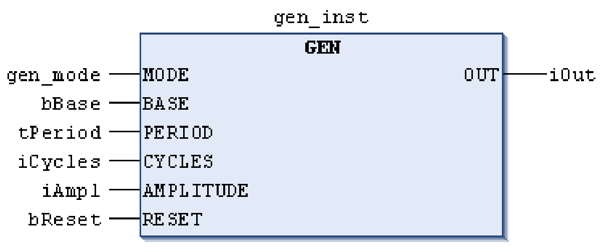
MODE is used to define the oscillation cycle mode generated. Here, the enumeration values TRIANGLE and TRIANGLE_POS are triangle waves, SAWTOOTH_RISE is an increasing sawtooth wave, SAWTOOTH_FALL is a decreasing sawtooth wave, RECTANGLE is a square wave, SINUS and COSINUS are sine and cosine waves respectively.

BASE is used to define whether the cycle period is defined using the set time (BASE=TRUE) or whether the cycle period is defined using a specific cycle value representing the number of times the function block is called (BASE=FALSE). PERIOD or CYCLES is used to define the corresponding cycle period. AMPLITUDE is used to define the amplitude produced. When RESET=TRUE, the signal generator is reset to 0.





Example in CFC:



6.19 Auxiliary Mathematical Function Blocks

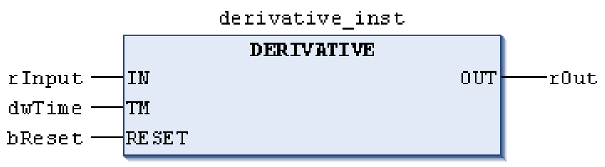
6.19.1 DERIVATIVE

This function block is used to determine local approximate derivatives.

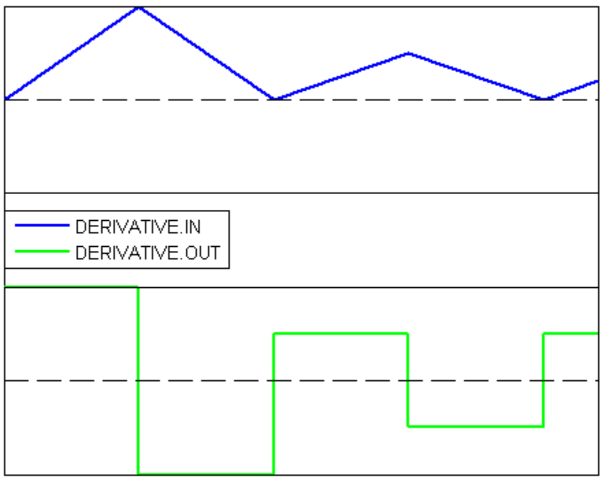
The input variable IN is of REAL type; TM is of DWORD type and represents time in milliseconds; RESET is of BOOL type, and when its value is TRUE, the function block is reset. The output variable OUT is of REAL type.

To achieve the most accurate result, DERIVATIVE approximates the last 4 values so that the inaccuracies introduced in the input parameters are minimized.

Example in FBD:



DERIVATIVE input and output:



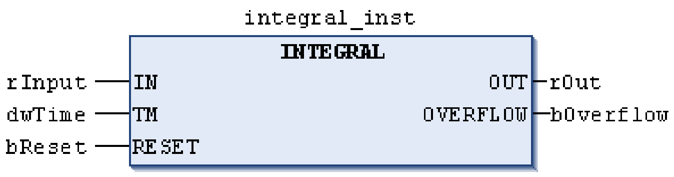
6.19.2 INTEGRAL

This function block is used to determine approximately the integral.

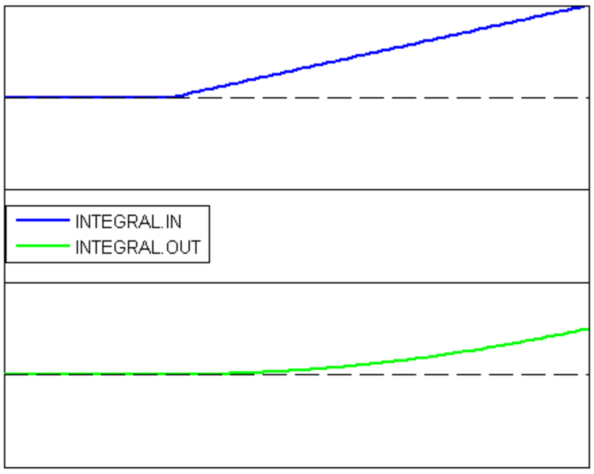
Similar to DERIVATIVE, the input variable IN is of REAL type; TM is of DWORD type and represents time in milliseconds; RESET is of BOOL type, and when the value is TRUE, the function block is reset. The output variable OUT is of REAL type.

The integral is approximated by two step functions and the average of the data is the approximate integral.

Example in FBD:



INTEGRAL input and output:

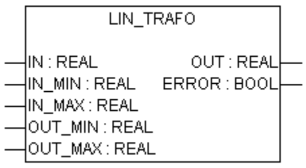


6.19.3 LIN_TRAFO

This function block transforms a real number within a range defined by an upper limit value and a lower limit value into a real number within a range defined by another upper limit value and another lower limit value.

The following expression is based on this transformation:

(IN-IN_MIN):(IN_MAX-IN)=(OUT-OUT_MIN):(OUT_MAX-OUT)



Input variables:

Variable	Data Type	Description
IN	REAL	Input variable
IN_MIN	REAL	Lower limit value of the variable range
IN_MAX	REAL	Upper limit value of the variable range
OUT_MIN	REAL	Lower limit value of the output range
OUT_MAX	REAL	Upper limit value of the output range

Output variables:

Variable	Data Type	Description
OUT	REAL	Output value
ERROR	BOOL	Error: TRUE if IN_MIN=IN_MAX, or if IN exceeds the specified input range

Application example:

A temperature sensor provides Volt-values (input IN). These are to be converted to temperature values in degree centigrade (output OUT). The input (Volt) values range is defined by the limits IN_MIN=0 and IN_MAX=10. The output (degree centigrade) value range is defined by the limits OUT_MIN=-20 and OUT_MAX=40. Thus for an input of 5 V, a temperature of 10℃ will be output.

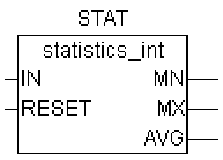
6.19.4 STATISTICS_INT

This function block is used to calculate some standard statistical values.

The input variable IN is of INT type. When the BOOL type input variable RESET is TRUE, all values are reinitialized.

The output variable MN is the minimum value of IN, MX is the maximum value of IN, and AVG is the average value. The three output variables are all of INT type.

Example in FBD:



6.19.5 STATISTICS_REAL

This function block is similar to STATISTICS_INT, except that the input variable IN and the output variables MN, MX, and AVG are all of REAL type.

6.19.6 VARIANCE

VARIANCE calculates the variance of an input value.

The input variable IN is of REAL type, RESET is of BOOL type, and the output variable OUT is of REAL type.

This function block is used to calculate the variance of an input value. When RESET=TRUE, VARIANCE will be reset.

The standard deviation can be easily obtained by taking the square root of the variance.

6.20 Operation Function Blocks

6.20.1 CHARCURVE

This function block is used to map an input value onto a characteristic curve.

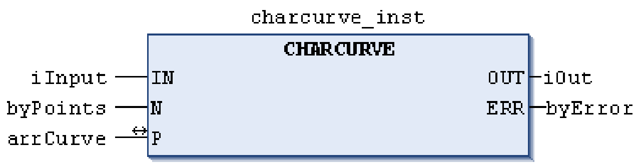
The input IN is of INT type and used to set the value to be processed; N is of BYTE type and used to set the number of points. P is a predefined POINT type based on two integer values (X and Y). The array P[0...10] is used to generate the characteristic curve.

The output variable OUT is of INT type and used to output processed data; ERR is of BYTE type and used to display errors.

The points P[0]...P[N-1] in the array must be stored according to the size of their X values; otherwise, ERR returns a value of 1. If the value of the input IN is not between P[0].X and P[N-1].X, ERR=2, and OUT is the corresponding limit value P[0].Y or P[N-1].Y.

If the value of N is outside the allowable value range of 2 to 11, then ERR=4.

Example in FBD:



Example in ST:

First, define the array P

VAR

...

CHARACTERISTIC_LINE:CHARCURVE;

KL:ARRAY[0..10]OFPOINT:=[(X:=0,Y:=0),(X:=250,Y:=50),

(X:=500,Y:=150),(X:=750,Y:=400),7((X:=1000,Y:=1000))];

COUNTER:INT;

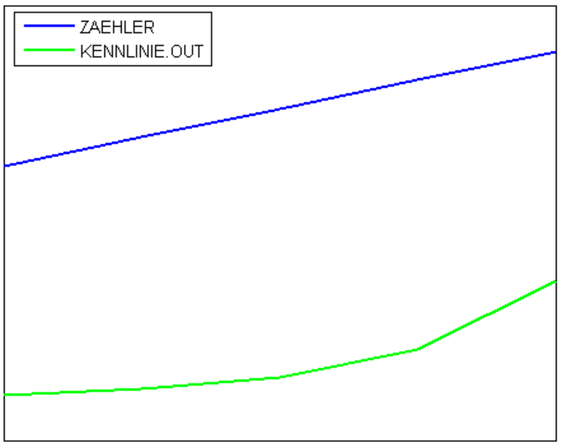
...

```
END_VAR
```

Then, define a CHARCURVE with an increasing value, for example:

```
COUNTER:=COUNTER+10;
CHARACTERISTIC_LINE(IN:=COUNTER,N:=5,P:=KL);
```

Illustration of the resulting curve:



6.20.2 RAMP_INT

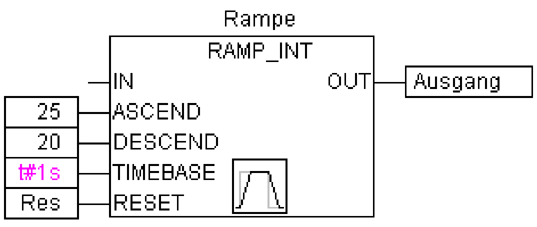
This function block is used to limit the rate of increase or decrease of an input value.

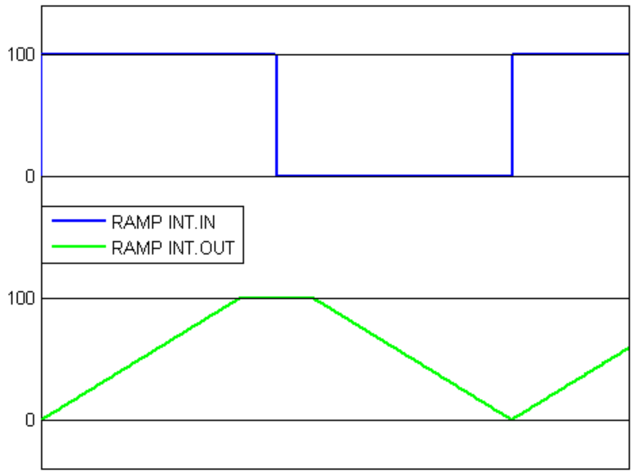
The input variables IN, ASCEND, and DESCEND are of INT type: IN is the input value, ASCEND and DESCEND are the maximum increment and decrement values within a given time. TIMEBASE is of TIME type and used to set a given time. When the value of RESET is TRUE, RAMP_INT will be reinitialized.

The output variable OUT is of INT type and contains the value with its rate of increase or decrease limited.

When the value of TIMEBASE is t#0s, the output OUT is independent of ASCEND and DESCEND and remains the same as IN.

Example in CFC:





6.20.3 RAMP_REAL

RAMP_REAL is similar to RAMP_INT in functionality, except that the inputs IN, ASCEND, and DESCEND and the output OUT of RAMP_REAL are of REAL type.

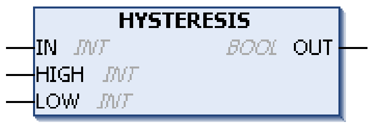
6.21 Analog Value Processing

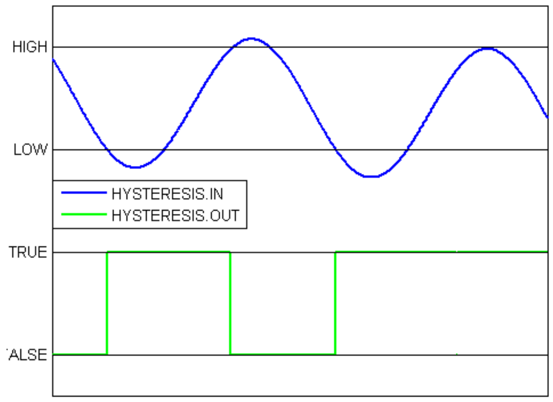
6.21.1 HYSTERESIS

The inputs of this function block include three INT variables: IN, HIGH, and LOW. The output OUT is of BOOL type.

If IN is below the lower limit value LOW, OUT is TRUE. If IN is above the upper limit value HIGH, OUT is FALSE.

Example in FBD:





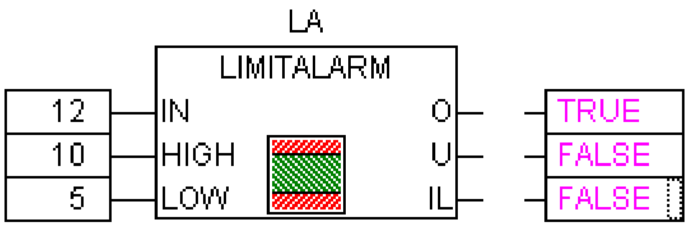
6.21.2 LIMITALARM

This function block is used to check whether the input value is within a certain range.

The input variables IN, HIGH, and LOW are all of INT type. The output variables O, U and IL are all of BOOL type.

If IN reaches the upper limit value HIGH, O will be set to TRUE, and when IN is below the lower limit value LOW, U will be set to TRUE. If IN is between LOW and HIGH, IL will be set to TRUE.

Example in FBD:



7 Motion Control Instructions

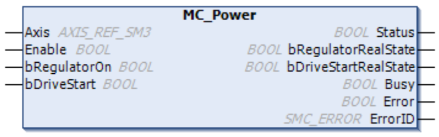
7.1 Single Axis Instructions

7.1.1 MC_Power

MC_Power: used to enable the servo drive.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_Power	Axis enabled	 MC_Power
		Axis AXIS_REF_SM3
		Enable BOOL
		bRegulatorOn BOOL
		bDriveStart BOOL
		BOOL Status
		BOOL bRegulatorRealState
		BOOL bDriveStartRealState
		BOOL Busy
		BOOL Error
		SMC_ERROR ErrorID
			MC_Power(
			Axis:=,
			Enable:=,
			bRegulatorOn:=,
			bDriveStart:=,
			Status=>,
			bRegulatorRealState=>,
			bDriveStartRealState=>,
			Busy=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge of the input will start the processing of the function block
bRegulatorOn	Execution condition	BOOL	TRUE, FALSE	FALSE	If it is TRUE, the axis is enabled
bDriveStart	Execution condition	BOOL	TRUE, FALSE	FALSE	High level input TRUE

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Status	Enabled	BOOL	TRUE, FALSE	FALSE	It becomes TRUE when the Enabled state is entered

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
bRegulatorRealState	Enabled	BOOL	TRUE, FALSE	FALSE	It becomes TRUE after bRegulatorOn is set to TRUE
bDriveStartRealState	Drive enabled	BOOL	TRUE, FALSE	FALSE	It becomes TRUE after bDriveStart is set to TRUE
Busy	Executing	BOOL	TRUE, FALSE	FALSE	It becomes TRUE After the instruction is received
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It becomes TRUE when an exception occurs.
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

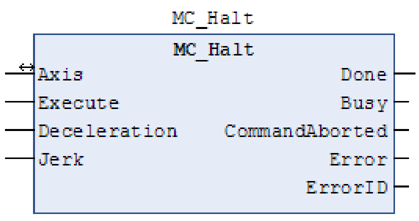
When Enable is set to TRUE, the axis specified by Axis enters the operational state. Setting the axis state to operational can implement axis control. When Enable is set to FALSE, the axis specified by Axis exits the operational state. After exiting the operational state, the axis does not receive any instruction, and therefore axis control cannot be implemented. In addition, the axis abnormally responds to motion instructions, but the axis can execute the MC_Power and MC_Reset instructions.

7.1.2 MC_Halt

MC_Halt: used to stop the motion of a specified axis.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_Halt	Instruction to stop an axis normally	 MC_Halt
		MC_Halt
		Axis
		Execute
		Deceleration
		Jerk
		Done
		Busy
		CommandAborted
		Error
		ErrorID
			MC_Halt(
			Axis:=,
			Execute:=,
			Deceleration:=,
			Jerk:=,
			Done=>,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge of the input will start the processing of the function block
Deceleration	Deceleration	LREAL	Positive or 0	0	Function block deceleration speed (μ/S2)

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Jerk	Execution condition	LREAL	Positive or 0	0	Specified jerk [instruction unit/S3]

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Instruction execution completed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE after the axis instruction is executed completely
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
CommandAborted	Instruction aborted	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is aborted
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

Starting this function block can stop the motion of an axis, but the execution of this function block can be terminated when another motion axis instruction is started.

This function block can be executed only when the axis is in running state.

This function block is started at the rising edge of the input variable execution condition.

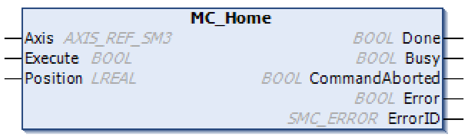
The axis state changes from DiscreteMotion during function block execution and to Standstill after the function block execution.

7.1.3 MC_Home

MC_Home: used to determine the home position of an axis.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_Home	Axis homing instruction	 MC_Home
		Axis AXIS_REF_SM3
		Execute BOOL
		Position LREAL
		BOOL Done
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		SMC_ERROR ErrorID
			MC_Home(
			Axis:=,
			Execute:=,
			Position:=,
			Done=>,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge of the input will start the processing of the function block
Position	Position that the axis reaches	LREAL	Data range	0	Home position of the axis

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Instruction execution completed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE after the axis instruction is executed completely
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
CommandAborted	Instruction aborted	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is aborted
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

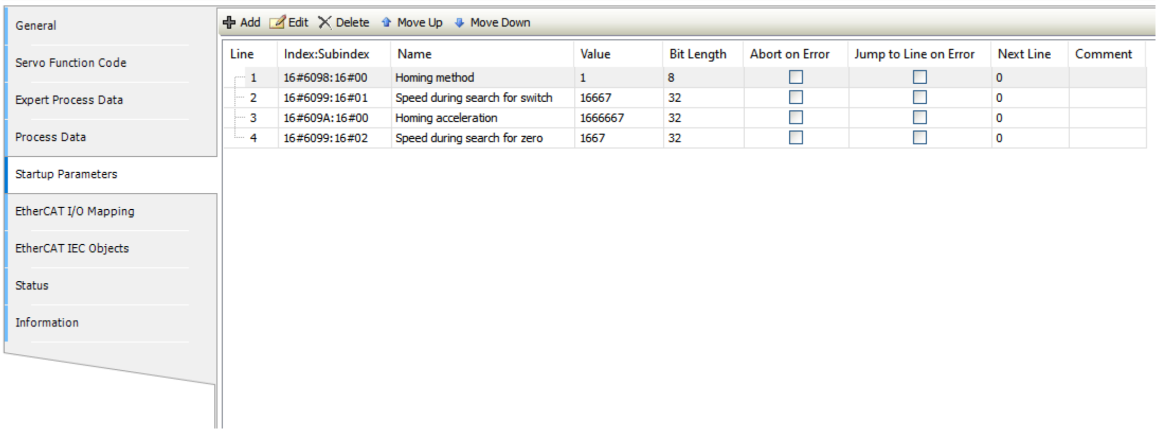
This function block is used for homing and it is started at the rising edge of the input variable execution condition. The position that the axis of the input variable reaches is the Home position. This function block can be executed only when the axis is in the Standstill state. In addition, the servo homing mode must be set before the execution, and the axis must be in the Homing state during the execution.

There are two methods for setting the homing mode:

- Method 1: Manually setting servo function codes, i.e. setting P5.10 on INVT servo DA200.
- Method 2: Setting startup parameters of AX series slaves. If communication modes are used, index and sub-index data must be set.

Item	Index	Sub-index	Description
Homing method	0x6098	-	Set parameters according to specific servo manuals
Origin finding speed	0x6099	0x01	Generally the speed is defined relatively high, reducing the homing time
Zero finding speed	0x6099	0x02	Generally the speed is defined relatively low
ACC/DEC for homing	0x609A	-	Acceleration or deceleration during homing
Homing timeout period	0x2005	0x24	If the homing time exceeds the specified time, the system reports "Err.601".

The corresponding Settings interface of AX series is as follows:



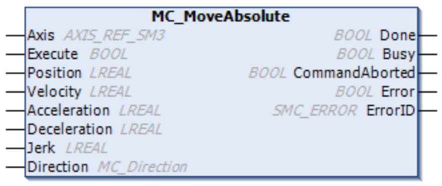
Line	Index:Subindex	Name	Value	Bit Length	Abort on Error	Jump to Line on Error	Next Line	Comment
1	16#6098:16#00	Homing method	1	8	☐	☐	0	
2	16#6099:16#01	Speed during search for switch	16667	32	☐	☐	0	
3	16#609A:16#00	Homing acceleration	1666667	32	☐	☐	0	
4	16#6099:16#02	Speed during search for zero	1667	32	☐	☐	0	

7.1.4 MC_MoveAbsolute

MC_MoveAbsolute: used to specify the destination position of absolute coordinates for positioning.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_MoveAbsolute	Axis absolute position control instruction	 MC_MoveAbsolute
		Axis AXIS_REF_SM3
		Execute BOOL
		Position LREAL
		Velocity LREAL
		Acceleration LREAL
		Deceleration LREAL
		Jerk LREAL
		Direction MC_Direction
		BOOL Done
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		SMC_ERROR ErrorID
			MC_MoveAbsolute(
			Axis:=,
			Execute:=,
			Position:=,
			Velocity:=,
			Acceleration:=,
			Deceleration:=,
			Jerk:=,
			Direction:=,
			Done=>,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge of the input will start the processing of the function block
Position	Position that the axis reaches	LREAL	Data range	0	Absolute position of the axis

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Velocity	Running speed	LREAL	Data range	0	Max. speed at which the axis runs to reach the destination position
Acceleration	Acceleration	LREAL	Data range	0	Acceleration when the speed increases
Deceleration	Deceleration	LREAL	Data range	0	Deceleration when the speed decreases
Jerk	Jerk	LREAL	Data range	0	Slope change value of the curve acceleration or deceleration
Direction	Instruction polarity	MC_DIRECTION	Negative, Shortest, Positive, Current, Fastest	Shortest	Negative: Move backward; Shortest: Select a direction depending on the shortest distance; Positive: Move forward; Current: Move in the current direction; Fastest: Automatically choose to move at fastest manner. **Note:** This function is valid only in rotary mode.

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Instruction execution completed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE after the axis instruction is executed completely
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
CommandAborted	Instruction aborted	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is aborted
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

This function block is the axis absolute-position instruction. Before executing this function block, the axis is in the Standstill state. After the function block is started at the rising edge of Execute, the axis is in the DiscreteMotion state and moves to the specified position. When Jerk is 0, the axis performs trapezoidal acceleration/deceleration movement; when Velocity, Acceleration, Deceleration, and Jerk are not empty, it performs S-curve acceleration/deceleration movement.

Figure 7-1 Trapezoidal Acceleration/Deceleration Action

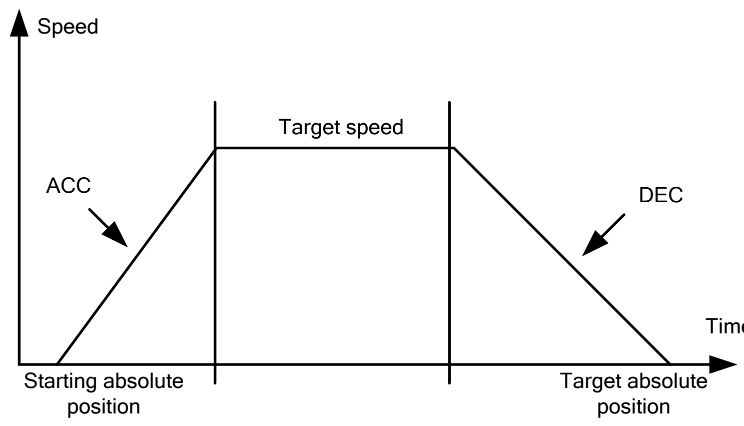
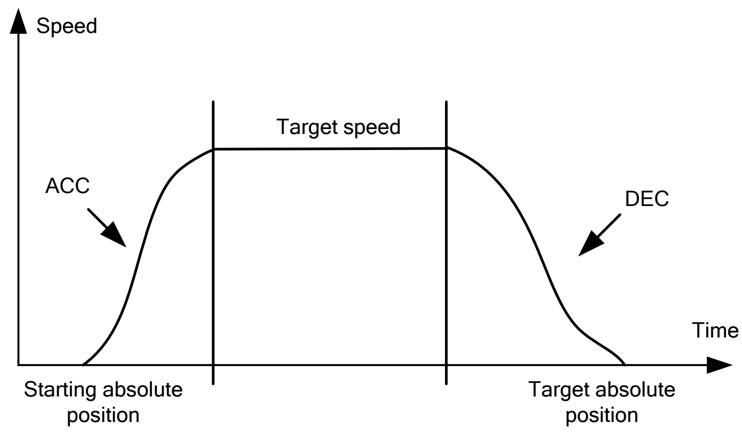
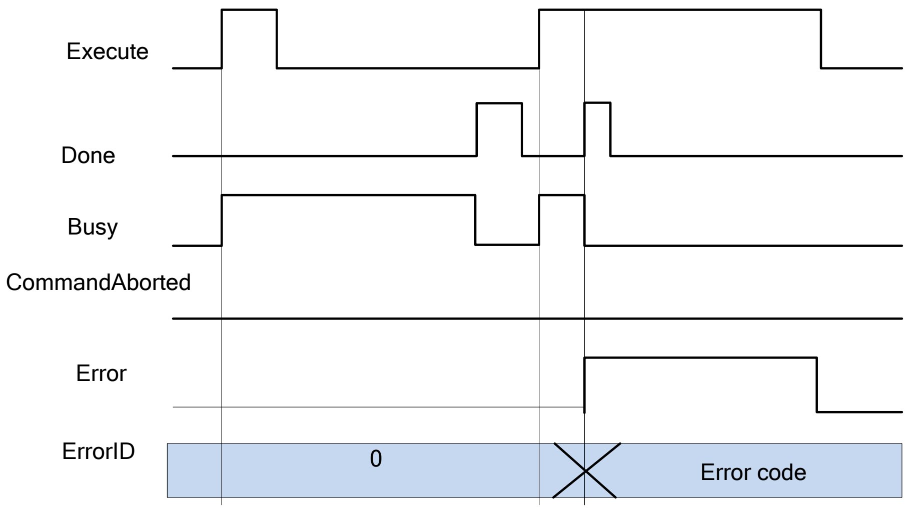


Figure 7-2 S-curve Acceleration/Deceleration Action



4. Timing diagram

- The axis must be in the Standstill state
- The function block is triggered at the rising edge of Execute.
- For the function block, when Done is TRUE, the execution is completed; otherwise, Busy is TRUE.

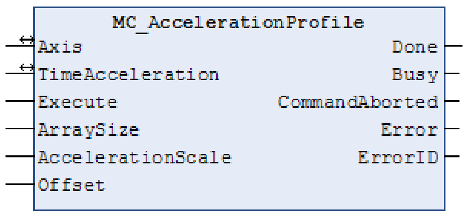


7.1.5 MC_AccelerationProfile

MC_AccelerationProfile: used to indicate the motion model of the time segment and acceleration/deceleration profile.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_AccelerationProfile	Acceleration profile instruction	 MC_AccelerationProfile
		Axis
		TimeAcceleration
		Execute
		ArraySize
		AccelerationScale
		Offset
		Done
		Busy
		CommandAborted
		Error
		ErrorID
			MC_AccelerationProfile(
			Axis:=,
			TimeAcceleration:=,
			Execute:=,
			ArraySize:=,
			AccelerationScale:=,
			Offset:=,
			Done=>,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3
TimeAcceleration	Acceleration time and description of the axis	MC_TA_REF	-	-	Acceleration time and data description of the axis. The acceleration data consists of multiple groups of data

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge of the input will start the processing of the function block
ArraySize	Dynamic array	INT	Data range	0	Number of arrays used in the motion profile
AccelerationScale	Comprehensive factor	LREAL	Positive or 0	1	Scale factor of acceleration or deceleration in MC_TA_REF
Offset	Offset	LREAL	-	0	Overall offset value of acceleration and deceleration

Output variables

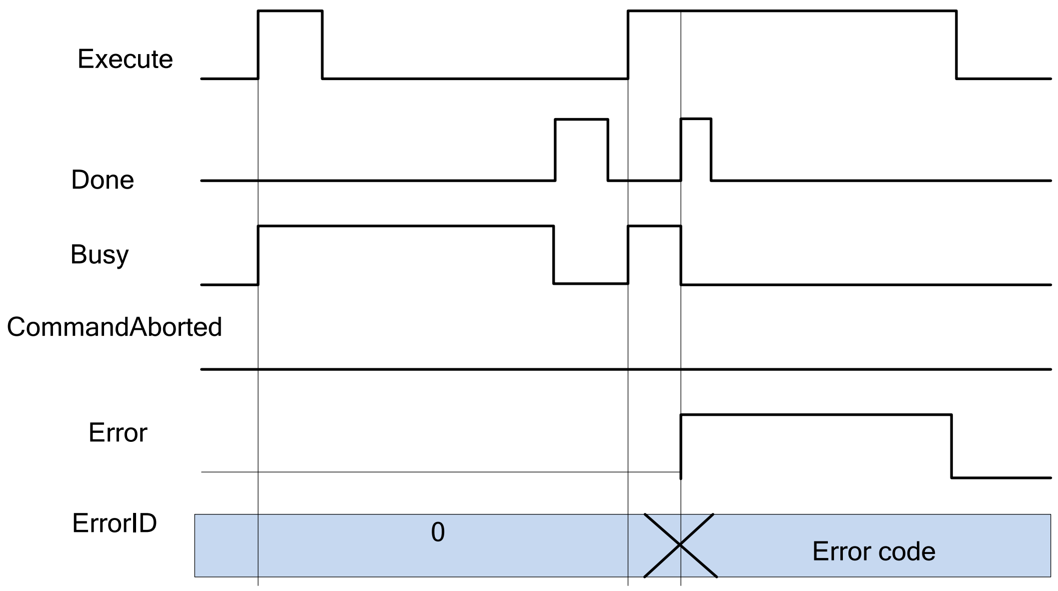
Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Instruction execution completed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE after the axis instruction is executed completely
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
CommandAborted	Instruction aborted	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is aborted
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
					exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

This function block is used to specify the motion model of the time segment and acceleration/deceleration profile. During the function block execution, the axis is in the DiscreteMotion state, and it uses the data in TimeAcceleration. The axis must be in the Standstill state before the function block execution and in the DiscreteMotion state during the execution. This function block is started at the rising edge of Execute. The execution of this function block superimposes the speeds of the axis that is in the DiscreteMotion state, which may cause system faults.

4. Timing diagram

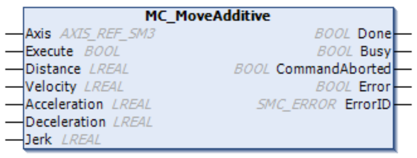


7.1.6 MC_MoveAdditive

MC_MoveAdditive: used for positioning when a specified distance is superimposed to the original position of an axis.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_MoveAdditive	Absolute motion superimposition instruction	 MC_MoveAdditive
		Axis AXIS_REF_SM3
		Execute BOOL
		Distance LREAL
		Velocity LREAL
		Acceleration LREAL
		Deceleration LREAL
		Jerk LREAL
		BOOL Done
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		SMC_ERROR ErrorID
			MC_MoveAdditive(
			Axis:=,
			Execute:=,
			Distance:=,
			Velocity:=,
			Acceleration:=,
			Deceleration:=,
			Jerk:=,
			Done=>,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge of the input will start the processing of the function block
Distance	Position that the axis reaches	LREAL	Data range	0	Superimposed position data of the axis
Velocity	Running speed	LREAL	Data range	0	Max. speed at which the axis runs to reach the destination position
Acceleration	Acceleration	LREAL	Data range	0	Acceleration when the speed increases
Deceleration	Deceleration	LREAL	Data range	0	Deceleration when the speed decreases
Jerk	Jerk	LREAL	Data range	0	Slope change value of the curve acceleration or deceleration

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Instruction execution completed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE after the axis instruction is executed completely
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
CommandAborted	Instruction aborted	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is aborted
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

The startup instruction is Execute, the rising edge triggers the function block, and Distance specifies the superimposed data of the axis. If the running state of this function block is DiscreteMotion, the CommandAbort values of other instructions are set; in the standstill state, this instruction can run independently to achieve relative positioning requirements; if Acceleration or Deceleration is zero, the instruction execution is abnormal, but the axis is in the DiscreteMotion state; When Jerk is 0, the axis performs trapezoidal acceleration/deceleration movement; when Velocity, Acceleration, Deceleration, and Jerk are not empty, it performs S-curve acceleration/deceleration movement.

Figure 7-3 Trapezoidal Acceleration/Deceleration Action

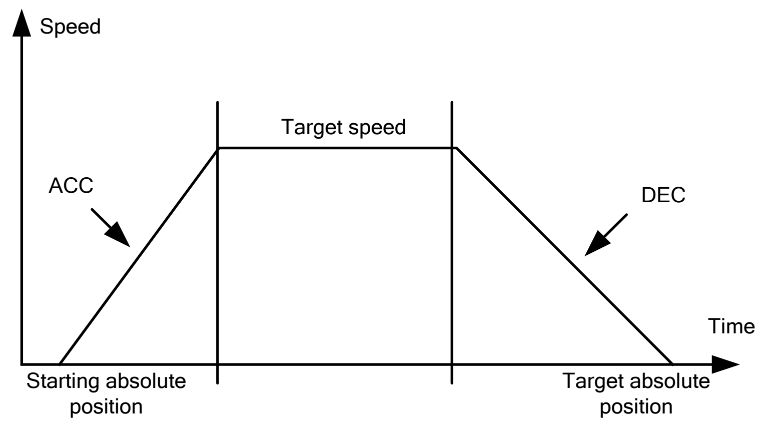
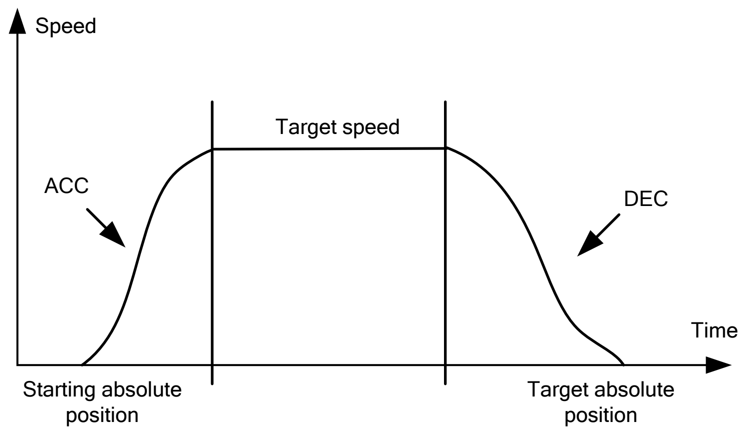
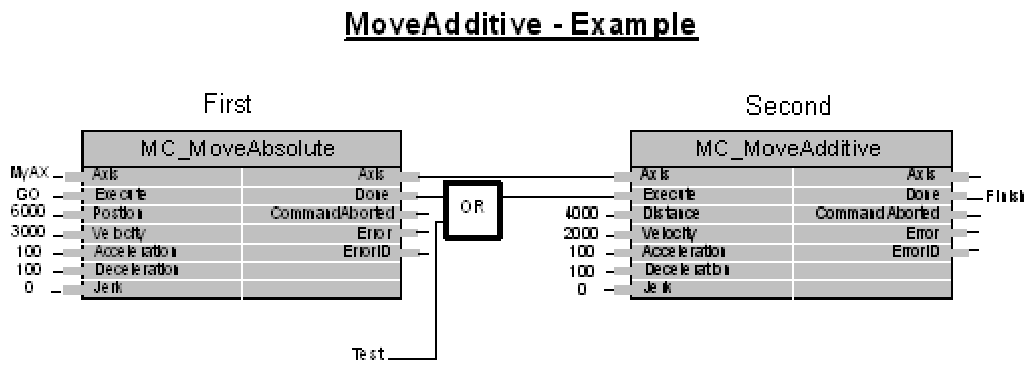


Figure 7-4 S-curve Acceleration/Deceleration Action

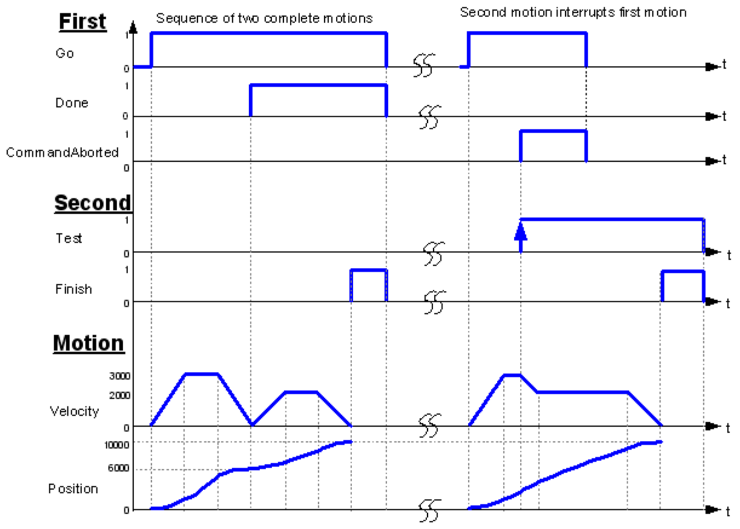


4. Timing diagram

Example



Timing description

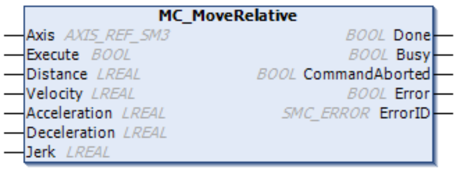


7.1.7 MC_MoveRelative

MC_Move Relative: used for positioning by specifying the movement distance from the current position.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_MoveRelative	Axis relative positioning instruction	 MC_MoveRelative
		Axis AXIS_REF_SM3
		Execute BOOL
		Distance LREAL
		Velocity LREAL
		Acceleration LREAL
		Deceleration LREAL
		Jerk LREAL
		BOOL Done
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		SMC_ERROR ErrorID
			MC_MoveRelative(
			Axis:=,
			Execute:=,
			Distance:=,
			Velocity:=,
			Acceleration:=,
			Deceleration:=,
			Jerk:=,
			Done=>,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge of the input will start the processing of the function block (FALSE→TRUE)
Distance	Relative position of motion	LREAL	Data range	0	The data is a relative position

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Velocity	Running speed	LREAL	Data range	0	Max. speed at which the axis runs to reach the destination position
Acceleration	Acceleration	LREAL	Data range	0	Acceleration when the speed increases
Deceleration	Deceleration	LREAL	Data range	0	Deceleration when the speed decreases
Jerk	Jerk	LREAL	Data range	0	Slope change value of the curve acceleration or deceleration

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Instruction execution completed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE after the axis instruction is executed completely
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
CommandAborted	Instruction aborted	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is aborted
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

The axis must be in the Standstill state before the function block execution and in the DiscreteMotion state during the execution. Pay attention to the axis state during the execution to prevent other instructions from interrupting the instruction execution of the axis. The startup instruction is Execute, and the rising edge (FALSE→TRUE) triggers the function block. The startup instruction can repeatedly make the rising edge valid when the axis is in the DiscreteMotion state, which always refreshes the position. When Acceleration or Deceleration is 0, the instruction execution is abnormal, but the axis is in the DiscreteMotion state.

