

**Vorlesung**

# **AUTOMATISIERUNGSTECHNIK**

**Sommersemester 2025 | Kapitel 5**

Prof. Dr.-Ing. Christian Schlosser  
Hochschule für Angewandte Wissenschaften Hamburg  
Institut für Antriebs- und Regelungstechnik



# KAPITEL 5: IEC61131-3: PROGRAMMIERUNG

Sie lernen

- die Programmierung einer SPS in Sprachen der IEC61131-3,

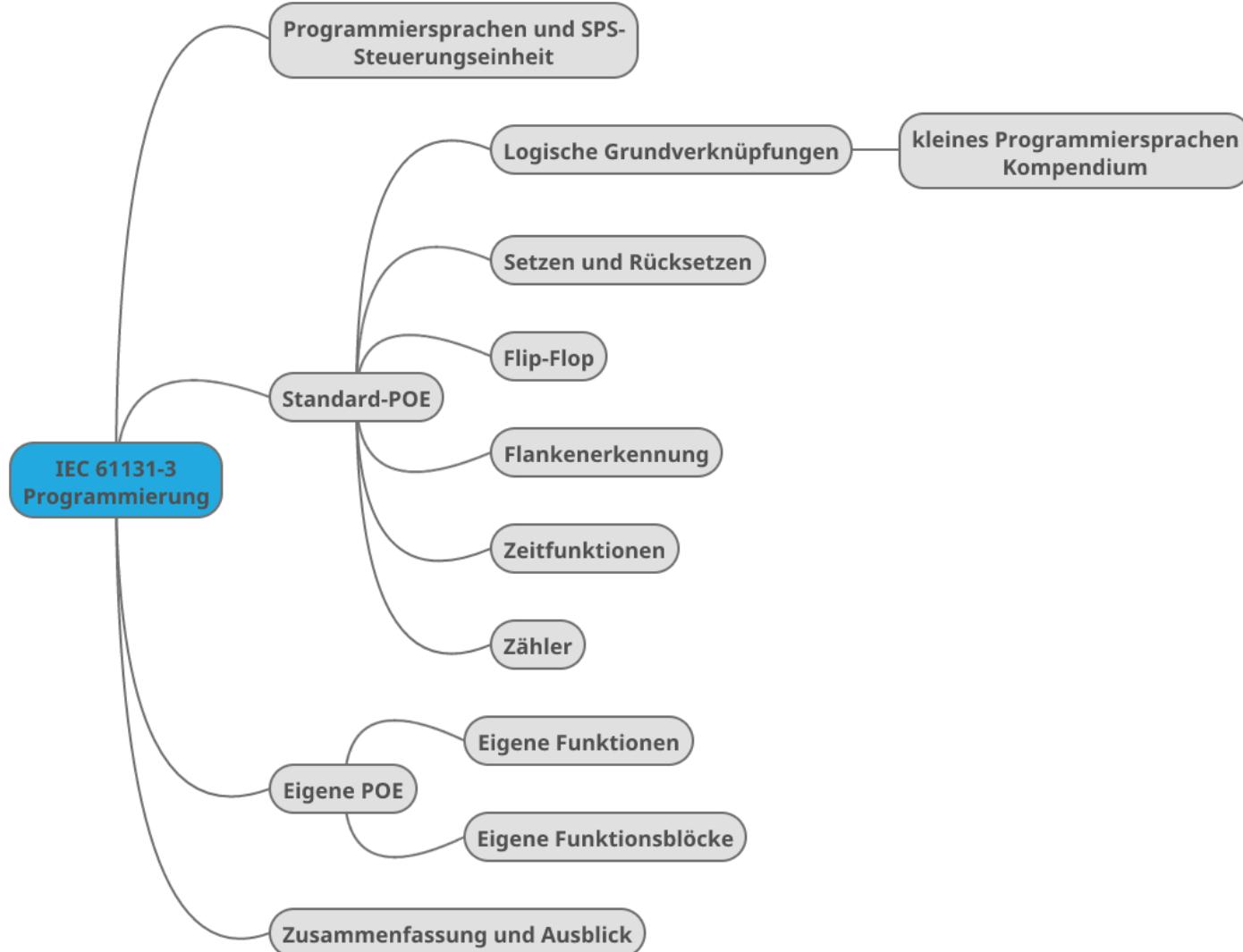
um damit

- einfache Aufgabenstellungen aus der Automatisierungstechnik modellieren und programmieren zu können.

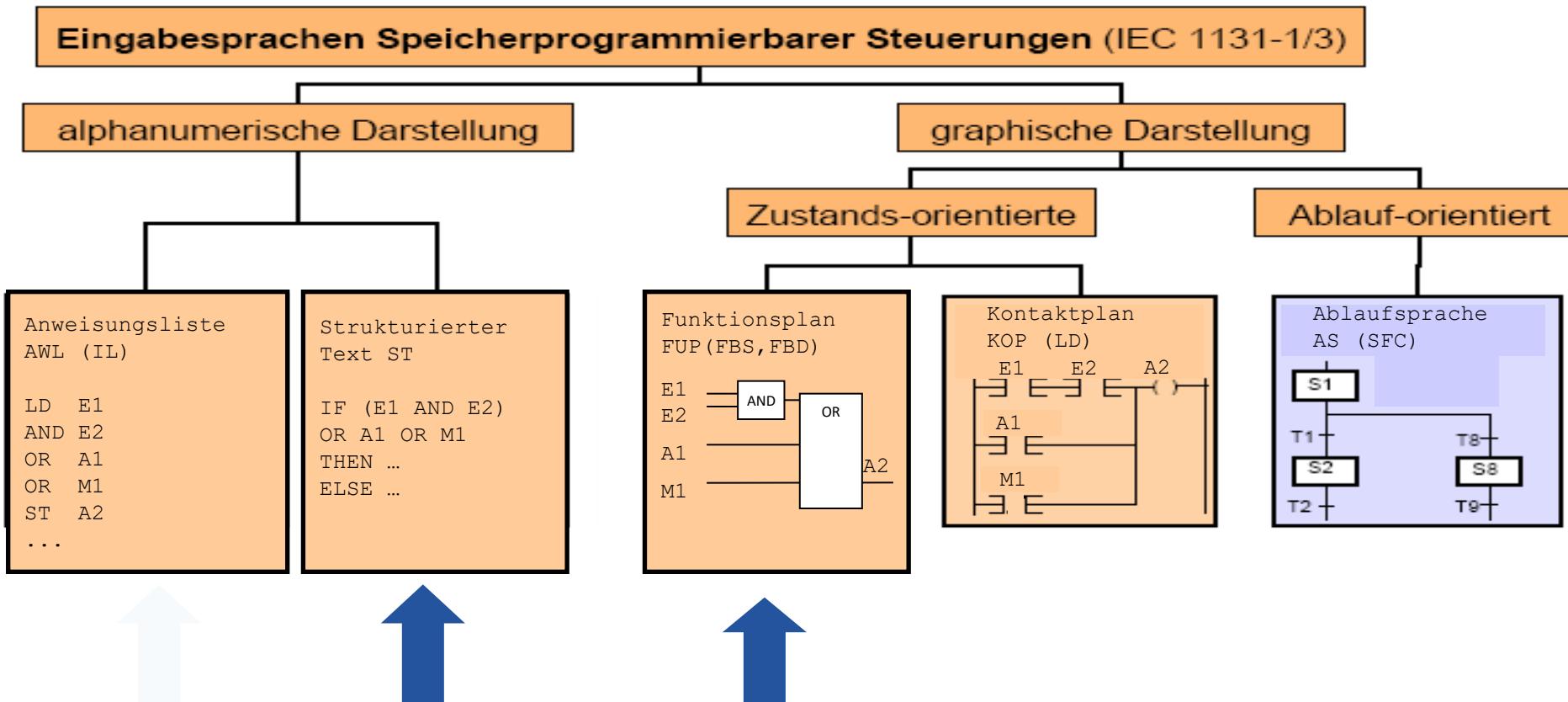
Sie lernen im Speziellen:

- die IEC-Programmiersprachen praktisch kennen und zu unterscheiden.
- den Umgang mit den Standardfunktionen und –Funktionsbausteinen.

# KAPITEL 5: PROGRAMMIERSPRACHEN FUP UND ST



# PROGRAMMIERSPRACHEN DER IEC 61131-3

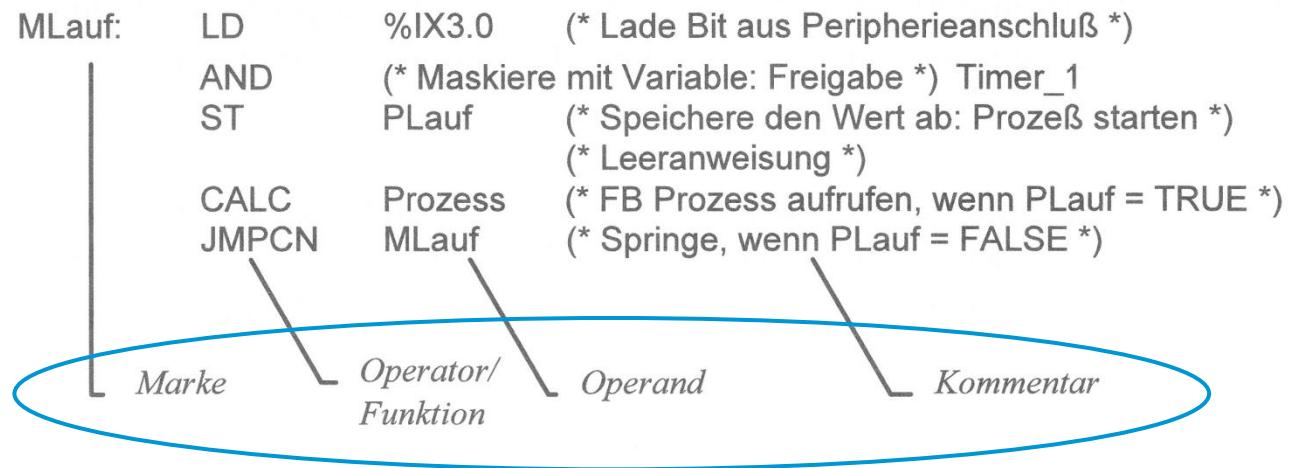


Schwerpunkte im Kapitel

Quelle: Wellenreuther

# KLEINES KOMPENDIUM AWL (1)

- Weit verbreitete SPS-Programmiersprache (noch)
- Schnell, universell, assemblerähnlich
- Virtueller Prozessor mit Akkumulator für Rechen- und Verknüpfungsergebnisse (keine Prozessorflags!): AE = Aktuelles Ergebnis
- Aufbau Programmzeile:



- Wahrscheinlich ab IEC-61131-3 Rev. 04 nicht mehr in der Norm (aktuelle Rev. von 2013)

# KLEINES KOMPENDIUM AWL (2)

## Operatoren mit booleschen Operanden

Operatoren	Auswirkung auf aktuelles Ergebnis (AE) im Akku	Erklärung
LD LDN	Erzeugen	Lädt Operand in AE (Aktuelles Ergebnis) Lädt Operand negiert in AE
AND ANDN	Weiter verarbeiten	UND-Verknüpfung des Operanden mit AE UND-Verknüpfung des negierten Operanden mit AE
AND( ANDN(	Undefiniert setzen	Operanden mit AE verknüpfen (Klammer auf) Operanden mit AE verknüpfen (Klammer auf)
OR ORN	Weiter verarbeiten	ODER-Verknüpfung des Operanden mit aE ODER-Verknüpfung des negierten Operanden mit AE
OR( ORN(	Undefiniert setzen	Operanden mit AE verknüpfen (Klammer auf) Operanden mit AE verknüpfen (Klammer auf)
XOR XORN	Weiter verarbeiten	Exklusiv-ODER-Verknüpfung des Operanden mit AE Exklusiv-ODER-Verknüpfung des negierten O.m. AE
XOR( XORN(	Undefiniert setzen	Operanden mit AE verknüpfen (Klammer auf) Negierten Operanden mit AE verknüpfen (Kl. auf)
ST STN	Unverändert	Speichert AE in den Operanden Speichert AE negiert in den Operanden
S	Unverändert	Setzt Operand auf TRUE, wenn AE = 1
R	Unverändert	Setzt Operand auf FALSE, wenn AE = 1
)	Unverändert	Schließen der Klammerebene

## Operatoren mit Operanden vom allgemeinen Datentyp (Byte, Word, Int, Real, Time,..)

Operatoren	Auswirkung auf aktuelles Ergebnis (AE) im Akku	Erklärung
LD	Erzeugen	Lädt Operand in AE (Aktuelles Ergebnis)
ST	Unverändert	Speichert AE als Operand
ADD ADD()	Weiter verarbeiten	Addiert Operand auf AE
SUB SUB()	Weiter verarbeiten	Subtrahiert Operand von AE
MUL MUL()	Weiter verarbeiten	Multipliziert Operand mit AE
DIV DIV()	Weiter verarbeiten	Dividiert AE durch Operand
GT GT()	Weiter verarbeiten	AE größer Operand
GE GE()	Weiter verarbeiten	AE größer gleich Operand
EQ EQ()	Weiter verarbeiten	AE gleich Operand
NE NE()	Weiter verarbeiten	AE ungleich Operand
LE LE()	Weiter verarbeiten	AE kleiner gleich Operand
LT LT()	Weiter verarbeiten	AE kleiner Operand
)	Undefiniert setzen	Schließen der Klammerebene

Quelle: Linnemann

# KLEINES KOMPENDIUM AWL (3)

## Operatoren für Sprünge und Aufrufe

Operatoren	Auswirkung auf aktuelles Ergebnis (AE) im Akku	Erklärung
JMP	Unverändert oder undefiniert <sup>1)</sup>	Unbedingter Sprung zu einer Marke
JMPC	Unverändert	Sprung zur Marke, wenn VE = 1
JMPCN	Unverändert	Sprung zur Marke, wenn VE = 0
CAL	Unverändert	Unbedingter Aufruf eines Funktionsbausteins
CALC	Unveränd. für folgenden Operator Undefiniert im aufgerufenen FB	Aufruf eines Funktionsbausteins, wenn VE = 1
CALCN	Unveränd. für folgenden Operator Undefiniert im aufgerufenen FB	Aufruf eines Funktionsbausteins, wenn VE = 0
RET	FB: Undefiniert FUN: Funktionswert	Unbedingter Rücksprung aus FB oder FUN
RETC	Unverändert	Rücksprung aus FB oder FUN, wenn VE = 1
RETCN	Unverändert	Rücksprung aus FB oder FUN, wenn VE = 0
FUNname	Weiter verarbeiten	Aufruf einer Funktion



- Weit verbreitete SPS-Sprache (noch)
- Für alle Steuerungssysteme verfügbar
- Sehr hohe Flexibilität im Vergleich zu KOP und FUP
- Hohe Effizienz durch Maschinen- und Steuerungsoptimierung
- Vollständiger Befehlsumfang der SPS verfügbar
- Kann andere grafische und textuelle Sprachen (AS) abbilden



- Unübersichtlich und fehleranfällig bei größeren Programmen / komplizierten Abläufen
- Wenig geeignet als alleinige Dokumentation
- Sprachmittel zur strukturierten Programmierung fehlen, keine Datenstrukturen verfügbar
- Abgekündigt (nicht mehr im aktuellen CoDeSys enthalten)

# KLEINES KOMPENDIUM ST (1)

- Textuelle Programmiersprache, ähnlich C oder PASCAL
- Mächtige Sprachkonstrukte -> gut für umfangreiche, komplexe Berechnungen
- Vorteile gegenüber AWL:
  - Kompakte, gute verständliche Formulierung
  - Strukturierung des Programms in übersichtliche Anweisungsblöcke
  - Gute Möglichkeit der Steuerung von Programmflüssen
- Nachteile gegenüber AWL:
  - Programm weniger laufzeiteffizient = langsamer
  - Speicherplatzverbrauch ist höher
- Merke: Anweisungen sind durch **Semikolon ;** getrennt!  
Kommentare wie in AWL (\* ..... \*)

## Operatoren

Operation	Symbol	Bemerkung / Beispiel
Klammerung	(<Ausdruck>)	X := (a+b) * (a-b);
Funktionsaufruf	<Funktionsname> (<Argumentliste>)	LN(a); MAX(x,y);
Potenzierung	**	X := 3 ** 2;
Negation	-	unäres Minus, Vorzeichen
Komplement	NOT	
Multiplikation	*	X := 3 * 5;
Division	/	X := 15 / 3;
Modulo	MOD	Rest bei Integer-Division
Addition	+	X := 2 + 3;
Subtraktion	-	X := 3 - 2;
Vergleiche	<, >, <=, >=, <>, =	IF a < b THEN ...
Boolesches UND	&, AND	beide Operatoren möglich
Bool. EXKLUSIV-ODER	XOR	IF a XOR b THEN ...
Boolesches ODER	OR	IF a OR b THEN ...

