

STL (S7-1500)

Manual




20.00.00.00

11/2024

Legal information

Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

| |
|--|
|  DANGER |
| indicates that death or severe personal injury will result if proper precautions are not taken. |
|  WARNING |
| indicates that death or severe personal injury may result if proper precautions are not taken. |
|  CAUTION |
| indicates that minor personal injury can result if proper precautions are not taken. |
| NOTICE |
| indicates that property damage can result if proper precautions are not taken. |


If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

Proper use of Siemens products

Note the following:

| |
|--|
|  WARNING |
| Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient |

conditions must be complied with. The information in the relevant documentation must be observed.

Trademarks

All names identified by ® are registered trademarks of Siemens Aktiengesellschaft. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

Siemens Aktiengesellschaft
Digital Industries
Postfach 48 48
90026 NÜRNBERG
GERMANY

11/2024 Subject to change

Copyright © Siemens 2024.
All rights reserved

Table of content

| | | |
|---------|--|-----------|
| 1 | STL (S7-1500) | 10 |
| 1.1 | Bit logic operations (S7-1500) | 10 |
| 1.1.1 | R_TRIG: Detect positive signal edge (S7-1500) | 10 |
| 1.1.2 | F_TRIG: Detect negative signal edge (S7-1500) | 11 |
| 1.2 | Timer operations (S7-1500) | 12 |
| 1.2.1 | TP: Generate pulse (S7-1500) | 12 |
| 1.2.2 | TON: Generate on-delay (S7-1500) | 16 |
| 1.2.3 | TOF: Generate off-delay (S7-1500) | 20 |
| 1.2.4 | TONR: Time accumulator (S7-1500) | 24 |
| 1.2.5 | RESET_TIMER: Reset timer (S7-1500) | 28 |
| 1.2.6 | PRESET_TIMER: Load time duration (S7-1500) | 31 |
| 1.3 | Counter operations (S7-1500) | 34 |
| 1.3.1 | CTU: Count up (S7-1500) | 34 |
| 1.3.2 | CTD: Count down (S7-1500) | 37 |
| 1.3.3 | CTUD: Count up and down (S7-1500) | 40 |
| 1.4 | Comparator operations (S7-1500) | 44 |
| 1.4.1 | CompType: Compare tag structured data types (S7-1500) | 44 |
| 1.4.2 | VARIANT (S7-1500) | 49 |
| 1.4.2.1 | EQ_Type: Compare data type for EQUAL with the data type of a tag (S7-1500) | 49 |
| 1.4.2.2 | NE_Type: Compare data type for UNEQUAL with the data type of a tag (S7-1500) | 51 |
| 1.4.2.3 | EQ_ElemType: Compare data type of an ARRAY element for EQUAL with the data type of a tag (S7-1500) | 53 |
| 1.4.2.4 | NE_ElemType: Compare data type of an ARRAY element for UNEQUAL with the data type of a tag (S7-1500) | 54 |
| 1.4.2.5 | IS_NULL: Check for EQUALS NULL pointer (S7-1500) | 56 |
| 1.4.2.6 | NOT_NULL: Check for UNEQUALS NULL pointer (S7-1500) | 57 |
| 1.4.2.7 | IS_ARRAY: Check for ARRAY (S7-1500) | 58 |
| 1.4.2.8 | EQ_TypeOfDB: Compare data type of an indirectly addressed DB for EQUAL with a data type (S7-1500) | 59 |
| 1.4.2.9 | NE_TypeOfDB: Compare data type of an indirectly addressed DB for UNEQUAL with a data type (S7-1500) | 61 |
| 1.5 | Math functions (S7-1500) | 62 |
| 1.5.1 | MIN: Get minimum (S7-1500) | 62 |
| 1.5.2 | MAX: Get maximum (S7-1500) | 64 |
| 1.5.3 | LIMIT: Set limit value (S7-1500) | 65 |
| 1.6 | Move operations (S7-1500) | 0 |
| 1.6.1 | MOVE: Move value (S7-1500) | 0 |
| 1.6.2 | Deserialize: Deserialize (S7-1500) | 0 |

| | | |
|----------|---|---|
| 1.6.3 | Serialize: Serialize (S7-1500) | 0 |
| 1.6.4 | MOVE_BLK: Move block (S7-1500) | 0 |
| 1.6.5 | MOVE_BLK_VARIANT: Move block (S7-1500) | 0 |
| 1.6.6 | UMOVE_BLK: Move block uninterruptible (S7-1500) | 0 |
| 1.6.7 | FILL_BLK: Fill block (S7-1500) | 0 |
| 1.6.8 | UFILL_BLK: Fill block uninterruptible (S7-1500) | 0 |
| 1.6.9 | SCATTER: Parse the bit sequence into individual bits (S7-1500) | 0 |
| 1.6.10 | SCATTER_BLK: Parse elements of an ARRAY of bit sequence into individual bits (S7-1500) | 0 |
| 1.6.11 | GATHER: Merge individual bits into a bit sequence (S7-1500) | 0 |
| 1.6.12 | GATHER_BLK: Merge individual bits into multiple elements of an ARRAY of bit sequence (S7-1500) | 0 |
| 1.6.13 | AssignmentAttempt: Attempt assignment to a reference (S7-1500) | 0 |
| 1.6.14 | ARRAY DB (S7-1500) | 0 |
| 1.6.14.1 | ReadFromArrayDB: Read from ARRAY data block (S7-1500) | 0 |
| 1.6.14.2 | WriteToArrayDB: Write to ARRAY data block (S7-1500) | 0 |
| 1.6.14.3 | ReadFromArrayDBL: Read from array data block in load memory (S7-1500) | 0 |
| 1.6.14.4 | WriteToArrayDBL: Write to array data block in load memory (S7-1500) | 0 |
| 1.6.15 | Read / Write access (S7-1500) | 0 |
| 1.6.15.1 | PEEK: Read memory address (S7-1500) | 0 |
| 1.6.15.2 | PEEK_BOOL: Read memory bit (S7-1500) | 0 |
| 1.6.15.3 | POKE: Write memory address (S7-1500) | 0 |
| 1.6.15.4 | POKE_BOOL: Write memory bit (S7-1500) | 0 |
| 1.6.15.5 | POKE_BLK: Write memory area (S7-1500) | 0 |
| 1.6.15.6 | READ_LITTLE: Read data in little endian format (S7-1500) | 0 |
| 1.6.15.7 | WRITE_LITTLE: Write data in little endian format (S7-1500) | 0 |
| 1.6.15.8 | READ_BIG: Read data in big endian format (S7-1500) | 0 |
| 1.6.15.9 | WRITE_BIG: Write data in big endian format (S7-1500) | 0 |
| 1.6.16 | VARIANT (S7-1500) | 0 |
| 1.6.16.1 | VariantGet: Read out VARIANT tag value (S7-1500) | 0 |
| 1.6.16.2 | VariantPut: Write VARIANT tag value (S7-1500) | 0 |
| 1.6.16.3 | CountOfElements: Get number of ARRAY elements (S7-1500) | 0 |
| 1.6.17 | Symbolic move (S7-1500) | 0 |
| 1.6.17.1 | Symbolic access during runtime (S7-1500) | 0 |
| 1.6.17.2 | ResolveSymbols: Resolve several symbols (S7-1500) | 0 |
| 1.6.17.3 | System data type ResolvedSymbol (S7-1500) | 0 |
| 1.6.17.4 | MoveResolvedSymbolsToBuffer: Read values from resolved symbols and write them into buffer (S7-1500) | 0 |
| 1.6.17.5 | MoveResolvedSymbolsFromBuffer: Read values from buffer and write them into resolved symbols (S7-1500) | 0 |
| 1.6.18 | ARRAY[*] (S7-1500) | 0 |
| 1.6.18.1 | LOWER_BOUND: Read out low ARRAY limit (S7-1500) | 0 |
| 1.6.18.2 | UPPER_BOUND: Read out high ARRAY limit (S7-1500) | 0 |
| 1.6.19 | Legacy (S7-1500) | 0 |
| 1.6.19.1 | BLKMOV: Move block (S7-1500) | 0 |
| 1.6.19.2 | UBLKMOV: Move block uninterruptible (S7-1500) | 0 |

| | | |
|-----------|--|---|
| 1.6.19.3 | FILL: Fill block (S7-1500) | 0 |
| 1.7 | Conversion operations (S7-1500) | 0 |
| 1.7.1 | SCALE_X: Scale (S7-1500) | 0 |
| 1.7.2 | NORM_X: Normalize (S7-1500) | 0 |
| 1.7.3 | VARIANT (S7-1500) | 0 |
| 1.7.3.1 | VARIANT_TO_DB_ANY: Convert VARIANT to DB_ANY (S7-1500) | 0 |
| 1.7.3.2 | DB_ANY_TO_VARIANT: Convert DB_ANY to VARIANT (S7-1500) | 0 |
| 1.7.4 | Legacy (S7-1500) | 0 |
| 1.7.4.1 | SCALE: Scale (S7-1500) | 0 |
| 1.7.4.2 | UNSCALE: Unscale (S7-1500) | 0 |
| 1.8 | Program control operations (S7-1500) | 0 |
| 1.8.1 | Runtime control (S7-1500) | 0 |
| 1.8.1.1 | ENDIS_PW: Locking and unlocking passwords of the CPU access levels (S7-1500) | 0 |
| 1.8.1.2 | RE_TRIGR: Restart cycle monitoring time (S7-1500) | 0 |
| 1.8.1.3 | STP: Exit program (S7-1500) | 0 |
| 1.8.1.4 | GET_ERROR: Get error locally (S7-1500) | 0 |
| 1.8.1.5 | GET_ERR_ID: Get error ID locally (S7-1500) | 0 |
| 1.8.1.6 | INIT_RD: Initialize all retain data (S7-1500) | 0 |
| 1.8.1.7 | WAIT: Configure time delay (S7-1500) | 0 |
| 1.8.1.8 | RUNTIME: Measure program runtime (S7-1500) | 0 |
| 1.9 | Word logic operations (S7-1500) | 0 |
| 1.9.1 | DECO: Decode (S7-1500) | 0 |
| 1.9.2 | ENCO: Encode (S7-1500) | 0 |
| 1.9.3 | SEL: Select (S7-1500) | 0 |
| 1.10 | Legacy (S7-1500) | 0 |
| 1.10.1 | DRUM: Implement sequencer (S7-1500) | 0 |
| 1.10.2 | DCAT: Discrete control-timer alarm (S7-1500) | 0 |
| 1.10.3 | MCAT: Motor control-timer alarm (S7-1500) | 0 |
| 1.10.4 | IMC: Compare input bits with the bits of a mask (S7-1500) | 0 |
| 1.10.5 | SMC: Compare scan matrix (S7-1500) | 0 |
| 1.10.6 | LEAD_LAG: Lead and lag algorithm (S7-1500) | 0 |
| 1.10.7 | SEG: Create bit pattern for seven-segment display (S7-1500) | 0 |
| 1.10.8 | BCDCPL: Create tens complement (S7-1500) | 0 |
| 1.10.9 | BITSUM: Count number of set bits (S7-1500) | 0 |
| 1.11 | STL Mnemonic (S7-1500) | 0 |
| 1.11.1 | Bit logic operations (S7-1500) | 0 |
| 1.11.1.1 | A: AND logic operation (S7-1500) | 0 |
| 1.11.1.2 | AN: Negated AND logic operation (S7-1500) | 0 |
| 1.11.1.3 | O: OR logic operation (S7-1500) | 0 |
| 1.11.1.4 | ON: Negated OR logic operation (S7-1500) | 0 |
| 1.11.1.5 | X: EXCLUSIVE OR logic operation (S7-1500) | 0 |
| 1.11.1.6 | XN: Negated EXCLUSIVE OR logic operation (S7-1500) | 0 |
| 1.11.1.7 | O: OR logic operation of AND functions (S7-1500) | 0 |
| 1.11.1.8 | A(: AND logic operation with branch (S7-1500) | 0 |
| 1.11.1.9 | AN(: Negated AND logic operation with branch (S7-1500) | 0 |
| 1.11.1.10 | O(: OR logic operation with branch (S7-1500) | 0 |

