OSCAT

BASIC:LIBRARY

Dokumentation

Version 3.33

Inhaltsverzeichnis

1. Rechtsgrundlagen

1.1. Haftungsausschluss

Die in der OSCAT Bibliothek enthaltenen Softwaremodule sind in der Absicht angeboten als Entwicklungsvorlage und Leitfaden zur Softwareentwicklung für SPS nach IEC61131-3 zu dienen. Eine Funktionsgarantie wird von den Entwicklern nicht übernommen und wird explizit ausgeschlossen. Da die in der Bibliothek enthaltenen Softwaremodule ohne jegliche Kosten bereitgestellt werden, besteht keinerlei Gewährleistung, soweit dies gesetzlich zulässig ist. Sofern nicht explizit schriftlich vereinbart, stellen die Copyright-Inhaber und / oder Dritte die Softwaremodule so zur Verfügung, „wie es ist“, ohne irgendeine Gewährleistung, weder ausdrücklich noch implizit, einschließlich - aber nicht begrenzt - auf Marktreife oder Verwendbarkeit für einen bestimmten Zweck. Das volle Risiko und die volle Verantwortung bezüglich Qualität, Fehlerfreiheit und Leistungsfähigkeit der Softwaremodule liegt beim Anwender selbst. Sollte sich die Bibliothek, oder Teile der Bibliothek als Fehlerhaft erweisen, liegen die Kosten für notwendigen Service, Reparatur und / oder Korrektur beim Anwender selbst. Sollten Teile oder die gesamte Bibliothek zur Erstellung von Anwendungssoftware verwendet werden, oder in Softwareprojekten eingesetzt werden, so haftet der Anwender für die Fehlerfreiheit, Funktion und Qualität der Anwendung. Eine Haftung durch OSCAT ist grundsätzlich ausgeschlossen. Der Anwender der OSCAT Bibliothek hat durch geeignete Tests und Freigaben und Qualitätssicherungsmaßnahmen dafür zu sorgen, dass durch eventuelle Fehler in der Bibliothek von OSCAT keine Schäden entstehen können. Die vorstehenden Lizenzbedingungen und Haftungsausschlüsse gelten gleichermaßen für die Softwarebibliothek, sowie die in diesem Handbuch angebotenen Beschreibungen und Erläuterungen, auch wenn dies nicht explizit erwähnt wird.

1.2. Lizenzbedingungen

Die Verwendung der OSCAT Bibliothek ist kostenfrei und kann ohne Lizenzvereinbarung für private oder gewerbliche Zwecke eingesetzt werden. Eine Verbreitung der Bibliothek ist ausdrücklich erwünscht, hat aber kostenfrei und unter Hinweis auf unsere Webpage WWW.OSCAT.DE zu erfolgen. Wird die Bibliothek in elektronischer Form zum Download bereitgestellt, oder auf Datenträgern verbreitet, so ist sicherzustellen, dass ein deutlich erkennbarer Hinweis auf OSCAT und ein Weblink zu WWW.OSCAT.DE entsprechend enthalten sind.

1.3. Eingetragene Markenzeichen

Alle in dieser Beschreibung benutzen Markennamen werden ohne Verweis auf deren Eintragung beziehungsweise Besitzer verwendet. Die Existenz solcher Rechte kann daher nicht ausgeschlossen werden. Die benutzten Markennamen sind Eigentum des jeweiligen Besitzers. Eine Verwendung der Beschreibung für kommerzielle Zwecke ist daher nicht gestattet, auch nicht auszugsweise.

1.4. Bestimmungsgemäßer Gebrauch

Die in der OSCAT Bibliothek enthaltenen und in dieser Dokumentation beschriebenen Softwaremodule sind ausschließlich für Fachpersonal mit einer Ausbildung in SPS Programmierung. Die Anwender sind verantwortlich für die Einhaltung aller geltenden Normen und Vorschriften die bei der Anwendung zum tragen kommen. OSCAT weist weder im Handbuch noch in der Software auf diese Normen und Vorschriften hin.

1.5. Sonstiges

Alle Rechtsverbindlichen Regelungen befinden sich ausschließlich im Kapitel 1 dieses Handbuchs. Eine Ableitung oder Bezug von rechtlichen Ansprüchen aufgrund des Inhalts dieses Manuals außer den Bestimmungen in Kapitel 1 ist gänzlich ausgeschlossen.

2. Einleitung

2.1. Ziele

OSCAT steht für "Open Source Community for Automation Technology".

OSCAT erstellt eine Open Source Bibliothek nach dem IEC61131-3 Standard, welche auf herstellerspezifische Funktionen verzichtet und deshalb auf alle IEC61131-3 kompatiblen Speicherprogrammierbaren Steuerungen portiert werden kann. Auch wenn Entwicklungen für SPS unter dem Einsatz von herstellerspezifischen Bibliotheken meist effizient zu lösen sind und diese Bibliotheken auch teilweise kostenfrei zur Verfügung gestellt werden, ergeben sich doch große Nachteile durch ihren Einsatz:

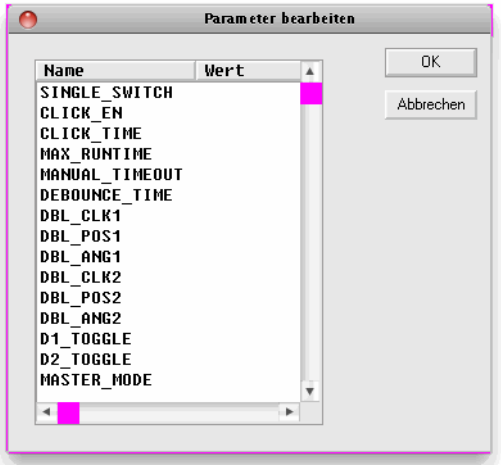
1. Die Bibliotheken der Hersteller sind praktisch alle geschützt und der Source Code ist nicht frei zugänglich, was im Fehlerfall eine Fehlersuche und auch die Behebung des Fehlers enorm schwierig, oft sogar unmöglich macht.
2. Die grafische Erstellung von Programmen kann mit herstellerspezifischen Bibliotheken schnell unübersichtlich, ineffizient und fehleranfällig werden, weil vorhandene Funktionen wegen des fehlenden Source Codes den eigentlichen Bedürfnissen nicht angepasst und erweitert werden können.
3. Ein Wechsel der Hardware, insbesondere der Wechsel zu einem anderen Hersteller, ist durch die geschützten Bibliotheken verhindert und die Vorteile, die ein Standard wie IEC61131 bieten würde, werden so eingeschränkt. An einen Austausch einer herstellerspezifischen Bibliothek mit der eines Wettbewerbers ist ausgeschlossen, denn die Bibliotheken der Hersteller unterscheiden sich enorm in Umfang und Inhalt.
4. Das Verständnis komplexer Module ist ohne einen Einblick in den Sourcecode oft sehr schwierig. Dadurch werden Programme ineffizient und fehleranfällig.

OSCAT will mit der quell offenen OSCAT Bibliothek einen mächtigen und umfassenden Standard für die Programmierung von SPS schaffen, der im Source Code zur Verfügung steht und durch vielfältige Anwendungen ausführlich verifiziert und getestet wurde. Weiterhin fließen durch die Vielzahl der Anwendungen umfangreiche Erkenntnisse und Anregungen in die Bibliothek ein. Dadurch kann die Bibliothek als sehr praxisnah bezeichnet werden. OSCAT versteht seine Bibliothek als Entwicklungsvorlage und nicht als ausgereiftes Produkt. Der Nutzer ist selbst verantwortlich dafür, eventuell in seiner Anwendung verwendete Module mit geeigneten Verfahren zu testen und die notwendige Fehlerfreiheit, Qualität und Funktionalität zu verifizieren. An dieser Stelle sei noch einmal auf die Lizenz-

bedingungen und den Haftungsausschluss in dieser Dokumentation hingewiesen.

2.2. Konventionen

5. Direkte Manipulationen im Speicher:
   Funktionen, die Eingangswerte über Pointer verändern, wie zum Beispiel _Array_Sort, beginnen alle mit einem Unterstrich „_“. _Array_Sort sortiert ein Array direkt im Speicher, was den gravierenden Vorteil hat, dass ein eventuell sehr großes Array erst gar nicht der Funktion übergeben werden muss und deshalb Speicher in der Größe des Arrays und die Zeit zum Kopieren eingespart wird. Es wird allerdings nur erfahrenen Anwendern empfohlen diese Funktionen einzusetzen, da eine Fehlanwendung zu gravierenden Fehlern und Abstürzen führen kann! Bei Anwendung von Funktionen die mit einem „_“ beginnen ist besondere Sorgfalt angebracht und insbesondere darauf zu achten, dass die Aufrufparameter niemals undefinierte Werte annehmen können.
6. Namensgebung bei Funktionen:
   Funktionsbausteine mit Zeitverhalten, wie etwa die Funktion PT1 werden durch die Namensgebung FT_Bausteinname (FT_PT1) beschrieben. Funktionen ohne Zeitbezug sind mit F_Bausteinname angegeben.
7. Logische Gleichungen:
   Innerhalb dieses Handbuchs werden die logischen Verknüpfungen & für UND bzw. AND, + für ODER bzw. OR, /A für ein negiertes A und # für XOR (exklusives ODER) verwendet.
8. Setup-Werte für Bausteine:
   Damit die Anwendung und Programmierung übersichtlich bleibt und komplexe Funktionen einfacher dargestellt werden können, haben viele der Bausteine der OSCAT Bibliothek einstellbare Parameter, die bei Anwendung durch einen Doppelklick auf das grafische Symbol des Bausteins bearbeitet werden können. Ein Doppelklick auf das Symbol öffnet eine Dialogbox, die das Bearbeiten der Setup-Werte erlaubt. Wird eine Funktion mehrfach verwendet, so können damit je Baustein die Setup-Werte einzeln festgelegt werden. Die Bearbeitung durch Doppelklick funktioniert unter CoDeSys ausschließlich in CFC. In ST müssen alle Parameter,



auch die Setup-Parameter, im Aufruf übergeben werden. Die Setup-Parameter werden einfach nach den normalen Eingängen angefügt. Die Parameter werden in der grafischen Oberfläche durch Doppelklick bearbeitet und dann wie Konstanten unter IEC61131 eingegeben. Es ist Dabei zu beachten dass Zeiten mit T#200ms und TRUE und FALSE in Großbuchstaben geschrieben sein müssen.

9. Error- und Status-Reporting (ESR):

   Komplexere Bausteine erhalten zum großen Teil einen Error- oder Status-Ausgang. Ein Error-Ausgang ist 0, falls kein Fehler bei der Ausführung auftritt. Tritt jedoch im Baustein ein Fehler auf, so nimmt dieser Ausgang einen Wert im Bereich 1 .. 99 an und meldet somit einen Fehler mit Fehlernummer. Ein Status- oder Error-Sammelmodul kann diese Meldungen sammeln und mit einem Zeitstempel versehen, in einer Datenbank bzw. Array speichern, oder per TCP/IP an übergeordnete Systeme weiterleiten. Ein Ausgang des Typs Status ist kompatibel zu einem Error-Ausgang mit identischer Funktion. Jedoch meldet ein Status-Ausgang nicht nur Fehler, sondern führt auch über Aktivitäten des Bausteins Protokoll. Werte zwischen 1 .. 99 sind weiterhin Fehlermeldungen. Zwischen 100 .. 199 befinden sind Meldungen über Zustandsveränderungen. Der Bereich 200 .. 255 ist reserviert für Debug-Meldungen. Mit dieser, innerhalb der OSCAT Bibliothek standardisierten Funktionalität, wird eine einfache und übergreifende Möglichkeit geboten, Betriebszustandsmeldungen und Fehlermeldungen auf einfache Weise zu integrieren, ohne die Funktion eines Systems zu beeinflussen. Bausteine, die dieses Verfahren unterstützen, werden ab der Revision 1.4 mit der Kennzeichnung „ESR-Fähig“ gekennzeichnet. Weitere Informationen zu den ESR-Bausteinen finden Sie im Abschnitt „Diverse Funktionen“.

2.3. Testumgebung

Die OSCAT Bibliothek wird unter CoDeSys entwickelt und auf verschiedenen Systemen getestet.

Die Testumgebung umfasst folgende Systeme:

1. Beckhoff BX 9000
   mit TwinCAT PLC Control Version 2.10.0
2. Beckhoff CX 9001-1001
   mit TwinCAT PLC Control Version 2.10.0
3. Wago 750-841
   mit CoDeSys Version 2.3.9.31
4. Möller EC4P222
   mit CoDeSys Version 2.3.9.31

5. CoDeSys Simulation auf I386 mit CoDeSys 2.3.9.31
6. CoDeSys Simulation auf i386 mit Codesys 3.4
7. S7 und STEP7: Die OSCAT Bibliothek wird seit Version 1.5 auf STEP7 übersetzt und auch verifiziert.
8. PCWORX / MULTIPROG: Die OSCAT Bibliothek wird seit Version 2.6 auf MULTIPROG übersetzt und verifiziert.
9. Bosch Rexroth IndraLogic XLC L25/L45/L65 mit Indraworks 12VRS
10. Bosch Rexroth IndraMotion MLC L25/L45/L65 mit Indraworks 12VRS
11. Bosch Rexroth IndraMotion MTX L45/L65/L85 mit Indraworks 12VRS

Wir sind stetig darum bemüht die OSCAT Bibliothek auch auf weiteren Testumgebungen zu testen.

2.4. Globale Konstanten

Die OSCAT Bibliothek versucht globale Variablen zu vermeiden, um möglichst einfach in fremde Umgebungen integriert werden zu können. Globale Variablen sind nicht für Funktion oder den Datenaustausch zwischen Bausteinen nötig. Das Setup und die Konfiguration ist vollständig innerhalb der Module realisiert, um höchste Modularität und Portabilität zu gewährleisten. Für physikalische und mathematische Konstanten haben wir uns aus Gründen der Übersichtlichkeit jedoch dazu entschlossen, Globale Konstanten zu verwenden.

MATH :

MATH definiert mathematische Konstanten. Die Konstanten sind in der TYP Definition CONSTANTS_MATH definiert.

PHYS :

PHYS definiert physikalische Konstanten. Die Konstanten sind in der TYP Definition CONSTANTS_PHYS definiert.

LANGUAGE :

LANGUAGE definiert Spracheinstellungen. Die Einstellungen sind in der TYP Definition CONSTANTS_LANGUAGE definiert.

SETUP :

SETUP definiert allgemeine Grundeinstellungen. Die Einstellungen sind in der TYP Definition CONSTANTS_SETUP definiert.

LOCATION :

LOCATION definiert Ortseinstellungen, hierzu zählen unter anderem auch Feiertagsdefinitionen. Die Einstellungen sind in der TYP Definition CONSTANTS_LOCATION definiert.

STRING_LENGTH : INT := 250.

STRING_LENGTH ist per Default 250 Zeichen, und wird von STRING Funktionen benutzt um eine Bereichsüberschreitung bei Verarbeitung von STRINGS zu vermeiden. STRING_LENGTH legt auch innerhalb der OSCAT LIB die maximale STRING Länge fest welche bei häufiger Nutzung des Datentyps STRING zu hohem Speicherverbrauch führen kann. Wenn in einer Anwendung nur kurze STRINGS verarbeitet werden müssen so kann über diese SETUP Konstante die STRING Länge entsprechend reduziert werden. Wir empfehlen STRINGS innerhalb der Anwendung unter Zuhilfenahme dieser Konstante zu definieren. Werden längere STRINGS als 80 Zeichen zur Verarbeitung nötig kann diese Konstante auf Werte bis 255 erhöht werden.

NEUER_STRING : STRING(STRING_LENGTH).

Durch diese Definition ist auch der neu definierte STRING automatisch auf die Länge von STRING_LENGTH definiert und kann wenn nötig global geändert werden.

LIST_LENGTH : INT := 250.

LIST_LENGTH ist per Default 250 Zeichen und legt die Größe von Listen fest.

2.5. Aktualität

OSCAT aktualisiert diese Beschreibung fortlaufend. Es wird empfohlen sich die jeweils aktuellste Version von der OSCAT Homepage unter www.OSCAT.DE zu laden. Hier wird das jeweils aktuellste Manual zum Download bereitgestellt. Neben dem Manual stellt OSCAT auch eine detaillierte Revision Historie bereit. Die "OSCAT Revision History" listet alle Revisionen der einzelnen Bausteine mit Änderungen und ab welcher Release der Bibliothek dieser Baustein enthalten ist.

2.6. Support

Support wird durch die Vielzahl der Anwender selbst im Forum unter WWW.OSCAT.DE zur Verfügung gestellt. Ein Anspruch auf Support besteht aber generell nicht, auch dann nicht, wenn sich herausstellen sollte, dass die Bibliothek oder Teile der Bibliothek fehlerhaft sind. Der im Rahmen des OSCAT Forums bereitgestellte Support wird von Anwendern freiwillig und untereinander bereitgestellt. Updates der Bibliothek und der Dokumentation werden in der Regel einmal im Monat auf der Homepage von OSCAT unter WWW.OSCAT.DE zur Verfügung gestellt. Ein Anspruch auf Wartung, Fehlerbehebung und Softwarepflege jedweder Art besteht generell nicht und ist als freiwillige Leistung von OSCAT anzusehen. Bitte senden Sie bei Support Anfragen keine email an OSCAT. Anfragen können schneller und effektiver bearbeitet werden wenn die Anfragen in unserem Forum gestellt werden.

3. Datentypen der OSCAT Bibliothek

Die OSCAT Bibliothek definiert neben den Standard Datentypen weitere Datentypen. Diese werden innerhalb der Bibliothek verwendet, können aber jederzeit von Anwender für eigene Deklarationen verwendet werden. Ein Löschen oder verändern von Datentypen kann dazu führen das Teile der Bibliothek sich nicht mehr kompilieren lassen.

3.1. CALENDAR

Eine Variable vom Typ CALENDAR kann benutzt werden um Programmweite Kalenderdaten zur Verfügung zu stellen. Im Abschnitt Datums und Zeitfunktionen finden sich diverse Funktionen um den Kalender fortlaufend zu aktualisieren.

*.UTC : DT	Universelle Weltzeit
*.LDT : DT	Lokalzeit
*.LDATE : DATE	Lokales Datum
*.LTOD : TOD	Lokale Tageszeit
*.YEAR : INT	Lokales Jahr
*.MONTH : INT	Lokales Monat
*.DAY : INT	Lokaler Tag
*.WEEKDAY : INT	Lokaler Wochentag
*.OFFSET : INT	Offset der Lokalzeit zur Weltzeit in Minuten
*.DST_EN : BOOL	Sommerzeit Enable
*.DST_ON : BOOL	Sommerzeit ist Aktiv
*.NAME : STRING(5)	Name der Zeitzone
*.LANGUAGE : INT	Sprache (Siehe Language Setup)
*.LONGITUDE : REAL	Längengrad des Ortes
*.LATITUDE : REAL	Breitengrad des Ortes
*.SUN_RISE : TOD	Zeitpunkt des Sonnenaufgangs (LTC)
*.SUN_SET : TOD	Zeitpunkt des Sonnenuntergangs (LTC)
*.SUN_MIDDAY : TOD	Weltzeit wenn Sonne im Süden Steht (LTC)
*.SUN_HEIGTH : REAL	höchster Sonnenstand über Horizont
*.SUN_HOR : REAL	Horizontaler Sonnenstand in Grad von Nord
*.SUN_VER : REAL	Vertikaler Sonnenstand über den Horizont

*.NIGHT : BOOL	TRUE wenn Nacht
*.HOLIDAY : BOOL	TRUE wenn Feiertag
*.HOLY_NAME : STRING(30)	Name des Feiertages
*.WORK_WEEK :_ INT	aktuelle Arbeitswoche

3.2. COMPLEX

Mit der Struktur COMPLEX werden komplexe Zahlen dargestellt-

*.RE (Realteil einer Komplexen Zahl)

*.IM Imaginärteil einer Komplexen Zahl)

3.3. CONSTANTS_LANGUAGE

Diese Struktur definiert verschiedene Sprachen als String Konstanten. Die Variable LANGUAGE der Globalen Variablenliste stellt diese in der Bibliothek zur Verfügung.

*.DEFAULT : INT := 1 definiert die default Sprache

(1=English, 2=Deutsch, 3= Französisch)

Die Default Sprache wird immer dann verwendet wenn die Sprache 0 aufgerufen wird. Ist die Spracheinstellung > 0 so wird die entsprechende Sprache gewählt.

Spracheinstellung: 0 (es wird die unter DEFAULT spezifizierte Sprache

verwendet) (1 = Englisch, 2 = Deutsch 3 = Französisch)

weitere Sprachen werden definiert indem die Struktur CONSTANTS_LANGUAGE erweitert wird.

*.LMAX : INT := 3 gibt an wie viele Sprachen zur Verfügung stehen

*.WEEKDAYS : ARRAY[1..3, 1..7] OF STRING(10) := 'Monday','Tuesday',

'Wednesday','Thursday','Friday','Saturday','Sunday','Montag',

'Dienstag','Mittwoch','Donnerstag','Freitag','Samstag','Sonntag';

*.WEEKDAYS2 : ARRAY[1..3, 1..7] OF STRING(2) :=

'Mo', 'Tu', 'We', 'Th', 'Fr', 'Sa', 'Su',

'Mo', 'Di', 'Mi', 'Do', 'Fr', 'Sa', 'So';

*.MONTHS : ARRAY[1..3, 1..12] OF STRING(10) := 'January', 'February',

'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December',

'Januar', 'Februar', 'März', 'April', 'Mai', 'Juni', 'Juli', 'August',

'September', 'Oktober', 'November', 'Dezember';

*.MONTHS3 : ARRAY[1..3, 1..12] OF STRING(3) :=

'Jan','Feb','Mar','Apr','May','Jun','Jul','Aug','Sep','Oct','Nov','Dec',

'Jan','Feb','Mrz','Apr','Mai','Jun','Jul','Aug','Sep','Okt','Nov','Dez';

*.DIRS : ARRAY[1..3,0..15] OF STRING(3) := 'N','NNE','NE','ENE','E',

'ESE','SE','SSE','S','SSW','SW','WSW','W','WNW','NW','NNW'

'N','NNO','NO','ONO','O','OSO','SO','SSO','S','SSW','SW','WSW',

'W','WNW','NW','NNW';

3.4. CONSTANTS_LOCATION

Diese Struktur definiert Ortsabhängige Konstanten. Die Variable LOCATION der Globalen Variablenliste stellt diese in der Bibliothek zur Verfügung.

*.DEFAULT : INT := 1;

(1=Deutschland, 2=Österreich, 3=Frankreich, 4=Belgien-Deutsch,

5= Italien-Südtirol).

*.LMAX : INT := 5; gibt an wie viele Orte definiert sind.

*.LANGUAGE : ARRAY[1..5] of INT := 2,2,3,2,2;

für jede Location wird die Sprache definiert.

3.5. CONSTANTS_MATH

Diese Struktur definiert mathematische Konstanten. Die Variable MATH der Globalen Variablenliste stellt diese in der Bibliothek zur Verfügung.

*.PI : REAL := 3.14159265358	Kreiszahl PI
*.PI2 : REAL := 6.28318530717	Kreiszahl PI * 2
*.PI4 : REAL := 12.566370614359	Kreiszahl PI * 4
*.PI05 : REAL := 1.5707963267949	Kreiszahl PI / 2
*.PI025 : REAL := 0.785398163397448	Kreiszahl PI / 4
*.PI_INV : REAL := 0.318309886183791	1 / PI

*.E : REAL := 2.718281828459045235	Eulersche Konstante e
*.E_INV := 0.367879441171442	1 / e
*.SQ2 : REAL := 1.4142135623731	Wurzel von 2
*.FACTS : ARRAY[0..12] of DINT	Fakultäten 0 - 12

3.6. CONSTANTS_PHYS

Diese Struktur definiert physikalische Konstanten. Die Struktur PHYS der Globalen Variablenliste stellt diese in der Bibliothek zur Verfügung.

*.C : REAL := 299792458	Lichtgeschwindigkeit in m/s
*.E : REAL := 1.60217653E-19	Elementarladung in Coulomb = A * s
*.G : REAL := 9.80665	Erdbeschleunigung in m / s^2
*.T0 : REAL := -273.15	absoluter Nullpunkt in °C
*.RU : REAL := 8.314472	Universelle Gaskonstante in J / (Mol · K)
*.PN : REAL := 101325	Normaldruck in Pa

3.7. CONSTANTS_SETUP

Diese Struktur definiert Ortsabhängige Konstanten. Die Variable SETUP der Globalen Variablenliste stellt diese in der Bibliothek zur Verfügung.

*.EXTENDED_ASCII : BOOL := TRUE erweitert den ASCII Zeichensatz um Umlaute z.B. ÄÖÜ

*.CHARNAMES[1..4] : STRING(253) speichert Unicode Zeichennamen

*.MTH_OFS : ARRAY[1..12] OF INT := 0, 31, 59, 90, 120, 151, 181, 212, 243, 273, 304, 334;

MTH_OFS wird in verschiedenen Datumsfunktionen verwendet, im Array steht der jeweilige Tagesoffset für die Monate des Jahres. Der 1. Februar eines Jahre ist z.B. der Tag 31 + 1.

*.DECADES : ARRAY[0..8] OF REAL := 1,10,100,1000,10000,100000, 1000000,10000000,100000000;

3.8. ESR_DATA

Die Struktur ESR_DATA wir für Error- und Status- Reporting Bausteine verwendet.

*.TYP : BYTE	Datentyp
*.ADRESS : STRING(10)	Adressbezeichnung
*.DS : DT	Datums- Zeit- Stempel
*.TS : TIME	Zeitstempel in Millisekunden
*.DATA : ARRAY[0..7] OF BYTE	Datenpaket

3.9. FRACTION

Der Datentyp FRACTION kann benutzt werden um einen Bruch zu Repräsentieren.

*.NUMERATOR : INT	Zähler des Bruches (Numerator)
*.DENOMINATOR : INT	Nenner des Bruches (Denumerator)

3.10. HOLIDAY_DATA

Die Struktur HOLIDAY_DATA findet Verwendung zur Definition und Beschreibung von Feiertagen.

*.NAME : STRING(30)	Name des Feiertages
*.DAY : SINT	Tag des Feiertages
*.MONTH : SINT	Monat des Feiertages
*.USE : SINT	aktiviert den Feiertag (0=aus, 1=ein)

Die Struktur wird zum Beispiel von Modulen wie CALENDAR_CALC und HOLIDAY verwendet, indem ein Array vom Datentyp HOLIDAY_DATA zur Definition der jährlichen Feiertage verwendet wird.

Es sind 3 verschiedene Arten von Definitionen vorgesehen:

DAY 1..31, MONTH 1..12, USE 1 (Feiertag an einem festen Datum)

DAY ±X, MONTH 0, USE 1 (Feiertag ist X Tage vor oder nach Ostern)

DAY 1..31, MONTH 1..12, USE -7..-1 (Feiertag an einem Wochentag vor einem Datum, hierbei ist -1 Montag und -7 Sonntag.

Beispiele:

(NAME := 'Neujahr', DAY := 1, MONTH := 1, USE := 1) Feiertag mit einem festen Datum USE=1 bedeutet er ist aktiv.

(NAME := 'Neujahr', DAY := 1, MONTH := 1, USE := 0) Feiertag mit einem festen Datum USE=0 bedeutet er ist nicht aktiv.

(NAME := 'Karfreitag', DAY := -2, MONTH := 0, USE := 1) Feiertag mit einem festen Offset vom Ostersonntag, in diesem Fall ist der Karfreitag 2 Tage vor dem Ostersonntag.

(NAME := 'Buß und Bettag', DAY := 23, MONTH := 11, USE := -3) Feiertag am letzten Mittwoch vor dem 23.11. eines Jahres.

Beispiele für Feiertagsdefinitionen:

```
Array für Bayerische Feiertage
HOLIDAY_DE : ARRAY[0..29] OF HOLIDAY_DATA := (name := 'Neujahr', day := 1, month := 1, use := 1),
                (name := 'Heilig Drei Könige', day := 6, month := 1, use := 1),
                (name := 'Karfreitag', day := -2, month := 0, use := 1),
                (name := 'Ostersonntag', day := 0, month := 0, use := 1),
                (name := 'Ostermontag', day := 1, month := 0, use := 1),
                (name := 'Tag der Arbeit', day := 1, month := 5, use := 1),
                (name := 'Christi Himmelfahrt', day := 39, month := 0, use := 1),
                (name := 'Pfingstsonntag', day := 49, month := 0, use := 1),
                (name := 'Pfingstmontag', day := 50, month := 0, use := 1),
                (name := 'Fronleichnam', day := 60, month := 0, use := 1),
                (name := 'Augsburger Friedensfest', day := 8, month := 8, use := 0),
                (name := 'Maria Himmelfahrt', day := 15, month := 8, use := 1),
                (name := 'Tag der Deutschen Einheit', day := 3, month := 10, use := 1),
                        (name := 'Reformationstag', day := 31, month := 10, use := 0),
                (name := 'Allerheiligen', day := 1, month := 11, use := 1),
                (name := 'Buss und Bettag', day := 23, month := 11, use := 0),
                (name := '1. Weihnachtstag', day := 25, month := 12, use := 1),
                (name := '2. Weihnachtstag', day := 26, month := 12, use := 1);

Array für Österreichische Feiertage
HOLIDAY_AT : ARRAY[0..29] OF HOLIDAY_DATA := (name := 'Neujahr', day := 1, month := 1, use := 1),
                (name := 'Heilig Drei Könige', day := 6, month := 1, use := 1),
                (name := 'Karfreitag', day := -2, month := 0, use := 1),
                (name := 'Ostersonntag', day := 0, month := 0, use := 1),
                (name := 'Ostermontag', day := 1, month := 0, use := 1),
                (name := 'Christi Himmelfahrt', day := 39, month := 0, use := 1),
                (name := 'Pfingstsonntag', day := 49, month := 0, use := 1),
                (name := 'Pfingstmontag', day := 50, month := 0, use := 1),
```

(name := 'Fronleichnam', day := 60, month := 0, use := 1),
(name := '', day := 8, month := 8, use := 0),
(name := 'Maria Himmelfahrt', day := 15, month := 8, use := 1),
(name := '', day := 3, month := 10, use := 0),
(name := '', day := 31, month := 10, use := 0),
(name := 'Allerheiligen', day := 1, month := 11, use := 1),
(name := 'Maria Empfängnis', day := 8, month := 12, use := 1),
(name := '1. Weihnachtstag', day := 25, month := 12, use := 1),
(name := '2. Weihnachtstag', day := 26, month := 12, use := 1);

Französische Feiertage

HOLIDAY_FR : ARRAY[0..29] OF HOLIDAY_DATA := (name := 'Nouvel an', day := 1, month := 1, use := 1),
(name := 'St Valentin', day := 14, month := 2, use := 0),
(name := 'Vendredi Saint (alsace)', day := -2, month := 0, use := 0),
(name := 'Dimanche de pâques', day := 0, month := 0, use := 1),
(name := 'Lundi de pâques', day := 1, month := 0, use := 1),
(name := 'Jeudi de Ascension', day := 39, month := 0, use := 1),
(name := 'dimanche de Pentecôte ', day := 49, month := 0, use := 1),
(name := 'jeudi de la Trinité', day := 60, month := 0, use := 0),
(name := 'Fête du travail', day := 1, month := 5, use := 1),
(name := 'Victoire 1945 ', day := 8, month := 5, use := 1),
(name := 'Prise de La bastille', day := 14, month := 7, use := 1),
(name := '15 Août 1944', day := 15, month := 8, use := 1),
(name := 'Halloween', day := 31, month := 10, use := 0),
(name := 'Armistice 1918', day := 11, month := 11, use := 1),
(name := 'Noël', day := 25, month := 12, use := 1),
(name := 'Saint Étienne (alsace)', day := 26, month := 12, use := 0),
(name := 'Fête de la musique', day := 21, month := 6, use := 0);

Belgien Deutschsprachig

HOLIDAY_BED : ARRAY[0..29] OF HOLIDAY_DATA := (name := 'Neujahr', day := 1, month := 1, use := 1),
(name := 'Ostersonntag', day := 0, month := 0, use := 1),
(name := 'Ostermontag', day := 1, month := 0, use := 1),
(name := 'Tag der Arbeit', day := 1, month := 5, use := 1),
(name := 'Christi Himmelfahrt', day := 39, month := 0, use := 1),
(name := 'Pfingsten', day := 49, month := 0, use := 1),
(name := 'Pfingstmontag', day := 50, month := 0, use := 1),
(name := 'Nationalfeiertag', day := 21, month := 7, use := 1),
(name := 'Mariä Himmelfahrt', day := 15, month := 8, use := 1),
(name := 'Allerheiligen', day := 1, month := 11, use := 1),
(name := 'Feiertag DG', day := 15, month := 11, use := 1),
(name := 'Heiligabend', day := 24, month := 12, use := 1),
(name := '1. Weihnachtstag', day := 25, month := 12, use := 1),

```
        (name := '2. Weihnachtstag', day := 26, month := 12, use := 1),
        (name := 'Silvester', day := 31, month := 12, use := 1);

Italien Südtirol
HOLIDAY_ITD : ARRAY[0..29] OF HOLIDAY_DATA := (name := 'Neujahr', day := 1, month := 1, use := 1),
        (name := 'Heilig Drei Könige', day := 6, month := 1, use := 1),
        (name := 'Ostersonntag', day := 0, month := 0, use := 1),
        (name := 'Ostermontag', day := 1, month := 0, use := 1),
        (name := 'Tag der Befeiung Italiens', day := 25, month := 4, use := 1),
        (name := 'Tag der Arbeit', day := 1, month := 5, use := 1),
        (name := 'Pfingsten', day := 49, month := 0, use := 1),
        (name := 'Pfingstmontag', day := 50, month := 0, use := 1),
        (name := 'Tag der Republik Italien', day := 2, month := 6, use := 1),
        (name := 'Mariä Himmelfahrt', day := 15, month := 8, use := 1),
        (name := 'Allerheiligen', day := 1, month := 11, use := 1),
        (name := 'Mariä Empfängnis', day := 8, month := 12, use := 1),
        (name := 'Heiligabend', day := 24, month := 12, use := 0),
        (name := '1. Weihnachtstag', day := 25, month := 12, use := 1),
        (name := '2. Stephanstag', day := 26, month := 12, use := 1); *)
```

3.11. REAL2

Die Struktur REAL2 simuliert auf Systemen ohne LREAL eine Gleitpunktzahl doppelter Genauigkeit. jedoch mit Einschränkungen und nur mit speziellen Funktionen.

*.R1 : REAL	Grobteil der Doppelzahl
*.RX : REAL	Feinteil der Doppelzahl

3.12. SDT

Die Struktur SDT definiert ein strukturiertes Zeit- Datums- Format.

*.YEAR : INT	Jahr
*.MONTH : INT	Monat
*.DAY : INT	Tag
*.WEEKDAY : INT	Wochentag (1 = Montag, 7 = Sonntag)
*.HOUR : INT	Stunden
*.MINUTE : INT	Minuten

*.SECOND : INT	Sekunden
*.MS : INT	Millisekunden

3.13. TIMER_EVENT

Die Struktur TIMER_EVENT definiert und speichert Vorgaben für Timer und Ereignisse.

*.TYP : BYTE	Ereignistyp
*.CHANNEL : BYTE	Kanal
*.DAY : BYTE	Tag oder Tage
*.START : TOD	Anfangszeit
*.DURATION : TIME	Dauer
*.LAND : BYTE	Logisches AND
*.LOR : BYTE	Logisches OR
*.LAST : DT	letzte Aktivität des Ereignisses

3.14. VECTOR_3

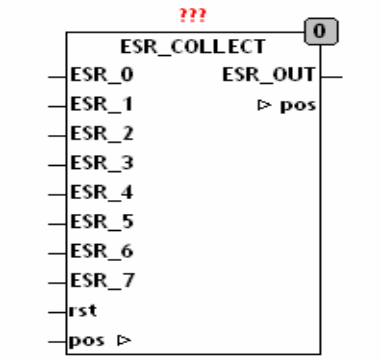
Die Struktur VECTOR_3 speichert die 3 Vektoren eines dreidimensionalen Vektorraums.

*.X : REAL	Komponente in X - Richtung
*.Y : REAL	Komponente in Y - Richtung
*.Z : REAL	Komponente in Z - Richtung

4. Diverse Funktionen

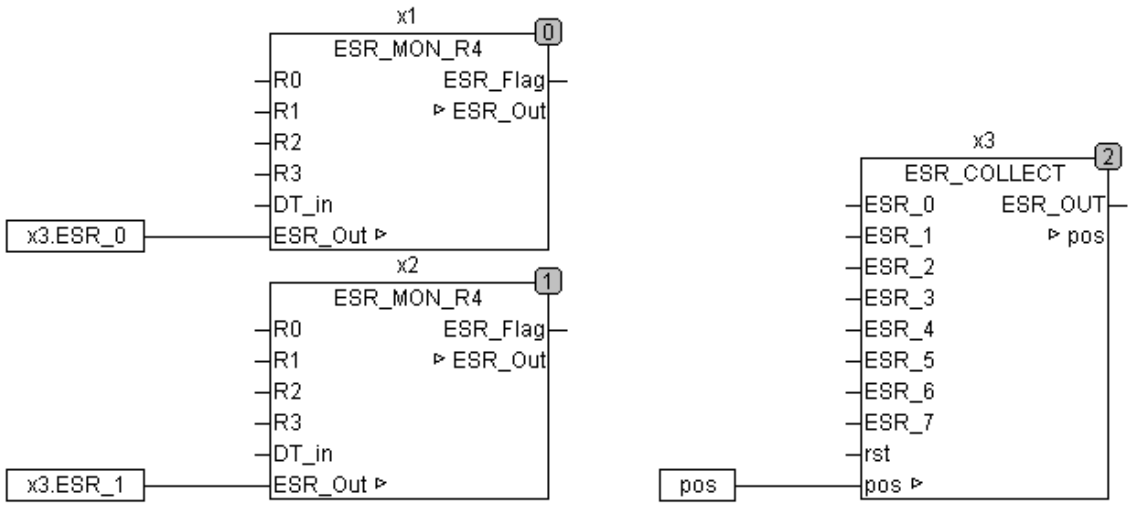
4.1. ESR_COLLECT

Type	Funktionsbaustein
Input	ESR_0..7 : ESR_DATA (ESR_Eingänge)
	RST : BOOL (Asynchroner Reset Eingang)
Output	ESR_OUT : ESR_DATA (Array mit dem ESR-Protokoll)
IN/OUT	POS : INT (Position des neusten ESR Protokolls im Array)



ESR_COLLECT sammelt ESR-Daten von bis zu 8 ESR-Bausteinen und speichert das Protokoll in einem Array. Der Ausgang POS zeigt an, an welcher Position des Arrays ESR_OUT sich die momentan letzte ESR-Meldung befindet. Sammelt der Baustein mehr als 64 Meldungen, so werden die Meldungen verworfen und bei Position 0 neu begonnen. Mit dem Asynchronen Reset Eingang kann der Baustein jederzeit zurückgesetzt werden. Durch einen Reset werden alle gesammelten Reset Daten gelöscht und der Positionszeiger auf -1 gestellt. Der Baustein sammelt Daten im Ausgangsarray ESR_OUT und setzt POS auf die letzte aktuelle Position des Arrays die Daten enthält. Wenn keine Meldungen anstehen bleibt POS auf -1. Werden die Ausgangsdaten ausgelesen, muss die Variable POS auf -1 gesetzt werden, oder falls nur teilweise ausgelesen wird kann POS auf den letzten gültigen Wert gesetzt werden.

Das folgende Beispiel demonstriert, wie ESR_COLLECT mit ESR-Bausteinen zusammen geschaltet wird.



Der Ausgang ESR_OUT setzt sich wie folgt zusammen:

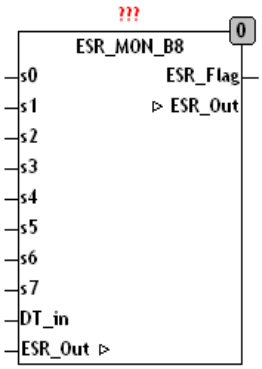
.TYP	.ADRESS	.DS	.TS	.DATA[0..7]	
1	Label	Date	TIME	Status, 1 Byte	ESR Error
2	Label	Date	TIME	Status, 1 Byte	ESR Status
3	Label	Date	TIME	Status, 1 Byte	ESR Debug
10	Label	Date	TIME	not used	Boolean input low transition
11	Label	Date	TIME	not used	Boolean input high transition
20	Label	Date	TIME	Byte 0 - 3 Real Value	Real Value change

Die ESR Daten enthalten folgende Informationen:

ESR_DATA.TYP	Datentyp siehe vorstehende Tabelle
ESR_DATA.ADRESS	bis zu 10 Zeichen langer String Bezeichner
ESR_DATA.DS	Datums Stempel von Typ DATATIME
ESR_DATA.TS	Zeitstempel vom Typ TIME (SPS Timer)
ESR_DATA.DATA	bis zu 8 Byte Datenblock

4.2. ESR_MON_B8

Type	Funktionsbaustein
Input	S0..7 : BOOL (Signaleingänge)
	DT_IN : DATE_TIME (Zeit- Datum-Eingang für Zeitstempel)
Output	ESR_FLAG : BOOL (TRUE, wenn ESR-Daten vorhanden sind)
IN/OUT	ESR_OUT : ESR_Data (ESR_Datenausgang)
Setup	A0..7 : STRING(10) (Bezeichnung der Eingänge)



ESR_MON_B8 überwacht bis zu 8 Binäre Signale auf Änderungen, und versieht sie mit einem Zeitstempel und einer Bezeichnung. Die gesammelten Meldungen werden gepuffert und an einen Protokollbaustein über ESR_OUT weitergereicht. Der Ausgang ESR_FLAG wird auf TRUE gesetzt, wenn Meldungen vorhanden sind.

Die ESR-Daten am Ausgang setzen sich wie folgt zusammen:

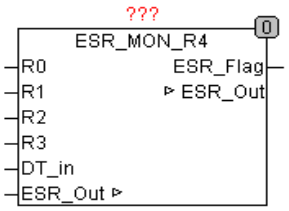
.TYP	11 steigende Flanke, 10 fallende Flanke
.ADRESS	Adresse Byte der ESR-Datenaufzeichnung
.LINE	Liniennummer (Eingang) der ESR-Datenaufzeichnung
.DS	Datumsstempel vom Typ DATE_TIME
.DT	Zeitstempel vom Typ TIME (SPS-Timer)
.Data	Datenblock von 8 Byte unbelegt

Ein Anwendungsbeispiel für den Baustein befindet sich in der Beschreibung von ESR_COLLECT.

4.3. ESR_MON_R4

Type	Funktionsbaustein

Input	R0..3 : REAL (Signaleingänge)
	DT_IN : DATE_TIME (Zeit- Datum-Eingang für Zeitstempel)
Output	ESR_FLAG : BOOL (TRUE, wenn ESR-Daten vorhanden sind)
IN/OUT	ESR_OUT : ESR_Data (ESR_Datenausgang)
Setup	A0..3 : STRING(10) (Signaladresse der Eingänge)
	S0..3 : REAL (Schwellenwerte für Signaländerung)



ESR_MON_R4 überwacht bis zu 4 Analoge Signale auf Änderungen und versieht sie mit einem Zeitstempel und der Signaladresse des entsprechenden Eingangs. Die gesammelten Meldungen werden gepuffert und an einen Protokollbaustein über ESR_OUT weitergereicht. Der Ausgang ESR_FLAG wird auf TRUE gesetzt, wenn Meldungen vorhanden sind. Eine Änderung eines Eingangs wird nur dann aufgezeichnet, wenn sich der Eingang um mehr als durch den Schwellenwert S vorgegebenen Wert geändert hat.

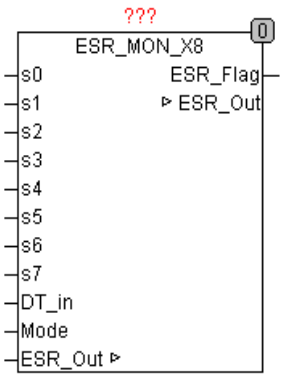
.TYP	20 Gleitpunktwert
.ADRESS	Adresse Byte der ESR-Datenaufzeichnung
.LINE	Liniennummer (Eingang) der ESR-Datenaufzeichnung
.DS	Datumsstempel vom Typ DATE_TIME
.DT	Zeitstempel vom Typ TIME (SPS-Timer)
.Data	Datenblock 4 Byte Real Wert

Ein Anwendungsbeispiel für den Baustein befindet sich in der Beschreibung von ESR_COLLECT.

4.4. ESR_MON_X8

Type	Funktionsbaustein
Input	S0..7 : Byte (Status Eingänge)
	DT_IN : DATE_TIME (Zeit-Datum-Eingang für Zeitstempel)
	Mode : Byte (legt die zu verarbeitende Art von Meldungen fest)

Output	ESR_FLAG : BOOL (TRUE, wenn Meldungen vorhanden sind)
IN/OUT	ESR_OUT : Array[0..7] of ESR_DATA (gesammelte Meldungen)
Setup	A0..7 : STRING(10) (Signaladresse der Eingänge)



ESR_MON_X8 sammelt Status-Meldungen von bis zu 8 ESR kompatiblen Bausteinen ein, versieht sie mit einem Zeitstempel, Datumsstempel, Eingangsnummer und einer Bausteinadresse. Die gesammelten Meldungen werden gepuffert und an einen Protokollbaustein weitergereicht. Wenn Meldungen zur Weitergabe vorhanden sind, wird dies mit dem Signal ESR_FLAG signalisiert. AM Eingang DT_IN sollte die aktuelle Uhrzeit, die für den Zeitstempel der Meldungen dient anliegen. Der Eingang MODE legt fest, welche Statusmeldungen weitergereicht werden sollen.

- Mode = 1, nur Fehlermeldungen werden verarbeitet.
- Mode = 2, Status- und Fehlermeldungen werden verarbeitet.
- Mode = 3, Fehler, Status und Debug-Meldungen werden verarbeitet.

Wenn der Eingang MODE nicht beschaltet wird, werden automatisch alle Meldungen verarbeitet.

Die ESR-Daten am Ausgang setzen sich wie folgt zusammen:

.TYP	1=Fehler, 2=Status, 3=Debug
.ADRESS	Adresse Byte der ESR-Datenaufzeichnung
.LINE	Liniennummer (Eingang) der ESR-Datenaufzeichnung
.DS	Datumsstempel vom Typ DATE_TIME
.DT	Zeitstempel vom Typ TIME (SPS-Timer)
.Data	Data Byte 0 enthält die Statusmeldung

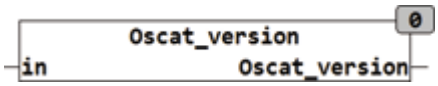
Ein Anwendungsbeispiel für den Baustein befindet sich in der Beschreibung von ESR_COLLECT.

4.5. OSCAT_VERSION

Type Funktion : DWORD

Input IN : BOOL (wenn TRUE liefert der Baustein das Release Datum)

Output (Version der Bibliothek)



OSCAT_VERSION gibt wenn IN = FALSE die aktuelle Versionsnummer als DWORD zurück. Wird IN auf TRUE gesetzt so wird das Release Datum der aktuellen Version als DWORD zurückgegeben.

Beispiel: OSCAT_VERSION(FALSE) = 201 für Version 2.60

DWORD_TO_DATE(OSCAT_VERSION(TRUE)) = 2008-1-1

4.6. STATUS_TO_ESR

Type Funktion : ESR_DATA

Input STATUS : BYTE (Status Byte)

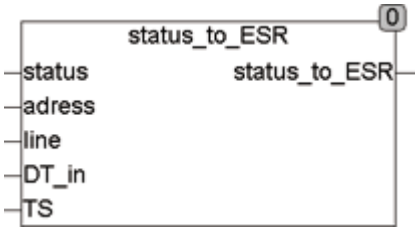
ADRESS : Byte (Adresse, Byte)

LINE : Byte (Eingangsnummer)

DT_IN : DATE_TIME (Zeit-Datum-Eingang)

TS : TIME (Zeit für Zeitstempel)

Output ESR_DATA (ESR-Datenblock)



STATUS_TO_ESR erzeugt einen ESR Datensatz aus den Eingangswerten.

Ein STATUS im Bereich zwischen 1 .. 99 ist eine Fehlermeldung und wird als Typ 1 gekennzeichnet. Status 100 .. 199 wird als Typ 2 gekennzeichnet und 200 .. 255 wird als Typ 3 gekennzeichnet (Debug-Information).

Die ESR-Daten am Ausgang setzen sich wie folgt zusammen:

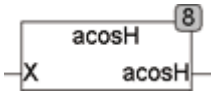
.TYP 1=Fehler, 2=Status, 3=Debug

.ADRESS	Adresse Byte der ESR-Datenaufzeichnung
.LINE	Liniennummer (Eingang) der ESR-Datenaufzeichnung
.DS	Datumsstempel vom Typ DATE_TIME
.DT	Zeitstempel vom Typ TIME (SPS-Timer)
.Data	Datenblock von 8 Byte

5. Mathematik

5.1. ACOSH

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
Output	REAL (Ausgangswert)

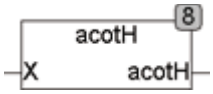


ACOSH Berechnet den Arcus Cosinus Hyperbolicus nach folgender Formel:

$$ACOSH = \ln\left(X + \sqrt{X^2 - 1}\right)$$

5.2. ACOTH

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
Output	REAL (Ausgangswert)



ACOTH Berechnet den Arcus Cotangens Hyperbolicus nach folgender Formel:

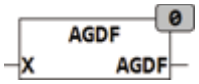
$$acoth = \frac{1}{2}\ln\left(\frac{(x+1)}{(x-1)}\right)$$

5.3. AGDF

Type	Funktion : REAL

Input X : REAL (Eingangswert)

Output REAL (Inverse Gundermannfunktion)



AGDF berechnet die Inverse Gundermannfunktion.

Die Berechnung erfolgt nach der Formel:

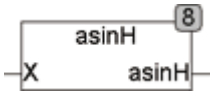
$$agdf = \ln\left(\frac{1+\sin(X)}{\cos(X)}\right)$$

5.4. ASINH

Type Funktion : REAL

Input X : REAL (Eingangswert)

Output REAL (Ausgangswert)



ASINH Berechnet den Arcus Sinus Hyperbolicus nach folgender Formel:

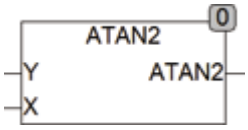
$$asinh = \ln\left(X + \sqrt{X^2+1}\right)$$

5.5. ATAN2

Type Funktion : REAL

Input X : REAL (Eingangswert)

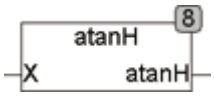
Output REAL (Ausgangswert)



ATAN2 Berechnet den Winkel von Koordinaten (Y, X) in RAD. Das Ergebnis liegt zwischen -π und +π

5.6. ATANH

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
Output	REAL (Ausgangswert)

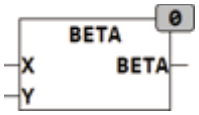


ATANH Berechnet den Arcus Tangens Hyperbolicus nach folgender Formel:

$$atanh = \frac{1}{2} \ln\left(\frac{(1+x)}{(1-X)}\right)$$

5.7. BETA

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
	Y : REAL (Eingangswert)
Output	REAL (Ausgangswert)

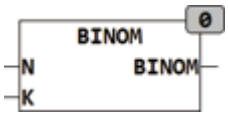


BETA berechnet die Eulersche Beta Funktion.

$$B(x,y) = \int_0^1 t^{x-1}(1-t)^{y-1}\,\mathrm{d}t$$

5.8. BINOM

Type	Funktion : DINT
Input	N : INT (Eingangswert)
	K : INT (Eingangswert)
Output	DINT (Ausgangswert)

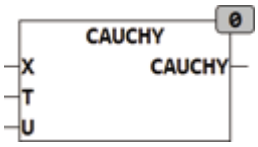


BINOM berechnet den Binominialkoeffizienten N über K für Ganzzahlige N und K.

$$\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

5.9. CAUCHY

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
	T : REAL (Eingangswert)
	U : REAL (Eingangswert)
Output	REAL (Ausgangswert)

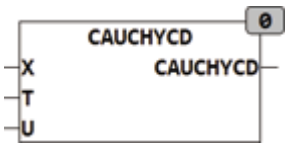


CAUCHY berechnet die Dichtefunktion nach Cauchy.

$$f(x) = \frac{1}{\pi} \cdot \frac{s}{s^2 + (x-t)^2}$$

5.10. CAUCHYCD

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
	T : REAL (Eingangswert)
	U : REAL (Eingangswert)
Output	REAL (Ausgangswert)

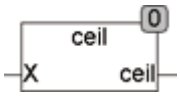


CAUCHYCD berechnet die Verteilungsfunktion nach Cauchy.

$$F(x) = \frac{1}{2} + \frac{1}{\pi} \cdot \arctan\left(\frac{x-t}{s}\right)$$

5.11. CEIL

Type	Funktion : INT
Input	X : REAL (Eingangswert)
Output	INT (Ausgangswert)

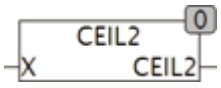


Die Funktion CEIL gibt den kleinsten ganzzahligen Wert größer gleich X zurück.

Beispiel:	CEIL(3.14) = 4
	CEIL(-3.14) = -3
	CEIL(2) = 2

5.12. CEIL2

Type	Funktion : DINT
Input	X : REAL (Eingangswert)
Output	DINT (Ausgangswert)

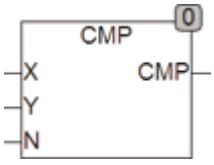


Die Funktion CEIL2 gibt den kleinsten ganzzahligen Wert größer gleich X zurück.

Beispiel: CEIL2(3.14) = 4
CEIL2(-3.14) = -3
CEIL2(2) = 2

5.13. CMP

Type	Funktion : BOOL
Input	X, Y : REAL (Eingangswerte)
	N : INT (Anzahl der Stellen die verglichen werden sollen)
Output	BOOL (Ergebnis)



CMP vergleicht 2 REAL Werte ob die ersten N Stellen gleich Sind.

Beispiele:

CMP(3.140,3.149,3) = TRUE CMP(3.140,3.149,4) = FALSE

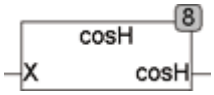
CMP(0.015,0,016,1) = TRUE CMP(0.015,0,016,2) = FALSE

Bei der Funktion CMP ist zu beachten das durch die Duale Kodierung der Zahlen eine 0.1 im Dezimalsystem nicht unbedingt immer als 0.1 im Binärsystem dargestellt werden kann. Vielmehr kann es vorkommen das die 0.1 etwas kleiner oder größer dargestellt wird weil die Auflösung der Zahlencodierung im Binärsystem nicht exakt eine 0.1 zulässt. Aus diesem Grund kann die Funktion nicht immer zu 100% einen Unterschied von 1 an

der letzten Stelle erkennen. Weiterhin ist zu beachten das ein Datentyp REAL mit 32 Bit nur eine Auflösung von 7 - 8 Dezimalstellen bietet.

5.14. COSH

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
Output	REAL (Ausgangswert)

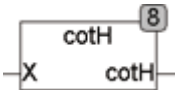


COSH Berechnet den Cosinus Hyperbolicus nach folgender Formel:

$$\cosh = \frac{e^{X} + e^{-X}}{2}$$

5.15. COTH

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
Output	REAL (Ausgangswert)



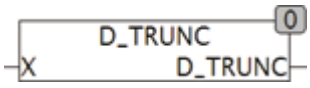
COTH Berechnet den Cotangens Hyperbolicus nach folgender Formel:

$$\coth = \frac{e^{2X} + 1}{e^{2X} - 1}$$

für Eingangswerte größer 20 oder kleiner als -20 liefert COTH den Näherungswert +1 beziehungsweise -1 was einer Genauigkeit besser 8 Stellen entspricht und somit unterhalb der Auflösung des Typs REAL liegt.

5.16. D_TRUNC

Type	Funktion : DINT
Input	X : REAL (Eingangswert)
Output	DINT (Ausgangswert)



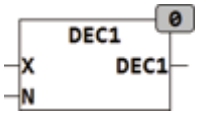
D_TRUNC liefert den ganzzahligen Teil eines REAL Wertes als DINT. Die IEC Routine TRUNC() unterstützt nicht auf allen Systemen ein TRUNC nach DINT so dass wir aus Gründen der Kompatibilität diese Routine nachgebildet haben. Leider liefert auch ein REAL_TO_DINT nicht auf allen Systemen dasselbe Ergebnis. D_TRUNC prüft welches Ergebnis die IEC Funktionen liefern und nutzt die passende Funktion um ein brauchbares Ergebnis zu liefern.

D_TRUNC(1.6) = 1

D_TRUNC(-1.6) = -1

5.17. DEC1

Type	Funktion : INT
Input	X :INT (Anzahl der Werte die X einnehmen kann)
	N : INT (Variable die Hochgezählt wird)
Output	INT (Rückgabewert)



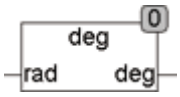
DEC1 zählt die Variable X von N-1 bis 0 und beginnt dann wieder bei N-1, so dass genau N verschiedene Werte beginnend bei N-1 bis einschließlich 0 erzeugt werden.

5.18. DEG

Type	Funktion : REAL

Input Rad : REAL (Winkel in Bogenmaß)

Output REAL (Winkel in Grad)



Die Funktion DEG konvertiert einen Winkelwert von Bogenmaß in Grad. Dabei wird berücksichtigt das der Eingang nicht größer als 2π sein darf. Wenn RAD größer als 2π ist wird entsprechend 2π abgezogen bis der Eingang RAD zwischen 0 und 2π liegt.

DEG(π) = 180 Grad, DEG(3π) = 180 Grad

DEG(0) = 0 Grad, DEG(2π) = 0 Grad

5.19. DIFFER

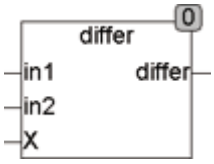
Type Funktion : BOOL

Input IN1 : REAL (Wert 1)

IN2 : REAL (Wert 2)

X : REAL (Mindestunterschied in1 zu in2)

Output BOOL (TRUE, wenn sich in1 und in2 um mehr als x voneinander unterscheiden)



Die Funktion DIFFER wird TRUE, wenn in1 und in2 sich durch mehr als X voneinander unterscheiden.

$$differ = |(in1 - in2)| > X$$

Beispiel: Differ(100, 120, 10) ergibt TRUE

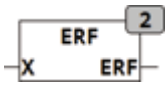
Differ(100,110,15) ergibt FALSE

5.20. ERF

Type Funktion : REAL

Input X : REAL (Eingangswert)

Output REAL (Ergebnis)



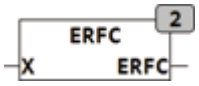
Die Funktion ERF berechnet die Fehlerfunktion von X. Die Fehlerfunktion wird mittels einer Näherungsformel berechnet, der maximale relative Fehler ist dabei kleiner als $1,3 * 10^{-4}$.

5.21. ERFC

Type Funktion : REAL

Input X : REAL (Eingangswert)

Output REAL (Ergebnis)



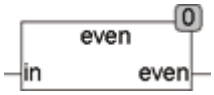
Die Funktion ERFC berechnet die inverse Fehlerfunktion von X.

5.22. EVEN

Type Funktion : BOOL

Input in : DINT (Eingangswert)

Output BOOL (TRUE, wenn in gerade)



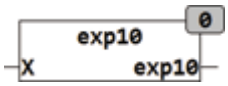
Die Funktion EVEN wird TRUE, wenn der Eingang IN gerade ist und FALSE für ungerade IN.

Beispiel: EVEN(2) ergibt TRUE

EVEN(3) ergibt FALSE

5.23. EXP10

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
Output	REAL (Exponentialfaktor zur Basis 10)



Die Funktion EXP10 liefert den Exponentialwert zur Basis 10.

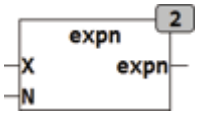
EXP10(2) = 100

EXP10(0) = 1

EXP10(3.14) = 1380.384265

5.24. EXPN

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
	N : INT (Exponentialwert)
Output	REAL (Ergebnis X^N)



EXPN berechnet den Exponentialwert von X^N für Ganzzahlige N. EXPN ist speziell für SPS ohne Floating Point Einheit geschrieben und ist ca. 30 mal schneller als die IEC Standard Funktion EXPT(). Zu beachten ist der Sonderfall das 0^0 wie mathematisch definiert eine 1 ergibt und nicht eine 0.

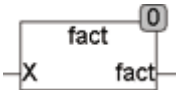
EXPN(10,-2) = 0.01 EXPN(1.5,2) = 2.25

EXPN(0,0) = 1

5.25. FACT

Type	Funktion : DINT
Input	X : INT (Eingangswert)

Output DINT (Fakultät von X)



Die Funktion FACT berechnet die Fakultät von X.

Sie ist definiert für Eingangswerte von 0 .. 12. Für Werte kleiner Null und größer 12 wird das Ergebnis -1. für die Fakultät von größeren Zahlen ist die Funktion GAMMA geeignet.

Für natürliche Zahlen X ist: X! = 1*2*3...*(X-1)*X, 0! = 1

Fakultäten für negative oder nicht ganze Zahlen sind nicht definiert.

Beispiel: 1! = 1

2! = 1*2 = 2

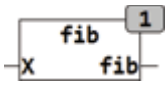
5! = 1*2*3*4*5 = 120

5.26. FIB

Type Funktion : DINT

Input X : INT (Eingangswert)

Output DINT (Fibonacci Zahl)



FIB berechnet die Fibonacci Zahl. Die Fibonacci Zahl ist wie folgt definiert:

FIB(0) = 0, FIB(1) = 1, FIB(2) = 1, FIB(3) = 2, FIB(4) = 3, FIB(5) = 5 .....

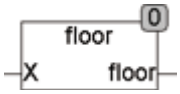
Die Fibonacci Zahl von X ist gleich der Summe der Fibonacci Zahlen von X-1 und X-2. Die Funktion kann die Fibonacci Zahlen bis 46 berechnen, ist X < 0 oder Größer als 46 gibt die Funktion -1 zurück.

5.27. FLOOR

Type Funktion : INT

Input X : REAL (Eingangswert)

Output INT (Ausgangswert)

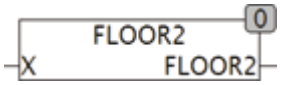


Die Funktion FLOOR gibt den größten ganzzahligen Wert kleiner gleich X zurück.

Beispiel: FLOOR(3.14) = 3
FLOOR(-3.14) = -4
FLOOR(2) = 2

5.28. FLOOR2

Type Funktion : DINT
Input X : REAL (Eingangswert)
Output DINT (Ausgangswert)

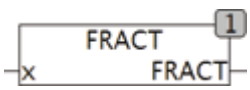


Die Funktion FLOOR2 gibt den größten ganzzahligen Wert kleiner gleich X zurück.

Beispiel: FLOOR2(3.14) = 3
FLOOR2(-3.14) = -4
FLOOR2(2) = 2

5.29. FRACT

Type Funktion : REAL
Input X : REAL (Eingangswert)
Output REAL (Nachkommateil von X)



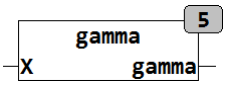
Die Funktion FRACT liefert den Nachkommateil von X.

Beispiel: FRACT(3.14) ergibt 0.14.

Für X größer oder kleiner als +/- 2.14 * 10^9 liefert FRACT stets ein Null zurück. Das die Auflösung eines 32Bit REAL maximal bei 8 Dezimalstellen liegt kann aus Zahlen größer oder kleiner als +/- 2.14 * 10^9 kein Nachkommateil ermittelt werden, weil dieser auch in einer REAL Variable nicht abgespeichert werden kann.

5.30. GAMMA

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
Output	REAL (Gamma Funktion)



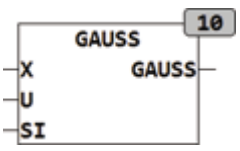
Die Funktion GAMMA berechnet die Gamma Funktion nach der Näherungsformel von NEMES.

$$GAMMA(x) = \sqrt{\frac{2\pi}{x}} * \left(\frac{1}{e} \left(x + \frac{1}{12\mathrm{x} - \frac{1}{10\mathrm{x}}} \right) \right)$$

Die Gamma Funktion kann für Ganzzahlige X als Ersatz für Die Fakultät verwendet werden.

5.31. GAUSS

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
	U : REAL (Lokalität der Funktion)
	SI : REAL (Sigma, Spreizung der Funktion)
Output	REAL (Gauss Funktion)



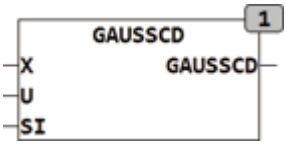
Die Funktion GAUSS berechnet die Normalverteilung nach folgender Formel:

$$gauss = \frac{1}{SI * \sqrt{2\pi}} * e^{-\frac{1}{2} * \left(\frac{X-U}{SI}\right)^2}$$

Die Normalverteilung ist die Dichtefunktion Normalverteilter Zufallsgrößen. Mit den Parameter U = 0 und SI = 1 entspricht sie der Standard Normalverteilung.

5.32. GAUSSCD

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
	U : REAL (Lokalität der Funktion)
	SI : REAL (Sigma, Spreizung der Funktion)
Output	REAL (Gauss Verteilungsfunktion)



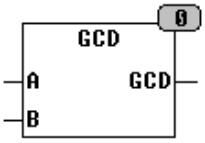
Die Funktion GAUSSCD berechnet die Verteilungsfunktion zur Normalverteilung nach folgender Formel:

$$gausscd = \frac{1}{2} erf\left((X-U) * SI * \sqrt{\pi}\right)$$

Die Normalverteilung ist die Dichtefunktion Normalverteilter Zufallsgrößen. Mit den Parameter U = 0 und SI = 1 entspricht sie der Standard Normalverteilung. Die Verteilungsfunktion (Cumulative Distribution Function).

5.33. GCD

Type	Funktion
Input	A : DINT (Eingangswert A)
	B : DINT (Eingangswert B)
Output	INT (Größter gemeinsamer Teiler)



Die Funktion GCD berechnet den größten gemeinsamen Teiler (GGT) von A und B.

5.34. GDF

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
Output	REAL (Gundermannfunktion)

GDF berechnet die Gundermannfunktion.

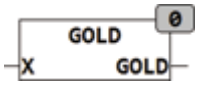
Die Berechnung erfolgt nach der Formel: $gdf = 2\,atan(e^{X}) - \frac{\pi}{2}$

Das Ergebnis von GDF liegt zwischen -π/2 und +π/2.

GDF(0) = 0

5.35. GOLD

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
Output	REAL (Ergebnis der Goldenen Funktion)

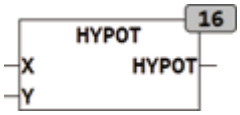


GOLD berechnet das Ergebnis der goldenen Funktion. GOLD(1) ergibt den Goldenen Schnitt, und GOLD(0) ergibt 1. GOLD(X) * GOLD(-X) ergibt immer 1. GOLD(X) ist das positive Ergebnis der Quadratischen Gleichung und -GOLD(-X) ist das negative Ergebnis der Quadratischen Gleichung.

Die Berechnung erfolgt nach der Formel: $gold = \frac{(X + \sqrt{X^2 + 4})}{2}$

5.36. HYPOT

Type	Funktion : REAL
Input	X : REAL (X - Wert)
	Y : REAL (Y - Wert)
Output	REAL (Länge der Hypotenuse)

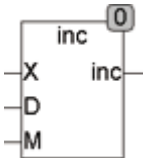


Die Funktion HYPOT berechnet die Hypotenuse eines Rechtwinkeligen Dreiecks nach dem Satz des Pythagoras.

$hypot = \sqrt{x^2 + Y^2}$

5.37. INC

Type	Funktion : INT
Input	X : INT (Eingangswert)
	D : INT (Wert, der zum Eingangswert addiert wird)
	M : INT (Maximalwert für den Ausgang)
Output	INT (Ausgangswert)



INC addiert zum Eingang X den Wert D und stellt sicher, dass der Ausgang INC nicht über den Wert M läuft. Ist das Ergebnis aus der Addition von X und D größer als M so wird wieder bei 0 begonnen. Die Funktion ist vor allem Sinnvoll beim adressieren von Arrays und Pufferspeichern. Auch beim Positionieren von Absolutwert Winkelgebern kann sie eingesetzt werden. INC kann auch mit einem negativen D zum decrementieren benutzt wer-

den, dabei stellt INC sicher dass das Ergebnis nicht unter Null läuft. Wird von Null eins abgezogen beginnt INC wieder bei M.

INC := X + D, weil D maximal den Wert M annehmen kann.

Wird INC > M so beginnt INC wieder bei 0.

Wird INC < 0 so beginnt INC wieder bei M

Beispiel: INC(3, 2, 5) ergibt 5

INC(4, 2, 5) ergibt 0

INC(0,-1,7) ergibt 7

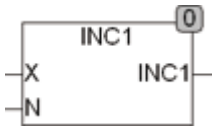
5.38. INC1

Type Funktion : INT

Input X : INT (Anzahl der Werte die X einnehmen kann)

N : INT (Variable die Hochgezählt wird)

Output INT (Rückgabewert)



INC1 zählt die Variable X von 0 .. N-1 und beginnt dann wieder bei 0, so dass genau N verschiedene Werte beginnend bei 0 erzeugt werden.

5.39. INC2

Type Funktion : INT

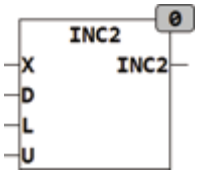
Input X : INT (Eingangswert)

D : INT (Wert, der zum Eingangswert addiert wird)

L : INT (unterer Grenzwert)

U : INT (oberer Grenzwert)

Output INT (Ausgangswert)



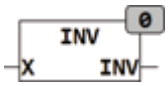
INC2 addiert zum Eingang X den Wert D und stellt sicher, dass der Ausgang INC nicht über den Wert U (oberer Grenzwert) oder unter den Wert L (unterer Grenzwert) läuft. Ist das Ergebnis aus der Addition von X und D größer als U so wird wieder bei L begonnen, Bei negativen D wird sichergestellt das bei Erreichen von L wieder bei U weiter gezählt wird. Die Funktion ist vor allem Sinnvoll beim adressieren von Arrays und Pufferspeichern. Auch beim Positionieren von Absolutwert Winkelgebern kann sie eingesetzt werden. INC2 kann auch mit einem negativen D zum decrementieren benutzt werden, dabei stellt INC2 sicher dass das Ergebnis nicht unter L läuft.

INC2 := X + D, wobei L <= INC2 <= U gilt.

Beispiel: INC2(2, 2, -1, 3) ergibt -1
INC2(2, -2, -1, 3) ergibt 0
INC2(2, 1, -1, 3) ergibt 3
INC2(0, -2, -1, 3) ergibt 3

5.40. INV

Type Funktion : REAL
Input X : REAL (Eingangswert)
Output REAL (1 / X)



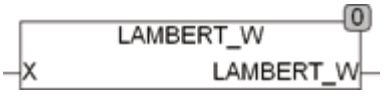
INV berechnet den Kehrwert von X:

$$INV = \frac{1}{X}$$

5.41. LAMBERT_W

Type Funktion : REAL
Input X : REAL (Eingangswert)

Output REAL (Ausgangswert)



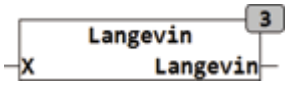
Die LAMBERT_W Funktion ist definiert für X >= -1/e. Bei Unterschreitung des Wertebereichs wird das Ergebnis -1000. Der Wertebereich der LAMBERT_W Funktion ist >= -1.

5.42. LANGEVIN

Type Funktion : REAL

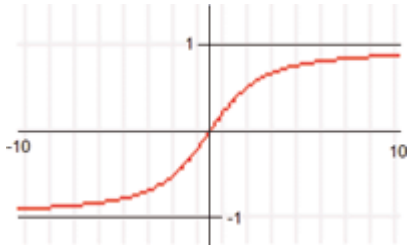
Input X : REAL (Eingangswert)

Output REAL (Ausgangswert)



Die Langevin Funktion ist der Sigmoidfunktion sehr ähnlich, nähert sich aber langsamer den Grenzwerten an. Im Gegensatz zur Sigmoidfunktion liegen die Grenzwerte bei -1 und +1. Die Langevin Funktion ist vor allem auf CPUs ohne Gleitkommaeinheit deutlich schneller als die Sigmoidfunktion.

Die folgende Grafik verdeutlicht den Verlauf der Langevin Funktion:



5.43. MAX3

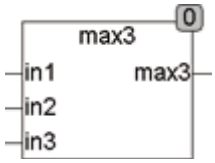
Type Funktion : REAL

Input IN1 : REAL (Eingang 1)

IN2 : REAL (Eingang 2)

IN3 : REAL (Eingang 3)

Output REAL (Maximalwert der 3 Eingänge)



Die Funktion MAX3 liefert den Maximalwert von 3 Eingängen. Grundsätzlich sollte die im Standard Funktionsumfang nach IEC61131-3 enthaltene Funktion MAX mit einer variablen Anzahl von Eingängen ausgestattet sein. Da aber in einigen Systemen die Funktion MAX nur 2 Eingänge unterstützt wird die Funktion MAX3 angeboten.

Beispiel: MAX3(1,3,2) = 3.

5.44. MID3

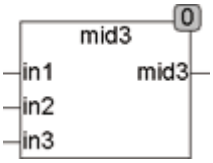
Type Funktion : REAL

Input IN1 : REAL (Eingang 1)

IN2 : REAL (Eingang 2)

IN3 : REAL (Eingang 3)

Output REAL (Mittlerer Wert der 3 Eingänge)



Die Funktion MID3 liefert den mittleren Wert von 3 Eingängen, nicht aber den Mathematischen Mittelwert.

Beispiel: MID3(1,5,2) = 2.

5.45. MIN3

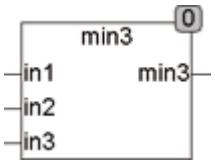
Type Funktion : REAL

Input IN1 : REAL (Eingang 1)

IN2 : REAL (Eingang 2)

IN3 : REAL (Eingang 3)

Output REAL (Minimalwert der 3 Eingänge)



Die Funktion MIN3 liefert den Minimalwert von 3 Eingängen. Grundsätzlich sollte die im Standard Funktionsumfang nach IEC61131-3 enthaltene Funktion MIN mit einer variablen Anzahl von Eingängen ausgestattet sein. Da aber in einigen Systemen die Funktion MIN nur 2 Eingänge unterstützt wird die Funktion MIN3 angeboten.

Beispiel: MIN3(1,3,2) = 1.

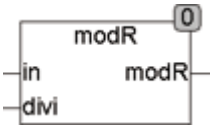
5.46. MODR

Type Funktion : REAL

Input IN : REAL (Dividend)

DIVI : REAL (Divisor)

Output REAL (Rest der Division)



Die Funktion MODR liefert den Rest einer Division analog der Standardfunktion MOD, allerdings für REAL-Zahlen. MODR nutzt intern das Datenformat vom Typ DINT. Hierbei kann es zu einem Überlauf kommen weil DINT maximal +/- 2.14 * 10^9 speichern kann. Der Wertebereich von MODR ist deshalb auf +/- 2.14 * 10^9 begrenzt. Für DIVI = 0 liefert die Funktion 0 zurück.

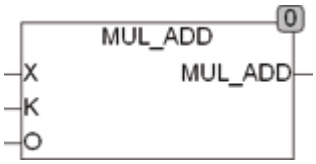
MODR(A, M) = A - M * FLOOR2(A / M).

Beispiel: MODR(5.5, 2.5) ergibt 0.5.

5.47. MUL_ADD

Type Funktion : REAL

Input X : REAL (Eingangswert)
K : REAL (Multiplikator)
O : REAL (Offset)
Output : REAL (Ausgangswert)

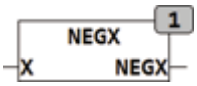


MUL_ADD multipliziert den Eingangswert X mit K und addiert O hinzu.

MUL_ADD = X * K + O.
MUL_ADD(0.5, 10, 2) ergibt 0.5 * 10 + 2 = 7.

5.48. NEGX

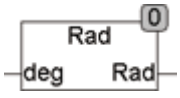
Type Funktion : REAL
Input X : REAL (Eingangswert)
Output REAL (-X)



NEGX liefert den negierten Eingangswert (- X) zurück.

5.49. RAD

Type Funktion : REAL
Input DEG : REAL (Winkel in Grad)
Output REAL (Winkel in Bogenmaß)



Die Funktion RAD konvertiert einen Winkelwert von Grad in Bogenmaß. Dabei wird berücksichtigt das DEG nicht größer als 360 sein darf. Wenn

DEG größer als 360 ist wird solange 360 abgezogen bis DEG zwischen 0 und 360 liegt.

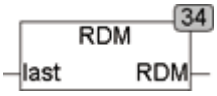
RAD(0) = 0 RAD(180) = π
RAD(360) = 0 RAD(540) = π

5.50. RDM

Type Funktion : REAL

Input LAST : REAL (Letzter berechneter Wert)

Output REAL (Zufallszahl zwischen 0 und 1)



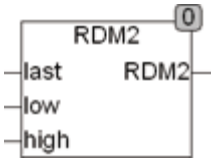
RDM berechnet eine Pseudo-Zufallszahl. Dazu wird der SPS-interne Timer ausgelesen und in eine Pseudo-Zufallszahl überführt. Da RDM als Funktion und nicht als Funktionsbaustein geschrieben wurde, kann es keine Daten zwischen 2 Aufrufen speichern und muss deshalb mit Vorsicht angewendet werden. Wird RDM nur einmal je Zyklus aufgerufen, liefert sie hinreichend gute Zahlen. Wird sie aber mehrfach innerhalb eines Zyklus aufgerufen, so liefert sie dieselbe Zahl, weil höchstwahrscheinlich der SPS Timer noch auf demselben Wert steht. Wird die Funktion mehrfach innerhalb eines Zyklus benötigt, so muss ihr bei jedem Aufruf eine unterschiedliche Startzahl (LAST) übergeben werden. Wird Sie hingegen nur einmal je Zyklus aufgerufen, genügt der Aufruf RDM(0). Als Startzahl kann bei jedem Aufruf die letzte ermittelte Zahl von RDM verwendet werden. Die von RDM gelieferten Zufallszahlen liegen zwischen 0 und 1, die 1 nicht enthalten (0 <= Zufallszahl < 1.

5.51. RDM2

Type Funktion : INT

Input LAST : INT (Letzter berechneter Wert)
LOW : INT (niedrigster generierter Wert)
HIGH : INT (höchster generierter Wert)

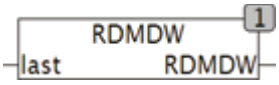
Output INT (Zufallszahl zwischen LOW und HIGH)



RDM2 erzeugt einen Integer Random Wert im Bereich von LOW bis HIGH, wobei LOW und HIGH im Wertebereich enthalten sind. Wird die Funktion nur einmal je Zyklus benutzt kann der Eingangswert LAST auf 0 bleiben. Die Funktion RDM2 benutzt die SPS interne Zeitbasis zur Erzeugung der Zufallszahl. Da RDM2 eine als LAST einen Integer benutzt welcher das letzte Ergebnis zwischen LOW und HIGH darstellt kann es zu einer Situation führen die darin resultiert das RDM2 innerhalb eines Zyklus solange sicher SPS Timer nicht verändert immer das gleiche Ergebnis liefert. Dies passiert immer dann wenn zufällig das Ergebnis auch dem Startwert entspricht. Da dann derselbe Startwert wieder verwendet wird wird innerhalb des gleichen Zyklus auch wieder das gleiche Ergebnis Zustande kommen. Dieser Fall tritt umso öfter auf je kleiner der durch LOW und HIGH bestimmte Bereich für das Ergebnis ist. Man kann diesen Effekt leicht umgehen indem man als Startwert einen Schleifenzähler verwendet der definitiv bei jedem Aufruf einen neuen Wert aufweist, oder noch besser einen Schleifenzähler addiert mit dem letzten Ergebnis als Startwert verwendet.

5.52. RDMDW

Type	Funktion : DWORD
Input	LAST : DWORD (Letzter berechneter Wert)
Output	DWORD (Zufälliges Bitmuster)



RDMDW berechnet eine Pseudo-Zufallszahl mit 32 Bit Länge im Format DWORD. Dazu wird der SPS-interne Timer ausgelesen und in eine Pseudo-Zufallszahl überführt. Da RDMDW als Funktion und nicht als Funktionsbaustein geschrieben wurde, kann es keine Daten zwischen 2 Aufrufen speichern und muss deshalb mit Vorsicht angewendet werden. Wird RDMDW nur einmal je Zyklus aufgerufen, liefert sie hinreichend gute Zahlen. Wird sie aber mehrfach innerhalb eines Zyklus aufgerufen, so liefert sie dieselbe Zahl, weil höchstwahrscheinlich der SPS Timer noch auf demselben Wert steht. Wird die Funktion mehrfach innerhalb eines Zyklus benötigt, so muss ihr bei jedem Aufruf eine unterschiedliche Startzahl (LAST) übergeben werden. Wird Sie hingegen nur einmal je Zyklus aufgerufen, genügt der Aufruf RDMDW(0). Als Startzahl kann bei jedem Aufruf die letzte ermit-

telte Zahl von RDMDW verwendet werden. Das von RDMDW gelieferte Ergebnis ist ein Zufälliges 32Bit breites Bit Muster.

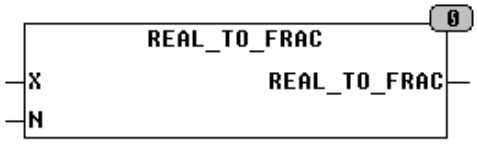
5.53. REAL_TO_FRAC

Type Funktion : FRACTION

Input X : REAL (Eingangswert)

N : INT (maximalwert des Nenners)

Output FRACTION (Ausgangswert)



REAL_TO_FRAC konvertiert eine Gleitpunktzahl (REAL) in einen Bruch. Die Funktion liefert den Datentyp FRACTION der aus einer Struktur mit 2 Werten besteht. Mit dem Eingang X kann die maximale Größe des Zählers vorgegeben werden.

Datentyp FRACTION:

*.NUMERATOR : INT (Zähler des Bruches)

*.DENOMINATOR : INT (Nenner des Bruches)

Beispiel:

REAL_TO_FRAC(3.1415926, 1000) ergibt 355 / 113.

355 / 133 ergibt die beste Approximation für Nenner < 1000.

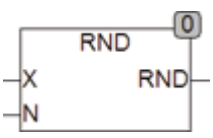
5.54. RND

Type Funktion : REAL

Input X : REAL (Eingangswert)

N : Integer (Anzahl der Stellen)

Output REAL (Gerundeter Wert)



Die Funktion RND rundet den Eingangswert IN auf N Stellen. Folgt der letzten Stelle eine Stelle größer 5 wird die letzte Stelle aufgerundet. RND nutzt intern die Standard Funktion TRUNC() welche den Eingangswert in einen INTEGER vom Typ DINT wandelt. Hierbei kann es zu einem Überlauf kommen weil DINT maximal +/- 2.14 * 10^9 speichern kann. Der Wertebereich von RND ist deshalb auf +/- 2.14 * 10^9 begrenzt. Siehe hierzu auch die Funktion ROUND welche den Eingangswert auf N Nachkommastellen rundet.

Beispiel: RND(355.55, 2) = 360

RND(3.555, 2) = 3.6

ROUND(3.555, 2) = 3.56

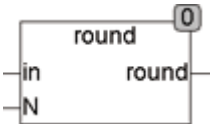
5.55. ROUND

Type Funktion : REAL

Input IN : REAL (Eingangswert)

N : Integer (Anzahl der Nachkommastellen)

Output REAL (Gerundeter Wert)



Die Funktion ROUND rundet den Eingangswert in auf N Nachkommastellen. Folgt der letzten Stelle ein Digit größer 5 wird die letzte Stelle aufgerundet. ROUND nutzt intern die Standard Funktion TRUNC() welche den Eingangswert in einen INTEGER vom Typ DINT wandelt. Hierbei kann es zu einem Überlauf kommen weil DINT maximal +/- 2.14 * 10^9 speichern kann. Der Wertebereich von ROUND ist deshalb auf +/- 2.14 * 10^9 begrenzt.

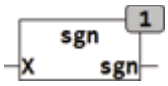
Beispiel: ROUND(3.555, 2) = 3.56

5.56. SGN

Type Funktion : INT

Input X : REAL (Eingangswert)

Output INT (Signum des Eingangs X)



Die Funktion SGN berechnet das Signum von X.

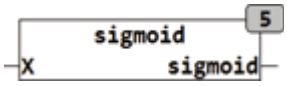
SGN = +1 wenn X > 0

SGN = 0 wenn X = 0

SGN = -1 wenn X < 0

5.57. SIGMOID

Type	Funktion : INT
Input	X : REAL (Eingangswert)
Output	REAL (Ergebnis der Sigmoidfunktion)



Die Sigmoidfunktion wird auch Schwanenhals- Funktion genannt und wird durch die folgende Gleichung beschrieben:

SIGMOID = 1 / (1 + EXP(-X))

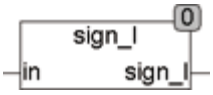
Die Sigmoidfunktion wird häufig als Aktivierungsfunktion verwendet. Durch seinen Verlauf eignet sich die Sigmoidfunktion für weiche Schaltübergänge.

Die folgende Grafik veranschaulicht den Verlauf der Sigmoidfunktion:

5.58. SIGN_I

Type	Funktion : BOOL
Input	IN : DINT (Eingangswert)

Output BOOL (TRUE, wenn der Eingang negativ ist)



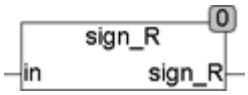
Die Funktion SIGN_I liefert TRUE zurück, wenn der Eingangswert negativ ist. Die Eingangswerte sind vom Typ DINT.

5.59. SIGN_R

Type Funktion : BOOL

Input IN : REAL(Eingangswert)

Output BOOL (TRUE, wenn der Eingang negativ ist)



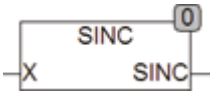
Die Funktion SIGN_R liefert TRUE zurück, wenn der Eingangswert negativ ist. Die Eingangswerte sind vom Typ REAL.

5.60. SINC

Type Funktion : REAL

Input X : REAL (Eingangswert)

Output REAL (Ausgangswert)

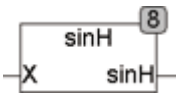


SINC berechnet den Sinus Kardinalis oder die Spaltfunktion.

$sinc = \frac{\sin(x)}{x}$ Mit SINC(0) = 1.

5.61. SINH

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
Output	REAL (Ausgangswert)

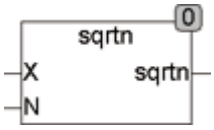


SINH Berechnet den Sinus Hyperbolicus nach folgender Formel:

$$\sinh = \frac{(e^{X} - e^{-X})}{2}$$

5.62. SQRTN

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
	N : INT (Eingangswert)
Output	REAL (Ausgangswert)

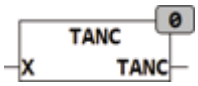


SQRTN Berechnet die N-fache Wurzel aus X nach folgender Formel:

$$sqrtn = \sqrt[N]{X}$$

5.63. TANC

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
Output	REAL (Ausgangswert)

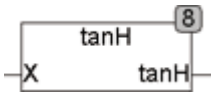


TANC Berechnet die TANC Funktion nach folgender Formel:

$$tanc = \frac{\tan(X)}{X}$$ mit TANC(0) = 1.

5.64. TANH

Type	Funktion : REAL
Input	X : REAL (Eingangswert)
Output	REAL (Ausgangswert)

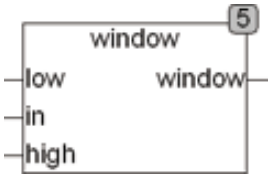


TANH Berechnet den Tangens Hyperbolicus nach folgender Formel:

$$\tanh = 1 - \frac{2}{e^{-2}+1}$$

5.65. WINDOW

Type	Funktion : BOOL
Input	LOW : REAL (unterer Grenzwert)
	IN : REAL (Eingangswert)
	HIGH : REAL (oberer Grenzwert)
Output	BOOL (TRUE, wenn in < HIGH und in > LOW)

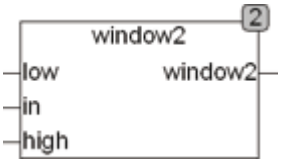


Die Funktion WINDOW testet, ob der Eingangswert IN innerhalb der Grenzen, die durch LOW und HIGH definiert sind, liegt.

WINDOW ist genau dann TRUE, wenn IN < HIGH und IN > LOW ist.

5.66. WINDOW2

Type	Funktion : BOOL
Input	LOW : REAL (unterer Grenzwert)
	IN : REAL (Eingangswert)
	HIGH : REAL (oberer Grenzwert)
Output	BOOL (TRUE, wenn in <= HIGH und in >= LOW)



Die Funktion WINDOW2 testet, ob der Eingangswert IN <= HIGH und IN >= LOW ist. Im Gegensatz zur Funktion WINDOW die TRUE liefert wenn IN innerhalb der Grenzen LOW und HIGH liegt liefert WINDOW2 FALSE wenn IN außerhalb der Grenzen LOW und HIGH liegt.

6. Array

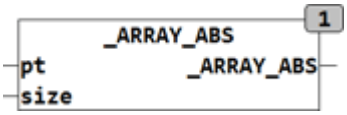
6.1. _ARRAY_ABS

Type Funktion : BOOL

Input PT : Pointer (Zeiger auf das Array)

SIZE : UINT (Größe des Arrays)

Output BOOL (TRUE)



Die Funktion _ARRAY_ABS rechnet die Elemente eines beliebigen Array of REAL in den Absolutwert um. Beim Aufruf wird der Funktion ein Pointer auf das zu initialisierende Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: _ARRAY_ABS(ADR(Array), SIZEOF(Array)), wobei Array der Name des zu manipulierenden Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt. Die Funktion liefert nur TRUE zurück. Das durch den Pointer angegebene Array wird direkt im Speicher manipuliert.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Aufruf: _ARRAY_ABS(ADR(bigarray), SIZEOF(bigarray))

Beipiel: [0,-2,3,-1-5] wird umgewandelt in [0,2,3,1,5]

6.2. _ARRAY_ADD

Type Funktion : BOOL

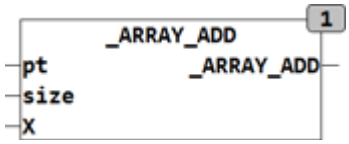
Input PT : Pointer (Zeiger auf das Array)

SIZE : UINT (Größe des Arrays)

X : REAL (zu addierender Wert)

Output BOOL (TRUE)

Die Funktion _ARRAY_ADD addiert zu jedem Element eines beliebigen Ar-



ray of REAL den Wert X. Beim Aufruf wird der Funktion ein Pointer auf das zu initialisierende Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: _ARRAY_ADD(ADR(Array), SIZEOF(Array), X), wobei Array der Name des zu manipulierenden Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt. Die Funktion liefert nur TRUE zurück. Das durch den Pointer angegebene Array wird direkt im Speicher manipuliert.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Aufruf: _ARRAY_ADD(ADR(bigarray), SIZEOF(bigarray), X)

Beipiel: [0,-2,3,-1-5] ; X = 3 wird umgewandelt in [3,1,6,2,-2]

6.3. _ARRAY_INIT

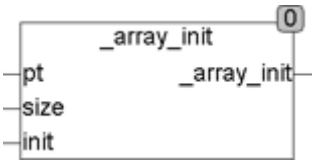
Type Funktion : BOOL

Input PT : Pointer (Zeiger auf das Array)

SIZE : UINT (Größe des Arrays)

INIT : REAL (Initialwert)

Output BOOL (TRUE)



Die Funktion _ARRAY_INIT initialisiert ein beliebiges Array of REAL mit einem Initialwert. Beim Aufruf wird der Funktion ein Pointer auf das zu initialisierende Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: _ARRAY_INIT(ADR(Array), SIZEOF(Array), INIT), wobei Array der Name des zu manipulierenden Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt. Die Funktion liefert nur TRUE zurück. Das durch den Pointer angegebene Array wird direkt im Speicher manipuliert.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel: _ARRAY_INIT(ADR(bigarray), SIZEOF(bigarray), 0)

initialisiert bigarray mit 0.

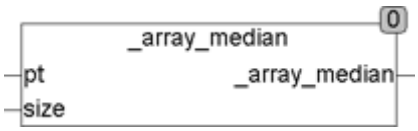
6.4. _ARRAY_MEDIAN

Type Funktion : REAL

Input PT : Pointer (Zeiger auf das Array)

SIZE : UINT (Größe des Arrays)

Output REAL (Medianwert des Arrays)



Die Funktion _ARRAY_MEDIAN ermittelt den Medianwert eines beliebigen Arrays of REAL. Beim Aufruf wird der Funktion ein Pointer auf das Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: _ARRAY_MEDIAN(ADR(Array), SIZEOF(Array)), wobei Array der Name des zu manipulierenden Arrays ist. ADR() ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF() ist eine Standardfunktion, die die Größe des Arrays ermittelt. Um den Medianwert zu ermitteln wird das durch den Pointer referenzierte Array direkt im Speicher sortiert und bleibt nach beenden der Funktion sortiert. Die Funktion _ARRAY_MEDIAN verändert also den Inhalt des Arrays.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Sollte ein Array bearbeitet werden, das nicht verändert werden darf, so ist es vor Übergabe des Pointer und Aufruf der Funktion in ein temporäres Array zu kopieren.

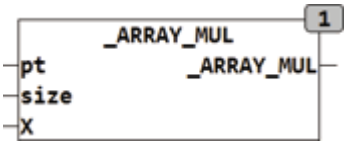
Beispiel: _ARRAY_MEDIAN(ADR(bigarray), SIZEOF(bigarray))

Medianwert:
Der Medianwert ist der Mittlere Wert einer sortierten Wertemenge.
Median von (12, 0, 4, 7, 1) ist 4. Nach dem Ausführen der Funktion bleibt das Array sortiert im Speicher zurück (0, 1, 4, 7, 12).

Wenn das Array eine gerade Anzahl von Elementen enthält ist der Medianwert der Mittelwert aus den beiden mittleren Werten des Arrays.

6.5. _ARRAY_MUL

Type	Funktion : BOOL
Input	PT : Pointer (Zeiger auf das Array)
	SIZE : UINT (Größe des Arrays)
	X : REAL (Multiplikator)
Output	BOOL (TRUE)



Die Funktion _ARRAY_MUL multipliziert jedes Element eines beliebigen Array of REAL mit den Wert X. Beim Aufruf wird der Funktion ein Pointer auf das zu initialisierende Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: _ARRAY_MUL(ADR(Array), SIZEOF(Array), X), wobei Array der Name des zu manipulierenden Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt. Die Funktion liefert nur TRUE zurück. Das durch den Pointer angegebene Array wird direkt im Speicher manipuliert.

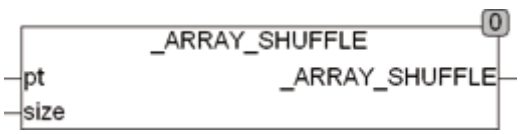
Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Aufruf: _ARRAY_MUL(ADR(bigarray), SIZEOF(bigarray), X)

Beispiel: [0,-2,3,-1-5]; X = 3 wird umgewandelt in [0,-6,9,-3,-15]

6.6. _ARRAY_SHUFFLE

Type	Funktion : BOOL
Input	PT : Pointer (Zeiger auf das Array)
	SIZE : UINT (Größe des Arrays)
Output	BOOL (ergibt TRUE)



Die Funktion _ARRAY_SHUFFLE vertauscht die Elemente eines beliebigen Arrays of REAL nach einem Zufallsprinzip. Beim Aufruf wird der Funktion ein Pointer auf das Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: _ARRAY_SHUFFLE(ADR(Array), SIZEOF(Array)), wobei Array der Name des zu manipulierenden Arrays ist. ADR() ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF() ist eine Standardfunktion, die die Größe des Arrays ermittelt. Das durch den Pointer referenzierte Array wird direkt im Speicher manipuliert und steht nach beenden der Funktion direkt zur Verfügung. Die Funktion _ARRAY_SHUFFLE verändert also den Inhalt des Arrays.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Sollte ein Array bearbeitet werden, das nicht verändert werden darf, so ist es vor Übergabe des Pointer und Aufruf der Funktion in ein temporäres Array zu kopieren.

Beispiel: _ARRAY_SHUFFLE(ADR(bigarray), SIZEOF(bigarray))

Ein Aufruf der Funktion _ARRAY_SHUFFLE könnte ein Array wie folgt verändern. Da die Funktion einen Pseudo Random Algorithmus verwendet wird das Ergebnis bei jedem Aufruf ein anderes sein, die Ergebnisse sind nicht reproduzierbar, auch nicht durch einen Neustart des Programms oder der Steuerung.

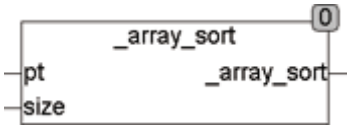
Ausgangsarray: (0,1,2,3,4,5,6,7,8,9)

Ergebnis: (5,0,3,9,7,2,1,8,4,6)

Das Ergebnis ist aber nicht wiederholbar, die Funktion liefert nach jedem Aufruf oder Auch Neustart eine neue Reihenfolge.

6.7. _ARRAY_SORT

Type	Funktion : BOOL
Input	PT : Pointer (Zeiger auf das Array)
	SIZE : UINT (Größe des Arrays)
Output	BOOL (TRUE)



Die Funktion _ARRAY_SORT sortiert ein beliebiges Array of REAL in aufsteigender Reihenfolge. Beim Aufruf wird der Funktion ein Pointer auf das zu

sortierende Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: _ARRAY_SORT(ADR(Array), SIZEOF(Array)), wobei Array der Name des zu sortierenden Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt.

