Crimson® 3.1 Funktionsblock

Referenzhandbuch | July 2020

LP1046 | Revision B

COPYRIGHT

SOFTWARE LICENSE

Red Lion Controls, Inc.
20 Willow Springs Circle
York, PA 17406

KONTAKTINFORMATIONEN:

AMERIKAS

In den USA: +1 (877) 432-9908
Außerhalb der USA: +1 (717) 767-6511
Stunden: 8 am-6 pm Östliche Standardzeit
(UTC/GMT -5 stunden)

ASIEN-PAZIFIK

Shanghai, V.R. China: +86 21-6113-3688 x767
Stunden: 9 am-6 pm China Standardzeit
(UTC/GMT +8 stunden)

EUROPA

Niederlande: +31 33-4723-225
Frankreich: +33 (0) 1 84 88 75 25
Deutschland: +49 (0) 1 89 5795-9421
GROßBRITANNIEN: +44 (0) 20 3868 0909
Stunden: 9 am-5 pm Mitteleuropäische Zeit
(UTC/GMT +1 stunde)

Webseite: www.redlion.net
Unterstützung: support.redlion.net

Inhaltsverzeichnis

Vorwort

Haftungsausschluss

Dieses Handbuch enthält spezifische Informationen zu den verschiedenen Funktionsblöcken und Operatoren, die beim Schreiben von Steuerungsprogrammen in der Steuerkategorie in Crimson® zur Verfügung stehen.

Obwohl alle Anstrengungen unternommen wurden, um sicherzustellen, dass dieses Dokument zum Zeitpunkt der Veröffentlichung vollständig und exakt ist, sind Änderungen an den darin enthaltenen Informationen vorbehalten. Red Lion Controls, Inc. ist nicht verantwortlich für Ergänzungen oder Änderungen des Originaldokuments. Industrielle Netzwerke variieren stark in der Konfiguration, Topologie und den Verkehrsbedingungen. Dieses Dokument ist nur als allgemeine Anleitung gedacht. Es wurde nicht auf alle möglichen Anwendungen getestet und es kann in manchen Situationen nicht vollständig oder genau sein.

Dieses Handbuch ist für Mitarbeiter gedacht, die für die Konfiguration und Inbetriebnahme von Crimson-Geräten für die Verwendung in Visualisierungs-, Überwachungs- und Steuerungsanwendungen verantwortlich sind. Benutzer dieses Dokuments sind aufgefordert, die Warnungen und Vorsichtsmaßnahmen im gesamten Dokument zu beachten.

Markenhinweise

Red Lion Controls, Inc. erkennt das Eigentum der folgenden in diesem Dokument verwendeten geschützten Begriffe an.

- Microsoft®, Windows® und Windows 7™ sind eingetragene Marken oder Marken der Microsoft Corporation in den USA und/oder anderen Ländern.

Alle anderen Firmen- und Produktnamen sind Marken der jeweiligen Eigentümer.

Dokumentverlauf und zugehörige Veröffentlichungen

Die Versionen dieses Dokuments auf Papier und elektronischen Medien werden nur bei der Veröffentlichung von Hauptversionen revidiert und enthalten deshalb möglicherweise nicht die neuesten Produktinformationen. Gegebenenfalls werden Technische Hinweise und/oder Produktzusätze mit den neuen Informationen oder Dokumentänderungen zwischen Hauptversionen zur Verfügung gestellt.

Die neueste Online-Version dieses Dokuments kann über die Red Lion-Website unter https://www.redlion.net/red-lion-software/crimson/crimson-31 aufgerufen werden.

Zusätzliche Produktinformationen

Zusätzliche Produktinformationen können von Ihrem lokalen Außendienstmitarbeiter oder von Red Lion über die auf der Innenseite der Vorderabdeckung aufgelisteten Kontaktnummern und/oder E-Mail-Adressen angefordert werden.

Kapitel 1 Einführung

Crimson® 3.1 ist die neueste Version der weithin anerkannten Crimson-Gerätekonfigurationssoftware von Red Lion. Dieses Funktionsblock-Referenzhandbuch ergänzt das Crimson 3.1 Benutzerhandbuch und das Crimson 3.1-Referenzhandbuch durch die Beschreibung der verschiedenen Funktionen und Vorgänge, die beim Schreiben von IEC 61131-3-Steuerungsprogrammen in der Steuerkategorie von Crimson 3.1 zur Verfügung stehen. Für jedes Element werden Informationen zu den Eingängen, dem Ausgang und zur Bedienung des Elements bereitgestellt.

Systemanforderungen

Crimson 3.1 ist für die Ausführung unter einer beliebigen Version von Microsoft Windows ab Windows 7 ausgelegt. Die Speicheranforderungen sind gering, und auf allen Systemen, die die Mindestsystemanforderungen für das Betriebssystem erfüllen, kann Crimson 3.1 ausgeführt werden. Für die Installation sind etwa 600 MB freier Speicherplatz erforderlich, und Sie sollten idealerweise einen Bildschirm mit ausreichender Auflösung haben, um die Bearbeitung von Bildschirmseiten ohne Scrollen zu ermöglichen.

Überprüfen auf Updates

Wenn Sie über eine Internetverbindung verfügen, können Sie mit dem Befehl „Auf Update überprüfen“ im Menü „Hilfe“ auf der Red Lion-Website nach einer neuen Version von Crimson 3.1 suchen. Wenn eine neuere Version als die von Ihnen verwendete Version gefunden wird, werden Sie von Crimson gefragt, ob Sie das Upgrade herunterladen und die Software automatisch aktualisieren möchten. Sie können das Upgrade auch manuell von der Red Lion-Website herunterladen, indem Sie die Seite „Downloads“ im Bereich „Support“ aufrufen.

Hilfe und Unterstützung

Wenn Sie auf ein Problem stoßen oder Hilfe benötigen, sind die folgenden Ressourcen verfügbar.

Technischer Support

Technische Unterstützung erhalten Sie im Internet unter: support.redlion.net
Sie können auch anrufen:
In den USA: +1 (877) 432-9908
Außerhalb der USA: +1 (717) 767-6511

Online-Foren

Es gibt eine Reihe von Online-Foren zur Unterstützung von Benutzern von SPS und HMIs. Red Lion empfiehlt das Q&A-Forum unter http://www.plctalk.net/qanda/. Das Diskussionsforum ist mit vielen hilfsbereiten Experten bevölkert, und die technischen Support-Mitarbeiter von Red Lion überwachen dieses Forum auf Fragen zu unseren Produkten.

Kapitel 2 Funktionsblöcke und Operatoren

In diesem Kapitel werden die verschiedenen Funktionsblöcke und Operatoren beschrieben, die beim Schreiben von IEC 61131-3-Steuerungsprogrammen in der Steuerkategorie von Crimson 3.1 zur Verfügung stehen. Für jedes Element werden Informationen zu den Eingängen, dem Ausgang und zur Bedienung des Elements bereitgestellt.

ABS

Funktion – Gibt den absoluten Wert des Eingangs zurück.

Eingänge

IN : ANY Wert.

Ausgänge

Q : ANY Ergebnis: Absoluter Wert von IN.

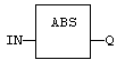
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

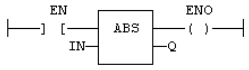
```
Q := ABS (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD    IN
        ABS
        ST    Q (* Q ist: ABS (IN) *)
```

Siehe auch

TRUNC LOG POW SQRT

ACOS ACOSL

Funktion – Berechnet einen Arcuscosinus.

Eingänge

IN : REAL/LREAL Reeller Zahlenwert.

Ausgänge

Q : REAL/LREAL Ergebnis: Arcuscosinus von IN.

Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

```
Q := ACOS (IN);
```

FBD-Sprache

KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.

AWL-Sprache

```
Op1:    LD      IN
        ACOS
        ST      Q (* Q ist: ACOS (IN) *)
```

Siehe auch

SIN COS TAN ASIN ATAN ATAN2

+ADD

Operator – Addiert alle Eingänge.

Eingänge

IN1 : ANY Erster Eingang.
IN2 : ANY Zweiter Eingang.

Ausgänge

Q : ANY Ergebnis: IN1 + IN2.

Anmerkungen

Alle Eingänge und der Ausgang müssen vom gleichen Typ sein. In der FBD-Sprache kann der Block bis zu 16 Eingänge haben. In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den gleichen Wert bei wie die Eingangsstufe. In der AWL-Sprache addiert der ADD-Befehl das aktuelle Ergebnis und den Operanden. Das aktuelle Ergebnis und der Operand müssen vom gleichen Typ sein.

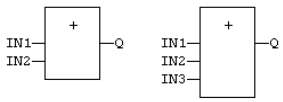
Die Addition kann mit Zeichenfolgen verwendet werden. Das Ergebnis ist die Verkettung der Eingangszeichenfolgen.

ST-Sprache

```
Q := IN1 + IN2;
MyString := 'Ha' + 'll ' + 'o'; (* MyString ist gleich 'Hallo' *)
```

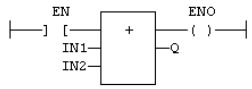
FBD-Sprache

Der Block kann bis zu 16 Eingänge haben:



KOP-Sprache

Die Addition wird nur ausgeführt, wenn EN *TRUE* ist.
ENO ist gleich EN.



AWL-Sprache

```
Op1:    LD    IN1
        ADD   IN2
        ST    Q (* Q ist gleich: IN1 + IN2 *)
Op2:    LD    IN1
        ADD   IN2
        ADD   IN3
        ST    Q (* Q ist gleich: IN1 + IN2 + IN3 *)
```

Siehe auch

- SUB * MUL / DIV

ALARM_A

Funktionsblock – Alarm mit automatischer Rückstellung.

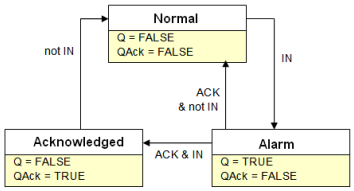
Eingänge

IN : BOOL Prozesssignal.
ACK : BOOL Bestätigungsbefehl.

Ausgänge

Q : BOOL *TRUE*, wenn Alarm aktiv ist.
QACK : BOOL *TRUE*, wenn Alarm bestätigt wurde.

Sequenz



Anmerkungen

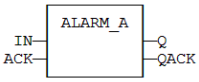
Kombinieren Sie diesen Block mit dem Block LIM_ALRM, um analoge Alarme zu verwalten.

ST-Sprache

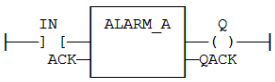
MyALARM wird als Instanz des Funktionsblocks ALARM_A deklariert.

```
MyALARM (IN, ACK, RST);
Q := MyALARM.Q;
QACK := MyALARM.QACK;
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

MyALARM wird als Instanz des Funktionsblocks ALARM_A deklariert.

```
Op1:    CAL   MyALARM (IN, ACK, RST)
        LD    MyALARM.Q
        ST    Q
        LD    MyALARM.QACK
        ST    QACK
```

Siehe auch

ALARM_MLIM_ALRM

ALARM_M

Funktionsblock – Alarm mit manueller Rückstellung.

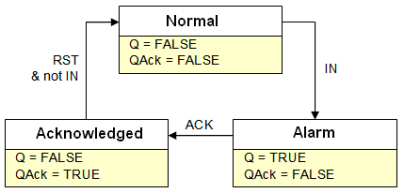
Eingänge

IN : BOOL Prozesssignal.
ACK : BOOL Bestätigungsbefehl.
RST : BOOL Rückstellungsbefehl.

Ausgänge

Q : BOOL *TRUE*, wenn Alarm aktiv ist.
QACK : BOOL *TRUE*, wenn Alarm bestätigt wurde.

Sequenz



Anmerkungen

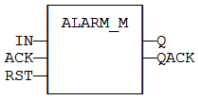
Kombinieren Sie diesen Block mit dem Block LIM_ALRM, um analoge Alarme zu verwalten.

ST-Sprache

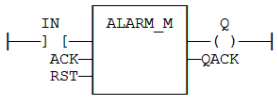
MyALARM wird als Instanz des Funktionsblocks ALARM_M deklariert.

```
MyALARM (IN, ACK, RST);
Q := MyALARM.Q;
QACK := MyALARM.QACK;
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

MyALARM wird als Instanz des Funktionsblocks ALARM_M deklariert.

```
Op1:    CALMyALARM (IN, ACK, RST)
LD      MyALARM.Q
ST      Q
LD      MyALARM.QACK
ST      QACK
```

Siehe auch

ALARM_A LIM_ALRM

AND ANDN &

Operator – Führt ein logisches UND aller Eingänge aus.

Eingänge

IN1 : BOOL Erster boolescher Eingang.
IN2 : BOOL Zweiter boolescher Eingang.

Ausgänge

Q : BOOL Boolesches AND aller Eingänge.

Wahrheitstabelle

IN1	IN2	Q
0	0	0
0	1	0
1	0	0
1	1	1

Anmerkungen

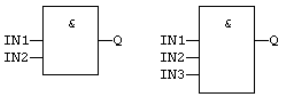
In der FBD-Sprache kann der Block bis zu 16 Eingänge haben. Der Block wird in FBD-Sprache als „&“ bezeichnet. In der KOP-Sprache wird eine AND-Operation durch serialisierte Kontakte dargestellt. In AWL-Sprache führt die AND-Anweisung ein logisches UND zwischen dem aktuellen Ergebnis und dem Operanden aus. Das aktuelle Ergebnis muss ein boolescher Wert sein. Der ANDN-Befehl führt ein UND zwischen dem aktuellen Ergebnis und der booleschen Negation des Operanden aus. In ST- und AWL-Sprachen kann „&“ anstelle von „AND“ verwendet werden.

ST-Sprache

```
Q := IN1 AND IN2;
Q := IN1 & IN2 & IN3;
```

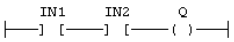
FBD-Sprache

Der Block kann bis zu 16 Eingänge haben:



KOP-Sprache

Serialisierte Kontakte:



AWL-Sprache

```
Op1:    LD      IN1
        &       IN2 (* „&“ oder „AND“ kann verwendet werden *)
        ST      Q(* Q ist gleich: IN1 AND IN2 *)
Op2:    LD      IN1
        AND     IN2
        &N      IN3(* „&N“ oder „ANDN“ kann verwendet werden *)
        ST      Q(* Q ist gleich: IN1 AND IN2 AND (NOT IN3) *)
```

Siehe auch

OR XOR NOT

AND_MASK

Funktion – Führt ein Bit-zu-Bit-AND zwischen zwei Ganzzahlwerten aus

Eingänge

IN : ANY Erster Eingang.
MSK : ANY Zweiter Eingang (AND-Maske).

Ausgänge

Q : ANY AND-Maske zwischen IN- und MSK-Eingängen.

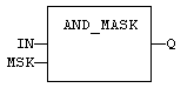
Anmerkungen

Argumente können aus Ganzzahlen mit oder ohne Vorzeichen von 8 bis 32 Bits bestehen.
In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den gleichen Wert bei wie die Eingangsstufe. In AWL-Sprache muss der erste Parameter (IN) in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann. Der andere Eingang sind die Operanden der Funktion.

ST-Sprache

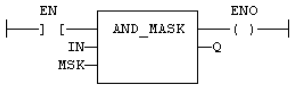
```
Q := AND_MASK (IN, MSK);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO ist gleich EN.



AWL-Sprache

```
Op1:    LD          IN
        AND_MASKMSK
        ST          Q
```

Siehe auch

OR_MASKXOR_MASKNOT_MASK

ANY_TO_BOOL

Operator – Konvertiert den Eingang in einen booleschen Wert.

Eingänge

IN : ANY Eingangswert.

Ausgänge

Q : BOOL Wert in booleschen Wert konvertiert.

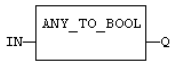
Anmerkungen

Bei den Eingangsdatentypen DINT, REAL und TIME lautet das Ergebnis *FALSE*, wenn der Eingang 0 ist. In allen anderen Fällen ist das Ergebnis *TRUE*. Bei STRING-Eingängen ist der Ausgang *TRUE*, wenn die Eingangszeichenfolge nicht leer ist, und *FALSE*, wenn die Zeichenfolge leer ist. In der KOP-Sprache wird die Konvertierung nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe ist das Ergebnis der Konvertierung. In AWL-Sprache konvertiert die Funktion ANY_TO_BOOL das aktuelle Ergebnis.

ST-Sprache

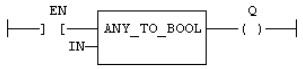
```
Q := ANY_TO_BOOL (IN);
```

FBD-Sprache



KOP-Sprache

Die Konvertierung wird nur ausgeführt, wenn EN *TRUE* ist.
Die Ausgangsstufe ist das Ergebnis der Konvertierung.
Die Ausgangsstufe ist *FALSE*, wenn EN *FALSE* ist.



AWL-Sprache

```
Op1:    LD              IN
        ANY_TO_BOOL
        ST              Q
```

Siehe auch

ANY_TO_SINT ANY_TO_INT ANY_TO_DINT ANY_TO_LINT
ANY_TO_REAL ANY_TO_LREAL ANY_TO_TIME ANY_TO_STRING

ANY_TO_DINT / ANY_TO_UINT

Operator – Konvertiert den Eingang in einen Ganzzahlwert.

Eingänge

IN : ANY Eingangswert.

Ausgänge

Q : DINT Wert in Ganzzahl konvertiert.

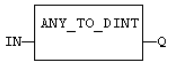
Anmerkungen

Bei BOOL-Eingangsdatentypen ist der Ausgang 0 oder 1. Für den Eingangsdatentyp REAL ist der Ausgang der ganzzahlige Teil des reellen Eingangs-Zahlenwerts. Beim Eingangsdatentyp TIME entspricht das Ergebnis der Anzahl der Millisekunden. Bei STRING-Eingängen ist der Ausgang die Zahl, die durch die Zeichenfolge dargestellt wird, oder 0, wenn die Zeichenfolge keine gültige Zahl darstellt. In der KOP-Sprache wird die Konvertierung nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL-Sprache konvertiert die Funktion ANY_TO_DINT das aktuelle Ergebnis.

ST-Sprache

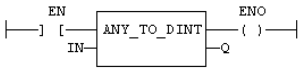
```
Q := ANY_TO_DINT (IN);
```

FBD-Sprache



KOP-Sprache

Die Konvertierung wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD          IN
        ANY_TO_DINT
        ST          Q
```

Siehe auch

ANY_TO_BOOL ANY_TO_SINT ANY_TO_INT ANY_TO_LINT
ANY_TO_REAL ANY_TO_LREAL ANY_TO_TIME ANY_TO_STRING

ANY_TO_INT / ANY_TO UINT

Operator – Konvertiert den Eingang in einen 16-Bit-Ganzzahlwert.

Eingänge

IN : ANY Eingangswert.

Ausgänge

Q : INT Wert in 16-Bit-Ganzzahl konvertiert.

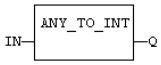
Anmerkungen

Bei BOOL-Eingangsdatentypen ist der Ausgang 0 oder 1. Für den Eingangsdatentyp REAL ist der Ausgang der ganzzahlige Teil des reellen Eingangs-Zahlenwerts. Beim Eingangsdatentyp TIME entspricht das Ergebnis der Anzahl der Millisekunden. Bei STRING-Eingängen ist der Ausgang die Zahl, die durch die Zeichenfolge dargestellt wird, oder 0, wenn die Zeichenfolge keine gültige Zahl darstellt. In der KOP-Sprache wird die Konvertierung nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL-Sprache konvertiert die Funktion ANY_TO_INT das aktuelle Ergebnis.

ST-Sprache

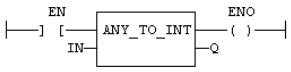
```
Q := ANY_TO_INT (IN);
```

FBD-Sprache



KOP-Sprache

Die Konvertierung wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD          IN
        ANY_TO_INT
        ST          Q
```

Siehe auch

ANY_TO_BOOL ANY_TO_SINT ANY_TO_DINT ANY_TO_LINT ANY_TO_REAL
ANY_TO_LREAL ANY_TO_TIME ANY_TO_STRING

ANY_TO_LINT

Operator – Konvertiert den Eingang in einen langen (64-Bit) Ganzzahlwert.

Eingänge

IN : ANY Eingangswert.

Ausgänge

Q : LINT Wert in lange (64-Bit) Ganzzahl konvertiert.

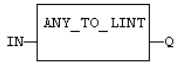
Anmerkungen

Bei BOOL-Eingangsdatentypen ist der Ausgang 0 oder 1. Für den Eingangsdatentyp REAL ist der Ausgang der ganzzahlige Teil des reellen Eingangs-Zahlenwerts. Beim Eingangsdatentyp TIME entspricht das Ergebnis der Anzahl der Millisekunden. Bei STRING-Eingängen ist der Ausgang die Zahl, die durch die Zeichenfolge dargestellt wird, oder 0, wenn die Zeichenfolge keine gültige Zahl darstellt. In der KOP-Sprache wird die Konvertierung nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL-Sprache konvertiert die Funktion ANY_TO_LINT das aktuelle Ergebnis.

ST-Sprache

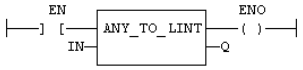
```
Q := ANY_TO_LINT (IN);
```

FBD-Sprache



KOP-Sprache

Die Konvertierung wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD          IN
        ANY_TO_LINT
        ST          Q
```

Siehe auch

ANY_TO_BOOL ANY_TO_SINT ANY_TO_INT ANY_TO_DINT ANY_TO_REAL ANY_TO_LREAL
ANY_TO_TIME ANY_TO_STRING

ANY_TO_LREAL

Operator – Konvertiert den Eingang in einen reellen Zahlenwert mit doppelter Genauigkeit.

