

# Industrial Automation

## Zusammenfassung

Joel von Rotz / [\(🔗 Quelldateien\)](#)

## Table of Contents

<b>Automatisierung</b>	<b>3</b>
Überblick - Maschinenautomatisierung	3
Überblick - Gebäudeautomatisierung	3
Automatisierungspyramide	3
Allgemeine	4
...Aufgaben	4
...Anforderungen	4
Bauformen	4
SPS	4
Prinzip	5
SPS vs Mikrocontroller	5
Numerische Steuerung	5
Zentralisiert / Feldbus	5
Spezifität	5
<b>Direkter/ Indirekter Reglerentwurf</b>	<b>6</b>
Analog vs Digitaler Regelkreis	6
Direkt vs Indirekt	6
Regelungsaufgabe	6
<b>Diskretisierung</b>	<b>7</b>
Laplace zu z-Bereich	7
Rückwärts-Rechteckregel	7
Trapezoidal-Regel (Tustin-Approx)	7
Vorwärts-Rechteckregel / Euler	7

Differenzengleichung	7
Prozesse	7
PT1 Prozess	7
PT2 Prozess	7
PID Regler	7
Proportionalität	7
Integration	8
Differential	8
D-Anteil gefiltert	8
Saturation	8
Antireset-Windup (ARW)	8
Ohne ARW	8
Mit ARW	8
<b>TwinCAT</b>	<b>8</b>
Datentypen	8
EN 61131-3 (IEC 1131)	9
AWL (IL): Anweisungsliste	9
KOP (LD): Kontaktplan	9
FBS (FBD): Funktionsbaustein-Sprache	9
ST: Strukturierter Text	9
AS (SFC): Ablaufsprache	9
Syntax	10
Deklaration & Implementation	10
<> IF-ELSE	10
<> FOR	10
<> CASE	10
<> WHILE	11
<> REPEAT	11
<> RETURN	11
<> JMP	11

<> EXIT .....	11
<> CONTINUE .....	11
<> Bitshifting SHR/SHL .....	11
⌚ Array .....	12
⌚ Structure .....	12
⌚ Enumeration .....	12
⌚ Global Variable List (GVL) .....	12
Hardware Verknüpfen / AT-Deklaration .....	13
Casting _TO_ .....	13
Programm .....	13
Funktion .....	13
Funktionsblock .....	13
TwinCat - Funktionsblöcke .....	14
⌚ Timer Off-Delay (TOF) .....	14
⌚ Timer On-delay (TON) .....	14
⌚ Timer Pulse Generator (TP) .....	14

<b>Beispiel Code</b> .....	<b>15</b>
<> Edge Detector <code>FB_EdgeDetector</code> .....	15
<> PWM Signal <code>F_PWMFunction</code> .....	15
<> Sinus Signal <code>F_SinusFunction</code> .....	15
<> PWM Signal <code>F_SquareFunction</code> .....	16
<> PT1 Prozess .....	16
<> PT2 Prozess .....	16
Struct Parameter .....	16
Prozess Implementation .....	16
<> PID Regler Prozess .....	17
SystemModus Enumeration .....	17
Struct Parameter .....	17

Regler Implementation .....	17
<b>Weise Seiten</b> .....	<b>19</b>
z-Transformation .....	19
Laplace Transformation .....	20

When the machine starts working again despite nobody making any changes...



All right, then. Keep your secrets.

**(i) Achtung, Achtung!**

Anstatt über die Fehler in der Zusammenfassung zu meckern, wäre ein *Pull Request* sehr töftel!

[Github Repo Link](#)

## Automatisierung

Automatisierung (DIN 19222): Das Ausrüsten einer Einrichtung, so dass sie ganz oder teilweise ohne Mitwirkung des Menschen geschieht.

### Maschinenautomatisierung

- Beispiele: Werkzeugmaschinen, Druckmaschinen, Textilmaschinen, Papiermaschinen, Biegemaschine, Wasserstrahlschneider, Industrieroboter (Stäubli), usw.

### Anlagenautomatisierung

- Beispiele: Fertigungsanlagen, Kraftwerke, Chemieanlagen, Postverteilanlagen, usw.

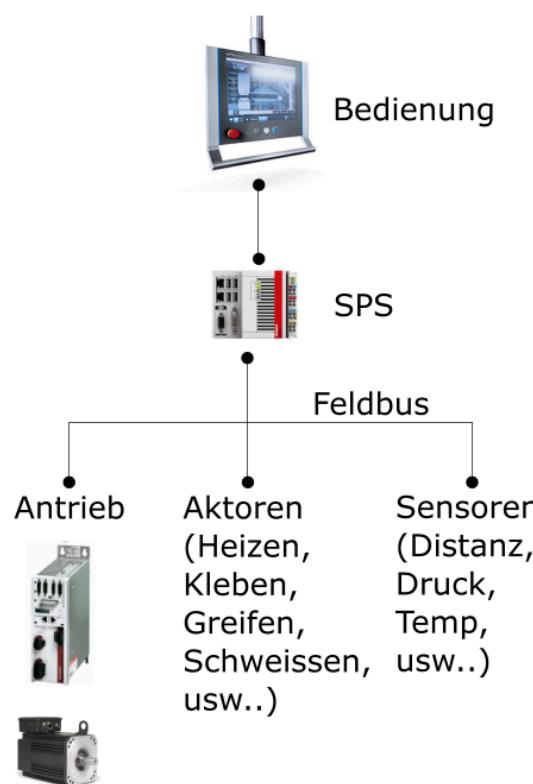
### Gebäudeautomatisierung

- Beispiele: Einfamilienhaus, Einkaufszentrum, Stadion, Bürogebäude, Hotel, usw.

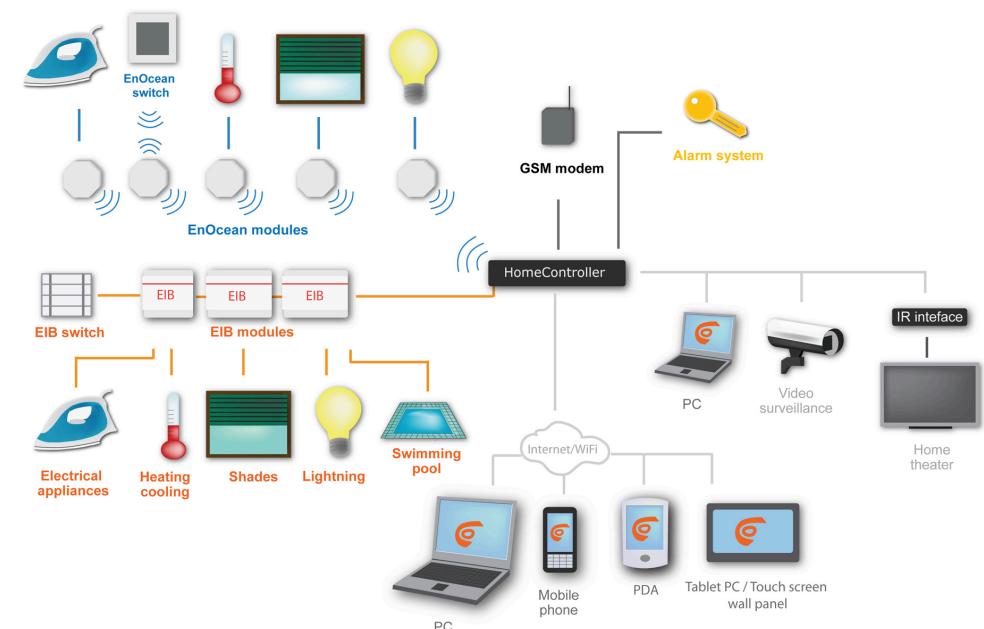
### Geräteautomatisierung

- Beispiele: Medizinische Geräte, Telefone, Computer, Haushaltsgeräte, usw.

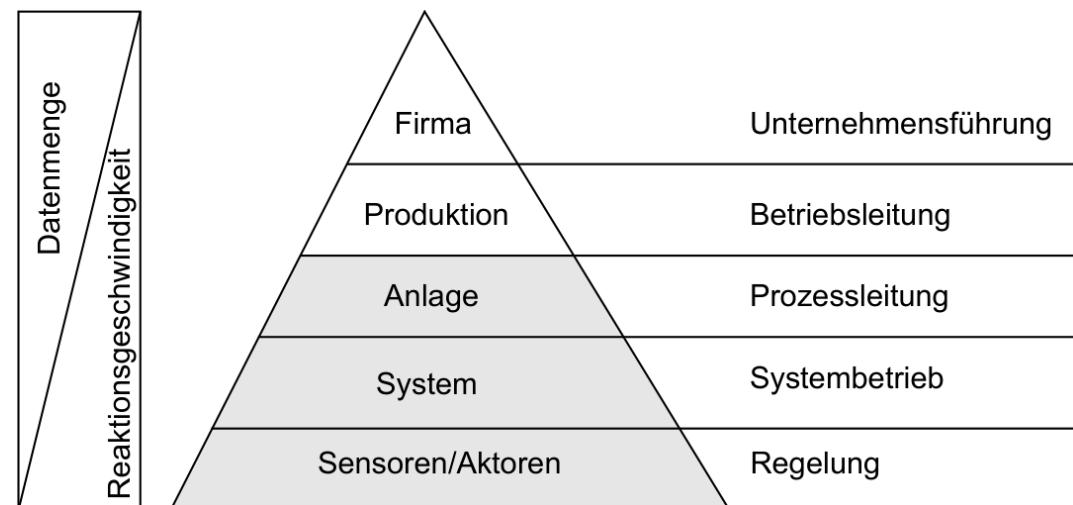
## Überblick - Maschinenautomatisierung



## Überblick - Gebäudeautomatisierung



## Automatisierungspyramide



**Allgemeine...****...Aufgaben**

Echtzeitanforderung

Regler  
Prozessabläufen  
usw.

keine  
Echtzeitanforderung

Mensch-Maschine-Interface (HMI)  
Produktionsplanung  
Archivierung  
usw.

