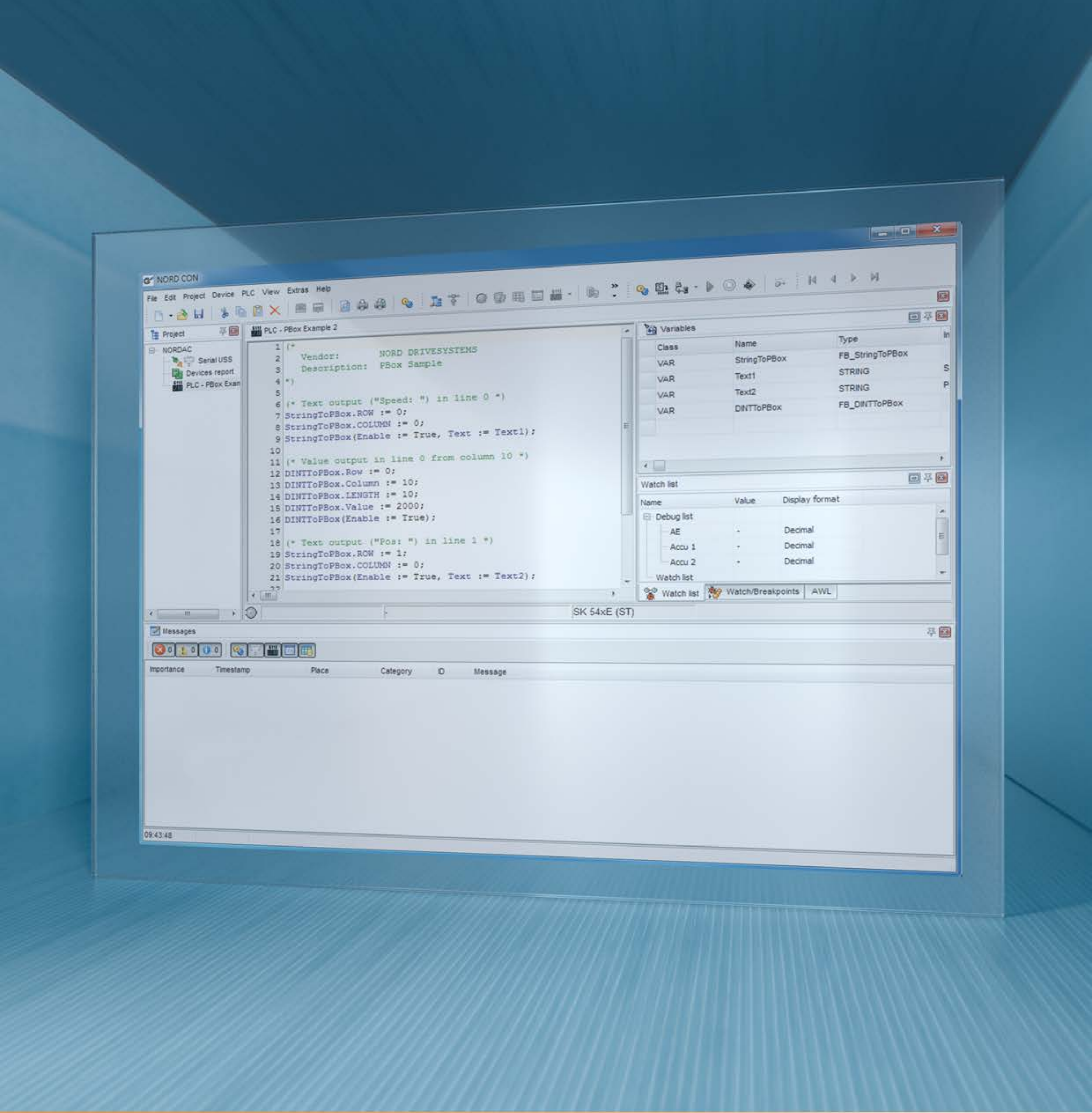
INTELLIGENT DRIVESYSTEMS, WORLDWIDE SERVICES



BU 0550 – en

PLC functionality

Supplementary manual for NORDAC - devices

NORD DRIVESYSTEMS

Table of Contents

1 Introduction

1.1 General

1.1.1 Documentation

Name:	BU 0550	
Part number:	6075502	
Series:	**PLC functionality for frequency inverter and motor starter series**	
	NORDAC *PRO*	(SK 500P … SK 550P)
		(SK 520E … SK 545E)
	NORDAC *Flex*	(SK 200E … SK 235E)
	NORDAC *Base*	(SK 180E / SK 190E)
	NORDAC *Link*	(SK 250E-FDS … SK 280E-FDS)
	NORDAC *Link*	(SK 155E-FDS / SK 175E-FDS)
	NORDAC *ON/ON+*	(SK 300P)

1.1.2 Document history

Edition Order number	Series	Version Software	Remarks
BU 0550, September 2011 **6075502**/ 3911	SK 540E … SK 545E	V 2.0 R0	First edition
More revisions: October 2011, February 2013, February 2017, May 2019 An overview of the changes in the above-mentioned editions can be found in the respective document.			
BU 0550, January 2021 **6075502**/ 0321	SK 500P … SK 550P SK 540E … SK 545E SK 520E … SK 535E SK 200E … SK 235E SK 180E / SK 190E SK 250E-FDS … SK 280E-FDS SK 155E-FDS / SK 175E-FDS SK 300P	V 1.2 R2 V 2.4 R2 V 3.2 R2 V 2.2 R1 V 1.3 R0 V 1.3 R1 V 1.2 R1 V 1.0 R1	• Implementation of device types NORDAC *ON/ON+* SK 300P • Modifications and corrections

1.1.3 Copyright notice

As an integral component of the device or the function described here, this document must be provided to all users in a suitable form.

Any editing or amendment or other utilisation of the document is prohibited.

1.1.4 Publisher

Getriebebau NORD GmbH & Co. KG

Getriebebau-Nord-Straße 1

22941 Bargteheide, Germany

http://www.nord.com/

Tel.: +49 (0) 45 32 / 289-0

Fax: +49 (0) 45 32 / 289-2253

1.1.5 About this manual

This manual is intended to help you with the commissioning of the PLC functionality of a frequency inverter or motor starter from Getriebebau NORD GmbH & Co. KG (NORD). It is intended for all qualified electricians who plan, install and set up the PLC programs for the device (🕮 Section 2.1 "Selection and qualification of personnel"). The information in this manual assumes that the qualified electricians who are entrusted with this work are familiar with the handling of electronic drive technology, in particular with NORD devices.

This manual only contains information and descriptions of the PLC functionality and the additional information which is relevant for the PLC functionality of devices manufactured by Getriebebau NORD GmbH & Co. KG.

1.2 Other applicable documents

This document is only valid in combination with the operating instructions for the frequency inverter which is used. Safe commissioning of the drive application depends on the availability of the information contained in this document.. A list of the documents can be found in 🕮 Section 5.2 "Documents and software".

The necessary documents can be found under www.nord.com.

1.3 Presentation conventions

1.3.1 Warning information

Warning information for the safety of the user and the bus interfaces are indicated as follows:

DANGER

This warning information warns against personal risks, which may cause severe injury or death.

WARNING

This warning information warns against personal risks, which may cause severe injury or death.

CAUTION

This warning information warns against personal risks, which may cause slight or moderate injuries.

NOTICE

This warning warns against damage to material.

1.3.2 Other information

Information

This information shows hints and important information.

1.4 Intended use

The PLC functionality from Getriebebau NORD GmbH & Co. KG is a software assisted, functional extension for NORD frequency inverters and motor starters. It is fully integrated into the particular device and cannot be used independently of this. Therefore the specific safety information for the particular device applies in full. This can be obtained from the relevant manual (🕮 Section 5.2 "Documents and software").

The PLC functionality essentially serves for the solution of complex drive tasks with one or more electronic drive technology devices, as well as for the simplification of control and monitoring functions close to the device by means of an appropriately equipped device.

2 Safety

2.1 Selection and qualification of personnel

The PLC functionality may only be installed and commissioned by qualified electricians. They must have the necessary knowledge of PLC functionality and the electronic drive technology which is used, as well as the configuration aids e.g. NORD CON software) and with the peripherals associated with the drive output (including the control unit).

In addition, the qualified electricians must also be familiar with the installation, commissioning and operation of sensors and electronic drive technology, as well as all of the accident prevention regulations, guidelines and laws which apply at the place of use.

2.1.1 Qualified personnel

Qualified personnel includes persons who due to their specialist training and experience have sufficient knowledge in a specialised area and are familiar with the relevant occupational safety and accident prevention regulations as well as the generally recognised technical rules.

These persons must be authorised to carry out the necessary work by the operator of the system.

2.1.2 Qualified electrician

An electrician is a person who, because of their technical training and experience, has sufficient knowledge with regard to

- Switching on, switching off, isolating, earthing and marking power circuits and devices,
- Proper maintenance and use of protective devices in accordance with defined safety standards.
- Emergency treatment of injured persons.

2.2 Safety information

Only use the technology function **PLC functionality** and the frequency inverter from Getriebebau NORD GmbH & Co. KG for their intended purposes as stated in 🕮 Section 1.4 "Intended use".

Observe the instructions in this manual in order to ensure the safe use of the technology function.

Only commission the frequency inverter in a technically unmodified form and not without the necessary covers. Take care that all connections and cables are in good condition.

Work on and with the frequency inverter must only be carried out by qualified personnel, 🕮 Section 2.1 "Selection and qualification of personnel".

3 PLC

3.1 General

The NORD frequency inverter series SK 180E/SK 190E, SK 2xxE, SK 2xxE-FDS, SK 300P, SK 520E – SK 545E and SK 5xxP as well as the motor starter series SK 155E-FDS/SK 175E-FDS contains logic processing which is similar to the current IEC61131-3 standard for memory programmable control units (SPS / PLC). The reaction speed or computing power of this PLC is suitable to undertake smaller tasks in the area of the inverter. Inverter inputs or information from a connected field bus can be monitored, evaluated and further processed into appropriate setpoint values for the frequency inverter. In combination with other NORD devices, visualisation of system statuses or the input of special customer parameters is also possible. Therefore, within a limited range, there is a potential for savings via the elimination of a previous external PC solution. AWL is supported as the programming language. AWL is a machine-orientated, text-based programming language whose scope and application is specified in IEC61131-3.

Information

Programming and download into the devices are possible exclusively via the NORD software NORDCON.

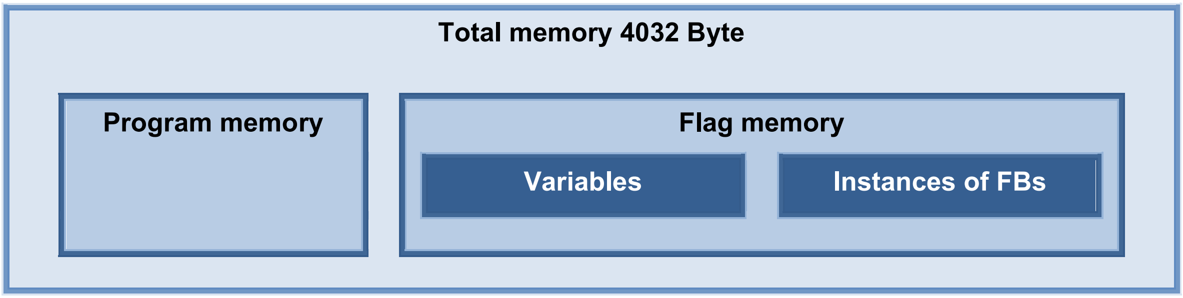
3.1.1 Specification of the PLC

Function	Specification		
Standard	Orientated to IEC61131-3		
Language	Instruction list (IL), structured text (ST)		
Task	A cyclic task, program call-up every 5 ms		
Computer performance	Approximately 200 IL commands per 1 ms		
Program memory	SK 5xxP, SK 520E … SK 545E, SK 2xxE, SK 2x0E-FDS, On, On+	SK 190E / SK 180E	SK 155E-FDS / SK 175E-FDS
	8128 bytes for flags, functions and the PLC program	2032 bytes for flags, functions and the PLC program	2028 bytes for flags, functions and the PLC program
Max. possible number of commands	Approximately 2580 commands	Approximately 660 commands	Approximately 660 commands
	Note: This is an average value. Heavy use of flags, process data and functions considerably reduces the possible number of lines; see Resources section.		
Freely accessible CAN mailboxes	20 (except for On/On+)		
Supported devices	SK 5xxP SK 54xE SK 53xE / SK 52xE from V3.0 On/On+ SK 2xxE from V2.0 SK 2x0E-FDS SK 180E / SK 190E SK 155E-FDS / SK 175E- FDS		

3.1.2 PLC structure

3.1.2.1 Memory

The PLC memory is divided into the program memory and the flag memory. In addition to the variables, instances of function blocks are saved in the area of the flag memory. FB instance is a memory area in which all internal input and output variables of function command are saved. Each function command declaration requires a separate instance. The boundary between the program memory and the flag memory is determined dynamically, depending on the size of the flag area.



In the flag memory, two different classes of variables are stored in the variable section:

[VAR]

Memory variable for saving auxiliary information and statuses. Variables of this type are initialised every time the PLC starts. The memory content is retained during the cyclic sequence of the PLC.

[VAR_ACCESS]

These are used to read and describe process data (inputs, outputs, setpoints, etc.) of the frequency inverter. These values are regenerated with every PLC cycle.

3.1.2.2 Image of the process

Several physical dimensions such as torque, speed, position, inputs, outputs etc. are available to the device. These dimensions are divided into actual and setpoint values. They can be loaded into the process image of the PLC and influenced by it. The required processes must be defined in the list of variables under the class VAR_ACCESS. With each PLC cycle, all of the process data for the inverter which is defined in the list of variables is newly read in. At the end of each PLC cycle the writable process data are transferred back to the inverter, see following illustration.

Because of this sequence it is important to program a cyclic program sequence. Programming loops in order to wait for a certain event (e.g. change of level at an input) does not produce the required result. This behaviour is different in the case of function blocks which access process values. Here, the process value is read on call-up of the function block and the process values are written immediately when the block is terminated.

Information

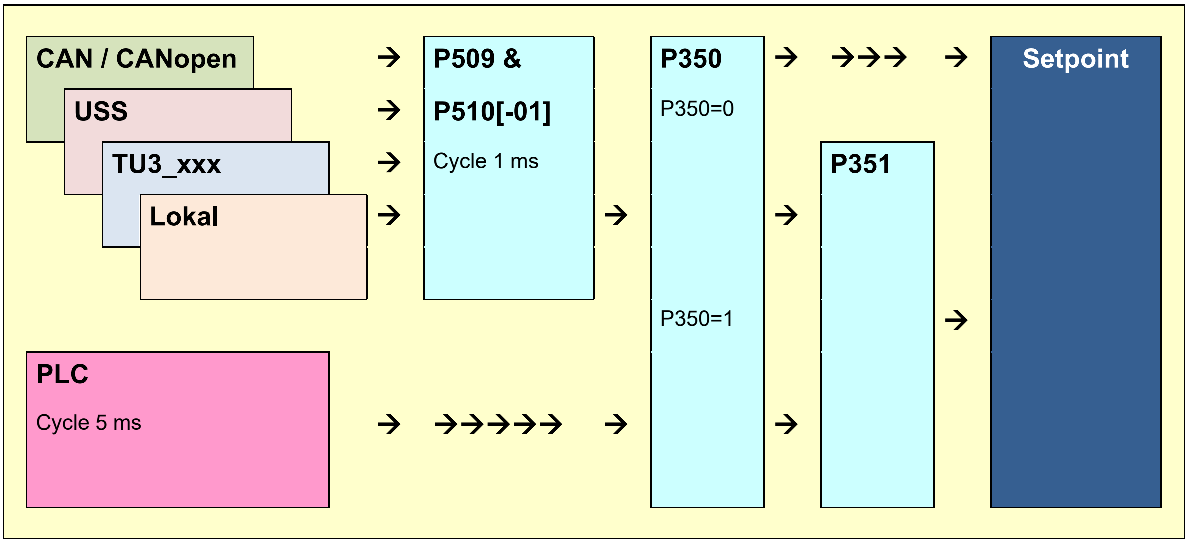
If the Motion blocks MC_Power, MC_Reset, MC_MoveVelocity, MC_Move, MC_Home or MC_Stop are used, the process values "PLC_Control_Word" and "PLC_Set_Val1" up to "PLC_Set_Val5" may not be used. Otherwise the values in the list of variables would always overwrite the changes to the function block..

3.1.2.3 Program Task

Execution of the program in the PLC is carried out as a single task. The task is called up cyclically every 5 ms and its maximum duration is 3 ms. If a longer program cannot be executed in this time, the program is interrupted and continued in the next 5 ms task.

3.1.2.4 Setpoint processing

The inverter has a variety of setpoint sources, which are ultimately linked via several parameters to form a frequency inverter setpoint.



If the PLC is activated (P350=1) preselection of setpoints from external sources (main setpoints) is carried out via P509 and P510[-01] Via P351, a final decision is made as to which setpoints from the PLC or values input via P509/P510[-01] are used. A mixture of both is also possible. No changes to the auxiliary setpoints (P510[-02]) are associated with the PLC function. All auxiliary setpoint sources and the PLC transfer their auxiliary setpoint to the frequency inverter with equal priority.

3.1.2.5 Data processing via accumulator

The accumulator forms the central computing unit of the PLC. Almost all AWL commands only function in association with the accumulator. The PLC has three accumulators. These are the 23 Bit Accumulator 1 and Accumulator 2 and the AE in BOOL format. The AE is used for all boolean loading, saving and comparison operations. If a boolean value is loaded, it is depicted in the AE Comparison operations transfer their results to the AE and conditional jumps are triggered by the AE. Accumulator 1 and Accumulator 2 are used for all operands in the data format BYTE, INT and DINT. Accumulator 1 is the main accumulator and Accumulator 2 is only used for auxiliary functions. All loading and storage operands are handled by Accumulator 1. All arithmetic operands save their results in Accumulator 1. With each Load command, the contents of Accumulator 1 are moved to Accumulator 2. A subsequent operator can link the two accumulators together or evaluate them and save the result in Accumulator 1, which in the following will generally be referred to as the "accumulator".

3.1.3 Scope of functions

The PLC supports a wide range or operators, functions and standard function modules, which are defined in IEC61131-3. There is a detailed description in the following sections. In addition, the function blocks which are also supported are explained.

3.1.3.1 Motion Control Lib

The Motion Control Lib is based on the PLCopen specification "Function blocks for motion control". This mainly contains function blocks which are used to move the drive. In addition, function blocks for reading and writing of parameters of the device are also provided.

3.1.3.2 Electronic gear with Flying Saw

The frequency inverter is equipped with the functions Electronic gear unit (synchronous operation in positioning mode) and Flying saw. Via these functions the inverter can follow another drive unit with angular synchronism. As well as this, with the additional function Flying saw it is possible to synchronize to the precise position of a moving drive unit. The operating mode Electronic gear unit can be started and stopped at any time. This enables a combination of conventional position control with its move commands and gear unit functions. For the gear function a NORDAC vector with internal CAN bus is required on the master axis.

3.1.3.3 Visualisation

Visualisation of the operating status and the parameterisation of the frequency inverter is possible with the aid of a ControlBox or a ParameterBox. Alternatively, the CANopen Master functionality of the PLC CAN bus panel can be used to display information.

ControlBox

The simplest version for visualisation is the ControlBox. The 4-digit display and the keyboard status can be accessed via two process values. This enables simple HMI applications to be implemented very quickly. P001 must be set to "PLC-ControlBox Value" so that the PLC can access the display. A further special feature is that the parameter menu is no longer accessed via the arrow keys. Instead, the "On" and "Enter" keys must be pressed simultaneously.

ParameterBox

In visualisation mode, each of the 80 characters in the ParameterBox display (4 rows of 20 characters) can be set via the PLC. It is possible to transfer both numbers and texts. In addition keyboard entries on the ParameterBox can be processed by the PLC. This enables the implementation of more complex HMI functions (display of actual values, change of window, transfer or setpoints etc.). Access to the ParameterBox display is obtained via the function blocks in the PLC. Visualisation is via the operating value display of the Parameter Box. The content of the operating value display is set via the ParameterBox parameter P1003. This parameter can be found under the main menu item "Display". P1003 must be set to the value "PLC display". After this, the operating value display can be selected again by means of the right and left arrow keys. The display controlled by the PLC is then shown. This setting remains in effect even after a further switch-on.

3.1.3.4 Process controller

The process controller is a PID-T1 controller with a limited output size. With the aid of this function module in the PLC it is possible to simply set up complex control functions, by means of which various processes, e.g. pressure regulation, can be implemented in a considerably more elegant manner than with the commonly used two-point controllers.

3.1.3.5 CANopen communication

In addition to the standard communication channels, the PLC provides further possibilities for communication. Via the CAN bus interface of the frequency inverter, it can set up additional communications with other devices. The protocol which is used for this is CANopen. Communications are restricted to PDO data transfer and NMT commands. The standard CANopen inverter communication via SDO, PDO1, PDO2 and Broadcast remains unaffected by this PLC function.

PDO (Process Data Objects)

Other frequency inverters can be controlled and monitored via PDO. However, it is also possible to connect devices from other manufacturers to the PLC. These may be IO modules, CANopen encoders, panels, etc. With this, the number of inputs/outputs of the frequency encoder can be extended as far as is required; analog outputs would then be possible.

NMT (Network Management Objects)

All CANopen devices must be set to the CANopen bus state "Operational" by the bus master. PDO communication is only possible in this bus state. If there is no bus master in the CANopen bus, this must be performed by the PLC. The function module FB_NMT is available for this purpose.

3.2 Creation of PLC programs

Creation of the PLC programs is carried out exclusively via the PC program NORDCON. The PLC editor is opened either via the menu item "File/New/PLC program" or via the symbol . This button is only active if a device with PLC functionality forms the focus of the device overview.

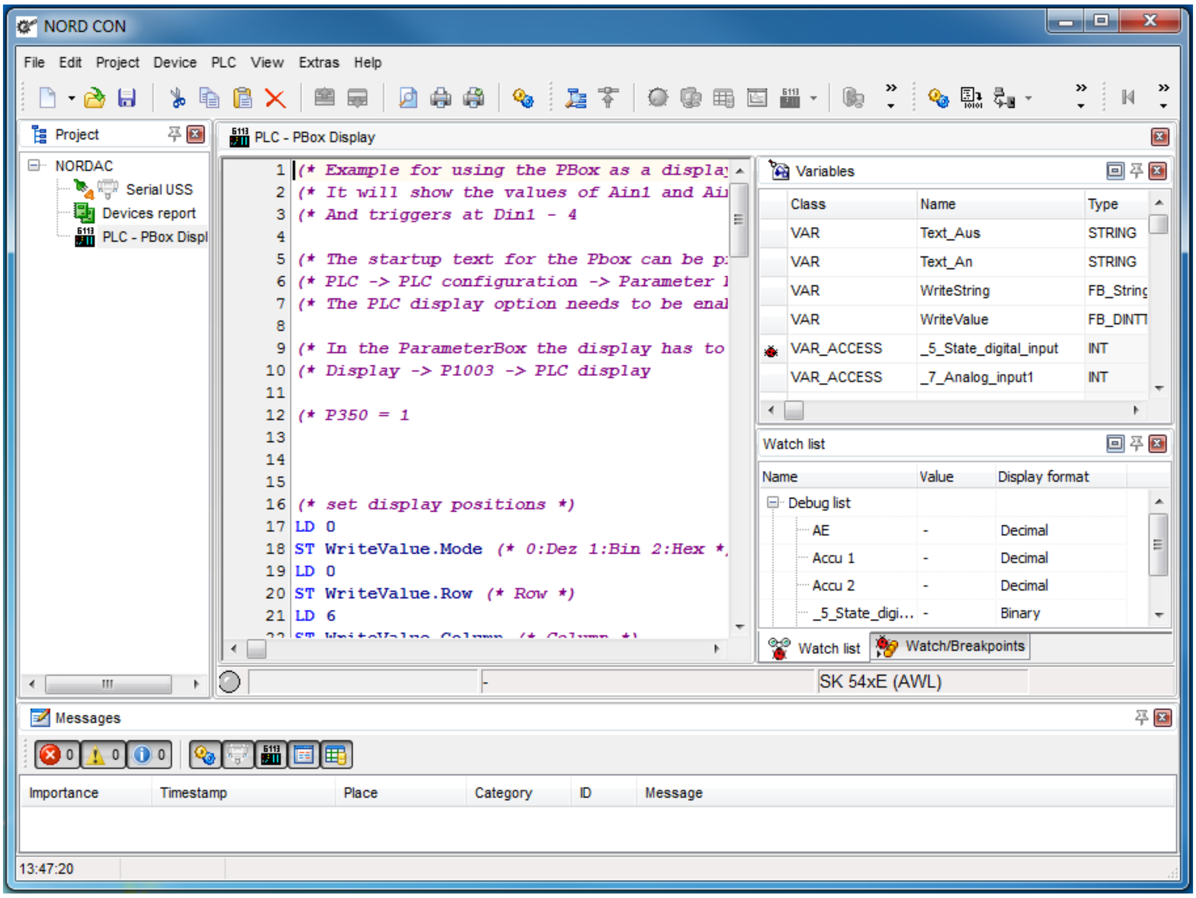
3.2.1 Loading, saving and printing

The functions Load, Save and Print are carried out via the appropriate entries in the main menu or in the symbol bars. When opening takes place, it is advisable to set the file type to "PLC Program" (*.awlx) in the "Open" dialogue. With this, only files which can be read by the PLC editor are displayed. If the PLC program which has been created is to be saved, the PLC Editor window must be active. The PLC program is saved by actuating "Save" or "Save as". With the "Save as" operation, this can also be detected from the entry of the file type (Program PLC (*.awlx)). The appropriate PLC window must be active in order to print the PLC program. Printout is then started via "File/Print" or the appropriate symbol.

