

# JUMO diraTRON/diraVIEW 104/108/116/132

Kompaktregler/Digitalanzeiger



ST-Editor-Anleitung



70211000T96Z000K000

V2.00/DE/00688819



<b>1</b>	<b>Einleitung</b> .....	<b>5</b>
1.1	Sicherheitshinweise .....	5
1.2	Bestimmungsgemäße Verwendung .....	5
1.3	Qualifikation des Personals .....	5
1.4	Inhalt dieses Dokuments. ....	5
<b>2</b>	<b>Programmoberfläche</b> .....	<b>7</b>
<b>3</b>	<b>Systemvariablen</b> .....	<b>15</b>
3.1	Eingangsvariablen .....	15
3.2	Ausgangsvariablen .....	16
3.3	Interne Variablen. ....	19
<b>4</b>	<b>Datentypen</b> .....	<b>21</b>
4.1	Felder .....	23
<b>5</b>	<b>Operatoren</b> .....	<b>25</b>
<b>6</b>	<b>Auswahanweisungen</b> .....	<b>27</b>
<b>7</b>	<b>Wiederholanweisungen</b> .....	<b>29</b>
<b>8</b>	<b>Funktionen</b> .....	<b>31</b>
8.1	Typumwandlung .....	31
8.2	Arithmetische Funktionen .....	33
8.3	Numerische Funktionen .....	34
8.4	Bitfolge-Funktionen .....	36
8.5	Logische Funktionen .....	37
8.6	Auswahl .....	39
8.7	Vergleich .....	40
8.8	Datum und Uhrzeit .....	41
8.9	Weitere Funktionen .....	42
<b>9</b>	<b>Funktionsbausteine</b> .....	<b>43</b>
9.1	Software-Up-/Down-Zähler .....	44
9.2	Pulsgeber .....	45
9.3	Einschaltverzögerung .....	47
9.4	Ausschaltverzögerung .....	49

---

# Inhalt

---

9.5	Flankenerkennung steigend .....	51
9.6	Flankenerkennung fallend .....	52
9.7	Bistabile Funktion SR .....	54
9.8	Bistabile Funktion RS .....	55
9.9	Reglerparameter setzen .....	56

## 1.1 Sicherheitshinweise

### Hinweisende Zeichen



#### **HINWEIS!**

Dieses Zeichen weist auf eine **wichtige Information** über das Produkt oder dessen Handhabung oder Zusatznutzen hin.

---



#### **VERWEIS!**

Dieses Zeichen weist auf **weitere Informationen** in anderen Abschnitten, Kapiteln oder anderen Anleitungen hin.

---

## 1.2 Bestimmungsgemäße Verwendung

Das Gerät ist für die Verwendung in industrieller Umgebung bestimmt, wie in den technischen Daten spezifiziert. Eine andere oder darüber hinausgehende Nutzung gilt als nicht bestimmungsgemäß.

Das Gerät ist entsprechend den gültigen Normen und Richtlinien sowie den geltenden sicherheitstechnischen Regeln gebaut. Dennoch können bei unsachgemäßer Verwendung Personen- oder Sachschäden entstehen.

Um Gefahren zu vermeiden, darf das Gerät nur benutzt werden:

- für die bestimmungsgemäße Verwendung
- in sicherheitstechnisch einwandfreiem Zustand
- unter Beachtung der mitgelieferten Technischen Dokumentation

Auch wenn das Gerät sachgerecht oder bestimmungsgemäß eingesetzt wird, können von ihm applikationsbedingte Gefahren ausgehen, z. B. durch fehlende Sicherheitseinrichtungen oder falsche Einstellungen.

## 1.3 Qualifikation des Personals

Dieses Dokument enthält die erforderlichen Informationen für den bestimmungsgemäßen Gebrauch des darin beschriebenen Gerätes.

Es wendet sich an technisch qualifiziertes Personal, das speziell ausgebildet ist und einschlägiges Wissen auf dem Gebiet der Automatisierungstechnik besitzt.

Die Kenntnis und das technisch einwandfreie Umsetzen der in der mitgelieferten Technischen Dokumentation enthaltenen Sicherheitshinweise und Warnungen sind Voraussetzungen für die gefahrlose Montage, Installation und Inbetriebnahme sowie für die Sicherheit während des Betriebes des beschriebenen Gerätes. Nur qualifiziertes Personal verfügt über das erforderliche Fachwissen, um die in diesem Dokument verwendeten Sicherheitshinweise und Warnungen im konkreten Einzelfall richtig zu interpretieren und in die Tat umzusetzen.

## 1.4 Inhalt dieses Dokuments



#### **HINWEIS!**

Dieses Dokument gilt sowohl für die Geräte der Geräteserie 70211x (Kompaktregler) als auch für die Geräte der Geräteserie 70151x (Digitalanzeiger).

---

Dieses Dokument beschreibt die Anwendung des ST-Editors, mit dem der Anwender eine eigene Applikation in der SPS-Programmiersprache „Strukturierter Text“ (ST) für das Gerät erstellen kann. Das Dokument richtet sich an Anwender, die einschlägige Programmierkenntnisse besitzen.

# 1 Einleitung

---

Die SPS-Programmiersprache „Strukturierter Text“ (ST) wird in der Norm DIN IEC 61131-3 beschrieben. Ausführliche Informationen zur Programmierung sind dieser Norm zu entnehmen. Das ST-Modul des betreffenden Gerätes unterstützt nur eine Teilmenge der in der Norm beschriebenen Programmiersprache.

Ergänzend zu diesem Dokument ist die Betriebsanleitung der betreffenden Geräteserie zu beachten:

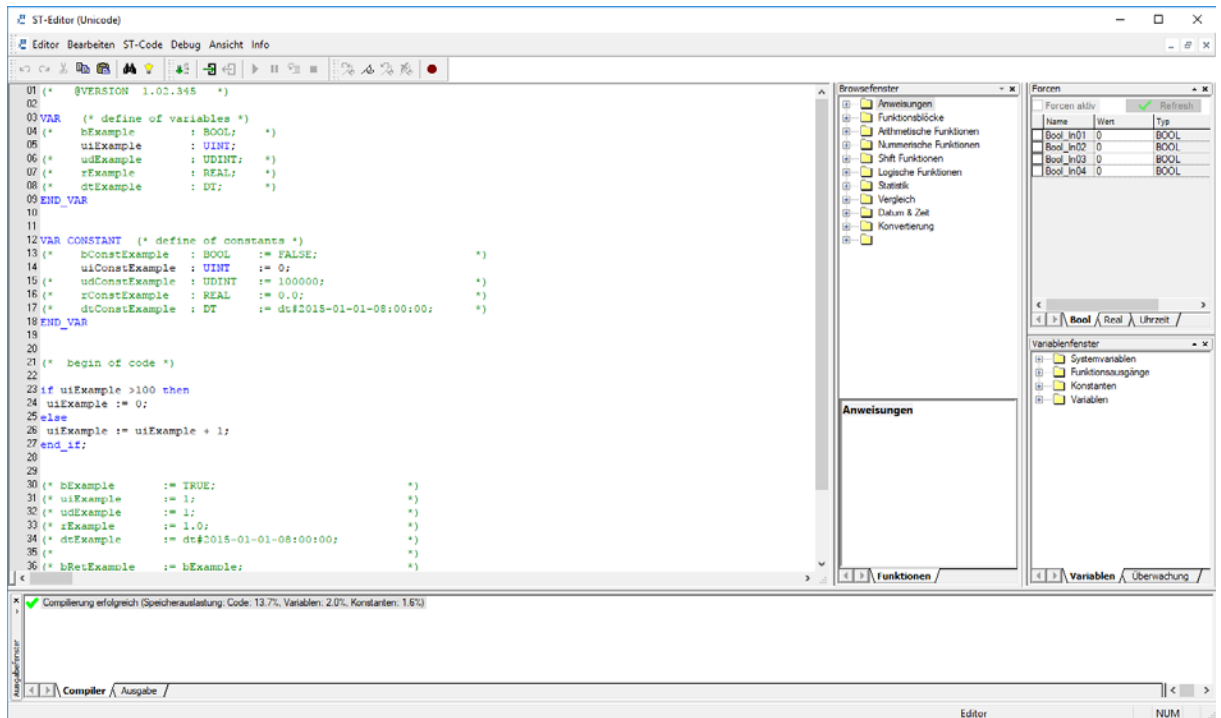
- Typen 701510, 701511, 701512, 701513, 701514 (Digitalanzeiger):  
Dokument 70151000T90Z...K...
- Typen 702110, 702111, 702112, 702113, 702114 (Kompaktregler):  
Dokument 70211000T90Z...K...

## 2 Programmoberfläche

Der ST-Editor ist Bestandteil des Setup-Programms und wird von diesem aus durch Anklicken der entsprechenden Schaltfläche im Fenster „ST-Code“ gestartet (siehe Betriebsanleitung).

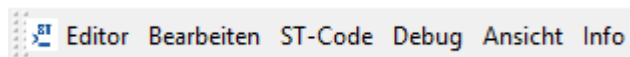
Die fertige Applikation wird als ST-Code zum Gerät übertragen und im integrierten ST-Modul ständig abgearbeitet.

### Übersicht



Die Programmoberfläche besteht aus mehreren Leisten und Fenstern, die in den folgenden Abschnitten kurz beschrieben werden.

### Menüleiste



Die einzelnen Menüs enthalten Funktionen zum Bearbeiten und Übersetzen eines Programms, zur Fehlersuche, zum Aus- und Einblenden von Leisten und Fenstern der Programmoberfläche sowie Versionsinformationen zum ST-Editor.

In der Statusleiste werden weitere Informationen zu den Funktionen der Menüleiste eingeblendet, wenn der Mauszeiger auf die einzelne Funktion im jeweiligen Menü zeigt.

## 2 Programmoberfläche

### Symbolleiste




Einige Funktionen der Menüleiste stehen auch in der Symbolleiste zur Verfügung und können hier durch einfachen Mausklick ausgewählt werden. Die Bedeutung der Symbole wird durch eine Tooltip-Funktion stichwortartig beschrieben (mit dem Mauszeiger auf das jeweilige Symbol zeigen). Zusätzlich werden dann in der Statusleiste weitere Informationen zu der betreffenden Funktion eingeblendet.

Symbol	Bedeutung	Beschreibung
	Rückgängig (Strg+Z)	Letzte Aktion rückgängig machen
	Wiederherstellen	Zuvor rückgängig gemachte Aktion wiederholen
	Ausschneiden (Strg+X)	Markierte Daten entfernen und in Zwischenablage übertragen
	Kopieren (Strg+C)	Markierte Daten kopieren und in Zwischenablage übertragen
	Einfügen (Strg+V)	Daten aus der Zwischenablage einfügen
	Suchen (Strg+F)	Eingegebenen Text suchen
	Funktionstext (F4)	Funktionstext aus dem Browse-Fenster einfügen Siehe auch Abschnitt „Browse-Fenster“, Seite 11.
	Übersetzen (F7)	ST-Code übersetzen
	Debugging starten	Debugging starten (Kaltstart) Nach dem Starten wird ein bereits im Gerät vorhandenes Programm beendet. Stattdessen wird das aktuelle Programm in das Gerät geladen. Siehe auch Abschnitt „Debugging“, Seite 13.
	Debugging beenden	Debugging beenden Nach dem Beenden bleibt das aktuelle Programm im Gerät erhalten und wird ausgeführt. Siehe auch Abschnitt „ST-Editor beenden“, Seite 13.
	Starten/Weiter (F5)	Programm starten oder nach Unterbrechung weiter ausführen
	Programm unterbrechen	Programm unterbrechen
	Einzelschritt (F11)	Programm per Einzelschritt ausführen
	Beenden	Programm beenden
	Nächstes Lesezeichen (F2)	Einfügemarke zum nächsten Lesezeichen verschieben
	Lesezeichen umschalten (Strg+F2)	Lesezeichen für die aktuelle Zeile umschalten
	Vorheriges Lesezeichen (Shift+F2)	Einfügemarke zum vorherigen Lesezeichen verschieben
	Lesezeichen löschen (Shift+Strg+F2)	Alle Lesezeichen löschen



## 2 Programmoberfläche

Symbol	Bedeutung	Beschreibung
	Haltepunkt umschalten (F9)	Haltepunkt für die aktuelle Zeile umschalten

### Bearbeitungsfenster

```
01 (* @VERSION 1.02.345 *)
02
03 VAR (* define of variables *)
04 (* bExample      : BOOL;    *)
05   uiExample      : UINT;
06 (* udExample     : UDINT;   *)
07 (* rExample      : REAL;    *)
08 (* dtExample     : DT;      *)
09 END_VAR
10
11
12 VAR CONSTANT (* define of constants *)
13 (* bConstExample : BOOL    := FALSE;          *)
14   uiConstExample : UINT    := 0;
15 (* udConstExample : UDINT   := 100000;       *)
16 (* rConstExample  : REAL    := 0.0;          *)
17 (* dtConstExample : DT      := dt#2015-01-01-08:00:00; *)
18 END_VAR
19
20
21 (* begin of code *)
22
23 if uiExample >100 then
24   uiExample := 0;
25 else
26   uiExample := uiExample + 1;
27 end_if;
28
```

Im Bearbeitungsfenster wird das Programm erstellt, indem Variablen und Konstanten deklariert werden, Programmcode erstellt wird und Kommentartexte verwendet werden.