Eingänge

IN : ANY Eingangswert.

Ausgänge

Q : LREAL Wert in reellen Zahlenwert mit doppelter Genauigkeit konvertiert.

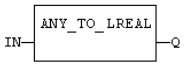
Anmerkungen

Bei BOOL-Eingangsdatentypen ist der Ausgang 0,0 oder 1,0. Bei DINT-Eingangsdatentypen ist der Ausgang dieselbe Zahl. Beim Eingangsdatentyp TIME entspricht das Ergebnis der Anzahl der Millisekunden. Bei STRING-Eingängen ist der Ausgang die Zahl, die durch die Zeichenfolge dargestellt wird, oder 0,0, wenn die Zeichenfolge keine gültige Zahl darstellt. In der KOP-Sprache wird die Konvertierung nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL-Sprache konvertiert die Funktion ANY_TO_LREAL das aktuelle Ergebnis.

ST-Sprache

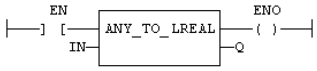
```
Q := ANY_TO_LREAL (IN);
```

FBD-Sprache



KOP-Sprache

Die Konvertierung wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD              IN
        ANY_TO_LREAL
        ST              Q
```

Siehe auch

ANY_TO_BOOL ANY_TO_SINT ANY_TO_INT ANY_TO_DINT ANY_TO_LINT ANY_TO_REAL
ANY_TO_TIME ANY_TO_STRING

ANY_TO_REAL

Operator – Konvertiert den Eingang in einen reellen Zahlenwert.

Eingänge

IN : ANY Eingangswert.

Ausgänge

Q : REAL Wert in reelle Zahl konvertiert.

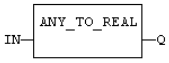
Anmerkungen

Bei BOOL-Eingangsdatentypen ist der Ausgang 0,0 oder 1,0. Bei DINT-Eingangsdatentypen ist der Ausgang dieselbe Zahl. Beim Eingangsdatentyp TIME entspricht das Ergebnis der Anzahl der Millisekunden. Bei STRING-Eingängen ist der Ausgang die Zahl, die durch die Zeichenfolge dargestellt wird, oder 0,0, wenn die Zeichenfolge keine gültige Zahl darstellt. In der KOP-Sprache wird die Konvertierung nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL-Sprache konvertiert die Funktion ANY_TO_REAL das aktuelle Ergebnis.

ST-Sprache

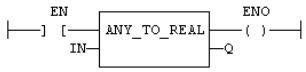
```
Q := ANY_TO_REAL (IN);
```

FBD-Sprache



KOP-Sprache

Die Konvertierung wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-SPRACHE

```
Op1:    LD              IN
        ANY_TO_REAL
        ST              Q
```

Siehe auch

ANY_TO_BOOL ANY_TO_SINT ANY_TO_INT ANY_TO_DINT ANY_TO_LINT ANY_TO_LREAL
ANY_TO_TIME ANY_TO_STRING

ANY_TO_SINT

Operator – Konvertiert den Eingang in einen kurzen (8-Bit) Ganzzahlwert.

Eingänge

IN : ANY Eingangswert.

Ausgänge

Q : SINT Wert in kurze (8-Bit) Ganzzahl konvertiert.

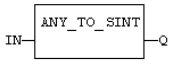
Anmerkungen

Bei BOOL-Eingangsdatentypen ist der Ausgang 0 oder 1. Für den Eingangsdatentyp REAL ist der Ausgang der ganzzahlige Teil des reellen Eingangs-Zahlenwerts. Beim Eingangsdatentyp TIME entspricht das Ergebnis der Anzahl der Millisekunden. Bei STRING-Eingängen ist der Ausgang die Zahl, die durch die Zeichenfolge dargestellt wird, oder 0, wenn die Zeichenfolge keine gültige Zahl darstellt. In der KOP-Sprache wird die Konvertierung nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL-Sprache konvertiert die Funktion ANY_TO_SINT das aktuelle Ergebnis.

ST-Sprache

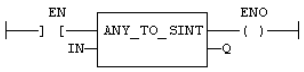
```
Q := ANY_TO_SINT (IN);
```

FBD-Sprache



KOP-Sprache

Die Konvertierung wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD          IN
        ANY_TO_SINT
        ST          Q
```

Siehe auch

ANY_TO_BOOL ANY_TO_INT ANY_TO_DINT ANY_TO_LINT ANY_TO_REAL ANY_TO_LREAL
ANY_TO_TIME ANY_TO_STRING

ANY_TO_STRING

Operator – Konvertiert den Eingang in einen Zeichenfolgenwert.

Eingänge

IN : ANY Eingangswert.

Ausgänge

Q : ANY Wert in Zeichenfolge konvertiert.

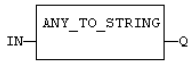
Anmerkungen

Bei BOOL-Eingangsdatentypen steht der Ausgang 1 für *TRUE* und der Ausgang 0 für *FALSE*. Bei den Eingangsdatentypen DINT, REAL und TIME ist der Ausgang die Zeichenfolgendarstellung der Eingangszahl. Dies ist die Anzahl von Millisekunden für TIME-Eingänge. In der KOP-Sprache wird die Konvertierung nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL-Sprache konvertiert die Funktion ANY_TO_STRING das aktuelle Ergebnis.

ST-Sprache

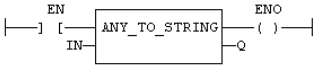
```
Q := ANY_TO_STRING (IN);
```

FBD-Sprache



KOP-Sprache

Die Konvertierung wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD              IN
        ANY_TO_STRING
        ST              Q
```

Siehe auch

ANY_TO_BOOL ANY_TO_SINT ANY_TO_INT ANY_TO_DINT ANY_TO_LINT ANY_TO_REAL
ANY_TO_LREAL ANY_TO_TIME

ANY_TO_TIME

Operator – Konvertiert den Eingang in einen Zeitwert.

Eingänge

IN : ANY Eingangswert.

Ausgänge

Q : TIME Wert in Zeit konvertiert.

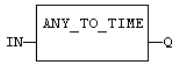
Anmerkungen

Bei BOOL-Eingangsdatentypen ist der Ausgang t#0ms oder t#1ms. Für den Eingangsdatentyp DINT oder REAL ist der Ausgang die durch die Eingangszahl angegebene Zeit, dargestellt als Anzahl von Millisekunden. Bei STRING-Eingängen ist der Ausgang die Zeit, die durch die Zeichenfolge dargestellt wird, oder t#0ms, wenn die Zeichenfolge keine gültige Zeit darstellt. In der KOP-Sprache wird die Konvertierung nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL-Sprache konvertiert die Funktion ANY_TO_TIME das aktuelle Ergebnis.

ST-Sprache

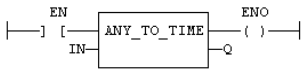
```
Q := ANY_TO_TIME (IN);
```

FBD-Sprache



KOP-Sprache

Die Konvertierung wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD          IN
        ANY_TO_TIME
        ST          Q
```

Siehe auch

ANY_TO_BOOL ANY_TO_SINT ANY_TO_INT ANY_TO_DINT ANY_TO_LINT ANY_TO_REAL ANY_TO_LREAL ANY_TO_STRING

ArrayToString / ArrayToStringU

Funktion – Kopiert ein Array aus SINT in einen STRING.

Eingänge

SRC : SINT Quell-Array von SINT kurzen Ganzzahlen (USINT für ArrayToStringU).
DST : STRING Ziel-STRING.
COUNT : DINT Anzahl der zu kopierenden Zeichen.

Ausgänge

Q : DINT Anzahl der kopierten Zeichen.

Anmerkungen

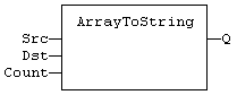
In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe.

Diese Funktion kopiert die ersten COUNT (Anzahl) Elemente des SRC-Arrays in die Zeichen der DST-Zeichenfolge. Die Funktion prüft die maximale Größe der Zielzeichenfolge und passt bei Bedarf die Anzahl für COUNT an.

ST-Sprache

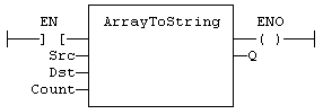
Q := ArrayToString (SRC, DST, COUNT);

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

Nicht verfügbar.

Siehe auch

StringToArray

ASCII

Funktion – Ruft den ASCII-Code eines Zeichens innerhalb einer Zeichenfolge ab.

Eingänge

IN : STRING Eingangszeichenfolge
POS : DINT Position des Zeichens innerhalb der Zeichenfolge. (Die erste gültige Position ist 1).

Ausgänge

CODE : DINT ASCII-Code des ausgewählten Zeichens oder 0, wenn die Position ungültig ist.

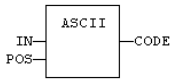
Anmerkungen

In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den gleichen Wert bei wie die Eingangsstufe. In AWL-Sprache muss der erste Parameter (IN) in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann. Der andere Eingang ist der Operand der Funktion.

ST-Sprache

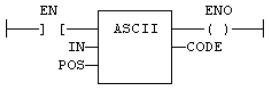
```
CODE := ASCII (IN, POS);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO ist gleich EN.



AWL-Sprache

```
Op1:    LD              IN
        AND_MASKMSK
        ST              CODE
```

Siehe auch

CHAR

ASIN / ASINL

Funktion – Berechnet einen Arcussinus.

Eingänge

IN : REAL/LREAL Reeller Zahlenwert.

Ausgänge

Q : REAL/LREAL Ergebnis: Arcussinus von IN.

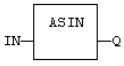
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

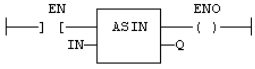
```
Q := ASIN (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        ASIN
        ST      Q (* Q ist: ASIN (IN) *)
```

Siehe auch

SIN COS TAN ACOS ATAN ATAN2

ATAN / ATANL

Funktion – Berechnet einen Arcustangens.

Eingänge

IN : REAL/LREAL Reeller Zahlenwert.

Ausgänge

Q : REAL/LREAL Ergebnis: Arcustangens von IN.

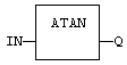
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

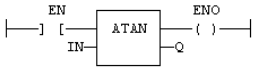
```
Q := ATAN (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        ATAN
        ST      Q (* Q ist: ATAN (IN) *)
```

Siehe auch

SIN COS TAN ASIN ACOS ATAN2

ATOH

Funktion – Konvertiert eine Zeichenfolge auf hexadezimaler Basis in eine Ganzzahl.

Eingänge

IN : STRING Zeichenfolge, die eine Ganzzahl im Hexadezimalformat darstellt.

Ausgänge

Q : DINT Ganzzahl, dargestellt durch die Zeichenfolge.

Wahrheitstabelle (Beispiele)

IN	Q
“	0
‘12’	18
‘a0’	160
A0zzz’	160

Anmerkungen

Bei der Funktion wird nicht zwischen Groß- und Kleinschreibung unterschieden. Das Ergebnis ist 0 für eine leere Zeichenfolge. Die Konvertierung wird vor dem ersten ungültigen Zeichen beendet. In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

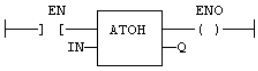
ST-Sprache

```
Q := ATOH (IN);
```

FBD-Sprache

KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        ATOH
        ST      Q
```

Siehe auch

HTOA

BCD_TO_BIN

Funktion – Konvertiert einen BCD-Wert (Binär codierte Dezimalstelle) in einen Binärwert.

Eingänge

IN : DINT Ganzzahliger Wert in BCD.

Ausgänge

Q : DINT Wert in Ganzzahl konvertiert oder 0, wenn IN kein gültiger positiver BCD-Wert ist.

Wahrheitstabelle (Beispiele)

IN	Q
-2	0 (ungültig)
0	0
16 (16#10)	10
15 (16#0F)	0 (ungültig)

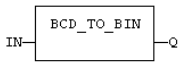
Anmerkungen

Der Eingang muss positiv sein und einen gültigen BCD-Wert darstellen. In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

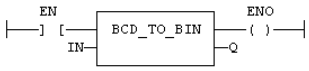
```
Q := BCD_TO_BIN (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        BCD_TO_BIN
        ST      Q
```

Siehe auch

BIN_TO_BCD

BIN_TO_BCD

Funktion – Konvertiert einen Binärwert in einen BCD-Wert (Binär codierte Dezimalstelle).

Eingänge

IN : DINT Ganzzahliger Wert

Ausgänge

Q : DINT Wert in BCD konvertiert oder 0, wenn IN kleiner als 0 ist.

Wahrheitstabelle (Beispiele)

IN	Q
-2	0 (ungültig)
0	0
10	16 (16#10)
22	34 (16#34)

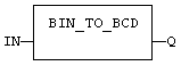
Anmerkungen

Der Eingang muss positiv sein. In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

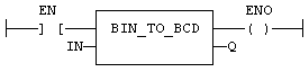
```
Q := BIN_TO_BCD (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        BIN_TO_BCD
        ST      Q
```

Siehe auch

BCD_TO_BIN

BLINK

Funktionsblock – Blinker.

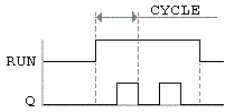
Eingänge

RUN : BOOL Aktivierungsbefehl.
CYCLE : TIME Zeitdauer des Blinkens.

Ausgänge

Q : BOOL Ausgangs-Blinksignal.

Zeitdiagramm



Anmerkungen

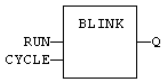
Das Ausgangssignal ist *FALSE*, wenn der RUN-Eingang *FALSE* ist. Der CYCLE-Eingang ist die gesamte Zeitdauer des Blinksignals. In der KOP-Sprache ist die Eingangsstufe der IN-Befehl. Die Ausgangsstufe ist das Q-Ausgangssignal.

ST-Sprache

MyBlinker ist eine deklarierte Instanz des Funktionsblocks BLINK.

```
MyBlinker (RUN, CYCLE);
Q := MyBlinker.Q;
```

FBD-Sprache



KOP-Sprache

AWL-Sprache

MyBlinker ist eine deklarierte Instanz des Funktionsblocks BLINK.

```
Op1:    CAL     MyBlinker (RUN, CYCLE)
        LD      MyBlinker.Q
        ST      Q
```

Siehe auch

TONTOFTP

BLINKA

Funktionsblock – Asymmetrischer Blinker.

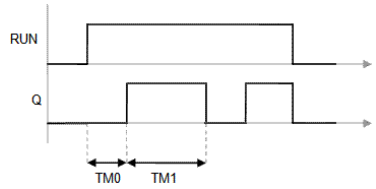
Eingänge

RUN : BOOL Aktivierungsbefehl.
TM0 : TIME Dauer des Status *FALSE* am Ausgang.
TM1 : TIME Dauer des Status *TRUE* am Ausgang.

Ausgänge

Q : BOOL Ausgangs-Blinksignal.

Zeitdiagramm



Anmerkungen

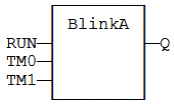
Das Ausgangssignal ist *FALSE*, wenn der RUN-Eingang *FALSE* ist. In der KOP-Sprache ist die Eingangsstufe der IN-Befehl. Die Ausgangsstufe ist das Q-Ausgangssignal.

ST-Sprache

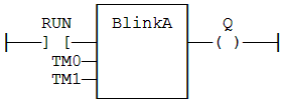
MyBlinker ist eine deklarierte Instanz des Funktionsblocks BLINKA.

```
MyBlinker (RUN, TM0, TM1);
Q := MyBlinker.Q;
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

MyBlinker ist eine deklarierte Instanz des Funktionsblocks BLINKA.

```
Op1:    CAL     MyBlinker (RUN, TM0, TM1)
        LD      MyBlinker.Q
        ST      Q
```

Siehe auch

TONTOFTP

CHAR

Funktion – Erstellt eine Zeichenfolge mit nur einem Zeichen.

Eingänge

CODE : DINT ASCII-Code des gewünschten Zeichens.

Ausgänge

Q : STRING STRING, der nur das angegebene Zeichen enthält.

Anmerkungen

In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den gleichen Wert bei wie die Eingangsstufe. In AWL-Sprache muss der Eingangsparameter (CODE) in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

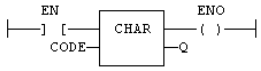
ST-Sprache

```
Q := CHAR (CODE);
```

FBD-Sprache

KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO ist gleich EN.



AWL-Sprache

```
Op1:    LD      CODE
        CHAR
        ST      Q
```

Siehe auch

ASCII

CMP

Funktionsblock – Vergleich mit detaillierten Ausgängen für Ganzzahl-Eingänge.

Eingänge

IN1 : DINT Erster Wert.
IN2 : DINT Zweiter Wert.

Ausgänge

LT : BOOL *TRUE*, wenn IN1 < IN2.
EQ : BOOL *TRUE*, wenn IN1 = IN2.
GT : BOOL *TRUE*, wenn IN1 > IN2.

Anmerkungen

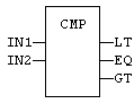
In der KOP-Sprache validiert der Stufeneingang (EN) den Vorgang. Der Stufenausgang ist das Ergebnis von LT (kleiner als der Vergleich).

ST-Sprache

MyCmp wird als Instanz des Funktionsblocks CMP deklariert:

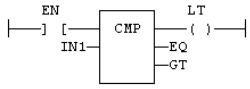
```
MyCMP (IN1, IN2);
bLT := MyCmp.LT;
bEQ := MyCmp.EQ;
bGT := MyCmp.GT;
```

FBD-Sprache



KOP-Sprache

Der Vergleich wird nur durchgeführt, wenn EN *TRUE* ist:



AWL-Sprache

MyCmp wird als Instanz des Funktionsblocks CMP deklariert:

```
Op1:    CAL     MyCmp (IN1, IN2)
        LD      MyCmp.LT
        ST      bLT
        LD      MyCmp.EQ
        ST      bEQ
        LD      MyCmp.GT
        ST      bGT
```

Siehe auch

>= GE > GT = EQ <> NE <= LE < LT

CONCAT

Funktion – Verkettung von Zeichenfolgen.

Eingänge

IN_1 : STRING Beliebige Zeichenfolgenvariable oder konstanter Ausdruck.
IN_N : STRING Beliebige Zeichenfolgenvariable oder konstanter Ausdruck.

Ausgänge

Q : STRING Verkettung aller Eingänge.

Anmerkungen

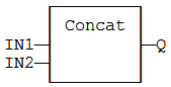
In der FBD- oder KOP-Sprache kann der Block bis zu 16 Eingänge haben. In AWL oder ST akzeptiert die Funktion eine variable Anzahl von Eingängen (mindestens 2).

Beachten Sie, dass Sie auch den Operator „+“ verwenden können, um Zeichenfolgen zu verketten.

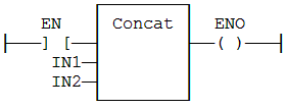
ST-Sprache

```
Q := CONCAT ('AB', 'CD', 'E');(* ist jetzt 'ABCDE' *)
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

```
Op1:    LD          'AB'
        CONCAT      'CD'        'E'
        ST          Q (* Q ist jetzt 'ABCDE' *)
```

COS / COSL

Funktion – Berechnet einen Cosinus.

Eingänge

IN : REAL/LREAL Reeller Zahlenwert.

Ausgänge

Q : REAL/LREAL Ergebnis: Cosinus von IN.

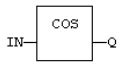
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

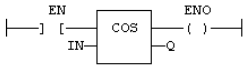
```
Q := COS (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        COS
        ST      Q       (* Q ist: COS (IN) *)
```

Siehe auch

SIN TAN ASIN ACOS ATAN ATAN2

CRC16

Funktion – Berechnet einen CRC16-Wert für die Zeichen einer Zeichenfolge.

Eingänge

IN : STRING Zeichenfolge.

