Application Software
Closed-Loop Control Toolbox

Reference Guide for Function Blocks

04/02 AWB2700-1451GB

MOELLER

We keep power under control.

1st published 2002, edition date 04/02

Author: Rainer Tenhagen
Editor: Thomas Kracht
Translator: Terence Osborn, Karin Klinke

Printed on bleached cellulose.
100 % free from chlorine and acid.

Contents

1 Basic Information and Technical Data

General

Tool for programming the IEC 1131-3 with easySoft-CoDeSys

The closed-loop control toolbox (CCT) can be integrated in **easySoft-CoDeSys** as a library. The function blocks are then available under the menu item "Add → Types → Standard function blocks".

What can be implemented with the CCT?

The CCT can be used for the following tasks:

- Standard closed-loop control; PID controller and PDM (pulse duration modulation, also known as pulse width modulation, PWM)
- Application solutions; optimised concept solutions can be implemented quickly by combining various types of function blocks, such as fuzzy logic and PID controllers.
- Special needs for closed-loop control; unusual functions, such as PDM based on noise shaping (suitable for highly dynamic processes) and autotuning controllers are also included in the CCT.
- The mathematical basis of the CCT can be used independent of the closed-loop control technology, to expand the range of a PLC's functions. As an example, there are function blocks for interpolation and trigonometric functions.

For which controllers can the CCT be used?

The CCT can be used with all controllers which can be programmed with easySoft-CoDeSys.

Scope of the CCT

The CCT consists of around 125 function blocks, divided into the following classes:

- Controllers: PID, 2-step, 3-step, autotuning
- Signal processing: scaling, filter
- Pulse duration modulation
- Simulations (requires no additional external tool): the simulation can be linked with a controller in the same main program.
- Fuzzy logic (requires no additional external tool)
- Mathematical functions: interpolations (up to 60 interpolation points), trigonometric functions, exponentiation, root extraction

Philosophy of the CCT

Hierarchical structure of the toolbox and function block classes and levels

The CCT has a modular structure consisting of several levels in the hierarchy (→ fig. 1). The level 1 function blocks have basic functions, such as limiters or mathematical tasks. Level 2 function blocks implement basic functions for closed-loop control, such as integrators or differentiators. Higher functions of closed-loop control are placed in level 3. The PID controller (→ fig. 1) illustrates how the upper-level function blocks access the function blocks of lower levels.

There are several advantages to the modular structure of the CCT:

- Self-contained functions can be tested and optimised.
- The upper-level function blocks often access the same lower-level function blocks, so the code is small compared with non modular programs.
- Complex algorithms can be implemented rapidly by combining the existing modular functions. Tested subblocks are used, which reduces programming errors.

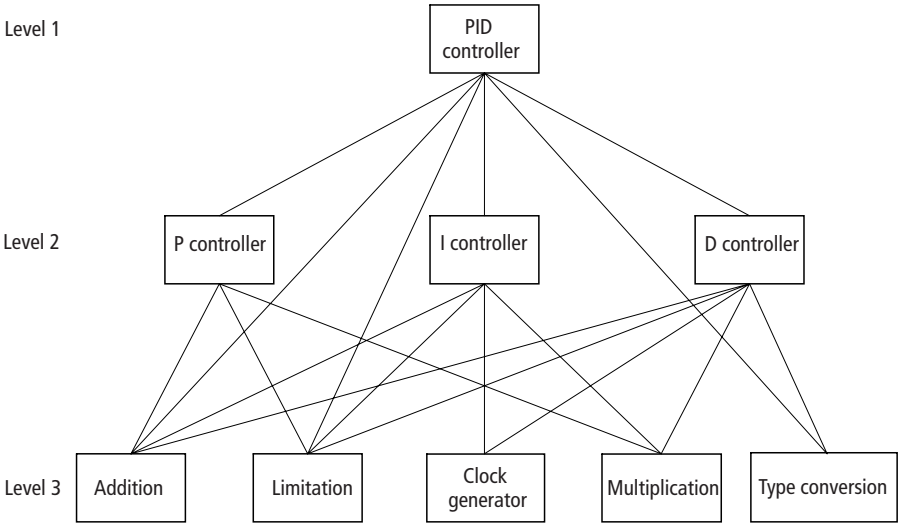


Figure 1: Hierarchical structure of the CCT function blocks in several levels, as exemplified by the PID controller

Self-explanatory variable and function block names

The names of the CCT variables and function blocks are detailed and self-explanatory. This makes it easy to learn how to use the CCT. Most function blocks can be added to the application program and parameterised without reference to the documentation.

The names of the function blocks are structured as follows:

- An initial "U_", so the entire library is arranged alphabetically, and appears together under the menu item "Application function blocks".
- There then follows an abbreviation (if there are several function blocks of the same type), so related function blocks are together. For example, all fuzzy logic function blocks start with "U_FUZ".
- If there are several function blocks of the same type, characters are added before the name to differentiate them, such as interpolations with data type UINT or INT.
- Finally, there is a descriptive name of the function block, e.g. PID_controller.

Examples:

U_LMA_INT_limit_monitor
=> Limit monitor for data type "Integer"

U_LMA_UINT_limit_monitor
=> Limit monitor for data type "unsigned Integer"

U_IP20_UINT_interpolation
=> 20-point interpolation for data type "unsigned Integer"

U_IP3_INT_interpolation
=> 3-point interpolation for data type "Integer"

The variable names are structured as follows:

- Initially a descriptive name, e.g. "setpoint".
- Then (if relevant) the unit or resolution, e.g. "12-bit", "percent" or "ms".
- The end of the variable name is, e.g. the data type "INT", "UINT" or "BOOL".

Examples:
Setpoint_12Bit_UINT
Ramp_time_ms_UINT
Tn_10thsec_UINT
Manipulated_variable_12Bit_UINT

Parameterising instead of programming

Using the CCT reduces your work from programming to simply entering parameters. This greatly shortens the time required to develop programs.

Minimizing application effort while maximizing functionality

The CCT function blocks are designed to be easy to use. This is accomplished by means of small function block interfaces, in conjunction with a wide selection of function blocks. As an example, for PID controllers there are the following selection criteria:

- PI controllers
- PID controllers
- Split-range PID controllers (heating/cooling)
- Autotuning PID controllers

The function blocks include a great deal of automatic processing. For instance, for PID controllers the following (→ chapter 4):

- Antiwindup procedures
- effective D computation (differentiation)
- standardized closed-loop control behaviour
- automatic setting of the internal scan time of the integrator and differentiator
- smooth acceptance of manual setpoint

Planning and handling

Planning information

Code is generated only once for each declared function block. Each additional instance (declaration with another instance name) simply creates another data field.

Due to the modular structure of the CCT, calling a single function block can lead to large code, files and function block instances. However, if function blocks are used which the PID autotuning controller uses as subroutine blocks, then the code does not grow larger. Thus, using many CCT function blocks reduces the relative size of the code per function block.

Commissioning a controller

When a PLC program is loaded with an untested controller, the control loop should not be closed. All of the equipment needs to be set up and tested, such as automation devices, actuators, sensors, etc. Then be sure to check if the controller function block is receiving the correct setpoint and actual values, and if the actuators are driven by the manipulated variable as intended. You might have to adjust the scaling and polarity.

Adjust the controller as follows:

- If the parameters are known, they can be used.
- For PTn systems, the autotuning control can parameterise itself. This powerful function block need only be used during commissioning. Replace it afterwards with the PID controller.
- For spontaneous changes of the manipulated variable, the reaction of the control system can usually be recorded (transfer function). With the system delay times and system dead times, use rules of thumb to calculate appropriate parameters.
- When optimising the parameters, take into account if the control loop must stabilise disturbance variables or changes of the setpoint.
- The control behaviour can be improved by including the disturbance variables in the control. They either can be measured, or correlate with a measureable process quantity.
- If the properties of the control system change (mass, volume, heat properties, etc.), then the control parameters can be adapted in infinitely variable steps using, e.g. a fuzzy logic function block. Or, in the simplest case, have the system toggle between two sets of parameters.

Simulations with the CCT

The Toolbox allows you to rapidly implement simulations for many applications. So you are recommended to first simulate the application. Afterwards, use the simulation to test the controllers for proper functioning. The following function blocks are especially suitable for simulations:

- interpolations for generating characteristic curves,
- PTn systems, dead time elements,
- ramps, oscillations.

Application example for multiple-zone temperature control

A multiple-zone temperature control is required for an extruder. Up to 36 zones will be controlled simultaneously; they influence one another. The following function blocks are required to implement the system:

- **PT1 filter**
  The input signals from the temperature sensors are smoothed and forwarded to the controller as the actual values.
- **PID split-range controller**
  For extruders, the zones are cooled and heated on an individual basis. For optimum control behaviour, an algorithm is required with two separate manipulated variable outputs for heating and cooling.
- **PDM**
  The PDM function block converts the two analogue manipulated variables of the controller to pulse duration-modulated digital signals. These signals directly drive the power contactors or semiconductor relays of the heating and cooling units.

Technical data and other information

Single and multiple instances

If a function block does not save any data for the next call, then it need only be instanced once (meaning that an instance name is assigned in the declaration part). Typical examples are the mathematical function blocks, the interpolation blocks and the fuzzy logic blocks.

Example:
In the application example "Interpol", the instance "two-point interpolation" is used for two different tasks. Initially, an analogue value is scaled to 12 bits. Then, a percentage value (+/–) is scaled to 14 bits.

Application example of the function block "U_Ip2_INT_interpolation" in the program "Interpol"

```
PROGRAM Interpol

VAR
    Two_point_INTERPOLATION : U_IP2_INT_INTERPOLATION ;
    Analog_Value_4_till_20_mA_WORD : WORD ;
    Analog_Value_12Bit_INT : INT ;
    Value_percent_INT : INT ;
    Value_14Bit_INT : INT ;
END_VAR

(*First call of the interpolation function block*)
(*=========================================*)
LD  Analog_Value_4_till_20_mA_WORD
WORD_TO_INT
ST  Two_point_INTERPOLATION.X_INT
```

```
CAL Two_point_INTERPOLATION(
    Suppress_extrapolation_BOOL :=1,
    X1_INT :=819,
    X2_INT :=4095,
    Y1_INT :=0,
    Y2_INT :=4095,
    Y_INT=>Analog_Value_12Bit_INT)

(*Second call of the interpolation function block*)
(*=============================================*)
CAL Two_point_INTERPOLATION(
    X_INT :=Value_percent_INT,
    Suppress_extrapolation_BOOL :=0,
    X1_INT :=-100,
    X2_INT :=100,
    Y1_INT :=0,
    Y2_INT :=16383,
    Y_INT=>Value_14Bit_INT)

END_PROGRAM
```

Multiple instances are required for function blocks which until the next call save data which is denoted below with an asterisk (*). Typical examples are integrators (including PI and PID controllers), PTn systems, weighted averaging, pulse duration modulation or dead time elements. If, for example, a PID controller is used for multiple zones, then there must be an instance for each zone.

Sample declaration for 5 zones:

```
VAR
    PID_Controller_Zone1 : U_PID_controller;
    PID_Controller_Zone2 : U_PID_controller;
    PID_Controller_Zone3 : U_PID_controller;
    PID_Controller_Zone4 : U_PID_controller;
    PID_Controller_Zone5 : U_PID_controller;
END_VAR
```

Instances per call and instance depth (hierarchy levels)

→ For technical reasons, these values might change.

Function block name	Sub- FBs	Instances	Instance depth
U_2_step_controller*	3	3	2
U_3_step_controller*	3	3	2
U_Add_INT_addition	2	2	1
U_Add_UINT_addition	2	2	1
U_adjustment_procedure	12	12	4
U_CHA_INT_change_input*	2	2	1
U_CHA_UINT_change_input*	2	2	1
U_controller _PDM_parameter_trans	23	45	5
U_CYC_cycle_time_computation*	2	2	1
U_CYCC_cycle_time_constant*	4	4	2
U_cyclecounter*	2	2	1
U_CYCM_cycle_time_max_value*	3	3	2
U_CYCS_cycle_time_setpoint_value	3	3	2
U_dead_time_delay*	13	22	3
U_Differentiation*	13	14	5
U_division	2	2	1

Function block name	Sub- FBs	Instances	Instance depth
U_DT1_system*	11	12	3
U_FR2_III_fractions_12Bit	2	2	1
U_FR2_INT_fractions_12Bit	2	2	1
U_FR2_UINT_fractions_12Bit	2	2	1
U_FRA_III_fractions	2	2	1
U_FRA_INT_fractions	2	2	1
U_FRA_UINT_fractions	2	2	1
U_FUZ_12_FUZZY_1I_2T	8	8	3
U_FUZ_13_FUZZY_1I_3T	9	9	3
U_FUZ_15_FUZZY_1I_5T	9	9	3
U_FUZ_17_FUZZY_1I_7T	9	9	3
U_FUZ_19_FUZZY_1I_9T	9	9	3
U_FUZ_22_FUZZY_2I_2T	8	9	3
U_FUZ_23_FUZZY_2I_3T	9	10	3
U_FUZ_25_FUZZY_2I_5T	9	10	3
U_FUZ_27_FUZZY_2I_7T	9	10	3
U_FUZ_32_FUZZY_3I_2T	9	25	3
U_FUZ_33_FUZZY_3I_3T	10	26	3
U_FUZ_35_FUZZY_3I_5T	7	7	3
U_FUZ_42_FUZZY_4I_2T	9	43	3
U_FUZ_43_FUZZY_4I_3T	10	44	4
U_FUZ_52_FUZZY_5I_2T	5	51	3
U_FUZ_53_FUZZY_5I_3T	6	52	4
U_FUZ_TL_TERM_LEFT	2	2	1
U_FUZ_TM_TERM_MIDDLE	2	2	1
U_FUZ_TR_TERM_RIGHT	2	2	1
U_FUZS12_weighted_output_variable	3	3	2
U_FUZS4_weighted_output_variable	3	3	2
U_FUZS8_weighted_output_variable	3	3	2

Function block name	Sub- FBs	Instances	Instance depth
U_hysteresis*	2	2	1
U_integrator*	9	11	3
U_IP10_INT_interpolation	6	6	3
U_IP10_UINT_interpolation	9	9	3
U_IP2_INT_interpolation	5	5	2
U_IP2_UINT_interpolation	8	8	2
U_IP20_INT_interpolation	6	6	3
U_IP20_UINT_interpolation	9	9	3
U_IP3_INT_interpolation	6	6	3
U_IP3_UINT_interpolation	9	9	3
U_IP4_INT_interpolation	6	6	3
U_IP4_UINT_interpolation	9	9	3
U_IP60_INT_interpolation	10	16	3
U_IP60_UINT_interpolation	10	16	3
U_LIM_INT_limiter	2	2	1
U_LIM_UINT_limiter	2	2	1
U_LMA_INT_limit_monitor	2	2	1
U_LMA_UINT_limit_monitor	2	2	1
U_LMR_INT_limit_monitor	2	2	1
U_LMR_UINT_limit_monitor	2	2	1
U_Mul_INT_multiplication	2	2	1
U_Mul_UINT_multiplication	2	2	1
U_off_delayed_timer	5	6	2
U_OSC_ms_sinusoidal_oscillation*	17	31	5
U_OSC_s_sinusoidal_oscillation*	17	31	5
U_OSC_saw_tooth_oscillation*	13	22	4
U_OSC_triangular_oscillation*	14	23	4

Function block name	Sub- FBs	Instances	Instance depth
U_OSZ1000_oscilloscope	7	17	2
U_OSZ8_oscilloscope	8	82	3
U_P_gain	4	4	2
U_PD_three_step_controller*	18	30	4
U_PDI_multi_variable_controller*	14	79	4
U_PDM_contactor*	5	5	2
U_PDM_dynamic_contactor*	7	7	2
U_PDM_solidstate* (conventional technique)	3	3	2
U_PDM_solidstate* (noise-shape technique)	3	3	2
U_PDM_splitrange*	6	10	3
U_PI_controller*	12	16	4
U_PI_split_range_controller*	22	39	4
U_PID_16Bit_controller	18	33	4
U_PID_autotuning_controller*	30	85	5
U_PID_controller*	15	29	4
U_PID_parameter_transfer	4	7	2
U_PID_split_range_controller*	22	39	4
U_PT1_16Bit_Filter	13	15	3
U_PT1_filter*	12	13	3
U_PT1_system*	12	14	3
U_PT3_filter*	13	39	4
U_PTn_system*	13	133	4
U_raising_to_power_n	3	3	2
U_Ramp_delay	11	13	4
U_rd_ms_ramp_delay	12	14	4
U_RMS_ramp*	10	10	3
U_RS_ramp*	10	10	3
U_scan_time_generator*	4	4	2
U_sign_change	2	2	1

Function block name	Sub- FBs	Instances	Instance depth
U_SIM_PTN_simulation*	17	149	5
U_SIM_Tg_Tu_Ks_simulation*	29	185	5
U_splitrange	2	2	1
U_square_root	2	2	1
U_SUB_INT_subtraction	2	2	1
U_SUB_UINT_subtraction	2	2	1
U_SUBA_INT_SUBSTRAKTION_ABS	2	2	1
U_SUBA_UINT_SUBSTRAKTION_ABS	2	2	1
U_threshold_value	2	2	1
U_TOB_INT_tolerance_band	2	2	1
U_TOB_UINT_tolerance_band	2	2	1
U_TRG_arc_SIN_COS_TAN	9	9	3
U_TRG_cosine	8	12	3
U_TRG_sine	10	16	3
U_TRG_tangent	10	17	3
U_TYPE_INT_type_conversion	2	2	1
U_TYPE_UINT_type_conversion	2	2	1
U_WMV_INT_weighted_averaging*	3	3	2
U_WMV_UINT_weighted_averaging*	3	3	2
U_ZSFB01_special_FB*	4	4	3
U_ZSFB02_special_FB*	14	28	3
U_ZSFB03_special_FB*	15	31	3

* Function block stores data until the next call.
So if the function block is used several times, multiple instances are required.

Special function blocks

Special function blocks do not have self-descriptive names. They should only be used when there is a special requirement. Be sure to read the relevant documentation when using such function blocks. The special function blocks include the following special solutions:

- Standard blocks with slight modifications (e.g. when parameters must be entered with a different unit)
- Function blocks whose use must be explained
- Special requirement which is not to appear in the menu of self-explanatory function blocks

Application solutions and other uses for the CCT

Combining and linking function blocks

The CCT can be used in many ways. For instance, "ready-made" standard functions such as a PID controller can be implemented in a program. However, this is only one of the strengths of the CCT. Its real potential is realized by combining and linking the function blocks. More sophisticated function blocks with special properties can be implemented, such as adaptive fuzzy logic PID designs. Another example is the PID autotuning controller, which was created by combining 29 function blocks. You can use your know-how to rapidly implement your designs.

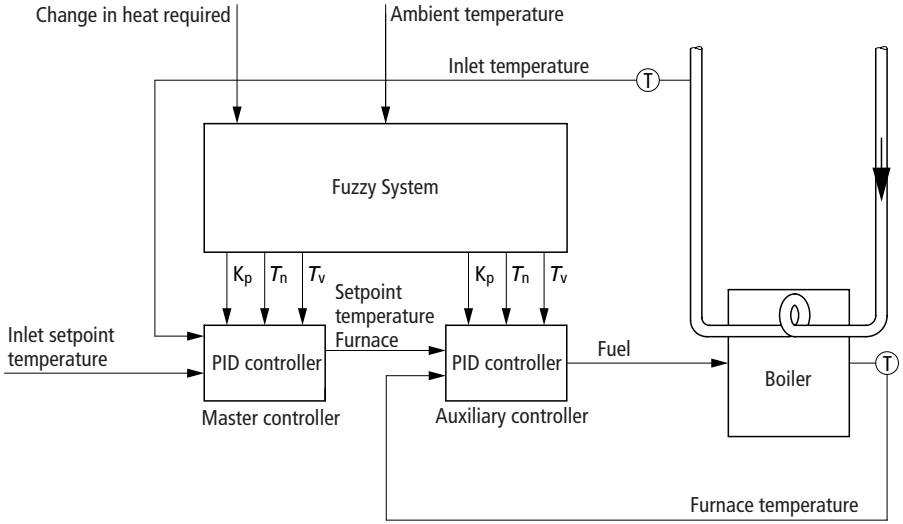


Figure 2: An adaptive design for boiler control is implemented by combining and cascading fuzzy logic and PID controller function blocks

Procedure for application solutions

With the CCT, simulations for common applications can be implemented rapidly. Therefore, you are recommended to first simulate the application and then to test the functionality of the solution with the simulation. Applications are simulated mostly with the following function blocks:

- Interpolation to implement desired characteristics
- PTn systems
- Dead time elements
- Ramps
- Oscillations

Using interpolation of characteristics to implement function curves

Any desired function curves can be approximated with the interpolation blocks of the CCT. The precision of the approximation depends mostly on the number of interpolation points. 2-, 3-, 4-, 10-, 20- and 60-point interpolation are available. For interpolation requiring more than 60 interpolation points, several interpolation function blocks can be combined.

Proven implementations with the CCT

Based on the CCT, the following control tasks have been solved:

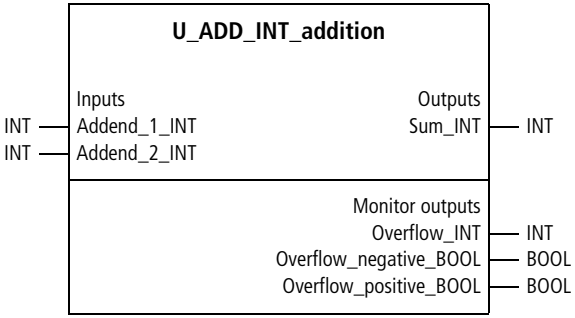
- Combined pressure/mixture controller for de-icing aeroplanes
- Dosing controller for the packaging industry
- Chlorine controller for inside and outside swimming pools
- Control of refrigerators for supermarkets
- Control of cooling units for ice skating rinks
- Temperature control of extruders (heating/cooling in one zone)
- Highly dynamic temperature control with autotuning
- Level control
- Pressure control
- Control tasks in buildings, e.g. controlling the temperature, humidity, pressure and air circulation
- Fuzzy logic control of printers
- Control designs for heating plants
- Control designs for combustion processes
- Control designs for sewage-treatment plants
- Control tasks in the chemical industry
- Temperature control for ball bearing test stands

2 Mathematical, logic and other Basic Function Blocks

Basic mathematical functions

Basic mathematical functions also includes function blocks for basic mathematical operations such as addition, subtraction, multiplication and division. These function blocks are secured against overflow violations. The function block "U_ADD_INT_addition" with integer variables (ranging between: –32768 and 32767) for the input and output calculates the following:

- Sum_INT = Addend_1_INT + Addend_2_INT



Interface of the function block "U_ADD_INT_addition"

Please note the following example:

Addition	Mathematical result	Function block "U_Add_INT_addition"	
		Result (with limitation of the value range)	Overflow
–32768 + (–1)	–32769	–32768	–1
–32768 + (–2)	–32770	–32768	–2
–20000 + (–25000)	–45000	–32768	–12232
32767 + 1	32768	32767	1
32767 + 2	32769	32767	2
20000 + 25000	45000	32767	12233

The table shows that an exact mathematical result that is outside of the value range can be represented and processed exactly by the output of the result and overflow in combination.

There are more reasons for using the function blocks of the basic mathematical functions.

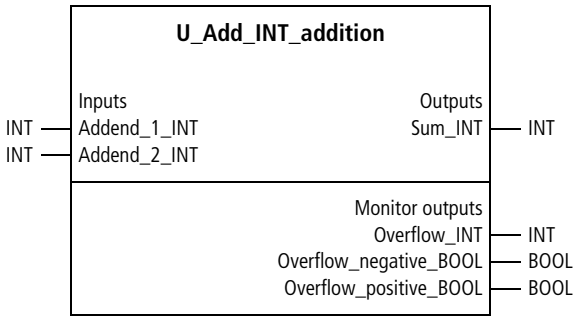
For example, subtracting two unsigned integer (UINT) variables is relatively complicated if the calculation involves the entire value range. The following must be taken into account:

- The result of a subtraction must be output as an integer variable, because the difference is negative, if the subtrahend is larger than the minuend (interface → the description of the function block "U_SUB_UINT_subtraction").

- Therefore, a type conversion from UINT to INT is required. The example: 50000 – 60000 = –10000 illustrates that a type conversion UINT_TO_INT for the minuend and subtrahend would cause an overflow and cannot be done. The calculation 50000 – 60000 also cannot be done for UINT variables, because the result –10000 also overflows the UINT value range (0 to 65535).
- The problems described above can be solved by separating some situations; this has been done and tested in the function block.

When an overflow is possible or there is a problem like with the function block "U_SUB_UINT_subtraction", you are recommended to program and test the required algorithms in a function block. This will help avoid careless mistakes when using this function frequently, and the multiple instances of a function block do not take more PLC memory space. In the complex toolbox function blocks, most calculations use the function blocks described in this chapter.

U_Add_INT_addition
Addition of integer values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Addend_1_INT	First addend	–32768 to 32767
Addend_2_INT	Second addend	–32768 to 32767
Outputs		
Sum_INT	Sum of addend 1 and 2	–32768 to 32767
Monitor outputs		
Overflow_INT	Out of data range	–32768 to 32767
Overflow_negativ_BOOL	Status indication: Overflow negative	0/1
Overflow_positiv_BOOL	Status indication: Overflow positive	0/1

Description

The function block calculates the following:

- Sum_INT = Addend_1_INT + Addend_2_INT

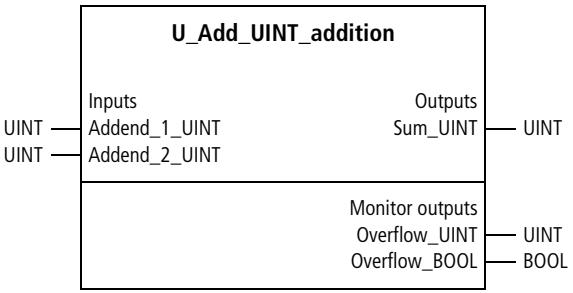
If "Sum_INT" exceeds the value range, then the limiting value is output, in contrast to the manufacturer function "Add" ... The monitor outputs display the magnitude and sign of the overflow (→ the beginning of the chapter "Basic mathematical functions").

Example:
–20000 + (–25000) = –45000
=> Sum_INT = –32768 (value range limitation)
=> Overflow_INT = –12232
=> Overflow_negative_BOOL = 1
=> Overflow_positive_BOOL = 0

Application of the function block "U_ADD_INT_addition" in the program "Addition"

```
PROGRAM Addition
VAR
  ADD_INT_ADDITION : U_ADD_INT_ADDITION ;
END_VAR

CAL ADD_INT_ADDITION(
  Addend_1_INT :=-20000,
  Addend_2_INT :=-25000,
  Sum_INT=>-32768,
  Overflow_INT=>-12232,
  Overflow_negative_BOOL=>1,
  Overflow_positive_BOOL=>0)

END_PROGRAM
```

U_Add_UINT_addition
Addition of unsigned integer values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Addend_1_UINT	First addend	0 to 65535
Addend_2_UINT	Second addend	0 to 65535
Outputs		
Sum_UINT	Sum of addend 1 and 2	0 to 65535
Monitor outputs		
Overflow_UINT	Exceeds data range limits	0 to 65535
Overflow_BOOL	Status indication: Overflow	0/1

Description

The function block calculates the following:

- Sum_UINT = Addend_1_UINT + Addend_2_UINT

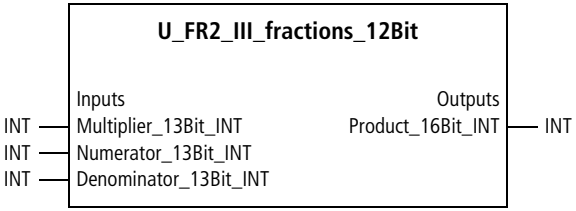
If "Sum_UINT" exceeds the value range, then the limiting value is output, in contrast to the manufacturer function "Add". The monitor outputs indicate if there is an overflow and its magnitude (→ the beginning of the chapter "Basic mathematical functions").

Example:
40000 + 35000 = 75000
=> Sum_UINT = 65535 (value range limiting)
=> Overflow_UINT = 9465
=> Overflow_BOOL = 1

Application of the function block "U_Add_UINT_addition" in the program "Sum2"

```
PROGRAM Sum2
VAR
  ADD_UINT_ADDITION : U_ADD_UINT_ADDITION ;
  a : UINT :=40000;
  b : UINT :=35000;
  c : UINT ;
END_VAR

CAL ADD_UINT_ADDITION(
  Addend_1_UINT :=a,
  Addend_2_UINT :=b,
  Sum_UINT=>65535,
  Overflow_UINT=>9465,
  Overflow_BOOL=>1)

END_PROGRAM
```

U_FR2_III_fractions_12Bit
Fractions, INT × INT/INT = INT



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Multiplier_13Bit_INT	multiplier	–4095 to 4095
Numerator_13Bit_INT	numerator	–4095 to 4095
Denominator_13Bit_INT	denominator	–4095 to 4095
Outputs		
Product_16Bit_INT	Result of the calculation: multiplier × numerator/denominator	–32768 to 32767

Description

The function block calculates the following:

- Product = multiplier × numerator/denominator

If "Product_INT" exceeds the value range, then the limiting value is output (→ the beginning of the chapter "Basic mathematical functions"). When the inputs are limited to 13 bits, calculation of the fraction requires less cycle time than when 16 bits are used.

Example:

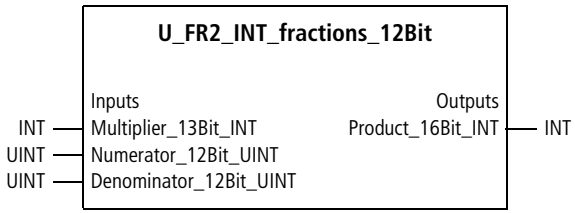
$$\frac{-4000 \times -88}{100} = -3520$$

→ The resolution of the output is 16 bits.

Application of the function block "U_FR2_III_fractions_12Bit" in the program "Percent"

```
PROGRAM Percent
VAR
  FR2_III_FRACTIONS_12BIT : U_FR2_III_FRACTIONS_12BIT ;
  a : INT :=-4000;
  b_percent : INT := -88;
END_VAR

CAL FR2_III_FRACTIONS_12BIT(
  Multiplier_13Bit_INT :=a,
  Numerator_13Bit_INT :=b_percent,
  Denominator_13Bit_INT :=-100,
  Product_16Bit_INT=>-3520
)
END_PROGRAM
```

U_FR2_INT_fractions_12Bit
Fractions, INT × UINT/UINT = INT



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Multiplier_13Bit_INT	multiplier	–4095 to 4095
Numerator_12Bit_UINT	numerator	0 to 4095
Denominator_12Bit_UINT	denominator	0 to 4095
Outputs		
Product_16Bit_INT	Result of the calculation: multiplier × numerator/denominator	–32768 to 32767

Description

The function block calculates the following:

- Product = multiplier × numerator/denominator

If "Product_16Bit_INT" exceeds the value range, then the limiting value is output (→ the beginning of the chapter "Basic mathematical functions"). When the inputs are limited to 12 bits (12 bits for the magnitude, so actually 13 bits for integer variables), calculation of the fraction requires less cycle time than when 16 bits are used.

→ The resolution of the output is 16 bits.

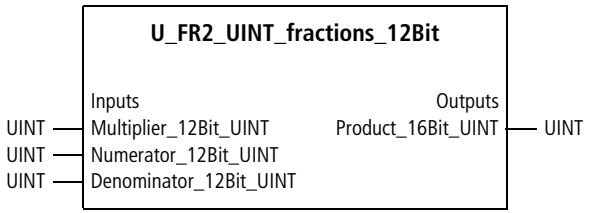
Example:

$$\frac{-4000 \times 588}{100} = 23520$$

Application of the function block "U_FR2_INT_fractions_12Bit" in the program "Percent"

```
PROGRAM Percent
VAR
  FR2_INT_FRACTIONS_12BIT : U_FR2_INT_FRACTIONS_12BIT ;
  a : INT :-4000 ;
  b_percent : UINT := 588 ;
  c : INT ;
END_VAR
CAL FR2_INT_FRACTIONS_12BIT(
  Multiplier_13Bit_INT :=a,
  Numerator_12Bit_UINT :=b_percent,
  Denominator_12Bit_UINT :=100,
  Product_16Bit_INT=>c)
END_PROGRAM
```

U_FR2_UINT_fractions_12Bit
Fractions, UINT × UINT/UINT = UINT



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Multiplier_12Bit_UINT	multiplier	0 to 4095
Numerator_12Bit_UINT	numerator	0 to 4095
Denominator_12Bit_UINT	denominator	0 to 4095
Outputs		
Product_16Bit_UINT	Result of the calculation: multiplier × numerator/denominator	0 to 65535

Description

The function block calculates the following:

- Product = multiplier × numerator/denominator

If "Product_UINT" exceeds the value range, then the limiting value is output (→ the beginning of the chapter "Basic mathematical functions"). When the inputs are limited to 12 bits, calculation of the fraction requires less cycle time than when 16 bits are used.

→ The input values may not exceed the permissible value range. The resolution of the output is 16 bits.

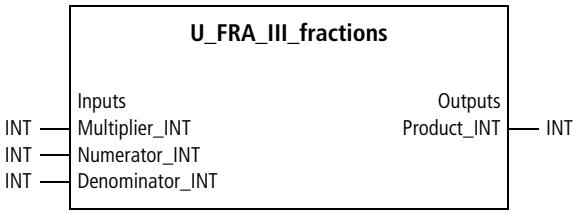
Example:

$$\frac{4111 \times 3333}{231} = 59315$$

Application of the function block "U_FR2_UINT_fractions_12Bit" in the program "C1_12Bit"

```
PROGRAM Cl_12Bit
VAR
  FR2_UINT_FRACTIONS_12BIT : U_FR2_UINT_FRACTIONS_12BIT ;
  a : UINT :=4111;
  b : UINT :=3333;
  c : UINT :=231;
  d : UINT ;
END_VAR
CAL FR2_UINT_FRACTIONS_12BIT(
  Multiplier_12Bit_UINT :=a,
  Numerator_12Bit_UINT :=b,
  Denominator_12Bit_UINT :=c,
  Product_16Bit_UINT=>d)
END_PROGRAM
```

U_FRA_III_fractions
Fractions, INT × INT/INT = INT



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Multiplier_INT	multiplier	–32768 to 32767
Numerator_INT	numerator	–32768 to 32767
Denominator_INT	denominator	–32768 to 32767
Outputs		
Product_INT	Result of the calculation: multiplier × numerator/denominator	–32768 to 32767

Description

The function block calculates the following:

- Product = multiplier × numerator/denominator

If "Product_INT" exceeds the value range, then the limiting value is output (⟶ the beginning of the chapter "Basic mathematical functions").

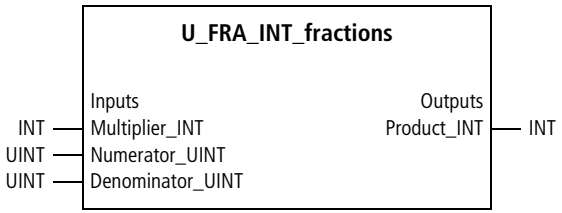
Example:

$$\frac{-19987 \times -88}{100} = 17588$$

Application of the function block "U_FRA_III_fractions" in the program "Percent"

```
PROGRAM Percent
VAR
  FRA_III_FRACTIONS : U_FRA_III_FRACTIONS ;
  a : INT :=-19987 ;
  b_percent : INT :=-88 ;
  c : INT;
END_VAR

CAL FRA_III_FRACTIONS(
  Multiplier_INT :=a,
  Numerator_INT :=b_percent,
  Denominator_INT :=-100,
  Product_INT=>c
)
END_PROGRAM
```

U_FRA_INT_fractions
Fractions, INT × UINT/UINT = INT



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Multiplier_INT	multiplier	–32768 to 32767
Numerator_UINT	numerator	0 to 65535
Denominator_UINT	denominator	0 to 65535
Outputs		
Product_INT	Result of the calculation: multiplier × numerator/denominator	–32768 to 32767

Description

The function block calculates the following:

- Product = multiplier × numerator/denominator

If "Product_INT" exceeds the value range, then the limiting value is output (⟶ the beginning of the chapter "Basic mathematical functions").

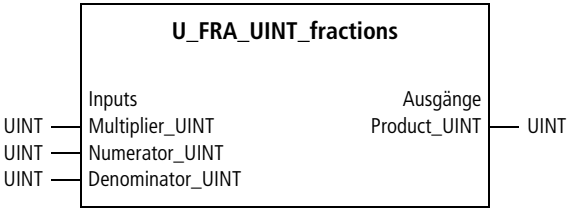
Example:

$$\frac{-9876 \times 88}{100} = -8690$$

Application of the function block "U_FRA_INT_fractions" in the program "Percent"

```
PROGRAM Percent
VAR
  FRA_INT_FRACTIONS : U_FRA_INT_FRACTIONS ;
  a : INT :=-9876 ;
  b_percent : UINT :=88 ;
  c : INT ;
END_VAR

CAL FRA_INT_FRACTIONS(
  Multiplier_INT :=a,
  Numerator_UINT :=b_percent,
  Denominator_UINT :=100,
  Product_INT=>c)
END_PROGRAM
```

U_FRA_UINT_fractions
Fractions, UINT × UINT/UINT = UINT



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Multiplier_UINT	multiplier	0 to 65535
Numerator_UINT	numerator	0 to 65535
Denominator_UINT	denominator	0 to 65535
Outputs		
Product_UINT	Result of the calculation: multiplier × numerator/denominator	0 to 65535

Description

The function block calculates the following:

- Product = multiplier × numerator/denominator

If "Product_UINT" exceeds the value range, then the limiting value is output (→ the beginning of the chapter "Basic mathematical functions").

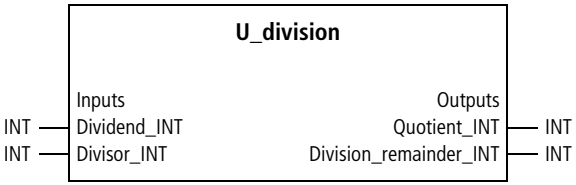
Example:

$$\frac{50000 \times 63333}{48888} = 64773$$

Application of the function block "U_FRA_UINT_fractions" in the program "C1_16Bit"

```
PROGRAM C1_16Bit
VAR
  FRA_UINT_FRACTIONS : U_FRA_UINT_FRACTIONS ;
  a : UINT :=50000;
  b : UINT :=63333;
  c : UINT :=48888;
  d : UINT ;
END_VAR

CAL FRA_UINT_FRACTIONS(
  Multiplier_UINT :=50000,
  Numerator_UINT :=63333,
  Denominator_UINT :=48888,
  Product_UINT=>d)

END_PROGRAM
```

U_division
Division of integer values and output of the non divisible remainder



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Dividend_INT	dividend (numerator)	0 to 65535
Divisor_INT	divisor (denominator)	0 to 65535
Outputs		
Quotient_INT	quotient (fraction)	–32768 to 32767
Division_remainder_INT	division_remainder (GetCarryRegister)	–32768 to 32767

Description

The function block calculates the following:

- Quotient = dividend/divisor

The nondivisible remainder is output.

Example:

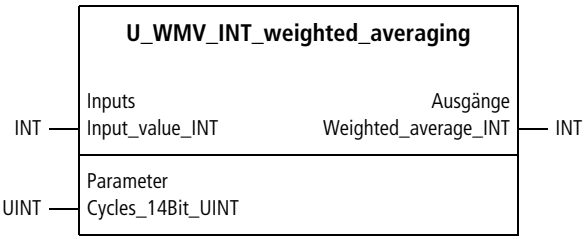
$$\frac{-4000}{-88} = 45$$

division remainder = –40

Application of the function block "U_division" in the program "Div_01"

```
PROGRAM Div_01
VAR
  DIVISION : U_DIVISION ;
END_VAR

CAL DIVISION(
  Dividend_INT :=-4000,
  Divisor_INT :=-88,
  Quotient_INT=>45,
  Division_remainder_INT=>-40)

END_PROGRAM
```

U_WMV_INT_weighted_averaging
Weighted average of integer values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_INT	input value	–32768 to 32767
Parameter		
Cycles_14Bit_UINT	PLC cycles used to calculate an average	0 to 16383
Outputs		
Weighted_average_INT	average resulting from the parameterised cycles and the (changing) input values	–32768 to 32767

Description

This function block calculates the average of the "Input values" of several PLC cycles, in accordance with the parameters of "Cycles_14Bit_UINT". For example, if you enter "Cycles_14Bit_UINT = 30", the average of the "Input values" of the last 30 PLC cycles will be calculated. The calculation of the average uses a weighting algorithm which may not result in the exact average (however, the average can be calculated over 16383 cycles).

If this function block is used several times, a new instance must be created each time because it saves data between the function block calls.

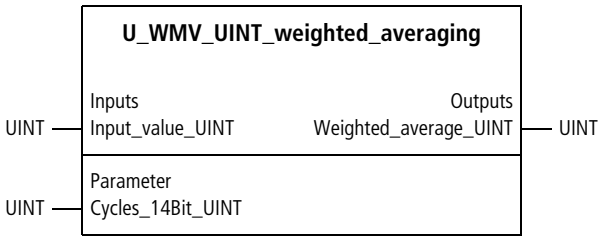
Use for signal smoothing:
Weighted averaging can be used to implement signal smoothing, similar to the function block "U_PT1_Filter". For example, signal smoothing over 10 s results from a cycle time of 20 ms (refer to the function block: U_cyc_cycle_time_computation), by calculating the weighted average over 500 cycles, since: 500 cycle × 20 ms/cycle = 10 s. The advantage of this function block compared to the PT1 filter block is the former requires much less cycle time (only 0.6 ms).

Example:
Assigning the parameter "Cycles_14Bit_UINT" the value 100 results in a weighted average of 100 PLC cycles.

Application of the function block "U_WMV_INT_weighted_averaging" in the program "signalsm"

```
PROGRAM signalsm
VAR
    signal_smoothing : U_WMV_INT_WEIGHTED_AVERAGING ;
    analog_value : WORD ;
    Actual_value_12Bit_INT : INT ;
END_VAR

LD  analog_value
WORD_TO_INT
ST  signal_smoothing.Input_value_INT
LD  100
ST  signal_smoothing.Cycles_14Bit_UINT
CAL signal_smoothing
LD  signal_smoothing.Weighted_average_INT
ST  Actual_value_12Bit_INT

END_PROGRAM
```

U_WMV_UINT_weighted_averaging
Weighted Average of Unsigned Integer Values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_UINT	input value	0 to 65535
Parameter		
Cycles_14Bit_UINT	PLC cycles over which an average is calculated	0 to 16383
Outputs		
Weighted_average_UINT	average, based on the parameterised cycles and the (changing) input values	0 to 65535

Description

This function block calculates the average of the "Input values" of several PLC cycles, in accordance with the parameters of "Cycles_14Bit_UINT". For example, if you enter "Cycles_14Bit_UINT = 30", the average of the "Input values" of the last 30 PLC cycles will be calculated. The calculation of the average uses a weighting algorithm which may not result in the exact average (however, the average can be calculated over 16383 cycles!).

→ If this function block is used several times, a new instance must be created each time because it saves data between the function block calls.

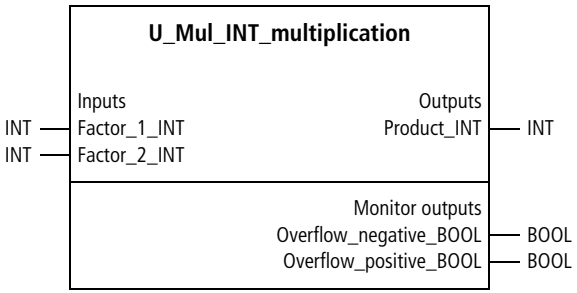
Use for signal smoothing:
Weighted averaging can be used to implement signal smoothing, similar to the function block "U_PT1_Filter". For example, signal smoothing over 10 s results from a cycle time of 20 ms (refer to the function block: U_cyc_cycle_time_computation), by calculating the weighted average over 500 cycles, since: 500 cycle × 20 ms/cycle = 10 s. The advantage of this function block compared to the PT1 filter block is the former requires much less cycle time (only 0.6 ms).

Example:
Assigning the parameter "Cycles_14Bit_UINT" the value 100 results in a weighted average of 100 PLC cycles.

Application of the function block "U_WMV_UINT_weighted_averaging" in the program "signalsm"

```
PROGRAM signalsm
VAR
    signal_smoothing : U_WMV_UINT_WEIGHTED_AVERAGING ;
    analog_value : WORD ;
    Actual_value_12Bit_UINT : UINT ;
END_VAR

LD  analog_value
WORD_TO_UINT
ST  signal_smoothing.Input_value_UINT
LD  100
ST  signal_smoothing.Cycles_14Bit_UINT
CAL signal_smoothing
LD  signal_smoothing.Weighted_average_UINT
ST  Actual_value_12Bit_UINT

END_PROGRAM
```

U_Mul_INT_multiplication
Multiplication of Integer Values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Factor_1_INT	First factor of a multiplication	–32768 to 32767
Factor_2_INT	Second factor of a multiplication	–32768 to 32767
Outputs		
Product_INT	Product of both factors	–32768 to 32767
Monitor outputs		
Overflow_negative_BOOL	Status indication: Overflow negative	0/1
Overflow_positive_BOOL	Status indication: Overflow positive	0/1

Description

This function block calculates the product of two factors, based on the equation:

- Product_INT = factor_1_INT × factor_2_INT

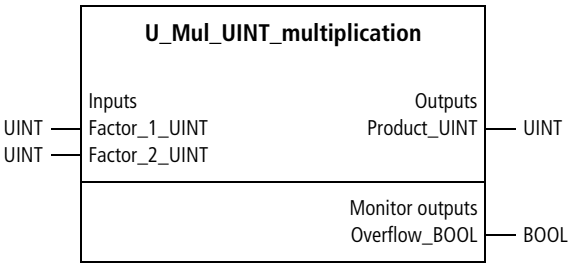
If "Product_INT" exceeds the value range, then the limiting value is output (→ the beginning of the chapter "Basic mathematical functions"). The monitor outputs indicate if the overflow is positive or negative.

Example:
–4000 × 1000 = –4000000
=> Product_INT = –32768 (value range limitation)
=> Overflow_negative_BOOL = 1
=> Overflow_positive_BOOL = 0

Application of the function block "U_Mul_INT_multiplication" in the program "mul_lim"

```
PROGRAM mul_lim
VAR
  MUL_INT_MULTIPLICATION : U_MUL_INT_MULTIPLICATION ;
  a : INT :=-4000;
END_VAR

CAL MUL_INT_MULTIPLICATION(
  Factor_1_INT :=-4000,
  Factor_2_INT :=1000,
  Product_INT=>-32768,
  Overflow_negative_BOOL=>1,
  Overflow_positive_BOOL=>0)

END_PROGRAM
```

U_Mul_UINT_multiplication
Multiplication of Unsigned Integer Values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Factor_1_UINT	First factor for multiplication	0 to 65535
Factor_2_UINT	Second factor for multiplication	0 to 65535
Outputs		
Product_UINT	The product of two factors	0 to 65535
Monitor outputs		
Overflow_BOOL	Status indication: Overflow negative	0/1

Description

This function block calculates the product of two factors, based on the equation:

- Product_UINT = Factor_1_UINT × Factor_2_UINT

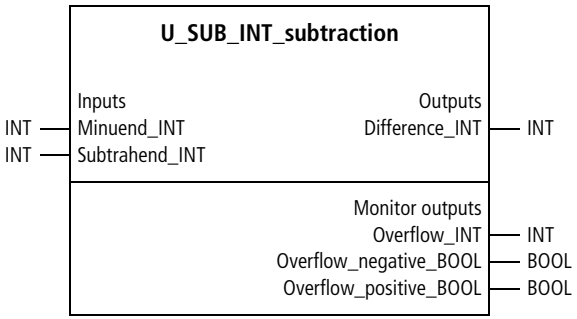
If "Product_UINT" exceeds the value range, then the limiting value is output (→ the beginning of the chapter "Basic mathematical functions").

Example:
2000 × 1000 = 2000000
=> Product_INT = 65535 (value range limitation)
=> Overflow_BOOL = 1

Application of the function block "U_Mul_UINT_multiplication" in the program "mul_lim"

```
PROGRAM mul_lim
VAR
  MUL_UINT_MULTIPLICATION : U_MUL_UINT_MULTIPLICATION ;
  a : UINT :=2000;
END_VAR

CAL MUL_UINT_MULTIPLICATION(
  Factor_1_UINT :=a,
  Factor_2_UINT :=1000,
  Product_UINT=>65535,
  Overflow_BOOL=>1)

END_PROGRAM
```

U_SUB_INT_subtraction
Subtraction of Integer Values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Minuend_INT	minuend of a subtraction	–32768 to 32767
Subtrahend_INT	subtrahend of a subtraction	–32768 to 32767
Outputs		
Difference_INT	difference of minuend and subtrahend	–32768 to 32767
Monitor outputs		
Overflow_INT	Outside of data range	–32768 to 32767
Overflow_negative_BOOL	Status indication: Overflow negative	0/1
Overflow_positive_BOOL	Status indication: Overflow positive	0/1

Description

The function block calculates the following:

- Difference_INT = Minuend_INT – Subtrahend_INT

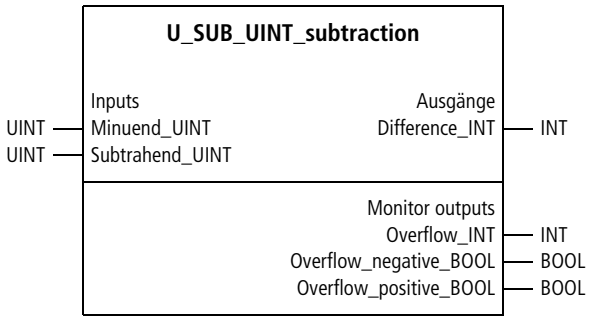
If "Difference_INT" exceeds the value range, then the limiting value is output, in contrast to the manufacturer function "sub" (→ the beginning of the chapter "Basic mathematical functions").

Example:
–20000 – 30000 = –50000
=> Difference_INT = –32768 (value range limitation)
=> Overflow_INT = –17232
=> Overflow_negative_BOOL = 1
=> Overflow_positive_BOOL = 0

Application of the function block "U_SUB_INT_subtraction" in the program "Sub_lim"

```
PROGRAM Sub_lim
VAR
  SUB_INT_SUBTRACTION : U_SUB_INT_SUBTRACTION ;
  a : INT :=-20000 ;
  b : INT :=30000 ;
  c : INT ;
END_VAR

CAL SUB_INT_SUBTRACTION(
  Minuend_INT := a,
  Subtrahend_INT :=b,
  Difference_INT=>-32768,
  Overflow_INT=>-17232,
  Overflow_negative_BOOL=>1,
  Overflow_positive_BOOL=>0)

END_PROGRAM
```

U_SUB_UINT_subtraction
Subtraction of Unsigned Integer Values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Minuend_UINT	minuend of a subtraction	0 to 65535
Subtrahend_UINT	subtrahend of a subtraction	0 to 65535
Outputs		
Difference_INT	difference of minuend and subtrahend	–32768 to 32767
Monitor outputs		
Overflow_INT	Outside of data range	–32768 to 32767
Overflow_negative_BOOL	Status indication: Overflow negative	0/1
Overflow_positive_BOOL	Status indication: Overflow positive	0/1

Description

The function block calculates the following:

- Difference_INT = Minuend_UINT – Subtrahend_UINT

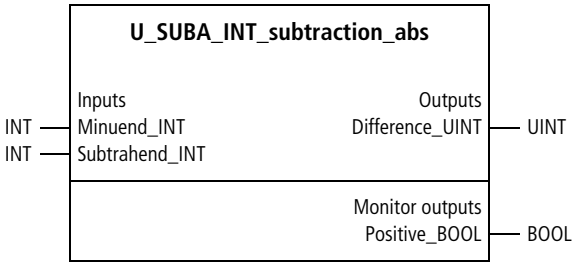
If "Difference_INT" exceeds the value range, then the limiting value is output, in contrast to the manufacturer function "sub" (→ the beginning of the chapter "Basic mathematical functions"). The monitor outputs show the overflow size and sign.

Example:
64988 – 65000 = –12
=> difference_INT = –12 (within the value range)

Application of the function block "U_SUB_UINT_subtraction" in the program "Sub_lim"

```
PROGRAM Sub_lim
VAR
  SUB_UINT_SUBTRACTION : U_SUB_UINT_SUBTRACTION ;
  a : UINT :=64988 ;
  b : UINT :=65000 ;
  c : INT ;
END_VAR

CAL SUB_UINT_SUBTRACTION(
  Minuend_UINT :=a,
  Subtrahend_UINT :=b,
  Difference_INT=>-12,
  Overflow_INT=>0,
  Overflow_negative_BOOL=>0,
  Overflow_positive_BOOL=>0)
END_PROGRAM
```

U_SUBA_INT_subtraction_abs
Subtraction with absolute Difference Output and Interger Values



Function block prototype

Meaning of operands

Designation	Significance	Value range
Inputs		
Minuend_INT	Minuend of a subtraction	–32768 to 32767
Subtrahend_INT	Subtrahend of a subtraction	–32768 to 32767
Outputs		
Difference_UINT	Absolute value and difference of minuend and subtrahend	0 to 65535
Monitor outputs		
Positive_BOOL	Indicates positive difference (1) or negative (0)	0/1

Description

The function block peforms the following calculation:

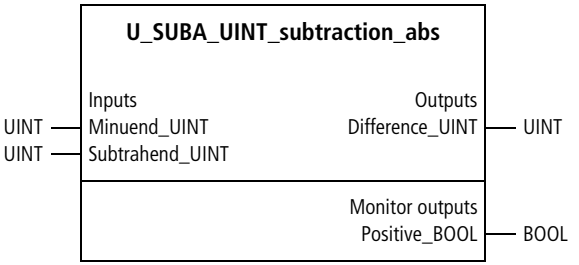
- Difference_UINT = Absolute value (Minuend_INT – Subtrahend_INT)

"Positive_BOOL=1" indicates that the difference is positive. "Positive_BOOL=0" indicates that the difference is negative.

Application of the function block "U_SUBA_INT_subtraction_abs" in the program "Subtra_i"

```
PROGRAM Subtra_i

VAR
  U_suba_INT_subtraction_abs : U_suba_INT_subtraction_abs ;
END_VAR

CAL U_suba_INT_subtraction_abs(
  Minuend_INT :=-1000,
  Subtrahend_INT :=900,
  Difference_UINT=>1900,
  positive_BOOL=>0)

END_PROGRAM
```

U_SUBA_UINT_subtraction_abs
Subtraction with Absolute Difference Output and Unsigned Integer Values



Function block prototype

Meaning of operands

Designation	Significance	Value range
Inputs		
Minuend_UINT	Minuend of a subtraction	0 to 65535
Subtrahend_UINT	Subtrahend of a subtraction	0 to 65535
Outputs		
Difference_UINT	Absolute value and difference of minuend and subtrahend	0 to 65535
Monitor outputs		
positive_BOOL	Indicates positive difference (1) or negative (0)	0/1

Description

The function block peforms the following calculation:

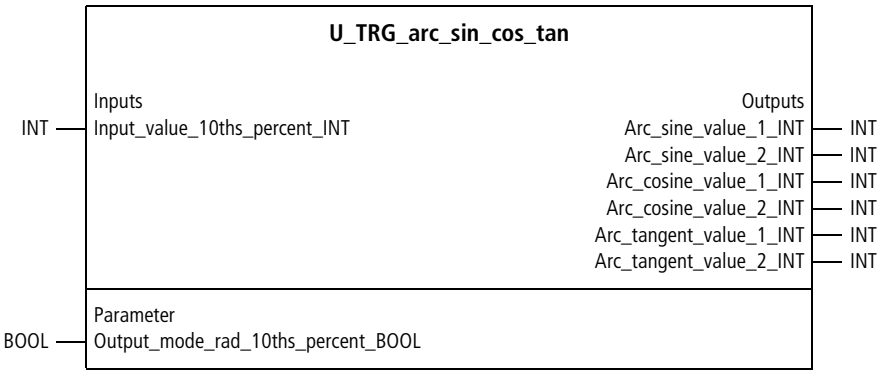
- Difference_UINT = Absolute value (Minuend_UINT – Subtrahend_UINT)

"Positive_BOOL=1" indicates that the difference is positive. "Positive_BOOL=0" indicates that the difference is negative.

Application of the function block "U_SUBA_UINT_subtraction_abs" in the program "Subtra_u"

```
PROGRAM Subtra_u

VAR
  Suba_UINT_Subtraction_abs : U_SUBA_UINT_subtraction_abs ;
END_VAR

CAL Suba_UINT_Subtraction_abs(
  Minuend_UINT :=1000,
  Subtrahend_UINT :=900,
  Difference_UINT=>100,
  positive_BOOL=>1)

END_PROGRAM
```

U_TRG_arc_sin_cos_tan
Arc Sine, Arc Cosine and Arc Tangent



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_10ths_ percent_INT	input value	sine, cosine: –1000 to 1000 tangent: –32768 to 32767
Parameter		
Output_mode_rad_10ths_ percent_BOOL	The trigonometric output values can be output in tenths of a degree (deg) or radians (rad) as a tenths percent value	0/1
Outputs		
Arc_sine_value_1_INT	arc sine value 1	0 to 6283
Arc_sine_value_2_INT	arc sine value 2	0 to 6283
Arc_cosine_value_1_INT	arc cosine value 1	0 to 6283
Arc_cosine_value_2_INT	arc cosine value 2	0 to 6283
Arc_tangent_value_1_INT	arc tangent value 1	0 to 6283
Arc_tangent_value_2_INT	arc tangent value 2	0 to 6283

Description

This function block takes the input value to calculate the arc sine, arc cosine and arc tangent. The trigonometric value is entered with the resolution ‰. The output value can be expressed in tenths of a degree (deg) for "Input_mode_rad_10ths_percent_BOOL:=0" and in radian measure (rad), as a tenths percent value, for "Input_mode_rad_10ths_percent_BOOL:=1". For the trigonometric arc functions, there are always two solutions and both are output.

Error tolerance:
The output values can deviate approx. 2 tenths of a degree (deg) or the corresponding radian measure (rad).

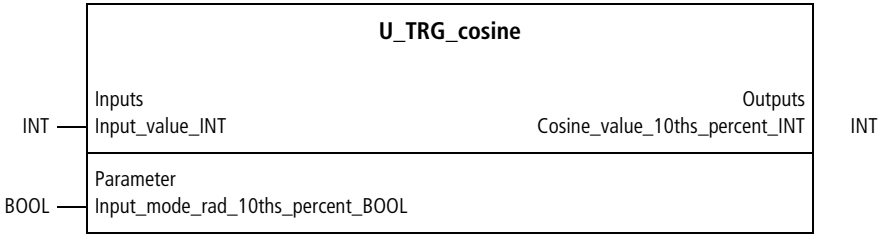
Example:
An input value of 0.5 (= 500 ‰) results in the following output values in tenths of a degree (→ application example):

- Arc sine = 300 and 1500
- Arc cosine = 3000 and 600
- Arc tangent = 260 and 2060

Application of the function block "U_TRG_arc_sin_cos_tan" in the program "arcustrg"

```
PROGRAM arcustrg
VAR
  TRG_ARC_SIN_COS_TAN : U_TRG_ARC_SIN_COS_TAN ;
END_VAR

CAL TRG_ARC_SIN_COS_TAN(
  Input_value_10th_percent_INT :=500,
  Output_mode_rad_10ths_percent_BOOL :=0,
  Arc_sine_value_1_INT=>300,
  Arc_sine_value_2_INT=>1500,
  Arc_cosine_value_1_INT=>3000,
  Arc_cosine_value_2_INT=>600,
  Arc_tangent_value_1_INT=>260,
  Arc_tangent_value_2_INT=>2060)

END_PROGRAM
```

U_TRG_cosine
Cosine Calculation



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_INT	Value from which the cosine is calculated. It can be entered either in tenths of a degree (deg) or in radian measure (rad) as a tenths percent value.	–32768 to 32767
Parameter		
Input_mode_rad_10ths_ percent_BOOL	Switches the input mode of the input value from tenths of a degree (deg) to radian measure (rad) Example: π = 3142 ‰ = 3.142	0/1
Outputs		
Cosine_value_10ths_ percent_INT	Cosine value with the resolution tenths percent Example: 500 ‰ => 0.5	–1000 to 1000

Description

The function block takes the input value to calculate the cosine value. The input value can be entered in tenths of a degree (deg) for "Input_mode_rad_10ths_percent_ BOOL:=0" and in radian measure (rad) for "Input_mode_rad_10ths_percent_BOOL:=1", as mentioned above. The cosine value is output with the resolution ‰ (example: 800 ‰ => 0.8).

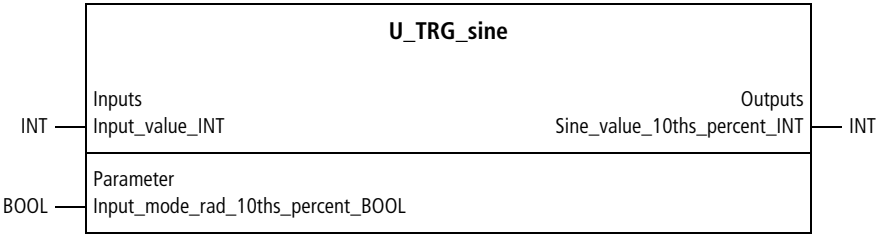
Error tolerance:
In the mode "Parameter entry in rad-tenths percent", the inaccuracy of the output value "Cosine_value_10ths_ percent_INT" is 2 ‰.

Example:
Cosine (deg) of 65.0° = 0.423
=> Cosine_value_10ths_percent_INT = 423

Application of the function block "U_TRG_cosine" in the program "CosinusA"

```
PROGRAM CosinusA
VAR
  TRG_COSINE : U_TRG_COSINE ;
END_VAR

CAL TRG_COSINE(
  Input_value_INT :=650,
  Input_mode_rad_10ths_percent_BOOL :=0
  )

END_PROGRAM
```

U_TRG_sine
Sine Calculation



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_INT	Value from which the sine is calculated. It can be entered either in tenths of a degree (deg) or in radian measure (rad) as a tenths percent value.	–32768 to 32767
Parameter		
Input_mode_rad_10ths_ percent_BOOL	Switches the input mode of the input value from tenths of a degree (deg) to radian measure (rad) Example: π = 3142 ‰ = 3.142	0/1
Outputs		
Sine_value_10ths_percent_INT	sine value, with the resolution tenths percent Example: 500 ‰ => 0.5	–1000 to 1000

Description

This function block takes the input value to calculate the sine value. The input value can be entered in tenths of a degree (deg) for "Input_mode_rad_10ths_percent_BOOL:=0" and in radian measure (rad) for "Input_mode_rad_10ths_percent_BOOL:=1", as mentioned above. The sinus value is output with the resolution ‰ (example: 800 ‰ => 0.8).

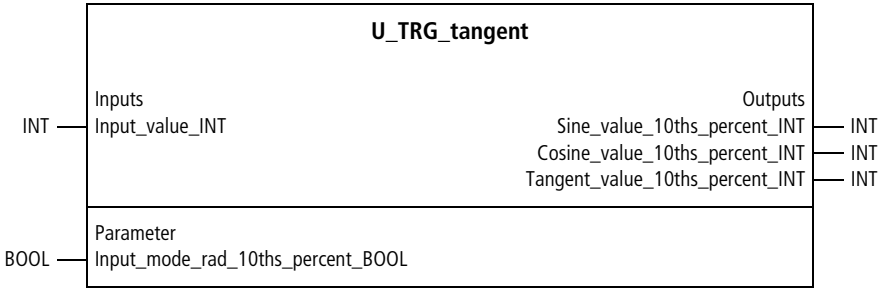
Error tolerance:
In the mode "Parameter entry in rad-tenths percent", the inaccuracy of the output value "Sine_value_10ths_percent_INT" is 2 ‰.

Example:
Sine (deg) of 420.0° = 0.866
=> Sine_value_10ths_percent_INT = 866

Application of the function block "U_TRG_sine" in the program "a_1"

```
PROGRAM a_1
VAR
  TRG_SINE : U_TRG_SINE ;
END_VAR

CAL TRG_SINE(
  Input_value_INT :=4200,
  Input_mode_rad_10ths_percent_BOOL :=0,
  Sine_value_10ths_percent_INT=>866)

END_PROGRAM
```

U_TRG_tangent
Tangent Calculation



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_INT	Value from which the sine, cosine and tangent are calculated. It can be entered either in tenths of a degree (deg) or in radian measure (rad) as a tenths percent value. Example: π = 3142 ‰ = 3,142	–32768 to 32767
Parameter		
Input_mode_rad_10ths_ percent_BOOL	Switches the input mode of the input value from tenths of a degree (deg) to radian measure (rad)	0/1
Outputs		
Sine_value_10ths_percent_INT	Sine value, with the resolution tenths percent Example: 500 ‰ => 0,5	–1000 to 1000
Cosine_value_10ths_ percent_INT	Cosine value, with the resolution tenths percent Example: –300 ‰ => –0,3	–1000 to 1000
Tangent_value_10ths_ percent_INT	Tangent value, with the resolution tenths percent Example: 20000 ‰ => 20	–32768 to 32767

Description

This function block takes the input value to calculate the sine, cosine and tangent value. The input value can be entered in tenths of a degree (deg) for "Input_mode_rad_10ths_percent_BOOL:=0" and in radian measure (rad) for "Input_mode_rad_10ths_percent_BOOL:=1", as mentioned above.

The trigonometric values are output with the resolution ‰ (example: 800 ‰ => 0.8).

Error tolerance:
In the mode "Parameter entry in rad-tenths percent", the inaccuracy of the output value is as follows:

- The maximum inaccuracy of sine and cosine is approx. 2 ‰.
- The maximum inaccuracy of tangent is approx. 4%.

Example:
Sine (rad) of 1.788 = 0.977
=> Sine_value_10ths_percent_INT = 977

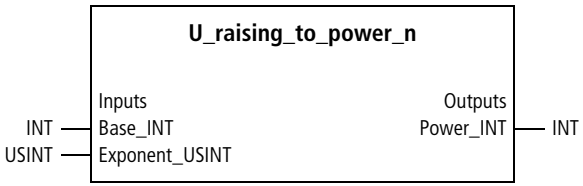
Cosine (rad) of 1.788 = –0.215
=> Cosine_value_10ths_percent_INT = –214

Tangent (rad) of 1.788 = –4.531
=> Tangent_value_10ths_percent_INT = –4565

Application of the function block "U_TRG_tangent" in the program "tangensA"

```
PROGRAM tangensA
VAR
  TRG_TANGENT : U_TRG_TANGENT ;
END_VAR

CAL TRG_TANGENT(
  Input_value_INT :=1788,
  Input_mode_rad_10ths_percent_BOOL :=1,
  Sine_value_10ths_percent_INT=>977,
  Cosine_value_10ths_percent_INT=>-214,
  Tangent_value_10ths_percent_INT=>-4565)

END_PROGRAM
```

U_raising_to_power_n
Raising integer values to power n



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Base_INT	base of a calculation of a power	–32768 to 32767
Exponent_USINT	exponent of a calculation of a power	0 to 255
Outputs		
Power_INT	power (result of the calculation of a power)	–32768 to 32767

Description

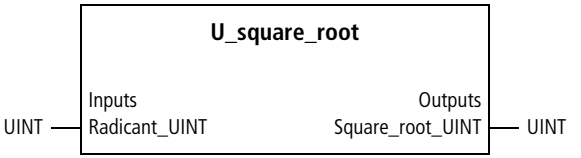
This function block raises a number (the base) to a power (the exponent). If "Power_INT" exceeds the value range, then the limiting value is output (→ the beginning of the chapter "Basic mathematical functions").

Example:
$5^6 = 15625$

Application of the function block "U_raising_to_power_n" in the program "expo6"

```
PROGRAM expo6
VAR
  RAISING_TO_POWER_N : U_RAISING_TO_POWER_N ;
END_VAR

CAL RAISING_TO_POWER_N(
  Base_INT :=5,
  Exponent_USINT :=6,
  Power_INT=>15625)

END_PROGRAM
```

U_square_root
Calculating the Square Root of an Unsigned Integer Value



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Radicant_UINT	Value whose square root is to be found	0 to 65535
Outputs		
Square_root_UINT	The square root of the radicand	0 to 65535

Description

This function block calculates the square root of the radicant.

Example:

$\sqrt{625} = 25$

Application of the function block "U_square_root" in the program "root_n"

```
PROGRAM root_n
VAR
  SQUARE_ROOT : U_SQUARE_ROOT ;
  n : UINT := 625 ;
END_VAR

CAL SQUARE_ROOT(
  Radicant_UINT :=n
)

END_PROGRAM
```

Interpolations

For linear interpolation, placing a straight line from one point to another creates an interpolation. Beyond the boundaries of the interpolation (here X1 and X3), extrapolation can be carried out (→ the above sketch) or the interpolation boundaries can be used (→ below sketch). The following functional relationship applies for the interpolation points (X1 | Y1) and (X2 | Y2) for interpolating (between the points) and extrapolating (beyond the points):

$$F(x) = Y = \frac{(X - X_1) \times (Y_1 - Y_2)}{X_1 - X_2} + Y_1$$

The graphic shows an arbitrary function. This function is approximated by using an interpolation with 3 X/Y interpolation points. The deviation between the function and the approximated interpolation depends on the number and position of the interpolation points and on the curvature of the function.

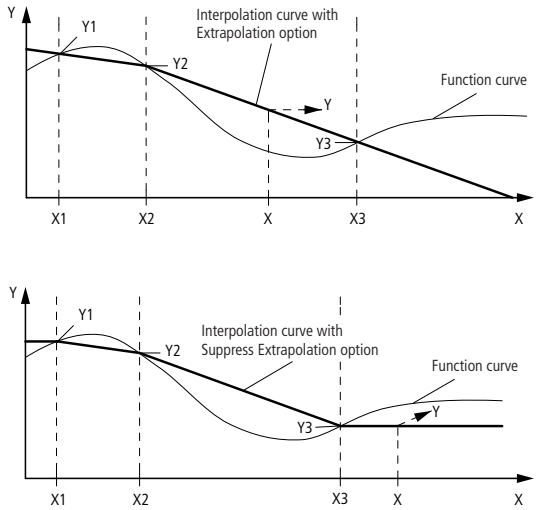


Figure 3: Interpolation

With extrapolation, the mathematical result may lie beyond the value range of an integer or unsigned integer. The function blocks of the toolbox then enter the boundary of the value range as the result (→ the beginning of the chapter "Basic mathematical functions").

Typical example of interpolation:

A characteristic curve is defined by ten interpolation points. The curve between the interpolation points is assumed to be linear. The function is created by interpolating between the interpolation points (→ fig. 4). Refer to the sample application programs of the function blocks "U_Ip10_INT_interpolation" and "U_Ip10_UINT_interpolation".

X	600	650	700	800	1000	1200	1500	1900	2500	2800
Y	0	500	1000	1500	2000	2500	3000	3500	4000	4095

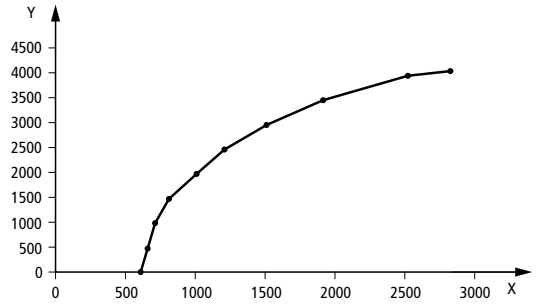
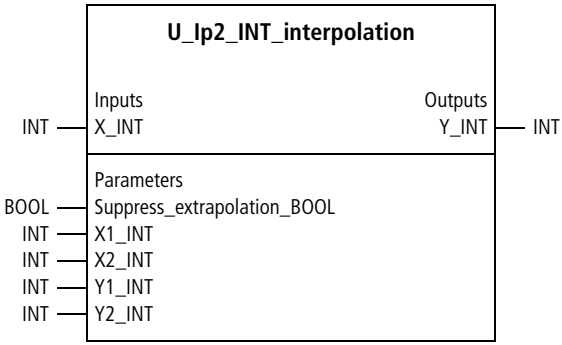


Figure 4: Creating a function by interpolation

U_Ip2_INT_interpolation
Interpolation with 2 X/Y Interpolation Points and Integer Values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
X_INT	Known X value	–32768 to 32767
Parameters		
Suppress_ extrapolation_BOOL	For X values beyond the interpolation boundaries, the following can be chosen: 0 => Extrapolate 1 => Extrapolating is suppressed. The interpolation boundaries are output	0/1
X1_INT	X value 1	–32768 to 32767
X2_INT	X value 2	–32768 to 32767
Y1_INT	Y value 1	–32768 to 32767
Y2_INT	Y value 2	–32768 to 32767
Outputs		
Y_INT	Interpolated (or extrapolated) Y value	–32768 to 32767

Description

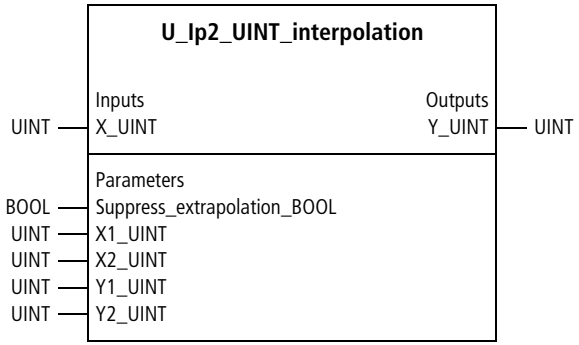
For the X value at the input, a linearly interpolated Y value is calculated between the X/Y interpolation points. Beyond the X/Y interpolation points, a linearly extrapolated Y value is calculated for "Suppress_extrapolation_BOOL:=0", for "Suppress_extrapolation_BOOL:=1" the Y interpolation limiting values are entered (→ fig. 3).