Um das Programm mit einer Versionsbezeichnung zu versehen, muss in einer Kommentarzeile das Schlüsselwort `@VERSION` gefolgt von der Versionsbezeichnung verwendet werden. Enthält das Programm mehrere Kommentarzeilen mit diesem Schlüsselwort, wird nur die erste Kommentarzeile ausgewertet. Nachdem der ST-Code dauerhaft zum Gerät übertragen wurde, lässt sich die Versionsbezeichnung im Gerät anzeigen (Geräteinfo > Versionen > ST-Code-Version).

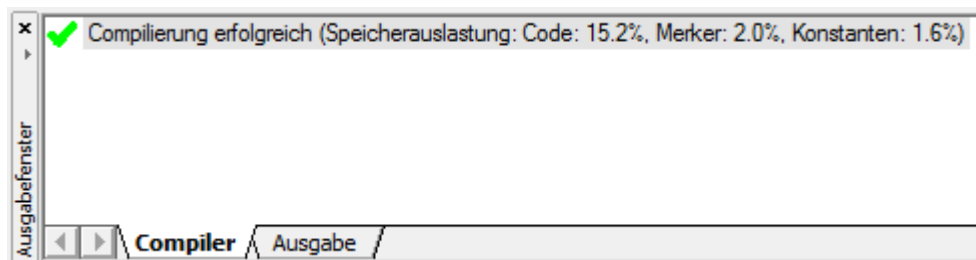
Durch Strg+Mausrad lässt sich die Schriftgröße ändern.

## 2 Programmoberfläche

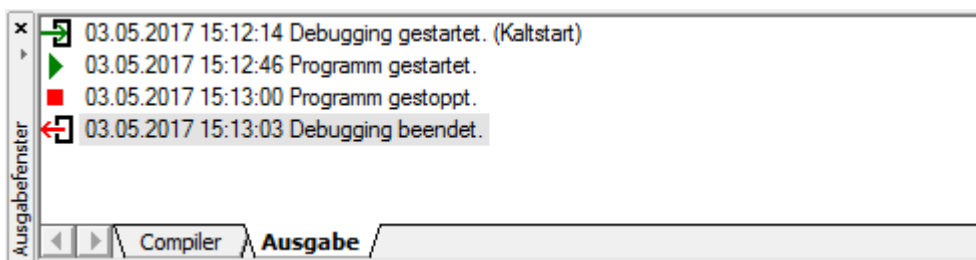
---

### Ausgabefenster

Das Ausgabefenster besteht aus zwei Teilen (Registern).

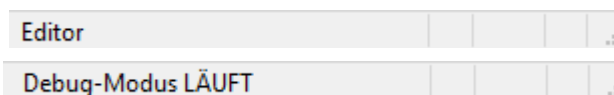


Im Register „Compiler“ werden Meldungen angezeigt, die das Übersetzen des Programmcodes betreffen.



Das Register „Ausgabe“ enthält Meldungen, die den Debug-Modus betreffen.

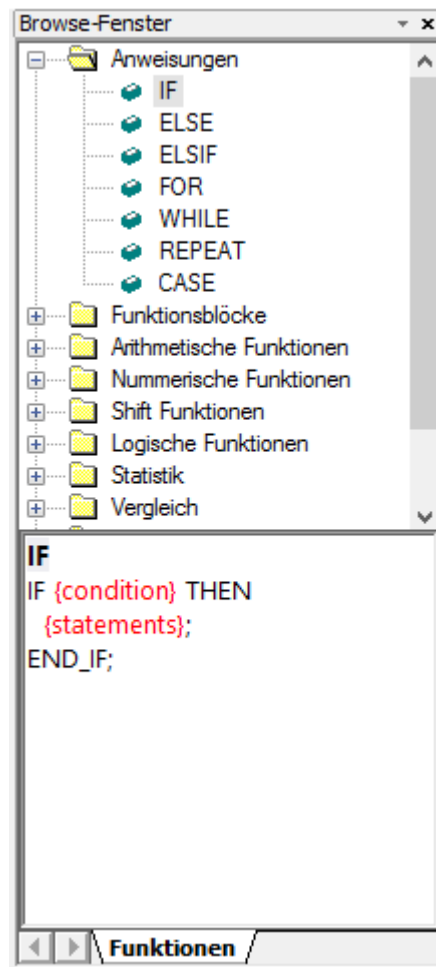
### Statusleiste



Die Statusleiste befindet sich am unteren Rand der Programmoberfläche und zeigt an, ob der Editor oder der Debug-Modus aktiv ist. Im Falle des Debug-Modus' werden die einzelnen Zustände (STOPP, LÄUFT, PAUSE) angezeigt.

Darüber hinaus werden in der Statusleiste weitere Informationen zu den Funktionen der Menüleiste und der Symbolleiste eingeblendet (mit dem Mauszeiger auf die einzelne Funktion im jeweiligen Menü bzw. auf das Symbol in der Symbolleiste zeigen).

### Browse-Fenster



Im Browse-Fenster sind alle Anweisungen und Funktionen aufgelistet, die im Editor genutzt werden können. Im unteren Teil des Fensters wird die Verwendung der Anweisung bzw. Funktion gezeigt.

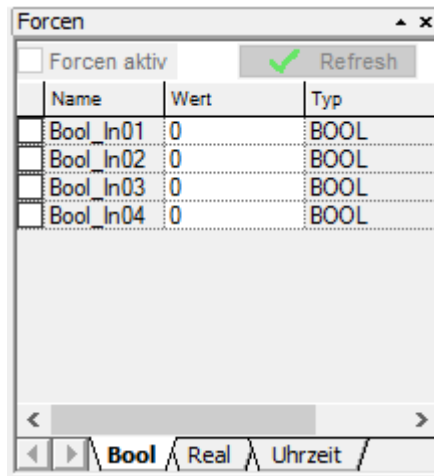
Der im unteren Teil des Browse-Fensters dargestellte Funktionstext kann per Drag & Drop (Text zuvor markieren), durch Klicken auf das Symbol „Funktionstext“ oder mit Funktionstaste F4 im Bearbeitungsfenster an der gewünschten Stelle in den ST-Code eingefügt werden (am Cursor oder anstelle eines markierten Textes). Nach den Einfügen wird durch wiederholtes Klicken auf das Symbol bzw. Drücken von F4 der nächste Platzhalter im Funktionstext markiert.

## 2 Programmoberfläche

---

### Forcen-Fenster

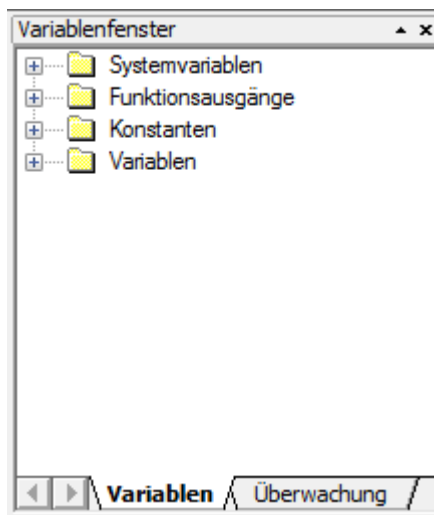
Das Forcen-Fenster besteht aus drei Teilen (Registern). Hier ist nur das Register „Bool“ dargestellt.



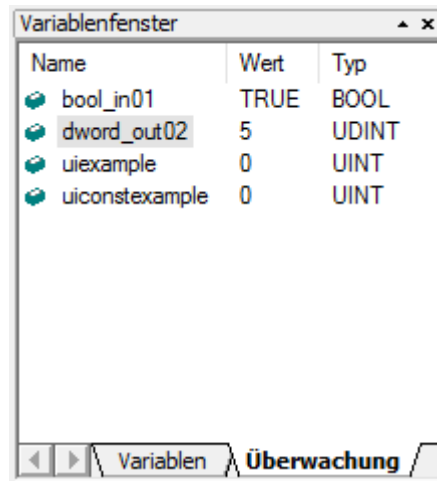
Im Forcen-Fenster können die Eingangsvariablen vom Typ „Bool“ und „Real“ sowie Datum und Uhrzeit vom Anwender gesetzt werden, um den Programmcode zu testen.

### Variablenfenster

Das Variablenfenster besteht aus zwei Teilen (Registern).



Im Register „Variablen“ werden alle Systemvariablen und Funktionsausgänge (Ausgänge der Funktionsblöcke) sowie die vom Anwender deklarierten Konstanten und Variablen (Merker) aufgelistet. Diese können per „Drag and Drop“ aus dem Variablenfenster kopiert und im Bearbeitungsfenster an der gewünschten Stelle in den ST-Code eingefügt werden.



Im Register „Überwachung“ können Variablen und Konstanten dargestellt und während des Debuggens beobachtet werden. Dazu müssen diese per „Drag and Drop“ aus dem Bearbeitungsfenster oder dem Register „Variablen“ kopiert und im Register „Überwachung“ eingefügt werden.

Das Einfügen und auch das Entfernen ist nur möglich, wenn das Programm nicht läuft. Dann stehen auch über das Kontextmenü (rechte Maustaste) verschiedene Funktionen zur Verfügung.

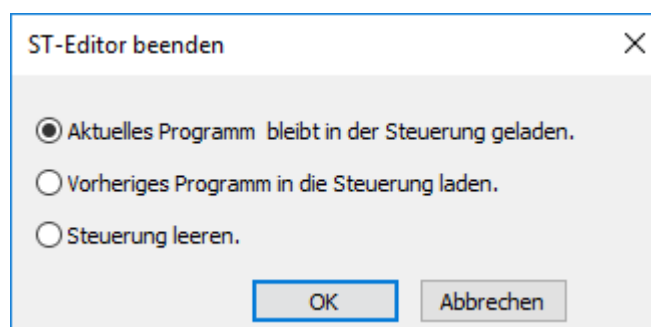
### Debugging

Mit der Funktion *Debug > Debugging starten* (Menüleiste) stehen folgende Varianten zur Verfügung:

- Kaltstart: Ein bereits im Gerät laufendes Programm wird beendet, und das aktuelle Programm aus dem ST-Editor wird in das Gerät geladen. Alle Variablen (Retain- und Nicht-Retain-Variablen) werden auf ihren Default-Wert zurückgesetzt.
- Warmstart: Die Retain-Variablen werden nicht zurückgesetzt (Geräte des Typs 70211x und 70151x unterstützen keine Retain-Variablen).
- Servicestart: Das aktuell im Gerät laufende Programm wird im ST-Editor verwendet und kann analysiert werden. Dabei werden keine Variablen zurückgesetzt. Voraussetzung für den Servicestart ist, dass der im ST-Editor kompilierte Code mit dem Code im Gerät übereinstimmt.