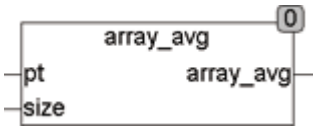
Die Funktion liefert nur TRUE zurück. Das durch den Pointer angegebene Array wird direkt im Speicher Manipuliert.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel: _ARRAY_SORT(ADR(bigarray), SIZEOF(bigarray))

6.8. ARRAY_AVG

Type	Funktion : REAL
Input	PT : Pointer (Zeiger auf das Array)
	SIZE : UINT (Größe des Arrays)
Output	REAL (Mittelwert des Arrays)



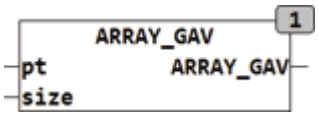
Die Funktion ARRAY_AVG ermittelt den Mittelwert eines beliebigen Arrays of REAL. Bei Aufruf wird der Funktion ein Pointer auf das Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: ARRAY_AVG(ADR(Array), SIZEOF(Array)), wobei Array der Name des zu durchsuchenden Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt. Um den Maximalwert zu ermitteln wird das durch den Pointer referenzierte Array direkt im Speicher durchsucht. Die Funktion ARRAY_AVG verändert den Inhalt des Arrays nicht.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel: ARRAY_AVG(ADR(bigarray), SIZEOF(bigarray))

6.9. ARRAY_GAV

Type	Funktion : REAL
Input	PT : Pointer (Zeiger auf das Array)
	SIZE : UINT (Größe des Arrays)
Output	REAL (Mittelwert des Arrays)



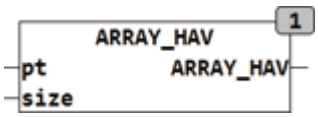
Die Funktion ARRAY_GAV ermittelt den geometrischen Mittelwert eines beliebigen Arrays of REAL. Bei Aufruf wird der Funktion ein Pointer auf das Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: ARRAY_GAV(ADR(Array), SIZEOF(Array)), wobei Array der Name des zu durchsuchenden Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt. Um den Maximalwert zu ermitteln wird das durch den Pointer referenzierte Array direkt im Speicher durchsucht. Die Funktion ARRAY_GAV verändert den Inhalt des Arrays nicht.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel: ARRAY_GAV(ADR(bigarray), SIZEOF(bigarray))

6.10. ARRAY_HAV

Type	Funktion : REAL
Input	PT : Pointer (Zeiger auf das Array)
	SIZE : UINT (Größe des Arrays)
Output	REAL (Mittelwert des Arrays)



Die Funktion ARRAY_HAV ermittelt den harmonischen Mittelwert eines beliebigen Arrays of REAL. Bei Aufruf wird der Funktion ein Pointer auf das Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der

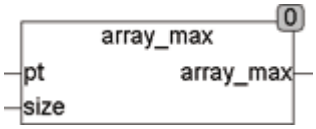
Aufruf: ARRAY_HAV(ADR(Array), SIZEOF(Array)), wobei Array der Name des zu durchsuchenden Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt. Um den Maximalwert zu ermitteln wird das durch den Pointer referenzierte Array direkt im Speicher durchsucht. Die Funktion ARRAY_HAV verändert den Inhalt des Arrays nicht.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel: ARRAY_HAV(ADR(bigarray), SIZEOF(bigarray))

6.11. ARRAY_MAX

Type	Funktion : REAL
Input	PT : Pointer (Zeiger auf das Array)
	SIZE : UINT (Größe des Arrays)
Output	REAL (Maximalwert des Arrays)



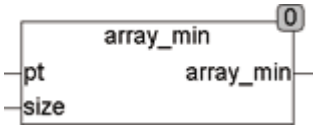
Die Funktion ARRAY_MAX ermittelt den Maximalwert eines beliebigen Arrays of REAL. Bei Aufruf wird der Funktion ein Pointer auf das Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: Array_MAX(ADR(Array), SIZEOF(Array)), wobei Array der Name des zu durchsuchenden Arrays ist. ADR ist eine Standard Funktion die den Pointer auf das Array ermittelt und SIZEOF ist eine Standard Funktion die die Größe des Arrays ermittelt. Um den Maximalwert zu ermitteln wird das durch den Pointer referenzierte Array direkt im Speicher durchsucht. Die Funktion ARRAY_MAX verändert den Inhalt des Arrays nicht.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel: ARRAY_MAX(ADR(bigarray), SIZEOF(bigarray))

6.12. ARRAY_MIN

Type	Funktion : REAL
Input	PT : Pointer (Zeiger auf das Array)
	SIZE : UINT (Größe des Arrays)
Output	REAL (Minimalwert des Arrays)



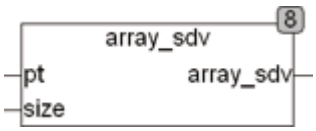
Die Funktion ARRAY_MIN ermittelt den Minimalwert eines beliebigen Arrays of REAL. Beim Aufruf wird der Funktion ein Pointer auf das Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: ARRAY_MIN(ADR(Array), SIZEOF(Array)), wobei Array der Name des zu durchsuchenden Arrays ist. ADR() ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF() ist eine Standardfunktion, die die Größe des Arrays ermittelt. Um den Maximalwert zu ermitteln wird das durch den Pointer referenzierte Array direkt im Speicher durchsucht. Die Funktion ARRAY_MIN verändert den Inhalt des Arrays nicht.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel: ARRAY_MIN(ADR(bigarray), SIZEOF(bigarray))

6.13. ARRAY_SDV

Type	Funktion : REAL
Input	PT : Pointer (Zeiger auf das Array)
	SIZE : UINT (Größe des Arrays)
Output	REAL (Standardabweichung des Arrays)



Die Funktion ARRAY_SDV ermittelt die Standardabweichung (Standard Deviation) eines beliebigen Arrays of REAL. Beim Aufruf wird der Funktion ein Pointer auf das Array und dessen Größe in Bytes übergeben. Unter CoDe-

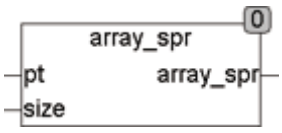
Sys lautet der Aufruf: ARRAY_SDV(ADR(Array), SIZEOF(Array)), wobei Array der Name des zu durchsuchenden Arrays ist. ADR() ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF() ist eine Standardfunktion, die die Größe des Arrays ermittelt. Um den Maximalwert zu ermitteln wird das durch den Pointer referenzierte Array direkt im Speicher durchsucht. Die Funktion ARRAY_SDV verändert den Inhalt des Arrays nicht.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel: ARRAY_SDV(ADR(bigarray), SIZEOF(bigarray))

6.14. ARRAY_SPR

Type	Funktion : REAL
Input	PT : Pointer (Zeiger auf das Array)
	SIZE : UINT (Größe des Arrays)
Output	REAL (Streuung des Arrays)



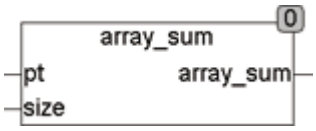
Die Funktion ARRAY_SPR ermittelt die Streuung eines beliebigen Arrays of REAL. Die Streuung ist der Maximalwert im Array minus dem Minimalwert des Arrays. Beim Aufruf wird der Funktion ein Pointer auf das Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: ARRAY_SPR(ADR(Array), SIZEOF(Array)), wobei Array der Name des zu durchsuchenden Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt. Um den Maximalwert zu ermitteln wird das durch den Pointer referenzierte Array direkt im Speicher durchsucht. Die Funktion ARRAY_SPR verändert den Inhalt des Arrays nicht.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel: ARRAY_SPR(ADR(bigarray), SIZEOF(bigarray)) =
Maximalwert(bigarray) - Minimalwert(bigarray)

6.15. ARRAY_SUM

Type	Funktion : REAL
Input	PT : Pointer (Zeiger auf das Array)
	SIZE : UINT (Größe des Arrays)
Output	REAL (Summe aller Werte des Arrays)



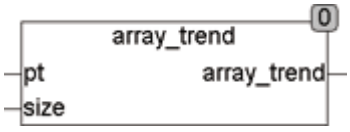
Die Funktion ARRAY_SUM ermittelt die Summe aller Werte eines beliebigen Arrays of REAL. Beim Aufruf wird der Funktion ein Pointer auf das Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: ARRAY_SUM(ADR(Array), SIZEOF(Array)) wobei Array der Name des zu durchsuchenden Arrays ist. ADR() ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF() ist eine Standardfunktion, die die Größe des Arrays ermittelt. Um den Maximalwert zu ermitteln wird das durch den Pointer referenzierte Array direkt im Speicher durchsucht. Die Funktion ARRAY_SUM verändert den Inhalt des Arrays nicht.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel: ARRAY_SUM(ADR(bigarray), SIZEOF(bigarray))

6.16. ARRAY_TREND

Type	Funktion : REAL
Input	PT : Pointer (Zeiger auf das Array)
	SIZE : UINT (Größe des Arrays)
Output	REAL (Trendentwicklung des Arrays)



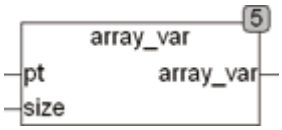
Die Funktion ARRAY_TREND ermittelt den Trend der Werte eines beliebigen Arrays of REAL. Beim Aufruf wird der Funktion ein Pointer auf das Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf:

ARRAY_TREND(ADR(Array), SIZEOF(Array)), wobei Array der Name des zu durchsuchenden Arrays ist. ADR() ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF() ist eine Standardfunktion, die die Größe des Arrays ermittelt. Um den Trend zu ermitteln wird das durch den Pointer referenzierte Array direkt im Speicher durchsucht. Die Funktion ARRAY_TREND verändert den Inhalt des Arrays nicht. Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen. Der Trend wird ermittelt, indem der Durchschnitt der unteren Hälfte der Werte des Arrays vom Durchschnitt der Werte der oberen Hälfte des Arrays abgezogen wird.

Beispiel: ARRAY_TREND(ADR(bigarray), SIZEOF(bigarray))

6.17. ARRAY_VAR

Type	Funktion : REAL
Input	PT : Pointer (Zeiger auf das Array)
	SIZE : UINT (Größe des Arrays)
Output	REAL (Varianz des Arrays)



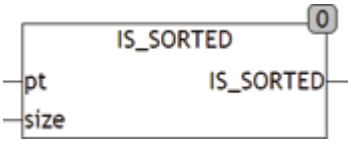
Die Funktion ARRAY_VAR ermittelt die Varianz eines beliebigen Arrays of REAL. Beim Aufruf wird der Funktion ein Pointer auf das Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: ARRAY_VAR(ADR(Array), SIZEOF(Array)) wobei Array der Name des zu durchsuchenden Arrays ist. ADR() ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF() ist eine Standardfunktion, die die Größe des Arrays ermittelt. Um den Maximalwert zu ermitteln wird das durch den Pointer referenzierte Array direkt im Speicher durchsucht. Die Funktion ARRAY_VAR verändert den Inhalt des Arrays nicht.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel: ARRAY_VAR(ADR(bigarray), SIZEOF(bigarray))

6.18. IS_SORTED

Type	Funktion : BOOL
Input	PT : Pointer (Zeiger auf das Array)
	SIZE : UINT (Größe des Arrays)
Output	BOOL (TRUE)



Die Funktion IS_SORTED prüft ob ein beliebiges Array of REAL in aufsteigender Reihenfolge sortiert ist. Beim Aufruf wird der Funktion ein Pointer auf das zu sortierende Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: _ARRAY_SORT(ADR(Array), SIZEOF(Array)), wobei Array der Name des zu sortierenden Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt.

Die Funktion liefert TRUE zurück wenn das Array in aufsteigender Reihenfolge sortiert ist.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel: IS_SORTED(ADR(bigarray), SIZEOF(bigarray))

7. Komplexe Mathematik

7.1. EINLEITUNG

Komplexe Zahlen werden mit dem definierten Typ COMPLEX abgebildet.

Der Typ COMPLEX setzt sich aus einem Realteil und einem Imaginärteil zusammen, beide Komponenten sind vom Typ REAL.

Die komplexe Zahl Z vom Typ COMPLEX besteht aus:

*.RE Realteil vom Typ REAL.

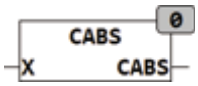
*.IM Imaginärteil vom Typ REAL.

7.2. CABS

Type Funktion : REAL

Input X : COMPLEX (Eingangswert)

Output REAL (Ergebnis)



CABS berechnet die Länge des Vektors einer komplexen Zahl. Der Absolutwert wird auch Modul oder Magnitude genannt.

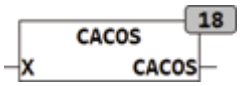
CABS = SQRT(X.RE² + X.IM²)

7.3. CACOS

Type Funktion : COMPLEX

Input X : COMPLEX (Eingangswert)

Output COMPLEX (Ergebnis)

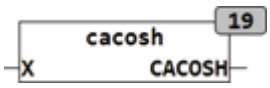


CACOS berechnet den Arcus Kosinus einer Komplexen Zahl.

Der Wertebereich des Ergebnisses liegt zwischen [0,π] für den Realteil und [-∞,+∞] für den Imaginärteil.

7.4. CACOSH

Type	Funktion : COMPLEX
Input	X : COMPLEX (Eingangswert)
Output	COMPLEX (Ergebnis)

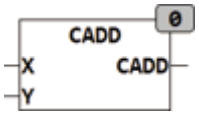


CACOSH berechnet den Arcus Kosinus Hyperbolicus einer Komplexen Zahl.

Der Wertebereich des Ergebnisses liegt zwischen [-π,+π] für den Imaginärteil.

7.5. CADD

Type	Funktion : COMPLEX
Input	X : COMPLEX (Eingangswert)
	Y : COMPLEX (Eingangswert)
Output	COMPLEX (Ergebnis)

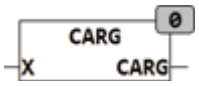


CADD addiert zwei Komplexe Zahlen X und Y.

7.6. CARG

Type	Funktion : REAL
Input	X : COMPLEX (Eingangswert)

Output REAL (Ergebnis)



CARG berechnet den Winkel einer Komplexen Zahl im Koordinatensystem.

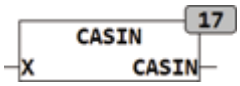
Der Wertebereich des Ergebnisses liegt zwischen [-π,+π].

7.7. CASIN

Type Funktion : COMPLEX

Input X : COMPLEX (Eingangswert)

Output COMPLEX (Ergebnis)



CASIN berechnet den Arcus Sinus einer Komplexen Zahl.

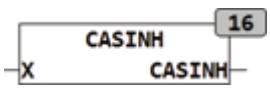
Der Wertebereich des Ergebnisses liegt zwischen [-π/2,+π/2] für den Realteil.

7.8. CASINH

Type Funktion : COMPLEX

Input X : COMPLEX (Eingangswert)

Output COMPLEX (Ergebnis)

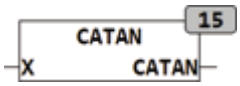


CASINH berechnet den Arcus Sinus Hyperbolicus einer Komplexen Zahl.

Der Wertebereich des Ergebnisses liegt zwischen [-π,+π] für den Imaginärteil.

7.9. CATAN

Type	Funktion : COMPLEX
Input	X : COMPLEX (Eingangswert)
Output	COMPLEX (Ergebnis)

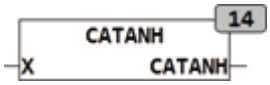


CATAN berechnet den Arcus Tangens einer Komplexen Zahl.

Der Wertebereich des Ergebnisses liegt zwischen [-π/2,+π/2] für den Realteil.

7.10. CATANH

Type	Funktion : COMPLEX
Input	X : COMPLEX (Eingangswert)
Output	COMPLEX (Ergebnis)

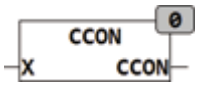


CATANH berechnet den Arcus Tangens Hyperbolicus einer Komplexen Zahl.

Der Wertebereich des Ergebnisses liegt zwischen [-π/2,+π/2] für den Imaginärteil.

7.11. CCON

Type	Funktion : COMPLEX
Input	X : COMPLEX (Eingangswert)
Output	COMPLEX (Ergebnis)



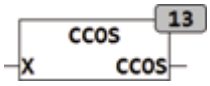
CCON berechnet die Konjugation einer komplexen Zahl.

CCON.RE = X.RE

CCON.IM = -X.IM

7.12. CCOS

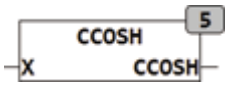
Type Funktion : COMPLEX
Input X : COMPLEX (Eingangswert)
Output COMPLEX (Ergebnis)



CCOS berechnet den Kosinus einer Komplexen Zahl.

7.13. CCOSH

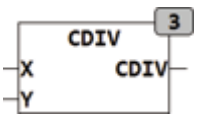
Type Funktion : COMPLEX
Input X : COMPLEX (Eingangswert)
Output COMPLEX (Ergebnis)



CCOSH berechnet den Kosinus Hyperbolicus einer Komplexen Zahl.

7.14. CDIV

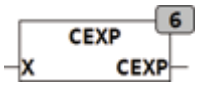
Type Funktion : COMPLEX
Input X : COMPLEX (Eingangswert)
Y : COMPLEX (Eingangswert)
Output COMPLEX (Ergebnis)



CDIV dividiert zwei komplexe Zahlen X / Y.

7.15. CEXP

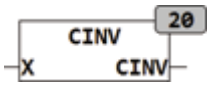
Type	Funktion : COMPLEX
Input	X : COMPLEX (Eingangswert)
Output	COMPLEX (Ergebnis)



CEXP berechnet den komplexen Exponent zur Basis E, CEXP = E^X.

7.16. CINV

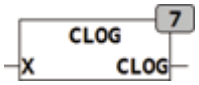
Type	Funktion : COMPLEX
Input	X : COMPLEX (Eingangswert)
Output	COMPLEX (Ergebnis)



CINV berechnet der Kehrwert einer Komplexen Zahl, CINV = 1/X

7.17. CLOG

Type	Funktion : COMPLEX
Input	X : COMPLEX (Eingangswert)
Output	COMPLEX(Ergebnis)

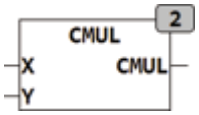


CLOG berechnet den natürlichen Logarithmus einer Komplexen Zahl zur Basis E.

CLOG(X) = LOG(e)(X).

7.18. CMUL

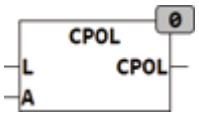
Type	Funktion : COMPLEX
Input	X : COMPLEX (Eingangswert)
	Y : COMPLEX (Eingangswert)
Output	COMPLEX (Ergebnis)



CMUL Multipliziert zwei Komplexe Zahlen X und Y.

7.19. CPOL

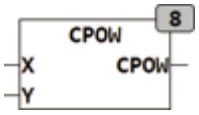
Type	Funktion : COMPLEX
Input	L : REAL (Länge oder Raduis)
	A : REAL (Winkelwert)
Output	COMPLEX (Ergebnis)



CPOL erzeugt eine Komplexe Zahl aus der Polarform. Die Eingangswerte L und A spezifizieren die Länge (Radius) und den Winkel.

7.20. CPOW

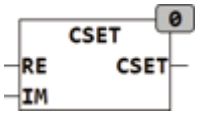
Type	Funktion : COMPLEX
Input	X : COMPLEX (Eingangswert)
	Y : COMPLEX (Eingangswert 2)
Output	COMPLEX (Ergebnis)



CPOW berechnet die Potenz zweier Komplexer Zahlen, CPOW = X^Y.

7.21. CSET

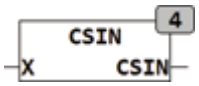
Type	Funktion : COMPLEX
Input	RE : COMPLEX (Realer Eingangswert)
	IM : REAL (Imaginärer Eingangswert)
Output	COMPLEX (Ergebnis)



CSET erzeugt aus den beiden Eingangskomponenten RE und IM eine komplexe Zahl vom Typ COMPLEX.

7.22. CSIN

Type	Funktion : COMPLEX
Input	X : COMPLEX (Eingangswert)
Output	COMPLEX (Ergebnis)

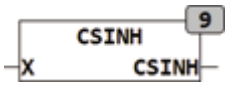


CSIN berechnet den Sinus einer Komplexen Zahl.

7.23. CSINH

Type	Funktion : COMPLEX
Input	X : COMPLEX (Eingangswert)

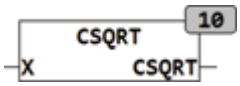
Output COMPLEX (Ergebnis)



CSINH berechnet den Sinus Hyperbolicus einer Komplexen Zahl.

7.24. CSQRT

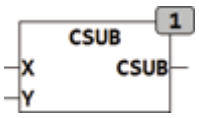
Type Funktion : REAL
Input X : COMPLEX (Eingangswert)
Output REAL (Ergebnis)



CSQRT berechnet die Quadratwurzel aus einer Komplexen Zahl.

7.25. CSUB

Type Funktion : COMPLEX
Input X : COMPLEX (Eingangswert)
Y : COMPLEX (Eingangswert)
Output COMPLEX (Ergebnis)

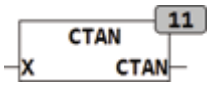


CSUB Subtrahiert zwei komplexe Zahlen, CSUB = X - Y.

7.26. CTAN

Type Funktion : COMPLEX

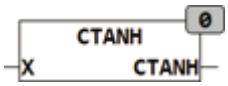
Input X : COMPLEX (Eingangswert)
Output COMPLEX (Ergebnis)



CTAN berechnet den Tangens einer Komplexen Zahl.

7.27. CTANH

Type Funktion : COMPLEX
Input X : COMPLEX (Eingangswert)
Output COMPLEX (Ergebnis)



CTANH berechnet den Tangens Hyperbolicus einer Komplexen Zahl.

8. Arithmetik doppelter Genauigkeit

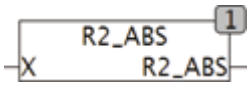
8.1. Einleitung

Gleitpunktzahlen werden im Format REAL abgespeichert. Ein gängiges Datenformat nach IEC754 benutzt dazu ein 24Bit breite Mantisse und einen 8Bit Exponenten. Daraus resultiert eine Genauigkeit von 7-8 Dezimalstellen. Normalerweise ist dies für Anwendungen in der Steuerungstechnik mehr als Ausreichend, kann aber in bestimmten Fällen zu einem Problem führen. Ein Typischer Fall der mit einfacher Genauigkeit nur unzureichend gelöst werden kann sind Verbrauchsmesser. Möchte man bis zu Mehreren Mwh (Megawattstunden) an Gesamtverbrauch Aufsummieren und dabei eine kleinste Leistung von 1mW (Milliwatt) im Abstand von 10ms Messen und berücksichtigen, so benötigt man eine Auflösung von 3.6 * 10^7 (entspricht 10MWs) und dabei würde man 1 * 10^-5 Ws aufaddieren wollen. Um dies zu bewerkstelligen benötigt man eine Auflösung von 12 Stellen.

Die von OSCAT implementierte Doppelte REAL Genauigkeit hat eine Auflösung von etwa 15 Stellen. Der hierfür Implementierte Datentyp REAL2 besteht aus R1 und RX, hierbei wird in RX der Wert mir den ersten 7-8 Stellen als Real gespeichert und der Rest in einem Weiteren REAL R1. Dieser Datentyp hat den Vorteil das keine Wandlung von REAL2 zu REAL notwendig, ist, vielmehr ist der Teil RX bereits der einfache REAL Wert.

8.2. R2_ABS

Type	Funktion : REAL2
Input	X : REAL2 (Eingangswert)
Output	REAL2 (Ergebnis mit Doppelter Genauigkeit)

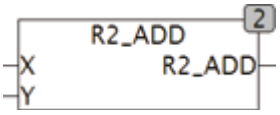


R2_ABS liefert den Absolutwert von X in doppelter Genauigkeit.

8.3. R2_ADD

Type	Funktion : REAL2

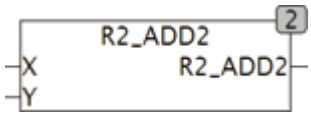
Input X : REAL2 (Eingangswert)
Y : REAL (zu Addierender Wert)
Output REAL2 (Ergebnis mit Doppelter Genauigkeit)



R2_ADD addiert zu einem Wert mit doppelter Genauigkeit X einen Wert einfacher Genauigkeit Y. Das Ergebnis hat wieder Doppelte Genauigkeit.

8.4. R2_ADD2

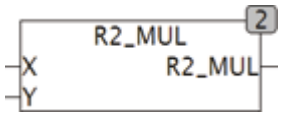
Type Funktion : REAL2
Input X : REAL2 (Eingangswert)
Y : REAL2 (zu Addierender Wert)
Output REAL2 (Ergebnis mit Doppelter Genauigkeit)



R2_ADD2 addiert zu einem Wert doppelter Genauigkeit X einen anderen Wert doppelter Genauigkeit Y. Das Ergebnis hat wieder doppelte Genauigkeit.

8.5. R2_MUL

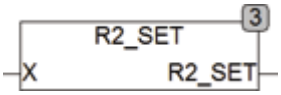
Type Funktion : REAL2
Input X : REAL2 (Eingangswert)
Y : REAL (Multiplikator)
Output REAL2 (Ergebnis mit Doppelter Genauigkeit)



R2_MUL Multipliziert einen Wert Doppelter Genauigkeit X mit einem Wert einfacher Genauigkeit Y. Das Ergebnis hat wieder Doppelte Genauigkeit.

8.6. R2_SET

Type	Funktion : REAL2
Input	X : REAL (Eingangswert)
Output	REAL2 (Ergebnis mit Doppelter Genauigkeit)

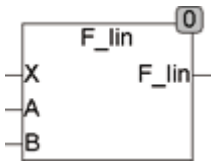


R2_SET setzt einen Wert Doppelter Genauigkeit auf den Eingangswert X mit einfacher Genauigkeit.

9. Arithmetische Funktionen

9.1. F_LIN

Type	Funktion : REAL
Input	X : REAL
	A : REAL
	B : REAL
Output	REAL (F_LIN = A*X + B)

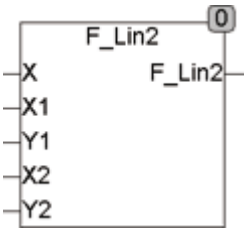


Die Funktion F_LIN Berechnet den Y-Wert einer linearen Gleichung.

F_LIN = A*X + B

9.2. F_LIN2

Type	Funktion : REAL
Input	X : REAL
	X1, Y1 : REAL (erste Koordinate)
	X2, Y2 : REAL (zweite Koordinate)
Output	REAL (Wert auf der Geraden, die durch die obigen 2 Punkte definiert ist.)

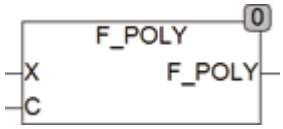


Die Funktion F_LIN2 Berechnet den Y-Wert einer linearen Gleichung.

Die Gerade ist dabei durch die Spezifikation zweier Koordinatenpunkte (X1,Y1; X2,Y2) definiert.

9.3. F_POLY

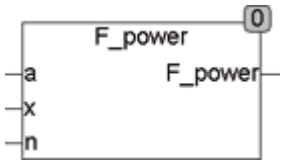
Type	Funktion : REAL
Input	X : REAL (Eingang)
	C : ARRAY[0..7] of REAL (Polynomkoeffizienten)
Output	REAL (Ergebnis der Polynomgleichung)



F_POLY errechnet ein Polynom 7ten Grades.
F_POLY = C[0] + C[1] * X^1 + C[2] * X^2 + ...C[7] * X^7

9.4. F_POWER

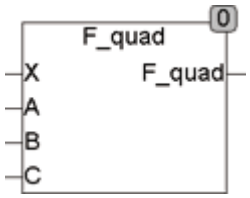
Type	Funktion : REAL
Input	A : REAL
	X : REAL
	N : REAL
Output	REAL (Ergebnis der Gleichung F_POWER = A * X^N)



F_Power berechnet die Potenzfunktion nach der Gleichung F_POWER = A * Xn.

9.5. F_QUAD

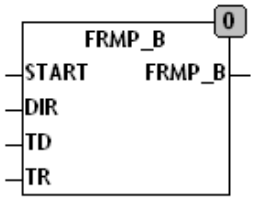
Type	Funktion : REAL
Input	X : REAL
	A, B, C : REAL
Output	REAL (F_QUAD = A * X² + B * X + C)



F_QUAD berechnet das Ergebnis einer quadratischen Gleichung nach der Formel f_QUAD = A * X² + B * X + C.

9.6. FRMP_B

Type	Funktion : BYTE
Input	START : BYTE (Startwert)
	DIR : BOOL (Richtung der Rampe)
	TD : TIME (Abgelaufene Zeit)
	TR : TIME (Gesamtzeit für Rampe)
Output	BYTE (Ausgang)



FRMP_B berechnet den Wert einer Rampe bei gegebenem Zeitablauf TD. Der Baustein stellt dabei sicher das kein Über- oder Unter-Lauf des Ausgangs stattfinden kann. Der Ausgangswert ist in allen Fällen auf 0 .. 255 begrenzt. TR gibt die Zeit für eine gesamte Rampe 0 .. 255 vor und TD ist die abgelaufene Zeit. Wenn DIR = TRUE wird eine steigende Rampe berechnet und wenn DIR = FALSE eine fallende Flanke. Mit den Startwert kann eine Flanke von einem beliebigen Startpunkt aus berechnet werden.

9.7. FT_AVG

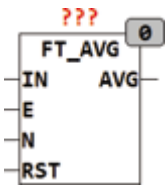
Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)

E : BOOL (Freigabe Eingang)

N : INT (Anzahl der Werte über die der Mittelwert gebildet wird)

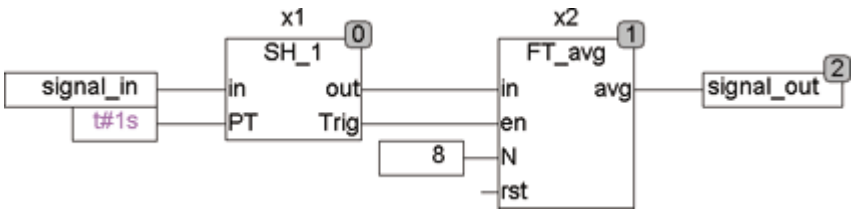
RST : BOOL (Reset Eingang)

Output REAL (Gleitender Mittelwert über die letzten N Werte)



Der Funktionsbaustein FT_AVG berechnet einen gleitenden Mittelwert jeweils über die letzten N Werte. Durch den Eingang RST können die gespeicherten Werte gelöscht werden. N ist definiert von 0 .. 32. N=0 bedeutet das Ausgangssignal = Eingangssignal ist. N=5 bildet den Mittelwert über die letzten 5 Werte. Der Mittelwert wird maximal über 32 Werte gebildet. Durch den Eingang E kann kontrolliert werden, wann der Eingang gelesen wird. Hierdurch kann auf einfache Weise ein Sample und Hold Baustein wie zum Beispiel SH_1 mit FT_AVG verknüpft werden. Beim ersten Aufruf von FT_AVG wird der Puffer mit dem Eingangssignal geladen, um zu vermeiden dass ein Ramp-up stattfindet.

Im folgenden Beispiel liest SH_1 einmal pro Sekunde den Eingangswert Signal_in und gibt diese Werte einmal pro Sekunde an FT_AVG weiter, welcher dann aus den letzten 8 Werten den Mittelwert bildet.



9.8. FT_MIN_MAX

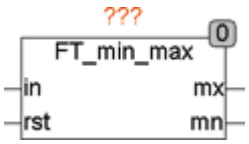
Type Funktionsbaustein

Input IN : REAL (Eingangssignal)

RST : BOOL (Reset Eingang)

Output MX : REAL (Maximalwert des Eingangssignal)

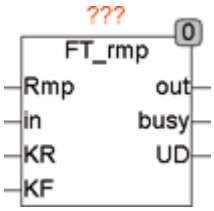
MN : REAL (Minimalwert des Eingangssignal)



FT_MIN_MAX Speichert den Minimal- und Maximalwert eines Eingangssignals IN und stellt diese beiden Werte an den Ausgängen MN und MX zur Verfügung bis er durch einen Reset wieder gelöscht wird. Ein Reset setzt MN und MX auf den zum Zeitpunkt des Resets anliegenden Eingangswert zurück.

9.9. FT_RMP

Type	Funktionsbaustein
Input	RMP : BOOL (Enable Signal)
	IN : REAL (Eingangssignal)
	KR : REAL (Geschwindigkeit des Anstiegs in 1 / Sekunden)
	KF : REAL (Geschwindigkeit des Abfalls in 1 / Sekunden)
Output	OUT : REAL (Ausgangssignal)
	BUSY : BOOL (Zeigt an ob Ausgang Steigt oder Fällt)
	UD : BOOL (TRUE, wenn Ausgang steigt und FALSE, wenn Ausgang fällt)

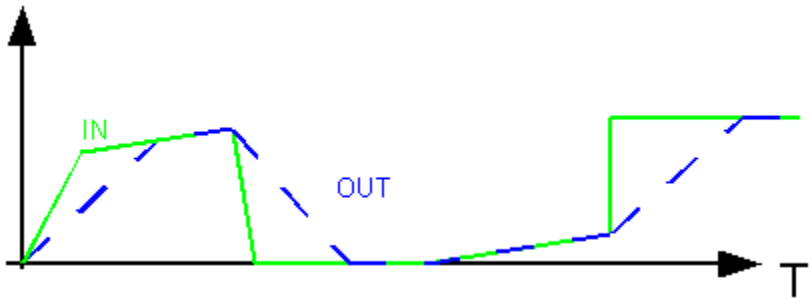


Der Ausgang OUT folgt dem Eingang mit einer linearen Rampe mit definierter Anstiegs- oder Abfall-Geschwindigkeit (KR und KF). K = 1 bedeutet, dass der Ausgang mit 1 Einheit pro Sekunde steigt oder fällt. Der K Faktor muss größer als 0 sein. Der Ausgang UD ist TRUE, wenn das Ausgangssignal steigt und FALSE, wenn es abfällt. Wenn der Ausgang den Eingangswert erreicht hat ist BUSY FALSE, ansonsten ist BUSY TRUE und zeigt an, dass eine steigende oder fallende Rampe aktiv ist.

Das Ausgangssignal folgt solange dem Eingangssignal wie die Anstiegs- oder Abfall-Geschwindigkeit des Eingangssignals kleiner ist, als die durch KR und KF definierte maximale Anstiegs- oder Abfall-Geschwindigkeit. Ver-

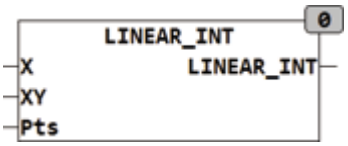
ändert sich das Eingangssignal schneller, so läuft der Ausgang mit der Geschwindigkeit KR oder KF dem Eingangssignal hinterher. Die Rampenerzeugung ist Zeitecht, was bedeutet, dass FT_RMP zu jeder Ausführung berechnet wo der Ausgang stehen sollte und diesen Wert auf den Ausgang legt. Die Ausgangsveränderung ist also abhängig von der Zykluszeit und erfolgt nicht in gleichen Schritten. Wird eine Rampe aus lauter gleichen Schritten benötigt, so stehen die Bausteine RMP_B und RMP_W zur Verfügung. Der Baustein ist nur dann aktiv wenn der Eingang RMP = TRUE ist.

Die folgende Grafik zeigt den Verlauf des Ausgangs in Abhängigkeit eines Eingangssignals:



9.10. LINEAR_INT

Type	Funktion
Input	X : REAL (Eingangssignal)
	XY : ARRAY[1..20,0..1] (Aufsteigend sortierte Wertepaare)
	PTS : INT (Anzahl der Wertepaare)
Output	REAL (Ausgangssignal)



LINEAR_INT ist ein linearer Interpolations Baustein. Eine beliebige Kennlinie wird mit maximal 20 Koordinatenwerten (X,Y) beschrieben und dadurch in bis zu 19 lineare Segmente zerlegt. Die Definition der Koordinatenwerte wird in einem Array übergeben welches die Kennlinie mit einzelnen X,Y Wertepaaren beschreibt. Die Wertepaare müssen aufsteigend nach dem X_Wert Sortiert sein. Wird ein X-Wert außerhalb des durch die Wertepaare beschriebenen Bereiches abgefragt, so wird das erste bezie-

hungsweise letzte lineare Segment extrapoliert und der entsprechende Wert Ausgegeben. Um die Anzahl der Definitionspunkte flexibel zu halten wird am Eingang PTS die Anzahl der Punkte vorgegeben. Die mögliche Punktezahl liegt im Bereich 3 bis 20, wobei jeder Einzelpunkt mit X- und Y-Wert dargestellt ist.

Beispiel:

```
VAR
        BEISPIEL : ARRAY[1..20,0..1] := -10,-0.53, 10,0.53, 100,88.3, 200,122.2;
END_VAR
```

für obige Definition ergeben sich folgende Ergebnisse:

LINEAR_INT(0, Beispiel, 4) = 0;

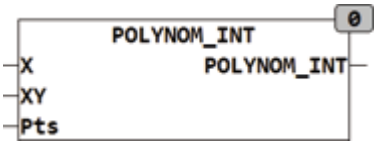
LINEAR_INT(30.0, Beispiel, 4) = 20.0344;

LINEAR_INT(66.41, Beispiel, 4) = 55.54229;

LINEAR_INT(800.0, Beispiel, 4) = 325.6;

9.11. POLYNOM_INT

Type	Funktion
Input	X : REAL (Eingangssignal)
	XY : ARRAY[1..5,0..1] (Aufsteigend sortierte Wertepaare)
	PTS : INT (Anzahl der Wertepaare)
Output	REAL (Ausgangssignal)



POLYNOM_INT interpoliert eine Anzahl von Wertepaaren mit einem Polynom N-fachen Grades. Die Anzahl der Wertepaare ist PTS und N entspricht der Anzahl der Wertepaare (PTS). Eine beliebige Kennlinie wird mit maximal 5 Koordinatenwerten (X,Y) beschrieben und intern mit einem Polynom beschrieben. Die Definition der Koordinatenwerte wird in einem Array übergeben welches die Kennlinie mit einzelnen X,Y Wertepaaren beschreibt. Die Wertepaare müssen aufsteigend nach dem X_Wert Sortiert sein. Wird ein X-Wert außerhalb des durch die Wertepaare beschriebenen Bereiches abgefragt, so wird dieser gemäß dem ermittelten Polygon berechnet, es ist hierbei aber zu beachten das durch ein Polynom höheren

Grades Schwingungen oberhalb und unterhalb des Definitionsbereiches auftreten können und berechnete Werte in diesem Bereich meist nicht sinnvoll sind. Vor der Anwendung einer Polynominterpolation sind unbedingt Grundlagen hierzu, zum Beispiel in Wikipedia, nachzulesen. Um die Anzahl der Definitionspunkte flexibel zu halten wird am Eingang PTS die Anzahl der Punkte vorgegeben. Die mögliche Punktezahl liegt im Bereich 3 bis 5, wobei jeder Einzelpunkt mit X- und Y-Wert dargestellt ist. Ein Polynominterpolation mit mehr als 5 Punkten führt zu erhöhter Schwingneigung und ist aus diesem Grunde abzulehnen.

Das folgende Beispiel zeigt die Definition für das Array XY und einige Werte:

```
VAR
    BEISPIEL : ARRAY[1..5,0..1] := -10,-0.53, 10,0.53, 100,88.3, 200,122.2;
END_VAR
```

für obige Definition ergeben sich folgende Ergebnisse:

POLYNOM_INT(0, Beispiel, 4) = -1.397069;

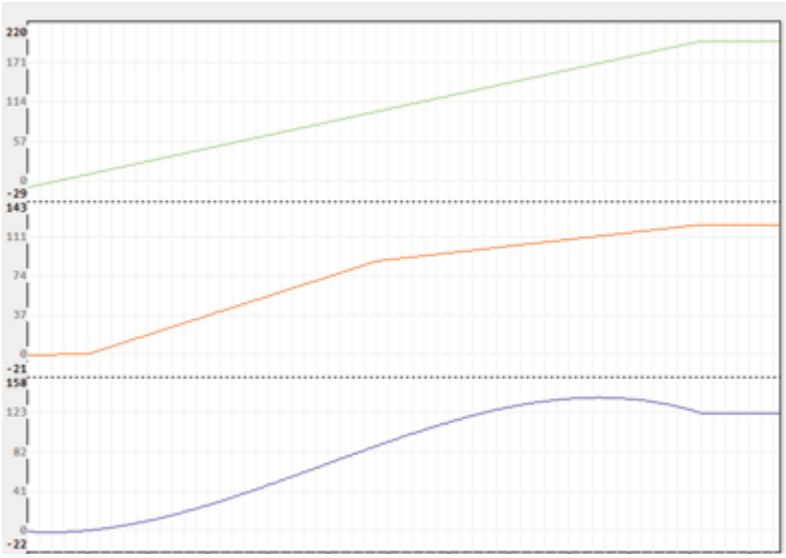
POLYNOM_INT(30.0, Beispiel, 4) = 11.4257;

POLYNOM_INT(66.41, Beispiel, 4) = 47.74527;

POLYNOM_INT(800.0, Beispiel, 4) = -19617.94;

Bei den Ergebnissen im Beispiel ist deutlich zu erkennen das der Wert von -19617.94 für den Eingang X = 800 keinen Sinn macht, da er außerhalb des definierten Bereichs von -10 bis +200 liegt.

Die folgende Trace Aufzeichnung zeigt den Verlauf des Ausgangs zum Eingang. Hierbei sind deutlich die Überschwinger des Polygons gegenüber einer linearen Interpolation zu erkennen. Grün = Eingang X, Rot = lineare Interpolation, Blau = Polynom Interpolation.


220
171
114
57
0
-29
143
111
74
37
0
-21
158
123
82
41
0
-22

10. Geometrische Funktionen

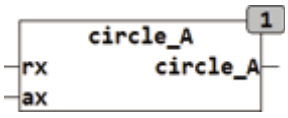
10.1. CIRCLE_A

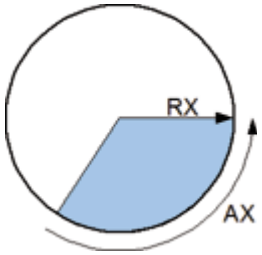
Type Funktion

Input RX : REAL (Kreisradius)

AX : REAL (Winkelsegment, 360 = gesamter Kreis)

Output REAL (Fläche des Kreissegments)





CIRCLE_A berechnet die Fläche eines Kreissegments mit den Winkel AX und dem Radius RX. Wird der Winkel AX = 360 gesetzt so wird die Kreisfläche berechnet.

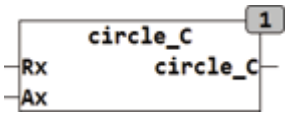
10.2. CIRCLE_C

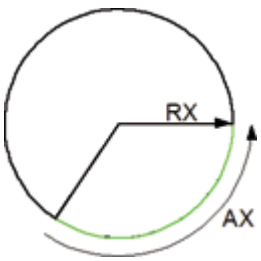
Type Funktion

Input RX : REAL (Kreisradius)

AX : REAL (Winkelsegment, 360 = gesamter Kreis)

Output REAL (Kreisbogenlänge bzw. Kreisumfang)





CIRCLE_C berechnet die Bogenlänge eines Kreisbogens mit den Winkel AX und dem Radius RX. Wird der Winkel AX = 360 gesetzt so wird der Kreisumfang berechnet.

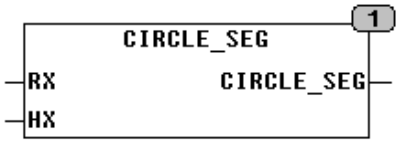
10.3. CIRCLE_SEG

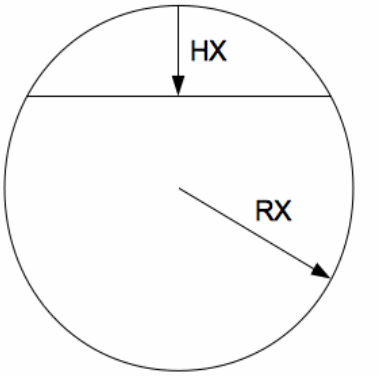
Type Funktion : REAL

Input RX : REAL (Kreisradius)

HX : REAL (Höhe der Sektantlinie)

Output Real : (Fläche des Segments)





CIRCLE_SEG berechnet die Fläche eines Kreissegments das durch eine Sektantlinie und den Kreis eingeschlossen ist.

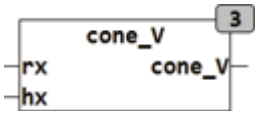
10.4. CONE_V

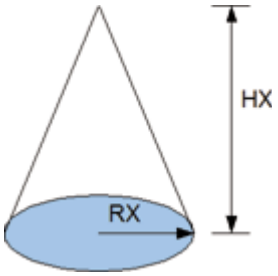
Type Funktion

Input RX : REAL (Kreisradius der Grundfläche)

HX : REAL (Höhe des Kegels)

Output REAL (Volumen des Kegels)



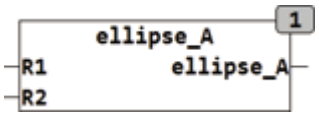


KONE_V berechnet das Volumen eines Kegels mit dem Radius RX und der Höhe HX.

10.5. ELLIPSE_A

Type Funktion

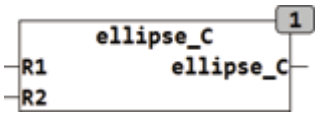
Input	R1 : REAL (Radius 1)
	R2 : REAL (Radius 2)
Output	REAL (Fläche der Ellipse)



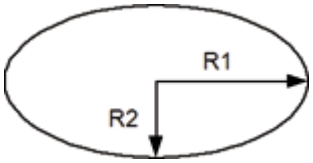
ELLIPSE_A berechnet die Fläche einer Ellipse die durch die Radien R1 und R2 definiert ist.

10.6. ELLIPSE_C

Type	Funktion
Input	R1 : REAL (Radius 1)
	R2 : REAL (Radius 2)
Output	REAL (Umfang der Ellipse)

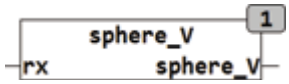


ELLIPSE_C berechnet den Umfang einer Ellipse die durch die Radien R1 und R2 definiert ist.



10.7. SPHERE_V

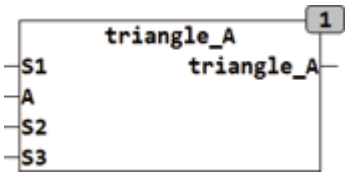
Type	Funktion
Input	RX : REAL (Radius)
Output	REAL (Volumen der Kugel)

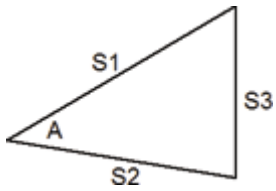


SPHERE_V berechnet das Volumen einer Kugel mit dem Radius RX.

10.8. TRIANGLE_A

Type	Funktion
Input	S1 : REAL (Seitenlänge 1)
	A : REAL (Winkel zwischen Seite 1 und Seite 2)
	S2 : REAL (Seitenlänge 2)
	S3 : REAL (Seitenlänge 3)
Output	REAL (Fläche des Dreiecks)





TRIANGLE_A berechnet die Fläche eines beliebigen Dreiecks. Das Dreieck kann wahlweise durch 2 Seiten und den durch die Seiten 1 und 2 aufgespannten Winkel (S1, S2 und A) definiert sein oder wenn A = 0 dann wird die Fläche aus den drei Seiten (S1, S2 und S3) berechnet.

11. Vektor Mathematik

11.1. Einleitung

Vektoren werden mit dem definierten Typ VEKTOR_3 abgebildet.

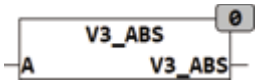
Der Typ VEKTOR_3 setzt sich aus 3 Komponenten X, Y und Z zusammen, alle Komponenten sind vom Typ REAL.

Der Vektor V vom Typ Vektor besteht aus:

V.X	X Komponente vom Typ REAL.
V.Y	Y Komponente vom Typ REAL.
V.Z	Z Komponente vom Typ REAL.

11.2. V3_ABS

Type	Funktion
Input	A : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)
Output	REAL (Absolute Länge des Vektors)

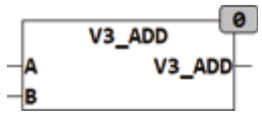


V3_ABS berechnet den Absolutwert (Länge) eines Vektors in einen dreidimensionalen Koordinatensystem.

V3_ABS(3,4,5) = 7.071...

11.3. V3_ADD

Type	Funktion
Input	A : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)
	B : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)
Output	VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)



V3_ADD addiert 2 dreidimensionale Vektoren.

V3_ADD([3,4,5],[1,2,3]) = (4,6,8)

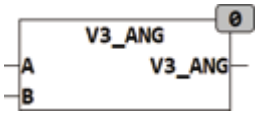
11.4. V3_ANG

Type Funktion

Input A : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)

B : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)

Output REAL (Winkel in Bogenmaß)



V3_ANG berechnet den Winkel zwischen 2 dreidimensionalen Vektoren

V3_ANG([1,0,0],[0,1,0]) = 1,57.. (π / 2)

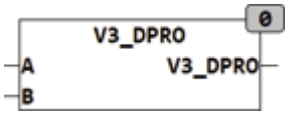
11.5. V3_DPRO

Type Funktion

Input A : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)

B : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)

Output REAL (Skalares Produkt)



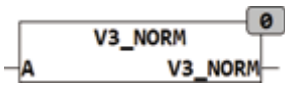
V3_DPRO berechnet das Skalare Produkt zweier dreidimensionaler Vektoren.

V3_DPRO([1,2,3],[3,1,2]) = 11

Das Skalare Produkt berechnet sich aus A.X*B.X + A.Y*B.Y + A.Z*B.Z

11.6. V3_NORM

Type	Funktion
Input	A : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)
Output	VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)

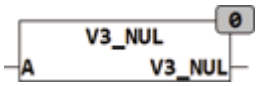


V3_NORM erzeugt aus einen beliebigen dreidimensionalen Vektor einen auf die Länge 1 Normierten Vektor mit gleicher Richtung. Ein Vektor mit der Länge 1 wird Einheitsvektor genannt

V3_NORM(3,0,0) = (1,0,0)

11.7. V3_NUL

Type	Funktion
Input	A : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)
Output	BOOL (TRUE wenn Vektor ein Nullvektor ist)

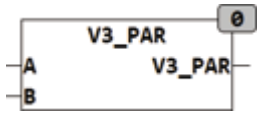


V3_NUL prüft ob der Vektor A ein Nullvektor ist. Ein Vektor ist dann ein Nullvektor wenn alle Komponenten (X, Y, Z) gleich Null sind.

V3_NUL(0,0,0) = TRUE

11.8. V3_PAR

Type	Funktion
Input	A : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)
	B : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)
Output	BOOL (TRUE wenn die beiden Vektoren parallel sind)



V3_PAR wird dann TRUE wenn die beiden Vektoren A und B Parallel sind. Ein Nullvektor ist Parallel zu jedem Vektor da er keine Richtung hat. Zwei Vektoren A und B die in Gegensätzliche Richtung zeigen sind Parallel.

V3_PAR([1,1,1],[2,2,2]) = TRUE

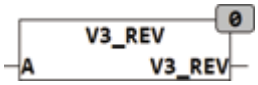
V3_PAR([1,1,1],[-1,-1,-1]) = TRUE

V3_PAR([1,2,3],[0,0,0]) = TRUE

V3_PAR([1,2,3],[1,0,0]) = FALSE

11.9. V3_REV

Type	Funktion
Input	A : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)
Output	VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)

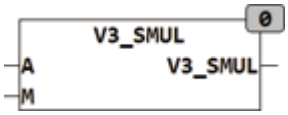


V3_REV erzeugt einen Vektor mit dem identischen Betrag von A aber mit entgegengesetzter Richtung. A - V3_REV(A) = 0.

V3_REV(1,2,3) = (-1,-2,-3)

11.10. V3_SMUL

Type	Funktion
Input	A : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)
	M : REAL (Skalarer Multiplikator)
Output	VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)

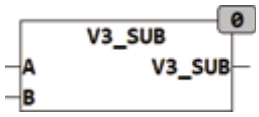


V3_SMUL Multipliziert einen dreidimensionalen Vektor A mit dem Skalar M.

V3_SMUL([1,2,3],10) = (10,20,30)

11.11. V3_SUB

Type	Funktion
Input	A : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)
	B : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)
Output	VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)



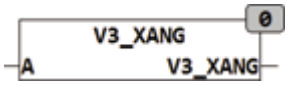
V3_SUB Subtrahiert den Vektor B von A

V3_SUB([3,3,3],[1,2,3]) = (2,1,0)

V3_SUB([1,2,3],[1,-2,-3]) = (0,4,6)

11.12. V3_XANG

Type	Funktion
Input	A : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)
Output	REAL (Winkel zur X-Achse)

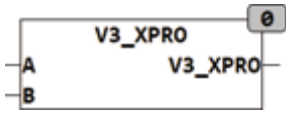


V3_XANG berechnet den Winkel zwischen Der X-Achse des Koordinaten-systems und einem dreidimensionalen Vektor A in Bogenmaß

V3_XANG(1,2,3) = 1.300..

11.13. V3_XPRO

Type	Funktion
Input	A : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)
	B : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)
Output	VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)

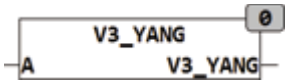


V3_XPRO berechnet das Kreuzprodukt zweier dreidimensionaler Vektoren A und B

V3_XPRO([1,2,3],[2,1,2]) = (1,4,-3)

11.14. V3_YANG

Type	Funktion
Input	A : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)
Output	REAL (Winkel zur Y-Achse)

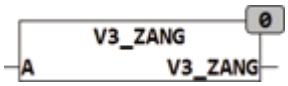


V3_YANG berechnet den Winkel zwischen Der Y-Achse des Koordinatensystems und einem dreidimensionalen Vektor A in Bogenmaß

V3_YANG(1,2,3) = 1.006..

11.15. V3_ZANG

Type	Funktion
Input	A : VECTOR_3 (Vektor mit den Koordinaten X, Y, Z)
Output	REAL (Winkel zur Z-Achse)



V3_ZANG berechnet den Winkel zwischen Der Z-Achse des Koordinatensystems und einem dreidimensionalen Vektor A in Bogenmaß

V3_ZANG(1,2,3) = 0.640..

12. Time & Date

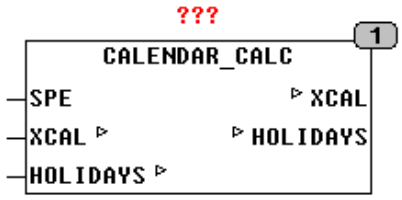
12.1. Einleitung

Die Zeit und Datumsfunktionen der OSCAT Bibliothek sind abhängig vom Zielsystem so Implementiert das sie die Unterschiede in der Implementierung der Datums / Zeittypen der Einzelsysteme berücksichtigen. Zum Beispiel ist auf CoDeSys Systemen das UNIX TIMEDATE Implementiert, was bedeutet das der Datentyp TD in Sekunden ab 1.1.1970-00:00 als 32 Bit Wert abgebildet wird. In STEP7 hingegen wird der Datentyp TD in Sekunden ab 1.1.1990 abgebildet. Der Wertebereich der Zeit / Datumsfunktionen ist bei CoDeSys Systemen 1.1.1970 – 31.12.2099 und bei STEP7 Systemen 1.1.1990 – 31.12.2099. Die Begrenzung des Wertebereichs auf das Jahr 2099 liegt vor allem an der Tatsache das im Jahr 2100 kein Schaltjahr sein wird.

Weiterhin entsprechen die Datums und Zeitfunktionen der ISO8601 (internationaler Standard für numerische Datumsfunktionen). Hier ist zum Beispiel die Implementierung der Wochentage mit 1 = Montag und 7 = Sonntag vorgeschrieben.

12.2. CALENDAR_CALC

Type	Funktionsbaustein
Input	SPE : BOOL (TRUE wird die aktuelle Sonnenposition berechnet)
I/O	XCAL : CALENDAR (externe Variable)
HOLIDAYS	HOLIDAY_DATA (Feiertagsliste)

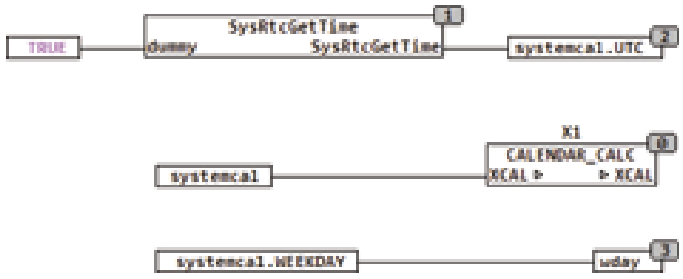


CALENDAR_CALC berechnet vollautomatisch alle Werte in einer Struktur vom Typ CALENDAR ausgehend vom Wert UTC in der Struktur. XCAL ist ein Pointer auf eine externe oder globale Variable vom Typ CALENDAR. CALENDAR_CALC kann so über die Struktur XCAL im gesamten Programm Kalenderwerte zur Verfügung stellen. CALENDAR_CALC ermittelt bei jeder Veränderung des Wertes UTC in XCAL automatisch alle anderen Werte in der

Struktur. Alleine der Wert UTC in der Struktur muss von einem RTC Baustein gespeist werden. Die Definition des strukturierten Datentyps CALENDAR finden Sie im Kapitel Datenstrukturen. Die fortlaufende Berechnung der Sonnenposition kann eine SPS ohne FPU stark belasten, deshalb wird der laufende Sonnenstand nur alle 25 Sekunde berechnet wenn SPE = TRUE ist. Dies entspricht einer Genauigkeit von 0,1 Grad was für Normale Anwendungen völlig Ausreichend ist. Bleibt SPE = FALSE wird die Sonnenposition nicht berechnet. Durch ein externes Array HOLIDAYS vom Typ HOLIDAY_DATA kann der Anwender gezielt Feiertage nach seinen Bedürfnissen spezifizieren, nähere Hinweise für die Definition von Feiertagen finden sie im Kapitel Datenstrukturen.

Falls Mehrere Strukturen vom Typ CALENDAR benötigt werden (zum Beispiel für UTC und oder verschiedene Lokalzeiten) dann können entsprechend mehrere Bausteine CALENDAR_CALC mit verschiedenen Strukturen vom TYP CALENDAR eingesetzt werden.

Das folgende Beispiel zeigt wie der Baustein SYSRTCGETTIME die RTC der CPU ausliest und die aktuelle Zeit in SYSTEMCAL.UTC schreibt. CALENDAR_CALC prüft bei jedem Zyklus ob sich der Wert in .UTC verändert hat und wenn ja ermittelt es die anderen Werte der Struktur automatisch. Der Ausgang WDAY zeigt wie man aus der Struktur Daten zur Weiterverarbeitung ausliest. CALENDAR_CALC berücksichtigt die Setup Daten aus der Datenstruktur (OFFSET, DST_EN, LONGITUDE, LATITUDE).



Im externen Array HOLIDAYS können bis zu 30 Feiertage definiert werden. Beispiele hierfür finden Sie bei der Beschreibung des Datentyps HOLIDAY_DATA. Dieses ARRAY of HOLIDAY_DATA muss außerhalb des Bausteins definiert werden und als Variable mit den Feiertagsdaten vorbelegt werden.

12.3. DATE_ADD

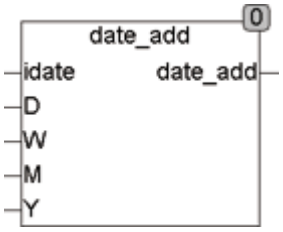
Type	Funktion
Input	IDATE : DATE (Eingangsdatum)
	D : INT (zu addierende Tage)

W : INT (zu addierende Wochen)

M : INT (zu addierende Monate)

Y : INT (zu addierende Jahre)

Output DATE (Ergebnisdatum)



Die Funktion DATE_ADD addiert Tage, Wochen, Monate, und Jahre zu einem Datum hinzu. Es werden erst die angegebenen Tage und Wochen addiert, dann die Monate und zuletzt die Jahre.

Die Eingangswerte können sowohl positiv, wie auch negativ sein. Es kann also auch von einem Datum subtrahiert werden.

Vor allem bei negativen Eingangswerten ist zu beachten das das Datum beim addieren von negativen Werten wie zum Beispiel -3000 Tage nicht unter den 1.1.1970 läuft, dies würde einen Überlauf des Datentyps DATE zur folge haben und undefinierte Werte ergeben.

Beispiel: DATE_ADD(1.1.2007,3,1,-1,-2) = 11.12.2005

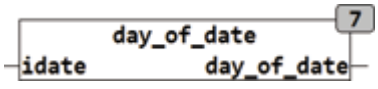
addiert 3 Tage und 1 Woche hinzu und zieht dann 1 Monat und 2 Jahre ab.

12.4. DAY_OF_DATE

Type Funktion : DINT

Input IDATE : DATE (Eingangsdatum)

Output DINT (Tag im Monat des Eingangsdatums)



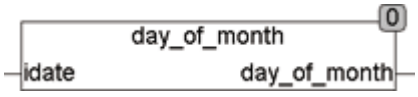
Die Funktion DAY_OF_DATE berechnet den Tag seit dem 1.1.1970. Das Ergebnis der Funktion ist vom Typ DINT weil der gesamte DATE Range 49710 Tage umfasst.

12.5. DAY_OF_MONTH

Type Funktion : INT

Input IDATE : DATE (Eingangsdatum)

Output INT (Tag im Monat des Eingangsdatums)



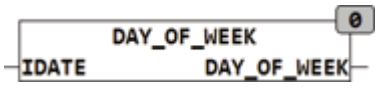
Die Funktion DAY_OF_MONTH berechnet den Tag des Monats aus dem Eingangsdatum IDATE.

12.6. DAY_OF_WEEK

Type Funktion : INT

Input IDATE : DATE (Eingangsdatum)

Output INT (Monat im Jahr des Eingangsdatums)



Die Funktion DAY_OF_WEEK berechnet den Wochentag aus dem Eingangsdatum IDATE.

Montag = 1... Sonntag = 7. Die Berechnung erfolgt gemäß ISO8601.

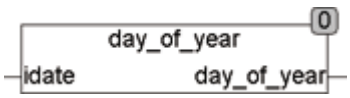
Beispiel: DAY_OF_WEEK(D#2007-1-8) = 1

12.7. DAY_OF_YEAR

Type Funktion : INT

Input IDATE : DATE (Eingangsdatum)

Output INT (Tag im Jahr des Eingangsdatums)



Die Funktion DAY_OF_YEAR berechnet den Tag des Jahres aus dem Eingangsdatum IDATE. Schaltjahre werden entsprechend dem gregorianischen Kalender berücksichtigt. Die Funktion ist definiert für die Jahre 1970 – 2099.

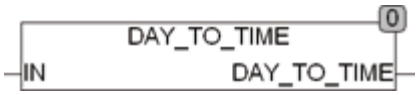
Beispiel: DAY_OF_YEAR(31.12.2007) = 365
DAY_OF_YEAR(31.12.2008) = 366

12.8. DAY_TO_TIME

Type Funktion : TIME

Input IN : REAL (Anzahl Tage mit Nachkommastellen)

Output TIME (TIME)



Die Funktion DAY_TO_TIME berechnet einen Zeitwert (TIME) aus dem Eingangswert in Tagen als REAL.

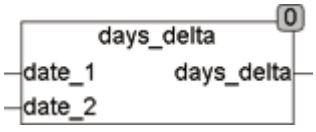
Beispiel: DAY_TO_TIME(1.1) = T#26h24m

12.9. DAYS_DELTA

Type Funktion : DINT

Input DATE_1 : DATE (Datum1)
DATE_2 : DATE (Datum2)

Output DINT (Differenz der beiden Eingangsdatums in Tagen)



Die Funktion DAYS_DELTA berechnet die Differenz zweier Daten in Tagen.

Beispiel: DAYS_DELTA(10.1.2007, 1.1.2007) = -9
DAYS_DELTA(1.1.2007, 10.1.2007) = 9

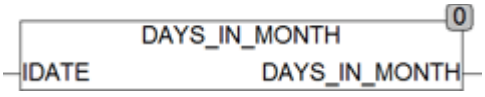
Das Ergebnis der Funktion ist vom Typ DINT weil der gesamte DATE Range 49710 Tage umfasst.

12.10. DAYS_IN_MONTH

Type Funktion : INT

Input IDATE : DATE (aktuelles Datum)

Output INT (Anzahl der Tage des aktuellen Monats)



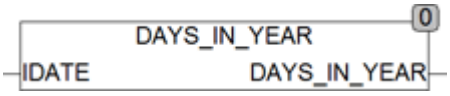
Die Funktion DAYS_IN_MONTH berechnet die Anzahl der Tage des aktuellen Monats.

12.11. DAYS_IN_YEAR

Type Funktion : INT

Input IDATE : DATE (aktuelles Datum)

Output INT (Anzahl der Tage des aktuellen Jahres)



Die Funktion DAYS_IN_YEAR berechnet die Anzahl der Tage des aktuellen Jahres.

12.12. DCF77

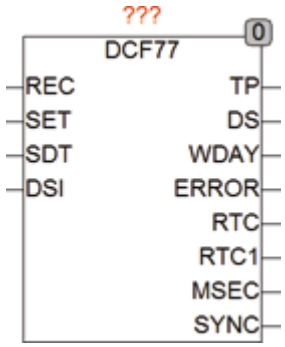
Type Funktionsbaustein

Input REC : BOOL (Eingang für den DCF77 Empfänger)

SET : BOOL (Asynchroner SET Eingang)

SDT : DT (Anfangswert für RTC)

	DSI : BOOL (Sommerzeit Eingang)
Output	TP : BOOL (Puls zum Setzen von nachgeschalteten Uhren)
	DS : BOOL (TRUE, wenn Sommerzeit herrscht)
	WDAY : INT (Wochentag)
	ERROR : BOOL (TRUE, wenn REC kein Signal liefert)
	RTC : DT (Synchronisierte Weltzeit UTC)
	RTC1 : DT (Synchronisierte Lokalzeit)
	MSEC : INT (Millisekunden von RTC und RTC1)
	SYNC : BOOL (TRUE, wenn RTC mit DCF synchron ist)
Setup	SYNC_TIMEOUT : TIME (Default = T#2m)
	TIME_OFFSET : INT (Zeit Offset für RTC1, Default = 1 Stunde)
	DST_EN : BOOL (Sommerzeit für RTC1, Default = TRUE)

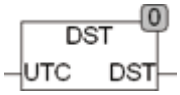


Die Funktion DCF77 dekodiert das Serielle Signal eines DCF77 Empfängers und steuert 2 interne Uhren RTC und RTC1, oder über den Ausgang TP auch externe (nachgeschaltete) Uhren. Ein Ausgang DS wird TRUE, wenn Sommerzeit herrscht. Der Ausgang WDAY gibt den Wochentag an (1 = Montag). Der Ausgang ERROR wird TRUE, wenn kein gültiges Signal Empfangen wird. Die Internen Uhren laufen aber trotzdem weiter, wenn Sie bereits synchronisiert sind. Ein weiterer Ausgang SYNC zeigt an dass die internen Uhren mit DCF77 synchronisiert sind und wird FALSE, wenn sie länger als durch die Setup-Variable SYNC_TIMEOUT festgelegte Zeit nicht mehr synchronisiert wurden. Die internen Uhren laufen in jedem Fall mit der Genauigkeit des SPS-Timers weiter. Durch einen Doppelklick auf das Symbol im CFC Editor können weitere Setup-Variablem definiert werden. Hierbei legt SYNC_TIMEOUT fest, nach welcher Zeit das Ausgangssignal SYNC, FALSE wird, wenn die internen Uhren RTC und RTC1 nicht mehr durch DCF77 synchronisiert wurden. Die Variable TIME_OFFSET legt die Zeitdifferenz der Lokalzeit (RTC1) von der UTC fest. Default ist 1 Stunde für MEZ (Mitteleuropäische Zeit). Die Variable TIME_OFFSET ist vom Typ INTEGER damit auch Zeitzonen mit negativem Offset (Westlich von Greenwich) möglich sind.

Durch DST_EN wird festgelegt, ob RTC1 automatisch auf Sommerzeit schalten soll oder nicht. Der Ausgang MSEC erweitert die auf RTC und RTC1 zur Verfügung gestellte Zeit um Millisekunden. Der Eingang SDT dient dazu die internen Uhren RTC und RTC1 auf einen definierten Anfangswert zu setzen damit sofort nach den Start eine gültige Uhrzeit zur Verfügung steht. Während des ersten Zyklus wird Datum und Zeit von SDT nach RTC kopiert, und läuft ab dem ersten Zyklus. Falls nötig kann mit den asynchronen Setz Eingang SET die interne UHR jederzeit neu gestellt werden. Sie wird aber nach einem Zyklus wieder von einem Gültigen DCF77 Signal überschrieben, ausser der Eingang SET bleibt auf TRUE. Sobald ein gültiges DCF77 Signal dekodiert wurde wird RTC und RTC1 auf die entsprechende genauere DCF77 Zeit synchronisiert. Am Eingang SDT kann zum Beispiel die Uhrzeit aus der in der SPS enthaltenen Hardware Uhr verwendet werden, es muss aber sichergestellt werden das DCF77 erst dann aufgerufen wird wenn bereits eine gültige Uhrzeit an SDT anliegt, den DCF77 liest diesen Wert nur ein einziges mal im ersten Zyklus ein, oder immer dann wenn der Eingang SET auf TRUE steht.

12.13. DST

Type	Funktion : BOOL
Input	UTC : DATE_TIME (Weltzeit)
Output	BOOL (TRUE, wenn Sommerzeit)



Die Funktion DST überprüft, ob im Augenblick Sommerzeit herrscht, oder nicht. Sie kann dazu benutzt werden eine vorhandene nicht-Sommerzeit fähige Uhr sekundengenau auf Sommer- und Winterzeit umzustellen.

Die Funktion DST schaltet am letzten Sonntag des März um 01:00 UTC (02:00 MEZ) auf Sommerzeit (03:00 MESZ) und am letzten Sonntag des Oktober um 01:00 UTC (03:00 MESZ) auf 02:00 MEZ zurück. Der Ausgang von DST ist dann TRUE, wenn Sommerzeit herrscht.

Die Sommerzeit wird aufgrund von UTC (Weltzeit) berechnet. Eine Berechnung von Ortszeit nach Sommerzeit ist generell nicht möglich weil im letzten Sonntag des Oktobers die Stunde von 02:00 - 03:00 MEZ beziehungsweise MESZ doppelt existiert. Die Sommerzeit wird in allen Ländern der EU seit 1992 zur selben Sekunde nach Weltzeit umgestellt. In Mitteleuropa um 02:00, in England um 01:00 und in Griechenland um 04:00. Durch die Berechnung mithilfe der Weltzeit wird die Sommerzeit für alle Europäischen Zeitzonen richtig berechnet.

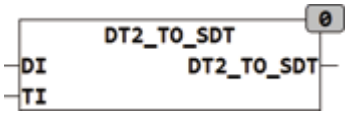
12.14. DT2_TO_SDT

Type Funktion : SDT

Input DI : DATE (Datum)

TI : TOD (Tageszeit)

Output SDT (Strukturierter Datums Zeit Wert vom Typ SDT)



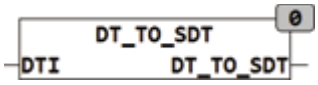
DT2_TO_SDT wandelt ein Datum und eine Tageszeit um in eine strukturierte Datum Tageszeit vom Typ SDT.

12.15. DT_TO_SDT

Type Funktion : SDT

Input DTI : DT (Datum Zeit Wert)

Output SDT (Strukturierter Datums Zeit Wert vom Typ SDT)



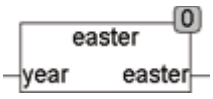
DT_TO_SDT wandelt ein Datum Zeitwert um in eine strukturierte Datum Tageszeit vom Typ SDT.

12.16. EASTER

Type Funktion : DATE

Input YEAR : INT (Jahr)

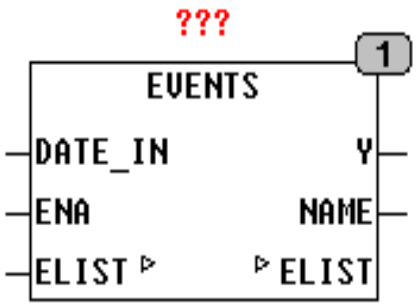
Output DATE (Datum des Ostersonntag für das angegebene Jahr)



Die Funktion EASTER berechnet für ein gegebenes Jahr das Datum des Ostersonntags. Die meisten kirchlichen Feiertage haben einen festen Abstand von Ostern, sodass für den Fall, dass Ostern für ein Jahr bekannt ist, diese Feiertage auch einfach ermittelt werden können. EASTER wird auch im Modul HOLIDAY verwendet um Feiertage zu berechnen.

12.17. EVENTS

Type	Funktionsbaustein
Input	DATE_IN : DATE (Eingangsdatum)
	ENA : BOOL (Enable Eingang)
I/O	ELIST : ARRAY[0..49] of HOLIDAY_DATA
Output	Y : BOOL (TRUE, wenn DATE_IN ein Event ist)
	Name : STRING(30) (Name des heutigen Events)



Der Baustein EVENTS zeigt am Ausgang Y mit TRUE besondere Tage an und liefert auch den Namen des entsprechenden Events am Ausgang NAME. EVENTS kann zusätzlich zu einzelnen Tagen auch Events über mehrere Tage berücksichtigen. Im Array ELIST werden Name, Datum und Dauer der Events festgelegt.

Im externen Array ELIST können nach folgendem Muster bis zu 50 solcher Events definiert werden.

*.NAME : STRING(30)	legt den Namen des Events fest
*.DAY : SINT	Monatstag des Events
*.MONTH : SINT	Monat des Events
*.USE : SINT	Dauer des Events in Tagen

Beispiele:

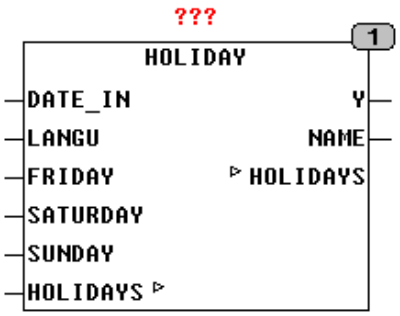
(NAME := 'Gründungstag', DAY := 13, MONTH := 7, USE := 1) festes Event „Gründungstag“ am 13. Juli für einen Tag.

(NAME := 'Gründungstag', DAY := 13, MONTH := 7, USE := 0) Event an einem festen Datum USE=0 bedeutet es ist nicht aktiv.

(NAME := 'Betriebsurlaub', DAY := 1, MONTH := 8, USE := 31) legt ein Event mit einer Dauer von 31 Tagen fest.

12.18. HOLIDAY

Type	Funktionsbaustein
Input	DATE_IN : DATE (Eingangsdatum)
	LANGU : INT (gewünschte Sprache)
	FRIDAY : BOOL (Y wird TRUE an Freitagen wenn TRUE)
	SATURDAY : BOOL (Y wird TRUE an Samstagen wenn TRUE)
	SUNDAY : BOOL (Y wird TRUE an Sonntagen wenn TRUE)
I/O	HOLIDAYS : ARRAY[0..29] of HOLIDAY_DATA
Output	Y : BOOL (TRUE, wenn DATE_IN ein Feiertag ist)
	Name : STRING(30) (Name des heutigen Feiertags)



Der Baustein HOLIDAY zeigt am Ausgang Y mit TRUE Feiertage an und liefert auch den Namen des aktuellen Feiertags am Ausgang NAME. HOLIDAY kann zusätzlich zu Feiergagen auch an den Wochentagen Freitag, Samstag oder Sonntag aktiv werden und am Ausgang Y TRUE liefern, abhängig davon ob die Eingänge FRIDAY, SATURDAY oder SUNDAY auf TRUE gesetzt sind. Im Array HOLIDAYS werden Name und Datum von Feiertagen definiert und sind auch dort für andere Länder universell anpassbar. Feiertage können als festes Datum, mit einem Abstand von Ostern oder Wochentag vor einem festen Datum definiert werden. Der Eingang LANGU wählt die entsprechende Sprache aus den Setup Daten aus damit die Ausgaben für Freitag, Samstag und Sonntag sprachspezifisch angepasst werden können.

Die Sprachen sind unter Globale Konstanten im Abschnitt "LANGUAGE SETUP" vordefiniert und können dort erweitert oder angepasst werden.

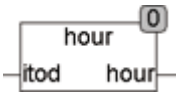
Im externen Array HOLIDAYS können bis zu 30 Feiertage definiert werden. Beispiele hierfür finden Sie bei der Beschreibung des Datentyps HOLIDAY_DATA.

12.19. HOUR

Type Funktion : INT

Input ITOD : TIMEOFDAY (Tageszeit)

Output INT (aktuelle Stunde)



Die Funktion HOUR extrahiert die aktuelle Stunde aus der Tageszeit.

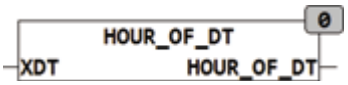
Beispiel: HOUR(22:55:13) = 22

12.20. HOUR_OF_DT

Type Funktion : INT

Input XDT : DATETIME (Eingangswert)

Output INT (aktuelle Stunde)



HOUR_OF_DT extrahiert die momentane Stunde aus einem DT Wert.

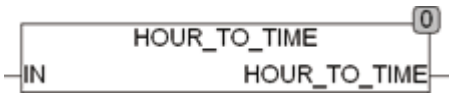
HOUR_OF_DT(DT#2008-6-6-10:22:20) = 10

12.21. HOUR_TO_TIME

Type Funktion : TIME

Input IN : REAL (Anzahl Stunden mit Nachkommastellen)

Output TIME (TIME)



Die Funktion HOUR_TO_TIME berechnet einen Zeitwert (TIME) aus dem Eingangswert in Stunden als REAL.

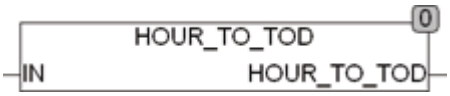
Beispiel: HOUR_TO_TIME(1.1) = T#1h6m

12.22. HOUR_TO_TOD

Type Funktion : TIME

Input IN : REAL (Anzahl Stunden mit Nachkommastellen)

Output TIME (Tageszeit)



Die Funktion HOUR_TO_TOD berechnet eine Tageszeit (TIMEOFDAY) aus dem Eingangswert in Stunden als REAL.

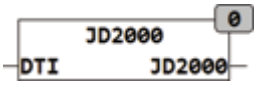
Beispiel: HOUR_TO_TOD(12.1) = 12:06:00

12.23. JD2000

Type Funktion : REAL

Input DTI : DT (Gregorianisches Datum)

Output REAL (astronomischer, julianischer Tag ab dem 1.1.2000 12:00)



JD2000 berechnet das astronomische Julianische Datum seit dem 1. Januar 2000 12:00 (dem Standardäquinoktium).

Das Julianische Datum gibt die Zeit in Tagen seit dem 1. Januar 4713 12:00 vor Chr. als Gleitkommazahl an. Der 1. Januar 2000 00:00 entspricht dem Julianischen Datum 2451544,5. Da ein Datum wie der 1. Januar 2000 be-

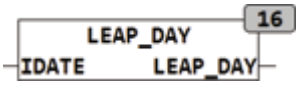
reits die Auflösungsgrenze eines REAL mit ca. 7 Stellen überschreiten würde kann das Julianische Datum nicht sinnvoll mit dem Datentyp REAL dargestellt werden. Die Funktion JD2000 zählt die Julianischen Tage seit dem 1.1.2000 12:00 Mittags und kann so ein aktuelle Datum sinnvoll im Datentyp REAL darstellen.

12.24. LEAP_DAY

Type Funktion : BOOL

Input IDATE : DATE (Eingangsdatum)

Output BOOL (TRUE, wenn der aktuelle Tag ein 29. Februar ist)



Die Funktion LEAP_DAY testet ob das Eingangsdatum ein Schalttag beziehungsweise ein 29. Februar ist. Der Test hat Gültigkeit für den Zeitrum 1970 - 2099. Im Jahr 2100 wird ein Schaltjahr angezeigt obwohl dies keines ist. Da aber der Wertebereich des Datums nach IEC61131-3 nur bis zum Jahr 2106 reicht wird auf diese Korrektur verzichtet.

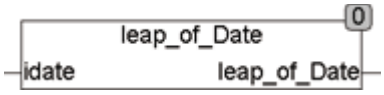
Beispiel: LEAP_DAY(D#2004-02-29) = TRUE

12.25. LEAP_OF_DATE

Type Funktion : BOOL

Input IDATE : DATE (Eingangsdatum)

Output BOOL (TRUE, wenn IDATE in einem Schaltjahr liegt)



Die Funktion LEAP_OF_DATE testet, ob das Eingangsdatum in einem Schaltjahr liegt. Die Funktion berechnet, ob ein Datum innerhalb eines Schaltjahres liegt und gibt gegebenenfalls TRUE aus. Der Test hat Gültigkeit für den Zeitrum 1970 - 2099. im Jahr 2100 wird ein Schaltjahr angezeigt obwohl dies keines ist. Da aber der Wertebereich des Datums nach IEC61131-3 nur bis zum Jahr 2106 reicht wird auf diese Korrektur verzichtet.

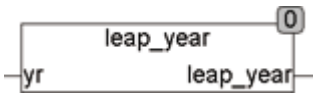
Beispiel: LEAP_OF_YEAR(D#2004-01-12) = TRUE

12.26. LEAP_YEAR

Type Funktion : BOOL

Input YR : INT (Jahreszahl)

Output BOOL (TRUE, wenn das angegebene Jahr ein Schaltjahr ist)



Die Funktion LEAP_YEAR testet, ob das Eingangsjahr ein Schaltjahr ist und gibt gegebenenfalls TRUE aus. Der Test hat Gültigkeit für den Zeitrum 1970 - 2099. im Jahr 2100 wird ein Schaltjahr angezeigt obwohl dies keines ist. Da aber der Wertebereich des Datums nach IEC61131-3 nur bis zum Jahr 2106 reicht wird auf diese Korrektur verzichtet.

12.27. LTIME_TO_UTC

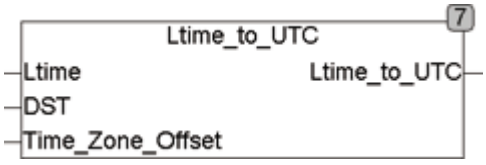
Type Funktion : DATE_TIME

Input LTIME : DATE_TIME (Lokalzeit)

DST : BOOL (TRUE, wenn Sommerzeit herrscht)

TIME_ZONE_OFFSET : INT (Zeitdifferenz zur Weltzeit in Min.)

Output DATE_TIME (UTC, Weltzeit)

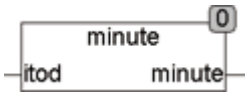


LTIME_TO_UTC errechnet UTC (Weltzeit) von einer vorgegebenen Lokalzeit. Die Weltzeit wird errechnet indem TIME_ZONE_OFFSET von der Lokalzeit LTIME subtrahiert wird. Wenn die Sommerzeit aktiv ist (DST = TRUE), wird eine weitere Stunde von LTIME abgezogen.

Anmerkung: Die Sommerzeit ist nicht in allen Ländern identisch geregelt. Die Funktion geht davon aus das bei Sommerzeit zusätzlich zum Offset eine weitere Stunde addiert wird.

12.28. MINUTE

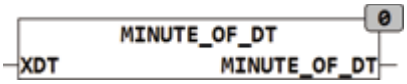
Type Funktion : INT
Input ITOD : TIMEOFDAY (Tageszeit)
Output INT (aktuelle Minute)



Die Funktion MINUTE extrahiert die aktuelle Minute aus der Tageszeit.
Beispiel: MINUTE(22:55:13) = 55

12.29. MINUTE_OF_DT

Type Funktion : INT
Input XDT : DATETIME (Eingangswert)
Output INT (aktuelle Minute)



MINUTE_OF_DT extrahiert die momentane Minute aus einem DT Wert.
MINUTE_OF_DT(DT#2008-6-6-10:22:20) = 22

12.30. MINUTE_TO_TIME

Type Funktion : TIME
Input IN : REAL (Anzahl Minuten mit Nachkommastellen)
Output TIME (TIME)

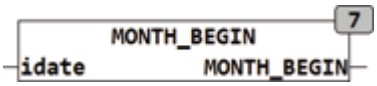
Die Funktion MINUTE_TO_TIME berechnet einen Zeitwert (TIME) aus dem Eingangswert in Minuten als REAL.
Beispiel: MINUTE_TO_TIME(122.5) = T#2h2m30s

12.31. MONTH_BEGIN

Type Funktion : DATE

Input IDATE : DATE (aktuelles Datum)

Output DATE (Datum des 1. Tages des aktuellen Monats)



MONTH_BEGIN berechnet das Datum des 1. Tages des aktuellen Monats und aktuellen Jahres.

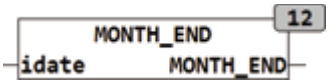
MONTH_BEGIN(D#2008-2-13) = D#2008-2-1

12.32. MONTH_END

Type Funktion : DATE

Input IDATE : DATE (aktuelles Datum)

Output DATE (Datum des letzten Tages des aktuellen Monats)



MONTH_END berechnet das Datum des letzten Tages des aktuellen Monats und aktuellen Jahres.

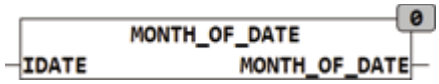
MONTH_END(D#2008-2-13) = D#2008-2-29

12.33. MONTH_OF_DATE

Type Funktion : INT

Input IDATE : DATE (Eingangsdatum)

Output INT (Monat im Jahr des Eingangsdatums)

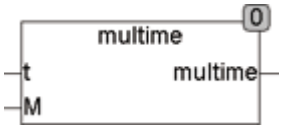


Die Funktion MONTH berechnet den Monat des Jahres aus dem Eingangsdatum IDATE.

Beispiel: MONTH_OF_DATE(D#2007-12-31) = 12
MONTH_OF_DATE(D#2006-1-1) = 1

12.34. MULTIME

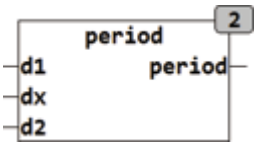
Type Funktion : TIME
Input T : TIME (Eingangszeit)
M : REAL (Multiplikator)
Output TIME (Ergebnis Eingangszeit Multipliziert mit M)



Die Funktion MULTIME Multipliziert einen Zeitwert mit einem Multiplikator.
Beispiel: MULTIME(T#1h10m, 2.5) = T#2h55m

12.35. PERIOD

Type Funktion : BOOL
Input D1 : DATE (Perioden Beginn)
DX : DATE (zu testendes Datum)
D2 : DATE (Perioden Ende)
Output BOOL (TRUE wenn DX innerhalb der Periode D1 .. D2 liegt)



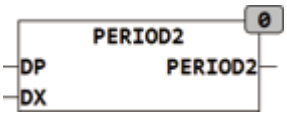
Die Funktion PERIOD prüft, ob ein Eingangsdatum DX größer gleich D1 und kleiner gleich D2 ist. Wenn das Datum DX im Zeitraum zwischen D1 und D2 (D1 und D2 eingeschlossen) liegt wird der Ausgang der Funktion TRUE gesetzt. PERIOD ignoriert dabei die Jahreszahlen in den Datumsangaben D1, D2 und DX. Die Prüfung erfolgt nur auf Monate und Tage, sodass diese Funktion für jedes Jahr funktioniert. Die zu Prüfende Periode kann auch über den 31. Dezember hinaus erstrecken, also zum Beispiel vom 1.9. – 15.3. Eine typische Anwendung ist die Prüfung, ob eine Heizperiode vor-

liegt. Damit PERIOD richtig rechnet dürfen die beiden Datumsangaben D1 und D2 nicht in einem Schaltjahr liegen. Es kann zum Beispiel immer das Jahr 2001 oder auch jedes beliebige andere Jahr das kein Schaltjahr ist verwendet werden.

PERIOD(1.10.2001, 12.11.2007, 31.3.2001) ergibt TRUE, weil das zu prüfende Datum innerhalb der Periode vom 1.10 - 31.3. liegt.

12.36. PERIOD2

Type	Funktion : BOOL
Input	DP : ARRAY[0..3,0..1] of DATE (Perioden)
	DX : DATE (zu testendes Datum)
Output	BOOL (TRUE wenn DX innerhalb einer der Perioden liegt)



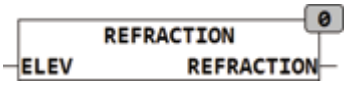
PERIOD2 prüft ob das Datum DX innerhalb einer von 4 spezifizierten Perioden liegt. Die Perioden werden in einem ARRAY[0..3,0..1] of DATE spezifiziert. Im Gegensatz zur Funktion PERIOD prüft PERIOD2 auch das Jahr. Die Perioden werden im ARRAY DP spezifiziert, wobei DP[N,0] das Anfangsdatum der Periode N DP[N,1] das Enddatum der Periode N ist.

Die Funktion Prüft nach der Formel: DX >= DP[N,0] AND DX <= DP[N,1]. Es wird Dabei jeweils N=0 bis 3 geprüft. Wenn DX in eine der 4 Perioden fällt wird der Ausgang auf TRUE gesetzt.

Die einzelnen Perioden Müssen nicht sortiert vorliegen. PERIOD2 kann benutzt werden um Ferien oder Urlaubszeiten zu definieren. PERIOD2 prüft nicht wiederkehrende Perioden, wobei die Funktion PERIOD jährlich wiederkehrende Perioden Prüft.

12.37. REFRACTION

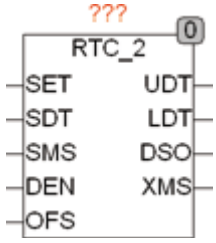
Type	Funktion : REAL
Input	ELEV : REAL (Elevation in Grad über Horizont)
Output	REAL (Refraktion in Grad)



REFRACTION berechnet die atmosphärische Brechung für außerhalb der Atmosphäre befindliche Himmelskörper. Ein Himmelskörper erscheint durch die Lichtbrechung in der Atmosphäre um die Refraktion höher über dem Horizont als er tatsächlich ist. Die Refraktion beträgt 0 am Zenith (um 12:00 Mittags) und nimmt nahe des Horizonts stark zu. Bei 0° (am Horizont) beträgt die Refraktion -0,59° und 10° über dem Horizont beträgt Sie 0,09°. Die Refraktion wird benötigt um berechnete Umlaufbahnen von Himmelskörpern oder auch Satelliten zu korrigieren so dass Sie mit der Beobachtung übereinstimmen. Der Baustein berechnet einen Mittelwert für einen Luftdruck von 1010mBar und 10°C. Wenn die Sonne tatsächlich bei 0° also exakt am Horizont steht erscheint sie wegen der Refraktion bei 0,59 ° über dem Horizont. Der Sichtbare Sonnenstand ist der tatsächliche (astronomische) Sonnenstand H + die Refraktion. Die Refraktion wird auch für Winkel unter dem Horizont (ELEV < - 2° berechnet, so daß unter dem Horizont immer die Refraktion zum astronomischen Winkel hinzuaddiert wird, damit z.B. der Abstand zum Sonnenaufgang jederzeit richtig errechnet werden kann. Für astronomische Winkel < -1.9 ° bleibt die Refraktion konstant bei 0.744 Grad.

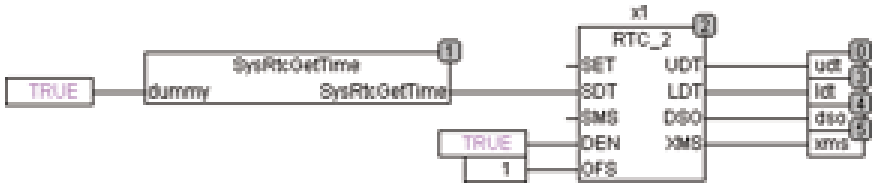
12.38. RTC_2

Type	Funktionsbaustein
Input	SET : BOOL (Set Eingang)
	SDT : DT (Set Datum und Zeit)
	SMS : INT (Set Millisekunden)
	DEN : BOOL (Automatische Sommerzeitumstellung Ein)
	OFS : INT (Offset der Lokalzeit in Minuten von UTC)
Output	UDT : DT (Datums und Zeit Ausgang für Weltzeit)
	LDT : DT (Lokalzeit)
	DSO : BOOL (Sommerzeit aktiv)
	XMS : INT (Millisekunden)



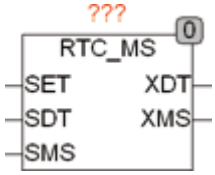
RTC_2 ist ein Uhrenbaustein der UTC und Lokale Zeit an den Ausgängen UDT und LDT zur Verfügung stellt. Die Uhrzeit wird automatisch beim ersten Start und immer dann wenn SET auf TRUE ist auf den Wert von SDT und SMS gesetzt. Wenn SET = FALSE läuft die Zeit selbständig weiter und liefert am Ausgang UDT das aktuelle Datum und die Uhrzeit für Weltzeit (UTC), sowie am Ausgang LDT die aktuelle Lokalzeit. Der Ausgang LDT entspricht UDT + OFS + Sommerzeit wenn diese aktuell ist. Die Sommerzeit wird wenn DEN TRUE ist automatisch am letzten Sonntag des März um 01:00 UTC (02:00 MEZ) auf Sommerzeit (03:00 MESZ) und am letzten Sonntag des Oktober um 01:00 UTC (03:00 MESZ) auf 02:00 MEZ zurück gestellt. Der Ausgang DSO ist dann TRUE, wenn Sommerzeit herrscht. Wenn DEN FALSE ist wird keine Sommerzeitumstellung vorgenommen. Die Genauigkeit der Uhr hängt vom Millisekunden Timer der SPS ab. Der Eingang OFS spezifiziert den Zeitversatz von LDT zu UDT, für MEZ ist dieser Wert 1 Stunde. OFS wird als INT in Minuten spezifiziert damit auch ein negativer Offset möglich ist. Für MEZ (Mitteleuropäische Zeit wird ein Offset von 60 Minuten eingestellt. RTC_2 übernimmt beim Power Up automatisch die an SDT anliegende Startzeit und Datum. Der Ausgang XMS stellt die Millisekunden zur Verfügung und zählt in jeder Sekunde von 0 – 999.

Im folgenden Beispiel wird beim Start die Systemzeit übernommen.



12.39. RTC_MS

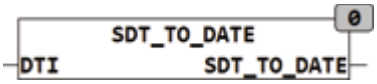
Type	Funktionsbaustein
Input	SET : BOOL (Set Eingang)
	SDT : DT (Set Datum und Zeit)
	SMS : INT (Set Millisekunden)
Output	XDT : DT (Datums und Zeit Ausgang)
	XMS : INT (Millisekunden Ausgang)



RTC_MS ist ein Uhrenbaustein mit einer Auflösung von Millisekunden und Datum. Die Uhrzeit wird automatisch beim ersten Start und immer dann wenn SET auf TRUE ist auf den Wert von SDT und SMS gesetzt. Wenn SET = FALSE läuft die Zeit selbständig weiter und liefert am Ausgang XDT das aktuelle Datum und die Uhrzeit, sowie am Ausgang XMS die Millisekunden. Der Ausgang XMS zählt in jeder Sekunde von 0 - 999 und beginnt mit der nächsten Sekunde wieder bei 0. Die Genauigkeit der Uhr hängt vom Millisekunden Timer der SPS ab.

12.40. SDT_TO_DATE

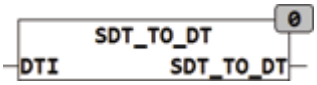
Type	Funktion : DATE
Input	DTI : SDT (Eingangswert als strukturierter Datums / Zeitwert)
Output	DATE (Datumswert)



SDT_TO_DATE erzeugt einen Datumswert aus einem strukturierten Datums- Zeit-Wert

12.41. SDT_TO_DT

Type	Funktion : DT
Input	DTI : SDT (Eingangswert als strukturierter Datums / Zeitwert)
Output	DT (Datums- Zeit-wert)

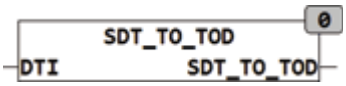


SDT_TO_DT erzeugt einen Datums- Zeit-wert aus einem strukturierten Datums- Zeit-Wert

12.42. SDT_TO_TOD

Type	Funktion : TOD
Input	DTI : SDT (Eingangswert als strukturierter Datums / Zeitwert)

Output TOD (Tageszeit)



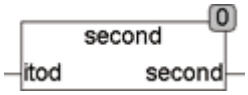
SDT_TO_TOD erzeugt eine Tageszeit aus einem strukturierten Datums-Zeit-Wert.

12.43. SECOND

Type Funktion : REAL

Input ITOD : TOD (Tageszeit)

Output REAL (Sekunden und Millisekunden der Tageszeit)



Die Funktion SECOND extrahiert den Sekundenanteil aus der Tageszeit

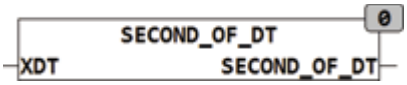
Beispiel: SECOND(22:10:12.331) = 12.331

12.44. SECOND_OF_DT

Type Funktion : INT

Input XDT : DATETIME (Eingangswert)

Output INT (aktuelle Sekunde)



SECOND_OF_DT extrahiert die momentane Sekunde aus einem DT Wert.

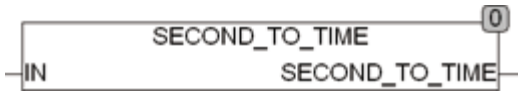
SECOND_OF_DT(DT#2008-6-6-10:22:20) = 20

12.45. SECOND_TO_TIME

Type Funktion : TIME

Input IN : REAL (Anzahl Sekunden mit Nachkommastellen)

Output TIME (TIME)



Die Funktion SECOND_TO_TIME berechnet einen Zeitwert (TIME) aus dem Eingangswert in Sekunden als REAL.

Beispiel: SECOND_TO_TIME(63.123) = T#1m3s123ms

12.46. SET_DATE

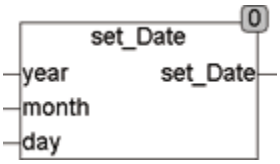
Type Funktion : DATE

Input YEAR : INT (Jahreszahl)

MONTH : INT (Monat)

DAY : INT (Tag)

Output DATE (Zusammengesetztes Datum)



Die Funktion SET_DATE berechnet einen Datum (DATE) aus den Eingangswerten Tag, Monat und Jahr. SET_DATE überprüft dabei nicht die Gültigkeit eines Datums. Zum Beispiel darf auch der 30. Februar gesetzt werden was natürlich den 1.3. oder bei einem Schaltjahr dem 2. März ergibt. SET_DATE kann deshalb auch benutzt werden um einen beliebigen Tag im Jahr zu erzeugen. Dies kann eine durchaus Sinnvolle Anwendung sein. In diesem Fall darf der Monat auch 0 betragen. Eine ungültige Monatsangabe ergibt immer ein Datum in Bezug auf den Januar. Ein ungültiger Monat (Monat < 1 oder Monat > 12) wird immer als Januar interpretiert.

Beispiel: SET_DATE(2007,1,365) = 31.12.2007

Beispiel: SET_DATE(2007, 1, 22) = 22.1.2007

Der Wertebereich der Funktion Eingänge ist:

YEAR {1970 .. 2099}

MONTH {1..12}

DAY {INT}

Wertebereich des Ausgangs: {1.1.1970 .. 31.12.2099}

12.47. SET_DT

Type Funktion : DATE_TIME

Input YEAR : INT (Jahreszahl)

MONTH : INT (Monat)

DAY : INT (Tag)

HOUR : INT (Stunde)

MINUTE : INT (Minute)

SECOND : INT (Sekunden)

Output DATE_TIME (Zusammengesetztes Zeitdatum)

Die Funktion SET_DT berechnet einen Zeit-Datumswert (DATE_TIME) aus den Eingangswerten Tag, Monat, Jahr, Stunde, Minute und Sekunden.

Beispiel: Set_DT(2007, 1, 22, 13, 10, 22) = DT#2007-1-22-13:10:22

12.48. SET_TOD

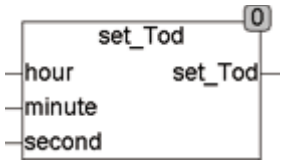
Type Funktion : TOD

Input HOUR : INT (Stunde)

MINUTE : INT (Minute)

SECOND : REAL (Sekunden und Millisekunden)

Output TOD (Ausgangswert Tageszeit)

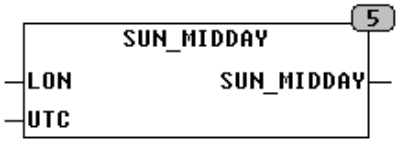


Die Funktion SET_TOD berechnet eine Tageszeit (TOD) aus den Eingangswerten Stunde, Minute und Sekunden.

Beispiel: Set_TOD(13, 10, 22.33) = 13:10:22.330

12.49. SUN_MIDDAY

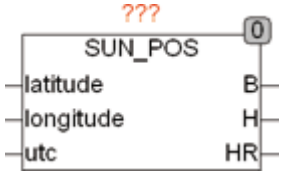
Type	Funktion
Input	LON : REAL (Längengrad des Bezugsortes)
	UTC : DATE (Weltzeit)
Output	TOD (Tageszeit wenn Sonne exakt im Süden steht)



Die Funktion SUN_MIDDAY berechnet abhängig vom Tagesdatum zu welcher Tageszeit die Sonne exakt im Süden steht. Die Berechnung erfolgt in UTC (Weltzeit).

12.50. SUN_POS

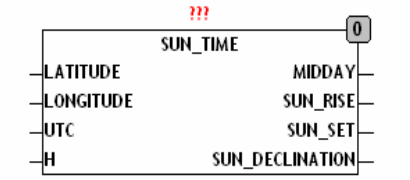
Type	Funktionsbaustein
Input	LATITUDE : REAL (Breitengrad des Bezugsortes)
	LONGITUDE : REAL (Längengrad des Bezugsortes)
	UTC : DATE_TIME (Weltzeit)
Output	B : REAL (Azimut in Grad von Nord)
	H : REAL (Astronomische Sonnenhöhe)
	HR : REAL (Sonnenhöhe in Grad über Horizont mit Refraktion)



SUN_POS berechnet die Position der Sonne (B, H) zur aktuellen Zeit. Die Zeit wird als Weltzeit (UTC) angegeben. Eine eventuell vorliegende Lokalzeit muss vorher in UTC umgerechnet werden. Beim Sonnenstand HR ist die atmosphärische Refraktion für 1010mbar und 10°C bereits berücksichtigt. Die Genauigkeit ist besser als 0,1 Grad für den Zeitraum von 2000 bis 2050. Mögliche Anwendungen von SUN_POS sind die Nachführung von Solarpanels oder eine vom Sonnenstand abhängige Nachführung der Lamellen von Jalousien. SUN_POS ist ein aufwendiger Algorithmus der aber exakte Werte liefert. Um die Belastung einer SPS so gering wie möglich zu halten kann die Berechnung zum Beispiel nur alle 10 Sekunden ausgeführt werden, was einer Ungenauigkeit von 0,04 Grad entspricht. Der Ausgang B gibt den Sonnenwinkel in Grad von Norden an (Süden = 180 °). H ist der Astronomische Winkel über dem Horizont (am Horizont = 0°). HR ist der Sonnenstand über dem Horizont der um die atmosphärische Brechung (Refraktion) korrigiert ist. Ein Beobachter auf der Erdoberfläche sieht die Sonne auf einer um die Refraktion angehobene Position über dem Horizont, was dazu führt das die Sonne bereits scheint obwohl sie noch leicht unter dem Horizont ist.

