

Zusammenfassung IAUT

Eric Lindegger – HS25

Danke fürs Herunterladen und Verwenden meiner IAUT Zusammenfassung! Hoffe ihr findet sie nützlich. Falls ihr Fehler findet oder Inputs habt, schreibt mir doch auf eric.lindegger@stud.hslu.ch

Inhaltsverzeichnis

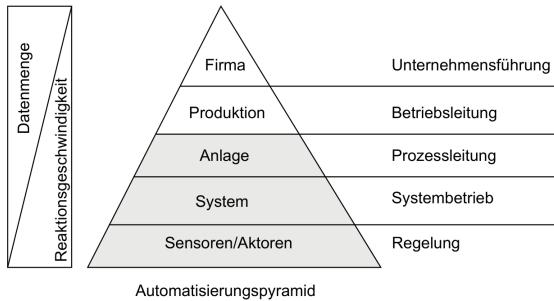
1 Automatisierung	2
1.1 Steuerung - Überblick	2
1.1.1 Klemmen	2
1.2 Allgemeine Aufgaben	2
1.3 Allgemeine Anforderungen	2
2 SPS	3
2.1 Feldbus	3
2.1.1 Spezifische Feldbusse	3
3 Programmierung	3
3.1 EN 61499	3
4 Digitale Regelung	3
5 FSM - Finite State Machine	3
6 ST-Language	3
6.1 Grundaufbau	3
6.1.1 Programm	4
6.1.2 Funktionsblock	4
6.1.3 Funktion	4
6.2 Variablentypen	4
6.2.1 System Datentypen	4
6.2.2 Array	4
6.2.3 DUT - Data Unit Type	4
6.2.4 Globale Variablen Liste (GVL)	5
6.2.5 Type Casting	5
6.3 Conditional Statements	5
6.3.1 If/Else	5
6.3.2 Switch-Case	5
6.4 Schleifen	5
6.4.1 WHILE	5
6.4.2 REPEAT	5
6.4.3 FOR	5
6.4.4 Schleifen Modifikatoren	5
7 TwinCat	5
7.1 Bitshifting	5
7.2 Limit	5
7.3 Timer	6
7.3.1 Timer Off-Delay (TOF)	6
7.3.2 Timer On-Delay (TON)	6
7.3.3 Timer Pulse Generator (TP)	6
8 Beispiel Code	6

1 Automatisierung

Definition Automatisierung

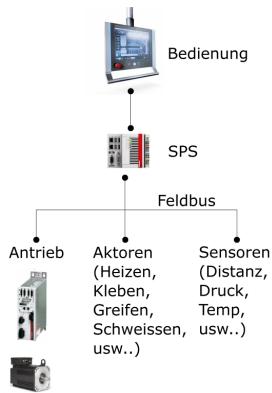
Unter Automatisierung (DIN19222) versteht man das gezielte Ausrüsten einer Einrichtung, so dass sie ganz oder teilweise ohne Mitwirkung des Menschen geschieht und arbeiten kann.

Die Automatisierungspyramide ist der Versuch die gesamte Automatisierungskette auf ein Bild abzubilden:



Die Darstellung ist wie folgt zu interpretieren: Je weiter unten, desto schneller müssen die Komponenten reagieren und je weiter oben, ist die Datenmenge grösser. Die Absicht der Automatisierung ist, dass die automatisierte Anlage bis ins ERP der Firma eingepflegt wird. So muss z.B. keine spezielle Benutzeroberfläche geladen werden um ein neuer Prozess auf der Anlage zu starten.

1.1 Steuerung - Überblick



Die Bedienung ist wo das GUI (Graphical User Interface) dargestellt wird. Hier werden vom Benutzer Eingaben getätigt, wie z.B. Job oder Rezeptwahl.

Der Controller innerhalb der Steuerung/ Anlage ist die CPU/SPS. Eine Automatisierung ist häufig eine (PID-)Regelungsaufgabe. Die SPS ist mit Aktoren und Sensoren verbunden:

- Sensoren liefern Informationen
- Aktoren lösen Aktion aus

Die Schnittstellen der Sensoren/Aktoren ist entweder eine Spannung oder einen Strom. Die Spannung ist jedoch eher weniger gängig im Moment, da die Spannung anfälliger auf Störeinflüsse ist als ein Strom. Die gängigsten Schnittstellen sind. Ein weiterer Vorteil der Stromschnittstelle ist, dass auch ein Kabelbruch detektiert werden kann, da der Strom nie 0 wird:

- $0V - 10V$
- $4mA - 20mA$

1.1.1 Klemmen

Die Klemmen stellen die Ein- und Ausgänge einer SPS dar und können auf der Feldebene der SPS über ein Bussystem erweitert werden:



Klemmen können über RS485 abgesetzt werden und an einem anderen Ort als direkt bei der SPS eingesetzt werden.