### ST-Editor beenden

Beim Beenden des ST-Editors mit *Editor > Speichern und beenden* (Menüleiste) stehen folgende Möglichkeiten zur Auswahl (sofern der Debugging-Modus aktiv war):



- *Aktuelles Programm bleibt in der Steuerung geladen.*  
Das zuletzt im Debugging-Modus ins Gerät geladene Programm soll verwendet werden.
- *Vorheriges Programm in die Steuerung laden.*  
Das zuletzt im Debugging-Modus ins Gerät geladene Programm soll nicht verwendet werden. Stattdessen wird das vorherige (dauerhaft ins Gerät geladene) Programm verwendet, sofern vorhanden.
- *Steuerung leeren.*  
Weder das zuletzt im Debugging-Modus ins Gerät geladene Programm, noch das vorherige Programm soll verwendet werden.

## 2 Programmoberfläche

---



### **HINWEIS!**

Im Debugging-Modus wird das Programm nur temporär in das Gerät geladen (keine Sicherung bei Netz-Ausfall ). Damit Änderungen im Gerät dauerhaft wirksam werden, muss nach dem Beenden des ST-Editors die Setup-Datei zum Gerät übertragen werden (Fenster ST-Code mit „OK“ schließen, Datentransfer zum Gerät).

---



### **HINWEIS!**

Der im ST-Editor erstellte Quellcode wird kompiliert ins Gerät übertragen. Es ist daher erforderlich, den Quellcode in einer Setup-Datei separat zu sichern.

---

Zu den Systemvariablen gehören die Eingangs- und Ausgangsvariablen sowie die internen Variablen. Über die Eingangs- und die Ausgangsvariablen können Werte unterschiedlichen Datentyps zwischen dem Gerät und dem im Gerät integrierten ST-Modul ausgetauscht werden. Die internen Variablen sind nur innerhalb des ST-Moduls von Bedeutung und können zur Realisierung bestimmter Funktionen verwendet werden.

## 3.1 Eingangsvariablen

Die Eingangsvariablen stellen Werte des Geräts für die Verwendung im ST-Modul zur Verfügung.

Die Zuweisung der Werte zu den Eingangsvariablen erfolgt im Setup-Programm, indem die betreffenden Signale aus den sogenannten Selektoren (Analogselektor bzw. Digitalselektor) ausgewählt werden. Darüber hinaus gibt es einige Variablen, die im Gerät fest zugeordnet sind und die deshalb nicht in den Selektoren vorkommen.

### bool\_in

Bezeichnung	Bezeichnung im Setup-Programm	Beschreibung
bool_in01 bis bool_in04	bool_in01 bis bool_in04	Boolesche Eingangsvariablen; flexible Zuweisung im Setup-Programm (Digitalselektor)

### real\_in

Bezeichnung	Bezeichnung im Setup-Programm	Beschreibung
real_in01 bis real_in06	real_in01 bis real_in06	Real-Eingangsvariablen; flexible Zuweisung im Setup-Programm (Analogselektor)

### dword\_in

Bezeichnung	Bezeichnung im Setup-Programm	Beschreibung
dword_in01 dword_in02	(keine Bezeichnung, keine feste Zuordnung)	(Diese Double-Word-Eingangsvariablen werden nicht verwendet.)

### rtc.cdt

Bezeichnung	Bezeichnung im Setup-Programm	Beschreibung
rtc.cdt	(keine Bezeichnung, feste Zuordnung)	Diese Eingangsvariable vom Typ DT liefert die Zeit seit Einschalten des Gerätes (Datum und Uhrzeit beginnen mit jedem Einschalten ab 01.01.1970 00:00:00).

# 3 Systemvariablen

## 3.2 Ausgangsvariablen

Mit den Ausgangsvariablen werden die im ST-Modul erzeugten Werte an das Gerät übergeben.

Die Ausgangsvariablen stehen im Setup-Programm und im Gerät in den Selektoren (Analogselektor bzw. Digitalselektor) für die Konfiguration zur Verfügung und können individuell verwendet werden. Darüber hinaus gibt es einige Variablen, die im Gerät bestimmte Funktionen haben (z. B. Textanzeige) und die deshalb nicht über die Selektoren auswählbar sind.

Zu bestimmten Ausgangsvariablen gibt es im Variablenfenster zugehörige Variablen mit den Bezeichnungen „...conf“ und „...sec“. Mit diesen Variablen lässt sich im ST-Code ein definiertes Verhalten im Fehlerfall realisieren.

- Die **Variable** „...conf“ kann im ST-Code gesetzt werden, um zu entscheiden, welchen Wert die zugehörige Ausgangsvariable im Fehlerfall annimmt: FALSE = aktueller Wert; TRUE = Ersatzwert.

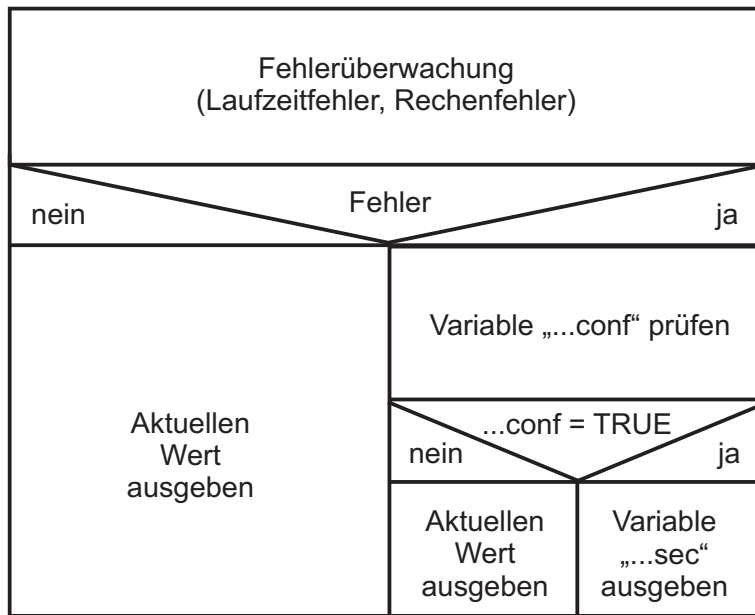
Default-Einstellung: FALSE

- Die **Variable** „...sec“ enthält den Ersatzwert; sie kann im ST-Code auf einen bestimmten Wert oder auf den Default-Wert gesetzt werden.

Im Falle der REAL-Variablen wird der Ersatzwert auf Einhaltung der Skalierungsgrenzen geprüft. Liegt er außerhalb der Grenzen, wird der Default-Wert verwendet.

Zum Setzen auf den Default-Wert steht die Funktion SET\_DEFAULT zur Verfügung.

⇒ 8.9 „Weitere Funktionen“, Seite 42



### bool\_out

Bezeichnung	Bezeichnung im Setup-Programm und im Gerät	Beschreibung
bool_out01 bis bool_out04	1. ST-Digitalausgang bis 4. ST-Digitalausgang	Boolesche Ausgangsvariablen; flexible Verwendung im Setup-Programm und im Gerät (Digitalselektor)  Im Setup-Programm kann jeder Variablen ein Beschreibungstext zugewiesen werden.



## 3 Systemvariablen

### real\_out

Bezeichnung	Bezeichnung im Setup-Programm und im Gerät	Beschreibung
real_out01 bis real_out06	1. ST-Analogausgang bis 6. ST-Analogausgang	Real-Ausgangsvariablen; flexible Verwendung im Setup-Programm und im Gerät (Analogselektor)  Im Setup-Programm kann jeder Variablen ein Beschreibungstext zugewiesen werden. Außerdem sind dort Einstellungen erforderlich, die die Variable betreffen (z. B. Skalierung).

### alarm

Bezeichnung	Bezeichnung im Setup-Programm und im Gerät	Beschreibung
alarm	ST-Alarm	Boolesche Ausgangsvariable; flexible Verwendung im Setup-Programm und im Gerät (Digitalselektor)  Diese Variable hat im Gegensatz zu den Variablen bool_out eine spezielle Bezeichnung. Sie kann jedoch im ST-Code frei verwendet werden.

### error\_out

Bezeichnung	Bezeichnung im Setup-Programm und im Gerät	Beschreibung
error_out	ST-Fehler	Boolesche Ausgangsvariable; flexible Verwendung im Setup-Programm und im Gerät (Digitalselektor)  Diese Variable wird auf TRUE gesetzt, wenn der ST-Code nicht weiter abgearbeitet werden kann, wie beispielsweise in folgenden Fällen: <ul style="list-style-type: none"><li>• Division durch 0</li><li>• falscher Array-Zugriff</li><li>• Laufzeitüberschreitung</li></ul> Die Variable wird wieder auf FALSE gesetzt, wenn der ST-Code nach Auftreten des Fehlers einmal komplett durchlaufen wurde.

### dword\_out

Bezeichnung	Bezeichnung im Setup-Programm und im Gerät	Beschreibung
dword_out01, dword_out02	(keine Bezeichnung, feste Zuordnung)	Double-Word-Ausgangsvariablen der Applikation; feste Zuordnung: Textanzeige im Gerätedisplay  Die Texte sind im Setup-Programm im Dialog „Anzeige/Bedienung“ unter „Anzeigetexte“ einzugeben.

### 3 Systemvariablen

---

Variable	Wert	Funktion
dword_out01	0	Keine Textanzeige
	1 bis 10	Geräte in den Formaten 108H, 108Q, 104: Der jeweilige Text (1 bis 10) wird in der 3. Zeile angezeigt.  Geräte in den Formaten 132, 116: Der jeweilige Text (1 bis 10) wird in der 1. Zeile angezeigt.
dword_out02	0	Keine Textanzeige
	1 bis 10	Geräte in den Formaten 108H, 108Q, 104: Der jeweilige Text (1 bis 10) wird in der 4. Zeile angezeigt.  Geräte in den Formaten 132, 116: Der jeweilige Text (1 bis 10) wird in der 2. Zeile angezeigt.

## 3.3 Interne Variablen

Die internen Variablen können zur Realisierung bestimmter Funktionen im ST-Code verwendet werden.

### ext\_error

Bezeichnung	Beschreibung
ext_error	Integer-Variable Die Variable ist 0, wenn die Funktion Set_CP (Reglerparameter setzen) fehlerfrei durchgeführt wurde. Im Fehlerfall liefert die Variable einen Wert > 0.

### reset\_flag

Bezeichnung	Beschreibung
reset_flag	Boolesche Variable Die Variable wird nach Netz-Ein auf TRUE gesetzt. Wird nach Netz-Ein der ST-Code einmal komplett durchlaufen, ohne dass ein Fehler auftritt, wird die Variable auf FALSE gesetzt. Anderenfalls bleibt sie auf TRUE. Debug-Modus: Beim ersten Start im Debug-Modus oder Start nach einem Programmstopp ist die Variable TRUE. Ab einem zweiten Durchlauf wird sie auf FALSE gesetzt.

### sys\_error

Bezeichnung	Beschreibung
sys_error	Real-Variable Die Variable liefert einen Fehlerwert, wenn während des Programmablaufs ein Fehler auftritt: 1.00E+37 = allgemeiner Fehlerwert 1.0E+37 = Bereichsunterschreitung (Unterlauf) 2.0E+37 = Bereichsüberschreitung (Überlauf) 3.0E+37 = ungültiger Eingangswert 4.0E+37 = Division durch Null 5.0E+37 = fehlerhafter Mathematikwert 7.0E+37 = ungültiger Wert (allgemein)

### week\_day

Bezeichnung	Beschreibung
week_day	Integer-Variable Die Variable enthält den aktuellen Wochentag: 0 = Sonntag, 1 = Montag, 2 = Dienstag, 3 = Mittwoch, 4 = Donnerstag, 5 = Freitag, 6 = Samstag Der Wochentag wird aus der Eingangsvariablen rtc.cdt abgeleitet.