# KLEINES KOMPENDIUM ST (2)

## Anweisungen:

Anweisung	Bezeichnung	Beispiel
<code>:=</code>	Zuweisung	<code>X := 10;</code>
	Funktionsaufruf	<code>MyFun (P1:=10; P2:=20);</code>
<code>RETURN</code>	Rücksprung	<code>RETURN</code>
<code>IF</code>	Verzweigung (Bedingte Anweisung)	<code>IF (a &lt; 10) THEN     a := a + 5; END_IF;</code>
<code>CASE</code>	Multiauswahl	<code>CASE fall OF     1: a := 11;     2: a := 22;     ELSE a := 0; END_CASE</code>
<code>FOR</code>	Zählschleife	<code>FOR A := 1 TO 10 BY 2 DO     a := a * a; END_FOR</code>
<code>WHILE</code>	Geschlossene Schleife	<code>WHILE a &lt; 10 DO     a = a + 2; END WHILE</code>
<code>REPEAT</code>	Offene Schleife	<code>REPEAT     a := a + 2;     UNTIL a &lt; 10; END_REPEAT</code>
<code>EXIT</code>	Schleifenabbruch	<code>EXIT;</code>
	Leeranweisung	<code>;;</code>

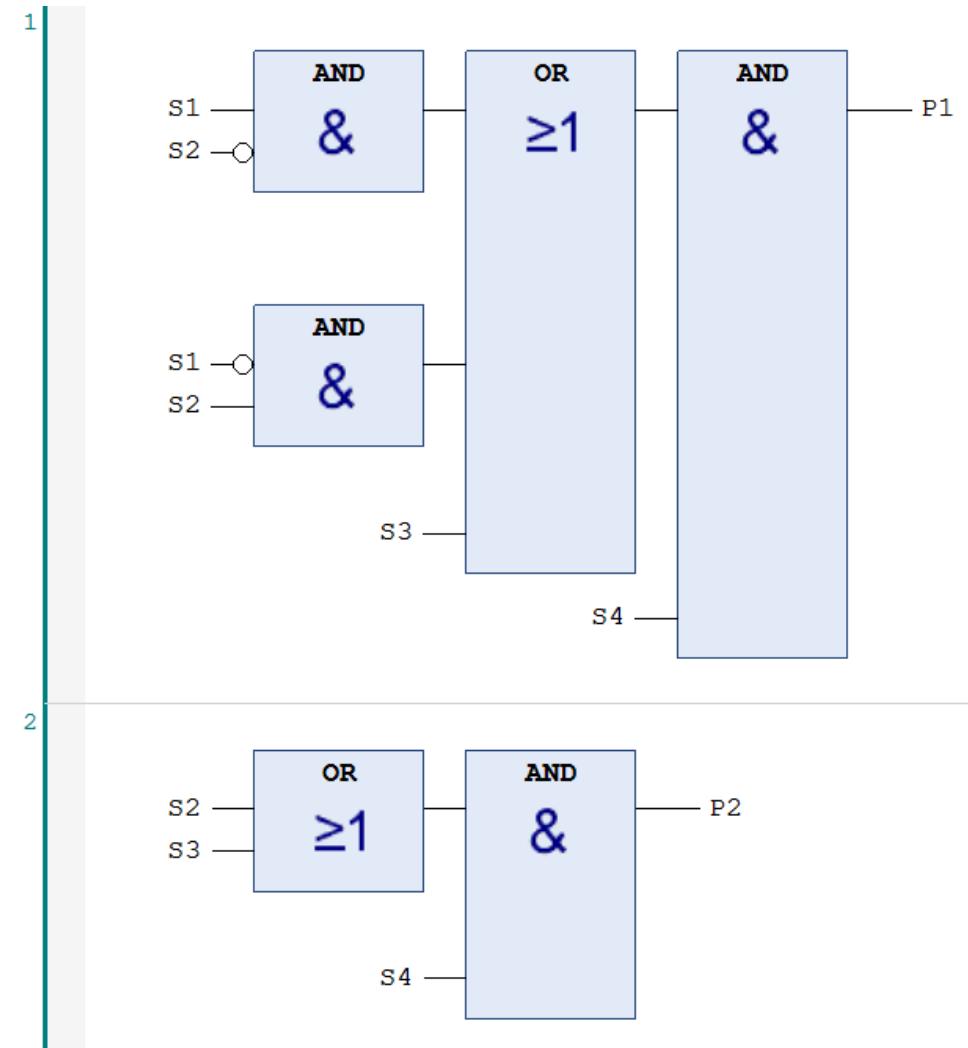
## Auswahl an Standard-Funktionen:

Name	Beschreibung
Allgemeine numerische Funktionen	
<code>ABS</code>	Absolutwert
<code>SQRT</code>	Quadratwurzel
Logarithmus-Funktionen	
<code>LN</code>	Natürlicher Logarithmus
<code>LOG</code>	Logarithmus zur Basis 10
<code>EXP</code>	Exponentialfunktion (e-Funktion)
Trigonometrische Funktionen	
<code>SIN</code>	Sinus mit Eingang im Bogenmaß
<code>COS</code>	Cosinus mit Eingang im Bogenmaß
<code>TAN</code>	Tangens mit Eingang im Bogenmaß
<code>ASIN</code>	Arcsin, Hauptwert
<code>ACOS</code>	Arcos, Hauptwert
<code>ATAN</code>	Arctan, Hauptwert
Funktionen zur Typumwandlung	
<code>*_TO_*</code>	Wandelt Typ * in Typ * um

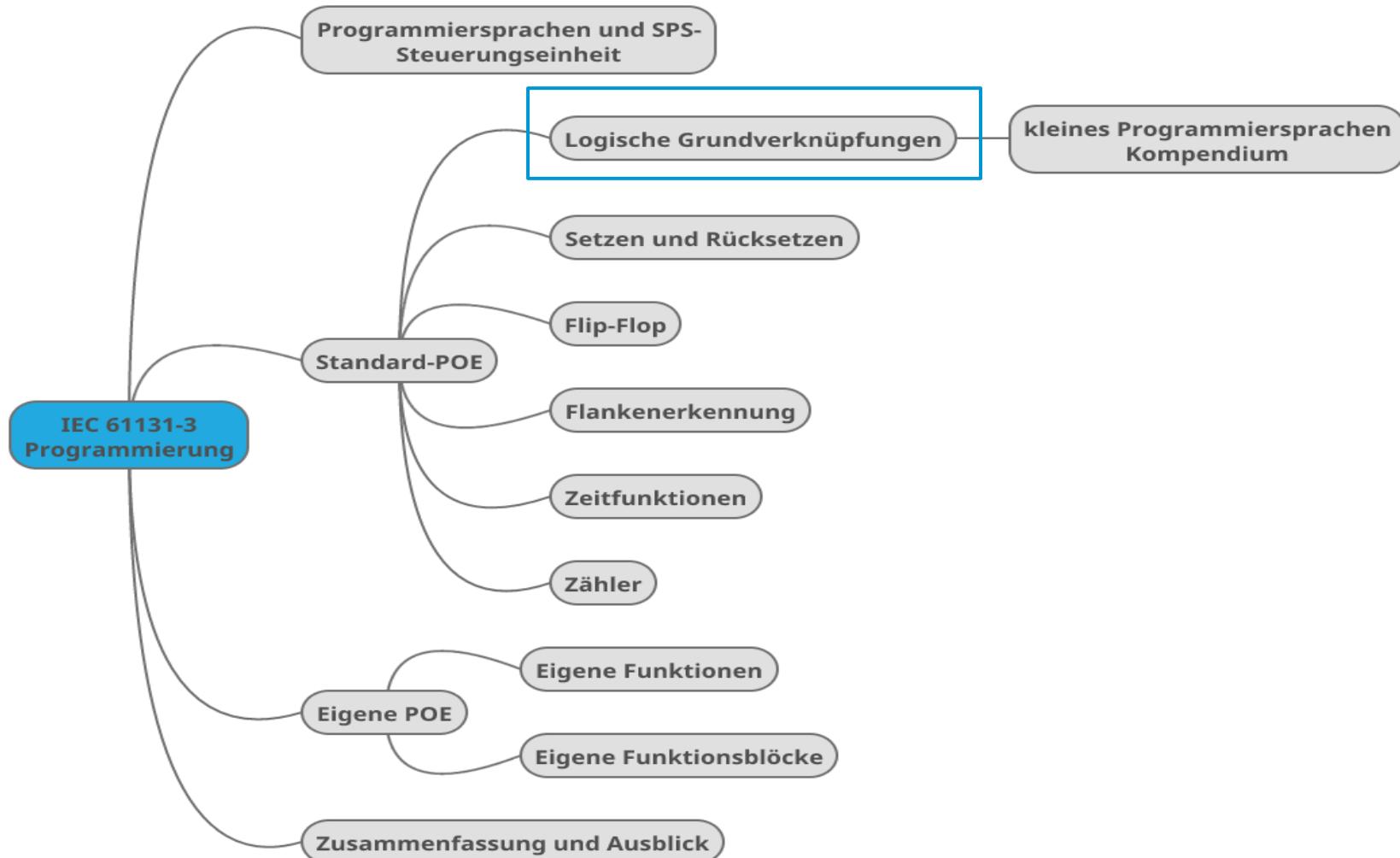
Name	Beschreibung
Arithmetische Standardfunktionen	
<code>ADD</code>	<code>+</code>
<code>MUL</code>	<code>*</code>
<code>SUB</code>	<code>-</code>
<code>DIV</code>	<code>/</code>
<code>MOD</code>	<code>MOD</code>
<code>EXPT</code>	<code>**</code>
<code>MOVE</code>	<code>:=</code>
Funktionen zur Auswahl	
<code>MAX</code>	Ermittelt Maxi-/Minimum einer Daten- aufzählung, z.B. MAX W1,W2,W3
<code>MIN</code>	
Funktionen für Bitmanipulation	
<code>SHL</code>	Schiebt Bits nach links (Byte, Word)
<code>SHR</code>	Schiebt Bits nach rechts (Byte, Word)
<code>ROR</code>	Bits rechts rotieren (Byte, Word)
<code>ROL</code>	Bits links rotieren (Byte, Word)

# KLEINES KOMPENDIUM FUP (1)

- In Europa sehr verbreitete Programmiersprache für einfache Verknüpfungslogik (in den USA eher KOP verbreitet)
- Aufteilung des POE in Netzwerke, welche die Anweisung von oben nach unten und von links nach rechts abarbeiten
- Die Logik wird durch „Boxen“ repräsentiert, die Logik in der Box auf die Eingangsvariablen anwenden und das Ergebnis der Logik auf die Ausgangsvariablen ausgeben
- Im Signalfluss können mehrere Boxen oder Signalmanipulationen (Negation) enthalten sein



# KAPITEL 5: LOGISCHE GRUNDVERKNÜPFUNGEN



# BINÄRE (LOGISCHE) GRUNDVERKNÜPFUNGEN

Grundverknüpfungen **UND / ODER / NICHT** in Darstellung  
Boolesche Algebra / Zeitdiagramm / FUP / AWL / ST:

Funktion	Zeitdiagramm	Funktionsplan (FUP)	ST	AWL
<b>UND</b> $A = E1 \wedge E2$ $A = E1 \& E2$ $A = E1 \cdot E2$	E1 E2 A	E1 -- AND --> A E2 -- AND --> A	A := E1 AND E2;	LD E1 AND E2 ST A
<b>ODER</b> $A = E1 \vee E2$	E1 E2 A	E1 -- OR --> A E2 -- OR --> A	A := E1 OR E2;	LD E1 OR E2 ST A
<b>NICHT</b> $A = \overline{E} = !E$	E A	E -- NOT --> A	A := NOT E;	LD E NOT ST A
<b>Ausgangs-NEGATION</b> $A = \overline{E1 \wedge E2}$	E1 E2 A	E1 -- AND --> A E2 -- AND --> A	A := NOT(E1 AND E2);	LD E1 AND E2 STN A

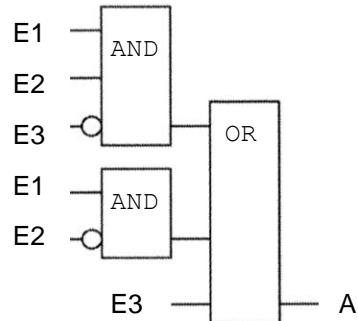
Quelle: Wellenreuther

# ZUSAMMENGESETzte BINÄRE GRUNDVERKNÜPFUNGEN

## UND-vor-ODER-Verknüpfung

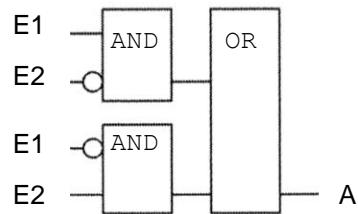
### 1) Allgemeiner Fall

$$A = E_1 E_2 \bar{E}_3 \vee E_1 \bar{E}_2 \vee E_3$$



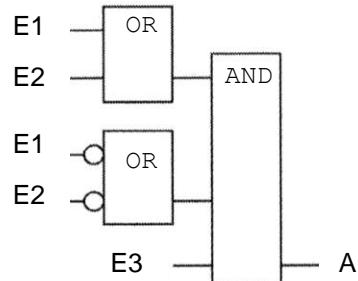
### 2) Spezieller Fall: Antivalenz (Exklusiv-ODER)

$$A = E_1 \neq E_2$$



## ODER-vor-UND-Verknüpfung

$$A = (E_1 \vee E_2) \wedge (\bar{E}_1 \vee \bar{E}_2) \wedge E_3$$



# EXKURS: KARNAUGHT-VEITCH-DIAGRAMM

Ziel: Optimierung von Bool'schen Ausdrücken

Zu optimierender Ausdruck

$$C = (\neg B \text{ AND } \neg A) \text{ OR } (\neg A \text{ AND } B) \text{ OR } (A \text{ AND } \neg B)$$



Schritt 1:  
Wahrheitstabelle aufstellen

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Schritt 2:

Tabelle ins KV-Diagramm übertragen und Gruppen aus  $2^n$  zusammenhängenden Einsen bilden.