Figure 7-5 Trapezoidal Acceleration/Deceleration Action

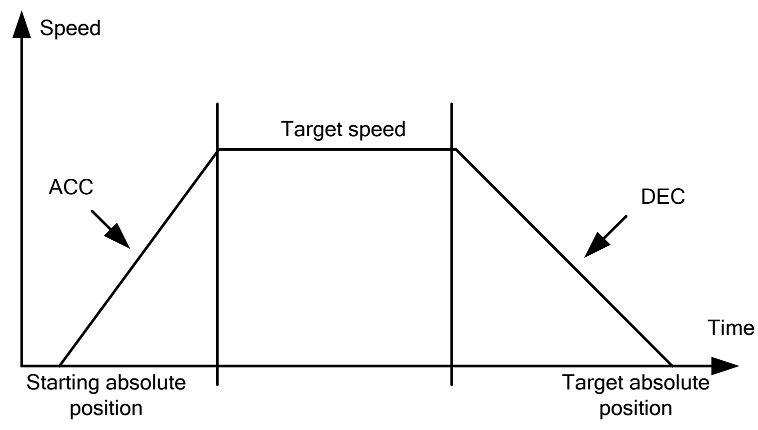
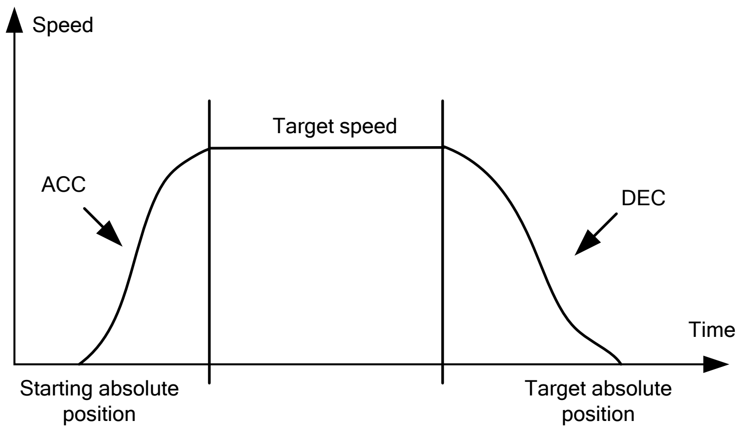
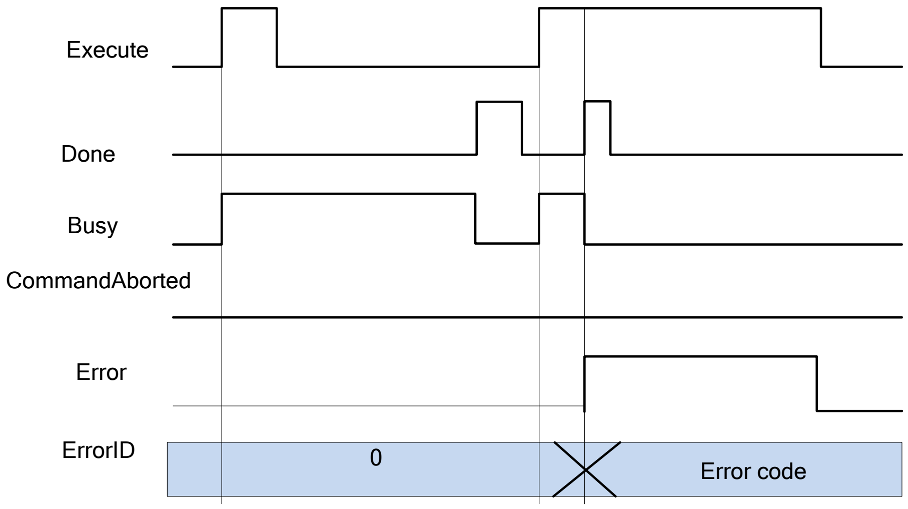


Figure 7-6 S-curve Acceleration/Deceleration Action



4. Timing diagram



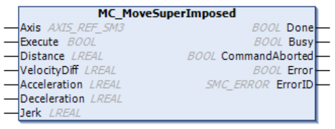
This function block is triggered at the rising edge of Execute. When Busy is set, the function block is being executed. After the execution is completed, Done is set.

7.1.8 MC_MoveSuperImposed

MC_MoveSuperImposed: used to superimpose speed and position data on the speed and position data in the running instruction, which brings no change to the entire original instruction execution time model.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_MoveSuperImposed	Relative motion superimposition instruction	 MC_MoveSuperImposed
		Axis AXIS_REF_SM3
		Execute BOOL
		Distance LREAL
		VelocityDiff LREAL
		Acceleration LREAL
		Deceleration LREAL
		Jerk LREAL
		BOOL Done
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		SMC_ERROR ErrorID
			MC_MoveSuperImposed(
			Axis:=,
			Execute:=,
			Distance:=,
			VelocityDiff:=,
			Acceleration:=,
			Deceleration:=,
			Jerk:=,
			Done=>,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge of the input will start the processing of the function block (FALSE→TRUE)
Distance	Relative position of motion	LREAL	Data range	0	The data is a relative position
VelocityDiff	Superimposition speed	LREAL	Data range	0	Superimposition speed for axis running
Acceleration	Acceleration	LREAL	Data range	0	Acceleration when the speed increases
Deceleration	Deceleration	LREAL	Data range	0	Deceleration when the speed decreases
Jerk	Jerk	LREAL	Data range	0	Slope change value of the curve acceleration or deceleration

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Instruction execution completed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE after the axis instruction is executed completely
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
CommandAborted	Instruction aborted	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is aborted
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

This function block is the position and speed superimposition instruction, which is started at the rising edge of Execute. VelocityDiff and Distance are superimposed to the speed and position of other instructions. In the motion mode, MC_MoveSuperImposed can be superimposed onto any other instruction. This function block can solve the error compensation for the clearance between the belt and gear, which can ensure motion consistency. To execute the function block, you need to set the parameter superimposition position.

Figure 7-7 Trapezoidal Acceleration/Deceleration Action

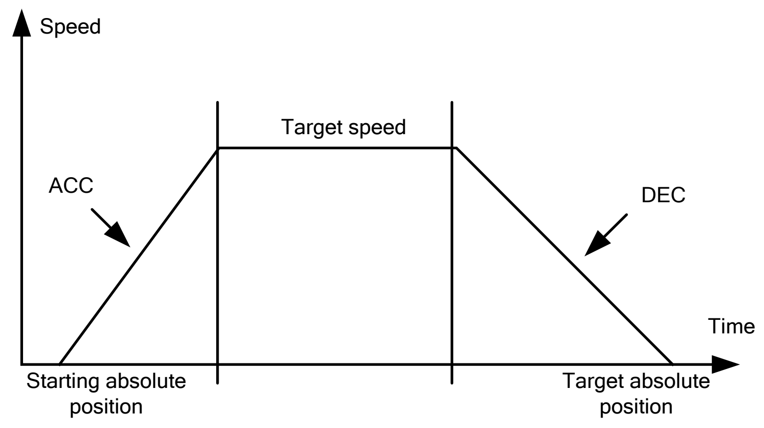
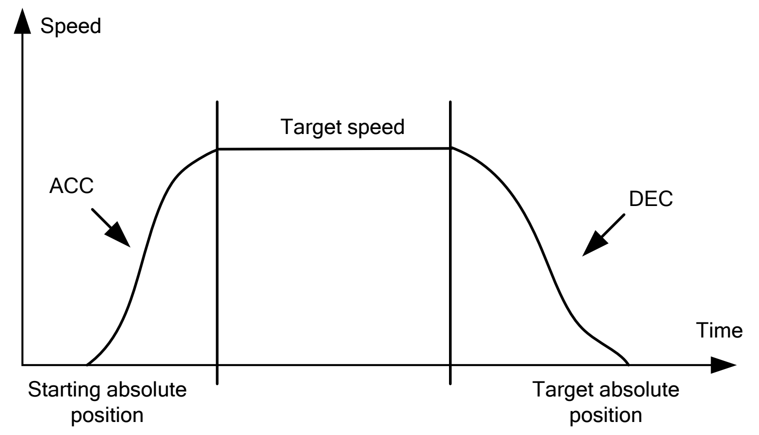
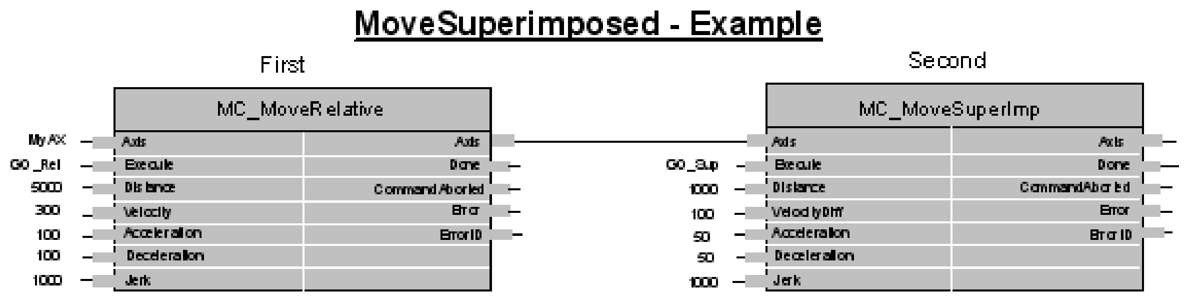


Figure 7-8 S-curve Acceleration/Deceleration Action

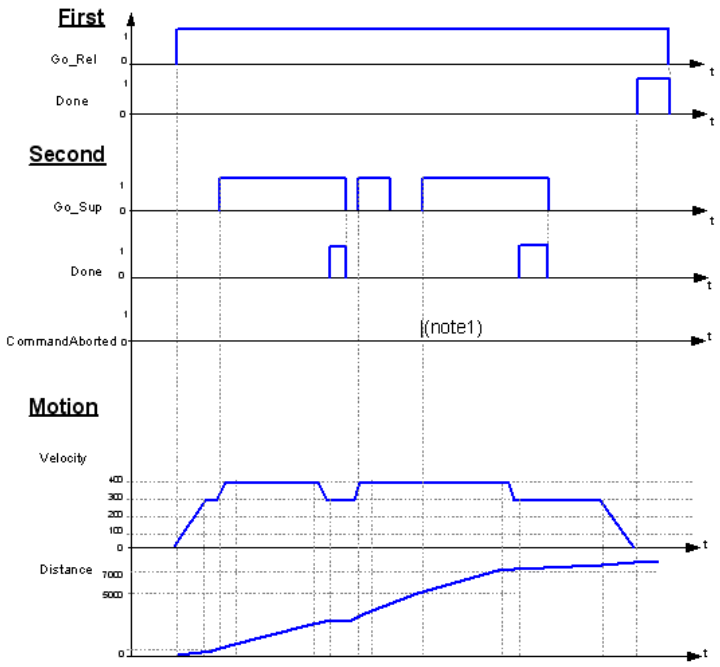


4. Timing diagram

Example



Timing description

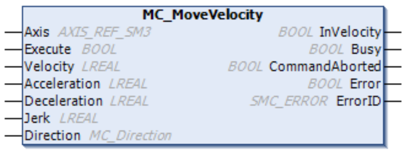


7.1.9 MC_MoveVelocity

MC_MoveVelocity: used to simulate speed control by using the servo drive position control mode.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_MoveVelocity	Speed control instruction		MC_MoveVelocity(
			Axis:=,
			Execute:=,
			Velocity:=,
			Acceleration:=,
			Deceleration:=,
			Jerk:=,
			Direction:=,
			InVelocity=>,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge of the input will start the processing of the function block (FALSE→TRUE)
Velocity	Running speed	LREAL	Data range	0	Specified speed for running
Acceleration	Acceleration	LREAL	Data range	0	Acceleration when the speed increases
Deceleration	Deceleration	LREAL	Data range	0	Deceleration when the speed decreases
Jerk	Jerk	LREAL	Data range	0	Slope change value of the curve acceleration or deceleration
Direction	Running direction	MC_Direction	Positive, Negative, Current	Current	Running direction

Output variables

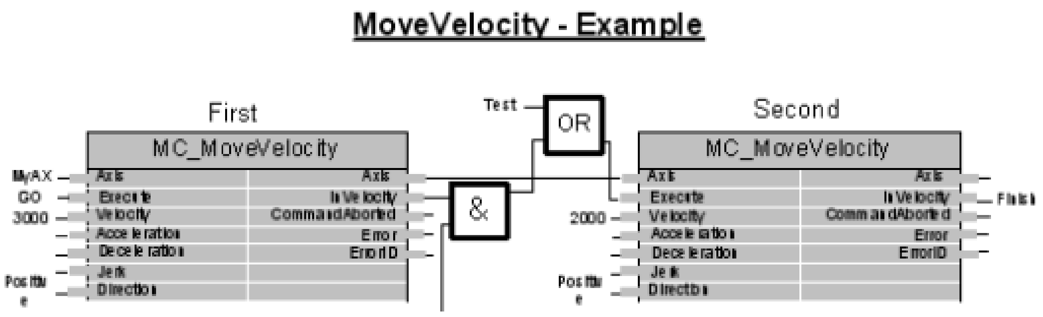
Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Instruction execution completed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE after the axis instruction is executed completely
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
CommandAborted	Instruction aborted	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is aborted
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

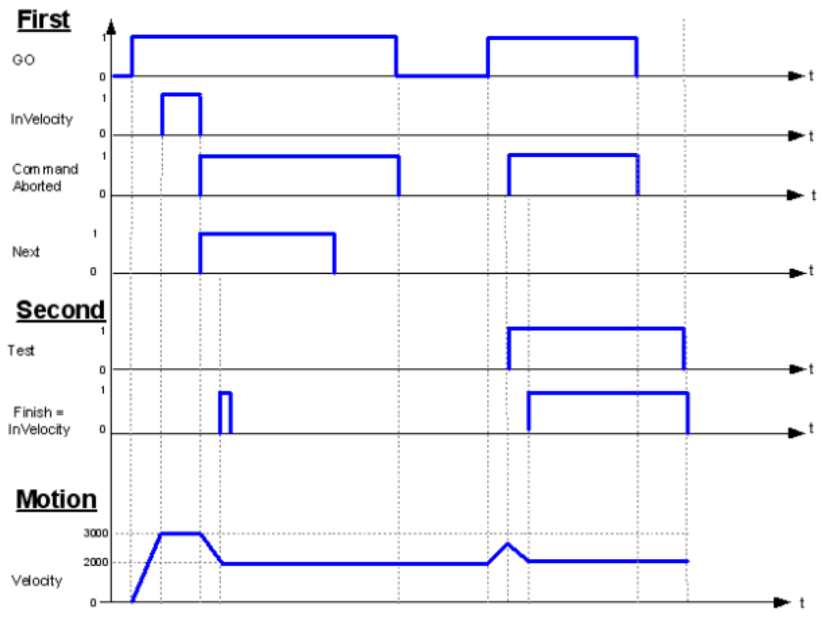
This function block is triggered at the rising edge of Execute. The drive performs speed control according to the value of Velocity. InVelocity indicates that the running speed in the function block has reached the specified value.

4. Timing diagram

Example



Timing description

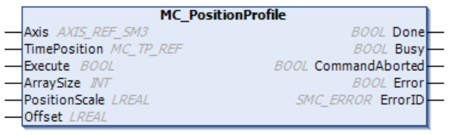


7.1.10 MC_PositionProfile

MC_PositionProfile: used to indicate the motion model of the time segment and position profile.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_PositionProfile	Position profile instruction	 MC_PositionProfile Axis AXIS_REF_SM3 TimePosition MC_TP_REF Execute BOOL ArraySize INT PositionScale LREAL Offset LREAL BOOL Done BOOL Busy BOOL CommandAborted BOOL Error SMC_ERROR ErrorID	MC_PositionProfile(
			Axis:=,
			TimePosition:=,
			Execute:=,
			ArraySize:=,
			PositionScale:=,
			Offset:=,
			Done=>,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3
TimePosition	Running time and position description	MC_TP_REF	-	-	Running time and position data description of the axis. The data consists of multiple groups of data

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge of the input will start the processing of the function block (FALSE→TRUE)
ArraySize	Array size	INT	Data range	0	Number of arrays used in the motion profile
PositionScale	Comprehensive factor	LREAL	Positive or 0	0	Position scaling factor in MC_TP_REF
Offset	Offset	LREAL	-	0	Overall offset value of the position

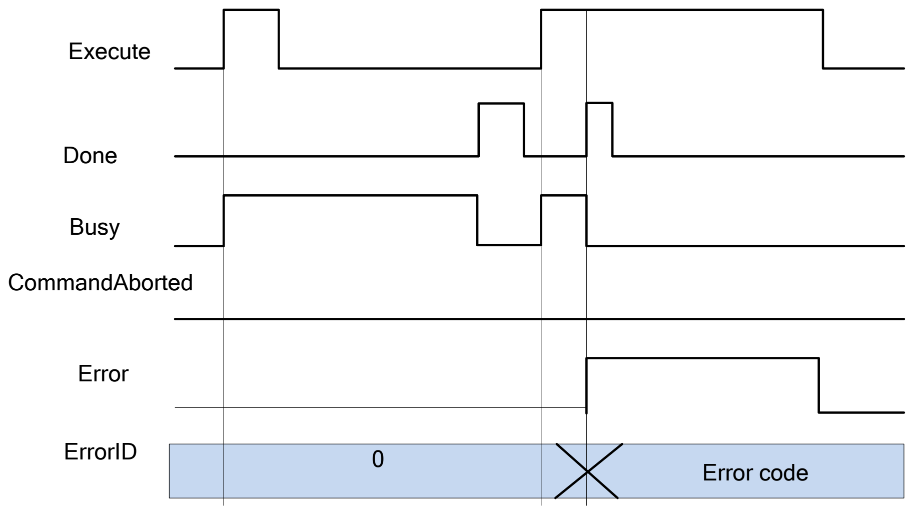
Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Instruction execution completed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE after the axis instruction is executed completely
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
CommandAborted	Instruction aborted	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is aborted
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

This function block is used to specify the motion model of the time segment and position profile, using the data in TimePosition. Before executing this function block, the axis is in the Standstill state. This function block is triggered at the rising edge of Execute. The axis is in the DiscreteMotion state during the function block execution.

4. Timing diagram

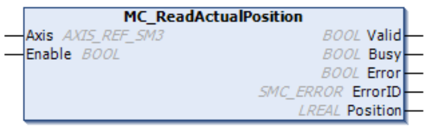


7.1.11 MC_ReadActualPosition

MC_ReadActualPosition: used to read the actual position of the drive and save it to a user-defined variable.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_
ReadActualPosition
	Actual position reading instruction	 MC_ReadActualPosition
		Axis AXIS_REF_SM3
		Enable BOOL
		BOOL Valid
		BOOL Busy
		BOOL Error
		SMC_ERROR ErrorID
		LREAL Position
			MC_ReadActualPosition(
			Axis:=,
			Enable:=,
			Valid=>,
			Busy=>,
			Error=>,
			ErrorID=>,
			Position=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge of the input will start the processing of the function block (FALSE→TRUE)

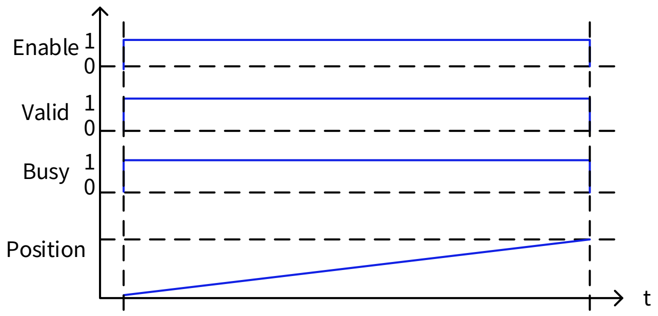
Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Valid	Obtainable flag of position data	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the drive position can be obtained correctly.
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output
Position	Obtained axis position	LREAL	Axis position	0	Axis position data that is read

3. Function description

This function block is triggered at the rising edge of Execute and it can read the axis position value. When Valid is TRUE, the read position value is valid. This function block can be repeatedly called, and the invoking does not affect the other.

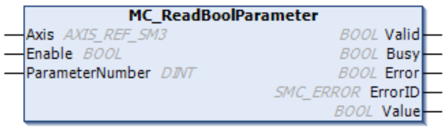
4. Timing diagram



7.1.12 MC_ReadBoolParameter

MC_ReadBoolParameter: used to read the bit parameters of the drive axis and save them to user-defined variables.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation
MC_ReadBoolParameter	Axis bit parameter reading instruction		MC_ReadBoolParameter(Axis:=, Enable:=, ParameterNumber:=, Valid=>, Busy=>, Error=>, ErrorID=>, Value=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Execution condition	BOOL	TRUE, FALSE	FALSE	When it is set to TRUE, this function block is started
ParameterNumber	Axis parameter number	DINT	-	0	Access index, sub-index, and number of the axis parameter

Note:

- ParameterNumber (DINT) = -DWORD_TO_DINT(SHL(USINT_TO_DOWRD(usiDataLength), 24) (Data length in the object dictionary) + SHL(UINT_TO_DWORD(uiIndex), 8) (Index in the object dictionary-16 bits) + usisubIndex (Sub-index in the object dictionary-8 bits).
- usiDataLength: Fill in according to the number of bytes: Byte 1 is 16#01; byte 2 is 16#02; byte 4 is 16#04, and so on.

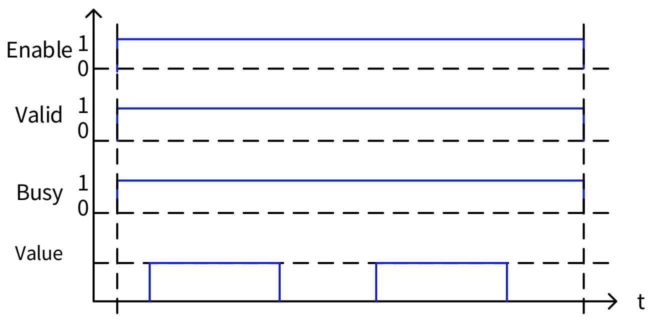
Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Valid	Obtainable flag of position data	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the drive position can be obtained correctly.
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output
Value	Obtained axis position	BOOL	TRUE, FALSE	FALSE	The value of the parameter ParameterNumber is read

3. Function description

Bit data status is read from the drive by executing MC_ReadBoolParam, which is valid when Enable is TRUE. This function block can be repeatedly executed without affecting each other. When Valid is TRUE, the bit status data is valid; when Busy is TRUE, the function block is being executed.

4. Timing diagram

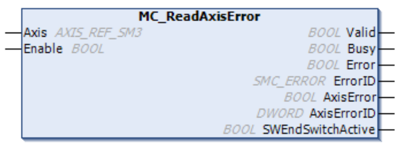


7.1.13 MC_ReadAxisError

MC_Read AxisError: used to read axis error information and save it to user-defined variables.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_ReadAxisError	Axis error reading instruction		MC_ReadAxisError(
			Axis:=,
			Enable:=,
			Valid=>,
			Busy=>,
			Error=>,
			ErrorID=>,
			AxisError=>,
			AxisErrorID=>,
			SWEndSwitchActive=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Execution condition	BOOL	TRUE, FALSE	FALSE	When it is set to TRUE, this function block is started

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Valid	Error data obtaining flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the drive position can be obtained correctly.
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	TRUE, FALSE	FALSE	When an exception occurs, the error ID is output
AxisError	Axis error flag	BOOL	TRUE, FALSE	FALSE	When an error is read, the corresponding flag is set
AxisErrorID	Axis error ID	DWORD	-	0	The axis error ID is read
SWEndSwitch Active	Soft limit switch valid	BOOL	TRUE, FALSE	FALSE	The soft limit switch status is checked during instruction reading

3. Function description

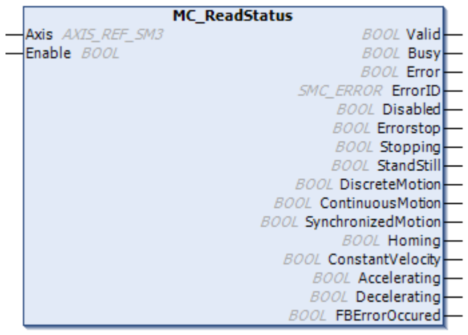
This function block is used to read axis error information, and it is valid when Enable is TRUE. When Valid is TRUE, AxisError and AxisErrorID are valid data values; when Busy is TRUE, the current function block is being executed. This function block can be repeatedly executed without affecting each other.

7.1.14 MC_ReadStatus

MC_ReadStatus: used to read axis status data and save it to user-defined variables.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_ReadStatus	Axis status reading instruction	 MC_ReadStatus
		Axis AXIS_REF_SM3
		Enable BOOL
		BOOL Valid
		BOOL Busy
		BOOL Error
		SMC_ERROR ErrorID
		BOOL Disabled
		BOOL Errorstop
		BOOL Stopping
		BOOL StandStill
		BOOL DiscreteMotion
		BOOL ContinuousMotion
		BOOL SynchronizedMotion
		BOOL Homing
		BOOL ConstantVelocity
		BOOL Accelerating
		BOOL Decelerating
		BOOL FBErrorOccured
			MC_ReadStatus(
			Axis:=,
			Enable:=,
			Valid=>,
			Busy=>,
			Error=>,
			ErrorID=>,
			Disabled=>,
			Errorstop=>,
			Stopping=>,

Instruction	Name	Graphical Representation	ST Representation
			StandStill=>, DiscreteMotion=>, ContinuousMotion=>, SynchronizedMotion=>, Homing=>, ConstantVelocity=>, Accelerating=>, Decelerating=>, FBErrorOccured=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Execution condition	BOOL	TRUE, FALSE	FALSE	When it is set to TRUE, this function block is started

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Valid	Error data obtaining flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the drive position can be obtained correctly.
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output
Disabled	Axis disabled	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis is disabled
Errorstop	Axis error status	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis is running abnormally
Stopping	Axis in stop process	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis is in the stop process
StandStill	Standard status of axis	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis is in the StandStill state (able to run)
DiscreteMotion	Discrete motion status of axis	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis is in the DiscreteMotion state
ContinuousMotion	Continuous motion status of axis	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis is in the ContinuousMotion state

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
SynchronizedMotion	Synchronous running status of axis	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis is in the SynchronizedMotion state
Homing	Homing status of axis	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis is in the Homing state
ConstantVelocity	Axis running speed reached	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis reaches the running speed
Accelerating	Acceleration status of axis	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis is in the Accelerating state
Dccelerating	Deceleration status of axis	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis is in the Dccelerating state
FBErrorOccured	Axis function block error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis function block encounters an error

3. Function description

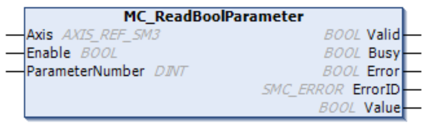
Axis status is read by executing MC_ReadStatus, which is valid when Enable is TRUE. This function block can be repeatedly executed without affecting each other. To execute the function block, set Enable to TRUE. When Valid is TRUE, the axis status data is valid; when Busy is TRUE, the function block is being executed.

7.1.15 MC_ReadParameter

MC_ReadParameter: used to read drive axis parameters and saves them to user-defined variables.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_ReadParameter	Axis parameter reading instruction	 MC_ReadBoolParameter
		Axis AXIS_REF_SM3
		Enable BOOL
		ParameterNumber DINT
		BOOL Valid
		BOOL Busy
		BOOL Error
		SMC_ERROR ErrorID
		BOOL Value
			MC_ReadParameter(
			Axis:=,
			Enable:=,
			ParameterNumber:=,
			Valid=>,
			Busy=>,
			Error=>,
			ErrorID=>,
			Value=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Execution condition	BOOL	TRUE, FALSE	FALSE	When it is set to TRUE, this function block is started
ParameterNumber	Axis parameter number	DINT	-	0	Access index, sub-index, and number of the axis parameter

Note:

- ParameterNumber (DINT) = -DWORD_TO_DINT(SHL(USINT_TO_DOWRD(usiDataLength), 24) (Data length in the object dictionary) + SHL(UINT_TO_DWORD(uiIndex), 8) (Index in the object dictionary-16 bits) + usisubIndex (Sub-index in the object dictionary-8 bits).
- usiDataLength: Fill in according to the number of bytes: Byte 1 is 16#01; byte 2 is 16#02; byte 4 is 16#04, and so on.

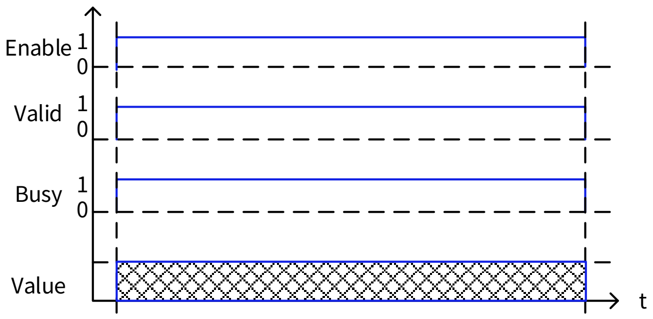
Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Valid	Obtainable flag of position data	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the drive position can be obtained correctly.
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output
Value	Obtained axis position	BOOL	TRUE, FALSE	FALSE	The value of the parameter ParameterNumber is read

3. Function description

Bit data status is read from the drive by executing MC_ReadBoolParam, which is valid when Enable is TRUE. The function block can be repeatedly executed without affecting each other. When Valid is TRUE, the bit status data is valid; when Busy is TRUE, the function block is being executed.

4. Timing diagram

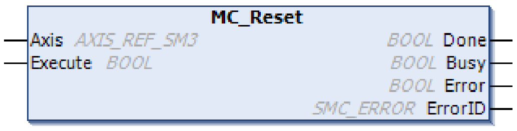


7.1.16 MC_Reset

MC_Reset: used to reset all errors of an axis.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_Reset	Axis error reset instruction	 MC_Reset
		Axis AXIS_REF_SM3
		Execute BOOL
		BOOL Done
		BOOL Busy
		BOOL Error
		SMC_ERROR ErrorID
			MC_Reset(
			Axis:=,
			Execute:=,
			Done=>,
			Busy=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge will start the processing of the function block

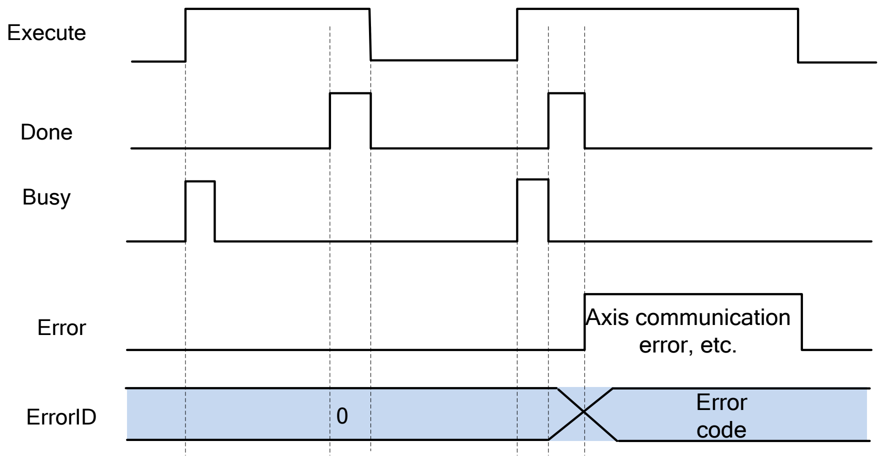
Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Instruction execution completed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE after the axis instruction is executed completely
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

This function can change the axis status from Errorstop to Standstill when the axis is in normal communication. If the axis cannot be reset from the Errostop state and Axis.bCommunication is FALSE, you must re-establish the communication between the master and slave axes.

4. Timing diagram

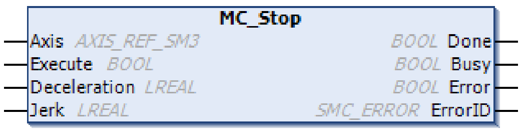


7.1.17 MC_Stop

MC_Stop: used to instruct an axis to decelerate to stop.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_Stop	Axis stop instruction	 MC_Stop
		Axis AXIS_REF_SM3
		Execute BOOL
		Deceleration LREAL
		Jerk LREAL
		BOOL Done
		BOOL Busy
		BOOL Error
		SMC_ERROR ErrorID
			MC_Stop(
			Axis:=,
			Execute:=,
			Deceleration:=,
			Jerk:=,
			Done=>,
			Busy=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge will start the processing of the function block
Deceleration	Deceleration	LREAL	Positive or 0	0	Function block deceleration speed (μ/S2)
Jerk	Jerk	LREAL	Positive or 0	0	Specified jerk [instruction unit/S3]

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Instruction execution completed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE after the axis instruction is executed completely
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

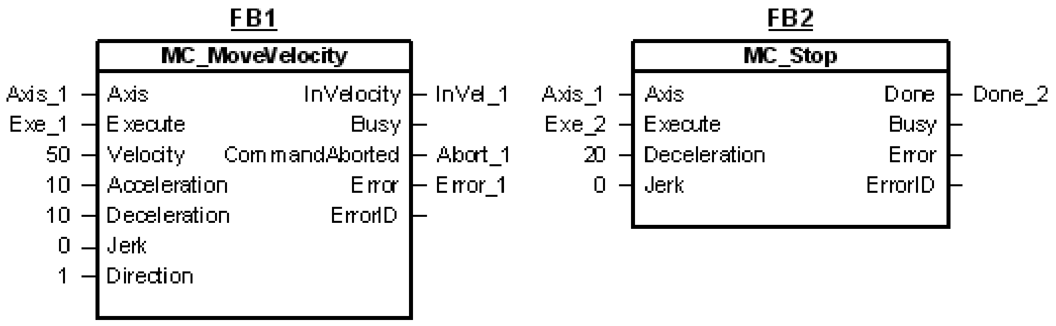
This function block is used to stop the motion of an axis that is in normal running. It does not take effect to the axis when it is in the Stopping state.

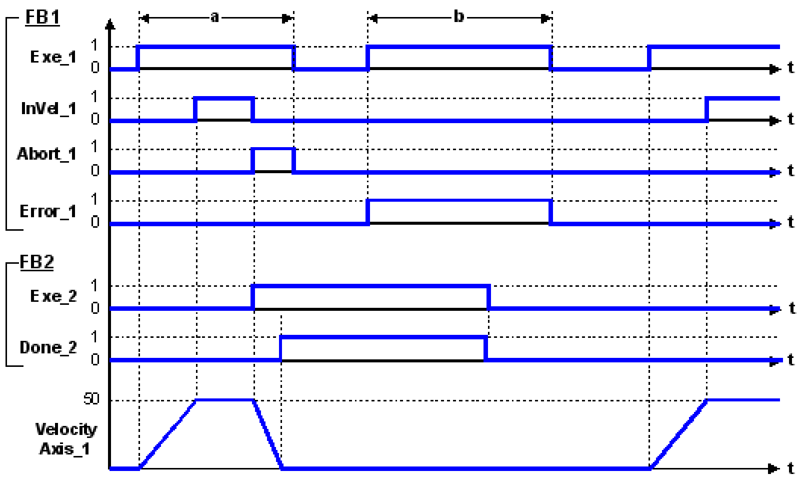
If the axis is in the Stopping state, Execute is FALSE, and Done is TRUE, and the axis status changes to Standstill. The function block is started at the rising edge of Execute. If Busy is TRUE when MC_Stop is in the execution process, the restart of MC_Stop will cause the axis to enter the Errorstop state. When the MC_Stop (forced stop) instruction is started, the instruction in execution changes to execute CommandAborted (execution aborted).

4. Timing diagram

Example

Flag bit difference in executing MC_MoveVelocity and MC_Stop:



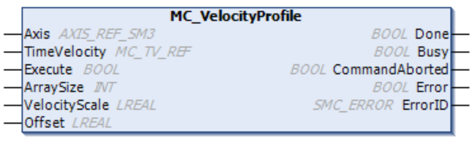


7.1.18 MC_VelocityProfile

MC_VelocityProfile: used to indicate the motion model of the time segment and speed profile.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_VelocityProfile	Speed profile instruction		MC_VelocityProfile(
			Axis:=,
			TimeVelocity:=,
			Execute:=,
			ArraySize:=,
			VelocityScale:=,
			Offset:=,
			Done=>,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
					instance of AXIS_REF_SM3
TimeVelocity	Running time and speed description of axis	MC_TV_REF	-	-	Running time and speed data description of the axis. The data consists of multiple groups of data

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge will start the processing of the function block
ArraySize	Dynamic array	INT	-	0	Number of arrays used in the motion profile
VelocityScale	Speed factor	LREAL	Positive or 0	1	Speed scaling factor
Offset	Offset	LREAL	-	0	Overall offset value of the speed

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Instruction execution completed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE after the axis instruction is executed completely
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

This function block is used to specify the motion model of the time segment and speed profile. The axis running mode is Continuous Motion, and the function block uses the data in TimeVelocity. The axis must be in the Standstill state before the function block execution and in the DiscreteMotion state during the execution. This function block is started at the rising edge of Execute. This function block can be repeatedly executed when the axis is in the DiscreteMotion state. TimeVelocity is of the MC_TV_REF data type.

MC_TV_REF is described as follows:

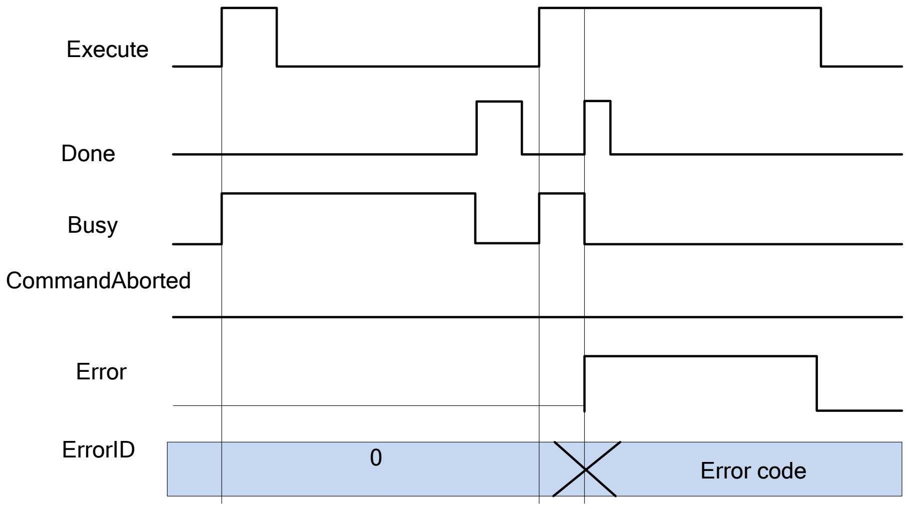
Member	Type	Initial Value	Description
Number_of_pairs	INT	0	Number of profile path segments
IsAbsolute	BOOL	TRUE	Absolute motion (TRUE) or relative motion (FALSE)
MC_TV_Array	ARRAY[1...N] OF SMC_TV	-	Data arrays of time and speed

SMC_TV is described as follows:

Member	Type	Initial Value	Description
delta_time	TIME	TIME#0ms	Time of a speed segment
Velocity	LREAL	0	Speed that is recorded currently

Note: The entire speed process represents the S curve with acceleration and deceleration, and the speed of each profile segment is calculated by superimposition; during repeated running, the speed is also superimposed to avoid the occurrence of speed limit exceeding; before repeated running, the axis status must be set to Standstill.

4. Timing diagram

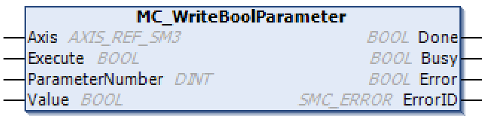


7.1.19 MC_WriteBoolParameter

MC_WriteBoolParameter: used to set the bit parameters of the drive axis.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_WriteBoolParameter	Axis bit parameter setting instruction	 MC_WriteBoolParameter
		Axis AXIS_REF_SM3
		Execute BOOL
		ParameterNumber DINT
		Value BOOL
		BOOL Done
		BOOL Busy
		BOOL Error
		SMC_ERROR ErrorID
			MC_WriteBoolParameter(
			Axis:=,
			Execute:=,
			ParameterNumber:=,
			Value:=,
			Done=>,
			Busy=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Execution	BOOL	TRUE,	FALSE	When it is set to TRUE, this

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
	condition		FALSE		function block is started
ParameterNumber	Axis parameter number	DINT	-	0	Access index, sub-index, and number of the axis parameter
Value	Setting	BOOL	TRUE, FALSE	FALSE	Used to set the bit parameters

Note:

- ParameterNumber (DINT) = -DWORD_TO_DINT(SHL(USINT_TO_DOWRD(usiDataLength), 24) (Data length in the object dictionary) + SHL(UINT_TO_DWORD(uiIndex), 8) (Index in the object dictionary-16 bits) + usisubIndex (Sub-index in the object dictionary-8 bits).
- usiDataLength: Fill in according to the number of bytes: Byte 1 is 16#01; byte 2 is 16#02; byte 4 is 16#04, and so on.