Example:
For the input parameters of the program listed below, the input value "Analogue_value_4_to_20_mA_ WORD:=1500" results in the output value "Analogue_value_12Bit_INT:=852".

Application of the function block "U_Ip2_INT_interpolation" in the program "A_4_20mA"

```
PROGRAM A_4_20mA
VAR
    Scaling_12bit : U_IP2_INT_INTERPOLATION ;
    Analog_Value_4_till_20_mA_WORD : WORD ;
    Analog_Value_12Bit_INT : INT ;
END_VAR

LD  Analog_Value_4_till_20_mA_WORD
WORD_TO_INT
ST  Scaling_12bit.X_INT
CAL Scaling_12bit(
    Suppress_extrapolation_BOOL :=1,
    X1_INT :=819,
    X2_INT :=4095,
    Y1_INT :=0,
    Y2_INT :=4095,
    Y_INT=>Analog_Value_12Bit_INT)

END_PROGRAM
```

U_Ip2_UINT_interpolation
Interpolation with 2 X/Y Interpolation Points and Unsigned Integer Values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
X_UINT	Known X value	0 to 65535
Parameters		
Suppress_ extrapolation_BOOL	For X values beyond the interpolation boundaries, the following can be chosen: 0 => Extrapolate 1 => Extrapolating is suppressed. The interpolation boundaries are output	0/1
X1_UINT	X value 1	0 to 65535
X2_UINT	X value 2	0 to 65535
Y1_UINT	Y value 1	0 to 65535
Y2_UINT	Y value 2	0 to 65535
Outputs		
Y_UINT	Interpolated (or extrapolated) Y value	0 to 65535

Description

For the X value at the input, a linearly interpolated Y value is calculated between the X/Y interpolation points. Beyond the X/Y interpolation points, a linearly extrapolated Y value is calculated for "Suppress_extrapolation_BOOL:=0", for "Suppress_extrapolation_BOOL:=1" the Y interpolation limiting values are entered (→ fig. 3).

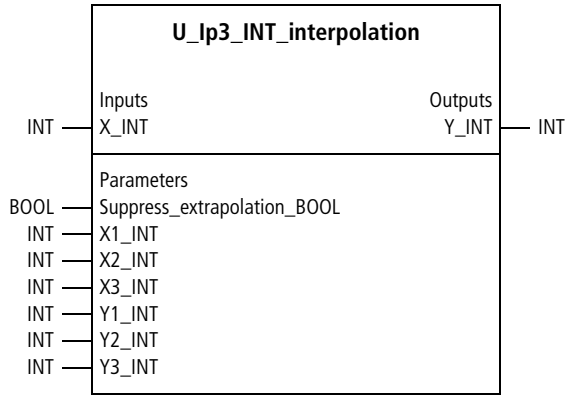
→ The X values must be entered in ascending order.

Example:
For the input parameters of the program listed below, the input value "Analogue_value_4_to_20_mA_ WORD:=2102" results in the output value "Analogue_value_12Bit_INT:=852".

Application of the function block "U_Ip2_UINT_interpolation" in the program "A_4_20mA"

```
PROGRAM A_4_20mA
VAR
    Scaling_12bit : U_IP2_UINT_INTERPOLATION ;
    Analog_Value_4_till_20_mA_WORD : WORD ;
    Analog_Value_12Bit_UINT : UINT ;
END_VAR

LD  Analog_Value_4_till_20_mA_WORD
WORD_TO_UINT
ST  Scaling_12bit.X_UINT
CAL Scaling_12bit(
    Suppress_extrapolation_BOOL :=1,
    X1_UINT :=819,
    X2_UINT :=4095,
    Y1_UINT :=0,
    Y2_UINT :=4095,
    Y_UINT=>Analog_Value_12Bit_UINT)

END_PROGRAM
```

U_Ip3_INT_interpolation
Interpolation with 3 X/Y Interpolation Points and Integer Values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
X_INT	Known X value	–32768 to 32767
Parameters		
Suppress_ extrapolation_BOOL	For X values beyond the interpolation boundaries, the following can be chosen: 0 => Extrapolate 1 => Extrapolating is suppressed. The interpolation boundaries are output	0/1

Designation	Significance	Value range
X1_INT	X value 1	–32768 to 32767
X2_INT	X value 2	–32768 to 32767
X3_INT	X value 3	–32768 to 32767
Y1_INT	Y value 1	–32768 to 32767
Y2_INT	Y value 2	–32768 to 32767
Y3_INT	Y value 3	–32768 to 32767
Outputs		
Y_INT	Interpolated (or extrapolated) Y value	–32768 to 32767

Description

For the X value at the input, a linearly interpolated Y value is calculated between the X/Y interpolation points. Beyond the X/Y interpolation points, a linearly extrapolated Y value is calculated for "Suppress_extrapolation_BOOL:=0", for "Suppress_extrapolation_BOOL:=1" the Y interpolation limiting values are entered (→ fig. 3).

→ The X values must be entered in ascending order.

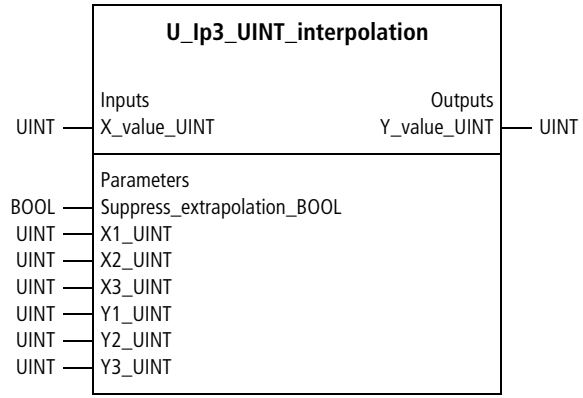
Example:
The application program converted the following characteristic curve:

	1	2	3
X	0	800	4095
Y	–500	–1000	–2000

Application of the function block "U_Ip3_INT_interpolation" in the program "charac03"

```
PROGRAM charac03
VAR
    Characteristic_curve_3point: U_IP3_INT_INTERPOLATION ;
    Analog_Value_12Bit_WORD : WORD ;
    Characteristic_curve_value_INT : INT ;
END_VAR

LD  Analog_Value_12Bit_WORD
WORD_TO_INT
ST  Characteristic_curve_3point.X_INT
CAL Characteristic_curve_3point(
    Suppress_extrapolation_BOOL :=1,
    X1_INT :=0,
    X2_INT :=800,
    X3_INT :=4095,
    Y1_INT :=-500,
    Y2_INT :=-1000,
    Y3_INT :=-2000,
    Y_INT=>Characteristic_curve_value_INT
)

END_PROGRAM
```

U_Ip3_UINT_interpolation
Interpolation with 3 X/Y Interpolation Points and Unsigned Integer Values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
X_value_UINT	Known X value	0 to 65535
Parameters		
Suppress_ extrapolation_BOOL	For X values beyond the interpolation boundaries, the following can be chosen: 0 => Extrapolate 1 => Extrapolating is suppressed. The interpolation boundaries are output	0/1

Designation	Significance	Value range
X1_UINT	X value 1	0 to 65535
X2_UINT	X value 2	0 to 65535
X3_UINT	X value 3	0 to 65535
Y1_UINT	Y value 1	0 to 65535
Y2_UINT	Y value 2	0 to 65535
Y3_UINT	Y value 3	0 to 65535
Outputs		
Y_value_UINT	Interpolated (or extrapolated) Y value	0 to 65535

Description

For the X value at the input, a linearly interpolated Y value is calculated between the X/Y interpolation points. Beyond the X/Y interpolation points, a linearly extrapolated Y value is calculated for "Suppress_extrapolation_BOOL:=0", for "Suppress_extrapolation_BOOL:=1" the Y interpolation limiting values are entered (→ fig. 3).

→ The X values must be entered in ascending order.

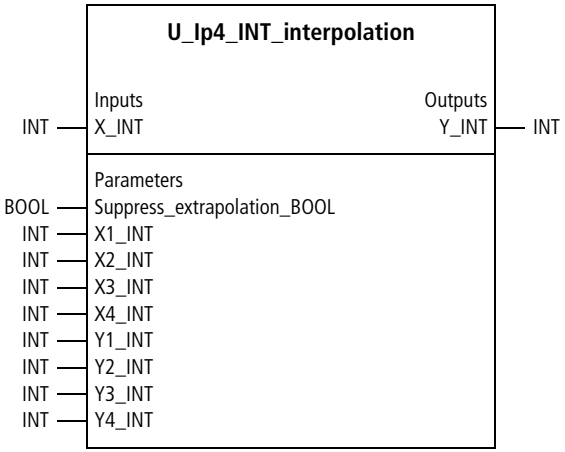
Example:
The application program converted the following characteristic curve:

	1	2	3
X	0	800	4095
Y	500	1000	2000

Application of the function block "U_Ip3_UINT_interpolation" in the program "charac03"

```
PROGRAM charac03
VAR
    Characteristic_curve_3point : U_IP3_UINT_INTERPOLATION ;
    Analog_Value_12Bit_WORD : WORD ;
    Characteristic_curve_value_UINT : UINT ;
END_VAR

LD  Analog_Value_12Bit_WORD
WORD_TO_UINT
ST  Characteristic_curve_3point.X_UINT
CAL Characteristic_curve_3point(
    Suppress_extrapolation_BOOL :=1,
    X1_UINT :=0,
    X2_UINT :=800,
    X3_UINT :=4095,
    Y1_UINT :=500,
    Y2_UINT :=1000,
    Y3_UINT :=2000,
    Y_UINT=>Characteristic_curve_value_UINT
)

END_PROGRAM
```

U_Ip4_INT_interpolation
Interpolation with 4 X/Y Interpolation Points and Integer Values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
X_INT	Known X value	–32768 to 32767
Parameters		
Suppress_ extrapolation_BOOL	For X values beyond the interpolation boundaries, the following can be chosen: 0 => Extrapolate 1 => Extrapolating is suppressed. The interpolation boundaries are output	0/1

Designation	Significance	Value range
X1_INT	X value 1	–32768 to 32767
X2_INT	X value 2	–32768 to 32767
X3_INT	X value 3	–32768 to 32767
X4_INT	X value 4	–32768 to 32767
Y1_INT	Y value 1	–32768 to 32767
Y2_INT	Y value 2	–32768 to 32767
Y3_INT	Y value 3	–32768 to 32767
Y4_INT	Y value 4	–32768 to 32767
Outputs		
Y_INT	Interpolated (or extrapolated) Y value	–32768 to 32767

Description

For the X value at the input, a linearly interpolated Y value is calculated between the X/Y interpolation points. Beyond the X/Y interpolation points, a linearly extrapolated Y value is calculated for "Suppress_extrapolation_BOOL:=0", for "Suppress_extrapolation_BOOL:=1" the Y interpolation limiting values are entered (→ fig. 3).

→ The X values must be entered in ascending order.

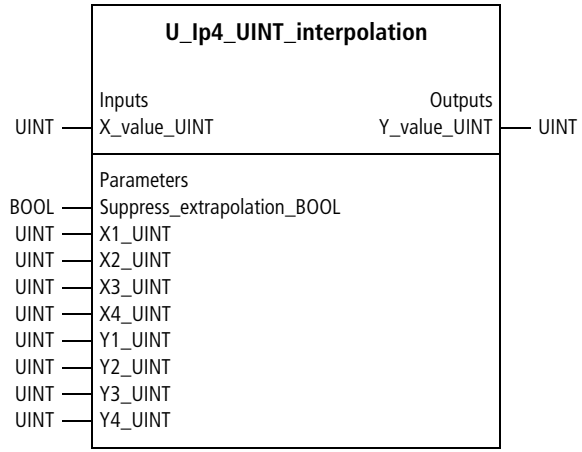
Example:
The application program converted the following characteristic curve:

	1	2	3	4
X	0	800	2500	4095
Y	–500	–1000	1000	2000

Application of the function block "U_Ip4_INT_interpolation" in the program "charac04"

```
PROGRAM charac04
VAR
    Characteristic_curve_4point : U_IP4_INT_INTERPOLATION ;
    Analog_Value_12Bit_WORD : WORD ;
    Characteristic_curve_value_INT : INT ;
END_VAR

LD  Analog_Value_12Bit_WORD
WORD_TO_INT
ST  Characteristic_curve_4point.X_INT
CAL Characteristic_curve_4point(
    Suppress_extrapolation_BOOL :=1,
    X1_INT :=0,
    X2_INT :=800,
    X3_INT :=2500,
    X4_INT :=4095,
    Y1_INT :=-500,
    Y2_INT :=-1000,
    Y3_INT :=1000,
    Y4_INT :=2000,
    Y_INT=>Characteristic_curve_value_INT
)

END_PROGRAM
```

U_Ip4_UINT_interpolation
Interpolation with 4 X/Y Interpolation Points and Unsigned Integer Values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
X_value_UINT	Known X value	0 to 65535
Parameters		
Suppress_ extrapolation_BOOL	For X values beyond the interpolation boundaries, the following can be chosen: 0 => Extrapolate 1 => Extrapolating is suppressed. The interpolation boundaries are output	0/1

Designation	Significance	Value range
X1_UINT	X value 1	0 to 65535
X2_UINT	X value 2	0 to 65535
X3_UINT	X value 3	0 to 65535
X4_UINT	X value 4	0 to 65535
Y1_UINT	Y value 1	0 to 65535
Y2_UINT	Y value 2	0 to 65535
Y3_UINT	Y value 3	0 to 65535
Y4_UINT	Y value 4	0 to 65535
Outputs		
Y_value_UINT	Interpolated (or extrapolated) Y value	0 to 65535

Description

For the X value at the input, a linearly interpolated Y value is calculated between the X/Y interpolation points. Beyond the X/Y interpolation points, a linearly extrapolated Y value is calculated for "Suppress_extrapolation_BOOL:=0", for "Suppress_extrapolation_BOOL:=1" the Y interpolation limiting values are entered (→ fig. 3).

→ The X values must be entered in ascending order.

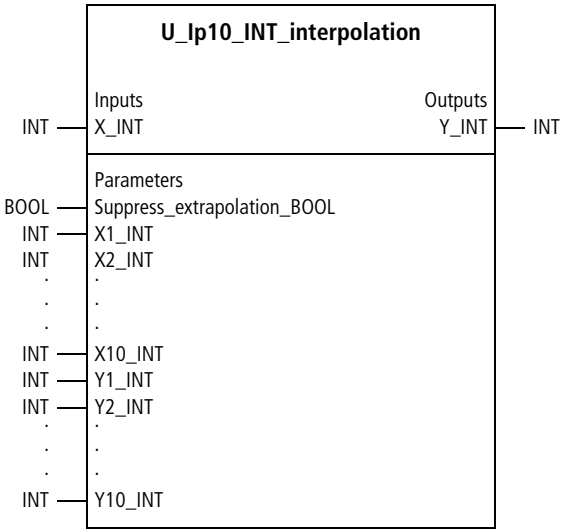
Example:
The application program converted the following characteristic curve:

	1	2	3	4
X	0	800	2500	4095
Y	500	1000	1500	2000

Application of the function block "U_Ip4_UINT_interpolation" in the program "charac04"

```
PROGRAM charac04
VAR
    Characteristic_curve_4point : U_IP4_UINT_INTERPOLATION ;
    Analog_Value_12Bit_WORD : WORD ;
    Characteristic_curve_value_UINT : UINT ;
END_VAR

LD  Analog_Value_12Bit_WORD
WORD_TO_UINT
ST  Characteristic_curve_4point.X_UINT
CAL Characteristic_curve_4point(
    Suppress_extrapolation_BOOL :=1,
    X1_UINT :=0,
    X2_UINT :=800,
    X3_UINT :=2000,
    X4_UINT :=4095,
    Y1_UINT :=500,
    Y2_UINT :=1000,
    Y3_UINT :=1500,
    Y4_UINT :=2000,
    Y_UINT=>Characteristic_curve_value_UINT
)

END_PROGRAM
```

U_Ip10_INT_interpolation
Interpolation with 10 X/Y Interpolation Points and Integer Values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
X_INT	Known X value	–32768 to 32767
Parameters		
Suppress_ extrapolation_BOOL	For X values beyond the interpolation boundaries, the following can be chosen: 0 => Extrapolate 1 => Extrapolating is suppressed. The interpolation boundaries are output	0/1

Designation	Significance	Value range
X1_INT	X value 1	–32768 to 32767
X2_INT	X value 2	–32768 to 32767
X3_INT	X value 3	–32768 to 32767
X4_INT	X value 4	–32768 to 32767
X5_INT	X value 5	–32768 to 32767
X6_INT	X value 6	–32768 to 32767
X7_INT	X value 7	–32768 to 32767
X8_INT	X value 8	–32768 to 32767
X9_INT	X value 9	–32768 to 32767
X10_INT	X value 10	–32768 to 32767
Y1_INT	Y value 1	–32768 to 32767
Y2_INT	Y value 2	–32768 to 32767
Y3_INT	Y value 3	–32768 to 32767
Y4_INT	Y value 4	–32768 to 32767
Y5_INT	Y value 5	–32768 to 32767
Y6_INT	Y value 6	–32768 to 32767
Y7_INT	Y value 7	–32768 to 32767
Y8_INT	Y value 8	–32768 to 32767
Y9_INT	Y value 9	–32768 to 32767
Y10_INT	Y value 10	–32768 to 32767
Outputs		
Y_value_INT	Interpolated (or extrapolated) Y value	–32768 to 32767

Description

For the X value at the input, a linearly interpolated Y value is calculated between the X/Y interpolation points. Beyond the X/Y interpolation points, a linearly extrapolated Y value is calculated for "Suppress_extrapolation_BOOL:=0", for "Suppress_extrapolation_BOOL:=1" the Y interpolation limiting values are entered (→ fig. 3).

→ The X values must be entered in ascending order.

Example:
The application program converted the following characteristic curve:

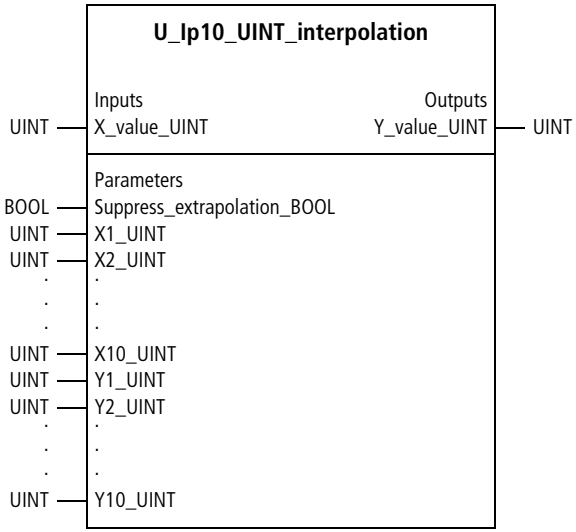
	1	2	3	4	5	6	7	8	9	10
X	0	100	200	300	400	500	600	700	800	900
Y	–50	–100	–200	–400	–800	–1600	–3200	–6400	–12800	–25600

Application of the function block "U_Ip10_INT_interpolation" in the program "charac10"

```
PROGRAM charac10
VAR
    Characteristic_curve_10point : U_IP10_INT_INTERPOLATION ;
    Analog_Value_12Bit_WORD : WORD ;
    Characteristic_curve_value_INT : INT ;
END_VAR

LD  Analog_Value_12Bit_WORD
WORD_TO_INT
ST  Characteristic_curve_10point.X_INT
```

```
CAL Characteristic_curve_10point(
    Suppress_extrapolation_BOOL :=1,
    X1_INT :=0,
    X2_INT :=100,
    X3_INT :=200,
    X4_INT :=300,
    X5_INT :=400,
    X6_INT :=500,
    X7_INT :=600,
    X8_INT :=700,
    X9_INT :=800,
    X10_INT :=900,
    Y1_INT :=-50,
    Y2_INT :=-100,
    Y3_INT :=-200,
    Y4_INT :=-400,
    Y5_INT :=-800,
    Y6_INT :=-1600,
    Y7_INT :=-3200,
    Y8_INT :=-6400,
    Y9_INT :=-12800,
    Y10_INT :=-25600,
    Y_INT=>Characteristic_curve_value_INT
)

END_PROGRAM
```

U_Ip10_UINT_interpolation
Interpolation with 10 X/Y Interpolation Points and Unsigned Integer Values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
X_UINT	Known X value	0 to 65535
Parameters		
Suppress_ extrapolation_BOOL	For X values beyond the interpolation boundaries, the following can be chosen: 0 => Extrapolate 1 => Extrapolating is suppressed. The interpolation boundaries are output	0/1

Designation	Significance	Value range
X1_UINT	X value 1	0 to 65535
X2_UINT	X value 2	0 to 65535
X3_UINT	X value 3	0 to 65535
X4_UINT	X value 4	0 to 65535
X5_UINT	X value 5	0 to 65535
X6_UINT	X value 6	0 to 65535
X7_UINT	X value 7	0 to 65535
X8_UINT	X value 8	0 to 65535
X9_UINT	X value 9	0 to 65535
X10_UINT	X value 10	0 to 65535
Y1_UINT	Y value 1	0 to 65535
Y2_UINT	Y value 2	0 to 65535
Y3_UINT	Y value 3	0 to 65535
Y4_UINT	Y value 4	0 to 65535
Y5_UINT	Y value 5	0 to 65535
Y6_UINT	Y value 6	0 to 65535
Y7_UINT	Y value 7	0 to 65535
Y8_UINT	Y value 8	0 to 65535
Y9_UINT	Y value 9	0 to 65535
Y10_UINT	Y value 10	0 to 65535
Outputs		
Y_value_UINT	Interpolated (or extrapolated) Y value	0 to 65535

Description

For the X value at the input, a linearly interpolated Y value is calculated between the X/Y interpolation points. Beyond the X/Y interpolation points, a linearly extrapolated Y value is calculated for "Suppress_extrapolation_BOOL:=0", for "Suppress_extrapolation_BOOL:=1" the Y interpolation limiting values are entered (→ fig. 3).

→ The X values must be entered in ascending order.

Example:
The application program converted the following characteristic curve:

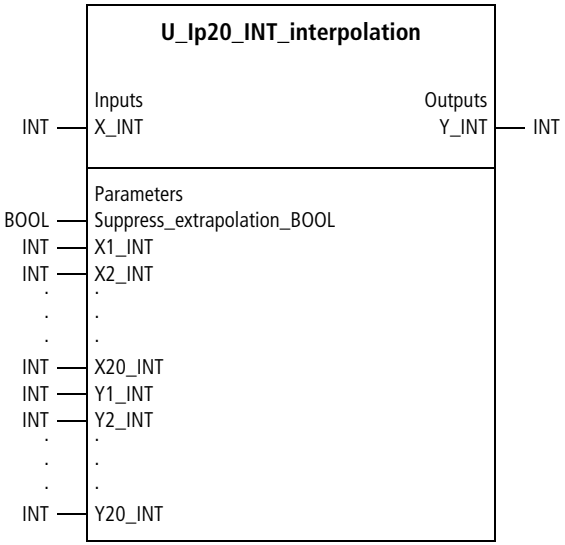
	1	2	3	4	5	6	7	8	9	10
X	1000	1500	2000	2500	3000	3500	4000	4095	4095	4095
Y	2000	5000	13000	9000	7200	7800	8000	10000	10000	10000

Application of the function block "U_Ip10_UINT_interpolation" in the program "charac10"

```
PROGRAM charac10
VAR
    Characteristic_curve_10point : U_IP10_UINT_INTERPOLATION ;
    Analog_Value_12Bit_WORD : WORD ;
    Characteristic_curve_value_UINT : UINT ;
END_VAR

LD  Analog_Value_12Bit_WORD
WORD_TO_UINT
ST  Characteristic_curve_10point.X_UINT
```

```
CAL Characteristic_curve_10point(
    Suppress_extrapolation_BOOL :=1,
    X1_UINT :=1000,
    X2_UINT :=1500,
    X3_UINT :=2000,
    X4_UINT :=2500,
    X5_UINT :=3000,
    X6_UINT :=3500,
    X7_UINT :=4000,
    X8_UINT :=4095,
    X9_UINT :=4095,
    X10_UINT :=4095,
    Y1_UINT :=2000,
    Y2_UINT :=5000,
    Y3_UINT :=13000,
    Y4_UINT :=9000,
    Y5_UINT :=7200,
    Y6_UINT :=7800,
    Y7_UINT :=8000,
    Y8_UINT :=10000,
    Y9_UINT :=10000,
    Y10_UINT :=10000,
    Y_UINT=>Characteristic_curve_value_UINT
)

END_PROGRAM
```

U_Ip20_INT_interpolation
Interpolation with 20 X/Y Interpolation Points and Integer Values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
X_INT	Known X value	–32768 to 32767
Parameters		
Suppress_ extrapolation_BOOL	For X values beyond the interpolation boundaries, the following can be chosen: 0 => Extrapolate 1 => Extrapolating is suppressed. The interpolation boundaries are output	0/1

Designation	Significance	Value range
X1_INT	X value 1	–32768 to 32767
X2_INT	X value 2	–32768 to 32767
X3_INT	X value 3	–32768 to 32767
X4_INT	X value 4	–32768 to 32767
X5_INT	X value 5	–32768 to 32767
X6_INT	X value 6	–32768 to 32767
X7_INT	X value 7	–32768 to 32767
X8_INT	X value 8	–32768 to 32767
X9_INT	X value 9	–32768 to 32767
X10_INT	X value 10	–32768 to 32767
X11_INT	X value 11	–32768 to 32767
X12_INT	X value 12	–32768 to 32767
X13_INT	X value 13	–32768 to 32767
X14_INT	X value 14	–32768 to 32767
X15_INT	X value 15	–32768 to 32767
X16_INT	X value 16	–32768 to 32767
X17_INT	X value 17	–32768 to 32767
X18_INT	X value 18	–32768 to 32767
X19_INT	X value 19	–32768 to 32767
X20_INT	X value 20	–32768 to 32767
Y1_INT	Y value 1	–32768 to 32767
Y2_INT	Y value 2	–32768 to 32767
Y3_INT	Y value 3	–32768 to 32767
Y4_INT	Y value 4	–32768 to 32767
Y5_INT	Y value 5	–32768 to 32767
Y6_INT	Y value 6	–32768 to 32767
Y7_INT	Y value 7	–32768 to 32767
Y8_INT	Y value 8	–32768 to 32767
Y9_INT	Y value 9	–32768 to 32767

Designation	Significance	Value range
Y10_INT	Y value 10	–32768 to 32767
Y11_INT	Y value 11	–32768 to 32767
Y12_INT	Y value 12	–32768 to 32767
Y13_INT	Y value 13	–32768 to 32767
Y14_INT	Y value 14	–32768 to 32767
Y15_INT	Y value 15	–32768 to 32767
Y16_INT	Y value 16	–32768 to 32767
Y17_INT	Y value 17	–32768 to 32767
Y18_INT	Y value 18	–32768 to 32767
Y19_INT	Y value 19	–32768 to 32767
Y20_INT	Y value 20	–32768 to 32767
Outputs		
Y_INT	Interpolated (or extrapolated) Y value	–32768 to 32767

Description

For the X value at the input, a linearly interpolated Y value is calculated between the X/Y interpolation points. Beyond the X/Y interpolation points, a linearly extrapolated Y value is calculated for "Suppress_extrapolation_BOOL:=0", for "Suppress_extrapolation_BOOL:=1" the Y interpolation limiting values are entered (→ fig. 3).

→ The X values must be entered in ascending order.

Example:
The application program converted the following characteristic curve. The result approximates a cosine function from 114 to 228° (deg).

	1	2	3	4	5	6	7	8	9	10
X	114	120	126	132	138	144	150	156	162	168
Y	914	866	809	743	669	588	500	407	309	208

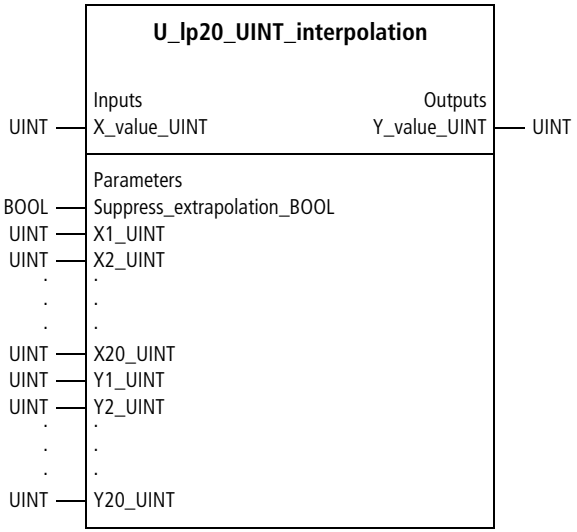
	11	12	13	14	15	16	17	18	19	20
X	174	180	186	192	198	204	210	216	222	228
Y	105	0	–105	–208	–309	–407	–500	–588	–669	–743

Application of the function block "U_Ip20_INT_interpolation" in the program "Cos_228"

```
PROGRAM Cos_228
VAR
    Cosine_approximation_114_till_228_rad : U_IP20_INT_INTERPOLATION ;
    Input_114_till_228_rad : INT ;
    Cosine_value_10ths_percent_INT : INT ;
END_VAR

CAL Cosine_approximation_114_till_228_rad(
    X_INT :=Input_114_till_228_rad,
    Suppress_extrapolation_BOOL :=1,
    X1_INT :=114,
    X2_INT :=120,
    X3_INT :=126,
    X4_INT :=132,
    X5_INT :=138,
    X6_INT :=144,
    X7_INT :=150,
    X8_INT :=156,
    X9_INT :=162,
    X10_INT :=168,
```

```
    X11_INT :=174,
    X12_INT :=180,
    X13_INT :=186,
    X14_INT :=192,
    X15_INT :=198,
    X16_INT :=204,
    X17_INT :=210,
    X18_INT :=216,
    X19_INT :=222,
    X20_INT :=228,
    Y1_INT :=914,
    Y2_INT :=866,
    Y3_INT :=809,
    Y4_INT :=743,
    Y5_INT :=669,
    Y6_INT :=588,
    Y7_INT :=500,
    Y8_INT :=407,
    Y9_INT :=309,
    Y10_INT :=208,
    Y11_INT :=105,
    Y12_INT :=0,
    Y13_INT :=-105,
    Y14_INT :=-208,
    Y15_INT :=-309,
    Y16_INT :=-407,
    Y17_INT :=-500,
    Y18_INT :=-588,
    Y19_INT :=-669,
    Y20_INT :=-743,
    Y_INT=>Cosine_value_10ths_percent_INT)

END_PROGRAM
```

U_Ip20_UINT_interpolation
Interpolation with 20 X/Y Interpolation Points and Unsigned Integer Values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
X_value_UINT	Known X value	0 to 65535
Parameters		
Suppress_ extrapolation_BOOL	For X values beyond the interpolation boundaries, the following can be chosen: 0 => Extrapolate 1 => Extrapolating is suppressed. The interpolation boundaries are output	0/1

Designation	Significance	Value range
X1_UINT	X value 1	0 to 65535
X2_UINT	X value 2	0 to 65535
X3_UINT	X value 3	0 to 65535
X4_UINT	X value 4	0 to 65535
X5_UINT	X value 5	0 to 65535
X6_UINT	X value 6	0 to 65535
X7_UINT	X value 7	0 to 65535
X8_UINT	X value 8	0 to 65535
X9_UINT	X value 9	0 to 65535
X10_UINT	X value 10	0 to 65535
X11_UINT	X value 11	0 to 65535
X12_UINT	X value 12	0 to 65535
X13_UINT	X value 13	0 to 65535
X14_UINT	X value 14	0 to 65535
X15_UINT	X value 15	0 to 65535
X16_UINT	X value 16	0 to 65535
X17_UINT	X value 17	0 to 65535
X18_UINT	X value 18	0 to 65535
X19_UINT	X value 19	0 to 65535
X20_UINT	X value 20	0 to 65535
Y1_UINT	Y value 1	0 to 65535
Y2_UINT	Y value 2	0 to 65535
Y3_UINT	Y value 3	0 to 65535
Y4_UINT	Y value 4	0 to 65535
Y5_UINT	Y value 5	0 to 65535
Y6_UINT	Y value 6	0 to 65535
Y7_UINT	Y value 7	0 to 65535
Y8_UINT	Y value 8	0 to 65535
Y9_UINT	Y value 9	0 to 65535

Designation	Significance	Value range
Y10_UINT	Y value 10	0 to 65535
Y11_UINT	Y value 11	0 to 65535
Y12_UINT	Y value 12	0 to 65535
Y13_UINT	Y value 13	0 to 65535
Y14_UINT	Y value 14	0 to 65535
Y15_UINT	Y value 15	0 to 65535
Y16_UINT	Y value 16	0 to 65535
Y17_UINT	Y value 17	0 to 65535
Y18_UINT	Y value 18	0 to 65535
Y19_UINT	Y value 19	0 to 65535
Y20_UINT	Y value 20	0 to 65535
Outputs		
Y_value_UINT	Interpolated (or extrapolated) Y value	0 to 65535

Description

For the X value at the input, a linearly interpolated Y value is calculated between the X/Y interpolation points. Beyond the X/Y interpolation points, a linearly extrapolated Y value is calculated for "Suppress_extrapolation_BOOL:=0", for "Suppress_extrapolation_BOOL:=1" the Y interpolation limiting values are entered (→ fig. 3).

→ The X values must be entered in ascending order.

Example:
The application program converted the following characteristic curve. The result approximates a cosine function from 0 to 114° (deg).

	1	2	3	4	5	6	7	8	9	10
X	0	6	12	18	24	30	36	42	48	54
Y	0	105	208	309	407	500	588	669	743	809

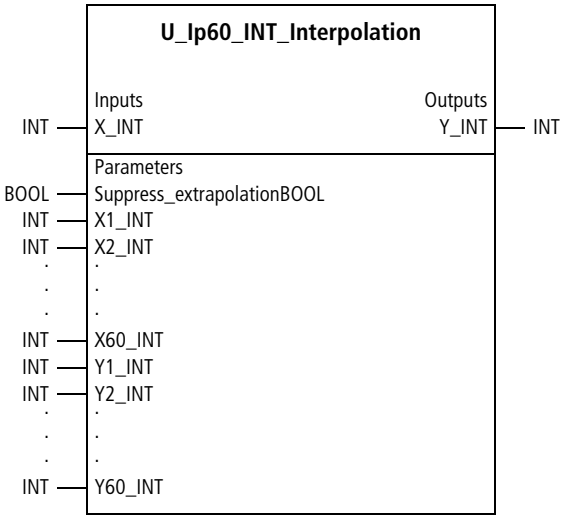
	11	12	13	14	15	16	17	18	19	20
X	60	66	72	78	84	90	96	102	108	114
Y	866	914	951	978	995	1000	995	978	951	914

Application of the function block "U_Ip20_UINT_interpolation" in the program "Cos_114"

```
PROGRAM Cos_114
VAR
    Cosine_approximation_0_till_114_rad : U_IP20_UINT_INTERPOLATION ;
    Input_0_till_114_rad : UINT ;
    Cosine_value_10ths_percent_UINT : UINT ;
END_VAR

CAL Cosine_approximation_0_till_114_rad(
    X_UINT :=Input_0_till_114_rad,
    Suppress_extrapolation_BOOL :=1,
    X1_UINT :=0,
    X2_UINT :=6,
    X3_UINT :=12,
    X4_UINT :=18,
    X5_UINT :=24,
    X6_UINT :=30,
    X7_UINT :=36,
    X8_UINT :=42,
    X9_UINT :=48,
    X10_UINT :=54,
```

```
    X11_UINT :=60,
    X12_UINT :=66,
    X13_UINT :=72,
    X14_UINT :=78,
    X15_UINT :=84,
    X16_UINT :=90,
    X17_UINT :=96,
    X18_UINT :=102,
    X19_UINT :=108,
    X20_UINT :=114,
    Y1_UINT :=0,
    Y2_UINT :=105,
    Y3_UINT :=208,
    Y4_UINT :=309,
    Y5_UINT :=407,
    Y6_UINT :=500,
    Y7_UINT :=588,
    Y8_UINT :=669,
    Y9_UINT :=743,
    Y10_UINT :=809,
    Y11_UINT :=866,
    Y12_UINT :=914,
    Y13_UINT :=951,
    Y14_UINT :=978,
    Y15_UINT :=995,
    Y16_UINT :=1000,
    Y17_UINT :=995,
    Y18_UINT :=978,
    Y19_UINT :=951,
    Y20_UINT :=914,
    Y_UINT=>Cosine_value_10ths_percent_UINT)

END_PROGRAM
```

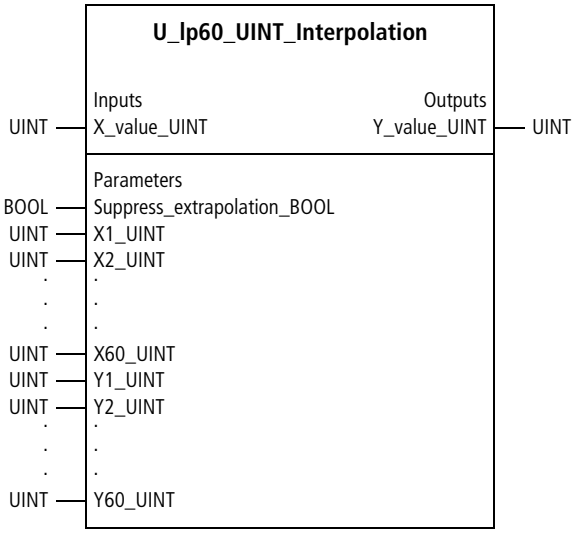
U_Ip60_INT_Interpolation
Interpolation mit 60 X/Y Interpolation Points and Integer Values



Function block prototype

→ The function block is identical to the function block "U_IP20_INT_Interpolation" apart from the number of X/Y interpolation points. This function block is described on page 102.

U_Ip60_UINT_Interpolation
Interpolation with 60 X/Y Interpolation Points and Unsigned Integer Values

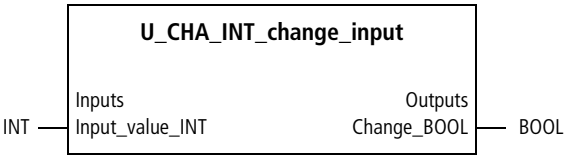


Function block prototype

→ The function block is identical to the function block "U_IP20_UINT_Interpolation" apart from the number of X/Y interpolation points. This function block is described on page 107.

Logic functions

U_CHA_INT_change_input
Changing the Integer Input Value



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_INT	It is determined if the value of this variable has changed	–32768 to 32767
Outputs		
Change_BOOL	This BOOL variable indicates if the input value has changed	0/1

Description

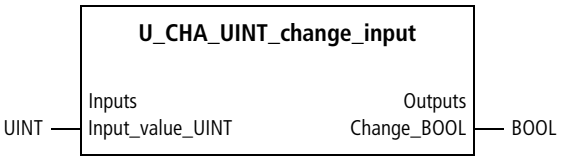
This function block indicates if the variable "Input_value_INT" has changed from the previous cycle to the current cycle. If so, the output "Change_BOOL" is a "1". If not, the output "Change_BOOL" is a "0".

→ If this function block is used several times, a new instance must be created each time because it saves data between the function block calls.

Application of the function block "U_CHA_INT_change_input" in the program "calcul_a"

```
PROGRAM calcul_a
VAR
        CHANGE_Parameter_a : U_CHA_INT_CHANGE_INPUT ;
        Parameter_a : INT ;
END_VAR

CAL CHANGE_Parameter_a(
        Input_value_INT :=Parameter_a
)
LD      CHANGE_Parameter_a.Change_BOOL
JMPCN   CALCULATION_NOT_NESECCARY
        (*
        Calculations or function block call,
        which depends on parameter "a" and need
        a lot of plc scan time.
        *)
CALCULATION_NOT_NESECCARY:

END_PROGRAM
```

U_CHA_UINT_change_input
Change of the Unsigned Integer Input Value



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_UINT	It is determined if the value of this variable has changed	0 to 65535
Outputs		
Change_BOOL	This BOOL variable indicates if the input value has changed	0/1

Description

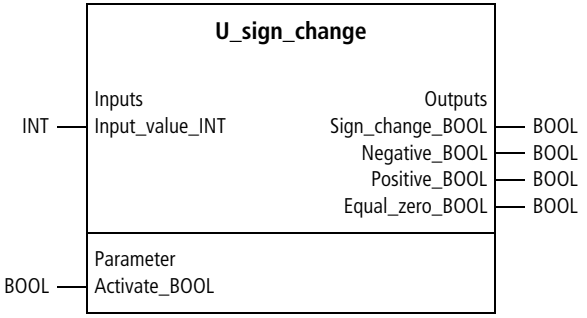
This function block indicates if the variable "Input_value_INT" has changed from the previous cycle to the current cycle. If so, the output "Change_BOOL" is a "1". If not, the output "Change_BOOL" is a "0".

→ If this function block is used several times, a new instance must be created each time because it saves data between the function block calls.

Application of the function block "U_CHA_UINT_change_input" in the program "calcul_b"

```
PROGRAM calcul_b
VAR
        CHANGE_Parameter_b : U_CHA_UINT_CHANGE_INPUT ;
        Parameter_b : UINT ;
END_VAR

CAL CHANGE_Parameter_b(
        Input_value_UINT :=Parameter_b
)
LD      CHANGE_Parameter_b.Change_BOOL
JMPCN   CALCULATION_NOT_NESECCARY
        (*
        Calculations or function block call,
        which depends on parameter "b" and need
        a lot of plc scan time
        *)
CALCULATION_NOT_NESECCARY:

END_PROGRAM
```

U_sign_change
Sign Change and Sign of an Integer Value



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_INT	The output status displays reflect this value. The value of the current cycle is compared with that of the previous cycle.	–32768 to 32767
Parameter		
Activate_BOOL	Activates the function block	0/1
Outputs		
Sign_change_BOOL	Message: the sign of the input value has changed from the previous cycle	0/1
Negative_BOOL	Message: input value is negative	0/1
Positive_BOOL	Message: input value is positive	0/1
Equal_zero_BOOL	Message: input value equals zero	0/1

Description

This function block indicates if the sign of the input value has changed from the previous cycle to the current cycle. It also indicates if the input value is negative, positive or zero.

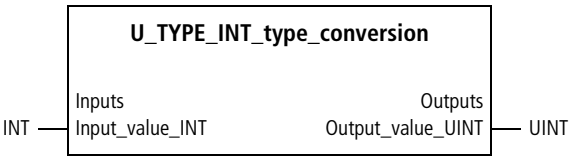
If this function block is used several times, a new instance must be created each time because it saves data between the function block calls.

Application of the function block "U_sign_change" in the program "Sign01"

```
PROGRAM Sign01
VAR
        SIGN_CHANGE01 : U_SIGN_CHANGE ;
        Manipulated_variable_bipolar_13Bit_INT : INT ;
END_VAR

CAL SIGN_CHANGE01(
        Input_value_INT :=Manipulated_variable_bipolar_13Bit_INT,
        Activate_BOOL :=1
)
LD      SIGN_CHANGE01.Sign_change_BOOL
JMPCN   NOT_THIS_SEQUENTIAL_PROGRAM
        (*
        Sequential program
        *)
NOT_THIS_SEQUENTIAL_PROGRAM:

END_PROGRAM
```

Other functions

U_TYPE_INT_type_conversion
Type Conversion from Data Type Integer to Data Type Unsigned Integer



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_INT	Integer input value to be converted to data type Unsigned Integer	–32768 to 32767
Outputs		
Output_value_UINT	Output value of data type Unsigned Integer	0 to 65535

Description

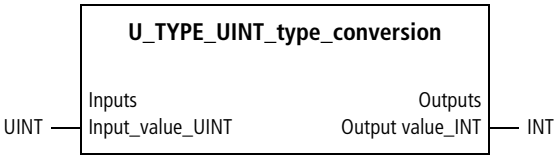
This function block converts an Integer value to an Unsigned Integer. In contrast to the manufacturer function "INT_TO_UINT", the value range limiting value is used if the value range overflows.

Example:
Variable_INT:=–1000
=> Type-converted variable_UINT:=0.

Application of the function block "U_TYPE_INT_type_conversion" in the program "INT_UINT"

```
PROGRAM INT_UINT
VAR
  TYPE_INT_TYPE_CONVERSION : U_TYPE_INT_TYPE_CONVERSION ;
  variable_INT : INT ;
  variable_UINT : UINT ;
END_VAR

CAL TYPE_INT_TYPE_CONVERSION(
  Input_value_INT :=variable_INT,
  Output_value_UINT=>variable_UINT
)

END_PROGRAM
```

U_TYPE_UINT_type_conversion
Type Conversion from Data Type Unsigned Integer to Data Type Integer



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_UINT	Unsigned Integer input value to be converted to data type Integer	0 to 65535
Outputs		
Output_value_INT	Output value of data type Integer	–32768 to 32767

Description

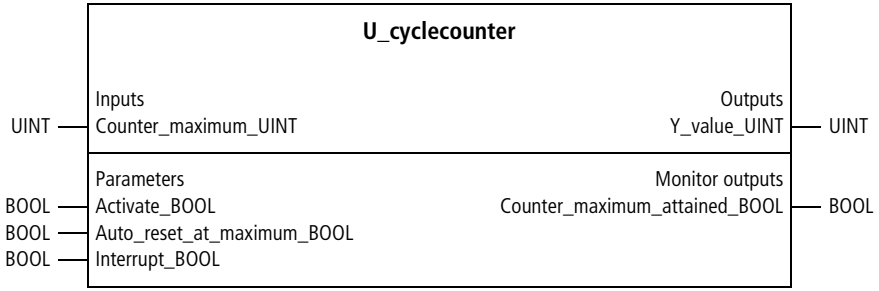
This function blocks converts an Unsigned Integer value to an Integer. In contrast to the manufacturer function "UINT_TO_INT", the value range limiting value is used if the value range overflows.

Example:
Variable_UINT:=60000
=> Type-converted variable_INT:=32767.

Application of the function block "U_TYPE_UINT_type_conversion" in the program "UINT_INT"

```
PROGRAM UINT_INT
VAR
  TYPE_UINT_TYPE_CONVERSION : U_TYPE_UINT_TYPE_CONVERSION ;
  variable_INT : INT ;
  variable_UINT : UINT :=60000;
END_VAR

CAL TYPE_UINT_TYPE_CONVERSION(
  Input_value_UINT :=variable_UINT
  Output_value_INT=>variable_INT
)

END_PROGRAM
```

U_cyclecounter
Counter



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Counter_maximum_UINT	Final counter value	0 to 65535
Parameters		
Activate_BOOL	Activates the counter. The counter begins at "1" (not at "0")	0/1
Auto_reset_at_maximum_BOOL	When the counter reaches the final value, "0" retains this value, while "1" resets the counter.	0/1
Interrupt_BOOL	Interrupts the counter	0/1
Outputs		
Counter_reading_UINT	Counter status	0 to 65535
Monitor outputs		
Counter_maximum_attained_ BOOL	Message: counter has reached the final value	0/1

Description

The function block "U_cyclecounter" can solve counter tasks such as the following. If "Activate_BOOL" is "1", then the counter begins to increment. If "Activate_BOOL" is "0", the counter is reset. The variable "Counter_maximum_ UINT" determines the final value for incrementing the counter. If "Auto_reset_at_maximum_BOOL" is "0", then the counter retains the final value when it is reached.

Example for counter maximum = 5:
=> 1; 2; 3; 4; 5; 5; 5; etc.

If "Auto_reset_at_maximum_BOOL" is "1", then the counter begins incrementation at "1".

Example for counter maximum = 5:
=> 1; 2; 3; 4; 5; 1; 2; 3; 4; 5; 1; etc.

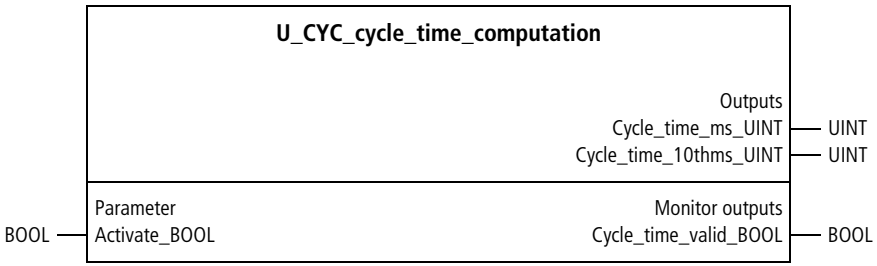
If "Interrupt_BOOL" is "1", then counting is interrupted. If "Interrupt_BOOL" is "0", then counting continues. "Counter_reading_UINT" displays the current counter status. When the counter reaches its final value, the output "Counter_maximum_attained_BOOL" outputs a "1".

Example:
The following application program produces the counter sequence:
1; 2; 3; 4; 5; 6; 7; 8; 9; 10; 1; 2; 3; etc.

Application of the function block "U_cyclecounter" in the program "scant_10"

```
PROGRAM scant_10
VAR
        CYCLECOUNTER01 : U_CYCLECOUNTER ;
END_VAR

CAL CYCLECOUNTER01(
        Counter_maximum_UINT :=10,
        Activate_BOOL :=1,
        Auto_reset_at_maximum_BOOL :=1,
        Interrupt_BOOL :=0
)
LD      CYCLECOUNTER01.Counter_maximum_attained_BOOL
JMPCN   NO_CALCULATION
        (*
        This sequential program is run through
        only every 10 plc scan time
        *)
NO_CALCULATION:

END_PROGRAM
```

U_CYC_cycle_time_computation
Computation of Average Cycle Time



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Parameter		
Activate_BOOL	Activates the function block	0 to 65535
Outputs		
Cycle_time_ms_UINT	Average cycle time in ms	0 to 6000
Cycle_time_10thms_UINT	Average cycle time in 10ths of a ms	0 to 60000
Monitor outputs		
Cycle_time_valid_BOOL	Display beginning at the first calculated cycle time (5 s after activation)	0/1

Description

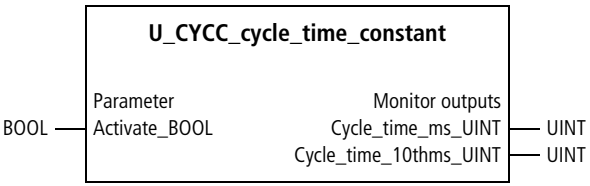
This function block calculates the average cycle time between individual function block calls. This cycle time equals the PLC cycle time if the function block is called each cycle. As soon as "Activate_BOOL" is "1", the first average cycle time is output after 5 s. The status display "Cycle_time_valid_BOOL" then displays a "1". More precise calculations are then made every 30 s.

Example:
The application program below calculates the cycle time and saves the result as the variable "Cycle time" for further processing. You are recommended to evaluate the cycle time whenever algorithms depend on the cycle time.

Application of the function block "U_CYC_cycle_time_computation" in the program "cycle_t"

```
PROGRAM cycle_t
VAR
  CYC_CYCLE_TIME_COMPUTATION : U_CYC_CYCLE_TIME_COMPUTATION ;
  Cycle_time_ms_UINT : UINT ;
END_VAR

CAL CYC_CYCLE_TIME_COMPUTATION(
  Activate_BOOL :=1,
  Cycle_time_ms_UINT=>Cycle_time_ms_UINT
)

END_PROGRAM
```

U_CYCC_cycle_time_constant
Automatic Setting of a Constant Cycle Time



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Parameter		
Activate_BOOL	Activates the function block	0/1
Monitor outputs		
Cycle_time_ms_UINT	Average cycle time in ms, refer to function block "U_CYC_cycle_time_computation"	
Cycle_time_10thms_UINT	Average cycle time in tenths of a ms, refer to function block "U_CYC_cycle_time_computation"	0 to 6000

Description

This function block sets constant PLC cycle times. It is called at the end of a main program. It sets a constant cycle time, which is around 1 ms greater than the maximum cycle time of the application program.

→ A certain amount of time may be required before the cycle times settle at a constant value. If RETAIN is added after the keyword "VAR" or "VAR_GLOBAL" and a blank, then after a warm start the specified constant cycle time is reapplied immediately.

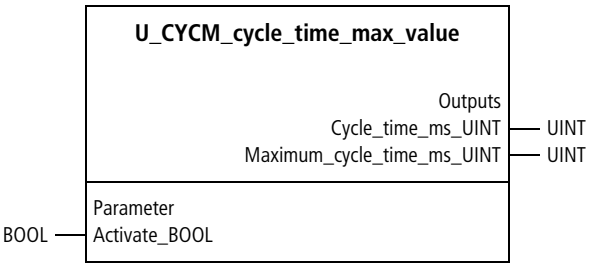
If the PLC operates periodically, this function block may not be used. It is not a requirement as automatic constant cycle times are present when the PLC operates periodically.

Example:
The application program below results in a constant cycle time of 20 ms. Without this function block, the cycle time would vary between 5.5 and 19.3 ms.

Application of the function block "U_CYCC_cycle_time_constant" in the program "constcyc"

```
PROGRAM constcyc
VAR
  CYCC_CYCLE_TIME_CONSTANT : U_CYCC_CYCLE_TIME_CONSTANT ;
END_VAR

(* The cal of other function blocks need a plc scan time from 5.5 ms to 19.3 ms. The
function block "U_CYCC_CYCLE_TIME_CONSTANT" effects, that the plc scan time is
constant 20 ms. *)

CAL CYCC_CYCLE_TIME_CONSTANT(
  Activate_BOOL :=1
)

END_PROGRAM
```

U_CYCM_cycle_time_max_value
Display of the Current and Maximum Cycle Time



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Parameter		
Activate_BOOL	Activates the function block	0 to 65535
Outputs		
Cycle_time_ms_UINT	Cycle time of the last PLC cycle in ms	0 to 65535
Maximum_cycle_time_ms_UINT	Maximum cycle time in ms	0 to 65535

Description

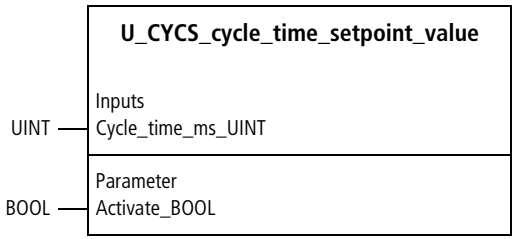
This function block displays the last PLC cycle time and the maximum PLC cycle time. "Activate_BOOL=1" starts the function block, and "Activate_BOOL=0" resets it.

Example:
The application program below displays the last and the maximum PLC cycle time.

Application of the function block "U_CYCM_cycle_time_max_value" in the program "MaxCycle"

```
PROGRAM MaxCycle
VAR
  CYCM_CYCLE_TIME_MAX_VALUE : U_CYCM_CYCLE_TIME_MAX_VALUE ;
END_VAR

CAL CYCM_CYCLE_TIME_MAX_VALUE(
  Activate_BOOL :=1
)

END_PROGRAM
```

U_CYCS_cycle_time_setpoint_value
Setting a (Constant) Cycle Time to Setpoint Value



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Cycle_time_ms_UINT	Desired cycle time	1 to 250
Parameter		
Activate_BOOL	Activates the function block	0/1

Description

This function block permits the input of a desired cycle time. This cycle time is set if the maximum actual cycle times of the application program are smaller than this value.

→ If the PLC cycle time exceeds the specified setpoint, nothing special happens (the PLC does not switch to "Halt"). This simply means the setpoint is not possible.

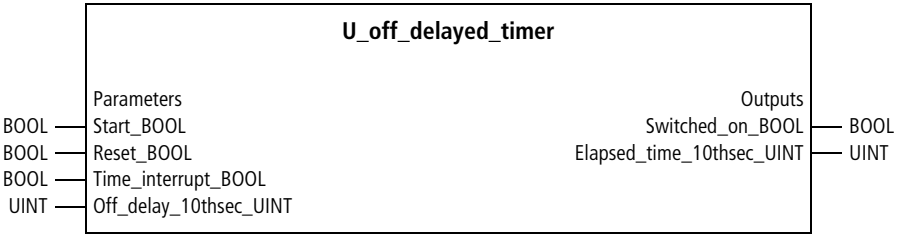
If the PLC operates periodically, this function block may not be used. It is not a requirement as automatic constant cycle times are present when the PLC operates periodically.

Example:
In the application program below, the instructions and function block calls generate an average cycle time of around 22 ms (4 ms). Assigning a setpoint of 30 ms changes the cycle times to this constant value.

Application of the function block "U_CYCS_cycle_time_setpoint_value" in the program "scantcon"

```
PROGRAM scantcon
VAR
  CYCS_CYCLE_TIME_SETPOINT_VALUE : U_CYCS_CYCLE_TIME_SETPOINT_VALUE ;
END_VAR

(* The cal of other function blocks generates a plc scan time from 24 ms to 28 ms. The
functionblock "U_CYCS_CYCLE_TIME_SETPOINT_VALUE" effects, that the plc scan time is
constant 30 ms *)

CAL CYCS_CYCLE_TIME_SETPOINT_VALUE(
  activate_BOOL :=1,
  cycle_time_ms_UINT :=30)

END_PROGRAM
```

U_off_delayed_timer
Delayed timer



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Parameters		
Start_BOOL	Start conditions raising edge	0/1
Reset_BOOL	Reset conditions	0/1
Time_interrupt_BOOL	Time interruption	0/1
Off_delay_10thsec_UINT	Time setpoint in tenths of a second	0 to 65535
Outputs		
Switched_on_BOOL	Control output signal	0/1
Elapsed_time_10thsec_UINT	Actual time in tenths of a second	0 to 65535

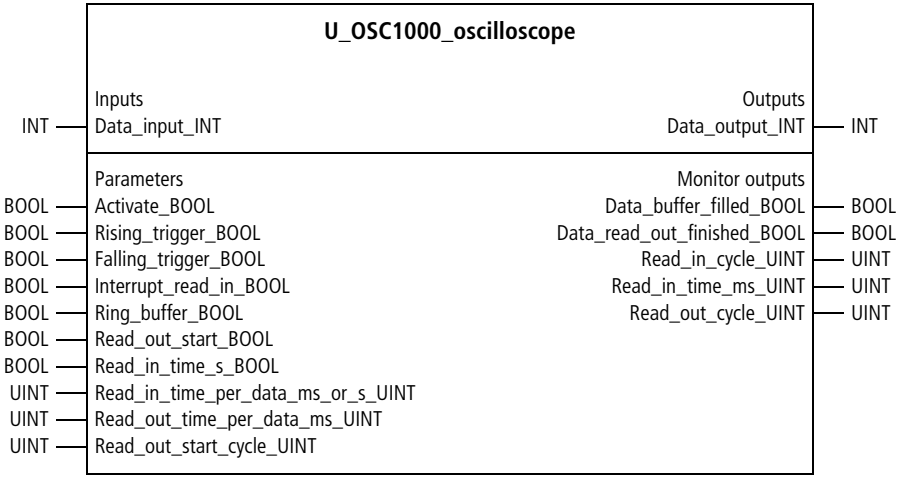
Description

Except for the times, this function block works just like the Moeller add-on function block "MS_TimeFalling". In contrast to the latter, the times "Off_delay_10thsec_UINT" and "Elapsed_time_10thsec_UINT" are specified in tenths of a second. The variable names are assigned as follows:

U_off_delayed_timer	MS_TimeFalling
Start_BOOL	Set
Reset_BOOL	ReSet
Time_interrupt_BOOL	Hold
Off_delay_10thsec_UINT	PresetTime
Switched_on_BOOL	OutputControl
Elapsed_time_10thsec_UINT	ElapsedTime

Visualisation

U_OSC1000_oscilloscope
Data buffer function block for saving 1000 data values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Data_input_INT	Data entered in quick succession	–32768 to 32767
Parameters		
Activate_BOOL	Activates the function block	0/1
Rising_trigger_BOOL	If this parameter has a rising edge, the read process is restarted	0/1
Falling_trigger_BOOL	If this parameter has a falling edge, the read process is restarted	0/1
Interrupt_read_in_BOOL	Interruption of the read process	0/1
Ring_buffer_BOOL	Mode: recursive data read-in (ring buffer)	0/1
Read_out_start_BOOL	Start of read-out process	0/1
Read_in_time_s_BOOL	Read-in time in ms (0) or s (1)	0/1
Read_in_time_per_data_ms_ or_s_UINT	Read-in time per data value in ms or s	0 to 65535
Read_out_time_per_data_ ms_UINT	Read-out time per data value in ms	0 to 65535
Read_out_start_cycle_UINT	Cycle, where read-out should start	0 to 1000
Outputs		
Data_output_INT	Data output slowly (in slow-motion)	–32768 to 32767
Monitor outputs		
Data_buffer_filled_BOOL	Status: the data buffer is full	0/1
Data_read_out_finished_BOOL	Status: the data buffer has been read out	0/1
Read_in_cycle_UINT	Read-in cycle	0 to 1000
Read_in_time_ms_UINT	Duration of read-in process in ms	0 to 65535
Read_out_cycle_UINT	Read-out cycle	0 to 1000

Description

1000 data values can be read in quickly (or slowly) with the function block, e. g. one value in every PLC-cycle. These values can be output at a slowed speed (slow-motion) later.

A fast-acting process, e. g. a highly-dynamic positioning operation, can be fully graphically represented with a relatively slow-action visualization tool. In this way, the function of an oscilloscope can be substituted.

After this function block has been activated, reading of the values can be initiated as required, on either the falling or rising edges. In the "Ring-buffer" mode, the first values are again overwritten after 1000 values have been read (Ring buffer). The read out time per value can be input in ms.

The monitor inputs indicate when the data buffer is full and when the data have been completely read out. Additionally, the "Read-in cycle", "Read-in time" and "Read-out cycle" information is available.

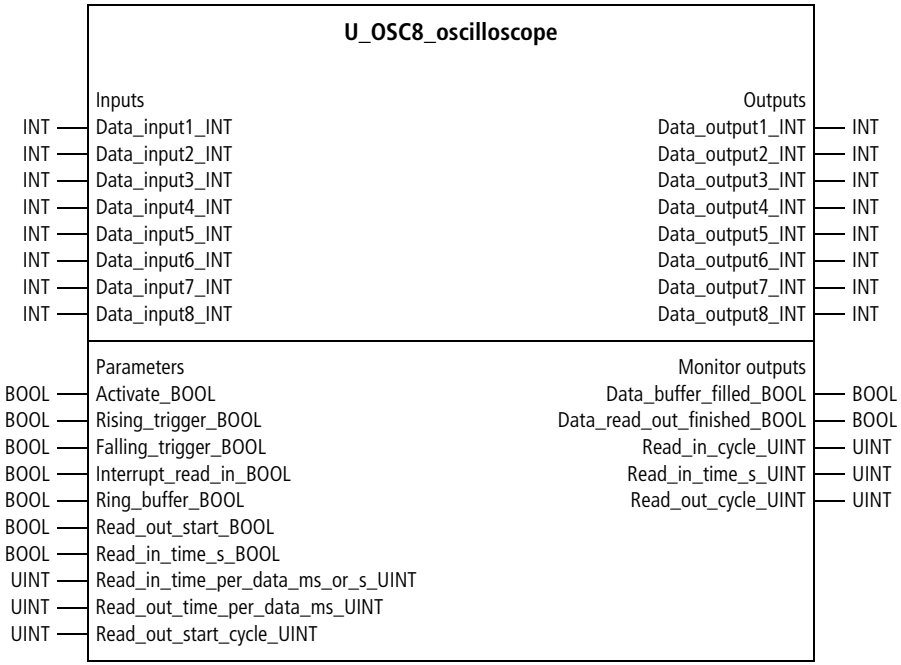
Example:

In the application example, the characteristic of the manipulated variable is recorded at the start of a positioning operation and output later with a time of 100 ms per value.

Application of the function block "U_OSC1000_oscilloscope" in the program "Oscillo"

```
PROGRAM Oscillo

VAR
    OSC1000_Oscilloscope : U_OSC1000_Oscilloscope ;
    Axis_01 : P_closed_loop_position_control ;
    Visualisation_value_INT : INT ;
END_VAR

CAL OSC1000_Oscilloscope(
    Data_input_INT :=Axis_01.Manipulated_variable_12Bit_INT,
    Activate_BOOL :=1,
    Rising_trigger_BOOL :=0,
    Falling_trigger_BOOL :=Axis_01.Positioning_job_finished_BOOL,
    Interrupt_read_in_BOOL :=0,
    Ring_buffer_BOOL :=0,
    Read_out_start_BOOL :=1,
    Read_in_time_s_BOOL :=0,
    Read_in_time_per_data_ms_or_s_UINT :=0,
    Read_out_time_per_data_ms_UINT :=100,
    Read_out_start_cycle_UINT :=0
    )

LD  OSC1000_Oscilloscope.Data_output_INT
ST  Visualisation_value_INT

END_PROGRAM
```

U_OSC8_oscilloscope
Data buffer function block for saving 8 × 1000 data values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Data_input1_INT	1. Data which is entered quickly	–32768 to 32767
Data_input2_INT	2. Data whichis entered quickly	–32768 to 32767
Data_input3_INT	3. Data which is entered quickly	–32768 to 32767
Data_input4_INT	4. Data which is entered quickly	–32768 to 32767
Data_input5_INT	5. Data whichis entered quickly	–32768 to 32767
Data_input6_INT	6. Data which is entered quickly	–32768 to 32767
Data_input7_INT	7. Data which is entered quickly	–32768 to 32767
Data_input8_INT	8. Data which is entered quickly	–32768 to 32767
Parameters		
Activate_BOOL	Activates the function block	0/1
Rising_trigger_BOOL	If this parameter has a rising edge, the read process is restarted	0/1
Falling_trigger_BOOL	If this parameter has a falling edge, the read process is restarted	0/1
Interrupt_read_in_BOOL	Interruption of the read process	0/1
Ring_buffer_BOOL	Mode: recursive data read-in (ring buffer)	0/1
Read_out_start_BOOL	Start of read-out process	0/1
Read_in_time_s_BOOL	Read-in time in ms (0) or s (1)	0/1
Read_in_time_per_ data_ms_or_s_UINT	Read-in time per data in ms or s	0 to 65535
Read_out_time_per_ data_ms_UINT	Read-out time per data value in ms	0 to 65535
Read_out_start_ cycle_UINT	Cycle where read-out should start	0 to 1000

Designation	Significance	Value range
Outputs		
Data_output1_INT	1. Data which is output slowly (slowmotion)	–32768 to 32767
Data_output2_INT	2. Data which is output slowly (slowmotion)	–32768 to 32767
Data_output3_INT	3. Data which is output slowly (slowmotion)	–32768 to 32767
Data_output4_INT	4. Data which is output slowly (slowmotion)	–32768 to 32767
Data_output5_INT	5. Data which is output slowly (slowmotion)	–32768 to 32767
Data_output6_INT	6. Data which is output slowly (slowmotion)	–32768 to 32767
Data_output7_INT	7. Data which is output slowly (slowmotion)	–32768 to 32767
Data_output8_INT	8. Data which is output slowly (slowmotion)	–32768 to 32767
Monitor outputs		
Data_buffer_filled_BOOL	Status: the data buffer is full	0/1
Data_read_out_ finished_BOOL	Status: the data buffer has been read out	0/1
Read_in_cycle_UINT	Read-in cycle	0 to 1000
Read_in_time_ms_UINT	Duration of read-in process in ms	0 to 65535
Read_out_cycle_UINT	Read-out cycle	0 to 1000

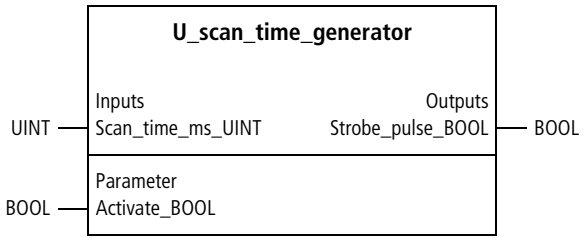
Description

See "U_OSZ1000_oscilloscope" function block. The only difference to the "U_OSZ1000_oscilloscope" function block is that 8 input values of 1000 data values each can be stored with this function block.

3 Basic Function Blocks for Closed-loop Control

Elementary basic function blocks for closed-loop control

U_scan_time_generator
Scan Time Generator



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Scan_time_ms_UINT	scan time	0 to 65535
Parameter		
Activate_BOOL	Activates the function block	0/1
Outputs		
Strobe_pulse_BOOL	strobe pulse	0/1

Description

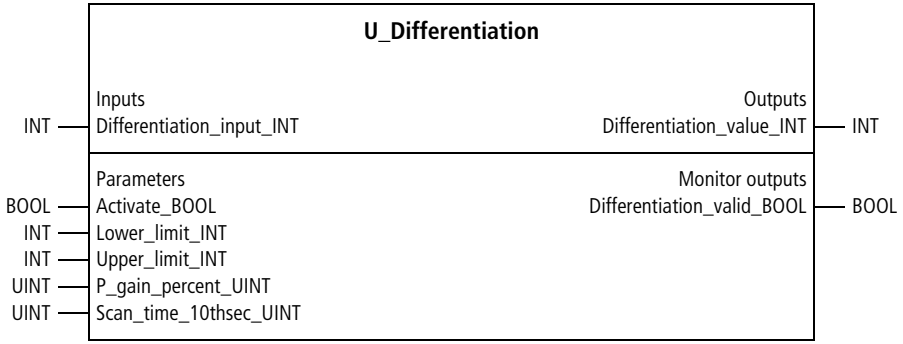
After a scan time (which can be specified), this function block generates a pulse. "Activate_BOOL=1" starts the function block (no pulse is generated when it starts). "Activate_BOOL=0" resets the function block.

Example:
The function block is used, for example, when program sequences are executed not always but only at regular intervals. The application example below includes a sequence which executes only every 20 s.

Application of the function block "U_scan_time_generator" in the program "scant20s"

```
PROGRAM scant20s
VAR
        SCAN_TIME_GENERATOR : U_SCAN_TIME_GENERATOR ;
END_VAR

CAL SCAN_TIME_GENERATOR(
        Activate_BOOL :=1,
        Scan_time_ms_UINT :=20000
)
LD      SCAN_TIME_GENERATOR.Strobe_pulse_BOOL
JMPCN   END_OF_SEQUENTIAL_PROGRAM
        (*Any program sequence*)
END_OF_SEQUENTIAL_PROGRAM:

END_PROGRAM
```

U_Differentiation
Differentiation of an Input Value



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Differentiation_input_INT	Value to be differentiated	–32768 to 32767
Parameters		
Activate_BOOL	Activates differentiation	0/1
Lower_limit_INT	Lower limit of the differentiation value at the output	–32768 to 32767
Upper_limit_INT	Upper limit of the differentiation value at the output	–32768 to 32767
P_gain_percent_UINT	Proportional gain	0 to 65535
Scan_time_10thsec_UINT	Scan time	0 to 65535
Outputs		
Differentiation_value_ INT	Differentiated value	–32768 to 32767
Monitor outputs		
Differentiation_valid_BOOL	Message: Differentiation has calculated an initial value.	0/1

Description

At regular intervals (which can be specified by "Scan_time_10thsec_UINT"), the current input value is compared with an earlier input value (→ fig. 5). The difference between the current and earlier input value (ΔX) is factored as "P_gain_percent_UINT". The function block is enabled with "Activate_BOOL=1". After an initial scan time, a first differentiation value can be calculated. The latter is displayed with "1=differentiation_valid_BOOL". "Activate_BOOL=0" resets the function block. With the variables "Lower_limit_INT" and "Upper_limit_INT", the output value "Differentiation_value_ INT" can be limited to a specific range.

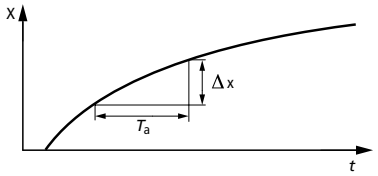


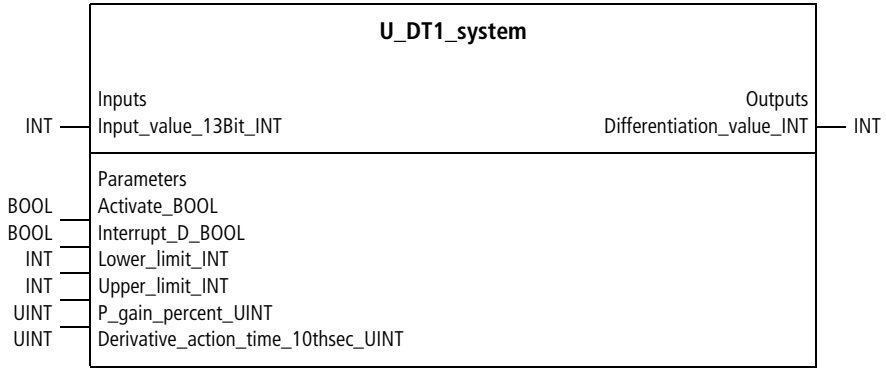
Figure 5: Relationship between the scan time T_a and the "differentiation" value ΔX

Example:
The application example calculates the increase in an actual value for a period of 10 s. The increase values are not factored (P gain = 100 = 1.0). The differentiation values are limited to the range –4095 to 4095.

