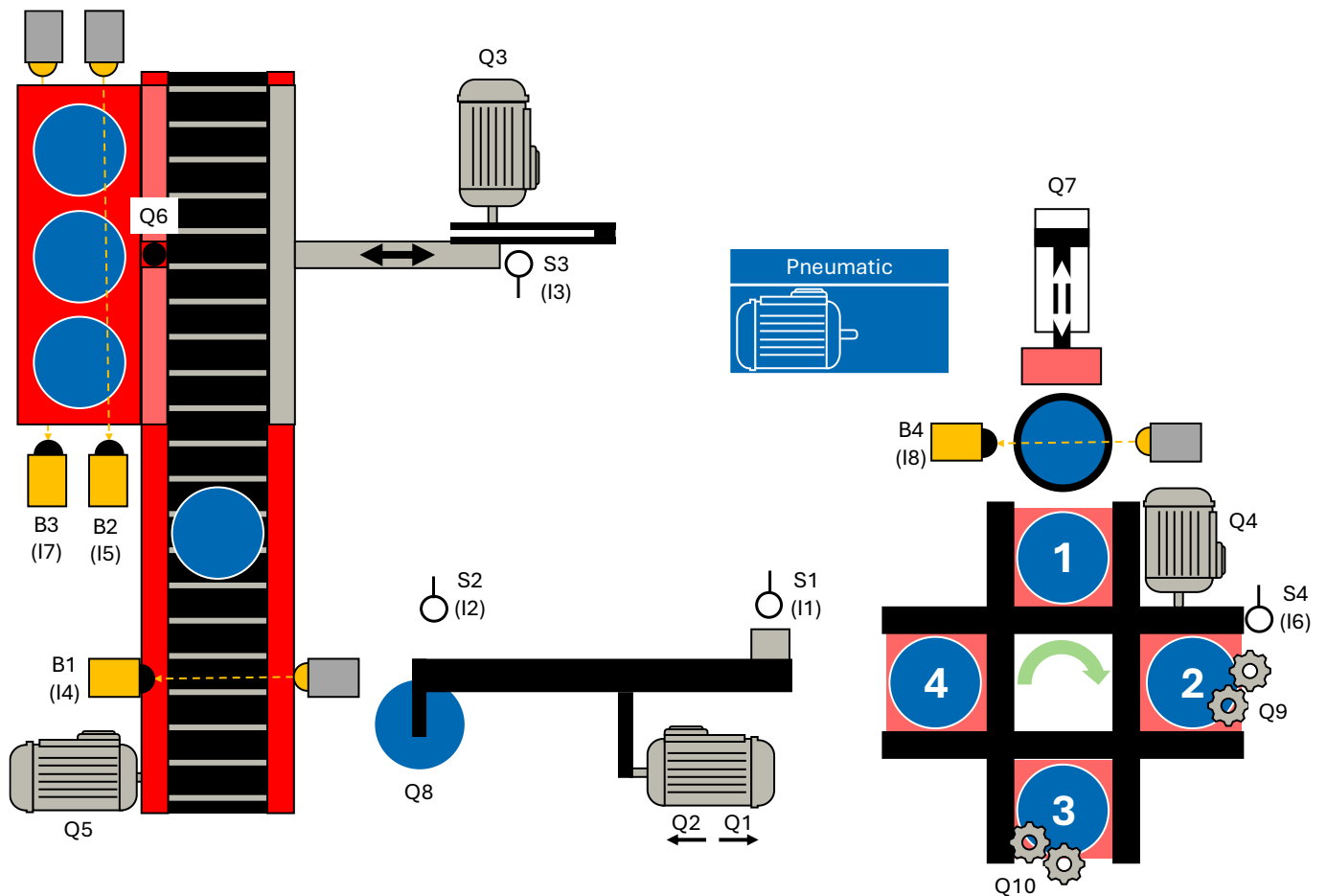


Fertigungsline 24V

Programmanweisungen



Inhaltsverzeichnis

7	Programmanweisungen	1
7.1	Flipflops.....	1
7.1.1	Dominant rücksetzendes Flipflop.....	2
7.1.2	Dominant setzendes Flipflop.....	5
7.1.3	Übung - Dominanzverhalten	8
7.2	Flanken	13
7.2.1	Positive Flanke erkennen R_TRIG.....	14
7.2.2	Negative Flanke erkennen F_TRIG	16
7.3	Zeiten	18
7.3.1	Einschaltverzögerung TON.....	20
7.3.2	Ausschaltverzögerung TOF.....	22
7.3.3	Impuls TP.....	24
7.3.4	Übung - IEC Zeitfunktionen	26
7.4	Zähler	31
7.4.1	Vorwärts zählen CTU.....	33
7.4.2	Rückwärts zählen CTD	35
7.4.3	Vorwärts und rückwärts zählen CTUD	37
7.4.4	Übung - IEC Zähler.....	39
7.5	IF-Anweisung [ST / SCL]	44
7.5.1	IF...THEN - Anweisung.....	44
7.5.2	IF...THEN...ELSE - Anweisung.....	45
7.5.3	IF...THEN...ELSIF - Anweisung	46
7.6	CASE-Struktur [ST / SCL]	48

7 Programmanweisungen

7.1 Flipflops

Ein Flipflop ist ein grundlegendes digitales Speicherelement, das einen binären Zustand (0 oder 1) speichern kann. Es hält einen kurzzeitig auftretenden Signalzustand fest und kann durch spezielle Steuerungsbefehle gesetzt oder zurückgesetzt werden.

Speicherfunktionen in der SPS-Programmierung spielen eine entscheidende Rolle, insbesondere bei der Steuerung von sequenziellen Abläufen und der Verwaltung von Zuständen. Sie sind grundlegend für die Implementierung von Logiken, die über einfache Ein-/Aus-Steuerungen hinausgehen.

Hier sind einige wesentliche Aspekte, wozu Speicherfunktionen dienen:

Zustandserhaltung:

Speicherfunktionen ermöglichen es einer SPS, den Zustand von Eingängen, internen Zuständen oder Ausgängen über die Zeit zu erhalten. Dies ist besonders wichtig in Anwendungen, wo Zustände über einen Zyklus oder sogar über mehrere Zyklen hinweg beibehalten werden müssen, wie bei Start/Stop-Vorgängen von Motoren oder der Überwachung von Sicherheitsfunktionen.

Steuerung von Abläufen:

In Produktionsprozessen, wo bestimmte Schritte in einer festgelegten Reihenfolge ablaufen müssen, helfen Speicherfunktionen, den aktuellen Schritt zu speichern und den nächsten Schritt basierend auf den erfüllten Bedingungen zu aktivieren.

Entprellung von Eingangssignalen:

Speicherfunktionen können verwendet werden, um die Auswirkungen von Prellen oder Störungen an Eingangssignalen zu minimieren. Durch das Speichern eines stabilen Zustands eines Eingangs vermeidet man, dass kurzzeitige Schwankungen zu ungewollten Aktionen führen.

Störungsbehandlung:

Speicherfunktionen können verwendet werden, um Fehlerzustände zu erfassen und zu speichern, sobald sie auftreten. Das Rücksetzen der Störung kann beispielsweise durch eine Anwenderquittierung erfolgen.

Prozesssteuerung:

Bei Prozessen, die nicht kontinuierlich ablaufen, sondern bei denen Aktionen in Abhängigkeit von bestimmten Ereignissen gestartet oder gestoppt werden müssen, sind Speicherfunktionen unverzichtbar. Sie ermöglichen die Speicherung von Ereignissen oder Zuständen, die zu einem späteren Zeitpunkt abgefragt und verarbeitet werden können.

Eine Speicherfunktion ist durch einen Setzbefehl und einen Rücksetzbefehl gekennzeichnet. Oberhalb der Speicherfunktion muss ein Datenbereich zum Speichern des Signalzustandes angegeben werden.

Die Speicherfunktion kann mit dominant (vorrangig) setzendem oder dominant rücksetzendem Speicherverhalten ausgeführt werden. Der dominante Eingang wird durch eine "1" gekennzeichnet.

7.1.1 Dominant rücksetzendes Flipflop

Mit der Anweisung setzen oder rücksetzen Sie das Bit eines angegebenen Operanden, abhängig vom Signalzustand an den Eingängen Set und Reset1. Der aktuelle Signalzustand des Operanden wird auf den Ausgang Q übertragen und kann an diesem abgefragt werden. Das Dominanzverhalten wird durch die "1" am Reset Eingang gekennzeichnet.

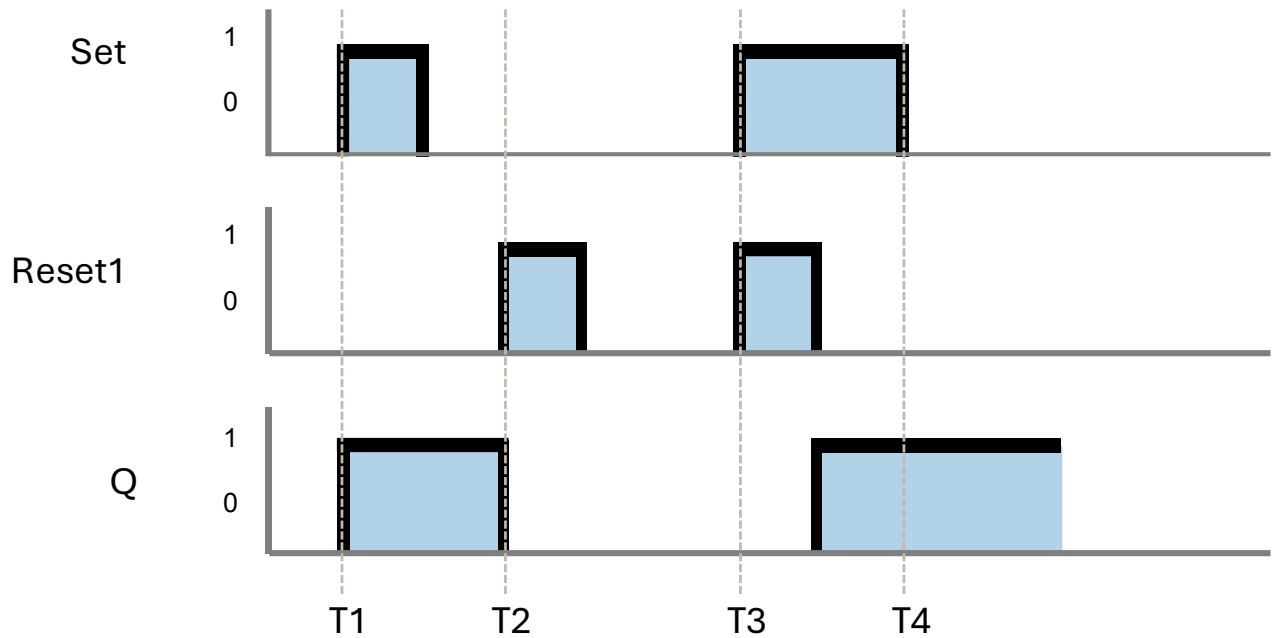


Bild 1 Impulsdiagramm - Flipflop dominant rücksetzend

T1

Wenn der Signalzustand am Eingang Set "1" und am Eingang Reset1 "0" ist, wird der angegebene Operand auf "1" gesetzt.

T2

Wenn der Signalzustand am Eingang Set "0" und am Eingang Reset1 "1" ist, wird der angegebene Operand auf "0" zurückgesetzt.

T3

Der Eingang Reset1 dominiert den Eingang Set. Bei einem Signalzustand "1" an beiden Eingängen Set und Reset1 wird der Signalzustand des angegebenen Operanden auf "0" zurückgesetzt.

T4

Bei einem Signalzustand "0" an beiden Eingängen Set und Reset1 wird die Anweisung nicht ausgeführt. Der Signalzustand des Operanden bleibt in diesem Fall unverändert.

Bei **Siemens** entspricht das dominant rücksetzende Flipflop der Anweisung "SR". Oberhalb des Bausteinaufrufes muss eine Variable vom Datentyp "BOOL", als Speicher, verschalten werden.

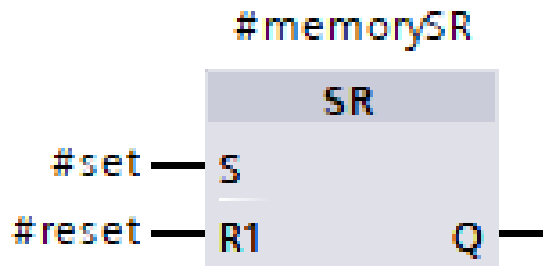


Bild 2 Flipflop dominant rücksetzend - Siemens

Bei **CODESYS** und **Beckhoff** wird dies durch die Anweisung "RS" realisiert. Oberhalb des Bausteinaufrufes muss eine Instanz vom Datentyp "RS" verschaltet werden, da es sich um einen Funktionsbausteinaufruf handelt.

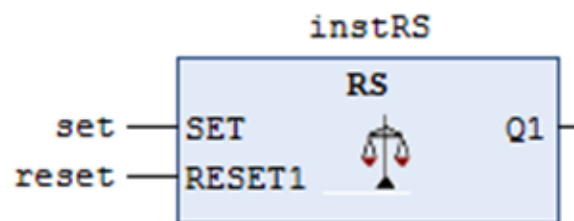


Bild 3 Flipflop dominant rücksetzend - Beckhoff

In der Textuellen Programmiersprache ST bieten Steuerungshersteller, wie **Beckhoff** oder **CODESYS** die Möglichkeit, das RS Glied, als Rücksetz-Dominantes Flipflop ebenfalls als Funktionsbaustein aufzurufen. Hierfür ist genau so wie in FUP, eine Instanz, vom Typ "RS", in der Bausteinschnittstelle zu deklarieren, welche dann im Implementierungsteil aufgerufen wird.

//Deklarationsteil, Bausteinschnittstelle

VAR

instRS : RS; //Deklaration der Instanzdaten (Multiinstanz)

END_VAR

//Anweisungsteil, Aufruf der Instanz

**instRS(SET := varBoolSet ,
RESET1 := varBoolReset,
Q1 => varBoolQ);**

Bild 4 ST-Anweisung - dominant rücksetzend

Bei **Siemenssteuerungen** besteht diese Möglichkeit nicht. Hier muss das Flipflop mittels IF-Anweisung ausprogrammiert werden.

```
// IF-Anweisung
IF Reset1 THEN
  Q := false;
ELSIF Set THEN
  Q := true;
END_IF;
```

Bild 5 IF-Anweisung - dominant rücksetzend



Die Funktionsweise der IF-Anweisung wird in diesem Kapitel unter "7.5 IF-Anweisung [ST/SCL]" detailliert beschrieben.