PLC programs can also be saved as a backed-up PLC program. To do this, the user must set the file type to “Backed-up AWL files” or “Backed up ST files” in the file selection dialogue. Then the PLC program is saved in an encrypted (*.awls or *.nsts) and normal (*.awlx, *.nstx) version. The encrypted PLC program can only be transmitted to the device (see).

3.2.2 Editor

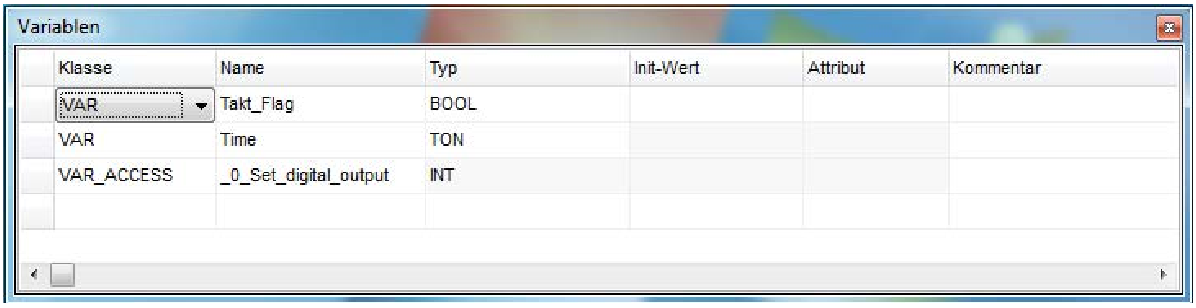
The PLC Editor is divided into four different windows.



The individual windows are described in more detail in the following sections.

3.2.2.1 Variables and FB declaration

All the variables, process values and function blocks which are required by the program are declared in this window.



Variables

Variables are created by setting the Class "VAR". The Name of the variable can be freely selected. In the Type field, a selection between BOOL, BYTE, INT and DINT can be made. A starting initialisation can be entered under Init-Value.

Process values

These are created by selecting the entry "VAR_ACCESS" under Class. The Name is not freely selectable and the field Init-Value is barred for this type.

Function modules

The entry "VAR" is selected under Class. The Name for the relevant instance of the function module (FB) can be freely selected. The required FB is selected under Type. An Init-Value cannot be set for function modules.

All menu items which relate to the variable window can be called up via the context menu. Via this, entries can be added and deleted. Variables and process variables for monitoring (Watchdog function) or debugging (Breakpoint) can be activated.

3.2.2.2 Input window

The input window is used to enter the program and to display the AWL program. It is provided with the following functions:

- Highlight syntax
- Bookmark
- Declaration of variables
- Debugging

Syntax Highlighting

If the command and the variable which is assigned to it are recognised by the Editor, the command is displayed in blue and the variable in black. As long as this is not the case, the display is in thin black italics.

Bookmarks

As programs in the Editor may be of considerable length, it is possible to mark important points in the program with the function Bookmark and to jump directly to these points. The cursor must be located in the relevant line in order to mark it. Via the menu item "Switch bookmark" (right mouse button menu) the line is marked with the required bookmark. The bookmark is accessed via the menu item "Go to bookmark".

Declaring Variables

Via the Editor menu "Add Variable" (right mouse button) new variables can be declared using the Editor.

Debugging

For the Debugging function, the positions of the breakpoints and watchpoints are specified in the Editor. This can be done via the menu items "Switch breakpoint" (Breakpoints) and "Switch monitoring point" (Watchpoints). The position of Breakpoints can also be specified by clicking on the left border of the Editor window. Variables and process values which are to be read out from the frequency inverter during debugging must be marked. This can be done in the Editor via the menu items "Debug variable" and "Watch variable". For this, the relevant variable must be marked before the required menu item is selected.

3.2.2.3 Watch and Breakpoint display window

This window has two tabs, which are explained below.

Holding points

This window displays all of the breakpoints and watchpoints which have been set. These can be switched on and off via the checkboxes and deleted with the "Delete key". A corresponding menu can be called up with the right mouse button.

Observation list

This displays all of the variables which have been selected for observation. The current content is displayed in the Value column. The display format can be selected with the Display column.

3.2.2.4 PLC message window

All PLC status and error messages are entered in this window. In case of a correctly translated program the message "Translated without error" is displayed. The use of resources is shown on the line below this. In case of errors in the PLC program, the message "Error X" is displayed. The number of errors is shown in X. The following lines show the specific error message in the format:

[Line number]: Error description

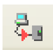
3.2.3 Transfer PLC program to device

There are several ways to transfer a PLC program to the device.

Transfer PLC program directly:

1. Select device in the project tree
2. Open popup menu (press the right mouse button)
3. Execute function "Transfer PLC program to device" 
4. Select file in the file selection dialogue and press "Open"

Transfer PLC program with PLC editor (offline):

1. Open PLC program (File->Open)
2. Connect PLC editor with a device (PLC->Connect)
3. Translate PLC program
4. Transfer PLC program to device 

Transfer PLC program with PLC editor (online):

1. Select device in the project tree
2. Open PLC editor 
3. Open PLC program
4. Import the file into online view
5. Translate PLC program
6. Transfer PLC program to device

Information **SK 1xxE-FDS – limited number of writing cycles**

In the devices SK 155E-FDS / SK 175E-FDS flash is used as a storage medium. The number of write cycles of Flash memory is very limited. By default, the program is loaded into the RAM. It can then be started and tested. If the PLC is then restarted, the program must be re-loaded to the device to initialize the PLC variables. Should the program be permanently stored in the device, you must execute the function "Transfer and store program to device".

3.2.4 Debugging

As programs only rarely function the very first time, the PLC provides several possibilities for finding faults. These possibilities can be roughly divided into two categories, which are described in detail below.

3.2.4.1 Observation points (Watchpoints)

The simplest debugging variant is the Watchpoint function. This provides a rapid overview of the behaviour of several variables. For this, an observation point is set at an arbitrary point in the program. When the PLC processes this line, up to 5 values are saved and displayed in the observation list (window "Observation List") The 5 values to be observed can be selected in the entry window or in the variable window using the context menu.

Information

In the current version, variables of functions cannot be added to the watch list!

3.2.4.2 Holding points (Breakpoints)

Via holding points it is possible to deliberately stop the PLC program at a specific line of the program. If the PLC runs into a Breakpoint, the AE, Accumulator 1 and Accumulator 2 are read out, as well as all variables which have been selected via the menu item "Debug variables". Up to 5 Breakpoints can be set in a PLC  program. This function is started via the  symbol. The program now runs until a holding point is triggered. Further actuation of the symbol bar allows the program to continue running until it reaches the next holding point. If the program is to continue running, the symbol is actuated.

3.2.4.3 Single Step

With this debugging method it is possible to execute the PLC program line for line. With each individual step, all the selected variables are read out of the PLC of the device and displayed in the "Observation list" window. The values to be observed can be selected in the input window or the variable window by means of the right mouse button menu. The condition for debugging in single steps is that at least one Breakpoint has been set before starting debugging. The debugging mode is switched on by actuating the  symbol. Only when the program has run into the first breakpoint, can the following lines be debugged via the  symbol. Some command lines contain several individual commands. Because of this, two or more individual steps may be processed before the step indicator jumps forward in the entry window. The actual position is shown by a small arrow in the left PLC Editor window. When the  symbol is actuated, the program continues running until the next holding point. If the program is to continue running, the  symbol is actuated.

3.2.5 PLC configuration

The PLC configuration dialogue is opened via the  symbol. Here, basic settings for the PLC can be made, which are described in further detail below.

Cycle time monitoring

This function monitors the maximum processing time for a PLC cycle. With this, unintended continuous program loops in the PLC program can be caught. Error E22.4 is triggered in the frequency inverter if this time is exceeded.

Allow ParameterBox function module

If visualisation via the ParameterBox is to be performed in the PLC program, this option must be enabled. Otherwise the corresponding function blocks generate a Compiler Error when the frequency inverter is started.

Invalid control data

The PLC can evaluate control words which are received from the possible bus systems. However, the control words can only get through if the bit "PZD valid" (Bit 10) is set. This option must be activated if control words which are not compliant with the USS protocol are to be evaluated by the PLC. Bit 10 in the first word is then no longer queried.

Do not pause the system time at holding point

The system time is paused during debugging if the PLC is in the holding point or in single step mode. The system time forms the basis for all timers in the PLC. This function must be activated if the system time is to continue running during debugging.

3.3 Function blocks

Function blocks are small programs, which can save their status values in internal variables. Because of this, a separate instance must be created in the NORDCON variable list for each function block. E.g. if a timer is to monitor 3 times in parallel, it must also be set up three times in the list of variables.

Information **Detecting a signal edge**

In order for the following function blocks to detect an edge at the input, it is necessary for the function call-up to be carried out twice with different statuses at the input.

3.3.1 CANopen

The PLC can configure, monitor and transmit on PDO channels via function blocks. The PDO can transmit or receive up to 8 bytes of process data via a PDO. Each of these PDOs is accessed via an individual address (COB-ID). Up to 20 PDOs can be configured in the PLC. For simpler operation, the COB-ID is not entered directly. Instead, the device address and the PDO number are communicated to the FB. The resulting COB-ID is determined on the basis of the Pre-Defined Connection Set (CiA DS301). This results in the following possible COB-IDs for the PLC.

Transmit PDO		**Receive PDO**	
PDO	**COB-ID**	**PDO**	**COB-ID**
PDO1	200h + Device address	PDO1	180h + Device address
PDO2	300h + Device address	PDO2	280h + Device address
PDO3	400h + Device address	PDO3	380h + Device address
PDO4	500h + Device address	PDO4	480h + Device address

NORD Frequency inverter use PDO1 to communicate process data. PDO2 is only used for setpoint/actual value 4 and 5.

3.3.1.1 Overview

Function module	**Description**
FB_PDOConfig	PDO configuration
FB_PDOSend	Transmit PDO
FB_PDOReceive	Receive PDO
FB_NMT	Enable and disable PDO

3.3.1.2 FB_NMT

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X		X	X	X	X

After a *Power UP* all CAN participants are in the Pre-Operational bus state. In this state, they can neither transmit nor receive a PDO. In order for the PLC to be able to communicate with other participants on the CAN bus, these must be set to the Operational state. Usually, this is realised by the bus master. If there is no bus master, this task can be taken over by the FB_NMT. The states of all participants connected to the bus can be controlled via the inputs **PRE**, **OPE** or **STOP**. The inputs are applied with a positive flank on **EXECUTE**. The function must be called until the output **DONE** or **ERROR** has been set to 1.

If the **ERROR** is set to 1, there is either no 24 V supply to the RJ45 CAN socket of the inverter, or the CAN driver of the inverter is in the status *Bus off*. With a negative flank on **EXECUTE**, all outputs are reset to 0.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
EXECUTE	Execute	BOOL	DONE	NMT command is transmitted	BOOL
PRE	Set all participants to Pre-Operational state	BOOL	ERROR	Error in FB	BOOL
OPE	Set all participants to Operational state	BOOL			
STOP	Set all participants to Stopped state	BOOL			

3.3.1.3 FB_PDOConfig

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X		X	X	X	X

The PDOs are configured with this FB. With an instance of this function, all of the required PDOs can be configured. The FB must only be called up once for each PDO. Up to 20 PDOs can be set up. Each PDO has its own parameterisation. The assignment of the PDOs in the other CANopen FBs is carried out via the Messagebox number. The **TARGETID** represents the address of the device. With NORD Frequency inverter, it is set in P515 or via DIP switches. The required Messagebox number is entered under PDO (see Introduction). **LENGTH** specifies the transmission length of a PDO. The transmission/reception direction is specified via **DIR**. The data is adopted with a positive flank on the **EXECUTE** input. The **DONE** output can be queried immediately after the call-up of the FB. If **DONE** is set to 1, the PDO channel has been configured. If **ERROR** = 1, there was a problem, whose precise cause is stored in **ERRORID**. With a negative flank on **EXECUTE**, all outputs are reset to 0.

Transmit PDO		**Monitored PDO**	
PDO	**COB-ID**	**PDO**	**COB-ID**
PDO1	200h + Device address	PDO1	180h + Device address
PDO2	300h + Device address	PDO2	280h + Device address
PDO3	400h + Device address	PDO3	380h + Device address
PDO4	500h + Device address	PDO4	480h + Device address
PDO5	180h + Device address	PDO5	200h + Device address
PDO6	280h + Device address	PDO6	300h + Device address
PDO7	380h + Device address	PDO7	400h + Device address
PDO8	480h + Device address	PDO8	500h + Device address

VAR_INPUT			VAR_OUTPUT		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
EXECUTE	Execute	BOOL	**DONE**	PDO configured	BOOL
NUMBER	Messagebox number Value range = 0 to 19	BYTE	**ERROR**	Error in FB	BOOL
TARGETID	Device address Value range = 1 to 127	BYTE	**ERRORID**	Error code	INT
PDO	PDO Value range = 1 to 4	BYTE			
LENGTH	PDO length Value range = 1 to 8	BYTE			
DIR	Transmit or receive Transmit = 1 / Receive = 0	BOOL			

ERRORID	Explanation
0	No error
1800h	Number value range exceeded
1801h	TARGETID value range exceeded
1802h	PDO value range exceeded
1803h	LENGT value range exceeded

Information **No dual use of the CAN ID**

CAN IDs already used by the device may not be parameterised!

Relevant reception addresses:

- CAN ID = 0x180 + P515[-01] PDO1
- CAN ID = 0x180 + P515[-01]+1 CAN ID for absolute encoder
- CAN ID = 0x280 + P515[-01] PDO2

Relevant transmission addresses:

- CAN ID = 0x200 + P515[-01] PDO1
- CAN ID = 0x300 + P515[-01] PDO2

Example in ST:

```
(* Configure PDO *)
PDOConfig(
    Execute := TRUE,
    (* Configure Messagebox 1 *)
    Number := 1,
    (* Set CAN node number *)
    TargetID := 50,
    (* Select PDO (Standard for PDO1 control word, setpoint1, setpoint2, setpoint3) *)
    PDO := 1,
    (* Specify length of data (Standard for PDO1 is 8 *)
    LENGTH := 8,
    (* Transmit *)
    Dir := 1);

or

(* Configure PDO *)
PDOConfig(
    Execute := TRUE,
    (* Configure Messagebox 1 *)
    Number := 2,
    (* Set CAN node number *)
    TargetID := 50,
    (* Select PDO (Standard for PDO2 setpoint4, setpoint5 SK540E) *)
    PDO := 2,
    (* Specify length of data (Standard for PDO2 is 4 *)
    LENGTH := 4,
    (* Transmit *)
    Dir := 1);

or

(* Configure PDO *)
PDOConfig(
    Execute := TRUE,
    (* Configure Messagebox 2 *)
    Number := 2,
    (* Set CAN node number *)
    TargetID := 50,
    (* Select PDO (Standard for PDO1 status word, actual value1, actual value2, actual
     value3) *)
    PDO := 1,
    (* Specify length of data (Standard for PDO1 is 8 *)
    LENGTH := 8,
    (* Receive *)
    Dir := 0);
```

3.3.1.4 FB_PDOReceive

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X		X	X	X	X

This FB monitors a previously configured PDO channel for incoming messages. Monitoring starts if the **ENABLE** input is set to 1. After the function has been called up, the **NEW** output must be checked. If it changes to 1, a new message has arrived. The **NEW** output is deleted with the next call-up of the function. The data which have been received are shown in **WORD1** to **WORD4**. The PDO channel can be monitored for cyclical reception via **TIME**. If a value between 1 and 32767 ms is entered in **TIME**, a message must be received during this period. Otherwise, the FB changes into the error state (**ERROR** = 1). This function can be disabled with the value 0. The monitoring timer runs in steps of 5 ms. In case of error, **ERROR** is set to 1. In this case, **DONE** is 0. The corresponding error code is then valid in **ERRORID**. With a negative flank on **ENABLE**, **DONE**, **ERROR** and **ERRORID** are reset.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
ENABLE	Execute	BOOL	**NEW**	New PDO received	BOOL
NUMBER	Messagebox number Value range = 0 to 19	BYTE	**ERROR**	Error in FB	BOOL
TIME	Watchdog function Value range = 0 to 32767 0 = Disabled 1 to 32767 = Monitoring time	INT	**ERRORID**	Error code	INT
			WORD1	Received data Word 1	INT
			WORD2	Received data Word 2	INT
			WORD3	Received data Word 3	INT
			WORD4	Received data Word 4	INT

ERRORID	**Explanation**
0	No error
1800h	Number value range exceeded
1804h	Selected box is not configured correctly
1805h	No 24 V for bus driver or bus driver is in “Bus off” state
1807h	Reception timeout (Watchdog function)

Information **PLC cycle time**

The PLC cycle is about 5 ms, i.e. with one call-up of the function in the PLC program, a CAN message can only be read every 5 ms. Messages may be overwritten if several messages are transmitted in quick succession.

Example in ST:

```
IF bFirstTime THEN
  (* Set device to Pre-Operational status *)
  NMT(Execute := TRUE, OPE := TRUE);
  IF not NMT.Done THEN
    RETURN;
  END_IF;

  (* Configure PDO *)
  PDOConfig(
    Execute := TRUE,
    (* Configure Messagebox 2 *)
    Number := 2,
    (* Set CAN node number *)
    TargetID := 50,
    (* Select PDO (Standard for PDO1 status word, actual value1, actual value2, actual
     value3) *)
    PDO := 1,
    (* Specify length of data (Standard for PDO1 is 8 *)
    Length := 8,
    (* Receive *)
    Dir := 0);
END_IF;

(* Read out status and actual values *)
PDOReceive(Enable := TRUE, Number := 2);
IF PDOReceive.New THEN
  State := PDOReceive.Word1;
  Sollwert1 := PDOReceive.Word2;
  Sollwert2 := PDOReceive.Word3;
  Sollwert3 := PDOReceive.Word4;
END_IF
```

3.3.1.5 FB_PDOSend

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X		X	X	X	X

With this FB, PDOs can be transmitted on a previously configured channel. These can be transmitted once or cyclically. The data to be transmitted is entered in **WORD1** to **WORD4**. PDOs can be transmitted irrespective of the frequency inverter's CANopen state. The previously configured PDO channel is selected via **NUMBER**. The data to be transmitted is entered in **WORD1** to **WORD4**. Single (setting = 0) or cyclical transmission can be selected via **CYCLE**. The PDO is sent with a positive flank on **EXECUTE**. If **DONE** = 1, all entries were correct and the PDO is transmitted. If **ERROR** = 1, there was a problem. The precise cause is stored in **ERRORID**. All outputs are reset with a negative flank on **EXECUTE**. The time base of the PLC is 5 ms; this also applies for the **CYCLE** input. Only transmission cycles with a multiple of 5 ms can be implemented.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
EXECUTE	Execute	**BOOL**	**DONE**	PDO transmitted = 1	BOOL
NUMBER	Messagebox number Value range = 0 to 19	**BYTE**	**ERROR**	Error in FB	BOOL
CYCLE	Transmission cycle Value range = 0 to 255 0 = Disabled 1 to 255 = Transmission cycle in ms	**BYTE**	**ERRORID**	Error code	INT
WORD1	Transmission data Word 1	**INT**			
WORD2	Transmission data Word 2	**INT**			
WORD3	Transmission data Word 3	**INT**			
WORD4	Transmission data Word 4	**INT**			

ERRORID	**Explanation**
0	No error
1800h	Number value range exceeded
1804h	Selected box is not configured correctly
1805h	No 24 V for bus driver or bus driver is in "Bus off" state

If **DONE** changes to 1, the message to be transmitted has been applied by the CAN module, but has not yet been sent. The actual transmission runs in parallel in the background. If several messages are now to be transmitted directly in sequence via an FB, it may be the case that the previous message has not yet been transmitted upon the new call-up. This can be seen by the fact that neither the **DONE** nor the **ERROR** signal have been set to 1 after the **CAL** call-up. The **CAL** call-up can be repeated until one of the two signals changes to 1. If several different CAN IDs are to be written on via a single FB, this is possible with a new configuration of the FB. However, this must not be done in the same

PLC cycle as the transmission. Otherwise, there is a danger that the message to be transmitted will be deleted during configuration by the FB_PDOConfig.

Example in ST:

```
IF bFirstTime THEN
  (* Set device to Pre-Operational status *)
  NMT(Execute := TRUE, OPE := TRUE);
  IF not NMT.Done THEN
    RETURN;
  END_IF;

  (* Configure PDO*)
  PDOConfig(
    Execute := TRUE,
    (*Configure Messagebox 1*)
    Number := 1,
    (* Set CAN node number *)
    TargetID := 50,
    (* Select PDO (Standard for PDO1 status word, actual value1, actual value2, actual
     value3) *)
    PDO := 1,
    (*Specify length of data (Standard for PDO1 is 8*)
    LENGTH := 8,
    (* Transmit *)
    Dir := 1);

  IF not PDOConfig.Done THEN
    RETURN;
  END_IF;

  (* Transmit PDO - Set Device control word to status "Ready to switch-on" *)
  PDOSend(Execute := TRUE, Number := 1, Word1 := 1150, Word2 := 0, Word3 := 0, Word4 := 0);
  IF NOT PDOSend.Done THEN
    RETURN;
  END_IF;

  PDOSend(Execute := FALSE);
  bFirstTime := FALSE;
END_IF;

CASE State OF
  0:
      (* Has digital input 1 been set? *)
      IF _5_State_digital_input.0 THEN
        (*Transmit PDO - Set Device control word to status "Ready to switch-on" *)
        PDOSend(Execute := TRUE, Number := 1, Word1 := 1150, Word2 := 0, Word3 := 0,
          Word4 := 0);
        State := 10;
        RETURN;
      END_IF;

      (*Has digital input 2 been set? *)
      IF _5_State_digital_input.1 THEN
        (* Transmit PDO - Enable device with 50% max. frequency *)
        PDOSend(Execute := TRUE, Number := 1, Word1 := 1151, Word2 := 16#2000, Word3 := 0,
          Word4 := 0);
        State := 10;
        RETURN;
      END_IF;

  10:
      PDOSend;
      IF PDOSend.Done THEN
        PDOSend(Execute := FALSE);
        State := 0;
      END_IF;
END_CASE;
```

3.3.2 Electronic gear unit with flying saw

For the electronic gear unit ("angularly synchronised operation") and the sub-function flying saw there are two function blocks which enable control of these functions. In addition, various parameters must be set for the correct execution of the two function blocks in the master and slave frequency inverters. An example of this is shown in the following table (explained by the example of a SK 540E).

Master FI			**Slave FI**		
Parameter	Settings	Description	Parameter	Settings	Description
P502[-01]	20	Setpoint frequency according to freq. Ramp	P509	10 *	CANopen Broadcast *
P502[-02]	15	Actual position in incl. High word	P510[-01]	10	CANopen Broadcast
P502[-03]	10	Actual position in incl. Low word	P510[-02]	10	CANopen Broadcast
P503	3	CANopen	P505	0	0,0 Hz
P505	0	0.0 Hz	P515[-02]	P515[-03]Master	Broadcast Slave address
P514	5	250 kBaud (min. 100 kBaud)	P546[-01]	4	Frequency addition
P515[-03]	P515[-02] Slave	Broadcast master address	P546[-02]	24	Setpoint pos. Incl. High Word
			P546[-03]	23	Setpoint pos. Incl. Low Word
			P600	1.2	Position control ON
			Only for FB_Gearing		
			P553[-01]	21	Position setpoint pos. Low word
			P553[-02]	22	Position setpoint pos. High word
* (P509) must not necessarily be set to {10} "CANopen Broadcast". However, in this case the Master (P502 [-01]) must be set to {21} "Actual frequency without slip".					

Information **Actual position – transmission format**

The actual position of the master MUST be communicated in "Increments" (Inc) format.

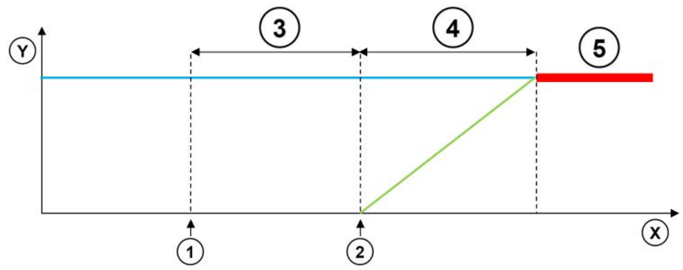
3.3.2.1 Overview

Function module	Description
FB_Gearing	FB for simple gear unit function
FB_FlyingSaw	FB for gear unit function with Flying Saw

3.3.2.2 FB_FlyingSaw

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	On+	X	X	X	

The flying saw function is an extension of the gear unit function. With the aid of this function it is possible to precisely synchronise to a running drive unit. In contrast to FB_Gearing, synchronisation is relative, i.e. the slave axis moves synchronously to the position of the master which applied at the start of the “Flying Saw”. The synchronisation process is illustrated in the figure below.