Application of the function block "U_Differentiation" in the program "Diff01"

```
PROGRAM Diff01
VAR
        DIFFERENTIATION : U_DIFFERENTIATION ;
        Actual_value : UINT ;
        Actual_value_differentiation_10s : INT ;
END_VAR

CAL DIFFERENTIATION(
        Differentiation_input_UINT :=Actual_value,
        Activate_BOOL :=1,
        Lower_limit_INT :=-4095,
        Upper_limit_INT :=4095,
        P_gain_percent_UINT :=100,
        Scan_time_10thsec_UINT :=100
)
LD      DIFFERENTIATION.Differentiation_valid_BOOL
JMPCN   NOT_READY
        LD  DIFFERENTIATION.Differentiation_value_INT
        ST  Actual_value_differentiation_10s
NOT_READY:

END_PROGRAM
```

U_DT1_system
DT1 System



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_13Bit_INT	Input value which is DT1-delayed	–4095 to 4095
Parameters		
Activate_BOOL	Activates the DT1 delay	0/1
Interrupt_D_BOOL	Interrupts the internal scan algorithms and immediately calculates a DT1-delayed value	0/1
Lower_limit_INT	Lower limit of the differentiation value at the output	–32768 to 32767
Upper_limit_INT	Upper limit of the differentiation value at the output	–32768 to 32767
P_gain_percent_UINT	Proportional gain	0 to 65535
Derivative_action_time_ 10thsec_UINT	Derivative action time	0 to 65535
Outputs		
Differentiation_value_INT	Differentiation value or DT1-delayed value	–32768 to 32767

Description

At regular intervals (which can be specified by "Derivative_action_time_10thsec_UINT"), the current input value is compared with an earlier input value (→ fig. 5, derivative action time = scan time). The difference between the current and earlier input values (ΔX) is factored as "P_gain_percent_UINT". The function block is enabled with "Activate_BOOL=1". "Activate_BOOL=0" resets the function block. With the variables "Lower_limit_INT" and "Upper_limit_INT", the output value "Differentiation_value_ INT" can be limited to a specific range. The rising edge of the variable "Interrupt_D_BOOL" interrupts the function block's internal algorithms. This way, a DT1-delayed value can be output without hesitation, e.g. in case of an instantaneous change in the value of a setpoint.

→ In contrast to the function block "U_Differentiation", this function block PT1-delays the calculated rates of increase. The advantage of this is that if the input value changes instantaneously (e.g. an instantaneous change in a setpoint => infinitely large increase, because Δt = 0), the corresponding increase can be processed "for real" (→ real PID controller = PIDT1 controller; → fig. 6, cf. block diagram D gate/DT1 gate). The disadvantage of the function block "U_DT1_system" is that, due to the PT1 delay, the exact increase can only be calculated gradually (→ fig. 6, PT1 element).

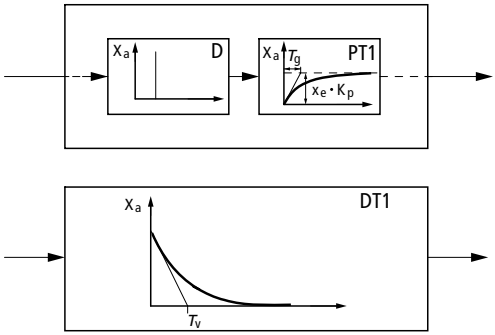


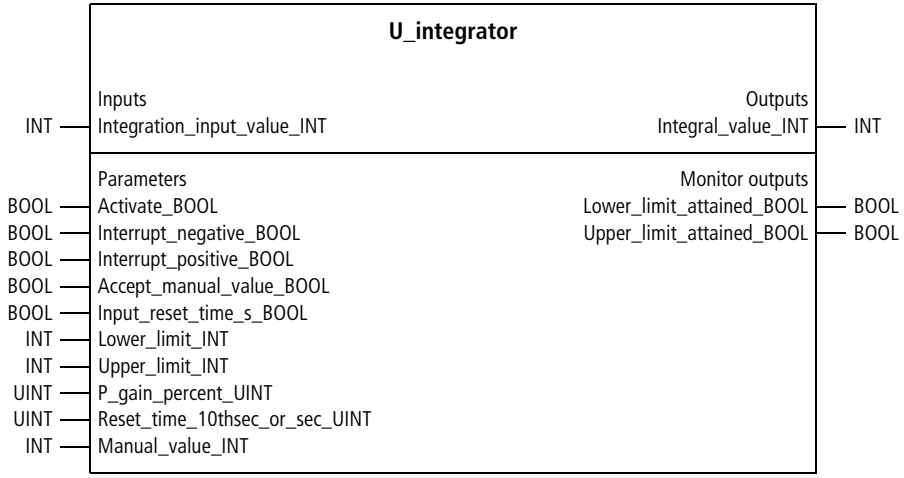
Figure 6: Transition function of the DT1 system. The DT1 behaviour results from the series connection of a D gate and a PT1 gate.

Example:
The application example calculates the increase of an actual value. The derivative action time was set to 10 s. The rates of increase are not factored (P gain = 100 = 1.0). The DT1-delayed values are limited to the range –4095 to 4095.

Application of the function block "U_DT1_system" in the program "Diff02"

```
PROGRAM Diff02
VAR
    DT1_SYSTEM : U_DT1_SYSTEM ;
    Actual_value : INT ;
    Actual_value_differentiation_10s : INT ;
END_VAR

CAL DT1_SYSTEM(
    Input_value_13Bit_INT :=Actual_value,
    Activate_BOOL :=1,
    Interrupt_D_BOOL :=0,
    Lower_limit_INT :=-4095,
    Upper_limit_INT :=4095,
    P_gain_percent_UINT :=100,
    Derivative_action_time_10thsec_UINT :=100,
    Derivative_value_INT=>Actual_value_differentiation_10s)

END_PROGRAM
```

U_integrator
Integrator



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Integration_input_value_INT	Input value which is integrated	–32768 to 32767
Parameters		
Activate_BOOL	Activates integration	0/1
Interrupt_negative_BOOL	Interrupts integration in negative direction	0/1
Interrupt_positive_BOOL	Interrupts integration in positive direction	0/1

Designation	Significance	Value range
Accept_manual_value_BOOL	Accepts a manual value	0/1
Input_reset_time_s_BOOL	Choice: the reset time can be entered in seconds or milliseconds	0/1
Lower_limit_INT	Lower limit of the integral value at the output	–32768 to 32767
Upper_limit_INT	Upper limit of the integral value at the output	–32768 to 32767
P_gain_percent_UINT	Proportional gain	0 to 65535
Reset_time_10thsec_or_sec_UINT	Reset time	0 to 65535
Manual_value_INT	Manual value	–32768 to 32767
Outputs		
Integral_value_INT	Integral value	–32768 to 32767
Monitor outputs		
Lower_limit_attained_BOOL	The integral value has reached the specified lower limit	0/1
Upper_limit_attained_BOOL	The integral value has reached the specified upper limit	0/1

Description

The integrator is started with "Activate_BOOL=1" and reset with "Activate_BOOL=0". With "Interrupt_negative_ BOOL=1" and "Interrupt_positive_BOOL=1", the integration can be stopped in the corresponding direction (→ fig. 7). With "Accept_manual_value_BOOL=1", the value specified by "Manual_value_INT" is accepted as an integral value. The reset time can be specified, in tenths of a second with "Input_reset_time_s_BOOL=0" and in seconds with "Input_reset_time_s_BOOL=1". With the variables "Lower_limit_INT" and "Upper_limit_INT", the output value "Integral_value_ INT" can be limited to a value range. When the specified limits of integration are reached, the variables "Lower_limit_attained_BOOL" and "Upper_limit_attained_BOOL" are assigned the value "1". During reset time, the integrator counts up by the value at the input, multiplied by the P gain (→ fig. 7).

Example:
Integration_input_value_INT = 100
P_gain_percent_UINT = 100 (= 1.0)
Reset_time_10thsec_or_sec_UINT = 200 (= 20 s)
=> "Integral_value_INT" integrated up in 20 s
by 100 increments (100 = 100 × 1,0).

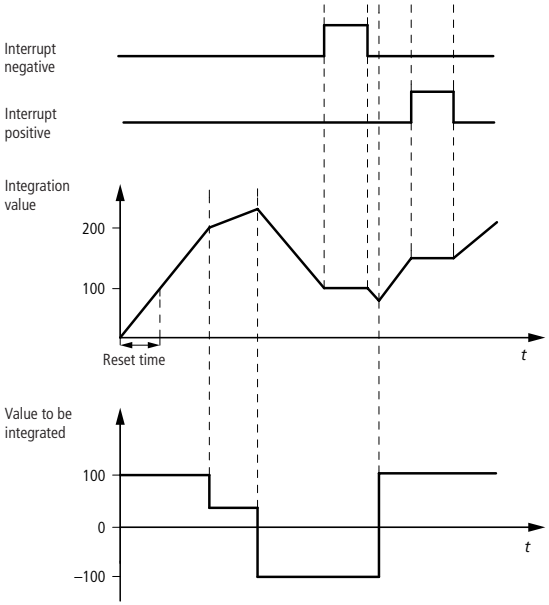


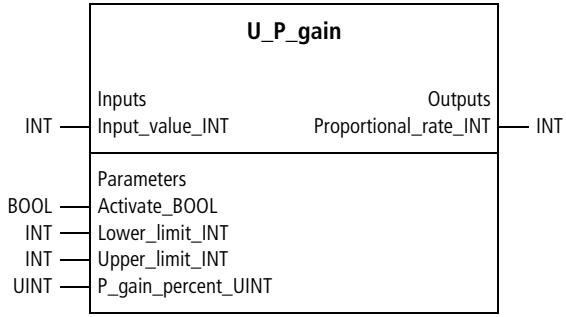
Figure 7: Functioning of the integrator in terms of the "Integration input value" and the command bit "Interrupt positive/negative". The proportional gain is 1.0 (neutral).

Example:
The application example integrates the output of a fuzzy system up, from 0 to 100. The reset time is 100 s.

Application of the function block "U_integrator" in the program "FuzzyInt"

```
PROGRAM FuzzyInt
VAR
         INTEGRATOR_a : U_INTEGRATOR ;
         Fuzzy : U_FUZ_22_FUZZY_2I_2T ;
         Integrated_fuzzy_value_INT : INT ;
END_VAR

CAL      Fuzzy
LD       Fuzzy.Output_variable_INT
ST       INTEGRATOR_a.Integration_input_value_INT
CAL INTEGRATOR_a(
         Activate_BOOL :=1,
         Interrupt_negative_BOOL :=0,
         Interrupt_positive_BOOL :=0,
         Accept_manual_value_BOOL :=0,
         Input_reset_time_sec_BOOL :=1,
         Lower_limit_INT :=0,
         Upper_limit_INT :=100,
         P_gain_percent_UINT :=100,
         Reset_time_10thsec_or_sec_UINT :=100,
         Manual_value_INT :=0
)
LD       INTEGRATOR_a.Integral_value_INT
ST       Integrated_fuzzy_value_INT

END_PROGRAM
```

U_P_gain
P Gain



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_INT	Input value for P gain	–32768 to 32767
Parameters		
Activate_BOOL	Activates P gain	0/1
Lower_limit_INT	Lower limit of the proportional rate	–32768 to 32767
Upper_limit_INT	Upper limit of the proportional rate	–32768 to 32767
P_gain_percent_UINT	P gain	0 to 65535
Outputs		
Proportional_rate_INT	Proportional rate	–32768 to 32767

Description

This function block generates a proportional gain of the input signal. The proportional rate can be limited to a value range, with the variables "Lower_limit_INT" and "Upper_limit_INT". When this function block is disabled (like disabling a P controller), the proportional rate Zero is output.

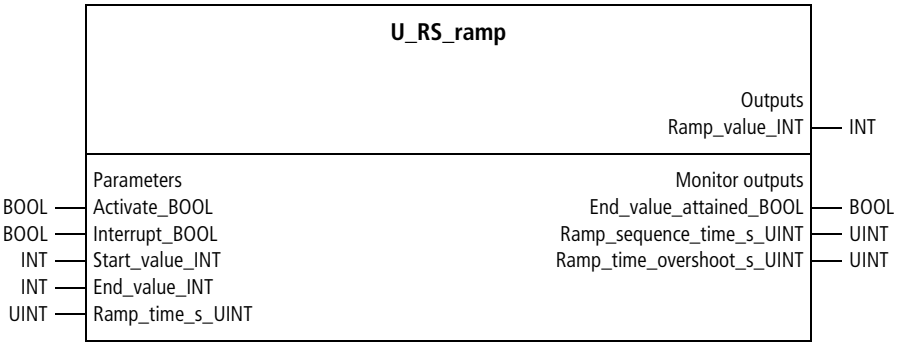
Example:
The application example implements a P controller with this function block. The proportional rate was limited to 13 bits (–4095 to 4095).

Application of the function block "U_P_gain" in the program "P_Control"

```
PROGRAM P_Control
VAR
    P_GAIN : U_P_GAIN ;
    System_deviation_INT : INT ;
    Proportional_rate_percent_UINT : UINT ;
    Manipulated_variable_P_13Bit_INT : INT ;
END_VAR

CAL P_GAIN(
    Input_value_INT :=System_deviation_INT,
    Activate_BOOL :=1,
    Lower_limit_INT :=-4095,
    Upper_limit_INT :=4095,
    P_gain_percent_UINT :=Proportional_rate_percent_UINT,
    Proportional_rate_INT=>Manipulated_variable_P_13Bit_INT
)

END_PROGRAM
```

Ramp function

U_RS_ramp
Ramp with Input in Seconds



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Parameters		
Activate_BOOL	Activates generation of the ramp	0/1
Interrupt_BOOL	Interrupts generation of the ramp	0/1
Start_value_INT	Ramp start value	–32768 to 32767
End_value_INT	Ramp end value	–32768 to 32767
Ramp_time_s_UINT	Ramp time from start value to end value	0 to 65535
Outputs		
Ramp_value_INT	Ramp value	–32768 to 32767
Monitor outputs		
End_value_attained_BOOL	Message: ramp end value reached	0/1
Ramp_sequence_time_s_UINT	Current ramp time	0 to 65535
Ramp_time_overshoot_s_UINT	Time by which the specified ramp time was exceeded	0 to 65535

Description

When this function block is activated (by the rising edge of "Activate_BOOL"), it generates a ramp. Within a period of time (which can be specified in seconds), it progresses from a start value to an end value (→ fig. 8). During the rising edge of "Activate_BOOL", the parameters are accepted. "Interrupt_BOOL=1" interrupts the progress of the ramp. A BOOL variable reports when the ramp end value has been reached (→ monitor output). The variable "Ramp_sequence_time_s_UINT" specifies how much of the ramp time has already expired. The variable "Ramp_time_overshoot_s_UINT" indicates whenever the specified ramp time is exceeded (at most one PLC cycle).

→ The ramp time can be entered in the unit of seconds. Thus, the maximum value is:

- 65535 s = 1092 min = 18.20 h

The function block "U_RS_ramp" generates the ramp value only in steps of seconds. If the ramp time is not longer than 65.535 s, you are recommended to use the function block "U_RMS_ramp", which generates the ramp value in millisecond steps.

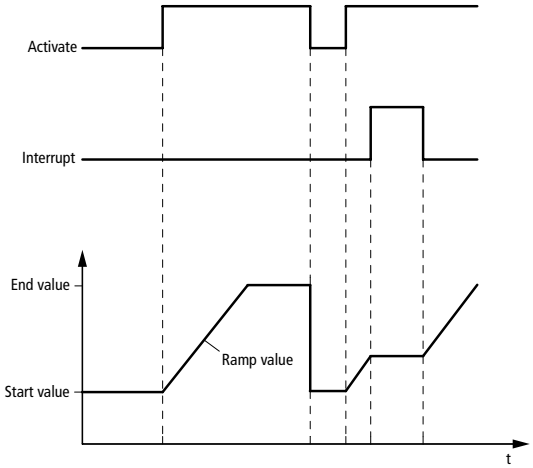


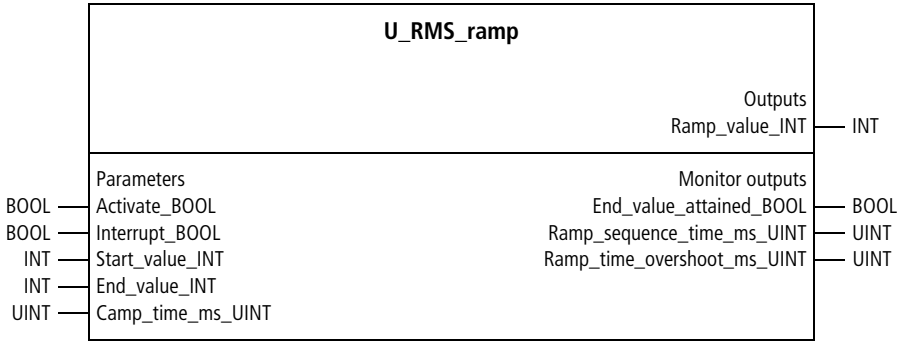
Figure 8: Generating ramps in conjunction with the command bits "Activate" and "Interrupt"

Example:
The application example combines two ramp curves. During the first phase, the ramp value increases from 1000 to 2000 within 10 minutes. Then, it decreases within 5 minutes from 2000 to 500. The application example for the next function block, "U_RMS_ramp" will illustrate how to re parameterise a ramp instance, to implement a ramp curve with varying ramp slopes.

Application of the function block "U_RS_ramp" in the program "Ramp_1_2"

```
PROGRAM Ramp_1_2
VAR
        RAMP01 : U_RS_RAMP ;
        RAMP02 : U_RS_RAMP ;
        Ramp01_activate_BOOL : BOOL ;
        Ramp_value_01_02_INT : INT ;
END_VAR

CAL RAMP01(
        Activate_BOOL :=Ramp01_activate_BOOL,
        Interrupt_BOOL :=0,
        Start_value_INT :=1000,
        End_value_INT :=2000,
        Ramp_time_s_UINT :=600)
CAL RAMP02(
        Activate_BOOL :=RAMP01.End_value_attained_BOOL,
        Interrupt_BOOL :=0,
        Start_value_INT :=2000,
        End_value_INT :=500,
        Ramp_time_s_UINT :=300)
LD      RAMP01.End_value_attained_BOOL
JMPCN   RAMP_VALUE_02
        LD  RAMP01.Ramp_value_INT
        ST  Ramp_value_01_02_INT
        JMP END_RAMP_VALUE_02
RAMP_VALUE_02:
        LD  RAMP02.Ramp_value_INT
        ST  Ramp_value_01_02_INT
END_RAMP_VALUE_02:

END_PROGRAM
```

U_RMS_ramp
Ramp with Input in Milliseconds



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Parameters		
Activate_BOOL	Activates generation of the ramp	0/1
Interrupt_BOOL	Interrupts generation of the ramp	0/1
Start_value_INT	Ramp start value	–32768 to 32767
End_value_INT	Ramp end value	–32768 to 32767
Ramp_time_ms_UINT	Ramp time from start value to end value	0 to 65535
Outputs		
Ramp_value_INT	Ramp value	–32768 to 32767
Monitor outputs		
End_value_attained_BOOL	Message: ramp end value reached	0/1
Ramp_sequence_time_ms_UINT	Current ramp time	0 to 65535
Ramp_time_overshoot_ms_UINT	Time by which the specified ramp time was exceeded	0 to 65535

Description

When this function block is activated (by the rising edge of "Activate_BOOL"), it generates a ramp. Within a period of time (which can be specified in milliseconds), it progresses from a start value to an end value (→ fig. 8). During the rising edge of "Activate_BOOL", the parameters are accepted. "Interrupt_BOOL=1" interrupts the progress of the ramp. A BOOL variable reports when the ramp end value has been reached (→ monitor output). The variable "Ramp_sequence_time_ms_UINT" specifies how much of the ramp time has already expired. The variable "Ramp_time_overshoot_ms_UINT" indicates whenever the specified ramp time is exceeded (at most one PLC cycle).

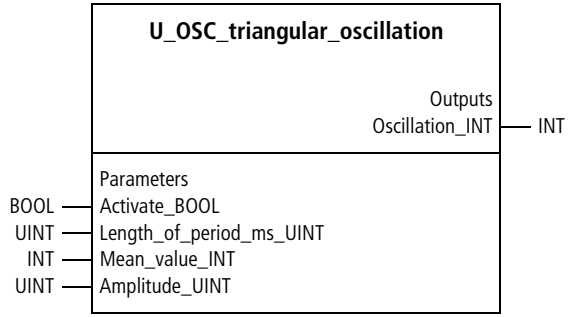
Example:
The application example generates a ramp curve with two areas having varying slopes. During the first phase, the ramp progresses from 500 to 4000 within 20 s. During the second phase, it runs from 4000 to 6000 within 5 s.

Application of the function block "U_RMS_ramp" in the program "Ramps"

```
PROGRAM Ramps
VAR
        RMS_RAMP : U_RMS_RAMP ;
        Rising_TRIG_RESET : R_Trig ;
        Rising_TRIG_END_VALUE_RAMP1 : R_Trig ;
        Reset_BOOL at %I0.0.0.0.0 : BOOL ;
        Initialize_Ramp1_BOOL : BOOL ;
        Initialize_Ramp2_BOOL : BOOL ;
        End_value_ramp1_attained_BOOL : BOOL ;
        Ramp_value_01_02_INT : INT ;
END_VAR

CAL Rising_TRIG_RESET(
        CLK :=Reset_BOOL)
LD      Rising_TRIG_RESET.Q
JMPCN   NO_RESET
        LD  1
        ST  Initialize_Ramp1_BOOL
        STN End_value_Ramp1_attained_BOOL
NO_RESET:

LD      Initialize_Ramp1_BOOL
JMPCN   END_INITIALIZATION_RAMP1
        LD  0
        ST  RMS_RAMP.Activate_BOOL
        CAL RMS_RAMP
        LD  500
        ST  RMS_RAMP.Start_value_INT
        LD  4000
        ST  RMS_RAMP.End_value_INT
        LD  20000
        ST  RMS_RAMP.Ramp_time_ms_UINT
        LD  1
```

```
        ST  RMS_RAMP.Activate_BOOL
        LD  0
        ST  Initialize_Ramp1_BOOL
END_INITIALIZATION_RAMP1:

LD      Initialize_Ramp2_BOOL
JMPCN   END_INITIALIZATION_RAMP2
        LD  0
        ST  RMS_RAMP.Activate_BOOL
        CAL RMS_RAMP
        LD  4000
        ST  RMS_RAMP.Start_value_INT
        LD  6000
        ST  RMS_RAMP.End_value_INT
        LD  5000
        ST  RMS_RAMP.Ramp_time_ms_UINT
        LD  1
        ST  RMS_RAMP.Activate_BOOL
        LD  0
        ST  Initialize_Ramp2_BOOL
END_INITIALIZATION_RAMP2:

CAL     RMS_RAMP

LD      RMS_RAMP.End_value_attained_BOOL
ST      End_value_ramp1_attained_BOOL
CAL     Rising_TRIG_END_VALUE_RAMP1(
        CLK :=End_value_ramp1_attained_BOOL)
LD      Rising_TRIG_END_VALUE_RAMP1.Q
ST      Initialize_Ramp2_BOOL

LD      RMS_RAMP.Ramp_value_INT
ST      Ramp_value_01_02_INT

END_PROGRAM
```

U_OSC_triangular_oscillation
Triangular Oscillation



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Parameters		
Activate_BOOL	Activates the oscillation	0/1
Length_of_period_ms_UINT	Length of the period	0 to 65535
Mean_value_INT	Mean value	–16380 to 16380
Amplitude_UINT	Amplitude	0 to 16380
Outputs		
Oscillation_INT	Oscillation value	–32768 to 32767

Description

This function block generates a triangular oscillation in accordance with the parameters "Length of the period", "Mean value" and "Amplitude" (→ fig. 9). The parameters are accepted during the rising edge of "Activate_BOOL".

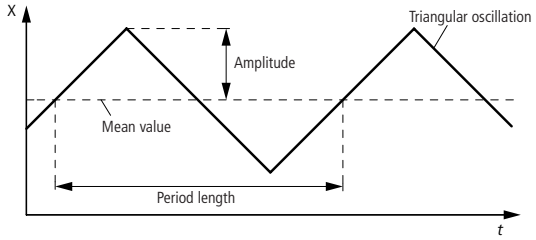


Figure 9: Triangular oscillation as a function of the parameters Mean value, Amplitude and Length of the period

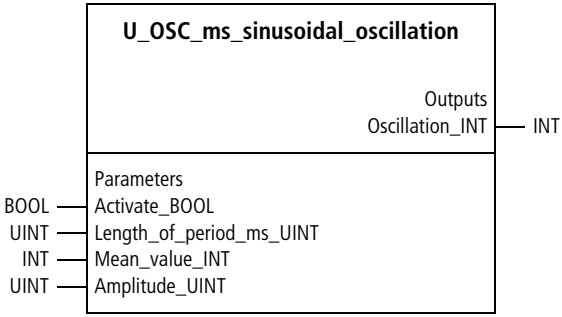
Example:
The application example implemented a triangular oscillation with the parameters:

- Length of the period = 30 s
- Mean value = 10000
- Amplitude = 5000
  => The oscillation lies in the range 5000 to 15000.

Application of the function block "U_OSC_triangular_oscillation" in the program "TrianOSC"

```
PROGRAM TrianOSC
VAR
  TRIANGULAR_OSCILLATION : U_OSC_TRIANGULAR_OSCILLATION ;
  Triangular_Oscillation_INT : INT ;
END_VAR

CAL TRIANGULAR_OSCILLATION(
  Activate_BOOL :=1,
  Length_of_period_ms_UINT :=30000,
  Mean_value_INT :=10000,
  Amplitude_UINT :=5000,
  Oscillation_INT=>Triangular_Oscillation_INT)
END_PROGRAM
```

U_OSC_ms_sinusoidal_oscillation
Sinusoidal Oscillation with Input in Milliseconds



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Parameters		
Activate_BOOL	Activates the oscillation	0/1
Length_of_period_ms_UINT	Length of the period	0 to 65535
Mean_value_INT	Mean value	–16380 to 16380
Amplitude_UINT	Amplitude	0 to 16380
Outputs		
Oscillation_INT	Oscillation value	–32768 to 32767

Description

This function block generates a sinusoidal oscillation in accordance with the parameters "Length of the period" (input in ms), "Mean value" and "Amplitude" (→ fig. 10). The parameters are accepted during the rising edge of "Activate_BOOL".

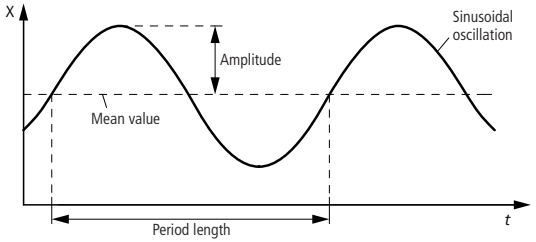


Figure 10: Sinusoidal oscillation as a function of the parameters Mean value, Amplitude and Length of the period

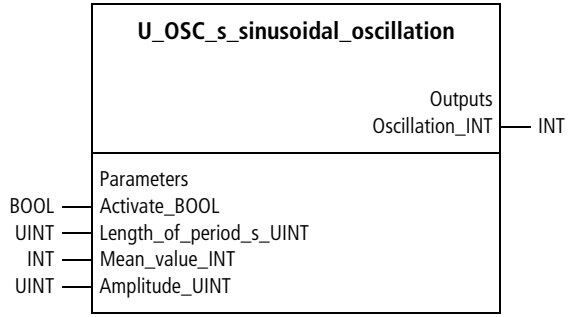
Example:
The application example implemented a sinusoidal oscillation with the parameters:

- Length of the period = 10 s
- Mean value = 0
- Amplitude = 100
  => The oscillation lies in the range –100 to 100.

Application of the function block "U_OSC_ms_sinusoidal_oscillation" in the program "Sinus_ms"

```
PROGRAM Sinus_ms
VAR
  MS_SINUSOIDAL_OSCILLATION : U_OSC_MS_SINUSOIDAL_OSCILLATION ;
  Sinusoidal_Oscillation_INT : INT ;
END_VAR

CAL MS_SINUSOIDAL_OSCILLATION(
  Activate_BOOL :=1,
  Length_of_period_ms_UINT :=10000,
  Mean_value_INT :=0,
  Amplitude_UINT :=100,
  Oscillation_INT=>Sinusoidal_Oscillation_INT)

END_PROGRAM
```

U_OSC_s_sinusoidal_oscillation
Sinusoidal Oscillation with Input in Seconds



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Parameters		
Activate_BOOL	Activates the oscillation	0/1
Length_of_period_s_UINT	Length of the period	0 to 65535
Mean_value_INT	Mean value	–16380 to 16380
Amplitude_UINT	Amplitude	0 to 16380
Outputs		
Oscillation_INT	Oscillation value	–32768 to 32767

Description

This function block generates a sinusoidal oscillation in accordance with the parameters "Length of the period" (input in seconds), "Mean value" and "Amplitude" (→ fig. 10). The parameters are accepted during the rising edge of "Activate_BOOL".

→ The length of the period can be entered in the unit of seconds. Thus, the maximum value is:

- 65535 s = 1092 min = 18.20 h

The function block "U_OSC_s_sinusoidal_oscillation" generates the sinusoidal oscillation only in steps of seconds. If the period is not longer than 65.535 s, you are recommended to use the function block "U_OSC_ms_sinusoidal_oscillation", which generates the ramp value in millisecond steps.

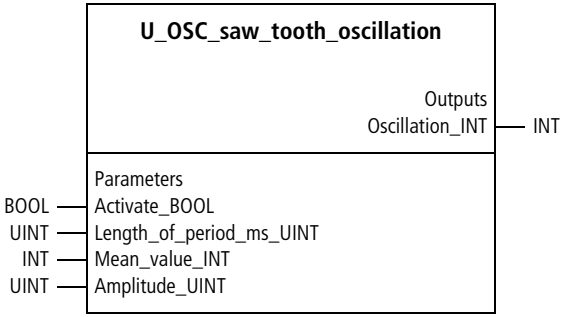
Example:
The application example implemented a sinusoidal oscillation with the parameters:

- Length of the period = 100 min
- Mean value = 0
- Amplitude = 100
  => The oscillation lies in the range –100 to 100.

Application of the function block "U_OSC_s_sinusoidal_oscillation" in the program "Sinus_s"

```
PROGRAM Sinus_s
VAR
  S_SINUSOIDAL_OSCILLATION : U_OSC_S_SINUSOIDAL_OSCILLATION ;
  Sinusoidal_Oscillation_INT : INT ;
END_VAR

CAL S_SINUSOIDAL_OSCILLATION(
  Activate_BOOL :=1,
  Length_of_period_s_UINT :=6000,
  Mean_value_INT :=0,
  Amplitude_UINT :=100,
  Oscillation_INT=>Sinusoidal_Oscillation_INT)

END_PROGRAM
```

U_OSC_saw_tooth_oscillation
Saw Tooth Oscillation



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Parameters		
Activate_BOOL	Activates the oscillation	0/1
Length_of_period_ms_UINT	Length of the period	0 to 65535
Mean_value_INT	Mean value	–16380 to 16380
Amplitude_UINT	Amplitude	0 to 16380
Outputs		
Oscillation_INT	Oscillation value	–32768 to 32767

Description

This function block generates a saw tooth oscillation in accordance with the parameters "Length of the period" (input in ms), "Mean value" and "Amplitude" (→ fig. 11). The parameters are accepted during the rising edge of "Activate_BOOL".

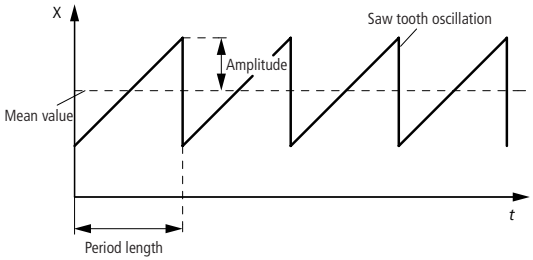


Figure 11: Saw tooth oscillation as a function of the parameters Mean value, Amplitude and Length of the period

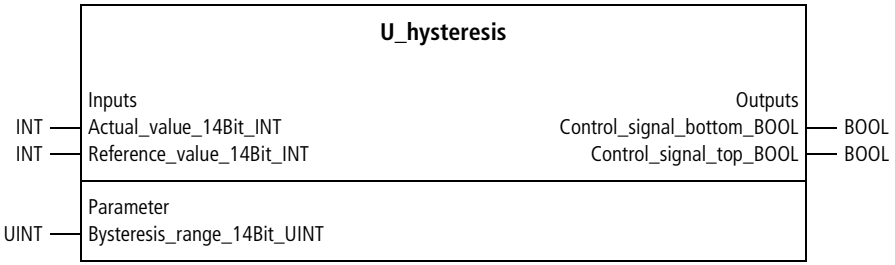
Example:
The application example implemented a saw tooth oscillation with the parameters:

- Length of the period = 100 ms
  (if the PLC cycle time is relatively large compared to the 100 ms, then the function block will generate stochastic values between 0 and 4094).
- Mean value = 2047
- Amplitude = 2047
  => The oscillation lies in the range 0 to 4094.

Application of the function block "U_OSC_saw_tooth_oscillation" in the program "SawtoOSC"

```
PROGRAM SawtoOSC
VAR
  SAW_TOOTH_OSCILLATION : U_OSC_SAW_TOOTH_OSCILLATION ;
  Saw_tooth_Oscillation_INT : INT ;
END_VAR

CAL SAW_TOOTH_OSCILLATION(
  Activate_BOOL :=1,
  Length_of_period_ms_UINT :=100,
  Mean_value_INT :=2047,
  Amplitude_UINT :=2047,
  Oscillation_INT=>Saw_tooth_Oscillation_INT)

END_PROGRAM
```

Non-linear control elements — U_hysteresis Hysteresis Element



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Actual_value_14Bit_INT	Actual value	0 to 16380
Reference_value_14Bit_INT	Reference value	0 to 16380
Parameter		
Hysteresis_range_14Bit_UINT	Hysteresis range	0 to 16380
Outputs		
Control_signal_bottom_BOOL	Bottom control signal	0/1
Control_signal_top_BOOL	Top control signal	0/1

Description

This function block implements a hysteresis function in accordance with the inputs for reference value and hysteresis range (→ fig. 12). Depending on the input value "Actual_value_14Bit_INT", the function block switches between "Control_signal_bottom_BOOL" and "Control_signal_top_BOOL" whenever the hysteresis range limits are exceeded.

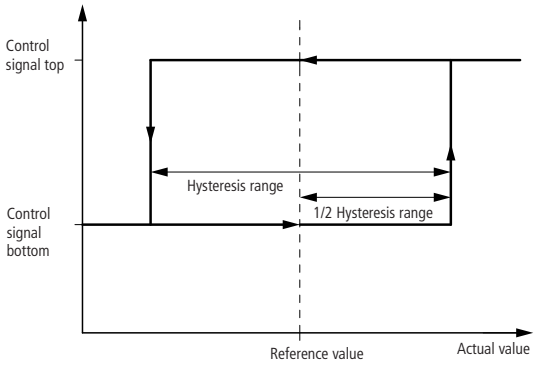


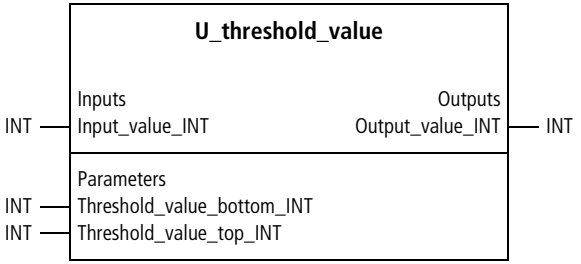
Figure 12: Hysteresis function for a reference value, bounded by a hysteresis range

Example:
The application example uses the hysteresis function block to implement a 2-step controller. The controller issues an output signal for heat whenever the actual value drops below the setpoint by 50 increments. An output signal for cooling is issued whenever the setpoint is exceeded by 50 increments.

Application of the function block "U_hysteresis" in the program "Contr_2P"

```
PROGRAM Contr_2P
VAR
  HYSTERESIS : U_HYSTERESIS ;
  Actual_value_12Bit_INT : INT ;
  Setpoint_value_12Bit_INT : INT ;
  Heat_BOOL : BOOL ;
  COOL_BOOL : BOOL ;
END_VAR

CAL HYSTERESIS(
  Actual_value_14Bit_INT :=Actual_value_12Bit_INT,
  Reference_value_14Bit_INT :=Setpoint_value_12Bit_INT,
  Hysteresis_range_14Bit_UINT :=100,
  Control_signal_top_BOOL=>Heat_BOOL,
  Control_signal_bottom_BOOL=>COOL_BOOL
)

END_PROGRAM
```

U_threshold_value
Threshold Value



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_INT	Input value	–32768 to 32767
Parameters		
Threshold_value_bottom_INT	Bottom threshold value	–32768 to 32767
Threshold_value_top_INT	Top threshold value	–32768 to 32767
Outputs		
Output_value_INT	Output value	–32768 to 32767

Description

This function block transfers the input value to the output only if the input exceeds the top threshold value (→ fig. 13) or drops below the bottom threshold value. Otherwise, a zero is output.

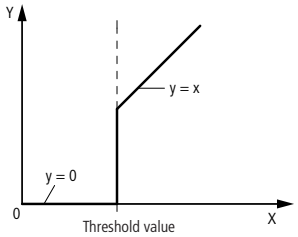


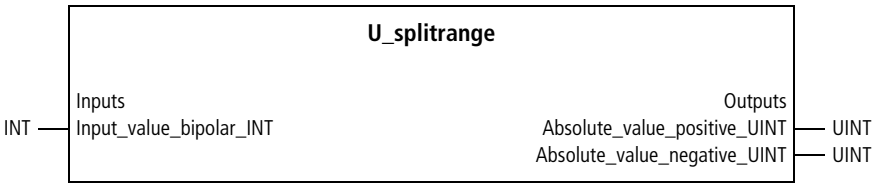
Figure 13: The input values X are only transferred if a threshold value is exceeded

Example:
The application example applies the threshold values (minimum control values) –500 and 500 to a bipolar 13-bit variable (–4095 to 4095). The function block only issues a non zero output if the bipolar variable is greater than 500 or less than –500.

Application of the function block "U_threshold_value" in the program "MinValue"

```
PROGRAM MinValue
VAR
    THRESHOLD_VALUE : U_THRESHOLD_VALUE ;
    Manipulated_variable_bipolar_13Bit_INT : INT ;
    Minimum_manipulated_variable_bipolar_13Bit_INT : INT ;
END_VAR

CAL THRESHOLD_VALUE(
    Input_value_INT :=Manipulated_variable_bipolar_13Bit_INT,
    Threshold_value_bottom_INT :=-500,
    Threshold_value_top_INT :=500
)
LD  THRESHOLD_VALUE.Output_value_INT
ST  Minimum_manipulated_variable_bipolar_13Bit_INT

END_PROGRAM
```

U_splitrange
Split Range



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_bipolar_INT	Bipolar input value	–32768 to 32767
Outputs		
Absolute_value_positive_UINT	Positive absolute value of the input value	0 to 32767
Absolute_value_negative_UINT	Negative absolute value of the input value	0 to 32767

Description

This function block splits a bipolar (positive or negative) input value in accordance with the criteria "positive" and "negative".

Example:

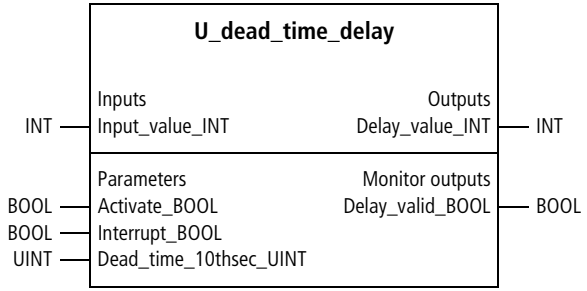
- Input_value_bipolar_INT = –1000
  => Absolute_value_positive_UINT = 0
  Absolute_value_negative_UINT = 1000
- Input_value_bipolar_INT = 2000
  => Absolute_value_positive_UINT = 2000
  Absolute_value_negative_UINT = 0

Example:
The application example splits a 13-bit, bipolar variable (–4095 to 4095) into negative (–4095 to –1) and positive (0 to 4095) values, and outputs them as absolute values. These absolute values are used as variables for a temperature control, with actuators for heating and cooling.

Application of the function block "U_splitrange" in the program "Split_MV"

```
PROGRAM Split_MV
VAR
  SPLITRANGE : U_SPLITRANGE ;
  Manipulated_variable_bipolar_13Bit_INT : INT ;
  Manipulated_variable_heat_13Bit_UINT : UINT ;
  Manipulated_variable_cool_13Bit_UINT : UINT ;
END_VAR

CAL SPLITRANGE(
  Input_value_bipolar_INT :=Manipulated_variable_bipolar_13Bit_INT,
  Absolute_value_positive_UINT=>Manipulated_variable_heat_13Bit_UINT,
  Absolute_value_negative_UINT=>Manipulated_variable_cool_13Bit_UINT
)

END_PROGRAM
```

U_dead_time_delay
Dead Time Delay



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_INT	Input value which is dead time-delayed	–32768 to 32767
Parameters		
Activate_BOOL	Activates the dead time delay	0/1
Interrupt_BOOL	Interruption	0/1
Dead_time_10thsec_UINT	Dead time delay	0 to 65535
Outputs		
Delay_value_INT	Dead time-delayed value	–32768 to 32767
Monitor outputs		
Delay_valid_BOOL	Message: The dead time delay has finished the initial phase and is current	0/1

Description

This function block provides a dead time delay of the input value. The dead time delay begins after "Activate_BOOL" switches from "0" to "1". The first dead time-delayed values can be output after an initial dead time phase.
The variable "Delay_valid_BOOL" is then "1".
If "Interrupt_BOOL=1", the dead time delay is prolonged by an amount of time equal to the interruption.

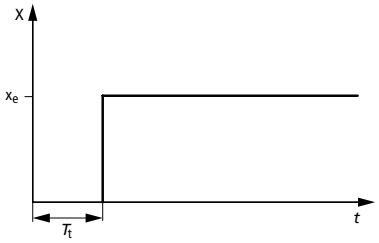


Figure 14: Dead time delay T_t between the input values X_e and the output values X

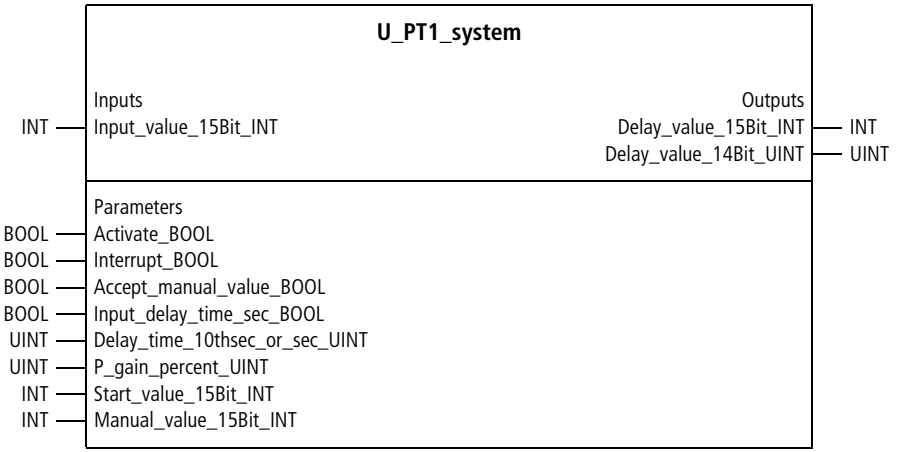
Example:
The application example compares a non delayed actual value with an actual value which occurred 100 s earlier. By subtracting the two values, the change in the value can be calculated.

Application of the function block "U_dead_time_delay" in the program "Gradient"

```
PROGRAM Gradient
VAR
        DEAD_TIME_DELAY : U_DEAD_TIME_DELAY ;
        Actual_value_UINT : UINT ;
        Gradient_actual_value_INT : INT ;
END_VAR

CAL DEAD_TIME_DELAY(
        Input_value_UINT :=Actual_value_UINT,
        Activate_BOOL :=1,
        Interrupt_BOOL :=0,
        Dead_time_10thsec_UINT :=1000
)
LD      DEAD_TIME_DELAY.Delay_valid_BOOL
JMPCN   END_DEAD_TIME_DELAY
        LD  Actual_value_UINT
        UINT_TO_INT
        SUB (DEAD_TIME_DELAY.Delay_value_UINT
            UINT_TO_INT
            )
        ST  Gradient_actual_value_INT
END_DEAD_TIME_DELAY:

END_PROGRAM
```

PTn systems

U_PT1_system
PT1 System



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_15Bit_INT	Input value which is PT1-delayed	–16380 to 16380
Parameters		
Activate_BOOL	Activates the PT1 delay	0/1
Interrupt_BOOL	Interruption	0/1
Accept_manual_value_BOOL	Accept manual value	0/1
Input_delay_time_sec_BOOL	Choice: The delay time can be entered in seconds or milliseconds	0/1

Designation	Significance	Value range
Delay_time_10thsec_or_sec_UINT	Delay time	0 to 65535
P_gain_percent_UINT	Proportional gain	0 to 65535
Start_value_15Bit_INT	Start value	–32768 to 32767
Manual_value_15Bit_INT	Manual value	–32768 to 32767
Outputs		
Delay_value_15Bit_INT	PT1-delayed value	–16380 to 16380
Delay_value_14Bit_UINT	PT1-delayed value	0 to 16380

Description

The PT1 behaviour of the function block is started with "Activate_BOOL=1" and reset with "Activate_BOOL=0" (→ fig. 15). After a reset, "Reactivation" begins the PT1 delay with "Start_value_15Bit_INT". "Interrupt_BOOL=1" additionally delays the PT1 delay by an amount of time equal to the interruption (the delay value is "frozen").

"Accept_manual_value_BOOL=1" accepts "Manual_value_INT" as the delay value. The delay time can be entered in tenths of a second with "Input_delay_time_sec_BOOL=0" and in seconds with "Input_delay_time_sec_BOOL=1". With "P_gain_percent_UINT", proportional gain of the input signal can be entered as a percentage (100 % = 1.0).

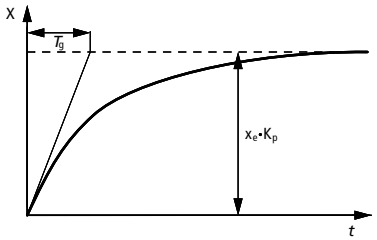


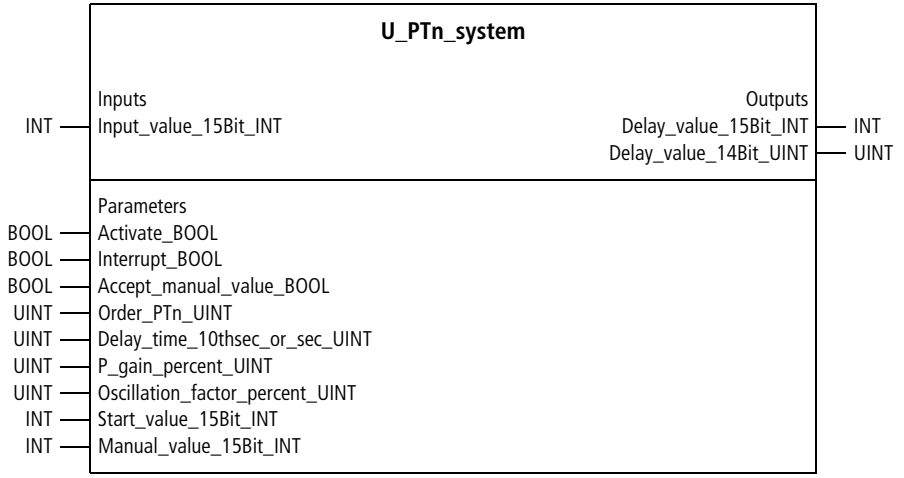
Figure 15: PT1-delayed transient behaviour between the input signal X_e and the output signal X

Example:
The application example creates a PT2 system by combining two PT1 systems. The system gain of the PT2 system is 2.0.

Application of the function block "U_PT1_system" in the program "PT2"

```
PROGRAM PT2
VAR
  PT1_SYSTEM_a : U_PT1_SYSTEM ;
  PT1_SYSTEM_b : U_PT1_SYSTEM ;
  Manipulated_variable_12Bit_INT : INT ;
  P_gain_percent_UINT : UINT ;
  Actual_value_UINT : UINT ;
END_VAR

CAL PT1_SYSTEM_a(
  Input_value_15Bit_INT :=Manipulated_variable_12Bit_INT,
  Activate_BOOL :=1,
  Interrupt_BOOL :=0,
  Accept_manual_value_BOOL :=0,
  Input_delay_time_sec_BOOL :=1,
  Delay_time_10thsec_or_sec_UINT :=25,
  P_gain_percent_UINT :=100,
  Start_value_15Bit_INT :=0,
  Manual_value_15Bit_INT :=0)
CAL PT1_SYSTEM_b(
  Input_value_15Bit_INT :=PT1_SYSTEM_a.Delay_value_15Bit_INT,
  Activate_BOOL :=1,
  Interrupt_BOOL :=0,
  Accept_manual_value_BOOL :=0,
  Input_delay_time_sec_BOOL :=1,
  Delay_time_10thsec_or_sec_UINT :=25,
  P_gain_percent_UINT :=P_gain_percent_UINT,
  Start_value_15Bit_INT :=0,
  Manual_value_15Bit_INT :=0,
  Delay_value_14Bit_UINT=>Actual_value_UINT)

END_PROGRAM
```

U_PTn_system
PTn System



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_15Bit_INT	Input value which is PTn-delayed	–16380 to 16380
Parameters		
Activate_BOOL	Activates the PTn delay	0/1
Interrupt_BOOL	Interruption	0/1
Accept_manual_value_BOOL	Accept manual value	0/1
Input_delay_time_sec_BOOL	Choice: The delay time can be entered in seconds or milliseconds	0/1
Order_PTn_UINT	Order of the PTn system	0 to 10

Designation	Significance	Value range
Delay_time_10thsec_or_sec_UINT	Delay time	0 to 65535
P_gain_percent_UINT	Proportional gain	0 to 65535
oscillation_factor_percent_UINT	Oscillation factor	0 to 65535
start_value_15Bit_INT	Start value	–32768 to 32767
manual_value_15Bit_INT	Manual value	–32768 to 32767
Outputs		
delay_value_15Bit_INT	PT1-delayed value	–16380 to 16380
delay_value_14Bit_UINT	PT1-delayed value	0 to 16380

Description

The PTn behaviour of the function block is started with "Activate_BOOL=1" and reset with "Activate_BOOL=0" (→ fig. 16). After a reset, "Reactivation" begins the PTn delay with "Start_value_15Bit_INT". "Interrupt_BOOL=1" additionally delays the PTn delay by an amount of time equal to the interruption (the delay value is "frozen").

"Accept_manual_value_BOOL=1" accepts "Manual_value_INT" as the delay value. The delay time can be entered in tenths of a second with "Input_delay_time_sec_BOOL=0" and in seconds with "Input_delay_time_sec_BOOL=1". With "P_gain_percent_UINT", proportional gain of the input signal can be entered as a percentage (100 % = 1.0). If "Oscillation_factor_percent_UINT" is non zero, the PTn systems oscillate.

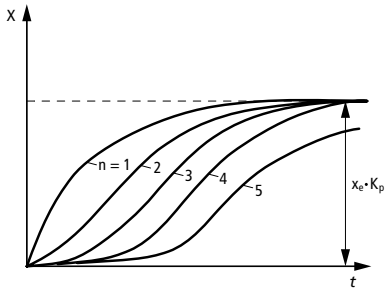


Figure 16: The transition behaviour of PTn gates of various orders (PT1 to PT5)

Example:
The application example creates a PT3 controlled system with the function block PTn system. An actual value and regulated quantity connect the controlled system with a controller.

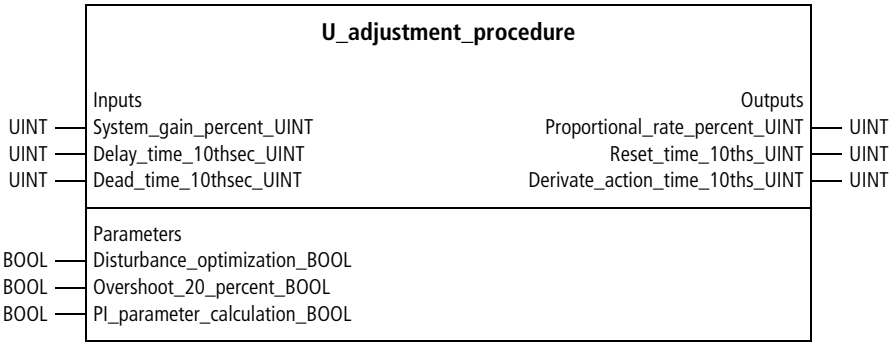
Application of the function block "U_PTn_system" in the program "Simulate"

```
PROGRAM Simulate
VAR
    PTN_SYSTEM : U_PTN_SYSTEM ;
    LIM_UINT_LIMITER : U_LIM_UINT_LIMITER ;
    PID_CONTROLLER : U_PID_CONTROLLER ;
    Manipulated_variable_12Bit_UINT : UINT ;
    P_gain_percent_UINT : UINT ;
    Actual_value_UINT : UINT ;
END_VAR
LD  Manipulated_variable_12Bit_UINT
UINT_TO_INT
ST  PTN_SYSTEM.Input_value_15Bit_INT
CAL PTN_SYSTEM(
    Activate_BOOL :=1,
    Interrupt_BOOL :=0,
    Accept_manual_value_BOOL :=0,
    Input_delay_time_sec_BOOL :=0,
    Order_PTn_UINT :=3,
    Delay_time_10thsec_or_sec_UINT :=200,
    P_gain_percent_UINT :=100,
    Oscillation_factor_percent_UINT :=0,
    Start_value_15Bit_INT :=0,
    Manual_value_15Bit_INT :=0
)
```

```
CAL LIM_UINT_LIMITER(
    Unlimited_value_UINT :=PTN_SYSTEM.Delay_value_14Bit_UINT,
    Upper_limit_UINT :=0,
    Lower_limit_UINT :=4095,
    Limited_value_UINT=>Actual_value_UINT
    )

CAL PID_CONTROLLER(
    Setpoint_value_12Bit_UINT :=2000,
    Actual_value_12Bit_UINT :=Actual_value_UINT,
    P_activate_BOOL :=1,
    I_activate_BOOL :=1,
    D_activate_BOOL :=1,
    Accept_manual_manipulated_variable_BOOL :=0,
    Proportional_rate_percent_UINT :=155,
    Reset_time_10ths_UINT :=130,
    Derivate_action_10ths_UINT :=32,
    Manual_manipulated_variable_12Bit_UINT :=0,
    Manipulated_variable_12Bit_UINT=>Manipulated_variable_12Bit_UINT
    )

END_PROGRAM
```

PID controller settings

U_adjustment_procedure
Setting Procedures for PID Controllers



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
System_gain_percent_UINT	System gain Ks	0 to 65535
Delay_time_10thsec_UINT	System delay time Tg	0 to 65535
Dead_time_10thsec_UINT	System dead time Tu	0 to 65535
Parameters		
Disturbance_optimization_BOOL	Optimization criterion: control behaviour/disturbance behaviour	0/1
Overshoot_20_percent_BOOL	Optimization criterion: aperiodic boundary case/20 % overshoot	0/1
PI_parameter_calculation_BOOL	Optimization adjustment: PID controller/PI controller	0/1
Outputs		
Proportional_rate_percent_UINT	Controller proportional gain Kp	0 to 65535
Reset_time_10ths_UINT	Controller reset time Tn	0 to 65535
Derivate_action_time_10ths_UINT	Controller derivative action time Tv	0 to 65535

Description

Based on the characteristics of a PTn controlled system,

- system gain Ks
- delay time Tg and
- dead time Tu

the function block calculates the adjustment parameters for a PID controller (or PI controller; → below)

- proportional gain Kp
- reset time Tn and
- derivative action time Tv (only for optimization of PID controllers).

The optimization can be based on the following criteria and adjustment options:

- control behaviour when "Disturbance_optimization_BOOL=0"
- disturbance behaviour when "Disturbance_optimization_BOOL=1"
- Aperiodic boundary case when "Overshoot_20_percent_BOOL=0"
- 20 % overshoot when "Overshoot_20_percent_BOOL=1"
- calculation for PID controllers when "PI_parameter_calculation_BOOL=0"
- calculation for PI controllers when "PI_parameter_calculation_BOOL=1"

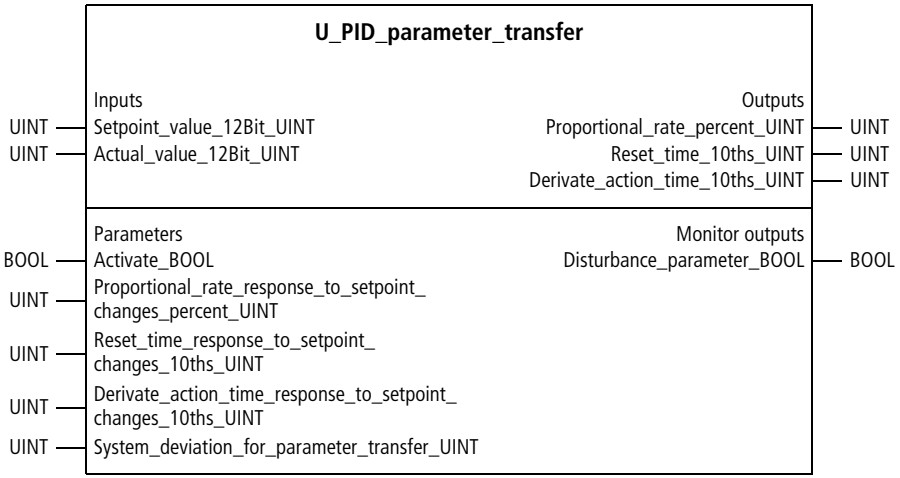
Example:
Based on the entered system data, the application example calculates the PID controller parameters (→ function block outputs) for the following optimization criteria:

- control behaviour
- aperiodic boundary case
- for PID (not only PI)

Application of the function block "U_adjustment_procedure" in the program "PID_PAR"

```
PROGRAM PID_PAR
VAR
  ADJUSTMENT_PROCEDURE : U_ADJUSTMENT_PROCEDURE ;
END_VAR

CAL ADJUSTMENT_PROCEDURE(
  System_gain_percent_UINT :=150,
  Delay_time_10thsec_UINT :=300,
  Dead_time_10thsec_UINT :=50,
  Disturbance_optimization_BOOL :=0,
  Overshoot_20_percent_BOOL :=0,
  PI_parameter_calculation_BOOL :=0
)

END_PROGRAM
```

U_PID_parameter_transfer
Parameter transfer from "optimized parameters for response to setpoint change" to "optimized parameters for disturbance" of a PID controller



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Setpoint_value_12Bit_UINT	Setpoint value	0 to 4095
Actual_value_12Bit_UINT	Actual value	0 to 4095

Designation	Significance	Value range
Parameters		
Activate_BOOL	Activates the function block	0/1
Proportional_rate_response_to_ setpoint_changes_percent_UINT	Proportional rate [%] optimized for response to setpoint changes	0 to 65535
Reset_time_response_to_ setpoint_changes_10ths_UINT	Reset time [0.1 s] optimized for response to setpoint changes	0 to 65535
Derivate_action_time_response_ to_setpoint_changes_10ths_UINT	Derivate action time [0.1 s] optimized for response to setpoint changes	0 to 65535
System_deviation_for_parameter_ transfer_UINT	System deviation for paramater transfer	0 to 65535
Outputs		
Proportional_rate_percent_UINT	Proportional rate for response to setpoint changes or disturbance optimization	0 to 65535
Reset_time_10ths_UINT	Reset time for response to setpoint changes or disturbance optimization	0 to 65535
Derivate_action_time_10ths_UINT	Derivate action time for response to setpoint changes or disturbance optimization	0 to 65535
Monitor outputs		
Disturbance_parameter_BOOL	Status signal: The disturbance parameters are active	0/1

Description

The function block changes the output parameters for "Disturbance optimization" in dependence of "System_deviation_for_parameter_transfer_UINT". In case of setpoint changes the function block changes the output parameter for "Setpoint change optimization". The result of the function block is, to optimize simultaneous for disturbance and setpoint changes.

Application of the function block "U_PID_parameter_transfer" in the program "PID_opti"

```
PROGRAM PID_opti

VAR
  PID_parameter_transfer_zone1 : U_PID_parameter_transfer ;
  PID_controller_zone1 : U_PID_controller ;
  Manipulated_variable_12Bit_UINT : UINT ;
  Setpoint_value_12Bit_UINT : UINT ;
  Actual_value_12Bit_UINT : UINT ;
END_VAR

CAL PID_parameter_transfer_zone1(
  Setpoint_value_12Bit_UINT :=Setpoint_value_12Bit_UINT,
  Actual_value_12Bit_UINT :=Actual_value_12Bit_UINT,
  Activate_BOOL :=1,
  Proportional_rate_response_to_setpoint_changes_percent_UINT :=300,
  Reset_time_response_to_setpoint_changes_10ths_UINT :=2000,
  Derivate_action_time_response_to_setpoint_changes_10ths_UINT :=300,
  System_deviation_for_parameter_transfer_UINT :=20,
  Proportional_rate_percent_UINT=>300,
  Reset_time_10ths_UINT=>2000,
  Derivate_action_time_10ths_UINT=>300,
  Disturbance_parameter_BOOL=>0)

  LD  PID_parameter_transfer_zone1.Proportional_rate_percent_UINT
  ST  PID_controller_zone1.Proportional_rate_percent_UINT
  LD  PID_parameter_transfer_zone1.Reset_time_10ths_UINT
  ST  PID_controller_zone1.Reset_time_10ths_UINT
  LD  PID_parameter_transfer_zone1.Derivate_action_time_10ths_UINT
  ST  PID_controller_zone1.Derivate_action_time_10ths_UINT
```

```
CAL PID_controller_zone1(
  Setpoint_value_12Bit_UINT :=Setpoint_value_12Bit_UINT,
  Actual_value_12Bit_UINT :=Actual_value_12Bit_UINT,
  P_activate_BOOL :=1,
  I_activate_BOOL :=1,
  D_activate_BOOL :=1,
  Accept_manual_manipulated_variable_BOOL :=0,
  Manual_manipulated_variable_12Bit_UINT :=0,
  Manipulated_variable_12Bit_UINT=>Manipulated_variable_12Bit_UINT)

END_PROGRAM
```

4 Controllers

Fundamentals and general information

A closed control system with a PID controller (→ fig. 17) consists of the following components:

- Setpoint
- Actual value
- Deviation = Setpoint – Actual value
- Controller (e.g. PID controller)
- Controlled system (e.g. PTn system)
- Disturbances

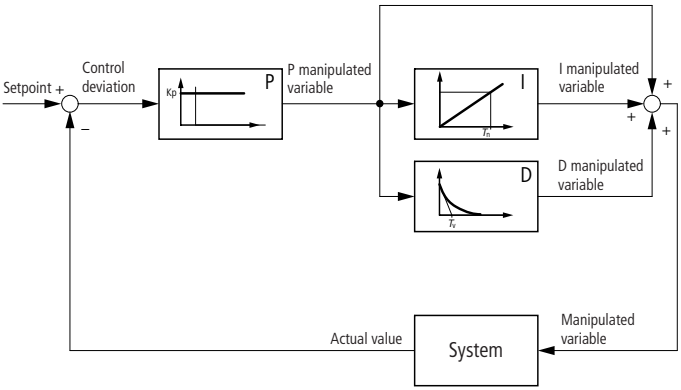


Figure 17: Block diagram of an actual PID controller (= PIDT1 controller)

Standardised control behaviour

The basis for control is a deviation, which is formed as follows:

- Deviation = Setpoint – Actual value

The following relationships apply for the manipulated variable components P, I and D:

$$\text{P\_manipulated variable} = \text{Kp} \times \text{deviation}$$

$$\text{I\_manipulated variable} = \text{I\_manipulated variable} + \frac{(t - t0) \times \text{P\_manipulated variable}}{\text{Tn}}$$

$$\text{D\_manipulated variable} = \frac{(\text{current control dev.} - \text{last control dev.}) \times \text{Kp} \times \text{Tv}}{\text{Sampling time}}$$

(The sampling time is the time during which the current deviation is compared with an earlier deviation. This time is calculated automatically. The D-variable is PT1-delayed in accordance with the input for Tv).

The following applies for the reset time Tn:
After the reset time Tn expires and with the P variable constant, the I variable increases constantly by an amount equal to the value of the P-variable.

The following applies for the derivative action time Tv:
Assuming the P-variable is a ramp function, the D variable currently has the value equivalent to the change of the P variable during the time Tv (→ fig. 18).

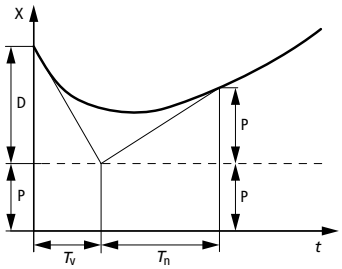


Figure 18: Transition function of an actual PID controller

Antiwindup procedure

The integrator of the PID controller stops automatically if the overall manipulated variable reaches its maximum value (4095) or minimum value (0). This improves the control behaviour with large changes of the setpoint.

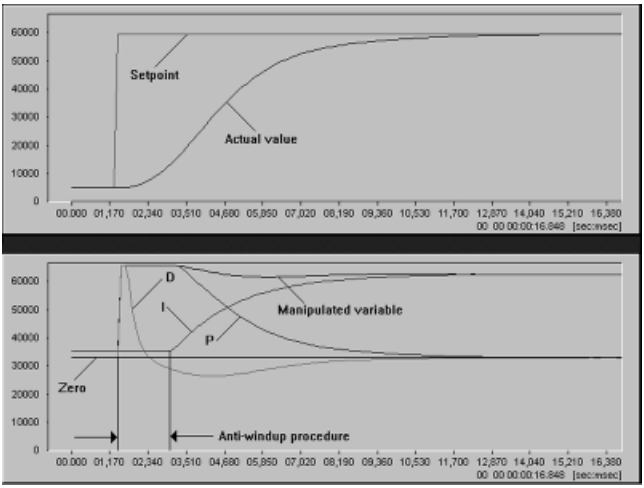


Figure 19: The Antiwindup technique stops the integrator (I component) of the PID controller when the overall manipulated variable reaches a limiting value

Real D component calculation

An ideal D component calculation is not possible in actual practice. In particular, implementation with real actuators and an actual process is not possible (an infinitely large setpoint would have to be implemented for an infinitesimal period of time). Therefore, the D component calculation is performed using "real world conditions", meaning the D-value output is delayed according to the parameter for the derivative action time Tv. This way, the D component of the manipulated variable progresses without oscillations.

Smooth acceptance of the manual manipulated variable (=> disturbance compensation)

The switchover from manual operation to automatic operation (the controller generates the manipulated variable) can be implemented without interruptions by readjusting the internal I component of the manipulated variable accordingly. This improves the disturbance behaviour, if the approximate time and magnitude of the disturbance are known (→ fig. 20). An example is temperature control of an extruder when the extruder worm begins to move after the heating phase. Another typical example is a temperature zone in which a fluid is injected after a heating phase.

When a disturbance occurs, the requirements made by the manipulated variable of the controller change. This results in a temporary control deviation, until the controller reaches the corresponding manipulated variable by means of its I component. To compensate the disturbance, when it occurs a manual manipulated variable can be sent to the controller. This can be either a relative change of the current manipulated variable, or an absolute value. In a sense, this indicates to the controller the direction in which it must change its manipulated variable, even before the disturbance becomes noticeable (→ fig. 20). The "smooth acceptance of the manual manipulated variable" to the controller is implemented as follows:

- ▶ Set "Accept_manual_manipulated_variable_BOOL" to "1"
- ▶ Assign "Manual_manipulated_variable_12Bit_UINT" to the value which you want the controller to accept
- ▶ Call the PID controller function block
- ▶ Set "Accept_manual_manipulated_variable_BOOL" to "0"
- ▶ Whenever the PID controller function block is called again, the controller will accept the manual value, and will continue to generate the manipulated variable without interruptions.

Example:
Before a disturbance begins, the system levels off (setpoint = actual value) at around 20 % of the magnitude of the manipulated variable. After the disturbance, there is a levelling off at about 80 % of the magnitude of the manipulated variable.

=> A manipulated variable of around 80 % is sent to the controller when the disturbance begins.

If the disturbance is no longer a factor, the procedure is carried out in the other direction.

=> A manipulated variable of around 20 % is sent to the controller when the disturbance ceases (→ fig. 20).

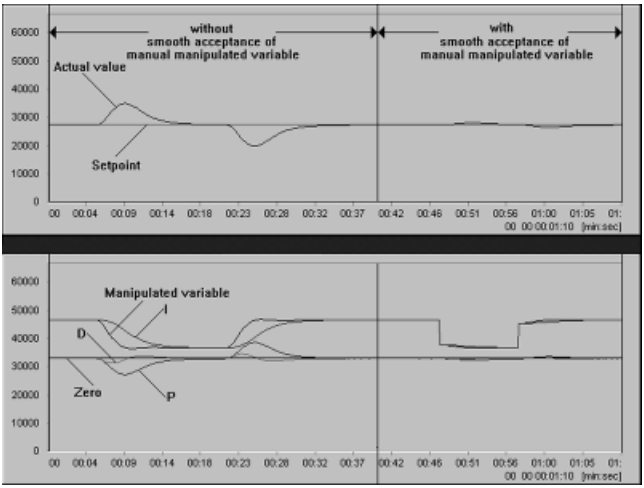


Figure 20: Compensating for a disturbance with "smooth acceptance of the manual manipulated variable"

Substituting existing PID controllers (e.g. hardware controllers)

If you have already implemented a functioning PID controller with hardware or software controllers, then you already have the parameters Kp, Tn and Tv. You can use the existing times Tn and Tv (note the units).

The value for Kp depends on the value ranges of the sensors (assignment: value range => 12 bits). If the value range has not changed, you can use the existing value for Kp. Otherwise, the relationship between Kp of the new controller and Kp of the existing one is as follows:

$$Kp\_new = \frac{Kp\_old \times Valuerange\_new}{Valuerange\_old}$$

Example:
The following values apply for a temperature control:

Table 1: Example of how to convert the known PID parameters for an existing solution to PID parameters of the toolbox controller

	Controller	
	"old"	**"new"**
12-bit value range (0 to 4095)	0 to 200 °C	–100 to 500 °C
Kp	1.20	360 [%] (= 3.6)
Tn	50 [s]	500 [0.1 s] (= 50 s)
Tv	8 [s]	80 [0.1 s] (= 8 s)

Kp is calculated as follows:

$$Kp\_new = \frac{Kp\_old \times (500\ °C - (-100\ °C))}{(200\ °C - 0\ °C)} = Kp\_old \times 3$$

$$Kp\_new = \frac{1.2 \times 600}{200} = 3.6$$

Rules for adjusting PID controllers according to Chien, Hrones and Reswick (CHR)

The adjustment rules of CHR are recommended for high-order PTn systems. The parameters are calculated in terms of the system gain Ks (= transfer coefficient), the system delay time Tg and the system dead time Tu.

The system parameters can be determined graphically, from the transition function for a PTn system (→ fig. 21). A PTn system is created by placing a spontaneous change of the manipulated variable on the system (e.g. from 0 % to 20 %). The parameters Tg and Tu are determined by placing a tangent to the reversing point. The system gain Ks is the quotient "Actual value/Manipulated variable" which results after the transition phase.

Example:
For a manipulated variable of 24 %, after a transition time the actual value settles at 96 % of the maximum measured value (= 4095 for 12-bit resolution) (→ fig. 21).

$$\Rightarrow \text{system gain } Ks = \frac{96\ \%}{24\ \%} = 4.0$$

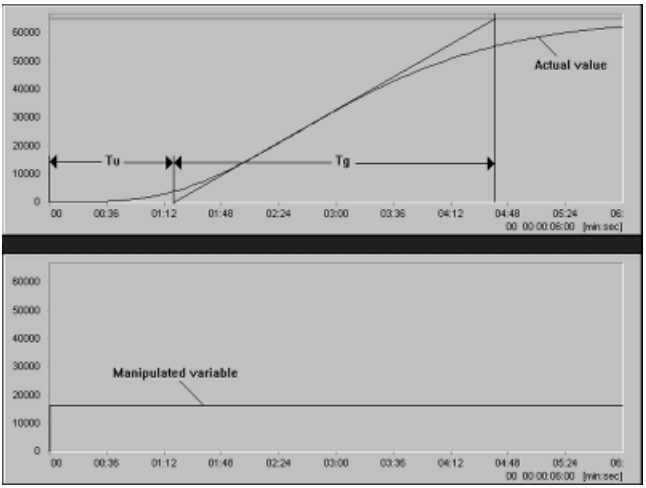


Figure 21: Transition function of a PT5 system with the system parameters: system gain Ks = 4.0, system delay time Tg = 200 s, system dead time Tu = 82 s

Table 2: PID adjustment rules for high order PTn systems according to Chien, Hrones and Reswick (CHR)

Controller	Aperiodic boundary case			20 % overshoot	
	Disturbance		Control	Disturbance	Control
P	Kp	0.3/Ks × Tg/Tu	0.3/Ks × Tg/Tu	0.7/Ks × Tg/Tu	0.7/Ks × Tg/Tu
PI	Kp	0.6/Ks × Tg/Tu	0.35/Ks × Tg/Tu	0.7/Ks × Tg/Tu	0.6/Ks × Tg/Tu
	Tn	4 × Tu	1.2 × Tg	2.3 × Tu	1.0 × Tg
PID	Kp	0.95/Ks × Tg/Tu	0.6/Ks × Tg/Tu	1.2/Ks × Tg/Tu	0.95/Ks × Tg/Tu
	Tn	2.4 × Tu	1.0 × Tg	2.0 × Tu	1.35 × Tg
	Tv	0.42 × Tu	0.5 × Tu	0.42 × Tu	0.47 × Tu

Empirical setting of a PID controller for PTn systems and optimization regarding control behaviour with aperiodic boundary case

Deactivate the D component of the controller, and first optimise the setting for a PI controller (→ U_PI_controller, page 232). Then calculate the PID-parameters as follows:

- Kp_PID = Kp_PI × 3.0
- Tn_PID = Tn_PI × 1.6
- Tv_PID = Tn_PI × 0.125

For a spontaneous change in the value of the setpoint, compare the controlled behaviour with figures 22 to 33. These figures compare optimised PID controller parameters with non optimized ones. Refer to these figures to determine if any parameters need adjustment.

If the settings are unexpected, first change Tv. Make it larger for high order PTn systems; make it smaller for low order PTn systems.