Bedingungen:

- Alle Einsen müssen in mindestens einer Gruppe sein.
- Größe der Gruppen maximal.
- Anzahl der Gruppen minimal.



	A	$\neg A$
B	0	1
$\neg B$	1	1

Schritt 3:

- Aus jeder Gruppe entsteht eine Und-Verknüpfung der Zustände. **Eingänge, die beide Zustände in der Gruppe haben, werden weggelassen.**
- Die Gruppen werden mit ODER verknüpft.



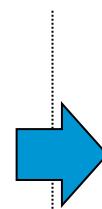
$$C = \neg B \text{ OR } \neg A$$

# BEISPIEL: KARNAUGHT-VEITCH-DIAGRAMM

Beispiel mit 3 Eingängen

$$A = (\neg E_1 \text{ AND } \neg E_2 \text{ AND } E_3) \text{ OR } (\neg E_1 \text{ AND } E_2 \text{ AND } \neg E_3) \text{ OR } (\neg E_1 \text{ AND } E_2 \text{ AND } E_3) \text{ OR } (E_1 \text{ AND } \neg E_2 \text{ AND } E_3)$$

E1	E2	E3	A
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



	E1	E1	$\neg E_1$	$\neg E_1$
E2	0	0	1	1
$\neg E_2$	1	0	0	1
	E3	$\neg E_3$	$\neg E_3$	E3

$(\neg E_1 \text{ AND } E_2)$   
 $A =$   
 $(E_2 \text{ AND } \neg E_1)$   
OR  
 $(\neg E_2 \text{ AND } E_3)$   
 $(\neg E_2 \text{ AND } E_3)$

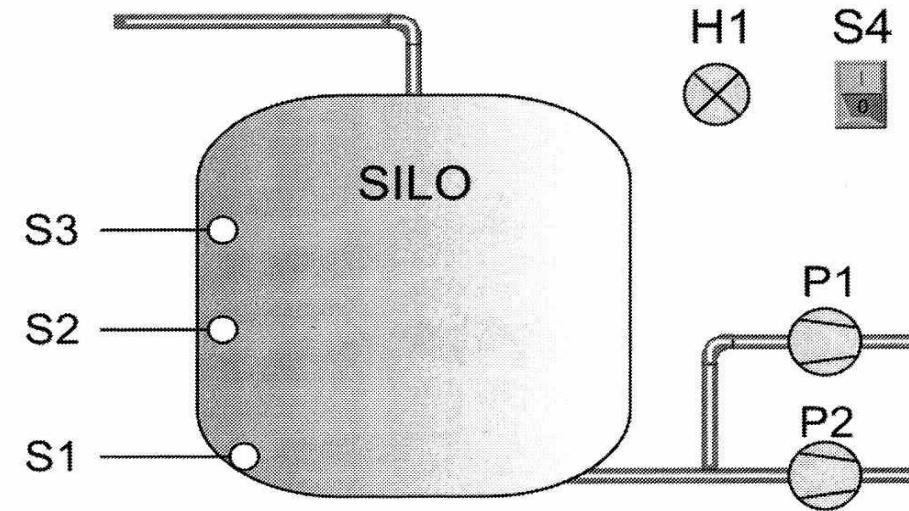
Gruppen können auch über die Tabelle hinaus gebildet werden!

# BEISPIEL: SILOENTLEERUNG

Der Inhalt eines Silos kann über die Pumpen P1 und P2 entleert werden. Welche der beiden Pumpen bei der Entleerung des Silos eingeschaltet sind, ist abhängig vom Silofüllstand. Befindet sich der Füllstand unterhalb von Sensor S2, ist Pumpe P1 einzuschalten. Liegt der Füllstand zwischen Sensor S2 und Sensor S3, wird die Pumpe P2 eingeschaltet. Bei Füllstand oberhalb von S3 laufen beide Pumpen. Die Entleerung des Silos wird mit dem Schalter S4 ein- und ausgeschaltet.

Beim Auftreten eines Sensorfehlers (z.B. S3 meldet und S2 meldet nicht) werden beide Pumpen und eine Störungsanzeige H1 eingeschaltet.

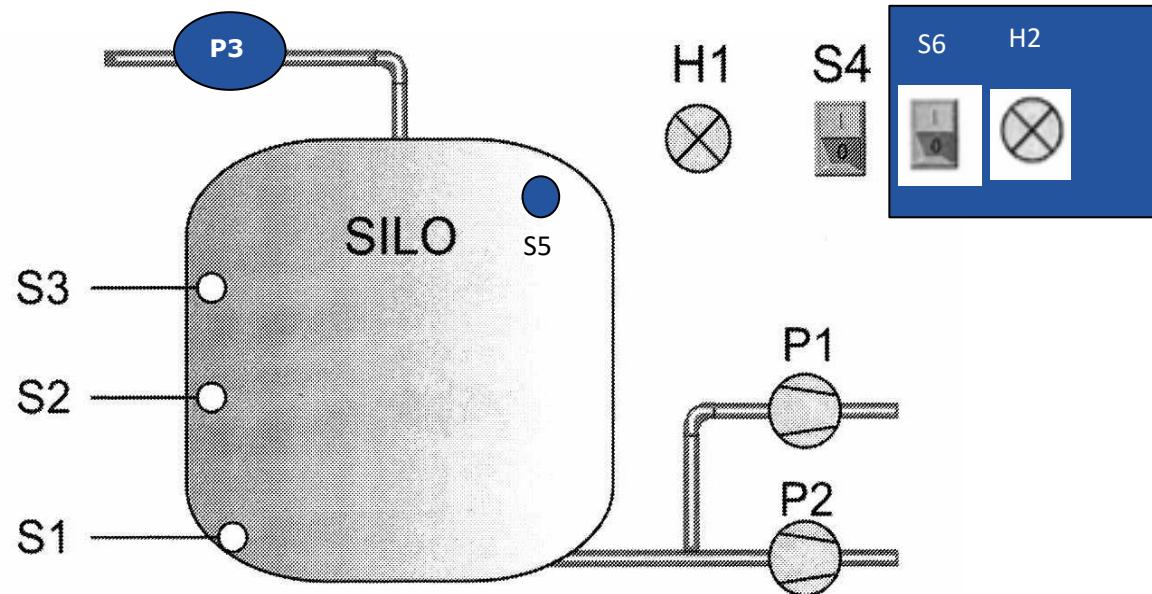
## Technologieschema:



## Aufgaben = Systematische Lösung:

- Funktionstabelle: Eingänge S3,S2,S1 -> Ausgänge P1,P2,H1 aufstellen!
- Boolesche Schaltfunktionen aus der Funktionstabelle ermitteln! (vereinfachen!)
- Kodierung in CoDeSYS!

# BEISPIEL: SILOENTLEERUNG (ERWEITERUNG)

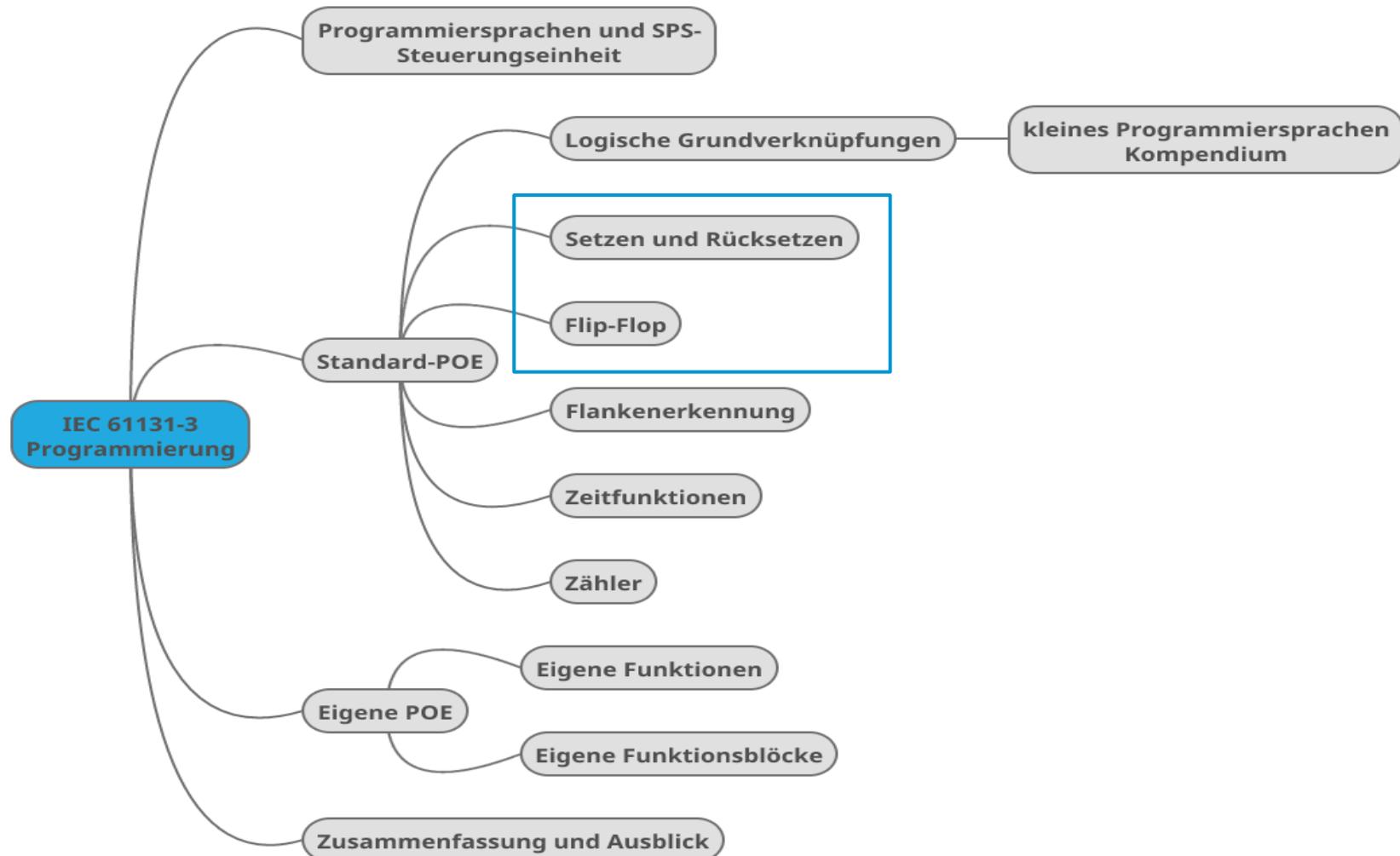


Erweitern Sie Ihre Funktion in ST und FUP um:

- Die Pumpe P3 geht an, wenn der Schalter S6 aktiv ist
- Die Pumpe P3 geht aus, wenn der Sensor S5 aktiv ist
- Ist der Behälter zu voll (S5), dann geht die Pumpe P3 aus und die Lampe H2 an.
- Alles wird über den Schalter S4 freigeschaltet

Frage: Netzwerke erweitern oder neue Netzwerke anlegen?

# KAPITEL 5: SPEICHERFUNKTIONEN



# SETZEN(S), RÜCKSETZEN(R)

Operator	ST	FUP	AWL
S	IF (E1 OR E2) THEN A := 1; END_IF;		LD E1 OR E2 S A
R	IF (E1 OR E2) THEN A := 0; END_IF;		LD E1 OR E2 R A

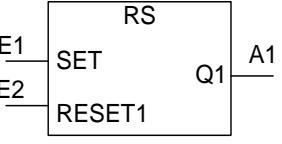
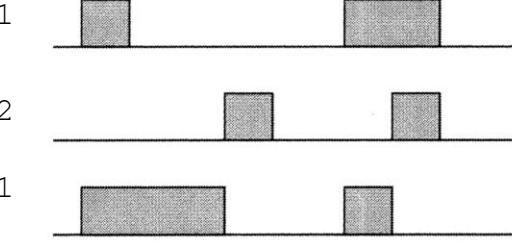
Einfluss vom „AE“:

AE=1 -> A wird gesetzt = „1“  
 AE=0 -> keine Änderung  
 Rücksetzen durch R

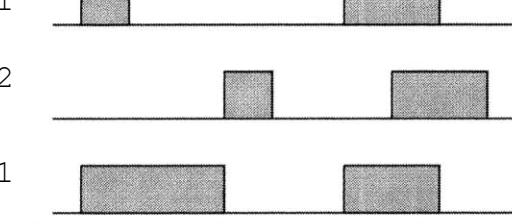
AE=1 -> A wird gelöscht = „0“  
 AE=0 -> keine Änderung  
 Setzen durch S

# RS- / SR-FLIPFLOP

## Speichern mit vorrangigem Rücksetzen

FUP	AWL	Zeitdiagramm	ST
	LD E1 S A1 LD E2 R A1		IF E1 THEN A1:=1; END_IF; IF E2 THEN A1:=0; END_IF;

## Speichern mit vorrangigem Setzen

FUP	AWL	Zeitdiagramm	ST
	LD E2 R A1 LD E1 S A1		IF E2 THEN A1:=0; END_IF; IF E1 THEN A1:=1; END_IF;

RS- Flipflop = **Element mit Speicherwirkung** mit Setz- (S) und Rücksetzeingang (R);

Optionen: **S- oder R-dominant**;

