YASKAWA

VIPA SPEED7 Library

OPL_SP7-LIB | SW90JS0MA V10.002 | Manual

HB00 | OPL_SP7-LIB | SW90JS0MA V10.002 | en | 18-28

Block library - Standard

www.vipa.com/en/service-support/manuals

VIPA CONTROLS

VIPA GmbH
Ohmstr. 4
91074 Herzogenaurach
Telephone: +49 9132 744-0
Fax: +49 9132 744-1864
Email: info@vipa.com
Internet: www.vipa.com

SW90JS0MA_000_OPL_SP7-LIB_Standard,2,EN -   © 2018

Table of contents

1 General

1.1 Copyright © VIPA GmbH

CE Conformity Declaration

Hereby, VIPA GmbH declares that the products and systems are in compliance with the essential requirements and other relevant provisions. Conformity is indicated by the CE marking affixed to the product.

Conformity Information

For more information regarding CE marking and Declaration of Conformity (DoC), please contact your local VIPA customer service organization.

Information product support

Contact your local VIPA Customer Service Organization representative if you wish to report errors or questions regarding the contents of this document. If you are unable to locate a customer service centre, contact VIPA as follows:

VIPA GmbH, Ohmstraße 4, 91074 Herzogenaurach, Germany

Telefax: +49 9132 744-1204

EMail: documentation@vipa.de

Technical support

Contact your local VIPA Customer Service Organization representative if you encounter problems with the product or have questions regarding the product. If you are unable to locate a customer service centre, contact VIPA as follows:

VIPA GmbH, Ohmstraße 4, 91074 Herzogenaurach, Germany

Tel.: +49 9132 744-1150 (Hotline)

EMail: support@vipa.de

1.2 About this manual

Objective and contents

The manual describes the block library *'Standard'* from VIPA:

- It contains a description of the structure, project implementation and usage in several programming systems.
- The manual is targeted at users who have a background in automation technology.
- The manual is available in electronic form as PDF file. This requires Adobe Acrobat Reader.
- The manual consists of chapters. Every chapter provides a self-contained description of a specific topic.
- The following guides are available in the manual:
  - An overall table of contents at the beginning of the manual
  - References with pages numbers

Icons Headings

Important passages in the text are highlighted by following icons and headings:

DANGER!
Immediate or likely danger. Personal injury is possible.

CAUTION!
Damages to property is likely if these warnings are not heeded.

Supplementary information and useful tips.

2 Important notes

2.1 General

In the following, you will find important notes, which must always be observed when using the blocks.

2.2 Internally used blocks

CAUTION!
The following blocks are used internally and must not be overwritten! The direct call of an internal block leads to errors in the corresponding instance DB! Please always use the corresponding function for the call.

FC/SFC	Designation	Description
FC/SFC 192	CP_S_R	is used internally for FB 7 and FB 8
FC/SFC 196	AG_CNTRL	is used internally for FC 10
FC/SFC 200	AG_GET	is used internally for FB/SFB 14
FC/SFC 201	AG_PUT	is used internally for FB/SFB 15
FC/SFC 202	AG_BSEND	is used internally for FB/SFB 12
FC/SFC 203	AG_BRCV	is used internally for FB/SFB 13
FC/SFC 204	IP_CONF	is used internally for FB 55 IP_CONF
FC/SFC 205	AG_SEND	is used internally for FC 5 AG_SEND
FC/SFC 206	AG_RECV	is used internally for FC 6 AG_RECV
FC/SFC 253	IBS_ACCESS	is used internally for SPEED bus INTERBUS masters
SFB 238	EC_RWOD	is used internally for EtherCAT Communication
SFB 239	FUNC	is used internally for FB 240, FB 241

3 Include library

Block library *'Standard'*

The block library can be found for download in the *'Service/Support'* area of www.vipa.com at *'Downloads ➔ VIPA Lib'* as *'Block library Standard - SW90JS0MA'*. The library is available as packed zip file. As soon as you want to use these blocks you have to import them into your project.

The following block libraries are available

File	Description
Standard_S7_V0001.zip	■ Block library for Siemens SIMATIC Manager. ■ For use in CPUs from VIPA or S7-300 CPUs from Siemens.
Standard_TIA_V0002.zip	■ Block library for Siemens TIA Portal V14. ■ For use in CPUs from VIPA or S7-300 CPUs from Siemens.

3.1 Integration into Siemens SIMATIC Manager

Overview

The integration into the Siemens SIMATIC Manager requires the following steps:

1. Load ZIP file
2. "Retrieve" the library
3. Open library and transfer blocks into the project

Load ZIP file

➔ Navigate on the web page to the desired ZIP file, load and store it in your work directory.

Retrieve library

1. Start the Siemens SIMATIC Manager with your project.
2. Open the dialog window for ZIP file selection via *'File ➔ Retrieve'*.
3. Select the according ZIP file and click at [Open].
4. Select a destination folder where the blocks are to be stored.
5. Start the extraction with [OK].

Open library and transfer blocks into the project

1. Open the library after the extraction.
2. Open your project and copy the necessary blocks from the library into the directory "blocks" of your project.

   ⇨ Now you have access to the VIPA specific blocks via your user application.

Are FCs used instead of SFCs, so they are supported by the VIPA CPUs starting from firmware 3.6.0.

3.2 Integration into Siemens TIA Portal

Overview

The integration into the Siemens TIA Portal requires the following steps:

1. Load ZIP file
2. Unzip the Zip file
3. "Retrieve" the library
4. Open library and transfer blocks into the project

Load ZIP file

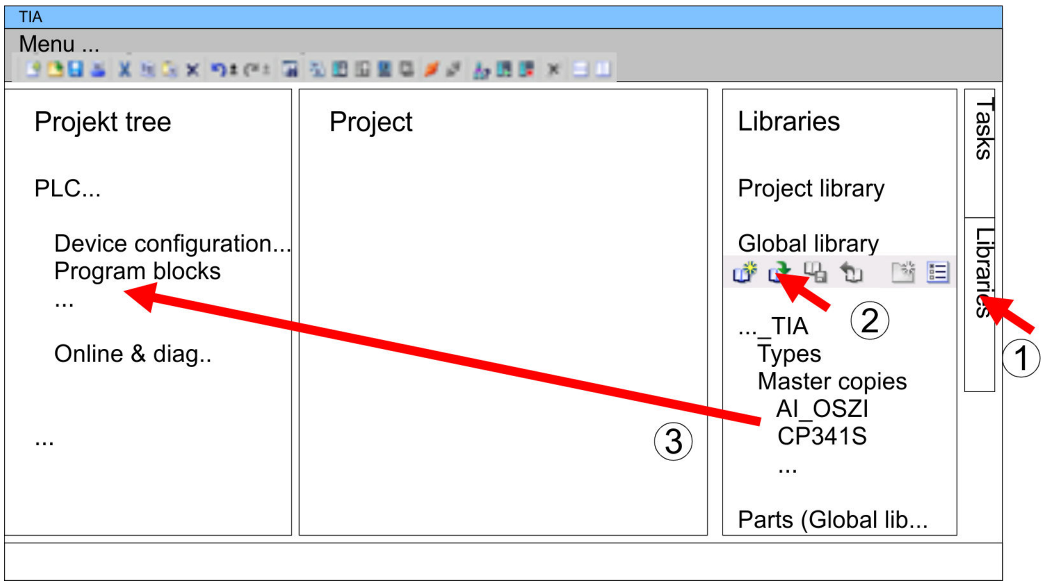
1. Navigate on the web page to the ZIP file, that matches your version of the program.
2. Load and store it in your work directory.

Unzip the Zip file

Unzip the zip file to a work directory of the Siemens TIA Portal with your unzip application.

Open library and transfer blocks into the project

1. Start the Siemens TIA Portal with your project.
2. Switch to the *Project view*.
3. Choose "Libraries" from the task cards on the right side.
4. Click at "Global libraries".
5. Click at "Open global libraries".
6. Navigate to your work directory and load the file ..._TIA.al1x.



7. Copy the necessary blocks from the library into the "Program blocks" of the *Project tree* of your project. Now you have access to the VIPA specific blocks via your user application.

4 Standard

4.1 Converting

4.1.1 FB 80 - LEAD_LAG - Lead/Lag Algorithm

Description

The Lead/Lag Algorithm LEAD_LAG function block allows signal processing to be done on an analog variable. An output *OUT* is calculated based on an input *IN* and the specified gain *GAIN*, lead *LD_TIME*, and lag *LG_TIME* values. The gain value must be greater than zero. The LEAD_LAG algorithm uses the following equation:

$$\text{and } OUT = \left[\frac{LG\_TIME}{LG\_TIME + SAMPLE\_T}\right] PREV\_OUT + GAIN\left[\frac{LD\_TIME + SAMPLE\_T}{LG\_TIME + SAMPLE\_T}\right] IN - GAIN\left[\frac{LD\_TIME}{LG\_TIME + SAMPLE\_T}\right] PREV\_IN$$

Typically, LEAD_LAG is used in conjunction with loops as a compensator in dynamic feed-forward control. LEAD_LAG consists of two parts. Phase lead shifts the phase of the function block's output so that it leads the input whereas phase lag shifts the output so that it lags the input. Because the lag operation is equivalent to an integration, it can be used as a noise suppressor or a low-pass filter. A lead operation is equivalent to a differentiation and is thus a high-pass filter. LEAD_LAG combined can cause the output phase to lag input at low frequency, and to lead input at high frequency, and can thus be used as a band-pass filter.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function block is executed without error
IN	Input	REAL	I, Q, M, D, L, P, constant	The input value of the current sample period to be processed
SAMPLE_T	Output	INT	I, Q, M, D, L, P, constant	Sample time
OUT	Output	REAL	I, Q, M, D, L, P, constant	The result of the LEAD_LAG operation
ERR_CODE	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000
LD_TIME	Static	REAL	I, Q, M, D, L, P, constant	Lead time in minutes
LG_TIME	Static	REAL	I, Q, M, D, L, P, constant	Lag time in minutes
GAIN	Static	REAL	I, Q, M, D, L, P, constant	Gain as % / % (the ratio of the change in output to the change in input as a steady state).
PREV_IN	Static	REAL	I, Q, M, D, L, P, constant	Previous input
PREV_OUT	Static	REAL	I, Q, M, D, L, P, constant	Previous output

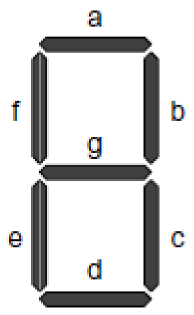
Error Information

If *GAIN* is less than or equal to 0, the function block is not executed. The signal state of *ENO* is set to 0 and *ERR_CODE* is set equal to W#16#0009.

4.1.2 FC 93 - SEG - Seven Segment Decoder

Description

The Seven Segment Decoder SEG function converts each of the four hexadecimal digits in the designated source data word *IN* into four equivalent 7-segment display codes and writes it to the output destination double word *OUT*. The Figure below shows the relationship between the input hex digits and the output bit patterns.



Parameters

Digit	– g f e d c b a	Display
0 0 0 0	0 0 1 1 1 1 1 1	0
0 0 0 1	0 0 0 0 0 1 1 0	1
0 0 1 0	0 1 0 1 1 0 1 1	2
0 0 1 1	0 1 0 0 1 1 1 1	3
0 1 0 0	0 1 1 0 0 1 1 0	4
0 1 0 1	0 1 1 0 1 1 0 1	5
0 1 1 0	0 1 1 1 1 1 0 1	6
0 1 1 1	0 0 0 0 0 1 1 1	7
1 0 0 0	0 1 1 1 1 1 1 1	8
1 0 0 1	0 1 1 0 0 1 1 1	9
1 0 1 0	0 1 1 1 0 1 1 1	A
1 0 1 1	0 1 1 1 1 1 0 0	b
1 1 0 0	0 0 1 1 1 0 0 1	C
1 1 0 1	0 1 0 1 1 1 1 0	d
1 1 1 0	0 1 1 1 1 0 0 1	E
1 1 1 1	0 1 1 1 0 0 0 1	F

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN	Input	WORD	I, M, D, P, or constant	Source data word in four hexadecimal digits
OUT	Output	DWORD	Q, M, D, L, P	Destination bit pattern in four bytes

Error Information

This function does not detect any error conditions.

4.1.3 FC 94 - ATH - ASCII to Hex

Description

The ASCII to Hex (ATH) function converts the ASCII character string pointed to by *IN* into packed hexadecimal digits and stores these in the destination table pointed to by *OUT*. Since 8 bits are required for the ASCII character and only 4 bits for the hexadecimal digit, the output word length is only half of the input word length. The ASCII characters are converted and placed into the hexadecimal output in the same order as they are read in. If there is an odd number of ASCII characters, the hexadecimal digit is padded with zeros in the right-most nibble of the last converted hexadecimal digit.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN	Input	Pointer*	I, Q, M, D, L	Points to the starting location of an ASCII string
N	Input	INT	I, Q, M, L, P	Number of ASCII input characters to be converted
RET_VAL	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000
OUT	Output	Pointer*	Q, M, D, L	Points to the starting location of the table
*) Double word pointer format for area-crossing register indirect addressing				

Error Information

If any ASCII character is found to be invalid, it is converted as 0. The signal state of *ENO* is set to 0 and *RET_VAL* is set equal to W#16#0007.

4.1.4 FC 95 - HTA - Hex to ASCII

Description

The Hex to ASCII (HTA) function converts packed hexadecimal digits, pointed to by *IN*, and stores them in the destination string pointed to by *OUT*. Since 8 bits are required for the character and only 4 bits for the hex digit, the output word length is two times that of the input word length. Each nibble of the hexadecimal digit is converted into a character in the same order as they are read in (left-most nibble of a hexadecimal digit is converted first, followed by the right-most nibble of that same digit).

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN	Input	Pointer *	I, Q, M, D	Points to the starting location of the hexadecimal digit string

Parameter	Declaration	Data Type	Memory Area	Description
N	Input	WORD	I, Q, M, L, P	Number of hex input bytes to be converted
OUT	Output	Pointer *	Q, M, D, L	Points to the starting location of the destination table
*) Double word pointer format for area-crossing register indirect addressing				

Error Information This function does not detect any error conditions.

4.1.5 FC 96 - ENCO - Encode Binary Position

Description The Encode Binary Position ENCO function converts the contents of *IN* to the 5-bit binary number corresponding to the bit position of the right-most set bit in *IN* and returns the result as the function's value. If *IN* is either 0000 0001 or 0000 0000, a value of 0 is returned.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN	Input	DWORD	I, M, D, L, P, or constant	Value to be encoded
RET_VAL	Input	INT	Q, M, D, L, P	Value returned (contains 5-bit binary number)

Error Information This function does not detect any error conditions.

4.1.6 FC 97 - DECO - Decode Binary Position

Description The Decode Binary Position DECO function converts a 5-bit binary number (0 – 31) from input *IN* to a value by setting the corresponding bit position in the function's return value. If *IN* is greater than 31, a modulo 32 operation is performed to get a 5-bit binary number.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN	Input	WORD	I, M, D, L, P, constant	Variable to decode
RET_VAL	Output	DWORD	Q, M, D, L, P	Value returned

Error Information

This function does not detect any error conditions.

4.1.7 FC 98 - BCDCPL - Tens Complement

Description

The Tens Complement BCDCPL function returns the Tens complement of a 7-digit BCD number *IN*. The mathematical formula for this operation is the following:

10000000 (in BCD) - 7digit BCD value = Tens complement value (in BCD)

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN	Input	DWORD	I, M, D, L, P, constant	7-digit BCD number
RET_VAL	Output	DWORD	Q, M, D, L, P	Value returned

Error Information

This function does not detect any error conditions.

4.1.8 FC 99 - BITSUM - Sum Number of Bits

Description

The Sum Number of Bits BITSUM function counts the number of bits that are set to a value of 1 in the input *IN* and returns this as the function’s value.

Parameter

Parameter	Deklaration	Datentyp	Speicherbereich	Beschreibung
EN	Input	BOOL	I, Q, M, D, L	Enable input with signal state of 1 activates the box
ENO	Output	BOOL	I, Q, M, D, L	Enable output has a signal state of 1 if the function is executed without error
IN	Input	DWORD	I, M, D, L, P, constant	Variable to count bits in
RET_VAL	Output	INT	Q, M, D, L, P	Value returned

Error Information

This function does not detect any error conditions.

4.1.9 FC 105 - SCALE - Scaling Values

Description

The Scaling Values SCALE function takes an integer value *IN* and converts it to a real value in engineering units scaled between a low and a high limit *LO_LIM* and *HI_LIM*. The result is written to *OUT*. The SCALE function uses the equation:

$$OUT = [((FLOAT\ (IN) - K1) / (K2 - K1)) \cdot (HI\_LIM - LO\_LIM)] + LO\_LIM$$

The constants K1 and K2 are set based upon whether the input value is *BIPOLAR* or *UNIPOLAR*.

- *BIPOLAR*:
  – The input integer value is assumed to be between -27648 and 27648, therefore, K1 = -27648,0 and K2 = +27648,0.
- *UNIPOLAR*:
  – The input integer value is assumed to be between 0 and 27648, therefore, K1 = 0,0 and K2 = +27648,0.

If the input integer value is greater than K2, the output *OUT* is clamped to *HI_LIM*, and an error is returned. If the input integer value is less than K1, the output *OUT* is clamped to *LO_LIM*, and an error is returned. Reverse scaling can be obtained by programming *LO_LIM* > *HI_LIM*. With reverse scaling, the value of the output decreases as the value of the input increases.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	INPUT	BOOL	I, Q, M, D, L	■ Enable – TRUE: activates the function – FALSE: deactivates the function
ENO	OUTPUT	BOOL	I, Q, M, D, L	■ Status – TRUE: function executed without error
IN	INPUT	INT	I, Q, M, D, L, constant	The input value to be scaled to a REAL value in engineering units
HI_LIM	INPUT	REAL	I, Q, M, D, L, P, constant	Upper limit in engineering units
LO_LIM	INPUT	REAL	I, Q, M, D, L, P, constant	Lower limit in engineering units
BIPOLAR	INPUT	BOOL	I, Q, M, D, L	A signal state of 1 indicates the input value is bipolar, a signal state of "0" indicates unipolar
OUT	OUTPUT	REAL	I, Q, M, D, L, P	The result of the scale conversion
RET_VAL	INPUT	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000

Error information

- If the input integer value is greater than K2, the output *OUT* is clamped to *HI_LIM*, and an error is returned.
- If the input integer value is less than K1, the output *OUT* is clamped to *LO_LIM*, and an error is returned.
- The signal state of *ENO* is set to FALSE and *RET_VAL* is set equal to W#16#0008.

4.1.10 FC 106 - UNSCALE - Unscaling Values

Description

The Unscaling Values UNSCALE function takes a real input value *IN* in engineering units scaled between a low and a high limit *LO_LIM* and *HI_LIM* and converts it to an integer value. The result is written to *OUT*. The UNSCALE function uses the equation:

$$OUT = [((IN - LO\_LIM) / (HI\_LIM - LO\_LIM)) \cdot (K2 - K1)] + K1$$

and sets the constants K1 and K2 based upon whether the input value is *BIPOLAR* or *UNIPOLAR*.

- *BIPOLAR*:
  - The input integer value is assumed to be between -27648 and 27648, therefore, K1 = -27648.0 and K2 = +27648.0.
- *UNIPOLAR*:
  - The input integer value is assumed to be between 0 and 27648, therefore, K1 = 0.0 and K2 = +27648.0.

If the input value is outside the *LO_LIM* and *HI_LIM* range, the output *OUT* is clamped to the nearer of either the low limit or the high limit of the specified range for its type (*BIPOLAR* or *UNIPOLAR*), and an error is returned.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	Input	BOOL	I, Q, M, D, L	■ Enable – TRUE: activates the function – FALSE: deactivates the function
ENO	Output	BOOL	I, Q, M, D, L	■ Status – TRUE: function executed without error
IN	Input	REAL	I, Q, M, D, L, P, constant	The input value to be unscaled to an integer value
HI_LIM	Input	REAL	I, Q, M, D, L, P, constant	Upper limit in engineering units
LO_LIM	Input	REAL	I, Q, M, D, L, P, constant	Lower limit in engineering units
BIPOLAR	Input	BOOL	I, Q, M, D, L	A signal state of 1 indicates the input value is bipolar and a signal state of "0" indicates unipolar
OUT	Output	INT	I, Q, M, D, L, P	The result of the scale conversion
RET_VAL	Output	WORD	I, Q, M, D, L, P	Returns a value of W#16#0000 if the instruction executes without error; see Error Information for values other than W#16#0000

Error Information

If the input real value is outside the *LO_LIM* and *HI_LIM* range the output *OUT* is clamped to the nearer of either the low limit or the high limit of the specified range for its type (*BIPOLAR* or *UNIPOLAR*), and an error is returned. The signal state of *ENO* is set to 0 and *RET_VAL* is set equal to W#16#0008.

4.1.11 FC 108 - RLG_AA1 - Issue an Analog Value

Description

The function RLG_AA1 (Issue an Analog Value) transforms an Input Value *XE* (Fixed Point Number) into an output value for an analog output module in accordance with the nominal range between *OGR* and *UGR*. If the nominal range is exceeded, an error message is displayed.

Parameter	Datentyp	Speicherbereich	Beschreibung
XE	INT	I, Q, M, L, D, constant	Input value *XE* as a fixed point number
BG	INT	I, Q, M, L, D, constant	Specify the module address
KNKT	WORD	I, Q, M, L, D, constant	Channel number KN Channel type KT
OGR	INT	I, Q, M, L, D, constant	Upper limit of the input value *XE*
UGR	INT	I, Q, M, L, D, constant	Lower limit of the input value *XE*
FEH	BOOL	I, Q, M, L, D	Error bit
BU	BOOL	I, Q, M, L, D	Range excess

Differences between S5 and S7

- The BG parameter
  – There is no address check. The range is the whole P area.

This function is only used to convert the FB251 of an existing S5 program of an S5 CPU 941 to 944 to a function of an S7 program for the S7-400 programmable controller.

4.1.12 FC 109 - RLG_AA2 - Write Analog Value 2

Description

The function RLG_AA2 (Issue an Analog Value) transforms an Input Value *XE* (Floating Point Number) into an output value for an analog output module in accordance with the nominal range between *OGR* and *UGR*. If the nominal range is exceeded, an error message is displayed.

Parameter	Data Type	Memory Area	Description
XE	REAL	I, Q, M, L, D, constant	Input value *XE* as a floating point number
BG	INT	I, Q, M, L, D, constant	Specify the module address
P_Q	WORD	I, Q, M, L, D, constant	Peripheriebereich normal/erweitert
KNKT	WORD	I, Q, M, L, D, constant	Channel number KN Channel type KT
OGR	REAL	I, Q, M, L, D, constant	Upper limit of the input value *XE*
UGR	REAL	I, Q, M, L, D, constant const.	Lower limit of the input value *XE*
FEH	BOOL	I, Q, M, L, D	Error bit
BU	BOOL	I, Q, M, L, D	Range excess

Differences between S5 and S7

- The BG parameter
  – There is no address check. The range is the whole P area.
- In S7, no value is assigned to the parameter *P_Q*.
- A process image of the S5 I/O areas P/Q/IM3/IM4 is made in the S7 I/O area. You must assign the I/O area in the configuration table.

This function is only used to convert the FB41 of an existing S5 program of an S5 CPU 928B, 945 or 948 to a function of an S7 program for the S7-400 programmable controller.

4.1.13 FC 110 - PER_ET1 - Read/Write Ext. Per. 1

Description

The function PER_ET1 (Reading and Writing for Expanded Peripheries) transfers either a peripheral area into a CPU-internal area or vice-versa (depending on the parameter assignment). In this way, input bytes can be read from, and output bytes written to, the expanded I/O. If a data block is selected as an internal area, the block must have been set up by the user with the necessary length prior to calling up the function.

Parameter	Data Type	Memory Area	Description
PBIB	WORD	I, Q, M, L, D, constant	Specify the areas to be processed
ANF	INT	I, Q, M, L, D, constant	Beginning of the internal area
ANEN	WORD	I, Q, M, L, D, constant	Beginning and end of the block on the interface module
E_A	BOOL	I, Q, M, L, D, constant	Transfer direction
PAFE	BOOL	I, Q, M, L, D	Parameter assignment error

Differences between S5 and S7

- The *PBIB* parameter
  – In S7, the I/O area is assigned values as follows:

	S5		S7
P area	0 to 255	P area	0 to 255
Q area	0 to 255	P area	256 to 511
IM3 area	0 to 255	P area	512 to 767
IM4 area	0 to 255	P area	768 to 1023
DB	0 to 255	DB	0 to 255
DX	0 to 255	DB	256 to 511
M	0 to 199	M	0 to 199
S		Error message: "Invalid range"	

- A process image of the S5 I/O areas P/Q/IM3/IM4 is made in the S7 I/O area. You must assign the I/O area in the configuration table.

This function is only used to convert the FB196 of an existing S5 program of an S5 CPU 95U, 103, 941 to 944, 945, 928B, 948 to a function of an S7 program for the S7-300/400 programmable controller.

4.1.14 FC 111 - PER_ET2 - Read/Write Ext. Per. 2

Description

The function PER_ET2 (Reading and Writing for Expanded Peripheries) transfers either a peripheral area into a CPU-internal area or vice-versa (depending on the parameter assignment). In this way, input bytes can be read from, and output bytes written to, the expanded I/O. If a data block is selected as an internal area, the block must have been set up by the user with the necessary length prior to calling up the function.