Ausgänge

Q : INT CRC16 für alle Zeichen der Zeichenfolge berechnet.

Anmerkungen

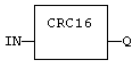
In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den gleichen Wert bei wie die Eingangsstufe. In AWL-Sprache muss der Eingangsparameter (IN) in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

Die Funktion berechnet einen MODBUS CRC16, initialisiert bei einem Wert von 16#FFFF.

ST-Sprache

```
Q := CRC16 (IN);
```

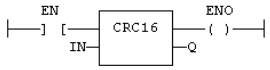
FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.

ENO ist gleich EN.



AWL-Sprache

```
Op1:    LD      IN
        CRC16
        ST      Q
```

CTD / CTDr

Funktionsblock – Abwärtszähler.

Eingänge

CD : BOOL Zählung aktivieren. Der Zähler wird bei jedem Aufruf verringert, wenn CD *TRUE* ist.

LOAD : BOOL Befehl erneut laden. Der Zähler wird auf PV gesetzt, wenn er mit LOAD gleich *TRUE* aufgerufen wird.

PV : DINT Programmierter Maximalwert.

Ausgänge

Q : BOOL *TRUE*, wenn der Zähler leer ist, d.h. wenn CV = 0.

CV : DINT Aktueller Wert des Zählers.

Anmerkungen

Der Zähler ist beim Start der Anwendung leer (CV = 0). Der Zähler enthält keine Impulserkennung für den CD-Eingang. Verwenden Sie den Funktionsblock R_TRIG oder F_TRIG zum Zählen von Impulsen des CD-Eingangssignals. In der KOP-Sprache ist CD die Eingangsstufe. Die Ausgangsstufe ist der Q-Ausgang.

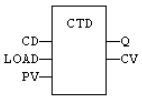
Die Funktionsblöcke CTUr, CTDr und CTUDr funktionieren genau wie andere Zähler, mit der Ausnahme, dass alle booleschen Eingänge (CU, CD, RESET, LOAD) eine implizite ansteigende Flankenerkennung enthalten. Diese Zähler werden auf einigen Zielsystemen möglicherweise nicht unterstützt.

ST-Sprache

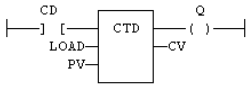
MyCounter ist eine deklarierte Instanz des Funktionsblocks CTD.

```
MyCounter (CD, LOAD, PV);
Q := MyCounter.Q;
CV := MyCounter.CV;
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

MyCounter ist eine deklarierte Instanz des Funktionsblocks CTD.

```
Op1:    CAL     MyCounter (CD, LOAD, PV)
        LD      MyCounter.Q
        ST      Q
        LD      MyCounter.CV
        ST      CV
```

Siehe auch

CTU CTUD

CTU / CTUr

Funktionsblock – Aufwärtszähler.

Eingänge

CU : BOOL Zählung aktivieren. Der Zähler wird bei jedem Aufruf erhöht, wenn CU *TRUE* ist.
RESET : BOOL Rückstellungsbefehl. Der Zähler wird bei Aufruf mit RESET gleich *TRUE* auf 0 zurückgesetzt.
PV : DINT Programmierter Maximalwert.

Ausgänge

Q : BOOL *TRUE*, wenn der Zähler voll ist, d.h. wenn CV = PV.
CV : DINT Aktueller Wert des Zählers.

Anmerkungen

Der Zähler ist beim Start der Anwendung leer (CV = 0). Der Zähler enthält keine Impulserkennung für den CU-Eingang. Verwenden Sie den Funktionsblock R_TRIG oder F_TRIG zum Zählen von Impulsen des CU-Eingangssignals. In der KOP-Sprache ist CU die Eingangsstufe. Die Ausgangsstufe ist der Q-Ausgang.

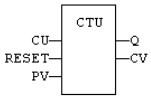
Die Funktionsblöcke CTUr, CTDr und CTUDr funktionieren genau wie andere Zähler, mit der Ausnahme, dass alle booleschen Eingänge (CU, CD, RESET, LOAD) eine implizite ansteigende Flankenerkennung enthalten. Diese Zähler werden auf einigen Zielsystemen möglicherweise nicht unterstützt.

ST-Sprache

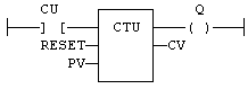
MyCounter ist eine deklarierte Instanz des Funktionsblocks CTU.

```
MyCounter (CU, RESET, PV);
Q := MyCounter.Q;
CV := MyCounter.CV;
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

MyCounter ist eine deklarierte Instanz des Funktionsblocks CTU.

```
Op1:    CAL     MyCounter (CU, RESET, PV)
        LD      MyCounter.Q
        ST      Q
        LD      MyCounter.CV
        ST      CV
```

Siehe auch

CTD CTUD

CTUD / CTUDr

Funktionsblock – Auf-/Abwärtszähler.

Eingänge

CU : BOOL Zählung aktivieren. Der Zähler wird bei jedem Aufruf erhöht, wenn CU *TRUE* ist.
CD : BOOL Zählung aktivieren. Der Zähler wird bei jedem Aufruf verringert, wenn CD *TRUE* ist.
RESET : BOOL Rückstellungsbefehl. Der Zähler wird bei Aufruf mit RESET gleich *TRUE* auf 0 zurückgesetzt.
LOAD : BOOL BOOL Befehl erneut laden. Der Zähler wird auf PV gesetzt, wenn er mit LOAD gleich *TRUE* aufgerufen wird.
PV : DINT Programmierter Maximalwert.

Ausgänge

QU : BOOL *TRUE*, wenn der Zähler voll ist, d.h. wenn CV = PV.
QD : BOOL *TRUE*, wenn der Zähler leer ist, d.h. wenn CV = 0.
CV : DINT Aktueller Wert des Zählers.

Anmerkungen

Der Zähler ist beim Start der Anwendung leer (CV = 0). Der Zähler enthält keine Impulserkennung für CU- und CD-Eingänge. Verwenden Sie die Funktionsblöcke R_TRIG oder F_TRIG zum Zählen der Impulse von CU- oder CD-Eingangssignalen. In der KOP-Sprache ist CU die Eingangsstufe. Die Ausgangsstufe ist der QU-Ausgang.

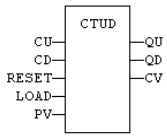
Die Funktionsblöcke CTUr, CTDr und CTUDr funktionieren genau wie andere Zähler, mit der Ausnahme, dass alle booleschen Eingänge (CU, CD, RESET, LOAD) eine implizite ansteigende Flankenerkennung enthalten. Diese Zähler werden auf einigen Zielsystemen möglicherweise nicht unterstützt.

ST-Sprache

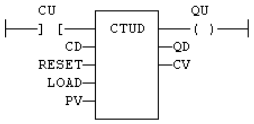
MyCounter ist eine deklarierte Instanz des Funktionsblocks CTUD.

```
MyCounter (CU, CD, RESET, LOAD, PV);
QU := MyCounter.QU;
QD := MyCounter.QD;
CV := MyCounter.CV;
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

MyCounter ist eine deklarierte Instanz des Funktionsblocks CTUD.

```
Op1:    CAL     MyCounter (CU, CD, RESET, LOAD, PV)
        LD      MyCounter.QU
        ST      QU
        LD      MyCounter.QD
        ST      QD
        LD      MyCounter.CV
        ST      CV
```

Siehe auch

CTU CTD

DELETE

Funktion – Löscht Zeichen in einer Zeichenfolge.

Eingänge

IN : STRING Zeichenfolge.
NBC : DINT Anzahl der zu löschenden Zeichen.
POS : DINT Position des ersten gelöschten Zeichens (Position des ersten Zeichens ist 1).

Ausgänge

Q : STRING Geänderte Zeichenfolge.

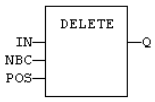
Anmerkungen

Die erste gültige Zeichenposition ist 1. In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der erste Eingang (die Zeichenfolge) in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann. Andere Argumente sind Operanden der Funktion, die durch Kommas getrennt sind.

ST-Sprache

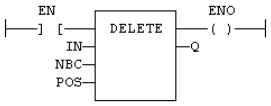
```
Q := DELETE (IN, NBC, POS);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        DELETE  NBC, POS
        ST      Q
```

Siehe auch

+ ADD MLEN INSERT FIND REPLACE LEFT RIGHT MID

/ DIV

Operator – Dividiert Eingänge.

Eingänge

IN1 : ANY_NUM Erster Eingang.
IN2 : ANY_NUM Zweiter Eingang.

Ausgänge

Q : ANY_NUM Ergebnis: IN1 / IN2.

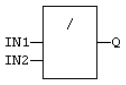
Anmerkungen

Alle Eingänge und der Ausgang müssen vom gleichen Typ sein. In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den gleichen Wert bei wie die Eingangsstufe. In AWL-Sprache teilt der DIV-Befehl das aktuelle Ergebnis durch den Operanden. Das aktuelle Ergebnis und der Operand müssen vom gleichen Typ sein.

ST-Sprache

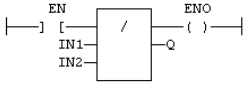
```
Q := IN1 / IN2;
```

FBD-Sprache



KOP-Sprache

Die Division wird nur ausgeführt, wenn EN *TRUE* ist.
ENO ist gleich EN.



AWL-Sprache

```
Op1:    LD      IN1
        DIV     IN2
        ST      Q       (* Q ist gleich: IN1 / IN2 *)
Op2:    LD      IN1
        DIV     IN2
        DIV     IN3
        ST      Q       (* Q ist gleich: IN1 / IN2 / IN3 *)
```

Siehe auch

+ ADD NEG - * MUL

DTAt

Funktionsblock – Generiert zu einer bestimmten Uhrzeit/Datum einen Impuls

Eingänge

YEAR :	DINT Gewünschtes Jahr (z. B. 2006).
MONTH :	DINT Gewünschter Monat (1 = Januar).
DAY :	DINT Gewünschter Tag (1 bis 31).
TMOFDAY :	TIME Gewünschte Uhrzeit.
RST :	BOOL Rückstellungsbefehl.

Ausgänge

QAT : BOOL	Impulssignal.
QPAST : BOOL	TRUE, falls abgelaufen.

Achtung

Die Echtzeituhr ist auf einigen Zielsystemen möglicherweise nicht verfügbar. Weitere Informationen zu den verfügbaren Funktionen finden Sie in den OEM-Anweisungen.

Anmerkungen

Parameter werden nicht ständig aktualisiert. Sie werden nur berücksichtigt, wenn:

- der Block der erste Mal aufgerufen wird.
- wenn der Rücksetzeingang (RST) *TRUE* ist.

In diesen beiden Situationen werden die Ausgänge auf *FALSE* zurückgesetzt.

Wenn der Block mit RST= *FALSE* das erste Mal aufgerufen wird und der angegebene Datumstempel übergeben wird, wird der Ausgang QPAST auf *TRUE* und der Ausgang QAT für genau einen Zyklus (Impulssignal) auf TRUE gesetzt.

Höchste Einheiten werden ignoriert, wenn sie auf 0 gesetzt sind. Wenn die Argumente beispielsweise year=0, month=0, day = 3, tmofday=t#10h lauten, wird der Block am nächsten 3. Tag des Monats um 10 Uhr ausgelöst.

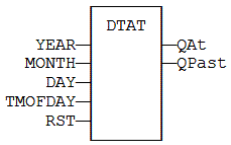
In der KOP-Sprache wird der Block nur aktiviert, wenn die Eingangsstufe *TRUE* ist.

ST-Sprache

MyDTAT ist eine deklarierte Instanz des Funktionsblocks DTAT.

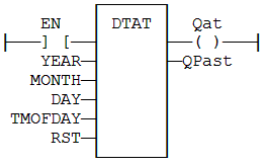
```
MyDTAT (YEAR, MONTH, DAY, TMOFDAY, RST);
QAT := MyDTAT.QAT;
QPAST := MyDTATA.QPAST;
```

FBD-Sprache



KOP-Sprache

Wird nur aufgerufen, wenn EN *TRUE* ist.



AWL-Sprache

MyDTAT ist eine deklarierte Instanz des Funktionsblocks DTAT.

```
Op1:    CAL     MyDTAT (YEAR, MONTH, DAY, TMOFDAY, RST)
        LD      MyDTAT.QAT
        ST      QAT
        LD      MyDTATA.QPAST
        ST      QPAST
```

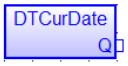
DTCurDate

Funktion: Aktuellen Datumstempel abrufen

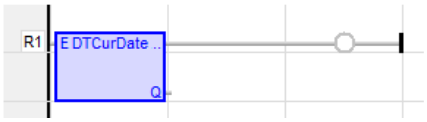
Q := DTCurDate ();

Q : DINT Numerischer Stempel, der das aktuelle Datum darstellt.

FBD-Sprache



KOP-Sprache



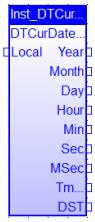
DTCurDateTime

Funktion: Aktuellen Zeitstempel abrufen (Funktionsblock)

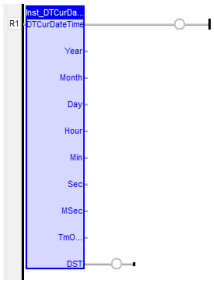
Inst_DTCurDateTime (bLocal);

bLocal :	BOOL *TRUE*, wenn lokale Zeit angefordert wird (GMT wenn *FALSE*).
.Year :	DINT Ausgang: Aktuelles Jahr
.Month :	DINT Ausgang: Aktueller Monat
.Day :	DINT Ausgang: Aktueller Tag
.Hour :	DINT Ausgang: Aktuelle Zeit: Stunden
.Min :	DINT Ausgang: Aktuelle Zeit: Minuten
.Sec :	DINT Ausgang: Aktuelle Zeit: Sekunden
.MSec :	DINT Ausgang: Aktuelle Zeit: Millisekunden
.TmOfDay :	TIME Ausgang: Aktuelle Tageszeit (seit Mitternacht)
.DST :	BOOL Ausgang: *TRUE*, wenn Sommerzeit aktiv ist

FBD-Sprache



KOP-Sprache



DTCurTime

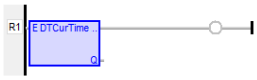
Funktion: Aktuellen Zeitstempel abrufen

Q := DTCurTime ();

Q : DINT Numerischer Stempel, der die aktuelle Uhrzeit des Tages darstellt.

FBD-Sprache

KOP-Sprache



DTDay

Funktion: Tag des Monats aus einem Datumstempel extrahieren

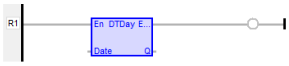
Q := DTDay (iDate);

IDATE : DINT Numerischer Stempel, der ein Datum darstellt.

Q : DINT Tag des Monats des Datums (1..31).

FBD-Sprache

KOP-Sprache



DTFormat

Funktion – Formatiert das aktuelle Datum/die aktuelle Uhrzeit in einer Zeichenfolge mit einem benutzerdefinierten Format

Eingänge

FMT : STRING Formatzeichenfolge.

Ausgänge

Q : STRING Zeichenfolge, die das formatierte Datum oder die formatierte Uhrzeit enthält.

Achtung

Die Echtzeituhr ist auf einigen Zielsystemen möglicherweise nicht verfügbar. Weitere Informationen zu den verfügbaren Funktionen finden Sie in den OEM-Anweisungen.

Anmerkungen

Die Formatzeichenfolge kann ein beliebiges Zeichen enthalten. Einige spezielle Markierungen, die mit dem Zeichen „%“ beginnen, weisen auf Datums-/Uhrzeitinformationen hin:

%Y Jahr einschließlich Jahrhundert (z. B. 2006)
%y Jahr ohne Jahrhundert (z. B. 06)
%m Monat (1..12)
%d Tag des Monats (1..31)
%H Stunden (0..23)
%M Minuten (0..59)
%S Sekunden (0..59)

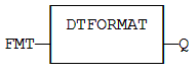
Beispiel

```
(* Angenommen, wir haben den 4. Juli 2006 um 18:45:20 *)
Q := DTFORMAT ('Heute ist %Y/%m/%d - %H:%M:%S');
(* Q lautet 'Heute ist 2006/07/04 - 18:45:20 *)
```

ST-Sprache

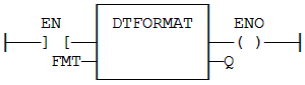
```
Q := DTFORMAT (FMT);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN TRUE ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD FMT
        DTFORMAT
        ST Q
```

DTHour

Funktion: Stunden aus einem Zeitstempel extrahieren

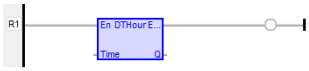
Q := DTHour (iTime);

ITIME : DINT Numerischer Stempel, der eine Zeit darstellt.

Q : DINT Stunden der Uhrzeit (0..23).

FBD-Sprache

KOP-Sprache



DTMin

Funktion: Stunden aus einem Zeitstempel extrahieren

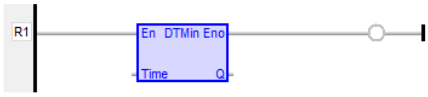
Q := DTMin (iTime);

ITIME : DINT Numerischer Stempel, der eine Zeit darstellt.

Q : DINT Stunden der Uhrzeit (0..59).

FBD-Sprache

KOP-Sprache



DTMonth

Funktion: Monat aus einem Datumstempel extrahieren

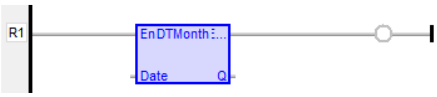
Q := DTMonth (iDate);

IDATE : DINT Numerischer Stempel, der ein Datum darstellt.

Q : DINT Monat des Datums (1..12).

FBD-Sprache

KOP-Sprache



DTMs

Funktion: Millisekunden aus einem Zeitstempel extrahieren

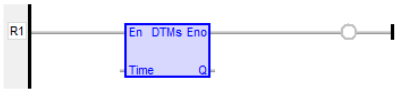
Q := DTMs (iTime);

ITIME : DINT Numerischer Stempel, der eine Zeit darstellt.

Q : DINT Millisekunden der Uhrzeit (0..999).

FBD-Sprache

KOP-Sprache



DTSec

Funktion: Sekunden aus einem Zeitstempel extrahieren

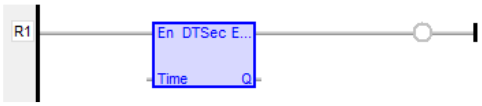
Q := DTSec (iTime);

ITIME : DINT Numerischer Stempel, der eine Zeit darstellt.

Q : DINT Sekunden der Uhrzeit (0..59).

FBD-Sprache

KOP-Sprache



DTYear

Funktion: Jahr aus einem Datumstempel extrahieren

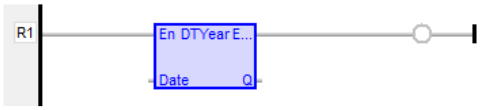
Q := DTYear (iDate);

IDATE : DINT Numerischer Stempel, der ein Datum darstellt.

Q : DINT Jahr des Datums (z. B. 2004).

FBD-Sprache

KOP-Sprache



= EQ

Operator – Test, ob der erste Eingang gleich dem zweiten Eingang ist.

Eingänge

IN1 : ANY Erster Eingang.
IN2 : ANY Zweiter Eingang.

Ausgänge

Q : BOOL *TRUE*, wenn IN1 = IN2.

Anmerkungen

Beide Eingänge müssen vom gleichen Typ sein. In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe ist das Ergebnis des Vergleichs. In AWL-Sprache führt der EQ-Befehl den Vergleich zwischen dem aktuellen Ergebnis und dem Operanden durch. Das aktuelle Ergebnis und der Operand müssen vom gleichen Typ sein.

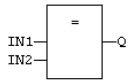
Vergleiche können mit Zeichenfolgen verwendet werden. In diesem Fall wird die lexikalische Reihenfolge zum Vergleichen der Eingangszeichenfolgen verwendet. Beispiel: „ABC" ist kleiner als „ZX"; „ABCD" ist größer als „ABC".

Vergleiche mit „gleich" können nicht mit TIME-Variablen verwendet werden. Der Grund dafür ist, dass der Timer tatsächlich die Auflösung des Zielzyklus aufweist und der Test unsicher sein kann, da einige Werte möglicherweise nie erreicht werden.

ST-Sprache

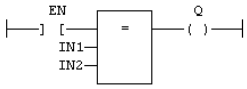
```
Q := IN1 = IN2;
```

FBD-Sprache



KOP-Sprache

Der Vergleich wird nur ausgeführt, wenn EN *TRUE* ist.



AWL-Sprache