If this does not result in an optimum setting, then reduce Kp. optimise PTn systems according to the criteria:

- Control behaviour/Disturbance behaviour
- Aperiodic boundary case/Overshoot 20 %

If you are able to optimise the setting for control behaviour with aperiodic boundary case, then you can optimise for the other criteria by assigning the parameters in accordance with table 2.

Example:

Table 3: Example for how to convert known PID parameters to PID parameters for other optimization criteria according to Chien, Hrones and Reswick (CHR)

	Aperiodic boundary case	**20 % overshoot**
Control behaviour	Kp = 100	Kp = 100 × 95/60 = 158
	Tn = 1000	Tn = 1000 × 135/100 = 1350
	Tv = 50 (known parameters)	Tv = 50 × 47/50 = 47
Disturbance behaviour	Kp = 100 × 95/60 = 158	Kp = 100 × 120/60 = 200
	Tn = 50 × 240/50 = 240	Tn = 50 × 200/50 = 200
	Tv = 50 × 42/50 = 42	Tv = 50 × 42/50 = 42

Diagnosis of PI controller settings

Optimised setting for later comparison with less than optimum settings:

Figure 22 illustrates the control behaviour of a PI controller whose parameters are optimised for the criteria "Control behaviour" with "Aperiodic boundary case". When the setpoint changes, the actual value approaches the setpoint without overshooting. Immediately after the spontaneous change of the setpoint, the overall manipulated variable increases suddenly. It then approaches the level needed to stabilise the changed setpoint. The I component of the manipulated variable (which is the overall manipulated variable in case of 100 % stabilisation) asymptotically approaches the overall manipulated variable and does not intersect it. Immediately after the setpoint change, the P component of the manipulated variable generates a large manipulated variable component which depends on the control deviation. The P component becomes zero as the actual value and setpoint approach.

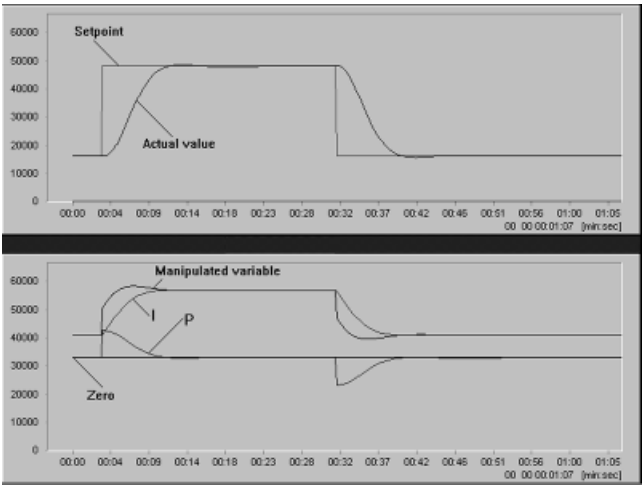


Figure 22: Optimised parameters of a PI controller for the criteria "control behaviour" with "aperiodic boundary case"

Defined incorrect parameter values (double or half) with reference to the optimum setting:

In figure 23 to 26, one parameter is double or half the optimum value specified in figure 22. These figures illustrate the effects of this on the control behaviour and the temporal behaviour of the manipulated variable components. If you want to analyse and modify non optimum parameter values, refer to these figures. They will help in determining which parameter is responsible to what degree for the non optimum control behaviour.

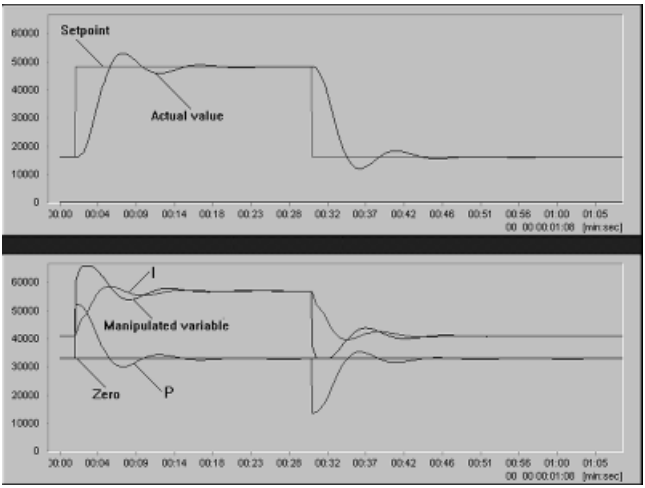


Figure 23: The proportional gain Kp is double the optimum value specified in figure 22

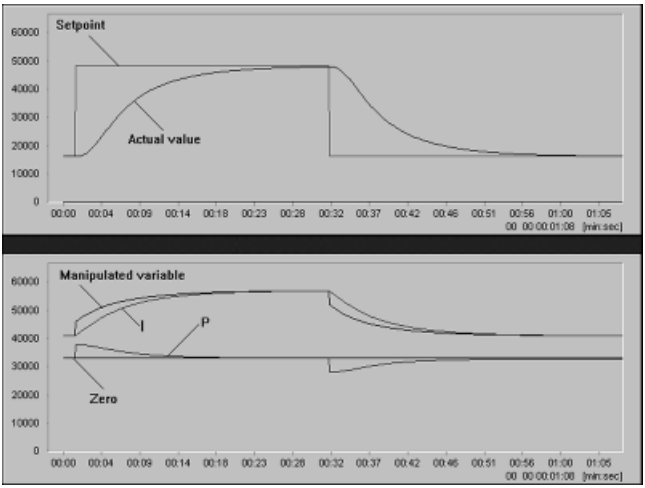


Figure 24: The proportional gain Kp is half the optimum value specified in figure 22

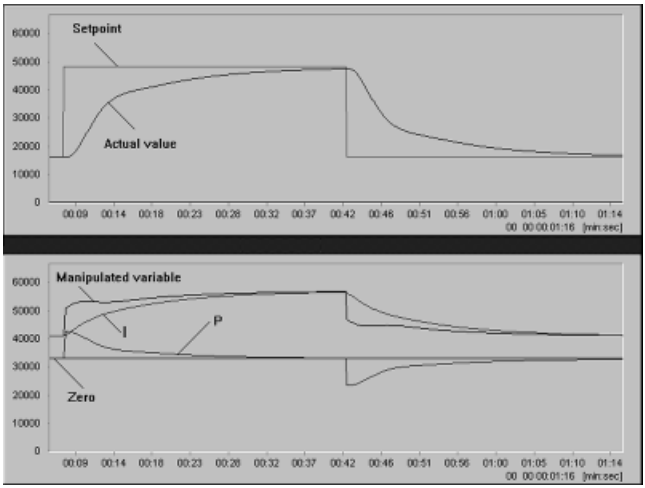


Figure 25: The reset time Tn is double the optimum value specified in figure 22

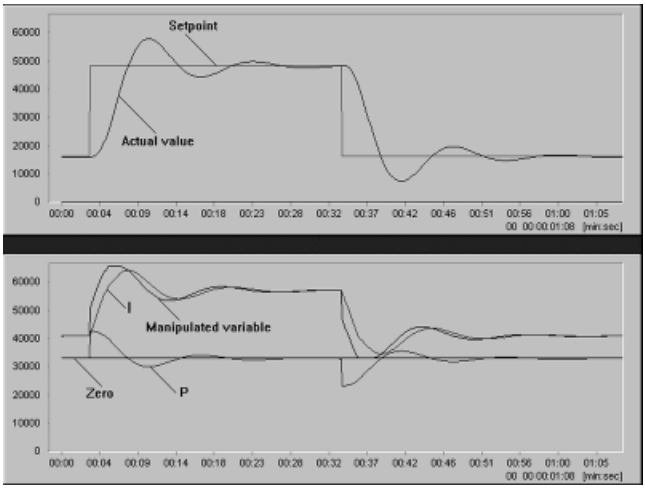


Figure 26: The reset time Tn is half the optimum value specified in figure 22

Diagnosis of PID controller settings

Optimised setting for later comparison with less than optimum settings:

Figure 27 illustrates the control behaviour of a PID controller whose parameters are optimised for the criteria "Control behaviour" with "Aperiodic boundary case". When the setpoint changes, the actual value approaches the setpoint without overshooting. Immediately after the spontaneous change of the setpoint, the overall manipulated variable outputs the maximum value. Then, without overshooting, it approaches the level required to stabilise the changed setpoint. The I component of the manipulated variable (which is the overall manipulated variable in case of 100 % stabilisation) asymptotically approaches the overall manipulated variable and does not intersect it.

Immediately after the setpoint change, the P and D components of the manipulated variable generate a large manipulated variable component, which depends on the control deviation and the change in the control deviation. These two components become zero as the actual value and setpoint approach without oscillation.

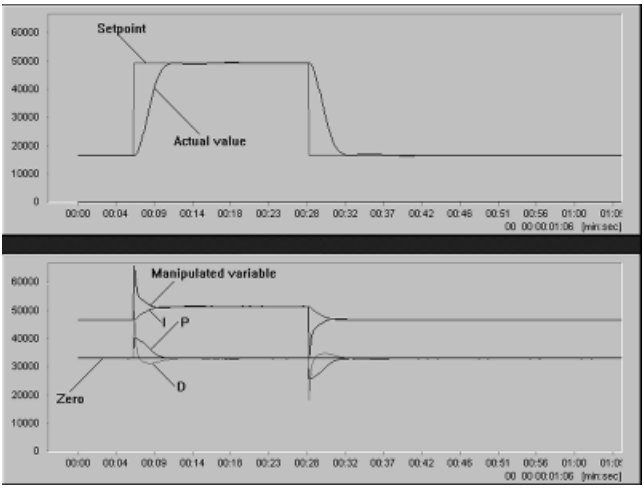


Figure 27: Optimised parameters of a PID controller for the criteria "control behaviour" with "aperiodic boundary case"

Defined incorrect parameter values (double or half) with reference to the optimum setting:

In figure 28 to 33, one parameter is double or half the optimum value specified in figure 27. These figures illustrate the effects of this on the control behaviour and the temporal behaviour of the manipulated variable components. If you want to analyse and modify non optimum parameter values, refer to these figures. They will help in determining which parameter is responsible to what degree for the non optimum control behaviour.

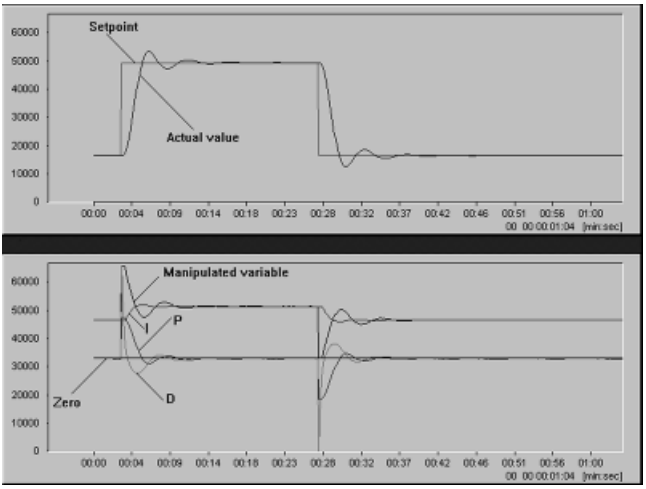


Figure 28: The proportional gain Kp is double the optimum value specified in figure 27

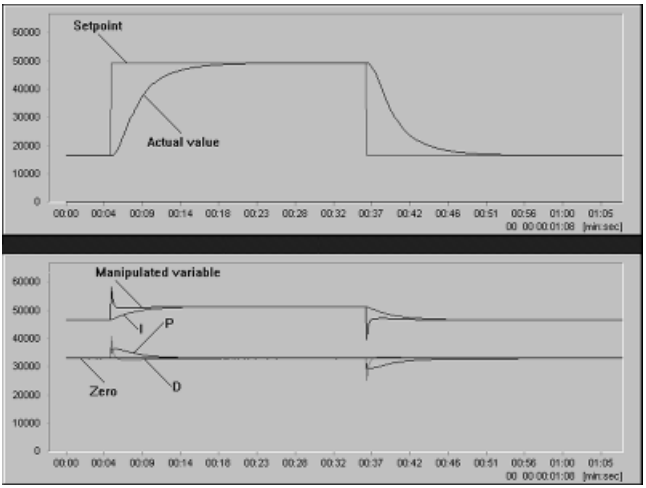


Figure 29: The proportional gain Kp is half the optimum value specified in figure 27

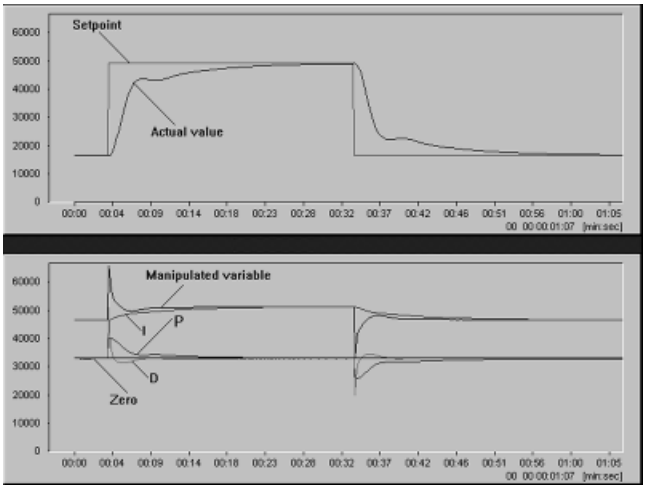


Figure 30: The derivate action time Tn is double the optimum value specified in figure 27

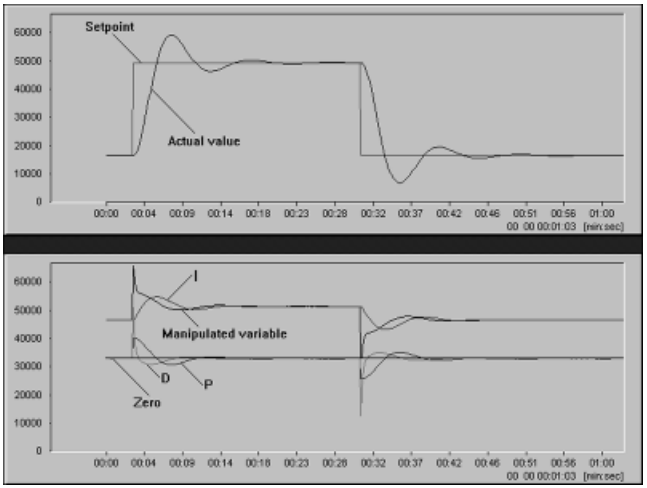


Figure 31: The derivate action time Tn is half the optimum value specified in figure 27

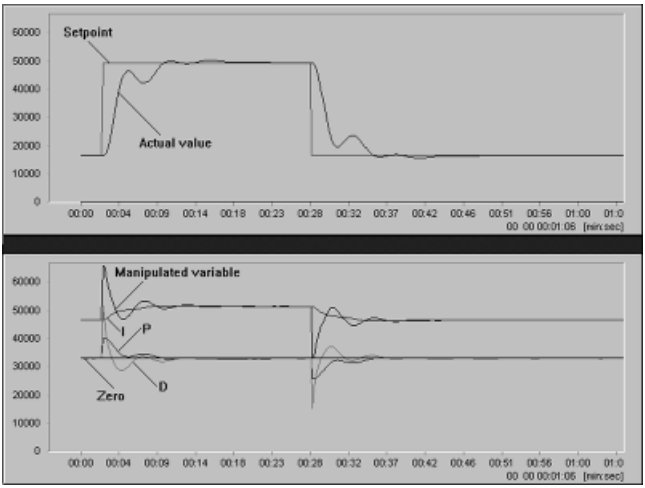


Figure 32: The derivate action time Tn is double the optimum value specified in figure 27

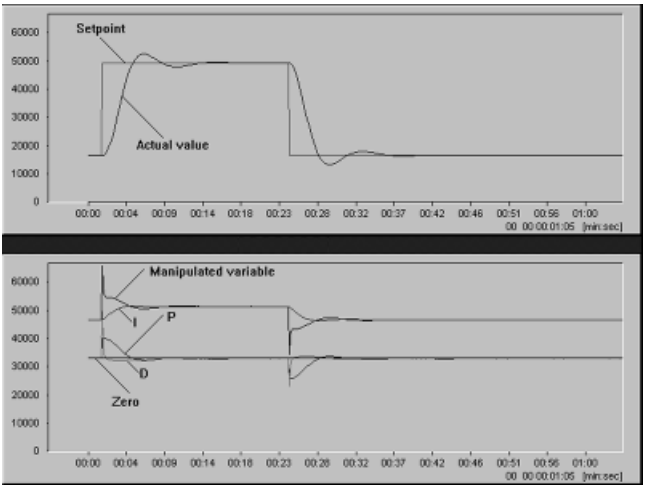


Figure 33: The derivate action time Tn is half the optimum value specified in figure 27

Comparison of PI and PID controllers

Figure 34 (control behaviour) and 35 (disturbance behaviour) compare the control behaviour of PI and PID controllers. The system used was a PT3 system with a system delay time of 20 s and a system gain of 1.0. The control results of the PID controller are much better than those of the PI controller (higher potential, especially for high order systems and large time constants). This is because the parameter values for a PID controller allow greater proportional gain Kp than for a PI controller. In addition, the D component contributes to the creation of an overall manipulated variable. The disadvantage of the PID controller is that the parameter values are more complicated.

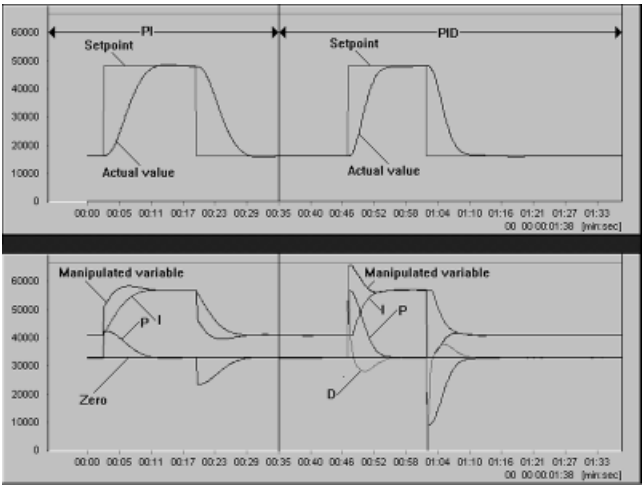


Figure 34: Comparison of the control behaviour of PI and PID controllers

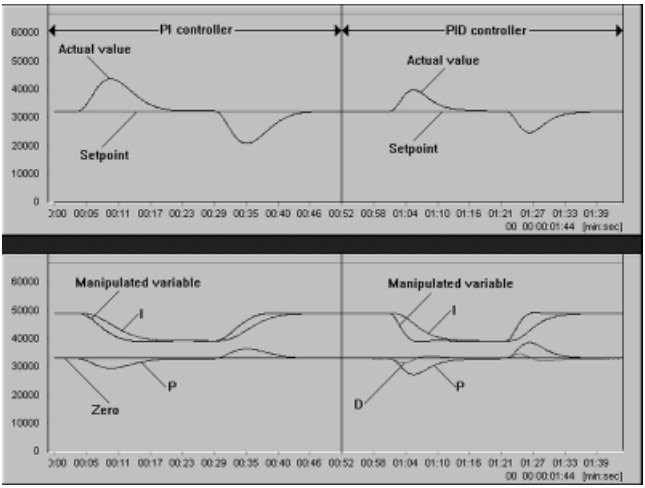


Figure 35: Comparison of the disturbance behaviour of PI and PID controllers

PID control combined with pulse duration modulation systems for several zones

When using a pulse duration modulation system, be sure to keep the PLC cycle time as short as possible. Otherwise, the system will make major errors when converting the manipulated variable.

Example:
PDM period = 10 s
Manipulated variable = 20 %
=> PDM pulse duty factor = 2 s
PLC cycle time = 200 ms
=> The imprecision of the conversion of the manipulated variable is around 10 %, because the PLC cycle time is 10 % of the PDM pulse duty factor.

If there are several control zones, minimise the cycle time for the PDM systems as follows:

- Do not call the controllers in each PLC cycle. Rather, call only one during each PLC cycle. As an example, if you have 20 zones, call each controller only during each 20th PLC cycle. For the controllers, this results in sampling times which are larger than the PLC cycle times, by an amount equal to the number of zones. In contrast to the sampling times of the PDM systems, this is insignificant.
- Call the PDM systems during each PLC cycle.

The following example uses 10 zones. It illustrates that the recommended technique reduces the PDM sampling time by a factor of 7, while only increasing the PID controller sampling time by a factor of 1.5:

- Cycle time component of PDM systems = 1 ms
- Cycle time component of PID controllers = 20 ms
- Other cycle time components = 1 ms

=> If all function blocks are called in each PLC cycle, the following cycle and sampling times will result:

- PLC cycle time = 220 ms
- Sampling time of the PDM systems = 220 ms
- Sampling time of the PID controllers = 220 ms

=> If the PID controllers are called only in each tenth PLC cycle and the PDM systems are called in each one, the following cycle and sampling times will result:

- PLC cycle time = (20 + 10 × 1 + 1) ms = 31 ms
- Sampling time of the PDM systems = 31 ms
- Sampling time of the PID controllers = 310 ms

Equidistant sampling

In order to calculate the I and D components, the function block carries out a cycle time analysis. As a result, the control behaviour is independent of PLC cycle times. If the PLC cycle times vary a great deal (e.g. intermittent starting when the PLC restarts), the I and D components might be calculated inaccurately at the transitions.

If the control behaviour must be very precise, a precision of around +/– 1 % can be attained by means of equidistant samples, or constant PLC cycle times. For this, use the function block "U_CYCS_cycle_time_setpoint_value" (→ chapter 2, page 133).

Starting behaviour and deactivating the I and D components

The sampling times (not PLC cycle time) of the I and D components are determined automatically. For this, the function block must analyse the PLC cycle times. An initial analysis can be carried out after around 120 ms. Thus, the I- and D components of the manipulated variable are created starting 120 ms after these components are activated. If this delay in adding the I or D component must be avoided, then set Tn or Tv to zero instead of deactivation (entering the zero at the corresponding BOOL input). The function block will begin as soon as the parameters change their spontaneous runtime value. If this is done, then the function block must also be called in the "idling" phase.

If the function block is not called at regular intervals, then it must be deactivated and reactivated with the corresponding BOOL input, because then the function block must be completely reset.

12-bit resolution of setpoint, actual value and manipulated variable

A resolution of 12 bits allows 4096 discrete steps (0 to 4095), which is around 41 times greater than a % resolution. If you want to convert the 12-bit values in the PLC (e.g. in % or °C), then you can use the function block "U_FR2_UINT_fractions_12Bit" for this.

The following example converts 12 bits to % with the function block "U_FR2_UINT_fractions_12Bit":

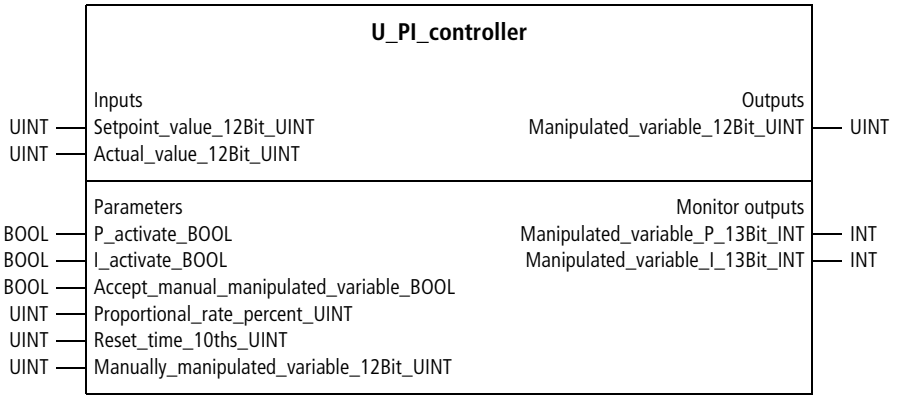
```
CAL FR2_UINT_FRACTIONS_12BIT(
  Multiplier_12Bit_UINT :=Variable_12Bit,
  Numerator_12Bit_UINT :=100,
  Denominator_12Bit_UINT :=4095,
  Product_16Bit_UINT=>Variable_Percent)
```

The following example converts % to 12 bits with the function block "U_FR2_UINT_fractions_12Bit":

```
CAL FR2_UINT_FRACTIONS_12BIT(
  Multiplier_12Bit_UINT :=Variable_Percent,
  Numerator_12Bit_UINT :=4095,
  Denominator_12Bit_UINT :=100,
  Product_16Bit_UINT=>Variable_12Bit)
```

PI-/PID controller with 12-bit standardisation

U_PI_controller
PI Controller with 12-Bit Inputs and Outputs



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Setpoint_value_12Bit_UINT	Setpoint	0 to 4095
Actual_value_12Bit_UINT	Actual value	0 to 4095
Parameters		
P_activate_BOOL	Activates the P component	0/1
I_activate_BOOL	Activates the I component	0/1
Accept_manual_manipulated_ variable_BOOL	"Soft" acceptance of the manual value	0/1
Proportional_rate_percent_UINT	Proportional gain Kp [%]	0 to 65535
Reset_time_10ths_UINT	Reset time Tn [0.1 s]	0 to 65535
Manual_manipulated_variable_ 12Bit_UINT	Manual value	0 to 4095

Designation	Significance	Value range
Outputs		
Manipulated_variable_12Bit_UINT	Manipulated variable (analogue, 12 bits)	0 to 4095
Monitor outputs		
Manipulated_variable_P_13Bit_INT	P component	–4095 to 4095
Manipulated_variable_I_13Bit_INT	I component	–4095 to 4095

Description

The components of the controller can be activated (= enabled) or deactivated separately, with the variables "P_activate_BOOL" and "I_activate_BOOL". Deactivating the I component resets the controller.

The controller's parameters are set with the standardised variables Kp [%] and Tn [0.1 s].

The controller provides the analogue output variable "Manipulated_variable_12Bit_UINT". The PI components of the manipulated variable are used for (remote) diagnosis of control behaviour. The overall manipulated variable is created by adding the individual components.

Manual operation:
The controller can be operated by hand, using the corresponding BOOL and UINT variables. If "Accept_manual_manipulated_variable_BOOL" is "1", then the controller will output to "Manipulated_variable_12Bit_UINT" the value assigned to the variable "Manual_manipulated_variable_12Bit_UINT". When "Accept_manual_manipulated_variable_BOOL" changes back to "0", then the controller accepts the manual value and continues control with this manipulated variable, without oscillations (→ "smooth acceptance of the manual value").

→ For a detailed description of how the controller works and how to use it, refer to the section on "Fundamentals and general information on PID controllers" at beginning of this chapter.

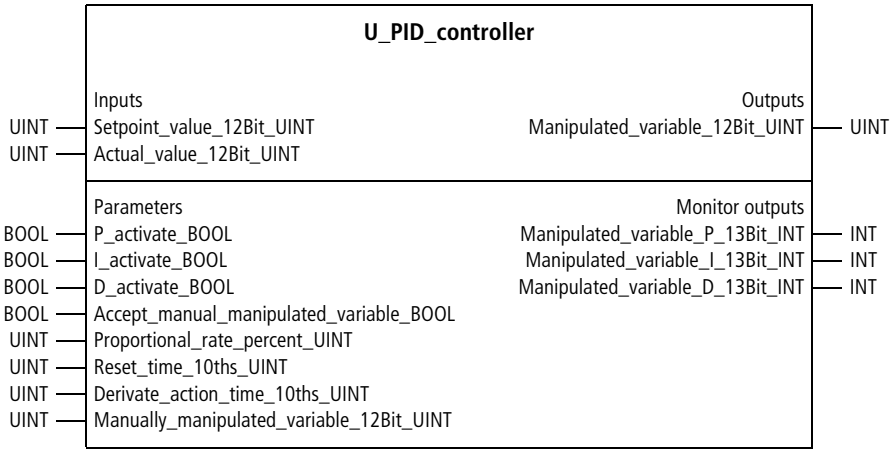
Example:
The application example "Zone1" calls a PI controller with the parameters:

- Kp = 1.2
- Tn = 30 s

Application of the function block "U_PI_controller" in the program "Zone1"

```
PROGRAM Zone1
VAR
  PI_CONTROLLER_zone1 : U_PI_CONTROLLER ;
  Setpoint_value_zone1 : UINT ;
  Actual_value_zone1 : UINT ;
  Enable_PI_CONTROLLER : BOOL ;
  Manual_operation_zone1 : BOOL ;
  Manual_manipulated_variable_zone1 : UINT ;
  Manipulated_variable_zone1 : UINT ;
  Manipulated_variable_P_zone1 : INT ;
  Manipulated_variable_I_zone1 : INT ;
END_VAR

CAL PI_CONTROLLER_zone1(
  Setpoint_value_12Bit_UINT :=Setpoint_value_zone1,
  Actual_value_12Bit_UINT :=Actual_value_zone1,
  P_activate_BOOL :=Enable_PI_CONTROLLER,
  I_activate_BOOL :=Enable_PI_CONTROLLER,
  Accept_manual_manipulated_variable_BOOL :=Manual_operation_zone1,
  Proportional_rate_percent_UINT :=120,
  Reset_time_10ths_UINT :=300,
  Manual_manipulated_variable_12Bit_UINT :=Manual_manipulated_variable_zone1,
  Manipulated_variable_12Bit_UINT=>Manipulated_variable_zone1,
  Manipulated_variable_P_13Bit_INT=>Manipulated_variable_P_zone1,
  Manipulated_variable_I_13Bit_INT=>Manipulated_variable_I_zone1)

END_PROGRAM
```

U_PID_controller
PID Controller with 12-Bit Inputs and Outputs, and Input of the Reset Time in Tenths of a Second



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Setpoint_value_12Bit_UINT	Setpoint	0 to 4095
Actual_value_12Bit_UINT	Actual value	0 to 4095
Parameters		
P_activate_BOOL	Activates the P component	0/1
I_activate_BOOL	Activates the I component	0/1
D_activate_BOOL	Activates the D component	0/1
Accept_manual_manipulated_ variable_BOOL	"Soft" acceptance of the manual value	0/1

Designation	Significance	Value range
Proportional_rate_percent_UINT	Proportional gain Kp [%]	0 to 65535
Reset_time_10ths_UINT	Reset time Tn [0.1 s]	0 to 65535
Derivate_action_time_10ths_UINT	Derivative action time Tv [0.1 s]	0 to 65535
Manual_manipulated_variable_ 12Bit_UINT	Manual value	0 to 4095
Outputs		
Manipulated_variable_12Bit_UINT	Manipulated variable (analogue, 12 Bit)	0 to 4095
Monitor outputs		
Manipulated_variable_P_13Bit_INT	P component	–4095 to 4095
Manipulated_variable_I_13Bit_INT	I component	–4095 to 4095
Manipulated_variable_D_13Bit_INT	D component	–4095 to 4095

Description

The components of the controller can be activated (= enabled) or deactivated separately, with the BOOL variables "P_activate_BOOL", "I_activate_BOOL" and "D_activate_BOOL". Deactivating the I component or D component resets the controller.

The controller's parameters are set with the standardised variables Kp [%], Tn [0.1 s] and Tv [0.1 s].

The controller provides the analogue output variable "Manipulated_variable_12Bit_UINT". For (remote) diagnosis of control behaviour, the PID components of the manipulated variable are used. The overall manipulated variable is created by adding the individual components.

Manual operation:
The controller can be operated by hand, using the corresponding BOOL and UINT variables. If "Accept_manual_manipulated_variable_BOOL" is "1", then the controller will output to "Manipulated_variable_12Bit_UINT" the value assigned to the variable "Manual_manipulated_variable_12Bit_UINT". When "Accept_manual_manipulated_variable_BOOL" changes back to "0", then the controller accepts the manual value and continues control with this manipulated variable, without oscillations (→ "smooth acceptance of the manual value").

For a detailed description of how the controller works and how to use it, refer to the section on "Fundamentals and general information on PID controllers" at beginning of this chapter.

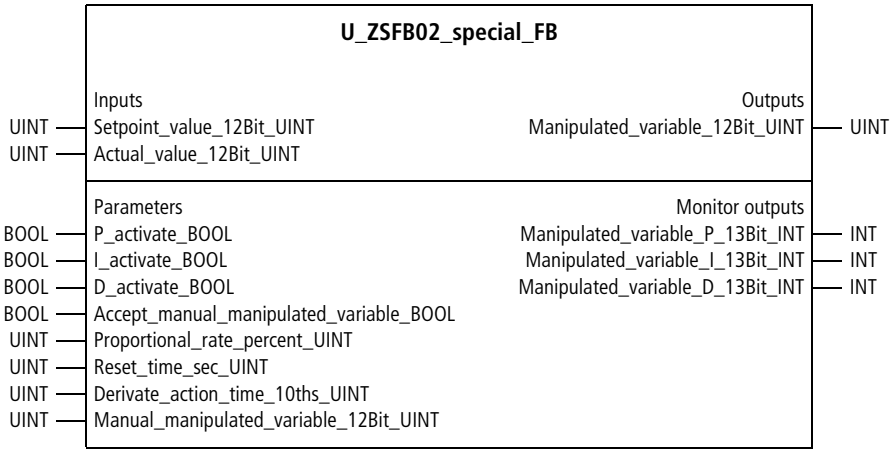
Example:
The application example "Zone2" calls a PID controller with the parameters:

- Proportional gain Kp = 1:2
- Reset time Tn = 30 s
- Derivative action time Tv = 3 s

Application of the function block "U_PID_controller" in the program "Zone2"

```
PROGRAM Zone2
VAR
  PID_CONTROLLER_zone2 : U_PID_CONTROLLER ;
  Setpoint_value_zone2 : UINT ;
  Actual_value_zone2 : UINT ;
  Enable_PID_CONTROLLER : BOOL ;
  Manual_operation_zone2 : BOOL ;
  Manual_manipulated_variable_zone2 : UINT ;
  Manipulated_variable_zone2 : UINT ;
  Manipulated_variable_P_zone2 : INT ;
  Manipulated_variable_I_zone2 : INT ;
  Manipulated_variable_D_zone2 : INT ;
END_VAR

CAL PID_CONTROLLER_zone2(
  Setpoint_value_12Bit_UINT :=Setpoint_value_zone2,
  Actual_value_12Bit_UINT :=Actual_value_zone2,
  P_activate_BOOL :=Enable_PID_CONTROLLER,
  I_activate_BOOL :=Enable_PID_CONTROLLER,
  D_activate_BOOL :=Enable_PID_CONTROLLER,
  Accept_manual_manipulated_variable_BOOL :=Manual_operation_zone2,
  Proportional_rate_percent_UINT :=120,
  Reset_time_10ths_UINT :=300,
  Derivate_action_time_10ths_UINT :=30,
  Manual_manipulated_variable_12Bit_UINT :=Manual_manipulated_variable_zone2,
  Manipulated_variable_12Bit_UINT=>Manipulated_variable_zone2,
  Manipulated_variable_P_13Bit_INT=>Manipulated_variable_P_zone2,
  Manipulated_variable_I_13Bit_INT=>Manipulated_variable_I_zone2,
  Manipulated_variable_D_13Bit_INT=>Manipulated_variable_D_zone2)

END_PROGRAM
```

U_ZSFB02_special_FB
PID Controller with 12-bit Inputs and Outputs, and Input of the Reset Time in Seconds



Function block prototype

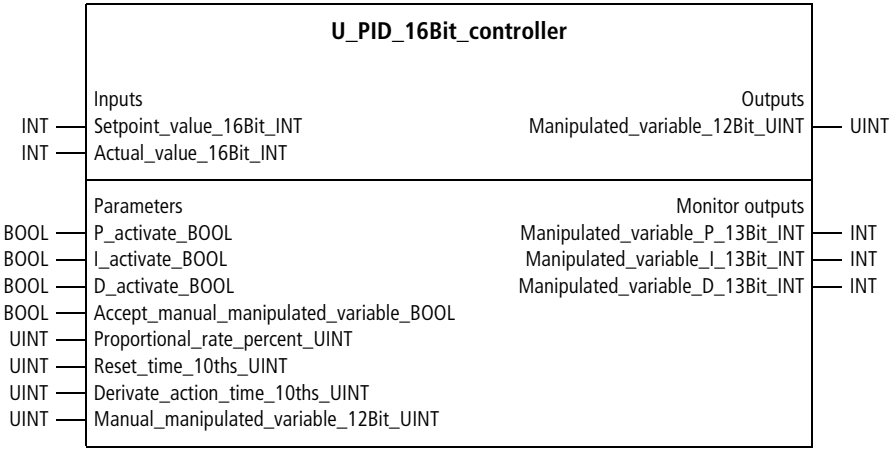
Meaning of the operands

Designation	Significance	Value range
Inputs		
Setpoint_value_12Bit_UINT	Setpoint	0 to 4095
Actual_value_12Bit_UINT	Actual value	0 to 4095
Parameters		
P_activate_BOOL	Activates the P component	0/1
I_activate_BOOL	Activates the I component	0/1
D_activate_BOOL	Activates the D component	0/1
Accept_manual_manipulated_ variable_BOOL	"Soft" acceptance of the manual value	0/1
Proportional_rate_percent__UINT	Proportional gain Kp [%]	0 to 65535
Reset_time_sec_UINT	Reset time Tn [s]	0 to 65535
Derivate_action_time_10ths_UINT	Derivative action time Tv [0.1 s]	0 to 65535
Manual_manipulated_variable_ 12Bit_UINT	Manual value	0 to 4095
Outputs		
Manipulated_variable_12Bit_UINT	Manipulated variable (analogue, 12 Bit)	0 to 4095
Monitor outputs		
Manipulated_variable_P_13Bit_INT	P component	–4095 to 4095
Manipulated_variable_I_13Bit_INT	I component	–4095 to 4095
Manipulated_variable_D_13Bit_INT	D component	–4095 to 4095

Description

See the function block "U_PID_controller". The only difference to the function block "U_PID_controller" is that the reset time Tn can be specified in seconds. So the longest time which can be entered is 18.2 h.

U_PID_16Bit_controller
PID controller with 16-bit inputs



Function block prototype

Meaning of the operands

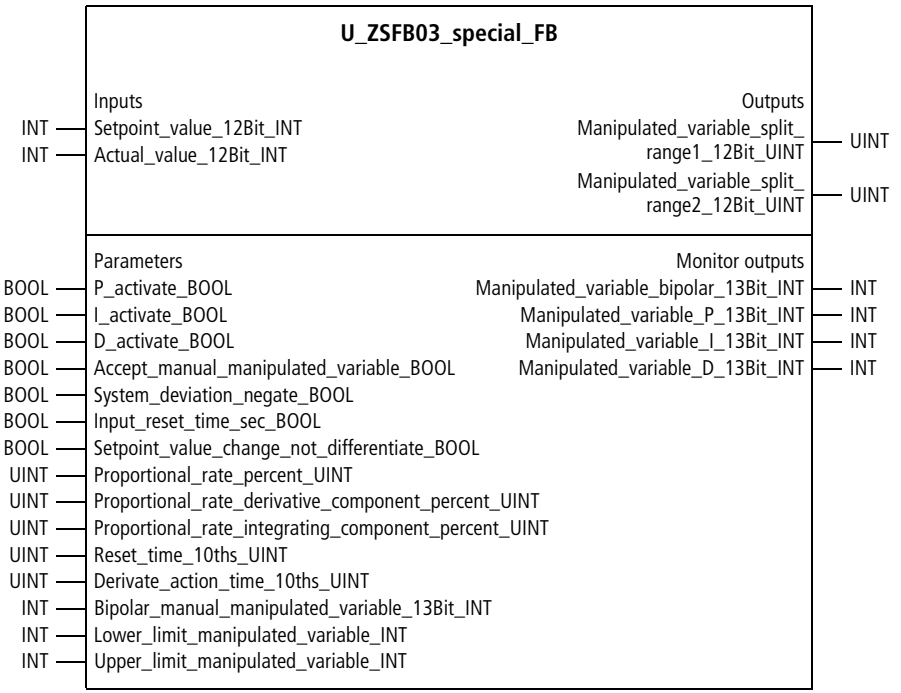
Designation	Significance	Value range
Inputs		
Setpoint_value_16Bit_INT	Setpoint value	–32768 to 32767
Actual_value_16Bit_INT	Actual value	–32768 to 32767
Parameters		
P_activate_BOOL	Activates the P component	0/1
I_activate_BOOL	Activates the I component	0/1
D_activate_BOOL	Activates the D component	0/1
Accept_manual_manipulated_ variable_BOOL	"Soft" acceptance of the manual value	0/1
Proportional_rate_percent_UINT	Proportional gain [%]	0 to 65535
Reset_time_10ths_UINT	Reset time [0.1 s]	0 to 65535
Derivate_action_time_10ths_UINT	Derivate action time [0.1 s]	0 to 65535
Manual_manipulated_variable_ 12Bit_UINT	Manual value	0 to 4095
Outputs		
Manipulated_variable_12Bit_UINT	Manipulated variable (analogue, 12 Bit)	0 to 4095
Monitor outputs		
Manipulated_variable_P_13Bit_INT	P component	–4095 to 4095
Manipulated_variable_I_13Bit_INT	I component	–4095 to 4095
Manipulated_variable_D_13Bit_INT	D component	–4095 to 4095

Description

See the function block "U_PID_controller". The only difference to the function block "U_PID_controller" is that this function block has 16 Bit-input variables (instead of 12 Bit).

PID controller with parameter options

U_ZSFB03_special_FB
PID Controller with 12-bit Inputs and "Parameter Options"



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Setpoint_value_12Bit_INT	Setpoint	–4095 to 4095
Actual_value_12Bit_INT	Actual value	–4095 to 4095
Parameters		
P_activate_BOOL	Activates the P component	0/1
I_activate_BOOL	Activates the I component	0/1

Designation	Significance	Value range
D_activate_BOOL	Activates the D component	0/1
Accept_manual_manipulated_ variable_BOOL	"Soft" acceptance of the manual value	0/1
System_deviation_negate_BOOL	Negates the system deviation	0/1
Input_reset_time_sec_BOOL	Enter reset time in 0.1 s or s	0/1
Setpoint_value_change_not_ differentiate_BOOL	Differentiation of spontaneous setpoint value changes is suppressed	0/1
Proportional_rate_percent__UINT	Proportional gain Kp [%]	0 to 65535
Proportional_rate_derivative_ component_percent_UINT	Proportional gain Kd [%]	0 to 65535
Proportional_rate_integrating_ component_percent_UINT	Proportional gain Ki [%]	0 to 65535
Reset_time_10ths_UINT	Reset time Tn [0.1 s]	0 to 65535
Derivate_action_time_10ths_UINT	Derivative action time Tv [0.1 s]	0 to 65535
Bipolar_manual_manipulated_ variable_13Bit_INT	Manual value	–4095 to 4095
Lower_limit_manipulated_ variable_INT	Lower limit of the manipulated variable	–4095 to 4095
Upper_limit_manipulated_ variable_INT	Upper limit of the manipulated variable	–4095 to 4095
Outputs		
Manipulated_variable_split_ range1_12Bit_UINT	Analogue manipulated variable for positive bipolar manipulated variable	0 to 4095
Manipulated_variable_split_ range2_12Bit_UINT	Analogue manipulated variable for negative bipolar manipulated variable	0 to 4095
Monitor outputs		
Manipulated_variable_bipolar_ 13Bit_INT	Bipolar manipulated variable	–4095 to 4095
Manipulated_variable_P_13Bit_INT	P component	–4095 to 4095
Manipulated_variable_I_13Bit_INT	I component	–4095 to 4095
Manipulated_variable_D_13Bit_INT	D component	–4095 to 4095

Description

The components of the controller can be activated (= enabled) or deactivated separately, with the BOOL variables "P_activate_BOOL", "I_activate_BOOL" and "D_activate_BOOL". Deactivating the I component or D component resets the controller.

"System_deviation_negate_BOOL=1" toggles the effect of the P-, I and D components. So a positive system deviation results in a negative P component, etc. With "Input_reset_time_sec_BOOL", the reset time can be entered either in [01 s] or in [s].

The D component can be calculated either in terms of the system deviation (= setpoint – actual value, including the setpoint) or in terms of the actual value, by assigning either "0" (system deviation) or "1" (actual value) to the variable "Setpoint_value_change_not_differentiate_BOOL". The main difference is that spontaneous changes of the setpoint do not result in a spontaneous increase in the D component.

The controller is dimensioned with the variables Kp [%], Kd [%], Ki [%], Tn [0.1 s] and Tv [0.1 s]. Within the range of [–4095 to 4095], the limited range for the manipulated variable can be specified with the variables "Lower_limit_manipulated_variable_INT" and "Upper_limit_manipulated_variable_INT" (for [0 to 4095], the manipulated variable is limited as for the function block "U_PID_12Bit_controller").

The controller provides the analog output variables "Manipulated_variable_split_range1_12Bit_UINT" and "Manipulated_variable_split_range2_12Bit_UINT". As for split-range controllers, these manipulated variables can be used for control systems with functions such as heating and cooling. The bipolar manipulated variable is divided as follows:

- Positive values for the bipolar manipulated variable => "Manipulated_variable_split_range1_12Bit_UINT"
- Negative values for the bipolar manipulated variable => "Manipulated_variable_split_range2_12Bit_UINT" (= magnitude of the negative manipulated variable).

For (remote) diagnosis of control behaviour, the PID components of the manipulated variable are used. The overall manipulated variable is created by adding the individual components.

Manual operation:
The controller can be operated by hand, using the corresponding BOOL and UINT variables. If "Accept_manual_manipulated_variable_BOOL" is "1", then the controller will output to "Manipulated_variable_12Bit_UINT" the value assigned to the variable "Manual_manipulated_variable_12Bit_UINT". The variable is divided into positive and negative values, as described above. When "Accept_manual_manipulated_variable_BOOL" changes back to "0", then the controller accepts the manual value and continues control with this manipulated variable, without oscillations (→ "smooth acceptance of the manual value").

→ Figure 51 in chapter 5 on the function block "U_PDM_splitrange" describes the interaction between split-range controllers and PDM.

→ For a detailed description of how the controller works and how to use it, refer to the section on "Fundamentals and general information on PID controllers" at beginning of this chapter.

Example:
The application example "Zone2" calls a PID controller with the following characteristics:

- Proportional gain Kp = 1.2
- Proportional gain Kd = 1.2
- Proportional gain Ki = 1.2
  (the standardised behaviour applies for the reset time and derivative action time, because the proportional gain factors of the I and D components agree with that of the P component).

Reset time Tn = 300 s
(due to "Input_reset_time_sec_BOOL:=1";
"Input_reset_time_sec_BOOL:=0" would result in:
reset time Tn = 30 s).

Setpoint changes lead to spontaneous changes of the D component, due to "Setpoint_value_change_ not_differentiate_BOOL:=0".

The relationship between the manipulated variable and the actual value is "normal" (large values for the manipulated variable result in large actual values), so "System_deviation_negate_BOOL:=0" was entered.

The manipulated variable is limited to 12 bits.

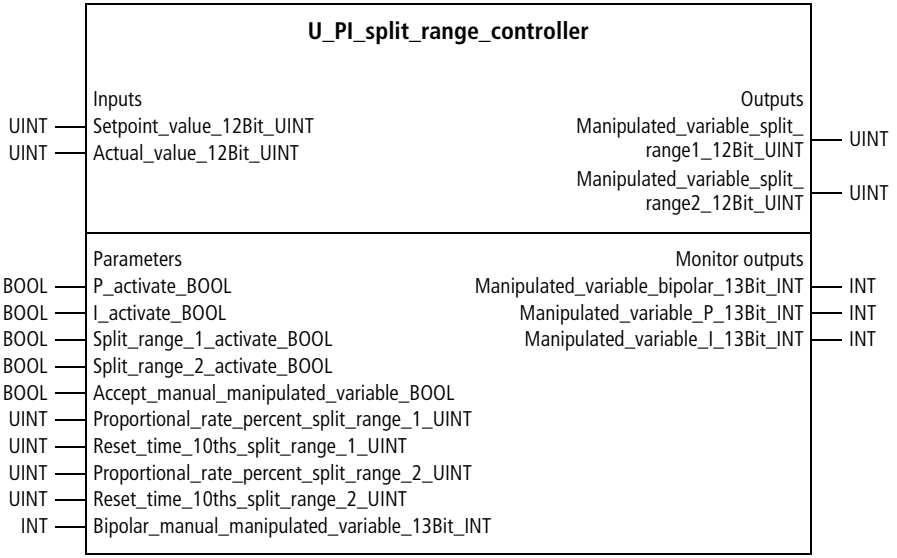
Application of the function block "U_ZSFB03_special_FB" in the program "Zone2"

```
PROGRAM Zone2
VAR
  PID_CONTROLLER_zone2 : U_ZSFB03_SPECIAL_FB ;
  Setpoint_value_zone2 : INT ;
  Actual_value_zone2 : INT ;
  Enable_PID_CONTROLLER : BOOL ;
  Manual_operation_zone2 : BOOL ;
  Manual_manipulated_variable_zone2 : INT ;
  Manipulated_variable_zone2 : UINT ;
  Manipulated_variable_heat_zone2 : UINT ;
  Manipulated_variable_cool_zone2 : UINT ;
  Manipulated_variable_bipolar_zone2 : INT ;
  Manipulated_variable_P_zone2 : INT ;
  Manipulated_variable_I_zone2 : INT ;
  Manipulated_variable_D_zone2 : INT ;
END_VAR

CAL PID_CONTROLLER_zone2(
  Setpoint_value_12Bit_INT :=Setpoint_value_zone2,
  Actual_value_12Bit_INT :=Actual_value_zone2,
  P_activate_BOOL :=Enable_PID_CONTROLLER,
  I_activate_BOOL :=Enable_PID_CONTROLLER,
  D_activate_BOOL :=Enable_PID_CONTROLLER,
  Accept_manual_manipulated_variable_BOOL :=Manual_operation_zone2,
  System_deviation_negate_BOOL :=0,
  Input_reset_time_sec_BOOL :=1,
  Setpoint_value_change_not_differentiate_BOOL :=0,
```

```
  Proportional_rate_percent_UINT :=120,
  Proportional_rate_derivative_component_percent_UINT :=120,
  Proportional_rate_integrating_component_percent_UINT :=120,
  Reset_time_10ths_or_s_UINT :=300,
  Derivate_action_time_10ths_UINT :=30,
  Manual_manipulated_variable_13Bit_INT :=Manual_manipulated_variable_zone2,
  Lower_limit_manipulated_variable_INT :=0,
  Upper_limit_manipulated_variable_INT :=4095,
  Manipulated_variable_split_range1_12Bit_UINT=>Manipulated_variable_heat_zone2,
  Manipulated_variable_split_range2_12Bit_UINT=>Manipulated_variable_cool_zone2,
  Manipulated_variable_bipolar_13Bit_INT=>Manipulated_variable_bipolar_zone2,
  Manipulated_variable_P_13Bit_INT=>Manipulated_variable_P_zone2,
  Manipulated_variable_I_13Bit_INT=>Manipulated_variable_I_zone2,
  Manipulated_variable_D_13Bit_INT=>Manipulated_variable_D_zone2)
END_PROGRAM
```

PI split range controller

U_PI_split_range_controller
PI Split Range Controller for "Heating" and "Cooling"



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Setpoint_value_12Bit_UINT	Setpoint	0 to 4095
Actual_value_12Bit_UINT	Actual value	0 to 4095
Parameters		
P_activate_BOOL	Activates the P component	0/1
I_activate_BOOL	Activates the I component	0/1
Split_range_1_activate_BOOL	Activates split range 1 (heating)	0/1
Split_range_2_activate_BOOL	Activates split range 2 (cooling)	0/1

Designation	Significance	Value range
Accept_manual_manipulated_ variable_BOOL	Smooth acceptance of the manual value	0/1
Proportional_rate_percent_split_ range_UINT	Proportional gain Kp split range 1 [%]	0/1
Reset_time_10ths_split_range_ 1_UINT	Reset time Tn split range 1 [0.1 s]	0/1
Proportional_rate_percent_split_ range_2_UINT	Proportional gain Kp split range 2 [%]	0 to 65535
Reset_time_10ths_split_range_ 2_UINT	Reset time Tn split range 2 [0.1 s]	0 to 65535
Bipolar_manual_manipulated_ variable_13Bit_INT	Manual value	0 to 65535
Outputs		
Manipulated_variable_split_ range1_12Bit_UINT	Manipulated variable split range 1 (heating)	0 to 4095
Manipulated_variable_split_ range2_12Bit_UINT	Manipulated variable split range 1 (cooling)	0 to 4095
Monitor outputs		
Manipulated_variable_bipolar_ 13Bit_INT	Bipolar manipulated variable	–4095 to 4095
Manipulated_variable_P_13Bit_INT	P component	–4095 to 4095
Manipulated_variable_I_13Bit_INT	I component	–4095 to 4095

Description

The PI split-range controller is suitable for control systems whose actual value can be influenced by two manipulated variables, such as a temperatore control system with heating and cooling equipment (→ fig. 36). The components of the controller can be activated (= enabled) or deactivated separately, with the corresponding variables. Deactivating the I component resets the controller.

The controller is dimensioned separately for split ranges 1 and 2, using the standardised variables Kp [%] and Tv [0.1 s].

The controller provides the analogue output variables "Manipulated_variable_split_range1_12Bit_UINT" and "Manipulated_variable_split_range2_12Bit_UINT". The PI components of the manipulated variable are used for (remote) diagnosis of control behaviour. The overall manipulated variable is created by adding the individual components.

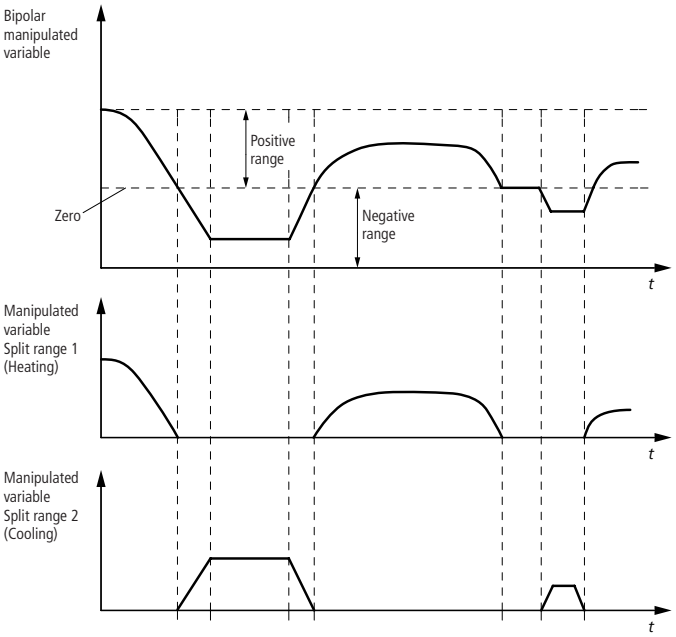


Figure 36: The bipolar (positive/negative) manipulated variable is split up according to the sign. For positive values, "manipulated variable-split range1" outputs a value. For negative values, "manipulated variable-split range2" outputs a value.

Manual operation:
If "Accept_manual_manipulated_variable_BOOL" is "1", then the controller will output to "Manipulated_variable_12Bit_UINT" the value assigned to the variable "Manual_manipulated_variable_12Bit_UINT". When "Accept_manual_manipulated_variable_BOOL" changes back to "0", then the controller accepts the manual value and continues control with this manipulated variable, without oscillations (→ "smooth acceptance of the manual value").

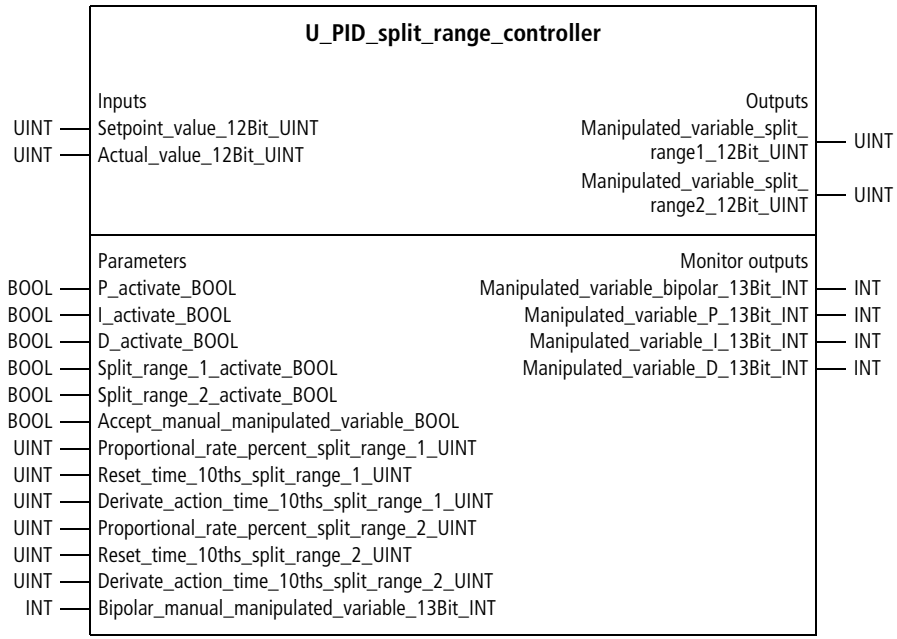
Example:
In the application example, enabling P, I, split range 1 (heating) and split range 2 (cooling) is linked with a BOOL variable. If split range 2 is not enabled, then the controller's function is downgraded to the simple PI controller.

Application of the function block "U_PI_split_range_controller" in the program "Zone2"

```
PROGRAM Zone2
VAR
  PI_SPLIT_RANGE_CONTROLLER_zone2 : U_PI_SPLIT_RANGE_CONTROLLER ;
  Setpoint_value_zone2 : UINT ;
  Actual_value_zone2 : UINT ;
  Enable_PI_SPLIT_RANGE_CONTROLLER : BOOL ;
  Manual_operation_zone2 : BOOL ;
  Manual_manipulated_variable_zone2 : INT ;
  Manipulated_variable_zone2 : UINT ;
  Manipulated_variable_heat_zone2 : UINT ;
  Manipulated_variable_cool_zone2 : UINT ;
  Manipulated_variable_bipolar_zone2 : INT ;
  Manipulated_variable_P_zone2 : INT ;
  Manipulated_variable_I_zone2 : INT ;
END_VAR
```

```
CAL PI_SPLIT_RANGE_CONTROLLER_zone2(
  Setpoint_value_12Bit_UINT :=Setpoint_value_zone2,
  Actual_value_12Bit_UINT :=Actual_value_zone2,
  P_activate_BOOL :=Enable_PI_SPLIT_RANGE_CONTROLLER,
  I_activate_BOOL :=Enable_PI_SPLIT_RANGE_CONTROLLER,
  Split_range1_activate_BOOL :=Enable_PI_SPLIT_RANGE_CONTROLLER,
  Split_range2_activate_BOOL :=Enable_PI_SPLIT_RANGE_CONTROLLER,
  Accept_manual_manipulated_variable_BOOL :=Manual_operation_zone2,
  Proportional_rate_split_range1_percent_UINT :=120,
  Reset_time_split_range1_10ths_UINT :=300,
  Proportional_rate_split_range2_percent_UINT :=300,
  Reset_time_split_range2_10ths_UINT :=70,
  Manipulated_variable_split_range1_12Bit_UINT=>Manipulated_variable_heat_zone2,
  Manipulated_variable_split_range2_12Bit_UINT=>Manipulated_variable_cool_zone2,
  Manipulated_variable_bipolar_13Bit_INT=>Manipulated_variable_bipolar_zone2,
  Manipulated_variable_P_13Bit_INT=>Manipulated_variable_P_zone2,
  Manipulated_variable_I_13Bit_INT=>Manipulated_variable_I_zone2)

END_PROGRAM
```

U_PID_split_range_controller
PID Split-range Controller for Heating and Cooling



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Setpoint_value_12Bit_UINT	Setpoint	0 to 4095
Actual_value_12Bit_UINT	Actual value	0 to 4095
Parameters		
P_activate_BOOL	Activates the P component	0/1
I_activate_BOOL	Activates the I component	0/1
D_activate_BOOL	Activates the D component	0/1
Split_range_1_activate_BOOL	Activates split range 1 (heating)	0/1
Split_range_2_activate_BOOL	Activates split range 2 (cooling)	0/1
Accept_manual_manipulated_ variable_BOOL	Smooth acceptance of the manual value	0/1
Proportional_rate_percent_split_ range_1_UINT	Proportional gain Kp split range 1 [%]	0 to 65535
Reset_time_10ths_split_range_ 1_UINT	Reset time Tn split range 1 [0.1 s]	0 to 65535
Derivate_action_time_10ths_split_ range_1_UINT	Der. action time Tv split range 1 [0.1 s]	0 to 65535
Proportional_rate_percent_split_ range_2_UINT	Proportional gain Kp split range 2 [%]	0 to 65535
Reset_time_10ths_split_range_ 2_UINT	Reset time Tn split range 2 [0.1 s]	0 to 65535
Derivate_action_time_10ths_split_ range_2_UINT	Der. action time Tv split range 2 [0.1 s]	0 to 65535
Bipolar_manual_manipulated_ variable_13Bit_INT	Manual value	–4095 to 4095
Outputs		
Manipulated_variable_split_range1_ 12Bit_UINT	Manipulated variable split range 1 (heating)	0 to 4095
Manipulated_variable_split_range2_ 12Bit_UINT	Manipulated variable split range 1 (cooling)	0 to 4095

Designation	Significance	Value range
Monitor outputs		
Manipulated_variable_bipolar_ 13Bit_INT	Bipolar manipulated variable	–4095 to 4095
Manipulated_variable_P_13Bit_INT	P component	–4095 to 4095
Manipulated_variable_I_13Bit_INT	I component	–4095 to 4095
Manipulated_variable_D_13Bit_INT	D component	–4095 to 4095

Description

The PID split-range controller is suitable for control systems whose actual value can be influenced by two manipulated variables, such as a temperature control system with heating and cooling equipment (→ fig. 37). The components of the controller can be activated (= enabled) or deactivated separately, with the corresponding variables. Deactivating the I and D components resets the controller.

The controller is dimensioned separately for split ranges 1 and 2, using the standardised variables Kp [%], Tn [0.1 s] and Tv [0.1 s].

The controller provides the analogue output variables "Manipulated_variable_split_range1_12Bit_UINT" and "Manipulated_variable_split_range2_12Bit_UINT". The PID components of the manipulated variable are used for (remote) diagnosis of control behaviour. The overall manipulated variable is created by adding the individual components.

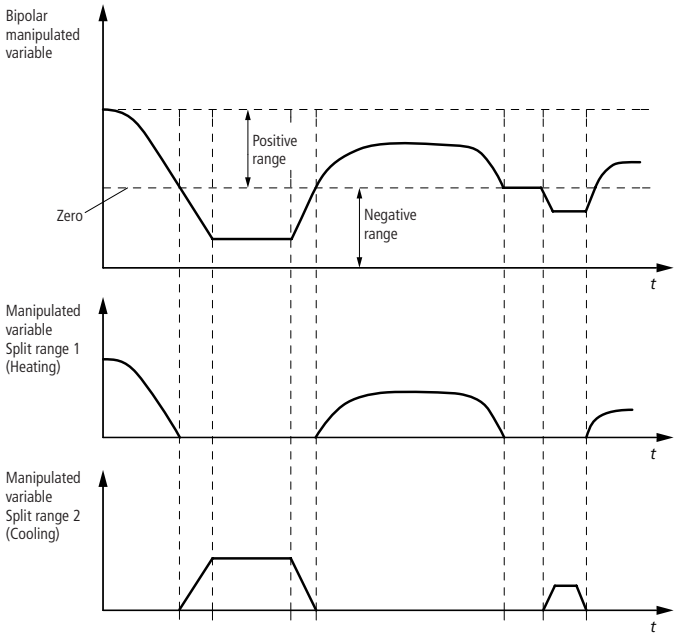


Figure 37: The bipolar (positive/negative) manipulated variable is split up according to the sign. For positive values, "manipulated variable-split range1" outputs a value. For negative values, "manipulated variable-split range2" outputs a value.

Manual operation:
If "Accept_manual_manipulated_variable_BOOL" is "1", then the controller will output to "Manipulated_variable_ 12Bit_UINT" the value assigned to the variable "Manual_manipulated_variable_12Bit_UINT". When "Accept_manual_manipulated_variable_BOOL" changes back to "0", then the controller accepts the manual value and continues control with this manipulated variable, without oscillations (→ "smooth acceptance of the manual value").

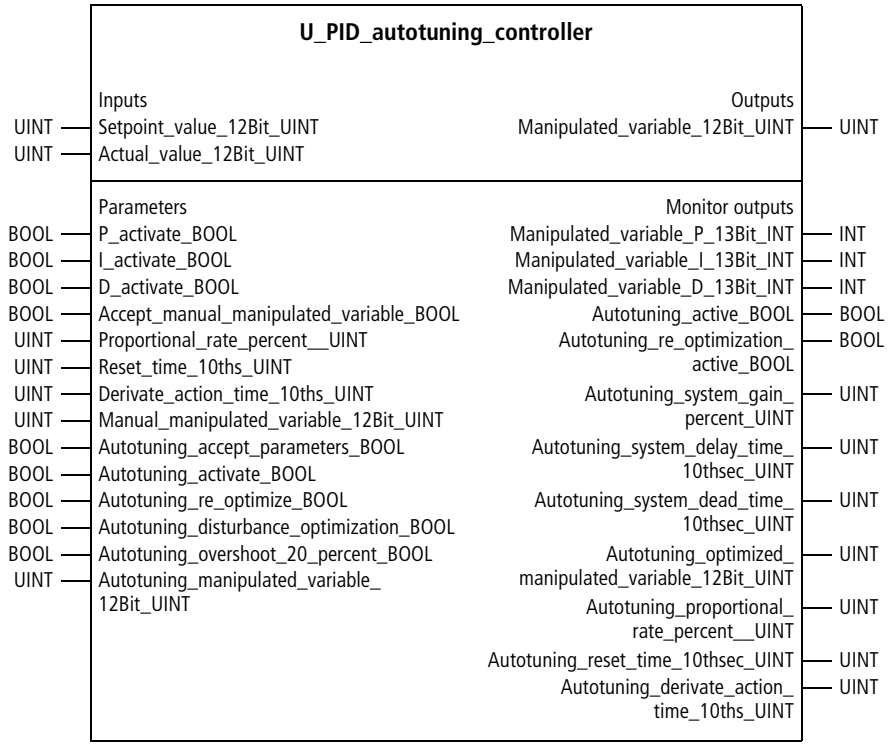
Example:
In the application example, enabling P-, I and D components, split range 1 (heating) and split range 2 (cooling) is linked with a BOOL variable. If split range 2 is not enabled, then the controller's function is downgraded to the simple PID controller. If the D component is not enabled, the result is a PI split range controller.

Application of the function block "U_PID_split_range_controller" in the program "Zone2"

```
PROGRAM Zone2
VAR
  PID_SPLIT_RANGE_CONTROLLER_zone2 : U_PID_SPLIT_RANGE_CONTROLLER ;
  Setpoint_value_zone2 : UINT ;
  Actual_value_zone2 : UINT ;
  Enable_PID_SPLIT_RANGE_CONTROLLER : BOOL ;
  Manual_operation_zone2 : BOOL ;
  Manual_manipulated_variable_zone2 : INT ;
  Manipulated_variable_zone2 : UINT ;
  Manipulated_variable_heat_zone2 : UINT ;
  Manipulated_variable_cool_zone2 : UINT ;
  Manipulated_variable_bipolar_zone2 : INT ;
  Manipulated_variable_P_zone2 : INT ;
  Manipulated_variable_I_zone2 : INT ;
  Manipulated_variable_D_zone2 : INT ;
END_VAR
```

```
CAL PID_SPLIT_RANGE_CONTROLLER_zone2(
  Setpoint_value_12Bit_UINT :=Setpoint_value_zone2,
  Actual_value_12Bit_UINT :=Actual_value_zone2,
  P_activate_BOOL :=Enable_PID_SPLIT_RANGE_CONTROLLER,
  I_activate_BOOL :=Enable_PID_SPLIT_RANGE_CONTROLLER,
  D_activate_BOOL :=Enable_PID_SPLIT_RANGE_CONTROLLER,
  Split_range1_activate_BOOL:=Enable_PID_SPLIT_RANGE_CONTROLLER,
  Split_range2_activate_BOOL :=Enable_PID_SPLIT_RANGE_CONTROLLER,
  Accept_manual_manipulated_variable_BOOL :=Manual_operation_zone2,
  Proportional_rate_split_range1_percent_UINT :=120,
  Reset_time_split_range1_10ths_UINT :=300,
  Derivate_action_time_split_range1_10ths_UINT :=30,
  Proportional_rate_split_range2_percent_UINT :=300,
  Reset_time_split_range2_10ths_UINT :=70,
  Derivate_action_time_split_range2_10ths_UINT :=0,
  Manual_manipulated_variable_bipolar_13Bit_INT :=Manual_manipulated_variable_zone2,
  Manipulated_variable_split_range1_12Bit_UINT=>Manipulated_variable_heat_zone2,
  Manipulated_variable_split_range2_12Bit_UINT=>Manipulated_variable_cool_zone2,
  Manipulated_variable_bipolar_13Bit_INT=>Manipulated_variable_bipolar_zone2,
  Manipulated_variable_P_13Bit_INT=>Manipulated_variable_P_zone2,
  Manipulated_variable_I_13Bit_INT=>Manipulated_variable_I_zone2,
  Manipulated_variable_D_13Bit_INT=>Manipulated_variable_D_zone2)
END_PROGRAM
```

PID autotuning controller U_PID_autotuning_controller
PID autotuning controller



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Setpoint_value_12Bit_UINT	Setpoint	0 to 4095
Actual_value_12Bit_UINT	Actual value	0 to 4095
Parameters		
P_activate_BOOL	Activates the P component	0/1
I_activate_BOOL	Activates the I component	0/1
D_activate_BOOL	Activates the D component	0/1
Accept_manual_manipulated_ variable_BOOL	"Soft" acceptance of the manual value	0/1
Proportional_rate_percent__UINT	Proportional gain Kp [%]	0 to 65535
Reset_time_10ths_UINT	Reset time Tn [0.1 s]	0 to 65535
Derivate_action_time_10ths_UINT	Derivative action time Tv [0.1 s]	0 to 65535
Manual_manipulated_variable_ 12Bit_UINT	Manual value	0 to 4095
Autotuning_accept_parameters_BOOL	Accept the autotuning parameters	0/1
Autotuning_activate_BOOL	Activates autotuning	0/1
Autotuning_re_optimize_BOOL	re-optimization	0/1
Autotuning_disturbance_ optimization_BOOL	Control optimization/disturbance optimization	0/1
Autotuning_overshoot_20_ percent_BOOL	Aperiodic boundary case/ 20 % overshoot	0/1
Autotuning_manipulated_variable_ 12Bit_UINT	Manipulated variable, for which autotuning is carried out	0 to 4095
Outputs		
Manipulated_variable_12Bit_UINT	Manipulated variable (analogue, 12 Bit)	0 to 4095

Designation	Significance	Value range
Monitor outputs		
Manipulated_variable_P_13Bit_INT	P component	–4095 to 4095
Manipulated_variable_I_13Bit_INT	I component	–4095 to 4095
Manipulated_variable_D_13Bit_INT	D component	–4095 to 4095
Autotuning_active_BOOL	Message: Autotuning active	0/1
Autotuning_re_optimization_ active_BOOL	Message: re-optimization active	0/1
Autotuning_system_gain_percent_UINT	System variable Ks [%]	0 to 65535
Autotuning_system_delay_time_ 10thsec_UINT	System variable Tg [0.1 s]	0 to 65535
Autotuning_system_dead_time_ 10thsec_UINT	System variable Tu [0.1 s]	0 to 65535
Autotuning_optimized_manipulated_ variable_UINT	Manipulated variable with which autotuning leads to an optimum result for this system	0 to 4095
Autotuning_proportional_rate_ percent_UINT	Autotuning parameter Kp [%]	0 to 65535
Autotuning_reset_time_10thsec_UINT	Autotuning parameter Tn [0.1 s]	0 to 65535
Autotuning_derivate_action_time_ 10ths_UINT	Autotuning parameter Tv [0.1 s]	0 to 65535

Description

Autotuning is suitable for PTn systems, such as PT3 or PT5 systems, → fig. 21. At the beginning of autotuning, a manipulated variable with a spontaneous value change is output (→ fig. 38, Autotuning, Phase 1). The system's response is then evaluated according to the tangent to the reversing point technique. The PID controller is preset such that the actual value reaches the setpoint (→ fig. 38, Autotuning, Phase 2). The parameters of the PID controller are then set for good, and autotuning is concluded.

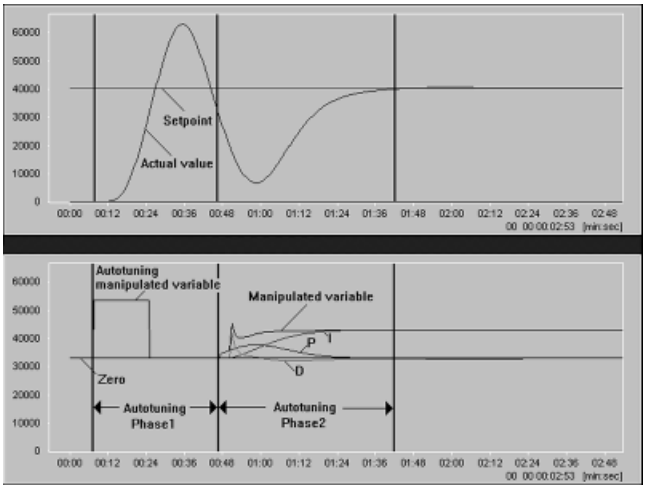


Figure 38: PID autotuning, showing the functions for the actual value and manipulated variable. During Phase 1, a spontaneous change of a manipulated variable is evaluated (autotuning manipulated variable). In Phase 2, with a preset PID controller the actual value is adjusted to the setpoint.