| | | |
|-----------|---|---|
| 1.11.1.11 | ON(: Negated OR logic operation with branch (S7-1500) | 0 |
| 1.11.1.12 | X(: EXCLUSIVE OR logic operation with branch (S7-1500) | 0 |
| 1.11.1.13 | XN(: Negated EXCLUSIVE OR logic operation with branch (S7-1500) | 0 |
| 1.11.1.14 |): Close branch (S7-1500) | 0 |
| 1.11.1.15 | =: Assignment (S7-1500) | 0 |
| 1.11.1.16 | R: Reset (S7-1500) | 0 |
| 1.11.1.17 | S: Set (S7-1500) | 0 |
| 1.11.1.18 | NOT: Invert RLO (S7-1500) | 0 |
| 1.11.1.19 | SET: Set RLO to 1 (S7-1500) | 0 |
| 1.11.1.20 | CLR: Reset RLO to 0 (S7-1500) | 0 |
| 1.11.1.21 | SAVE: Save RLO in BR bit (S7-1500) | 0 |
| 1.11.1.22 | FN: Scan RLO for negative signal edge (S7-1500) | 0 |
| 1.11.1.23 | FP: Scan RLO for positive signal edge (S7-1500) | 0 |
| 1.11.2 | Timer operations (S7-1500) | 0 |
| 1.11.2.1 | FR: Enable timer (S7-1500) | 0 |
| 1.11.2.2 | L: Load timer value (S7-1500) | 0 |
| 1.11.2.3 | LC: Load BCD-coded timer value (S7-1500) | 0 |
| 1.11.2.4 | R: Reset timer (S7-1500) | 0 |
| 1.11.2.5 | SP: Start pulse timer (S7-1500) | 0 |
| 1.11.2.6 | SE: Start extended pulse timer (S7-1500) | 0 |
| 1.11.2.7 | SD: Start on-delay timer (S7-1500) | 0 |
| 1.11.2.8 | SS: Start retentive on-delay timer (S7-1500) | 0 |
| 1.11.2.9 | SF: Start off-delay timer (S7-1500) | 0 |
| 1.11.3 | Counter operations (S7-1500) | 0 |
| 1.11.3.1 | FR: Enable counter (S7-1500) | 0 |
| 1.11.3.2 | L: Load counter (S7-1500) | 0 |
| 1.11.3.3 | LC: Load BCD-coded counter value (S7-1500) | 0 |
| 1.11.3.4 | R: Reset counter (S7-1500) | 0 |
| 1.11.3.5 | S: Set counter (S7-1500) | 0 |
| 1.11.3.6 | CU: Count up (S7-1500) | 0 |
| 1.11.3.7 | CD: Count down (S7-1500) | 0 |
| 1.11.4 | Comparator operations (S7-1500) | 0 |
| 1.11.4.1 | ? I: Compare 16-bit integers (S7-1500) | 0 |
| 1.11.4.2 | ? D: Compare 32-bit integers (S7-1500) | 0 |
| 1.11.4.3 | ? R: Compare floating-point numbers (S7-1500) | 0 |
| 1.11.5 | Math functions (S7-1500) | 0 |
| 1.11.5.1 | Integers (S7-1500) | 0 |
| 1.11.5.2 | Floating-point numbers (S7-1500) | 0 |
| 1.11.5.3 | Advanced floating-point numbers (S7-1500) | 0 |
| 1.11.6 | Load and transfer (S7-1500) | 0 |
| 1.11.6.1 | Load (S7-1500) | 0 |
| 1.11.6.2 | Transfer (S7-1500) | 0 |
| 1.11.7 | Conversion operations (S7-1500) | 0 |
| 1.11.7.1 | BTI: Convert BCD to integer (16 bit) (S7-1500) | 0 |
| 1.11.7.2 | ITB: Convert integer (16 bit) to BCD (S7-1500) | 0 |
| 1.11.7.3 | BTD: Convert BCD to integer (32 bit) (S7-1500) | 0 |

| | | |
|-----------|---|---|
| 1.11.7.4 | ITD: Convert integer (16 bit) to integer (32 bit) (S7-1500) | 0 |
| 1.11.7.5 | DTB: Convert integer (32 bit) to BCD (S7-1500) | 0 |
| 1.11.7.6 | DTR: Convert integer (32 bit) to floating-point number (S7-1500) | 0 |
| 1.11.7.7 | INVI: Create ones complement integer (16 bit) (S7-1500) | 0 |
| 1.11.7.8 | INVD: Create ones complement double integer (32 bit) (S7-1500) | 0 |
| 1.11.7.9 | NEGI: Negate integer (16 bit) (S7-1500) | 0 |
| 1.11.7.10 | NEGD: Negate integer (32 bit) (S7-1500) | 0 |
| 1.11.7.11 | NEGR: Negate floating-point number (S7-1500) | 0 |
| 1.11.7.12 | CAW: Switch bytes in the right word of accumulator 1 (S7-1500) | 0 |
| 1.11.7.13 | CAD: Switch all bytes in accumulator 1 (S7-1500) | 0 |
| 1.11.7.14 | RND: Round numerical value (S7-1500) | 0 |
| 1.11.7.15 | TRUNC: Truncate numerical value (S7-1500) | 0 |
| 1.11.7.16 | RND+: Generate next higher integer from floating-point number (S7-1500) | 0 |
| 1.11.7.17 | RND-: Generate next lower integer from floating-point number (S7-1500) | 0 |
| 1.11.8 | Program control operations (S7-1500) | 0 |
| 1.11.8.1 | Jumps (S7-1500) | 0 |
| 1.11.8.2 | Data blocks (S7-1500) | 0 |
| 1.11.8.3 | Code blocks (S7-1500) | 0 |
| 1.11.9 | Word logic operations (S7-1500) | 0 |
| 1.11.9.1 | AW: AND logic operation word by word (S7-1500) | 0 |
| 1.11.9.2 | OW: OR logic operation word by word (S7-1500) | 0 |
| 1.11.9.3 | XOW: EXCLUSIVE OR logic operation word by word (S7-1500) | 0 |
| 1.11.9.4 | AD: AND logic operation double word by double word (S7-1500) | 0 |
| 1.11.9.5 | OD: OR logic operation double word by double word (S7-1500) | 0 |
| 1.11.9.6 | XOD: EXCLUSIVE OR logic operation double word by double word (S7-1500) | 0 |
| 1.11.10 | Shift and rotate (S7-1500) | 0 |
| 1.11.10.1 | Shift (S7-1500) | 0 |
| 1.11.10.2 | Rotate (S7-1500) | 0 |
| 1.11.11 | Additional instructions (S7-1500) | 0 |
| 1.11.11.1 | Accumulator (S7-1500) | 0 |
| 1.11.11.2 | Address register (S7-1500) | 0 |
| 1.11.11.3 | Null instructions (S7-1500) | 0 |

STL (S7-1500)

This section contains information on the following topics:

- Bit logic operations (S7-1500)
- Timer operations (S7-1500)
- Counter operations (S7-1500)
- Comparator operations (S7-1500)
- Math functions (S7-1500)
- Move operations (S7-1500)
- Conversion operations (S7-1500)
- Program control operations (S7-1500)
- Word logic operations (S7-1500)
- Legacy (S7-1500)
- STL Mnemonic (S7-1500)

1.1 Bit logic operations (S7-1500)

1.1.1 R_TRIG: Detect positive signal edge (S7-1500)

Description

With the "Detect positive signal edge" instruction, you can detect a state change from "0" to "1" at the CLK input. The instruction compares the current value at the CLK input with the state of the previous query (edge memory bit) that is saved in the specified instance. If the instruction detects a state change at the CLK input from "0" to "1", a positive signal edge is generated at the Q output, i.e., the output has the value TRUE or "1" for exactly one cycle.

In all other cases, the signal state at the output of the instruction is "0".

Parameters


The following table shows the parameters of the "Detect positive signal edge" instruction:

| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|-----------|-------------|-------------|
|------------|-------------|-----------|-------------|-------------|

| | | | | |
|-----|--------|------|---------------------------|--|
| CLK | Input | BOOL | I, Q, M, D, L or constant | Incoming signal, the edge of which is to be queried. |
| Q | Output | BOOL | I, Q, M, D, L | Result of edge evaluation |

Example

The following example shows how the instruction works:

| STL  | Explanation |
|---|--|
| CALL R_TRIG, "R_TRIG_DB" | // The instruction is called. |
| CLK := "TagIn" | // Positive signal edge is detected. |
| Q := "TagOut" | // Signal state "1" on positive signal edge. |

The previous state of the tag at the CLK input is stored in the "R_TRIG_DB" tag. If a change in the signal state from "0" to "1" is detected in the "TagIn" operand, the "TagOut" output has signal state "1" for one cycle.

See also

[Overview of the valid data types](#)
[Basic information on the status word \(S7-1500\)](#)
[Inserting STL instructions \(S7-300, S7-400, S7-1500\)](#)
[Editing STL instructions \(S7-300, S7-400, S7-1500\)](#)
[Instances](#)
[Querying and setting status bits in STL \(S7-1500\)](#)
[Memory areas \(S7-1500\)](#)

1.1.2 F_TRIG: Detect negative signal edge (S7-1500)

Description

With the "Detect negative signal edge" instruction, you can detect a state change from "1" to "0" at the CLK input. The instruction compares the current value at the CLK input with the state of the previous query (edge memory bit) that is saved in the specified instance. If the instruction detects a state change at the CLK input from "1" to "0", a negative signal edge is generated at the Q output, i.e., the output has the value TRUE or "1" for exactly one cycle.

In all other cases, the signal state at the output of the instruction is "0".

Note

Behavior after CPU startup

The IEC61131 standard describes that the instruction "F_TRIG" sets the output "Q" to TRUE for one cycle if the input "CLK" has the value FALSE at CPU startup.

In order for "F_TRIG" to show the behavior described in the standard after a CPU startup, the "Stat_Bit" instance must be initialized with TRUE.


Parameters

The following table shows the parameters of the "Detect negative signal edge" instruction:

| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|-----------|---------------------------|--|
| CLK | Input | BOOL | I, Q, M, D, L or constant | Incoming signal, the edge of which is to be queried. |
| Q | Output | BOOL | I, Q, M, D, L | Result of edge evaluation |

Example

The following example shows how the instruction works:

| STL  | Explanation |
|---|--|
| CALL F_TRIG, "F_TRIG_DB" | // The instruction is called. |
| CLK := "TagIn" | // Negative signal edge is detected. |
| Q := "TagOut" | // Signal state "1" on negative signal edge. |

The previous state of the tag at the CLK input is stored in the "F_TRIG_DB" tag. If a change in the signal state from "1" to "0" is detected in the "TagIn" operand, the "TagOut" output has signal state "1" for one cycle.

See also

Overview of the valid data types
Basic information on the status word (S7-1500)
Inserting STL instructions (S7-300, S7-400, S7-1500)
Editing STL instructions (S7-300, S7-400, S7-1500)
Instances
Querying and setting status bits in STL (S7-1500)
Memory areas (S7-1500)

1.2 Timer operations (S7-1500)

1.2.1 TP: Generate pulse (S7-1500)

Description

You can use the "Generate pulse" instruction to set the Q output for a programmed duration. The instruction is started when the result of logic operation (RLO) of the IN parameter changes

from "0" to "1" (positive signal edge). The programmed time PT begins when the instruction starts. The Q parameter is set for the time PT, regardless of the subsequent changes in the input signal. While the time PT is running, the detection of a new positive signal edge at the IN input has no influence on the signal state at the Q output.

You can query the current time value at the ET output. The time value starts at T#0s and ends when the value of the time PT is reached. When the time PT has elapsed and the signal state at input IN is "0", the ET output is reset. If the instruction is not called in the program because it is skipped, for example, the ET output returns a constant value as soon as the time has expired.

The operating system resets the instances of the "Generate pulse" instruction during a cold restart. If you want instances of the instruction to be initialized after a warm restart, call the instances to be initialized in a startup OB with the value "0" set for the PT parameter. If instances of the "Generate pulse" instruction are located within another block, you can reset these instances, for example, by initializing the higher-level block.

Use the "Call block" (CALL) instruction in the program code to call the "Generate pulse" instruction.

After the selection of the data type from the drop-down list "???", the "Call options" dialog opens.

Each call of the "Generate pulse" instruction must be assigned to an IEC timer in which the instance data is stored. An IEC Timer is a structure of the data type IEC_TIMER, IEC_LTIMER, TP_TIME or TP_LTIME that you can declare as follows:

- Declaration of a data block of system data type IEC_TIMER or IEC_LTIMER (for example, "MyIEC_TIMER")
- Declaration as a local tag of the type TP_TIME or TP_LTIME in the "Static" section of a block (for example, #MyTP_TIMER)

Updating the actual values in the instance data

The instance data from "Generate pulse" is updated according to the following rules:

- IN input

The "Generate pulse" instruction compares the current RLO with the RLO from the previous query, which is saved in the IN parameter in the instance data. If the instruction detects a change in the RLO from "0" to "1", there is a positive signal edge and the time measurement is started. After the "Generate pulse" instruction has been processed, the value of the IN parameter is updated in the instance data and is used as edge memory bit for the next query.

Note that the edge evaluation is disrupted when the actual values of the IN parameter are written or initialized by other functions.

- PT input

The value at the PT input is written to the PT parameter in the instance data when the edge changes at the IN input.

- Q and ET outputs

The actual values of the Q and ET outputs are updated in the following cases:

- When the instruction is called, if the ET or Q outputs are interconnected.

Or

- At an access to Q or ET.

If the outputs are not interconnected and also not queried, the current time value at the Q and ET outputs is not updated. The outputs are not updated, even if the instruction is skipped in the program.

The internal parameters of the "Generate pulse" instruction are used to calculate the time values for Q and ET. Note that the time measurement is disrupted when the actual values of the instruction are written or initialized by other functions.



DANGER

Danger when reinitializing the actual values

Reinitializing the actual values of an IEC timer while the time measurement is running disrupts the function of the IEC timer. Changing the actual values can result in inconsistencies between the program and the actual process. This can cause serious damage to property and personal injury.

The following functions can cause the actual values to be reinitialized:

- Loading the block with reinitialization
- Loading snapshots as actual values
- Controlling or forcing the actual values
- The "WRIT_DBL" instruction

Before you execute these functions, take the following precautions:

- Make sure that the plant is in a safe state before you overwrite the actual values.
- Make sure that the IEC timer has expired before initializing its actual values.
- If you overwrite the actual values with a snapshot, make sure that the snapshot was taken at a time when the system was in a safe state.
- Make sure that the program does not read or write the affected data during transmission.