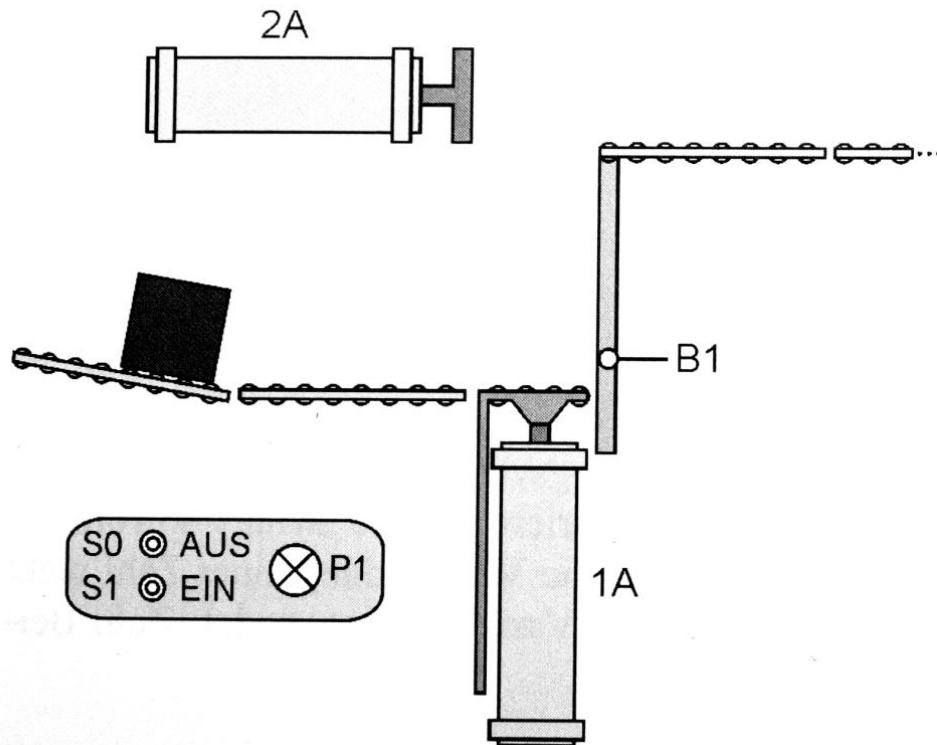
Einsatz: Gegenseitige Verriegelung / Reihenfolgeverriegelung

# FLIPFLOPS BEISPIELAUFGABE: PAKETSCHIEBER

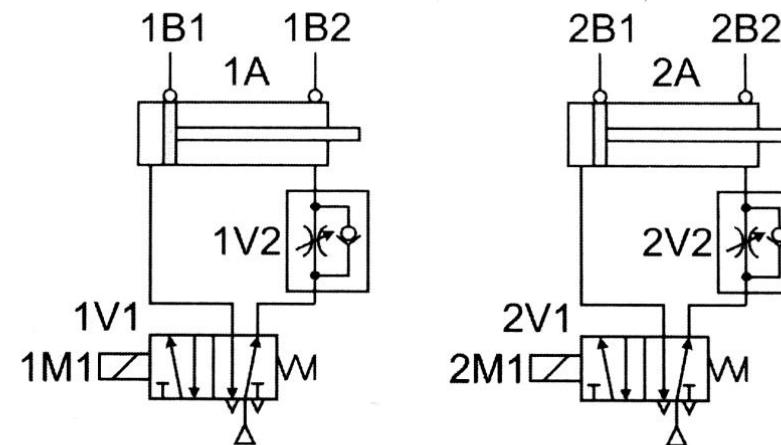
Wenn die Steuerung mit S1 eingeschaltet ist, werden die über einen Rollengang ankommenden Pakete mit dem Zylinder 1A gehoben und mit dem Zylinder 2A auf den nächsten Rollengang geschoben. Der Sensor B1 meldet ein ankommendes Paket. Das Ausschalten der Steuerung mit S0 wird erst nach Beendigung eines Hebevorgangs wirksam. Der eingeschaltete Zustand der Steuerung wird mit P1 angezeigt.

Die beiden Zylinder besitzen jeweils zwei Endlagengeber. Die Ansteuerung der Zylinder erfolgt durch 5/2-Wegeventile mit Federrückstellung und einseitig elektromagnetischer Betätigung.

Technologieschema:

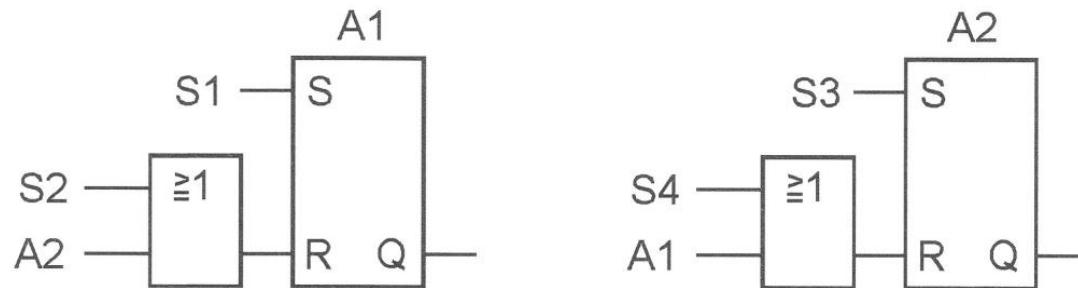


Pneumatischer Schaltplan:

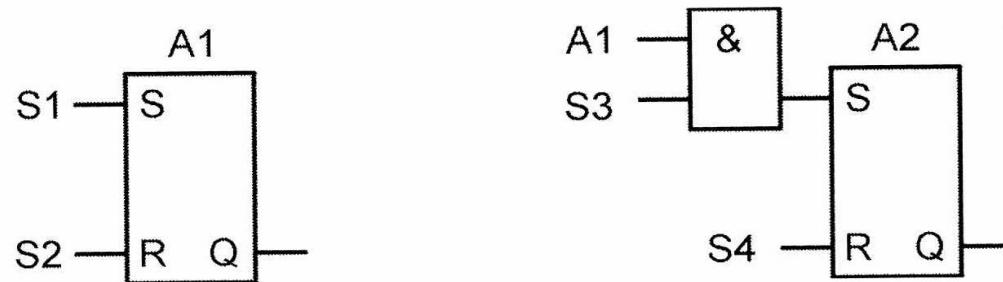


# FLIPFLOPS: GEGENSEITIGES VERRIEGELN

**Prinzip Gegenseitige Verriegelung über Rücksetzeingänge (R-dominate FFs)**  
(hier vereinfachte graphische Darstellung!)



**Prinzip Reihenfolgeverriegelung über Setzeingang: (S-dominante FFs)**

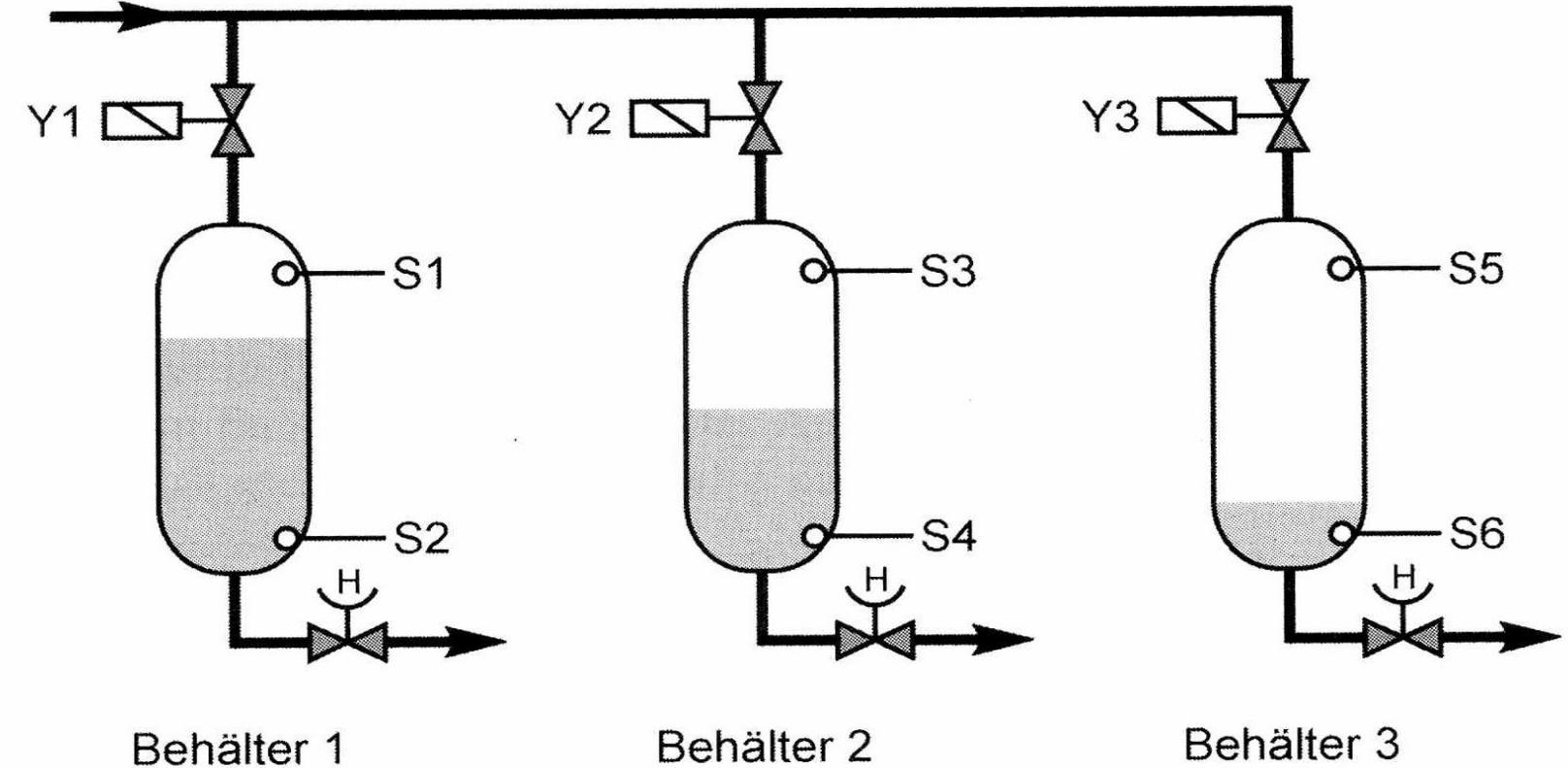


# FLIPFLOPS BEISPIELAUFGABE: BEHÄLTER-FÜLLANLAGE

Die drei Behälter sollen bei einer Leermeldung an den Sensoren S2, S4 oder S6 gefüllt werden. Dazu wird das entsprechende Ventil Y1, Y2 oder Y3 so lange geöffnet, bis der Sensor S1, S3 bzw. S5 einen vollen Tank meldet.

Es soll immer nur ein Behälter gefüllt werden, auch wenn zwei Leermeldungen vorhanden sind. Der zweite Tank wird erst nach Abschluss des ersten Tanks gefüllt

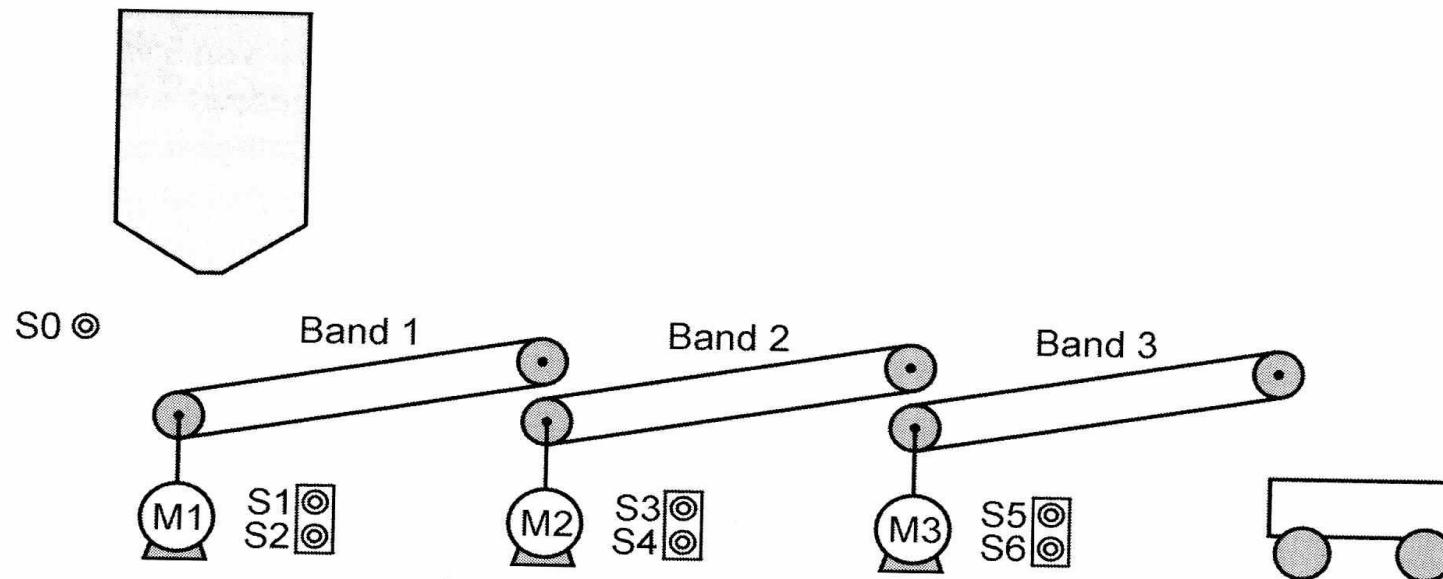
Schreiben Sie ein Programm  
in der Sprache FUP, dass die  
Aufgabe mit Flipflops lst.



# FLIPFLOPS BEISPIELAUFGABE: KIESFÖRDERANLAGE

- Band 1 darf nur eingeschaltet werden wenn Band 2 läuft.
- Band 2 darf nur eingeschaltet werden wenn Band 3 läuft.
- Umgekehrt beim Ausschalten.
- S0 = Not-Aus.

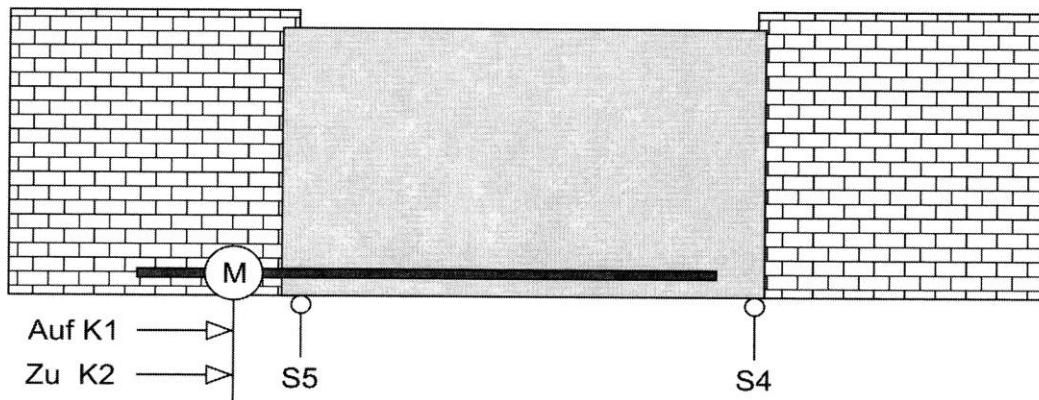
Entwerfen Sie ein Programm in der Sprache FUP, dass die Aufgabe mittels RS-Flipflops löst.