Differences between S5 and S7:

- The *PBIB* parameter (defined in DB)
  – In S7, the I/O area is assigned values as follows:

	S5		S7
P area	0 to 255	P area	0 to 255
Q area	0 to 255	P area	256 to 511
IM3 area	0 to 255	P area	512 to 767
IM4 area	0 to 255	P area	768 to 1023
DB	0 to 255	DB	0 to 255
DX	0 to 255	DB	256 to 511
M	0 to 199	M	0 to 199
S		Error message: "Invalid range"	

- A process image of the S5 I/O areas P/Q/IM3/IM4 is made in the S7 I/O area. You must assign the I/O area in the configuration table.

This function is only used to convert the FB197 of an existing S5 program of an S5 CPU 95U, 103, 941 to 944, 945, 928B, 948 to a function of an S7 program for the S7-300/400 programmable controller.

4.2 IEC

4.2.1 Date and time as complex data types

Actual parameters for DATE_AND_TIME

The DATE_AND_TIME data type is a complex data type like ARRAY, STRING, and STRUCT. The permissible memory areas for complex data types are the data block (DB) and local data (L stack) areas. If you use the data type DATE_AND_TIME as formal parameter in an instruction, due to the complex data type you can specify only one of the following formats:

- A block-specific symbol from the variable declaration table for a specific block
- A symbolic name for a data block, such as e.g. "DB_sys_info.System_Time", made up of the following parts:
  - A name defined in the symbol table for the number of the data block (e.g. "DB_sys_info" for DB 5)
  - A name defined within the data block for the DATE_AND_TIME element (e.g. "Time" for a variable of data type DATE_AND_TIME contained in DB 5)

You cannot pass constants as actual parameters to formal parameters of the complex data types, including DATE_AND_TIME. Also, you cannot pass absolute addresses as actual parameters to DATE_AND_TIME.

4.2.2 FC 1 - AD_DT_TM - Add duration to instant of time

Description

The function FC 1 adds a duration *D* (time) to an instant of time *T* (date and time) and provides a new instant of time (date and time) as the result. The instant of time *T* must be in the range DT#1990-01-01-00:00:00.000 ... DT#2089-12-31-23:59:59.999. The function does not check the input parameters. If the result of the addition is not within the valid range, the result is limited to the corresponding value and the binary result (BR) bit of the status word is set to "0".

Parameter

Parameter	Declaration	Data type	Memory area	Description
T*	INPUT	DATE_AND_TIME	D, L	Instant of time in format DT
D	INPUT	TIME	I, Q, M, D, L Constant	Duration in Format TIME
RET_VAL*	OUTPUT	DATE_AND_TIME	D, L	Sum in format DT
*) You can assign only a symbolically defined variable for the parameter.				

4.2.3 FC 2 - CONCAT - Concatenate two STRING variables

Description

The function FC 2 concatenates two STRING variables together to form one string. If the resulting string is longer than the variable given at the output parameter, the result string is limited to the maximum set length and the BR bit is set to "0".

Parameter

Parameter	Declaration	Data type	Memory area	Description
IN1*	INPUT	STRING	D, L	Input variable in format STRING
IN2*	INPUT	STRING	D, L	Input variable in format STRING
RET_VAL*	OUTPUT	STRING	D, L	Concatenated string
*) You can assign only a symbolically defined variable for the parameter.				

4.2.4 FC 3 - D_TOD_DT - Combine DATE and TIME_OF_DAY

Description

The function FC 3 combines the data formats DATE and TIME_OF_DAY (TOD) and converts these formats to the data format DATE_AND_TIME (DT). The input value *IN1* must be in the range DATE#1990-01-01 ... DATE#2089-12-31. The function does not check the input parameters and does not report any errors.

Parameter

Parameter	Declaration	Data type	Memory area	Description
IN1	INPUT	DATE	I, Q, M, D, L Constant	Input variable in format DATE
IN2	INPUT	TIME_OF_DAY	I, Q, M, D, L Constant	Input variable in format TOD
RET_VAL*	OUTPUT	DATE_AND_TIME	D, L	Return value in format DT
*) You can assign only a symbolically defined variable for the parameter.				

4.2.5 FC 4 - DELETE - Delete in a STRING variable

Description

The function FC 4 deletes a number of characters *L* from the character at position *P* (inclusive) in a string. The function does not report any errors.

- If *L* and/or *P* are equal to zero or if *P* is greater than the current length of the input string, the input string is returned.
- If the sum of *L* and *P* is greater than the input string, the string is deleted up to the end.
- If *L* and/or *P* is negative, a blank string is returned and the BR bit is set to "0".

Parameter

Parameter	Declaration	Data type	Memory area	Description
IN*	INPUT	STRING	D, L	STRING variable to be deleted in
L	INPUT	INT	I, Q, M, D, L Constant	Number of characters to be deleted
P	INPUT	INT	I, Q, M, D, L Constant	Position of 1. character to be deleted

Parameter	Declaration	Data type	Memory area	Description
RET_VAL*	OUTPUT	STRING	D, L	Result string
*) You can assign only a symbolically defined variable for the parameter.				

4.2.6 FC 5 - DI_STRNG - Convert DINT to STRING

Description

The function FC 5 converts a variable in DINT data format to a string. The string is shown preceded by a sign. If the variable given at the return parameter is too short, no conversion takes place and the BR bit is set to "0".

Parameter

Parameter	Declaration	Data type	Memory area	Description
I	INPUT	DINT	I, Q, M, D, L Constant	Input value
RET_VAL*	OUTPUT	STRING	D, L	Result string
*) You can assign only a symbolically defined variable for the parameter.				

4.2.7 FC 6 - DT_DATE - Extract DATE from DT

Description

The function FC 6 extracts the data format DATE from the format DATE_AND_TIME. DATE value is between the limits DATE#1990-1-1 and DATE#2089-12-31. The function does not report any errors.

Parameter

Parameter	Declaration	Data type	Memory area	Description
IN*	INPUT	DATE_AND_TIME	D, L	Input variable in format DT
RET_VAL	OUTPUT	DATE	I, Q, M, D, L	Return value in format DATE
*) You can assign only a symbolically defined variable for the parameter.				

4.2.8 FC 7 - DT_DAY - Extract day of the week from DT

Description

The function FC 7 extracts the day of the week from the format DATE_AND_TIME. The function does not report any errors. The day of the week is returned as INTEGER value.

- 1: Sunday
- 2: Monday
- 3: Tuesday
- 4: Wednesday
- 5: Thursday
- 6: Friday
- 7: Saturday

Parameter

Parameter	Declaration	Data type	Memory area	Description
IN*	INPUT	DATE_AND_TIME	D, L	Input variable in format DT
RET_VAL	OUTPUT	INT	I, Q, M, D, L	Return value in format INT
*) You can assign only a symbolically defined variable for the parameter.				

4.2.9 FC 8 - DT_TOD - Extract TIME_OF_DAY from DT

Description

The function FC 8 extracts the data format TIME_OF_DAY from the format DATE_AND_TIME. The function does not report any errors.

Parameter

Parameter	Declaration	Data type	Memory area	Description
IN*	INPUT	DATE_AND_TIME	D, L	Input variable in format DT
RET_VAL	OUTPUT	TIME_OF_DAY	I, Q, M, D, L	Return value in format TOD
*) You can assign only a symbolically defined variable for the parameter.				

4.2.10 FC 9 - EQ_DT - Compare DT for equality

Description

The function FC 9 compares the contents of two variables in the data type format DATE_AND_TIME to determine if they are equal and outputs the result of the comparison as a return value. The return value has the signal state "1" if the time at parameter *DT1* is the same as the time at parameter *DT2*. The function does not report any errors.

Parameter

Parameter	Declaration	Data type	Memory area	Description
DT1*	INPUT	DATE_AND_TIME	D, L	Input variable in format TD
DT2*	INPUT	DATE_AND_TIME	D, L	Input variable in format TD
RET_VAL	OUTPUT	BOOL	I, Q, M, D, L	Comparison result
*) You can assign only a symbolically defined variable for the parameter.				

4.2.11 FC 10 - EQ_STRNG - Compare STRING for equal

Description

The function FC 10 compares the contents of two variables in the format STRING to determine if they are equal and outputs the result of the comparison as a return value. The return value has the signal state "1" if the string at parameter *S1* is the same as the string at parameter *S2*. The function does not report any errors.

Parameter

Parameter	Declaration	Data type	Memory area	Description
S1*	INPUT	STRING	D, L	Input variable in format STRING
S2*	INPUT	STRING	D, L	Input variable in format STRING

Parameter	Declaration	Data type	Memory area	Description
RET_VAL	OUTPUT	BOOL	I, Q, M, D, L	Comparison result
*) You can assign only a symbolically defined variable for the parameter.				

4.2.12 FC 11 - FIND - Find in a STRING variable

Description

The function FC 11 provides the position of the second string *IN2* within the first string *IN1*. The search starts on the left; the first occurrence of the string is reported. If the second string is not found in the first, zero is returned. The function does not report any errors.

Parameter

Parameter	Declaration	Data type	Memory area	Description
IN1*	INPUT	STRING	D, L	STRING variable to be searched in
IN2*	INPUT	STRING	D, L	STRING variable to be found
RET_VAL	OUTPUT	INT	I, Q, M, D, L	Position of the string found
*) You can assign only a symbolically defined variable for the parameter.				

4.2.13 FC 12 - GE_DT - Compare DT for greater than or equal

Description

The function FC 12 compares the contents of two variables in the data format DATE_AND_TIME to determine if one is greater or equal to the other and outputs the result of the comparison as a return value. The return value has the signal state "1" if the time at parameter *DT1* is greater (younger) than the time at parameter *DT2* or if both instants of time are the same. The function does not report any errors.

Parameter

Parameter	Declaration	Data type	Memory area	Description
DT1*	INPUT	DATE_AND_TIME	D, L	Input variable in format TD
DT2*	INPUT	DATE_AND_TIME	D, L	Input variable in format TD
RET_VAL	OUTPUT	BOOL	I, Q, M, D, L	Comparison result
*) You can assign only a symbolically defined variable for the parameters.				

4.2.14 FC 13 - GE_STRNG - Compare STRING for greater than or equal

Description

The function FC 13 compares the contents of two variables in the data format STRING to determine if one is greater or equal to the other and outputs the result of the comparison as a return value. The return value has the signal state "1" if the string at parameter *S1* is greater than or equal to the string at parameter *S2*. The characters are compared by their ASCII code (e.g. 'a' is greater than 'A'), starting from the left. The first character to be different decides the result of the comparison. If the left part of the longer string is identical to the shorter string, the longer string is considered as greater. The function does not report any errors.

Parameter

Parameter	Declaration	Data type	Memory area	Description
S1*	INPUT	STRING	D, L	Input variable in format STRING
S2*	INPUT	STRING	D, L	Input variable in format STRING
RET_VAL	OUTPUT	BOOL	I, Q, M, D, L	Comparison result
*) You can assign only a symbolically defined variable for the parameter.				

4.2.15 FC 14 - GT_DT - Compare DT for greater than

Description

The function FC 14 compares the contents of two variables in the data format DATE_AND_TIME to determine if one is greater to the other and outputs the result of the comparison as a return value. The return value has the signal state "1" if the time at parameter *DT1* is greater (younger) than the time at parameter *DT2*. The function does not report any errors.

Parameter

Parameter	Declaration	Data type	Memory area	Description
DT1*	INPUT	DATE_AND_TIME	D, L	Input variable in format TD
DT2*	INPUT	DATE_AND_TIME	D, L	Input variable in format TD
RET_VAL	OUTPUT	BOOL	I, Q, M, D, L	Comparison result
*) You can assign only a symbolically defined variable for the parameter.				

4.2.16 FC 15 - GT_STRNG - Compare STRING for greater than

Description

The function FC 15 compares the contents of two variables in the data format STRING to find out if the first is greater than the other and outputs the result of the comparison as a return value. The return value has the signal state "1" if the string at parameter *S1* is greater than the string at parameter *S2*. The characters are compared by their ASCII code (e.g. 'a' is greater than 'A'), starting from the left. The first character to be different decides the result of the comparison. If the left part of the longer string is identical to the shorter string, the longer string is considered as greater. The function does not report any errors.

Parameter

Parameter	Declaration	Data type	Memory area	Description
S1*	INPUT	STRING	D, L	Input variable in format STRING
S2*	INPUT	STRING	D, L	Input variable in format STRING
RET_VAL	OUTPUT	BOOL	I, Q, M, D, L	Comparison result
*) You can assign only a symbolically defined variable for the parameter.				

4.2.17 FC 16 - I_STRNG - Convert INT to STRING

Description

The function FC 16 converts a variable in DINT data format to a string. The string is shown preceded by a sign. If the variable given at the return parameter is too short, no conversion takes place and the BR bit is set to "0".

Parameter

Parameter	Declaration	Data type	Memory area	Description
I	INPUT	INT	I, Q, M, D, L Constant	Input value
RET_VAL*	OUTPUT	STRING	D, L	Result string
*) You can assign only a symbolically defined variable for the parameter.				

4.2.18 FC 17 - INSERT - Insert in a STRING variable

Description

The function FC 17 inserts a string at parameter *IN2* into the string at parameter *IN1* after the character at position *P*.

- If *P* equals zero, the second string is inserted before the first string.
- If *P* is greater than the current length of the first string, the second string is appended to the first.
- If *P* is negative, a blank string is output and the BR bit is set to "0". The binary result bit is also set to "0" if the resulting string is longer than the variable given at the output parameter; in this case the result string is limited to the maximum set length.

Parameter

Parameter	Declaration	Data type	Memory area	Description
IN1*	INPUT	STRING	D, L	STRING variable to be inserted into
IN2*	INPUT	STRING	D, L	STRING variable to be inserted
P	INPUT	INT	I, Q, M, D, L Constant	Insert position
RET_VAL*	OUTPUT	STRING	D, L	Result string
*) You can assign only a symbolically defined variable for the parameter.				

4.2.19 FC 18 - LE_DT - Compare DT for smaller than or equal

Description

The function FC 18 compares the contents of two variables in the format DATE_AND_TIME to determine if one is smaller or equal to the other and outputs the result of the comparison as a return value. The return value has the signal state "1" if the time at parameter *DT1* is smaller (older) than the time at parameter *DT2* or if both instants of time are the same. The function does not report any errors.

Parameter

Parameter	Declaration	Data type	Memory area	Description
DT1*	INPUT	DATE_AND_TIME	D, L	Input variable in format TD
DT2*	INPUT	DATE_AND_TIME	D, L	Input variable in format TD
RET_VAL*	OUTPUT	BOOL	I, Q, M, D, L	Comparison result
*) You can assign only a symbolically defined variable for the parameter.				

4.2.20 FC 19 - LE_STRNG - Compare STRING for smaller then or equal

Description

The function FC 19 compares the contents of two variables in the format STRING to determine if one is smaller or equal to the other and outputs the result of the comparison as a return value. The return value has the signal state "1" if the string at parameter *S1* is smaller than or equal to the string at parameter *S2*. The characters are compared by their ASCII code (e.g. 'A' smaller than 'a'), starting from the left. The first character to be different decides the result of the comparison. If the left part of the longer character string and the shorter character string are the same, the shorter string is smaller. The function does not report any errors.

Parameter

Parameter	Declaration	Data type	Memory area	Description
S1*	INPUT	STRING	D, L	Input variable in format STRING
S2*	INPUT	STRING	D, L	Input variable in format STRING
RET_VAL	OUTPUT	BOOL	I, Q, M, D, L	Comparison result
*) You can assign only a symbolically defined variable for the parameter.				

4.2.21 FC 20 - LEFT - Left part of a STRING variable

Description

The function FC 20 provides the first *L* characters of a string.

- If *L* is greater than the current length of the STRING variable, the input value is returned.
- With *L* = 0 and with a blank string as the input value, a blank string is returned.
- If *L* is negative, a blank string is returned and the BR bit of the status word is set to "0".

Parameter

Parameter	Declaration	Data type	Memory area	Description
IN*	INPUT	STRING	D, L	Input variable in format STRING
L	INPUT	INT	I, Q, M, D, L Constant	Length of the left character string
RET_VAL*	OUTPUT	STRING	D, L	Output variable in format STRING
*) You can assign only a symbolically defined variable for the parameter.				

4.2.22 FC 21 - LEN - Length of a STRING variable

Description

A STRING variable contains two lengths:

- Maximum length
  – It is given in square brackets when the variables are being defined.
- Current length
  – This is the number of currently valid characters.

The current length is smaller or equal to the maximum length. The number of bytes occupied by a string is 2 greater than the maximum length. The function FC 21 outputs the current length of a string (number of valid characters) as a return value. A blank string (' ') has the length zero. The maximum length is 254. The function does not report any errors.

Parameter

Parameter	Declaration	Data type	Memory area	Description
S*	INPUT	STRING	D, L	Input variable in format STRING
RET_VAL	OUTPUT	INT	I, Q, M, D, L	Number of current characters
*) You can assign only a symbolically defined variable for the parameter.				

4.2.23 FC 22 - LIMIT

Description

The function FC 22 limits the number value of a variable to limit values which can have parameters assigned.

- Variables of the data types INT, DINT, and REAL are permitted as input values.
- All variables with parameters assigned must be of the same data type.
- The variable type is recognized by the ANY pointer.
- *MN* may not be greater as *MX*.
- The output value remains unchanged and the BR bit is set to "0" if:
  – a variable with parameters assigned has an invalid data type.
  – all variables with parameters assigned do not have the same data type.
  – the lower limit value is greater than the upper limit value.
  – a REAL variable does not represent a valid floating-point number.

Parameter

Parameter	Declaration	Data type	Memory area	Description
MN	INPUT	ANY	I, Q, M, D, L	Lower limit
IN	INPUT	ANY	I, Q, M, D, L	Input variable
MX	INPUT	ANY	I, Q, M, D, L	Upper limit
RET_VAL	OUTPUT	ANY	I, Q, M, D, L	Limited output variable

4.2.24 FC 23 - LT_DT - Compare DT for smaller than

Description

The function FC 23 compares the contents of two variables in the format DATE_AND_TIME to determine if one is smaller to the other and outputs the result of the comparison as a return value. The return value has the signal state "1" if the time at parameter *DT1* is smaller (older) than the time at parameter *DT2*. The function does not report any errors.

Parameter

Parameter	Declaration	Data type	Memory area	Description
DT1*	INPUT	DATE_AND_TIME	D, L	Input variable in format TD
DT2*	INPUT	DATE_AND_TIME	D, L	Input variable in format TD
RET_VAL	OUTPUT	BOOL	I, Q, M, D, L	Comparison result
*) You can assign only a symbolically defined variable for the parameter.				

4.2.25 FC 24 - LT_STRNG - Compare STRING for smaller

Description

The function FC 24 compares the contents of two variables in the format STRING to determine if one is smaller to the other and outputs the result of the comparison as a return value. The return value has the signal state "1" if the string at parameter *S1* is smaller than the string at parameter *S2*. The characters are compared by their ASCII code (e.g. 'A' smaller than 'a'), starting from the left. The first character to be different decides the result of the comparison. If the left part of the longer character string and the shorter character string are the same, the shorter string is smaller. The function does not report any errors.

Parameter

Parameter	Declaration	Data type	Memory area	Description
S1*	INPUT	STRING	D, L	Input variable in format STRING
S2*	INPUT	STRING	D, L	Input variable in format STRING
RET_VAL	OUTPUT	BOOL	I, Q, M, D, L	Comparison result
*) You can assign only a symbolically defined variable for the parameter.				

4.2.26 FC 25 - MAX - Select maximum

Description

The function FC 25 selects the largest of three numerical variable values.

- Variables of the data types INT, DINT, and REAL are permitted as input values.
- All variables with parameters assigned must be of the same data type.
- The variable type is recognized by the ANY pointer.
- The output value remains unchanged and the BR bit is set to "0" if:
  - a variable with parameters assigned has an invalid data type.
  - all variables with parameters assigned do not have the same data type.
  - a REAL variable does not represent a valid floating-point number.

Parameter

Parameter	Declaration	Data type	Memory area	Description
IN1	INPUT	ANY	I, Q, M, D, L	1. Input value
IN2	INPUT	ANY	I, Q, M, D, L	2. Input value
IN3	INPUT	ANY	I, Q, M, D, L	3. Input value
RET_VAL	OUTPUT	ANY	I, Q, M, D, L	Largest of the input values

The admitted data types INT, DINT and REAL must be entered in the ANY pointer. Such parameters as "MD20" are also admitted, but you must define the corresponding data type of "MD20" in "Symbol".

Example in STL:

```
CALL FC 25
IN1 := P#M 10.0 DINT 1
IN2 := MD20
IN3 := P#DB1.DBX 0.0 DINT 1
RET_VAL := P#M 40.0 DINT 1
= M 0.0
```

4.2.27 FC 26 - MID - Middle part of a STRING variable

Description

The function FC 26 provides the middle part of a string (*L* characters from the character *P* inclusive).

- If the sum of *L* and (*P*-1) exceeds the current length of the STRING variables, a string is returned from the character *P* to the end of the input value.
- In all other cases (*P* is outside the current length, *P* and/or *L* are equal to zero or negative), a blank string is returned and the BR bit is set to "0".

Parameter

Parameter	Declaration	Data type	Memory area	Description
IN*	INPUT	STRING	D, L	Input variable in format STRING
L	INPUT	INT	I, Q, M, D, L Constant	Length of the middle character string
P	INPUT	INT	I, Q, M, D, L Constant	Position of first character
RET_VAL*	OUTPUT	STRING	D, L	Output variable in format STRING

*) You can assign only a symbolically defined variable for the parameter.

4.2.28 FC 27 - MIN - Select minimum

Description

The function FC 27 selects the smallest of three numerical variable values.

- Variables of the data types INT, DINT, and REAL are permitted as input values.
- All variables with parameters assigned must be of the same data type.
- The variable type is recognized by the ANY pointer.
- The output value remains unchanged and the BR bit is set to "0" if:
  - a variable with parameters assigned has an invalid data type.
  - all variables with parameters assigned do not have the same data type.
  - a REAL variable does not represent a valid floating-point number.

Parameter

Parameter	Declaration	Data type	Memory area	Description
IN1	INPUT	ANY	I, Q, M, D, L	1. Input value
IN2	INPUT	ANY	I, Q, M, D, L	2. Input value
IN3	INPUT	ANY	I, Q, M, D, L	3. Input value
RET_VAL	OUTPUT	ANY	I, Q, M, D, L	Smallest of the input values

The admitted data types INT, DINT and REAL must be entered in the ANY pointer. Such parameters as "MD20" are also admitted, but you must define the corresponding data type of "MD20" in "Symbol".

Example in STL:

```
CALL FC 27
IN1 := P#M 10.0 DINT 1
IN2 := MD20
IN3 := P#DB1.DBX 0.0 DINT 1
RET_VAL := P#M 40.0 DINT 1
= M 0.0
```

4.2.29 FC 28 - NE_DT - Compare DT for unequal

Description

The function FC 28 compares the contents of two variables in the format DATE_AND_TIME to determine if they are unequal and outputs the result of the comparison as a return value. The return value has the signal state "1" if the time at parameter *DT1* is unequal the time at parameter *DT2*. The function does not report any errors.

Parameter

Parameter	Declaration	Data type	Memory area	Description
DT1*	INPUT	DATE_AND_TIME	D, L	Input variable in format TD
DT2*	INPUT	DATE_AND_TIME	D, L	Input variable in format TD
RET_VAL	OUTPUT	BOOL	I, Q, M, D, L	Comparison result
*) You can assign only a symbolically defined variable for the parameter.				

4.2.30 FC 29 - NE_STRNG - Compare STRING for unequal

Description

The function FC 29 compares the contents of two variables in the format STRING to determine if they are unequal and outputs the result of the comparison as a return value. The return value has the signal state "1" if the string at parameter *S1* is unequal to the string at parameter *S2*. The function does not report any errors.

Parameter

Parameter	Declaration	Data type	Memory area	Description
S1*	INPUT	STRING	D, L	Input variable in format STRING
S2*	INPUT	STRING	D, L	Input variable in format STRING
RET_VAL	OUTPUT	BOOL	I, Q, M, D, L	Comparison result
*) You can assign only a symbolically defined variable for the parameter.				

4.2.31 FC 30 - R_STRNG - Convert REAL to STRING

Description

The function FC 30 converts a variable in REAL data format to a string.

- The string is shown with 14 digits:
  ±v.nnnnnnnE±xx
  - ±: Sign
  - v: 1 digit before the decimal point
  - n: 7 digits after the decimal point
  - x: 2 exponential digits
- If the variable given at the return parameter is too short or if no valid floating-point number is given at parameter IN, no conversion takes place and the BR bit is set to "0".

Parameter

Parameter	Declaration	Data type	Memory area	Description
IN	INPUT	REAL	I, Q, M, D, L Constant	Input value
RET_VAL*	OUTPUT	STRING	D, L	Result string
*) You can assign only a symbolically defined variable for the parameter.				

4.2.32 FC 31 - REPLACE - Replace in a STRING variable

Description

The function FC 31 replaces a number of characters *L* of the first string *IN1* starting at the character at position *P* (inclusive) with the entire second string *IN2*.

- If *L* is equal to zero and *P* is not equal to zero, the first string is returned.
- If *L* is equal to zero and *P* is equal to zero, the second string is precent to the first string.
- If *L* is not equal to zero and *P* is equal to zero or one, the string is replaced from the 1. character (inclusive).
- If *P* is outside the first string, the second string is appended to the first string.
- If *L* and/or *P* is negative, a blank string is returned and the BR bit is set to "0". The BR bit is also set to "0" if the resulting string is longer than the variable given at the output parameter; in this case the result string is limited to the maximum set length.