Quick reference guide (page 265 to 270)

Caution!
Never let autotuning run without supervision. Operating errors of the autotuning controller might make manual intervention necessary, to avoid damage to your system or machine.

To start autotuning, carry out the following steps in sequence:

Actual value and setpoint
Make sure the actual value of your control system (correct zone?) was transferred to the variable "Actual_value_12Bit_UINT".

Make sure the setpoint is set for which autotuning is to determine the optimum PID controller setting.

Set BOOL variables to zero
Set all BOOL variables of the autotuning controller to zero, in particular:

- P_activate_BOOL=0
- I_activate_BOOL=0
- D_activate_BOOL=0
- Accept_manual_manipulated_variable_BOOL=0
- Autotuning_activate_BOOL=0
- Autotuning_re_optimize_BOOL=0

Ambient value check
When autotuning begins, the actual value must equal the ambient value in order for autotuning to determine meaningful parameters. Therefore, make sure the actual value is approximately equal to the ambient value. If it is not, wait until the actual value approaches the ambient value. This happens automatically, when the manipulated variable Zero is input to the control system.

Example:
Temperature controls
=> ambient value = ambient temperature or room temperature

Pressure control
=> ambient value = ambient pressure (normally 1 bar)

Autotuning manipulated variable
During the initial autotuning phase (→ figure 38, 39 and 40), a manipulated variable is output as per the variable "Autotuning_manipulated_variable_12Bit_UINT", until the actual value is around 85 % of the setpoint. Select a manipulated variable with which you want to carry out the initial phase of autotuning. Enter the 12-bit value (4095 = 100 %) in "Autotuning_manipulated_variable_ 12Bit_UINT". You can estimate a suitable autotuning component as follows:

- Estimate which manipulated variable is required to keep the actual value at the setpoint (e.g. 15 %).
- Multiply the estimated value by 2 and calculate the corresponding 12-bit value for the autotuning component (e.g. 15 % × 2 = 30 % => 4095 × 30 % = 1229 increments).

When autotuning is finished, an optimised value of the autotuning component will be output. You can repeat autotuning with this value if necessary. If you have a low order control system (small energy reserves), then selection of the autotuning manipulated quantity is uncritical. Then you can use whatever is the maximum manipulated variable.

Caution!
If the autotuning manipulated quantity is selected too large, then depending on the system it might continue to oscillate at levels much greater than the setpoint (→ fig. 39). This could damage your system or machine. For further information on the autotuning component, refer to the corresponding section below.

Setting autotuning BOOL variables to "1"
Set the following "autotuning variables" to "1":

- Autotuning_accept_parameters_BOOL=1
- Autotuning_activate_BOOL=1
- Autotuning_re_optimize_BOOL=1

Enabling the PID controller
Autotuning begins only when the PID controller is enabled. Set the following variables to "1":

- P_activate_BOOL=1
- I_activate_BOOL=1
- D_activate_BOOL=1

Information:
Autotuning starts with each leading edge of "Autotuning_activate_BOOL=1" plus "P_activate_BOOL=1" (AND relationship), that is, only when both BOOL variables are "1".

Terminating system identification
During autotuning phases 1 and 2 (→ fig. 38), the control system is identified. The following system variables are determined:

- System gain Ks
- System delay time Tg
- System dead time Tu

During system identification, the following two status displays are output:

- Autotuning phase 1: autotuning_active_BOOL=1
- Autotuning_re_optimization_active_BOOL=0
- Autotuning phase 2: autotuning_active_BOOL=1
- Autotuning_re_optimization_active_BOOL=1

When the system is identified, both status displays are reset to "0":

- Autotuning_active_BOOL=0
- Autotuning_re_optimization_active_BOOL=0

Calculating PID parameters according to selectable optimization criteria

If you carried out autototuning as described above, then the PID parameters were calculated according to the following optimization criteria (depending on the system variables):

- Control behaviour
- Aperiodic boundary case

By assigning different values to the command bits, you can use other optimization criteria for the PID parameter calculation:

- Leave "Autotuning_activate_BOOL=1"
- Change the values of the command bits, as described below in the section on "Calculating the PID autotuning parameters".

Accepting the PID autotuning parameters

If you carried out the steps described above, the autotuning parameters which were determined will be accepted. Otherwise, set the variable "Autotuning_accept_ parameters_BOOL" to "1".
=> The autotuning parameters will be sent to the internal PID controller

Aborting autotuning
Autotuning is aborted with the falling edge of "Autotuning_activate_BOOL" plus "P_activate_BOOL" (AND relationship). That is, whenever either "Autotuning_activate_BOOL=0" or "P_activate_BOOL=0" or when both BOOL variables equal zero.

Terminating autotuning
After autotuning, there are two ways of proceeding to PID control of your control system. To directly accept the autotuning parameters:

- Set autotuning_accept_parameters_BOOL=1
- Set all other input variables which begin with "Autotuning_" to zero, to avoid restarting autotuning by mistake.

It is better to accept the PID parameters from the input. Transfer the autotuning parameters to the input section of the function block, as follows:

- Proportional_rate_percent_UINT
- Reset_time_10ths_UINT
- Derivate_action_time_10ths_UINT
- Set autotuning_accept_parameters_BOOL=0
- Set all other input variables which start with "Autotuning_" to "0".

Recommended procedure

Carry out autotuning once for each control system. After the optimum PID parameters are found, transfer them to the input section of the controller and keep them there. When operating the controller, set the following input variables (which start with "Autotuning_") to "0":

- Autotuning_accept_parameters_BOOL=0
- Autotuning_activate_BOOL=0
- Autotuning_re_optimize_BOOL=1

Problems with autotuning

- Autotuning component chosen poorly
  - For an autotuning component optimised for the given control system, autotuning proceeds as shown in figure 38.
  - If the autotuning proceeds as shown in figure 39 (overshooting to the edge of the measurement range), then the autotuning component is too large.
    => Halve the autotuning component.
  - If the autotuning proceeds as shown in figure 40 (overshooting to the edge of the measurement range), then the autotuning component is too small.
    => Double the autotuning component.

- Autotuning was not started at ambient values (e.g. ambient temperature)
  - The system was not identified properly.
    => Restart autotuning at ambient values
    (→ Quick reference guide, from page 265).

- Autotuning did not result in the desired control behaviour, although autotuning was successful (→ fig. 38)
  - You want to use the determined autotuning parameters; however, you set the variable "Autotuning_accept_parameters_BOOL" to "0".
    => Set "Autotuning_accept_parameters_BOOL" to "1".
  - You transferred the autotuning parameters to the input section of the controller and want to use these values; however, you set "Autotuning_accept_parameters_BOOL" to "1", or you made a mistake when transferring the parameters.
    => Set "Autotuning_accept_parameters_BOOL" and all other BOOL variables which start with "Autotuning_" to "0".
    => Check the entered parameters in the input section.

- Although very small values were entered for the autotuning component, the actual value goes far beyond the setpoint and does not settle
  - Your control system might be an "integrating system" without compensation (the actual value increases steadily for a constant manipulated variable). This is not a PTn system.
    => The autotuning controller is not suitable for this kind of system. For control purposes, use a PD controller (PID controller with the I component disabled) or the function block "U_PD_three_step_controller".

Autotuning manipulated variable

Autotuning produces exact PID parameter values if the manipulated variable, with which autotuning was carried out, is tailored to the control system. The autotuning manipulated variable can be within a large range (+/– 1.5 times the ideal value). Precise PID parameter values cannot be determined if the autotuning manipulated variable is too large (→ fig. 39). If the autotuning manipulated variable is too small, a minimum value might not be reached. In that case, too, the PID parameters cannot be determined (→ fig. 40).

Estimating the autotuning manipulated variable

You can estimate the value for the autotuning component as follows:

- Estimate which manipulated variable is required, to keep the actual value at the setpoint (e.g. 15 %).
- Multiply the estimated value by 2. Calculate the corresponding 12-bit value for the autotuning component (e.g. 15 % × 2 = 30 % => 4095 × 30 % = 1229 increments).
- If you have no information on the relationships described above, initially set the manipulated variable to around 40 % (= 1638 12-bit increments). If the first autotuning procedure is not successful as in figure 39 and 40, modify the autotuning component accordingly and repeat the autotuning procedure.

When autotuning is carried out once, an optimised autotuning manipulated variable is output in the output section of the function block. The value depends on the calculated system gain, Ks. (The value can be determined precisely by repeating the optimization process, as described above.) If this value deviates greatly from the manipulated variable (> factor of 2) for which autotuning was performed, then you are recommended to repeat autotuning with the optimised autotuning manipulated variable.

For low order control systems, the choice of the autotuning manipulated variable is not critical. That means you can select a large value even though the gain of your control system is large. As an example, for air-heating controls autotuning can be carried out with the maximum manipulated variable (= 4095), because the boundary of the measurement range is not reached even with large setpoints.

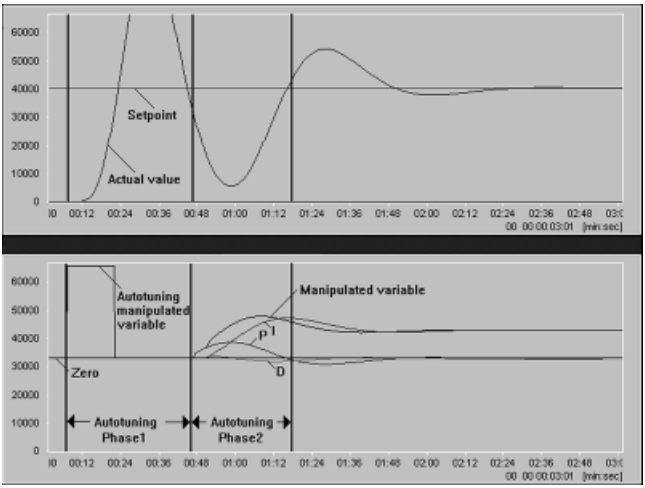


Figure 39: PID autotuning with "autotuning manipulated variable" too large. Actual value goes considerably above the setpoint and possibly exceeds the measurement range.

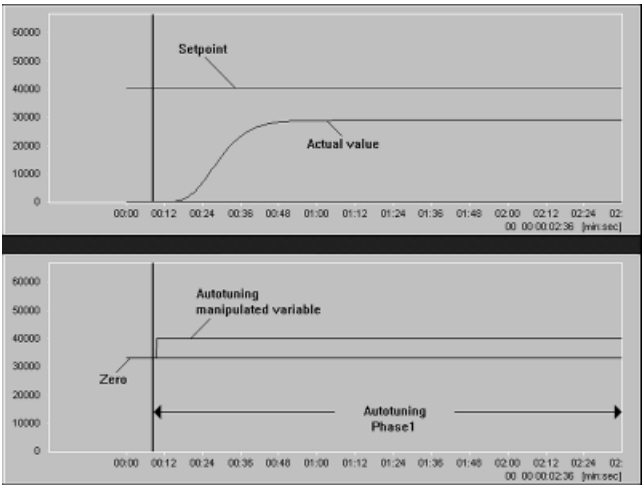


Figure 40: PID autotuning when the "autotuning manipulated variable" is too small. The actual value does not reach the setpoint.

Automatic re-optimization

The simplest autotuning option is to start "Autotuning_activate_BOOL" and "Autotuning_re_ optimize_BOOL" simultaneously (set them to "1"), as described in the Quick reference guide from page 265 above. The first phase of autotuning ("Autotuning_active_BOOL=1") will then execute. Immediately afterwards, a re-optimization will be carried out (→ status display "Autotuning_re_ optimization_active_BOOL=1"). Autotuning terminates the first time the setpoint is reached (→ fig. 38). If the actual value overshoots the setpoint, then the automatic re-optimization will identify the system incorrectly. In such cases, you are recommended to carried out re-optimization manually (→ below) or repeat autotuning with another (reduced) autotuning component (→ above).

Manual re-optimization

Manual re-optimization calculates the system parameters more precisely, including the PID parameters. This is especially true in cases in which the actual value overshoots the first time it reaches the setpoint at the end of autotuning phase 2 (→ fig. 40).

The first autotuning phase is started without re-optimization ("Autotuning_activate_BOOL=1" and "Autotuning_re_optimize_BOOL=0") (comp. Quick reference guide from page 265).

When the actual value settles at the setpoint (as shown on the right-hand side of figure 39), then re-optimization is initiated by hand ("Autotuning_re_optimize_BOOL=1"). Make sure both the actual value and the manipulated variable no longer change.

Online re-optimization

If the system gain changes or there are major disturbances during control, then initiating re-optimization (leading edge "Autotuning_re_optimize_BOOL=1" while autotuning is activated) tailors the system to the control relationships. Make sure the actual value settles at the setpoint (like for manual re-optimization) and the manipulated variable no longer changes.

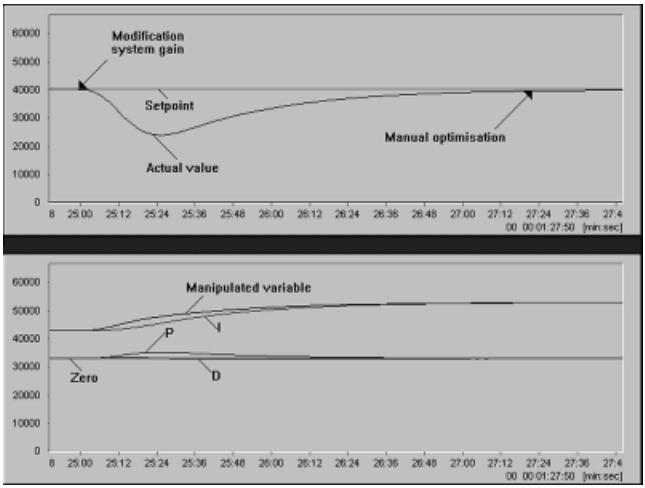


Figure 41: Online re-optimization when the system gain changes

PTn system parameters

The PTn system parameters (→ fig. 21)

- System gain Ks
- System delay time Tg
- and system dead time Tu

are calculated and output in the output section of the function block. After any control technology adjustment procedure, you can set the optimum parameters for the PID controller by hand.

Calculating the PID autotuning parameters

Once the system parameters have been determined (these values no longer change), the autotuning parameters Kp, Tn and Tv can then be calculated in accordance with different optimization criteria.

Example:
Initially, autotuning was carried out with the following optimization criteria:

- Control behaviour
- aperiodic boundary case

Following autotuning, the PID autotuning parameters were recalculated according to the following optimization criteria:

- Disturbance behaviour
- 20 % overshoot

Recalculating the PID autotuning parameters in accordance with different optimization criteria:

- Keep the variable "Autotuning_activate_BOOL" at "1". If this variable was set to "0", then the optimization criteria for calculating the PID autotuning parameters cannot be changed.
- With each rising edge (from "0" to "1") of "Autotuning_accept_parameters_BOOL", the PID autotuning parameters are recalculated in accordance with the specified optimization criteria.
- After each online re-optimization, the parameters are also recalculated in accordance with the specified optimization criteria.

Optimization criterion "Control behaviour/disturbance behaviour":

- Control
  => Autotuning_disturbance_optimization_BOOL=0
- Disturbance
  => Autotuning_disturbance_optimization_BOOL=1

Optimization criteria "Aperiodic boundary case/ 20 % overshoot":

- Control behaviour with aperiodic boundary case => Autotuning_overshoot_20_percent_BOOL=0
- Control behaviour with 20 % overshoot => Autotuning_overshoot_20_percent_BOOL=1

Using the controller without autotuning

The autotuning PID controller is based on the function block "U_PID_controller" (→ above). The corresponding I/O section (all variables which do not begin with Autotuning) is identical with the above-mentioned function block.

If autotuning is not activated and "Autotuning_accept_parameters_BOOL" is set to "0", then the behaviour of the function block is reduced to that of the above-mentioned function block. The controller then accepts the controller parameters of the input section. If "Autotuning_accept_parameters_BOOL" is set to "1", then the determined autotuning parameters are used.

Testing for proper functioning

In order to become familiar with using the autotuning controller, you can use the function block U_PTn_system. It allows you to simulate a PTn system. For a third order system with a system gain of 2 and a system delay time of 30 s, the input section of this function block should be set as follows:

- Input_value_15Bit_INT:=manipulated_variable_12Bit,
- Activate_BOOL:=1,
- Interrupt_BOOL:=0,
- Accept_manual_value_BOOL:=0,
- Order_PTn_UINT:=3,
- Delay_time_Tg_10thsec_UINT:=300,
- P_gain_percent_UINT:=200,
- Start_value_15Bit_INT:=0,
- Manual_value_15Bit_INT:=0

The actual value can be taken from the output section of the function block. The actual value is limited to 12 bits and can be sent to the controller.

Description of the application example

The PID autotuning controller performs re-optimization for the following settings:

- Control behaviour
  => autotuning_disturbance_optimization_BOOL=0
- aperiodic boundary case
  => autotuning_overshoot_20_percent_BOOL=0
- Autotuning is carried out with an "Autotuning component" of 3000 "12-bit increments" (= 73 %). If autotuning is not active, the determined autotuning parameters are accepted, because of "Autotuning_accept_parameters_BOOL=1", rather than Kp, Tn and Tv from the input section of the controller.

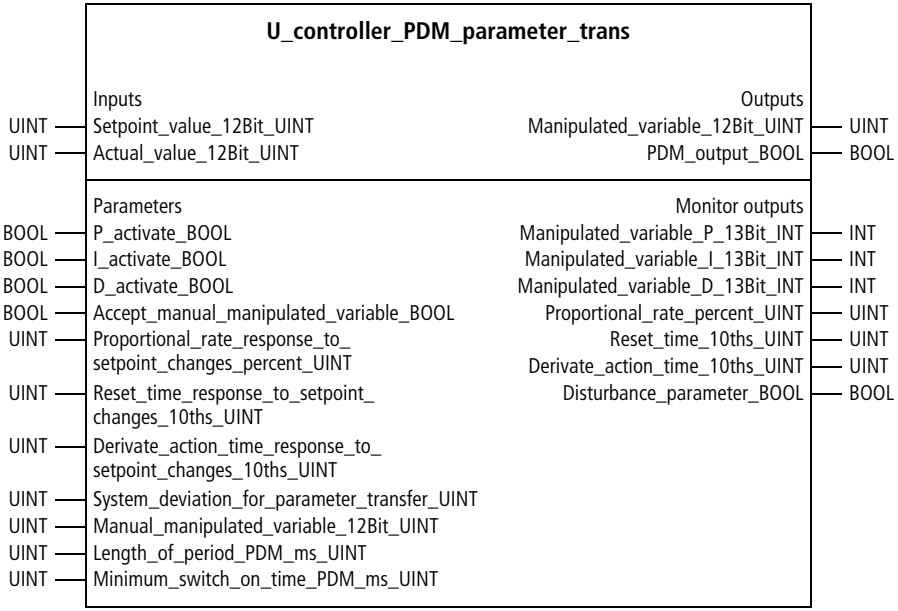
Application of the function block "U_PID_autotuning_controller" in the program "Auto_PT3"

```
PROGRAM Auto_PT3
VAR
  PID_AUTOTUNING_CONTROLLER : U_PID_AUTOTUNING_CONTROLLER ;
  Setpoint_value : UINT ;
  Actual_value : UINT ;
  Enable_Autotuning_controller : BOOL ;
  Manual_operation : BOOL ;
  Manual_manipulated_variable : UINT ;
  Enable_Autotuning : BOOL ;
  Enable_re_optimize : BOOL ;
  Manipulated_variable_Autotuning : UINT ;
  Manipulated_variable_P_Autotuning : INT ;
  Manipulated_variable_I_Autotuning : INT ;
  Manipulated_variable_D_Autotuning : INT ;
END_VAR

CAL PID_AUTOTUNING_CONTROLLER(
  Setpoint_value_12Bit_UINT :=Setpoint_value,
  Actual_value_12Bit_UINT :=Actual_value,
  P_activate_BOOL :=Enable_Autotuning_controller,
  I_activate_BOOL :=Enable_Autotuning_controller,
  D_activate_BOOL :=Enable_Autotuning_controller,
  Accept_manual_manipulated_variable_BOOL :=Manual_operation,
  Proportional_rate_percent_UINT :=120,
  Reset_time_10ths_UINT :=300,
  Derivate_action_time_10ths_UINT :=30,
  Manual_manipulated_variable_12Bit_UINT :=Manual_manipulated_variable,
```

```
  Autotuning_acception_parameters_BOOL :=1,
  Autotuning_activate_BOOL :=Enable_Autotuning,
  Autotuning_re_optimize_BOOL :=Enable_re_optimize,
  Autotuning_disturbance_optimization_BOOL :=0,
  Autotuning_overshoot_20_percent_BOOL :=0,
  Autotuning_manipulated_variable_12Bit_UINT :=3000,
  Manipulated_variable_12Bit_UINT=>Manipulated_variable_Autotuning,
  Manipulated_variable_P_13Bit_INT=>Manipulated_variable_P_Autotuning,
  Manipulated_variable_I_13Bit_INT=>Manipulated_variable_I_Autotuning,
  Manipulated_variable_D_13Bit_INT=>Manipulated_variable_D_Autotuning
  )

END_PROGRAM
```

PID controller

U_controller_PDM_parameter_trans
Combination of "U_PID_controller", "U_PDM_ Contactor" and "U_PID_parameter_transfer" (disturbance <-> response of setpoint change)



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Setpoint_value_12Bit_UINT	Setpoint value	0 to 4095
Actual_value_12Bit_UINT	Actual value	0 to 4095
Parameters		
P_activate_BOOL	Activates the P component	0/1
I_activate_BOOL	Activates the I component	0/1
D_activate_BOOL	Activates the D component	0/1
Accept_manual_manipulated_ variable_BOOL	"Soft" acceptance of the manual value	0/1
Proportional_rate_response_to_ setpoint_changes_percent_UINT	Proportional rate [%] optimized for response to setpoint changes	0 to 65535
Reset_time_response_to_setpoint_ changes_10ths_UINT	Reset time [0.1 s] optimized for response to setpoint changes	0 to 65535
Derivate_action_time_response_ to_setpoint_changes_10ths_UINT	Derivate action time [0.1 s] optimized for response to setpoint changes	0 to 65535
System_deviation_for_parameter_ transfer_UINT	System deviation for paramater transfer	0 to 65535
Manual_manipulated_variable_ 12Bit_UINT	Manual value	0 to 4095
Length_of_period_PDM_ms_UINT	Length of period [ms] (PDM)	0 to 65535
Minimum_switch_on_time_ PDM_ms_UINT	Minimum switch on time [ms] (PDM)	0 to 65535

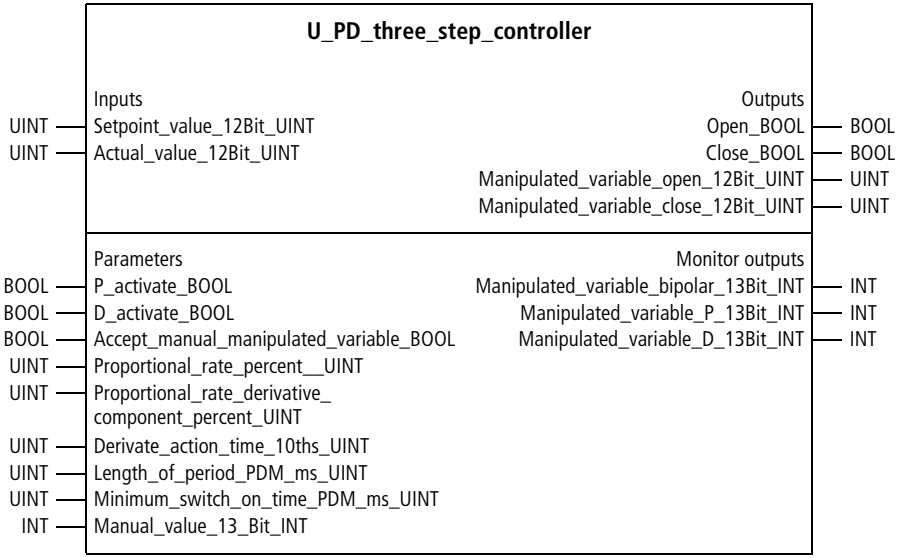
Designation	Significance	Value range
Outputs		
Manipulated_variable_12Bit_UINT	Manipulated variable (analogue, 12 Bit)	0 to 4095
PDM_output_BOOL	Digital output of the PDM system	0/1
Monitor outputs		
Manipulated_variable_P_13Bit_INT	P component	–4095 to 4095
Manipulated_variable_I_13Bit_INT	I component	–4095 to 4095
Manipulated_variable_D_13Bit_INT	D component	–4095 to 4095
Proportional_rate_percent_UINT	Proportional gain [%]	0 to 65535
Reset_time_10ths_UINT	Reset time [0.1 s]	0 to 65535
Derivate_action_time_10ths_UINT	Derivate action time [0.1 s]	0 to 65535
Disturbance_parameter_BOOL	Status signal: Disturbance parameters active	0/1

Description

The function block is a combination of "U_PID_controller", "U_PDM_Contactor" and "U_PID_parameter_transfer" (disturbance <-> response of setpoint change). See the description of this function blocks.

PD three-step controller

U_PD_three_step_controller
PD Controller with Three-step Behaviour For "Opening" and "Closing" Valves



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Setpoint_value_12Bit_UINT	Setpoint	0 to 4095
Actual_value_12Bit_UINT	Actual value	0 to 4095

Designation	Significance	Value range
Parameters		
P_activate_BOOL	Activates the P component	0/1
D_activate_BOOL	Activates the D component	0/1
Accept_manual_manipulated_ variable_BOOL	Smooth acceptance of the manual value	0/1
Proportional_rate_percent__UINT	Proportional gain Kp [%]	0 to 65535
Proportional_rate_derivative_ component_percent_UINT	D component of the proportional gain Kd [%]	0 to 65535
Derivate_action_time_10ths_UINT	Derivative action time Tv [0.1 s]	0 to 65535
Length_of_period_PDM_ms_UINT	Length of the period for PDM [ms]	0 to 65535
Minimum_switch_on_time_ PDM_ms_UINT	Minimum switch-on time for PDM [ms]	0 to 65535
Manual_value_13_Bit_INT	Manual value	–4095 to 4095
Outputs		
Open_BOOL	Opens something, such as a valve	0/1
Close_BOOL	Closes something, such as a valve	0/1
Manipulated_variable_open_ 12Bit_UINT	Analogue manipulated variable "open"	0 to 4095
Manipulated_variable_close_ 12Bit_UINT	Analogue manipulated variable "close"	0 to 4095
Monitor outputs		
Manipulated_variable_bipolar_ 13Bit_INT	Bipolar manipulated variable	–4095 to 4095
Manipulated_variable_P_13Bit_INT	P component	–4095 to 4095
Manipulated_variable_D_13Bit_INT	D component	–4095 to 4095

→ Do not make the ratio "period length/minimum switch-on time" too small. Otherwise, relatively large manipulated variables will be suppressed. See the section on minimum switch-on time on page 310.

Description

This controller is suitable for "integrating systems", meaning systems without compensation (so an I component is not necessary), e.g. flow control with a valve with the following options:

- open
- close
- stay (neither open nor closed)

The P and D components of the controller can be activated (= enabled) or deactivated separately, with the BOOL variables "P_activate_BOOL" and "D_activate_BOOL". Deactivating the D components resets the controller.

If "Accept_manual_manipulated_variable_BOOL" is "1", then the controller will output to "Manipulated_variable_12Bit_UINT" the value assigned to the variable "Manual_manipulated_variable_12Bit_UINT".

The controller is dimensioned with the variables Kp [%], Kd [%] and Tv [0.1 s]. To calculate the D component, a sampling time (Δt) is determined from the derivative action time (→ fig. 42). The determined changes of the actual value are factored with Kd. The following equation appplies:

- D component = Kd × ΔX
  ΔX = current_actual_value – earlier_actual_value ~ Tv
  => D component ~ Tv × Kd

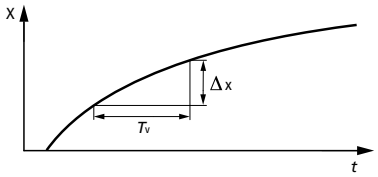


Figure 42: Calculating changes of the actual value, as a function of Tv

Example:
Derivative action time Tv = 10 s
D-gain Kd = 500 %
Within 10 s, the actual value changed 100 increments
=> D component = 100 × 5 = 500

If the derivative action time Tv is halved (5 s) and the D-gain Kd is doubled (1000 %), then the D component in the above example would stay the same, as follows:

- => D component = 50 × 10 = 500

Remember the following:

- Choose small Tv values to minimise the maximum change of the actual value during this time.
  => The resolution of the calculation of the D-value decreases.
  => The D component becomes unsteady.
- Choosing small Tv values decreases the time required to calculate the current D-value.
  => The calculation of the D-value is delayed by a factor equivalent to Tv.

For the digital outputs "Open_BOOL" and "Close_BOOL", the length of the period and minimum switch-on time for pulse duration modulation must be entered. The controller outputs the analogue values "Manipulated_variable_open_12Bit_UINT" and "Manipulated_variable_close_12Bit_UINT". The PD components of the manipulated variable are used for (remote) diagnosis of control behaviour. The overall manipulated variable is created by adding the individual components.

Example:
When values are assigned to the parameters Kd and Tv, the change of the system deviation (which takes effect within 2000 s) is factored with 5, producing the D component. The P component results from multiplying the system deviation by 0.8. The resulting "Overall manipulated variable" (Manipulated_variable_bipolar_13Bit_INT) is an input signal for pulse duration modulation, with a period of one minute and a minimum switch-on time of 3 s.

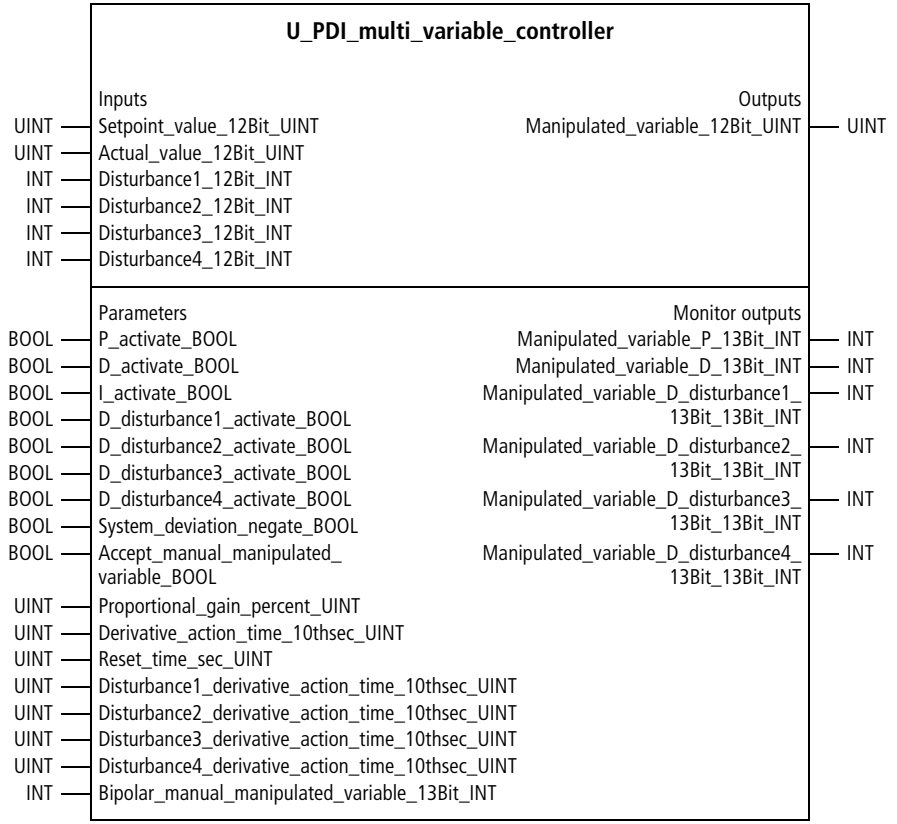
Application of the function block "U_PD_three_step_controller" in the program "Valve_1"

```
PROGRAM Valve_1
VAR
  Valve_1 : U_PD_THREE_STEP_CONTROLLER ;
  Setpoint_value_valve_1 : UINT ;
  Actual_value_valve_1 : UINT ;
  Enable_PD_Controller_valve_1 : BOOL ;
  Manual_operation_valve_1 : BOOL ;
  Manual_manipulated_variable_valve_1 : INT ;
  Open_valve_1 : BOOL ;
  Close_valve_1 : BOOL ;
END_VAR
```

```
CAL Valve_1(
  Setpoint_value_12Bit_UINT :=Setpoint_value_valve_1,
  Actual_value_12Bit_UINT :=Actual_value_valve_1,
  P_activate_BOOL :=Enable_PD_Controller_valve_1,
  D_activate_BOOL :=Enable_PD_Controller_valve_1,
  Accept_manual_manipulated_variable_BOOL :=Manual_operation_valve_1,
  Proportional_rate_P_percent_UINT :=80,
  Proportional_rate_D_percent_UINT :=500,
  Derivate_action_time_1Oths_UINT :=20000,
  Length_of_period_PDM_ms_UINT :=60000,
  Minimum_switch_on_time_PDM_ms_UINT :=3000,
  Manual_manipulated_variable_13Bit_INT :=Manual_manipulated_variable_valve_1,
  Open_BOOL=>Open_valve_1,
  Close_BOOL=>Close_valve_1
  )

END_PROGRAM
```

PDI multi-variable controller

U_PDI_multi_variable_controller
PD Controller Followed by an Integrator and Compensation for Four Disturbance Variables



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Setpoint_value_12Bit_UINT	Setpoint	0 to 4095
Actual_value_12Bit_UINT	Actual value	0 to 4095
Disturbance1_12Bit_INT	Disturbance 1	0 to 4095
Disturbance2_12Bit_INT	Disturbance 2	0 to 4095
Disturbance3_12Bit_INT	Disturbance 3	0 to 4095
Disturbance4_12Bit_INT	Disturbance 4	0 to 4095
Parameters		
P_activate_BOOL	Activates the P component	0/1
D_activate_BOOL	Activates the D component	0/1
I_activate_BOOL	Activates the I component	0/1
D_disturbance1_activate_BOOL	Activates the D component of disturbance 1	0/1
D_disturbance2_activate_BOOL	Activates the D component of disturbance 2	0/1
D_disturbance3_activate_BOOL	Activates the D component of disturbance 3	0/1
D_disturbance4_activate_BOOL	Activates the D component of disturbance 4	0/1
System_deviation_negate_BOOL	Negates the system deviation	0/1
Accept_manual_manipulated_ variable_BOOL	Smooth acceptance of the manual value	0/1
Proportional_gain_percent_UINT	Proportional gain Kp [%]	0 to 65535
Derivative_action_time_10thsec_UINT	Derivative action time Tv [0.1 s]	0 to 65535
Reset_time_sec_UINT	Reset time Tn [s]	0 to 65535

Designation	Significance	Value range
Disturbance1_derivative_action_time_10thsec_UINT	Derivative action time Tv of disturbance 1 [0.1 s]	0 to 65535
Disturbance2_derivative_action_time_10thsec_UINT	Derivative action time Tv of disturbance 2 [0.1 s]	0 to 65535
Disturbance3_derivative_action_time_10thsec_UINT	Derivative action time Tv of disturbance 3 [0.1 s]	0 to 65535
Disturbance4_derivative_action_time_10thsec_UINT	Derivative action time Tv of disturbance 4 [0.1 s]	0 to 65535
Bipolar_manual_manipulated_variable_13Bit_INT	Manual value	–4095 to 4095
Outputs		
Manipulated_variable_12Bit_UINT	Manipulated variable (analogue, 12 bits)	0 to 4095
Monitor outputs		
Manipulated_variable_P_13Bit_INT	P component	–4095 to 4095
Manipulated_variable_D_13Bit_INT	D component	–4095 to 4095
Manipulated_variable_D_disturbance1_13Bit_INT	D component of disturbance 1	–4095 to 4095
Manipulated_variable_D_disturbance2_13Bit_INT	D component of disturbance 2	–4095 to 4095
Manipulated_variable_D_disturbance3_13Bit_INT	D component of disturbance 3	–4095 to 4095
Manipulated_variable_D_disturbance4_13Bit_INT	D component of disturbance 4	–4095 to 4095

Description

This controller can generate PI behaviour with disturbance variable compensation for up to four variables (→ fig. 43).

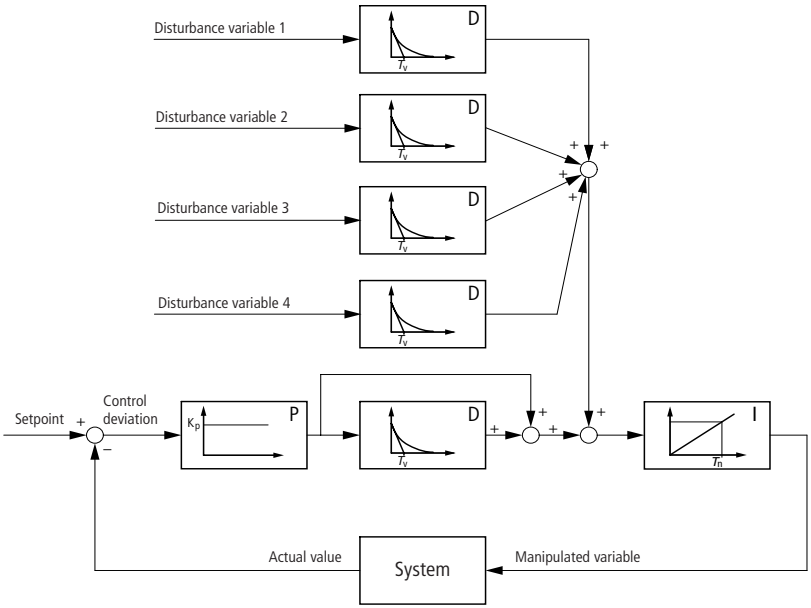


Figure 43: Block diagram of the PDI multi-variable controller

In contrast to the PID controller, the basic controller is a PD controller followed by an integrator. The result is PI-behaviour, due to:

- Integration of P => I
- Integration of D => P

Compared to a "usual" PI controller, the advantage of this "PI controller" (PD controller + integration) is that more variables can be added to the control algorithm. In contrast to P components, it is not necessary to enter a setpoint in order to differentiate the variables (disturbance variables). (Also, there is no meaningful setpoint.) With regard to the disturbance variables, adding integration results in P-behaviour, without the input of a setpoint.

The components of the controller can be activated (= enabled) or deactivated separately, with the corresponding BOOL variables. "System_deviation_ negate_BOOL=1" toggles the effect of the control deviation and a change of the control deviation on the manipulated variable (only regarding the P and D components of the basic controller). Deactivating the I and D components resets the controller.

If "Accept_manual_manipulated_variable_BOOL" is "1", then the controller will output to "Manipulated_variable_ 12Bit_UINT" the value assigned to the variable "Manual_manipulated_variable_12Bit_UINT". When "Accept_manual_manipulated_variable_BOOL" changes back to "0", then the controller accepts the manual value and continues control with this manipulated variable, without oscillations (→ "smooth acceptance of the manual value").

The controller is dimensioned with the standardised variables Kp [%], Tn [0.1 s] and Tv [0.1 s]. In order to calculate the D components, the derivative action times (→ fig. 44) are used to determine the corresponding sampling times (Δt). The calculated changes of the actual value are factored with the respective Kd's.

Example:
Derivative action time = 10 s
Kd = 500 %
Within 10 s, the actual value (the disturbance) has changed 100 increments
=> D component = 100 × 5 = 500

If the derivative action time Tv is halved (5 s) and the D-gain Kd is doubled (1000 %), then the D component in the above example would stay the same, as follows:

- => D component = 50 × 10 = 500

Remember the following:

- Choose small Tv values to minimise the maximum change of the actual value during this time.
  => The resolution of the calculation of the D-value decreases. => The D component becomes unsteady.
- Choosing small Tv values decreases the time required to calculate the current D-value.
  => The calculation of the D-value is delayed by a factor equivalent to Tv.

The controller provides the analogue output variable "Manipulated_variable_12Bit_UINT". The PD components of the controller and the D components of the manipulated variables are used for (remote) diagnosis of control behaviour.

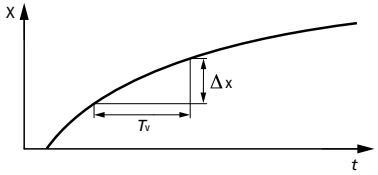


Figure 44: Calculating changes of the actual value and disturbance variables, as a function of Tv

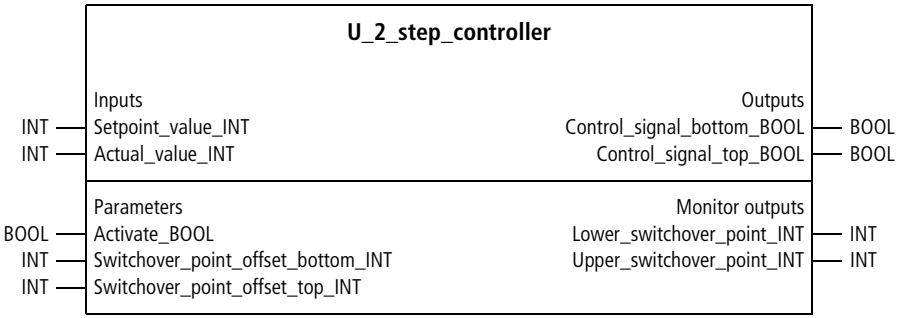
Example:
For a temperature control (setpoint and actual value), the disturbance variables ambient temperature and brightness are included for the control. The manipulated variable component of the disturbance variable "ambient temperature" results from the change of the ambient temperature within 10 minutes, factored with 5. The manipulated variable component "Brightness" results from the change of the brightness within 5 minutes, factored with 4.

Application of the Function Block "U_PDI_multi_variable_controller" in the Program "Hothouse"

```
PROGRAM Hothouse
VAR
  PDI_MULTI_VARIABLE_CONTROLLER : U_PDI_MULTI_VARIABLE_CONTROLLER ;
  Actual_temperature : UINT ;
  Disturbance_outdoor_temperature : INT ;
  Disturbance_brightness : INT ;
  Enable_Controller : BOOL ;
  Manual_operation : BOOL ;
  Manual_manipulated_variable : INT ;
  Manipulated_variable_PDI_Controller : UINT ;
END_VAR

CAL PDI_MULTI_VARIABLE_CONTROLLER(
  Actual_value_12Bit_UINT :=Actual_temperature,
  Disturbance1_12Bit_INT :=Disturbance_outdoor_temperature,
  Disturbance2_12Bit_INT :=Disturbance_brightness,
  Disturbance3_12Bit_INT :=0,
  Disturbance4_12Bit_INT :=0,
  P_activate_BOOL :=Enable_Controller,
  D_activate_BOOL :=Enable_Controller,
  I_activate_BOOL :=Enable_Controller,
  D_disturbance1_activate_BOOL :=Enable_Controller,
  D_disturbance2_activate_BOOL :=Enable_Controller,
  D_disturbance3_activate_BOOL :=0,
  D_disturbance4_activate_BOOL :=0,
```

```
  System_deviation_negate_BOOL :=0,
  Accept_manual_manipulated_variable_BOOL :=Manual_operation,
  Proportional_gain_percent_UINT :=200,
  Derivative_action_gain_percent_UINT :=200,
  Derivative_action_time_10thsec_UINT :=1000,
  Reset_time_sec_UINT :=20,
  Disturbance1_proportional_D_gain_percent_UINT :=500,
  Disturbance1_derivative_action_time_10thsec_UINT :=6000,
  Disturbance2_proportional_D_gain_percent_UINT :=400,
  Disturbance2_derivative_action_time_10thsec_UINT :=3000,
  Disturbance3_proportional_D_gain_percent_UINT :=0,
  Disturbance3_derivative_action_time_10thsec_UINT :=0,
  Disturbance4_proportional_D_gain_percent_UINT :=0,
  Disturbance4_derivative_action_time_10thsec_UINT :=0,
  Manual_manipulated_variable_13Bit_INT :=Manual_manipulated_variable,
  Manipulated_variable_12Bit_UINT=>Manipulated_variable_PDI_Controller
  )

END_PROGRAM
```

Unsteady controllers **U_2_step_controller**
Two-Step Controller



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Setpoint_value_INT	Setpoint	–16383 to 16383
Actual_value_INT	Actual value	–16383 to 16383
Parameters		
Activate_BOOL	Activates the controller	0/1
Switchover_point_offset_bottom_INT	Relative offset of the lower switchover point	–16383 to 16383
Switchover_point_offset_top_INT	Relative offset of the upper switchover point	–16383 to 16383
Outputs		
Control_signal_bottom_BOOL	Control signal for dropping below the lower switchover point	0/1
Control_signal_top_BOOL	Control signal for exceeding the upper switchover point	0/1
Monitor outputs		
Lower_switchover_point_INT	Lower switchover point	–32768 to 32767
Upper_switchover_point_INT	Upper switchover point	–32768 to 32767

Description

This function block is activated (enabled) with "Activate_BOOL=1". When it is deactivated, both control signal outputs are "0". The function block calculates the lower and upper switchover points by adding the setpoint and the respective offsets (→ fig. 45).

When the upper switchover point is exceeded, "Control_signal_top_BOOL=1", when the value drops below the lower switchover point, "Control_signal_bottom_BOOL=1". In each case, the other control signal is set to "0".

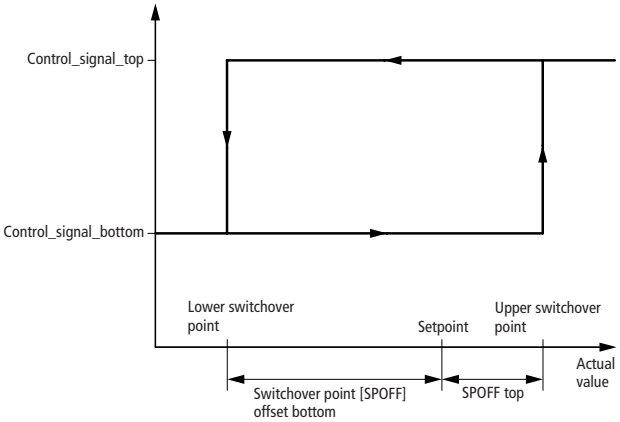


Figure 45: Switching the 2-step controller with hysteresis

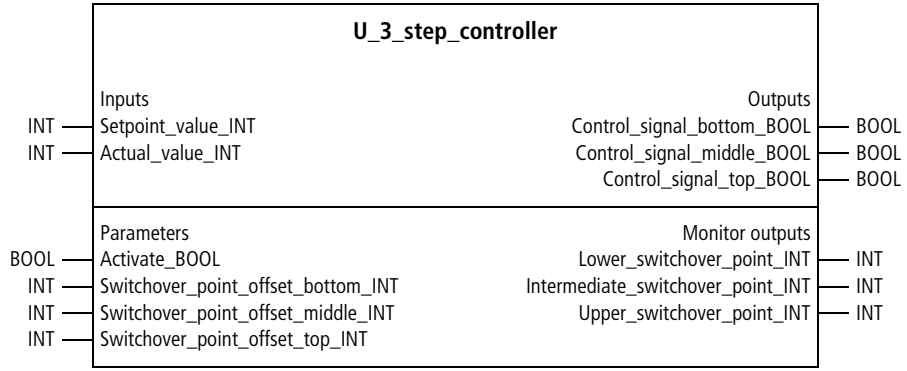
Example:
The following switchover points apply for the 2-step controller parameterised below:

- Lower switchover point = 2000 – 200 = 1800
- Upper switchover point = 2000 + 300 = 2300

Application of the function block "U_2_step_controller" in the program "Step_2"

```
PROGRAM Step_2
VAR
  Two_step_controller : U_2_STEP_CONTROLLER ;
  Setpoint_value : INT ;
  Actual_value : INT ;
  Enable_Controller : BOOL ;
  Digital_output_0_0 : BOOL ;
  Digital_output_0_1 : BOOL ;
END_VAR

CAL Two_step_controller(
  Setpoint_value_INT :=Setpoint_value,
  Actual_value_INT :=Actual_value,
  Activate_BOOL :=Enable_Controller,
  Switchover_point_offset_bottom_INT :=-200,
  Switchover_point_offset_top_INT :=300,
  Control_signal_bottom_BOOL=>Digital_output_0_0,
  Control_signal_top_BOOL=>Digital_output_0_1,
  Lower_switchover_point_INT=>1800,
  Upper_switchover_point_INT=>2300)

END_PROGRAM
```

U_3_step_controller
Three-Step Controller



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Setpoint_value_INT	Setpoint	–16383 to 16383
Actual_value_INT	Actual value	–16383 to 16383
Parameters		
Activate_BOOL	Activates the controller	0/1
Switchover_point_offset_bottom_INT	Relative offset of the lower switchover point	–16383 to 16383
Switchover_point_offset_middle_INT	Relative offset of the middle switchover point	–16383 to 16383
Switchover_point_offset_top_INT	Relative offset of the top switchover point	–16383 to 16383

Designation	Significance	Value range
Outputs		
Control_signal_bottom_BOOL	Control signal for dropping below the lower switchover point	0/1
Control_signal_middle_BOOL	Control signal for exceeding the middle switchover point and "Control_signal_bottom_BOOL=1" or dropping below the middle switchover point and "Control_signal_top_BOOL=1"	0/1
Control_signal_top_BOOL	Control signal for exceeding the upper switchover point	0/1
Monitor outputs		
Lower_switchover_point_INT	Lower switchover point	–32768 to 32767
Intermediate_switchover_point_INT	Intermediate switchover point	–32768 to 32767
Upper_switchover_point_INT	Upper switchover point	–32768 to 32767

Description

This function block is activated (enabled) with "Activate_BOOL=1". When it is deactivated, the three control signal outputs are "0". The function block calculates the lower, middle and upper switchover points by adding the setpoint and the respective offsets (→ fig. 46).

When the upper switchover point is exceeded, "Control_signal_top_BOOL=1", when the value drops below the lower switchover point, "Control_signal_bottom_BOOL=1".

"Control_signal_middle_BOOL" switches to "1" in the following cases:

- The middle switchover point is exceeded and "Control_signal_bottom_BOOL=1"
- The value drops below the middle switchover point and "Control_signal_top_BOOL=1"

In each case, the other control signals are set to "0".

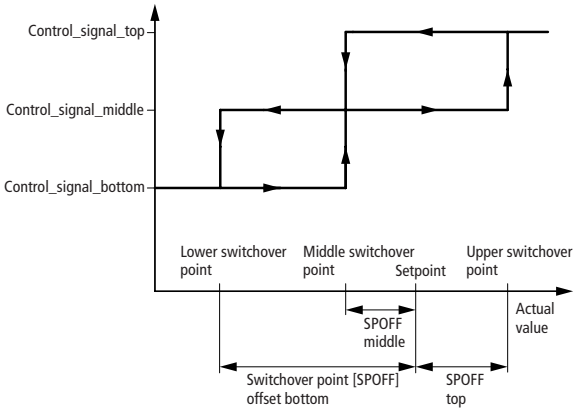


Figure 46: Switching the 3-step controller with hysteresis

Example:
The following switchover points apply for the 3-step controller parameterised below:

- Lower switchover point = 2000 – 200 = 1800
- Middle switchover point = 2000 – 50 = 1950
- Upper switchover point = 2000 + 100 = 2100

Application of the function block "U_3_step_controller" in the program "Step_3"

```
PROGRAM Step_3
VAR
  Three_step_controller : U_3_STEP_CONTROLLER ;
  Setpoint_value : INT ;
  Actual_value : INT ;
  Enable_Controller : BOOL ;
  Digital_output_0_0 : BOOL ;
  Digital_output_0_1 : BOOL ;
  Digital_output_0_2 : BOOL ;
END_VAR

CAL Three_step_controller(
  Setpoint_value_INT :=Setpoint_value,
  Actual_value_INT :=Actual_value,
  Activate_BOOL :=Enable_Controller,
  Switchover_point_offset_bottom_INT :=-200,
  Switchover_point_offset_middle_INT :=-50,
  Switchover_point_offset_top_INT :=100,
  Control_signal_bottom_BOOL=>Digital_output_0_0,
  Control_signal_middle_BOOL=>Digital_output_0_1,
  Control_signal_top_BOOL=>Digital_output_0_2
)

END_PROGRAM
```

5 Pulse Duration Modulation Systems

Fundamentals and general information

When actuators are not driven by analogue signals (such as in extruder heating systems), it is necessary to link the analogue manipulated variable of a PI- or PID controller with a pulse duration modulation (PDM, also known as pulse width modulation, PWM) system (for non-trivial control systems with large system dead times, satisfactory control results are not possible with 2-step or 3-step controllers).

Pulse duration modulation (PDM, also known as pulse width modulation, PWM) systems are used to relate the analog manipulated variable of a controller to a length of time (length of the period of the PDM). The switch-on time (pulse duration) is thus determined for the period. The periods remain constant, while the switch-on times (pulse durations) vary from period to period (→ fig. 47).

A PDM system processes the following steps in sequence (→ fig. 47):

- At the beginning of a period, the switch-on time is calculated, as a function of the length of the period and the analogue input signal (12-bit manipulated variable of the control)
- Switch-on phase
- Switch-off phase
- At the beginning of the next period, the next switch-on time is calculated

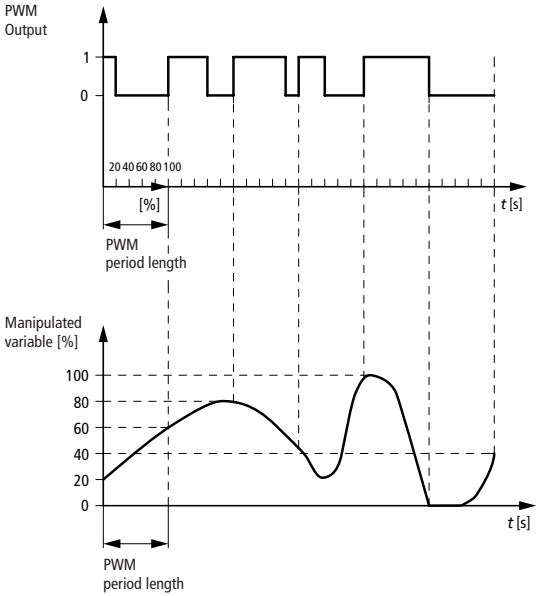


Figure 47: pulse duration modulation as a function of the manipulated variable of a control

Example:

- 1st Period
  - 12-bit analogue manipulated variable = 819
    (819/4095 = 20 %)
    length of period = 60 s
    =>
    switch-on time (pulse) = 12 s
    switch-off time = 48 s

- 2nd Period
  - 12-bit analogue manipulated variable = 2457
    (2457/4095 = 60 %)
    length of period = 60 s
    =>
    switch-on time (pulse) = 36 s
    switch-off time = 24 s

- 3rd Period
  - 12-bit analogue manipulated variable = 3276
    (3276/4095 = 80 %)
    length of period = 60 s
    =>
    switch-on time (pulse) = 48 s
    switch-off time = 12 s

etc.

Length of period

Select the length of period of your PDM system in accordance with the temporal behaviour of the control system (system inertia).

The following applies:

- Short periods result in small oscillations from the digital actuator circuitry (→ fig. 48)
- Long periods prolong the useful life of mechanical contactors
- Long periods allow precise temporal implementation of the analog manipulated variable

Example:
PLC cycle time = 100 ms
PDM period = 10000 ms

The PLC cycle time is 1 % of the PDM period. So an analog manipulated variable is implemented with a precision of at most 1 %.

Select PDM periods between 10 s (little mass) and 65 s (much mass) for a typical heating zone with energy reserves (metallic mass) such as extruders or ovens.

For dynamic and highly dynamic temperature zones (air-heating controls), you are recommended to select short periods (PLC cycle time × 100) or to use the noise-shape PDM technique (→ PDM_solid state), which permits optimum switching frequency.

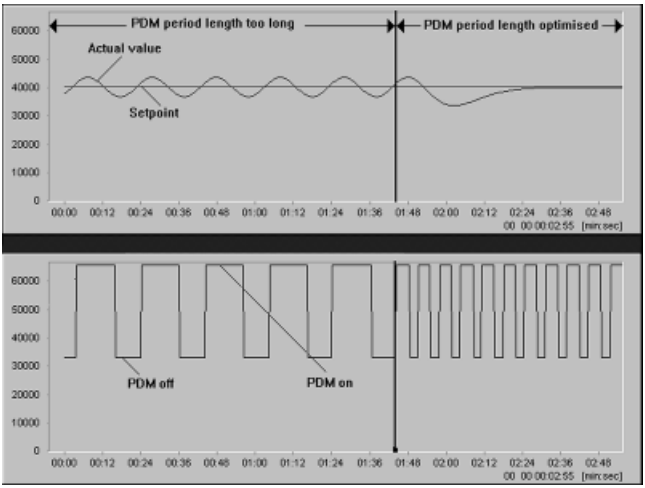


Figure 48: Control behaviour when the PDM period is too large and is optimised

Minimum switch-on time

The ratio "Length of period/minimum switch-on time" ("P/M") determines which manipulated variable components are ignored. So select a very small minimum switch-on time, to make P/M as large as possible. However, if the actuators are unable to respond to a very short switch-on time, then suppress these brief switch-on times to protect the hardware. In such cases, do not make the period too short.

Examples:

- heating/contactor
  => minimum switch-on time = 1 s
  => length of period = 40 s
  => manipulated variables less than 2.5 % are suppressed
- heating/solid-state relay
  => minimum switch-on time = 0 s
  => length of period = 20 s
  => No manipulated variables are suppressed
- fan/contactor or solid-state relay
  => minimum switch-on time = 2 s
  => length of period = 40 s
  => manipulated variables less than 5 % are suppressed

PID control combined with pulse duration modulation systems for several zones

When using a pulse duration modulation system, be sure to keep the PLC cycle time as short as possible. Otherwise, the system will make major errors when converting the manipulated variable.

Example:
PDM length of period = 10 s
(cannot be greater, due to the nature of the process)
manipulated variable = 20 %
=> PDM switch-on pulse time = 2 s
PLC cycle time = 200 ms

The imprecision of the conversion of the manipulated variable is around 10 %, because the PLC cycle time is 10 % of the PDM pulse duty factor.

If there are several control zones, minimise the cycle time for the PDM systems as follows:

- Do not call the controllers in each PLC cycle. Rather, call only one during each PLC cycle. As an example, if you have 20 zones, call each controller only during each 20th PLC cycle. For the controllers, this results in sampling times which are larger than the PLC cycle times, by an amount equal to the number of zones. In contrast to the sampling times of the PDM systems, this is insignificant.
- Call the PDM systems during each PLC cycle.

Example:
The following example uses 10 zones. It illustrates that the recommended technique reduces the PDM sampling time by a factor of 7, while only increasing the PID controller sampling time by a factor of 1.5:

- cycle time component of PDM systems = 1 ms
- cycle time component of PID controllers = 20 ms
- other cycle time components = 1 ms

=> If all function blocks are called in each PLC cycle, the following cycle and sampling times will result:

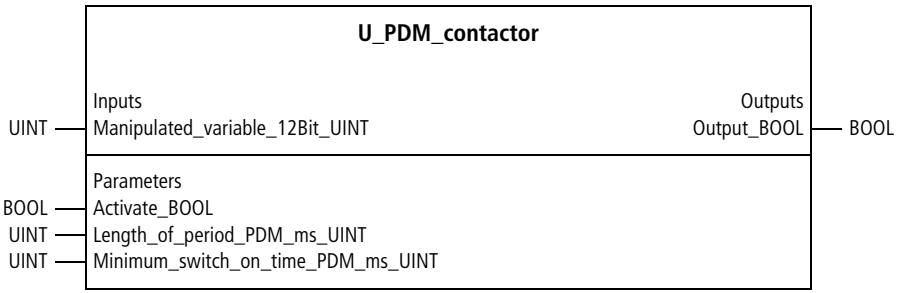
- PLC cycle time = 220 ms
- sampling time of the PDM systems = 220 ms
- sampling time of the PID controllers = 220 ms

=> If the PID controllers are called only in each tenth PLC cycle and the PDM systems are called in each one, the following cycle and sampling times will result:

- PLC cycle time = (20 + 10 × 1 + 1) ms = 31 ms
- sampling time of the PDM systems = 31 ms
- sampling time of the PID controllers = 310 ms

PDM with time inputs

U_PDM_contactor
Pulse duration modulation with input of time in milliseconds, suitable for contactors



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Manipulated_variable_12Bit_UINT	12-bit input variable (usually the output variable of a controller) of the PDM system	0 to 4095
Parameters		
Activate_BOOL	Enables the PDM system	0/1
Length_of_period_PDM_ms_UINT	Length of period [ms]	0 to 65535
Minimum_switch_on_time_PDM_ms_UINT	Minimum switch-on time [ms]	0 to 65535
Outputs		
Output_BOOL	Digital output of the PDM system	0/1

→ Do not make the ratio "period length/minimum switch-on time" too small. Otherwise, relatively large manipulated variables will be suppressed (→ section "Minimum switch-on time", page 310).

Description

The function block "U_PDM_contactor" can be used to connect mechanical contactors. The PDM system is usually linked with the 12-bit manipulated variable (4095 = 100 %) of a controller. For this reason, the input section includes the variable "manipulated_variable_12Bit_UINT".

"Activate_BOOL=1" starts the PDM system, and "Output_BOOL=1" outputs one pulse (→ fig. 49) when the specified minimum switch-on time is exceeded. "Activate_BOOL=0" sets the output signal to zero and resets the current processes. Reactivation starts a new period. The length of period and minimum switch-on time can be entered in ms. The maximum input value of 65535 results in times of 65,535 s. Further explanations can be found at the beginning of chapter 5.

→ If large values are required for length of period and minimum switch-on time (to 65535 s = 18.2 h), you can use the special function block "U_ZSFB01_special_FB", page 317. However, this function block should not be used if short periods are required. This special function block is only capable of outputting pulses with a precision of 1 s.

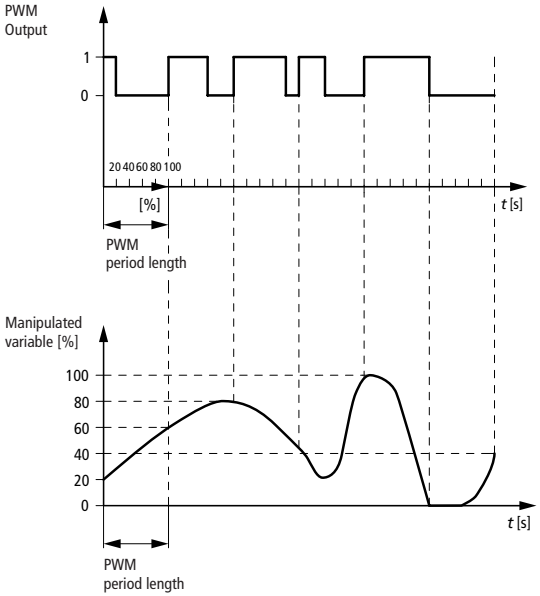


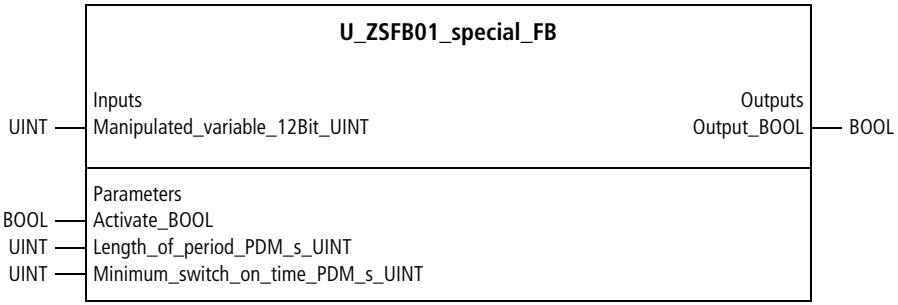
Figure 49: pulse duration modulation as a function of the manipulated variable of a controller

Example:
The application example links a PDM system with the PID controller of the same zone, by placing the 12-bit manipulated variable of the controller on the input of the PDM system. The length of period is set to 60 s and the minimum switch-on time to 3 s.

Application of the function block "U_PDM_contactor" in the program "zone1PDM"

```
PROGRAM zone1PDM
VAR
    PDM_zone1 : U_PDM_CONTACTOR ;
    PID_controller_manipulated_variable_12Bit_UINT : UINT ;
    Enable_zone1 : BOOL ;
    Digital_output_0_0 : BOOL ;
END_VAR
LD  PID_controller_manipulated_variable_12Bit_UINT
ST  PDM_zone1.Manipulated_variable_12Bit_UINT
CAL PDM_zone1(
    Activate_BOOL :=Enable_zone1,
    Length_of_period_PDM_ms_UINT :=60000,
    Minimum_switch_on_time_PDM_ms_UINT :=3000,
    Output_BOOL=>Digital_output_0_0
)

END_PROGRAM
```

U_ZSFB01_special_FB
Pulse Duration Modulation with Input of Time in Seconds, Suitable for Contactors



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Manipulated_variable_12Bit_UINT	12-bit input variable (usually the output variable of a controller) of the PDM system	0 to 4095
Parameters		
Activate_BOOL	Enables the PDM system	0/1
Length_of_period_PDM_s_UINT	Length of period [s]	0 to 65535
Minimum_switch_on_time_ PDM_s_UINT	Minimum switch-on time [s]	0 to 65535
Outputs		
Output_BOOL	Digital output of the PDM system	0/1

→ Do not make the ratio "period length/minimum switch-on time" too small. Otherwise, relatively large manipulated variables will be suppressed (→ section "Minimum switch-on time", page 310).

Description

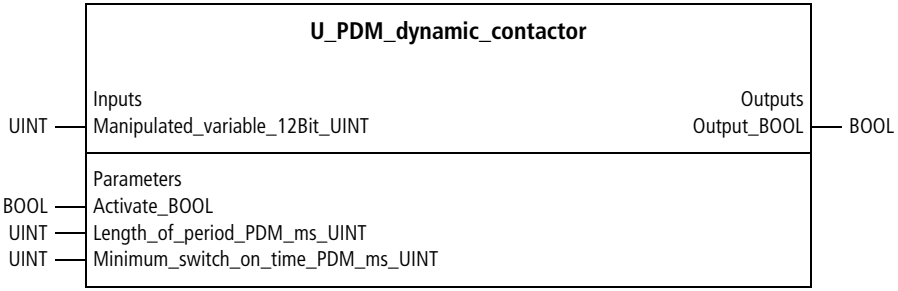
This function block is identical to the one "U_PDM_contactor", except that length of period and minimum switch-on time are entered in s. Thus, a maximum value of 65535 results in a maximum time of 65,535 s.

This function block should not be used if periods shorter than 65,535 s are required. This special function block is only capable of outputting pulses with a precision of 1 s.

Further explanations can be found at the beginning of chapter 5 and in the description of the function block "U_PDM_contactor", page 313.

PDM with time inputs and dynamic length of period

U_PDM_dynamic_contactor
Pulse Duration Modulation with Variable Length of Period, Suitable for Contactors



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Manipulated_variable_12Bit_UINT	12-bit input variable (usually the output variable of a controller) of the PDM system	0 to 4095
Parameters		
Activate_BOOL	Enables the PDM system	0/1
Length_of_period_PDM_ms_UINT	Length of period [ms]	0 to 65535
Minimum_switch_on_time_ PDM_ms_UINT	Minimum switch-on time [ms]	0 to 65535
Outputs		
Output_BOOL	Digital output of the PDM system	0/1

Description

The function block "U_PDM_dynamic_contactor" can be used to connect mechanical contactors. The PDM system is usually linked with the 12-bit manipulated variable (4095 = 100 %) of a controller. For this reason, the input section includes the variable "Manipulated_variable_12Bit_UINT". "Activate_BOOL=1" starts the PDM system, and "Output_BOOL=1" outputs one pulse (→ fig. 49) when the specified minimum switch-on time is exceeded. "Activate_BOOL=0" sets the output signal to zero and resets the current processes. Reactivation starts a new period. The length of period and minimum switch-on time can be entered in ms. The maximum input value of 65535 results in times of 65,535 s. Further explanations can be found at the beginning of chapter 5.

In contrast to the function block "U_PDM_contactor", this function block has an "intelligent" algorithm.

With it, the system does not run through the periods as for conventional PDM. Rather, when needed, the rhythm of the periods is automatically interrupted. This lowers the reaction time for large changes of the controller's manipulated variable (→ fig. 50).

→ If large values are required for length of period and minimum switch-on time (to 65535 s = 18.2 h), you can use the special function block "U_ZSFB01_special_FB", page 317. However, this function block should not be used if short periods are required. This special function block is only capable of outputting pulses with a precision of 1 s.

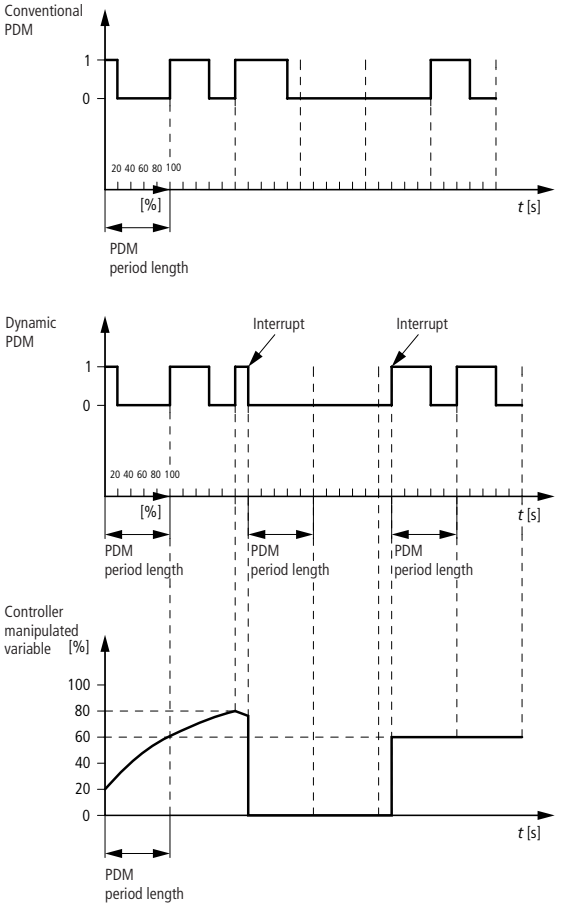


Figure 50: A comparison of dynamic and conventional PDM. Reaction times are shorter for dynamic PDM.

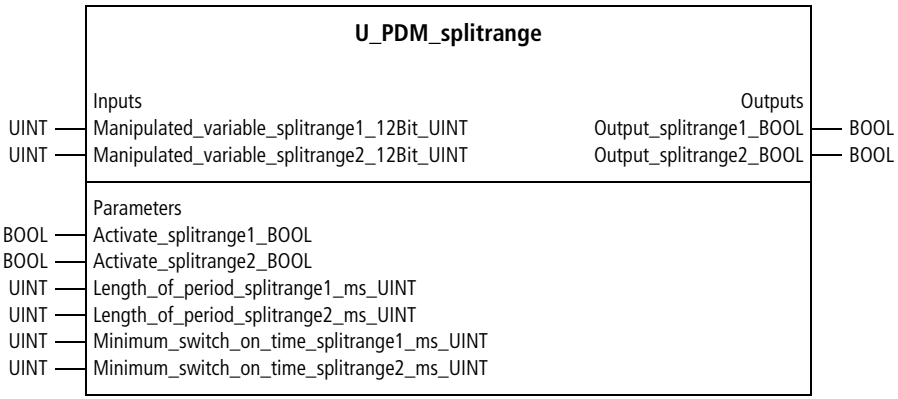
Example:
The application example links a PDM system with the PID controller of the same zone, by placing the 12-bit manipulated variable of the controller on the input of the PDM system. The length of period is set to 60 s and the minimum switch-on time to 3 s.

Application of the function block "U_PDM_dynamic_contactor" in the program "zone1PDM"

```
PROGRAM zone1PDM
VAR
    PDM_zone1 : U_PDM_DYNAMIC_CONTACTOR ;
    PID_controller_manipulated_variable_12Bit_UINT : UINT ;
    Enable_zone1 : BOOL ;
    Digital_output_0_0 : BOOL ;
END_VAR
LD  PID_controller_manipulated_variable_12Bit_UINT
ST  PDM_zone1.Manipulated_variable_12Bit_UINT
CAL PDM_zone1(
    Activate_BOOL :=Enable_zone1,
    Length_of_period_PDM_ms_UINT :=60000,
    Minimum_switch_on_time_PDM_ms_UINT :=3000,
    Output_BOOL=>Digital_output_0_0
)

END_PROGRAM
```

PDM for split range controllers

U_PDM_splitrange
Pulse Duration Modulation for Split Range Processes, Suitable for Contactors



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Manipulated_variable_splitrange1_ 12Bit_UINT	12-bit input variable for split range 1	0 to 4095
Manipulated_variable_splitrange2_ 12Bit_UINT	12-bit input variable for split range 2	0 to 4095
Parameters		
Activate_splitrange1_BOOL	Enables the PDM system for split range 1	0/1
Activate_splitrange2_BOOL	Enables the PDM system for split range 2	0/1
Length_of_period_splitrange1_ ms_UINT	Length of period [ms] for split range 1	0 to 65535
Length_of_period_splitrange2_ ms_UINT	Length of period [ms] for split range 2	0 to 65535
Minimum_switch_on_time_ splitrange1_ms_UINT	Minimum switch-on time [ms] for split range 1	0 to 65535
Minimum_switch_on_time_ splitrange2_ms_UINT	Minimum switch-on time [ms] for split range 2	0 to 65535
Outputs		
Output_splitrange1_BOOL	Digital output signal for split range 1	0/1
Output_splitrange2_BOOL	Digital output signal for split range 2	0/1

→ Do not make the ratio "period length/minimum switch-on time" too small. Otherwise, relatively large manipulated variables will be suppressed (→ section "Minimum switch-on time", page 310).