## 3 Systemvariablen

---

Das ST-Modul unterstützt folgende Datentypen:

- Boolescher Wert (BOOL)
- Integer-Zahl ohne Vorzeichen, 2 Byte (UINT)
- Double-Integer-Zahl ohne Vorzeichen, 4 Byte (UDINT)
- Gleitkommazahl (REAL)
- Datum und Uhrzeit (DT oder DATE\_AND\_TIME)



### HINWEIS!

Bei Operationen findet keine Bereichsüberwachung der Datentypen statt.  
Ausnahmen: Quadratwurzelfunktion (SRQT), Kehrwert (1/x)

---



### HINWEIS!

Das ST-Modul unterstützt keine Variablen, die über Netz-Aus gespeichert werden (Retain-Variablen).

---

## Boolescher Wert

**Schlüsselwort:** BOOL

**Wertebereich:** TRUE (1) oder FALSE (0)

**Deklaration einer Variablen (Beispiel):**

```
VAR
    bExample : BOOL;
END_VAR
```

**Deklaration einer Konstanten (Beispiel):**

```
VAR CONSTANT
    bConstExample : BOOL := FALSE;
END_VAR
```

**Zuweisung (Beispiel):**

```
bExample := TRUE;
```

## Integer-Zahl (2 Byte)

**Schlüsselwort:** UINT

**Wertebereich:** 0 bis 65535 (0 bis  $2^{16}-1$ )

**Deklaration einer Variablen (Beispiel):**

```
VAR
    uiExample : UINT;
END_VAR
```

**Deklaration einer Konstanten (Beispiel):**

```
VAR CONSTANT
    uiConstExample : UINT := 0;
END_VAR
```

**Zuweisung (Beispiel):**

```
uiExample := 1;
```

# 4 Datentypen

---

## Double-Integer-Zahl (4 Byte)

**Schlüsselwort:** UDINT

**Wertebereich:** 0 bis 4294967295 (0 bis  $2^{32}-1$ )

**Deklaration einer Variablen (Beispiel):**

```
VAR
    udExample : UDINT;
END_VAR
```

**Deklaration einer Konstanten (Beispiel):**

```
VAR CONSTANT
    udConstExample : UDINT := 100000;
END_VAR
```

**Zuweisung (Beispiel):**

```
udExample := 200000;
```

## Gleitkommazahl

**Schlüsselwort:** REAL

**Wertebereich:** -1.5E-45 bis 1.0E37  
(Werte  $\geq 1.0E37$  werden als Fehlerwerte interpretiert.)

**Deklaration einer Variablen (Beispiel):**

```
VAR
    rExample : REAL;
END_VAR
```

**Deklaration einer Konstanten (Beispiel):**

```
VAR CONSTANT
    rConstExample : REAL := 0.0;
END_VAR
```

**Zuweisung (Beispiel):**

```
rExample := 1.0;
```

## Datum und Uhrzeit

**Schlüsselworte:** DT oder DATE\_AND\_TIME

**Wertebereich:** yyyy-mm-dd-hh:mm:ss

**Deklaration einer Variablen (Beispiel):**

```
VAR
    dtExample : DT;
END_VAR
```

**Deklaration einer Konstanten (Beispiel):**

```
VAR CONSTANT
    dtConstExample : DT := dt#2017-12-31-23:59:59;
END_VAR
```

**Zuweisung (Beispiel):**

```
dtExample := dt#2017-01-01-08:00:00;
```

## 4.1 Felder

Alle Datentypen können auch als eindimensionales oder zweidimensionales Feld (Array) deklariert werden (siehe Norm DIN IEC 61131-3). In den folgenden Abschnitten wird exemplarisch der Datentyp REAL verwendet.

### Eindimensionales Feld

#### Deklaration von Variablen (Beispiel):

```
VAR
    rArrayExample : ARRAY [0..2] OF REAL; (* Feld mit 3 Variablen *)
END_VAR
```

#### Deklaration von Konstanten (Beispiel):

```
VAR CONSTANT
    rArrayConstExample : ARRAY [0..2] OF REAL := [0.0, 0.1, 0.2]; (* Feld mit 3 Konstanten *)
END_VAR
```

#### Zuweisung (Beispiel):

```
rArrayExample[0] := 0.0;
rArrayExample[1] := 0.1;
rArrayExample[2] := 0.2;
```

### Zweidimensionales Feld

#### Deklaration von Variablen (Beispiel):

```
VAR
    rArrayExample : ARRAY [0..2, 0..1] OF REAL; (* Feld mit 3 x 2 Variablen *)
END_VAR
```

#### Deklaration von Konstanten (Beispiel):

```
VAR CONSTANT
    rArrayConstExample : ARRAY [0..2, 0..1] OF REAL :=
        [0.0, 0.1, 0.2,
         1.0, 1.1, 1.2]; (* Feld mit 3 x 2 Konstanten *)
END_VAR
```

#### Zuweisung (Beispiel):

```
rArrayExample[0,0] := 0.0;
rArrayExample[1,0] := 0.1;
rArrayExample[2,0] := 0.2;
rArrayExample[0,1] := 1.0;
rArrayExample[1,1] := 1.1;
rArrayExample[2,1] := 1.2;
```

## 4 Datentypen

---



## 5 Operatoren

In der folgenden Tabelle sind alle Operatoren aufgeführt, die das ST-Modul unterstützt. Die Reihenfolge in der Tabelle richtet sich nach der Rangfolge der Operatoren, beginnend mit dem höchsten Rang.

Operation	Symbol	Zulässige Datentypen	Beispiel
Klammerung	(Ausdruck)		a := 3.0 * (b - 1.0);
Funktion	Bezeichner (Argumentliste)		i := MIN (3, j);
Negation	-	REAL	a := -a;
Komplement	NOT	BOOL, UINT, UDINT <sup>a</sup>	a := NOT b;
Multiplikation	*	UINT, UDINT, REAL	i := 5 * j;
Division	/		a := 5.0 / b;
Modulo	MOD	UINT, UDINT	j := i MOD 10;
Addition	+	UINT, UDINT, REAL	i := 5 + j;
Subtraktion	-		a := b - 5.0E20;
Vergleich	<, >, <=, >=	UINT, UDINT, REAL, DATE_AND_TIME	bExample := 5 <= j;
Gleichheit	=	BOOL, UINT, UDINT, REAL, DATE_AND_TIME	bExample := 5 = i;
Ungleichheit	<>		bExample := 5 <> j;
UND	& oder AND	BOOL, UINT, UDINT <sup>a</sup>	bExample := x AND y;
Exklusives ODER	XOR	BOOL, UINT, UDINT <sup>a</sup>	bExample := x XOR y;
ODER	OR	BOOL, UINT, UDINT <sup>a</sup>	bExample := x OR y;

<sup>a</sup> Bei Booleschen Variablen findet eine logische Verknüpfung statt, bei Integer-Variablen eine bitweise Verknüpfung.

## 5 Operatoren

---

## 6 Auswahanweisungen

---

Eine Auswahanweisung führt aufgrund einer festgelegten Bedingung eine Anweisung oder eine Gruppe von Anweisungen aus.

### IF-Anweisung

Die IF-Anweisung legt fest, dass eine Gruppe von Anweisungen nur ausgeführt wird, wenn der zugehörige boolesche Ausdruck den Wert TRUE (wahr) liefert. Falls die Bedingung falsch ist, darf entweder keine Anweisung ausgeführt werden oder die Anweisungsgruppe ist auszuführen, die dem Schlüsselwort ELSE folgt (oder dem Schlüsselwort ELSIF, falls seine zugehörige boolesche Bedingung wahr ist).

#### Schlüsselworte:

IF, THEN, ELSIF, ELSE, END\_IF

#### Beispiel:

VAR

    i : UINT;

END\_VAR

IF reset\_flag THEN

    bool\_out01 := false; (\* nach dem Reset werden zunächst die Ausgänge bool\_out01 und \*)

    bool\_out02 := false; (\* bool\_out02 zurückgesetzt und \*)

    i := 1; (\* i auf 1 gesetzt \*)

END\_IF;

if i>0 and i<=100 then

    bool\_out01 := true; (\* im Durchlauf 1 bis 100 wird der Ausgang bool\_out01 gesetzt ... \*)

    bool\_out02 := false; (\* und bool\_out02 zurückgesetzt \*)

elsif i>100 and i<=200 then

    bool\_out01 := false; (\* im Durchlauf 101 bis 200 wird der Ausgang bool\_out01 zurückgesetzt ... \*)

    bool\_out02 := false; (\* und bool\_out02 gesetzt \*)

else

    i := 1; (\* im Durchlauf 201 wird i auf 1 gesetzt und ... \*)

    bool\_out01 := false; (\* die Ausgänge bool\_out01 und ... \*)

    bool\_out02 := false; (\* bool\_out02 werden zurückgesetzt; das ganze beginnt von vorne \*)

end\_if;

i := i + 1; (\* i wird um 1 erhöht \*)

# 6 Auswahlanweisungen

---

## CASE-Anweisung

Mit der CASE-Anweisung werden mehrere bedingte Anweisungen zusammengefasst. Sie besteht aus einem Ausdruck, der eine Integer-Variable enthält, und einer Liste von Anweisungsgruppen. Jede Gruppe ist durch eine Marke gekennzeichnet, die aus einer oder mehreren (durch Komma getrennten) ganzen Zahlen besteht. Der Wert der Variablen bestimmt die Marke und somit die Anweisungsgruppe, die ausgeführt wird. Optional kann auch das Schlüsselwort ELSE verwendet werden. Dann wird, wenn kein Wert zutrifft, die Anweisungsgruppe ausgeführt, die dem Schlüsselwort ELSE folgt.

### Schlüsselworte:

CASE, OF, ELSE, END\_CASE

### Beispiel:

CASE i OF (\* Variable i muss UINT oder UDINT sein \*)

1: b1 := TRUE;

2, 3, 4: b2 := TRUE;

ELSE

b1 := FALSE;

b2 := FALSE;

END\_CASE;

# 7 Wiederholanweisungen

---

Mit Wiederholanweisungen (Iterationen) werden Anweisungen und Gruppen von Anweisungen wiederholt ausgeführt.

Es gibt drei Typen von Wiederholanweisungen:

- FOR
- WHILE
- REPEAT UNTIL

## EXIT

Mit EXIT wird eine Wiederholanweisung beendet, bevor die Endbedingung der Iteration erfüllt ist. Wird EXIT innerhalb eines geschachtelten Wiederholungskonstrukts ausgeführt, wird die innerste Schleife verlassen, innerhalb der EXIT liegt.



### HINWEIS!

Bei Wiederholanweisungen muss darauf geachtet werden, dass sich keine Endlosschleifen oder sehr lange laufende Schleifen ergeben. Die Laufzeit wird überwacht.

---

## FOR-Anweisung

Die FOR-Anweisung wird verwendet, wenn die Anzahl der Wiederholungen festliegt.

### Schlüsselworte:

FOR, TO, BY, DO, END\_FOR

### Beispiele:

```
FOR i := 10 TO 100 BY 10 DO (* Laufvariable muss UINT sein *)
    j := j + i; (* i läuft von 10 bis 100 in Schritten von 10 (10, 20, 30, ..., 100) *)
END_FOR;
```

```
FOR i := 1 TO 5 DO (* Ohne Angabe von „BY x“ wird ... *)
    a := a + b; (* die Laufvariable in jedem Durchlauf um 1 erhöht (1, 2, 3, 4, 5) *)
    a := a * 3.0;
END_FOR;
```

## WHILE-Anweisung

Die WHILE-Anweisung bewirkt, dass eine Gruppe von Anweisungen wiederholt ausgeführt wird, bis der zugehörige boolesche Ausdruck falsch (FALSE, unwahr) ist. Ist der boolesche Ausdruck von Beginn an falsch, wird die Gruppe von Anweisungen überhaupt nicht ausgeführt.

### Schlüsselworte:

WHILE, DO, END\_WHILE

### Beispiel:

```
WHILE j < 100 DO
    j := j + 2;
END_WHILE;
```

# 7 Wiederholanweisungen

---

## REPEAT-Anweisung

Die REPEAT-Anweisung bewirkt, dass eine Gruppe von Anweisungen bis zum Schlüsselwort UNTIL wiederholt (und mindestens einmal) ausgeführt wird, bis die zugehörige boolesche Bedingung wahr (TRUE) wird.