Parameter

Parameter	Declaration	Data type	Memory area	Description
IN1*	INPUT	STRING	D, L	STRING variable to be inserted into
IN2*	INPUT	STRING	D, L	STRING variable to be inserted
L	INPUT	INT	I, Q, M, D, L Constant	Number of characters to be replaced
P	INPUT	INT	I, Q, M, D, L Constant	Position of 1. character to be replaced
RET_VAL*	OUTPUT	STRING	D, L	Result string
*) You can assign only a symbolically defined variable for the parameter.				

4.2.33 FC 32 - RIGHT - Right part of a STRING variable

Description

The function FC 32 provides the last *L* characters of a string.

- If *L* is greater than the current length of the STRING variable, the input value is returned.
- With *L* = 0 and with a blank string as the input value, a blank string is returned.
- If *L* is negative, a blank string is returned and the BR bit is set to "0".

Parameter

Parameter	Declaration	Data type	Memory area	Description
IN*	INPUT	STRING	D, L	Input variable in format STRING
L	INPUT	INT	I, Q, M, D, L Constant	Length of the right character string
RET_VAL*	OUTPUT	STRING	D, L	Output variable in format STRING
*) You can assign only a symbolically defined variable for the parameter.				

4.2.34 FC 33 - S5TI_TIM - Convert S5TIME to TIME

Description

The function FC 33 converts the data format S5TIME to the data format TIME. If the result of the conversion is outside the TIME range, the result is limited to the corresponding value and the binary result (BR) bit is set to "0".

Parameter

Parameter	Declaration	Data type	Memory area	Description
IN	INPUT	S5TIME	I, Q, M, D, L Constant	Input variable in format S5TIME
RET_VAL	OUTPUT	TIME	I, Q, M, D, L	Return value in format TIME

4.2.35 FC 34 - SB_DT_DT - Subtract two instants of time

Description

The function FC 34 subtracts two instants of time *DTx* (date and time) and provides a duration (time) as the result. The instants of time *DTx* must be in the range DT#1990-01-01-00:00:00.000 ... DT#2089-12-31-23:59:59.999. The function does not check the input parameters. It is valid:

- With *DT1* > *DT2* the result is positive.
- With *DT1* < *DT2* the result is negative.
- If the result of the subtraction is outside the TIME range, the result is limited to the corresponding value and the binary result (BR) bit is set to "0".

Parameter

Parameter	Declaration	Data type	Memory area	Description
DT1*	INPUT	DATE_AND_TIME	D, L	1. instant of time in format DT
DT2*	INPUT	DATE_AND_TIME	D, L	2. Instant of time in format DT
RET_VAL	OUTPUT	TIME	I, Q, M, D, L	Difference in format TIME
*) You can assign only a symbolically defined variable for the parameter.				

4.2.36 FC 35 - SB_DT_TM - Subtract a duration from a time

Description

The function FC 35 subtracts a duration *D* (TIME) from a time *T* (DT) and provides a new time (DT) as the result. The time *T* must be between DT#1990-01-01-00:00:00.000 and DT#2089-12-31-23:59:59.999. The function does not run an input check. If the result of the subtraction is not within the valid range, the result is limited to the corresponding value and the binary result (BR) bit of the status word is set to "0".

Parameter

Parameter	Declaration	Data type	Memory area	Description
T*	INPUT	DATE_AND_TIME	D, L	Time in format DT
D	INPUT	TIME	I, Q, M, D, L, constant	Duration in format TIME
RET_VAL *	OUTPUT	DATE_AND_TIME	D, L	Difference in format DT
*) You can assign only a symbolically defined variable for the parameter.				

4.2.37 FC 36 - SEL - Binary selection

Description

The function FC 36 selects one of two variable values depending on a switch *G*.

- Variables with all data types which correspond to the data width bit, byte, word, and double word (not data types DT and STRING) are permitted as input values at the parameters *IN0* and *IN1*.
- *IN0*, *IN1* and *RET_VAL* must be of the same data type.
- The output value remains unchanged and the BR bit is set to "0" if:
  – a variable with parameters assigned has an invalid data type.
  – all variables with parameters assigned do not have the same data type.
  – a REAL variable does not represent a valid floating-point number.

Parameter

Parameter	Declaration	Data type	Memory area	Description
G	INPUT	BOOL	I, Q, M, D, L Constant	Selection switch
IN0	INPUT	ANY	I, Q, M, D, L	1. Input value
IN1	INPUT	ANY	I, Q, M, D, L	2. Input value
RET_VAL	OUTPUT	ANY	I, Q, M, D, L	Selected input value

4.2.38 FC 37 - STRNG_DI - Convert STRING to DINT

Description

The function FC 37 converts a string to a variable in DINT data format.

- The first character in the string may be a sign or a number, the characters which then follow must be numbers.
- If the length of the string is equal to zero or greater than 11, or if invalid characters are found in the string, no conversion takes place and the BR bit is set to "0".
- If the result of the conversion is outside the DINT range, the result is limited to the corresponding value and the BR bit is set to "0".

Parameter

Parameter	Declaration	Data type	Memory area	Description
S*	INPUT	STRING	D, L	Input string
RET_VAL	OUTPUT	DINT	I, Q, M, D, L	Result
*) You can assign only a symbolically defined variable for the parameter.				

4.2.39 FC 38 - STRNG_I - Convert STRING to INT

Description

The function FC 38 converts a string to a variable in INT data format.

- The first character in the string may be a sign or a number, the characters which then follow must be numbers.
- If the length of the string is equal to zero or greater than 6, or if invalid characters are found in the string, no conversion takes place and the BR bit is set to "0".
- If the result of the conversion is outside the INT range, the result is limited to the corresponding value and the BR bit is set to "0".

Parameter

Parameter	Declaration	Data type	Memory area	Description
S*	INPUT	STRING	D, L	Input string
RET_VAL	OUTPUT	INT	I, Q, M, D, L	Result
*) You can assign only a symbolically defined variable for the parameter.				

4.2.40 FC 39 - STRNG_R - Convert STRING to REAL

Description

The function FC 39 converts a string to a variable in REAL data format.

- The string must have the following format:
  ±v.nnnnnnnE±xx
  – ±: Sign
  – v: 1 digit before the decimal point
  – n: 7 digits after the decimal point
  – x: 2 exponential digits
- If the length of the string is smaller than 14, or if it is not structured as shown above, no conversion takes place and the BR bit is set to "0".
- If the result of the conversion is outside the REAL range, the result is limited to the corresponding value and the BR bit is set to "0".

Parameter

Parameter	Declaration	Data type	Memory area	Description
S*	INPUT	STRING	D, L	Input string
RET_VAL	OUTPUT	REAL	I, Q, M, D, L	Result
*) You can assign only a symbolically defined variable for the parameter.				

4.2.41 FC 40 - TIM_S5TI - Convert TIME to S5TIME

Description

The function FC 40 converts the data format TIME to the format S5TIME. Here is always rounded down. If the input parameter is greater than the displayable S5TIME format (TIME#02:46:30.000), S5TIME#999.3 is output as result and the binary result (BR) bit is set to "0".

Parameter

Parameter	Declaration	Data type	Memory area	Description
IN	INPUT	TIME	I, Q, M, D, L Constant	Input variable in format TIME
RET_VAL	OUTPUT	S5TIME	I, Q, M, D, L	Return value in format S5TIME

4.3 IO

4.3.1 FB 20 - GETIO - PROFIBUS/PROFINET read all Inputs

Description

With the FB 20 GETIO you consistently read out all inputs of a PROFIBUS DP slave/ PROFINET IO device. In doing so, FB 20 calls the SFC 14 DPRD_DAT. If there was no error during the data transmission, the data that have been read are entered in the target area indicated by *INPUTS*. The target area must have the same length that you configured for the selected component. In the case of a PROFIBUS DP slave with a modular structure or with several DP IDs, you can only access the data for one component/DP ID with an FB 20 call each time at the configured start address.

Parameter	Declaration	Data Type	Memory Area	Description
ID	INPUT	DWORD	I, Q, M, D, L constant	■ Low word: logical address of the DP slave/PROFINET IO component (module or submodule) ■ High word: irrelevant
STATUS	OUTPUT	DWORD	I, Q, M, D, L	Contains error information for SFC 14 DPRD_DAT in the form DW#16#40xxxx00
LEN	OUTPUT	INT	I, Q, M, D, L	Amount of data read in bytes
INPUTS	IN_OUT	ANY	I, Q, M, D	Target area for the read data. It must have the same length as the area that you configured for the selected DP slave/PROFINET IO component. Only the data type BYTE is permitted.

Error Information Please refer to SFC 14 - DPRD_DAT - Read consistent data.

4.3.2 FB 21 - SETIO - PROFIBUS/PROFINET write all Outputs

Description With the FB 21 SETIO you consistently transfer the data from the source area indicated by *OUTPUTS* to the addressed PROFIBUS DP slave/PROFINET IO device, and, if necessary, to the process image (in the case where you have configured the affected address area for the DP standard slave as a consistency area in a process image). In doing so, FB 21 calls the SFC 15 DPWR_DAT. The source area must have the same length that you configured with for the selected component. In the case of a DP standard slave with a modular structure or with several DP IDs, you can only access the data for one component/DP ID with an FB 20 call each time at the configured start address.

Parameter	Declaration	Data Type	Memory Area	Description
ID	INPUT	DWORD	I, Q, M, D, L, constant	■ Low word: logical address of the DP slave/PROFINET IO component ■ (module or submodule) ■ High word: irrelevant
LEN	INPUT	INT	I, Q, M, D, L	Irrelevant
STATUS	OUTPUT	DWORD	I, Q, M, D, L	Contains error information for SFC 15 DPRD_DAT in the form DW#16#40xxxx00
OUTPUTS	IN_OUT	ANY	I, Q, M, D	Source area for the read data to be read. It must have the same length as the area that you configured for the selected DP slave/ PROFINET IO component. Only the data type BYTE is permitted.

Error Information Please refer to SFC 15 - DPWR_DAT - Write consistent data.

4.3.3 FB 22 - GETIO_PART - PROFIBUS/PROFINET read a part of the Inputs

Description With the FB 22 GETIO_PART you consistently read a part of the process image area belonging to a PROFIBUS DP slave/PROFINET IO device. In doing so, FB 22 calls the SFC 81 UBLKMOV.

You must assign a process image partition for inputs to the OB in which FB 22 GETIO_PART is called. Furthermore, before calling FB 22 you must add the associated PROFIBUS DP slave or the associated PROFINET IO device to this process image partition for inputs. If your CPU does not recognize any process image partitions or you want to call FB 22 in OB 1, you must add the associated PROFIBUS DP slave or the associated PROFINET IO device to this process image partition for inputs before calling FB 22. You use the OFFSET and LEN parameters to specify the portion of the process image area to be read for the components addressed by means of their ID. If there was no error during the data transmission, ERROR receives the value FALSE, and the data that have been read are entered in the target area indicated by INPUTS. If there was an error during the data transmission, ERROR receives the value TRUE, and STATUS receives the SFC 81 error information UBLKMOV. If the target area (INPUTS parameter) is smaller than LEN, then as many bytes as INPUTS can accept are transferred. ERROR receives the value FALSE. If the target area is greater than LEN, then the first LEN bytes in the target area are written. ERROR receives the value FALSE.

The FB 22 GETIO_PART does not check the process image for inputs for delimiters between data belonging to different PROFIBUS DP or PROFINET IO components. Because of this, you yourself must make sure that the process image area specified by means of OFFSET and LEN belongs to one component. Reading of data for more than one component cannot be guaranteed for future systems and compromises the transferability to systems from other manufacturers.

Parameter	Declaration	Data Type	Memory Area	Description
ID	INPUT	DWORD	I, Q, M, D, L constant	■ Low word: logical address of the DP slave/ PROFINET IO component (module or submodule) ■ High word: irrelevant
OFFSET	INPUT	INT	I, Q, M, D, L constant	Number of the first byte to be read in the process image for the component (smallest possible value: 0)
LEN	INPUT	INT	I, Q, M, D, L constant	Amount of bytes to be read
STATUS	OUTPUT	DWORD	I, Q, M, D, L	Contains error information for SFC 81 UBLKMOV in the form DW#16#40xxxx00 if *ERROR* = TRUE

Parameter	Declaration	Data Type	Memory Area	Description
ERROR	OUTPUT	BOOL	I, Q, M, D, L	Error display: *ERROR* = TRUE if an error occurs when calling SFC 81 UBLKMOV.
INPUTS	IN_OUT	ANY	I, Q, M, D	Target area for read data: ■ If the target area is smaller than *LEN*, then as many bytes as *INPUTS* can accept are transferred. ERROR receives the value FALSE. ■ If the target area is greater than *LEN*, then the first *LEN* bytes of the target area are written. *ERROR* receives the value FALSE.

Error Information

Please refer to SFC 81 - UBLKMOV - Copy data area without gaps.

4.3.4 FB 23 - SETIO_PART - PROFIBUS/PROFINET write a part of the Outputs

Description

With the FB 23 SETIO_PART you transfer data from the source area indicated by *OUTPUTS* into a part of the process image area belonging to a PROFIBUS DP slave/ PROFINET IO device. In doing so, FB 23 calls the SFC 81 UBLKMOV.

You must assign a process image partition for outputs to the OB in which FB 23 SETIO_PART is called. Furthermore, before calling FB 23 you must add the associated PROFIBUS DP slave or the associated PROFINET IO device to this process image partition for outputs. If your CPU does not recognize any process image partitions or you want to call FB 23 in OB 1, you must add the associated PROFIBUS DP slave or the associated PROFINET IO device to this process image partition for outputs before calling FB 23. You use the OFFSET and LEN parameters to specify the portion of the process image area to be written for the components addressed by means of their ID. If there was no error during the data transmission, ERROR receives the value FALSE. If there was an error during the data transmission, ERROR receives the value TRUE, and STATUS receives the SFC 81 error information UBLKMOV. If the source area (OUTPUTS parameter) is smaller than LEN, then as many bytes as OUTPUTS contains are transferred. ERROR receives the value FALSE. If the source area is greater than LEN, then the first LEN bytes are transferred from OUTPUTS. ERROR receives the value FALSE.

The FB 23 SETIO_PART does not check the process image for inputs for delimiters between data that belong to different PROFIBUS DP or PROFINET IO components. Because of this, you yourself must make sure that the process image area specified by means of OFFSET and LEN belongs to one component. Writing of data for more than one component cannot be guaranteed for future systems and compromises the transferability to systems from other manufacturers.

Parameter	Declaration	Data Type	Memory Area	Description
ID	INPUT	DWORD	I, Q, M, D, L, constant	■ Low word: logical address of the DP slave/PROFINET IO component (module or submodule) ■ High word: irrelevant
OFFSET	INPUT	INT	I, Q, M, D, L, constant	Number of the first byte to be written in the process image for the component (smallest possible value: 0)
LEN	INPUT	INT	I, Q, M, D, L, constant	Amount of bytes to be written
STATUS	OUTPUT	DWORD	I, Q, M, D	Contains error information for SFC 81 UBLKMOV in the form DW#16#40xxxx00 if *ERROR* = TRUE
ERROR	OUTPUT	BOOL	I, Q, M, D	Error display: *ERROR* = TRUE if an error occurs when calling SFC 81 UBLKMOV.
OUTPUTS	IN_OUT	ANY	I, Q, M, D	Source area for the data to be written: ■ If the source area is smaller than *LEN*, then as many bytes as OUTPUTS contains are transferred. *ERROR* receives the value FALSE. ■ If the source area is greater than *LEN*, then the first *LEN* bytes are transferred from *OUTPUTS*. *ERROR* receives the value FALSE.

Error Information

Please refer to SFC 81 - UBLKMOV - Copy data area without gaps.

4.4 S5 Converting

4.4.1 FC 112 - Sine(x) - Sine

Description

The function FC 112 expects the input value in ACCU 1 as a floating point number.

1. The input value must be within the range between zero

   (REAL = +0.0000000e+00) ... 2 x π (REAL = +0.6283185e+01)
2. The function also stores the result in ACCU 1 as a floating point number.
3. The input value DWORD = DW#16#0000 0000 is treated the same way as the floating point value zero (REAL = +0.0000000e+00 in accordance with DWORD = DW#16#8000 0000).

   ⇨ If the calculation is carried out correctly, the RLO *ENO* is FALSE after the function has been called up.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	INPUT	BOOL	I, Q, M, D, L	■ Enable – TRUE: activates the function – FALSE: deactivates the function
ENO	OUTPUT	BOOL	I, Q, M, D, L	■ Status – TRUE: function executed with error

Error information

In the event of an error, the function sets the RLO to signal state *ENO* to TRUE (if the input value is out of range from 0 to 2 x π). In this case, the contents of ACCU 1 remain unchanged. The assignment of the remaining registers and the auxiliary flags are not changed.

This function is only used to convert the FB 101 of an existing S5 program to a function of an S7 program programmable controller.

4.4.2 FC 113 - Cosine(x) - Cosine

Description

The function FC 113 expects the input value in ACCU 1 as a floating point number.

1. The input value must be within the range between zero

   (REAL = +0.0000000e+00) ... 2 x π (REAL = +0.6283185e+01)
2. The function also stores the result in ACCU 1 as a floating point number.
3. The input value DWORD = DW#16#0000 0000 is treated the same way as the floating point value zero (REAL = +0.0000000e+00 in accordance with DWORD = DW#16#8000 0000).

   ⇨ If the calculation is carried out correctly, the RLO *ENO* is FALSE after the function has been called up.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	INPUT	BOOL	I, Q, M, D, L	■ Enable – TRUE: activates the function – FALSE: deactivates the function
ENO	OUTPUT	BOOL	I, Q, M, D, L	■ Status – TRUE: function executed with error

Error information

In the event of an error, if the input value is out of range from 0 ... 2 x π, the function sets the RLO to signal state *ENO* to TRUE. In this case, the contents of ACCU 1 remain unchanged. The assignment of the remaining registers and the auxiliary flags are not changed.

This function is only used to convert the FB 102 of an existing S5 program to a function of an S7 program programmable controller.

4.4.3 FC 114 - Tangent(x) - Tangent

Description

The function FC 114 expects the input value in ACCU 1 as a floating point number.

1. The input value must be within the range between zero

   (REAL = +0.0000000e+00) ... 2 x π (REAL = +0.6283185e+01)
2. The function also stores the result in ACCU 1 as a floating point number.
3. The input value DWORD = DW#16#0000 0000 is treated the same way as the floating point value zero (REAL = +0.0000000e+00 in accordance with DWORD = DW#16#8000 0000).

   ⇨ If the calculation is carried out correctly, the RLO *ENO* is FALSE after the function has been called up.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	INPUT	BOOL	I, Q, M, D, L	■ Enable – TRUE: activates the function – FALSE: deactivates the function
ENO	OUTPUT	BOOL	I, Q, M, D, L	■ Status – TRUE: function executed with error

Error information

In the event of an error, the function sets the RLO to signal state *ENO* to TRUE. In this case, the contents of accumulator 1 remain unchanged. One of the following errors has occurred:

- The input value is out of range from 0 ... 2 x π.
- A number range overflow occurred during calculation of the function.
- The input value amounts to π/2 or 3 x π/2. In this case, the function value is infinite.

The assignment of the remaining registers and the auxiliary flags are not changed.

This function is only used to convert the FB 103 of an existing S5 program to a function of an S7 program programmable controller.

4.4.4 FC 115 - Cotangent(x) - Cotangent

Description

The function FC 115 expects the input value in ACCU 1 as a floating point number.

1. The input value must be within the range between zero

   (REAL = +0.0000000e+00) ... 2 x π (REAL = +0.6283185e+01)

2. The function also stores the result in ACCU 1 as a floating point number.
3. The input value DWORD = DW#16#0000 0000 is treated the same way as the floating point value zero (REAL = +0.0000000e+00 in accordance with DWORD = DW#16#8000 0000).

   ⇨ If the calculation is carried out correctly, the RLO *ENO* is FALSE after the function has been called up.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	INPUT	BOOL	I, Q, M, D, L	■ Enable – TRUE: activates the function – FALSE: deactivates the function
ENO	OUTPUT	BOOL	I, Q, M, D, L	■ Status – TRUE: function executed with error

Error information

In the event of an error, the function sets the RLO to signal state *ENO* to TRUE. In this case, the contents of accumulator 1 remain unchanged. One of the following errors has occurred:

- The input value is out of range from REAL = +0.2938734e-34 and REAL = +0.6283184e+01.
- A number range overflow occurred during calculation of the function.
- The input value amounts to zero or π or 2 x π. In this case, the function value is infinite.

The assignment of the remaining registers and the auxiliary flags are not changed.

This function is only used to convert the FB 103 of an existing S5 program to a function of an S7 program programmable controller.

4.4.5 FC 116 - Arc Sine(x) - Arcussine

Description

The function FC 116 expects the input value in ACCU 1 as a floating point number.

1. The input value must be within the range between

   -1 (REAL = -0.1000000e+01) ... +1 (REAL = +0.1000000e+01)
2. The function also stores the result in ACCU 1 as a floating point number.
3. The input value DWORD = DW#16#0000 0000 is treated the same way as the floating point value zero (REAL = +0.0000000e+00 in accordance with DWORD = DW#16#8000 0000).

   ⇨ If the calculation is carried out correctly, the RLO *ENO* is FALSE after the function has been called up.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	INPUT	BOOL	I, Q, M, D, L	■ Enable – TRUE: activates the function – FALSE: deactivates the function
ENO	OUTPUT	BOOL	I, Q, M, D, L	■ Status – TRUE: function executed with error

Error information

In the event of an error, if the input value is out of range of -1 ... +1, the function sets the RLO signal state *ENO* to TRUE. The assignment of the remaining registers and the auxiliary flags are not changed.

This function is only used to convert the FB 105 of an existing S5 program to a function of an S7 program programmable controller.

4.4.6 FC 117 - Arc Cosine(x) - Arcuscosine

Description

The function FC 117 expects the input value in ACCU 1 as a floating point number.

1. The input value must be within the range between

   -1 (REAL = -0.1000000e+01) ... +1 (REAL = +0.1000000e+01)
2. The function also stores the result in ACCU 1 as a floating point number.
3. The input value DWORD = DW#16#0000 0000 is treated the same way as the floating point value zero (REAL = +0.0000000e+00 in accordance with DWORD = DW#16#8000 0000).

   ⇨ If the calculation is carried out correctly, the RLO *ENO* is FALSE after the function has been called up.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	INPUT	BOOL	I, Q, M, D, L	■ Enable – TRUE: activates the function – FALSE: deactivates the function
ENO	OUTPUT	BOOL	I, Q, M, D, L	■ Status – TRUE: function executed with error

Error information

In the event of an error, if the input value is out of range of -1 ... +1, the function sets the RLO signal state *ENO* to TRUE. The assignment of the remaining registers and the auxiliary flags are not changed.

This function is only used to convert the FB 106 of an existing S5 program to a function of an S7 program programmable controller.

4.4.7 FC 118 - Arc Tangent(x) - Arcustangent

Description

The function FC 118 expects the input value in ACCU 1 as a floating point number.

1. The input value must be within the range between

   -1 (REAL = -0.1000000e+01) ... +1 (REAL = +0.1000000e+01)
2. The function also stores the result in ACCU 1 as a floating point number.
3. The input value DWORD = DW#16#0000 0000 is treated the same way as the floating point value zero (REAL = +0.0000000e+00 in accordance with DWORD = DW#16#8000 0000).
4. If the input value is greater than REAL = +0.1209486e+07, the result + π/2 is issued.

   If the input value is less than REAL = -0.5773456e+07, the result π/2 is issued.

   ⇨ The RLO *ENO* is set to signal state FALSE.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	INPUT	BOOL	I, Q, M, D, L	■ Enable – TRUE: activates the function – FALSE: deactivates the function
ENO	OUTPUT	BOOL	I, Q, M, D, L	■ Status – TRUE: function executed with error

Error information

In the event of an error, if the input value is out of range of -1 ... +1, the function sets the RLO signal state *ENO* to TRUE. The assignment of the remaining registers and the auxiliary flags are not changed.

This function is only used to convert the FB107 of an existing S5 program to a function of an S7 program programmable controller.

4.4.8 FC 119 - Arc Cotangent(x) - Arcuscotangent

Description

The function FC 119 expects the input value in ACCU 1 as a floating point number.

1. The input value must be within the range between

   -1 (REAL = -0.1000000e+01) ... +1 (REAL = +0.1000000e+01)
2. The function also stores the result in ACCU 1 as a floating point number.
3. The input value DWORD = DW#16#0000 0000 is treated the same way as the floating point value zero (REAL = +0.0000000e+00 in accordance with DWORD = DW#16#8000 0000).
4. If the input value is greater than REAL = +1.209486e+07, the result +π/2 is issued.

   If the input value is less than REAL = -0.5773456e+07, the result π/2 is issued.

   ⇨ The RLO *ENO* is set to signal state FALSE.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	INPUT	BOOL	I, Q, M, D, L	■ Enable – TRUE: activates the function – FALSE: deactivates the function
ENO	OUTPUT	BOOL	I, Q, M, D, L	■ Status – TRUE: function executed with error

Error information

In the event of an error, if the input value is not in the range of -1 ... +1, the function sets the RLO signal state *ENO* to TRUE. The assignment of the remaining registers and the auxiliary flags are not changed.