Description

The function block "U_PDM_splitrange" can be used to connect mechanical contactors. This function block can be linked with the function block "PID_split_range_controller". It is then linked with the two 12-bit manipulated variables (4095 = 100 %) of the controller for split range 1 (heating) and split range 2 (cooling). "Activate_splitrange1_ BOOL=1" or "Activate_splitrange2_BOOL=1" starts the corresponding part of the PDM system, and "Output_ splitrange1_BOOL" or "Output_splitrange2_BOOL" outputs one pulse (→ fig. 51) when the specified minimum switch-on time is exceeded. Deactivating split range 1 or 2 sets the corresponding output signal to zero and resets the current processes. Reactivation starts a new period. The length of period and minimum switch-on time can be entered in ms. The maximum input value of 65535 results in times of 65,535 s. Further explanations can be found at the beginning of chapter 5.

→ If large values are required for length of period and minimum switch-on time (to 65535 s = 18.2 h), you can use the special function block "U_ZSFB01_special_FB", page 317. However, this function block should not be used if short periods are required. This special function block is only capable of outputting pulses with a precision of 1 s.

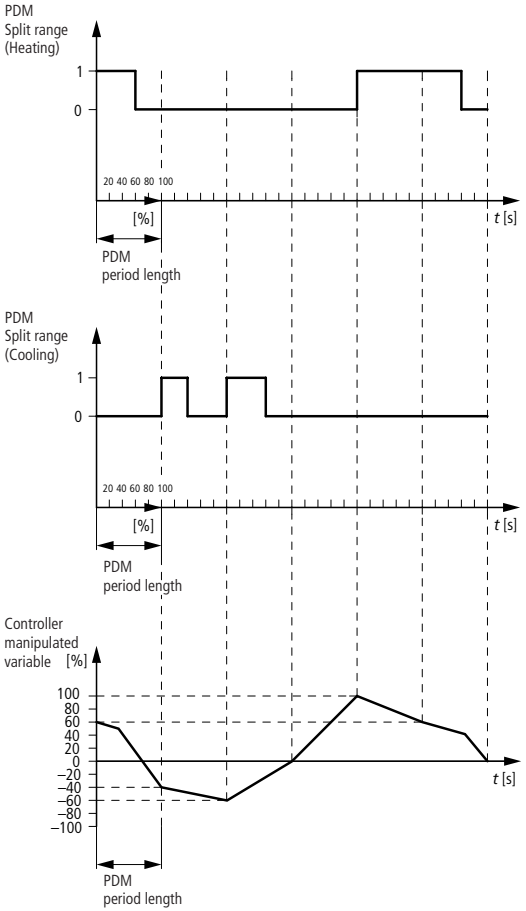


Figure 51: PDM for split range control (heating/cooling) as a function of the bipolar (positive/negative) manipulated variable of the controller

Example:
For a zone with heating and cooling actuators, the application example links a split range PDM system with the split range PID controller of the same zone, by placing the 12-bit manipulated variables of the controller on the input of the PDM system. For heating, the length of period is set to 60 s and the minimum switch-on time to 3 s.

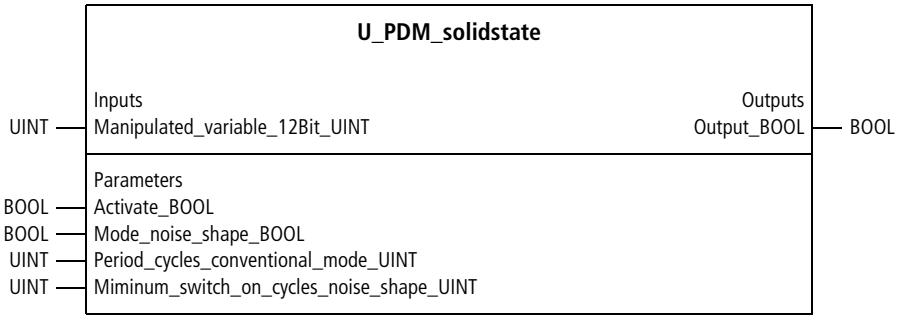
For cooling, the length of period is set to 20 s and the minimum switch-on time to 2 s.

Application of the function block "U_PDM_splitrange" in the program "Zone2PDM"

```
PROGRAM Zone2PDM
VAR
    PDM_heat_cool_zone2 : U_PDM_SPLITRANGE ;
    PID_controller_manipulated_variable_heat_zone2 : UINT ;
    PID_controller_manipulated_variable_cool_zone2 : UINT ;
    Enable_zone2 : BOOL ;
    Digital_output_heat : BOOL ;
    Digital_output_cool : BOOL ;
END_VAR

LD  PID_controller_manipulated_variable_heat_zone2
ST  PDM_heat_cool_zone2.Manipulated_variable_splitrange1_12Bit_UINT
LD  PID_controller_manipulated_variable_cool_zone2
ST  PDM_heat_cool_zone2.Manipulated_variable_splitrange2_12Bit_UINT
CAL PDM_heat_cool_zone2(
    Activate_splitrange1_BOOL :=Enable_zone2,
    Activate_splitrange2_BOOL :=Enable_zone2,
    Length_of_period_splitrange1_ms_UINT :=60000,
    Length_of_period_splitrange2_ms_UINT :=20000,
    Minimum_switch_on_time_splitrange1_ms_UINT :=3000,
    Minimum_switch_on_time_splitrange2_ms_UINT :=2000,
    Output_splitrange1_BOOL=>Digital_output_heat,
    Output_splitrange2_BOOL=>Digital_output_cool)

END_PROGRAM
```

PDM for solid-state relays

U_PDM_solidstate
Pulse Duration Modulation with Entry of Cycles (Conventional PDM Technique) or Minimum Cycles (Noise-Shape Technique), Suitable for Solid-State Relays



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Manipulated_variable_12Bit_UINT	12-bit input variable of the PDM system	0 to 4095
Parameters		
Activate_BOOL	Enables the PDM system	0/1
Mode_noise_shape_BOOL	Selects the PDM algorithm	0/1
Period_cycles_conventional_ mode_UINT	Period in number of PLC cycles	0 to 65535
Miminum_switch_on_cycles_ noise_shape_UINT	Minimum number of switched-on PLC cycles	0 to 65535
Outputs		
Output_BOOL	Digital output of the PDM system	0/1

→ Do not make the ratio "period length/minimum switch-on time" too small. Otherwise, relatively large manipulated variables will be suppressed (→ section "Minimum switch-on time", page 310).

Description

This function block is suitable for solid-state relays (electronically switching load relays). In contrast to the PDM function blocks described above, this function block relates the pulse durations to PLC cycles rather than time spans. The value assigned to "Manipulated_variable_12Bit_UINT" (4095 = 100 %) results in a switch-on time or switch-on cycles. Two different modes can be selected.

A conventional PDM algorithm, based on period cycles (→ fig. 52 below):

- The conventional PDM algorithm is selected with "Mode_noise_shape_BOOL=0".
- With the variables "Manipulated_variable_12Bit_UINT" and "Period_cycles_conventional_mode_UINT", a number of PLC cycles are calculated for switching on (pulse duration). In conjunction with the PLC cycle time, a switch-on time results.

Example:
Manipulated_variable_12Bit_UINT = 1300
(1300/4095 = 31.76 %)
Period_cycles_conventional_mode_UINT = 100
=> switch-on cycles = 31
a PLC cycle time of 100 ms results in a switch-on time of 3.1 s and a switch-off time of 6.9 s, for a period of 10 s.

Noise-shape technique (→ fig. 52 above):

- The noise-shape technique is selected with "mode_noise_shape_BOOL=1". A specific switch-on time is implemented with a precision of 12 bits. Switching on and off is distributed as widely as possible.

Examples:

- switch-on time = 50 %
  => The signal is switched on for one PLC cycle and switched off for one cycle (→ fig. 52 above).
- switch-on time = 66 %
  => The signal is switched on for two PLC cycles and switched off for one cycle.
- switch-on time = 25 %
  => The signal is switched on for one PLC cycle and switched off for three cycles.

If you enter a value not equal to 0 or 1 for the parameter "Miminum_switch_on_cycles_noise_shape_UINT", then the distribution of switch-on and switch-off phases will be implemented accordingly.

Examples:
Three minimum switch-on cycles are set for the noise-shape technique.

- switch-on time = 50 %
  => The signal is switched on for three PLC cycles and switched off for three cycles (→ fig. 52 above).
- switch-on time = 66 %
  => The signal is switched on for six PLC cycles and switched off for three cycles.
- switch-on time = 25 %
  => The signal is switched on for three PLC cycles and switched off for nine cycles.

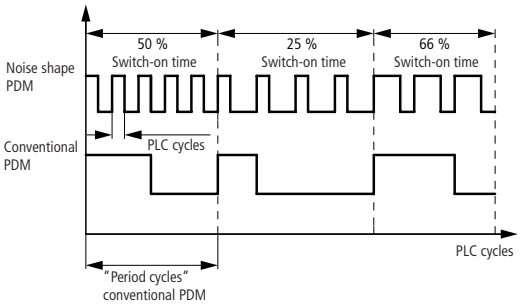


Figure 52: Comparison of conventional PDM with noise-shape PDM. The switch-on frequency is much greater for noise-shape PDM.

→ Both PDM modes relate to PLC cycles. For this reason, it is recommendable to generate equidistant PLC cycle times. This can be implemented with the function block "U_CYCS_cycletime_setpoint_value". For the mode "noise-shape technique", for example, a constant PLC cycle time of 18 ms is recommended. Due to the line frequency of 50 Hz, a PLC cycle time of 20 ms is poor and can result in surges.

Example:
The application example links a PDM system with the PID controller of the same zone by placing the 12-bit manipulated variable of the controller on the input of the PDM system. If you switch to the mode "conventional PDM technique" (Mode_noise_shape_BOOL=0), then the pulse duration is related to 100 PLC cycles and does not influence the noise-shape technique. No minimum switch-on times are set for the noise-shape technique. This maximises the switch-on frequency.

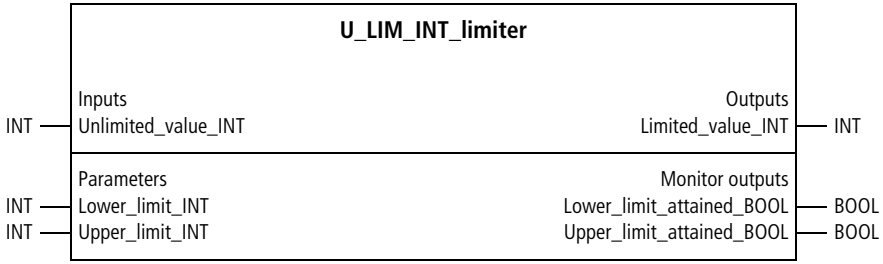
Application of the function block "U_PDM_solidstate" in the program "zone3PDM"

```
PROGRAM zone3PDM
VAR
    PDM_zone3 : U_PDM_SOLIDSTATE ;
    PID_controller_manipulated_variable_zone3 : UINT ;
    Enable_zone3 : BOOL ;
    Digital_output : BOOL ;
END_VAR

LD  PID_controller_manipulated_variable_zone3
ST  PDM_zone3.Manipulated_variable_12Bit_UINT
CAL PDM_zone3(
    Activate_BOOL :=Enable_zone3,
    Mode_noise_shape_BOOL :=0,
    Period_cycles_conventional_mode_UINT :=100,
    Miminum_switch_on_cycles_noise_shape_UINT :=0,
    Output_BOOL=>Digital_output
)

END_PROGRAM
```

6 Signal Filters, Processing and Limiting

Signal limiting function blocks

U_LIM_INT_limiter
Limiting integer values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Unlimited_value_INT	Unlimited input value	–32768 to 32767
Parameters		
Lower_limit_INT	Lower limit	–32768 to 32767
Upper_limit_INT	Upper limit	–32768 to 32767
Outputs		
Limited_value_INT	Limited output value	–32768 to 32767
Monitor outputs		
Lower_limit_attained_BOOL	Message: value reached or dropped below the lower limit	0/1
Upper_limit_attained_BOOL	Message: value reached or exceeded the upper limit	0/1

Description

This function block limits the input value to a range between a lower and upper limit (→ fig. 53). BOOL variables indicate when the value has reached the limits, dropped below the lower limit, or exceeded the upper limit.

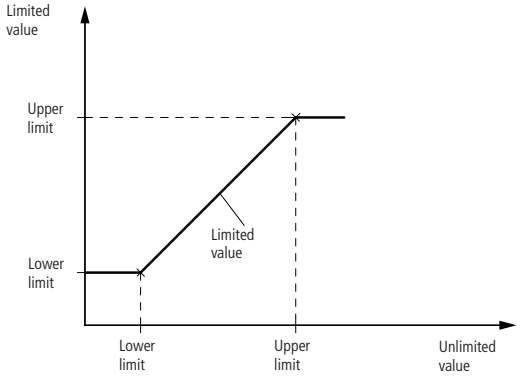


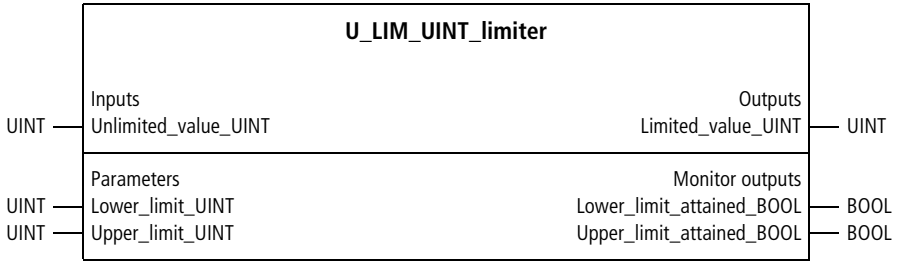
Figure 53: The input values are limited to specified limiting values

Example:
The application example "BgrVar_a" limits the variable "Var_a" to a range between –4095 and 4095.

Application of the function block "U_LIM_INT_limiter" in the program "LimVar_a"

```
PROGRAM LimVar_a
VAR
  LIM_INT_LIMITER : U_LIM_INT_LIMITER ;
  var_a : INT ;
  var_a_limited_13Bit : INT ;
END_VAR

CAL LIM_INT_LIMITER(
  Unlimited_value_INT :=var_a,
  Lower_limit_INT :=4095,
  Upper_limit_INT :=-4095,
  Limited_value_INT=>var_a_limited_13Bit
)

END_PROGRAM
```

U_LIM_UINT_limiter
Limiting Unsigned Integer Values



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Unlimited_value_UINT	Unlimited input value	0 to 65535
Parameters		
Lower_limit_UINT	Lower limit	0 to 65535
Upper_limit_UINT	Upper limit	0 to 65535
Outputs		
Limited_value_UINT	Limited output value	0 to 65535
Monitor outputs		
Lower_limit_attained_BOOL	Message: the value has reached or dropped below the lower limit	0/1
Upper_limit_attained_BOOL	Message: the value has reached or exceeded the upper limit	0/1

Description

This function block limits the input value to a range between a lower and upper limit (→ fig. 53). BOOL variables indicate when the value has reached the limits, dropped below the lower limit, or exceeded the upper limit.

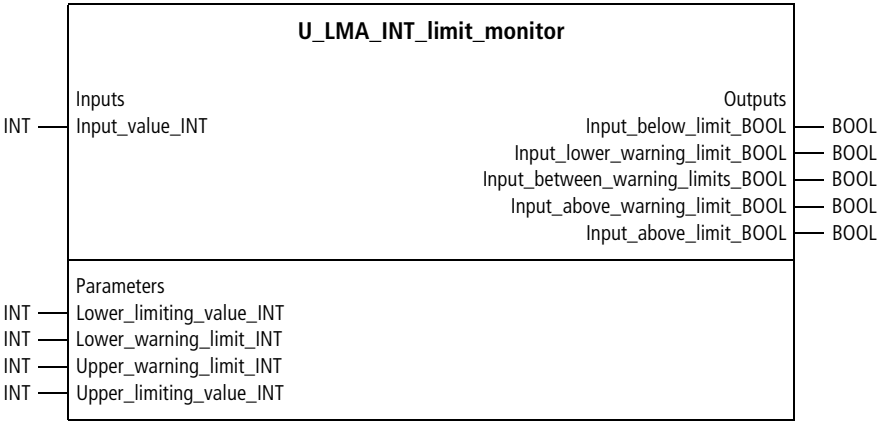
Example:
The application example "BgrVar_b" limits the variable "Var_b" to a range between 0 and 100.

Application of the function block "U_LIM_UINT_limiter" in the program "LimVar_b"

```
PROGRAM LimVar_b
VAR
  LIM_UINT_LIMITER : U_LIM_UINT_LIMITER ;
  var_b : UINT ;
  var_b_smaller_100 : UINT ;
END_VAR

CAL LIM_UINT_LIMITER(
  Unlimited_value_UINT :=var_b,
  Lower_limit_UINT :=0,
  Upper_limit_UINT :=100,
  Limited_value_UINT=>var_b_smaller_100
)

END_PROGRAM
```

Monitors for limiting values and tolerance bands

U_LMA_INT_limit_monitor
Limiting Value Monitor for Absolute Boundary and Warning Values (Integers)



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_INT	Input value	–32768 to 32767
Parameters		
Lower_limiting_value_INT	Lower limiting value	–32768 to 32767
Lower_warning_limit_INT	Lower warning limit	–32768 to 32767
Upper_warning_limit_INT	Upper warning limit	–32768 to 32767
Upper_limiting_value_INT	Upper limiting value	–32768 to 32767

Designation	Significance	Value range
Outputs		
Input_below_limit_BOOL	The input value is below the lower limiting value	0/1
Input_lower_warning_limit_BOOL	The input value is below the lower warning limit	0/1
Input_between_warning_limits_BOOL	The input value is between the warning limits	0/1
Input_above_warning_limit_BOOL	The input value is above the upper warning limit	0/1
Input_above_limit_BOOL	The input value is above the upper limiting value	0/1

Description

This function block indicates if the input value exceeds or drops below absolute warning limits or limiting values (→ fig. 54). It also indicates when the input value is between the warning limits (window of tolerance).

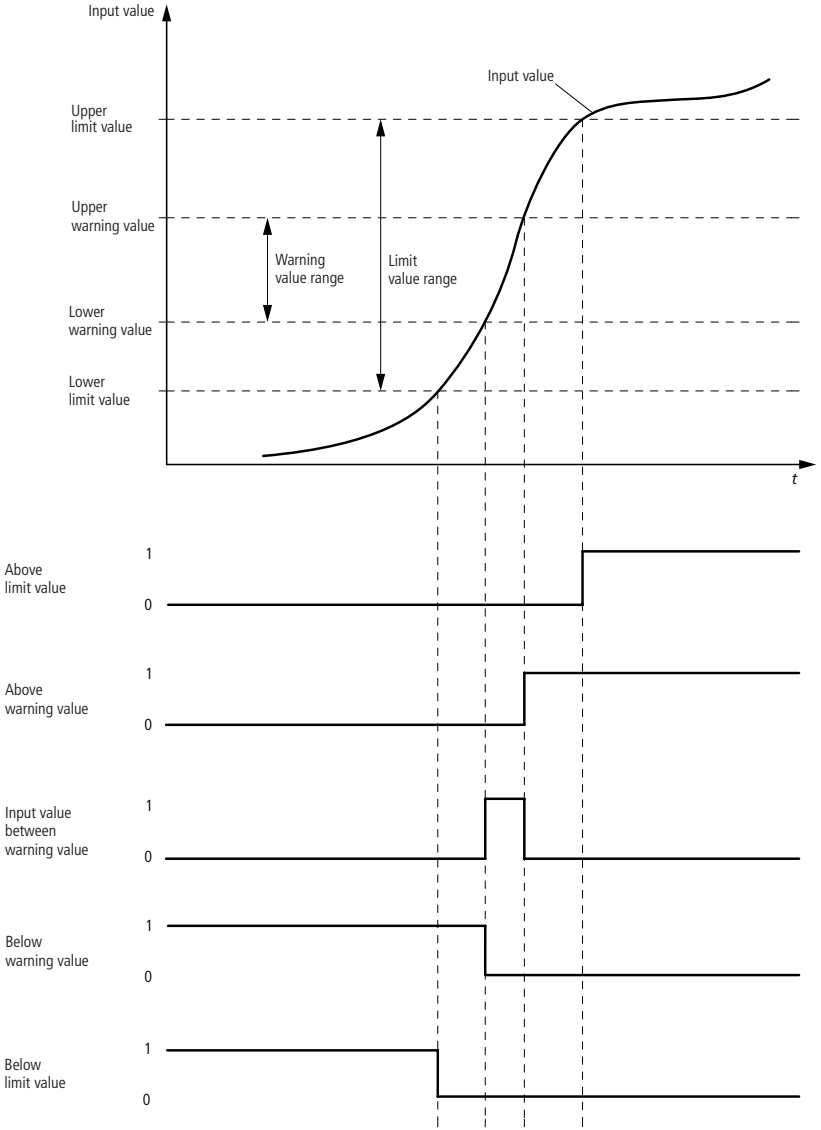


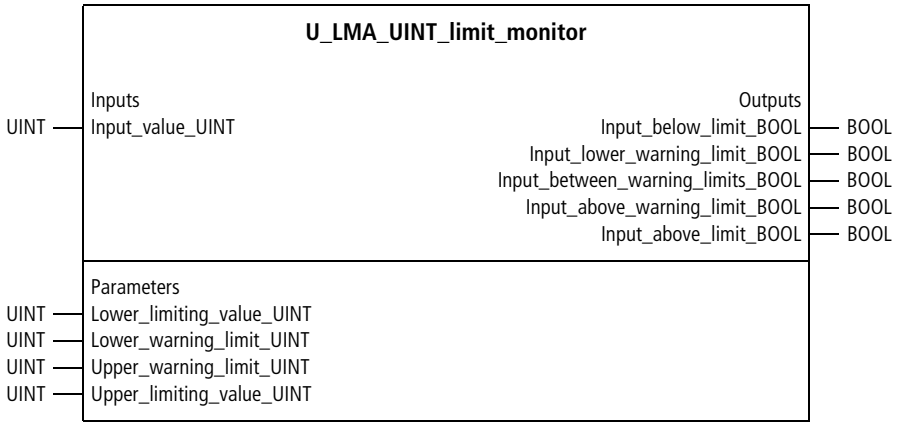
Figure 54: Exceeding and dropping below absolute warning limits and limiting values

Example:
The application example "Gr_u_War" checks if the actual value is between the specified absolute limiting values and warning limits.

Application of the function block "U_LMA_INT_limit_monitor" in the program "LMA_01"

```
PROGRAM LMA_01
VAR
  LMA_INT_LIMIT_MONITOR : U_LMA_INT_LIMIT_MONITOR ;
  actual_value : INT ;
  status_signal_low : BOOL ;
  status_signal_low_warning : BOOL ;
  status_signal_OK : BOOL ;
  status_signal_high_warning : BOOL ;
  status_signal_high : BOOL ;
END_VAR

CAL LMA_INT_LIMIT_MONITOR(
  Input_value_INT :=actual_value,
  Lower_limiting_value_INT :=1500,
  Lower_warning_limit_INT :=1900,
  Upper_warning_limit_INT :=2050,
  Upper_limiting_value_INT :=2100,
  Input_below_limit_BOOL=>status_signal_low,
  Input_below_warning_limit_BOOL=>status_signal_low_warning,
  Input_between_warning_limits_BOOL=>status_signal_OK,
  Input_above_warning_limit_BOOL=>status_signal_high_warning,
  Input_above_limit_BOOL=>status_signal_high)

END_PROGRAM
```

U_LMA_UINT_limit_monitor
Limiting Value Monitor for Absolute Limiting Values and Warning Limits (Unsigned Integers)



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_UINT	Input value	0 to 65535
Parameters		
Lower_limiting_value_UINT	Lower limiting value	0 to 65535
Lower_warning_limit_UINT	Lower warning limit_	0 to 65535
Upper_warning_limit_UINT	Upper warning limit	0 to 65535
Upper_limiting_value_UINT	Upper limiting value	0 to 65535

Designation	Significance	Value range
Outputs		
Input_below_limit_BOOL	The input value is below the lower limiting value	0/1
Input_lower_warning_limit_BOOL	The input value is below the lower warning limit	0/1
Input_between_warning_limits_BOOL	The input value is between the warning limits	0/1
Input_above_warning_limit_BOOL	The input value is above the upper warning limit	0/1
Input_above_limit_BOOL	The input value is above the upper imiting value	0/1

Description

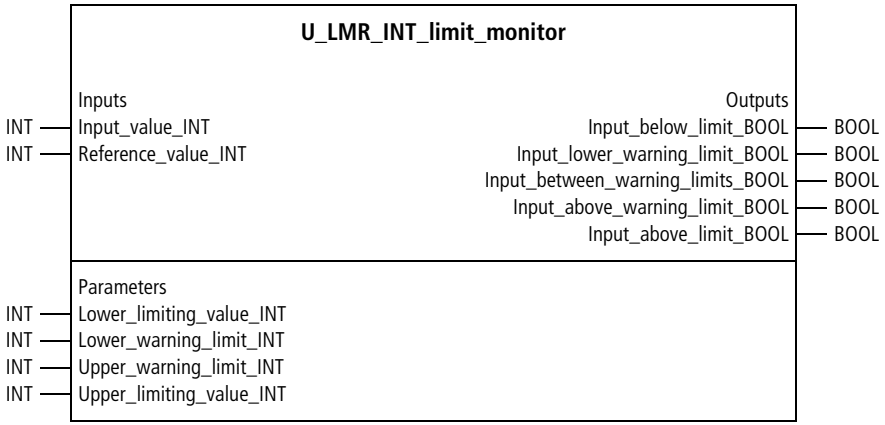
This function block indicates if the input value exceeds or drops below absolute warning limits or limiting values (→ fig. 54). It also indicates when the input value is between the warning limits (window of tolerance).

Example:
The application example "Gr_u_W" checks if the actual value is between the specified absolute limiting values and warning limits.

Application of the function block "U_LMA_UINT_limit_monitor" in the program "LMA_02"

```
PROGRAM LMA_02
VAR
  LMA_UINT_LIMIT_MONITOR : U_LMA_UINT_LIMIT_MONITOR ;
  actual_value : UINT ;
  status_signal_low : BOOL ;
  status_signal_low_warning : BOOL ;
  status_signal_OK : BOOL ;
  status_signal_high_warning : BOOL ;
  status_signal_high : BOOL ;
END_VAR

CAL LMA_UINT_LIMIT_MONITOR(
  Input_value_UINT :=actual_value,
  Lower_limiting_value_UINT :=1500,
  Lower_warning_limit_UINT :=1900,
  Upper_warning_limit_UINT :=2050,
  Upper_limiting_value_UINT :=2100,
  Input_below_limit_BOOL=>status_signal_low,
  Input_below_warning_limit_BOOL=>status_signal_low_warning,
  Input_between_warning_limits_BOOL=>status_signal_OK,
  Input_above_warning_limit_BOOL=>status_signal_high_warning,
  Input_above_limit_BOOL=>status_signal_high)

END_PROGRAM
```

U_LMR_INT_limit_monitor
Limiting Value Monitor for Relative Limiting Values and Warning Limits (Integers)



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_INT	Input value	–32768 to 32767
Reference_value_INT	Reference value	
Parameters		
Lower_limiting_value_INT	Relative lower limiting value	–32768 to 32767
Lower_warning_limit_INT	Relative lower warning limit	–32768 to 32767
Upper_warning_limit_INT	Relative upper warning limit	–32768 to 32767
Upper_limiting_value_INT	Relative upper limiting value	–32768 to 32767

Designation	Significance	Value range
Outputs		
Input_below_limit_BOOL	The input value is below the lower limiting value	0/1
Lower_warning_limit_BOOL	The input value is below the lower warning limit	0/1
Input_between_warning_limits_BOOL	The input value is between the warning limits	0/1
Input_above_warning_limit_BOOL	The input value is above the upper warning limit	0/1
Input_above_limit_BOOL	The input value is above the upper limiting value	0/1

Description

This function block indicates if the input value exceeds or drops below relative warning limits or limiting values (→ fig. 55). It also indicates when the input value is between the warning limits (window of tolerance). The warning limits and limiting values are calculated by adding and subtracting the relative values from a reference value (→ fig. 55 and application example).

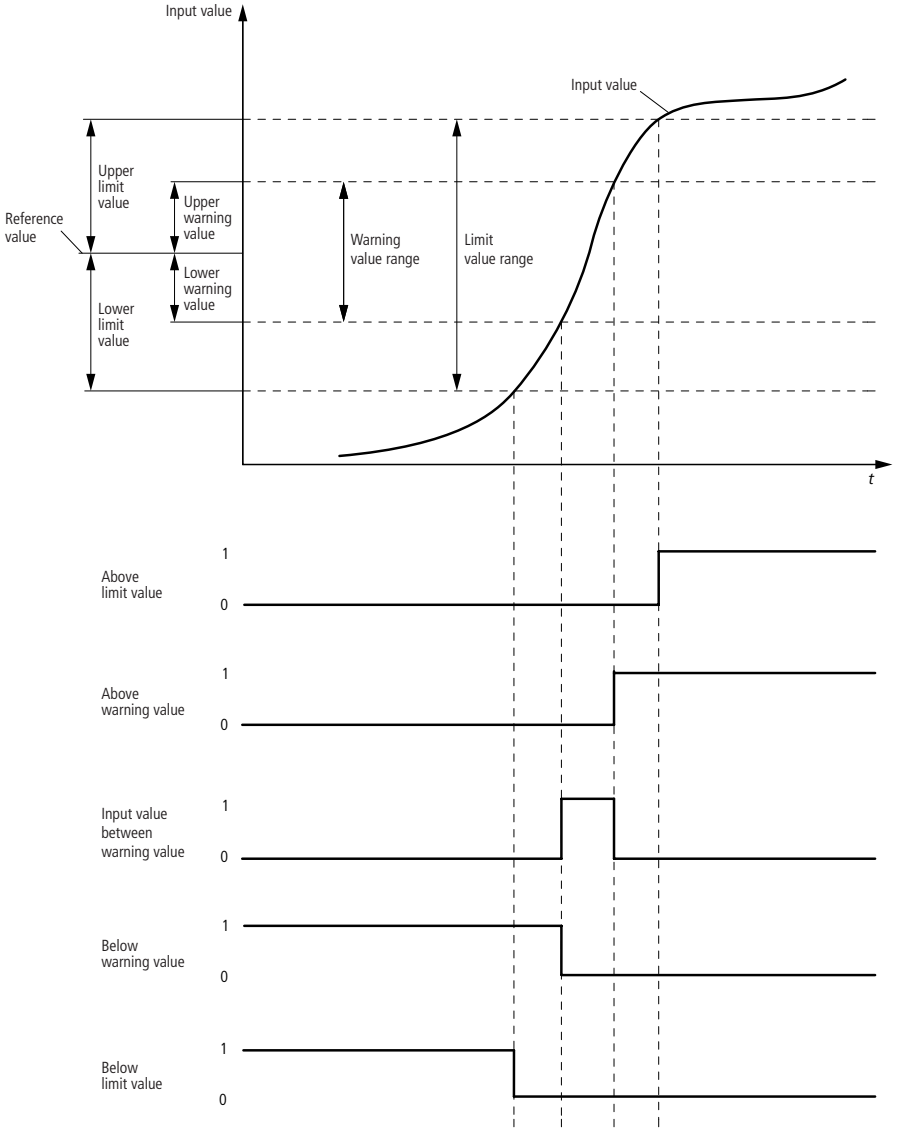


Figure 55: Exceeding and dropping below relative warning limits and limiting values

Example:
The application example "Rel_Gr_W" checks if the actual value is within the specified relative limiting values and warnings limits, relative to the setpoint. A setpoint of 2000, for example, results in the following limiting values and warnings limits:

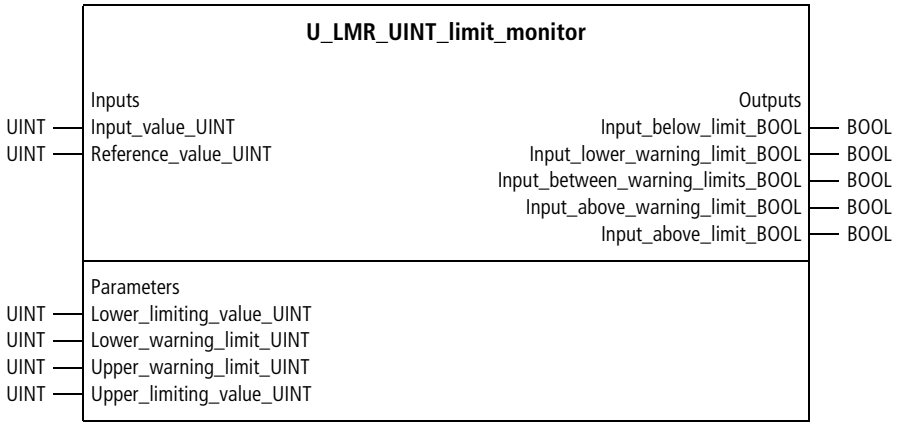
- Lower limiting value = 2000 – 100 = 1900
- Lower warning value = 2000 – 50 = 1950
- Upper warning limit = 2000 + 40 = 2040
- Upper limiting value = 2000 + 80 = 2080

Application of the function block "U_LMR_INT_limit_monitor" in the program "LMR_01"

```
PROGRAM LMR_01
VAR
  LMR_INT_LIMIT_MONITOR : U_LMR_INT_LIMIT_MONITOR ;
  actual_value : INT ;
  setpoint_value : INT ;
  status_signal_low : BOOL ;
  status_signal_low_warning : BOOL ;
  status_signal_OK : BOOL ;
  status_signal_high_warning : BOOL ;
  status_signal_high : BOOL ;
END_VAR
```

```
CAL LMR_INT_LIMIT_MONITOR(
  Input_value_INT :=actual_value,
  Reference_value_INT :=setpoint_value,
  Lower_limiting_value_INT :=100,
  Lower_warning_limit_INT :=50,
  Upper_warning_limit_INT :=40,
  Upper_limiting_value_INT :=80,
  Input_below_limit_BOOL=>status_signal_low,
  Input_below_warning_limit_BOOL=>status_signal_low_warning,
  Input_between_warning_limits_BOOL=>status_signal_OK,
  Input_above_warning_limit_BOOL=>status_signal_high_warning,
  Input_above_limit_BOOL=>status_signal_high)
END_PROGRAM
```

U_LMR_UINT_limit_monitor
Limiting Value Monitor for Relative Limiting Values and Warning Limits (Unsigned Integers)



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_UINT	Input value	0 to 65535
Reference_value_UINT	Reference value	
Parameters		
Lower_limiting_value_UINT	Relative lower limiting value	0 to 65535
Lower_warning_limit_UINT	Relative lower warning limit	0 to 65535
Upper_warning_limit_UINT	Relative upper warning limit	0 to 65535
Upper_limiting_value_UINT	Relative upper limiting value	0 to 65535

Designation	Significance	Value range
Outputs		
Input_below_limit_BOOL	The input value is below the lower limiting value	0/1
Input_lower_warning_limit_BOOL	The input value is below the lower warning limit	0/1
Input_between_warning_limits_BOOL	The input value is between the warning limits	0/1
Input_above_warning_limit_BOOL	The input value is above the upper warning limit	0/1
Input_above_limit_BOOL	The input value is above the upper limiting value	0/1

Description

This function block indicates if the input value exceeds or drops below relative warning limits or limiting values (→ fig. 55). It also indicates when the input value is between the warning limits (window of tolerance). The warning limits and limiting values are calculated by adding and subtracting the relative values from a reference value (→ fig. 55 and application example).

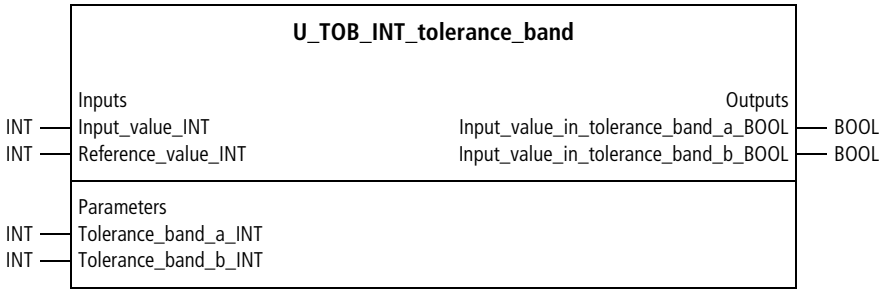
Example:
The application example "Rel_GrWa" checks if the actual value is within the specified relative limiting values and warnings limits, relative to the setpoint. A setpoint of 1000, for example, results in the following limiting values and warnings limits:

- Lower limiting value = 1000 – 100 = 900
- Lower warning value = 1000 – 50 = 950
- Upper warning limit = 1000 + 40 = 1040
- Upper limiting value = 1000 + 80 = 2080

Application of the function block "U_LMR_UINT_limit_monitor" in the program "LMR_02"

```
PROGRAM LMR_02
VAR
  LMR_UINT_LIMIT_MONITOR : U_LMR_UINT_LIMIT_MONITOR ;
  actual_value : UINT ;
  setpoint_value : UINT ;
  status_signal_low : BOOL ;
  status_signal_low_warning : BOOL ;
  status_signal_OK : BOOL ;
  status_signal_high_warning : BOOL ;
  status_signal_high : BOOL ;
END_VAR

CAL LMR_UINT_LIMIT_MONITOR(
  Input_value_UINT :=actual_value,
  Reference_value_UINT :=setpoint_value,
  Lower_limiting_value_UINT :=100,
  Lower_warning_limit_UINT :=50,
  Upper_warning_limit_UINT :=40,
  Upper_limiting_value_UINT :=80,
  Input_below_limit_BOOL=>status_signal_low,
  Input_below_warning_limit_BOOL=>status_signal_low_warning,
  Input_between_warning_limits_BOOL=>status_signal_OK,
  Input_above_warning_limit_BOOL=>status_signal_high_warning,
  Input_above_limit_BOOL=>status_signal_high)

END_PROGRAM
```

U_TOB_INT_tolerance_band
Tolerance Band Monitor for Relative Tolerance Bands (Integers)



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_INT	Input value	–32768 to 32767
Reference_value_INT	Reference value	–32768 to 32767
Parameters		
Tolerance_band_a_INT	Tolerance band a	–32768 to 32767
Tolerance_band_b_INT	Tolerance band b	–32768 to 32767
Outputs		
Input_value_in_tolerance_band_a_BOOL	The input value is within the tolerance band a	0/1
Input_value_in_tolerance_band_b_BOOL	The input value is within the tolerance band b	0/1

Description

This function block indicates if the input value is within tolerance bands a and b (→ fig. 56). The tolerance bands are calculated by adding and subtracting from "Reference_value_INT" and "Tolerance_band_a_INT" or "Tolerance_band_b_INT" (→ fig. 56 and application example).

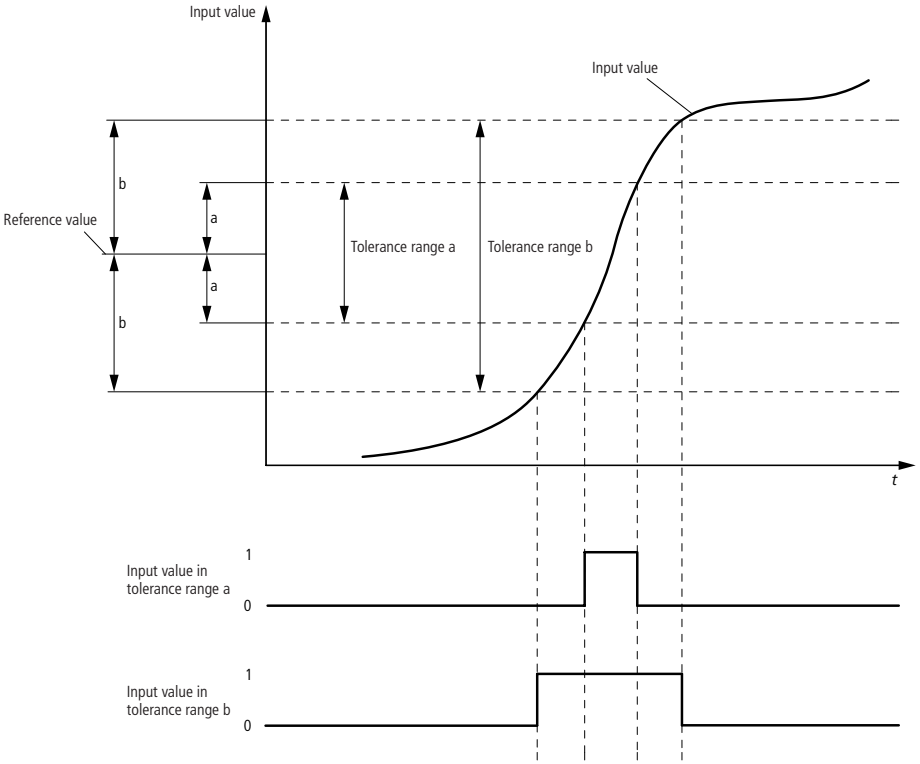


Figure 56: Exceeding and dropping below tolerance bands

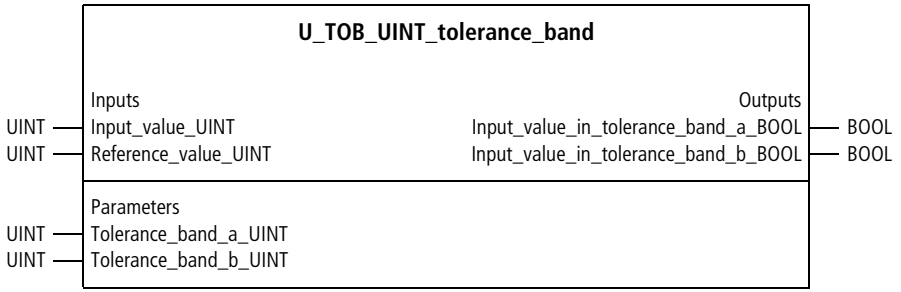
Example:
The application example "TOB_01" checks if the actual value is within a narrow and/or a wide tolerance band around the setpoint. A setpoint of 1000, for example, results in the following tolerance bands:

- Tolerance band a = 980 to 1020
- Tolerance band b = 950 to 1050

Application of the function block "U_TOB_INT_tolerance_band" in the program "TOB_01"

```
PROGRAM TOB_01
VAR
    TOB_INT_TOLERANCE_BAND : U_TOB_INT_TOLERANCE_BAND ;
    actual_value : INT ;
    setpoint_value : INT ;
    actual_value_in_tolerance_band_a : BOOL ;
    actual_value_in_tolerance_band_b : BOOL ;
END_VAR

CAL TOB_INT_TOLERANCE_BAND(
    Input_value_INT :=actual_value,
    Reference_value_INT :=setpoint_value,
    Tolerance_band_a_INT :=20,
    Tolerance_band_b_INT :=50)
LD  TOB_INT_TOLERANCE_BAND.Input_value_in_tolerance_band_a_BOOL
ST  actual_value_in_tolerance_band_a
LD  TOB_INT_TOLERANCE_BAND.Input_value_in_tolerance_band_b_BOOL
ST  actual_value_in_tolerance_band_b

END_PROGRAM
```

U_TOB_UINT_tolerance_band
Tolerance Band Monitor for Relative Tolerance Bands (Unsigned Integers)



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_UINT	Input value	0 to 65535
Reference_value_UINT	Reference value	0 to 65535
Parameters		
Tolerance_band_a_UINT	Tolerance band a	0 to 65535
Tolerance_band_b_UINT	Tolerance band b	0 to 65535
Outputs		
Input_value_in_tolerance_band_a_BOOL	The input value is within the tolerance band a	0/1
Input_value_in_tolerance_band_b_BOOL	The input value is within the tolerance band b	0/1

Description

The function block indicates if the input value is within the tolerance bands a and b (→ fig. 56). The tolerance bands are calculated by adding and subtracting "Reference_value_UINT" and "Tolerance_band_a_UINT" or "Tolerance_band_b_UINT" (→ fig. 56 and application example).

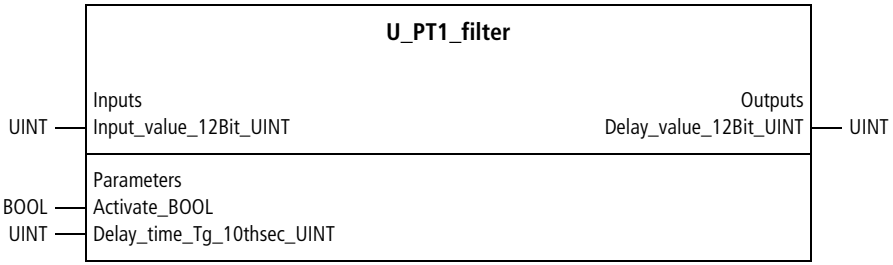
Example:
The application example "TOB_02" checks if the actual value is within a narrow and/or a wide tolerance band around the setpoint. A setpoint of 1000, for example, results in the following tolerance bands:

- Tolerance band a = 920 to 1080
- Tolerance band b = 950 to 1050

Application of the function block "U_TOB_UINT_tolerance_band" in the program "TOB_02"

```
PROGRAM TOB_02
VAR
    TOB_UINT_TOLERANCE_BAND : U_TOB_UINT_TOLERANCE_BAND ;
    actual_value : UINT ;
    setpoint_value : UINT ;
    actual_value_in_tolerance_band_a : BOOL ;
    actual_value_in_tolerance_band_b : BOOL ;
END_VAR

CAL TOB_UINT_TOLERANCE_BAND(
    Input_value_UINT :=actual_value,
    Reference_value_UINT :=setpoint_value,
    Tolerance_band_a_UINT :=80,
    Tolerance_band_b_UINT :=50)
LD  TOB_UINT_TOLERANCE_BAND.Input_value_in_tolerance_band_a_BOOL
ST  actual_value_in_tolerance_band_a
LD  TOB_UINT_TOLERANCE_BAND.Input_value_in_tolerance_band_b_BOOL
ST  actual_value_in_tolerance_band_b
END_PROGRAM
```

Signal smoothing filters U_PT1_filter
PT1 Filter for Signal Smoothing



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_12Bit_UINT	Input value	0 to 4095
Parameters		
Activate_BOOL	Activates the function block	0/1
Delay_time_Tg_10thsec_UINT	Delay time Tg	0 to 65535
Outputs		
Delay_value_12Bit_UINT	PT1-delayed output value	0 to 4095

Description

This function block is used to smooth noisy signals. "Delay_time_Tg_10thsec_UINT" is used to set the time span for smoothing (→ fig. 57). The delay time should not be greater than necessary. Otherwise, the signals will be delayed more than necessary for smoothing (signal delay is an unavoidable "side effect" of signal smoothing). "Activate_BOOL=1" starts the function block. "Activate_BOOL=0" resets it. The first time the function block is called after a reset (or the first PLC cycle), "Delay_value_12Bit_UINT" is initialised with the input value (PT1-delay does not begin at zero), in order to accelerate the start of PT1.

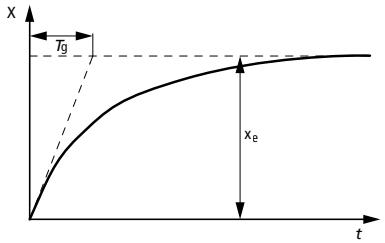


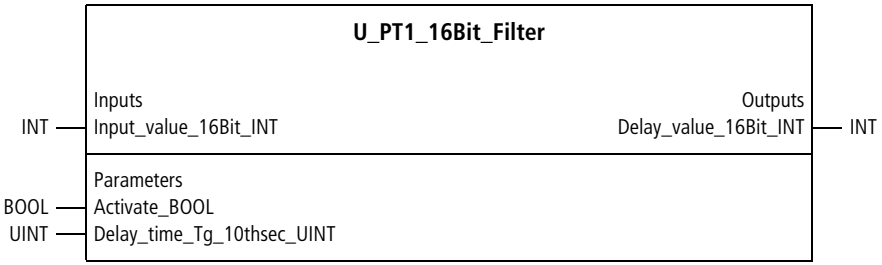
Figure 57: PT1-delay of the filter, as a function of T_g and the spontaneous change of the input X_e

Example:
The application program "GlaetPT1" smooths the noisy input value for half a second.

Application of the function block "U_PT1_filter" in the program "Smooth01"

```
PROGRAM Smooth01
VAR
  PT1_FILTER_zone1 : U_PT1_FILTER ;
  actual_value : UINT ;
  actual_value_smoothed : UINT ;
END_VAR

CAL PT1_FILTER_zone1(
  Input_value_12Bit_UINT :=actual_value,
  Activate_BOOL :=1,
  Delay_time_Tg_10thsec_UINT :=5,
  Delay_value_12Bit_UINT=>actual_value_smoothed)

END_PROGRAM
```

U_PT1_16Bit_Filter
PT1 Filter for Signal Smoothing with a 16-Bit Input Value



Function block prototype

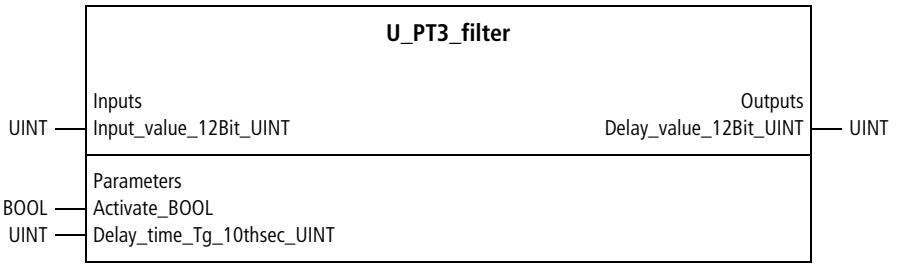
Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_16Bit_INT	Input value	–32768 to 32767
Parameters		
Activate_BOOL	Activates the function block	0/1
Delay_time_Tg_10thsec_UINT	Delay time Tg [10thsec]	0 to 65535
Outputs		
Delay_value_16Bit_INT	PT1-delayed output value	–32768 to 32767

Description

See the function block "U_PT1_Filter". The only difference to the function block "U_PT1_Filter" is that this function block has 16 Bit-input variables (instead of 12 Bit).

U_PT3_filter
PT3 Filter for Signal Smoothing



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_12Bit_UINT	Input value	0 to 4095
Parameters		
Activate_BOOL	Activates the function block	0/1
Delay_time_Tg_10thsec_UINT	Delay time Tg	0 to 65535
Outputs		
Delay_value_12Bit_UINT	PT3-delayed output value	0 to 4095

Description

This function block is used to smooth noisy signals. "Delay_time_Tg_10thsec_UINT" is used to set the time span for smoothing (→ fig. 58). The delay time should not be greater than necessary. Otherwise, the signals will be delayed more than necessary for smoothing (signal delay is an unavoidable "side effect" of signal smoothing). "Activate_BOOL=1" starts the function block. "Activate_BOOL=0" resets it. The first time the function block is called after a reset (or the first PLC cycle), "Delay_value_12Bit_UINT" is initialised with the input value (PT3-delay does not begin at zero), in order to accelerate the start of PT3.

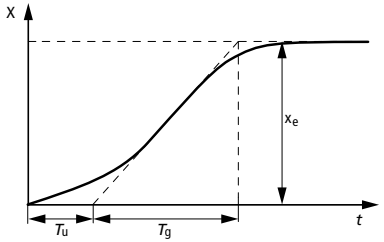


Figure 58: PT3-delay of the filter as a function of T_g and the spontaneous change of the input X_e. T_u (not adjustable) results from T_g and the order of the PTn filter.

→ The PT3 filter is used to smooth large, high-frequency interfering signals. Usually, however, the PT1 filter (→ above) suffices to smooth the signals and should be used instead of the PT3 filter. The disadvantages of the PT3 filter compared to the PT1 filter:

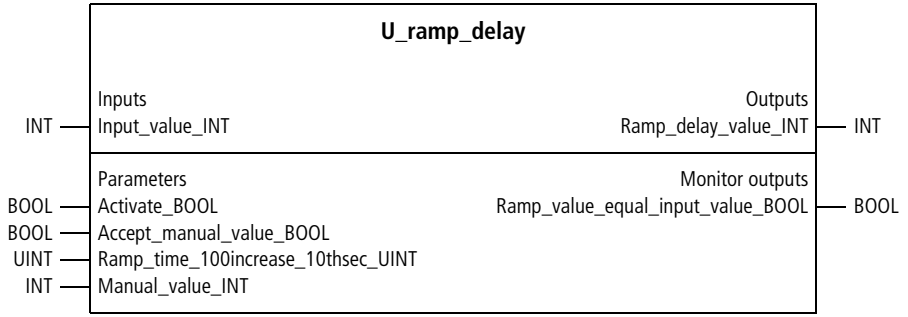
- larger cycle time required
- longer code and more data

Example:
The application program "Glaet_PT3" smooths the noisy actual value for ten seconds.

Application of the function block "U_PT3_filter" in the program "Smooth03"

```
PROGRAM Smooth03
VAR
  PT3_FILTER_zone1 : U_PT3_FILTER ;
  actual_value : UINT ;
  actual_value_smoothed : UINT ;
END_VAR

CAL PT3_FILTER_zone1(
  Input_value_12Bit_UINT :=actual_value,
  Activate_BOOL :=1,
  Delay_time_Tg_10thsec_UINT :=100,
  Delay_value_12Bit_UINT=>actual_value_smoothed)
END_PROGRAM
```

U_ramp_delay
Ramp Delay of an Input Value



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_INT	Input value	–32768 to 32767
Parameters		
Activate_BOOL	Activate the function block	0/1
Accept_manual_value_BOOL	Accept the manual value	0/1
Ramp_time_100increase_ 10tels_UINT	Ramp time required for an increase of 100 increments, in tenths of a second	0 to 65535
Manual_value_INT	Manual value for smooth acceptance	–32768 to 32767
Outputs		
Ramp_delay_value_INT	Ramp delay value	–32768 to 32767
Monitor outputs		
Ramp_value_equal_input_ value_BOOL	Indication: Ramp value same as input value	0/1

Description

The function block delays the input value according to a maximum ramp increase (similar to a PT1 filter). The ramp increase can be defined via "Ramp_time_100increase_ 10thsec_UINT ". The "Ramp_delay_value_INT " will increase (or decrease) by no more than 100 increments within the specified ramp time.

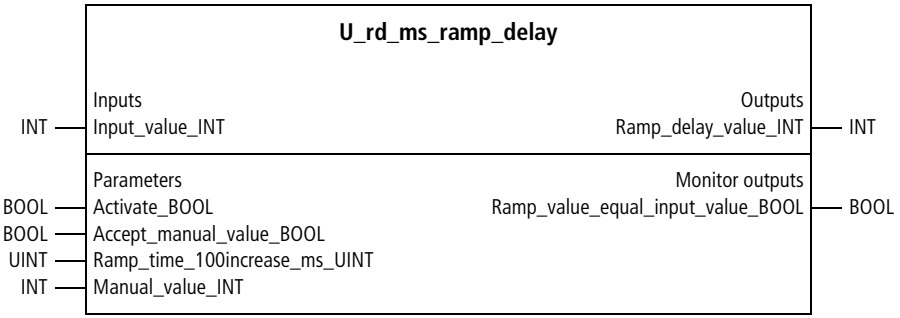
Example:
"Ramp_time_100increase_10thsec_UINT = 200" was entered. This means that the ramp delay value will increase by 100 increments every 20 s if "Input_value_INT" is greater than "Ramp_delay_value_INT ". It will decrease by 100 increments every 20 s if "Input_value_INT" is less than "Ramp_delay_value_INT ".

If the input value and the (ramp) delayed vale are the same, "Ramp_value_equal_input_value__BOOL" will be "1", otherwise "0". "Accept_manual_value_BOOL =1" causes "Manual_value_INT " to be accepted.

Application of the function block "U_ramp_delay" in the program "MV_delay"

```
PROGRAM MV_delay
VAR
    RAMP_DELAY: U_RAMP_DELAY ;
    ManipulatedVariable_12Bit_UINT : UINT ;
    ManipulatedVariable_delayed_12Bit_UINT : UINT ;
END_VAR

LD  Manipulated_variable_12Bit_UINT
UINT_TO_INT
ST  RAMP_DELAY.input_value_INT
CAL Rampenverzoegerung(
    Activate_BOOL :=1,
    Accept_manual_value_BOOL :=0,
    Ramp_time_100increase_10thsec_UINT :=200,
    Manual_value_INT :=0
)
LD  RAMP_DELAY.ramp_delay_value_INT
INT_TO_UINT
ST  Manipulated_variable_delayed_12Bit_UINT

END_PROGRAM
```

U_rd_ms_ramp_delay
Ramp Delay of an Input Value with Input of the Ramp Time in ms



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_value_INT	Input value	–32768 to 32767
Parameters		
Activate_BOOL	Activates the function block	0/1
Accept_manual_value_BOOL	Accept the manual value	0/1
Ramp_time_100increase_ms_UINT	Ramp time required for an increase of 100 increments, in ms.	0 to 65535
Manual_value_INT	Manual value for smooth acceptance	–32768 to 32767
Outputs		
Ramp_delay_value_INT	Ramp delay value	–32768 to 32767
Monitor outputs		
Ramp_value_equal_input_value_BOOL	Indication: Ramp value same as input value	0/1

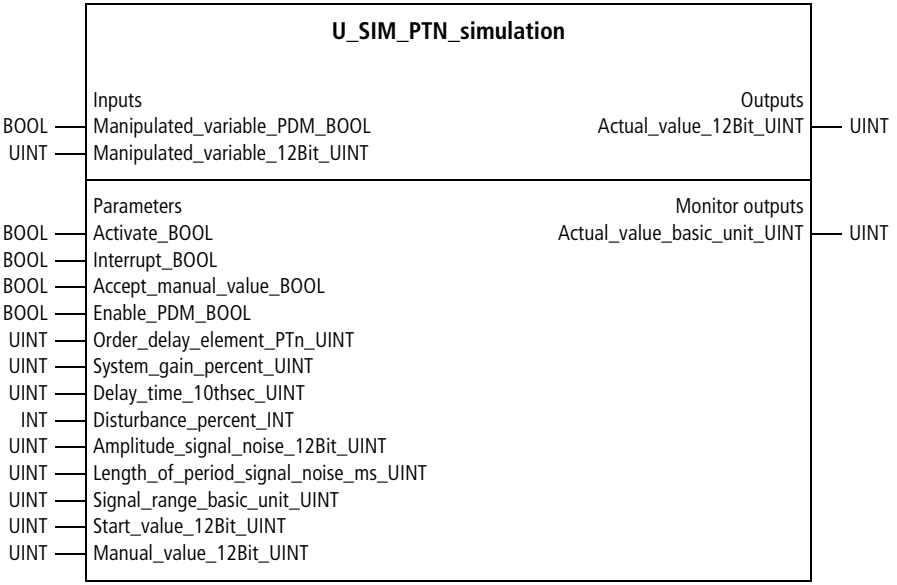
Description

See the function block "U_ramp_delay". The only difference to the function block "U_ramp_delay" is that the ramp time can be specified in ms.

7 System Simulations

Simulation of a PTn control system

U_SIM_PTN_simulation
Simulation of a PTn system with input of the system order



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Manipulated_variable_PDM_BOOL	Pulse duration modulation input	0/1
Manipulated_variable_12Bit_UINT	Analogue manipulated variable input	0 to 4095
Parameters		
Activate_BOOL	Enables the simulation	0/1
Interrupt_BOOL	Interrupts the simulation	0/1
Accept_manual_value_BOOL	Accepts the manual value	0/1
Enable_PDM_BOOL	Enables the PDM input	0/1
Order_delay_element_PTn_UINT	Order of the PTn system	0 to 10
Delay_time_10thsec_UINT	Delay time Tg	0 to 65535
System_gain_percent_UINT	System gain Ks	0 to 65535
Disturbance_percent_INT	Disturbance	–100 to 100
Amplitude_signal_noise_12Bit_UINT	Amplitude of heterodyned signal noise	0 to 65535
Length_of_period_signal_noise_ms_UINT	Length of the period of heterodyned signal noise	0 to 65535
Signal_range_basic_unit_UINT	Signal range of a basic unit	0 to 65535
Start_value_12Bit_UINT	Start value of the simulation	0 to 4095
Manual_value_12Bit_UINT	Manual value of the simulation	0 to 4095
Outputs		
Actual_value_12Bit_UINT	12-bit output value of the simulation = actual value of a control system	0 to 4095
Monitor outputs		
Actual_value_basic_unit_UINT	Simulation output value in a basic unit	0 to 65535

Description

This function block generates the simulation of a PTn control system. The analogue input signal is "Manipulated_ variable_12Bit_UINT". It can be linked with the manipulated variable output signal of a PID controller (or with other 12-bit analogue values). The input "Manipulated_variable_PDM_BOOL" can be linked with the output of a pulse duration modulation system or with other controllers whose outputs are digital, such as 2- or 3-step controllers. The function block's output signal "Actual_value_12Bit_UINT" can be applied to the input variables of controllers.

The function block is started with "Activate_BOOL=1" (start-up phase = 5 s). "Activate_BOOL=0" resets it. "Interrupt_BOOL=1" "freezes" the current actual value. The setting "Accept_manual_value_BOOL=1" takes control of the simulation with the "Manual_value_12Bit_INT". When manual operation is terminated, the simulation continues with the manual value. Depending on the variable "Enable_PDM_BOOL", the input options 12-bit manipulated variable ("Enable_PDM_BOOL=0") or PDM manipulated variable ("Enable_PDM_BOOL=1") can be chosen. The initial call of the function block (after reset) starts the simulation with the value of the variable "Start_value_12Bit_UINT".

A PTn system up to an order of ten can be specified with "Order_delay_element_PTn_UINT". The transfer function of the system can be specified with "System_gain_ percent_UINT" (100 % = 1.0 = neutral, no gain). The system dynamics can be specified with "Delay_time_10thsec_UINT". For further information and illustrations of PTn systems, refer to chapter 3 from page 192.

Disturbances can be initiated with the variable "Disturbance_percent_INT". The manipulated quantity must be opposite but of the same magnitude as the disturbance. This results in the actual value which would be achieved without the influence of the disturbance.

Example:
Without a disturbance, a 12-bit manipulated variable 2048 (= 50 %) results in an actual value of 1000.

For the input "Disturbance_percent_INT=20", the 12-bit manipulated variable would have to be increased to 2867 (= 70 %) for the actual value 1000.

For the input "Disturbance_percent_INT=10", the 12-bit manipulated variable would have to be reduced to 1638 (= 40 %) for the actual value of 1000.

With "Amplitude_signal_noise_12Bit_UINT" and "Length_of_period_signal_noise_ms_UINT", signal noise in the form of triangular oscillation can be heterodyned with the actual value (simulation output). The monitor output "Actual_value_basic_unit_UINT" outputs a scaled actual value, in conjunction with "Signal_range_basic_unit_UINT".

Example:
Actual values between 0 and 240 °C are to be displayed:

=> Signal_range_basic_unit_UINT=240.

The range 0 to 240 is assigned to the 12-bit range 0 to 4095, in order for the actual value to be output in the corresponding basic unit (→ application example below).

Inverting the simulation behaviour:

If you want the actual value (output of the simulation) to decrease when the manipulated variables become larger (e.g. temperature control system with a refrigeration unit => large values for the manipulated variable of the refrigeration unit result in small actual values), then invert the manipulated variable inputs as follows:

- 12-bit manipulated variable of the simulation = 4095 to manipulated variable of the controller
- BOOL manipulated variable of the simulation = inversion of the PDM manipulated variable

Precision of the simulation behaviour:

The simulation produces the most precise results when you use the input "Manipulated_variable_12Bit_UINT" (setting: "Enable_PDM_BOOL=0").

In the mode "Enable_PDM_BOOL=1", the problem is that the ratio between the period length and PLC cycle time is relatively small, so the simulation is made with limited resolution.

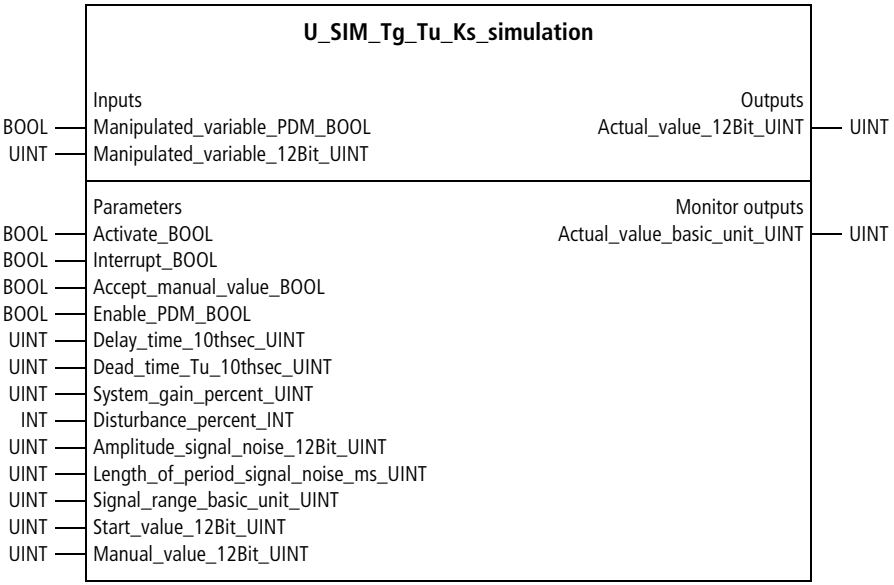
The most precise simulation results are produced in conjunction with equidistant PLC cycles. For this, the function block "U_CYC_cycletime_setpoint_value" can be used.

Example:
The application example "PT5_Sim" associates the simulation with the 12-bit analogue manipulated variable of a PID controller. The PTn order is 5, the delay time Tg is 10 s and the system gain is 1.50.

Application of the function block "U_SIM_PTN_simulation" in the program "PT5_Sim"

```
PROGRAM PT5_Sim
VAR
    PID_controller : U_PID_controller ;
    PT5_SIMULATION : U_SIM_PTN_SIMULATION ;
    Digital_input_0 : BOOL ;
END_VAR
LD  PID_controller.Manipulated_variable_12Bit_UINT
ST  PT5_SIMULATION.Manipulated_variable_12Bit_UINT
CAL PT5_SIMULATION(
    Manipulated_variable_PDM_BOOL :=0,
    Activate_BOOL :=Digital_input_0,
    Interrupt_BOOL :=0,
    Accept_manual_value_BOOL :=0,
    Enable_PDM_BOOL :=0,
    Order_delay_element_PTn_UINT :=5,
    System_gain_percent_UINT :=100,
    Delay_time_10thsec_UINT :=150,
    Disturbance_percent_INT :=0,
    Amplitude_signal_noise_12Bit_UINT :=0,
    Length_of_period_signal_noise_ms_UINT :=0,
    Signal_range_basic_unit_UINT :=240,
    Start_value_12Bit_UINT :=0,
    Manual_value_12Bit_UINT :=0
    )
LD  PT5_SIMULATION.Actual_value_12Bit_UINT
ST  PID_controller.Actual_value_12Bit_UINT

END_PROGRAM
```

U_SIM_Tg_Tu_Ks_simulation
Simulation of a PTn System with Input of the System Variables delay timeTg, dead time Tu and system gain Ks



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Manipulated_variable_PDM_BOOL	Pulse duration modulation input	0/1
Manipulated_variable_12Bit_UINT	Analogue manipulated variable input	0 to 4095
Parameters		
Activate_BOOL	Enables the simulation	0/1
Interrupt_BOOL	Interrupts the simulation	0/1
Accept_manual_value_BOOL	Accepts the manual value	0/1
Enable_PDM_BOOL	Enable the PDM input	0/1
Delay_time_10thsec_UINT	Delay time Tg	0 to 65535
Dead_time_Tu_10thsec_UINT	Dead time Tu	0 to 65535
System_gain_percent_UINT	System gain Ks	0 to 65535
Disturbance_percent_INT	Disturbance	–100 to 100
Amplitude_signal_noise_12Bit_UINT	Amplitude of heterodyned signal noise	0 to 65535
Length_of_period_signal_noise_ms_UINT	Period length of heterodyned signal noise	0 to 65535
Signal_range_basic_unit_UINT	Signal range of a basic unit	0 to 65535
Start_value_12Bit_UINT	Start value of the simulation	0 to 4095
Manual_value_12Bit_UINT	Manual value of the simulation	0 to 4095
Outputs		
Actual_value_12Bit_UINT	12-bit output value of the simulation = actual value of a control system	0 to 4095
Monitor outputs		
Actual_value_basic_unit_UINT	Simulation output value in a basic unit	0 to 65535

Description

This function block generates the simulation of a PTn control system with the system variables delay time Tg, system dead time Tu and system gain Ks. The analogue input signal is "Manipulated_variable_12Bit_UINT". It can be linked with the manipulated variable output signal of a PID controller (or with other 12-bit analogue values). The input "Manipulated_variable_PDM_BOOL" can be linked with the output of a pulse duration modulation system or with other controllers whose outputs are digital, such as 2- or 3-step controllers. The function block's output signal "Actual_value_12Bit_UINT" can be applied to the input variables of controllers.

The function block is started with "Activate_BOOL=1" (start-up phase = 5 s). "Activate_BOOL=0" resets it. "Interrupt_BOOL=1" "freezes" the current actual value. The setting "Accept_manual_value_BOOL=1" takes control of the simulation with the "Manual_value_12Bit_INT". When manual operation is terminated, the simulation continues with the manual value. Depending on the variable "Enable_PDM_BOOL", the input options 12-bit manipulated variable ("Enable_PDM_BOOL=0") or PDM manipulated variable ("Enable_PDM_BOOL=1") can be chosen. The initial call of the function block (after reset) starts the simulation with the value of the variable "Start_value_12Bit_UINT".

The system variables "Delay_time_10thsec_UINT", "Dead_time_Tu_10thsec_UINT" and "System_gain_percent_UINT" can be entered for a PTn control system. For further information and illustrations of PTn systems, refer to chapter 3 from page 192. Figure 21 in chapter 4 shows the variables Tg and Tu in graphical form.

Disturbances can be initiated with the variable "Disturbance_percent_INT". The manipulated quantity must be opposite but of the same magnitude as the disturbance. This results in the actual value which would be achieved without the influence of the disturbance.

Example:
Without a disturbance, a 12-bit manipulated variable 2048 (= 50 %) results in an actual value of 1000.

For the input "Disturbance_percent_INT=20", the 12-bit manipulated variable would have to be increased to 2867 (= 70 %) for the actual value 1000.

For the input "Disturbance_percent_INT=10", the 12-bit manipulated variable would have to be reduced to 1638 (= 40 %) for the actual value of 1000.

With "Amplitude_signal_noise_12Bit_UINT" and "Length_of_period_signal_noise_ms_UINT", signal noise in the form of triangular oscillation can be heterodyned with the actual value (simulation output). The monitor output "Actual_value_basic_unit_UINT" outputs a scaled actual value, in conjunction with "Signal_range_basic_unit_UINT".

Example:
The actual values are to be displayed in %:

=> Signal_range_basic_unit_UINT=100.

The range 0 to 100 is assigned to the 12-bit range 0 to 4095, so the actual value is output as a percentage (→ application example below).

Inverting the simulation behaviour:

If you want the actual value (output of the simulation) to decrease when the manipulated variables become larger (e.g. temperature control system with a refrigeration unit => large values for the manipulated variable of the refrigeration unit result in small actual values), then invert the manipulated variable inputs as follows:

- 12-bit manipulated variable of the simulation = 4095 to manipulated variable of the controller
- BOOL manipulated variable of the simulation = (logical) negation of the PDM manipulated variable

Precision of the simulation behaviour:

The simulation produces the most precise results when you use the input "manipulated_variable_12Bit_UINT" (setting: "Enable_PDM_BOOL=0").

In the mode "Enable_PDM_BOOL=1", the problem is that the ratio between the period length and PLC cycle time is relatively small, so the simulation is made with limited resolution.

The most precise simulation results are produced in conjunction with equidistant PLC cycles. For this, the function block "U_CYC_cycletime_setpoint_value" can be used.

Example:
The application example "SimTg_Tu" associates the simulation with the 12-bit analogue manipulated variable of a PID controller. When the function block is reset, the simulation begins with the start value 1000. The system variables are:

- Delay time Tg = 240 s
- System dead time Tu = 30 s
- System gain = 0.30

Application of the function block "U_SIM_Tg_Tu_Ks_simulation" in the program "SimTg_Tu"

```
PROGRAM SimTg_Tu
VAR
    PID_controller : U_PID_controller ;
    SIMULATION : U_SIM_TG_TU_KS_SIMULATION ;
    Digital_input_0 : BOOL ;
END_VAR

LD  PID_controller.Manipulated_variable_12Bit_UINT
ST  SIMULATION.Manipulated_variable_12Bit_UINT
CAL Simulation(
    Manipulated_variable_PDM_BOOL :=0,
    Activate_BOOL :=Digital_input_0,
    Interrupt_BOOL :=0,
    Accept_manual_value_BOOL :=0,
    Enable_PDM_BOOL :=0,
    Delay_time_Tg_10thsec_UINT :=2400,
    Dead_time_Tu_10thsec_UINT :=300,
    System_gain_Ks_percent_UINT :=30,
    Disturbance_percent_INT :=0,
    Amplitude_signal_noise_12Bit_UINT :=0,
    Length_of_period_signal_noise_ms_UINT :=0,
    Signal_range_basic_unit_UINT :=100,
    Start_value_12Bit_UINT :=1000,
    Manual_value_12Bit_UINT :=0
    )
LD  SIMULATION.Actual_value_12Bit_UINT
ST  PID_controller.Actual_value_12Bit_UINT

END_PROGRAM
```

8 Fuzzy Logic Systems

Fundamentals and general information

General information

The fuzzy logic function blocks of the Toolbox (CCT) can be used without in-depth knowledge of fuzzy logic. The reason is that the fuzzy-specific operations such as fuzzification, inference, defuzzification, selection of term shapes (triangle, trapezoid, etc.) are automatically processed in the function block (black box). The user is only required to set parameter values based on practical experience which can be described verbally (for details, → below). After working through some of the numerous examples in this chapter, you will be able to use these function blocks without exhaustive knowledge of fuzzy logic. The fundamental principles required to understand fuzzy logic function blocks are summarized below.