**...Anforderungen**

Echtzeitfähigkeit

Betriebssysteme  
Speicherarchitektur

Ein- Ausgabe von  
Prozesssignalen

Aktoren  
Sensoren  
Kommunikationssysteme

Sicherheit  
Zuverlässigkeit

Hochwertige Komponente  
Redundanz

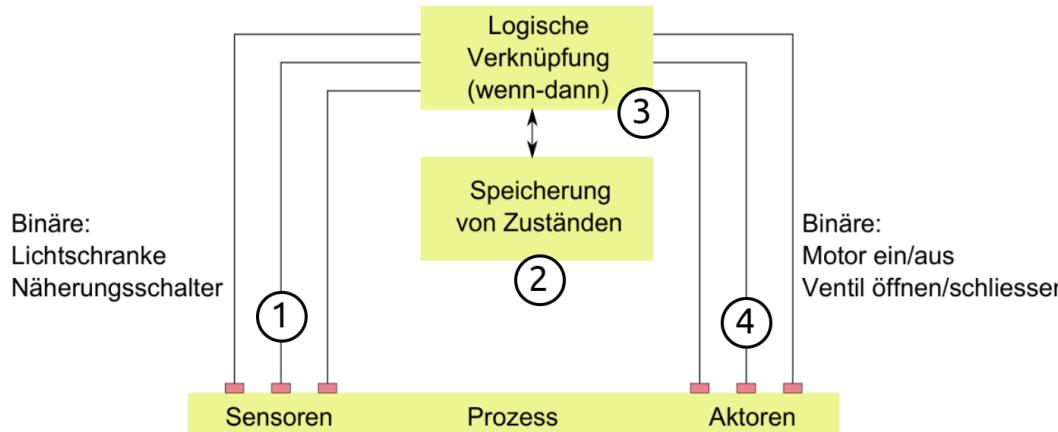
Resistenz gegen  
Umwelteinflüsse

Spezielle Gehäuse  
Schaltschrankmontage

**Bauformen****SPS****SPS** Speicherprogrammierbare Steuerung**PLC** Programmable Logic Controller

→ Wenn-dann-Regel

## Prinzip



**Reihenfolge** ① Sensoren auslesen ; ② Werte Zwischenspeichern ; ③ Logische Verknüpfungen anwenden ; ④ Aktoren setzen

## SPS vs Mikrocontroller

- In Automation möchte man etwas möglichst standardisiertes anwenden, anstatt eine Eigenlösung zu brauchen.
  - Aufwand ist kleiner
  - Die wichtigsten Funktionen sind bereits auf dem SPS implementiert
- Modularität
  - Die SPS ist erweiterbar durch Module (z.B. Motor-Modul, Encoder-Modul)
- It's also a bit like asking why would someone buy a PC when they could build their own.* [\[Link\]](#)

## Numerische Steuerung

Als Numerische Steuerung bezeichnet man ein Gerät zur Steuerung von Maschinen, das Steuerbefehle liest, die als Code auf einem Datenträger vorliegen, und in Arbeits- bzw. Bewegungsabläufe umsetzt. [Wiki](#)

Lustige Biegemaschine →

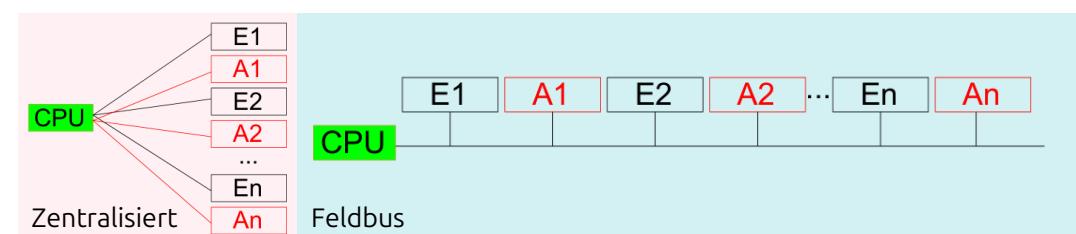


- Regelungsaufgabe
- Steuerungsaufgabe
- Importieren von CAD Dateien (IGES, DXF, usw.)
- HMI (Human Machine Interface)
- Prozessabläufe (SPS like)

**Beispiele:** Koordinaten-basierte Maschinen wie Laser Cutter oder CNC Fräsmaschinen, Industrie-PCs

## Zentralisiert / Feldbus

Ein Feldbus verbindet in einer Anlage Feldgeräte wie Messfühler (Sensoren) und Stellglieder (Aktoren) zwecks Kommunikation mit einem Steuerungsgerät



### ⊕ Vorteile Feldbus

- geringerer Verkabelungsaufwand
- Erweiterungen oder Änderungen

### ⊗ Nachteile Feldbus

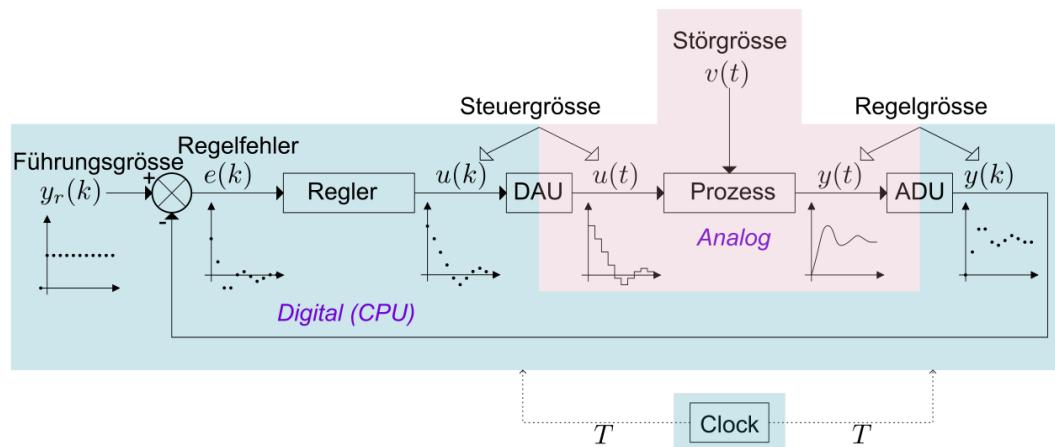
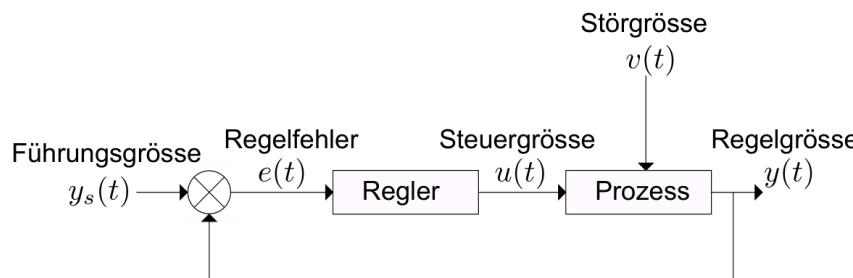
- Komplexität
- Preis
- Aufwendige Messgeräte (analyzer)
- Längere Reaktionszeit

## Spezifität

- Gebäude:** LON, EIB, usw.
- Anlage/Maschine:** CAN/CANOpen, Profibus, Profinet, EtherCAT, Varan, Ethernet/IP, SERCOS, usw.