This function is only used to convert the FB 108 of an existing S5 program to a function of an S7 program programmable controller.

4.4.9 FC 120 - Naperian Logarithm In(x) - Naperian Logarithm

Description

The function FC 120 expects the input value in accumulator 1 as a floating point number.

1. The input value must be within the range between

   -1 (REAL = -0.1000000e+01) and +1 (REAL = +0.1000000e+01).
2. The function also stores the result in accumulator 1 as a floating point number.
3. If the calculation is carried out correctly, the RLO is FALSE after the function has been called up.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	INPUT	BOOL	I, Q, M, D, L	■ Enable – TRUE: activates the function – FALSE: deactivates the function
ENO	OUTPUT	BOOL	I, Q, M, D, L	■ Status – TRUE: function executed with error

Error information

In the event of an error, the function sets the *ENO* to signal state TRUE (if the input value is less than or equal to zero). In this case, the contents of accumulator 1 remain unchanged. The assignment of the remaining registers and that of the auxiliary flags are not changed.

This function is only used to convert the FB 109 of an existing S5 program to a function of an S7 program programmable controller.

4.4.10 FC 121 - Decimal Logarithm lg(x) - Decimal Logarithm

Description

The function FC 121 expects the input value in accumulator 1 as a bit floating point number.

1. The input value must be within the range between

   -1 (REAL = -0.1000000e+01) and +1 (REAL = +0.1000000e+01).
2. The function also stores the result in accumulator 1 as a floating point number.
3. If the calculation is carried out correctly, the RLO is FALSE after the function has been called up.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	INPUT	BOOL	I, Q, M, D, L	■ Enable – TRUE: activates the function – FALSE: deactivates the function
ENO	OUTPUT	BOOL	I, Q, M, D, L	■ Status – TRUE: function executed with error

Error information

In the event of an error, the function sets the *ENO* to signal state TRUE (if the input value is less than or equal to zero). In this case, the contents of accumulator 1 remain unchanged. The assignment of the remaining registers and that of the auxiliary flags are not changed.

This function is only used to convert the FB 110 of an existing S5 program to a function of an S7 program programmable controller.

4.4.11 FC 122 - Gen. Logarithm to Base b - General Logarithm log (x) to base b

Description

The function FC 122 expects both the input value for the base (b) in ACCU 2 and the input value for the antilogarithm (x) in ACCU 1 as floating point numbers.

1. Both input values must be greater than zero and in addition, the base may not have the value +1.
2. If the calculation is carried out correctly, the result is stored in ACCU 1 as a floating point number, the previous contents of ACCU 3 are in ACCU 2, and the previous contents of ACCU 4 are in ACCU 3. The contents of ACCU 4 are not changed. The assignment of the remaining registers and that of the auxiliary flags are not changed.
3. In the case of a calculation without errors, the RLO *ENO* is FALSE after the function has been called up.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	INPUT	BOOL	I, Q, M, D, L	■ Enable – TRUE: activates the function – FALSE: deactivates the function
ENO	OUTPUT	BOOL	I, Q, M, D, L	■ Status – TRUE: function executed with error

Error information

In case of an error, if one of the input values is less than or equal to zero, or if the base has the value +1, the function sets the link result *ENO* to the signal state TRUE. Then the contents of the ACCUs remain unchanged.

This function is only used to convert the FB 111 of an existing S5 program to a function of an S7 program programmable controller.

4.4.12 FC 123 - E to Power n - E high n

Description

The function FC 123 expects the input value in ACCU 1 as a floating point number.

1. The input value DWORD = DW#16#0000 0000 is treated the same way as the floating point value zero (REAL = +0.0000000e+00 in accordance with DWORD = DW#16#8000 0000).
2. The function also stores the result in ACCU 1 as a floating point number.
3. If the calculation is carried out correctly, the RLO *ENO* is FALSE after the function has been called up.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	INPUT	BOOL	I, Q, M, D, L	■ Enable – TRUE: activates the function – FALSE: deactivates the function
ENO	OUTPUT	BOOL	I, Q, M, D, L	■ Status – TRUE: function executed with error

Error information

In the event of an error, if the input value is not within the range from REAL = -0.8802962e+02 to REAL = +0.8802966e+02 (than the value would be outside the number range), the function sets the RLO *ENO* to signal state TRUE. In this case, the contents of ACCU 1 remain unchanged. The assignment of the auxiliary flags is not changed.

This function is only used to convert the FB 112 of an existing S5 program to a function of an S7 program programmable controller.

4.4.13 FC 124 - 10 to Power n - 10 high n

Description

The function FC 124 expects the input value in ACCU 1 as a floating point number.

1. The input value DWORD = DW#16#0000 0000 is treated the same way as the floating point value zero (REAL = +0.0000000e+00 in accordance with DWORD = DW#16#8000 0000).
2. The function also stores the result in ACCU 1 as a floating point number.
3. If the calculation is carried out correctly, the RLO *ENO* is FALSE after the function has been called up.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	INPUT	BOOL	I, Q, M, D, L	■ Enable – TRUE: activates the function – FALSE: deactivates the function
ENO	OUTPUT	BOOL	I, Q, M, D, L	■ Status – TRUE: function executed with error

Error information

In the event of an error, if the input value is not within the range from -0.3823079e+02 ... REAL = + 0.3823080e+02 (than the value would be outside the number range), the function sets the RLO *ENO* to signal state TRUE. In this case, the contents of ACCU 1 remain unchanged. The assignment of the auxiliary flags is not changed.

This function is only used to convert the FB 113 of an existing S5 program to a function of an S7 program programmable controller.

4.4.14 FC 125 - ACCU 2 to Power ACCU 1 - ACCU 2 high ACCU 1

Description

The function FC 125 expects both the input value for the base in ACCU 2 and the input value for the exponent in ACCU 1 as floating point numbers.

1. The input value for the base must be positive.

   An input value DWORD = DW#16#0000 0000 is treated the same way as the floating point value zero (REAL = +0.0000000e+00 in accordance with DWORD = DW#16#8000 0000).

   For zero high zero the result is zero.
2. The function also stores the result in ACCU 1 as a floating point number.
3. If the calculation is carried out correctly, the RLO *ENO* is FALSE after the function has been called up.

Parameters

Parameter	Declaration	Data Type	Memory Area	Description
EN	INPUT	BOOL	I, Q, M, D, L	■ Enable – TRUE: activates the function – FALSE: deactivates the function
ENO	OUTPUT	BOOL	I, Q, M, D, L	■ Status – TRUE: function executed with error

Error information

If the RLO *ENO* is TRUE, one of the following errors has occurred:

- the input value for the base is less than zero
- a number range overflow occurred during calculation of the function

In the event of an error, the contents of ACCU 1 and 2 remain unchanged.

This function is only used to convert the FB 114 of an existing S5 program to a function of an S7 program programmable controller.

4.5 PID Control

4.5.1 FB 41 - CONT_C - Continuous control

Description

FB 41 CONT_C is used to control technical processes with continuous input and output variables. During parameter assignment, you can activate or deactivate subfunctions of the PID controller to adapt the controller to the process.

Parameters

Parameter	Declaration	Data Type	Description
COM_RST	INPUT	BOOL	COMPLETE RESTART ■ The block has a complete restart routine that is processed when the input *COM_RST* is set. ■ Default: FALSE
MAN_ON	INPUT	BOOL	MANUAL VALUE ON ■ If the input *MAN_ON* is set, the control loop is interrupted. A manual value is set as the manipulated value. ■ Default: TRUE
PVPER_ON	INPUT	BOOL	PROCESS VARIABLE PERIPHERY ON ■ If the process variable is read from the I/Os, the input *PV_PER* must be connected to the I/Os and the input *PVPER_ON* must be set. ■ Default: FALSE
P_SEL	INPUT	BOOL	PROPORTIONAL ACTION ON ■ The PID actions can be activated or deactivated individually in the PID algorithm. The P action is on when the input *P_SEL* is set. ■ Default: TRUE

Parameter	Declaration	Data Type	Description
I_SEL	INPUT	BOOL	INTEGRAL ACTION ON ■ The PID actions can be activated or deactivated individually in the PID algorithm. The I action is on when the input *I_SEL* is set. ■ Default: TRUE
INT_HOLD	INPUT	BOOL	INTEGRAL ACTION HOLD ■ The output of the integrator can be "frozen" by setting the input *INT_HOLD*. ■ Default: FALSE
I_ITL_ON	INPUT	BOOL	INITIALIZATION OF THE INTEGRAL ACTION ■ The output of the integrator can be connected to the input *I_ITL_VAL* by setting the input *I_ITL_ON*. ■ Default: FALSE
D_SEL	INPUT	BOOL	DERIVATIVE ACTION ON ■ The PID actions can be activated or deactivated individually in the PID algorithm. The D action is on when the input *D_SEL* is set. ■ Default: FALSE
CYCLE	INPUT	TIME	SAMPLE TIME ■ The time between the block calls must be constant. The *CYCLE* input specifies the time between block calls. ■ Default: T#1s ■ Range of Values: ≥ 1ms
SP_INT	INPUT	REAL	INTERNAL SETPOINT ■ The *SP_INT* input is used to specify a setpoint. ■ Default: 0.0 ■ Range of Values: -100.0...100. 0 (%) or phys. value[1]
PV_IN	INPUT	REAL	PROCESS VARIABLE IN ■ An initialization value can be set at the *PV_IN* input or an external process variable in floating point format can be connected. ■ Default: 0.0 ■ Range of Values: -100.0...100. 0 (%) or phys. value[1]
PV_PER	INPUT	WORD	PROCESS VARIABLE PERIPHERY ■ The process variable in the I/O format is connected to the controller at the *PV_PER* input. ■ Default: W#16#0000
MAN	INPUT	REAL	MANUAL VALUE ■ The *MAN* input is used to set a manual value using the operator interface functions. ■ Default: 0.0 ■ Range of Values: -100.0...100. 0 (%) or phys. value[2]

Parameter	Declaration	Data Type	Description
GAIN	INPUT	REAL	PROPORTIONAL GAIN ■ The *GAIN* input specifies the controller gain. ■ Default: 2.0 ■ Range of Values: ≥ CYCLE
TI	INPUT	TIME	RESET TIME ■ The *TI* input determines the time response of the integrator. ■ Default: T#20s ■ Range of Values: ≥ CYCLE
TD	INPUT	TIME	DERIVATIVE TIME ■ The *TD* input determines the time response of the derivative unit. ■ Default: T#10s ■ Range of Values: ≥ CYCLE
TM_LAG	INPUT	TIME	TIME LAG OF THE DERIVATIVE ACTION ■ The algorithm of the D action includes a time lag that can be assigned at the *TM_LAG* input. ■ Default: T#2s ■ Range of Values: ≥ CYCLE/2
DEADB_W	INPUT	REAL	DEAD BAND WIDTH ■ A dead band is applied to the error. The *DEADB_W* input determines the size of the dead band. ■ Default: 0.0 ■ Range of Values: ≥ 0.0 (%) or phys. value[1]
LMN_HLM	INPUT	REAL	MANIPULATED VALUE HIGH LIMIT ■ The manipulated value is always limited by an upper and lower limit. The *LMN_HLM* input specifies the upper limit. ■ Default: 100.0 ■ Range of Values: *LMN_LLM* ...100.0 (%) or phys. value[2]
LMN_LLM	INPUT	REAL	MANIPULATED VALUE LOW LIMIT ■ The manipulated value is always limited by an upper and lower limit. The *LMN_LLM* input specifies the lower limit. ■ Default: 0.0 ■ Range of Values: -100.0... *LMN_HLM* (%) or phys. value[2]
PV_FAC	INPUT	REAL	PROCESS VARIABLE FACTOR ■ The *PV_FAC* input is multiplied by the process variable. The input is used to adapt the process variable range. ■ Default: 1.0

Parameter	Declaration	Data Type	Description
PV_OFF	INPUT	REAL	PROCESS VARIABLE OFFSET ■ The *PV_OFF* input is added to the process variable. The input is used to adapt the process variable range. ■ Default: 0.0
LMN_FAC	INPUT	REAL	MANIPULATED VALUE FACTOR ■ The *LMN_FAC* input is multiplied by the manipulated value. The input is used to adapt the manipulated value range. ■ Default: 1.0
LMN_OFF	INPUT	REAL	MANIPULATED VALUE OFFSET ■ The *LMN_OFF* is added to the manipulated value. The input is used to adapt the manipulated value range. ■ Default: 0.0
I_ITLVAL	INPUT	REAL	INITIALIZATION VALUE OF THE INTEGRAL ACTION ■ The output of the integrator can be set at input *I_ITL_ON*. The initialization value is applied to the input *I_ITLVAL*. ■ Default: 0.0 ■ Range of Values: -100.0...100. 0 (%) or phys. value^2
DISV	INPUT	REAL	DISTURBANCE VARIABLE ■ For feed forward control, the disturbance variable is connected to input *DISV*. ■ Default: 0.0 ■ Range of Values: -100.0...100. 0 (%) or phys. value^2
LMN	OUTPUT	REAL	MANIPULATED VALUE ■ The effective manipulated value is output in floating point format at the *LMN* output. ■ Default: 0.0
LMN_PER	OUTPUT	WORD	MANIPULATED VALUE PERIPHERY ■ The manipulated value in the I/O format is connected to the controller at the *LMN_PER* output. ■ Default: W#16#0000
QLMN_HLM	OUTPUT	BOOL	HIGH LIMIT OF MANIPULATED VALUE REACHED ■ The manipulated value is always limited to an upper and lower limit. The output *QLMN_HLM* indicates that the upper limit has been exceeded. ■ Default: FALSE
QLMN_LLM	OUTPUT	BOOL	LOW LIMIT OF MANIPULATED VALUE REACHED ■ The manipulated value is always limited to an upper and lower limit. The output *QLMN_LLM* indicates that the lower limit has been exceeded. ■ Default: FALSE

Parameter	Declaration	Data Type	Description
LMN_P	OUTPUT	REAL	PROPORTIONALITY COMPONENT ■ The *LMN_P* output contains the proportional component of the manipulated variable. ■ Default: 0.0
LMN_I	OUTPUT	REAL	INTEGRAL COMPONENT ■ The *LMN_I* output contains the integral component of the manipulated value. ■ Default: 0.0
LMN_D	OUTPUT	REAL	DERIVATIVE COMPONENT ■ The *LMN_D* output contains the derivative component of the manipulated value.. ■ Default: 0.0
PV	OUTPUT	REAL	PROCESS VARIABLE ■ The effective process variable is output at the *PV* output. ■ Default: 0.0
ER	OUTPUT	REAL	ERROR SIGNAL ■ The effective error is output at the *ER* output. ■ Default: 0.0
1) Parameters in the setpoint and process variable branches with the same unit 2) Parameters in the manipulated value branch with the same unit			

Application

You can use the controller as a PID fixed setpoint controller or in multi-loop controls as a cascade, blending or ratio controller. The functions of the controller are based on the PID control algorithm of the sampling controller with an analog signal, if necessary extended by including a pulse generator stage to generate pulse duration modulated output signals for two or three step controllers with proportional actuators.

Apart from the functions in the setpoint and process value branches, the FB implements a complete PID controller with continuous manipulated variable output and the option of influencing the manipulated value manually.

Setpoint Branch

The setpoint is entered in floating-point format at the *SP_INT* input.

Process Variable Branch

The process variable can be input in the peripheral (I/O) or floating-point format. The *CRP_IN* function converts the *PV_PER* peripheral value to a floating-point format of -100 to +100 % according to the following formula:

$$Output\ of\ CPR\_IN = PV\_PER * \frac{100}{27648}$$

The *PV_NORM* function normalizes the output of *CRP_IN* according to the following formula:

$$Output\ of\ PV\_NORM = (Output\ of\ CPR\_IN) * PV\_FAC + PV\_OFF$$

PV_FAC has a default of 1 and *PV_OFF* a default of 0.

Error Signal

The difference between the setpoint and process variable is the error signal. To suppress a small constant oscillation due to the manipulated variable quantization (for example in pulse duration modulation with PULSEGEN), a dead band is applied to the error signal (DEADBAND). If *DEADB_W* = 0, the dead band is switched off.

PID Algorithm

The PID algorithm operates as a position algorithm. The proportional, integral (*INT*), and derivative (*DIF*) actions are connected in parallel and can be activated or deactivated individually. This allows P, PI, PD, and PID controllers to be configured. Pure I and D controllers are also possible.

Manual Value

It is possible to switch over between a manual and an automatic mode. In the manual mode, the manipulated variable is corrected to a manually selected value. The integrator (*INT*) is set internally to *LMN* - *LMN_P* - *DISV* and the derivative unit (*DIF*) to 0 and matched internally. This means that a switchover to the automatic mode does not cause any sudden change in the manipulated value.

Manipulated Value

The manipulated value can be limited to a selected value using the *LMNLIMIT* function. Signaling bits indicate when a limit is exceeded by the input variable. The *LMN_NORM* function normalizes the output of *LMNLIMIT* according to the following formula:

$$LMN = (Output\ of\ LMNLIMIT) * LMN\_FAC + LMN\_OFF$$

LMN_FAC has the default 1 and *LMN_OFF* the default 0.

The manipulated value is also available in the peripheral format. The *CRP_OUT* function converts the floating-point value *LMN* to a peripheral value according to the following formula:

$$LMN\_PER = LMN * \frac{27648}{100}$$

Feedforward Control

A disturbance variable can be fed forward at the *DISV* input.

Modes

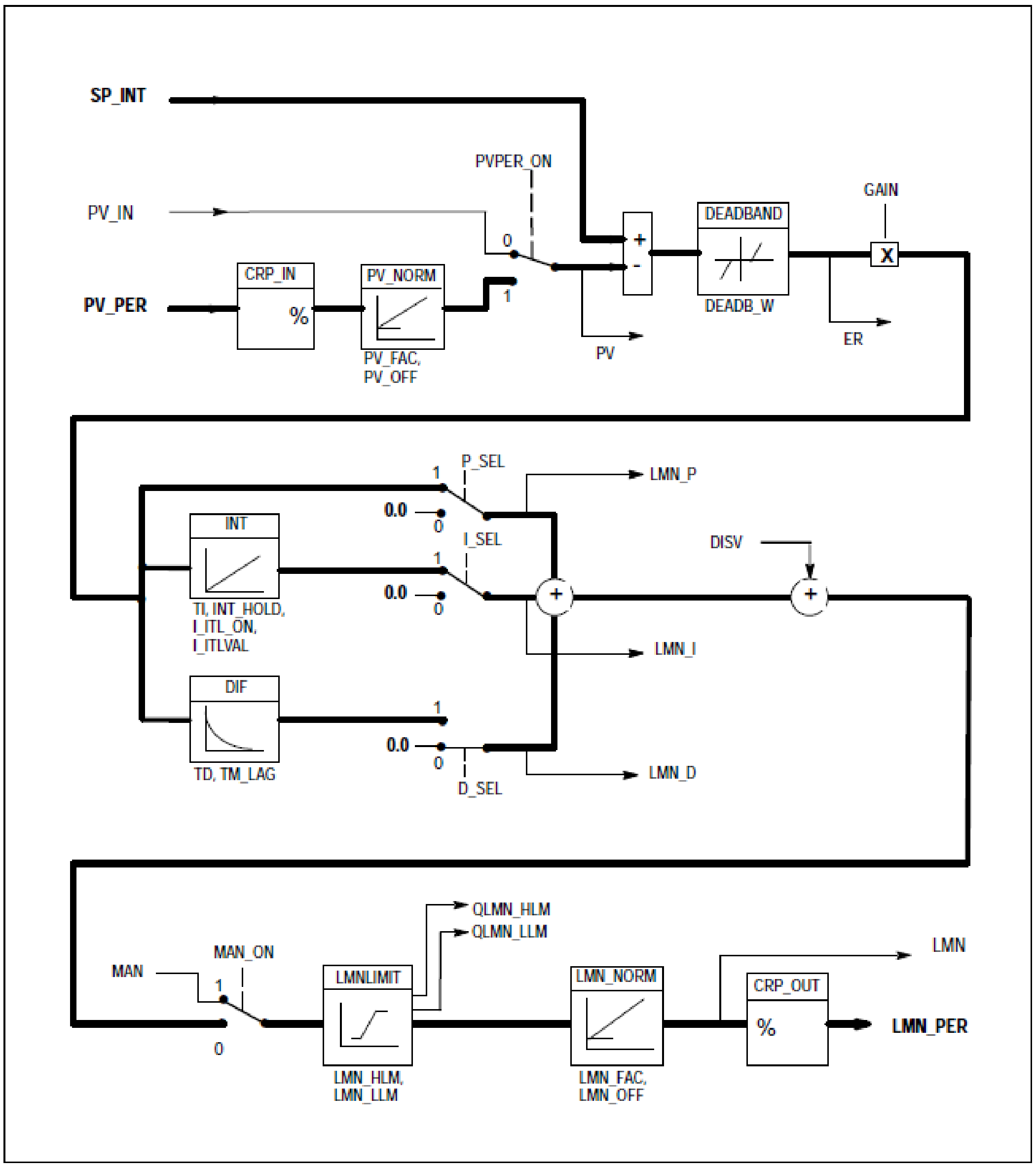
Complete Restart/Restart

- FB 41 CONT_C has a complete restart routine that is run through when the input parameter *COM_RST* = TRUE is set.
- During startup, the integrator is set internally to the initialization value *I_ITVAL*. When it is called in a cyclic interrupt priority class, it then continues to work starting at this value.
- All other outputs are set to their default values.

Error Information

The block does not check for errors, so no error Information is output.

Block Diagram



4.5.2 FB 42 - CONT_S - Step Control

Description

FB42 CONT_S is used to control technical processes with digital manipulated value output signals for integrating actuators. During parameter assignment, you can activate or deactivate subfunctions of the PI step controller to adapt the controller to the process.

Parameter

Parameter	Declaration	Data Type	Description
COM_RST	INPUT	BOOL	COMPLETE RESTART ■ The block has a complete restart routine that is processed when the input *COM_RST* is set. ■ Default: FALSE
LMNR_HS	INPUT	BOOL	HIGH LIMIT SIGNAL OF REPEATED MANIPULATED VALUE ■ The "actuator at upper limit stop" signal is connected to the *LMNR_HS* input. – *LMNR_HS* = TRUE means the actuator is at upper limit stop. ■ Default: FALSE
LMNR_LS	INPUT	BOOL	LOW LIMIT SIGNAL OF REPEATED MANIPULATED VALUE ■ The "actuator at lower limit stop" signal is connected to the *LMNR_LS* input. – *LMNR_LS* = TRUE means the actuator is at lower limit stop. ■ Default: FALSE
LMNS_ON	INPUT	BOOL	MANIPULATED SIGNALS ON ■ The actuating signal processing is switched to manual at the *LMNS_ON* input.. ■ Default: FALSE
LMNUP	INPUT	BOOL	MANIPULATED SIGNALS UP ■ With manual actuating value signals, the output signal *QLMNUP* is set at the input *LMNUP*. ■ Default: FALSE
LMNDN	INPUT	BOOL	MANIPULATED SIGNALS DOWN ■ With manual actuating value signals, the output signal *QLMNDN* is set at the input *LMNDN*. ■ Default: FALSE
PVPER_ON	INPUT	BOOL	PROCESS VARIABLE PERIPHERY ON ■ If the process variable is read in from the I/Os, the input *PV_PER* must be connected to the I/Os and the input *PVPER_ON* must be set. ■ Default: FALSE
CYCLE	INPUT	TIME	SAMPLE TIME ■ The time between the block calls must be constant. The *CYCLE* input specifies the time between block calls. ■ Default: T#1s ■ Range of Values: ≥ 1ms
SP_INT	INPUT	REAL	INTERNAL SETPOINT ■ The *SP_INT* input is used to specify a setpoint. ■ Default: 0.0 ■ Range of Values: -100.0...100. 0 (%) or phys. value[1]

Parameter	**Declaration**	**Data Type**	**Description**
PV_IN	INPUT	REAL	PROCESS VARIABLE IN ■ An initialization value can be set at the *PV_IN* input or an external process variable in floating point format can be connected. ■ Default: 0.0 ■ Range of Values: -100.0...100. 0 (%) or phys. value[1]
PV_PER	INPUT	WORD	PROCESS VARIABLE PERIPHERY ■ The process variable in the I/O format is connected to the controller at the *PV_PER* input. ■ Default: W#16#0000
GAIN	INPUT	REAL	PROPORTIONAL GAIN ■ The *GAIN* input sets the controller gain. ■ Default: 2.0 ■ Range of Values: ≥ *CYCLE*
TI	INPUT	TIME	RESET TIME ■ The *TI* input determines the time response of the integrator. ■ Default: T#20s ■ Range of Values: ≥ *CYCLE*
DEADB_W	INPUT	REAL	DEAD BAND WIDTH ■ A dead band is applied to the error. The *DEADB_W* input determines the size of the dead band. ■ Default: 1.0 ■ Range of Values: 0.0...100.0 (%) or phys. value[1]
PV_FAC	INPUT	REAL	PROCESS VARIABLE FACTOR ■ The *PV_FAC* input is multiplied by the process variable. The input is used to adapt the process variable range. ■ Default: 1.0
PV_OFF	INPUT	REAL	PROCESS VARIABLE OFFSET ■ The *PV_OFF* input is added to the process variable. The input is used to adapt the process variable range. ■ Default: 0.0
PULSE_TM	INPUT	TIME	MINIMUM PULSE TIME ■ A minimum pulse duration can be assigned with the parameter *PULSE_TM*. ■ Default: T#3s ■ Range of Values: ≥ *CYCLE*
BREAK_TM	INPUT	TIME	MINIMUM BREAK TIME ■ A minimum break duration can be assigned with the parameter *BREAK_TM*. ■ Default: T#3s ■ Range of Values: ≥ *CYCLE*

Parameter	Declaration	Data Type	Description
MTR_TM	INPUT	TIME	MOTOR MANIPULATED VALUE ■ The time required by the actuator to move from limit stop to limit stop is entered at the *MTR_TM* parameter. ■ Default: T#30s ■ Range of Values: ≥ *CYCLE*
DISV	INPUT	REAL	DISTURBANCE VARIABLE ■ For feed forward control, the disturbance variable is connected to input *DISV*. ■ Default: 0.0 ■ Range of Values: -100.0...100. 0 (%) or phys. value^2
QLMNUP	OUTPUT	BOOL	MANIPULATED SIGNAL UP ■ If the output *QLMNUP* is set, the actuating valve is opened. ■ Default: FALSE
QLMNDN	OUTPUT	BOOL	MANIPULATED SIGNAL DOWN ■ If the output *QLMNDN* is set, the actuating valve is opened. ■ Default: FALSE
PV	OUTPUT	REAL	PROCESS VARIABLE ■ The effective process variable is output at the *PV* output. ■ Default: 0.0
ER	OUTPUT	REAL	ERROR SIGNAL ■ The effective error is output at the *ER* output. ■ Default: 0.0
1) Parameters in the setpoint and process variable branches with the same unit 2) Parameters in the manipulated value branch with the same unit			

Application

You can use the controller as a PI fixed setpoint controller or in secondary control loops in cascade, blending or ratio controllers, however not as the primary controller. The functions of the controller are based on the PI control algorithm of the sampling controller supplemented by the functions for generating the binary output signal from the analog actuating signal.