1	Position of the initiator
2	Starting point of the slave
3	Distance of the initiator from the start position of the slave frequency inverter
4	Acceleration
5	Synchronous mode for both drives
X	Position
Y	Speed

If the function is started, the slave frequency inverter accelerates to the speed of the master axis. The acceleration ramp is specified via the **ACCELERATION** path. At low speeds, the ramp is flatter and at high speeds, there is a steep ramp for the slave frequency inverter. The acceleration path is stated in revolutions (1000 = 1.000 rev.) if P553 is specified as the setpoint position. If the setpoint position INC is used for P553, the acceleration path is specified in increments.

If the initiator, with the distance saved in **ACCELERATION**, is set in front of the position of the slave drive, the slave is precisely synchronised with the triggering position from the master drive.

The FB must be switched on via the **ENABLE** input. The function can be started either via the digital input (P420[-xx]=64, *Start flying saw*) or via **EXECUTE**. The frequency inverter then accelerates to the speed of the master axis. When synchronisation with the master axis is achieved, the **DONE** output is switched to 1.

Via the **STOP** input or the digital input function P420[-xx] = 77, *Flying saw stopped*, the gear unit function is switched off, the frequency inverter decelerates to 0 Hz and remains at a standstill. Via the **HOME** input, the inverter is made to move to the absolute position 0. After termination of the **HOME** or **STOP** command, the relevant allocated output is active. The gear unit function can be restarted by reactivating **EXECUTE** or the digital input. With the digital input function (P420[-xx] = 63, *Synchronous mode off*), the gear unit function can be stopped and then moved to the absolute position 0.

If the function is interrupted by the MC_Stop function, **ABORT** is set to 1. In case of error, **ERROR** is set to 1 and the error code is set in **ERRORID**. These three outputs are reset if **ENABLE** is switched to 0.

VAR_INPUT			VAR_OUTPUT		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
ENABLE	Enable	BOOL	**VALID**	Specified set point frequency reached	BOOL
EXECUTE	Start of synchronisation	BOOL	**DONEHOME**	Home run completed	
STOP	Stop synchronisation	BOOL	**DONESTOP**	Stop command executed	
HOME	Moves to position 0	BOOL	**ABORT**	Command cancelled	BOOL
ACCELERATION	Acceleration path (1 rev. = 1,000)	DINT	**ERROR**	Error in FB	BOOL
			ERRORID	Error code	INT

ERRORID	Explanation
0	No error
1000h	FI is not enabled
1200h	Position control is not activated

3.3.2.3 FB_Gearing

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	On+	X	X	X	

Via function module FB_Gearing, the position and speed of the frequency inverter can be synchronised to that of a master inverter. The slave which uses this function always follows the movements of the master inverter.

Synchronisation is absolute, i.e. the positions of the slave and the master are always the same.

Information

If the slave is switched to gear unit mode at a different position than the master, the slave moves with maximum frequency to the master's position.

If a gear ratio is specified, this also results in a new position when switched on again.

The position value to which synchronisation is carried out, as well as the speed, must be transmitted via the Broadcast channel. The function is enabled via the **ENABLE** input. For this, the position control and the output stage must be enabled. The output stage can be enabled, e.g. with the MC_Power function. If **ENABLE** is set to 0, the frequency inverter decelerates to 0 Hz and remains at a standstill. The inverter is now in position control mode again. If MC_Stop is activated, the frequency inverter exits the gear unit mode and the **ABORT** output changes to 1. In case of errors in the FB, **ERROR** changes to 1 and the error cause is indicated in **ERRORID.** By setting **ENABLE** to 0, **ERROR, ERRORID** and **ABORT** can be reset.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
ENABLE	Synchronous mode active	BOOL	**VALID**	Gear unit function is active	BOOL
RELATIVE	Relative mode (V2.1 and above)	BOOL	**ABORT**	Command cancelled	BOOL
			ERROR	Error in FB	BOOL
			ERRORID	Error code	INT

ERRORID	**Explanation**
0	No error
1000h	FI is not enabled
1200h	Position control is not activated
1201h	The PLC set value position High is not parameterised
1202h	The PLC set value position Low is not parameterised

3.3.3 Motion Control

The Motion Control Lib is based on the PLCopen specification "Function blocks for motion control". It contains function blocks for controlling and moving a frequency inverter and provides access to its parameters. Several settings must be made to the parameters of the device in order for the Motion Blocks to function.

Function blocks	Required settings
MC_MoveVelocity	P350 = PLC active P351 = Main setpoint comes from the PLC P553 [-xx] = Setpoint frequency P600 = Position control (positioning mode) is disabled
MC_MoveAbsolute MC_MoveRelative MC_MoveAdditive MC_Home	P350 = PLC active P351 = Main setpoint comes from the PLC P600 = Position control (positioning mode) is enabled In P553 [-xx] (PLC_Setpoints) the setpoint position High Word must be parameterised In P553 [-xx] (PLC_Setpoints) the setpoint position Low Word must be parameterised In P553 [-xx] (PLC_Setpoints) the setpoint frequency must be parameterised
MC_Power MC_Reset MC_Stop	P350 = PLC active P351 = Control word comes from the PLC

Information

The PLC_Setpoints 1 to 5 and the PLC control word can also be described via process variables. However, if the Motion Control FBs are used, no corresponding process variable may be declared in the table of variables, as otherwise the outputs of the Motion Control FBs would be overwritten.

Information **Detecting a signal edge**

In order for the following function blocks to detect an edge at the input, it is necessary for the function call-up to be carried out twice with different statuses at the input.

Function blocks	**Description**
MC_ReadParameter	Reading access to parameters of the device
MC_WriteParameter	Writing access to parameters of the device
MC_MoveVelocity	Move command in speed mode
MC_MoveAbsolute	Move command with specification of absolute position
MC_MoveRelative	Move command with specification of relative position
MC_MoveAdditive	Move command with additive specification of position
MC_Home	Starts a home run
MC_Power	Switches the motor voltage on or off
MC_ReadStatus	Status of the device
MC_ReadActualPos	Reads out the actual position
MC_Reset	Error reset in the device
MC_Stop	Stops all active movement commands

3.3.3.1 MC_Control

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	

This function block is used to control the FI and provides the option of producing the FI control word in a form which is somewhat more detailed than is possible with MC_Power. The FI is controlled via the inputs **ENABLE (ENABLE_RIGHT)**, **ENABLE_LEFT, DISABLEVOLTAGE** and **QUICKSTOP**, please refer to the following table.

Module inputs				**Frequency inverter behaviour**
ENABLE (RIGHT)	**ENABLE_ LEFT**	**QUICKSTOP**	**DISABLE VOLTAGE**	
High	Low	Low	Low	The frequency inverter is switched on (enable right).
X	High	Low	Low	The frequency inverter is switched on (enable left).
Low	Low	Low	Low	The frequency inverter decelerates to 0 Hz (P103) and then disconnects the motor from the voltage supply.
X	X	X	High	The frequency inverter is disconnected from the voltage supply immediately and the motor runs to a standstill without deceleration.
X	X	High	Low	The frequency inverter performs a Quick stop (P426) and then disconnects the motor from the voltage supply.

The active parameter set can be set via the input **PARASET**.

If the output is in **STATUS** = 1, the FI is switched on and the motor is supplied with power.

VAR_INPUT			VAR_OUTPUT		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
ENABLE	Enable	BOOL	**STATUS**	Motor supplied with power	BOOL
DISABLEVOLTAGE	Disconnect voltage	BOOL	**ERROR**	Error in FB	BOOL
QUICKSTOP	Quick stop	BOOL	**ERRORID**	Error code	INT
PARASET	Active parameter set Value range: 0 – 3	BYTE			
ENABLE_RIGHT	Enable right (as for **ENABLE**) (SK5xxP)	BOOL			
ENABLE_LEFT	Enable left (SK5xxP)	BOOL			

ERRORID	Explanation
0	No error
1001h	Stop function is active
1300h	The FI is in a state where the selected function cannot be executed.

Example in ST:

```
(* Device enabled with Dig3*)
Control.Enable := _5_State_digital_input.2;
(* Parameter sets are specified via Dig1 and Dig2. *)
Control.ParaSet := INT_TO_BYTE(_5_State_digital_input and 2#11);
Control;
(* Is the device enabled? *)
if Control.Status then
  (* Is a different position to be moved to? *)
  if SaveBit3 <> _5_State_digital_input.3 then
    SaveBit3 := _5_State_digital_input.3;
    if SaveBit3 then
      Move.Position := 500000;
    else
      Move.Position := 0;
    end_if;

    Move(Execute := False);
  end_if;
end_if;

(* Move to position if the device is enabled.  *)
Move(Execute := Control.Status);
```

3.3.3.2 MC_Control_MS

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability								X

This FB is used to control the starter (MS).

Module inputs				**Frequency inverter behaviour**
ENABLE_RIGHT	**ENABLE_LEFT**	**QUICKSTOP**	**DISABLEVOLTAGE**	
High	Low	Low	Low	MS is switched on, clockwise
Low	High	Low	Low	MS is switched on, counter-clockwise
High	High	Low	Low	MS is switched off
Low	Low	Low	Low	MS decelerates to 0 Hz (P103) and then disconnects the motor from the voltage supply
X	X	X	High	MS is disconnected from the voltage supply immediately and the motor runs to a standstill without deceleration
X	X	High	Low	MS performs a Quick stop (P426) and then disconnects the motor from the voltage supply.

(X = The level at the input is irrelevant)

If the output is **STATUS** = 1, the MS is switched on and the motor is supplied with power.

If **OPENBRAKE** is set to 1, the brake is opened.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
ENABLE_RIGHT	Enable right	BOOL	**STATUS**	Motor supplied with power	BOOL
ENABLE_LEFT	Enable left	BOOL	**ERROR**	Error in FB	BOOL
DISABLEVOLTAGE	Disconnect voltage	BOOL	**ERRORID**	Error code	INT
QUICKSTOP	Quick stop	BOOL			
OPENBRAKE	Open brake	BOOL			
ERRORID	**Explanation**				
0	No error				
1001h	Stop function is active				
1300h	MS is in an unexpected state				

3.3.3.3 MC_Home

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability		X	X		X	X	X	X

Causes the frequency inverter to start a reference point run if **EXECUTE** changes from 0 to 1 (flank). The frequency inverter moves with the setpoint frequency which is set in **VELOCITY**. If the input with the position reference signal (P420[-xx] =) becomes active, a change of direction of rotation occurs. On the negative flank of the position reference signal the value in **POSITION** is adopted. The frequency inverter then decelerates to 0 Hz and the **DONE** signal changes to 1. During the entire **HOME** run the **BUSY** output is active. If the input **MODE** is set to **True**, the drive adopts the average value of both positions during the reference point run (positive flank → negative flank) when the reference point switch is passed over and sets this value as the reference point. The drive reverses and therefore stops at the reference point which has been thus determined. The input **POSITION** cannot be used.

If the process is cancelled (e.g. by another MC function module), **COMMANDABORTED** is set.

In case of error, **ERROR** is set to 1. In this case, **DONE** is 0. The corresponding error code is then valid in **ERRORID**.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
EXECUTE	Enable	BOOL	**DONE**	Specified setpoint position reached	BOOL
POSITION	Setpoint position	DINT	**COMMAND-ABORTED**	Command cancelled	BOOL
VELOCITY	Setpoint frequency	INT	**ERROR**	Error in FB	BOOL
MODE (V2.1 and higher)	See below	BOOL	**ERRORID**	Error code	INT
			BUSY	Home run active	BOOL

ERRORID	**Explanation**
0	No error
1000h	FI is not enabled
1200h	Position control is not activated
1201h	The High position is not entered in the PLC setpoint values (P553)
1202h	The Low position is not entered in the PLC setpoint values (P553)
1D00h	Absolute encoders are not supported

3.3.3.4 MC_Home (SK 5xxP)

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X			On+				

Causes the frequency inverter to start a reference point run if **EXECUTE** changes from 0 to 1 (flank). The frequency inverter moves with the setpoint frequency which is set in **VELOCITY**. If the input with the position reference signal (P420[-xx] =) becomes active, a change of direction of rotation occurs. On the negative flank of the position reference signal the value in **POSITION** is adopted. The frequency inverter then decelerates to 0 Hz and the **DONE** signal changes to 1. During the entire **HOME** run the **BUSY** output is active.

If the process is cancelled (e.g. by another MC function module), **COMMANDABORTED** is set.

In case of error, **ERROR** is set to 1. In this case, **DONE** is 0. The corresponding error code is then valid in **ERRORID**.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
EXECUTE	Enable	BOOL	**DONE**	Specified setpoint position reached	BOOL
POSITION	Setpoint position	DINT	**COMMAND-ABORTED**	Command cancelled	BOOL
VELOCITY	Setpoint frequency	INT	**ERROR**	Error in FB	BOOL
MODE	See below	BYTE	**ERRORID**	Error code	INT
			BUSY	Home run active	BOOL

ERRORID	**Explanation**
0	No error
1000h	FI is not enabled
1200h	Position control is not activated
1201h	The High position is not entered in the PLC setpoint values (P553)
1202h	The Low position is not entered in the PLC setpoint values (P553)
1D00h	Absolute encoders are not supported
1D01h	Value range from "Mode" input exceeded or undershot (P623)

Mode

Value	Explanation
1..14	For reference point method see P623
15	Once the reference point has been reached, the drive reverses. When the reference point switch is left (negative flank), this is adopted as the reference point. The reference point is therefore typically in the side of the reference point switch on which the reference point run started. **Note:** If the reference point switch is passed over (switch too narrow, speed too high), this is also taken as the reference point when leaving the reference point switch (negative flank). The reference point is therefore not on the side of the reference point switch from which the reference point run was started. (P623 = [15] Nord method 1)
16	As for 15, however passing over the reference point switch does not result in adoption as the reference point. A negative flank only results in adoption as the reference point after reversal has been completed. The reference point is therefore definitely on the side of the reference point switch from which the reference point run was started. (P623 = [16] Nord method 2)
17	If the reference point switch is passed over during the reference point run (positive flank → negative flank) the drive adopts the average value of both positions and sets this as the reference point. The drive reverses and therefore stops at the reference point which has been thus determined. (P623 = [17] Nord method 3)
18..34	For reference point method see P623

3.3.3.5 MC_MoveAbsolute

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	On+	X	X	X	

Writes a position and speed setpoint to the frequency inverter if **EXECUTE** changes from 0 to 1 (flank). The set point frequency **VELOCITY** is transferred according to the scaling explained in MC_MoveVelocity.

POSITION:

MODE = False:

The setpoint position results from the value transferred into **POSITION**.

MODE = True:

The value transferred into **POSITION** corresponds to the index from parameter P613 increased by 1. The position stored in this parameter index corresponds to the setpoint position.

Example:

Mode = True; Position = 12

The FB moves to the position which is in the current parameter set of P613[-13].

If the inverter has reached the setpoint position, **DONE** is set to 1. **DONE** is deleted by resetting **EXECUTE**. If **EXECUTE** is deleted before the target position is reached, **DONE** is set to 1 for one cycle. During movement to the setpoint position, **BUSY** is active. If the process is cancelled (e.g. by another MC function module), **COMMANDABORTED** is set. In case of error, **ERROR** is set to 1 and the corresponding error code is set in **ERRORID**. In this case, **DONE** is 0. With a negative flank on **EXECUTE**, all outputs are reset to 0.

VAR_INPUT			VAR_OUTPUT		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
EXECUTE	Enable	BOOL	**DONE**	Specified setpoint position reached	BOOL
POSITION	Setpoint position	DINT	**BUSY**	Setpoint position not reached	BOOL
VELOCITY	Setpoint frequency	INT	**COMMAND-ABORTED**	Command cancelled	BOOL
MODE	Mode Source Setpoint position	BOOL	**ERROR**	Error in FB	BOOL
			ERRORID	Error code	INT

ERRORID	Explanation
0	No error
0x1000	FI is not enabled
0x1200	Position control is not activated
0x1201	The High position is not entered in the PLC setpoint values (P553)
0x1202	The Low position is not entered in the PLC setpoint values (P553)

Example in ST:

```
(* The device is enabled if DIG1 = TRUE *)
Power(Enable := _5_State_digital_input.0);
IF Power.Status THEN
  (* The device is enabled and moves to position 20000 with 50% max. frequency.
     For this action, the motor requires an encoder, and position control must be enabled.
     *)
  MoveAbs(Execute := _5_State_digital_input.1, Velocity := 16#2000, Position := 20000);
END_IF
```

3.3.3.6 MC_MoveAdditive

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	On+	X	X	X	

Except for the **DISTANCE** input, this corresponds in all points with MC_MoveAbsolute. The setpoint position results from the addition of the actual setpoint position and the transferred **DISTANCE**.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
EXECUTE	Enable	BOOL	**DONE**	Specified setpoint position reached	BOOL
DISTANCE	Setpoint position	DINT	**COMMAND-ABORTED**	Command cancelled	BOOL
VELOCITY	Setpoint frequency	INT	**ERROR**	Error in FB	BOOL
MODE	Mode Source Setpoint position	BOOL	**ERRORID**	Error code	INT
			BUSY	Setpoint position not reached	BOOL

ERRORID	**Explanation**
0	No error
1000h	FI is not enabled
1200h	Position control is not activated
1201h	The High position is not entered in the PLC setpoint values (P553)
1202h	The Low position is not entered in the PLC setpoint values (P553)

3.3.3.7 MC_MoveRelative

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	On+	X	X	X	

Except for the **DISTANCE** input, this corresponds in all points with MC_MoveAbsolute. The setpoint position results from the addition of the actual current position and the transferred **DISTANCE**.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
EXECUTE	Enable	BOOL	**DONE**	Specified setpoint position reached	BOOL
DISTANCE	Setpoint position	DINT	**COMMAND-ABORTED**	Command cancelled	BOOL
VELOCITY	Setpoint frequency	INT	**ERROR**	Error in FB	BOOL
MODE	Mode Source Setpoint position	BOOL	**ERRORID**	Error code	INT
			BUSY	Setpoint position not reached	BOOL

ERRORID	**Explanation**
0	No error
1000h	FI is not enabled
1200h	Position control is not activated
1201h	The High position is not entered in the PLC setpoint values (P553)
1202h	The Low position is not entered in the PLC setpoint values (P553)

3.3.3.8 MC_MoveVelocity

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	

Sets the set point frequency for the frequency inverter if **EXECUTE** changes from 0 to 1 (flank). If the frequency inverter has reached the set point frequency, **INVELOCITY** is set to 1. While the FI is accelerating to the set point frequency, the **BUSY** output is active. If **EXECUTE** has already been set to 0, **INVELOCITY** is set to 1 for only one cycle. If the process is cancelled (e.g. by another MC function module), **COMMANDABORTED** is set.

With a negative flank on **EXECUTE**, all outputs are reset to 0.

VELOCITY is entered with scaling according to the following formula:

VELOCITY = (Set point frequency (Hz) × 0x4000) / P105

VAR_INPUT			VAR_OUTPUT		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
EXECUTE	Enable	BOOL	**INVELOCITY**	Specified set point frequency reached	BOOL
VELOCITY	Setpoint frequency	INT	**BUSY**	Set point frequency not yet reached	BOOL
			COMMAND-ABORTED	Command cancelled	BOOL
			ERROR	Error in FB	BOOL
			ERRORID	Error code	INT

ERRORID	Explanation
0	No error
1000h	FI is not enabled
1100h	FI not in speed mode (position control enabled)
1101h	No set point frequency parameterised (P553)

Example IL:

```
CAL Power
CAL Move

LD TRUE
ST Power.Enable

(* Set 20 Hz (Max. 50 Hz) *)
LD DINT#20
MUL 16#4000
DIV 50

DINT_TO_INT
ST Move.Velocity

LD Power.Status
ST Move.Execute
```

Example in ST:

```
(* Device ready for operation if DIG1 set *)
Power(Enable := _5_State_digital_input.0);
IF Power.Status THEN
  (* Device enabled with 50% of max. frequency if DIG2 set *)
  MoveVelocity(Execute := _5_State_digital_input.1, Velocity := 16#2000);
END_IF
```

3.3.3.9 MC_Power

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

The output stage of the device can be switched on and off with this function. If the **ENABLE** input is set to 1, the output stage is enabled. The prerequisite for this is that the device is in “Switch-on inhibit” or “Ready to switch-on” state. If the device is in “Fault” or “Fault response active” state, the fault must first be remedied and acknowledged. Only then, enabling can be carried out via this block. If the device is in “Not ready to switch-on ” state, switch-on is not possible. In all cases, the FB goes into error state and **ENABLE** must be set to 0 to acknowledge the fault.

If the **ENABLE** input is set to 0, the device is switched off. If this happens while the motor is running, it is first decelerated to 0 Hz via the ramp set in P103.

The **STATUS** output is 1 if the output stage of the device is switched on; otherwise it is 0.

ERROR and **ERRORID** are reset if **ENABLE** is switched to 0.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
ENABLE	Enable	BOOL	**STATUS**	Motor supplied with power	BOOL
			ERROR	Error in FB	BOOL
			ERRORID	Error code	INT

ERRORID	**Explanation**
0	No error
1001h	Stop function is active
1300h	Device is not in the state “Ready to switch-on” or “Switch-on inhibit”

Example in IL:

```
CAL Power
CAL Move

LD TRUE
ST Power.Enable

(* Set 20 Hz (Max. 50 Hz) *)
LD DINT#20
MUL 16#4000
DIV 50

DINT_TO_INT
ST Move.Velocity

LD Power.Status
ST Move.Execute
```

Example in ST:

```
(* Enable power block *)
Power(Enable := TRUE);
IF Power.Status THEN
  (* The device is ready to switch-on *)
END_IF
```

3.3.3.10 MC_ReadActualPos

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	On+	X	X	X	

Continually delivers the actual current position of the frequency inverter if **ENABLE** is set to 1. As soon as there is a valid current position at the output, **VALID** is set to valid. In case of error, **ERROR** is set to 1 and in this case, **VALID** is 0.

Position scaling: 1 motor revolution = 1000

VAR_INPUT			VAR_OUTPUT		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
ENABLE	Enable	BOOL	**VALID**	Output is valid	BOOL
			ERROR	Error in FB	BOOL
			POSITION	Actual current position of the FI	DINT

Example in ST:

```
ReadActualPos(Enable := TRUE);
IF ReadActualPos.Valid THEN
  Pos := ReadActualPos.Position;
END_IF
```

3.3.3.11 MC_ReadParameter

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

Reads out a parameter cyclically from the device as long as **ENABLE** is set to 1. The read parameter is stored in Value and is valid if **DONE** is set to 1. For the duration of the reading process, the **BUSY** output is set to 1. If **ENABLE** remains 1, the parameter is constantly read out cyclically. The parameter number and index can be changed at any time when **ENABLE** is active. However, it is difficult to identify when the new value is read out, as the **DONE** signal remains 1 for the whole time. In this case, it is advisable to set the **ENABLE** signal to 0 for one cycle, as the **DONE** signal is then reset. The parameter index results from the index in the documentation minus 1. For example, P700 Index 3 (“Reason FI blocked”) is queried via parameter index 2. In case of error, **ERROR** is set to 1. In this case, **DONE** is 0 and the **ERRORID** contains the error code. If the **ENABLE** signal is set to 0, all signals and the **ERRORID** are deleted.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
ENABLE	Enable	BOOL	**DONE**	Value is valid	BOOL
PARAMETERNUMBER	Parameter number	INT	**ERROR**	Reading process failed	BOOL
PARAMETERINDEX	Parameter index	INT	**BUSY**	The process is not complete	BOOL
			ERRORID	Error code	INT
			VALUE	Read out parameter	DINT

ERRORID	**Explanation**
0	Invalid parameter number
3	Incorrect parameter index
4	No array
201	Invalid order element in the last order received
202	Internal response label cannot be depicted

Example in ST:

```
(* Motion module FB_ReadParameter *)
ReadParam(Enable := TRUE,Parameternumber := 102, ParameterIndex := 0);
IF ReadParam.Done THEN
  Value := ReadParam.Value;
  ReadParam(Enable := FALSE);
END_IF
```

3.3.3.12 MC_ReadStatus

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

Reads out the status of the device. The status machine is orientated to the PLCopen specification "Function blocks for motion control". The status is read out as long as **ENABLE** is set to 1.

VAR_INPUT			VAR_OUTPUT		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
ENABLE	Enable	BOOL	**VALID**	Output is valid	BOOL
			ERROR	Error in FB	BOOL
			ERRORSTOP	The device has an error	BOOL
			DISABLED	The output stage of the device is switched off	BOOL
			STOPPING	A Stop command is active	BOOL
			DISCRETEMOTION	One of the three positioning FBs is active	BOOL
			CONTINUOUSMOTION	The MC_Velocity is active	BOOL
			HOMING	The MC_Home is active	BOOL
			STANDSTILL	The device has no active Move command. It is at a standstill with 0 rpm and the output stage switched on.	BOOL

Example in ST:

```
ReadStatus(Enable := TRUE);
IF ReadStatus.Valid THEN
  fError := ReadStatus.ErrorStop;
  fDisable := ReadStatus.Disabled;
  fStopping := ReadStatus.Stopping;
  fInMotion := ReadStatus.DiscreteMotion;
  fInVelocity := ReadStatus.ContinuousMotion;
  fInHome := ReadStatus.Homing;
  fStandStill := ReadStatus.StandStill;
end_if
```

3.3.3.13 MC_Reset

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

Resets an error in the device (fault acknowledgement), on a rising flank from **EXECUTE**. In case of error, **ERROR** is set to 1 and the cause of the fault is entered in **ERRORID**. With a negative flank on **EXECUTE** all errors are reset.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
EXECUTE	Start	BOOL	**DONE**	Device error reset	BOOL
			ERROR	Error in FB	BOOL
			ERRORID	Error code	INT
			BUSY	Reset process is still active	BOOL

ERRORID	**Explanation**
0	No error
1001h	Stop function is active
1700h	An error reset could not be performed, because the cause of the error is still present.