Alternativ lässt sich dies auch mittels logischer Verknüpfung einer Selbsthaltung umsetzen.

```
//Logische Verknüpfung
Q := NOT Reset1 AND (Q OR Set);
```

Bild 6 Logische Verknüpfung - dominant rücksetzend

7.1.2 Dominant setzendes Flipflop

Mit der Anweisung rücksetzen oder setzen Sie das Bit eines angegebenen Operanden, abhängig vom Signalzustand an den Eingängen Reset und Set1. Der aktuelle Signalzustand des Operanden wird auf den Ausgang Q übertragen und kann an diesem abgefragt werden.

Das Dominanzverhalten wird durch die "1" am Set Eingang gekennzeichnet.

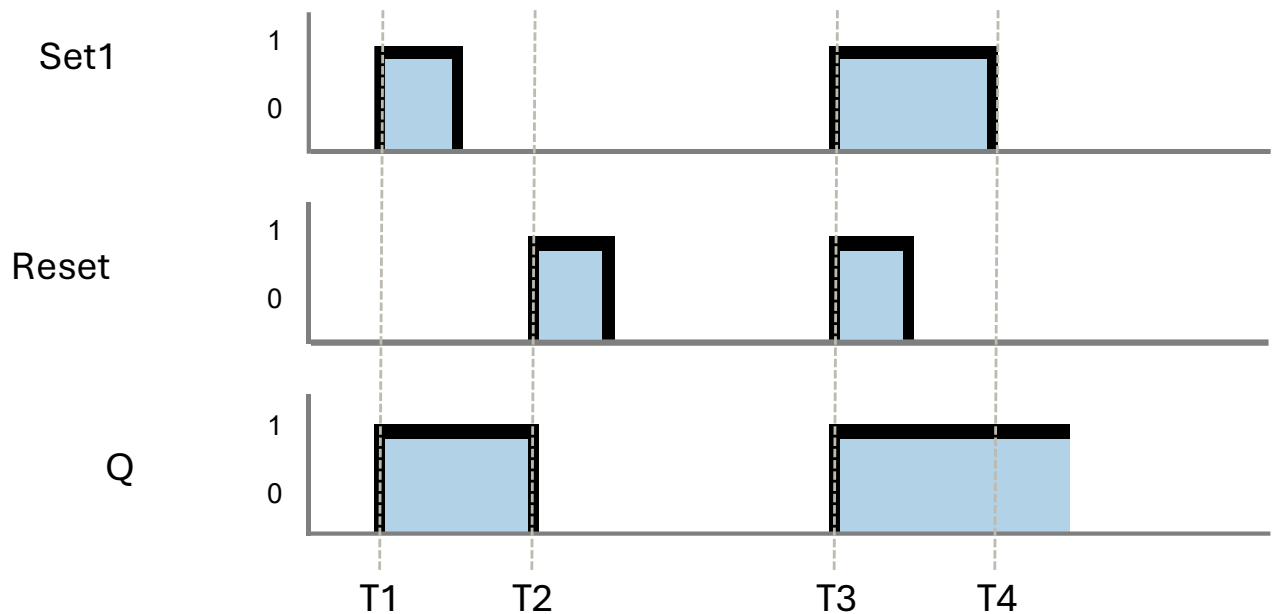


Bild 7 Impulsdiagramm - Flipflop dominant setzend

T1

Wenn der Signalzustand am Eingang Set1 "1" und am Eingang Reset "0" ist, wird der angegebene Operand auf "1" gesetzt.

T2

Wenn der Signalzustand am Eingang Reset "1" und am Eingang Set1 "0" ist, wird der angegebene Operand auf "0" zurückgesetzt.

T3

Der Eingang Set1 dominiert den Eingang Reset. Bei einem Signalzustand "1" an beiden Eingängen, Reset und Set1, wird der Signalzustand des angegebenen Operanden auf "1" gesetzt.

T4

Bei einem Signalzustand "0" an beiden Eingängen Reset und Set1 wird die Anweisung nicht ausgeführt. Der Signalzustand des Operanden bleibt in diesem Fall unverändert.

Bei **Siemens** entspricht das dominant setzende Flipflop der Anweisung "RS". Oberhalb des Bausteinaufrufes muss eine Variable vom Datentyp "BOOL", als Speicher, verschalten werden.

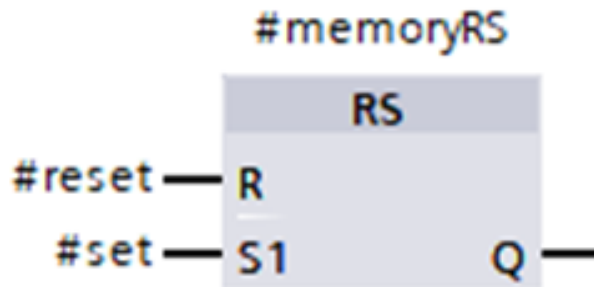


Bild 8 Flipflop dominant setzend - Siemens

Bei **CODESYS** und **Beckhoff** wird dies durch die Anweisung "SR" realisiert. Oberhalb des Bausteinaufrufes muss eine Instanz vom Datentyp "SR" verschaltet werden, da es sich um einen Funktionsbausteinaufruf handelt.

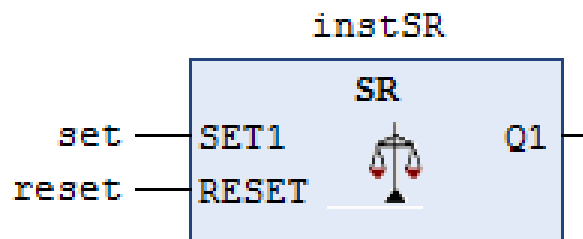


Bild 9 Flipflop dominant setzend - Beckhoff

In der Textuellen Programmiersprache ST bieten Steuerungshersteller, wie **Beckhoff** oder **CODESYS** die Möglichkeit, das SR Glied, als Setz-Dominantes Flipflop ebenfalls als Funktionsbaustein aufzurufen. Hierfür ist genau so wie in FUP, eine Instanz, vom Typ "SR", in der Bausteinschnittstelle zu deklarieren, welche dann im Implementierungsteil aufgerufen wird.

//Deklarationsteil, Bausteinschnittstelle

VAR

instSR : SR; //Deklaration der Instanzdaten (Multiinstanz)

END_VAR

//Anweisungsteil, Aufruf der Instanz

```
instSR(SET1 := varBoolSet,
      RESET := varBoolReset,
      Q1 => varBoolQ);
```

Bild 10 ST-Anweisung - dominant setzend

Bei **Siemenssteuerungen** besteht diese Möglichkeit nicht. Hier muss das Flipflop mittels IF-Anweisung ausprogrammiert werden.

```
//IF-Anweisung  
IF Set1 THEN  
    Q := true;  
ELSIF Reset THEN  
    Q := false;  
END_IF;
```

Bild 11 IF-Anweisung - dominant setzend



Die Funktionsweise der IF-Anweisung wird in diesem Kapitel unter "7.5 IF-Anweisung [ST/SCL]" detailliert beschrieben.

Alternativ lässt sich dies auch mittels logischer Verknüpfung einer Selbsthaltung umsetzen.

```
//Logische Verknüpfung  
Q := (NOT Reset AND Q) OR Set1;
```

Bild 12 Logische Verknüpfung - dominant setzend



7.1.3 Übung - Dominanzverhalten

Ziel

In dieser Übung soll das Dominanzverhalten der beiden Flipflops SR und RS in einem Test-Funktionsbaustein programmiert und praktisch kennengelernt werden.

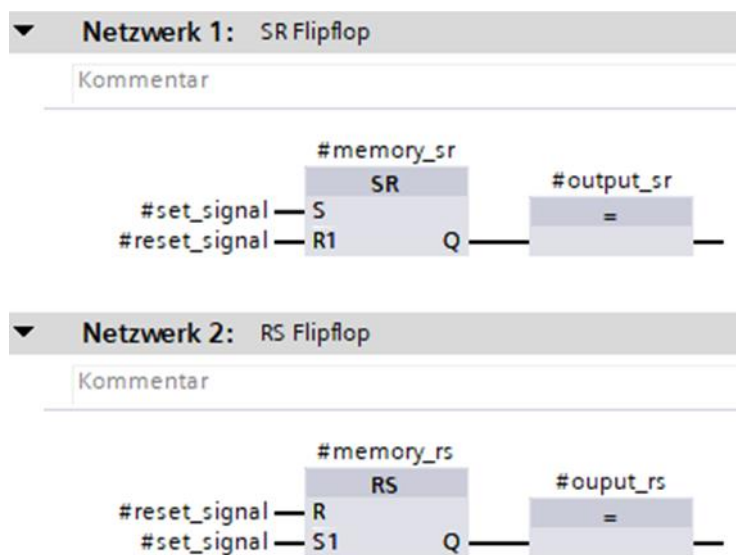
Über den Eingangsparameter "set_signal" wird der Setzeingang der beiden Speicherglieder angesteuert. Der Parameter "reset_signal" ist für das Rücksetzen zuständig. Der Zustand des SR Flipflops wird über den Ausgangsparameter "output_sr" angezeigt, der es RS Flipflops über den Ausgang "output_rs".

Aufgabe

- Erstellen Sie einen Funktionsbaustein mit folgender Bausteinschnittstelle:



	Name	Datentyp	Kommentar
1	Input		
2	set_signal	Bool	Set
3	reset_signal	Bool	Reset
4	Output		
5	output_sr	Bool	Output SR
6	ouput_rs	Bool	Output RS
7	InOut		
8	<Hinzufügen>		
9	Static		
10	memory_sr	Bool	Internal Memory SR
11	memory_rs	Bool	Internal Memory RS
12	Temp		
13	<Hinzufügen>		
14	Constant		
15	<Hinzufügen>		

- Es soll folgendes Programm umgesetzt werden:



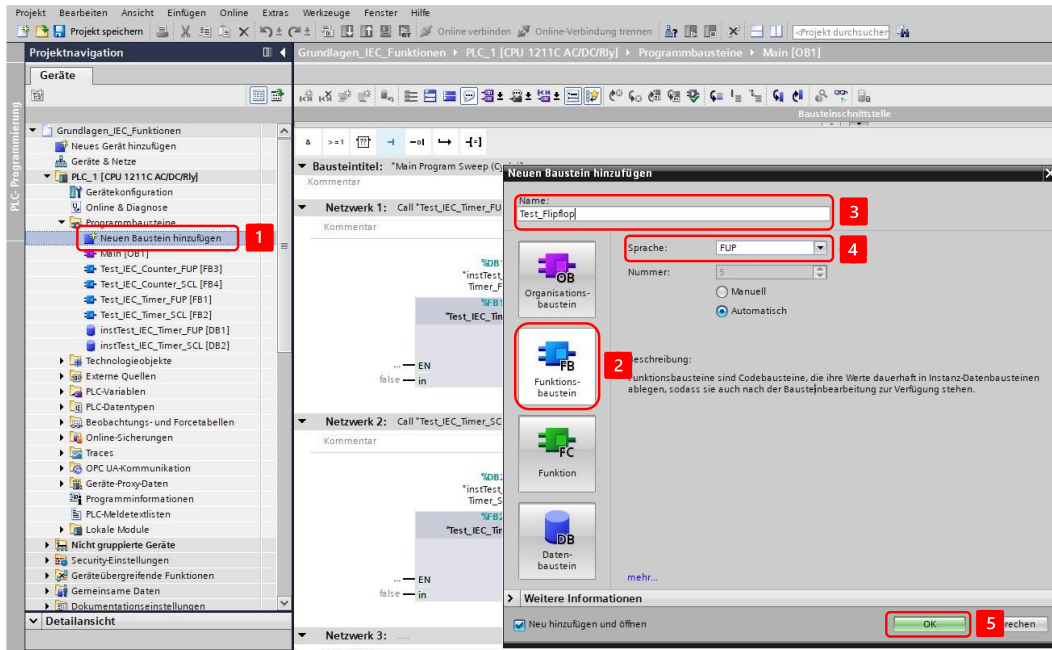
- Rufen Sie den erstellten Baustein im "MAIN" auf.
- Über die Instanz können die Eingangsvariablen gesteuert und die Ausgangsvariablen beobachtet werden. Alternativ, falls vorhanden, können diese über die Bausteinschnittstelle beim Aufruf auch mit Tastern und Anzeigeelementen verschaltet werden.

Wird jeweils nur der Eingang "set_signal" oder "reset_signal" betätigt, verhalten sich beide Speicherglieder identisch. Werden "set_signal" und "reset_signal" zeitgleich auf "TRUE" gesetzt, unterscheiden sich die Ausgänge "output_sr" und "output_rs" in ihrem Signalzustand aufgrund des unterschiedlichen Dominanzverhaltens.

-  Alternativ kann auch der vorbereitete Baustein "Test_Flipflop [FB5]" aus dem Vorlageprojekt "Grundlagen_Programmanweisungen.zap17" verwendet werden.
-  Zusätzliche Hilfestellung zur Interpretation des Programmstatus kann das Kapitel "Inbetriebnahme (Software)" liefern.