# FLIPFLOPS BEISPIELAUFGABE: TORSTEUERUNG

Ein Werkstor wird mit einem Elektromotor auf und zu gesteuert. Die Ansteuerung des Elektromotors erfolgt mit den Leistungsschützen K1 (Rechtslauf Tor auf) und K2 (Linkslauf Tor zu). Die Endlagen des Schiebetors werden mit den Initiatoren S4 (Tor zu) und S5 (Tor auf) gemeldet.

## Technologieschema:



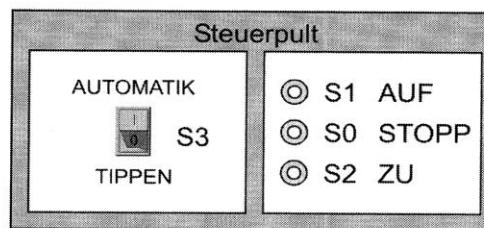
Zur Bedienung des Schiebetors ist beim Pförtner ein Bedienpult angebracht. Steht der Wahlschalter S3 in Stellung Automatik, kann durch kurzzeitiges Drücken des S1- bzw. S2-Tasters das Tor auf- bzw. zugesteuert werden. Ein Umschalten der Drehrichtung des Motors ist nur über S0 (STOPP) möglich.

Bei Betätigung des Tasters S0 bleibt das Tor sofort stehen. Zum Öffnen oder Schließen ist dann eine erneute Betätigung der Taster S1 oder S2 erforderlich. Steht der Wahlschalter S3 in Stellung Tippen, so wird das Werkstor mit den Tastern S1 und S2 nur solange geöffnet bzw. geschlossen, wie die entsprechende Taste betätigt wird.

Die Initiatoren S4 und S5 beenden bei Betätigung jeweils sofort das Öffnen bzw. Schließen des Tores.

## Aufgaben:

- Stellen Sie die Bedingungen der zur Ansteuerung der Setz- und Rücksetzeingänge der beiden RS-FFs K1 und K2 auf.
- Setzen Sie das Programm in der Sprache FUP um.

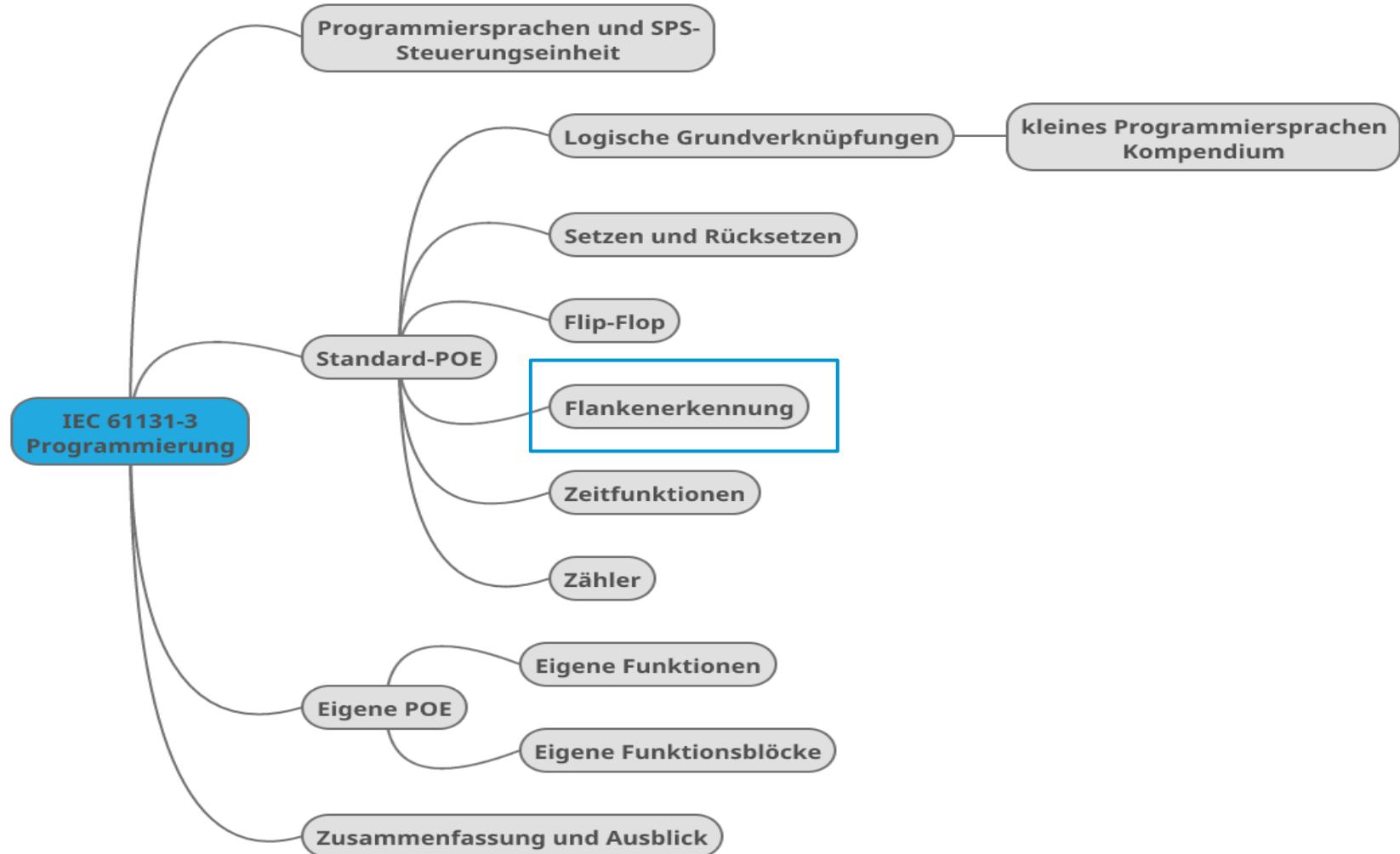


## Hinweis:

S0, S4, S5 sind low-active!

Alle anderen Signale sind high-active!

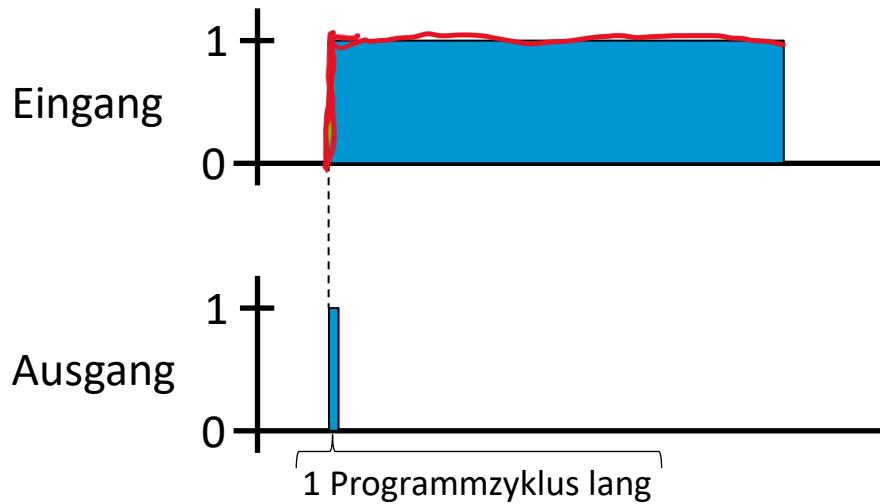
# KAPITEL 5: FLANKENERKENNUNG



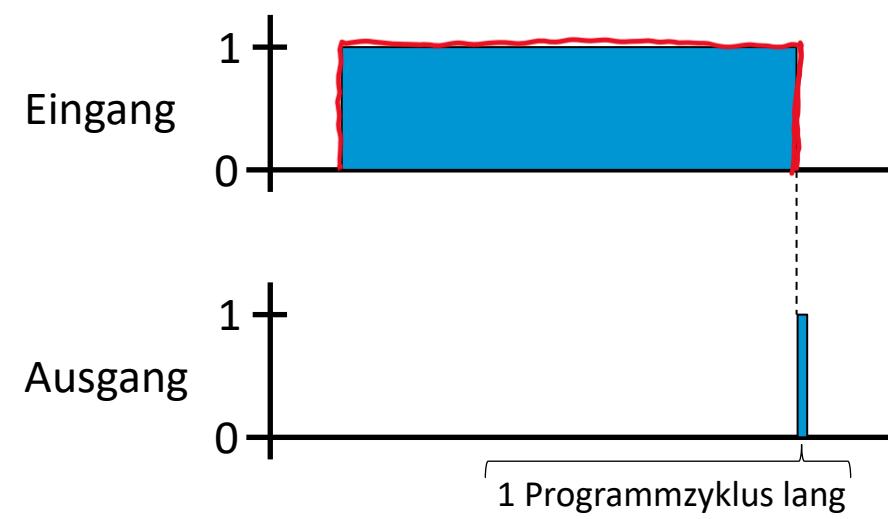
# FLANKENAUSWERTUNG - GRUNDPRINZIP

- Es wird die Änderung eines Signalzustandes erfasst
  - Wechsel 0  $\rightarrow$  1 : Positive Flanke (Steigende Flanke)
  - Wechsel 1  $\rightarrow$  0 : Negative Flanke (Fallende Flanke)
- Aus einem Dauersignal wird ein Impuls gebildet, der einen Zyklus lang ist

**Positive Flankenauswertung (rising edge)**



**Negative Flankenauswertung (falling edge)**



# FLANKENAUSWERTUNG BEISPIELAUFGABE: TOGGLE-SCHALTER

## Problemstellung:

Sie wollen mit einem Schalter (S1) die Funktionalität Ein- und Ausschalten einer Anlage realisieren (Toggle-Schalter).

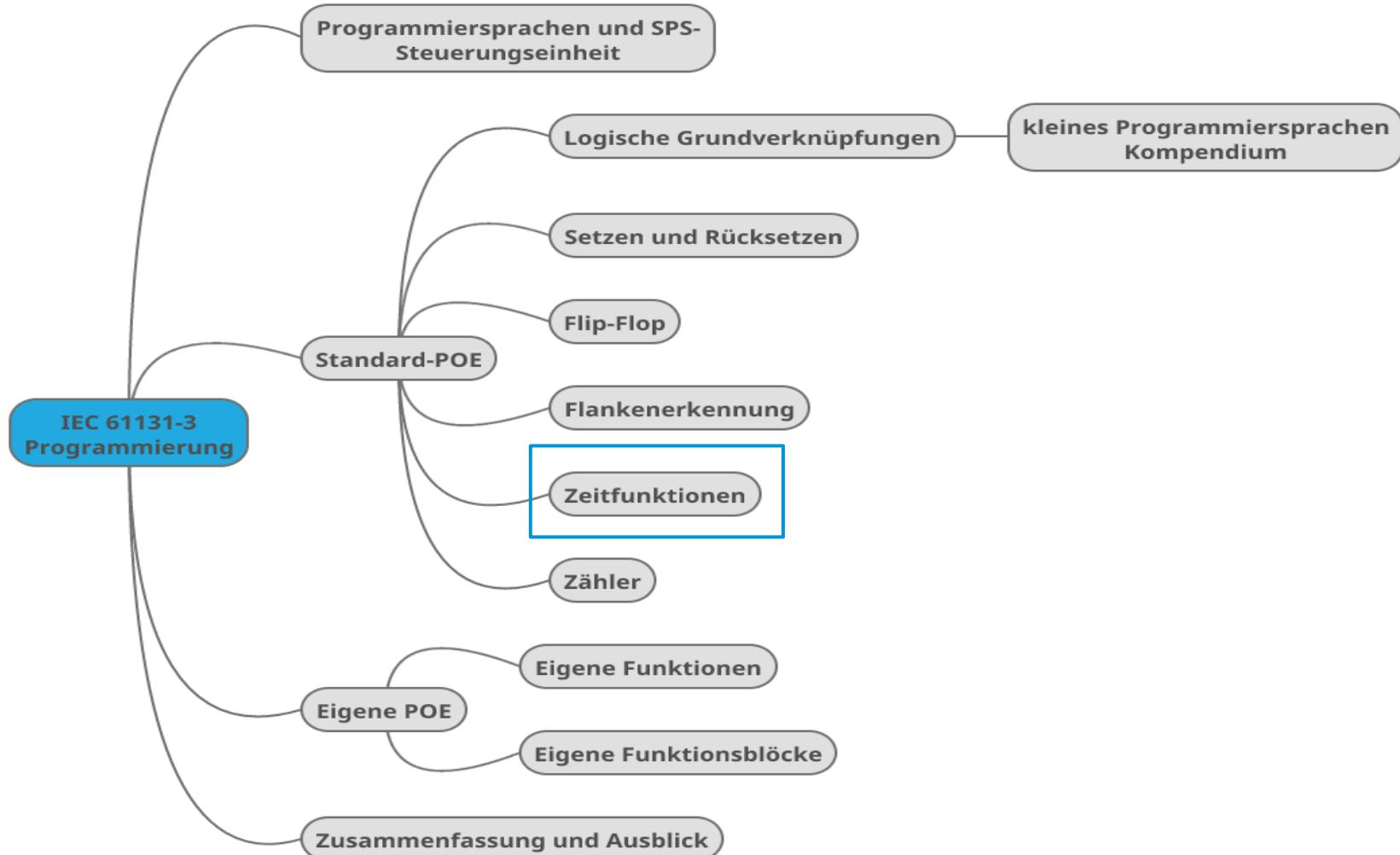
Dabei soll die Variable „Anlage\_an“ auf TRUE gesetzt werden, wenn S1 gedrückt wird und die Anlage ist aus („Anlage\_an“ == 0).

Wenn die Anlage eingeschaltet ist („Anlage\_an“ == 1 ) und S1 gedrückt wird, wird „Anlage\_an“ auf FALSE gesetzt.

## Aufgabe:

Schreiben Sie ein Programm in der Sprache FUP und nutzen dabei die Flankenauswertung zur Lösung der Aufgabe.