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Setting succeeded	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the setting operation succeeds.
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

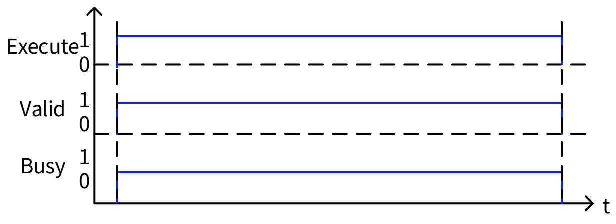
Axis bit parameters are set by executing MC_WriteBoolParameter, which is started at the rising edge. This function block can be repeatedly executed

without affecting each other.

4. Timing diagram

- The function block can be triggered only at the rising edge.
- When Done is TRUE, the setting operation is successful.
- When Busy is TRUE, the function block is being executed.

Timing description:

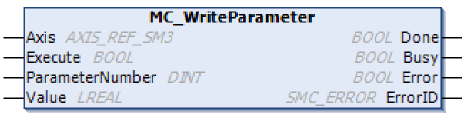


7.1.20 MC_WriteParameter

MC_WriteParameter: used to set the parameters of the drive axis.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_WriteParameter	Axis parameter setting instruction	 MC_WriteParameter
		Axis AXIS_REF_SM3
		Execute BOOL
		ParameterNumber DINT
		Value LREAL
		BOOL Done
		BOOL Busy
		BOOL Error
		SMC_ERROR ErrorID
			MC_WriteParameter(
			Axis:=,
			Execute:=,
			ParameterNumber:=,
			Value:=,
			Done=>,
			Busy=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Execution condition	BOOL	TRUE, FALSE	FALSE	When it is set to TRUE, this function block is started
ParameterNumber	Axis parameter number	DINT	-	0	Access index, sub-index, and number of the axis parameter
Value	Setting	BOOL	TRUE, FALSE	FALSE	Used to set the bit parameters

Note:

- ParameterNumber (DINT) = -DWORD_TO_DINT(SHL(USINT_TO_DOWRD(usiDataLength), 24) (Data length in the object dictionary) + SHL(UINT_TO_DWORD(uiIndex), 8) (Index in the object dictionary-16 bits) + usisubIndex (Sub-index in the object dictionary-8 bits).
- usiDataLength: Fill in according to the number of bytes: Byte 1 is 16#01; byte 2 is 16#02; byte 4 is 16#04, and so on.

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Setting succeeded	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the setting operation succeeds.
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

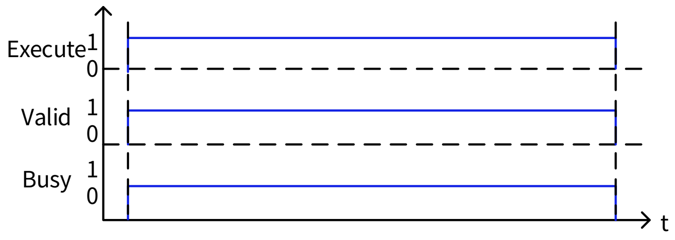
3. Function description

Axis bit parameters are set by executing MC_WriteParameter, which is started at the rising edge. This function block can be repeatedly executed without affecting each other.

4. Timing diagram

- The function block can be triggered only at the rising edge.
- When Done is TRUE, the setting operation is successful.
- When Busy is TRUE, the function block is being executed.

Timing description:

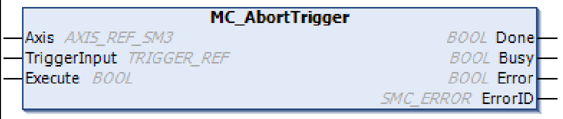


7.1.21 MC_AbortTrigger

MC_AbortTrigger: used to terminate the association features of latch related events, in conjunction with MC_Touchprobe.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_AbortTrigger	Event association termination instruction		MC_AbortTrigger(
			Axis:=,
			TriggerInput:=,
			Execute:=,
			Done=>,
			Busy=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3
TriggerInput	Trigger signal	TRIGGER_REF			Description of trigger signal and attributes

TRIGGER_REF description:

Input/Output Variable	Name	Data Type	Initial Value	Description
TRIGGER_REF	iTriggerNumber	INT	-1	Used to select a function to lock in the drive mode:

Input/Output Variable	Name	Data Type	Initial Value	Description
				0: Rising edge latching for probe 1 1: Falling edge latching for probe 1 2: Rising edge latching for probe 2 3: Falling edge latching for probe 2
	bFastLatching	BOOL	TRUE	Used to specify the latching trigger mode: TRUE: Driver mode FALSE: Controller mode
	bInput	BOOL	FALSE	When bFastLatching=FALSE, the controller inputs a signal for trigger
	bActive	BOOL	FALSE	Valid signal for trigger

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge will start the processing of the function block

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Setting succeeded	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the setting operation succeeds.
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

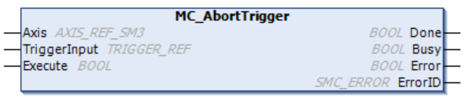
The MC_AbortTrigger function block is used to terminate the association between the trigger signal/attribute and the associated trigger instruction. The function block can be triggered only at the rising edge of Execute. When Done is TRUE, the setting operation is successful; when Busy is TRUE, the function block is being executed.

7.1.22 MC_ReadActualTorque

MC_ReadActualTorque: used to read the actual torque of the drive and save it to a user-defined variable.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_ReadActualTorque	Actual torque reading instruction	 MC_AbortTrigger
		Axis AXIS_REF_SM3
		TriggerInput TRIGGER_REF
		Execute BOOL
		BOOL Done
		BOOL Busy
		BOOL Error
		SMC_ERROR ErrorID
			MC_ReadActualTorque(
			Axis:=,
			Enable:=,
			Valid=>,
			Busy=>,
			Error=>,

Instruction	Name	Graphical Representation	ST Representation

			ErrorID=>,
			Torque=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Execution condition	BOOL	TRUE, FALSE	FALSE	When it is set to TRUE, this function block is started

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Valid	Actual torque obtaining flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the drive torque can be obtained correctly
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output
Torque	Actual torque obtaining	LREAL	Torque	0	Actual torque data that is read

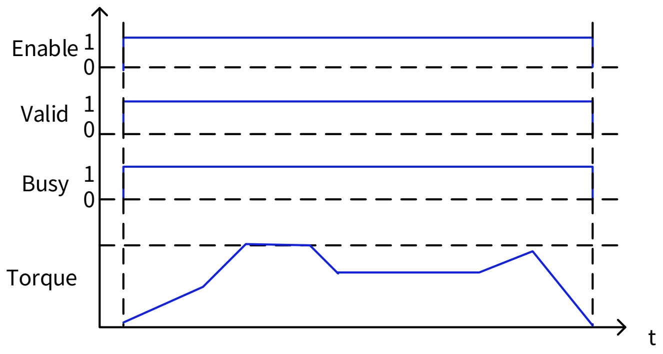
3. Function description

Actual torque data is read by executing MC_ReadActualTorque, which is valid when Enable is TRUE. This function block can be repeatedly executed without affecting each other.

4. Timing diagram

- Enable must be TRUE.
- When Valid=TRUE, the read torque is valid.
- When Busy is TRUE, the function block is being executed.

Timing description

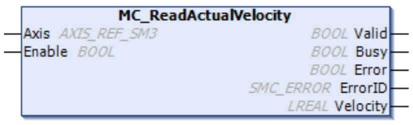


7.1.23 MC_ReadActualVelocity

MC_ReadActualVelocity: used to read the actual speed of the drive and save it to a user-defined variable.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_ReadActualVelocity	Actual speed reading instruction	 MC_ReadActualVelocity
		Axis AXIS_REF_SM3
		Enable BOOL
		BOOL Valid
		BOOL Busy
		BOOL Error
		SMC_ERROR ErrorID
		LREAL Velocity
			MC_ReadActualVelocity(
			Axis:=,
			Enable:=,
			Valid=>,
			Busy=>,
			Error=>,
			ErrorID=>,
			Velocity=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Execution condition	BOOL	TRUE, FALSE	FALSE	When it is set to TRUE, this function block is started

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Valid	Actual torque obtaining flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the drive torque can be obtained correctly
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output
Velocity	Actual speed obtaining	LREAL	Speed	0	Actual speed data that is read

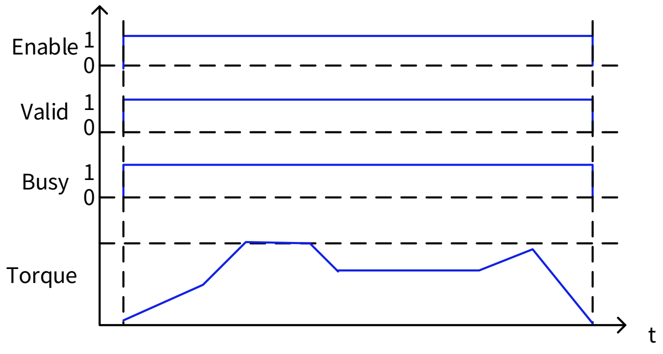
3. Function description

Actual speed data is read by executing MC_ReadActualVelocity, which is valid when Enable is TRUE. This function block can be repeatedly executed without affecting each other.

4. Timing diagram

- Enable must be TRUE.
- When Valid=TRUE, the read torque is valid.
- When Busy is TRUE, the function block is being executed.

Timing description

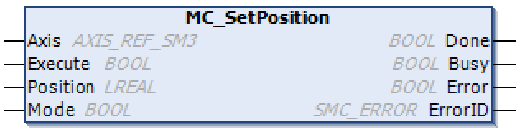


7.1.24 MC_SetPosition

MC_SetPosition: used to set the position data in the instruction as the position data of an axis, without causing any movement for setting position data. It is designed for shifting the coordinate system of an axis.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_SetPosition	Position change instruction	 MC_SetPosition Axis AXIS_REF_SM3 Execute BOOL Position LREAL Mode BOOL BOOL Done BOOL Busy BOOL Error SMC_ERROR ErrorID	MC_SetPosition(
			Axis:=,
			Execute:=,
			Position:=,
			Mode:=,
			Done=>,
			Busy=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge will start the processing of the function block
Position	Axis position data	LREAL	-	0	Position data
Mode	Setting	BOOL	TRUE, FALSE	FALSE	Position mode TRUE: relative position (RELATIVE) FALSE: absolute position (ABSOLUTE)

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Setting	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the setting

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
	succeeded				operation succeeds.
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

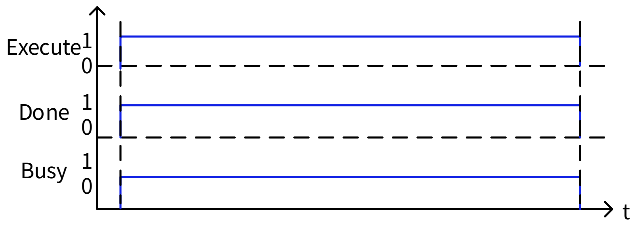
3. Function description

Axis position parameters are set by executing MC_SetPosition, without any movement caused but with coordinate system offset caused. This function block is triggered at the rising edge of Execute and it can be repeatedly executed without affecting each other.

4. Timing diagram

- The function block can be triggered only at the rising edge.
- When Done is TRUE, the setting operation is successful.
- When Busy is TRUE, the function block is being executed.

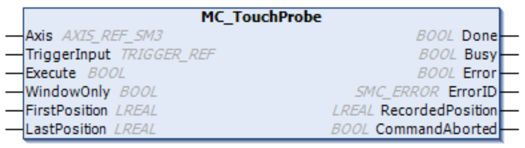
Timing description



7.1.25 MC_TouchProbe

MC_TouchProbe: used to save the axis position when a trigger event is raised.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation
MC_ TouchProbe	External locking enabling		MC_TouchProbe(Axis:=, TriggerInput:=, Execute:=, WindowOnly:=, FirstPosition:=, LastPosition:=, Done=>, Busy=>, Error=>, ErrorID=>, RecordedPosition=>, CommandAborted=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3
TriggerInput	Trigger signal	TRIGGER_REF	-	-	Association attributes such as trigger signal and attributes

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge will start the processing of the function block
WindowOnly	Trigger window	BOOL	TRUE, FALSE	FALSE	-
FirstPosition	Trigger start position	LREAL	-	0	Used to specify the start position for receiving trigger
LastPosition	Trigger end position	LREAL	-	0	Used to specify the end position for receiving trigger

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Setting succeeded	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the setting operation succeeds.
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output
RecordedPosition	Trigger recording position	LREAL	-	-	Position where the trigger occurs
CommandAbort	Instruction aborted	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is aborted

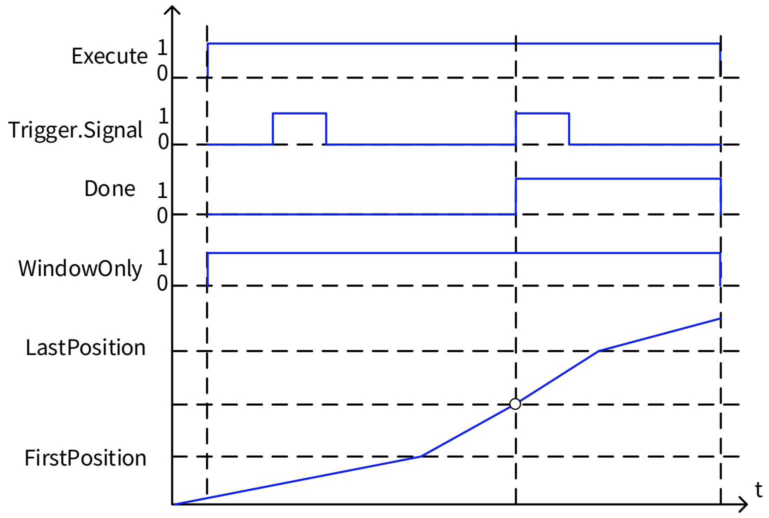
3. Function description

The actual position of the axis is recorded when TruggerInput of the MC_TouchProbe function block is triggered. When the rising edge executes drive latching, the latching signal collected by the drive is in the recorded position.

4. Timing diagram

- The function block can be triggered only at the rising edge.
- When Done is TRUE, the setting operation is successful.

Timing description

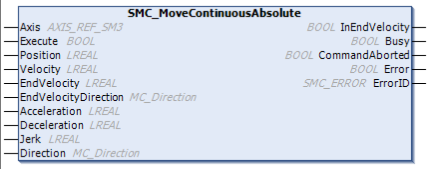


7.1.26 MC_MoveContinuousAbsolute

MC_MoveContinuousAbsolute: used to specify that an axis runs at the continuous absolute position (the unit is axis-depended). The absolute position is specified by Distance and the running end speed is specified by EndVelocity.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_MoveContinuousAbsolute	Axis absolute position continuous control instruction	 SMC_MoveContinuousAbsolute
		Axis AXIS_REF_SM3
		Execute BOOL
		Position LREAL
		Velocity LREAL
		EndVelocity LREAL
		EndVelocityDirection MC_Direction
		Acceleration LREAL
		Deceleration LREAL
		Jerk LREAL
		Direction MC_Direction
		BOOL InEndVelocity
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		SMC_ERROR ErrorID
			SMC_MoveContinuousAbsolute(
			Axis:=,
			Execute:=,
			Position:=,
			Velocity:=,
			EndVelocity:=,
			EndVelocityDirection:=,
			Acceleration:=,
			Deceleration:=,
			Jerk:=,
			Direction:=,
			InEndVelocity=>,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge trigger will start the processing of the function block
Distance	Relative position of motion	LREAL	Data range	0	The data is a relative position
Velocity	Running speed	LREAL	Data range	0	Max. speed at which the axis runs to reach the destination position
EndVelocity	Running end speed	LREAL	Data range	0	Running speed after instruction execution
EndVelocity-Direction	Direction of running at end speed	MC_Direction	positive, negative, current;	Current	Options: Positive, Negative, Current; Not allowed: Shortest, Fastest
Acceleration	Acceleration	LREAL	Data range	0	Acceleration when the speed increases
Deceleration	Deceleration	LREAL	Data range	0	Deceleration when the speed decreases
Direction	Running direction	shortest	Data range	shortest	For linear/straight axes: positive, negative; For rotary/circular axes: positive, negative, current, shortest, fastest

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
InEndVelocity	Instruction position reaching	BOOL	TRUE, FALSE	FALSE	It is set to TRUE after the position in the instruction is reached
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output
CommandAbort	Instruction aborted	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is aborted

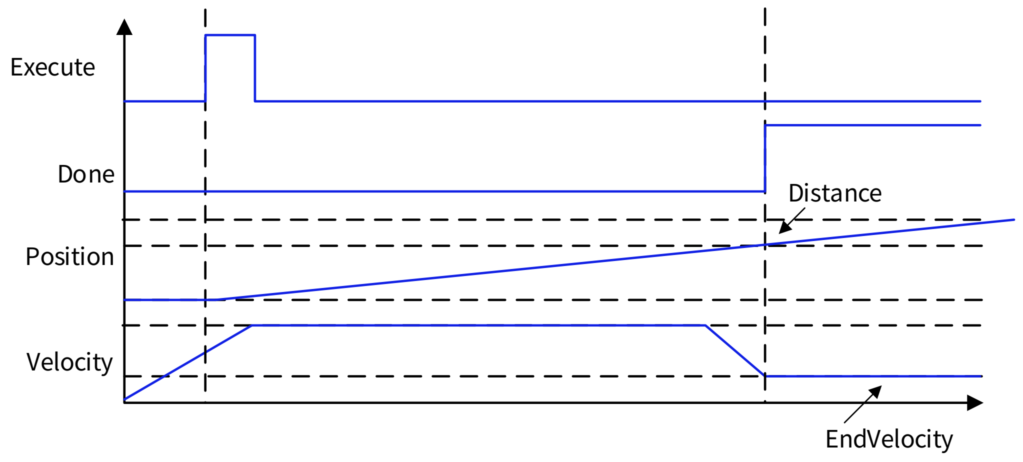
3. Function description

This function block is the axis absolute position instruction, in which Distance specifies the axis absolute position. The axis must be in the Standstill state before the function block execution and in the DiscreteMotion state during the execution. The axis status must be controlled throughout the complete running process. This function block is started at the rising edge of Execute. The startup instruction can repeatedly make the rising edge valid when the axis is in the DiscreteMotion state, which always refreshes the position. When Acceleration or Deceleration is 0, the instruction execution is abnormal, but the axis is in the DiscreteMotion state.

4. Timing diagram

- The function block can be executed only when the axis is in the Standstill state.
- The function block can be triggered only at the rising edge.
- When Busy is TRUE, the function block is being executed.

Timing description

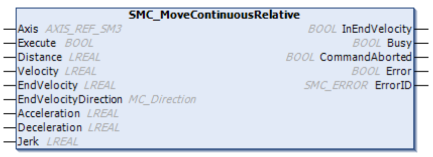


7.1.27 MC_MoveContinuousRelative

MC_MoveContinuousRelative: used to specify that an axis runs at the continuous relative position (the unit is axis-depended). The absolute position is specified by Distance and the running end speed is specified by EndVelocity

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_MoveContinuousRelative	Axis relative position continuous control instruction	 SMC_MoveContinuousRelative
		Axis AXIS_REF_SM3
		Execute BOOL
		Distance LREAL
		Velocity LREAL
		EndVelocity LREAL
		EndVelocityDirection MC_Direction
		Acceleration LREAL
		Deceleration LREAL
		Jerk LREAL
		BOOL InEndVelocity
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		SMC_ERROR ErrorID
			SMC_MoveContinuousRelative
			Axis:=,
			Execute:=,
			Distance:=,
			Velocity:=,
			EndVelocity:=,
			EndVelocityDirection:=,
			Acceleration:=,
			Deceleration:=,
			Jerk:=,
			InEndVelocity=>,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Execution condition	BOOL	TRUE, FALSE	FALSE	A rising edge will start the processing of the function block
Distance	Relative position of motion	LREAL	Data range	0	The data is a relative position
Velocity	Running speed	LREAL	Data range	0	Max. speed at which the axis runs to reach the destination position
EndVelocity	Running end speed	LREAL	Data range	0	Running speed after instruction execution
EndVelocity-Direction	Direction of running at end speed	MC_Direction	Positive, Negative, Current	Current	Options: Positive, Negative, Current Not allowed: Shortest, Fastest
Acceleration	Acceleration	LREAL	Data range	0	Acceleration when the speed increases
Deceleration	Deceleration	LREAL	Data range	0	Deceleration when the speed decreases
Direction	Running direction	Shortest	Data range	Shortest	For linear/straight axes: Positive, Negative For rotary/circular axes: Positive, Negative, Current, Shortest, Fastest

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
InEndVelocity	Instruction position reaching	BOOL	TRUE, FALSE	FALSE	It is set to TRUE after the position in the instruction is reached
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
CommandAbort	Instruction aborted	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is aborted
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

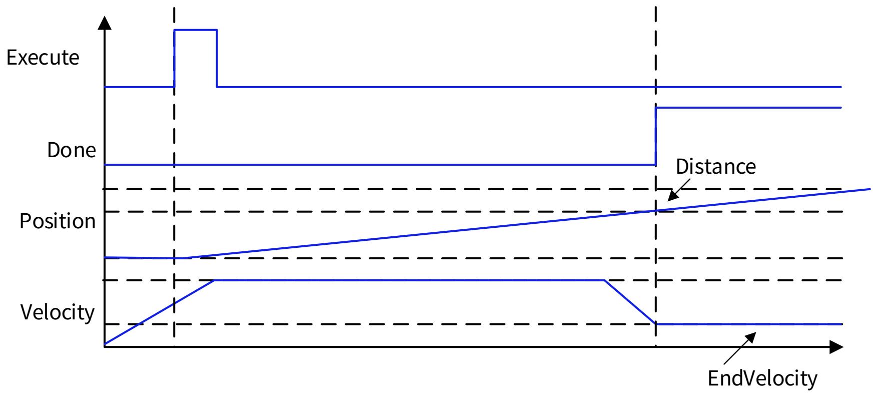
3. Function description

The axis must be in the Standstill state before the function block execution and in the DiscreteMotion state during the execution. Pay attention to the axis state during the execution to prevent other instructions from interrupting the instruction execution of the axis. This function block is started at the rising edge of Execute. The startup instruction can repeatedly make the rising edge valid when the axis is in the DiscreteMotion state, which always refreshes the position. When Acceleration or Deceleration is 0, the instruction execution is abnormal, but the axis is in the DiscreteMotion state.

4. Timing diagram

- The function block can be executed only when the axis is in the Standstill state.
- The function block can be triggered only at the rising edge.
- When Busy is TRUE, the function block is being executed.

Timing description

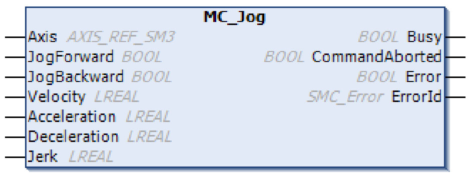


7.1.28 MC_Jog

MC_Jog: used to instruct an axis to jog at a specified speed.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_Jog	Axis jogging instruction	 MC_Jog
		Axis AXIS_REF_SM3
		JogForward BOOL
		JogBackward BOOL
		Velocity LREAL
		Acceleration LREAL
		Deceleration LREAL
		Jerk LREAL
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		SMC_Error ErrorId
			MC_Jog(
			Axis:=,
			JogForward:=,
			JogBackward:=,
			Velocity:=,
			Acceleration:=,
			Deceleration:=,
			Jerk:=,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorId=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
JogForward	Valid at forward jogging	BOOL	TRUE, FALSE	FALSE	If it is TRUE, the axis moves forward. If it is FALSE, the axis stops moving forward

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
JogBackward	Valid at backward jogging	BOOL	TRUE, FALSE	FALSE	If it is TRUE, the axis moves backward. If it is FALSE, the axis stops moving backward
Velocity	Target velocity	LREAL	Positive or 0	0	Specified target speed. Unit: [Instruction unit/s]
Acceleration	Acceleration	LREAL	Positive or 0	0	Specified acceleration. Unit: [Instruction unit/s]
Deceleration	Deceleration	LREAL	Positive or 0	0	Specified deceleration. Unit: [Instruction unit/s]
Jerk	Jerk	LREAL	Data range	0	Slope change value of the curve acceleration or deceleration

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
CommandAborted	Instruction aborted	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is aborted
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorId	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

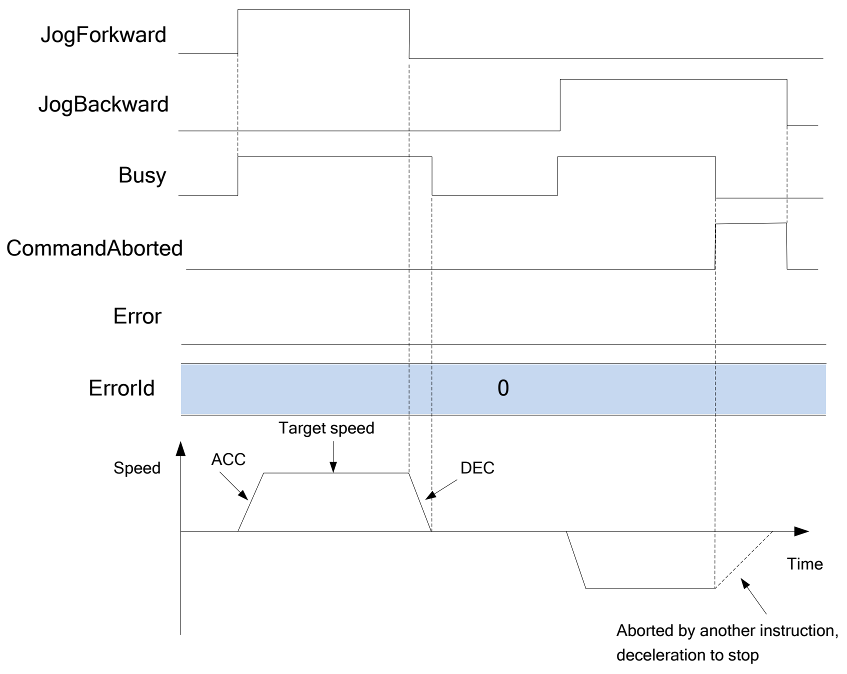
The function block is used to instruct the axis to jog at the target speed specified by Velocity. When the axis needs to run forward, set JogForward to TRUE; when the axis needs to run backward, set JogBackward to TRUE. When both JogForward and JogBackward are set to TRUE at the same time, the axis does not move. If the speed value in MC_Jog exceeds the max. jogging speed in the axis parameters, the axis moves at the max. jogging speed

4. Timing diagram

When JogForward or JogBackward is set to TRUE, the value of Busy changes to TRUE; when the falling edge of JogForward or JogBackward starts deceleration until the axis is stopped, the value of Busy changes to FALSE.

If another instruction is used to terminate the execution of this function block, the value of CommandAborted changes to TRUE, and the value of Busy changes to FALSE.

Timing description

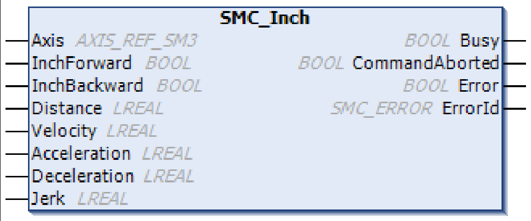


7.1.29 MC_Inch

MC_Inch: used to cause a gradual motion on an axis, which is carried out step by step.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_Inch	Axis relative positioning instruction	 SMC_Inch
		Axis AXIS_REF_SM3
		InchForward BOOL
		InchBackward BOOL
		Distance LREAL
		Velocity LREAL
		Acceleration LREAL
		Deceleration LREAL
		Jerk LREAL
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		SMC_ERROR ErrorId
			SMC_Inch(
			Axis:=,
			InchForward:=,
			InchBackward:=,
			Distance:=,
			Velocity:=,
			Acceleration:=,
			Deceleration:=,
			Jerk:=,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorId=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
InchForward	Forward inching	BOOL	TRUE, FALSE	FALSE	If InchForward is TRUE, the axis runs at the given speed in the forward direction until it reaches the destination. The input must be set to FALSE and then TRUE to restart the running. If InchForward is set to FALSE before the destination is reached, the axis decelerates to 0 at once, and Busy is set to FALSE. If InchBackward is set to TRUE in simulation mode, the axis does not move.
InchBackward	Backward inching	BOOL	TRUE, FALSE	FALSE	If InchBackward is TRUE, the axis runs at the given speed in the reverse direction until it reaches the destination. The input must be set to FALSE and then TRUE to restart the running. **Note:** If both InchBackward and InchForward are set to TRUE at the same time, the axis does not move.
Distance	Moving distance	LREAL	Data range	0	This data is the moving distance
Velocity	Running speed	LREAL	Data range	0	Max. speed at which the axis runs to reach the destination position
Acceleration	Acceleration	LREAL	Data range	0	Acceleration when the speed increases
Deceleration	Deceleration	LREAL	Data range	0	Deceleration when the speed decreases

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
CommandAborted	Instruction aborted	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is aborted
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
ErrorId	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

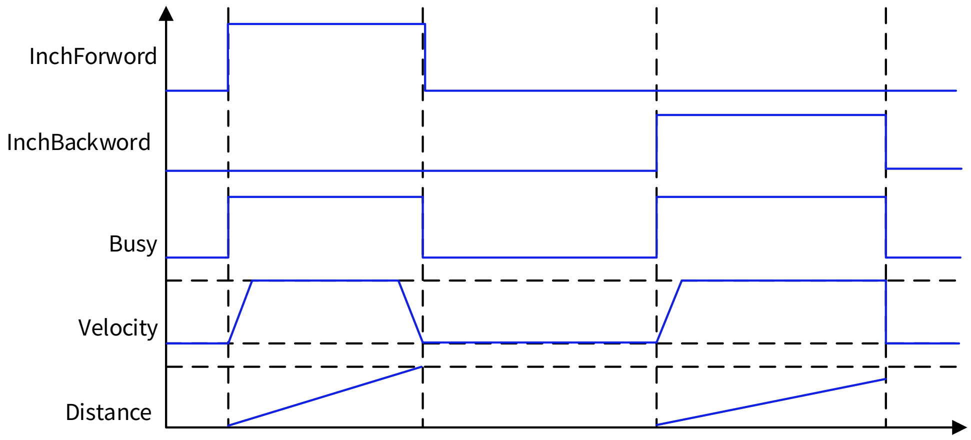
3. Function description

The axis must be in the Standstill state before the function block execution and in the DiscreteMotion state during the execution. Pay attention to the axis state during the execution to prevent other instructions from interrupting the instruction execution of the axis. When Acceleration or Deceleration is 0, the instruction execution is abnormal, but the axis is in the DiscreteMotion state.

4. Timing diagram

- InchForward and InchBackward must be set to TRUE or FALSE.
- When Busy is TRUE, the function block is being executed.

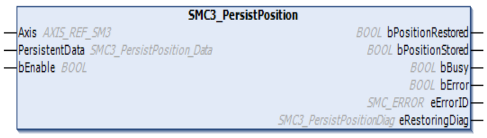
Timing description



7.1.30 SMC3_PersistPosition

SMC3_PersistPosition: used to persist the axis position of a multi-turn absolute encoder with real axis. (The controller that is restarted due to power failure uses the position recorded before the power failure.) If the servo motor uses an absolute encoder, use this function block.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation
SMC3_PersisitPosition	Axis positi on persis ting instru ction	 SMC3_PersistPosition Axis AXIS_REF_SM3 PersistentData SMC3_PersistPosition_Data bEnable BOOL BOOL bPositionRestored BOOL bPositionStored BOOL bBusy BOOL bError SMC_ERROR eErrorID SMC3_PersistPositionDiag eRestoringDiag	SMC3_PersistPosition(Axis:=, PersistentData:=, bEnable:=, bPositionRestored=>, bPositionStored=>, bBusy=>, bError=>, eErrorID=>, eRestoringDiag=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3
PersistentData	Data to persist	SMC3_PersistPosition_Data	-	-	Structure of position data stored at power failure

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Enabling	BOOL	TRUE, FALSE	FALSE	TRUE indicates executing the function block, while FALSE indicates not executing the function block

Output variables

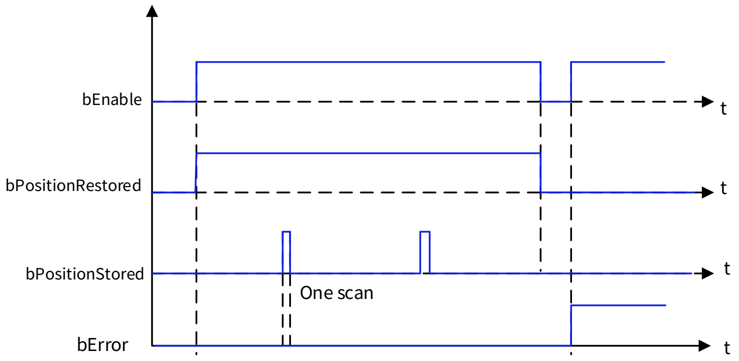
Output Variable	Name	Data Type	Valid Range	Initial Value	Description
bPositionRestored	Position restored	BOOL	TRUE, FALSE	FALSE	TRUE indicates the position data is restored after the axis restart
bPositionStored	Position stored	BOOL	TRUE, FALSE	FALSE	TRUE indicates the position data is stored after the function block is called
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output
eRestoringDiag	Restoring diagnosis	SMC3_Persist-PositionDiag	-	-	Diagnosis information for position restoring SMC3_PPD_RESTORING_OK: Position restored successfully SMC3_PPD_AXIS_PROP_CHANGED: Failed to restore the position due to axis parameter changes SMC3_PPD_DATA_STORED_DURING_WRITING: The function block copies data from the axis data structure but not from PersistentData. Possible causes: Asynchronous persistent variables, and controller crash.

3. Function description

When the PLC is restarted and bEnable is TRUE, bPositionRestroed is TRUE.

4. Timing diagram

When Busy is TRUE, the function block is being executed.

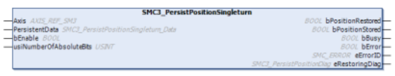


7.1.31 SMC3_PersistPositionSingleturn

SMC3_PersistPositionSingleturn: used to persist the axis position of a single-turn absolute encoder with real axis (The controller that is restarted due to power failure uses the position recorded before the power failure.). If the servo motor uses a single-turn absolute encoder, use this function block.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

SMC3_PersisitPositionSingleturn	Axis position persisting instruction	 SMC3_PersistPositionSingleturn
		Axis AXIS_REF_SM3
		PersistentData SMC3_PersistPositionSingleturn_Data
		bEnable BOOL
		usiNumberOfAbsoluteBits USINT
		BOOL bPositionRestored
		BOOL bPositionStored
		BOOL bBusy
		BOOL bError
		SMC_ERROR eErrorID
		SMC3_PersistPositionDiag eRestoringDiag
			SMC3_PersistPositionSingleturn(
			Axis:=,
			PersistentData:=,
			bEnable:=,
			usiNumberOfAbsoluteBits:=,
			bPositionRestored=>,
			bPositionStored=>,
			bBusy=>,
			bError=>,
			eErrorID=>,
			eRestoringDiag=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3
PersistentData	Data to persist	SMC3_PersistPosition_Data	-	-	Structure of position data stored at power failure

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Enabling	BOOL	TRUE, FALSE	FALSE	TRUE indicates executing the function block, while FALSE indicates not executing the function block

Output variables

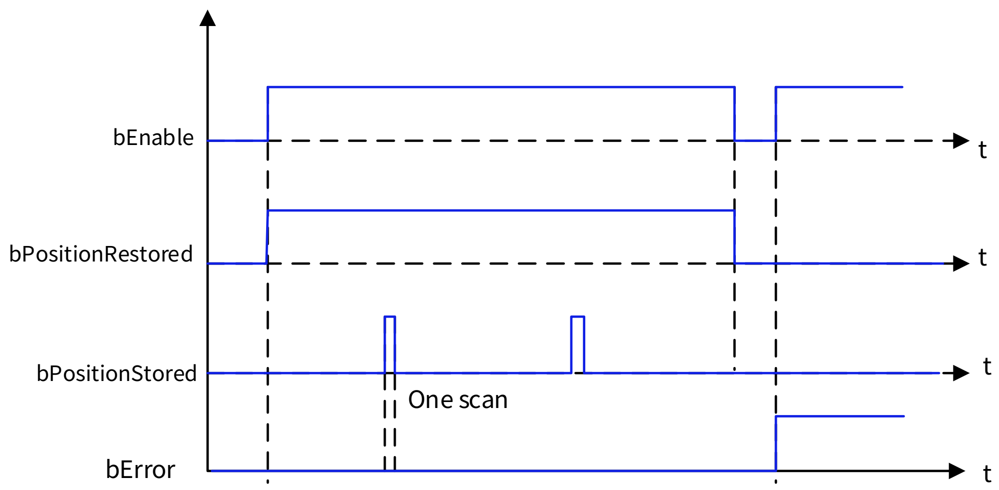
Output Variable	Name	Data Type	Valid Range	Initial Value	Description
bPositionRestored	Position restored	BOOL	TRUE, FALSE	FALSE	TRUE indicates the position data is restored after the axis restart
bPositionStored	Position stored	BOOL	TRUE, FALSE	FALSE	TRUE indicates the position data is stored after the function block is called
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output
eRestoringDiag	Restoring diagnosis	SMC3_Persist-PositionDiag	-	-	Diagnosis information for position restoring SMC3_PPD_RESTORING_OK: Position restored successfully; SMC3_PPD_AXIS _PROP_CHANGED: Failed to restore the position due to axis parameter changes; SMC3_PPD _DATA_STORED_DURING_WRI TING: The function block copies data from the axis data structure but not from PersistentData. Possible causes: Asynchronous persistent variables, and controller crash.

3. Function description

When the PLC is restarted and bEnable is TRUE, bPositionRestroed is TRUE.

4. Timing diagram

When Busy is TRUE, the function block is being executed.

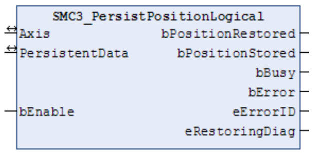


7.1.32 SMC3_PersistPositionLogical

SMC3_PersistPositionSingleturn: used to persist the axis position of a single-turn absolute encoder with real axis (The controller that is restarted due to power failure uses the position recorded before the power failure.). If the servo motor uses a single-turn absolute encoder, use this function block.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

SMC3_PersisitPositionLogical	Axis position persisting instruction	 SMC3_PersistPositionLogical
		Axis
		PersistentData
		bEnable
		bPositionRestored
		bPositionStored
		bBusy
		bError
		eErrorID
		eRestoringDiag
			SMC3_PersistPositionLogical(
			Axis:=,
			PersistentData:=,
			bEnable:=,
			bPositionRestored=>,
			bPositionStored=>,
			bBusy=>,
			bError=>,
			eErrorID=>,
			eRestoringDiag=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3
PersistentData	Data to persist	SMC3_Persist Position_Data	-	-	Structure of position data stored at power failure

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Enabling	BOOL	TRUE, FALSE	FALSE	TRUE indicates executing the function block, while FALSE indicates not executing the function block

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
bPositionRestored	Position restored	BOOL	TRUE, FALSE	FALSE	TRUE indicates the position data is restored after the axis restart
bPositionStored	Position stored	BOOL	TRUE, FALSE	FALSE	TRUE indicates the position data is stored after the function block is called
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the axis instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs

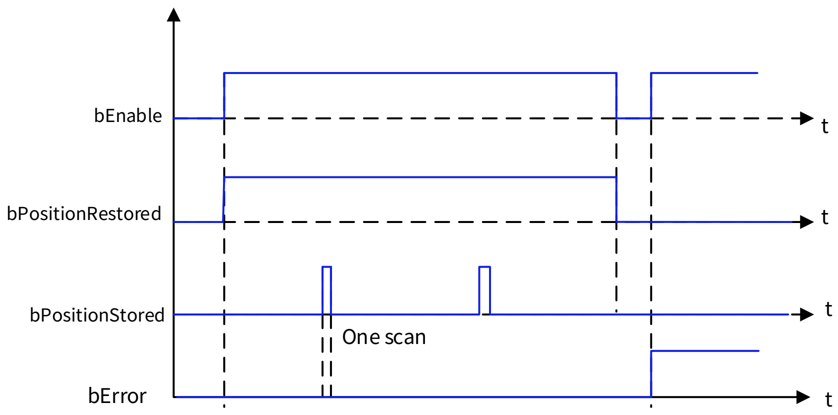
Output Variable	Name	Data Type	Valid Range	Initial Value	Description
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output
eRestoringDiag	Restoring diagnosis	SMC3_Persist-PositionDiag	-	-	Diagnosis information for position restoring SMC3_PPD_RESTORING_OK: Position restored successfully SMC3_PPD_AXIS_PROP_CHANGED: Failed to restore the position due to axis parameter changes SMC3_PPD_DATA_STORED_DURING_WRITING: The function block copies data from the axis data structure but not from PersistentData. Possible causes: Asynchronous persistent variables, and controller crash.