## Direkter/Indirekter Reglerentwurf

### Analog vs Digitaler Regelkreis



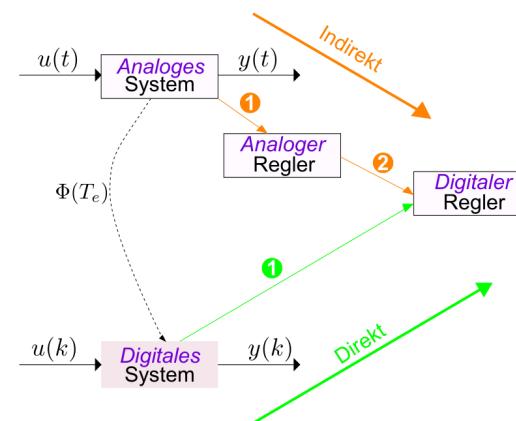
### ✓ Vorteile Regelkreis

- Reglerparameter können per Software geändert werden
- platzsparend und kostengünstig
- nicht nur PID, können auch andere (komplexe) Regler implementiert werden
- Rechner kann noch andere Aufgaben übernehmen

### ✗ Nachteile Regelkreis

- Diskretisieren ist nicht vernachlässigbar im Design
- Minderung der dynamischen Leistung
- Konzept weniger intuitiv als in der analogen Welt

## Direkt vs Indirekt



### ✓ Vorteile Indirekt

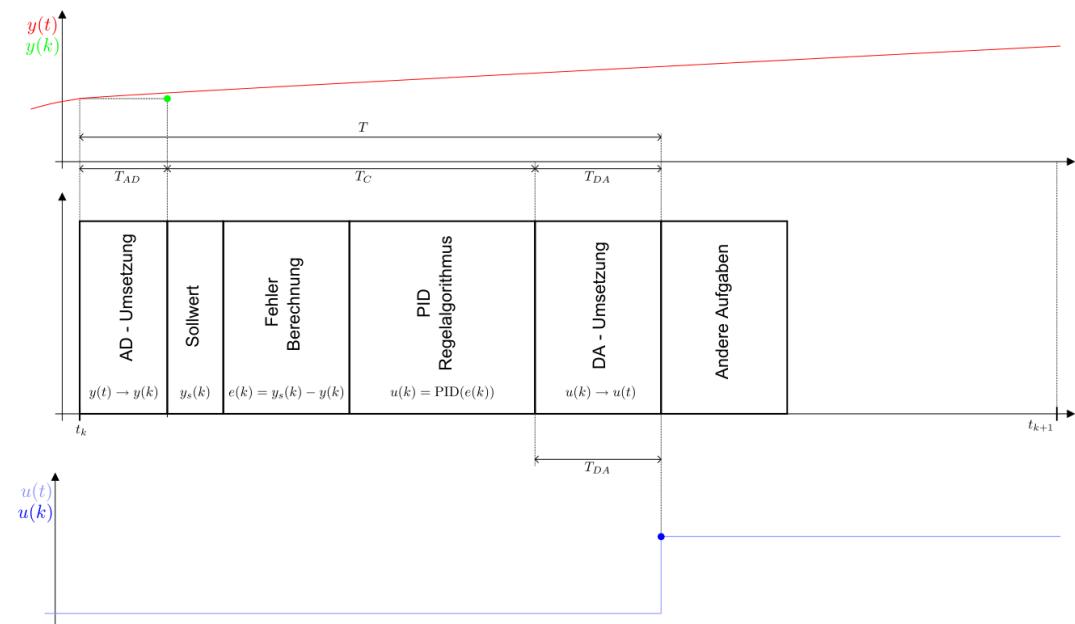
- Keine neuen Methoden notwendig
- Intuitivität, Interpretation

### ✗ Nachteile Indirekt

- Auf analogen Regler beschränkt (PID)
- Keine explizite Berücksichtigung der Diskretisierung

## Reglungsaufgabe

Ein wichtiger Aspekt eines digitalen Reglers ist die Dauer einer Reglersequenz: AD Umsetzung, PID Regelalgorithmus, DA Umwandlung. Das alles benötigt Zeit!



$$(T_{AD} + T_C + T_{DA}) \ll T$$

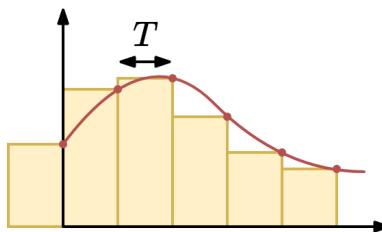
## Diskretisierung

### Laplace zu z-Bereich

Der I-Anteil eines Reglers kann anhand drei Varianten berechnet werden: Rückwärts-Rechteckregel, Vorwärts-Rechteckregel und die Regel

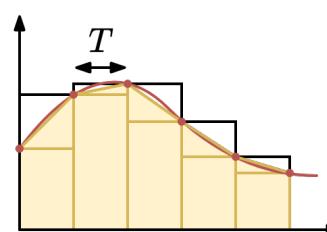
#### Rückwärts-Rechteckregel

$$\int_0^T e[\tau] d\tau \approx \sum_{i=0}^T e[k] \cdot T \quad s = \frac{z-1}{Tz}$$



#### Trapezoidal-Regel (Tustin-Approx)

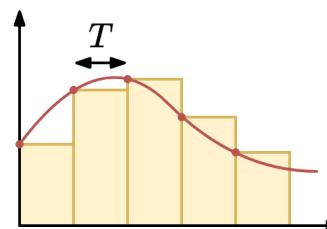
$$\int_0^T e[\tau] d\tau \approx \sum_{i=0}^T \frac{e[k] + e[k-1]}{2} \cdot T \quad s = \frac{2}{T} \cdot \frac{z-1}{z+1}$$



#### Vorwärts-Rechteckregel / Euler

$$s = \frac{z-1}{T} \rightarrow y[k] = \frac{1}{T} \cdot (x[k+1] - x[k])$$

Wird eher weniger angewendet, da „in die Zukunft blicken“ schwer implementierbar ist.



### Differenzengleichung

$$H(z) = \frac{Y(z)}{X(z)} = \frac{U(z)}{E(z)} = \frac{\sum_{m=0}^M b_m \cdot z^{-m}}{\sum_{n=0}^N a_n \cdot z^{-n}}$$

### Prozesse

#### PT1 Prozess

$$G(s) = \frac{Y(s)}{U(s)} = \frac{1}{1 + \tau \cdot s} \rightarrow s = \frac{z-1}{Tz} \rightarrow y[k] = (1 + T/\tau) \cdot u[k] + (1 + \tau/T) \cdot y[k-1]$$

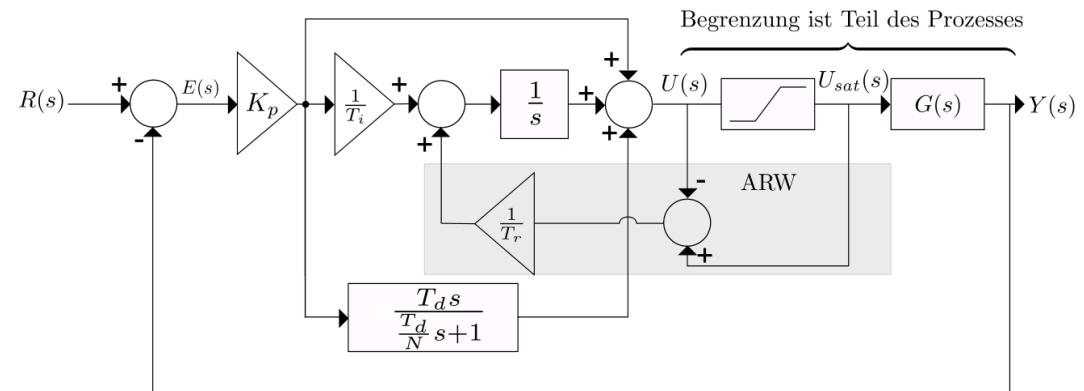
#### PT2 Prozess

$$G(s) = \frac{Y(s)}{U(s)} = \frac{K}{(1 + \tau_1 s)(1 + \tau_2 s)} \rightarrow s = \frac{z-1}{Tz} \rightarrow H(z) = \frac{b_0}{1 + a_1 \cdot z^{-1} + a_2 \cdot z^{-2}}$$

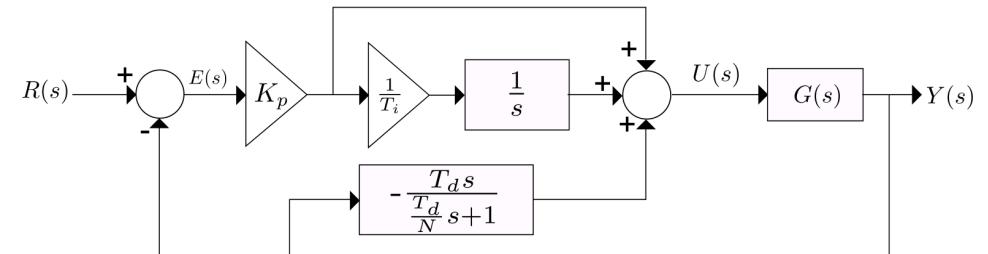
$$\alpha_1 = T + \tau_1 \quad ; \quad \alpha_2 = T + \tau_2 \quad ; \quad b_0 = \frac{K \cdot T^2}{\alpha_1 \alpha_2} \quad a_1 = -\left(\frac{\tau_1}{\alpha_1} + \frac{\tau_2}{\alpha_2}\right) \quad ; \quad a_2 = \frac{\tau_1 \tau_2}{\alpha_1 \alpha_2}$$

$$y[k] = -a_1 \cdot y[k-1] - a_2 \cdot y[k-2] + b_0 \cdot u[k]$$

#### PID Regler



$$u(t) = u_{k,a(e(t))} + u_{i,a(e(t))} + u_{d,a(e(t))} = K_P \left( e(t) + \frac{1}{T_{i,a}} \int_0^t e(\tau) d\tau + T_{d,a} \cdot \dot{e}(t) \right)$$



#### Proportionalität

$$u_p[k] = K_P \cdot e[k]$$

$K_P$  ist equivalent zur analogen Version  $K_a$

## Integration

$$\sum_0^k = \sum_0^{k-1} + \alpha(k) \Rightarrow u_{i,d}[k] = u_i[k-1] + \frac{K_P}{T_i} \cdot \frac{e[k] + e[k-1]}{2} \cdot T \leftarrow \text{Trapez}$$

- $T_{i,d}$  ist equivalent zur analogen Version  $T_{i,a}$

$$u_{i,d}[k] = u_i[k-1] + \frac{K_P}{T_{i,d}} \cdot e[k] \cdot T \leftarrow \text{Rückwärts}$$

## Differential

$$u_{d,d}[k] = \frac{e[k] - e[k-1]}{T} \cdot K_P \cdot T_{d,d}$$

$T_{d,d}$  ist equivalent zur analogen Version  $T_{d,a}$

## D-Anteil gefiltert

Je kleiner die Abtastrate, umso grösser sind die Overshoots des D-Anteils. Durch das Quantisieren des AD-Wandlers entsteht ein zusätzlicher Fehler (Bit „Flackern“, wenn die gemessene Spannung zwischen zwei Bit-Werten ist).