# KAPITEL 5: ZEITFUNKTIONEN



# ZEITFUNKTIONEN: TP - IMPULS

Name	Funktion Grafische Darstellung	Zeitdiagramm
TP	<p>Erzeugen eines Impulses</p> <pre> +-----+        TP     BOOL---  IN    Q  ---BOOL TIME---  PT    ET  ---TIME +-----+ </pre>	<p>The diagram shows two horizontal axes labeled 't' (time) at the bottom. The top axis is labeled 'IN' and has a step-up from 0 to 1 at time t=0. The bottom axis is labeled 'Q' and has three rectangular pulses. Each pulse is triggered by a rising edge of 'IN'. The width of each pulse is labeled 'PT' and the height is 1. The time between the start of one pulse and the next is labeled 'ET'. The area under the 'Q' pulse is shaded grey.</p>
AWL		<p>IN = Startbedingung : steigende Flanke!</p>
<pre> VAR  TPIInst: TP; T_ab: TIME; END_VAR CAL TPIInst(IN:= %IX0.1, PT:=T#4s) LD  TPIInst.Q ST  %QX4.1 LD  TPIInst.ET ST  T_ab </pre>		<p>PT = Zeitwertvorgabe</p>
FUP		<p>Q = Binärer Zustand des Zeitgebers</p>
<pre> VAR  TPIInst: TP; T_ab: TIME; END_VAR TPIInst(IN:= %IX0.1, PT:=T#4s); %QX4.1:= TPIInst.Q; T_ab:= TPIInst.ET </pre>		<p>ET = Aktueller Zeitwert</p>

Quelle: Wellenreuther

# ZEITFUNKTIONEN: TON - EINSCHALTVERZÖGERUNG

Name	Funktion Grafische Darstellung	Zeitdiagramm
TON	Einschaltverzögerung <pre>+-----+         TON     BOOL---  IN      Q  ---BOOL TIME---  PT      ET  ---TIME +-----+ (Statt TON auch: T---0)</pre>	
AWL		<p>IN = Startbedingung : <b>steigende Flanke</b></p> <p>startetet TON</p> <p>PT = Zeitwertvorgabe</p> <p>Q = Binärer Zustand des Zeitgebers : (<math>t \geq PT</math> und <math>IN=1</math>) <math>\Rightarrow Q</math> auf „1“</p> <p>ET = Aktueller Zeitwert</p> <p>Quelle: Wellenreuther</p>
<pre>VAR TONInst: TON; T_ab: TIME; END_VAR CAL TONInst(IN:= %IX0.2, PT:=T#4s) LD TONInst.Q ST %QX4.2 LD TONInst.ET ST T_ab</pre>		
ST		
<pre>VAR TONInst: TON; T_ab: TIME; END_VAR TONInst(IN:= %IX0.2, PT:=T#4s); %QX4.2:= TONInst.Q; T_ab:= TONInst.ET</pre>		

# ZEITFUNKTIONEN: TOF - AUSSCHALTVERZÖGERUNG

Name	Funktion Grafische Darstellung	Zeitdiagramm
TOF	<p>Ausschaltverzögerung</p> <p>+-----+   TOF   BOOL---  IN Q  ---BOOL TIME---  PT ET  ---TIME +-----+</p> <p>(Statt TOF auch: 0---T)</p>	

AWL	FUP
<pre>VAR TOFInst: TOF; T_ab: TIME; END_VAR CAL TOFInst(IN:= %IX0.3, PT:=T#4s) LD TOFInst.Q ST %QX4.3 LD TOFInst.ET ST T_ab</pre>	<pre>TOFInst   TOF     IN --&gt; %IX0.3     PT --&gt; T#4s     Q --&gt; %QX4.3     ET --&gt; T_ab</pre>
ST	
<pre>VAR TOFInst: TOF; T_ab: TIME; END_VAR TOFInst(IN:= %IX0.3, PT:=T#4s); %QX4.3:= TOFInst.Q; T_ab:= TOFInst.ET</pre>	

IN = Startbedingung : fallende Flanke startet TOF

PT = Zeitwertvorgabe

Q = Binärer Zustand des Zeitgebers

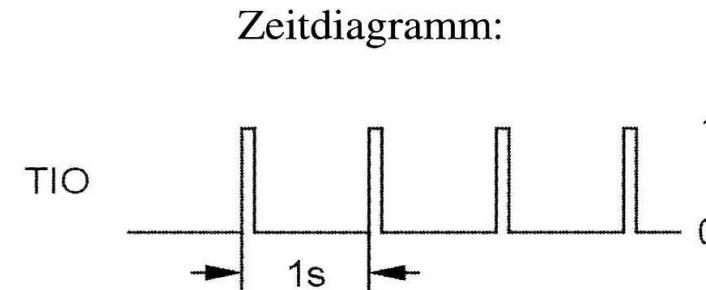
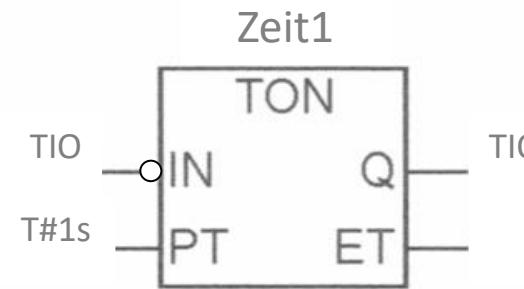
ET = Aktueller Zeitwert

Quelle: Wellenreuther

# ZEITFUNKTIONEN: BEISPIEL TAKTGEBER

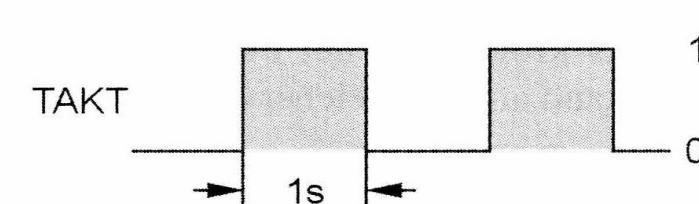
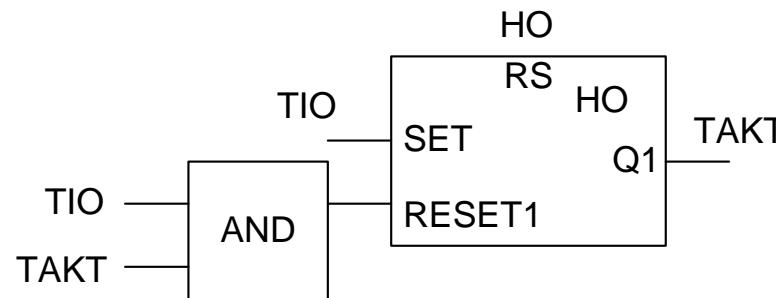
## Schritt 1 (Verwendung TON-Funktion)

- Bei Programmstart TIO=0
- TIO-Ausgang führt nach jeweils 1Sekunde für die Dauer eines Programmzyklus lang ein „1“ Signal

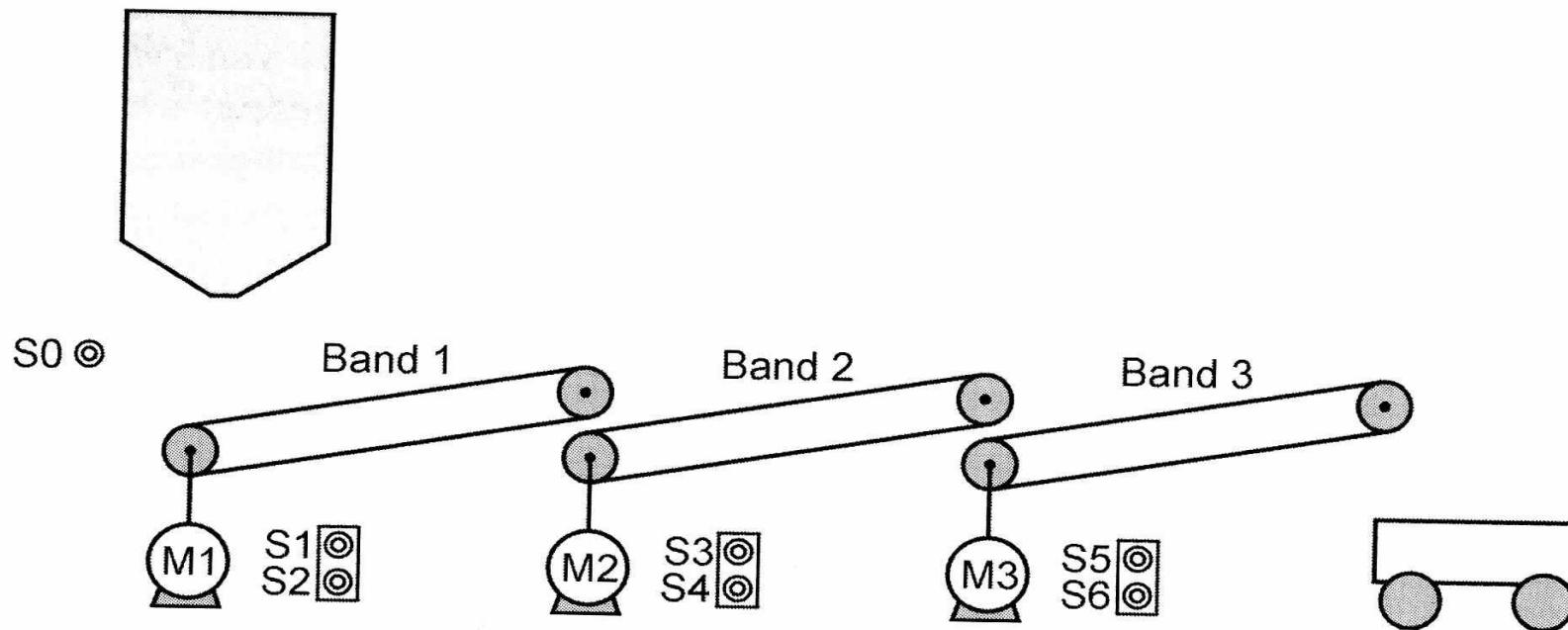


## Schritt 2:

TIO-Signal steuert Binäruntersetzer an: → periodisches Taktsignal

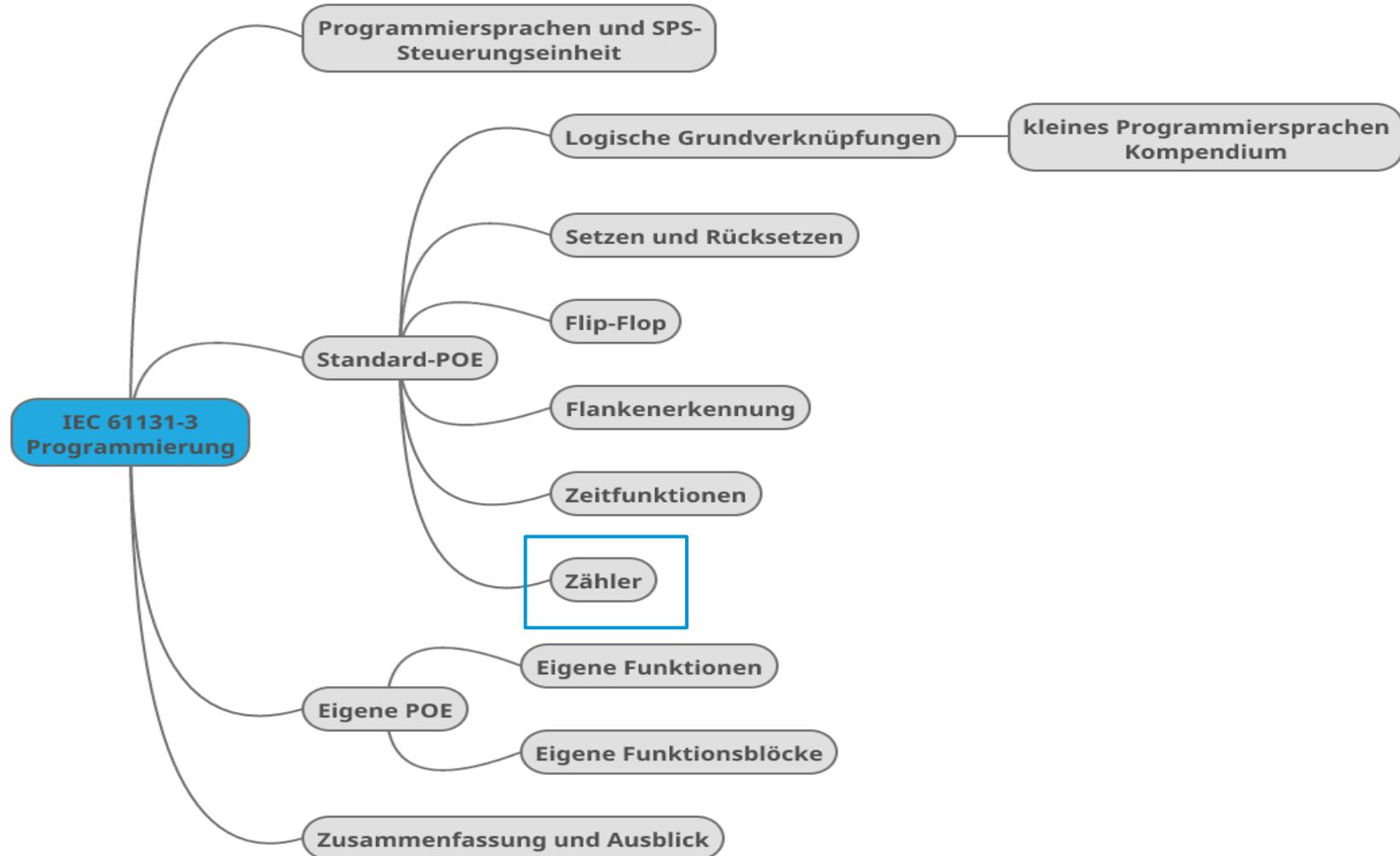


# ZEITFUNKTIONEN BEISPIELAUFGABE: KIESFÖRDERANLAGE 2



- Band 1 kann erst eingeschaltet werden, wenn Band 2 seit 5 Sekunden läuft.
- Band 2 kann erst eingeschaltet werden, wenn Band 3 seit 7 Sekunden läuft.
- Wenn Band 1 für 5 Sekunden ausgeschaltet ist, kann Band 2 ausgeschaltet werden.
- Wenn Band 2 für 5 Sekunden ausgeschaltet ist, kann Band 3 ausgeschaltet werden.