3. Function description

When the PLC is restarted and bEnable is TRUE, bPositionRestroed is TRUE.

4. Timing diagram

When Busy is TRUE, the function block is being executed.

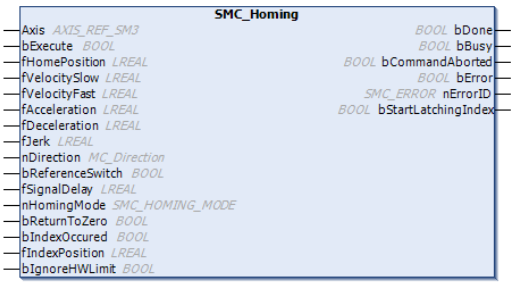


7.1.33 SMC_Homing

SMC_Homing: axis home instruction, different from MC_Home. MC_Home specifies the homing mode controlled by the servo controller, while SMC_Homing specifies the homing mode controlled by the PLC.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

SMC_Homing	Axis homing instruction	 SMC_Homing
		Axis AXIS_REF_SM3
		bExecute BOOL
		fHomePosition LREAL
		fVelocitySlow LREAL
		fVelocityFast LREAL
		fAcceleration LREAL
		fDeceleration LREAL
		fJerk LREAL
		nDirection MC_Direction
		bReferenceSwitch BOOL
		fSignalDelay LREAL
		nHomingMode SMC_HOMING_MODE
		bReturnToZero BOOL
		bIndexOccured BOOL
		fIndexPosition LREAL
		bIgnoreHWLimit BOOL
		BOOL bDone
		BOOL bBusy
		BOOL bCommandAborted
		BOOL bError
		SMC_ERROR nErrorID
		BOOL bStartLatchingIndex
			SMC_Homing(
			Axis:=,
			bExecute:=,
			fHomePosition:=,
			fVelocitySlow:=,
			fVelocityFast:=,
			fAcceleration:=,
			fDeceleration:=,
			fJerk:=,
			nDirection:=,
			bReferenceSwitch:=,
			fSignalDelay:=,
			nHomingMode:=,
			bReturnToZero:=,
			bIndexOccured:=,
			fIndexPosition:=,
			bIgnoreHWLimit:=,
			bDone=>,
			bBusy=>,
			bCommandAborted=>,
			bError=>,
			nErrorID=>,
			bStartLatchingIndex=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
bExecute	Executing	BOOL	TRUE, FALSE	FALSE	TRUE indicates executing the function block, while FALSE indicates not executing the function block
fHomePosition	Home position	LREAL	-	0	Home position after zeroing, using the unit after user calibration
fVelocitySlow	Low speed	LREAL	-	0	Used to drive out of the reference switch
fVelocityFast	High speed	LREAL	-	0	Used until the reference switch is found
fAcceleration	Acceleration	LREAL	-	0	Acceleration setting
fDeceleration	Deceleration	LREAL	-	0	Deceleration setting

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
fJerk	Jerk	LREAL	-	0	Jerk setting
nDirection	Homing direction	MC_DIRECTION	-	Negative	Homing start direction
bReferenceSwitch	Reference switch	BOOL	TRUE, FALSE	FALSE	Reference switch status. TRUE: The reference switch is open. FALSE: The reference switch is closed
fSignalDelay	Delay	LREAL	-	0	Reference switch transmission time, used to compensate for the deadzone time. Unit: second.
nHomingMode	Homing mode	SMC_HOMING_MODE	FAST_BSLOW_S_STOP, FAST_BSLOW_STOP_S, FAST_BSLOW_I_S_STOP, FAST_SLOW_S_STOP, FAST_SLOW_STOP_S, FAST_SLOW_I_S_STOP	0	Homing mode
bReturnTozero	Returning to zero	BOOL	TRUE, FALSE	FALSE	TRUE: The axis moves to zero after homing (**Note:** If fHomePosition=10, the axis position is 10 after homing, and when bReturnTozero is TRUE, the axis reversely moves by 10 units to zero.)
bIndexOccured	Pulse signal	BOOL	TRUE, FALSE	FALSE	TRUE: Index pulse is detected. It is valid at the homing modes FAST_BSLOW_I_S_STOP and FAST_SLOW_I_S_STOP
fIndexPosition	Index position	LREAL	-	0	Position where the index occurs
bIgnoreHWLimit	Ignoring hardware position limit	BOOL	TRUE, FALSE	FALSE	TRUE: The hardware position limit switch is disabled. If the same physical switch is used as both the hardware position limit switch and the reference switch, hardware control is set to FALSE.

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
bDone	Setting succeeded	BOOL	TRUE, FALSE	FALSE	TRUE, homing completed
bBusy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	TRUE, the function block is being executed
bCommandAborted	Instruction aborted	BOOL	TRUE, FALSE	FALSE	TRUE, the function block is aborted by other action instructions
bError	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
nErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output
bStartLatchingIndex	Start latching iIndex	BOOL	TRUE, FALSE	FALSE	Generated by "bIndexOccured" and "fIndexPosition"

The homing modes are described as follows:

Mode	Type	Initial Value	Description
FAST_BSLOW_S_STOP	SMC_HOMING_MODE	0	The axis follows the set direction to the home switch at a high speed, and leaves the home switch at a low speed in the reverse direction after touching the home switch. After leaving, the controller executes MC_setPosition to set the present position to the setting of fHomePosition, and then executes MC_stop
FAST_BSLOW_STOP_S	SMC_HOMING_MOD	1	The axis follows the set direction to the home switch at a high speed, and leaves the home switch at a low speed in the reverse direction after touching the home switch. After leaving, the controller executes MC_stop to stop the axis, and then executes MC_setPosition to set the present position to the setting of fHomePosition
FAST_BSLOW_I_S_STOP	SMC_HOMING_MOD	2	The axis follows the set direction to the home switch at a high speed, and leaves the home switch at a low speed in the reverse direction after touching the home switch. When receiving the bIndexOccured signal, the controller executes MC_setPosition and then MC_stop
FAST_SLOW_S_STOP	SMC_HOMING_MOD	4	The axis follows the set direction to the home switch at a high speed, and leaves the home switch at a low speed after touching the home switch. After leaving, the controller executes MC_setPosition to set the present position to the setting of

Mode	Type	Initial Value	Description
			fHomePosition, and then executes MC_stop
FAST_SLOW_STOP_S	FAST_SLOW_STOP_S SMC_HOMING_MOD	5	The axis follows the set direction to the home switch at a high speed, and leaves the home switch at a low speed after touching the home switch. After leaving, the controller executes MC_stop to stop the axis, and then executes MC_setPosition to set the present position to the setting of fHomePosition
FAST_SLOW_I_S_STOP	SMC_HOMING_MOD	6	The axis follows the set direction to the home switch at a high speed, and leaves the home switch at a low speed in the reverse direction after touching the home switch. When receiving the bIndexOccured signal, the controller executes MC_setPosition and then MC_stop

3. Function description

After SMC_HOMING is started at the rising edge of bExecute, the axis moves at the speed specified by fVelocityFast in the direction specified by nDirection, which does not end until bReferenceSwitch=FALSE. The axis slowly stops and leaves the reference switch at the speed specified by fVelocitySlow in the reverse direction. When bReferenceSwitch=TRUE, homing is completed

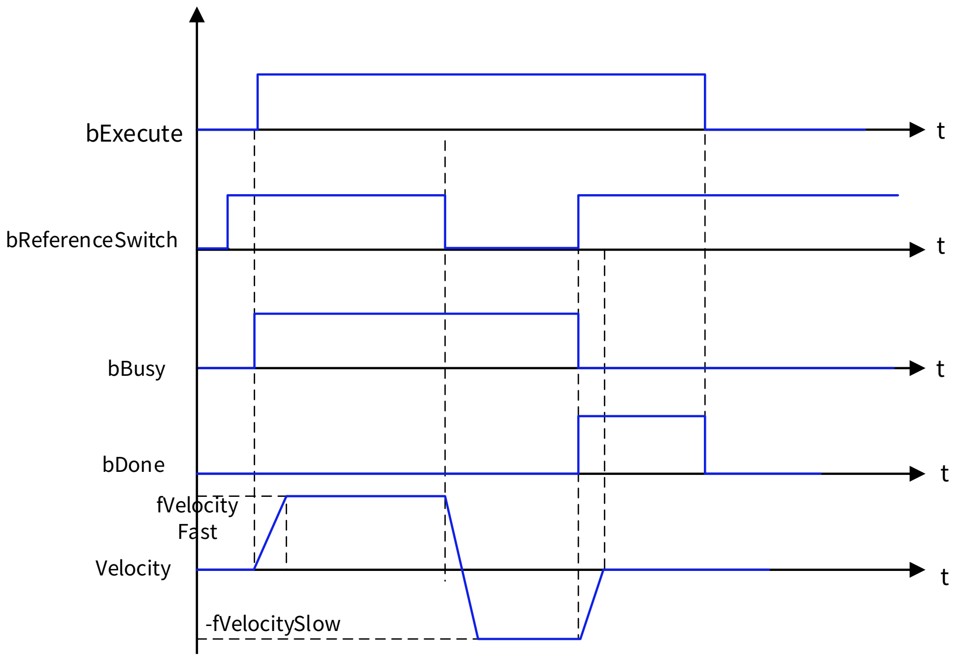
After the homing instruction is enabled, the status change sequence of bReferenceSwitch is ON→OFF→ON, the homing is completed at the rising edge of OFF→ON, and the reference position is set. Reference position = fHomePostion + [(fSignalDelay*1000 + 1 DC cycle)/1000] * fVelocitySlow, which actually compensates for the bReferenceSwitch sampling delay and one-communication-cycle displacement delay.

If bReturnToZero=TRUE, the reference position is set to {fHomePostion + [(fSignalDelay*1000 + 1 DC cycle)/1000] * fVelocitySlow} at the rising edge of OFF→ON of bReferenceSwitch, the axis moves to zero at the speed specified by fVelocityFast.

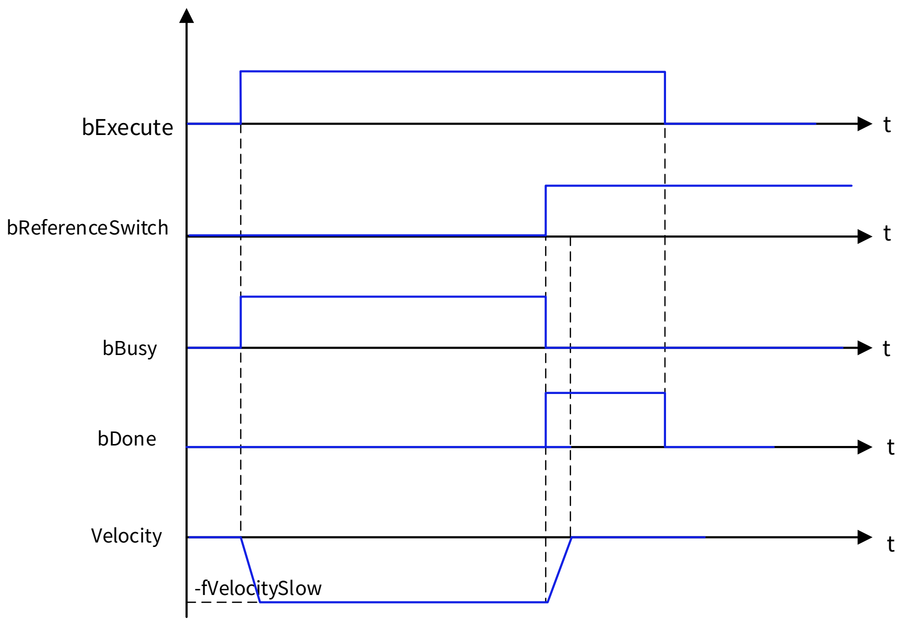
Note: After the Done signal is completed, the axis position is set to fHomePosition. The setting time is related to nHomingMode.

4. Timing diagram

- The instruction is executed When bReferenceSwitch=TRUE:



- The instruction is executed When bReferenceSwitch=FALSE:

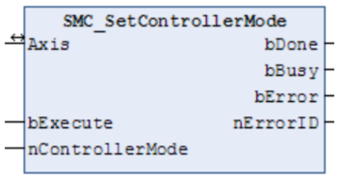


7.1.34 SMC_SetControllerMode

SMC_SetControllerMode: used to set the current running mode of the servo, which is cyclic synchronous position control by default. For the control mode-related settings, refer to the servo manual. For DA200, the position mode is 8, the speed mode is 9, and the torque mode is 10.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

SMC_SetControllerMode	Axis control mode setting instruction	 SMC_SetControllerMode
		Axis
		bDone
		bBusy
		bError
		bExecute
		nErrorID
		nControllerMode
			SMC_SetControllerMode(
			Axis:=,
			bExecute:=,
			nControllerMode:=,
			bDone=>,

Instruction	Name	Graphical Representation	ST Representation

			bBusy=>,
			bError=>,
			nErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
bExecute	Executing	BOOL	TRUE, FALSE	FALSE	TRUE indicates executing the function block, while FALSE indicates not executing the function block
nControllerMode	Control mode	SMC_Controller_MODE	-	SMC_Position	Axis control mode 1: Torque control mode, SMC_torque 2: Speed control mode, SMC_Velocity 3: Position control mode, SMC_Position 4: Current control mode, SMC_Current

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
bDone	Setting succeeded	BOOL	TRUE, FALSE	FALSE	TRUE, homing completed
bBusy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	TRUE, the function block is being executed
bError	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
nErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

Preconditions for using this function block:

1. The axis must meet these control conditions, for example, the virtual axis cannot use this function block.

2. The synchronization cycle supported by each mode must be consistent.

3. The axis must NOT be in the state "errorstop", "stopping", or "homing” when this instruction is executed; otherwise, an error will be reported.

4. If the axis still does not change to the set control mode after the instruction executes 1000 task cycles, the instruction reports an error and bError changes from FALSE to TRUE.

5. When switching from a low level to a high level control mode (torque→velocity, torque→position, velocity→position), the function block calculates the set value of the high level signal. For example, when switching from torque mode to position mode, the function block superimposes an expected position distance (calculated by the current actual speed and the time offset in the task cycle) based on the current actual position of the axis to compensate for the time lag between the actual and set values.

6. After the instruction is executed, when the actual control mode of the axis is changed to the set control mode, the bDone signal is triggered. The axis will still run during the time between the instruction triggering and the bDone signal triggering, and during this time, the function block will calculate the appropriate set value according to the set control mode. However, if the bDone signal is triggered but there is no other control instruction to continue to set the value for the axis, the axis will stop immediately and report an error. Therefore, the rising edge of the bDone signal is required to trigger MC_Halt, MC_MoveVelocity, MC_MoveAbsolute, and other instructions to smoothly control the axis.

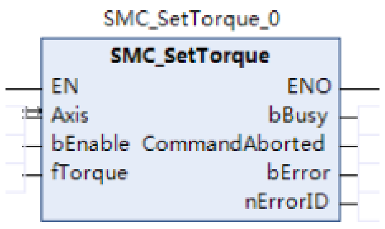
Note: When the control mode is switched to torque control, a torque control instruction (such as SMC_SetTorque) is required to smoothly control the axis.

7.1.35 SMC_SetTorque

SMC_SetTorque: used to set the torque of an axis (valid in torque control mode).

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

SMC_SetTorque	Torque setting instruction	 SMC_SetTorque_0
		SMC_SetTorque
		EN
		ENO
		Axis
		bBusy
		bEnable
		CommandAborted
		fTorque
		bError
		nErrorID
			SMC_SetTorque(
			Axis:=,
			bEnable:=,
			fTorque:=,
			bBusy=>,
			bError=>,
			nErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis	Axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
bEnable	Enabling	BOOL	TRUE, FALSE	FALSE	TRUE indicates executing the function block, while FALSE indicates not executing the function block
fTorque	Set torque	LREAL	-	0	The unit is 0.1% (Axis.fFactorTor:=1;)

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Busy	Instruction being executed	BOOL	TRUE, FALSE	FALSE	TRUE, the function block is being executed
bError	Error flag	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when an exception occurs
nErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

This function block is started at the rising edge of bEnable. If there is no error, bBusy is TURE. This instruction is only used to set the torque value of an axis and is not for torque control. The axis control mode is valid in the torque control mode.

The torque setting instruction can only be run in the synchronous torque mode. When enabling this instruction, you must first use MC_SetControlMode to switch the control mode to

the synchronous torque mode.

The actual torque of the drive is limited by the maximum positive/negative torque set in the configuration parameters.

To stop the execution of this instruction, you can use the MC_Stop (forced stop) instruction. After stopping, the drive switches to the synchronous position mode.

4. Error description

If the axis reports an error, Error outputs TRUE; if the axis input is valid, Error outputs TRUE.

If an axis control mode error is reported, Error is TRUE, and the error code is SMC_ST_WRONG_CONTROLLER_MODE.

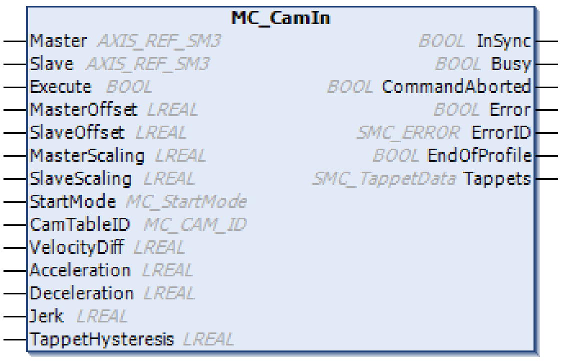
7.2 Master-slave Axis Instructions

7.2.1 MC_CamIn

MC_CamIn: used to designate a cam table to start the execution of the e-cam actions, and specify the offset value, scaling ratio and working mode of the master and slave axes according to application requirements.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_Camin	Cam action start instruction	 MC_CamIn
		Master AXIS_REF_SM3
		Slave AXIS_REF_SM3
		Execute BOOL
		MasterOffset LREAL
		SlaveOffset LREAL
		MasterScaling LREAL
		SlaveScaling LREAL
		StartMode MC_StartMode
		CamTableID MC_CAM_ID
		VelocityDiff LREAL
		Acceleration LREAL
		Deceleration LREAL
		Jerk LREAL
		TappetHysteresis LREAL
		BOOL InSync
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		SMC_ERROR ErrorID
		BOOL EndOfProfile
		SMC_TappetData Tappets
			MC_CamIn(
			Master:=,
			Slave:=,
			Execute:=,
			MasterOffset:=,
			SlaveOffset:=,
			MasterScaling:=,
			SlaveScaling:=,
			StartMode:=,
			CamTableID:=,
			VelocityDiff:=,
			Acceleration:=,
			Deceleration:=,

Instruction	Name	Graphical Representation	ST Representation

			Jerk:=,
			TappetHysteresis:=,
			InSync=>,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorID=>,
			EndOfProfile=>,
			Tappets=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Master	Master axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3
Slave	Slave axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Note: The master axis and the slave axis must be different axes. Otherwise, errors may be reported.

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Cam function entry	BOOL	TRUE, FALSE	FALSE	The rising edge starts the execution of the function block
MasterOffset	Master axis offset	LREAL	Negative, Positive, or 0	0	The phase of the master axis is moved by the specified offset value
SlaveOffset	Slave axis offset	LREAL	Negative, Positive, or 0	0	The phase of the slave axis is moved by the specified offset value
MasterScaling	Pre-compiling scaling factor of the master axis	LREAL	>0.0	1	The phase of the master axis is scaled up or down by the specified value
SlaveScaling	Pre-compiling scaling factor of the slave axis	LREAL	>0.0	1	The phase of the slave axis is scaled up or down by the specified value
StartMode	Output mode of the slave axis in relative to cam	MC_StartMode	-	absolute	0: Absolute position 1: Relative position 2: ramp_in (ramp switching in) 3: ramp_in_pos (forward ramp switching in) 4: ramp_in_neg (reverse ramp switching in)

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
CamTableID	Table ID	MC_CAM_ID	-	-	Used to define a cam table, in conjunction with output points of MC_CamTableSelect
VelocityDiff	Speed	LREAL	-	-	Max. speed, different from ramp_in
Acceleration	Acceleration	LREAL	-	-	Acceleration for ramp_in
Deceleration	Deceleration	LREAL	-	-	Deceleration for ramp_in
Jerk	Jerk	LREAL	-	-	Jerk for ramp_in
TappetHysteresis	Tappet damping	LREAL	-	-	Damping factor of the tappet

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
InSync	Cam taking effect	BOOL	TRUE, FALSE	FALSE	After the master axis and the slave axis establish a cam relationship, InSync is set. When the execution condition of the instruction is OFF, InSync is reset.
Busy	Synchronous running	BOOL	TRUE, FALSE	FALSE	When the rising edge of Execute is detected, it is set to TRUE, which indicates that the cam relationship is being coupled and you need to use Cam_out for reset. The instruction execution condition reset cannot reset the status.
CommandAborted	Instruction aborted	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the slave axis is aborted by another control instruction
Error	Error flag	BOOL	TRUE, FALSE	FALSE	Error is set when an error is detected. Error is reset when the instruction execution condition is OFF
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output
EndOfProfile	Profile completed	BOOL	TRUE, FALSE	FALSE	If Periodic is 0 (acyclic) when MC_CamTableSelect is executed, EndOfProfile is set after the cam profile is completed for one time, and EndOfProfile is reset when the instruction execution condition is OFF.

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Tappets	Tappet table	SMC_TappetData	-	-	Associated cam tappet, which can be read by MC_GetTappetValue

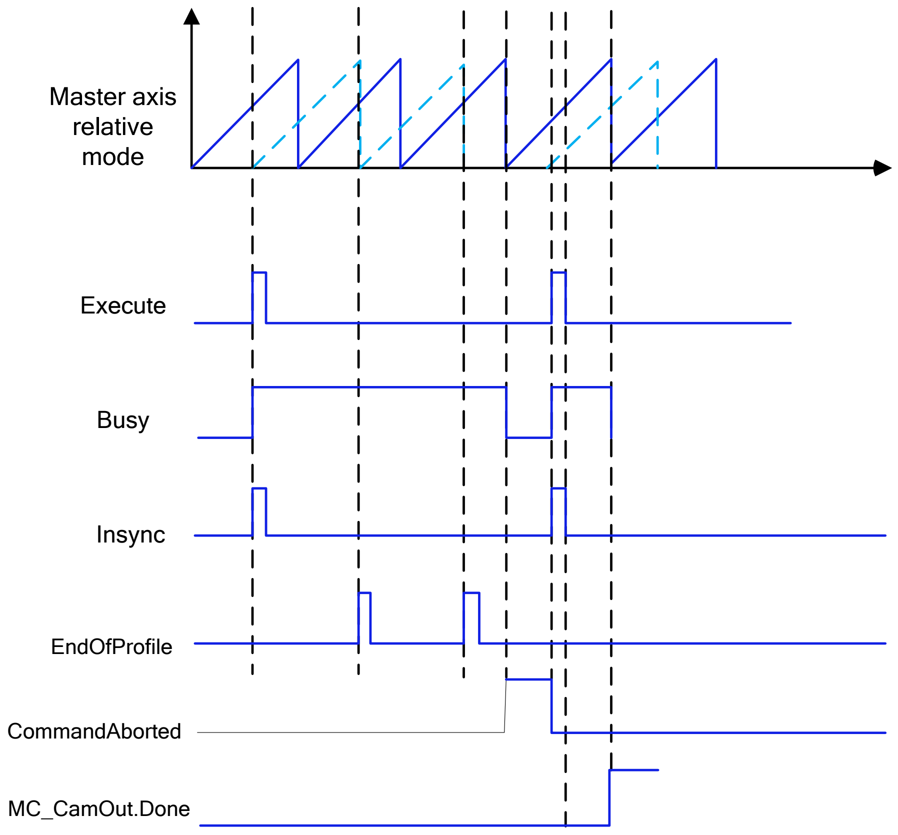
3. Function description

Under the condition that correct cam tables are selected and axes do not encounter errors, the rising edge of Execute triggers the function block. In a cam motion system, to call a cam profile, call the MC_CamTableSelect instruction to select the corresponding cam table, and then execute MC_CamIn; to change the cam profile, call the MC_CamTableSelect instruction to reselect a cam table. You need to use the Camout instruction to decouple the cam relationship between the master axis and slave axis. When the instruction is being executed, if another instruction is applied to the slave axis at this time, the cam relationship between the master axis and slave axis is decoupled, and Command-Aborted outputs TRUE.

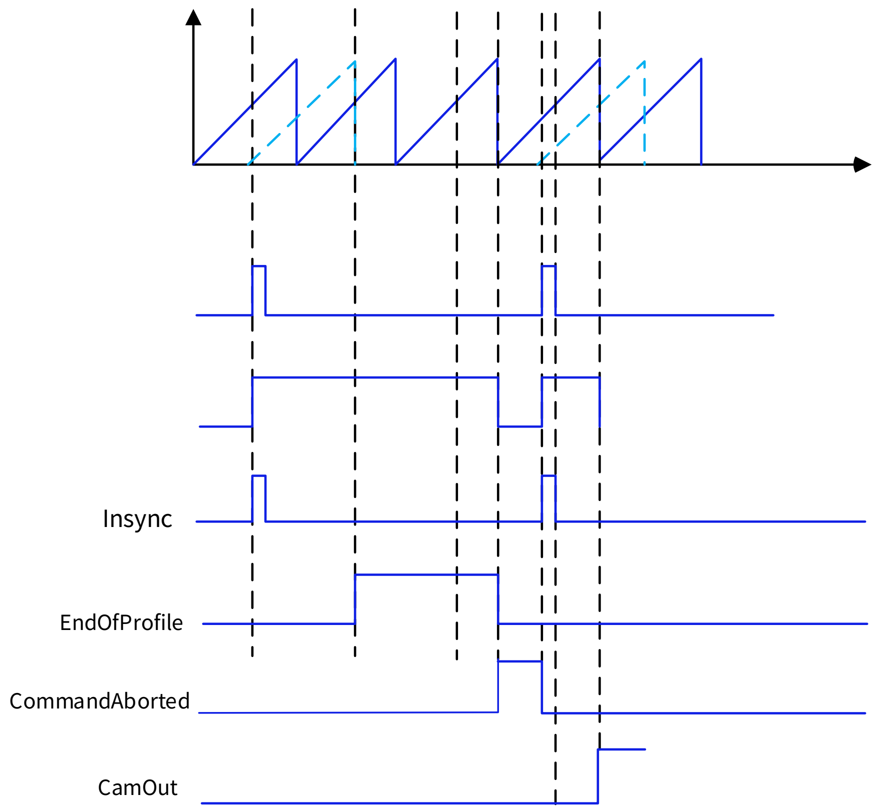
4. Timing diagram

Cyclic mode (MC_CamTableSelect.Periodic is TRUE):

Note: The MC_Camout instruction only decouples the cam relationship between the master axis and slave axis. If the slave axis speed is not 0 during the decoupling, the slave axis does not automatically decelerate to 0, which indicates using MC_STOP is required.



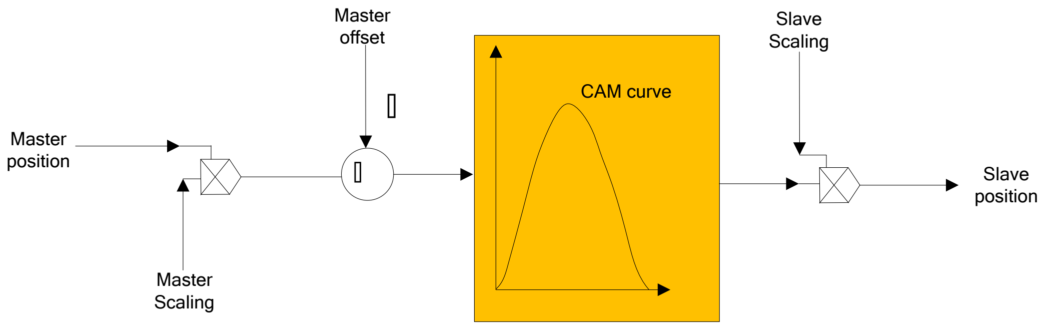
Non-cyclic mode (MC_CamTableSelect.Periodic is FALSE):



5. Function block description

The instruction can be started in any state during master axis stop, position control, speed control, and synchronization control.

The calculation method of the engaging points in the cam profile is as follows:



The following formula is obtained according to the figure:

Position_Slave = SlaveScaling*CAM(MasterScaling*MasterPosition + MasterOffset) + SlaveOffset

The positions of the master and slave axes in the formula do not represent the actual physical axis positions, but the positions of the master and slave axes related to the cam function curve.

The relationship between the master/slave axis positions and the master/slave real axis position is described in detail.

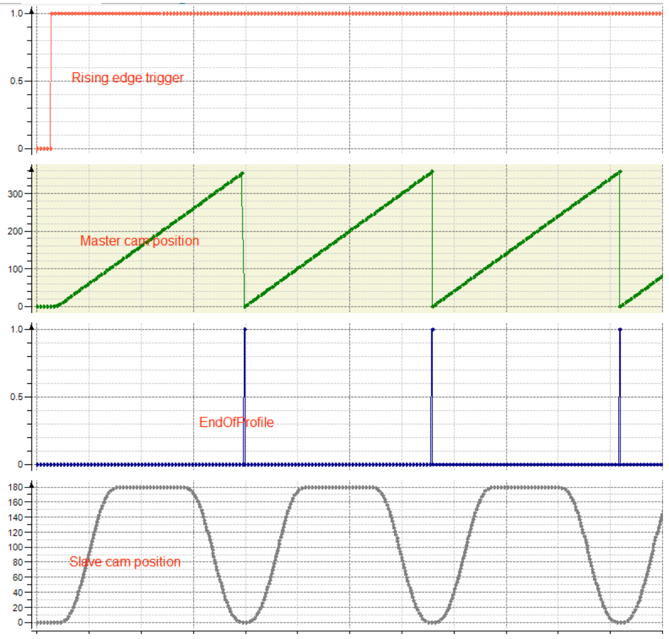
Note: The positions of the master and slave axes refer to the positions of the master and slave axes required by the cam function curve, but not the physical real axis positions of the master and slave axes.

Relationship between the cyclic mode and EndOfProfile: Whether the cyclic or non-cyclic mode determines whether the e-cam needs to be performed again after the master axis reaches the end position.

In non-cyclic mode: Periodic is FALSE in the MC_CamTableSelect instruction. When the cam is completed, EndofProfile outputs TRUE; when Execute inputs FALSE, EndofProfile outputs FALSE. At this time, the cam only runs one master axis cycle.

Note: The master axis cycle indicates the range from the start position to the end position of the master axis of the e-cam.

In cyclic mode: Periodic is TRUE in the MC_CamTableSelect instruction.

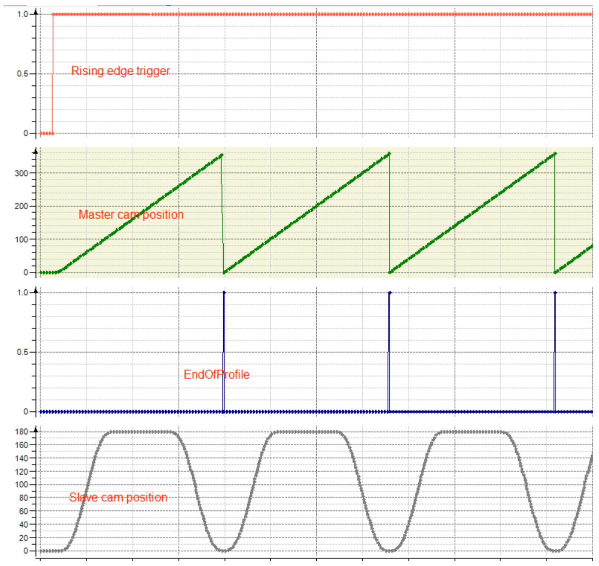


At this time, after completing one master axis cycle, the cam starts the next cycle, and the TRUE output of the EndofProfile signal only maintains one task cycle.

Note: When the cam master-axis position is greater than or equal to the cam end position, the EndofProfile signal outputs TRUE, and the cam master-axis position is updated to (Cam start position + Actual position - End position).

For example: The start position and end position of the cam master axis are 0 and 360, the master-slave axis scaling is set to 1, the master-slave axis offset value is set to 0, the task cycle is 2 ms, and the master axis speed is 100. When the cam master-axis position in a certain task cycle is 359.99, the output of EndofProfile in the next cycle is TRUE and the master axis position becomes 359.99+100*0.002-360=0.19.

The start position and end position of the cam profile designed in cyclic mode need to maintain a smooth transition; otherwise, jumping may be caused. For example, if the start speed is 0 and the end speed is not 0, jumping is caused when the master axis transits from the end of the cycle and the beginning of the new cycle.



The master/slave axis absolute/relative mode relationship in StartMode and MC_CamTableSlect is as follows:

Absolute mode: At the beginning of a new e-cam cycle, the calculation of the e-cam has no relationship with the present slave axis position. If the start position of the slave axis relative to the master axis is different from the end position of the slave axis relative to the master axis, jumping is caused.

Relative mode: The new e-cam cycle changes according to the present position of the slave axis; that is, the position of the slave axis at the end of the previous e-cam cycle is considered as "slave axis offset" in the present e-cam movement, therefore added. However, if the position of the slave axis corresponding to the start position of the master axis is not 0 in the e-cam definition, jumping is caused.

Ramp input: Potential jumping at the beginning of the e-cam is prevented by adding a compensation movement (The movement is based on VelocityDiff, acceleration, and deceleration. Therefore, as long as the slave axis is rotating, the forward ramp input can only use forward compensation, and the reverse ramp input can only use reverse compensation. For the slave axis in linear motion, the compensation direction can be realized automatically, that is, the forward ramp input and the reverse ramp input can be interpreted by the ramp input.

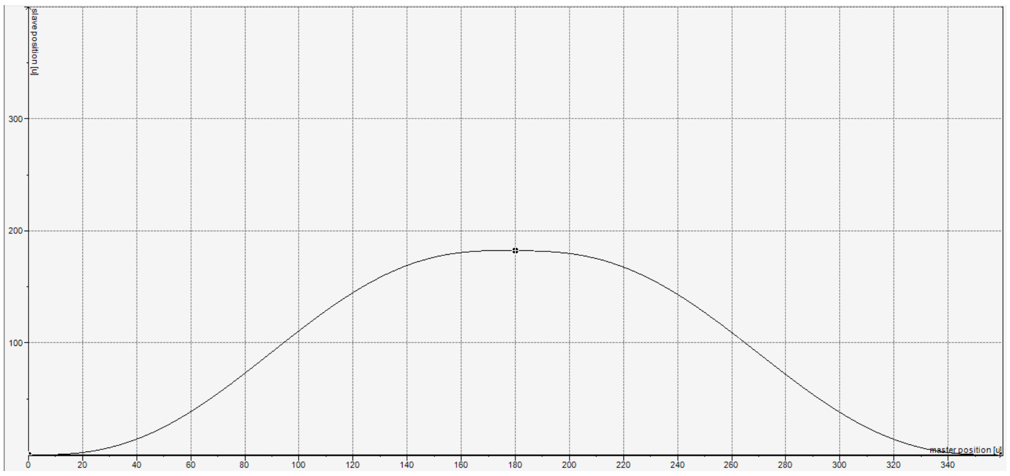
The relationship table is as follows:

MC_CamTableSelect.MasterAbsolute	Master Axis Mode
absolute	Absolute mode
relative	Relative mode

MC_CamIn.StartMode	MC_CamTableSelect.SlaveAbsolute	Slave Axis Mode
absolute	TRUE	Absolute mode
absolute	FALSE	Relative mode
relative	TRUE	Relative mode
relative	FALSE	Relative mode
ramp_in	TRUE	Absolute mode of ramp switching in
ramp_in	FALSE	Relative mode of ramp switching in
ramp_in_pos	TRUE	Absolute mode of forward ramp switching in
ramp_in_pos	FALSE	Relative mode of forward ramp switching in
ramp_in_neg	TRUE	Absolute mode of reverse ramp switching in
ramp_in_neg	FALSE	Relative mode of reverse ramp switching in

The relationship is described as follows:

Cam master-axis range: 0–360; cam slave-axis range: 0–180; cyclic mode; master/slave axis offset value: 0; master/slave axis scaling ratio: 1. The designed cam table is shown in the following figure.



StartMode=0 (Absolute mode)

In MC_CamTableSlect, when MasterAbsolute is set to FALSE and SlaveAbsolute is set to TRUE, the master axis is working in relative mode and the slave axis is working in absolute mode. When the rising edge of Execute starts the cam, the master axis of the cam starts from the "start position" (0) in the cam table, and the cam slave axis is calculated and output according to the above-mentioned "cam table engaging formula". The real axis instruction position of the slave axis is equal to the output value of the engaging calculation. For example, if the start position of the cam slave axis is 0, and the real axis position of the slave axis is 20 when the cam is started, the real axis position instruction of the slave axis is 0 at the start, which causes jumping.

Note: In this case, jumping occurs when the slave axis (real axis) start position is not the slave axis start position of the cam.

In MC_CamTableSlect, when MasterAbsolute is set to FALSE and SlaveAbsolute is set to FALSE, the master axis is working in relative mode and the slave axis is working in relative mode. When the rising edge of Execute starts the cam, the master axis of the cam starts from the "start position" (0) in the cam table, the cam slave axis is calculated and output according to the above-mentioned "cam table engaging formula". The real axis instruction position of the slave axis is equal to [Output value of engaging calculation, or cam slave-axis position) + (Real axis position of the slave axis at startup)].

For example, when the cam is started, if the real axis position of the slave axis is 20, and the slave axis start position in the cam table is 0, then the real axis instruction position of the slave axis is 20 when the cam is started, the position in the following is 20 plus the calculated value of the cam table, and the highest value is 20 plus the max. calculated value (180) of the cam table, that is, 200.

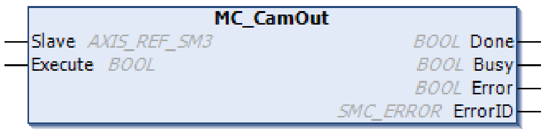
7.2.2 MC_Camout

MC_Camout: used to decouple the cam relationship of the slave axis.