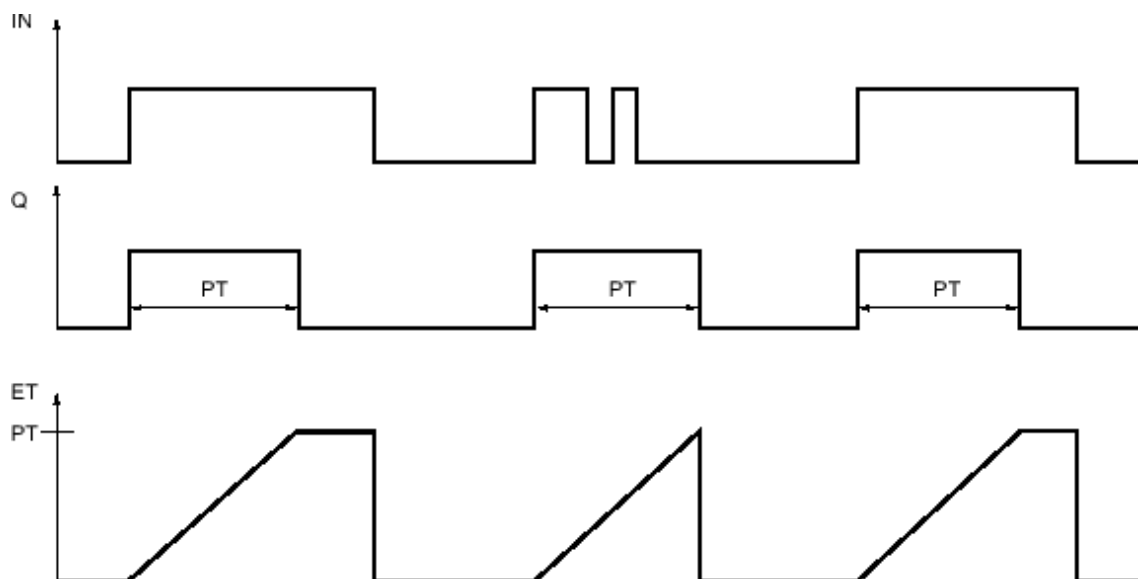
Parameters

The following table shows the parameters of the "Generate pulse" instruction:

| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|-------------|------------------------------|---|
| IN | Input | BOOL | I, Q, M, D, L, P or constant | Start input |
| PT | Input | TIME, LTIME | I, Q, M, D, L, P or constant | Duration of the pulse. The value of the PT parameter must be positive. |
| Q | Output | BOOL | I, Q, M, D, L, P | Pulse output |
| ET | Output | TIME, LTIME | I, Q, M, D, L, P | Current timer value |

Pulse timing diagram

The following figure shows the behavior of the "Generate pulse" instruction after the start:



Example

The following example shows how the instruction works:

| STL | Explanation |
|--------------------------------|---|
| <code>CALL TP, "TP_DB"</code> | // The instruction is called. The "TP_DB" data block is assigned to the instruction. // Select the required data type from the "???" drop-down list. |
| <code>IN := "Tag_Start"</code> | // The instruction is executed on a rising signal edge of the operand. |

```

PT :=                                // Duration of the pulse
"Tag_PresetTIME"

Q := "Tag_Output" // The operand is set for the duration
                  that was specified by the
                  "Tag_PresetTIME" operand.

ET :=                                // Current timer value
"Tag_ElapsedTIME"

```

See also

Overview of the valid data types

Basic information on the status word (S7-1500)

Inserting STL instructions (S7-300, S7-400, S7-1500)

Editing STL instructions (S7-300, S7-400, S7-1500)

Instances

Querying and setting status bits in STL (S7-1500)

Memory areas (S7-1500)

1.2.2 TON: Generate on-delay (S7-1500)

Description

You can use the "Generate on-delay" instruction to delay setting of the Q output by the programmed time PT. The instruction is started when the result of logic operation (RLO) of the IN parameter changes from "0" to "1" (positive signal edge). The programmed time PT begins when the instruction starts. When the PT duration has expired, the Q parameter returns signal state "1". The Q parameter remains set as long as the start input IN is still "1". When the signal state at the start input changes from "1" to "0", the Q parameter is reset. The timer function is started again when a new positive signal edge is detected at the start input.

You can query the current time value at the ET output. The time value starts at T#0s and ends when the value of the time PT is reached. The ET parameter is reset as soon as the signal state of the IN parameter changes to "0". If the instruction is not called in the program because it is skipped, for example, the ET output returns a constant value as soon as the time PT has expired.

In the program code, you call the "Generate on-delay" instruction with the "Call block" (CALL) instruction.

After the selection of the data type from the drop-down list "???", the "Call options" dialog opens.

Each call of the "Generate on-delay" instruction must be assigned to an IEC timer in which the instance data is stored. An IEC Timer is a structure of the data type IEC_TIMER, IEC_LTIMER, TON_TIME or TON_LTIME that you can declare as follows:

- Declaration of a data block of system data type IEC_TIMER or IEC_LTIMER (for example, "MyIEC_TIMER")
- Declaration as a local tag of the type TON_TIME or TON_LTIME in the "Static" section of a block (for example, #MyTON_TIMER)

Updating the actual values in the instance data

The instance data from "Generate on-delay" is updated according to the following rules:

- IN input

The "Generate on-delay" instruction compares the current RLO with the RLO from the previous query, which is saved in the IN parameter in the instance data. If the instruction detects a change in the RLO from "0" to "1", there is a positive signal edge and the time measurement is started. After the "Generate on-delay" instruction has been processed, the value of the IN parameter is updated in the instance data and is used as edge memory bit for the next query.

Note that the edge evaluation is disrupted when the actual values of the IN parameter are written or initialized by other functions.

- PT input

The value at the PT input is written to the PT parameter in the instance data when the edge changes at the IN input.

- Q and ET outputs

The actual values of the Q and ET outputs are updated in the following cases:

- When the instruction is called, if the ET or Q outputs are interconnected.

Or

- At an access to Q or ET.

If the outputs are not interconnected and also not queried, the current time value at the Q and ET outputs is not updated. The outputs are not updated, even if the instruction is skipped in the program.

The internal parameters of the "Generate on-delay" instruction are used to calculate the time values for Q and ET. Note that the time measurement is disrupted when the actual values of the instruction are written or initialized by other functions.

DANGER

Danger when reinitializing the actual values

Reinitializing the actual values of an IEC timer while the time measurement is running disrupts the function of the IEC timer. Changing the actual values can result in inconsistencies between the program and the actual process. This can cause serious damage to property and personal injury.

The following functions can cause the actual values to be reinitialized:

- Loading the block with reinitialization
- Loading snapshots as actual values
- Controlling or forcing the actual values
- The "WRIT_DBL" instruction

Before you execute these functions, take the following precautions:

- Make sure that the plant is in a safe state before you overwrite the actual values.
- Make sure that the IEC timer has expired before initializing its actual values.
- If you overwrite the actual values with a snapshot, make sure that the snapshot was taken at a time when the system was in a safe state.
- Make sure that the program does not read or write the affected data during transmission.

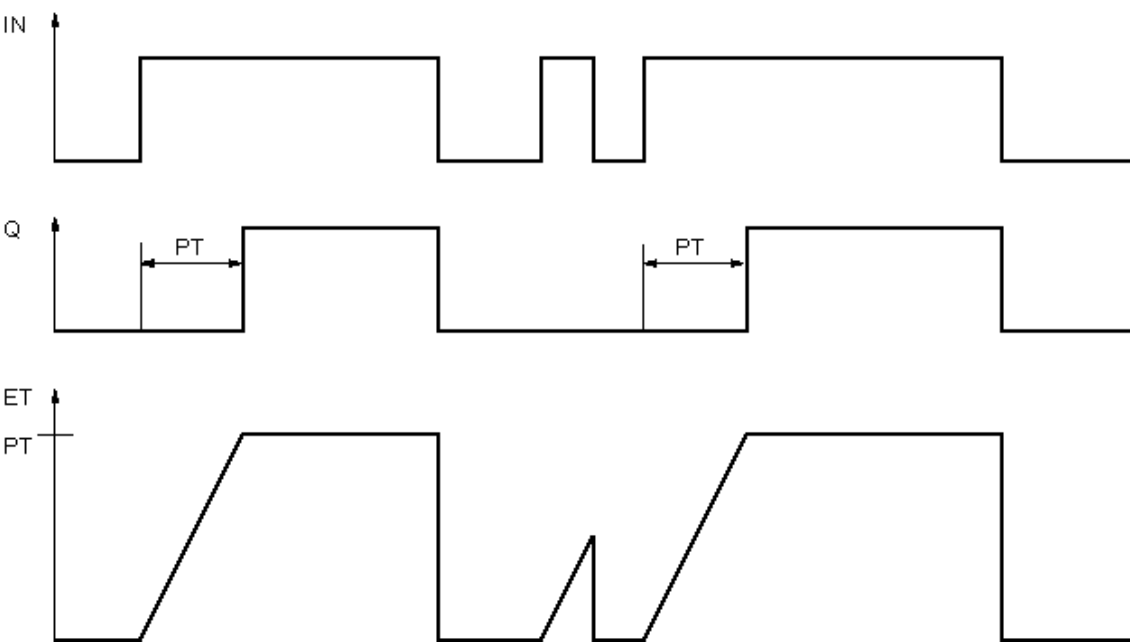
Parameters

The following table shows the parameters of the "Generate on-delay" instruction:

| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|-------------|------------------------------|---|
| IN | Input | BOOL | I, Q, M, D, L, P or constant | Start input |
| PT | Input | TIME, LTIME | I, Q, M, D, L, P or constant | Duration of the on-delay The value of the PT parameter must be positive. |
| Q | Output | BOOL | I, Q, M, D, L, P | Signal state that is delayed by the time PT. |
| ET | Output | TIME, LTIME | I, Q, M, D, L, P | Current timer value |


Pulse timing diagram

The following figure shows the behavior of the "Generate on-delay" instruction after the start:



Example

The following example shows how the instruction works:

| STL  | Explanation |
|---|--|
| <pre>CALL "TON", "TON_DB"</pre> | <pre>// The instruction is called. The "TON_DB" data block is assigned to the instruction.</pre> |
| | <pre>// Select the required data type from the "???" drop-down list.</pre> |
| <pre>IN := "Tag_Start"</pre> | <pre>// The instruction is executed on a rising signal edge of the operand.</pre> |
| <pre>PT := "Tag_PresetTIME"</pre> | <pre>// Specifies the time by which the rising signal edge of the IN parameter is delayed.</pre> |
| <pre>Q := "Tag_Output"</pre> | <pre>// The operand is set if the duration PT that was specified by the "Tag_PresetTIME" tag has expired.</pre> <pre>// The Q parameter remains set as long as the "Tag_Start" variable is "1".</pre> <pre>// When the signal state at the start input changes from "1" to "0", the operand of the Q parameter is reset.</pre> |
| <pre>ET := "Tag_ElapsedTIME"</pre> | <pre>// Current timer value</pre> |

See also

Overview of the valid data types
Basic information on the status word (S7-1500)
Inserting STL instructions (S7-300, S7-400, S7-1500)
Editing STL instructions (S7-300, S7-400, S7-1500)
Instances
Querying and setting status bits in STL (S7-1500)
Memory areas (S7-1500)

1.2.3 TOF: Generate off-delay (S7-1500)

Description

You can use the "Generate off-delay" instruction to reset the Q output by the programmed time PT. The Q output is set when the result of logic operation (RLO) at input IN changes from "0" to "1" (positive signal edge). When the signal state at input IN changes back to "0" (negative signal edge), the programmed time PT starts. Output Q remains set as long as the time duration PT is running. When the PT time duration expires, the Q output is reset. If the signal state at input IN changes to "1" before the PT time duration expires, the timer is reset. The signal state at the output Q continues to be "1".

The current time value can be queried in the ET parameter. The time value starts at T#0s and ends when the value of the time PT is reached. When the time PT expires, the ET parameter remains set to the current value until the IN parameter changes back to "1". If input IN changes to "1" before the time PT elapses, the ET output is reset to the value T#0s. If the instruction is not called in the program because it is skipped, for example, the ET output returns a constant value as soon as the time has expired.

In the program code, you call the "Generate off-delay" instruction with the "Call block" (CALL) instruction.

After the selection of the data type from the drop-down list "???", the "Call options" dialog opens.

Each call of the "Generate off-delay" instruction must be assigned to an IEC timer in which the instance data is stored. An IEC Timer is a structure of the data type IEC_TIMER, IEC_LTIMER, TOF_TIME or TOF_LTIME that you can declare as follows:

- Declaration of a data block of system data type IEC_TIMER or IEC_LTIMER (for example, "MyIEC_TIMER")
- Declaration as a local tag of the type TOF_TIME or TOF_LTIME in the "Static" section of a block (for example, #MyTOF_TIMER)

The operating system resets the instances of the "Generate off-delay" instruction during a cold restart. If you want instances of the instruction to be initialized after a warm restart, call the instances to be initialized in a startup OB with the value "0" set for the PT parameter. If instances of the "Generate off-delay" instruction are located within another block, you can reset these instances, for example, by initializing the higher-level block.

Updating the actual values in the instance data

The instance data from "Generate off-delay" is updated according to the following rules:

- IN input

The "Start off-delay timer" instruction compares the current RLO with the RLO from the previous query, which is saved in the "IN" parameter in the instance data. If the instruction detects a change in the RLO from "1" to "0", there is a negative signal edge and the time measurement is started. After the "Start off-delay timer" instruction has been processed, the value of the IN parameter is updated in the instance data and is used as edge memory bit for the next query.

Note that the edge evaluation is disrupted when the actual values of the "IN" parameter are written or initialized by other functions.

- PT input

The value at the PT input is written to the PT parameter in the instance data when the edge changes at the IN input.

- Q and ET outputs

The actual values of the Q and ET outputs are updated in the following cases:

- When the instruction is called, if the ET or Q outputs are interconnected.

Or

- At an access to Q or ET.

If the outputs are not interconnected and also not queried, the current time value at the Q and ET outputs is not updated. The outputs are not updated, even if the instruction is skipped in the program.

The internal parameters of the "Start off-delay timer" instruction are used to calculate the time values for Q and ET. Note that the time measurement is disrupted when the actual values of the instruction are written or initialized by other functions.



DANGER

Danger when reinitializing the actual values

Reinitializing the actual values of an IEC timer while the time measurement is running disrupts the function of the IEC timer. Changing the actual values can result in inconsistencies between the program and the actual process. This can cause serious damage to property and personal injury.

The following functions can cause the actual values to be reinitialized:

- Loading the block with reinitialization
- Loading snapshots as actual values
- Controlling or forcing the actual values
- The "WRIT_DBL" instruction

Before you execute these functions, take the following precautions:

- Make sure that the plant is in a safe state before you overwrite the actual values.
- Make sure that the IEC timer has expired before initializing its actual values.
- If you overwrite the actual values with a snapshot, make sure that the snapshot was taken at a time when the system was in a safe state.
- Make sure that the program does not read or write the affected data during transmission.