The fundamentals of fuzzy logic

With fuzzy logic, practical experience which can be described verbally is implemented qualitatively or quantitatively, in the form of IF-THEN assignments (fuzzy logic rules)". The block diagram of a fuzzy logic system is shown in figure 59. Several input variables are linked with an output variable. In the context of fuzzy logic, we often speak of linguistic input and output variables. This is because when fuzzy logic rules are stated, the input and output variables are added verbally (→ Example 3).

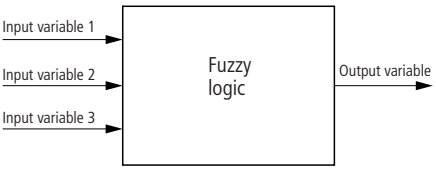


Figure 59: Block diagram of a fuzzy logic system with 3 inputs (linguistic) and one output (linguistic)

For the input variables, practical experience is only required for some discrete terms. You need to have this experience in either qualitative or quantitative form (→ Example 1).

Example 1:

- Qualitative: "small", "medium", "large"
- Quantitative: "20 %", "50 %", "80 %"

Information processing takes place by linking input and output variables. These "IF-THEN assignments" are known as fuzzy logic rules (→ Example 2).

Example 2:

- IF (input variable 1 = small) AND
  IF (input variable 2 = large) AND
  IF (input variable 3 = small)
  THEN (output variable = large)
- IF (input variable 1 = small) AND
  IF (input variable 2 = small) AND
  IF (input variable 3 = large)
  THEN (output variable = small)

If a fuzzy logic system consists only of an input variable and an output variable, then the fuzzy logic function blocks (→ page 398 to page 412) are analogous with the interpolation function blocks. We will therefore first show you how to implement practical experience for this special case (only one input variable) with an interpolation function block.

Example 3:

Assume practical experience in controlling the temperature of a room as a function of an input variable, as follows:

- input variable = ambient temperature
- output variable = lead temperature of the boiler

Rule 1

- input variable 1:
  IF (ambient temperature = small (< 0 °C))
- output variable:
  THEN (lead temperature of the boiler = large (80 °C))

Rule 2

- input variable 1:
  IF (ambient temperature = large (> 10 °C))
- output variable:
  THEN (lead temperature of the boiler = small (60 °C))

The practical experience (as stated in the two IF-THEN assignments, known as Rules 1 and 2) can be implemented with an interpolation, as shown in figure 60.

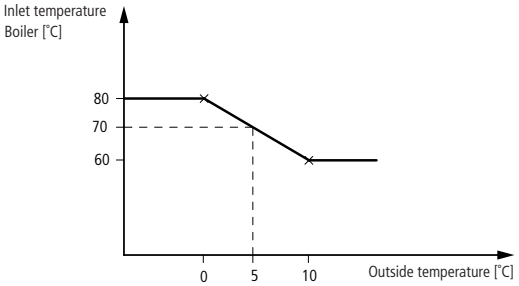


Figure 60: Implementing practical experience through interpolation. Except for the terms, the characteristic curve stays at the limiting value

At this point, it is apparent that interpolation and fuzzy logic are closely related. Interpolation results in characteristic curves, fuzzy logic systems with several input variables produce characteristic fields. In contrast, fuzzy logic systems and PID-controllers are quite different. A fuzzy-logic PD controller is created only when control deviation and change of the control deviation (the variables specific to PID controllers) are used. Adding an integration mechanism creates an algorithm similar to a PID controller.

A controller for room temperature is implemented in Example 4, just like the above example. But now practical experience regarding two input variables and an output variable is implemented. To process this knowledge, fuzzy logic must now be used. Regarding fuzzy logic rules 1 to 4, the terms "small" and "large" were permutated for the input variables. That is, all possible combinations of the input variables were assigned to a random output variable. The information of the verbally stated fuzzy logic rules can be expressed in the compact form of a table (matrix) (→ table 4).

Example 4:

Assume practical experience in controlling the temperature of a room as a function of two input variables, as follows:

- input variable 1 = ambient temperature
- input variable 2 = desired temperature of the room
- output variable = lead temperature of the boiler

Rule 1

- input variable 1:
  IF (ambient temperature = small (< 0 °C)) AND
- input variable 2:
  IF (desired temperature of the room = small (< 15 °C))
- output variable:
  THEN (lead temperature of the boiler = medium (75 °C))

Rule 2

- input variable 1:
  IF (ambient temperature = small (< 0 °C)) AND
- input variable 2:
  IF (desired temperature of the room = large (> 20 °C))
- output variable:
  THEN (lead temperature of the boiler = large (80 °C))

Rule 3

- input variable 1:
  IF (ambient temperature = large (> 10 °C)) AND
- input variable 2:
  IF (desired temperature of the room = small (< 15 °C))
- output variable:
  THEN (lead temperature of the boiler = small (60 °C)

Rule 4

- input variable 1:
  IF (ambient temperature = large (> 10 °C)) AND
- input variable 2:
  IF (desired temperature of the room = large (> 20 °C))
- output variable:
  THEN (lead temperature of the boiler = medium (75 °C))

Table 4: Fuzzy logic rules (basis for the rules) in the form of a matrix

Desired temperature of the room [°C]	Ambient temperature [°C]	
	0	10
15	TVK = 75 (Rule 1)	TVK = 60 (Rule 3)
20	TVK = 80 (Rule 2)	TVK = 75 (Rule 4)

lead temperature of the boiler = TVK

Figure 61 shows the corresponding fuzzy logic characteristic field for the information in table 4. The two input variables form the horizontal surface and the output variable forms the vertical component of the characteristic field. The four fuzzy logic rules produce four terms (input variable1 | input variable 2 | output variable) in the characteristic field. The terms are (all in °C):

- term 1: 0 | 15 | 75
- term 2: 0 | 20 | 80
- term 3: 10 | 15 | 60
- term 4: 10 | 20 | 75

If the input variables are within this range (input variable 1 between 0 and 10 °C, input variable 2 between 15 and 20 °C), then the output variable is in the central, non-linear area. If one input variable is outside the outermost term, then only interpolation takes place in the resulting characteristic area. This is because the field changes as a function of only one input variable (cf. fig. 60). For example, this is the case for "desired temperatures of the room" < 15 °C. The interpolation area can be seen in the front of the figure. If both input variables are outside the outermost terms, then a plane parallel to the horizontal surface will result in the characteristic field. In this plane, the same output variable will be assigned to multiple input variables.

For example, this is the case for "desired temperatures of the room" < 15 °C and "ambient temperatures" > 10 °C. This produces the characteristic field area in the right-hand front corner (square surface), in which the output variable "lead temperature of the boiler" is constant at 60 °C.

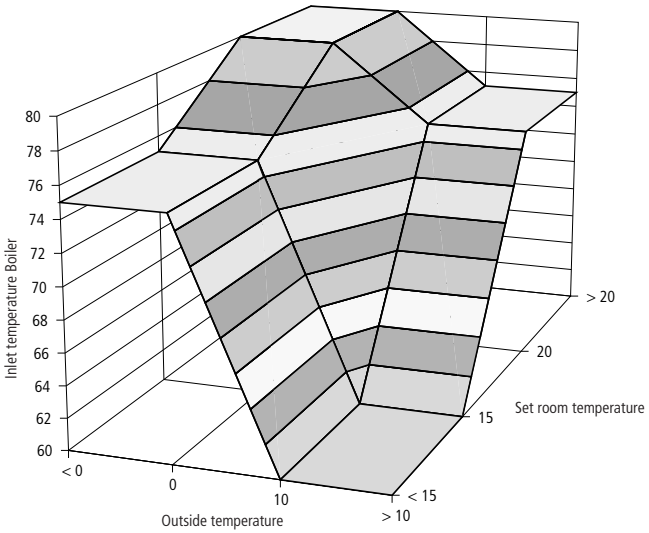


Figure 61: Implementing practical experience with fuzzy logic. Fuzzy logic characteristic field for 2 input variables, 2 terms and 4 fuzzy logic rules (=> 4 terms in the characteristic field)

Levels of activity (= effectiveness = truth content) of the fuzzy logic rules

Enabling diagnostic mode displays the levels of activity of the fuzzy logic rules in tenths of a percent, in the output section of the fuzzy logic function blocks (the function block's cycle time requirement is doubled when the diagnosis is enabled). The sum of the activity levels is always 1000 ‰ (rounding-off errors are possible). The level of activity of a given fuzzy rule reflects the degree to which the rule is involved in generating the output variable. The fuzzy logic rules (THEN assignments) can be modified wherever the greatest effect can be attained (this can be done online).

Example:

For the fuzzy logic system in table 5, the terms are assigned values such that input variable 1 has a range in percent (0 to 100) and input variable 2 has a range in tenths of a percent (0 to 1000).

Table 5: Fuzzy logic rules (basis for the rules) in the form of a matrix

Input variable 2 [‰]	**Input variable 1 [%]**	
	0	**100**
0	(Rule 1)	(Rule 3)
1000	(Rule 2)	(Rule 4)

The levels of activity are independent of the assignments for the fuzzy logic rules.

The abbreviations used below mean:

- E1 = input variable 1
- E2 = input variable 2
- a = small term (left term) of an input variable (→ fig. 62 to 64)
- b = next larger term (right term, when only 2 terms can be entered) of an input variable
- c = next larger term . . .

Example:

E1a = small term regarding input variable 1

Some constellations of the input variables result in the following activity levels of the fuzzy logic rules:

- input variable 1 = 30
- input variable 2 = 400
- Activity level of fuzzy rule 1 (combination [E1a | E2a]) = 0.7 × 0.6 = 0.420
- Activity level of fuzzy rule 2 (combination [E1a | E2b]) = 0.7 × 0.4 = 0.280
- Activity level of fuzzy rule 3 (combination [E1b | E2a]) = 0.3 × 0.6 = 0.180
- Activity level of fuzzy rule 4 (combination [E1b | E2b]) = 0.3 × 0.4 = 0.120

- input variable 1 = 80
- input variable 2 = 100
- Activity level of fuzzy rule 1 (combination [E1a | E2a]) = 0.2 × 0.9 = 0.180
- Activity level of fuzzy rule 2 (combination [E1a | E2b]) = 0.2 × 0.1 = 0.020
- Activity level of fuzzy rule 3 (combination [E1b | E2a]) = 0.8 × 0.9 = 0.720
- Activity level of fuzzy rule 4 (combination [E1b | E2b]) = 0.8 × 0.1 = 0.080

Terms/membership functions
Within the function block, the terms are automatically determined by the values for the following variables:

- input variable 1:
  - Center_term_input_variable_1a_INT (for 2, 3, 5, 7 and 9 terms)
  - Center_term_input_variable_1b_INT (for 2, 3, 5, 7 and 9 terms)
  - Center_term_input_variable_1c_INT (for 3, 5, 7 and 9 terms)
  - Center_term_input_variable_1d_INT (for 5, 7 and 9 terms)
  - Center_term_input_variable_1e_INT (for 5, 7 and 9 terms)
  - Center_term_input_variable_1f_INT (for 7 and 9 terms)
  - Center_term_input_variable_1g_INT (for 7 and 9 terms)
  - Center_term_input_variable_1h_INT (for 9 terms)
  - Center_term_input_variable_1i_INT (for 9 terms)

- input variable 2:
  - Center_term_input_variable_2a_INT (for 2, 3, 5, 7 and 9 terms)
  - Center_term_input_variable_2b_INT (for 2, 3, 5, 7 and 9 terms)
  - Center_term_input_variable_2c_INT (for 3, 5, 7 and 9 terms)
  - Center_term_input_variable_2d_INT (for 5, 7 and 9 terms)
  - Center_term_input_variable_2e_INT (for 5, 7 and 9 terms)
  - Center_term_input_variable_2f_INT (for 7 and 9 terms)
  - Center_term_input_variable_2g_INT (for 7 and 9 terms)
  - Center_term_input_variable_2h_INT (for 9 terms)
  - Center_term_input_variable_2i_INT (for 9 terms)

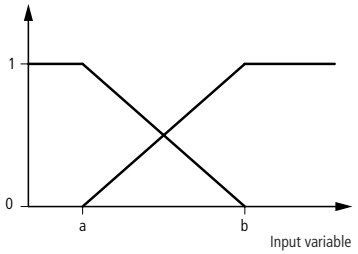


Figure 62: For fuzzy logic systems with 2 terms, the values assigned to a and b automatically result in this fixed sequence of terms for each input variable (membership functions)

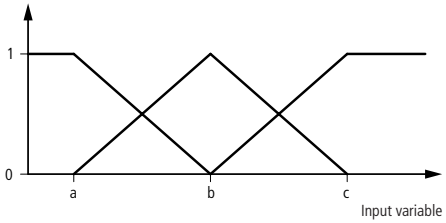


Figure 63: For fuzzy logic systems with 3 terms, the values assigned to a, b and c automatically result in this fixed sequence of terms for each input variable (membership functions)

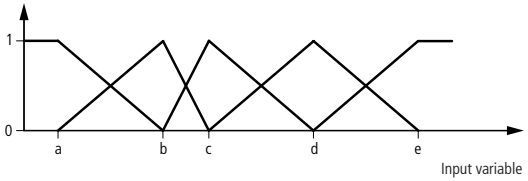


Figure 64: For fuzzy logic systems with 5 terms, the values assigned to a, b, c, d and e automatically result in this fixed sequence of terms for each input variable (membership functions)

Inference method

The product technique is used as the permanently set inference method. This technique produces fuzzy logic characteristic fields without irregularities.

Defuzzification

Defuzzification is performed according to the centre of gravity method, using singletons.

Increased non-linearity of a characteristic field (characteristic curve)

If you want to use interpolation to generate a characteristic non-linear curve, you must increase the number of terms. Figure 65 illustrates a characteristic curve interpolation with four terms, like Example 3.

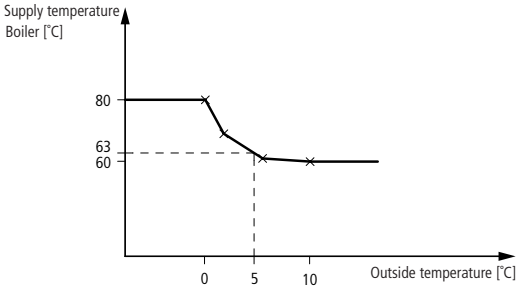


Figure 65: Interpolation with several terms for generating a characteristic curve with increased non-linearity

If you need a fuzzy logic characteristic field with increased non-linearity (like interpolation), then increase the number of terms of the characteristic field (→ fig. 65). By selecting fuzzy logic systems with more terms per input variable, you can increase the number of resulting characteristic field terms.

The following principle applies to planning of fuzzy logic systems: "Select the fuzzy logic system as small (number of terms per input variable) as possible and as large as necessary (depending on the required non-linearity)".

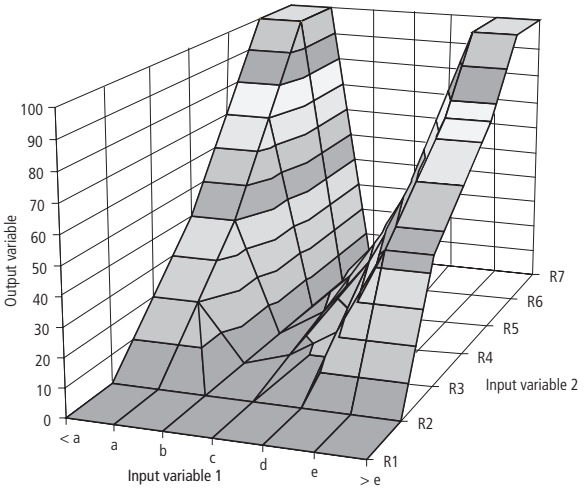


Figure 66: fuzzy logic system with 2 input variables, 5 terms and 25 fuzzy logic rules (=> 25 terms in the characteristic field) for generation of a characteristic field with increased non-linearity

Linking and cascading fuzzy logic function blocks

For complicated systems, you can link or cascade several fuzzy logic function blocks. Figure 67 illustrates how to implement six input variables with three function blocks. Figure 68 substitutes the function block "U_FUZ_42_FUZZY_4I_2T" with three function blocks of the type "U_FUZ_22_FUZZY_2I_2T".

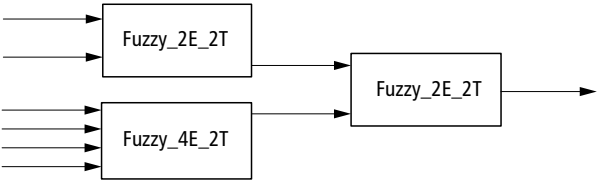


Figure 67: Processing six input variables by cascading fuzzy logic function blocks

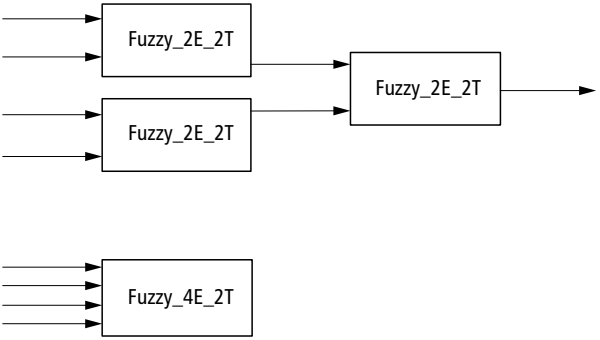


Figure 68: Alternative to the fuzzy logic function block "U_FUZ_42_FUZZY_4I_2T" (4 inputs)

Overview of the CCT fuzzy logic function blocks

The fuzzy logic function blocks have one, two, three, four or five input variables and one output variable. The number of terms can be two, three, five, seven or nine. Raising "number of terms" (base) to the "number of input variables" (exponent) result the "number of fuzzy logic rules per function block" (= number of characteristic field terms).

If serveral output variables are required for identical input variables, then use several function blocks with identical input parameter values.

Table 6: Overview – fuzzy systems

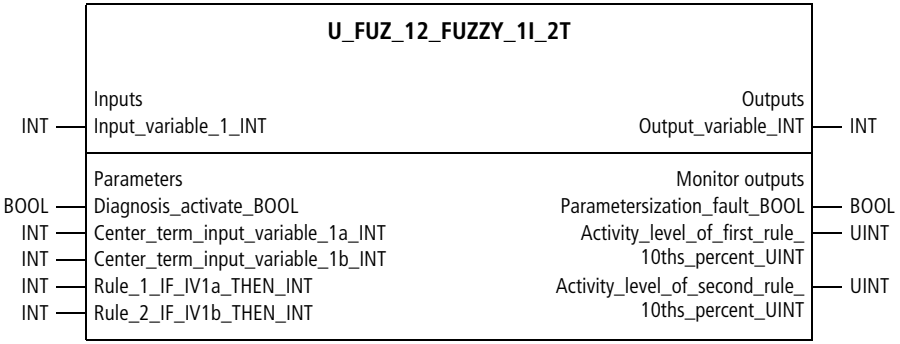
Function block	Number of		
	input variables	terms per input variable	fuzzy logic rules per function block
U_FUZ_12_FUZZY_1I_2T	1	2	2
U_FUZ_13_FUZZY_1I_3T	1	3	3
U_FUZ_15_FUZZY_1I_5T	1	5	5
U_FUZ_17_FUZZY_1I_7T	1	7	7
U_FUZ_19_FUZZY_1I_9T	1	9	9
U_FUZ_22_FUZZY_2I_2T	2	2	4
U_FUZ_23_FUZZY_2I_3T	2	3	9
U_FUZ_25_FUZZY_2I_5T	2	5	25
U_FUZ_27_FUZZY_2I_7T	2	7	49
U_FUZ_32_FUZZY_3I_2T	3	2	8
U_FUZ_33_FUZZY_3I_3T	3	3	27
U_FUZ_35_FUZZY_3I_5T	3	5	125
U_FUZ_42_FUZZY_4I_2T	4	2	16
U_FUZ_43_FUZZY_4I_3T	4	3	81
U_FUZ_52_FUZZY_5I_2T	5	2	32
U_FUZ_53_FUZZY_5I_3T	5	3	243

Table 7: Overview – fuzzy-weigthing manipulated variables

Fuzzy-weighting function block	Number of manipulated variables to be weighted
U_FUZ4_weighted_output_variable	4
U_FUZ8_weighted_output_variable	8
U_FUZ12_weighted_output_variable	12

Fuzzy logic system: 1 input, 2 terms

U_FUZ_12_FUZZY_1I_2T
Fuzzy Logic System with 1 Input Variable, 2 Terms per Input Variable and 2 Fuzzy Logic Rules



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_variable_1_INT	First linguistic input variable	–32768 to 32767
Parameters		
Diagnosis_activate_BOOL	Activates diagnostic mode	0/1
Center_term_input_variable_1a_INT	Input variable 1: term a	–32768 to 32767
Center_term_input_variable_1b_INT	Input variable 1: term b	–32768 to 32767
Rule_1_IF_IV1a_THEN_INT	Fuzzy rule 1	–32768 to 32767
Rule_2_IF_IV1b_THEN_INT	Fuzzy rule 2	–32768 to 32767
Outputs		
Output_variable_INT	Linguistic output variable	–32768 to 32767

Designation	Significance	Value range
Monitor outputs		
Parameterization_fault_BOOL	Status: parameterization fault	0/1
Activity_level_of_first_rule_ 10ths_percent_UINT	First active fuzzy rule	0 to 1000
Activity_level_of_second_rule_ 10ths_percent_UINT	Second active fuzzy rule	0 to 1000

Description

For each of the linguistic input variables "Input_variable_ 1_INT" two terms can be assigned values with the variables "Center_term_input_variable_1_. . ._INT". This results in the the arrangement of terms shown in figure 62 (→ above). The values assigned to the terms can be in any unit, such as percentage, tenths of a percent, °C, 12-bit, etc. Each possible combination of the input variables for these terms is assigned to an output variable (= THEN assignment of the fuzzy rule = characteristic curve term), resulting in 2 fuzzy logic rules.

Example:

- Fuzzy rule 1:
  With the variable "Rule_1_IF_IV1a_THEN_INT", an output variable is assigned to "term a, input variable 1". The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1a"
    THEN the output variable has the value "X1"
    (any integer value can be assigned).

- Fuzzy rule 2: With the variable "Rule_2_IF_IV1b_THEN_INT", an output variable is assigned to "term b, input variable 1". The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1b"
    THEN the output variable has the value "X2"
    (any integer value can be assigned).

In the output section of the function block, an output variable is calculated as a function of the input variables. This output variable has the same unit as the THEN assignments of the fuzzy logic rules. Standardisation or rescaling of parameter values is not required for the fuzzy logic function blocks.

Enter the terms of the input variables in ascending order. If this is not done, the status display "Parameterization_fault_BOOL" will be "1".

"Diagnosis_activate_BOOL=1" displays the activity levels of the fuzzy logic rules in the output section. (The function block's cycle time requirement is doubled when the diagnosis is enabled.) Detailed descriptions of fuzzy logic characteristic fields and the activity levels of fuzzy logic rules can be found at the beginning of this chapter.

Example:
Assume practical experience in optimising the desired temperature of a hothouse as a function of one input variable, as follows:

- input variable 1 = humidity [‰]
- output variable = desired temperature [1/10 °C]

Minimum humidity value = 150
Maximum humidity value = 650

Fuzzy rule 1:
[Humidity = 150] => desired temperature = 200
Fuzzy rule 2:
[Humidity = 650] => desired temperature = 260

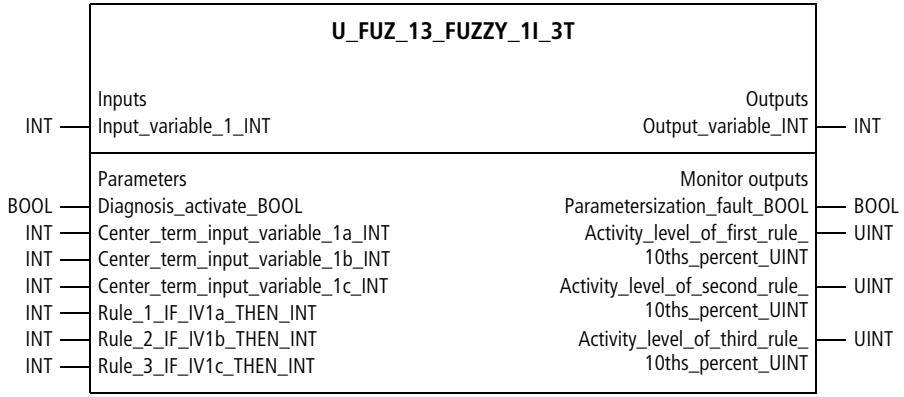
The input variables "humidity = 300" result in a desired temperature of 21.8 °C. The corresponding levels of activity are shown in the output section of the function block.

Application of the function block "U_FUZ_12_FUZZY_1I_2T" in the program "Fuz12"

```
PROGRAM Fuz12

VAR
    Fuzzy_setpoint_optimization : U_FUZ_12_FUZZY_1I_2T ;
    Humidity_10ths_percent : INT :=300 ;
    Setpoint_temperature_10th_degree_celsius : INT ;
END_VAR

CAL Fuzzy_setpoint_optimization(
    Input_variable_1_INT :=Humidity_10ths_percent,
    Diagnosis_activate_BOOL :=1,
    Center_term_input_variable_1a_INT :=150,
    Center_term_input_variable_1b_INT :=650,
    Rule_1_IF_IV1a_THEN_INT :=200,
    Rule_2_IF_IV1b_THEN_INT :=260,
    Output_variable_INT=>218,
    Parameterization_fault_BOOL=>0,
    Activity_level_of_first_rule_10ths_percent_UINT=>700,
    Activity_level_of_second_rule_10ths_percent_UINT=>300
    )
LD  Fuzzy_setpoint_optimization.output_variable_INT
ST  Setpoint_temperature_10ths_degree_celsius
END_PROGRAM
```

Fuzzy logic system:
1 input, 3 terms

U_FUZ_13_FUZZY_1I_3T
Fuzzy Logic System with 1 Input Variable, 3 Terms per Input Variable and 3 Fuzzy Logic Rules



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_variable_1_INT	First linguistic input variable	–32768 to 32767
Parameters		
Diagnosis_activate_BOOL	Activates diagnostic mode	0/1
Center_term_input_variable_1a_INT	Input variable 1: term a	–32768 to 32767
Center_term_input_variable_1b_INT	Input variable 1: term b	–32768 to 32767
Center_term_input_variable_1c_INT	Input variable 1: term c	–32768 to 32767
Rule_1_IF_IV1a_THEN_INT	Fuzzy rule 1	–32768 to 32767
Rule_2_IF_IV1b_THEN_INT	Fuzzy rule 2	–32768 to 32767
Rule_3_IF_IV1c_THEN_INT	Fuzzy rule 3	–32768 to 32767

Designation	Significance	Value range
Outputs		
Output_variable_INT	Linguistic output variable	–32768 to 32767
Monitor outputs		
Parameterization_fault_BOOL	Status: parameterization fault	0/1
Activity_level_of_first_rule_ 10ths_percent_UINT	First active fuzzy rule	0 to 1000
Activity_level_of_second_rule_ 10ths_percent_UINT	Second active fuzzy rule	0 to 1000
Activity_level_of_third_rule_ 10ths_percent_UINT	Third active fuzzy rule	0 to 1000

Description

For the linguistic input variables "Input_variable_1_INT", three terms can be assigned values, with the variables "Center_term_input_variable_1_. . ._INT". This results in the arrangement of terms shown in figure 63 (→ above). The values assigned to the terms can be in any unit, such as percentage, tenths of a percent, °C, 12-bit, etc. Each possible combination of the input variables for these terms is assigned to an output variable (= THEN assignment of the fuzzy rule = characteristic curve term), resulting in 3 fuzzy logic rules.

Example:

- Fuzzy rule 1:
  With the variable "Rule_1_IF_IV1a_THEN_INT", an output variable is assigned to "term a, input variable 1". The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1a"
    THEN the output variable has the value "X1"
    (any integer value can be assigned).

- Fuzzy rule 2: With the variable "Rule_2_IF_IV1b_THEN_INT", an output variable is assigned to "term b, input variable 1". The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1b"
    THEN the output variable has the value "X2"
    (any integer value can be assigned).

In the output section of the function block, an output variable is calculated as a function of the input variables. This output variable has the same unit as the THEN assignments of the fuzzy logic rules. Standardisation or rescaling of parameter values is not required for the fuzzy logic function blocks.

Enter the terms of the input variables in ascending order. If this is not done, the status display "Parameterization_fault_BOOL" will be "1".

"Diagnosis_activate_BOOL=1" displays the activity levels of the fuzzy logic rules in the output section. (The function block's cycle time requirement is doubled when the diagnosis is enabled.) Detailed descriptions of fuzzy logic characteristic fields and the activity levels of fuzzy logic rules can be found at the beginning of this chapter.

Example:
Assume practical experience in optimising the desired temperature of a hothouse as a function of one input variable, as follows:

- input variable 1 = humidity [‰]
- output variable = desired temperature [1/10 °C]

Minimum humidity value = 150
Medium humidity value = 400
Maximum humidity value = 650

Fuzzy rule 1:
[Humidity = 150] => desired temperature = 200
Fuzzy rule 2:
[Humidity = 400] => desired temperature = 220
Fuzzy rule 3:
[Humidity = 650] => desired temperature = 260

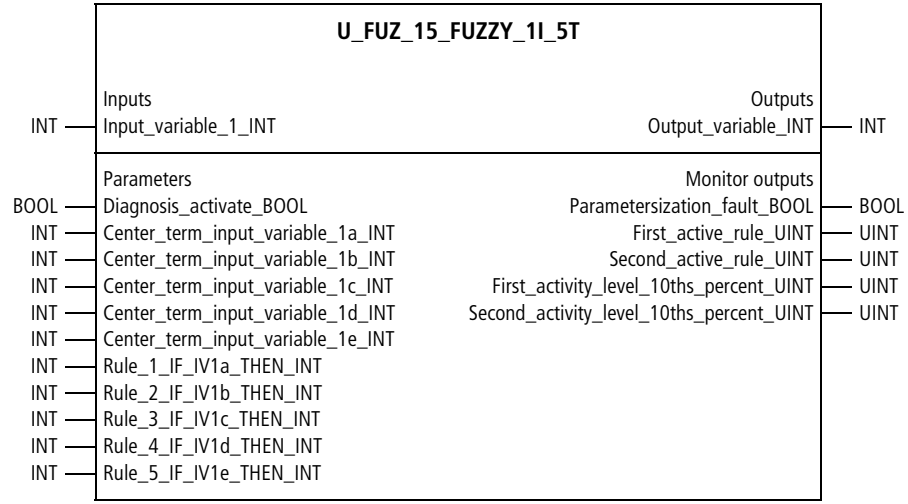
The input variables "humidity = 300" result in a desired temperature of 21.2 °C. The corresponding levels of activity are shown in the output section of the function block.

Application of the function block "U_FUZ_13_FUZZY_1I_3T" in the program "Fuz13"

```
PROGRAM Fuz13

VAR
    Fuzzy_setpoint_optimization : U_FUZ_13_FUZZY_1I_3T ;
    Humidity_10ths_percent : INT :=300 ;
    Setpoint_temperature_10th_degree_celsius : INT ;
END_VAR

CAL Fuzzy_setpoint_optimization(
    Input_variable_1_INT :=Humidity_10ths_percent,
    Diagnosis_activate_BOOL :=1,
    Center_term_input_variable_1a_INT :=150,
    Center_term_input_variable_1b_INT :=400,
    Center_term_input_variable_1c_INT :=650,
    Rule_1_IF_IV1a_THEN_INT :=200,
    Rule_2_IF_IV1b_THEN_INT :=220,
    Rule_3_IF_IV1c_THEN_INT :=260,
    Output_variable_INT=>212,
    Parameterization_fault_BOOL=>0,
    Activity_level_of_first_rule_10ths_percent_UINT=>400,
    Activity_level_of_second_rule_10ths_percent_UINT=>600,
    Activity_level_of_third_rule_10ths_percent_UINT=>0
    )

LD  Fuzzy_setpoint_optimization.Output_variable_INT
ST  Setpoint_temperature_10th_degree_celsius

END_PROGRAM
```

Fuzzy logic system: 1 input, 5 terms

U_FUZ_15_FUZZY_1I_5T
Fuzzy Logic System with 1 Input Variable, 5 Terms per Input Variable and 5 Fuzzy Logic Rules



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_variable_1_INT	First linguistic input variable	–32768 to 32767
Parameters		
Diagnosis_activate_BOOL	Activates diagnostic mode	0/1
Center_term_input_variable_1a_INT	Input variable 1: term a	–32768 to 32767
Center_term_input_variable_1b_INT	Input variable 1: term b	–32768 to 32767
Center_term_input_variable_1c_INT	Input variable 1: term c	–32768 to 32767
Center_term_input_variable_1d_INT	Input variable 1: term d	–32768 to 32767
Center_term_input_variable_1e_INT	Input variable 1: term e	–32768 to 32767

Designation	Significance	Value range
Rule_1_IF_IV1a_THEN_INT	Fuzzy rule 1	–32768 to 32767
Rule_2_IF_IV1b_THEN_INT	Fuzzy rule 2	–32768 to 32767
Rule_3_IF_IV1c_THEN_INT	Fuzzy rule 3	–32768 to 32767
Rule_4_IF_IV1d_THEN_INT	Fuzzy rule 4	–32768 to 32767
Rule_5_IF_IV1e_THEN_INT	Fuzzy rule 5	–32768 to 32767
Outputs		
Output_variable_INT	Linguistic output variable	–32768 to 32767
Monitor outputs		
Parameterization_fault_BOOL	Status: parameterization fault	0/1
First_active_rule_UINT	First active fuzzy rule	0 to 5
Second_active_rule_UINT	Second active fuzzy rule	0 to 5
First_activity_level_10ths_ percent_UINT	Activity level of the first active rule	0 to 1000
Second_activity_level_10ths_ percent_UINT	Activity level of the second active rule	0 to 1000

Description

For the linguistic input variables "Input_variable_1_INT", five terms can be assigned values, with the variables "Center_term_input_variable_1_. . ._INT". This results in the arrangement of terms shown in figure 64 (→ above). The values assigned to the terms can be in any unit, such as percentage, tenths of a percent, °C, 12-bit, etc. Each possible combination of the input variables for these terms is assigned to an output variable (= THEN assignment of the fuzzy rule = characteristic curve term), resulting in 3 fuzzy logic rules.

Example:

- Fuzzy rule 1:
  With the variable "Rule_1_IF_IV1a_THEN_INT", an output variable is assigned to "term a, input variable 1". The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1a"
    THEN the output variable has the value "X1"
    (any integer value can be assigned).

- Fuzzy rule 2: With the variable "Rule_2_IF_IV1b_THEN_INT", an output variable is assigned to "term b, input variable 1". The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1b"
    THEN the output variable has the value "X2"
    (any integer value can be assigned).

In the output section of the function block, an output variable is calculated as a function of the input variables. This output variable has the same unit as the THEN assignments of the fuzzy logic rules. Standardisation or rescaling of parameter values is not required for the fuzzy logic function blocks.

Enter the terms of the input variables in ascending order. If this is not done, the status display "Parameterization_fault_BOOL" will be "1".

"Diagnosis_activate_BOOL=1" displays the activity levels of the fuzzy logic rules in the output section. (The function block's cycle time requirement is doubled when the diagnosis is enabled.) Detailed descriptions of fuzzy logic characteristic fields and the activity levels of fuzzy logic rules can be found at the beginning of this chapter.

Example:

Assume practical experience in optimising the desired temperature of a hothouse as a function of one input variable, as follows:

- input variable 1 = humidity [‰]
- output variable = desired temperature [1/10 °C]

Very small humidity value = 150
Minimum humidity value = 250
Medium humidity value = 350
Maximum humidity value = 500
Very high humidity value = 650

Fuzzy rule 1:
[Humidity = 150] => desired temperature = 200
Fuzzy rule 2:
[Humidity = 250] => desired temperature = 210
Fuzzy rule 3:
[Humidity = 350] => desired temperature = 220
Fuzzy rule 4:
[Humidity = 500] => desired temperature = 240
Fuzzy rule 5:
[Humidity = 650] => desired temperature = 260

The input variables "Humidity = 300" result in a desired temperature of 21.5 °C. The corresponding levels of activity are shown in the output section of the function block.

Application of the function block "U_FUZ_15_FUZZY_1I_5T" in the program "Fuz15"

```
PROGRAM Fuz15

VAR
    Fuzzy_setpoint_optimization : U_FUZ_15_FUZZY_1I_5T ;
    Humidity_10ths_percent : INT :=300 ;
    Setpoint_temperature_10th_degree_celsius : INT ;
END_VAR

CAL Fuzzy_setpoint_optimization(
    Input_variable_1_INT :=Humidity_10ths_percent,
    Diagnosis_activate_BOOL :=1,
    Center_term_input_variable_1a_INT :=150,
    Center_term_input_variable_1b_INT :=250,
    Center_term_input_variable_1c_INT :=350,
    Center_term_input_variable_1d_INT :=500,
    Center_term_input_variable_1e_INT :=650,
    Rule_1_IF_IV1a_THEN_INT :=200,
    Rule_2_IF_IV1b_THEN_INT :=210,
    Rule_3_IF_IV1c_THEN_INT :=220,
    Rule_4_IF_IV1d_THEN_INT :=240,
    Rule_5_IF_IV1e_THEN_INT :=260,
    Output_variable_INT=>215,
    Parameterization_fault_BOOL=>0,
    First_active_rule_UINT=>2,
    Second_active_rule_UINT=>3,
    First_activity_level_10ths_percent_UINT=>500,
    Second_activity_level_10ths_percent_UINT=>500
    )
LD  Fuzzy_setpoint_optimization.Output_variable_INT
ST  Setpoint_temperature_10th_degree_celsius

END_PROGRAM
```

Fuzzy logic system:
1 input, 7 terms

U_FUZ_17_FUZZY_1I_7T
Fuzzy Logic System with 1 Input Variable, 7 Terms per Input Variable and 7 Fuzzy Logic Rules

Description

See the function block "U_FUZ_15_FUZZY_1I_5T".
The only difference to the function block "U_FUZ_15_FUZZY_1I_5T" is that this function block has: 1 fuzzy input variables, 7 terms and 7 rules.

Fuzzy logic system:
1 input, 9 terms

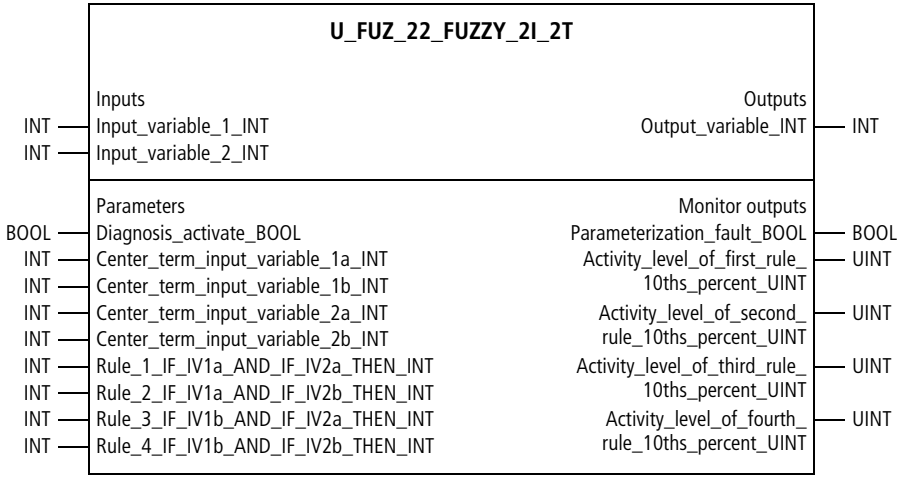
U_FUZ_19_FUZZY_1I_9T
Fuzzy Logic System with 1 Input Variable, 9 Terms per Input Variable and 9 Fuzzy Logic Rules

Description

See the function block "U_FUZ_15_FUZZY_1I_5T".
The only difference to the function block "U_FUZ_15_FUZZY_1I_5T" is that this function block has: 1 fuzzy input variables, 9 terms and 9 rules.

Fuzzy logic system: 2 inputs, 2 terms

U_FUZ_22_FUZZY_2I_2T
Fuzzy Logic System with 2 Input Variables, 2 Terms per Input Variable and 4 Fuzzy Logic Rules



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_variable_1_INT	First linguistic input variable	–32768 to 32767
Input_variable_2_INT	Second linguistic input variable	–32768 to 32767
Parameters		
Diagnosis_activate_BOOL	Activates diagnostic mode	0/1
Center_term_input_variable_1a_INT	Input variable 1: term a	–32768 to 32767
Center_term_input_variable_1b_INT	Input variable 1: term b	–32768 to 32767
Center_term_input_variable_2a_INT	Input variable 2: term a	–32768 to 32767
Center_term_input_variable_2b_INT	Input variable 2: term b	–32768 to 32767

Designation	Significance	Value range
Rule_1_IF_IV1a_AND_IF_IV2a_ THEN_INT	Fuzzy rule 1	–32768 to 32767
Rule_2_IF_IV1a_AND_IF_IV2b_ THEN_INT	Fuzzy rule 2	–32768 to 32767
Rule_3_IF_IV1b_AND_IF_IV2a_ THEN_INT	Fuzzy rule 3	–32768 to 32767
Rule_4_IF_IV1b_AND_IF_IV2b_ THEN_INT	Fuzzy rule 4	–32768 to 32767
Outputs		
Output_variable_INT	Linguistic output variable	–32768 to 32767
Monitor outputs		
Parameterization_fault_BOOL	Status: parameterization fault	0/1
Activity_level_of_first_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 1	0 to 1000
Activity_level_of_second_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 2	0 to 1000
Activity_level_of_third_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 3	0 to 1000
Activity_level_of_fourth_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 4	0 to 1000

Description

For each of the linguistic input variables "Input_variable_1_INT" and "Input_variable_2_INT", two terms can be assigned values, with the variables "Center_term_input_variable_. . ._INT".

This results in the arrangement of terms shown in figure 62 (→ above). The values assigned to the terms can be in any unit, such as percentage, tenths of a percent, °C, 12-bit, etc. Each possible combination of the input variables for these terms is assigned to an output variable (= THEN assignment of the fuzzy rule = characteristic curve term), resulting in 4 fuzzy logic rules.

Example:

- Fuzzy rule 1: With the variable "Rule_1_IF_IV1a_AND_IF_IV2a_THEN_", an output variable is assigned to the combination ["term a, input variable 1" | "term a, input variable 2"]. The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1a" AND
    IF input variable 2 is "2a"
    THEN the output variable has the value "X1"
    (any integer value can be assigned).

- Fuzzy rule 2: With the variable "Rule_2_IF_IV1a_AND_IF_IV2b_THEN_INT", an output variable is assigned to the combination ["term a, input variable 1" | "term b, input variable 2"]. The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1a" AND
    IF input variable 2 is "2b"
    THEN the output variable has the value "X2"
    (any integer value can be assigned).

In the output section of the function block, an output variable is calculated as a function of the input variables. This output variable has the same unit as the THEN assignments of the fuzzy logic rules. Standardisation or rescaling of parameter values is not required for the fuzzy logic function blocks.

Enter the terms of the input variables in ascending order. If this is not done, the status display "Parameterization_fault_BOOL" will be "1". Figure 69 shows a fuzzy logic characteristic field resulting from the values of the terms and from the fuzzy logic rules (→ application example).

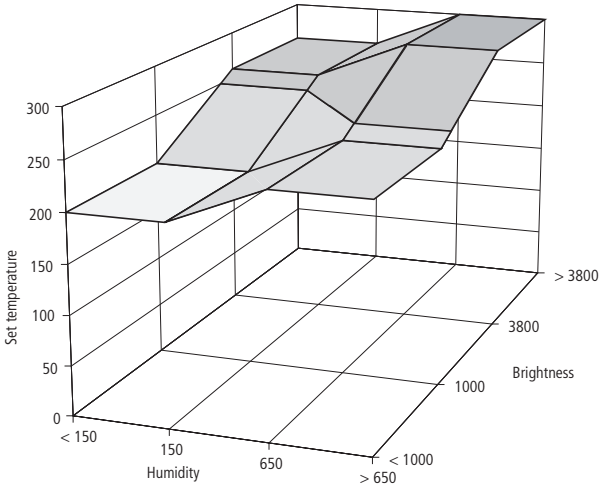


Figure 69: Fuzzy logic characteristic field for 2 input variables and 2 terms (→ application example)

"Diagnosis_activate_BOOL=1" displays the activity levels of the fuzzy logic rules in the output section (the function block's cycle time requirement is doubled when the diagnosis is enabled). Detailed descriptions of fuzzy logic characteristic fields and the activity levels of fuzzy logic rules can be found at the beginning of this chapter.

Example:

Assume practical experience in optimising the desired temperature of a hothouse as a function of two input variables, as follows:

- input variable 1 = humidity [‰]
- input variable 2 = brightness [12 bits]
- output variable = desired temperature [1/10 °C]

Minimum humidity value = 150
Maximum humidity value = 650
Minimum brightness value = 1000
Maximum brightness value = 3800

Fuzzy rule	Humidity [‰]	Brightness [12 bits]	Desired temperature [1/10 °C]
1	150	1000	200
2	150	3800	260
3	650	1000	240
4	650	3800	300

Table 8: Fuzzy logic rules (basis for the rules) in the form of a matrix

Brightness [12 bits]	Humidity [‰]	
	150	650
1000	Rule 1 = 200	Rule 3 = 240
3800	Rule 2 = 260	Rule 4 = 300

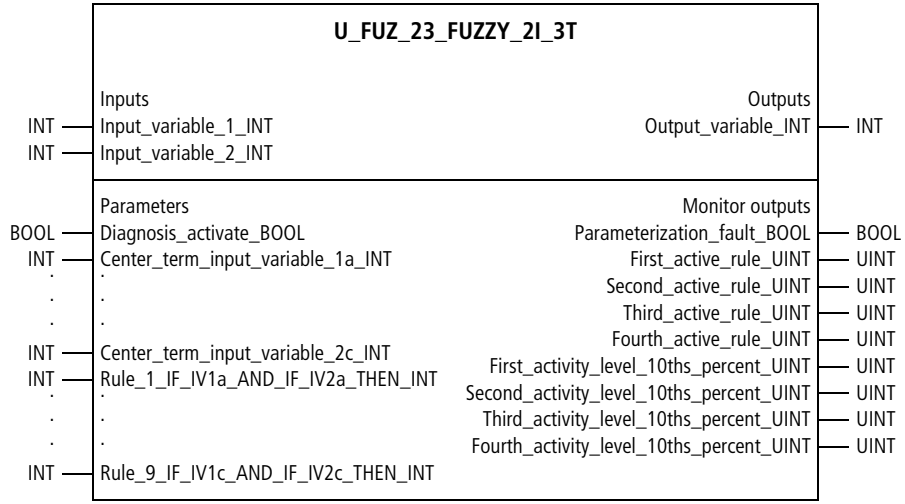
The input variables "Humidity = 300" and "Brightness = 2500" result in a desired temperature of 24.4 °C. The corresponding levels of activity are shown in the output section of the function block.

Application of the function block "U_FUZ_22_FUZZY_2I_2T" in the program "FuzSetpo"

```
PROGRAM FuzSetpo
VAR
    Fuzzy_setpoint_optimization : U_FUZ_22_FUZZY_2I_2T ;
    Humidity_10ths_percent : INT :=300 ;
    Brightness_12Bit : INT :=2500 ;
    Setpoint_temperature_10th_degree_celsius : INT ;
END_VAR

CAL Fuzzy_setpoint_optimization(
    Input_variable_1_INT :=Humidity_10ths_percent,
    Input_variable_2_INT :=Brightness_12Bit,
    Diagnosis_activate_BOOL :=1,
    Center_term_input_variable_1a_INT :=150,
    Center_term_input_variable_1b_INT :=650,
    Center_term_input_variable_2a_INT :=1000,
    Center_term_input_variable_2b_INT :=3800,
    Rule_1_IF_IV1a_AND_IF_IV2a_THEN_INT :=200,
    Rule_2_IF_IV1a_AND_IF_IV2b_THEN_INT :=260,
    Rule_3_IF_IV1b_AND_IF_IV2a_THEN_INT :=240,
    Rule_4_IF_IV1b_AND_IF_IV2b_THEN_INT :=300,
    Output_variable_INT=>244,
    Parameterization_fault_BOOL=>0,
    Activity_level_of_first_rule_10ths_percent_UINT=>325,
    Activity_level_of_second_rule_10ths_percent_UINT=>374,
    Activity_level_of_third_rule_10ths_percent_UINT=>139,
    Activity_level_of_fourth_rule_10ths_percent_UINT=>160)
LD  Fuzzy_setpoint_optimization.Output_variable_INT
ST  Setpoint_temperature_10th_degree_celsius

END_PROGRAM
```

Fuzzy logic system: 2 inputs, 3 terms

U_FUZ_23_FUZZY_2I_3T
Fuzzy Logic System with 2 Input Variables, 3 Terms per Input Variable and 9 Fuzzy Logic Rules



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_variable_1_INT	First linguistic input variable	–32768 to 32767
Input_variable_2_INT	Second linguistic input variable	–32768 to 32767

Designation	Significance	Value range
Parameters		
Diagnosis_activate_BOOL	Activates diagnostic mode	0/1
Center_term_input_variable_1a_INT	Input variable 1: term a	–32768 to 32767
Center_term_input_variable_1b_INT	Input variable 1: term b	–32768 to 32767
Center_term_input_variable_1c_INT	Input variable 1: term c	–32768 to 32767
Center_term_input_variable_2a_INT	Input variable 2: term a	–32768 to 32767
Center_term_input_variable_2b_INT	Input variable 2: term b	–32768 to 32767
Center_term_input_variable_2c_INT	Input variable 2: term c	–32768 to 32767
Rule_1_IF_IV1a_AND_IF_IV2a_ THEN_INT	Fuzzy rule 1	–32768 to 32767
Rule_2_IF_IV1a_AND_IF_IV2b_ THEN_INT	Fuzzy rule 2	–32768 to 32767
Rule_3_IF_IV1a_AND_IF_IV2c_ THEN_INT	Fuzzy rule 3	–32768 to 32767
Rule_4_IF_IV1b_AND_IF_IV2a_ THEN_INT	Fuzzy rule 4	–32768 to 32767
Rule_5_IF_IV1b_AND_IF_IV2b_ THEN_INT	Fuzzy rule 5	–32768 to 32767
Rule_6_IF_IV1b_AND_IF_IV2c_ THEN_INT	Fuzzy rule 6	–32768 to 32767
Rule_7_IF_IV1c_AND_IF_IV2a_ THEN_INT	Fuzzy rule 7	–32768 to 32767
Rule_8_IF_IV1c_AND_IF_IV2b_ THEN_INT	Fuzzy rule 8	–32768 to 32767
Rule_9_IF_IV1c_AND_IF_IV2c_ THEN_INT	Fuzzy rule 9	–32768 to 32767

Designation	Significance	Value range
Outputs		
Output_variable_INT	Linguistic output variable	–32768 to 32767
Monitor outputs		
Parameterization_fault_BOOL	Status: parameterization fault	0/1
First_active_rule_UINT	First active fuzzy rule	1 to 9
Second_active_rule_UINT	Second active fuzzy rule	1 to 9
Third_active_rule_UINT	Third active fuzzy rule	1 to 9
Fourth_active_rule_UINT	Fourth active fuzzy rule	1 to 9
First_activity_level_10ths_ percent_UINT	Activity level of the first active rule	0 to 1000
Second_activity_level_10ths_ percent_UINT	Activity level of the second active rule	0 to 1000
Third_activity_level_10ths_ percent_UINT	Activity level of the third active rule	0 to 1000
Fourth_activity_level_10ths_ percent_UINT	Activity level of the fourth active rule	0 to 1000

Description

For each of the linguistic input variables "Input_variable_1_INT" and "Input_variable_2_INT", three terms can be assigned values, with the variables "Center_term_input_variable_. . ._INT". This results in the arrangement of terms shown in figure 63 (→ above). The values assigned to the terms can be in any unit, such as percentage, tenths of a percent, °C, 12-bit, etc. Each possible combination of the input variables for these terms is assigned to an output variable (= THEN assignment of the fuzzy rule = characteristic curve term), resulting in 9 fuzzy logic rules.

Example:

- Fuzzy rule 1:
  With the variable "Rule_1_IF_IV1a_AND_IF_ IV2a_THEN_", an output variable is assigned to the combination ["term a, input variable 1" | "term a, input variable 2"]. The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1a" AND
    IF input variable 2 is "2a"
    THEN the output variable has the value "X1"
    (any integer value can be assigned).

- Fuzzy rule 2:
  With the variable "Rule_2_IF_IV1a_AND_IF_ IV2b_THEN_INT", an output variable is assigned to the combination ["term a, input variable 1" | "term b, input variable 2"]. The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1a" AND
    IF input variable 2 is "2b"
    THEN the output variable has the value
    "X2"(any integer value can be assigned).

In the output section of the function block, an output variable is calculated as a function of the input variables. This output variable has the same unit as the THEN assignments of the fuzzy logic rules. Standardisation or rescaling of parameter values is not required for the fuzzy logic function blocks.

Enter the terms of the input variables in ascending order. If this is not done, the status display "Parameterization_fault_BOOL" will be "1". Figure 70 shows a fuzzy logic characteristic field resulting from the values of the terms and from the fuzzy logic rules (→ application example).

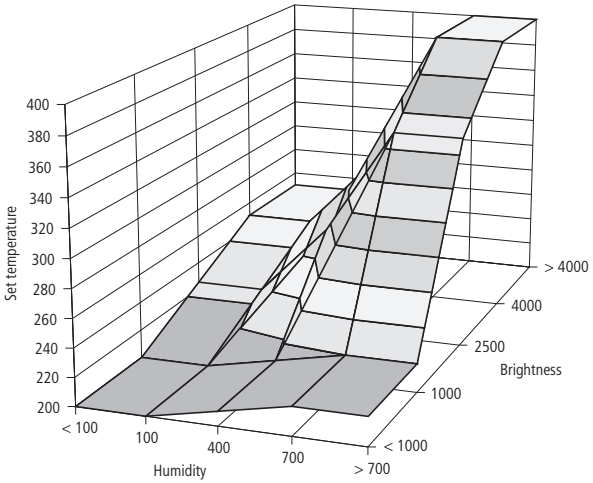


Figure 70: Fuzzy logic characteristic field for 2 input variables and 3 terms (→ application example)

"Diagnosis_activate_BOOL=1" displays the activity levels of the fuzzy logic rules in the output section (the function block's cycle time requirement is doubled when the diagnosis is enabled). The diagnosis shows which rules are active (always 4 rules simultaneously, → application example) and the level of activity for each rule. Detailed descriptions of fuzzy logic characteristic fields and the activity levels of fuzzy logic rules can be found at the beginning of this chapter.

Example:
Assume practical experience in optimising the desired temperature of a hothouse as a function of two input variables, as follows:

- input variable 1 = humidity [‰]
- input variable 2 = brightness [12 bits]
- output variable = desired temperature [1/10 °C]

Minimum humidity value = 100
Medium humidity value = 400
Maximum humidity value = 700
Minimum brightness value = 1000
Medium brightness value = 2500
Maximum brightness value = 4000

Fuzzy rule	Humidity [‰]	Brightness [12 bits]	Desired temperature [1/10 °C]
1	100	1000	200
2	100	2500	225
3	100	4000	250
4	400	1000	210
5	400	2500	275
6	400	4000	300
7	700	1000	220
8	700	2500	350
9	700	4000	400

Table 9: Fuzzy logic rules (basis for the rules) in the form of a matrix

Brightness [12 bits]	Humidity [‰]		
	150	650	700
1000	Rule 1 = 200	Rule 4 = 210	Rule 7 = 220
2500	Rule 2 = 225	Rule 5 = 275	Rule 8 = 350
4000	Rule 3 = 250	Rule 6 = 300	Rule 9 = 400

The input variables "Humidity = 300" and "Brightness = 3000" result in a desired temperature of 26.6 °C. The fuzzy logic rules 2, 3, 5 and 6 are active. The corresponding activity levels are 222, 111, 444 and 221 ‰ (→ the output section of the function block).

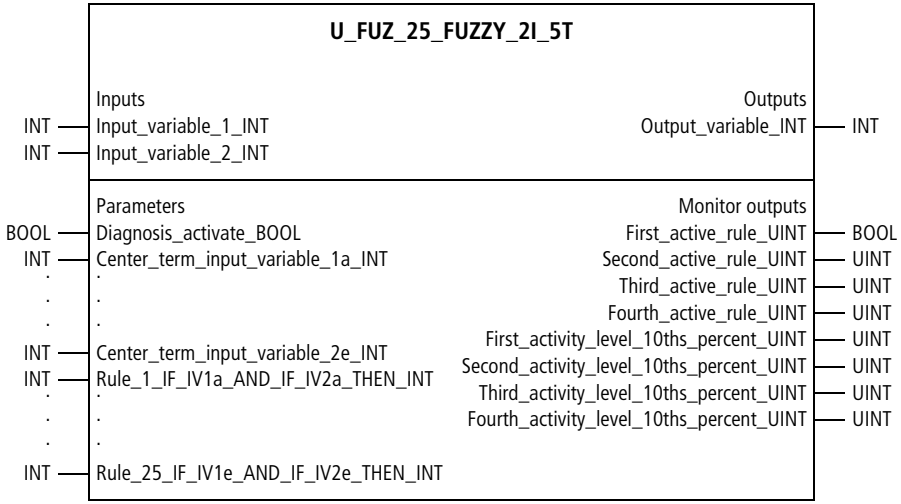
Application of the function block "U_FUZ_23_FUZZY_2I_3T" in the program "Fuz_23"

```
PROGRAM Fuz_23
VAR
    Fuzzy_setpoint_optimization : U_FUZ_23_FUZZY_2I_3T ;
    Humidity_10ths_percent : INT :=300 ;
    Brightness_12Bit : INT :=3000 ;
    Setpoint_temperature_10th_degree_celsius : INT ;
END_VAR

CAL Fuzzy_setpoint_optimization(
    Input_variable_1_INT :=Humidity_10ths_percent,
    Input_variable_2_INT :=Brightness_12Bit,
    Diagnosis_activate_BOOL :=1,
    Center_term_input_variable_1a_INT :=100,
    Center_term_input_variable_1b_INT :=400,
    Center_term_input_variable_1c_INT :=700,
    Center_term_input_variable_2a_INT :=1000,
    Center_term_input_variable_2b_INT :=2500,
    Center_term_input_variable_2c_INT :=4000,
    Rule_1_IF_IV1a_AND_IF_IV2a_THEN_INT :=200,
    Rule_2_IF_IV1a_AND_IF_IV2b_THEN_INT :=225,
    Rule_3_IF_IV1a_AND_IF_IV2c_THEN_INT :=250,
    Rule_4_IF_IV1b_AND_IF_IV2a_THEN_INT :=210,
    Rule_5_IF_IV1b_AND_IF_IV2b_THEN_INT :=275,
    Rule_6_IF_IV1b_AND_IF_IV2c_THEN_INT :=300,
    Rule_7_IF_IV1c_AND_IF_IV2a_THEN_INT :=220,
    Rule_8_IF_IV1c_AND_IF_IV2b_THEN_INT :=350,
    Rule_9_IF_IV1c_AND_IF_IV2c_THEN_INT :=400,
```

```
    Output_variable_INT=>266,
    Parameterization_fault_BOOL=>0,
    First_active_rule_UINT=>2,
    Second_active_rule_UINT=>3,
    Third_active_rule_UINT=>5,
    Fourth_active_rule_UINT=>6,
    First_activity_level_10ths_percent_UINT=>222,
    Second_activity_level_10ths_percent_UINT=>111,
    Third_activity_level_10ths_percent_UINT=>444,
    Fourth_activity_level_10ths_percent_UINT=>221)
LD  Fuzzy_setpoint_optimization.Output_variable_INT
ST  Setpoint_temperature_10th_degree_celsius

END_PROGRAM
```

Fuzzy logic system: 2 inputs, 5 terms

U_FUZ_25_FUZZY_2I_5T
Fuzzy Logic System with 2 Input Variables, 5 Terms per Input Variable and 25 Fuzzy Rules



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_variable_1_INT	First linguistic input variable	–32768 to 32767
Input_variable_2_INT	Second linguistic input variable	–32768 to 32767
Parameters		
Diagnosis_activate_BOOL	Activates diagnostic mode	0/1
Center_term_input_variable_1a_INT	Input variable 1: term a	–32768 to 32767
Center_term_input_variable_1b_INT	Input variable 1: term b	–32768 to 32767
Center_term_input_variable_1c_INT	Input variable 1: term c	–32768 to 32767
Center_term_input_variable_1d_INT	Input variable 1: term d	–32768 to 32767
Center_term_input_variable_1e_INT	Input variable 1: term e	–32768 to 32767

Designation	Significance	Value range
Center_term_input_variable_2a_INT	Input variable 2: term a	–32768 to 32767
Center_term_input_variable_2b_INT	Input variable 2: term b	–32768 to 32767
Center_term_input_variable_2c_INT	Input variable 2: term c	–32768 to 32767
Center_term_input_variable_2d_INT	Input variable 2: term d	–32768 to 32767
Center_term_input_variable_2e_INT	Input variable 2: term e	–32768 to 32767
Rule_1_IF_IV1a_AND_IF_IV2a_ THEN_INT	Fuzzy rule 1	–32768 to 32767
Rule_2_IF_IV1a_AND_IF_IV2b_ THEN_INT	Fuzzy rule 2	–32768 to 32767
Rule_3_IF_IV1a_AND_IF_IV2c_ THEN_INT	Fuzzy rule 3	–32768 to 32767
Rule_4_IF_IV1a_AND_IF_IV2d_ THEN_INT	Fuzzy rule 4	–32768 to 32767
Rule_5_IF_IV1a_AND_IF_IV2e_ THEN_INT	Fuzzy rule 5	–32768 to 32767
Rule_6_IF_IV1b_AND_IF_IV2a_ THEN_INT	Fuzzy rule 6	–32768 to 32767
Rule_7_IF_IV1b_AND_IF_IV2b_ THEN_INT	Fuzzy rule 7	–32768 to 32767
Rule_8_IF_IV1b_AND_IF_IV2c_ THEN_INT	Fuzzy rule 8	–32768 to 32767
Rule_9_IF_IV1b_AND_IF_IV2d_ THEN_INT	Fuzzy rule 9	–32768 to 32767
Rule_10_IF_IV1b_AND_IF_IV2e_ THEN_INT	Fuzzy rule 10	–32768 to 32767
Rule_11_IF_IV1c_AND_IF_IV2a_ THEN_INT	Fuzzy rule 11	–32768 to 32767
Rule_12_IF_IV1c_AND_IF_IV2b_ THEN_INT	Fuzzy rule 12	–32768 to 32767

Designation	Significance	Value range
Rule_13_IF_IV1c_AND_IF_IV2c_ THEN_INT	Fuzzy rule 13	–32768 to 32767
Rule_14_IF_IV1c_AND_IF_IV2d_ THEN_INT	Fuzzy rule 14	–32768 to 32767
Rule_15_IF_IV1c_AND_IF_IV2e_ THEN_INT	Fuzzy rule 15	–32768 to 32767
Rule_16_IF_IV1d_AND_IF_IV2a_ THEN_INT	Fuzzy rule 16	–32768 to 32767
Rule_17_IF_IV1d_AND_IF_IV2b_ THEN_INT	Fuzzy rule 17	–32768 to 32767
Rule_18_IF_IV1d_AND_IF_IV2c_ THEN_INT	Fuzzy rule 18	–32768 to 32767
Rule_19_IF_IV1d_AND_IF_IV2d_ THEN_INT	Fuzzy rule 19	–32768 to 32767
Rule_20_IF_IV1d_AND_IF_IV2e_ THEN_INT	Fuzzy rule 20	–32768 to 32767
Rule_21_IF_IV1e_AND_IF_IV2a_ THEN_INT	Fuzzy rule 21	–32768 to 32767
Rule_22_IF_IV1e_AND_IF_IV2b_ THEN_INT	Fuzzy rule 22	–32768 to 32767
Rule_23_IF_IV1e_AND_IF_IV2c_ THEN_INT	Fuzzy rule 23	–32768 to 32767
Rule_24_IF_IV1e_AND_IF_IV2d_ THEN_INT	Fuzzy rule 24	–32768 to 32767
Rule_25_IF_IV1e_AND_IF_IV2e_ THEN_INT	Fuzzy rule 25	–32768 to 32767
Outputs		
Output_variable_INT	Linguistic output variable	–32768 to 32767

Designation	Significance	Value range
Monitor outputs		
Parameterization_fault_BOOL	Status: parameterization fault	0/1
First_active_rule_UINT	First active fuzzy rule	1 to 25
Second_active_rule_UINT	Second active fuzzy rule	1 to 25
Third_active_rule_UINT	Third active fuzzy rule	1 to 25
Fourth_active_rule_UINT	Fourth active fuzzy rule	1 to 25
First_activity_level_10ths_ percent_UINT	Activity level of the first active rule	0 to 1000
Second_activity_level_10ths_ percent_UINT	Activity level of the second active rule	0 to 1000
Third_activity_level_10ths_ percent_UINT	Activity level of the third active rule	0 to 1000
Fourth_activity_level_10ths_ percent_UINT	Activity level of the fourth active rule	0 to 1000

Description

For each of the linguistic input variables "Input_variable_1_INT" and "Input_variable_2_INT", five terms can be assigned values, with the variables "Center_term_input_variable_. . ._INT". This results in the arrangement of terms shown in figure 64 (→ above). The values assigned to the terms can be in any unit, such as percentage, tenths of a percent, °C, 12-bit, etc. Each possible combination of the input variables for these terms is assigned to an output variable (= THEN assignment of the fuzzy rule = characteristic curve term), resulting in 25 fuzzy logic rules.

Example:

- Fuzzy rule 1:
  With the variable "Rule_1_IF_IV1a_AND_IF_ IV2a_THEN_", an output variable is assigned to the combination ["term a, input variable 1" | "term a, input variable 2"]. The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1a" AND
    IF input variable 2 is "2a"
    THEN the output variable has the value "X1"
    (any integer value can be assigned).

- Fuzzy rule 2:
  With the variable "Rule_2_IF_IV1a_AND_IF_IV2b_ THEN_INT", an output variable is assigned to the combination ["term a, input variable 1" | "term b, input variable 2"]. The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1a" AND
    IF input variable 2 is "2b"
    THEN the output variable has the value "X2"
    (any integer value can be assigned).

In the output section of the function block, an output variable is calculated as a function of the input variables. This output variable has the same unit as the THEN assignments of the fuzzy logic rules. Standardisation or rescaling of parameter values is not required for the fuzzy logic function blocks.

Enter the terms of the input variables in ascending order. If this is not done, the status display "Parameterization_fault_BOOL" will be "1". Figure 71 shows a fuzzy logic characteristic field resulting from the values of the terms and from the fuzzy logic rules (→ application example).

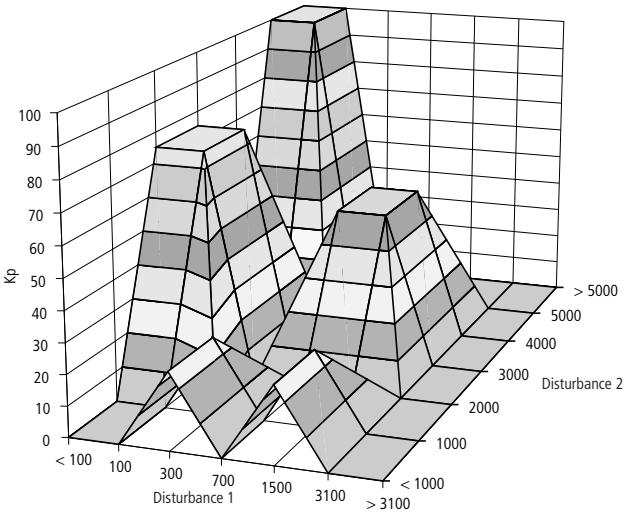


Figure 71: Fuzzy logic characteristic field for 2 input variables and 5 terms (→ application example)

"Diagnosis_activate_BOOL=1" displays the activity levels of the fuzzy logic rules in the output section (the function block's cycle time requirement is doubled when the diagnosis is enabled). The diagnosis shows which rules are active (always 4 rules simultaneously, → application example) and the level of activity for each rule. Detailed descriptions of fuzzy logic characteristic fields and the activity levels of fuzzy logic rules can be found at the beginning of this chapter.

Example:
Assume practical experience about the Kp setting of a PID-controller as a function of two disturbance variables, as follows:

- input variable 1 = disturbance variable 1 [‰] (values greater than 1000 are possible, e.g. 3524 ‰ = 3.524)
- input variable 2 = disturbance variable 2 [‰]
- output variable = Kp [%]

disturbance variable 1, very small = 100
disturbance variable 1, small = 300
disturbance variable 1, medium = 700

disturbance variable 1, large = 1500
disturbance variable 1, very large = 3100

disturbance variable 2, very small = 1000
disturbance variable 2, small = 2000
disturbance variable 2, medium = 3000
disturbance variable 2, large = 4000
disturbance variable 2, very large = 5000

Fuzzy rule	Disturbance variable 1 [‰]	Disturbance variable 2 [12 Bit]	Output variable (Kp) [1/10 °C]
1	100	1000	0
2	100	2000	75
3	100	3000	75
4	100	4000	0
5	100	5000	100
6	300	1000	0
7	300	2000	75
8	300	3000	75
9	300	4000	0
10	300	5000	100
11	700	1000	25
12	700	2000	0
13	700	3000	0
14	700	4000	0
15	700	5000	0

Fuzzy rule	Disturbance variable 1 [‰]	Disturbance variable 2 [12 Bit]	Output variable (Kp) [1/10 °C]
16	1500	1000	0
17	1500	2000	0
18	1500	3000	50
19	1500	4000	50
20	1500	5000	0
21	3100	1000	25
22	3100	2000	0
23	3100	3000	50
24	3100	4000	50
25	3100	5000	0

Table 10: Fuzzy logic rules (basis for the rules) in the form of a matrix

Input variable 2 [‰]	Input variable 1 [‰]				
	100	400	700	1500	3100
1000	Rule 1 = 0	Rule 6 = 0	Rule 11 = 25	Rule 16 = 0	Rule 21 = 25
2000	Rule 2 = 75	Rule 7 = 75	Rule 12 = 0	Rule 17 = 0	Rule 22 = 0
3000	Rule 3 = 75	Rule 8 = 75	Rule 13 = 0	Rule 18 = 50	Rule 23 = 50
4000	Rule 4 = 0	Rule 9 = 0	Rule 14 = 0	Rule 19 = 50	Rule 24 = 50
5000	Rule 5 = 100	Rule 10 = 100	Rule 15 = 0	Rule 20 = 0	Rule 25 = 0

The input variables "Disturbance variable 1 = 150" and "Disturbance variable 2 = 3250" produce the output variable "Kp = 57". Fuzzy logic rules 3, 4, 8 and 9 are active. The corresponding levels of activity are 562, 187,187 and 62 ‰ (→ the output section of the function block).