Example in ST:

```
Reset(Execute := TRUE);
IF Reset.Done THEN
 (* The error has been reset *)
 Reset(Execute := FALSE);
ELSIF Reset.Error THEN
 (* Reset could not be executed, as the cause of the error is still present *)
 Reset(Execute := FALSE);
END_IF
```

3.3.3.14 MC_Stop

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

With a rising flank (0 to 1) the device is set to the state **STANDINGSTILL**. All motion functions which are active are cancelled. The device brakes to 0 Hz and switches off the output stage. As long as the Stop command is active **(EXECUTE** = 1), all other Motion FBs are blocked. The **BUSY** output becomes active with the rising flank on **EXECUTE** and remains active until there is a falling flank on **EXECUTE**.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
EXECUTE	Start	BOOL	**DONE**	Command has been executed	BOOL
			BUSY	Command is active	BOOL

3.3.3.15 MC_WriteParameter_16 / MC_WriteParameter_32

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

Writes a 16/32 Bit parameter into the device if **EXECUTE** changes from 0 to 1 (flank). The parameter has been written if **DONE** is set to 1. For the duration of the reading process, the **BUSY** output is set to 1. In case of error, **ERROR** is set to 1 and the **ERRORID** contains the error code. The signals **DONE, ERROR, ERRORID** remain set until **EXECUTE** changes back to 0. If the **EXECUTE** signal changes to 0, the writing process is not cancelled. Only the **DONE** signal remains set for 1 PLC cycle.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
EXECUTE	Enable	BOOL	**DONE**	Value is valid	BOOL
PARAMETERNUMBER	Parameter number	INT	**BUSY**	The writing process is active	BOOL
PARAMETERINDEX	Parameter index	INT	**ERROR**	Reading process failed	BOOL
VALUE	Value to be written	INT	**ERRORID**	Error code	INT
RAMONLY	Saves the value only in RAM (version V2.1 and higher)	BOOL			

ERRORID	**Explanation**
0	Invalid parameter number
1	Parameter value cannot be changed
2	Lower or upper value limit exceeded
3	Incorrect parameter index
4	No array
5	Invalid data type
6	Only resettable (only 0 may be written)
7	Description element cannot be changed
201	Invalid order element in the last order received
202	Internal response label cannot be depicted

Example in ST:

```
WriteParam16(Execute := TRUE, ParameterNumber := 102, ParameterIndex := 0, Value := 300);
  IF WriteParam16.Done THEN
    WriteParam16(Execute := FALSE);
  END_IF;
```

3.3.4 Standard

3.3.4.1 CTD downward counter

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

With a rising flank on **CD** the counter of the function block **CV** is reduced by one, as long as CV is greater than -32768. If **CV** is less than or equal to 0, the output **Q** remains TRUE. Via **LD** the counter **CV** can be set to the value saved in **PV**.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
CD	Counter input	**BOOL**	**Q**	TRUE, if CV <= 0	**BOOL**
LD	Load starting value	**BOOL**	**CV**	Actual counter reading	**INT**
PV	Starting value	**INT**			

Example in IL:

```
LD VarBOOL1
ST CTDInst.CD
LD VarBOOL2
ST CTDInst.LD
LD VarINT1
ST CTDInst.PV
CAL CTDInst
LD CTDInst.Q
ST VarBOOL3
LD CTDInst.CV
ST VarINT2
```

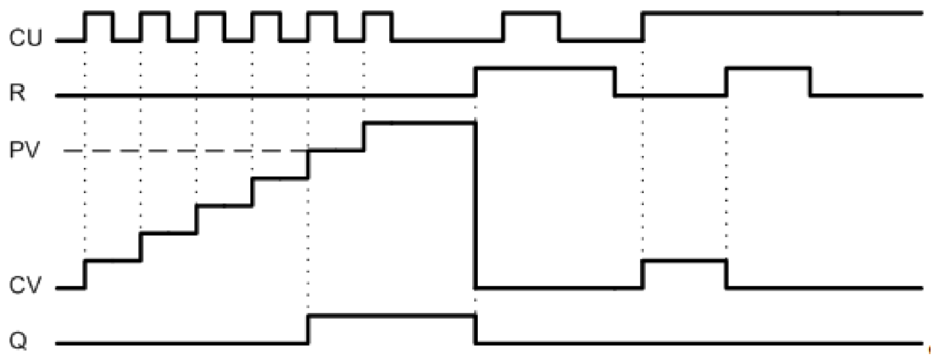
Example in ST:

```
CTDInst(CD := VarBOOL1, LD := VarBOOL2, PV := VarINT1);
VarBOOL3 := CTDInst.Q;
VarINT2  := CTDInst.CV;
```

3.3.4.2 CTU upward counter

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

With a rising flank on **CU**, the counter of the function block **CV** is increased by one. **CV** can be counted up to the value 32767. As long as **CV** is greater than or equal to **PV**, output **Q** remains TRUE. Via **R** the counter **CV** can be reset to zero.



VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
CU	Counter input	**BOOL**	**Q**	TRUE, if CV >= PV	**BOOL**
R	Reset: counter reading	**BOOL**	**CV**	Actual counter reading	**INT**
PV	Limit value	**INT**			

Example in IL:

```
LD VarBOOL1
ST CTUInst.CU
LD VarBOOL2
ST CTUInst.R
LD VarINT1
ST CTUInst.PV
CAL CTUInst(CU := VarBOOL1, R := VarBOOL2, PV := VarINT1)
LD CTUInst.Q
ST VarBOOL3
LD CTUInst.CV
ST VarINT2
```

Example in ST:

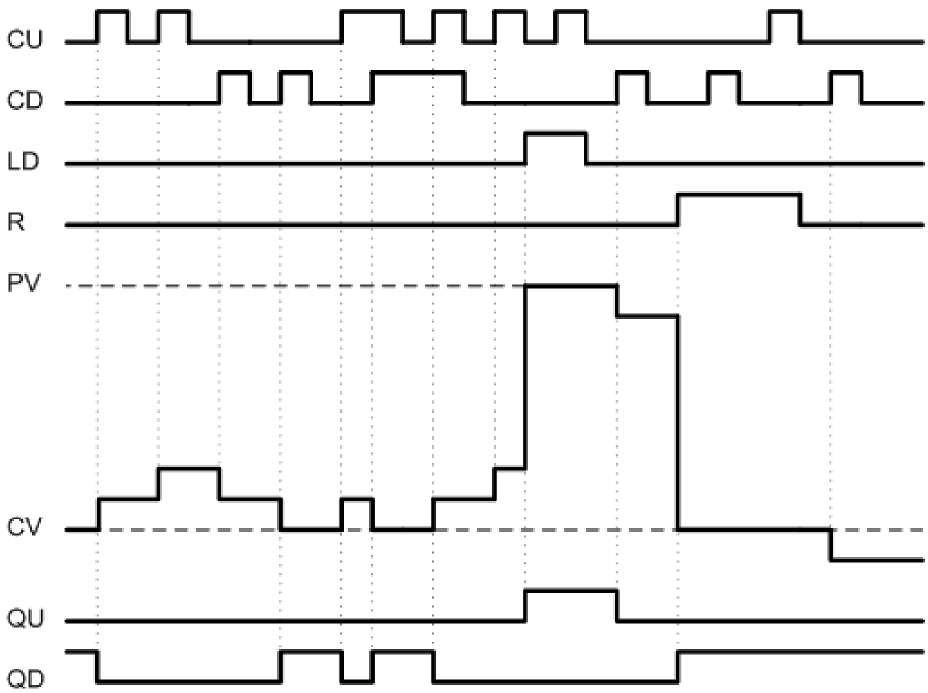
```
CTUInst(CU := VarBOOL1, R := VarBOOL2, PV := VarINT1);
VarBOOL3 := CTUInst.Q;
VarINT2 := CTUInst.CV;
```

3.3.4.3 CTUD upward and downward counter

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

With a rising flank on **CU** the counter CV is increased by one, as long as CV is less than 32767. With a rising flank on **CD** the counter of the function block **CV** is reduced by one, as long as **CV** is greater than -32768. Via **R** the counter **CV** can be set to zero. Via **LD** the value saved in **PV** is copied to **CV**.

R has priority over **LD**, **CU** and **CV**. **PV** can be changed at any time, **QU** always relates to the value which is currently set.



VAR_INPUT			VAR_OUTPUT		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
CU	Counting upwards	BOOL	**QU**	TRUE, if CV >= PV	BOOL
CD	Counting downwards	BOOL	**QD**	TRUE, if CV <= 0	BOOL
R	Reset: counter reading	BOOL	**CV**	Actual counter reading	INT
LD	Load starting value	BOOL			
PV	Starting value / Limit value	INT			

Example in IL:

```
LD VarBOOL1
ST CTUDInst.CU
LD VarBOOL3
ST CTUDInst.R
LD VarBool4
ST CTUDInst.LD
LD VarINT1
ST CTUInst.PV
CAL CTUDInst
LD CTUDInst.QU
ST VarBOOL5
LD CTUDInst.QD
ST VarBOOL5
LD CTUInst.CV
ST VarINT2
```

Example in ST:

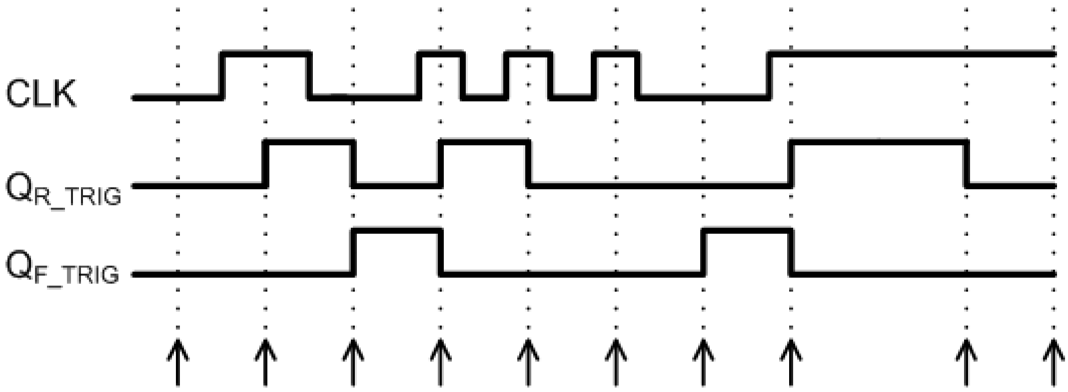
```
CTUDInst(CU:=VarBOOL1, R:=VarBOOL3, LD:=VarBOOL4, PV:=VarINT1);
VarBOOL5 := CTUDInst.QU;
VarBOOL5 := CTUDInst.QD;
VarINT2 := CTUDInst.CV;
```

3.3.4.4 R_TRIG and F_TRIG

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

Both functions are used for flank detection. If a flank is detected on **CLK, Q** is set to TRUE until the next function call-up, after which it is reset to FALSE. Only with a new flank can **Q** become TRUE again for a cycle.

- R_TRIG = rising flank
- F_TRIG = falling flank



Function call-ups

VAR_INPUT			VAR_OUTPUT		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
CLK	set	**BOOL**	**Q**	Output	**BOOL**

Example in IL:

```
LD VarBOOL1
ST RTRIGInst.CLK
CAL RTRIGInst
LD RTRIGInst.Q
ST VarBOOL2
```

Example in ST:

```
RTRIGInst(CLK:= VarBOOL1);
VarBOOL2 := RTRIGInst.Q;
```

Information

The output of the function only changes if the function is called up. Because of this it is advisable to continually call up the flank detection with the PLC cycle.

3.3.4.5 RS Flip Flop

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

Bi-stable function: via **S** the output **Q1** is set and via **R1** it is deleted again. If **R1** and **S** are both TRUE, **R1** is dominant.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
S	set	BOOL	**Q1**	Output	BOOL
R1	Reset	BOOL			

Example in IL:

```
LD VarBOOL1
ST RSInst.S
LD VarBOOL2
ST RSInst.R1
CAL RSInst
LD RSInst.Q1
ST VarBOOL3
```

Example in ST:

```
RSInst(S:= VarBOOL1 , R1:=VarBOOL2);
VarBOOL3 := RSInst.Q1;
```

3.3.4.6 SR Flip Flop

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

Bi-stable function; via **S1** the output **Q1** is set and via **R** it is deleted again. If **R1** and **S** are both TRUE, **S1** is dominant.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
S1	set	BOOL	**Q1**	Output	BOOL
R	Reset	BOOL			

Example in IL:

```
LD VarBOOL1
ST SRInst.S1
LD VarBOOL2
ST SRInst.R
CAL RSInst
LD SRInst.Q1
ST VarBOOL3
```

Example in ST:

```
SRInst(S1:= VarBOOL1 , R:=VarBOOL2);
VarBOOL3 := SRInst.Q1;
```

3.3.4.7 TOF switch-off delay

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

If **IN** = TRUE, then **Q** is set to TRUE. If **IN** changes to FALSE, the timer counts upwards. As long as the timer is running (**ET** < **PT)** **Q** remains set to TRUE. If (**ET** = **PT**) the timer stops, **Q** becomes FALSE. With a new rising flank on **IN**, the timer **ET** is reset to zero.

Here, literals can be used for simplified input, e.g.

- LD TIME#50s20ms = 50.020 seconds
- LD TIME#1d30m = 1 day and 30 minutes

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
IN	Timer active	BOOL	**Q**	TRUE ß (ET<PT)	BOOL
PT	Duration	DINT	**ET**	Current timer reading	DINT

Example in IL:

```
LD VarBOOL1
ST TOFInst.IN
LD DINT#5000
ST TOFInst.PT
CAL TOFInst
LD TOFInst.Q
ST VarBOOL2
```

Example in ST:

```
TOFInst(IN := VarBOOL1, PT:= T#5s);
VarBOOL2 := TOFInst.Q;
```

Information **Timer ET**

The time ET runs independently of a PLC cycle. Starting of the timer with IN and setting of the output Q are only executed with the function call-up "CAL". The function call-up takes place within a PLC cycle. However, with PLC programs which are longer than 5 ms this may result in the occurrence of jitter.

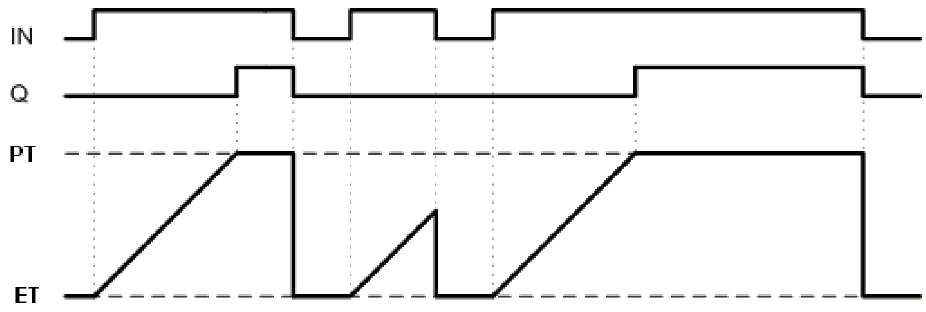
3.3.4.8 TON switch-on delay

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

If **IN** = TRUE is set, the timer counts upwards. If **ET** = PT, **Q** is set to TRUE and the timer stops. **Q** remains TRUE for as long as **IN** is also TRUE. With a new rising flank on **IN** the counter starts again from zero. **PT** can be changed while the timer is running. The time period in **PT** is entered in milliseconds. This enables a time delay between 5ms and 24.8 days. As the time base of the PLC is 5ms, the minimum time delay is also 5ms.

Here, literals can be used for simplified input, e.g.

- LD TIME#50s20ms = 50.020 seconds
- LD TIME#1d30m = 1 day and 30 minutes



VAR_INPUT			VAR_OUTPUT		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
IN	Timer active	BOOL	**Q**	TRUE ß (IN=TRUE & ET=PT)	BOOL
PT	Duration	DINT	**ET**	Current timer reading	DINT

Example in IL:

```
LD VarBOOL1
ST TONInst.IN
LD DINT#5000
ST TONInst.PT
CAL TONInst
LD TONInst.Q
ST VarBOOL2
```

Example in ST:

```
TONInst(IN := VarBOOL1, PT:= T#5s);
VarBOOL2 := TONInst.Q;
```

Information **Timer ET**

The time ET runs independently of a PLC cycle. Starting of the timer with IN and setting of the output Q are only executed with the function call-up "CAL". The function call-up takes place within a PLC cycle. However, with PLC programs which are longer than 5 ms this may result in the occurrence of jitter.

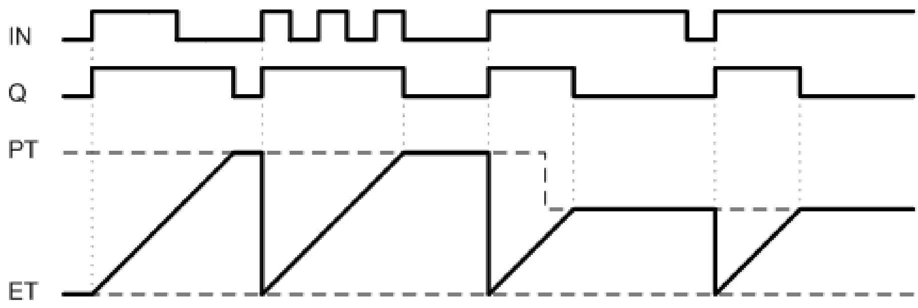
3.3.4.9 TP time pulse

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

With a positive flank on **IN** the timer is started with the value 0. The timer runs up to the value which is entered **PT** and then stops. This process cannot be interrupted! PT can be changed during counting. The output **Q** is TRUE, as long as the timer **ET** is less than **PT**. If **ET = PT** and a rising flank is detected on **IN**, the timer is started again at 0.

Here, literals can be used for simplified input, e.g.

- LD TIME#50s20ms = 50.020 seconds
- LD TIME#1d30m = 1 day and 30 minutes



VAR_INPUT			VAR_OUTPUT		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
IN	Timer active	BOOL	**Q**	TRUE ß (ET<PT)	BOOL
PT	Duration	DINT	**ET**	Current timer reading	DINT

Example in IL:

```
LD VarBOOL1
ST TPInst.IN
LD DINT#5000
ST TPInst.PT
CAL TPInst
LD TPInst.Q
ST VarBOOL2
```

Example in ST:

```
TPInst(IN := VarBOOL1, PT:= T#5s);
VarBOOL2 := TPInst.Q;
```

Information **Timer ET**

The time ET runs independently of a PLC cycle. Starting of the timer with IN and setting of the output Q are only executed with the function call-up "CAL". The function call-up takes place within a PLC cycle. However, with PLC programs which are longer than 5 ms this may result in the occurrence of jitter.

3.3.5 Access to memory areas of the frequency inverter

If the intermediate saving of large quantities of data, its transmission to or reception from other devices is necessary, the modules FB_WriteTrace and FB_ReadTrace should be used.

3.3.5.1 FB_ReadTrace

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X		

The memory areas of the FI can be read out directly with the aid of this FB.

If the FB detects a positive flank on **ENABLE**, all parameters which are present on the input are adopted. The memory address which is to be read out is indicated with **STARTINDEX** and **MEMORY**. If the reading process is successful, the **VALID** output changes to 1 and the value which has been read out is in **VALUE**.

If the FB is now called up several times and the **ENABLE** input remains at 1, with each call up the memory address which is to be read out is increased by 1 and the content of the new memory address is immediately copied to the output **VALUE**.

The current memory index for the next access can be read out under the output **ACTINDEX**. If the end of the memory has been reached, the **READY** changes to 1 and the reading process is stopped.

Values can be read in INT or DINT format. For INT values, only the Low component is evaluated by the **VALUE** output. Allocation is carried out via the **SIZE** input; a 0 stands for INT and a 1 for DINT values.

Allocation of the memory areas is carried out via the MEMORY input:

MEMORY = 1 to P613[0-251]	corresponds to 504 INT or 252 DINT values
MEMORY = 0 to P900[0-247] up to P906[0-111]	corresponds to 3200 INT or 1600 DINT values

The FB cannot be interrupted by other blocks

With a negative flank on ENABLE, all outputs are set to 0 and the function of the FB is terminated.

VAR_INPUT			VAR_OUTPUT		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
ENABLE	Execute	BOOL	**VALID**	Reading process successful	BOOL
SIZE	Memory format	BOOL	**READY**	The entire memory has been read out	BOOL
MEMORY	Selection of memory area	BYTE	**ERROR**	the FB has an error	BOOL
STARTINDEX	Indicates the memory cell to be written to	INT	**ERRORID**	Error code	INT
			ACTINDEX	Actual memory index, to which will be read in the next cycle	INT
			VALUE	Value read out	DINT

ERRORID	Explanation
0	No error
1A00h	STARTINDEX value range exceeded
1A01h	MEMORY value range exceeded

3.3.5.2 FB_WriteTrace

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X		

Via this FB, individual values or large numbers of values can be intermediately saved in the FI. The values are not permanently saved, i.e. the values are lost if the FI is restarted.

If the FB detects a positive flank on **ENABLE**, all parameters with are present on the input are adopted. The value in VALUE is written to the storage address indicated in **STARTINDEX** and **MEMORY**. If the writing process is successful, the VALID output changes to 1.

If the FB is now called up several times and the **ENABLE** input remains at 1, then with each call up of the FB the input **VALUE** is read and saved and the memory address is increased by 1. The current memory index for the next access can be read out under the output **ACTINDEX**. If the end of the memory is reached, the output FULL changes to 1 and the saving process is stopped. However, if the input **OVERWRITE** is set to 1, the memory index is reset to the **STARTINDEX** and the values which have been previously written are overwritten.

Values can be saved in INT or DINT format. For INT values, only the Low component is evaluated by the **VALUE** input. Allocation is carried out via the **SIZE** input; a 0 stands for INT and a 1 for DINT values.

Allocation of the memory areas is carried out via the MEMORY input:

MEMORY = 1 to P613[0-251]	corresponds to 504 INT or 252 DINT values
MEMORY = 0 to P900[0-247] up to P906[0-111]	corresponds to 3200 INT or 1600 DINT values

The FB cannot be interrupted by other blocks

With a negative flank on **ENABLE**, all outputs are set to 0 and the function of the FB is terminated.

VAR_INPUT			VAR_OUTPUT		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
ENABLE	Execute	BOOL	**VALID**	Writing process successful	BOOL
SIZE	Memory format	BOOL	**FULL**	Entire memory is full	BOOL
OVERWRITE	Memory can be overwritten	BOOL	**ERROR**	the FB has an error	BOOL
MEMORY	Selection of memory area	BYTE	**ERRORID**	Error code	INT
STARTINDEX	Indicates the memory cell to be written to	INT	**ACTINDEX**	Actual memory index, to which saving will be carried out in the next cycle	DINT
VALUE	Value to be saved	DINT			

ERRORID	Explanation
0	No error
1A00h	STARTINDEX value range exceeded
1A01h	MEMORY value range exceeded

Information

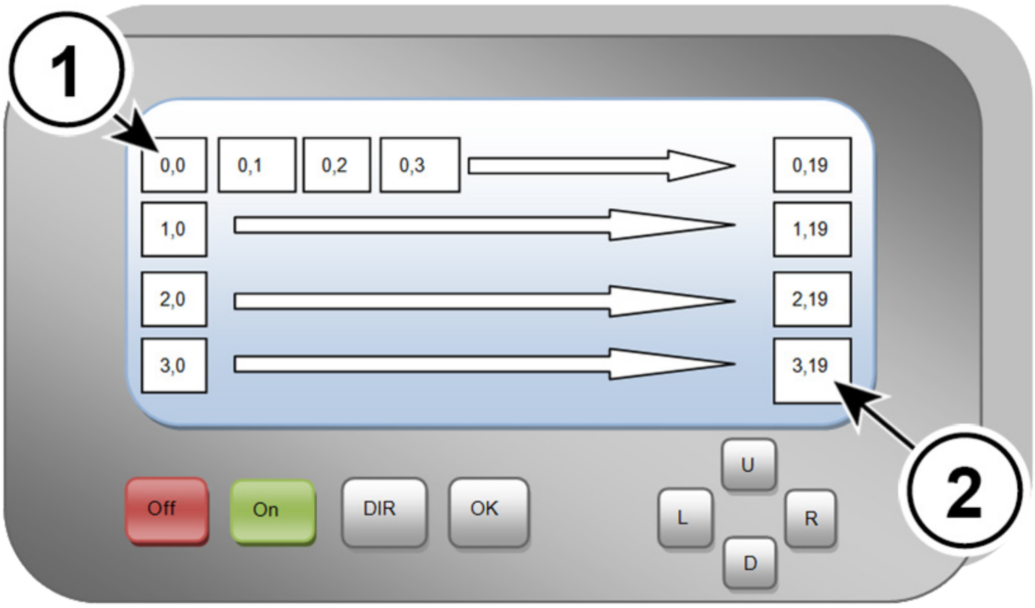
Please note: The memory area in the setting MEMORY = 0 is also used by the Scope function. Use of the Scope function overwrites the saved values!