1.2 Allgemeine Aufgaben

Echtzeitanforderung
Regler
Prozessabläufen
usw.

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

1.3 Allgemeine Anforderungen

Echtzeitfähigkeit
Betriebssysteme
Speicherarchitektur

Ein- Ausgabe von Prozesssignalen
Aktoren
Sensoren
Kommunikationssysteme

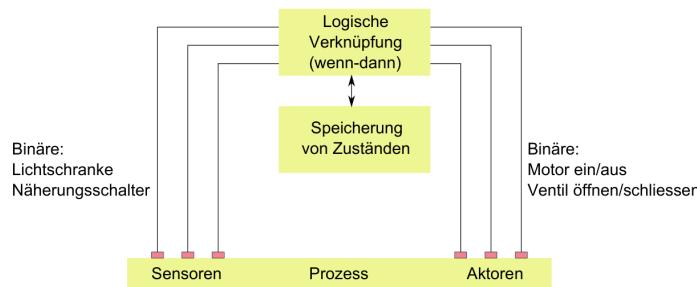
Sicherheit
Zuverlässigkeit
Hochwertige Komponente
Redundanz

Resistenz gegen Umwelteinflüsse
Spezielle Gehäuse
Schaltschrankmontage

2 SPS



Eine SPS ist eine 'Speicherprogrammierbare Steuerung'. Auf Englisch wird sie PLC (Programmable Logic Controller) genannt. Sehr wichtig: die SPS arbeitet konventionell möglichst auf if/else Statements und verarbeitet sämtlicher Code in jedem Zyklus einmal. Das Arbeitsprinzip ist:



2.1 Feldbus

Der Feldbus ist die Kommunikationsschnittstelle der SPS zu den Klemmen und entsprechend Ein- und Ausgängen. Es hängen alle E/A seriell an einem Bus. Wo der Feldbus aktiv ist, wird Feldebene genannt. Der Aufbau des Feldbusses kann wie folgt vorgestellt werden:



Definition Feldbus

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

Durch den Feldbus entstehen einige wesentliche Vor- und Nachteile:

- + geringer Verkabelungsaufwand
- + Erweiterungen oder Änderungen sind einfach
- Komplexität
- Preis
- Aufwendige Messgeräte (Analyzer)
- Längere Reaktionszeit

2.1.1 Spezifische Feldbusse

Im Gebäude sind die gängigen Feldbusse:

- LON
- EIB
- KNX

Innerhalb der Maschine werden andere Feldbusse verwendet:

- CAN/CANOpen
- Profibus, Profinet
- EtherCAT
- Ethernet/IP

3 Programmierung

Definition SPS-Programmierung

Die EN 61131-3 ist die einzige weltweit gültige Norm für Programmiersprachen von SPS-Steuerungen. Sie definiert die folgenden fünf Sprachen:

- AWL - Anweisungsliste (Assembler)
- KOP - Kontaktplan
- FBS - Funktionsbausteinsprache
- ST - Strukturierter Text
- AS - Ablausprache

Die einzige (relevante) Programmiersprache hinsichtlich des IAUT Unterrichts ist ST. Die anderen werden für einfache logische Verknüpfungen gebraucht, oder für z.B. I/O-Zuweisungen. ST wird in Europa oft für die Programmierung von SPS gewählt.

3.1 EN 61499

Diese Norm stellt eine objektorientierte Erweiterung der EN61131 dar.

4 Digitale Regelung

5 FSM - Finite State Machine

6 ST-Language

Dieses Kapitel fasst den Aufbau und Syntax der SPS-Programmiersprache ST zusammen.