```
Op1:    LD    IN1
        EQ    IN2
        ST    Q      (* Q ist true, wenn IN1 = IN2 *)
```

Siehe auch

> GT < LT >= GE <= LE <> NE CMP

EXP / EXPL

Funktion – Berechnet die natürliche Exponentialfunktion des Eingangs.

Eingänge

IN : REAL/LREAL Reeller Zahlenwert.

Ausgänge

Q : REAL/LREAL Ergebnis: Natürliche Exponentialfunktion von IN.

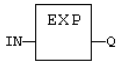
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

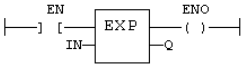
```
Q := EXP (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD    IN
        EXP
        ST    Q (* Q ist: EXP (IN) *)
```

Siehe auch

ABS TRUNC POW SQRT

EXPT

Funktion – Berechnet eine Potenz.

Eingänge

IN : REAL Reeller Zahlenwert.
EXP : DINT Exponent.

Ausgänge

Q : REAL Ergebnis: Potenz „EXP" von IN.

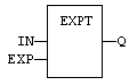
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann. Der Exponent (zweiter Eingang der Funktion) muss der Operand der Funktion sein.

ST-Sprache

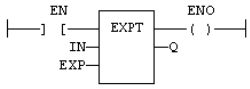
```
Q := EXPT (IN, EXP);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        EXPT EXP
        ST      Q (* Q ist: (IN ** EXP) *)
```

Siehe auch

ABS TRUNC LOG SQRT

F_TRIG

Funktionsblock – Erkennung abfallender Impulse.

Eingänge

CLK : BOOL Boolesches Signal.

Ausgänge

Q : BOOL *TRUE*, wenn der Eingang von TRUE zu *FALSE* wechselt.

Wahrheitstabelle

CLK	CLK vorh.	Q
0	0	0
0	1	1
1	0	0
1	1	0

Anmerkungen

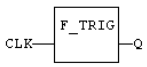
Obwohl die Kontakte vom Typ]P[und]N[in der KOP-Sprache verwendet werden können, wird empfohlen, deklarierte Instanzen der Funktionsblöcke R_TRIG oder F_TRIG zu verwenden, um ein unerwartetes Verhalten während einer Online-Änderung zu vermeiden.

ST-Sprache

MyTrigger wird als Instanz des Funktionsblocks F_TRIG deklariert:

```
MyTrigger (CLK);
Q := MyTrigger.Q;
```

FBD-Sprache



KOP-Sprache

Das Eingangssignal ist die Stufe - die Stufe ist der Ausgang:

AWL-Sprache

MyTrigger wird als Instanz des Funktionsblocks F_TRIG deklariert:

```
Op1:    CAL   MyTrigger (CLK)
        LD    MyTrigger.Q
        ST    Q
```

Siehe auch

R_TRIG

FIFO

Funktionsbaustein – Verwaltet eine First-In/First-Out-Liste (älteste Eingänge werden zuerst ausgegeben).

Eingänge

PUSH : BOOL Neuen Wert übertragen (bei ansteigender Flanke).
POP : BOOL Neuen Wert entnehmen (bei ansteigender Flanke).
RST : BOOL Liste zurücksetzen.
NEXTIN : ANY Zu übertragender Wert.
NEXTOUT : ANY Wert des ältesten übertragenen Werts – wird nach dem Aufruf aktualisiert!
BUFFER : ANY Array zum Speichern von Werten.

Ausgänge

EMPTY : BOOL *TRUE*, wenn die Liste leer ist.
OFLO : BOOL *TRUE*, wenn ein PUSH-Befehl überläuft.
COUNT : DINT Anzahl der Werte in der Liste.
PREAD : DINT Index im Puffer des ältesten übertragenen Werts.
PWRITE : DINT Index im Puffer der nächsten Übertragungsposition.

Anmerkungen

NEXTIN, NEXTOUT und BUFFER müssen denselben Datentyp haben und dürfen nicht STRING sein.

Das NEXTOUT-Argument gibt eine Variable an, die nach dem Aufruf des Blocks mit dem ältesten Push-Wert ausgefüllt wird.

Werte werden im BUFFER-Array gespeichert. Die Daten werden als Rollover-Puffer angeordnet und niemals verschoben oder zurückgesetzt. Nur Lese- und Schreibzeiger und übertragene Werte werden aktualisiert. Die maximale Größe der Liste ist die Dimension des Arrays.

Wenn der Block zum ersten Mal aufgerufen wird, merkt er sich, auf welchem Array er funktionieren sollte. Wenn Sie später dieselbe Instanz mit einem anderen BUFFER-Eingang aufrufen, wird der Aufruf als ungültig betrachtet und bewirkt nichts. In diesem Fall wird eine leere Liste ausgegeben.

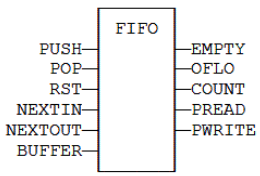
In der KOP-Sprache ist die Eingangsstufe der PUSH-Eingang. Die Ausgangsstufe ist der EMPTY-Ausgang.

ST-Sprache

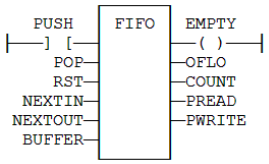
MyFIFO ist eine deklarierte Instanz des Funktionsblocks FIFO.

```
SMyFIFO (PUSH, POP, RST, NEXTIN, NEXTOUT, BUFFER);
EMPTY := MyFIFO.EMPTY;
OFLO := MyFIFO.OFLO;
COUNT := MyFIFO.COUNT;
PREAD := MyFIFO.PREAD;
PWRITE := MyFIFO.PWRITE;
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

MyFIFO ist eine deklarierte Instanz des Funktionsblocks FIFO.

```
Op1:    CAL   MyFIFO (PUSH, POP, RST, NEXTIN, NEXTOUT, BUFFER)
        LD    MyFIFO.EMPTY
        ST    EMPTY
        LD    MyFIFO.OFLO
        ST    OFLO
        LD    MyFIFO.COUNT
        ST    COUNT
        LD    MyFIFO.PREAD
        ST    PREAD
        LD    MyFIFO.PWRITE
        ST    PWRITE
```

Siehe auch

LIFO

FIND

Funktion – Sucht die Position von Zeichen in einer Zeichenfolge.

Eingänge

IN : STRING Zeichenfolge.
STR : STRING Zeichenfolge, welche die gesuchten Zeichen enthält.

Ausgänge

POS : DINT Position des ersten Zeichens von STR in IN oder 0, wenn nicht gefunden.

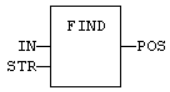
Anmerkungen

Die erste gültige Zeichenposition ist 1. Ein Rückgabewert von 0 bedeutet, dass die STR-Zeichenfolge nicht gefunden wurde. Bei der Suche wird zwischen Groß- und Kleinschreibung unterschieden. In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der erste Eingang (die Zeichenfolge) in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann. Das zweite Argument ist der Operand der Funktion.

ST-Sprache

```
POS := FIND (IN, STR);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.

AWL-Sprache

```
Op1:    LD      IN
        FIND    STR
        ST      POS
```

Siehe auch

+ADD MLEN DELETE INSERT REPLACE LEFT RIGHT MID

FLIPFLOP

Funktionsblock – Bistabiler Flipflop.

Eingänge

IN : BOOL Tauschbefehl (bei ansteigender Flanke).
RST : BOOL Rückstellung auf *FALSE*.

Ausgänge

Q : BOOL Ausgang.

Anmerkungen

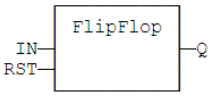
Der Ausgang wird systematisch auf *FALSE* zurückgesetzt, wenn RST *TRUE* ist. Der Ausgang ändert sich bei jeder ansteigenden Flanke des IN-Eingangs, wenn RST *FALSE* ist.

ST-Sprache

MyFlipFlop wird als Instanz des Funktionsblocks FLIPFLOP deklariert:

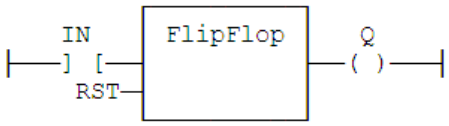
```
MyFlipFlop (IN, RST);
Q := MyFlipFlop.Q;
```

FBD-Sprache



KOP-Sprache

Der IN-Befehl ist die Stufe – die Stufe ist der Ausgang:



AWL-Sprache

MyFlipFlop wird als Instanz des Funktionsblocks FLIPFLOP deklariert:

```
Op1:    CAL     MyFlipFlop (IN, RST)
        LD      MyFlipFlop.Q
        ST      Q1
```

Siehe auch

RS SR

>=GE

Operator – Test, ob der erste Eingang größer als oder gleich dem zweiten Eingang ist.

Eingänge

IN1 : ANY Erster Eingang.
IN2 : ANY Zweiter Eingang.

Ausgänge

Q : BOOL *TRUE*, wenn IN1 >= IN2 .

Anmerkungen

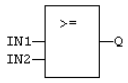
Beide Eingänge müssen vom gleichen Typ sein. In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe ist das Ergebnis des Vergleichs. In AWL-Sprache führt der GE-Befehl den Vergleich zwischen dem aktuellen Ergebnis und dem Operanden durch. Das aktuelle Ergebnis und der Operand müssen vom gleichen Typ sein.

Vergleiche können mit Zeichenfolgen verwendet werden. In diesem Fall wird die lexikalische Reihenfolge zum Vergleichen der Eingangszeichenfolgen verwendet. Beispiel: „ABC" ist kleiner als „ZX"; „ABCD" ist größer als „ABC".

ST-Sprache

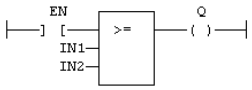
```
Q := IN1 >= IN2;
```

FBD-Sprache



KOP-Sprache

Der Vergleich wird nur ausgeführt, wenn EN *TRUE* ist.



AWL-Sprache

```
Op1:    LD    IN1
        GE    IN2
        ST    Q (* Q ist true, wenn IN1 >= IN2 *)
```

Siehe auch

> GT < LT <= LE = EQ <> NE CMP

>GT

Operator – Test, ob der erste Eingang größer ist als der zweite Eingang.

Eingänge

IN1 : ANY Erster Eingang.
IN2 : ANY Zweiter Eingang.

Ausgänge

Q : BOOL *TRUE*, wenn IN1 > IN2.

Anmerkungen

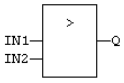
Beide Eingänge müssen vom gleichen Typ sein. In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe ist das Ergebnis des Vergleichs. In AWL-Sprache führt der GT-Befehl den Vergleich zwischen dem aktuellen Ergebnis und dem Operanden durch. Das aktuelle Ergebnis und der Operand müssen vom gleichen Typ sein.

Vergleiche können mit Zeichenfolgen verwendet werden. In diesem Fall wird die lexikalische Reihenfolge zum Vergleichen der Eingangszeichenfolgen verwendet. Beispiel: „ABC" ist kleiner als „ZX"; „ABCD" ist größer als „ABC".

ST-Sprache

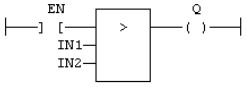
```
Q := IN1 > IN2;
```

FBD-Sprache



KOP-Sprache

Der Vergleich wird nur ausgeführt, wenn EN *TRUE* ist.



AWL-Sprache

```
Op1:    LD    IN1
        GT    IN2
        ST    Q (* Q ist true, wenn IN1 > IN2 *)
```

Siehe auch

< LT >= GE <= LE = EQ <> NE CMP

HIBYTE

Funktion – Ruft das höchstwertige Byte eines Wortes ab

Eingänge

IN : UINT 16-Bit-Register.

Ausgänge

Q : USINT Das höchstwertige Byte.

Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

```
Q := HIBYTE (IN);
```

FBD-Sprache

KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.

AWL-Sprache

```
Op1:    LD      IN
        HIBYTE
        ST      Q
```

Siehe auch

LOBYTE LOWORD HIWORD MAKEWORD MAKEDWORD

HIWORD

Funktion – Ruft das höchstwertige Wort eines Doppelworts ab.

Eingänge

IN : UDINT 32-Bit-Register.

Ausgänge

Q : UINT Das höchstwertige Wort.

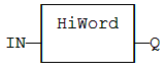
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

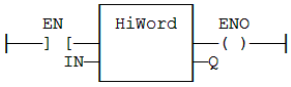
```
Q := HIWORD (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        HIWORD
        ST      Q
```

Siehe auch

LOBYTE HIBYTE LOWORD MAKEWORD MAKEDWORD

HTOA

Funktion – Konvertiert eine Ganzzahl auf hexadezimaler Basis in eine Zeichenfolge.

Eingänge

IN : DINT Ganzzahliger Wert.

Ausgänge

Q : STRING Zeichenfolge, die eine Ganzzahl im Hexadezimalformat darstellt.

Wahrheitstabelle (Beispiele)

IN	Q
0	'0'
18	'12'
160	'A0'

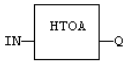
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

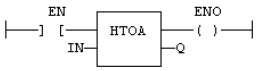
```
Q := HTOA (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        HTOA
        ST      Q
```

Siehe auch

ATOH

INSERT

Funktion – Fügt Zeichen in eine Zeichenfolge ein.

Eingänge

IN : STRING Zeichenfolge.
STR : STRING Zeichenfolge, welche die einzufügenden Zeichen enthält.
POS : DINT Position des ersten eingefügten Zeichens (Position des ersten Zeichens ist 1).

Ausgänge

Q : STRING Geänderte Zeichenfolge.

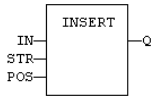
Anmerkungen

Die erste gültige Zeichenposition ist 1. In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der erste Eingang (die Zeichenfolge) in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann. Andere Argumente sind Operanden der Funktion, die durch Kommas getrennt sind.

ST-Sprache

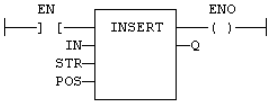
```
Q := INSERT (IN, STR, POS);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD          IN
        INSERTSTR,  POS
        ST          Q
```

Siehe auch

+ ADD MLEN DELETE FIND REPLACE LEFT RIGHT MID

LE

Operator – Test, ob der erste Eingang kleiner als oder gleich dem zweiten Eingang ist.

Eingänge

IN1 : ANY Erster Eingang.
IN2 : ANY Zweiter Eingang.

Ausgänge

Q : BOOL *TRUE*, wenn IN1 <= IN2.

Anmerkungen

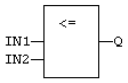
Beide Eingänge müssen vom gleichen Typ sein. In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe ist das Ergebnis des Vergleichs. In AWL-Sprache führt der LE-Befehl den Vergleich zwischen dem aktuellen Ergebnis und dem Operanden durch. Das aktuelle Ergebnis und der Operand müssen vom gleichen Typ sein.

Vergleiche können mit Zeichenfolgen verwendet werden. In diesem Fall wird die lexikalische Reihenfolge zum Vergleichen der Eingangszeichenfolgen verwendet. Beispiel: „ABC" ist kleiner als „ZX"; „ABCD" ist größer als „ABC".

ST-Sprache

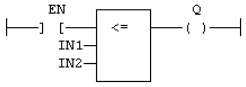
```
Q := IN1 <= IN2;
```

FBD-Sprache



KOP-Sprache

Der Vergleich wird nur ausgeführt, wenn EN *TRUE* ist:



AWL-Sprache

```
Op1:    LD    IN1
        LE    IN2
        ST    Q (* Q ist true, wenn IN1 <= IN2 *)
```

Siehe auch

> GT < LT >= GE = EQ <> NE CMP

LEFT

Funktion – Extrahiert Zeichen einer Zeichenfolge auf der linken Seite.

Eingänge

IN : STRING Zeichenfolge.
NBC : DINT Anzahl der zu extrahierenden Zeichen.

Ausgänge

Q : STRING Zeichenfolge, die die ersten NBC-Zeichen von IN enthält.

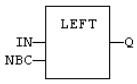
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der erste Eingang (die Zeichenfolge) in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann. Das zweite Argument ist der Operand der Funktion.

ST-Sprache

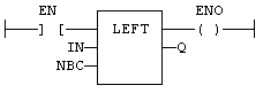
```
Q := LEFT (IN, NBC);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        LEFT    NBC
        ST      Q
```

Siehe auch

+ ADD MLEN DELETE INSERT FIND REPLACE RIGHT MID

LEN

Funktion – Ruft die Anzahl der Zeichen in einer Zeichenfolge ab.

Eingänge

IN : STRING Zeichenfolge.

Ausgänge

NBC : INT Anzahl der Zeichen, die sich derzeit in der Zeichenfolge befinden. 0, wenn die Zeichenfolge leer ist.

Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

```
NBC := LEN (IN);
```

LIFO

Funktionsbaustein – Verwaltet einen Last-In/First-Out-Stapel (neueste Eingänge werden zuerst ausgegeben).

Eingänge

PUSH : BOOL Neuen Wert übertragen (bei ansteigender Flanke).
POP : BOOL Neuen Wert entnehmen (bei ansteigender Flanke).
RST : BOOL Liste zurücksetzen.
NEXTIN : ANY Zu übertragender Wert.
NEXTOUT : ANY Wert oben auf dem Stapel – wird nach dem Aufruf aktualisiert!
BUFFER : ANY Array zum Speichern von Werten.

Ausgänge

EMPTY : BOOL *TRUE*, wenn der Stapel leer ist.
OFLO : BOOL *TRUE*, wenn ein PUSH-Befehl überläuft.
COUNT : DINT Anzahl der Werte im Stapel.
PREAD : DINT Index im Puffer oben auf dem Stapel.
PWRITE : DINT Index im Puffer der nächsten Übertragungsposition.

Anmerkungen

NEXTIN, NEXTOUT und BUFFER müssen denselben Datentyp haben und dürfen nicht STRING sein.

Das Argument NEXTOUT gibt eine Variable an, die nach dem Aufruf des Blocks mit dem Wert oben auf dem Stapel ausgefüllt wird.

Werte werden im BUFFER-Array gespeichert. Daten werden niemals verschoben oder zurückgesetzt. Nur Lese- und Schreibzeiger und übertragene Werte werden aktualisiert. Die maximale Größe des Stapels ist die Dimension des Arrays.

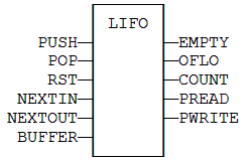
Wenn der Block zum ersten Mal aufgerufen wird, merkt er sich, auf welchem Array er funktionieren sollte. Wenn Sie später dieselbe Instanz mit einem anderen BUFFER-Eingang aufrufen, wird der Aufruf als ungültig betrachtet und bewirkt nichts. In diesem Fall wird ein leerer Stapel ausgegeben.

In der KOP-Sprache ist die Eingangsstufe der PUSH-Eingang. Die Ausgangsstufe ist der EMPTY-Ausgang.

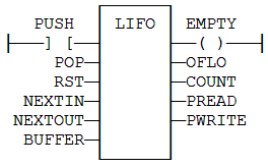
ST-Sprache

```
MyLIFO ist eine deklarierte Instanz des Funktionsblocks LIFO.
MyLIFO (PUSH, POP, RST, NEXTIN, NEXTOUT, BUFFER);
EMPTY := MyLIFO.EMPTY;
OFLO := MyLIFO.OFLO;
COUNT := MyLIFO.COUNT;
PREAD := MyLIFO.PREAD;
PWRITE := MyLIFO.PWRITE;
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

MyLIFO ist eine deklarierte Instanz des Funktionsblocks LIFO.

```
Op1:    CAL   MyLIFO (PUSH, POP, RST, NEXTIN, NEXTOUT, BUFFER)
        LD    MyLIFO.EMPTY
        ST    EMPTY
        LD    MyLIFO.OFLO
        ST    OFLO
        LD    MyLIFO.COUNT
        ST    COUNT
        LD    MyLIFO.PREAD
        ST    PREAD
        LD    MyLIFO.PWRITE
        ST    PWRITE
```

Siehe auch

FIFO

LIMIT

Funktion – Begrenzt eine ganze Zahl zwischen dem unteren und oberen Grenzwert.

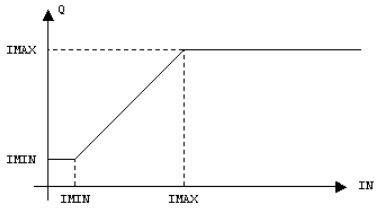
Eingänge

IMIN : DINT Untergrenze.
IN : DINT Eingangswert.
IMAX : DINT Obergrenze.

Ausgänge

Q : DINT IMIN wenn IN < IMIN; IMAX wenn IN > IMAX; andernfalls IN.

Funktionsdiagramm



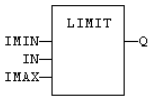
Anmerkungen

In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den Status des Eingangsstrompfads bei. In der AWL-Sprache muss der erste Eingang vor dem Funktionsaufruf geladen werden. Andere Eingänge sind Operanden der Funktion, die durch Komma getrennt sind.