Apart from the functions in the process value branch, the FB implements a complete PI controller with a digital manipulated value output and the option of influencing the manipulated value manually. The step controller operates without a position feedback signal.

Setpoint Branch

The setpoint is entered in floating-point format at the *SP_INT* input.

Process Variable Branch

The process variable can be input in the peripheral (I/O) or floating-point format. The *CRP_IN* function converts the PV_PER peripheral value to a floating-point format of -100 to +100 % according to the following formula:

$$Output\ of\ CPR\_IN = PV\_PER * \frac{100}{27648}$$

The *PV_NORM* function normalizes the Output of *CRP_IN* following formula:

$$Output\ of\ PV\_NORM = (Output\ of\ CPR\_IN) * PV\_FAC + PV\_OFF$$

PV_FAC has a default of 1 and *PV_OFF* a default of 0.

Error Signal

The difference between the setpoint and process variable is the error signal. To suppress a small constant oscillation due to the manipulated variable quantization (for example due to a limited resolution of the manipulated value by the actuator valve), a dead band is applied to the error signal (DEADBAND). If *DEADB_W* = 0, the dead band is switched off.

PI Step Algorithm

The FB operates without a position feedback signal. The I action of the PI algorithm and the assumed position feedback signal are calculated in one integrator (INT) and compared with the remaining P action as a feedback value. The difference is applied to a three-step element (THREE_ST) and a pulse generator (PULSEOUT) that creates the pulses for the actuator. The switching frequency of the controller can be reduced by adapting the threshold on of the three-step element.

Feedforward Control

A disturbance variable can be fed forward at the *DISV* input.

Modes

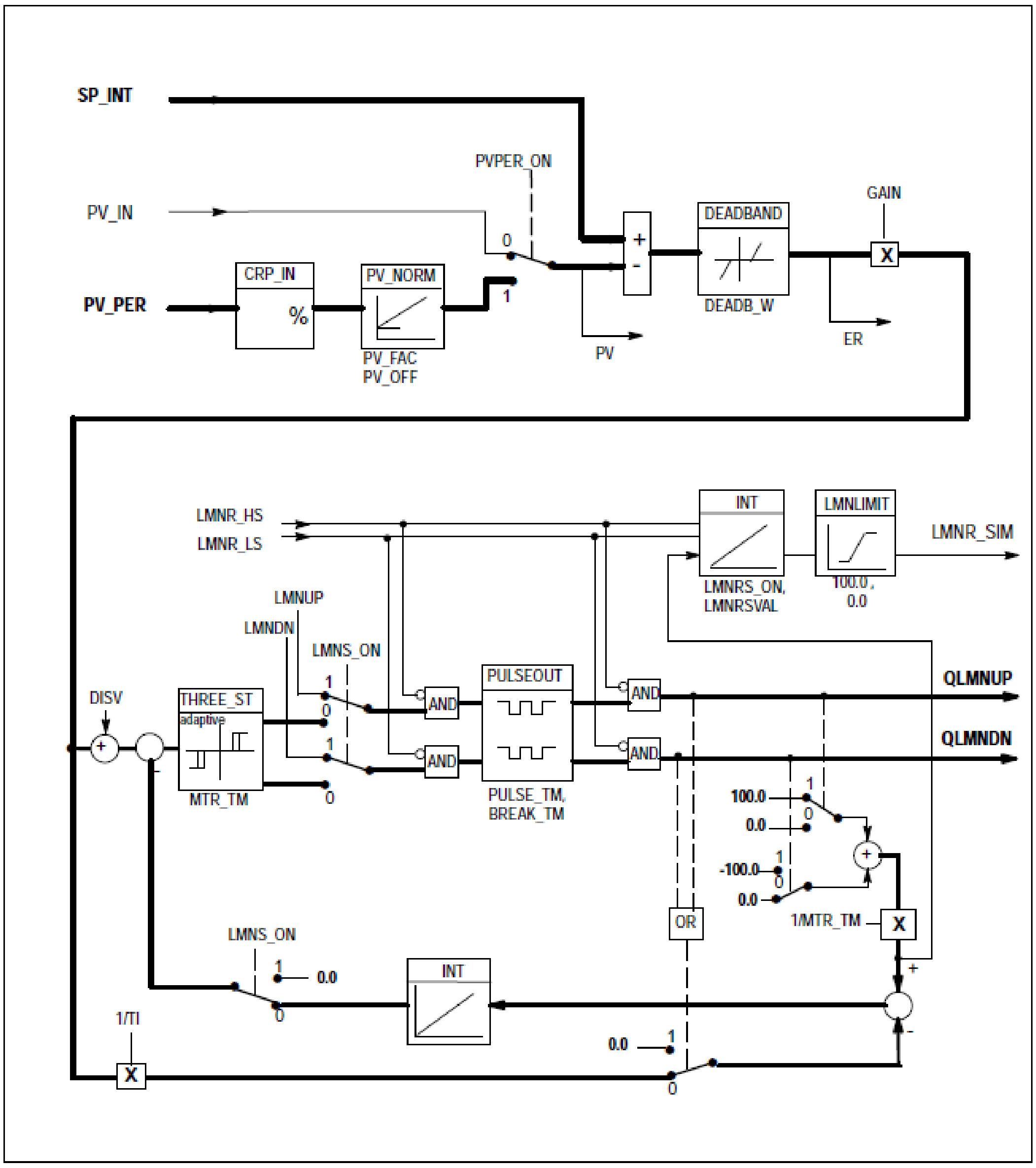
Complete Restart/Restart

- FB42 CONT_S has a complete restart routine that is run through when the input parameter *COM_RST* = TRUE is set.
- All other outputs are set to their default values.

Error Information

The block does not check for errors, so no error Information is output.

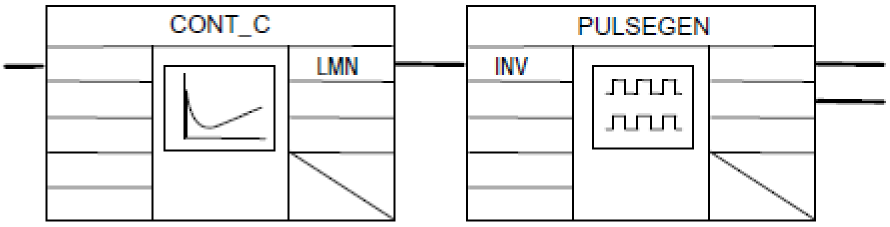
Block Diagram



4.5.3 FB 43 - PULSGEN - Pulse generation

Description

FB 43 PULSEGEN is used to structure a PID controller with pulse output for proportional actuators. Using FB43, PID two or three step controllers with pulse duration modulation can be configured. The function is normally used in conjunction with the continuous controller CONT_C.



Parameters

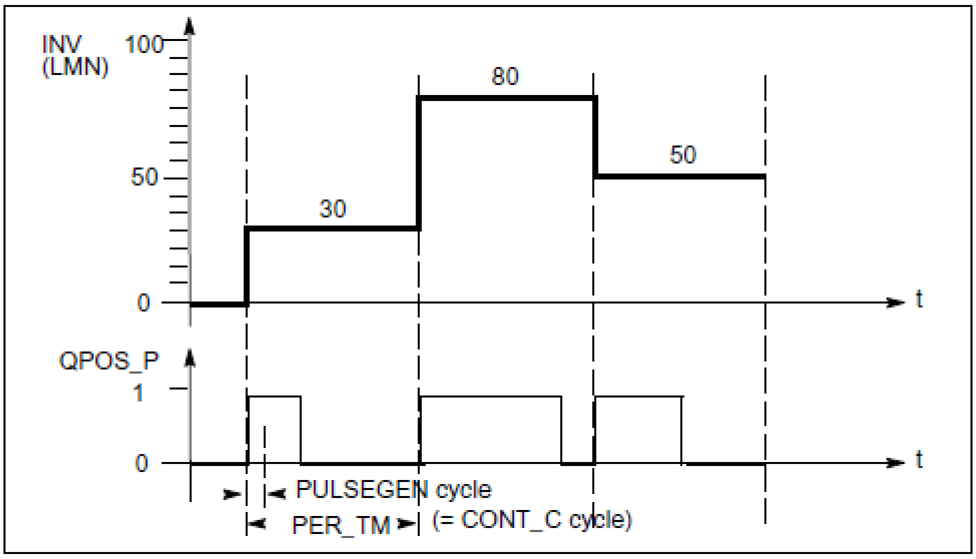
Parameter	Declaration	Data Type	Description
INV	INPUT	REAL	INPUT VARIABLE ■ An analog manipulated value is connected to the input parameter *INV*. ■ Default: 0.0 ■ Range of Values: -100.0...100.0 (%)
PER_TM	INPUT	TIME	PERIOD TIME ■ The constant period of pulse duration modulation is input with the *PER_TM* input parameter. This corresponds to the sampling time of the controller. The ratio between the sampling time of the pulse generator and the sampling time of the controller determines the accuracy of the pulse duration modulation. ■ Default: T#1s ■ Range of Values: ≥ 20**CYCLE*
P_B_TM	INPUT	TIME	MINIMUM PULSE/BREAK TIME ■ A minimum pulse or minimum break time can be assigned at the input parameters *P_B_TM*. ■ Default: T#50ms ■ Range of Values: ≥ *CYCLE*
RATIOFAC	INPUT	REAL	RATIO FACTOR ■ The input parameter *RATIOFAC* can be used to change the ratio of the duration of negative to positive pulses. In a thermal process, this would, for example, allow different time constants for heating and cooling to be compensated (for example, in a process with electrical heating and water cooling). ■ Default: 1.0 ■ Range of Values: 0.1 ...10.0
STEP3_ON	INPUT	BOOL	THREE STEP CONTROL ON ■ The *STEP3_ON* input parameter activates this mode. In three-step control, both output signals are active. ■ Default: TRUE
ST2BI_ON	INPUT	BOOL	TWO STEP CONTROL FOR BIPOLAR MANIPULATED VALUE RANGE ON ■ With the input parameter *ST2BI_ON* you can select between the modes "two-step control for bipolar manipulated value" and "two-step control for monopolar manipulated value range". The parameter *STEP3_ON* = FALSE must be set. ■ Default: FALSE
MAN_ON	INPUT	BOOL	MANUAL MODE ON ■ By setting the input parameter *MAN_ON*, the output signals can be set manually. ■ Default: FALSE

Parameter	Declaration	Data Type	Description
POS_P_ON	INPUT	BOOL	POSITIVE MODE ON ■ In the manual mode with three-step control, the output signal *QPOS_P* can be set at the input parameter *POS_P_ON*. In the manual mode with two-step control, *QNEG_P* is always set inversely to *QPOS_P*. ■ Default: FALSE
NEG_P_ON	INPUT	BOOL	NEGATIVE PULSE ON ■ In the manual mode with three-step control, the output signal *QNEG_P* can be set at the input parameter *NEG_P_ON*. In the manual mode with two-step control, *QNEG_P* is always set inversely to *QPOS_P*. ■ Default: FALSE
SYN_ON	INPUT	BOOL	SYNCHRONISATION ON ■ By setting the input parameter *SYN_ON*, it is possible to synchronize automatically with the block that updates the input variable *INV*. This ensures that a changing input variable is output as quickly as possible as a pulse. ■ Default: TRUE
COM_RST	INPUT	BOOL	COMPLETE RESTART ■ The block has a complete restart routine that is processed when the *COM_RST* input is set. ■ Default: FALSE
CYCLE	INPUT	TIME	SAMPLE TIME ■ The time between block calls must be constant. The *CYCLE* input specifies the time between block calls. ■ Default: T#10ms ■ Range of Values: ≥ 1ms
QPOS_P	OUTPUT	BOOL	OUTPUT POSITIVE PULSE ■ The output parameter *QPOS_P* is set when a pulse is to be output. In three-step control, this is always the positive pulse. In two-step control, *QNEG_P* is always set inversely to *QPOS_P*. ■ Default: FALSE
QNEG_P	OUTPUT	BOOL	OUTPUT NEGATIVE PULSE ■ The output parameter *QNEG_P* is set when a pulse is to be output. In three-step control, this is always the negative pulse. In two-step control, *QNEG_P* is always set inversely to *QPOS_P*. ■ Default: FALSE

The values of the input parameters are not limited in the block. There is no parameter check.

Application

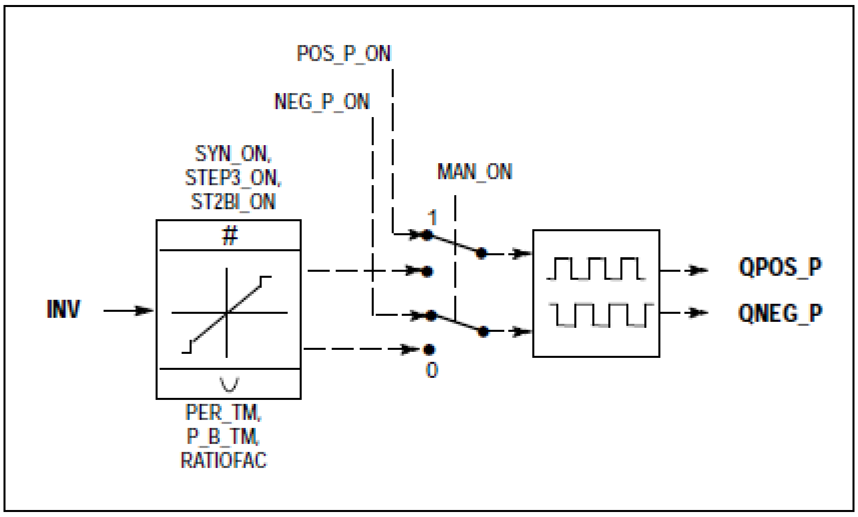
The PULSEGEN function transforms the input variable *INV* (= manipulated value of the PID controller) by modulating the pulse duration into a pulse train with a constant period, corresponding to the cycle time at which the input variable is updated and which must be assigned in *PER_TM*. The duration of a pulse per period is proportional to the input variable. The cycle assigned to *PER_TM* is not identical to the processing cycle of the FB PULSEGEN. The *PER_TM* cycle is made up of several processing cycles of FB PULSEGEN, whereby the number of FB PULSEGEN calls per *PER_TM* cycle is the yardstick for the accuracy of the pulse duration modulation.



An input variable of 30% and 10 FB PULSEGEN calls per *PER_TM* means the following:

- "1" at the *QPOS* output for the first three calls of FB PULSEGEN (30% of 10 calls)
- "0" at the *QPOS* output for seven further calls of FB PULSEGEN (70% of 10 calls)

Block Diagram



Accuracy of the Manipulated Value

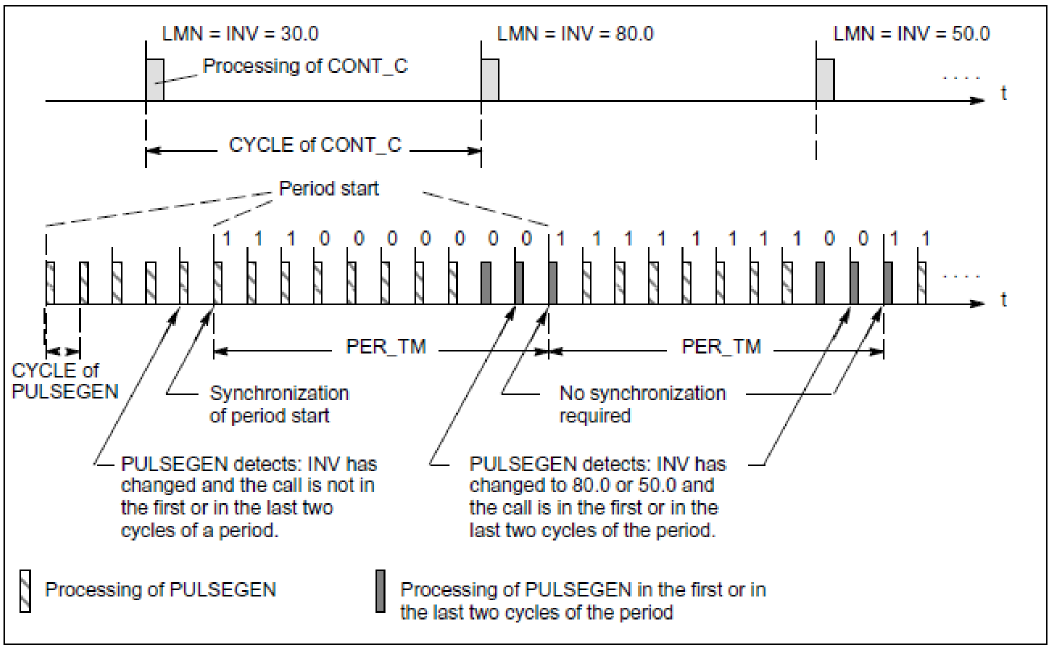
With a "sampling ratio" of 1:10 (CONT_C calls to PULSEGEN calls) the accuracy of the manipulated value in this example is restricted to 10 %, in other words, set input values *INV* can only be simulated by a pulse duration at the *QPOS* output in steps of 10 %. The accuracy is increased as the number of FB PULSEGEN calls per CONT_C call is increased. If PULSEGEN is called, for example 100 times more often than CONT_C, a resolution of 1 % of the manipulated value range is achieved.

The call frequency must be programmed by the user.

Automatic Synchronization

It is possible to synchronize the pulse output with the block that updates the input variable *INV* (for example CONT_C). This ensures that a change in the input variable is output as quickly as possible as a pulse. The pulse generator evaluates the input value *INV* at intervals corresponding to the period *PER_TM* and converts the value into a pulse signal of corresponding length. Since, however, *INV* is usually calculated in a slower cyclic interrupt class, the pulse generator should start the conversion of the discrete value into a pulse signal as soon as possible after the updating of *INV*. To allow this, the block can synchronize the start of the period using the following procedure:

- If *INV* changes and if the block call is not in the first or last two call cycles of a period, the synchronization is performed. The pulse duration is recalculated and in the next cycle is output with a new period.


LMN = INV = 30.0
LMN = INV = 80.0
LMN = INV = 50.0
Processing of CONT_C
t
CYCLE of CONT_C
Period start
1 1 1 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 1
t
PER_TM
PER_TM
CYCLE of PULSEGEN
Synchronization of period start
No synchronization required
PULSEGEN detects: INV has changed and the call is not in the first or in the last two cycles of a period.
PULSEGEN detects: INV has changed to 80.0 or 50.0 and the call is in the first or in the last two cycles of the period.
Processing of PULSEGEN
Processing of PULSEGEN in the first or in the last two cycles of the period

The automatic synchronization can be disabled at the *SYN_ON* input (= FALSE).

With the beginning of a new period, the old value of INV (in other words, of LMN) is simulated in the pulse signal more or less accurately following the synchronization.

Modes

Depending on the parameters assigned to the pulse generator, PID controllers with a three-step output or with a bipolar or monopolar two-step output can be configured. The following table illustrates the setting of the switch combinations for the possible modes.

Mode	**Switch**		
	MAN_ON	**STEP3_ON**	**ST2BI_ON**
Three-step control	FALSE	TRUE	Any
Two-step control with bipolar control range (-100 % to +100 %)	FALSE	FALSE	TRUE
Two-step control with monopolar control range (0 % ... 100 %)	FALSE	FALSE	FALSE
Manual mode	TRUE	Any	Any

Three-Step Control

In the three-step control mode, the actuating signal can adopt three states. The values of the binary output signals *QPOS_P* and *QNEG_P* are assigned to the statuses of the actuator. The table shows the example of a temperature control:

Output signal	**Actuator**		
	Heat	**Off**	**Cool**
QPOS_P	TRUE	FALSE	FALSE
QNEG_P	FALSE	FALSE	TRUE

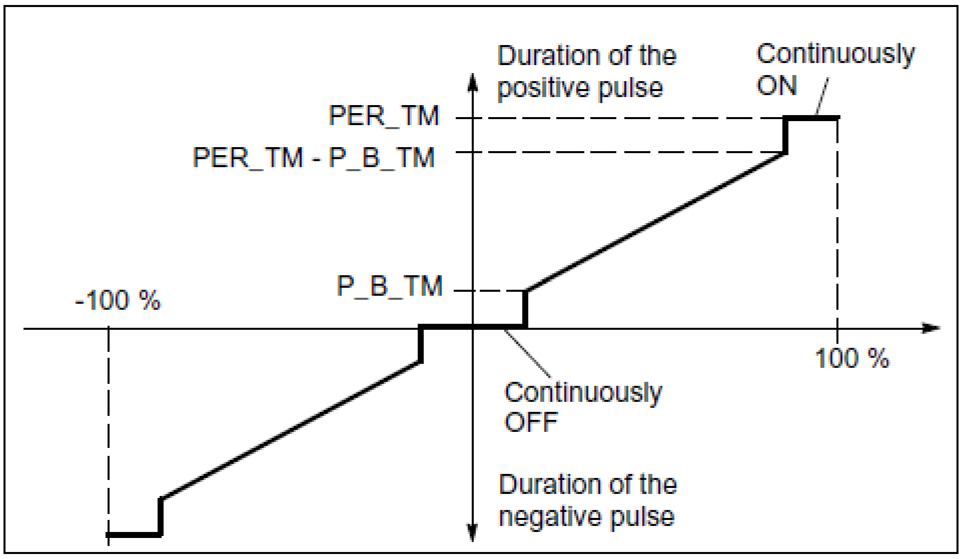
Based on the input variable, a characteristic curve is used to calculate a pulse duration. The form of the characteristic curve is defined by the minimum pulse or minimum break time and the ratio factor. The normal value for the ratio factor is 1. The “doglegs” in the curves are caused by the minimum pulse or minimum break times.

- *Minimum Pulse or Minimum Break Time*
  A correctly assigned minimum pulse or minimum break time *P_B_TM* can prevent short on/off times that reduce the working life of switching elements and actuators.

Small absolute values at the input variable LMN that could otherwise generate a pulse duration shorter than P_B_TM are suppressed. Large input values that would generate a pulse duration longer than (PER_TM - P_B_TM) are set to 100 % or -100 %.

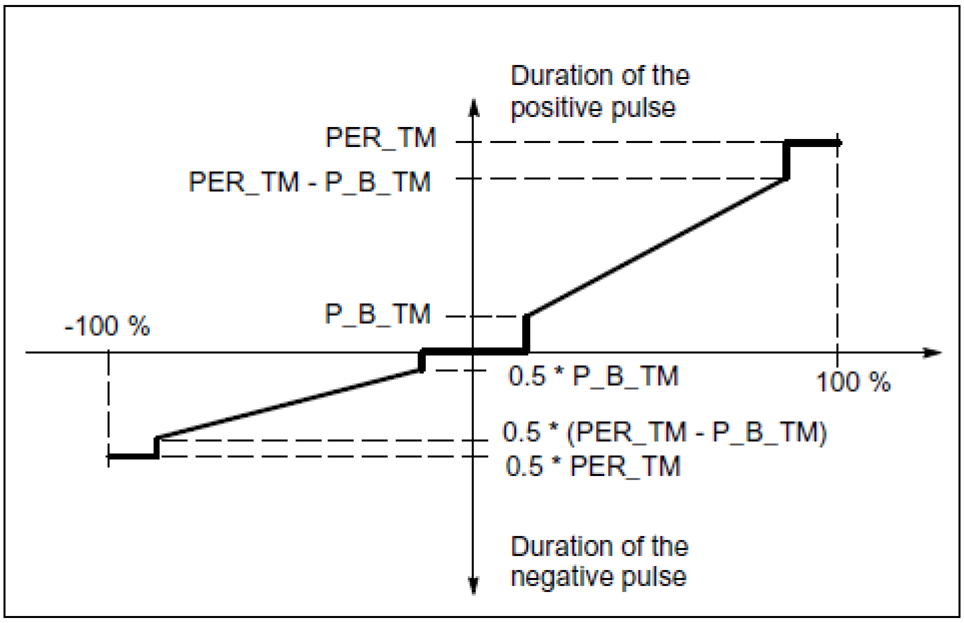
The positive and negative pulse duration is calculated by multiplying the input variable (in %) with the period time:

$$Pulse\ duration = \frac{INV}{100} * PER\_TM$$



Three-Step Control Asymmetrical

Using the ratio factor *RATIOFAC*, the ratio of the duration of positive to negative pulses can be changed. In a thermal process, for example, this would allow different system time constants for heating and cooling. The ratio factor also influences the minimum pulse or minimum break time. A ratio factor < 1 means that the threshold value for negative pulses is multiplied by the ratio factor.