12.51. SUN_TIME

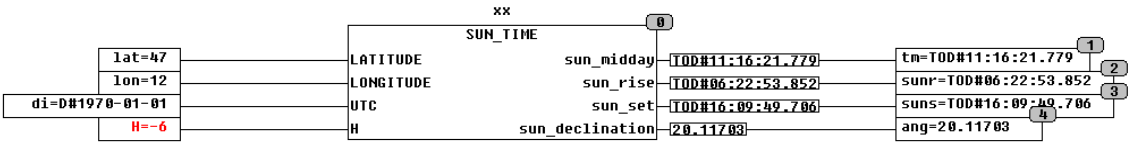
Type	Funktionsbaustein
Input	LATITUDE : REAL (Breitengrad des Bezugsortes)
	LONGITUDE : REAL (Längengrad des Bezugsortes)
	UTC : DATE (Weltzeit)
	H : Real (Winkel über dem Horizont in Grad)
Output	MIDDAY : TOD (Sonnenstand exakt Süden)
	SUN_RISE : TOD (Zeit des Sonnenaufgangs)
	SUN_SET : TOD (Zeit des Sonnenuntergangs)
	SUN_DECLINATION : REAL (Höhe bei Sonnenstand Süd)



Der Funktionsbaustein SUN_TIME ist ein Astrotimer. Er berechnet Sonnenaufgang und Sonnenuntergang für einen beliebigen Tag, definiert durch den Eingang UTC. Außer Sonnen Auf- und Untergang wird auch die Zeit des Sonnenazimut (Tageshöchststand im Süden) und der

Sonnenwinkel über dem Horizont im Azimut berechnet. Damit SUN_TIME unabhängig vom Einsatzort funktioniert werden alle Zeiten in UTC (Weltzeit) berechnet und können bei Bedarf wieder in Lokalzeit umgerechnet werden. Zusätzlich zu den Zeiten für Sonnenaufgang und Sonnenuntergang berechnet der Baustein auch noch den Winkel der Sonneneinstrahlung über dem Horizont SUN_DECLINATION. SUN_TIME benutzt einen aufwendigen Algorithmus, um die Belastung einer SPS so gering wie möglich zu halten sollten die Werte mit SUN_TIME nur einmal pro Tag errechnet werden. SUN_TIME wird für die Steuerung von Jalousien benutzt, um sie kurz vor Sonnenaufgang hochzuziehen, damit man im Schlafzimmer die Dämmerung genießen kann. Weitere Anwendungen sind im Gartenbau um Bewässerung anhand des Sonnen auf- und Untergangs zu steuern oder auch zum Nachführen von Solarpanels. Weitere Berechnungen des Sonnenstandes stellt der Baustein SUN_POS zur Verfügung. SUN_TIME stellt nur in Breitengraden zwischen 65° Süd und 65° Nord zur Verfügung. Der Ausgang MIDDAY zeit an zu welcher Zeit die Sonne genau im Süden steht und SUN_DECLINATION gibt dabei den Winkel über dem Horizont in Grad an.

Beispiel für den 1.1.1970, 12° Ost und 47° Nord:



Die Zeiten für Sonnen-Aufgang und - Untergang sind in UTC (Weltzeit), der höchste Sonnenstand wird um 11:16 UTC bei 20° über dem Horizont sein, Da am Eingang -6° vorgegeben sind berechnet der Baustein die Bürgerliche Dämmerung.

SUN_RISE ist diejenige Zeit wenn die Oberkante der Sonne am Horizont sichtbar wird. SUN_SET ist diejenige Zeit wenn die Oberkante der Sonne Hinter dem Horizont verschwindet. Der Horizont ist jedoch nur über dem Offenen Meer konstant bei 0° je nach Geländeform und Lage von Hügeln und Bergen können diese Zeiten für verschiedene Orte deutlich abweichen. Eine entsprechende Korrektur kann nur Ortsabhängig erfolgen, wobei als Grundlage für Korrekturen gilt das die Sonne in einer Minute 4° zurücklegt. Für praktische Anwendungen ausser auf dem offenen Meer müssen sowohl Aufgangs wie auch Untergangszeiten entsprechend korrigiert werden. Mit dem Eingang H kann definiert werden wie viele Grad vor beziehungsweise nach dem Horizont SUN_RISE und SUN_SET ermittelt wird. Wird am Eingang H nichts vorgegeben arbeitet der Baustein intern mit den Default von -0.83333 Grad was die Refraktion am Horizont kompensiert. Für bürgerliche, nautische oder astronomische Dämmerung werden am Eingang H die entsprechenden Werte (-6°, -12°, -18°) vorgegeben.

Für Sonnenaufgang und Sonnenuntergang gibt es verschiedenen Definitionen sowie Festlegungen:

Die Bürgerliche Dämmerung beschreibt diejenige Zeit wenn die Sonne 6° unter dem Horizont steht, es ist die Zeit wo bereits Tageshelle erreicht ist.

Als Nautische Dämmerung bezeichnet die Zeit wenn die Sonne 12° unter dem Horizont steht, es ist diejenige Zeit wenn die erste Aufhellung am Horizont feststellbar ist.

Die Astronomische Dämmerung ist die Zeit wenn die Sonne 18° unter dem Horizont steht, es ist die Zeit an der keinerlei Aufhellung durch die Sonne mehr messbar ist.

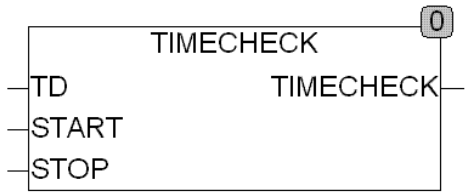
Zusätzliche Informationen zu Sonnenaufgangs und Untergangszeiten sind auf folgenden Webseiten zu finden:

http://www.calsky.com/cs.cgi

http://lexikon.astronomie.info/java/sunmoon/

12.52. TIMECHECK

Type	Funktion : BOOL
Input	TD : TOD (Tageszeit)
	START : TOD (Startzeit)
	STOP : TOD (Stoppzeit)
Output	BOOL (Rückgabewert)



TIMECHECK prüft ob die Tageszeit TD zwischen den Zeiten START und STOP liegt. TIMECHECK liefert TRUE wenn TD >= START und TD < STOP ist. Wird START und STOP so definiert das START > STOP ist so wird der Ausgang mit Start auf TRUE gesetzt und bleibt über Mitternacht TRUE bis am nächsten Tag STOP erreicht wird.

Für die Funktion gilt folgende Definition:

START < STOP : TD >= START AND TD < STOP

START > STOP : TD >= START OR TD < STOP

12.53. UTC_TO_LTIME

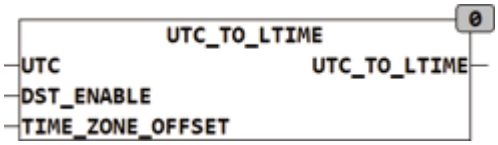
Type Funktionsbaustein

Input UTC : DATE_TIME (Weltzeit)

DST_ENABLE : BOOL (TRUE erlaubt Sommerzeit)

TIME_ZONE_OFFSET : INT (Zeitdifferenz zur Weltzeit in Minuten)

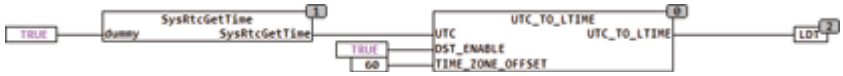
Output DT : DATE_TIME (Lokalzeit)



Der Funktionsbaustein UTC_TO_LTIME errechnet aus der Weltzeit am Eingang UTC eine Lokalzeit (LOCAL_DT) mit automatischer Sommerzeitumschaltung falls DST_ENABLE auf TRUE steht. Ist DST_ENABLE FALSE wird die Lokalzeit ohne Sommerzeitumschaltung berechnet.

Dieser Funktionsbaustein benötigt UTC am Eingang, welche normalerweise von der SPS zur Verfügung gestellt wird und durch eine Routine des Herstellers gelesen werden kann.

Im folgenden Beispiel ist die Anwendung für eine WAGO 750-841 CPU dargestellt. Das Auslesen der internen Uhr wird durch die Herstellerroutine SYSRTCGETTIME erledigt. Die SPS-Uhr muss in diesem Fall auf Weltzeit eingestellt werden.

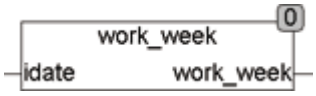


12.54. WORK_WEEK

Type Funktion : INT

Input IDATE : DATE (Eingangsdatum)

Output INT (Arbeitswoche des Eingangsdatums)



Die Funktion WORK_WEEK berechnet die Kalenderwoche aus dem Eingangsdatum IDATE. Die Kalenderwoche startet mit 1 für die erste Woche des Jahres. Der erste Donnerstag im Jahr liegt immer in der ersten Kalenderwoche. Wenn ein Jahr mit einem Donnerstag anfängt oder mit einem Donnerstag endet hat dieses Jahr 53 Kalenderwochen. Ist der erste Tag eines Jahres ein Dienstag, Mittwoch oder Donnerstag so beginnt die Kalenderwoche 1 bereits im Dezember des Vorjahres. Wenn der erste Tag eines Jahres Freitag, Samstag oder Sonntag ist so erstreckt sich die letzte Kalenderwoche des Vorjahres in den Januar. Die Berechnung erfolgt gemäß ISO8601.

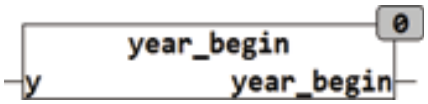
Da die Arbeitswoche (Work Week) international nicht immer einheitlich Verwendung findet ist vor der Anwendung der Funktion zu klären ob die Arbeitswoche gemäß ISO8601 der in der Anwendung gewünschten Funktion entspricht.

12.55. YEAR_BEGIN

Type Funktion : DATE

Input Y : INT (Jahreszahl)

Output DATE (Datum des 1. Januars für die Jahreszahl)



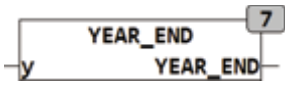
YEAR_BEGIN berechnet das Datum des 1. Januars für das Jahr Y.

12.56. YEAR_END

Type Funktion : DATE

Input Y : INT (Jahreszahl)

Output DATE (Datum des 31. Dezembers für die Jahreszahl)



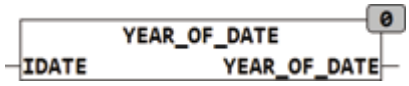
YEAR_END berechnet das Datum des 31. Dezembers für das Jahr Y.

12.57. YEAR_OF_DATE

Type Funktion : INT

Input IDATE : DATE (Eingangsdatum)

Output INT (Jahr des Eingangsdatums)



Die Funktion YEAR_OF_DATE berechnet das entsprechende Jahr aus dem Eingangsdatum IDATE.

Beispiel: YEAR_OF_DATE(31.12.2007) = 2007

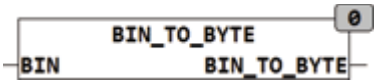
13. String Funktionen

13.1. BIN_TO_BYTE

Type Funktion : BYTE

Input BIN : STRING(12) (Oktale Zeichenkette)

Output BYTE (Ausgangswert)



Die Funktion BIN_TO_BYTE konvertiert eine binär kodierte Zeichenkette in einen BYTE Wert. Es werden dabei nur die binären Zeichen sind '0' und '1' interpretiert, alle anderen in BIN vorkommenden Zeichen werden ignoriert.

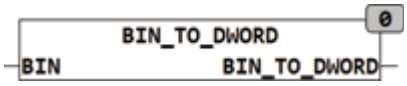
Beispiel: BIN_TO_BYTE('11') ergibt 3.

13.2. BIN_TO_DWORD

Type Funktion : DWORD

Input BIN : STRING(40) (Oktale Zeichenkette)

Output DWORD (Ausgangswert)



Die Funktion BIN_TO_DWORD konvertiert eine binär kodierte Zeichenkette in einen BYTE Wert. Es werden dabei nur die binären Zeichen sind '0' und '1' interpretiert, alle anderen in BIN vorkommenden Zeichen werden ignoriert.

Beispiel: BIN_TO_DWORD('11') ergibt 3.

13.3. BYTE_TO_STRB

Type Funktion : STRING

Input IN : BYTE (Eingangswert)

Output STRING(8) (Ergebnis STRING)

BYTE_TO_STRB konvertiert ein Byte in einen STRING fester Länge. Der Ausgangsstring ist exakt 8 Stellen lang und entspricht der bitweisen Schreibweise des Wertes IN. Der Ausgangsstring besteht aus den Zeichen '0' und '1'. Das niederwertigste Bit ist rechts im STRING. Falls ein STRING mit weniger als 8 Zeichen benötigt wird kann dieser mit der Standardfunktion RIGHT() entsprechend abgeschnitten werden. Der Aufruf RIGHT(BYTE_TO_STRB(X),4) ergibt einen STRING mit 4 Zeichen, die dem Inhalt der untersten 4 Bit von X entspricht.

Beispiel: BYTE_TO_STRB(3) = '00000011'

13.4. BYTE_TO_STRH

Type Funktion : STRING

Input IN : BYTE (Eingangswert)

Output STRING(2) (Ergebnis STRING)

BYTE_TO_STRH konvertiert ein Byte in einen STRING fester Länge. Der Ausgangsstring ist exakt 2 Stellen lang und entspricht der Hexadezimalen Schreibweise des Wertes von IN. Der Ausgangsstring besteht aus den Zeichen '0' .. '9' und 'A' .. 'F'. Das niederwertigste Zeichen steht rechts im STRING.

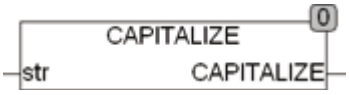
Beispiel: BYTE_TO_STRH(15) = '0F'

13.5. CAPITALIZE

Type Funktion : STRING

Input STR : STRING (Eingangsstring)

Output STRING (Ergebnis STRING)

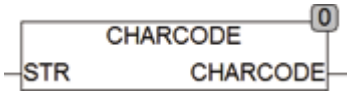


CAPITALIZE setzt alle Anfangsbuchstaben in STR auf Großbuchstaben. Bei der Konvertierung wird die Globale Setup Konstante EXTENDED_ASCII berücksichtigt. Wenn EXTENDED_ASCII = TRUE werden Zeichen des erweiterten ASCII Zeichensatzes nach ISO 8859-1 berücksichtigt.

CAPITALIZE('hugo wallenstein') = 'Hugo Wallenstein'

13.6. CHARCODE

Type	Funktion : BYTE
Input	STR : STRING(10) (Eingangsstring)
Output	BYTE (Zeichencode)



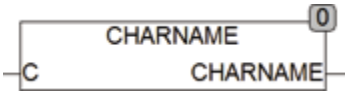
CHARCODE liefert den Byte Code eines Named Characters. Eine Liste der Codes mit Namen befindet sich unter der Funktion CHARNAME. Falls für den Namen in STR keine Zeichenname bekannt ist wird 0 zurückgegeben. Besteht STR nur aus einem Zeichen, so wird der Code dieses Zeichens zurückgegeben. CHARCODE benutzt die globalen Variablen SETUP.CHARNAMES die die Liste der Namen mit Codes enthalten.

Beispiel: CHARCODE('euro') = 128 und entspricht dem Zeichen €

CHARCODE(',') = 44

13.7. CHARNAME

Type	Funktion : STRING(10)
Input	C : BYTE (Zeichencode)
Output	STRING (Zeichenname)



CHARNAME ermittelt den Zeichennamen für einen Zeichencode.

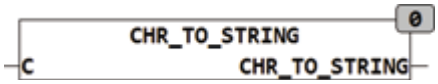
Beispiel: CHARNAME(128) = 'euro'

char	*code*	*name*	*char*	*code*	*name*	*char*	*code*	*name*
"	34	quot	½	189	frac12	ß	223	szlig
&	38	amp	¾	190	frac34	à	224	agrave
<	60	lt	¿	191	iquest	á	225	aacute
>	62	gt	À	192	Agrave	â	226	acirc
€	128	euro	Á	193	Aacute	ã	227	atilde
	160	nbsp	Â	194	Acirc	ä	228	auml
¡	161	iexcl	Ã	195	Atilde	å	229	aring
¢	162	cent	Ä	196	Auml	æ	230	aelig
£	163	pound	Å	197	Aring	ç	231	ccedil
¤	164	curren	Æ	198	AElig	è	232	egrave
¥	165	yen	Ç	199	Ccedil	é	233	eacute
¦	166	brvbar	È	200	Egrave	ê	234	ecirc
§	167	sect	É	201	Eacute	ë	235	euml
¨	168	uml	Ê	202	Ecirc	ì	236	igrave
©	169	copy	Ë	203	Euml	í	237	iacute
ª	170	ordf	Ì	204	Igrave	î	238	icirc
«	171	laquo	Í	205	Iacute	ï	239	iuml
¬	172	not	Î	206	Icirc	ð	240	eth
-	173	shy	Ï	207	Iuml	ñ	241	ntilde
®	174	reg	Ð	208	ETH	ò	242	ograve
¯	175	macr	Ñ	209	Ntilde	ó	243	oacute
°	176	deg	Ò	210	Ograve	ô	244	ocirc
±	177	plusmn	Ó	211	Oacute	õ	245	otilde
²	178	sup2	Ô	212	Ocirc	ö	246	ouml
³	179	sup3	Õ	213	Otilde	÷	247	divide
´	180	acute	Ö	214	Ouml	ø	248	oslash
µ	181	micro	×	215	times	ù	249	ugrave
¶	182	para	Ø	216	Oslash	ú	250	uacute
·	183	middot	Ù	217	Ugrave	û	251	ucirc
¸	184	cedil	Ú	218	Uacute	ü	252	uuml
¹	185	sup1	Û	219	Ucirc	ý	253	yacute
º	186	ordm	Ü	220	Uuml	þ	254	thorn
»	187	raquo	Ý	221	Yacute	ÿ	255	yuml
¼	188	frac14	Þ	222	THORN			

Falls für einen Code kein Name bekannt ist wird der Code als einzelnes Zeichen zurückgegeben. Für den Code 0 wird eine leere Zeichenkette zurückgegeben. CHARCODE benutzt die globalen Variablen SETUP.CHARNAMES die die Liste der Namen mit Codes enthalten.

13.8. CHR_TO_STRING

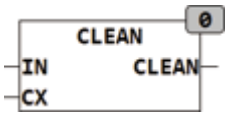
Type	Funktion : STRING
Input	C : Byte (Eingangswert)
Output	STRING (Ergebnis STRING)



CHR_TO_STRING formt ein ASCII Zeichen aus einem Byte und liefert es als einen ein Zeichen langen STRING.

13.9. CLEAN

Type	Funktion : STRING
Input	IN : STRING (Eingangswert)
	CX : STRING (Alle Zeichen die nicht gelöscht werden sollen)
Output	STRING (Ergebnis STRING)

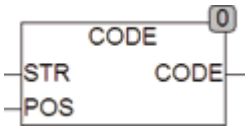


CLEAN löscht alle Zeichen aus einer Zeichenkette die nicht in der Zeichenkette CX enthalten sind.

CLEAN('Nr.1 23#', '0123456789') = '123'

13.10. CODE

Type	Funktion : BYTE
Input	STR : STRING (Zeichenkette)
	POS : INT (Position an der das Zeichen gelesen wird)
Output	BYTE (Code des Zeichens an der Position POS)

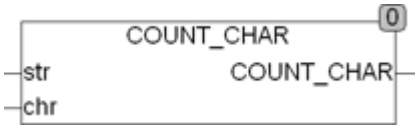


CODE ermittelt den Numerischen Code eines Zeichens an der Stelle POS in STR. wird CODE mit einer Position kleiner 1 oder größer der Länge von STR aufgerufen wird 0 zurückgegeben.

Beispiel: CODE('ABC 123',4) = 32 (Das Zeichen ' ' wird mit dem Wert 32 kodiert.

13.11. COUNT_CHAR

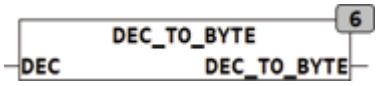
Type	Funktion : STRING
Input	STR : STRING (Zeichenkette)
	CHR : Byte (Suchzeichen)
Output	STRING (Ergebnis STRING)



COUNT_CHAR ermittelt wie oft das Zeichen CHR in der Zeichenkette STR vorkommt. Um auch nach Sonderzeichen und Steuerzeichen suchen zu können wird das Suchzeichen CHR als BYTE angegeben.

13.12. DEC_TO_BYTE

Type	Funktion : BYTE
Input	DEC : STRING(10) (dezimale kodierte Zeichenkette)
Output	BYTE (Ausgangswert)

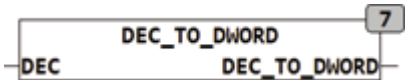


Die Funktion DEC_TO_BYTE konvertiert eine dezimal kodierte Zeichenkette in einen BYTE Wert. Es werden dabei nur die dezimalen Zeichen sind '0'..'9' interpretiert, alle anderen in DEC vorkommenden Zeichen werden ignoriert.

Beispiel: DEC_TO_BYTE('34') ergibt 34.

13.13. DEC_TO_DWORD

Type	Funktion : DWORD
Input	DEC : STRING(20) (dezimale kodierte Zeichenkette)
Output	DWORD (Ausgangswert)



Die Funktion DEC_TO_DWORD konvertiert eine dezimal kodierte Zeichenkette in einen BYTE Wert. Es werden dabei nur die dezimalen Zeichen sind '0'..'9' interpretiert, alle anderen in DEC vorkommenden Zeichen werden ignoriert.

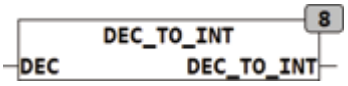
Beispiel: DEC_TO_DWORD('34') ergibt 34.

13.14. DEC_TO_INT

Type Funktion : INT

Input DEC : STRING(10) (dezimale kodierte Zeichenkette)

Output INT (Ausgangswert)



Die Funktion DEC_TO_INT konvertiert eine dezimal kodierte Zeichenkette in einen BYTE Wert. Es werden dabei nur die dezimalen Zeichen sind '0'..'9' und '-' interpretiert, alle anderen in DEC vorkommenden Zeichen werden ignoriert.

Beispiel: DEC_TO_INT('-34') ergibt -34.

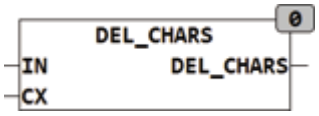
13.15. DEL_CHARS

Type Funktion : STRING

Input IN : STRING (Eingangswert)

CX : STRING (Alle Zeichen die gelöscht werden sollen)

Output STRING (Ergebnis STRING)

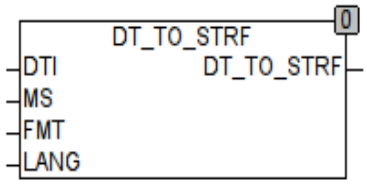


DEL_CHARS löscht alle Zeichen aus einer Zeichenkette die in der Zeichenkette CX enthalten sind.

CLEAN('Nr.1 23#', ' #ABCDEFG') = 'Nr.123'

13.16. DT_TO_STRF

Type	Funktion : STRING
Input	DTI : DT (Datum und Zeit Eingangswert)
	MS : INT (Millisekunden Eingang)
	FMT : STRING (Vorgabe für Ausgangs Format)
	LANG : INT (Sprachvorgabe)
Output	STRING (Ergebnis String)



DT_TO_STRF konvertiert einen DATETIME Wert in eine formatierte Zeichenkette. Am Eingang DTI liegt der zu konvertierende DATETIME Wert an und mit der Zeichenkette FMT wird das entsprechende Ausgangsformat bestimmt. Der Eingang LANG bestimmt dabei die zu benutzende Sprache (0= LANGUAGE_DEFAULT, 1= Englisch und 2 = Deutsch). Die Spracheinstellungen werden im entsprechenden Absatz der Globalen Konstanten vorgenommen und können dort angepasst oder Erweitert werden. Zusätzlich zu Datum und Zeit können am Eingang MS auch Millisekunden verarbeitet werden.

Die erzeugte Zeichenkette entspricht der Zeichenkette FMT wobei in der Zeichenkette alle Zeichen '#' gefolgt von einem Großbuchstaben mit dem entsprechenden Wert ersetzt werden. Die folgende Tabelle definiert die Formatierungszeichen:

#A	Jahreszahl mit 4 Stellen (2008)
#B	Jahreszahl 2-Stellig z.B. (08)
#C	Monat 1-2 Stellig (1,12)
#D	Monat 2 Stellig (01, 12)
#E	Monat 3 Buchstaben (Jan)
#F	Monat ausgeschrieben (Januar)
#G	Tag 1 oder 2 stellig (1, 31)
#H	Tag 2 Stellig (01, 31)
#I	Wochentag als Zahl (1 = Montag, 7= Sonntag)
#J	Wochentag 2 Buchstaben (Mo)
#K	Wochentag ausgeschrieben (Montag)
#L	AM oder PM für Amerikanische Datumsformate
#M	Stunde in 24 Stunden Format 1 - 2 Stellig (0, 23)
#N	Stunde in 24 Stunden Format 2 Stellig (00, 23)
#O	Stunde in 12 Stunden Format 1 - 2 Stellig (1, 12)
#P	Stunde in 12 Stunden Format 2 Stellig (01, 12)
#Q	Minuten 1 - 2 Stellig (0, 59)
#R	Minuten 2 Stellig (00, 59)
#S	Sekunden 1 - 2 Stellig (0, 59)
#T	Sekunden 2 Stellig (00, 59)
#U	Millisekunden 1 - 3 Stellig (0, 999)
#V	Millisekunden 3 Stellig (000, 999)
#W	Tag 2 Stellig aber vorne mit Blank aufgefüllt (' 1' ..'31')
#X	Monat 2 Stellig aber vorne mit Blank aufgefüllt (' 1' ..'12')

Beispiele:

DT_TO_STRF(DT#2008-1-1, 'Datum '#C. #F #A', 2) = '1. Januar 2008'

DT_TO_STRF(DT#2008-1-1-13:43:12, '#J #M:#Q am #C. #E #A', 2) =

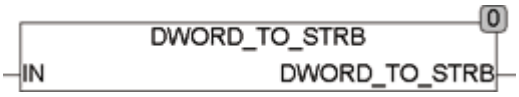
'Di 13:43 am 1. Jan 2008'

13.17. DWORD_TO_STRB

Type Funktion : STRING

Input IN : DWORD (Eingangswert)

Output STRING(32) (Ergebnis String)



DWORD_TO_STRB konvertiert ein DWORD, Word oder Byte in einen STRING fester Länge. Der Ausgangsstring ist exakt 32 Stellen lang und entspricht der bitweisen Schreibweise des Wertes IN. Der Ausgangsstring besteht aus den Zeichen '0' und '1'. Das niederwertigste Bit steht links im STRING. DWORD_TO_STRB kann Eingangsformate Byte, Word und DWORD Typen verarbeiten. Der Ausgang ist aber unabhängig vom Eingangstyp immer ein STRING mit 32 Zeichen. Falls ein kürzerer STRING benötigt wird, kann dieser mit der Standard Funktion RIGHT() entsprechend abgeschnitten werden. Der Aufruf RIGHT(DWORD_TO_STRB(X),8) ergibt einen STRING mit 8 Zeichen die dem Inhalt des untersten Bytes von X entsprechen.

Beispiel:

DWORD_TO_STRB(127) = '00000000000000000000000001111111'

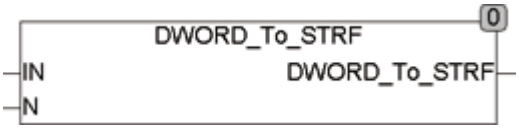
13.18. DWORD_TO_STRF

Type Funktion : STRING

Input IN : DWORD (Eingangswert)

N : Int (Länge des Ergebnis Strings)

Output STRING (Ergebnis String)



DWORD_TO_STRF konvertiert ein DWORD, Word oder Byte in einen STRING fester Länge. Der Ausgangsstring ist exakt N Stellen lang, wobei

führende Nullen eingefügt werden oder führende Stellen abgeschnitten werden. Die maximale erlaubte Länge N ist 20 Digits.

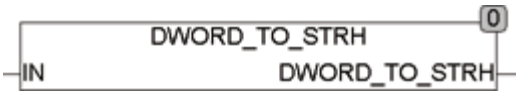
Beispiel: DWORD_TO_STRF(5123, 6) = '005123'
DWORD_TO_STRF(5123, 3) = '123'

13.19. DWORD_TO_STRH

Type Funktion : STRING

Input IN : DWORD (Eingangswert)

Output STRING(8) (Ergebnis String)



DWORD_TO_STRH konvertiert ein DWORD, Word oder Byte in einen STRING fester Länge. Der Ausgangsstring ist exakt 8 Stellen lang und entspricht der hexadezimalen Schreibweise des Wertes IN. Der Ausgangsstring besteht aus den Zeichen '0' .. '1' und 'A' .. 'F'. Das niederwertigste Hexadezimal-Zeichen steht rechts im STRING. DWORD_TO_STRH kann als Input Byte, Word und DWORD Typen verarbeiten. Der Ausgang ist aber unabhängig vom Eingangstyp immer ein STRING mit 32 Zeichen. Falls ein kürzerer STRING benötigt wird, kann dieser mit der Standardfunktion RIGHT() entsprechend abgeschnitten werden. Der Aufruf RIGHT(DWORD_TO_STRH(X),4) ergibt einen STRING mit 4 Zeichen die dem Inhalt der untersten 2 Bytes von X entsprechen.

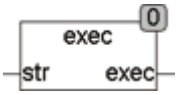
Beispiel: DWORD_TO_STRH(127) = '0000007F'

13.20. EXEC

Type Funktion : STRING

Input STR : STRING (Eingabe STRING)

Output STRING (Ergebnis STRING)



Die Funktion EXEC arbeitet Mathematische Ausdrücke ab und liefert das Ergebnis als STRING zurück. Der Ausdruck darf nur ein einfacher Ausdruck

mit einem Operator und ohne Klammern sein. Bei Fehlern, wie zum Beispiel einem Teilen durch Null liefert EXEC den Rückgabestring 'ERROR'.

Die zulässigen Operatoren sind: +, - *, /, ^, SIN, COS, TAN, SQRT.

Als Zahlen sind REAL und Integer-Zahlen zulässig.

Beispiel: EXEC('3^2') = '9'

EXEC('4-2') = '2'

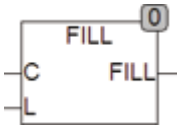
13.21. FILL

Type Funktion : STRING

Input C : BYTE (Character Code)

L : INT (Länge der Zeichenkette)

Output STRING (Ergebnis STRING)



FILL erzeugt eine Zeichenkette bestehend aus dem Zeichen C mit der Länge L.

FILL(49,5) = '11111'

Die Funktion FILL wertet auch die Globale Setup Konstante STRING_LENGTH aus und begrenzt die maximale Länge L der Zeichenkette auf STRING_LENGTH.

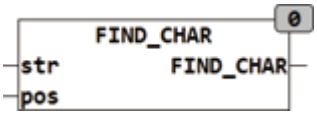
13.22. FIND_CHAR

Type Funktion : INT

Input STR : STRING (Eingabe STRING)

POS : INT (Startposition)

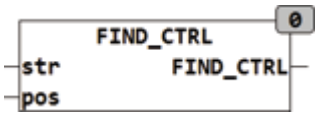
Output INT (Position des ersten Zeichens das kein Steuerzeichen ist)



FIND_CHAR durchsucht die Zeichenkette STR ab der Position POS und gibt die Position zurück an der das erste Zeichen steht das kein Steuerzeichen ist. Steuerzeichen sind alle Zeichen deren Wert kleiner 32 oder 127 ist. Bei der Prüfung wird die Globale Setup Konstante EXTENDED_ASCII berücksichtigt. Wenn EXTENDED_ASCII = TRUE ist werden Zeichen des erweiterten ASCII Zeichensatzes nach ISO 8859-1 berücksichtigt. Umlaute wie Ä,Ö,Ü werden nur dann berücksichtigt wenn die Globale Konstante EXTENDED_ASCII = TRUE ist. Wenn EXTENDED_ASCII = FALSE ist werden Zeichen des erweiterten Zeichensatzes mit einem Wert > 127 als Steuerzeichen interpretiert.

13.23. FIND_CTRL

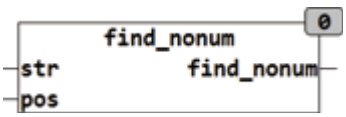
Type	Funktion : INT
Input	STR : STRING (Eingabe STRING)
	POS : INT (Startposition)
Output	INT (Position des ersten Zeichens das ein Steuerzeichen ist)



FIND_CTRL durchsucht die Zeichenkette STR ab der Position POS und gibt die Position zurück an der das nächste Steuerzeichen steht. Steuerzeichen sind alle Zeichen deren Wert kleiner 32 oder 127 ist.

13.24. FIND_NONUM

Type	Funktion : INT
Input	STR : STRING (Eingabestring)
	POS : INT (Position an der die Suche beginnt)
Output	INT (Position des ersten Zeichens, das keine Zahl oder Punkt ist)



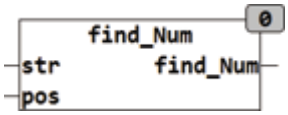
Die Funktion FIND_NONUM durchsucht STR ab der Startposition POS von links nach rechts und liefert die erste Stelle die keine Nummer ist zurück.

Nummern sind die Buchstaben "0..9" und "."

Beispiel: FIND_NONUM('4+33',1) = 2

13.25. FIND_NUM

Type	Funktion : INT
Input	STR : STRING (Eingabestring)
	POS : INT (Position an der die Suche beginnt)
Output	INT (Position des ersten Zeichens, das eine Zahl oder Punkt ist)



Die Funktion FIND_NUM durchsucht STR ab der Position POS von links nach rechts und liefert die erste Stelle die eine Nummer ist zurück.

Nummern sind die Buchstaben "0..9" und "."

Beispiel: FIND_NONUM('4+33',1) = 1

13.26. FINDB

Type	Funktion : INT
Input	STR1 : STRING (Eingabestring)
	STR2 : STRING (Suchstring)
Output	INT (Position des letzten Vorkommens von STR2 in STR1)

Die Funktion FINDB durchsucht STR1 auf das Vorkommen von STR2 und liefert die letzte Position von STR2 in STR1 zurück.

Falls STR2 nicht gefunden wird, wird eine 0 zurückgegeben.

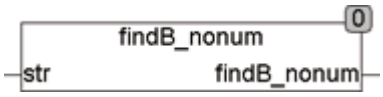
Beispiel: FINDB('abs12fir12bus12', '12') = 14

13.27. FINDB_NONUM

Type Funktion : INT

Input STR : STRING (Eingabestring)

Output INT (Position des letzten Buchstaben, der keine Nummer ist)



Die Funktion FINDB_NONUM durchsucht STR von rechts nach links und liefert die letzte Stelle die keine Nummer ist zurück.

Nummern sind die Buchstaben "0..9" und ".".

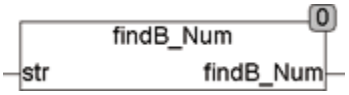
Beispiel: FINDB_NONUM('4+33+1') = 5

13.28. FINDB_NUM

Type Funktion : INT

Input STR : STRING (Eingabestring)

Output INT (Position des letzten Zeichens, das eine Zahl oder Punkt ist)



Die Funktion FINDB_NUM durchsucht STR von Rechts nach links und liefert die letzte Stelle die eine Nummer ist.

Nummern sind die Buchstaben "0..9" und ".".

Beispiel: FINDB_NUM('4+33+1hh') = 6

13.29. FINDP

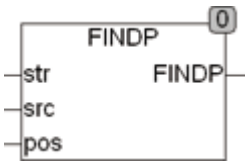
Type Funktion : INT

Input STR : STRING (Eingabestring)

SRC : STRING (Suchstring)

POS : INT (Position ab der gesucht wird)

Output INT (Position des ersten Buchstabens des gefundenen Strings)



FINDP sucht in einer Zeichenkette STR ab der Position POS nach einer Zeichenkette SRC. Wird die Zeichenkette SRC gefunden so wird die Position des ersten Zeichens von SRC innerhalb von STR ausgegeben. Wird die Zeichenkette ab der Position POS nicht gefunden wird eine 0 ausgegeben. Wird eine Leere Zeichenkette als Suchstring vorgegeben liefert der Baustein das Ergebnis 0.

Beispiel: FINDP('ein Fuchs ist ein Tier','ein',1) = 1;

FINDP('ein Fuchs ist ein Tier','ein',2) = 15;

FINDP('ein Fuchs ist ein Tier','ein',16) = 0;

13.30. FIX

Type Funktion : STRING

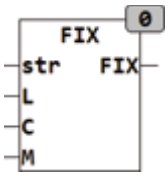
Input STR : STRING (Eingabestring)

L : INT (feste Länge des Ausgabestrings)

C : BYTE (Füllzeichen beim Auffüllen)

M : INT (Mode zum Auffüllen)

Output STRING (Zeichenkette mit fester Länge N)



FIX erzeugt eine Zeichenkette mit fester Länge N. Die Zeichenkette STR am Eingang wird auf die Länge N abgeschnitten beziehungsweise mit dem Füllzeichen C aufgefüllt. Wenn die Zeichenkette STR kürzer ist als die zu erzeugende Länge L wird abhängig von M die Zeichenkette mit dem Füllzeichen C aufgefüllt. Wenn M = 0 werden die Füllzeichen am Ende der Zei-

chenkette angehängt, ist M = 1 werden die Füllzeichen am Anfang angehängt, und wenn M = 2 wird die Zeichenkette zwischen Füllzeichen zentriert. Falls die Anzahl der nötigen Füllzeichen ungerade ist wird bei M = 2 am Ende ein Füllzeichen mehr als am Anfang angehängt. Die Funktion FIX wertet auch die Globale Setup Konstante STRING_LENGTH aus und begrenzt die maximale Länge L der Zeichenkette auf STRING_LENGTH.

13.31. FLOAT_TO_REAL

Type Funktion : REAL

Input FLT : STRING(20) (Gleitkommazahl)

Output REAL (REAL Wert der Gleitkommazahl)

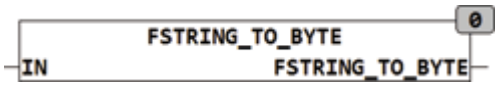
FLOAT_TO_REAL wandelt eine als STRING vorliegende Gleitpunktzahl in einen Datentyp REAL um. Bei Der Umwandlung werden '.' oder ',' als Komma interpretiert und 'E' oder 'e' als Trennzeichen des Exponenten. Die Zeichen '-0123456789' werden Ausgewertet und alle anderen in FLT vorkommenden Zeichen werden ignoriert.

13.32. FSTRING_TO_BYTE

Type Funktion : BYTE

Input IN : STRING(12) (Eingabestring)

Output BYTE (Byte Wert)



FSTRING_TO_BYTE konvertiert eine formatierten Zeichenkette in einen Byte Wert. Es werden folgende Eingabeformate unterstützt:

2#0101 (binär), 8#345 (oktal), 16#2a33 (hexadezimal) und 234 (dezimal).

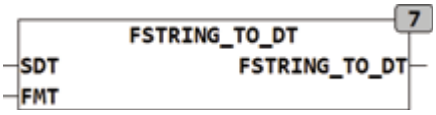
13.33. FSTRING_TO_DT

Type Funktion : DT

Input SDT : STRING(60) (Eingabestring)

FMT : STRING(60) (Formatierung)

Output DT (ermitteltes Datum und Uhrzeit)



FSTRING_TO_DT konvertiert eine formatierte Zeichenkette in einen DATE-TIME Wert. Mithilfe der Zeichenkette FMT wird eine Formatierung zur Dekodierung vorgegeben. Das Zeichen '#' gefolgt von einem Buchstaben definiert die zu Dekodierende Information.

#Y	Jahr in der Schreibweise 08 oder 2008
#M	Monat in der Schreibweise 01 oder 1
#N	Monat in der Schreibweise 'JAN' oder 'Januar' (Groß und Kleinschreibung wird ignoriert)
#D	Tag in der Schreibweise 01 oder 1
#h	Stunde in der Schreibweise 01 oder 1
#m	Minute in der Schreibweise 01 oder 1
#s	Sekunde in der Schreibweise 01 oder 1

Beispiele:

FSTRING_TO_DT('25. September 2008 um 10:01:00', '#D. #N #Y ** #h:#m:#s')

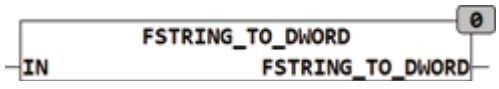
FSTRING_TO_DT('13:14', '#h:#m')

13.34. FSTRING_TO_DWORD

Type Funktion : DWORD

Input IN : STRING(40) (Eingabestring)

Output DWORD (32bit Wert)

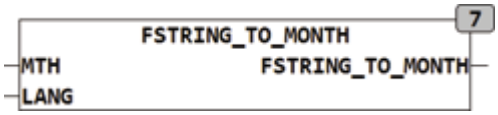


FSTRING_TO_DWORD konvertiert eine formatierten Zeichenkette in einen 32bit Wert. Es werden folgende Eingabeformate unterstützt:

2#0101 (binär), 8#345 (oktal), 16#2a33 (hexadezimal) und 234 (dezimal).

13.35. FSTRING_TO_MONTH

Type	Funktion : INT
Input	MTH : STRING(20) (Eingabestring)
	LANG : INT (Sprachauswahl)
Output	INT (Monatszahl 1..12)



FSTRING_TO_MONTH ermittelt aus einer Zeichenkette mit einem Monatsnamen oder Kürzel den Zahlenwert des Monats. Die Funktion kann als Eingang sowohl die Monatsnamen und Kürzel als auch eine Monatszahl verarbeiten.

FSTRING_TO_MONTH('Januar',2) = 1

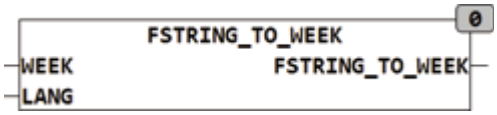
FSTRING_TO_MONTH('Jan',2) = 1

FSTRING_TO_MONTH('11',0) = 11

Der Eingang LANG selektiert die zu verwendende Sprache, 0 = die im Setup eingestellte Default Sprache, 1 = Englisch .... nähere Infos zu den Spracheinstellungen finden Sie im Kapitel Datentypen.

13.36. FSTRING_TO_WEEK

Type	Funktion : BYTE
Input	WEEK : STRING(60) (Eingabestring)
	LANG : INT (Sprachauswahl)
Output	BYTE (Bitpattern der Wochentage)



FSTRING_TO_WEEK dekodiert eine Liste von Wochentagen in der Form 'MO,DI,3' in ein Bitpattern (Bit6 = MO...Bit0 = So). Für die Auswertung werden jeweils die ersten beiden Buchstaben der Listenelemente ausgewertet, alle folgenden werden ignoriert. Falls die Zeichenkette Leerzeichen enthält werden diese entfernt. Die Wochentage können sowohl in Groß- oder Klein- Schreibung vorliegen. LANG spezifiziert die zu verwendende Sprache, 1= Englisch, 2= Deutsch, 0 ist die im Setup definierte Default Sprache.

Mo = 1; Di, Tu = 2; We, Mi = 3; Th, Do = 4; Fr = 5; Sa = 6; So, Su = 7

Da die Funktion nur die ersten beiden Zeichen auswertet, können die Wochentage auch in ausgeschriebener Form (Montag) vorliegen.

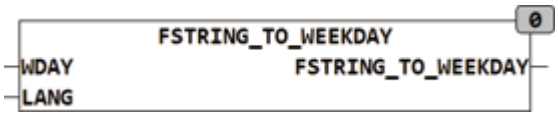
Als alternative Form kann der Wochentag auch als Zahl 1..7 angegeben werden.

Die Liste enthält die einzelnen Wochentage unsortiert mit Komma getrennt.

FSTRING_TO_WEEK('Mo,Di,Sa',2) = 2#01100010.

13.37. FSTRING_TO_WEEKDAY

Type	Funktion : INT
Input	WDAY : STRING(20) (Eingabestring)
	LANG : INT (Sprachauswahl)
Output	INT (Wochentag)



FSTRING_TO_WEEKDAY dekodiert einen Wochentag in der Form 'MO' in einen Integer, 1 = MO...7= So. Für die Auswertung werden die ersten beiden Buchstaben der Zeichenkette WDAY ausgewertet, alle folgenden werden ignoriert. Falls die Zeichenkette Leerzeichen enthält werden diese entfernt. Die Wochentage können sowohl in Groß- oder Klein- Schreibung vorliegen. Da die Funktion nur die ersten beiden Zeichen auswertet, können die Wochentage auch in ausgeschriebener Form (Montag) vorliegen.

Mo = 1; Di, Tu = 2; We, Mi = 3; Th, Do = 4; Fr = 5; Sa = 6; So, Su = 7

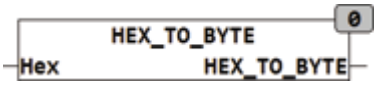
Als alternative Form kann der Wochentag auch als Zahl 1..7 angegeben werden. LANG spezifiziert die zu verwendende Sprache, 1= Englisch, 2= Deutsch, 0= die im Setup definierte Default Sprache.

13.38. HEX_TO_BYTE

Type Funktion : BYTE

Input HEX : STRING(5) (Hexadezimale Zeichenkette)

Output BYTE (Ausgangswert)



Die Funktion HEX_TO_BYTE konvertiert eine hexadezimale Zeichenkette in einen BYTE Wert. Es werden dabei nur die hexadizimalen Zeichen sind '0'..'9', 'a..f' und 'A' .. 'F' interpretiert, alle anderen in HEX vorkommenden Zeichen werden ignoriert.

Beispiel: HEX_TO_BYTE('FF') ergibt 255.

13.39. HEX_TO_DWORD

Type Funktion : DWORD

Input HEX : STRING(20) (Hexadezimale Zeichenkette)

Output DWORD (Ausgangswert)

Die Funktion HEX_TO_DWORD konvertiert eine hexadezimale Zeichenkette in einen DWORD Wert. Es werden dabei nur die hexadizimalen Zeichen sind '0'..'9', 'a..f' und 'A' .. 'F' interpretiert, alle anderen in HEX vorkommenden Zeichen werden ignoriert.

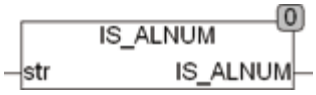
Beispiel: HEX_TO_DWORD('FF') ergibt 255.

13.40. IS_ALNUM

Type Funktion : BOOL

Input STR : STRING (Eingabestring)

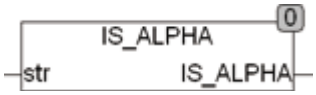
Output BOOL (TRUE wenn STR nur Buchstaben oder Zahlen enthält)



IS_ALNUM testet ob in der Zeichenkette STR nur Buchstaben oder Zahlen enthalten sind. Wird ein falsches, nicht alphanumerisches Zeichen gefunden gibt die Funktion FALSE zurück. Enthält STR nur Buchstaben oder Zahlen ist das Ergebnis TRUE. Buchstaben sind die Zeichen A..Z und a..z, und Zahlen sind die Zeichen 0..9. Bei der Prüfung wird die Globale Setup Konstante EXTENDED_ASCII berücksichtigt. Wenn EXTENDED_ASCII = TRUE ist werden Zeichen des erweiterten ASCII Zeichensatzes nach ISO 8859-1 berücksichtigt. Umlaute wie Ä,Ö,Ü werden nur dann berücksichtigt wenn die Globale Konstante EXTENDED_ASCII = TRUE ist.

13.41. IS_ALPHA

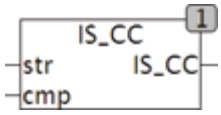
Type	Funktion : BOOL
Input	STR : STRING (Eingabestring)
Output	BOOL (TRUE wenn STR nur Buchstaben enthält)



IS_ALPHA testet ob in der Zeichenkette STR nur Buchstaben enthalten sind. Wird ein falsches, nicht alphabetisches Zeichen gefunden gibt die Funktion FALSE zurück. Wenn in STR nur Buchstaben enthalten sind ist das Ergebnis TRUE. Buchstaben sind die Zeichen A..Z und a..z. Bei der Prüfung wird die Globale Setup Konstante EXTENDED_ASCII berücksichtigt. Wenn EXTENDED_ASCII = TRUE ist werden Zeichen des erweiterten ASCII Zeichensatzes nach ISO 8859-1 berücksichtigt. Umlaute wie Ä,Ö,Ü werden nur dann berücksichtigt wenn die Globale Konstante EXTENDED_ASCII = TRUE ist.

13.42. IS_CC

Type	Funktion : BOOL
Input	STR : STRING (Eingabestring)
	CMP : STRING (Vergleichszeichen)
Output	BOOL (TRUE wenn STR nur die im STRING CMP aufgelisteten Zeichen enthält)



IS_CC testet ob in der Zeichenkette STR nur die in STR aufgelisteten Zeichen enthalten sind. Wird ein anderes Zeichen gefunden gibt die Funktion FALSE zurück.

Beipiele:

IS_CC('3.14', '0123456789.') = TRUE

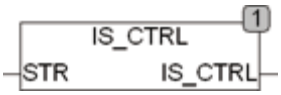
IS_CC('-3.14', '0123456789.') = FALSE

13.43. IS_CTRL

Type Funktion : BOOL

Input STR : STRING (Eingabestring)

Output BOOL (TRUE wenn STR nur Kontrollzeichen enthält)



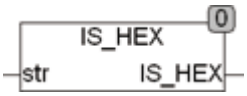
IS_CTRL testet ob in der Zeichenkette STR nur Kontrollzeichen enthalten sind. Wird ein anderes Zeichen gefunden gibt die Funktion FALSE zurück. Sind in STR nur Kontrollzeichen enthalten gibt die Funktion TRUE zurück. Kontrollzeichen sind die Zeichen mit dem Dezimalcode 0..31 und 127.

13.44. IS_HEX

Type Funktion : BOOL

Input STR : STRING (Eingabestring)

Output BOOL (TRUE wenn STR nur Hexadezimalzeichen enthält)



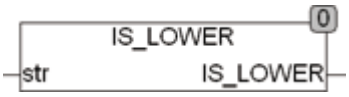
IS_HEX testet ob in der Zeichenkette STR nur Hexadezimalzeichen enthalten sind. Wird ein anderes Zeichen gefunden gibt die Funktion FALSE zurück. Sind in STR nur Hexadezimalzeichen enthalten gibt die Funktion TRUE zurück. Hexadezimalzeichen sind die Zeichen mit dem Dezimalcode 0..9, a..f und A..F.

13.45. IS_LOWER

Type Funktion : BOOL

Input STR : STRING (Eingabestring)

Output BOOL (TRUE wenn STR nur Kleinbuchstaben enthält)



IS_LOWER testet ob in der Zeichenkette STR nur Kleinbuchstaben enthalten sind. Wird etwas anderes als ein Kleinbuchstabe gefunden gibt die Funktion FALSE zurück. Sind in STR nur Kleinbuchstaben enthalten gibt die Funktion TRUE zurück. Bei der Prüfung wird die Globale Setup Konstante EXTENDED_ASCII berücksichtigt. Wenn EXTENDED_ASCII = TRUE ist werden Zeichen des erweiterten ASCII Zeichensatzes nach ISO 8859-1 berücksichtigt. Umlaute wie Ä,Ö,Ü werden nur dann berücksichtigt wenn die Globale Konstante EXTENDED_ASCII = TRUE ist.

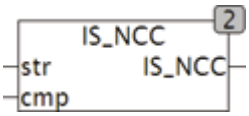
13.46. IS_NCC

Type Funktion : BOOL

Input STR : STRING (Eingabestring)

CMP : STRING (Vergleichszeichen)

Output BOOL (TRUE wenn STR keine der im STRING CMP aufgelisteten Zeichen enthält)



IS_NCC testet ob in der Zeichenkette STR keine der in STR aufgelisteten Zeichen enthalten sind. Wird ein Zeichen aus CMP in STR gefunden gibt die Funktion FALSE zurück.

Beipiele:

IS_NCC('3.14', ',-+()') = TRUE

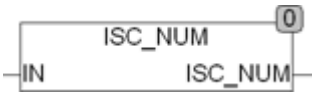
IS_NCC('-3.14', ',-+()') = FALSE

13.47. IS_NUM

Type Funktion : BOOL

Input STR : STRING (Eingabestring)

Output BOOL (TRUE wenn STR keine Großbuchstaben enthält)



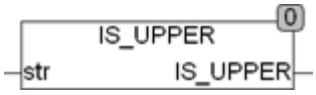
IS_NUM testet ob in der Zeichenkette STR nur Zahlen enthalten sind. Wird ein anderes Zeichen gefunden gibt die Funktion FALSE zurück. Sind in STR nur Zahlen enthalten gibt die Funktion TRUE zurück. Zahlen sind die Zeichen 0..9.

13.48. IS_UPPER

Type Funktion : BOOL

Input STR : STRING (Eingabestring)

Output BOOL (TRUE wenn STR nur Großbuchstaben enthält)



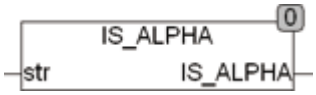
IS_UPPER testet ob in der Zeichenkette STR nur Großbuchstaben enthalten sind. Wird etwas anders als ein Großbuchstabe gefunden gibt die Funktion FALSE zurück. Sind in STR nur Großbuchstaben enthalten gibt die Funktion TRUE zurück. Bei der Prüfung wird die Globale Setup Konstante EXTENDED_ASCII berücksichtigt. Wenn EXTENDED_ASCII = TRUE ist werden Zeichen des erweiterten ASCII Zeichensatzes nach ISO 8859-1 berücksichtigt. Umlaute wie Ä,Ö,Ü werden nur dann berücksichtigt wenn die Globale Konstante EXTENDED_ASCII = TRUE ist.

13.49. ISC_ALPHA

Type Funktion : BOOL

Input IN : BYTE (Zeichen)

Output BOOL (TRUE IN ein Zeichen a..z, A..Z oder Umlaut ist)



ISC_ALPHA testet ob das Zeichen IN ein alphabetisches Zeichen ist, Ist IN ein Zeichen A..Z, a..z oder ein beliebiger Umlaut gibt die Funktion TRUE zurück, wenn nicht gibt die Funktion FALSE zurück. Bei der Prüfung wird die Globale Setup Konstante EXTENDED_ASCII berücksichtigt. Wenn EXTENDED_ASCII = TRUE ist werden Zeichen des erweiterten ASCII Zeichensatzes nach ISO 8859-1 ausgewertet. Umlaute wie Ä,Ö,Ü werden nur dann berücksichtigt wenn die Globale Konstante EXTENDED_ASCII = TRUE ist.

Die folgende Tabelle Erläutert die Codes:

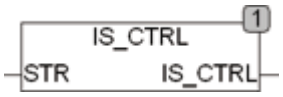
Code	EXTENDED_ASCII = TRUE	EXTENDED_ASCII = FAS-
0..64	FALSE	FALSE
65..90	TRUE	TRUE
91..96	FALSE	FALSE
97..122	TRUE	TRUE
123..191	FALSE	FALSE
192..214	TRUE	FALSE
215	FALSE	FALSE
216..246	TRUE	FALSE
247	FALSE	FALSE
248..255	TRUE	FALSE

13.50. ISC_CTRL

Type Funktion : BOOL

Input IN : BYTE (Zeichen)

Output BOOL (TRUE IN ein Zeichen 0..9 ist))



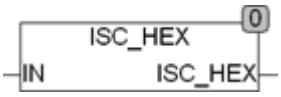
ISC_CTRL testet ob ein Zeichen IN ein Kontrollzeichen ist, Ist IN ein Kontrollzeichen gibt die Funktion TRUE zurück, wenn nicht gibt die Funktion FALSE zurück. Kontrollzeichen sind alle Zeichen mit dem Code < 32 oder 127.

13.51. ISC_HEX

Type Funktion : BOOL

Input IN : BYTE (Zeichen)

Output BOOL (TRUE IN ein Zeichen 0..9 ist))



ISC_HEX testet ob ein Zeichen IN ein Hexadezimales Zeichen ist, Ist IN ein Zeichen 0..9, A..F, a..f gibt die Funktion TRUE zurück, wenn nicht gibt die Funktion FALSE zurück.

Die Zeichen 0..9 haben die Codes (48..57)

Die Zeichen A..F haben die Codes (65..70)

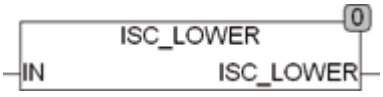
Die Zeichen a..f haben die Codes (97..102)

13.52. ISC_LOWER

Type Funktion : BOOL

Input IN : BYTE (Zeichen)

Output BOOL (TRUE IN ein Zeichen 0..9 ist)



ISC_LOWER testet ob ein Zeichen IN ein Kleinbuchstabe ist, Ist IN ein Kleinbuchstabe gibt die Funktion TRUE zurück, wenn nicht gibt die Funktion FALSE zurück. Bei der Prüfung wird die Globale Setup Konstante EXTENDED_ASCII berücksichtigt. Wenn EXTENDED_ASCII = TRUE ist werden Zeichen des erweiterten ASCII Zeichensatzes nach ISO 8859-1 ausgewertet.

Die folgende Tabelle Erläutert die Zeichencodes:

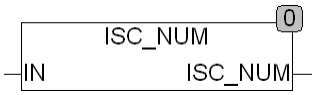
Code	EXTENDED_ASCII = TRUE	EXTENDED_ASCII = FASLE
0..96, 123..223, 247, 255	FALSE	FALSE
97..122	TRUE	TRUE
224..246	TRUE	FALSE
248..254	TRUE	FALSE

13.53. ISC_NUM

Type Funktion : BOOL

Input IN : BYTE (Zeichen)

Output BOOL (TRUE IN ein Zeichen 0..9 ist)



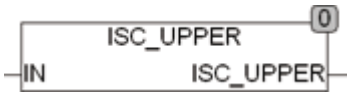
ISC_NUM testet ob ein Zeichen IN ein Numerisches Zeichen ist, Ist IN ein Zeichen 0..9 gibt die Funktion TRUE zurück, wenn nicht gibt die Funktion FALSE zurück. Die Zeichen von 0..9 haben die Zeichencodes (48..57).

13.54. ISC_UPPER

Type Funktion : BOOL

Input IN : BYTE (Zeichen)

Output BOOL (TRUE IN ein Zeichen 0..9 ist)



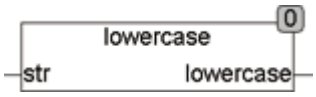
ISC_UPPER testet ob ein Zeichen IN ein Großbuchstabe ist, Ist IN ein Großbuchstabe gibt die Funktion TRUE zurück, wenn nicht gibt die Funktion FALSE zurück. Bei der Prüfung wird die Globale Setup Konstante EXTENDED_ASCII berücksichtigt. Wenn EXTENDED_ASCII = TRUE ist werden Zeichen des erweiterten ASCII Zeichensatzes nach ISO 8859-1 ausgewertet.

Die folgende Tabelle Erläutert die Zeichencodes:

Code	EXTENDED_ASCII=TRUE	EXTENDED_ASCII = FASLE
0..64,91..191,215, 223..255	FALSE	FALSE
65..90	TRUE	TRUE
192..214	TRUE	FALSE
216..222	TRUE	FALSE

13.55. LOWERCASE

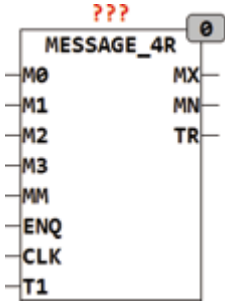
Type	Funktion : STRING
Input	STR : STRING (Eingabestring)
Output	STRING (STRING in Kleinbuchstaben)



Die Funktion LOWERCASE wandelt den String STR in Kleinbuchstaben um. Bei der Konvertierung wird die Globale Setup Konstante EXTENDED_ASCII berücksichtigt. Wenn EXTENDED_ASCII = TRUE werden Zeichen des erweiterten ASCII Zeichensatzes nach ISO 8859-1 ausgewertet. Umlaute wie Ä,Ö,Ü werden nur dann berücksichtigt wenn die Globale Konstante EXTENDED_ASCII = TRUE ist. Eine Detaillierte Beschreibung der Codewandlung ist bei der Funktion TO_LOWER zu finden.

13.56. MESSAGE_4R

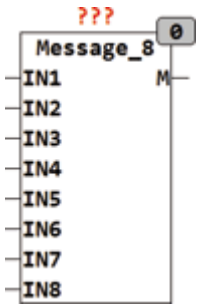
Type	Funktionsbaustein
Input	M0 :. M3 STRING(STRING_LENGTH) (Meldungen)
	MM : INT (Meldung die maximal angezeigt wird)
	ENQ : BOOL (Freigabe Eingang)
	CLK : BOOL (Eingang zum weiter Schalten)
	T1 : TIME (Zeit für automatisches Weiterschalten)
Output	MX : STRING(STRING_LENGTH) (Ausgabestring)
	MN : INT (derzeit aktive Meldung)
	TR : BOOL (Trigger Ausgang)



MESSAGE_4R stellt am Ausgang MX eine von bis zu 4 Meldungen bereit. Es wird immer nur eine von bis zu 4 Meldungen auf MX bereitgestellt. Die Anzahl der Meldungen kann mit dem Eingang MM begrenzt werden. Wird MM auf 2 gesetzt so werden nur die Meldungen M0 .. M2 hintereinander ausgegeben. Wird MM nicht gesetzt so werden alle Meldungen M0..M3 ausgegeben. Mit jeder steigenden Flanke von CLK wird die nächste Meldung auf MX ausgegeben, bleibt CLK dauerhaft auf TRUE so wird nach Ablauf der Zeit T1 automatisch die nächste Meldung ausgegeben, solange bis CLK wieder FALSE wird. Wird der Freigabeeingang ENQ auf FALSE gesetzt, wird am Ausgang MX '' ausgegeben und der Baustein hat keinerlei Funktion. Der Ausgang MN zeigt an welche Meldung gerade am Ausgang MX ausgegeben wird. Der Ausgang TR wird immer dann für genau einen Zyklus TRUE wenn sich die Meldung am Ausgang MX verändert hat, er dient vor allem Dazu Bausteine zur Weiterverarbeitung der Meldungen anzusteuern.

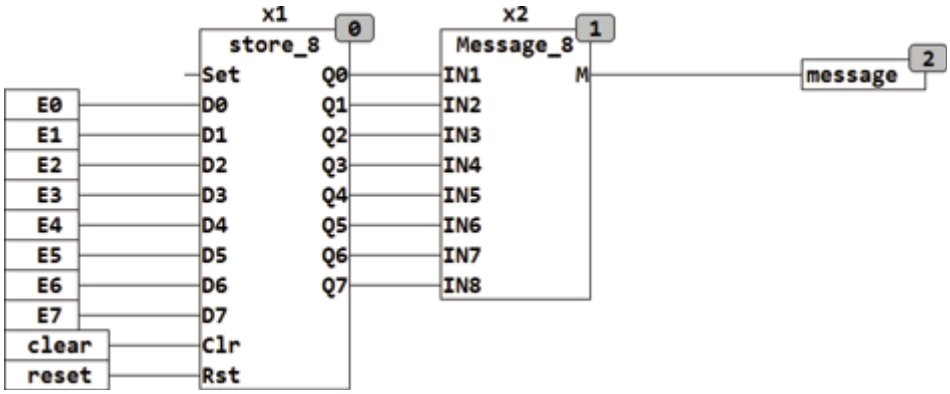
13.57. MESSAGE_8

Type	Funktionsbaustein
Input	IN1..IN8 : BOOL (Auswahleingänge)
Setup	S1..S8 : STRING (Nachrichtenvorgabe)
Output	M : STRING (Ausgangsstring)



MESSAGE_8 erzeugt eine von 8 Nachrichten am Ausgang M. wenn keiner der Eingänge IN1..IN8 auf TRUE ist wird an M ein Leere Zeichenkette ausgegeben, ansonsten einer der in S1..S8 gespeicherten Nachrichten. Der Baustein gibt immer die Nachricht mit der höchsten Priorität aus. IN1 hat dabei die höchste Priorität und IN8 die niedrigste. MESSAGE_8 kann zusammen mit dem Baustein STORE_8 benutzt werden um Fehlerereignisse zu speichern und Anzuzeigen.

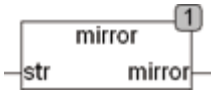
Im folgenden Beispiel werden bis zu 8 Fehlerereignisse (E0..E7)



gespeichert und jeweils die am höchsten priorisierte am Ausgang M von MESSAGE_8 angezeigt. Mit dem Eingang CLEAR kann man durch Tasten jeweils die letzte Meldung löschen und zur nächsten anstehenden Fehlermeldung Weiterschalten. MIT dem Eingang RESET kann man alle anstehenden Fehlermeldungen löschen.

13.58. MIRROR

Type	Funktion : STRING
Input	STR : STRING (Eingangsstring)
Output	STRING (Eingangsstring Rückwärts gelesen)

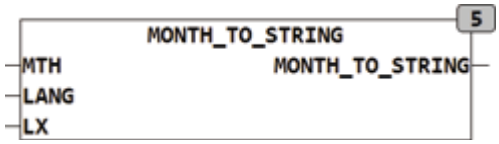


MIRROR liest die Zeichenkette an STR Rückwärts und gibt die Zeichen in umgekehrter Reihenfolge wieder aus.

Beispiel: MIRROR('Das ist ein Test') = 'tseT nie tsi saD'

13.59. MONTH_TO_STRING

Type	Funktion : STRING(10)
Input	MTH : INT (Monat 1..12)
	LANG : INT (Sprachauswahl 0 = Default)
	LX : INT (Länge der Zeichenkette)
Output	STRING(10) (Ausgangswert)



MONTH_TO_STRING wandelt eine Monatszahl in die entsprechende Zeichenkette. Der Eingang MTH gibt den entsprechenden Monat an: 1 = Januar und 12 = Dezember. Der Eingang LANG wählt die gewünschte Sprache aus: 1 = Englisch und 2 = Deutsch. LANG = 0 benutzt die als Default Sprache in der Globalen Setup Variable LANGUAGE_DEFAULT festgelegte Sprache. Der Eingang LX legt die Länge der zu erzeugenden Zeichenkette fest: 0 = Voller Monatsname, 3 = Abkürzung mit 3 Buchstaben, alle anderen Werte am Eingang LX sind nicht definiert.

Die vom Baustein erzeugten Zeichenketten sowie die unterstützten Sprachen sind im Bereich Global Constants definiert und können dort erweitert und verändert werden.

MONTH_TO_STRING(1,0,0) = 'January'

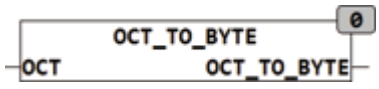
abhängig von der Globalen Konstante LANGUAGE_DEFAULT

MONTH_TO_STRING(1,2,0) = 'Januar'

MONTH_TO_STRING(1,2,3) = 'Jan'

13.60. OCT_TO_BYTE

Type	Funktion : BYTE
Input	OCT : STRING(10) (Oktale Zeichenkette)
Output	BYTE (Ausgangswert)



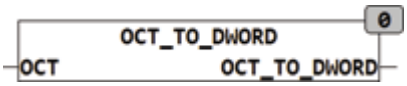
Die Funktion OCT_TO_BYTE konvertiert eine oktal kodierte Zeichenkette in einen BYTE Wert. Es werden dabei nur die oktalen Zeichen sind '0'..'7' interpretiert, alle anderen in HEX vorkommenden Zeichen werden ignoriert.

Beispiel: OCT_TO_BYTE('11') ergibt 9.

13.61. OCT_TO_DWORD

Type	Funktion : DWORD
Input	OCT : STRING(20) (Oktale Zeichenkette)

Output DWORD (Ausgangswert)



Die Funktion OCT_TO_DWORD konvertiert eine oktal kodierte Zeichenkette in einen BYTE Wert. Es werden dabei nur die oktalen Zeichen sind '0'..'7' interpretiert, alle anderen in HEX vorkommenden Zeichen werden ignoriert.

Beispiel: OCT_TO_DWORD('11') ergibt 9.

13.62. REAL_TO_STRF

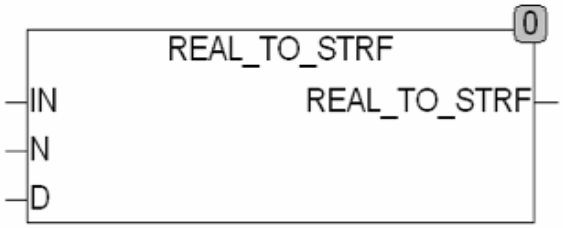
Type Funktion : STRING(20)

Input IN : REAL (Eingangswert)

N : INT (Anzahl der Nachkommastellen)

D : STRING(1) (Dezimalpunktzeichen)

Output STRING (Ausgangsstring)



REAL_TO_STRF konvertiert einen REAL-Wert in einen STRING mit einer festen Anzahl von Nachkommastellen N. Bei der Konvertierung wird ausschließlich in Normales Zahlenformat XXXdNNN umgewandelt. Bei der Umwandlung wird IN auf N Stellen nach dem Komma gerundet und dann in einen String mit dem Format XXXdNNN gewandelt. Wenn N = 0 wird die REAL Zahl auf 0 Stellen hinter dem Komma gerundet und das Ergebnis als Integer ohne Punkt und Nachkommastellen ausgegeben. Wenn die Zahl IN kleiner ist als mit N Nachkommastellen erfasst werden können wird eine Null ausgegeben. Die Nachkommastellen werden immer auf N Stellen mit Nullen aufgefüllt. Die maximale Länge der Zeichenkette beträgt 20 Stellen. Der Eingang D legt fest mit welchem Zeichen der Dezimalpunkt dargestellt wird.

Beispiele:

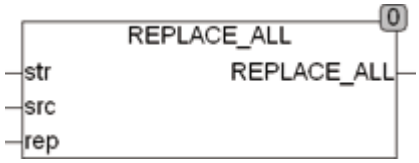
REAL_TO_STRF(3.14159,4,'.') = '3.1416'

REAL_TO_STRF(3.14159,0,'.') = '3'

REAL_TO_STRF(0.04159,3,'.') = '0.042'
REAL_TO_STRF(0.001,2',') = '0,00'

13.63. REPLACE_ALL

Type Funktion : STRING
Input STR : STRING (Eingangs String)
SRC : STRING (Suchstring)
REP : STRING (Ersatzstring)
Output STRING (Ausgangsstring)

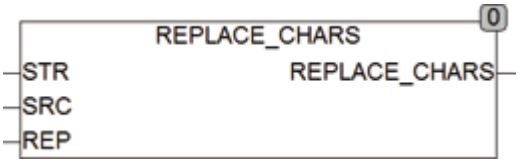


REPLACE_ALL ersetzt alle in der Zeichenkette STR vorkommenden Strings SRC mit REP. Eine leere Zeichenkette an SRC ergibt keine Suchergebnisse.

Beispiel: REPLACE_ALL('123BB456BB789BB','BB','/') = '123/456/789/'
REPLACE_ALL('123BB456BB789BB','BB','') = '123456789'

13.64. REPLACE_CHARS

Type Funktion : STRING
Input STR : STRING (Eingangsstring)
SRC : STRING (Suchzeichen)
REP : STRING (Ersatzzeichen)
Output STRING (Ausgangsstring)



REPLACE_CHARS ersetzt alle Zeichen in SRC die in STR vorkommen mit den Zeichen an der gleichen Stelle in REP.

Beispiel: REPLACE_CHARS('abc123', '0123456789', ABCDEFGHIJ') = 'abcABC'

13.65. REPLACE_UML

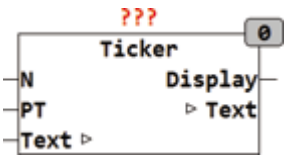
Type	Funktion : STRING
Input	STR : STRING (Eingangsstring)
Output	STRING (Ausgangsstring)

REPLACE_UML ersetzt Umlaute mit einer Kombination aus 2 Zeichen so dass das Ergebnis keine Umlaute mehr enthält. Die Groß und Kleinschreibung wird dabei beachtet. Wenn ein Wort nur aus Großbuchstaben besteht und darin ein Umlaut enthalten ist wird dieser mit einem Großbuchstaben gefolgt von einem Kleinbuchstaben ersetzt, im Falle eines ß welches keine Großschreibung kennt wird es immer mit zwei Kleinbuchstaben ersetzt. Wird die Funktion REPLACE_UML auf ein Wort das nur aus Großbuchstaben besteht angewendet muss anschließend mit der Funktion UPPERCASE() sichergestellt werden dass die Kleinbuchstaben wieder in Großbuchstaben gewandelt werden.

Ä > Ae, Ö > Oe, Ü > Ue, ä > ae, ö > oe, ü > oe, ß > ss.

13.66. TICKER

Type	Funktionsbaustein
Input	N : INT (Länge des Display Strings)
	PT : TIME (Schiebedelay, Default = T#1s)
I/O	TEXT : STRING (Eingangsstring)
Output	DISPLAY : STRING (Ausgangsstring)



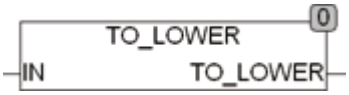
TICKER erzeugt am Ausgang DISPLAY eine Laufschrift. Am Ausgang DISPLAY wird ein Teilstring von TEXT mit der Länge N ausgegeben. Display wird in einem Zeitraster von PT ausgegeben und beginnt bei jeder Ausgabe eine Stelle weiter von Links des Eingangsstrings TEXT. Die Laufschrift wird nur dann erzeugt wenn N < als die Länge von TEXT ist. Wird N >= Länge von TEXT dann wird der String TEXT direkt am Ausgang DISPLAY dargestellt.

13.67. TO_LOWER

Type Funktion : BYTE

Input IN : BYTE (Zeichen das konvertiert werden soll)

Output BYTE (konvertiertes Zeichen)



TO_LOWER wandelt einzelne Zeichen in Kleinbuchstaben um. Bei der Konvertierung wird die Globale Setup Konstante EXTENDED_ASCII berücksichtigt. Wenn EXTENDED_ASCII = TRUE werden Zeichen des erweiterten ASCII Zeichensatzes nach ISO 8859-1 berücksichtigt.

Die folgende Tabelle Erläutert die Codewandlung:

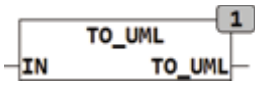
Code	EXTENDED_ASCII = TRUE	EXTENDED_ASCII = FALSE
0..64	0..64	0..64
65..90	97..122	97..122
91..191	91..191	91..191
192..214	224..246	192..214
215	215	215
216..222	248..254	216..254
223..255	223..255	223..255

13.68. TO_UML

Type Funktion : STRING(2)

Input IN : BYTE (Zeichen das konvertiert werden soll)

Output STRING(2) (konvertiertes Zeichen)



TO_UML wandelt einzelne Sonderzeichen des Zeichensatzes größer 127 in eine Kombination zweier Buchstaben um. Es handelt sich dabei um Zeichen des erweiterten ASCII Zeichensatzes nach ISO 8859-1(Latin1).

Es werden folgende Zeichen umgewandelt:

Ä >> Ae ä >> ae Ö >> Oe ö >> oe Ü >> Ue ü >> ue

ß >> ss

Alle andern Zeichen werden als STRING mit dem Zeichen IN zurückgegeben.

13.69. TO_UPPER

Type	Funktion : BYTE
Input	IN : BYTE (Zeichen das konvertiert werden soll)
Output	BYTE (konvertiertes Zeichen)

TO_UPPER wandelt einzelne Zeichen in Großbuchstaben um. Bei der Konvertierung wird die Globale Setup Konstante EXTENDED_ASCII berücksichtigt. Wenn EXTENDED_ASCII = TRUE werden Zeichen des erweiterten ASCII Zeichensatzes nach ISO 8859-1 berücksichtigt.

Die folgende Tabelle Erläutert die Codewandlung:

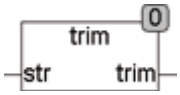
Code	EXTENDED_ASCII = TRUE	EXTENDED_ASCII = FALSE
0..64	0..64	0..64
65..90	97..122	97..122
91..191	91..191	91..191
192..214	224..246	192..214
215	215	215
216..222	248..254	216..254
223..255	223..255	223..255

13.70. TRIM

Type Funktion : STRING

Input STR : STRING (Eingabestring)

Output STRING (STR ohne Leerzeichen)



Die Funktion TRIM entfernt alle Leerzeichen aus STR.

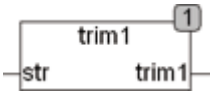
Beispiel: TRIM('find BX12') = 'findBX12'

13.71. TRIM1

Type Funktion : STRING

Input STR : STRING (Eingabestring)

Output STRING (STR ohne doppelte Leerzeichen)



Die Funktion TRIM1 ersetzt mehrfache Leerzeichen mit nur einem Leerzeichen. Leerzeichen am Anfang und Am Ende von STR werden dabei komplett gelöscht.

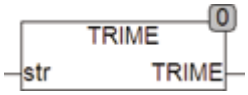
Beispiel: TRIM1(' find BX12 ') = 'find BX12'

13.72. TRIME

Type Funktion : STRING

Input STR : STRING (Eingabestring)

Output STRING (Ausgabestring)



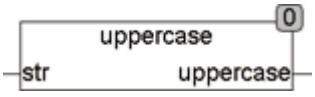
Die Funktion TRIME entfernt Leerzeichen am Anfang und Am Ende von STR. Leerzeichen innerhalb des Strings werden ignoriert, auch wenn Sie mehrfach vorkommen.

13.73. UPPERCASE

Type Funktion : STRING

Input STR : STRING (Eingabestring)

Output STRING (STRING in Großbuchstaben)

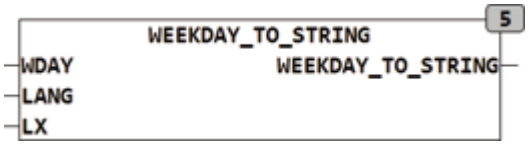


Die Funktion UPPERCASE wandelt alle Buchstaben in STR in Großbuchstaben um. Bei der Konvertierung wird die Globale Setup Konstante EXTENDED_ASCII berücksichtigt. Wenn EXTENDED_ASCII = TRUE werden Zeichen des erweiterten ASCII Zeichensatzes nach ISO 8859-1 berücksichtigt. Umlaute wie Ä,Ö,Ü werden nur dann berücksichtigt wenn die Globale Konstante EXTENDED_ASCII = TRUE ist. Eine Detaillierte Beschreibung der Codewandlung ist bei der Funktion TO_UPPER zu finden.

Beispiel: UPPERCASE('find BX12') = FIND BX12

13.74. WEEKDAY_TO_STRING

Type	Funktion : STRING(10)
Input	WDAY : INT (Wochentag 1..7)
	LANG : INT (Sprachauswahl 0 = Default)
	LX : INT (Länge der Zeichenkette)
Output	STRING(10) (Ausgangswert)



WEEKDAY_TO_STRING wandelt einen Wochentag in die entsprechende Zeichenkette. Der Eingang WDAY gibt den entsprechenden Wochentag an: 1 = Montag und 7 = Sonntag. Der Eingang LANG wählt die gewünschte Sprache aus: 1 = Englisch und 2 = Deutsch. LANG = 0 benutzt die als Default Sprache in der Globalen Setup Variable LANGUAGE_DEFAULT festgelegte Sprache. Der Eingang LX legt die Länge der zu erzeugenden Zeichenkette fest: 0 = Voller Monatsname, 2 = Abkürzung mit 2 Buchstaben, alle anderen Werte am Eingang LX sind nicht definiert.

Die vom Baustein erzeugten Zeichenketten sowie die unterstützten Sprachen sind im Bereich Global Constants definiert und können dort erweitert und verändert werden.

WEEKDAY_TO_STRING(1,0,0) = 'Monday'

abhängig von der Globalen Konstante LANGUAGE_DEFAULT

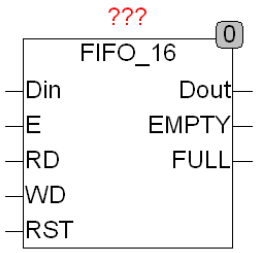
WEEKDAY_TO_STRING(1,2,0) = 'Montag'

WEEKDAY_TO_STRING(1,0,2) = 'Mo'

14. Speicherbausteine

14.1. FIFO_16

Type	Funktionsbaustein
Input	DIN : DWORD (Daten Eingang)
	E : BOOL (Freigabe Eingang)
	RD : BOOL (Lese Kommando)
	WD : BOOL (Schreib Kommando)
	RST : BOOL (Reset Eingang)
Output	DOUT : DWORD (Daten Ausgang)
	EMPTY : BOOL (EMPTY = TRUE bedeutet: Speicher ist Leer)
	FULL : BOOL (FULL = TRUE bedeutet: Speicher ist Voll)



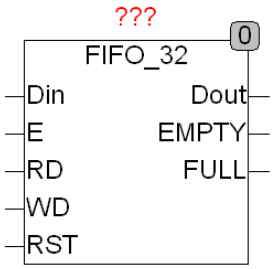
FIFO_16 ist ein First-In First-Out Speicher mit 16 Speicherstellen für DWORD Daten. Die beiden Ausgänge EMPTY und FULL zeigen an, wann der Speicher voll oder leer ist. Der Eingang RST löscht den gesamten Inhalt des Speichers. Der FIFO wird mit DIN beschrieben, indem ein TRUE auf den Eingang WD, und ein TRUE-Puls auf den Eingang E gegeben werden. Ein Lesebefehl wird ausgeführt indem TRUE auf RD und TRUE auf E gelegt wird. Lesen und Schreiben kann gleichzeitig in einem Zyklus ausgeführt werden. Der Baustein Liest oder Schreibt in jedem Zyklus solange das entsprechende Kommando (RD, WD) auf TRUE steht.

14.2. FIFO_32

Type	Funktionsbaustein
Input	DIN : DWORD (Daten Eingang)
	E : BOOL (Freigabe Eingang)

RD : BOOL (Lese Kommando)
WD : BOOL (Schreib Kommando)
RST : BOOL (Reset Eingang)

Output DOUT : DWORD (Daten Ausgang)
EMPTY : BOOL (EMPTY = TRUE bedeutet: Speicher ist Leer)
FULL : BOOL (FULL = TRUE bedeutet: Speicher ist Voll)



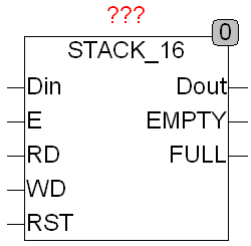
FIFO_32 ist ein First-In First-Out Speicher mit 32 Speicherstellen für DWORD Daten. Die beiden Ausgänge EMPTY und FULL zeigen an, wann der Speicher voll oder leer ist. Der Eingang RST löscht den gesamten Inhalt des Speichers. Der FIFO wird mit DIN beschrieben, indem ein TRUE auf den Eingang WD, und ein TRUE-Puls auf den Eingang E gegeben werden. Ein Lesebefehl wird ausgeführt indem TRUE auf RD und TRUE auf E gelegt wird. Lesen und Schreiben kann gleichzeitig in einem Zyklus ausgeführt werden. Der Baustein Liest oder Schreibt in jedem Zyklus solange das entsprechende Kommando (RD, WD) auf TRUE steht.

14.3. STACK_16

Type Funktionsbaustein

Input DIN : DWORD (Daten Eingang)
E : BOOL (Freigabe Eingang)
RD : BOOL (Lese Kommando)
WD : BOOL (Schreib Kommando)
RST : BOOL (Reset Eingang)

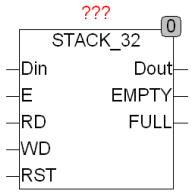
Output DOUT : DWORD (Daten Ausgang)
EMPTY : BOOL (EMPTY = TRUE bedeutet: Speicher ist Leer)
FULL : BOOL (FULL = TRUE bedeutet: Speicher ist Voll)



STACK_16 ist ein Stapelspeicher (STACK) mit 16 Speicherstellen für DWORD Daten. Die beiden Ausgänge EMPTY und FULL zeigen an, wann der Speicher voll oder leer ist. Der Eingang RST löscht den gesamten Inhalt des Speichers. Der FIFO wird mit DIN beschrieben, indem ein TRUE auf den Eingang WD, und ein TRUE auf den Eingang E gegeben werden. Ein Lesebefehl wird ausgeführt indem TRUE auf RD und TRUE auf E gelegt wird. Lesen und Schreiben kann gleichzeitig in einem Zyklus ausgeführt werden. Der Baustein Liest oder Schreibt in jedem Zyklus solange das entsprechende Kommando (RD, WD) auf TRUE steht.

14.4. STACK_32

Type	Funktionsbaustein
Input	DIN : DWORD (Daten Eingang)
	E : BOOL (Freigabe Eingang)
	RD : BOOL (Lese Kommando)
	WD : BOOL (Schreib Kommando)
	RST : BOOL (Reset Eingang)
Output	DOUT : DWORD (Daten Ausgang)
	EMPTY : BOOL (EMPTY = TRUE bedeutet: Speicher ist Leer)
	FULL : BOOL (FULL = TRUE bedeutet: Speicher ist Voll)



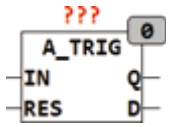
STACK_32 ist ein Stapelspeicher (STACK) mit 32 Speicherstellen für DWORD Daten. Die beiden Ausgänge EMPTY und FULL zeigen an, wann der Speicher voll oder leer ist. Der Eingang RST löscht den gesamten Inhalt des Speichers. Der FIFO wird mit DIN beschrieben, indem ein TRUE auf den Eingang WD, und ein TRUE auf den Eingang E gegeben werden. Ein Lesebefehl wird ausgeführt indem TRUE auf RD und TRUE auf E gelegt

wird. Lesen und Schreiben kann gleichzeitig in einem Zyklus ausgeführt werden. Der Baustein Liest oder Schreibt in jedem Zyklus solange das entsprechende Kommando (RD, WD) auf TRUE steht.

15. Takt Generatoren

15.1. A_TRIG

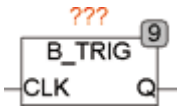
Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	RES : REAL (Auflösung für Eingangsveränderung)
Output	Q : BOOL (Ausgangssignal)
	D : REAL (letzte Veränderung des Eingangssignals)



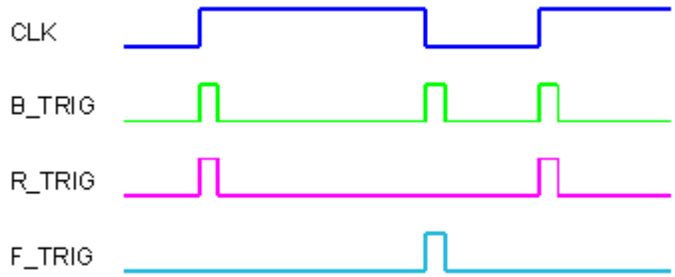
A_TRIG überwacht einen Eingangswert auf Veränderung und immer wenn der Eingangswert sich um mehr als RES verändert hat erzeugt der Baustein einen Ausgangsimpuls für einen Zyklus damit der neue Wert verarbeitet werden kann. Gleichzeitig merkt sich der Baustein den aktuellen Eingangswert mit denen er dann in den nächsten Zyklen den Eingang IN vergleicht. Am Ausgang D wird die Differenz zwischen dem gespeicherten Wert und IN angezeigt.

15.2. B_TRIG

Type	Funktionsbaustein
Input	CLK : BOOL (Eingangssignal)
Output	Q : BOOL (Ausgangssignal)

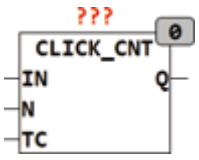


Der Funktionsbaustein B_TRIG erzeugt nach einem Flankenwechsel am Eingang CLK einen Ausgangsimpuls für exakt einen SPS-Zyklus. Im Gegensatz zu den beiden Standardfunktionsbausteinen R_TRIG und F_TRIG, die jeweils nur bei fallender oder steigender Flanke einen Puls erzeugen, erzeugt B_TRIG bei fallender und steigender Flanke einen Ausgangspuls.

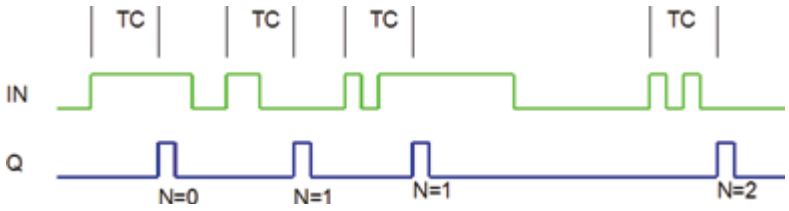


15.3. CLICK_CNT

Type	Funktionsbaustein
Input	IN : BOOL (Eingangssignal)
	N : INT (Anzahl der zu dekodierenden Clicks)
	TC : TIME (Zeit in der die Clicks stattfinden müssen)
Output	Q : BOOL (Ausgangssignal)

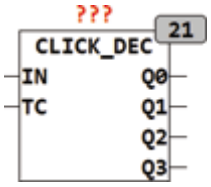


CLICK_CNT ermittelt die Anzahl der Impulse innerhalb der Zeiteinheit TC.am Eingang IN. Eine steigende Flanke an IN startet einen internen Timer mit der Zeit TC. Während des Ablaufs des Timers zählt der Baustein die fallenden Flanken an IN und überprüft nach Ablauf der Zeit TC ob N Impulse innerhalb der Zeit TC stattgefunden haben. Nur wenn exakt N Impulse innerhalb TC vorkommen wird der Ausgang Q für einen SPS Zyklus TRUE gesetzt. Der Baustein decodiert auch N=0 was einer steigenden Flanke aber keiner fallenden Flanke innerhalb von TC entspricht.

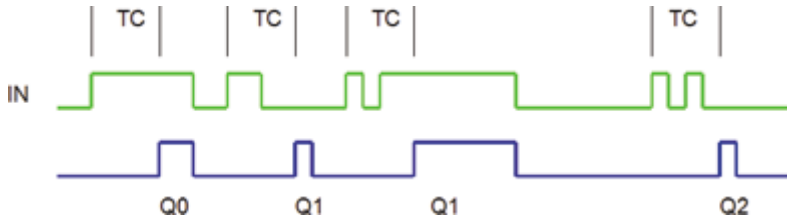


15.4. CLICK_DEC

Type	Funktionsbaustein
Input	IN : BOOL (Eingangssignal)
	TC : TIME (Zeit in der die Clicks stattfinden müssen)
Output	Q0 : BOOL (Ausgangssignal für steigende Flanke ohne fallende Flanke)
	Q1 : BOOL (Ausgangssignal für einen Impuls innerhalb TC)
	Q2 : BOOL (Ausgangssignal für 2 Impulse innerhalb TC)
	Q3 : BOOL (Ausgangssignal für 3 Impulse innerhalb TC)



CLICK_DEC dekodiert mehrfach Tastendrücke und signalisiert an verschiedenen Ausgängen die Anzahl der Impulse. Ein Eingangssignal ohne fallende Flanke innerhalb TC wird an Q0 ausgegeben und bleibt solange TRUE bis IN auf FALSE geht. Ein Impuls gefolgt von einem TRUE wird auf Q1 ausgegeben, usw. Wird innerhalb TC ein Impuls registriert, der vom Zustand FALSE gefolgt wird, so erscheint am entsprechenden Ausgang für einen SPS Zyklus ein TRUE.



15.5. CLK_DIV

Type	Funktionsbaustein
Input	CLK : BOOL (Clock Eingang)
	RST : BOOL (Reset Eingang)
Output	Q0 : BOOL (Teiler Ausgang CLK / 2)

Q1 : BOOL (Teiler Ausgang CLK / 4)

Q2 : BOOL (Teiler Ausgang CLK / 8)

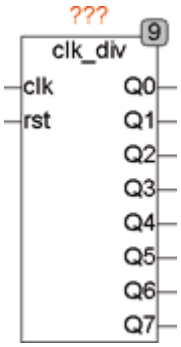
Q3 : BOOL (Teiler Ausgang CLK / 16)

Q4 : BOOL (Teiler Ausgang CLK / 32)

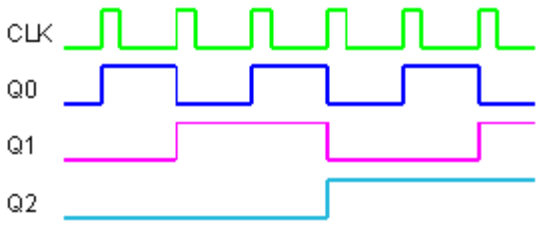
Q5 : BOOL (Teiler Ausgang CLK / 64)

Q6 : BOOL (Teiler Ausgang CLK / 128)

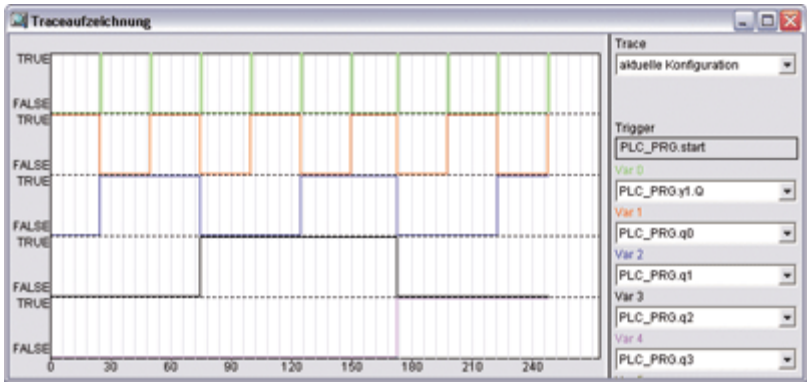
Q7 : BOOL (Teiler Ausgang CLK / 256)



Der Funktionsbaustein CLK_DIV ist ein Teilerbaustein, der ein Eingangssignal CLK in 8 Stufen durch jeweils 2 teilt, sodass am Ausgang Q0 die halbe Frequenz des Eingangs CLK mit 50% Tastverhältnis zur Verfügung steht. Der Ausgang Q1 stellt die halbierte Frequenz von Q0 zur Verfügung und so weiter, bis an Q7 die Eingangsfrequenz geteilt durch 256 bereitsteht. Ein Reset Eingang RST setzt asynchron alle Ausgänge auf FALSE. CLK darf jeweils nur einen Zyklus auf TRUE sein, falls CLK dies nicht tut muss CLK über ein TP_R bereitgestellt werden.

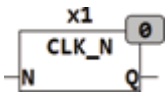


Das folgende Beispiel ist eine Testschaltung mit Startsignal über ENI / ENO Funktionalität Realisiert. Bild 2 zeigt eine entsprechende Traceaufzeichnung der Schaltung:



15.6. CLK_N

Type	Funktionsbaustein
Input	N : INT (Clock Teiler)
Output	Q : BOOL (Taktausgang)

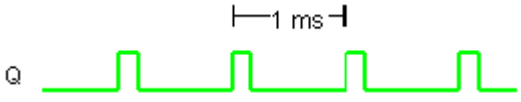


CLK_N erzeugt einen Impuls alle X Millisekunden basierend auf der SPS internen 1ms Referenz. Die Impulse sind exakt einen SPS Zyklus lang und werden alle 2^N Millisekunden erzeugt.

Die Periodendauer beträgt 1ms für N=0, 2ms für N=1, 4ms für N=2 usw.

CLK_N ersetzt die Bausteine CLK_1ms, CLK_2ms, CLK_4ms und CLK_8ms aus älteren Bibliotheken.

Das nachfolgende Bild zeigt das Ausgangssignal für N=0:

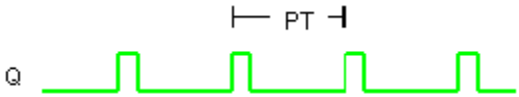


15.7. CLK_PRG

Type	Funktionsbaustein
Input	PT : TIME (Zykluszeit)

Output Q : BOOL (Taktausgang)

CLK_PRG erzeugt Taktimpulse mit einer programmierbaren Periodendauer PT. Die Ausgangsimpulse sind jeweils nur einen SPS Zyklus.



15.8. CLK_PULSE

Type Funktionsbaustein

Input PT : TIME (Zykluszeit)

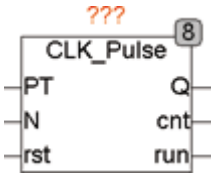
N : INT (Anzahl der zu erzeugenden Impulse)

RST : BOOL (Reset)

Output Q : BOOL (Taktausgang)

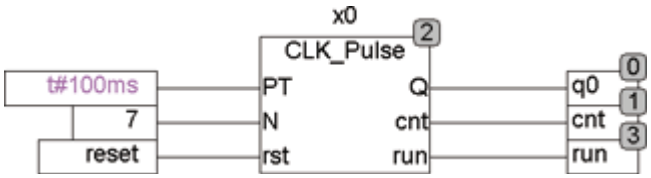
CNT : INT (Zähler der Ausgangsimpulse)

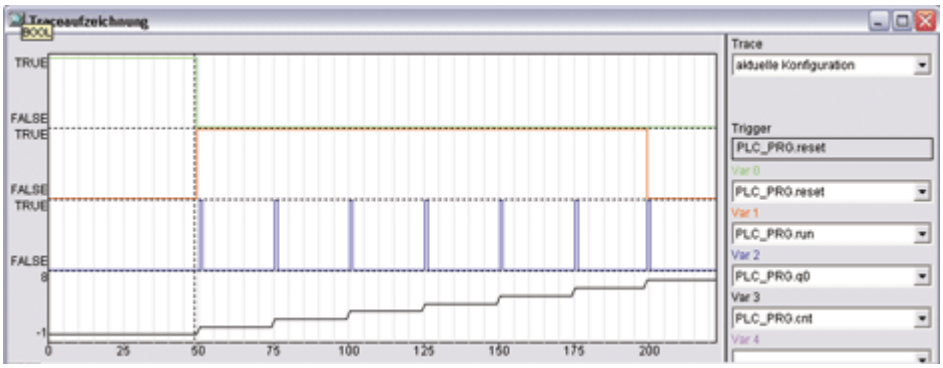
RUN : BOOL (TRUE, wenn Pulsgenerator läuft)



CLK_PULSE erzeugt eine definierte Anzahl von Taktimpulsen mit einem programmierbaren Tastverhältnis. PT legt das Tastverhältnis fest und N die Anzahl der zu erzeugenden Impulse. Durch einen Reset Eingang RST kann der Generator jederzeit erneut gestartet werden. Der Ausgang CNT zählt die erzeugten Impulse und RUN = TRUE zeigt an, dass der Generator noch Impulse generiert. Ein Eingangswert N = 0 erzeugt eine unendliche Impulsfolge, Die Maximale Anzahl der Impulse ist auf 32767 begrenzt.

Das folgende Beispiel zeigt eine Anwendung von CLK_PULSE zur Erzeugung von 7 Impulsen mit einem Tastverhältnis von 100ms.

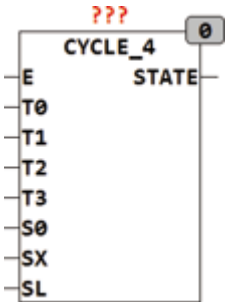




Die Traceaufzeichnung zeigt, wie der RESET (Grün) inaktiv wird und dadurch RUN (Rot) aktiv wird. Der Generator erzeugt dann 7 Impulse (Blau), wie am Eingang N spezifiziert. Der Ausgang CNT zählt dabei von 1 beim ersten Puls nach 7 beim letzten Puls. Nach Ablauf der Sequenz wird RUN wieder inaktiv und der Zyklus ist beendet, bis er durch einem neuen Reset wieder gestartet wird.

15.9. CYCLE_4

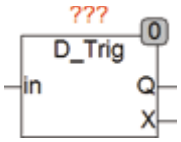
Type	Funktionsbaustein
Input	E : BOOL (Enable Eingang)
	T0 .. T3 : TIME (Laufzeit der einzelnen States)
	S0 : BOOL (kontinuierlicher Zyklus Enable)
	SX : INT (State wenn SL = TRUE)
	SL : BOOL (asynchroner Load Eingang)
Output	STATE : INT (Statusausgang)



CYCLE_4 erzeugt wenn E = TRUE die States 0..3. Die Dauer jedes einzelnen States kann durch die Zeitvorgaben T0..T3 festgelegt werden. Der Eingang SL startet wenn TRUE ab einem vorgegebenen STATE SX. Der Eingang E hat den internen Default TRUE, so dass er auch offen gelassen werden kann. Nach einer Steigenden Flanke an E startet der Baustein immer mit STATE = 0, und wenn E = FALSE bleibt der Ausgang STATE auf 0. Mit dem Eingang S0 wird der Zyklische Modus eingeschaltet, Wenn S0 = FALSE stoppt der Baustein bei State = 3, ist S0 = TRUE so beginnt der Baustein nach State 3 wieder mit State 0.