### Schlüsselworte:

REPEAT, UNTIL, END\_REPEAT

### Beispiel:

```
REPEAT
    i := i - 2;
    a := a + 2.0;
UNTIL i = 0 END_REPEAT;
```

Das ST-Modul unterstützt folgende Funktionen:

- Typumwandlung (Konvertierung)
- Arithmetische Funktionen
- Numerische Funktionen
- Bitfolge-Funktionen (Shift-Funktionen)
- Logische Funktionen
- Auswahl (Statistik)
- Vergleich
- Datum und Uhrzeit
- Spezifische Funktionen (nicht Bestandteil der Norm)

## 8.1 Typumwandlung

### Zulässige Datentypen

Argument: UINT, UDINT

Ergebnis: BOOL, UINT, UDINT, REAL

### INT\_TO\_REAL

Wandelt eine INTEGER- in eine REAL-Zahl um.

**Beispiel:**

```
a := INT_TO_REAL(10); (* a := 10.0 *)
```

### INT\_TO\_DINT

Wandelt eine INTEGER- in eine DOUBLE-INTEGGER-Zahl um.

**Beispiel:**

```
a := INT_TO_DINT(10); (* a := 10 *)
```

### INT\_TO\_BOOL

Wandelt eine INTEGER-Zahl in einen BOOL-Wert um.

Das Ergebnis ist FALSE, wenn das Argument 0 ist. In allen anderen Fällen ist das Ergebnis TRUE.

**Beispiele:**

```
a := INT_TO_BOOL(0); (* a = FALSE *)
```

```
b := INT_TO_BOOL(1); (* b = TRUE *)
```

```
c := INT_TO_BOOL(8); (* c = TRUE *)
```

### DINT\_TO\_REAL

Wandelt eine DOUBLE-INTEGGER-Zahl in eine REAL-Zahl um.

**Beispiele:**

```
a := DINT_TO_REAL(100000); (* a = 100000.0 *)
```

# 8 Funktionen

---

## DINT\_TO\_INT

Wandelt eine DOUBLE-INTEGGER-Zahl in eine INTEGER-Zahl um.

Dabei werden nur die unteren beiden Byte ausgewertet (Bit 0 bis Bit 15).

**Beispiele:**

a := DINT\_TO\_INT(1); (\* a = 1; 1 = 0000 0000 0000 0001 = 0x0001 \*)

b := DINT\_TO\_INT(65535); (\* b = 65535; 65535 = 1111 1111 1111 1111 = 0xFFFF \*)

c := DINT\_TO\_INT(65536); (\* c = 0; 65536 = 0001 0000 0000 0000 0000 = 0x10000 \*)

d := DINT\_TO\_INT(65537); (\* d = 1, 65537 = 0001 0000 0000 0000 0001 = 0x10001 \*)

## DINT\_TO\_BOOL

Wandelt eine DOUBLE-INTEGGER-Zahl in einen BOOL-Wert um.

Das Ergebnis ist FALSE, wenn das Argument 0 ist. In allen anderen Fällen ist das Ergebnis TRUE.

**Beispiele:**

a := DINT\_TO\_BOOL(0); (\* a = FALSE: \*)

b := DINT\_TO\_BOOL(1); (\* b = TRUE \*)

c := DINT\_TO\_BOOL(100000); (\* c = TRUE \*)

## REAL\_TO\_INT

Wandelt eine REAL-Zahl mit Rundung in eine INTEGER-Zahl um.

**Beispiele:**

a := REAL\_TO\_INT(1.378); (\* a = 1 \*)

b := REAL\_TO\_INT(1.897); (\* b = 2 \*)

## REAL\_TO\_DINT

Wandelt eine REAL-Zahl mit Rundung in eine DOUBLE-INTEGGER-Zahl um.

**Beispiele:**

a := REAL\_TO\_DINT(100000.378); (\* a = 100000 \*)

b := REAL\_TO\_DINT(100000.897); (\* b = 100001 \*)



## 8.2 Arithmetische Funktionen

### Zulässige Datentypen

Argument: UINT, UDINT, REAL (bei Negation nur REAL, bei Modulo nur UINT oder UDINT)

Ergebnis: UINT, UDINT, REAL

### Addition

Die Addition ist eine erweiterbare Funktion. Als Ergebnis wird die Summe der Argumente zurückgegeben.

#### Beispiel:

OUT := IN1 + IN2 + ... INn;

### Subtraktion

Als Ergebnis wird das Zweite von dem ersten Argument abgezogen.

#### Beispiel:

OUT := IN1 - IN2;

### Multiplikation

Die Multiplikation ist eine erweiterbare Funktion. Das Ergebnis ergibt sich aus der Multiplikation der Argumente.

#### Beispiel:

OUT := IN1 \* IN2 \* ...INn;

### Division

Als Ergebnis wird der Quotient aus den beiden Argumenten zurückgegeben.

Das Ergebnis der Division von ganzen Zahlen (UINT oder UDINT) ist eine ganze Zahl mit Abschneiden der Nachkommastellen (Beispiel:  $7/3 = 2$ ).

#### Beispiel:

OUT := IN1 / IN2;

### Negation

REAL-Zahlen können negiert werden.

#### Beispiel:

OUT := -IN;

### Modulo

Die Argumente der Modulo-Funktion (%) müssen ganze Zahlen (UINT oder UDINT) sein. Das Ergebnis der Modulo-Funktion ist gleich der Bearbeitung der folgenden Anweisung:

```
IF (IN2 = 0) THEN OUT := 0;  
ELSE OUT := IN1 - (IN1 / IN2) * IN2;  
END_IF
```

Das Ergebnis der Division der ganzen Zahlen (UINT oder UDINT) im ELSE-Zweig der obigen IF-Auswahlanweisung ist eine ganze Zahl mit Abschneiden der Nachkommastellen.

#### Beispiel:

```
IN1 := 17;  
IN2 := 3;  
OUT := IN1 % IN2; (* OUT := 2 *)
```

# 8 Funktionen

---

## 8.3 Numerische Funktionen

### Zulässige Datentypen

Argument: REAL

Ergebnis: REAL

### ABS (IN)

Liefert den Absolutwert von IN.

#### Beispiel:

IN := -2.3;

OUT := ABS(IN); (\* OUT = 2.3 \*)

### SQRT (IN)

Liefert die Quadratwurzel von IN.

Ist IN negativ, wird der Fehlerwert 5.0E+37 geliefert.

#### Beispiel:

IN := 4.0;

OUT := SQRT(IN); (\* OUT = 2.0 \*)

### LN (IN)

Liefert den natürlichen Logarithmus von IN.

#### Beispiel:

IN := 2.718282; (\* Zahl e (gerundet) \*)

OUT := LN(IN); (\* OUT = 1.0 (gerundet) \*)

### LOG (IN)

Liefert den Logarithmus zur Basis 10 von IN.

#### Beispiel:

IN := 100.0;

OUT := LOG(IN); (\* OUT = 2.0 \*)

### EXP (IN)

Liefert die natürliche Exponentialfunktion von IN.

#### Beispiel:

IN := 2.0;

OUT := EXP(IN); (\* OUT = 7.389056 \*)

### **SIN (IN)**

Liefert den Sinus von IN (Bogenmaß).

**Beispiel:**

IN := 1.5708; (\* entspricht 90° \*)

OUT := SIN(IN); (\* OUT = 1.0 \*)

### **COS (IN)**

Liefert den Cosinus von IN (Bogenmaß).

**Beispiel:**

IN := 0.0; (\* entspricht 0° \*)

OUT := COS(IN); (\* OUT = 1.0 \*)

### **TAN (IN)**

Liefert den Tangens von IN (Bogenmaß).

**Beispiel:**

IN := 0.7854; (\* entspricht 45° \*)

OUT := TAN(IN); (\* OUT = 1.0 \*)

### **ASIN (IN)**

Liefert den Arcussinus (Bogenmaß) von IN.

**Beispiel:**

IN := 1.0;

OUT := ASIN(IN); (\* OUT = 1.5708; entspricht 90° \*)

### **ACOS (IN)**

Liefert den Arcuscossinus (Bogenmaß) von IN.

**Beispiel:**

IN := 1.0;

OUT := ACOS(IN); (\* OUT = 0.0; entspricht 0° \*)

### **ATAN (IN)**

Liefert den Arcustangens (Bogenmaß) von IN.

**Beispiel:**

IN := 1.0;

OUT := ATAN(IN); (\* OUT = 0.7854; entspricht 45° \*)

# 8 Funktionen

---

## 8.4 Bitfolge-Funktionen

### Zulässige Datentypen

Argument: UINT, UDINT

Ergebnis: UINT, UDINT

### SHL (IN, n)

Verschiebt die Bitfolge des Arguments IN nach links um n Bit. Nachrückende Stellen von rechts werden mit 0 gefüllt.

#### Beispiel:

IN := 255; (\* Bitfolge: 0000 0000 1111 1111 \*)

OUT := SHL(IN, 4); (\* OUT = 4080; Bitfolge: 0000 1111 1111 0000 \*)

### SHR (IN, n)

Verschiebt die Bitfolge des Arguments IN nach rechts um n Bit. Nachrückende Stellen werden von links mit 0 gefüllt.

#### Beispiel:

IN := 255; (\* Bitfolge: 0000 0000 1111 1111 \*)

OUT := SHR(IN, 4); (\* OUT = 15; Bitfolge: 0000 0000 0000 1111 \*)

### ROL (IN, n)

Rotiert die Bitfolge des Arguments IN nach links um n Bit im Kreis.

#### Beispiel:

IN := 43690; (\* Bitfolge: 1010 1010 1010 1010 \*)

OUT := ROL(IN, 1); (\* OUT = 21845; Bitfolge: 0101 0101 0101 0101 \*)

### ROR (IN, n)

Rotiert die Bitfolge des Arguments IN nach rechts um n Bit im Kreis.

#### Beispiel:

IN := 21845; (\* Bitfolge: 0101 0101 0101 0101 \*)

OUT := ROR(IN, 1); (\* OUT = 43690; Bitfolge: 1010 1010 1010 1010 \*)

## 8.5 Logische Funktionen

Bei booleschen Variablen wird logisch, bei UINT- und UDINT-Variablen bitweise verknüpft.  
Bei mehr als zwei Parametern wird immer paarweise von links nach rechts verknüpft.

### Zulässige Datentypen

Operand: BOOL, UINT, UDINT  
Ergebnis: BOOL, UINT, UDINT

### AND

UND-Verknüpfung

#### Beispiele:

Logische Verknüpfung:

x := TRUE;  
y := FALSE;  
z := x AND y; (\* z = FALSE \*)

Bitweise Verknüpfung:

a := 5; .....	0101 = 5
b := 6; .....	.0110 = 6
.....	.....
c := a AND b; (* c = 4 *) .....	0100 = 4

### OR

ODER-Verknüpfung

#### Beispiele:

Logische Verknüpfung:

x := TRUE;  
y := FALSE;  
z := x OR y; (\* z = TRUE \*)

Bitweise Verknüpfung:

a := 5; .....	0101 = 5
b := 6; .....	.0110 = 6
.....	.....
c := a OR b; (* c = 7 *) .....	.0111 = 7

### XOR

Exklusiv-ODER-Verknüpfung

#### Beispiele:

Logische Verknüpfung:

x := TRUE;  
y := FALSE;  
z := x XOR y; (\* z = TRUE \*)

Bitweise Verknüpfung:

a := 5; .....	0101 = 5
b := 6; .....	.0110 = 6
.....	.....
c := a XOR b; (* c = 3 *) .....	.0011 = 3

# 8 Funktionen

---

## NOT

Invertierung (Komplement bilden)

### Beispiele:

IN := 1;

OUT := NOT IN; (\* OUT = 0 \*)

IN := 43690; (\* IN = 1010 1010 1010 1010 \*)

OUT := NOT IN; (\* OUT = 21845; OUT = 0101 0101 0101 0101 \*)

## 8.6 Auswahl

### Zulässige Datentypen

Argument: UINT, UDINT, REAL (bei SEL zusätzlich BOOL für den Selektor)

Ergebnis: UINT, UDINT, REAL

### MAX

Gibt das größte Argument als Ergebnis zurück.