Um diese Overshoots zu korrigieren, wird ein Tiefpass-Filter zum D-Anteil hinzugefügt.

$$u_d = K_P \cdot T_d \cdot s \cdot \frac{1}{\frac{T_d}{N} \cdot s + 1} \rightarrow s = \frac{1 - z^{-1}}{T}$$

$$u_d[k] = K_P \cdot \frac{T_d}{T + T_F} \cdot (e[k] - e[k-1]) + \frac{T_F}{T + T_F} \cdot u[k-1] \quad \text{mit } T_F = \frac{T_d}{N}$$

## Saturation

Die Stellgrößen für den Prozess haben (immer) einen begrenzten Bereich (z.B.  $\pm 24V$  für einen Motor).

$$u[k] = \begin{cases} u_{\text{sat},\text{max}} & \text{if } u_{\text{nosat}}[k] > u_{\text{sat},\text{max}} \\ u_{\text{sat},\text{min}} & \text{if } u_{\text{nosat}}[k] < u_{\text{sat},\text{min}} \\ u_{\text{nosat}}[k] & \text{else} \end{cases}$$

Da mit der Saturation der I-Anteil kontinuierlich einen Fehler aufaddieren kann (bis ein Overflow oder andere Probleme entstehen), wird dies folgend noch mit einem Antireset-Windup gelöst!

## Antireset-Windup (ARW)

Der Wandel von Digital zu Analog führt zu Overshoots im  $u[]$  Anteil, da die Diskretisierung Tod-Zeiten einführt, wo der Regler nicht reagieren kann!

Mit dem ARW wird gegen den starken Einfluss des I-Anteils in die Stellgröße bei grossen Differenzen (welche zur Saturation führt) gewirkt.

### Ohne ARW

$$u_i[k] = u_i[k-1] + K_P \frac{T}{T_i} \cdot \frac{e[k] + e[k-1]}{2}$$

$$u_{\text{nosat}}[k] = u_p[k] + u_i[k] + u_d[k]$$

⇒ Durch die Saturation Funktion drücken!

### Mit ARW

$$u_i[k] = u_i[k-1] + K_P \frac{T}{T_i} \cdot \frac{e[k] + e[k-1]}{2}$$

$$+ \frac{T}{T_r} (u[k] - u_{\text{nosat}}[k])$$

$$u_{\text{nosat}}[k] = u_p[k] + u_i[k-1] + u_d[k]$$

## TwinCAT

### Datentypen

Type	Lower	Upper	Bits	Note
BOOL	0	1	8	>1 are invalid values
BYTE	0	255	8	
WORD	0	65535	16	
DWORD	0	4294967295	32	
SINT	-128	127	8	
USINT	0	255	8	
INT	-32768	32767	16	
UINT	0	65535	16	
DINT	-2147483648	2147483647	32	
UDINT	0	4294967295	32	
REAL	$-3.402823 \times 1038$	$3.402823 \times 1038$	64	
LREAL	$\sim -1.79769 \times 1038$	$\sim 1.79769 \times 1038$	64	
STRING	—	—	n+1	Appends \0
TIME	T#0ms	T#71582ms47s295ms	32	
TOD	TOD#00:00	TOD#1193:02:47.295	32	
DATE	D#1970-01-01	D#2106-02-06	32	
DT	DT#1970-01-01-00:00	DT#2106-02-06-06:28:15	32	

## EN 61131-3 (IEC 1131)

Ist die einzige weltweit gültige Norm für Programmiersprachen von speicherprogrammierbaren Steuerungen. Sie definiert die folgenden fünf Sprachen:

Englisch		Deutsch	
IL	Instruction List	AWL	Anweisungsliste
LD	Ladder Diagram	KOP	Kontaktplan
FBD	Function Block Diagram	FBS	Funktionsbaustein-Sprache
ST	Structured Text	ST	Strukturierter Text
SFC	Sequential Function Chart	AS	Ablausprache

### AWL (IL): Anweisungsliste

```

VAR
    EL_Takt : BOOL;
    Qn1_Zeit : BOOL;
    STO11 : BOOL;
    STO12 : BOOL;
    F01 : BOOL;
    F02 : BOOL;
END_VAR

VAR CONSTANT
END_VAR

VAR_INPUT
END_VAR

VAR_OUTPUT
END_VAR

(* Netzwerk 0 *)

```

```

LD      In_bit_0_1
ANDN   Out_bit_4_1
ANDN   Out_bit_5_3
S      Out_bit_4_1
LD      In_bit_0_3
R      Out_bit_4_1

```

```

LD      In_bit_0_2
ANDN   Out_bit_4_1
ANDN   Out_bit_5_3
S      Out_bit_4_2
LD      In_bit_0_4
R      Out_bit_4_2

```

```

LDN    Out_bit_4_1
OR(    true
AND    Out_bit_4_1
AND    Bi_Takt
)

```

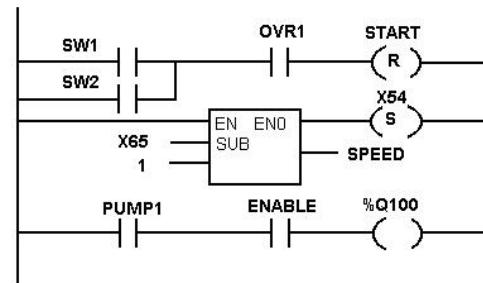
```

ST      Out_bit_4_3
LDN    Out_bit_4_2
OR(    true

```

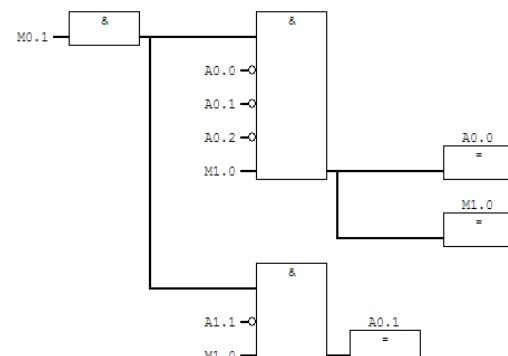
- Assembler für SPS
- Urmutter der SPS Programmierung
- Geeignet für einfachen sequentiellen Programme (keine Schleife)

### KOP (LD): Kontaktplan



- Gut für Ersatz für eine verdrahtete Logik
- Nicht geeignet für komplexe Programme

### FBS (FBD): Funktionsbaustein-Sprache



- Wird gerne bei Bit-Verknüpfungen verwendet
- Datenfluss übersichtlich

- Geeignet für einfachen sequentiellen Programme (keine Schleife)

### ST: Strukturierter Text

IF TRUE THEN

CASE Mode OF

- (\* einfaches rau mit Wiederholung \*)
 LLAWL(Len:=Len, BlinkBitsArray:=BlinkBitsArray);
- (\* rau und runter \*)
 LLFUP(Len:=Len, BlinkBitsArray:=BlinkBitsArray);
- (\* auf halber Laenge 2 bit rau und runter \*)
 LLKOP(Len:=Len, BlinkBitsArray:=BlinkBitsArray);
- (\* auf halber Laenge alle bit rau und runter \*)
 LLAS(LEN:=LEN, BlinkBitsArray:=BlinkBitsArray);

END\_CASE

IF Mode ↔ altMode

THEN

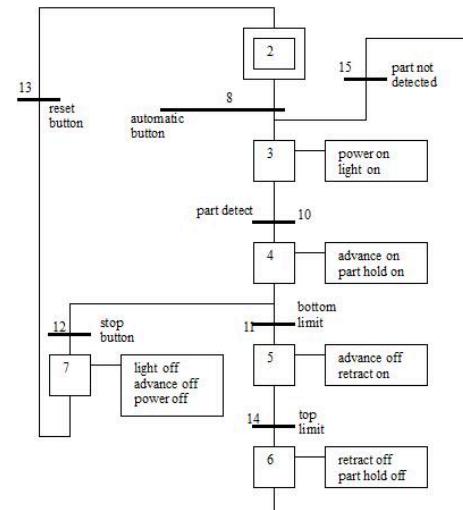
```

FOR i := 0 TO Len DO
    BlinkBitsArray[i] := FALSE;
END_FOR
i := 1;
END_IF
altMode := Mode;

(* copy to DWORD *)
dwOut := 16#00000002;
FOR count := 1 TO Len DO