15.10. D_TRIG

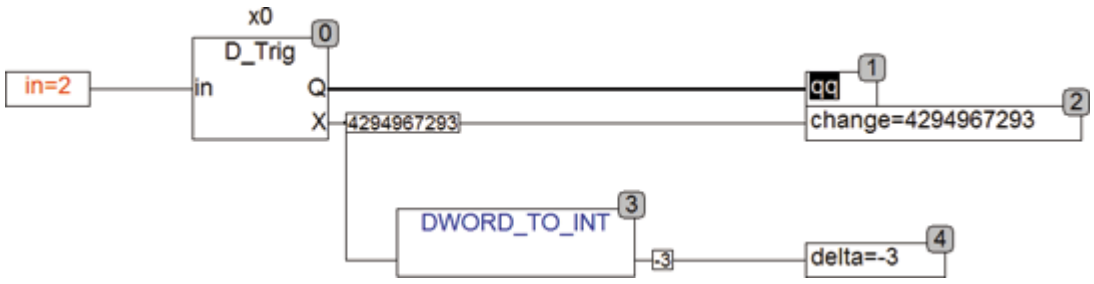
Type	Funktionsbaustein
Input	IN : DWORD (Eingangssignal)
Output	Q : BOOL (Ausgangssignal)
	X : DWORD (Veränderung des Eingangssignals)



Der Funktionsbaustein D_TRIG erzeugt nach einer Veränderung am Eingang IN einen Ausgangsimpuls für exakt einen SPS Zyklus. Der Baustein funktioniert vergleichbar mit den Standardfunktionsbausteinen R_TRIG und F_TRIG und dem in der OSCAT Bibliothek enthaltenen Baustein B_TRIG. Während B_TRIG, R_TRIG und F_TRIG einen Booleschen Eingang überwachen, triggert der Baustein D_TRIG auf jede Veränderung des DWORD-Eingangs IN. Wenn sich der Eingangswert verändert hat, so wird der Ausgang Q für einen SPS Zyklus auf TRUE gesetzt und der Ausgang X gibt an, um welchen Wert sich der Eingang IN verändert hat. Der Eingang, sowie der Ausgang sind von Typ DWORD. Der Eingang kann auch WORD und BYTE Typen verarbeiten. Beim Ausgang X ist zu beachten, dass DWORD kein Vorzeichen hat und deshalb eine Veränderung um -1 am Eingang nicht -1 sondern die Zahl 2^32-2 ausgibt. Mit der Standardfunktion DWORD_TO_INT kann der Ausgang X in einen Integer

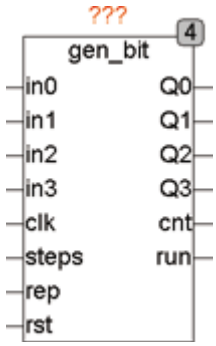
umgewandelt werden, welcher dann auch negative Veränderungen richtig darstellt.

Das nachfolgende Beispiel zeigt die Anwendung von D_TRIG wenn sich der Eingang vom Wert 5 nach 2 verändert:



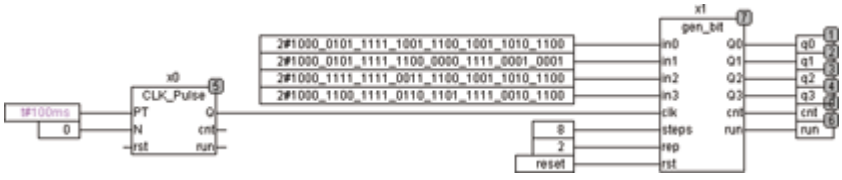
15.11. GEN_BIT

Type	Funktionsbaustein
Input	IN0 : DWORD (Bitsequenz für Q0)
	IN1 : DWORD (Bitsequenz für Q1)
	IN2 : DWORD (Bitsequenz für Q1)
	IN3 : DWORD (Bitsequenz für Q1)
	CLK : BOOL (Clock Eingang)
	STEPS : INT (Anzahl der zu erzeugenden Takte)
	REP : INT
	RST : BOOL
Output	Q0 : BOOL (Bitsequenz Q0)
	Q1 : BOOL (Bitsequenz Q1)
	Q2 : BOOL (Bitsequenz Q2)
	Q3 : BOOL (Bitsequenz Q3)
	CNT : INT (Anzahl der bereits erzeugten Ausgangsbits)
	RUN : BOOL (TRUE, wenn der Sequenzer läuft)

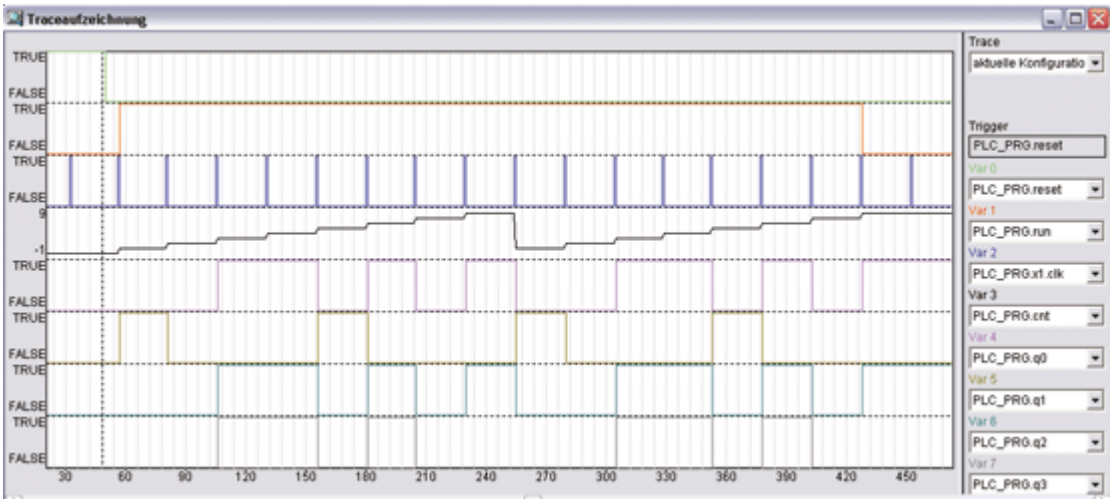


GEN_BIT ist ein frei programmierbarer Bitmustergenerator. An den Eingängen in0 .. in7 liegen die Bitmuster jeweils als DWORD an und werden durch den Eingang CLK je Taktimpuls beginnend von Bit 0 an aufsteigend an die Ausgänge Q0 .. Q3 geschoben. Nach dem ersten Taktimpuls am Eingang CLK liegt am Ausgang Q0 Bit 0 von IN0, an Q1 liegt Bit 0 von In1 ... an Q7 liegt Bit 0 von IN3. Nach dem nächsten Taktimpuls am CLK Eingang wird jeweils das Bit 1 der Eingänge IN an die Ausgänge Q geschoben und so weiter, bis die Sequenz beendet ist. Der Eingang STEPS legt fest, wie viele Bits der Eingangs-DWORDS an die Ausgänge geschoben werden. Der Eingang REP legt fest, wie oft diese Sequenz wiederholt wird. Wird der Eingang auf 0 gesetzt, so wird die Sequenz fortlaufend wiederholt. Ein asynchroner Reset kann jederzeit den Sequenzer zurücksetzen. Die Ausgänge RUN und CNT zeigen an, welches Bit gerade am Ausgang anliegt und ob der Sequenzer läuft, oder die Sequenz (RUN inaktiv) beendet ist. Nachdem die Sequenzen abgelaufen sind bleibt das letzte Bitmuster an den Ausgängen vorhanden, bis ein Reset den Generator neu startet.

Beispiel:



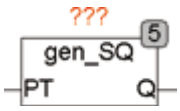
In diesem Beispiel werden die untersten 8 Bits (Bit 0 .. 7) an den Eingängen IN auf die Ausgänge Q geschoben. Die Sequenz beginnt jeweils bei Bit 0 und endet bei Bit 7 (8 Steps sind durch den Eingang 8 definiert). Diese Sequenz wird 2 mal (2 Wiederholungen am Eingang REP) wiederholt und dann gestoppt.



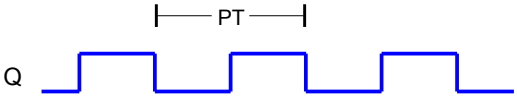
Die Trace Aufzeichnung zeigt das inaktiv werdende Reset Signal (Grün), welches den Generator startet und nach dem ersten Taktimpuls Bit 0 an die Ausgänge schiebt.

15.12. GEN_SQ

Type	Funktionsbaustein
Input	PT : TIME (Periodendauer)
Output	Q : BOOL (Ausgangssignal)



Gen_SQ ist ein Rechteckgenerator mit programmierbarer Periodendauer und einem festen Tastverhältnis von 50%. Der Eingang PT legt die Periodendauer fest und am Ausgang Q steht das Ausgangssignal zur Verfügung.



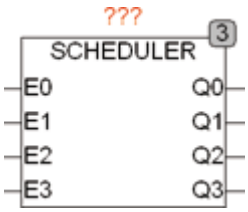
15.13. SCHEDULER

Type	Funktionsbaustein

Input E0..3 : BOOL (Freigabesignale für Q0..3)

Setup T0..3 : TIME (Zykluszeit)

Output Q0..3 : BOOL (Ausgangssignale)



SCHEDULER wird benutzt um Programmteile Zeitabhängig aufzurufen. Z.B. können aufwendige Berechnungen die nur selten gebraucht werden in bestimmten Zeitabständen aufgerufen werden. Die Ausgänge Q* Des Bausteins werden jeweils nur für einen Zyklus aktiv und schalten dadurch die Abarbeitung des entsprechenden Programmteils frei. Die Setup Zeiten T* Legen fest in welchen Zeitabständen die Ausgänge aktiviert werden. SCHEDULER prüft je CPU Zyklus nur einen Ausgang, so dass maximal ein Ausgang je Zyklus aktiv sein kann. Im Extremfall wenn alle Aufrufzeiten T* T#0s sind wird in jedem Zyklus jeweils ein Ausgang gesetzt sein, so dass erst Q0, dann Q1 usw. bis Q3 gesetzt sind um dann wieder mit Q0 zu beginnen. Die Aufrufzeiten können deshalb um bis zu 3 CPU Zyklen vom vorgegebenen Wert T* Abweichen.

15.14. SCHEDULER_2

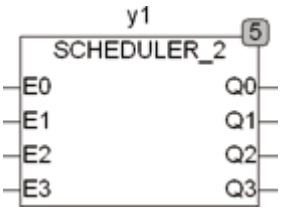
Type Funktionsbaustein

Input E0..3 : BOOL (Freigabesignale für Q0..3)

Setup C0..3 : UINT (Der Ausgang Q? Wird alle C? Zyklen aktiviert)

O0..3 : UINT (Verzögerung für die Ausgänge)

Output Q0..3 : BOOL (Ausgangssignale)

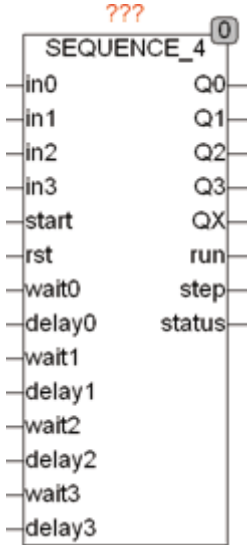


SCHEDULER_2 aktiviert abhängig von den Setup Variablen C? Und O? Die Ausgänge Q?. SCHEDULER_2 kann einen Ausgang Q? Alle C? Zyklen aktivieren um damit Programmteile mit verschiedenen Zykluszeiten zu starten. Ein optionaler Setup Parameter O? Dient dazu einen Zeitversatz von

O? Zyklen für den entsprechenden Ausgang zu definieren um ein gleichzeitiges aktivieren der Ausgänge im ersten Zyklus zu verhindern.

15.15. SEQUENCE_4

Type	Funktionsbaustein
Input	IN0..3 : BOOL (Freigabesignal für Q0..3)
	START : BOOL (Startflanke für den Sequenzer)
	RST : BOOL (Asynchroner Reset-Eingang)
	WAIT 0..3 : TIME (Wartezeit für das Eingangssignal an IN 0..3)
	DELAY 0..3 : TIME (Verzögerungszeit bis das Eingangssignal IN0..3 geprüft wird)
Output	Q 0..3 : BOOL (Steuerausgänge)
	QX : BOOL (TRUE, wenn einer der Ausgänge Q0..Q3 aktiv ist)
	RUN : BOOL (RUN ist TRUE, wenn der Sequenzer läuft)
	STEP : INT (gibt den momentanen Schritt an)
	STATUS : BYTE (zu ESR kompatibler Status-Ausgang)



SEQUENCE_4 ist ein 4 Bit Sequenzer mit Steuereingängen. Nach einer steigenden Flanke an START wird RUN TRUE und der Sequenzer wartet die Zeit Wait0 auf ein TRUE-Signal am Eingang IN0. Nachdem das Signal am IN0 TRUE ist, wird der Ausgang Q0 gesetzt und die Zeit Delay0 abgewartet. Nach Ablauf der Zeit Delay0 wird im nächsten Zyklus für die Zeit wait1 auf ein Eingangssignal an in1 gewartet und Q0 bleibt solange TRUE, bis Q1 gesetzt wird. Das ganze wird solange wiederholt, bis alle 4 Zyklen ab-

gelaufen sind. Falls während den Wartezeiten wait0..3 der entsprechende Eingang nicht TRUE wird, wird ein Fehler gesetzt, indem die entsprechende Error-Nummer am Ausgang STATUS angezeigt wird und abhängig von der Setup-Variable STOP_ON_ERROR wird der Sequenzer angehalten oder nicht. Der STATUS-Ausgang ist 110 für warten auf Startsignal und 111 wenn die Sequenz durchlaufen wird und zeigt mit 1 .. 4 Fehler an. Ein Error = 1 bedeutet, dass das Signal am Eingang in0 nicht aktiv wurde eine 2 entspricht in1 usw.
Die Ausgänge RUN und STEP zeigen an, ob der Sequenzer läuft und in welchem Zyklus er sich gerade befindet. Der Ausgang QX ist TRUE, wenn einer der Ausgänge Q0..Q3 TRUE ist.

Ein asynchroner Reset-Eingang kann den Sequenzer jederzeit zurücksetzen. Dieser Reset-Eingang kann auch mit einem der Ausgänge Q0..Q3 beschaltet werden um den Sequenzer vor dem vollen Ablauf zu stoppen. Der Sequenzer kann auch jederzeit mit einer steigenden Flanke am Eingang START wieder von neuem gestartet werden. Das gilt auch, wenn er eine Sequenz noch nicht beendet hat.

Sollte eine Flankenprüfung an einem oder mehreren Eingängen IN nicht nötig sein, so können sie einfach offen gelassen werden, denn der Vorgabe-Wert für diese Eingänge ist TRUE.

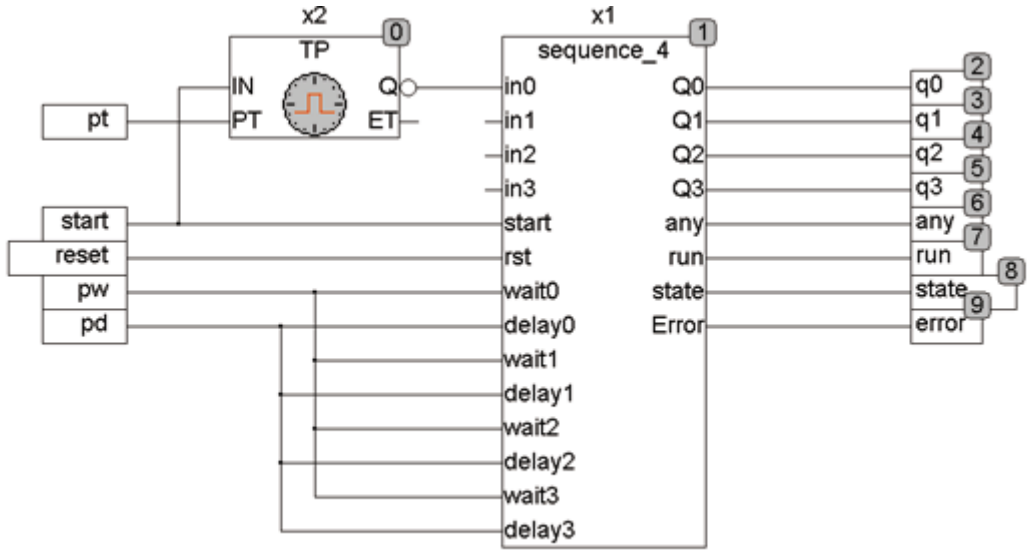
Der Ausgang Status ist ESR kompatibel und zeigt durch einen Wert 1- 4 an, dass ein Fehler aufgetreten ist. Ein Fehler tritt dann auf, wenn das entsprechende Eingangssignal an IN nicht während der entsprechenden Wartezeit auftritt.

Error = 1 bedeutet, dass in0 nicht innerhalb der Wartezeit aktiv geworden ist. Error 2 .. 4 entspricht den Eingängen 1 .. 3.

Ein Statuswert von 110 bedeutet Wartestellung und 111 bedeutet, dass gerade eine Sequenz durchlaufen wird.

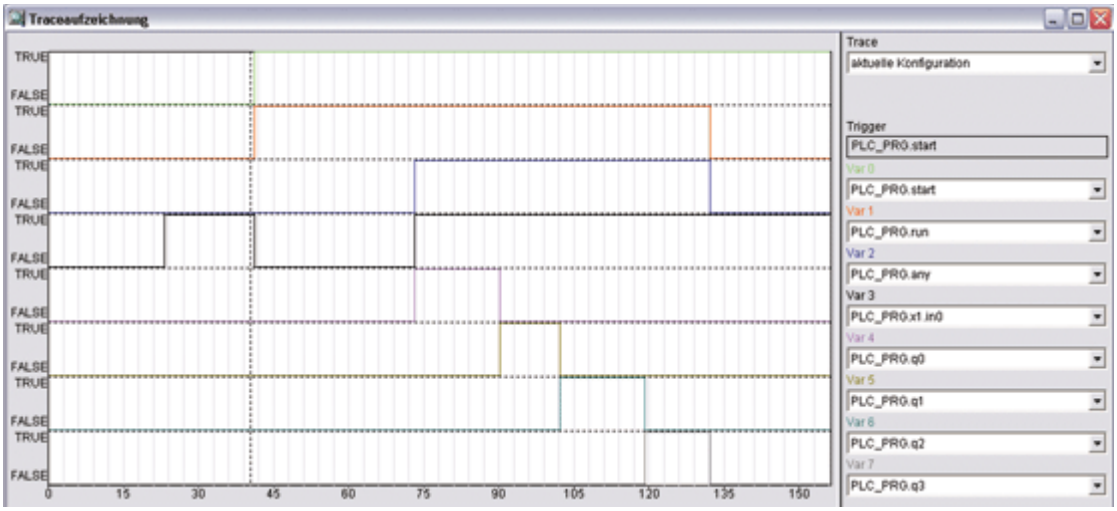
Beispiel:

Im Folgenden Beispiel wird mit einer steigenden Flanke an Start der Sequenzer gestartet. Gleichzeitig wird ein Puls-Generator TP mit 2 Sekunden gestartet und damit das Startsignal mit 2 Sekunden Verzögerung auf den Eingang IN0 gelegt. Der Sequenzer setzt unmittelbar nach dem Startimpuls das Ausgangssignal RUN und wartet dann für maximal 5 Sekunden auf ein Signal an IN0. Mit der steigenden Flanke an IN0, dass nach 2 Sekunden von TP generiert wird, wird Q0 gesetzt und ein Delay von 1 Sekunde abgewartet. Damit ist der erste Schritt beendet und die restlichen Schritte werden ohne auf ein Eingangssignal an in1..3 zu warten ausgeführt. Die Vorgabewerte für die Eingänge IN sind TRUE, wenn sie unbeschaltet sind.



Die Traceaufzeichnung zeigt das Startsignal (Grün) und das Signal RUN (Rot). Nach 2 Sekunden wird die steigende Flanke am Eingang auf in0 und danach auf die Ausgangssignale Q0..3 und QX gelegt.

Das Signal QX (Blau) ist dann aktiv, wenn eines der Ausgangssignale aktiv ist und das Signal RUN (Rot) ist vom Start bis zum Ende aktiv.



15.16. SEQUENCE_64

Type Funktionsbaustein

Input START : BOOL (steigende Flanke startet die Sequenz)

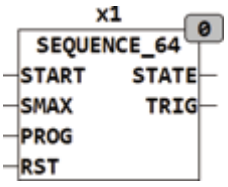
SMAX : INT (letzter State der Sequenz)

PROG : ARRAY[0..63] OF TIME (Zeitdauer der einzelnen States)

RST : BOOL (Asynchroner Reset-Eingang)

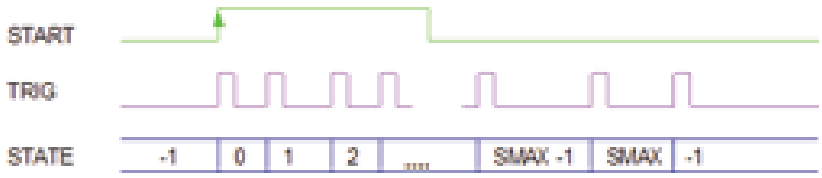
Output STATE : INT (State Ausgang)

TRIG : BOOL (Zeigt Zustandsveränderungen mit TRUE an)



SEQUENCE_64 erzeugt eine Zeitsequenz von bis zu 64 Zuständen. Im Ruhezustand steht der Ausgang STATE auf -1 und zeigt damit an das der Baustein nicht aktiv ist. Eine steigende Flanke an START startet die Sequenz und der Ausgang STATE schaltet auf 0. nach Ablauf der Wartezeit PROG[0] schaltet der Baustein weiter auf STATE = 1, Wartet die Zeit PROG[1] ab, schaltet auf STATE = 2, usw... bis der Ausgang STATE den Wert von SMAX erreicht hat. Nach Ablauf der Wartezeit PROG[SMAX] geht der Baustein wieder in den Ruhezustand (STATE = -1). Einen Wechsel auf einen neuen Zustand von STATE signalisiert der Ausgang TRIG mit einem TRUE für einen SPS Zyklus. Mit TRIG können bequem nachgeschaltet Bausteine gesteuert werden. Mit dem Eingang RST kann der Baustein jederzeit auch während des Ablaufs einer Sequenz in den Ausgangszustand zurückgesetzt werden.

Signalverlauf von SEQUENCE_64:



15.17. SEQUENCE_8

Type Funktionsbaustein

Input IN0..7 : BOOL (Freigabesignal für Q0..7)

START : BOOL (Startflanke für den Sequenzer)

RST : BOOL (Asynchroner Reset Eingang)

WAIT 0..7 : TIME (Wartezeit für das Eingangssignal an IN 0..7)

DELAY 0..7 : TIME (Verzögerungszeit, bis das Eingangssignal IN0..7 geprüft wird)

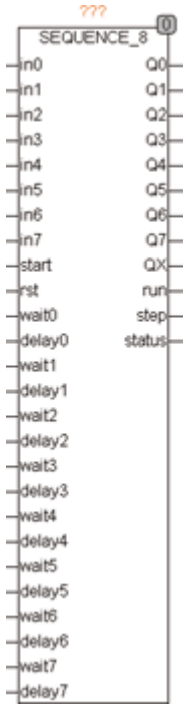
Output Q 0..7 : BOOL (Steuerausgänge)

QX : BOOL (TRUE, wenn einer der Ausgänge Q0 .. Q7 aktiv ist)

RUN : BOOL (RUN ist TRUE, wenn der Sequenzer läuft)

STEP : INT (gibt den momentanen Schritt an)

STATUS : BYTE (0 wenn kein Fehler vorliegt, sonst > 0)



Eine Funktionsbeschreibung von SEQUENCE_8 ist unter SEQUENCE_4 zu finden. SEQUENCE_8 ist Funktionsidentisch mit SEQUENCE_4. Er hat 8 anstelle von 4 Kanälen. SEQUENCE_8 findet Anwendung in der OSCAT Bibliothek im Modul LEGIONELLA.

15.18. TMAX

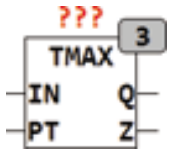
Type Funktionsbaustein

Input IN : BOOL (Eingangssignal)

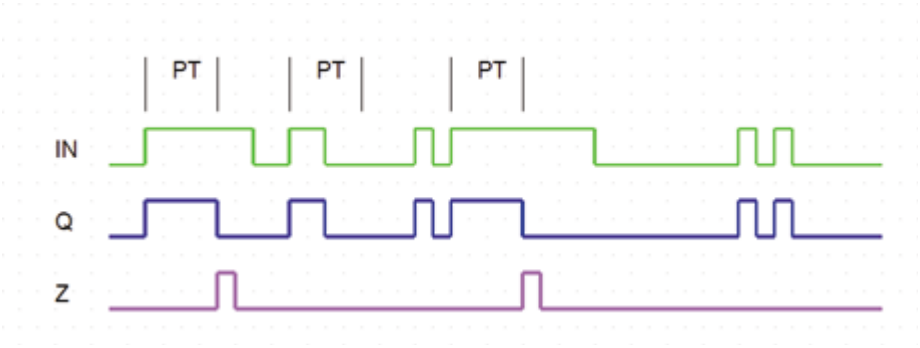
PT : TIME (Ausschaltverzögerung)

Output Q : BOOL (Ausgangssignal)

Z : BOOL (Trigger Ausgang)

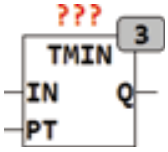


TMAX begrenzt die Dauer des Ausgangsimpulses auf die Zeit PT. Der Ausgang Q folgt dem Eingang IN, solange die TRUE Zeit von IN kürzer als PT ist. Ist IN länger als PT auf TRUE so wird der Ausgangsimpuls verkürzt. Immer dann wenn ein Ausgang durch eine Zeitüberschreitung auf FALSE geht wird der Ausgang Z für einen Zyklus auf TRUE gesetzt.

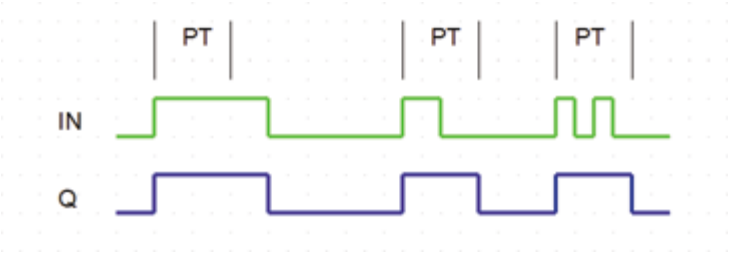


15.19. TMIN

Type	Funktionsbaustein
Input	IN : BOOL (Eingangssignal)
	PT : TIME (Ausschaltverzögerung)
Output	Q : BOOL (Ausgangssignal)

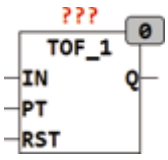


TMIN stellt sicher das der Ausgangsimpuls Q mindestens PT auf TRUE bleibt, auch wenn der Eingangsimpuls an IN kürzer als PT ist. ansonsten folgt der Ausgang Q dem Eingang IN.

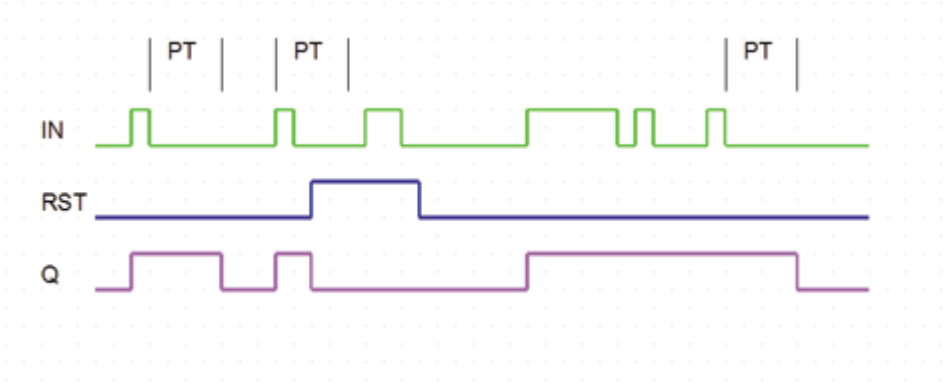


15.20. TOF_1

Type	Funktionsbaustein
Input	IN : BOOL (Eingangssignal)
	PT : TIME (Ausschaltverzögerung)
	RST : BOOL (asynchroner Reset)
Output	Q : BOOL (Ausgangssignal)

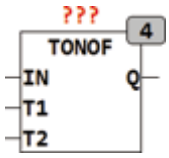


TOF_1 verlängert einen Eingangsimpuls an IN um die Zeit PT. TOF_1 hat die gleiche Funktionalität wie TOF aus der Standard LIB, jedoch mit einem zusätzlichen asynchronen Reset Eingang.



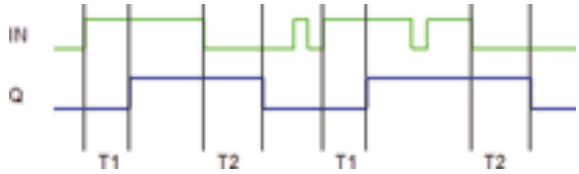
15.21. TONOF

Type	Funktionsbaustein
Input	IN : BOOL (Eingangssignal)
	T1 : TIME (Einschaltverzögerung)
	T2 : TIME (Ausschaltverzögerung)
Output	Q : BOOL (Ausgangspuls)



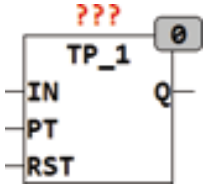
TONOF erzeugt eine Einschaltverzögerung T1 und eine Ausschaltverzögerung T2

Das steigende Flanke des Eingangssignals IN wird um T1 verzögert und die Fallende Flanke von IN wird um T2 verzögert.



15.22. TP_1

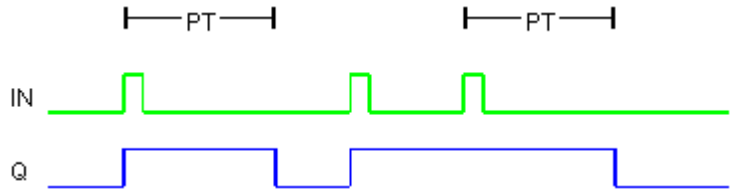
Type	Funktionsbaustein
Input	IN : BOOL (Eingangssignal)
	PT : TIME (Impulsdauer)
	RST : BOOL (asynchroner Reset)
Output	Q : BOOL (Ausgangspuls)



TP_1 ist ein flankengetriggerter Pulsgenerator der bei steigender Flanke an IN einen Ausgangsimpuls an Q mit der Dauer PT erzeugt. Wird während

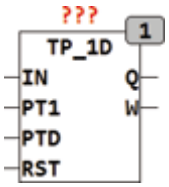
des Ausgangsimpulses eine weitere steigende Flanke an IN erzeugt, so verlängert sich der Ausgangsimpuls so dass nach der letzten steigenden Flanke der Ausgang für die Dauer von PT TRUE bleibt. Der Baustein kann jederzeit mit einem TRUE am Eingang RST zurückgesetzt werden.

Zeitverhalten von TP_1:



15.23. TP_1D

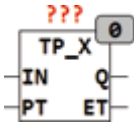
Type	Funktionsbaustein
Input	IN : BOOL (Eingangssignal)
	PT1 : TIME (Impulsdauer)
	PTD : TIME (Delay bis neuer Impuls erzeugt werden kann)
	RST : BOOL (asynchroner Reset)
Output	Q : BOOL (Ausgangspuls)



TP_1D ist ein flankengetriggerter Pulsgenerator der bei steigender Flanke an IN einen Ausgangsimpuls an Q mit der Dauer PT1 erzeugt. Wird während des Ausgangsimpulses eine weitere steigende Flanke an IN erzeugt, so verlängert sich der Ausgangsimpuls so dass nach der letzten steigenden Flanke der Ausgang für die Dauer von PT TRUE bleibt. Nach Ablauf der Pulsdauer PT1 blockiert der Baustein für die Zeit PTD den Ausgang. ein neuer Impuls kann erst nach Ablauf der Zeit PTD wieder gestartet werden. Der Baustein kann jederzeit mit einem TRUE am Eingang RST zurückgesetzt werden. Der Ausgang W zeigt an das der Baustein im Wartezyklus ist und solange W = TRUE kann kein neuer Impuls gestartet werden.

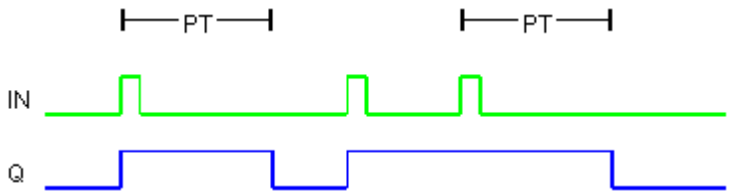
15.24. TP_X

Type	Funktionsbaustein
Input	IN : BOOL (Eingangssignal)
	PT : TIME (Impulsdauer)
Output	Q : BOOL (Ausgangspuls)
	ET : TIME (Zähle die Abgelaufene Zeit des Ausgangspulses)



TP_X ist ein mehrfach triggerbarer Impulsgenerator. Im Gegensatz zum Standardbaustein TP kann dieser Baustein mehrfach getriggert werden und dadurch der Ausgangsimpuls verlängert werden. Der Ausgang Q bleibt nach dem letzten Triggerereignis (steigende Flanke an IN) für die Zeit PT auf ein. Während Q TRUE ist, kann jederzeit durch eine weitere Eingangsflanke an IN der Timer wieder getriggert werden und der Ausgangsimpuls dadurch verlängert werden. Im Gegensatz zu TOF wird bei TP_X die Zeit PT ab der letzten steigenden Flanke gemessen, unabhängig wie lange IN auf TRUE bleibt. Das bedeutet das der Ausgang Q nach Ablauf der Zeit PT gemessen von der letzten steigenden Flanke an IN auf FALSE geht, auch wenn der Eingang IN noch TRUE ist.

Zeitverhalten von TP_X:



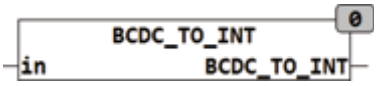
16. Logik Bausteine

16.1. BCDC_TO_INT

Type Funktion : INT

Input IN : BYTE (BCD codierter Eingang)

Output INT (Ausgangswert)



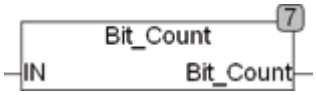
BCDC_TO_INT konvertiert ein BCD codiertes Eingangs BYTE in einen Integer Wert.

16.2. BIT_COUNT

Type Funktion : INT

Input IN : DWORD (Eingang)

Output INT (Anzahl der Bits, welche in IN den Wert TRUE (1) besitzen)



BIT_COUNT ermittelt die Anzahl der Bits in IN, welche den Wert TRUE (1) besitzen. Der Eingang IN ist DWORD und Kann auch die Typen Byte und Word verarbeiten.

16.3. BIT_LOAD_B

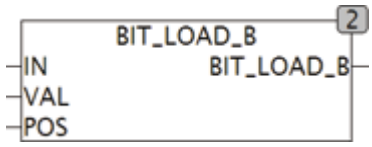
Type Funktion : BYTE

Input IN : BYTE (Eingang)

VAL : BOOL (Wert des zu ladenden Bits)

POS : INT (Position des zu ladenden Bits)

Output BYTE (Ausgang)



BIT_LOAD_B kopiert das am Eingang VAL anliegende Bit an die Position N im Byte IN. Das niederwertigste Bit B0 wird mit der Position 0 bezeichnet.

16.4. BIT_LOAD_B2

Type Funktion : BYTE

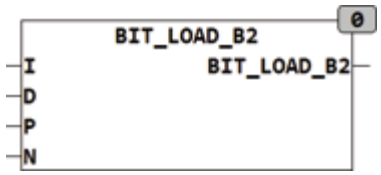
Input I : BYTE (Eingangs Wert)

D : BOOL (Wert der zu ladenden Bits)

P : INT (Position des zu ladenden Bits)

N : INT (Anzahl der Bits die ab Position P geladen werden)

Output BYTE (Ausgang)



BIT_LOAD_B2 kann mehrere Bits in einem Byte gleichzeitig setzen oder löschen. Die Position wird mit 0 für Bit0 und 7 für Bit7 angegeben. N gibt an wie viele Bits ab der angegebenen Position verändert werden. Wird N = 0 werden keine Bits verändert. Wird die P und N so spezifiziert das die zu schreibenden Bits über das höchste (Bit 7) hinausreicht so wird wieder bei Bit 0 begonnen.

BIT_LOAD_B2(2#1111_0000, TRUE, 1, 2) = 2#1111_0110

BIT_LOAD_B2(2#1111_1111, FALSE, 7, 2) = 2#0111_1110

16.5. BIT_LOAD_DW

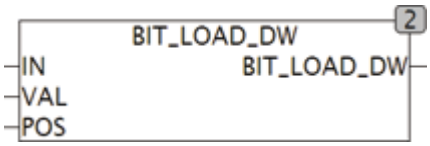
Type Funktion : DWORD

Input IN : DWORD (Eingang)

VAL : BOOL (Wert des zu ladenden Bits)

POS : INT (Position des zu ladenden Bits)

Output DWORD (Ausgang)



BIT_LOAD_DW kopiert das am Eingang VAL anliegende Bit an die Position N im DWORD IN. Das niederwertigste Bit B0 wird mit der Position 0 bezeichnet.

16.6. BIT_LOAD_DW2

Type Funktion : DWORD

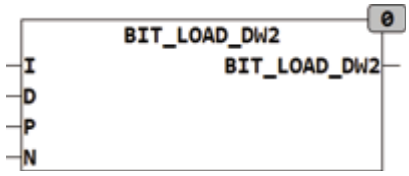
Input I : DWORD (Eingangs Wert)

D : BOOL (Wert der zu ladenden Bits)

P : INT (Position des zu ladenden Bits)

N : INT (Anzahl der Bits die ab Position P geladen werden)

Output DWORD (Ausgang)

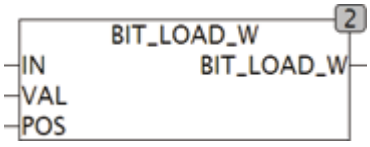


BIT_LOAD_DW2 kann mehrere Bits in einem DWORD gleichzeitig setzen oder löschen. Die Position wird mit 0 für Bit 0 und 31 für Bit 31 angegeben. N gibt an wie viele Bits ab der angegebenen Position verändert werden. Wird N = 0 werden keine Bits verändert. Wird P und N so spezifiziert, dass die zu schreibenden Bits über das höchste (Bit 31) hinausreicht so wird wieder bei Bit 0 begonnen.

Beispiele siehe unter BIT_LOAD_B2

16.7. BIT_LOAD_W

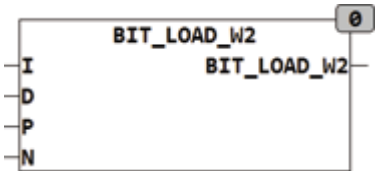
Type	Funktion : WORD
Input	IN : WORD (Eingang)
	VAL : BOOL (Wert des zu ladenden Bits)
	POS : INT (Position des zu ladenden Bits)
Output	WORD (Ausgang)



BIT_LOAD_W kopiert das am Eingang VAL anliegende Bit an die Position N im WORD IN. Das niederwertigste Bit B0 wird mit der Position 0 bezeichnet.

16.8. BIT_LOAD_W2

Type	Funktion : WORD
Input	I : WORD (Eingangs Wert)
	D : BOOL (Wert der zu ladenden Bits)
	P : INT (Position des zu ladenden Bits)
	N : INT (Anzahl der Bits die ab Position P geladen werden)
Output	WORD (Ausgang)

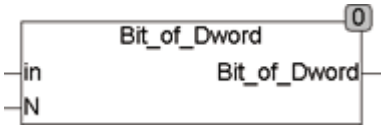


BIT_LOAD_W2 kann mehrere Bits in einem WORD gleichzeitig setzen oder löschen. Die Position wird mit 0 für Bit 0 und 15 für Bit 15 angegeben. N gibt an wie viele Bits ab der angegebenen Position verändert werden. Wird N = 0 werden keine Bits verändert. Wird P und N so spezifiziert, dass die zu schreibenden Bits über das höchste (Bit 15) hinausreicht so wird wieder bei Bit 0 begonnen.

Beispiele siehe unter BIT_LOAD_B2

16.9. BIT_OF_DWORD

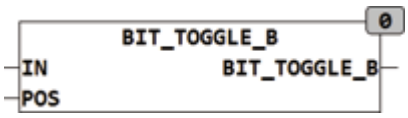
Type	Funktion : BOOL
Input	IN : DWORD (Eingang)
	N : INT (Nummer des Bits 0..31)
Output	BOOL (Ausgangsbit)



BIT_OF_DWORD extrahiert ein Bit aus dem DWORD am Eingang IN.
Bit0 für N=0, Bit1 für N=1 usw.

16.10. BIT_TOGGLE_B

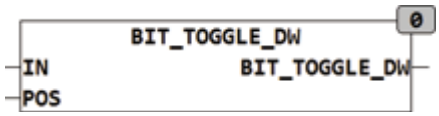
Type	Funktion : BYTE
Input	IN : BYTE (Eingangs Daten)
	POS : INT (Position)
Output	BYTE (Ausgangsbyte)



BIT_TOGGLE_B invertiert ein mit POS spezifiziertes Bit von IN.
BIT_TOGGLE_B(2#0000_1111, 2) = 2#0000_1011
BIT_TOGGLE_B(2#0000_1111, 7) = 2#1000_1111

16.11. BIT_TOGGLE_DW

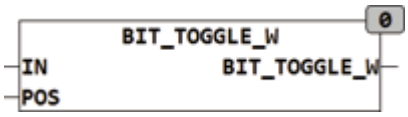
Type	Funktion : DWORD
Input	IN : DWORD (Eingangs Daten)
	POS : INT (Position)
Output	DWORD (Ausgangsbyte)



BIT_TOGGLE_DW invertiert ein mit POS spezifiziertes Bit von IN.
BIT_TOGGLE_DW(2#0000_1111, 2) = 2#0000_1011
BIT_TOGGLE_DW(2#0000_1111, 7) = 2#1000_1111

16.12. BIT_TOGGLE_W

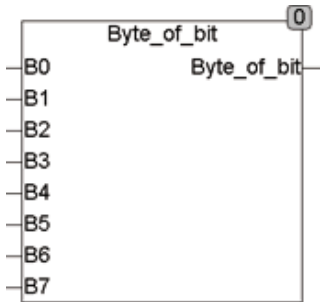
Type	Funktion : WORD
Input	IN : WORD (Eingangs Daten)
	POS : INT (Position)
Output	WORD (Ausgangsbyte)



BIT_TOGGLE_W invertiert ein mit POS spezifiziertes Bit von IN.
BIT_TOGGLE_B(2#0000_1111, 2) = 2#0000_1011
BIT_TOGGLE_B(2#0000_1111, 7) = 2#1000_1111

16.13. BYTE_OF_BIT

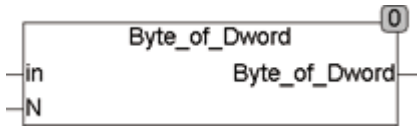
Type	Funktion : BYTE
Input	B0 .. B7 : BOOL (Eingangs Bits)
Output	BYTE (Ausgangsbyte)



BYTE_OF_BIT setzt ein Byte aus 8 einzelnen Bits (B0 .. B7) zusammen.

16.14. BYTE_OF_DWORD

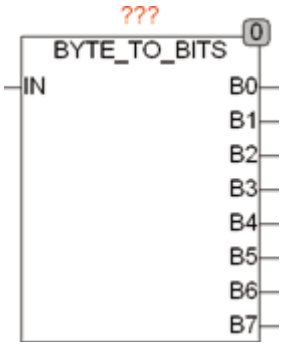
Type	Funktion : BYTE
Input	IN : DWORD (Eingangs-DWORD)
Output	BYTE (Ausgangsbyte)



BYTE_OF_DWORD extrahiert ein Byte (B0 .. B3) aus einem DWORD. Die einzelnen Bytes werden mit 0 - 3 am Eingang IN ausgewählt.

16.15. BYTE_TO_BITS

Type	Funktionsbaustein
Input	IN : BYTE (Eingangs Byte)
Output	B0 .. B7 : BOOL (Ausgangs Bits)



BYTE_TO_BITS zerlegt ein Byte (IN) in seine einzelnen Bits (B0 .. B7). Der Eingang IN ist als DWORD definiert, um wahlweise Byte, Word oder DWORD am Eingang verarbeiten zu können. Wird ein Word oder DWORD am Eingang verwendet, so werden nur die Bits 0..7 verarbeitet. Ein DWORD kann dann mit dem Standardbefehl SHR um 8 Bits nach Rechts verschoben und so das nächste Byte verarbeitet werden.

16.16. BYTE_TO_GRAY

Type Funktion : BYTE

Input IN : BYTE (Eingangs Byte)

Output Byte (Wert in Gray Code)

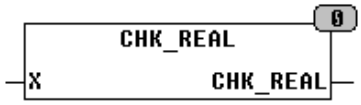
BYTE_TO_GRAY wandelt einen Byte Wert (IN) in den Gray Code.

16.17. CHK_REAL

Type Funktion : BYTE

Input X : REAL (Zu testender Wert)

Output BYTE (Rückgabewert)



CHK_REAL prüft X auf gültigen Wertebereich.

Die Rückgabewerte sind:

#00 gültige Gleitpunktzahl

#20 + unendlich

#40 - unendlich

#80 NAN

für weitere Informationen siehe auch die Floating Point Spezifikation IEEE754.

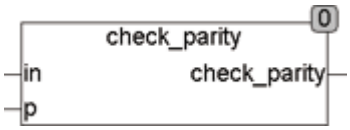
16.18. CHECK_PARITY

Type Funktion : BOOL

Input IN : BYTE (Eingangsbyte)

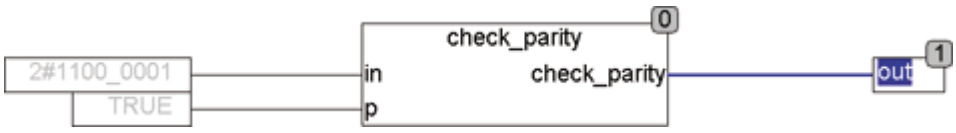
P : BOOL (Parity-Bit)

Output BYTE (Ausgang ist bei gerader Parität TRUE)

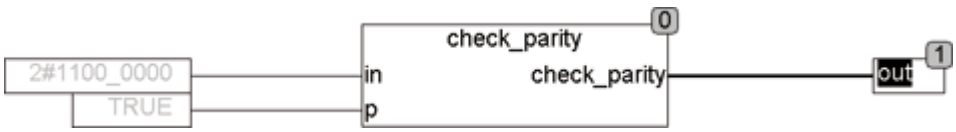


CHECK_PARITY überprüft ein Eingangsbyte IN und ein zugehöriges Parity-Bit P auf gerade Parität. Der Ausgang ist TRUE, wenn die Anzahl der Bits im Byte IN die den Wert TRUE besitzen zusammen mit dem Parity-Bit P eine gerade Anzahl ergeben.

Beispiel für Ausgang = TRUE:



Beispiel für Ausgang = FALSE:



16.19. CRC_CHECK

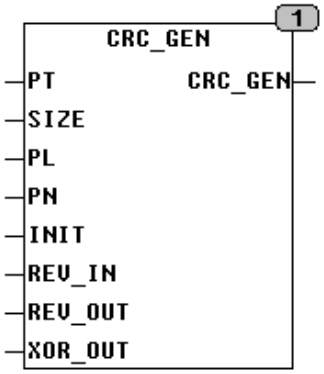
Der Baustein CRC_CHECK wurde aus der Library entfernt weil die Funktionalität auch uneingeschränkt mit dem Baustein CRC_GEN erfüllt werden kann.

Üblicherweise erzeugt CRC_GEN eine Checksumme die an die originäre Nachricht angehängt wird. Bildet man nun abermals die Checksumme über die Nachricht mit angehängter Checksumme so wird die neue Checksumme 0.

bei einigen speziellen CRC's bei denen dies nicht der Fall ist wird man die Checksumme nach Empfang einer Message nochmals neu über alle Nutzbytes ohne die übermittelte Checksumme bilden und dann mit der übermittelten Checksumme vergleichen.

16.20. CRC_GEN

Type	Funktion : DWORD
Input	PT : POINTER TO ARRAY OF BYTE (Datenpaket)
	SIZE : UINT (Größe des Arrays)
Setup	PL : UINT (Länge des Polynoms)
	PN : DWORD (Polynom)
	INIT : DWORD (INIT Daten)
	REV_IN : BOOL (Eingangsdaten Bytes Umkehren)
	REV_OUT : BOOL (Ausgangsdaten Umkehren)
	XOR_OUT : DWORD (Letztes XOR des Ausgangs)
Output	DWORD (errechnete CRC-Checksumme)



CRC_GEN generiert eine CRC-Checksumme aus einen beliebig großen Array of Byte. Beim Aufruf wird der Funktion ein Pointer auf das zu bearbeitende Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: CRC_GEN(ADR(Array), SIZEOF(Array),...), wobei Array der Name des zu bearbeitenden Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt. Das Polynom kann ein beliebiges POLYNOM bis maximal 32 Bit Länge sein. Ein Polynom $X^3 + X^2 + 1$ wird mit 101 dargestellt ($1*X^3 + 1*X^2 + 0*X^1 + 1* X^0$). Das höchstwertige Bit, in diesem Fall $1*X^3$ wird dabei im Polynom nicht angegeben den es ist immer eins. Es können Polynome bis X^{32} (CRC 32) verarbeitet werden.Durch den Wert INIT kann dem CR eine Startwert übergeben werden, üblich sind hier 0000 und FFFF. Der zu verwendende Startwert ist der in der Literatur übliche „Direct Initial Value“. Der Eingan XOR_OUT legt fest mit welcher Bitfolge die Checksumme am Ende mit XOR verknüpft wird. Die Eingänge REV_IN und REV_OUT legen die Bitfolge der Daten Fest. Wenn REV_IN = TRUE wird jedes Byte mit LSB beginnend verarbeitet, REV_IN = FALSE wird jeweils mit MSB begonnen. REV_OUT=TRUE dreht entsprechend die Bitfolge der Checksumme um. Der Bautein benötigt eine Mindestlänge der zu verarbeitenden Daten von 4 Bytes, und ist nach oben nur durch die maximale Array Größe begrenzt.

Die weiter unten folgende CRC Tabelle gibt nähere Auskunft über gebräuchliche CRC's und deren Setup Daten für CRC_GEN. Aufgrund der Vielzahl von möglichen und auch gebräuchlichen CRC's ist es uns nicht möglich eine vollständige Liste aufzuführen.

Für weitergehende Recherchen ist die Webseite http://regregex.bbcmicro.net/crc-catalogue.htm zu empfehlen.

Online Berechnungen zum testen sind mit folgendem Java Tool möglich: http://zorc.breitbandkatze.de/crc.html

Gebräuchliche CRC'S UND Polynome:

CRC	PL	PN [Hex]	INIT [Hex]	REV IN	REV OUT	XOUT [Hex]
CRC-3/ROHC	3	3	7	T	T	0
CRC-4/ITU	4	3	0	T	T	0
CRC-5/EPC	5	9	9	F	F	0
CRC-5/ITU	5	15	0	T	T	0
CRC-5/USB	5	5	1F	T	T	1F
CRC-6/DARC	6	19	0	T	F	0
CRC-6/ITU	6	3	0	T	T	0

CRC-7	7	9	0	F	F	0
CRC-7/ROHC	7	4F	7F	T	T	0
CRC-8	8	7	0	F	F	0
CRC-8/DARC	8	39	0	T	T	0
CRC-8/I-CODE	8	1D	FD	F	F	0
CRC-8/ITU	8	7	0	F	F	55
CRC-8/MAXIM	8	31	0	T	T	0
CRC-8/ROHC	8	7	FF	T	T	0
CRC-8/WCDMA	8	9B	0	T	T	0
CRC-10	10	233	0	F	F	0
CRC-11	11	385	1A	F	F	0
CRC-12/3GPP	12	80F	0	F	T	0
CRC-12/DECT	12	80F	0	F	F	0
CRC-14/DARC	14	805	0	T	T	0
CRC-15	15	4599	0	F	F	0
CRC-16/LHA	16	8005	0	T	T	0
CRC-16/CCITT-AUG	16	1021	1D0F	F	F	0
CRC-16/BUYPASS	16	8005	0	F	F	0
CRC-16/CCITT-FALSE	16	1021	FFFF	F	F	0
CRC-16/DDS	16	8005	800D	F	F	0
CRC-16/DECT-R	16	589	0	F	F	1
CRC-16/DECT-X	16	589	0	F	F	0
CRC-16/DNP	16	3D65	0	T	T	FFFF
CRC-16/EN13757	16	3D65	0	F	F	FFFF
CRC-16/GENIBUS	16	1021	FFFF	F	F	FFFF
CRC-16/MAXIM	16	8005	0	T	T	FFFF
CRC-16/MCRF4XX	16	1021	FFFF	T	T	0
CRC-16/RIELLO	16	1021	B2AA	T	T	0
CRC-16/T10-DIF	16	8BB7	0	F	F	0
CRC-16/TELEDISK	16	A097	0	F	F	0

CRC-16/USB	16	8005	FFFF	T	T	FFFF
CRC-16/CCITT-TRUE	16	1021	0	T	T	0
CRC-16/MODBUS	16	8005	FFFF	T	T	0
CRC-16/X-25	16	1021	FFFF	T	T	FFFF
CRC-16/XMODEM	16	1021	0	F	F	0
CRC-24/OPENPGP	24	864CFB	B704CE	F	F	0
CRC-24/FLEXRAY-A	24	5D6DCB	FEDCBA	F	F	0
CRC-24/FLEXRAY-B	24	5D6DCB	ABCDEF	F	F	0
CRC-32/PKZIP	32	04C11DB7	FFFFFFFF	T	T	FFFFFFFF
CRC-32/BZIP2	32	04C11DB7	FFFFFFFF	F	F	FFFFFFFF
CRC-32/CASTAGNOLI	32	1EDC6F41	FFFFFFFF	T	T	FFFFFFFF
CRC-32/D	32	A833982B	FFFFFFFF	T	T	FFFFFFFF
CRC-32/MPEG2	32	04C11DB7	FFFFFFFF	F	F	0
CRC-32/POSIX	32	04C11DB7	0	F	F	FFFFFFFF
CRC-32/Q	32	814141AB	0	F	F	0
CRC-32/JAM	32	04C11DB7	FFFFFFFF	T	T	0
CRC-32/XFER	32	AF	0	F	F	0

16.21. DEC_2

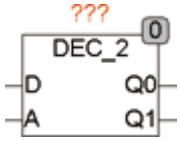
Type Funktionsbaustein

Input D : BOOL (Eingangs Bit)

A : BOOL (Adresse)

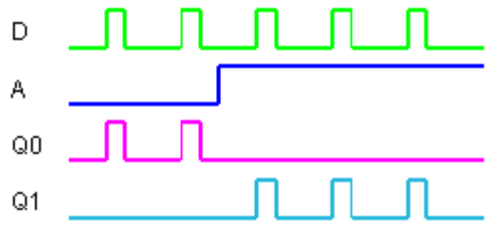
Output Q0 : BOOL (TRUE bei A=0)

Q1 : BOOL (TRUE bei A=1)



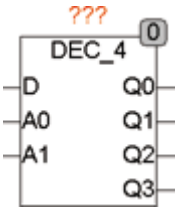
DEC_2 ist ein 2-Bit Dekodierbaustein. Ist A=0, so wird der Eingang D auf Ausgang Q0 geschaltet. Ist A=1, so wird D auf Q1 geschaltet. Mit anderen Worten: Q0=1 wenn D=1 und A=0.

Logische Verknüpfung: Q0 = D & /A; Q1 = D & A



16.22. DEC_4

Type	Funktionsbaustein
Input	D : BOOL (Eingangs Bit)
	A0 : BOOL (Adresse Bit0)
	A1 : BOOL (Adresse Bit1)
Output	Q0 : BOOL (TRUE bei A0=0 und A1=0)
	Q1 : BOOL (TRUE bei A0=1 und A1=0)
	Q2 : BOOL (TRUE bei A0=0 und A1=1)
	Q3 : BOOL (TRUE bei A0=1 und A1=1)



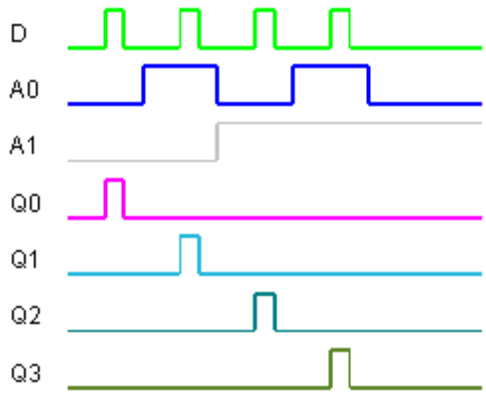
DEC_4 ist ein 4-Bit Dekodierbaustein. Ist A0=0 und A1=0 wird der Eingang D auf Ausgang Q0 geschaltet. Wenn A0=1 und A1=1 wird D auf Q3 geschaltet. Mit anderen Worten: Q0=1, wenn D=1 und A0=0 und A1=0.

Logische Verknüpfung: Q0 = D & /A0 & /A1

Q1 = D & A0 & /A1

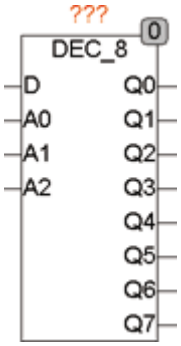
Q2 = D & /A0 & A1

Q3 = D & A0 & A1



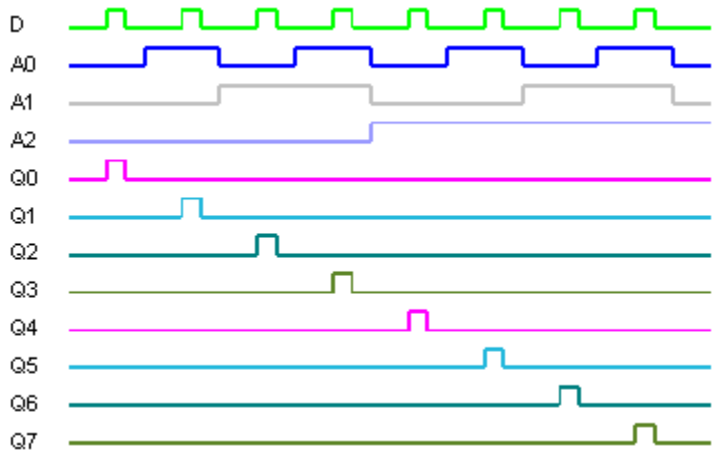
16.23. DEC_8

Type	Funktionsbaustein
Input	D : BOOL (Eingangs Bit)
	A0 : BOOL (Adresse Bit0)
	A1 : BOOL (Adresse Bit1)
	A2 : BOOL (Adresse Bit2)
Output	Q0 : BOOL (TRUE bei A0=0 und A1=0 und A2=0)
	Q1 : BOOL (TRUE bei A0=1 und A1=0 und A2=0)
	Q2 : BOOL (TRUE bei A0=0 und A1=1 und A2=0)
	Q3 : BOOL (TRUE bei A0=1 und A1=1 und A2=0)
	Q4 : BOOL (TRUE bei A0=0 und A1=0 und A2=1)
	Q5 : BOOL (TRUE bei A0=1 und A1=0 und A2=1)
	Q6 : BOOL (TRUE bei A0=0 und A1=1 und A2=1)
	Q7 : BOOL (TRUE bei A0=1 und A1=1 und A2=1)



DEC_8 ist ein 8-Bit Dekodierbaustein. Ist A0=0 und A1=0 und A2=0 wird der Eingang D auf Ausgang Q0 geschaltet, wenn A0=1 und A1=1 und A2=1 wird D auf Q3 geschaltet. Mit anderen Worten: Q0=1 wenn D=1 und A0=0 und A1=0 und A2=0.

Das folgende Schaubild verdeutlicht die Logik des Bausteins:



Logische Verknüpfung:

Q0 = D & /A0 & /A1 & /A2 Q1 = D & A0 & /A1 & /A2

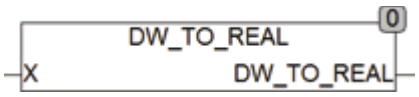
Q2 = D & /A0 & A1 & /A2 Q3 = D & A0 & A1 & /A

Q4 = D & /A0 & /A1 & A2 Q5 = D & A0 & /A1 & A2

Q6 = D & /A0 & A1 & A2 Q7 = D & A0 & A1 & A2

16.24. DW_TO_REAL

Type	Funktion : REAL
Input	X : DWORD (Eingang)
Output	REAL (Ausgangswert)



DW_TO_REAL kopiert das Bitmuster eines DWORD (IN) in einen REAL. Es werden dabei die einzelnen Bits kopiert ohne auf deren Bedeutung zu achten. Die Funktion REAL_TO_DW ist die Umkehrfunktion so das die Konvertierung von REAL_TO_DW und anschließend DW_TO_REAL wiederum den

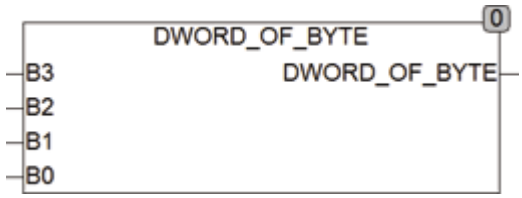
Ausgangswert ergeben. Die IEC Standardfunktion DWORD_TO_REAL wandelt den Wert des DWORDS in einen REAL Wert.

16.25. DWORD_OF_BYTE

Type Funktion : DWORD

Input B3 : Byte (Eingangs Byte 3)
B2 : Byte (Eingangs Byte 2)
B1 : Byte (Eingangs Byte 1)
B0 : Byte (Eingangs Byte 0)

Output DWORD (Ergebnis-DWORD)



DWORD_OF_BYTE setzt aus 4 separaten Bytes B0 .. B3 ein DWORD zusammen.

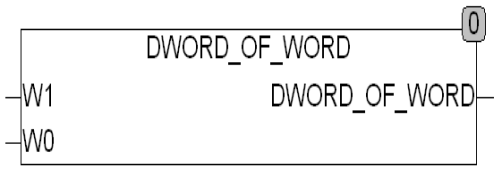
Ein DWORD setzt sich zusammen wie folgt: B3-B2-B1-B0.

16.26. DWORD_OF_WORD

Type Funktion : DWORD

Input W1 : WORD (Eingangs WORD 1)
W0 : WORD (Eingangs WORD 0)

Output DWORD (Ergebnis-DWORD)



DWORD_OF_WORD setzt aus 2 separaten WORDS W0 und W1 ein DWORD zusammen.

Ein DWORD setzt sich zusammen wie folgt: W1-W0.

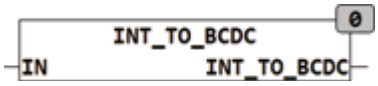
16.27. GRAY_TO_BYTE

Type	Funktion
Input	IN : BYTE (Gray codierter Wert)
Output	Byte (Binärer Wert)

GRAY_TO_BYTE wandelt einen Gray codierten Wert (IN) in ein Byte.

16.28. INT_TO_BCDC

Type	Funktion : BYTE
Input	IN : INT (Eingangswert)
Output	BYTE (BCD codierter Ausgangswert)



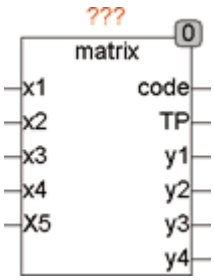
INT_TO_BCDC konvertiert den Eingangswert IN in einen BCD codierten Ausgangswert.

16.29. MATRIX

Type	Funktionsbaustein
Input	X1 .. X5 : BOOL (Zeileneingänge)
Setup	RELEASE : BOOL (Ein Tastencode wird beim Drücken und loslassen einer Taste erzeugt)
Output	CODE : Byte (Ausgang für Tastencode)

TP : BOOL (TP ist für einen Zyklus TRUE, wenn ein neuer Tastencode ansteht)

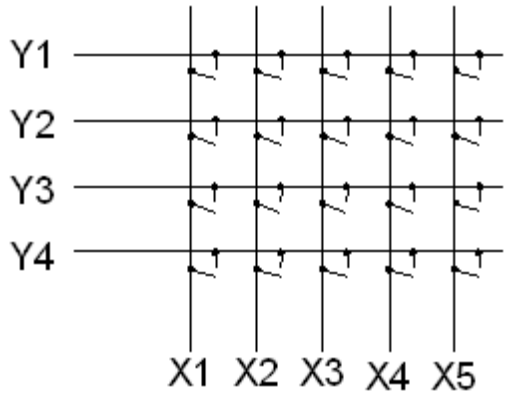
Y1 .. Y4 : BOOL (Zeilenausgänge)



MATRIX ist ein Matrix-Tastatur-Controller für maximal 4 Spalten und 5 Zeilen. Mit jedem SPS Zyklus schaltet MATRIX den Spaltenausgang um eine Spalte weiter, sodass die Zeilen Y1 bis Y4 nacheinander abgefragt werden. Für jede Spalte werden die Zeileneingänge X1 bis X5 abgefragt und falls eine Taste gedrückt ist, wird der entsprechende Tastencode am Ausgang angezeigt. Der Ausgang TP ist genau dann einen Zyklus TRUE, wenn der Ausgang CODE einen neuen Wert anzeigt. Wenn die Setup-Variable RELEASE auf TRUE gesetzt wird, dann wird für das Drücken und das Loslassen einer Taste jeweils ein Tastencode gesendet. Falls RELEASE auf FALSE gesetzt ist, wird nur beim Betätigen einer Taste ein Tastencode erzeugt. Der Tastencode des Ausgangs setzt sich wie folgt zusammen:

Bit	CODE Output
7	1 when key is pressed, 0 when key is released
6	Line number Bit 2
5	Line number Bit 1
4	Line number Bit 0
3	Always 0
2	Row number Bit 2
1	Row Number Bit 1
0	Row Number Bit 0

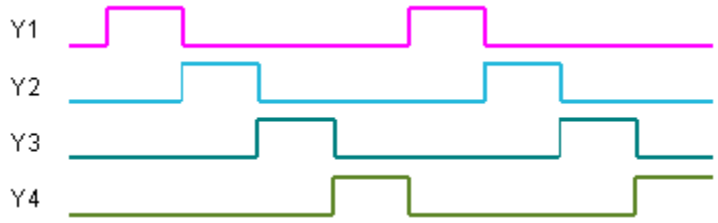
Der Matrixcontroller wird wie folgt beschaltet:



Bei dieser einfachen Beschaltung können bis zu 20 (4 * 5) Taster ausgewertet werden. Jedoch ist hierbei zu beachten, dass nur bedingt mehrere Tasten gleichzeitig gedrückt werden können. Der Controller kann mit dieser Beschaltung mehrere Tasten in einer Spalte sicher erkennen, jedoch

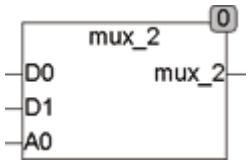
nicht wenn Tasten an verschiedenen Spalten gleichzeitig gedrückt werden. Die Beschaltung kann jedoch erweitert werden, indem jeder einzelne Taster über Dioden entkoppelt wird und damit die Beeinflussung verschiedener
Spalten untereinander verhindert wird. Bei der Beschaltung mit Dioden können beliebig viele Tasten gleichzeitig gedrückt und sicher Ausgewertet werden. Die Ausgänge des Matrixcontrollers scannen kontinuierlich die Zeilen der Tastaturmatrix ab. Je SPS Zyklus wird eine Zeile eingelesen. Sind in einer Zeile mehrere Tasten gedrückt bzw. verändert worden, so werden die Änderungen als Codes über die folgenden Zyklen ausgegeben. Der Baustein merkt sich die einzelnen Tastencodes und gibt je Zyklus immer nur einen Code aus, sodass kein Code verloren gehen kann.

Das folgende Timing Diagramm zeigt das Abtasten der Tastenreihen:



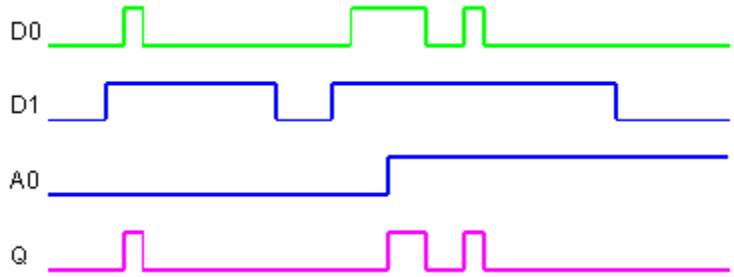
16.30. MUX_2

Type	Funktion : BOOL
Input	D0 : BOOL (Bit 0)
	D1 : BOOL (Bit 1)
	A0 : BOOL (Adresse)
Output	BOOL (D0, wenn A0=0 und D1, wenn A0=1)



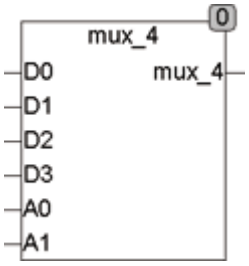
MUX_2 ist ein 2-Bit Multiplexer. Der Ausgang entspricht D0, wenn A0=0 und er entspricht D1, wenn A0=1.

Logische Verknüpfung: MUX_2 = D0 & /A0 + D1 & A0



16.31. MUX_4

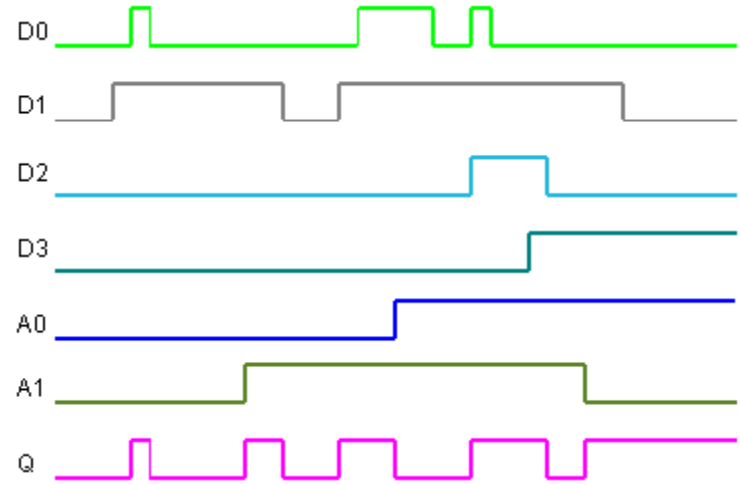
Type	Funktion : BOOL
Input	D0 : BOOL (Eingang 0)
	D1 : BOOL (Eingang 1)
	D2 : BOOL (Eingang 2)
	D3 : BOOL (Eingang 3)
Output	BOOL (D0, wenn A0=0 und A1 = 0, usw...)



MUX_4 ist ein 4-Bit Multiplexer. Der Ausgang entspricht D0, wenn A0=0 und A1=0. Er entspricht D3, wenn A0=1 und A1=1.

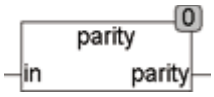
Logische Verknüpfung: MUX_4 = D0 & /A0 & /A1 + D1 & A0 & /A1

+ D2 & /A0 & A1 + D3 & A0 & A1



16.32. PARITY

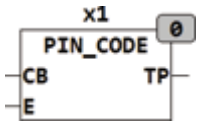
Type	Funktion : BOOL
Input	IN : BYTE (Eingangs BYTE)
Output	BOOL (Ausgang ist TRUE, wenn Parität gerade ist)



PARITY bildet eine gerade Parität über das Eingangsbyte IN. Der Ausgang ist TRUE wenn die Anzahl der TRUE Bits im Byte (In) ungerade ist.

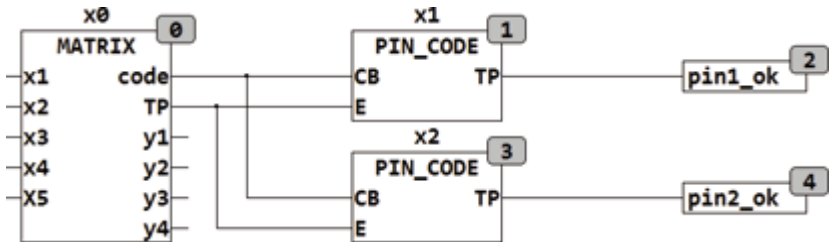
16.33. PIN_CODE

Type	Funktionsbaustein
Input	CB : BYTE (Eingang)
	E : BOOL (Enable Eingang)
SETUP	PIN : STRING(8) (zu prüfender String)
Output	TP (Trigger Ausgang)



PIN_CODE überprüft einen Datenstrom von Bytes auf das Vorkommen einer bestimmten Sequenz. Tritt die Sequenz auf, wird dies mit einem TRUE am Ausgang TP signalisiert.

Im folgenden Beispiel werden 2 Bausteine PIN_CODE verwendet um 2 CODE_SEQUENZEN einer Matrix Tastatur zu dekodieren.

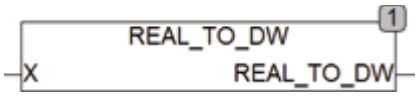


16.34. REAL_TO_DW

Type Funktion : DWORD

Input IN : REAL (Eingang)

Output DWORD (Ausgangswert)



REAL_TO_DW kopiert das Bitmuster eines REAL (IN) in ein DWORD. Es werden dabei die einzelnen Bits kopiert ohne auf deren Bedeutung zu achten. Die Funktion DW_TO_REAL ist die Umkehrfunktion so das die Konvertierung von REAL_TO_DW und anschließend DW_TO_REAL wiederum den Ausgangswert ergibt. Die IEC Standardfunktion REAL_TO_DWORD wandelt den REAL Wert in einen Festzahlenwert und Rundet an der kleinsten Stelle des DWORD.

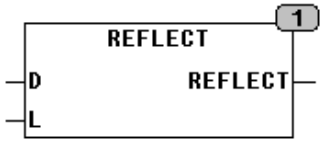
16.35. REFLECT

Type Funktion : DWORD

Input D : DWORD (Eingangswert)

L : INT (Anzahl der zu drehenden Bits)

Output DWORD (Ausgangswert)



REFLECT dreht die Reihenfolge der durch die Anzahl L spezifizierten Bits-Bits in einem DWORD um. Die höherwertigen Bits als die durch die Länge L spezifizierten bleiben unverändert.

Beispiel: REVERSE(10101010 00000000 11111111 10011110, 8) ergibt 10101010 00000000 11111111 01111001

Beispiel: REVERSE(10101010 00000000 11111111 10011110, 32) ergibt 01111001 11111111 00000000 01010101

folgendes Beispiel in ST würde alle Bytes in einem DWORD X umdrehen, jedoch die Byte Reihenfolge belassen:

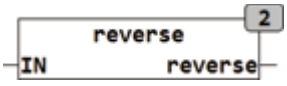
```
FOR i := 0 TO 3 DO
        REVERSE(X, 8);
        ROR(X,8);
END_FOR
```

16.36. REVERSE

Type Funktion : BYTE

Input IN : BYTE (Eingangs BYTE)

Output BYTE(Ausgangs Byte)



REVERSE dreht die Reihenfolge der Bits in einem Byte um. Bit7 von IN wird zu Bit 0, Bit 6 wird zu Bit 1 usw.

Beispiel: REVERSE(10011110) = 01111001

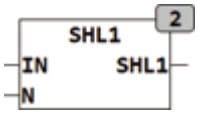
16.37. SHL1

Type Funktion : DWORD

Input IN : DWORD (Eingangsdaten)

N : INT (Anzahl der zu schiebenden Bits)

Output DWORD (Ergebnis)



SHL1 schiebt das Eingangs DWORD um N Bits nach Links und füllt die N Rechten Bits mit 1en auf. Im Gegensatz zur IEC Standard Funktion SHL die beim Schieben mit Nullen aufgefüllt wird bei SHL1 mit Einsen Aufgefüllt.

Beispiel: SHL1(11110000,2) ergibt 11000011

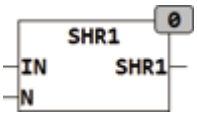
16.38. SHR1

Type Funktion : DWORD

Input IN : DWORD (Eingangsdaten)

N : INT (Anzahl der zu schiebenden Bits)

Output DWORD (Ergebnis)



SHR1 schiebt das Eingangs DWORD um N Bits nach Rechts und füllt die N linken Bits mit 1en auf. Im Gegensatz zur IEC Standard Funktion SHL die beim Schieben mit Nullen aufgefüllt wird bei SHR1 mit Einsen Aufgefüllt.

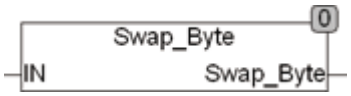
Beispiel: SHR1(11110000,2) ergibt 11111100

16.39. SWAP_BYTE

Type Funktion : WORD

Input IN : WORD (Eingangsdaten)

Output WORD (Ergebnis)



SWAP_BYTE tauscht das High und Low Byte in einem WORD.

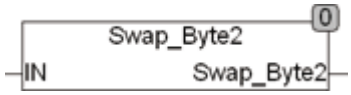
Beispiel: SWAP_BYTE(16#33df) = 16#df33.

16.40. SWAP_BYTE2

Type Funktion : DWORD

Input IN : DWORD (Eingangsdaten)

Output DWORD (Ergebnis)



SWAP_BYTE2 kehrt die Reihenfolge der Bytes in einem DWORD um.

Beispiel: SWAP_BYTE2(16#33df1122) = 16#2211df33.

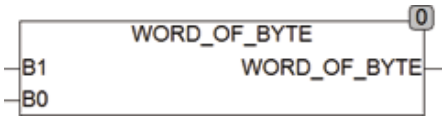
16.41. WORD_OF_BYTE

Type Funktion : WORD

Input B1 : Byte (Eingangs Byte 1)

B0 : Byte (Eingangs Byte 0)

Output Word (Ergebnis Word)



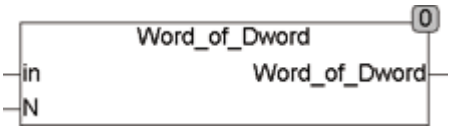
WORD_OF_BYTE setzt aus 2 separaten Bytes B0 und B1 ein Word zusammen.

16.42. WORD_OF_DWORD

Type Funktion : WORD

Input IN : DWORD (Eingangs DWORD)

Output WORD (Ausgangs WORD)

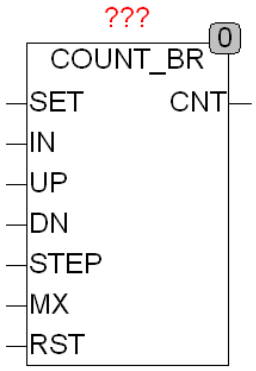


WORD_OF_DWORD extrahiert ein Word (W0 .. W1) aus einem DWORD.

17. Latches, Flip-Flop und Schieberegister

17.1. COUNT_BR

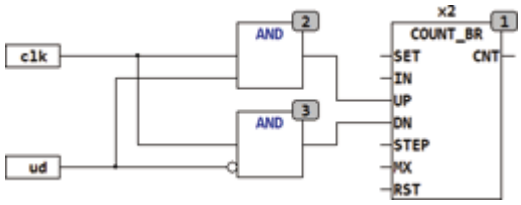
Type	Funktionsbaustein
Input	SET : BOOL (asynchroner Set)
	IN : BYTE (Vorgabewert für Set)
	UP : BOOL (Vorwärts Schalter flankengetriggert)
	DN : BOOL (Rückwärts Schalter flankengetriggert)
	STEP : BYTE (Schrittweite des Counters)
	MX : BYTE (Maximalwert des Counters)
	RST : BOOL (asynchroner Reset)
Output	CNT : BYTE (Ausgang)



COUNT_BR ist ein Byte Zähler der von 0 bis MX zählt und dann wieder bei 0 beginnt. Der Zähler kann mittels 2 flankengetriggerten Eingängen UP und DN sowohl vorwärts als auch Rückwärts Zählen. beim erreichen eines Endwerts 0 oder MX wird wieder bei 0 beziehungsweise MX weiter gezählt. Der Eingang STEP legt die Schrittweite des Zählers fest. Mit einem TRUE am Eingang SET wird der Zähler auf den an IN anliegenden Wert gesetzt. Ein Reset Eingang RST setzt den Zähler jederzeit auf 0.

	SET	IN	UP	DN	STEP	RST	CNT
Reset	-	-	-	-	-	1	0
Set	1	N	-	-	-	0	N
up	0	-	↑	0	N	0	CNT + N
down	0	-	0	↑	N	0	CNT - N

Falls die unabhängigen Eingänge UP und DN mit CLK und einen Steuereingang UP/DN ersetzt werden sollen kann dies mittels zwei AND Gattern vor den Eingängen erfolgen:



COUNT_BR kann bei jedem UP oder Down Befehl mit individueller Schrittweite Arbeiten, dabei ist zu beachten das der Zähler sich so verhält als ob er intern die Anzahl von STEP Schritte Vorwärts oder Rückwärts zählt.

Beispiel:

MX = 50, STEP=10

Der Zähler Arbeitet dann wie folgt:

0,10,20,30,40,50,9,19,.....

Wird in diesem Beispiel 50 erreicht, so wird dies als Maximalwert erkannt und bei 0 weitergezählt. Intern sieht dies wie folgt aus:

50,0,1,2,3,4,5,6,7,8,9 also genau 50 + 10 wenn nach 50 wieder die 0 kommt.

Die Implementation eines Zählers 0...50 in Zehnerschritten sieht wie folgt aus:

MX = 59, STEP = 10: ergibt die folge 0,10...50,0,10

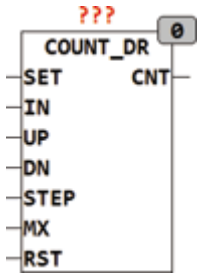
der Übergang von 50 auf 0 ist dann genau 10 Schritte.

17.2. COUNT_DR

Type Funktionsbaustein

Input SET : BOOL (asynchroner Set)
IN : DWORD (Vorgabewert für Set)
UP : BOOL (Vorwärts Schalter flankengetriggert)
DN : BOOL (Rückwärts Schalter flankengetriggert)
STEP : DWORD (Schrittweite des Counters)
MX : DWORD (Maximalwert des Counters)
RST : BOOL (asynchroner Reset)

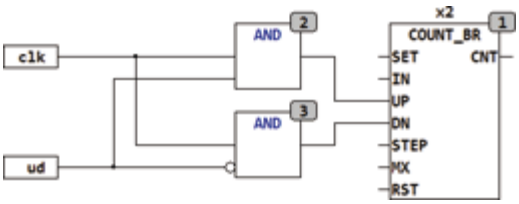
Output CNT : DWORD (Ausgang)



COUNT_DR ist ein DWORD (32-Bit) Zähler der von 0 bis MX zählt und dann wieder bei 0 beginnt. Der Zähler kann mittels 2 flankengetriggerten Eingängen UP und DN sowohl vorwärts als auch Rückwärts Zählen. beim erreichen eines Endwerts 0 oder MX wird wieder bei 0 beziehungsweise MX weiter gezählt. Der Eingang STEP legt die Schrittweite des Zählers fest. Mit einem TRUE am Eingang SET wird der Zähler auf den an IN anliegenden Wert gesetzt. Ein Reset Eingang RST setzt den Zähler jederzeit auf 0.

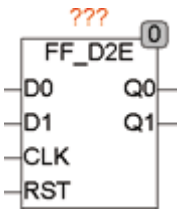
	SET	IN	UP	DN	STEP	RST	CNT
Reset	-	-	-	-	-	1	0
Set	1	N	-	-	-	0	N
up	0	-	↑	0	N	0	CNT + N
down	0	-	0	↑	N	0	CNT - N

Falls die unabhängigen Eingänge UP und DN mit CLK und einen Steuereingang UP/DN ersetzt werden sollen kann dies mittels zwei AND Gattern vor den Eingängen erfolgen:

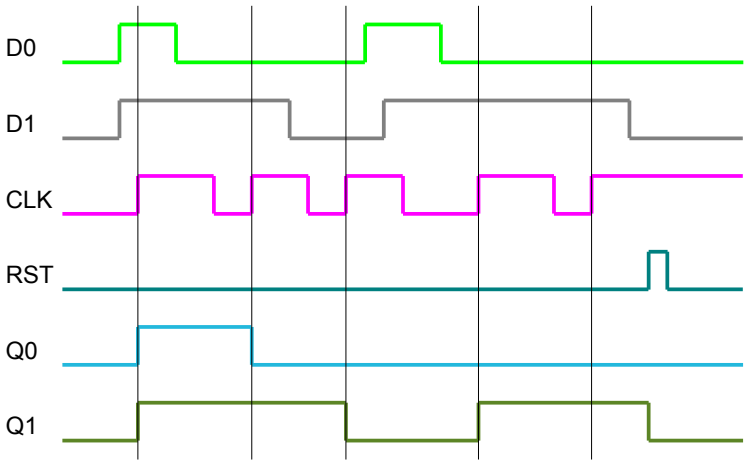


17.3. FF_D2E

Type	Funktionsbaustein
Input	D0 : BOOL (Data 0 in)
	D1 : BOOL (Data 1 in)
	CLK : BOOL (Takteingang)
	RST : BOOL (asynchroner Reset)
Output	Q0 : BOOL (Data 0 out)
	Q1 : BOOL (Data 1 out)

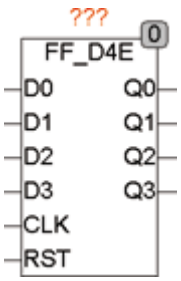


FF_D2E ist ein 2 Bit flankengetriggertes D-Flip-Flop mit asynchronem Reset-Eingang. Das D-Flip-Flop speichert die Werte am Eingang D mit einer steigenden Flanke am CLK Eingang.



17.4. FF_D4E

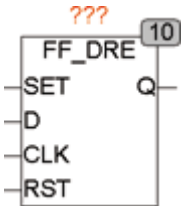
Type	Funktionsbaustein
Input	D0 : BOOL (Data 0 in)
	D1 : BOOL (Data 1 in)
	D2 : BOOL (Data 2 in)
	D3 : BOOL (Data 3 in)
	CLK : BOOL (Takteingang)
	RST : BOOL (asynchroner Reset)
Output	Q0 : BOOL (Data 0 Out)
	Q1 : BOOL (Data 1 Out)
	Q2 : BOOL (Data 2 Out)
	Q3 : BOOL (Data 3 Out)



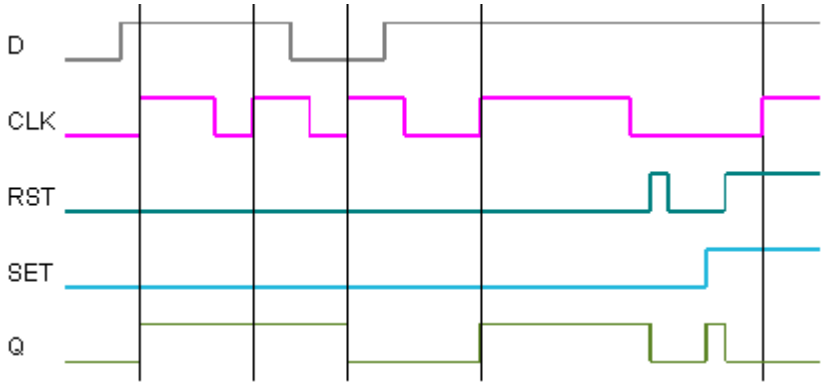
FF_D2E ist ein 4 Bit flankengetriggertes D-Flip-Flop mit asynchronem Reset-Eingang. Das D-Flip-Flop speichert die Werte am Eingang D mit einer steigenden Flanke an CLK. Detaillierte Angaben finden Sie beim Baustein FF_D2E.

17.5. FF_DRE

Type	Funktionsbaustein
Input	SET : BOOL (asynchroner Set)
	D : BOOL (Data in)
	CLK : BOOL (Takteingang)
	RST : BOOL (asynchroner Reset)
Output	Q : BOOL (Data Out)

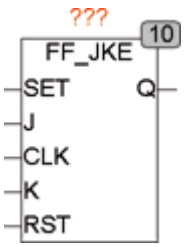


FF_DRE ist ein flankengetriggertes D-Flip-Flop mit asynchronem Set und Reset Eingang. Eine steigende Flanke an CLK speichert den Eingang D auf den Ausgang Q. Ein TRUE am SET oder RST-Eingang setzt oder löscht den Ausgang Q zu jeder Zeit unabhängig von CLK. Der Reset Eingang hat Vorrang vor den Set Eingang. Wenn beide aktiv (TRUE) sind wird ein Reset ausgeführt und Set ignoriert.

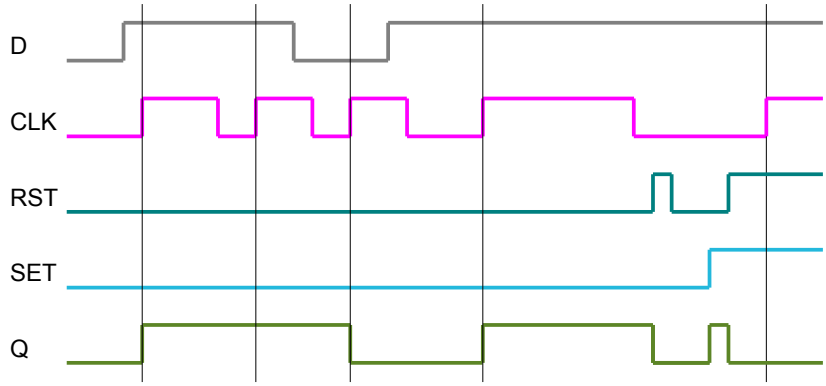


17.6. FF_JKE

Type	Funktionsbaustein
Input	SET : BOOL (asynchroner Set)
	J : BOOL (Takt-synchroner Set)
	CLK : BOOL (Takteingang)
	K : BOOL (Takt-synchroner Reset)
	RST : BOOL (asynchroner Reset)
Output	Q : BOOL (Ausgang)

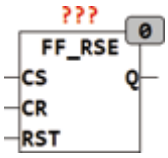


FF_JKE ist ein flankengetriggertes JK-Flop-Flop mit asynchronen Set und Reset Eingängen. Das JK-Flip-Flop setzt den Ausgang Q, wenn bei einer steigenden Flanke von CLK der Input J TRUE ist. Q wird FALSE, wenn bei einer steigenden Taktflanke der Eingang K TRUE ist. Sind die beiden Eingänge J und K bei einer Steigenden Taktflanke TRUE, so wird der Ausgang negiert. Er schaltet bei jedem Takt das Ausgangssignal um.



17.7. FF_RSE

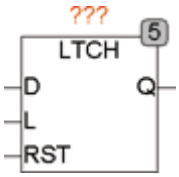
Type	Funktionsbaustein
Input	CS : BOOL (flankensensitiver Set)
	CR : BOOL (flankensensitiver Reset)
	RST : BOOL (asynchroner Reset)
Output	Q : BOOL (Ausgang)



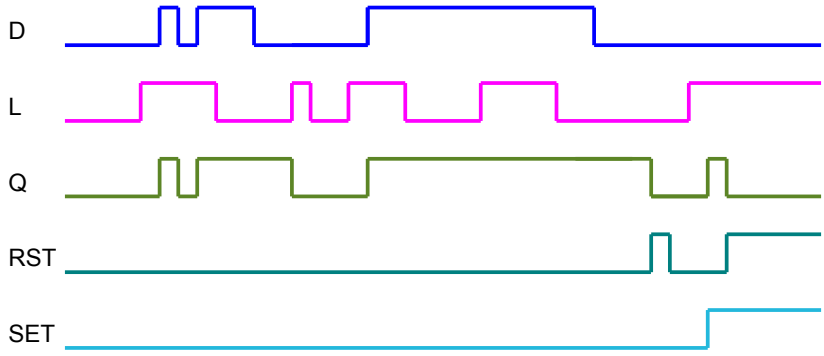
FF_RSE ist ein flankengetriggertes RS Flip-Flop. Der Ausgang Q wird durch eine steigende Flanke an CS gesetzt und durch eine steigende Flanke an CR gelöscht. Treten beide Flanken (CS und CR) gleichzeitig auf, so wird der Ausgang auf FALSE gesetzt. Ein Asynchroner Reset Eingang RST setzt den Ausgang jederzeit auf FALSE.

17.8. LTCH

Type	Funktionsbaustein
Input	D : BOOL (Data in)
	L : BOOL (Latch enable Signal)
	RST : BOOL (asynchroner Reset)
Output	Q : BOOL (Data Out)

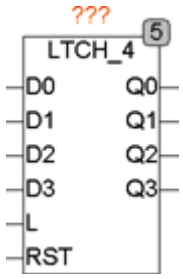


LTCH ist ein transparentes Speicherelement (Latch). Solange L TRUE ist, folgt Q dem Eingang D und mit der fallenden Flanke von L speichert der Ausgang Q das aktuelle Eingangssignal an D. Mit dem asynchronen Reset-Eingang kann das Latch jederzeit unabhängig von L gelöscht werden.



17.9. LATCH4

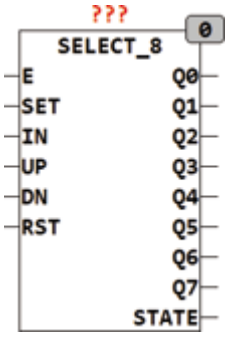
Type	Funktionsbaustein
Input	D0 .. D3 : BOOL (Data in)
	L : BOOL (Latch enable Signal)
	RST : BOOL (asynchroner Reset)
Output	Q0 .. Q3 : BOOL (Data Out)



LTCH4 ist ein transparentes Speicherelement (Latch). Solange L TRUE ist folgen Q0 - Q3 den Eingängen D0 - D3 und mit der fallenden Flanke von L speichern die Ausgänge Q0 - Q3 das aktuelle Eingangssignal von D0 - D3. Mit dem asynchronen Reset-Eingang kann das Latch jederzeit unabhängig von L gelöscht werden. Weitere Erläuterungen und Angaben finden Sie beim Baustein LTCH.

17.10. SELECT_8

Type	Funktionsbaustein
Input	E : BOOL (Enable für Ausgänge)
	SET : BOOL (asynchroner Set)
	IN : BYTE (Vorgabewert für Set)
	UP : BOOL (Vorwärts Schalter flankengetriggert)
	DN : BOOL (Rückwärts Schalter flankengetriggert)
	RST : BOOL (asynchroner Reset)
Output	Q0 .. Q7 : BOOL (Ausgänge)
	STATE : BYTE (Status Ausgang)



SELECT_8 setzt immer nur einen Ausgang auf TRUE solange E auf TRUE ist. Der aktive Ausgang Q0..Q7 kann mittels des SET Eingangs und dem Wert am Eingang IN Selektiert werden. Ein TRUE an SET und ein Wert von 5 am Eingang IN setzen den Ausgang Q5 auf TRUE während alle anderen Ausgänge auf FALSE gesetzt werden. Ein TRUE am Eingang RST setzt Ausgang Q0 auf TRUE. Mit den Eingängen UP wird von einem Ausgang Qn auf Qn+1 weiter geschaltet, während der Eingang DN von einen Ausgang Qn auf Qn-1 schaltet. Der Eingang EN muss TRUE sein damit ein Ausgang TRUE wird, ist EN FALSE werden alle Ausgänge FALSE. Ein FALSE an E beeinflusst aber nicht die Funktion der anderen Eingänge. So kann auch bei einem FALSE am Eingang EN mit UP oder DN hoch oder runter geschaltet werden. Die Eingänge UP und DN sind flankengetriggert und reagieren nur auf die steigende Flanke. Der Ausgang STATE zeigt immer an welcher Ausgang gerade selektiert ist.

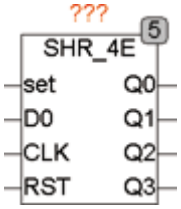
	E	SET	IN	UP	DN	RST	Q	STATE
Reset	X	-	-	-	-	1	Q0 if EN=1	0
Set	X	1	N	-	-	0	QN if EN=1	N
up	X	0	-	↑	0	0	QN+1 if EN=1	N + 1
down	X	0	-	0	↑	0	QN-1 if EN=1	N - 1

17.11. SHR_4E

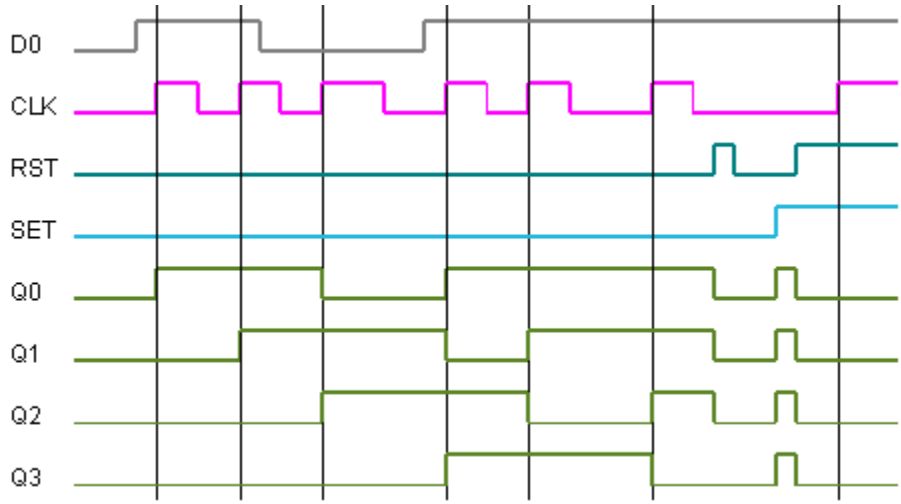
Type Funktionsbaustein

Input SET : BOOL (asynchroner Set)
D0 : BOOL (Data Input)
CLK : BOOL (Takteingang)
RST : BOOL (asynchroner Reset)

Output Q0 : BOOL (Data Out 0)
Q1 : BOOL (Data Out 1)
Q2 : BOOL (Data Out 2)
Q3 : BOOL (Data Out 3)

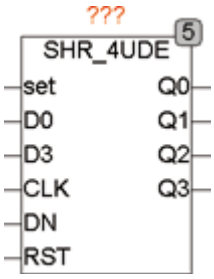


SHR_4E ist ein 4 Bit Shift Register mit asynchronem Set- und Reset-Eingang. Mit einer steigenden Flanke an CLK wird Q2 nach Q3 geschoben, Q1 nach Q2, Q0 nach Q1 und D0 nach Q0 gespeichert. Mit einem TRUE am Set-Eingang werden alle Ausgänge (Q0 .. Q3) auf TRUE gesetzt und mit RST werden alle auf FALSE gesetzt.