Note: After executing this instruction, the slave axis continues to run at the speed used before the decoupling. Therefore, this instruction needs to be used in conjunction with instructions such as MC_Stop.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_Camout	Cam decoupling instruction	 MC_CamOut
		Slave AXIS_REF_SM3
		Execute BOOL
		BOOL Done
		BOOL Busy
		BOOL Error
		SMC_ERROR ErrorID
			MC_CamOut(
			Slave:=,
			Execute:=,
			Done=>,
			Busy=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Slave	Slave axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Cam function exit	BOOL	TRUE, FALSE	FALSE	The rising edge starts the execution of the function block

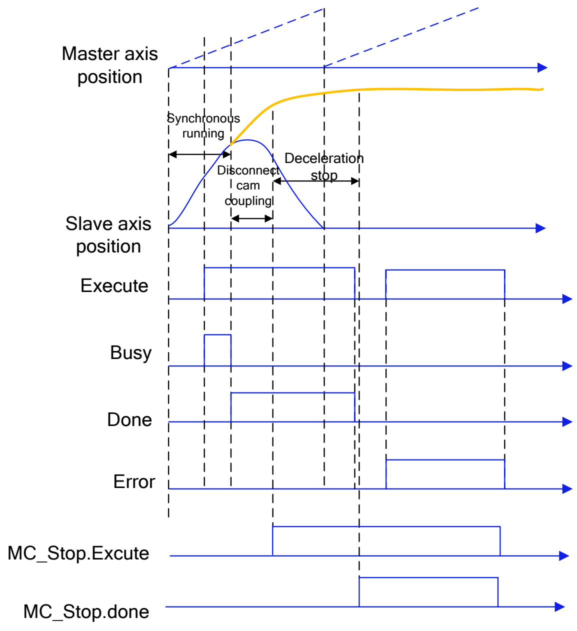
Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Completed	BOOL	TRUE, FALSE	FALSE	The cam relationship with the master slave has been decoupled
Busy	Synchronous running	BOOL	TRUE, FALSE	FALSE	The instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	Error is set when an error is detected.
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

The instruction is used to decouple the cam relationship of the slave axis. At the rising edge, the cam relationship of the slave axis is decoupled. After the decoupling, the salve axis may or may not stop. If the slave axis speed is not 0 before the instruction is executed, the cam relationship is decoupled after the DONE signal is completed, but the slave axis still runs at the speed before the relationship is decoupled. If the slave axis does not have a cam coupling relationship, ERROR is output.

4. Timing diagram

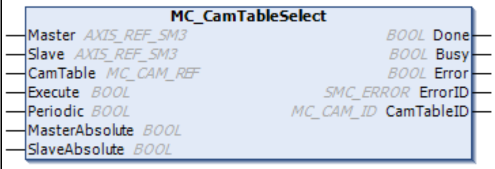


7.2.3 MC_CamTableSelect

MC_CamTableSelect: used to select a cam table in conjunction with MC_CamIn.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_CamTableSelect	Cam table selection instruction	 MC_CamTableSelect
		Master AXIS_REF_SM3
		Slave AXIS_REF_SM3
		CamTable MC_CAM_REF
		Execute BOOL
		Periodic BOOL
		MasterAbsolute BOOL
		SlaveAbsolute BOOL
		BOOL Done
		BOOL Busy
		BOOL Error
		SMC_ERROR ErrorID
		MC_CAM_ID CamTableID
			MC_CamTableSelect(
			Master:=,
			Slave:=,
			CamTable:=,
			Execute:=,
			Periodic:=,
			MasterAbsolute:=,
			SlaveAbsolute:=,
			Done=>,
			Busy=>,
			Error=>,
			ErrorID=>,
			CamTableID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Master	Master axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3
Slave	Slave axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3
CamTable	Table selection	MC_CAM_REF	-	-	Reference to cam table description, that is, an instance of MC_CAM_REF

Note: The master axis and the slave axis must be different axes. Otherwise, errors may be reported. The cam table specified by CamTable must be correct; otherwise, errors may be reported. The master and slave axes may be real or virtual axes.

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Executing	BOOL	TRUE, FALSE	FALSE	The rising edge starts the execution of the function block
Periodic	Repeated mode	BOOL	TRUE, FALSE	FALSE	Used to specify whether the cam table is executed only once or repeatedly: TRUE: Repeatedly FALSE: Not repeatedly
MasterAbsolute	Master axis absolute mode	BOOL	TRUE, FALSE	FALSE	Used to specify whether the coordinate system of master axis tracking uses an absolute or relative position: 1: Absolute position 0: Relative position

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
SlaveAbsolute	Slave axis absolute mode	BOOL	TRUE, FALSE	FALSE	Used with StartMode in MC_CamIn to specify whether the present instruction position of the slave axis is the absolute (cam table output value corresponding to the current master axis position) or relative (slave axis position at the start of the cam table output value superposition instruction) position output of the cam table: 1: Absolute position; 0: Relative position

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Completed	BOOL	TRUE, FALSE	FALSE	The cam relationship with the master slave has been decoupled
Busy	Synchronous running	BOOL	TRUE, FALSE	FALSE	The instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	Error is set when an error is detected.
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output
CamTableID	Effective cam ID	MC_CAM_ID	-	-	Used to select the effective cam ID, which is used together with CamTableID in MC_CamIn

3. Function description

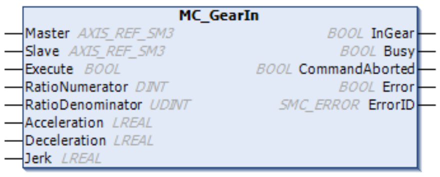
The instruction specifies the cam table required for e-cam running. Therefore, before using this instruction, you must edit the cam table (with a cam editor or online). The specified cam table can be executed at the rising edge of Execute or refreshed after cam table update. When the Done signal is TRUE, the variable "CamTableID" is output and takes effect. During instruction execution, Busy is TRUE; when Done is TRUE, Busy is FALSE. For details about MasterAbsolute, SlaveAbsolute, and Periodic, see MC_CamIn.

7.2.4 MC_GearIn

MC_GearIn: used to set the gear ratio between the slave axis and the master axis to perform electronic gearing.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_GearIn	E-gear function block	 MC_GearIn
		Master AXIS_REF_SM3
		Slave AXIS_REF_SM3
		Execute BOOL
		RatioNumerator DINT
		RatioDenominator UDINT
		Acceleration LREAL
		Deceleration LREAL
		Jerk LREAL
		BOOL InGear
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		SMC_ERROR ErrorID
			MC_GearIn(
			Master:=,
			Slave:=,
			Execute:=,
			RatioNumerator:=,
			RatioDenominator:=,

Instruction	Name	Graphical Representation	ST Representation

			Acceleration:=,
			Deceleration:=,
			Jerk:=,
			InGear=>,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Master	Master axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3
Slave	Slave axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Executing	BOOL	TRUE, FALSE	FALSE	The function block is triggered at the rising edge
RatioNumerator	Numerator of gear ratio	DINT	Positive, negative	1	Numerator of gear ratio
RatioDenominator	Denominator of gear ratio	UDINT	Positive number	1	Denominator of gear ratio
Acceleration	Acceleration	LREAL	Positive or 0	-	Specified acceleration
Deceleration	Deceleration	LREAL	Positive or 0	-	Specified deceleration
Jerk	Jerk	LREAL	Positive or 0	-	Jerk

Output variables

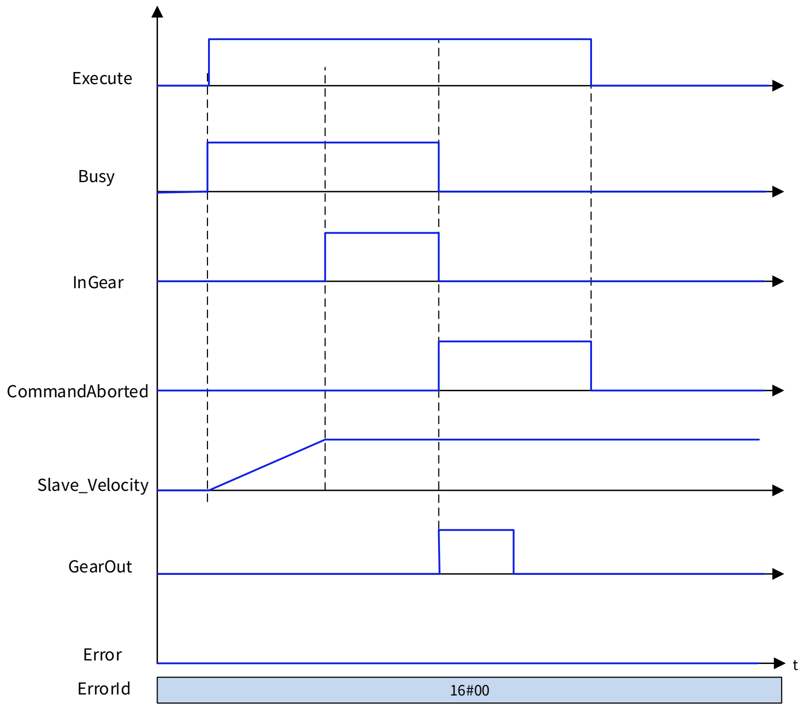
Output Variable	Name	Data Type	Valid Range	Initial Value	Description
InGear	Gear ratio reached	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the slave axis reaches the target speed
Busy	Synchronous running	BOOL	TRUE, FALSE	FALSE	The instruction is being executed
CommandAborted	Interruption	BOOL	TRUE, FALSE	FALSE	It is set to TRUE when the instruction is aborted by another control instruction
Error	Error flag	BOOL	TRUE, FALSE	FALSE	Error is set when an error is detected.
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

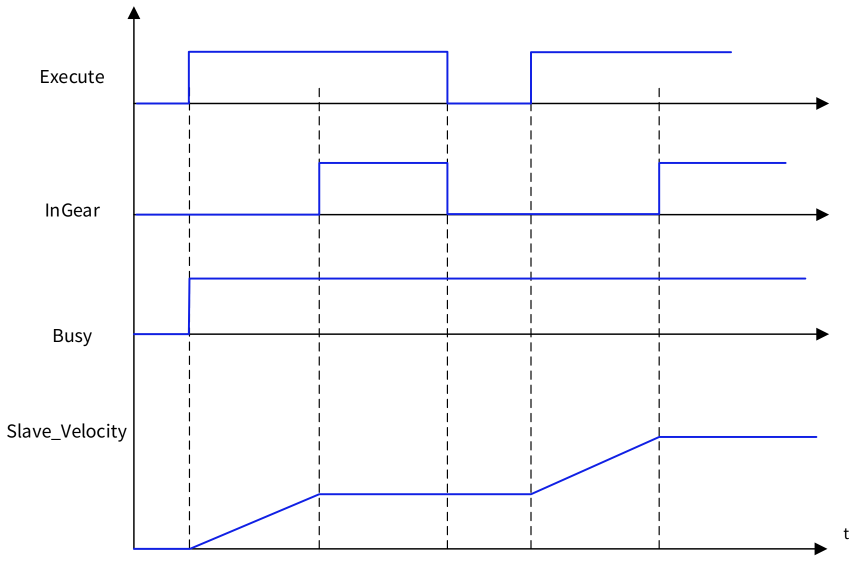
The e-gear action is started at the rising edge of Execute. To achieve decoupling after executing the e-gear, the GearOut instruction must be used. This instruction is a speed e-gear function, and the synchronization distance loss caused during acceleration will not be automatically compensated. When the Busy signal is TRUE during instruction execution, if the slave axis target speed is not reached, the new rising edge of Execute will not affect it. When the Busy signal is TRUE during instruction execution, if the slave axis target speed is reached, the new rising edge of Execute will not affect it. When the target speed is reached, InGear is TRUE, and then: Slave axis movement amount = Master axis movement amount * RatioNumerator/RatioDenominator. If the master axis speed changes in real time, exercise caution before using this instruction.

Note: Do not use the MC_SetPosition instruction during instruction execution to avoid accidents caused by the rapid motor running.

4. Timing diagram



The timing diagram of the restart after a gear ratio parameter change is as follows:

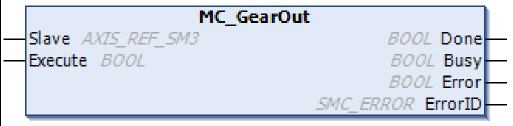


7.2.5 MC_GearOut

MC_GearOut: used to terminate the MC_GearIn and MC_GearInPos instructions that are being executed.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_GearOut	E-gear decoupling instruction	 MC_GearOut
		Slave AXIS_REF_SM3
		Execute BOOL
		BOOL Done
		BOOL Busy
		BOOL Error
		SMC_ERROR ErrorID
			MC_GearOut(
			Slave:=,
			Execute:=,
			Done=>,
			Busy=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Slave	Slave axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Executing	BOOL	TRUE, FALSE	FALSE	The function block is triggered at the rising edge

Output variables

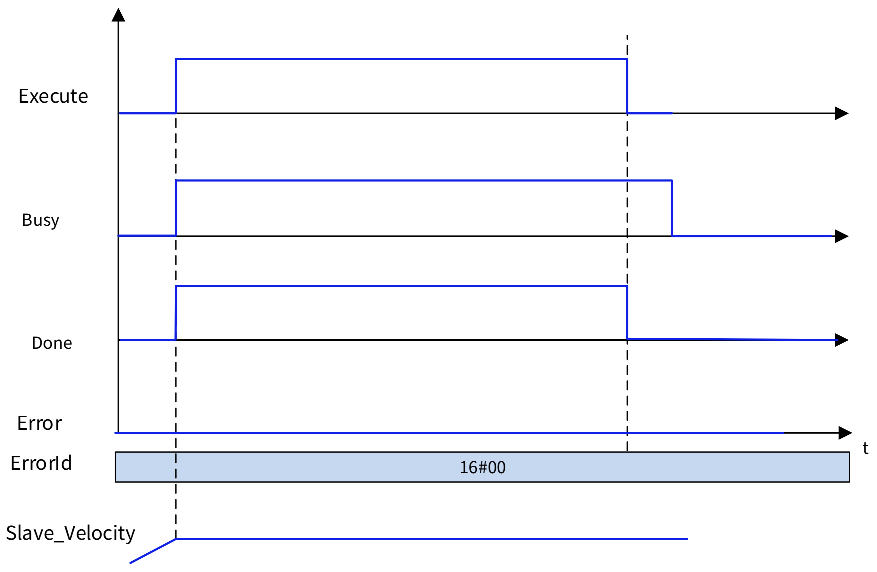
Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Completed	BOOL	TRUE, FALSE	FALSE	It is TRUE when the e-gear relationship between the slave axis and the master axis is decoupled
Busy	Synchronous running	BOOL	TRUE, FALSE	FALSE	The instruction is being executed
Error	Error flag	BOOL	TRUE, FALSE	FALSE	Error is set when an error is detected.
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

The e-gear decoupling action is started at the rising edge of Execute. If Execute is TRUE and ERROR is FALSE, Busy is TRUE and Done is TRUE.

After the e-gear decoupling action is completed, the slave axis speed used before decoupling is used. Therefore, the slave axis is stopped in conjunction with the MC_Stop instruction. At the falling edge of Execute, Done is FALSE.

4. Timing diagram

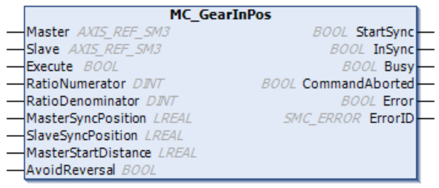


7.2.6 MC_GearInPos

MC_GearInPos: used to set the e-gear ratio between the slave axis and the master axis to perform electronic gearing. It specifies the master axis position, slave axis position, and master axis distance from the synchronization start to switch into e-gear actions.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_GearInPos	E-gear coupling switching-in position	 MC_GearInPos
		Master AXIS_REF_SM3
		Slave AXIS_REF_SM3
		Execute BOOL
		RatioNumerator DINT
		RatioDenominator DINT
		MasterSyncPosition LREAL
		SlaveSyncPosition LREAL
		MasterStartDistance LREAL
		AvoidReversal BOOL
		BOOL StartSync
		BOOL InSync
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		SMC_ERROR ErrorID
			MC_GearInPos(
			Master:=,
			Slave:=,
			Execute:=,
			RatioNumerator:=,
			RatioDenominator:=,
			MasterSyncPosition:=,
			SlaveSyncPosition:=,
			MasterStartDistance:=,
			AvoidReversal:=,
			StartSync=>,
			InSync=>,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Master	Master axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3
Slave	Slave axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Executing	BOOL	TRUE, FALSE	FALSE	The function block is triggered at the rising edge
RatioNumerator	Numerator of gear ratio	DINT	TRUE, FALSE	-	Numerator of the master/slave speed ratio
RatioDenominator	Denominator of gear ratio	DINT	-	-	Denominator of the master/slave speed ratio
MasterSync Position	Master axis synchronization position	LREAL	-	-	Master axis position when the master/slave axis gear ratios are coupled
SlaveSyncPosition	Slave axis synchronization position	LREAL	-	-	Slave axis position when the master/slave axis gear ratios are coupled
MasterStartDistance	Master axis position of synchronization execution	LREAL	-	-	According to this position value, -MasterSyncPosition, and the SlaveSyncPosition value, a smooth curve is calculated to make the slave axis gear synchronized with the master axis gear when the slave axis is at SlaveSyncPosition. The master axis range of the curve is [MasterStartDistance, MasterSyncPosition]
AvoidReversal	Disabling reverse running	BOOL	TRUE, FALSE	FALSE	It is set to FALSE if the physical position of the slave axis leads. It is set to TRUE if the slave axis cannot implement reverse running physically or the reverse running may cause danger. It is applicable only to modal axes. If reverse running cannot be avoided, the axis will stop due to exceptions.

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
StartSync	Coupling start	BOOL	TRUE, FALSE	FALSE	TRUE: The e-gear coupling is started
InSync	Coupling	BOOL	TRUE, FALSE	FALSE	TRUE: The e-gear coupling is completed, and the master/slave axis gear ratios are being coupled
Busy	Synchronous running	BOOL	TRUE, FALSE	FALSE	The instruction is being executed

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
CommandAborted	Instruction aborted	BOOL	TRUE, FALSE	FALSE	Aborted by another control instruction
Error	Error flag	BOOL	TRUE, FALSE	FALSE	Error is set when an error is detected.
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

The instruction is started at the rising edge of Execute.

After the action starts, the slave axis accelerates or decelerates at the target speed that is the master axis speed multiplied by the gear ratio.

The essential of the process from the synchronization start to the end is an e-cam where the slave axis follows the master axis in the synchronization interval. At this time, the instruction automatically designs a cam profile according to the master axis range (MasterSyncPosition-MasterStartDistance, MasterSyncPosition), the slave axis range (current position, SlaveSyncPosition), and the gear ratios. When synchronization is performed, the slave axis follows the master axis to complete the cam action.

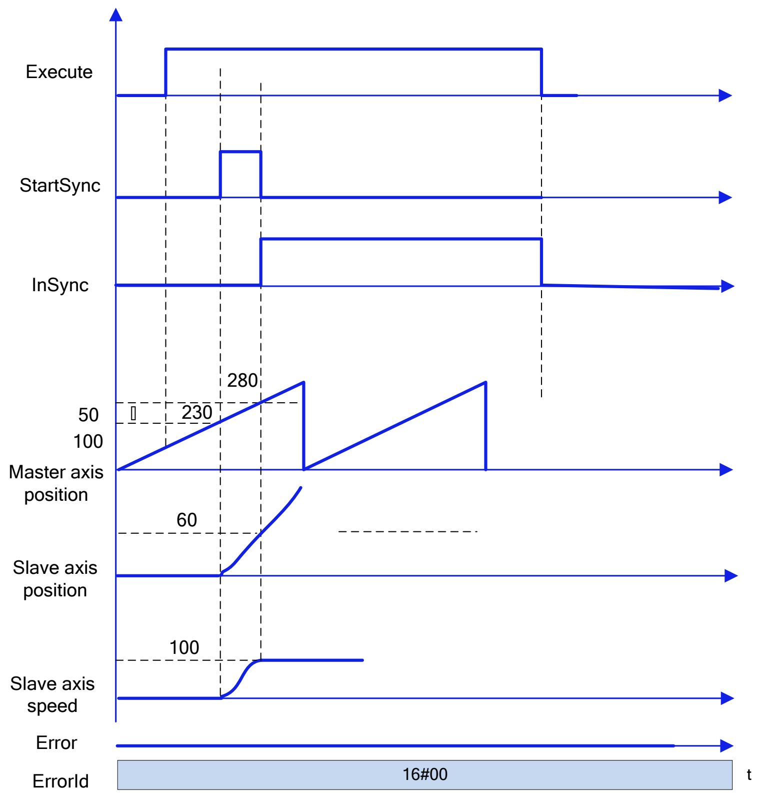
Note: If the master and slave axes work in linear mode, you need to ensure that the above-mentioned parameters are set properly; otherwise, the gear action cannot be performed correctly. Therefore, it is recommended that the master and slave axes work in cyclic mode when this instruction is used

For example: Both the master and slave axes move forward in linear mode. If the master axis position > MasterSyncPosition-MasterStartDistance, or the slave axis position > SlaveSyncPosition, when the instruction is executed, the e-gear movement cannot be switched in.

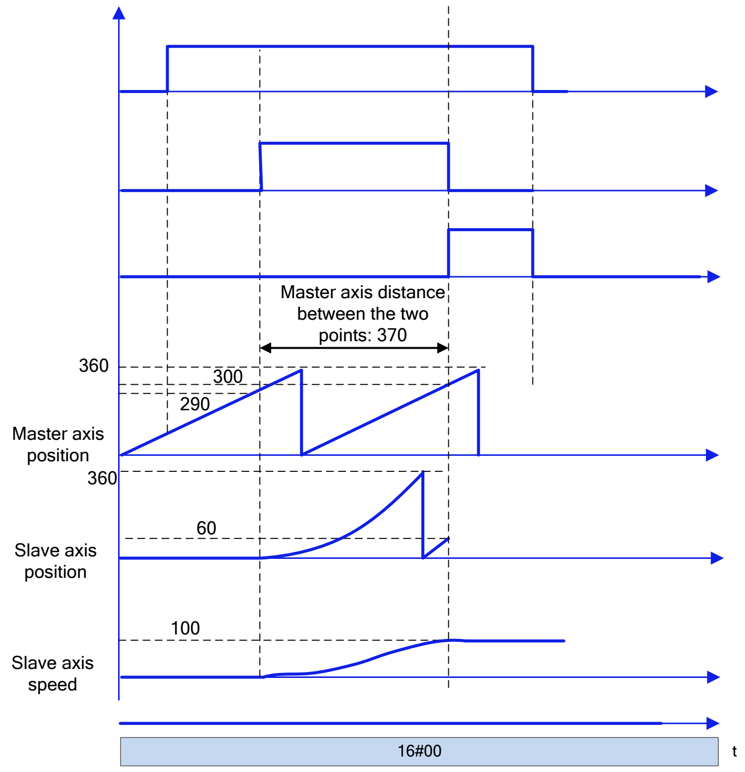
The timing diagram instances with different parameters are provided:

When both the master axis and the slave axis work in cyclic mode (360 cycles):

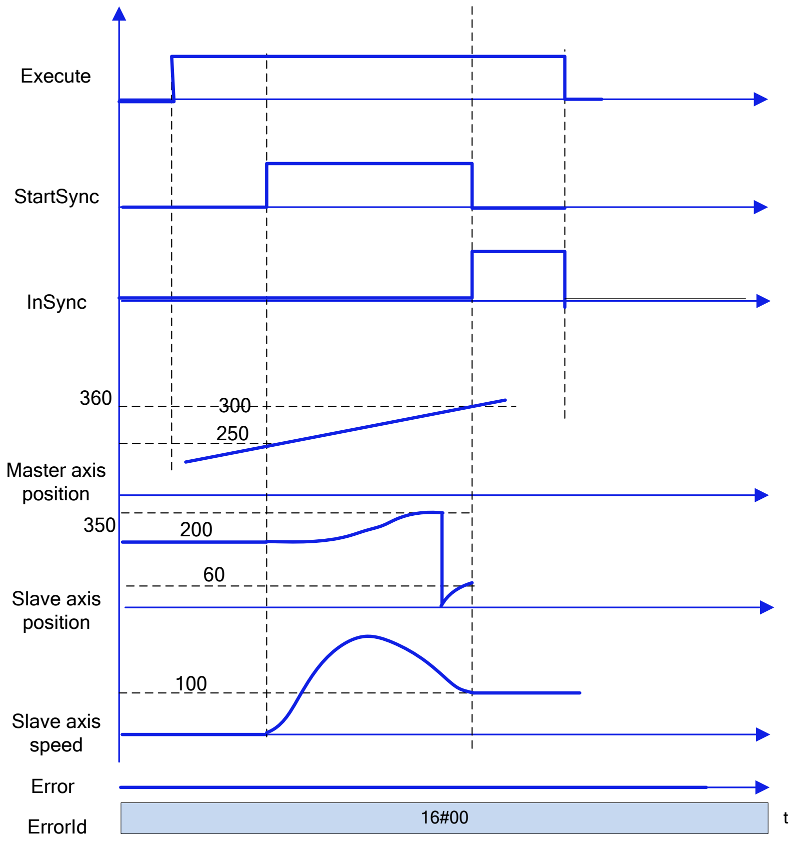
- MasterSyncPosition=280, MasterStartDistance=50, SlaveSyncPosition=60, Master axis speed=50, AvoidReversal=FALSE.



- MasterSyncPosition=300, MasterStartDistance=370, SlaveSyncPosition=60, Master axis speed=50, AvoidReversal=FALSE.



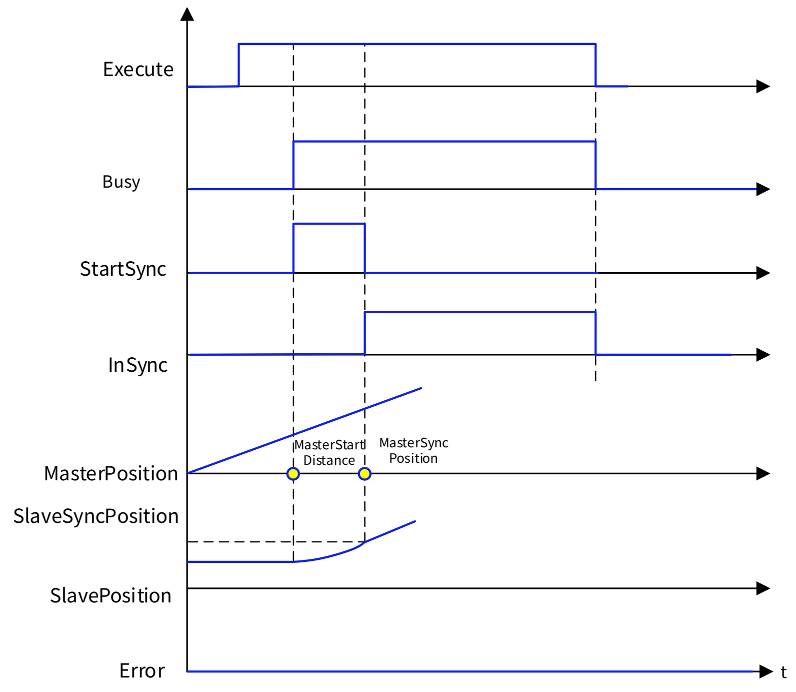
- MasterSyncPosition=300, MasterStartDistance=50, SlaveSyncPosition=60, Master axis speed=50, AvoidReversal=FALSE, Slave axis start position > 60.



When the synchronization is completed, InSync is TRUE, the target speed is reached also, and then: Slave axis movement amount = Master axis movement amount * RatioNumerator/RatioDenominator.

For AvoidReversal: If the slave axis is a modal axis and the master axis speed (a gear ratio multiple) is not relative to the slave axis speed, MC_GearInPos will try to avoid the reversal of the slave axis. It attempts to "stretch" the movement of the slave axis by adding 5 slave axis cycles. If the "stretch" is invalid, an error occurs and the slave axis stops abnormally. If the slave axis speed is related to the master axis speed (a gear ratio multiple), an error occurs and the slave axis stops abnormally. If the slave axis is a modal axis in linear mode, an error occurs when Execute inputs the rising edge.

4. Timing diagram

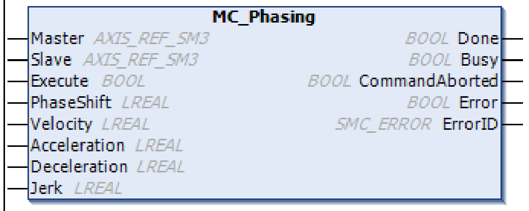


7.2.7 MC_Phasing

MC_Phasing: used to specify the phase difference between the master axis and the slave axis.

1. Instruction format

Instruction	Name	Graphical Representation	ST Representation

MC_Phasing	E-gear decoupling instruction		MC_Phasing
			Master:=,
			Slave:=,
			Execute:=,
			PhaseShift:=,
			Velocity:=,
			Acceleration:=,
			Deceleration:=,
			Jerk:=,
			Done=>,
			Busy=>,
			CommandAborted=>,
			Error=>,
			ErrorID=>);

2. Associated variables

Input/output variables

Input/Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Master	Master axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3
Slave	Slave axis	AXIS_REF	-	-	Reference to axis, that is, an instance of AXIS_REF_SM3

Input variables

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Execute	Executing	BOOL	TRUE, FALSE	FALSE	The function block is triggered at the rising edge
PhaseShift	Phase difference between the master axis and the slave axis	LREAL	-	0	Phase difference between the master axis and slave axis. A positive number indicates the slave axis lags
Velocity	Speed	LREAL	-	0	Max. speed at phase shift execution
Acceleration	Acceleration	LREAL	-	0	Max. acceleration at phase shift execution
Deceleration	Deceleration	LREAL	-	0	Max. deceleration at phase shift execution
Jerk	Jerk	LREAL	-	0	Max. jerk at phase shift execution

Output variables

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Done	Completed	BOOL	TRUE, FALSE	FALSE	It is TRUE when the e-gear relationship between the slave axis and the master axis is decoupled
Busy	Synchronous running	BOOL	TRUE, FALSE	FALSE	The instruction is being executed
Command Aborted	Instruction aborted	BOOL	TRUE, FALSE	FALSE	Aborted by another control instruction
Error	Error flag	BOOL	TRUE, FALSE	FALSE	Error is set when an error is detected.
ErrorID	Error ID	SMC_ERROR	-	0	When an exception occurs, the error ID is output

3. Function description

The phase shift is executed at the rising edge of Execute. The slave axis automatically calculates a smooth curve, completing the phase shift relative to the master axis. The master/slave axis phase difference is the value of PhaseShift in the input signal. When the value is a positive number, the slave axis lags behind the master axis.

After the phase shift is completed, Done is TRUE.

The master/slave axis phase difference is compensated according to PhaseShift, Velocity, Acceleration, and Deceleration.

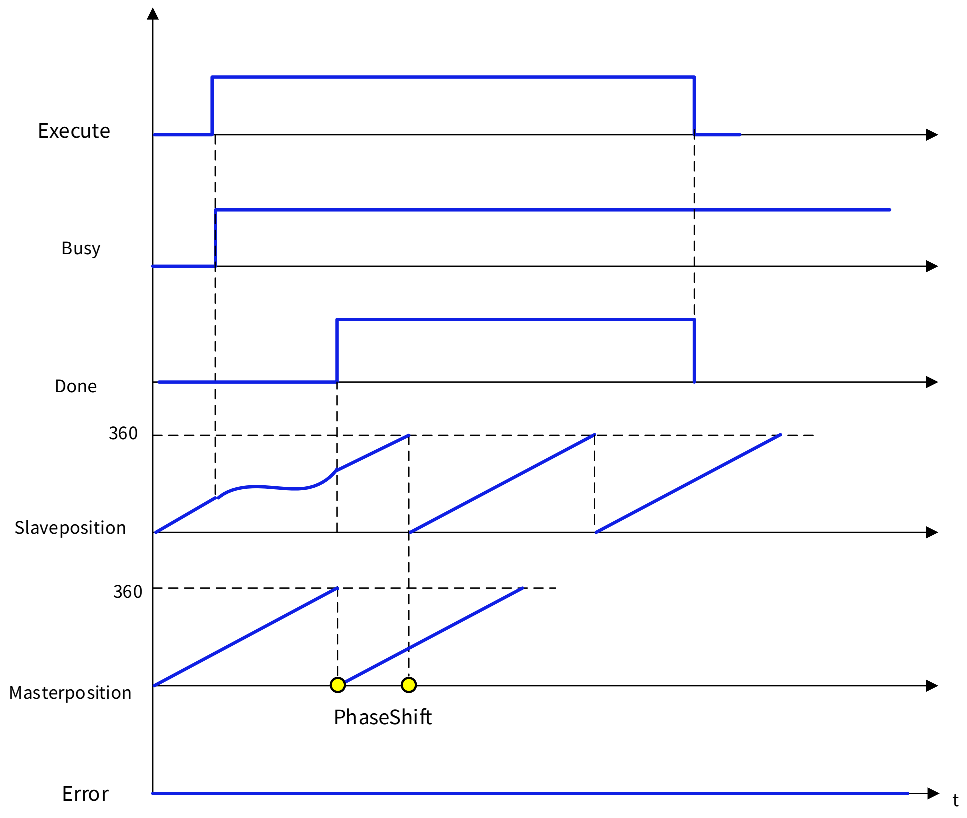
When the master/slave axis phase difference reaches PhaseShift, the Done signal is output.

During the instruction execution, if the master axis instruction position and feedback position remain unchanged, the slave axis is adjusted. Then the master/slave axis phase difference is PhaseShift.

The final result of this instruction is the phase shift between the given axis values, and therefore the actual feedback value of a real axis may be inconsistent with the final shift.

4. Timing diagram

The master and slave axes move in 360 cycles, and the adjustment is performed at the rising edge of the Execute signal. After the adjustment is completed, the phase shift between the slave axis and the master axis is the value of PhaseShift.



8 Communication Instructions

Freeport communication is a point-to-point communication method that uses a peer-to-peer approach to transmit data, and each end can receive and send data. It employs the full-duplex transmission mode, that is, data can be sent and received at both ends at the same time.

8.1 Serial Freeport Instructions

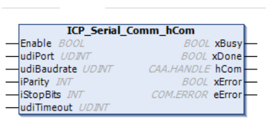
8.1.1 Instruction List

Instruction Category	Name	Function
RS485 freeport communication instruction	ICP_Serial_Comm_hCom	Create a RS485 freeport communication connection
	ICP_Serial_Comm_Read	RS485 freeport communication read data
	ICP_Serial_Comm_Write	RS485 freeport communication write data

8.1.2 ICP_Serial_Comm_hCom

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

ICP_Serial_Comm_hCom	RS485 freeport communication connection instruction	 ICP_Serial_Comm_hCom
		Enable BOOL
		udiPort UDINT
		udiBaudrate UDINT
		iParity INT
		iStopBits INT
		udiTimeout UDINT
		BOOL xBusy
		BOOL xDone
		CAA.HANDLE hCom
		BOOL xError
		COM.ERROR eError
			ICP_Serial_Comm_hCom(
			Enable:=,
			udiPort:=,
			udiBaudrate:=,
			iParity:=,
			iStopBits:=,
			udiTimeout:=,
			xBusy=>,
			xDone=>,
			hCom=>,
			xError=>,
			eError=>);

Associated variables:

Input Type	Name	Data Type	Valid Range	Initial Value	Description
Enable	Enabled	BOOL	TRUE, FALSE	FALSE	If it is TRUE, the function block is enabled
udiPort	Hardware serial port number	UDINT	-	-	Corresponding to hardware serial ports 1 and 2
udiBaudrate	Baud rate	UDINT	-	-	Serial port baud rate
iParity	Check bit	INT	-	-	Serial port check bit
iStopBits	Stop bit	INT	-	-	Serial port stop bit
udiTimeout	Timeout period	UDINT	-	-	Timeout period

Output Type	Name	Data Type	Valid Range	Initial Value	Description
xBusy	Function block running	BOOL	TRUE, FALSE	FALSE	Run flag
xDone	Completed signal	BOOL	TRUE, FALSE	FALSE	Completed flag
hCom	Handle	CAA.HANDLE	-	-	Communication establishment handle
xError	Error flag	BOOL	TRUE, FALSE	FALSE	Error flag
eError	Error flag	COM.ERROR	-	0	Error code

Program example:

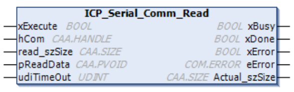
For the RS485 freeport communication connection instruction, when the input variable Enable of the ICP_Serial_Comm_hCom instruction is TRUE, a valid RS485 freeport communication handle will be created (hCom is greater than 0), and xBusy and xDone are TRUE.

```
////Open Port
ICP_Serial_Comm_hCom_1(
    Enable:= Open,                                      //Enable function block
    udiPort:= 1,                                        //PLC Hardware port
    udiBaudrate:= 19200,                                //Baudrate
    iParity:= COM.PARITY.NONE ,                         //Parity    0=COM.PARITY.EVEN, 1=COM.PARITY.ODD, 2=COM.PARITY.NONE ,
    iStopBits:= COM.STOPBIT.ONESTOPBIT ,                //StopBits    0=COM.STOPBIT.ONESTOPBIT, 1=ONE5STOPBITS, 2=COM.STOPBIT.TWOSTOPBITS,
    udiTimeout:= 1000,                                  //Timeout
    xBusy=> ,                                           //Function block active
    xDone=> ,                                           //Function block completion
    hCom=> ,                                            //specific Outputs
    xError=> ,                                          //Function block error
    eError=> );                                         //Local library error ID
```

8.1.3 ICP_Serial_Comm_Read

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

ICP_Serial_Comm_Read	RS485 freeport communication read data	 ICP_Serial_Comm_Read
		xExecute BOOL
		hCom CAA.HANDLE
		read_szSize CAA.SIZE
		pReadData CAA.PVOID
		udiTimeOut UDINT
		BOOL xBusy
		BOOL xDone
		BOOL xError
		COM.ERROR eError
		CAA.SIZE Actual_szSize
			ICP_Serial_Comm_Read(
			xExecute:=,
			hCom:=,
			read_szSize:=,
			pReadData:=,
			udiTimeOut:=,
			xBusy=>,
			xDone=>,
			xError=>,
			eError=>,
			actual_szSize=>);

Associated variables:

Input Type	Name	Data Type	Valid Range	Initial Value	Description
xExecute	Trigger	BOOL	TRUE, FALSE	FALSE	The function block is triggered at the rising edge
hCom	Connection handle	CAA.HANDLE	-	-	Communication establishment handle
read_szSize	Data length	CAA.SIZE	-	-	Read data length
pReadData	Data storage	CAA.PVOID	-	-	Read data storage address

udiTimeOut	Timeout period	UDINT	-	-	Communication timeout period
Output Type	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
xBusy	Function block running	BOOL	TRUE, FALSE	FALSE	Function block running flag
xDone	Completed flag	BOOL	TRUE, FALSE	FALSE	Completed flag
xError	Error flag	BOOL	TRUE, FALSE	FALSE	Read error
eError	Error flag	COM.ERROR	-	0	Error code
actual_szSize	Data length	CAA.SIZE	-	-	Actual length of read data

Program example:

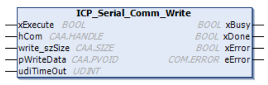
For the RS485 freeport communication read data instruction, when xExecute in the ICP_Serial_Comm_Read instruction is TRUE, data is read from the RS485 freeport communication buffer area, and xBusy is TRUE. If the data is read successfully, xDone is set for one scan cycle, and the read data will be placed in the variable with the address of pReadData.

```
//Read data module
ICP_Serial_Comm_Read_1(
    xExecute:= Read_xExecute,                                   //xExecute function block
    hCom:= ICP_Serial_Comm_hCom_1.hCom,                          //Handle to the open COM port. Ist returned by the COM.Open function block.
    read_szSize:= read_szSize_1,                                 //Maximum size of the pBuffer parameter in bytes
    pReadData:=  ADR(bReadData_1),                               //Pointer to a buffer to get the received data from the COM port
    udiTimeOut:= ,                                               //Timeout
    xBusy=> ,                                                    //Function block active
    xDone=> ,                                                    //Function block completion
    xError=> ,                                                   //Function block error
    eError=> ,                                                   //Local library error ID
    actual_szSize=> );                                           //Returns the number of received data bytes in pBuffer
```

8.1.4 ICP_Serial_Comm_Write

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

ICP_Serial_Comm_Write	RS485 freeport communication write data	 ICP_Serial_Comm_Write
		xExecute BOOL
		hCom CAA.HANDLE
		write_szSize CAA.SIZE
		pWriteData CAA.PVOID
		udiTimeOut UDINT
		BOOL xBusy
		BOOL xDone
		BOOL xError
		COM.ERROR eError
			ICP_Serial_Comm_Write(
			xExecute:=,
			hCom:=,
			write_szSize:=,
			pWriteData:=,
			udiTimeOut:=,
			xBusy=>,
			xDone=>,
			xError=>,
			eError=>);

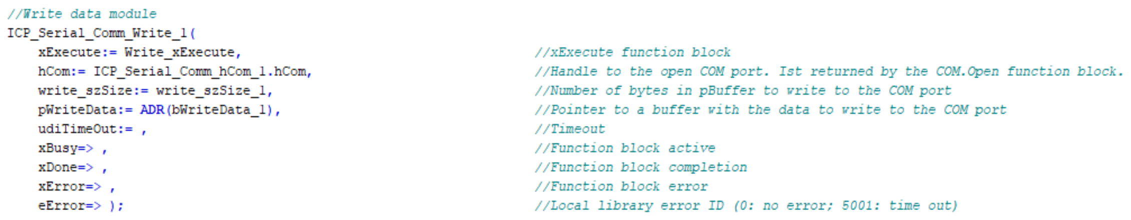
Associated variables:

Input Type	Name	Data Type	Valid Range	Initial Value	Description
xExecute	Trigger	BOOL	TRUE, FALSE	FALSE	The function block is triggered at the rising edge
hCom	Connection handle	CAA.HANDLE	-	-	Communication establishment handle
write_szSize	Data length	CAA.SIZE	-	-	Write data length
pWriteData	Data storage	CAA.PVOID	-	-	Write data storage address

udiTimeOut	Timeout period	UDINT	-	-	Communication timeout period
Output Type	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
xBusy	Function block running	BOOL	TRUE, FALSE	FALSE	Function block running flag
xDone	Completed flag	BOOL	TRUE, FALSE	FALSE	Completed flag
xError	Error flag	BOOL	TRUE, FALSE	FALSE	Write error
eError	Error flag	COM.ERROR	-	0	Error code

Program example:

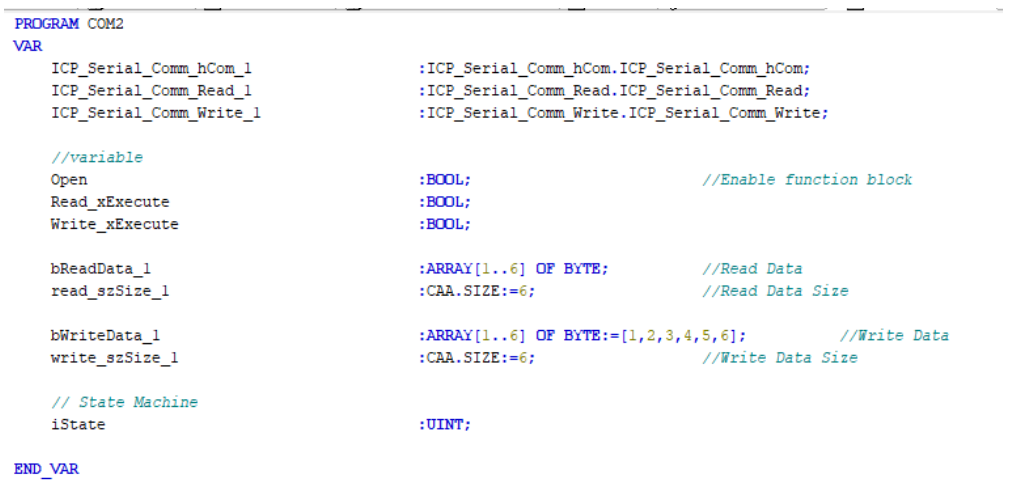
For the RS485 freeport communication read data instruction, when xExecute in the ICP_Serial_Comm_Write instruction is TRUE, the data with the starting address of pWriteData and the length of write_szSize in the send buffer set by the user will be sent to the target device. If the data is sent successfully in the timeout period udiTimeOut, Done is set to TRUE.



```
//Write data module
ICP_Serial_Comm_Write_1(
    xExecute:= Write_xExecute,                        //xExecute function block
    hCom:= ICP_Serial_Comm_hCom_1.hCom,               //Handle to the open COM port. Ist returned by the COM.Open function block.
    write_szSize:= write_szSize_1,                    //Number of bytes in pBuffer to write to the COM port
    pWriteData:= ADR(bWriteData_1),                   //Pointer to a buffer with the data to write to the COM port
    udiTimeOut:= ,                                    //Timeout
    xBusy=> ,                                         //Function block active
    xDone=> ,                                         //Function block completion
    xError=> ,                                        //Function block error
    eError=> );                                       //Local library error ID (0: no error; 5001: time out)
```

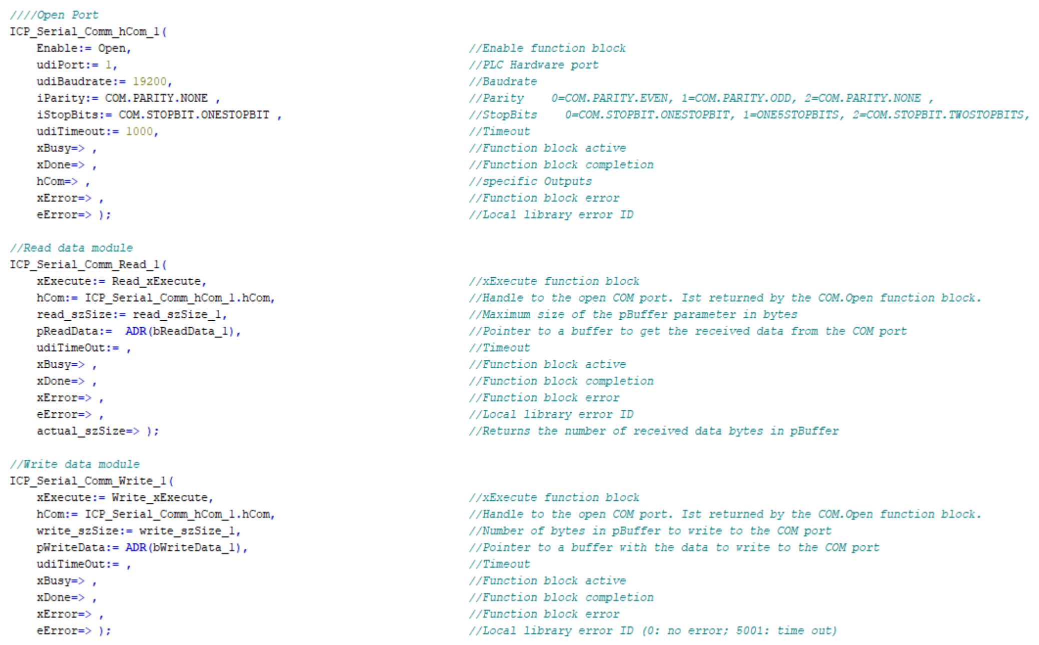
Reference program logic:

- Variable Declaration

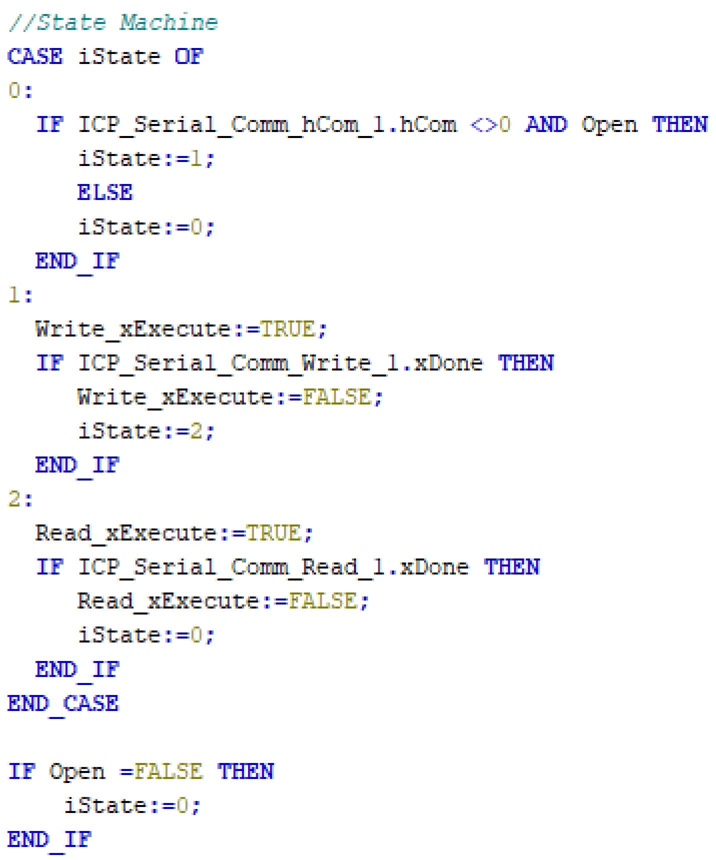


```
PROGRAM COM2
VAR
    ICP_Serial_Comm_hCom_1              :ICP_Serial_Comm_hCom.ICP_Serial_Comm_hCom;
    ICP_Serial_Comm_Read_1              :ICP_Serial_Comm_Read.ICP_Serial_Comm_Read;
    ICP_Serial_Comm_Write_1             :ICP_Serial_Comm_Write.ICP_Serial_Comm_Write;

    //variable
    Open                                :BOOL;                      //Enable function block
    Read_xExecute                       :BOOL;
    Write_xExecute                      :BOOL;

    bReadData_1                         :ARRAY[1..6] OF BYTE;       //Read Data
    read_szSize_1                       :CAA.SIZE:=6;               //Read Data Size

    bWriteData_1                        :ARRAY[1..6] OF BYTE:=[1,2,3,4,5,6];        //Write Data
    write_szSize_1                      :CAA.SIZE:=6;               //Write Data Size

    // State Machine
    iState                              :UINT;

END_VAR
```

- Program routine



```
////Open Port
ICP_Serial_Comm_hCom_1(
    Enable:= Open,                                      //Enable function block
    udiPort:= 1,                                        //PLC Hardware port
    udiBaudrate:= 19200,                                //Baudrate
    iParity:= COM.PARITY.NONE ,                         //Parity    0=COM.PARITY.EVEN, 1=COM.PARITY.ODD, 2=COM.PARITY.NONE ,
    iStopBits:= COM.STOPBIT.ONESTOPBIT ,                //StopBits    0=COM.STOPBIT.ONESTOPBIT, 1=ONE5STOPBITS, 2=COM.STOPBIT.TWOSTOPBITS,
    udiTimeout:= 1000,                                  //Timeout
    xBusy=> ,                                           //Function block active
    xDone=> ,                                           //Function block completion
    hCom=> ,                                            //specific Outputs
    xError=> ,                                          //Function block error
    eError=> );                                         //Local library error ID

//Read data module
ICP_Serial_Comm_Read_1(
    xExecute:= Read_xExecute,                           //xExecute function block
    hCom:= ICP_Serial_Comm_hCom_1.hCom,                 //Handle to the open COM port. Ist returned by the COM.Open function block.
    read_szSize:= read_szSize_1,                        //Maximum size of the pBuffer parameter in bytes
    pReadData:=  ADR(bReadData_1),                      //Pointer to a buffer to get the received data from the COM port
    udiTimeOut:= ,                                      //Timeout
    xBusy=> ,                                           //Function block active
    xDone=> ,                                           //Function block completion
    xError=> ,                                          //Function block error
    eError=> ,                                          //Local library error ID
    actual_szSize=> );                                  //Returns the number of received data bytes in pBuffer

//Write data module
ICP_Serial_Comm_Write_1(
    xExecute:= Write_xExecute,                          //xExecute function block
    hCom:= ICP_Serial_Comm_hCom_1.hCom,                 //Handle to the open COM port. Ist returned by the COM.Open function block.
    write_szSize:= write_szSize_1,                      //Number of bytes in pBuffer to write to the COM port
    pWriteData:= ADR(bWriteData_1),                     //Pointer to a buffer with the data to write to the COM port
    udiTimeOut:= ,                                      //Timeout
    xBusy=> ,                                           //Function block active
    xDone=> ,                                           //Function block completion
    xError=> ,                                          //Function block error
    eError=> );                                         //Local library error ID (0: no error; 5001: time out)
```



```
//State Machine
CASE iState OF
0:
  IF ICP_Serial_Comm_hCom_1.hCom <>0 AND Open THEN
     iState:=1;
     ELSE
     iState:=0;
  END_IF
1:
  Write_xExecute:=TRUE;
  IF ICP_Serial_Comm_Write_1.xDone THEN
     Write_xExecute:=FALSE;
     iState:=2;
  END_IF
2:
  Read_xExecute:=TRUE;
  IF ICP_Serial_Comm_Read_1.xDone THEN
     Read_xExecute:=FALSE;
     iState:=0;
  END_IF
END_CASE

IF Open =FALSE THEN
    iState:=0;
END_IF
```

8.2 TCP Freeport Communication Instructions

8.2.1 Instruction List

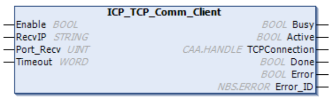
Instruction Category	Name	Function
TCP freeport communication instruction	ICP_TCP_Comm_Client	Create a TCP client communication service
	ICP_TCP_Comm_Write	Send TCP communication data
	ICP_TCP_Comm_Read	Rceive TCP communication data

Instruction Category	Name	Function
	ICP_TCP_Comm_Connect	Create a TCP connection to the server
	ICP_TCP_Comm_Server	Create a TCP server communication service

8.2.2 ICP_TCP_Comm_Client

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

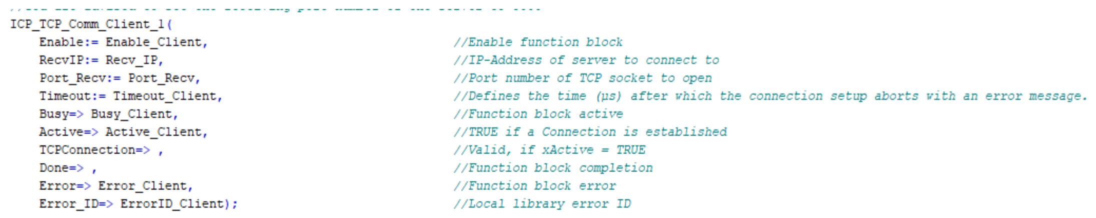
ICP_TCP_Comm_Client	Create a TCP client communication service	 ICP_TCP_Comm_Client
		Enable BOOL
		RecvIP STRING
		Port_Recv UINT
		Timeout WORD
		BOOL Busy
		BOOL Active
		CAA.HANDLE TCPConnection
		BOOL Done
		BOOL Error
		NBS.ERROR Error_ID
			ICP_TCP_Comm_Client(
			Enable:=,
			RecvIP:=,
			Port_Recv:=,
			Timeout:=,
			Busy=>t,
			Active=>,
			TCPConnection=>,
			Done=>,
			Error=>,
			Error_ID=>);

Associated variables:

Input Type	Name	Data Type	Valid Range	Initial Value	Description
Enable	Enabled	BOOL	TRUE, FALSE	FALSE	If it is TRUE, the function block is enabled
RecvIP	Receiving IP	STRING	-	-	Receiving controller IP
Port_Recv	Receiving port	UINT	-	-	Receiving controller port
Timeout	Timeout period	WORD	-	1000	Timeout period for requesting a connection
Output Type	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Busy	Function block running	BOOL	TRUE, FALSE	FALSE	Running signal
Active	Connection succeeded flag	BOOL	TRUE, FALSE	-	The flag indicating that the server and the client have established a communication successfully
TCPConnection	Connection handle	CAA.HANDLE	-	-	The connection handle for establishing a communication between the server and the client
Done	Completed flag	BOOL	TRUE, FALSE	FALSE	Completed signal
Error	Error flag	BOOL	TRUE, FALSE	FALSE	Connection error
Error_ID	Error ID	NBS.ERROR	-	0	Error code

Program example:

To create a TCP client communication service, when the input variable Enable of the ICP_TCP_Comm_Client instruction is TRUE, the local client monitors the connection request from the remote server. When the client is successfully connected to the server, a valid communication handle will be created between the client and the remote server (TCPConnection is greater than 0).

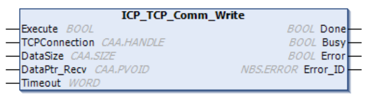


```
ICP_TCP_Comm_Client_1(
    Enable:= Enable_Client,                        //Enable function block
    RecvIP:= Recv_IP,                              //IP-Address of server to connect to
    Port_Recv:= Port_Recv,                         //Port number of TCP socket to open
    Timeout:= Timeout_Client,                      //Defines the time (µs) after which the connection setup aborts with an error message.
    Busy=> Busy_Client,                            //Function block active
    Active=> Active_Client,                        //TRUE if a Connection is established
    TCPConnection=> ,                              //Valid, if xActive = TRUE
    Done=> ,                                       //Function block completion
    Error=> Error_Client,                          //Function block error
    Error_ID=> ErrorID_Client);                    //Local library error ID
```

8.2.3 ICP_TCP_Comm_Write

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

ICP_TCP_Comm_Write	Send TCP communication data	 ICP_TCP_Comm_Write
		Execute BOOL
		TCPConnection CAA.HANDLE
		DataSize CAA.SIZE
		DataPtr_Recv CAA.PVOID
		Timeout WORD
		BOOL Done
		BOOL Busy
		BOOL Error
		NBS.ERROR Error_ID
			ICP_TCP_Comm_Write(
			Execute:=,
			TCPConnection:=,
			DataSize:=,
			DataPtr_Recv:=,
			Timeout:=,
			Done=>,
			Busy=>,
			Error=>,
			Error_ID=>);

Associated variables:

Input Type	Name	Data Type	Valid Range	Initial Value	Description
Enable	Enabled	BOOL	TRUE, FALSE	FALSE	If it is TRUE, the function block is enabled
TCPConnection	Connection handle	CAA.HANDLE	-	-	Connection handle
DataSize	Data length	CAA.SIZE	-	-	Data length, byte
DataPtr_Recv	Data address	CAA.PVOID	-	-	Data address
Timeout	Timeout period	WORD	-	1000	Timeout period for requesting a connection
Output Type	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Done	Completed flag	BOOL	TRUE, FALSE	-	Completed signal
Busy	Function block running	BOOL	TRUE, FALSE	FALSE	Running signal
Error	Error flag	BOOL	TRUE, FALSE	FALSE	Sending error
Error_ID	Error ID	NBS.ERROR	-	0	Error code

Program example:

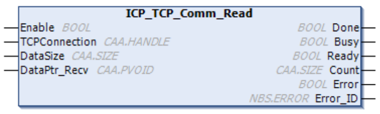
To send TCP communication data, when the input variable Enable of the ICP_TCP_Comm_Write instruction is TRUE, the data with the starting address of DataPtr_Recv and the length of DataSize in the send buffer set by the user will be sent to the target device connected to the TCPConnection handle. If the data is sent successfully within the timeout period Timeout, Done is set to TRUE.

```
ICP_TCP_Comm_Write_1(
    Execute:= Execute_Write,                             //Execute function block
    TCPConnection:= ICP_TCP_Comm_Client_1.TCPConnection ,  //
    DataSize:= Data_Length_Write,                        //Number of bytes in pBuffer to write to the port
    DataPtr_Recv:= ADR(Data_Write),                      //Pointer to a buffer with the data to write to the  port
    Timeout:= Timeout_Write,                             //Defines the time (µs) after which the connection setup aborts with an error message.
    Done=> Done_Write,                                   //Function block completion
    Busy=> Busy_Write,                                   //Function block active
    Error=> Error_Write,                                 //Function block error
    Error_ID=> );                                        //Local library error ID
```

8.2.4 ICP_TCP_Comm_Read

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

ICP_TCP_Comm_Read	Receive TCP communication data	 ICP_TCP_Comm_Read
		Enable BOOL
		TCPConnection CAA.HANDLE
		DataSize CAA.SIZE
		DataPtr_Recv CAA.PVOID
		BOOL Done
		BOOL Busy
		BOOL Ready
		CAA.SIZE Count
		BOOL Error
		NBS.ERROR Error_ID
			ICP_TCP_Comm_Read(
			Enable:=,
			TCPConnection:=,
			DataSize:=,
			DataPtr_Recv:=,
			Done=>,
			Busy=>,
			Ready=>,
			Count=>,
			Error=>,
			Error_ID=>);

Associated variables:

Input Type	Name	Data Type	Valid Range	Initial Value	Description
Enable	Enabled	BOOL	TRUE, FALSE	FALSE	If it is TRUE, the function block is enabled
TCPConnection	Connection handle	CAA.HANDLE	-	-	Connection handle
DataSize	Data length	CAA.SIZE	-	-	Data length, byte
DataPtr_Recv	Data address	CAA.PVOID	-	-	Data address
Output Type	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Done	Completed flag	BOOL	TRUE, FALSE	FALSE	Completed signal
Busy	Function block running	BOOL	TRUE, FALSE	FALSE	Running signal
Ready	Connection succeeded	BOOL	TRUE, FALSE	FALSE	Read data from the buffer, and if there is data, set the flag bit for one scan cycle
Count	Data length	CAA.SIZE	-	-	Actual length of received data
Error	Error flag	BOOL	TRUE, FALSE	FALSE	Receiving error
Error_ID	Error ID	NBS.ERROR	-	0	Error code

Program example:

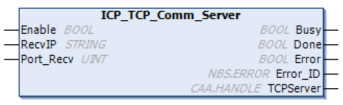
To receive TCP communication data, when the input variable Enable of the ICP_TCP_Comm_Read instruction is TRUE, data will be read from the TCP communication buffer, and Busy is TRUE. If the data is read successfully, Done is set for one scan cycle; the read data will be placed in the variable with the address DataPtr_Recv; at the same time, Ready is set for one scan cycle; the actual size value of the received data area will be assigned to Count, and the value of Count will be cleared after one scan cycle.

```
ICP_TCP_Comm_Read_1(
    Enable:= Enable_Read,                                //Enable function block
    TCPConnection:= ICP_TCP_Comm_Client_1.TCPConnection, //
    DataSize:= Data_Length_Read,                         //Maximum size of the pBuffer parameter in bytes
    DataPtr_Recv:= ADR(Data_Read),                       //Pointer to a buffer to get the received data from the port
    Done=> ,                                             //
    Busy=> Busy_Read,                                    //Function block active
    Ready=> Ready_Read,                                  //
    Count=> ,                                            //Returns the number of received data bytes in pBuffer
    Error=> Error_Read,                                  //Function block error
    Error_ID=> ErrorID_Read);                            //Local library error ID
```

8.2.5 ICP_TCP_Comm_Server

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

ICP_TCP_Comm_Server	Create a TCP server communication service	 ICP_TCP_Comm_Server
		Enable BOOL
		RecvIP STRING
		Port_Recv UINT
		BOOL Busy
		BOOL Done
		BOOL Error
		NBS.ERROR Error_ID
		CAA.HANDLE TCPServer
			ICP_TCP_Comm_Server(
			Enable:=,
			RecvIP:=,
			Port_Recv:=,
			Busy=>,
			Done=>,
			Error=>,
			Error_ID=>,
			TCPServer=>);

Associated variables:

Input Type	Name	Data Type	Valid Range	Initial Value	Description
Enable	Enabled	BOOL	TRUE, FALSE	FALSE	If it is TRUE, the function block is enabled
RecvIP	Controller IP	STRING	-	-	Server controller IP
Port_Recv	Controller port number	UINT	-	-	Server controller port number
Output Type	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Busy	Function block running	BOOL	TRUE, FALSE	FALSE	Run flag
Done	Completed flag	BOOL	TRUE, FALSE	FALSE	Completed signal
Error	Error flag	BOOL	TRUE, FALSE	FALSE	Connection error
Error_ID	Error ID	NBS.ERROR	-	-	Error code
TCPServer	Server handle	CAA.HANDLE	-	-	TCP server handle

Program example:

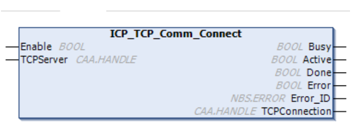
To create a TCP server communication service, when the input variable Enable of the ICP_TCP_Comm_Server instruction is TRUE, a valid communication handle will be created between the server and the remote client (TCPServer is not equal to 0).

```
//You are advised to set the receiving port number of the server to 5000

ICP_TCP_Comm_Server_1(
    Enable:= Enable_Server,            //Enable function block
    RecvIP:= Recv_IP,                  //IP-Address of server to connect to
    Port_Recv:= Port_Recv,             //Port number of TCP socket to open
    Busy=> ,                           //Function block active
    Done=> ,                           //
    Error=> Error_Server,              //Function block error
    Error_ID=> ErrorID_Server,         //Local library error ID
    TCPServer=> );                     //CAA.HANDLE
```

8.2.6 ICP_TCP_Comm_Connect

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

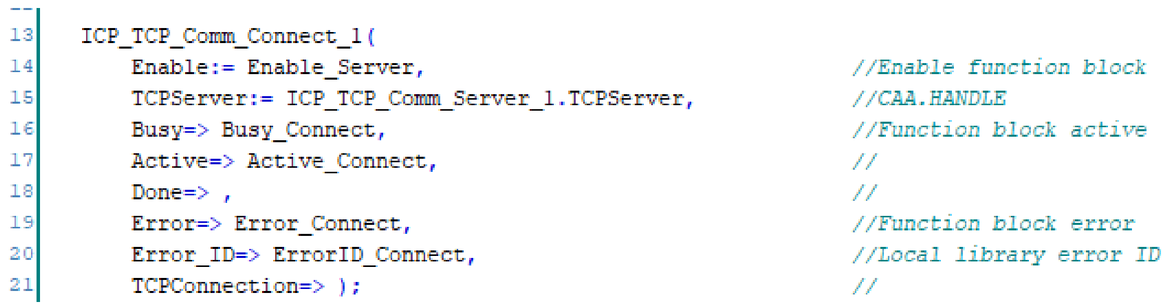
ICP_TCP_Comm_Connect	Create a TCP connection to the server	 ICP_TCP_Comm_Connect
		Enable BOOL
		TCPServer CAA.HANDLE
		BOOL Busy
		BOOL Active
		BOOL Done
		BOOL Error
		NBS.ERROR Error_ID
		CAA.HANDLE TCPConnection
			ICP_TCP_Comm_Connect(
			Enable:=,
			TCPServer:=,
			Busy=>,
			Active=>,
			Done=>,
			Error=>,
			Error_ID=>,
			TCPConnection=>);

Associated variables:

Input Type	Name	Data Type	Valid Range	Initial Value	Description
Enable	Enabled	BOOL	TRUE, FALSE	FALSE	If it is TRUE, the function block is enabled
TCPServer	Connection handle	CAA.HANDLE	-	-	TCP connection handle
Output Type	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Busy	Function block running	BOOL	TRUE, FALSE	FALSE	Run flag
Active	Connection succeeded	BOOL	TRUE, FALSE	FALSE	The server monitors the remote client connection handle flag
Done	Completed flag	BOOL	TRUE, FALSE	FALSE	Completed signal
Error	Error flag	BOOL	TRUE, FALSE	FALSE	Connection error
Error_ID	Error ID	NBS.ERROR	-	-	Error code
TCPConnection	Connection handle	CAA.HANDLE	-	-	The connection handle for establishing a communication between the server and the client

Program example:

To create a TCP connection to the server instruction, when the input variable Enable of the ICP_TCP_Comm_Connect instruction is TRUE, the local server monitors the connection request from the remote client. When the client successfully connects to the server through the server handle TCPServer, a valid communication handle will be created between the server and the remote client (TCPConnection is greater than 0).



```
ICP_TCP_Comm_Connect_1(
    Enable:= Enable_Server,                                    //Enable function block
    TCPServer:= ICP_TCP_Comm_Server_1.TCPServer,               //CAA.HANDLE
    Busy=> Busy_Connect,                                       //Function block active
    Active=> Active_Connect,                                   //
    Done=> ,                                                   //
    Error=> Error_Connect,                                     //Function block error
    Error_ID=> ErrorID_Connect,                                //Local library error ID
    TCPConnection=> );                                         //
```

8.3 UDP Freeport Communication Instructions

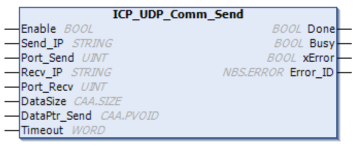
8.3.1 Instruction List

Instruction Category	Name	Function
UDP freeport communication instruction	ICP_UDP_Comm_Send	Send UDP communication data
	ICP_UDP_Comm_Receive	Rceive UDP communication data

8.3.2 ICP_UDP_Comm_Send

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

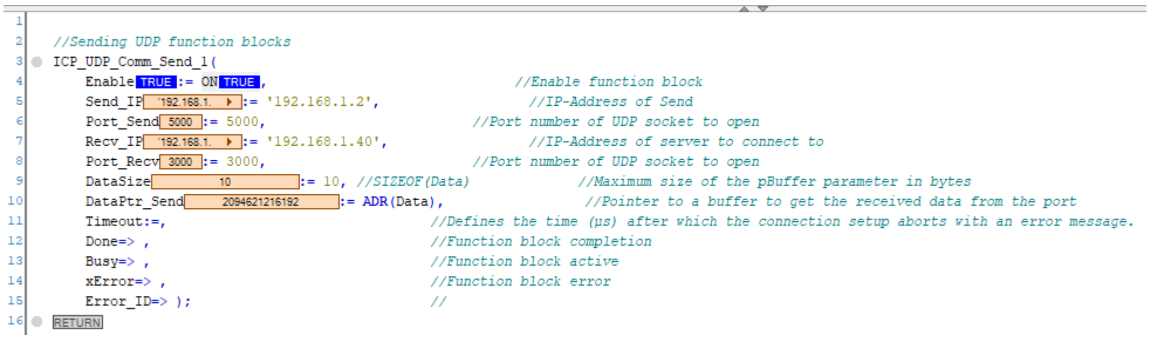
ICP_UDP_Comm_Send	Send UDP communication data	 ICP_UDP_Comm_Send
		Enable BOOL
		Send_IP STRING
		Port_Send UINT
		Recv_IP STRING
		Port_Recv UINT
		DataSize CAA.SIZE
		DataPtr_Send CAA.PVOID
		Timeout WORD
		BOOL Done
		BOOL Busy
		BOOL xError
		NBS.ERROR Error_ID
			ICP_UDP_Comm_Send(
			Enable:=,
			Send_IP:=,
			Port_Send:=,
			Recv_IP:=,
			Port_Recv:=,
			DataSize:=,
			DataPtr_Send:=,
			Timeout:=,
			Done=>,
			Busy=>,
			xError=>);

Associated variables:

Input Type	Name	Data Type	Valid Range	Initial Value	Description
Enable	Enabled	BOOL	TRUE, FALSE	FALSE	If it is TRUE, the function block is enabled
Send_IP	Sending IP	STRING	-	-	Sending controller IP
Port_Send	Sending port	UINT	-	-	Sending controller port
Recv_IP	Receiving IP	STRING	-	-	Receiving controller IP
Port_Recv	Receiving port	UINT	-	-	Receiving controller port
DataSize	Data length	CAA.SIZE	-	-	Data length to be sent
DataPtr_Send	Data storage	CAA.PVOID	-	-	Received data storage address
Timeout	Timeout period	WORD	-	1000	Timeout period for requesting a connection
Output Type	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Done	Completed flag	BOOL	TRUE, FALSE	FALSE	Completed signal
Busy	Function block running	BOOL	TRUE, FALSE	FALSE	Run flag
xError	Error flag	BOOL	TRUE, FALSE	FALSE	Sending error
Error_ID	Error ID	NBS.ERROR	-	0	Error code

Program example:

To send UDP communication data, when the input variable Enable of the ICP_UDP_Comm_Send instruction is TRUE, the data with the starting address of DataPtr_Send and the length of DataSize in the send buffer set by the user will be sent to the target device with the receiving IP of Recv_IP. If the data is sent successfully within the timeout period Timeout, Done is set to TRUE.

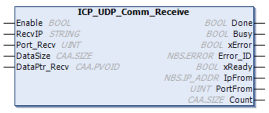


```
//Sending UDP function blocks
ICP_UDP_Comm_Send_1(
    Enable TRUE := ON TRUE ,                          //Enable function block
    Send_IP '192.168.1. ▸ := '192.168.1.2',              //IP-Address of Send
    Port_Send 5000 := 5000,                        //Port number of UDP socket to open
    Recv_IP '192.168.1. ▸ := '192.168.1.40',             //IP-Address of server to connect to
    Port_Recv 3000 := 3000,                        //Port number of UDP socket to open
    DataSize 10 := 10, //SIZEOF(Data)                        //Maximum size of the pBuffer parameter in bytes
    DataPtr_Send 2094621216192 := ADR(Data),                  //Pointer to a buffer to get the received data from the port
    Timeout:=,                                  //Defines the time (µs) after which the connection setup aborts with an error message.
    Done=> ,                                    //Function block completion
    Busy=> ,                                    //Function block active
    xError=> ,                                  //Function block error
    Error_ID=> );                               //
RETURN
```

8.3.3 ICP_UDP_Comm_Receive

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

ICP_UDP_Comm_Receive	Receive UDP communication data		ICP_UDP_Comm_Receive(
			Enable:=,
			RecvIP:=,
			Port_Recv:=,
			DataSize:=,
			DataPtr_Recv:=,
			Done=>,
			Busy=>,
			xError=>,
			Error_ID=>,
			xReady=>,
			IpFrom=>,
			PortFrom=>,
			Count=>);

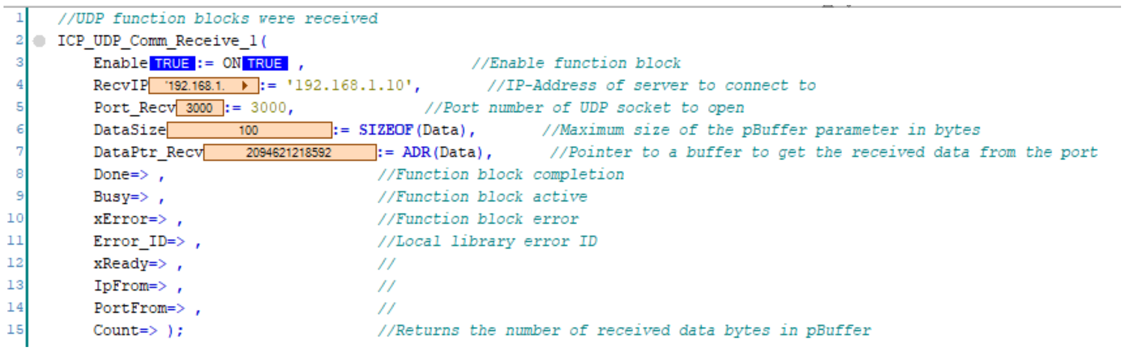
Associated variables:

Input Type	Name	Data Type	Valid Range	Initial Value	Description
Enable	Enabled	BOOL	TRUE, FALSE	FALSE	If it is TRUE, the function block is enabled
Recv_IP	Receiving IP	STRING	-	-	Receiving controller IP
Port_Recv	Receiving port	UINT	-	-	Receiving controller port
DataSize	Data length	CAA.SIZE	-	-	Data length to be sent
DataPtr_Recv	Data storage	CAA.PVOID	-	-	Received data storage address
Output Type	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Done	Completed flag	BOOL	TRUE, FALSE	FALSE	Completed signal
Busy	Function block running	BOOL	TRUE, FALSE	FALSE	Run flag
xError	Error flag	BOOL	TRUE, FALSE	FALSE	Sending error
Error_ID	Error ID	NBS.ERROR	-	0	Error code
xReady	Succeeded flag	BOOL	TRUE, FALSE	FALSE	Receiving succeeded flag
IpFrom	Sending IP	NBS.IP_ADDR	-	-	Data sending controller IP
PortFrom	Sending port	UINT	-	-	Data sending controller

					port
Count	Data length	CAA.SIZE	-	-	Actual length of received data

Program example:

To receive UDP communication data, when the input variable of the ICP_UDP_Comm_Receive instruction is TRUE, data will be read from the UDP communication buffer and Busy is TRUE. If the data is read successfully, Done is set for one scan cycle; the read data will be placed in the variable with the address DataPtr_Recv; at the same time, xReady is set for one scan cycle; the actual size value of the received data area will be assigned to Count, and the value of Count will be cleared after one scan cycle.



```
//UDP function blocks were received
ICP_UDP_Comm_Receive_1(
    Enable TRUE := ON TRUE ,                      //Enable function block
    RecvIP '192.168.1. := '192.168.1.10',         //IP-Address of server to connect to
    Port_Recv 3000 := 3000,                  //Port number of UDP socket to open
    DataSize 100 := SIZEOF(Data),                 //Maximum size of the pBuffer parameter in bytes
    DataPtr_Recv 2094621218592 := ADR(Data),      //Pointer to a buffer to get the received data from the port
    Done=> ,                        //Function block completion
    Busy=> ,                        //Function block active
    xError=> ,                      //Function block error
    Error_ID=> ,                    //Local library error ID
    xReady=> ,                      //
    IpFrom=> ,                      //
    PortFrom=> ,                    //
    Count=> );                      //Returns the number of received data bytes in pBuffer
```

9 Pulse Output Instructions

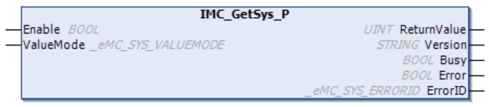
9.1 Auxiliary Instructions

Note: The content of this chapter is only applicable to the TM series PLC.

9.1.1 IMC_GetSys_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_GetSys_P	Get system information	 IMC_GetSys_P
		Enable BOOL
		ValueMode _eMC_SYS_VALUEMODE
		UINT ReturnValue
		STRING Version
		BOOL Busy
		BOOL Error
		_eMC_SYS_ERRORID ErrorID
			IMC_GetSys_P(
			Enable:= ,
			ValueMode:= ,
			ReturnValue=>
			Version=> ,
			Busy=> ,
			Error=> ,
			ErrorID=>);

Associated variables:

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Function block enabling bit	BOOL	TRUE, FALSE	TRUE	Valid if it is TRUE, and invalid if it is FALSE
ValueMode	Value group type	_eMc_Sys_ValueMode	_mcSysAxisNum, _mcSysGroupNum, _mcSysCAMNum, _mcSysGearNum, _mcSysFlyCutNum, _mcSysTraceNum	_mcSysAxisNum	Value group type, the number of axes supported by the system: _mcSysAxisNum; the number of axis groups supported by the system: _mcSysGroupNum; the number of cam tables supported by the system: _mcSysCAMNum; the number of electronic gear groups supported by the system: _mcSysGearNum; the number of flying shear groups supported by the system: _mcSysFlyCutNum; the number of tracking shear groups supported by the system: _mcSysTraceNum

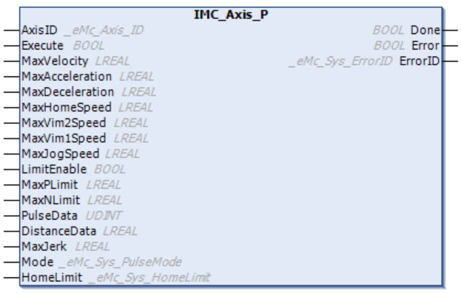
Output Variable	Name	Data Type	Valid Range	Initial Value	Description
ReturnValue	Return value	UINT	-	0	Return value, for example: 4 (axis), 2 (axis group), 3 (number of cam/gear/flying shear/tracking shear groups)
Version	Version	STRING	-	V0.0.0.1	Return the version number
Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE - The function block is being executed FALSE - The function block is not executed
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE - error, FALSE - no error
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError_NULL	Error code

Function description: This function block is used to get system information.