Vorgehensweise:

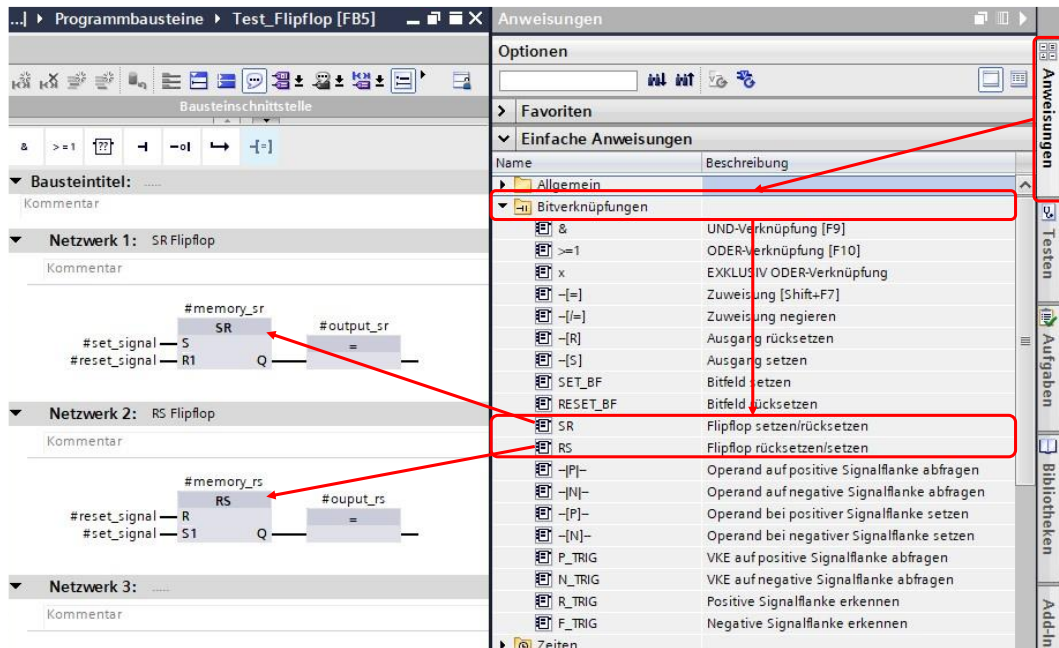
1. Erstellen Sie einen neuen Funktionsbaustein, wählen die gewünschte Programmiersprache und vergeben einen aussagekräftigen Namen:



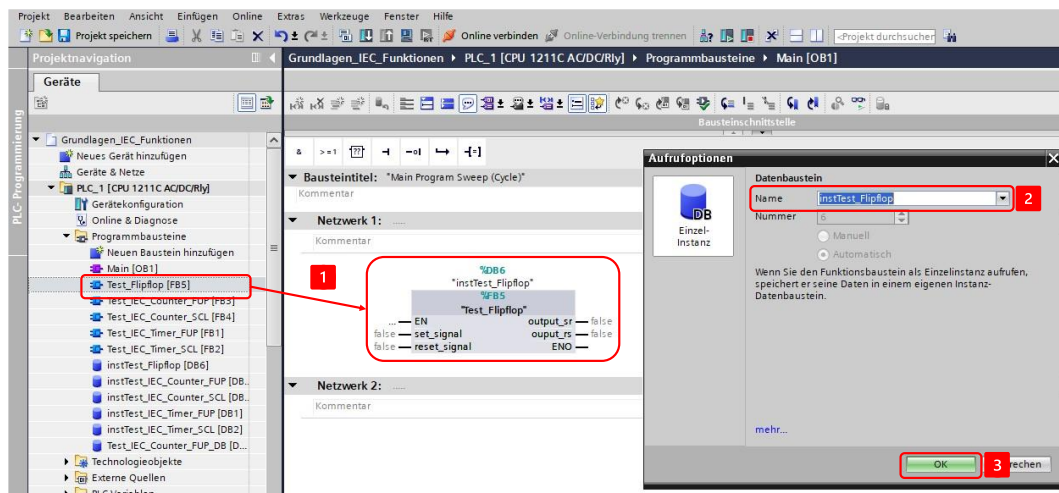
2. Deklarieren Sie folgende Variablen in der Bausteinschnittstelle:

	Name	Datentyp	Kommentar
1	Input		
2	set_signal	Bool	Set
3	reset_signal	Bool	Reset
4	Output		
5	output_sr	Bool	Output SR
6	ouput_rs	Bool	Output RS
7	InOut		
8	<Hinzufügen>		
9	Static		
10	memory_sr	Bool	Internal Memory SR
11	memory_rs	Bool	Internal Memory RS
12	Temp		
13	<Hinzufügen>		
14	Constant		
15	<Hinzufügen>		

- Setzen Sie folgendes Programm um, die Speicherfunktionen finden Sie in der TaskCard unter "Anweisungen" → "Einfache Anweisungen" → "Bitverknüpfungen". Diese können mittels "Drag & Drop" in den Arbeitsbereich gezogen werden:



- Rufen Sie den Funktionsbaustein im "MAIN" auf, und erstellen Sie eine Instanz.



- Steuern Sie über die Instanz die Eingangsvariable "set_signal" und "reset_signal" zunächst nacheinander auf "TRUE", anschließend steuern Sie auch beide Eingangsvariablen zeitgleich auf "TRUE" und beobachten Sie die Ausgangsvariablen "output_sr" und "output_rs":

The screenshot shows the LabVIEW interface with the 'instTest_Flipflop' instance selected. The 'Suchen und ersetzen' (Find and Replace) dialog is open, and the 'Steuern' (Control) dialog is also visible. The 'set_signal' and 'reset_signal' inputs are highlighted with red boxes and numbered 1 and 2 respectively. The 'Steuern' dialog shows the 'set_signal' input being set to 'true' with a red box and number 3. The 'OK' button is highlighted with a red box and number 4.

The screenshot shows the LabVIEW interface with the 'instTest_Flipflop' instance selected. The 'set_signal' and 'reset_signal' inputs are highlighted with red boxes. The 'output_sr' and 'output_rs' outputs are also highlighted with red boxes. The 'memory_sr' and 'memory_rs' static inputs are highlighted with red boxes.

7.2 Flanken

Flanken in der SPS-Programmierung sind wichtig, um bestimmte Ereignisse zu erkennen, die bei einer Änderung des Signals auftreten. Bei einer Flankenauswertung wird erfasst, ob sich der Zustand eines binären Signals im Vergleich zum vorherigen Programmzyklus verändert hat.

Nach IEC 61131 stehen folgende Flanken zur Verfügung:

- Positive Flanke (R_TRIG)
- Negative Flanke (F_TRIG)

Flanken werden zum Beispiel benötigt zur:

Ereigniserkennung:

Flanken ermöglichen es, spezifische Ereignisse zu erkennen, die genau beim Übergang des Signals auftreten. Dies ist nützlich, um Aktionen genau dann auszulösen, wenn ein Signal sich ändert, nicht während es in einem bestimmten Zustand verharrt.

Taktgesteuerte Prozesse:

In vielen Anwendungen ist es notwendig, Prozesse synchron zu bestimmten Signaländerungen zu starten.

Ereignisgesteuerte Aktionen:

In der Automatisierungstechnik müssen häufig bestimmte Aktionen genau zu dem Zeitpunkt gestartet werden, wenn ein Schalter betätigt wird oder ein Sensor ein Ereignis erkennt. Flankenerkennung ermöglicht dies präzise und verhindert Fehlfunktionen.

Interrupts und Timings:

In Echtzeitanwendungen können Flanken zur Erzeugung von Interrupts verwendet werden, um auf externe Ereignisse sofort zu reagieren. Sie ermöglichen auch präzise Timings und Zeitmessungen.

Beim Signalwechsel (Flanke) eines binären Signals gibt die Flankenauswertung einen Impuls aus. Dieser Impuls steht nur für einen Zyklus an.

Um den Signalwechsel (Flanke) zu erkennen, wird der aktuelle Zustand (Ist-Zustand) mit dem vorherigen Zustand (Vergangenheitswert) des Signals verglichen. In jedem Programmzyklus erfolgt ein Vergleich des alten Signalzustandes (Vergangenheitswert) mit dem aktuellen Signalzustand. Dazu benötigt die Auswertung einen Speicher (Instanz).

7.2.1 Positive Flanke erkennen R_TRIG

Mit der IEC-Anweisung "R_TRIG", positive Signalflanke erkennen, können Sie eine Zustandsänderung von "0" auf "1" am Eingang CLK erkennen.

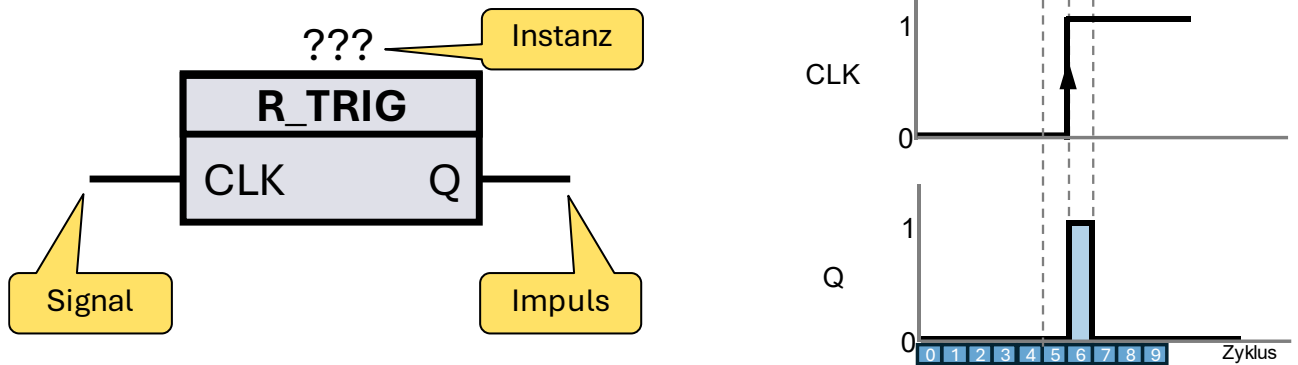


Bild 13 FUP-Anweisung R_TRIG und Impulsdiagramm

Aktueller Zyklus 6

Die Anweisung vergleicht den aktuellen Wert am Eingang CLK (Zyklus 6) mit dem Zustand der vorherigen Abfrage (Flankenspeicher, Zyklus 5), welcher in der Instanz gespeichert ist.

- Wenn CLK "1" ist und der Flankenspeicher "0" ist wurde eine positive Flanke erkannt.
- Wenn die Anweisung einen Zustandswechsel am Eingang CLK von "0" auf "1" erkannt hat, wird der Ausgang Q auf "1" gesetzt.
- Der Flankenspeicher wird auf den Signalzustand des aktuellen Signals gesetzt. Flankenspeicher "1".

Aktueller Zyklus 7

Die Anweisung vergleicht den aktuellen Wert am Eingang CLK (Zyklus 7) mit dem Zustand der vorherigen Abfrage (Flankenspeicher, Zyklus 6), welcher in der Instanz gespeichert ist.

- Wenn CLK "1" ist und der Flankenspeicher "1" ist wurde keine Flanke erkannt.
- Wenn die Anweisung keinen Signalwechsel am Eingang CLK von "0" auf "1" erkannt hat, wird der Ausgang Q auf "0" gesetzt.
- Der Flankenspeicher wird auf den Signalzustand des aktuellen Signals gesetzt. Flankenspeicher "1".

Wenn die Anweisung einen Signalwechsel am Eingang CLK von "0" auf "1" erkennt, wird am Ausgang Q ein Impuls erzeugt, d. h. der Ausgang führt für einen Zyklus lang den Wert "1".

In allen anderen Fällen ist der Signalzustand am Ausgang der Anweisung "0".

Jedem Aufruf der Anweisung "Positive Signalflanke erkennen" muss eine Instanz vom Datentyp "R_TRIG" zugeordnet werden, in der die Instanzdaten gespeichert werden.

Textuelle Umsetzung positive Signalflanke erkennen (R_TRIG)

Die Instanziierung des Flankenbausteins R_TRIG erfolgt ähnlich wie bei einem Funktionsbaustein. Für jeden Aufruf des Funktionsbausteins "R_TRIG" wird ein eigener Speicherbereich benötigt. Die Instanzdaten können entweder als Einzelinstanz oder als Multiinstanz (in der Bausteinschnittstelle) angelegt werden.

Beispiel:

//Deklarationsteil, Bausteinschnittstelle

VAR

instRtrig : R_TRIG; **//Deklaration der Instanzdaten (Multiinstanz)**

END_VAR

//Anweisungsteil, Aufruf der Instanz

instRtrig(CLK := varBoolClk,
Q => varBoolQ);

Bild 14 ST/SCL-Anweisung R_TRIG

7.2.2 Negative Flanke erkennen F_TRIG

Mit der Anweisung "Negative Signalflanke erkennen" können Sie eine Zustandsänderung von "1" auf "0" am Eingang CLK erkennen.

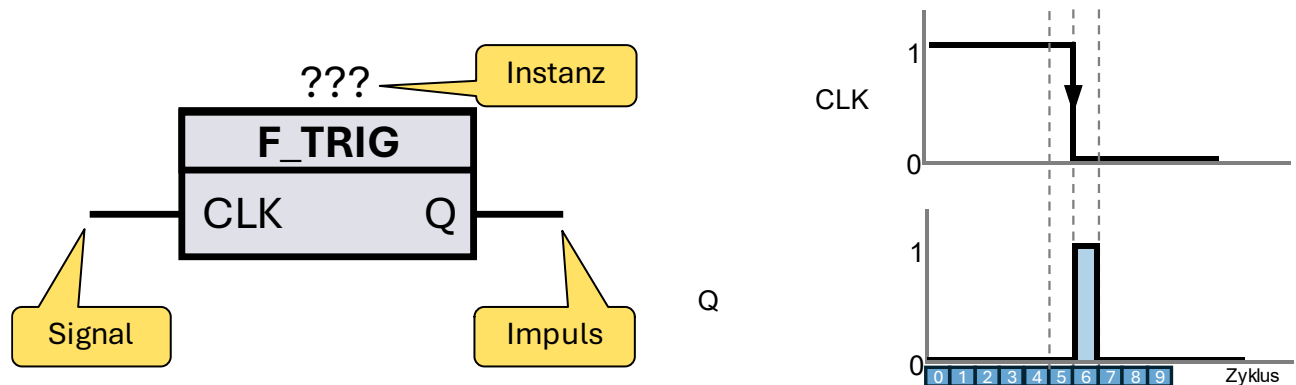


Bild 15 FUP-Anweisung F_TRIG und Impulsdiagramm

Aktueller Zyklus 6

Die Anweisung vergleicht den aktuellen Wert am Eingang CLK (Zyklus 6) mit dem Zustand der vorherigen Abfrage (Flankenspeicher, Zyklus 5), welcher in der Instanz gespeichert ist.

- Wenn CLK "0" ist und der Flankenspeicher "1" ist wurde eine negative Flanke erkannt.
- Wenn die Anweisung einen Signalwechsel am Eingang CLK von "1" auf "0" erkannt hat, wird der Ausgang Q auf "1" gesetzt.
- Der Flankenspeicher wird auf den Signalzustand des aktuellen Signals gesetzt. Flankenspeicher "0".

Aktueller Zyklus 7

Die Anweisung vergleicht den aktuellen Wert am Eingang CLK (Zyklus 7) mit dem Zustand der vorherigen Abfrage (Flankenspeicher, Zyklus 6), welcher in der Instanz gespeichert ist.

- Wenn CLK "0" ist und der Flankenspeicher "0" ist wurde keine Flanke erkannt.
- Wenn die Anweisung keinen Signalwechsel am Eingang CLK von "1" auf "0" erkannt hat, wird der Ausgang Q auf "0" gesetzt.
- Der Flankenspeicher wird auf den Signalzustand des aktuellen Signals gesetzt. Flankenspeicher "0".

Wenn die Anweisung einen Zustandswechsel am Eingang CLK von "1" auf "0" erkennt, wird am Ausgang Q ein Impuls erzeugt, d. h. der Ausgang führt einen Zyklus lang den Wert "1".

In allen anderen Fällen ist der Signalzustand am Ausgang der Anweisung "0".

Jedem Aufruf der Anweisung "Negative Signalflanke erkennen" muss eine Instanz vom Datentyp "F_TRIG" zugeordnet werden, in der die Instanzdaten gespeichert werden.

