

Informations- und Automatisierungstechnik

Kapitel 6: Rechnerarchitekturen

Univ.-Prof. Dr.-Ing. Mike Barth



Rückblick DT

- Darstellung von Informationen im Rechnersystem
 - Grundlage sind **Wahrheitswerte**
- 8 Elemente, sogenannte **Bits** (Binary Digits) werden zu einer Information zusammengefasst → zu einem **Byte**.
 - Bytes sind **00000000** oder **01010101**
- Den beiden Zuständen eines Bits ordnet man die Werte 0 bzw. 1 zu.
 - Damit lassen sich aus 8 Bits eines Bytes $2^8 = 256$ verschiedene Bitmuster bilden.
- Erleichterung der Lesbarkeit großer Bitfolgen: Hexadezimalschreibweise (**hexa** kommt aus dem Griechischen und bedeutet „sechs“, während **decem** das lateinische Wort für „zehn“ ist)
 - Zahlen zur Basis 16 → 16 mögliche Ziffernsymbole anstatt 2 bei Binärsystem oder 10 bei Dezimalsystem

Rückblick DT

0000	\Rightarrow 0	0100	\Rightarrow 4	1000	\Rightarrow 8	1100	\Rightarrow C
0001	\Rightarrow 1	0101	\Rightarrow 5	1001	\Rightarrow 9	1101	\Rightarrow D
0010	\Rightarrow 2	0110	\Rightarrow 6	1010	\Rightarrow A	1110	\Rightarrow E
0011	\Rightarrow 3	0111	\Rightarrow 7	1011	\Rightarrow B	1111	\Rightarrow F

- Hexadezimalschreibweise:

- Beispiel:

```
Student* stud2 = new Student("Mustermann", "Hanna", 1234123);
std::cout<<&stud2<<std::endl; //0x33687ff738
```

- Präfix **0x** und das Suffix **h** kommen insbesondere in der Programmierung zum Einsatz.

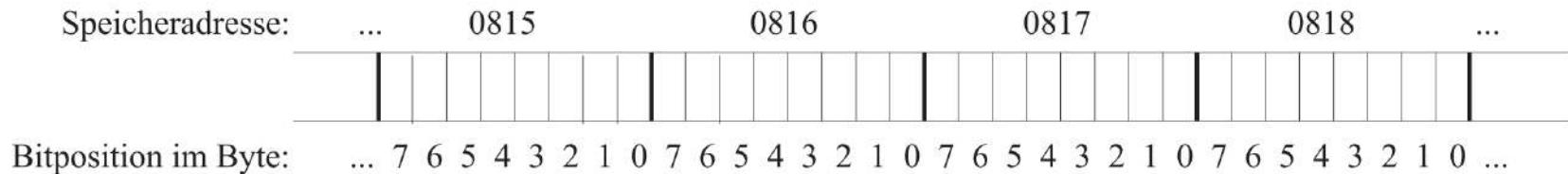
HEX: 0x33687ff738

BIN: 11 0011 0110 1000 0111 1111 1111 0111 0011 1000
(Hinweis für Folgefolie: mehr als 32 bit)

DEC: 220.796.548.920

Rückblick DT

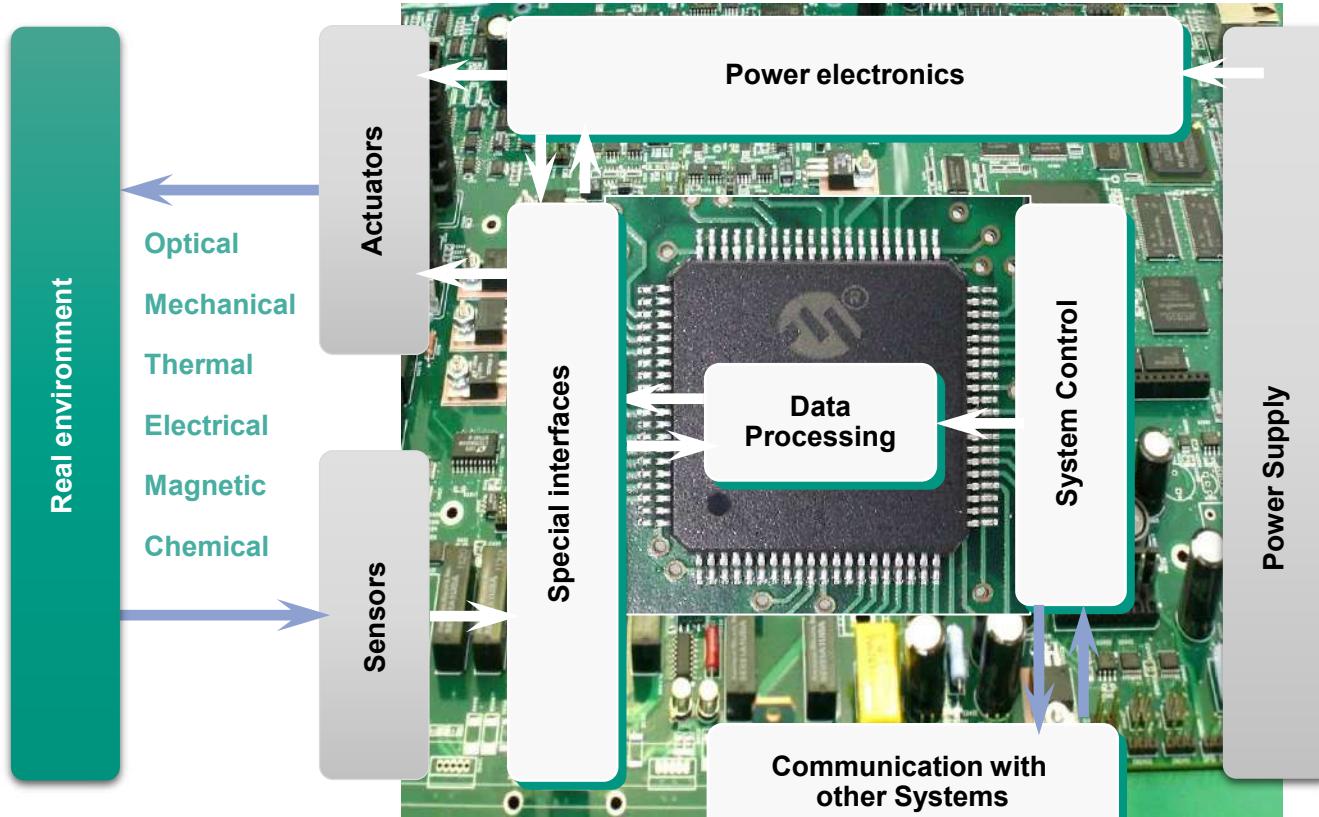
- Ein Byte bildet eine adressierbare Speichereinheit.
 - (Es gibt auch andere Architekturen mit 4 Byte als Speichereinheit)
- Byte-Einheiten bekommen laufenden Nummern: Speicheradresse



Rückblick DT

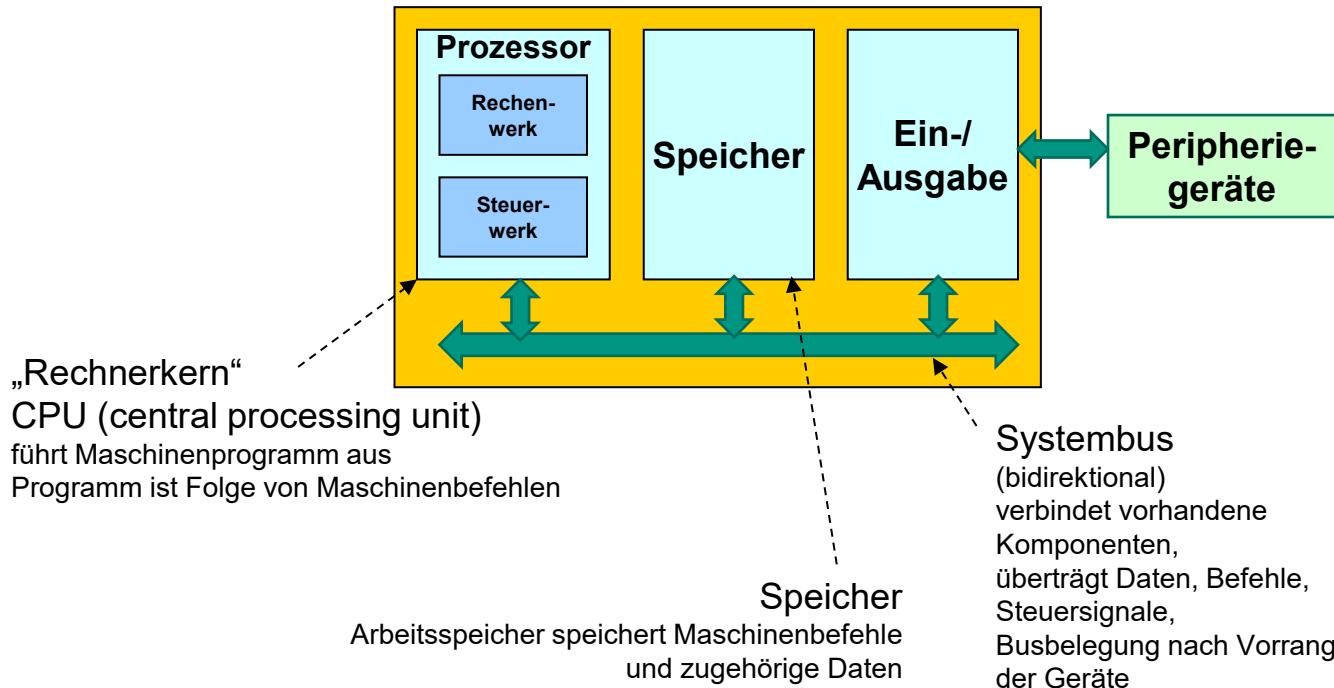
- Die Adressierung der Speichereinheiten beginnt bei 0 und endet bei einem 2^n Byte großen Speicher mit der Adresse $2^n - 1$
 - n ist dabei von der Größe der Hardware abhängig
 - Die Größe des Speichers wird in den üblichen Einheiten angegeben:
 - Beispiel: 1 GB = 1024 MB = 2^{30} Byte = 1.048.576 KB = 1.073.741.824 Byte = 8.589.934.592 Bit $\approx 10^9$ Byte
- Betrachtung eines Speichersystems, das durch 32 Bit lange Adressen seine 1 Byte großen Speicherzellen adressiert.
 - Wie groß kann der Speicher maximal sein?
 - 32 Bit ermöglichen 2^{32} Bitmuster, d.h. ≈ 4 GB können im Speicher adressiert werden.
 - 32 Bit Architektur kann lediglich 4 GB Arbeitsspeicher verwalten!!
 - 64 Bit Architektur kann theoretisch 2^{64} Byte, also 16 Exabyte → Kein Problem mehr ☺

Wiederholung Kapitel 1



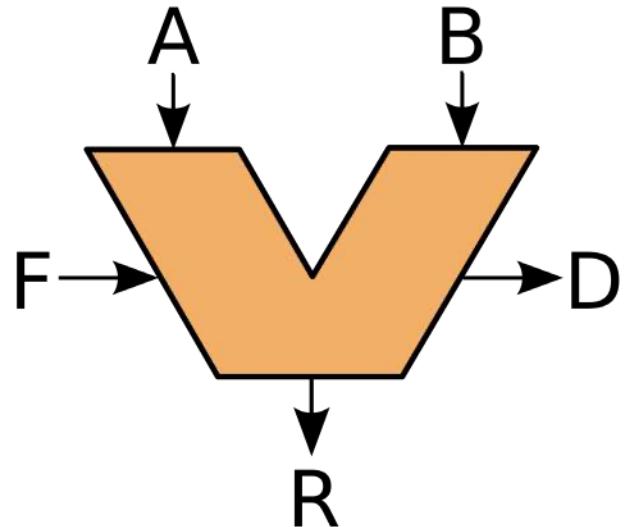
Rechnerarchitekturen (vgl. DT)

■ Interne Architektur nach John von Neumann (1903-1957)



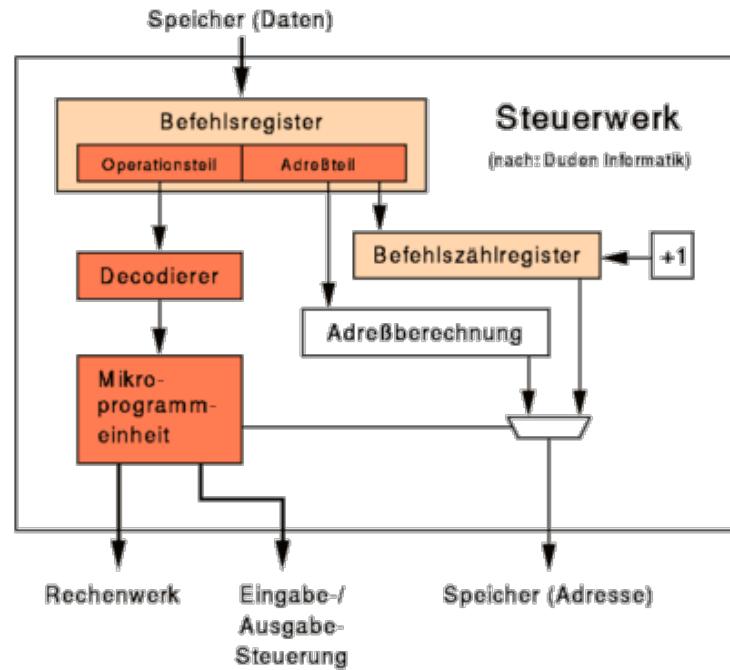
Der Prozessor

- Rechnerkern, Central Processing Unit (**CPU**), zentrale Recheneinheit
 - Ruft die Befehle (Instruktionen) der Programme ab, prüft sie und führt sie nacheinander aus.
 - Aktuelle Technologie: 2010 zeigte ein australisches Team einen funktionstüchtigen Transistor aus 7 Atomen mit einer Länge von 4 nm.
- Unterteilung in Daten- und Befehlsprozessor:
 - **Datenprozessor** verarbeitet die Daten (z.B. Negation) und führt Berechnungen (z.B. Addition) durch.
 - Enthält ein **Rechenwerk**: ALU (Arithmetic Logical Unit) und (mindestens) drei Speicherplätze (Register) zur Aufnahme von Operanden
 - **A** Akkumulator, **B** Datenregister, **F** Funktion, **D** Statusausgabe, **R** Ergebnis (Result)



Der Prozessor

- **Befehlsprozessor:** entschlüsselt Befehle und steuert deren Ausführung
 - Der aktuell zu bearbeitende Befehl befindet sich im Befehlsregister (engl. Instruction Register IR)
 - Die Adresse des Speicherplatzes, der als nächstes angesprochen wird, ist im Speicherregister (SP) vorhanden.
 - Die Adresse des nächsten auszuführenden Befehles wird im Befehlszählregister (engl. Program Counter PC) gespeichert.
 - Die Entschlüsselung eines Befehls erfolgt durch einen separaten Befehlsdecodierer.
 - Die Steuerung der Ausführung erfolgt durch das **Steuerwerk**

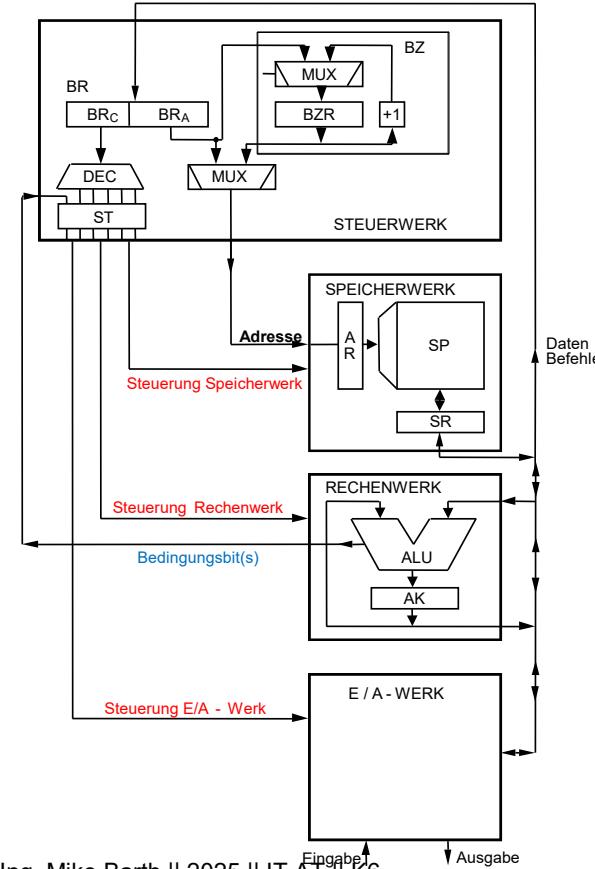
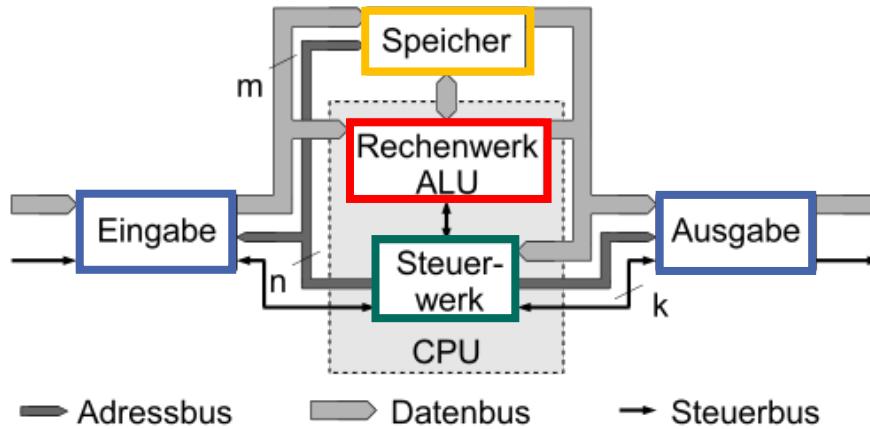


Ein-/Ausgabewerk

- Kommunikation mit dem Benutzer
 - Ein-/Ausgabegeräte (Tastatur, Maus, Bildschirm, Sensor)
 - Peripheriegeräte (z.B. Drucker, Plattenspeicher; Aktuator)
- Eingabe:
 - Übernahme Daten von extern angeschlossenen Geräten (z.B. Sensor)
 - Umsetzung in eine geeignete, normierte Darstellung zur weiteren Verarbeitung
- Ausgabe:
 - Datenformatmäßige und elektrische Aufbereitung der Daten
 - Weiterleitung an externe Ausgabegeräte
 - Eventuell Auswahl des Ausgabegerätes
 - Ansteuern eines Aktors

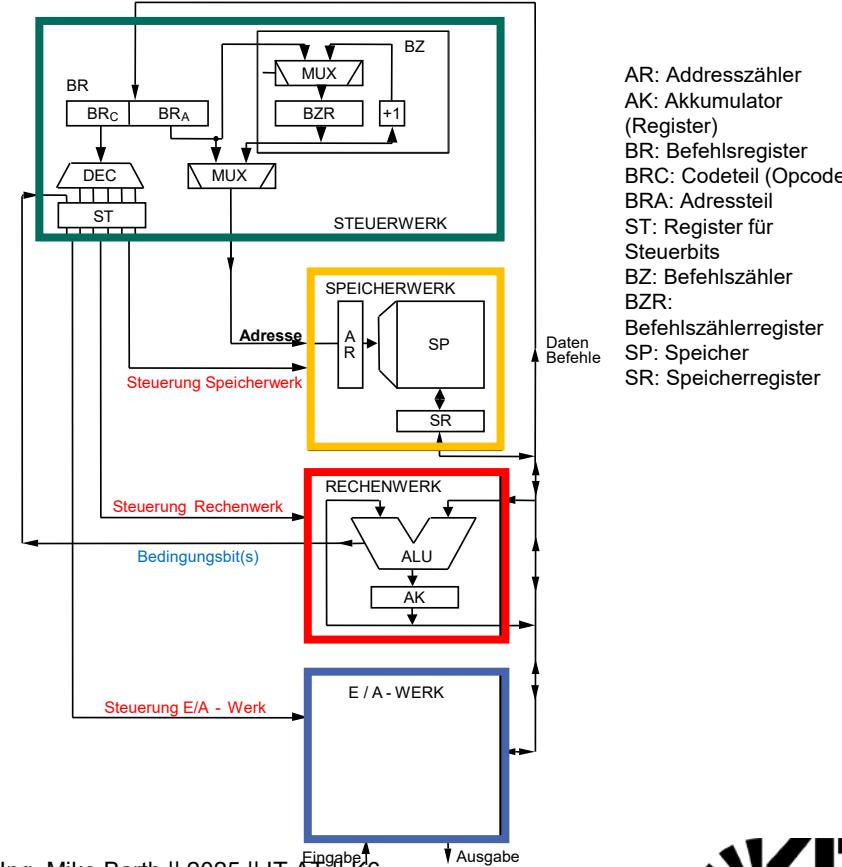
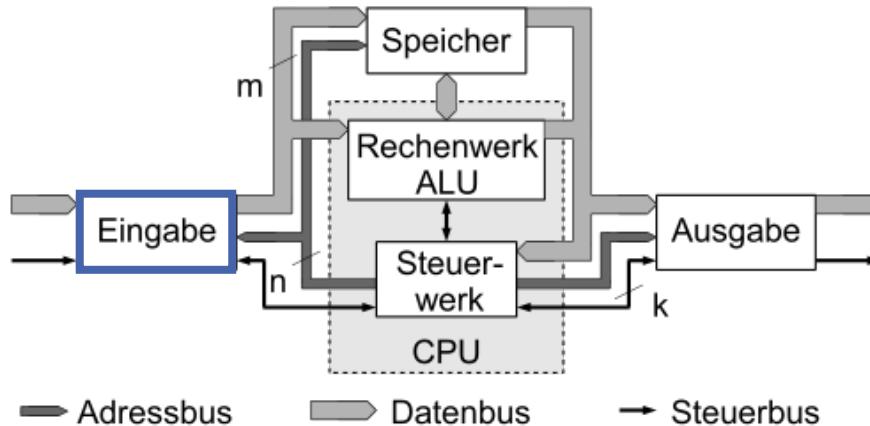
Aufbau eines von Neumann Rechners

Detaillierte Sichtweise



Aufbau eines von Neumann Rechners

Detaillierte Sichtweise



Aufbau eines von Neumann Rechners

Detaillierte Sichtweise

Steuerwerk

- BR: Befehlsregister
 - BCR: Codeteil (Opcode)
 - BRA: Adressteil
- ST: Register für Steuerbits
- BZ: Befehlszähler
 - BZR: Befehlszählerregister

Speicherwerk

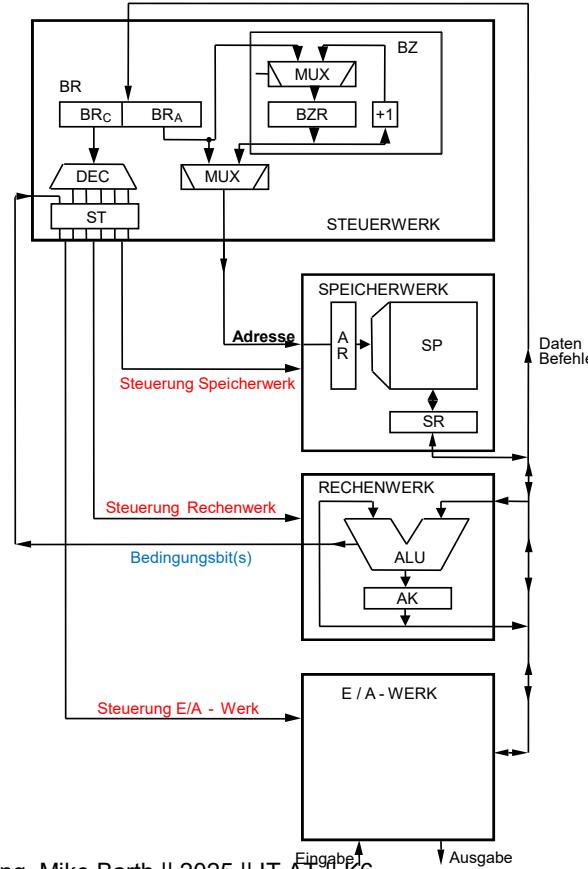
- AR: Adresszähler
- SP: Speicher
- SR: Speicherregister

Rechenwerk

- ALU: Arithmetische-, logische Einheit
- AK: Akkumulator (Register)

E/A-Werk

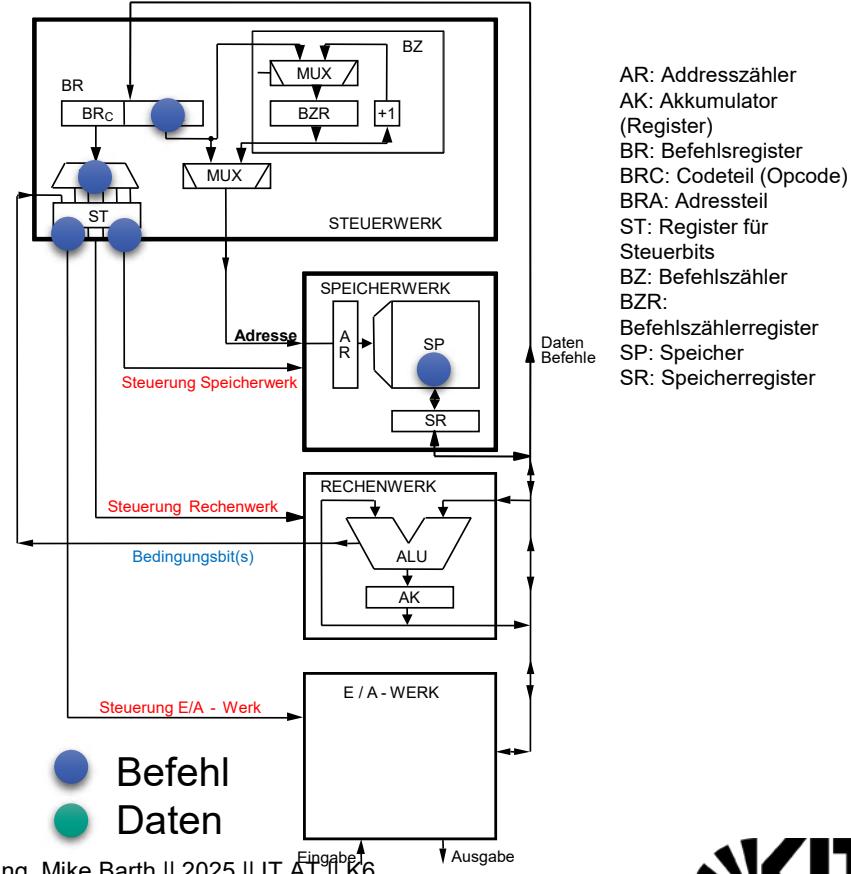
- E/A: Eingabe/Ausgabe



AR: Addresszähler
AK: Akkumulator (Register)
BR: Befehlsregister
BCR: Codeteil (Opcode)
BRA: Adressteil
ST: Register für Steuerbits
BZ: Befehlszähler
BZR: Befehlszählerregister
SP: Speicher
SR: Speicherregister

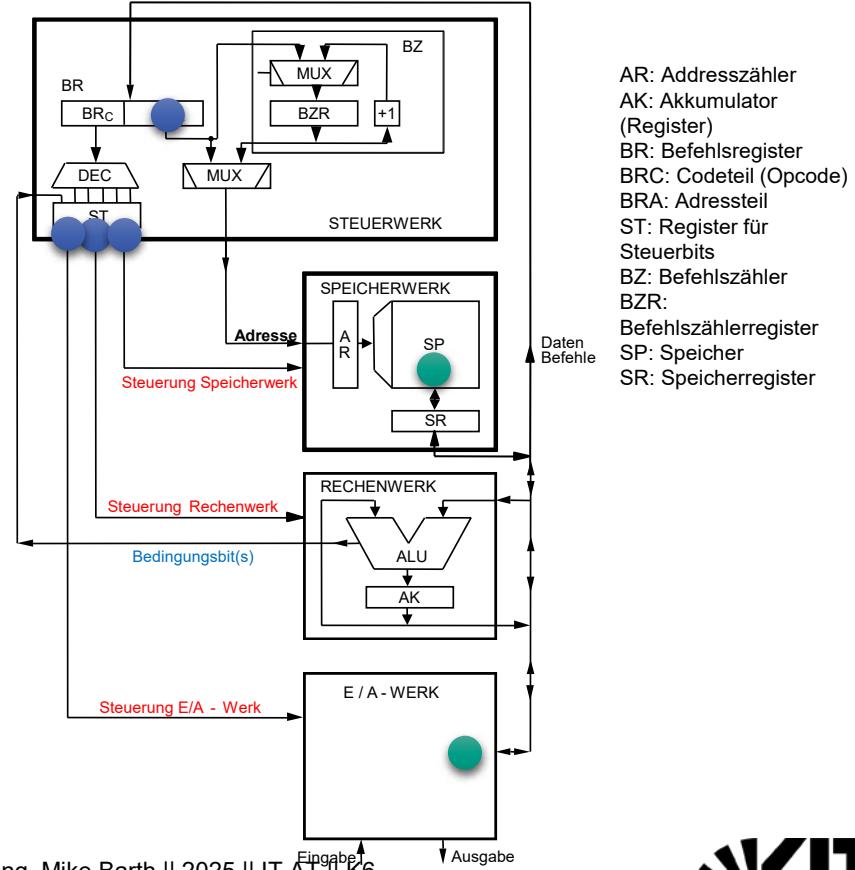
Von-Neumann-Zyklus (Arbeitsweise)

- 1. FETCH: Befehl aus dem Speicher in den Prozessor holen (lesender Zugriff)
- 2. DECODE: Befehl im Steuerwerk dekodieren
- 3. FETCH OPERANDS: gegebenenfalls Operanden aus dem Speicher oder der Peripherie in den Prozessor holen (lesender Zugriff)
- 4. EXECUTE: Befehl im Rechenwerk ausführen
- 5. WRITE BACK: gegebenenfalls Ergebnis in den Speicher oder die Peripherie schreiben (schreibender Zugriff)
- 6. Weiter bei Schritt 1



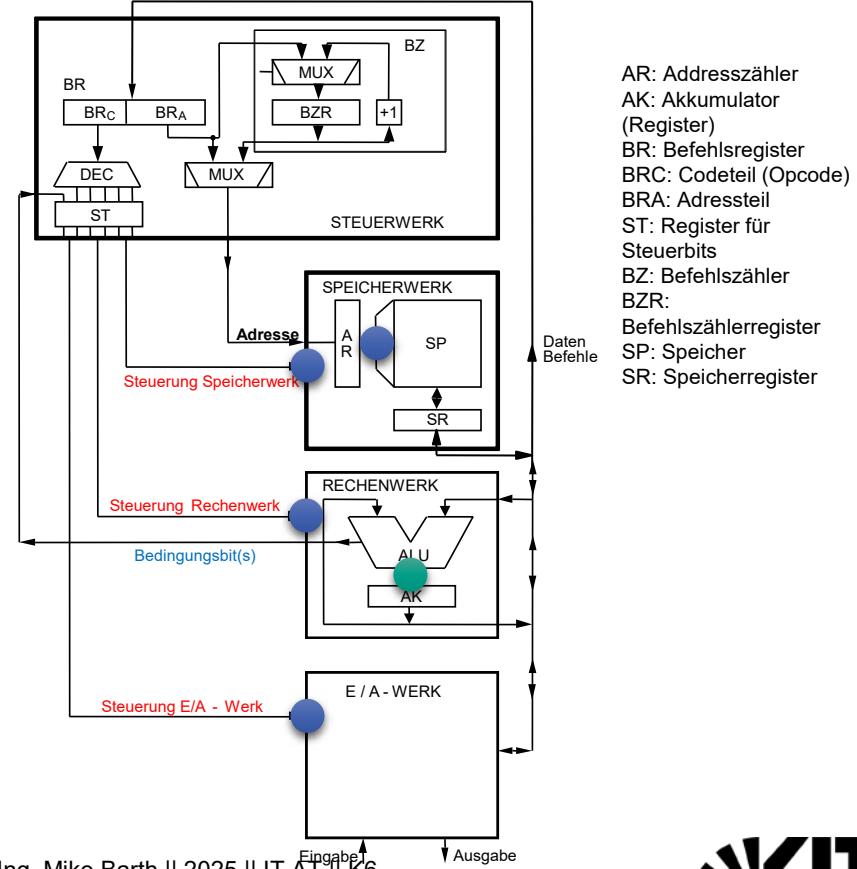
Von-Neumann-Zyklus (Arbeitsweise)

1. FETCH: Befehl aus dem Speicher in den Prozessor holen (lesender Zugriff)
2. DECODE: Befehl im Steuerwerk dekodieren
3. → FETCH OPERANDS: gegebenenfalls Operanden aus dem Speicher oder der Peripherie in den Prozessor holen (lesender Zugriff)
4. → EXECUTE: Befehl im Rechenwerk ausführen
5. WRITE BACK: gegebenenfalls Ergebnis in den Speicher oder die Peripherie schreiben (schreibender Zugriff)
6. Weiter bei Schritt 1

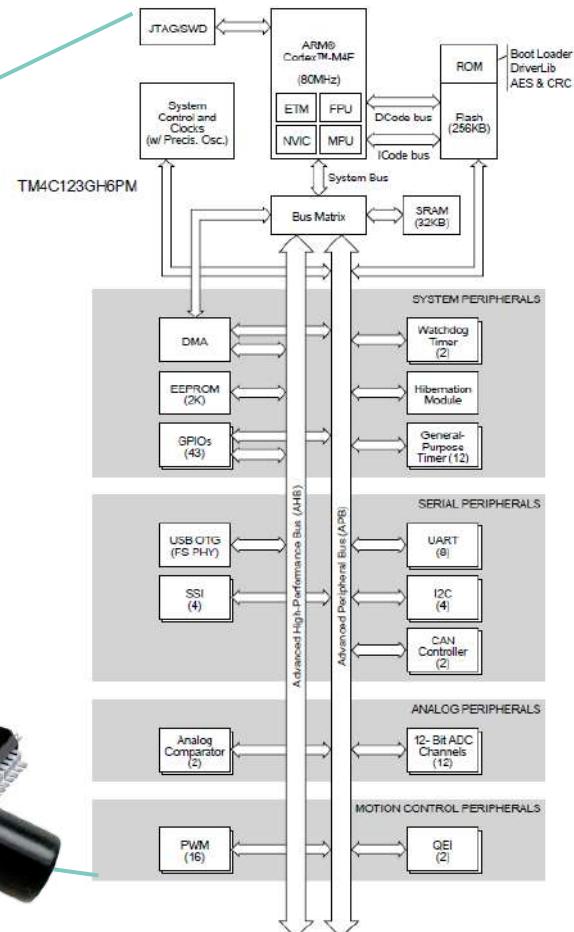
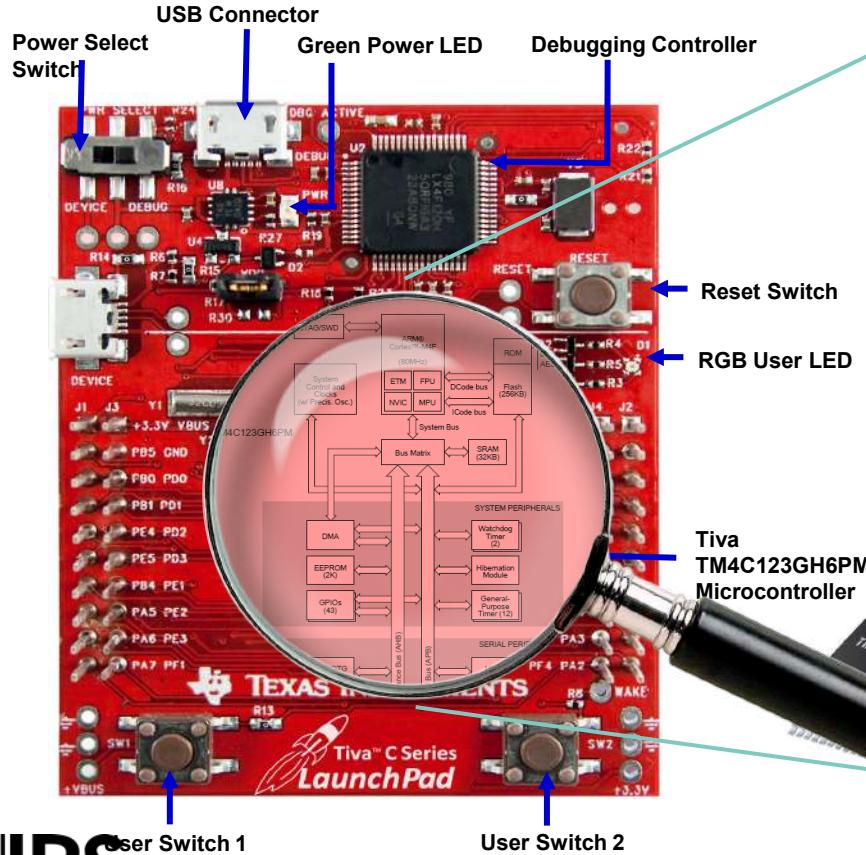