ST-Sprache

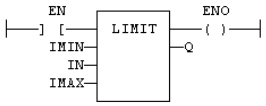
Q := LIMIT (IMIN, IN, IMAX);

FBD-Sprache



KOP-Sprache

Der Vergleich wird nur ausgeführt, wenn EN *TRUE* ist.
ENO hat den gleichen Wert wie EN.



AWL-Sprache

Op1:	LD	IMIN
	LIMIT	IN, IMAX
	ST	Q

Siehe auch

MIN MAX MOD ODD

LN

Funktion – Berechnet den natürlichen Logarithmus des Eingangs.

Eingänge

IN : REAL/LREAL Reeller Zahlenwert.

Ausgänge

Q : REAL/LREAL Ergebnis: Natürlicher Logarithmus von IN.

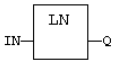
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

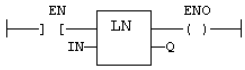
```
Q := LN (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD    IN
        LN
        ST    Q (* Q ist: LN (IN) *)
```

Siehe auch

ABS TRUNC POW SQRT

LoadString

Funktion – Lädt eine Zeichenfolge aus der aktiven Zeichenfolgentabelle.

Eingänge

ID : DINT ID der Zeichenfolge, wie in der Zeichenfolgentabelle deklariert.

Ausgänge

Q : STRING Geladene Zeichenfolge oder leere Zeichenfolge im Falle eines Fehlers.

Anmerkungen

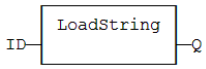
Diese Funktion lädt eine Zeichenfolge aus der aktiven Zeichenfolgentabelle und speichert sie in einer STRING-Variable. Die Funktion StringTable() wird zur Auswahl der aktiven Zeichenfolgentabelle verwendet.

Der ID-Eingang (die Zeichenfolgenelement-ID) ist eine Kennung, die in der Zeichenfolgentabellen-Ressource deklariert ist. Sie müssen diese ID nicht erneut „definieren”. Das System erledigt das für Sie.

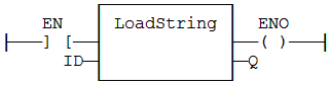
ST-Sprache

```
Q := LoadString (ID);
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

```
Op1:    LD          ID
        LoadString
        ST          Q
```

Siehe auch

StringTableZeichenfolgentabellen

LOBYTE

Funktion – Ruft das niederwertige Byte eines Wortes ab.

Eingänge

IN : UINT 16-Bit-Register.

Ausgänge

Q : USINT Das niedrigstwertige Byte.

Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

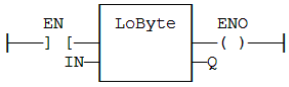
ST-Sprache

```
Q := LOBYTE (IN);
```

FBD-Sprache

KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        LOBYTE
        ST      Q
```

Siehe auch

HIBYTE LOWORD HIWORD MAKEWORD MAKEDWORD

LOG

Funktion – Berechnet den Logarithmus (Basis 10) des Eingangs.

Eingänge

IN : REAL Reeller Zahlenwert.

Ausgänge

Q : REAL Ergebnis: Logarithmus (Basis 10) von IN.

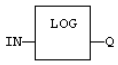
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

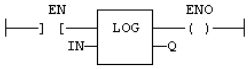
```
Q := LOG (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD    IN
        LOG
        ST    Q (* Q ist: LOG (IN) *)
```

Siehe auch

ABS TRUNC POW SQRT

LOWORD

Funktion – Ruft das niederwertige Wort eines Doppelworts ab.

Eingänge

IN : UDINT 32-Bit-Register.

Ausgänge

Q : UINT Das niedrigstwertige Wort.

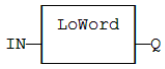
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

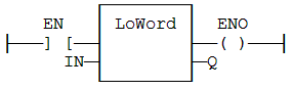
```
Q := LOWORD (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        LOWORD
        ST      Q
```

Siehe auch

LOBYTE HIBYTE HIWORD MAKEWORD MAKEDWORD

<LT

Operator – Test, ob der erste Eingang kleiner ist als der zweite Eingang.

Eingänge

IN1 : ANY Erster Eingang.
IN2 : ANY Zweiter Eingang.

Ausgänge

Q : BOOL *TRUE*, wenn IN1 < IN2.

Anmerkungen

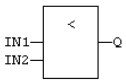
Beide Eingänge müssen vom gleichen Typ sein. In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe ist das Ergebnis des Vergleichs. In AWL-Sprache führt der LT-Befehl den Vergleich zwischen dem aktuellen Ergebnis und dem Operanden durch. Das aktuelle Ergebnis und der Operand müssen vom gleichen Typ sein.

Vergleiche können mit Zeichenfolgen verwendet werden. In diesem Fall wird die lexikalische Reihenfolge zum Vergleichen der Eingangszeichenfolgen verwendet. Beispiel: „ABC" ist kleiner als „ZX"; „ABCD" ist größer als „ABC".

ST-Sprache

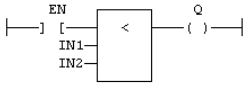
```
Q := IN1 < IN2;
```

FBD-Sprache



KOP-Sprache

Der Vergleich wird nur ausgeführt, wenn EN *TRUE* ist:



AWL-Sprache

```
Op1:    LD    IN1
        LT    IN2
        ST    Q (* Q ist true wenn IN1 < IN2 *)
```

Siehe auch

> GT >= GE <= LE = EQ <> NE CMP

MAKEDWORD

Funktion – Legt ein Doppelwort als Verkettung von zwei Wörtern fest.

Eingänge

HI : UINT Das höchstwertige Wort.
LO : UINT Das niedrigstwertige Wort.

Ausgänge

Q : UDINT 32-Bit-Register.

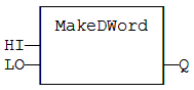
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der erste Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

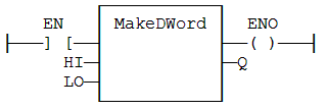
```
Q := MAKEDWORD (HI, LO);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD          HI
        MAKEDWORD   LO
        ST          Q
```

Siehe auch

LOBYTE HIBYTE LOWORD HIWORD MAKEWORD

MAKEWORD

Funktion – Erstellt ein Wort als Verkettung von zwei Bytes.

Eingänge

HI : USINT Das höchstwertige Byte.
LO : USINT Das niedrigstwertige Byte.

Ausgänge

Q : UINT 16-Bit-Register.

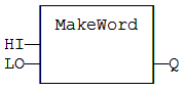
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der erste Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

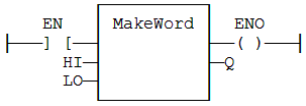
```
Q := MAKEWORD (HI, LO);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD          HI
        MAKEWORD    LO
        ST          Q
```

Siehe auch

LOBYTE HIBYTE LOWORD HIWORD MAKEDWORD

MAX

Funktion – Ruft den höheren von zwei Werten ab.

Eingänge

IN1 : ANY Erster Eingang.
IN2 : ANY Zweiter Eingang.

Ausgänge

Q : ANY IN1 wenn IN1 > IN2; andernfalls IN2.

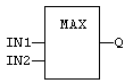
Anmerkungen

In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den Status des Eingangsstrompfads bei. In der AWL-Sprache muss der erste Eingang vor dem Funktionsaufruf geladen werden. Der zweite Eingang ist der Operand der Funktion.

ST-Sprache

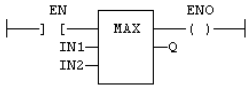
```
Q := MAX (IN1, IN2);
```

FBD-Sprache



KOP-Sprache

Der Vergleich wird nur ausgeführt, wenn EN *TRUE* ist.
ENO hat den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN1
        MAX     IN2
        ST      Q (* Q ist das Maximum von IN1 und IN2 *)
```

Siehe auch

MIN LIMIT MOD ODD

MBSHIFT

Funktion – Multibyte-Verschiebung/-Rotation.

Eingänge

Buffer : SINT/USINT Byte-Array.
Pos : DINT Basisposition im Array.
NbByte : DINT Anzahl der zu verschiebenden/zu rotierenden Bytes.
NbShift : DINT Anzahl der Verschiebungen oder Rotationen.
ToRight : BOOL *TRUE* für rechts / *FALSE* für links.
Rotate : BOOL *TRUE* für Rotieren / *FALSE* für Verschieben.
InBit : BOOL Bei einer Verschiebung einzusetzendes Bit.

Ausgänge

Q : BOOL *TRUE*, wenn erfolgreich.

Anmerkungen

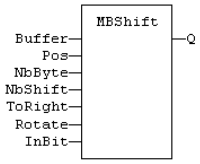
Verwenden Sie das Argument ToRight, um eine Verschiebung nach links (*FALSE*) oder nach rechts (*TRUE*) anzugeben. Verwenden Sie das Argument Rotate, um entweder eine Verschiebung (*FALSE*) oder eine Rotation (*TRUE*) anzugeben. Im Falle einer Verschiebung gibt das Argument InBit den Wert des Bits an, welches das zuletzt verschobene Bit ersetzt.

In der KOP-Sprache validiert der Stufeneingang (EN) den Vorgang. Der Stufenausgang ist das Ergebnis (Q).

ST-Sprache

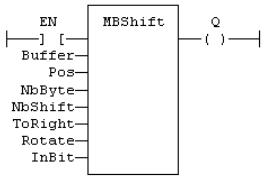
```
Q := MBShift (Buffer, Pos, NbByte, NbShift, ToRight, Rotate, InBit);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur aufgerufen, wenn EN *TRUE* ist.



AWL-Sprache

Nicht verfügbar.

MID

Funktion – Extrahiert Zeichen einer Zeichenfolge an einer beliebigen Position.

Eingänge

IN : STRING Zeichenfolge.
NBC : DINT Anzahl der zu extrahierenden Zeichen.
POS : DINT Position des ersten zu extrahierenden Zeichens (erstes Zeichen von IN ist an Position 1).

Ausgänge

Q : STRING Zeichenfolge, die die ersten NBC-Zeichen von IN enthält.

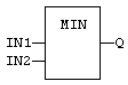
Anmerkungen

Die erste gültige Position ist 1. In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der erste Eingang (die Zeichenfolge) in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann. Andere Argumente sind Operanden der Funktion, die durch Kommas getrennt sind.

ST-Sprache

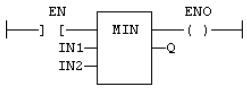
```
Q := MID (IN, NBC, POS);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        MID     NBC, POS
        ST      Q
```

Siehe auch

+ ADD MLEN DELETE INSERT FIND REPLACE LEFT RIGHT

MIN

Funktion – Ruft den niedrigeren von zwei Werten ab.

Eingänge

IN1 : ANY Erster Eingang.
IN2 : ANY Zweiter Eingang.

Ausgänge

Q : ANY IN1 wenn IN1 < IN2; andernfalls IN2.

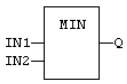
Anmerkungen

In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den Status des Eingangsstrompfads bei. In der AWL-Sprache muss der erste Eingang vor dem Funktionsaufruf geladen werden. Der zweite Eingang ist der Operand der Funktion.

ST-Sprache

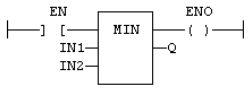
```
Q := MIN (IN1, IN2);
```

FBD-Sprache



KOP-Sprache

Der Vergleich wird nur ausgeführt, wenn EN *TRUE* ist.
ENO hat den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN1
        MIN     IN2
        ST      Q (* Q ist das Minimum von IN1 und IN2 *)
```

Siehe auch

MAX LIMIT MOD ODD

MLEN

Funktion – Ruft die Anzahl der Zeichen in einer Zeichenfolge ab.

Eingänge

IN : STRING Zeichenfolge.

Ausgänge

NBC : DINT Anzahl der Zeichen, die sich derzeit in der Zeichenfolge befinden. 0, wenn die Zeichenfolge leer ist.

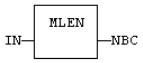
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

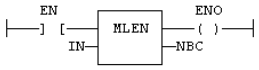
```
NBC := MLEN (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD    IN
        MLEN
        ST    NBC
```

Siehe auch

+ ADD DELETE INSERT FIND REPLACE LEFT RIGHT MID

MOD / MODR / MODLR

Funktion – Berechnet den Teilungsrest.

Eingänge

IN : DINT/REAL/LREAL Eingangswert.
BASE : DINT/REAL/LREAL Basis des Teilungsrests.

Ausgänge

Q : DINT/REAL/LREAL Teilungsrest: Rest der Ganzzahl-Division (IN / BASE).

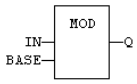
Anmerkungen

In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den Status des Eingangsstrompfads bei. In der AWL-Sprache muss der erste Eingang vor dem Funktionsaufruf geladen werden. Der zweite Eingang ist der Operand der Funktion.

ST-Sprache

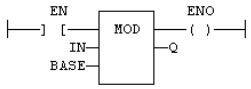
```
Q := MOD (IN, BASE);
```

FBD-Sprache



KOP-Sprache

Der Vergleich wird nur ausgeführt, wenn EN *TRUE* ist.
ENO hat den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        MOD     BASE
        ST      Q (* Q ist der Rest der Ganzzahl-Division: IN / BASE *)
```

Siehe auch

MIN MAX LIMIT ODD

* MUL

Operator – Führt eine Multiplikation aller Eingänge durch.

Eingänge

IN1 : ANY_NUM Erster Eingang.
IN2 : ANY_NUM Zweiter Eingang.

Ausgänge

Q : ANY_NUM Ergebnis: IN1 * IN2.

Anmerkungen

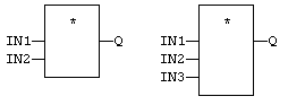
Alle Eingänge und der Ausgang müssen vom gleichen Typ sein. In der FBD-Sprache kann der Block bis zu 16 Eingänge haben. In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den gleichen Wert bei wie die Eingangsstufe. In AWL-Sprache multipliziert der MUL-Befehl das aktuelle Ergebnis mit dem Operanden. Das aktuelle Ergebnis und der Operand müssen vom gleichen Typ sein.

ST-Sprache

```
Q := IN1 * IN2;
```

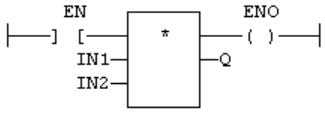
FBD-Sprache

Der Block kann bis zu 16 Eingänge haben:



KOP-SPRACHE

Die Multiplikation wird nur ausgeführt, wenn EN *TRUE* ist.
ENO ist gleich EN.



AWL-Sprache

```
Op1:    LD      IN1
        MUL     IN2
        ST      Q (* Q ist gleich: IN1 * IN2 *)
Op2:    LD      IN1
        MUL     IN2
        MUL     IN3
        ST      Q (* Q ist gleich: IN1 * IN2 * IN3 *)
```

Siehe auch

+ ADD - SUB / DIV

MUX4

Funktion – Wählt einen der Eingänge aus – 4 Eingänge.

Eingänge

SELECT : DINT Auswahlbefehl.
IN1 : ANY Erster Eingang.
IN2 : ANY Zweiter Eingang.
... :
IN4 : ANY Letzter Eingang.

Ausgänge

Q : ANY IN1 oder IN2 ... oder IN4 je nach SELECT (siehe Wahrheitstabelle).

Wahrheitstabelle

SELECT	Q
0	IN1
2	IN2
3	IN3
4	IN4
Andere	0

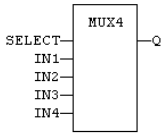
Anmerkungen

In der KOP-Sprache aktiviert die Eingangsstufe (EN) die Auswahl. Die Ausgangsstufe behält den gleichen Status bei wie die Eingangsstufe. In AWL-Sprache muss der erste Parameter (Selektor) in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann. Andere Eingänge sind Operanden der Funktion, die durch Kommas getrennt sind.

ST-Sprache

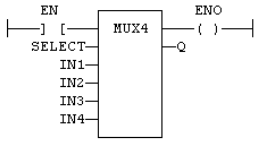
Q := MUX4 (SELECT, IN1, IN2, IN3, IN4);

FBD-Sprache



KOP-Sprache

Die Auswahl wird nur durchgeführt, wenn EN *TRUE* ist.
ENO hat den gleichen Wert wie EN.



AWL-Sprache

Op1:	LD	SELECT
	MUX4	IN1, IN2, IN3, IN4
	ST	Q

Siehe auch

SEL MUX8

MUX8

Funktion – Wählt einen der Eingänge aus – 8 Eingänge.

Eingänge

SELECT : DINT Auswahlbefehl.
IN1 : ANY Erster Eingang.
IN2 : ANY Zweiter Eingang.
... :
IN8 : ANY Letzter Eingang.

Ausgänge

Q : ANY IN1 oder IN2 ... oder IN8 je nach SELECT (siehe Wahrheitstabelle).

Wahrheitstabelle

SELECT	Q
0	IN1
1	IN2
2	IN3
3	IN4
4	IN5
5	IN6
6	IN7
7	IN8
Andere	0

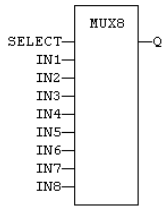
Anmerkungen

In der KOP-Sprache aktiviert die Eingangsstufe (EN) die Auswahl. Die Ausgangsstufe behält den gleichen Status bei wie die Eingangsstufe. In AWL-Sprache muss der erste Parameter (Selektor) in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann. Andere Eingänge sind Operanden der Funktion, die durch Kommas getrennt sind.

ST-Sprache

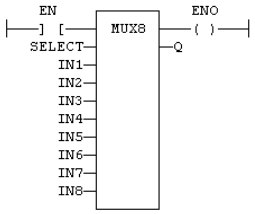
Q := MUX8 (SELECT, IN1, IN2, IN3, IN4, IN5, IN6, IN7, IN8);

FBD-Sprache



KOP-Sprache

Die Auswahl wird nur durchgeführt, wenn EN TRUE ist.
ENO hat den gleichen Wert wie EN.



AWL-Sprache

Op1:	LD	SELECT
	MUX8	IN1, IN2, IN3, IN4, IN5, IN6, IN7, IN8
	ST	Q

Siehe auch

SEL MUX4

<> NE

Operator – Test, ob der erste Eingang nicht gleich dem zweiten Eingang ist.

Eingänge

IN1 : ANY Erster Eingang.
IN2 : ANY Zweiter Eingang.

Ausgänge

Q : BOOL *TRUE*, wenn IN1 nicht gleich IN2 ist.

Anmerkungen

Beide Eingänge müssen vom gleichen Typ sein. In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe ist das Ergebnis des Vergleichs. In AWL-Sprache führt der NE-Befehl den Vergleich zwischen dem aktuellen Ergebnis und dem Operanden durch. Das aktuelle Ergebnis und der Operand müssen vom gleichen Typ sein.

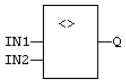
Vergleiche können mit Zeichenfolgen verwendet werden. In diesem Fall wird die lexikalische Reihenfolge zum Vergleichen der Eingangszeichenfolgen verwendet. Beispiel: „ABC" ist kleiner als „ZX"; „ABCD" ist größer als „ABC".

Vergleiche mit „gleich" können nicht mit TIME-Variablen verwendet werden. Der Grund dafür ist, dass der Timer tatsächlich die Auflösung des Zielzyklus aufweist und der Test unsicher sein kann, da einige Werte möglicherweise nie erreicht werden.

ST-Sprache

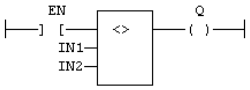
```
Q := IN1 <> IN2;
```

FBD-Sprache



KOP-Sprache

Der Vergleich wird nur ausgeführt, wenn EN *TRUE* ist:



AWL-Sprache

```
Op1:    LD    IN1
        NE    IN2
        ST    Q (* Q ist true, wenn IN1 ungleich IN2 * ist)
```

Siehe auch

> GT < LT >= GE <= LE = EQ CMP

NEG -

Operator – Führt eine Ganzzahl-Negation des Eingangs durch.

Eingänge

IN : DINT Ganzzahliger Wert.

Ausgänge

Q : DINT Ganzzahl-Negation des Eingangs.

Wahrheitstabelle (Beispiele)

IN	Q
0	0
1	-1
-123	123

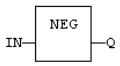
Anmerkungen

In den Sprachen FBD und KOP kann der Block NEG verwendet werden. In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. Diese Funktion ist in AWL-Sprache nicht verfügbar. In der ST-Sprache kann auf „-" ein komplexer boolescher Ausdruck zwischen Klammern folgen.

ST-Sprache

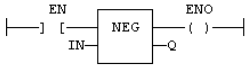
```
Q := -IN;
Q := - (IN1 + IN2);
```

FBD-Sprache



KOP-Sprache

Die Negation wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

Nicht verfügbar.