Textuelle Umsetzung negative Signalflanke erkennen (F_TRIG)

Die Instanziierung des Flankenbausteins F_TRIG erfolgt ähnlich wie bei einem Funktionsbaustein. Für jeden Aufruf des Funktionsbausteins "F_TRIG" wird ein eigener Speicherbereich benötigt. Die Instanzdaten können entweder als Einzelinstanz oder als Multiinstanz (in der Bausteinschnittstelle) angelegt werden.

Beispiel:

//Deklarationsteil, Bausteinschnittstelle

VAR

instFtrig : F_TRIG; **//Deklaration der Instanzdaten (Multiinstanz)**

END_VAR

//Anweisungsteil, Aufruf der Instanz

instFtrig(CLK := varBoolClk,
Q => varBoolQ);

Bild 16 ST/SCL-Anweisung F_TRIG



Verhalten nach CPU-Anlauf

Die Norm IEC 61131 beschreibt, dass die Anweisung "F_TRIG" den Ausgang "Q" einen Zyklus lang auf TRUE setzt, wenn beim CPU-Anlauf der Eingang "CLK" den Wert FALSE hat.

7.3 Zeiten

Zeitfunktionen können genutzt werden, um zeitgesteuerte Aktionen im Programm auszulösen. Sie werden durch ein binäres Signal getriggert. Das Ergebnis der Auswertung ist ebenfalls ein binäres Signal. Zeitfunktionen sind Anweisungen, die eine Instanz besitzen.

Nach IEC 61131 stehen folgende Zeitfunktionen zur Verfügung:

- Impulserzeugung (TP)
- Einschaltverzögerung (TON)
- Ausschaltverzögerung (TOF)

Der maximal darstellbare Zeitbereich ist abhängig vom gewählten Datentyp.

- Format TIME (32-Bit) maximaler Zeitbereich: ~24 Tage
- Format LTIME (64-Bit) maximaler Zeitbereich: ~106751 Tage (292 Jahre)

Im TIA-Portal kann der Datentyp mittels Dropdownmenü direkt an der Anweisung eingestellt werden.

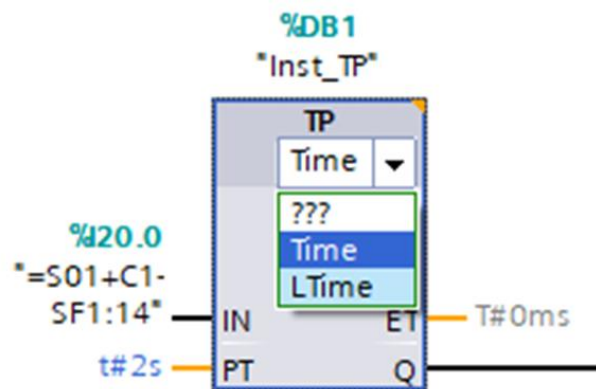


Bild 17 Datentyp der Zeitfunktion (Siemens)

Bei Codesys / Beckhoff stehen für 64-Bit Zeitoperationen eigene Funktionsbausteine zur Verfügung:

- LTP
- LTON
- LTOF

Beispiele für Zeitwerte:

Syntax	Bedeutung
T#5s	5 Sekunden
T#1d3h5m30s500ms	1 Tag, 3 Stunden, 5 Minuten, 30 Sekunden und 500 Millisekunden
LTIME#50d3h	50 Tage und 3 Stunden

Hier sind einige wichtige Anwendungen und der Nutzen von Zeitfunktionen in der SPS-Programmierung:

Verzögerungen steuern:

Zeitfunktionen ermöglichen das Einführen von Verzögerungen in Steuerungsprozesse. Beispielsweise kann das Starten eines Motors verzögert nach der Auslösung eines Triggersignals erfolgen.

Sequenzsteuerung:

Wenn Prozesse in bestimmten Zeitintervallen ausgeführt werden müssen, sorgen Zeitfunktionen dafür, dass jeder Schritt so lange wie definiert aktiviert wird.

Überwachung von Zeitbedingungen:

Zeitfunktionen helfen bei der Überwachung von Prozessen, indem sie sicherstellen, dass bestimmte Aktionen innerhalb festgelegter Zeitlimits abgeschlossen werden.

Impulsgenerierung:

In vielen Anwendungen werden pulsierende Signale benötigt. Zeitfunktionen erzeugen diese Impulse in vordefinierten Intervallen.

Wartezeiten implementieren:

Zeitfunktionen sind nützlich, um Wartezeiten einzurichten, beispielsweise um die Maschine nicht zu überlasten.

7.3.1 Einschaltverzögerung TON

Mit der Anweisung "Einschaltverzögerung erzeugen" wird das Setzen des Ausgangs Q (Output) um die parametrisierte Zeitdauer PT (Preset Time) verzögert. Am Ausgang ET (Elapsed Time) kann der aktuelle Zeitwert abgefragt werden. Der Zeitwert beginnt bei T#0s und endet, wenn die Zeitdauer PT erreicht ist. Sobald der Signalzustand am Eingang IN (Input) auf "0" wechselt, wird der Ausgang ET zurückgesetzt.

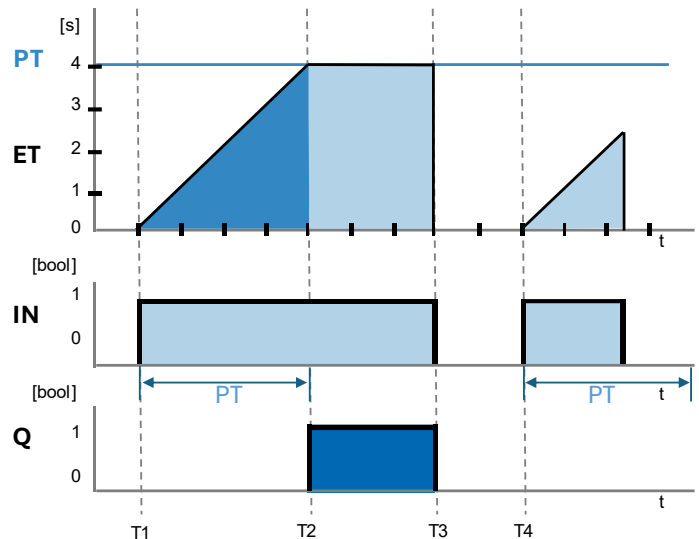
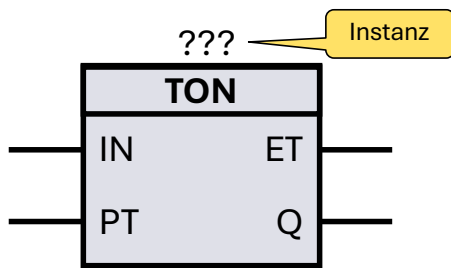


Bild 18 FUP-Anweisung TON und Impulsdiagramm

T1

Die Anweisung startet, wenn das Verknüpfungsergebnis (VKE) am Eingang IN von "0" auf "1" wechselt (positive Signalfanke). Ab diesem Moment beginnt die programmierte Zeitdauer PT zu laufen.

T2

Sobald die Zeitdauer PT abgelaufen ist, wird der Ausgang Q auf den Signalzustand "1" gesetzt.

T3

Q bleibt gesetzt, solange der Starteingang "1" führt. Wechselt der Signalzustand am Starteingang von "1" auf "0", wird der Ausgang Q zurückgesetzt.

T4

Eine neue positive Signalfanke am Starteingang startet die Zeitfunktion erneut.

Jedem Aufruf der Anweisung "Einschaltverzögerung erzeugen" muss eine Instanz vom Datentyp "TON" zugeordnet werden, in der die Instanzdaten gespeichert werden.

Textuelle Umsetzung Einschaltverzögerung erzeugen (TON)

Die Instanziierung der Einschaltverzögerung TON erfolgt ähnlich wie bei einem Funktionsbaustein. Für jede Instanz des Funktionsbausteins "TON" wird ein eigener Speicherbereich benötigt. Die Instanzdaten können entweder als Einzelinstanz oder als Multiinstanz (in der Bausteinschnittstelle) angelegt werden.

Beispiel:

//Deklarationsteil, Bausteinschnittstelle

VAR

instTon : TON; **//Deklaration der Instanzdaten (Multiinstanz)**

END_VAR

//Anweisungsteil, Aufruf der Instanz

```
instTon(IN := varBoolIn,
        PT := varTimePt,
        Q => varBoolQ,
        ET => varTimeEt);
```

Bild 19 ST/SCL-Anweisung TON

7.3.2 Ausschaltverzögerung TOF

Mit der Anweisung "Ausschaltverzögerung erzeugen" verzögern Sie das Rücksetzen des Ausgangs Q (Output) um die programmierte Zeitdauer PT (Preset Time). Am Ausgang ET (Elapsed Time) kann der aktuelle Zeitwert abgefragt werden. Der Zeitwert startet bei T#0s und endet, wenn die Zeitdauer PT erreicht ist. Nach Ablauf der Zeitdauer PT bleibt der Ausgang ET auf dem aktuellen Wert stehen, bis der Eingang IN wieder auf "1" wechselt. Wenn der Eingang IN (Input) vor Ablauf der Zeitdauer PT auf "1" wechselt, wird der Ausgang ET auf T#0s zurückgesetzt.

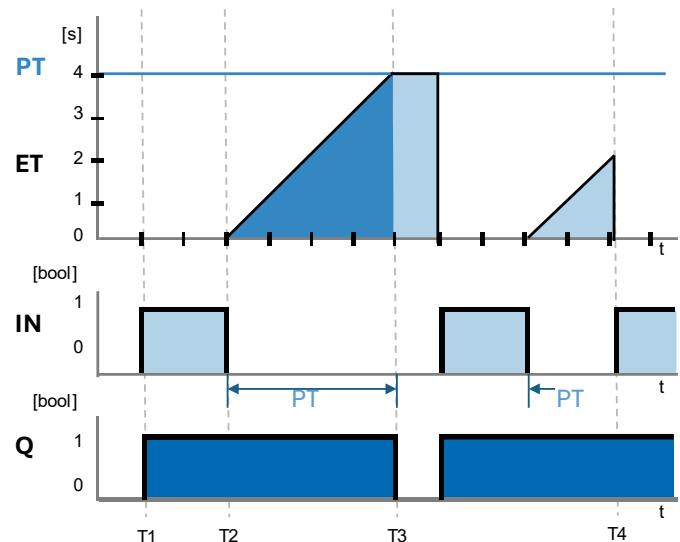
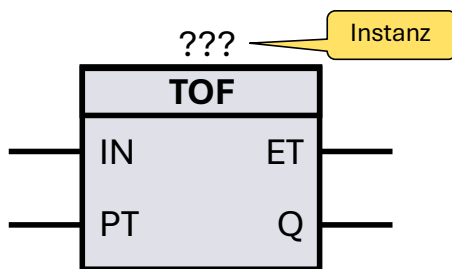


Bild 20 FUP-Anweisung TOF und Impulsdiagramm

T1

Der Ausgang Q wird aktiviert, wenn das Verknüpfungsergebnis (VKE) am Eingang IN von "0" auf "1" wechselt (positive Flanke).

T2

Wenn der Eingang IN wieder auf "0" wechselt (negative Flanke), beginnt die programmierte Zeitdauer PT zu laufen. Der Ausgang Q bleibt aktiviert, solange die Zeitdauer PT läuft.

T3

Nach Ablauf von PT wird der Ausgang Q zurückgesetzt.

T4

Wechselt das Eingangssignal IN auf "1", bevor die Zeitdauer PT abgelaufen ist, wird die Zeit zurückgesetzt und der Ausgang Q bleibt weiterhin aktiviert.

Jedem Aufruf der Anweisung "Ausschaltverzögerung erzeugen" muss eine Instanz vom Datentyp "TOF" zugeordnet werden, in der die Instanzdaten gespeichert werden.

Textuelle Umsetzung Ausschaltverzögerung erzeugen (TOF)

Die Instanziierung der Ausschaltverzögerung TOF erfolgt ähnlich wie bei einem Funktionsbaustein. Für jeden Aufruf des Funktionsbausteins "TOF" wird ein eigener Speicherbereich benötigt. Die Instanzdaten können entweder als Einzelinstanz oder als Multiinstanz (in der Bausteinschnittstelle) angelegt werden.

Beispiel:

//Deklarationsteil, Bausteinschnittstelle

VAR

instTof : TOF; **//Deklaration der Instanzdaten (Multiinstanz)**

END_VAR

//Anweisungsteil, Aufruf der Instanz

```
instTof(IN := varBoolIn,
        PT := varTimePt,
        Q => varBoolQ,
        ET => varTimeEt);
```

Bild 21 ST/SCL-Anweisung TOF

7.3.3 Impuls TP

Mit der Anweisung "Impuls erzeugen" aktivieren Sie den Ausgang Q (Output) für eine programmierte Zeitdauer. Am Ausgang ET (Elapsed Time) können Sie den aktuellen Zeitwert abfragen. Der Zeitwert startet bei T#0s und endet, wenn der Wert von PT (Preset Time) erreicht ist. Wenn PT abgelaufen ist und das Eingangssignal IN (Input) "0" beträgt, wird der Ausgang ET zurückgesetzt.

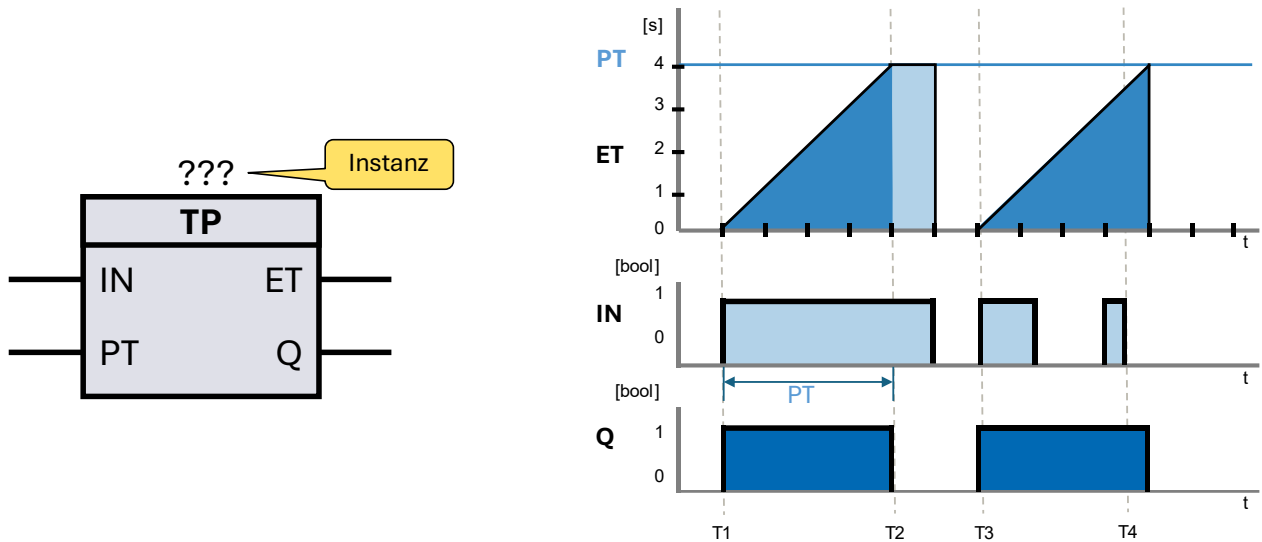


Bild 22 FUP-Anweisung TP und Impulsdiagramm

T1

Wenn der Eingang IN von "0" auf "1" wechselt beginnt die programmierte Zeitdauer PT zu laufen und der Ausgang Q wechselt auf "1".