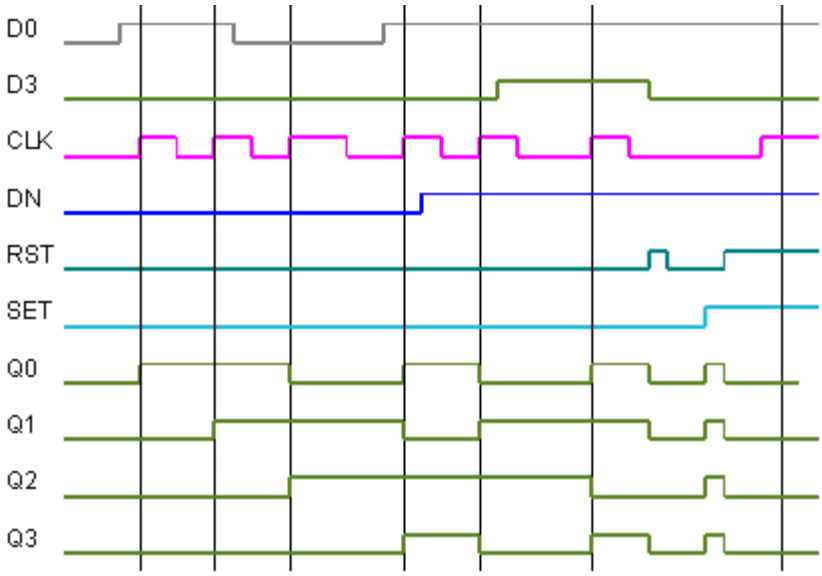


17.12. SHR_4UDE

Type	Funktionsbaustein
Input	SET : BOOL (asynchroner Set)
	D0 : BOOL (Data Input Bit 0)
	D3 : BOOL (Data Input Bit 3)
	CLK : BOOL (Takteingang)
	DN : BOOL (Steuereingang Up / Down, TRUE = Down)
	RST : BOOL (asynchroner Reset)
Output	Q0 : BOOL (Data Out 0)
	Q1 : BOOL (Data Out 1)
	Q2 : BOOL (Data Out 2)
	Q3 : BOOL (Data Out 3)



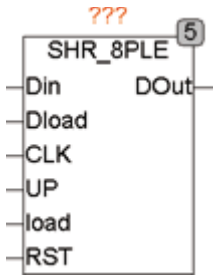
SHR_4UDE ist ein 4 Bit Schieberegister mit Up / Down Schieberichtungen. Mit einer steigenden Flanke an CLK wird Q2 nach Q3 geschoben, Q1 nach Q2, Q0 nach Q1 und D0 nach Q0 gespeichert. Die Schieberichtung kann mit einem TRUE am Eingang DN umgekehrt werden, dann wird D3 nach Q3 – nach Q2 – nach Q1 – nach Q0 geschoben. Mit einem TRUE am Set-Eingang werden alle Ausgänge (Q0 .. Q3) auf TRUE gesetzt und mit RST werden alle Eingänge auf FALSE gesetzt.



17.13. SHR_8PLE

Type	Funktionsbaustein
Input	DIN : BOOL (Shift Data Input)
	DLOAD : Byte (Datenwort zum Parallel Load)
	CLK : BOOL (Takteingang)
	UP : BOOL (Steuereingang Up / Down, TRUE = Up)
	LOAD : BOOL (Steuereingang zum Laden des Registers)
	RST : BOOL (asynchroner Reset)

Output DOUT : BOOL (Data Out)



SHR_8PLE ist ein 8 Bit Schieberegister mit Parallel Load und asynchronem Reset. Die Schieberichtung kann mit dem Eingang UP umgekehrt werden. Wenn UP=1, wird Bit 7 zuerst auf DOUT geschoben und wenn UP=0, wird Bit0 zuerst an DOUT geschoben. Für Up-Shift wird Bit 0 mit DIN geladen und bei Down-Shift wird Bit 7 mit DIN geladen. Am Eingang DLOAD liegt ein Byte Daten an, das bei Parallel Load (LOAD=1 und steigende Flanke an CLK) ins interne Register geladen wird. Im Falle von Parallel Load wird zuerst ein Shift durchgeführt und anschließend das Register geladen. Ein RST kann jederzeit asynchron das Register löschen. Eine eingehende Beschreibung eines Schieberegisters finden Sie beim Modul SHR_4E.

17.14. SHR_8UDE

Type Funktionsbaustein

Input SET : BOOL (asynchroner Set)

D0 : BOOL (Data Input Bit 0)

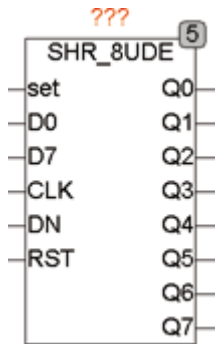
D3 : BOOL (Data Input Bit 3)

CLK : BOOL (Takteingang)

DN : BOOL (Steuereingang Up / Down, TRUE = Down)

RST : BOOL (asynchroner Reset)

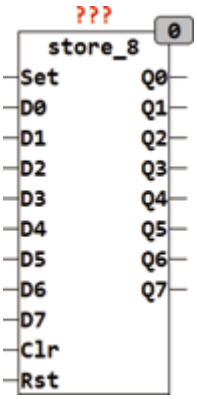
Output Q0 .. Q7 : BOOL (Data Out)



SHR_8UDE ist ein 8 Bit Schieberegister mit Up / Down Schieberichtung. Mit einer steigenden Flanke an CLK werden die Daten Q0 nach Q7 um jeweils einen Schritt geschoben. Q0 wird anschließend mit D0 geladen. Die Schieberichtung kann mit einem TRUE am Eingang DN umgekehrt werden. Dann wird D7 nach Q6, Q5, Q4, Q3, Q2, Q1, Q0 geschoben und Q7 mit D7 geladen. Mit einem TRUE am Set-Eingang werden alle Ausgänge (Q0 .. Q3) auf TRUE gesetzt und mit RST werden alle Ausgänge auf FALSE gesetzt. Weitergehende Erläuterungen zu Schieberegistern finden Sie unter SHR_4E und speziell beim Modul SHR_4UDE, welches die gleiche Funktion für 4 Bits wie SHR_8UDE für 8 Bits erfüllt.

17.15. STORE_8

Type	Funktionsbaustein
Input	SET : BOOL (asynchroner Set)
	D0..D7 : BOOL (Data Input Bit 0..7)
	CLR : BOOL (Schrittweise Rücksetzen Eingang)
	RST : BOOL (Asynchroner Set Eingang)
Output	Q0..Q7 : BOOL (Ereignis Ausgänge)

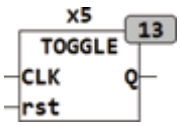


STORE_8 ist ein 8-fach Ereignisspeicher. ein TRUE an einem der Eingänge D0..D7 setzt den entsprechenden Ausgang Q0..Q7. Die Asynchronen Set und Reset Eingänge (SET, RST) setzen alle Ausgänge gleichzeitig auf TRUE oder FALSE. ist während eines Resets einer der Eingänge TRUE wird nach dem Reset der entsprechende Ausgang sofort wieder auf TRUE gesetzt. Falls flankengetriggerte Eingänge gewünscht werden, so sind vor dem Baustein STRORE_8 TP_R Bausteine einzusetzen. Dies erlaubt es dem Anwender sowohl flanken- wie auch zustands- getriggerte Eingänge gleichzeitig zu verwenden. Der Eingang CLR löscht mit einer steigenden Flanke an CLR immer nur ein Ereignis, beginnend mit dem am höchsten priorisierten Ausgang der gerade TRUE ist. Wird mit CLR ein Ausgang Q gelöscht

dessen Eingang D noch TRUE ist so wird der Ausgang D mit dem nächsten Zyklus wieder auf TRUE gesetzt.

17.16. TOGGLE

Type	Funktionsbaustein
Input	CLK: BOOL (Takteingang)
	RST : BOOL (asynchroner Reset)
Output	Q : BOOL (Ausgang)

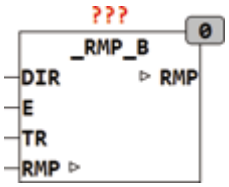


TOGGLE ist ein flankengetriggertes Toggle Flip-Flop mit asynchronem Reset-Eingang. Das TOGGLE Flip-Flop invertiert den Ausgang Q bei einer steigenden Flanke von CLK. Der Ausgang ändert bei jeder steigenden Flanke von CLK seinen Zustand.

18. Signalgeneratoren

18.1. _RMP_B

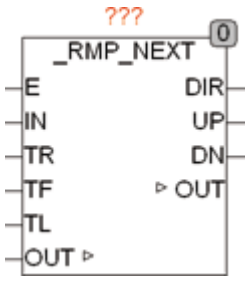
Type	Funktionsbaustein
Input	DIR : BOOL (Richtung, TRUE bedeutet Aufwärts)
	E : BOOL (Enable Eingang)
	TR : TIME (Zeit zum Durchlauf einer vollen Rampe)
I/O	RMP : BYTE (Ausgangssignal)



_RMP_B ist ein 8-Bit Rampen-Generator. Die Rampe wird in einer extern deklarierten Variable erzeugt. Die Rampe ist steigend wenn DIR = TRUE und fallend wenn DIR = FALSE. Erreicht die Rampe einen Endwert so bleibt der Generator auf diesem Wert stehen. Mit dem Eingang E kann die Rampe jederzeit angehalten werden, wenn E=TRUE läuft die Rampe. Der Eingang TR gibt an welche Zeit benötigt wird um die Rampe von 0 - 255 oder umgekehrt zu durchlaufen.

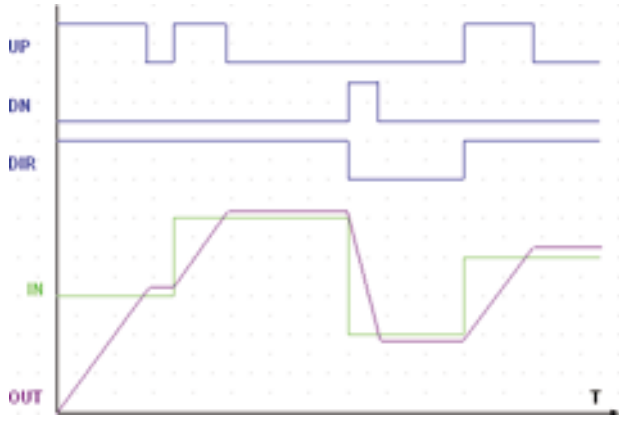
18.2. _RMP_NEXT

Type	Funktionsbaustein
Input	E : BOOL (Enable Eingang)
	IN : BOOL (Eingang)
	TR : TIME (Anstiegszeit für Rampe 0..255)
	TF : TIME (Abfallzeit für Rampe 255..0)
	TL : TIME (Sperrzeit zwischen einer Richtungsumkehr)
I/O	OUT : Byte (Ausgangssignal)
OUTPUT	DIR : BOOL (Richtung der Änderung an IN)
	UP : BOOL (signalisiert eine steigende Rampe)
	DN : BOOL (signalisiert eine fallende Rampe)



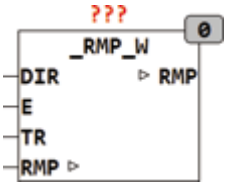
RMP_NEXT folgt am Ausgang OUT dem Eingangssignal IN mit durch TR und TF definierten steigenden- oder fallenden- Flanken. Im Gegensatz zu RMP_SOFT läuft die Flanke von RMP_NEXT solange bis sie den Endpunkt über- oder unter-schritten hat und ist deshalb auch für Regelungsaufgaben geeignet. Verändert sich der Wert von IN so wird eine steigende Rampe mit TR oder eine fallende Flanke mit TF am Ausgang OUT gestartet bis der Wert an OUT den Eingangswert von IN über- beziehungsweise unter-schritten hat. Der Ausgang bleibt dann auf diesen Wert stehen. Die Ausgänge UP und DN zeigen an ob gerade eine steigende oder eine fallende Flanke erzeugt wird. Der Ausgang DIR gibt die Richtung der Veränderung an IN an, verändert sich IN nicht bleibt dieser Ausgang auf dem letzten Zustand. Die Sperrzeit TL legt fest wie lange die Totzeit zwischen einer Richtungsumkehr ist.

Die folgende Graphik zeigt den Signalverlauf an OUT bei Änderung des Eingangsignals an IN:



18.3. _RMP_W

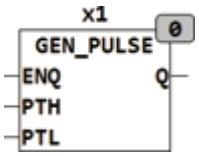
Type	Funktionsbaustein
Input	DIR : BOOL (Richtung, TRUE bedeutet Aufwärts)
	E : BOOL (Enable Eingang)
	TR : TIME (Zeit zum Durchlauf einer vollen Rampe)
I/O	RMP : WORD (Ausgangssignal)



_RMP_B ist ein 16-Bit Rampen-Generator. Die Rampe wird in einer extern deklarierten Variable erzeugt. Die Rampe ist steigend wenn DIR = TRUE und fallend wenn DIR = FALSE. Erreicht die Rampe einen Endwert so bleibt der Generator auf diesem Wert stehen. Mit dem Eingang E kann die Rampe jederzeit angehalten werden, wenn E=TRUE läuft die Rampe. Der Eingang TR gibt an welche Zeit benötigt wird um die Rampe von 0 - 65535 oder umgekehrt zu durchlaufen.

18.4. GEN_PULSE

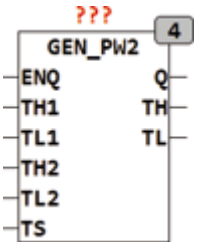
Type	Funktionsbaustein
Input	ENQ : BOOL (Enable Eingang)
	PTH : TIME (Impulsdauer HIGH)
	PTL : TIME (Impulsdauer LOW)
Output	Q : BOOL (Ausgangssignal)



GEM_PULSE erzeugt am Ausgang Q ein Ausgangssignal das für die Zeit PTH auf TRUE ist und anschließend für PTL LOW bleibt. Der Generator startet nach ENQ = TRUE immer mit einer steigenden Flanke an Q und bleibt für die Zeit PTH TURE. Solange ENQ = TRUE werden kontinuierliche Impulse am Ausgang Q erzeugt. Ist eine der Zeiten (PTH, PTL) oder beide gleich 0 so wird die Zeit auf einen SPS Zyklus begrenzt. GEN_PULSE(ENQ := TRUE, PTH := T#0s, PTL := T#0s) erzeugt ein Ausgangssignal das einen Zyklus TRUE ist und einen Zyklus FALSE . Der Default Wert für ENQ ist TRUE.

18.5. GEN_PW2

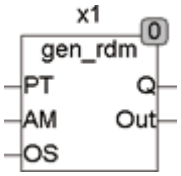
Type	Funktionsbaustein
Input	ENQ : BOOL (Enable Eingang)
	TH1 : TIME (Vorgabezeit HIGH wenn TS = LOW)
	TL1 : TIME (Vorgabezeit LOW wenn TS = LOW)
	TH2 : TIME (Vorgabezeit HIGH wenn TS = HIGH)
	TL2 : TIME (Vorgabezeit LOW wenn TS = HIGH)
	TS : BOOL (Auswahl für Ablaufzeiten)
Output	Q : BOOL (Binäres Ausgangssignal)
	TL : TIME (Ablaufzeit wenn Q = FALSE)
	TH : TIME (Ablaufzeit wenn Q = TRUE)



GEN_PW2 erzeugt ein Ausgangssignal mit einer definierbaren Zeit TH? für HIGH und TL für LOW. Mithilfe des Eingangs TS wird zwischen 2 Parametersätzen (TL1, TH1 und TL2, TH2) umgeschaltet. Beim Start oder nach einem ENQ = TRUE beginnt der Baustein mit der LOW Phase am Ausgang.

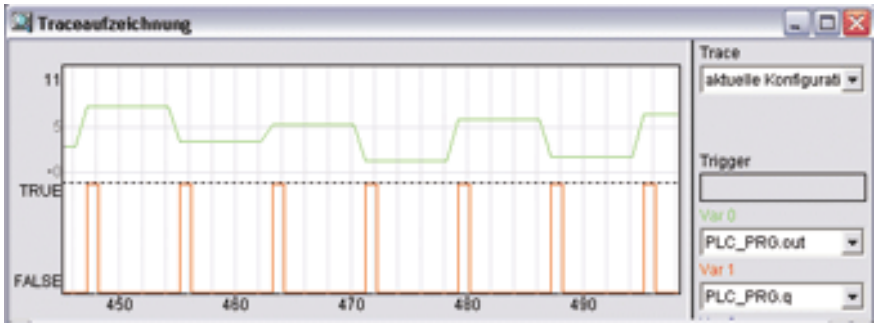
18.6. GEN_RDM

Type	Funktionsbaustein
Input	PT : TIME (Periodendauer)
	AM : REAL (Signal Amplitude)
	OS : REAL (Signal Offset)
Output	Q : BOOL (Binäres Ausgangssignal)
	OUT: REAL (Analoges Ausgangssignal)



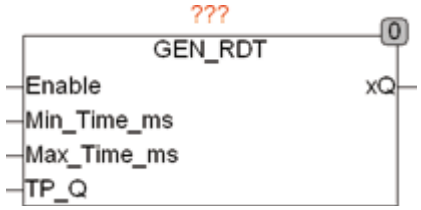
GEN_RDM ist ein Zufalls-Signalgenerator. Er erzeugt am Ausgang OUT einen neuen Wert in PT Zeitabständen. Der Ausgang Q wird für genau einen Zyklus TRUE, wenn sich der Ausgang OUT verändert hat. Der Eingang AM und OS legen die Amplitude und den Offset für den Ausgang OUT fest. Wenn die Eingänge OS und AM nicht beschaltet werden sind die Vorgabewerte 0 und 1.

Das folgende Beispiel zeigt eine Traceaufzeichnung für die Eingangswerte PT=100ms, AM=10 und OS=5. Der Ausgang erzeugt Werte alle 100ms im Bereich von 0 .. 10.



18.7. GEN_RDT

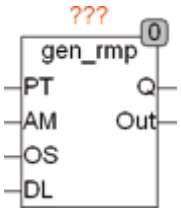
Type	Funktionsbaustein
Input	ENABLE : BOOL (Freigabeeingang)
	MIN_TIME_MS : TIME (Minimale Periodendauer)
	MAX_TIME_MS : TIME (Maximale Periodendauer)
	TP_Q : TIME (Pulsbreite des Ausgangspulses an XQ)
Output	XQ : BOOL (Binäres Ausgangssignal)



GEN_RDT erzeugt Impulse mit definierter Pulsbreite und Zufälligen Abstand. Die Ausgangsimpulse mit der Pulsbreite TP_Q werden in Zufälligen Zeitabständen TX erzeugt. TX schwankt zufällig zwischen der Zeit MIN_TIME_MS und MAX_TIME_MS. Der Baustein erzeugt nur Impulse an Ausgang XQ wenn der Eingang ENABLE auf TRUE ist.

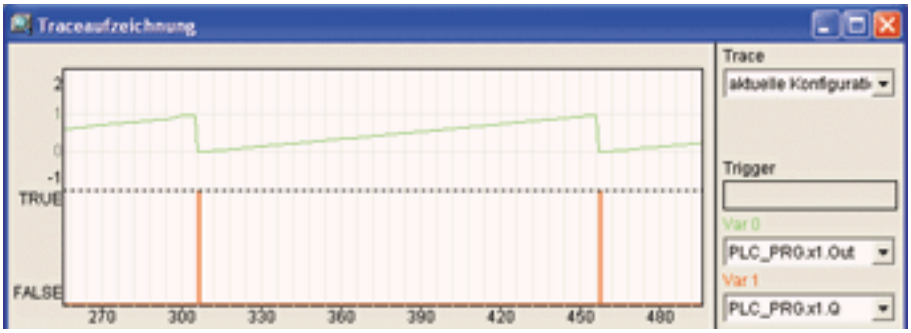
18.8. GEN_RMP

Type	Funktionsbaustein
Input	PT : TIME (Periodendauer)
	AM : REAL (Signal Amplitude)
	OS : REAL (Signal Offset)
	DL : REAL (Signal Verzögerung 0..1 * PT)
Output	Q : BOOL (Binäres Ausgangssignal)
	OUT : REAL (Analoges Ausgangssignal)



GEN_RMP ist ein Sägezahngenerator. Er erzeugt am Ausgang OUT eine Rampe mit der Dauer PT und wiederholt diese fortlaufend. Der Ausgang Q wird für genau einen Zyklus TRUE, wenn die Rampe am Ausgang OUT beginnt. Der Eingang AM und OS legen die Amplitude und den Offset für den Ausgang OUT fest. Wenn die Eingänge OS und AM nicht beschaltet werden sind die Vorgabewerte 0 und 1. Der Ausgang OUT erzeugt dann ein Sägezahnsignal von 0 .. 1. Der Eingang DL kann das Ausgangssignal um bis zu eine Periode (PT) verschieben und dient dazu, mehrere zueinander verschobene Signale, zu erzeugen. Eine 0 am Eingang DL bedeutet keine Verschiebung. Ein Wert zwischen 0 und 1 verschiebt das Signal um bis zu einer Periode.

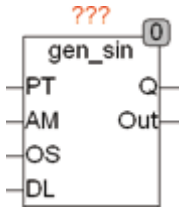
Das folgende Beispiel zeigt eine Traceaufzeichnung für die Eingangswerte PT = 10s, AM = 1 und OS = 0.



18.9. GEN_SIN

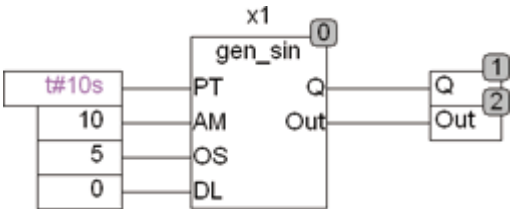
Type	Funktionsbaustein
Input	PT : TIME (Periodendauer)
	AM : REAL (Signal Amplitude)
	OS : REAL (Signal Offset)
	DL : REAL (Signal Verzögerung 0..1 * PT)
Output	Q : BOOL (Binäres Ausgangssignal)

OUT : REAL (Analoges Ausgangssignal)

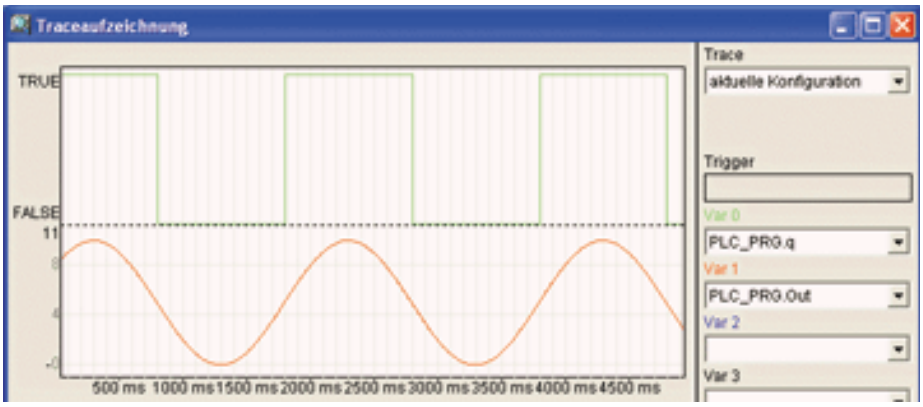


GEN_SIN ist ein Sinusgenerator mit programmierbarer Periodendauer, einstellbarer Amplitude und Signal Offset. Als Besonderheit kann auch noch ein Delay eingestellt werden, damit mit mehreren Generatoren überlappende Signale erzeugt werden können. Ein Binärer Ausgang Q stellt ein Logisches Signal zur Verfügung das Phasengleich mit dem Sinussignal erzeugt wird. Der Eingang DL gibt ein Delay für das Ausgangssignal vor. Das Delay wird spezifiziert mit DL * PT. Ein DL von 0.5 verzögert das Signal um eine halbe Periode.

Das folgende Beispiel zeigt GEN_SIN mit einer Traceaufzeichnung des Sinussignals und des binären Ausgangs Q.

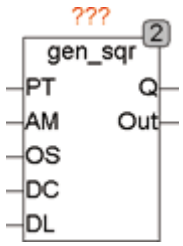


Obiges Beispiel Erzeugt ein Sinussignal mit 0.1 HZ (PT = 10 s) und einen unteren Spitzenwert von 0 und oberen Spitzenwert von 10.



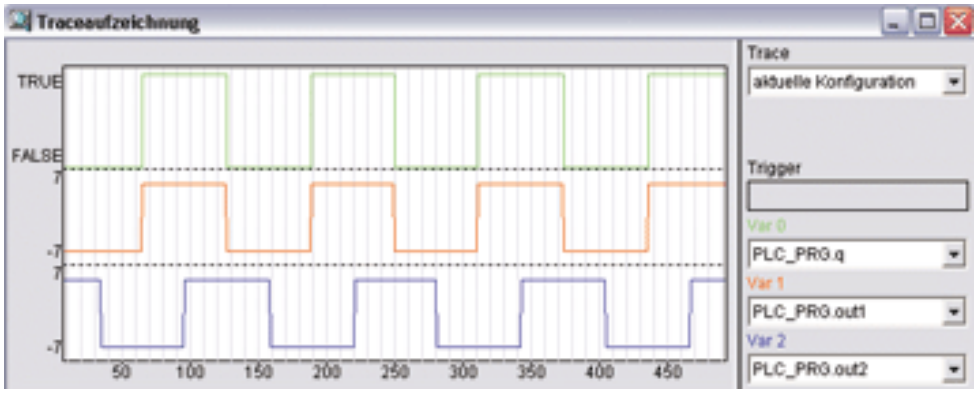
18.10. GEN_SQR

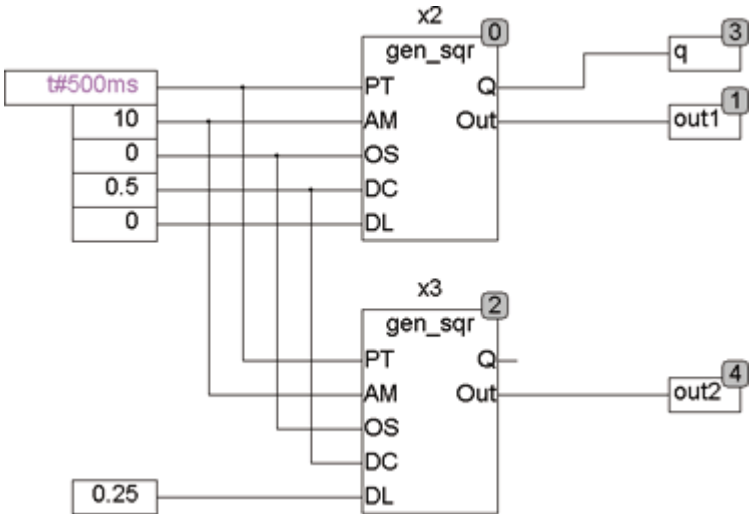
Type	Funktionsbaustein
Input	PT : TIME (Periodendauer)
	AM : REAL (Signal Amplitude)
	OS : REAL (Signal Offset)
	DC : REAL (Tastverhältnis 0..1)
	DL : REAL (Signal Verzögerung 0..1 * PT)
Output	Q : BOOL (Binäres Ausgangssignal)
	OUT : REAL (Analoges Ausgangssignal)



GEN_SQR ist ein Rechteckgenerator mit programmierbarer Periodendauer, einstellbarer Amplitude und Signal Offset sowie Tastverhältnis DC (Duty Cycle). Als Besonderheit kann auch noch ein Delay eingestellt werden, damit mit mehreren Generatoren überlappende Signale erzeugt werden können.

Das folgende Beispiel zeigt 2 GEN_SQR, wobei einer davon mit einem Delay von 0.25 (¼ Periode) läuft. In der Traceaufzeichnung ist deutlich das Signal des ersten Generators und das verzögerte Signal des zweiten Generators zu sehen.



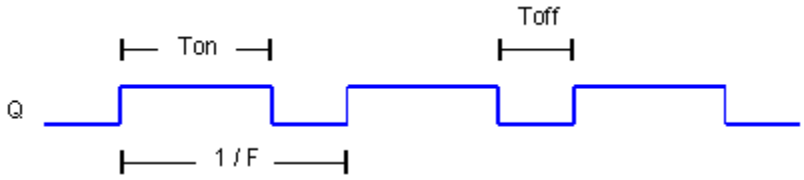


18.11. PWM_DC

Type	Funktionsbaustein
Input	F : REAL (Ausgangsfrequenz)
	DC : REAL (Tastverhältnis 0..1)
Output	Q : BOOL (Ausgangssignal)

PWM_DC ist ein Duty-Cycle modulierter Frequenzgenerator. Der Generator erzeugt eine feste Frequenz F mit einem Tastverhältnis (TON / TOFF), welche über den Eingang DC moduliert (Eingestellt) werden kann. Ein Wert von 0.5 am Eingang DC erzeugt ein Tastverhältnis von 50%.

Das Folgende Signalbild zeigt ein Ausgangssignal mit einem Duty-Cycle von 2 / 1, was einem DC (Tastverhältnis) von 0.67 entspricht.

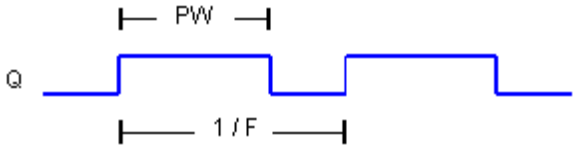


18.12. PWM_PW

Type	Funktionsbaustein

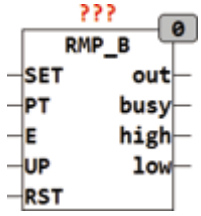
Input	F : REAL (Ausgangsfrequenz)
	PW : TIME (Impulsdauer High)
Output	Q : BOOL (Ausgangssignal)

PWM_PW ist ein Puls breiten modulierter Frequenzgenerator. Der Generator erzeugt eine feste Frequenz F mit einem Tastverhältnis (TON / TOFF) das über den Eingang PW moduliert (eingestellt) werden kann. Der Eingang gibt die Zeit vor die das Signal auf TRUE bleibt.



18.13. RMP_B

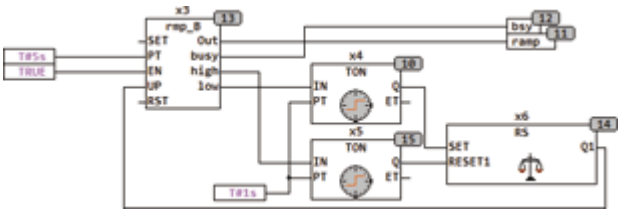
Type	Funktionsbaustein
Input	SET : BOOL (Set-Eingang)
	PT : TIME (Dauer einer Rampe 0..255)
	E : BOOL (Freigabeeingang)
	UP : BOOL (Richtung UP=TRUE bedeutet Up)
	RST : BOOL (Reset-Eingang)
Output	OUT : Byte (Ausgangssignal)
	BUSY : BOOL (TRUE, wenn Rampe läuft)
	HIGH : BOOL (Maximaler Ausgangswert ist erreicht)
	LOW : BOOL (Minimaler Ausgangswert ist erreicht)



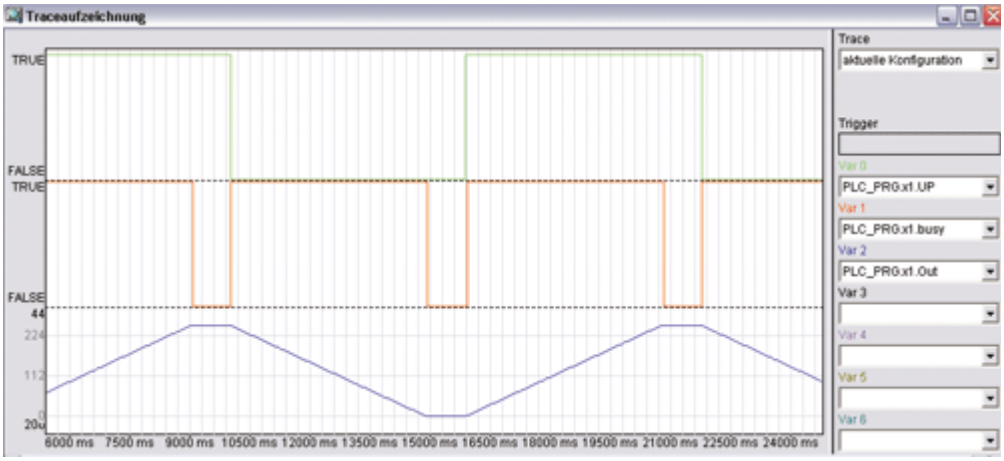
RMP_B ist ein Rampengenerator mit 8 Bit (1 Byte) Auflösung. Die Rampe von 0..255 wird in maximal 255 Schritte unterteilt und in einer Zeit von PT einmal komplett durchlaufen. Ein Freigabesignal E schaltet den Rampengenerator an oder aus. Ein asynchroner Reset setzt jederzeit den Ausgang auf 0 und ein Impuls am Set-Eingang setzt den Ausgang auf 255. Mit einem UD-Eingang kann die Richtung AUF (UD = TRUE) oder Ab (UD = FALSE) vorgegeben werden. Der Ausgang BUSY = TRUE zeigt an, dass eine Rampe aktiv ist. BUSY = FALSE bedeutet der Ausgang ist stabil. Die Ausgänge HIGH und LOW werden TRUE wenn der Ausgang OUT das untere oder obere Limit (0 bzw. 255) erreicht hat.

Beim festlegen von PT ist zu beachten, dass eine SPS mit 5ms Zykluszeit 256*5 =1275 Millisekunden für eine Rampe benötigt. Wird die Zeit PT kürzer als die Zykluszeit mal 256 gewählt, wird die Flanke in entsprechend größere Sprünge übersetzt. Die Rampe wird in diesen Fall aus weniger als 256 Schritten je Zyklus zusammengesetzt. PT darf T#0s sein, dann schaltet der Ausgang zwischen Minimal- und Maximal-Wert hin und her.

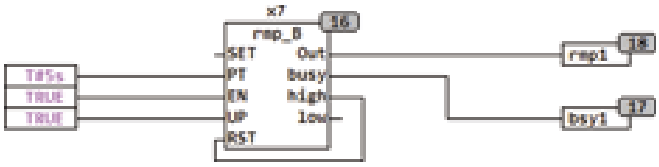
Das folgende Beispiel zeigt eine Anwendung von RMP_B. Die Ausgänge HIGH und LOW Triggern die beiden TON (X4, X5) jeweils 1 Sekunde verzögert und schalten über das RS Flip-Flop (X6) den UP-Eingang des Rampengenerators um. Das Ergebnis ist eine Rampe von 5 Sekunden, gefolgt von einer Pause von 1 Sekunde und dann die umgekehrte Rampe von 5 Sekunden und wieder eine Pause von 1 Sekunde. In der Traceaufzeichnung ist der Verlauf der Signale zu erkennen.



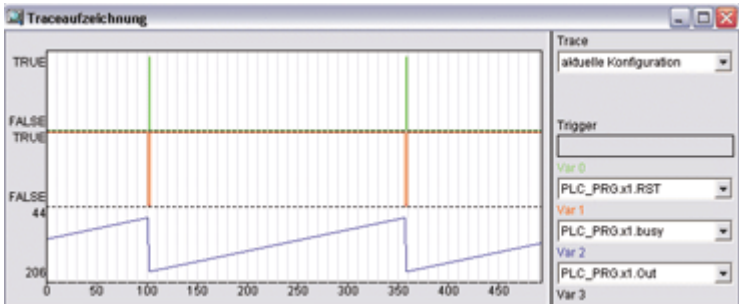
Timing Diagramm für Up / Down Rampe:



Ein weiteres Beispiel zeigt den Einsatz von RMP_B als Sägezahngenerator.



Timing Diagramm für Sägezahngenerator:

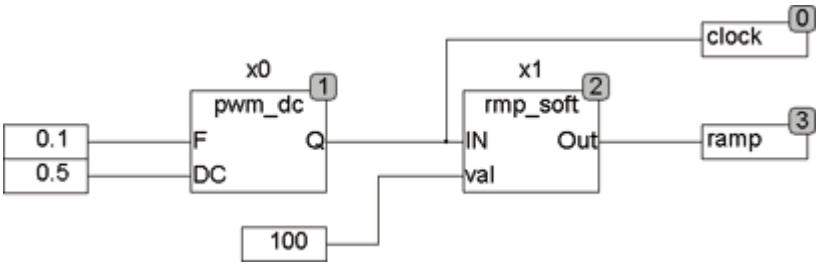


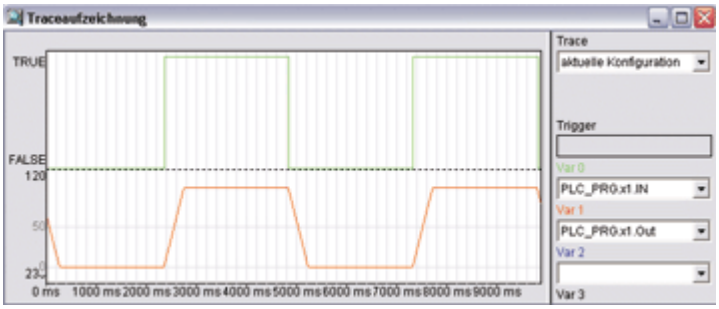
18.14. RMP_SOFT

Type	Funktionsbaustein
Input	IN : BOOL (Freigabeeingang)
	VAL : Byte (Maximaler Ausgangswert)
Setup	PT_ON : TIME (Anstiegszeit, Default ist 100 ms)
	PT_OFF : TIME (Abfallzeit, Default ist 100 ms)
Output	OUT : Byte (Ausgangssignal)

RMP_SOFT glättet die Rampen eines Eingangssignals VAL. Das Signal Out folgt dem Eingangssignal VAL, wobei sowohl Anstiegszeit wie auch Abfallzeit durch PT_ON und PT_OFF begrenzt werden können. Die Anstiegszeit und Abfallzeit der Rampen sind durch Setup-Parameter im Modul RMP_SOFT definiert. Die Setup-Zeit PT_ON Gibt an, wie lange die Rampe von 0..255 dauert. Eine Rampe die durch VAL begrenzt ist, ist entsprechend kürzer. PT_OFF definiert entsprechend die fallende Rampe. Wird der Eingang IN auf FALSE gesetzt entspricht dies einem VAL Wert von 0, somit kann durch schalten des Eingangs IN zwischen 0 und VAL umgeschaltet werden.

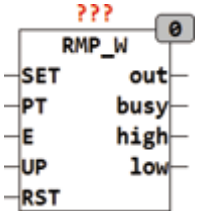
Beispiel:





18.15. RMP_W

Type	Funktionsbaustein
Input	SET : BOOL (Set Eingang)
	PT : TIME (Dauer einer Rampe 0..65535)
	E : BOOL (Freigabeeingang)
	UP : BOOL (Richtung UP=TRUE bedeutet Up)
	RST : BOOL (Reset Eingang)
Output	OUT : Byte (Ausgangssignal)
	BUSY : BOOL (TRUE, wenn Rampe läuft)
	HIGH : BOOL (Maximaler Ausgangswert ist erreicht)
	LOW : BOOL (Minimaler Ausgangswert ist erreicht)



RMP_W ist ein Rampengenerator mit 16 Bit (2 Byte) Auflösung. Die Rampe von 0.. 65535 wird in maximal 65536 Schritte unterteilt und in einer Zeit von PT einmal komplett durchlaufen. Ein Freigabesignal E schaltet den Rampengenerator an oder aus. Ein asynchroner Reset setzt jederzeit den Ausgang auf 0 und ein Impuls am Set-Eingang setzt den Ausgang auf 65535. Mit einem UD-Eingang kann die Richtung AUF (UD = TRUE) oder Ab (UD = FALSE) vorgegeben werden. Der Ausgang BUSY = TRUE zeigt an, dass eine Rampe aktiv ist. BUSY = FALSE bedeutet der Ausgang ist stabil. Die Ausgänge HIGH und LOW werden TRUE wenn der Ausgang OUT das untere oder obere Limit (0 bzw. 65535) erreicht hat.

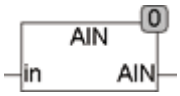
Beim festlegen von PT ist zu beachten, dass eine SPS mit 5ms Zykluszeit 65536*5 = 327 Sekunden für eine Rampe benötigt. Wird die Zeit PT kürzer als die Zykluszeit mal 65536 gewählt, wird die Flanke in entsprechend größere Sprünge übersetzt. Die Rampe wird in diesen Fall aus weniger als 65536 Schritten je Zyklus zusammengesetzt. PT darf T#0s sein, dann schaltet der Ausgang zwischen Minimal- und Maximal-Wert hin und her.

Eine detaillierte Beschreibung finden Sie beim Modul RMP_B. Die Funktion ist absolut identisch mit der Ausnahme, dass der Ausgang OUT 8 Bit anstelle von 16 Bit weit ist.

19. Signalverarbeitung

19.1. AIN

Type	Funktion
Input	IN : DWORD (Eingang vom A/D Wandler)
Output	REAL (Ausgangswert)
Setup	BITS : Byte (Anzahl der Bits, 16 für ein komplettes Wort)
	SIGN : Byte (Sign Bit, 15 für Bit 15)
	LOW : REAL (kleinster Wert des Ausgangs)
	HIGH : REAL (größter Wert des Ausgangs)



Analoge Eingänge von A/D Wandlern liefern in der Regel ein WORD (16 Bit) oder DWORD (32 Bit), wobei sie selbst meist nicht 16 Bit oder 32 Bit Auflösung besitzen. Weiterhin digitalisieren A/D Wandler einen festen Eingangsbereich (z B. -10 .. + 10 V), was zum Beispiel mit den digitalen Werten 0 .. 65535 (Bei 16 Bit) repräsentiert wird. Die Funktion AIN wird durch Setup-Parameter Konfiguriert und rechnet die Ausgangswerte des A/D Wandlers entsprechend um, sodass nach dem Modul AIN ein REAL-Wert zur Verfügung steht, der mit dem echten gemessenen Wert übereinstimmt. Weiterhin kann das Modul ein Sign-Bit an beliebiger Stelle extrahieren und umrechnen. Durch einen Doppelklick auf den Baustein können mehrere Setup-Variablen gesetzt werden. Bits definiert wie viele Bits des Eingangs-DWORD verarbeitet werden sollen. Für einen 12 Bit Wandler ist dieser Wert 12. Es werden dann nur die Bits 0 – 11 ausgewertet. Sign definiert, ob ein Vorzeichenbit vorhanden ist und wo im Eingangswort es zu finden ist. Sign=255 bedeutet, dass kein Vorzeichenbit vorhanden ist und 15 bedeutet, dass Bit 15 im DWORD das Vorzeichen enthält. Der Vorgabewert für SIGN ist 255. LOW und HIGH definieren den kleinsten und höchsten Ausgangswert. Ist ein Sign-Bit definiert (SIGN < 255), dann müssen LOW und HIGH positiv sein. Ohne Sign-Bit können Sie sowohl positiv als auch negativ sein.

Beispiel:

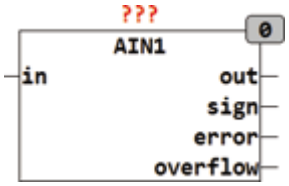
Ein 12 Bit A/D Wandler ohne Vorzeichen und Eingangsbereich 0 – 10 wird wie folgt definiert: Bits=12, Sign=255, LOW=0, HIGH=10.

Ein 14 Bit A/D Wandler mit Vorzeichen in Bit 14 und Eingangsbereich -10 - +10 wird folgendermaßen definiert: Bits=14, Sign=14, LOW=0, HIGH=+10.

Ein 24 Bit A/D Wandler ohne Vorzeichen und Eingangsbereich -10 - +10 wird folgendermaßen definiert: Bits=24, Sign=255, LOW=-10, HIGH=+10.

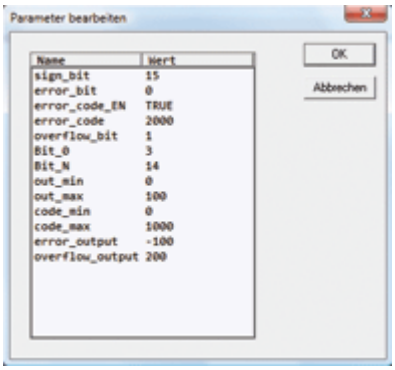
19.2. AIN1

Type	Funktionsbaustein
Input	IN : DWORD (Eingang vom A/D Wandler)
Output	OUT :REAL (Ausgangswert)
	SIGN : BOOL (Vorzeichen)
	ERROR : BOOL (Error Bit)
	OVERFLOW : BOOL (Overflow Bit)
Setup	SIGN_BIT : INT (Bitnummer des Vorzeichens)
	ERROR_BIT : INT (Bitnummer des Fehlerbits)
	ERROR_CODE_EN : BOOL (Auswertung des Error Codes Ein)
	ERROR_CODE : DWORD (Fehlercode des Eingangs IN)
	OVERFLOW_BIT : INT (Bitnummer des Overflow Bits)
	OVERFLOW_CODE_EN : BOOL (Overflow Codes Auswertung Ein)
	OVERFLOW_CODE : DWORD (Overflow Code des Eingangs IN)
	BIT_0 : INT (Bitnummer des niederwertigsten Datenbits)
	BIT_N : INT (Bitnummer des höchstwertigsten Datenbits)
	OUT_MIN : REAL (Ausgangswert bei CODE_MIN)
	OUT_MAX : REAL (Ausgangswert bei CODE_MAX)
	CODE_MIN : DWORD (Minimaler Eingangswert)
	CODE_MAX : DWORD (Maximaler Eingangswert)
	ERROR_OUTPUT : REAL (Ausgangswert der bei ERROR)
	OVERFLOW_OUTPUT: REAL (Ausgangswert der Bei OVERFLOW)



AIN1 setzt den Digitalen Ausgangswert eines A/D Wandlers in einen dem Messwert entsprechenden REAL Wert um. Der Baustein kann mittels Setup Variablen auf die unterschiedlichsten Digitalwandler angepasst werden.

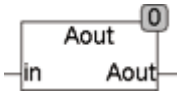
Ein SIGN_BIT legt fest an welchen Bit der D/A Wandler das Vorzeichen



übermittelt. Wird diese Variable nicht definiert oder auf einen Wert größer 31 gesetzt so wird kein Vorzeichen ausgewertet. Der Inhalt des SIGN_BIT wird am Ausgang SIGN angezeigt. Wird ein ERROR_BIT spezifiziert, wird der Inhalt des Error Bits am Ausgang ERROR angezeigt. Manche A/D Wandler liefern anstatt eines Error Bits einen festgelegten Ausgangswert der außerhalb des spezifizierten Messbereichs liegt und Signalisieren dadurch einen Fehler. Mit der Setup Variablen ERROR_CODE wird der entsprechende Error Code spezifiziert und mit ERROR_CODE_EN wird die Auswertung des ERROR_CODE Festgelegt. Wenn ERROR = TRUE wird am Ausgang OUT der Wert ERROR_OUTPUT ausgegeben. Mithilfe des OVERFLOW_BITS wird eine Bereichsüberschreitung des D/A Wandlers signalisiert und am Ausgang OVERFLOW ausgegeben. Mithilfe der Setup Variablen OVERFLOW_CODE_EN und OVERFLOW_CODE kann ein bestimmter Code am Eingang IN abgefragt werden und bei Auftreten dieses Codes das Overflow Bit gesetzt werden. Zusätzlich zum OVERFLOW_BIT kann mittels CODE_MIN und CODE_MAX ein zulässiger Bereich für die Eingangsdaten spezifiziert werden. Wird dieser Bereich über- beziehungsweise unterschritten wird ebenfalls der OVERFLOW Ausgang gesetzt. Bei einem Überlauf wird am Ausgang OUT der Wert OVERFLOW_OUTPUT ausgegeben. Die Setup Variablen BIT_0 und BIT_N legen fest wie der D/A Wandler den Messwert Bereitstellt. Mit Bit_0 wird festgelegt bei welchem Bit das Datenwort beginnt und mit BIT_N an welchem Bit das Datenwort endet. Im obigen Beispiel wird das Datenwort von Bit 3 bis Bit 14 übertragen (Bit 3 = Bit 0 des Datenwortes und Bit 14 = Bit 12 des Datenwortes). Das empfangene Datenwort wird entsprechend den Setup Variablen CODE_MIN, CODE_MAX und OUT_MIN, OUT_MAX umgerechnet und falls ein Vorzeichen vorhanden ist wird wenn SIGN = TRUE der Ausgangswert an OUT invertiert.

19.3. AOUT

Type	Funktion
Input	IN : REAL (Eingangswert)
Output	DWORD (Ausgangswort zum A/D Wandler)
Setup	BITS : Byte (Anzahl der Bits, 16 für ein komplettes Wort)
	SIGN : Byte (Sign Bit, 15 für Bit 15)
	LOW : REAL (kleinster Wert des Eingangs)
	HIGH : REAL (größter Wert des Eingangs)



Eingänge von D/A Wandlern benötigen in der Regel ein WORD (16 Bit) oder DWORD (32 Bit), wobei Sie selbst meist nicht 16 Bit oder 32 Bit Auflösung haben. Weiterhin erzeugen D/A Wandler einen festen Ausgangsbereich (z B. -10 .. + 10 V) was zum Beispiel mit den digitalen Werten 0 .. 65535 (Bei 16 Bit) repräsentiert wird. Die Funktion AOUT wird durch Setup-Parameter konfiguriert und rechnet die Eingangswerte (IN) entsprechend um, sodass nach dem Modul AOUT ein digitaler Wert zur Verfügung steht, der am Ausgang des D/A Wandlers einen Wert erzeugt, der mit dem REAL-Wert IN übereinstimmt. Weiterhin kann das Modul ein Sign-Bit an beliebiger Stelle einfügen falls der D/A Wandler ein Sign-Bit benötigt. Durch einen Doppelklick auf den Baustein können mehrere Setup-Variablen gesetzt werden. Bits definiert wie viele Bits der D/A Wandler verarbeiten kann. Für einen 12 Bit Wandler ist dieser Wert 12. Es werden dann nur die Bits 0 – 11 belegt. Sign definiert, ob ein Vorzeichenbit benötigt wird und wo im Ausgangs-DWORD es zu platzieren ist. Sign=255 bedeutet, dass kein Vorzeichenbit benötigt wird, und 15 bedeutet das Bit 15 im DWORD das Vorzeichen enthält. LOW und HIGH definieren den kleinsten und höchsten Eingangswert. Ist ein Sign-Bit definiert, so müssen LOW und HIGH positiv sein. Ohne Sign-Bit können Sie sowohl positiv als auch negativ sein.

Beispiele:

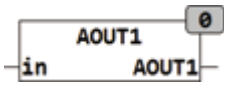
Ein 12 Bit D/A Wandler ohne Vorzeichen und Ausgangsbereich 0 – 10 wird wie folgt definiert: Bits=12, Sign=255, LOW=0, HIGH=10.

Ein 14 Bit D/A Wandler mit Vorzeichen in Bit 14 und Ausgangsbereich -10 - +10 wird folgendermaßen definiert: Bits=14, Sign=14, LOW=0, HIGH=+10.

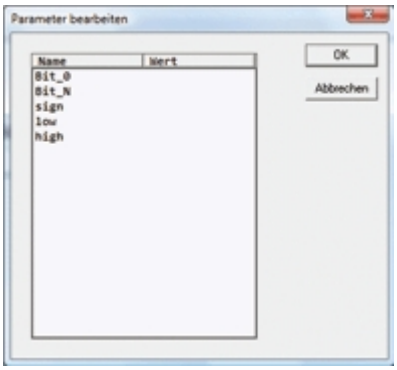
Ein 24 Bit D/A Wandler ohne Vorzeichen und Ausgangsbereich -10 - +10 wird folgendermaßen definiert: Bits=24, Sign=255, LOW=-10, HIGH=+10.

19.4. AOUT1

Type	Funktion
Input	IN : REAL (Eingangswert)
Output	DWORD (Ausgangswort zum A/D Wandler)
Setup	BIT_0 : INT (Stelle des niederwertigesten Bits des Datenwortes)
	BIT_N : INT (Stelle des höchstwertigsten Bits des Datenwortes)
	SIGN : INT (Sign Bit, 15 für Bit 15)
	LOW : REAL (kleinster Wert des Eingangs)
	HIGH : REAL (größter Wert des Eingangs)

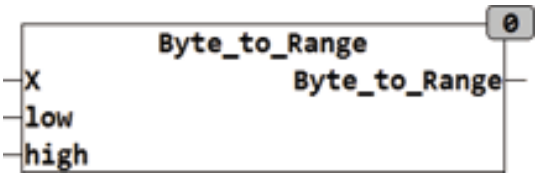


AOUT1 erzeugt aus dem REAL Eingangswert IN einen digitalen Ausgangswert für D/A Wandler oder andere Ausgangsbausteine die Digitale Daten verarbeiten. Mittels Setup Variablen kann der digitale Ausgangswert an verschiedenste Bedürfnisse angepasst werden. Der Eingangswert IN wird mittels den Angaben in LOW und HIGH sowie der mit BIT_0 und BIT_N spezifizierten Länge Des Datenwortes umgewandelt und am Ausgang bereitgestellt. BIT_0 spezifiziert die Position des niderwertigsten (Bit0) Datenbits in den Ausgangsdaten und BIT_N spezifiziert die Position des höchstwertigen Datenbits in den Ausgangsdaten. Die Länge des Datenbereichs wird durch BIT_N - BIT_0 + 1 automatisch errechnet. Wenn mit SIGN die Position eines Vorzeichen Bits angegeben wird so wird das Vorzeichen aus dem Eingangswert auf die spezifizierte Position von SIGN in den Ausgangsdaten kopiert.



19.5. BYTE_TO_RANGE

Type	Funktion
Input	IN : BYTE (Eingangswert)
	LOW : REAL (Ausgangswert bei X = 0)
	HIGH : REAL (Ausgangswert bei X = 255)
Output	REAL (Ausgangswert)



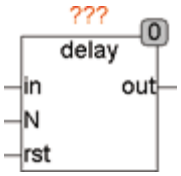
BYTE_TO_RANGE wandelt einen BYTE Wert in einen REAL. Ein Eingangswert von 0 entspricht dabei dem REAL Wert von LOW und eine Eingangswert von 255 entspricht dem Eingangswert von HIGH.

Um einen BYTE Wert von 0..255 in einen Prozentwert von 0..100 zu Wandeln wird der Baustein wie folgt aufgerufen:

BYTE_TO_RANGE(X,0,100)

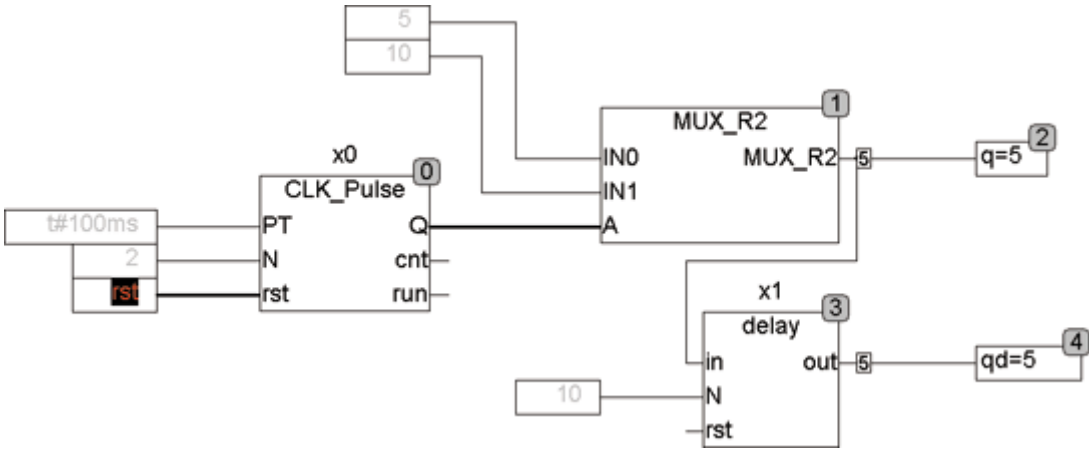
19.6. DELAY

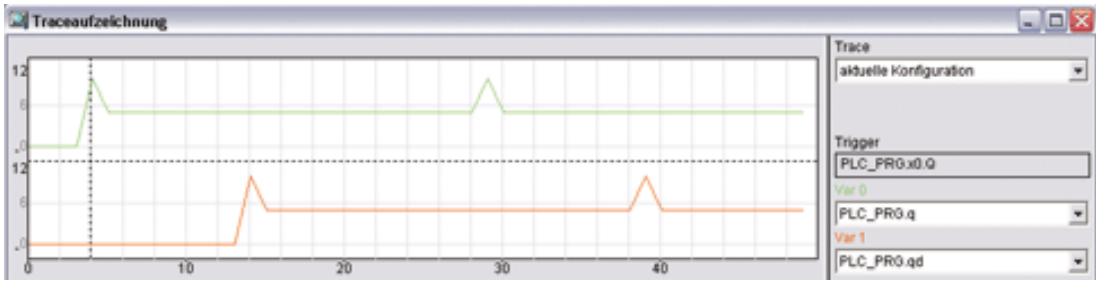
Type	Funktionsbaustein
Input	IN : REAL (Eingangswert)
	N : INT (Anzahl der Verzögerungs Zyklen)
	RST : BOOL (asynchroner Reset)
Output	OUT : REAL (verzögerter Ausgangswert)



DELAY verzögert ein Eingangssignal (in) um N Zyklen. Der Eingang Reset ist asynchron und kann jederzeit den Delay Puffer löschen.

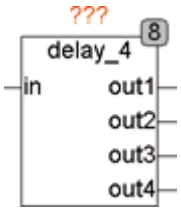
Das Beispiel zeigt einen Generator, der Impulse von 5 nach 10 erzeugt und ein Delay, das 10 Zyklen Verzögerung erzeugt.





19.7. DELAY_4

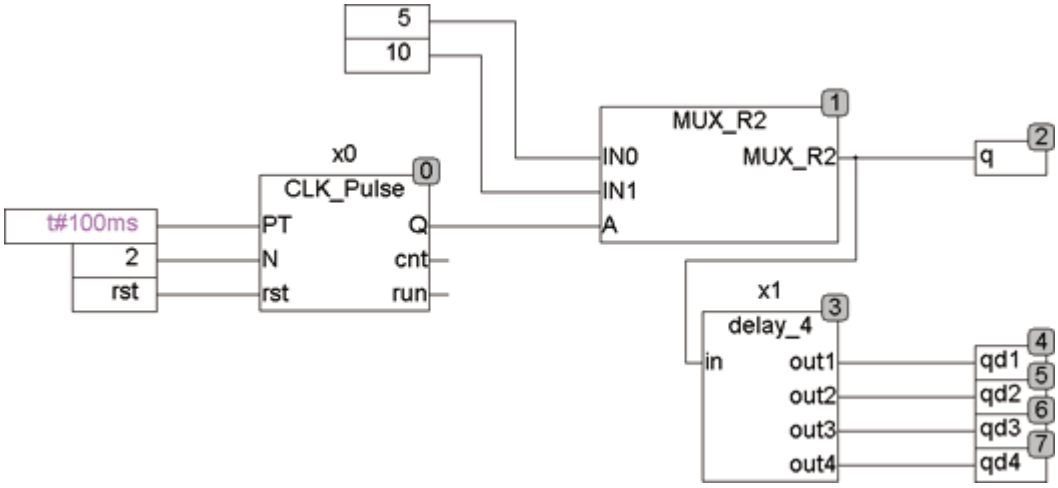
Type	Funktionsbaustein
Input	IN : REAL (Eingangswert)
Output	OUT1 : REAL (um 1 Zyklus verzögerter Ausgangswert)
	OUT2 : REAL (um 2 Zyklen verzögerter Ausgangswert)
	OUT3 : REAL (um 3 Zyklen verzögerter Ausgangswert)
	OUT4 : REAL (um 4 Zyklen verzögerter Ausgangswert)

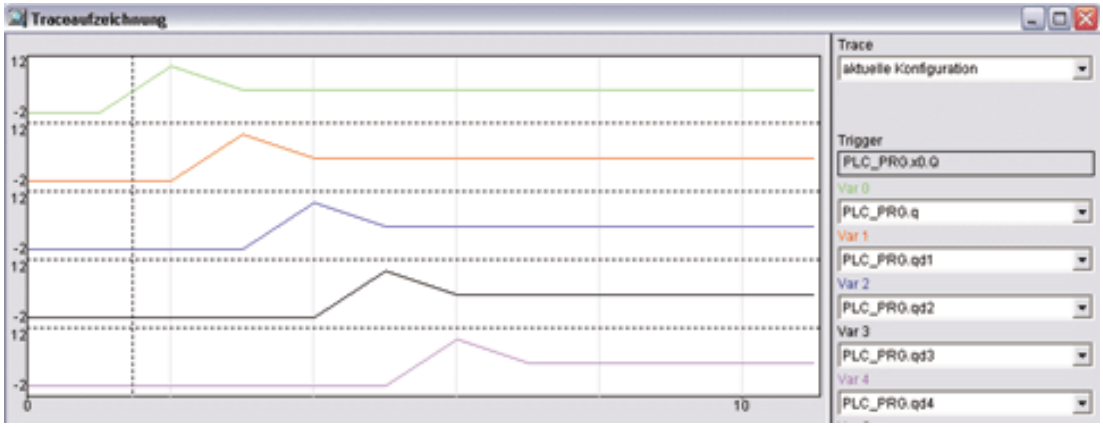


DELAY_4 verzögert eine Eingangssignal um maximal 4 Zyklen. An den Ausgängen Out1..4 stehen die letzten 4 Werte zur Verfügung. Out1 ist um

einen Zyklus verzögert, Qut2 um 2 Zyklen, Out3 um 3 Zyklen und Out4 um 4 Zyklen.

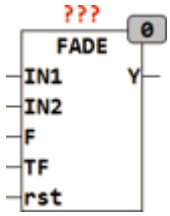
Beispiel:





19.8. FADE

Type	Funktionsbaustein
Input	IN1 : REAL (Eingangswert 1)
	IN2 : REAL (Eingangswert 2)
	F : BOOL (Auswahl Eingang TRUE = IN2)
	TF : TIME (Überblendzeit)
	RST : BOOL (Asynchroner Reset)
Output	Y : REAL (Ausgangswert)



FADE wird benutzt um zwischen 2 Eingängen IN1 und IN2 mit einem weichen Übergang umzuschalten. Die Umschaltzeit wird dabei mit TF angegeben. Ein asynchroner Reset (RST) setzt den Baustein ohne Verzögerung auf IN1 wenn F = FALSE oder auf IN2 wenn F = TRUE. Ein Umschaltvorgang wird durch eine Wertänderung an F ausgelöst. Anschließend wird innerhalb der Zeit TF zwischen den beiden Eingängen Umgeschaltet. Die Umschaltung erfolgt indem während der Umschaltzeit die beiden Eingänge gemischt werden. Am Anfang der Umschaltzeit stehen am Ausgang 0% des neuen Wertes und 100% des alten Wertes an. nach der halben Umschaltzeit (TF/2) ist der Ausgang jeweils 50% der beiden Eingangswerte (Y = in1*0.5 + in2*0.5). nach Ablauf der Zeit TF liegt dann am Ausgang der neue Wert zu 100% an.

Während der Umschaltzeit beträgt der Ausgang Y:

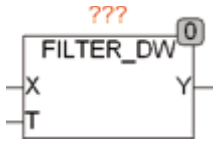
Y = TU/TF * IN1 + (1 - TU/TF) * IN2.

TU ist dabei die seit Beginn der Umschaltung vergangene Zeit.

Da der Ausgang von FADE dynamisch berechnet wird kann der Baustein auch zur Umschaltung von dynamischen Signalen verwendet werden. Die Umschaltung wird in bis zu 65535 Stufen eingeteilt, die jedoch durch die Zykluszeit der SPS begrenzt werden können. Eine SPS mit einer Zykluszeit von 10ms und einer TF von einer Sekunde wird lediglich in 1s/10ms = 100 Stufen Umschalten.

19.9. FILTER_DW

Type	Funktion : DWORD
Input	X : DWORD (Eingangswert)
	T : TIME (Zeitkonstante des Filters)
Output	Y : DWORD (gefilterter Wert)



FILTER_DW ist ein Filter ersten Grades für 32 Bit DWORD Daten. Die Hauptanwendung ist das Filtern von Sensorsignalen zur Rauschunterdrückung. Die grundlegende Funktionalität eines Filters ersten Grades kann beim Baustein FT_PT1 nachgelesen werden.

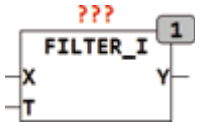
19.10. FILTER_I

Type Funktion : INT

Input X : DWORD (Eingangswert)

T : TIME (Zeitkonstante des Filters)

Output Y : DWORD (gefilterter Wert)



FILTER_I ist ein Filter ersten Grades für 16 Bit INT Daten. Die Hauptanwendung ist das Filtern von Sensorsignalen zur Rauschunterdrückung. Die grundlegende Funktionalität eines Filters ersten Grades kann beim Baustein FT_PT1 nachgelesen werden.

19.11. FILTER_MAV_DW

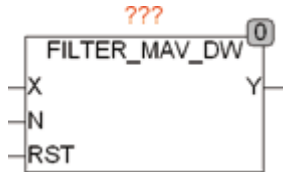
Type Funktion : DWORD

Input X : DWORD (Eingangswert)

N : UINT (Anzahl der ermittelten Werte)

RST : BOOL (asynchroner Reset Eingang)

Output Y : DWORD (gefilterter Wert)



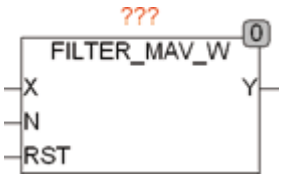
FILTER_MAV_DW ist ein Filter mit gleitendem Mittelwert. Beim Filter mit gleitendem Mittelwert (auch Moving Average Filter genannt) wird der Mittelwert von N aufeinander folgenden Messwerten als Mittelwert ausgegeben.

Y:= (X0 + X1 + ... + Xn-1) / N

X0 ist der Wert X im momentanen Zyklus, X1 ist der Wert im Zyklus davor usw. Die Anzahl der Werte über die der Mittelwert gebildet werden soll wird am Eingang N spezifiziert, der Wertebereich von N liegt zwischen 1 und 32.

19.12. FILTER_MAV_W

Type	Funktion : WORD
Input	X : WORD (Eingangswert)
	N : UINT (Anzahl der ermittelten Werte)
	RST : BOOL (asynchroner Reset Eingang)
Output	Y : WORD (gefilterter Wert)



FILTER_MAV_W ist ein Filter mit gleitendem Mittelwert. Beim Filter mit gleitendem Mittelwert (auch Moving Average Filter genannt) wird der Mittelwert von N aufeinander folgenden Messwerten als Mittelwert ausgegeben.

Y:= (X0 + X1 + ... + Xn-1) / N

X0 ist der Wert X im momentanen Zyklus, X1 ist der Wert im Zyklus davor usw. Die Anzahl der Werte über die der Mittelwert gebildet werden soll wird am Eingang N spezifiziert, der Wertebereich von N liegt zwischen 1 und 32.

19.13. FILTER_W

Type	Funktion : WORD
Input	X : WORD (Eingangswert)
	T : TIME (Zeitkonstante des Filters)
Output	Y : WORD (gefilterter Wert)

FILTER_W ist ein Filter ersten Grades für 16 Bit WORD Daten. Die Hauptanwendung ist das Filtern von Sensorsignalen zur Rauschunterdrückung. Die grundlegende Funktionalität eines Filters ersten Grades kann beim Baustein FT_PT1 nachgelesen werden.

19.14. FILTER_WAV

Type	Funktion : REAL
Input	X : DWORD (Eingangswert)
	W : ARRAY[0..15] of REAL (Gewichtungsfaktoren)
	RST : BOOL (asynchroner Reset Eingang)
Output	Y : REAL (gefilterter Wert)

FILTER_WAV ist ein Filter mit gewichtetem Mittelwert. Beim Filter mit gewichteten Mittelwert (auch FIR Filter genannt) werden die einzelnen Werte im Puffer mit unterschiedlicher Gewichtung bewertet.

Y:= X0 * W0 + X1 * W1 + ...+ X15 * W15

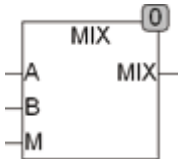
X0 ist der Wert X im momentanen Zyklus, X1 ist der Wert im Zyklus davor usw. Die Faktoren W werden als Array dem Eingang W übergeben. Bei der Anwendung des FIR Filters ist darauf zu achten, dass geeignete Faktoren für die Gewichtung eingesetzt werden. Die Anwendung macht nur dann Sinn wenn diese Faktoren mit geeigneten Methoden oder Entwurfssoftware ermittelt werden.

19.15. MIX

Type	Funktion : REAL
Input	A : REAL (Eingangswert 1)
	B : REAL (Eingangswert 2)

M : REAL (Mischungsverhältnis)

Output REAL (Wert aus dem Mischungsverhältnis M zwischen A und B)



MIX stellt am Ausgang einen mit dem Mischungsverhältnis M gemischten Wert aus A und B zur Verfügung. Der Eingang M gibt den Anteil vom B in Bereich 0..1 an.

MIX = (M-1)*A + M*B

19.16. MUX_R2

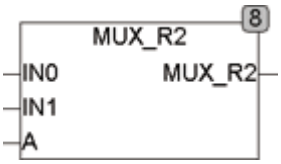
Type Funktion

Input IN0 : REAL (Eingangswert 0)

IN1 : REAL (Eingangswert 1)

A : BOOL (Adresseingang)

Output REAL (IN0 wenn A=0, IN1 wenn A=1)



MUX_R2 wählt einen von 2 Eingangswerten aus. Die Funktion liefert den Wert von IN0 zurück, wenn A=0 und den Wert von IN1, wenn A=1.

19.17. MUX_R4

Type Funktion

Input IN0 : REAL (Eingangswert 0)

IN1 : REAL (Eingangswert 1)

IN2 : REAL (Eingangswert 0)

IN3 : REAL (Eingangswert 1)

A0 : BOOL (Adresseingang Bit 0)

A1 : BOOL (Adresseingang Bit 1)

Output REAL (IN0 wenn A0=0 und A1=0, IN3 wenn A0=1 und A3=1)

MUX_R4 wählt einen von 4 Eingangswerten aus.

Logische Verknüpfung: IN0 wenn A0=0 & A1=0,
IN1 wenn A0=1 & A1=0;
IN2 wenn A0=0 & A1=1;
IN3 wenn A0=1 & A1=1;

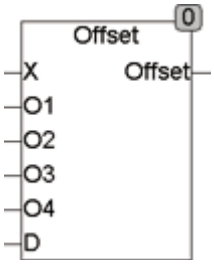
19.18. OFFSET

Type Funktion

Input X : REAL (Eingangssignal)
O1 : BOOL (Enable Offset 1)
O2 : BOOL (Enable Offset 2)
O3 : BOOL (Enable Offset 3)
D : BOOL (Enable Default)

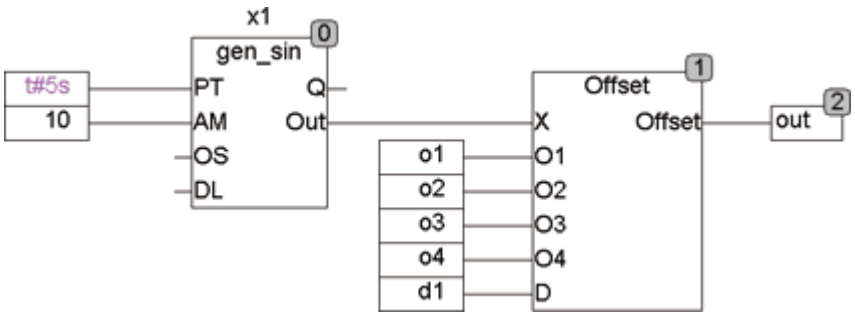
Output REAL (Ausgangswert mit Offset)

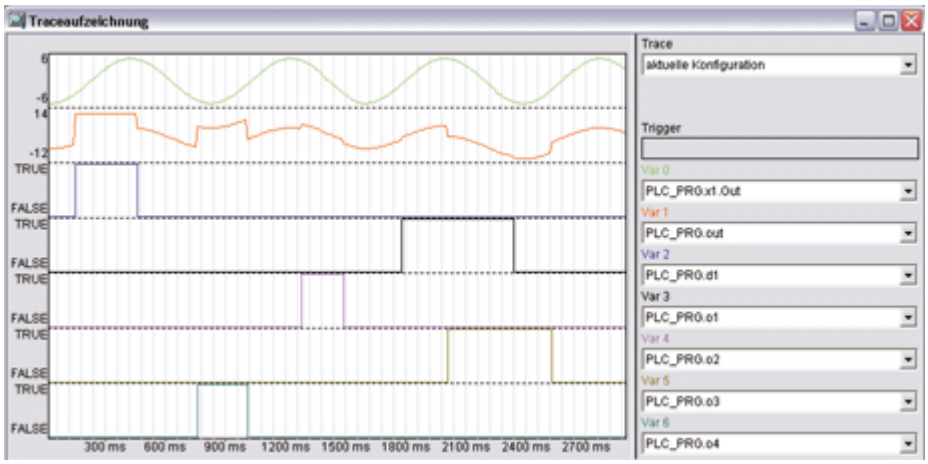
Setup Offset_1 : REAL (Offset der addiert wird, wenn O1=TRUE)
Offset_2 : REAL (Offset der addiert wird, wenn O2=TRUE)
Offset_3 : REAL (Offset der addiert wird, wenn O3=TRUE)
Offset_4 : REAL (Offset der addiert wird, wenn O4=TRUE)
Default : REAL (Wird anstelle von X verwendet, wenn D=TRUE)



Die Funktion Offset addiert verschiedene Offsets zu einem Eingangssignal abhängig vom Binären Wert an O1 .. O4. Die Offsets können einzeln oder gleichzeitig addiert werden. Mit dem Eingang D kann ein Default-Wert anstelle des Eingangs X auf den Addierer geschaltet werden. Die Offsets und der Default-Wert werden über Setup-Variablen definiert.

Das folgende Beispiel verdeutlicht die Funktionsweise von Offset:

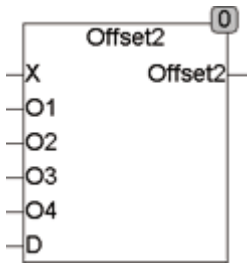




19.19. OFFSET2

Type	Funktion
Input	X : REAL (Eingangssignal)
	O1 : REAL (Enable Offset 1)

	O2 : REAL (Enable Offset 2)
	O3 : REAL (Enable Offset 3)
	D : BOOL (Enable Default)
Output	REAL (Ausgangswert mit Offset)
Setup	Offset_1 : REAL (Offset der addiert wird, wenn O1=TRUE)
	Offset_2 : REAL (Offset der addiert wird, wenn O2=TRUE)
	Offset_3 : REAL (Offset der addiert wird, wenn O3=TRUE)
	Offset_4 : REAL (Offset der addiert wird, wenn O4=TRUE)
	DEFAULT : REAL (Wird anstelle von X verwendet, wenn TRUE)



Die Funktion OFFSET2 addiert einen Offset zu einem Eingangssignal abhängig vom Binären Wert an O1 .. O4. Werden mehrere Offsets gleichzeitig selektiert, so wird der Offset mit der höchsten Nummer addiert und die anderen ignoriert. Werden O1 und O3 gleichzeitig TRUE, so wird nur Offset_3 addiert und nicht Offset_1. Mit dem Eingang D kann ein Default-Wert anstelle des Eingangs X auf den Addierer geschaltet werden. Die Offsets und der Defaultwert werden über Setup-Variablen definiert.

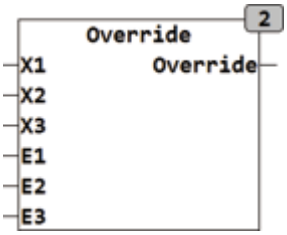
Weitere Erläuterungen und ein Beispiel finden Sie unter Offset, welches sehr ähnliche Funktion aufweist. Offset2 addiert nur jeweils einen (den mit der höchsten Nummer) Offset, während Offset alle selektierten gleichzeitig addiert.

19.20. OVERRIDE

Type	Funktion
Input	X1 : REAL (Eingangssignal 1)
	X2 : REAL (Eingangssignal 2)
	X3 : REAL (Eingangssignal 3)
	E1 : BOOL (Enable Signal 1)
	E2 : BOOL (Enable Signal 2)

E2 : BOOL (Enable Signal 3)

Output REAL (Ausgangswert



OVERRIDE liefert am Ausgang Y den Eingangswert (X1, X2, X3) dessen Absoluter Wert der größere von allen ist. Die Eingänge X1, X2 und X3 können jeder individuell mit den Eingängen E1, E2 und E3 freigeschaltet werden. wenn eines der Eingangssignale E1, E2 oder E3 auf FALSE steht wird der zugehörige Eingang X1, X2 oder X3 nicht berücksichtigt. Eine von vielen Anwendungsmöglichkeiten von OVERRIDE ist zum Beispiel die Abfrage von 3 Sensoren wobei der mit dem Höchsten Wert die anderen überschreibt. Mit den Eingängen E kann im Diagnosefall jeder Sensor einzeln abgefragt werden, oder ein defekter Sensor abgeschaltet werden.

Beispiel:

OVERRIDE(10,-12,11, TRUE, TRUE, TRUE) = -12

OVERRIDE(10,-12,11, TRUE, FALSE, TRUE) = 11

OVERRIDE(10,-12,11, FALSE, FALSE, FALSE) = 0

19.21. RANGE_TO_BYTE

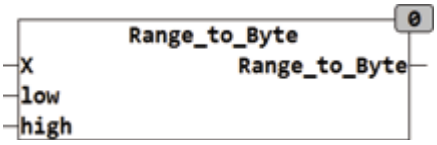
Type Funktion

Input X : REAL (Eingangswert)

LOW : REAL (untere Bereichsgrenze)

HIGH : REAL (Obere Bereichsgrenze)

Output BYTE (Ausgangswert)

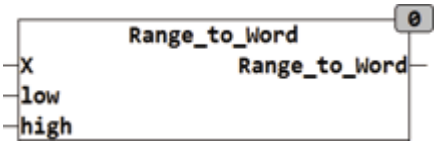


RANGE_TO_BYTE wandelt einen REAL Wert in einen BYTE Wert. Ein Eingangswert X der dem Wert LOW entspricht wird dabei in einen Ausgangswert von 0 gewandelt und ein Eingangswert X der dem Eingangswert HIGH entspricht wird in einen Ausgangswert von 255 gewandelt. Der Eingang X

wird auf den Bereich von LOW bis HIGH begrenzt, ein Überlauf des BYTE Ausgangs kann deshalb nicht stattfinden.

19.22. RANGE_TO_WORD

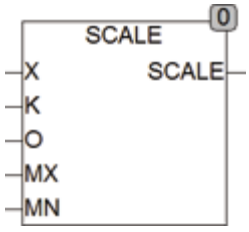
Type	Funktion
Input	X : REAL (Eingangswert)
	LOW : REAL (untere Bereichsgrenze)
	HIGH : REAL (Obere Bereichsgrenze)
Output	WORD (Ausgangswert)



RANGE_TO_WORD wandelt einen REAL Wert in einen WORD Wert. Ein Eingangswert X der dem Wert LOW entspricht wird dabei in einen Ausgangswert von 0 gewandelt und ein Eingangswert X der dem Eingangswert HIGH entspricht wird in einen Ausgangswert von 65535 gewandelt. Der Eingang X wird auf den Bereich von LOW bis HIGH begrenzt, ein Überlauf des WORD Ausgangs kann deshalb nicht stattfinden.

19.23. SCALE

Type	Funktion : REAL
Input	X : Byte (Eingangswert)
	K : Byte (Multiplikator)
	O : REAL (Offset)
	MX : REAL (maximaler Ausgangswert)
	MN : REAL (minimaler Ausgangswert)
Output	REAL (Ausgangswert)

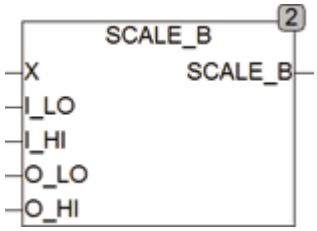


SCALE Multipliziert den Eingang X mit K und addiert den Offset O. Der so errechnete Wert wird dann auf die Werte MN und MX begrenzt und das Ergebnis am Ausgang zur Verfügung gestellt.

SCALE = LIMIT(MN, X * K + O, MX)

19.24. SCALE_B

Type	Funktion : REAL
Input	X : DWORD (Eingangswert)
	I_LO : DWORD (Eingangswert min)
	I_HI : DWORD (Eingangswert max)
	O_LO : REAL (Ausgangswert min)
	O_HI : REAL (Ausgangswert max)
Output	REAL (Ausgangswert)



SCALE_B skaliert einen Eingangswert BYTE und errechnet einen Ausgangswert in REAL. Der Eingangswert X wird dabei auf I_LO und I_HI begrenzt. SCALE_D(IN, 0, 255, 0, 100) skaliert einen Eingang mit 8 Bit Auflösung auf den Ausgang 0..100.

19.25. SCALE_B2

Type	Funktion : REAL
Input	IN1 : Byte (Eingangswert 1)
	IN2 : Byte (Eingangswert 2)

K : REAL (Multiplikator)

O : REAL (Offset)

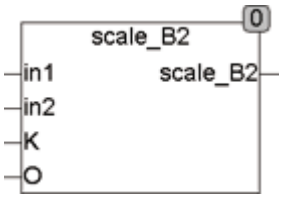
Output REAL (Ausgangswert)

Setup IN1_MIN : REAL (kleinster Wert für IN1)

IN1_MAX : REAL (größter Wert für IN1)

IN2_MIN : REAL (kleinster Wert für IN2)

IN2_MAX : REAL (größter Wert für IN2)

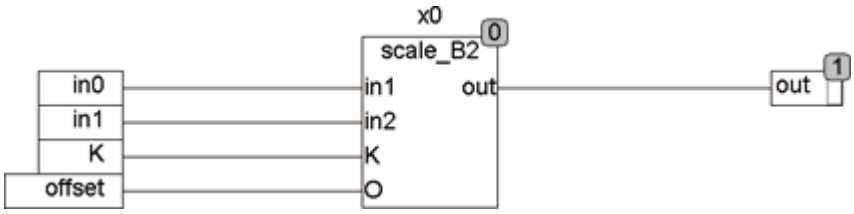


SCALE_B2 berechnet aus dem Eingangswert IN und den Setup-Werten IN_MIN und IN_MAX einen internen Wert, addiert alle internen Werte, multipliziert die Summe mit K und addiert den Offset O. Ein Eingangswert IN1=0 bedeutet IN1_MIN wird berücksichtigt, IN1=255 bedeutet IN1_MAX wird berücksichtigt. Wird K nicht beschaltet so ist der Multiplikator 1.

Out = (in1 * (IN1_MAX – IN1_MIN) / 255 + IN1_MIN + in2 * (IN2_MAX – IN2_MIN) / 255 + IN2_MIN) * K + O

SCALE_B2 kann zum Beispiel verwendet werden um Gesamtluftmengen in Lüftungsanlagen zu berechnen. Auch überall dort wo gesteuerte Mischer eingesetzt werden und die resultierende Gesamtmenge berechnet werden muss.

Beispiel:



IN0 ist eine Luftklappe, die die Luftmenge zwischen 100m³/h und 600m³/h für die Stellwerte in0 (0 – 255) regelt.

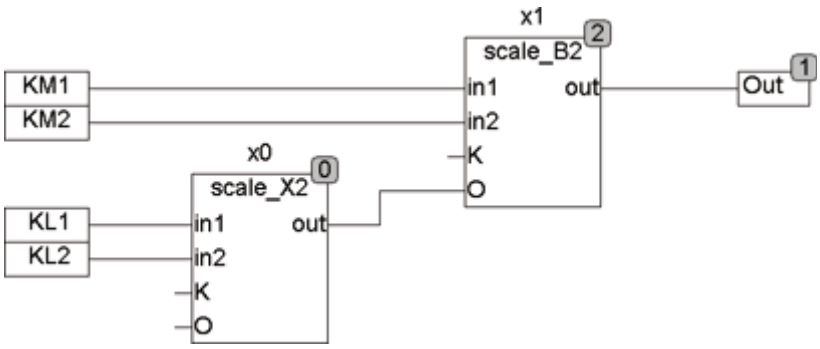
IN1 ist eine Abluftgerät, das die Abluftmenge von 0 m³/h bis 400 m³/h für die Stellwerte in1 von 0 – 255 regelt.

Die Setup-Werte für diese Anwendung sind: IN0_MIN = 100, IN0_MAX = 600, IN1_MIN = 0, IN1_MAX = -400.

Die resultierende Gesamtluftmenge bei K=1 und O=0 (kein Multiplikator und kein Offset) variiert dann von -300 (in0=0 und in1=255) bis +600 (in=255 und in1=0).

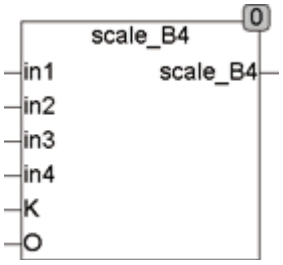
Für einen Eingangswert in0=128 (Klappe 50%) und IN1=128 (Lüfter auf 50%) ist der Ausgangswert 250 m³ – 200 m³ = 50 m³.

Der Eingang Offset kann auch dazu benutzt werden um Bausteine zu kaskadieren.



19.26. SCALE_B4

Type	Funktion : REAL
Input	IN1.. IN4 : Byte (Eingangswerte)
	K : REAL (Multiplikator)
	O : REAL (Offset)
Output	REAL (Ausgangswert)
Setup	IN1_MIN : REAL (kleinster Wert für IN1)
	IN1_MAX : REAL (größter Wert für IN1)
	IN2_MIN : REAL (kleinster Wert für IN2)
	IN2_MAX : REAL (größter Wert für IN2)
	IN3_MIN : REAL (kleinster Wert für IN3)
	IN3_MAX : REAL (größter Wert für IN3)
	IN4_MIN : REAL (kleinster Wert für IN4)
	IN4_MAX : REAL (größter Wert für IN4)



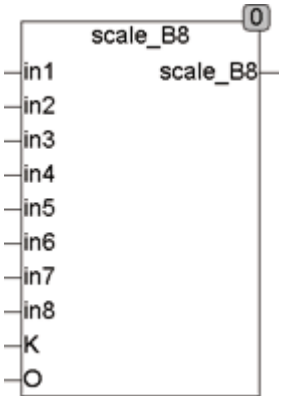
SCALE_B4 berechnet aus den Eingangswerten IN und den Setup-Werten IN_MIN und IN_MAX interne Werte, addiert alle internen Werte, multipliziert die Summe mit K und addiert den Offset O. Ein Eingangswert IN=0 bedeutet IN_MIN wird berücksichtigt, IN=255 bedeutet IN_MAX wird berücksichtigt. Wird K nicht beschaltet, so ist der Multiplikator 1.

OUT = (in1 * (IN1_MAX – IN1_MIN) / 255 + IN1_MIN
+ in2 * (IN2_MAX – IN2_MIN) / 255 + IN2_MIN
+ in3 * (IN3_MAX – IN3_MIN) / 255 + IN3_MIN
+ in4 * (IN4_MAX – IN4_MIN) / 255 + IN4_MIN) * K + O

SCALE_B4 kann zum Beispiel verwendet werden um Gesamtluftmengen in Lüftungsanlagen zu berechnen, oder überall dort, wo gesteuerte Mischer eingesetzt werden und die resultierende Gesamtmenge berechnet werden muss. Weitergehende Erläuterungen zur Funktionsweise finden Sie auch unter SCALE_B2.

19.27. SCALE_B8

Type	Funktion : REAL
Input	IN1 .. 8 : Byte (Eingangswerte)
	K : REAL (Multiplikator)
	O : REAL (Offset)
Output	REAL (Ausgangswert)
Setup	IN_MIN : REAL (kleinster Wert für IN)
	IN_MAX : REAL (größter Wert für IN)



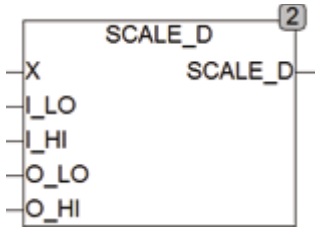
SCALE_B8 berechnet aus den Eingangswerten IN und den Setup-Werten IN_MIN und IN_MAX interne Werte, addiert alle internen Werte, multipliziert die Summe mit K und addiert den Offset O. Ein Eingangswert IN=0 bedeutet IN_MIN wird berücksichtigt, IN=255 bedeutet IN_MAX wird berücksichtigt. Wird K nicht beschaltet so ist der Multiplikator 1.

OUT = (in1 * (IN1_MAX – IN1_MIN) / 255 + IN1_MIN
+ in2 * (IN2_MAX – IN2_MIN) / 255 + IN2_MIN
+ in3 * (IN3_MAX – IN3_MIN) / 255 + IN3_MIN
+ in4 * (IN4_MAX – IN4_MIN) / 255 + IN4_MIN
+ in5 * (IN5_MAX – IN5_MIN) / 255 + IN5_MIN
+ in6 * (IN6_MAX – IN6_MIN) / 255 + IN6_MIN
+ in7 * (IN7_MAX – IN7_MIN) / 255 + IN7_MIN
+ in8 * (IN8_MAX – IN8_MIN) / 255 + IN8_MIN) * K + O

SCALE_B8 kann zum Beispiel verwendet werden um Gesamtluftmengen in Lüftungsanlagen zu berechnen, oder überall dort, wo gesteuerte Mischer eingesetzt werden und die resultierende Gesamtmenge berechnet werden muss. Weitergehende Erläuterungen zur Funktionsweise finden Sie auch unter SCALE_B2.

19.28. SCALE_D

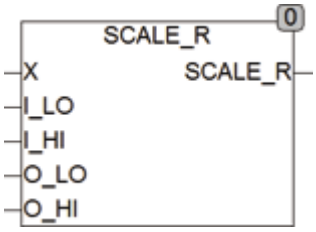
Type	Funktion : REAL
Input	X : DWORD (Eingangswert)
	I_LO : DWORD (Eingangswert min)
	I_HI : DWORD (Eingangswert max)
	O_LO : REAL (Ausgangswert min)
	O_HI : REAL (Ausgangswert max)
Output	REAL (Ausgangswert)



SCALE_D skaliert einen Eingangswert DWORD und errechnet einen Ausgangswert in REAL. Der Eingangswert X wird dabei auf I_LO und I_HI begrenzt. SCALE_D(IN, 0, 8191, 0, 100) skaliert einen Eingang mit 14 Bit Auflösung auf den Ausgang 0..100. SCALE_D kann auch mit negativen Ausgangswerten und negativer Steigung arbeiten, die Werte I_LO und I_HI müssen aber immer so spezifiziert werden, dass ILO < I_HI ist.

19.29. SCALE_R

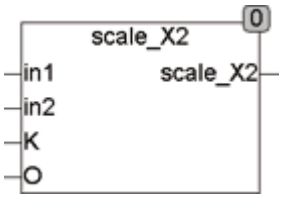
Type	Funktion : REAL
Input	X : REAL (Eingangswert)
	I_LO : REAL (Eingangswert min)
	I_HI : REAL (Eingangswert max)
	O_LO : REAL (Ausgangswert min)
	O_HI : REAL (Ausgangswert max)
Output	REAL (Ausgangswert)



SCALE_R skaliert einen Eingangswert REAL und errechnet einen Ausgangswert in REAL. Der Eingangswert X wird dabei auf I_LO und I_HI begrenzt. SCALE_D(IN, 4, 20, 0, 100) skaliert einen Eingang mit 4 .. 20 mA auf den Ausgang 0..100. SCALE_R kann auch mit negativen Ausgangswerten und negativer Steigung arbeiten, die Werte I_LO und I_HI müssen aber immer so spezifiziert werden, dass ILO < I_HI ist.

19.30. SCALE_X2

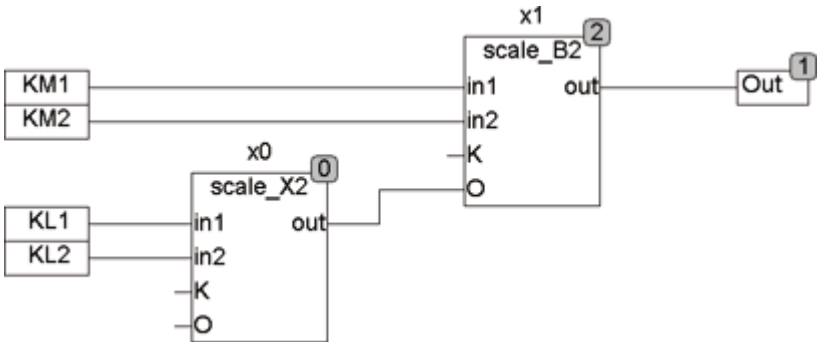
Type	Funktion
Input	IN1 .. 2 : BOOL (Eingangswerte)
	K : REAL (Multiplikator)
	O : REAL (Offset)
Output	REAL (Ausgangswert)
Setup	IN_MIN : REAL (kleinster Wert für IN)
	IN_MAX : REAL (größter Wert für IN)



SCALE_X2 berechnet aus den Eingangswerten IN und den Setup-Werten IN_MIN und IN_MAX interne Werte, addiert alle internen Werte, multipliziert die Summe mit K und addiert den Offset O. Ein Eingangswert IN=FALSE bedeutet IN_MIN wird berücksichtigt, in=TRUE bedeutet IN_MAX wird berücksichtigt. Die Summe wird Mit K Multipliziert und Offset O addiert. Wird K nicht beschaltet so ist der Multiplikator 1.

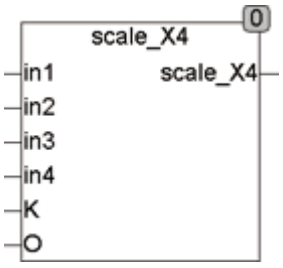
SCALE_X2 kann zum Beispiel verwendet werden um Gesamtluftmengen in Lüftungsanlagen zu berechnen, oder überall dort wo gesteuerte Klappen eingesetzt werden und die resultierende Gesamtmenge berechnet werden muss. Durch den Eingang Offset kann SCALE_X2 einfach mit den anderen SCALE Bausteinen kaskadiert werden.

Im folgenden Beispiel werden zwei motorische Klappen KM1 und km² mit 2 Ein/Aus Klappen KL1 und KL2 verknüpft und die resultierende Luftmenge berechnet.



19.31. SCALE_X4

Type	Funktion : REAL
Input	IN1 .. 4 : BOOL (Eingangswerte)
	K : REAL (Multiplikator)
	O : REAL (Offset)
Output	REAL (Ausgangswert)
Setup	IN_MIN : REAL (kleinster Wert für IN)
	IN_MAX : REAL (größter Wert für IN)



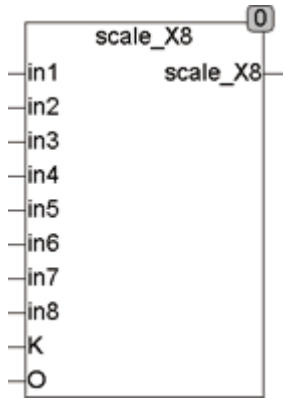
SCALE_X4 berechnet aus den Eingangswerten IN und den Setup-Werten IN_MIN und IN_MAX interne Werte, addiert alle internen Werte, multipliziert die Summe mit K und addiert den Offset O. Ein Eingangswert IN=FALSE bedeutet IN_MIN wird berücksichtigt, in=TRUE bedeutet IN_MAX wird berücksichtigt. Die Summe wird Mit K Multipliziert und Offset O addiert. Wird K nicht beschaltet so ist der Multiplikator 1.

SCALE_X4 kann zum Beispiel verwendet werden um Gesamtluftmengen in Lüftungsanlagen zu berechnen, oder überall dort wo gesteuerte Klappen eingesetzt werden und die resultierende Gesamtmenge berechnet werden muss. Durch den Eingang Offset kann SCALE_X2 einfach mit den anderen SCALE Bausteinen kaskadiert werden. Weitere Erläuterungen und Beispiele finden Sie unter SCAE_X2.

19.32. SCALE_X8

Type	Funktion : REAL
Input	IN1 .. 8 : BOOL (Eingangswerte)
	K : REAL (Multiplikator)
	O : REAL (Offset)
Output	REAL (Ausgangswert)
Setup	IN_MIN : REAL (kleinster Wert für IN)

IN_MAX : REAL (größter Wert für IN)

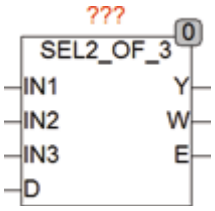


SCALE_X8 berechnet aus den Eingangswerten IN und den Setup-Werten IN_MIN und IN_MAX interne Werte, addiert alle internen Werte, multipliziert die Summe mit K und addiert den Offset O. Ein Eingangswert IN=FALSE bedeutet IN_MIN wird berücksichtigt, in=TRUE bedeutet IN_MAX wird berücksichtigt. Die Summe wird Mit K Multipliziert und Offset O addiert. Wird K nicht beschaltet so ist der Multiplikator 1.

SCALE_X8 kann zum Beispiel verwendet werden um Gesamtluftmengen in Lüftungsanlagen zu berechnen, oder überall dort wo gesteuerte Klappen eingesetzt werden und die resultierende Gesamtmenge berechnet werden muss. Durch den Eingang Offset kann SCALE_X2 einfach mit den anderen SCALE Bausteinen kaskadiert werden. Weitere Erläuterungen und Beispiele finden Sie unter SCAE_X2.

19.33. SEL2_OF_3

Type	Funktion : REAL
Input	IN1 : REAL (Eingangswert 1)
	IN2 : REAL (Eingangswert 2)
	IN3 : REAL (Eingangswert 3)
	D : REAL (Toleranzgrenze)
Output	Y : REAL (Ausgangswert)
	W : INT (Warnung)
	E : BOOL (Error Ausgang)

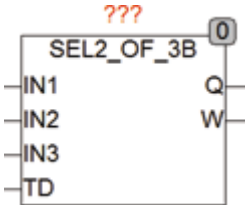


SEL2_OF_3 wertet 3 Eingänge (IN1 .. IN3) aus und prüft ob die Abweichung der Eingangswert kleiner oder gleich D ist. Der Mittelwert aus den 3 Eingängen wird am Ausgang Y ausgegeben. Die einzelnen Eingänge werden nur dann berücksichtigt wenn Sie nicht weiter als D von einem anderen Eingang entfernt liegen. Wird der Mittelwert nur von 2 Eingängen gebildet, so wird die Nummer des nicht berücksichtigten Eingangs am Ausgang W ausgegeben. Ist W = 0 werden alle 3 Eingänge berücksichtigt. Falls alle 3 Eingänge mehr als D voneinander abweichen wird der Ausgang W = 4 gesetzt und der Ausgang E = TRUE gesetzt. Der Ausgang Y wird in diesem Falle nicht verändert und bleibt auf dem letzten gültigen Wert stehen.

Eine typische Anwendung für den Baustein ist die Erfassung von 3 Sensoren die dieselbe Prozessgröße Messen um zum Beispiel Messfehler durch unterschiedliche Erfassung oder Drahtbruch zu verringern.

19.34. SEL2_OF_3B

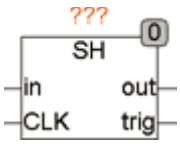
Type	Funktion : BOOL
Input	IN1 : BOOL (Eingang 1)
	IN2 : BOOL (Eingang 2)
	IN3 : BOOL (Eingang 3)
	TD : TIME (Delay für Ausgang W)
Output	Q : BOOL (Ausgang)
	W : BOOL (Warnmeldung)



SEL2_OF_3B wertet 3 redundante Binäre Eingänge aus und liefert am Ausgang Q den Wert den mindestens 2 der 3 Eingänge anliegen haben. Falls einer der 3 Eingänge einen anderen Wert als die beiden anderen hat wird der Ausgang W als Warnung gesetzt. Damit bei nicht exakt Zeitgleichen Umschalten der Eingänge der Ausgang W nicht anspricht kann für diesen eine Ansprechverzögerung TD festgelegt werden.

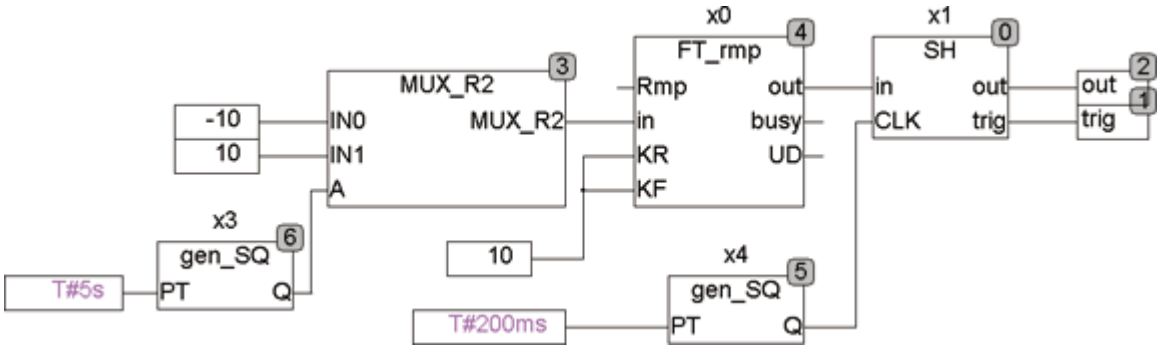
19.35. SH

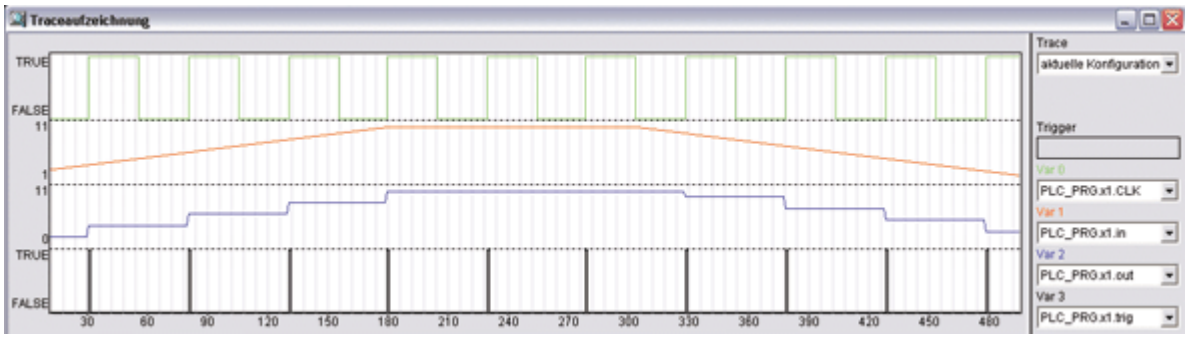
Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	CLK : BOOL (Takteingang)
Output	OUT : REAL (Ausgangssignal)
	TRIG : BOOL (Trigger Output)



SH ist ein Sample and Hold Baustein. Er speichert bei jeder steigenden Flanke von CLK das Eingangssignal IN am Ausgang OUT. Nach jedem update von OUT bleibt TRIG für einen Zyklus TRUE.