NOT

Operator – Führt eine boolesche Negation des Eingangs durch.

Eingänge

IN : BOOL Boolescher Wert.

Ausgänge

Q : BOOL Boolesche Negation des Eingangs.

Wahrheitstabelle

IN	Q
0	1
1	0

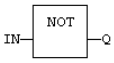
Anmerkungen

In der FBD-Sprache kann der Block NOT verwendet werden. Alternativ können Sie auch einen Link verwenden, der durch eine o-Negation beendet wird. In der KOP-Sprache können negierte Kontakte und Spulen verwendet werden. In AWL-Sprache kann der Modifikator N mit den Befehlen LD, AND, OR, XOR und ST verwendet werden. Dies stellt eine Negation des Operanden dar. In der ST-Sprache kann auf NOT ein komplexer boolescher Ausdruck zwischen Klammern folgen.

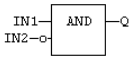
ST-Sprache

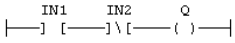
```
Q := NOT IN;
Q := NOT (IN1 OR IN2);
```

FBD-Sprache



Explizite Verwendung des NOT-Blocks:

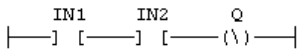




Verwendung eines negierten Links: Q ist IN1 AND NOT IN2:

KOP-SPRACHE

Negierter Kontakt: Q ist: IN1 AND NOT IN2:



Negierte Spule: Q ist NOT (IN1 AND IN2):

AWL-Sprache

```
Op1:    LDN  IN1
        OR   IN2
        ST   Q (* Q ist gleich: (NOT IN1) OR IN2 *)
Op2:    LD   IN1
        AND  IN2
        STN  Q (* Q ist gleich: NOT (IN1 AND IN2) *)
```

Siehe auch

AND OR XOR

NOT_MASK

Funktion – Führt eine Bit-zu-Bit-Negation eines Ganzzahlwerts durch.

Eingänge

IN : ANY Ganzzahleingang.

Ausgänge

Q : ANY Bit-zu-Bit-Negation des Eingangs.

Anmerkungen

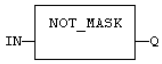
Argumente können aus Ganzzahlen mit oder ohne Vorzeichen von 8 bis 32 Bits bestehen.

In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den gleichen Wert bei wie die Eingangsstufe. In AWL-Sprache muss der Parameter (IN) in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

```
Q := NOT_MASK (IN);
```

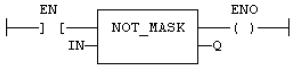
FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.

ENO ist gleich EN.



AWL-Sprache

```
Op1:    LD    IN
        NOT_MASK
        ST    Q
```

Siehe auch

AND_MASK OR_MASK XOR_MASK

NUM_TO_STRING

Funktion – Konvertiert eine Zahl in einen Zeichenfolgenwert.

Eingänge

IN : ANY Eingangszahl.
WIDTH : DINT Gewünschte Länge für die Ausgangszeichenfolge (siehe Anmerkungen)
DIGITS : DINT Anzahl der Stellen nach dem Dezimalzeichen.

Ausgänge

Q : STRING Wert in Zeichenfolge konvertiert.

Anmerkungen

Diese Funktion konvertiert jeden numerischen Wert in eine Zeichenfolge. Im Gegensatz zur Funktion ANY_TO_STRING können Sie hier eine gewünschte Länge und eine Anzahl von Stellen nach dem Dezimalzeichen angeben.

Wenn WIDTH 0 ist, wird die Zeichenfolge mit der erforderlichen Länge formatiert.

Wenn WIDTH größer als 0 ist, wird die Zeichenfolge mit vorangestellten Leerzeichen vervollständigt, um dem Wert von WIDTH zu entsprechen.

Wenn WIDTH größer als 0 ist, wird die Zeichenfolge mit nachgestellten Leerzeichen vervollständigt, um dem absoluten Wert von WIDTH zu entsprechen.

Wenn DIGITS gleich 0 ist, werden weder Dezimalstellen noch Dezimalzeichen hinzugefügt.

Wenn DIGITS größer als 0 ist, wird die entsprechende Anzahl Dezimalstellen hinzugefügt. Bei Bedarf werden „0” Ziffern hinzugefügt.

Wenn der Wert für die angegebene Breite zu lang ist, wird die Zeichenfolge mit „*”-Zeichen aufgefüllt.

Beispiele

```
Q := NUM_TO_STRING (123.4, 8, 2);(* Q ist ' 123.40' *)
Q := NUM_TO_STRING (123.4, -8, 2);(* Q ist '123.40 ' *)
Q := NUM_TO_STRING (1.333333, 0, 2);(* Q ist '1.33' *)
Q := NUM_TO_STRING (1234, 3, 0); (* Q ist '***' *)
```

ODD

Funktion – Test, ob eine ganze Zahl ungerade ist.

Eingänge

IN : DINT Eingangswert.

Ausgänge

Q : BOOL *TRUE*, wenn IN ungerade ist. FALSE, wenn IN gerade ist.

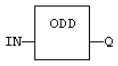
Anmerkungen

In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe ist das Ergebnis der Funktion. In der AWL-Sprache muss der Eingang vor dem Funktionsaufruf geladen werden.

ST-Sprache

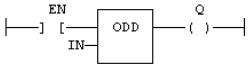
```
Q := ODD (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.



AWL-Sprache

```
Op1:    LD    IN
        ODD
        ST    Q (* Q ist TRUE, wenn IN ungerade ist *)
```

Siehe auch

MIN MAX LIMIT MOD

OR / ORN

Operator – Führt ein logisches OR aller Eingänge aus.

Eingänge

IN1 : BOOL Erster boolescher Eingang.
IN2 : BOOL Zweiter boolescher Eingang.

Ausgänge

Q : BOOL Boolesches OR aller Eingänge.

Wahrheitstabelle

IN2	IN2	Q
0	0	0
0	1	1
1	0	1
1	1	1

Anmerkungen

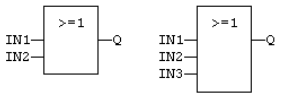
In der FBD-Sprache kann der Block bis zu 16 Eingänge haben. Der Block wird in FBD-Sprache als „>=1" bezeichnet. In der KOP-Sprache wird ein OR-Vorgang durch parallele Kontakte dargestellt. In der AWL-Sprache führt der OR-Befehl ein logisches OR zwischen dem aktuellen Ergebnis und dem Operanden aus. Das aktuelle Ergebnis muss ein boolescher Wert sein. Der ORN-Befehl führt ein OR zwischen dem aktuellen Ergebnis und der booleschen Negation des Operanden aus.

ST-Sprache

```
Q := IN1 OR IN2;
Q := IN1 OR IN2 OR IN3;
```

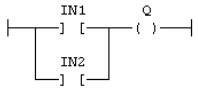
FBD-Sprache

Der Block kann bis zu 16 Eingänge haben:



KOP-Sprache

Parallele Kontakte:



AWL-Sprache

```
Op1:    LD      IN1
        OR      IN2
        ST      Q (* Q ist gleich: IN1 OR IN2 *)
Op2:    LD      IN1
        ORN     IN2
        ST      Q (* Q ist gleich: IN1 OR (NOT IN2) *)
```

Siehe auch

AND XOR NOT

OR_MASK

Funktion – Führt ein Bit-zu-Bit-OR zwischen zwei Ganzzahlwerten aus.

Eingänge

IN : ANY Erster Eingang.
MSK : ANY Zweiter Eingang (OR-Maske).

Ausgänge

Q : ANY OR-Maske zwischen IN- und MSK-Eingängen.

Anmerkungen

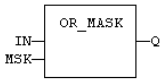
Argumente können aus Ganzzahlen mit oder ohne Vorzeichen von 8 bis 32 Bits bestehen.

In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den gleichen Wert bei wie die Eingangsstufe. In AWL-Sprache muss der erste Parameter (IN) in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann. Der andere Eingang sind die Operanden der Funktion.

ST-Sprache

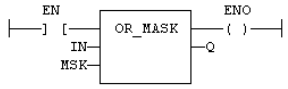
```
Q := OR_MASK (IN, MSK);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO ist gleich EN.



AWL-Sprache

```
Op1:    LD          IN
        OR_MASKMSK
        ST          Q
```

Siehe auch

AND_MASK XOR_MASK NOT_MASK

PACK8

Funktion – Erstellt ein Byte mit Bits.

Eingänge

IN0 : BOOL Niederwertiges Bit.
...
IN7 : BOOL Höchstwertiges Bit.

Ausgänge

Q : USINT Mit Eingangsbits erstelltes Byte.

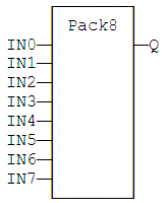
Anmerkungen

In der KOP-Sprache ist die Eingangsstufe der IN0-Eingang. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

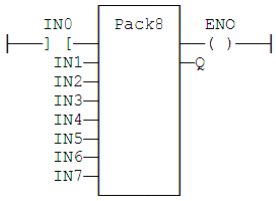
Q := PACK8 (IN0, IN1, IN2, IN3, IN4, IN5, IN6, IN7)

FBD-Sprache



KOP-Sprache

ENO behält den gleichen Wert wie EN.



AWL-Sprache

Op1:	LD	IN0
	PACK8	IN1, IN2, IN3, IN4, IN5, IN6, IN7
	ST	Q

Siehe auch

UNPACK8

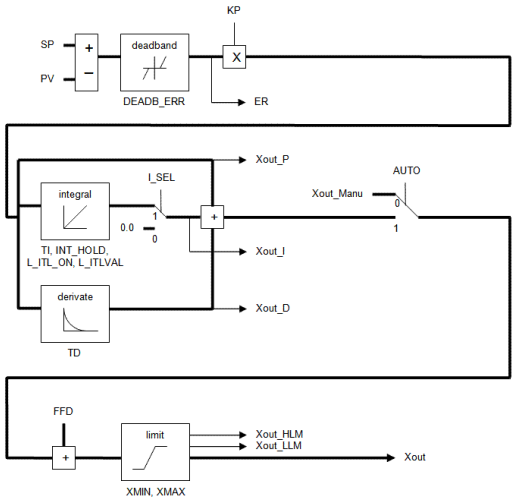
PID

Funktionsblock – PID-Regelkreis.

INPUT	TYP	BESCHREIBUNG
AUTO	BOOL	*TRUE* = Normalbetrieb – *FALSE* = manueller Modus.
PV	REAL	Prozesswert.
SP	REAL	Sollwert.
Xout_Manu	REAL	Ausgangswert im manuellen Modus.
KP	REAL	Verstärkung.
TI	REAL	Integrationsfaktor.
TD	REAL	Ableitungsfaktor.
TS	TIME	Probenahmezeitraum.
XMIN	REAL	Minimal zulässiger Ausgangswert.
XMAX	REAL	Maximaler Ausgangswert.
I_SEL	BOOL	Bei *FALSE* wird der integrierte Wert ignoriert.
INT_HOLD	BOOL	*TRUE* gibt an, dass der integrierte Wert eingefroren ist.
I_ITL_ON	BOOL	*TRUE* gibt an, dass der integrierte Wert auf I_ITLVAL zurückgesetzt wird.
I_ITLVAL	REAL	Wert für Integration zurücksetzen, wenn I_ITL_ON *TRUE* ist.
DEADB_ERR	REAL	Hysterese am PV. PV wird als unverändert angenommen, wenn größer als (PVprev - DEADBAND_W) und kleiner als (PRprev + DEADBAND_ W).
FFD	REAL	Störgröße am Ausgang.

OUTPUT	TYP	BESCHREIBUNG
Xout	REAL	Ausgangsbefehlswert.
ER	REAL	Letzter berechneter Fehler.
Xout_P	REAL	Letzter berechneter Anteilswert.
Xout_I	REAL	Letzter berechneter integrierter Wert.
Xout_D	REAL	Letzter berechneter abgeleiteter Wert.
Xout_HLM	BOOL	*TRUE*, wenn der Ausgangswert auf XMIN gesättigt wird.
Xout_LLM	BOOL	*TRUE*, wenn der Ausgangswert auf XMAX gesättigt wird.

Diagramm



Anmerkungen

Für die Stabilität der Kontrolle ist es wichtig, dass der TS-Probenahmezeitraum viel größer ist als die Zykluszeit.

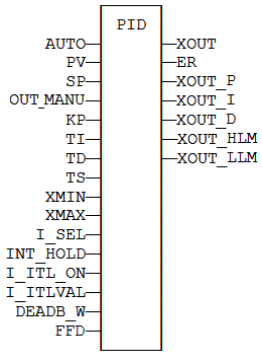
In der KOP-Sprache hat die Ausgangsstufe denselben Wert wie der AUTO-Eingang, der dem Eingangsstrompfad entspricht.

ST-Sprache

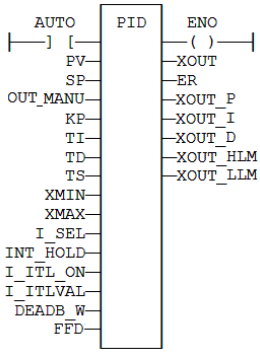
MyPID ist eine deklarierte Instanz des Funktionsblocks PID.

```
MyPID (AUTO, PV, SP, XOUT_MANU, KP, TI, TD, TS, XMIN, XMAX,
I_SEL, I_ITL_ON, I_ITLVAL, DEADB_ERR, FFD);
XOUT := MyPID.XOUT;
ER := MyPID.ER;
XOUT_P := MyPID.XOUT_P;
XOUT_I := MyPID.XOUT_I;
XOUT_D := MyPID.XOUT_D;
XOUT_HLM := MyPID.XOUT_HLM;
XOUT_LLM := MyPID.XOUT_LLM;
```

FBD-Sprache



KOP-Sprache



ENO hat den gleichen Status wie die Eingangsstufe.

AWL-Sprache

MyPID ist eine deklarierte Instanz des Funktionsblocks PID.

```
Op1:    CAL   MyPID (AUTO, PV, SP, XOUT_MANU, KP, TI, TD, TS,
                     XMIN, XMAX, I_SEL, I_ITL_ON, I_ITLVAL,
                     DEADB_ERR, FFD)
        LD    MyPID.XOUT
        ST    XOUT
        LD    MyPID.ER
        ST    ER
        LD    MyPID.XOUT_P
        ST    XOUT_P
        LD    MyPID.XOUT_I
        ST    XOUT_I
        LD    MyPID.XOUT_D
        ST    XOUT_D
        LD    MyPID.XOUT_HLM
        ST    XOUT_HLM
        LD    MyPID.XOUT_LLM
        ST    XOUT_LLM
```

PLS

Funktionsblock – Impulssignalgenerator:

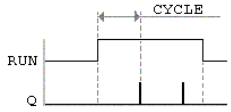
Eingänge

RUN : BOOL Aktivierungsbefehl.
CYCLE : TIME Signaldauer.

Ausgänge

Q : BOOL Ausgangs-Impulssignal.

Zeitdiagramm



Anmerkungen

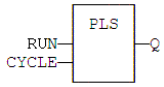
In jedem Zeitraum wird der Ausgang für genau einen Zyklus auf TRUE gesetzt. In der KOP-Sprache ist die Eingangsstufe der IN-Befehl. Die Ausgangsstufe ist das Q-Ausgangssignal.

ST-Sprache

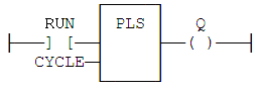
MyPLS ist eine deklarierte Instanz des Funktionsblocks PLS:

```
MyPLS (RUN, CYCLE);
Q := MyPLS.Q;
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

MyPLS ist eine deklarierte Instanz des Funktionsblocks PLS:

```
Op1:    CAL     MyPLS (RUN, CYCLE)
        LD      MyPLS.Q
        ST      Q
```

Siehe auch

TON TOF TP

POW ** POWL

Funktion – Berechnet eine Potenz.

Eingänge

IN : REAL/LREAL Reeller Zahlenwert.
EXP : REAL/LREAL Exponent.

Ausgänge

Q : REAL/LREAL Ergebnis: Potenz 'EXP' von IN.

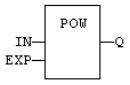
Anmerkungen

Alternativ kann in der ST-Sprache auch der Operator ** verwendet werden. In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann. Der Exponent (zweiter Eingang der Funktion) muss der Operand der Funktion sein.

ST-Sprache

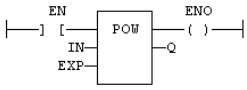
```
Q := POW (IN, EXP);
Q := IN ** EXP;
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        POW     EXP
        ST      Q (* Q ist: (IN ** EXP) *)
```

Siehe auch

ABS TRUNC LOG SQRT

QOR

Operator – Zählt die Anzahl der *TRUE*-Eingänge.

Eingänge

IN1 .. INn : BOOL Boolesche Eingänge.

Ausgänge

Q : DINT Anzahl der Eingänge, die *TRUE* sind.

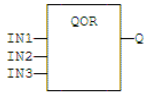
Anmerkungen

Der Block akzeptiert eine nicht festgelegte Anzahl von Eingängen.

ST-Sprache

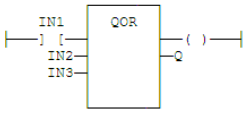
```
Q := QOR (IN1, IN2);
Q := QOR (IN1, IN2, IN3, IN4, IN5, IN6);
```

FBD-Sprache



Der Block kann bis zu 16 Eingänge haben:

KOP-Sprache



Der Block kann bis zu 16 Eingänge haben:

AWL-Sprache

```
Op1:    LD      IN1
        QOR     IN2,   IN3
        ST      Q
```

R

Operator – Zwingt einen booleschen Ausgang auf *FALSE*.

Eingänge

RESET : BOOL Bedingung.

Ausgänge

Q : BOOL Zu erzwingender Ausgang.

Wahrheitstabelle

RESET	Q VORH.	Q
0	0	0
0	1	1
1	0	0
1	1	0

Anmerkungen

S- und R-Operatoren sind in der IL-Sprache als Standardanweisungen verfügbar. In KOP-Sprachen werden sie durch (S)- und (R)-Spulen dargestellt. In der FBD-Sprache können Sie (S)- und (R)-Spulen verwenden, aber Sie sollten RS- und SR-Funktionsblöcke bevorzugen. Set- und Reset-Vorgänge sind in der ST-Sprache nicht verfügbar.

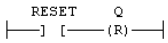
ST-Sprache

Nicht verfügbar.

FBD-Sprache

Nicht verfügbar. Verwenden Sie RS- oder SR-Funktionsblöcke.

KOP-Sprache



Verwendung der "R"-Spule:

AWL-Sprache

```
Op1:    LD    RESET
        R     Q (* Q wird auf FALSE gesetzt, wenn RESET TRUE ist *)
              (* Q bleibt unverändert, wenn RESET FALSE ist *)
```

Siehe auch

S RS SR

R_TRIG

Funktionsblock – Erkennung ansteigender Impulse.

Eingänge

CLK : BOOL Boolesches Signal.

Ausgänge

Q : BOOL *TRUE*, wenn der Eingang von *FALSE* zu *TRUE* wechselt.

Wahrheitstabelle

CLK	CLK VORH.	Q
0	0	0
0	1	0
1	0	1
1	1	0

Anmerkungen

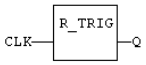
Obwohl die Kontakte vom Typ]P[und]N[in der KOP-Sprache verwendet werden können, wird empfohlen, deklarierte Instanzen der Funktionsblöcke R_TRIG oder F_TRIG zu verwenden, um ein unerwartetes Verhalten während einer Online-Änderung zu vermeiden.

ST-Sprache

MyTrigger wird als Instanz des Funktionsblocks R_TRIG deklariert:

```
MyTrigger (CLK);
Q := MyTrigger.Q;
```

FBD-Sprache



KOP-Sprache

Das Eingangssignal ist die Stufe – die Stufe ist der Ausgang:

AWL-Sprache

MyTrigger wird als Instanz des Funktionsblocks R_TRIG deklariert:

```
Op1:    CAL   MyTrigger (CLK)
        LD    MyTrigger.Q
        ST    Q
```

Siehe auch

F_TRIG

REPLACE

Funktion – Ersetzt Zeichen in einer Zeichenfolge.

Eingänge

IN : STRING Zeichenfolge.
STR : STRING Zeichenfolge, welche die einzufügenden Zeichen enthält.
Anstelle von NDEL entfernte Zeichen.
NDEL : DINT Anzahl der Zeichen, die vor dem Einfügen von STR gelöscht werden sollen.
POS : DINT Position, an der Zeichen ersetzt werden (erste Zeichenposition ist 1).

Ausgänge

Q : STRING Geänderte Zeichenfolge.

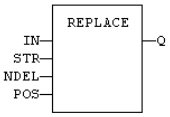
Anmerkungen

Die erste gültige Zeichenposition ist 1. In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der erste Eingang (die Zeichenfolge) in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann. Andere Argumente sind Operanden der Funktion, die durch Kommas getrennt sind.

ST-Sprache

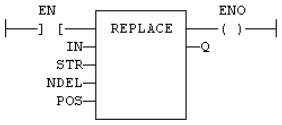
```
Q := REPLACE (IN, STR, NDEL, POS);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        REPLACE STR, NDEL, POS
        ST      Q
```

Siehe auch

+ ADD MLEN DELETE INSERT FIND LEFT RIGHT MID

RIGHT

Funktion – Extrahiert Zeichen einer Zeichenfolge auf der rechten Seite.

Eingänge

IN : STRING Zeichenfolge.
NBC : DINT Anzahl der zu extrahierenden Zeichen.

Ausgänge

Q : STRING Zeichenfolge, die die letzten NBC-Zeichen von IN enthält.

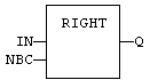
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der erste Eingang (die Zeichenfolge) in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann. Das zweite Argument ist der Operand der Funktion.

ST-Sprache

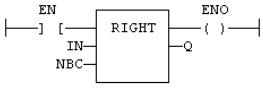
```
Q := RIGHT (IN, NBC);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        RIGHT   NBC
        ST      Q
```

Siehe auch

+ ADD MLEN DELETE INSERT FIND REPLACE LEFT MID

ROL

Funktion – Rotiert Bits in einem Register nach links.

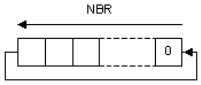
Eingänge

IN : ANY Register.
NBR : ANY Anzahl der Rotationen (jede Rotation ist 1 Bit).

Ausgänge

Q : ANY Rotiertes Register.

Diagramm



Anmerkungen

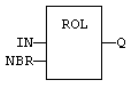
Argumente können aus Ganzzahlen mit oder ohne Vorzeichen von 8 bis 32 Bits bestehen.

In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den Status des Eingangsstrompfads bei. In der AWL-Sprache muss der erste Eingang vor dem Funktionsaufruf geladen werden. Der zweite Eingang ist der Operand der Funktion.

ST-Sprache

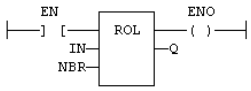
```
Q := ROL (IN, NBR);
```

FBD-Sprache



KOP-Sprache

Die Rotation wird nur ausgeführt, wenn EN *TRUE* ist.
ENO hat den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD    IN
        ROL   NBR
        ST    Q
```

Siehe auch

SHL SHR ROR SHLb SHRb ROLb RORb SHLw SHRw ROLw RORw

ROOT

Funktion – Berechnet die n-te Wurzel des Eingangs.