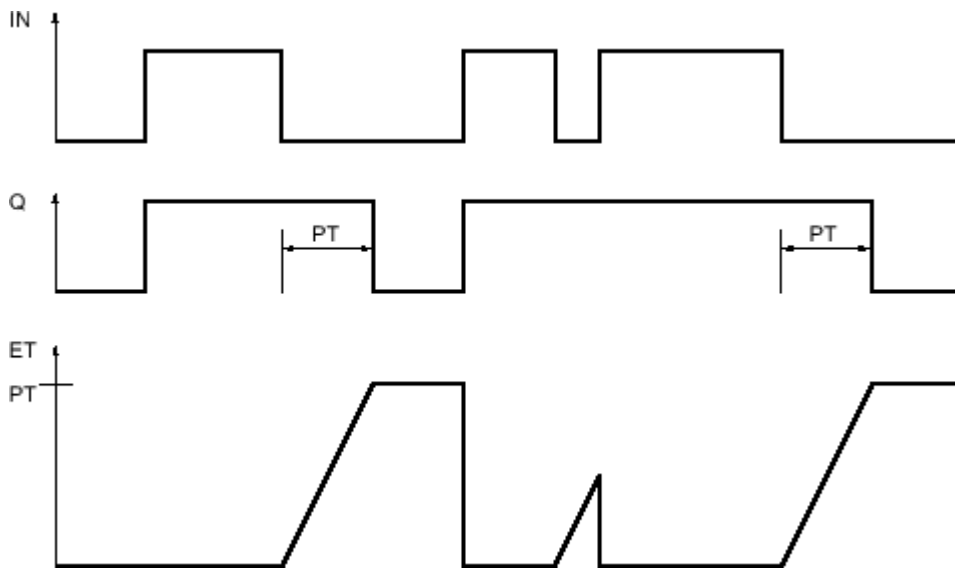
Parameters

The following table shows the parameters of the "Generate off-delay" instruction:

| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|-------------|------------------------------|--|
| IN | Input | BOOL | I, Q, M, D, L, P or constant | Start input |
| PT | Input | TIME, LTIME | I, Q, M, D, L, P or constant | Duration of the off delay The value of the PT parameter must be positive. |
| Q | Output | BOOL | I, Q, M, D, L, P | Signal state that is delayed by the time PT. |
| ET | Output | TIME, LTIME | I, Q, M, D, L, P | Current timer value |


Pulse timing diagram

The following figure shows the behavior of the "Generate off-delay" instruction after the start:



Example

The following example shows how the instruction works:

| STL  | Explanation |
|--|--|
| <code>CALL TOF, "TOF_DB"</code> | // The instruction is called. The "TOF_DB" data block is assigned to the instruction. // Select the required data type from the "???" drop-down list. |
| <code>IN := "Tag_Start"</code> | // The instruction is executed on a rising signal edge of the operand. |
| <code>PT := "Tag_PresetTIME"</code> | // Specifies the time by which the falling signal edge of the IN parameter is delayed. |
| <code>Q := "Tag_Output"</code> | // The operand is set when the instruction is started by a rising signal edge of the IN parameter. // When the value at the IN parameter changes from "1" to "0", the "Tag_Output" operand remains set as long as the time specified by the "Tag_PresetTIME" tag is running. // When the time of the PT parameter expires, the operand is reset. |
| <code>ET := "Tag_ElapsedTIME"</code> | // Current timer value |

See also

Overview of the valid data types
Basic information on the status word (S7-1500)
Inserting STL instructions (S7-300, S7-400, S7-1500)
Editing STL instructions (S7-300, S7-400, S7-1500)
Instances
Querying and setting status bits in STL (S7-1500)
Memory areas (S7-1500)

1.2.4 TONR: Time accumulator (S7-1500)

Description

The instruction "Time accumulator" is used to accumulate time values within a period set by the parameter PT. When the signal state at the IN input changes from "0" to "1" (positive signal edge), the time measurement is executed and the time PT starts. While the time PT is running, the time values are accumulated that are recorded when the IN input has signal state "1". The accumulated time is written to the ET output and can be queried there. When the duration PT expires, the output Q has signal state "1". The Q parameter remains set to "1", even when the signal state at the IN parameter changes from "1" to "0" (negative signal edge).

The R input resets the ET and Q outputs regardless of the signal state of the start input.

The "Time accumulator" instruction can be placed within or at the end of the network. It requires a preceding logic operation.

In the program code, you call the "Time accumulator" instruction with the "Call block" (CALL) instruction.

Each call of the "Time accumulator" instruction must be assigned an IEC timer in which the instance data is stored. An IEC Timer is a structure of the data type IEC_TIMER, IEC_LTIMER, TONR_TIME or TONR_LTIME that you can declare as follows:

- Declaration of a data block of system data type IEC_TIMER or IEC_LTIMER (for example, "MyIEC_TIMER")
- Declaration as a local tag of the type TONR_TIME or TONR_LTIME in the "Static" section of a block (for example, #MyTONR_TIMER)

After the selection of the data type from the drop-down list "???", the "Call options" dialog opens.

Updating the actual values in the instance data

The instance data from "Time accumulator" is updated according to the following rules:

- IN input

The "Time accumulator" instruction compares the current RLO with the RLO from the previous query, which is saved in the "IN" parameter in the instance data. If the instruction detects a change in the RLO from "0" to "1", there is a positive signal edge and the time measurement is continued. If the instruction in the RLO detects a change from "1" to "0", there is a negative signal edge and the time measurement is interrupted. After the "Time

"accumulator" instruction has been processed, the value of the IN parameter is updated in the instance data and is used as edge memory bit for the next query.

Note that the edge evaluation is disrupted when the actual values of the IN parameter are written or initialized by other functions.

- PT input

The value at the PT input is written to the PT parameter in the instance data when the edge changes at the IN input.

- R input

The signal "1" at input R resets the time measurement and blocks it. Edges at the IN input are ignored. The signal "0" at input R enables time measurement again.

- Q and ET outputs

The actual values of the Q and ET outputs are updated in the following cases:

- When the instruction is called, if the ET or Q outputs are interconnected.

Or

- At an access to Q or ET.

If the outputs are not interconnected and also not queried, the current time value at the Q and ET outputs is not updated. The outputs are not updated, even if the instruction is skipped in the program.

The internal parameters of the "Time accumulator" instruction are used to calculate the time values for Q and ET. Note that the time measurement is disrupted when the actual values of the instruction are written or initialized by other functions.



DANGER

Danger when reinitializing the actual values

Reinitializing the actual values of an IEC timer while the time measurement is running disrupts the function of the IEC timer. Changing the actual values can result in inconsistencies between the program and the actual process. This can cause serious damage to property and personal injury.

The following functions can cause the actual values to be reinitialized:

- Loading the block with reinitialization
- Loading snapshots as actual values
- Controlling or forcing the actual values
- The "WRIT_DBL" instruction

Before you execute these functions, take the following precautions:

- Make sure that the plant is in a safe state before you overwrite the actual values.
- Make sure that the IEC timer has expired before initializing its actual values.
- If you overwrite the actual values with a snapshot, make sure that the snapshot was taken at a time when the system was in a safe state.
- Make sure that the program does not read or write the affected data during transmission.

Parameters

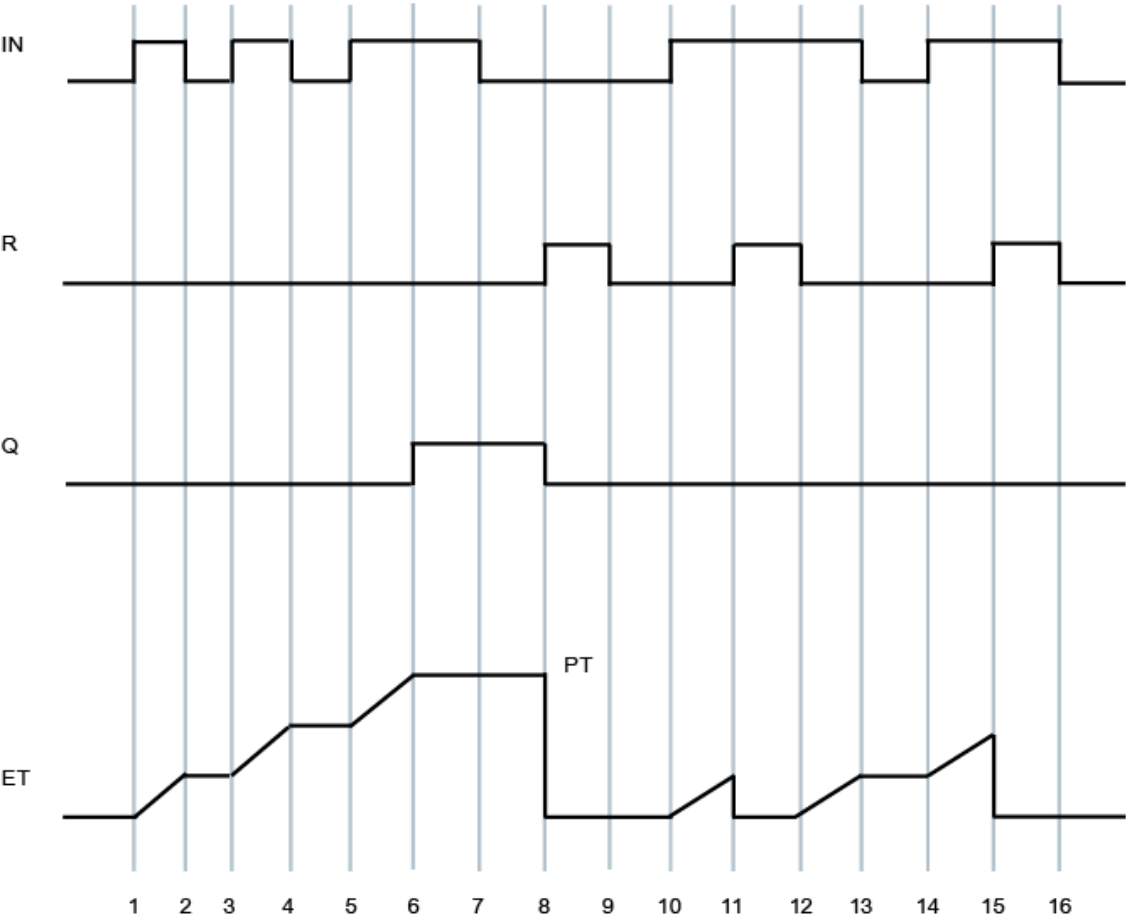
The following table shows the parameters of the "Time accumulator" instruction:

| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|-------------|------------------------------|---|
| IN | Input | BOOL | I, Q, M, D, L, P or constant | Start input |
| R | Input | BOOL | I, Q, M, D, L, P or constant | Reset input |
| PT | Input | TIME, LTIME | I, Q, M, D, L, P or constant | Maximum duration of time recording The value of the PT parameter must be positive. |
| Q | Output | BOOL | I, Q, M, D, L, P | Output that is set when time PT expires. |

| | | | | |
|----|--------|-------------|------------------|------------------|
| ET | Output | TIME, LTIME | I, Q, M, D, L, P | Accumulated time |
|----|--------|-------------|------------------|------------------|


Pulse timing diagram

The following figure shows the pulse timing diagram of the "Time accumulator" instruction:



Example

The following example shows how the instruction works:

| | |
|---|--|
| STL  | Explanation |
| CALL TONR, "TONR_DB" | // The instruction is called. The "TONR_DB" data block is assigned to the instruction. |
| | // Select the required data type from the "???" drop-down list. |
| IN := "Tag_Start" | // The instruction executes on a rising signal edge of the operand and the time duration at the PT input is started. |
| | While the time PT is running, the time values are accumulated that are |

```

recorded when the IN input has signal
state "1".

R := "Tag_Reset" // The R input resets the outputs ET and Q
regardless of the signal state at the
start input.

PT := // Specifies how long the time values are
"Tag_PresetTIME" accumulated.

Q := "Tag_Output" // The operand is set if the time at PT
has expired.

// The Q parameter remains set to "1",
even when the signal state at the IN
parameter changes from "1" to "0"
(negative signal edge).

ET := // The accumulated time is written to
"Tag_ElapsedTIME" output ET and can be queried there.

```

See also

- Overview of the valid data types
- Basic information on the status word (S7-1500)
- Inserting STL instructions (S7-300, S7-400, S7-1500)
- Editing STL instructions (S7-300, S7-400, S7-1500)
- Instances
- Querying and setting status bits in STL (S7-1500)
- Memory areas (S7-1500)

1.2.5 RESET_TIMER: Reset timer (S7-1500)

Description

You can use the "Reset timer" instruction to reset an IEC timer to "0". The structure components of the timer in the specified data block are reset to "0".

The instruction does not influence the RLO. At the TIMER parameter, the "Reset timer" instruction is assigned an IEC timer declared in the program.

Updating of the actual values

The instruction data is updated only when the instruction is called and not each time the assigned IEC timer is accessed. Querying the data is only identical from the call of the instruction to the next call of the instruction.



DANGER

Danger when reinitializing the actual values

Reinitializing the actual values of an IEC timer while the timer is running disrupts the function of the IEC timer. Changing the actual values can result in inconsistencies between the program and the actual process. This can cause serious damage to property and personal injury.

The following functions can cause the actual values to be reinitialized:

- Loading the block with reinitialization
- Loading snapshots as actual values
- Controlling or forcing the actual values
- The "WRIT_DBL" instruction

Before you execute these functions, take the following precautions:

- Make sure that the plant is in a safe state before you overwrite the actual values.
- Make sure that the IEC timer has expired before initializing its actual values.
- If you overwrite the actual values with a snapshot, make sure that the snapshot was taken at a time when the system was in a safe state.
- Make sure that the program does not read or write the affected data during transmission.

Parameters


The following table shows the parameters of the "Reset timer" instruction:

| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|--|-------------|-------------------------|
| TIMER | Output | IEC_TIMER, IEC_LTIMER, TP_TIME, TP_LTIME, TON_TIME, TON_LTIME, TOF_TIME, TOF_LTIME, TONR_TIME, TONR_LTIME | D, L | IEC timer that is reset |

You can select the data type of the instruction from the "???" drop-down list.