Von-Neumann-Zyklus (Arbeitsweise)

1. FETCH: Befehl aus dem Speicher in den Prozessor holen (lesender Zugriff)
 2. DECODE: Befehl im Steuerwerk dekodieren
 3. FETCH OPERANDS: gegebenenfalls Operanden aus dem Speicher oder der Peripherie in den Prozessor holen (lesender Zugriff)
 4. EXECUTE: Befehl im Rechenwerk ausführen
- WRITE BACK: gegebenenfalls Ergebnis in den Speicher oder die Peripherie schreiben (schreibender Zugriff)
- Weiter bei Schritt 1



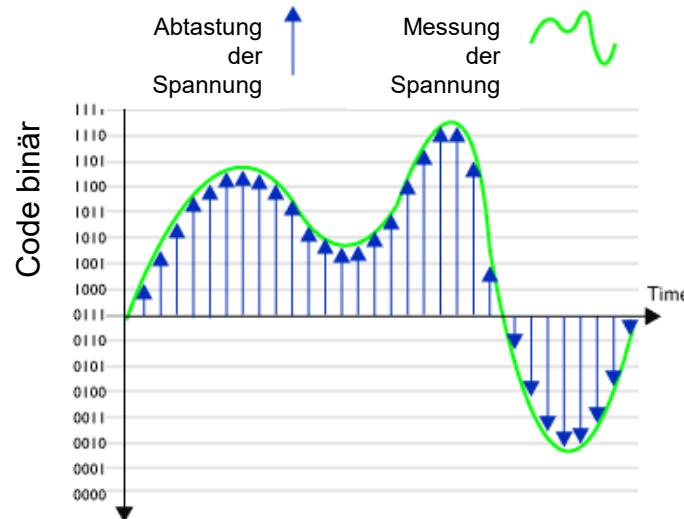
Beispiel ARM Cortex-M4



Verallgemeinerte Prozessorarchitektur für Cortex-M4

Analog/Digital Wandler (ADC)

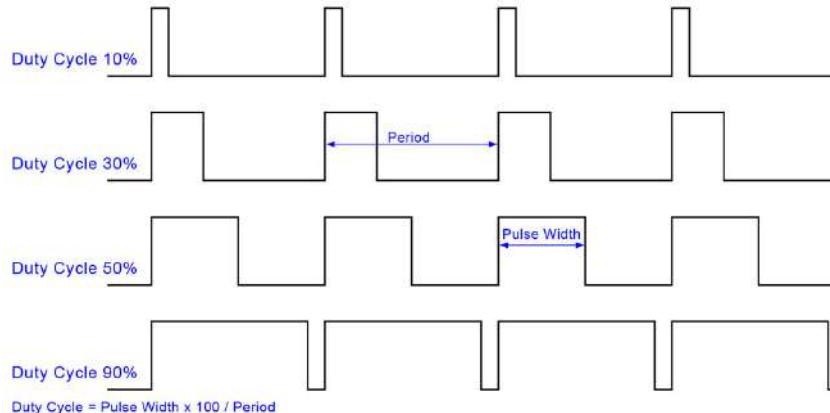
- Wandelt analoge Eingangssignale in digitale Werte
- Amplitude der kontinuierlichen analogen Signale werden diskret abgetastet und als digitale Werte abgespeichert.
- Der abtastbare Spannungsbereich hängt von der Versorgungsspannung des Microcontrollers ab



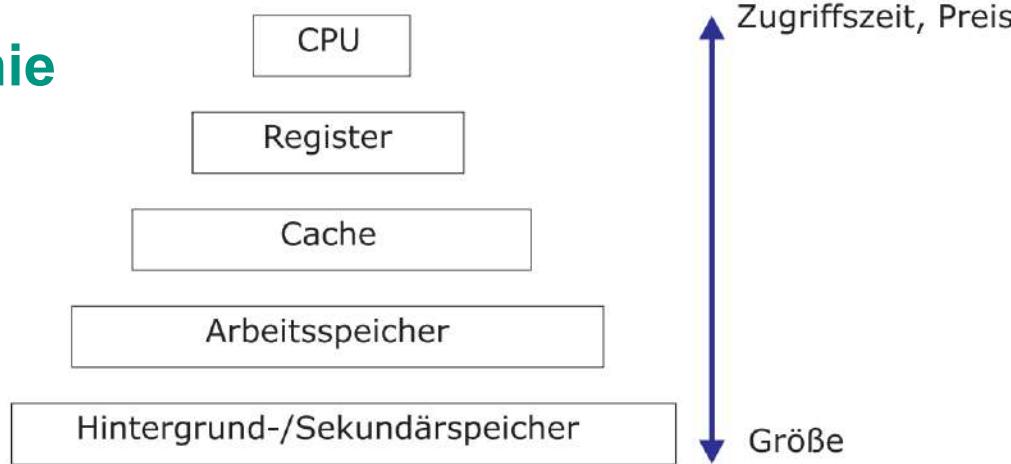
Joystick

Pulsweitenmodulation (PWM)

- Änderung des Puls-Pause-Verhältnisses bei gleicher Frequenz
 - übertragene Energie hängt von Puls-Pause-Verhältnis ab
- Durch Änderung der Pulsbreite pro Periode kann Drehzahl von Motoren digital geändert werden
 - 0 → 1 Wechsel immer zum Start der Periode
 - 1 → 0 Wechsel wird durch parallel laufenden Zähler ermittelt



Speicherhierarchie



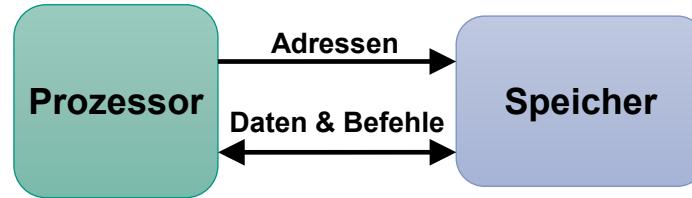
- **Register:** Unmittelbar der CPU zugeordnet, zur Zwischenspeicherung von Werten und zum Ablegen von Informationen, die für die Befehlsausführung benötigt werden.
- **Cache:** Auf der zweiten Hierarchieebene befindet sich der Cache-Speicher, d.h. das Bindeglied zwischen CPU und Arbeitsspeicher, in dem die als nächstes oder die am häufigsten benutzten Daten und Befehle zwischengespeichert werden. Die zugrundeliegende Idee ist die sogenannte 90:10 Regel. Bei 90% aller Zugriffe werden nur 10% aller Daten, mit denen ein Programm insgesamt arbeitet, benötigt.

Speicherhierarchie

- **Hauptspeicher:** Der Arbeitsspeicher (= Hauptspeicher) hat in der Regel (heute) eine Größe im Gigabyte-Bereich.
 - Wie beim Hintergrundspeicher gibt es Arbeitsspeicher im wesentlichen in zwei Typen:
 - ROM read only memory, z. B. für Systemfunktionen wie System-Boot
 - RAM random access memory
- Der **Sekundär- oder Hintergrundspeicher** dient zur Aufnahme von Daten und Programmen, auf die relativ selten zugegriffen werden muss. Außerdem stellt er die einzige Möglichkeit zur persistenten Speicherung, also auch für sichere Langzeitarchivierung, dar → zum Beispiel externe Festplatte
- Je „näher“ der Speicher an der CPU ist, desto teurer ist die Realisierung einer gewissen Speichergröße, da eine geringere Zugriffszeit erforderlich ist.

Parallelisierung „Daten- und Befehlszugriffe“

■ Von Neumann-Architektur:



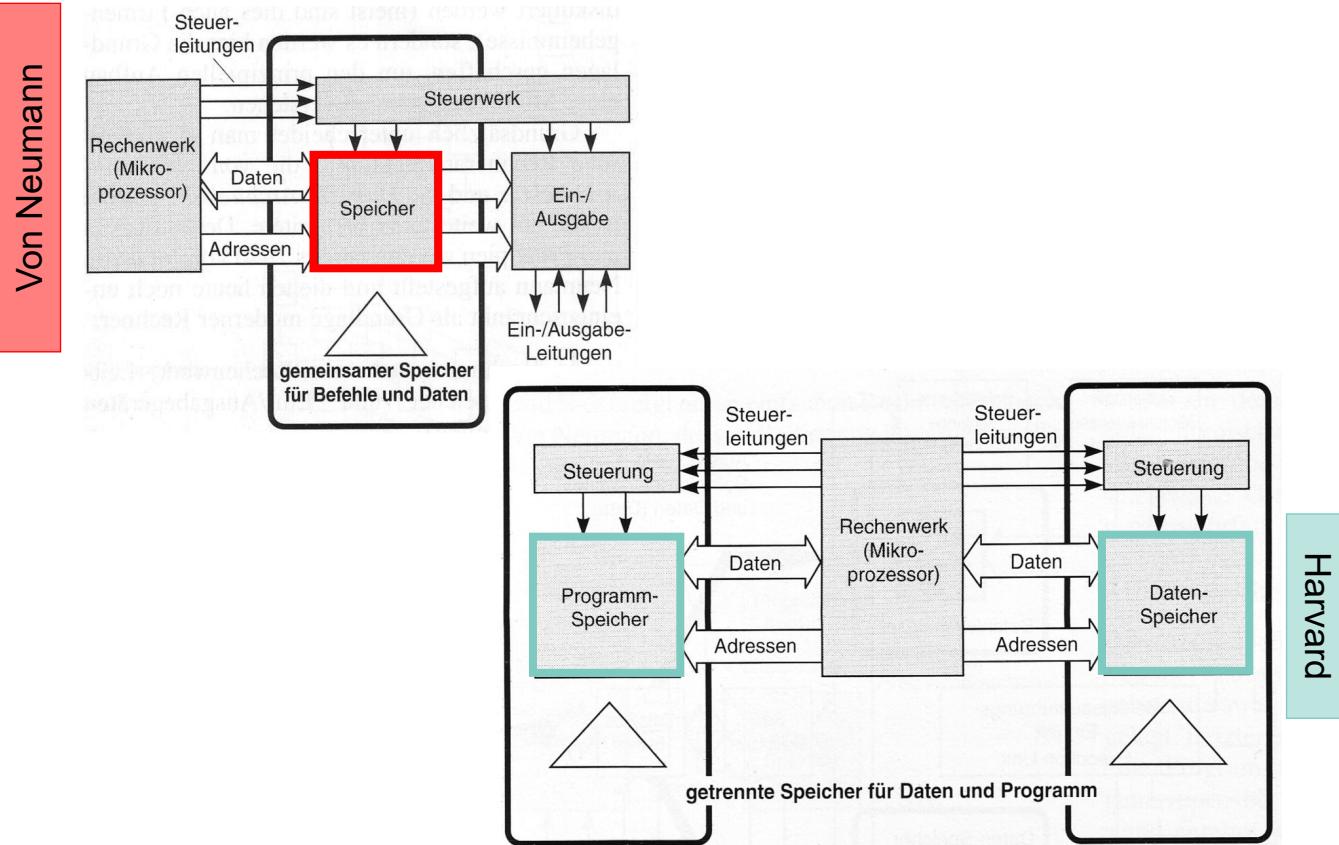
Befehlausführungszeiten im Prozessor heute um Faktor 10-100 mal kleiner als Speicherzugriff: „Von-Neumann-Bottleneck“

- mehr Speicher im Prozessor (Register, Cache-Speicher)
- separate Speicher und Busse für Daten und Befehle (→ Harvard-Architektur)

■ Harvard-Architektur:



Von Neumann und Harvard Architektur

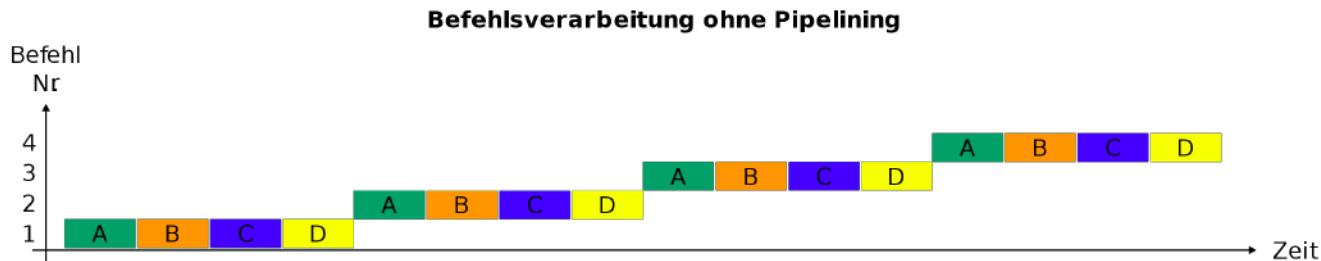
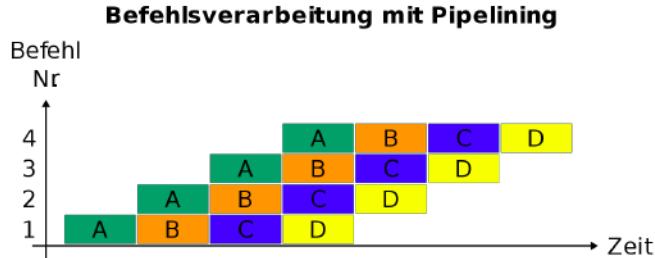


Unterschied RISC und CISC

- Klassische RISC-Prozessoren werden hauptsächlich in Smartphones und Tablets eingesetzt. Die ARM-Architektur ist ein typischer Vertreter. Intel- und AMD-Prozessoren werden als typische CISC-Prozessoren verstanden, die intern aber auch mit RISC-artigen Strukturen arbeiten.
- **CISC – Complex Instruction Set Computing**
 - großen Befehlsumfang und komplexe Adressierungsmöglichkeiten
- **RISC – Reduced Instruction Set Computing**
 - Weniger Befehle und einfachere Befehlsstruktur
 - Kürzere Ausführungszeit pro Befehl
 - Pipelining: Siehe nächste Folie
 - Heute der Standard!

RISC mit Pipelining

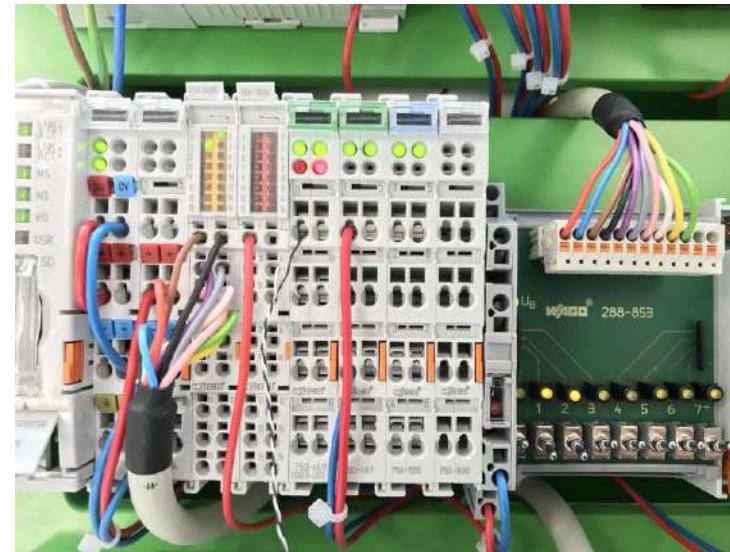
- **Pipelining** bezeichnet eine Art „Fließband“, mit dem die Abarbeitung der Maschinenbefehle in Teilaufgaben zerlegt wird, die für mehrere Befehle parallel durchgeführt werden.



Informations- und Automatisierungstechnik

Kapitel 7: Grundlagen der Automatisierungstechnik

Univ.-Prof. Dr.-Ing. Mike Barth





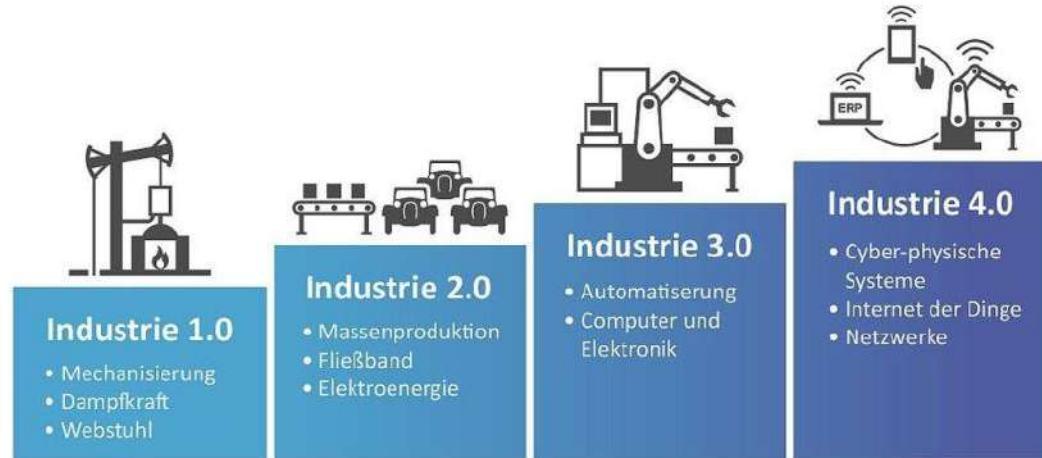
Quelle: BASF



Gebäudeautomation



Motivation

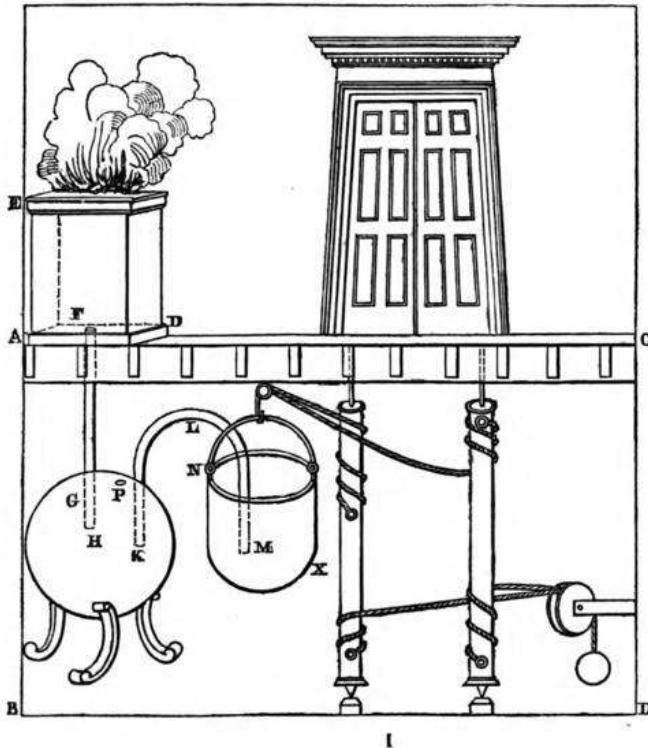


Begriffsdefinition

- Der Begriff **Automation** leitet sich vom griechischen Wort „automatos“ ab, was „sich selbst bewegend“ oder „selbsttätig“ bedeutet.
 - Im antiken Griechenland verehrte man die Göttin „Fortuna“ welche den Beinamen „Automatia“ hatte: sie war die Göttin des Zufalls und des Glückes und der **ohne menschliches Zutun** eintretenden Ereignisse
- Die **Automatisierungstechnik** ist eine Disziplin der **Ingenieurwissenschaften**.
- Nach DIN 19233 ist die Automatisierungstechnik dadurch gekennzeichnet, dass man **künstliche Mittel einsetzt**, um einen Prozess selbstständig ablaufen zu lassen.
- Sie umfasst Methoden, Verfahren sowie die Werkzeuge, die benötigt werden, um einem **System** ein **zielorientiertes, sicheres** und **selbsttätig** ablaufendes Verhalten aufzuprägen.
- DIN IEC 60050-351: Als „selbsttätig/automatisch“ werde ein Prozess oder eine Einrichtung bezeichnet, der oder die unter festgelegten Bedingungen **ohne** menschliches Eingreifen abläuft oder arbeitet.

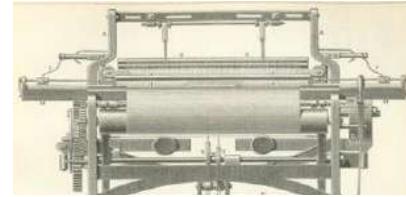
Begriffsdefinition

- Schon im antiken Griechenland nutzte man naturwissenschaftliche Kenntnisse und entwarf erste Automaten, wie zum Beispiel die sich selbst öffnenden Tempeltüren von Alexandria.
- Heron von Alexandria (100 v. Chr.)



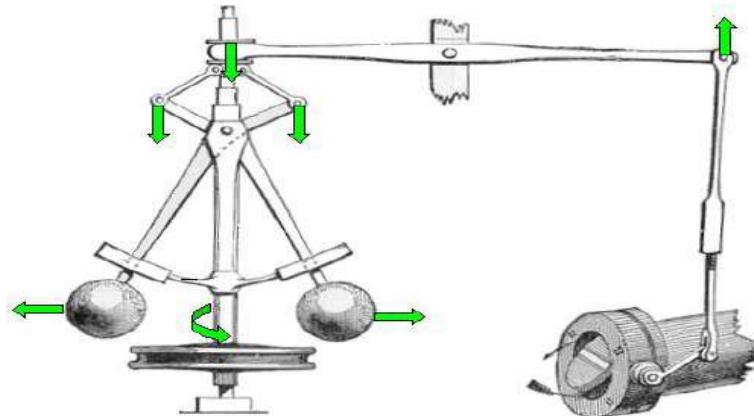
Von der Mechanisierung zur Automatisierung

- 1745: Automatisierung von Windmühlen durch Lee (engl. Schmied)
- 1785: „Power Loom“ Vollautomatisierter Webstuhl von Edmond Cartwright



Von der Mechanisierung zur Automatisierung

- 1788: Fliehkraftregler (James Watt)



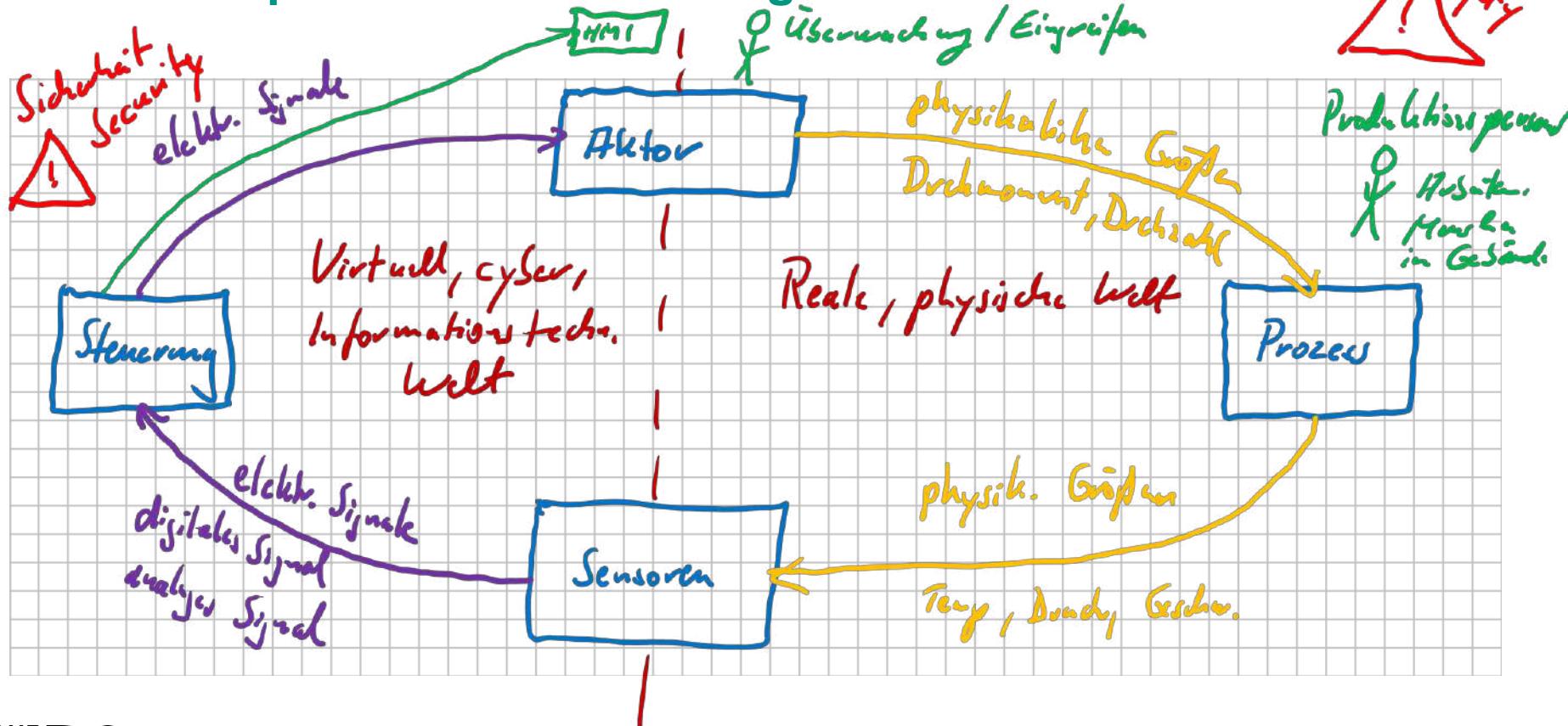
Moderne AT-Anwendungsfelder



Das größte zu automatisierenden Feld der Welt



Das Prinzip der Automatisierungstechnik



Produktionsprozess
! Motor, Motor in Gebäude

Einordnung der AT

1. Hantierung:

- Es werden Werkzeuge eingesetzt. Der Mensch muss für die Bewältigung der technischen Aufgabe die notwendige Energie als Muskelkraft aufbringen und die Reihenfolge der Operationen beachten.



Einordnung der AT

2. Mechanisierung:

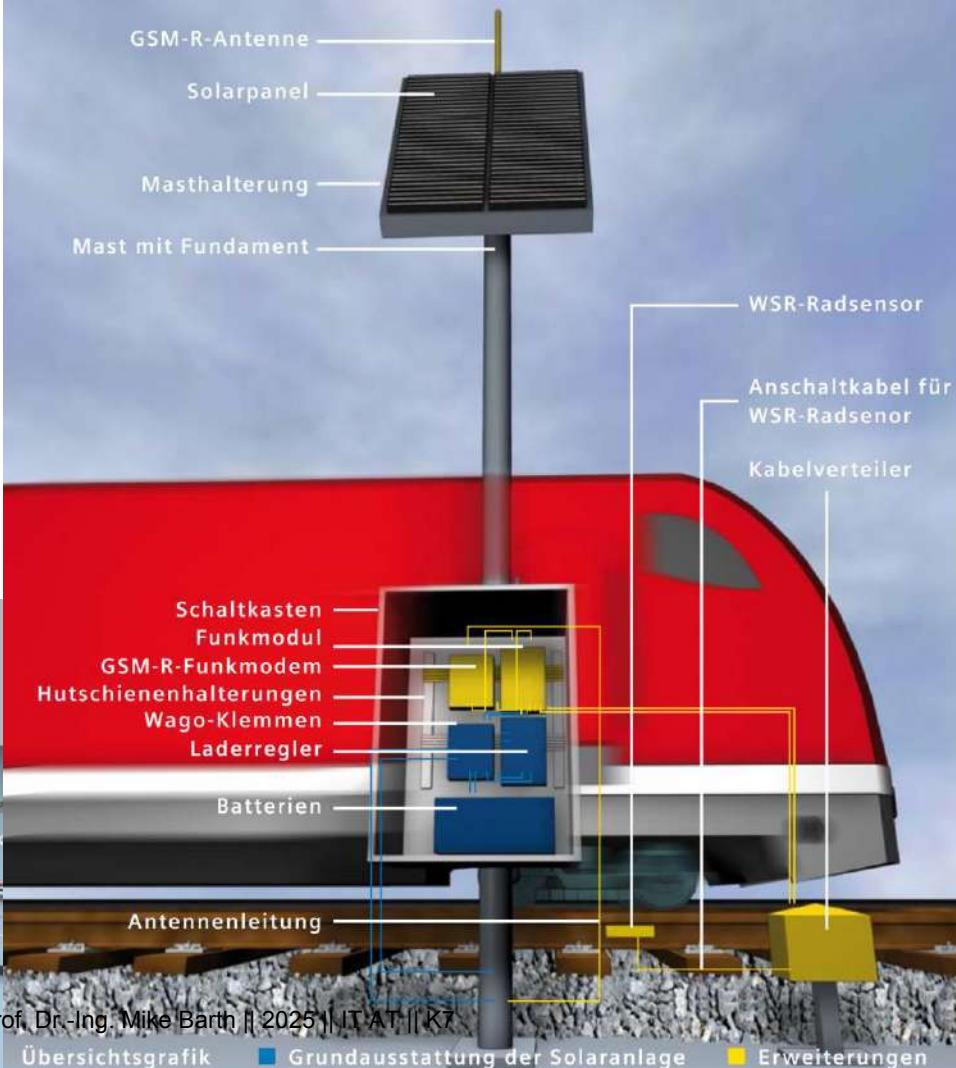
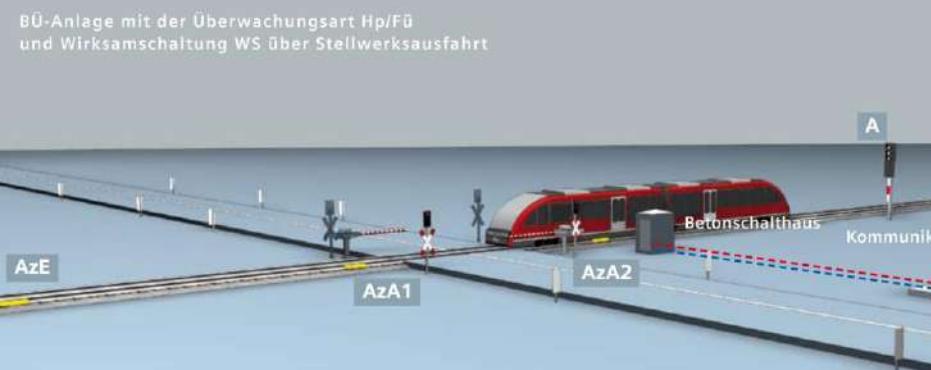
- Maschinen werden für einzelne Operationen eingesetzt. Der Mensch bestimmt jederzeit die Aktion der Maschine.



Einordnung der AT

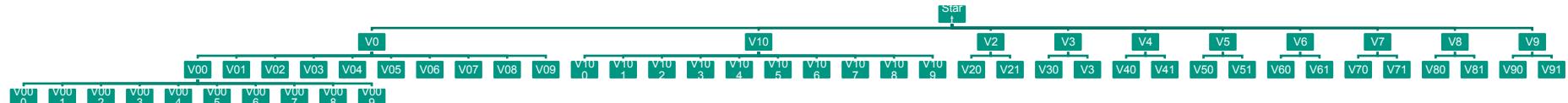
3. Automatisierung:

- Mehrere, immer wiederkehrende Teilschritte bei der technischen Aufgabenbewältigung werden selbsttätig ausgeführt. **Der Mensch hat überwachende Funktion.**



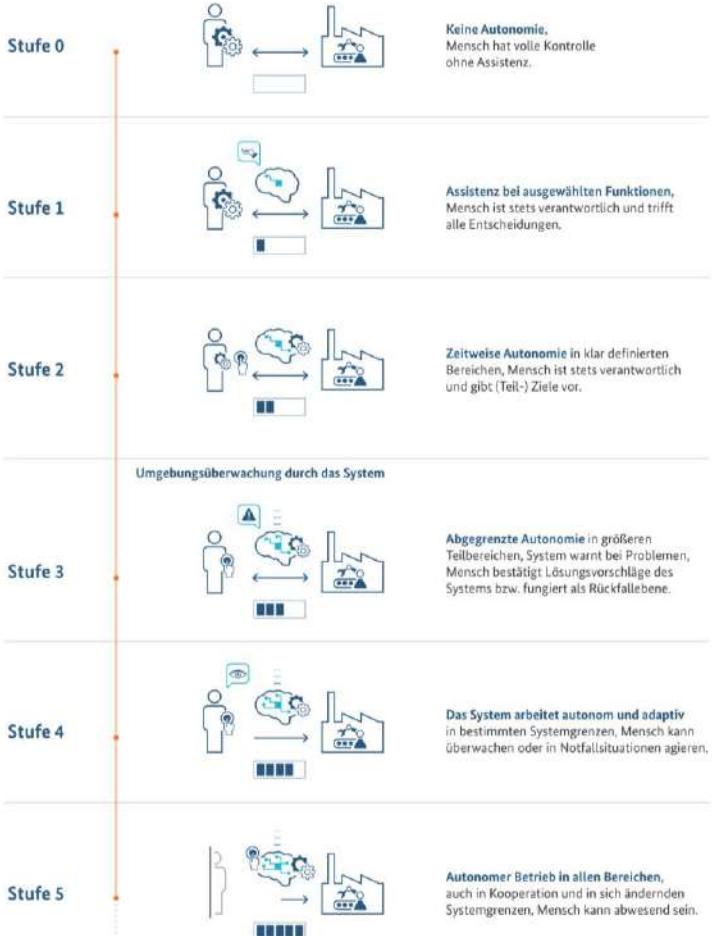
Abgrenzung zu autonomen Systemen

- „Wenn alles im Vorfeld genau durchgedacht ist, auch wenn es noch so kompliziert ist, und wir dem System die durchdachten kausalen Zusammenhänge einprogrammieren, dann reden wir von automatisiert. Wenn wir die kausalen Zusammenhänge aber gar nicht richtig erfassen und dem System mit KI-Ansätze nur indirekt sagen, wie es sich in einer bestimmten Situation verhalten soll, dann reden wir von autonom.“ [Quelle: Autonom oder hochautomatisiert? - Fraunhofer IESE Blog](#)
 - Vgl. Autonomes Fahren
- Automatisiert: die Systeme folgen einem zu 100% deterministischen Verhalten. Aber auch das kann sehr komplex sein.
- Vgl. Entscheidungsbaum mit 10 Varianten pro Ebene resultiert in der 4. Ebene bereits in 10.000 Variationen



Autonomiestufen der Industrie

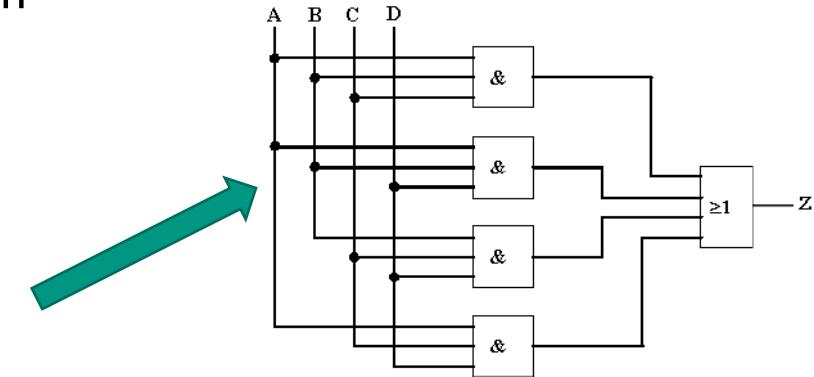
- Abgeleitet aus dem Modell für Autonomiestufen von Fahrzeugen existieren Autonomie-Level auch in der industriellen Automatisierungstechnik.
- Normale Automatisierungssysteme schaffen problemlose Stufe 1-2.
- Stufe 3 erfordert den Einsatz von lernenden Verfahren, welche die selbstständige Entwicklung von Lösungsstrategien durch die Maschine ermöglichen.
 - Der Lösungsraum und Problemraum wird ab hier jedoch deutlich größer



Verbindung zur Informationstechnik

- Die Automatisierungstechnik basierte im Kern lange auf der Konfiguration von Schaltnetzen, Schaltwerken und Automaten anhand standardisierter grafischer und textbasierter Elemente.
- Beispiele:

- Implementierung von Schaltnetzen wie in der Digitaltechnik
- Implementierung von Textanweisungen mit Kontrollfluss-Elementen
- Implementierung von Stromlaufplänen, als Reminiszenz an die Ursprünge der Relais-basierten Steuerungen
- Dies wird im Kapitel zu den IEC 61131-3 Sprachen noch behandelt.