Eingänge

IN : REAL Reelle Zahl
N : DINT Basisebene

Ausgänge

Q : REAL Ergebnis: n-te Wurzel von IN

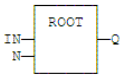
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

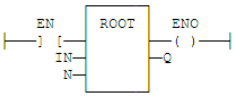
```
Q := ROOT (IN, N);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        ROOT    N
        ST      Q (* Q ist: ROOT (IN) *)
```

ROR

Funktion – Rotiert Bits in einem Register nach rechts.

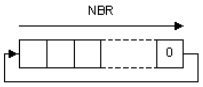
Eingänge

IN : ANY Register.
NBR : ANY Anzahl der Rotationen (jede Rotation ist 1 Bit).

Ausgänge

Q : ANY Rotiertes Register.

Diagramm



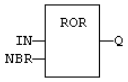
Anmerkungen

Argumente können aus Ganzzahlen mit oder ohne Vorzeichen von 8 bis 32 Bits bestehen.
In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den Status des Eingangsstrompfads bei. In der AWL-Sprache muss der erste Eingang vor dem Funktionsaufruf geladen werden. Der zweite Eingang ist der Operand der Funktion.

ST-Sprache

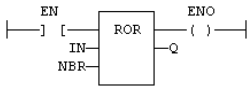
```
Q := ROR (IN, NBR);
```

FBD-Sprache



KOP-Sprache

Die Rotation wird nur ausgeführt, wenn EN *TRUE* ist.
ENO hat den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        ROR     NBR
        ST      Q
```

Siehe auch

SHL SHR ROL SHLb SHRb ROLb RORb SHLw SHRw ROLw RORw

RORb ROR_SINT ROR_USINT ROR_BYTE

Funktion – Rotiert Bits in einem Register nach rechts.

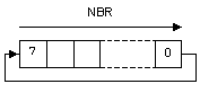
Eingänge

IN : SINT 8-Bit-Register.
NBR : SINT Anzahl der Rotationen (jede Rotation ist 1 Bit).

Ausgänge

Q : SINT Rotiertes Register.

Diagramm



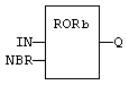
Anmerkungen

In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den Status des Eingangsstrompfads bei. In der AWL-Sprache muss der erste Eingang vor dem Funktionsaufruf geladen werden. Der zweite Eingang ist der Operand der Funktion.

ST-Sprache

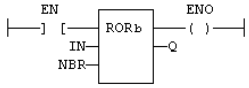
```
Q := RORb (IN, NBR);
```

FBD-Sprache



KOP-Sprache

Die Rotation wird nur ausgeführt, wenn EN *TRUE* ist.
ENO hat den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        RORb    NBR
        ST      Q
```

Siehe auch

SHL SHR ROL ROR SHLb SHRb ROLb SHLw SHRw ROLw RORw

RORw ROR_INT ROR_UINT ROR_WORD

Funktion – Rotiert Bits in einem Register nach rechts.

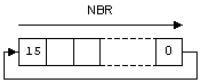
Eingänge

IN : INT 16-Bit-Register.
NBR : INT Anzahl der Rotationen (jede Rotation ist 1 Bit).

Ausgänge

Q : INT Rotiertes Register.

Diagramm



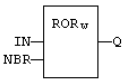
Anmerkungen

In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den Status des Eingangsstrompfads bei. In der AWL-Sprache muss der erste Eingang vor dem Funktionsaufruf geladen werden. Der zweite Eingang ist der Operand der Funktion.

ST-Sprache

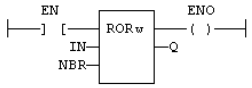
Q := RORw (IN, NBR);

FBD-Sprache



LD Language

Die Rotation wird nur ausgeführt, wenn EN *TRUE* ist.
ENO hat den gleichen Wert wie EN.



AWL-Sprache

Op1:	LD	IN
	RORw	NBR
	ST	Q

Siehe auch

SHL SHR ROL ROR SHLb SHRb ROLb RORb SHLw SHRw ROLw

RS

Funktionsblock – Setzt dominanten bistabilen Wert zurück.

Eingänge

SET : BOOL Bedingung für Erzwingen von *TRUE*.
RESET1 : BOOL Bedingung für Erzwingen von FALSE (Befehl mit höchster Priorität).

Ausgänge

Q1 : BOOL Zu erzwingender Ausgang.

Wahrheitstabelle

SET	RESET1	Q1 VORH.	Q1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Anmerkungen

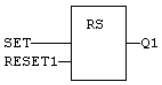
Der Ausgang bleibt unverändert, wenn beide Eingänge *FALSE* sind. Wenn beide Eingänge *TRUE* sind, wird der Ausgang auf *FALSE* gesetzt (dominanten Wert zurücksetzen).

ST-Sprache

MyRS wird als Instanz des Funktionsblocks RS deklariert:

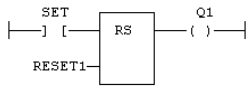
```
MyRS (SET, RESET1);
Q1 := MyRS.Q1;
```

FBD-Sprache



KOP-Sprache

Der SET-Befehl ist die Stufe – die Stufe ist der Ausgang:



AWL-Sprache

MyRS wird als Instanz des Funktionsblocks RS deklariert:

```
Op1:    CAL   MyRS (SET, RESET1)
        LD    MyRS.Q1
        ST    Q1
```

Siehe auch

R S SR

S

Operator – Zwingt einen booleschen Ausgang auf *TRUE*.

Eingänge

SET : BOOL Bedingung.

Ausgänge

Q : BOOL Zu erzwingender Ausgang.

Wahrheitstabelle

SET	Q VORH.	Q
0	0	0
0	1	1
1	0	1
1	1	1

Anmerkungen

S- und R-Operatoren sind in der IL-Sprache als Standardanweisungen verfügbar. In KOP-Sprachen werden sie durch (S)- und (R)-Spulen dargestellt. In der FBD-Sprache können Sie (S)- und (R)-Spulen verwenden, aber Sie sollten RS- und SR-Funktionsblöcke bevorzugen. Set- und Reset-Vorgänge sind in der ST-Sprache nicht verfügbar.

ST-Sprache

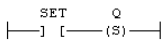
Nicht verfügbar.

FBD-Sprache

Nicht verfügbar. Verwenden Sie RS- oder SR-Funktionsblöcke.

KOP-Sprache

Verwendung der S-Spule:



AWL-Sprache

```
Op1:    LD    SET
        S     Q     (* Q wird auf TRUE gesetzt, wenn SET TRUE ist *)
                    (* Q bleibt unverändert, wenn SET FALSE ist *)
```

Siehe auch

R RS SR

ScaleLin

Funktion – Skalierung – lineare Konvertierung.

Eingänge

IN : REAL Reeller Zahlenwert.
IMIN : REAL Minimaler Eingangswert.
IMAX : REAL Maximaler Eingangswert.
OMIN : REAL Minimaler Ausgangswert.
OMAX : REAL Maximaler Ausgangswert.

Ausgänge

OUT : REAL Ergebnis: OMIN + IN * (OMAX - OMIN) / (IMAX - IMIN).

Wahrheitstabelle

EINGÄNGE	OUT
IMIN >= IMAX	= IN
IN < IMIN	= OMIN
IN > IMAX	= OMAX
Andere	= OMIN + IN * (OMAX - OMIN) / (IMAX - IMIN)

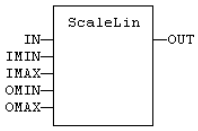
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

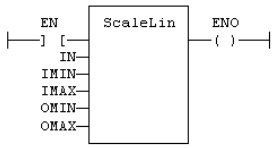
```
OUT := ScaleLin (IN, IMIN, IMAX, OMIN, OMAX);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD          IN
        ScaleLin    IMAX, IMIN, OMAX, OMIN
        ST          OUT
```

SEL

Funktion – Wählt einen der Eingänge aus – 2 Eingänge.

Eingänge

SELECT : BOOL Auswahlbefehl.
IN0 : ANY Erster Eingang.
IN1 : ANY Zweiter Eingang.

Ausgänge

Q : ANY IN1 wenn SELECT *FALSE* ist; IN2 wenn SELECT *TRUE* ist

Wahrheitstabelle

SELECT	Q
0	IN0
1	IN1

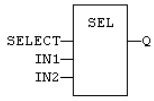
Anmerkungen

In der KOP-Sprache ist der Selektor-Befehl die Eingangsstufe. Die Ausgangsstufe behält den gleichen Status bei wie die Eingangsstufe. In AWL-Sprache muss der erste Parameter (Selektor) in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann. Andere Eingänge sind Operanden der Funktion, die durch Kommas getrennt sind.

ST-Sprache

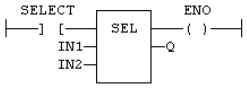
```
Q := SEL (SELECT, IN0, IN1);
```

FBD-Sprache



KOP-Sprache

Der Eingangsstrompfad ist der Selektor.
ENO hat den gleichen Wert wie SELECT.



AWL-Sprache

```
Op1:    LD    SELECT
        SEL   IN1, IN2
        ST    Q
```

Siehe auch

MUX4 MUX8

SEMA

Funktionsblock – Semaphor.

Eingänge

CLAIM : BOOL Übernimmt den Semaphor.
RELEASE : BOOL Gibt den Semaphor frei.

Ausgänge

BUSY : BOOL *TRUE*, wenn der Semaphor ausgelastet ist.

Anmerkungen

Der Funktionsblock implementiert den folgenden Algorithmus:

```
BUSY := mem;
if CLAIM then
      mem := TRUE;
else if RELEASE then
      BUSY := FALSE;
      mem := FALSE;
end_if;
```

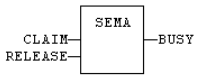
In der KOP-Sprache ist die Eingangsstufe der CLAIM-Befehl. Die Ausgangsstufe ist das BUSY-Ausgangssignal.

ST-Sprache

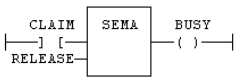
MySema ist eine deklarierte Instanz des Funktionsblocks SEMA:

```
MySema (CLAIM, RELEASE);
BUSY := MyBlinker.BUSY;
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

MySema ist eine deklarierte Instanz des Funktionsblocks SEMA:

```
Op1:    CAL   MySema (CLAIM, RELEASE)
        LD    MyBlinker.BUSY
        ST    BUSY
```

SETBIT

Funktion – Legt ein Bit in einem Ganzzahl-Register fest.

Eingänge

IN : ANY 8- bis 32-Bit-Ganzzahl-Register.
BIT : DINT Bitnummer (0 = niederwertiges Bit).
VAL : BOOL Anzuwendender Bitwert.

Ausgänge

Q : ANY Geändertes Register.

Anmerkungen

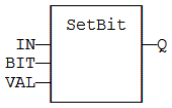
Die Typen LINT, REAL, LREAL, TIME und STRING werden für IN und Q nicht unterstützt. IN und Q und müssen denselben Typ haben. Bei ungültigen Argumenten (ungültige Bitnummer oder ungültiger Eingangstyp) gibt die Funktion den Wert IN ohne Änderung zurück.

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) TRUE ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe.

ST-Sprache

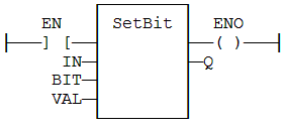
Q := SETBIT (IN, BIT, VAL);

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

Nicht verfügbar.

Siehe auch

TESTBIT

SetWithin

Funktion – Erzwingt einen Wert innerhalb eines Intervalls

Eingänge

IN : ANY Eingang
MIN : ANY Untergrenze des Intervalls
MAX : ANY Obergrenze des Intervalls
VAL : ANY Wert, der innerhalb des Intervalls angewendet werden soll

Ausgänge

Q : BOOL Ergebnis

Wahrheitstabelle

IN	Q
IN < MIN	IN
IN > MAX	IN
MIN < IN < MAX	VAL

Anmerkungen

Der Ausgang wird auf VAL gesetzt, wenn der IN-Wert innerhalb des Intervalls [MIN .. MAX] liegt. Liegt er außerhalb des Intervalls, wird er auf IN gesetzt.

SHL

Funktion – Verschiebt Bits in einem Register nach links.

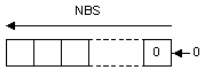
Eingänge

IN : ANY Register.
NBS : ANY Anzahl der Verschiebungen (jede Verschiebung ist 1 Bit).

Ausgänge

Q : ANY Verschobenes Register.

Diagramm



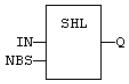
Anmerkungen

Argumente können aus Ganzzahlen mit oder ohne Vorzeichen von 8 bis 32 Bits bestehen.
In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den Status des Eingangsstrompfads bei. In der AWL-Sprache muss der erste Eingang vor dem Funktionsaufruf geladen werden. Der zweite Eingang ist der Operand der Funktion.

ST-Sprache

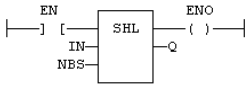
Q := SHL (IN, NBS);

FBD-Sprache



KOP-Sprache

Die Verschiebung wird nur ausgeführt, wenn EN *TRUE* ist.
ENO hat den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD    IN
        SHL   NBS
        ST    Q
```

Siehe auch

SHR ROL ROR SHLb SHRb ROLb RORb SHLw SHRw ROLw RORw

SHR

Funktion – Verschiebt Bits in einem Register nach rechts.

Eingänge

IN : ANY Register.
NBS : ANY Anzahl der Verschiebungen (jede Verschiebung ist 1 Bit).

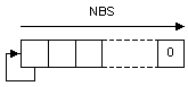
Ausgänge

Q : ANY Verschobenes Register.

WICHTIG!

Wenn im Feld „Project settings / Advanced" (Projekteinstellungen / Erweitert) die Option „SHR: do not duplicate the most significant bit" (SHR: Das höchstwertige Bit nicht duplizieren) aktiviert ist, wird das höchstwertige Bit auf *FALSE* gesetzt.

Wenn die Option nicht aktiviert ist, wird das höchstwertige Bit dupliziert:



Anmerkungen

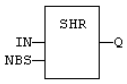
Argumente können aus Ganzzahlen mit oder ohne Vorzeichen von 8 bis 32 Bits bestehen.

In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den Status des Eingangsstrompfads bei. In der AWL-Sprache muss der erste Eingang vor dem Funktionsaufruf geladen werden. Der zweite Eingang ist der Operand der Funktion.

ST-Sprache

Q := SHR (IN, NBS);

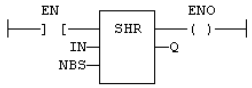
FBD-Sprache



KOP-Sprache

Die Verschiebung wird nur ausgeführt, wenn EN *TRUE* ist.

ENO hat den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD    IN
        SHR   NBS
        ST    Q
```

Siehe auch

SHL ROL ROR SHLb SHRb ROLb RORb SHLw SHRw ROLw RORw

SIN SINL

Funktion – Berechnet einen Sinus.

Eingänge

IN : REAL/LREAL Reeller Zahlenwert.

Ausgänge

Q : REAL/LREAL Ergebnis: Sinus von IN.

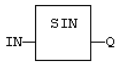
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

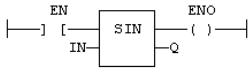
```
Q := SIN (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD    IN
        SIN
        ST    Q (* Q ist: SIN (IN) *)
```

Siehe auch

COS TAN ASIN ACOS ATAN ATAN2

SQRT SQRTL

Funktion – Berechnet die Quadratwurzel des Eingangs.

Eingänge

IN : REAL/LREAL Reeller Zahlenwert.

Ausgänge

Q : REAL/LREAL Ergebnis: Quadratwurzel von IN.

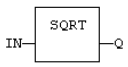
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

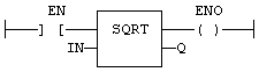
```
Q := SQRT (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD    IN
        SQRT
        ST    Q (* Q ist: SQRT (IN) *)
```

Siehe auch

ABS TRUNC LOG POW

SR

Funktionsblock – Legt dominanten bistabilen Wert fest.

Eingänge

SET1 : BOOL Bedingung für Erzwingen von *TRUE* (Befehl mit höchster Priorität).
RESET : BOOL Bedingung für Erzwingen von *FALSE*.

Ausgänge

Q1 : BOOL Zu erzwingender Ausgang.

Wahrheitstabelle

SET1	RESET	Q1 VORH.	Q1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1

Anmerkungen

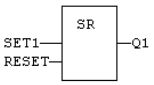
Der Ausgang bleibt unverändert, wenn beide Eingänge *FALSE* sind. Wenn beide Eingänge *TRUE* sind, wird der Ausgang auf *TRUE* gesetzt (dominanten Wert festlegen).

ST-Sprache

MySR wird als Instanz des Funktionsblocks SR deklariert:

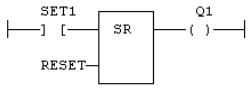
```
MySR (SET1, RESET);
Q1 := MySR.Q1;
```

FBD-Sprache



KOP-Sprache

Der Befehl SET1 ist die Stufe – die Stufe ist der Ausgang:



AWL-Sprache

MySR wird als Instanz des Funktionsblocks SR deklariert:

```
Op1:    CAL   MySR (SET1, RESET)
        LD    MySR.Q1
        ST    Q1
```

Siehe auch

R S RS

StringTable

Funktion – Wählt die aktive Zeichenfolgentabelle aus.

Eingänge

TABLE : STRING Name der Zeichenfolgentabellen-Ressource – muss eine Konstante sein.
COL : STRING Name der Spalte in der Tabelle – muss eine Konstante sein.

Ausgänge

OK : BOOL *TRUE*, wenn OK.

Anmerkungen

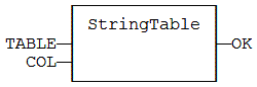
Diese Funktion wählt eine Spalte einer gültigen Zeichenfolgentabellen-Ressource aus, die zur aktiven Zeichenfolgentabelle wird. Die Funktion LoadString() bezieht sich immer auf die aktive Zeichenfolgentabelle. Argumente müssen konstante Zeichenfolge -Ausdrücke sein und in eine deklarierte Zeichenfolgentabelle und einen gültigen Spaltennamen in dieser Tabelle passen.

Wenn Sie nur eine Zeichenfolgentabelle haben, in der nur eine Spalte in Ihrem Projekt definiert ist, müssen Sie diese Funktion nicht aufrufen, da sie ohnehin die Standard-Zeichenfolgentabelle ist.

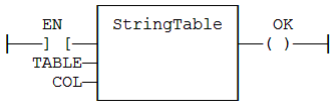
ST-Sprache

```
OK := StringTable ('MyTable', 'FirstColumn");
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

```
Op1:    LD              'MyTable'
        StringTable'    First Column'
        ST              OK
```

Siehe auch

LoadString Zeichenfolgentabellen

StringToArray StringToArrayU

Funktion – Kopiert die Zeichen aus einem STRING in ein SINT-Array.