```

### AS (SFC): Ablausprache



- Hochsprache (C, Pascal)

- Schleifenprogrammierung auch ohne Sprungbefehl möglich
- In Europa sehr oft gewählt

- Grafisch

- Zustandsautomaten
- Gut lesbar
- Geeignet für übergeordneter Zustandsabläufe

► **Beispiel:** Programm einer Operationsmanager-SPS wird mit AS geschrieben und die Arbeiter-SPS in ST

## Syntax .....

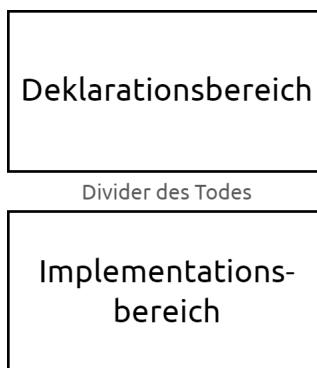
### Deklaration & Implementation

Die Umgebung eines Programmes, Funktionsblockes oder Funktion ist immer in zwei Bereichen aufgeteilt:

- Deklaration
- Implementation

In der Deklaration, werden alle Inputs, Outputs, Putputs deklariert und allenfalls direkt initialisiert. In der Implementation wird der eigentliche Code geschrieben.

Die Deklaration wird einmal zu Beginn ausgeführt, die Implementation wird im konfigurierten Zyklus wiederholt.



### Deklaration

Bei Programmen gibt es nur den `VAR` (& resp. `END_VAR`) Bereich.

```

1 VAR
2   <name> : <type> := <initial-value>
3   bTrig : BOOL := FALSE;
4   rValue : REAL := 0.0;
5 END_VAR

```

Bei Funktionsblöcken und Funktionen gibt es zusätzlich noch `VAR_INPUT`, `VAR_OUTPUT` und `VAR_IN_OUT`

```

1 VAR_INPUT
2   bTrig : BOOL := FALSE;
3 END_VAR
4
5 VAR_OUTPUT
6   bMotorEnable : BOOL := TRUE;
7 END_VAR
8
9 VAR_IN_OUT
10  sString : STRING := TRUE;
11 END_VAR

```

### Namenskonvention

Auch für die SPS gibst eine Namenskonvention, welche nicht unbedingt eingehalten werden muss. Variablennamen werden jeweils ein Kürzel des Datentyps vorangestellt. Konstanten werden komplett gross geschrieben (ausser das Kürzel).

Beispiel: `bTrigger` für eine `BOOL`-Variable

### <> IF-ELSE

```

1 IF <condition1> THEN
2   // Stuff 1
3 ELSIF <condition2> THEN
4   // Stuff 2
5 ELSE
6   // Other stuff
7 END_IF

```

### <> FOR

```

1 FOR nCounter := 1 TO 5 BY 1 DO
2   nVar1 := nVar1*2;
3 END_FOR;
4 nErg := nVar1;

```

### <> CASE

```

1 CASE nVar OF
2   1,5 :
3     bVar1 := TRUE;
4     bVar3 := FALSE;
5   2 :
6     bVar2 := FALSE;
7     bVar3 := TRUE;
8   10..20 :
9     bVar1 := TRUE;
10    bVar3 := TRUE;
11  ELSE
12    bVar1 := NOT bVar1;
13    bVar2 := bVar1 OR bVar2;
14 END_CASE;

```

Switch-Cases können auch für Enums verwendet werden:

```

1 CASE eSystemMode OF
2   ENUM_SystemModus.System_Init:
3     bVar1 := TRUE;
4     bVar2 := FALSE;
5   ENUM_SystemModus.System_Run:
6     bVar1 := FALSE;
7     bVar2 := TRUE;
8 ELSE // for unused enums
9   bVar1 := FALSE;
10  bVar2 := FALSE;

```

## <> WHILE

```

1 WHILE nCounter <> 0 DO
2   nVar1 := nVar1*2
3   nCounter := nCounter-1;
4 END_WHILE;

```

## <> REPEAT

Wird mindestens 1x ausgeführt.

```

1 REPEAT
2   nVar1 := nVar1*2;
3   nCounter := nCounter-1;
4 UNTIL
5   nCounter = 0
6 END_REPEAT;

```

## <> RETURN

```

1 IF bVar1 = TRUE THEN
2   RETURN;
3 END_IF;
4 nVar2 := nVar2 + 1;

```

Unterschied zu EXIT → verlässt die Funktion/der Funktionsblock komplett.

## <> JMP

Kannste den gleichen Fehler wie in C machen (`goto`)

```

1 nVar1 := 0;
2 _label1 : nVar1 := nVar1+1;
3 (*instructions*)
4 IF (nVar1 < 10) THEN
5   JMP _label1;
6 END_IF;

```

## <> EXIT

Mit `EXIT` können `FOR`, `REPEAT` und `WHILE`-Loops ausgebrochen werden (wie `break` in C).

## <> CONTINUE

Ähnlich wie `continue` in C.

```

1 FOR nCounter :=1 TO 5 BY 1 DO
2   nInt1:=nInt1/2;
3   IF nInt1=0 THEN
4     CONTINUE; (* to avoid a division by zero *)
5   END_IF
6   nVar1:=nVar1/nInt1; (* executed, if nInt1 is not 0 *)
7 END_FOR;
8 nRes:=nVar1;

```

## <> Bitshifting SHR/SHL

```

1 PROGRAM Shr_st// right shifting
2 VAR
3   nInByte : BYTE:=16#45; // 2#01000101
4   nInWord : WORD:=16#0045; // 2#000000001000101
5   nResByte : BYTE;
6   nResWord : WORD;
7   nVar : BYTE := 2; // amount of shifting
8 END_VAR
9
10 nResByte := SHR(nInByte,nVar); //Result is 16#11, 2#00010001
11 nResWord := SHR(nInWord,nVar); //Result is 16#0011, 2#0000000000010001

```

⇒ Nächste Seite für die Implementation ⇒

```

1 PROGRAM Shl_st // left shifting
2 VAR
3   nInByte : BYTE:=16#45; // 2#01000101
4   nInWord : WORD:=16#0045; // 2#0000000001000101
5   nResByte : BYTE;
6   nResWord : WORD;
7   nVar : BYTE := 2; // amount of shifting
8 END_VAR
9
10 nResByte := SHL(nInByte,nVar); // Result is 16#14, 2#00010100
11 nResWord := SHL(nInWord,nVar); // Result is 16#0114, 2#0000000100010100

```

## ⓘ Circular Shifting

Mit `ROL` und `ROR` können Bits rotiert werden (herausgeschobene werden am anderen Ende wieder hineingeschoben).

## ⓧ Array

```

1 <variable name> : ARRAY[ <dimension> ] OF <data type> ( := 
2   <initialization> )? ;
3 // (...)? : Optional
4
5 VAR
6   aCounter : ARRAY[0..9] OF INT;
7   // Short notation for [10, 10, 20, 20]
8   aCardGame : ARRAY[1..2, 3..4] OF INT := [2(10),2(20)];
9 END_VAR

```

## ⓧ Structure

```

1 TYPE <structure name> :
2 STRUCT
3   (<variable declaration optional with initialization>)+*
4 END_STRUCT
5 END_TYPE
6
7 TYPE ST_POLYGONLINE :
8 STRUCT
9   aStart : ARRAY[1..2] OF INT := [-99, -99];
10  aPoint1 : ARRAY[1..2] OF INT;
11  aPoint2 : ARRAY[1..2] OF INT;

```

```

12  aPoint3 : ARRAY[1..2] OF INT;
13  aPoint4 : ARRAY[1..2] OF INT;
14  aEnd : ARRAY[1..2] OF INT := [99, 99];
15 END_STRUCT
16 END_TYPE

```

## ⓧ Enumeration

```

1 {attribute 'strict'}
2 TYPE <enumeration name>:
3 (
4   <component declaration>,
5   <component declaration>
6 ) <basic data type> := <default variable initialization>
7 ;
8 END_TYPE
9
10 {attribute 'qualified_only'}
11 {attribute 'strict'}
12 TYPE E_ColorBasic :
13 (
14   eRed, // = 0 (default, if not defined)
15   eYellow, // = 1
16   eGreen := 10,
17   eBlue, // = 11
18   eBlack // = 12
19 ) := eYellow // Basic data type is INT, default initialization for all
20 E_ColorBasic variables is eYellow
21 ;
21 END_TYPE

```

## ⓧ Global Variable List (GVL)

```

1 {attribute 'qualified_only'}
2 VAR_GLOBAL
3   uGlobalEncoderCount AT %I* : UINT;
4   bGlobalEncoderOverflow AT %I* : BOOL;
5   bGlobalEncoderUnderflow AT %I* : BOOL;
6   bGlobalMotorEnable AT %Q* : BOOL;
7   iGlobalMotorVelocity AT %Q* : INT;
8 END_VAR