T2

Nachdem die Zeitdauer PT abgelaufen ist, wechselt der Ausgang Q von "1" auf "0", unabhängig davon das das Eingangssignal IN noch "1" ist.

T3

Wenn die Zeitdauer PT abgelaufen ist und der Eingang IN von "0" auf "1" wechselt, beginnt die programmierte Zeitdauer PT erneut zu laufen und der Ausgang Q wechselt auf "1".

T4

Der Ausgang Q bleibt für die Dauer von PT aktiviert, unabhängig davon, wie sich das Eingangssignal weiter verhält. Während PT läuft, hat eine neue positive Flanke am Eingang IN keinen Einfluss auf den Ausgang Q und der abgelaufen Zeit PT.

Jedem Aufruf der Anweisung "Impuls erzeugen" muss eine Instanz vom Datentyp "TP" zugeordnet werden, in der die Instanzdaten gespeichert werden.

Textuelle Umsetzung Impuls erzeugen (TP)

Die Instanziierung des Impulsbausteins TP erfolgt ähnlich wie bei einem Funktionsbaustein. Für jeden Aufruf des Funktionsbausteins "TP" wird ein eigener Speicherbereich benötigt. Die Instanzdaten können entweder als Einzelinstanz oder als Multiinstanz (in der Bausteinschnittstelle) angelegt werden.

Beispiel:

//Deklarationsteil, Bausteinschnittstelle

VAR

instTp : TP; //Deklaration der Instanzdaten (Multiinstanz)

END_VAR

//Anweisungsteil, Aufruf der Instanz

```
instTp(IN := varBoolIn,
      PT := varTimePt,
      Q => varBoolQ,
      ET => varTimeEt);
```

Bild 23 ST/SCL-Anweisung TP



7.3.4 Übung - IEC Zeitfunktionen

Ziel

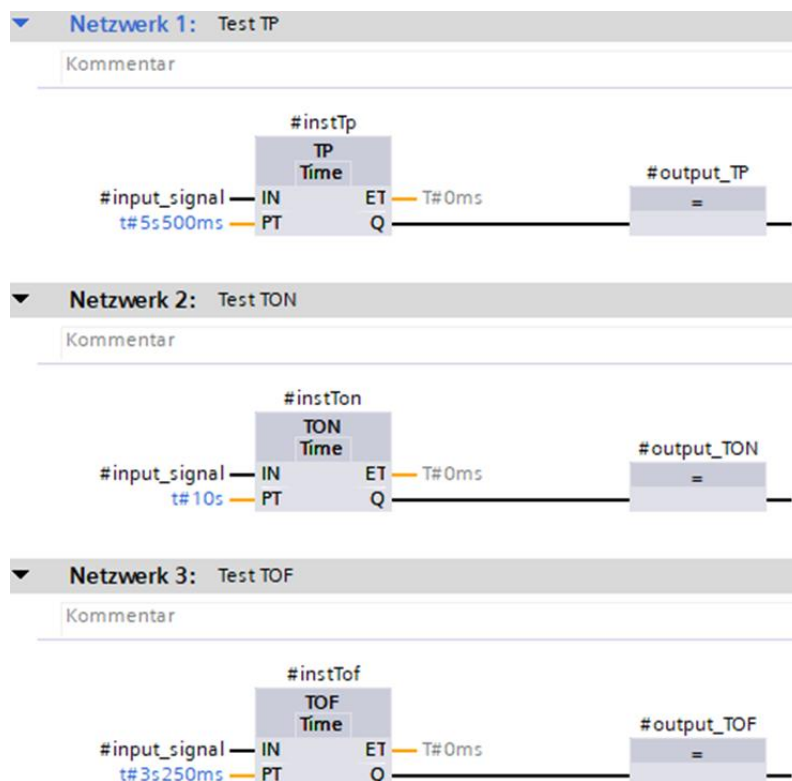
In dieser Übung sollen die 3 IEC Zeitfunktionen (TP, TON, TOF) in einem Test-Funktionsbaustein programmiert und praktisch kennengelernt werden. Über den Eingangsparameter "input_signal" können die Zeitfunktionen gestartet werden. Der Zustand wird über die Ausgangsparameter "output_TP", "output_TON", "output_TOF" angezeigt.

Aufgabe

- Erstellen Sie einen Funktionsbaustein mit folgender Bausteinschnittstelle:

	Name	Datentyp	Kommentar
1	▼ Input		
2	input_signal	Bool	Test Input
3	▼ Output		
4	output_TP	Bool	Test Output TP
5	output_TON	Bool	Test Output TON
6	output_TOF	Bool	Test Output TOF
7	▼ InOut		
8	<Hinzufügen>		
9	▼ Static		
10	▶ instTp	TP_TIME	Instance TP
11	▶ instTon	TON_TIME	Instance TON
12	▶ instTof	TOF_TIME	Instance TOF
13	▼ Temp		

- Abhängig der gewählten Programmiersprache soll folgendes Programm umgesetzt werden:





```

1  //Test TP
2  ▢ #instTp(IN := #input_signal,
3    PT := t#5s500ms,
4    //ET => ,
5    Q => #output_TP);
6
7  //Test TON
8  ▢ #instTon(IN := #input_signal,
9    PT := t#10s,
10   //ET => ,
11   Q => #output_TON);
12
13
14 //Test TOF
15 ▢ #instTof(IN := #input_signal,
16   PT := t#3s250ms,
17   //ET => ,
18   Q => #output_TOF);

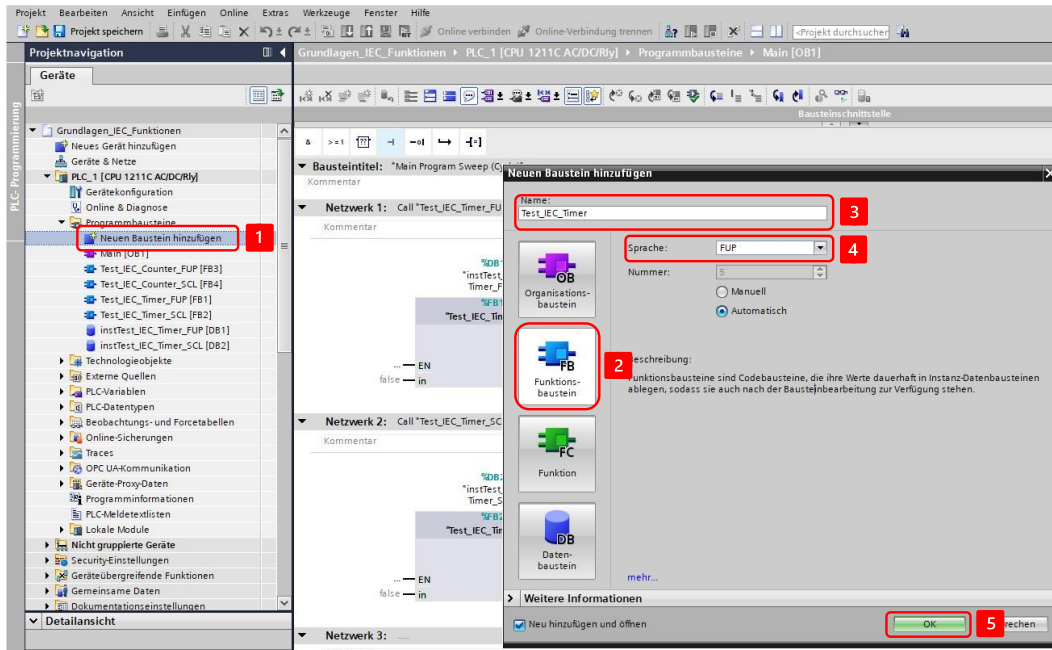
```

- Rufen Sie den erstellten Baustein im "MAIN" auf.
- Über die Instanz können die Eingangsvariablen gesteuert und die Ausgangsvariablen beobachtet werden. Alternativ, falls vorhanden, können diese über die Bausteinschnittstelle beim Aufruf auch mit Tastern und Anzeigeelementen verschaltet werden. Das Ablaufen der Zeit kann in den einzelnen Instanzen der Zeitfunktionen ebenfalls beobachtet werden.

-  Alternativ können auch die vorbereiteten Bausteine "Test_IEC_Timer_FUP[FB1]" oder "Test_IEC_Timer_SCL[FB2]" aus dem Vorlageprojekt "Grundlagen_Programmanweisungen.zap17" verwendet werden.
-  Zusätzliche Hilfestellung zur Interpretation des Programmstatus kann das Kapitel "Inbetriebnahme (Software)" liefern.

Vorgehensweise:

1. Erstellen Sie einen neuen Funktionsbaustein, wählen die gewünschte Programmiersprache und vergeben einen aussagekräftigen Namen:



2. Deklarieren Sie folgende Variablen in der Bausteinschnittstelle:

	Name	Datentyp	Kommentar
1	Input		
2	input_signal	Bool	Test Input
3	Output		
4	output_TP	Bool	Test Output TP
5	output_TON	Bool	Test Output TON
6	output_TOF	Bool	Test Output TOF
7	InOut		
8	<Hinzufügen>		
9	Static		
10	instTp	TP_TIME	Instance TP
11	instTon	TON_TIME	Instance TON
12	instTof	TOF_TIME	Instance TOF
13	Temp		

3. Setzen Sie folgendes Programm um, die Zeitfunktionen finden Sie in der TaskCard unter "Anweisungen" → "Einfache Anweisungen" → "Zeiten":

The screenshot shows the 'Anweisungen' (Instructions) task card on the right side of the STEP 7 interface. The 'Einfache Anweisungen' (Simple Instructions) category is expanded, and the 'Zeiten' (Timers) sub-category is selected. The 'Zeiten' list includes: TP (Impuls erzeugen), TON (Einschaltverzögerung erzeugen), TOF (Ausschaltverzögerung erzeugen), TONR (Zeit akkumulieren), and TOF (Zeit als Impuls starten). The left side shows three networks: 'Netzwerk 1: Test TP' with a TP timer block, 'Netzwerk 2: Test TON' with a TON timer block, and 'Netzwerk 3: Test TOF' with a TOF timer block. Red arrows point from the 'Zeiten' list to these timer blocks.

4. Rufen Sie den Funktionsbaustein im "MAIN" auf, und erstellen Sie eine Instanz:

The screenshot shows the 'MAIN' network in the LAD editor. A function block call for 'Test_IEC_Timer_FUP' is shown. A red box highlights the function block call, and a red arrow points to the 'Aufrufoptionen' (Call Options) dialog box. The dialog box shows the 'Name' field set to 'instTest_IEC_Timer_FUP' and the 'Manuell' (Manual) option selected. The 'OK' button is highlighted with a red box and the number 3.

5. Steuern Sie über die Instanz die Eingangsvariable "input_signal" und beobachten Sie die Ausgangsvariablen "output_TP", "output_TON", "output_TOF", sowie die einzelnen Instanzen der Zeitfunktionen im entsprechenden Instanzdatenbaustein:

The screenshot displays the Siemens STEP 7 LAD editor interface. On the left, three networks are visible: 'Netzwerk 1: Test TP', 'Netzwerk 2: Test TON', and 'Netzwerk 3: Test TOF'. Each network contains a timer function block (TP, TON, TOF) with associated inputs and outputs. The right pane shows the 'instTest_IEC_Timer_FUP' data block, which lists various variables and their current values. A 'Steuern' (Control) dialog box is open, showing the 'input_signal' variable being set to 'true'. A context menu is also visible over the data block, showing options like 'Operand steuern...' and 'Zur nächsten Verwendungsstelle gehen'.

This screenshot shows the same STEP 7 LAD editor interface as the previous one, but with different variable values. The 'input_signal' variable is now 'TRUE'. The 'output_TP' variable is 'TRUE', 'output_TON' is 'FALSE', and 'output_TOF' is 'TRUE'. The time settings for the TP, TON, and TOF instances are also highlighted in red, showing values like 'T#25_891MS', 'T#10S', and 'T#0MS'.

7.4 Zähler

Zählfunktionen können auf den Impuls eines binären Signals einen Zahlenwert hoch oder runter zählen. Es sind Anweisungen, die eine Instanz benötigen.

Nach IEC 61131 stehen folgende Zählfunktionen zur Verfügung:

- Vorwärtszähler (CTU)
- Rückwärtszähler (CTD)
- Vor- und Rückwärtszähler (CTUD)

Der maximal darstellbare Zählbereich ist abhängig vom gewählten Datentyp.

Im TIA-Portal kann der Datentyp mittels Dropdownmenü direkt an der Anweisung eingestellt werden.

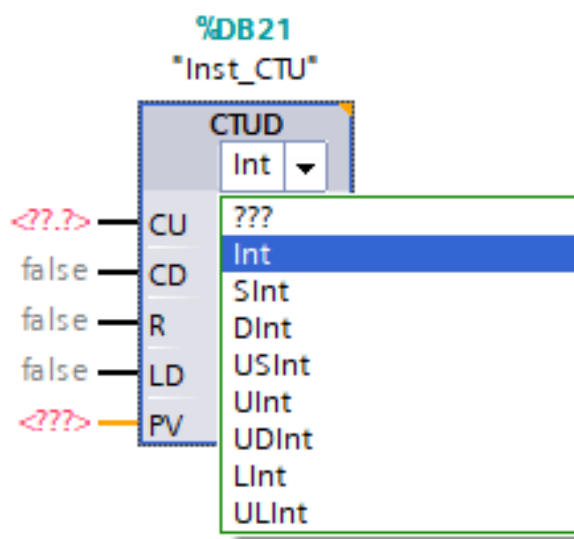


Bild 24 Datentyp der Zählfunktion (Siemens)

Bei Codesys stehen für 64-Bit Zähloperationen eigene Funktionsbausteine zur Verfügung:

- LCTU
- LCTD
- LCTUD

Hier einige wichtige Anwendungen und der Nutzen von Zählfunktionen in der Automatisierungstechnik:

Produktionsüberwachung:

Zählfunktionen werden verwendet, um die Anzahl der produzierten Einheiten zu überwachen. Dies ist besonders in der Fertigungsindustrie wichtig, um sicherzustellen, dass die Produktionsziele erreicht werden.