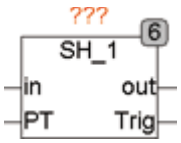
Das Folgende Beispiel erläutert die Funktionsweise von SH:





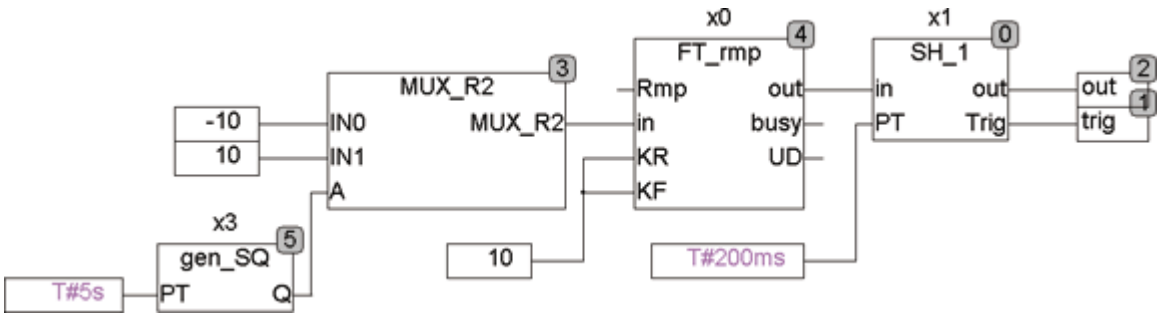
19.36. SH_1

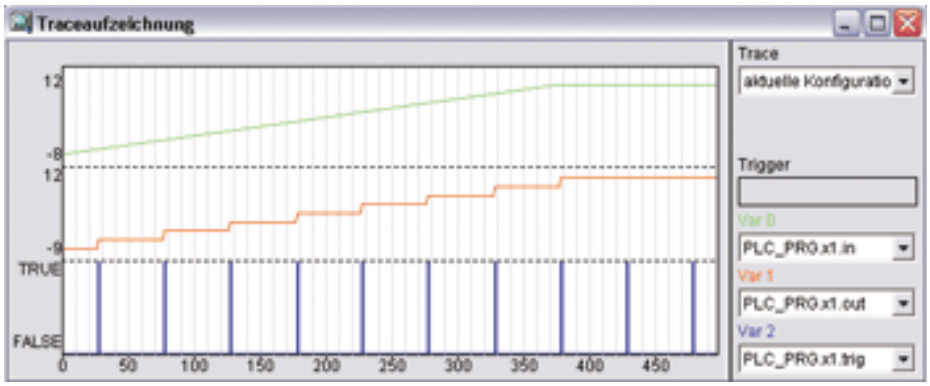
Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	PT : TIME (Abtastzeit)
Output	OUT : REAL (Ausgangssignal)
	TRIG: BOOL (Trigger Output)



SH_1 ist ein Sample und Hold Baustein mit einstellbarer Abtastzeit. Er speichert alle PT das Eingangssignal IN am Ausgang OUT. Nach jedem update von OUT bleibt TRIG für einen Zyklus TRUE.

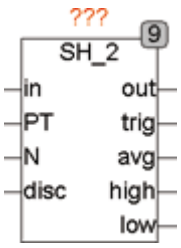
Das Folgende Beispiel erläutert die Funktionsweise von SH_1:





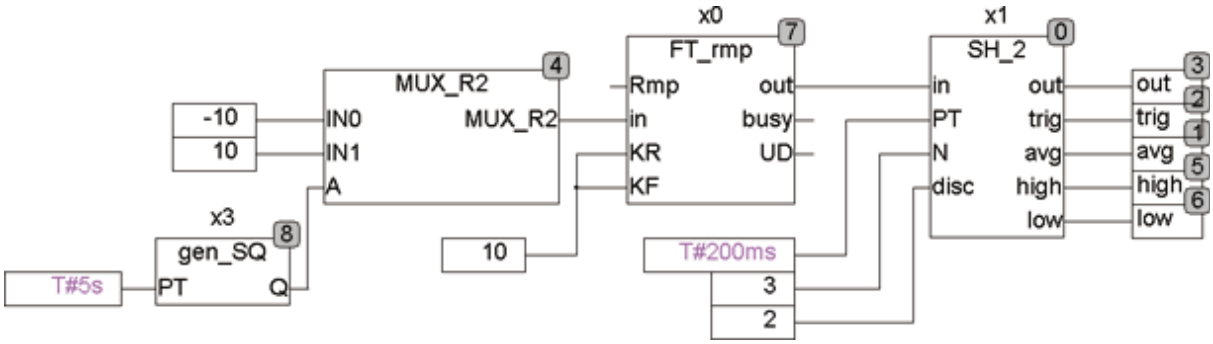
19.37. SH_2

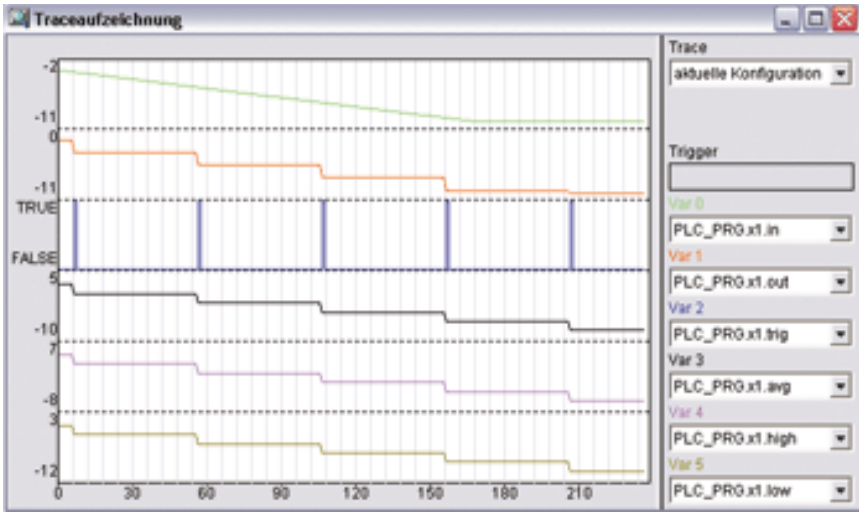
Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	PT : TIME (Abtastzeit)
	N : INT (Anzahl der Samples für Statistik)
	DISC : INT (Verwerfe DISC Werte)
Output	OUT : REAL (Ausgangssignal)
	TRIG : BOOL (Trigger Output)
	AVG : REAL (Durchschnittswert)
	HIGH : REAL (Maximalwert)
	LOW : REAL (Minimalwert)



SH_2 ist ein Sample and Hold Baustein mit einstellbarer Abtastzeit. Er speichert alle PT das Eingangssignal IN am Ausgang OUT. Nach jedem update von OUT bleibt TRIG für einen Zyklus TRUE. Zusätzlich zur Funktion eines Sample and Hold Bausteins bietet SH_2 bereits integrierte Funktionalität bezüglich der Statistik. Mit dem Eingang N kann spezifiziert werden, über wie viele Samples (maximal 16) ein Mittelwert, Minimalwert und Maximalwert gebildet werden. Als weitere Eigenschaft können aus den N Samples für die Statistik auch kleinste und größte Werte ignoriert werden, was sehr sinnvoll sein kann um Extremwerte zu ignorieren. Der Eingangswert DISC=0 bedeutet alle Samples benutzen, eine 1 bedeutet den niedrigsten Wert ignorieren, 2 bedeutet den niedrigsten und den höchsten Wert ignorieren usw. Wenn zum Beispiel N=5 und DISC=2, dann werden 5 Samples gesammelt, der niedrigste und der höchste Wert werden verworfen und über die 3 verbleibenden Samples wird der Mittelwert, Minimalwert und Maximalwert gebildet.

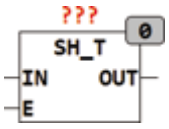
Das Folgende Beispiel erläutert die Funktionsweise von SH_2:





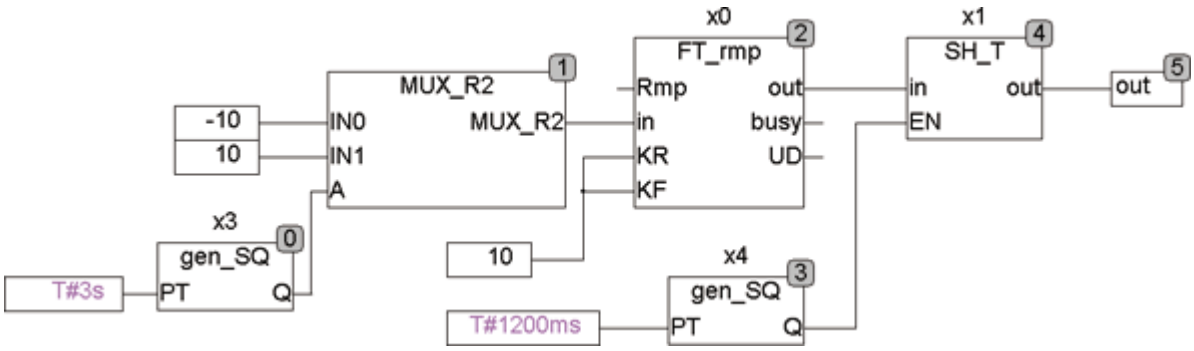
19.38. SH_T

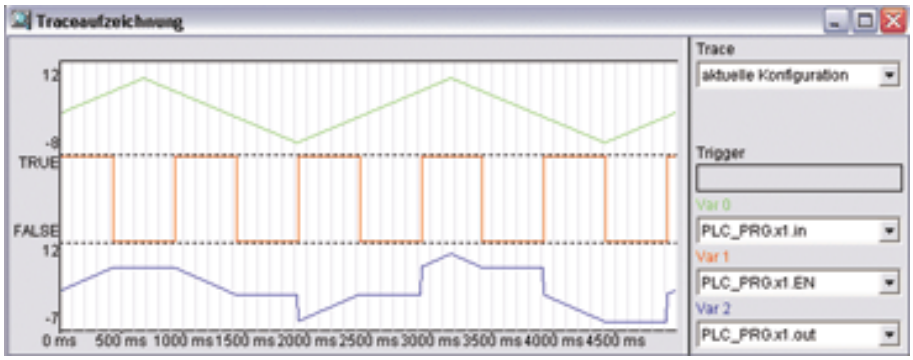
Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	E : BOOL (enable Signal)
Output	OUT : REAL (Ausgangssignal)



SH_T ist ein transparenter Sample and Hold Baustein. Das Eingangssignal in ist solange am Ausgang verfügbar, wie E gleich TRUE ist. Mit einer fallenden Flanke von E wird der Wert von in am Ausgang OUT gespeichert und bleibt solange bestehen bis E wieder TRUE wird und dadurch wieder in auf OUT geschaltet wird.

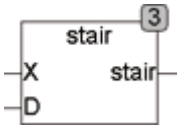
Das folgende Beispiel verdeutlicht die Arbeitsweise von SH_T:





19.39. STAIR

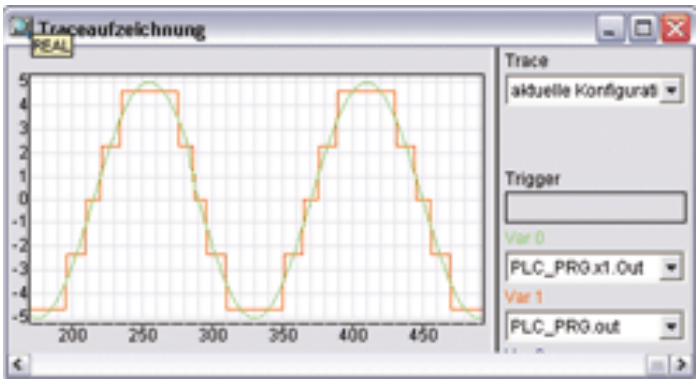
Type	Funktion
Input	X : REAL (Eingangssignal)
	D : REAL (Schrittweite des Ausgangssignals)
Output	REAL (Ausgangssignal)



Das Ausgangssignal von STAIR folgt dem Eingangssignal X mit einer Treppenfunktion. Die Höhe der Stufen ist vorgegeben durch D. Wird X = 0, so folgt das Ausgangssignal direkt dem Eingangssignal. STAIR ist jedoch nicht zum filtern von Eingangssignalen geeignet, denn wenn der Eingang um eine Treppenschwelle schwankt, schaltet der Ausgang zwischen zwei benachbarten Werten hin und her. Für diesen Zweck empfehlen wir den

Einsatz von Stair2, der mit einer Hysterese arbeitet und instabile Zustände vermeidet.

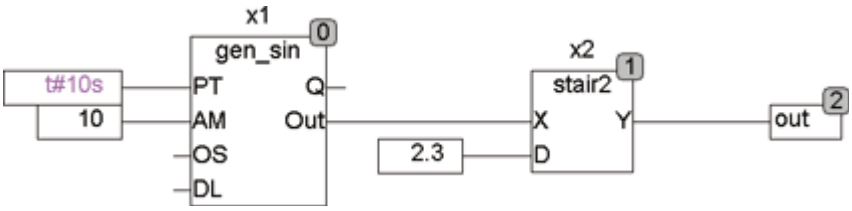
Das folgende Beispiel verdeutlicht die Arbeitsweise von STAIR:



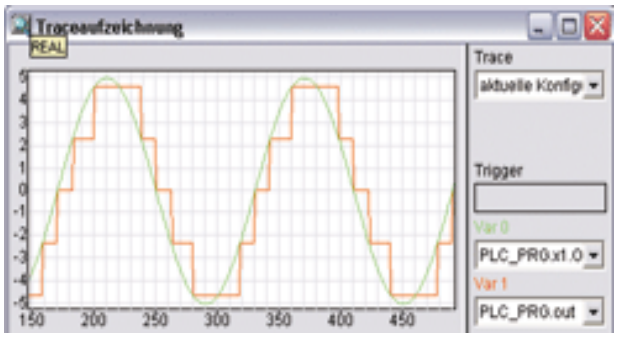
19.40. STAIR2

Type	Funktionsbaustein
Input	X : REAL (Eingangssignal)
	D : REAL (Schrittweite des Ausgangssignals)
Output	Y : REAL (Ausgangssignal)

Das Ausgangssignal von STAIR2 folgt dem Eingangssignal X mit einer Treppenfunktion. Die Höhe der Stufen ist vorgegeben durch D. Wird D = 0, so folgt das Ausgangssignal direkt dem Eingangssignal. Das Signal folgt der Treppe aber mit einer Hysterese von D so dass ein verrauschtes Eingangssignal keine Sprünge zwischen Treppenwerten auslösen kann. STAIR2 ist auch als Eingangsfilter geeignet.

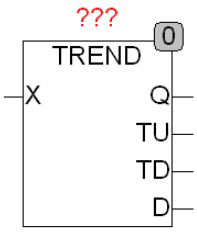


Das folgende Beispiel verdeutlicht die Arbeitsweise von STAIR2:



19.41. TREND

Type	Funktionsbaustein
Input	X : REAL (Eingangssignal)
Output	Q : BOOL (X steigend = TRUE)
	TU : BOOL (TRUE wenn sich Eingang X erhöht)
	TD : BOOL (TRUE wenn sich Eingang X reduziert)
	D : REAL (Deltas der Eingangsänderung)

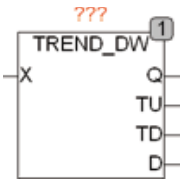


TREND überwacht den Eingang X und zeit am Ausgang Q an ob X steigt (Q = TRUE) oder X fällt (Q = FALSE). wenn sich X nicht verändert bleibt Q auf seinen letzten Wert stehen. Erhöht sich X so wird der Ausgang TU für einen Zyklus TRUE und am Ausgang D wird X – LAST_X angezeigt. Ist X niedriger als LAST_X so wird TD für einen Zyklus TRUE und am Ausgang D

wird LAST_X – X ausgegeben. LAST_X ist ein interner Wert des Bausteins und ist der Wert von X im letzten Zyklus.

19.42. TREND_DW

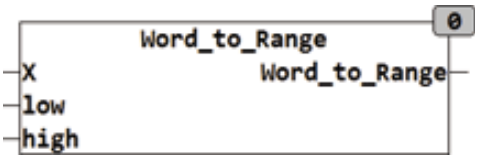
Type	Funktionsbaustein
Input	X : DWORD (Eingangssignal)
Output	Q : BOOL (X steigend = TRUE)
	TU : BOOL (TRUE wenn sich Eingang X erhöht)
	TD : BOOL (TRUE wenn sich Eingang X reduziert)
	D : DWORD (Delta der Eingangsänderung)



TREND_DW überwacht den Eingang X und zeit am Ausgang Q an ob X steigt (Q = TRUE) oder X fällt (Q = FALSE). wenn sich X nicht verändert bleibt Q auf seinen letzten Wert stehen. Erhöht sich X so wird der Ausgang TU für einen Zyklus TRUE und am Ausgang D wird X – LAST_X angezeigt. Ist X niedriger als LAST_X so wird TD für einen Zyklus TRUE und am Ausgang D wird LAST_X – X ausgegeben. LAST_X ist ein interner Wert des Bausteins und ist der Wert von X im letzten Zyklus.

19.43. WORD_TO_RANGE

Type	Funktion
Input	X : WORD (Eingangswert)
	LOW : REAL (Ausgangswert bei X = 0)
	HIGH : REAL (Ausgangswert bei X = 65535)
Output	REAL (Ausgangswert)



WORD_TO_RANGE wandelt einen WORD Wert in einen REAL Wert. Ein Eingangswert von 0 entspricht dabei dem REAL Wert von LOW und eine Eingangswert von 65535 entspricht dem Eingangswert von HIGH.

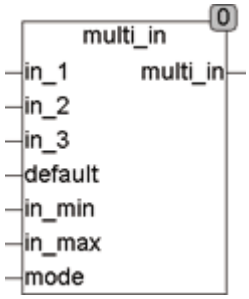
Um einen WORD Wert von 0..65535 in einen Prozentwert von 0..100 zu Wandeln wird der Baustein wie folgt aufgerufen:

WORD_TO_RANGE(X,100,0)

20. Sensorik

20.1. MULTI_IN

Type	Funktion : REAL
Input	IN_1 : REAL (Eingang 1)
	IN_2 : REAL (Eingang 2)
	IN_3 : REAL (Eingang 3)
	DEFAULT : REAL (Vorgabewert)
	IN_MIN : REAL (unterer Grenzwert für Eingänge)
	IN_MAX : REAL (oberer Grenzwert für Eingänge)
	MODE : Byte (Auswahl des Betriebsmodus)
Output	REAL (Ausgangssignal)



MULTI_IN ist ein Sensorinterface, das bis zu 3 Sensoren einliest, auf Fehler überprüft und abhängig vom Eingangsmodus einen Ausgangswert berechnet.

Mode	Funktion
0	MULTI_in = Durchschnitt der Eingänge in_1 .. 3
1	MULTI_in = Eingang in_1
2	MULTI_in = Eingang in_2
3	MULTI_in = Eingang in_3
4	MULTI_in = Default Eingang
5	MULTI_in = kleinster Wert der Eingänge in_1 .. 3
6	MULTI_in = größter Wert der Eingänge in_1 .. 3
7	MULTI_in = mittlerer Wert der Eingänge in_1..3
>7	MULTI_in = 0

Unabhängig vom eingestellten Mode werden Eingangswerte, die größer als IN_MAX oder kleiner als IN_MIN sind ignoriert. Ist keine Berechnung wie durch Mode vorgegeben mehr möglich, wird der Eingang Default als Ausgangswert benutzt. Multi_in wird eingesetzt, wenn verschiedene Sensoren den gleichen Wert messen und hohe Sicherheit und Zuverlässigkeit gefragt ist. Eine mögliche Anwendung ist die Messung der Außentemperatur an verschiedenen Stellen und die Überwachung auf Kabel oder Sensorbruch.

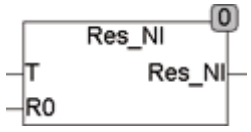
20.2. RES_NI

Type Funktion : REAL

Input T : REAL (Temperatur in °C)

R0 : REAL (Widerstand bei 0 °C)

Output REAL (Widerstandswert)



RES_NI berechnet den Widerstand eines NI-Widerstandsfühlers aus den Eingangswerten T (Temperatur in °C) und R0 (Widerstand bei 0°C).

Die Berechnung erfolgt nach der Formel:

$RES\_NI = R0 + A*T + B*T^2+C*T4$

A = 0.5485

B = 0.665E-3

C = 2.805E-9

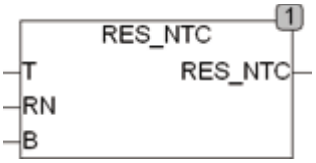
Die Berechnung ist geeignet für einen Temperaturen von -60 .. +180 °C.

Auszug aus DIN 43760 für Ni100

°C	R	°C	R	°C	R	°C	R	°C	R
-60	69,5	-10	94,6	40	123,0	90	154,9	140	190,9
-55	71,9	-5	97,3	45	126,0	95	158,3	145	194,8
-50	74,3	0	100,0	50	129,1	100	161,8	150	198,7
-45	76,7	5	102,8	55	132,2	105	165,3	155	202,6
-40	79,1	10	105,6	60	135,3	110	168,8	160	206,6
-35	81,6	15	108,4	65	138,5	115	172,4	165	210,7
-30	84,2	20	111,2	70	141,7	120	176,0	170	214,8
-25	86,7	25	114,1	75	145,0	125	179,6	175	219,0
-20	89,3	30	117,1	80	148,3	130	183,3	180	223,2
-15	91,9	35	120,0	85	151,6	135	187,1		

20.3. RES_NTC

Type	Funktion : REAL
Input	T : REAL (Temperatur in °C)
	RN : REAL (Widerstand bei 25 °C)
	B : REAL (Charakteristischer wert des Sensors)
Output	REAL (Widerstandswert)



RES_NTC berechnet den Widerstand eines NTC-Widerstandsfühlers aus den Eingangswerten T (Temperatur in °C) und RN (Widerstand bei 25°C). Der Eingangswert B ist eine Konstante die aus den Sensor Datenblättern ent-

nommen werden muss. Typische Werte für NTC Sensoren liegen bei 2000 – 4000 Kelvin.

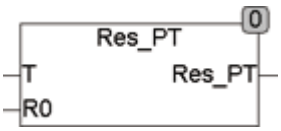
Die Berechnung erfolgt nach der Formel:

$$R_{\mathrm{H}} = R_{\mathrm{N}} \cdot \mathrm{e}^{b\left(\frac{1}{T} - \frac{1}{T_{\mathrm{N}}}\right)}$$

Die Formel liefert eine hinreichende Genauigkeit für kleine Temperaturbereiche wie z.B. 0-100 °C. Für weite Temperaturbereiche ist die Formal nach Steinhart besser geeignet.

20.4. RES_PT

Type	Funktion : REAL
Input	T : REAL (Temperatur in °C)
	R0 : REAL (Widerstand bei 0 °C)
Output	REAL (Widerstandswert)



RES_PT berechnet den Widerstand eines PT-Widerstandsfühlers aus den Eingangswerten T (Temperatur in °C) und R0 (Widerstand bei 0°C).

Die Berechnung erfolgt nach der Formel:

für Temperaturen > 0 °C

RES_PT = R0 * (1 + A*T + B*T²)

und für Temperaturen < 0°C

RES_PT = R0 * (1 + A*T + B*T² + C*(T-100)*T³

A = 3.90802E-3

B = -5.80195E-7

C = -427350E-12

Die Berechnung ist geeignet für Temperaturen von -200 .. +850 °C.

Auszug aus DIN 43760 für Pt100

°C	R	°C	R	°C	R	°C	R	°C	R	°C	R
-200	18,49	-100	60,25	0	100,00	100	138,50	200	175,84	300	212,02
-195	20,65	-95	62,28	5	101,95	105	140,39	205	177,68	305	213,80
-190	22,80	-90	64,30	10	103,90	110	142,29	210	179,51	310	215,57
-185	24,94	-85	66,31	15	105,85	115	144,17	215	181,34	315	217,35
-180	27,08	-80	68,33	20	107,79	120	146,06	220	183,17	320	219,12
-175	29,20	-75	70,33	25	109,73	125	147,94	225	184,99	325	220,88
-170	31,32	-70	72,33	30	111,67	130	149,82	230	186,82	330	222,65
-165	33,43	-65	74,33	35	113,61	135	151,70	235	188,63	335	224,41
-160	35,33	-60	76,33	40	115,54	140	153,58	240	190,45	340	226,17
-155	37,63	-55	78,32	45	117,47	145	155,45	245	192,26	345	227,92
-150	39,71	-50	80,31	50	119,40	150	157,31	250	194,07	350	229,67
-145	41,79	-45	82,29	55	121,32	155	159,18	255	195,88	355	231,42
-140	43,87	-40	84,27	60	123,24	160	161,04	260	197,69	360	233,17
-135	45,94	-35	86,25	65	125,16	165	162,90	265	199,49	365	234,91
-130	48,00	-30	88,22	70	127,07	170	164,76	270	201,29	370	236,65
-125	50,06	-25	90,19	75	128,98	175	166,61	275	203,08	375	238,39
-120	52,11	-20	92,16	80	130,89	180	168,46	280	204,88	380	240,13
-115	54,15	-15	94,12	85	132,80	185	170,31	285	206,67	385	241,86
-110	56,19	-10	96,09	90	134,70	190	172,16	290	208,45	390	243,59
-105	58,22	-5	98,04	95	136,60	195	174,00	295	210,24	395	245,31

20.5. RES_SI

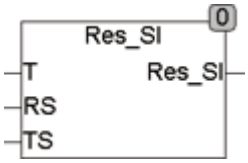
Type Funktion : REAL

Input T : REAL (Temperatur in °C)

RS : REAL (Widerstand bei TS °C)

TS : REAL (Temperatur bei RS)

Output REAL (Widerstandswert)



RES_SI berechnet den Widerstand eines SI-Widerstandsfühlers aus den Eingangswerten T (Temperatur in °C) und RS, Widerstand bei TS in °C. Im Gegensatz zu den Bausteinen RES_NI und RES_PT deren R0 bei 0°C angegeben wird ist die Widerstandsangabe RS bei SI Fühler bei unterschiedlichen Temperaturen (z.B. 25°C bei KTY10). Deshalb hat der Baustein einen Eingang für RS und einen weiteren für TS.

Die Berechnung erfolgt nach der Formel:

$$RES\_SI = RS + A*(T-TS) + B*(T-TS)^2$$

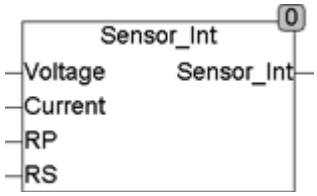
$$A = 7.64E-3$$

$$B = 1.66E-5$$

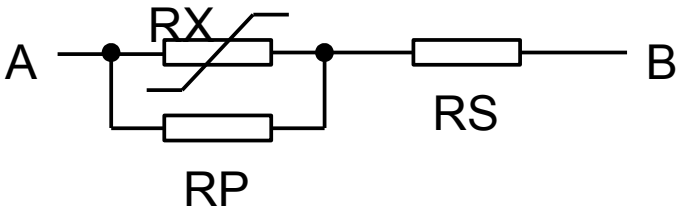
Die Berechnung ist geeignet für einen Temperaturbereich von -50..+150°C.

20.6. SENSOR_INT

Type	Funktion : REAL
Input	VOLTAGE : REAL (gemessen Spannung in Volt)
	CURRENT : REAL (gemessener Strom in Ampere)
	RP : REAL (paralleler parasitärer Widerstand in Ohm)
	RS : REAL (serieller parasitärer Widerstand in Ohm)
Output	REAL (Widerstandswert des Sensors)



SENSOR_INT berechnet den Sensorwiderstand unter Berücksichtigung der Parasitären Widerstände, die die Messung normalerweise verfälschen. Der A/D Wandler misst entweder Strom bei fester Spannung oder Spannung bei festem Strom. Der daraus resultierende Widerstand ist aber nicht nur der Widerstand des Sensors, sondern setzt sich zusammen aus dem Widerstand des Sensors und 2 parasitären Widerständen RP und RS. Da die parasitären Widerstände konstant bleiben, können Sie wieder kompensiert werden und der wirkliche Widerstand des Sensors errechnet werden.

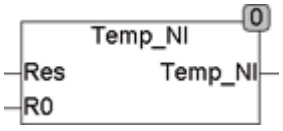


Der zwischen den Anschlüssen A und B gemessene Widerstand (gemessen durch Strom und Spannung ist ein Gesamtwiderstand aus dem Sensorwiderstand parallel mit dem parasitären Widerstand RP und dem Leitungswiderstand RS. RS und RP werden kompensiert und der wirkliche Widerstand RX berechnet. Mit den Modulen TEMP_ kann dann zum Beispiel die exakte Temperatur berechnet werden.

20.7. TEMP_NI

Type	Funktion : REAL
Input	RES : REAL (Widerstandswert in Ohm)
	R0 : REAL (Widerstand bei 0 °C)

Output REAL (gemessene Temperatur)



RES_NI berechnet die Temperatur eines NI-Widerstandsfühlers aus den Eingangswerten RES (gemessener Widerstandswert) und R0 (Widerstand bei 0°C).

Die Berechnung ist geeignet für einen Temperaturbereich von -60 .. +180 °C und erfolgt nach folgender Formal:

$RES\_NI = R0 + A*T + B*T^2+C*T4$

A = 0.5485; B = 0.665E-3; C = 2.805E-9

Auszug aus DIN 43760 für Ni100

°C	R	°C	R	°C	R	°C	R	°C	R
-60	69,5	-10	94,6	40	123,0	90	154,9	140	190,9
-55	71,9	-5	97,3	45	126,0	95	158,3	145	194,8
-50	74,3	0	100,0	50	129,1	100	161,8	150	198,7
-45	76,7	5	102,8	55	132,2	105	165,3	155	202,6
-40	79,1	10	105,6	60	135,3	110	168,8	160	206,6
-35	81,6	15	108,4	65	138,5	115	172,4	165	210,7
-30	84,2	20	111,2	70	141,7	120	176,0	170	214,8
-25	86,7	25	114,1	75	145,0	125	179,6	175	219,0
-20	89,3	30	117,1	80	148,3	130	183,3	180	223,2
-15	91,9	35	120,0	85	151,6	135	187,1		

20.8. TEMP_NTC

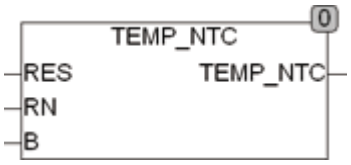
Type FUNKTION : REAL

Input RES : REAL (gemessener Widerstandswert in Ohm)

RN : REAL (Widerstandswert des Sensors bei 25 °C)

B : REAL (Spezifikation des Sensors)

Output REAL (gemessene Temperatur)



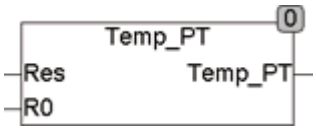
TEMP_NTC errechnet aus dem gemessenen Widerstand und den Parametern des Sensors die gemessene Temperatur. RN ist der Widerstandswert des Sensors bei 25 °C und B ist Abhängig vom Sensor und der Spezifikation des Sensors zu entnehmen.

Der Baustein errechnet die Temperatur nach folgender Formel:

$$T = \frac{b \cdot T_N}{b + \ln\left(\frac{R_H}{R_N}\right) \cdot T_N}$$

20.9. TEMP_PT

Type	Funktion : REAL
Input	RES : REAL (gemessener Widerstandswert in Ohm)
	R0 : REAL (Widerstand bei 0 °C)
Output	REAL (gemessene Temperatur)



TEMP_PT berechnet die Temperatur eines PT-Widerstandsfühlers aus den Eingangswerten RES (gemessener Widerstandswert) und R0 (Widerstand bei 0°C). Wenn die Eingänge eine Temperatur außerhalb des Wertebereichs von -200 .. + 850°C ergeben wird am Ausgang die Temperatur +10000.0 °C ausgegeben.

Die Berechnung erfolgt nach der Formel:

für Temperaturen > 0 °C

$$RES\_PT = R0 * (1 + A*T + B*T^2)$$

und für Temperaturen < 0°C

$$RES\_PT = R0 * (1 + A*T + B*T^2 + C*(T-100)*T^3$$

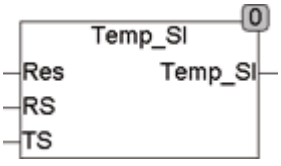
A = 3.90802E-3; B = -5.80195E-7; C = -427350E-12

Auszug aus DIN 43760 für Pt100

°C	R	°C	R	°C	R	°C	R	°C	R	°C	R
-200	18,49	-100	60,25	0	100,00	100	138,50	200	175,84	300	212,02
-195	20,65	-95	62,28	5	101,95	105	140,39	205	177,68	305	213,80
-190	22,80	-90	64,30	10	103,90	110	142,29	210	179,51	310	215,57
-185	24,94	-85	66,31	15	105,85	115	144,17	215	181,34	315	217,35
-180	27,08	-80	68,33	20	107,79	120	146,06	220	183,17	320	219,12
-175	29,20	-75	70,33	25	109,73	125	147,94	225	184,99	325	220,88
-170	31,32	-70	72,33	30	111,67	130	149,82	230	186,82	330	222,65
-165	33,43	-65	74,33	35	113,61	135	151,70	235	188,63	335	224,41
-160	35,33	-60	76,33	40	115,54	140	153,58	240	190,45	340	226,17
-155	37,63	-55	78,32	45	117,47	145	155,45	245	192,26	345	227,92
-150	39,71	-50	80,31	50	119,40	150	157,31	250	194,07	350	229,67
-145	41,79	-45	82,29	55	121,32	155	159,18	255	195,88	355	231,42
-140	43,87	-40	84,27	60	123,24	160	161,04	260	197,69	360	233,17
-135	45,94	-35	86,25	65	125,16	165	162,90	265	199,49	365	234,91
-130	48,00	-30	88,22	70	127,07	170	164,76	270	201,29	370	236,65
-125	50,06	-25	90,19	75	128,98	175	166,61	275	203,08	375	238,39
-120	52,11	-20	92,16	80	130,89	180	168,46	280	204,88	380	240,13
-115	54,15	-15	94,12	85	132,80	185	170,31	285	206,67	385	241,86
-110	56,19	-10	96,09	90	134,70	190	172,16	290	208,45	390	243,59
-105	58,22	-5	98,04	95	136,60	195	174,00	295	210,24	395	245,31

20.10. TEMP_SI

Type	Funktion : REAL
Input	RES : REAL (gemessener Widerstand in Ohm)
	RS : REAL (Widerstand bei 0 °C)
	TS : REAL (Temperatur bei der RS definiert ist)
Output	REAL (gemessene Temperatur)



TEMP_SI berechnet die Temperatur eines SI-Widerstandsfühlers aus den Eingangswerten RES (Widerstand in Ohm) und RS, Widerstand bei TS in °C. Im Gegensatz zu den Bausteinen TEMP_NI und TEMP_PT deren R0 bei 0°C angegeben wird, ist die Widerstandsangabe RS bei SI Fühler bei unterschiedlichen Temperaturen (z.B. 25°C bei KTY10) angegeben. Deshalb hat der Baustein einen Eingang für RS und einen weiteren für TS.

Die Berechnung erfolgt nach der Formel:

$$RES\_SI = RS + A*(T-TS) + B*(T-TS)^2$$

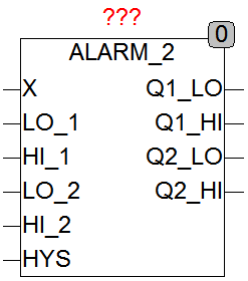
$$A = 7.64E-3; B = 1.66E-5$$

Die Berechnung ist geeignet für einen Temperaturen von -50 .. +150 °C.

21. Messbausteine

21.1. ALARM_2

Type	Funktionsbaustein
Input	X : REAL (Eingangswert)
	RST : BOOL (Reset Eingang für Alarm Ausgang)
Output	LOW : BOOL (TRUE, wenn X < TRIGGER_LOW)



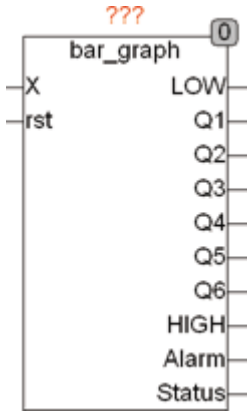
ALARM_2 prüft ob X die Grenzen HI_1 oder HI_2 nach oben übersteigt und setzt dabei die Ausgänge Q1_HI oder Q2_HI auf TRUE. Falls die Grenzen LO_1 oder LO_2 unterschritten werden wird entsprechend Q1_LO oder Q2_LO auf TRUE gesetzt. Die Ausgänge bleiben solange TRUE wie die entsprechende Grenze über- beziehungsweise unter-schritten wird. Um ein Flattern der Ausgänge zu unterbinden kann alternativ auch eine Hysterese HYS eingestellt werden. Wird HYS auf einen Wert > 0 gesetzt, so wird der entsprechende Ausgang erst dann gesetzt wenn die Grenze um mehr als HYS / 2 überschritten beziehungsweise unterschritten wird. Entsprechend muss der Eingang X die Grenze um mehr als HYS / 2 unter beziehungsweise überschreiten bevor die entsprechenden Ausgänge wieder gelöscht werden.

ALARM_2 kann zum Beispiel mit HI_1 und LO_1 den Pegel eines Flüssigkeitsbehälters steuern und mit HI_2 und LO_2 einen Alarm auslösen wenn ein kritischer Pegel unterschritten beziehungsweise überschritten wird.

21.2. BAR_GRAPH

Type	Funktionsbaustein
Input	X : REAL (Eingangswert)
	RST : BOOL (Reset Eingang für Alarm Ausgang)

Output	LOW : BOOL (TRUE, wenn X < TRIGGER_LOW)
	Q1 .. Q6 : BOOL (Triggerausgänge)
	HIGH : BOOL (TRUE, wenn X >= TRIGGER_HIGH)
	ALARM : BOOL (Alarmausgang)
	STATUS : Byte (ESR Status Ausgang)
Setup	TRIGGER_LOW : REAL (Triggerschwelle für LOW Output)
	TRIGGER_HIGH : REAL (Triggerschwelle für HIGH Output)
	ALARM_LOW : BOOL (Enable Alarm für Low Output)
	ALARM_HIGH : BOOL (Enable Alarm für High Output)
	LOG_SCALE : BOOL (Ausgang ist Logarithmisch wenn TRUE)

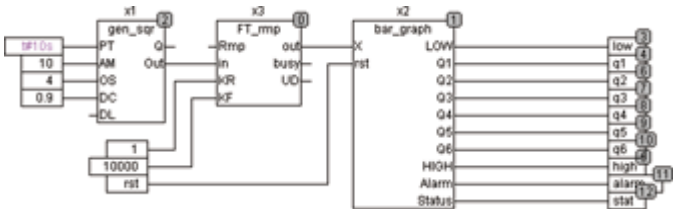


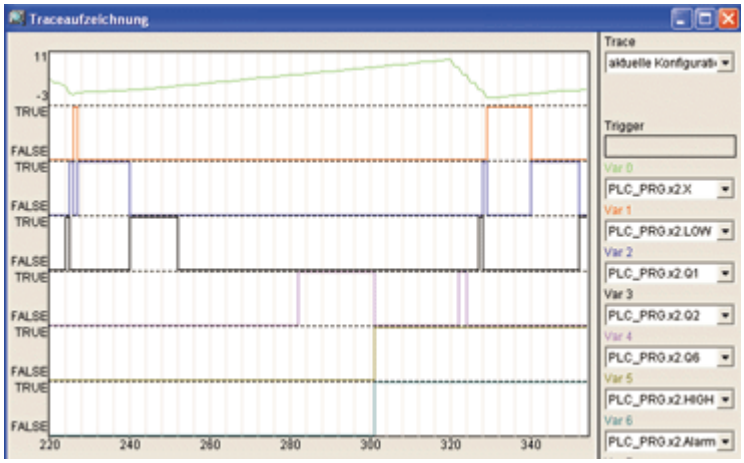
BAR_GRAPH ist ein Level Detector, der abhängig vom Eingangswert einen Ausgang aktiviert. Der Ansprechwert für die LOW und HIGH Ausgänge ist durch die Setup-Variablen TRIGGER_LOW und TRIGGER_HIGH einstellbar. LOW wird TRUE, wenn X kleiner als TRIGGER_LOW ist und HIGH wird TRUE, wenn X größer gleich TRIGGER_HIGH ist. Sind die Setup-Variablen ALARM_LOW und / oder ALARM_HIGH auf TRUE gesetzt, so wird bei unterschreiten von TRIGGER_LOW oder überschreiten von TRIGGER_HIGH der Ausgang ALARM auf TRUE gesetzt und der Ausgang LOW oder HIGH und ALARM bleibt solange auf TRUE, bis der Eingang RST TRUE wird und den Alarm zurücksetzt. Die Ausgänge Q1 bis Q6 unterteilen den Bereich zwischen TRIGGER_LOW und TRIGGER_HIGH in 7 gleiche Bereiche. Ist die Setup Variable LOG_SCALE gesetzt, so wird der Bereich zwischen TRIGGER_LOW und TRIGGER_HIGH logarithmisch aufgeteilt.

Der Ausgang Status ist ein ESR kompatibler Ausgang, der Zustände und Alarme an ESR Bausteine weiterreicht.

Status	
110	Eingang liegt zwischen Trigger_Low und Trigger_High
111	Eingang kleiner Trigger_Low, Ausgang LOW ist TRUE
112	Eingang größer Trigger_High, Ausgang HIGH ist TRUE
1	Eingang kleiner als Trigger_low und Alarm_Low ist TRUE
2	Eingang größer als Trigger_high und Alarm_High ist TRUE

Das folgende Beispiel zeigt eine n Signalverlauf an Bar_Graph:

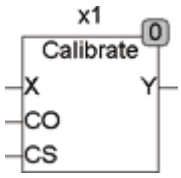




21.3. CALIBRATE

Type	Funktionsbaustein
Input	X : REAL (Eingangswert)
	CO : BOOL (Puls zum Speichern des Offsets)
	CS : BOOL (Puls zum Speichern des Verstärkungsfaktors)
Output	Y : REAL (kalibriertes Ausgangssignal)
Setup	Y_OFFSET : REAL (Y Wert bei dem der Offset gesetzt wird)

Y_SCALE : REAL (Y Wert bei dem der Verstärkungsfaktor gesetzt wird)



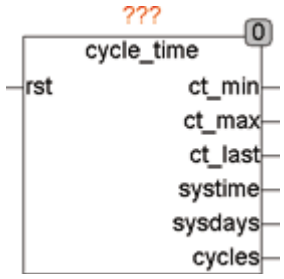
CALIBRATE dient zum Kalibrieren eines Analogen Signals. Um ein Kalibrieren zu ermöglichen müssen 2 Referenzwerte (Y_OFFSET und Y_SCALE) durch einen Doppelklick auf das Symbol des Bausteins gesetzt werden. Y_OFFSET ist der Ausgangswert, bei dem der Offset durch einen Puls an CO gesetzt wird und Y_SCALE ist der Wert, bei dem der Verstärkungsfaktor ermittelt wird. Eine Kalibrierung ist nur dann erfolgreich, wenn zuerst Offset und danach Verstärkung kalibriert werden.

Beispiel:

Ein Eingangssignal von 4..20 mA soll auf die Temperaturwerte von 0 .. 70 °C kalibriert werden. Hierzu werden die Setup- Variablen Y_OFFSET = 0 und Y_SCALE = 70 gesetzt. Dann wird der Sensor in Eiswasser gelegt und nach der Ansprechzeit des Sensors ein Puls am Eingang C0 ausgelöst, der den Baustein veranlasst einen Korrekturwert für Offset zu errechnen und intern abzuspeichern. Als nächstes wird dann der Sensor mit 70 °C beaufschlagt und nach der Ansprechzeit ein Puls am Eingang CS ausgelöst, wodurch im Baustein ein Verstärkungsfaktor berechnet wird, der intern gespeichert wird. Die Kalibrationswerte werden permanent gespeichert. Das heißt, sie gehen auch dann nicht verloren, wenn ein Reset ausgeführt wird oder die Stromversorgung an der SPS ausgeschaltet wird.

21.4. CYCLE_TIME

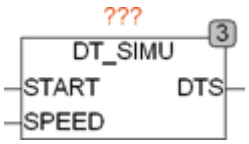
Type	Funktionsbaustein
Input	RES : BOOL (Reset)
Output	CT_MIN : TIME (minimale gemessene Zykluszeit)
	CT_MAX : TIME (maximale gemessene Zykluszeit)
	CT_LAST : TIME (zuletzt gemessene Zykluszeit)
	SYSTIME : TIME (Laufzeit seit dem letzten Start)
	SYSDAYS : INT (Anzahl der Tage seit dem letzten Start)
	CYCLES : DWORD (Anzahl der Zyklen seit dem letzten Start)



CYCLE_TIME überwacht die Zykluszeiten einer SPS und stellt dem Anwender eine Reihe von Informationen über Zykluszeiten und Laufzeiten zur Verfügung. Die Gesamtzahl der Zyklen wird ebenfalls gemessen. Hiermit kann der Anwender zum Beispiel. sicherstellen, dass eine Funktion alle 100 Zyklen aufgerufen wird. Regelbausteine können Fehler melden, wenn die Zykluszeit zu lange wird und deshalb die Regelparameter nicht mehr garantiert werden können.

21.5. DT_SIMU

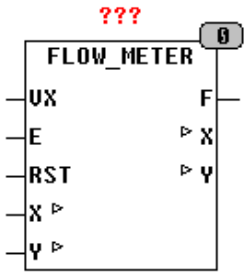
Type	Funktionsbaustein
Input	START : DT (Anfangs DATETIME)
	SPEED : REAL (Geschwindigkeit für den Ausgang DTS)
Output	DTS : DT (Simulierte DATETIME)



DT_SIMU simuliert am Ausgang DTS einen Zeit Datumswert der mit dem Anfangswert START beginnt und mit der Geschwindigkeit SPEED weiter läuft. Wird der Eingangswert SPEED nicht belegt arbeitet der Baustein mit dem internen Vorgabewert 1.0 und der Ausgang DTS läuft mit 1 Sekunde / Sekunde vorwärts. Mit dem Eingang SPEED kann am Ausgang DTS eine beliebig schnelle oder langsame Uhr simuliert werden. Der Baustein kann in der Simulationsumgebung eingesetzt werden um eine RTC zu simulieren und zusätzlich die Geschwindigkeit der RTC zum Testen angepasst werden. Wird der Eingang SPEED = 0 gesetzt, so wird der Ausgang DTS bei jedem SPS Zyklus um eine Sekunde weitergestellt.

21.6. FLOW_METER

Type	Funktionsbaustein
Input	VX : REAL (Volumenstrom je Stunde)
	E : BOOL (Enable Eingang)
	RST : BOOL (Reset Eingang)
I/O	X : REAL (Flussmenge Nachkommateil)
	Y : UDINT (Flussmenge Ganzzahlenteil)
SETUP	PULSE_MODE : BOOL (Puls Zähler wenn TRUE)
	UPDATE_TIME : TIME (Messzeit für F)
Output	F : REAL (momentaner Volumenstrom)



Der Funktionsbaustein FLOW_METER ermittelt den Volumenstrom / Zeiteinheit und Zählt Mengen. FLOW_METER ermittelt den Volumenstrom aus dem Eingang VX und E. Der Baustein unterstützt 2 Betriebsmodi die durch die Setup Variable PULSE_MODE bestimmt werden. Wenn PULSE_MODE = TRUE wird der Volumenstrom und die Menge ermittelt indem bei jeder steigenden Flanke an E der Wert an VX aufaddiert wird. Wenn PULSE_MODE = FALSE wird der Eingang VX als Flussmenge pro Zeiteinheit interpretiert und solange aufsummiert wie E = TRUE ist. Mittels des Eingangs RST kann der interne Zähler jederzeit auf Null gesetzt werden. X und Y sind extern zu deklarierende Variablen die auch Remanent / Permanent deklariert werden können um auch bei Stromausfall erhalten zu bleiben. Der Baustein liefert den momentanen Durchflusswert F als Real abhängig von der an VX anliegenden Einheit. Wird z.B. an VX ein Wert in Liter / Stunde angelegt so ist auch der Messwert am Ausgang F in l/h. Der Ausgang F wird in konstanten Abständen UPDATE_TIME gesetzt. Die Ausgänge X und Y bilden Zusammen den über die Zeit aufsummierten Messwert wobei X in REAL die Nachkommastellen und Y in UDINT den Ganzzahligen Teil darstellen. Ein Zählerstand von 234.111234 wird durch eine 0.111234 an X und einem Wert von 234 an Y dargestellt. Wird zur Zählung nur ein REAL verwendet so beträgt die Auflösung (für Real nach IEEE32) lediglich 7-8 Stellen. Durch die oben beschriebene Methode können insgesamt mehr als 9 Stellen vor dem Komma (2^32-1) und mindestens 7 Stellen nach dem Komma dargestellt werden. Da hierbei X immer kleiner als 1

bleibt kann für Ausgaben ohne Kommastellen nur Y alleine betrachtet werden. Die beiden Variablen X und Y müssen extern deklariert werden und können wie im folgenden Beispiel auch gegen Stromausfall gesichert werden.

Beispiel1:

VX := 4 m³/h;

PULSE_MODE := FALSE;

UPDATE_TIME := T#100ms;

Der Baustein misst den Durchfluss in m³/h und Zählt die Flussmenge solange wie der Eingang E auf TRUE steht. Der Ausgang F wird (4.0m³/h) zeigen solange E TRUE ist, sonst (0.0). Der Wert an F wird alle 100 Millisekunden neu Berechnet.

Beispiel2:

VX := 0,024 l/Puls;

PULS_MODE := TRUE;

UPDATE_TIME := T#1s;

In diesem Beispiel wird der Durchfluss am Ausgang F in l/h angezeigt und mit jeder steigenden Flanke an E wird der Zähler um 0,024 l erhöht.

21.7. M_D

Type	Funktionsbaustein
Input	START : BOOL (Eingangssignal)
	STOP : BOOL (Eingangssignal)
	TMAX : TIME (Timeout für ET)
	RST : BOOL (Reset Eingang)
Output	PT : TIME (gemessene Zeit)
	ET : TIME (Abgelaufene Zeit seit letzter steigender Flanke)
	RUN : BOOL (TRUE wenn Messung läuft)

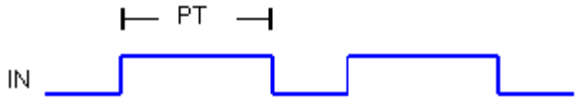
M_D misst die Zeit zwischen einer steigenden Flanke an START und einer steigenden Flanke an STOP. PT ist das Ergebnis der letzten Messung. Der Ausgang ET ist die abgelaufene Zeit seit der letzten steigenden Flanke an START. M_D benötigt eine steigende Flanke, um die Messung zu starten. Falls beim ersten Aufruf START bereits TRUE ist, wird dies nicht als steigende Flanke gewertet. Auch wenn STOP TRUE ist, wird eine steigende Flanke an START nicht gewertet. Nur wenn alle Startbedingungen (STOP = FALSE, RST := FALSE und steigende Flanke an START) vorliegen geht der Ausgang RUN auf TRUE und eine Messung wird gestartet. Mit TRUE am Eingang RST können die Ausgänge jederzeit auf 0 zurückgesetzt werden. Erreicht ET den Wert TMAX wird automatisch im Baustein ein Reset erzeugt und alle Ausgänge auf 0 zurückgesetzt. TMAX ist intern mit einem Vorgabewert von T#10d belegt und kann im Normalfall unbeschaltet bleiben. TMAX dient dazu einen maximalen Wertebereich für PT vorzugeben. Der Ausgang RUN ist TRUE wenn eine Messung läuft.

21.8. M_T

Type	Funktionsbaustein
Input	IN : BOOL (Eingangssignal)
	TMAX : TIME (Timeout für ET)
	RST : BOOL (Reset Eingang)
Output	PT : TIME (gemessene Impulsdauer von der steigenden bis zur fallenden Flanke)
	ET : TIME (Abgelaufene Zeit seit letzter Steigender Flanke)

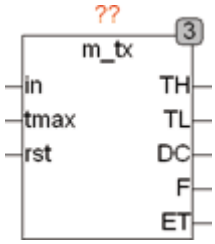
M_T misst die Zeit wie lange IN TRUE war. PT ist die Zeit von der steigenden Flanke des IN Signals bis zur fallenden Flanke des IN Signals. Der Ausgang ET ist die abgelaufene Zeit seit der letzten steigenden Flanke bis zur fallenden Flanke. Solange das Eingangssignal FALSE ist, ist ET = 0. M_T benötigt eine steigende Flanke um die Messung zu triggern. Falls beim ersten Aufruf in bereits TRUE ist, wird dies nicht als steigende Flanke gewertet. Weitere Beispiele sind in der Beschreibung von M_TX zu finden. Mit TRUE am Eingang RST können die Ausgänge jederzeit auf 0 zurückgesetzt werden. Erreicht ET den Wert TMAX wird automatisch im Baustein ein Reset erzeugt und alle Ausgänge auf 0 zurückgesetzt. TMAX

ist intern mit einem Vorgabewert von T#10d belegt und kann im Normalfall unbeschaltet bleiben.



21.9. M_TX

Type	Funktionsbaustein
Input	IN : BOOL (Eingangssignal)
	TMAX : TIME (Timeout für ET)
	RST : BOOL (Reset Eingang)
Output	TH : TIME (Ontime des Eingangssignals)
	TL : TIME (Offtime des Eingangssignals)
	DC : REAL (Tastverhältnis / Duty Cycle des Eingangssignals)
	F : REAL (Frequenz des Eingangssignals in Hz)
	ET : TIME (Vergangene Zeit während der Messung)

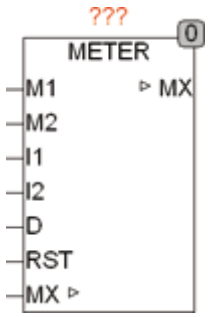


M_TX ermittelt aus dem Eingangssignal IN die Zeit, welche das Signal IN auf TRUE war (TH) und die Zeit die das Signal auf FALSE war (TL). Die Zeiten TH und TL werden nur nach einer steigenden beziehungsweise fallenden Flanke gemessenen. Ist IN beim ersten Aufruf des Bausteins bereits high wird dies nicht als steigende Flanke gewertet. Aus den gemessenen Werten TH und TL werden der Duty Cycle und die Frequenz in Hz errechnet. Ein Duty Cycle von 0,4 bedeutet das Signal war 40% TRUE und 60% FALSE. Der Ausgang ET vom Typ TIME wird mit jeder steigenden Flanke bei 0 gestartet und läuft aufwärts, bis die nächste steigende Flanke ihn wieder bei 0 startet. Mit einem TRUE am Eingang RST können die Ausgänge zu jederzeit auf 0 zurückgesetzt werden. Der Eingang TMAX legt fest, nach welcher abgelaufenen Zeit an ET automatisch die Ausgänge zurückgesetzt werden. TMAX ist intern mit einem Vorgabewert von T#10d belegt und kann im Normalfall offen bleiben. Der Eingang TMAX dient vor allem dazu,

bei fehlendem Eingangssignal nach einer definierten Zeit die Ausgänge zurückzusetzen. Ein Beispiel für eine mögliche Anwendung ist die Messung der Drehzahl einer Welle, die nach Ausbleiben der Sensorsignale die Drehzahl (Frequenz) 0 anzeigt. TMAX ist aber mit Vorsicht anzuwenden, da zum Beispiel ein TMAX von 10 Sekunden gleichzeitig die kleinste Messbare Frequenz auf 0,1 HZ begrenzt.

21.10. METER

Type	Funktionsbaustein
Input	M1 : REAL (Verbrauchswert 1)
	M2 : REAL (Verbrauchswert 2)
	I1 : BOOL (Freigabeeingang 1)
	I2 : BOOL (Freigabeeingang 2)
	D : REAL (Teiler für den Ausgang)
	RST : BOOL (Reset Eingang)
I/O	MX : REAL (Verbrauchswert)



METER ist ein Verbrauchszähler, der 2 unabhängige Eingänge (M1 und M2) über die Zeit aufaddiert. Die Verbrauchszählung wird durch die Eingänge I1 und I2 gesteuert. Mit einem Reset-Eingang RST kann der Zähler jederzeit zurückgesetzt werden. Der Wert M1 wird je Sekunde zum Ausgangswert addiert solange I1 auf TRUE steht und analog wird der Wert M2 je Sekunde zum Ausgang addiert wenn I2 TRUE ist. Sind I1 und I2 TRUE, so wird der Wert M1 + M2 je Sekunde einmal zum Ausgang hinzu addiert. Der Eingang D Teilt den Ausgang MX. Damit können z B. Wattstunden anstatt Wattsekunden gezählt werden. Der Baustein benutzt intern den OSCAT spezifischen Datentyp REAL2 der eine Auflösung von 15 Dezimalstellen erlaubt. Dadurch ist es dem Baustein möglich kleinste Verbrauchswerte an den Eingängen M1 und M2 mit kurzen Zykluszeiten zu erfassen und zu hohen Gesamtwerten am Ausgang MX addieren. Die Auflösung des Bausteins kann wie folgt ermittelt werden. MX ist als I/O definiert und muss auf eine externe Variable vom Typ REAL gelegt werden. Die externe Varia-

ble kann auf Wunsch als Remanent und / oder Persistent deklariert werden um den Wert bei Spannungsausfall zu erhalten.

MX / 10^15 entspricht der minimalen Auflösung an den Eingängen M1 und M2.

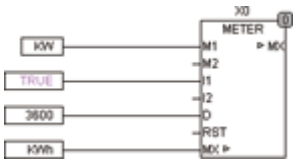
Beispiel:

MX = 10E6	der Verbrauchszähler steht bei 10 MWh
M1 = 0.09 Watt	Momentanverbrauch liegt bei 0.1 Watt
D = 3600	Ausgang arbeitet in Wh (Wattstunden)

Zykluszeit beträgt 10ms

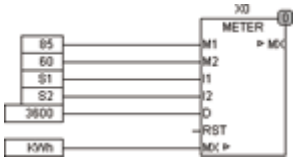
In diesem Beispiel wird je Zyklus ein Wert von 0.09[W] * 0.01 [S] / 3600 = 2.5E-7[Wh] zum Ausgang MX addiert. Dies entspricht einer Veränderung an der 14 Dezimalstelle des Ausgangs.

Beispiel 1 Stromverbrauchszähler:



Der Stromverbrauchszähler zählt die Kilowattsekunden am Eingang M1. Durch den Eingang D wird der Ausgang durch 3600 geteilt, sodass der Ausgang Kilowattstunden anzeigt.

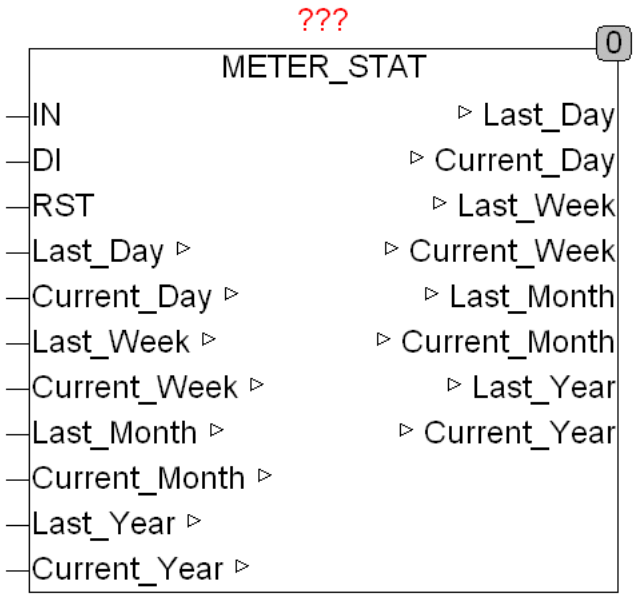
Beispiel 2 Verbrauchsberechnung für einen 2 Stufen Brenner:



In diesem Beispiel ist die Leistung von Stufe 1 (M1) 85KW und Stufe 2 (M2) 60KW. Die Eingänge S1 und S1 (I1 und I2) werden TRUE, wenn die entsprechende Stufe läuft. Durch die Konstante 3600 an D wird der Ausgang durch 3600 geteilt, sodass Kilowattstunden angezeigt werden.

21.11. METER_STAT

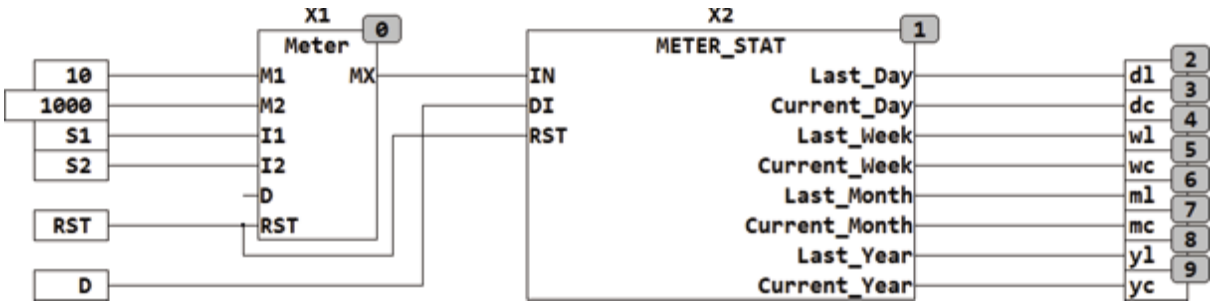
Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	DI : DATE (Datumseingang)
	RST : BOOL (Reset Eingang)
I/O	LAST_DAY : REAL (Verbrauchswert des vergangenen Tages)
	CURRENT_DAY : REAL (Verbrauchswert des aktuellen Tages)
	LAST_WEEK : REAL (Verbrauchswert der vergangenen Woche)
	CURRENT_WEEK : REAL (Verbrauchswert der aktuellen Woche)
	LAST_MONTH : REAL (Verbrauchswert des letzten Monats)
	CURRENT_MONTH : REAL (Verbrauch des aktuellen Monats)
	LAST_YEAR : REAL (Verbrauchswert des vergangenen Jahres)
	CURRENT_YEAR : REAL (Verbrauchswert des aktuellen Jahres)



METER_STAT errechnet den Verbrauch des aktuellen Tages, Woche, Monat und Jahr und zeigt den Wert des letzten Vergleichszeitraumes. Der aufad-

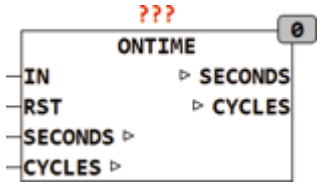
dierte Verbrauchswert liegt am Eingang IN, während am Eingang DI das aktuelle Datum anliegt. Mit dem Eingang RST kann der Zähler jederzeit zurückgesetzt werden. Zur einfacheren Speicherung im Persistent- und Remanent-speicher sind die Ausgänge als I/O definiert.

Das folgende Beispiel zeigt die Anwendung von METER_STAT mit dem Baustein METER:



21.12. ONTIME

Type	Funktionsbaustein
Input	IN : BOOL (Eingangssignal)
	RST : BOOL (Reset Eingang)
Output	SECONDS : UDINT (Betriebszeit in Sekunden)
	CYCLES : UDINT (Einschaltzyklen des Eingangs IN)



ONTIME ist ein Betriebsstundenzähler. Es wird die gesamte Zeit aufsummiert, die das Signal IN seit dem letzten Reset auf TRUE war. Zusätzlich wird die Anzahl der gesamten Ein / Aus Zyklen ermittelt. Die Ausgabewerte sind vom Typ UDINT. Mit dem Eingang RST können die Ausgangswerte jederzeit zurückgesetzt werden. Die Ausgangswerte sind nicht in Variablen des Bausteins gespeichert, sondern werden extern angelegt und über IO (POINTER) angebunden. Dies hat den entscheidenden Vorteil das je nach Wunsch des Anwenders die Variablen als RETAIN und / oder PERSISTENT festgelegt werden können. Es ist damit auch möglich alte Betriebsstunden abzuspeichern und später wieder herzustellen, zum Beispiel bei CPU Wechsel.

Die Deklaration der Variablen an den Eingängen SECONDS und CYCLES müssen vom Typ UDINT sein und können wahlweise als VAR, VAR RETAIN oder VAR RETAIN PERSISTENT angelegt werden.

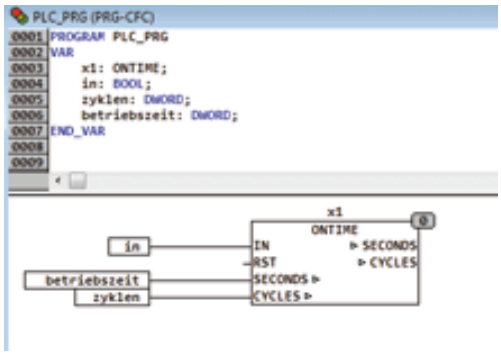
Die DEKLARATION Der Variablen für die Betriebszeit und die Zyklen muss vom Typ UDINT sein und kann alternativ RETAIN und oder PERSISTENT erfolgen.

```
VAR RETAIN PERSISTENT
        Betriebszeit_in_Sekunden : UDINT;
        Zyklen : UDINT;
END_VAR
```

Die folgende Tabelle erläutert RETAIN und PERSISTENT:

x = Wert bleibt erhalten - = Wert wird neu initialisiert

nach Online-Befehl	VAR	VAR RETAIN	VAR PERSISTENT	VAR RETAIN PERSISTENT VAR PERSISTENT RETAIN
Reset	-	x	-	x
Reset Kalt	-	-	-	-
Reset Ursprung	-	-	-	-
Laden (=Download)	-	-	x	x
Online Change	x	x	x	x

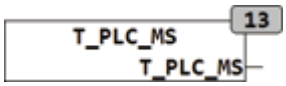


Variablen vom Typ Retain und Persistent behalten ihren Wert bei Download, Online Change und Reset. Bei einem Reset-Kalt oder Reset-Ursprung jedoch verlieren auch diese Variablen Ihre Werte. Der Anwender kann jedoch die Werte im File-System oder in Netzwerk abspeichern und sie selbst z.B. nach einem Wechsel der CPU wiederherstellen .

21.13. T_PLC_MS

Type Funktion : DWORD

Output DWORD (SPS Timer in Millisekunden)



T_PLC_MS liefert die aktuelle interne SPS Zeit in Millisekunden. Dies hat nichts mit einer eventuell vorhandenen Uhr (Real Time Baustein) zu tun, sondern ist der interne Timer einer SPS, der als Zeitreferenz benutzt wird.

Der Quelltext des Bausteins hat folgende Eigenschaften:

```
FUNCTION T_PLC_MS : DWORD
VAR CONSTANT
        DEBUG : BOOL := FALSE;
        N : INT := 0;
        OFFSET := 0;
END_VAR
VAR
        TEMP : DWORD := 1;
END_VAR
T_PLC_MS := TIME_TO_DWORD(TIME());
IF DEBUG THEN
        T_PLC_MS := SHL(T_PLC_US,N) OR SHL(TEMP,N)-1 + OFFSET;
END_IF;
```

Im Normalbetrieb liest der Baustein mit der Funktion TIME() den internen Timer der SPS aus und liefert diesen zurück. Der Interne Timer der SPS nach IEC Norm hat 1 Millisekunde Auflösung.

Eine weitere Eigenschaft von T_PLC_MS ist ein Debug-Modus, der es erlaubt den Überlauf des SPS internen Timers zu testen und die erstellte Software entsprechend sicher zu verifizieren. Der interne Timer jeder SPS hat unabhängig von Hersteller und Art der Implementierung nach einer festen Zeit einen Überlauf. Das heißt, er läuft gegen FF..FFFF (höchster Wert der im entsprechenden Typ gespeichert werden kann) und beginnt dann wieder bei 000..0000. bei Standard SPS Timern beträgt diese Überlaufzeit 2^32 -1 Millisekunden, was in etwa 49,71 Tagen entspricht. Da es sich bei diesem Timer um einen in Hardware implementierten Timer handelt kann auch sein Anfangswert nicht gesetzt werden, sodass er nach dem Einschalten der SPS immer bei 0 anfängt und bis zum Maximalwert hoch läuft. Nach erreichen des Maximalwertes entsteht dann der berüchtigte Timer-Überlauf, der fatale Auswirkungen in der Anwendungssoftware hervor ruft, aber nur extrem schwer getestet werden kann.

T_PLC_MS bietet mehrere Möglichkeiten zum Testen des Überlaufs und zeitabhängiger Software. Mit der Konstante DEBUG kann der Test-Modus eingeschaltet werden und dann mittels der Konstanten N und Offset der Timer ab einem bestimmten Wert beginnen, damit gezielt der Überlauf getestet werden kann ohne die 49 Tage abzuwarten. Offset legt hierbei den Wert fest, der zum internen Timer addiert wird. Mit der Konstanten N wird festgelegt, um wie viele Bits der interne Timer Wert nach links verschoben wird und dabei die unteren N Bits mit 1 gefüllt werden. Mit N kann dadurch die Geschwindigkeit des internen Timers um die Faktoren 2,4,8,16 usw. erhöht werden.

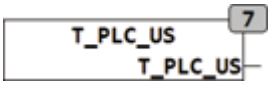
T_PLC_US bietet also alle Möglichkeiten zum Test zeitabhängiger Software, sowohl für die Problematik des Überlaufs, als auch für sehr langsame zeitabhängige Funktionen.

Die Konstanten DEBUG, N und OFFSET wurden absichtlich nicht als Eingänge der Funktion implementiert um eine versehentliche Fehlbedienung zu vermeiden.

21.14. T_PLC_US

Type Funktion : DWORD

Output DWORD (SPS Timer in Mikrosekunden)



T_PLC_US liefert die aktuelle interne SPS Zeit in Mikrosekunden. Dies hat nichts mit einer eventuell vorhandenen Uhr (Real Time Baustein) zu tun, sondern ist der interne Timer einer SPS der als Zeitreferenz benutzt wird.

Der Quelltext des Bausteins hat folgende Eigenschaften:

```
FUNCTION T_PLC_US : DWORD
VAR CONSTANT
        DEBUG : BOOL := FALSE;
        N : INT := 0;
        OFFSET := 0;
END_VAR
VAR
        TEMP : DWORD := 1;
END_VAR
T_PLC_US := TIME_TO_DWORD(TIME())*1000;
IF DEBUG THEN
```

```
        T_PLC_US := SHL(T_PLC_US,N) OR SHL(TEMP,N)-1 + OFFSET;
END_IF;
```

Im Normalbetrieb liest der Baustein mit der Funktion TIME() den internen Timer der SPS aus. Da der interne Timer der SPS nach IEC Norm nur mit 1 Millisekunde Auflösung bietet wird der gelesene Wert mit 1000 multipliziert um den Wert in Mikrosekunden zurück zu liefern. Diese Funktion wurde aus Kompatibilitätsgründen so erstellt, damit sie auch auf Steuerungen die keine bessere Auflösung als Millisekunden bieten ein Mikrosekunden Timer zur Verfügung stellt, der dann in anderen Bausteinen verwendet werden kann. Falls die vorhandene SPS Mikrosekunden unterstützt, kann diese Funktion ganz einfach nur an dieser einen Stelle entsprechend angepasst werden und die Genauigkeit ändert sich durch diesen simplen Eingriff in allen Modulen die diese Funktion aufrufen. Die Software bleibt so portierbar und zukunftssicher. Bereits heute unterstützen praktisch alle SPS Controller eine Auflösung in Mikrosekunden. Diese wird allerdings nicht über standardisierte Routinen ausgelesen, sondern herstellerspezifisch und nicht standardisiert zur Verfügung gestellt. Der Baustein T_PLC_US stellt also ein geeignetes Interface zu diesen herstellerspezifischen Timern her.

Eine weitere Eigenschaft von T_PLC_US ist ein Debug-Modus, der es erlaubt den Überlauf des SPS internen Timers zu erzeugen und die erstellte Software entsprechend sicher zu testen. Der interne Timer jeder SPS hat unabhängig von Hersteller und Art der Implementierung nach einer festen Zeit einen Überlauf. Das heißt er läuft gegen FF..FFFF (höchster Wert der im entsprechenden Typ gespeichert werden kann und beginnt dann wieder bei 000..0000. bei Standard SPS Timern beträgt diese Überlaufzeit 2^32 -1 Millisekunden, was etwa 49,71 Tagen entspricht. Da es sich bei diesem Timer um einen in Hardware implementierten Timer handelt kann auch sein Anfangswert nicht gesetzt werden, sodass er nach dem Einschalten der SPS immer bei 0 anfängt und bis zum Maximalwert hoch läuft. Nach erreichen des Maximalwertes entsteht dann der berüchtigte Timer-Überlauf, der fatale Auswirkungen in der Anwendungssoftware hervor ruft, aber nur extrem schwer getestet werden kann.

T_PLC_US bietet mehrere Möglichkeiten zum testen des Überlaufs und zeitabhängiger Software. Mit der Konstante DEBUG kann der Test Modus eingeschaltet werden und dann mittels der Konstanten N und Offset der Timer ab einem bestimmten Wert beginnen, damit gezielt der Überlauf getestet werden kann ohne die 49 Tage abzuwarten. Offset legt hierbei den Wert fest der zum internen Timer addiert wird. Mit der Konstanten N wird festgelegt, um wie viele Bits der interne Timer Wert nach links verschoben wird und dabei die unteren N Bits mit 1 gefüllt werden. Mit N kann dadurch die Geschwindigkeit des internen Timers um die Faktoren 2,4,8,16 usw. erhöht werden.

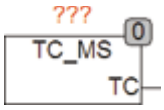
T_PLC_US bietet also alle Möglichkeiten zum Test zeitabhängiger Software, sowohl für die Problematik des Überlaufs, als auch für sehr langsame zeitabhängige Funktionen. Die Konstanten DEBUG, N und OFFSET wurden ab-

sichtlich nicht als Eingänge der Funktion implementiert, um eine versehentliche Fehlbedienung zu vermeiden.

21.15. TC_MS

Type Funktionsbaustein

Output TC : DWORD (letzte Zykluszeit in Millisekunden)

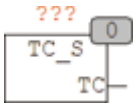


TC_MS ermittelt die letzte Zykluszeit, das ist die Zeit die seit dem letzten Aufruf des Bausteins vergangen ist. Die Zeit wird in Millisekunden geliefert.

21.16. TC_S

Type Funktionsbaustein

Output TC : REAL (letzte Zykluszeit in Sekunden)

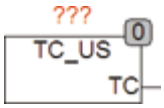


TC_S ermittelt die letzte Zykluszeit, das ist die Zeit die seit dem letzten Aufruf des Bausteins vergangen ist. Die Zeit wird in Sekunden geliefert, hat aber eine Genauigkeit in Mikrosekunden. Der Baustein ruft die Funktion T_PLC_US() auf. T_PLC_US() liefert den SPS internen Timer in Mikrosekunden mit einer Schrittweite von 1000 Mikrosekunden. Wird eine höhere Auflösung gewünscht muss die Funktion T_PLC_US() dem entsprechenden System angepasst werden.

21.17. TC_US

Type Funktionsbaustein

Output TC : DWORD (letzte Zykluszeit in Mikrosekunden)

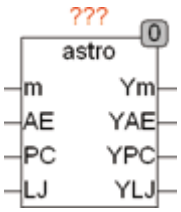


TC_US ermittelt die letzte Zykluszeit, das ist die Zeit die seit dem letzten Aufruf des Bausteins vergangen ist. Die Zeit wird in Mikrosekunden geliefert. Der Baustein ruft die Funktion T_PLC_US() auf. T_PLC_US() liefert den SPS internen Timer in Mikrosekunden mit einer Schrittweite von 1000 Mikrosekunden. Wird eine höhere Auflösung gewünscht muss die Funktion T_PLC_US() dem entsprechenden System angepasst werden.

22. Umrechnungen

22.1. ASTRO

Type	Funktionsbaustein
Input	M : REAL (Entfernung in Meter)
	AE : REAL (Entfernung in Astronomischen Einheiten)
	PC : REAL (Entfernung in Parsec)
	LJ : REAL (Entfernung in Lichtjahren)
Output	YM : REAL (Entfernung in Meter)
	YAE : REAL (Entfernung in Astronomischen Einheiten)
	YPC : REAL (Entfernung in Parsec)
	YLJ : REAL (Entfernung in Lichtjahren)



Der Baustein ASTRO rechnet verschiedene in der Astronomie gebräuchliche Entfernungseinheiten um. Normalerweise wird nur der zu konvertierende Eingang belegt und die restlichen Eingänge bleiben frei. Werden jedoch mehrere Eingänge gleichzeitig mit Werten beaufschlagt, so werden die Werte aller Eingänge entsprechend umgewandelt und dann aufsummiert.

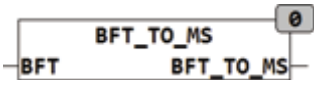
1 AE = 149,597870 * 109 m

1 PC = 206265 AE

1 LJ = 9,460530 * 1015 m = 63240 AE = 0,30659 PC

22.2. BFT_TO_MS

Type	Funktion
Input	BFT : INT (Windstärke nach der Beaufort Skala)
Output	MS : REAL (Windstärke in Meter / Sekunde)



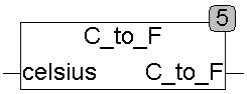
BFT_TO_MS rechnet Windgeschwindigkeiten nach der Beaufort Skala in Meter Pro Sekunde um.

Die Berechnung erfolgt nach der Formel:

BFT_TO_MS = 0.836m/s * B^3/2

22.3. C_TO_F

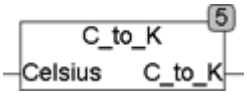
Type	Funktion : REAL
Input	CELSIUS : REAL (Temperaturwert in °C)
Output	REAL (Temperaturwert in Fahrenheit)



C_TO_F rechnet einen Temperaturwert von Celsius in Fahrenheit um.

22.4. C_TO_K

Type	Funktion : REAL
Input	CELSIUS : REAL (Temperaturwert in °C)
Output	REAL (Temperaturwert in Kelvin)

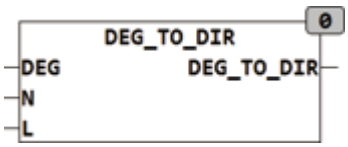


C_TO_K rechnet einen Temperaturwert von Celsius in Kelvin um.

22.5. DEG_TO_DIR

Type	Funktion : STRING(3)

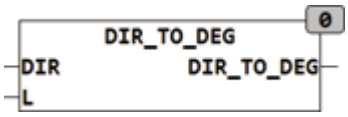
Input	DEG : INT (Himmelsrichtung in Grad)
	N : INT (Maximale Länge der Zeichenkette)
	L : INT (Spracheinstellung: siehe Sprachdefnitionen)
Output	STRING(3) (Kompassangaben)



DEG_TO_DIR rechnet eine Himmelsrichtung (0..360 Grad) in Kompass Angaben um. Am Eingang DEG liegt die Himmelsrichtung in Grad an (0 = Nord, 90 = Ost, 180 = Süd und 270 = Westen). Der Ausgang stellt die Himmelsrichtung als String in der Form NNO zur Verfügung. Mit dem Eingang N wird die Maximale Länge der Richtungsangabe begrenzt. Wenn N=1 werden nur in die 4 Himmelsrichtungen N, E, S, W aufgelöst. Ist N = 2 wird zwischen jede Himmelsrichtung eine weitere eingefügt: NE, SE, SW, NW. Bei N=3 werden auch Richtungen wie NNO ... aufgelöst, Mit N = 3 werden insgesamt 16 Richtungen Ausgewertet. Der Eingang L erlaubt das Umschalten der im Sprachen Setup definierten Sprachen. 0L=0 bedeutet Default Sprache, eine Zahl > 0 ist eine der Vordefinierten Sprachen. nähere Infos zu den Vordefinierten Sprachen finden sie unter Datentypen CONSTANTS_LANGUAGE.

22.6. DIR_TO_DEG

Type	Funktion : INT
Input	DIR : STRING(3) (Himmelsrichtung in Kompassangaben)
	L : INT (Sprachauswahl)
Output	INT (Himmelsrichtung in Grad)



DIR_TO_DEG wandelt eine Himmelsrichtung in der Form NNO in Grad um. Es werden dabei bis zu 3 Stellen ausgewertet, was einer Auflösung von 22.5° entspricht. Der Ausgang ist vom Typ Integer. Der Eingang muss in Großbuchstaben vorliegen und Osten darf mit O oder E bezeichnet werden. Die Zeichenkette NO wird in 45° gewandelt. L spezifiziert die zu ver-

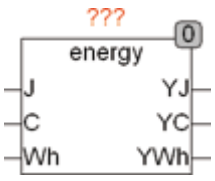
wendende Sprache, für detaillierte Informationen siehe Datentyp CONSTANTS_LANGUAGE.

Die Himmelsrichtungen sind: 0° = Nord, 90° = Ost, 180° = Süd und 270° = Westen. Die Umwandlung erfolgt nach folgender Tabelle:

N	0°	NNO, NNE	23°	NO	45°	ONO, ENE	68°
O	90°	OSO, ESE	113°	SO, SE	135°	SSO, SSE	158°
S	180°	SSW	203°	SW	225°	WSW	248°
W	270°	WNW	293°	NW	315°	NNW	338°

22.7. ENERGY

Type	Funktionsbaustein
Input	J : REAL (Joule)
	C : REAL (Kalorie)
	WH : REAL (Wattstunden)
Output	YJ : REAL (Joule)
	YC : REAL (Kalorie)
	YWH : REAL (Wattstunden)



Der Baustein ENERGY konvertiert verschiedene in der Praxis gebräuchliche Einheiten für Energie. Normalerweise wird nur der zu konvertierende Eingang belegt und die restlichen Eingänge bleiben frei. Werden jedoch mehrere Eingänge gleichzeitig mit Werten beaufschlagt, so werden die Werte aller Eingänge entsprechend umgewandelt und dann aufsummiert.

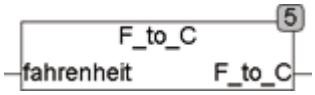
1 J = 1 Ws (Watt * Sekunden) = 1 Nm (Newton * Meter)

1 C = 4,1868 J = 1,163 * 10-3 Wh (Watt * Stunden)

1 Wh = 3,6 * 103 J = 860 C

22.8. F_TO_C

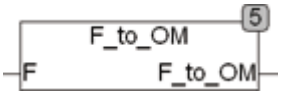
Type	Funktion : REAL
Input	FAHRENHEIT : REAL (Temperaturwert in Fahrenheit)
Output	REAL (Temperaturwert in °C)



F_TO_C rechnet einen Temperaturwert von Fahrenheit in °C um.

22.9. F_TO_OM

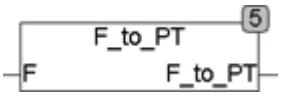
Type	Funktion : REAL
Input	F : REAL (Frequenz in Hz)
Output	REAL (Frequenz in Hz)



F_TO_OM berechnet die Kreisfrequenz Omega aus der Frequenz in Hz.

22.10. F_TO_PT

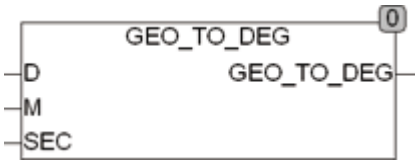
Type	Funktion : REAL
Input	F : REAL (Frequenz)
Output	TIME (Periodendauer)



F_TO_PT rechnet einen Frequenzwert von Hz in die entsprechende Periodendauer um.

22.11. GEO_TO_DEG

Type	Funktion : REAL
Input	D : INT (Winkel in Grad)
	M : INT (Winkelminuten)
	SEC : REAL (Winkelsekunden)
Output	REAL (Winkelangabe in Dezimal Grad)

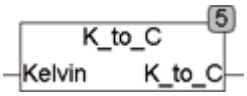


GEO_TO_DEG errechnet eine Winkelangabe in Grad aus den Eingangsdaten Grad. Minuten, Sekunden.

GEO_TO_DEG(2,59,60.0) ergibt 3.0 Grad

22.12. K_TO_C

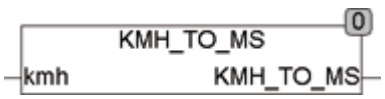
Type	Funktion : REAL
Input	KELVIN : REAL (Temperaturwert in Kelvin)
Output	REAL (Temperaturwert in °C)



K_TO_C rechnet einen Temperaturwert von Kelvin in °C um.

22.13. KMH_TO_MS

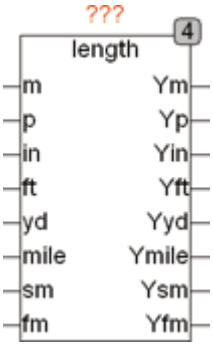
Type	Funktion : REAL
Input	KMH : REAL (Geschwindigkeit in m/s)
Output	TIME (Geschwindigkeit in km/h)



KMH_TO_MS rechnet einem Geschwindigkeitswert von Kilometer / Stunde in Meter / Sekunde um. KMH_TO_MS := KMH / 3.6

22.14. LENGTH

Type	Funktionsbaustein
Input	M : REAL (Meter)
	P : REAL (Typographischer Punkt)
	IN : REAL (Inch)
	FT : REAL (Foot)
	YD : REAL (Yard)
	MILE : REAL (Mile)
	SM : REAL (Internationale Seemeile)
	FM : REAL (Fathom)
Output	YM : REAL (Meter)
	YP : REAL (Typographischer Punkt)
	YIN : REAL (Inch)
	YFT : REAL (Foot)
	YYD : REAL (Yard)
	YMILE : REAL (Mile)
	YSM : REAL (Internationale Seemeile)
	YFM : REAL (Fathom)



Der Baustein LENGTH konvertiert verschiedene in der Praxis gebräuchliche Einheiten für Längeneinheiten. Normalerweise wird nur der zu konvertierende Eingang belegt und die restlichen Eingänge bleiben frei. Werden jedoch mehrere Eingänge gleichzeitig mit Werten beaufschlagt, so werden

die Werte aller Eingänge entsprechend umgewandelt und dann aufsummiert.

1 P = 0,376065 mm (Einheit aus dem Druckereigewerbe)

1 IN = 25,4 mm

1 FT = 0,3048 m

1 YD = 0,9144 m

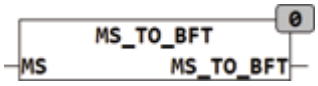
1 MILE = 1609,344 m

1 SM = 1852 m

1 FM = 1,829 m

22.15. MS_TO_BFT

Type	Funktion
Input	MS : INT (Windstärke nach der Beaufort Skala)
Output	MS : REAL (Windstärke in Meter / Sekunde)



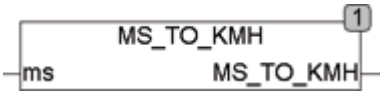
MS_TO_BFT rechnet Windgeschwindigkeiten von Meter / Sekunde in die Beaufort Skala um.

Die Berechnung erfolgt nach der Formel:

MS_TO_BFT = (MS * 1.196172)^2/3

22.16. MS_TO_KMH

Type	Funktion : REAL
Input	MS : REAL (Geschwindigkeit in km/h)
Output	REAL (Geschwindigkeit in m/s)

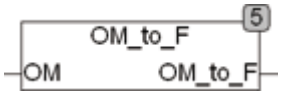


MS_TO_KMH rechnet einem Geschwindigkeitswert von Meter / Sekunde in Kilometer / Stunde um.

MS_TO_KMH := MS * 3.6

22.17. OM_TO_F

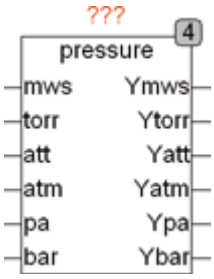
Type Funktion : REAL
Input OM : REAL (Kreisfrequenz Omega)
Output REAL (Frequenz in Hz)



OM_TO_F berechnet die Frequenz in Hz aus der Kreisfrequenz Omega.

22.18. PRESSURE

Type Funktionsbaustein
Input MWS : REAL (Wassersäule in Meter)
TORR : REAL (Torr Bzw. Quecksilbersäule in mm)
ATT : REAL (Atmosphäre technisch)
ATM : REAL (Atmosphäre physikalisch)
PA : REAL (Pascal)
BAR : REAL (Bar)
Output YMWS : REAL (Wassersäule in Meter)
YTORR : REAL (Torr Bzw. Quecksilbersäule in mm)
YATT : REAL (Atmosphäre technisch)
YATM : REAL (Atmosphäre physikalisch)
YPA : REAL (Pascal)
YBAR : REAL (Bar)



Der Baustein PRESSURE konvertiert verschiedene in der Praxis gebräuchliche Einheiten für Druck. Normalerweise wird nur der zu konvertierende Eingang belegt und die restlichen Eingänge bleiben frei. Werden jedoch mehrere Eingänge gleichzeitig mit Werten beaufschlagt, so werden die Werte aller Eingänge entsprechend umgewandelt und dann aufsummiert.

1 MWS = 1 Meter Wassersäule = 0,0980665 Bar

1 TORR = 1 mm Quecksilbersäule = 0,133322 Bar = 101325 / 760 Pa

1 ATT = 1 kp / cm² = 0,980665 Bar

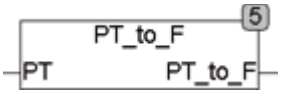
1 ATM = 1,01325 Bar

1 PA = 1 N / m²

1 BAR = 105 Pa

22.19. PT_TO_F

Type	Funktion : REAL
Input	PT : TIME (Periodendauer in Sekunden)
Output	REAL (Frequenz in Hz)



PT_TO_F rechnet eine Periodendauer in Sekunden in die entsprechende Frequenz in Hz um.

22.20. SPEED

Type	Funktionsbaustein
Input	MS : REAL (Meter / Sekunde)

KMH : REAL (Kilometer / Stunde)

KN : REAL (Knoten = Seemeilen / Stunde)

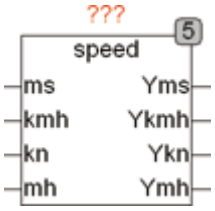
MH : REAL (Meilen / Stunde)

Output YMS : REAL (Meter / Sekunde)

YKMH : REAL (Kilometer / Stunde)

YKN : REAL (Knoten = Seemeilen / Stunde)

YMH : REAL (Meilen / Stunde)



Der Baustein SPEED konvertiert verschiedene in der Praxis gebräuchliche Einheiten für Geschwindigkeiten. Normalerweise wird nur der zu konvertierende Eingang belegt und die restlichen Eingänge bleiben frei. Werden jedoch mehrere Eingänge gleichzeitig mit Werten beaufschlagt, so werden die Werte aller Eingänge entsprechend umgewandelt und dann aufsummiert.

1 ms = Meter / Sekunde = 3,6 km/h

1 kmh = Kilometer / Stunde = 1/3,6 m/s

1 kn = Knoten = 1 Seemeile / Stunde = 0,5144 m/s

1 mh = Meilen / Stunde = 0,44704 m/s

22.21. TEMPERATURE

Type Funktionsbaustein

Input K : REAL (Temperatur nach Kelvin Skala)

C : REAL (Temperatur nach Celsius Skala)

F : REAL (Temperatur nach Fahrenheit Skala)

RE : REAL (Temperatur nach Reaumur Skala)

RA : Real (Temperatur nach Rankine Skala)

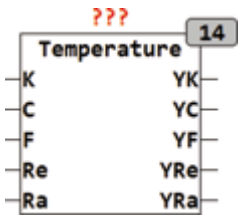
Output YK : REAL (Temperatur nach Kelvin Skala)

YC : REAL (Temperatur nach Celsius Skala)

YF : REAL (Temperatur nach Fahrenheit Skala)

YRE : REAL (Temperatur nach Reaumur Skala)

YRA : Real (Temperatur nach Rankine Skala)



Der Baustein TEMP konvertiert verschiedene in der Praxis gebräuchliche Einheiten für Temperatur. Normalerweise wird nur der zu konvertierende Eingang belegt und die restlichen Eingänge bleiben frei. Werden jedoch mehrere Eingänge gleichzeitig mit Werten beaufschlagt, so werden die Werte aller Eingänge entsprechend umgewandelt und dann aufsummiert.

1 K = 273.15 °C

1 °C = 273.15 K

1 °F = °C * 1.8 + 32

1 Re = °C * 0.8

1 Ra = K * 1.8

23. Regelungstechnik

23.1. Einleitung

Im Bereich der Regelungstechnik werden Bausteine zum Aufbau von Reglern und Regelstrecken zur Verfügung gestellt. Soweit möglich messen die Bausteine selbst die Zykluszeit und errechnen mit der jeweils aktuellen Zykluszeit die Ausgangsveränderung. Dieses Verfahren hat gegenüber einer fest eingestellten Zykluszeit den Vorteil das Regelstrecken verschiedenster Geschwindigkeit innerhalb derselben Task verarbeitet werden können. Ein weiterer Vorteil ist die Tatsache das bei niedrig priorisierten Tasks die Zykluszeit schwanken kann und ein Regler mit fixer Zykluszeit ungenaue Ausgangswerte erzeugt. Der Anwender hat beim Einsatz sicherzustellen das die Zykluszeit der Task entsprechend den Anforderungen der Regelstrecke eingestellt ist.

Übersicht über die Regelkreiselemente:

TYP	Name	Parameter	Übertragungsfunktion	Berechnung
P	Proportionalglied	KP	KP	Y = X * KP
I	Integrator	KI		Y = Ya + X * KI * ΔT
D	Differentiator	KD		Y = KD * ΔX / ΔT
PT1	Tiefpaß 1.Ordnung	T1	KP / (1 + T1s)	
PT2	Tiefpaß 2.Ordnung	T1, T2		
PI	PI-Glied	KP, KI	KP (1 + 1/TNs)	Y = Ya + KP ((1 + ΔT/TN)X - Xa)
PD	PD-Glied	KP, KD	KP (1 + TDs)	
PDT1	PD-Glied, Verzögert	KP, TV, T1	KP (1 + TVs/(1+T1s))	
PID	PID-Glied	KP, TN, TV	KP (1 + 1/TNs + TVs	
PIDT1	PID-Glied, Verzögert	KP, TN, TV, T1	KP (1 + 1/TNs + TVs/(1+T1s))	

Wind-Up:

Der Wind-Up Effekt betrifft alle Regler mit I-Anteil. Da reale Regler einen eingeschränkten Stellbereich haben würde der Integrator bei großen Regelabweichungen und erreichen eines Ausgangslimits stets weiter anwachsen. Würde nach einiger Zeit der Prozesswert den Sollwert übersteigen so müsste abgewartet werden bis der Integrator seinen hohen Wert wieder abgebaut hat. Dies führt zu einem unerwünschten und ungünsti-

gen Verhalten des Reglers. Der Regler würde für die Zeit aussetzen die der Integrator benötigt seinen hohen Wert wieder abzubauen, was um so länger wäre je länger der Regler in der Begrenzung war. Deshalb sind bei Reglern mit I-Anteil Anti Wind-Up Maßnahmen notwendig.

Die einfachste Maßnahme zur Verhinderung des Wind-Up ist den Integrator bei erreichen eines Limits anzuhalten und erst bei Rückkehr in den Arbeitsbereich mit dem letzten Wert des Integrators weiterzuarbeiten. Dieses Verfahren hat allerdings den Nachteil das Veränderungen der Regelabweichung während der Ausgang an einem Limit ansteht weiterhin zu unnötig erhöhten Werten des Integrators führen können. Bausteine der Bibliothek die mit diesem Verfahren arbeiten sind mit einen W am Ende gekennzeichnet.

Eine verfeinerte Anti Wind-Up Maßnahme ist ein Verfahren das den Ausgangswert des Integrators auf einen Wert begrenzt der zusammen mit den anderen Regelanteilen exakt zu dem Ausgangslimit führt. Dieses Verfahren hat den Vorteil das bei eintritt in den Arbeitsbereich der Regler ohne Zeitverzug sofort Einsatzfähig ist und reagieren kann. Bausteine der Bibliothek die mit diesem verbesserten Verfahren arbeiten sind mit einen WL am Ende gekennzeichnet.

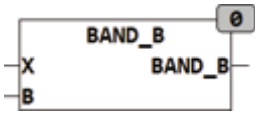
23.2. BAND_B

Type Funktion : BYTE

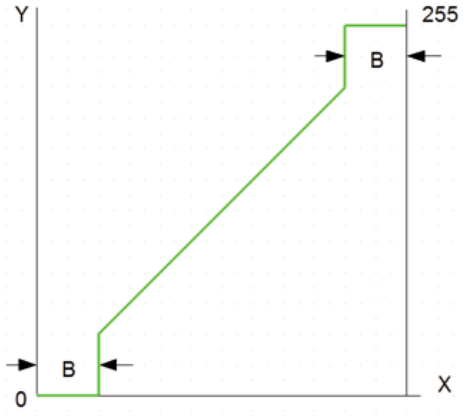
Input X : BYTE (Eingangswert)

B : BYTE (Begrenzungsbereich)

Output BYTE (Ausgangswert)

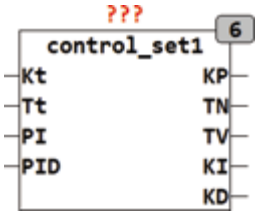


BAND_B blendet von Eingangsbereich 0..255 die Bereiche 0..B und 255-B..255 aus, in diesen Bereichen wird der Ausgang 0 beziehungsweise 255.



23.3. CONTROL_SET1

Type	Funktionsbaustein
Input	KT : REAL (Kritische Verstärkung)
	TT : REAL (Periodendauer der kritischen Schwingung)
	PI : BOOL (TRUE wenn Parameter für PI Regler bestimmt sind)
	PID : BOOL (TRUE wenn Parameter für PID Regler)
Setup	P_K : REAL := 0.5 (Vorgabewert KP für P Regler)
	PI_K : REAL := 0.45 (Vorgabewert KP für PI Regler
	PI_TN : REAL := 0.83 (Vorgabewert TN für PI Regler)
	PID_K : REAL := 0.6 (Vorgabewert KP für PID Regler)
	PID_TN : REAL := 0.5 (Vorgabewert TN für PID Regler)
	PID_TV : REAL := 0.125 (Vorgabewert TV für PID Regler)
Output	KP : REAL (Regelverstärkung KP)
	TN : REAL (Nachstellzeit des Integrators)
	TV : REAL (Vorhaltezeit des Differenzierers)
	KI : REAL (Verstärkungsfaktor des Integrators)
	KD : REAL (Verstärkungsfaktor des Differenzierers)



CONTROL_SET1 berechnet Einstellparameter für P, PI und PID Controller nach den Ziegler-Nichols Verfahren. Hierbei wird die kritische Verstärkung KT, und die Periodendauer der kritischen Schwingung TT angegeben. Die Parameter werden ermittelt indem der Regler als reiner P-Regler betrieben wird und die Verstärkung solange hochgefahren wird bis eine Dauerschwingung konstanter Amplitude einsetzt. Die entsprechenden Werte KT und TT werden dann ermittelt. Nachteil dieses Verfahrens ist das nicht jeder reale Regelkreis an die Stabilitätsgrenze gefahren werden kann, und das diese verfahren für langsame Regelkreise wie Raumregelungen sehr viel Zeit in Anspruch nimmt.