Example

The following example shows how the instruction works:

| STL  | Explanation |
|---|--|
| CALL TON, "TON_DB" | <p>// The instruction is called. The "TON_DB" instance data block is assigned to the instruction.</p> <p>// Select the required data type from the "???" drop-down list.</p> |
| IN := "Tag_Start" | // The instruction executes on a rising signal edge of the operand "Tag_Start". |
| PT := "Tag_PresetTIME" | // The IEC timer stored in the instance data block "TON_DB" is started with the time duration that is specified by the operand "Tag_PresetTIME". |
| Q := "Tag_Output" | <p>// The operand "Tag_Output" is set if the time duration PT specified by the operand "Tag_PresetTIME" has expired.</p> <p>// The parameter Q remains set as long as the "Tag_Start" operand still has the signal state "1".</p> <p>// When the signal state at the start input changes from "1" to "0", the operand of the Q parameter is reset.</p> |
| ET := "Tag_ElapsedTIME" | // Current timer value |
| A "Tag_Input_1" | // When the operand "Tag_Input_1" and |
| A "Tag_Input_2" | // The operand "Tag_Input_2" returns the signal state "1", |
| CALL RESET_TIMER | <p>// The "Reset timer" instruction is called.</p> <p>// Select the required data type from the "???" drop-down list.</p> |
| TIMER := "TON_DB" | // IEC timer is reset. |

See also

Overview of the valid data types

Basic information on the status word (S7-1500)

Inserting STL instructions (S7-300, S7-400, S7-1500)

Editing STL instructions (S7-300, S7-400, S7-1500)

Instances
Querying and setting status bits in STL (S7-1500)
Memory areas (S7-1500)

1.2.6 PRESET_TIMER: Load time duration (S7-1500)

Description

You can use the "Load time duration" instruction to set the time for an IEC timer. The instruction is executed in each cycle. The instruction writes the specified time to the structure of the specified IEC timer.

You assign an IEC timer declared in the program to the "Load time duration" instruction.

Note

If the specified IEC timer is running while the instruction executes, the instruction overwrites the current time of the specified IEC timer. This can change the timer status of the IEC timer.

Updating of the actual values

The instruction data is updated only when the instruction is called and each time the assigned IEC timer is accessed. The query on Q or ET (for example, "MyTimer".Q or "MyTimer".ET) updates the IEC_TIMER structure.



DANGER

Danger when reinitializing the actual values

Reinitializing the actual values of an IEC timer while the timer is running disrupts the function of the IEC timer. Changing the actual values can result in inconsistencies between the program and the actual process. This can cause serious damage to property and personal injury.

The following functions can cause the actual values to be reinitialized:

- Loading the block with reinitialization
- Loading snapshots as actual values
- Controlling or forcing the actual values
- The "WRIT_DBL" instruction

Before you execute these functions, take the following precautions:

- Make sure that the plant is in a safe state before you overwrite the actual values.
- Make sure that the IEC timer has expired before initializing its actual values.
- If you overwrite the actual values with a snapshot, make sure that the snapshot was taken at a time when the system was in a safe state.
- Make sure that the program does not read or write the affected data during transmission.

Parameters


The following table shows the parameters of the "Load time duration" instruction:

| Parameters | Declaration | Data type | Memory area | Description |
|-----------------|-------------|--|---------------------------|--|
| <Time duration> | Input | TIME, LTIME | I, Q, M, D, L or constant | Duration with which the IEC timer runs |
| <IEC timer> | Output | IEC_TIMER, IEC_LTIMER, TP_TIME, TP_LTIME, TON_TIME, TON_LTIME, TOF_TIME, TOF_LTIME, TONR_TIME, TONR_LTIME | D, L | IEC timer whose duration is set |

You can select the data type of the instruction from the "???" drop-down list.

Example

The following example shows how the instruction works:

| STL  | Explanation |
|---|---|
| CALL TON, "TON_DB" | // The instruction is called. The "TON_DB" instance data block is assigned to the instruction. // Select the required data type from the "???" drop-down list. |
| IN := "Tag_Start" | // The instruction executes on a rising signal edge of the operand "Tag_Start". |
| PT := "Tag_PresetTIME" | // The IEC timer stored in the instance data block "TON_DB" is started with the time duration that is specified by the operand "Tag_PresetTIME". |
| Q := "Tag_Output" | // The operand "Tag_Output" is set if the time duration PT specified by the operand "Tag_PresetTIME" has expired. // The parameter Q remains set as long as the "Tag_Start" operand still has the signal state "1". // When the signal state at the start input changes from "1" to "0", the operand of the Q parameter is reset. |
| ET := "Tag_ElapsedTIME" | // Current timer value |
| A "Tag_Input_1" | // When the operand "Tag_Input_1" and |
| A "Tag_Input_2" | // The operand "Tag_Input_2" returns the signal state "1", |
| CALL PRESET_TIMER | // The "Time accumulator" instruction is called. // Select the required data type from the "???" drop-down list. |
| PT := "Tag_PresetTIME_new" | // The instruction writes the time duration "Tag_PresetTIME_new" in the instance data block "TON_DB", thereby overwriting the time value of the operand "Tag_PresetTIME" within the |

instance data block. The signal state of the timer status may therefore change at the next query or when "MyTimer".Q or "MyTimer".ET are accessed.

```
TIMER := // IEC timer is reset.  
"TON_DB"
```

See also

Overview of the valid data types

Basic information on the status word (S7-1500)

Inserting STL instructions (S7-300, S7-400, S7-1500)

Editing STL instructions (S7-300, S7-400, S7-1500)

Instances

Querying and setting status bits in STL (S7-1500)

Memory areas (S7-1500)

1.3 Counter operations (S7-1500)

1.3.1 CTU: Count up (S7-1500)

Description

You can use the "Count up" instruction to increment the value at the CV parameter. When the signal state of the parameter CU changes from "0" to "1" (positive signal edge), the instruction is executed and the current counter value of the parameter CV is incremented by one. The counter value is incremented each time a positive signal edge is detected, until it reaches the high limit for the data type specified at the output CV. When the high limit is reached, the signal state of the CU parameter no longer has an effect on the instruction.

You can query the count status of the Q parameter. The signal state of the Q parameter is determined by the PV parameter. When the current counter value is greater than or equal to the value of the PV parameter, the Q parameter is set to signal state "1". In all other cases, the signal state of the Q parameter is "0".

The value of the CV parameter is reset to zero when the signal state at the R parameter changes to "1". As long as the signal state of the R parameter is "1", the signal state of the CU parameter has no effect on the instruction.

In the program code, you call the "Count up" instruction with the "Call block" (CALL) instruction.

Note

Only use a counter at a single point in the program to avoid the risk of counting errors.

Each call of the "Count up" instruction must be assigned an IEC counter in which the instruction data is stored. An IEC counter is a structure with one of the following data types:

The operating system resets the instances of the "Count up" instruction during a cold restart. If you want instances of the instruction to be initialized after a warm restart, call the instances to be initialized in a startup OB with the value "1" set for the R parameter of the instruction. If instances of the "Count up" instruction are located within another block, you can reset these instances, for example, by initializing the higher-level block.

Instance data block of system data type IEC_<Counter> (shared DB / single instance)

The system data type of the instance data block is derived from the data type of the instruction:

| Data type of the instruction | System data type of the instance data block (shared DB) |
|------------------------------|---|
| SINT / USINT | IEC_SCOUNTER / IEC_USCOUNTER |
| INT / UINT | IEC_COUNTER / IEC_UCOUNTER |
| DINT / UDINT | IEC_DCOUNTER / IEC_UDCOUNTER |
| LINT / ULINT | IEC_LCOUNTER / IEC_ULCOUNTER |

When you set up the IEC Counter in a single instance, the instance data block is created by default with "optimized block access" and the individual tags are defined as retentive.

For additional information on setting retentivity in an instance data block, refer to "See also".

Local tag (multi-instance)

The data type of the local tag is derived from the data type of the instruction:

| Data type of the instruction | Data type of the local tag |
|------------------------------|---|
| SINT / USINT | CTU_SINT / CTU_USINT / IEC_SCOUNTER / IEC_USCOUNTER |
| INT / UINT | CTU_INT / CTU_UINT / IEC_COUNTER / IEC_UCOUNTER |
| DINT / UDINT | CTU_DINT / CTU_UDINT / IEC_DCOUNTER / IEC_UDCOUNTER |
| LINT / ULINT | CTU_LINT / CTU_ULINT / IEC_LCOUNTER / IEC_ULCOUNTER |

When you set up the IEC Counter as multi-instance in a function block with "optimized block access", it is defined as retentive in the block interface.

Declaring IEC Counters

After the selection of the data type from the drop-down list "???", the "Call options" dialog opens. You can then declare an IEC Counter as follows:

- Single instance: Declaration of an instance data block of system data type IEC_<Counter> (for example, "MyIEC_COUNTER")

- Multi-instance: Declaration as a local tag of the type CTU_<Data type> or IEC_<Counter> in the "Static" section of a block (for example #MyCTU_COUNTER)


Parameters

The following table shows the parameters of the "Count up" instruction:

| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|-----------------------------|------------------------------|-------------------------------------|
| CU | Input | BOOL | I, Q, M, D, L or constant | Count input |
| R | Input | BOOL | I, Q, M, D, L, P or constant | Reset input |
| PV | Input | Integers | I, Q, M, D, L, P or constant | Value at which the Q output is set. |
| Q | Output | BOOL | I, Q, M, D, L | Counter status |
| CV | Output | Integers, CHAR, WCHAR, DATE | I, Q, M, D, L, P | Current counter value |

Example

The following example shows how the instruction works:

| STL  | Explanation |
|---|---|
| CALL CTU, "CTU_DB" | // The instruction is called. The "CTU_DB" data block is assigned to the instruction. // Select the required data type from the "???" drop-down list. |
| CU := "Tag_StartCTU" | // When the signal state of the "Tag_StartCTU" operand changes from "0" to "1", the instruction is executed and the current counter value of the "Tag_CounterValue" operand is incremented by one. // The counter value is incremented until the high limit of INT = 32767 is reached. |
| R := "Tag_ResetCounter" | // When the signal state of the "Tag_ResetCounter" operand changes to |

```

"1", the "Tag_CounterValue" operand is
reset to "0".

PV :=                                // The operand determines the value at
"Tag_PresetValue"                  which the operand of the Q parameter is
                                  set.

Q :=                                // The operand is set as long as the
"Tag_CounterStatus"               current counter value is greater than or
                                  equal to the value of the PV parameter.

CV :=                                // Current counter value
"Tag_CounterValue"

```

See also

Overview of the valid data types
 Setting retentivity in an instance data block
 Querying and setting status bits in STL (S7-1500)
 Memory areas (S7-1500)
 STL Basics (S7-300, S7-400, S7-1500)

1.3.2 CTD: Count down (S7-1500)

Description

The "Count down" instruction is used to decrement the value at the parameter CV. When the signal state of the CD parameter changes from "0" to "1" (positive signal edge), the instruction is executed and the current counter value of the CV parameter is decremented by one. Each time a positive signal edge is detected, the counter value is decremented until it reaches the low limit of the specified data type. When the low limit is reached, the signal state of the CD parameter no longer has an effect on the instruction.

You can query the count status of the Q parameter. If the current counter value is less than or equal to zero, the Q parameter is set to signal state "1". In all other cases, the signal state of the Q parameter is "0".

The value of the CV parameter is set to the value of the PV parameter when the signal state of the LD parameter changes to "1". As long as the signal state of the LD parameter is "1", the signal state of the CD parameter has no effect on the instruction.

In the program code, you call the "Count down" instruction with the "Call block" (CALL) instruction.

Note

Only use a counter at a single point in the program to avoid the risk of counting errors.

Each call of the "Count down" instruction must be assigned an IEC counter in which the instruction data is stored. An IEC counter is a structure with one of the following data types:

The operating system resets the instances of the "Count down" instruction during a cold restart. If you want instances of the instruction to be initialized after a warm restart, call the instances to be initialized with the value "1" for the LD parameter of the instruction in a startup OB. In this case the desired initial value for the CV parameter is specified in the PV parameter. If instances of

the "Count down" instruction are located within another block, you can reset these instances, for example, by initializing the higher-level block.

Instance data block of system data type IEC_<Counter> (shared DB / single instance)

The system data type of the instance data block is derived from the data type of the instruction:

| Data type of the instruction | System data type of the instance data block (shared DB) |
|------------------------------|---|
| SINT / USINT | IEC_SCOUNTER / IEC_USCOUNTER |
| INT / UINT | IEC_COUNTER / IEC_UCOUNTER |
| DINT / UDINT | IEC_DCOUNTER / IEC_UDCOUNTER |
| LINT / ULINT | IEC_LCOUNTER / IEC_ULCOUNTER |

When you set up the IEC Counter in a single instance, the instance data block is created by default with "optimized block access" and the individual tags are defined as retentive.

For additional information on setting retentivity in an instance data block, refer to "See also".

Local tag (multi-instance)

The data type of the local tag is derived from the data type of the instruction:

| Data type of the instruction | Data type of the local tag |
|------------------------------|---|
| SINT / USINT | CTD_SINT / CTD_USINT / IEC_SCOUNTER / IEC_USCOUNTER |
| INT / UINT | CTD_INT / CTD_UINT / IEC_COUNTER / IEC_UCOUNTER |
| DINT / UDINT | CTD_DINT / CTD_UDINT / IEC_DCOUNTER / IEC_UDCOUNTER |
| LINT / ULINT | CTD_LINT / CTD_ULINT / IEC_LCOUNTER / IEC_ULCOUNTER |

When you set up the IEC Counter as multi-instance in a function block with "optimized block access", it is defined as retentive in the block interface.