Application of the function block "U_FUZ_25_FUZZY_2I_5T" in the program "Fuzzy_P"

```
PROGRAM Fuzzy_P
VAR
    Fuzzy_proportional_rate_adjustment : U_FUZ_25_FUZZY_2I_5T ;
    PID_CONTROLLER_1 : U_PID_CONTROLLER ;
    Disturbance1_10ths_percent : INT :=150 ;
    Disturbance2_10ths_percent : INT :=3250 ;
END_VAR

CAL Fuzzy_proportional_rate_adjustment(
    Input_variable_1_INT :=disturbance1_10ths_percent,
    Input_variable_2_INT :=disturbance2_10ths_percent,
    Diagnosis_activate_BOOL :=0,
    Center_term_input_variable_1a_INT :=100,
    Center_term_input_variable_1b_INT :=300,
    Center_term_input_variable_1c_INT :=700,
    Center_term_input_variable_1d_INT :=1500,
    Center_term_input_variable_1e_INT :=3100,
    Center_term_input_variable_2a_INT :=1000,
    Center_term_input_variable_2b_INT :=2000,
    Center_term_input_variable_2c_INT :=3000,
    Center_term_input_variable_2d_INT :=4000,
    Center_term_input_variable_2e_INT :=5000,
    Rule_1_IF_IV1a_AND_IF_IV2a_THEN_INT :=0,
    Rule_2_IF_IV1a_AND_IF_IV2b_THEN_INT :=75,
    Rule_3_IF_IV1a_AND_IF_IV2c_THEN_INT :=75,
    Rule_4_IF_IV1a_AND_IF_IV2d_THEN_INT :=0,
    Rule_5_IF_IV1a_AND_IF_IV2e_THEN_INT :=100,
    Rule_6_IF_IV1b_AND_IF_IV2a_THEN_INT :=0,
    Rule_7_IF_IV1b_AND_IF_IV2b_THEN_INT :=75,
    Rule_8_IF_IV1b_AND_IF_IV2c_THEN_INT :=75,
    Rule_9_IF_IV1b_AND_IF_IV2d_THEN_INT :=0,
    Rule_10_IF_IV1b_AND_IF_IV2e_THEN_INT :=100,
```

```
    Rule_11_IF_IV1c_AND_IF_IV2a_THEN_INT :=25,
    Rule_12_IF_IV1c_AND_IF_IV2b_THEN_INT :=0,
    Rule_13_IF_IV1c_AND_IF_IV2c_THEN_INT :=0,
    Rule_14_IF_IV1c_AND_IF_IV2d_THEN_INT :=0,
    Rule_15_IF_IV1c_AND_IF_IV2e_THEN_INT :=0,
    Rule_16_IF_IV1d_AND_IF_IV2a_THEN_INT :=0,
    Rule_17_IF_IV1d_AND_IF_IV2b_THEN_INT :=0,
    Rule_18_IF_IV1d_AND_IF_IV2c_THEN_INT :=50,
    Rule_19_IF_IV1d_AND_IF_IV2d_THEN_INT :=50,
    Rule_20_IF_IV1d_AND_IF_IV2e_THEN_INT :=0,
    Rule_21_IF_IV1e_AND_IF_IV2a_THEN_INT :=25,
    Rule_22_IF_IV1e_AND_IF_IV2b_THEN_INT :=0,
    Rule_23_IF_IV1e_AND_IF_IV2c_THEN_INT :=50,
    Rule_24_IF_IV1e_AND_IF_IV2d_THEN_INT :=50,
    Rule_25_IF_IV1e_AND_IF_IV2e_THEN_INT :=0,
    Output_variable_INT=>57,
    Parameterization_fault_BOOL=>0,
    First_active_rule_UINT=>3,
    Second_active_rule_UINT=>4,
    Third_active_rule_UINT=>8,
    Fourth_active_rule_UINT=>9,
    First_activity_level_10ths_percent_UINT=>562,
    Second_activity_level_10ths_percent_UINT=>187,
    Third_activity_level_10ths_percent_UINT=>187,
    Fourth_activity_level_10ths_percent_UINT=>62)
LD  Fuzzy_proportional_rate_adjustment.Output_variable_INT
INT_TO_UINT
ST  PID_CONTROLLER_1.Proportional_rate_percent_UINT
END_PROGRAM
```

**Fuzzy logic system:
2 inputs, 7 terms**

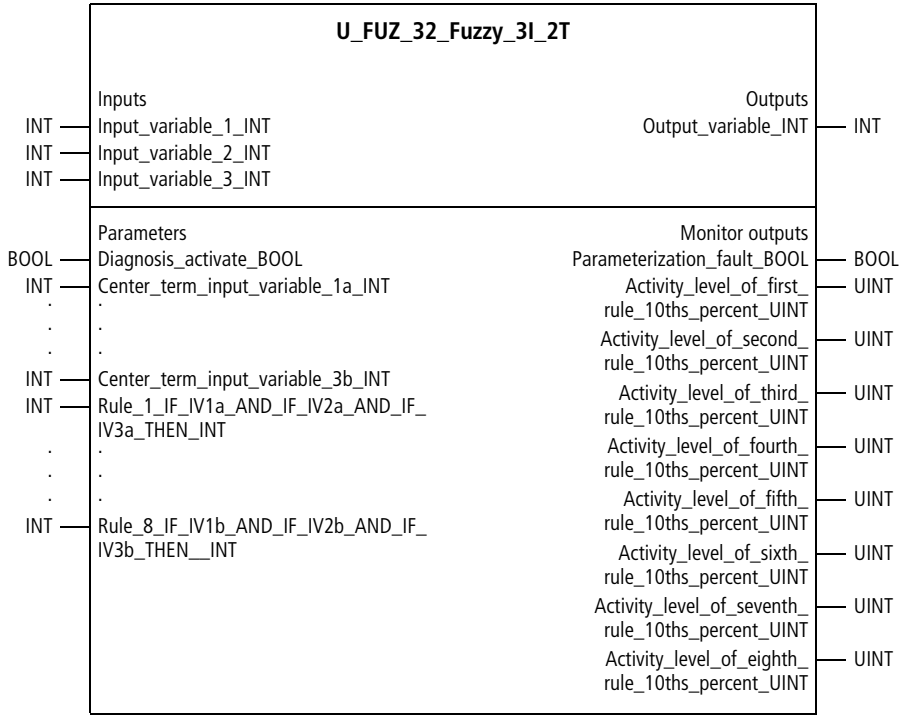
U_FUZ_27_FUZZY_2I_7T
Fuzzy Logic System with 2 Input Variable, 7 Terms per Input Variable and 49 Fuzzy Logic Rules

Description

See the function block "U_FUZ_25_FUZZY_2I_5T".
The only difference to the function block "U_FUZ_25_FUZZY_2I_5T" is that this function block has: 2 fuzzy input variables, 7 terms and 49 rules.

Fuzzy logic system: 3 inputs, 2 terms

U_FUZ_32_FUZZY_3I_2T Fuzzy System with 3 Input Variables, 2 Terms per Input Variable and 8 Fuzzy Logic Rules



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_variable_1_INT	First linguistic input variable	–32768 to 32767
Input_variable_2_INT	Second linguistic input variable	–32768 to 32767
Input_variable_3_INT	Third linguistic input variable	–32768 to 32767

Designation	Significance	Value range
Parameters		
Diagnosis_activate_BOOL	Activates diagnostic mode	0/1
Center_term_input_variable_1a_INT	Input variable 1: term a	–32768 to 32767
Center_term_input_variable_1b_INT	Input variable 1: term b	–32768 to 32767
Center_term_input_variable_2a_INT	Input variable 2: term a	–32768 to 32767
Center_term_input_variable_2b_INT	Input variable 2: term b	–32768 to 32767
Center_term_input_variable_3a_INT	Input variable 3: term a	–32768 to 32767
Center_term_input_variable_3b_INT	Input variable 3: term b	–32768 to 32767
Rule_1_IF_IV1a_AND_IF_IV2a_ AND_IF_IV3a_THEN_INT	Fuzzy rule 1	–32768 to 32767
Rule_2_IF_IV1a_AND_IF_IV2a_ AND_IF_IV3b_THEN_INT	Fuzzy rule 2	–32768 to 32767
Rule_3_IF_IV1a_AND_IF_IV2b_ AND_IF_IV3a_THEN_INT	Fuzzy rule 3	–32768 to 32767
Rule_4_IF_IV1a_AND_IF_IV2b_ AND_IF_IV3b_THEN_INT	Fuzzy rule 4	–32768 to 32767
Rule_5_IF_IV1b_AND_IF_IV2a_ AND_IF_IV3a_THEN_INT	Fuzzy rule 5	–32768 to 32767
Rule_6_IF_IV1b_AND_IF_IV2a_ AND_IF_IV3b_THEN_INT	Fuzzy rule 6	–32768 to 32767
Rule_7_IF_IV1b_AND_IF_IV2b_ AND_IF_IV3a_THEN_INT	Fuzzy rule 7	–32768 to 32767
Rule_8_IF_IV1b_AND_IF_IV2b_ AND_IF_IV3b_THEN_INT	Fuzzy rule 8	–32768 to 32767
Outputs		
Output_variable_INT	Linguistic output variable	–32768 to 32767

Designation	Significance	Value range
Monitor outputs		
Parameterization_fault_BOOL	Status: parameterization fault	0/1
Activity_level_of_first_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 1	0 to 1000
Activity_level_of_second_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 2	0 to 1000
Activity_level_of_third_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 3	0 to 1000
Activity_level_of_fourth_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 4	0 to 1000
Activity_level_of_fifth_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 5	0 to 1000
Activity_level_of_sixth_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 6	0 to1000
Activity_level_of_seventh_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 7	0 to 1000
Activity_level_of_eighth_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 8	0 to 1000

Description

For each of the linguistic input variables "Input_variable_1_INT", "Input_variable_2_INT" and "Input_variable_3_INT", two terms can be assigned values, with the variables "Center_term_input_variable_. . ._INT". This results in the arrangement of terms shown in figure 62 (→ above). The values assigned to the terms can be in any unit, such as percentage, tenths of a percent, °C, 12-bit, etc. Each possible combination of the input variables for these terms is assigned to an output variable (= THEN assignment of the fuzzy logic rule = characteristic curve term), resulting in 8 fuzzy logic rules.

Example:

- Fuzzy rule 1:
  With the variable "Rule_1_IF_IV1a_AND_IF_IV2a_ AND_IF_IV3a_THEN_INT", an output variable is assigned to the combination ["term a, input variable 1" | "term a, input variable 2" | "term a, input variable 3"]. The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1a" AND
    IF input variable 2 is "2a" AND
    IF input variable 3 is "3a"
    THEN the output variable has the value "X1"
    (any integer value can be assigned).

- Fuzzy rule 2:
  With the variable "Rule_2_IF_IV1a_AND_IF_IV2a_ AND_IF_IV3b_THEN_INT", an output variable is assigned to the combination ["term a, input variable 1" | "term a, input variable 2" | "term b, input variable 3"]. The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1a" AND
    IF input variable 2 is "2a" AND
    IF input variable 3 is "3b"
    THEN the output variable has the value "X2"
    (any integer value can be assigned).

In the output section of the function block, an output variable is calculated as a function of the input variables. This output variable has the same unit as the THEN assignments of the fuzzy logic rules. Standardisation or rescaling of parameter values is not required for the fuzzy logic function blocks.

Enter the terms of the input variables in ascending order. If this is not done, the status display "Parameterization_fault_BOOL" will be "1". The fuzzy logic function block generates a four-dimensional characteristic field which cannot be shown in graphical form. If one input variable is kept constant, then a three-dimensional characteristic field can be drawn as a function of the other input variables (as for the fuzzy logic function blocks with two input variables).

"Diagnosis_activate_BOOL=1" displays the activity levels of the fuzzy logic rules in the output section (the function block's cycle time requirement is doubled when the diagnosis is enabled). Detailed descriptions of fuzzy logic characteristic fields and the activity levels of fuzzy logic rules can be found at the beginning of this chapter.

Example:
Assume practical experience about the Kp setting of a PID-controller as a function of three disturbance variables, as follows:

- input variable 1 = disturbance variable 1 [‰]
  (values greater than 1000 are possible,
  e.g. 3524 ‰ = 3.524)
- input variable 2 = disturbance variable 2 [‰]
- input variable 3 = disturbance variable 3 [‰]
- output variable = Kp [%]

disturbance variable 1, small = 1300
disturbance variable 1, large = 2500
disturbance variable 2, small = 1300
disturbance variable 2, large = 2000
disturbance variable 3, small = 1000
disturbance variable 3, large = 3000

Fuzzy rule	Disturbance variable 1 [‰]	Disturbance variable 2 [‰]	Disturbance variable 3 [‰]	Output variable (Kp) [%]
1	1300	1300	1000	10
2	1300	1300	3000	20
3	1300	2000	1000	30
4	1300	2000	3000	40
5	2500	1300	1000	50
6	2500	1300	3000	60
7	2500	2000	1000	70
8	2500	2000	3000	80

The input variables "Disturbance variable 1 = 1600", "Disturbance variable 2 = 1400" and "Disturbance variable 3 = 2500" produce the output variable "Kp = 30". The activity levels of the fuzzy logic rules are "161", "482", "27", "80", "54", "161", "9" and "27" (→ the output section of the function block).

Application of the function block "U_FUZ_32_FUZZY_3I_2T" in the program "Fuz32_P"

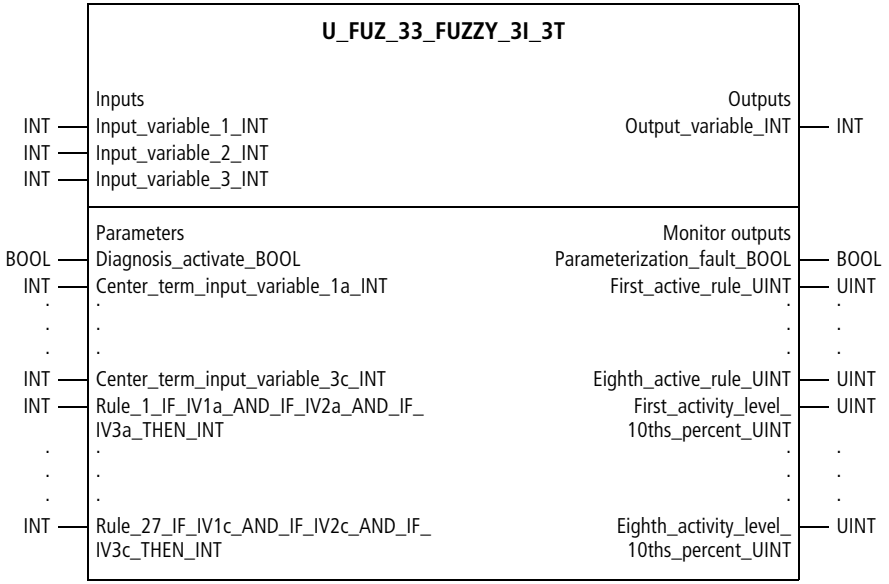
```
PROGRAM Fuz32_P
VAR
    Fuzzy_proportional_rate_adjustment : U_FUZ_32_FUZZY_3I_2T ;
    PID_CONTROLLER_1 : U_PID_CONTROLLER ;
    Disturbance1_10ths_percent : INT :=1600 ;
    Disturbance2_10ths_percent : INT :=1400 ;
    Disturbance3_10ths_percent : INT :=2500 ;
END_VAR
```

```
CAL Fuzzy_proportional_rate_adjustment(
    Input_variable_1_INT :=disturbance1_10ths_percent,
    Input_variable_2_INT :=disturbance2_10ths_percent,
    Input_variable_3_INT :=disturbance3_10ths_percent,
    Diagnosis_activate_BOOL :=1,
    Center_term_input_variable_1a_INT :=1300,
    Center_term_input_variable_1b_INT :=2500,
    Center_term_input_variable_2a_INT :=1300,
    Center_term_input_variable_2b_INT :=2000,
    Center_term_input_variable_3a_INT :=1000,
    Center_term_input_variable_3b_INT :=3000,
    Rule_1_IF_IV1a_AND_IF_IV2a_AND_IF_IV3a_THEN_INT :=10,
    Rule_2_IF_IV1a_AND_IF_IV2a_AND_IF_IV3b_THEN_INT :=20,
    Rule_3_IF_IV1a_AND_IF_IV2b_AND_IF_IV3a_THEN_INT :=30,
    Rule_4_IF_IV1a_AND_IF_IV2b_AND_IF_IV3b_THEN_INT :=40,
    Rule_5_IF_IV1b_AND_IF_IV2a_AND_IF_IV3a_THEN_INT :=50,
    Rule_6_IF_IV1b_AND_IF_IV2a_AND_IF_IV3b_THEN_INT :=60,
    Rule_7_IF_IV1b_AND_IF_IV2b_AND_IF_IV3a_THEN_INT :=70,
    Rule_8_IF_IV1b_AND_IF_IV2b_AND_IF_IV3b_THEN_INT :=80,
    Output_variable_INT=>30,
    Parameterization_fault_BOOL=>0,
    Activity_level_of_first_rule_10ths_percent_UINT=>161,
    Activity_level_of_second_rule_10ths_percent_UINT=>482,
    Activity_level_of_third_rule_10ths_percent_UINT=>27,
    Activity_level_of_fourth_rule_10ths_percent_UINT=>80,
    Activity_level_of_fifth_rule_10ths_percent_UINT=>54,
    Activity_level_of_sixth_rule_10ths_percent_UINT=>161,
    Activity_level_of_seventh_rule_10ths_percent_UINT=>9,
    Activity_level_of_eighth_rule_10ths_percent_UINT=>27)
LD  Fuzzy_proportional_rate_adjustment.Output_variable_INT
INT_TO_UINT
ST  PID_CONTROLLER_1.Proportional_rate_percent_UINT

END_PROGRAM
```

Fuzzy logic system: 3 inputs, 3 terms

U_FUZ_33_FUZZY_3I_3T

Fuzzy logic system with 3 input variables, 3 terms per input variable and 27 fuzzy logic rules



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_variable_1_INT	First linguistic input variable	–32768 to 32767
Input_variable_2_INT	Second linguistic input variable	–32768 to 32767
Input_variable_3_INT	Third linguistic input variable	–32768 to 32767

Designation	Significance	Value range
Parameters		
Diagnosis_activate_BOOL	Activates diagnostic mode	0/1
Center_term_input_variable_1a_INT	Input variable 1: term a	–32768 to 32767
Center_term_input_variable_1b_INT	Input variable 1: term b	–32768 to 32767
Center_term_input_variable_1c_INT	Input variable 1: term c	–32768 to 32767
Center_term_input_variable_2a_INT	Input variable 2: term a	–32768 to 32767
Center_term_input_variable_2b_INT	Input variable 2: term b	–32768 to 32767
Center_term_input_variable_2c_INT	Input variable 2: term c	–32768 to 32767
Center_term_input_variable_3a_INT	Input variable 3: term a	–32768 to 32767
Center_term_input_variable_3b_INT	Input variable 3: term b	–32768 to 32767
Center_term_input_variable_3c_INT	Input variable 3: term c	–32768 to 32767
Rule_1_IF_IV1a_AND_IF_IV2a_ AND_IF_IV3a_THEN_INT	Fuzzy rule 1	–32768 to 32767
Rule_2_IF_IV1a_AND_IF_IV2a_ AND_IF_IV3b_THEN_INT	Fuzzy rule 2	–32768 to 32767
Rule_3_IF_IV1a_AND_IF_IV2a_ AND_IF_IV3c_THEN _NT	Fuzzy rule 3	–32768 to 32767
Rule_4_IF_IV1a_AND_IF_IV2b_ AND_IF_IV3a_THEN_INT	Fuzzy rule 4	–32768 to 32767
Rule_5_IF_IV1a_AND_IF_IV2b_ AND_IF_IV3b_THEN_INT	Fuzzy rule 5	–32768 to 32767
Rule_6_IF_IV1a_AND_IF_IV2b_ AND_IF_IV3c_THEN_INT	Fuzzy rule 6	–32768 to 32767
Rule_7_IF_IV1a_AND_IF_IV2c_ AND_IF_IV3a_INT	Fuzzy rule 7	–32768 to 32767
Rule_8_IF_IV1a_AND_IF_IV2c_ AND_IF_IV3b_THEN_INT	Fuzzy rule 8	–32768 to 32767
Rule_9_IF_IV1a_AND_IF_IV2c_ AND_IF_IV3c_THEN_INT	Fuzzy rule 9	–32768 to 32767
Rule_10_IF_IV1b_AND_IF_IV2a_ AND_IF_IV3a_THEN_INT	Fuzzy rule 10	–32768 to 32767

Designation	Significance	Value range
Rule_11_IF_IV1b_AND_IF_IV2a_ AND_IF_IV3b_THEN_INT	Fuzzy rule 11	–32768 to 32767
Rule_12_IF_IV1b_AND_IF_IV2a_ AND_IF_IV3c_THEN_INT	Fuzzy rule 12	–32768 to 32767
Rule_13_IF_IV1b_AND_IF_IV2b_ AND_IF_IV3a_THEN_INT	Fuzzy rule 13	–32768 to 32767
Rule_14_IF_IV1b_AND_IF_IV2b_ AND_IF_IV3b_THEN_INT	Fuzzy rule 14	–32768 to 32767
Rule_15_IF_IV1b_AND_IF_IV2b_ AND_IF_IV3c_THEN_INT	Fuzzy rule 15	–32768 to 32767
Rule_16_IF_IV1b_AND_IF_IV2c_ AND_IF_IV3a_THEN_INT	Fuzzy rule 16	–32768 to 32767
Rule_17_IF_IV1b_AND_IF_IV2c_ AND_IF_IV3b_THEN_INT	Fuzzy rule 17	–32768 to 32767
Rule_18_IF_IV1b_AND_IF_IV2c_ AND_IF_IV3c_THEN_INT	Fuzzy rule 18	–32768 to 32767
Rule_19_IF_IV1c_AND_IF_IV2a_ AND_IF_IV3a_THEN_INT	Fuzzy rule 19	–32768 to 32767
Rule_20_IF_IV1c_AND_IF_IV2a_ AND_IF_IV3b_THEN_INT	Fuzzy rule 20	–32768 to 32767
Rule_21_IF_IV1c_AND_IF_IV2a_ AND_IF_IV3c_THEN_INT	Fuzzy rule 21	–32768 to 32767
Rule_22_IF_IV1c_AND_IF_IV2b_ AND_IF_IV3a_THEN_INT	Fuzzy rule 22	–32768 to 32767
Rule_23_IF_IV1c_AND_IF_IV2b_ AND_IF_IV3b_THEN_INT	Fuzzy rule 23	–32768 to 32767
Rule_24_IF_IV1c_AND_IF_IV2b_ AND_IF_IV3c_THEN_INT	Fuzzy rule 24	–32768 to 32767
Rule_25_IF_IV1c_AND_IF_IV2c_ AND_IF_IV3a_THEN_INT	Fuzzy rule 25	–32768 to 32767
Rule_26_IF_IV1c_AND_IF_IV2c_ AND_IF_IV3b_THEN_INT	Fuzzy rule 26	–32768 to 32767
Rule_27_IF_IV1c_AND_IF_IV2c_ AND_IF_IV3c_THEN_INT	Fuzzy rule 27	–32768 to 32767

Designation	Significance	Value range
Outputs		
Output_variable_INT	Linguistic output variable	–32768 to 32767
Monitor outputs		
Parameterization_fault_BOOL	Status: parameterization fault	0/1
First_active_rule_UINT	First active fuzzy rule	0 to 27
Second_active_rule_UINT	Second active fuzzy rule	0 to 27
Third_active_rule_UINT	Third active fuzzy rule	0 to 27
Fourth_active_rule_UINT	Fourth active fuzzy rule	0 to 27
Fifth_active_rule_UINT	Fifth active fuzzy rule	0 to 27
Sixth_active_rule_UINT	Sixth active fuzzy rule	0 to 27
Seventh_active_rule_UINT	Seventh active fuzzy rule	0 to 27
Eighth_active_rule_UINT	Eighth active fuzzy rule	0 to 27
First_activity_level_10ths_ percent_UINT	Activity level of the first active rule	0 to 1000
Second_activity_level_10ths_ percent_UINT	Activity level of the second active rule	0 to 1000
Third_activity_level_10ths_ percent_UINT	Activity level of the third active rule	0 to 1000
Fourth_activity_level_10ths_ percent_UINT	Activity level of the fourth active rule	0 to 1000
Fifth_activity_level_10ths_ percent_UINT	Activity level of the fifth active rule	0 to 1000
Sixth_activity_level_10ths_ percent_UINT	Activity level of the sixth active rule	0 to 1000
Seventh_activity_level_10ths_ percent_UINT	Activity level of the seventh active rule	0 to 1000
Eighth_activity_level_10ths_ percent_UINT	Activity level of the eighth active rule	0 to 1000

Description

For each of the linguistic input variables "Input_variable_1_INT", "Input_variable_2_INT" and "Input_variable_3_INT", three terms can be assigned values, with the variables "Center_term_input_ variable_. . ._INT". This results in the arrangement of terms shown in figure 63 (→ above). The values assigned to the terms can be in any unit, such as percentage, tenths of a percent, °C, 12-bit, etc. Each possible combination of the input variables for these terms is assigned to an output variable (= THEN assignment of the fuzzy logic rule = characteristic curve term), resulting in 27 fuzzy logic rules.

Example:

- Fuzzy rule 1:
  With the variable "Rule_1_IF_IV1a_AND_IF_IV2a_ AND_IF_IV3a_THEN_INT", an output variable is assigned to the combination ["term a, input variable 1" | "term a, input variable 2" | "term a, input variable 3"]. The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1a" AND
    IF input variable 2 is "2a" AND
    IF input variable 3 is "3a"
    THEN the output variable has the value "X1"
    (any integer value can be assigned).

- Fuzzy rule 2:
  With the variable "Rule_2_IF_IV1a_AND_IF_IV2a_ AND_IF_IV3b_THEN_INT", an output variable is assigned to the combination ["term a, input variable 1" | "term a, input variable 2" | "term b, input variable 3"]. The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1a" AND
    IF input variable 2 is "2a" AND
    IF input variable 3 is "3b"
    THEN the output variable has the value "X2"
    (any integer value can be assigned).

In the output section of the function block, an output variable is calculated as a function of the input variables. This output variable has the same unit as the THEN assignments of the fuzzy logic rules. Standardisation or rescaling of parameter values is not required for the fuzzy logic function blocks.

Enter the terms of the input variables in ascending order. If this is not done, the status display "Parameterization_fault_BOOL" will be "1". The fuzzy logic function block generates a four-dimensional characteristic field which cannot be shown in graphical form. If one input variable is kept constant, then a three-dimensional characteristic field can be drawn as a function of the other input variables (as for the fuzzy logic function blocks with two input variables).

"Diagnosis_activate_BOOL=1" displays the activity levels of the fuzzy logic rules in the output section (the function block's cycle time requirement is doubled when the diagnosis is enabled). The diagnosis shows which rules are active (always 8 rules simultaneously, → application example) and the level of activity for each rule. Detailed descriptions of fuzzy logic characteristic fields and the activity levels of fuzzy logic rules can be found at the beginning of this chapter.

Example:
Assume practical experience about the Kp setting of a PID-controller as a function of three disturbance variables, as follows:

- input variable 1 = disturbance variable 1 [‰] (values greater than 1000 are possible, e.g. 3524 ‰ = 3.524)
- input variable 2 = disturbance variable 2 [‰]
- input variable 3 = disturbance variable 3 [‰]
- output variable = Kp [%]

disturbance variable 1, small = 1300
disturbance variable 1, medium = 1800
disturbance variable 1, large = 2500
disturbance variable 2, small = 1100
disturbance variable 2, medium = 1600
disturbance variable 2, large = 2000
disturbance variable 3, small = 1000
disturbance variable 3, medium = 2000
disturbance variable 3, large = 3000

Fuzzy rule	**Disturbance variable 1** [‰]	**Disturbance variable 2** [‰]	**Disturbance variable 3** [‰]	**Output variable (Kp)** [%]
1	1300	1100	1000	10
2	1300	1100	2000	20
3	1300	1100	3000	30
4	1300	1600	1000	40
5	1300	1600	2000	50
6	1300	1600	3000	60
7	1300	2000	1000	70
8	1300	2000	2000	80
9	1300	2000	3000	90
10	1800	1100	1000	100

Fuzzy rule	Disturbance variable 1 [‰]	Disturbance variable 2 [‰]	Disturbance variable 3 [‰]	Output variable (Kp) [%]
11	1800	1100	2000	110
12	1800	1100	3000	120
13	1800	1600	1000	130
14	1800	1600	2000	140
15	1800	1600	3000	150
16	1800	2000	1000	160
17	1800	2000	2000	170
18	1800	2000	3000	180
19	2500	1100	1000	190
20	2500	1100	2000	200
21	2500	1100	3000	210
22	2500	1600	1000	220
23	2500	1600	2000	230
24	2500	1600	3000	240
25	2500	2000	1000	250
26	2500	2000	2000	260
27	2500	2000	3000	270

The input variables "Disturbance variable 1 = 1500", "Disturbance variable 2 = 1500" and "Disturbance variable 3 = 1500" produce the output variable "Kp = 75". Fuzzy logic rules 1, 2, 4, 5, 10, 11, 13 and 14 are active. The activity levels of the corresponding fuzzy logic rules are "60", "60", "240", "240", "40", "40", "160" and "160" (→ the output section of the function block).

Application of the function block "U_FUZ_33_FUZZY_3I_3T" in the program "Fuz33_P"

```
PROGRAM Fuz33_P
VAR
    Fuzzy_proportional_rate_adjustment : U_FUZ_33_FUZZY_3I_3T ;
    PID_CONTROLLER_1 : U_PID_CONTROLLER ;
    Disturbance1_10ths_percent : INT :=1500 ;
    Disturbance2_10ths_percent : INT :=1500 ;
    Disturbance3_10ths_percent : INT :=1500 ;
END_VAR

CAL Fuzzy_proportional_rate_adjustment(
    Input_variable_1_INT :=disturbance1_10ths_percent,
    Input_variable_2_INT :=disturbance2_10ths_percent,
    Input_variable_3_INT :=disturbance3_10ths_percent,
    Diagnosis_activate_BOOL :=1,
    Center_term_input_variable_1a_INT :=1300,
    Center_term_input_variable_1b_INT :=1800,
    Center_term_input_variable_1c_INT :=2500,
    Center_term_input_variable_2a_INT :=1100,
    Center_term_input_variable_2b_INT :=1600,
    Center_term_input_variable_2c_INT :=2000,
    Center_term_input_variable_3a_INT :=1000,
    Center_term_input_variable_3b_INT :=2000,
    Center_term_input_variable_3c_INT :=3000,
    Rule_1_IF_IV1a_AND_IF_IV2a_AND_IF_IV3a_THEN_INT :=10,
    Rule_2_IF_IV1a_AND_IF_IV2a_AND_IF_IV3b_THEN_INT :=20,
    Rule_3_IF_IV1a_AND_IF_IV2a_AND_IF_IV3c_THEN_INT :=30,
    Rule_4_IF_IV1a_AND_IF_IV2b_AND_IF_IV3a_THEN_INT :=40,
    Rule_5_IF_IV1a_AND_IF_IV2b_AND_IF_IV3b_THEN_INT :=50,
    Rule_6_IF_IV1a_AND_IF_IV2b_AND_IF_IV3c_THEN_INT :=60,
    Rule_7_IF_IV1a_AND_IF_IV2c_AND_IF_IV3a_THEN_INT :=70,
    Rule_8_IF_IV1a_AND_IF_IV2c_AND_IF_IV3b_THEN_INT :=80,
    Rule_9_IF_IV1a_AND_IF_IV2c_AND_IF_IV3c_THEN_INT :=90,
```

```
    Rule_10_IF_IV1b_AND_IF_IV2a_AND_IF_IV3a_THEN_INT :=100,
    Rule_11_IF_IV1b_AND_IF_IV2a_AND_IF_IV3b_THEN_INT :=110,
    Rule_12_IF_IV1b_AND_IF_IV2a_AND_IF_IV3c_THEN_INT :=120,
    Rule_13_IF_IV1b_AND_IF_IV2b_AND_IF_IV3a_THEN_INT :=130,
    Rule_14_IF_IV1b_AND_IF_IV2b_AND_IF_IV3b_THEN_INT :=140,
    Rule_15_IF_IV1b_AND_IF_IV2b_AND_IF_IV3c_THEN_INT :=150,
    Rule_16_IF_IV1b_AND_IF_IV2c_AND_IF_IV3a_THEN_INT :=160,
    Rule_17_IF_IV1b_AND_IF_IV2c_AND_IF_IV3b_THEN_INT :=170,
    Rule_18_IF_IV1b_AND_IF_IV2c_AND_IF_IV3c_THEN_INT :=180,
    Rule_19_IF_IV1c_AND_IF_IV2a_AND_IF_IV3a_THEN_INT :=190,
    Rule_20_IF_IV1c_AND_IF_IV2a_AND_IF_IV3b_THEN_INT :=200,
    Rule_21_IF_IV1c_AND_IF_IV2a_AND_IF_IV3c_THEN_INT :=210,
    Rule_22_IF_IV1c_AND_IF_IV2b_AND_IF_IV3a_THEN_INT :=220,
    Rule_23_IF_IV1c_AND_IF_IV2b_AND_IF_IV3b_THEN_INT :=230,
    Rule_24_IF_IV1c_AND_IF_IV2b_AND_IF_IV3c_THEN_INT :=240,
    Rule_25_IF_IV1c_AND_IF_IV2c_AND_IF_IV3a_THEN_INT :=250,
    Rule_26_IF_IV1c_AND_IF_IV2c_AND_IF_IV3b_THEN_INT :=260,
    Rule_27_IF_IV1c_AND_IF_IV2c_AND_IF_IV3c_THEN_INT :=270,
    Output_variable_INT=>75,
    Parameterization_fault_BOOL=>0,
    First_active_rule_UINT=>1,
    Second_active_rule_UINT=>2,
    Third_active_rule_UINT=>4,
    Fourth_active_rule_UINT=>5,
    Fifth_active_rule_UINT=>10,
    Sixth_active_rule_UINT=>11,
    Seventh_active_rule_UINT=>13,
    Eighth_active_rule_UINT=>14,
    First_activity_level_10ths_percent_UINT=>60,
    Second_activity_level_10ths_percent_UINT=>60,
    Third_activity_level_10ths_percent_UINT=>240,
    Fourth_activity_level_10ths_percent_UINT=>240,
```

```
    Fifth_activity_level_10ths_percent_UINT=>40,
    Sixth_activity_level_10ths_percent_UINT=>40,
    Seventh_activity_level_10ths_percent_UINT=>160,
    Eighth_activity_level_10ths_percent_UINT=>160)
LD  Fuzzy_proportional_rate_adjustment.Output_variable_INT
INT_TO_UINT
ST  PID_CONTROLLER_1.Proportional_rate_percent_UINT

END_PROGRAM
```

Fuzzy logic system:
3 inputs, 5 terms

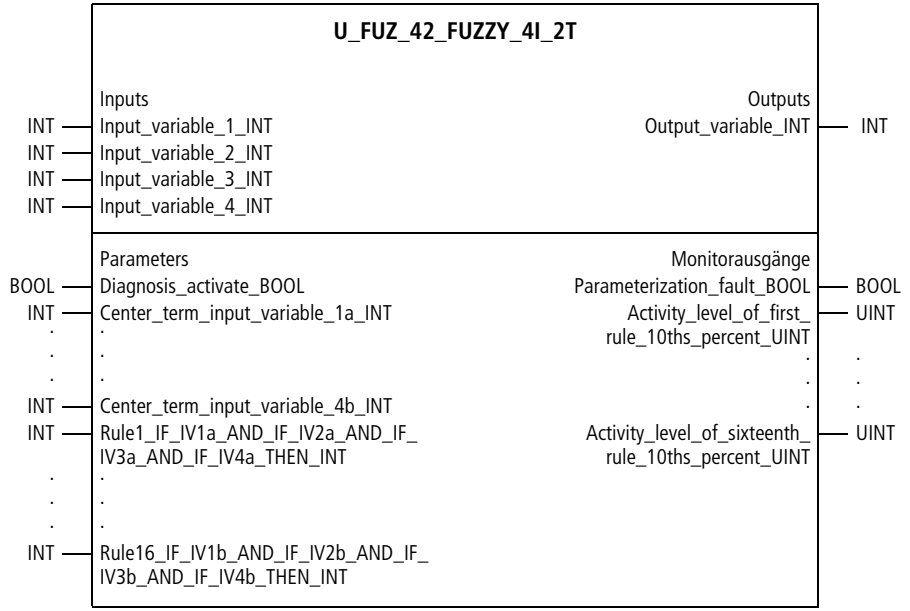
U_FUZ_35_FUZZY_3I_5T
Fuzzy Logic System with 3 Input Variable, 5 Terms per Input Variable and 125 Fuzzy Logic Rules

Description

See the function block "U_FUZ_33_FUZZY_3I_3T".
The only difference to the function block "U_FUZ_33_FUZZY_3I_3T" is that this function block has: 3 fuzzy input variables, 5 terms and 125 rules.

Fuzzy logic system:
4 inputs, 2 terms

U_FUZ_42_FUZZY_4I_2T
Fuzzy System with 4 Input Variables, 2 Terms per Input Variable and 16 Fuzzy Logic Rules



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Input_variable_1_INT	First linguistic input variable	–32768 to 32767
Input_variable_2_INT	Second linguistic input variable	–32768 to 32767
Input_variable_3_INT	Third linguistic input variable	–32768 to 32767
Input_variable_4_INT	Fourth linguistic input variable	–32768 to 32767

Designation	Significance	Value range
Parameters		
Diagnosis_activate_BOOL	Activates diagnostic mode	0/1
Center_term_input_variable_1a_INT	Input variable 1: term a	–32768 to 32767
Center_term_input_variable_1b_INT	Input variable 1: term b	–32768 to 32767
Center_term_input_variable_2a_INT	Input variable 2: term a	–32768 to 32767
Center_term_input_variable_2b_INT	Input variable 2: term b	–32768 to 32767
Center_term_input_variable_3a_INT	Input variable 3: term a	–32768 to 32767
Center_term_input_variable_3b_INT	Input variable 3: term b	–32768 to 32767
Center_term_input_variable_4a_INT	input variable 4: Term a	–32768 to 32767
Center_term_input_variable_4b_INT	input variable 4: Term b	–32768 to 32767
Rule1_IF_IV1a_AND_IF_IV2a_AND_ IF_IV3a_AND_IF_IV4a_THEN_INT	Fuzzy rule 1	–32768 to 32767
Rule2_IF_IV1a_AND_IF_IV2a_AND_ IF_IV3a_AND_IF_IV4b_THEN_INT	Fuzzy rule 2	–32768 to 32767
Rule3_IF_IV1a_AND_IF_IV2a_AND_ IF_IV3b_AND_IF_IV4a_THEN_INT	Fuzzy rule 3	–32768 to 32767
Rule4_IF_IV1a_AND_IF_IV2a_AND_ IF_IV3b_AND_IF_IV4b_THEN_INT	Fuzzy rule 4	–32768 to 32767
Rule5_IF_IV1a_AND_IF_IV2b_AND_ IF_IV3a_AND_IF_IV4a_THEN_INT	Fuzzy rule 5	–32768 to 32767
Rule6_IF_IV1a_AND_IF_IV2b_AND_ IF_IV3a_AND_IF_IV4b_THEN_INT	Fuzzy rule 6	–32768 to 32767
Rule7_IF_IV1a_AND_IF_IV2b_AND_ IF_IV3b_AND_IF_IV4a_THEN_INT	Fuzzy rule 7	–32768 to 32767
Rule8_IF_IV1a_AND_IF_IV2b_AND_ IF_IV3b_AND_IF_IV4b_THEN_INT	Fuzzy rule 8	–32768 to 32767
Rule9_IF_IV1b_AND_IF_IV2a_AND_ IF_IV3a_AND_IF_IV4a_THEN_INT	Fuzzy rule 9	–32768 to 32767
Rule10_IF_IV1b_AND_IF_IV2a_AND_ IF_IV3a_AND_IF_IV4b_THEN_INT	Fuzzy rule 10	–32768 to 32767

Designation	Significance	Value range
Rule11_IF_IV1b_AND_IF_IV2a_AND_ IF_IV3b_AND_IF_IV4a_THEN_INT	Fuzzy rule 11	–32768 to 32767
Rule12_IF_IV1b_AND_IF_IV2a_AND_ IF_IV3b_AND_IF_IV4b_THEN_INT	Fuzzy rule 12	–32768 to 32767
Rule13_IF_IV1b_AND_IF_IV2b_AND_ IF_IV3a_AND_IF_IV4a_THEN_INT	Fuzzy rule 13	–32768 to 32767
Rule14_IF_IV1b_AND_IF_IV2b_AND_ IF_IV3a_AND_IF_IV4b_THEN_INT	Fuzzy rule 14	–32768 to 32767
Rule15_IF_IV1b_AND_IF_IV2b_AND_ IF_IV3b_AND_IF_IV4a_THEN_INT	Fuzzy rule 15	–32768 to 32767
Rule16_IF_IV1b_AND_IF_IV2b_AND_ IF_IV3b_AND_IF_IV4b_DANN_INT	Fuzzy rule 16	–32768 to 32767
Outputs		
Output_variable_INT	Linguistic output variable	–32768 to 32767
Monitor outputs		
Parameterization_fault_BOOL	Status: parameterization fault	0/1
Activity_level_of_first_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 1	0 to 1000
Activity_level_of_second_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 2	0 to 1000
Activity_level_of_third_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 3	0 to 1000
Activity_level_of_fourth_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 4	0 to 1000
Activity_level_of_fifth_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 5	0 to 1000
Activity_level_of_sixth_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 6	0 to 1000
Activity_level_of_seventh_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 7	0 to 1000
Activity_level_of_eighth_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 8	0 to 1000

Designation	Significance	Value range
Activity_level_of_ninth_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 9	0 to 1000
Activity_level_of_tenth_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 10	0 to 1000
Activity_level_of_eleventh_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 11	0 to 1000
Activity_level_of_twelfth_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 12	0 to 1000
Activity_level_of_thirteenth_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 13	0 to 1000
Activity_level_of_fourteenth_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 14	0 to 1000
Activity_level_of_fifteenth_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 15	0 to 1000
Activity_level_of_sixteenth_rule_ 10ths_percent_UINT	Activity level of fuzzy rule 16	0 to 1000

Description

For each of the linguistic input variables "Input_variable_1_INT", "Input_variable_2_INT", "Input_variable_3_INT" and "Input_variable_4_INT", two terms can be assigned values, with the variables "Center_term_input_variable_. . ._INT". This results in the arrangement of terms shown in figure 62 (→ above). The values assigned to the terms can be in any unit, such as percentage, tenths of a percent, °C, 12-bit, etc. Each possible combination of the input variables for these terms is assigned to an output variable (= THEN assignment of the fuzzy logic rule = characteristic curve term), resulting in 16 fuzzy logic rules.

Example:

- Fuzzy rule 1:
  With the variable "Rule_1_IF_IV1a_AND_IF_IV2a_ AND_IF_IV3a_AND_IF_IV4a_THEN_INT", an output variable is assigned to the combination ["term a, input variable 1" | "term a, input variable 2" | "term a, input variable 3" | "term a, input variable 4"]. The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1a" AND
    IF input variable 2 is "2a" AND
    IF input variable 3 is "3a" AND
    IF input variable 4 is "4a"
    THEN the output variable has the value "X1"
    (any integer value can be assigned).

- Fuzzy rule 2:
  With the variable "Rule_2_IF_IV1a_AND_IF_IV2a_ AND_IF_IV3b_AND_IF_IV4b_THEN_INT", an output variable is assigned to the combination ["term a, input variable 1" | "term a, input variable 2" | "term a, input variable 3" | "term b, input variable 4"]. The specifically fuzzy statement of this input is:
  – IF input variable 1 is "1a" hat AND
    IF input variable 2 is "2a" AND
    IF input variable 3 is "3a" hat AND
    IF input variable 4 is "4b"
    THEN the output variable has the value "X2"
    (any integer value can be assigned).

In the output section of the function block, an output variable is calculated as a function of the input variables. This output variable has the same unit as the THEN assignments of the fuzzy logic rules. Standardisation or rescaling of parameter values is not required for the fuzzy logic function blocks.

Enter the terms of the input variables in ascending order. If this is not done, the status display "Parameterization_fault_BOOL" will be "1". The fuzzy logic function block generates a five-dimensional characteristic field which cannot be shown in graphical form. If two input variables are kept constant, then a three-dimensional characteristic field can be drawn as a function of the other two (as for the fuzzy logic function blocks with two input variables).

"Diagnosis_activate_BOOL=1" displays the activity levels of the fuzzy logic rules in the output section (the function block's cycle time requirement is doubled when the diagnosis is enabled). Detailed descriptions of fuzzy logic characteristic fields and the activity levels of fuzzy logic rules can be found at the beginning of this chapter.

Example:
Assume practical experience about the Kp setting of a PID-controller as a function of four disturbance variables, as follows:

- input variable 1 = disturbance variable 1 [‰]
  (values greater than 1000 are possible,
  e.g. 3524 ‰ = 3.524)
- input variable 2 = disturbance variable 2 [‰]
- input variable 3 = disturbance variable 3 [‰]
- input variable 4 = disturbance variable 4 [‰]
- output variable = Kp [%]

disturbance variable 1, small = 1000
disturbance variable 1, large = 2000
disturbance variable 2, small = 1500
disturbance variable 2, large = 2500
disturbance variable 3, small = 2000
disturbance variable 3, large = 3000
disturbance variable 4, small = 1000
disturbance variable 4, large = 3000

Fuzzy rule	Disturbance variable 1 [‰]	Disturbance variable 2 [‰]	Disturbance variable 3 [‰]	Disturbance variable 4 [‰]	Output variable (Kp) [%]
1	1000	1500	2000	1000	10
2	1000	1500	2000	3000	20
3	1000	1500	3000	1000	30
4	1000	1500	3000	3000	40
5	1000	2500	2000	1000	50
6	1000	2500	2000	3000	60
7	1000	2500	3000	1000	70
8	1000	2500	3000	3000	80
9	2000	1500	2000	1000	90
10	2000	1500	2000	3000	100
11	2000	1500	3000	1000	110
12	2000	1500	3000	3000	120
13	2000	2500	2000	1000	130
14	2000	2500	2000	3000	140
15	2000	2500	3000	1000	150
16	2000	2500	3000	3000	160

The input variables "Disturbance variable 1 = 1500", "Disturbance variable 2 = 2000", "Disturbance variable 3 = 2500" and "Disturbance variable 4 = 2000" result in the output variable "Kp = 85". All fuzzy logic rules have the same level of activity, because the input variables are placed exactly in the middle of the related terms. 62 is displayed for each level of activity (→ the output section of the function block).

Application of the function block "U_FUZ_42_FUZZY_4I_2T" in the program "Fuz42_P"

```
PROGRAM Fuz42_P
VAR
    Fuzzy_proportional_rate_adjustment : U_FUZ_42_FUZZY_4I_2T ;
    PID_CONTROLLER_1 : U_PID_CONTROLLER ;
    Disturbance1_10ths_percent : INT :=1500 ;
    Disturbance2_10ths_percent : INT :=2000 ;
    Disturbance3_10ths_percent : INT :=2500 ;
    Disturbance4_10ths_percent : INT :=2000 ;
END_VAR

CAL Fuzzy_proportional_rate_adjustment(
    Input_variable_1_INT :=disturbance1_10ths_percent,
    Input_variable_2_INT :=disturbance2_10ths_percent,
    Input_variable_3_INT :=disturbance3_10ths_percent,
    Input_variable_4_INT :=disturbance4_10ths_percent,
    Diagnosis_activate_BOOL :=1,
    Center_term_input_variable_1a_INT :=1000,
    Center_term_input_variable_1b_INT :=2000,
    Center_term_input_variable_2a_INT :=1500,
    Center_term_input_variable_2b_INT :=2500,
    Center_term_input_variable_3a_INT :=2000,
    Center_term_input_variable_3b_INT :=3000,
    Center_term_input_variable_4a_INT :=1000,
    Center_term_input_variable_4b_INT :=3000,
    Rule1_IF_IV1a_AND_IF_IV2a_AND_IF_IV3a_AND_IF_IV4a_THEN_INT :=10,
    Rule2_IF_IV1a_AND_IF_IV2a_AND_IF_IV3a_AND_IF_IV4b_THEN_INT :=20,
    Rule3_IF_IV1a_AND_IF_IV2a_AND_IF_IV3b_AND_IF_IV4a_THEN_INT :=30,
    Rule4_IF_IV1a_AND_IF_IV2a_AND_IF_IV3b_AND_IF_IV4b_THEN_INT :=40,
    Rule5_IF_IV1a_AND_IF_IV2b_AND_IF_IV3a_AND_IF_IV4a_THEN_INT :=50,
    Rule6_IF_IV1a_AND_IF_IV2b_AND_IF_IV3a_AND_IF_IV4b_THEN_INT :=60,
    Rule7_IF_IV1a_AND_IF_IV2b_AND_IF_IV3b_AND_IF_IV4a_THEN_INT :=70,
    Rule8_IF_IV1a_AND_IF_IV2b_AND_IF_IV3b_AND_IF_IV4b_THEN_INT :=80,
```

```
    Rule9_IF_IV1b_AND_IF_IV2a_AND_IF_IV3a_AND_IF_IV4a_THEN_INT :=90,
    Rule10_IF_IV1b_AND_IF_IV2a_AND_IF_IV3a_AND_IF_IV4b_THEN_INT :=100,
    Rule11_IF_IV1b_AND_IF_IV2a_AND_IF_IV3b_AND_IF_IV4a_THEN_INT :=110,
    Rule12_IF_IV1b_AND_IF_IV2a_AND_IF_IV3b_AND_IF_IV4b_THEN_INT :=120,
    Rule13_IF_IV1b_AND_IF_IV2b_AND_IF_IV3a_AND_IF_IV4a_THEN_INT :=130,
    Rule14_IF_IV1b_AND_IF_IV2b_AND_IF_IV3a_AND_IF_IV4b_THEN_INT :=140,
    Rule15_IF_IV1b_AND_IF_IV2b_AND_IF_IV3b_AND_IF_IV4a_THEN_INT :=150,
    Rule16_IF_IV1b_AND_IF_IV2b_AND_IF_IV3b_AND_IF_IV4b_DANN_INT :=160,
    Output_variable_INT=>85,
    Parameterization_fault_BOOL=>0,
    Activity_level_of_first_rule_10ths_percent_UINT=>62,
    Activity_level_of_second_rule_10ths_percent_UINT=>62,
    Activity_level_of_third_rule_10ths_percent_UINT=>62,
    Activity_level_of_fourth_rule_10ths_percent_UINT=>62,
    Activity_level_of_fifth_rule_10ths_percent_UINT=>62,
    Activity_level_of_sixth_rule_10ths_percent_UINT=>62,
    Activity_level_of_seventh_rule_10ths_percent_UINT=>62,
    Activity_level_of_eighth_rule_10ths_percent_UINT=>62,
    Activity_level_of_ninth_rule_10ths_percent_UINT=>62,
    Activity_level_of_tenth_rule_10ths_percent_UINT=>62,
    Activity_level_of_eleventh_rule_10ths_percent_UINT=>62,
    Activity_level_of_twelfth_rule_10ths_percent_UINT=>62,
    Activity_level_of_thirteenth_rule_10ths_percent_UINT=>62,
    Activity_level_of_fourteenth_rule_10ths_percent_UINT=>62,
    Activity_level_of_fifteenth_rule_10ths_percent_UINT=>62,
    Activity_level_of_sixteenth_rule_10ths_percent_UINT=>62
)

LD  Fuzzy_proportional_rate_adjustment.Output_variable_INT
INT_TO_UINT
ST  PID_CONTROLLER_1.Proportional_rate_percent_UINT

END_PROGRAM
```

Fuzzy logic system:
4 inputs, 3 terms

U_FUZ_43_FUZZY_4I_3T
Fuzzy Logic System with 4 Input Variable, 3 Terms per Input Variable and 81 Fuzzy Logic Rules

Description

See the function block "U_FUZ_42_FUZZY_4I_2T".
The only difference to the function block "U_FUZ_42_FUZZY_4I_2T" is that this function block has: 4 fuzzy input variables, 3 terms and 81 rules.

Fuzzy logic system:
5 inputs, 2 terms

U_FUZ_52_FUZZY_5I_2T
Fuzzy Logic System with 5 Input Variable, 2 Terms per Input Variable and 32 Fuzzy Logic Rules

Description

See the function block "U_FUZ_42_FUZZY_4I_2T".
The only difference to the function block "U_FUZ_42_FUZZY_4I_2T" is that this function block has: 5 fuzzy input variables, 2 terms and 32 rules.

Fuzzy logic system:
5 inputs, 3 terms

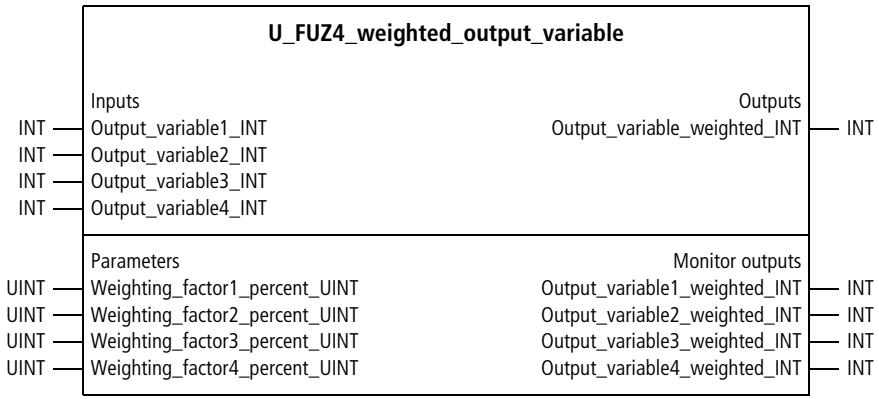
U_FUZ_53_FUZZY_5I_3T
Fuzzy Logic System with 5 Input Variable, 3 Terms per Input Variable and 243 Fuzzy Logic Rules

Description

See the function block "U_FUZ_42_FUZZY_4I_2T".
The only difference to the function block "U_FUZ_42_FUZZY_4I_2T" is that this function block has: 5 fuzzy input variables, 3 terms and 243 rules.

Fuzzy basic function block

U_FUZ4_weighted_output_variable
Function block for weighting 4 manipulated variables (fuzzy output variables)



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Output_variable1_INT	First manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable2_INT	Second manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable3_INT	Third manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable4_INT	Fourth manipulated variable (fuzzy output variable)	–32768 to 32767
Parameters		
Weighting_factor1_percent_UINT	First weighting factor	0 to 65535
Weighting_factor2_percent_UINT	Second weighting factor	0 to 65535
Weighting_factor3_percent_UINT	Third weighting factor	0 to 65535
Weighting_factor4_percent_UINT	Fourth weighting factor	0 to 65535

Designation	Significance	Value range
Outputs		
Output_variable_weighted_INT	Weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Monitor outputs		
Output_variable1_weighted_INT	First weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable2_weighted_INT	Second weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable3_weighted_INT	Third weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable4_weighted_INT	Fourth weighted manipulated variable (fuzzy output variable)	–32768 to 32767

Description

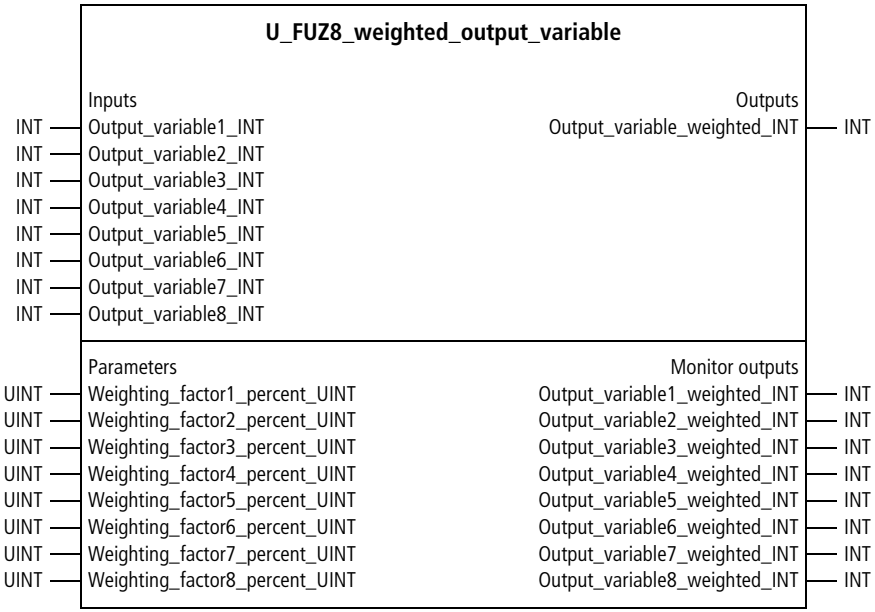
This function block is used for weighting four manipulated variables (fuzzy output variables). If less manipulated variables are required than the maximum number, the unused inputs and parameters must be assigned with "0".

Application of the function block "U_FUZ4_weighted_output_variable" in the program "output3"

```
PROGRAM output3
VAR
    FUZ4_weighted_output_variable : U_FUZ4_weighted_output_variable ;
    Fuzzy_output_variable_1 : INT ;
    Fuzzy_output_variable_2 : INT ;
    Fuzzy_output_variable_3 : INT ;
    Fuzzy_output_variables_1_to_3_weighted : INT ;
END_VAR
```

```
LD  100
ST  Fuzzy_output_variable_1
LD  60
ST  Fuzzy_output_variable_2
LD  20
ST  Fuzzy_output_variable_3

CAL FUZ4_weighted_output_variable(
    Output_variable1_INT :=Fuzzy_output_variable_1,
    Weighting_factor1_percent_UINT :=120,
    Output_variable2_INT :=Fuzzy_output_variable_2,
    Weighting_factor2_percent_UINT :=50,
    Output_variable3_INT :=Fuzzy_output_variable_3,
    Weighting_factor3_percent_UINT :=30,
    Output_variable4_INT :=0,
    Weighting_factor4_percent_UINT :=0,
    Output_variable_weighted_INT=>78,
    Output_variable1_weighted_INT=>60,
    Output_variable2_weighted_INT=>15,
    Output_variable3_weighted_INT=>3,
    Output_variable4_weighted_INT=>0
    )
LD  FUZ4_weighted_output_variable.Output_variable_weighted_INT
ST  Fuzzy_output_variables_1_to_3_weighted

END_PROGRAM
```

U_FUZ8_weighted_output_variable
Function block for weigthing 8 manipulated variables (fuzzy ouput variable)



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Output_variable1_INT	First manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable2_INT	Second manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable3_INT	Third manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable4_INT	Fourth manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable5_INT	Fifth manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable6_INT	Sixth manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable7_INT	Seventh manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable8_INT	Eighth manipulated variable (fuzzy output variable)	–32768 to 32767
Parameters		
Weighting_factor1_percent_UINT	First weighting factor	0 to 65535
Weighting_factor2_percent_UINT	Second weighting factor	0 to 65535
Weighting_factor3_percent_UINT	Third weighting factor	0 to 65535
Weighting_factor4_percent_UINT	Fourth weighting factor	0 to 65535
Weighting_factor5_percent_UINT	Fifth weighting factor	0 to 65535
Weighting_factor6_percent_UINT	Sixth weighting factor	0 to 65535
Weighting_factor7_percent_UINT	Seventh weighting factor	0 to 65535
Weighting_factor8_percent_UINT	Eighth weighting factor	0 to 65535
Outputs		
Output_variable_weighted_INT	Weighted manipulated variable (fuzzy output variable)	–32768 to 32767

Designation	Significance	Value range
Monitor outputs		
Output_variable1_weighted_INT	First weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable2_weighted_INT	Second weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable3_weighted_INT	Third weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable4_weighted_INT	Fourth weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable5_weighted_INT	Fifth weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable6_weighted_INT	Sixth weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable7_weighted_INT	Seventh weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable8_weighted_INT	Eighth weighted manipulated variable (fuzzy output variable)	–32768 to 32767

Description

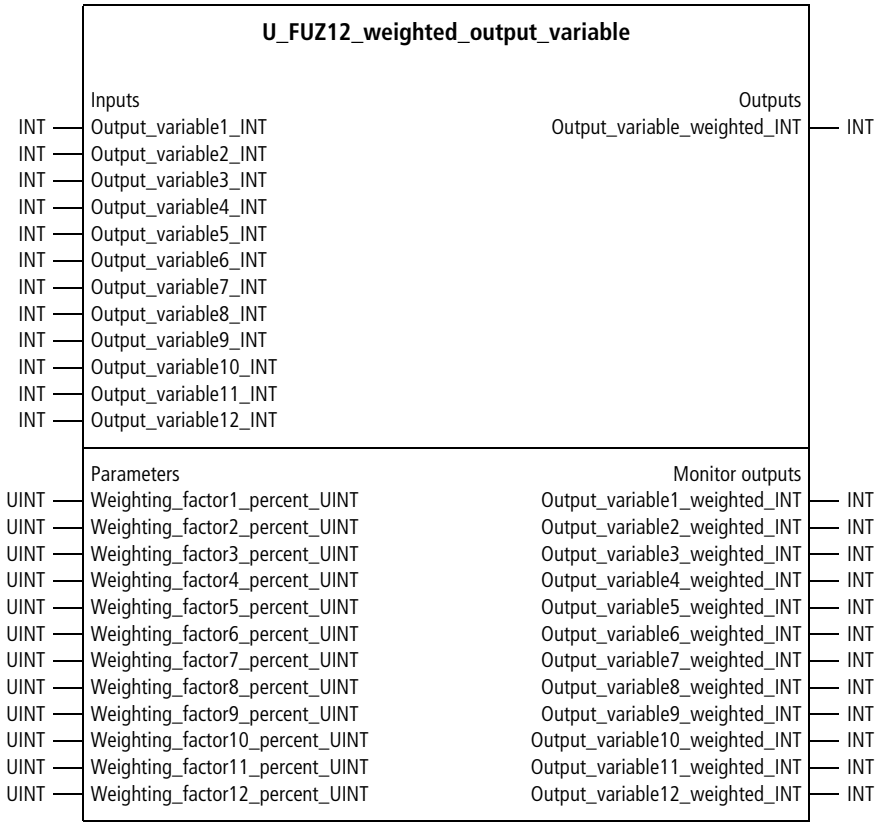
This function block is used for weighting eight manipulated variables (fuzzy output variables). If less manipulated variables are required than the maximum number, the unused inputs and parameters must be assigned with "0".

Application of the function block "U_FUZ8_weighted_output_variable" in the program "output7"

```
PROGRAM output7

VAR
    FUZ8_weighted_output_variable : U_FUZ8_weighted_output_variable ;
    Fuzzy_output_variable_1 : INT ;
    Fuzzy_output_variable_2 : INT ;
    Fuzzy_output_variable_3 : INT ;
    Fuzzy_output_variable_4 : INT ;
    Fuzzy_output_variable_5 : INT ;
    Fuzzy_output_variable_6 : INT ;
    Fuzzy_output_variable_7 : INT ;
    Fuzzy_output_variables_1_to_7_weighted : INT ;
END_VAR

LD  100
ST  Fuzzy_output_variable_1
LD  60
ST  Fuzzy_output_variable_2
LD  20
ST  Fuzzy_output_variable_3
LD  200
ST  Fuzzy_output_variable_4
LD  120
ST  Fuzzy_output_variable_5
LD  300
ST  Fuzzy_output_variable_6
LD  20
ST  Fuzzy_output_variable_7
```

```
CAL FUZ8_weighted_output_variable(
    Output_variable1_INT :=Fuzzy_output_variable_1,
    Weighting_factor1_percent_UINT :=120,
    Output_variable2_INT :=Fuzzy_output_variable_2,
    Weighting_factor2_percent_UINT :=50,
    Output_variable3_INT :=Fuzzy_output_variable_3,
    Weighting_factor3_percent_UINT :=30,
    Output_variable4_INT :=Fuzzy_output_variable_4,
    Weighting_factor4_percent_UINT :=150,
    Output_variable5_INT :=Fuzzy_output_variable_5,
    Weighting_factor5_percent_UINT :=10,
    Output_variable6_INT :=Fuzzy_output_variable_6,
    Weighting_factor6_percent_UINT :=40,
    Output_variable7_INT :=Fuzzy_output_variable_7,
    Weighting_factor7_percent_UINT :=80,
    Output_variable8_INT :=0,
    Weighting_factor8_percent_UINT :=0,
    Output_variable_weighted_INT=>125,
    Output_variable1_weighted_INT=>24,
    Output_variable2_weighted_INT=>6,
    Output_variable3_weighted_INT=>1,
    Output_variable4_weighted_INT=>62,
    Output_variable5_weighted_INT=>2,
    Output_variable6_weighted_INT=>25,
    Output_variable7_weighted_INT=>3,
    Output_variable8_weighted_INT=>0
    )

LD  FUZ8_weighted_output_variable.Output_variable_weighted_INT
ST  Fuzzy_output_variables_1_to_7_weighted

END_PROGRAM
```

U_FUZ12_weighted_output_variable
Function block for weighting 12 manipulated variables (fuzzy output variable)



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
Output_variable1_INT	First manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable2_INT	Second manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable3_INT	Third manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable4_INT	Fourth manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable5_INT	Fifth manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable6_INT	Sixth manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable7_INT	Seventh manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable8_INT	Eighth manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable9_INT	Ninth manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable10_INT	Tenth manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable11_INT	Eleventh manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable12_INT	Twelfth manipulated variable (fuzzy output variable)	–32768 to 32767
Parameters		
Weighting_factor1_percent_UINT	First weighting factor	0 to 65535
Weighting_factor2_percent_UINT	Second weighting factor	0 to 65535
Weighting_factor3_percent_UINT	Third weighting factor	0 to 65535
Weighting_factor4_percent_UINT	Fourth weighting factor	0 to 65535
Weighting_factor5_percent_UINT	Fifth weighting factor	0 to 65535
Weighting_factor6_percent_UINT	Sixth weighting factor	0 to 65535

Designation	Significance	Value range
Weighting_factor7_percent_UINT	Seventh weighting factor	0 to 65535
Weighting_factor8_percent_UINT	Eighth weighting factor	0 to 65535
Weighting_factor9_percent_UINT	Ninth weighting factor	0 to 65535
Weighting_factor10_percent_UINT	tenth weighting factor	0 to 65535
Weighting_factor11_percent_UINT	Eleventh weighting factor	0 to 65535
Weighting_factor12_percent_UINT	Twelfth weighting factor	0 to 65535
Outputs		
Output_variable_weighted_INT	Weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Monitor outputs		
Output_variable1_weighted_INT	First weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable2_weighted_INT	Second weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable3_weighted_INT	Third weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable4_weighted_INT	Fourth weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable5_weighted_INT	Fifth weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable6_weighted_INT	Sixth weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable7_weighted_INT	Seventh weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable8_weighted_INT	Eighth weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable9_weighted_INT	Ninth weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable10_weighted_INT	Tenth weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable11_weighted_INT	Eleventh weighted manipulated variable (fuzzy output variable)	–32768 to 32767
Output_variable12_weighted_INT	Twelfth weighted manipulated variable (fuzzy output variable)	–32768 to 32767

Description

This function block is used for weighting 12 manipulated variables (fuzzy output variables). If less manipulated variables are required than the maximum number, the unused inputs and parameters must be assigned with "0".

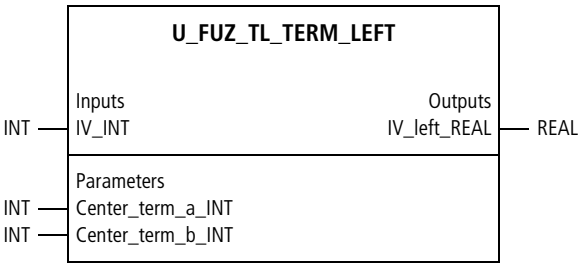
Application of the function block "U_FUZ12_weighted_output_variable" in the program "output10"

```
PROGRAM output10


VAR
    FUZ12_weighted_output_variable : U_FUZ12_weighted_output_variable ;
    Fuzzy_output_variable_1 : INT ;
    Fuzzy_output_variable_2 : INT ;
    Fuzzy_output_variable_3 : INT ;
    Fuzzy_output_variable_4 : INT ;
    Fuzzy_output_variable_5 : INT ;
    Fuzzy_output_variable_6 : INT ;
    Fuzzy_output_variable_7 : INT ;
    Fuzzy_output_variable_8 : INT ;
    Fuzzy_output_variable_9 : INT ;
    Fuzzy_output_variable_10 : INT ;
    Fuzzy_output_variables_1_to_10_weighted : INT ;
END_VAR
```

```
LD  100
ST  Fuzzy_output_variable_1
LD  60
ST  Fuzzy_output_variable_2
LD  20
ST  Fuzzy_output_variable_3
LD  200
ST  Fuzzy_output_variable_4
LD  120
ST  Fuzzy_output_variable_5
LD  300
ST  Fuzzy_output_variable_6
LD  20
ST  Fuzzy_output_variable_7
LD  40
ST  Fuzzy_output_variable_8
LD  80
ST  Fuzzy_output_variable_9
LD  160
ST  Fuzzy_output_variable_10

CAL FUZ12_weighted_output_variable(
    Output_variable1_INT :=Fuzzy_output_variable_1,
    Weighting_factor1_percent_UINT :=120,
    Output_variable2_INT :=Fuzzy_output_variable_2,
    Weighting_factor2_percent_UINT :=50,
    Output_variable3_INT :=Fuzzy_output_variable_3,
    Weighting_factor3_percent_UINT :=30,
    Output_variable4_INT :=Fuzzy_output_variable_4,
    Weighting_factor4_percent_UINT :=150,
    Output_variable5_INT :=Fuzzy_output_variable_5,
    Weighting_factor5_percent_UINT :=10,
    Output_variable6_INT :=Fuzzy_output_variable_6,
    Weighting_factor6_percent_UINT :=40,
```

```
    Output_variable7_INT :=Fuzzy_output_variable_7,
    Weighting_factor7_percent_UINT :=80,
    Output_variable8_INT :=Fuzzy_output_variable_8,
    Weighting_factor8_percent_UINT :=80,
    Output_variable9_INT :=Fuzzy_output_variable_9,
    Weighting_factor9_percent_UINT :=80,
    Output_variable10_INT :=Fuzzy_output_variable_10,
    Weighting_factor10_percent_UINT :=80,
    Output_variable11_INT :=0,
    Weighting_factor11_percent_UINT :=0,
    Output_variable12_INT :=0,
    Weighting_factor12_percent_UINT :=0,
    Output_variable_weighted_INT=>115,
    Output_variable1_weighted_INT=>16,
    Output_variable2_weighted_INT=>4,
    Output_variable3_weighted_INT=>0,
    Output_variable4_weighted_INT=>41,
    Output_variable5_weighted_INT=>1,
    Output_variable6_weighted_INT=>16,
    Output_variable7_weighted_INT=>2,
    Output_variable8_weighted_INT=>4,
    Output_variable9_weighted_INT=>8,
    Output_variable10_weighted_INT=>17,
    Output_variable11_weighted_INT=>0,
    Output_variable12_weighted_INT=>0
    )

LD  FUZ12_weighted_output_variable.Output_variable_weighted_INT
ST  Fuzzy_output_variables_1_to_10_weighted

END_PROGRAM
```

U_FUZ_TL_TERM_LEFT
Fuzzy Term (Membership Function (ZF)) at the Left



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
IV_INT	Linguistic input variable	–32768 to 32767
Parameters		
Center_term_a_INT	Left term of the membership function	–32768 to 32767
Center_term_b_INT	Right term of the membership function	–32768 to 32767
Outputs		
IV_left_REAL	Truth value	Floating-point number with an exponent ±38

Description

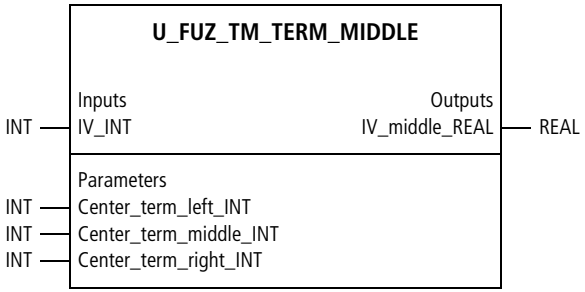
This function block is a subblock for the fuzzy logic function blocks with three and four inputs. The parameters "Center_term_a_INT" and "center_term_b_INT" determine the membership function (term) at the left edge (→ fig. 63). Based on the input variable "IV_INT", a truth value is calculated and output with the variable "IV_left_REAL".

Example:
The function block of the application example outputs a truth value of 0.428571 as a function of the parameter values and the input value.

Application of the function block "U_FUZ_TL_TERM_LEFT" in the program "TermLeft"

```
PROGRAM TermLeft
VAR
  Term_left : U_FUZ_TL_TERM_LEFT ;
  Value_term_left : REAL ;
END_VAR

CAL Term_left(
  Center_term_a_INT :=100,
  Center_term_b_INT :=800,
  IV_INT :=500,
  IV_left_REAL=>Value_term_left)

END_PROGRAM
```

U_FUZ_TM_TERM_MIDDLE
Fuzzy Logic Term (Membership Function (ZF)) in the Middle



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
IV_INT	Linguistic input variable	–32768 to 32767
Parameters		
Center_term_left_INT	Left term of the membership function	–32768 to 32767
Center_term_middle_INT	Middle term of the membership function	–32768 to 32767
Center_term_right_INT	Right term of the membership function	–32768 to 32767
Outputs		
IV_middle_REAL	Truth value	Floating-point number with exponent ±38

Description

This function block is a subblock for the fuzzy logic function blocks with three and four inputs. The parameters "Center_term_left_INT", "Center_term_middle_INT" and "Center_term_right_INT" determine the membership function (term) in the middle (→ fig. 63 and 64).

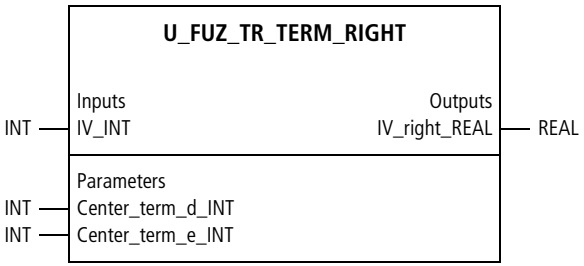
In figure 63, the middle membership function is determined by the parameters "a", "b" and "c". Based on the input variable "IV_INT", a truth value is calculated and output with the variable "IV_middle_REAL".

Example:
The function block of the application example outputs a truth value of 0.6 as a function of the parameter values and the input value.

Application of the function block "U_FUZ_TM_TERM_MIDDLE" in the program "TermMidd"

```
PROGRAM TermMidd
VAR
  Term_middle : U_FUZ_TM_TERM_MIDDLE ;
  a : INT :=100 ;
  b : INT :=300 ;
  c : INT :=800 ;
  Value_term_middle : REAL ;
END_VAR

CAL Term_middle(
  Center_term_left_INT :=a,
  Center_term_middle_INT :=b,
  Center_term_right_INT :=c,
  IV_INT :=500,
  IV_middle_REAL=>Value_term_middle)

END_PROGRAM
```

U_FUZ_TR_TERM_RIGHT
Fuzzy Term (Membership Function (ZF)) at the Right



Function block prototype

Meaning of the operands

Designation	Significance	Value range
Inputs		
IV_INT	Linguistic input variable	–32768 to 32767
Parameters		
Center_term_d_INT	Left term of the membership function	–32768 to 32767
Center_term_e_INT	Right term of the membership function	–32768 to 32767
Outputs		
IV_left_REAL	Truth value	Floating-point number with exponent ±38

Description

This function block is a subblock for the fuzzy logic function blocks with three and four inputs. The parameters "Center_term_d_INT" and "Center_term_e_INT" determine the membership function (term) at the right (→ fig. 64). Based on the input variable "IV_INT", a truth value is calculated and output with the variable "IV_right_REAL".

Example:
The function block of the application example outputs a truth value of 0.8 as a function of the parameter values and the input value.

Application of the function block "U_FUZ_TL_TERM_LEFT" in the program "TermRigh"

```
PROGRAM TermRigh
VAR
  Term_right : U_FUZ_TR_TERM_RIGHT ;
  d : INT :=300 ;
  e : INT :=800 ;
  Value_term_right : REAL ;
END_VAR

CAL Term_right(
  Center_term_d_INT :=d,
  Center_term_e_INT :=e,
  IV_INT :=700,
  IV_right_REAL=>Value_term_right)

END_PROGRAM
```

Index