# KAPITEL 5: ZÄHLER



# ZÄHLER: CTU AUFWÄRTSZÄHLER

AWL	FUP
<pre> VAR  CTUInst: CTU;  END_VAR CAL CTUInst(CU:=%IX0.0, _RESET:=%IX0.1,             PV:=250) LD   CTUInst.CV ST   %MW10 LD   CTUInst.Q ST   %QX4.1 </pre>	
ST	
<pre> VAR  CTUInst: CTU  END_VAR CTUInst(CU:=%IX0.0, RESET:=%IX0.1, PV:=250); %QX4.1:= CTUInst.Q; %MW10:= CTUInst.CV; </pre>	

CU : BOOL      Counter Up : **positive Flanke** -> CV := CV+1

PV : WORD      Zählgrenze

CV : WORD      Counter Value = Zählerstand

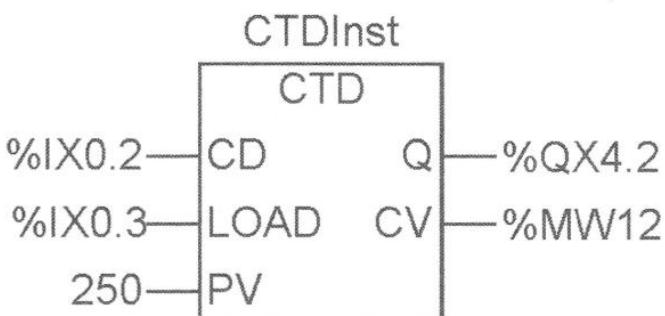
RESET : BOOL    TRUE -> CV := 0

Q : BOOL        TRUE, falls CV >= PV

Achtung: Norm sagt INT,  
CoDeSys: WORD bzw. DWORD

Quelle: Wellenreuther

# ZÄHLER: CTD ABWÄRTSZÄHLER

AWL	FUP
<pre>VAR   CTDInst: CTD; END_VAR CAL  CTUInst(CD:=%IX0.2, LOAD:=%IX0.3,               PV:=250) LD   CTDInst.CV ST   %MW12 LD   CTDInst.Q ST   %QX4.2</pre>	
ST	<pre>VAR   CTDInst: CTD   END_VAR CTDInst(CD:=%IX0.2, LOAD:=%IX0.3, PV:=250); %QX4.2:= CTDInst.Q; %MW12:= CTDInst.CV;</pre>

CD : BOOL      Counter Down : positive Flanke -> CV := CV-1 mit CV >=0

PV : WORD      Anfangswert

CV : WORD      Counter Value = Zählerstand

LOAD : BOOL    TRUE -> CV := PV

Q : BOOL        TRUE, falls CV = 0

Quelle: Wellenreuther

# ZÄHLER: CTUD AUF- UND ABWÄRTSZÄHLER

AWL	FUP
<pre> VAR   CTUDInst: CTUD; END_VAR CAL CTUDInst(CU:=%IX0.4, CD:=%IX0.5,               RESET:=%IX0.6, LOAD:=%IX0.7,               PV:=250) LD   CTUDInst.CV ST   %MW14 LD   CTUDInst.QU ST   %QX4.3 LD   CTUDInst.QD ST   %QX4.4 </pre>	<pre> CTUDInst   CTUD     CU      QU --&gt; %QX4.3     CD      QD --&gt; %QX4.4     RESET   CV --&gt; %MW14     LOAD     250     PV </pre>
ST	Quelle: Wellenreuther

```

VAR   CTUDInst: CTUD; END_VAR
CTUDInst(CU:=%IX0.4, CD:=%IX0.5, RESET:=%IX0.6, LOAD:=%IX0.7, PV:=250);
%QX4.3:= CTUDInst.QU; %QX4.4:= CTUDInst.QD; %MW14:= CTUDInst.CV;

```

CU : BOOL	Counter Up : pos. Flanke -> CV := CV+1	LOAD : BOOL	TRUE -> CV := PV
CD : BOOL	Counter Down : pos. Flanke -> CV := CV-1 mit CV >= 0	CV : WORD	Counter Value = Zählerstand
RESET : BOOL	Initialisierung: CV := 0	QU : BOOL	TRUE, falls CV >= PV
PV : WORD	Anfangswert / Vergleichswert	QD : BOOL	TRUE, falls CV = 0

# ZEITFUNKTIONEN BEISPIELAUFGABE: ZERKLEINERUNGSANLAGE

In einer Zerkleinerungsanlage für Steingut wird das zerkleinerte Material aus einer Mühle über ein Transportband in einen Wagen verladen.

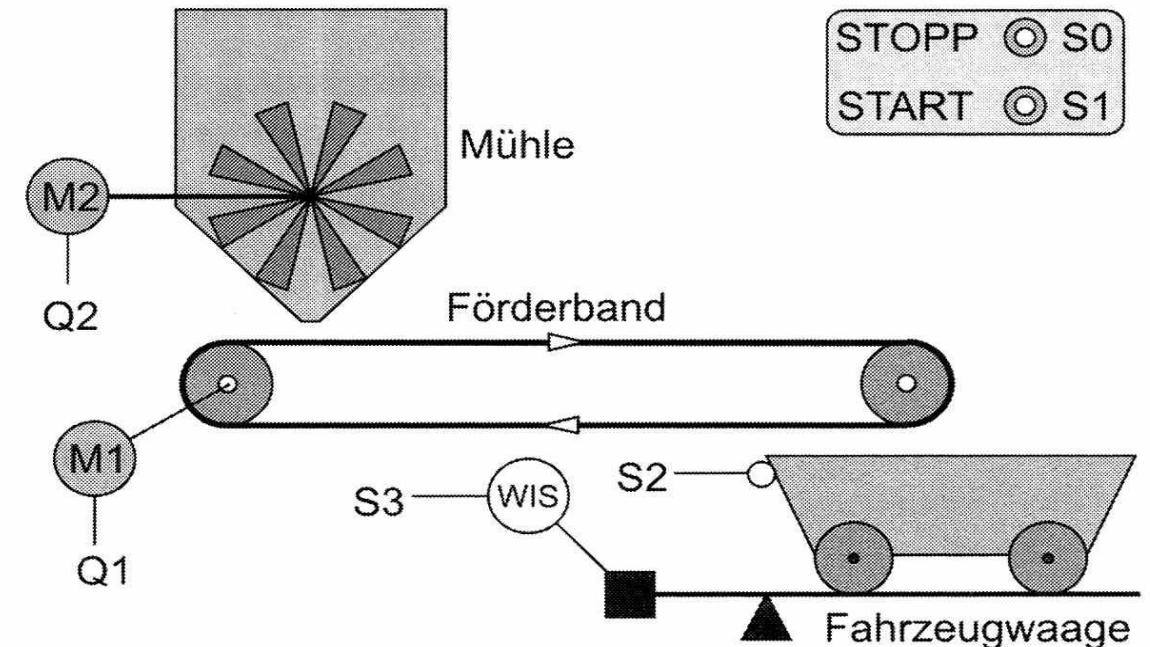
Der Abfüllvorgang für den Wagen kann durch Betätigen des **Tasters S1** gestartet werden, wenn ein Wagen an der Rampe steht ( $S2= „1“$ ). Um Stauungen des Fördergutes auf dem Transportband zu vermeiden, muss das Band **zwei Sekunden** laufen, bevor die Mühle mit **Q2** eingeschaltet wird. Meldet die Waage mit **S3=0**, dass der Wagen gefüllt ist, wird die Mühle ausgeschaltet. Das Förderband läuft noch drei Sekunden nach, um das Steingut vollständig vom Band zu entfernen.

Durch Betätigen des Stopp-Tasters **S0** wird der Abfüllvorgang sofort unterbrochen und das Förderband abgeschaltet.

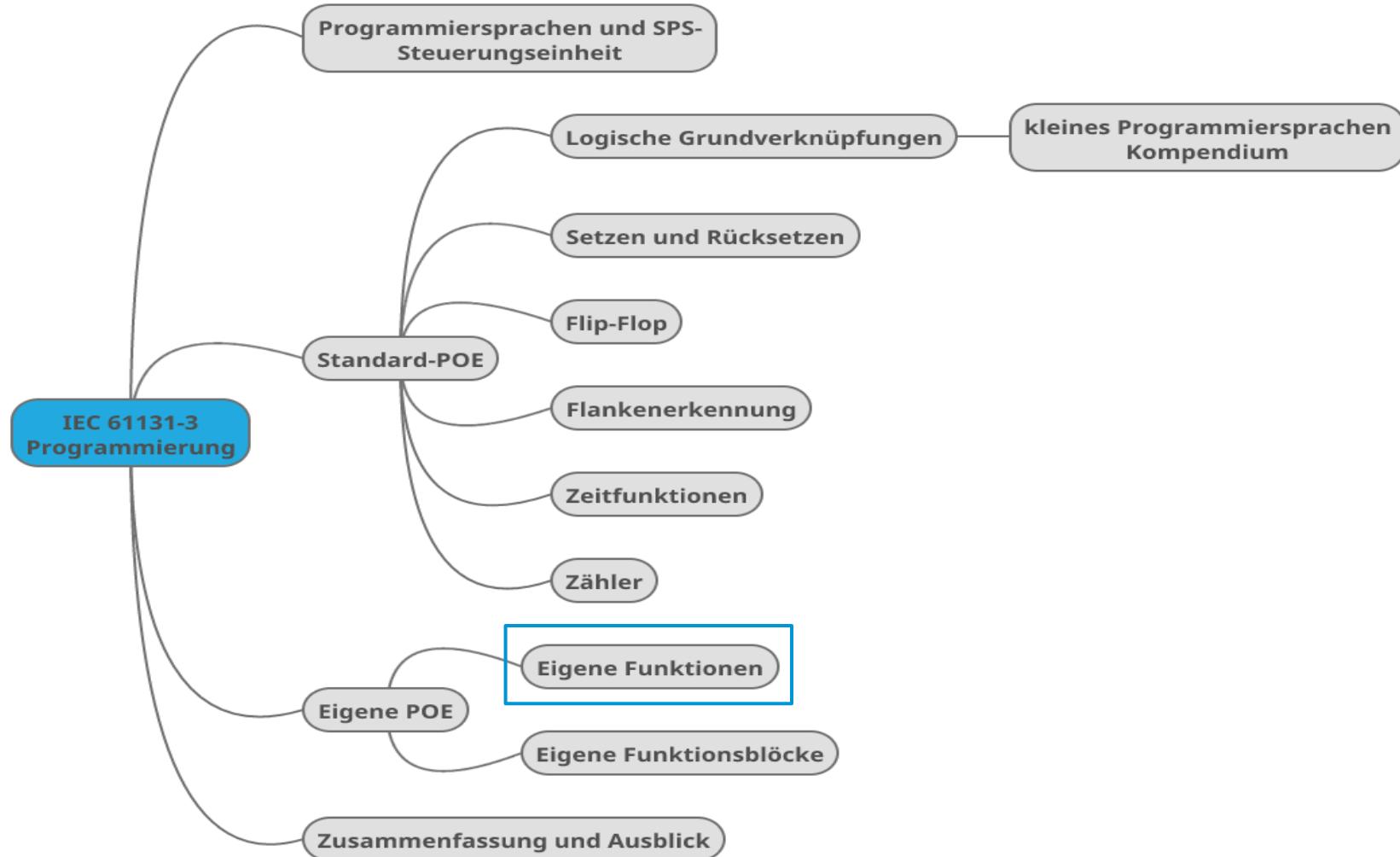
## Aufgabe:

- Stellen Sie einen vollständigen Befüllvorgang mit den Größen **S1**, **S2**, **S3**, **Q1**, **Q2** in einem Zeitdiagramm dar.
- Schreiben Sie ein Programm in FUP ~~und ST~~, dass die oben beschriebene Aufgabe erfüllt.
- Erweitern Sie Ihr Programm um einen Zähler, der die Anzahl der gefüllten Wagen bis zum Ausschalten zählt.

## Technologieschema:

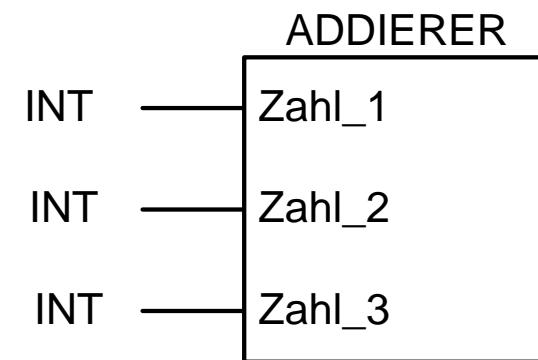


# KAPITEL 5: EIGENE FUNKTIONEN



# FUNKTIONEN FC (FUNCTION CODE) - ÜBERSICHT

- Parametrierbarer Codebaustein:
  - Formalparameter im Deklarationsteil deklariert
  - Formalparameter durch Aktualparameter beim Aufruf ersetzt.
- Prinzip: Gleiche Eingangswerte liefern immer denselben Funktionswert (und dieselben Ausgangswerte) : „seiteneffektfrei“
- Genau ein Funktionswert / Rückgabewert -> AE  
Beliebiger Datentyp, auch abgeleiteter zulässig.
- Speichert keine Informationen = „Ohne Gedächtnis“
- Deklarationsbereich:
  - VAR\_INPUT ... END\_VAR Eingangsvariable (Formalparameter)
  - VAR ... END\_VAR Lokale Variable
- Keine Rekursion, Kein Aufruf von FBs erlaubt
- Keine Verwendung von Globalen Variablen in FC erlaubt
- Keine Deklaration von physikalischen Speicheradressen erlaubt
- Standard- und Anwenderfunktionen



# FUNKTION FC – BEISPIEL ADDIERER FÜR 3 INT

Realisierung:

1	FUNCTION ADDIERER : INT
2	VAR_INPUT
3	(* Formalparameter *)
4	Zahl_1, Zahl_2, Zahl_3 : INT;
5	END_VAR
6	VAR
7	END_VAR

1	LD	Zahl_1
	ADD	Zahl_2
	ADD	Zahl_3
	ST	ADDIERER

Aufruf in ST:

Nur Aktualparameter:

Erg:=ADDIERER (111,222,333);

Mit Formalparametern

Erg:=ADDIERER (Zahl\_1:=111, Zahl\_2:=222, Zahl\_3:=333);

Aufruf in FUP:

TBA

# FUNKTION FC BEISPIELAUFGABE: TEMPERATURUMRECHNUNG

Ein Sensor misst die Temperatur in einem Gewächshaus.

Dieser Wert steht Ihnen in Fahrenheit in der Variable rTempPanelF zur Verfügung.