- *Ratio Factor < 1*
  The pulse duration at the negative pulse output calculated from the input variable multiplied by the period time is reduced by the ratio factor.

$$Duration\ of\ the\ positive\ pulse = \frac{INV}{100} * PER\_TM$$

$$Duration\ of\ the\ negative\ pulse = \frac{INV}{100} * PER\_TM * RATIOFAC$$

- *Ratio Factor > 1*
  The pulse duration at the positive pulse output calculated from the input variable multiplied by the period time is reduced by the ratio factor.

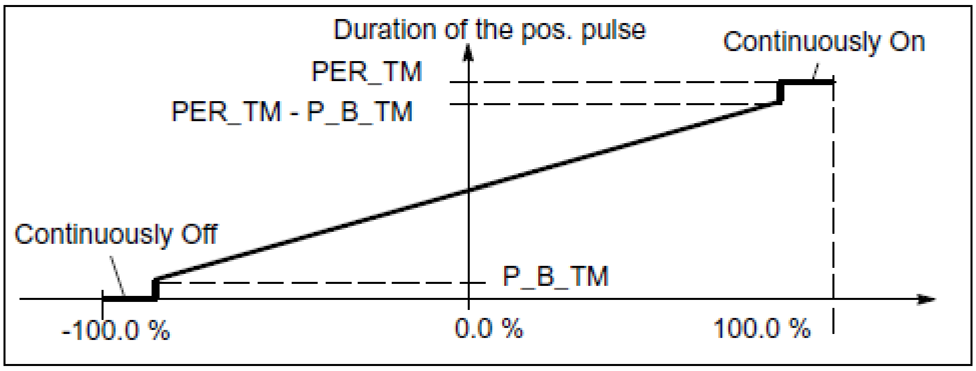
$$Duration\ of\ the\ negative\ pulse = \frac{INV}{100} * PER\_TM$$

$$Duration\ of\ the\ positive\ pulse = \frac{INV}{100} * \frac{PER\_TM}{RATIOFAC}$$

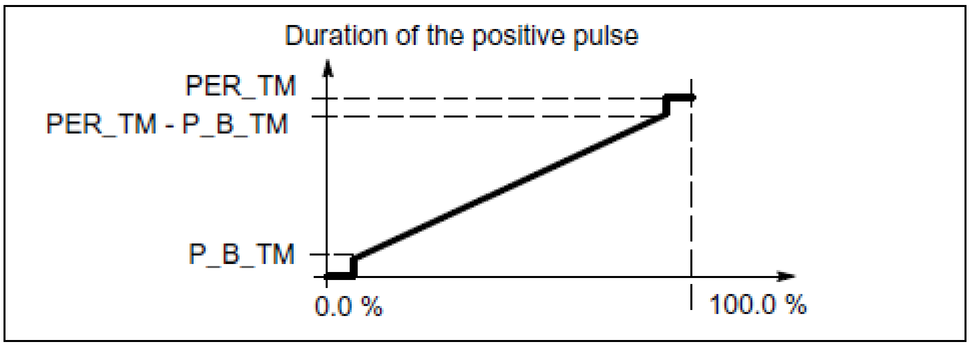
Two-Step Control

In two-step control, only the positive pulse output *QPOS_P* of PULSEGEN is connected to the on/off actuator. Depending on the manipulated value range being used, the two-step controller has a bipolar or a monopolar manipulated value range.

- *Two-Step Control with Bipolar Manipulated Variable Range (-100 % to 100 %)*



- *Two-Step Control with Monopolar Manipulated Variable Range (0 % to 100 %)*



The negated output signal is available at *QNEG_P* if the connection of the two-step controller in the control loop requires a logically inverted binary signal for the actuating pulses.

Pulse	Actuator	
	On	Off
QPOS_P	TRUE	FALSE
QNEG_P	FALSE	TRUE

Manual Mode in Two/ Three-Step Control

In the manual mode (*MAN_ON* = TRUE), the binary outputs of the three-step or two-step controller can be set using the signals *POS_P_ON* and *NEG_P_ON* regardless of *INV*.

	POS_P_ON	NEG_P_ON	QPOS_P	QNEG_P
Three-step control	FALSE	FALSE	FALSE	FALSE
	TRUE	FALSE	TRUE	FALSE
	FALSE	TRUE	FALSE	TRUE
	TRUE	TRUE	FALSE	FALSE
Two-step control	FALSE	Any	FALSE	TRUE
	TRUE	Any	TRUE	FALSE

Modes

Complete Restart/Restart

- During a complete restart, all the signal outputs are set to 0.

Error Information

The block does not check for errors, so no error Information is output.

4.5.4 FB 58 - TCONT_CP - Continuous Temperature Control

Description

FB 58 TCONT_CP is used to control temperature processes with continuous or pulsed control signals. You can set parameters to enable or disable subfunctions of the PID controller and adapt it to the process.

Parameters

Parameter	Declaration	Data Type	Description
PV_IN	INPUT	REAL	PROCESS VARIABLE IN ■ An initialization value can be set at the *PV_IN* input or an external process variable in floating-point format can be connected. ■ Default: 0.0 ■ Dependent on the sensors used
PV_PER	INPUT	WORD	PROCESS VARIABLE PERIPHERY ■ The process variable in the peripheral I/O format is connected to the controller at the *PV_PER* input. ■ Default: 0
DISV	INPUT	REAL	DISTURBANCE VARIABLE ■ For feed forward control, the disturbance variable is connected to the *DISV* input. ■ Default: 0.0
INT_HPOS	INPUT	BOOL	INTEGRAL ACTION HOLD IN POSITIVE DIRECTION ■ The output of the integral action can be blocked in a positive direction. To achieve this, the *INT_HPOS* input must be set to TRUE. In a cascade control, the *INT_HPOS* of the primary controller is interconnected to *QLMN_HLM* of the secondary controller. ■ Default: FALSE

Parameter	Declaration	Data Type	Description
INT_HNEG	INPUT	BOOL	INTEGRAL ACTION HOLD IN NEGATIVE DIRECTION ■ The output of the integral action can be blocked in a positive direction. To achieve this, the *INT_HPOS* input must be set to TRUE. In a cascade control, the *INT_HPOS* of the primary controller is intercon-nected to *QLMN_HLM* of the secondary controller. ■ Default: FALSE
SELECT	INPUT	BOOL	SELECTION OF CALL PID AND PULSE GENERATOR ■ If the pulse generator is activated, there are several ways of calling the PID algorithm and pulse gener-ator: – *SELECT* = 0: The controller is called in a fast cyclic interrupt level and the PID algorithm and pulse generator are processed. – *SELECT* = 1: The controller is called in OB1 and only the PID algorithm is processed. – *SELECT* = 2: The controller is called in a fast cyclic interrupt level and only the pulse gener-ator is processed. – *SELECT* = 3: The controller is called in a slow cyclic interrupt level only the PID algorithm is processed. ■ Default: 0 ■ Range of Values: 0 ... 3
PV	OUTPUT	REAL	PROCESS VARIABLE ■ The effective process variable is output at the *PV* output. ■ Default: 0.0 ■ Range of Values: Dependent on the sensors used
LMN	OUTPUT	REAL	MANIPULATED VALUE ■ The effective value of the manipulated variable is output in floating-point format at the *LMN* output. ■ Default: 0.0
LMN_PER	OUTPUT	WORD	MANIPULATED VALUE PERIPHERY ■ The value of the manipulated variable in the periph-eral format is connected to the controller at the *LMN_PER* output. ■ Default: 0
QPULSE	OUTPUT	BOOL	QUTPUT PULSE SIGNAL ■ The value of the manipulated variable is output pulse duration modulated at the *QPULSE* output. ■ Default: FALSE
QLMN_HLM	OUTPUT	BOOL	HIGH LIMIT OF MANIPULATED VALUE REACHED ■ The value of the manipulated variable is always limited to an upper and lower limit. The *QLMN_HLM* output indicates when the upper limit is exceeded. ■ Default: FALSE

Parameter	Declaration	Data Type	Description
QLMN_LLM	OUTPUT	BOOL	LOW LIMIT OF MANIPULATED VALUE REACHED ■ The value of the manipulated variable is always limited to an upper and lower limit. The *QLMN_LLM* output indicates when the lower limit is exceeded. ■ Default: FALSE
QC_ACT	OUTPUT	BOOL	NEXT CYCLE, THE CONTINUOUS CONTROLLER IS WORKING ■ This parameter indicates whether or not the continuous controller stage will be executed at the next block call (relevant only when *SELECT* has the value 0 or 1). ■ Default: TRUE
CYCLE	INPUT/ OUTPUT	REAL	SAMPLE TIME OF CONTINUOUS CONTROLLER [s] ■ This sets the sampling time for the PID algorithm. The tuner calculates the sampling time in Phase 1 and enters this in *CYCLE*. ■ Default: 0.1s ■ Range of Values: ≥ 1ms
CYCLE_P	INPUT/ OUTPUT	REAL	SAMPLE TIME OF PULSE GENERATOR [s] ■ At this input, you enter the sampling time for the pulse generator stage. FB 58 "TCONT_CP" calculates the sampling time in Phase 1 and enters it in *CYCLE_P*. ■ Default: 0.2s ■ Range of Values: ≥ 1ms
SP_INT	INPUT/ OUTPUT	REAL	INTERNAL SETPOINT ■ The *SP_INT* input is used to specify a setpoint. ■ Default: 0.0 ■ Range of Values: Value range of the process value
MAN	INPUT/ OUTPUT	REAL	MANUAL VALUE ■ The *MAN* input is used to specify a manual value. In automatic mode, it is corrected to the manipulated variable. ■ Default: 0.0
COM_RST	INPUT/ OUTPUT	REAL	COMPLETE RESTART ■ The block has an initialization routine that is processed when the *COM_RST* input is set. ■ Default: FALSE
MAN_ON	INPUT/ OUTPUT	REAL	MANUAL OPERATION ON ■ If the *MAN_ON* input is set, the control loop is interrupted. The MAN manual value is set as the value of the manipulated variable. ■ Default: TRUE

Internal Parameters

Parameter	Declaration	Data type	Description
DEADB_W	INPUT	REAL	DEAD BAND WIDTH ■ The error passes through a dead band. The *DEADB_W* input decides the size of the dead band. ■ Default: 0.0 ■ Range of Values: Dependent on the sensors used
I_ITLVAL	INPUT	REAL	INITIALIZATION VALUE OF THE INTEGRAL ACTION ■ The output of the integral action can be set at the *I_ITL_ON* input. The initialization value is applied to the *I_ITLVAL* input. ■ During a restart *COM_RST* = TRUE, the I action is set to the initialization value. ■ Default: 0.0 ■ Range of Values: 0 to 100 %
LMN_HLM	INPUT	REAL	MANIPULATED VARIABLE HIGH LIMIT ■ The value of the manipulated variable is always limited to an upper and lower limit. The *LMN_HLM* input specifies the upper limit. ■ Default: 100.0 ■ Range of Values: > *LMN_ LLM*
LMN_LLM	INPUT	REAL	MANIPULATED VARIABLE LOW LIMIT ■ The value of the manipulated variable is always limited to an upper and lower limit. The *LMN_LLM* input specifies the lower limit. ■ Default: 0.0 ■ Range of Values: < *LMN_HLM*
PV_FAC	INPUT	REAL	PROCESS VARIABLE FACTOR ■ The *PV_FAC* input is multiplied by the *PV_PER*. The input is used to adapt the process variable range. ■ Default: 1.0
PV_OFFS	INPUT	REAL	PROCESS VARIABLE OFFSET ■ The *PV_OFFS* input is added to the *PV_PER*. The input is used to adapt the process variable range. ■ Default: 0.0
LMN_FAC	INPUT	REAL	MANIPULATED VARIABLE FACTOR ■ The *LMN_FAC* input is multiplied by the manipu-lated variable. The input is used to adapt the manipulated variable range. ■ Default: 1.0
LMN_OFFS	INPUT	REAL	MANIPULATED VARIABLE OFFSET ■ The *LMN_OFFS* input is added to the value of the manipulated variable. The input is used to adapt the manipulated variable range. ■ Default: 0.0

Parameter	Declaration	Data type	Description
PER_TM	INPUT	REAL	PERIOD TIME [s] ■ The pulse repetition period of the pulse duration modulation is entered at the *PER_TM* parameter. The relationship of the pulse repetition period to the sampling time of the pulse generator decides the accuracy of the pulse duration modulation. ■ Default: 1.0 s ■ Range of Values: ≥ *CYCLE*
P_B_TM	INPUT	REAL	MINIMUM PULSE/BREAK TIME [s] ■ A minimum pulse or minimum break time can be set at the *P_B_TM* parameter. *P_B_TM* is limited internally to > *CYCLE_P*. ■ Default: 0.02 s ■ Range of Values: ≥ 0.0
TUN_DLMN	INPUT	REAL	DELTA MANIPULATED VARIABLE FOR PROCESS EXCITATION ■ Process excitation for controller tuning results from a setpoint step change at *TUN_DLMN*. ■ Default: 20.0 ■ Range of Values: -100.0 ... 100.0 %
PER_MODE	INPUT	INT	PERIPHERY MODE ■ You can enter the type of the I/O module at this switch. The process variable at input *PV_PER* is then normalized to °C at the *PV* output. – *PER_MODE* = 0: standard – *PER_MODE* = 1: climate – *PER_MODE* = 2: current/voltage ■ Default: 0 ■ Range of Values: 0, 1, 2
PVPER_ON	INPUT	BOOL	PROCESS VARIABLE PERIPHERY ON ■ If you want the process variable to be read in from the I/O, the *PV_PER* input must be connected to the I/O and the *PVPER_ON* input must be set. ■ Default: FALSE
I_ITL_ON	INPUT	BOOL	INITIALIZATION OF THE INTEGRAL ACTION ON ■ The output of the integral action can be set to the *I_ITLVAL* input. The *I_ITL_ON* input must be set. ■ Default: FALSE
PULSE_ON	INPUT	BOOL	PULSE GENERATOR ON ■ If *PULSE_ON* = TRUE is set, the pulse generator is activated ■ Default: FALSE
TUN_KEEP	INPUT	BOOL	KEEP TUNING ON ■ The mode changes to automatic only when *TUN_KEEP* changes to FALSE. ■ Default: FALSE

Parameter	Declaration	Data type	Description
ER	OUTPUT	REAL	ERROR SIGNAL ■ The effective error is output at the *ER* output. ■ Default: 0.0 ■ Range of Values: Dependent on the sensors used
LMN_P	OUTPUT	REAL	PROPORTIONALITY COMPONENT ■ The *LMN_P* contains the proportional action of the manipulated variable. ■ Default: 0.0
LMN_I	OUTPUT	REAL	INTEGRAL COMPONENT ■ The *LMN_I* contains the integral action of the manipulated variable. ■ Default: 0.0
LMN_D	OUTPUT	REAL	DERIVATIVE COMPONENT ■ The *LMN_D* contains the derivative action of the manipulated variable. ■ Default: 0.0
PHASE	OUTPUT	INT	PHASE OF SELF TUNING ■ The current phase of the controller tuning is indicated at the *PHASE* output (0...7). ■ Default: 0 ■ Range of Values: 0, 1, 2, 3, 4, 5, 7
STATUS_H	OUTPUT	INT	STATUS HEATING OF SELF TUNING ■ *STATUS_H* indicates the diagnostic value of the search for the point of inflection when heating. ■ Default: 0
STATUS_D	OUTPUT	INT	STATUS CONTROLLER DESIGN OF SELF TUNING ■ *STATUS_D* indicated the diagnostic value of the controller design when heating. ■ Default: 0
QTUN_RUN	OUTPUT	BOOL	TUNING IS ACTIVE (PHASE 2) ■ The tuning manipulated variable has been applied, tuning has started and is still in phase 2 (locating the point of inflection). ■ Default: 0
PI_CON	OUTPUT	STRUCT	PI CONTROLLER PARAMETERS
GAIN	OUTPUT	REAL	PI PROPORTIONAL GAIN ■ Default: 0.0 ■ Range of Values: % / phys. unit
TI	OUTPUT	REAL	PI RESET TIME [s] ■ Default: 0.0 s ■ Range of Values: ≥ 0.0 s
PID_CON	OUTPUT	STRUCT	PID CONTROLLER PARAMETERS/ PID Reglerparameter

Parameter	Declaration	Data type	Description
GAIN	OUTPUT	REAL	PID PROPORTIONAL GAIN ■ Default: 0.0
TI	OUTPUT	REAL	PID RESET TIME [s ■ Default: 0.0 s ■ Range of Values: ≥ 0.0 s
TD	OUTPUT	REAL	PID DERIVATIVE TIME [s] ■ Default: 0.0 s ■ Range of Values: ≥ 0.0 s
PAR_SAVE	OUTPUT	STRUCT	SAVED CONTROLLER PARAMETERS ■ The PID parameters are saved in this structure.
PFAC_SP	INPUT/ OUTPUT	REAL	PROPORTIONAL FACTOR FOR SETPOINT CHANGES ■ Default: 1.0 ■ Range of Values: 0.0 ... 1.0
GAIN	OUTPUT	REAL	PROPORTIONAL GAIN ■ Default: 0.0 ■ Range of Values: % / phys. unit
TI	INPUT/ OUTPUT	REAL	RESET TIME [s] ■ Default: 40.0 s ■ Range of Values: ≥ 0.0 s
TD	INPUT/ OUTPUT	REAL	DERIVATIVE TIME [s] ■ Default: 10.0 s ■ Range of Values: ≥ 0.0 s
D_F	OUTPUT	REAL	DERIVATIVE FACTOR ■ Default: 5.0 ■ Range of Values: 5.0 ... 10.0
CON_ZONE	OUTPUT	REAL	CONTROL ZONE ON ■ Default: 100.0 ■ Range of Values: ≥ 0.0
CONZ_ON	OUTPUT	REAL	CONTROL ZONE ■ Default: FALSE
PFAC_SP	INPUT/ OUTPUT	REAL	PROPORTIONAL FACTOR FOR SETPOINT CHANGES ■ *PFAC_SP* specifies the effective P action when there is a setpoint change. This is set between 0 and 1. – 1: P action has full effect if the setpoint changes. – 0: P action has no effect if the setpoint changes. ■ Default: 1.0 ■ Range of Values: 0.0 ... 1.0

Parameter	Declaration	Data type	Description
GAIN	INPUT/ OUTPUT	REAL	PROPORTIONAL GAIN ■ The *GAIN* input specifies the controller gain. The direction of control can be reversed by giving GAIN a negative sign. ■ Default: 0.0 ■ Range of Values: % / phys. Value
TI	INPUT/ OUTPUT	REAL	RESET TIME [s] ■ The *TI* input (integral time) decides the integral action response. ■ Default: 40.0 s ■ Range of Values: ≥ 0.0 s
TD	INPUT/ OUTPUT	REAL	DERIVATIVE TIME [s] ■ The *TD* input decides the derivative action response. ■ Default: 10.0 s ■ Range of Values: ≥ 0.0 s
D_F	INPUT/ OUTPUT	REAL	DERIVATIVE FACTOR ■ The derivative factor *D_F* decides the lag of the D-action. – *D_F* = derivative time / "lag of the D-action" ■ Default: 5.0 ■ Range of Values: 5.0 ... 10.0
CON_ZONE	INPUT/ OUTPUT	REAL	CONTROL ZONE ON ■ If the error is greater than the control zone width *CON_ZONE*, the upper manipulated variable limit is output as the manipulated variable. ■ If the error is less than the negative control zone width, the lower manipulated variable limit is output as the manipulated variable. ■ Default: 100.0 ■ Dependent on the sensors used
CONZ_ON	INPUT/ OUTPUT	BOOL	CONTROL ZONE ■ *CONZ_ON* =TRUE activates the control zone. ■ Default: FALSE
TUN_ON	INPUT/ OUTPUT	BOOL	SELF TUNING ON ■ If *TUN_ON* = TRUE is set, the manipulated value is averaged until the manipulated variable excitation *TUN_DLMN* is activated either by a setpoint step change or by *TUN_ST* = TRUE. ■ Default: FALSE
TUN_ST	INPUT/ OUTPUT	BOOL	START SELF TUNING ■ If the setpoint is to remain constant during controller tuning at the operating point, a manipulated variable step change by the amount of *TUN_DLMN* is activated by *TUN_ST* = TRUE. ■ Default: FALSE

Parameter	Declaration	Data type	Description
UNDO_PAR	INPUT/ OUTPUT	BOOL	UNDO CHANGE OF CONTROLLER PARAMETERS ■ Loads the controller parameters *PFAC_SP*, *GAIN*, *TI*, *TD*, *D_F*, *CONZ_ON* and *CON_ZONE* from the data structure *PAR_SAVE* (only in manual mode). ■ Default: FALSE
SAVE_PAR	INPUT/ OUTPUT	BOOL	SAVE CURRENT CONTROLLER PARAMETERS ■ Saves the controller parameters *PFAC_SP*, *GAIN*, *TI*, *TD*, *D_F*, *CONZ_ON* and *CON_ZONE* in the data structure *PAR_SAVE*. ■ Default: FALSE
LOAD_PID	INPUT/ OUTPUT	BOOL	LOAD OPTIMIZED PI/PID PARAMETERS ■ Loads the controller parameters *GAIN*, *TI*, *TD* depending on *PID_ON* from the data structure *PI_CON* or *PID_CON* (only in manual mode) ■ Default: FALSE
PID_ON	INPUT/ OUTPUT	BOOL	PID MODE ON ■ At the *PID_ON* input, you can specify whether or not the tuned controller will operate as a PI or PID controller. – PID controller: *PID_ON* = TRUE – PI controller: *PID_ON* = FALSE ■ It is nevertheless possible that with certain process types, only a PI controller will be designed despite *PID_ON* = TRUE. ■ Default: TRUE
GAIN_P	OUTPUT	REAL	PROZESS PROPORTIONAL GAIN ■ Identified process gain. For the process type I, *GAIN_P* tends to be estimated too low. ■ Default: 0.0
TU	OUTPUT	REAL	DELAY TIME [s] ■ Identified delay of the process. ■ Default: 0.0 ■ Range of Values: ≥ 3**CYCLE*
TA	OUTPUT	REAL	RECOVERY TIME [s] ■ Identified system time constant of the process. For the process type I, *TA* tends to be estimated too low. ■ Default: 0.0
KIG	OUTPUT	REAL	MAXIMAL ASCENT RATIO OF PV WITH 100 % LMN CHANGE ■ *GAIN_P* = 0.01 * *KIG* * *TA* ■ Default: 0.0
N_PTN	OUTPUT	REAL	PROCESS ORDER ■ The parameter specifies the order of the process. "Non-integer values" are also possible. ■ Default: 0.0 ■ Range of Values: 1.01 to 10.0

Parameter	Declaration	Data type	Description
TM_LAG_P	OUTPUT	REAL	TIME LAG OF PTN MODEL [s] ■ Time lag of PTN model (values only for *N_PTN* ≥ 2). ■ Default: 0.0
T_P_INF	OUTPUT	REAL	TIME TO POINT OF INFLECTION [s] ■ Time from process excitation until the point of inflection. ■ Default: 0.0
P_INF	OUTPUT	REAL	PV AT POINT OF INFLECTION - PV0 ■ Process variable change from process excitation until the point of inflection. ■ Default: 0.0 ■ Range of Values: Value range of the process value
LMN0	OUTPUT	REAL	MANIPULATED VAR. AT BEGIN OF TUNING ■ Detected in phase 1 (mean value). ■ Default: 0.0 ■ Range of Values: 0 ... 100 %
PV0	OUTPUT	REAL	PROCESS VALUE AT BEGIN OF TUNING ■ Default: 0.0 ■ Range of Values: Value range of the process value
PVDT0	OUTPUT	REAL	RATE OF CHANGE OF PV AT BEGIN OF TUNING [1/s] ■ Sign adapted ■ Default: 0.0
PVDT	OUTPUT	REAL	CURRENT RATE OF CHANGE OF PV [1/s] ■ Sign adapted ■ Default: 0.0
PVDT_MAX	OUTPUT	REAL	MAX. RATE OF CHANGE OF PV PER SECOND [1/s] ■ Maximum rate of change of the process variable at the point of inflection at the (sign adapted, always > 0), used to calculate *TU* and *KIG*. ■ Default: 0.0
NOI_PVDT	OUTPUT	REAL	RATIO OF NOISE IN PVDT_MAX IN % ■ The higher the proportion of noise, less accurate (less aggressive) the control parameters. ■ Default: 0.0
NOISE_PV	OUTPUT	REAL	ABSOLUTE NOISE IN PV ■ Difference between maximum and minimum process variable in phase 1. ■ Default: 0.0

Parameter	Declaration	Data type	Description
FIL_CYC	OUTPUT	INT	NO OF CYCLES FOR MEAN-VALUE FILTER ■ The process variable is averaged over *FIL_CYC* cycles. When necessary, *FIL_CYC* is increased automatically from 1 to a maximum of 1024. ■ Default: 1 ■ Range of Values: 1 ... 1024
POI_CMAX	OUTPUT	INT	MAX NO OF CYCLES AFTER POINT OF INFLECTION ■ This time is used to find a further (in other words better) point of inflection when measurement noise is present. The tuning is completed only after this time. ■ Default: 2
POI_CYCL	OUTPUT	INT	NUMBER OF CYCLES AFTER POINT OF INFLECTION ■ Default: 0

Application

- The functionality is based on the PID control algorithm with additional functions for temperature processes. The controller supplies analog manipulated values and pulse-duration modulated actuating signals. The controller outputs signals to one actuator; in other words, with one controller, you can either heat or cool but not both.
- FB 58 TCONT_CP can be used either purely for heating or purely for cooling. If you use the block for cooling, *GAIN* must be assigned a negative value. This inversion of the controller means that, for example if the temperature rises, the manipulated variable *LMN* and with it the cooling effort is increased.
- Apart from the functions in the setpoint and process value branches, the FB implements a complete PID temperature controller with a continuous and binary manipulated variable output. To improve the control response with temperature processes, the block includes a control zone and reduction of the P-action if there is a setpoint step change. The block can set the PI/PID parameters itself using the controller tuning function.