Inventur- und Materialflusssteuerung:

Sie helfen bei der Überwachung der Materialbewegungen innerhalb einer Produktionsstätte. Zum Beispiel kann gezählt werden, wie oft ein Teil durch eine bestimmte Station läuft, was zur Optimierung des Materialflusses und zur Bestandskontrolle beiträgt.

Sicherheitsanwendungen:

Zählfunktionen können auch in sicherheitskritischen Anwendungen eingesetzt werden, um die Anzahl der Vorgänge zu überwachen, bei denen ein Gerät oder eine Maschine benutzt wurde. Dies kann wichtig sein, um Wartungsintervalle einzuhalten.

Steuerung sequenzieller Vorgänge:

Zählfunktionen bieten eine präzise Steuerung der Häufigkeit, mit der bestimmte Aktionen in einem Programm ausgeführt werden. Sie sind besonders nützlich in Situationen, in denen ein Vorgang mehrfach wiederholt werden muss, bevor der Programmablauf zum nächsten Schritt übergeht.

Batch-Verarbeitung:

In der chemischen Industrie oder bei der Lebensmittelverarbeitung können Zählfunktionen dazu verwendet werden, die Anzahl der Chargen zu überwachen, die durch einen Prozess gegangen sind, was zur Qualitätskontrolle und Dokumentation beiträgt.

7.4.1 Vorwärts zählen CTU

Mit dem IEC-CTU-Zähler können Sie einen Vorwärtszähler programmieren. Über eine positive Flanke am Eingang CU (Count Up) erhöhen Sie den Wert am Ausgang CV (Current Value). Ist der am Eingang PV (Preset Value) vorgegebene Wert erreicht, oder überschritten so wird der Ausgang Q (Output) geschaltet. Über den Eingang R (Reset) kann der Zählwert auf 0 gesetzt werden.

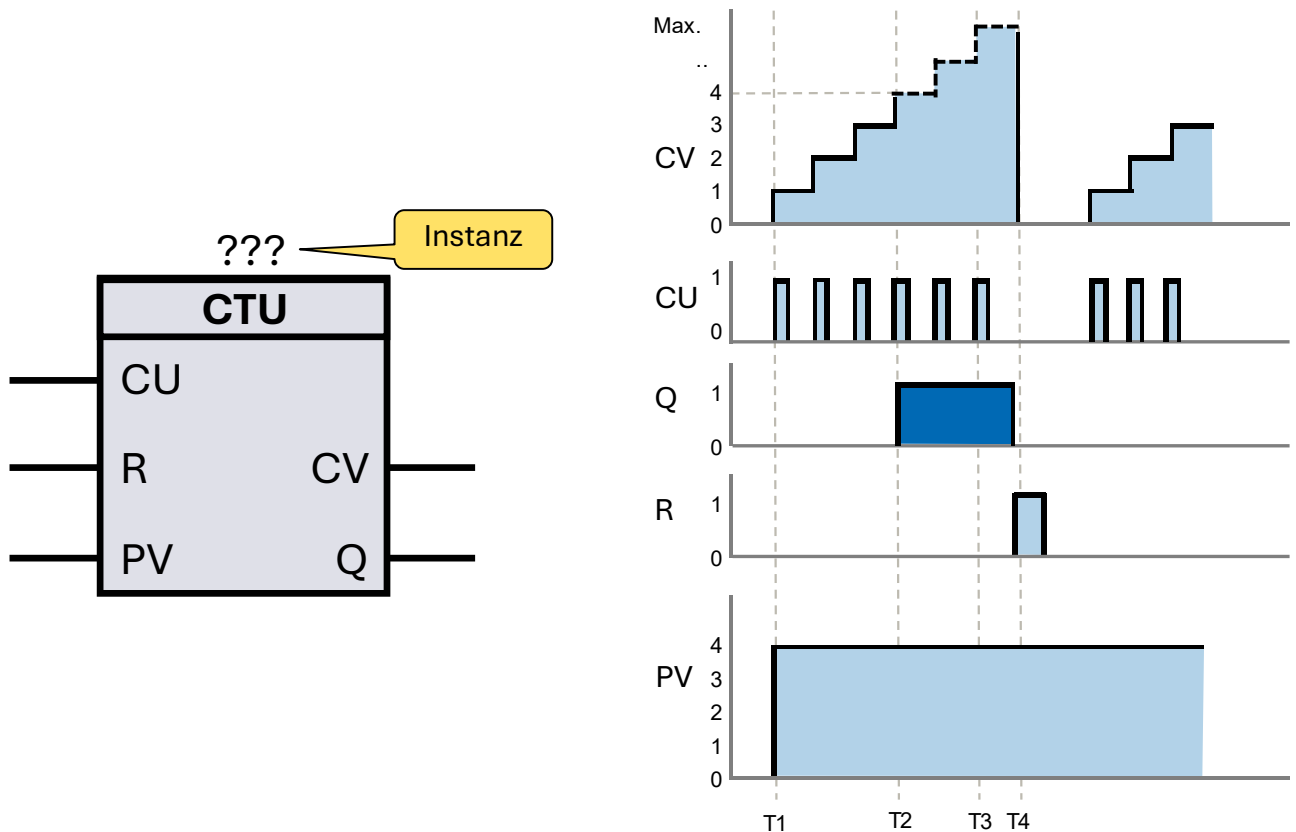


Bild 25 FUP-Anweisung CTU und Impulsdiagramm

T1

Wechselt das Signal am Eingang CU von "0" auf "1" (positive Flanke), wird der Zählwert am Ausgang CV um eins erhöht.

T2

Der Ausgang Q zeigt den Status des Zählers an. Der Signalzustand von Q wird durch den parametrisierten Sollwert PV bestimmt. Wenn der aktuelle Zählwert den parametrisierten Sollwert PV erreicht oder überschreitet, wird der Ausgang Q auf "1" gesetzt. Andernfalls bleibt der Signalzustand von Q "0". Am Parameter PV kann eine Konstante oder eine Variable angegeben werden.

T3

Der Zählvorgang wiederholt sich bei jeder positiven Flanke, bis der Zählwert den maximalen Wert des am Ausgang CV definierten Datentyps erreicht. Nach Erreichen dieses Grenzwertes bleibt der Zählwert konstant, unabhängig vom Signalzustand am Eingang CU.

T4

Bei einer positiven Flanke am Eingang R wird der Zählwert am Ausgang CV auf "0" zurückgesetzt.

Jedem Aufruf der Anweisung "Vorwärts zählen" muss eine Instanz vom Datentyp "CTU" zugeordnet werden, in der die Instanzdaten gespeichert werden.

Textuelle Umsetzung Vorwärtszähler (CTU)

Die Instanziierung des Vorwärtszählers CTU erfolgt ähnlich wie bei einem Funktionsbaustein. Für jeden Aufruf des Funktionsbausteins "CTU" wird ein eigener Speicherbereich benötigt. Die Instanzdaten können entweder als Einzelinstanz oder als Multiinstanz (in der Bausteinschnittstelle) angelegt werden.

Beispiel:

//Deklaratioinsteil, Bausteinschnittstelle

VAR

instCtu : CTU; **//Deklaration der Instanzdaten (Multiinstanz)**

END_VAR

//Anweisungsteil, Aufruf der Instanz

```
instCtu(CU := varBoolCu,
        R := varBoolR,
        PV := varIntPv,
        CV => varIntCv,
        Q => varBoolQ);
```

Bild 26 ST/SCL-Anweisung CTU

7.4.2 Rückwärts zählen CTD

Mit dem IEC-CTD-Zähler können Sie einen Rückwärtszähler programmieren. Bei einer positiven Flanke am Eingang CD (Count Down), also wenn der Signalzustand von "0" auf "1" wechselt, wird der aktuelle Zählwert am Ausgang CV (Current Value) um eins reduziert. Der Ausgang Q (Output) zeigt den Status des Zählers an. Sobald der Zählwert kleiner oder gleich "0" ist, schaltet der Ausgang Q auf "1". Wenn der Signalzustand am Eingang LD (Load) von "0" auf "1" wechselt, wird der Zählwert am Ausgang CV auf den Wert des Parameters PV (Preset Value) geladen.

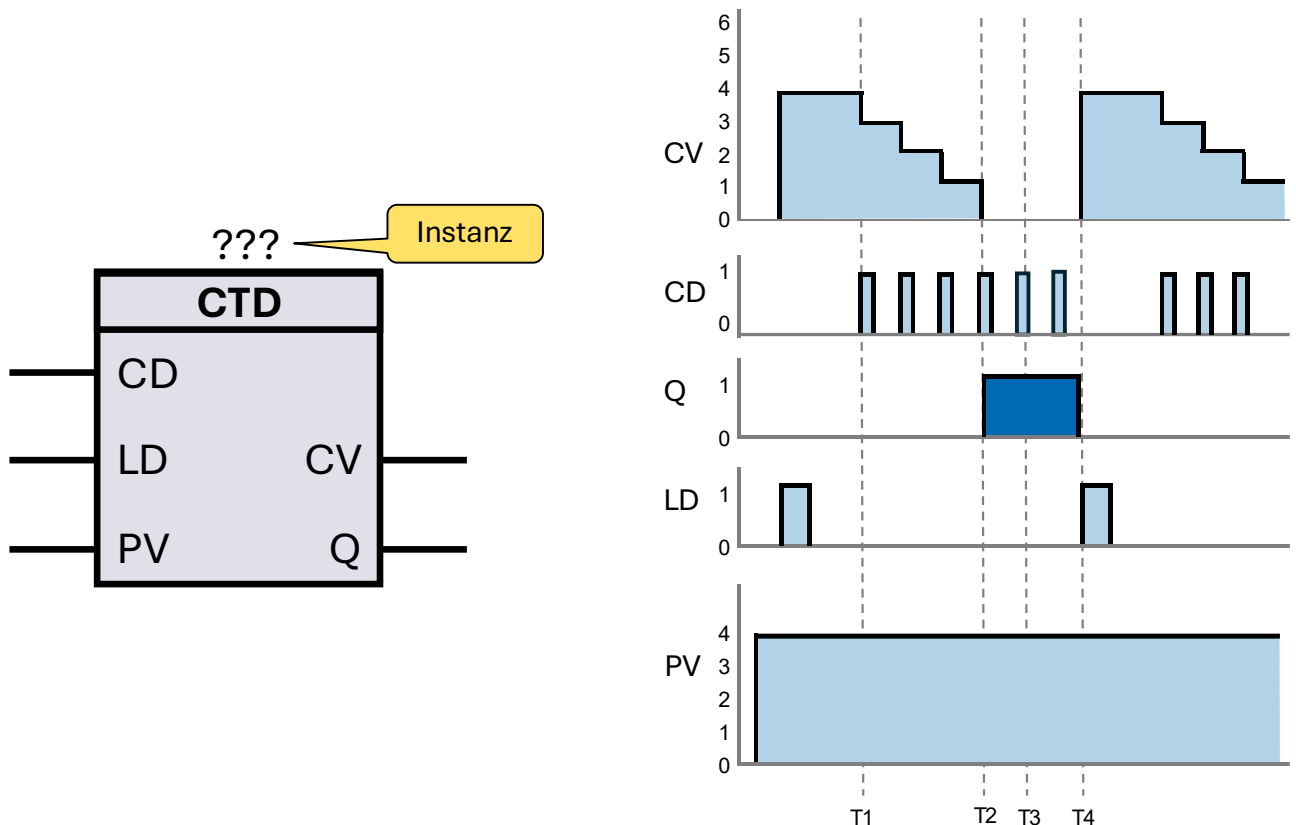


Bild 27 FUP-Anweisung CTD und Impulsdiagramm

T1

Bei einer positiven Flanke am Eingang CD wird der aktuelle Zählwert am Zählwert CV um eins reduziert.

T2

Dieser Vorgang wiederholt sich bei jeder positiven Flanke, bis der Zählwert CV den unteren Grenzwert des angegebenen Datentyps erreicht. Ist der aktuelle Zählwert CV kleiner oder gleich 0 wird der Ausgang Q auf "1" gesetzt.

T3

Ist der Untere Grenzwert des angegebenen Datentyps erreicht bleibt der Ausgang Q auf "1" der Eingang CD hat keinen weiteren Einfluss.

T4

Bei einer positiven Flanke am Eingang LD wird der am Eingang PV angegebene Wert in den CV geladen.

Jedem Aufruf der Anweisung "Rückwärts zählen" muss eine Instanz vom Datentyp "CTD" zugeordnet werden, in der die Instanzdaten gespeichert werden.

Textuelle Umsetzung Rückwärtszähler (CTD)

Die Instanziierung des Rückwärtszählers CTD erfolgt ähnlich wie bei einem Funktionsbaustein. Für jeden Aufruf des Funktionsbausteins "CTD" wird ein eigener Speicherbereich benötigt. Die Instanzdaten können entweder als Einzelinstanz oder als Multiinstanz (in der Bausteinschnittstelle) angelegt werden.

Beispiel:

//Deklarationsteil, Bausteinschnittstelle

VAR

instCtd : CTD; **//Deklaration der Instanzdaten (Multiinstanz)**

END_VAR

//Anweisungsteil, Aufruf der Instanz

```
instCtd(CD := varBoolCd,
        LD := varBoolLd,
        PV := varIntPv,
        CV => varIntCv,
        Q => varBoolQ);
```

Bild 28 ST/SCL-Anweisung CTD

7.4.3 Vorwärts und rückwärts zählen CTUD

Mit der Anweisung "Vorwärts und rückwärts zählen" wird der Zählwert am Ausgang CV (Current Value) sowohl erhöht als auch verringert. Eine positive Flanke am Eingang CU (Count Up) erhöht den Zählwert um eins, während eine positive Signalflanke am Eingang CD (Count Down) den Zählwert um eins verringert. Eine positive Flanke am Eingang R (Reset) setzt den Ausgang CV auf 0 zurück. Eine positive Flanke am Eingang LD (Load) lädt den am Eingang PV (Preset Value) angegebenen Wert in den Zählwert CV. Der Zähler hat einen Ausgang QU (Count Up Output), welcher gesetzt wird, wenn der aktuelle Zählwert CV größer oder gleich dem am Eingang PV angegebenen Wert ist. Der Zähler hat einen Ausgang QD (Count Down Output), welcher gesetzt wird, wenn der aktuelle Zählwert CV kleiner oder gleich 0 ist.

i Tritt in einem Programmzyklus an beiden Eingängen eine positive Signalflanke auf, bleibt der Zählwert unverändert.