Declaring IEC Counters

After the selection of the data type from the drop-down list "???", the "Call options" dialog opens. You can then declare an IEC Counter as follows:

- Single instance: Declaration of an instance data block of system data type IEC_<Counter> (for example, "MyIEC_COUNTER")
- Multi-instance: Declaration as a local tag of the type CTD_<Data_type> or IEC_<Counter> in the "Static" section of a block (for example #MyCTD_COUNTER)


Parameters

The following table shows the parameters of the "Count down" instruction:

| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|-----------------------------|------------------------------|--|
| CD | Input | BOOL | I, Q, M, D, L or constant | Count input |
| LD | Input | BOOL | I, Q, M, D, L, P or constant | Load input |
| PV | Input | Integers | I, Q, M, D, L, P or constant | Value to which the CV output is set with LD = 1. |
| Q | Output | BOOL | I, Q, M, D, L | Counter status |
| CV | Output | Integers, CHAR, WCHAR, DATE | I, Q, M, D, L, P | Current counter value |

Example

The following example shows how the instruction works:

| STL  | Explanation |
|---|---|
| CALL CTD, "CTD_DB" | // The instruction is called. The "CTD_DB" data block is assigned to the instruction. // Select the required data type from the "???" drop-down list. |
| CD := "Tag_StartCTD" | // When the signal state of the "Tag_StartCTD" operand changes from "0" to "1", the instruction is executed and the current counter value of the "Tag_CounterValue" operand is decremented by one. // The counter value of the CV parameter is decremented until the low |

```

limit of INT = -32768 is reached.

LD := "Tag_LoadPV" // When the signal state of the
                    "Tag_LoadPV" operand changes to "1", the
                    "Tag_CounterValue" operand is set to the
                    value of the "Tag_PresetValue" operand.

PV :=              // Specifies the value to which the
"Tag_PresetValue"  counter is set when the signal state is
                    "1" in the LD parameter.

Q :=              // The operand is set when the current
"Tag_CounterStatus" counter value is less than or equal to
                    zero.

CV :=              // Current counter value
"Tag_CounterValue"

```

See also

Overview of the valid data types
 Setting retentivity in an instance data block
 Querying and setting status bits in STL (S7-1500)
 Memory areas (S7-1500)
 STL Basics (S7-300, S7-400, S7-1500)

1.3.3 CTUD: Count up and down (S7-1500)

Description

Use the "Count up and down" instruction to increment or decrement the counter value at the CV parameter. When the signal state at the CU parameter changes from "0" to "1" (positive signal edge), the current counter value is incremented by one and stored in the CV parameter. When the signal state of the CD parameter changes from "0" to "1" (positive signal edge), the counter value of the CV parameter is decremented by one. If there is a positive signal edge at the CU and CD inputs in a program cycle, the current counter value of the CV parameter remains unchanged.

The counter value can be incremented until it reaches the high limit of the data type specified at the CV parameter. When the high limit is reached, the counter value is no longer incremented on a positive signal edge. The counter value is no longer decremented once the low limit of the specified data type has been reached.

When the signal state of the LD parameter changes to "1", the counter value of the CV parameter is set to the value of the PV parameter. As long as the LD parameter has signal state "1", the signal state of the CU and CD inputs has no effect on the instruction.

The counter value is set to zero when the signal state of the R parameter changes to "1". As long as the R parameter has signal state "1", a change in the signal state of the CU, CD and LD parameters has no effect on the "Count up and down" instruction.

You can query the status of the up counter at the QU parameter. When the current counter value is greater than or equal to the value of the PV parameter, the QU parameter is set to signal state "1". In all other cases, the signal state of the QU parameter is "0". You can also specify a constant for the PV parameter.

You can query the status of the down counter at the QD parameter. If the current counter value is less than or equal to zero, the QD parameter is set to signal state "1". In all other cases, the signal state of the QD parameter is "0".

In the program code, you call the "Count up and down" instruction with the "Call block" (CALL) instruction.

Note

Only use a counter at a single point in the program to avoid the risk of counting errors.

Each call of the "Count up and down" instruction must be assigned an IEC counter in which the instruction data is stored. An IEC counter is a structure with one of the following data types:

The operating system resets the instances of the "Count up and down" instruction during a cold restart. If you want instances of the instruction to be initialized after a warm restart, you must call the instances to be initialized with the following parameter values in a startup OB:

- When used as up counter, the value at the R parameter must be set to "1".
- When used as down counter, the value at the LD parameter must be set to "1". In this case you specify the desired initial value for the CV parameter in the PV parameter.

If instances of the "Count up and down" instruction are located within another block, you can reset these instances, for example, by initializing the higher-level block.

Instance data block of system data type IEC_<Counter> (shared DB / single instance)

The system data type of the instance data block is derived from the data type of the instruction:

| Data type of the instruction | System data type of the instance data block (shared DB) |
|------------------------------|---|
| SINT / USINT | IEC_SCOUNTER / IEC_USCOUNTER |
| INT / UINT | IEC_COUNTER / IEC_UCOUNTER |
| DINT / UDINT | IEC_DCOUNTER / IEC_UDCOUNTER |
| LINT / ULINT | IEC_LCOUNTER / IEC_ULCOUNTER |

When you set up the IEC Counter in a single instance, the instance data block is created by default with "optimized block access" and the individual tags are defined as retentive.

For additional information on setting retentivity in an instance data block, refer to "See also".

Local tag (multi-instance)

The data type of the local tag is derived from the data type of the instruction:

| Data type of the instruction | Data type of the local tag |
|------------------------------|----------------------------|
|------------------------------|----------------------------|

| | |
|--------------|---|
| SINT / USINT | CTUD_SINT / CTUD_USINT / IEC_SCOUNTER / IEC_USCOUNTER |
| INT / UINT | CTUD_INT / CTUD_UINT / IEC_COUNTER / IEC_UCOUNTER |
| DINT / UDINT | CTUD_DINT / CTUD_UDINT / IEC_DCOUNTER / IEC_UDCOUNTER |
| LINT / ULINT | CTUD_LINT / CTUD_ULINT / IEC_LCOUNTER / IEC_ULCOUNTER |

When you set up the IEC Counter as multi-instance in a function block with "optimized block access", it is defined as retentive in the block interface.

Declaring IEC Counters

After the selection of the data type from the drop-down list "???", the "Call options" dialog opens. You can then declare an IEC Counter as follows:

- Single instance: Declaration of an instance data block of system data type IEC_<Counter> (for example, "MyIEC_COUNTER")
- Multi-instance: Declaration as a local tag of the type CTUD_<Data type> or IEC_<Counter> in the "Static" section of a block (for example #MyCTUD_COUNTER)

Parameters


The following table shows the parameters of the "Count up and down" instruction:

| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|-----------|------------------------------|--|
| CU | Input | BOOL | I, Q, M, D, L or constant | Count up input |
| CD | Input | BOOL | I, Q, M, D, L or constant | Count down input |
| R | Input | BOOL | I, Q, M, D, L, P or constant | Reset input |
| LD | Input | BOOL | I, Q, M, D, L, P or constant | Load input |
| PV | Input | Integers | I, Q, M, D, L, P or constant | Value at which the QU output is set / value to which the CV output is set when LD = 1. |
| QU | Output | BOOL | I, Q, M, D, L | Status of the up counter |

| | | | | |
|----|--------|-----------------------------|------------------|----------------------------|
| QD | Output | BOOL | I, Q, M, D, L | Status of the down counter |
| CV | Output | Integers, CHAR, WCHAR, DATE | I, Q, M, D, L, P | Current counter value |

Example

The following example shows how the instruction works:

| STL  | Explanation |
|---|--|
| CALL CTUD, "CTUD_DB" | <p>// The instruction is called. The "CTUD_DB" data block is assigned to the instruction.</p> <p>// Select the required data type from the "???" drop-down list.</p> |
| CU := "Tag_StartCTU" | <p>// When the signal state of the "Tag_StartCTU" operand changes from "0" to "1", the instruction is executed and the current counter value of the "Tag_CounterValue" operand is incremented by one.</p> <p>// The counter value is incremented on a rising signal edge of the CU parameter until it reaches the high limit of INT = 32767.</p> |
| CD := "Tag_StartCTD" | <p>// When the signal state of the "Tag_StartCTD" operand changes from "0" to "1", the instruction is executed and the current counter value of the "Tag_CounterValue" operand is decremented by one.</p> <p>// The counter value of the CV parameter is decremented until the low limit of -32768 is reached.</p> |
| R := "Tag_ResetCounter" | <p>// When the signal state of the "Tag_ResetCounter" operand changes to "1", the "Tag_CounterValue" operand is reset to "0".</p> |

```

LD := "Tag_LoadPV"    // When the signal state of the
                        "Tag_LoadPV" operand changes to "1", the
                        "Tag_CounterValue" operand is set to the
                        value of the "Tag_PresetValue" operand.

PV :=                  // Specifies the value to which the
"Tag_PresetValue"      counter is set when the signal state is
                        "1" in the LD parameter.

QU :=                  // The operand is set as long as the
"Tag_CounterStatus"    current counter value is greater than or
                        equal to the value of the PV parameter.

QD :=                  // The operand is set when the current
"Tag_CounterStatus"    counter value is less than or equal to
                        zero.

CV :=                  // Current counter value
"Tag_CounterValue"

```

See also

Overview of the valid data types
 Setting retentivity in an instance data block
 Querying and setting status bits in STL (S7-1500)
 Memory areas (S7-1500)
 STL Basics (S7-300, S7-400, S7-1500)

1.4 Comparator operations (S7-1500)

1.4.1 CompType: Compare tag structured data types (S7-1500)

Description

You can use the "Compare tag structured data types" instruction to determine if the first comparison value of a structured tag (IN1) is equal to or not equal to a second comparison value of another structured tag (IN2).

If the condition of the comparison is fulfilled, the instruction returns the result of logic operation (RLO) "1" at the OUT parameter. If the comparison condition is not fulfilled, the instruction returns RLO "0".

Comparison of floating-point numbers

When floating-point numbers are compared, the operands to be compared must have the same data type regardless of the setting for the IEC Check.

The special bit patterns of invalid floating-point numbers (NaN) that are the outcome of undefined results (e.g. root of -1) are not comparable. This means that if one of the operands has the value NaN, both the "CompType EQ" and "CompType NE" instructions return FALSE as the result.

Comparison of character strings

The individual characters are compared by means of their code (for example, 'a' is greater than 'A') during the comparison of the strings. The comparison is performed from left to right. The first character to be different decides the result of the comparison.

The following table shows examples of EQ string comparisons:

| <Operand1> | <Operand2> | RLO of the instruction |
|---------------|--------------|------------------------|
| 'AA' | 'AA' | 1 |
| 'Hello World' | 'HelloWorld' | 0 |
| 'AA' | 'aa' | 0 |
| 'aa' | 'aaa' | 0 |

The following table shows examples of NE string comparisons:

| <Operand1> | <Operand2> | RLO of the instruction |
|---------------|--------------|------------------------|
| 'AA' | 'aa' | 1 |
| 'Hello World' | 'HelloWorld' | 1 |
| 'AA' | 'AA' | 0 |
| 'aa' | 'aaa' | 1 |

You can also compare individual characters of a string. The number of the character to be compared is specified in square brackets next to the operand name. "MyString[2]", for example, compares the second character of the "MyString" string.

Comparison of timers, date and time

Bit patterns of invalid timers, date and times, e.g. DT#2015-13-33-25:62:99.999_999_999, cannot be compared. This means that if one of the operands has an invalid value, the instructions return the following results:

- "=: Equal" has the result FALSE.
- "<>: Not equal" has the result TRUE.

Not all times can be compared directly with each other, such as S5TIME. In this case they are implicitly converted into another time so that they can be compared, for example to TIME.

If you want to compare dates and times of different data types, the value of the smaller date or time data type is implicitly converted into the larger date or time data type. This means the two date and time data types DATE and DTL, for example, are compared on the basis of DTL.

When the implicit conversions fail, the comparison result is FALSE.

Comparison of structures

Note

Availability of comparison of structures

The option to compare structures is available for a CPU of the S7-1500 series as of firmware version ≥ 2.0 .

You can compare the values of two structured operands when both tags are of the same structure data type. When structures are compared, the operands to be compared must have the same data type regardless of the setting for the IEC Check. An exception is comparisons in which one of the two operands is a VARIANT or an ANY. If the data type is not yet known when the program is created, you can use the VARIANT data type. In this case you can also compare the operand with a structured tag of any data type. You can also compare two tags of the data type VARIANT or ANY with each other.

The following data types are possible:

- PLC data type
- STRUCT (the structure of the data type STRUCT must be contained in a PLC data type (UDT) or the two structures to be compared are two elements of an ARRAY of STRUCT. Anonymous structures are not permitted.)
- Tag to which ANY is pointing
- Tag to which VARIANT is pointing

The following requirements must be met to compare two tags of the data type ARRAY:

- The elements must each have the same data type.
- The two ARRAYS must have the same dimension.
- All dimensions must have the same number of elements. The exact ARRAY limits do not have to match.