```

## Hardware Verknüpfen / AT-Deklaration

Da die SPS ein Gerät ist, welches mit anderer Hardware interagiert, können deren Pins mit Variablen verknüpft werden. Dies wird mit dem **AT** Keyword gemacht:

```
1 <identifier> AT <address> : <data type>;
2 // _____
3 %<memory area prefix> ( <size prefix> )? <memory position>
```

```
1 <memory area prefix> : I | Q | M
2 <size prefix> : X | B | W | D
3 <memory position> : * | <number> ( .<number> )*
```

- I**: Input (*sensors*); **Q**: Output (*actuator*) ; **M**: Memory/Speicher
- X**: Single Bit ; **B**: Byte (8 bits) ; **W**: Word (16 bits) ; **D**: Double Word (32 bits)

Beispiele:

```
1 IbSensor1 AT %I* : BOOL;
2 IbSensor2 AT %IX7.5 : BOOL;
```

## Casting \_TO\_

TwinCat ST meckert, wenn man teils-ungültige Datentypen kombiniert (z.B. Zuweisung von Typ INT nach Typ DINT). Um dies zu lösen, können Casting-Funktionen verwendet werden:

```
1 dstValue := <src-type>_TO_<dst-type>(srcValue)
2
3 iEncCntDifference := UINT_TO_DINT(uEncNewCnt) - UINT_TO_DINT(uEncOldCnt);
4 iMotorSpeed := REAL_TO_INT(32767.0 * rU/24.0);
```

## Programm

- Können als Funktion gestartet werden: z.B. **ProgrammXYZ()**;
- Können keinen Rückgabewert haben

```
1 PROGRAM Spaghetti_Sauce
2 VAR
3 // declare Stuff here
4 bTrig : BOOL := FALSE;
5 rValue : REAL := 0.0;
6 END_VAR
```

⇒ Nächster Abschnitt ist die Implementation ⇒

```
1 // do stuff in the main program
2 IF (bTrig = FALSE) AND (rValue ≥ -1.0) THEN // also just 'NOT bTrig' works
3   rValue := 2.0;
4 END_IF
5
6
7 P_Program(); // call other programs
```

## Funktion

- Kann ohne Instanzierung verwendet
- Keine statische Werte / Merkt sich Werte nicht!
- Return-Wert wird durch zuweisen eines Wertes auf den Funktionsnamen gemacht

```
1 FUNCTION F_CalculateVelocity : REAL
2 VAR_INPUT
3   uEncoderNewCount : UINT;
4   uEncoderOldCount : UINT;
5   rT : real;
6 END_VAR
7 VAR
8   iEncoderCountDifference : DINT;
9 END_VAR
```

```
1 iEncoderCountDifference := UINT_TO_DINT(uEncoderNewCount)
2           - UINT_TO_DINT(uEncoderOldCount);
3 IF (iEncoderCountDifference < 0) AND (GVL_Hardware.bGlobalEncoderOverflow) THEN
4   // (new + max) - old
5   iEncoderCountDifference := DINT_TO_INT(
6     (UINT_TO_DINT(uEncoderNewCount) + 65535)
7     - UINT_TO_DINT(uEncoderOldCount));
8 END_IF
9 // calculate rotation speed: 1/rT * 60 converts task cycle to rpm
10 F_CalculateVelocity := DINT_TO_REAL(iEncoderCountDifference)/ 2048.0 * (1/rT *
60);
```

## Funktionsblock

- Muss im Deklarationsbereich zuerst instanziert werden: **fbCounter : FB\_Counter**;
- FB\_Counter()**; geht nicht
- Interne Variablen werden zwischengespeichert.

Funktionsblöcke können wie gefolgt aufgerufen werden:

```

1 fbTMR : TON;
2 fbTMR (IN := %OX5, PT := T#300ms);
3 bVarA := fbTMR.Q;

```

oder

```

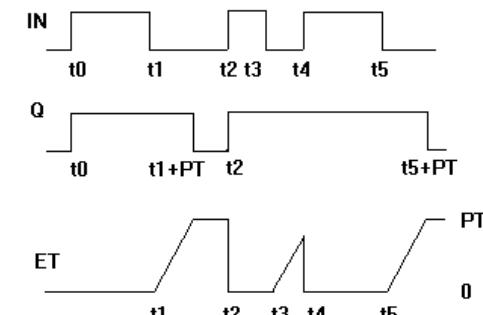
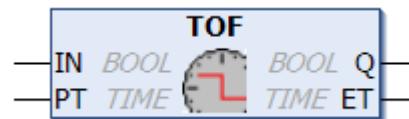
1 fbTMR : TON;
2 fbTMR (IN := %OX5, PT := T#300ms, fbTMR.Q => bVarA);

```

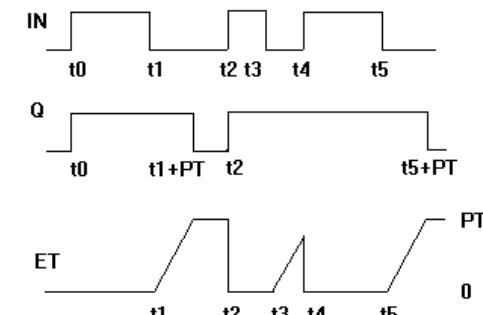
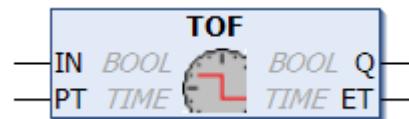
$\Rightarrow$  eine Funktions-Variable von VAR\_INPUT wird ein Wert zugewiesen  
 $\Rightarrow$  der Wert nach Funktionsaufruf wird einer Variable zugewiesen

## TwinCat - Funktionsblöcke

### Timer On-delay (TON)



### Timer Off-Delay (TOF)



IN Eingangssignal (1  $\rightarrow$  0 startet Timer)

PT Verzögerungszeit

Q Ausgang des Timers (1 gehalten bis PT erreicht)

ET abgelaufene Zeit

```

1 VAR
2 fbTOF : TOF;
3 END_VAR

```

```

1 fbTOF(IN := bInput, PT := T#500MS);
2 bLamp := fbTOF.Q;

```

```

1 VAR
2 fbTON : TON;
3 END_VAR

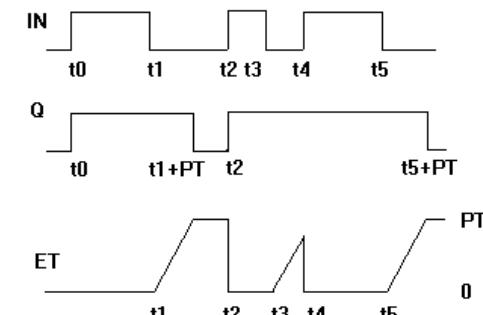
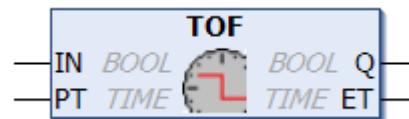
```

```

1 fbTON(IN := bInput, PT := T#500MS);
2 bLamp := fbTON.Q;

```

### Timer Pulse Generator (TP)



IN Eingangssignal (0  $\rightarrow$  1 startet Timer)

PT Pulselänge

Q Ausgang des Timers (1 wenn  $t < PT$ )

ET abgelaufene Zeit

```

1 VAR
2 fbTP : TP;
3 END_VAR

```

```

1 fbTP(IN := bInput, PT := T#500MS);
2 bLamp := fbTP.Q;

```

## Beispiel Code

### <> Edge Detector FB\_EdgeDetector

```

1 FUNCTION_BLOCK FB_EdgeDetector
2 VAR_INPUT
3   bInputValue : BOOL;
4 END_VAR
5
6 VAR_OUTPUT
7   bRisingEdge : BOOL;
8   bFallingEdge : BOOL;
9   bDoubleTap : BOOL;
10 END_VAR
11 VAR
12   bOldValue : BOOL := FALSE;
13   fbTimerDoubleTap : TON;
14   bTimerDoubleTapFlag : BOOL := FALSE;
15   tTimerDuration : TIME := T#2000MS;
16 END_VAR

```

```

1 IF bInputValue > bOldValue THEN
2   bRisingEdge := TRUE;
3   bFallingEdge := FALSE;
4 ELSIF bInputValue < bOldValue THEN
5   bRisingEdge := FALSE;
6   bFallingEdge := TRUE;
7 ELSE
8   bRisingEdge := FALSE;
9   bFallingEdge := FALSE;
10 END_IF
11
12 IF bRisingEdge OR bDoubleTap OR fbTimerDoubleTap.Q THEN
13   IF bTimerDoubleTapFlag = FALSE AND bDoubleTap = FALSE THEN // first tap
14     bTimerDoubleTapFlag := TRUE; // start the timer
15   ELSIF bTimerDoubleTapFlag = TRUE AND fbTimerDoubleTap.Q = FALSE THEN
16     bDoubleTap := TRUE;
17     bTimerDoubleTapFlag := FALSE;
18   ELSE
19     bTimerDoubleTapFlag := FALSE;
20     bDoubleTap := FALSE;
21   END_IF
22 END_IF
23

```

```

24   fbTimerDoubleTap(IN:=bTimerDoubleTapFlag, PT:= tTimerDuration);
25
26   bOldValue := bInputValue;