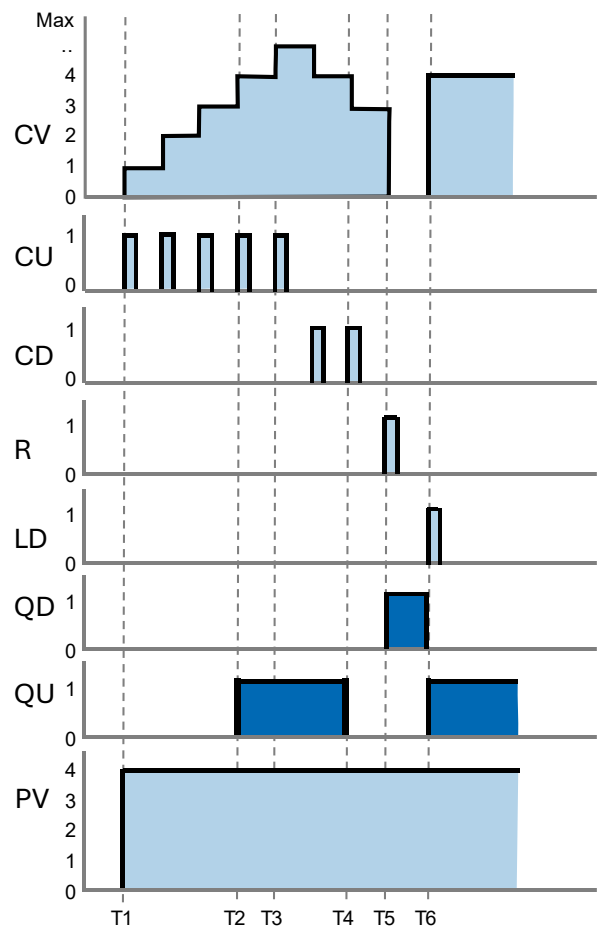
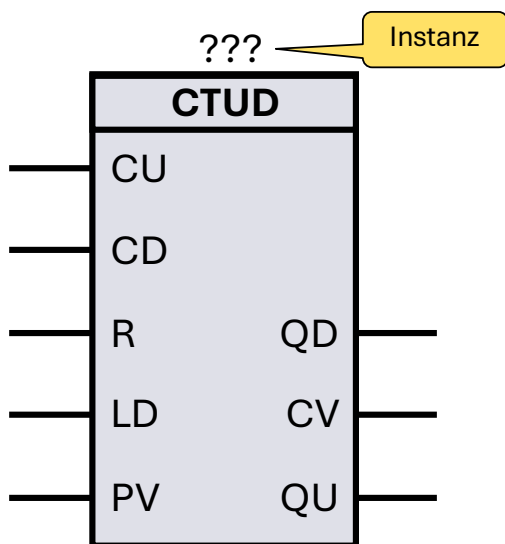


Bild 29 FUP-Anweisung CTUD und Impulsdiagramm

T1

Eine positive Flanke am Eingang CU erhöht den Zählwert CV um eins.

T2

Der Ausgang QU wird auf "1" gesetzt, wenn der aktuelle Zählwert CV größer oder gleich dem am Eingang PV angegebenen Wert ist.

T3

Der Zählwert CV kann bis zum oberen bzw. unteren Grenzwert des angegebenen Datentyps gezählt werden.

T4

Jede positive Flanke am Eingang CD verringert den aktuellen Zählwert CV um eins. Ist der aktuelle Zählwert CV kleiner oder gleich dem am Eingang PV angegebenen Wert, so wird der Ausgang QD zurückgesetzt.

T5

Eine positive Flanke am Eingang R setzt den aktuellen Zählwert auf "0", wodurch der Ausgang QD gesetzt wird.

T6

Eine positive Flanke am Eingang LD setzt den aktuellen Zählwert auf den am Eingang PV vorgegebenen Wert, wodurch der Ausgang QD gesetzt wird.

Jedem Aufruf der Anweisung "Vorwärts und rückwärts zählen" muss eine Instanz vom Datentyp "CTUD" zugeordnet werden, in der die Instanzdaten gespeichert werden.

Textuelle Umsetzung Vorwärts- und Rückwärtszähler (CTUD)

Die Instanziierung des Vorwärts- und Rückwärtszähler CTUD erfolgt ähnlich wie bei einem Funktionsbaustein. Für jeden Aufruf des Funktionsbausteins "CTUD" wird ein eigener Speicherbereich benötigt. Die Instanzdaten können entweder als Einzelinstanz oder als Multiinstanz (in der Bausteinschnittstelle) angelegt werden.

Beispiel:

//Deklarationsteil, Bausteinschnittstelle

VAR

instCtud : CTUD; //Deklaration der Instanzdaten (Multiinstanz)

END_VAR

//Anweisungsteil, Aufruf der Instanz

```
instCtud(CU := varBoolCu,
        CD := varBoolCd,
        R := varBoolR,
        LD := varBoolLd,
        PV := varIntPv,
        QD => varBoolQd,
        CV => varIntCv,
        QU => varBoolQu);
```

Bild 30 ST/SCL-Anweisung CTUD



7.4.4 Übung - IEC Zähler

Ziel

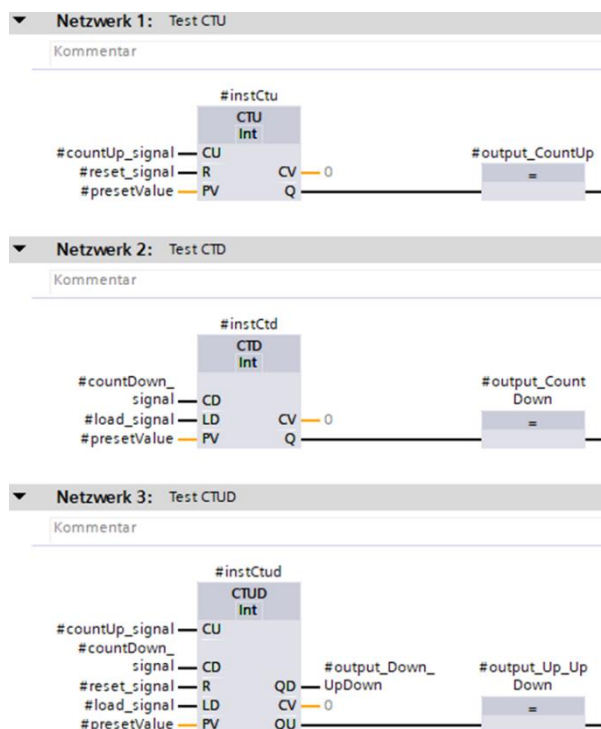
In dieser Übung sollen die 3 IEC Zähler (CTU, CTD, CTUD) in einem Test-Funktionsbaustein programmiert und praktisch kennengelernt werden. Über die Eingangsparameter "countUp_signal" und "countDown_signal" können die Zählstände positiv oder negativ beeinflusst werden, sowie über die Eingangsparameter "reset_signal", "load_signal", "presetValue" auf definierte Werte gesetzt werden. Der Zustand wird über die Ausgangsparameter "output_CountUp", "output_CountDown", "output_Up_UpDown", "output_Down_UpDown" angezeigt.

Aufgabe

- Erstellen Sie einen Funktionsbaustein mit folgender Bausteinschnittstelle:

	Name	Datentyp	Kommentar
1	Input		
2	countUp_signal	Bool	Test Input Count Up
3	countDown_signal	Bool	Test Input Count Down
4	reset_signal	Bool	Test Input Reset Counter
5	load_signal	Bool	Test Input Load
6	presetValue	Int	Test Input PV
7	Output		
8	output_CountUp	Bool	Test Output Q CTU
9	output_CountDown	Bool	Test Output Q CTD
10	output_Up_UpDown	Bool	Test Output QU CTUD
11	output_Down_UpDown	Bool	Test Output QD CTUD
12	InOut		
13	<Hinzufügen>		
14	Static		
15	instCtu	CTU_INT	Instance CTU
16	instCtd	CTD_INT	Instance CTD
17	instCtud	CTUD_INT	Instance CTUD
18	Temp		

- Abhängig der gewählten Programmiersprache soll folgendes Programm umgesetzt werden:





```

2  //Test CTU
3  #instCtu(CU := #countUp_signal,
4           R := #reset_signal,
5           PV := #presetValue,
6           //CV => ,
7           Q => #output_CountUp);
8
9  //Test CTD
10 #instCtd(CD := #countDown_signal,
11          LD := #load_signal,
12          PV := #presetValue,
13          //CV=> ,
14          Q => #output_CountDown);
15
16 //Test CTUD
17 #instCtud(CU := #countUp_signal,
18           CD := #countDown_signal,
19           R := #reset_signal,
20           LD := #load_signal,
21           PV := #presetValue,
22           //CV=> ,
23           QU => #output_Up_UpDown,
24           QD => #output_Down_UpDown);

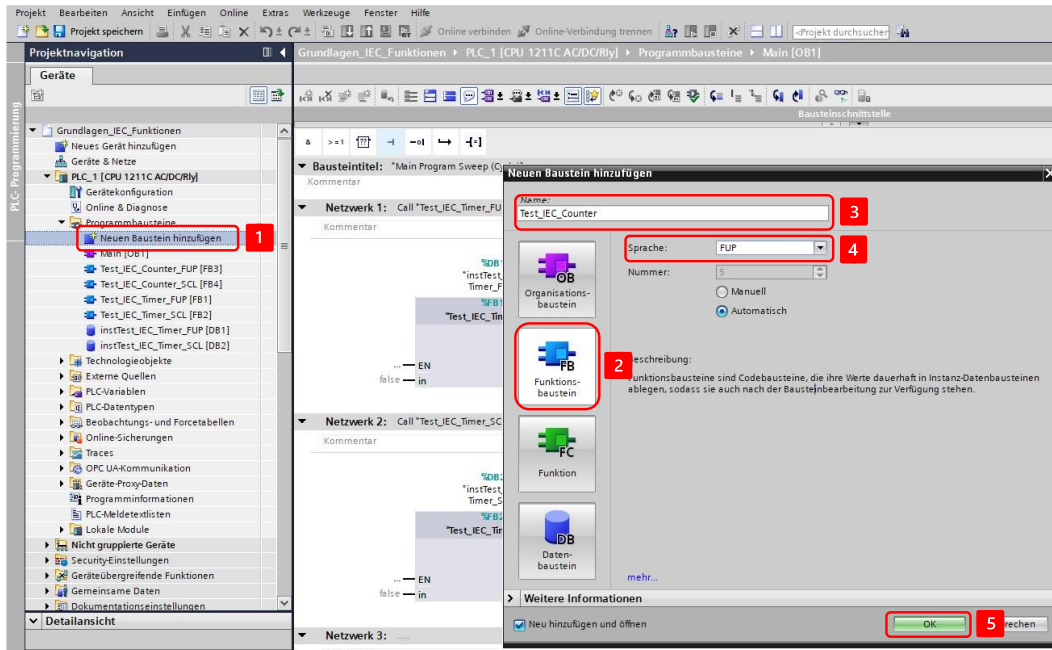
```

- Rufen Sie den erstellten Baustein im "MAIN" auf.
- Über die Instanz können die Eingangsvariablen gesteuert und die Ausgangsvariablen beobachtet werden. Alternativ, falls vorhanden, können diese über die Bausteinschnittstelle beim Aufruf auch mit Tastern und Anzeigeelementen verschaltet werden. Der Zählstand kann in den einzelnen Instanzen der Zähler ebenfalls beobachtet werden.

-  Alternativ können auch die vorbereiteten Bausteine "Test_IEC_Counter_FUP[FB3]" oder "Test_IEC_Counter_SCL[FB4]" aus dem Vorlageprojekt "Grundlagen_Programmanweisungen.zap17" verwendet werden.
-  Zusätzliche Hilfestellung zur Interpretation des Programmstatus kann das Kapitel "Inbetriebnahme (Software)" liefern.

Vorgehensweise:

1. Erstellen Sie einen neuen Funktionsbaustein, wählen die gewünschte Programmiersprache und vergeben einen aussagekräftigen Namen:



2. Deklarieren Sie folgende Variablen in der Bausteinschnittstelle:

	Name	Datentyp	Kommentar
1	Input		
2	countUp_signal	Bool	Test Input Count Up
3	countDown_signal	Bool	Test Input Count Down
4	reset_signal	Bool	Test Input Reset Counter
5	load_signal	Bool	Test Input Load
6	presetValue	Int	Test Input PV
7	Output		
8	output_CountUp	Bool	Test Output Q CTU
9	output_CountDown	Bool	Test Output Q CTD
10	output_Up_UpDown	Bool	Test Output QU CTUD
11	output_Down_UpDown	Bool	Test Output QD CTUD
12	InOut		
13	<Hinzufügen>		
14	Static		
15	instCtu	CTU_INT	Instance CTU
16	instCtd	CTD_INT	Instance CTD
17	instCtud	CTUD_INT	Instance CTUD
18	Temp		

3. Setzen Sie folgendes Programm um, die Zährefunktionen finden Sie in der TaskCard unter "Anweisungen" → "Einfache Anweisungen" → "Zähler":

The screenshot shows the Siemens STEP 7 LAD editor with three networks:

- Netzwerk 1: Test CTU**: Shows a CTU block with inputs #countUp_signal (CU), #reset_signal (R), and #presetValue (PV). The output is #output_CountUp.
- Netzwerk 2: Test CTD**: Shows a CTD block with inputs #countDown_signal (CD), #load_signal (LD), and #presetValue (PV). The output is #output_CountDown.
- Netzwerk 3: Test CTUD**: Shows a CTUD block with inputs #countUp_signal (CU), #countDown_signal (CD), #reset_signal (R), #load_signal (LD), and #presetValue (PV). The outputs are #output_Down (QD) and #output_Up (QU).

The 'Optionen' sidebar on the right shows the 'Anweisungen' (Instructions) category expanded, with 'Einfache Anweisungen' (Simple Instructions) selected. The 'Zähler' (Counter) sub-category is highlighted, showing CTU, CTD, and CTUD blocks.

4. Rufen Sie den Funktionsbaustein im "MAIN" auf, und erstellen Sie eine Instanz:

The screenshot shows the Siemens STEP 7 interface with the 'Projekt' (Project) window on the left. The 'Grundlagen_IEC_Funktionen' (Basic IEC Functions) library is expanded, and the 'Test_IEC_Counter_FUP' function block is selected. The main editor shows the function block being called in the 'Netzwerk 3' (Network 3). The 'Aufrufoptionen' (Call Options) dialog is open, showing the 'Name' field set to 'Test_IEC_Counter_FUP' and the 'Nummer' field set to '4'. The 'Einzel-Instanz' (Single Instance) radio button is selected.

5. Steuern Sie über die Instanz die Eingangsvariablen und beobachten Sie die Ausgangsvariablen, sowie die einzelnen Instanzen der Zählfunktionen im entsprechenden Instanzdatenbaustein:

The screenshot displays the Siemens STEP 7 LAD editor with three networks (Test CTU, Test CTD, Test CTUD) and the variable declaration table for the IEC Counter function block. Red boxes highlight specific variables and the 'Steuern' dialog box.