#### Beispiel:

OUT := MAX(8, 9, 10, 11, 12); (\* OUT := 12 \*)

### MIN

Gibt das kleinste Argument als Ergebnis zurück.

#### Beispiel:

OUT := MIN(8, 9, 10, 11, 12); (\* OUT := 8 \*)

### LIMIT

Prüft, ob ein Wert (IN) innerhalb eines bestimmten Bereichs (MIN, MAX) liegt. Liegt der Wert innerhalb des Bereichs, wird der Wert selbst zurückgegeben. Liegt der Wert unterhalb des Bereichs, wird die untere Bereichsgrenze zurückgegeben. Liegt der Wert oberhalb des Bereichs, wird die obere Bereichsgrenze zurückgegeben.

#### Beispiele:

OUT := LIMIT(IN, MIN, MAX);

OUT := LIMIT(15, 10, 20); (\* OUT = 15 \*)

OUT := LIMIT(5, 10, 20); (\* OUT = 10 \*)

OUT := LIMIT(25, 10, 20); (\* OUT = 20 \*)

### SEL

Wählt in Abhängigkeit einer booleschen Variablen (G) einen von zwei Werten (IN0, IN1) aus. Der Datentyp der Werte IN0 und IN1 muss identisch sein.

#### Beispiele:

OUT := SEL(G, IN0, IN1);

OUT := SEL(G, 10, 20); (\* G = FALSE: OUT = 10 \*)

OUT := SEL(G, 10, 20); (\* G = TRUE: OUT = 20 \*)

# 8 Funktionen

---

## 8.7 Vergleich

### Zulässige Datentypen

Argument bei LT, LE, GT, GE: UINT, UDINT, REAL, DT

Argument bei EQ und NE: BOOL, UINT, UDINT, REAL, DT

Ergebnis: BOOL

### < (LT)

Vergleicht Argumente darauf, ob eines kleiner ist als ein anderes.

#### Beispiele:

bOUT := IN1 < IN2; (\* bOUT := TRUE, wenn IN1 kleiner IN2 ist \*)

bOUT := (IN1 < IN2) & (IN2 > IN3) & ... & (INn-1 > INn);

### <= (LE)

Vergleicht Argumente darauf, ob eines kleiner oder gleich einem anderen ist.

#### Beispiele:

bOUT := IN1 <= IN2; (\* bOUT := TRUE, wenn IN1 kleiner oder gleich IN2 ist \*)

bOUT := (IN1 <= IN2) & (IN2 <= IN3) & ... & (INn-1 <= INn);

### > (GT)

Vergleicht Argumente darauf, ob eines größer als ein anderes ist.

#### Beispiele:

bOUT := IN1 > IN2; (\* bOUT := TRUE, wenn IN1 größer IN2 ist \*)

bOUT := (IN1 > IN2) & (IN2 > IN3) & ... & (INn-1 > INn);

### >= (GE)

Vergleicht Argumente darauf, ob eines größer oder gleich einem anderen ist.

#### Beispiel:

bOUT := IN1 >= IN2; (\* bOUT := TRUE, wenn IN1 größer oder gleich IN2 ist \*)

### = (EQ)

Vergleicht Argumente darauf, ob sie gleich sind.

#### Beispiel:

bOUT := IN1 = IN2; (\* bOUT := TRUE, wenn IN1 gleich IN2 ist \*)

### <> (NE)

Vergleicht Argumente darauf, ob sie ungleich sind.

#### Beispiel:

bOUT := IN1 <> IN2; (\* bOUT := TRUE, wenn IN1 ungleich IN2 ist \*)



## 8.8 Datum und Uhrzeit

Hierbei handelt es sich um spezifische Funktionen, die nicht Bestandteil der Norm DIN IEC 61131-3 sind.

Mit diesen Funktionen kann beispielsweise die Eingangsvariable `rtc.cdt` abgefragt werden, die die Zeit seit Einschalten des Gerätes liefert (Datum und Uhrzeit beginnen mit jedem Einschalten ab 01.01.1970 00:00:00).

### Zulässige Datentypen

Argument: DT (oder DATE\_AND\_TIME)

Ergebnis: UINT

### GET\_YEAR

Gibt das Jahr (mit Jahrhundert) zurück.

#### Beispiele:

`iOUT := GET_YEAR(dt#2017-03-20-17:45:12); (* iOUT = 2017 *)`

### GET\_MONTH

Gibt den Monat zurück.

#### Beispiel:

`iOUT := GET_MONTH(dt#2017-03-20-17:45:12); (* iOUT = 03 *)`

### GET\_DAY

Gibt den Tag zurück.

#### Beispiel:

`iOUT := GET_DAY(dt#2017-03-20-17:45:12); (* iOUT = 20 *)`

### GET\_HOUR

Gibt die Stunde zurück.

#### Beispiel:

`iOUT := GET_HOUR(dt#2017-03-20-17:45:12); (* iOUT = 17 *)`

### GET\_MINUTE

Gibt die Minute zurück.

#### Beispiel:

`iOUT := GET_MINUTE(dt#2017-03-20-17:45:12); (* iOUT = 45 *)`

### GET\_SECOND

Gibt die Sekunde zurück.

#### Beispiel:

`iOUT := GET_SECOND(dt#2017-03-20-17:45:12); (* iOUT = 12 *)`

`iOUT := GET_SECOND(rtc_cdt); (* iOUT = 59, wenn rtc_cdt = 1970-01-01 00:00:59 *)`

# 8 Funktionen

---

## 8.9 Weitere Funktionen

Hierbei handelt es sich um spezifische Funktionen, die nicht Bestandteil der Norm DIN IEC 61131-3 sind.

### IS\_VALID

Prüft die Gültigkeit einer REAL-Zahl. Das Ergebnis ist ein boolescher Wert (gültig = TRUE, ungültig = FALSE).

Werte  $\geq 1.00E+37$  (allgemeiner Fehlerwert) gelten als ungültig, ebenso die Werte „Unendlich“ (INF) und „Keine Zahl“ (NaN).

**Beispiele:**

```
bOUT := IS_VALID(1.2); (* bOUT = TRUE *)
```

### SET\_DEFAULT

Setzt eine REAL- oder eine BOOL-Zahl auf den Default-Wert.

Default-Wert REAL: 5.0E+37 (Mathematik-Fehlerwert)

Default-Wert BOOL: FALSE

**Beispiel:**

```
SET_DEFAULT(bOUT); (* bOUT = FALSE *)
```

```
SET_DEFAULT(rOUT); (* rOUT = 5.0E+37 *)
```

## Instanz

Ein Funktionsbaustein – im ST-Editor als Funktionsblock bezeichnet – ist eine Programm-Organisations-einheit, die bei Ausführung einen oder mehrere Werte liefert (siehe auch Norm DIN IEC 61131-3). Ein Funktionsbaustein kann in mehreren Instanzen (Kopien) existieren. Jede dieser Instanzen besitzt einen zugehörigen Bezeichner. Die Instanzen eines Funktionsbausteins sind unabhängig voneinander.

Existiert ein Funktionsbaustein namens TON z. B. in 4 Instanzen, so werden die einzelnen Instanzen mit TON01, TON02, TON03, TON04 angesprochen.

Das ST-Modul enthält folgende Funktionsbausteine:

Bezeichner	Instanzen	Funktion
CTUD	4	Software-Up-/Down-Zähler
TP	4	Pulsgeber
TON	4	Einschaltverzögerung
TOF	4	Ausschaltverzögerung
R_TRIG	4	Flankenerkennung steigend (0 -> 1)
F_TRIG	4	Flankenerkennung fallend (1 -> 0)
SR	4	Bistabiler Funktionsbaustein (vorrangig setzen)
RS	4	Bistabiler Funktionsbaustein (vorrangig rücksetzen)
SET_CP	1	Reglerparameter vorgeben

## Parameter

Den Funktionsbausteinen wird ein oder werden mehrere Parameter (Eingänge) übergeben. Die Parameter folgen nach der Angabe der Instanz in runden Klammern. Die Reihenfolge der Parameter ist fest vorgegeben. Der Aufruf eines Funktionsbausteins wird, wie jede Anweisung, durch ein „;“ abgeschlossen.

### Beispiel:

```
TON01(TRUE,5,1);
```

## Ein-/Ausgänge

Die Zustände der Ein- und Ausgänge aller Instanzen der Funktionsbausteine bleiben immer bis zum nächsten Aufruf der Instanz des Funktionsbausteins konstant. Die Ausgänge können abgefragt und in beliebigen Operationen verwendet werden.

## Zugriff auf Ausgänge

Auf Ausgänge (Strukturelemente) von Funktionsbausteinen wird wie in der Programmiersprache Pascal üblich zugegriffen.

### Beispiel:

Abfragen des Ausgangs ET des Funktionsbausteins TON mit der Instanz 01:

```
OUT := TON01.ET;
```

# 9 Funktionsbausteine

## 9.1 Software-Up-/Down-Zähler

Der Zähler stellt folgende Funktionen bereit:

- Aufwärtszählen
- Abwärtszählen
- Rücksetzen des Zählerstands auf 0
- Setzen eines oberen Zählwertes
- Abfragen auf Überschreiten des oberen Zählwertes
- Abfragen auf Zählerstand 0
- Zählwert abfragen

Der Zähler zählt die positiven Flanken an den Eingängen CU (aufwärts zählen) und CD (abwärts zählen).

### Aufruf

CTUD<Instanz> (CU, CD, R, LD, PV);

### Eingänge

Die Parameter R, LD, CU und CD sind in der Tabelle entsprechend ihrer Priorität absteigend sortiert.

Parameter	Datentyp	Beschreibung
<Instanz>		01 bis 04 (Instanz zweistellig angeben)
R	BOOL	TRUE = Zählerstand auf 0 setzen
LD	BOOL	TRUE = oberen Zählwert setzen
CU	BOOL	positive Flanke = aufwärts zählen
CD	BOOL	positive Flanke = abwärts zählen
PV	UINT	oberer Zählwert

### Ausgänge

Parameter	Datentyp	Abfrage; (* Beschreibung *)
CV	UINT	OUT := CTUD<Instanz>.CV; (* OUT = Zählerstand *)
QU	BOOL	OUT := CTUD<Instanz>.QU; (* OUT = TRUE, wenn Zählerstand > oberer Zählwert *)
QD	BOOL	OUT := CTUD<Instanz>.QD; (* OUT = TRUE, wenn Zählerstand <= 0 *)

### Bemerkung

Um mit LD den oberen Zählwert zu setzen, darf R nicht TRUE sein.

Die Werte der Ein-/Ausgänge bleiben nicht über Netz-Aus erhalten.