```

### <> PWM Signal F\_PWMFunction

```

1 FUNCTION F_PwmSignal : REAL
2 VAR_INPUT
3   rYMax : REAL := 1;
4   rYMin : REAL := 0;
5   rDutyCycle : REAL := 0.5;
6   rYOffset : REAL := 0;
7   rAngularFrequency : REAL;
8   rTime : REAL;
9 END_VAR
10 VAR
11   rPeriod : REAL;
12   rModuleTime : REAL;
13   rModulo : INT;
14   rThreshold : REAL;
15 END_VAR

```

```

1   rPeriod := 2.0 * PI / rAngularFrequency;
2
3   rModulo := (TRUNC_INT(rTime/rPeriod));
4   rModuleTime := rTime - rModulo * rPeriod;
5   rThreshold := (rPeriod * rDutyCycle);
6   IF rModuleTime ≥ rThreshold THEN
7     PwmSignal := rYMin + rYOffset;
8   ELSE
9     PwmSignal := rYMax + rYOffset;
10 END_IF

```

### <> Sinus Signal F\_SinusFunction

```

1 FUNCTION F_SinusFunction : REAL
2 VAR_INPUT
3   rAmplitude : REAL := 1;
4   rYOffset : REAL := 0;
5   rAngularFrequency : REAL;
6   rTime : REAL;
7 END_VAR

```

```
1 SinusFunction := SIN(rAngularFrequency * rTime) * rAmplitude + rYOffset;
```

## <> PWM Signal F\_SquareFunction .....

```
1 FUNCTION F_SquareSignal : REAL
2 VAR_INPUT
3   rYMax : REAL := 1;
4   rYMin : REAL := 0;
5   rYOffset : REAL := 0;
6   rAngularFrequency : REAL;
7   rTime : REAL;
8 END_VAR
9 VAR
10  rPeriod : REAL;
11  rModuleTime : REAL;
12  rModulo : INT;
13 END_VAR
```

```
1 rPeriod := 2.0 * PI / rAngularFrequency;
2
3 rModulo := (TRUNC_INT(rTime/rPeriod));
4 rModuleTime := rTime - rModulo * rPeriod;
5
6 IF rModuleTime ≥ (rPeriod * 0.5) THEN
7   SquareSignal := rYMin + rYOffset;
8 ELSE
9   SquareSignal := rYMax + rYOffset;
10 END_IF
```

## <> PT1 Prozess .....

```
1 FUNCTION_BLOCK FB_LowpassFilter
2 VAR_INPUT
3   rInputValue : REAL; // x[n]
4   rDeltaTimeSec : REAL;
5   rTau : REAL;
6 END_VAR
7 VAR_OUTPUT
8   rOutputValue : REAL; // y[n]
9 END_VAR
10 VAR
11   rLastOutputValue : REAL := 0; // y[n-1]
```

```
12 END_VAR
```

```
1 rOutputValue := (rDeltaTimeSec * rInputValue + rTau * rLastOutputValue) /
(rDeltaTimeSec + rTau);
2 rLastOutputValue := rOutputValue;
```

## <> PT2 Prozess .....

### Struct Parameter

```
1 TYPE STRUCT_PT1T2_Parameter :
2 STRUCT
3   rK : REAL;
4   rT1 : REAL;
5   rT2 : REAL;
6 END_STRUCT
7 END_TYPE
```

### Prozess Implementation

```
1 FUNCTION_BLOCK FB_SecondOrderLag
2 VAR_INPUT
3   rInputValue : REAL; // x[n]
4   rDeltaTimeSec : REAL;
5   rRotFreq : REAL;
6   rDamping : REAL;
7 END_VAR
8 VAR_OUTPUT
9   rOutputValue : REAL; // y[n]
10 END_VAR
11 VAR
12   rLastOutputValues : ARRAY[0..1] OF REAL := [0,0]; // y[n-1], y[n-2]
13 END_VAR
```

```
1 rOutputValue := (rInputValue * EXPT(rDeltaTimeSec,2) * EXPT(rRotFreq,2)
2           + rLastOutputValues[0] * (2 + 2 * rDamping * rRotFreq *
rDeltaTimeSec)
3           - rLastOutputValues[1])
4           / (1 + 2 * rDamping * rRotFreq * rDeltaTimeSec + EXPT(rDeltaTimeSec,2) *
EXPT(rRotFreq,2));
5
6 // Shift InputValue into the lastValues array
```

```

7 rLastOutputValues[1] := rLastOutputValues[0];
8 rLastOutputValues[0] := rOutputValue;

```

## <> PID Regler Prozess .....

### SystemModus Enumeration

```

1 {attribute 'qualified_only'}
2 {attribute 'strict'}
3 TYPE ENUM_SystemModus :
4 (
5   System_Init := 0,
6   System_Run,
7   OpenLoop,
8   ClosedLoop_Real,
9   ClosedLoop_Simulation
10 );
11 END_TYPE

```

### Struct Parameter

```

1 TYPE STRUCT_PID_Parameter :
2 STRUCT
3   rKp : REAL; // P-Anteil
4   rTi : REAL; // I-Anteil
5   rTd : REAL; // D-Anteil
6   uN : UINT; // D-Anteil Filterung
7   bARW : BOOL; // Anti-Reset Windup aktiv (true: yes)
8   rTr : REAL; // Anti-Reset Windup Gewichtung (1/Tr)
9   rUmin: REAL; // Steuergrösse minimaler Wert
10  rUmax: REAL; // Steuergrösse maximaler Wert
11 END_STRUCT
12 END_TYPE

```

### Regler Implementation

```

1 FUNCTION_BLOCK FB_PID_Regler
2 VAR_INPUT
3   rR : REAL;
4   rY : REAL;
5   eSystemModus : ENUM_SystemModus;
6   sParam : STRUCT_PID_Parameter;

```

```

7   rT : REAL;
8   END_VAR
9   VAR_OUTPUT
10  rU : REAL;
11  rUsat : REAL;
12 END_VAR
13 VAR
14  rTau : REAL;
15  rE : REAL;
16  rE_old : REAL;
17  rUp : REAL := 0;
18  rUi : REAL := 0;
19  rUi_old : REAL := 0;
20  rUsat_old : REAL := 0;
21  rI : REAL := 0;
22  rI_old : REAL := 0;
23  rUd : REAL := 0;
24  rY_old : REAL := 0;
25  rUd_old : REAL := 0;
26  cIntegralFactor : REAL;
27  cARWIntegralFactor : ARRAY[0..2] OF REAL;
28  cDiffFactor : ARRAY[0..1] OF REAL;
29 END_VAR

```

```

1 CASE eSystemModus OF
2   ENUM_SystemModus.System_Init:
3     rTau := sParam.rTd/UINT_TO_REAL(sParam.uN);
4
5     // precalculation of factor for I-Value
6     cIntegralFactor := rT * sParam.rKp/(2 * sParam.rTi);
7
8     IF sParam.bARW THEN
9       cARWIntegralFactor[0] := sParam.rKp/sParam.rTi;
10      cARWIntegralFactor[1] := 1/sParam.rTr;
11      cARWIntegralFactor[2] := rT/2;
12    END_IF
13
14    // precalculation of factor for D-Value
15    cDiffFactor[0] := sParam.rTd * sParam.rKp / (rTau + rT); // for e[k] &
16    e[k-1]
17    cDiffFactor[1] := rTau / (rTau + rT); // for u[k-1]
18
19   ENUM_SystemModus.ClosedLoop_Real, ENUM_SystemModus.ClosedLoop_Simulation,
20   ENUM_SystemModus.System_Run:
21     // calculate error

```

```
20 rE := rR - rY;
21
22 // calculate P value
23 rUp := rE * sParam.rKp;
24
25 // calculate I value
26 IF sParam.bARW THEN // anti-reset windup
27 // substituted value
28 rI := cARWIntegralFactor[0] * rE + cARWIntegralFactor[1] * (rUsat_old -
29 rU);
30
31 // integration
32 rUi := rUi_old + cARWIntegralFactor[2] * (rI + rI_old);
33 rI_old := rI;
34 ELSE
35 rUi := rUi_old + cIntegralFactor * (rE + rE_old);
36 END_IF
37
38 // calculate D value
39 rUd := cDiffFactor[0] * (rE - rE_old) + cDiffFactor[1] * rUd_old;
40
41 // calculate U
42 rU := rUp + rUi + rUd;
43
44 IF rU > sParam.rUmax THEN
45 rUsat := sParam.rUmax;
46 ELSIF rU < sParam.rUmin THEN
47 rUsat := sParam.rUmin;
48 ELSE
49 rUsat := rU;
50 END_IF
51
52 rE_old := rE;
53 rUi_old := rUi;
54 rUd_old := rUd;
55 rUsat_old := rUsat;
56 rY_old := rY;
57 END_CASE
```

## Weise Seiten

### z-Transformation

#### Definition

zweiseitig:	einseitig:
$X(z) = \sum_{k=-\infty}^{\infty} x[k] z^{-k}$	$X(z) = \sum_{k=0}^{\infty} x[k] z^{-k}$
Konvergenzgebiet $\mathcal{K}$ : $a <  z  < b$	Konvergenzgebiet $\mathcal{K}$ : $ z  > a$

#### Inverse z-Transformation

$$x[k] = \frac{1}{2\pi j} \oint_{\mathcal{C}} X(z) z^{k-1} dz = \sum_{\alpha_i \in \mathcal{A}} \operatorname{Res} \left\{ X(z) \cdot z^{k-1}; \alpha_i \right\}$$

$\mathcal{C}$ : pos. orientierte Kurve in  $\mathcal{K}$ .  $\mathcal{A}$ : Menge aller von  $\mathcal{C}$  umschlossenen Pole.