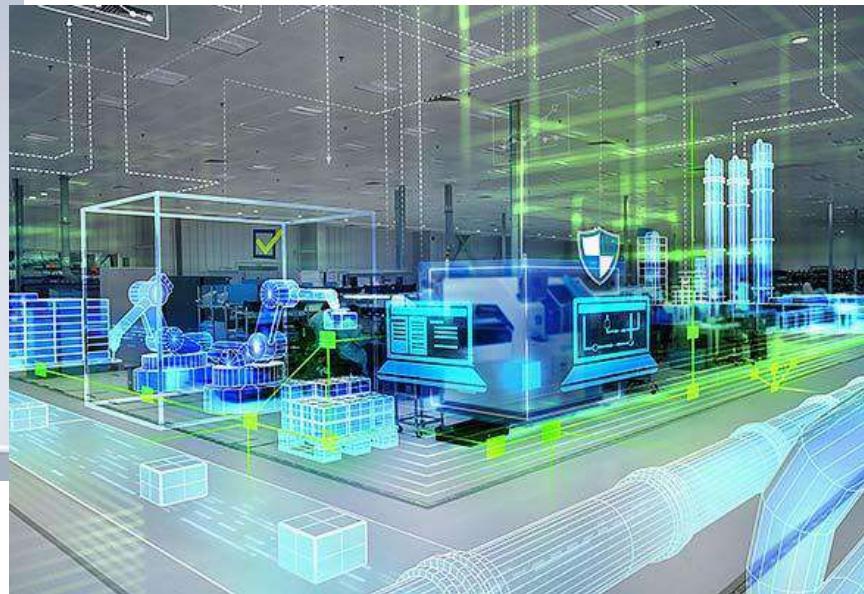
```
MAIN (PRG-ST)
0001 PROGRAM MAIN
0002 VAR
0003   iState      : UINT  := 0;
0004   bStartRequest : BOOL := TRUE;
0005 END_VAR
0006
0001 CASE iState OF
0002   0: (* Idle *)
0003     IF bStartRequest THEN
0004       bStartRequest := FALSE;
0005       iState      := 1;
0006     END_IF
```

Verbindung zur Informationstechnik

- Heute umfasst die AT deutlich mehr in Bezug auf die in IT kennengelernten Aspekte:
- Objektorientierung: Die Automatisierungstechnik kennt Klassen (z.B. Bausteine) und Instanzen.
- Zunehmend auch Einsatz von Hochsprachen in der AT (z.B. C++)
- Klassische IT-Kommunikationstechnologien & -protokolle finden Einzug in die AT → z.B. Ethernet-basierte Feldbusse
- Daten-, Informations- und physikalische Modelle spielen eine stetig zunehmende Rolle in AT-Systemen
- Voraussetzung für substanzIELLE Ergebnisse in der AT sind ein Verständnis sowie die notwendigen Kenntnisse in der Software-Entwicklung (Kennen und Können).
 - Daher in diesem Modul zunächst IT und dann AT (oder anders gesagt: Sie haben die besten Voraussetzungen für die AT bereits geschaffen)

Das HEUTE der Automatisierungstechnik

- Die Zukunft kommt ganz automatisch....

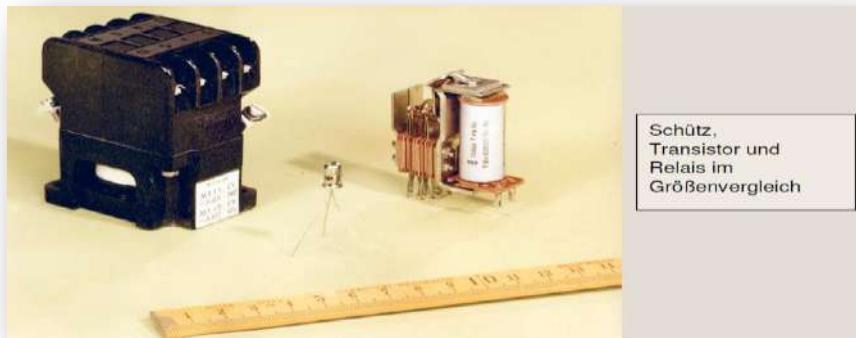


Aufgaben der Automatisierungstechnik

- Steuern und Regeln → Kernaufgabe
 - Messen → vgl. Sensorik, Signalverarbeitung, Messtechnik
 - Aktorik ansteuern/regeln → vgl. Aktoren, elektrische Maschinen, etc.
- Optimieren → z.B. den Energieverbrauch eines Prozesses senken
- Melden & Alarmieren → ereignisbasiert
- Kommunizieren → Daten über Netzwerke (Feldbussysteme) versenden
- Menschzentrierung → Visualisierung und Eingabe → Mensch-Maschine-Schnittstelle
- Dokumentieren & Archivieren → z.B. Zeitreihendaten in DB schreiben
- Sicherheit gewährleisten → Safety (Sicherheit für Mensch, Maschine und Umwelt) aber auch Security (z.B. Sicherheit vor Schadsoftware, Angriffen, etc.)

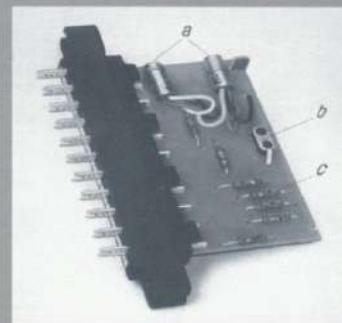
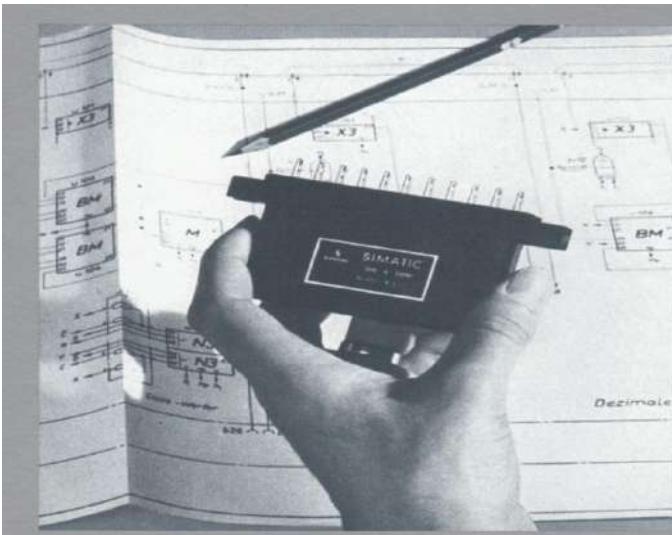
Historische Entwicklung der AT

- Die Entwicklung von Speicherprogrammierbaren Steuerungen (SPS) reicht bis in die 60er Jahre zurück, als noch verbindungsprogrammierte Relaischaltungen die Anlage steuerten. Hierbei ist das Steuerungsprogramm durch fest verdrahtete aufgebaut und dementsprechend unflexibel.
 - 1958 Transistoren in der Steuerungstechnik

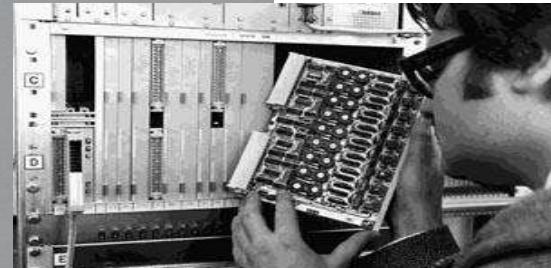


Historische Entwicklung der AT

- 1959: Erste Version der Siemens Simatic Familie basiert auf Germanium Transistoren Schaltungen
- Beispiel UND-Glied von 1959



So übersichtlich sah ein Simatic-Bauteil vor dem Vergießen mit Harz zu einem so genannten Simatic-Block aus (a = Transistoren, b = Gleichrichter, c = Widerstände).

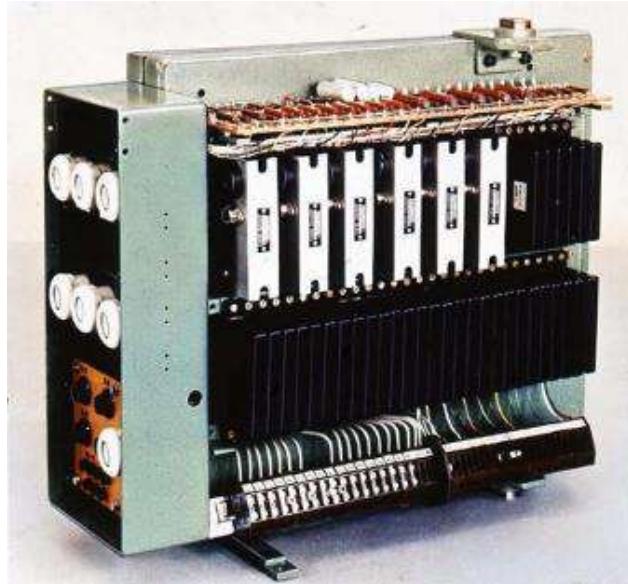


Ein Block der „Schwarzen Simatic“. In diesem konkreten Fall enthielt er zwei unabhängige „Und“-Verknüpfungen mit je drei Eingängen (1959).

Historische Entwicklung der AT

■ Simatic Steuerungen

ca. 0,6 m



1959 mit 6 E/A

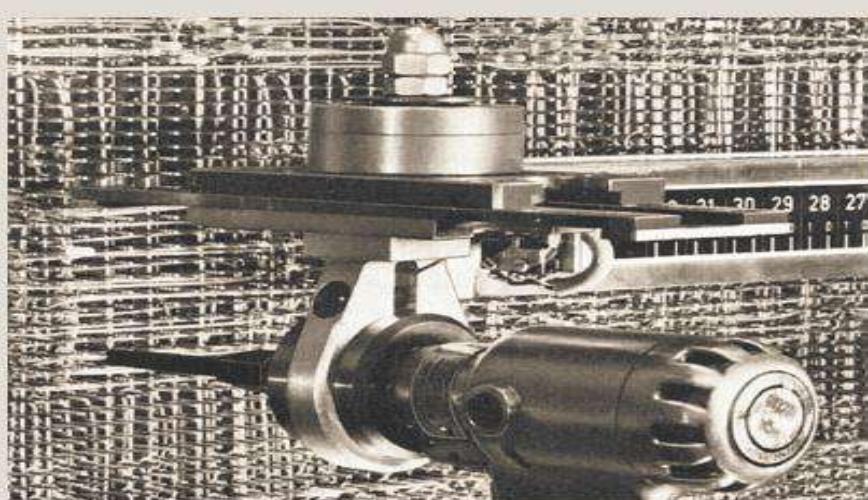
ca. 0,15 m



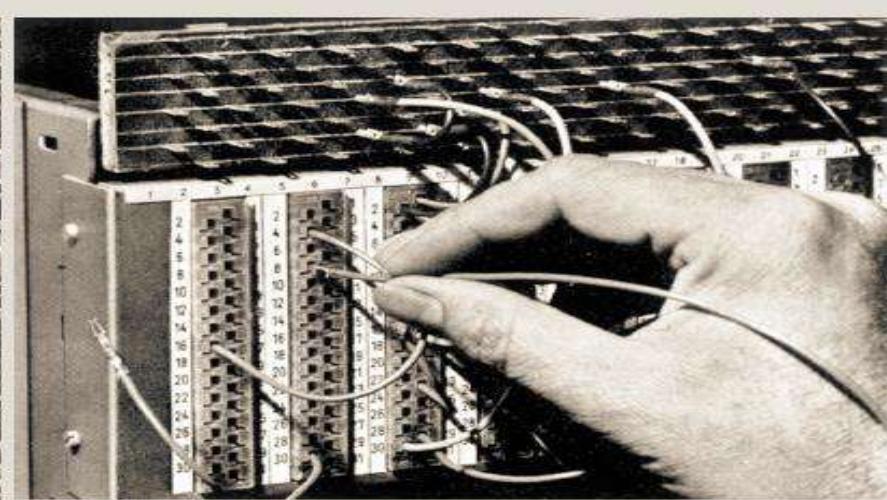
Heute mit > 50 E/A + Feldbus + Display...

Historische Entwicklung der AT

- Kundenspezifische Verdrahtung der Steuerungen



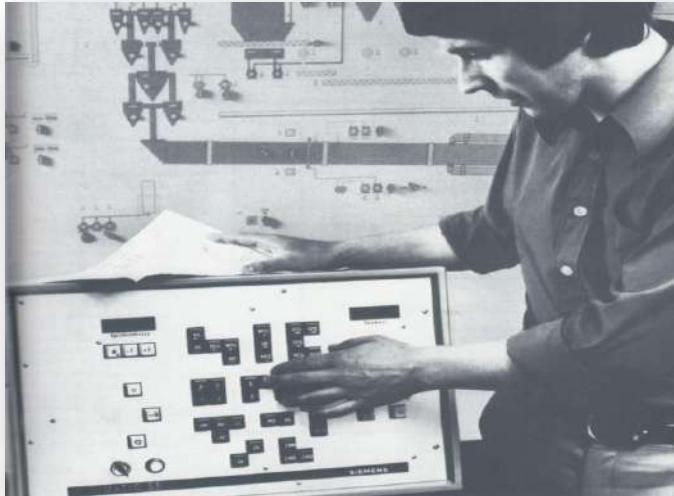
maschinell



per Hand

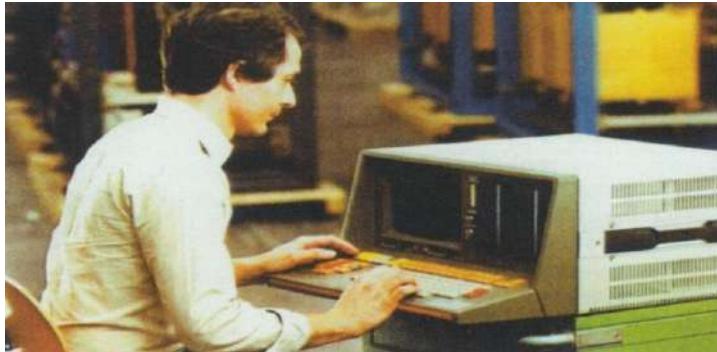
Historische Entwicklung der AT

- Weiterer Meilenstein für die Steuerungstechnik
 - 1968 Speicherprogrammierbare Steuerungen
 - Einführung des Begriffs: **Speicherprogrammierbare Steuerung (SPS)**
 - Erstes SPS-Programmiergerät PGO – Gewicht 100 kg



Historische Entwicklung der AT

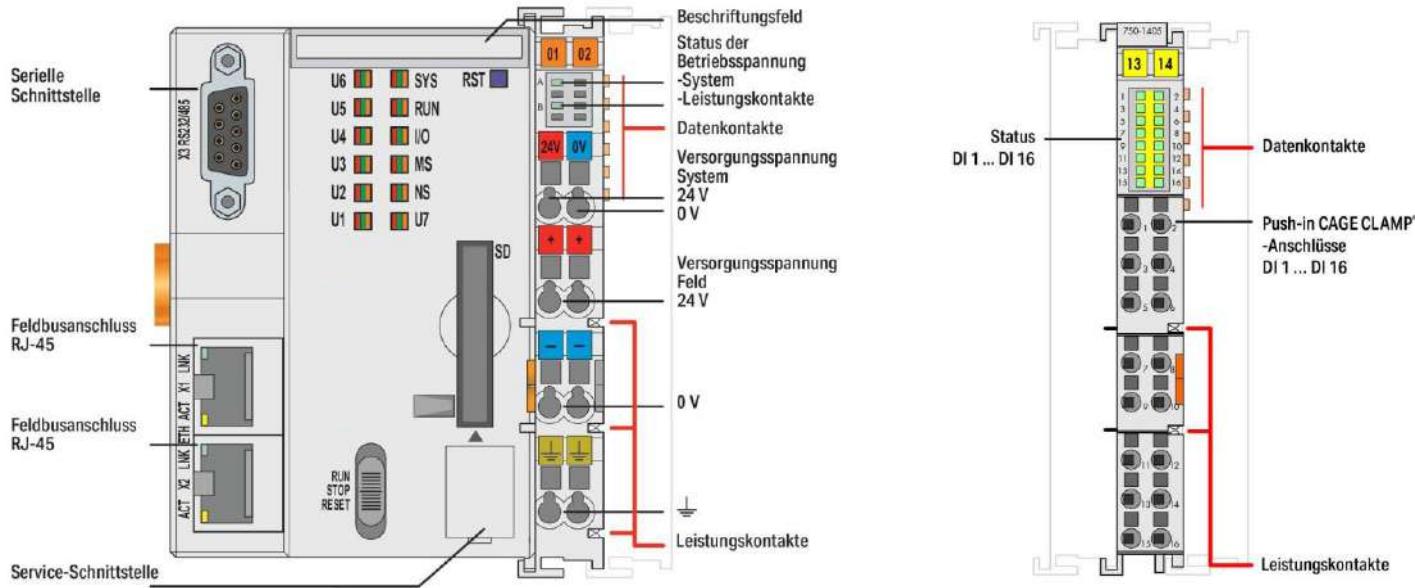
- Weiterer Meilenstein für die Steuerungstechnik
 - 1980: Erstes S5-Programmiergerät von 1980: Transport mit Spezialkoffer auf Rädern



S5 Programmiergeräte von 1990

Heutige Steuerung – Speicherprogrammierbare Steuerung (SPS)

■ Speicherprogrammierbare Steuerung (SPS)



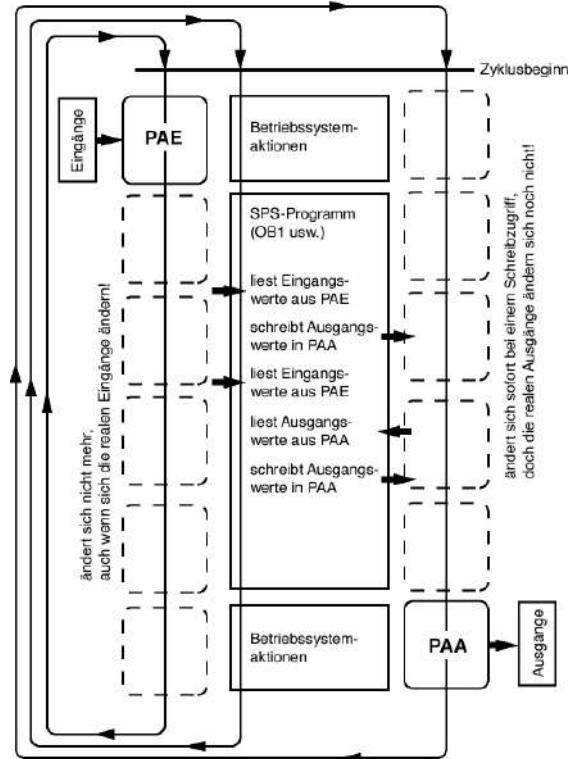
Speicherprogrammierbare Steuerung

- Eine Speicherprogrammierbare Steuerung (SPS, engl. Programmable Logic Controller, PLC) ist eine ICS (Industrial Control System) - Komponente.
- Sie übernimmt Steuerungs- und Regelaufgaben in der Betriebstechnik (engl. Operational Technology, OT).
- Die Grenzen zwischen verschiedenen Geräteklassen und Bauformen sind heute fließend: Industrial PC, Remote Terminal Unit, Edge-Gateway, Industrial Controller → prinzipiell gilt die in Kapitel 1 und 7 kennengelernte Struktur eines Eingebetteten Systems.
- So wird der Begriff SPS heute als Überbegriff für eine Steuerungshardware im industriellen Kontext eingesetzt.

Arbeitsweise einer SPS

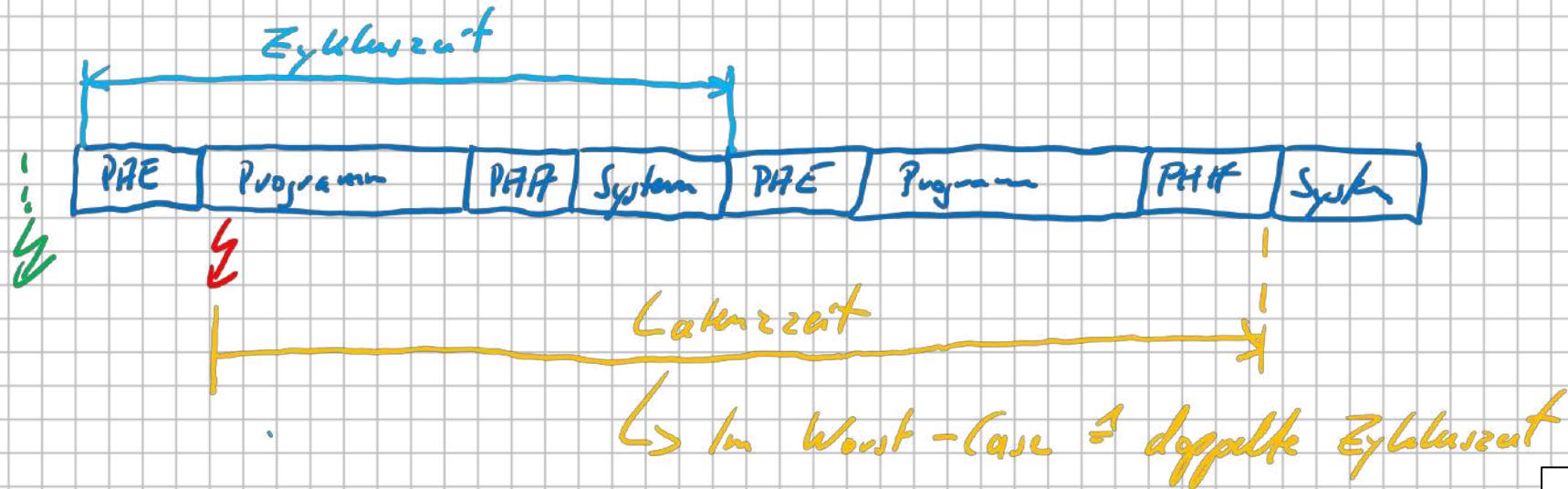
Zyklische Arbeitsweise

- Zu Beginn eines Zyklus wird der Zustand der SPS-Eingänge in das PAE (Prozessabbild der Eingänge) übertragen. Danach bleibt eine Änderung der realen Eingänge für das PAE ohne Folgen.
- Im Anschluss wird das Steuerungsprogramm seriell, also in Reihenfolge der Befehle abgearbeitet. Eingangswerte kommen aus dem PAE, Zuweisungen an Ausgänge erfolgen an das PAA (Prozessabbild der Ausgänge).
- Schließlich wird der Zustand der gespeicherten Variablen der Ausgänge (das PAA) auf die externen SPS-Ausgänge übertragen. Erst jetzt ändern sich die Zustände der Ausgänge.



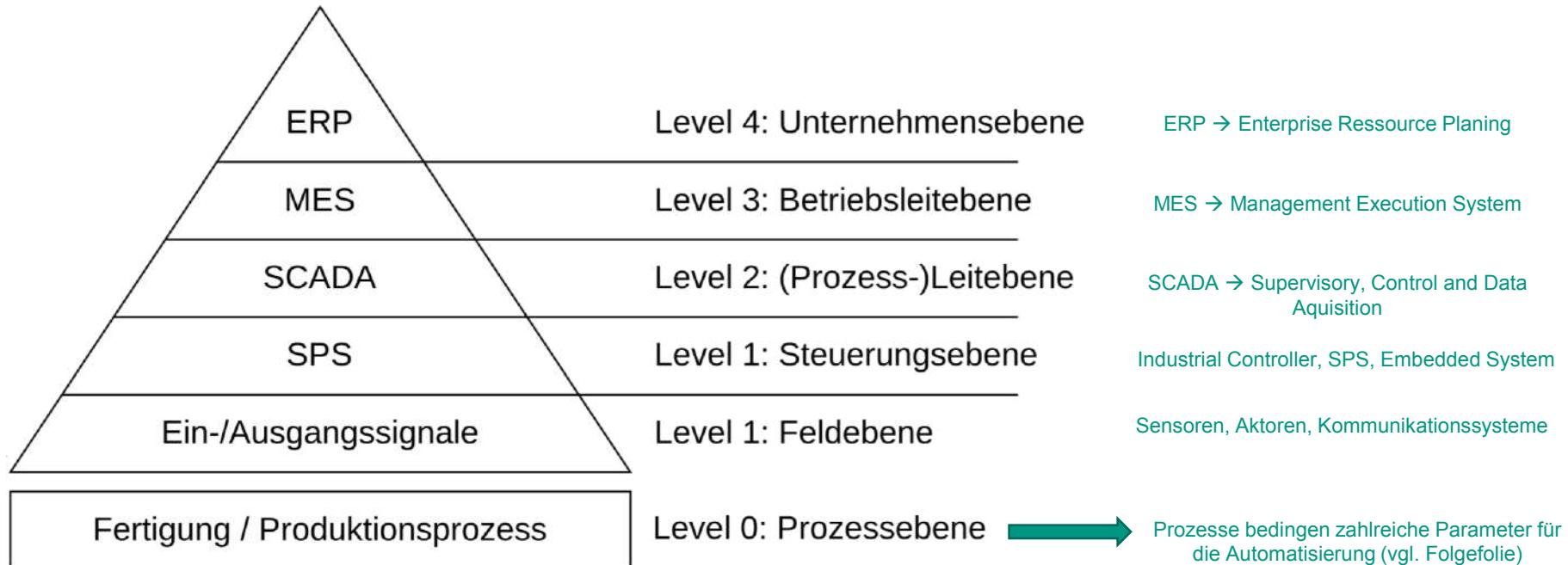
Arbeitsweise einer SPS

- Zykluszeit, Eingangsverzögerung, Latenzzeit



Grundlegende Architektur von AT-Systemen

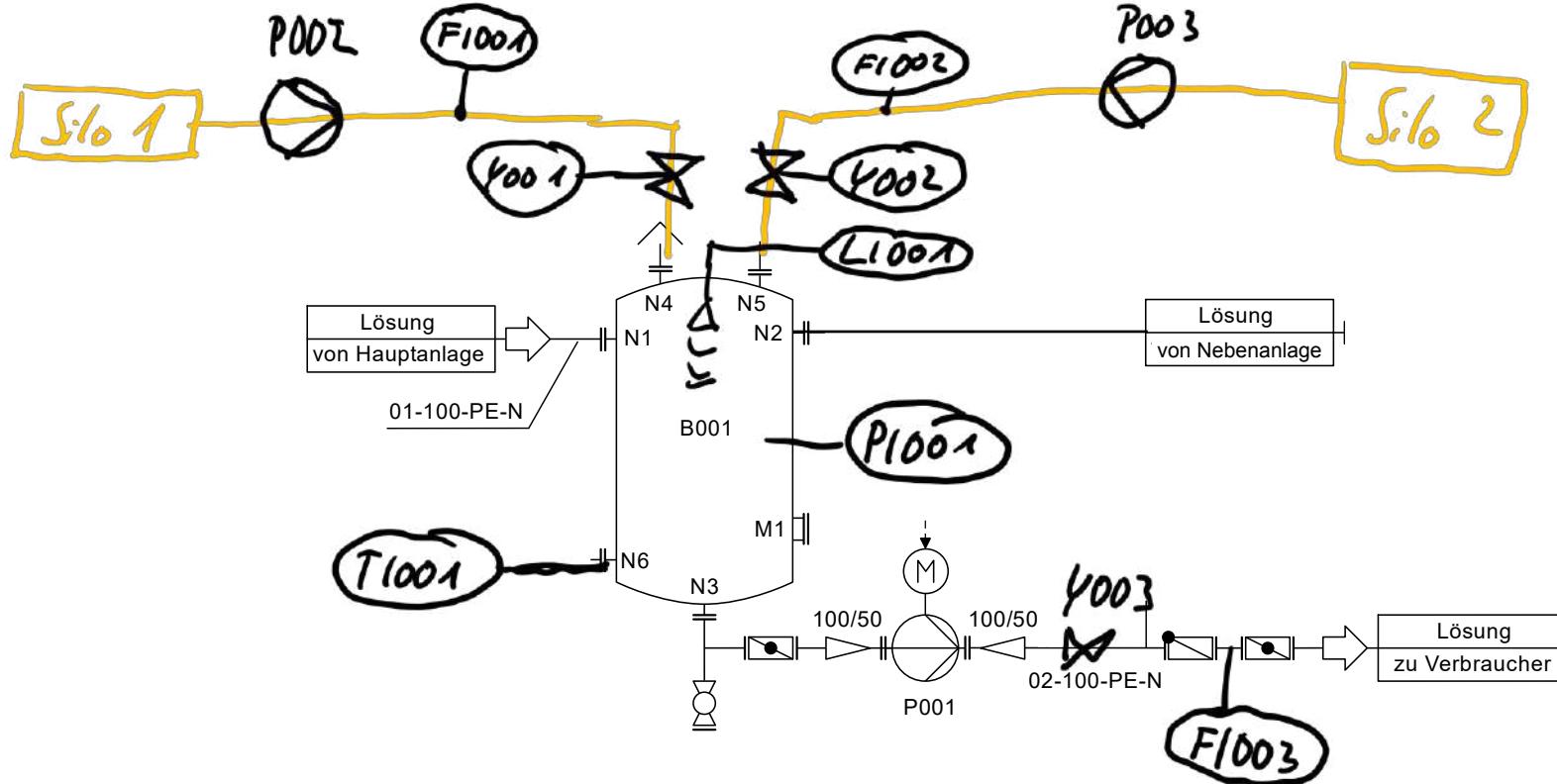
■ Die AT-Pyramide



Prozessklassifizierung

- Welche Prozesse werden automatisiert?
- Kontinuierliche Prozesse
 - Ständiger Zu- und Abfluss von Material bzw. Energie
 - Beispiel: Kraftwerk, Konti-Anlage in der chemischen Industrie, Raffinerie
- Diskontinuierliche Prozesse
 - Stückprozesse
 - Beispiel: Verarbeitung und Transport von identifizierbaren Einzelobjekten zu aufeinanderfolgenden diskreten Zeitpunkten (Koffertransportband, Automobilherstellung)
 - Chargenprozesse
 - Beispiel: Der Einsatz von Stoffmengen erfolgt zu unterschiedlichen diskreten Zeitpunkten (Bierherstellung, Pharmazie)

Einführungsbeispiel



Einschub: Rückblick Automatentheorie

- Schaltnetze - ermöglichen die Bildung von Ausgangsgrößen unmittelbar aus Eingangsgrößen.
- Erweiterung zum Schaltwerk:

Verallgemeinerung des Problems:

- endliches Eingabealphabet: $E = \{E_1, E_2, \dots, E_g, \dots, E_u\}$
- endliches Ausgabealphabet: $A = \{A_1, A_2, \dots, A_h, \dots, A_v\}$

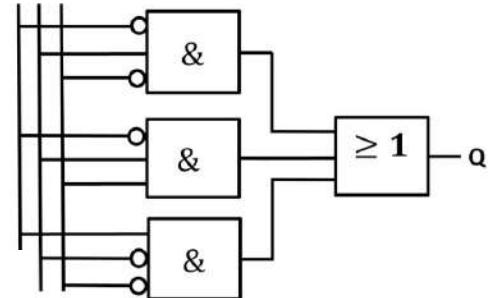
➤ das **neue Konzept** beschrieben durch eine **Einheit AT**, bei der eine **zeitliche Folge** von **Elementen** des **Eingabealphabets F_E** in eine **zeitliche Folge** von **Elementen** des **Ausgabealphabets F_A** abgebildet wird:

$$F_E \Rightarrow \boxed{\text{AT}} \Rightarrow F_A$$

- zur **Unterscheidung zeitlicher Reihung**: Einführung eines **Ordnungsindizes v**

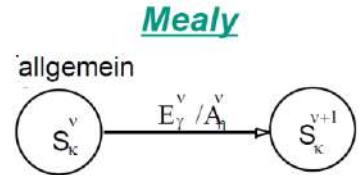
$$F_E = E_g^v \ E_g^{v-1} \ E_g^{v-2} \ \dots$$

$$F_A = A_h^v \ A_h^{v-1} \ A_h^{v-2} \ \dots$$

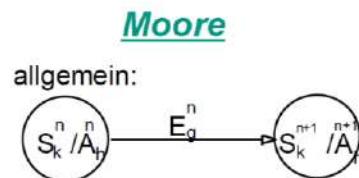
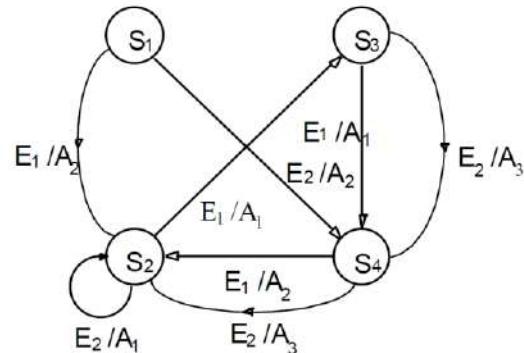


Einschub: Rückblick Automatentheorie

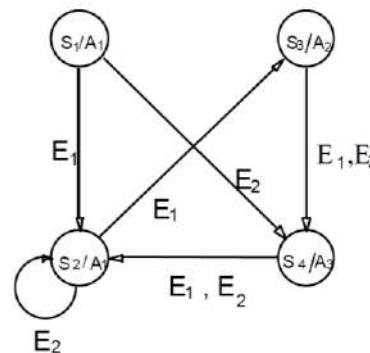
- endliche, diskrete und deterministische Automaten



Beispiel:

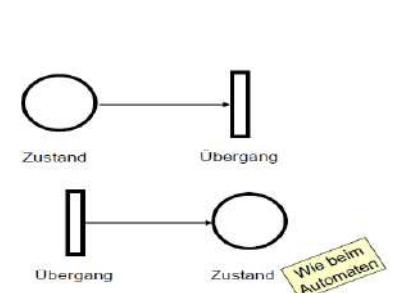


Beispiel:



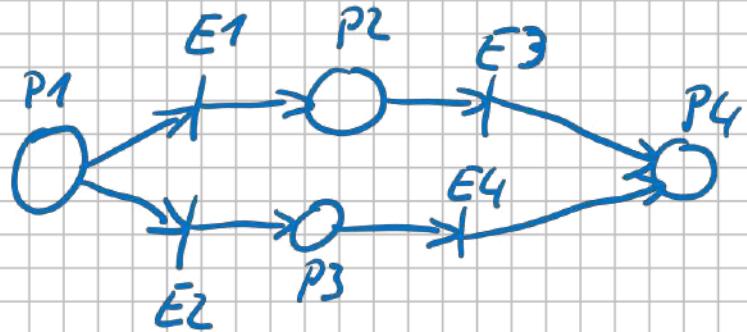
Übergang zu Petri-Netzen

- Grenzen von E/A-Automaten und Einführung von Petri-Netzen
 - Für eine Parallelverzweigung gibt es keine Entsprechung im Automaten-Modell.
 - Versucht man trotzdem, die verschiedenen Gesamtzustände, die ein System mit Parallelverzweigungen haben kann, mit Automaten nachzubilden, so wird der Automat groß und unübersichtlich.
 - Lösung: Einführung von Petri-Netzen
 - Elemente eines Petri-Netzes:
 - Stelle → entspricht Zustand im Automat 
 - Transition → entspricht einem Übergang im Automat 
 - Kante → Zusammenhang zwischen Stelle und Transition 



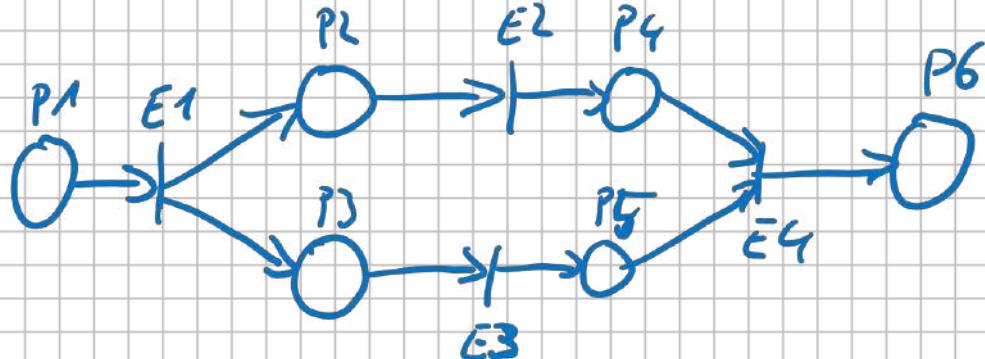
Einführung von Petri-Netzen

- Zu einer Stelle können mehrere Nachfolgetransitionen gehören.
 - Vgl. alternative Zustandsübergänge
- Zu einer Stelle können mehrere Vorgängertransitionen gehören
 - Alternative Zustandsübergänge führen zum selben Zustand



Einführung von Petri-Netzen

- Zu einer Transition können mehrere Nachfolgestellen gehören.
 - Mehrere unabhängige Zustände sind Folge eines Zustands
- Zu einer Transition können mehrere Vorgängerstellen gehören
 - Mehrere unabhängige Stellen sind Bedingung für einen Zustandsübergang



Einführung von Petri-Netzen

- Petri-Netze können dynamische Vorgänge in einem System abbilden. Hierzu werden Marken eingesetzt, welche zwischen den Zuständen bewegt werden. Für diese Marken gelten Schaltregeln. Am Beispiel eines Bedingungs-Ereignis-Netzwerkes gelten hier folgende Regeln.
- 1. Schaltregel (Konzessionsregel): Eine Transition ist schaltbereit, wenn
 - alle ihre Vor-Stellen markiert sind und
 - alle ihre Nach-Stellen unmarkiert sind.