3.3.6 Visualisation with ParameterBox

In the ParameterBox, the entire display can be used for the display of information. For this, the ParameterBox must be switched to visualisation mode. This is possible with the ParameterBox (Parameter P1308) firmware version V4.3 or higher, and is carried out as follows:

- In the menu item "Display", set the parameter P1003 to "PLC Display"
- Switch to the operating value display with the left or right arrow key
- PLC display is now enabled in the ParameterBox and remains permanently enabled.

In the visualisation mode of the ParameterBox, the content of the display can be set with the two FBs described below. However, before the item "Allow ParameterBox function modules" must be activated in the PLC configuration dialogue (Button ). With the process value "Parameterbox_key_state", the keyboard status of the box can also be queried. With this, input into the PLC program can be implemented. The display structure and the keys to be read out for the ParameterBox can be seen in the figure below.



1	First character	(0,0 → row = 0 , column = 0)
2	Last character	(3,19 → row = 3 , column = 19)

3.3.6.1 Overview visualisation

Function module	Description
FB_STRINGToPBox	Copies a string into the P-Box
FB_DINTToPBox	Copies a DINT value to the P-Box

3.3.6.2 FB_DINTToPBOX

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

This function module converts a DINT value into an ASCII string and copies this into the ParameterBox. The output can be in decimal, binary or hexadecimal format; the selection is performed via **MODE**. Via **ROW** and **COLUMN** the starting point of the string is set in the ParameterBox display. The parameter **LENGTH** transfers the length of the string in characters. In decimal **MODE** the parameter **POINT** positions a decimal point in the number which is to be displayed. In **POINT** it is stated how many characters are to the right of the decimal point. With the setting 0 the **POINT** function is disabled. If the number contains more characters than the length allows and no decimal point is set, the overflow is indicated by the character "#". If there is a decimal point in the number, all numbers behind the decimal point may be omitted if required. In hexadecimal and binary MODE the lowest value bits are displayed if the set length is too short. As long as **ENABLE** is set to 1, all changes to the inputs are adopted immediately. If **VALID** changes to 1, the string has been correctly transferred. In case of error, **ERROR** is set to 1. In this case, **VALID** is 0. The corresponding error code is then valid in **ERRORID**. With a negative flank on **ENABLE**, **VALID**, **ERROR** and **ERRORID** are reset.

Examples:

Setting	Number to be displayed	P-Box display
Length = 5 Point = 0	12345	12345
Length = 5 Point = 0	-12345	#####
Length = 10 Point = 3	123456789	123456.789
Length = 8 Point = 3	123456789	123456.7

VAR_INPUT			VAR_OUTPUT		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
ENABLE	Transfer of the string	BOOL	**VALID**	String transferred	BOOL
MODE	Display format 0 = Decimal 1 = Binary 2 = Hexadecimal Value range = 0 to 2	BYTE	**ERROR**	Error in FB	BOOL
ROW	Line of the display Value range = 0 to 3	BYTE	**ERRORID**	Error code	INT
COLUMN	Column of the display Value range = 0 to 19	BYTE			
POINT	Position of decimal point Value range = 0 to 10 0 = Function is disabled	BYTE			
LENGTH	Output length Value range = 1 to 11	BYTE			
VALUE	Number to be output	DINT			

ERRORID	Explanation
0	No error
1500h	String overwrites the memory area of the P-Box array
1501h	Value range exceeded at LINE input
1502h	Value range exceeded at ROW input
1504h	Value range exceeded at POINT input
1505h	Value range exceeded at LENGTH input
1506h	Value range exceeded at MODE input

Example in ST:

```
(* Initialisation *)
if FirstTime then
  StringToPBox.ROW := 1;
  StringToPBox.Column := 16;
  FirstTime := False;
end_if;

(* Query actual position *)
ActPos(Enable := TRUE);
if ActPos.Valid then
  (* Display position in the PBox displays (PBox P1003 = PLC display ) *)
  DintToPBox.Value := ActPos.Position;
  DintToPBox.Column := 9;
  DintToPBox.LENGTH := 10;
  DintToPBox(Enable := True);
end_if;

(* Switch device on or off via DIG1 *)
Power(Enable := _5_State_digital_input.0);
if OldState <> Power.Status then
  OldState := Power.Status;
  (* Is device switched on? *)
  if Power.Status then
    StringToPBox(Enable := False, Text := TextOn);
  else
    StringToPBox(Enable := False, Text := TextOff);
  end_if;

  StringToPBox(Enable := TRUE);
else
  StringToPBox;
end_if;
```

3.3.6.3 FB_STRINGToPBOX

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

This function module copies a string (chain of characters) into the memory array of the ParameterBox. Via **ROW** and **COLUMN** the starting point of the string is set in the ParameterBox display. The parameter **TEXT** transfers the required string to the function module; the name of the string can be obtained from the table of variables. As long as **ENABLE** is set to 1, all changes to the inputs are adopted immediately. If the **CLEAR** input is set, the entire display content is overwritten with space characters before the selected string is written. If **VALID** changes to 1, the string has been correctly transferred. In case of error, **ERROR** is set to 1. In this case, **VALID** is 0. The corresponding error code is then valid in **ERRORID**. With a negative flank on **ENABLE**, **VALID**, **ERROR** and **ERRORID** are reset.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
ENABLE	Transfer of the string	BOOL	**VALID**	String transferred	BOOL
CLEAR	Clear display	BOOL	**ERROR**	Error in FB	BOOL
ROW	Line of the display Value range = 0 to 3	BYTE	**ERRORID**	Error code	INT
COLUMN	Column of the display Value range = 0 to 19	BYTE			
TEXT	Text to be displayed	STRING			

ERRORID	**Explanation**
0	No error
1500h	String overwrites the memory area of the P-Box array
1501h	Value range exceeded at ROW input
1502h	Value range exceeded at COLUMN input
1503h	The selected string number does not exist
1506h	The option "Allow ParameterBox function modules" is not activated in the PLC configuration.

Example in ST:

```
(* Initialisation *)
if FirstTime then
  StringToPBox.ROW := 1;
  StringToPBox.Column := 16;
  FirstTime := False;
end_if;

(* Query actual position *)
ActPos(Enable := TRUE);
if ActPos.Valid then
  (* Display position in the PBox displays (PBox P1003 = PLC display ) *)
  DintToPBox.Value := ActPos.Position;
  DintToPBox.Column := 9;
  DintToPBox.LENGTH := 10;
  DintToPBox(Enable := True);
end_if;

(* Switch device on or off via DIG1 *)
Power(Enable := _5_State_digital_input.0);
if OldState <> Power.Status then
  OldState := Power.Status;
  (* Is device switched on? *)
  if Power.Status then
    StringToPBox(Enable := False, Text := TextOn);
  else
    StringToPBox(Enable := False, Text := TextOff);
  end_if;

  StringToPBox(Enable := TRUE);
else
  StringToPBox;
end_if;
```

3.3.7 FB_Capture (Detection of rapid events)

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X		

The cycle time of the PLC is 5ms. This cycle may be too long to detect very rapid external events. Via FB Capture it is possible to capture certain physical values on flanks at the FI inputs. Monitoring of the inputs is carried out in a 1ms cycle. The values which are saved can be read by the PLC later.

With a positive flank on **EXECUTE** all inputs are read in and the Capture function is enabled. The FI input which is to be monitored is selected via the **INPUT** input. Via **EDGE**, the type of flank and the behaviour of the module are selected.

EDGE = 0 With the first positive flank, the selected value is saved under **OUTPUT1** and **DONE1** is set to 1. The next positive flank saves under **OUTPUT2** and **DONE2** is set to 1. The FB is then disabled.

EDGE = 1 Behaviour as for **EDGE** = 0, with the difference that triggering is with the negative flank.

EDGE = 2 With the first positive flank, the selected value is saved under **OUTPUT1** and **DONE1** is set to 1. The next negative flank saves under **OUTPUT2** and **DONE2** is set to 1. The FB is then disabled.

EDGE = 3 Behaviour as for **EDGE** = 2, with the difference that triggering is first with the negative and then with the positive flank.

If the input **CONTINUOUS** is set to 1, then only the settings 0 and 1 are relevant to **EDGE**. The FB continues to run and always saves the last triggering event under **OUTPUT1**. **DONE1** remains active from the first event. **DONE2** and **OUTPUT2** are not used.

The **BUSY** output remains active until both Capture events (**DONE1** and **DONE2**) have occurred.

The function of the module can be terminated at any time with a negative flank on **EXECUTE**. All outputs retain their values. With a positive flank on **EXECUTE** first, all outputs are deleted and then the function of the module is started.

VAR_INPUT			VAR_OUTPUT		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
EXECUTE	Execute	BOOL	**DONE1**	Value in OUTPUT1 valid	BOOL
CONTINUOUS	Single execution or continuous operation	BOOL	**DONE2**	Value in OUT valid	BOOL
INPUT	**SK54xE** Input to be monitored 0 = Input 1 - - - - 7 = Input 8 **SK52xE, SK53xE, SK2xxE, SK2xx-EFDS** Input to be monitored 0 = Input 1 - - - - 3 = Input 4	BYTE	**BUSY**	FB still waiting for a Capture event	BOOL
EDGE	Triggering flank	BYTE	**ERROR**	the FB has an error	BOOL
SOURCE	Value to be saved 0 = Position in rotations 1 = Actual frequency 2 = Torque	BYTE	**ERRORID**	Error code	INT
			OUTPUT1	Value for 1st Capture event	DINT
			OUTPUT2	Value for 2nd Capture event	DINT

ERRORID	Explanation
0	No error
1900h	INPUT value range exceeded
1901h	EDGE value range exceeded
1902h	SOURCE value range exceeded
1903h	More than two instances are active

Example in ST:

```
Power(ENABLE := TRUE);
IF Power.STATUS THEN
    Move(EXECUTE := TRUE, POSITION := Pos, VELOCITY := 16#2000);
    (* The FB waits for a High signal on DIG1. When this is detected, the FB saves the actual
    position. The value can be queried with the property "OUTPUT1". *)
    Capture(EXECUTE := TRUE, INPUT := 0);

    IF Capture.DONE1 THEN
      Pos := Capture.OUTPUT1;
      Move(EXECUTE := FALSE);
    END_IF;
END_IF;
```

Information

Several instances of this FB may exist in the PLC program. However, only two instances may be active at the same time!

3.3.8 FB_DinCounter

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	V1.1 and above	

This FB is used to count pulses via the digital inputs. All flanks (Low - High and High - Low) are counted. The minimum pulse width is 1 ms.

The FB is enabled via ENABLE. With the positive flanks, the inputs PV, UD, DIN and MODE are adopted and all outputs are deleted.

UD defines the counting direction

- 0 = larger numbers
- 1 = smaller numbers

A counter value can be entered at PV. Depending on the setting of the **MODE** input this has different effects.

MODE

- 0 = Overflow, the counter is operated as a continuous counter. It can overflow in both positive and negative directions. When the function is started, CV = PV is set. In this Mode BUSY remains always 1 and Q always 0.
- 1 = without overflow
  - Counting forwards to CV starts at 0, BUSY = 1, and runs until CV=>PV. Then BUSY changes to 0 and Q to 1. The counting process stops.
  - Counting backwards to CV starts at PV and runs until CV<= 0. During this time is BUSY = 1 and changes to 0 when the end of the count is reached. In return, Q changes to 1.
  - The restart of the counter is reached at the ENABLE input via a new flank.

DIN defines the measuring input. The number of inputs depends on the respective FI (max. 4).

- Input 1 = 0
- Input 2 = 1
- Input 3 = 2
- Input 4 = 3

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
ENABLE	Enable	BOOL	**Q**	Counting completed	BOOL
UD	Counting direction 0 = larger numbers 1 = smaller numbers	BOOL	**BUSY**	Counter runs	BOOL
PV	Counter value	INT	**ERROR**	the FB has an error	BOOL
MODE	Mode	BYTE	**ERRORID**	Error code	INT
DIN	Measuring input	BYTE	**CV**	Counter value	INT
			CF	Counting frequency (resolution of 0.1) 1)	INT

ERRORID	**Explanation**
0	No error
0x1E00	Digital input is already used by other counter
0x1E01	Digital input does not exist
0x1E02	MODE value range exceeded

1) Measuring range 0,1 Hz to 1 kHz

3.3.9 FB_FunctionCurve

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	

This function module produces a mapping control. Defined points can be communicated to the function block, with which it emulates a function. The output then behaves according to the saved map. Linear interpolation is carried out between the individual base points. The base point are defined with X and Y values. The X values are always of the **INT** type, the Y values can either be of the **INT** or the **DINT** type, depending on the size of the largest base point. More memory is required if **DINT** is used. The base points are entered in the column "Init Value" in the variables window. If TRUE is detected at the **ENABLE** input, on the basis of the input value **INVALUE** the corresponding output value **OUTVALUE** is calculated. **VALID** = TRUE indicates that the output value **OUTVALUE** is valid. As long as **VALID** is FALSE, the output **OUTVALUE** has the value 0. If the input value **INVALUE** exceeds the upper or the lower end of the characteristic range, the first or the last output value of the characteristic range remain until the **INVALUE** returns to within the area of the characteristic range. If the characteristic range is exceeded or undershot, the appropriate output **MINLIMIT** or **MAXLIMIT** is set to TRUE. **ERROR** becomes TRUE, if the abscissa values (X values) of the characteristic range do not continuously increase or if no table is initialised. The appropriate error is output by **ERRORID** and the starting value is 0. The error is reset if **ENABLE** = FALSE.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
ENABLE	Execute	BOOL	**VALID**	Output value is valid	BOOL
INVALUE	Input value (x)	INT	**ERROR**	Error in FB	BOOL
			ERRORID	Error code	INT
			MAXLIMIT	Maximum limit reached	BOOL
			MINLIMIT	Minimum limit reached	BOOL
			OUTVALUE	Output value (y)	DINT

ERRORID	**Explanation**
0	No error
1400h	Abscissa values (X values) of the characteristic range do not always increase
1401h	No map initialised

3.3.10 FB_PIDT1

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

The P-I-DT1 is a freely parameterisable individual controller. If individual components or the P, I or DT1 component are not required, their parameters are written as 0. The T1 component only functions together with the D component. Therefore a PT1 controller cannot be parameterised. Due to internal memory limitations, the control parameters are restricted to the following areas:

Permissible value range for control parameters			
Parameters	**Value range**	**Scaling**	**Resulting value range**
P (Kp)	0 – 32767	1/100	0.00 – 327.67
I (Ki)	0 – 10240	1/100	0.00 – 102.40
D (Kd)	0 – 32767	1/1000	0.000 – 32.767
T1 (ms)	0 – 32767	1/1000	0.000 – 32.767
Max.	-32768 – 32767		
Min.	-32768 – 32767		

The controller starts to calculate when **ENABLE** is set to TRUE. The control parameters are only adopted with a rising flank from**ENABLE**. While **ENABLE** is TRUE, changes to the control parameters have no effect. If **ENABLE** is set to FALSE, the output remains at its last value.

The output bit **VALID** is set, as long as the output value of Q is within the Min and Max limits and the **ENABLE** input is TRUE.

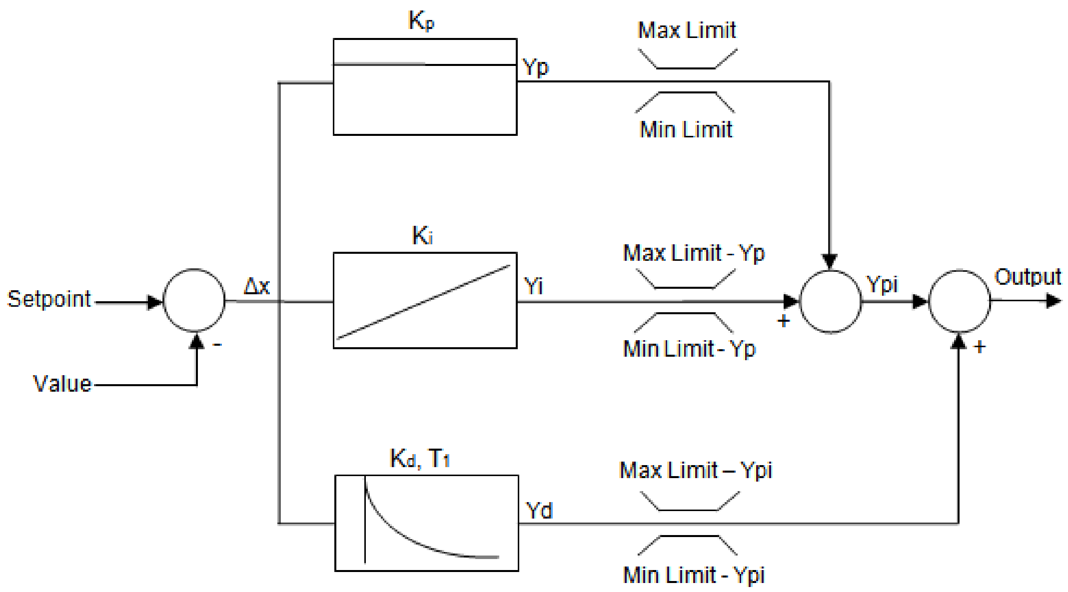
ERROR is set as soon as an error occurs. The **VALID** bit is then FALSE and the cause of the fault can be identified from the **ERRORID** (see table below).

If the **RESET** bit is set to TRUE, the content of the integrator and the differentiator are set to 0. If the **ENABLE** input is FALSE, the **OUTPUT** output is also set to 0. If the **ENABLE** input is set to TRUE, only the P component has an effect on the **OUTPUT** output.

If the output value **OUTPUT** is outside of the range of the maximum or minimum output values, the corresponding bit **MAXLIMIT** or **MINLIMIT** is set and the **VALID** bit is set to FALSE.

Information

If the entire program cannot be executed within a PLC cycle, the controller calculates the output value a second time with the old scanning values. This ensures a constant scanning rate. Because of this it is essential that the CAL command for the PIDT1 controller is executed in each PLC cycle and only at the end of the PLC program.



<table>
<tr><th colspan="3">VAR_INPUT</th><th colspan="3">VAR_OUTPUT</th></tr>
<tr><th>Input</th><th>Explanation</th><th>Type</th><th>Output</th><th>Explanation</th><th>Type</th></tr>
<tr><td>ENABLE</td><td>Execute</td><td>BOOL</td><td>VALID</td><td>Output value is valid</td><td>BOOL</td></tr>
<tr><td>RESET</td><td>Reset outputs</td><td>BOOL</td><td>ERROR</td><td>Error in FB</td><td>BOOL</td></tr>
<tr><td>P</td><td>P component (Kp)</td><td>INT</td><td>ERRORID</td><td>Error code</td><td>INT</td></tr>
<tr><td>I</td><td>I component (Ki)</td><td>INT</td><td>MAXLIMIT</td><td>Maximum limit reached</td><td>BOOL</td></tr>
<tr><td>D</td><td>D component (Kd)</td><td>INT</td><td>MINLIMIT</td><td>Minimum limit reached</td><td>BOOL</td></tr>
<tr><td>T1</td><td>T1 component in ms</td><td>INT</td><td>OUTPUT</td><td>Output value</td><td>INT</td></tr>
<tr><td>MAX</td><td>Maximum output value</td><td>INT</td><td></td><td></td><td></td></tr>
<tr><td>MIN</td><td>Minimum output value</td><td>INT</td><td></td><td></td><td></td></tr>
<tr><td>SETPOINT</td><td>Setpoint</td><td>INT</td><td></td><td></td><td></td></tr>
<tr><td>VALUE</td><td>Actual value</td><td>INT</td><td></td><td></td><td></td></tr>
<tr><th>ERRORID</th><th colspan="5">Explanation</th></tr>
<tr><td>0</td><td colspan="5">No error</td></tr>
<tr><td>1600h</td><td colspan="5">P component not within value range</td></tr>
<tr><td>1601h</td><td colspan="5">I component not within value range</td></tr>
<tr><td>1602h</td><td colspan="5">D component not within value range</td></tr>
<tr><td>1603h</td><td colspan="5">T1 component not within value range</td></tr>
</table>

3.3.11 FB_ResetPostion

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	V2.3 and above	V3.1 and above	On+	V2.1 and above	X	V1.2 and above	

With a flank on the **EXECUTE** input, the current position (P601) is set to the value entered in Position. If a position offset is entered in parameter P609, this offset is subtracted from the position.

With absolute encoders the current position can only be reset to 0. The value is not used in the position.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
EXECUTE	Execute	BOOL			
Position	Position	DINT			

3.3.12 FB_Weigh

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	V2.3 and above	V3.1 and above	X	V2.1 and above	X	V1.2 and above	

This module is used to determine the average torque during movement at a constant speed. From this value, physical values, such as the weight which is being moved can be determined.

The FB is started via a positive flank on the **EXECUTE** input. With the flank, all inputs are adopted by the FB. The FI moves with the speed which is set in **SPEED**. The measurement is started after the elapse of the time which is set in **STARTTIME**. The duration of the measurement is defined under **MEASURETIME**. The FI stops after the elapse of the measurement time. If the input **REVERSE** = 1, the measurement process starts again, but with a negative speed. Otherwise the measurement is complete, the output **DONE** changes to 1 and the measurement result is in VALUE.

As long as the measurement process is running, **BUSY** is active.

The scaling of the measurement result **VALUE** is 1 = 0.01% of the rated torque of the motor.

Call-up of another Motion FB stops the measurement function and the output **ABORT** changes to 1.

All outputs of the FB are reset with a new positive flank on **EXECUTE**.

VAR_INPUT			**VAR_OUTPUT**		
Input	**Explanation**	**Type**	**Output**	**Explanation**	**Type**
EXECUTE	Execute	BOOL	**DONE**	Measurement ended	BOOL
REVERSE	Change of rotation direction	BOOL	**BUSY**	Measurement running	BOOL
STARTTIME	Time to start of measurement in ms	INT	**ABORT**	Measurement aborted	BOOL
MEASURETIME	Measurement time in ms	INT	**ERROR**	the FB has an error	BOOL
SPEED	Measuring speed in % (standardised to the maximum frequency, 16#4000 corresponds to 100%)	INT	**ERRORID**	Error code	INT
			VALUE	Measurement result	INT

ERRORID	**Explanation**
0	No error
0x1000	FI not switched on
0x1101	Setpoint frequency not parameterised as a setpoint (P553)
0x1C00	STARTTIME value range exceeded
0x1C01	MEASURETIME value range exceeded
0x1C02	The tolerance of the measurement values with respect to each other is greater than 1/8

Example in ST:

```
(* Enable device *)
Power(Enable := TRUE);
(* Is the device enabled? *)
if Power.Status then
  (* Specify starting time 2000ms *)
  Weigh.STARTTIME := 2000;
  (* Specify measuring time 1000ms *)
  Weigh.MEASURETIME := 1000;
  (* Specify speed 25% of maximum speed *)
  Weigh.SPEED := 16#1000;
end_if;

Weigh(EXECUTE := Power.Status);
(* Was weighing completed? *)
if Weigh.done then
  Value := Weigh.Value;
end_if;
```

Information

Only one instance of this FB is permissible in the PLC program!

3.4 Operators

3.4.1 Arithmetical operators

Information

Some of the following operators may also contain further commands. These must be placed in brackets behind the operator. It must be noted that a space must be included behind the opened bracket. The closing bracket must be placed on a separate line of the program.

```
LD Var1
ADD( Var2
SUB Var3
)
```

3.4.1.1 ABS

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	ÎNT	DINT
Data type			X	X

Forms the absolute value from the accumulator

Example in IL:

```
LD -10 (* Loads the value -10 *)
ABS (* Accumulator = 10 *)
ST Value1 (* Saves the value 10 in Value1 ab *)
```

Example in ST:

```
Value1 := ABS(-10); (* The result is 10 *)
```

3.4.1.2 ADD and ADD(

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	ÎNT	DINT
Data type		X	X	X

Adds variables and constants together with the correct prefixes. The first value for addition is in the AE/accumulator, the second is loaded with the ADD command or is inside the bracket. Several variables or constants can be added to the ADD command. For bracket addition, the accumulator is added to the result of the expression in brackets. Up to 6 bracket levels are possible. The values to be added must belong to the same type of variable.

Example in IL:

```
LD 10
ADD 204 (* Addition of two constants *)
ST Value
LD 170 (* Addition of a constant and 2 variables. *)
ADD Var1, Var2 (* 170dez + Var1 + Var2 *)
ST Value
LD Var1
ADD( Var2
SUB Var3 (* Var1 + ( Var2 - Var3 ) *)
)
ST Value
```

Example in ST:

```
Ergebnis := 10 + 30; (* The result is 40 *)
Ergebnis := 10 + Var1 + Var2;
```

3.4.1.3 DIV and DIV(

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	ÎNT	DINT
Data type		X	X	X

Divides the accumulator by the operands For divisions by zero, the maximum possible result is entered into the accumulator, e.g. for a division with INT values, this is the value 0x7FFF or the value 0x8000 if the divisor is negative. For bracket division, the accumulator is divided by the result of the expression in brackets. Up to 6 bracket levels are possible. The values to be divided must belong to the same type of variable.