Netzwerk 1: Test CTU

Netzwerk 2: Test CTD

Netzwerk 3: Test CTUD

instTest_IEC_Counter_FUP

Name	Datentyp	Beobachtungswert	Kommentar
countUp_signal	Bool	TRUE	Test Input Count Up
countDown_signal	Bool	FALSE	Test Input Count Down
reset_signal	Bool	FALSE	Test Input Reset Counter
load_signal	Bool	FALSE	Test Input Load
presetValue	Int	5	Test Input PV
output_CountUp	Bool	TRUE	Test Output Q CTU
output_CountDown	Bool	TRUE	Test Output Q CTD
output_UpUpDownDown	Bool	FALSE	Test Output QU CTUD
output_DownDownUpUp	Bool	FALSE	Test Output QD CTUD
instCtu	CTU_INT		Instance CTU
CU	Bool	TRUE	
CD	Bool	FALSE	
R	Bool	FALSE	
LD	Bool	FALSE	
QU	Bool	TRUE	
QD	Bool	FALSE	
PV	Int	5	
instCtd	CTD_INT		Instance CTD
CU	Bool	FALSE	
CD	Bool	TRUE	
R	Bool	FALSE	
LD	Bool	FALSE	
QU	Bool	FALSE	
QD	Bool	TRUE	
PV	Int	5	
instCtud	CTUD_INT		Instance CTUD
CU	Bool	TRUE	
CD	Bool	FALSE	
R	Bool	FALSE	
LD	Bool	FALSE	
QU	Bool	FALSE	
QD	Bool	FALSE	
PV	Int	5	
CV	Int	3	

Steuern

Operand: "instTest_IEC_Counter_FUP".countUp

Steuernwert: true

Datentyp: Bool

OK

7.5 IF-Anweisung [ST / SCL]

In der Programmiersprache ST / SCL wird die IF-Anweisung genutzt, um bedingte Abläufe zu steuern. Mit dieser Anweisung kann der Programmfluss abhängig von bestimmten Bedingungen gesteuert werden. Im Folgenden wird die Verwendung der IF, ELSE und ELSIF-Anweisungen in SCL beschrieben.

7.5.1 IF...THEN - Anweisung

Die IF...THEN-Anweisung (Bedingt ausführen) dient dazu, einen Anweisungsblock abhängig von einer Bedingung auszuführen.

Die Grundstruktur dieser-Anweisung sieht folgendermaßen aus:

```
IF (*Bedingung*) THEN
  //Anweisungen

END_IF;
```

Bild 31 Codebeispiel IF...THEN-Anweisung

Struktrogramm

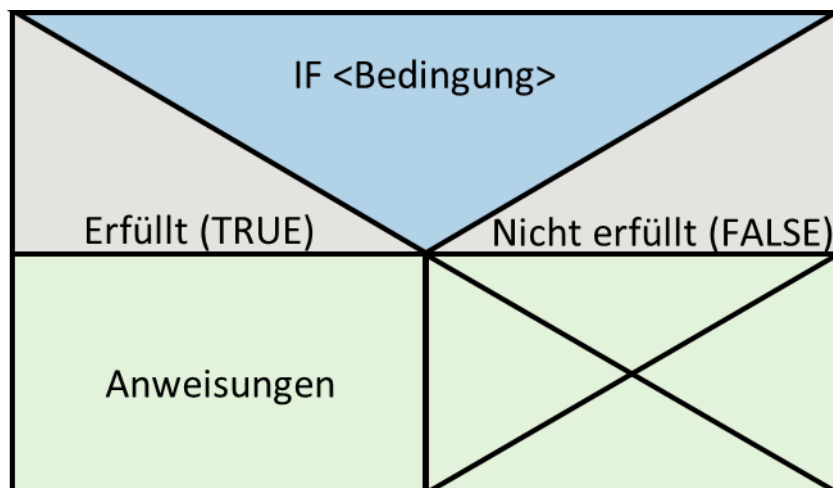


Bild 32 Struktogramm IF...THEN-Anweisung

Beispiel

```
IF Temperatur > 100 THEN
  Alarm := TRUE;
END_IF;
```

Bild 33 Beispiel IF...THEN-Anweisung

In diesem Beispiel wird die Variable "Alarm" auf "TRUE" gesetzt, wenn die Bedingung "Temperatur > 100" erfüllt ist.

7.5.2 IF...THEN...ELSE - Anweisung

Die IF...THEN...ELSE-Anweisung (Bedingt verzweigen) dient dazu, einen Anweisungsblock abhängig von einer Bedingung einfach zu verzweigen. Die ELSE-Anweisung wird verwendet, um alternative Anweisungen auszuführen, wenn die Bedingung der IF-Anweisung nicht erfüllt ist.

Die Grundstruktur dieser Anweisung sieht folgendermaßen aus:

```
IF (*Bedingung*) THEN
    //Anweisungen

ELSE
    //Alternative Anweisungen

END_IF;
```

Bild 34 Codebeispiel IF...THEN...ELSE-Anweisung

Struktogramm

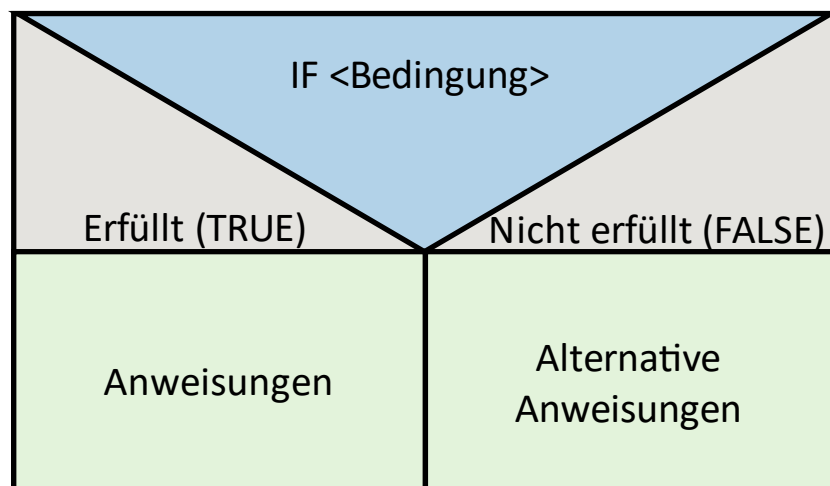


Bild 35 Struktogramm IF...THEN...ELSE-Anweisung

Beispiel

```
IF Temperatur > 100 THEN
    Alarm := TRUE;
ELSE
    Alarm := FALSE;
END_IF;
```

Bild 36 Beispiel IF...THEN...ELSE-Anweisung

In diesem Beispiel wird die Variable "Alarm" auf "TRUE" gesetzt, wenn die Bedingung "Temperatur > 100" erfüllt ist. Andernfalls wird "Alarm" auf "FALSE" gesetzt.

7.5.3 IF...THEN...ELSIF - Anweisung

Die IF...THEN...ELSIF Anweisung (Mehrfach bedingt verzweigen) dient dazu, einen Anweisungsblock abhängig von mehreren Bedingungen zu verzweigen.

Die ELSIF-Anweisung ermöglicht es, mehrere Bedingungen in einer IF-Anweisung zu prüfen. Wenn die erste Bedingung nicht erfüllt ist, wird die nächste Bedingung geprüft, und so weiter.

Die Grundstruktur dieser Anweisung sieht folgendermaßen aus:

```

IF (*Bedingung 1*) THEN
  //Anweisungen 1

ELSIF (*Bedingung 2*) THEN
  //Anweisungen 2

ELSE
  //Anweisungen 3

END_IF;
    
```

Bild 37 Codebeispiel IF...THEN...ELSIF-Anweisung

Struktrogramm

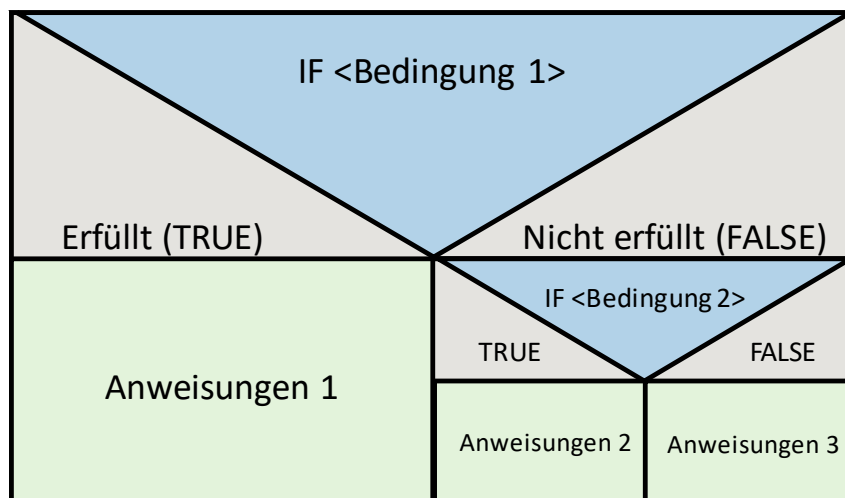


Bild 38 Struktogramm IF...THEN...ELSE-Anweisung

Beispiel:

```

IF Temperatur > 100 THEN
  Alarm := TRUE;
ELSIF Temperatur > 80 THEN
  Warnung := TRUE;
ELSE
  Alarm := FALSE;
  Warnung := FALSE;
END_IF;
    
```

Bild 39 Beispiel IF...THEN...ELSE-Anweisung

In diesem Beispiel wird:

- Die Variable "Alarm" auf "TRUE" gesetzt, wenn "Temperatur > 100".
- Die Variable "Warnung" auf "TRUE" gesetzt, wenn die Temperatur größer als 80, aber kleiner oder gleich 100 ist.
- Andernfalls werden "Alarm" und "Warnung" auf "FALSE" gesetzt.

7.6 CASE-Struktur [ST / SCL]

Mit der CASE-Anweisung (Mehrfach verzweigen / Fallunterscheidung) bearbeiten Sie, abhängig vom Wert eines numerischen Ausdrucks, eine von mehreren Anweisungsfolgen.

Der Wert des Ausdrucks muss eine Ganzzahl sein. Bei der Ausführung der Anweisung wird der Wert des Ausdrucks mit den Werten mehrerer Konstanten verglichen. Wenn der Wert des Ausdrucks mit dem Wert einer Konstante übereinstimmt, werden die Anweisungen ausgeführt, die direkt nach dieser Konstanten programmiert sind.

Die Konstanten können dabei die folgenden Werte annehmen:

- Eine Ganzzahl (z.B. 3)
- Ein Bereich aus Ganzzahlen (z.B. 5...8)
- Eine Aufzählung aus Ganzzahlen und Bereichen (z.B. 11, 17... 25)
- Eine Bitfolge (0001)

In Abhängigkeit vom Wert des Ausdruckes wird eine der folgenden Alternativen (CASE1 ... CASEN) gewählt und die entsprechende Anweisungsfolge (Anweisung 1 bis Anweisung N) ausgeführt. Trifft keine der Alternativen zu, wird der ELSE-Zweig ausgeführt, sofern er vorhanden ist.

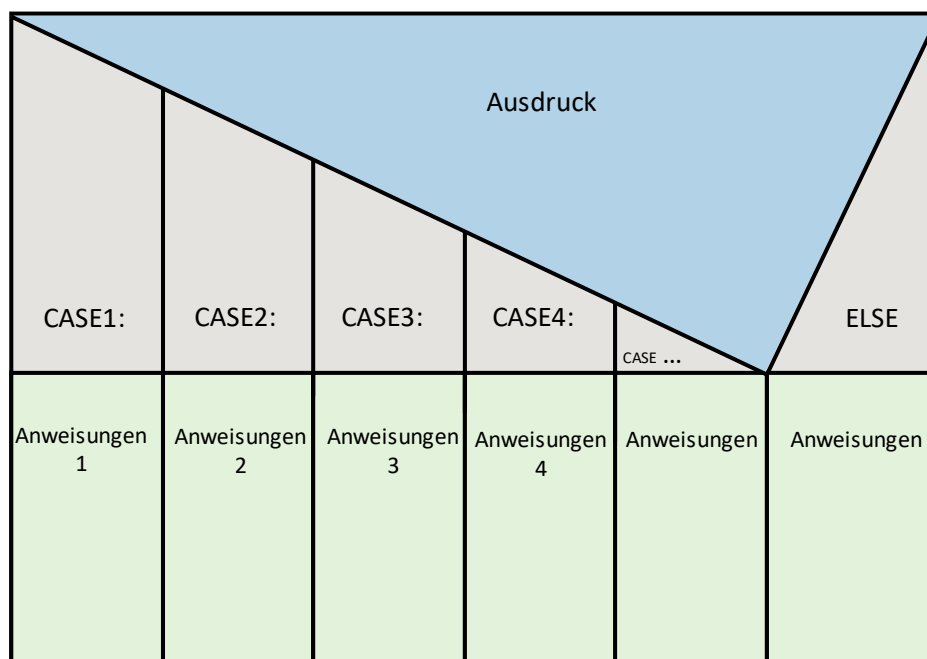


Bild 40 CASE-Anweisung Struktogramm

Die Syntax einer CASE-Anweisung sieht folgendermaßen aus:

```
CASE (*Variable*) OF
1: //CASE 1
   //Anweisungen 1

2: //CASE 2
   //Anweisungen 2

3: //CASE 3
   //Anweisungen 3

4: //CASE 4
   //Anweisungen 4

ELSE
   //Anweisungen

END_IF;
```

Bild 41 Syntax CASE-Anweisung

Die CASE-Struktur unterstützt auch die Angabe von Wertebereichen und das Gruppieren mehrerer Werte.

```
CASE (*Variable*) OF
1..9: //Variable 1 - 9
     //Anweisungen

10, 15, 19: //Variable 10, 15, 19
           //Anweisungen

ELSE
   //Anweisungen

END_IF;
```

Bild 42 CASE-Anweisung mit Mehrfachauswahl

In diesem Beispiel wird die Variable 'Alter' geprüft, und je nach Wertebereich wird die Kategorie zugewiesen.

```
CASE Alter OF
  0..12: //Kind
    Kategorie := 'Kind';

  13..19: //Teenager
    Kategorie := 'Teenager';

  20..64: //Erwachsener
    Kategorie := 'Erwachsener';

  65..100: //Senior
    Kategorie := 'Senior';

ELSE
  Kategorie := 'Ungültig';

END_IF;
```

Bild 43 Beispiel CASE

Die CASE-Struktur ermöglicht eine klare und effiziente Methode zur Auswahl zwischen mehreren Alternativen basierend auf dem Wert einer Variablen. Sie verbessert die Lesbarkeit und Wartbarkeit des Codes, besonders wenn viele Bedingungen geprüft werden müssen, gegenüber der IF-Anweisung. Die Möglichkeit, Wertebereiche und mehrere Werte pro Fall anzugeben, bietet zusätzliche Flexibilität.