The values in the controller blocks are only calculated correctly if the block is called at regular intervals. Therefore, you have to call the controller blocks in a cyclic interrupt OB (OB 30 ... 38) at regular intervals. The sampling time is predefined on the parameter CYCLE.

Setpoint Branch

The setpoint is entered at input *SP_INT* in floating-point format as a physical value or percentage. The setpoint and process value used to form the error must have the same unit.

Process Value Options (*PVPER_ON*)

Depending on *PVPER_ON*, the process value can be acquired in the peripheral (I/O) or floating-point format.

PVPER_ON	Process Value Input
TRUE	The process value is read in via the analog peripheral I/Os (PIW xxx) at input *PV_PER*.
FALSE	The process value is acquired in floating-point format at input PV_IN.

Process Value Format Conversion *CRP_IN* (*PER_MODE*)

The *CRP_IN* function converts the peripheral value *PV_PER* to a floating-point format depending on the switch *PER_MODE* according to the following rules:

PER_MODE	**Output of *CRP_IN***	**Analog Input Type**	**Unit**
0	*PV_PER* * 0.1	Thermoelements; PT100/ NI100; standard	°C; °F
1	*PV_PER* * 0.01	PT100/NI100; climate	°C; °F
2	*PV_PER* * 100/27648	Voltage/current	%

Process Value Normalization *PV_NORM* (*PV_FAC*, *PV_OFFS*)

The *PV_NORM* function calculates the output of *CRP_IN* according to the following rule:
*Output of PV_NORM = Ausgang von CPR_IN * PV_FAC + PV_OFFS*

It can be used for the following purposes:

- Process value correction with *PV_FAC* as the process value factor and *PV_OFFS* as the process value offset.
- Normalization of temperature to percentage
  You want to enter the setpoint as a percentage and must now convert the measured temperature value to a percentage.
- Normalization of percentage to temperature
  You want to enter the setpoint in the physical temperature unit and must now convert the measured voltage/current value to a temperature.

Calculation of the parameters:

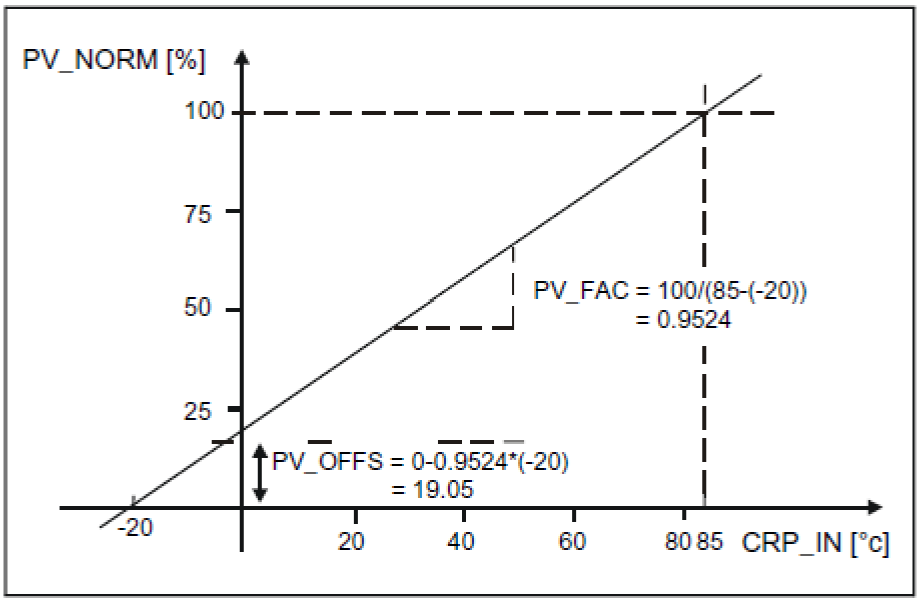
- *PV_FAC = range of PV_NORM/range of CRP_IN*
- *PV_OFFS = LL(PV_NORM) - PV_FAC * LL(CRP_IN)*; where *LL* is the lower limit

With the default values (*PV_FAC* = 1.0 and *PV_OFFS* = 0.0), normalization is disabled. The effective process value is output at the *PV* output.

With pulse control, the process value must be transferred to the block in the fast pulse call (reason: mean value filtering). Otherwise, the control quality can deteriorate.

Example of Process Variable Normalization

If you want to enter the setpoint as a percentage, and you have a temperature range of -20 ... 85 °C applied to *CRP_IN*, you must normalize the temperature range as a percentage. The schematic below shows an example of adapting the temperature range -20 ... 85 °C to an internal scale of 0 ... 100 %:

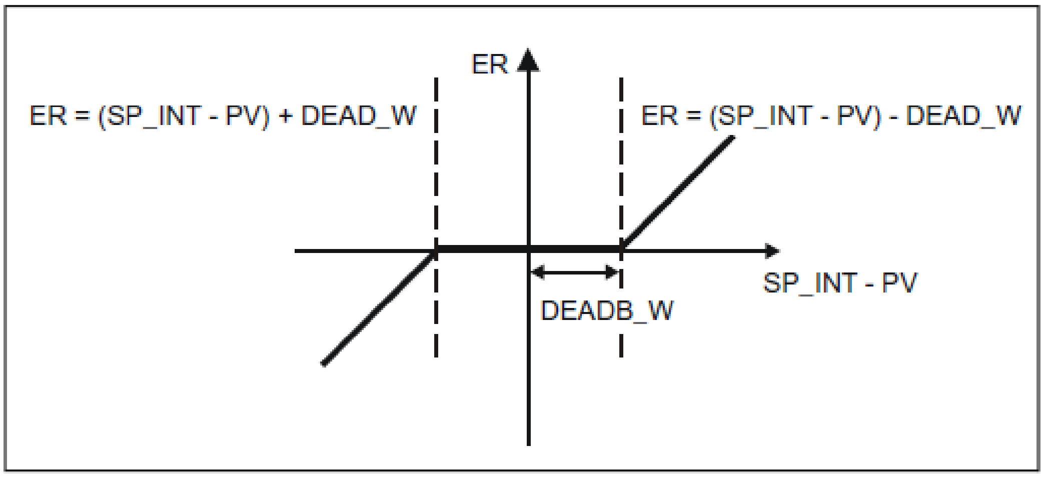


Forming the Error

The difference between the setpoint and process value is the error before the deadband. The setpoint and process value must exist in the same unit.

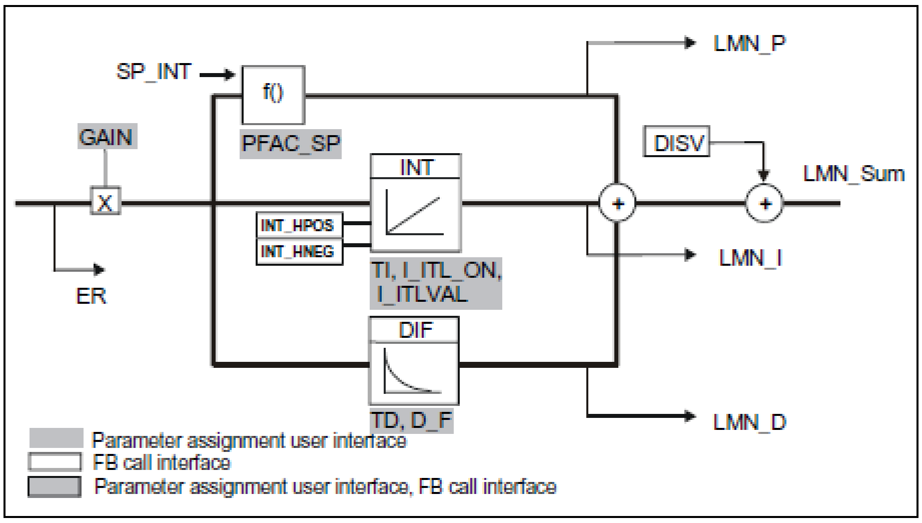
Deadband (*DEADB_W*)

To suppress a small constant oscillation due to the manipulated variable quantization (for example in pulse duration modulation with PULSEGEN) a deadband (DEADBAND) is applied to the error. If *DEADB_W* = 0.0, the deadband is deactivated. The effective error is indicated by the *ER* parameter.



PID Algorithm

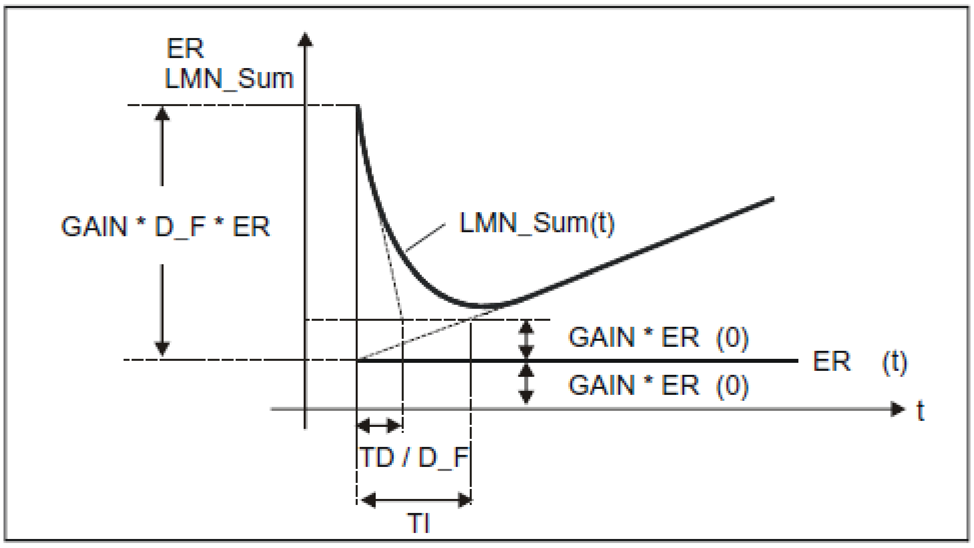
The schematic below is the block diagram of the PID algorithm:



PID Algorithm (*GAIN, TI, TD, D_F*)

- The PID algorithm operates as a position algorithm. The proportional, integral (*INT*), and derivative (*DIF*) actions are connected in parallel and can be activated or deactivated individually. This allows P, PI, PD, and PID controllers to be configured.
- The controller tuning supports PI and PID controllers. Controller inversion is implemented using a negative *GAIN* (cooling controller).
- If you set *TI* and *TD* to 0.0, you obtain a pure P controller at the operating point.

$$LMN\_Sum(t) = GAIN * ER(0)\left(1+\frac{1}{TI}*t+D\_F*e^{\frac{-t}{TD/D\_F}}\right)$$



LMN_Sum(t)	manipulated variable in automatic mode of the controller
ER (0)	step change of the normalized error
GAIN	controller gain
TI	integral time
TD	derivative time
D_F	derivative factor

Integrator (*TI, I_ITL_ON, I_ITLVAL*)

In the manual mode, it is corrected as follows: *LMN_I = LMN - LMN_P - DISV*

If the manipulated variable is limited, the I-action is stopped. If the error moves the I-action back in the direction of the manipulated variable range, the I-action is enabled again.

The I-action is also modified by the following measures:

- The I-action of the controller is deactivated by *TI* = 0.0
- Weakening the P-action when setpoint changes occur
- Control zone
- The limits of the manipulated variable can be changed online

Weakening the P-Action when Setpoint Changes Occur (*PFAC_SP*)

To prevent overshoot, you can weaken the P-action using the "proportional factor for setpoint changes" parameter (*PFAC_SP*). Using *PFAC_SP*, you can select continuously between 0.0 and 1.0 to decide the effect of the P-action when the setpoint changes:

- *PFAC_SP* = 1.0: P-action has full effect if the setpoint changes
- *PFAC_SP* = 0.0: P-action has no effect if the setpoint changes

The weakening of the P-action is achieved by compensating the I-action.

Derivative Action Element (*TD, D_F*)

- The D-action of the controller is deactivated with *TD* = 0.0.
- If the D-action is active, the following relationship should apply: *TD* = 0.5 * *CYCLE* * *D_F*

Parameter Settings of a P or PD Controller with Operating Point

In the user interface, deactivate the I-action (TI = 0.0) and possible also the D-action (TD = 0.0). Then make the following parameter settings:

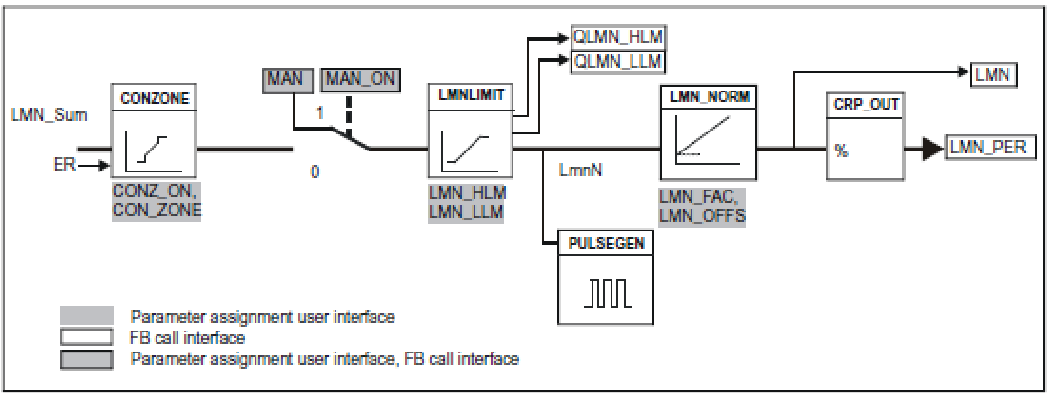
- *I_ITL_ON* = TRUE
- *I_ITLVAL* = operating point;

Feedforward Control (*DISV*)

A feedforward variable can be added at the *DISV* input.

Calculating the Manipulated Variable

The schematic below is the block diagram of the manipulated variable calculation:

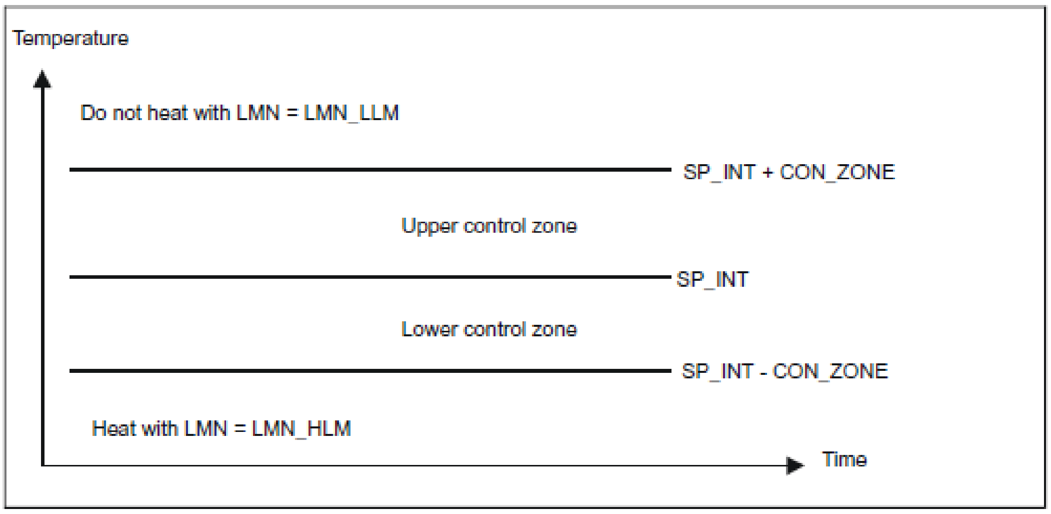


Control Zone (*CONZ_ON, CON_ZONE*)

If *CONZ_ON* = TRUE, the controller operates with a control zone. This means that the controller operates according to the following algorithm:

- If *PV* exceeds *SP_INT* by more than *CON_ZONE*, the value *LMN_LLM* is output as the manipulated variable (controlled closed-loop).
- If *PV* falls below *SP_INT* by more than *CON_ZONE*, the value *LMN_HLM* is output as the manipulated variable (controlled closed-loop).
- If *PV* is within the control zone (*CON_ZONE*), the manipulated variable takes its value from the PID algorithm *LMN_Sum* (automatic closed-loop control).

The changeover from controlled closed-loop to automatic closed-loop control takes into account a hysteresis of 20% of the control zone.



Before activating the control zone manually, make sure that the control zone band is not too narrow. If the control zone band is too small, oscillations will occur in the manipulated variable and process variable.

Advantage of the Control Zone

When the process value enters the control zone, the D-action causes an extremely fast reduction of the manipulated variable. This means that the control zone is only useful when the D-action is activated. Without a control zone, basically only the reducing P-action would reduce the manipulated variable. The control zone leads to faster settling without overshoot or undershoot if the output minimum or maximum manipulated variable is a long way from the manipulated variable required for the new operating point.

Manual Value Processing (*MAN_ON, MAN*)

You can switch over between manual and automatic operation. In the manual mode, the manipulated variable is corrected to a manual value. The integral action (*INT*) is set internally to *LMN - LMN_P - DISV* and the derivative action (*DIF*) is set to 0 and synchronized internally. Switching over to automatic mode is therefore bumpless.

During tuning, the MAN_ON parameter has no effect.

Manipulated Variable Limitation *LMNLIMIT* (*LMN_HLM, LMN_LLM*)

The value of the manipulated variable is limited to the *LMN_HLM* and *LMN_LLM* limits by the *LMNLIMIT* function. If these limits are reached, this is indicated by the message bits *QLMN_HLM* and *QLMN_LLM*. If the manipulated variable is limited, the I-action is stopped. If the error moves the I-action back in the direction of the manipulated variable range, the I-action is enabled again.

Changing the Manipulated Variable Limits Online

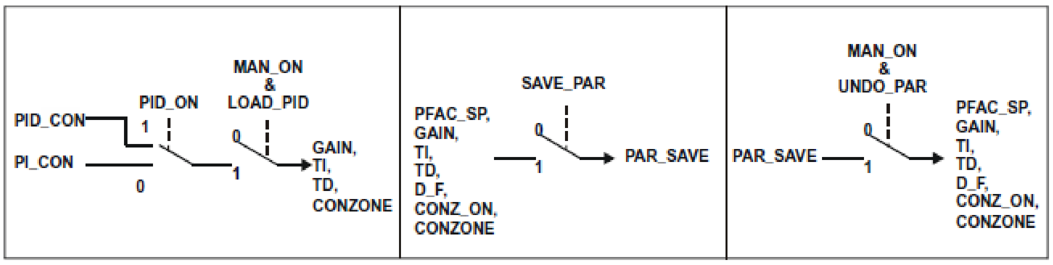
If the range of the manipulated variable is reduced and the new unlimited value of the manipulated variable is outside the limits, the I-action and therefore the value of the manipulated variable shifts. The manipulated variable is reduced by the same amount as the manipulated variable limit changed. If the manipulated variable was unlimited prior to the change, it is set exactly to the new limit (described here for the upper manipulated variable limit).

Manipulated Variable Normalization *LMN_NORM* (*LMN_FAC, LMN_OFFS*)

- The *LMN_NORM* function normalizes the manipulated variable according to the following formula:
  *LMN = LmnN * LMN_FAC + LMN_OFFS*
- It can be used for the following purposes:
  Manipulated variable adaptation with *LMN_FAC* as manipulated variable factor and *LMN_OFFS* manipulated variable offset
- The value of the manipulated variable is also available in the peripheral format. The *CRP_OUT* function converts the LMN floating-point value to a peripheral value according to the following formula:
  *LMN_PER = LMN * 27648/100*
  With the default values (*LMN_FAC* = 1.0 and *LMN_OFFS* = 0.0), normalization is disabled. The effective manipulated variable is output at output *LMN*.

Saving and Reloading Controller Parameters

The schematic below shows the block diagram:



Saving Controller Parameters *SAVE_PAR*

If the current parameter settings are usable, you can save them in a special structure in the instance DB of FB 58 TCONT_CP prior to making a manual change. If you tune the controller, the saved parameters are overwritten by the values that were valid prior to tuning. *PFAC_SP, GAIN, TI, TD, D_F, CONZ_ON* and *CON_ZONE* are written to the *PAR_SAVE* structure.

Reloading Saved Controller Parameters *UNDO_PAR*

The last controller parameter settings you saved can be activated for the controller again using this function (in manual mode only).

Changing Between PI and PID Parameters *LOAD_PID* (*PID_ON*)

Following tuning, the PI and PID parameters are stored in the *PI_CON* and *PID_CON* structures. Depending on *PID_ON*, you can use *LOAD_PID* in the manual mode to write the PI or PID parameters to the effective controller parameters.

PID parameter *PID_ON* = TRUE		**PI parameter PID_ON = FALSE**	
GAIN	= PID_CON.GAIN	GAIN	= PI_CON.GAIN
TI	= PID_CON.TI	TI	= PI_CON.TI
TD	= PID_CON.TD		

- *The controller parameters are only written back to the controller with UNDO_PAR or LOAD_PID when the controller gain is not 0:*
  For LOAD_PID, the parameters are only copied if the respective GAIN <> 0 (either from the PI or PID parameter set). This takes into account the case that no optimization has yet been performed or PID parameters are missing. If PID_ON = TRUE and PID.GAIN = FALSE, PID_ON is set to FALSE and the PI parameters are copied.
- *D_F, PFAC_SP are set to default values by the tuning. These can then be modified by the user. LOAD_PID does not change these parameters.*
- *With LOAD_PID, the control zone is always recalculated (CON_ZONE = 250/GAIN) even when CONZ_ON = FALSE is set.*

4.5.5 FB 59 - TCONT_S - Temperature Step Control

Description

FB 59 TCONT_S is used to control technical temperature processes with binary controller output signals for integrating actuators. By setting parameters, subfunctions of the PI step controller can be activated or deactivated and the controller adapted to the process.