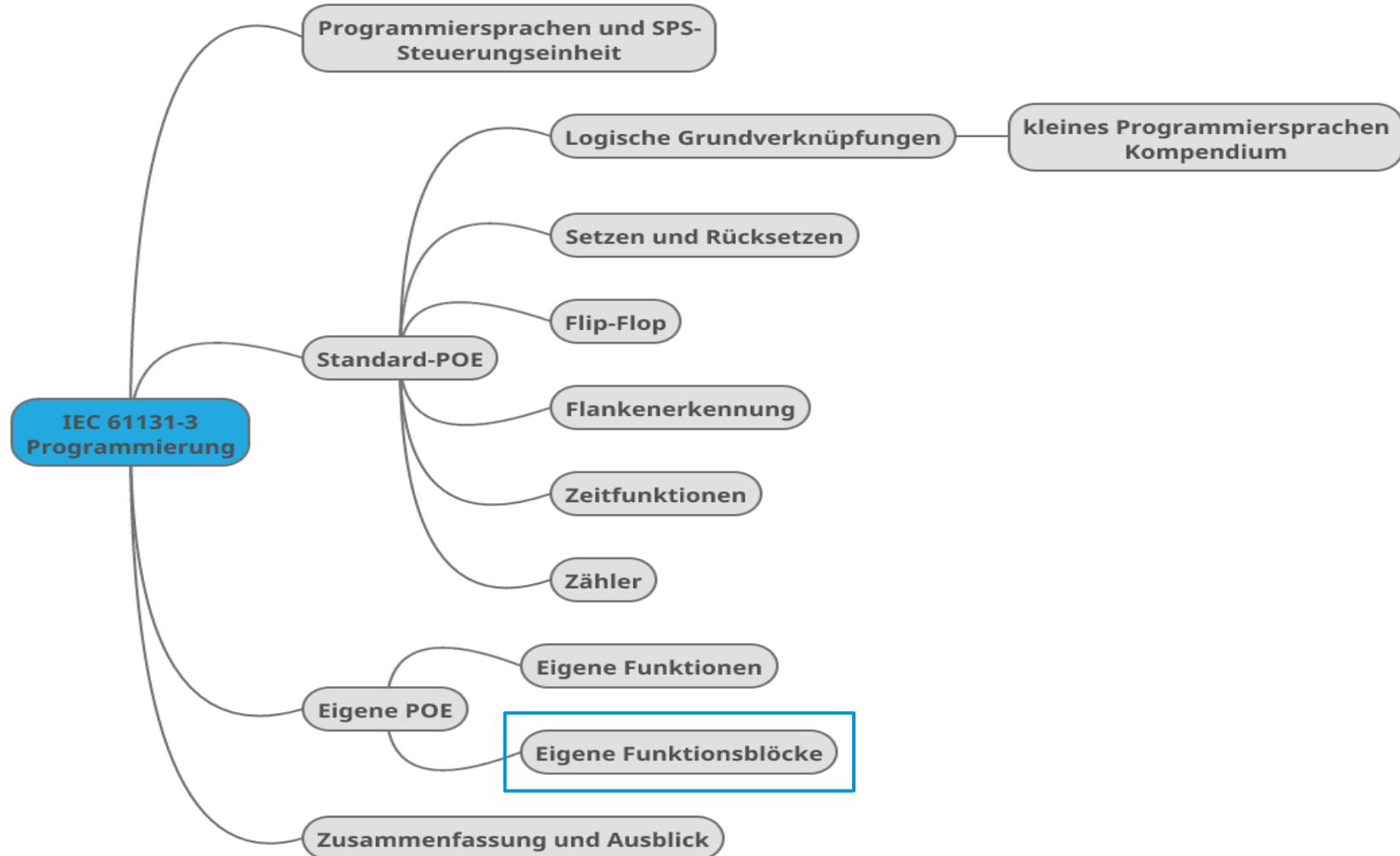
Für eine interne Temperaturregelung benötigen Sie die Gewächshaustemperatur als Ist-Größe in °C

## Aufgabe:

Schreiben Sie eine Funktion, welche die Temperatur von F in °C umrechnet

$$T_{inC} = (T_{inF} - 32) \cdot 5 : 9$$

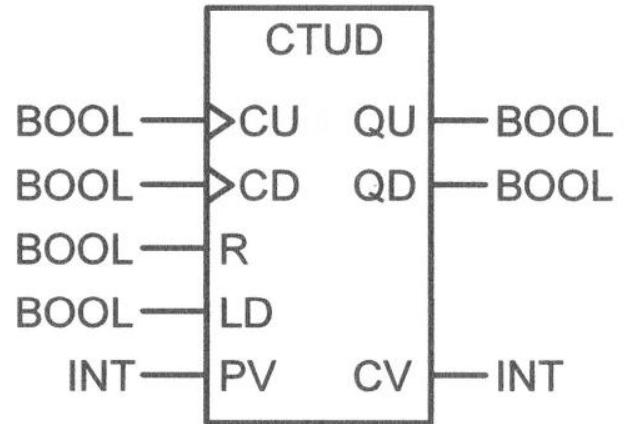
# KAPITEL 5: EIGENE FUNKTIONSBAUSTEINE



# FUNKTIONSBAUSTEIN FB - ÜBERSICHT

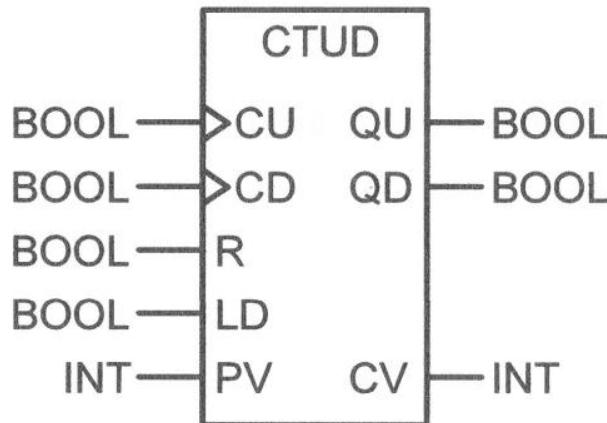
- **Parametrierbarer** Codebaustein:
  - **Formalparameter** im Deklarationsteil deklariert
  - Formalparameter durch **Aktualparameter** beim Aufruf ersetzt.
- FB ist eine eigenständige, nach außen gekapselte Datenstruktur mit einer auf ihr definierten Berechnungsvorschrift
- **Speichert Informationen = „Mit Gedächtnis“**
- **Instanzierung von FBs (objektorientierter Gedanke):**  
Gleicher Programmcode + Getrennte Datenbereiche
- Deklarationsbereich:

VAR_INPUT, VAR_OUTPUT, VAR_IN_OUT	Formalparameter
VAR_EXTERNAL	Zugriff auf globale Variablen
VAR	Lokale Variable – statisch!
- FBs können FBs und FCs aufrufen
- Keine Deklaration von physikalischen Speicheradressen erlaubt
- Standard- und Anwenderfunktionsbausteine



# FUNKTIONSBAUSTEIN FB – BEISPIEL CTUD

## Grafische Darstellung mit Aufrufsschnittstelle



(Beispiel-) Codierung des FBS CTUD

FUNCTION_BLOCK	CTUD	(* Vorwärts/Rückwärts-Zähler *)
VAR_INPUT		
CU : BOOL	R_EDGE;	(* CU mit steigender Flanke *)
CD : BOOL	R_EDGE;	(* CD mit steigender Flanke *)
R : BOOL;		
LD : BOOL;		
PV : INT;		
END_VAR		
VAR_OUTPUT		
QU : BOOL;		
QD : BOOL;		
CV : INT;		
END_VAR		
IF R THEN		(* Zähler vorrangig rücksetzen *)
CV := 0;		
ELSIF LD THEN		
CV := PV;		(* auf Zählerwert setzen *)
ELSE		
IF NOT (CU AND CD) THEN		
IF CU AND ( CV < PV) THEN		
CV := CV + 1; (* hochzählen *)		
ELSIF CD AND ( CV > PV) THEN		
CV := CV - 1; (* runterzählen *)		
ENDIF;		
ENDIF;		
ENDIF		
QU := (CV >= PV);		(* Grenzwert erreicht *)
QD := (CV <= 0);		(* Null erreicht *)
END_FUNCTION_BLOCK		

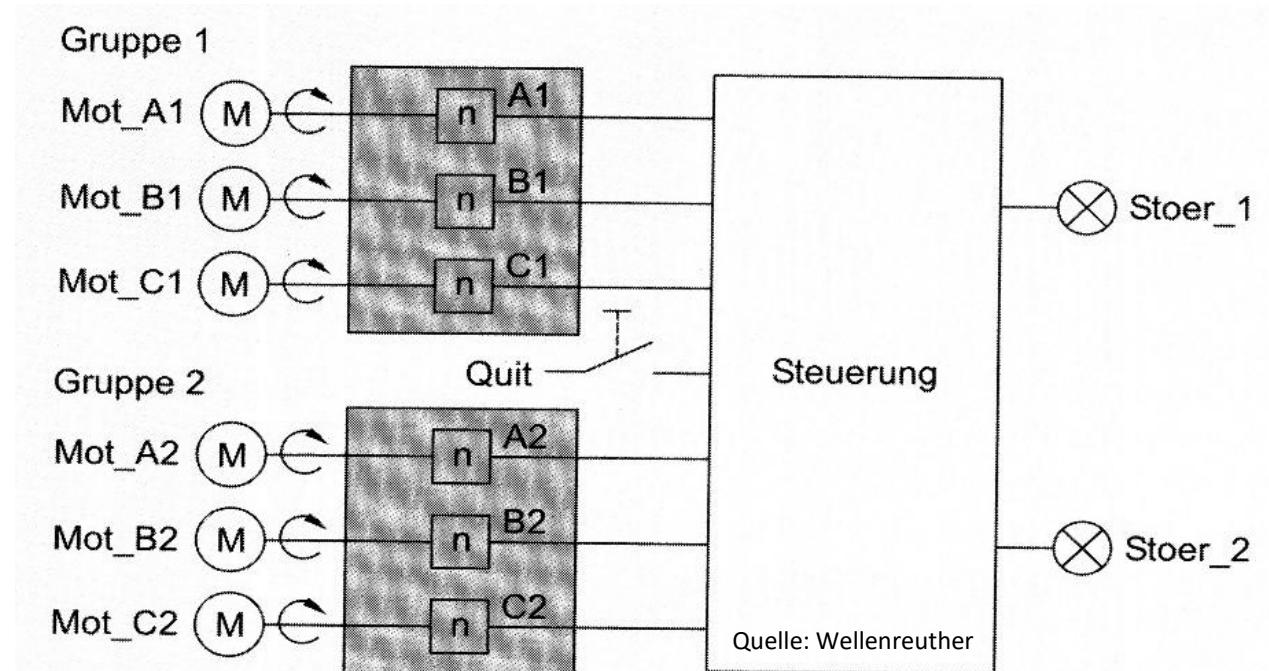
Quelle: John/Tiegelkamp

# FUNKTIONSBAUSTEIN FB BEISPIELAUFGABE - MOTORÜBERWACHUNG

Zwei Motorgruppen mit den Motoren A,B,C sollen überwacht werden. Jeder Motor ist mit einem Drehzahlwächter ausgerüstet, der mit 1- oder 0-Signal meldet, ob der Motor dreht oder nicht.

Die Störungsanzeige jeder Motorenguppe soll in folgenden Fällen aufleuchten:

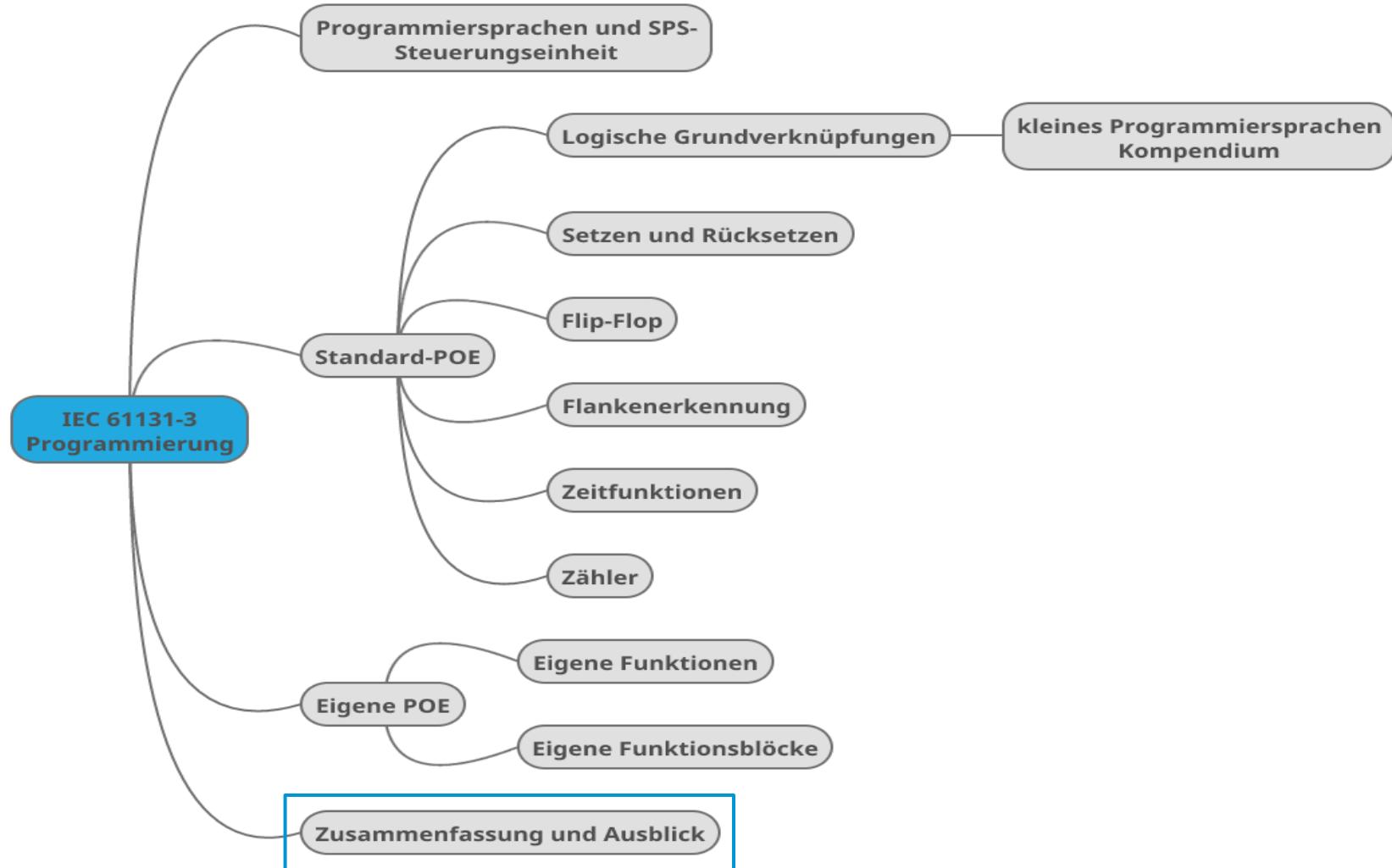
1. Fall: Wenn 2 der 3 Motoren länger als 5 Sekunden ausgefallen sind
2. Fall: Wenn alle 3 Motoren ausgefallen sind (sofortige Meldung)



Die Störungsanzeige soll selbstständig verlöschen, wenn die Störungsursachen behoben sind, also wenigsten 2 Motoren wieder laufen. Bei Ausfall aller 3 Motoren muss nach der Störungsbeseitigung noch zusätzlich die für beide Motorgruppe geltenden Quittierungstaste **Quit** betätigt werden, um die Störmeldung abzuschalten.

Für die Motorgruppenüberwachung ist ein Funktionsbaustein zu entwerfen, der allgemein einsetzbar ist!

# KAPITEL 5: EIGENE FUNKTIONSBAUSTEINE



# ZUSAMMENFASSUNG

- Umsetzung von Verknüpfungslogik in AWL, ST und FUP
- Verwendung von Standardbausteinen der Automatisierungstechnik (Domänenelemente)
  - Flip-Flop
  - Flankenerkennung
  - Timer
  - Counter
- Erstellung eigenen modularen und damit wiederverwendbaren POEs
  - Funktion
  - Funktionsblock