Example in IL:

```
LD 10
DIV 3 (* Division of two constants *)
ST iValue (* The result is 9 *)
LD 170 (* Division of a constant and 2 variables. *)
DIV Var1, Var2 (* (170dez : Var1) : Var2 *)
ST Value
LD Var1 (* Divide Var1 by the content of the brackets *)
DIV( Var2
SUB Var3
) (* Var1 : ( Var2 - Var3 ) *)
ST Value
```

Example in ST:

```
Ergebnis := 30 / 10; (* The result is 3 *)
Ergebnis := 30 / Var1 / Var2;
```

3.4.1.4 LIMIT

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	ÎNT	DINT
Data type		X	X	X

The command limits the value in the accumulator to the transferred minimum and maximum values. Values If this is exceeded, the maximum value is entered in the accumulator and if it is undershot, the minimum value is entered. If the value lies between the limits, there is no effect.

Example in IL:

```
LD 10 (* Loads the value 10 into the accumulator *)
LIMIT 20, 30 (* The value is compared with the limits 20 and 30. *)
(* The value in the accumulator is smaller, the accumulator is overwritten with 20 *)
ST iValue (* Saves the value 20 in Value1 *)
```

Example in ST:

```
Ergebnis := Limit(10, 20, 30); (* The result is 20 *)
```

3.4.1.5 MAX

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	ÎNT	DINT
Data type		X	X	X

This value determines the maximum value of two variables or constants For this, the current value of the accumulator is compared with the value transferred in the MAX command. After the command, the larger of the two values is in the accumulator. Both values must belong to the same type of variable.

Example in IL:

```
LD 100 (* Load 100 into the accumulator *)
MAX 200 (* Compare with the value 200 *)
ST iValue (* Save 200 in Value2 (because larger value) *)
```

Example in ST:

```
Ergebnis := Max(100, 200); (* The result is 200 *)
```

3.4.1.6 MIN

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	ÎNT	DINT
Data type		X	X	X

This command determines the minimum value of two variables or constants. For this, the current value of the accumulator is compared with the value transferred in the MIN command. After the command, the smaller of the two values is in the accumulator. Both values must belong to the same type of variable.

Example in IL:

```
LD 100 (* Load 100 into the accumulator *)
MIN 200 (* Compare with the value 200 *)
ST Value2 (* Save 100 in Value2 (because smaller value) *)
```

Example in ST:

```
Ergebnis := Min(100, 200); (* Save 100 in Value2 (because smaller value) *)
```

3.4.1.7 MOD and MOD(

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	ÎNT	DINT
Data type		X	X	X

The Accumulator is divided by one or more variables or constant and the remainder of the division is the result in the accumulator. For the bracket Modulus, the accumulator is divided by the result of the expression in the brackets and the modulus is formed from this. Up to 6 bracket levels are possible.

Example in IL:

```
LD 25  (* Load the dividend *)
MOD 20 (* Division 25/20 per modulus = 5 *)
ST Var1 (* Save result 5 in Var1 *)
LD 25  (* Load the dividend *)
MOD( Var1 (* Result = 25/(Var1 + 10) per modulus into the accumulator *)
ADD 10
)
ST Var3 (* Save result 10 in Var3 *)
```

Example in ST:

```
Ergebnis := 25 MOD 20; (* Save result 5 in Var1 *)
Ergebnis := 25 MOD (Var1 + 10); (* Result = 25/(Var1 + 10) per modulus into the accumulator *)
```

3.4.1.8 MUL and MUL(

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	ÎNT	DINT
Data type		X	X	X

Multiplication of the accumulator with one or more variables or constants. For bracket multiplication, the accumulator is multiplied by the result of the expression in brackets. Up to 6 bracket levels are possible. Both values must belong to the same type of variable.

Example in IL:

```
LD 25  (* Load the multiplier *)
MUL Var1, Var2 (* 25 * Var1 * Var2 *)
ST Var2  (* Save result *)
LD 25  (* Load the multiplier *)
MUL( Var1  (* Result = 25*(Var1 + Var2) *)
ADD Var2
ST Var3  (* Save result as variable Var3 *)
)
```

Example in ST:

```
Ergebnis := 25 * Var1 * Var2;
Ergebnis := 25 * (Var1 + Var2);
```

3.4.1.9 MUX

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	ÎNT	DINT
Data type		X	X	X

Various constants or variables can be selected via an index, which is located in front of the command in the accumulator. The first value is accessed via the Index 0. The selected value is loaded into the accumulator. The number of values is only limited by the program memory.

Example in IL:

```
LD 1 (* Select the required element *)
MUX 10,20,30,40,Value1 (* MUX command with 4 constants and a variable *)
ST Value (* Save result = 20 *)
```

Example in ST:

```
Ergebnis := Mux(1, 10, 20, 30, 40, Value1) (* Save result = 20 *)
```

3.4.1.10 SUB and SUB(

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	ÎNT	DINT
Data type		X	X	X

Subtracts the accumulator from one or more variables or constants. For bracket subtraction, the accumulator is subtracted from the result of the expression in brackets. Up to 6 bracket levels are possible. The values to be subtracted must belong to the same type of variable.

Example in IL:

```
LD 10
SUB Var1 (* Result = 10 - Var1 *)
ST Ergebnis
LD 20
SUB Var1, Var2, 30 (* Result = 20 - Var1 - Var12 - 30 *)
ST Ergebnis
LD 20
SUB( 6 (* Substract 20 from the contents of the bracket *)
AND 2
) (* Result = 20 - (6 AND 2) *)
ST Ergebnis (* Result = 18 *)
```

Example in ST:

```
Ergebnis := 10 - Value1;
```

3.4.2 Extended mathematical operators

Information

The operators listed here require intensive computing. This may result in a considerably longer running time for the PLC program.

3.4.2.1 COS, ACOS, SIN, ASIN, TAN, ATAN

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X		

	BOOL	BYTE	ÎNT	DINT
Data type				X

Calculation of the relevant mathematical function. The value to be calculated must be available in minutes of arc. The scaling corresponds to 1 = 1000.

Conversion: Angle in radians = (Angle in degrees * PI / 180) * 1000 e.g. an angle of 90° is converted as follows: 90° * 3,14 / 180) *1000 = 1571

$$AE = \sin\left(\frac{AE}{1000}\right) \cdot 1000 \qquad AE = \cos\left(\frac{AE}{1000}\right) \cdot 1000 \qquad AE = \tan\left(\frac{AE}{1000}\right) \cdot 1000$$

Example in IL:

```
LD 1234
SIN
ST Ergebnis (* Result = 943 *)
```

Example in ST:

```
Ergebnis := COS(1234); (* Result = 330 *)
Ergebnis := ACOS(330); (* Result = 1234 *)
Ergebnis := SIN(1234); (* Result = 943 *)
Ergebnis := ASIN(943); (* Result = 1231 *)
Ergebnis := TAN(999); (* Result = 1553 *)
Ergebnis := ATAN(1553); (* Result = 998 *)
```

3.4.2.2 EXP

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X		

	BOOL	BYTE	ÎNT	DINT
Data type				X

Forms the exponential function to the base of Euler's Number (2.718) from the Accumulator. Up to 3 places behind the decimal point may be stated, i.e. 1.002 must be entered as 1002.

$$AE = e^{\left(\frac{AE}{1000}\right)} \cdot 1000$$

Example in IL:

```
LD 1000
EXP
ST Ergebnis (* Result = 2718 *)
```

Example in ST:

```
Ergebnis := EXP(1000); (* Result = 2718 *)
```

3.4.2.3 LN

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X		

	BOOL	BYTE	ÎNT	DINT
Data type				X

Logarithm to base e (2.718). Up to 3 places behind the decimal point may be stated, i.e. 1.000 must be entered as 1000.

$$AE = \ln\left(\frac{AE}{1000}\right) \cdot 1000$$

Example in IL:

```
LD 1234
LN
ST Ergebnis
```

Example in ST:

```
Ergebnis := LN(1234); (* Result = 210 *)
```

3.4.2.4 LOG

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X		

	BOOL	BYTE	ÎNT	DINT
Data type				X

Forms the base 10 logarithm from the accumulator. Up to 3 places behind the decimal point may be stated, i.e. 1.000 must be entered as 1000.

$$AE = log_{10}\left(\frac{AE}{1000}\right) \cdot 1000$$

Example in IL:

```
LD 1234
LOG
ST Ergebnis (* Result = 91 *)
```

Example in ST:

```
Ergebnis := LOG(1234); (* Result = 91 *)
```

3.4.2.5 SQRT

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X		

	BOOL	BYTE	ÎNT	DINT
Data type				X

Forms the square root from the accumulator. Up to 3 places behind the decimal point may be stated, i.e. 1.000 must be entered as 1000.

$$AE = \sqrt{\left(\frac{AE}{1000}\right)} \cdot 1000$$

Example in IL:

```
LD 1234
SQRT
ST Ergebnis (* Result = 1110 *)
```

Example in ST:

```
Ergebnis := SQRT(1234); (* Result = 1110 *)
```

3.4.3 Bit operators

3.4.3.1 AND and AND(

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	ÎNT	DINT
Data type	X	X	X	X

Bit-wise AND link of the AE/accumulator with one or two variables or constants. Bit-wise AND(...) linking with the AE/accumulator and the AE/accumulator which was previously formed in the bracket. Up to 6 bracket levels are possible. All values must belong to the same type of variable.

Example in IL:

```
LD 170
AND 204 (* AND link between 2 constants *)
(* Accumulator = 136 (See example in the table) *)

LD 170 (* Link between a constant and 2 variables.*)
AND Var1, Var2 (* Accumulator = 170dec AND Var1 AND Var2 *)

LD Var1
AND ( Var2 (* AE/Accumulator = Var1 AND ( Var2 OR Var3 ) *)
OR Var3
)
```

Example in ST:

```
Ergebnis := 170 AND 204; (* Result = 136dec *)
```

Var2	Var1	Result
0	0	0
0	1	0
1	0	0
1	1	1

Example: 170dec (1010 1010bin) AND 204dec (1100 1100bin) = (1000 1000bin) 136dec

3.4.3.2 ANDN and ANDN(

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	ÎNT	DINT
Data type	X	X	X	X

Bit-wise AND linking of the AE/accumulator with a negated operand. Bit-wise AND (...) linking of the AE/accumulator and the negated result of the bracket. Up to 6 bracket levels are possible. The values to be linked must belong to the same type of variable.

Example in IL:

```
LD 2#0000_1111
ANDN 2#0011_1010 (* ANDN link between 2 constants *)
(* Accu = 2#1111_0101 *)

LD 170 (* Link between a constant and 2 variables. *)
ANDN Var1, Var2 (* Accumulator = 170d ANDN Var1 ANDN Var2 *)

LD Var1
ANDN ( Var2 (* AE/Accumulator = Var1 ANDN ( Var2 OR Var3 ) *)
OR Var3
)
```

Var2	Var1	Result
0	0	1
0	1	1
1	0	1
1	1	0

Example: 170dec (1010 1010bin) AND 204dec (1100 1100bin) = (1000 1000bin) 136dec

3.4.3.3 NOT

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	ÎNT	DINT
Data type	X	X	X	X

Bit-wise negation of the accumulator.

Example in IL:

```
LD BYTE#10 (* Load the value 10dec into the ACCU in Byte format *)
NOT (* The value is resolved on the Bit level (0000 1010), *)
(* negated bit-wise (1111 0101) and then converted back *)
(* converted, result = 245dec *)
ST Var3 (* Save result as variable Var3 *)
```

Example in ST:

```
Ergebnis := not BYTE#10; (* Result = 245dez *)
```

3.4.3.4 OR and OR(

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	ÎNT	DINT
Data type	X	X	X	X

Bit-wise OR link of the AE/accumulator with one or two variables or constants. Bit-wise OR(...) linking with the AE/accumulator and the AE/accumulator which was previously formed in the bracket. Up to 6 bracket levels are possible. All values must belong to the same type of variable.

Example in IL:

```
LD 170
OR 204 (* OR link between 2 constants *)

LD 170 (* Link between a constant and 2 variables. *)
OR Var1, Var2 (* Accumulator = 170d OR Var1OR Var2 *)

LD Var1
OR ( Var2 (* AE/Accumulator = Var1 OR ( Var2 AND Var3 ) *)
AND Var3
)
```

Example in ST:

```
Ergebnis := 170 or 204; (* Result = 238 *)
```

Var2	Var1	Result
0	0	0
0	1	1
1	0	1
1	1	1

3.4.3.5 ORN andORN(

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	ÎNT	DINT
Data type	X	X	X	X

Bit-wise OR linking of the AE/accumulator with a negated operand. Bit-wise OR (...) linking of the AE/accumulator and the negated result of the bracket. Up to 6 bracket levels are possible. The values to be linked must belong to the same type of variable.

Example in IL:

```
LD 2#0000_1111
ORN 2#0011_1010 (* ORN link between 2 constants *)
(* Accumulator = 2#1100_0000 *)

LD 170 (* Link between a constant and 2 variables. *)
ORN Var1, Var2 (* Accumulator = 170d ORN Var1 ORN Var2 *)

LD Var1
ORN ( Var2 (* AE/Accumulator = Var1 ORN ( Var2 OR Var3 ) *)
OR Var3
)
```

Example in ST:

```
Ergebnis := 2#0000_1111 ORN 2#0011_1010; (* Result = 2#1100_0000 *)
```

Var2	Var1	Result
0	0	1
0	1	0
1	0	0
1	1	0

3.4.3.6 ROL

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type		X	X	X

Bit-wise rotation of the accumulator to the left. The content of the accumulator is shifted n times to the left, whereby the left bit is inserted again on the right.

Example in IL:

```
LD 175     (* Loads the value 1010_1111*)
ROL 2      (* Accumulator content is rotated 2x to the left *)
ST Value1  (* Saves the value 1011_1110 *)
```

Example in ST:

```
Ergebnis := ROL(BYTE#175, 2); (* Result = 2#1011_1110 *)
Ergebnis := ROL(INT#175, 2); (* Result = 16#C02B *)
```

3.4.3.7 ROR

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type		X	X	X

Bit-wise rotation of the accumulator to the right. The content of the accumulator is shifted n times to the right, whereby the right bit is inserted again on the left.

Example in IL:

```
LD 175     (* Loads the value 1010_1111*)
ROR 2      (* Accumulator content is rotated 2x to the right *)
ST Value1  (* Saves the value 1110_1011 *)
```

Example in ST:

```
Ergebnis := ROR(BYTE#175, 2); (* Result = 2#1110_1011 *)
```

3.4.3.8 S and R

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	ÎNT	DINT
Data type	X			

Sets and resets a boolean variable if the result of the previous link (the AE) was TRUE.

Example in IL:

```
LD TRUE    (* Loads the AE with TRUE *)
S Var1     (* VAR1 is set to TRUE *)
R Var1     (* VAR1 is set to FALSE *)
```

Example in ST:

```
Ergebnis := TRUE;
Ergebnis := FALSE;
```

3.4.3.9 SHL

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	ÎNT	DINT
Data type		X	X	X

Bit-wise left shift of the accumulator. The content of the accumulator is shifted n times to the left and the bits which are pushed out are lost.

Example in IL:

```
LD 175    (* Loads the value 1010_1111*)
SHL 2     (* Accumulator content is shifted 2x to the left *)
ST Value1 (* Saves the value 1011_1100 *)
```

Example in ST:

```
Ergebnis := SHL(BYTE#175, 2); (* Result = 2#1011_1100 *)
Ergebnis := SHL(INT#175, 2); (* Result = 16#2BC *)
```

3.4.3.10 SHR

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type		X	X	X

Bit-wise right shift of the accumulator. The content of the accumulator is shifted n times to the right and the bits which are pushed out are lost.

Example in IL:

```
LD 175    (* Loads the value 1010_1111*)
SHR 2     (* Accumulator content is shifted 2x to the right *)
ST Value1 (* Saves the value 0010_1011 *)
```

Example in ST:

```
Ergebnis := SHR(BYTE#175, 2); (* Result = 2#0010_1011 *)
```

3.4.3.11 XOR and XOR(

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type	X			

Bit-wise "exclusive OR" link between the AE/accumulator and one or two variables or constants. The first value is located in the AE/accumulator and the second is loaded with the command or is within the brackets. The values to be linked must belong to the same type of variable.

Example in IL:

```
LD 2#0000_1111
XOR 2#0011_1010 (* XOR link between 2 constants *)
                  (* Accu = 2#0011_0101 *)

LD 170          (* Link between a constant and 2 variables. *)
XOR Var1, Var2  (* Accumulator = 170d XOR Var1 XOR Var2 *)

LD Var1
XOR ( Var2      (* AE/Accumulator = Var1 XOR ( Var2 OR Var3 ) *)
OR Var3
)
```

Example in ST:

```
Ergebnis := 2#0000_1111 XOR 2#0011_1010; (* Result = 2#0011_0101 *)
```

Var2	Var1	Result
0	0	0
0	1	1
1	0	1
1	1	0

3.4.3.12 XORN and XORN(

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type	X			

Bit-wise Exclusive OR linking of the AE/accumulator with a negated operand. Bit-wise Exclusive OR (...) linking of the AE/accumulator and the negated result of the bracket. Up to 6 bracket levels are possible. The values to be linked must belong to the same type of variable.

Example in IL:

```
LD 2#0000_1111
XORN 2#0011_1010  (* XORN link between 2 constants *)
                  (* Accu = 2#1100_1010 *)

LD 170            (* Link between a constant and 2 variables. *)
XORN Var1, Var2   (* Accumulator = 170d XORN Var1 XORN Var2 *)

LD Var1
XORN ( Var2       (* AE/Accumulator = Var1 XORN ( Var2 OR Var3 ) *)
OR Var3
)
```

Example in ST:

```
Ergebnis := 2#0000_1111 XORN 2#0011_1010; (* Result = 2#1100_1010 *)
```

Var2	Var1	Result
0	0	1
0	1	0
1	0	0
1	1	1

3.4.4 Loading and storage operators (AWL)

3.4.4.1 LD

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type	X	X	X	X

Loads a constant or a variable into the AE or the accumulator.

Example in IL:

```
LD 10 (* Loads 10 as BYTE *)
LD -1000 (* Loads -1000 as INT *)
LD Value1 (* Loads the variable Value1 *)
```

3.4.4.2 LDN

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type	X			

Loads a negated boolean variable into the AE.

Example in IL:

```
LDN Value1 (* Value1 = TRUE at AE = FALSE *)
ST Value2 (* Save to Value2 = FALSE *)
```

3.4.4.3 ST

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type	X	X	X	X

Saves the content of the AE/accumulator to a variable. The variable to be saved must match the previously loaded and processed data type.

Example in IL:

```
LD 100 (* Loads the value 1010_1111 *)
ST Value1 (* Accumulator content 100 is saved in Value1 *)
```

3.4.4.4 STN

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type	X			

Saves the content of the AE to a variable and negates it. The variable to be saved must match the previously loaded and processed data type.

Example in IL:

```
LD Value1 (* Value1 = TRUE at AE = TRUE *)
STN Value2 (* Save to Value2 = FALSE *)
```

3.4.5 Comparison operators

3.4.5.1 EQ

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type		X	X	X

Compares the content of the accumulator with a variable or constant. If the values are equal, then AE is set to TRUE.

Example in IL:

```
LD Value1 (* Value1 = 5 *)
EQ 10 (* AE = Is 5 equal to 10 ? *)
JMPC NextStep (* AE = FALSE - program does not jump *)
ADD 1
NextStep:
ST Value1
```

Example in ST:

```
(* Is value = 10 *)
if Value = 10 then
  Value2 := 5;
end_if;
```

3.4.5.2 GE

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type		X	X	X

Compares the content of the accumulator with a variable or constant. If the value in the accumulator is greater or equal to the variable or constant, then AE is set to TRUE.

Example in IL:

```
LD Value1 (* Value1 = 5*)
GE 10 (* Is 5 greater than or equal to 10? *)
JMPC NextStep (* AE = FALSE - program does not jump *)
ADD 1

NextStep:
ST Value1
```

Example in ST:

```
(* Is 5 greater than or equal to 10? *)
if Value >= 10 then
  Value := Value - 1
end_if;
```

3.4.5.3 GT

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type		X	X	X

Compares the content of the accumulator with a variable or constant. If the value in the accumulator is greater than the variable or constant, then AE is set to TRUE.

Example in IL:

```
LD Value1(* Value1 = 12 *)
GT 8 (* Is 12 greater than 8? *)
JMPC NextStep (* AE = TRUE - program jumps *)
ADD 1
NextStep:
ST Value1
```

Example in ST:

```
(* Is 12 greater than 8? *)
if Value > 8 then
  Value := 0;
end_if;
```

3.4.5.4 LE

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type		X	X	X

Compares the content of the accumulator with a variable or constant. If the value in the Accumulator is less than or equal to the variable or constant, then AE is set to TRUE.

Example in IL:

```
LD Value1 (* Value1 = 5*)
LE 10 (* Is 5 less than or equal to 10? *)
JMPC NextStep:
ST Value1
```

Example in ST:

```
(* Is Value less than or equal to 10?*)
if Value <= 10 then
  Value := 11;
end_if;
```

3.4.5.5 LT

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type		X	X	X

Compares the content of the accumulator with a variable or constant. If the value in the accumulator is less than the variable or constant the AE is set to TRUE.

Example in IL:

```
LD Value1 (* Value1 = 12 *)
LT 8 (* Is 12 less than 8 ? *)
JMPC NextStep (* AE = FALSE - program does not jump *)
ADD 1
NextStep:
ST Value1
```

Example in ST:

```
(* Is Value less than 0? *)
if Value < 0 then
  Value := 0;
end_if;
```

3.4.5.6 NE

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type		X	X	X

Compares the content of the accumulator with a variable or constant. If the value in the Accumulator is not equal to the variable or constant, then AE is set to TRUE.

Example in IL:

```
LD Value1 (* Value1 = 5 *)
NE 10 (*Is 5 not equal to 10 ?*)
JMPC NextStep (* AE = TRUE - program jumps *)
ADD 1
NextStep:
ST Value1
```

Example in ST:

```
if Value <> 5 then
Value := 5;
end_if;
```

3.5 Processing values

All analogue and digital inputs and outputs or bus setpoints and actual values can be read and processed by the PLC or can be set by the PLC (if they are output values). Access to the individual values is via the process values listed below. For all output values, the output (e.g. digital outputs or PLC setpoint) must be programmed so that the PLC is the source of the event. All process data is read in from the PLC by the device at the start of each cycle and is only written to the device at the end of the program. The following table lists all of the values which can be directly accessed by the PLC. All other process values must be accessed via the function blocks MC_ReadParameter or MC_WriteParameter.

3.5.1 Inputs and outputs

All process values which describe the I/O interface of the device are summarised here.