Parameters

Parameter	Declaration	Data type	Description
CYCLE	INPUT	REAL	SAMPLE TIME OF STEP CONTROLLER [s] ▪ At this input *CYCLE*, you enter the sampling time for the controller. ▪ Default: 0.0 ▪ Range of Values: ≥ 0.001
SP_INT	INPUT	REAL	INTERNAL SETPOINT ▪ The *SP_INT* input is used to specify a setpoint. ▪ Default: 0.0 ▪ Range of Values: Dependent on the sensors used
PV_IN	INPUT	REAL	PROCESS VARIABLE IN ▪ An initialization value can be set at the *PV_PER* input or an external process variable in floating-point format can be connected. ▪ Default: 0.0 ▪ Range of Values: Dependent on the sensors used
PV_PER	INPUT	WORD	PROCESS VARIABLE PERIPHERY ▪ The process variable in the peripheral I/O format is connected to the controller at the *PV_PER* input. ▪ Default: 0
DISV	INPUT	REAL	DISTURBANCE VARIABLE ▪ For feed forward control, the disturbance variable is connected to the *DISV* input. ▪ Default: 0.0

Parameter	Declaration	Data type	Description
LMNR_HS	INPUT	BOOL	HIGH LIMIT SIGNAL OF REPEATED MANIPULATED VALUE ■ The signal "valve at upper limit stop" is connected to the *LMNR_HS*. ■ *LMNR_HS* = TRUE: The valve is at the upper limit stop. ■ Default: FALSE
LMNR_LS	INPUT	BOOL	LOW LIMIT SIGNAL OF REPEATED MANIPULATED VALUE ■ The signal "valve at upper lower stop" is connected to the input *LMNR_LS*. ■ *LMNR_LS* = TRUE: The valve is at the lower limit stop. ■ Default: FALSE
LMNS_ON	INPUT	BOOL	MANIPULATED SIGNALS ON ■ The processing of the controller output signal is set to manual at the *LMNS_ON* input. ■ Default: TRUE
LMNUP	INPUT	BOOL	MANIPULATED SIGNALS UP ■ With the controller output signals set to manual, the *QLMNUP* output signal is applied to the *LMNUP* input. ■ Default: FALSE
LMNDN	INPUT	BOOL	MANIPULATED SIGNALS DOWN ■ With the controller output signals set to manual, the *QLMNDN* output signal is applied to the *LMNDN* input. ■ Default: FALSE
QLMNUP	OUTPUT	BOOL	MANIPULATED SIGNAL UP ■ If the *QLMNUP* output is set, the valve will be opened. ■ Default: FALSE
QLMNDN	OUTPUT	BOOL	MANIPULATED SIGNAL DOWN ■ If the *QLMNDN* output is set, the valve will be closed. ■ Default: FALSE
PV	OUTPUT	REAL	PROCESS VARIABLE ■ The effective process variable is output at the *PV* output. ■ Default: 0.0
PE	OUTPUT	REAL	ERROR SIGNAL ■ The effective error is output at the *PE* output. ■ Default: 0.0
COM_RST	INPUT/ OUTPUT	BOOL	COMPLETE RESTART ■ The block has an initialization routine that is processed when the *COM_RST* input is set. ■ Default: FALSE

Internal Parameters

Parameter	Declaration	Data Type	Description
PV_FAC	INPUT	REAL	PROCESS VARIABLE FACTOR ■ The *PV_FAC* input is multiplied by the "process value". The input is used to adapt the process variable range. ■ Default: 1.0
PV_OFFS	INPUT	REAL	PROCESS VARIABLE OFFSET ■ The *PV_OFFS* input is added to the process variable. The input is used to adapt the process variable range. ■ Default: 0.0 ■ Range of Values: Dependent on the sensors used
DEADB_W	INPUT	REAL	DEAD BAND WIDTH ■ The error passes through a dead band. The *DEADB_W* input decides the size of the dead band. ■ Default: 0.0 ■ Range of Values: Dependent on the sensors used
PFAC_SP	INPUT	REAL	PROPORTIONAL FACTOR FOR SETPOINT CHANGES [0..1] ■ *PFAC_SP* specifies the effective P action when there is a setpoint change. This is set between 0 and 1. – 1: P action has full effect if the setpoint changes. – 0: P action has no effect if the setpoint changes. ■ Default: 1.0 ■ Range of Values: 0.0 ... 1.0
GAIN	INPUT	REAL	PROPORTIONAL GAIN ■ The *GAIN* input specifies the controller gain. The direction of control can be reversed by giving GAIN a negative sign. ■ Default: 2.0 ■ Range of Values: %/phys. unit
TI	INPUT	REAL	RESET TIME [s] ■ The *TI* input (integral time) decides the integral action response. ■ Default: 40.0 s ■ Range of Values: ≥ 0.0 s
MTR_TM	INPUT	REAL	MOTOR ACTUATING TIME ■ The operating time of the valve from limit stop to limit stop is entered in the *MTR_TM* parameter. ■ Default: 30 s ■ Range of Values: ≥ *CYCLE*

Parameter	Declaration	Data Type	Description
PULSE_TM	INPUT	REAL	MINIMUM PULSE TIME ■ A minimum pulse time can be set with the *PULSE_TM* parameter. ■ Default: 0.1s ■ Range of Values: ≥ 0.0 s
BREAK_TM	INPUT	REAL	MINIMUM BREAK TIME ■ A minimum break time can be set with the *BREAK_TM* parameter. ■ 0.1s ■ Range of Values: ≥ 0.0 s
PER_MODE	INPUT	INT	PERIPHERIE MODE ■ You can enter the type of the I/O module at this switch. The process variable at input *PV_PER* is then normalized to °C at the *PV* output. – *PER_MODE* = 0: standard – *PER_MODE* = 1: climate – *PER_MODE* = 2: current/voltage ■ Default: 0 ■ Range of Values: 0, 1, 2
PVPER_ON	INPUT	BOOL	PROCESS VARIABLE PERIPHERY ON ■ If you want the process variable to be read in from the I/O, the *PV_PER* input must be connected to the I/O and the *PVPER_ON* input must be set. ■ Default: FALSE

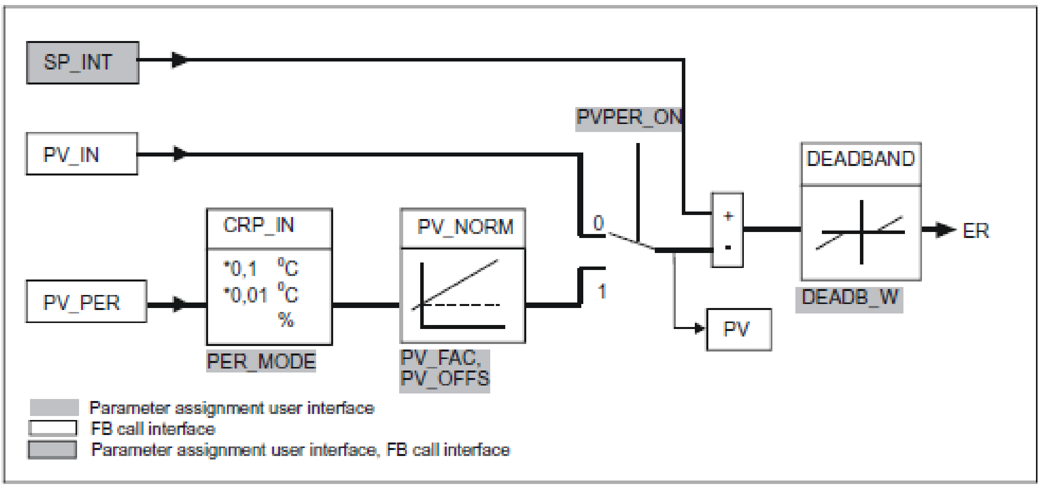
Application

- The functionality is based on the PI control algorithm of the sampling controller. This is supplemented by the functions for generating the binary output signal from the analog actuating signal.
- You can also use the controller in a cascade control as a secondary position controller. You specify the actuator position via the setpoint input *SP_INT*. In this case, you must set the process value input and the parameter *TI* (integral time) to zero. An application might be, for example, temperature control with heating power control using pulse-break activation and cooling control using a butterfly valve. To close the valve completely, the manipulated variable (*ER* * *GAIN*) should be negative.
- Apart from the functions in the process variable branch, FB 59 TCONT_S implements a complete PI controller with binary manipulated value output and the option of influencing the controller output signals manually. The step controller operates without a position feedback signal.

The values in the controller blocks are only calculated correctly if the block is called at regular intervals. Therefore, you have to call the controller blocks in a cyclic interrupt OB (OB 30 ... 38) at regular intervals. The sampling time is predefined on the parameter CYCLE.

Forming the Error

Block Diagram



Setpoint Branch

The setpoint is entered at input *SP_INT* in floating-point format as a physical value or percentage. The setpoint and process value used to form the error must have the same unit.

Process Value Options (*PVPER_ON*)

Depending on *PVPER_ON*, the process value can be acquired in the peripheral (I/O) or floating-point format.

PVPER_ON	Process Value Input
TRUE	The process value is read in via the analog peripheral I/Os (PIW xxx) at input *PV_PER*.
FALSE	The process value is acquired in floating-point format at input *PV_IN*.

Process Value Format Conversion *CRP_IN* (*PER_MODE*)

The *CRP_IN* function converts the peripheral value *PV_PER* to a floating-point format depending on the switch *PER_MODE* according to the following rules:

PER_MODE	Output of *CRP_IN*	Analog Input Type	Unit
0	*PV_PER* * 0.1	Thermoelements; PT100/ NI100; standard	°C; °F
1	*PV_PER* * 0.01	PT100/NI100; climate	°C; °F
2	*PV_PER* * 100/27648	Voltage/current	%

Process Value Normalization *PV_NORM* (*PF_FAC, PV_OFFS*)

The *PV_NORM* function calculates the output of *CRP_IN* according to the following rule:

*Output of PV_NORM = Output of CPR_IN * PV_FAC + PV_OFFS*

This can be used for the following purposes:

- Process value correction with *PV_FAC* as the process value factor and *PV_OFFS* as the process value offset.
- Normalization of temperature to percentage
  You want to enter the setpoint as a percentage and must now convert the measured temperature value to a percentage.
- Normalization of percentage to temperature
  You want to enter the setpoint in the physical temperature unit and must now convert the measured voltage/current value to a temperature.

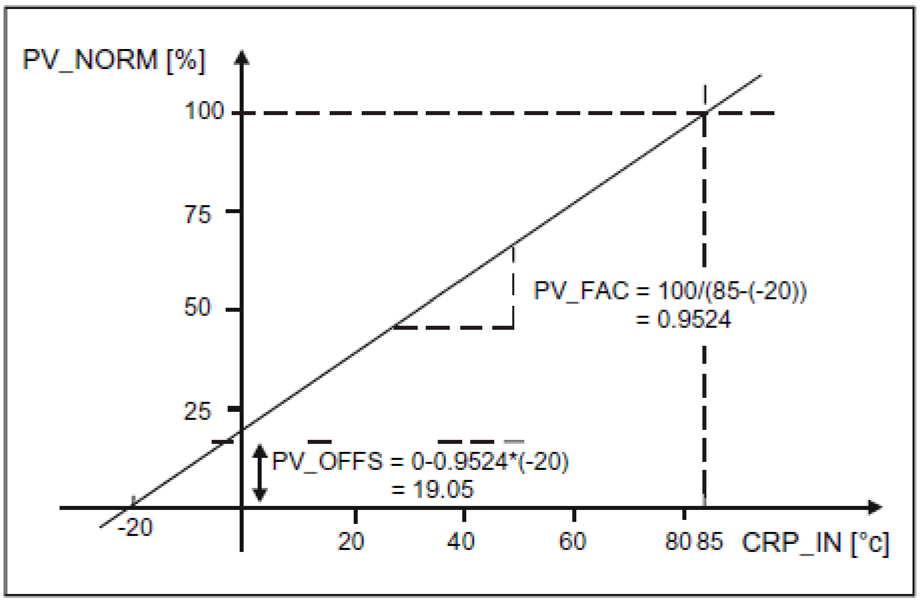
Calculation of the parameters:

- *PV_FAC = range of PV_NORM / range of CRP_IN*
- *PV_OFFS = LL(PV_NORM) - PV_FAC * LL(CRP_IN)*; where *LL* is the lower limit

With the default values (*PV_FAC* = 1.0 and *PV_OFFS* = 0.0), normalization is disabled. The effective process value is output at the *PV* output.

Example of Process Variable Normalization

If you want to enter the setpoint as a percentage, and you have a temperature range of -20 to 85 °C applied to *CRP_IN*, you must normalize the temperature range as a percentage. The schematic below shows the adaptation of the temperature range from -20 ... 85°C to an internal scale of 0 ... 100 %:

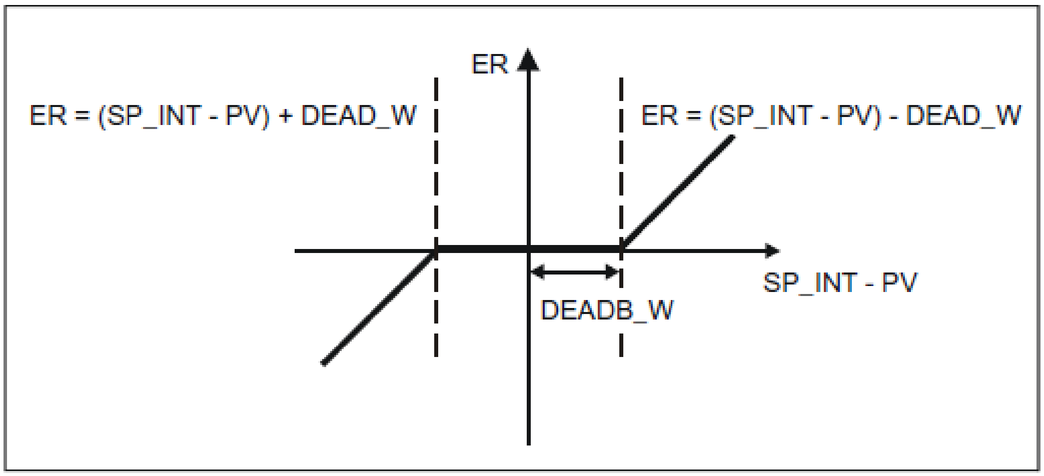


Forming the Error

The difference between the setpoint and process value is the error before the deadband. The setpoint and process value must exist in the same unit.

Deadband (*DEADB_W*)

To suppress a small constant oscillation due to the manipulated variable quantization (for example in pulse duration modulation with PULSEGEN) a deadband (DEADBAND) is applied to the error. If *DEADB_W* = 0.0, the deadband is deactivated.



PI Step Controller Algorithm

FB 59 TCONT_S works without a position feedback signal (see following block diagram). The I-action of the PI algorithm and the assumed position feedback signal are calculated in an integrator (INT) and compared as a feedback value with the remaining P-action. The difference is applied to a three-step element (THREE_ST) and a pulse generator (PULSEOUT) that forms the pulses for the valve. Adapting the response threshold of the three-step element reduces the switching frequency of the controller.

Weakening the P-Action when Setpoint Changes Occur

To prevent overshoot, you can weaken the P-action using the "proportional factor for setpoint changes" parameter (*PFAC_SP*). Using *PFAC_SP*, you can now select continuously between 0.0 and 1.0 to decide the effect of the P-action when the setpoint changes:

- *PFAC_SP* = 1.0: P-action has full effect if the setpoint changes
- *PFAC_SP* = 0.0: P-action has no effect if the setpoint changes

A value for *PFAC_SP* < 1.0 can reduce the overshoot as with the continuous controller if the motor run time *MTR_TM* is small compared with the recovery time *TA* and the ratio *TU/TA* is < 0.2. If *MTR_TM* reaches 20 % of TA, only a slight improvement can be achieved.

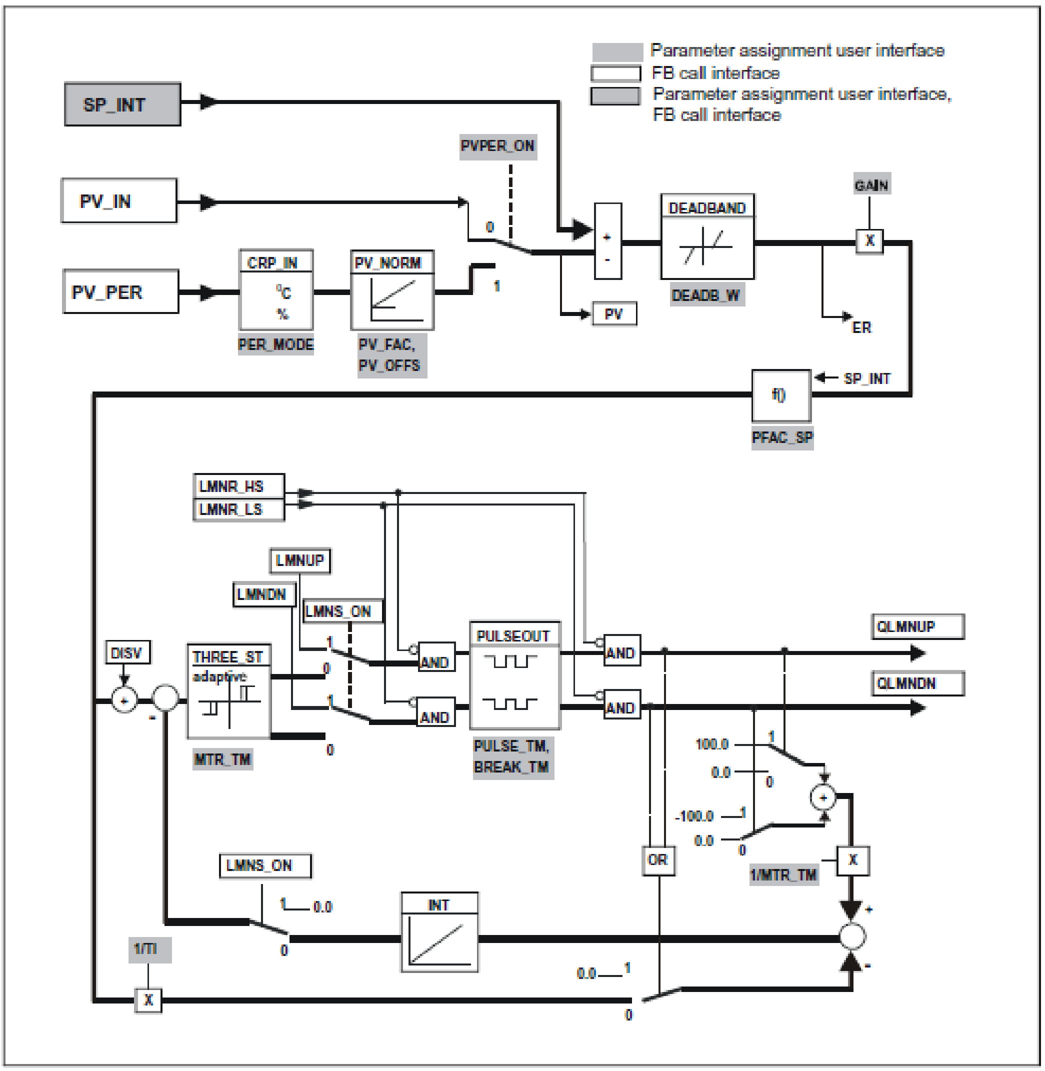
Feedforward Control

A load can be added at the *DISV* input.

Manual Value Processing (*LMNS_ON*)

With *LMNS_ON*, you can change between manual and automatic mode. In manual mode, the actuator and the integrator (INT) are set to 0 internally. Using *LMNUP* and *LMNDN*, the actuator can be adjusted to OPEN and CLOSED. Switching over to automatic mode therefore involves a bump. As a result of the *GAIN*, the existing error leads to a step change in the internal manipulated variable. The integral component of the actuator, however, results in a ramp-shaped excitation of the process.

Block Diagram



4.6 Time Functions

4.6.1 UDT 60 - WS_RULES - Rule DB

Description

Your system must provide certain information in a DB that is evaluated by the various blocks. You create this data block as a DB of the type UDT60 and enter the values that apply to your location (in local time!).

Calculation of base time < - > local time and "set alarm acc. to local time"

Name	Type	Start value	Comment
B2L	STRUCT		Base time < - > Local time
S	INT	2	Offset base time -> local time [30 min] in winter permitted: -24 .. +24.
T	INT	3	Difference summer to winter time [30 min] permitted: 2

Rule for: standard -> daylight-saving time. Default: Last Sunday in March; 2:00 o'clock

Name	Type	Start value	Comment
W2S	STRUCT		W2S must be specified in STANDARD TIME!
M	BYTE	B#16#3	Month of switchover
W	BYTE	B#16#9	nth occurrence of the weekday (1 = first, 2 = second,. , 9 = last)
D	BYTE	B#16#1	Day of week (Sunday = 1)
H	BYTE	B#16#2	Hour

Rule for: daylight-saving -> standard time. Default: Last Sunday in October 3:00 o'clock

Name	Type	Start value	Comment
S2W	STRUCT		S2W must be specified in DAYLIGHT-SAVING TIME
M	BYTE	B#16#10	Month of switchover
W	BYTE	B#16#9	nth occurrence of the weekday (1 = first, 2 = second,. , 9 = last)
D	BYTE	B#16#1	Day of week (Sunday = 1)
H	BYTE	B#16#3	Hour

All the parameters that have the format BYTE are interpreted as BCD values!

The specification of the daylight-saving/standard time switchover points by a rule is mandatory in the EU as of 2002.

4.6.2 FC 61 - BT_LT - Convert base timer to local time

Description

The FC 61 calculates the local time for the base time specified at the input.

Parameter

Parameter	Declaration	Data type	Description
BT	INPUT	DATE_AND_TIME	Base time
WS_DAT	INPUT	BLOCK_DB	Information on the time zone for standard/daylight saving switchover (Rule DB)
RET_VAL	OUTPUT	INT	Error code
LT	OUTPUT	DATE_AND_TIME	Local time

How It Works

The base time entered at input *BT* is converted to the local time using the data stored in a DB and applied to output *LT*. The DB contains the number of 30-minute units by which the base time and local time differ and the difference between daylight-saving time and standard time also in units of 30 minutes. (Rule DB) If the calculation results in a date overflow, this is indicated by a special return value.

Calling OBs

FC 61 BT_LT can be called in any priority class.

Call Environment

Internally, FC 61 uses the following functions. These functions must be loaded in your project with the numbers shown here. FC1 (AD_DT_TM), FC7 (DT_DAY), FC35 (SB_DT_TM)

Output Values / Errors

RET_VAL	LT	Description
0	Local time	Block executed error-free
1	Local time	No error, but date jump
8082	DT#90-01-01-0:0:0	Invalid data in the rule data block

4.6.3 FC 62 - LT_BT - Convert local time to base time

Description

The FC 62 calculates the base time for the local time specified at the input.

Parameters

Parameter	Deklaration	Datentyp	Beschreibung
LT	INPUT	DATE_AND_TIME	Local time
WS_DAT	INPUT	BLOCK_DB	Information on the time zone for standard/daylight saving switchover (Rule DB)
RET_VAL	OUTPUT	INT	Error code
LT	OUTPUT	DATE_AND_TIME	Base time

How It Works

The local time entered at input *LT* is converted to the base time using the data stored in a DB and applied to output *BT*. The DB contains the number of 30-minute units by which the base time and local time differ and the difference between daylight-saving time and standard time also in units of 30 minutes. (Rule DB) If the calculation results in a date overflow, this is indicated by a special return value.

"Forbidden Hour"

During the switchover from standard to daylight-saving time the local time is put forward one hour. This, however, means that the hour in between is not run through. If there is an *LT* (local time) within this hour, FC62 LT_BT "thinks" in daylight-saving time. This is reported with return value 4 or 5.

"Double Hour"

During the switchover from daylight-saving to standard time the local time is put back one hour. This, however, means that one hour is run through twice. (For CE(S)T the designators 2A and 2B apply). For an *LT* (local time) within this hour, no unique identification relative to a base time is possible. FC LT_BT receives an *LT* as an input parameter and must decide whether the time is standard or daylight-saving before converting it to BT. If the *LT* is within the double hour, the *LT* is interpreted as standard time. This is reported with return value 2 or 3.

Calling OBs

FC 62 LT_BT can be called in any priority class.

Call Environment

Internally, FC 62 uses the following functions. These functions must be loaded in your project with the numbers shown here. FC1 (AD_DT_TM), FC7 (DT_DAY), FC35 (SB_DT_TM)

Output Values / Errors

RET_VAL	LT	Description
0	Base time	Block executed error-free
1	Base time	No error, but date jump
2	Base time	The LT at the input is within the "double" hour
3	Base time	As 2, also date jump
4	Base time	The LT at the input is within the "forbidden" hour
5	Base time	As 4, also date jump
8082	DT#90-01-01-0:0:0	Invalid data in the rule data block

4.6.4 FC 63 - S_LTINT - Set time interrupt in local time

Description

The FC sets the required time-of-day interrupt at the set time. This time is output in local time.

Parameters

Parameter	Declaration	Data type	Description
OB_NR	INPUT	INT	No of the OB to be started (permitted 10 – 17)
SDT	INPUT	BLOCK_DB	Start date and time-of-day in local time (see SFC28)
PERIOD	INPUT	INT	Period from start point SDT: ■ W#16#0000 = once ■ W#16#0201 = every minute ■ W#16#0401 = every hour ■ W#16#1001 = daily ■ W#16#1201 = weekly ■ W#16#1401 = monthly ■ W#16#1801 = annually ■ W#16#2001 = at end of month

Parameter	Declaration	Data type	Description
WS_DAT	INPUT	DATE_AND_TIME	Information on the time zone for standard/daylight saving switchover (see above)
RET_VAL	OUTPUT	INT	Error code

How It Works

The local time entered at input *LT* is converted to the base time using the rule stored in a DB. The DB contains the number of 30-minute units by which the base time and local time differ and the difference between daylight-saving time and standard time also in units of 30 minutes (see above). The specified time-of-day interrupt OB is assigned parameter values and activated using the calculated base time. If the calculation results in a date overflow, this is indicated by a special return value.

"Forbidden Hour"

During the switchover from standard to daylight-saving time the local time is put forward one hour. This, however, means that the hour in between is not run through. If there is an *LT* (local time) within this hour, FC S_LTINT "thinks" in daylight-saving time. This is reported with return value 4 or 5.

"Double Hour"

During the switchover from daylight-saving to standard time the local time is put back one hour. This, however, means that one hour run through twice. (For CE(S)T the designators 2A and 2B apply). For an *LT* (local time) within this hour, no unique identification relative to a base time is possible. FC S_LTINT receives an *LT* as input parameter and must decide whether the time is standard or daylight-saving before converting it to *BT*. If the *LT* is within the double hour, the *LT* is interpreted as standard time. This is reported with return value 2 or 3.

Calling OBs

FC S_LTINT can be called in any priority class. Internally, FC S_LTINT uses the following functions. These functions must be loaded in your project with the numbers shown here. FC7 (DT_DAY), FC35 (SB_DT_TM)

Output Values / Errors

RET_VAL	Description
0	Block executed error-free
1	No error, but date jump
2	The LT at the input was within the "double" hour
3	As 2, also date jump
4	The LT at the input is within the "forbidden" hour
5	As 4, also date jump
8082	Invalid data in the rule data block
8090	Bad OB_NR parameter
8091	Bad SDT parameter
8092	Bad PERIOD parameter
80A1	The set start time is in the past
80A2	OB is not loaded
80A3	OB cannot be started