#### Eigenschaften und Rechenregeln

	Zeitbereich	Bildbereich	Konvergenz
Linearität	$c_1 x_1[k] + c_2 x_2[k]$	$c_1 X_1(z) + c_2 X_2(z)$	$\mathcal{K}_{x_1} \cap \mathcal{K}_{x_2}$
Faltung	$x[k] * y[k]$	$X(z) \cdot Y(z)$	$\mathcal{K}_x \cap \mathcal{K}_y$
Verschiebung	$x[k - k_0]$	$z^{-k_0} X(z)$	$\mathcal{K}_x$
Dämpfung	$a^k \cdot x[k]$	$X\left(\frac{z}{a}\right)$	$ a  \cdot \mathcal{K}_x$
lineare Gewichtung	$k \cdot x[k]$	$-z \cdot \frac{d}{dz} X(z)$	$\mathcal{K}_x$
konj. komplexes Signal	$x^*[k]$	$X^*(z^*)$	$\mathcal{K}_x$
Zeitinversion	$x[-k]$	$X\left(\frac{1}{z}\right)$	$1/\mathcal{K}_x$
diskrete Ableitung	$x[k] - x[k - 1]$	$X(z) \cdot \frac{z-1}{z}$	$\mathcal{K}_x$
diskrete Integration	$\sum_{i=\infty}^k x[i]$	$X(z) \cdot \frac{z}{z-1}$	$\mathcal{K}_x \cap \{ z  > 1\}$
periodische Fortsetzung	$\sum_{i=0}^{\infty} x[k - i N_p]$	$X(z) \cdot \frac{1}{1-z^{-N_p}}$	$\mathcal{K}_x \cap \{ z  > 1\}$
Upsampling	$x\left[\frac{k}{N}\right]$	$X(z^N)$	$\sqrt[N]{\mathcal{K}_x}$

#### Spezielle Eigenschaften der einseitigen z-Transformation

Verschiebung links	$x[k + k_0], k_0 > 0$	$z^{k_0} X(z) - \sum_{i=0}^{k_0-1} x[i] z^{k_0-i}$
Verschiebung rechts	$x[k - k_0], k_0 > 0$	$z^{-k_0} X(z) + \sum_{i=-k_0}^{-1} x[i] z^{-k_0-i}$
Anfangswertsatz	$x[0] = \lim_{z \rightarrow \infty} X(z)$ , falls Grenzwert existiert	
Endwertsatz	$\lim_{k \rightarrow \infty} x[k] = \lim_{z \rightarrow 1} (z-1) X(z)$ , falls $X(z)$ nur Pole mit $ z  < 1$ oder bei $z = 1$	

#### Korrespondenzen der z-Transformation

Nr.	$x[k]$	$X(z)$	$\mathcal{K}$
1	$\delta[k]$	1	$z \in \mathbb{C}$
2	$\delta[k - k_0]$	$z^{-k_0}$	$0 <  z  < \infty$
3	$\varepsilon[k]$	$\frac{z}{z-1}$	$ z  > 1$
4	$k \cdot \varepsilon[k]$	$\frac{z}{(z-1)^2}$	$ z  > 1$
5	$a^k \cdot \varepsilon[k]$	$\frac{z}{z-a}$	$ z  >  a $
6	$\binom{k}{m} a^{k-m} \cdot \varepsilon[k]$	$\frac{z}{(z-a)^{m+1}}$	$ z  >  a $
7	$\sin(\Omega_0 k) \cdot \varepsilon[k]$	$\frac{z \cdot \sin(\Omega_0)}{z^2 - 2z \cdot \cos(\Omega_0) + 1}$	$ z  > 1$
8	$\cos(\Omega_0 k) \cdot \varepsilon[k]$	$\frac{z \cdot [z - \cos(\Omega_0)]}{z^2 - 2z \cdot \cos(\Omega_0) + 1}$	$ z  > 1$
9	$a^k \cdot \varepsilon[-k - 1]$	$-\frac{z}{z-a}$	$ z  <  a $
10	$a^{ k },  a  < 1$	$\frac{z \cdot (a - \frac{1}{a})}{(z-a)(z-\frac{1}{a})}$	$ a  <  z  <  \frac{1}{a} $
11	$\frac{1}{k!} \cdot \varepsilon[k]$	$e^{\frac{1}{z}}$	$ z  > 0$

## Laplace Transformation

### Definition

zweiseitig:	einseitig:
$X(s) = \int_{-\infty}^{\infty} x(t) e^{-st} dt$	$X(s) = \int_{0^-}^{\infty} x(t) e^{-st} dt$
Konvergenzgebiet $\mathcal{K}$ : $a < \operatorname{Re}\{s\} < b$	Konvergenzgebiet $\mathcal{K}$ : $\operatorname{Re}\{s\} > a$

### Eigenschaften und Rechenregeln

	Zeitbereich	Bildbereich	Konvergenz
Linearität	$c_1 x_1(t) + c_2 x_2(t)$	$c_1 X_1(s) + c_2 X_2(s)$	$\mathcal{K}_{x_1} \cap \mathcal{K}_{x_2}$
Faltung	$x(t) * y(t)$	$X(s) \cdot Y(s)$	$\mathcal{K}_x \cap \mathcal{K}_y$
Verschiebung	$x(t - t_0)$	$e^{-st_0} \cdot X(s)$	$\mathcal{K}_x$
Dämpfung	$e^{at} \cdot x(t)$	$X(s - a)$	$\mathcal{K}_x + \operatorname{Re}\{a\}$
lineare Gewichtung	$t \cdot x(t)$	$-\frac{d}{ds} X(s)$	$\mathcal{K}_x$
Differentiation	$\frac{d}{dt} x(t)$	$s \cdot X(s)$	$\mathcal{K}_x$
Integration	$\int_{-\infty}^t x(\tau) d\tau$	$\frac{1}{s} \cdot X(s)$	$\mathcal{K}_x \cap \{\operatorname{Re}\{s\} > 0\}$
Skalierung	$x(at)$	$\frac{1}{ a } \cdot X\left(\frac{s}{a}\right)$	$a \cdot \mathcal{K}_x$
konj. komplexes Signal	$x^*(t)$	$X^*(s^*)$	$\mathcal{K}_x$

### Spezielle Eigenschaften der einseitigen Laplace-Transformation

Verschiebung links	$x(t + t_0), t_0 > 0$	$e^{st_0} \left[ X(s) - \int_{0^-}^{t_0} x(t) \cdot e^{-st} dt \right]$
Verschiebung rechts	$x(t - t_0), t_0 > 0$	$e^{-st_0} \left[ X(s) + \int_{-t_0}^{0^-} x(t) \cdot e^{-st} dt \right]$
Differentiation	$\frac{d}{dt} x(t)$	$s \cdot X(s) - x(0^-)$
Anfangswertsatz	$x(0^+) = \lim_{s \rightarrow \infty} s \cdot X(s), \text{ falls } x(0^+) \text{ existiert}$	
Endwertsatz	$\lim_{t \rightarrow \infty} x(t) = \lim_{s \rightarrow 0} s \cdot X(s), \text{ falls } \lim_{t \rightarrow \infty} x(t) \text{ existiert}$	

### Korrespondenzen der Laplace-Transformation

Nr.	$x(t)$	$X(s)$	$\mathcal{K}$
1	$\delta(t)$	1	$s \in \mathbb{C}$
2	$\varepsilon(t)$	$\frac{1}{s}$	$\operatorname{Re}\{s\} > 0$
3	$\rho(t) = t \cdot \varepsilon(t)$	$\frac{1}{s^2}$	$\operatorname{Re}\{s\} > 0$
4	$e^{at} \cdot \varepsilon(t)$	$\frac{1}{s-a}$	$\operatorname{Re}\{s\} > \operatorname{Re}\{a\}$
5	$\frac{t^m}{m!} e^{at} \cdot \varepsilon(t)$	$\frac{1}{(s-a)^{m+1}}$	$\operatorname{Re}\{s\} > \operatorname{Re}\{a\}$
6	$\sin(\omega_0 t) \cdot \varepsilon(t)$	$\frac{\omega_0}{s^2 + \omega_0^2}$	$\operatorname{Re}\{s\} > 0$
7	$\cos(\omega_0 t) \cdot \varepsilon(t)$	$\frac{s}{s^2 + \omega_0^2}$	$\operatorname{Re}\{s\} > 0$
8	$\sin(\omega_0 t + \varphi_0) \cdot \varepsilon(t)$	$\frac{s \cdot \sin(\varphi_0) + \omega_0 \cdot \cos(\varphi_0)}{s^2 + \omega_0^2}$	$\operatorname{Re}\{s\} > 0$
9	$\delta(t - t_0)$	$e^{-st_0}$	$s \in \mathbb{C}$