### Beispiele

CTUD01 (IN, FALSE, FALSE, FALSE, 100); (\* aufwärts zählen der positiven Flanken von IN \*)

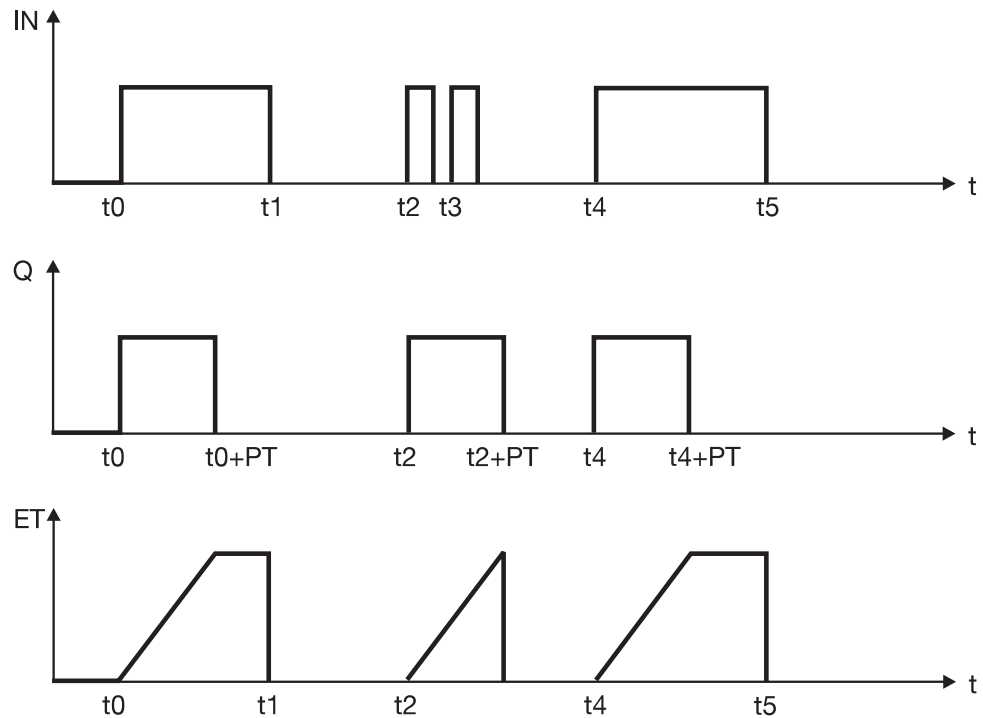
CTUD01 (FALSE, IN, FALSE, FALSE, 100); (\* abwärts zählen der positiven Flanken von IN \*)

CTUD01 (IN, FALSE, TRUE, FALSE, 100); (\* Zählerstand auf 0 setzen \*)

CTUD01 (IN, FALSE, FALSE, TRUE, 100); (\* oberen Zählwert setzen: 100 \*)

## 9.2 Pulsgeber

Die Funktionsweise des Pulsgebers veranschaulicht folgendes Diagramm:



### Aufruf

TP<Instanz> (IN, PT, TimeBase);

### Eingänge

Parameter	Datentyp	Beschreibung
<Instanz>		01 bis 04 (Instanz zweistellig angeben)
IN	BOOL	positive Flanke an IN setzt Q für die Zeit PT auf TRUE
PT	UINT	Zeit (Einheit: TimeBase), für die Q = TRUE wird bei IN = TRUE
TimeBase	UINT	Einheit von PT: 1 = ms 2 = s 3 = min

### Ausgänge

Parameter	Datentyp	Abfrage; (* Beschreibung *)
Q	BOOL	OUT := TP<Instanz>.Q; (* OUT = TRUE für die Zeit PT, wenn IN := 0 -> 1 *)
ET	UINT	OUT := TP<Instanz>.ET; (* OUT = vergangene Zeit seit IN = TRUE bis Q = FALSE (vergangene Zeit seit Beginn der aktiven Pulsphase) *)

### Bemerkung

Der Parameter TimeBase ist eine Erweiterung zur DIN IEC 61131-3. Diese Erweiterung stellt keine Einschränkung gegenüber der Norm dar. Die Zeitvorgabe ist laut Norm „implementierungsabhängig“.

Die Werte der Ein-/Ausgänge bleiben nicht über Netz-Aus erhalten.

## 9 Funktionsbausteine

---

Die zeitliche Auflösung der Pulslänge beträgt 150 ms (Zykluszeit des Gerätes).

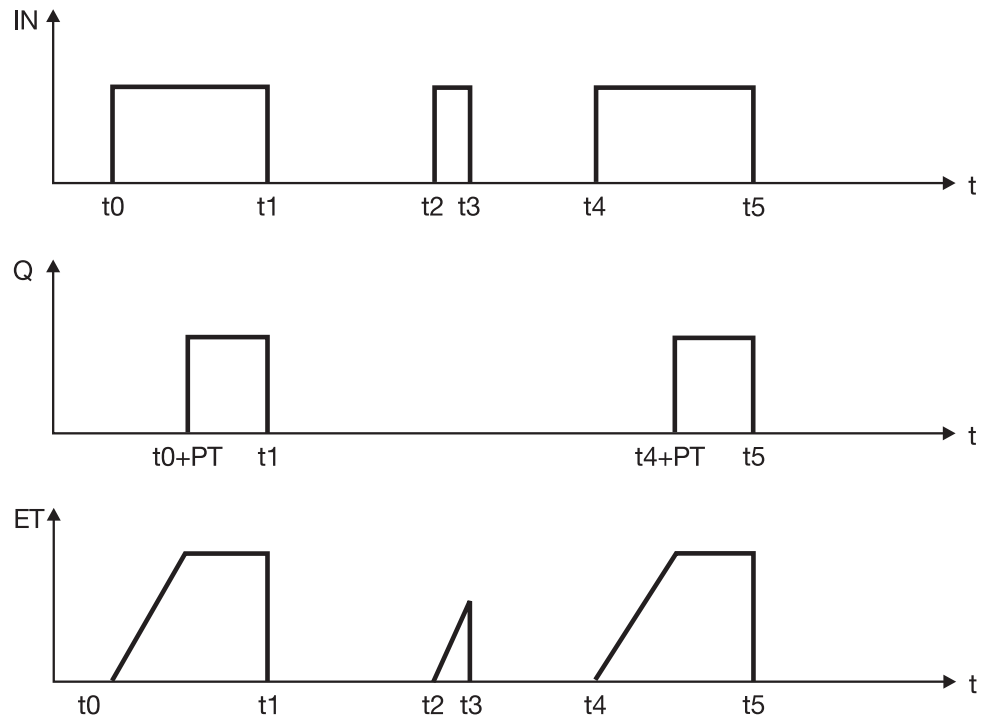
### Beispiel

TP01(IN, 5, 3);

(\* Positive Flanke an IN setzt TP01.Q für ca. 5 Minuten auf TRUE (zeitliche Auflösung beachten).  
TP01.ET liefert die vergangene Zeit in Minuten seit Beginn der aktiven Pulsphase. \*)

## 9.3 Einschaltverzögerung

Die Funktionsweise der Einschaltverzögerung veranschaulicht folgendes Diagramm:



### Aufruf

TON<Instanz> (IN, PT, TimeBase);

### Eingänge

Parameter	Datentyp	Beschreibung
<Instanz>		01 bis 04 (Instanz zweistellig angeben)
IN	BOOL	IN = TRUE setzt Q = TRUE, <b>nachdem</b> die Zeit PT abgelaufen ist
PT	UINT	Zeitverzögerung (Einheit: TimeBase)
TimeBase	UINT	Einheit von PT: 1 = ms 2 = s 3 = min

### Ausgänge

Parameter	Datentyp	Abfrage; (* Beschreibung *)
Q	BOOL	OUT := TON<Instanz>.Q; (* OUT = TRUE, wenn IN = TRUE und PT abgelaufen *)
ET	UINT	OUT := TON<Instanz>.ET; (* OUT = vergangene Zeit seit IN = TRUE bis Q = TRUE oder bis IN = FALSE *)

### Bemerkung

Der Parameter TimeBase ist eine Erweiterung zur DIN IEC 61131-3. Diese Erweiterung stellt keine Einschränkung gegenüber der Norm dar. Die Zeitvorgabe ist laut Norm „implementierungsabhängig“.

## 9 Funktionsbausteine

---

Die Werte der Ein-/Ausgänge bleiben nicht über Netz-Aus erhalten.

Die zeitliche Auflösung der Verzögerungszeit beträgt 150 ms (Zykluszeit des Gerätes).

### Beispiel

TON01(IN, 6, 2);

(\* IN = TRUE setzt nach 6 Sekunden TON01.Q auf TRUE.

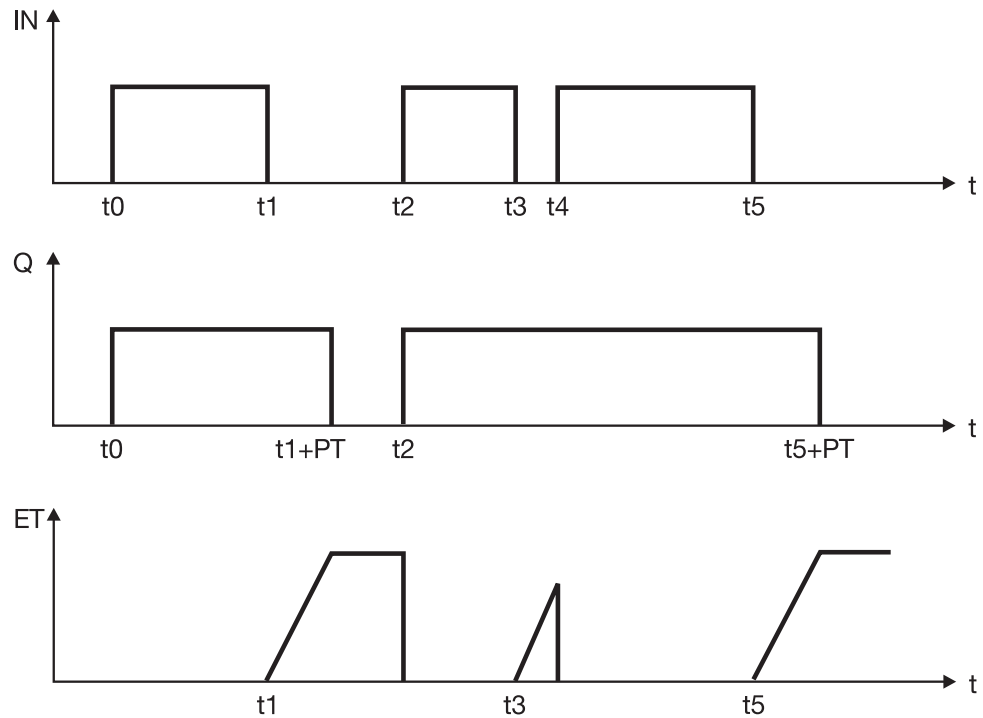
TON01.Q bleibt solange TRUE, wie IN = TRUE ist.

TON01.ET liefert die vergangene Zeit in Sekunden von IN = TRUE bis TON01.Q = TRUE (max. 6 Sekunden) oder bis IN = FALSE. \*)



## 9.4 Ausschaltverzögerung

Die Funktionsweise der Ausschaltverzögerung veranschaulicht folgendes Diagramm:



### Aufruf

TOF<Instanz> (IN, PT, TimeBase);

### Eingänge

Parameter	Datentyp	Beschreibung
<Instanz>		01 bis 04 (Instanz zweistellig angeben)
IN	BOOL	IN = TRUE setzt Q = TRUE IN = FALSE setzt Q um die Zeit PT verzögert auf FALSE
PT	UINT	Zeitverzögerung (Einheit: TimeBase)
TimeBase	UINT	Einheit von PT: 1 = ms 2 = s 3 = min

### Ausgänge

Parameter	Datentyp	Abfrage; (* Beschreibung *)
Q	BOOL	OUT := TOF<Instanz>.Q; (* OUT = TRUE, wenn IN = TRUE oder wenn IN = FALSE ist und PT noch nicht abgelaufen ist *)
ET	UINT	OUT := TOF<Instanz>.ET; (* OUT = vergangene Zeit seit IN = FALSE bis Q = FALSE oder bis IN = TRUE *)

## 9 Funktionsbausteine

---

### Bemerkung

Der Parameter TimeBase ist eine Erweiterung zur DIN IEC 61131-3. Diese Erweiterung stellt keine Einschränkung gegenüber der Norm dar. Die Zeitvorgabe ist laut Norm „implementierungsabhängig“.

Die Werte der Ein-/Ausgänge bleiben nicht über Netz-Aus erhalten.

Die zeitliche Auflösung der Verzögerungszeit beträgt 150 ms (Zykluszeit des Gerätes).

### Beispiel

TOF01(IN, 450, 1);

(\* IN = TRUE setzt TOF01.Q auf TRUE.

Nach IN = FALSE bleibt TOF01.Q TRUE, bis die Zeit PT = 450 ms abgelaufen ist.

TOF01.ET liefert die vergangene Zeit in Millisekunden von IN = FALSE bis TOF01.Q = FALSE (max. 450 ms) oder bis IN = TRUE. \*)

## 9.5 Flankenerkennung steigend

Dieser Funktionsbaustein erkennt eine steigende Flanke (Übergang 0 -> 1) .