The tables below show examples of a comparison of structures for "Equal":

| <Operand1> | | | <Operand2> | | | RLO of the instruction |
|---------------------------------------|------|-----------|---------------------------------------|------|-----------|------------------------|
| Tag of data type A <PLC data type> | | Tag value | Tag of data type A <PLC data type> | | Tag value | 1 |
| | BOOL | FALSE | | BOOL | FALSE | |
| | INT | 2 | | INT | 2 | |
| | | | | | | |
| <Operand1> | | | <Operand2> | | | RLO of the instruction |

| | | | | | | |
|------------------------------------|------|-----------|------------------------------------|------|-----------|---|
| Tag of data type A <PLC data type> | | Tag value | Tag of data type B <PLC data type> | | Tag value | 0 |
| | BOOL | FALSE | | BOOL | TRUE | |
| | INT | 2 | | INT | 3 | |

| | | | | | | |
|------------------------------------|------|-----------|--|------|-----------|------------------------|
| <Operand1> | | | <Operand2> | | | RLO of the instruction |
| Tag of data type A <PLC data type> | | Tag value | VARIANT (supplied with tag of data type A) | | Tag value | 1 |
| | BOOL | FALSE | | BOOL | FALSE | |
| | INT | 2 | | INT | 2 | |

The tables below show examples of a comparison of structures for "Not equal":

| | | | | | | |
|------------------------------------|------|-----------|------------------------------------|------|-----------|------------------------|
| <Operand1> | | | <Operand2> | | | RLO of the instruction |
| Tag of data type A <PLC data type> | | Tag value | Tag of data type A <PLC data type> | | Tag value | 0 |
| | BOOL | FALSE | | BOOL | FALSE | |
| | INT | 2 | | INT | 2 | |

| | | | | | | |
|------------------------------------|------|-----------|------------------------------------|------|-----------|------------------------|
| <Operand1> | | | <Operand2> | | | RLO of the instruction |
| Tag of data type A <PLC data type> | | Tag value | Tag of data type B <PLC data type> | | Tag value | 1 |
| | BOOL | FALSE | | BOOL | TRUE | |
| | INT | 2 | | INT | 3 | |

| | | | | | | |
|------------|--|--|------------|--|--|------------------------|
| <Operand1> | | | <Operand2> | | | RLO of the instruction |
|------------|--|--|------------|--|--|------------------------|

| | | | | | | |
|------------------------------------|------|-----------|--|------|-----------|---|
| Tag of data type A <PLC data type> | | Tag value | VARIANT (supplied with tag of data type A) | | Tag value | 0 |
| | BOOL | FALSE | | BOOL | FALSE | |
| | INT | 2 | | INT | 2 | |

Parameters


The following table shows the parameters of the instruction "Compare tag structured data types":

| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|--|------------------|-------------------------|
| IN1 | Input | Binary numbers, integers, floating-point numbers, character strings, timers, date and time, ARRAY of <data type> with fixed and variable ARRAY limits, STRUCT, VARIANT, ANY, PLC data type | I, Q, M, D, L, P | First comparison value |
| IN2 | Input | Binary numbers, integers, floating-point numbers, character strings, timers, date and time, ARRAY of <data type> with fixed and variable ARRAY limits, STRUCT, VARIANT, ANY, PLC data type | I, Q, M, D, L, P | Second value to compare |

| OUT | Output | BOOL | I, Q, M, D, L | Result of the instruction |
|---|--------|------|---------------|---------------------------|
| As detailed above, the data types ARRAY, STRUCT (in a PLC data type), VARIANT, ANY and PLC data type (UDT) are only available as of firmware version 2.0. | | | | |

Example

The following example shows how the instruction works:

| STL  | Explanation |
|---|--|
| CALL CompType | // The instruction is called. // Select the required function. Either "EQ" or "NE" from the drop-down list. |
| IN1 := "Tag_Operand1" | // First comparison value |
| IN2 := "Tag_Operand2" | // Second comparison value |
| OUT := "Tag_Result" | // The "Tag_Result" output returns the signal state "1" if the condition of the comparison instruction is fulfilled. i.e. the "Tag_Operand1" operand is equal to "Tag_Operand2". |

See also

[Overview of the valid data types](#)
[Basic information on VARIANT \(S7-1200, S7-1500\)](#)
[Querying and setting status bits in STL \(S7-1500\)](#)
[Memory areas \(S7-1500\)](#)
[STL Basics \(S7-300, S7-400, S7-1500\)](#)

1.4.2 VARIANT (S7-1500)

1.4.2.1 EQ_Type: Compare data type for EQUAL with the data type of a tag (S7-1500)

Description

You can use the "Compare data type for EQUAL with the data type of a tag" instruction to query the data type of a tag to which the VARIANT points. You are comparing the data type of the tag at IN1 parameter, which you declared in the block interface, with the data type of a tag at IN2 parameter for "Equal".

The tag at the IN1 parameter must have the VARIANT data type. The tag at the IN2 parameter can be an elementary data type or a PLC data type.

Comparison of timers, date and time

Not all times can be compared directly with each other, such as S5TIME. In this case they are implicitly converted into another time so that they can be compared, for example to TIME.

If you want to compare dates and times of different data types, the value of the smaller date or time data type is implicitly converted into the larger date or time data type. This means the two date and time data types DATE and DTL, for example, are compared on the basis of DTL.

When the implicit conversions fail, the comparison result is FALSE.

Comparison of structures

For the comparison of structures, you can use the instruction "CompType". Anonymous structures cannot generally be compared unless they are part of the same ARRAY.

For additional information on the comparison of structures, please refer to: CompType: Compare tag structured data types


Parameters

The following table shows the parameters of the "Compare data type for EQUAL with the data type of a tag" instruction:

| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|---|---|---------------------------|
| IN1 | Input | VARIANT | L (The declaration is possible in the "Input", "InOut" and "Temp" sections of the block interface.) | First operand |
| IN2 | Input | Binary numbers, integers, floating-point numbers, timers, date and time, character strings, ARRAY, PLC data types | I, Q, M, D, L, P | Second operand |
| RET_VAL | Output | BOOL | I, Q, M, D, L | Result of the instruction |

Example

The following example shows how the instruction works:

| | |
|---|-------------------------------|
| STL  | Explanation |
| CALL EQ_Type | // The instruction is called. |

```

IN1 :=          // First operand to be compared
#Tag_Operand1

IN2 :=          // Second operand to be compared
"Tag_Operand2"

RET_VAL :=      // The "Tag_Result" output returns the signal
"Tag_Result"    state "1" if the condition of the comparison
                 instruction is fulfilled. i.e. the
                 #Tag_Operand1 operand is not equal to
                 "Tag_Operand2".

```

See also

Overview of the valid data types
 Basic information on VARIANT (S7-1200, S7-1500)
 Querying and setting status bits in STL (S7-1500)
 Memory areas (S7-1500)
 STL Basics (S7-300, S7-400, S7-1500)

1.4.2.2 NE_Type: Compare data type for UNEQUAL with the data type of a tag (S7-1500)

Description

You can use the "Compare data type for UNEQUAL with the data type of a tag" instruction to query the data type which a tag does not have to which a VARIANT points. You are comparing the data type of the tag at IN1 parameter, which you declared in the block interface, with the data type of a tag at IN2 parameter for "Not equal".

The tag at the IN1 parameter must have the VARIANT data type. The tag at the IN2 parameter can be an elementary data type or a PLC data type.

Comparison of timers, date and time

Not all times can be compared directly with each other, such as S5TIME. In this case they are implicitly converted into another time so that they can be compared, for example to TIME.

If you want to compare dates and times of different data types, the value of the smaller date or time data type is implicitly converted into the larger date or time data type. This means the two date and time data types DATE and DTL, for example, are compared on the basis of DTL.

When the implicit conversions fail, the comparison result is FALSE.

Comparison of structures

For the comparison of structures, you can use the instruction "CompType". Anonymous structures cannot generally be compared unless they are part of the same ARRAY.

For additional information on the comparison of structures, please refer to: CompType: Compare tag structured data types


Parameters

The following table shows the parameters of the "Compare data type for UNEQUAL with the data type of a tag" instruction:

| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|---|---|---------------------------|
| IN1 | Input | VARIANT | L (The declaration is possible in the "Input", "InOut" and "Temp" sections of the block interface.) | First operand |
| IN2 | Input | Binary numbers, integers, floating-point numbers, timers, date and time, character strings, ARRAY, PLC data types | I, Q, M, D, L, P | Second operand |
| RET_VAL | Output | BOOL | I, Q, M, D, L | Result of the instruction |

Result

The following example shows how the instruction works:

| STL  | Explanation |
|---|---|
| CALL NE_Type | // The instruction is called. |
| IN1 := #Tag_Operand1 | // First operand to be compared |
| IN2 := "Tag_Operand2" | // Second operand to be compared |
| RET_VAL := "Tag_Result" | // The "Tag_Result" output returns the signal state "1" if the condition of the comparison instruction is fulfilled. i.e. the #Tag_Operand1 operand is not equal to "Tag_Operand2". |

See also

[Overview of the valid data types](#)
[Basic information on VARIANT \(S7-1200, S7-1500\)](#)
[Querying and setting status bits in STL \(S7-1500\)](#)
[Memory areas \(S7-1500\)](#)
[STL Basics \(S7-300, S7-400, S7-1500\)](#)

1.4.2.3 EQ_ElemType: Compare data type of an ARRAY element for EQUAL with the data type of a tag (S7-1500)

Description

You can use the "Compare data type of an ARRAY element for EQUAL with the data type of a tag" instruction to query the data type of a tag to which the VARIANT points. You are comparing the data type of the tag at IN1 parameter, which you declared in the block interface, with the data type of a tag at IN2 parameter for "Equal".

The tag at the IN1 parameter must have the VARIANT data type. The tag at the IN2 parameter can be an elementary data type or a PLC data type.

If the data type of the VARIANT tag is an ARRAY, the data type of the ARRAY elements is compared.

Comparison of timers, date and time

Not all times can be compared directly with each other, such as S5TIME. In this case they are implicitly converted into another time so that they can be compared, for example to TIME.

If you want to compare dates and times of different data types, the value of the smaller date or time data type is implicitly converted into the larger date or time data type. This means the two date and time data types DATE and DTL, for example, are compared on the basis of DTL.

When the implicit conversions fail, the comparison result is FALSE.

Comparison of structures

For the comparison of structures, you can use the instruction "CompType". Anonymous structures cannot generally be compared unless they are part of the same ARRAY.

For additional information on the comparison of structures, please refer to: CompType: Compare tag structured data types

Parameters


The following table shows the parameters of the "Compare data type of an ARRAY element for EQUAL with the data type of a tag" instruction:

| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|--|---|---------------|
| IN1 | Input | VARIANT | L (The declaration is possible in the "Input", "InOut" and "Temp" sections of the block interface.) | First operand |
| IN2 | Input | Binary numbers, integers, floating-point | | |

| | | | | |
|---------|--------|--|------------------|---------------------------|
| | | numbers, timers, date and time, character strings, ARRAY, PLC data types | I, Q, M, D, L, P | Second operand |
| RET_VAL | Output | BOOL | I, Q, M, D, L | Result of the instruction |

Result

The following example shows how the instruction works:

| STL  | Explanation |
|---|---|
| CALL EQ_ElemType | // The instruction is called. |
| IN1 := #Tag_Operand1 | // First operand to be compared |
| IN2 := "Tag_Operand2" | // Second operand to be compared |
| RET_VAL := "Tag_Result" | // The "Tag_Result" output returns the signal state "1" if the condition of the comparison instruction is fulfilled. i.e. the #Tag_Operand1 operand is not equal to "Tag_Operand2". |

See also

[Overview of the valid data types](#)
[Basic information on VARIANT \(S7-1200, S7-1500\)](#)
[Querying and setting status bits in STL \(S7-1500\)](#)
[Memory areas \(S7-1500\)](#)
[STL Basics \(S7-300, S7-400, S7-1500\)](#)

1.4.2.4 NE_ElemType: Compare data type of an ARRAY element for UNEQUAL with the data type of a tag (S7-1500)

Description

You can use the "Compare data type of an ARRAY element for UNEQUAL with the data type of a tag" instruction to query the data type which the tag does not have to which the VARIANT points. You are comparing the data type of the tag at IN1 parameter, which you declared in the block interface, with the data type of a tag at IN2 parameter for "Not equal".

The tag at the IN1 parameter must have the VARIANT data type. The tag at the IN2 parameter can be an elementary data type or a PLC data type.

If the data type of the VARIANT tag is an ARRAY, the data type of the ARRAY elements is compared.

Comparison of timers, date and time

Not all times can be compared directly with each other, such as S5TIME. In this case they are implicitly converted into another time so that they can be compared, for example to TIME.

If you want to compare dates and times of different data types, the value of the smaller date or time data type is implicitly converted into the larger date or time data type. This means the two date and time data types DATE and DTL, for example, are compared on the basis of DTL.

When the implicit conversions fail, the comparison result is FALSE.

Comparison of structures

For the comparison of structures, you can use the instruction "CompType". Anonymous structures cannot generally be compared unless they are part of the same ARRAY.

For additional information on the comparison of structures, please refer to: CompType: Compare tag structured data types


Parameters

The following table shows the parameters of the "Compare data type of an ARRAY element for UNEQUAL with the data type of a tag" instruction:

| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|---|---|---------------------------|
| IN1 | Input | VARIANT | L (The declaration is possible in the "Input", "InOut" and "Temp" sections of the block interface.) | First operand |
| IN2 | Input | Binary numbers, integers, floating-point numbers, timers, date and time, character strings, ARRAY, PLC data types | I, Q, M, D, L, P | Second operand |
| RET_VAL | Output | BOOL | I, Q, M, D, L | Result of the instruction |

Result

The following example shows how the instruction works:

| | |
|---|-------------------------------|
| STL  | Explanation |
| CALL NE_ElemType | // The instruction is called. |

```

IN1 :=          // First operand to be compared
#Tag_Operand1

IN2 :=          // Second operand to be compared
"Tag_Operand2"

RET_VAL :=      // The "Tag_Result" output returns the signal
"Tag_Result"    state "1" if the condition of the comparison
                 instruction is fulfilled. i.e. the
                 #Tag_Operand1 operand is not equal to
                 "Tag_Operand2".

```

See also

Overview of the valid data types
 Basic information on VARIANT (S7-1200, S7-1500)
 Querying and setting status bits in STL (S7-1500)
 Memory areas (S7-1500)
 STL Basics (S7-300, S7-400, S7-1500)

1.4.2.5 IS_NULL: Check for EQUALS NULL pointer (S7-1500)

Description

You can use the instruction "Check for EQUALS NULL pointer" to query whether the VARIANT or the reference points to a NULL pointer and therefore does not point to an object.

The tag at the parameter OPERAND must have the data type VARIANT or REF_TO <data type>.

Note

VARIANT tag points to an ANY pointer

If the VARIANT tag points to an ANY pointer, the instruction always returns the result RLO = "0" even if the ANY pointer is NULL.