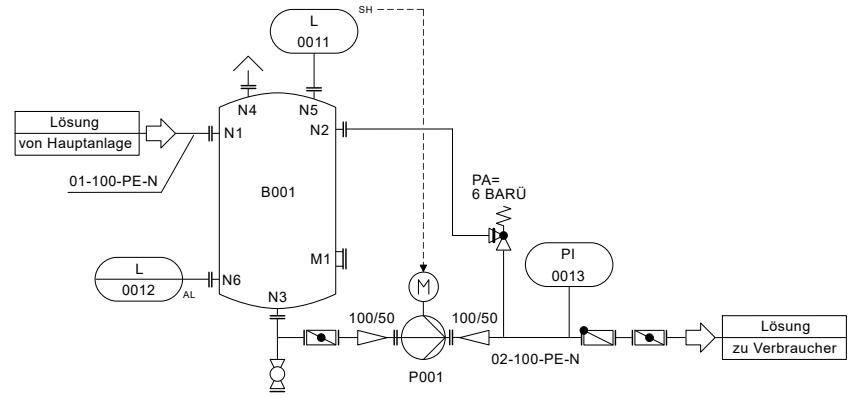
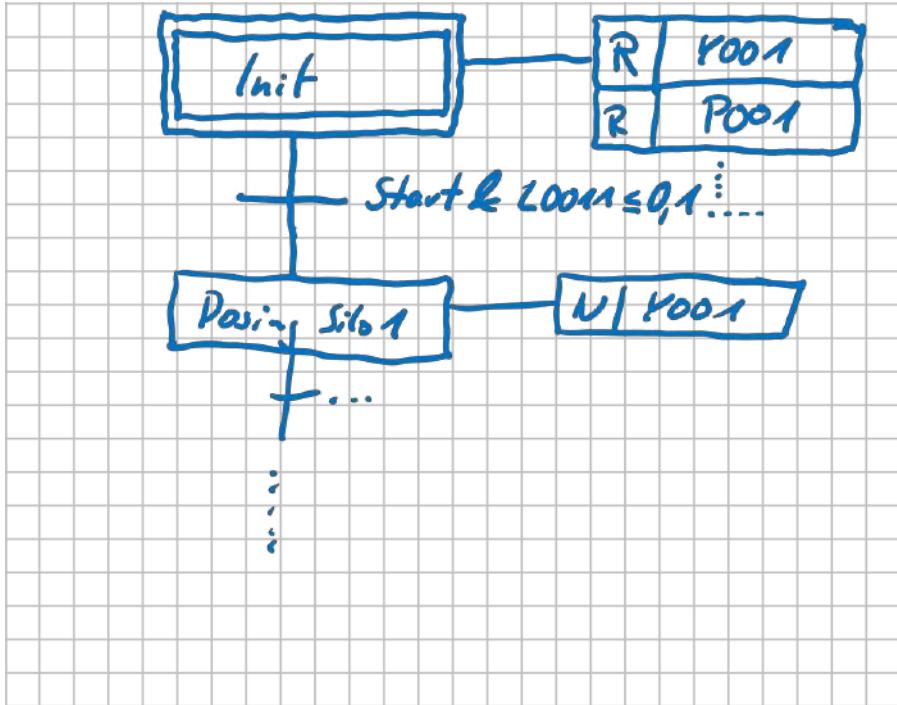


Einführung von Petri-Netzen

- Petri-Netze können dynamische Vorgänge in einem System abbilden. Hierzu werden Marken eingesetzt, welche zwischen den Zuständen bewegt werden. Für diese Marken gelten Schaltregeln. Am Beispiel eines Bedingungs-Ereignis-Netzwerkes gelten hier folgende Regeln.
- 2. Schaltregel (Markenflussregel): Wenn eine Transition T_i schaltet,
 - wird allen ihren Vor-Stellen die Marke entzogen und
 - werden alle ihre Nach-Stellen markiert.

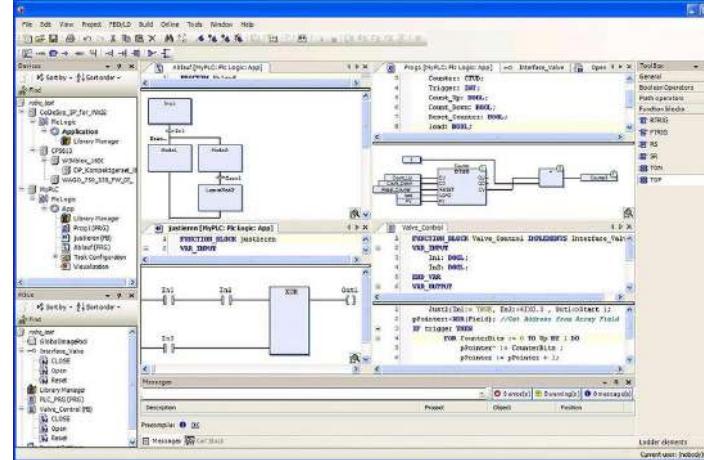
Siehe Folie 40

Einführungsbeispiel - Schrittkette



Übungen zu AT

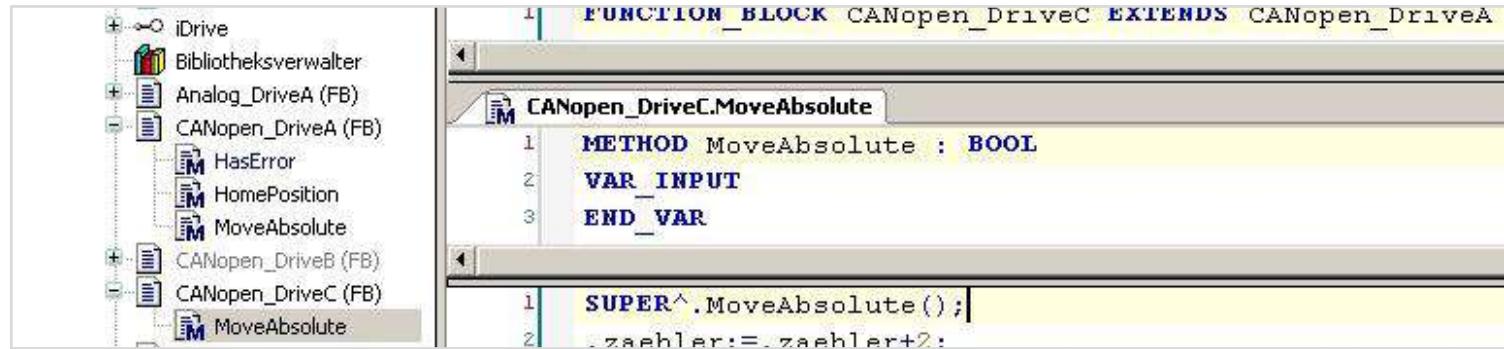
- Alle Übungen im AT-Teil können ohne Tool-Unterstützung durchgeführt werden.
- Freiwillig: für die Übungen zur AT verwenden wir das kostenlose Engineering-System CODESYS – Erhältlich unter: [CODESYS Store International](https://www.codesys.com/store/international) | [CODESYS Store International](https://www.codesys.com/store/international)
- IEC 61131-Sprachen
- Visualisierung
- Sprachelemente
- Strukturelemente
- Virtuelle Steuerung und Testmodus



Informations- und Automatisierungstechnik

Kapitel 8: Steuerungskonfiguration – Teil 1

Univ.-Prof. Dr.-Ing. Mike Barth



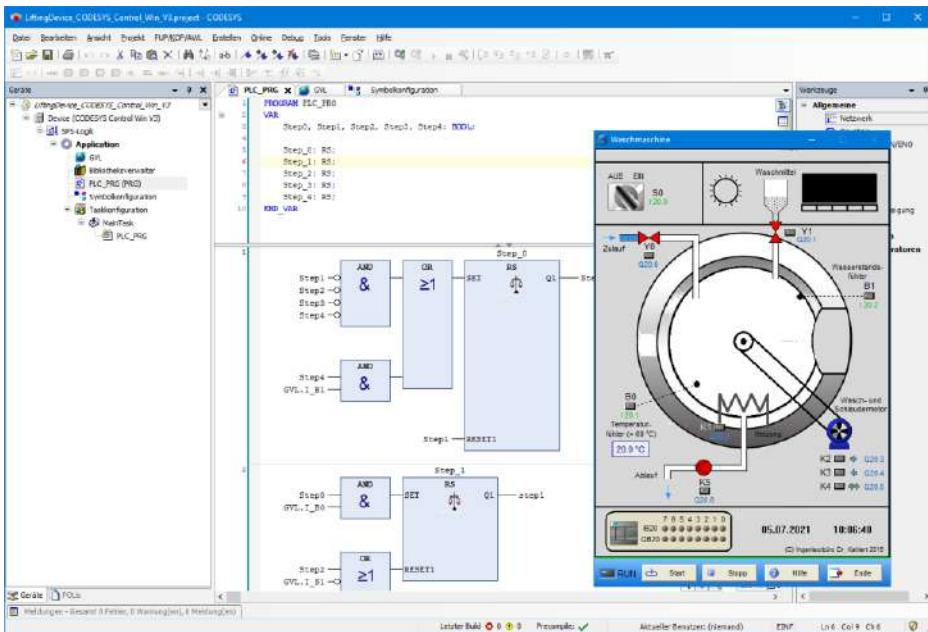
The screenshot shows a SIMATIC Manager interface. On the left, a tree view displays function blocks under a project named 'iDrive'. The 'CANopen_DriveA (FB)' node is expanded, showing sub-methods: 'HasError', 'HomePosition', and 'MoveAbsolute'. The 'CANopen_DriveC (FB)' node is also expanded, showing another 'MoveAbsolute' method. On the right, the code editor displays the source code for the 'MoveAbsolute' method of the 'CANopen_DriveC' function block. The code is written in a structured text-like language:

```
FUNCTION_BLOCK CANopen_DriveC EXTENDS CANopen_DriveA
  METHOD MoveAbsolute : BOOL
    VAR_INPUT
    END_VAR

  SUPER^.MoveAbsolute();
  .zaehler:=.zaehler+2;
```

Steuerungsimplementierung mit IEC 61131-3

- Dieses Kapitel wird mit mehreren Beispielen und Live-Demos ergänzt, die in CoDeSys (während Vorlesung & Übung) implementiert werden.
- Die Beispiele sind Teil der Erläuterung und werden nicht vollständig in diesem Skriptteil aufgeführt.
- Es wird dringend dazu angehalten, die Beispiele selbstständig im Nachgang zu implementieren und dadurch nachzuvollziehen.

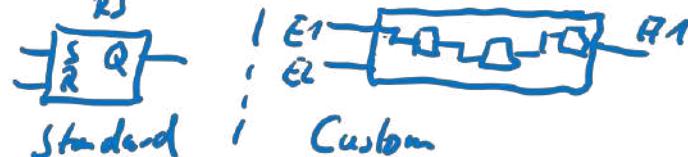


Programmstruktur in Industrie-Steuerungen

- Jede Steuerung besteht aus sogenannten Programmorganisationseinheiten (POE; engl: POU).
- Es gibt drei Arten:

① ↳ Programm → selbstständige Einheit die in einer Task läuft
↳ vgl. IT (zur C++ Programm)

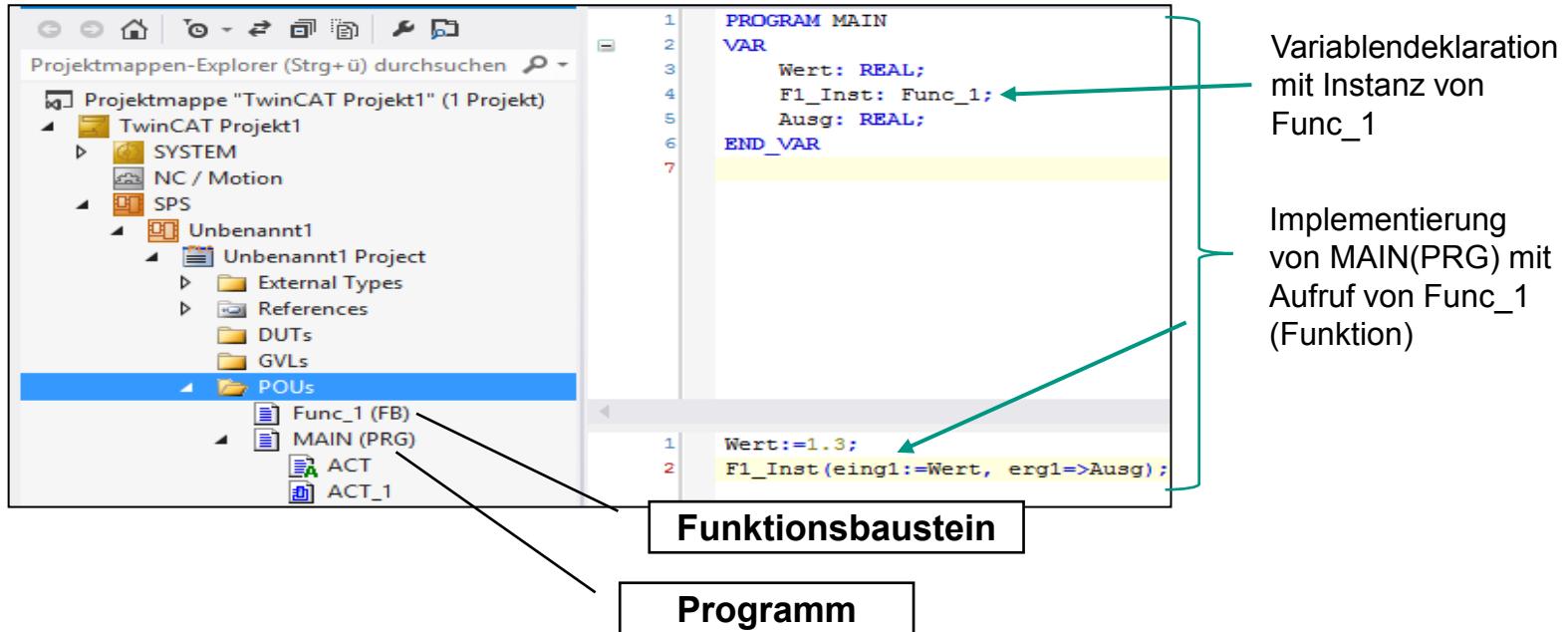
② ↳ Funktionsbaustein → Block-Box mit definierten Ein- & Ausgängen



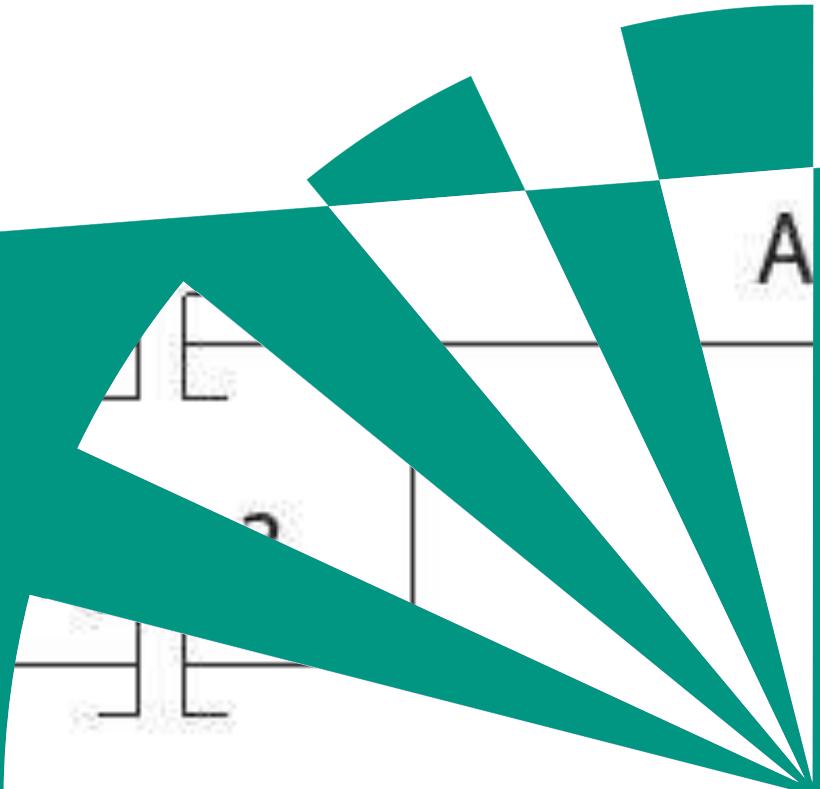
③ ↳ Funktionen add(a,b)
↳ Strukturierter Text

Programmstruktur in Industrie-Steuerungen

■ Beispiel für den Aufbau eines SPS-Projektes



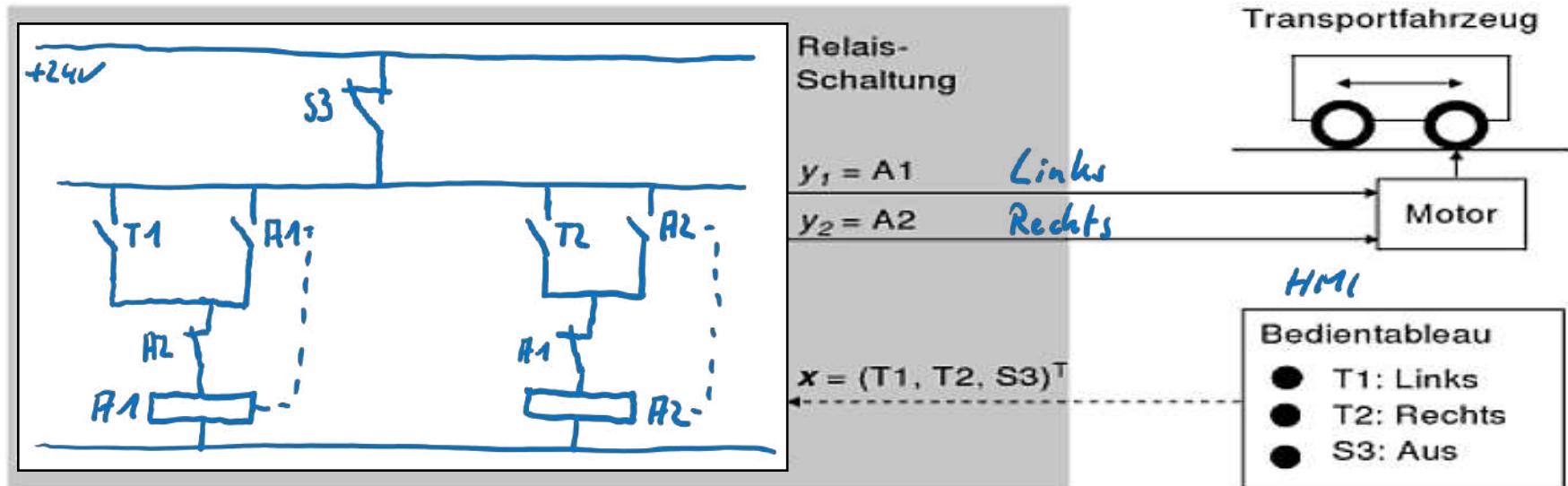
Kontaktplan



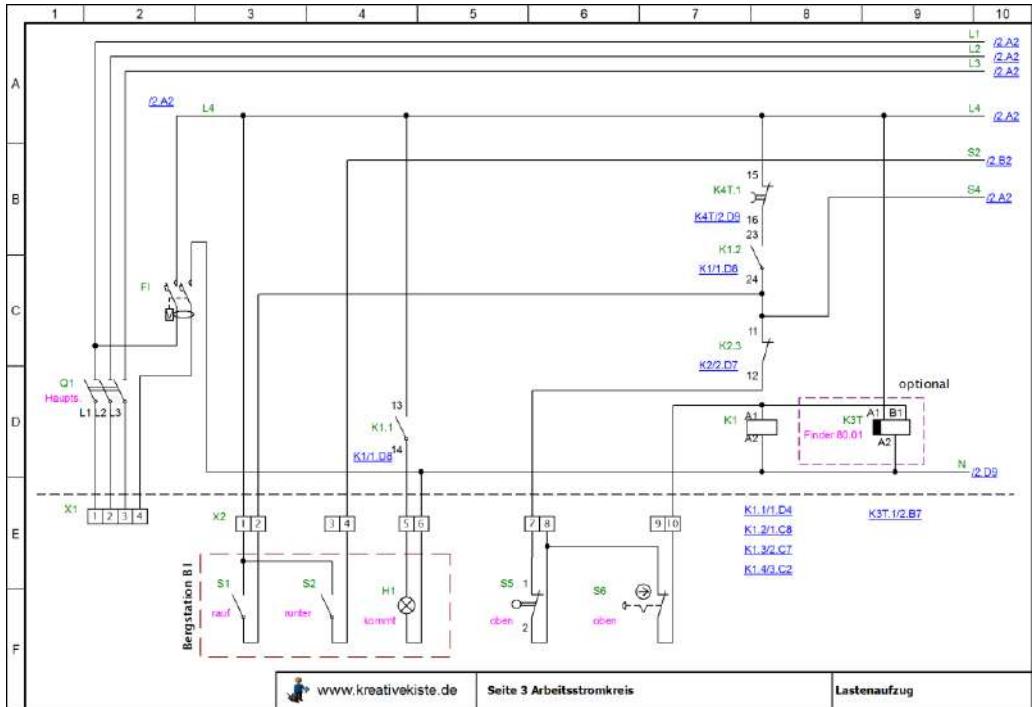
Kontaktplan → Ursprung

■ Beispiel: Relaisschaltung

- Die Räder eines Transportfahrzeugs werden jeweils durch einen Motor mit veränderbarer Drehrichtung (Links-/Rechtslauf) angetrieben. An einem Bedientableau kann der Bediener vor Ort den Wagen über die Taster T1, T2 nach links bzw. rechts bewegen und durch Betätigen der Taste S3 anhalten.



Stromlaufplan als Basis für Relaissteuerungen



IEC 61131-3: Kontaktplan

- Historisch bedingt sind die ersten SPS-Programmiersprachen an fest verdrahteten Lösungen orientiert (z.B. Kontaktplan und Funktionsplan)



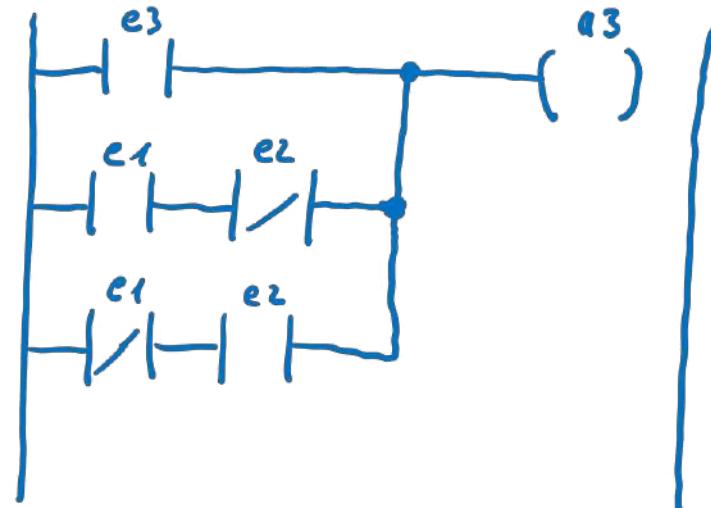
Der Wirkungszusammenhang wird durch den fiktiven Stromfluss von links nach rechts beschrieben. Bei mehreren Netzwerken erfolgt die Bearbeitung von oben nach unten.

- UND-Verknüpfung durch *Reihenschaltung*
- ODER-Verknüpfung durch *Parallelschaltung*

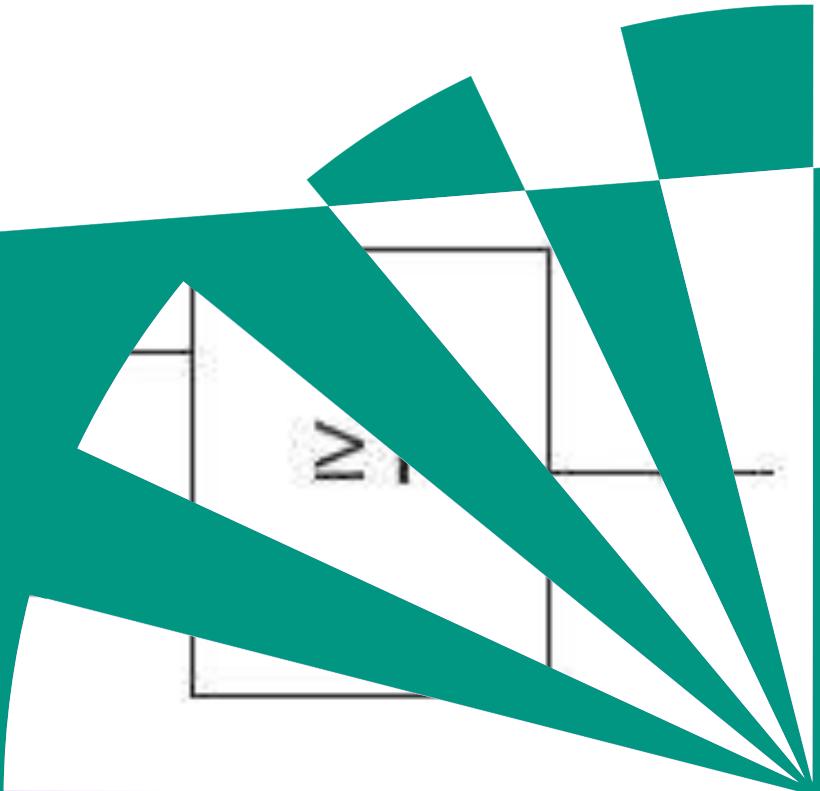
IEC 61131-3: Kontaktplan

- Kontaktplan (KOP) besteht aus:
 - der linken und rechten Stromschiene,
 - Verbindungen,
 - Kontakte und Spulen,
 - Elemente zur Ausführungssteuerung und
 - Konnektoren.
- Vertikale Verbindungen werden im Kontaktplan als logische ODER-Verknüpfung aller linken Signalwerte interpretiert und das Ergebnis an alle rechten Verbindungen weitergeleitet:

$$a_3 = e_3 \vee (e_1 \wedge \bar{e}_2) \vee (\bar{e}_1 \wedge e_2)$$

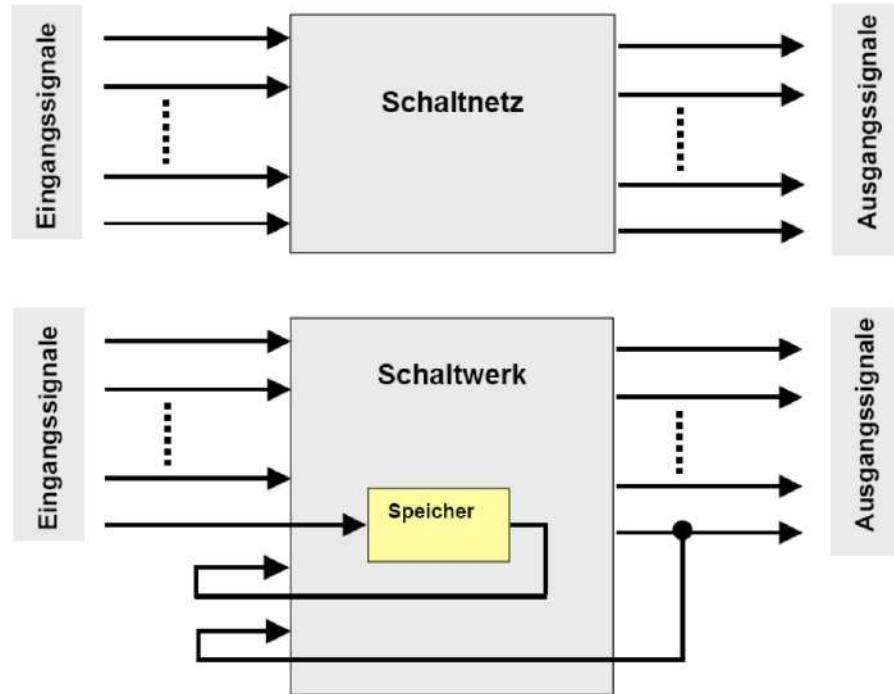


Funktionsbausteinsprache



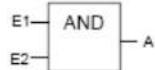
IEC 61131-3: Funktionsbausteinsprache

- Die Funktionsbausteinsprache (auch Funktionsplan genannt) ist eine grafische Programmiersprache, die aus Funktionsblöcken besteht und an die klassische Digitaltechnik angelehnt ist.
- Anlehnung an die Digitaltechnik
- Es können Schaltwerke und Schaltnetze konfiguriert werden
- Die Darstellung erfolgt über Netzwerke, die sequentiell abgearbeitet werden.
- Die Netzwerke helfen bei der Programm- und Ablaufstrukturierung
- Grund-Bausteine sind bekannt: UND, ODER, RS-FF, TON, TOF....