9.1.2 IMC_Axis_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_Axis_P	Axis parameter configuration	 IMC_Axis_P
		AxisID _eMc_Axis_ID
		Execute BOOL
		MaxVelocity LREAL
		MaxAcceleration LREAL
		MaxDeceleration LREAL
		MaxHomeSpeed LREAL
		MaxVim2Speed LREAL
		MaxVim1Speed LREAL
		MaxJogSpeed LREAL
		LimitEnable BOOL
		MaxPLimit LREAL
		MaxNLimit LREAL
		PulseData UDINT
		DistanceData LREAL
		MaxJerk LREAL
		Mode _eMc_Sys_PulseMode
		HomeLimit _eMc_Sys_HomeLimit
		BOOL Done
		BOOL Error
		_eMc_Sys_ErrorID ErrorID
			IMC_Axis_P(
			AxisID:= ,
			Execute:= ,
			MaxVelocity:= ,
			MaxAcceleration:= ,
			MaxDeceleration:= ,
			MaxHomeSpeed:= ,
			MaxVim2Speed:= ,
			MaxVim1Speed:= ,
			MaxJogSpeed:= ,
			LimitEnable:= ,
			MaxPLimit:= ,
			MaxNLimit:= ,
			PulseData:= ,
			DistanceData:= ,
			MaxJerk:= ,
			Mode:= ,
			HomeLimit:= ,
			Done=> ,
			Error=> ,
			ErrorID=>);

Associated variables:

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axle number	_eMc_Axis_ID	0–3	255	Axis number (Note: The initial value 255 is a protection measure to avoid the situation where no initial value is assigned, the same below)
Execute	Function block enabling bit	BOOL	TRUE, FALSE	FALSE	Valid at the rising edge, and invalid at the falling edge
MaxVelocity	Max. running speed of the axis	LREAL	>1	5000	Max. running speed of the current axis (unit/s), set by the user
MaxAcceleration	Max. running acceleration of the axis	LREAL	>0	500	Max. allowable acceleration of the current axis (unit/s^2)
MaxDeceleration	Max. running deceleration of the axis	LREAL	>0	500	Max. allowable deceleration of the current axis (unit/s^2)
MaxHomeSpeed	Max. homing speed of the axis	LREAL	1–80	10	Max. homing speed of the current axis (unit/s)
MinVim2Speed	Max. homing speed at step 2	LREAL	1–2	1	Max. homing speed at step 2 of the current axis (unit/s)
MaxVim1Speed	Max. homing speed at stage 1	LREAL	1–10	3	Max. homing speed at step 1 of the current axis (unit/s)
MaxJogSpeed	Max. jogging speed of the axis	LREAL	1–5000	5000	Max. jogging speed of the current axis (unit/s)
LimitEnable	Soft limit enabling flag	BOOL	TRUE, FALSE	TRUE	TRUE->Soft limit on; FALSE->Soft limit off (not considered in single-axis speed mode)
MaxPLimit	Max. positive limit position	LREAL	>0	999999999.999	Max. positive limit position of the current axis (unit), not considered in single-axis speed mode
MaxNLimit	Max. negative limit position	LREAL	<0	-999999999.999	Max. negative limit position of the current axis (unit), not considered in single-axis speed mode

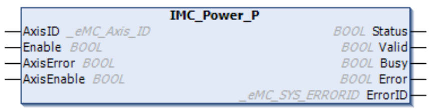
Input Variable	Name	Data Type	Valid Range	Initial Value	Description
PulseData	Number of pulses required for one revolution of the current axis	UDINT	>0	10000	Number of pulses required for one revolution of the current axis (unit/pulse)
DistanceData	Running distance of one revolution of the current axis	LREAL	>0	10	Running distance of one revolution of the current axis (unit/mm)
MaxJerk	Max. jerk of the current axis	LREAL	10–400	100	Max. jerk of the current axis (unit/s^3)
Mode	Pulse control mode of the current axis	_eMc_Sys_PulseMode	_mcPulseSign, _mcCWCCW, _mcQEP	_mcPulseSign	Pulse control mode of the current axis: _mcPulseSign: "pulse + sign" mode; _mcCWCCW: FWD/REV pulse train mode; _mcQEP: quadrature-encoded pulse mode
HomeLimit	Stop when hard limit is detected during homing	_eMc_Sys_HomeLimit	_mcLimitDecStop,_mcLimitImmediatelyStop	_mcLimitDecStop	_mcLimitDecStop: Decelerate to stop _mcLimitImmediatelyStop: Stop immediately
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Done	Function block completed state	BOOL	TRUE, FALSE	FALSE	TRUE - Axis definition completed FALSE - Axis definition not completed
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE - error, FALSE - no error
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcErr, _NULL	Error ID

Function description: This function block is used to set the associated parameters of the instruction axis. The pulse type only supports 4 axes.

9.1.3 IMC_Power_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_Power_P	Axis enabling	 IMC_Power_P
		AxisID _eMC_Axis_ID
		Enable BOOL
		AxisError BOOL
		AxisEnable BOOL
		BOOL Status
		BOOL Valid
		BOOL Busy
		BOOL Error
		_eMC_SYS_ERRORID ErrorID
			IMC_Power_P(
			AxisID:= ,
			Enable:= ,
			AxisError:= ,
			AxisEnable:= ,
			Status=> ,
			Valid=> ,
			Busy=> ,
			Error=> ,
			ErrorID=>);

Associated variables:

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axle number	_eMc_Axis_ID	0-3	255	Axle number
Enable	Function block enabling bit	BOOL	TRUE, FALSE	TRUE	Valid if it is TRUE, and invalid if it is FALSE
AxisError	Axis alarm signal	BOOL	TRUE, FALSE	FALSE	TRUE: Report an alarm FALSE: Do not report an alarm
AxisEnable	Axis enabling signal	BOOL	TRUE, FALSE	TRUE	TRUE: The axis has been enabled FALSE: The axis is not enabled
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Status	Axis status	BOOL	TRUE, FALSE	FALSE	TRUE: The axis is ready FALSE: The axis is not ready
Valid	Axis enabling state	BOOL	TRUE, FALSE	FALSE	TRUE: The axis has been enabled TRUE: The axis is not enabled
Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE: The function block is being executed FALSE: The function block is not executed
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE: An error occurs FALSE: No error occurs
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError_NULL	Error code

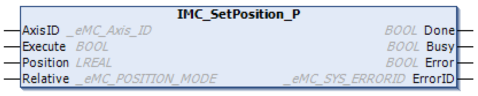
Function description: If Enable is TRUE, the function block is executed; if AxisEnable is TRUE, the axis is enabled.

Note: This function block can only enable an internal axis.

9.1.4 IMC_SetPosition_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_SetPosition_P	Set the current instruction position of the axis	 IMC_SetPosition_P
		AxisID _eMC_Axis_ID
		Execute BOOL
		Position LREAL
		Relative _eMC_POSITION_MODE
		BOOL Done
		BOOL Busy
		BOOL Error
		_eMC_SYS_ERRORID ErrorID
			IMC_SetPosition_P(
			AxisID:= ,
			Execute:= ,
			Position:= ,
			Relative:= ,
			Done=> ,
			Busy=> ,
			Error=> ,
			ErrorID=>);

Associated variables:

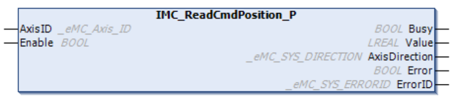
Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axle number	_eMc_Axis_ID	0-3	255	Axle number
Execute	Function block enabling bit	BOOL	TRUE, FALSE	FALSE	Valid at the rising edge, and invalid at the falling edge
Position	Position value	LREAL	Real number	0	Current position setting value
Relative	Position mode	_eMc_Position_Mode	_mcAbsolute, _mcRelative	_mcAbsolute	_mcAbsolute - Absolute mode _mcRelative - Relative mode
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Done	Function block completed state	BOOL	TRUE, FALSE	FALSE	TRUE - Position setting is completed FALSE - Position setting is not completed
Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE - The function block is being executed FALSE - The function block is not executed
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE - error, FALSE - no error
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError_NULL	Error code

Function description: This function block is triggered at the rising edge of Execute. It indicates the absolute position mode when Relative is set to _mcAbsolute, and the relative position mode when Relative is set to _mcRelative.

9.1.5 IMC_ReadCmdPosition_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_ReadCmdPosition_P	Read the actual position of the axis	 IMC_ReadCmdPosition_P
		AxisID _eMC_Axis_ID
		Enable BOOL
		BOOL Busy
		LREAL Value
		_eMC_SYS_DIRECTION AxisDirection
		BOOL Error
		_eMC_SYS_ERRORID ErrorID
			`IMC_ReadCmdPosition_P(`
			`AxisID:= ,`
			`Enable:= ,`
			`Busy=> ,`
			`Value=> ,`
			`AxisDirection=> ,`
			`Error=> ,`
			`ErrorID=> );`

Associated variables:

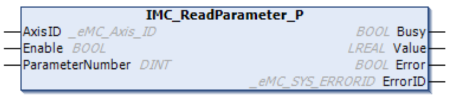
Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axle number	_eMc_Axis_ID	0–3	255	Axle number
Enable	Function block enabling bit	BOOL	TRUE, FALSE	TRUE	Valid if it is TRUE, and invalid if it is FALSE
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE: The function block is being executed FALSE: The function block is not executed
Value	Position value	LREAL	Real number	0	Read the actual position value of the specified axis
AxisDirection	Axis direction	_eMc_Sys_Direction	_mcNegative, _mcForward	_mcNegative	_mcNegative: Negative _mcForward: Forward
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE: An error occurs FALSE: No error occurs
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError_NULL	Error code

Function description: This function block is triggered at the rising edge of Enable to read the actual position of the specified axis, and Value returns the actual position value of the axis.

9.1.6 IMC_ReadParameter_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_ReadParameter_P	Read axis definition parameters	 IMC_ReadParameter_P
		AxisID _eMC_Axis_ID
		Enable BOOL
		ParameterNumber DINT
		BOOL Busy
		LREAL Value
		BOOL Error
		_eMC_SYS_ERRORID ErrorID
			`IMC_ReadParameter_P(`
			`AxisID:= ,`
			`Enable:= ,`
			`ParameterNumber:= ,`
			`Busy=> ,`
			`Value=> ,`
			`Error=> ,`
			`ErrorID=> );`

Associated variables:

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axle number	_eMc_Axis_ID	0–3	255	Axle number

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Function block enabling bit	BOOL	TRUE, FALSE	TRUE	Valid if it is TRUE Invalid if it is False
ParameterNumber	Parameter number	DINT	1000-1012	1000	Axis definition parameter number For example: 1000 is the max. running speed
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE: The function block is being executed FALSE: The function block is not executed
Value	Return value	LREAL	1000-1012	0	Axis definition parameter output
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE: An error occurs, FALSE: No error occurs
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcErr, _NULL	Error code

Function description: valid if it is TRUE. Read the corresponding value according to the parameter sequence number listed below.

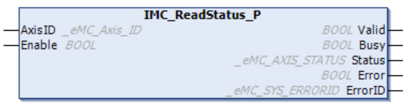
The parameter sequence number is as follows:

Sequence Number	Name	Description
1000	MaxVelocity	Max. running speed of the current axis (unit/s)
1001	MaxAcceleration	Max. allowable acceleration of the current axis (unit/s^2)
1002	MaxHomeSpeed	Max. homing speed of the current axis (unit/s)
1003	MinVim2Speed	Max. homing speed at step 2 of the current axis (unit/s)
1004	MaxVim1Speed	Min. homing speed at step 2 of the current axis (unit/s)
1005	MaxJogSpeed	Max. jogging speed of the current axis (unit/s)
1006	LimitEnable	Soft limit enabling flag TRUE: soft limit on; FALSE: soft limit off
1007	MaxPLimit	Max. positive limit position of the current axis (unit)
1008	MaxNLimit	Max. negative limit position of the current axis (unit)
1009	PulseData	Number of pulses required for one revolution of the current axis (unit/pulse)
1010	DistanceData	Running distance of one revolution of the current axis (unit/mm)
1011	MaxJerk	Max. allowable jerk of the current axis
1012	Mode	Pulse control mode of the current axis: _mcPulseSign: “pulse + sign” mode _mcCWCCW: FWD/REV pulse train mode _mcQEP: quadrature-encoded pulse mode

9.1.7 IMC_ReadStatus_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_ReadStatus_P	Read the running status of the axis	 IMC_ReadStatus_P
		AxisID _eMC_Axis_ID
		Enable BOOL
		BOOL Valid
		BOOL Busy
		_eMC_AXIS_STATUS Status
		BOOL Error
		_eMC_SYS_ERRORID ErrorID
			`IMC_ReadStatus_P(`
			`AxisID:= ,`
			`Enable:= ,`
			`Valid=> ,`
			`Busy=> ,`
			`Status=> ,`
			`Error=> ,`
			`ErrorID=> );`

Associated variables:

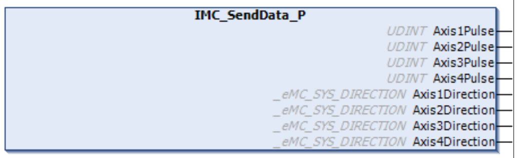
Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axle number	_eMc_Axis_ID	0–3	255	Axle number
Enable	Function block enabling bit	BOOL	TRUE, FALSE	TRUE	Valid if it is TRUE, and invalid if it is FALSE
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Valid	Validity status	BOOL	TRUE, FALSE	FALSE	TRUE - Valid output, FALSE - Invalid output
Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE - The function block is being executed FALSE - The function block is not executed
Status	Axis status	_eMc_Axis_Status	-	_mcDisabled	Return the status of the axis
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE - error, FALSE - no error
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError_NULL	Error code

Function description: If it is TRUE, the function block returns the current state of the axis.

9.1.8 IMC_SendData_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_SendData_P	Send data	 IMC_SendData_P
		UDINT Axis1Pulse
		UDINT Axis2Pulse
		UDINT Axis3Pulse
		UDINT Axis4Pulse
		_eMC_SYS_DIRECTION Axis1Direction
		_eMC_SYS_DIRECTION Axis2Direction
		_eMC_SYS_DIRECTION Axis3Direction
		_eMC_SYS_DIRECTION Axis4Direction
			`IMC_SendData_P(`
			`Axis1Pulse=> ,`
			`Axis2Pulse=> ,`
			`Axis3Pulse=> ,`
			`Axis4Pulse=> ,`
			`Axis1Direction=> ,`
			`Axis2Direction=> ,`
			`Axis3Direction=> ,`
			`Axis4Direction=> );`

Associated variables:

Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis1Pulse	Axis 1 pulse feedback	UINT	Positive number	0	Axis 1 pulse feedback
Axis2Pulse	Axis 2 pulse feedback	UINT	Positive number	0	Axis 2 pulse feedback

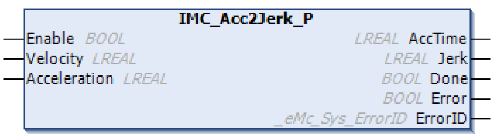
Output Variable	Name	Data Type	Valid Range	Initial Value	Description
Axis3Pulse	Axis 3 pulse feedback	UINT	Positive number	0	Axis 3 pulse feedback
Axis4Pulse	Axis 4 pulse feedback	UINT	Positive number	0	Axis 4 pulse feedback
Axis1Direction	Axis 1 direction	_eMc_Sys_Direction	_mcNegative, mcForward	_mcNegative	_mcNegative: Negative _mcForward: Forward
Axis2Direction	Axis 2 direction	_eMc_Sys_Direction	_mcNegative, mcForward	_mcNegative	_mcNegative: Negative _mcForward: Forward
Axis3Direction	Axis 3 direction	_eMc_Sys_Direction	_mcNegative, mcForward	_mcNegative	_mcNegative: Negative _mcForward: Forward
Axis4Direction	Axis 4 direction	_eMc_Sys_Direction	_mcNegative,_ mcForward	_mcNegative	_mcNegative: Negative _mcForward: Forward

Function description: This function block is used to send data to the underlayer. This module must be added after all motion control function blocks are called in pulse mode.

9.1.9 IMC_Acc2Jerk_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_Acc2Jerk_P	Calculate the jerk from the axis acceleration		IMC_Acc2Jerk_P(
			Enable:= ,
			Velocity:= ,
			Acceleration:= ,
			AccTime=> ,
			Jerk=> ,
			Done=> ,
			Error=> ,
			ErrorID=>);

Associated variables:

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Function block enabling bit	BOOL	TRUE, FALSE	TRUE	Valid if it is TRUE, and invalid if it is FALSE
Velocity	Speed	LREAL	Positive number	100	Target velocity
Acceleration	Acceleration	LREAL	Positive number	10	Target acceleration, unit/s^2
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
AccTime	ACC time	LREAL	Positive number	0	Acceleration time, s
Jerk	Jerk	LREAL	Positive number	0	Jerk, unit/s^3
Done	Function block	BOOL	TRUE, FALSE	FALSE	TRUE: The function block

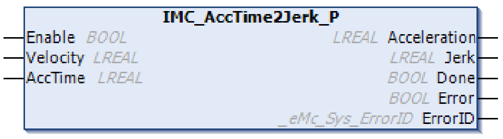
	completed state				has been executed FALSE: The function block is not executed
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE: An error occurs FALSE: No error occurs
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError_NULL	Error code

Function description: This function block is used to calculate the AccTime and Jerk values according to the speed and acceleration values.

9.1.10 IMC_AccTime2Jerk_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_AccTime2Jerk_P	Calculate the jerk from the acceleration time	 IMC_AccTime2Jerk_P
		Enable BOOL
		Velocity LREAL
		AccTime LREAL
		LREAL Acceleration
		LREAL Jerk
		BOOL Done
		BOOL Error
		_eMc_Sys_ErrorID ErrorID
			IMC_AccTime2Jerk_P(
			Enable:= ,
			Velocity:= ,
			AccTime:= ,
			Acceleration=> ,
			Jerk=> ,
			Done=> ,
			Error=> ,
			ErrorID=>);

Associated variables:

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
Enable	Function block enabling bit	BOOL	TRUE, FALSE	TRUE	Valid if it is TRUE, and invalid if it is FALSE
Velocity	Speed	LREAL	Positive number	100	Target speed, unit/s
AccTime	ACC time	LREAL	Positive number	1	Target acceleration time, s
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Acceleration	Acceleration	LREAL	Positive number	0	Target acceleration, unit/s^2
Jerk	Jerk	LREAL	Positive number	0	Jerk, unit/s^3
Done	Function block completed state	BOOL	TRUE, FALSE	FALSE	TRUE: The function block has been executed FALSE: The function block is not executed
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE: An error occurs FALSE: No error occurs
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError_NULL	Error code

Function description: This function block is used to calculate the Acceleration and Jerk values according to the speed and acceleration time values.

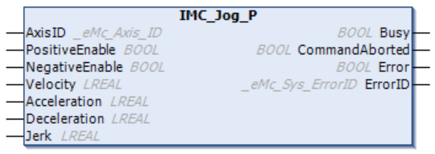
9.2 Single Axis Instructions

Note: The content of this chapter is only applicable to the TM series PLC.

9.2.1 IMC_Jog_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_Jog_P	Continuous jogging	 IMC_Jog_P
		AxisID _eMc_Axis_ID
		PositiveEnable BOOL
		NegativeEnable BOOL
		Velocity LREAL
		Acceleration LREAL
		Deceleration LREAL
		Jerk LREAL
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		_eMc_Sys_ErrorID ErrorID
			`IMC_Jog_P(`
			`AxisID:= ,`
			`PositiveEnable:= ,`
			`NegativeEnable:= ,`
			`Velocity:= ,`
			`Acceleration:= ,`
			`Deceleration:= ,`
			`Jerk:= ,`
			`Busy=> ,`
			`CommandAborted=> ,`
			`Error=> ,`
			`ErrorID=> );`

Associated variables:

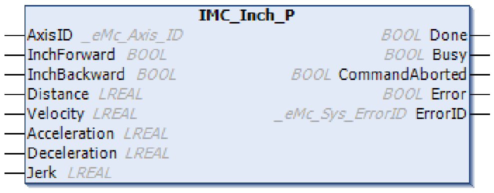
Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axle number	_eMc_Axis_ID	0–3	255	Axle number
PositiveEnable	Forward running	BOOL	TRUE, FALSE	FALSE	Start at the rising edge, and stop at the falling edge
NegativeEnable	Backward running	BOOL	TRUE, FALSE	FALSE	Start at the rising edge, and stop at the falling edge
Velocity	Speed	LREAL	Positive number	10	Axis running speed (unit/s)
Acceleration	Acceleration	LREAL	Positive number	10	Axis running acceleration (unit/s^2)
Deceleration	Deceleration	LREAL	Positive number	10	Axis running deceleration (unit/s^2)
Jerk	Jerk	LREAL	Positive or 0	0	Jerk (unit/s^3)
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE: The function block is being executed FALSE: The function block is not executed
CommandAborted	Function block aborted	BOOL	TRUE, FALSE	FALSE	TRUE: Aborted FALSE: Not aborted
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE: An error occurs FALSE: No error occurs
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError_NULL	Error code

Function description: The axis is standstill initially. The PositiveEnable parameter controls the start and stop of the axis in the forward direction, and the NegativeEnable parameter controls the start and stop of the axis in the backward direction.

9.2.2 IMC_Inch_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_Inch_P	Inching	 IMC_Inch_P
		AxisID _eMc_Axis_ID
		InchForward BOOL
		InchBackward BOOL
		Distance LREAL
		Velocity LREAL
		Acceleration LREAL
		Deceleration LREAL
		Jerk LREAL
		BOOL Done
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		_eMc_Sys_ErrorID ErrorID
			`IMC_Inch_P(`
			`AxisID:= ,`
			`InchForward:= ,`
			`InchBackward:= ,`
			`Distance:= ,`
			`Velocity:= ,`
			`Acceleration:= ,`
			`Deceleration:= ,`
			`Jerk:= ,`
			`Done=> ,`
			`Busy=> ,`
			`CommandAborted=> ,`
			`Error=> ,`
			`ErrorID=> );`

Associated variables:

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axle number	_eMc_Axis_ID	0–3	255	Axle number
InchForward	Forward running	BOOL	TRUE, FALSE	FALSE	Start at the rising edge, and stop at the falling edge
InchBackward	Backward running	BOOL	TRUE, FALSE	FALSE	Start at the rising edge, and stop at the falling edge
Distance	Distance	LREAL	Real number	0.5	Axis running distance (unit)
Velocity	Speed	LREAL	Positive number	10	Axis running speed (unit/s)
Acceleration	Acceleration	LREAL	Positive number	10	Axis running acceleration (unit/s^2)
Deceleration	Deceleration	LREAL	Positive number	10	Axis running deceleration (unit/s^2)
Jerk	Jerk	LREAL	Positive or 0	0	Jerk (unit/s^3)
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Done	Function block completed state	BOOL	TRUE, FALSE	FALSE	TRUE: The inching operation is completed, FALSE: The inching operation is not completed
Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE: The function block is being executed FALSE: The function block is not executed

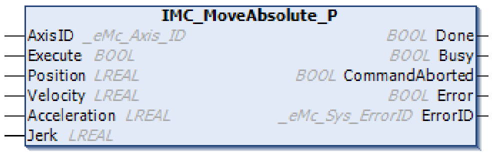
CommandAborted	Function block aborted	BOOL	TRUE, FALSE	FALSE	TRUE: Aborted FALSE: Not aborted
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE: An error occurs FALSE: No error occurs
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError_NULL	Error ID

Function description: The InchForward parameter controls the inching movement in the forward direction, the InchBackward parameter controls the inching movement in the backward direction, and it supports S-type acceleration and deceleration.

9.2.3 IMC_MoveAbsolute_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_MoveAbsolute_P	Single axis absolute motion	 IMC_MoveAbsolute_P
		AxisID _eMc_Axis_ID
		Execute BOOL
		Position LREAL
		Velocity LREAL
		Acceleration LREAL
		Jerk LREAL
		BOOL Done
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		_eMc_Sys_ErrorID ErrorID
			IMC_MoveAbsolute_P(
			AxisID:= ,
			Execute:= ,
			Position:= ,
			Velocity:= ,
			Acceleration:= ,
			Jerk:= ,
			Done=> ,
			Busy=> ,
			CommandAborted=> ,
			Error=> ,
			ErrorID=>);

Associated variables:

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axle number	_eMc_Axis_ID	0–3	255	Axle number
Execute	Function block enabling bit	BOOL	TRUE, FALSE	FALSE	Valid at the rising edge, and invalid at the falling edge
Position	Absolute position	LREAL	Real number	0	Absolute running distance of the axis (unit)
Velocity	Running speed	LREAL	Positive number	10	Speed (unit/s)
Acceleration	Acceleration	LREAL	Positive number	10	Axis running acceleration (unit/s^2)
Jerk	JerK	LREAL	Positive or 0	0	Jerk (unit/s^3)
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Done	Function block completed state	BOOL	TRUE, FALSE	FALSE	TRUE: The function block has been executed FALSE: The function block is not executed
Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE: The function block is being executed FALSE: The function block is not executed

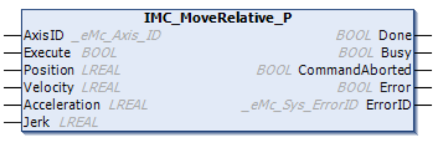
CommandAborted	Function block aborted	BOOL	TRUE, FALSE	FALSE	TRUE: Aborted FALSE: Not aborted
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE: An error occurs FALSE: No error occurs
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError_NULL	Error ID

Function description: This function block is triggered at the rising edge of Execute. The position instruction runs in absolute position mode and supports trapezoidal and S-curve acceleration and deceleration.

9.2.4 IMC_MoveRelative_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_MoveRelative_P	Single axis relative motion	 IMC_MoveRelative_P
		AxisID _eMc_Axis_ID
		Execute BOOL
		Position LREAL
		Velocity LREAL
		Acceleration LREAL
		Jerk LREAL
		BOOL Done
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		_eMc_Sys_ErrorID ErrorID
			`IMC_MoveRelative_P(`
			`AxisID:= ,`
			`Execute:= ,`
			`Position:= ,`
			`Velocity:= ,`
			`Acceleration:= ,`
			`Jerk:= ,`
			`Done=> ,`
			`Busy=> ,`
			`CommandAborted=> ,`
			`Error=> ,`
			`ErrorID=> );`

Associated variables:

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axle number	_eMc_Axis_ID	0–3	255	Axle number
Execute	Function block enabling bit	BOOL	TRUE, FALSE	FALSE	Valid at the rising edge, and invalid at the falling edge
Position	Relative position	LREAL	Real number	0	Relative running distance of the axis (unit)
Velocity	Running speed	LREAL	Positive number	10	Speed (unit/s)
Acceleration	Acceleration	LREAL	Positive number	10	Axis running acceleration (unit/s^2)
Jerk	Jerk	LREAL	Positive or 0	0	Jerk (unit/s^3)
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Done	Function block completed state	BOOL	TRUE, FALSE	FALSE	TRUE: The function block has been executed FALSE: The function block is not executed
Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE: The function block is being executed FALSE: The function block is not executed

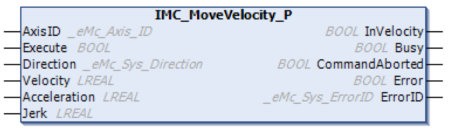
CommandAborted	Function block aborted	BOOL	TRUE, FALSE	FALSE	TRUE: Aborted FALSE: Not aborted
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE: An error occurs FALSE: No error occurs
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError_NULL	Error ID

Function description: This function block is triggered at the rising edge of Execute. The position instruction runs in relative position mode and supports trapezoidal and S-curve acceleration and deceleration.

9.2.5 IMC_MoveVelocity_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_MoveVelocity_P	Speed mode	 IMC_MoveVelocity_P
		AxisID _eMc_Axis_ID
		Execute BOOL
		Direction _eMc_Sys_Direction
		Velocity LREAL
		Acceleration LREAL
		Jerk LREAL
		BOOL InVelocity
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		_eMc_Sys_ErrorID ErrorID
			IMC_MoveVelocity_P(
			    AxisID:= ,
			    Execute:= ,
			    Direction:= ,
			    Velocity:= ,
			    Acceleration:= ,
			    Jerk:= ,
			    InVelocity=> ,
			    Busy=> ,
			    CommandAborted=> ,
			    Error=> ,
			    ErrorID=>);

Associated variables:

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axle number	_eMC_Axis_ID	0–3	255	Axle number
Execute	Function block enabling bit	BOOL	TRUE, FALSE	FALSE	Valid at the rising edge, and invalid at the falling edge
Direction	Axis motion direction	_eMc_Sys_Direction	_mcNegative,_mcForward	_mcNegative	_mcNegative: Backward running, _mcForward: Forward running
Velocity	Running speed	LREAL	Positive number	10	Speed (unit/s)
Acceleration	Acceleration	LREAL	Positive number	10	Axis running acceleration (unit/s^2)
Jerk	Jerk	LREAL	Positive or 0	0	Jerk (unit/s^3)
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
InVelocity	Function block completed state	BOOL	TRUE, FALSE	FALSE	The instruction speed value is reached for the first time
Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE: The function block is being executed FALSE: The function block is not executed
Command Aborted	Function block aborted	BOOL	TRUE, FALSE	FALSE	TRUE: Aborted FALSE: Not aborted

Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE: An error occurs FALSE: No error occurs
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError_NULL	Error ID

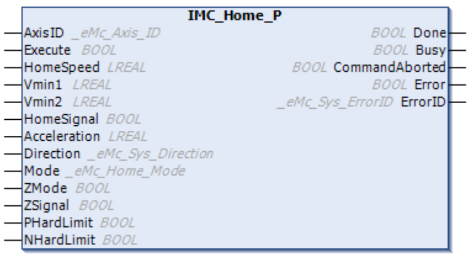
Function description:

This function block is triggered at the rising edge of Execute to control the axis to run at a constant speed.

9.2.6 IMC_Home_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

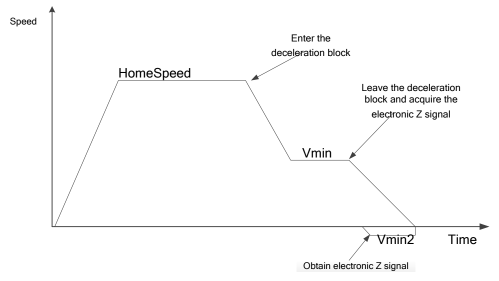
IMC_Home_P	Home	 IMC_Home_P
		AxisID _eMc_Axis_ID
		Execute BOOL
		HomeSpeed LREAL
		Vmin1 LREAL
		Vmin2 LREAL
		HomeSignal BOOL
		Acceleration LREAL
		Direction _eMc_Sys_Direction
		Mode _eMc_Home_Mode
		ZMode BOOL
		ZSignal BOOL
		PHardLimit BOOL
		NHardLimit BOOL
		BOOL Done
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		_eMc_Sys_ErrorID ErrorID
			`IMC_Home_P(`
			`AxisID:= ,`
			`Execute:= ,`
			`HomeSpeed:= ,`
			`Vmin1:= ,`
			`Vmin2:= ,`
			`HomeSignal:= ,`
			`Acceleration:= ,`
			`Direction:= ,`
			`Mode:= ,`
			`ZMode:= ,`
			`ZSignal:= ,`
			`PHardLimit:= ,`
			`NHardLimit:= ,`
			`Done=> ,`
			`Busy=> ,`
			`CommandAborted=> ,`
			`Error=> ,`
			`ErrorID=> );`

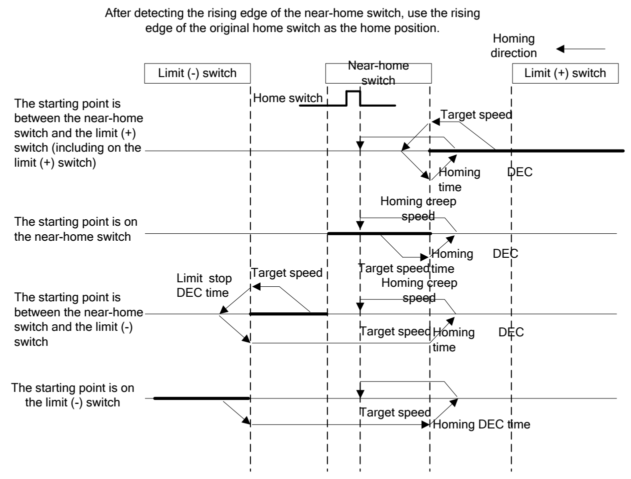
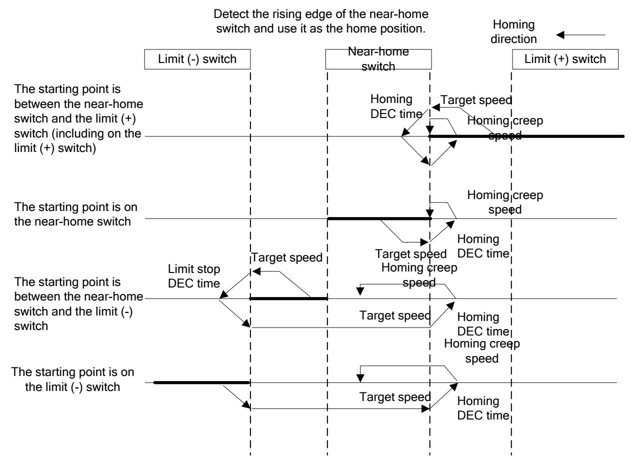
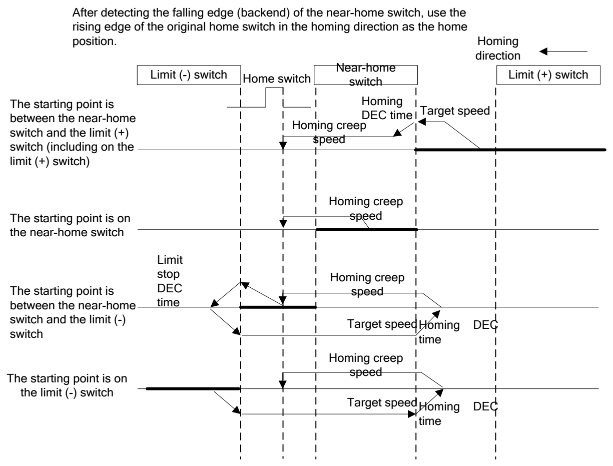
Associated variables:

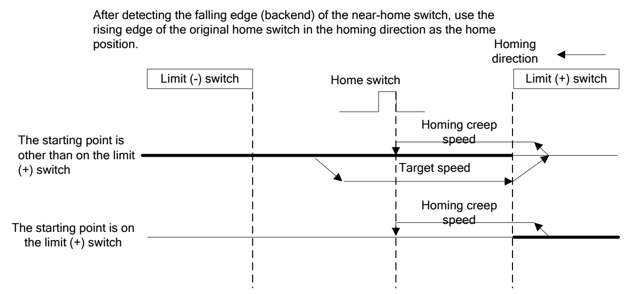
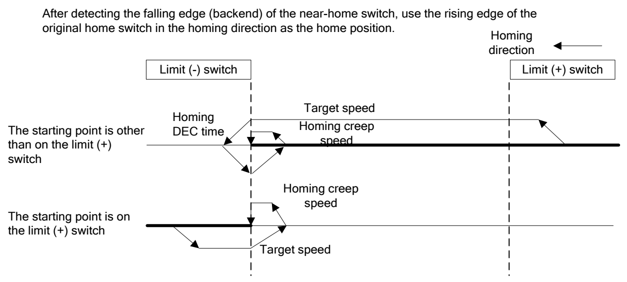
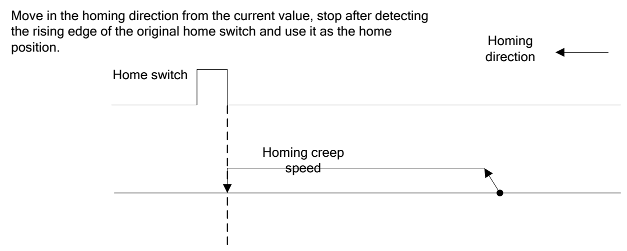
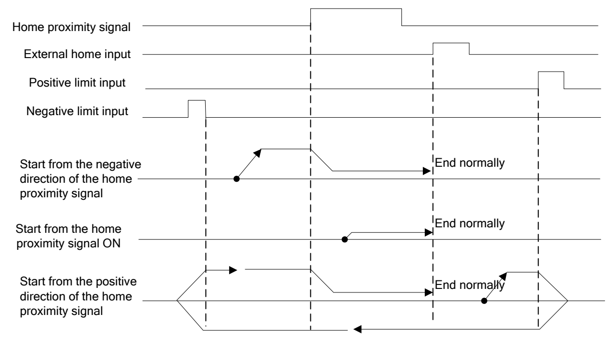
Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axle number	_eMc_Axis_ID	0–3	255	Axle number
Execute	Function block enabling bit	BOOL	TRUE, FALSE	FALSE	Valid at the rising edge, and invalid at the falling edge
HomeSpeed	Homing speed	LREAL	Positive number	15	Homing speed (unit/s)
Vmin1	Running speed at the deceleration block	LREAL	Positive number	3	Running speed at the deceleration block (unit/s)
Vmin2	Running speed when waiting for the Z signal of motor	LREAL	Positive number	1	Running speed when waiting for the Z signal of motor (unit/s)
HomeSignal	Homing signal	BOOL	TRUE, FALSE	FALSE	Homing signal
Acceleration	Acceleration	LREAL	Positive number	10	Acceleration (unit/s^2)
Direction	Homing direction	_eMc_Sys_Direction	_mcNegative, _mcForward	_mcForward	_mcNegative: Backward running _mcForward: Forward running
Mode	Homing mode	_eMc_Home_Mode	1–9	2	Homing mode

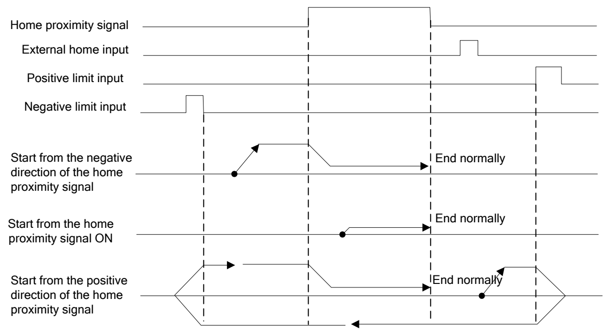
Input Variable	Name	Data Type	Valid Range	Initial Value	Description
ZMode	Z signal acquisition mode	BOOL	TRUE, FALSE	FALSE	FALSE: Z signal input internally (motor) TRUE: Z signal input externally
ZSignal	External Z signal	BOOL	TRUE, FALSE	FALSE	ZMode is valid when it is TRUE, and the Z signal is valid at the rising edge of ZSignal
PHardLimit	Positive hard limit signal	BOOL	TRUE, FALSE	FALSE	Positive hard limit signal
NHardLimit	Negative hard limit signal	BOOL	TRUE, FALSE	FALSE	Negative hard limit signal
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Done	Function block completed state	BOOL	TRUE, FALSE	FALSE	TRUE: Homing completed FALSE: Homing not completed
Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE: The function block is being executed FALSE: The function block is not executed
CommandAborted	Function block aborted	BOOL	TRUE, FALSE	FALSE	TRUE - Aborted FALSE: Not aborted
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE: An error occurs FALSE: No error occurs
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError_NULL	Error ID

Function description: This function block is used to control the internal current axis to return to the home position, with 9 homing modes.