Reglertyp	PI	PID	KP	TN	TV
P-Regler	0	0	P_K * KT		

PI-Regler	1	0	PI_K * KT	PI_TN * TT	
PID-Regler	0	1	PID_K * KT	PID_TN * TT	PID_TV * TT

Die Vorgabewerte der Einstellregeln sind in Setup Variablen definiert und können vom Anwender verändert werden. Die folgende Tabelle zeigt die Default Werte

Reglertyp	PI	PID	KP	TN	TV
P-Regler	0	0	P_K = 0.5		
PI-Regler	1	0	PI_K = 0.45	PI_TN = 0.83	
PID-Regler	0	1	PID_K = 0.6	PID_TN = 0.5	PID_TV = 0.125

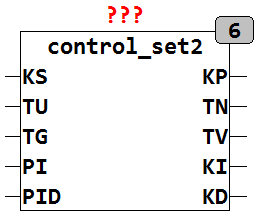
23.4. CONTROL_SET2

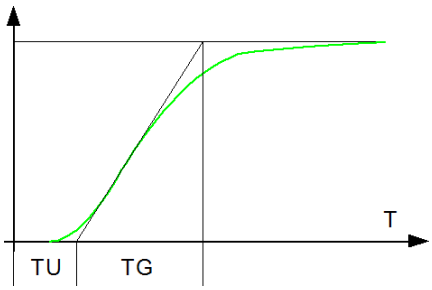
Type Funktionsbaustein

Input KT : REAL (Kritische Verstärkung)
TT : REAL (Periodendauer der kritischen Schwingung)
PI : BOOL (TRUE wenn Parameter für PI Regler bestimmt sind)
PID : BOOL (TRUE wenn Parameter für PID Regler)

Config P_K : REAL := 0.5 (Vorgabewert KP für P Regler)
PI_K : REAL := 0.45 (Vorgabewert KP für PI Regler
PI_TN : REAL := 0.83 (Vorgabewert TN für PI Regler)
PID_K : REAL := 0.6 (Vorgabewert KP für PID Regler)
PID_TN : REAL := 0.5 (Vorgabewert TN für PID Regler)
PID_TV : REAL := 0.125 (Vorgabewert TV für PID Regler)

Output KP : REAL (Regelverstärkung KP)
TN : REAL (Nachstellzeit des Integrators)
TV : REAL (Vorhaltezeit des Differenzierers)
KI : REAL (Verstärkungsfaktor des Integrators)
KD : REAL (Verstärkungsfaktor des Differenzierers)





CONTROL_SET2 berechnet Einstellparameter für P, PI und PID Controller nach den Ziegler-Nichols Verfahren. Hierbei wird die Verzugszeit TU und die Ausgleichszeit TG angegeben. Die Parameter werden ermittelt indem die Sprungantwort der Regelstrecke gemessen wird. TU entspricht der Zeit nach der der Ausgang der Regelstrecke 5% seines Maximalwertes erreicht hat. TG ist die Zeit die Zeit der Wendetangente der Regelstrecke. KS ist Istwertänderung der Regelstrecke / Stellgrößenveränderung.

Die folgende Grafik zeigt die Ermittlung von TU und TG mit dem Wendetangentenverfahren:

Reglertyp	PI	PID	KP	TN	TV
P-Regler	0	0	P_K * TG / TU / KS		
PI-Regler	1	0	PI_K * TG / TU / KS	PI_TN * TU	
PID-Regler	0	1	PID_K * TG / TU / KS	PID_TN * TU	PID_TV * TU

Die Vorgabewerte der Einstellregeln sind in Config Variablen definiert und können vom Anwender verändert werden. Die folgende Tabelle zeigt die Default Werte:

Reglertyp	PI	PID	KP	TN	TV
P-Regler	0	0	P_K = 1.0		
PI-Regler	1	0	PI_K = 0.9	PI_TN = 3.33	
PID-Regler	0	1	PID_K = 1.2	PID_TN = 2	PID_TV = 0.5

23.5. CTRL_IN

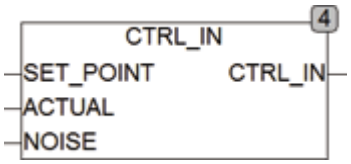
Type Funktion : REAL

Input SET_POINT : REAL (Vorgabewert)

ACTUAL : REAL (Prozesswert)

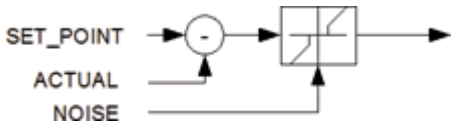
NOISE : REAL (Ansprechschwelle)

Output REAL (Prozessabweichung)



CTRL_IN berechnet die Prozessabweichung (SET_POINT _ ACTUAL) und gibt diese am Ausgang aus. Ist die Abweichung kleiner als der Wert am Eingang NOISE bleibt der Ausgang auf 0. CTRL_IN kann benutzt werden um eigene Regelbausteine Aufzubauen.

Blockschaltbild von CTRL_IN:



23.6. CTRL_OUT

Type Funktionsbaustein

Input CI : REAL (Eingang vom Controller)

OFFSET : REAL (Ausgangsoffset)

MAN_IN : REAL (Manueller Eingangswert)

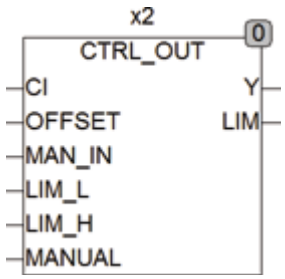
LIM_L : REAL (untere Ausgangsbegrenzung)

LIM_H : REAL (obere Ausgangsbegrenzung)

MANUAL : BOOL (Umschalter für Handbetrieb)

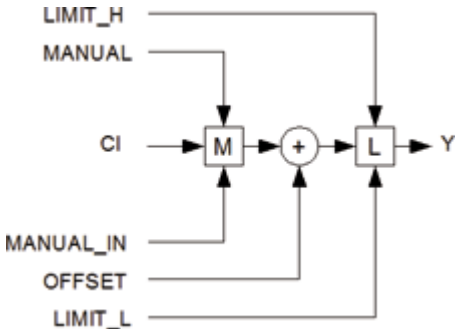
Output Y : REAL (Steuersignal)

LIM : BOOL (TRUE wenn Steuersignal ein Limit erreicht)



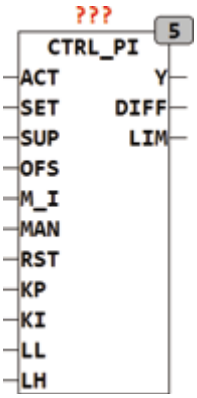
CTRL_OUT addiert zum Eingang CI den Wert von OFFSET und gibt das Ergebnis auf Y aus wenn MANUAL = FALSE. Wenn MANUAL = TRUE wird am Ausgang Y Der Eingangswert von MAN_IN + OFFSET ausgegeben. Y wird jederzeit auf die durch LIM_L und LIM_H definierten Grenzen begrenzt. Erreicht Y eine der Grenzen, so wird der Ausgang LIM TRUE. CTRL_OUT kann benutzt werden um eigene Regelbausteine Aufzubauen.

Blockschaltbild von CTRL_OUT:



23.7. CTRL_PI

Type	Funktionsbaustein
Input	ACT : REAL (gemessener Wert nach der Strecke)
	SET : REAL (Vorgabewert)
	SUP : REAL (Rauschunterdrückung)
	OFS : REAL (Offset für den Ausgang)
	M_I : REAL (Eingangswert für manuellen Betrieb)
	MAN : BOOL (Umschalten auf Handbetrieb, MANUAL = TRUE)
	RST : BOOL (Asynchroner Reset-Eingang)
	KP : REAL (Proportionaler Anteil des Reglers)
	KI : REAL (Integraler Anteil des Reglers)
	LL : REAL (untere Ausgangsbegrenzung)
	LH : REAL (obere Ausgangsbegrenzung)
Output	Y : REAL (Ausgang des Reglers)
	DIFF : Real (Regelabweichung)
	LIM : BOOL (TRUE, wenn der Ausgang ein Limit erreicht hat)



CTRL_PI ist ein PI-Regler mit dynamischen Anti Wind-Up und manuellem Steuereingang. Der PI Regler arbeitet nach der Formel:

Y = KP * DIFF + KI * INTEG(DIFF) + OFFSET

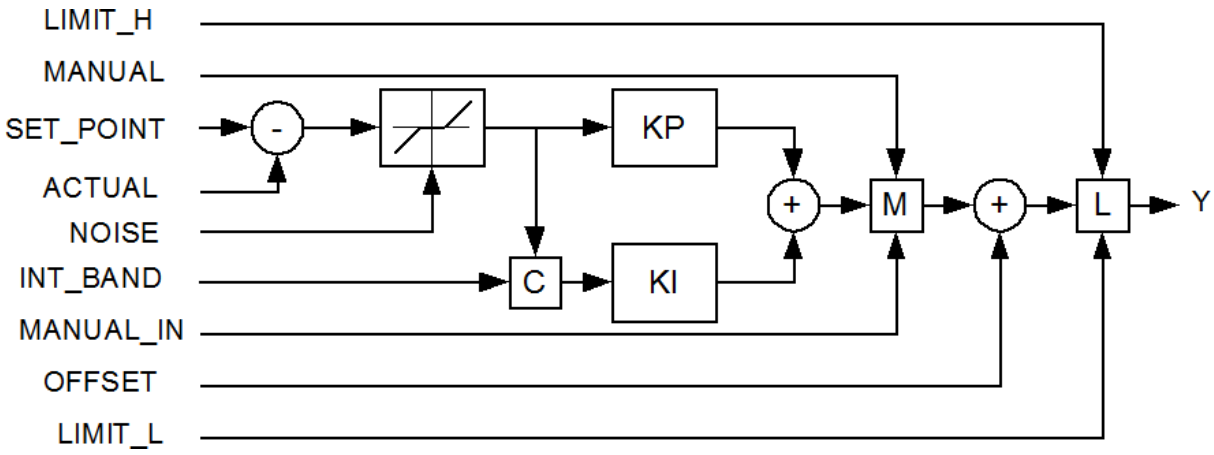
wobei DIFF = SET_POINT – ACTUAL

Im Handbetrieb (Manual = TRUE) gilt: Y = MANUAL_IN + OFFSET

ACT ist der gemessene Wert nach der Regelstrecke und Set ist die Sollwertvorgabe für den Regler. Die Eingangswerte LH und LL begrenzen den Ausgangswert Y. Mit RST kann der interne Integrator jederzeit auf 0 gesetzt werden. Der Ausgang LIM signalisiert das der Regler an eine der Grenzen LL oder LH gelaufen ist. Der PI-Regler arbeitet frei laufend und benutzt zur Berechnung des Integrators die Trapezregel für höchste Genauigkeit und optimale Geschwindigkeit. Die Default-Werte der Eingangsparameter sind wie folgt vordefiniert: KP = 1, KI = 1, LIMIT_L = -1000 und LIMIT_H = +1000. Mit dem Eingang SUP wird eine Rauschunterdrückung eingestellt, der Wert am Eingang SUP legt fest ab welcher Regeldifferenz der Regler einschaltet. Mit SUP wird vermieden das der Ausgang des Reglers dauern schwankt. Der Wert am Eingang SUP sollte so bemessen sein das er das Rauschen der Regelstrecke und der Sensoren unterdrückt. Wird zum Beispiel der Eingang SUP auf 0.1 gesetzt so wird der Regler erst bei Regelabweichungen größer als 0.1 aktiv. Der Ausgang DIFF stellt die gemessene und durch ein Noise Filter (DEAD_BAND) gefilterte Regelabweichung zur Verfügung. DIFF wird in einer Regelstrecke normalerweise nicht benötigt, kann aber zur Beeinflussung der Regelparameter benutzt werden. Der Eingang OFS wird als letzter Wert zum Ausgang addiert, und dient vor allem zum kompensieren von Störsignalen, deren Wirkung auf den Regelkreis abgeschätzt werden kann.

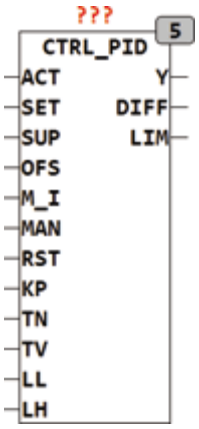
Der Regler arbeitet mit einem Dynamischen Wind-Up das verhindert dass der Integrator bei erreichen eines Ausgangslimits und weiterer Regelabweichung unbegrenzt weiter läuft und die Regeleigenschaften negativ beeinflusst. In der Einleitung des Kapitel Regelungstechnik finden sich weitere Details zum Thema Anti Wind-Up.

Die folgende Grafik verdeutlicht die interne Struktur des Reglers:



23.8. CTRL_PID

Type	Funktionsbaustein
Input	ACT : REAL (gemessener Wert nach der Strecke)
	SET : REAL (Vorgabewert)
	SUP : REAL (Rauschunterdrückung)
	OFS : REAL (Offset für den Ausgang)
	M_I : REAL (Eingangswert für manuellen Betrieb)
	MAN : BOOL (Umschalten auf Handbetrieb, MANUAL = TRUE)
	RST : BOOL (Asynchroner Reset-Eingang)
	KP : REAL (Verstärkung des Reglers)
	TN : REAL (Nachstellzeit des Reglers)
	TV : REAL (Vorhaltezeit des Reglers)
	LL : REAL (untere Ausgangsbegrenzung)
	LH : REAL (obere Ausgangsbegrenzung)
Output	Y : REAL (Ausgang des Reglers)
	DIFF : Real (Regelabweichung)
	LIM : BOOL (TRUE, wenn der Ausgang ein Limit erreicht hat)



CTRL_PID ist ein PID-Regler mit dynamischen Anti Wind-Up und manuellem Steuereingang. Der PID Regler arbeitet nach der Formel:

Y = KP * (DIFF + 1/TN * INTEG(DIFF) + TV *DERIV(DIFF)) + OFFSET

wobei DIFF = SET_POINT – ACTUAL

Im Handbetrieb (Manual = TRUE) gilt: Y = MANUAL_IN + OFFSET

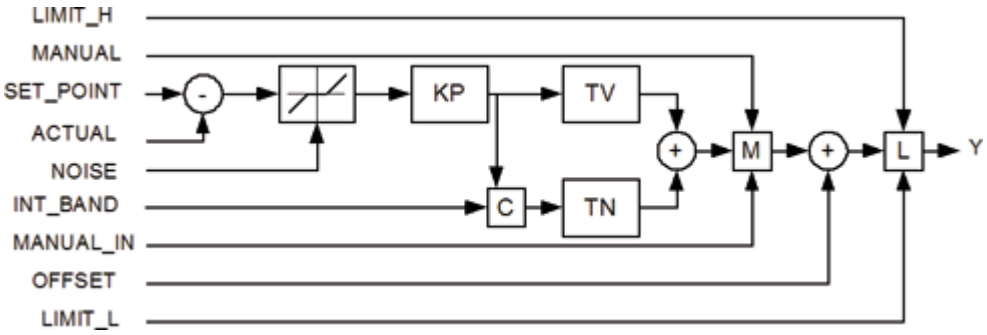
ACT ist der gemessene Wert nach der Regelstrecke und SET ist die Sollwertvorgabe für den Regler. Die Eingangswerte LH und LL begrenzen den Ausgangswert Y. Mit RST kann der interne Integrator jederzeit auf 0 gesetzt werden. Der Ausgang LIM signalisiert das der Regler an eine der Grenzen LL oder LH gelaufen ist. Der PID-Regler arbeitet frei laufend und benutzt zur Berechnung die Trapezregel für höchste Genauigkeit und optimale Geschwindigkeit. Die Default-Werte der Eingangsparameter sind wie folgt vordefiniert: KP = 1, TN = 1, TV = 1, LIMIT_L = -1000 und LIMIT_H = +1000. Mit dem Eingang SUP wird eine Rauschunterdrückung eingestellt, der Wert am Eingang SUP legt fest ab welcher Regeldifferenz der Regler einschaltet. Mit SUP wird vermieden das der Ausgang des Reglers dauern schwankt. Der Wert am Eingang SUP sollte so bemessen sein das er das Rauschen der Regelstrecke und der Sensoren unterdrückt. Wird zum Beispiel der Eingang SUP auf 0.1 gesetzt so wird der Regler erst bei Regelabweichungen größer als 0.1 aktiv. Der Ausgang DIFF stellt die gemessene und durch ein Noise Filter (DEAD_BAND) gefilterte Regelabweichung zur Verfügung. DIFF wird in einer Regelstrecke normalerweise nicht benötigt, kann aber zur Beeinflussung der Regelparameter benutzt werden. Der Eingang OFS wird als letzter Wert zum Ausgang addiert, und dient vor allem zum kompensieren von Störsignalen, deren Wirkung auf den Regelkreis abgeschätzt werden kann.

Der Regler arbeitet mit einem Dynamischen Wind-Up das verhindert dass der Integrator bei erreichen eines Ausgangslimits und weiterer Regelabweichung unbegrenzt weiter läuft und die Regeleigenschaften negativ beeinflusst. In der Einleitung des Kapitel Regelungstechnik finden sich weitere Details zum Thema Anti Wind-Up.

Die Regelparameter werden in der Form KP, TN und TV angegeben, falls die Parameter als KP, KI und KD vorliegen können sie entsprechend der folgenden Formel umgerechnet werden:

TN = KP/KI und TV = KD/KP

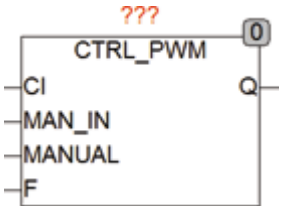
Die folgende Grafik zeigt die interne Struktur des PID Reglers:



Im folgenden Beispiel wir ein PID Regler gezeigt, dessen SET_POINT durch den Baustein TUNE2 mittels Taster erzeugt wird. Der Ausgang DIFF wird an einen Baustein PARSET2 geleitet welcher abhängig von der Regelabweichung am Ausgang DIFF die Parameter KP, TN und TV verändert.

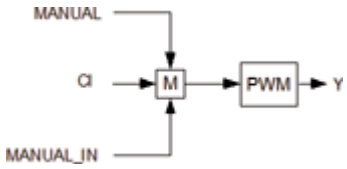
23.9. CTRL_PWM

Type	Funktionsbaustein
Input	CI : REAL (Eingang vom Controller)
	MAN_IN : REAL (Manueller Eingangswert)
	MANUAL : BOOL (Umschalter für Handbetrieb)
	F : REAL (Frequenz der Ausgangsimpulse in Hz)
Output	Q : BOOL (Steuersignal)

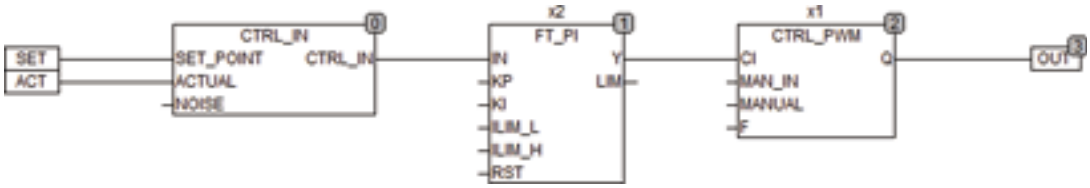


CTRL_PWM wandelt den Eingangswert CI (0..1) in ein Pulsweitenmoduliertes Ausgangssignal Q. Wenn MANUAL = TRUE wird am Ausgang Q Der Eingangswert von MAN_IN ausgegeben. CTRL_OUT kann benutzt werden um eigene Regelbausteine Aufzubauen.

Blockschaltbild von CTRL_PWM:

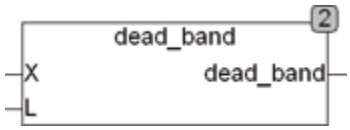


DAS folgende Beispiel zeigt einen PI Regler mit PWM Ausgang:



23.10. DEAD_BAND

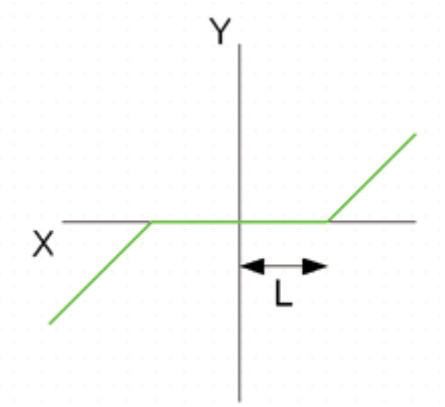
Type	Funktion
Input	X : REAL (Eingangswert)
	L : REAL (Lockout Wert)
Output	REAL (Ausgangswert)



DEAD_BAND ist eine lineare Übertragungsfunktion mit Totzone. Die Funktion verschiebt den positiven Teil der Kurve um -L und den negativen Teil der Kurve um +L. DEAD_BAND wird benutzt um Quantisierungsrauschen und andere Rauschanteile aus einem Signal zu filtern. DEAD_BAND wird zum Beispiel in Regelkreisen eingesetzt um zu verhindern das der Regler dauernd in kleinen Schritten schaltet und dabei das Stellglied übermäßig belastet und abnutzt.

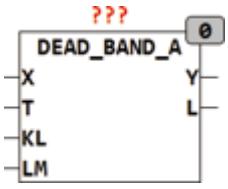
DEAD_BAND = X - SGN(X)*L wenn ABS(X) > L wenn ABS(X) > L

DEAD_BAND = 0 wenn ABS(X) <= L



23.11. DEAD_BAND_A

Type	Funktionsbaustein
Input	X : REAL (Eingangswert)
	T : TIME (Verzögerungszeit des Tiefpasses)
	KL : REAL (Verstärkung des Filters)
	LM : REAL (Maximalwert der HF Amplitude)
Output	Y : REAL (Ausgangswert)
	L : REAL (Amplitude der Hochfrequenz)



DEAD_BAND_A ist eine selbst adaptierende lineare Übertragungsfunktion mit Totzone. Die Funktion verschiebt den positiven Teil der Kurve um -L und den negativen Teil der Kurve um +L. DEAD_BAND_A wird benutzt um Rauschanteile um den Nullpunkt aus einem Signal zu filtern. DEAD_BAND_A wird zum Beispiel in Regelkreisen eingesetzt um zu verhindern das der Regler dauernd in kleinen Schritten schaltet und dabei das Stellglied übermäßig belastet und abnutzt.

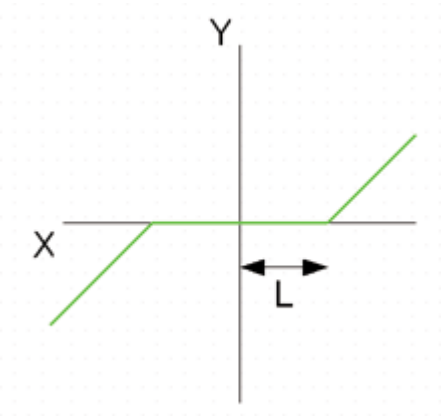
Die Größe L wird berechnet indem aus dem Eingangssignal X die HF Anteile über einen Tiefpaß mit der Zeitkonstante T gefiltert werden und aus der Amplitude des HF Anteils wird die Totzone L berechnet. Die Empfindlichkeit des Bausteins kann über den Parameter KL verändert werden. KL ist mit 1 vordefiniert und kann deshalb unbeschaltet beliben. Sinnvolle Werte für KL liegen zwischen 1 - 5.

L = HF_Amplitude (effektiv) * KL.

Damit der Baustein auch bei extremen Betriebsbedingungen stabil bleibt wird über den Eingang LM der Maximalwert von L begrenzt.

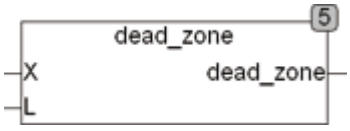
DEAD_BAND = X - SGN(X)*L wenn ABS(X) > L wenn ABS(X) > L

DEAD_BAND = 0 wenn ABS(X) <= L



23.12. DEAD_ZONE

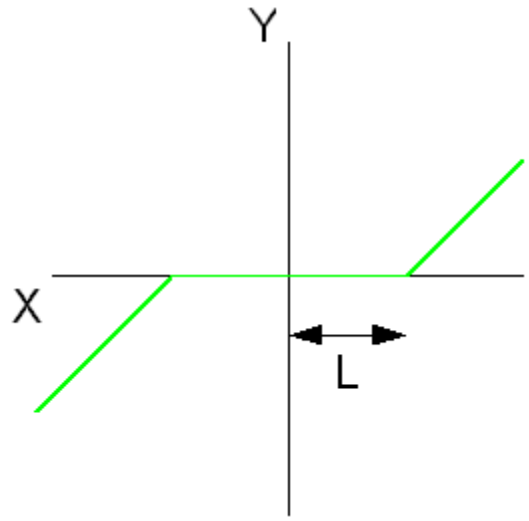
Type	Funktion : REAL
Input	X : REAL (Eingangswert)
	L : REAL (Lockout Wert)
Output	REAL (Ausgangswert)



DEAD_ZONE ist eine lineare Übertragungsfunktion mit Totzone. Der Ausgang entspricht dem Eingangssignal, wenn der Absolutwert des Eingangs größer als L ist.

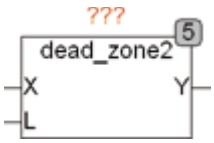
DEAD_ZONE = X wenn ABS(X) > L

DEAD_ZONE = 0 wenn ABS(X) <= L



23.13. DEAD_ZONE2

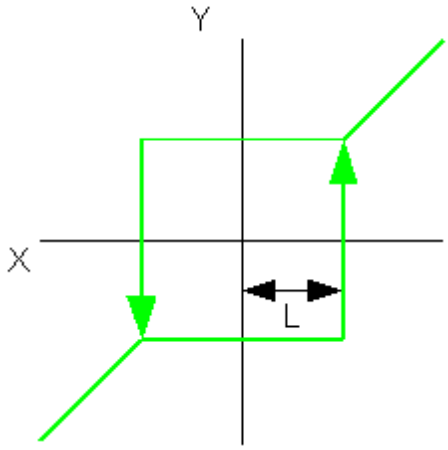
Type	Funktionsbaustein
Input	X : REAL (Eingangswert)
	L : REAL (Lockout Wert)
Output	Y : REAL (Ausgangswert)



DEAD_ZONE2 ist eine lineare Übertragungsfunktion mit Totzone und Hysterese. Der Ausgang entspricht dem Eingangssignal, wenn der Absolutwert des Eingangs größer als L ist.

DEAD_ZONE2 = X wenn ABS(X) > L

DEAD_ZONE2 = +/- L wenn ABS(X) <= L



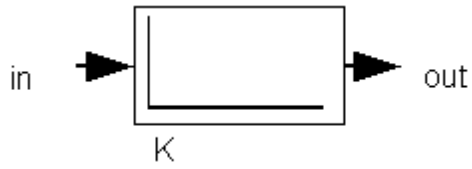
23.14. FT_DERIV

Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	K : REAL (Multiplikator)
	RUN : BOOL (Freigabe Eingang)
Output	OUT : REAL (Ableitung des Eingangssignals K * ,X / ,T)

FT_DERIV ist ein D-Glied, oder auch LZI-Übertragungsglied, welches ein differenzierendes Übertragungsverhalten aufweist. Am Ausgang von FT_DERIV steht die Ableitung über die Zeit T in Sekunden zur Verfügung. Wenn das Eingangssignal in einer Sekunde von 3 auf 4 steigt so ist der Ausgang 1*K (K * ,X / ,T = 1 * (4 – 3) / 1 = 1.

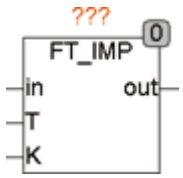
Anders ausgedrückt ist die Ableitung des Eingangssignals die momentane Steigung des Eingangssignals. Mit dem Eingang RUN kann FT_DERIV enabled, beziehungsweise disabled werden. FT_DERIV arbeitet intern in Mikrosekunden und wird dadurch auch den Anforderungen sehr schneller SPS Controller mit Zykluszeiten unter einer Millisekunde gerecht.

Strukturbild:



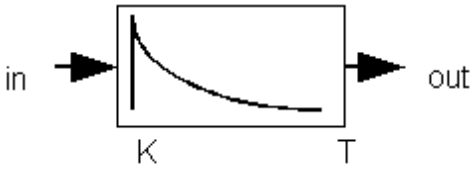
23.15. FT_IMP

Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	T : TIME (Zeitkonstante)
	K : REAL (Multiplikator)
Output	OUT : REAL (Hochpass Mit Zeitkonstante T)



FT_IMP Ist ein Hochpass-Filter mit Zeitkonstante T und Multiplikator K. Eine sprunghafte Änderung am Eingang wird am Ausgang sichtbar, kling aber nach der Zeit T bereits um 63% und nach 3 * T um 95% ab. Somit ist nach einer sprunghaften Änderung des Eingangssignals von 0 auf 10 der Ausgang zum Zeitpunkt der Eingangsänderung 10 und kling nach 1 * T auf 3,7 und nach 3 * T auf 0,5 ab und wird dann allmählich wieder 0.

Strukturbild:



23.16. FT_INT

Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	K : REAL (Multiplikator)
	RUN : BOOL (Freigabe Eingang)

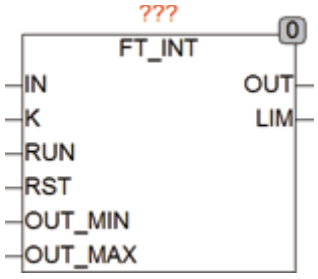
RST : BOOL (Reset Eingang)

OUT_MIN : REAL (unteres Ausgangs Limit)

OUT_MAX : REAL (oberes Ausgangs Limit)

Output OUT : REAL (Ausgangssignal)

LIM : BOOL (TRUE wenn der Ausgang an einem Limit steht)

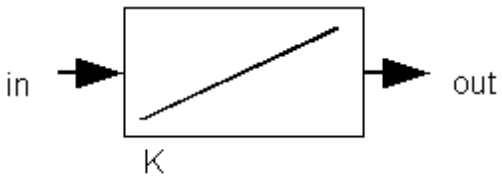


FT_INT ist ein Integratorbaustein der das Integral über das Eingangssignal am Ausgang bereitstellt. Der Eingang K ist ein Multiplikator für das Ausgangssignal. Run schaltet den Integrator ein wenn TRUE und Aus wenn FALSE. RST (Reset) setzt den Ausgang auf 0. Die Eingänge OUT_MIN und OUT_MAX dienen dazu obere und untere Grenzwerte für den Ausgang des Integrators festzulegen . FT_INT arbeitet intern in Mikrosekunden und wird dadurch auch den Anforderungen sehr Schneller SPS Controller mit Zykluszeiten unter einer Millisekunde gerecht.

Ein Grundsatzproblem bei Integratoren ist die Auflösung, So hat der Ausgang vom Typ Real eine Auflösung von 7-8 Stellen was zur Folge hat das bei einem errechneten Integrationsschritt von 1 bei einem Ausgangswert von größer als hundert Millionen (1E8) dieser Schritt nicht mehr aufaddiert werden kann da er unter die Auflösungsgrenze von maximal 8 Stellen beim Typ Real fällt. Diese Limitation ist bei Einsatz von FT_INT zu beachten.

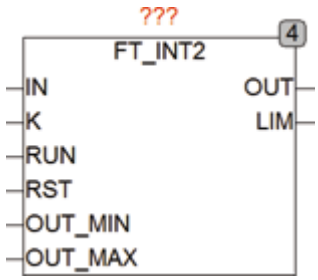
Beispiel: ein Eingangssignal von 0,0001 bei einer Abtastzeit von 1 Millisekunde und einem Ausgangswert von 100000 würde einen Wert von 0,0001 * 0,001 Sekunden = 0,000001 zum Ausgangswert von 100000 addieren, was unweigerlich wieder den Wert von 100000 ergibt, denn die Auflösung des Datentyps Real kann nur maximal 8 Stellen erfassen. Dies ist vor allem zu beachten wenn FT_INT als Verbrauchszähler oder ähnlichen Anwendungen zum Einsatz kommen soll.

Strukturbild:



23.17. FT_INT2

Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	K : REAL (Multiplikator)
	RUN : BOOL (Freigabe Eingang)
	RST : BOOL (Reset Eingang)
	OUT_MIN : REAL (unteres Ausgangs Limit)
	OUT_MAX : REAL (oberes Ausgangs Limit)
Output	OUT : REAL (Ausgangssignal)
	LIM : BOOL (TRUE wenn der Ausgang an einem Limit steht)



FT_INT2 ist ein Integratorbaustein der intern mit doppelter Genauigkeit rechnet und eine Auflösung von 14 Dezimalstellen sicherstellt. Dies macht FT_INT2 im Gegensatz zu FT_INT für z.B. Verbrauchszähler und ähnliche Anwendungen geeignet.

Beispiel: ein Eingangssignal von 0,0001 bei einer Abtastzeit von 1 Millisekunde und einem Ausgangswert von 100000 ergibt einen Wert von 0,0001 * 0,001 Sekunden = 0,000001 der zum Ausgangswert von 100000 addiert werden soll, was unweigerlich wieder den Wert von 100000 ergibt, denn die Auflösung des Datentyps Real kann nur maximal 8 Stellen erfassen. FT_INT2 löst dieses Problem indem er intern mit doppelter Genauigkeit (14

Dezimalstellen) rechnet und addiert auch kleinste Eingangswerte auf so dass keine Informationen verloren gehen.

23.18. FT_PD

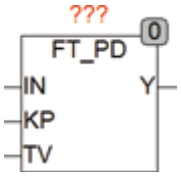
Type Funktionsbaustein

Input IN : REAL (Eingangssignal)

KP : REAL (Proportionaler Anteil des Reglers)

TV : REAL (Nachstellzeit des Differenzierers in Sekunden)

Output Y : REAL (Ausgang des Reglers)

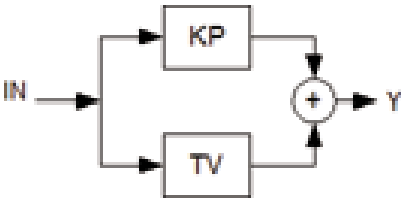


FT_PD ist ein PD-Regler der nach folgender Formel arbeitet:

Y = KP * (IN + DERIV(IN))

FT_PD kann zusammen mit den Bausteinen CTRL_IN und CTRL_OUT zum Aufbau eines PD Reglers benutzt werden.

Die folgende Grafik verdeutlicht die interne Struktur des Reglers:



23.19. FT_PDT1

Type Funktionsbaustein

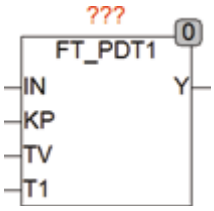
Input IN : REAL (Eingangssignal)

KP : REAL (Proportionaler Anteil des Reglers)

TV : REAL (Nachstellzeit des Differenzierers in Sekunden)

T1 : REAL (T1 des PT1 Gliedes in Sekunden)

Output Y : REAL (Ausgang des Reglers)

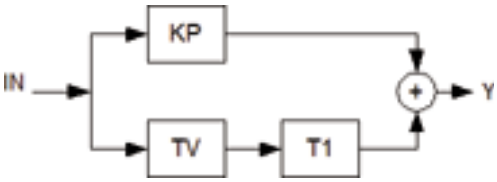


FT_PDT1 ist ein PD-Regler mit T1 Glied im D-Anteil. Der Baustein arbeitet nach folgender Formel:

Y = KP * (IN + PT1(DERIV(IN))

FT_PDT1 kann zusammen mit den Bausteinen CTRL_IN und CTRL_OUT sowie weiteren Regelungstechnischen Bausteinen zum Aufbau komplexer Reglerschaltungen benutzt werden.

Interner Aufbau des Bausteins:



23.20. FT_PI

Type Funktionsbaustein

Input IN : REAL (Eingangssignal)

KP : REAL (Proportionaler Anteil des Reglers)

KI : REAL (Integraler Anteil des Reglers)

ILIM_L : REAL (untere Ausgangsbegrenzung des Integrators)

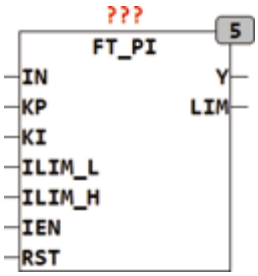
ILIM_H : REAL (obere Ausgangsbegrenzung des Integrators)

IEN : BOOL (Enable für den Integrator)

RST : BOOL (Asynchroner Reset-Eingang)

Output Y : REAL (Ausgang des Reglers)

LIM : BOOL (TRUE, wenn der Ausgang ein Limit erreicht hat)



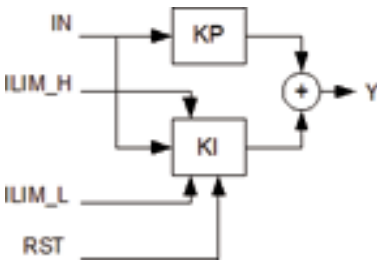
FT_PI ist ein PI-Regler der nach folgender Formel arbeitet:

Y = KP * IN + KI * INTEG(IN)

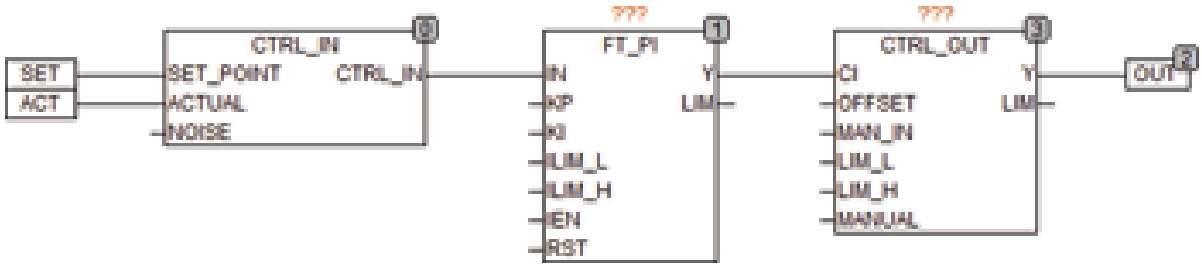
Die Eingangswerte ILIM_H und ILIM_L begrenzen den Arbeitsbereich des internen Integrators. Mit RST kann der interne Integrator jederzeit auf 0 gesetzt werden. Der Ausgang LIM signalisiert das der Integrator an eine der Grenzen ILIM_L oder ILIM_H gelaufen ist. Der PI-Regler arbeitet frei laufend und benutzt zur Berechnung des Integrators die Trapezregel für höchste Genauigkeit und optimale Geschwindigkeit. Die Default-Werte der Eingangsparameter sind wie folgt vordefiniert: KP = 1, KI = 1, ILIM_L = -1E38 und ILIM_H = +1E38.

Anti Wind-Up: Regelbausteine mit Interalanteil neigen zu dem so genannten Wind Up Effekt. Ein Wind-Up bedeutet das der Integratorbaustein kontinuierlich weiter läuft weil z.B. das Stellsignal Y an einem Anschlag steht und die Regelung über längere Zeit nicht in der Lage ist die Regelabweichung auszugleichen, was dann nach anschließendem Übergang in den Regelbereich erst zu einem langen und Zeitaufwendigen Abbau des Integratorwertes führt und die Regelung nur verzögert reagiert. Da der Integralanteil nur für den Ausgleich der Regelabweichung nach allen anderen Regelanteilen nötig ist, kann und sollte der Bereich des Integrators mit den Werten ILIM begrenzt werden. Der Integrator wird dann bei Erreichen eines Limits gestoppt und verharrt auf dem letzten gültigen Wert. Für andere Wind-Up Maßnahmen kann der Integrator jederzeit mit dem Eingang IEN = FALSE separat gesteuert werden, der Integrator läuft nur wenn IEN = TRUE.

Die folgende Grafik verdeutlicht die interne Struktur des Reglers:

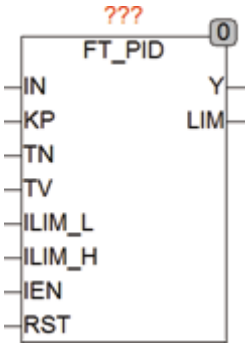


FT_PI kann zusammen mit den Bausteinen CTRL_IN und CTRL_OUT zum Aufbau eines PI Reglers benutzt werden.



23.21. FT_PID

Type	Funktionsbaustein
Input	IN : REAL (Eingangswert)
	KP : REAL (Verstärkung des Reglers)
	TN : REAL (Nachstellzeit des Reglers in Sekunden)
	TV : REAL (Vorhaltezeit des Reglers in Sekunden)
	ILIM_L : REAL (untere Ausgangsbegrenzung des Integrators)
	ILIM_H : REAL (obere Ausgangsbegrenzung des Integrators)
	IEN : BOOL (Enable für den Integrator)
	RST : BOOL (Asynchroner Reset-Eingang)
Output	Y : REAL (Ausgang des Reglers)
	LIM : BOOL (TRUE, wenn der Ausgang ein Limit erreicht hat)



FT_PID ist ein PID-Regler der nach folgender Formel arbeitet:

Y = KP * (IN + 1/TN * INTEG(IN) + TV *DERIV(IN))

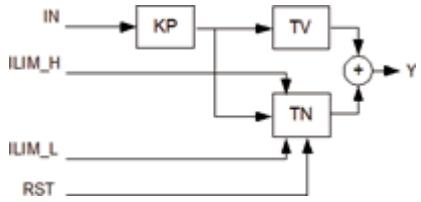
Die Regelparameter werden in der Form KP, TN und TV angegeben, falls die Parameter als KP, KI und KD vorliegen können sie entsprechend der folgenden Formel umgerechnet werden:

TN = KP/KI und TV = KD/KP

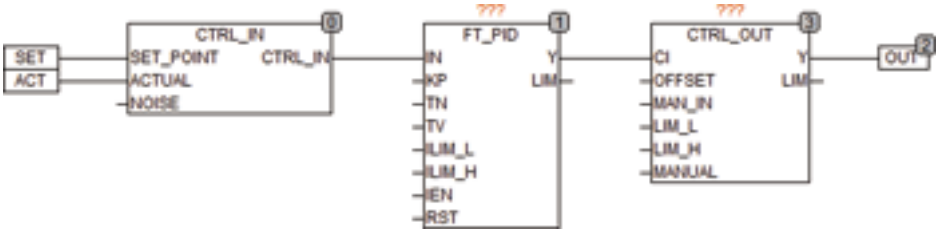
Die Eingangswerte ILIM_H und ILIM_L begrenzen den Arbeitsbereich des internen Integrators. Mit RST kann der interne Integrator jederzeit auf 0 gesetzt werden. Der Ausgang LIM signalisiert das der Integrator an eine der Grenzen ILIM_L oder ILIM_H gelaufen ist. Der PI-Regler arbeitet frei laufend und benutzt zur Berechnung des Integrators die Trapezregel für höchste Genauigkeit und optimale Geschwindigkeit. Die Default-Werte der Eingangsparameter sind wie folgt vordefiniert: KP = 1, TN = 1s, TV = 1s, ILIM_L = -1E38 und ILIM_H = +1E38.

Anti Wind-Up: Regelbausteine mit Interalanteil neigen zu dem so genannten Wind Up Effekt. Ein Wind-Up bedeutet das der Integratorbaustein kontinuierlich weiter läuft weil z.B. das Stellsignal Y an einem Anschlag steht und die Regelung über längere Zeit nicht in der Lage ist die Regelabweichung auszugleichen, was dann nach anschließendem Übergang in den Regelbereich erst zu einem langen und Zeitaufwendigen Abbau des Integratorwertes führt und die Regelung nur verzögert reagiert. Da der Integralanteil nur für den Ausgleich der Regelabweichung nach allen anderen Regelanteilen nötig ist, kann und sollte der Bereich des Integrators mit den Werten ILIM begrenzt werden. Der Integrator wird dann bei Erreichen eines Limits gestoppt und verharrt auf dem letzten gültigen Wert. Für andere Wind-Up Maßnahmen kann der Integrator jederzeit mit dem Eingang IEN = FALSE separat gesteuert werden, der Integrator läuft nur wenn IEN = TRUE.

Die folgende Grafik zeigt die interne Struktur des PID Reglers:

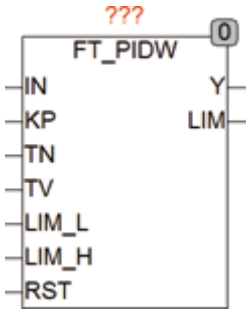


FT_PID kann zusammen mit den Bausteinen CTRL_IN und CTRL_OUT zum Aufbau eines PID Reglers benutzt werden.



23.22. FT_PIDW

Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	KP : REAL (Proportionaler Anteil des Reglers)
	TN : REAL (Nachstellzeit des Reglers in Sekunden)
	TV : REAL (Vorhaltezeit des Reglers in Sekunden)
	LIM_L : REAL (untere Ausgangsbegrenzung des Integrators)
	LIM_H : REAL (obere Ausgangsbegrenzung des Integrators)
	RST : BOOL (Asynchroner Reset-Eingang)
Output	Y : REAL (Ausgang des Reglers)
	LIM : BOOL (TRUE, wenn der Ausgang ein Limit erreicht hat)



FT_PIDW ist ein PID-Regler mit Anti Wind-Up Hold der nach folgender Formel arbeitet:

Y = KP * (IN + 1/TN * INTEG(IN) + TV *DERIV(IN))

Die Regelparameter werden in der Form KP, TN und TV angegeben, falls die Parameter als KP, KI und KD vorliegen können sie entsprechend der folgenden Formel umgerechnet werden:

TN = KP/KI und TV = KD/KP

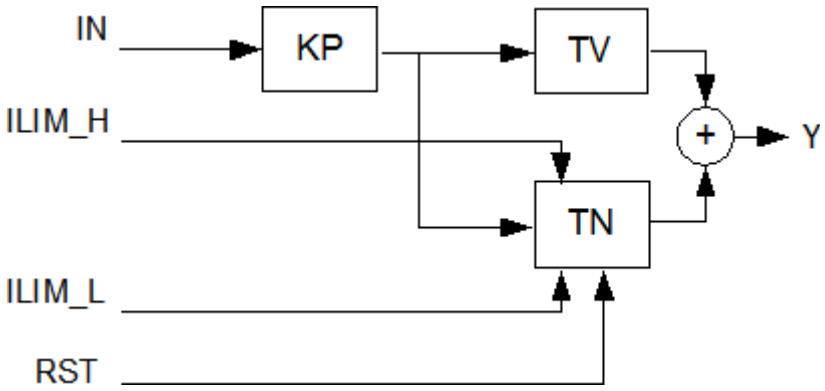
Die Eingangswerte LIM_H und LIM_L begrenzen den Wertebereich des Ausgangs Y. Mit RST kann der interne Integrator jederzeit auf 0 gesetzt werden. Der Ausgang LIM signalisiert das der Ausgang Y an eine der Grenzen LIM_L oder LIM_H gelaufen ist. Der Regler arbeitet frei laufend und benutzt zur Berechnung des Integrators die Trapezregel für höchste Genauigkeit und optimale Geschwindigkeit. Die Default-Werte der Eingangsparameter sind wie folgt vordefiniert: KP = 1, TN = 1s, TV = 1s, ILIM_L = -1E38 und ILIM_H = +1E38.

Anti Wind-Up: Regelbausteine mit Interalanteil neigen zu dem so genannten Wind-Up Effekt. Ein Wind-Up bedeutet das der Integratorbaustein kontinuierlich weiter läuft weil z.B. das Stellsignal Y an einem Anschlag steht und die Regelung über längere Zeit nicht in der Lage ist die Regelabwei-

chung auszugleichen, was dann nach anschließendem Übergang in den Regelbereich erst zu einem langen und Zeitaufwendigen Abbau des Integratorwertes führt und die Regelung nur verzögert reagiert. Da der Integralanteil nur für den Ausgleich der Regelabweichung nach allen anderen Regelanteilen nötig ist, kann und sollte der Bereich des Integrators mit den Werten ILIM begrenzt werden.

Der Baustein FT_PIDW hat einen so genannten Wind-Up Hold der bei erreichen einer Ausgangsbegrenzung (LIM_L, LIM_H) den Integrator auf dem letzten Wert einfriert und so einen Wind-Up verhindert.

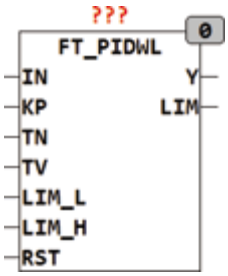
Die folgende Grafik zeigt die interne Struktur des PIDW Reglers:



FT_PIDW kann zusammen mit den Bausteinen CTRL_IN und CTRL_OUT zum Aufbau eines PID Reglers benutzt werden.

23.23. FT_PIDWL

Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	KP : REAL (Proportionaler Anteil des Reglers)
	TN : REAL (Nachstellzeit des Reglers in Sekunden)
	TV : REAL (Vorhaltezeit des Reglers in Sekunden)
	LIM_L : REAL (untere Ausgangsbegrenzung des Integrators)
	LIM_H : REAL (obere Ausgangsbegrenzung des Integrators)
	RST : BOOL (Asynchroner Reset-Eingang)
Output	Y : REAL (Ausgang des Reglers)
	LIM : BOOL (TRUE, wenn der Ausgang ein Limit erreicht hat)



FT_PIDWL ist ein PID-Regler mit dynamischen Wind-Up Reset der nach folgender Formel arbeitet:

Y = KP * (IN + 1/TN * INTEG(IN) + TV *DERIV(IN))

Die Regelparameter werden in der Form KP, TN und TV angegeben, falls die Parameter als KP, KI und KD vorliegen können sie entsprechend der folgenden Formel umgerechnet werden:

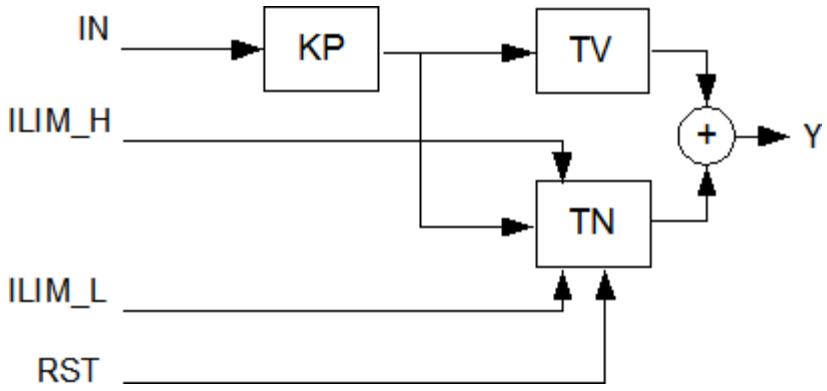
TN = KP/KI und TV = KD/KP

Die Eingangswerte LIM_H und LIM_L begrenzen den Wertebereich des Ausgangs Y. Mit RST kann der interne Integrator jederzeit auf 0 gesetzt werden. Der Ausgang LIM signalisiert das der Ausgang Y an eine der Grenzen LIM_L oder LIM_H gelaufen ist. Der Regler arbeitet frei laufend und benutzt zur Berechnung des Integrators die Trapezregel für höchste Genauigkeit und optimale Geschwindigkeit. Die Default-Werte der Eingangsparameter sind wie folgt vordefiniert: KP = 1, TN = 1s, TV = 1s, ILIM_L = -1E38 und ILIM_H = +1E38.

Anti Wind-Up: Regelbausteine mit Interalanteil neigen zu dem so genannten Wind-Up Effekt. Ein Wind-Up bedeutet das der Integratorbaustein kontinuierlich weiter läuft weil z.B. das Stellsignal Y an einem Anschlag steht und die Regelung über längere Zeit nicht in der Lage ist die Regelabweichung auszugleichen, was dann nach anschließendem Übergang in den Regelbereich erst zu einem langen und Zeitaufwendigen Abbau des Integratorwertes führt und die Regelung nur verzögert reagiert. Da der Integralanteil nur für den Ausgleich der Regelabweichung nach allen anderen Regelanteilen nötig ist, kann und sollte der Bereich des Integrators mit den Werten ILIM begrenzt werden.

Der Baustein FT_PIW hat einen so genannten dynamischen Wind-Up Reset der bei erreichen einer Ausgangsbegrenzung (LIM_L, LIM_H) den Integrator auf einen Wert zurücksetzt der dem Ausgangslimit entspricht. Wenn nach erreichen einen Limits der Regler wieder in den Arbeitsbereich eintritt muss der Integrator nicht erst Auf- oder Ab- integriert werden, und der Regler ist ohne Verzögerung sofort Einsatzbereit. Die dynamische Anti Wind-Up Methode ist die in den meisten Fällen ohne Nachteile vorzuziehende Methode, da Sie den Regler nicht negativ beeinflusst und die Nachteile des Wind_Up verhindert.

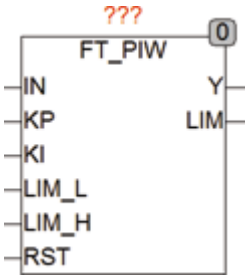
Die folgende Grafik zeigt die interne Struktur des PIDW Reglers:



FT_PIDWL kann zusammen mit den Bausteinen CTRL_IN und CTRL_OUT zum Aufbau eines PID Reglers benutzt werden.

23.24. FT_PIW

Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	KP : REAL (Proportionaler Anteil des Reglers)
	KI : REAL (Integraler Anteil des Reglers)
	LIM_L : REAL (untere Ausgangsbegrenzung des Integrators)
	LIM_H : REAL (obere Ausgangsbegrenzung des Integrators)
	RST : BOOL (Asynchroner Reset-Eingang)
Output	Y : REAL (Ausgang des Reglers)
	LIM : BOOL (TRUE, wenn der Ausgang ein Limit erreicht hat)



FT_PIW ist ein PI-Regler mit Anti Wind-Up Hold der nach folgender Formel arbeitet:

Y = KP * IN + KI * INTEG(IN)

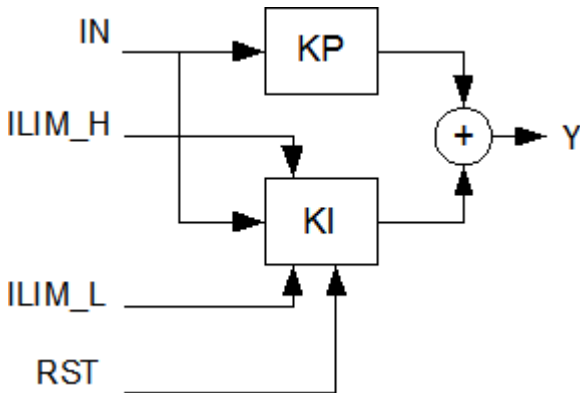
Die Eingangswerte LIM_H und LIM_L begrenzen den Wertebereich des Ausgangs Y. Mit RST kann der interne Integrator jederzeit auf 0 gesetzt werden. Der Ausgang LIM signalisiert das der Ausgang Y an eine der Grenzen

LIM_L oder LIM_H gelaufen ist. Der Regler arbeitet frei laufend und benutzt zur Berechnung des Integrators die Trapezregel für höchste Genauigkeit und optimale Geschwindigkeit. Die Default-Werte der Eingangsparameter sind wie folgt vordefiniert: KP = 1, KI = 1, ILIM_L = -1E38 und ILIM_H = +1E38.

Anti Wind-Up: Regelbausteine mit Interalanteil neigen zu dem so genannten Wind Up Effekt. Ein Wind-Up bedeutet das der Integratorbaustein kontinuierlich weiter läuft weil z.B. das Stellsignal Y an einem Anschlag steht und die Regelung über längere Zeit nicht in der Lage ist die Regelabweichung auszugleichen, was dann nach anschließendem Übergang in den Regelbereich erst zu einem langen und Zeitaufwendigen Abbau des Integratorwertes führt und die Regelung nur verzögert reagiert. Da der Integralanteil nur für den Ausgleich der Regelabweichung nach allen anderen Regelanteilen nötig ist, kann und sollte der Bereich des Integrators mit den Werten ILIM begrenzt werden.

Der Baustein FT_PIW hat einen so genannten Wind-Up Hold der bei erreichen einer Ausgangsbegrenzung (LIM_L, LIM_H) den Integrator auf dem letzten Wert einfriert und so einen Wind-Up verhindert.

Die folgende Grafik verdeutlicht die interne Struktur des Reglers:



FT_PIW kann zusammen mit den Bausteinen CTRL_IN und CTRL_OUT zum Aufbau komplexer Regler benutzt werden.

23.25. FT_PIWL

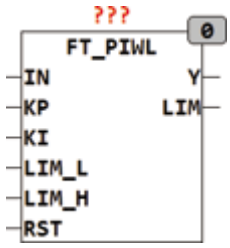
Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	KP : REAL (Proportionaler Anteil des Reglers)
	KI : REAL (Integraler Anteil des Reglers)
	LIM_L : REAL (untere Ausgangsbegrenzung des Integrators)

LIM_H : REAL (obere Ausgangsbegrenzung des Integrators)

RST : BOOL (Asynchroner Reset-Eingang)

Output Y : REAL (Ausgang des Reglers)

LIM : BOOL (TRUE, wenn der Ausgang ein Limit erreicht hat)



FT_PIWL ist ein PI-Regler mit dynamischen Anti Wind-Up der nach folgender Formel arbeitet:

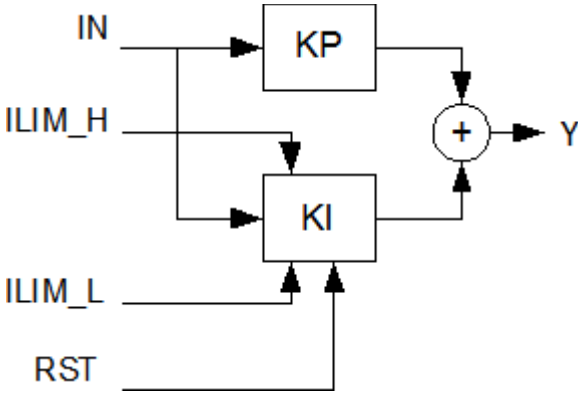
Y = KP * IN + KI * INTEG(IN)

Die Eingangswerte LIM_H und LIM_L begrenzen den Wertebereich des Ausgangs Y. Mit RST kann der interne Integrator jederzeit auf 0 gesetzt werden. Der Ausgang LIM signalisiert das der Ausgang Y an eine der Grenzen LIM_L oder LIM_H gelaufen ist. Der Regler arbeitet frei laufend und benutzt zur Berechnung des Integrators die Trapezregel für höchste Genauigkeit und optimale Geschwindigkeit. Die Default-Werte der Eingangsparameter sind wie folgt vordefiniert: KP = 1, KI = 1, ILIM_L = -1E38 und ILIM_H = +1E38.

Anti Wind-Up: Regelbausteine mit Interalanteil neigen zu dem so genannten Wind Up Effekt. Ein Wind-Up bedeutet das der Integratorbaustein kontinuierlich weiter läuft weil z.B. das Stellsignal Y an einem Anschlag steht und die Regelung über längere Zeit nicht in der Lage ist die Regelabweichung auszugleichen, was dann nach anschließendem Übergang in den Regelbereich erst zu einem langen und Zeitaufwendigen Abbau des Integratorwertes führt und die Regelung nur verzögert reagiert. Da der Integralanteil nur für den Ausgleich der Regelabweichung nach allen anderen Regelanteilen nötig ist, kann und sollte der Bereich des Integrators mit den Werten ILIM begrenzt werden.

Der Baustein FT_PIWL hat einen so genannten dynamischen Wind-Up Reset der bei erreichen einer Ausgangsbegrenzung (LIM_L, LIM_H) den Integrator auf einen Wert zurücksetzt der dem Ausgangslimit entspricht. Wenn nach erreichen einen Limits der Regler wieder in den Arbeitsbereich eintritt muss der Integrator nicht erst Auf- oder Ab- integriert werden, und der Regler ist ohne Verzögerung sofort Einsatzbereit. Die dynamische Anti Wind-Up Methode ist die in den meisten Fällen ohne Nachteile vorzuziehende Methode, da Sie den Regler nicht negativ beeinflusst und die Nachteile des Wind_Up verhindert.

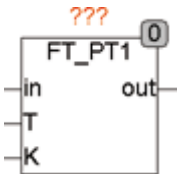
Die folgende Grafik verdeutlicht die interne Struktur des Reglers:



FT_PIWL kann zusammen mit den Bausteinen CTRL_IN und CTRL_OUT zum Aufbau komplexer Regler benutzt werden.

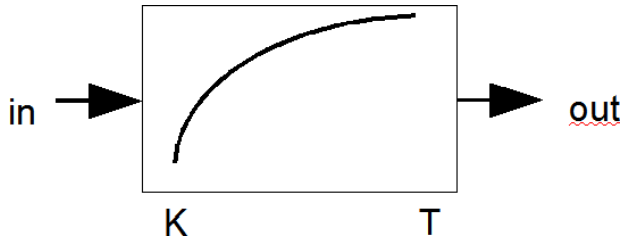
23.26. FT_PT1

Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	T : TIME (Zeitkonstante)
	K : REAL (Multiplikator)
Output	OUT : REAL (Ausgangssignal)

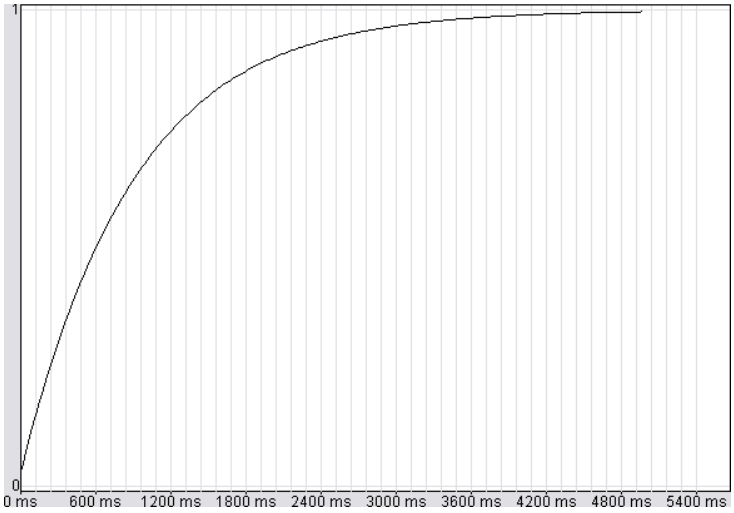


FT_PT1 ist ein LZI-Übertragungsglied mit einem proportionalen Übertragungsverhalten 1. Ordnung, auch als Tiefpass Filter 1. Ordnung bezeichnet. Der Multiplikator K legt den Verstärkungsfaktor (Multiplikator) fest und T die Zeitkonstante.
Eine Änderung am Eingang wird am Ausgang gedämpft sichtbar. Das Ausgangssignal steigt innerhalb von T auf 63% des Eingangswerts und nach 3 * T auf 95% des Eingangswerts an. Somit ist nach einer sprunghaften Änderung des Eingangssignals von 0 auf 10 der Ausgang zum Zeitpunkt der Eingangsänderung 0, steigt nach 1 * T auf 6,3 an und erreicht nach 3 * T 9,5 und nähert sich dann asymptotisch dem Wert 10 an. Beim ersten Aufruf wird der Ausgang OUT mit dem Eingangswert IN initialisiert um ein definiertes Anlauf Verhalten zu gewährleisten. Falls der Eingang T gleich T#0s ist entspricht der Ausgang OUT = K * IN.

Strukturbild:

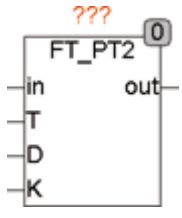


Sprungantwort für T=1s, K=1



23.27. FT_PT2

Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	T : REAL (Zeitkonstante)
	D : REAL (Dämpfung)
	K : REAL (Multiplikator)
Output	OUT : REAL (Ausgangssignal)



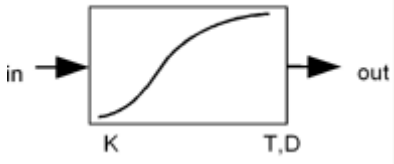
FT_PT2 ist ein LZI-Übertragungsglied mit einem proportionalen Übertragungsverhalten 2. Ordnung, auch als Tiefpass Filter 2. Ordnung bekannt. Der Multiplikator K legt den Verstärkungsfaktor (Multiplikator) fest, T und D die Zeitkonstante und die Dämpfung. Falls der Eingang T gleich T#0s ist entspricht der Ausgang OUT = K * IN.

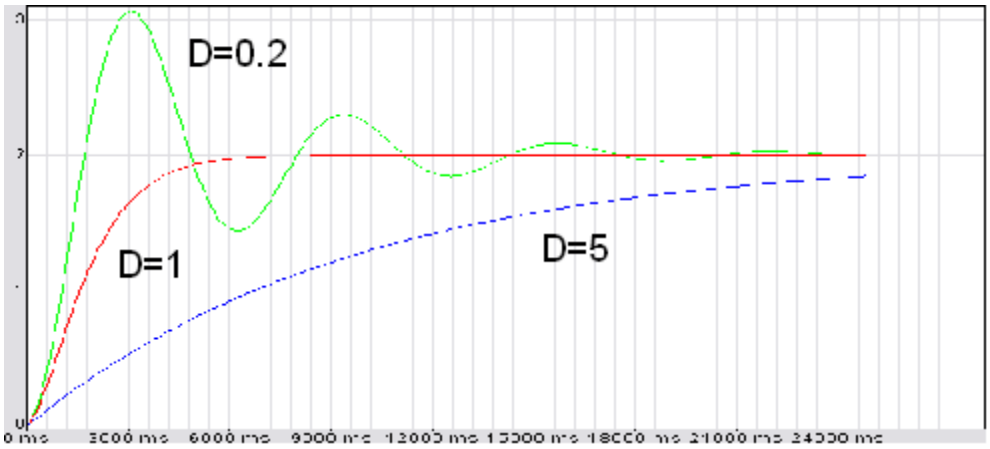
Die entsprechende Funktionalbeziehung in Zeitbereich ist folgende Differenzialgleichung gegeben:

$T^2 * OUT''(T) + 2 * D * T * OUT'(T) + OUT(T) = K * in(T)$.

Strukturbild:

Sprungantwort für T=1, K=2, D=0,2 / 1 / 5

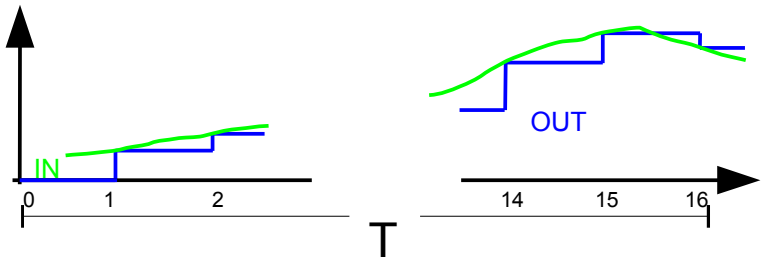




23.28. FT_TN16

Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	T : REAL (Verzögerungszeit)
Output	OUT : REAL (Ausgangssignal)

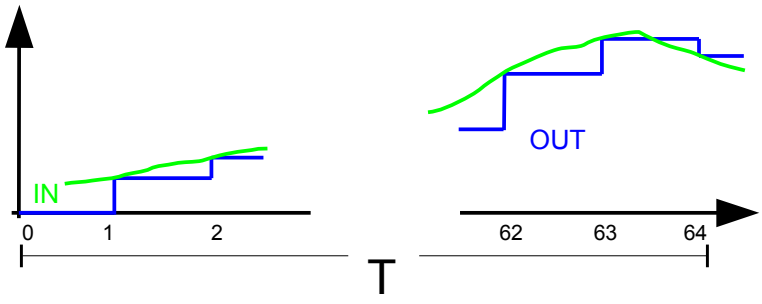
FT_TN16 verzögert ein Eingangssignal um eine einstellbare Zeit T und tastet es in der Zeit T 16 mal ab. Nach jedem update des Ausgangssignals OUT wird TRIG für einen Zyklus TRUE.



23.29. FT_TN64

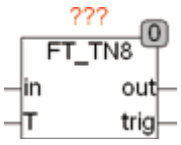
Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	T : REAL (Verzögerungszeit)
Output	OUT : REAL (Ausgangssignal)

FT_TN64 verzögert ein Eingangssignal um eine einstellbare Zeit T und tastet es in der Zeit T 64 mal ab. Nach jedem update des Ausgangssignals OUT wird TRIG für einen Zyklus TRUE.

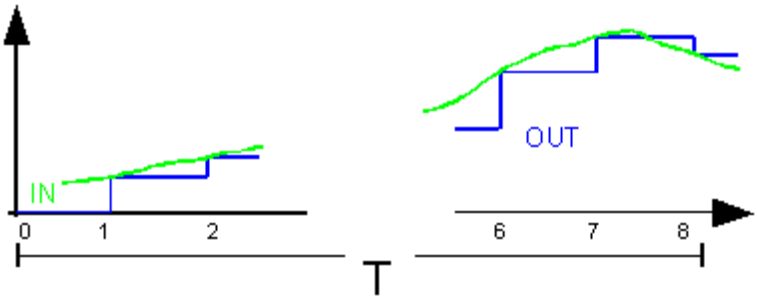


23.30. FT_TN8

Type	Funktionsbaustein
Input	IN : REAL (Eingangssignal)
	T : REAL (Verzögerungszeit)
Output	OUT : REAL (Ausgangssignal)

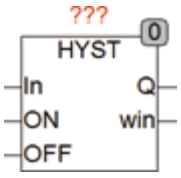


FT_TN8 verzögert ein Eingangssignal um eine einstellbare Zeit T und tastet es in der Zeit T 8 mal ab. Nach jedem update des Ausgangssignals OUT wird TRIG für einen Zyklus TRUE.



23.31. HYST

Type	Funktionsbaustein
Input	IN : REAL (Eingangswert)
	ON : REAL (oberer Schwellenwert)
	OFF : REAL (unterer Schwellenwert)
Output	Q : BOOL (Ausgangssignal)
	WIN : BOOL (zeigt an, dass In zwischen ON und OFF liegt)



HYST ist ein Standard Hysteresebaustein, Seine Funktion hängt von den Eingangswerten ON und OFF ab.

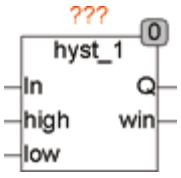
Ist ON > OFF so wird der Ausgang TRUE gesetzt wenn IN > ON und er wird FALSE gesetzt wenn IN < OFF.

Ist ON < OFF so wird der Ausgang TRUE gesetzt wenn IN < ON und er wird FALSE gesetzt wenn IN > OFF.

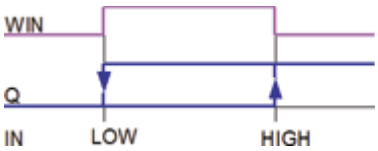
Der Ausgang WIN wird TRUE wenn IN zwischen ON und OFF liegt, liegt IN außerhalb des Bereichs ON – OFF wird WIN FALSE.

23.32. HYST_1

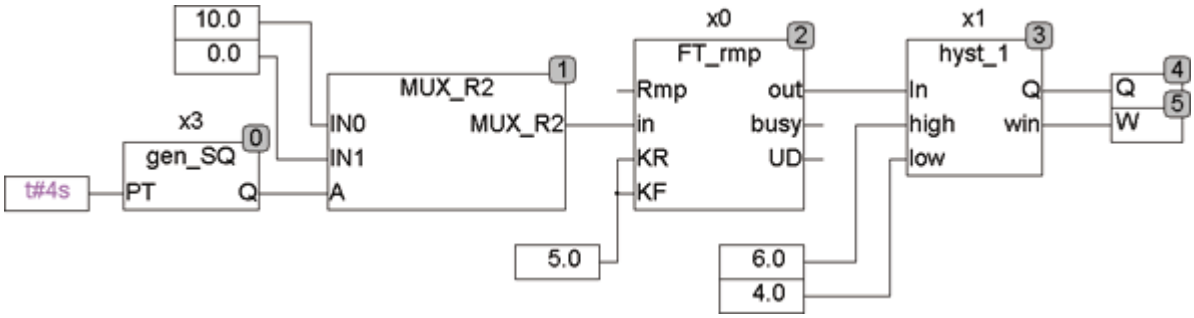
Type	Funktionsbaustein
Input	IN : REAL (Eingangswert)
	HIGH : REAL (oberer Schwellenwert)
	LOW : REAL (unterer Schwellenwert)
Output	Q : BOOL (Ausgangssignal)
	WIN : BOOL (zeigt an, dass In zwischen LOW und HIGH liegt)



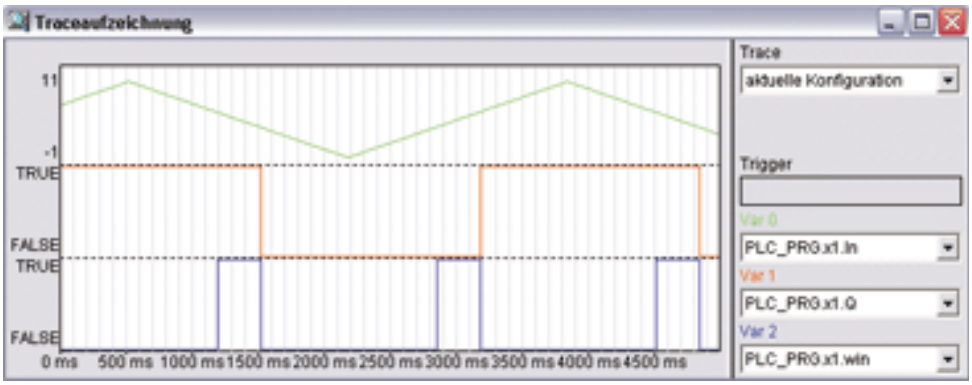
HYST_1 ist ein Hysterese Baustein der mit oberen und unterem Limit arbeitet. Der Ausgang Q wird nur dann TRUE, wenn das Eingangssignal an IN den Wert HIGH überschritten hat. Es bleibt dann solange TRUE, bis das Eingangssignal wieder LOW unterschreitet und Q FALSE wird. Ein weiterer Ausgang WIN zeigt an, ob sich das Eingangssignal zwischen LOW und HIGH befindet.



Das folgende Beispiel zeigt eine Dreiecksgenerator mit nachgeschalteten Hysterese Baustein HYST_1.



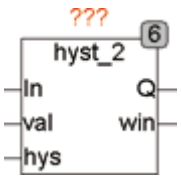
Die grüne Kurve zeigt das Eingangssignal, Rot den Hysterese Ausgang und



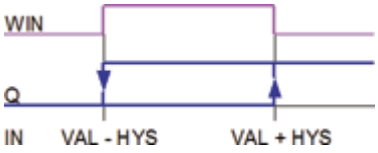
Blau den WIN Ausgang.

23.33. HYST_2

Type	Funktionsbaustein
Input	IN : REAL (Eingangswert)
	VAL : REAL (Mittelwert der Hysterese)
	HYS : REAL (Breite der Hysterese)
Output	Q : BOOL (Ausgangssignal)
	WIN : BOOL (zeigt an das In zwischen LOW und HIGH liegt)



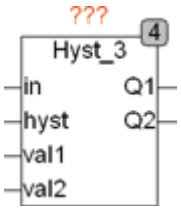
HYST_2 ist ein Hysterese Baustein bei dem Die Schaltschwellen durch einen Mittelwert und die zugehörige Hysterese definiert wird. Die untere Schaltschwelle liegt bei VAL – HYS / 2 und die obere Schaltschwelle bei VAL + HYS / 2.



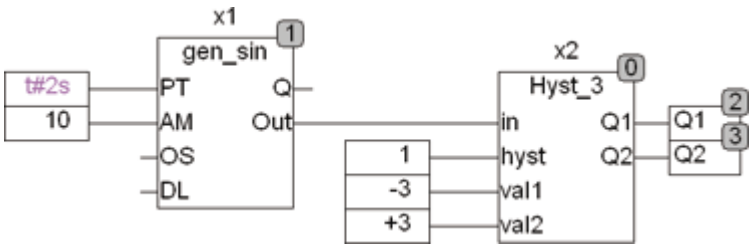
Eine eingehende Beschreibung der Hysteresefunktion und ein Anwendungsbeispiel finden Sie unter HYST_1.

23.34. HYST_3

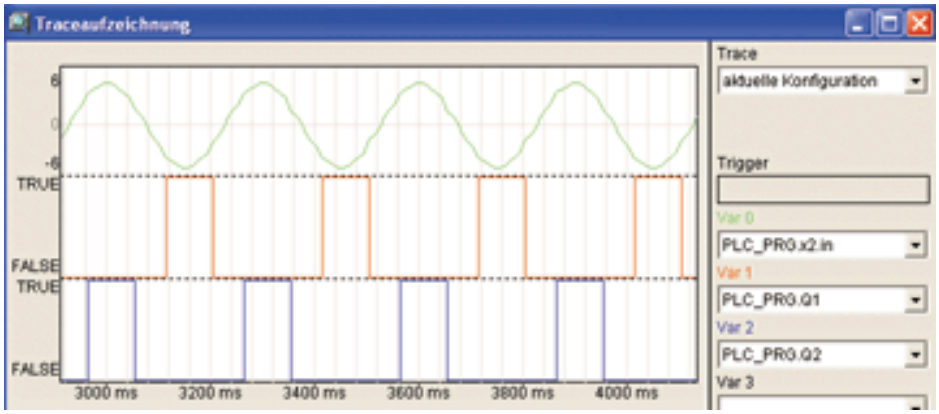
Type	Funktionsbaustein
Input	IN : REAL (Eingangswert)
	HYST : REAL (Breite der Hysterese)
	VAL1 : REAL (Mittelwert der Hysterese 1)
	VAL2 : REAL (Mittelwert der Hysterese 2)
Output	Q1 : BOOL (Ausgangssignal 1)
	Q2 : BOOL (Ausgangssignal 2)

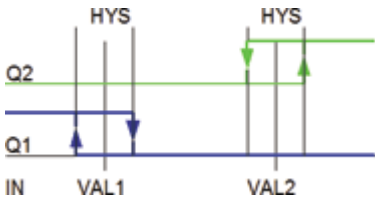


HYST_3 ist ein Dreipunktregler. Der Dreipunktregler besteht aus 2 Hysteresefunktionen. Q1 ist eine Hysterese mit val1 als Schwellenwert und HYST als Hysterese. Q1 wird TRUE, wenn IN kleiner ist als VAL1 – HYST / 2 und wird FALSE, wenn IN größer ist als VAL1 + HYST / 2. Q2 funktioniert analog mit Val2. Der Dreipunktregler wird vor allen dann eingesetzt, wenn man motorische Klappen steuert, die dann mit Q1 auf und mit Q2 ab gesteuert werden. Ist der Wert von IN zwischen VAL1 und VAL2 bleiben beide Ausgänge FALSE und der Motor bleibt stehen.



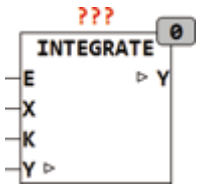
Folgendes Beispiel zeigt den Signalverlauf an einem 3-Punkt Regler:





23.35. INTEGRATE

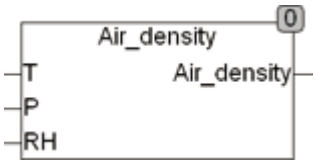
Type	Funktionsbaustein
Input	E : BOOL (Enable Eingang, Default = TRUE)
	X : REAL (Eingangswert)
	K : REAL (Integrationsbeiwert in 1/s)
I/O	Y : REAL (Integrator Ausgang)



INTEGRATE ist ein Integrator der den Wert X auf einen externen Wert Y auf integriert. Der Integrator arbeitet wenn E = TRUE, der interne Default von E = TRUE.

23.36. AIR_DENSITY

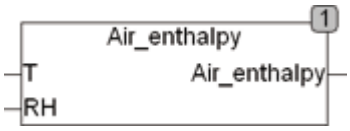
Type	Funktion : REAL
Input	T : REAL (Temperatur der Luft in °C)
	P : REAL (Luftdruck in Pascal)
	RH : REAL (Luftfeuchtigkeit in %)
Output	(Dichte der Luft in kg/m³)



AIR_DENSITY berechnet die Dichte der Luft in kg/m³ abhängig von Druck, Feuchte und Temperatur. Die Temperatur wird in °C, Druck in Pascal und die Feuchte in % (50 = 50%) angegeben.

23.37. AIR_ENTHALPY

Type	Funktion : REAL
Input	T : REAL (Temperatur der Luft)
	RH : REAL (Relative Feuchte der Luft)
Output	(Enthalpie der Luft in J/g)



AIR_ENTHALPY berechnet die Enthalpie von Feuchter Luft aus den Angaben T für Temperatur in Grad Celsius und der relativen Feuchte RH in % (50 = 50%). Die Enthalpie wird in Joule / Gramm berechnet.

23.38. BOILER

Type	Funktionsbaustein
Input	T_UPPER : REAL (Eingang oberer Temperatursensor)
	T_LOWER : REAL (Eingang unterer Temperatursensor)
	PRESSURE : REAL (Eingang Drucksensor)
	ENABLE : BOOL (Warmwasseranforderung)
	REQ_1 : BOOL (Anforderungseingang für vordefinierte Temperatur 1)
	REQ_2 : BOOL (Anforderungseingang für vordefinierte Temperatur 2)
	BOOST : BOOL (Anforderungseingang für sofortige Bereitstellung)
Output	HEAT : BOOL (Ausgang für Ladekreis)
	ERROR : BOOL (Fehlersignal)
	STATUS : Byte (ESR kompatibler Status Ausgang)
Setup	T_UPPER_MIN : REAL (Mindesttemperatur für oben)
	Default = 50
	T_UPPER_MAX : REAL (Maximaltemperatur für oben)
	Default = 60

T_LOWER_ENABLE : BOOL (FALSE, wenn unterer Temperatursensor nicht vorhanden ist)

T_LOWER_MAX : REAL (Maximaltemperatur des unten)

Default = 60

T_REQUEST_1 : REAL (Temperatur bei Anforderung 1)

Default = 70

T_REQUEST_2 : REAL(Temperatur bei Anforderung 2)

Default = 50

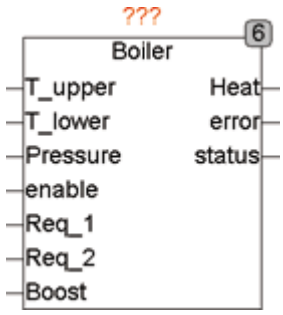
T_REQUEST_HYS : REAL (Hysterese für Regelung) Default = 5

T_PROTECT_HIGH : REAL (obere Grenztemperatur,

Default = 80)

T_PROTECT_LOW : REAL (untere Grenztemperatur,

Default = 10)

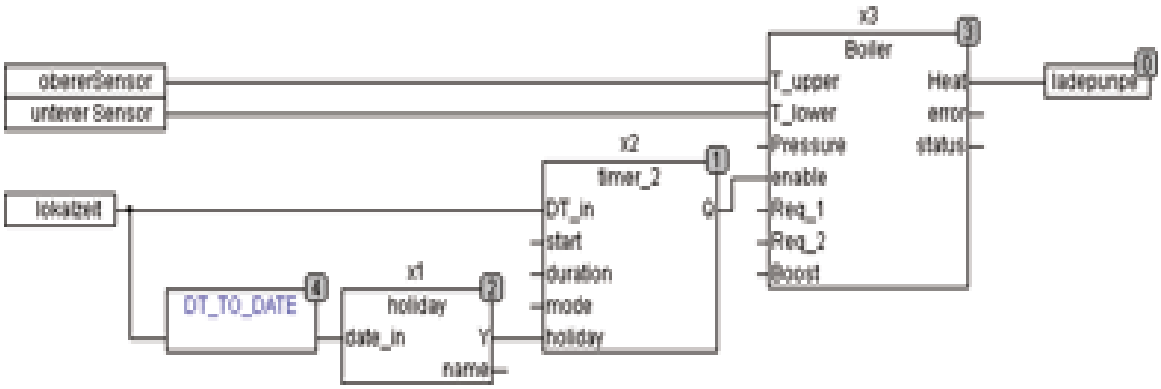


BOILER ist ein Controller für Pufferspeicher wie etwa Warmwasserspeicher. Durch 2 separate Temperatursensor-Eingänge können auch Schichtenspeicher geregelt werden. Mit der Setup-Variable T_LOWER_ENABLE kann der untere Temperatursensor aus- und eingeschaltet werden. Wenn der Eingang ENABLE auf TRUE ist, wird der Boiler aufgeheizt (HEAT = TRUE) bis die Vorgabetemperatur T_LOWER_MAX im unteren Bereich des Puffers erreicht ist und dann die Heizung ausgeschaltet, bis die untere Grenztemperatur des oberen Bereichs (T_UPPER_MIN) erreicht wird. Falls T_LOWER_ENABLE auf FALSE ist, wird der untere Sensor nicht ausgewertet und die Temperatur zwischen T_UPPER_MIN und T_UPPER_MAX im oberen Bereich geregelt. Ein PRESSURE-Eingang schützt den Boiler und verhindert die Ladung, wenn nicht genügend Wasserdruck im Boiler vorhanden ist. Falls ein Drucksensor nicht vorhanden ist, bleibt der Eingang unbeschaltet. Als weitere Schutzfunktion stehen die Vorgabewerte T_PROTECT_LOW (Frostschutz) und T_PROTECT_HIGH zur Verfügung und verhindern das die Temperatur im Puffer einen oberen Grenzwert nicht übersteigt und ein unterer Grenzwert nicht unterschritten wird. Bei Auftreten eines Fehlers wird der Ausgang ERROR auf TRUE gesetzt und gleichzeitig ein Statusbyte am Ausgang Status gemeldet, welches durch Bausteine wie ESR_COLLECT

weiter ausgewertet werden kann. Durch eine steigende Flanke am Eingang BOOST wird die Puffertemperatur unmittelbar auf T_UPPER_MAX (T_LOWER_ENABLE = FALSE) beziehungsweise T_LOWER_MAX (T_LOWER_ENABLE = TRUE) aufgeheizt. BOOST kann zur außerplanmäßigen Aufheizung des Boilers, wenn ENABLE auf FALSE ist, benutzt werden. Die Aufheizung durch BOOST ist flankengetriggert und führt bei jeder steigenden Flanke an BOOST zu genau einem Aufheizvorgang. Durch eine steigende Flanke an BOOST während ENABLE TRUE ist wird die Heizung sofort gestartet bis die maximale Temperatur erreicht ist. Der Boiler wird also nachgeladen, um maximale Wärmekapazität bereitzustellen. Die Eingänge REQ_1 und REQ_2 dienen dazu, jederzeit eine vordefinierte Temperatur (T_REQUEST_1 oder T_REQUEST_2) bereitzustellen. REQ kann zum Beispiel zur Bereitstellung einer höheren Temperatur zur Legionellendesinfektion oder auch zu anderen Zwecken verwendet werden. Die Bereitstellung der Request-Temperaturen erfolgt durch Messung am oberen Temperatursensor und mit einer 2-Punkt Regelung deren Hysterese durch T_REQUEST_HYS voreingestellt wird.

Status	
1	oberer Temperatursensor hat die obere Grenztemperatur überschritten
2	oberer Temperatursensor hat die untere Grenztemperatur unterschritten
3	unterer Temperatursensor hat die obere Grenztemperatur überschritten
4	unterer Temperatursensor hat die untere Grenztemperatur unterschritten
5	Wasserdruck im Puffer ist zu gering
100	Standby
101	BOOST Nachladung
102	Standard Nachladung
103	Nachladung auf Request Temperatur 1
104	Nachladung auf Request Temperatur 2

Das folgende Beispiel zeigt die Anwendung von BOILER mit einem TIMER und einer Feiertagsschaltung:



23.39. BURNER

Type	Funktionsbaustein
Input	IN : BOOL (Steuereingang)
	STAGE2 : BOOL (Steuereingang Stufe 2)
	OVER_TEMP : BOOL (Temperaturbegrenzung des Kessels)
	OIL_TEMP : BOOL (Thermostat der Ölvorerwärmung)
	FLAME : BOOL (Flammwächter)
	RST : BOOL (Reset-Eingang Für Störrücksetzung)
	RST_TIMER : BOOL (Reset für die Betriebszähler)
Output	MOTOR : BOOL (Steuersignal für den Motor)
	COIL1 : BOOL (Steuersignal für Ölventil Stufe 1)
	COIL2 : BOOL (Steuereingang für Ölventil Stufe 2)
	PRE_HEAT : BOOL (Ölvorerwärmung)
	IGNITE : BOOL (Zündung)
	KWH : REAL (Kilowattstundenzähler)
	STATUS : Byte (ESR Kompatibler Statusausgang)
	FAIL : BOOL (Störmeldung: TRUE, wenn Fehler Auftritt)
I/O	RUNTIME1 : UDINT (Betriebszeit Stufe 1)
	RUNTIME2 : UDINT (Betriebszeit Stufe 2)
	CYCLES : UDINT (Anzahl der Brenner Starts)
Setup	PRE_HEAT_TIME : TIME (maximale Zeit für die Ölvorerwärmung)
	PRE_VENT_TIME : TIME (Vorbelüftungszeit)
	PRE_IGNITE_TIME : TIME (Vorzündungszeit)
	POST_IGNITE_TIME : TIME (Nachzündungszeit)
	STAGE2_DELAY : TIME (Verzögerung Stufe 2)

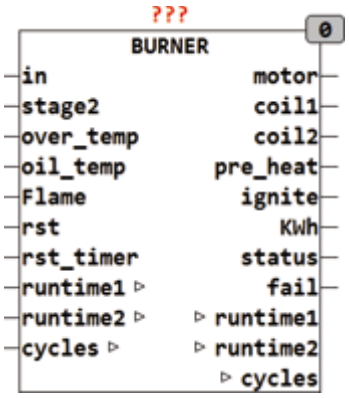
SAFETY_TIME : TIME ()

LOCKOUT_TIME : TIME (Zeit die vergehen muss, bevor mit einem RST eine Störung gelöscht werden kann)

MULTIPLE_IGNITION : BOOL ()

KW1 : REAL (Leistung des Brenners auf Stufe 1 in KW)

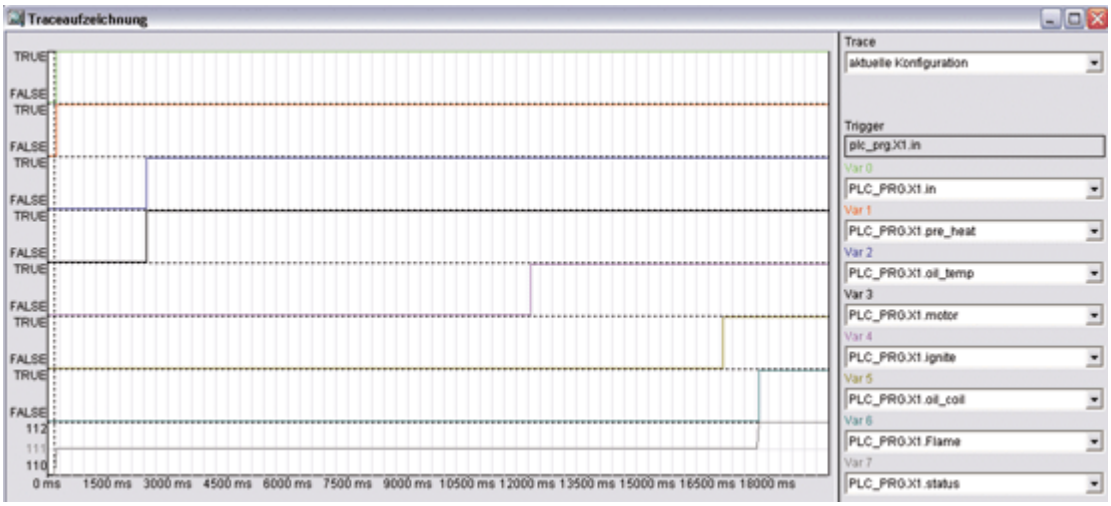
KW2 : REAL (Leistung des Brenners auf Stufe 2 in KW)



BURNER ist ein Steuerinterface für Öl- oder Gasbrenner mit Betriebszähler und Kilowattstundenzähler. Der Baustein steuert einen zweistufigen Brenner mit optionaler Ölvorerwärmung. Der Eingang IN ist der Steuereingang, der den Brenner nur dann startet, wenn der Eingang OVER_TEMP FALSE ist. OVER_TEMP ist der Kesselschutzthermostat, der TRUE wird, wenn die Kesseltemperatur die maximal zulässige Temperatur erreicht hat. Ein Brennerstart beginnt mit der Ölvorerwärmung, indem PRE_HEAT TRUE wird. Dann wird auf ein Signal am Eingang OIL_TEMP gewartet. Falls innerhalb der PRE_HEAT_TIME das Signal OIL_TEMP nicht TRUE wird und die Öltemperatur nicht erreicht wird, wird die Startsequenz unterbrochen und der Ausgang Störung gesetzt. Gleichzeitig wird am Ausgang Status Der Fehler 1 ausgegeben. Nach der Ölvorerwärmung wird der Motor eingeschaltet und damit der Ventilator in Betrieb gesetzt. Anschließend wird nach definierter Zeit die Zündung eingeschaltet und danach das Ölventil geöffnet. Sollte dann nach spezifizierter Zeit (SAFETY_TIME) der Flammwächter nicht ansprechen, so geht der Baustein auf Störung. Eine Störung wird auch dann signalisiert, wenn der Flammwächter bereits vor der Zündung anspricht. Falls nach erfolgreicher Zündung die Flamme abreißt und die Setup-Variable MULTIPLE_IGNITION = TRUE steht, wird sofort wieder gezündet. Eine zweite Stufe wird nach Ablauf der STAGE2_DELAY Zeit automatisch zugeschaltet wenn der Eingang STAGE2 TRUE ist.

Tritt eine Störung auf, so wird der Baustein für eine feste Zeit LOCKOUT_TIME blockiert und erst danach kann ein RST den Betrieb wieder starten. Während der LOCKOUT_TIME muss der RST-Eingang FALSE sein. Ein TRUE am Eingang OVER_TEMP stoppt sofort jede Aktion und meldet den Fehler 9.

Der Status-Ausgang signalisiert den momentanen Zustand des Bausteins:



110 = Warten auf Startsignal (Standby)
111 = Startsequenz wird durchlaufen
112 = Brenner Läuft auf Stufe 1
113 = Brenner läuft auf Stufe 2

Eine Reihe von Fehlerzuständen werden am Ausgang STATUS bereitgestellt, wenn ein Fehler Auftritt:

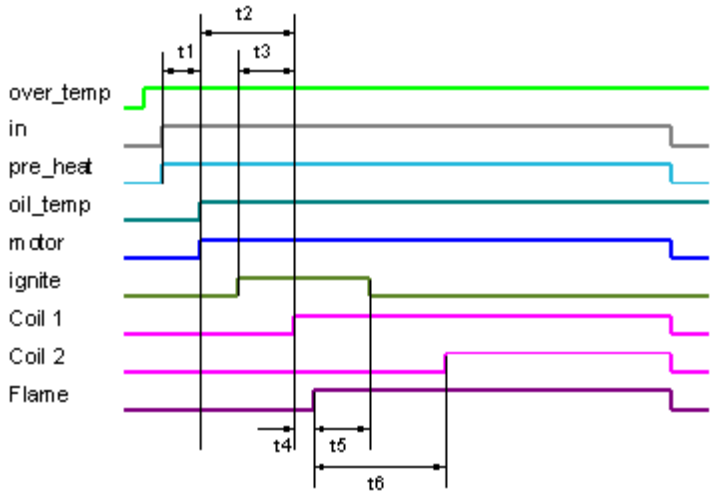
1 = Ölvorerwärmung hat innerhalb der PRE_HEAT_TIME nicht angesprochen
2 = Flammwächter ist aktiv während der Ölvorerwärmung (PRE_HEAT_TIME)
3 = Flammwächter ist aktiv während der Belüftungszeit (PRE_VENTILATION_TIME)
4 = Sicherheitszeit (Safety_TIME) wurde ohne Flamme überschritten
5 = Flamme ist im Betrieb Abgerissen
9 = Kessel Übertemperatur Kontakt hat Ausgelöst

Traceaufzeichnung einer Normalen Startsequenz:

Das Signal IN startet die Sequenz mit dem Ausgang PRE_HEAT. Nach erreichen der Öltemperatur (OIL_TEMP = TRUE) wird der Motor gestartet und die PRE_VENTILATION_TIME (Zeit von Motor Start bis Ölventil offen ist) abgewartet. Nach einer einstellbaren Zeit (PPR_IGNITION_TIME) vor dem Öffnen des Ölventils wird die Zündung eingeschaltet. Die Zündung bleibt dann solange ein, bis die POST_IGNITION_TIME abgelaufen ist. Die Betriebszeit wird je Stufe unabhängig in Sekunden gemessen.

In	over tem	Oil tem	Flame	Rst	mo-tor	Oil coil	Pre hea	ig-nite	Sta-tus	fail	
0	0	-	-	0	0	0	0	0	110	0	Wartezustand
1	0	0	0	0	0	0	1	0	111	0	Ölvorwärmphase
1	0	1	0	0	1	0	1	0	111	0	Vorbelüftungsphase
1	0	1	0	0	1	0	1	1	111	0	Vorzündphase
1	0	1	0	0	1	1	1	1	111	0	Ventil Stufe 1 öffnen
1	0	1	1	0	1	1	1	1	112	0	Flamme brennt Nachzündphase
1	0	1	1	0	1	1	1	0	112	0	Brenner läuft
1	0	1	0	0	1	1	1	1	111	0	Nachzündung nach Flammabriß
-	1	-	-	-	-	-	-	-	9	1	Kessel Übertemperatur
1	0	1	1	0	1	0	1	0	3	1	Fremdlichtfehler

Das folgende Zeitdiagramm erläutert die verschiedenen Setup-Zeiten und den Ablauf:



Das Zeitdiagramm gibt den genauen Zeitverlauf wieder:

t1 = Vorheizzeit (PRE_HEAT_TIME)

t2 = Vorbelüftungszeit (PRE_VENT_TIME)

t3 = Vorzündungszeit (PRE_IGNITE_TIME)

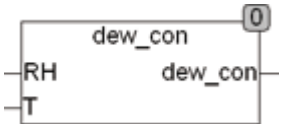
t4 = Sicherheitszeit (SAFETY_TIME)

t5 = Nachzündungszeit (POST_IGNITE_TIME)

t6 = Verzögerung für Stufe 2 (STAGE2_DELAY)

23.40. DEW_CON

Type	Funktion : REAL
Input	RH : REAL (Relative Feuchte)
	T : REAL (Temperatur in °C)
Output	REAL (Wasserdampf Konzentration in Gramm / m³)

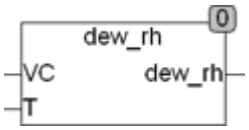


Der Baustein DEW_CON berechnet aus der Relativen Feuchte (RH) und der Temperatur (T in °C) die Wasserdampfkonzentration in der Luft. Das Ergebnis wird in Gramm / m³ ermittelt. RH ist in % (50 = 50%) anzugeben und die Temperatur in °C.

Die Baustein ist für Temperaturen von -40°C bis +90°C geeignet.

23.41. DEW_RH

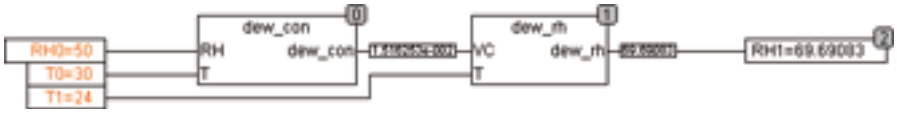
Type	Funktion : REAL
Input	VC : REAL (Wasserdampfkonzentration in Luft in Gramm / m³)
	T : REAL (Temperatur in °C)
Output	REAL (Relative Luftfeuchtigkeit in %)



Der Baustein DEW_RH berechnet aus der Wasserdampfkonzentration (VC) und der Temperatur (T in °C) die relative Luftfeuchtigkeit in % (50 = 50%). Die Wasserdampfkonzentration wird in Gramm/m³ angegeben. DEW_CON kann für Berechnungen in beide Richtungen (aufheizen und abkühlen) verwendet werden. Wird zu stark abgekühlt, so ist die maximale relative Feuchte auf 100% begrenzt. Für Berechnungen des Taupunktes wird der Baustein DEW_TEMP empfohlen.

Im folgenden Beispiel wird der Fall berechnet, wenn Luft von 30°C und relativer Feuchte von 50% um 6 Grad abgekühlt wird. Der Baustein DEW_CON liefert die Feuchtigkeitskonzentration in der Ausgangsluft von 30° und DEW_RH berechnet die resultierende relative Luftfeuchtigkeit RH von 69,7%. Diese Berechnungen sind wichtig, wenn Luft abgekühlt oder

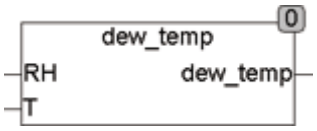
aufgeheizt wird. In Klimaanlagen ist eine resultierende relative Feuchte von 100% wegen Taubildung und den daraus resultierenden Problemen zu vermeiden.