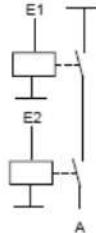


Einschub: Logische Grundfunktionen

UND (Konjunktion)

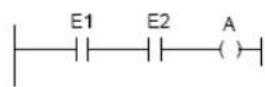


LD Eingang_1
AND Eingang_2
ST Ausgang

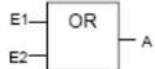


E1	E2	A
0	0	0
0	1	0
1	0	0
1	1	1

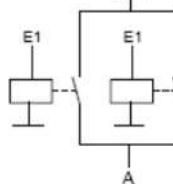
$$A = E1 \cdot E2$$



ODER (Disjunktion)

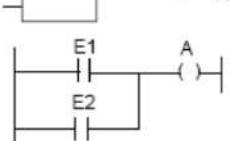


LD Eingang_1
OR Eingang_2
ST Ausgang

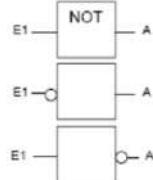


E1	E2	A
0	0	0
0	1	1
1	0	1
1	1	1

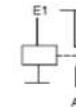
$$A = E1 + E2$$



Nicht (NEGATION)



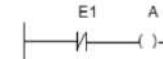
LD Eingang_1
NOT
ST Ausgang
oder
LDN E1
ST A



E1	A
0	1
1	0

$$A = \overline{E1}$$

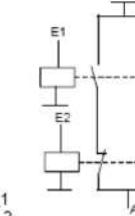
Step7
U "Eingang_1"
NOT
= "Ausgang"
oder
UN "Eingang_1"
= "Ausgang"



EXKLUSIV-ODER

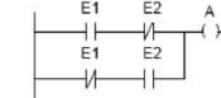
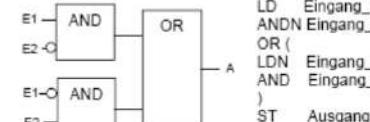


XOR mit Grundoperationen
ausgeführt



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

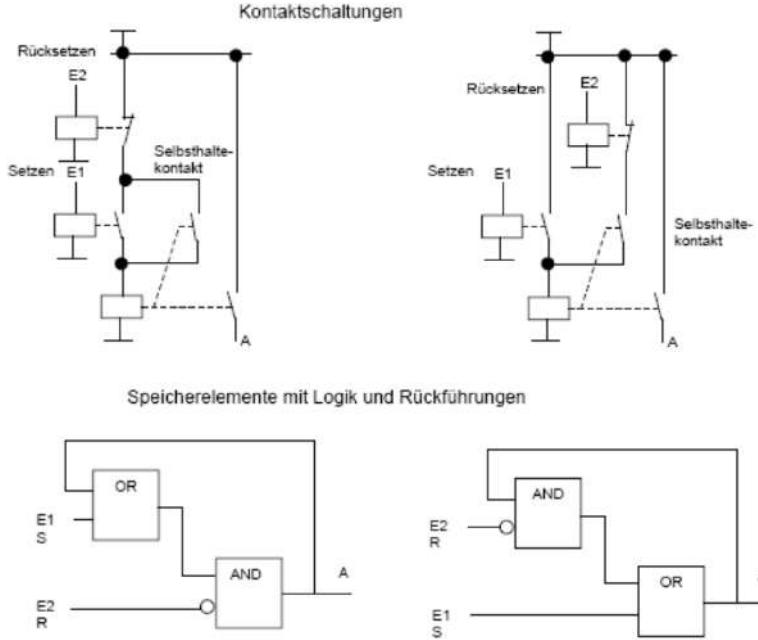
$$A = E1 \cdot \overline{E2} + \overline{E1} \cdot E2$$



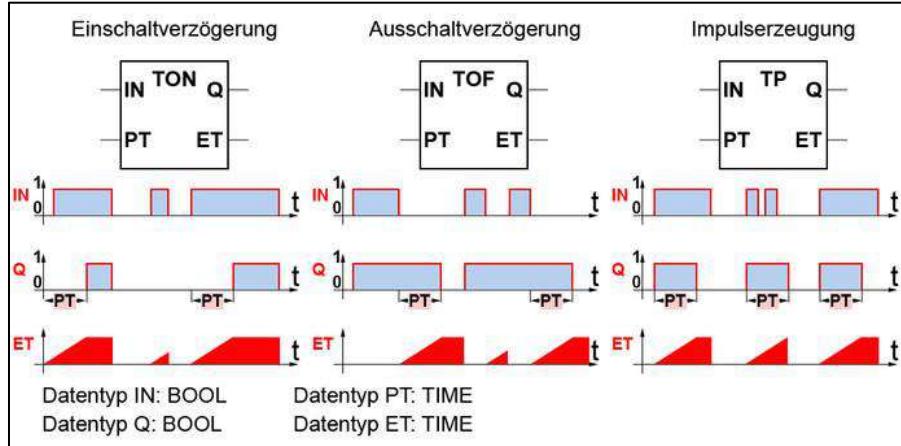
Einschub: Speicherelemente

Realisierung von Speicherelementen (Flip-Flop):

Dominierend Rücksetzen (Dominierend AUS) Dominierend Setzen (Dominierend EIN)

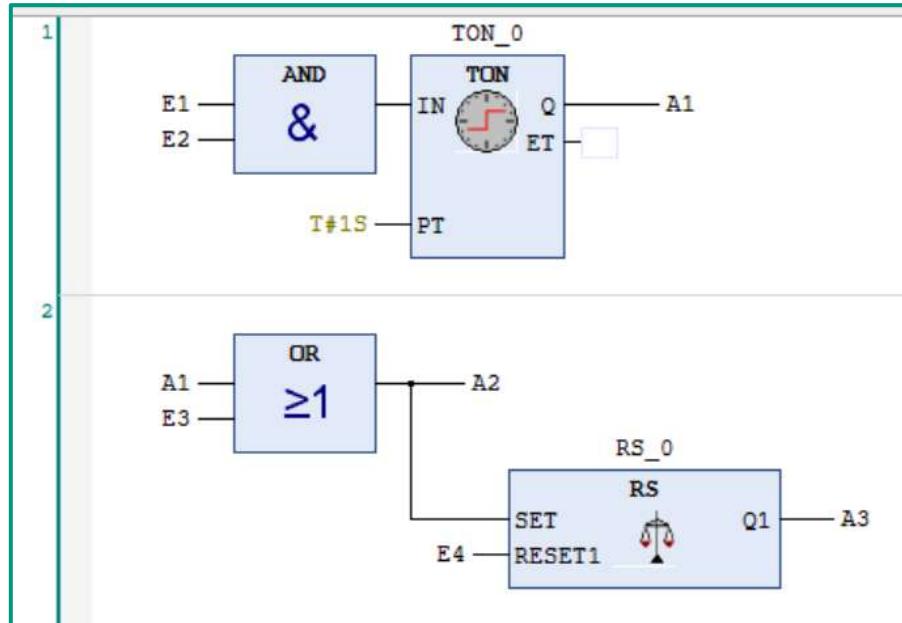


Darstellung der Flip-Flops nach IEC: Die Zuordnung der Ziffer 1 bestimmt dominierendes Setzen oder Rücksetzen

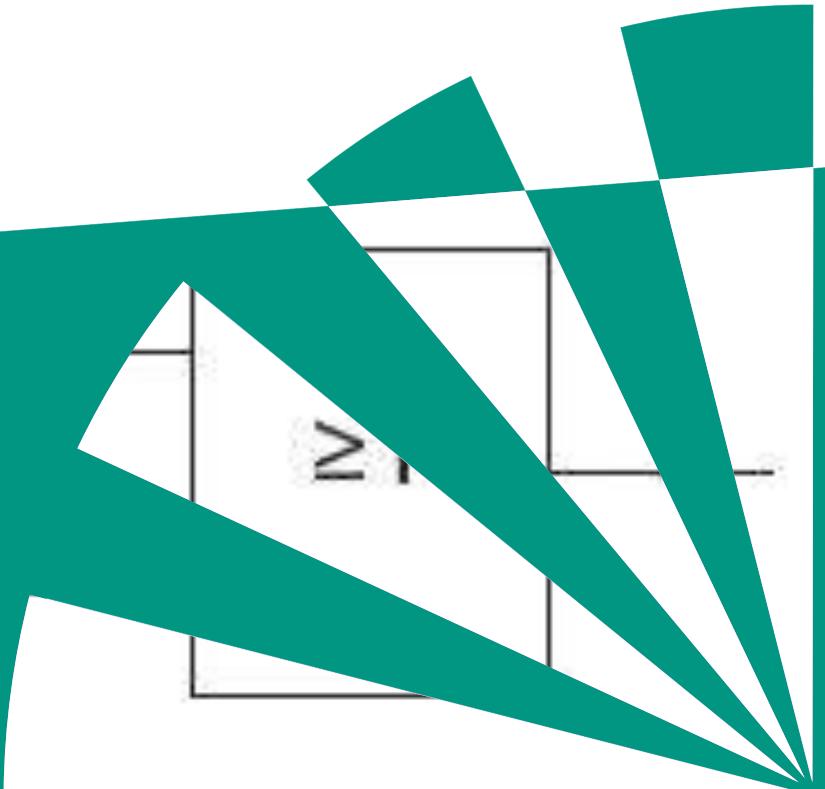


IEC 61131-3: Funktionsbausteinsprache

■ Beispiel für FBS in CoDeSys

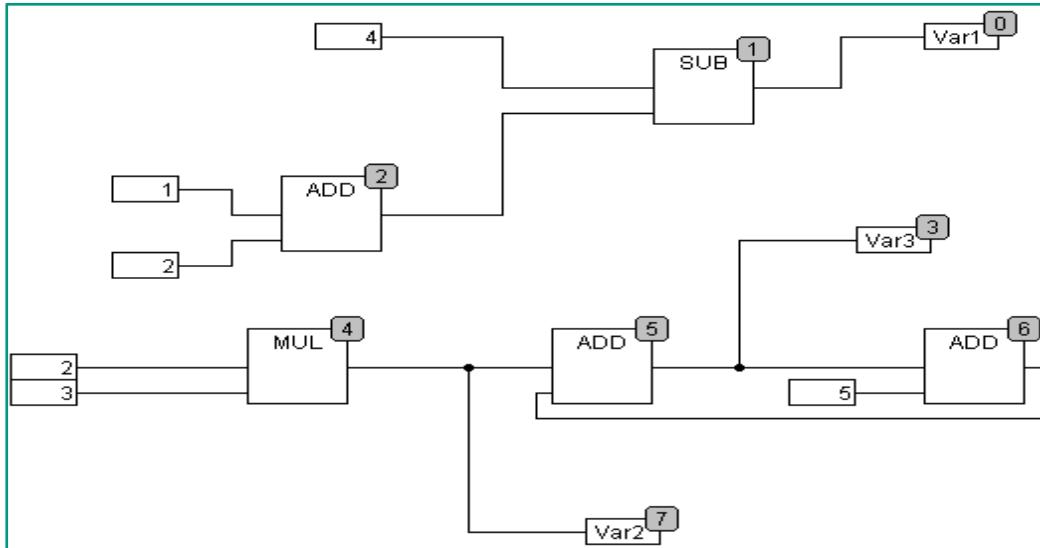


Continuous Function Chart



IEC 61131-3: Continuous Function Chart (CFC)

- Rein grafische Programmiersprache
 - CFC kann als eine Erweiterung der Funktionsbausteinsprache betrachtet werden, in der keine strikte zeilenweise Abarbeitung von links oben nach rechts unten erzwungen wird, die Funktionsblöcke frei positionierbar sind.
 - Beispiel:



Strukturierter Text



The diagram illustrates the execution flow of structured text logic. It shows a main loop structure with nested IF statements. The flow starts at the top left, enters the outermost IF block labeled "IF", which contains two parallel blocks: "Anweisung1;" and "Anweisung2;". These parallel blocks then merge back into the main loop body. From there, the flow enters the innermost IF block labeled "IF", which contains the single block "Anweisung1;". This inner block then merges back into the main loop body. Finally, the flow exits the main loop via the "END_IF;" statement at the bottom right.

```
IF
  Anweisung1;
  Anweisung2;
END_IF;
IF
  Anweisung1;
END_IF;
END_IF;
```

IEC 61131-3: Strukturierter Text

- Strukturierter Text (ST) hat Ähnlichkeiten zu einer höheren Programmiersprache (ähnliche C++).
- Wie bei allen höheren Programmiersprachen gilt:
 - Kompaktere und übersichtlichere Programmierung.
 - Eignung für Personen mit Vorkenntnissen in einer Programmiersprache
→ Kontrollstrukturen, Variablen-deklarationen, etc.
- Strukturierter Text ist nicht zeilenbasiert, sondern eine Abfolge von durch Strichpunkt getrennten Anweisungen (Zeilenumbruch spielt keine Rolle).

```
IF (MASCHINE_EINGESCHALTET = TRUE) THEN
    SOLLPOSITION := SOLLPOSITION + 100;
    AUSGANG1 := EINGANG1 AND EINGANG2;
ELSE
    AUSGANG1 := FALSE;
END_IF;
```

IEC 61131-3: Strukturierter Text

■ Beispiel – siehe CoDeSys

```
1 PROGRAM POU
2 VAR
3 MotorOverload : ARRAY[1..3] OF BOOL;
4 MotorOverCurrent : ARRAY[1..3] OF BOOL;
5 MotorPBStart : ARRAY[1..3] OF BOOL;
6 MotorPBStop : ARRAY[1..3] OF BOOL;
7 MotorCoil : ARRAY[1..3] OF BOOL;
8 i : INT;
9
10 END_VAR
11

1
2 FOR i:=1 TO 3 BY 1 DO
3
4 IF MotorPBStop[i] OR MotorOverload[i] OR MotorOverCurrent[i] THEN
5     MotorCoil[i]:=0;
6 ELSIF MotorPBStart[i] THEN
7     MotorCoil[i]:=1;
8 END_IF
9 END_FOR
```

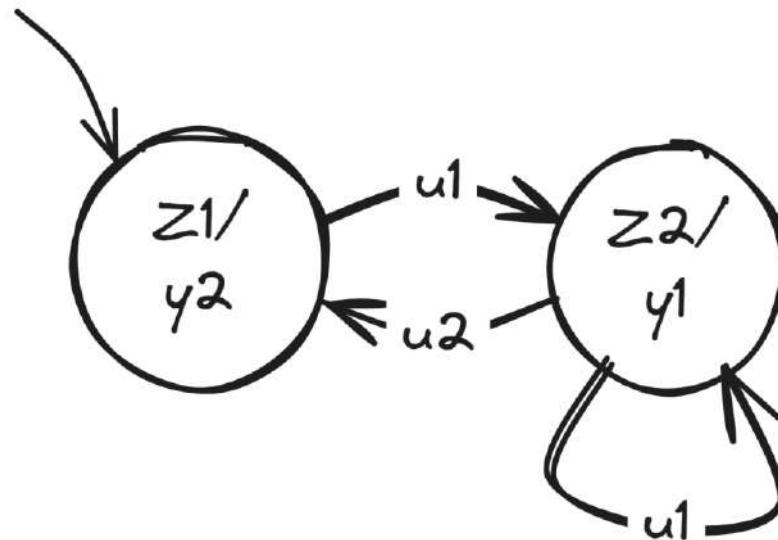
IEC 61131-3: Strukturierter Text

■ Beispiel – siehe CoDeSys

Operator	Beschreibung	Beispiel
()	Klammerung	$(1*2)+(3*4) (=14)$
	Funktionsaufruf	
-	Negation	-3
NOT	Komplement	NOT TRUE (=FALSE)
**	Potenzierung	$10 ** 2 (=100)$
*	Multiplikation	$10 * 2 (=20)$
/	Division	$10 / 2 (=5)$
MOD	Modulo	$12 \text{ MOD } 7 (=5)$
+	Addition	$12 + 7 (=19)$
-	Subtraktion	$12 - 7 (=5)$
< , > , <= , >= , = , <>	Vergleich	$12 < 7 (=FALSE)$
& , AND	boolesches UND	TRUE AND FALSE (=FALSE)
XOR	boolesches XOR	TRUE XOR FALSE (=TRUE)
OR	boolesches ODER	TRUE OR FALSE (=TRUE)

Übung zu IEC 61131-3: Strukturierter Text

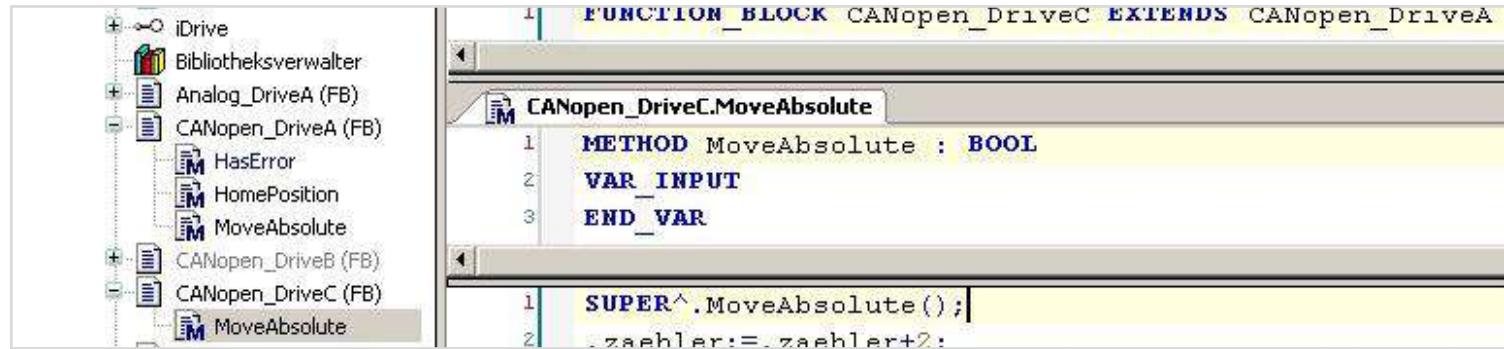
- Bilden Sie den folgenden Automaten in ST nach.



Informations- und Automatisierungstechnik

Kapitel 9: Steuerungskonfiguration – Teil 2

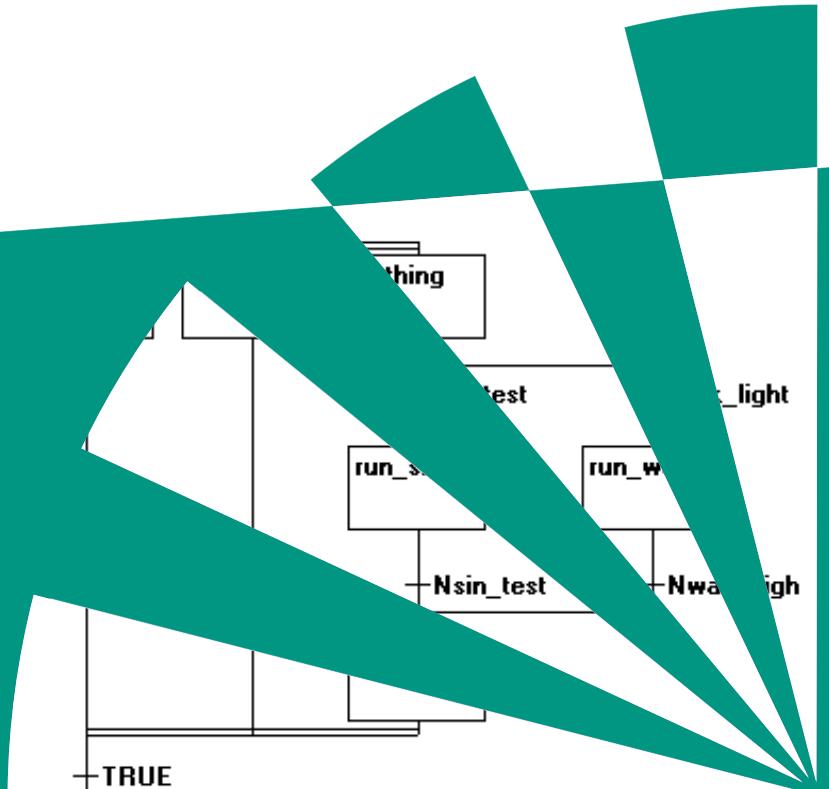
Univ.-Prof. Dr.-Ing. Mike Barth



The screenshot shows a SIMATIC Manager interface. On the left, a tree view displays function blocks under a project named 'iDrive'. The 'CANopen_DriveC (FB)' node is expanded, showing three sub-methods: 'MoveAbsolute', 'HasError', and 'HomePosition'. On the right, the code editor shows the implementation of the 'MoveAbsolute' method for the 'CANopen_DriveC' function block. The code is as follows:

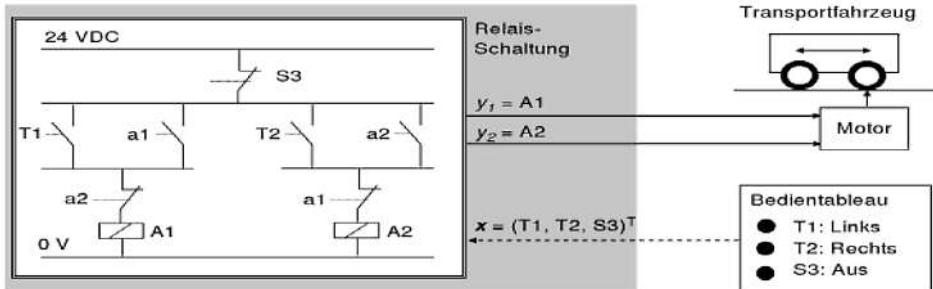
```
1 FUNCTION_BLOCK CANopen_DriveC EXTENDS CANopen_DriveA
2
3   METHOD MoveAbsolute : BOOL
4   VAR_INPUT
5   END_VAR
6
7   SUPER^.MoveAbsolute();
8   .zaehler:=.zaehler+2;
```

Ablausprache

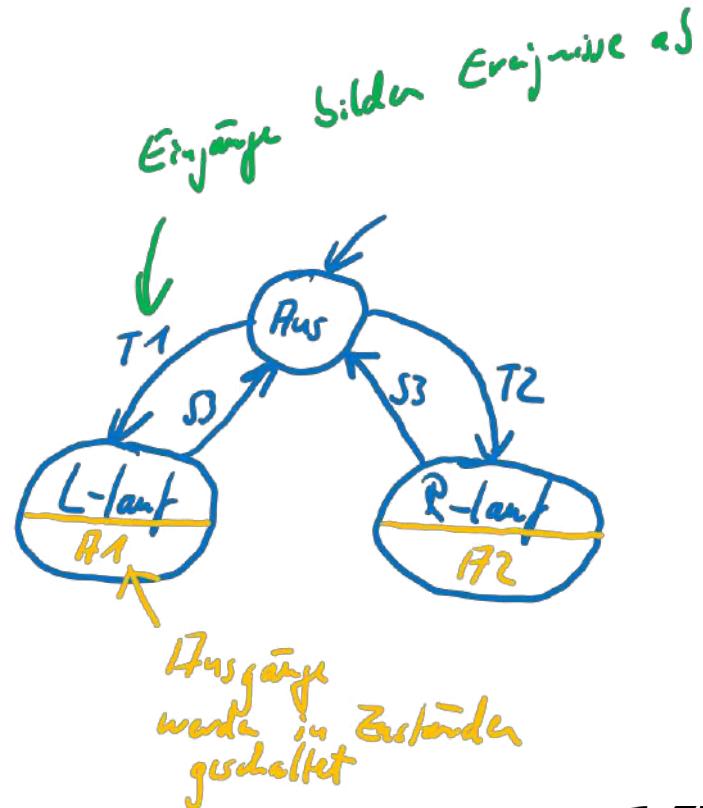


Einstiegsbeispiel

- Die Steuerung kennt unterschiedliche Zustände:
 - Rechtslauf, Linkslauf und Aus

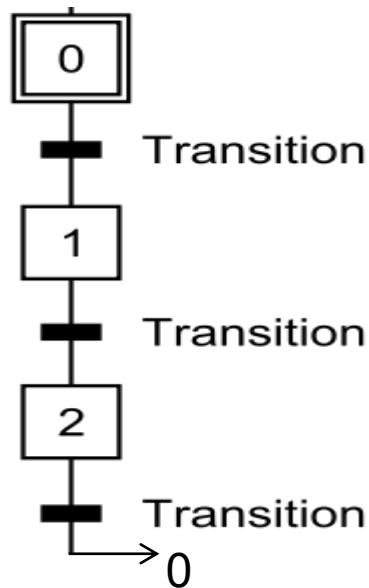


- Zustände, Zustandsübergänge, Eingaben und Ausgaben:



Einführung in AS

- Wie die anderen grafischen Programmiersprachen, besteht die Ablausprache aus Netzwerken mit Schritten (= Zuständen bei Automaten) und Transitionen (= Transitionen bei Automaten).

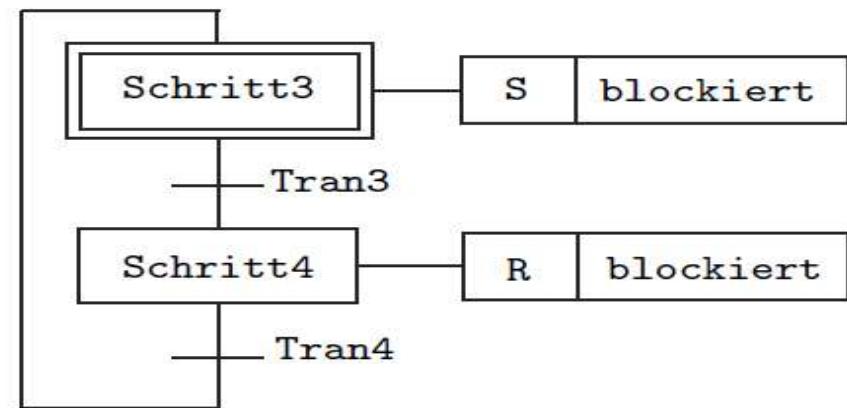
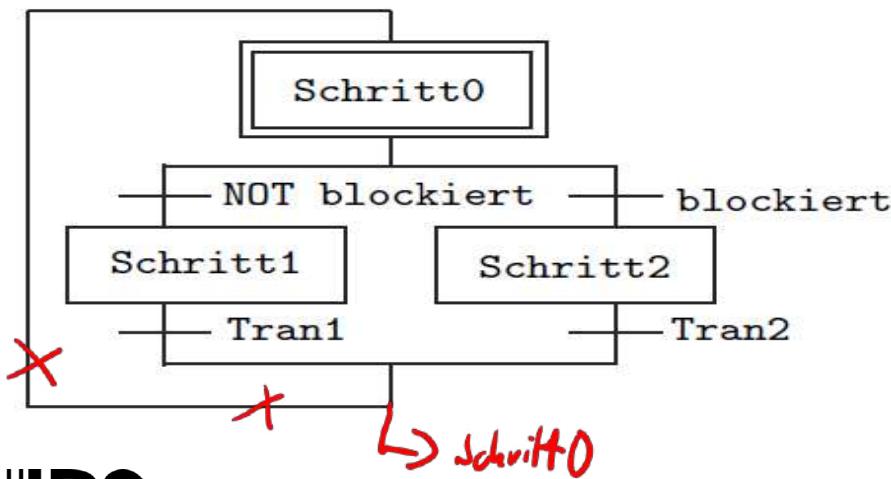


Einführung in AS

- Die Ablausprache
 - wurde entwickelt, um komplexe Aufgaben in Teilaufgaben zu zerlegen.
 - kontrolliert den Aufruf von Teilaufgaben.
 - eignet sich vor allem dort, wo sich Automaten eignen
 - ist eine grafische Programmiersprache, kann aber auch textuell programmiert werden.
- Die Teilaufgaben können in einer der anderen Sprachen (AWL, ST, FBS, KOP) oder in Ablausprache selbst programmiert werden.

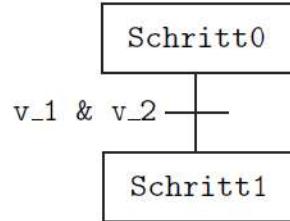
Einführung in AS

- Ähnlich wie bei Automaten können in den Zuständen Aktionen / Funktionen ausgeführt werden. Die hierfür verwendeten Aktionsblöcke bestimmen unter welchen Bedingungen Aktionen ausgeführt werden sollen. Im Aktionsblock einer Aktion können Anweisungen in AWL, ST, FBS und KOP stehen. Im einfachsten Fall werden boolesche Variablen gesetzt:

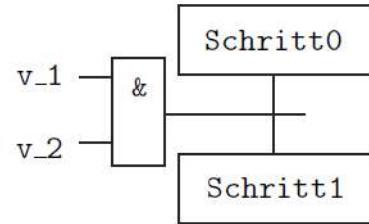


Einführung in AS

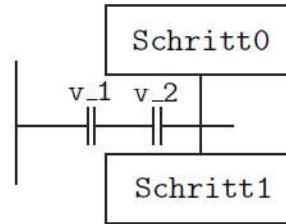
Direkt:



Ablausprache

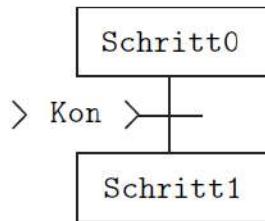


Funktionsbausteinsprache

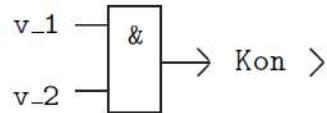


Kontaktplan

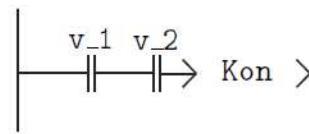
Konnektor:



Konnektor



Funktionsbausteinsprache



Kontaktplan

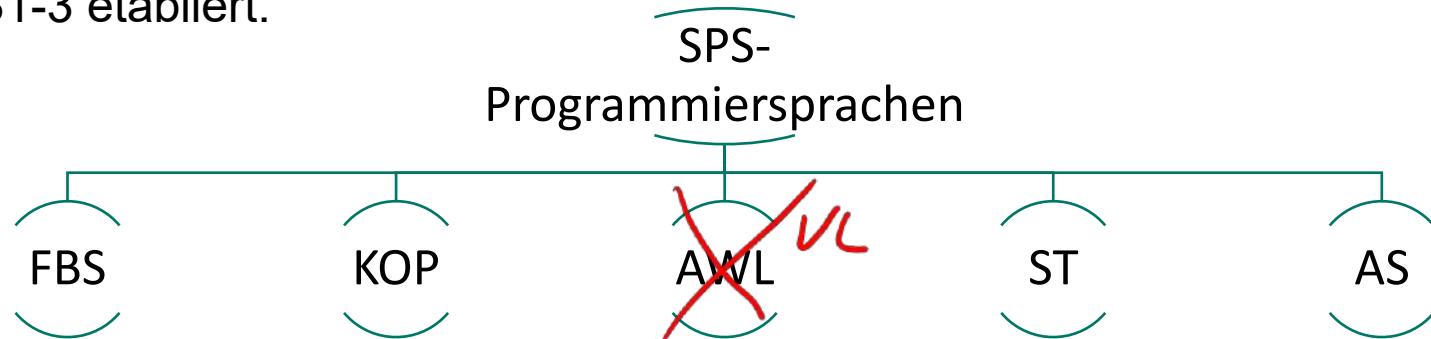
Einführung in AS

■ Bestimmungszeichen (BZ) für Aktionsblöcke

BZ	Beschreibung	
<i>leer</i>		wie N
N	nicht speichern	
R	rücksetzen	Setze und speichere Wert zu FALSE
S	setzen	Setze und speichere Wert zu TRUE
L	zeitbegrenzt	Liefere TRUE bis Zeit abgelaufen oder Schritt deaktiviert
D	zeitverzögert	Liefere TRUE nach einer Zeitverzögerung bis Schritt deaktiviert
P	Impuls	Liefere TRUE nach einem Übergang von FALSE nach TRUE

Die SPS-Programmiersprachen - Zusammenfassung

- Über die Jahre haben sich spezielle, herstellerspezifische Lösungen herausgebildet.
 - Ständige Zunahme von Programmiersprachen und herstellerspezifischen Dialekt(en) erschwert die Ausbildung und Wartung der Anlagen.
 - Wiederverwendung für andere Anlagen wird erschwert, wenn andere Steuerungshardware eingesetzt wird.
- Lösung: Standardisierung: Seit 1993 hat sich die internationale Norm IEC 61131-3 etabliert.



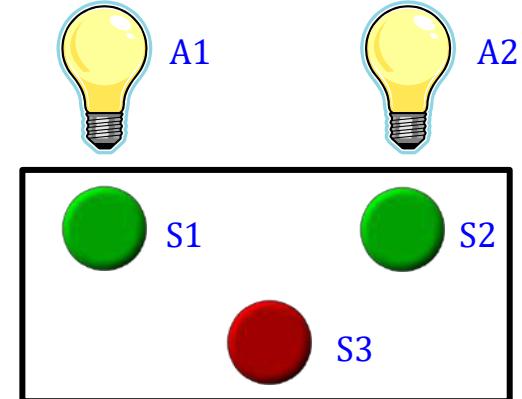
Übungsbeispiel: Lampensteuerung

Steuerungsaufgabe

- Einschalten der Lampe mit dem jeweiligen Taster
- Ausschalten aller Lampen mit S3

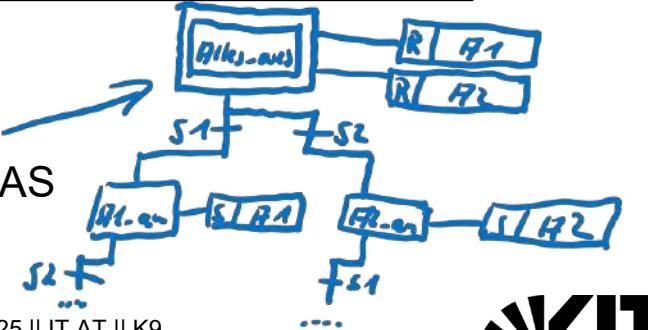
Sensoren und Aktuatoren

S1	Taster Licht 1	TRUE, wenn gedrückt
S2	Taster Licht 2	TRUE, wenn gedrückt
S3	Taster Aus	TRUE, wenn gedrückt
A1	Lampe 1	TRUE → Lampe 1 an
A2	Lampe 2	TRUE → Lampe 2 an



Aufgabe

- Steuerung mit einem Automaten, Programmierung in AS
- Steuerung mit mehreren Automaten, Programmierung in AS

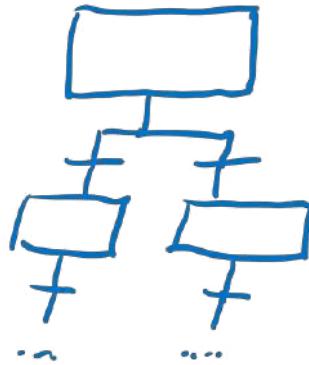


Alternativ- und Parallelverzweigung in AS

Alternativ & Parallelverzweigung

■ Alternativverzweigung:

- Es wird genau ein Ast abgearbeitet
- Transitionen in die einzelnen Äste nach der Verzweigung
- üblicherweise komplementär zueinander, Auswertung von links nach rechts

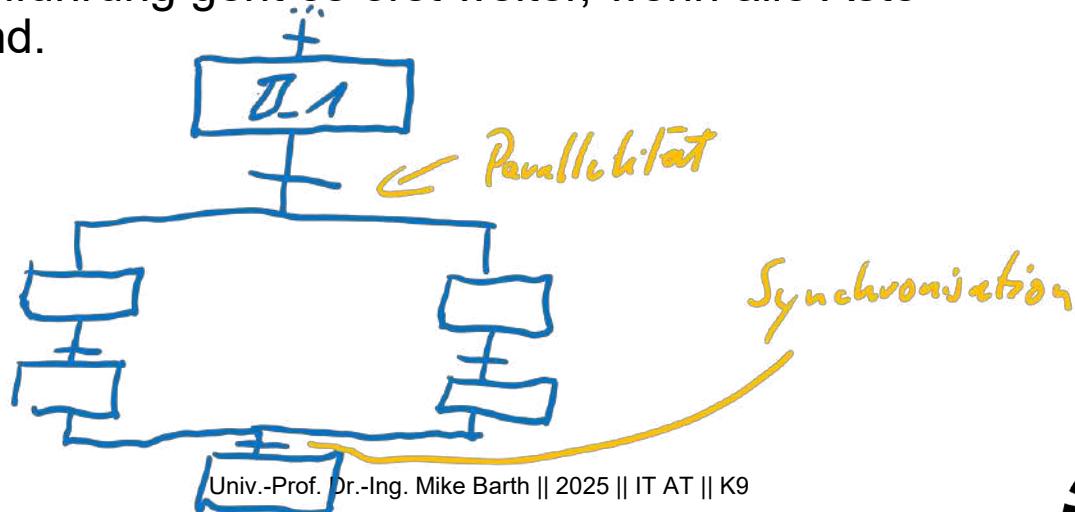


Alternativ- und Parallelverzweigung in AS

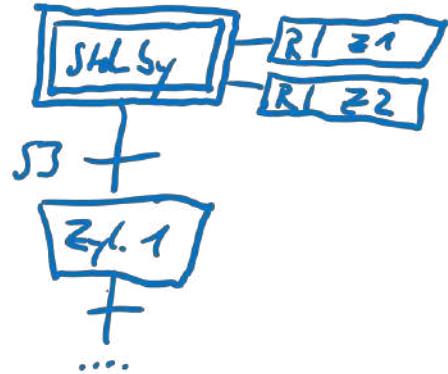
Alternativ & Parallelverzweigung

■ Parallelverzweigung

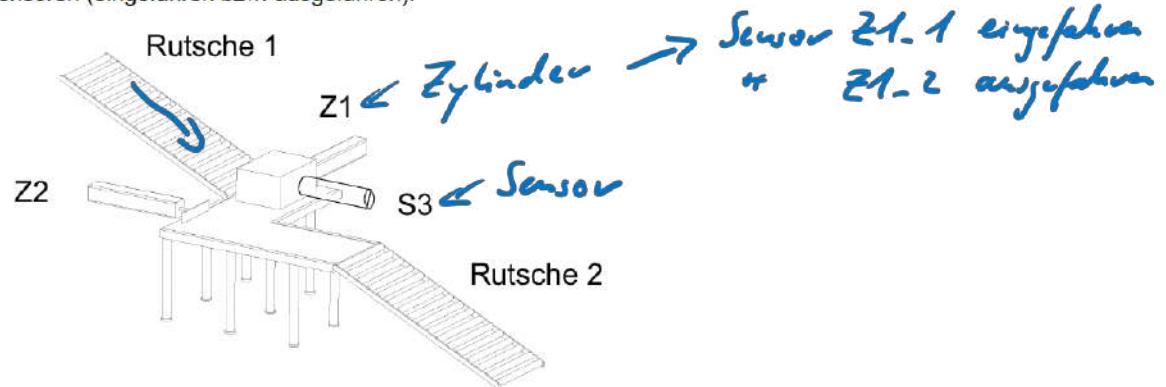
- Es werden alle Äste parallel abgearbeitet
- Transitionen vor der Verzweigung
- Bei der Zusammenführung geht es erst weiter, wenn alle Äste "angekommen" sind.