### Aufruf

R\_TRIG<Instanz> (CLK);

### Eingang

Parameter	Datentyp	Beschreibung
<Instanz>		01 bis 04 (Instanz zweistellig angeben)
CLK	BOOL	Durch eine steigende Flanke (0 -> 1) an CLK wird Q = TRUE

### Ausgang

Parameter	Datentyp	Abfrage
Q	BOOL	OUT := R_TRIG<Instanz>.Q;

### Bemerkung

Die Funktionsweise des Funktionsbausteins entspricht folgendem Programmcode:

```
VAR CLK : BOOL; END_VAR  
VAR Q : BOOL; END_VAR  
VAR M : BOOL := FALSE; END_VAR  
Q := CLK AND NOT M;  
M := CLK;
```

Der Q-Ausgang bleibt von einem Aufruf zum nächsten auf seinem booleschen Wert. Er folgt dem Übergang des Eingangssignals (CLK) von „0“ nach „1“ und kehrt beim nächsten Aufruf nach „0“ zurück.

### Beispiel

```
IF reset_flag THEN  
  bool_out01 := FALSE;  
END_IF;  
R_TRIG01 (bool_in01); (* Erkennung einer steigenden Flanke an Eingang bool_in01 *)  
bool_out01 := R_TRIG01.Q; (* kurzer Impuls an Ausgang bool_out01 bei steigender Flanke am Eingang *)
```

# 9 Funktionsbausteine

## 9.6 Flankenerkennung fallend

Dieser Funktionsbaustein erkennt eine fallende Flanke (Übergang 1 -> 0) .

### Aufruf

F\_TRIG<Instanz> (CLK);

### Eingang

Parameter	Datentyp	Beschreibung
<Instanz>		01 bis 04 (Instanz zweistellig angeben)
CLK	BOOL	Durch eine fallende Flanke (1 -> 0) an CLK wird Q = TRUE

### Ausgang

Parameter	Datentyp	Abfrage
Q	BOOL	OUT := F_TRIG<Instanz>.Q;

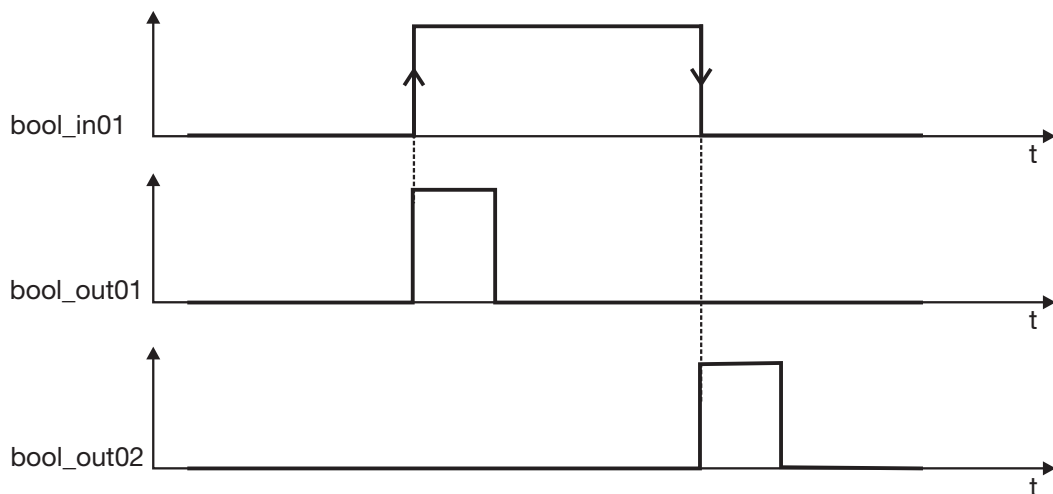
### Bemerkung

Die Funktionsweise des Funktionsbausteins entspricht folgendem Programmcode:

```
VAR CLK : BOOL; END_VAR  
VAR Q : BOOL; END_VAR  
VAR M : BOOL := TRUE; END_VAR  
Q := NOT CLK AND NOT M;  
M := NOT CLK;
```

Der Q-Ausgang bleibt von einem Aufruf zum nächsten auf seinem booleschen Wert. Er folgt dem Übergang des Eingangssignals (CLK) von „1“ nach „0“ und kehrt beim nächsten Aufruf nach „0“ zurück.

### Beispiel mti R\_TRIG und F\_TRIG



```
IF reset_flag THEN  
bool_out01 := FALSE;  
bool_out02 := FALSE;  
END IF;
```

```
R_TRIG01 (bool_in01); (* Erkennung einer steigenden Flanke an Eingang bool_in01 *)  
bool_out01 := R_TRIG01.Q; (* kurzer Impuls an Ausgang bool_out01 bei steigender Flanke am Eingang *)
```

## 9 Funktionsbausteine

---

```
F_TRIG01 (bool_in01); (* Erkennung einer fallenden Flanke an Eingang bool_in01 *)  
bool_out02 := F_TRIG01.Q; (* kurzer Impuls an Ausgang bool_out02 bei fallender Flanke am Eingang *)
```

# 9 Funktionsbausteine

## 9.7 Bistabile Funktion SR

Bistabile Funktion (vorrangig Setzen) mit Selbsthaltung

### Aufruf

SR<Instanz> (S1, R);

### Eingänge

Parameter	Datentyp	Beschreibung
<Instanz>		01 bis 04 (Instanz zweistellig angeben)
S1	BOOL	S1 = TRUE setzt Q1 = TRUE
R	BOOL	R = TRUE setzt Q1 = FALSE, wenn S1 = FALSE

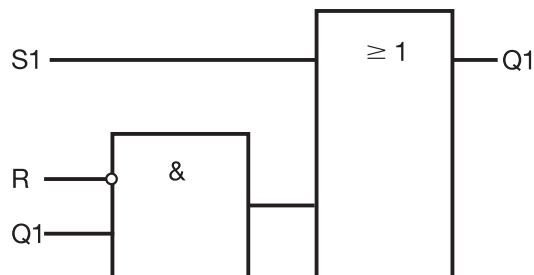
### Ausgänge

Parameter	Datentyp	Abfrage
Q1	BOOL	OUT := SR<Instanz>.Q1;

### Bemerkung

Der Ausgang Q1 bleibt nach Setzen durch S1 so lange TRUE, bis er mit R zurückgesetzt wird.  
Werden S1 und R gleichzeitig gesetzt (TRUE), wird der Ausgang Q1 ebenfalls gesetzt (TRUE).

### Logikfunktion



### Beispiel

SR1 (bool\_in01, bool\_in02); (\* Setzen mit Eingang bool\_in01, Rücksetzen mit Eingang bool\_in02 \*)  
bool\_out01 := SR1.Q1; (\* Der Zustand wird über Ausgang bool\_out01 ausgegeben. \*)

## 9.8 Bistabile Funktion RS

Bistabile Funktion (vorrangig Rücksetzen) mit Selbsthaltung

### Aufruf

RS<Instanz> (S, R1);

### Eingänge

Parameter	Datentyp	Beschreibung
<Instanz>		01 bis 04 (Instanz zweistellig angeben)
R1	BOOL	R1 = TRUE setzt Q1 = FALSE
S	BOOL	S = TRUE setzt Q1 = TRUE, wenn R1 = FALSE

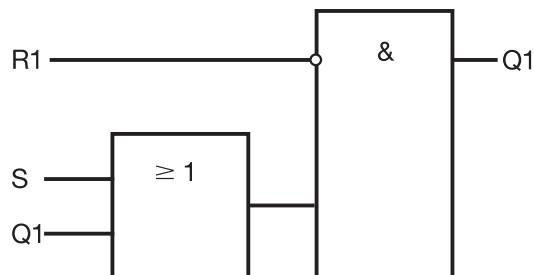
### Ausgänge

Parameter	Datentyp	Abfrage
Q1	BOOL	OUT := RS<Instanz>.Q1;

### Bemerkung

Der Ausgang Q1 bleibt nach Zurücksetzen durch R1 so lange FALSE, bis er mit S gesetzt wird.  
Werden R1 und S gleichzeitig gesetzt (TRUE), wird der Ausgang Q1 zurückgesetzt (FALSE).

### Logikfunktion



### Beispiel

```
RS1 (bool_in01, bool_in02); (* Rücksetzen mit Eingang bool_in02, Setzen mit Eingang bool_in01 *)
bool_out02 := RS1.Q1; (* Der Zustand wird über Ausgang bool_out02 ausgegeben. *)
```

# 9 Funktionsbausteine

## 9.9 Reglerparameter setzen

Mit diesem Funktionsbaustein können die Reglerparameter des **zweiten** Parametersatzes geändert werden (spezifische Funktion, nicht Bestandteil der Norm DIN IEC 61131-3).

### Aufruf

Set\_CP (Set, Key, Value);

### Eingänge

Parameter	Datentyp	Beschreibung
Controller	UINT	Regler-Nr.: 0 = Regler 1; 1 = Regler 2; ... Set = 0, wenn das Gerät nur einen Regler (Reglerkanal) hat.
Parameter	UINT	Parameter: 0 = Xp1 (Proportionalbereich 1) 1 = Xp2 (Proportionalbereich 2) 2 = Tv1 (Vorhaltezeit 1) 3 = Tv2 (Vorhaltezeit 2) 4 = Tn1 (Nachstellzeit 1) 5 = Tn2 (Nachstellzeit 2) 6 = Cy1 (Schaltperiodendauer 1) 7 = Cy2 (Schaltperiodendauer 2) 8 = Y1 (max. Stellgradbegrenzung) 9 = Y2 (min. Stellgradbegrenzung)
Value	REAL	Wert des betreffenden Parameters

### Ausgänge

Dieser Funktionsbaustein hat keine Ausgänge.

### Bemerkung

Die Werte der Parameter werden vor der Übernahme in den Regler geprüft. Liegt der Wert außerhalb des zulässigen Wertebereichs, wird die Übernahme verweigert und ein Fehlerwert in der Variablen `ext_error` zurückgegeben (kann im ST-Code ausgewertet werden). Ist der Wert gültig, wird ab diesem Zeitpunkt der neue Wert verwendet.

Die geänderten Werte werden nicht in der Reglerkonfiguration gespeichert. Nach einem Reset (Netz-Ein) werden wieder die gespeicherten Werte als Reglerparameter verwendet.

Um mehrere Parameter zu ändern, muss der Funktionsbaustein mehrfach aufgerufen werden (siehe Beispiel: zwei Aufrufe).



### Beispiel

```
VAR
  controller : UINT; (* Regler-Nr. *)
  para1 : UINT; (* Parameter 1 *)
  value1 : REAL; (* Wert von Parameter 1 *)
  para2 : UINT; (* Parameter 2 *)
  value2 : REAL; (* Wert von Parameter 2 *)
END_VAR
SET_CP (controller, para1, value1);
SET_CP (controller, para2, value2);
```







#### **JUMO GmbH & Co. KG**

Moritz-Juchheim-Straße 1  
36039 Fulda, Germany

Telefon: +49 661 6003-727  
Telefax: +49 661 6003-508  
E-Mail: mail@jumo.net  
Internet: www.jumo.net

#### Lieferadresse:

Mackenrodtstraße 14  
36039 Fulda, Germany

#### Postadresse:

36035 Fulda, Germany

#### Technischer Support Deutschland:

Telefon: +49 661 6003-9135  
Telefax: +49 661 6003-881899  
E-Mail: service@jumo.net

#### **JUMO Mess- und Regelgeräte GmbH**

Pfarrgasse 48  
1230 Wien, Austria

Telefon: +43 1 610610  
Telefax: +43 1 6106140  
E-Mail: info.at@jumo.net  
Internet: www.jumo.at

#### Technischer Support Österreich:

Telefon: +43 1 610610  
Telefax: +43 1 6106140  
E-Mail: info.at@jumo.net

#### **JUMO Mess- und Regeltechnik AG**

Laubisrütistrasse 70  
8712 Stäfa, Switzerland

Telefon: +41 44 928 24 44  
Telefax: +41 44 928 24 48  
E-Mail: info@jumo.ch  
Internet: www.jumo.ch

#### Technischer Support Schweiz:

Telefon: +41 44 928 24 44  
Telefax: +41 44 928 24 48  
E-Mail: info@jumo.ch