Mode	Timing Diagram	Brief Description
Mode 1	Near-home switch pulse edge detection rear-end reference + home switch (Z signal)  Speed HomeSpeed Enter the deceleration block Leave the deceleration block and acquire the electronic Z signal Vmin Vmin2 Time Obtain electronic Z signal	For example: The axis accelerates to HomeSpeed → A signal that it hits the deceleration block is given → The axis decelerates to Vmin → The axis leaves the deceleration block and the motor Z signal is acquired → The axis decelerates to 0 and then accelerates to Vmin2 reversely → The motor Z signal is acquired → The homing operation is completed

Mode	Timing Diagram	Brief Description
Mode 2	Near-home switch pulse edge detection front-end reference + home switch (Z signal)  After detecting the rising edge of the near-home switch, use the rising edge of the original home switch as the home position. Homing direction Limit (-) switch Near-home switch Limit (+) switch Home switch The starting point is between the near-home switch and the limit (+) switch (including on the limit (+) switch) Target speed Homing time DEC Homing creep speed The starting point is on the near-home switch Homing time DEC Limit stop DEC time Target speed Target speed Homing creep speed The starting point is between the near-home switch and the limit (-) switch Target speed Homing time DEC The starting point is on the limit (-) switch Target speed Homing DEC time	Case 1: The axis hits the block (near-home switch) at the target speed (HomeSpeed) → The axis runs reversely at 0 → The axis leaves the block at the target speed (HomeSpeed) → The axis runs reversely at 0 → The axis hits the block at Vim2 → The axis hits the home switch (motor Z signal) → The homing operation is completed
Mode 3	Near-home switch pulse edge detection front-end reference  Detect the rising edge of the near-home switch and use it as the home position. Homing direction Limit (-) switch Near-home switch Limit (+) switch The starting point is between the near-home switch and the limit (+) switch (including on the limit (+) switch) Homing DEC time Target speed Homing creep speed The starting point is on the near-home switch Homing creep speed Homing DEC time Limit stop DEC time Target speed Target speed Homing creep speed The starting point is between the near-home switch and the limit (-) switch Target speed Homing DEC time Homing creep speed The starting point is on the limit (-) switch Target speed Homing DEC time	Case 1: The axis hits the block (near-home switch) at the target speed (HomeSpeed) → The axis runs reversely at 0 → The axis leaves the block at the target speed (HomeSpeed) → The axis runs reversely at 0 → The axis hits the block at Vim2 → The homing operation is completed
Mode 4	Near-home switch pulse edge detection rear-end reference + home switch (Z signal)  After detecting the falling edge (backend) of the near-home switch, use the rising edge of the original home switch in the homing direction as the home position. Homing direction Limit (-) switch Home switch Near-home switch Limit (+) switch The starting point is between the near-home switch and the limit (+) switch (including on the limit (+) switch) Homing DEC time Target speed Homing creep speed Homing creep speed The starting point is on the near-home switch Limit stop DEC time Homing creep speed The starting point is between the near-home switch and the limit (-) switch Target speed Homing time DEC Homing creep speed The starting point is on the limit (-) switch Target speed Homing time DEC	Case 1: The axis hits the block (near-home switch) at the target speed (HomeSpeed) → The axis hits the block at Vim2 → The axis hits the home switch (motor Z signal) → The homing operation is completed

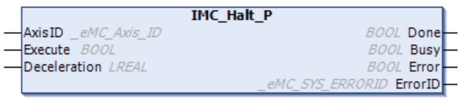
Mode	Timing Diagram	Brief Description
Mode 5	Limit switch pulse edge detection + home switch (Z signal)  After detecting the falling edge (backend) of the near-home switch, use the rising edge of the original home switch in the homing direction as the home position. Homing direction Limit (-) switch Home switch Limit (+) switch The starting point is other than on the limit (+) switch Homing creep speed Target speed The starting point is on the limit (+) switch Homing creep speed	Case 1: The axis hits the negative limit reversely at the target speed (HomeSpeed) → The axis runs reversely at 0 → The axis runs forward at Vim2 → The axis hits the home switch (motor Z signal) → The homing operation is completed
Mode 6	Limit switch pulse edge detection  After detecting the falling edge (backend) of the near-home switch, use the rising edge of the original home switch in the homing direction as the home position. Homing direction Limit (-) switch Limit (+) switch Target speed Homing DEC time Homing creep speed The starting point is other than on the limit (+) switch Homing creep speed The starting point is on the limit (+) switch Target speed	Case 1: The axis runs at the target speed (HomeSpeed) → The axis hits the negative limit → The axis runs reversely at 0 → The axis leaves the negative limit at the target speed (HomeSpeed) → The axis runs reversely at 0 → The axis hits the positive limit at Vim2 → The homing operation is completed
Mode 7	Home switch pulse edge detection  Move in the homing direction from the current value, stop after detecting the rising edge of the original home switch and use it as the home position. Homing direction Home switch Homing creep speed	For example: The axis returns to the home position in positive and negative directions-20 → The axis hits the home switch (motor Z signal) → The homing operation is completed
Mode 8	Near-home switch pulse edge detection rear-end reference + home switch (Z signal)  Home proximity signal External home input Positive limit input Negative limit input Start from the negative direction of the home proximity signal End normally Start from the home proximity signal ON End normally Start from the positive direction of the home proximity signal End normally	Case 1: The axis returns to the home position in positive and negative directions → home signal (deceleration block signal) → A signal that the axis leaves the home position at Vim2 is given (deceleration block signal) → A signal that the axis hits the external home position (motor Z signal) → The homing operation is completed

Mode	Timing Diagram	Brief Description
Mode 9	Near-home switch pulse edge detection back-end reference  Home proximity signal External home input Positive limit input Negative limit input Start from the negative direction of the home proximity signal End normally Start from the home proximity signal ON End normally Start from the positive direction of the home proximity signal End normally	Case 1: The axis returns to the home position in positive and negative directions → Home position signal (deceleration block signal) → A signal that the axis leaves the home position at Vim2 is given (deceleration block signal) → The homing operation is completed

9.2.7 IMC_Halt_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_Halt_P	Single axis stop	 IMC_Halt_P
		AxisID _eMC_Axis_ID
		Execute BOOL
		Deceleration LREAL
		BOOL Done
		BOOL Busy
		BOOL Error
		_eMC_SYS_ERRORID ErrorID
			`IMC_Halt_P(`
			`AxisID:= ,`
			`Execute:= ,`
			`Deceleration:= ,`
			`Done=> ,`
			`Busy=> ,`
			`Error=> ,`
			`ErrorID=> );`

Associated variables:

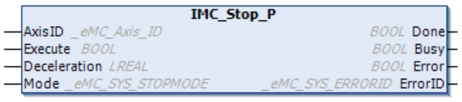
Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axis ID	_eMC_Axis_ID	0–3	255	Axle number
Execute	Function block enabling bit	BOOL	TRUE, FALSE	FALSE	Valid at the rising edge, and invalid at the falling edge
Deceleration	Deceleration	LREAL	Positive number	10	Deceleration (unit/s^2)
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Done	Function block completed state	BOOL	TRUE, FALSE	FALSE	TRUE: The function block has been executed, FALSE: The function block is not executed
Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE: The function block is being executed FALSE: The function block is not executed
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE: An error occurs, FALSE: No error occurs
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError_NULL	Error ID

Function description: This function block is triggered at the rising edge of Execute. There is no need to reset after the single axis stops since the single axis motion function block is triggered again and runs normally.

9.2.8 IMC_Stop_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_Stop_P	Single axis stop	 IMC_Stop_P
		AxisID _eMC_Axis_ID
		Execute BOOL
		Deceleration LREAL
		Mode _eMC_SYS_STOPMODE
		BOOL Done
		BOOL Busy
		BOOL Error
		_eMC_SYS_ERRORID ErrorID
			`IMC_Stop_P(`
			`AxisID:= ,`
			`Execute:= ,`
			`Deceleration:= ,`
			`Mode:= ,`
			`Done=> ,`
			`Busy=> ,`
			`Error=> ,`
			`ErrorID=> );`

Associated variables:

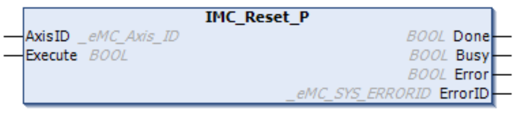
Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axis ID	_eMc_Axis_ID	0–3	255	Axle number
Execute	Function block enabling bit	BOOL	TRUE, FALSE	FALSE	Valid at the rising edge, and invalid at the falling edge
Deceleration	Deceleration	LREAL	Positive number	10	Deceleration (unit/s^2)
Mode	Stop mode	_eMc_Sys_StopMode	_mcDecStop, _mcImmediatelyStop	_mcDecStop	_mcDecStop: Deceleration to stop _mcImmediatelyStop: Stop immediately
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Done	Function block completed state	BOOL	TRUE, FALSE	FALSE	TRUE: The function block has been executed, FALSE: The function block is not executed
Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE: The function block is being executed FALSE: The function block is not executed
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE: An error occurs FALSE: No error occurs
ErrorID	Error ID	_eMc_Sys ErrorID	-	_mcError NULL	Error ID

Function description: This function block is triggered at the rising edge of Execute. If the mode is set to _mcDecStop, the axis state will be switched to Stopping after the axis stops. You need to set Execute to FALSE to release the axis Stopping state. If the mode is set to _mcImmediatelyStop, the axis state will be switched to Stopped after the axis stops. If the axis reports an error, you need to use the IMC_Reset function block to reset and release it.

9.2.9 IMC_Reset_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_Reset_P	Single axis reset		`IMC_Reset_P(`
			`AxisID:= ,`
			`Execute:= ,`
			`Done=> ,`
			`Busy=> ,`
			`Error=> ,`
			`ErrorID=> );`

Associated variables:

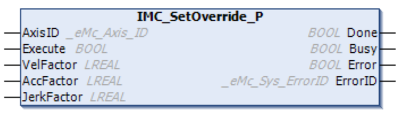
Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axis ID	_eMc_Axis_ID	0–3	255	Axle number
Execute	Function block enabling bit	BOOL	TRUE, FALSE	FALSE	Valid at the rising edge, and invalid at the falling edge
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Done	Function block completed state	BOOL	TRUE, FALSE	FALSE	TRUE: The function block has been executed FALSE: The function block is not executed
Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE: The function block is being executed FALSE: The function block is not executed
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE: An error occurs, FALSE: No error occurs
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError NULL	Error ID

Function description: The rising edge of Execute triggers the axis to reset. Generally, the single axis reset function block is required after the axis reaches the positive or negative limit or when an emergency stop occurs on the axis.

9.2.10 IMC_SetOverride_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_SetOverride _P	Single axis speed regulation		`IMC_SetOverride_P(`
			`AxisID:= ,`
			`Execute:= ,`
			`VelFactor:= ,`
			`AccFactor:= ,`
			`JerkFactor:= ,`
			`Done=> ,`
			`Busy=> ,`
			`Error=> ,`
			`ErrorID=> );`

Associated variables:

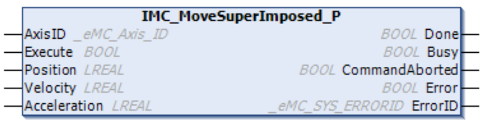
Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axis ID	_eMc_Axis_ID	0–3	255	Axle number
Execute	Function block enabling bit	BOOL	TRUE, FALSE	FALSE	Valid at the rising edge, and invalid at the falling edge
VelFactor	Instruction speed	LREAL	Positive number	10	Instruction speed after speed regulation (unit/s)
AccFactor	Acceleration value	LREAL	Positive number	10	Acceleration value after speed regulation (unit/s^2)
JerkFactor	Jerk value	LREAL	Positive or 0	0	Jerk value after speed regulation (unit/s^3)
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Done	Function block completed state	BOOL	TRUE, FALSE	FALSE	TRUE: Speed regulation has been completed, FALSE: Speed regulation is not completed
Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE: The function block is being executed FALSE: The function block is not executed
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE: An error occurs FALSE: No error occurs
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError _NULL	Error ID

Function description: This function block is triggered at the rising edge of Execute to adjust the motion parameters (speed, acceleration, acceleration change time, etc.) to the values set by users.

9.2.11 IMC_MoveSuperImposed_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_MoveSuperImposed_P	Single axis position superposition	 IMC_MoveSuperImposed_P
		AxisID _eMC_Axis_ID
		Execute BOOL
		Position LREAL
		Velocity LREAL
		Acceleration LREAL
		BOOL Done
		BOOL Busy
		BOOL CommandAborted
		BOOL Error
		_eMC_SYS_ERRORID ErrorID
			`IMC_MoveSuperImposed_P(`
			`AxisID:= ,`
			`Execute:= ,`
			`Position:= ,`
			`Velocity:= ,`
			`Acceleration:= ,`
			`Done=> ,`
			`Busy=> ,`
			`CommandAborted=> ,`
			`Error=> ,`
			`ErrorID=> );`

Associated variables:

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axis ID	_eMc_Axis_ID	0–3	255	Axle number

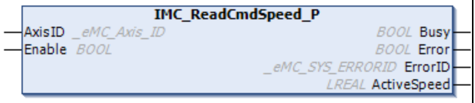
Execute	Function block enabling bit	BOOL	TRUE, FALSE	FALSE	Valid at the rising edge, and invalid at the falling edge
Position	Compensation distance	LREAL	≥-10,≤10	0.2	Superimposed compensation distance (unit)
Velocity	Running speed	LREAL	＞0,≤5	2	Speed (unit/s)
Acceleration	Acceleration	LREAL	Positive number	10	Acceleration (unit/s^2)
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**
Done	Function block completed state	BOOL	TRUE, FALSE	FALSE	TRUE: The function block has been executed FALSE: The function block is not executed
Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE: The function block is being executed FALSE: The function block is not executed
CommandAborted	Function block aborted	BOOL	TRUE, FALSE	FALSE	TRUE: Aborted, FALSE: Not aborted
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE: An error occurs, FALSE: No error occurs
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError_NULL	Error ID

Function description: The single axis position superposition function superimposes a track on the current axis track, and only supports the position superposition of single axis absolute motion, single axis relative motion, electronic cam, flying shear, and tracking shear function blocks.

9.2.12 IMC_ReadCmdSpeed_P

Instruction format:

Instruction	Name	Graphical Representation	ST Representation

IMC_ReadCmdSpeed_P	Single axis instruction speed reading	 IMC_ReadCmdSpeed_P
		AxisID _eMC_Axis_ID
		Enable BOOL
		BOOL Busy
		BOOL Error
		_eMC_SYS_ERRORID ErrorID
		LREAL ActiveSpeed
			`IMC_ReadCmdSpeed_P(`
			`AxisID:= ,`
			`Enable:= ,`
			`Busy=> ,`
			`Error=> ,`
			`ErrorID=> ,`
			`ActiveSpeed=> );`

Associated variables:

Input Variable	Name	Data Type	Valid Range	Initial Value	Description
AxisID	Axis ID	_eMc_Axis_ID	0–3	255	Axle number
Enable	Function block enabling bit	BOOL	TRUE, FALSE	FALSE	Valid if it is TRUE, and invalid if it is FALSE
Output Variable	**Name**	**Data Type**	**Valid Range**	**Initial Value**	**Description**

Busy	Function block executing	BOOL	TRUE, FALSE	FALSE	TRUE - The function block is being executed FALSE - The function block is not executed
Error	Function block error flag	BOOL	TRUE, FALSE	FALSE	TRUE: An error occurs FALSE: No error occurs
ErrorID	Error ID	_eMc_Sys_ErrorID	-	_mcError_ NULL	Error ID
ActiveSpeed	Speed	LREAL	Positive or 0	0	Actual speed of the axis (unit/s)

Function description: This instruction is used to read the speed of the specified single axis, and ActiveSpeed returns the actual speed of the axis.

Note: Single axis speed reading cannot be called in multi-axis running mode.

10 Fault Codes

10.1 SMC_ERROR Fault Codes (General Error Information for 402 Axis)

Error Number	Module	ENUM Variable	Description
0	All function blocks	SMC_NO_ERROR	No error
1	DriveInterface	SMC_DI_GENERAL_COMMUNICATION_ERROR	Communication error. For example, sercos ring has broken
2	DriveInterface	SMC_DI_AXIS_ERROR	Axis error
10	DriveInterface	SMC_DI_SWLIMITS_EXCEEDED	Position output within the allowed range (SWLimit)
11	DriveInterface	SMC_DI_HWLIMITS_EXCEEDED	Hard limit switch is active
13	DriveInterface	SMC_DI_HALT_OR_QUICKSTOP_NOT_SUPPORTED	Drive status Halt or Quickstop is not supported
14	DriveInterface	SMC_DI_VOLTAGE_DISABLED	The drive is not enabled
15	DriveInterface	SMC_DI_IRREGULAR_ACTPOSITION	Current position given from the drive seems to be irregular. Check the communication.
16	DriveInterface	SMC_DI_POSITIONLAGERROR	Position lag error. Difference between set and current position exceeds the given limit
20	All motion generating function blocks	SMC_REGULATOR_OR_START_NOT_SET	The controller is not enabled or the brake is applied
21	Axis in wrong controller mode	SMC_WRONG_CONTROLLER_MODE	The axis is under wrong controller mode
30	DriveInterface	SMC_FB_WASNT_CALLED_DURING_MOTION	The module created by motion control is not called before the motion is completed
31	All function blocks	SMC_AXIS_IS_NO_AXIS_REF	The given AXIS_REF variable is not of the type AXIS_REF
32	Axis in wrong controller mode	SMC_AXIS_REF_CHANGED_DURING_OPERATION	AXIS_REF variables have been changed while the modules being activated
33	DriveInterface	SMC_FB_ACTIVE_AXIS_DIABLED	The axis is not activated while moving (MC_Power.bRegulatorOn)
34	All motion generating function blocks	SMC_AXIS_NOT_READY_FOR_MOTION	Axis in its current state cannot execute a motion instruction
40	VirtualDrive	SMC_VD_MAX_VELOCITY_EXCEEDED	Maximum velocity (fMaxVelocity) exceeded

Error Number	Module	ENUM Variable	Description
41	VirtualDrive	SMC_VD_MAX_ACCELERATION_EXCEEDED	Maximum acceleration (fMaxAcceleration) exceeded
42	VirtualDrive	SMC_VD_MAX_DECELERATION_EXCEEDED	Maximum deceleration (fMaxDeceleration) exceeded
50	SMC_Homing	SMC_3SH_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
51	SMC_Homing	SMC_3SH_MODE_NEEDS_HWLIMIT	Mode requests use of limit switches for safety reasons
70	SMC_SetControllerMode	SMC_SCM_NOT_SUPPORTED	Mode not supported
71	SMC_SetControllerMode	SMC_SCM_AXIS_IN_WRONG_STATE	The controller mode cannot be changed in the current state
75	SMC_SetTorque	SMC_ST_WRONG_CONTROLLER_MODE	The axis is under the wrong controller mode
80	SMC_ResetAxisGroup	SMC_RAG_ERROR_DURING_STARTUP	Error occurs when the axis group is activated
90	SMC_ChangeGearingRatio	SMC_CGR_ZERO_VALUES	Invalid values
91	SMC_ChangeGearingRatio	SMC_CGR_DRIVE_POWERED	The gear ratio parameters of the drive cannot be modified when it is under control
92	SMC_ChangeGearingRatio	SMC_CGR_INVALID_POSPERIOD	Invalid position period (<=0)
110	MC_Power	SMC_P_FTASKCYCLE_EMPTY	Axis contains no information in the scan cycle (fTaskCycle=0)
120	MC_Reset	SMC_R_NO_ERROR_TO_RESET	Axis reset without error
121	MC_Reset	SMC_R_DRIVE_DOESNT_ANSWER	Axis does not perform error-reset
122	MC_Reset	SMC_R_ERROR_NOT_RESETTABLE	Error could not be reset
123	MC_Reset	SMC_R_DRIVE_DOESNT_ANSWER_IN_TIME	Communication with the axis did not work
130	MC_ReadParameter, MC_ReadBoolParameter	SMC_RP_PARAM_UNKNOWN	Parameter number unknown
131	MC_ReadParameter, MC_ReadBoolParameter	SMC_RP_REQUESTING_ERROR	Error during parameter transmission to the drive. See error number in function block instance ReadDriveParameter (SM_DriveBasic.lib)
140	MC_WriteParameter, MC_WriteBoolParameter	SMC_WP_PARAM_INVALID	Parameter number unknown or writing not allowed
141	MC_WriteParameter, MC_WriteBoolParameter	SMC_WP_SENDING_ERROR	See error number in function block instance WriteDriveParameter (Drive_Basic.lib)

Error Number	Module	ENUM Variable	Description
170	MC_Home	SMC_H_AXIS_WASNT_STANDSTILL	Axis has not been in standstill state
171	MC_Home	SMC_H_AXIS_DIDNT_START_HOMING	Error at start of homing action
172	MC_Home	SMC_H_AXIS_DIDNT_ANSWER	Communication error
173	MC_Home	SMC_H_ERROR_WHEN_STOPPING	Error at stop after homing. Check whether deceleration is set
180	MC_Stop	SMC_MS_UNKNOWN_STOPPING_ERROR	Unknown error at stop
181	MC_Stop	SMC_MS_INVALID_ACCDEC_VALUES	Invalid velocity or acceleration values
182	MC_Stop	SMC_MS_DIRECTION_NOT_APPLICABLE	Direction=shortest not applicable
183	MC_Stop	SMC_MS_AXIS_IN_ERRORSTOP	Drive is in errorstop status. Stop cannot be executed
184	MC_Stop	SMC_BLOCKING_MC_STOP_WASNT_CALLED	Instance of MC_Stop blocking the axis by Execute=TRUE has not been called yet. Please call MC_Stop (Execute=FALSE).
201	MC_MoveAbsolute	SMC_MA_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
202	MC_MoveAbsolute	SMC_MA_INVALID_DIRECTION	Direction error
226	MC_MoveRelative	SMC_MR_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
227	MC_MoveRelative	SMC_MR_INVALID_DIRECTION	Direction error
251	MC_MoveAdditive	SMC_MAD_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
252	MC_MoveAdditive	SMC_MAD_INVALID_DIRECTION	Direction error
276	MC_MoveSuperImposed	SMC_MSI_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
277	MC_MoveSuperImposed	SMC_MSI_INVALID_DIRECTION	Direction error
301	MC_MoveVelocity	SMC_MV_INVALID_ACCDEC_VALUES	Invalid velocity or acceleration values
302	MC_MoveVelocity	SMC_MV_DIRECTION_NOT_APPLICABLE	Direction=shortest/fastest not applicable
325	MC_PositionProfile	SMC_PP_ARRAYSIZE	Erroneous array size
326	MC_PositionProfile	SMC_PP_STEP0MS	Step time = t#0s
350	MC_VelocityProfile	SMC_VP_ARRAYSIZE	Erroneous array size
351	MC_VelocityProfile	SMC_VP_STEP0MS	Step time = t#0s
375	MC_AccelerationProfile	SMC_AP_ARRAYSIZE	Erroneous array size
376	MC_AccelerationProfile	SMC_AP_STEP0MS	Step time = t#0s
400	MC_TouchProbe	SMC_TP_TRIGGEROCCUPIED	Trigger already active

Error Number	Module	ENUM Variable	Description
401	MC_TouchProbe	SMC_TP_COULDNT_SET_WINDOW	Drive interface does not support the window function
402	MC_TouchProbe	SMC_TP_COMM_ERROR	Communication error
410	MC_AbortTrigger	SMC_AT_TRIGGERNOTOCCUPIED	Trigger already de-allocated
426	SMC_MoveContinuousRelative	SMC_MCR_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
427	SMC_MoveContinuousRelative	SMC_MCR_INVALID_DIRECTION	Direction error
451	SMC_MoveContinuousAbsolute	SMC_MCA_INVALID_VELACC_VALUES	Invalid velocity or acceleration values
452	SMC_MoveContinuousAbsolute	SMC_MCA_INVALID_DIRECTION	Direction error
453	SMC_MoveContinuousAbsolute	SMC_MCA_DIRECTION_NOT_APPLICABLE	Direction=fastest not applicable
600	SMC_CamRegister	SMC_CR_NO_TAPPETS_IN_CAM	Cam does not contain any tappets
601	SMC_CamRegister	SMC_CR_TOO_MANY_TAPPETS	Tappet group ID exceeds MAX_NUM_TAPPETS
602	SMC_CamRegister	SMC_CR_MORE_THAN_32_ACCESSES	More than 32 accesses in one CAM_REF
625	MC_CamIN	SMC_CI_NO_CAM_SELECTED	No cam selected
626	MC_CamIN	SMC_CI_MASTER_OUT_OF_SCALE	Master axis out of valid range
627	MC_CamIN	SMC_CI_RAMPIN_NEEDS_VELACC_VALUES	Velocity and acceleration values must be specified for ramp_in function
628	MC_CamIN	SMC_CI_SCALING_INCORRECT	Scaling variables fEditor/TableMasterMin/Max are not correct
640	SMC_CAMBounds, SMC_CamBounds_Pos	SMC_CB_NOT_IMPLEMENTED	Function block for the given cam format is not implemented
675	MC_GearIn	SMC_GI_RATIO_DENOM	RatioDenominator=0
676	MC_GearIn	SMC_GI_INVALID_ACC	Acceleration invalid
677	MC_GearIn	SMC_GI_INVALID_DEC	Deceleration invalid
725	MC_Phase	SMC_PH_INVALID_VELACCDEC	Velocity and acceleration/deceleration values invalid
726	MC_Phase	SMC_PH_ROTARYAXIS_PERIOD0	Rotation axis with fPositionPeriod = 0
750	All modules using MC_CAM_REF as input	SMC_NO_CAM_REF_TYPE	Type of given cam is not MC_CAM_REF.
751	MC_CamTableSelect	SMC_CAM_TABLE_DOES_NOT_COVER_MASTER_SCALE	Master axis area (xStart and xEnd) from CamTable is not covered by curve data

Error Number	Module	ENUM Variable	Description
775	MC_GearInPos	SMC_GIP_MASTER_DIRECTION_CHANGE	During coupling of slave axis, master axis has changed direction of rotation
800	SMC_BacklashCompensation	SMC_BC_BL_TOO_BIG	Gear backlash fBacklash too large (> position periode/2)
1000	CNC function blocks which are supervising the licensing	SMC_NO_LICENSE	Target is not licensed for CNC
1001	SMC_Interpolator	SMC_INT_VEL_ZERO	Path cannot be processed because set velocity = 0
1002	SMC_Interpolator	SMC_INT_NO_STOP_AT_END	Last object of path has Vel_End>0
1003	SMC_Interpolator	SMC_INT_DATA_UNDERRUN	Warning: GEOINFO-List processed in DataIn but end of list not reached. Reason: EndOfList of the queue in DataIn not be set or SMC_Interpolator faster than path generating function blocks
1004	SMC_Interpolator	SMC_INT_VEL_NONZERO_AT_STOP	Velocity at Stop > 0
1005	SMC_Interpolator	SMC_INT_TOO_MANY_RECURSIONS	Too many SMC_Interpolator recursions. SoftMotion error.
1006	SMC_Interpolator	SMC_INT_NO_CHECKVELOCITIES	Input-OutQueueDataIn is not the last processed function block of SMC_CHeckVelocities
1007	SMC_Interpolator	SMC_INT_PATH_EXCEEDED	Internal or numeric error
1008	SMC_Interpolator	SMC_INT_VEL_ACC_DEC_ZERO	Velocity and acceleration / deceleration is null or too low
1009	SMC_Interpolator	SMC_INT_DWIPOTIME_ZERO	FB called with dwIpoTime = 0
1050	SMC_Interpolator2Dir	SMC_INT2DIR_BUFFER_TOO_SMALL	Data buffer too small
1051	SMC_Interpolator2Dir	SMC_INT2DIR_PATH_FITS_NOT_IN_QUEUE	Path does not go completely in queue
1100	SMC_CheckVelocities	SMC_CV_ACC_DEC_VEL_NONPOSITIVE	Velocity and acceleration/deceleration values non-positive
1120	SMC_Controlaxisbypos	SMC_CA_INVALID_ACCDEC_VALUES	Values of fGapVelocity / fGapAcceleration / fGapDeceleration non-positive
1200	SMC_NCDecoder	SMC_DEC_ACC_TOO_LITTLE	Acceleration value not allowed
1201	SMC_NCDecoder	SMC_DEC_RET_TOO_LITTLE	Deceleration value not allowed
1202	SMC_NCDecoder	SMC_DEC_OUTQUEUE_RAN_EMPTY	Data underrun. Queue has been read and is empty.

Error Number	Module	ENUM Variable	Description
1203	SMC_NCDecoder	SMC_DEC_JUMP_TO_UNKNOWN_LINE	Jump to line cannot be executed because line number is unknown
1204	SMC_NCDecoder	SMC_DEC_INVALID_SYNTAX	Syntax invalid
1205	SMC_NCDecoder	SMC_DEC_3DMODE_OBJECT_NOT_SUPPORTED	Objects are not supported in 3D mode
1300	SMC_GCodeViewer	SMC_GCV_BUFFER_TOO_SMALL	Buffer too small
1301	SMC_GCodeViewer	SMC_GCV_BUFFER_WRONG_TYPE	Buffer elements have wrong types
1302	SMC_GCodeViewer	SMC_GCV_UNKNOWN_IPO_LINE	Current line of the Interpolator could not be found
1500	All function blocks using SMC_CNC_REF	SMC_NO_CNC_REF_TYPE	Given CNC program is not of the type SMC_CNC_REF
1501	All function blocks using SMC_OUTQUEUE	SMC_NO_OUTQUEUE_TYPE	Given OutQueue is not of the type SMC_OUTQUEUE
1600	CNC function blocks	SMC_3D_MODE_NOT_SUPPORTED	Function block only works with 2D paths
2000	SMC_ReadNCFile	SMC_RNCF_FILE_DOESNT_EXIST	File does not exist
2001	SMC_ReadNCFile	SMC_RNCF_NO_BUFFER	No buffer allocated
2002	SMC_ReadNCFile	SMC_RNCF_BUFFER_TOO_SMALL	Buffer too small
2003	SMC_ReadNCFile	SMC_RNCF_DATA_UNDERRUN	Data underrun. Buffer has been read and is empty
2004	SMC_ReadNCFile	SMC_RNCF_VAR_COULDNT_BE_REPLACED	Placeholder variable could not be replaced
2005	SMC_ReadNCFile	SMC_RNCF_NOT_VARLIST	Input pvl does not point to a SMC_VARLIST object
2050	SMC_ReadNCQueue	SMC_RNCQ_FILE_DOESNT_EXIST	File could not be opened
2051	SMC_ReadNCQueue	SMC_RNCQ_NO_BUFFER	No buffer defined
2052	SMC_ReadNCQueue	SMC_RNCQ_BUFFER_TOO_SMALL	Buffer too small
2053	SMC_ReadNCQueue	SMC_RNCQ_UNEXPECTED_EOF	Unexpected end of file
2100	SMC_AxisDiagnosticLog	SMC_ADL_FILE_CANNOT_BE_OPENED	File could not be opened
2101	SMC_AxisDiagnosticLog	SMC_ADL_BUFFER_OVERRUN	Buffer overrun. WriteToFile must be called more frequently
2200	SMC_ReadCAM	SMC_RCAM_FILE_DOESNT_EXIST	File could not be opened
2201	SMC_ReadCAM	SMC_RCAM_TOO_MUCH_DATA	Saved cam too big
2202	SMC_ReadCAM	SMC_RCAM_WRONG_COMPILE_TYPE	Wrong compilation mode

Error Number	Module	ENUM Variable	Description
2203	SMC_ReadCAM	SMC_RCAM_WRONG_VERSION	Wrong file version
2204	SMC_ReadCAM	SMC_RCAM_UNEXPECTED_EOF	Unexpected end of file
3001	SMC_WriteDriveParamsToFile	SMC_WDPF_CHANNEL_OCCUPIED	File channel occupied
3002	SMC_WriteDriveParamsToFile	SMC_WDPF_CANNOT_CREATE_FILE	File could not be created
3003	SMC_WriteDriveParamsToFile	SMC_WDPF_ERROR_WHEN_READING_PARAMS	Error at reading the parameters
3004	SMC_WriteDriveParamsToFile	SMC_WDPF_TIMEOUT_PREPARING_LIST	Timeout during preparing the parameter list
5000	SMC_Encoder	SMC_ENC_DENOM_ZERO	Nominator of the conversion factor dwRatioTechUnitsDenom of the Encoder reference is 0
5001	SMC_Encoder	SMC_ENC_AXISUSEDBYOTHERFB	Other module trying to process motion on the Encoder axis
5002	DriveInterface	SMC_ENC_FILTER_DEPTH_INVALID	Filter depth invalid

10.2 PLC Error Code Table (for TM and TP series PLCs)

Error Type		Error Location	Major Error Code	Sub-error Code	Error Description
CPU	System-related	Hardware error	0001	0001	Button cell not installed or battery voltage too low
				0002	Device supply voltage too low (less than 19V)
	System component-related	Clock system component error	0008	0001	Error in setting time
				0002	Error in writing RTC clock
				0003	Error in reading RTC clock
		IP system component error	0009	0001	IP segments of IP1 and IP2 repeated
				XXX	Reserved
				0011	Read: IP1 module - Error in opening files
				0012	Read: IP1 module - Unable to get IP information
				0013	Write: IP1 module - IP address configuration error
				0014	Write: IP1 module - Mask configuration error
				0015	Write: IP1 module -Gateway configuration error
				0016	Write: IP1 module - Repeated segments with USB
				0017	Write: IP1 module - IP and gateway in different segments
				XXX	Reserved

Error Type		Error Location	Major Error Code	Sub-error Code	Error Description
				0021	Read: IP2 module - Error in opening files
				0022	Read: IP2 module - Unable to get IP information
				0023	Write: IP2 module - IP address error
				0024	Write: IP2 module - Mask error
				0025	Write: IP2 module - Gateway error
				0026	Write: IP2 module - Repeated segments with USB
				0027	Write: IP2 module - IP and gateway in different segments
Backplane bus	Backplane bus-related	CPU IO error	0030	0001	Module configuration error
				0002	Module parameter configuration error
		Digital quantity error	0031	0001	DI - Module configuration error
				0002	DI - Module parameter configuration error
				XXX	Reserved
				2001	DO - Module configuration error
				2002	DO - Module parameter configuration error
				2003	DO - Module output port power supply failure
				2004	DO - Module output error
				XXX	Reserved
				XXX	Reserved
		Analog quantity error	0032	0001	Module configuration error
				XXX	Reserved
				0012	AD - Channel 0 parameter configuration error
				0015	AD- Channel 0 signal source open circuit
				0016	AD - Channel 0 sampling signal over-limit
				0017	AD - Channel 0 sampling signal above-upper-limit
				0018	AD - Channel 0 sampling signal below-lower-limit
				XXX	Reserved
				0022	AD - Channel 1 parameter configuration error
				0025	AD- Channel 1 signal source open circuit
				0026	AD - Channel 1 sampling signal over-limit
				0027	AD - Channel 1 sampling signal above-upper-limit
				0028	AD - Channel 1 sampling signal below-lower-limit
				XXX	Reserved
				0032	AD - Channel 2 parameter configuration error
				0035	AD- Channel 2 signal source open circuit

Error Type		Error Location	Major Error Code	Sub-error Code	Error Description
				0036	AD - Channel 2 sampling signal over-limit
				0037	AD - Channel 2 sampling signal above-upper-limit
				0038	AD - Channel 2 sampling signal below-lower-limit
				XXX	Reserved
				0042	AD - Channel 3 parameter configuration error
				0045	AD- Channel 3 signal source open circuit
				0046	AD - Channel 3 sampling signal over-limit
				0047	AD - Channel 3 sampling signal above-upper-limit
				0048	AD - Channel 3 sampling signal below-lower-limit
				XXX	Reserved
				0003	Module output port power supply failure
				XXX	Reserved
				2012	Channel 0 parameter configuration error
				2014	Channel 0 output error
				XXX	Reserved
				2022	Channel 1 parameter configuration error
				2024	Channel 1 output error
				XXX	Reserved
				2032	Channel 2 parameter configuration error
				2034	Channel 2 output error
				XXX	Reserved
				2042	Channel 3 parameter configuration error
				2044	Channel 3 output error
				XXX	Reserved
		Temperature measuring module error	0033	0001	Module configuration error
				XXX	Reserved
				0012	Channel 0 parameter configuration error
				0015	Channel 0 signal source open-circuit
				0017	Channel 0 sampling signal above-upper-limit
				0018	Channel 0 sampling signal below-lower-limit
				XXX	Reserved
				0022	Channel 1 parameter configuration error
				0025	Channel 1 signal source open-circuit

Error Type		Error Location	Major Error Code	Sub-error Code	Error Description
				0027	Channel 1 sampling signal above-upper-limit
				0028	Channel 1 sampling signal below-lower-limit
				XXX	Reserved
				0032	Channel 2 parameter configuration error
				0035	Channel 2 signal source open-circuit
				0037	Channel 2 sampling signal above-upper-limit
				0038	Channel 2 sampling signal below-lower-limit
				XXX	Reserved
				0042	Channel 3 parameter configuration error
				0045	Channel 3 signal source open-circuit
				0047	Channel 3 sampling signal above-upper-limit
				0048	Channel 3 sampling signal below-lower-limit
Fieldbus	Modbus-related	Modbus_RTU Master1	0040	0001	Illegal function code
				0002	Illegal address
				0003	Wrong number of data
				0004	Slave device failure
				0005	Communication timeout. An error occurs since the communication time exceeds the maximum communication time set by the user
				XXX	Reserved
				0008	Received data frame non-conforming to the Modbus protocol
				0009	CRC/LRC check error
				XXX	Reserved
				000B	The length of received data does not conform to the protocol or the number exceeds the maximum limit specified by the function code
				000C	The received slave address does not match the requested slave address
				000D	The received function code does not match the requested function code
				000E	Instruction execution failed
		Modbus_RTU Master2	0041	0001	Illegal function code
				0002	Illegal address
				0003	Wrong number of data
				0004	Slave device failure
				0005	Communication timeout. An error occurs since the communication time exceeds the maximum communication time set by the user

Error Type		Error Location	Major Error Code	Sub-error Code	Error Description
				XXX	Reserved
				0008	Received data frame non-conforming to the Modbus protocol
				0009	CRC/LRC check error
				XXX	Reserved
				000B	The length of received data does not conform to the protocol or the number exceeds the maximum limit specified by the function code
				000C	The received slave address does not match the requested slave address
				000D	The received function code does not match the requested function code
				000E	Instruction execution failed
		Modbus_RTU Slave1	0042	0001	Illegal function code
				0002	Illegal address
				0003	Wrong number of data
				0004	Slave device failure
				0005	Communication timeout. An error occurs since the communication time exceeds the maximum communication time set by the user
				XXX	Reserved
				0008	Received data frame non-conforming to the Modbus protocol
				0009	CRC/LRC check error
				XXX	Reserved
				000B	The length of received data does not conform to the protocol or the number exceeds the maximum limit specified by the function code
				000C	The received slave address does not match the requested slave address
				000D	The received function code does not match the requested function code
				000E	Instruction execution failed
		Modbus_RTU Slave2	0043	0001	Illegal function code
				0002	Illegal address
				0003	Wrong number of data
				0004	Slave device failure
				0005	Communication timeout. An error occurs since the communication time exceeds the maximum communication time set by the user
				XXX	Reserved
				0008	Received data frame non-conforming to the Modbus protocol
				0009	CRC/LRC check error
				XXX	Reserved

Error Type		Error Location	Major Error Code	Sub-error Code	Error Description
				000B	The length of received data does not conform to the protocol or the number exceeds the maximum limit specified by the function code
				000C	The received slave address does not match the requested slave address
				000D	The received function code does not match the requested function code
				000E	Instruction execution failed
		ModbusTCP Master1	00A0	0001	Illegal function code
				0002	Illegal address
				0003	Wrong number of data
				0004	Slave device failure
				0005	Communication timeout. An error occurs since the communication time exceeds the maximum communication time set by the user
				XXX	Reserved
				0008	Received data frame non-conforming to the Modbus protocol
				0009	CRC/LRC check error
				XXX	Reserved
				000B	The length of received data does not conform to the protocol or the number exceeds the maximum limit specified by the function code
				000C	The received slave address does not match the requested slave address
				000D	The received function code does not match the requested function code
	ModbusTCP-related			000E	Instruction execution failed
		ModbusTCP Master2	00A1	0001	Illegal function code
				0002	Illegal address
				0003	Wrong number of data
				0004	Slave device failure
				0005	Communication timeout. An error occurs since the communication time exceeds the maximum communication time set by the user
				XXX	Reserved
				0008	Received data frame non-conforming to the Modbus protocol
				0009	CRC/LRC check error
				XXX	Reserved
				000B	The length of received data does not conform to the protocol or the number exceeds the maximum limit specified by the function code
				000C	The received slave address does not match the requested slave address

Error Type		Error Location	Major Error Code	Sub-error Code	Error Description
				000D	The received function code does not match the requested function code
				000E	Instruction execution failed
		ModbusTCP Slave1	00A2	0001	Illegal function code
				0002	Illegal address
				0003	Wrong number of data
				0004	Slave device failure
				0005	Communication timeout. An error occurs since the communication time exceeds the maximum communication time set by the user
				XXX	Reserved
				0008	Received data frame non-conforming to the Modbus protocol
				0009	CRC/LRC check error
				XXX	Reserved
				000B	The length of received data does not conform to the protocol or the number exceeds the maximum limit specified by the function code
				000C	The received slave address does not match the requested slave address
				000D	The received function code does not match the requested function code
				000E	Instruction execution failed
		ModbusTCP Slave2	00A3	0001	Illegal function code
				0002	Illegal address
				0003	Wrong number of data
				0004	Slave device failure
				0005	Communication timeout. An error occurs since the communication time exceeds the maximum communication time set by the user
				XXX	Reserved
				0008	Received data frame non-conforming to the Modbus protocol
				0009	CRC/LRC check error
				XXX	Reserved
				000B	The length of received data does not conform to the protocol or the number exceeds the maximum limit specified by the function code
				000C	The received slave address does not match the requested slave address
				000D	The received function code does not match the requested function code
				000E	Instruction execution failed

Your Trusted Industry Automation Solution Provider

Shenzhen INVT Electric Co., Ltd.
Address: INVT Guangming Technology Building, Songbai Road, Matian,
Guangming District, Shenzhen, China
INVT Power Electronics (Suzhou) Co., Ltd.
Address: No. 1 Kunlun Mountain Road, Science & Technology Town,
Gaoxin District, Suzhou, Jiangsu, China
Website: www.invt.com

INVT mobile website

INVT e-manual

66001-01388

Copyright© INVT. Manual information may be subject to change without prior notice.

202409 (V1.0)