Übungsaufgabe

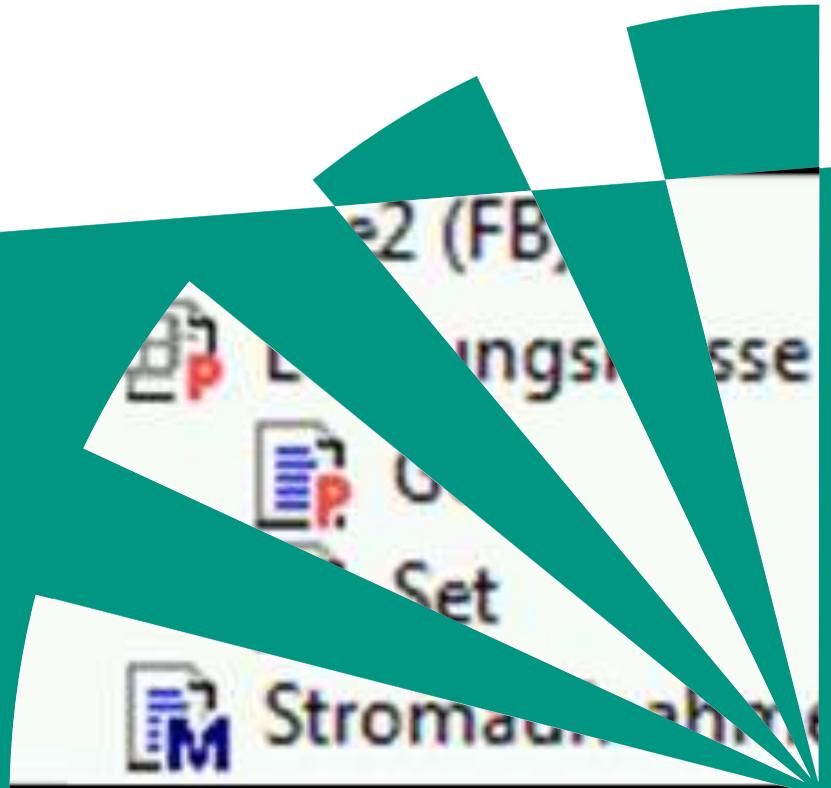


Auf einer Transporteinrichtung rollen Pakete von der Rutsche 1 auf einen Schiebetisch. Der Sensor S3 meldet, wenn ein Paket angekommen ist. Dann wird es vom Zylinder Z1 nach vorne geschoben. Danach fährt die Kolbenstange von Zylinder Z1 in die Ausgangsstellung zurück, bevor Zylinder Z2 das Paket senkrecht zur bisherigen Richtung auf die Rutsche 2 schiebt. Danach fährt die Kolbenstange von Zylinder Z2 in die Ausgangsstellung zurück. Jeder Zylinder hat zwei eingebaute Endlagensensoren (eingefahren bzw. ausgefahren).



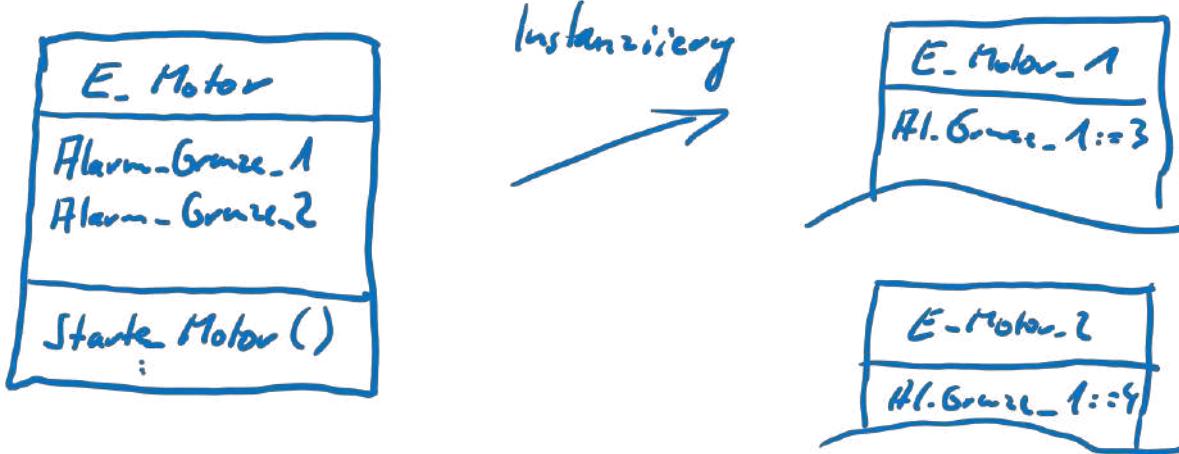
Erstellen Sie die Steuerung in AS

Objektorientierung in der AT



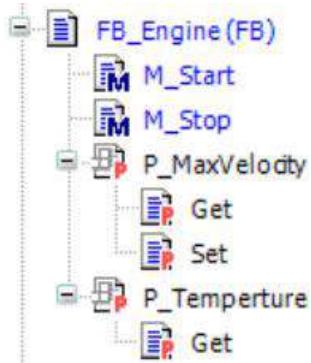
Objektorientierung in IEC 61131

- Eine Klasse ist eine Beschreibung einer Menge von Objekten, die ein gemeinsames Verhalten und gemeinsame Eigenschaften (=Merkmale, Properties) aufweisen. Sie definiert die innere Struktur aller nach ihrem Muster erzeugten Objekte und stellt somit ein Konstruktionsprinzip dar.
- Zu einer Klasse (Bauplan) gibt es kein, ein oder viele Exemplare (Objekte, Instanzen).
 - Beispiel (Elektromotor)



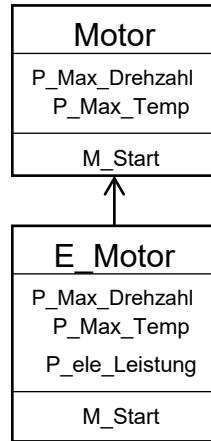
Datenkapselung in der IEC 61131

- Eigenschaften können in der Objektorientierung mit sog. „GET“ und „SET“-Funktionen gelesen bzw. beschrieben werden.
- Durch Weglassen der GET und/oder SET-Funktion kann unbeabsichtigtes Beschreiben oder Lesen verhindert werden
 - → Datenkapselung



Vererbung in IEC 61131

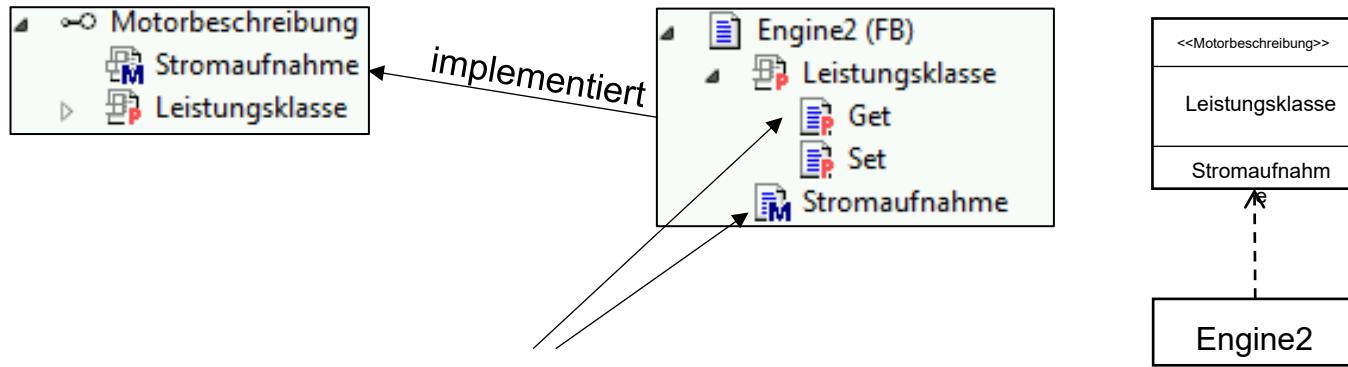
- Oftmals werden Varianten von Objekten benötigt, die sich in gewissen Aspekten unterscheiden. Hierbei kann die Vererbung von objektorientierten Bausteinen helfen.



Interfaces in der IEC 61131-3

- Bei Interfaces handelt es sich um eine Definition von Methoden und Properties, die in eine Klasse implementiert wurden.
- Klassen, die das gleiche Interface implementieren, sehen von außen identisch aus und können gleichwertig behandelt werden.
- Die genaue Implementierung der vorgeschriebenen Methoden und Properties wird nicht durch das Interface beschrieben.
 - Jede Klasse, die das Interface implementiert, kann die interne Gestaltung selber festlegen.

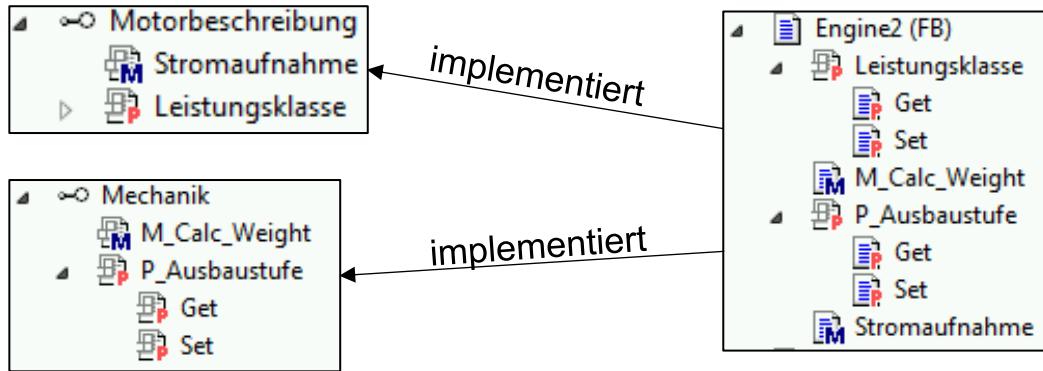
Interfaces in der IEC 61131-3



Genaue Implementierung muss
für jede Klasse definiert werden

Implementierung von Interfaces

- Eine Klasse kann mehrere Interfaces implementieren



Übung zur Objektorientierung

- Erstellen Sie einen Baustein „FB_Robot“ für die Ansteuerung eines Roboters
 - Hierin sollen folgende Eigenschaften gelesen und geschrieben werden
 - Achsenanzahl
 - zul. Traglast
 - Wiederholgenauigkeit
 - Gewicht
 - max. Reichweite
 - Der Baustein soll eine Methode für die Berechnung der aus der max. Reichweite resultierenden max. Arbeitsfläche erhalten
 - Der Baustein soll ein Interface „iRobot“ referenzieren, welches alle Eigenschaften und Methoden definiert.

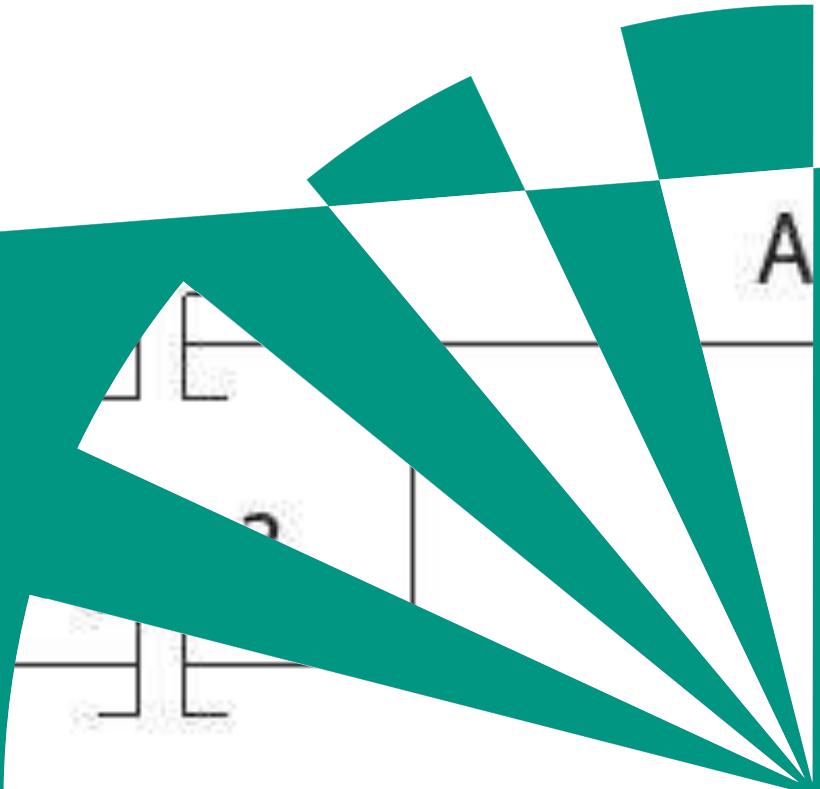


Erweiterung der Übung zur Objektorientierung

- Leiten Sie aus dem Baustein „FB_Robot“ einen Baustein „FB_6Axis“ ab, der zusätzlich ein Property „IP_Protection“ (elektrisch) bekommt.
- Darüber hinaus bekommt dieser neue Baustein eine Methode „Start_Program“ welche die Programmnummer übergeben bekommt sowie eine Methode „Stopp_Program“.



Variablenarten

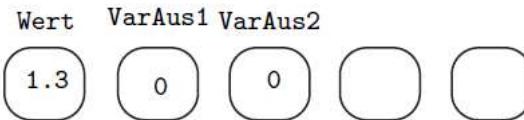


Variablenarten

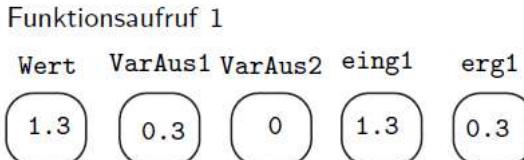
- **VAR Lokale Variable** nur innerhalb der POE sichtbar.
 - Beispielsweise innerhalb eines Programms oder eines Funktionsbausteins
- **VAR INPUT** Eingangsvariable, nur innerhalb der POE definiert und dort nur lesbar (call-by-value).
 - Zum Beispiel: Eingangsvariable in einen Funktionsbaustein
- **VAR OUTPUT** Ausgangsvariable, außerhalb der POE lesbar, innerhalb les- und schreibbar (call-by-value).
 - Zum Beispiel: Ausgangsvariable in einen Funktionsbaustein
- **VAR IN OUT** Ein- und Ausgangsvariable, daher intern und extern les- und schreibbar (call-by-reference).
 - Zum Beispiel: Bidirektonaler Pin an Funktionsbaustein
- **VAR GLOBAL** Globale Variable, die in einer Konfiguration deklariert wird. Kann von anderen POEs gelesen und beschrieben werden.
 - Zum Beispiel: Variablen die auf einem Bedienbild visualisiert werden sollen

Variablenarten – Call-by-Value

```
FUNCTION_BLOCK Fun_1
VAR_INPUT eing1 : REAL END_VAR
VAR_OUTPUT erg1 : REAL END_VAR
erg1 := eing1 - 1;
END_FUNCTION_BLOCK
```

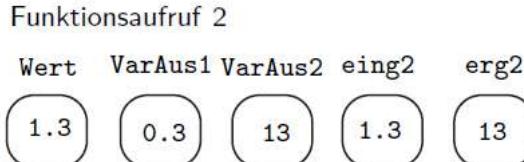


```
FUNCTION_BLOCK Fun_2
VAR_INPUT eing2 : REAL END_VAR
VAR_OUTPUT erg2 : REAL END_VAR
erg2 := eing2 * 10;
END_FUNCTION_BLOCK
```



Funktionsaufrufe:

```
Wert := 1.3;
F1_Inst(eing1:=Wert,erg1=>VarAus1);
F2_Inst(eing2:=Wert,erg2=>VarAus2);
```



Wert bleibt unverändert

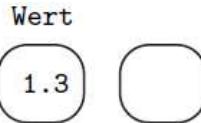
Variablenarten – Call-by-Reference

```
FUNCTION_BLOCK Fun_1
VAR_IN_OUT eing : REAL END_VAR
eing := eing - 1;
END_FUNCTION_BLOCK
```

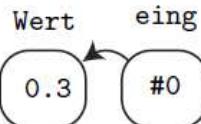
```
FUNCTION_BLOCK Fun_2
VAR_IN_OUT eing : REAL END_VAR
eing := eing * 10;
END_FUNCTION_BLOCK
```

Funktionsaufrufe:

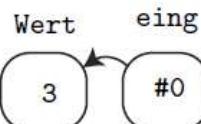
```
Wert := 1.3;
F1_Inst(eing:=Wert);
F2_Inst(eing:=Wert);
```



Funktionsaufruf 1

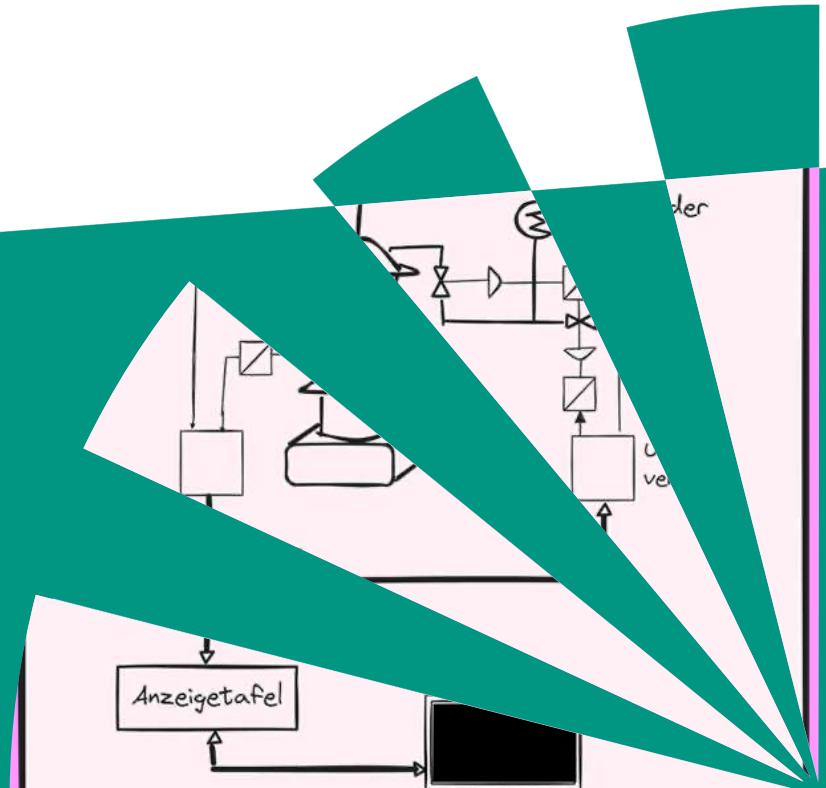


Funktionsaufruf 2

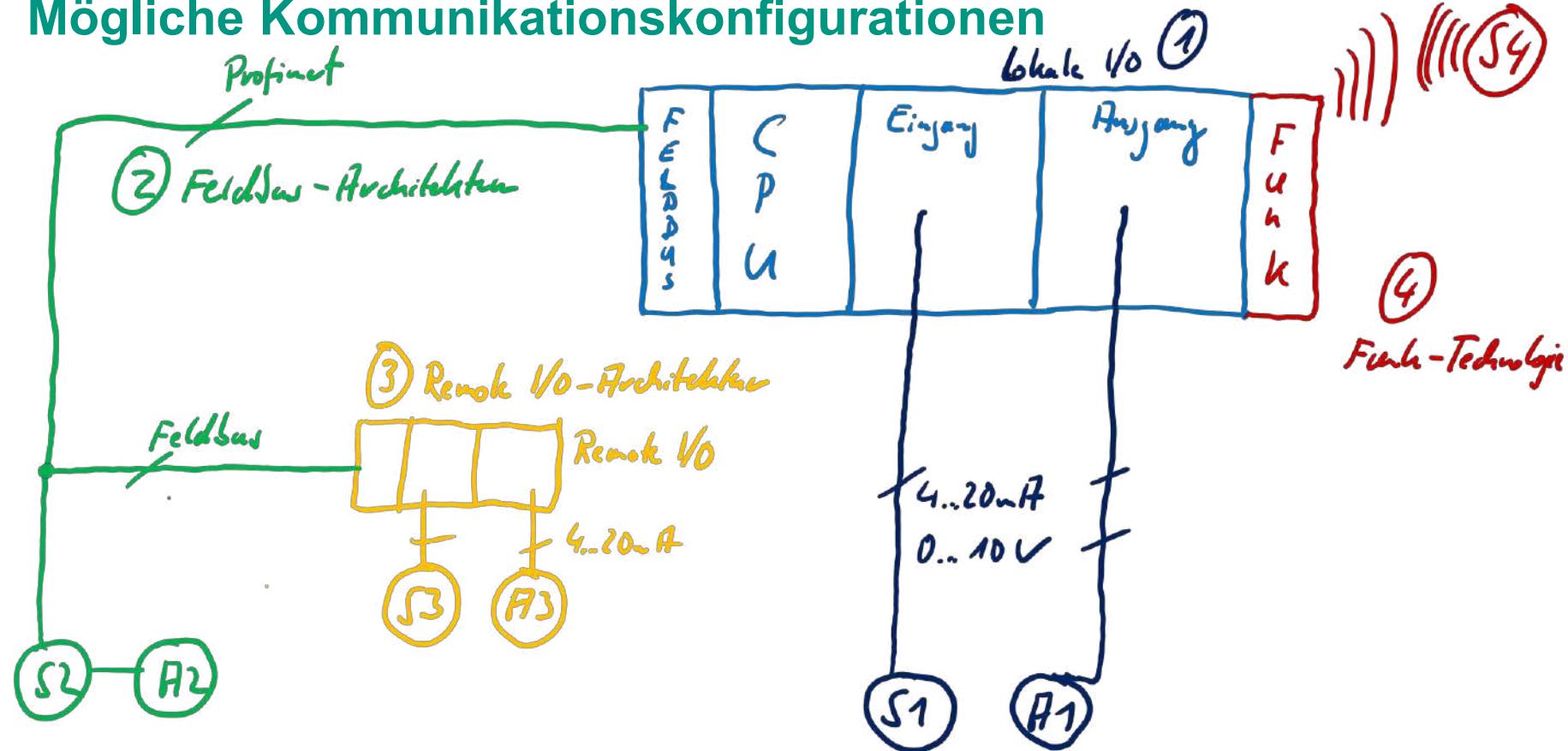


Wert wird verändert

AT-Architektur



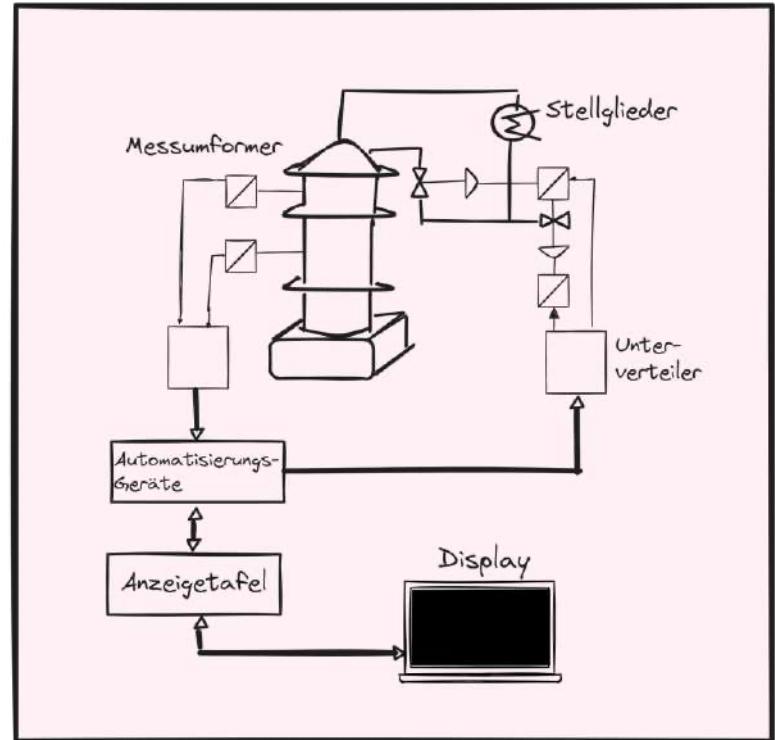
Mögliche Kommunikationskonfigurationen



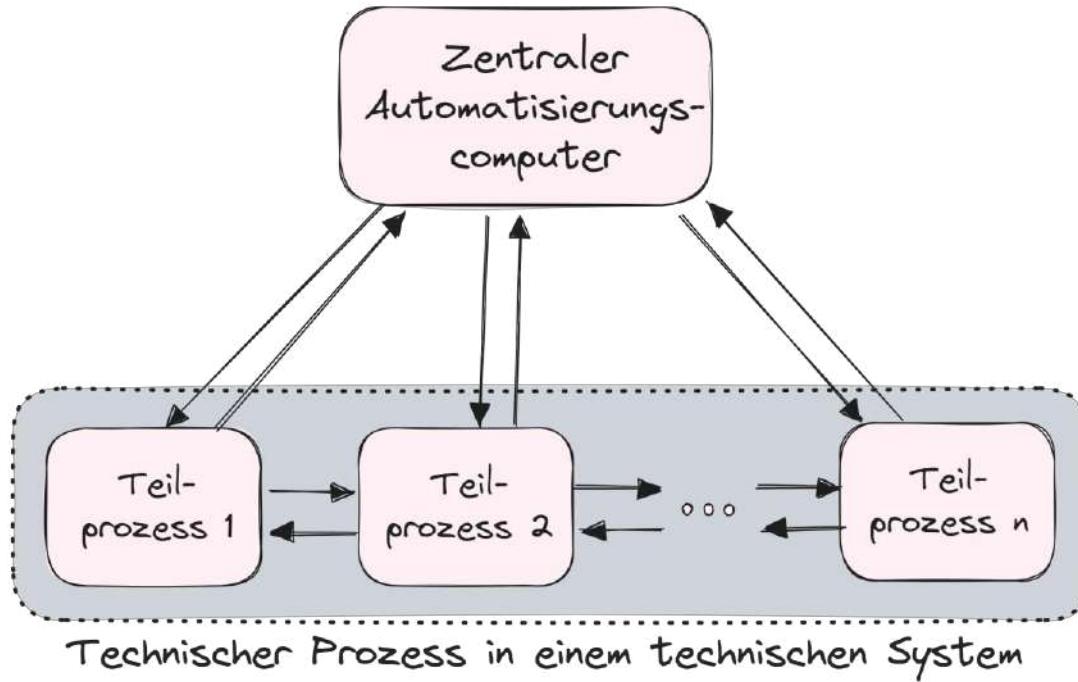
Automatisierungsarchitekturen: Zentral

■ Zentrale Automatisierung

- Wenige zentralisierte Automatisierungsgeräte (z.B. SPS) in einem Schaltschrank für die gesamten Anlage / Maschine.
- Auf eine Anlage / Maschine zugeschnitten
- Geeignet für statischen Aufbau mit Single-Use

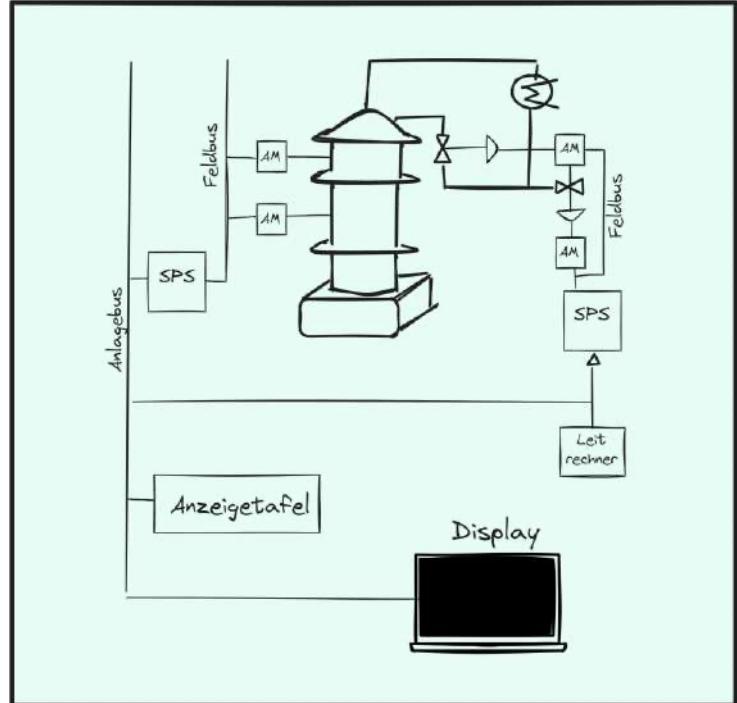


Automatisierungsarchitekturen: Zentral

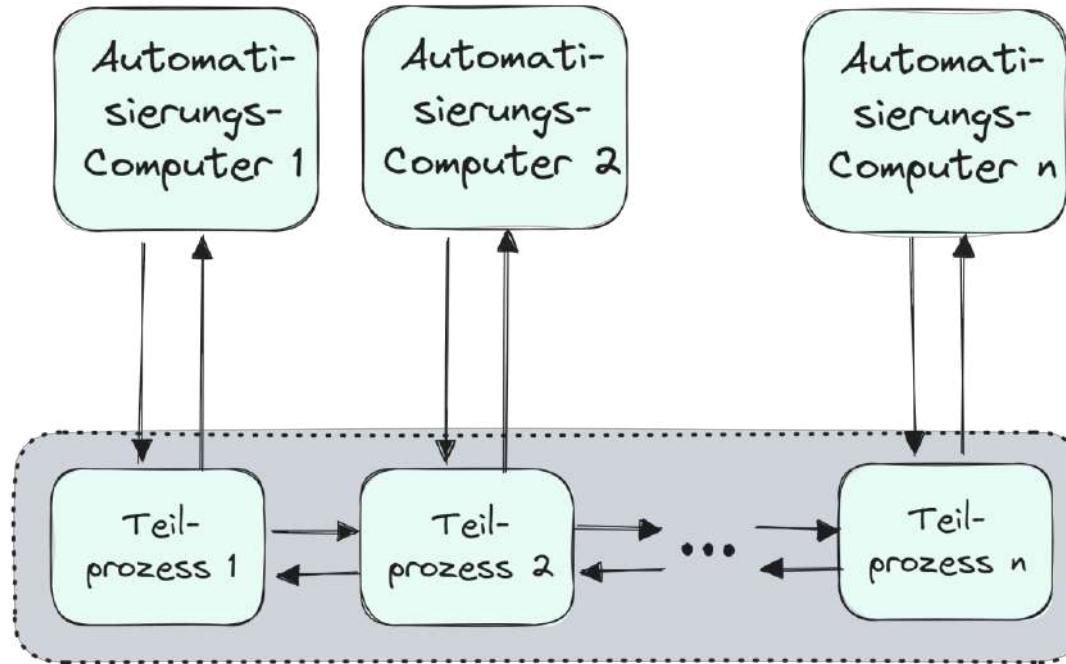


Automatisierungsarchitekturen: Dezentral

- Dezentrale Automatisierung
 - Verteilte Steuerungen (SPS) die für dedizierte Anlagenteile eingesetzt werden.
 - Übergeordnete Orchestrierung notwendig, z.B. mit einem Prozessleitsystem
 - Aufteilung von Aufgaben und Funktionen auf Teilanlagensteuerungen
 - Geeignet für Reduktion der Komplexität der Steuerungskonfiguration
 - Nachteil: zusätzlicher Aufwand zur Kommunikation der einzelnen Automatisierungseinheiten



Automatisierungsarchitekturen: Dezentral



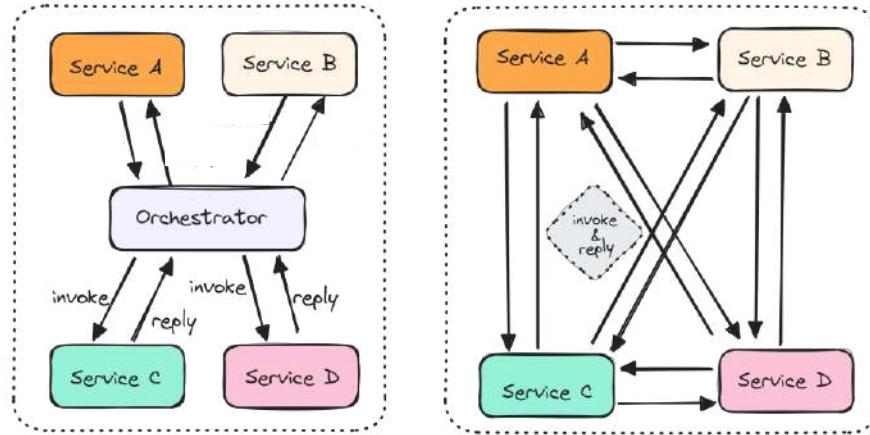
Automatisierungsarchitekturen: Dezentral

Ziele der dezentralen Automatisierung und verteilter Systeme

- Hohe Zuverlässigkeit durch Fehlertoleranz
 - Ausfall eines dezentralen Rechners (Steuerung) führt nicht zu Gesamtausfall
 - Eingrenzung des Fehlers durch Rekonfiguration
- Erhöhung der Verfügbarkeit durch schnelle Wartung und Instandsetzung
 - gegenseitige Überwachung mit Fehlerdiagnose
- Gegenseitige Aushilfe bei hoher Belastung
 - selbsttätige Anpassung der Aufgabenverteilung
 - Reduzierung der Reserven der einzelnen Einheiten

Automatisierungsarchitekturen

- Dezentrale Automatisierung und Modularisierung
 - Modulsteuerungen werden durch übergeordnete Steuerung orchestriert → vgl. dezentrale Automationsarchitektur
 - Alternativ bzw. parallel hierzu können die Modulsteuerungen direkt miteinander kommunizieren → Choreographie → vgl. Dronenschwarm
 - Neu: Service-basierte Konzepte aus dem klassischen Software-Engineering



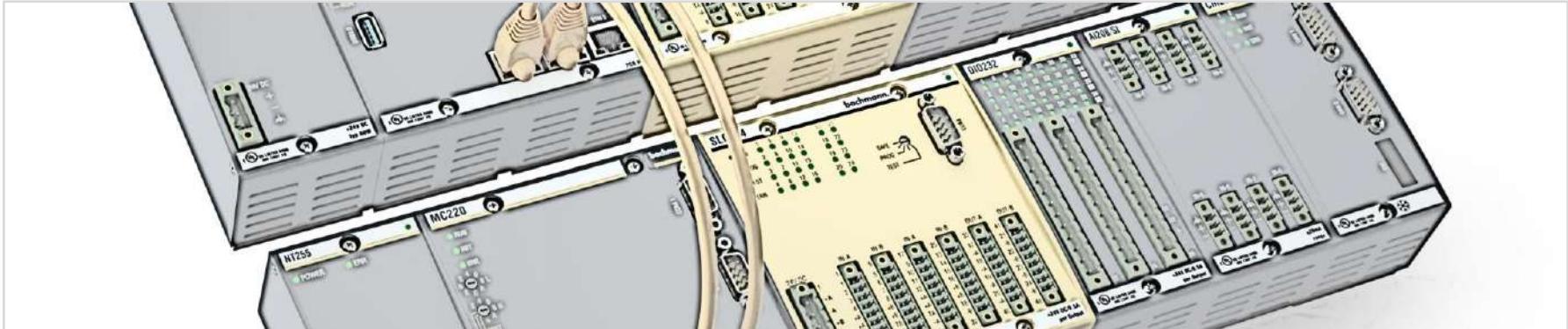
Automatisierungsarchitekturen

- Dezentrale Automatisierung und Modularisierung
 - Modulsteuerungen werden durch übergeordnete Steuerung orchestriert → vgl. dezentrale Automationsarchitektur
 - Alternativ bzw. parallel hierzu können die Modulsteuerungen direkt miteinander kommunizieren → Choreographie → vgl. Drohnenschwarm
 - Neu: Service-basierte Konzepte aus dem klassischen Software-Engineering
- Ziele:
 - Vgl. dezentrale AT-Architektur
 - Zzgl. Verwendung einheitlicher Schnittstellen zur Realisierung eines „Plug- & Produce“-Gedankens.
 - Module können von unterschiedlichen Herstellern (Modulbauern) stammen, da Schnittstellen (insbesondere Softwaretechnisch) standardisiert sind → z.B. Verwendung von OPC UA.