Siehe hierzu auch die Bausteine DEW_CON und DEW_TEMP.

23.42. DEW_TEMP

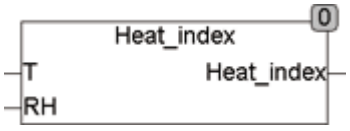
Type	Funktion : REAL
Input	RH : REAL (Relative Feuchte)
	T : REAL (Temperatur in °C)
Output	REAL (Taupunkttemperatur)



Der Baustein DEW_TEMP berechnet aus der Relativen Feuchte (RH) und der Temperatur (T in °C) die Taupunkttemperatur. Die Relative Feuchte wird in % angegeben (50 = 50%).

23.43. HEAT_INDEX

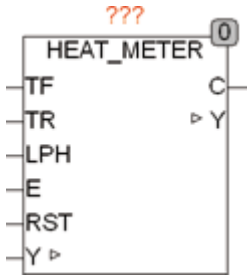
Type	Funktion : REAL
Input	T : REAL (Temperatur in °C)
	RH : REAL (Relative Feuchte)
Output	REAL (Heat Index Temperatur)



HEAT_INDEX berechnet die bei hohen Temperaturen und hoher Feuchte gefühlte Temperatur. Die Funktion ist definiert für Temperaturen größer 20 °C und relativer Feuchte > 10%. Für Werte außerhalb des Definitionsbereichs wird die Eingangstemperatur ausgegeben.

23.44. HEAT_METER

Type	Funktion : REAL
Input	TF : REAL (Vorlauftemperatur in °C)
	TR : REAL (Rücklauftemperatur in °C)
	LPH : REAL (Durchflussmenge in L/h bzw. L/Impuls)
	E : BOOL (Enable Signal)
	RST : BOOL (asynchroner Reset Eingang)
Setup	CP : REAL (Spezifische Wärmekapazität der 2ten Komponente)
	DENSITY : REAL (Dichte der 2ten Komponente)
	CONTENT : REAL (Anteil, 1 = 100%)
	PULSE_MODE : BOOL (Impulszähler wenn TRUE)
	RETURN_METER : BOOL (Durchflussmesser im Rücklauf wenn TRUE)
	AVG_TIME : TIME (Zeitintervall für Momentanen Verbrauch)
Output	C : REAL (aktueller Verbrauch in Joule / Stunde)
I/O	Y : REAL (Wärmemenge in Joule)



HEAT_METER ist ein Wärmemengenzähler. Die Wärmemenge Y wird in Joule gemessen. Die Eingänge TF und TR sind die Vorlauf und Rücklauftemperatur des Mediums. Am Eingang LPH wird die Durchflussmenge in Liter / Stunde beziehungsweise die Durchflussmenge je Impuls an E spezifiziert. Die Eigenschaft von E wird durch die Setup Variable PULSE_MODE bestimmt. PULSE_MODE = FALSE bedeutet das die Wärmemenge kontinu-

ierlich aufaddiert wird solange E auf TRUE ist. PULSE_MODE = TRUE bedeutet das die Wärmemenge mit jeder steigenden Flanke von E aufaddiert wird. Der PULSE_MODE ist bei Verwendung von Wärmezählern einzuschalten, während am Eingang LPH die Flussmenge in Litern je Impuls anzugeben ist und am Eingang E wird der Wärmezähler angeschlossen. Wenn kein Flussmengenmesser Vorhanden ist, so wird am Eingang E das Pumpensignal angeschlossen und am Eingang LPH die Pumpenleistung in Litern / Stunde angegeben. Bei Verwendung eines Flussmengenmessers mit Analogem Ausgang wird der Ausgang entsprechend auf Liter / Stunde umgerechnet und auf den Eingang LPH gelegt, Der Eingang E wird hierbei auf TRUE gesetzt. Mit den Setup- Variablen CP, DENSITY und CONTENT wird die 2te Komponente des Mediums spezifiziert. Für den Betrieb mit reinem Wasser sind keinerlei Angaben von CP, DENSITY und CONTENT nötig. Wenn ein Gemisch aus Wasser und einem 2ten Medium vorhanden ist werden mit CP die Spezifische Wärmekapazität in J/KgK, mit DENSITY die DICHTE in KG/l und mit CONTENT der Anteil der 2ten Komponente spezifiziert. Ein Anteil von 0.5 bedeutet 50% und 1 wäre entsprechend 100%. Mit der Setup Variablen RETURN_METER wird angegeben ob der Durchflussmesser im Vorlauf oder Rücklauf sitzt. RETRUN_METER = TRUE steht für Rücklaufmessung und FALSE für Vorlaufmessung. Am Ausgang C stellt der Baustein den momentanen Verbrauch zur Verfügung. Der momentane Verbrauch wird in Joule / Stunde angegeben, und in den Zeitabständen von AVG_TIME ermittelt.

Der Baustein hat folgende Vorgabewerte, die aktiv sind wenn die entsprechenden Werte vom Anwender nicht gesetzt werden:

PULSE_MODE = FALSE

RETURN_METER = FALSE

AVG_TIME = T#5s

23.45. HEAT_TEMP

Type	Funktionsbaustein
Input	T_EXT : REAL (Außentemperatur)
	T_INT : REAL (Soll Raumtemperatur)
	OFFSET : REAL (Absenkung oder Anhebung der Raumtemperatur)
	T_REQ : REAL (Temperaturanforderung)
Output	TY : REAL (Heizkreisvorlauftemperatur)
	HEAT : BOOL (Heizungsanforderung)
Setup	TY_MAX : REAL (maximale Heizkreistemperatur, 70°C)

TY_MIN : REAL (Minimale Heizkreistemperatur, 25°C)

TY_C : REAL (Auslegungstemperatur, 70°C)

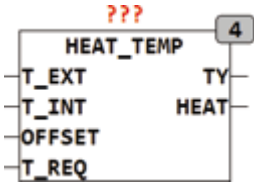
T_INT_C : REAL (Auslegungstemperatur Raum, 20°C)

T_EXT_C : REAL (T_EXT bei Auslegungstemperatur -15°C)

T_DIFF_C : REAL (Vor- Rücklaufdifferenz 10°C)

C : REAL (Konstante des Heizsystems, DEFAULT = 1,33)

H : REAL (Schwelle für Heizungsanforderung 3°C)



HEAT_TEMP berechnet die Vorlauftemperatur aus der Außentemperatur nach folgender Formel:

TY = TR + T_DIFF / 2 * TX + (TY_Setup - T_DIFF / 2 - TR) * TX ^ (1 / C)

mit: TR = T_INT + OFFSET

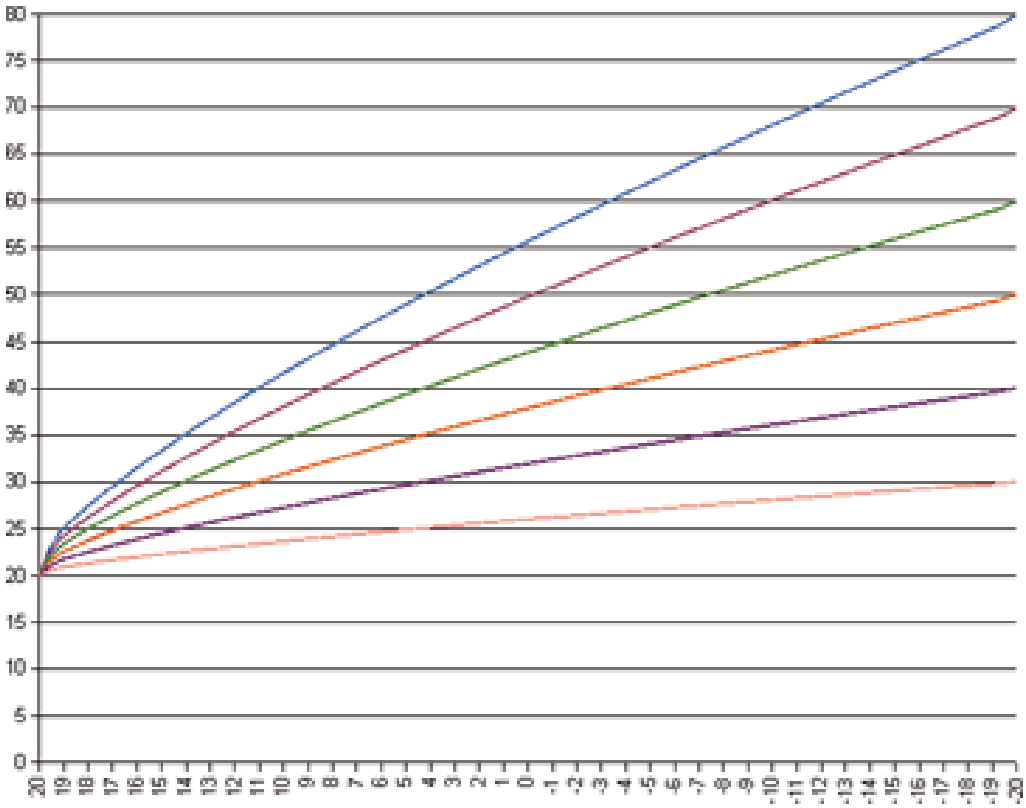
TX := (TR - T_EXT) / (T_INT_Setup - T_EXT_Setup);

Die Parameter der Heizkurve werden durch die Setup Variablen TY_C (Auslegungsvorlauftemperatur), T_INT_C (Raumtemperatur im Auslegungspunkt), T_EXT_C (Außentemperatur im Auslegungspunkt) und T_DIFF_C (Differenz Vor- Rücklauf im Auslegungspunkt) vorgegeben. Mit dem Eingang Offset kann die Heizkurve an Raumabsenkung (negativer Offset) oder Raumanhebung (positiver Offset) angepasst werden. Mit den Setup Variablen TY_MIN und TY_MAX kann die Vorlauftemperatur auf einen Minimal- und Maximalwert begrenzt werden. Der Eingang T_REQ dient dazu, externe Temperaturanforderungen wie z.B. vom Boiler zu unterstützen. Ist T_REQ größer als der aus der Heizkurve berechnete Wert für TY, so wird TY auf T_REQ gesetzt. Die Begrenzung auf TY_MAX gilt nicht für die Anforderung durch T_REQ. Durch die Setup Variable H wird festgelegt ab welcher Außentemperatur die Heizkurve berechnet wird, solange T_EXT + H >= als T_INT + OFFSET ist bleibt TY auf 0 und HEAT ist FALSE. Wird T_EXT + H < als T_INT + OFFSET wird HEAT TRUE und TY gibt die berechnete Vorlauftemperatur aus. Die Setup Variable C legt die Krümmung der Heizkurve fest. Die Krümmung ist abhängig vom verwendeten Heizsystem.

Konvektoren:	C = 1.25 – 1.45
Plattenheizkörper:	C = 1.20 – 1.30
Radiatoren:	C = 1.30
Rohre:	C = 1.25
Fußbodenheizung:	C = 1.1

Je größer der Wert von C, desto stärker ist die Heizkurve gekrümmt. Ein Wert von 1.0 ergibt eine Gerade als Heizkurve. Typische Heizsysteme liegen zwischen 1.0 und 1.5.

Die Grafik zeigt Heizkurven für Auslegungstemperaturen von 30 – 80 °C Vorlauftemperatur bei -20°C Außentemperatur und bei einem C von 1.33:

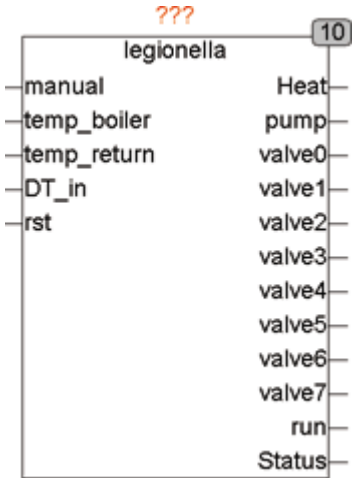


23.46. LEGIONELLA

Type	Funktionsbaustein
Input	MANUAL : BOOL (Manual Start Input)
	TEMP_BOILER : REAL (Boiler Temperatur)
	TEMP_RETURN : REAL (Temperatur der Zirkulationsleitung)
	DT_IN : DATE_TIME (Momentane Tageszeit und Datum)
	RST : BOOL (Asynchroner Reset)
Output	HEAT : BOOL (Steuersignal für Warmwasserheizung)

PUMP : BOOL (Steuersignal für Zirkulationspumpe)
STATUS : Byte (ESR kompatibler Statusausgang)
VALVE0..7 : BOOL (Steuerausgänge für Ventile der Zirkulation)
RUN : BOOL (TRUE wenn Sequenz läuft)

Setup
T_START : TOD (Tageszeit zu der die Desinfizierung startet)
DAY : INT (Wochentag an dem die Desinfizierung startet)
TEMP_SET : REAL (Temperatur des Boilers)
TEMP_OFFSET : REAL ()
TEMP_HYS : REAL ()
T_MAX_HEAT : TIME (maximale Zeit zum Aufheizen des Boilers)
T_MAX_RETURN : TIME (maximale Zeit, bis der Eingang
TEMP_RETURN nach VALVE aktiv wird)
TP_0 .. 7 : TIME (Desinfektionszeit für Kreise 0..7)



LEGIONELLA hat eine integrierte Schaltuhr, die an einem bestimmten Wochentag (DAY) zu einer bestimmten Tageszeit (T_START) die Desinfektion startet. Hierzu ist die externe Anschaltung der Lokalzeit nötig (DT_IN). Jederzeit kann mit einer steigenden Flanke an MANUAL die Desinfektion auch von Hand gestartet werden.

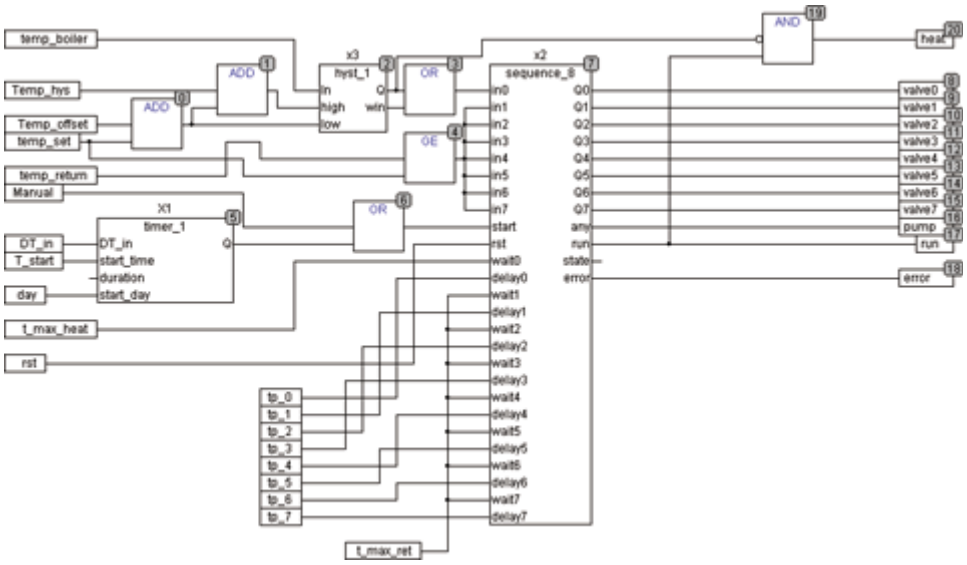
Der Ablauf eines Desinfektionszyklus wird mit einem internen Start aufgrund von DT_IN, DAY und T_START, oder durch eine steigende Flanke an MANUAL gestartet. Der Ausgang HEAT wird TRUE und steuert die Heizung des Boilers an. Innerhalb der Aufheizzeit T_MAX_HEAT muss dann das Eingangssignal TEMP_BOILER auf TRUE gehen. Wird die Temperatur nicht innerhalb von T_MAX_HEAT gemeldet, geht der Ausgang Status auf Störung. Die Desinfektion läuft aber trotzdem weiter. Nach der Aufheizphase wird die Boilertemperatur gemessen und falls nötig durch TRUE am Ausgang

HEAT wieder nachgeheizt. Sobald die Boilertemperatur erreicht ist, wird PUMP TRUE und die Zirkulationspumpe eingeschaltet. Dann werden nacheinander die einzelnen Ventile geöffnet und gemessen, ob innerhalb der Zeit T_MAX_RETURN die Temperatur am Rücklauf der Zirkulationsleitung erreicht wurde. Falls ein Rückflussthermometer nicht vorhanden ist, kann der Eingang T_MAX_RETURN einfach offen bleiben.

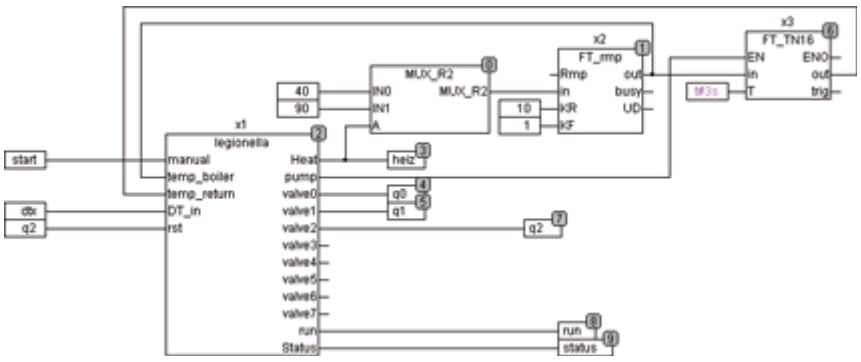
Der Ausgang Status ist ESR kompatibel und kann folgende Meldungen abgeben:

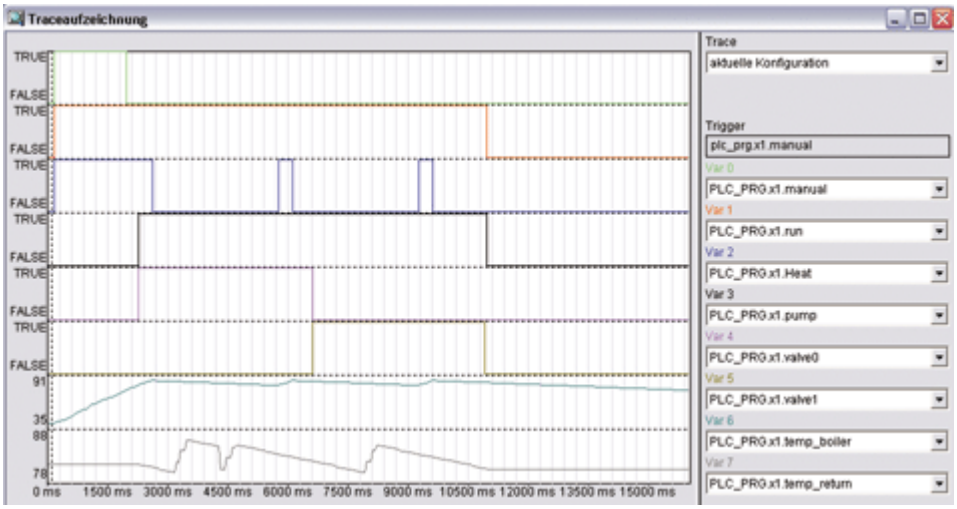
110	Wartestellung
111	Sequenz läuft
1	Boiler Temperatur wurde nicht erreicht
2	Rücklauftemperatur bei Ventil0 wurde nicht erreicht
3..8	Rücklauftemperatur bei Ventil1..7 wurde nicht erreicht

Schematischer interner Aufbau von LEGIONELLA:



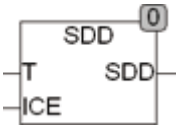
Das Folgende Beispiel zeigt eine Simulation für 2 Desinfektionskreise mit Traceaufzeichnung. In diesem Aufbau ist VALVE2 auf den Eingang RST geschaltet und unterbricht damit die Sequenz nach 2 Kreisen:





23.47. SDD

Type	Funktion : REAL
Input	T : REAL (Temperatur der Luft in °C)
	ICE : BOOL (TRUE für Luft über Eis und FALSE für Luft über Wasser)
Output	REAL (Sättigungsdampfdruck in Pa)



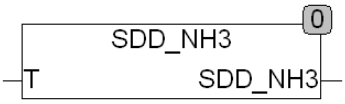
SDD berechnet den Sättigungsdampfdruck für Wasserdampf in Luft. Die Temperatur T wird in °C angegeben. Das Ergebnis kann sowohl für Luft über Eis (ICE = TRUE) und für Luft über Wasser (ICE = FALSE) berechnet werden. Der Gültigkeitsbereich der Funktion liegt bei -30°C bis 70°C über Wasser und bei -60°C bis 0°C über Eis. Die Berechnung wird nach der Magnusformel ausgeführt.

23.48. SDD_NH3

Type	Funktion : REAL

Input T : REAL (Temperatur in °C)

Output REAL (Sättigungsdampfdruck in Pa)



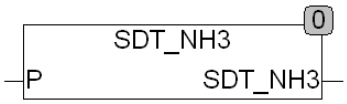
SDD_NH3 berechnet den Sättigungsdampfdruck für Ammoniak (NH3). Die Temperatur T wird in °C angegeben. Der Gültigkeitsbereich der Funktion liegt bei -109°C bis 98°C.

23.49. SDT_NH3

Type Funktion : REAL

Input T : REAL (Temperatur in °C)

Output REAL (Sättigungsdampfdruck in Pa)



SDT_NH3 berechnet die Sättigungsdampftemperatur für Ammoniak (NH3). Der Druck P wird in °C angegeben. Der Gültigkeitsbereich der Funktion liegt bei 0.001 bar bis 60 bar.

23.50. T_AVG24

Type Funktionsbaustein

Input TS : INT (Außentemperatur Sensor)

DTI : DT (Datum und Tageszeit)

RST : BOOL (Reset)

Setup T_FILTER : TIME (T des Eingangsfilters)

SCALE : REAL := 1.0 (Skalierungsfaktor)

OFS : REAL (Nullpunktabgleich)

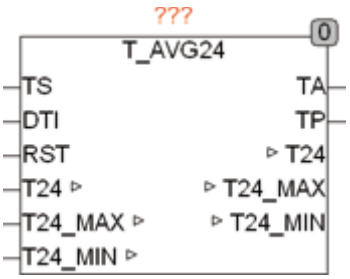
Output TA : REAL (Momentane Außentemperatur)

TP : BOOL (TRUE wenn T24 erneuert wird

I/O T24 : REAL (Tagesmitteltemperatur)

T24_MAX : REAL (Maximaltemp. in den letzten 24 Stunden)

T24_MIN : REAL (Minimaltemperatur in den letzten 24 Stunden)



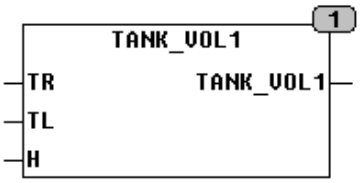
T_AVG24 ermittelt die Tagesmitteltemperatur T24. Der Sensor Eingang TS ist vom Typ INT und stellt die Temperatur * 10 dar (ein Wert von 234 bedeutet 23,4 °C). Die Daten von Filter laufen zur Unterdrückung von Störungen über ein Tiefpassfilter mit der Zeit T_FILTER. Mittels SCALE und OFS können Nullpunktfehler und Skalierung des Sensors angepasst werden. Der Ausgang TA gibt die aktuelle Außentemperatur, welche jede halbe und volle Stunde gemessen wird, an. Der Baustein schreibt alle 30 Minuten den über die letzten 48 Werte ermittelten Tagesmittelwert in die I/O Variable T24, die extern definiert werden muss und dadurch auch REMANENT oder PERSISTENT definiert werden kann. Wird beim ersten Start ein Wert von -1000 in T24 vorgefunden, so initialisiert sich der Baustein beim ersten Aufruf mit dem aktuellen Sensorwert, so dass danach alle 30 Minuten ein gültiger Mittelwert ausgegeben werden kann. Hat T24 einen beliebigen anderen Wert als -1000, so initialisiert sich der Baustein mit diesem Wert und Rechnet den Mittelwert basierend aus diesem Wert weiter. Dies ermöglicht bei Stromausfall und remanenter Speicherung von T24 ein sofortiges Weiterarbeiten nach dem wieder Einschalten. Ein Reset Eingang kann jederzeit einen Neustart des Bausteins erzwingen, wobei abhängig vom Wert in T24 der Baustein entweder mit TS oder dem alten Wert von T24 initialisiert wird. Möchte man den Baustein auf einen bestimmten Mittelwert setzen, so wird der gewünschte Wert in T24 geschrieben und dann ein Reset erzeugt.

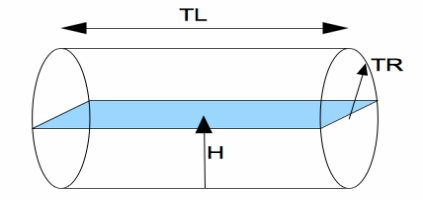
T24_MAX und T24_MIN geben den Maximal- und Minimal-Wert der letzten 24 Stunden aus. Zur Ermittlung des Maximal und Minimal Wertes werden die Temperaturen zur jeweils halben Stunde berücksichtigt. Eine Temperaturwert der zwischen 2 Messungen auftritt wird hierbei nicht berücksichtigt.

23.51. TANK_VOL1

Type Funktion : REAL

Input	TR : REAL	(Radius des Tanks)
	TL : REAL	(Länge des Tanks)
	H : REAL	(Füllhöhe des Tanks)
Output	Real	(Inhalt des Tanks bis zur Füllhöhe)

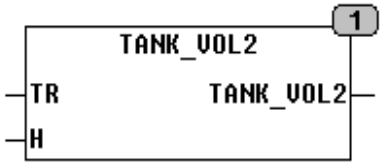


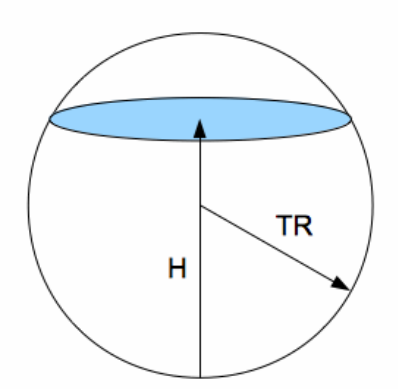


TANK_VOL1 berechnet den Inhalt eines Rohrförmigen Tanks der bis zur Höhe H gefüllt ist.

23.52. TANK_VOL2

Type	Funktion : REAL	
Input	TR : REAL	(Radius des Tanks)
	H : REAL	(Füllhöhe des Tanks)
Output	Real	(Inhalt des Tanks bis zur Füllhöhe)



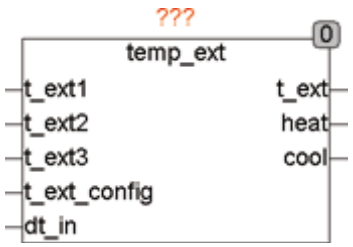


TANK_VOL2 berechnet den Inhalt eines Kugelförmigen Tanks der bis zur Höhe H gefüllt ist.

23.53. TEMP_EXT

Type Funktionsbaustein

Input	T_EXT1 : REAL (Außentemperatur Sensor 1)
	T_EXT2 : REAL (Außentemperatur Sensor 2)
	T_EXT3 : REAL (Außentemperatur Sensor 3)
	T_EXT_Setup : BYTE (Abfragemodus)
	DT_IN : DATE_TIME (Tageszeit)
Output	T_EXT : REAL (Ausgang Außentemperatur)
	HEAT : BOOL (Heizsignal)
	COOL : BOOL (Kühlsignal)
Setup	T_EXT_MIN : REAL (Minimum Außentemperatur)
	T_EXT_MAX : REAL (Maximum Außentemperatur)
	T_EXT_DEFAULT : REAL (Default Außentemperatur)
	HEAT_PERIOD_START : DATE (Beginn der Heizperiode)
	HEAT_PERIOD_STOP : DATE (Ende der Heizperiode)
	COOL_PERIOD_START : DATE (Beginn der Kühlperiode)
	COOL_PERIOD_STOP : DATE (Ende der Kühlperiode)
	HEAT_START_TEAMP_DAY (Heiztriggertemperatur Tag)
	HEAT_START_TEAMP_NIGHT (Heiztriggertemperatur Nacht)
	HEAT_STOP_TEMP : REAL (Heizen Stopp Temperatur)
	COOL_START_TEAMP_DAY (Kühl Start Temperatur Tag)
	COOL_START_TEMP_NIGHT (Kühl Start Temperatur Nacht)
	COOL_STOP_TEMP : REAL (Kühl Stopp Temperatur)
	START_DAY : TOD (Anfang des Tages)
	START_NIGHT : TOD (Anfang der Nacht)
	CYCLE_TIME : TIME (Abfragezeit für Außentemperatur)



TEMP_EXT verarbeitet bis zu 3 Außentemperaturfühler und stellt eine durch Mode selektierte Außentemperatur der Heizungsregelung zur Verfügung. Es errechnet Signale für Heizung und Kühlung abhängig von Außentemperatur, Datum und Uhrzeit. Mit dem Eingang T_EXT_Setup wird festgelegt, wie der Ausgangswert T_EXT ermittelt wird. Wird T_EXT_Setup nicht beschaltet, so ist der Vorgabewert 0. Die Setup-Werte T_EXT_MIN

und T_EXT_Max legen den Mindestwert und Maximalwert der Außentemperatureingänge fest. Werden diese Grenzen über- oder unterschritten, so wird von einem Fehler im Sensor oder Drahtbruch ausgegangen und an Stelle des Messwertes der Vorgabewert T_EXT_DEFAULT benutzt.

T_EXT_Setup	T_EXT
0	Durchschnittswert von T_EXT1, T_ext2 und T_ext3
1	T_EXT1
2	T_EXT2
3	T_EXT3
4	T_EXT_DEFAULT
5	Niedrigster Wert der 3 Eingänge
6	Höchster Wert der 3 Eingänge
7	Mittlerer Wert der 3 Eingänge

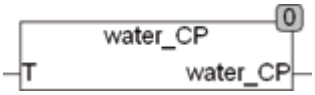
Mit den Setup-Variablen HEAT_PERIOD und COOL_PERIOD wird definiert, wann Heizen und wann Kühlen erlaubt ist. Die Entscheidung, ob der Ausgang HEAT oder COOL TRUE wird hängt weiterhin von den Setup-Werten HEAT_START- HEAT_STOP und COOL_START und COOL_STOP ab. Diese Werte können separat für Tag und Nacht definiert werden. Der Start für eine Tag- und Nachtperiode kann durch die Setup-Variablen START_DAY und START_NIGHT festgelegt werden. Ein Variable CYCLE_TIME legt fest, wie oft die Außentemperatur abgefragt werden soll.

23.54. WATER_CP

Type Funktion : REAL

Input T : REAL (Temperatur des Wassers in °C)

Output REAL (Spezifische Wärmekapazität bei der Temperatur T)



WATER_CP berechnet die spezifische Wärmekapazität von flüssigem Wassers in Abhängigkeit von der Temperatur bei Normaldruck. Die Berechnung

ist gültig im Temperaturbereich von 0 bis 100 Grad Celsius und wird in Joule / (Gramm * Kelvin) errechnet. Die Temperatur T wird in °C angegeben.

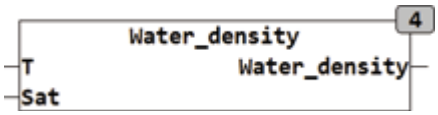
23.55. WATER_DENSITY

Type Funktion : REAL

Input T : REAL (Temperatur des Wassers)

SAT : BOOL (TRUE, wenn das Wasser mit Luft gesättigt ist)

Output REAL (Dichte des Wassers in Gramm / Liter)



WATER_DENSITY berechnet die Dichte von flüssigem Wassers in Abhängigkeit von der Temperatur bei Normaldruck. Die Temperatur T wird in °C angegeben. Die höchste Dichte erreicht Wasser bei 3,983 °C mit 999,974950 Gramm pro Liter. WATER_DENSITY berechnet die Dichte für flüssiges Wasser, nicht für gefrorenes oder verdampftes Wasser. WATER_DENSITY berechnet die Dichte von luftfreiem Wasser wenn SAT = FALSE und von luftgesättigtem Wasser wenn SAT = TRUE. Die Berechneten Werte werden nach einer Näherungsformel berechnet und liefert Werte mit einer Genauigkeit besser las 0,01% im Temperaturbereich von 0 - 100 °C bei einem konstanten Druck von 1013 mBar.

Die Abweichung der Dichte von mit Luft gesättigtem Wasser wird nach der Formel vom Bignell korrigiert.

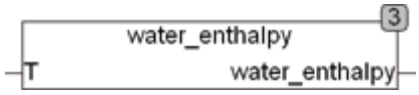
Die Abhängigkeit der Wasserdichte vom Druck ist verhältnismäßig gering mit ca. 0,046 kg/m³ je 1 Bar Druckerhöhung Im Bereich bis 50 Bar. Die geringe Druckabhängigkeit hat in praktischen Anwendungsfällen keinen nennenswerten Einfluss.

23.56. WATER_ENTHALPY

Type Funktion : REAL

Input T : REAL (Temperatur des Wassers)

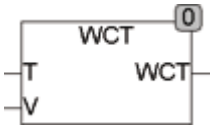
Output REAL (Enthalpie des Wassers in J/Gramm bei der Temperatur T)



WATER_ENTHALPY berechnet die Enthalpie (Wärmeinhalt) von flüssigem Wasser in Abhängigkeit von der Temperatur bei Normaldruck. Die Temperatur T wird in °C angegeben. Die Berechnung ist gültig für eine Temperatur von 0 bis 100 °C und das Ergebnis ist die Wärmemenge die benötigt wird um das Wasser von 0°C auf die Temperatur von T zu Erwärmen. Das Ergebnis wird in Joule/Gramm J/g Beziehungsweise KJ/Kg ausgegeben. Die Berechnung erfolgt durch lineare Interpolation in Schritten von 10° und erreicht damit eine für nicht wissenschaftliche Anwendungen hinreichende Genauigkeit. Eine mögliche Anwendung von WATER_ENTHALPY ist die Berechnung der Energiemenge die benötigt wird um zum Beispiel einen Pufferspeicher um X (T2 – T1) Grad zu erwärmen. Aus der benötigten Energie kann dann die Laufzeit eines Heizkessels berechnet werden und exakt die benötigte Energie bereitgestellt werden. Da Temperaturmesswerte in der Praxis stark zeitverzögert vorliegen ist mit dieser Methode eine bessere Aufheizung möglich.

23.57. WCT

Type	Funktion : REAL
Input	T : REAL (Außentemperatur in °C)
	V : REAL (Windgeschwindigkeit in km/h)
Output	REAL (Windchill Temperatur)

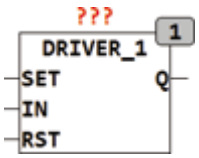


WCT berechnet die Windchill-Temperatur abhängig von der Windgeschwindigkeit in Km/h und der Außentemperatur in °C. Die Windchill-Temperatur ist nur definiert für Windgeschwindigkeiten größer als 5 Km/h und Temperaturen kleiner 10 °C. Für Werte außerhalb des definierten Bereichs wird die Eingangstemperatur ausgegeben.

24. Automation

24.1. DRIVER_1

Type	Funktionsbaustein
Input	SET : BOOL (asynchroner Set Eingang)
	IN : BOOL (Schalteingang)
	RST : BOOL (asynchroner Reset Eingang)
Setup	TOGGLE_MODE : BOOL (Mode des Eingangs IN)
	TIMEOUT : TIME (Maximale Einschaltdauer der Ausgänge)
Output	Q0 : BOOL (Schaltausgang)

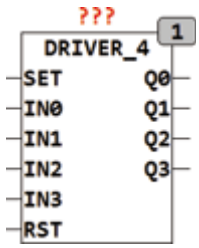


DRIVER_1 ist ein Treiberbaustein dessen Ausgang Q durch den Eingang IN wenn TOGGLE_MODE = FALSE gesetzt werden kann. Der Ausgang bleibt dann solange auf TRUE bis er entweder durch einen asynchronen Reset (RST) auf FALSE gesetzt wird oder bis die maximale Schaltzeit (TIMEOUT) abgelaufen ist. Weitere Impulse am Eingang IN verlängern dabei die TRUE Phase am Ausgang indem mit jeder steigenden Flanke an IN der Timeout von Neuem beginnt. Wenn TOGGLE_MODE = TRUE, dann schaltet der Ausgang Q mit jeder steigenden Flanke an IN den Zustand zwischen TRUE und FALSE. Auch im TOGGLE_MODE wird mit TIMEOUT die maximale TRUE Phase am Ausgang Q begrenzt. Wird TIMEOUT auf T#0s gesetzt (Default) dann ist kein Timeout aktiv. Die asynchronen SET und RST Eingänge setzten den Ausgang Q auf TRUE oder FALSE. Der Baustein DRIVER_4 stellt bei gleicher Funktionalität 4 Schaltausgänge zur Verfügung.

24.2. DRIVER_4

Type	Funktionsbaustein
Input	SET : BOOL (asynchroner Set Eingang)
	IN0..IN3 : BOOL (Schalteingänge)
	RST : BOOL (asynchroner Reset Eingang)

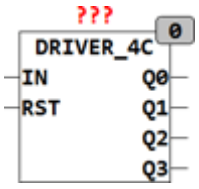
Setup	TOGGLE_MODE : BOOL (Mode des Eingangs IN)
	TIMEOUT : TIME (Maximale Ontime der Ausgänge)
Output	Q0 .. Q3 : BOOL (Schaltausgänge)



DRIVER_1 ist ein Treiberbaustein dessen Ausgänge Q durch die Eingänge IN geschaltet werden. eine detaillierte Beschreibung des Bausteins befindet sich unter DRIVER_1. DRIVER_4 stellt im Gegensatz zu DRIVER_1 4 Schaltausgänge zur Verfügung, hat aber ansonsten die gleiche Funktionalität.

24.3. DRIVER_4C

Type	Funktionsbaustein
Input	IN : BOOL (Schalteingang)
	RST : BOOL (asynchroner Reset Eingang)
Setup	TIMEOUT : TIME (Maximale Einschaltzeit des Bausteins)
	SX : ARRAY[1..7] OF BYTE := 1,2,4,8,0,0,0;
	(Voreinstellung der Schaltsequenz)
Output	Q0 .. Q3 : BOOL (Schaltausgänge)



DRIVER_4C ist ein Treiberbaustein dessen Ausgangszustände mit einer steigenden Flanke an IN geschaltet werden. Die Ausgangszustände sind im Setup Array SX vordefiniert und können vom Anwender jederzeit geändert werden. Im Array SX[1..6] sind die Ausgangszustände für jeden Schaltzustand SN einzeln Bitweise definiert. Bit 0 eines Elements schaltet Q0, Bit1 schaltet Q1, Bit2 Q2 und Bit3 Q3, die oberen 4 Bits werden jeweils ignoriert. Das Array ist Vorbelegt mit Bit0 = TRUE für SN = 1, Bit1 für SN =

2. Bit2 für SN = 3 und Bit3 für SN = 4. Somit wird am Ausgang die Sequenz (0000,0001,0010,0100,1000,0000) für (Q3,Q2,Q1,Q0) durchlaufen. Ist das Element SX[SN] des Arrays 0 so springt SN automatisch auf 0 zurück, so dass ein leeres Element die Sequenz abbricht. Bei Ablauf des Timeout springt der Baustein automatisch in den Zustand SN = 0 zurück. Der Timeout ist nur dann aktiv wenn die Variable TIMEOUT > t#0s ist.

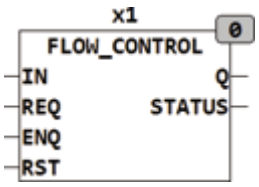
Beispiel:

SX = 1,3,7,15,7,3,1 erzeugt folgende Sequenz:

Q3,Q2,Q1,Q0 = 0000,0001,0011,0111,1111,0111,0011,0001,0000,.....

24.4. FLOW_CONTROL

Type	Funktionsbaustein
Input	IN : BOOL (Steuereingang)
	REQ : BOOL (Request für Automatikmodus)
	ENQ : BOOL (Enable für Ausgang Q)
	RST : BOOL (asynchroner Reset Eingang)
Setup	T_AUTO : TIME (Ventil Einschaltzeit im Automatikmodus)
	T_DELAY : TIME(Ventil Disable Zeit im Automatikmodus)
Output	Q : BOOL (Schaltausgang für Ventil)
	STATUS : BYTE (ESR kompatibler Statusausgang)



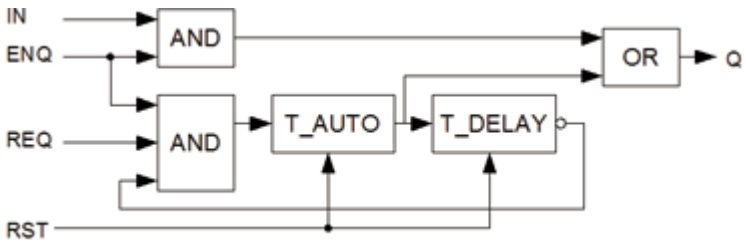
FLOW_CONTROL Schaltet ein Ventil am Ausgang Q wenn der Eingang IN = TRUE. Zusätzlich kann das Ventil auch über den Eingang REQ geschaltet werden, Wenn REQ = TRUE schaltet das Ventil für die Zeit T_AUTO ein und wird dann für die Zeit T_DELAY gesperrt. nach Ablauf der Zeit T_DELAY kann über REQ das Ventil wieder Eingeschaltet werden. Während der Sperrzeit T_DELAY kann das Ventil jedoch über den Eingang IN gesteuert werden. Ein ESR kompatibler Status Ausgang STATUS signalisiert den Zustand des Bausteins. Sowohl REQ als auch IN können den Ausgang Q nur dann schalten wenn der Eingang ENQ auf TRUE steht.

Status = 100	Betriebsbereit
Status = 101	Ventil ein durch TRUE an IN

Status = 102 Ventil ein durch TRUE an REQ

Status = 103 Reset wurde ausgeführt

Das folgernde Schema verdeutlicht den Aufbau von FLOW_CONTROL:



24.5. FT_PROFILE

Type Funktionsbaustein

Input K : REAL (Multiplikator)
O : REAL (Offset)
M : REAL (Zeitmultiplikator)
E : BOOL (Startsignal)

Output Y : REAL (Signalausgang)
RUN : BOOL (TRUE, wenn Ausgangssignal erzeugt wird)
ET : TIME (Zeit seit Start des Ausgangsprofils)

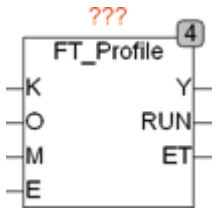
Setup VALUE_0 : REAL (Ausgangswert des Ausgangs zum Start)
TIME_1 : TIME (Zeitpunkt wenn die Rampe VALUE_1 erreicht)
VALUE_1 : REAL (Wert der Rampe zum Zeitpunkt TIME_1)
TIME_2 : TIME (Zeitpunkt wenn die Rampe VALUE_2 erreicht)
VALUE_2 : REAL (Wert der Rampe zum Zeitpunkt TIME_2)
TIME_3 : TIME (Zeitpunkt wenn die Rampe VALUE_3 erreicht)
VALUE_3 : REAL (Wert der Rampe zum Zeitpunkt TIME_3)
TIME_10 : TIME (Zeitpunkt wenn die Rampe VALUE_10 erreicht)
VALUE_10 : REAL (Wert der Rampe zum Zeitpunkt TIME_10)
TIME_11 : TIME (Zeitpunkt wenn die Rampe VALUE_11 erreicht)
VALUE_11 : REAL (Wert der Rampe zum Zeitpunkt TIME_11)

TIME_12 : TIME (Zeitpunkt wenn die Rampe VALUE_12 erreicht)

VALUE_12 : REAL (Wert der Rampe zum Zeitpunkt TIME_12)

TIME_13 : TIME (Zeitpunkt wenn die Rampe VALUE_13 erreicht)

VALUE_13 : REAL (Wert der Rampe zum Zeitpunkt TIME_13)



FT_PROFILE erzeugt ein zeitabhängiges Ausgangssignal. Das Ausgangssignal wird definiert durch Zeit – Wertepaare. FT_PROFILE erzeugt ein Ausgangssignal Y, indem die Wertepaare durch Rampen verbunden werden. Ein typische Anwendung für FT_PROFILE ist die Erzeugung eines Temperaturprofils für einen Glühofen, aber auch jede Anwendung die ein zeitabhängiges Steuersignal benötigt stellt ein Anwendungsgebiet dar. Das zeitabhängige Ausgangssignal wird durch eine steigende Flanke an E gestartet und läuft dann selbsttätig ab. Nach dem Wertepaar (TIME_10, VALUE_10) verharrt das Ausgangssignal auf VALUE_10, bis der Eingang E FALSE wird. Mit einer Flanke an E kann also das Signal gestartet werden und zusätzlich kann der Eingang E auch benutzt werden um das Signal beliebig lange zu strecken. Dadurch ist es möglich einen Verlauf bis zum Wert VALUE_3 zu erzeugen, mit E zu strecken und nach der fallenden Flanke von E wieder einen Verlauf zurück zum Ausgangswert zu erzeugen. Mit den Eingängen K, O und M kann das Ausgangssignal dynamisch gestreckt und skaliert werden.

Y = erzeugter Wert * K + O

Der Eingang M dient zum Strecken des Signals über die Zeit. Der tatsächliche Zeitverlauf entspricht dem über Setup definierten Zeitverlauf multipliziert mit M. Um geradlinige Rampen zu gewährleisten wirkt eine Zeitstreckung durch M nur nach Abschluss einer Flanke. Der Ausgang RUN wird mit einer steigenden Flanke von E auf TRUE gesetzt und wird erst nach Abschluss des Zeitprofils wieder FALSE. Am Ausgang ET kann die seit Start verstrichene Zeit abgelesen werden.

Die folgenden Graphiken zeigen das Ausgangssignal für die Wertepaare:

VALUE_0 = 0

TIME_1, VALUE_1 = 1s, 50

TIME_2, VALUE_2 = 3s, 50

TIME_3, VALUE_3 = 4s, 100

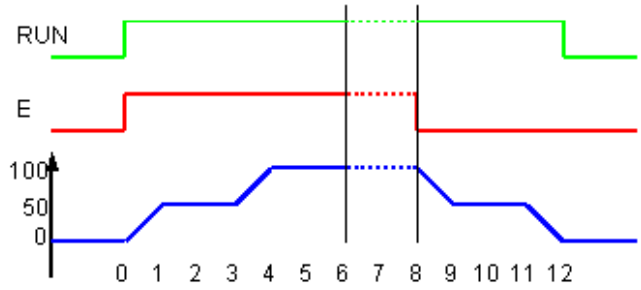
TIME_10, VALUE_10 = 6s, 100

TIME_11, VALUE_11 = 7s, 50

TIME_12, VALUE_12 = 9s, 50

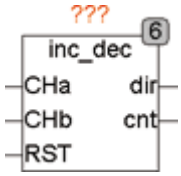
TIME_13, VALUE_13 = 10s, 0

Die Schaubilder stellen das Ausgangssignal sowohl mit gestreckter Phase 3 durch E dar, als auch ohne Streckung.



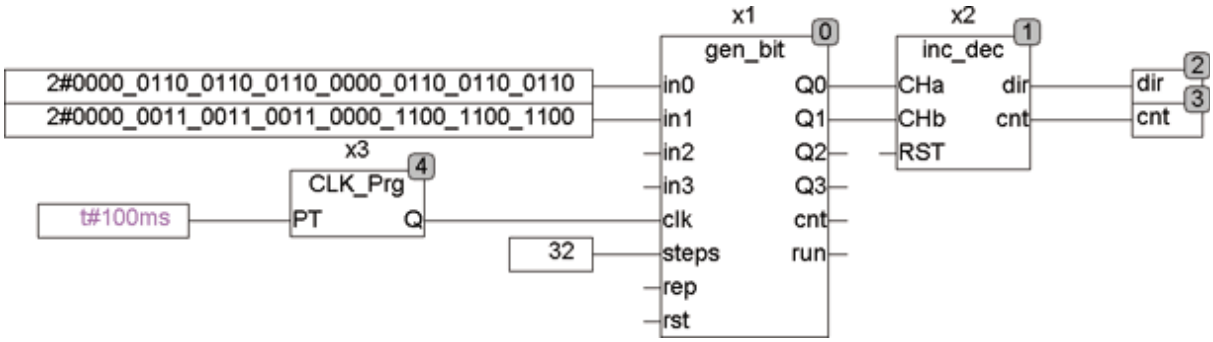
24.6. INC_DEC

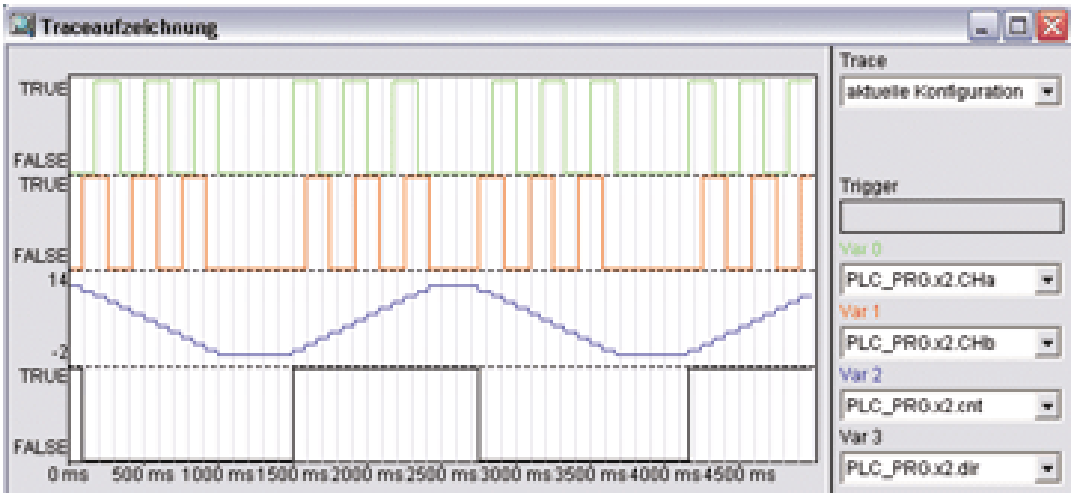
Type	Funktionsbaustein
Input	CHA : BOOL (Kanal A des Gebers)
	CHB : BOOL (Kanal B des Gebers)
	RST : BOOL (Reset)
Output	DIR : BOOL (Drehrichtung)
	CNT : INT (Zählerwert)



INC_DEC ist ein Decoder für Inkrementalgeber. Inkrementalgeber (Drehgeber) liefern 2 überlappende Impulse, Kanal A und Kanal B. Aus den beiden Kanälen wird die Drehrichtung und der Drehwinkel dekodiert. INC_DEC detektiert jede einzelne Flanke des Drehgebers, sodass 4-fache Auflösung erreicht wird. Der Ausgang DIR zeigt die Drehrichtung an, und am Ausgang CNT steht ein Integer-Wert bereit, der die Anzahl der gezählten Impulse ausgibt. Für eine volle Umdrehung eines Drehgebers mit 100 Impulsen zählt CNT bis 400, weil an jeder Flanke beider Kanäle gezählt wird, sodass 4-fache Auflösung erreicht wird. Ein RST Eingang erlaubt jederzeit den Zähler auf 0 zu stellen. Der Zähler zählt aufwärts, wenn DIR = TRUE und abwärts, wenn DIR = FALSE.

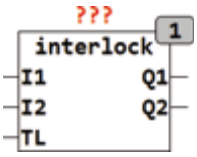
Im Folgenden Beispiel wird ein Bitmustergenerator GEN_BIT benutzt um einen Drehgeber zu simulieren, der immer abwechselnd 3 Schritte im Uhrzeigersinn und 3 im Gegenuhrzeigersinn macht. In der Traceaufzeichnung ist zu sehen, wie der INC_DEC die Bewegung in 12 Zahlschritte zerlegt und die Richtung dekodiert.





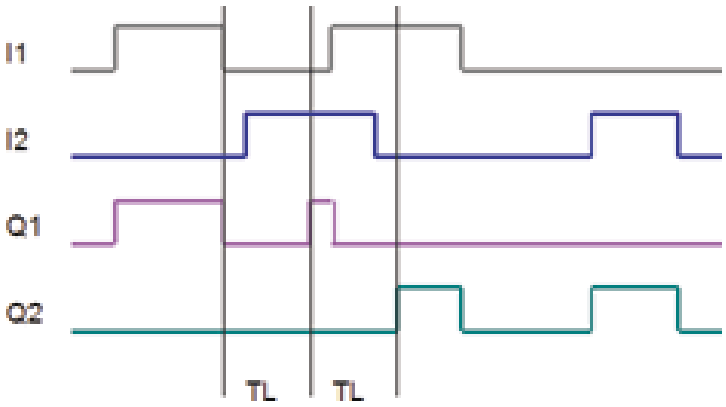
24.7. INTERLOCK

Type	Funktionsbaustein
Input	I1 : BOOL (Eingang 1)
	I2 : BOOL (Eingang 2)
	TL : TIME (Totzeit)
Output	Q1 : BOOL (Ausgang 1)
	Q2 : BOOL (Ausgang 2)



Der Baustein INTERLOCK hat 2 Eingänge I1 und I2 die jeweils die Ausgänge Q1 und Q2 schalten. Q1 und Q2 sind jedoch gegenseitig verriegelt so dass immer nur ein Ausgang auf TRUE stehen kann. Die Zeit TL legt eine Totzeit zwischen den Beiden Ausgängen fest. Ein Ausgang kann nur dann TRUE werden wenn der andere Ausgang für mindestens für die Zeit TL FALSE war.

I1	I2	Q1	Q2
0	0	0	0
0	1	0	1
1	0	1	0
1	1	0	0

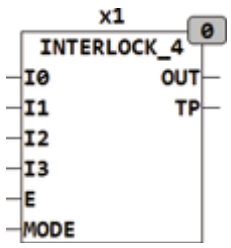


24.8. INTERLOCK_4

Type	Funktionsbaustein
Input	I0 : BOOL (Eingangssignal 0)
	I1 : BOOL (Eingangssignal 1)
	I2 : BOOL (Eingangssignal 2)
	I3 : BOOL (Eingangssignal 3)
	E : BOOL (Enable Eingang)
	MODE : INT (Betriebsmodus)

Output OUT : BOOL (Ausgangssignal)

TP : BOOL (TRUE wenn sich Ausgang verändert hat)



INTERLOCK_4 packt die 4 Eingangswerte I0..I3 in die Bits (0..3) des Ausgangs OUT. Bei jeder Veränderung des Ausgangs wird der Ausgang TP für einen Zyklus TRUE damit weitere Bausteine zur Verarbeitung getriggert werden können. Ist der Eingang E = FALSE bleiben alle Ausgänge auf 0 bzw. FALSE. Der Eingang MODE stellt verschiedene Betriebsmode des Bausteins ein.

MODE	Bedeutung
0	Eingänge werden direkt im Ausgangsbyte dargestellt z.B. I0, I2 = TRUE OUT = 2#0000_0101
1	Nur der Eingang mit der höchsten Eingangsnummer wird ausgegeben, die anderen werden ignoriert. z.B. I0,I1,I2 = TRUE: OUT = 2#0000_0100
2	Nur der zuletzt aktivierte Eingang wird ausgegeben.
3	Ein aktivierter Eingang disabled alle anderen Eingänge.

24.9. MANUAL

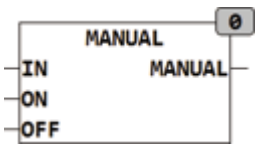
Type Funktion : BOOL

Input IN : BOOL (Eingangssignal)

ON : BOOL (Handbetrieb Ein)

OFF : BOOL (Handbetrieb Aus)

Output BOOL (Ausgangssignal)

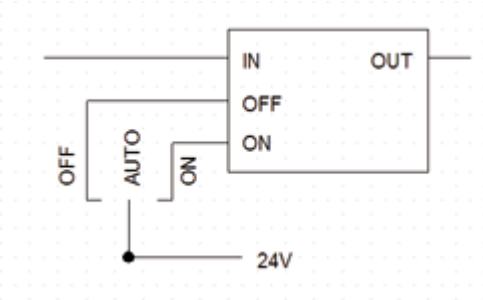


MANUAL kann ein Eingangssignal IN mit TRUE oder mit FALSE überschreiben.

IN	ON	OFF	Q	
0	0	0	0	
1	0	0	1	
-	-	1	0	Handbetrieb Stellung AUS
-	1	0	1	Handbetrieb Stellung EIN

Die typische Verwendung von MANUAL ist mittels eines Schalters mit 3 Stellungen (AUS, AUTO, EIN) wobei die Anschlüsse AUS auf OFF und EIN auf ON geschaltet werden und AUTO des Schalters offen bleibt.

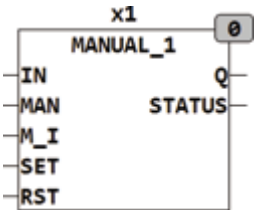
Das folgende Schema zeigt die mögliche Anschaltung eines Schalters mit 3 Stellungen:



24.10. MANUAL_1

Type	Funktionsbaustein
Input	IN : BOOL (Eingangssignal)
	MAN : BOOL (Umschaltung auf Handbetrieb)
	M_I : BOOL (Signalpegel bei Handbetrieb)
	SET : BOOL (Asynchroner Set bei Handbetrieb)
	RST : BOOL (Asynchroner Reset bei Handbetrieb)
Output	Q : BOOL (Ausgangssignal)
	STATUS : BYTE (ESR kompatibler Status Ausgang)

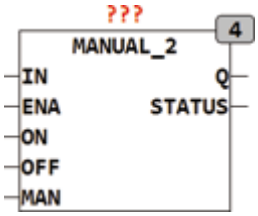
MANUAL_1 kann ein digitales Signal im Handbetrieb überschreiben. Solange MAN = FALSE folgt der Ausgang Q Eingang direkt dem Eingang IN. So-



bald MAN = TRUE wird folgt der Ausgang dem Zustand des Eingangs M_I. Mit den Eingängen SET und RST kann im Handbetrieb ein asynchrones Setzen und Löschen des Ausgangs erzeugt werden. SET und RST sind nur während des Handbetriebs aktiv. Wird im Handbetrieb an SET oder RST eine steigende Flanke registriert, so folgt der Ausgang nicht mehr dem Eingang M_I sondern bleibt auf dem Zustand den eine steigende Flanke an SET (Ausgang = TRUE) beziehungsweise RST (Ausgang = FALSE) erzeugt hat. Sobald der Eingang MAN wieder auf FLASE geht folgt der Ausgang Q wieder dem Eingang IN.

24.11. MANUAL_2

Type	Funktionsbaustein
Input	IN : BOOL (Eingangssignal)
	ENA : BOOL (Baustein Enable Eingang)
	ON : BOOL (Zwingt den Ausgang auf TRUE)
	OFF : BOOL (Zwingt den Ausgang auf FALSE)
	MAN : BOOL (Ausgangszustand bei Manualbetrieb)
Output	Q : BOOL (Ausgangssignal)
	STATUS : BYTE (ESR kompatibler Status Ausgang)



MANUAL_2 kann ein digitales Signal überschreiben und schaltet zwischen Hand und Automatikbetrieb um. der Baustein ist so ausgelegt das ein 3-stufiger Schalter zwischen Aus, Automatik und Ein schaltet. Das Signal im Automatikzustand wird auf IN gelegt, im Falle von erzwungenem Aus wird OFF auf TRUE gelegt und im Falle von erzwungenem Ein wird ON auf TRUE gelegt. die Beiden Eingänge ON und OFF auf FALSE schalten den Eingang IN direkt auf den Ausgang Q. werden jedoch beide Eingänge ON und OFF gleichzeitig auf TRUE gesetzt, so wird der Zustand des Eingangs MAN auf den Ausgang geschaltet. Der Eingang MAN kann auch dazu benutzt wer-

den um einen Vorrang für ON oder OFF zu definieren, den der Wert von MAN wird immer dann auf den Ausgang gelegt wenn beide Eingänge ON und OFF gleichzeitig auf TRUE stehen. Ist der Eingang ENA auf FALSE so bleibt der Ausgang immer auf FALSE, der Baustein ist disabled. Die folgende Tabelle definiert die Betriebszustände des Bausteins. Der Ausgang STATUS ist ESR kompatibel und meldet die Zustände des Bausteins an entsprechende ESR Bausteine weiter.

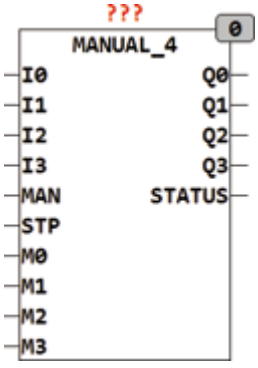
IN	ENA	ON	OFF	MAN	Q	STATUS	
-	L	-	-	-	L	104	Disabled
X	H	L	L	-	X	100	Auto Mode
-	H	H	L	-	H	101	force High
-	H	L	H	-	L	102	force Low
-	H	H	H	X	X	103	Manual Input
-	H	H	H	L	L	103	Force with Priority for OFF
-	H	H	H	H	H	103	Force with Priority for ON

24.12. MANUAL_4

Type Funktionsbaustein

Input I0..I3 : BOOL (Eingangssignale)
MAN : BOOL (Umschaltung auf Handbetrieb)
M0..M3 : BOOL (Eingangssignale bei Handbetrieb)
STP : BOOL (Asynchroner Step bei Handbetrieb)

Output Q0..Q3 : BOOL (Ausgangssignale)
STATUS : BYTE (ESR kompatibler Status Ausgang)

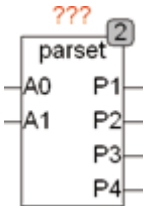


MANUAL_4 kann 4 digitale Signal im Handbetrieb überschreiben. Solange MAN = FALSE folgen die Ausgänge Q direkt den Eingängen I. Sobald MAN = TRUE wird folgen die Ausgänge den Zuständen der Eingänge M. Mit dem Eingang STP kann im Handbetrieb ein rotierendes Setzen der Ausgänge erzeugt werden. STP ist nur während des Handbetriebs aktiv. Wird im Handbetrieb an STP eine steigende Flanke registriert, so folgen die Ausgänge nicht mehr den Eingängen MX sondern werden mit STP zyklisch durchgeschaltet. Bei der ersten steigenden Flanke an STP wird nur der Ausgang Q0 aktiv und bei der nächsten Flanke an STP schaltet der Baustein auf den Ausgang Q1 und so weiter. Sobald der Eingang MAN wieder auf FALSE geht, folgen die Ausgänge Q wieder den Eingängen I. Der ESR kompatible Status Ausgang meldet Schaltzustände weiter.

STATUS	Zustand
100	Automatic Mode MAN = FALSE, Q0 = I0, Q1 = I1, Q2 = I2, Q3 = I3
101	Manual Mode MAN = TRUE, Q0 = M0, Q1 = M1, Q2 = M2, Q3 = M3
110,111,112,113	Step Mode for Output Q0, Q1, Q2, Q3

24.13. PARSET

Type	Funktionsbaustein
Input	A0 : BOOL (Auswahl Eingang 0)
	A1 : BOOL (Auswahl Eingang 1)
Setup	X01, X11, X21, X31 : REAL (Werte für Parameter P1)
	X02, X12, X22, X32 : REAL (Werte für Parameter P2)
	X03, X13, X23, X33 : REAL (Werte für Parameter P3)
	X04, X14, X24, X34 : REAL (Werte für Parameter P4)
	TC : TIME (Rampenzeit zu einem neuen Wert an den)
Output	P1 : REAL (Parameter 1 Ausgang)
	P2 : REAL (Parameter 2 Ausgang)
	P3 : REAL (Parameter 3 Ausgang)
	P4 : REAL (Parameter 4 Ausgang)

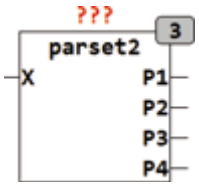


PARSET wählt aus bis zu 4 Parametersätzen je einen aus und liefert die Werte an den Ausgängen P1 bis P4. Die Werte für die Parametersätze werden mit Setup Variablen definiert. Wird die Setup Variable TC auf einen Wert > 0 gesetzt, so schalten die Ausgänge nicht sprunghaft auf einen neuen Wert, sondern laufen in einer Rampe auf den neuen Wert zu so dass der Endwert nach der Zeit TC erreicht wird. Dies erlaubt den weichen Übergang zwischen verschiedenen Parametersätzen. Die Auswahl der Parameter wird durch die Eingänge A0 und A1 gesteuert.

A1,A0	P1	P2	P3	P4
00	X01	X02	X03	X04
01	X11	X12	X13	X14
10	X21	X22	X23	X24
11	X31	X32	X33	X34

24.14. PARSET2

Type	Funktionsbaustein
Input	X : REAL (Signaleingang)
Setup	X01, X11, X21, X31 : REAL (Werte für Parameter P1)
	X02, X12, X22, X32 : REAL (Werte für Parameter P2)
	X03, X13, X23, X33 : REAL (Werte für Parameter P3)
	X04, X14, X24, X34 : REAL (Werte für Parameter P4)
	L1, L2, L3 : REAL (Limits für die Parameterumschaltung
	TC : TIME (Rampenzeit an den Ausgängen P)
Output	P1 : REAL (Parameter 1 Ausgang)
	P2 : REAL (Parameter 2 Ausgang)
	P3 : REAL (Parameter 3 Ausgang)
	P4 : REAL (Parameter 4 Ausgang)

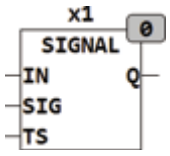


PARSET2 wählt aus bis zu 4 Parametersätzen je einen aus und liefert die Werte an den Ausgängen P1 bis P4. Die Werte für die Parametersätze werden mit Setup Variablen definiert. Wird die Setup Variable TC auf einen Wert > 0 gesetzt, so schalten die Ausgänge nicht sprunghaft auf einen neuen Wert, sondern laufen in einer Rampe auf den neuen Wert zu so dass der Endwert nach der Zeit TC erreicht wird. Dies erlaubt den weichen Übergang zwischen verschiedenen Parametersätzen. Die Auswahl der Parameter wird durch die Steuergröße X und die Schaltschwellen L1 bis L3 festgelegt.

X	P1	P2	P3	P4
X < L1	X01	X02	X03	X04
L1 < X < L2	X11	X12	X13	X14
L2 < X < L3	X21	X22	X23	X24
X >= L3	X31	X32	X33	X34

24.15. SIGNAL

Type	Funktionsbaustein
Input	IN : BOOL (Freigabeeingang)
	SIG : BYTE (Bitpattern)
	TS : TIME (Schaltzeit)
Output	Q : BOOL (Ausgangssignal)

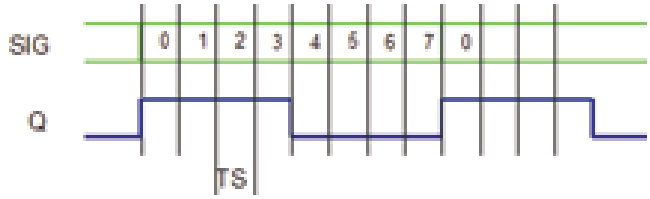


SIGNAL erzeugt ein Ausgangssignal Q das dem Bitpattern in SIG entspricht. Das Bitpattern wird in TS langen Schritten Ausgegeben. Durch verschiedene Bitmuster in SIG können verschiedene Ausgangssignale erzeugt werden. Wird der Eingang IN auf TRUE geschaltet beginnt der Baustein den Ausgang Q entsprechend dem in SIG bereitgestellten Bitpattern Ein-

zuschalten. Durch die Anpassung des Bitpattern können verschiedene Ausgangssignale erzeugt werden. Ein Pattern von 10101010 erzeugt eine Ausgangssignal mit 50% Duty Cycle und einer Frequenz die 1 / 2*S entspricht. Ein Pattern von 11110000 erzeugt hingegen ein Ausgangssignal von 50% DC und einer Frequenz von 1 / 8*TS. Der Start eines Ausgangssignals ist zufällig. Die Bitsequenz beginnt ab einem Beliebigen Bit sobald der Eingang IN auf TRUE geht. Wird am Eingang TS keine Zeit vorgegeben so verwendet der Baustein intern eine Vorgabe von 1024ms je Zyklus (ein Zyklus ist der Durchlauf aller 8 Bits einer Sequenz). Typische Anwendungen für SIGNAL ist die Signalerzeugung für Sirenen oder Signallampen.

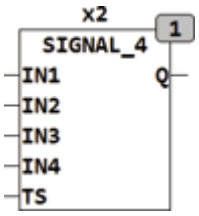
Die folgende Grafik verdeutlicht die Funktionsweise von SIGANL für

SIG = 2#1111_0000:



24.16. SIGNAL_4

Type	Funktionsbaustein
Input	IN1..IN4 : BOOL (Eingang für Bitpattern S1..S4)
	TS : TIME (Schaltzeit)
Setup	S1 .. S4 : BYTE (Bitpattern S1 .. S4)
Output	Q : BOOL (Ausgangssignal)



SIGNAL_4 erzeugt ein Ausgangssignal Q das einem von 4 Bitpattern (S1 .. S4) entspricht. Das Bitpattern wird in TS langen Schritten Ausgegeben. Die Eingänge IN1..IN4 sind priorisierte Eingänge. Ein TRUE an IN1 überschreibt allen anderen Eingänge, IN2 überschreibt IN3 und IN4 hat die niedrigste Priorität. Eine weiterführende Beschreibung der Funktion von SIGNAL_4 befindet sich unter SIGNAL. Die 4 verschiedenen Bitpattern sind in Setup Variablen S1 .. S4 abgelegt und können vom Anwender jederzeit angepasst werden.

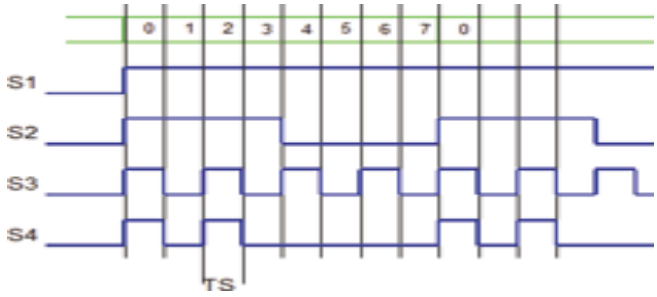
Der Baustein hat per folgende Bitpattern voreingestellt, welche aber vom Anwender bei Bedarf verändert werden können:

S1 = 2#1111_1111

S2 = 2#1111_0000

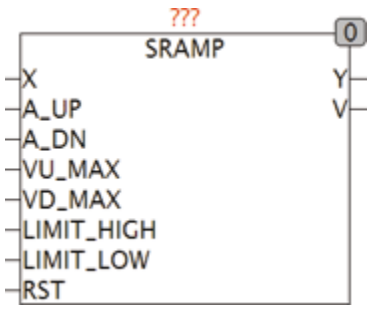
S3 = 2#1010_1010

S4 = 2#1010_0000



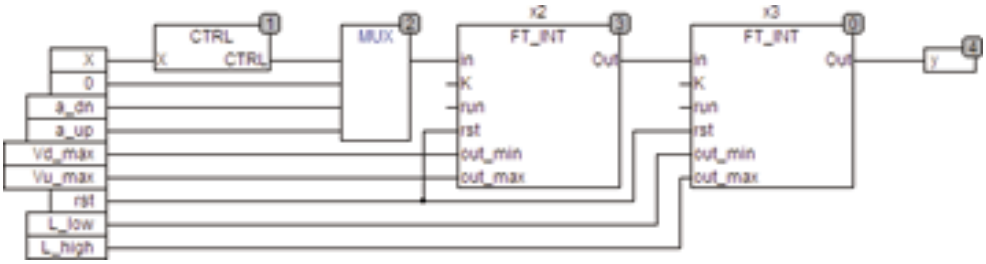
24.17. SRAMP

Type	Funktionsbaustein
Input	X: REAL (Eingangssignal)
	A_UP : REAL (Maximale Beschleunigung Aufwärts)
	A_DN : REAL (Maximale Beschleunigung Abwärts)
	VU_MAX : REAL (Maximale Geschwindigkeit Aufwärts)
	VD_MAX : REAL (Maximale Geschwindigkeit Abwärts)
	LIMIT_HIGH : REAL (Ausgangslimit High)
	LIMIT_LOW : REAL (Ausgangslimit Low)
	RST : BOOL (Asynchroner Reset)
Output	Y : REAL (Ausgangssignal)
	V : REAL (Momentane Geschwindigkeit des Ausgangssignals)



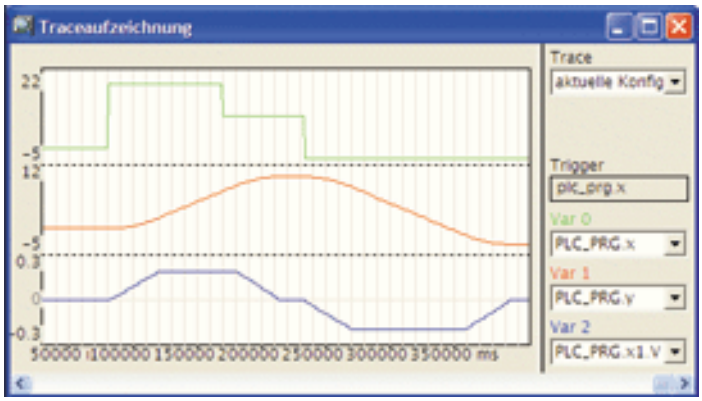
SRAMP erzeugt ein Ausgangssignal das durch einstellbare Parameter begrenzt wird. Das Ausgangssignal folgt dem Eingangssignal und wird dabei durch maximale Geschwindigkeit (VU_MAX und VD_MAX), oberer und unterer Grenzwert (LIMIT_LOW und LIMIT_HIGH), sowie eine maximale Beschleunigung (A_UP und A_DN) begrenzt. SRAMP wird verwendet um zum Beispiel Motoren anzusteuern. Der Ausgang V gibt die momentane Geschwindigkeit des Ausgangs an.

Im folgenden Schema ist die Interne Arbeitsweise von SRAMP ersichtlich. Ein Rampengenerator X2 stellt die Geschwindigkeit mit der sich der Ausgang verändern darf ein und ein Zweiter Rampengenerator X3 steuert den Ausgang.



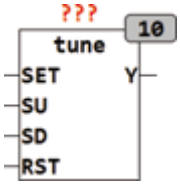
Die Trace Aufzeichnung zeigt ein Beispiel für SRAMP. Der Eingang (grün) steigt von 0 auf 20 und gleich danach auf 10 wobei der Ausgang mit der maximal zulässigen Beschleunigung auf die maximal zulässige Geschwindigkeit ansteigt, anschließend wird verdeutlicht das der Eingang auch während des Signalverlaufs sich ändern kann. In diesem Beispiel wird rechtzeitig wieder abgebremst, so das exakt bei 10 der Ausgang zum Stillstand kommt. Nach erreichen des Endwertes 10 schaltet der Eingang auf -3 und der Ausgang Y folgt entsprechend.

Die Eingangswerte für A_UP und VU_MAX müssen mit positivem Vorzeichen angegeben werden, A_DN und VD_MAX benötigen ein negatives Vorzeichen.



24.18. TUNE

Type	Funktionsbaustein
Input	SET : BOOL (Asynchroner Set Eingang)
	SU, SD : BOOL (Eingänge für Auf und Ab)
	RST : BOOL (Asynchroner Reset Eingang)
Setup	SS : REAL (Schrittweite für kleiner Schritt)
	Limit_L : REAL (unterer Grenzwert)
	Limit_H : REAL (oberer Grenzwert)
	RST_VAL : REAL (Ausgangswert nach Reset)
	SET_VAL : REAL (Ausgangswert nach SET)
	T1 : TIME (Zeit nach der die erste Rampe anläuft)
	T1 : TIME (Zeit nach der die zweite Rampe anläuft)
	S1 : REAL (Geschwindigkeit für erste Rampe)
	S2 : REAL (Geschwindigkeit für zweite Rampe)
Output	Y : REAL (Ausgangssignal)

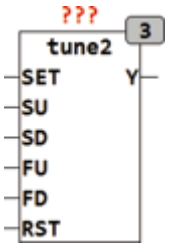


TUNE setzt mithilfe von Auf- und Ab- Tastern ein Ausgangssignal Y. Durch entsprechende Setup Variablen kann die Schrittweite individuell Programmiert werden. Ein oberer und unterer Grenzwert für den Ausgang Y kann mittels LIMIT_L und LIMIT_H vorgegeben werden. mit den Tastern SU und SD werden Schritte Auf oder Ab erzeugt. Wird eine Taste länger als die Zeit T1 gedrückt gehalten, so wird der Ausgang Y kontinuierlich Auf oder Ab verstellt. Die Geschwindigkeit mit der der Ausgang verstellt wird ist hierbei mit S1 vorgegeben. S1 und S2 geben die Einheiten je Sekunde an. Wird eine Taste länger als die Zeit T2 gedrückt gehalten, so schaltet der Baustein automatisch auf eine zweite Geschwindigkeit S2 um. Mit den Eingängen RST und SET kann der Ausgang jederzeit auf einen durch RST_VAL beziehungsweise SET_VAL vorgegebenen Wert gestellt werden.

24.19. TUNE2

Type	Funktionsbaustein

Input	SET : BOOL (Asynchroner Set Eingang)
	SU, SD : BOOL (Eingänge für Auf und Ab in kleinen Schritten)
	FU, FD : BOOL (Eingänge für Auf und Ab in großen Schritten)
	RST : BOOL (Asynchroner Reset Eingang)
Setup	SS : REAL (Schrittweite für kleine Schritte)
	FS : REAL (Schrittweite für große Schritte)
	Limit_L : REAL (unterer Grenzwert)
	Limit_H : REAL (oberer Grenzwert)
	RST_VAL : REAL (Ausgangswert nach Reset)
	SET_VAL : REAL (Ausgangswert nach SET)
	TR : TIME (Zeit nach der die Rampe anläuft)
	S1 : REAL (Geschwindigkeit für kleine Rampe)
	S2 : REAL (Geschwindigkeit für große Rampe)
Output	Y : REAL (Ausgangssignal)



TUNE2 setzt mithilfe von Auf- und Ab- Tastern ein Ausgangssignal Y. Durch entsprechende Setup Variablen kann die Schrittweite für kleine und große Schritte individuell Programmiert werden. Ein oberer und unterer Grenzwert für den Ausgang Y kann mittels LIMIT_L und LIMIT_H vorgegeben werden. mit den Tastern SU und SD werden kleine Schritte Auf oder Ab erzeugt. Die Taster FU und FD erzeugen jeweils große Schritte am Ausgang Y. Wird eine Taste länger als TR gedrückt gehalten, so wird der Ausgang Y kontinuierlich Auf oder Ab verstellt. Die Geschwindigkeiten mit der der Ausgang verstellt wird ist hierbei für die beiden Tastenpaare mit S1 und S2 individuell einstellbar. S1 und S2 geben die Einheiten je Sekunde an. S1 ist die Geschwindigkeit für die Taster SU und SD, und S2 entsprechend für FU und FD. Mit den Eingängen RST und SET kann der Ausgang jederzeit auf einen durch RST_VAL beziehungsweise SET_VAL vorgegebenen Wert gestellt werden.

25. BUFFER Management

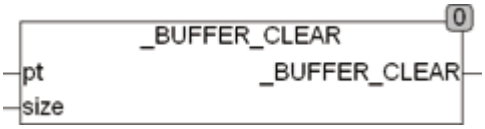
25.1. _BUFFER_CLEAR

Type Funktion : BOOL

Input PT : POINTER TO BYTE (Adresse des Puffers)

SIZE : UINT (Größe des Puffers)

Output BOOL (Returns TRUE)



Die Funktion _BUFFER_CLEAR initialisiert ein beliebiges Array of Byte mit 0. Beim Aufruf wird der Funktion ein Pointer auf das zu initialisierende Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: _BUFFER_CLEAR(ADR(Array), SIZEOF(Array)), wobei Array der Name des zu manipulierenden Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt. Die Funktion liefert nur TRUE zurück. Das durch den Pointer angegebene Array wird direkt im Speicher manipuliert.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel: _BUFFER_CLEAR(ADR(bigarray), SIZEOF(bigarray))
initialisiert bigarray mit 0.

25.2. _BUFFER_INIT

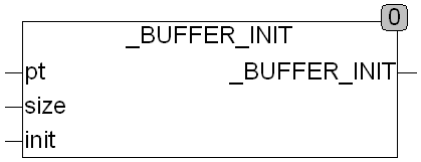
Type Funktion : BOOL

Input PT : POINTER TO BYTE (Adresse des Puffers)

SIZE : UINT (Größe des Puffers)

INIT : BYTE (Initialwert)

Output BOOL (Returns TRUE)



Die Funktion _BUFFER_INIT initialisiert ein beliebiges Array of Byte mit dem Wert INIT. Beim Aufruf wird der Funktion ein Pointer auf das zu initialisierende Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: _BUFFER_INIT(ADR(Array), SIZEOF(Array), INIT), wobei Array der Name des zu manipulierenden Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt. Die Funktion liefert nur TRUE zurück. Das durch den Pointer angegebene Array wird direkt im Speicher manipuliert.

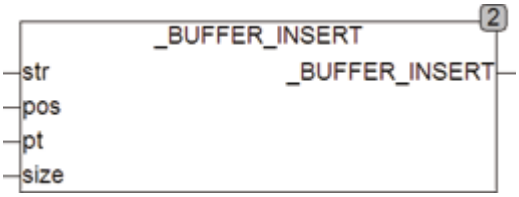
Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel: _BUFFER_INIT(ADR(bigarray), SIZEOF(bigarray),3)

initialisiert bigarray mit 3.

25.3. _BUFFER_INSERT

Type	Funktion : INT
Input	STR : STRING (zu kopierender String)
	POS : INT (Position ab der der String in den Puffer kopiert wird)
	PT : POINTER TO BYTE (Adresse des Puffers)
	SIZE : UINT (Größe des Puffers)
Output	INT (Position in Buffer nach dem eingesetzten String)



Die Funktion _BUFFER_INSERT kopiert einen String in ein beliebiges Array of Byte und Verschiebt den Rest des Array um die Länge des Strings. Der String wird ab einer beliebigen Position POS im Puffer abgelegt. Das erste Element im Array hat die Positionsnummer 0. Beim Aufruf wird der Funktion ein Pointer auf das zu bearbeitende Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: _BUFFER_INSERT(STR, POS,

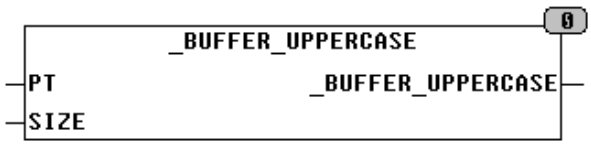
ADR(Array), SIZEOF(Array)), wobei Array der Name des zu manipulierenden Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt. Die Funktion liefert die erste Position im Buffer nach dem eingesetzten String zurück. Das durch den Pointer angegebene Array wird direkt im Speicher manipuliert.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel: _BUFFER_INSERT(STR, POS, ADR(bigarray), SIZEOF(bigarray))

25.4. _BUFFER_UPPERCASE

Type	Funktion : BOOL
Input	PT : POINTER TO BYTE (Adresse des Puffers)
	SIZE : UINT (Größe des Puffers)
Output	BOOL (Returns TRUE)



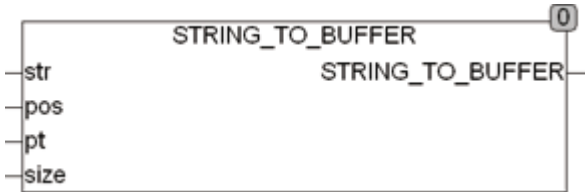
Die Funktion _BUFFER_UPPERCASE interpretiert jedes Byte im Buffer als ASCII Zeichen und wandelt es in Großbuchstaben um. Beim Aufruf wird der Funktion ein Pointer auf das zu initialisierende Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: _BUFFER_INIT(ADR(Array), SIZEOF(Array), INIT), wobei Array der Name des zu manipulierenden Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt. Die Funktion liefert nur TRUE zurück. Das durch den Pointer angegebene Array wird direkt im Speicher manipuliert.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel: _BUFFER_UPPERCASE(ADR(bigarray), SIZEOF(bigarray))

25.5. _STRING_TO_BUFFER

Type	Funktion : INT
Input	STR : STRING (zu kopierender String)
	POS : INT (Position ab der der String in den Puffer kopiert wird)
	PT : POINTER TO BYTE (Adresse des Puffers)
	SIZE : UINT (Größe des Puffers)
Output	INT (Liefert die nächste Position im Buffer nach dem eingesetzten String zurück)



Die Funktion _STRING_TO_BUFFER kopiert einen String in ein beliebiges Array of Byte. Der String wird ab einer beliebigen Position POS im Puffer abgelegt. Das erste Element im Array hat die Positionsnummer 0. Beim Aufruf wird der Funktion ein Pointer auf das zu bearbeitende Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: _STRING_TO_BUFFER(STR, POS, ADR(Array), SIZEOF(Array)), wobei Array der Name des zu manipulierenden Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt. Die Funktion liefert die erste Position im Buffer nach dem eingesetzten String zurück. Das durch den Pointer angegebene Array wird direkt im Speicher manipuliert.

Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel:

_STRING_TO_BUFFER(STR, POS, ADR(bigarray), SIZEOF(bigarray))

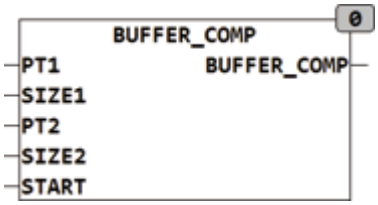
25.6. BUFFER_COMP

Type	Funktion : INT
Input	PT1 : POINTER (Adresse des ersten Puffers)
	SIZE1 : INT (Größe des ersten Puffers)
	PT2 : POINTER (Adresse des zweiten Puffers)

SIZE2 : INT (Größe des zweiten Puffers)

START : INT (Suchbeginn ab Start)

Output INT (gefundene Position)



Die Funktion BUFFER_COMP überprüft ob der Inhalt des Arrays PT2 im Array PT1 ab der Position START vorkommt. Wird PT2 in PT1 gefunden, so gibt die Funktion die Position in PT1 beginnend bei 0 an. Wird PT2 nicht in PT1 gefunden, wird -1 zurückgegeben. BUFFER_COMP kann auch zum Vergleich von 2 gleichgroßen Arrays verwendet werden.

Beim Aufruf wird der Funktion ein Pointer auf das zu bearbeitende Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: BUFFER_COMP(ADR(BUF1), SIZEOF(BUF1), ADR(BUF2), SIZEOF(BUF2)), wobei BUF1 und BUF2 die Namen des zu manipulierenden Arrays sind. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt. Die Funktion liefert nur TRUE zurück. Das durch den Pointer angegebene Array wird direkt im Speicher manipuliert. Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.

Beispiel:

BUFFER_COMP(ADR(BUF1), SIZEOF(BUF1), ADR(BUF2), SIZEOF(BUF2))

25.7. BUFFER_SEARCH

Type Funktion : INT

Input PT : POINTER (Adresse des Puffers)

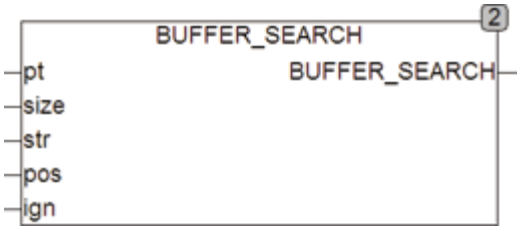
SIZE : UINT (Größe des Puffers)

STR : STRING (Suchstring)

POS : INT (Position ab der gesucht wird)

IGN : BOOL (Search Groß / Kleinschreibung)

Output INT (Position an der die Zeichenkette gefunden wurde)



Die Funktion BUFFER_SEARCH durchsucht ein beliebiges Array of Byte auf den Inhalt einer Zeichenkette und meldet die Position des ersten Zeichens der Zeichenkette im Array wenn eine Übereinstimmung gefunden wird. Der Puffer wird ab einer beliebigen Position POS durchsucht. Das erste Element im Array hat die Positionsnummer 0. Beim Aufruf wird der Funktion ein Pointer auf das zu bearbeitende Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: BUFFER_SEARCH(ADR(Array), SIZEOF(ARRAY), STR, POS, IGN), wobei ARRAY der Name des Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt. Die Funktion liefert die aus dem Puffer kopierte Zeichenkette als STRING zurück. Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen. Wenn IGN = TRUE werden sowohl Groß- als auch Klein-buchstaben als Übereinstimmung gefunden, STR muss dabei in Großbuchstaben vorhanden sein. Wenn IGN = FALSE wird Case sensitiv gesucht.

Beispiel: BUFFER_SEARCH(ADR(Array), SIZEOF(ARRAY), 'FIND', 0, TRUE)

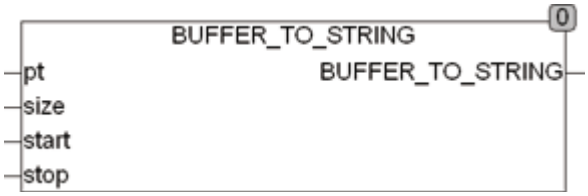
Findet 'FIND', 'Find', 'find' .... im Array.

Beispiel: BUFFER_SEARCH(ADR(Array), SIZEOF(ARRAY), 'FIND', 0, FALSE)

Findet nur 'FIND' im Array.

25.8. BUFFER_TO_STRING

Type	Funktion : STRING
Input	PT : POINTER TO BYTE (Adresse des Puffers)
	SIZE : UINT (Größe des Puffers)
	START : UINT (Position ab der der String aus dem Puffer kopiert wird)
	STOP : UINT (Ende des Strings im Puffer)
Output	STRING (Zeichenkette die aus dem Puffer kopiert wurde)



Die Funktion BUFFER_TO_STRING extrahiert einen String aus einem beliebigen Array of Byte. Der String wird ab einer beliebigen Position START aus dem Puffer kopiert und endet an der Position STOP. Das erste Element im Array hat die Positionsnummer 0. Beim Aufruf wird der Funktion ein Pointer auf das zu bearbeitende Array und dessen Größe in Bytes übergeben. Unter CoDeSys lautet der Aufruf: BUFFER_TO_STRING(ADR(Array), SIZEOF(ARRAY), START, STOP), wobei ARRAY der Name des Arrays ist. ADR ist eine Standardfunktion, die den Pointer auf das Array ermittelt und SIZEOF ist eine Standardfunktion, die die Größe des Arrays ermittelt. Die Funktion liefert die aus dem Puffer kopierte Zeichenkette als STRING zurück. Diese Art der Bearbeitung von Arrays ist äußerst effizient, da kein zusätzlicher Speicher benötigt wird und keine Übergabewerte kopiert werden müssen.
Beispiel: BUFFER_TO_STRING(ADR(Array), SIZEOF(ARRAY), START, STOP)

26. Listen Verarbeitung

26.1. Einleitung

Die hier beschriebenen Listen sind als STRING(LIST_LENGTH) gespeicherte Listen die Elemente der Liste beginnen mit dem Zeichen SEP gefolgt vom Element. Die Elemente können alle in Strings zulässigen Zeichen enthalten und können auch eine Leere Zeichenkette sein. Eine Leere Liste wird durch den STRING '' abgebildet, der STRING enthält keine Elemente. Die Länge einer Liste ist die Anzahl der Elemente die in der Liste enthalten sind, die Länge ist bei einer Leeren Liste 0. Die Funktionen zur Verarbeitung von Listen benutzen für die Listen I/O Variablen, damit die unter Umständen langen Listen nicht bei jedem Funktionsaufruf erst im Speicher kopiert werden müssen. Das Separationszeichen SEP der Listen kann vom Anwender Frei festgelegt werden und wird den Funktionen mit dem Eingang SEP übergeben. Das Separationszeichen ist immer nur ein einzelnes Zeichen und kann jedes in einem STRING zulässige Zeichen sein.

In den folgenden Beispielen wird als Separationszeichen ein '§' verwendet.

Leere Liste:
''

Liste mit einem Leeren Element: '§'

Liste mit 2 Elementen
'§1§NIX'

Liste mit 6 Elementen wovon eines Leer ist '§1§§33§/§1§2'

26.2. LIST_ADD

Type	Funktion : BOOL
Input	SEP : BYTE (Separationszeichen der Liste)
	INS : STRING (Neues Element)
I/O	LIST : STRING(LIST_LENGTH) (Eingangsliste)
Output	BOOL (TRUE)

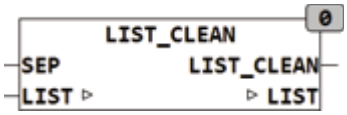
LIST_ADD addiert ein weiteres Element an das Ende einer Liste. Die Liste besteht aus Zeichenketten (Elementen) die mit dem Separationszeichen SEP beginnen.

Beispiel:

LIST_ADD('&ABC&23&&NEXT', 38, 'NEW') = '&ABC&23&&NEXT&NEW'

26.3. LIST_CLEAN

Type	Funktion : BOOL
Input	SEP : BYTE (Separationszeichen der Liste)
I/O	LIST : STRING(LIST_LENGTH) (Eingangsliste)
Output	BOOL (TRUE)



LIST_CLEAN bereinigt eine Liste von leeren Elementen. Die Liste besteht aus Zeichenketten (Elementen) die mit dem Separationszeichen SEP beginnen.

LIST_CLEAN('&ABC$23&&NEXT', 38) = '&ABC&23&NEXT'

LIST_CLEAN('&&23&&NEXT&', 38) = '&23&NEXT'

LIST_CLEAN('&&&&', 38) = ''

26.4. LIST_GET

Type	Funktion : STRING(LIST_LENGTH)
Input	SEP : BYTE (Separationszeichen der Liste)
	POS : INT (Position des Listenelements)
I/O	LIST : STRING(LIST_LENGTH) (Eingangsliste)
Output	STRING (Ausgangsstring)

LIST_GET lieferte das Element an der Stelle POS aus einer Liste. Die Liste besteht aus Zeichenketten (Elementen) die mit dem Separationszeichen SEP beginnen. Das erste Element der Liste hat die Position 1.

Beispiel:

LIST_GET('&ABC&23&&NEXT', 38, 1) = 'ABC'

LIST_GET('&ABC&23&&NEXT', 38, 2) = '23'

LIST_GET('&ABC&23&&NEXT', 38, 3) = ''

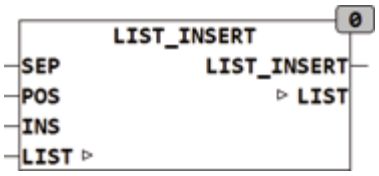
LIST_GET('&ABC&23&&NEXT', 38, 4) = 'NEXT'

LIST_GET('&ABC&23&&NEXT', 38, 5) = ''

LIST_GET('&ABC&23&&NEXT', 38, 0) = ''

26.5. LIST_INSERT

Type	Funktion : BOOL
Input	SEP : BYTE (Separationszeichen der Liste)
	POS : INT (Position des Listenelements)
	INS : STRING (Neues Element)
I/O	LIST : STRING(LIST_LENGTH) (Eingangsliste)
Output	BOOL (TRUE)



LIST_INSERT setzt ein Element an der Stelle POS in eine Liste ein. Die Liste besteht aus Zeichenketten (Elementen) die mit dem Separationszeichen SEP beginnen. Das erste Element der Liste hat die Position 1. Wird eine Position größer als das letzte Element der Liste angegeben werden leere Elemente an die Liste angehängt bis INS an seiner vorgesehenen Position am Ende der Liste steht. Ist POS = 0 wird das neue Element an den Anfang der Liste gestellt.

Beispiel:

LIST_INSERT('&ABC&23&&NEXT',38,0,'NEW')= '&NEW&ABC&23&&NEXT'

LIST_INSERT('&ABC&23&&NEXT',38,1,'NEW')= '&NEW&ABC&23&&NEXT'

LIST_INSERT('&ABC&23&&NEXT',38,3,'NEW')= '&ABC&23&NEW&&NEXT'

LIST_INSERT('&ABC&23&&NEXT',38,6,'NEW')= '&ABC&23&&NEXT&&NEW'

26.6. LIST_LEN

Type	Funktion : INT
Input	SEP : BYTE (Separationszeichen der Liste)
I/O	LIST : STRING(LIST_LENGTH) (Eingangsliste)
Output	INT (Anzahl der Elemente in der Liste)

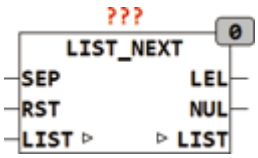
LIST_LEN ermittelt die Anzahl der Elemente in einer Liste.

LIST_LEN('&0&1&2&3', 38) = 4

LIST_LEN('',21) = 0

26.7. LIST_NEXT

Type	Funktion : STRING
Input	SEP : BYTE (Separationszeichen der Liste)
	RST : BOOL (Asynchroner Reset)
I/O	LIST : STRING(LIST_LENGTH) (Eingangsliste)
Output	LEL : STRING(LIST_LENGTH) (Listenelement)
	NUL : BOOL (TRUE wenn Liste abgearbeitet oder Leer ist)



LIST_NEXT liefert jeweils das nächste Element aus einer Liste. Die Liste ist ein STRING dessen Elemente mit den Zeichen SEP beginnen. Das erste Element der Liste hat die Position 1. Nach dem ersten Aufruf von LIST_NEXT oder einem Reset wird am Ausgang LEL das erste Element der Liste ausgegeben. Bei jedem weiteren Aufruf liefert der Baustein das nächste Element der Liste. Wenn das Ende der Liste erreicht ist wird eine Leere Zeichenkette Ausgegeben und der Ausgang NUL = TRUE gesetzt. Mit dem Kommando RST = TRUE kann die LISTE wieder von neuem bearbeitet werden.

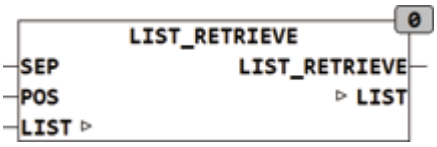
Beispiel für die Anwendung:

```
FUNCTION_BLOCK testll
VAR_INPUT
        s1 : STRING(255);
END_VAR
VAR
        element : ARRAY[0..20] OF STRING(LIST_LENGTH);
        list_n : LIST_NEXT;
        pos : INT;
END_VAR

pos := 0;
list_n(LIST := s1, SEP := 44);
WHILE NOT list_n.NUL and pos <= 20 DO
        element[pos] := list_n.LEL;
        list_n(list := s1);
        pos := pos + 1;
END_WHILE;
```

26.8. LIST_RETRIEVE

Type	Funktion : STRING
Input	SEP : BYTE (Separationszeichen der Liste)
	POS : INT (Position des Listenelements)
I/O	LIST : STRING(LIST_LENGTH) (Eingangsliste)
Output	STRING(LIST_LENGTH) (Ausgangsstring)



LIST_RETRIEVE lieferte das Element an der Stelle POS aus einer Liste und löscht das entsprechende Element aus der Liste. Die Liste besteht aus Zeichenketten (Elementen) die mit dem Separationszeichen SEP beginnen. Das erste Element der Liste hat die Position 1. Die Funktion liefert einen leeren String wenn an der Stelle POS kein Element vorhanden ist.

Beispiel:

LIST_RETRIEVE('&ABC&23&&NX&, 38, 1) = 'ABC' LIST = '&23&&NX&'

```
LIST_RETRIEVE('&ABC&23&&NX', 38, 2) = '23'      LIST = '&ABC&&NX'
LIST_RETRIEVE('&ABC&23&&NX', 38, 3) = ''        LIST = '&ABC&23&NX'
LIST_RETRIEVE('&ABC&23&&NX', 38, 4) = 'NEXT'    LIST = '&ABC&23&'
LIST_RETRIEVE('&ABC&23&&NX', 38, 5) = ''        LIST = '&ABC&23&&NX'
LIST_RETRIEVE('&ABC&23&&NX', 38, 0) = ''        LIST = '&ABC&23&&NX'
```

26.9. LIST_RETRIEVE_LAST

Type	Funktion : STRING(LIST_LENGTH)
Input	SEP : BYTE (Separationszeichen der Liste)
I/O	LIST : STRING(LIST_LENGTH) (Eingangsliste)
Output	STRING(LIST_LENGTH) (Ausgangsstring)

LIST_RETRIEVE_LAST lieferte das letzte Element aus einer Liste und löscht das entsprechende Element aus der Liste. Die Liste besteht aus Zeichenketten (Elementen) die mit dem Separationszeichen SEP beginnen.

Verzeichnis der Funktionsbausteine