Parameters

The following table shows the parameters of the "Check for EQUALS NULL pointer" instruction:


| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|-------------------------------------|---|--|
| OPERAND | Input | VARIANT or REF_TO <data type> | L (The declaration is possible in the "Input", "InOut" and "Temp" sections of the block interface.) | Operand that is compared for EQUALS NULL |

| | | | | |
|---------|--------|------|---------------|---------------------------|
| RET_VAL | Output | BOOL | I, Q, M, D, L | Result of the instruction |
|---------|--------|------|---------------|---------------------------|

You can find more information on the valid data types under "See also".

Example

The following example shows how the instruction works:

| | |
|---|---|
| STL  | Explanation |
| CALL IS_NULL | // The instruction is called. |
| OPERAND := #Tag_Operand | // Operand to be compared |
| RET_VAL := "Tag_Result" | // The "Tag_Result" output returns the signal state "1" if the condition of the comparison instruction is fulfilled. i.e. the #Tag_Operand operand does not point to an object. |

See also

[Overview of the valid data types](#)
[Basic information on VARIANT \(S7-1200, S7-1500\)](#)
[Querying and setting status bits in STL \(S7-1500\)](#)
[Memory areas \(S7-1500\)](#)
[STL Basics \(S7-300, S7-400, S7-1500\)](#)

1.4.2.6 NOT_NULL: Check for UNEQUALS NULL pointer (S7-1500)

Description

You can use the instruction "Check for UNEQUALS NULL pointer" to query whether the VARIANT or the reference does not point to a NULL pointer and therefore points to an object.

The tag at the parameter OPERAND must have the data type VARIANT or REF_TO <data type>.

Note

VARIANT tag points to an ANY pointer

If the VARIANT tag points to an ANY pointer, the instruction always returns the result RLO = "1" even if the ANY pointer is NULL.

Parameters

The following table shows the parameters of the "Check for UNEQUALS NULL pointer" instruction:


| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|-----------|-------------|-------------|
|------------|-------------|-----------|-------------|-------------|

| | | | | |
|---------|--------|----------------------------------|---|--|
| OPERAND | Input | VARIANT or REF_TO <data type> | L (The declaration is possible in the "Input", "InOut" and "Temp" sections of the block interface.) | Operand that is compared for UNEQUALS NULL |
| RET_VAL | Output | BOOL | I, Q, M, D, L | Result of the instruction |

For more information on valid data types, refer to "See also".

Example

The following example shows how the instruction works:

| | |
|---|---|
| STL  | Explanation |
| CALL NOT_NULL | // The instruction is called. |
| OPERAND := #Tag_Operand | // Operand to be compared |
| RET_VAL := "Tag_Result" | // The "Tag_Result" output returns the signal state "1" if the condition of the comparison instruction is fulfilled. i.e. the #Tag_Operand operand does not point to an object. |

See also

Overview of the valid data types
Basic information on VARIANT (S7-1200, S7-1500)
Querying and setting status bits in STL (S7-1500)
Memory areas (S7-1500)
STL Basics (S7-300, S7-400, S7-1500)

1.4.2.7 IS_ARRAY: Check for ARRAY (S7-1500)

Description

You can use the "Check for ARRAY" instruction to query whether the OPERAND parameter points to a tag of the ARRAY data type.

The tag at the OPERAND parameter has the VARIANT or ResolvedSymbol data type. The result of the query is output at the RET_VAL parameter.

Parameters

The following table shows the parameters of the "Check for ARRAY" instruction:

| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|-----------|-------------|-------------|
|------------|-------------|-----------|-------------|-------------|

| | | | | |
|---------|--------|----------------------------|---|-----------------------------------|
| OPERAND | Input | VARIANT, ResolvedSymbol | L (The declaration is possible in the "Input", "InOut" and "Temp" sections of the block interface.) | Operand that is queried for ARRAY |
| RET_VAL | Output | BOOL | I, Q, M, D, L | Result of the instruction |

For more information on valid data types, refer to "See also".


Note

Checking an ARRAY data block

If you use the IS_ARRAY instruction with an ArrayDB and generate the VARIANT input parameter via DB_ANY_TO_VARIANT, a symbolic use of the ArrayDB must be present elsewhere in the program as an actual parameter of a formal parameter of the data type VARIANT. To work correctly, it is sufficient if the point of use is downloaded. It is not necessary to execute it.

Example

The following example shows how the instruction works:

| STL  | Explanation |
|---|--|
| CALL IS_ARRAY | // The instruction is called. |
| OPERAND := #Tag_Input | // Operand to be queried at ARRAY |
| RET_VAL := "Tag_Result" | // The "Tag_Result" output returns the signal state "1" if the condition of the comparison instruction is fulfilled. // In other words, if the VARIANT at the #Tag_Input operand points to an ARRAY. |

See also

Overview of the valid data types
Basic information on VARIANT (S7-1200, S7-1500)
Querying and setting status bits in STL (S7-1500)
Memory areas (S7-1500)
STL Basics (S7-300, S7-400, S7-1500)

1.4.2.8 EQ_TypeOfDB: Compare data type of an indirectly addressed DB for EQUAL with a data type (S7-1500)

Description

The "Compare data type of an indirectly addressed DB for EQUAL with a data type" instruction is used to query which data type of the data block has that the tag of the DB_ANY data type addresses. You compare the data type of the DB addressed by the tag at the IN1 parameter either with the data type of another tag or directly with a data type at the IN2 parameter for "Equal".

The tag at the IN1 parameter must have the DB_ANY data type. The tag at the IN2 parameter, for example, can be a PLC data type, a system data type, an axis or an FB.

If the condition of the comparison is fulfilled, the instruction returns the result of logic operation (RLO) "1". If the comparison condition is not fulfilled, the instruction returns RLO "0".

Parameter


The following table shows the parameters of the instruction:

| Parameter | Declaration | Data type | Memory area | Description |
|-----------|-------------|-----------|---|----------------------|
| IN1 | Input | DB_ANY | L (The declaration is possible in the "Input", "InOut" and "Temp" sections of the block interface.) | First operand |
| IN2 | Input | TYPE_ID | I, Q, M, D, L, P | Second operand |
| RET_VAL | Output | BOOL | I, Q, M, D, L | Result of comparison |

You can find additional information on valid data types under "See also".

Example

The following example shows how the instruction works:

| STL  | Description |
|---|--|
| CALL EQ_TypeOfDB | // Call the instruction |
| IN1 := #InputDBAny | // Operand to query for DB_ANY |
| IN2 := TO_SpeedAxis | // The condition of the comparison instruction is fulfilled if the data type of the #InputDBAny operand addressed DB is equal with the data type TO_SpeedAxis. |

```
RET_VAL := "TagOut"           // Result of comparison
```

The "TagOut" output is not set when the following conditions are fulfilled:

- The number of the data block is "0".
- The data block does not exist.
- The data block is an ARRAY DB.
- The data block contains a tag of the data type UDT (PLC data type).

See also

Overview of the valid data types
Querying and setting status bits in STL (S7-1500)
Memory areas (S7-1500)
STL Basics (S7-300, S7-400, S7-1500)
Using the DB_ANY data type (S7-1200, S7-1500)

1.4.2.9 NE_TypeOfDB: Compare data type of an indirectly addressed DB for UNEQUAL with a data type (S7-1500)

Description

The "Compare data type of an indirectly addressed DB for UNEQUAL with a data type" instruction is used to query which data type the data block does not have, that the tag of the DB_ANY data type addresses. You compare the data type of the DB addressed by the tag at the IN1 parameter either with the data type of another tag or directly with the data type at the IN2 parameter for "Not equal".

The tag at the IN1 parameter must have the DB_ANY data type. The tag at the IN2 parameter, for example, can be a PLC data type, a system data type, an axis or an FB.

If the condition of the comparison is fulfilled, the instruction returns the result of logic operation (RLO) "1". If the comparison condition is not fulfilled, the instruction returns RLO "0".

Parameter

The following table shows the parameters of the instruction:


| Parameter | Declaration | Data type | Memory area | Description |
|-----------|-------------|-----------|---|----------------|
| IN1 | Input | DB_ANY | L (The declaration is possible in the "Input", "InOut" and "Temp" sections of the block interface.) | First operand |
| IN2 | Input | TYPE_ID | I, Q, M, D, L, P | Second operand |

| | | | | |
|---------|--------|------|---------------|----------------------|
| RET_VAL | Output | BOOL | I, Q, M, D, L | Result of comparison |
|---------|--------|------|---------------|----------------------|

You can find additional information on valid data types under "See also".

Example

The following example shows how the instruction works:

| STL  | Description |
|---|--|
| CALL NE_TypeOfDB | // Call the instruction |
| IN1 := #InputDBAny | // Operand to query for DB_ANY |
| IN2 := TO_SpeedAxis | // The condition of the comparison instruction is fulfilled if the data type of the #InputDBAny operand addressed DB is unequal with the data type TO_SpeedAxis. |
| RET_VAL := "TagOut" | // Result of comparison |

The "TagOut" output is not set when the following conditions are fulfilled:

- The number of the data block is "0".
- The data block does not exist.
- The data block is an ARRAY DB.
- The data block contains a tag of the data type UDT (PLC data type).

See also

Overview of the valid data types
 Querying and setting status bits in STL (S7-1500)
 Memory areas (S7-1500)
 STL Basics (S7-300, S7-400, S7-1500)
 Using the DB_ANY data type (S7-1200, S7-1500)

1.5 Math functions (S7-1500)

1.5.1 MIN: Get minimum (S7-1500)

Description

The "Get minimum" instruction compares the values at the inputs IN1, IN2 and IN3 and writes the lowest value to the OUT output. The instruction is only executed if the tags of all inputs are of the same data type.

The value of the OUT parameter is invalid if one of the following conditions is met:

- The specified tags are not of the same data type.

- A floating-point number has an invalid value.

Parameters


The following table shows the parameters of the "Get minimum" instruction:

| Parameters | Declaration | Data type | Memory area | Description |
|---|-------------|----------------------------------|------------------------------|--------------------|
| IN1 | Input | Integers, floating-point numbers | I, Q, M, D, L, P or constant | First input value |
| IN2 | Input | Integers, floating-point numbers | I, Q, M, D, L, P or constant | Second input value |
| IN3 | Input | Integers, floating-point numbers | I, Q, M, D, L, P or constant | Third input value |
| OUT | Output | Integers, floating-point numbers | I, Q, M, D, L, P | Result |
| If the IEC check is not activated, you can also use tags with the data types TIME, LTIME, TOD, LTOD, DATE and LDT by selecting a bit string or integer with the same length as the data type of the instruction (e.g. instead of TIME => DINT, UDINT or DWORD = 32 bits). | | | | |

You can select the data type for the parameters INn and OUT from the "???" drop-down list.

Example

The following example shows how the instruction works:

| STL  | Explanation |
|---|--|
| CALL MIN | // The instruction is called. // Select the required data type from the "???" drop-down list. |
| IN1 := "TagIn_Value1" | // First input value is compared. |
| IN2 := "TagIn_Value2" | // Second input value to be compared. |
| IN3 := "TagIn_Value3" | // Third input value to be compared. |
| OUT := "Tag_Minimum" | The "Tag_Minimum" operand is written with the value of the "TagIn_Value1" operand because this has the lowest value. |

The following table shows how the instruction works using specific operand values:

| Parameters | Operand | Value |
|------------|--------------|-------|
| IN1 | TagIn_Value1 | 12222 |
| IN2 | TagIn_Value2 | 14444 |
| IN3 | TagIn_Value3 | 13333 |
| OUT | Tag_Minimum | 12222 |

See also

Overview of the valid data types
Querying and setting status bits in STL (S7-1500)
Memory areas (S7-1500)
STL Basics (S7-300, S7-400, S7-1500)

1.5.2 MAX: Get maximum (S7-1500)

Description

The "Get maximum" instruction compares the values at the inputs IN1, IN2 and IN3 and writes the highest value to the OUT output. The instruction is only executed if the tags of all inputs are of the same data type.

The value of the OUT output is invalid if one of the following conditions is fulfilled:

- The specified tags are not of the same data type.
- A floating-point number has an invalid value.

Parameters

The following table shows the parameters of the "Get maximum" instruction:


| Parameters | Declaration | Data type | Memory area | Description |
|------------|-------------|----------------------------------|------------------------------|--------------------|
| IN1 | Input | Integers, floating-point numbers | I, Q, M, D, L, P or constant | First input value |
| IN2 | Input | Integers, floating-point numbers | I, Q, M, D, L, P or constant | Second input value |
| IN3 | Input | Integers, floating-point numbers | I, Q, M, D, L, P or constant | Third input value |

| | | | | |
|---|--------|----------------------------------|------------------|--------|
| OUT | Output | Integers, floating-point numbers | I, Q, M, D, L, P | Result |
| If the IEC check is not activated, you can also use tags with the data types TIME, LTIME, TOD, LTOD, DATE and LDT by selecting a bit string or integer with the same length as the data type of the instruction (e.g. instead of TIME => DINT, UDINT or DWORD = 32 bits). | | | | |

You can select the data type for the parameters INn and OUT from the "???" drop-down list.

Example

The following example shows how the instruction works:

| | |
|---|--|
| STL  | Explanation |
| CALL MAX | // The instruction is called. |
| | // Select the required data type from the "???" drop-down list. |
| IN1 := "TagIn_Value1" | // First input value is compared. |
| IN2 := "TagIn_Value2" | // Second input value to be compared. |
| IN3 := "TagIn_Value3" | // Third input value to be compared. |
| OUT := "Tag_Maximum" | // The "Tag_Maximum" operand is written with the value of the "TagIn_Value2" operand because this has the highest value. |

The following table shows how the instruction works using specific operand values:

| Parameters | Operand | Value |
|------------|--------------|-------|
| IN1 | TagIn_Value1 | 12222 |
| IN2 | TagIn_Value2 | 14444 |
| IN3 | TagIn_Value3 | 13333 |
| OUT | Tag_Maximum | 14444 |

See also

[Overview of the valid data types](#)
[Querying and setting status bits in STL \(S7-1500\)](#)
[Memory areas \(S7-1500\)](#)
[STL Basics \(S7-300, S7-400, S7-1500\)](#)