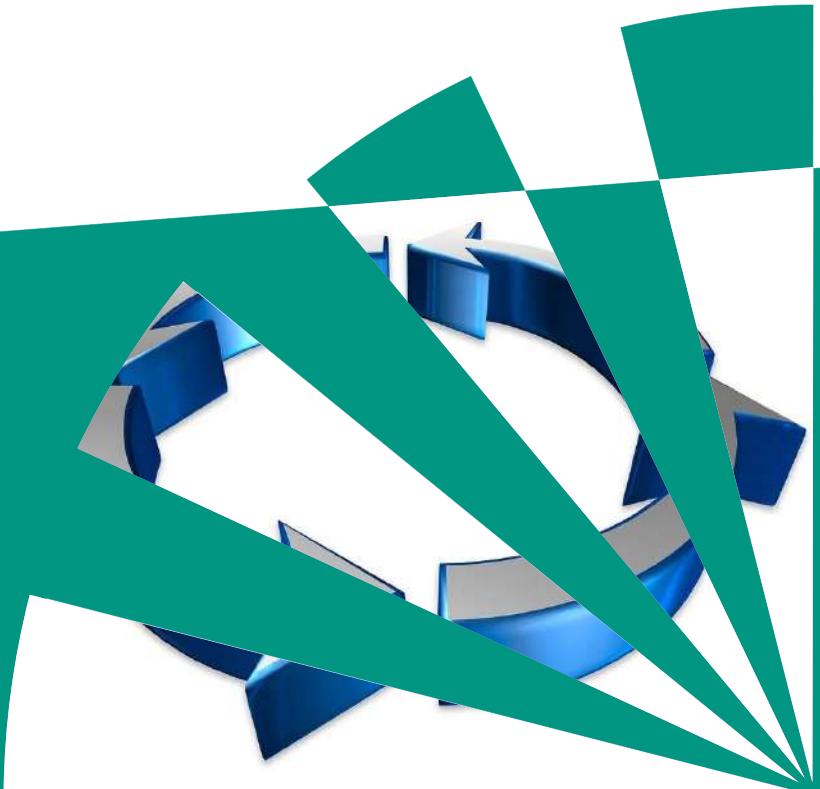
Informations- und Automatisierungstechnik

Kapitel 10: Redundanz und Fehler

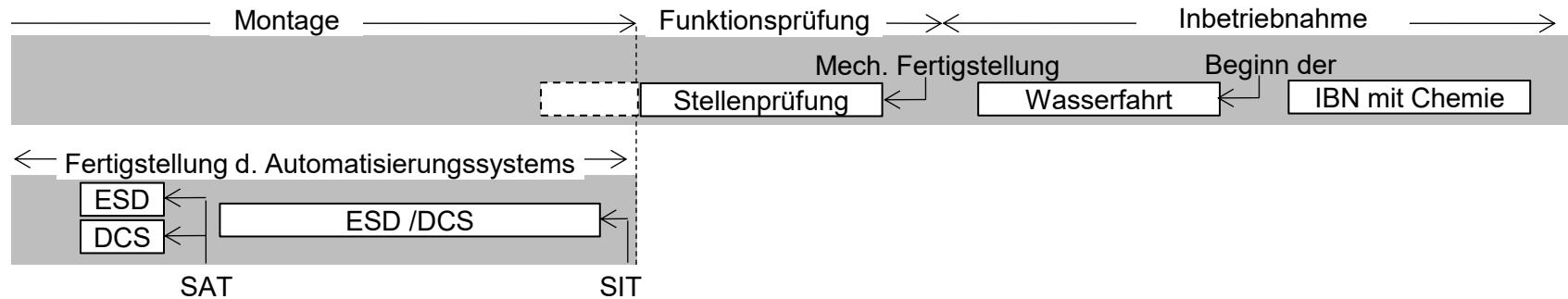
Univ.-Prof. Dr.-Ing. Mike Barth



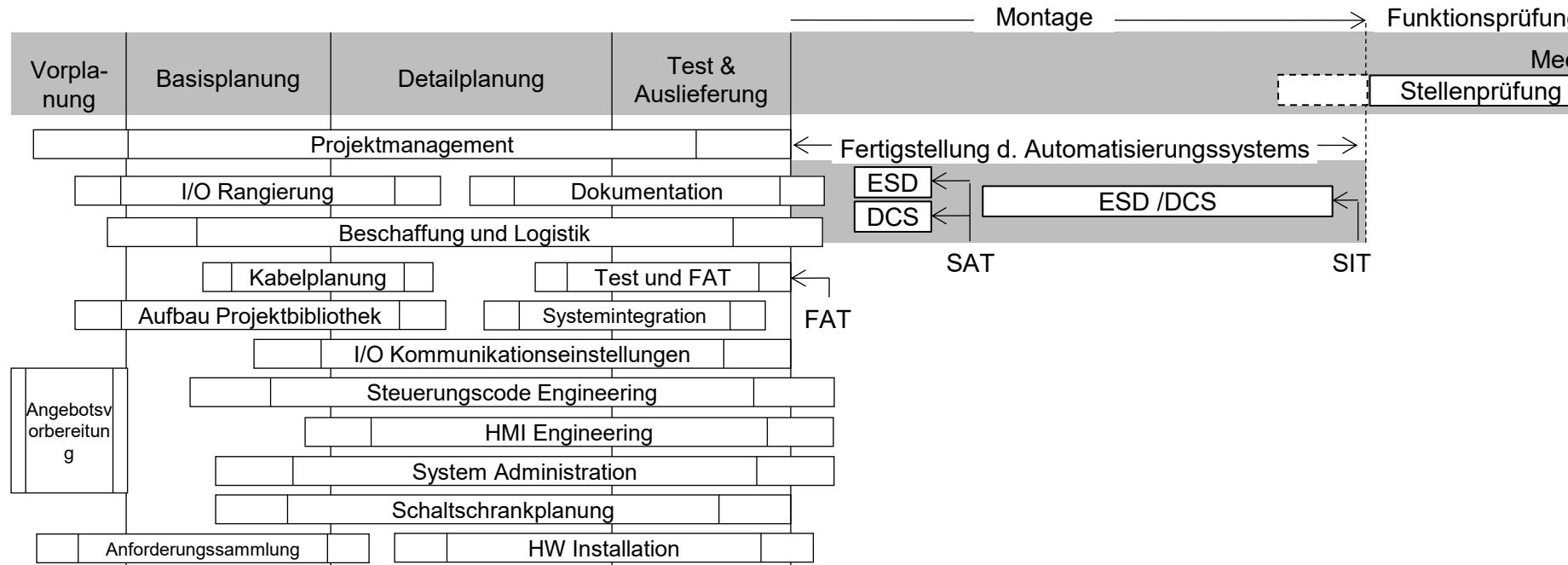
Engineering von AT-Systemen



Engineering-Zyklus von AT-Systemen



Engineering-Zyklus von AT-Systemen



Redundanzstrukturen



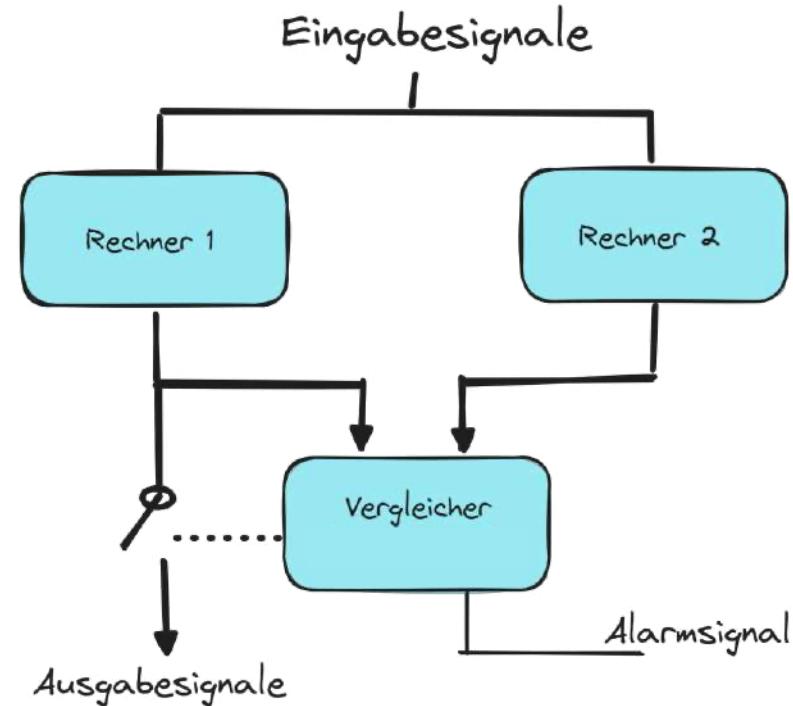
Redundanzstrukturen

■ Formen der Redundanz

- Hardware-Redundanz → redundante Hardware
- Software-Redundanz → redundante Software
- Messwert-Redundanz → redundante Messgrößen; abhängige Messgrößen
- Zeit-Redundanz → mehrfache Abfrage des gleichen Messwertes in bestimmten Zeitabständen
- Insb. Hardware- und Softwareredundanz bedeutet höherer Aufwand
 - erhöhte Verfügbarkeit
 - sicherheitsrelevante Systeme

Hardware-Redundanz - Doppelstrukturen

- Zwei Rechner (z.B. SPS) arbeiten parallel an den gleichen Abläufen.
- Ein Vergleicher analysiert potenzielle Unterschiede und gibt die Ausgabesignale nur dann frei, wenn kein Unterschied erkannt wird.
- Reaktion bei Fehler in einem Rechner: System schaltet ab → **hohe** Sicherheit bei gleichzeitig **geringer** Verfügbarkeit.

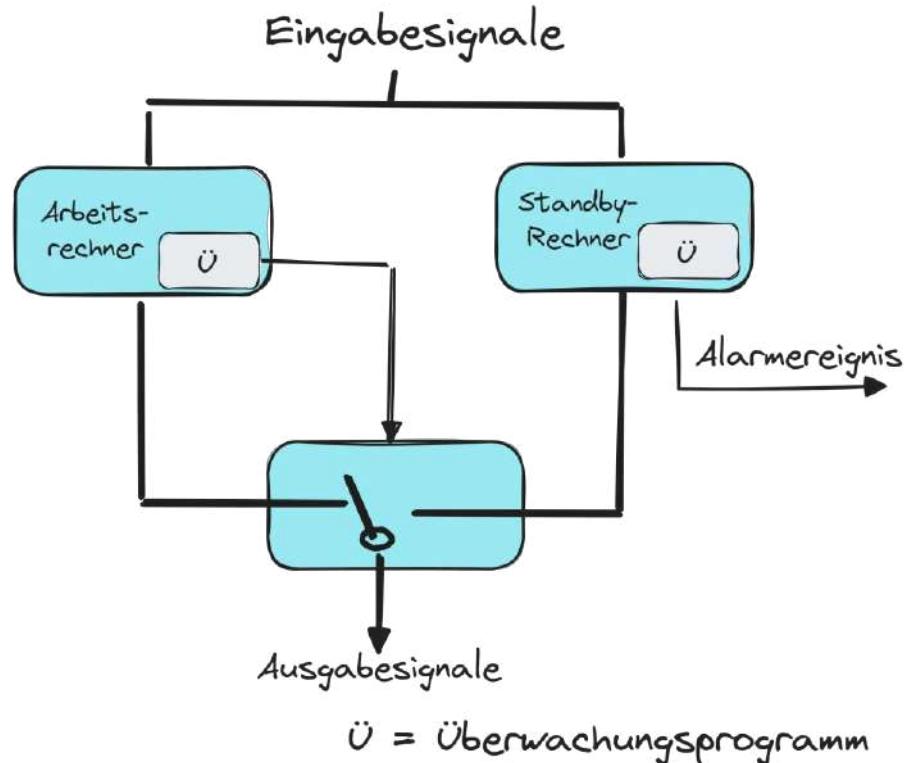


Einschub: Sicherheit vs. Verfügbarkeit

- Die **Verfügbarkeit** eines Systems beschreibt, wie **häufig oder zuverlässig** es **betriebsbereit** ist – also wie oft es **nicht** durch Fehler oder Wartung ausfällt.
- **Sicherheit** (im Sinne der funktionalen Sicherheit) beschreibt die **Fähigkeit eines Systems, gefährliche Fehler zu erkennen und in einen sicheren Zustand zu überführen**, um **Unfälle oder Schäden zu vermeiden**.
 - Ein System ist **sicher**, wenn es bei einem Fehler **nicht weiterarbeitet**, sondern **abschaltet oder in einen sicheren Zustand übergeht**.
 - Die Sicherheit hängt davon ab, **wie gut Fehler erkannt und behandelt werden** – z. B. durch Redundanz, Selbsttests oder Vergleichslogik.

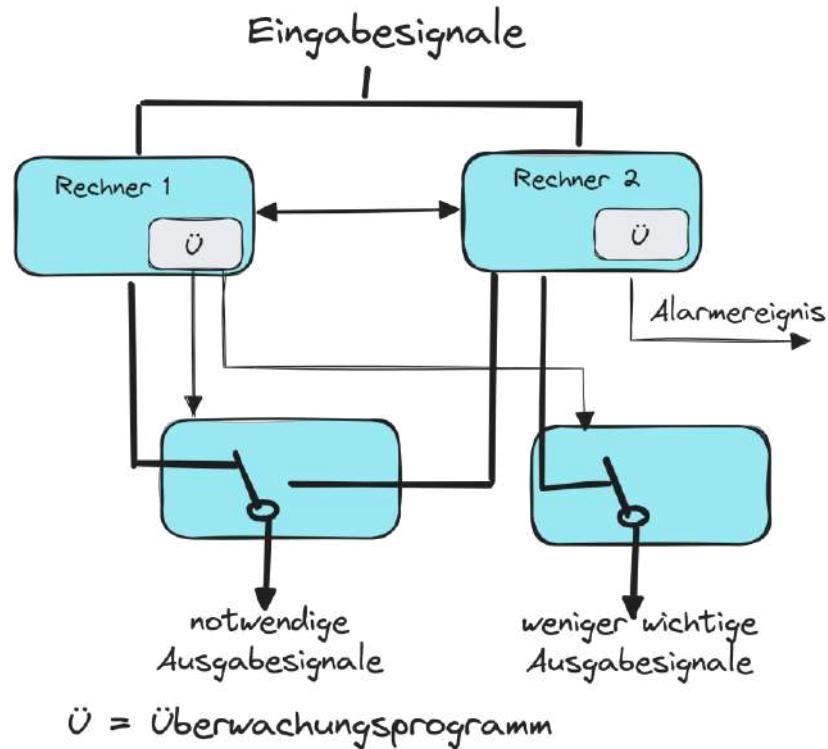
Hardware-Redundanz - Doppelstrukturen

- Ein Arbeits-Rechner (z.B. SPS) arbeitet das Steuerungsprogramm ab.
- Parallel ruht ein Standby-Rechner.
- Eine Überwachung „Ü“ schaltet dabei die Ausgabesignale:
 - Wenn Arbeitsrechner OK → Ausgabesignale des Arbeitsrechners
 - Wenn Arbeitsrechner nOK → Aktivierung Standby-Rechner und Umschaltung auf dessen Ausgabesignale.
 - Wenn Ü von Standby-Rechner einen Fehler auf dem Stand-by-Rechner erkennt, wird ein Alarm ausgelöst.
- Hohe Verfügbarkeit (Durch den Standby-Rechner kann das System bei Ausfall des Arbeitsrechners weiterarbeiten) bei gleichzeitig hoher Sicherheit (wenn Fehler zuverlässig erkannt werden).



Hardware-Redundanz - Doppelstrukturen

- Die Arbeitslast wird auf mehrere Rechner (hier Rechner 1 und 2) verteilt.
- Jeder Rechner hat eine Überwachung.
 - Alles OK: notwendige Ausgabesignale kommen von Rechner 1 und weniger wichtige Ausgabesignale kommen von Rechner 2
 - Rechner 1 Fehler: notwendige Ausgangssignale werden von Rechner 2 berechnet. Weniger wichtige Ausgabesignale werden zugunsten der notwendigen Ausgangssignale nicht berechnet.



Fehlerarten und -identifikation



Fehler im System: Zusammenhang zwischen Begriffen

■ Zuverlässigkeit:

- Wahrscheinlichkeit, dass ein System innerhalb eines Zeitintervalls unter zulässigen Bedingungen seine Funktion erfüllt.

■ Ausfall:

- Übergang von funktionsfähig in fehlerhaft
- Ausfall ist ein Zustandsübergang (Ereignis, kein Zustand)

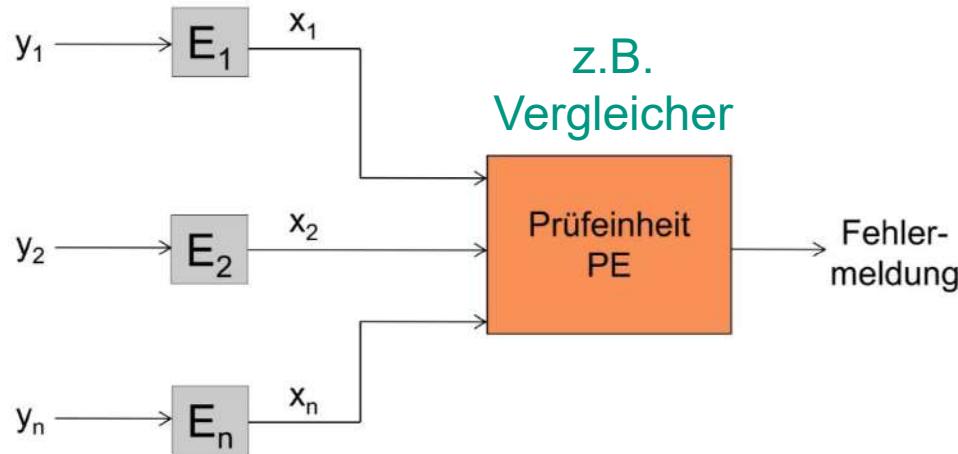
■ Fehler:

- Nichterfüllung mindestens einer Anforderung
- Fehler ist ein Zustand

Fehler im System: Fehlerarten

- Bauelemente- und Gerätefehler
 - Dimensionierungsfehler
 - Fertigungsfehler
 - Schaltungsfehler
 - Verdrahtungsfehler
 - Entwurfsfehler
 - Konzeptfehler
- Fehlerarten der Programmsystemen
 - Spezifikationsfehler
 - Entwurfsfehler
 - Implementierungsfehler
 - Dokumentationsfehler
- Änderungsgröße
 - Änderungsausfall
 - Verkleinerung
 - Vergrößerung
 - Extremausfall
 - Kurzschluss
 - Unterbrechung
- Zeitverlauf
 - Sprungausfall
 - Driftausfall
- Zeitpunkt
 - Frühausfall
 - Zufallsausfall
 - Verschleißausfall
- Dauer
 - Sporadischer Ausfall
 - Statischer Ausfall
- Statistisches Verhalten
 - Zufallsausfall
 - Systematischer Ausfall
 - Deterministischer Ausfall
- Zahl
 - Einzelausfall / Mehrfachausfall
- Ursache
 - Primärausfall
 - Folgeausfall
- Sicherheit
 - sicherheitsbezogener Ausfall
 - nicht Sicherheitsbezogener Ausfall
- Schwere
 - Kritischer Ausfall
 - Nicht kritischer Ausfall
- Umfang
 - Vollausfall (Totalausfall)

Fehler im System: Fehleridentifikation



y_1, y_2, \dots, y_n unterschiedliche Eingangswerte für die Datenverarbeitung

E_1, E_2, \dots, E_n unterschiedliche Einheiten der Datenverarbeitung

x_1, x_2, \dots, x_n unterschiedliche Ausgangswerte für die Prüfung (sie müssen vorgegebene Zusammenhänge erfüllen)

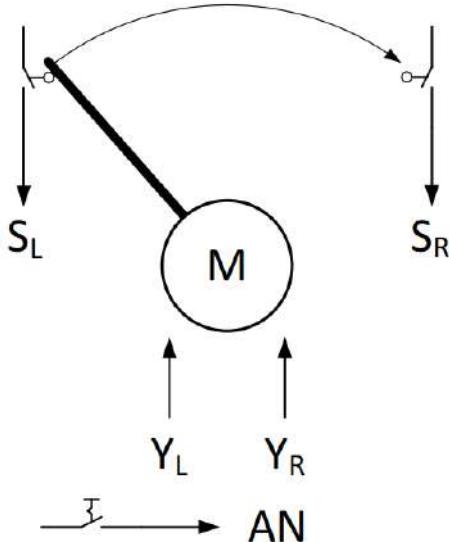
Fehler im System: Fehleridentifikation

- Bevor der Fehler behandelt werden kann, muss er zunächst erkannt werden.
- Die Fehlererkennung verlangt Kreativität.
- Grundlegende Ansätze sind:
 1. das Ansprechen (extra eingebauter) Sensoren, wie Türkontakte oder Detektoren für Teile,
 2. das Erkennen „unlogischer“ Zustände wie in den Zeilen 4 und 8 der Wahrheitstabelle des Scheibenwischers oder
 3. zu langes Verweilen in einem Zustand (Timeout)

Fehler im System: Fehleridentifikation

■ Beispiel Scheibenwischer

Bez.	Ein-/Ausgang	Bemerkung
Y_L	Ausgang	0: nicht links fahren 1: nach links fahren
Y_R	Ausgang	0: nicht rechts fahren 1: nach rechts fahren
S_L	Eingang	0: nicht ganz links 1: ganz links
S_R	Eingang	0: nicht ganz rechts 1: ganz rechts
An	Eingang	0: ausgeschaltet 1: eingeschaltet



Fehler im System: Fehleridentifikation

■ Beispiel Scheibenwischer

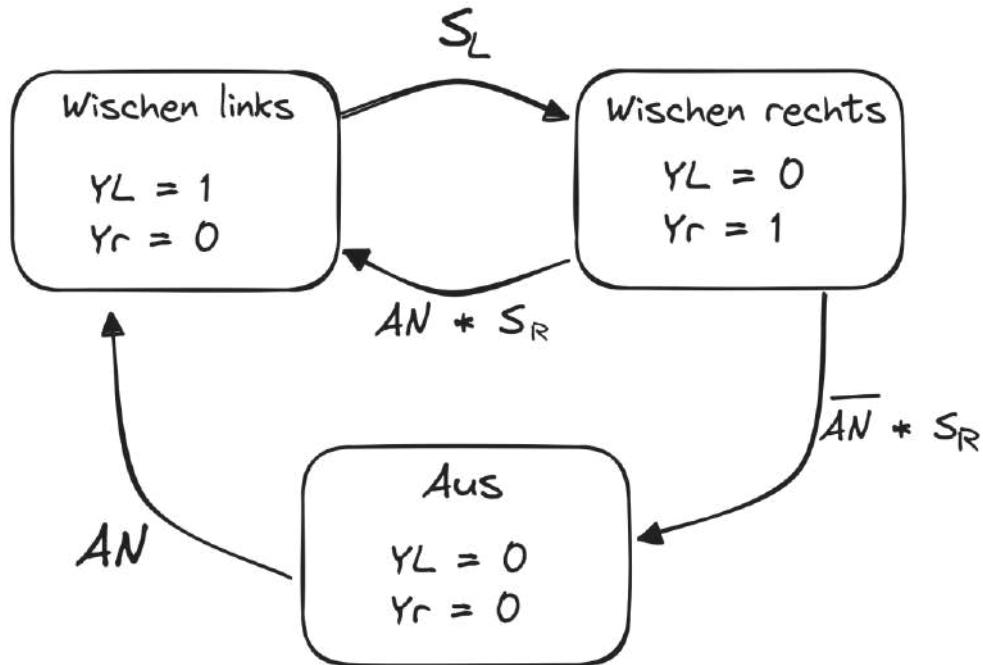
■ Vier dieser Zeilen sind „logisch“: 0 0 1, 0 1 0, 1 0 1 und 1 1 0. Hier befindet sich der Scheibenwischer jeweils am rechten bzw. linken Anschlag. Zwei weitere Zeilen sind „unlogisch“: 0 1 1 und 1 1 1. Hier sprechen der rechte und der linke Grenztaster gleichzeitig an. Das kann nicht sein, also sollte unsere Automatisierung einen Fehler melden.

■ Was ist bei 0 0 0 und 1 0 0 ?

AN	S _L	S _R	Y _L	Y _R	Situation
0	0	0	?	?	Aus, dazwischen
0	0	1	0	0	Aus, ganz rechts
0	1	0	?	?	Aus, ganz links
0	1	1	?	?	„unlogisch“ → Fehler
1	0	0	?	?	An, dazwischen
1	0	1	1	0	An, ganz rechts
1	1	0	0	1	An, ganz links
1	1	1	?	?	„unlogisch“ → Fehler

Fehler im System: Fehleridentifikation

- Beispiel: Scheibenwischer → Entwicklung als Zustandsautomat



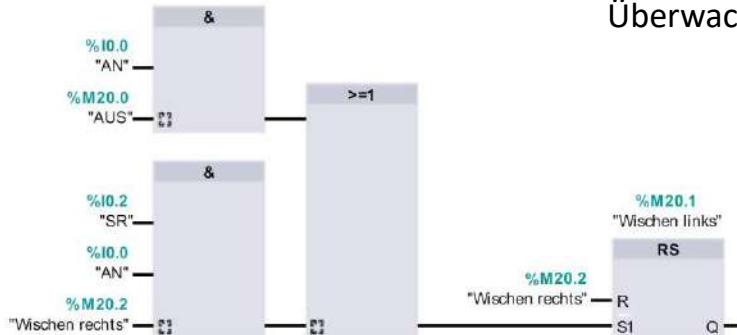
Fehler im System: Fehleridentifikation

- „Unlogische“ Zustände, wie gleichzeitig ansprechende linke und rechte Endlagengeber oder hohe Temperaturwerte trotz abgeschalteter Heizung, lassen sich bei einigem Nachdenken über die kombinatorischen Möglichkeiten als spezielle Kombination der binären Eingänge erkennen. → Vergleiche Scheibenwischerbeispiel
- Die Überwachung des zeitlichen Verhaltens des Ablaufs liefert weitere Ansatzpunkte zur Fehlererkennung. Typische Ansatzpunkte sind Schritte, in denen Ausgänge geschaltet und dann im Schritt verweilt wird, bis ein oder mehrere Eingänge angesprochen haben. Beim Scheibenwischer sollte das Wischen nur eine bestimmte Zeit dauern und damit die Steuerung nicht länger als diese Zeit in den Schritten Wischen rechts oder Wischen links bleiben. Wenn die Steuerung länger als die festgesetzte Zeit in einem der Schritte verharrt, ist wahrscheinlich der Sensor, der ansprechen soll, oder der zugehörige Aktor defekt oder der mechanische Weg blockiert.

Fehler im System: Fehleridentifikation

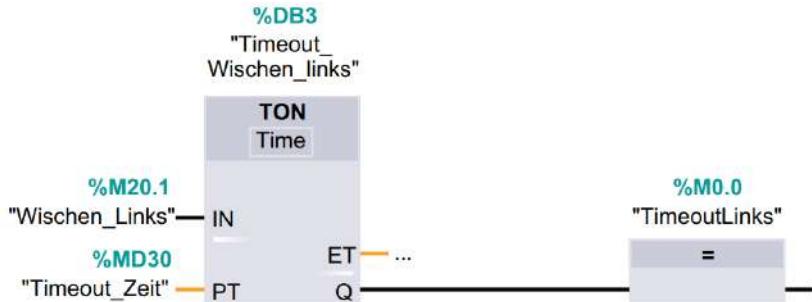
■ Timeouts

Netzwerk 1: Zustand Wischen links



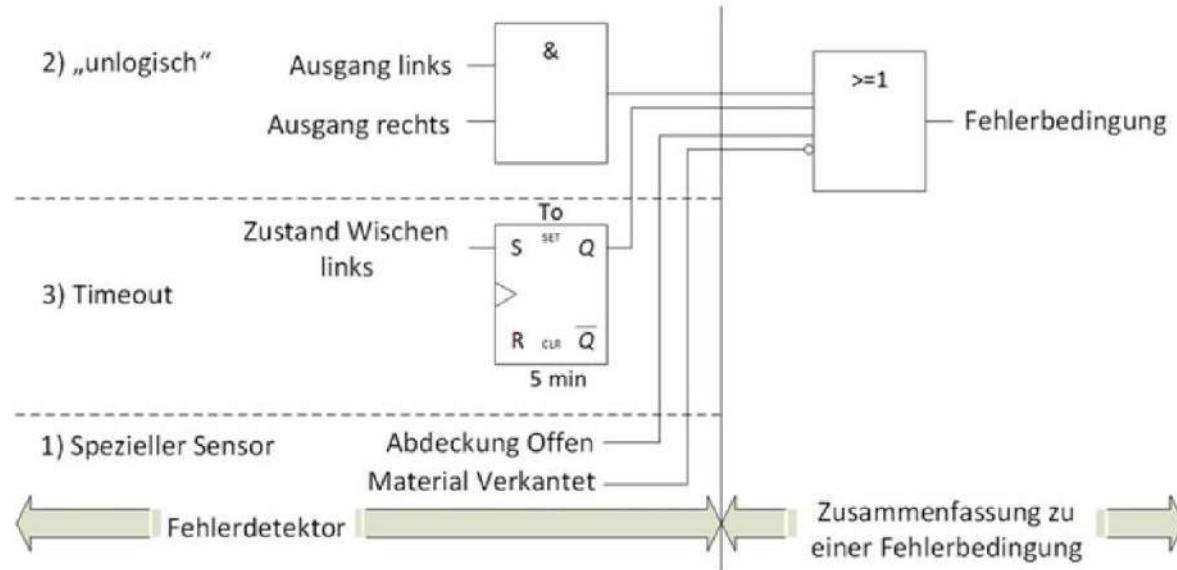
Die Zeitüberwachung erlaubt es, auch komplexe Fehler ohne zusätzliche Sensorik zu erkennen. Die einzige Hürde besteht darin, dass Vorgabezeiten für die zu überwachende Vorgangsdauer bestimmt werden müssen. Diese können beispielsweise bei der Inbetriebnahme der Anlage gemessen und dann in der Software abgelegt werden. Programmierer bezeichnen eine derartige Überwachung auch als „timeout“.

Netzwerk1:Timeout

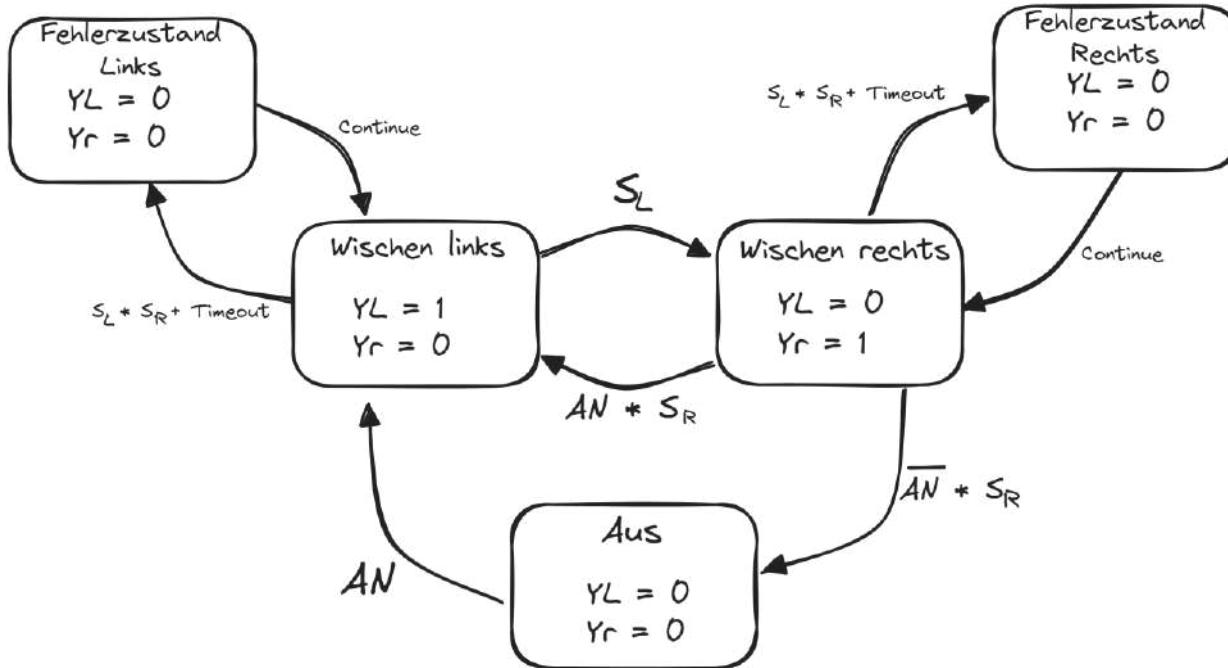


Fehler im System: Fehleridentifikation

- Zusammenfassen von Fehlerdetektoren zu einer Fehlerbedingung



Implementierung von Fehlerzuständen



Zwischenfazit

- Zusätzlich eingebaute Sensoren zur Fehlererkennung werden in der Praxis eher sparsam eingesetzt, denn ein derartiger Sensor muss beschafft, montiert und justiert werden.
- Unabweisbare Gründe für deren Einsatz liegen im Schutz von Mensch, wie Lichtgitter in Aufzugtüren, oder Maschine, wie bei der Überwachung der Kühlwassertemperatur im Auto.
- Sensorik zum Schutz von Menschen fällt im Allgemeinen in die Kategorie „sicherheitsgerichtete Bauteile“ und unterliegt strenger Regulierung.
 - Funktionale Sicherheit ist die Fähigkeit eines Automatisierungssystems bei Auftreten zufälliger und/oder systematischer Ausfälle mit gefahrbringender Wirkung im sicheren Zustand zu bleiben bzw. einen sicheren Zustand einzunehmen.