Name	Function	Standardisation	Type	Access	Device
_0_Set_digital_output	Set digital outputs	Bit 0: Mfr1 Bit 1: Mfr2 Bit 2: DOUT 1 Bit 3: DOUT 2 Bit 4: DOUT 1 CU5-MLT Bit 5: DOUT 2 CU5-MLT Bit 6: DOUT 3 CU5-MLT Bit 7: DOUT 4 CU5-MLT Bit 8: dig. function AOUT Bit 9: vacant Bit 10: BusIO Bit0 Bit 11: BusIO Bit1 Bit 12: BusIO Bit2 Bit 13: BusIO Bit3 Bit 14: BusIO Bit4 Bit 15: BusIO Bit5	UINT	R/W	SK 5xxP On/On+
_0_Set_digital_output	Set digital outputs	Bit 0: Mfr1 Bit 1: Mfr2 Bit 2: DOUT1 Bit 3: DOUT2 Bit 4: dig. function AOUT Bit 5: DOUT3 (Din7) Bit 6: Status word Bit 10 Bit 7: Status word Bit 13 Bit 8: BusIO Bit0 Bit 9: BusIO Bit1 Bit 10: BusIO Bit2 Bit 11: BusIO Bit3 Bit 12: BusIO Bit4 Bit 13: BusIO Bit5 Bit 14: BusIO Bit6 Bit 15: BusIO Bit7	UINT	R/W	SK 54xE
_0_Set_digital_output	Set digital outputs	Bit 0: Mfr1 Bit 1: Mfr2 Bit 2: DOUT1	UINT	R/W	SK 52xE SK 53xE

Name	Function	Standardisation	Type	Access	Device
		Bit 3: DOUT2 Bit 4: dig. function AOUT Bit 5: vacant Bit 6: Status word Bit 10 Bit 7: Status word Bit 13 Bit 8: BusIO Bit0 Bit 9: BusIO Bit1 Bit 10: BusIO Bit2 Bit 11: BusIO Bit3 Bit 12: BusIO Bit4 Bit 13: BusIO Bit5 Bit 14: BusIO Bit6 Bit 15: BusIO Bit7			
_0_Set_digital_output	Set digital outputs	Bit 0: DOUT1 Bit 1: BusIO Bit0 Bit 2: BusIO Bit1 Bit 3: BusIO Bit2 Bit 4: BusIO Bit3 Bit 5: BusIO Bit4 Bit 6: BusIO Bit5 Bit 7: BusIO Bit6 Bit 8: BusIO Bit7 Bit 9: Bus PZD Bit 10 Bit 10: Bus PZD Bit 13 Bit 11: DOUT2	UINT	R/W	SK 2xxE SK 2xxE-FDS
_0_Set_digital_output	Set digital outputs	Bit 0: DOUT1 Bit 1: DOUT2 Bit 2: BusIO Bit0 Bit 3: BusIO Bit1 Bit 4: BusIO Bit2 Bit 5: BusIO Bit3 Bit 6: BusIO Bit4 Bit 7: BusIO Bit5 Bit 8: BusIO Bit6 Bit 9: BusIO Bit7 Bit 10: Bus PZD Bit 10 Bit 11: Bus PZD Bit 13	UINT	R/W	SK 180E SK 190E
_0_Set_digital_output	Set digital outputs	Bit 0: DOUT1 Bit 1: DOUT2 Bit 2: DOUT_BRAKE Bit 3: DOUT_BUS1 Bit 4: DOUT_BUS2	UINT	R/W	SK 155E-FDS SK 175E-FDS
_1_Set_analog_output	Set FI analogue output	10.0V = 100	BYTE	R/W	SK 5xxP SK 54xE SK 53xE SK 52xE On/On+
_2_Set_external_ analog_out1	Set analogue output 1. IOE	10.0V = 100	BYTE	R/W	SK 5xxP SK 54xE

Name	Function	Standardisation	Type	Access	Device
					SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_3_Set_external_ analog_out2	Set analogue output 2. IOE	10.0V = 100	BYTE	R/W	SK 5xxP SK 54xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_4_State_digital_output	Status of digital outputs	Bit 0: Mfr1 Bit 1: Mfr2 Bit 2: DOUT 1 Bit 3: DOUT 2 Bit 4: DOUT 1 CU5-MLT Bit 5: DOUT 2 CU5-MLT Bit 6: DOUT 3 CU5-MLT Bit 7: DOUT 4 CU5-MLT Bit 8: dig. function AOUT Bit 9: vacant Bit 10: DOUT1 IOE1 Bit 11: DOUT2 IOE1 Bit 12: DOUT1 IOE2 Bit 13: DOUT2 IOE2 Bit 14: vacant Bit 15: vacant	INT	R	SK 5xxP On/On+
_4_State_digital_output	Status of digital outputs	Bit 0: Mfr1 Bit 1: Mfr2 Bit 2: DOUT1 Bit 3: DOUT2 Bit 4: dig. function AOUT Bit 5: DOUT3 (Din7) Bit 6: Status word Bit 8 Bit 7: Status word Bit 9 Bit 8: BusIO Bit0 Bit 9: BusIO Bit1 Bit 10: BusIO Bit2 Bit 11: BusIO Bit3 Bit 12: BusIO Bit4 Bit 13: BusIO Bit5 Bit 14: BusIO Bit6 Bit 15: BusIO Bit7	INT	R	SK 54xE
_4_State_digital_output	Status of digital outputs	P711	BYTE	R	SK 52xE SK 53xE SK 2xxE SK 2xxE-FDS SK 180E

Name	Function	Standardisation	Type	Access	Device
					SK 190E
_4_State_digital_output	Status of digital outputs	Bit 0: DOUT1 Bit 1: DOUT2 Bit 2: DOUT_BRAKE Bit 3: DOUT_BUS1 Bit 4: DOUT_BUS2	BYTE	R	SK 155E-FDS SK 175E-FDS
_5_State_Digital_input	Status of digital inputs	Bit 0: DIN1 Bit 1: DIN2 Bit 2: DIN3 Bit 3: DIN4 Bit 4: DIN5 Bit 5: DIN6 Bit 6: DIN1 CU5-MLT Bit 7: DIN2 CU5-MLT Bit 8: DIN3 CU5-MLT Bit 9 DIN4 CU5-MLT Bit 10 vacant Bit 11 vacant Bit 12: Digital function AIN1 Bit 8: Digital function AIN2	INT	R	SK 5xxP On/On+
_5_State_Digital_input	Status of digital inputs	Bit 0: DIN1 Bit 1: DIN2 Bit 2: DIN3 Bit 3: DIN4 Bit 4: DIN5 Bit 5: DIN6 Bit 6: DIN7 Bit 7: Digital function AIN1 Bit 8: Digital function AIN2	INT	R	SK 54xE
_5_State_Digital_input	Status of digital inputs	Bit 0: DIN1 Bit 1: DIN2 Bit 2: DIN3 Bit 3: DIN4 Bit 4: DIN5 Bit 5: DIN6 Bit 6: DIN7	INT	R	SK 52xE SK 53xE
_5_State_Digital_input	Status of digital inputs	Bit 0: DIN1 Bit 1: DIN2 Bit 2: DIN3 Bit 3: DIN4 Bit 4: vacant Bit 5: Thermistor Bit 6: vacant Bit 7: vacant Bit 8: DIN1 IOE 1	INT	R	SK 2xxE

Name	Function	Standardisation	Type	Access	Device
		Bit 9: DIN2 IOE 1 Bit 10: DIN3 IOE 1 Bit 11: DIN4 IOE 1 Bit 12: DIN1 IOE 2 Bit 13: DIN2 IOE 2 Bit 14: DIN3 IOE 2 Bit 15: DIN4 IOE 2			
_5_State_Digital_input	Status of digital inputs	Bit 0: DIN1 Bit 1: DIN2 Bit 2: DIN3 Bit 3: AIN1 Bit 4: AIN2 Bit 5: Thermistor Bit 6: vacant Bit 7: vacant Bit 8: DIN1 IOE 1 Bit 9: DIN2 IOE 1 Bit 10: DIN3 IOE 1 Bit 11: DIN4 IOE 1 Bit 12: DIN1 IOE 2 Bit 13: DIN2 IOE 2 Bit 14: DIN3 IOE 2 Bit 15: DIN4 IOE 2	INT	R	SK 180E SK 190E
_5_State_Digital_input	Status of digital inputs	Bit 0: DIN1 Bit 1: DIN2 Bit 2: DIN3 Bit 3: TF (thermistor) Bit 4: DIN-BUS1 (ASil1) Bit 5: DIN-BUS2 (ASil2) Bit 6: DIN-BUS3 (ASil3) Bit 7: DIN-BUS4 (ASil4) Bit 8: BDDI1 (ASIO3) Bit 9: BDDI2 (ASIO4) Bit 10: STO	INT	R	SK 155E-FDS SK 175E-FDS
_5_State_Digital_input	Status of digital inputs	Bit 0: DIN1 Bit 1: DIN2 Bit 2: DIN3 Bit 3: DIN4 Bit 4: DIN5 Bit 5: DIN6/AIN1 Bit 6: DIN7/AIN2 Bit 7: Thermistor Bit 8: DIN1 IOE 1 Bit 9: DIN2 IOE 1 Bit 10: DIN3 IOE 1 Bit 11: DIN4 IOE 1 Bit 12: DIN1 IOE 2 Bit 13: DIN2 IOE 2 Bit 14: DIN3 IOE 2 Bit 15: DIN4 IOE 2	INT	R	SK 2xxE-FDS

Name	Function	Standardisation	Type	Access	Device
_6_Delay_digital_inputs	Status of digital inputs according to P475	Bit 0: DIN1 Bit 1: DIN2 Bit 2: DIN3 Bit 3: DIN4 Bit 4: DIN5 Bit 5: DIN6 Bit 6: DIN7 Bit 7: Digital function AIN1 Bit 8: Digital function AIN2	INT	R	SK 5xxP SK 54xE On/On+
_6_Delay_digital_inputs	Status of digital inputs according to P475	Bit 0: DIN1 Bit 1: DIN2 Bit 2: DIN3 Bit 3: DIN4 Bit 4: DIN5 Bit 5: DIN6 Bit 6: DIN7	INT	R	SK 52xE SK 53xE
_6_Delay_digital_inputs	Status of digital inputs according to P475	Bit 0: DIN1 Bit 1: DIN2 Bit 2: DIN3 Bit 3: AIN1 Bit 4: AIN2 Bit 5: Thermistor Bit 6: vacant Bit 7: vacant Bit 8: DIN1 IOE 1 Bit 9: DIN2 IOE 1 Bit 10: DIN3 IOE 1 Bit 11: DIN4 IOE 1 Bit 12: DIN1 IOE 2 Bit 13: DIN2 IOE 2 Bit 14: DIN3 IOE 2 Bit 15: DIN4 IOE 2	INT	R	SK 2xxE SK 180E SK 190E
_6_Delay_digital_inputs	Status of digital inputs according to P475	Bit 0: DIN1 Bit 1: DIN2 Bit 2: DIN3 Bit 3: DIN4 Bit 4: DIN5 Bit 5: DIN6/AIN1 Bit 6: DIN7/AIN2 Bit 7: Thermistor Bit 8: DIN1 IOE 1 Bit 9: DIN2 IOE 1 Bit 10: DIN3 IOE 1 Bit 11: DIN4 IOE 1 Bit 12: DIN1 IOE 2 Bit 13: DIN2 IOE 2 Bit 14: DIN3 IOE 2 Bit 15: DIN4 IOE 2	INT	R	SK 2xxE-FDS

Name	Function	Standardisation	Type	Access	Device
_7_Analog_input1	Value of analogue input 1 (AIN1)	10.00V = 1000	INT	R	All
_8_Analog_input2	Value of analogue input 2 (AIN2)	10.00V = 1000	INT	R	All
_9_Analog_input3	Value of analogue function DIN2	10.00V = 1000	INT	R	SK 5xxP SK 54xE SK 155E-FDS SK 175E-FDS
_10_Analog_input4	Value of analogue function DIN3	10.00V = 1000	INT	R	SK 5xxP SK 54xE SK 155E-FDS SK 175E-FDS
_11_External_analog_input1	Value of analogue input 1 (1.IOE)	10.00V = 1000	INT	R	SK 5xxP SK 54xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E
_12_External_analog_input2	Value of analogue input 2 (1.IOE)	10.00V = 1000	INT	R	SK 5xxP SK 54xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E
_13_External_analog_input3	Value of analogue input 1 (2.IOE)	10.00V = 1000	INT	R	SK 5xxP SK 54xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E
_14_External_analog_input4	Value of analogue input 2 (2.IOE)	10.00V = 1000	INT	R	SK 5xxP SK 54xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E
_15_State_analog_output	Status of analogue output	10.0V = 100	BYTE	R	SK 5xxP SK 54xE
_16_State_ext_analog_out1	Status of analogue output (1. IOE)	10.00V = 1000	INT	R	SK 5xxP SK 54xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E
_17_State_ext_analog_out2	Status of analogue output (2. IOE)	10.00V = 1000	INT	R	SK 5xxP SK 54xE

Name	Function	Standardisation	Type	Access	Device
					SK 2xxE SK 180E SK 190E
_18_Dip_switch_state	DIP switch status	Bit 0: DIP1 Bit 1: DIP2 Bit 2: DIP3 Bit 3: DIP4 Bit 4: DIP_I1 Bit 5: DIP_I2 Bit 6: DIP_I3 Bit 7: DIP_I4	INT	R	SK 155E-FDS SK 175E-FDS
_19_State_digital_input _IOE	Status of digital inputs (IOE)	Bit 0: DIN1 IOE 2 Bit 1: DIN2 IOE 2 Bit 2: DIN3 IOE 2 Bit 3: DIN4 IOE 2 Bit 4: DIN1 IOE 1 Bit 5: DIN2 IOE 1 Bit 6: DIN3 IOE 1 Bit 7: DIN4 IOE 1	INT	R	SK 5xxP On/On+

3.5.2 PLC setpoint and actual values

The process values listed here form the interface from the PLC to the device. The function of the PLC setpoints is specified in (P553).

Information

The process value PLC_control_word overwrites the function block MC_Power. The PLC setpoints overwrite the function blocks MC_Move…. und MC_Home.

Name	Function	Standardisation	Type	Access	Device
_20_PLC_control_word	PLC control word	Corresponds to the USS profile	INT	R/W	All
_21_PLC_set_val1	PLC setpoint 1	100% = 4000h	INT	R/W	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_22_PLC_set_val2	PLC setpoint 2	100% = 4000h	INT	R/W	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_23_PLC_set_val3	PLC setpoint 3	100% = 4000h	INT	R/W	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_24_PLC_set_val4	PLC setpoint 4	100% = 4000h	INT	R/W	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS On/On+
_25_PLC_set_val5	PLC setpoint 5	100% = 4000h	INT	R/W	SK 5xxP SK 54xE SK 53xE

Name	Function	Standardisation	Type	Access	Device
					SK 52xE SK 2xxE SK 2xxE-FDS On/On+
_26_PLC_additional_control_word1	PLC additional control word 1	Corresponds to the USS profile	INT	R/W	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_27_PLC_additional_control_word2	PLC additional control word 2	Corresponds to the USS profile	INT	R/W	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_28_PLC_status_word	PLC status word	Corresponds to the USS profile	INT	R/W	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_29_PLC_act_val1	PLC actual value 1	100% = 4000h	INT	R/W	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_30_PLC_act_val2	PLC actual value 2	100% = 4000h	INT	R/W	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+

Name	Function	Standardisation	Type	Access	Device
_31_PLC_act_val3	PLC actual value 3	100% = 4000h	INT	R/W	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_32_PLC_act_val4	PLC actual value 4	100% = 4000h	INT	R/W	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS On/On+
_33_PLC_act_val5	PLC actual value 5	100% = 4000h	INT	R/W	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS On/On+
_34_PLC_Busmaster_ Control_word	Master function control word (bus master function) via PLC	Corresponds to the USS profile	INT	R/W	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_35_PLC_32Bit_set_ val1	32Bit PLC setpoint - P553[1] = Low part of the 32Bit value - P553[2] = High part of the 32Bit value	_	LONG	R/W	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_36_PLC_32Bit_act_ val1	32Bit PLC actual value - PLC actual value 1 = Low part of the 32Bit value - PLC 2 = High part of the 32Bit value	_	LONG	R/W	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+

Name	Function	Standardisation	Type	Access	Device
_37_PLC_status_bits	Virtual status outputs of the PLC	Bit 0: PLC-DOUT1 Bit 1: PLC-DOUT2	INT	R/W	SK 155E-FDS SK 175E-FDS
_38_PLC_control_bits	Virtual control outputs of the PLC	Bit 0: PLC-DIN1 Bit 1: PLC-DIN2 Bit 2: PLC-DIN3 Bit 3: PLC-DIN4 Bit 4: PLC-DIN5 Bit 5: PLC-DIN6 Bit 6: PLC-DIN7 Bit 7: PLC-DIN8	INT	R/W	SK 155E-FDS SK 175E-FDS
_39_PLC_set_digital_ output_bus	Outgoing PLC BusI/O data	Bit 0: BusIO Bit0 Bit 1: BusIO Bit1 Bit 2: BusIO Bit2 Bit 3: BusIO Bit3 Bit 4: BusIO Bit4 Bit 5: BusIO Bit5 Bit 6: BusIO Bit6 Bit 7: BusIO Bit7 Bit 8: Flag 1 Bit 9: Flag 2 Bit 10: Status word Bit 11 Bit 11: Status word Bit 12	INT	R/W	SK 5xxP On/On+

3.5.3 Bus setpoints and actual values

These process values reflect all setpoints and actual values which are transferred to the device via the various bus systems.

Name	Function	Standardisation	Type	Access	Device
_40_Inverter_status	FU status word	Corresponds to the USS profile	INT	R	All
_41_Inverter_act_val1	FU actual value 1	100% = 4000h	INT	R	All
_42_Inverter_act_val2	FU actual value 2	100% = 4000h	INT	R	All
_43_Inverter_act_val3	FU actual value 3	100% = 4000h	INT	R	All
_44_Inverter_act_val4	FU actual value 4	100% = 4000h	INT	R	SK 5xxP SK 54xE On/On+
_45_Inverter_act_val5	FU actual value 5	100% = 4000h	INT	R	SK 5xxP SK 54xE On/On+
_46_Inverter_lead_val1	Broadcast Master Function: Master value 1	100% = 4000h	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_47_Inverter_lead_val2	Broadcast Master Function: Master value 2	100% = 4000h	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_48_Inverter_lead_val3	Broadcast Master Function: Master value 3	100% = 4000h	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_49_Inverter_lead_val4	Broadcast Master Function: Master value 4	100% = 4000h	INT	R	SK 5xxP SK 54xE On/On+

Name	Function	Standardisation	Type	Access	Device
_50_Inverter_lead_val5	Broadcast Master Function: Master value 5	100% = 4000h	INT	R	SK 5xxP SK 54xE On/On+
_51_Inverter_control_word	Resulting bus control word	Corresponds to the USS profile	ÎNT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_52_Inverter_set_val1	Resulting main bus setpoint 1	100% = 4000h	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_53_Inverter_set_val2	Resulting main bus setpoint 2	100% = 4000h	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_54_Inverter_set_val3	Resulting main bus setpoint 3	100% = 4000h	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_55_Inverter_set_val4	Resulting main bus setpoint 4	100% = 4000h	INT	R	SK 5xxP SK 54xE On/On+
_56_Inverter_set_val5	Resulting main bus setpoint 5	100% = 4000h	INT	R	SK 5xxP SK 54xE On/On+
_57_Broadcast_set_val1	Broadcast Slave: Auxiliary setpoint 1	100% = 4000h	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE

Name	Function	Standardisation	Type	Access	Device
					SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_58_Broadcast_set_val 2	Broadcast Slave: Auxiliary setpoint 2	100% = 4000h	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 180E SK 190E On/On+
_59_Broadcast_set_val 3	Broadcast Slave: Auxiliary setpoint 3	100% = 4000h	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 180E SK 190E On/On+
_60_Broadcast_set_val 4	Broadcast Slave: Auxiliary setpoint 4	100% = 4000h	INT	R	SK 5xxP SK 54xE On/On+
_61_Broadcast_set_val 5	Broadcast Slave: Auxiliary setpoint 5	100% = 4000h	INT	R	SK 5xxP SK 54xE On/On+
_62_Inverter_32Bit_set _val1	Resulting 32Bit main setpoint 1 Bus	- Low part in P546[1] - High part in P546[2]	LONG	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 180E SK 190E On/On+
_63_Inverter_32Bit_act _val1	FI 32Bit actual value 1	- Low part in P543[1] - High part in P543[2]	LONG	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 180E SK 190E On/On+
_64_Inverter_32Bit_lea d_val1	32Bit lead value 1	- Low part in P502[1] - High part in P502[2]	LONG	R	SK 5xxP SK 54xE SK 2xxE SK 180E SK 190E On/On+

Name	Function	Standardisation	Type	Access	Device
_65_Broadcast_32Bit_set_val1	32Bit Broadcast Slave auxiliary setpoint 1	- Low part in P543[1] - High part in P543[2]	LONG	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 180E SK 190E On/On+
_66_BusIO_input_bits	Incoming Bus I/O data	- Bit0 – 7 = Bus I/O In Bit 0 – 7 - Bit 8 = Flag 1 - Bit 9 = Flag 2 - Bit 10 = Bit8 of Bus control word - Bit 11 = Bit9 of Bus control word	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_67_BusIO_output_bits	Outgoing Bus I/O data	Bit0 = Bus / AS-i Dig Out1 Bit1 = Bus / AS-i Dig Out2 Bit2 = Bus / AS-i Dig Out3 Bit3 = Bus / AS-i Dig Out4 Bit4 = Bus / 1.IOE Dig Out1 Bit5 = Bus / 1.IOE Dig Out2 Bit6 = Bus / 2.IOE Dig Out1 Bit7 = Bus / 2.IOE Dig Out2 Bit8 = Bit 10 Bus status word Bit9 = Bit 11 Bus status word	INT	R	SK 5xxP SK 54xE On/On+
_67_BusIO_output_bits	Outgoing Bus I/O data	Bit0 = Bus / AS-i Dig Out1 Bit1 = Bus / AS-i Dig Out2 Bit2 = Bus / AS-i Dig Out3 Bit3 = Bus / AS-i Dig Out4 Bit4 = AS-i Actuator 1 Bit5 = AS-i Actuator 2 Bit6 = Flag 1 Bit7 = Flag 2 Bit8 = Bit 10 Bus status word Bit9 = Bit 11 Bus status	INT	R	SK 53xE SK 52xE

Name	Function	Standardisation	Type	Access	Device
		word			
_67_BusIO_output_bits	Outgoing Bus I/O data	Bit0 = Bus / AS-i Dig Out1 Bit1 = Bus / AS-i Dig Out2 Bit2 = Bus / AS-i Dig Out3 Bit3 = Bus / AS-i Dig Out4 Bit4 = Bus / IOE Dig Out1 Bit5 = Bus / IOE Dig Out2 Bit6 = Bus / 2nd IOE Dig Out1 Bit7 = Bus / 2nd IOE Dig Out2 Bit8 = Bit 10 Bus status word Bit9 = Bit 11 Bus status word	INT	R	SK 2xxE
_67_BusIO_output_bits	Outgoing Bus I/O data	Bit0 = Bus / AS-i Dig Out1 Bit1 = Bus / AS-i Dig Out2 Bit2 = Bus / AS-i Dig Out3 Bit3 = Bus / AS-i Dig Out4 Bit4 = Bus / AS-i Dig Out5 Bit5 = Bus / AS-i Dig Out6 Bit6 = Bus / 2nd IOE Dig Out1 Bit7 = Bus / 2nd IOE Dig Out2 Bit8 = Bit 10 Bus status word Bit9 = Bit 11 Bus status word	INT	R	SK 2xxE-FDS

3.5.4 ControlBox and ParameterBox

The ControlBox can be accessed via the process values listed here. This enables implementation of simple HMI applications.

Information

In order for the "key-states" to be displayed in the PLC, the ControlBox and ParameterBox must be in PLC display mode. Otherwise only the value "0" is displayed

Name	Function	Standardisation	Type	Access	Device
_70_Set_controlbox_show_val	ControlBox display value	Display value = Bit 29 – Bit 0 Decimal point = Bit 31 – Bit 30	DINT	R/W	All
_71_Controlbox_key_state	ControlBox keyboard status	Bit 0: ON Bit 1: OFF Bit 2: DIR Bit 3: UP Bit 4: DOWN Bit 5: Enter	BYTE	R	All
_72_Parameterbox_key_state	ParameterBox keyboard status	Bit 0: ON Bit 1: OFF Bit 2: DIR Bit 3: UP Bit 4: DOWN Bit 5: Enter Bit 6: Right Bit 7: Left	BYTE	R	All

3.5.5 Information parameters

The main actual values of the device are listed here.

Name	Function	Standardisation	Type	Access	Device
_80_Current_fault	Actual fault number	Error 10.0= 100	BYTE	R	All
_81_Current_warning	Actual warning	Warning 10.0= 100	BYTE	R	All
_82_Current_reason_ FI_blocked	Actual reason for the switch-on inhibit state	Problem 10.0= 100	BYTE	R	All
_83_Input_voltage	Actual mains voltage	100 V = 100	INT	R	All
_84_Current_frequenz	Actual frequency	10Hz = 100	INT	R	All
_85_Current_set_ point_frequency1	Actual setpoint frequency from the setpoint source	10Hz = 100	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_86_Current_set_ point_frequency2	Actual inverter set point frequency	10Hz = 100	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_87_Current_set_ point_frequency3	Actual set point frequency after ramp	10Hz = 100	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_88_Current_Speed	Actually calculated speed	100rpm= 100	INT	R	All
_89_Actual_current	Actual output current	10.0A = 100	INT	R	All
_90_Actual_torque_ current	Actual torque current	10.0A = 100	INT	R	All
_91_Current_voltage	Actual voltage	100V = 100	ÎNT	R	All

Name	Function	Standardisation	Type	Access	Device
_92_Dc_link_voltage	Actual link circuit voltage	100V = 100	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_93_Actual_field_ current	Actual field current	10.0A = 100	INT	R	All
_94_Voltage_d	Actual voltage component d-axis	100V = 100	INT	R	All
_95_Voltage_q	Actual voltage component q-axis	100V = 100	INT	R	All
_96_Current_cos_phi	Actual cos (phi)	0.80 = 80	BYTE	R	All
_97_Torque	Actual torque	100% = 100	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_98_Field	Actual field	100% = 100	BYTE	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_99_Apparent_power	Actual apparent power	1.00KW = 100	INT	R	All
_100_Mechanical_ power	Actual mechanical power	1.00KW = 100	INT	R	All
_101_Speed_encoder	Actual measured speed	100rpm= 100	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE On/On+
_102_Usage_rate_ motor	Actual usage rate of motor	100% = 100	INT	R	All

Name	Function	Standardisation	Type	Access	Device
_103_Usage_rate_motor_I2t	Actual usage rate of motor I2t	100% = 100	INT	R	SK 5xxP SK 54xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_104_Usage_rate_brake_resistor	Actual brake resistor usage rate.	100% = 100	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_105_Head_sink_temp	Actual heat sink temperature	100°C = 100	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_106_Inside_temp	Actual internal temperature	100°C = 100	INT	R	SK 5xxP SK 54xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_107_Motor_temp	Actual motor temperature	100°C = 100	INT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 2xxE-FDS SK 180E SK 190E On/On+
_108_Actual_net_frequency	Actual mains frequency	10Hz = 100	INT	R	SK 155E-FDS SK 175E-FDS
_109_Mains_phase_sequence	Actual mains phase sequence	0=CW, 1=CCW	BYTE	R	SK 155E-FDS SK 175E-FDS

Name	Function	Standardisation	Type	Access	Device
_141_Pos_Sensor_Inc	Position of incremental encoder	0.001 rotation	DINT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 180E SK 190E On/On+
_142_Pos_Sensor_Abs	Position of absolute encoder	0.001 rotation	DINT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 180E SK 190E On/On+
_143_Pos_Sensor_Uni	Position of universal encoder	0.001 rotation	DINT	R	SK 5xxP SK 54xE On/On+
_144_Pos_Sensor_HTL	Position of HTL encoder	0.001 rotation	DINT	R	SK 5xxP SK 54xE On/On+
_145_Actual_pos	Actual position	0.001 rotation	DINT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 180E SK 190E On/On+
_146_Actual_ref_pos	Actual setpoint position	0.001 rotation	DINT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 180E SK 190E On/On+
_147_Actual_pos_diff	Position difference between setpoint and actual value	0.001 rotation	DINT	R	SK 5xxP SK 54xE SK 53xE SK 52xE SK 2xxE SK 180E SK 190E On/On+
_150_Direct_dc_link_ voltage	Actual link circuit voltage (unfiltered)	100V = 100	INT	R	SK 5xxP On/On+

Name	Function	Standardisation	Type	Access	Device
_151_Direct_torque_current	Actual torque current (unfiltered)	ST: `Value := INT_TO_DINT(_151_Direct_torque_current) * INT_TO_DINT(_153_Factor_InFu_B)) / DINT#819` 1A = 100	INT	R	SK 5xxP On/On+

_152_Direct_field_current	Actual field current (unfiltered)	IL:
		`LD _153_Factor_InFu_B`
		`INT_TO_DINT`
		`ST Num_InFu`
		
		`LD _151_Direct_torque_current`
		`INT_TO_DINT`
		`MUL Num_InFu`
		`DIV DINT#819`
		`ST Value`
		
		1A = 100
			INT	R	SK 5xxP On/On+

_153_Factor_InFu_B	Factor for calculation of the actual torque or field current		INT	R	SK 5xxP On/On+

3.5.6 PLC errors

The device errors E23.0 to E24.7 can be set from the PLC program via the User Error Flags.