Eingänge

SRC : STRING Quell-STRING.
DST : SINT Ziel-Array von SINT kurzen Ganzzahlen (USINT für ArrayToStringU).

Ausgänge

Q : DINT Anzahl der kopierten Zeichen.

Anmerkungen

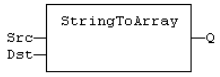
In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

Diese Funktion kopiert die Zeichen der SRC-Zeichenfolge in die ersten Zeichen des DST-Arrays. Die Funktion prüft die maximale Größe von Ziel-Arrays und reduziert bei Bedarf die Anzahl der kopierten Zeichen.

ST-Sprache

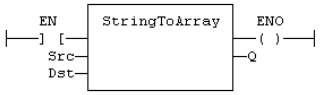
```
Q := StringToArray (SRC, DST);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

Op1:	LD	SRC
	StringToArray	DST
	ST	Q

Siehe auch

ArrayToString

-SUB

Operator – Führt eine Subtraktion der Eingänge durch.

Eingänge

IN1 : ANY_NUM / TIME Erster Eingang.
IN2 : ANY_NUM / TIME Zweiter Eingang.

Ausgänge

Q : ANY_NUM / TIME Ergebnis: IN1 - IN2.

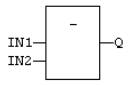
Anmerkungen

Alle Eingänge und der Ausgang müssen vom gleichen Typ sein. In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den gleichen Wert bei wie die Eingangsstufe. In der AWL-Sprache führt der SUB-Befehl eine Subtraktion zwischen dem aktuellen Ergebnis und dem Operanden durch. Das aktuelle Ergebnis und der Operand müssen vom gleichen Typ sein.

ST-Sprache

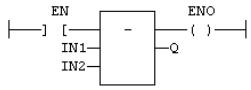
```
Q := IN1 - IN2;
```

FBD-Sprache



KOP-Sprache

Die Subtraktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO ist gleich EN.



AWL-Sprache

```
Op1:    LD    IN1
        SUB   IN2
        ST    Q (* Q ist gleich: IN1 - IN2 *)
Op2:    LD    IN1
        SUB   IN2
        SUB   IN3
        ST    Q (* Q ist gleich: IN1 - IN2 - IN3 *)
```

Siehe auch

+ ADD * MUL / DIV

TAN TANL

Funktion – Berechnet einen Tangens.

Eingänge

IN : REAL/LREAL Reeller Zahlenwert.

Ausgänge

Q : REAL/LREAL Ergebnis: Tangens von IN.

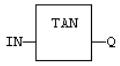
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

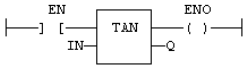
```
Q := TAN (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD    IN
        TAN
        ST    Q (* Q ist: TAN (IN) *)
```

Siehe auch

SIN COS ASIN ACOS ATAN ATAN2

TESTBIT

Funktion – Testet ein Bit in einem Ganzzahl-Register.

Eingänge

IN : ANY 8- bis 32-Bit-Ganzzahl-Register.
BIT : DINT Bitnummer (0 = niederwertiges Bit).

Ausgänge

Q : BOOL Bitwert.

Anmerkungen

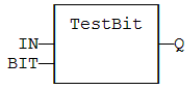
Die Typen LINT, REAL, LREAL, TIME und STRING werden für IN und Q nicht unterstützt. IN und Q und müssen denselben Typ haben. Bei ungültigen Argumenten (ungültige Bitnummer oder ungültiger Eingangstyp) gibt die Funktion *FALSE* zurück.

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE ist*. Die Ausgangsstufe ist die Ausgabe der Funktion.

ST-Sprache

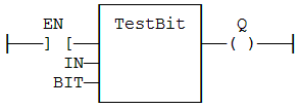
Q := TESTBIT (IN, BIT);

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.



AWL-Sprache

Nicht verfügbar.

Siehe auch

SETBIT

TMD

Funktionsblock – Stoppuhr mit Abwärtszählung.

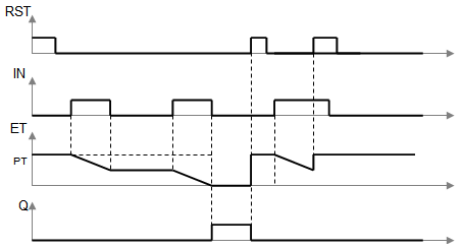
Eingänge

IN : BOOL Die Zeit wird gemessen, wenn dieser Eingang *TRUE* ist.
RST : BOOL Timer wird auf PT zurückgesetzt, wenn dieser Eingang *TRUE* ist.
PT : TIME Programmierte Zeit.

Ausgänge

Q : BOOL Ausgangssignal für abgelaufenen Timer.
ET : TIME Verstrichene Zeit.

Zeitdiagramm



Anmerkungen

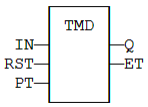
Der Timer zählt hoch, wenn der Eingang IN *TRUE* ist. Er stoppt, wenn die programmierte Zeit abgelaufen ist. Der Timer wird zurückgesetzt, wenn der RST-Eingang *TRUE* ist. Er wird nicht zurückgesetzt, wenn IN *FALSE* ist.

ST-Sprache

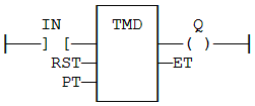
MyTimer ist eine deklarierte Instanz des Funktionsblocks TMD.

```
MyTimer (IN, RST, PT);
Q := MyTimer.Q;
ET := MyTimer.ET;
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

MyTimer ist eine deklarierte Instanz des Funktionsblocks TMD.

```
Op1:    CAL   MyTimer (IN, RST, PT)
        LD    MyTimer.Q
        ST    Q
        LD    MyTimer.ET
        ST    ET
```

Siehe auch

TMU

TMU TMUsec

Funktionsblock – Stoppuhr mit Aufwärtszählung.

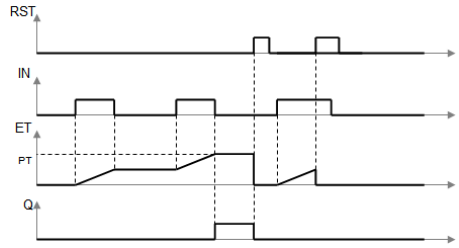
Eingänge

IN : BOOL Die Zeit wird gemessen, wenn dieser Eingang TRUE ist.
RST : BOOL Timer wird auf 0 zurückgesetzt, wenn dieser Eingang TRUE ist.
PT : TIME Programmierte Zeit. (TMU)
PTsec : UDINT Programmierte Zeit. (TMUsec - Sekunden)

Ausgänge

Q : BOOL Ausgangssignal für abgelaufenen Timer.
ET : TIME Verstrichene Zeit. (TMU)
ETsec : UDINT Verstrichene Zeit. (TMU - Sekunden)

Zeitdiagramm



Anmerkungen

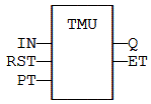
Der Timer zählt hoch, wenn der Eingang IN *TRUE* ist. Er stoppt, wenn die programmierte Zeit abgelaufen ist. Der Timer wird zurückgesetzt, wenn der RST-Eingang *TRUE* ist. Er wird nicht zurückgesetzt, wenn IN *FALSE* ist.

ST-Sprache

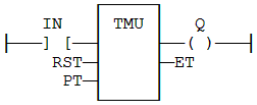
MyTimer ist eine deklarierte Instanz des Funktionsblocks TMU.

```
MyTimer (IN, RST, PT);
Q := MyTimer.Q;
ET := MyTimer.ET;
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

MyTimer ist eine deklarierte Instanz des Funktionsblocks TMU.

```
Op1:    CAL   MyTimer (IN, RST, PT)
        LD    MyTimer.Q
        ST    Q
        LD    MyTimer.ET
        ST    ET
```

Siehe auch

TMD

TOF TOFR

Funktionsblock – Timer aus.

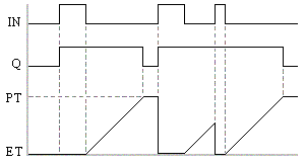
Eingänge

IN : BOOL Timer-Befehl.
PT : TIME Programmierte Zeit.
RST : BOOL Zurücksetzen (nur TOFR).

Ausgänge

Q : BOOL Ausgangssignal für abgelaufenen Timer.
ET : TIME Verstrichene Zeit.

Zeitdiagramm



Anmerkungen

Der Timer startet bei einem abfallenden Impuls des Eingangs IN. Er stoppt, wenn die verstrichene Zeit der programmierten Zeit entspricht. Ein ansteigender Impuls des Eingangs IN setzt den Timer auf 0 zurück. Das Ausgangssignal wird auf *TRUE* gesetzt, wenn der Eingang IN auf *TRUE* ansteigt, und auf *FALSE* zurückgesetzt, nachdem die programmierte Zeit abgelaufen ist.

TOFR ist mit TOF identisch, verfügt jedoch über einen zusätzlichen Eingang zum Zurücksetzen des Timers.

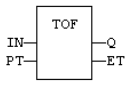
In der KOP-Sprache ist die Eingangsstufe der IN-Befehl. Die Ausgangsstufe ist das Ausgangssignal Q.

ST-Sprache

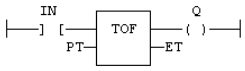
MyTimer ist eine deklarierte Instanz des Funktionsblocks TOF.

```
MyTimer (IN, PT);
Q := MyTimer.Q;
ET := MyTimer.ET;
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

MyTimer ist eine deklarierte Instanz des Funktionsblocks TOF.

```
Op1:    CAL   MyTimer (IN, PT)
        LD    MyTimer.Q
        ST    Q
        LD    MyTimer.ET
        ST    ET
```

Siehe auch

TON TP BLINK

TON

Funktionsblock – Timer ein.

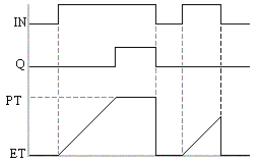
Eingänge

IN : BOOL Timer-Befehl.
PT : TIME Programmierte Zeit.

Ausgänge

Q : BOOL Ausgangssignal für abgelaufenen Timer.
ET : TIME Verstrichene Zeit.

Zeitdiagramm



Anmerkungen

Der Timer startet bei einem ansteigenden Impuls des Eingangs IN. Er stoppt, wenn die verstrichene Zeit der programmierten Zeit entspricht. Ein abfallender Impuls des Eingangs IN setzt den Timer auf 0 zurück. Das Ausgangssignal wird bei Ablauf der programmierten Zeit auf *TRUE* gesetzt und bei abfallendem Eingangsbefehl auf *FALSE* zurückgesetzt.

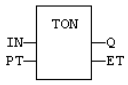
In der KOP-Sprache ist die Eingangsstufe der IN-Befehl. Die Ausgangsstufe ist das Ausgangssignal Q.

ST-Sprache

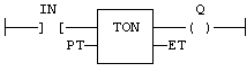
MyTimer ist eine deklarierte Instanz des Funktionsblocks TON.

```
MyTimer (IN, PT);
Q := MyTimer.Q;
ET := MyTimer.ET;
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

MyTimer ist eine deklarierte Instanz des Funktionsblocks TON.

```
Op1:    CAL   MyTimer (IN, PT)
        LD    MyTimer.Q
        ST    Q
        LD    MyTimer.ET
        ST    ET
```

Siehe auch

TOF TP BLINK

TP TPR

Funktionsblock – Impuls-Timer.

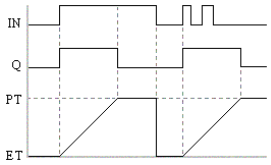
Eingänge

IN : BOOL Timer-Befehl.
PT : TIME Programmierte Zeit.
RST : BOOL Zurücksetzen (nur TPR).

Ausgänge

Q : BOOL Ausgangssignal für abgelaufenen Timer.
ET : TIME Verstrichene Zeit.

Zeitdiagramm



Anmerkungen

Der Timer startet bei einem ansteigenden Impuls des Eingangs IN. Er stoppt, wenn die verstrichene Zeit der programmierten Zeit entspricht. Ein abfallender Impuls des Eingangs IN setzt den Timer nur dann auf 0 zurück, wenn die programmierte Zeit abgelaufen ist. Alle Impulse von IN werden ignoriert, während der Timer läuft. Das Ausgangssignal wird auf *TRUE* gesetzt, während der Timer läuft.

TPR ist mit TP identisch, verfügt jedoch über einen zusätzlichen Eingang zum Zurücksetzen des Timers.

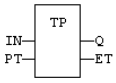
In der KOP-Sprache ist die Eingangsstufe der IN-Befehl. Die Ausgangsstufe ist das Ausgangssignal Q.

ST-Sprache

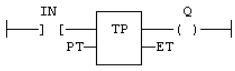
MyTimer ist eine deklarierte Instanz des Funktionsblocks TP.

```
MyTimer (IN, PT);
Q := MyTimer.Q;
ET := MyTimer.ET;
```

FBD-Sprache



KOP-Sprache



AWL-Sprache

MyTimer ist eine deklarierte Instanz des Funktionsblocks TP.

```
Op1:    CAL   MyTimer (IN, PT)
        LD    MyTimer.Q
        ST    Q
        LD    MyTimer.ET
        ST    ET
```

Siehe auch

TON TOF BLINK

TRUNC TRUNCL

Funktion – Kürzt den Dezimalteil des Eingangs.

Eingänge

IN : REAL/LREAL Reeller Zahlenwert.

Ausgänge

Q : REAL/LREAL Ergebnis: Ganzzahliger Teil von IN.

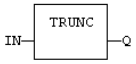
Anmerkungen

In der KOP-Sprache wird der Vorgang nur ausgeführt, wenn die Eingangsstufe (EN) TRUE ist. Die Ausgangsstufe (ENO) behält den gleichen Wert bei wie die Eingangsstufe. In AWL muss der Eingang in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann.

ST-Sprache

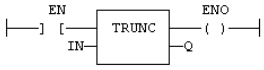
```
Q := TRUNC (IN);
```

FBD-Sprache



KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO behält den gleichen Wert wie EN.



AWL-Sprache

```
Op1:    LD      IN
        TRUNC
        ST      Q (* Q ist der ganzzahlige Teil von IN *)
```

Siehe auch

ABS LOG POW SQRT

UNPACK8

Funktionsbaustein – Extrahiert Bits eines Bytes.

Eingänge

IN : USINT 8-Bit-Register.

Ausgänge

Q0 : BOOL Niederwertiges Bit.
...
Q7 : BOOL Höchstwertiges Bit.

Anmerkungen

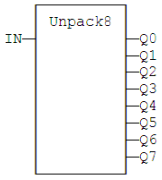
In der KOP-Sprache ist die Ausgangsstufe der Q0-Ausgang. Der Vorgang wird nur ausgeführt, wenn die Eingangsstufe (EN) *TRUE* ist.

ST-Sprache

MyUnpack ist eine deklarierte Instanz des Funktionsblocks UNPACK8.

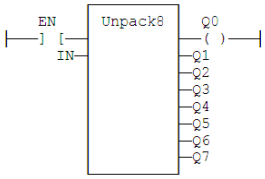
```
MyUnpack (IN);
Q0 := MyUnpack.Q0;
Q1 := MyUnpack.Q1;
Q2 := MyUnpack.Q2;
Q3 := MyUnpack.Q3;
Q4 := MyUnpack.Q4;
Q5 := MyUnpack.Q5;
Q6 := MyUnpack.Q6;
Q7 := MyUnpack.Q7;
```

FBD-Sprache



KOP-Sprache

Der Vorgang wird ausgeführt, wenn EN = *TRUE*:



AWL-Sprache

MyUnpack ist eine deklarierte Instanz des Funktionsblocks UNPACK8.

```
Op1:    CAL   MyUnpack (IN)
        LD    MyUnpack.Q0
        ST    Q0
        (* ... *)
        LD    MyUnpack.Q7
        ST    Q7
```

Siehe auch

PACK8

UseDegrees

Funktion – Legt die Einheit für Winkel in allen trigonometrischen Funktionen fest.

Eingänge

IN : BOOL Wenn *TRUE*, werden alle trigonometrischen Funktionen auf die Verwendung von Grad gesetzt. Wenn *FALSE*, werden alle trigonometrischen Funktionen auf die Verwendung von Radianten (Standard) gesetzt.

Ausgänge

Q : BOOL TRUE, wenn Funktionen vor dem Aufruf Grad verwenden.

Anmerkungen

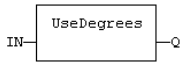
Mit dieser Funktion wird die Arbeitseinheit für die folgenden Funktionen eingestellt:

CODE	FUNKTION
SIN	Sinus
COS	Cosinus
TAN	Tangens
ASIN	Arcussinus
ACOS	Arcuscosinus
ATAN	Arcustangens
ATAN2	Arcustangens von Y / X

ST-Sprache

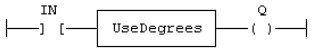
```
Q := UseDegrees (IN);
```

FBD-Sprache



KOP-Sprache

Der Eingang ist der Strompfad. Die Stufe ist der Ausgang.



AWL-Sprache

```
Op1:    LD    IN
        UseDegrees
        ST    Q
```

XOR XORN

Operator – Führt ein exklusives OR aller Eingänge aus.

Eingänge

IN1 : BOOL Erster boolescher Eingang.
IN2 : BOOL Zweiter boolescher Eingang.

Ausgänge

Q : BOOL Exklusives OR aller Eingänge.

Wahrheitstabelle

IN1	IN2	Q
0	0	0
0	1	1
1	0	1
1	1	0

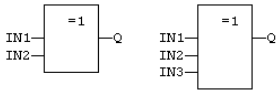
Anmerkungen

Der Block wird in den Sprachen FBD und KOP „=1” genannt. In der AWL-Sprache führt der XOR-Befehl ein exklusives OR zwischen dem aktuellen Ergebnis und dem Operanden aus. Das aktuelle Ergebnis muss ein boolescher Wert sein. Der Befehl XORN führt eine exklusives OR zwischen dem aktuellen Ergebnis und der booleschen Negation des Operanden aus.

ST-Sprache

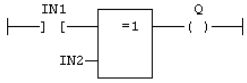
```
Q := IN1 XOR IN2;
Q := IN1 XOR IN2 XOR IN3;
```

FBD-Sprache



KOP-Sprache

Der erste Eingang ist der Strompfad. Die Stufe ist der Ausgang:



AWL-Sprache

```
Op1:    LD      IN1
        XOR     IN2
        ST      Q (* Q ist gleich: IN1 XOR IN2 *)
Op2:    LD      IN1
        XORN    IN2
        ST      Q (* Q ist gleich: IN1 XOR (NOT IN2) *)
```

Siehe auch

AND OR NOT

XOR_MASK

Funktion

Führt ein exklusives Bit-zu-Bit-OR zwischen zwei Ganzzahlwerten aus

Eingänge

IN : ANY Erster Eingang.
MSK : ANY Zweiter Eingang (XOR-Maske).

Ausgänge

Q : ANY Exklusive OR-Maske zwischen IN- und MSK-Eingängen.

Anmerkungen

Argumente können aus Ganzzahlen mit oder ohne Vorzeichen von 8 bis 32 Bits bestehen.

In der KOP-Sprache aktiviert die Eingangsstufe (EN) den Vorgang, und die Ausgangsstufe behält den gleichen Wert bei wie die Eingangsstufe. In AWL-Sprache muss der erste Parameter (IN) in das aktuelle Ergebnis geladen werden, bevor die Funktion aufgerufen werden kann. Der andere Eingang sind die Operanden der Funktion.

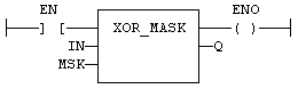
ST-Sprache

```
Q := XOR_MASK (IN, MSK);
```

FBD-Sprache

KOP-Sprache

Die Funktion wird nur ausgeführt, wenn EN *TRUE* ist.
ENO ist gleich EN.



AWL-Sprache

```
Op1:    LD          IN
        XOR_MASK    MSK
        ST          Q
```

Siehe auch

AND_MASK OR_MASK NOT_MASK