Sicherheitsrelevante Systeme



IEC 62061 - Zuverlässigkeit eines sicherheitsrelevanten Steuerungssystems

- Die IEC 62061 befasst sich mit der Frage, wie zuverlässig ein sicherheitsrelevantes Steuerungssystem sein muss.
 - Kombination aus einer Matrix und einem quantitativen Ansatz
 - Validierung von Sicherheitsfunktionen auf der Grundlage von strukturellen und statistischen Methoden
- Für jedes Risiko, das ein sicherheitsrelevantes Steuerungssystem erfordert, muss das Risiko abgeschätzt und die vom Steuerungssystem abhängige Risikominderung (SIL) definiert werden.
 - SIL → Sicherheits-Integritätslevel
- Das mit der Sicherheitsfunktion im Zusammenhang stehende Risiko wird gemäß IEC 62061 unter Berücksichtigung der folgenden Parameter abgeschätzt:
 - Schwere der Verletzung (S)
 - Häufigkeit und Dauer der Gefährdungsexposition (F)
 - Wahrscheinlichkeit des Auftretens eines gefahrbringenden Ereignisses (W)
 - Möglichkeit zu Vermeidung oder Begrenzung des Schadens (P)

Klassifikation des SIL nach der IEC 62061

■ Klassifikation der Schwere (S)

Auswirkung	Schwere (S)
irreversibel: Tod, Verlust eines Auges oder Arms	4
irreversibel: gebrochene Gliedmaßen, Verlust eines/mehrerer Finger	3
reversibel: Behandlung durch einen Mediziner erforderlich	2
reversibel: Erste Hilfe erforderlich	1

■ Klassifikation der Häufigkeit und Dauer der Exposition (F)

Häufigkeit der Exposition	Dauer (F) <= 10 min	Dauer (F) > 10 min
≥ 1 pro h	5	5
< 1 pro h bis ≥ 1 pro Tag	4	5
< 1 pro Tag bis ≥ 1 alle 2 Wochen	3	4
< 1 alle 2 Wochen bis ≥ 1 pro Jahr	2	3
< 1 pro Jahr	1	2

Klassifikation des SIL nach der IEC 62061

■ Klassifikation der Wahrscheinlichkeit (W)

Wahrscheinlichkeit des Auftretens	Wahrscheinlichkeit (W)
sehr hoch	5
wahrscheinlich	4
möglich	3
selten	2
vernachlässigbar	1

■ Klassifikation der Möglichkeit, einen Schaden zu vermeiden oder zu begrenzen (P)

Möglichkeit der Vermeidung oder Begrenzung	Vermeidung und Begrenzung (P)
unmöglich	5
selten	3
wahrscheinlich	1

Klassifikation des SIL nach der IEC 62061

- Matrixzuordnung zur Ermittlung des erforderlichen SIL für eine Sicherheitsfunktion

Auswirkungen	Schwere S	Klasse K = F + W + P																
		3	4	5	6	7	8	9	10	11	12	13	14	15				
Tod, Verlust eines Auges oder Armes	4	SIL 1		SIL 2		SIL 2		SIL 3		SIL 3		SIL 3						
		PL _r b PL _r c		PL _r d		PL _r d		PL _r e		PL _r e		PL _r e						
permanent, Verlust von Fingern	3	AM		SIL 1		SIL 2		SIL 3		SIL 3		SIL 3						
		PL _r a		PL _r b		PL _r c		PL _r d		PL _r e		PL _r e						
reversibel, medizinische Behandlung	2	Kein SIL (oder PL) erforderlich				AM		SIL 1		SIL 2		SIL 2						
						PL _r a		PL _r b		PL _r c		PL _r d						
reversibel, Erste Hilfe	1	AM: andere Maßnahmen							AM		SIL 1		SIL 1					
									PL _r a		PL _r b		PL _r c					

Folgen der SIL-Klassifikation

- Die Wahrscheinlichkeit eines gefahrbringenden Ausfalls jeder Safety-Related Control Function (SRCF) als Folge gefahrbringender zufälliger Hardwareausfälle muss gleich oder kleiner als der in der Spezifikation der Sicherheitsanforderungen festgelegte Ausfallgrenzwert sein.

SIL	PFD	PFD als Potenz von 10	RRF	PFH	PFH als Potenz von 10	MTBF (h)
1	0,1...0,01	$10^{-1}...10^{-2}$	10...100	0,00001...0,000001	$10^{-5}...10^{-6}$	100000...1000000
2	0,01...0,001	$10^{-2}...10^{-3}$	100...1 000	0,000001...0,0000001	$10^{-6}...10^{-7}$	1000000...10000000
3	0,001...0,000 1	$10^{-3}...10^{-4}$	1 000...10 000	0,0000001...0,00000001	$10^{-7}...10^{-8}$	10000000...100000000
4	0,000 1...0,000 0 1	$10^{-4}...10^{-5}$	10 000...100 000	0,00000001...0,000000001	$10^{-8}...10^{-9}$	100000000...1000000000

- PFD: Abkürzung für probability of failure on demand, also Wahrscheinlichkeit für Versagen bei Anforderung;
- RRF: Abkürzung für risk reduction factor, also Faktor der Risikoverringerung;
- PFH: Abkürzung für Probability of failure per hour, also Wahrscheinlichkeit für Versagen je (Betriebs-)Stunde;
- MTBF: Abkürzung für Mean Time Between Failures, also mittlere (Betriebs-)Dauer zwischen zwei Ausfällen.

Folgen der SIL-Klassifikation

- Als Folge reduziert eine nach SIL 1 ausgelegte Sicherheitsfunktion das Anlagenrisiko um Faktor 10-100, eine nach SIL 2 ausgelegte Funktion das Anlagenrisiko um 100 – 1000, usw.
- Eine Sicherheitsfunktion (SIS – Safety Integrated Function) besteht grundsätzlich aus drei Teilen:
 - Einer Sensorik, die einen kritischen Zustand eines Anlagenteils erkennen muss (z. B. Druck- oder Temperatursensoren zur Erkennung von Überdruck oder einer kritischen Temperatur)
 - Einer Sicherheitssteuerung, die aufgrund der Sensorik den kritischen Anlagenzustand erkennt, bewertet und ggf. Sicherheitsmaßnahmen einleitet, die zum sicheren Zustand der Anlage führen.
 - Einer Aktorik, die das Erreichen eines sicheren Anlagenzustands ermöglicht. (z. B. Schütze zum Spannungsfreischalten von Anlagenteilen, oder Absperrventile zum Stoppen des Materialzuflusses)

MooN-Architekturen

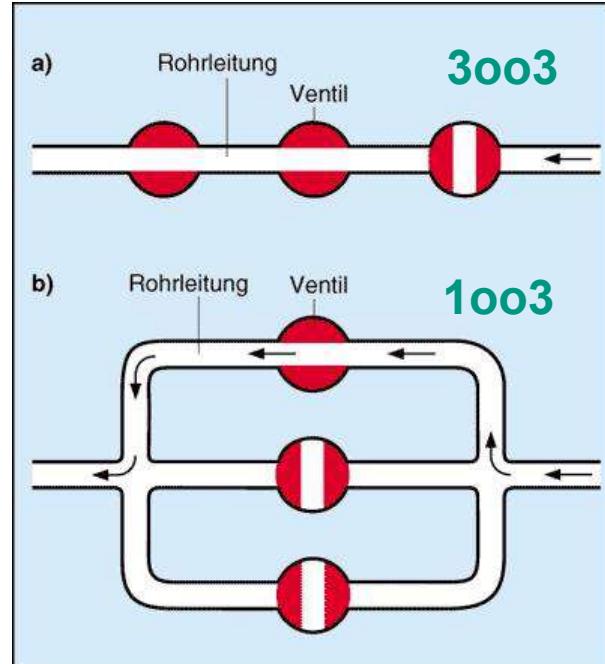


MooN (M out of N – Architekturen)

- Beim MooN-Prinzip („M out of N“) müssen mindestens M von N Komponenten korrekt funktionieren, damit das System eine sicherheitsrelevante Aktion ausführt oder in einem sicheren Zustand bleibt.
 - **M** = Anzahl der Komponenten, die **gleichzeitig korrekt arbeiten müssen**.
 - **N** = Gesamtanzahl der vorhandenen Komponenten (z. B. Sensoren, Steuerungen).
 - Das Ziel ist es, durch **Redundanz** entweder die **Sicherheit** oder die **Verfügbarkeit** zu erhöhen – je nach gewählter MooN-Konfiguration.
- MooN beschreibt **nicht** direkt die Sicherheit, sondern die Logik, wie viele Komponenten übereinstimmen müssen, damit das System reagiert.
 - Ob ein System sicher oder verfügbar ist, hängt davon ab, wie M und N gewählt sind und wie das System auf Fehler reagiert.

MooN (M out of N – Architekturen)

- Beispiel a: Der Durchfluss ist **nur möglich**, wenn **alle drei Ventile geöffnet** sind.
 - **MooN-Prinzip: 3oo3** („Three out of Three“)
 - **Alle 3 Komponenten müssen funktionieren**, damit das System arbeitet.
 - **Eigenschaften:**
 - ✓ **Hohe Sicherheit** (ein Fehler → kein Durchfluss)
 - ⚠ **Geringe Verfügbarkeit** (ein defektes Ventil → System blockiert)
- Beispiel b: Der Durchfluss ist möglich, **wenn mindestens eines der Ventile geöffnet** ist.
 - **MooN-Prinzip: 1oo3** („One out of Three“)
 - **Nur 1 von 3 Komponenten muss funktionieren**, damit das System arbeitet.
 - **Eigenschaften:**
 - ✓ **Hohe Verfügbarkeit** (zwei Ventile können ausfallen)
 - ⚠ **Geringe Sicherheit** (ein defektes Ventil könnte unbemerkt bleiben)



MooN (M out of N – Architekturen)

■ Weitere Beispiele: 1oo2

- Zwei unabhängige Drucksensoren überwachen einen Reaktor
- Beide Sensoren sind an eine Sicherheitssteuerung angeschlossen.
- Die Steuerung ist so konfiguriert, dass **bereits ein Sensor** mit einem kritischen Wert die **Abschaltung** auslöst.
- Nur **einer von zwei** Sensoren muss einen Fehler oder Grenzwert erkennen, damit das System reagiert.
- Das System geht bei einem Fehler in einen sicheren Zustand (z. B. Abschaltung, Druckentlastung).

MooN (M out of N – Architekturen)

- Weitere Beispiele:
 - 2003: Dreifache Verarbeitung mit Verwendung des Mehrheitsergebnisses:
Voting ('2 von 3'-Wähler):
 - Drei **Lagesensoren** (Gyroskope) messen die Fluglage (z. B. Neigung, Gier, Rollwinkel).
 - Die Sensoren arbeiten **unabhängig** voneinander.
 - Die Flugsteuerung vergleicht die Werte:
 - **Mindestens zwei Sensoren müssen übereinstimmen**, damit das System die Werte akzeptiert und steuert.
 - Beurteilen Sie die Sicherheit und Verfügbarkeit des Systems

MooN (M out of N – Architekturen)

Architektur	Bedeutung	Sicherheit	Verfügbarkeit	Typische Anwendung
1oo1	1 von 1	Niedrig	Hoch	Einfache Steuerungen
1oo2	1 von 2	Mittel	Hoch	Prozessüberwachung mit Alarm
1oo2D	1 von 2 + Diagnose	Hoch	Hoch	SIL 2–3 Anwendungen
2oo2	2 von 2	Sehr hoch	Niedrig	Not-Aus, Reaktorschutz
2oo3	2 von 3	Hoch	Hoch	Kritische Systeme mit Voting
3oo3	3 von 3	Extrem hoch	Sehr niedrig	Höchstsicherheitsbereiche

MooN (M out of N – Architekturen)

- Gängige Auslegungen:
 - 1oo1 Einkanalige Architektur ohne Redundanz ($HFT = 0$)
 - 1oo2 Fahrerlose Transportsysteme ($HFT = 1$)
 - 2oo3 Flugzeuge, Chemische Anlagen ($HFT = 1$)
 - 2oo4 Raumfahrt, Atomkraftwerke ($HFT = 2$)
- Hardware-Fehlertoleranz: $HFT = N - M \rightarrow N+1$ gleichzeitige Hardwarefehler führen zum Verlust der definierten Sicherheitsfunktion.
Anders ausgedrückt: Ein System mit $HFT = 0$ kann bereits durch einen einzelnen Fehler die Sicherheitsfunktion verlieren und dadurch das Gesamtsystem ausfallen, während $HFT = 1$ bedeutet, dass ein Fehler noch abgefangen wird und erst beim zweiten Fehler die Sicherheitsfunktion versagt. Hohe Sicherheitsanforderungen erfordern folglich $HFT > 0$, was typischerweise durch redundante Hardware erreicht wird.

Abschlussdiskussion zu MooN

- In einer Architektur mit zwei Sensoren, bei der grundsätzlich einer ausfallen darf, sprechen wir von einer 1-out-of-2 (1oo2) MooN-Architektur.
→ Ein Sensor muss funktionieren, damit das System eine gültige Entscheidung treffen kann.
- Was passiert, wenn beide Sensoren unterschiedliche Werte liefern?
 - Das ist ein klassisches Problem bei redundanten Systemen
 - In der Praxis gibt es dafür mehrere Strategien:
 - **Voting-Mechanismus (z. B. bei 2oo3-Systemen):**
 - Bei drei Sensoren kann man Mehrheitsentscheidungen treffen.
 - Bei zwei Sensoren (1oo2) ist das nicht möglich – es gibt nur zwei Meinungen.
 - **Diagnose-Logik:**
 - Wenn die Werte zu stark voneinander abweichen, wird ein Fehler gemeldet.
 - Das System geht in einen sicheren Zustand (z. B. Abschaltung oder Alarm).

Abschlussdiskussion zu MooN

- Diagnose-Logik:
 - Wenn die Werte zu stark voneinander abweichen, wird ein Fehler gemeldet.
- Unterschied zwischen 1oo2D und 2oo2:
 - 1oo2D:
 - Sensor A misst 75 °C, Sensor B misst 76 °C → innerhalb Toleranz → System läuft weiter.
 - Sensor A misst 75 °C, Sensor B misst 120 °C → Diagnose erkennt Fehler → sicherer Zustand.
 - **1oo2D versucht, Verfügbarkeit und Sicherheit zu kombinieren:** Es erlaubt Weiterbetrieb bei einem Fehler, **solange keine Inkonsistenz erkannt wird.**
 - 2oo2:
 - Sensor A misst 75 °C, Sensor B misst 76 °C → nicht exakt gleich → System geht in sicheren Zustand, auch wenn beide Sensoren technisch funktionieren.
 - **2oo2 ist kompromisslos sicher:** **Jede Abweichung oder jeder Fehler führt zur Abschaltung.**

Abschlussbeispiel

- In einer Chemieanlage wird ein Reaktor betrieben, dessen Temperatur kritisch überwacht werden muss. Ein unkontrollierter Temperaturanstieg kann zu einer Explosion führen. Daher ist eine sichere Abschaltung bei Übertemperatur zwingend erforderlich.
- Technische Umsetzung:
 - Sensorik: 3 Temperaturfühler (T1, T2, T3) messen unabhängig voneinander die Reaktortemperatur.
 - Steuerung: Eine Sicherheits-SPS wertet die Sensoren aus.
 - Aktorik: Ein Not-Aus-Ventil unterbricht die Reaktion bei Gefahr.
- MooN-Architektur:
 - Zwei von drei Sensoren müssen eine Übertemperatur melden, damit das System abschaltet.
 - Vorteil: Fehlertoleranz bei einem defekten oder fehlerhaften Sensor.
 - Voting-Logik: Verhindert unnötige Abschaltungen durch einzelne Ausreißer.

Aspekt	Erklärung
Sicherheit	Nur bei bestätigter Gefahr (2 von 3) wird abgeschaltet.
Verfügbarkeit	Ein Sensor darf ausfallen, ohne dass das System stoppt.
Diagnosefähigkeit	Abweichende Sensorwerte können erkannt und gemeldet werden.
Praxisrelevant	Häufig in der Prozessindustrie, z. B. Chemie, Pharma, Energie.

Informations- und Automatisierungstechnik

Ankündigungen zur Klausur im Sommersemester 2025

Univ.-Prof. Dr.-Ing. Mike Barth

Ablauf der Klausur

- Klausur findet vollständig in ARTEMIS statt.
- Klausur findet am 08.09.2025 in den Poolräumen des SCC statt
- 4 Räume à 40-55 Plätze (+ 1 zus. Raum mit Ersatzrechnern)
- Aufgrund der Platzbegrenzung werden zwei Gruppen gebildet
 - Gruppe 2 schreibt die Klausur direkt im Anschluss an Gruppe 1
- Die Bearbeitungszeit der Klausur beträgt 120 Minuten
- Es sind keine Hilfsmittel erlaubt, außer:
 - Formelsammlung: 1 DIN-A4-Blatt (2 Seiten)
 - Handgeschrieben oder digital erstellt
 - Falls am Rechner erstellt: muss als ausgedrucktes Blatt zur Klausur mitgebracht werden

Festlegungen

- Ihr Zeitslot in der Klausur ist **verbindlich** und kann **nicht** geändert werden!
 - MIT-Studierende werden aufgrund einer Parallelklausur **vornehmlich** im zweiten Block eingeteilt (HM1 8:00-10:00 Uhr)
- Falls während der Klausur technische Probleme auftreten, bestehen Ausweichmöglichkeiten → Melden Sie sich **umgehend** bei der Aufsicht. Verlorene Arbeitszeit kann erstattet werden!
- Ihr Arbeitsfortschritt wird permanent gespeichert!
- Die Abgabe der Klausur erfolgt durch:
 - Ablauf der Bearbeitungszeit oder
 - durch Ihre vorzeitige Abgabe per Button-Klick in Artemis.

Probeklausur

- Klausur findet am 01.08.2025 in den Poolräumen des SCC statt
- Aufteilung: Gebäude 20.21
 - 17.30 Uhr: A - K (Kommen Sie spätestens um 17.15 Uhr im SCC Foyer)
 - 18:30 Uhr: L – Z (Kommen Sie spätestens um 18.15 Uhr im SCC Foyer)
- **Sie benötigen die Zugangsdaten für Ihren KIT-Account!!!**
- Es handelt sich nicht um eine vollständige Klausur. Es geht insbesondere um die Organisation und Prozedur:
 - Kennenlernen Räume; Log-In in Artemis; Start Übungsklausur; Bearbeitung Aufgaben; Abgabe
 - sowie das Kennenlernen der Aufgabentypen in Artemis
- Vollständige Alt-Klausuren finden Sie im ILIAS-Kurs.

Aufgabentypen in Artemis

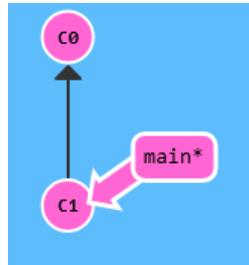
- Programmieraufgaben
 - Programmierung im Artemis-Online-Editor
 - Feedback während der Klausur:
 - Compile-Test-Ergebnisse werden angezeigt
→ Kann das Programm kompiliert und ausgeführt werden
 - Tests zur eigentlichen Fragestellung sind versteckt
→ Ergebnisse erst nach der Klausur-Korrektur einsehbar
- Interpretationsaufgaben
 - Sie bekommen einen Programmausschnitt und müssen die Funktion analysieren.
 - Eingabe von Freitext.
- Modellierungsaufgaben (UML) → vgl. Übungen
- Wissensfragen (Multiple-Choice / Freitextantworten)

Verständnisfragen zu Vorlesungsinhalten

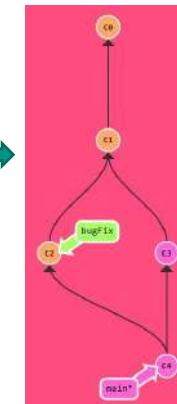
- Was wird unter einem Pointer verstanden?
- Was kennzeichnet eine Klasse (OOP)?
- Was unterscheidet Smart Pointer von herkömmlichen Pointern?
- Was kennzeichnet die Von-Neumann-Architektur?

Anwendung von Methoden aus Vorlesung/Übung

- Geben Sie die git-Befehle an, um den linken git-Graphen in den rechten zu überführen.
 - Das Verändern von Dateien zwischen den commits und das Hinzufügen zum staging Bereich mit git add dürfen sie weglassen.
 - Die commit Nachrichten dürfen Sie beliebig wählen



- git checkout -b bugFix
- git commit -m „beliebig“
- git checkout main
- git commit -m „beliebiger“
- git merge bugFix



- Erklären Sie den Befehl git commit / merge / ...

Anwendung von Methoden aus Vorlesung/Übung

- Fertigen Sie ein UML-Klassendiagramm an, welches eine sinnvolle Modellierung der folgenden Beschreibung umsetzt:
 - Szenario:
 - Ein Unternehmen produziert Waschmaschinen.
 - Eine Waschmaschine besteht aus einer Klappe und einer Trommel (Komposition).
 - Sie nutzt Ressourcen wie Wäsche, Strom und Wasser (Assoziationen).
 - Jede Waschmaschine besitzt die Eigenschaften: Masse, Farbe, Fassungsvermögen und eine eindeutige Seriennummer.
 - Sie kann gestartet, beendet und mit verschiedenen Waschprogrammen betrieben werden (Methoden).
 - Es gibt eine Waschmaschinenmarke namens „Barth“ unter der zwei unterschiedliche Modelle geführt werden: Barthmaschine A, B und C. Die Modelle unterscheiden sich in Front- und Top-Loader.
 - Anforderungen an das UML-Diagramm:
 - Modellieren Sie alle relevanten Klassen mit sinnvollen Attributen und Methoden.
 - Stellen Sie die Beziehungen zwischen den Klassen korrekt dar (z. B. Vererbung, Komposition, Assoziation).
 - Verwenden Sie geeignete Sichtbarkeiten (+, -, #) für Attribute und Methoden.
 - Nutzen Sie Vererbung, um die zwei Barthmaschinen-Modelle als Spezialisierungen darzustellen.
 - Achten Sie auf eine saubere und übersichtliche Darstellung gemäß dem UML -Standard.

Anwendung von Methoden aus Vorlesung/Übung

- Fügen Sie die Werte 39, 28, 38 und 0 der Hashtabelle mit doppeltem Hashing hinzu. Verdeutlichen Sie die Schritte rechnerisch und graphisch.
 - $h(k) = (h_1(k) + h_2(k) \cdot i) \bmod m$
 - $h_1(k) = k \bmod m$
 - $h_2(k) = 1 + (k \bmod m')$

0	39
1	1
2	15
3	
4	17
5	
6	
7	
8	8
9	
10	23
11	
12	

C++ - Codeinterpretation

- Benennen Sie 4 Code-Zeilen, in denen ein Fehler vorhanden ist und erklären Sie diese.

Code-Zeile	Grund des Fehlers
19	B hat kein Attribut num
28	Falscher Konstruktorauf调用
42	Methode print() von B ist private und kann nicht aufgerufen werden
44	a ist zwar ein Pointer auf C, wird aber in einen Pointer auf A gespeichert und set_ac kann somit nicht (ohne casting) aufgerufen werden
46	Pointer auf A kann nicht im Typ A gespeichert werden
48	Methode get_num in C ist private

```
#include <string>
#include <iostream>

class A {
protected:
    int num = 0;

public:
    int get_num() { return num; }
};

class B {
    std::string str;

    void print() {
        std::cout << str << std::endl;
    }

public:
    B(const std::string &s) {
        str = s;
        this->num = 1;
    }
};

class C : public A {
    A a_obj;
    int get_num() { return num; }

public:
    A() {
    }

    void set_ac(C* value) {
        a_obj = *value;
        this->num = value->get_num();
    }

    void set_ab(B *value) {
        a_obj = *value;
    }
};

int main(int argc, const char** argv) {
    A* a = new C;
    B b("Hallo Welt!");
    b.print();
    std::cout << a->get_num() << std::endl;
    a->set_ac(new C);
    A a2 = new A;
    C c;
    return c.get_num();
}
```

C++ - Codeinterpretation

- Ordnen Sie die folgenden Zeilen in der richtigen Reihenfolge, sodass sich ein lauffähiges Programm mit folgender Ausgabe ergibt:

!!!

Mein erstes Programm!!!

Mit einer zweiten Ausgabezeile

!!!

```
1: return 0;
2: std::cout << std :: endl;
3: int main ()
4: std::cout << " !!! " << std::endl;
5: {
6: //Dies ist die Hauptfunktion
7: #include <iostream>
8: std::cout << std::endl << " !!! ";
9: std::cout << "\tMein erstes Programm";
10: }
11: /* Die ganze Funktionalitaet dieses Programms */
12: std::cout << "\tMit einer zweiten Ausgabezeile" << std::endl;
13: std::cout << " !!! " << std::endl;
```

Transferfragen

- Wählen Sie einen passenden primitiven Datentyp und definieren Sie eine Variable *value* von diesem in C++ Syntax. Verwenden Sie für Ganzzahlen den Integer-Typ mit kleinstmöglichen Wertebereich und wenn möglich vorzeichenlose Typen. Begründen Sie Ihre Antwort.
 - Stunde einer Uhrzeit `unsigned short value;`
 - Luftdruck im Fahrradreifen `float value;`
 - Rückgabewerte einer Funktion *isFull*, die prüft, ob der Kaffeesatzbehälter voll ist `bool value = isFull();`

Systemisches Wissen

- Was ist die Ausgabe des Programms?
- Nennen und erklären Sie 3 Konzepte der OOP, die hier angewendet werden.
- Welche Ergänzung müssten Sie vornehmen, um Matrikelnummern von Studierenden im Nachhinein ändern zu können. Begründen Sie die Sinnhaftigkeit ihre Antwort.
 - Setter-Funktion
 - Matrikelnummer → public, da kein Problem für Datenkapselung

```
#include <iostream>
#include <string>

class Person{
public:
    Person(std::string name, std::string forename){
        this->m_name=name;
        this->m_forename=forename;
    }

    void display(){
        std::cout<<"Die Person heißt: "<<m_forename<<" "<<m_name<< std::endl;
    }
protected:
    std::string m_name{};
    std::string m_forename{};
};

class Student:public Person{
public:
    Student(std::string name, std::string forename, int matriculationNum):
    Person(name,forename), m_matriculationNum(matriculationNum){};

    void display(){
        std::cout<<"Der Student "<<m_name<<" hat die Matrikelnummer:
        "<<m_matriculationNum << std::endl;
    }
private:
    int m_matriculationNum;
};

int main() {
    Person P1("Monika", "Mustermann");
    P1.display();
    Student stud_1("Hans", "Maier", 123445);
    stud_1.display();
    return 0;
}
```

Vorstellung Aufgabentypen für den Klausurteil AT



Prof. Dr.-Ing. Mike Barth
M.Sc. Marwin Madsen (Übung)

Verständnisfragen zu Vorlesungsinhalten

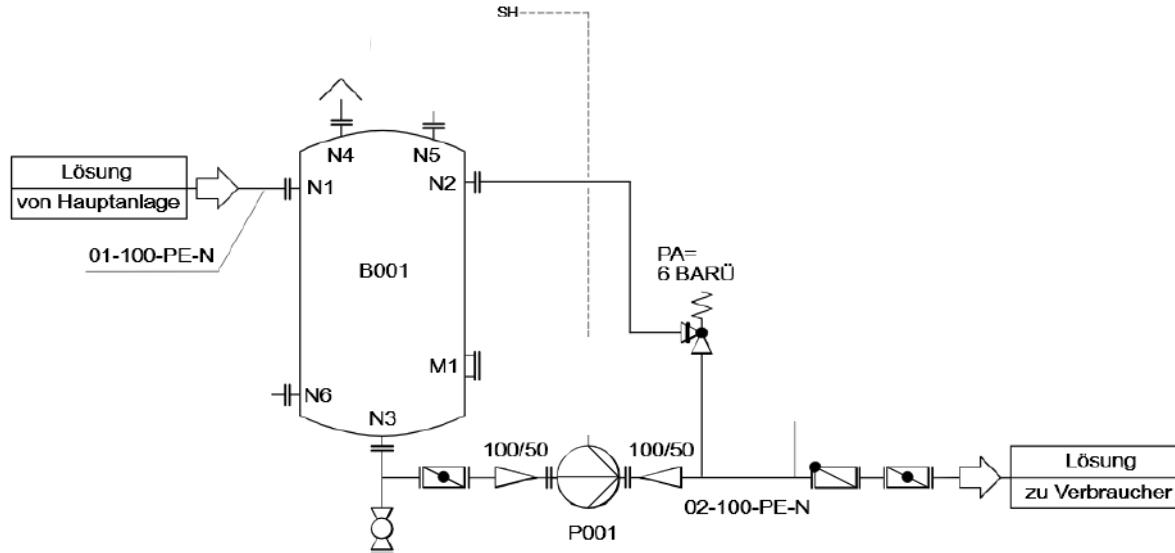
- Was verstehen Sie unter einer MoN – Architektur in der funktionalen Sicherheit von Automatisierungssystemen?
 - Erläutern Sie eine 1oo2 Architektur mit Beispiel.
- Welche Ebenen beinhaltet die AT-Pyramide? Beschreiben Sie diese mit jeweils einem Beispiel.
- Aus welchen grundlegenden Elementen besteht jedes Automatisierungssystem?
- Wie bildet sich die OOP in der AT ab? Erläutern Sie dies an einem konkreten Beispiel mit Aspekten der IEC 61131-3.
- Welche Aufgabe haben Interfaces in der IEC 61131-3?
- Nennen und erläutern Sie drei Aufgaben der Automatisierungstechnik.

Verständnisfragen zu Vorlesungsinhalten

- Was versteht man in der Automatisierungstechnik unter parallelen Prozessen und wie werden diese wieder synchronisiert? Erläutern Sie dies anhand einer Skizze in der Ablaufsprache.
- Erläutern Sie den Unterschied zwischen N, S und R beim Setzen von Ausgängen in AS.
- Ist ein Eingang in einen Funktionsbaustein (z.B. in FUP) standardmäßig Call-by-Value oder Call-by-Reference?
 - Welche Variablen Deklaration führt zu Call-by-Reference?

Anwendung von Methoden aus Vorlesung/Übung

- Im abgebildeten Kessel B001 sollen 1000 Liter einer Flüssigkeit erhitzt, gerührt und unter Druck gesetzt werden. Ergänzen Sie die notwendigen Sensoren und Aktoren zum System. Führen Sie hierzu Bezeichnungen für die Sensoren ein und ergänzen eine kurze Legende.

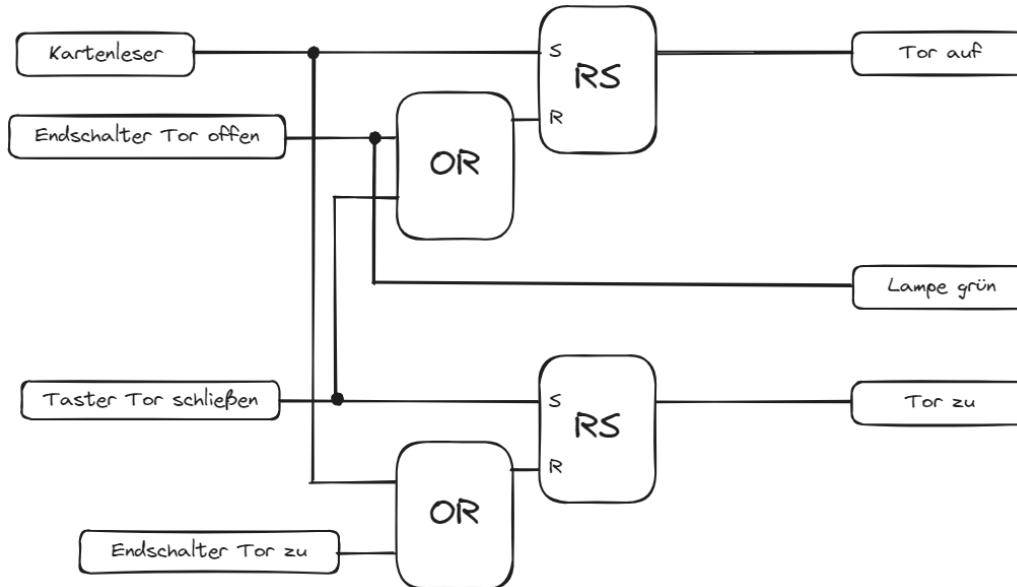


Anwendung von Methoden aus Vorlesung/Übung

- Überführen Sie die folgende Spezifikation in Kontaktplan.
 - Ein Ausgang A1 soll geschaltet werden, wenn der Taster T1 betätigt und der Taster T2 nicht betätigt wird. A1 kann auch geschaltet werden, wenn der Taster T3 betätigt wird.
- Überführen Sie die folgende Spezifikation in CFC.
 - Ein Badlüfter B1 soll eingeschaltet werden, sobald der Schalter S1 eingeschaltet wird. Wenn S1 wieder ausgeschaltet wird, soll B1 noch 30 Sekunden laufen.
 - Ein Taster T4 soll im Bad eine Handtuchheizung H1 aktivieren, wenn dieser 2 Sekunden lang gedrückt wurde. H1 soll nach Aktivierung dann 30 Minuten aktiv sein.

Interpretation der IEC 61131-3

- Interpretieren Sie die folgende Steuerung eines Werkstores. Erstellen Sie die Textspezifikation für die dargestellte Implementierung.

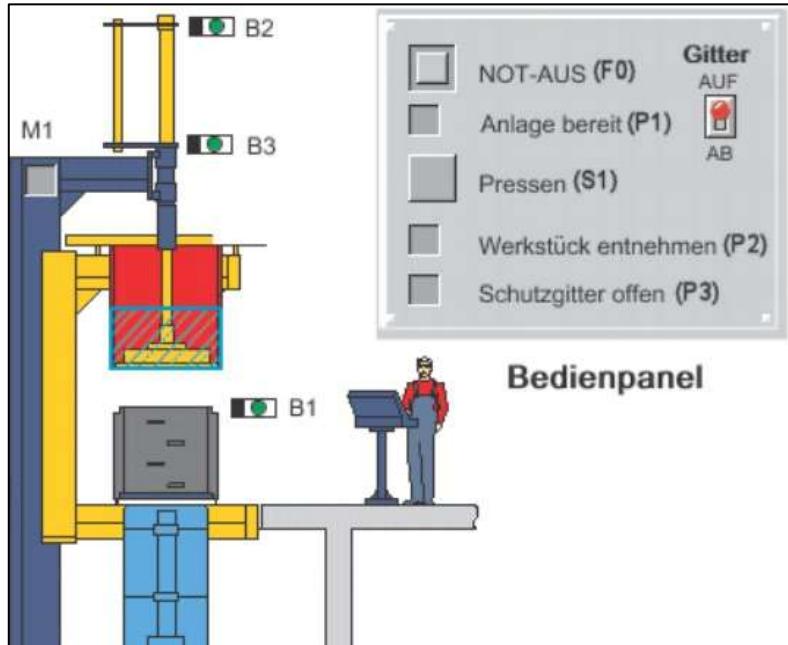


Transfer von Vorlesungs-/Übungswissen

- Eine industrielle Anlage soll 15 Jahre lang ein und dasselbe Produkt herstellen (Single-Purpose). Die Anlage ist nicht sonderlich komplex (wenige Sensoren und Aktoren) und besteht aus einem Hauptprozess. Die Anlage selbst hat einen Umfang von lediglich wenigen Metern. Sie ist nicht mit anderen Produktionsprozessen verbunden.
 - Wählen Sie eine geeignete AT-Architektur aus.
- Die Sensoren und Aktoren in der Anlage basieren auf der 420mA Technologie.
 - Wählen Sie eine geeignete AT-Kommunikationsarchitektur aus.

Transfer von Vorlesungs-/Übungswissen

- Die dargestellte Presse wird durch die in der Tabelle gelisteten Signale automatisiert. Erstellen Sie ein Programm in AS, welches die Steuerung der Presse ermöglicht.
- Wenn S1 betätigt wird soll zunächst das Schutzbretter und danach die Presse nach unten fahren.
- Die Presse fährt nachdem sie unten war nach einer Sekunde selbst wieder nach oben. Wenn die Presse oben angekommen ist fährt das Schutzbretter wieder nach oben.
- Berücksichtigen Sie auch die Leuchtmelder als Statussignale an das Bedienpersonal.



Symbol	Adresse	Typ	Beschreibung
M1	A4.0	BOOL	Ventil zum Zylinder „Presse nach unten fahren“
P1	A4.1	BOOL	Leuchtmelder, weiß, „Anlage bereit“
P2	A4.2	BOOL	Leuchtmelder, weiß, „Werkstück entnehmen“
P3	A4.3	BOOL	Leuchtmelder, weiß, „Schutzbretter offen“
F0	E0.0	BOOL	Not-Aus-Relais F0 freigegeben
S1	E0.1	BOOL	Taster "Pressen" (S)
B1	E0.2	BOOL	Sensor Schutzbretter geschlossen (S)
B2	E0.3	BOOL	Sensor Presse oben (S)
B3	E0.4	BOOL	Sensor Presse unten (S)

Transfer von Vorlesungs-/Übungswissen

- Welche MooN-Architektur ist in der abgebildeten Schaltung dargestellt, und wie lässt sich dies anhand des Stromflusses und der Schalterstellung erklären?