Name	Function	Standardisation	Type	Access	Device
_110_ErrorFlags	Generates user error in device	Bit 0: E 23.0 Bit 1: E 23.1 Bit 2: E 23.2 Bit 3: E 23.3 Bit 4: E 23.4 Bit 5: E 23.5 Bit 6: E 23.6 Bit 7: E 23.7	BYTE	R/W	all
_111_ErrorFlags_ext	Generates user error in device	Bit 0: E 24.0 Bit 1: E 24.1 Bit 2: E 24.2 Bit 3: E 24.3 Bit 4: E 24.4 Bit 5: E 24.5 Bit 6: E 24.6 Bit 7: E 24.7	BYTE	R/W	all

3.5.7 PLC parameters

The PLC parameters P355, P356 and P360 can be directly accessed via this group of process data.

Name	Function	Standardisation	Type	Access	Device
_115_PLC_P355_1	PLC INT parameter P355 [-01]	-	INT	R	all
_116_PLC_P355_2	PLC INT parameter P355 [-02]	-	INT	R	all
_117_PLC_P355_3	PLC INT parameter P355 [-03]	-	INT	R	all
_118_PLC_P355_4	PLC INT parameter P355 [-04]	-	INT	R	all
_119_PLC_P355_5	PLC INT parameter P355 [-05]	-	INT	R	all
_120_PLC_P355_6	PLC INT parameter P355 [-06]	-	INT	R	all
_121_PLC_P355_7	PLC INT parameter P355 [-07]	-	INT	R	all
_122_PLC_P355_8	PLC INT parameter P355 [-08]	-	INT	R	all
_123_PLC_P355_9	PLC INT parameter P355 [-09]	-	INT	R	all
_124_PLC_P355_10	PLC INT parameter P355 [-10]	-	INT	R	all
_125_PLC_P356_1	PLC LONG parameter P356 [-01]	-	DINT	R	all
_126_PLC_P356_2	PLC LONG parameter P356 [-02]	-	DINT	R	all
_127_PLC_P356_3	PLC LONG parameter P356 [-03]	-	DINT	R	all
_128_PLC_P356_4	PLC LONG parameter P356 [-04]	-	DINT	R	all
_129_PLC_P356_5	PLC LONG parameter P356 [-05]	-	DINT	R	all
_130_PLC_P360_1	PLC display parameter P360[-01]	-	DINT	R/W	all
_131_PLC_P360_2	PLC display parameter P360[-02]	-	DINT	R/W	all
_132_PLC_P360_3	PLC display parameter P360[-03]	-	DINT	R/W	all
_133_PLC_P360_4	PLC display parameter P360[-04]	-	DINT	R/W	all
_134_PLC_P360_5	PLC display parameter	-	DINT	R/W	all

Name	Function	Standardisation	Type	Access	Device
	P360[-05]				
_135_PLC_Scope_Int_1	PLC scope display value 1	-	INT	R/W	all
_136_PLC_Scope_Int_2	PLC scope display value 2	-	INT	R/W	all
_137_PLC_Scope_Int_3	PLC scope display value 3	-	INT	R/W	all
_138_PLC_Scope_Int_4	PLC scope display value 4	-	INT	R/W	all
_139_PLC_Scope_Bool_1	PLC scope display value 5	-	INT	R/W	all
_140_PLC_Scope_Bool_2	PLC scope display value 6	-	INT	R/W	all

3.6 Languages

3.6.1 Instruction list (AWL / IL)

3.6.1.1 General

Data types

The PLC supports the data types listed below.

Name	Required memory space	Value range
BOOL	1 Bit	0 to 1
BYTE	1 Byte	0 to 255
INT	2 Byte	-32768 to 32767
DINT	4 Byte	-2147483648 to 2147483647
LABEL_ADDRESS	2 Byte	Jump marks

Literal

For greater clarity it is possible to enter constants for all data types in various display formats. The following table gives an overview of all possible variants.

Literal	Example	Number displayed in decimal
Bool	FALSE	0
	TRUE	1
	BOOL#0	0
	BOOL#1	1
Dual (Base 2)	2#01011111	95
	2#0011_0011	51
	BYTE#2#00001111	15
	BYTE#2#0001_1111	31
Oktal (Base 8)	8#0571	377
	8#05_71	377
	BYTE#8#10	8
	BYTE#8#111	73
	BYTE#8#1_11	73
Hexadecimal (Base 16)	16#FFFF	-1
	16#0001_FFFF	131071
	INT#16#1000	4096
	DINT#16#0010_2030	1056816
Integer (Base 10)	10	10
	-10	-10
	10_000	10000
	INT#12	12
	DINT#-100000	-100000
Time	TIME#10s50ms	10.050 seconds
	T#5s500ms	5.5 seconds
	TIME#5.2s	5.2 seconds
	TIME#5D10H15M	5days+10hours+15minutes
	T#1D2H30M20S	1day+2hours+30minutes+20seconds

Comments

It is advisable to provide the sections of the program with comments in order to make the PLC program understandable at a later date. In the application program these comments are marked by starting with the character sequence "(*" and finishing with "*)" as shown in the following examples.

Example:

```
 (* Comment about a program block *)
LD 100 (* Comment after a command *)
ADD 20
```

Jump marks

With the aid of the operators JMP, JMPC or JMPCN whole sections of the program can be bypassed. A jump mark is given as the target address. With the exception of diacritics and „ß" it may contain all letters, the numbers 0 to 9 and underscores; other characters are not permitted. The jump mark is terminated with a colon. This may stand on its own. There may also be further commands after in the same line after the jump mark.

Possible variants may appear as follows:

Example:

```
Jump mark:
LD 20

This_is_a_jumpmark:
ADD 10

MainLoop: LD 1000
```

A further variant is the transfer of a jump mark as a variable. This variable must be defined as type LABLE_ADDRESS in the variable table, then this can be loaded into the variable 'jump marks'. With this, status machines can be created very simply, see below.

Example:

```
LD FirstTime
JMPC AfterFirstTime
(* The label address must be initialized at the beginning *)
LD Address_1
ST Address_Var
LD TRUE
ST FirstTime
AfterFirstTime:

JMP Address_Var

Address_1:
LD Address_2
ST Address_Var
JMP Ende

Address_2:
LD Address_3
ST Address_Var
JMP Ende

Address_3:
LD Address_1
ST Address_Var

Ende:
```

Function call-ups

The Editor supports one form of function call-ups. In the following version, the function CTD is called up via the instance I_CTD. The results are saved in variables. The meaning of the functions used below is described in further detail later in the manual.

Example

```
LD 10000
ST I_CTD.PV
LD LoadNewVar
ST I_CTD.LD
LD TRUE
ST I_CTD.CD
CAL I_CTD
LD I_CTD.Q
ST ResultVar
LD I_CTD.CV
ST CurrentCountVar
```

Bit-wise access to variables

A simplified form is possible for access to a bit from a variable or a process variable.

Command	Description
LD Var1.0	Loads Bit 0 of Var1 into the AE
ST Var1.7	Stores the AE on Bit 7 of Var1
EQ Var1.4	Compares the AE with Bit 4 of Var1

3.6.2 Structured text (ST)

Structured text consists of a series of instruction, which are executed as in plain language ("IF..THEN..ELSE) or in loops (WHILE.. DO).

Example:

```
IF value < 7 THEN
  WHILE value < 8 DO
    value := value + 1;
  END_WHILE;
END_IF;
```

3.6.2.1 Common

Data types in ST

The PLC supports the data types listed below.

Name	Memory required	Value range
BOOL	1 Bit	0 to 1
BYTE	1 Byte	0 to 255
INT	2 Byte	-32768 to 32767
DINT	4 Byte	-2,147,483,648 to 2,147,483,647

Information

For numbers it is advisable to state the data type in order to create an efficient PLC program, e.g.: VarInt := INT#-32768, VarDINT := DINT#-2147483648.

Assignment operator

On the left hand side of an assignment there is an operand (variable, address) to which the value of an expression on the right hand side is assigned with the assignment operator "=".

Example:

```
Var1 := Var2 * 10;
```

After execution of this line, Var1 has ten times the value of Var2.

Call-up of function blocks in ST

A function block is called in ST by writing the name of the instance of the function block and then assigning the values of the parameters in brackets. In the following example a timer is called up with assignment of its parameters IN and PT Then the result variable Q is assigned to the variable A.

The result variable is accessed as in IL with the name of the function block, a following period and the name of the variable.

Example:

```
Timer(IN := TRUE, PT := 300);
A := Timer.Q;
```

Evaluation of expressions

The evaluation of the expression is performed by processing the operators according to certain linking rules. The operator with the strongest link is processed first and then the operator with the next strongest link, etc. until all of the operators have been processed. Operators with links of the same strength are processed from left to right.

The table below shows the ST operators in the order of the strength of their links:

Operation	Symbol	Link strength
Brackets	(Expression)	Strongest
Function call	Function name (parameter list)	
Negated complement formation	NOT	
Multiply Divide Modulus AND	* / MOD AND	
Add Subtract OR XOR	+ - OR XOR	
Compare Equality Inequality	<,>,<=,>= = <>	Light

3.6.2.2 Procedure

Return

The RETURN instruction can be used to jump to the end of the program, for example, depending on a condition.

IF

With the IF instruction, a condition can be tested and instructions carried out depending on this condition.

Syntax:

```
IF <Boolean_Expression1> THEN
  <IF_Instruction>
ELSIF <Boolean_Expression2> THEN
  <ELSIF_Instruction1>
ELSIF <Boolean_Expression n> THEN
  <ELSIF_Instruction n-1>
ELSE
  <ELSE_Instruction>}
END_IF;
```

The part in the curly brackets {} is optional. If <Boolean_Expression1> is TRUE, then only the <IF_Instructions> are executed and none of the other instructions.. Otherwise, starting with <Boolean_Expression2>, the boolean expressions are evaluated in sequence until one of the expressions is TRUE. Then, only the expressions following this boolean expression and before the next ELSE or ELSIF are evaluated. If none of the boolean expressions is TRUE, only the <ELSE_Instructions> are evaluated.

Example:

```
IF temp < 17 THEN
  Bool1 := TRUE;
ELSE
  Bool2 := FALSE;
END_IF;
```

CASE

With the CASE instruction, several conditional instructions with the same condition variables can be combined into a construct.

Syntax:

```
CASE <Var1> OF
  <Value1>: <Instruction 1>
  <Value2>: <Instruction 2>
  <Value3, Value4, Value5: <Instruction 3>
  <Value6 .. Value10 : <Instruction 4>
  ...
  <Value n>: <Instruction n>
ELSE <ELSE-Instruction>
END_CASE;
```

A CASE instruction is processed according to the following pattern:

- If the variable in <Var1> has the value <Value i>, the instruction <Instruction i> is executed.
- If <Var 1> does not have any of the stated values, the <ELSE instruction> is executed.
- If the same instruction is to be executed for several values of the variable, these values can be written separately in sequence, separated with commas as the condition of the common instruction.
- If the same instruction is to be executed for a range of values of the variable, the initial value and the end value can be written separated by a colon as the condition for the common instruction.

Example:

```
CASE INT1 OF
 1, 5:
   BOOL1 := TRUE;
   BOOL3 := FALSE;
 2:
   BOOL2 := FALSE;
   BOOL3 := TRUE;
 10..20:
   BOOL1 := TRUE;
   BOOL3:= TRUE;
 ELSE
   BOOL1 := NOT BOOL1;
   BOOL2 := BOOL1 OR BOOL2;
END_CASE;
```

FOR loop

Repetitive processes can be programmed with the FOR loop.

Syntax:

```
FOR <INT_Var> := <INIT_VALUE> TO <END_VALUE> {BY <STEP>} DO
 <Instruction>
END_FOR;
```

The part in the curly brackets {} is optional. The <Instructions> are executed as long as the counter <INT-Var> is not larger than the <END_VALUE>. This is checked before the execution of the <Instructions> so that the <Instructions> are never executed if the <INIT_VALUE> is larger than the <END_VALUE>. Whenever the <Instructions> are executed, the <INIT-Var> is increased by a <Step size>. The step size can have any integer value. If this is missing, it is set to 1. The loop must terminate, as <INT_Var> is larger.

Example:

```
FOR Zaehler :=1 TO 5 BY 1 DO
  Var1 := Var1 * 2;
END_FOR;
```

REPEAT loop

The REPEAT loop is different from the WHILE loop in that the termination condition is only tested after the loop has been executed. As a result, the loop must be run through at least once, regardless of the termination condition.

Syntax:

```
REPEAT
  <Instruction>
UNTIL
  <Boolean Expression>
END_REPEAT;
```

The <Instructions> are executed until the <Boolean Expression> is TRUE. If the <Boolean Expression> is TRUE with the first evaluation, the <Instructions> are executed exactly once. If the <Boolean Expression> is never TRUE, the <Instructions> will be executed endlessly, which will create a runtime error.

Information

The programmer must ensure that no endless loops are created by changing the condition in the instruction part of the loop, for example a counter which counts upwards or downwards.

Example:

```
REPEAT
  Var1 := Var1 * 2;
  Count := Count - 1;
UNTIL
  Count = 0
END_REPEAT
```

WHILE loop

The WHILE loop can be used in the same way as the FOR loop, with the difference that the termination condition can be any boolean expression. This means that a condition is stated, which, if it is true, will result in the execution of the loop.

Syntax:

```
WHILE <Boolean Expression> DO
  <Instructions>
END_WHILE;
```

The <Instructions> are executed repeatedly for as long as the <Boolean_Expression> is TRUE. If the <Boolean_Expression> is FALSE in the first evaluation, the <Instructions> will never be executed. If the <Boolean_Expression> is never FALSE, the <Instructions> will be repeated endlessly.

Information

The programmer must ensure that no endless loops are created by changing the condition in the instruction part of the loop, for example a counter which counts upwards or downwards.

Example:

```
WHILE Count <> 0 DO
  Var1 := Var1*2;
  Count := Count - 1;
END_WHILE
```

Exit

If the EXIT instruction occurs in a FOR, WHILE or REPEAT loop, the innermost loop will be terminated, regardless of the termination condition.

3.7 Jumps

3.7.1 JMP

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X

Unconditional jump to a jump mark.

Example AWL:

```
JMP NextStep (* Unconditional jump to NextStep *)
ADD 1

NextStep:
ST Value1
```

3.7.2 JMPC

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X

Conditional jump to a jump point If AE = TRUE, the command JMPC jumps to the stated jump point.

Example AWL:

```
LD 10
JMPC NextStep (* AE = TRUE - program jumps *)
ADD 1

NextStep:
ST Value1
```

3.7.3 JMPCN

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X

Conditional jump to a jump point JMPCN jumps if the AE register = FALSE. Otherwise the program continues with the next instruction.

Example AWL:

```
LD 10
JMPCN NextStep (* AE = TRUE - program does not jump *)
ADD 1

NextStep:
ST Value1
```

3.8 Type conversion

3.8.1 BOOL_TO_BYTE

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type	X			

Converts the data type of the AE from BOOL to BYTE. If the AE is FALSE, the accumulator is converted to 0. If the AE is TRUE, the accumulator is converted to 1.

Example in IL:

```
LD TRUE
BOOL_TO_BYTE (* AE = 1 *)
```

Example in ST:

```
Ergebnis := BOOL_TO_BYTE(TRUE); (* Result = 1 *)
```

3.8.2 BYTE_TO_BOOL

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type		X		

Converts the data type from BYTE to BOOL. As long as BYTE is not equal to zero, this always gives the conversion result TRUE.

Example in IL:

```
LD 10
BYTE_TO_BOOL (* AE = TRUE *)
```

Example in ST:

```
Ergebnis := BYTE_TO_BOOL(10); (* Result = TRUE *)
```

3.8.3 BYTE_TO_INT

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type		X		

Converts the data type from BYTE to INT. The BYTE is copied into the Low component of the INT and the High component of INT is set to 0.

Example in IL:

```
LD 10
BYTE_TO_INT (* Accumulator = 10 *)
```

Example in ST:

```
Ergebnis := BYTE_TO_INT(10); (* Result = 10 *)
```

3.8.4 DINT_TO_INT

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type				X

Converts the data type from DINT to INT. The High component of the DINT value is not transferred.

Example in IL:

```
LD 200000
DINT_TO_INT      (* Accumulator = 3392 *)

LD DINT# -5000
DINT_TO_INT      (* Accumulator = -5000 *)

LD DINT# -50010
DINT_TO_INT      (* Accumulator = 15526 *)
```

Example in ST:

```
Ergebnis := DINT_TO_INT(200000); (* Result = 3392 *)
Ergebnis := DINT_TO_INT(-5000); (* Result = -5000 *)
Ergebnis := DINT_TO_INT(-50010); (* Result = 15526 *)
```

3.8.5 INT_TO_BYTE

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type			X	

Converts the data type from INT to BYTE. Here, the High component of the INT value is not transferred. Prefixes are lost as the BYTE type does not have prefixes.

Example in IL:

```
LD 16#5008
INT_TO_BYTE (* Accumulator = 8 *)
```

Example in ST:

```
Ergebnis := INT_TO_BYTE(16#5008); (* Result = 8 *)
```

3.8.6 INT_TO_DINT

	SK 5xxP	SK 54xE	SK 53xE SK 52xE	On/On+	SK 2xxE	SK 2xxE-FDS	SK 180E SK 190E	SK 155E-FDS SK 175E-FDS
Availability	X	X	X	X	X	X	X	X

	BOOL	BYTE	INT	DINT
Data type			X	

Converts the data type from INT to DINT. The INT is copied into the Low component of the DINT and the High component of the DINT is set to 0.

Example in IL:

```
LD 10
INT_TO_DINT (* Accumulator = 10 *)
```

Example in ST:

```
Ergebnis := INT_TO_DINT(10); (* Result = 10 *)
```

3.9 PLC Error messages

Error messages cause the device to switch off, in order to prevent a device fault. With PLC error messages execution by the PLC is stopped and the PLC goes into the status "PLC Error". With other error messages the PLC continues operation. The PLC restarts automatically after the error has been acknowledged.

The PLC continues to operate with PLC User Fault 23.X and 24.X!

SimpleBox		Fault Text in the ParameterBox	Cause Remedy
Group	**Details in P700[-01] / P701**		
E022	**22.0**	**No PLC program**	The PLC has been started but there is no PLC program in the device - Load PLC program into the FI
	22.1	**PLC program is faulty**	The checksum check via the PLC program produced an error. - Restart the device (Power ON) and try again - Alternatively, reload PLC program
	22.2	**Incorrect jump address**	Program error, behaviour as for Error 22.1
	22.3	**Stack overflow**	More than 6 bracket levels were opened during the run time of the program - Check the program for run time errors
	22.4	**Max. PLC cycles exceeded**	The stated maximum cycle time for the PLC program was exceeded - Change the cycle time or check the program
	22.5	**Unknown command code**	A command code in the program cannot be executed because it is not known. - Program error, behaviour as for Error 22.1 - Version of the PLC and the NORDCON version do not match
	22.6	**PLC write access**	The program content has been changed while the PLC program was running
	22.9	**PLC Error**	The cause of the fault cannot be precisely determined - Behaviour as in Error 22.1
E023/ E024	**23.0 to 23.7**	**PLC User Fault 1 to 8**	This error can be triggered by the PLC program in order to externally indicate problems in the execution of the PLC program. Triggered by writing the process variable "ErrorFlags".
	24.0 to 24.7	**PLC User Fault 9 to 16**	

4 Parameters

The relevant device parameters for PLC functionality are described in detail in the manual for the relevant frequency inverter or motor starter.

5 Appendix

5.1 Service and commissioning information

In case of problems, e.g. during commissioning, please contact our Service department:

☎ +49 4532 289-2125

Our Service department is available 24/7 and can help you best if you have the following information about the device and its accessories to hand:

- Type designation,
- Serial number,
- Firmware version

5.2 Documents and software

Documents and software can be downloaded from our website www.nord.com .

Other applicable documents and further information

Documentation	Contents
BU 0155	Manual for field distributor motor starter NORDAC *LINK* **SK 180E / SK 190E**
BU 0180	Manual for frequency inverter NORDAC *BASE* **SK 180E / SK 190E**
BU 0200	Manual for frequency inverter NORDAC *FLEX* **SK 200E .. SK 235E**
BU 0250	Manual for frequency inverter NORDAC *LINK* **SK 250E-FDS .. SK 280E-FDS**
BU 0500	Manual for frequency inverter NORDAC *PRO* **SK 500E .. SK 535E**
BU 0505	Manual for frequency inverter NORDAC *PRO* **SK 540E .. SK 545E**
BU 0600	Manual for frequency inverter NORDAC *PRO* **SK 500P .. SK 550P**
BU 0800	Manual for frequency inverter NORDAC *ON/ON+* **SK 300P**
BU 0000	Manual for use of NORDCON software
BU 0040	Manual for use of NORD parameterisation units

Software

Software	Description
NORDCON	Parametrisation and diagnostic software

5.3 Abbreviations

• **AE**	Actual event
• **AIN**	Analogue input
• **AOUT**	Analogue output
• **AWL**	Application list (also IL)
• **COB-ID**	Communication Object Identifier
• **DI / DIN**	Digital input
• **DO/ DOUT**	Digital output
• **E/A or I/O**	Input /Output
• **EEPROM**	Non-volatile memory
• **EMC**	Electromagnetic compatibility
• **FB**	Function block
• **FI**	Frequency inverter
• **HSW**	Main setpoint
• **IL**	Instruction List (see also AWL)
• **ISD**	Field current (current vector control)
• **LED**	Light-emitting diode
• **MC**	Motion Control
• **NSW**	Auxiliary setpoint
• **P**	Parameter set dependent parameter, i.e. A parameter which can be assigned different functions or values in each of the 4 parameter sets.
• **P-BOX**	ParameterBox
• **PDO**	Process Data Object
• **PLC**	PLC (Programmable Logic Controller)
• **S**	Supervisor parameter, i.e. A parameter which is only visible if the correct Supervisor Code is entered in parameter **P003**
• **SW**	Software version (see parameter **P707**)
• **STW**	Control word
• **ZSW**	Status word

Key word index

NORD DRIVESYSTEMS Group

Headquarters and Technology Centre
in Bargteheide, close to Hamburg

Innovative drive solutions
for more than 100 branches of industry

Mechanical products
parallel shaft, helical gear, bevel gear and worm gear units

Electrical products
IE2/IE3/IE4 motors

Electronic products
centralised and decentralised frequency inverters,
motor starters and field distribution systems

7 state-of-the-art production plants
for all drive components

Subsidiaries and sales partners
in 98 countries on 5 continents
provide local stocks, assembly, production,
technical support and customer service

More than 4,000 employees throughout the world
create customer oriented solutions

www.nord.com/locator

Headquarters:

Getriebebau NORD GmbH & Co. KG
Getriebebau-Nord-Straße 1
22941 Bargteheide, Germany
T: +49 (0) 4532 / 289 - 0
F: +49 (0) 4532 / 289 - 22 53
info@nord.com, www.nord.com

Member of the NORD DRIVESYSTEMS Group

NORD® DRIVESYSTEMS

6075502 / 0321