6.1 Grundaufbau

Der Aufbau eines Programmes/Funktionsblockes oder Funktion ist immer in zwei Bereiche aufgeteilt: Deklaration und Implementation. Im Deklarationsteil werden alle Inputs, Outputs deklariert und ggf. initialisiert. In der Implementation wird der eigentliche Code geschrieben. Wichtig: die Implementation wird in jedem Clock Zyklus wiederholt.

```

1 // Deklarationsteil
2 VAR_INPUT
3   // input-variablen
4   iVarIn AT %I* : INT;
5 END_VAR
6 VAR_OUTPUT
7   // output-variablen
8   iVarOut AT %Q* : INT;
9 END_VAR
10 VAR_IN_OUT
11   // in/out-variablen
12 END_VAR
13 VAR
14   // lokale variablen
15   tmp : INT;
16 END_VAR
17
18 // Implementationsteil
19 IF iVarIn > 10 THEN
20   tmp := 0;
21 ELSE
22   tmp := iVarIn;
23 END_IF

```

6.1.1 Programm

Programme können als Funktionen aufgerufen werden und besitzen keinen Rückgabewert.

```

1 PROGRAM ProgramName
2
3     VAR
4         bTrig : BOOL;
5         iVal : INT;
6     END_VAR
7
8     IF(bTrig = TRUE) THEN
9         iVal := 15;
10    END_IF
11 END_PROGRAM

```

6.1.2 Funktionsblock

Ein Funktionsblock ist Zustandsbehaftet: Interne Variablen behalten ihren Wert über die SPS-Zyklen weg. Jede Instanz besitzt einen eigenen Speicher. Jedoch müssen die Speicher instanziert werden.

```

1 FUNCTION_BLOCK Count
2     VAR_INPUT
3         bTrig : BOOL;
4     END_VAR
5     VAR
6         iCount : INT;
7     END_VAR
8
9     IF bTrig = TRUE THEN
10        iCount := iCount + 1;
11    END_IF

```

6.1.3 Funktion

Eine Funktion kann ohne Instanzierung verwendet werden. Eine Funktion besitzt keine statische Werte, dafür einen Rückgabewert, welcher auf den Funktionsnamen zugewiesen wird.

```

1 FUNCTION fFunc : INT // Rückgabewert
2     VAR_INPUT
3         iTime : INT;
4         iT : INT;
5     END_VAR
6
7     iTmp := iTime;
8     fFunc := iTmp / iT;

```

6.2 Variablenarten

6.2.1 System Datentypen

Die gängigen Datentypen sind auch in ST vorhanden. Die Tabelle fasst die Datentypen zusammen:

Typ	Min	Max	Bits	Bemerkung
bool	0	1	8	Nur 1 / 0 / TRUE / FALSE sind gültig!
BYTE	0	255	8	
WORD	0	65535	16	
DWORD	0	4294967295	32	
SINT	-128	127	8	
USINT	0	255	8	
INT	-32768	32767	16	
UINT	0	65535	16	
DINT	-2147483648	2147483647	32	
UDINT	0	4294967295	32	
REAL	$-3.4 \cdot 10^{38}$	$-3.4 \cdot 10^{38}$	64	
LREAL	$-1.8E+308$	$1.8E+308$	64	
STRING	—	—	n+1	fügt Nullterminierung an.
TIME	T#0ms	T#71582m 47s295ms	32	
TOD	TOD#00:00	TOD#1193:02: 47.295	32	
DATE	D#1970-01-01	D#2106-02-06	32	
DT	DT#1970-0-0 -00:00	DT#2106-02-06- 06:28:15	32	

6.2.2 Array

```

1     VAR
2         aValues : ARRAY[0..99] OF INT;
3     END_VAR

```

6.2.3 DUT - Data Unit Type

Über DUT können benutzerdefinierte Datentypen angelegt werden. Die relevanten sind **STRUCT** und **ENUM**. Durch diese Typen kann ein Programm bedeutend lesbarer gestaltet werden. Für die Erstellung eines DUTs wird ein neues File in der DUT Ordnerstruktur erstellt. Die Definition geht wie folgt:

```

1 // Struct
2 TYPE stDataType:
3     STRUCT
4         iVar1 : INT;
5         iPointX : INT;
6         iPointY : INT;
7     END_STRUCT;
8     END_TYPE

```

```

1 // Enumeration
2 TYPE eSystemState:
3 (
4     None, // = 0 - default
5     init, // = 1
6     run
7 ) := init; // default state
8 END_TYPE

```

6.2.4 Globale Variablen Liste (GVL)

Die GVL deklariert Variablen, welche in jedem Programm, Funktionsblock und Funktion aufrufbar. Vielfach macht Sinn die In-/Outputs, welche mit der Hardware kommunizieren in einer GVL zu deklarieren.

6.2.5 Type Casting

VariablenTypen können mit Hilfe von Casting Funktionen umgewandelt werden. Fast alle Typen können gewandelt werden:

```
1 rRealVal := INT_TO_REAL(iVal);
```

Casting ist die zielführende Methode, um keine Fehler im Implementieren des Programms zu generieren. TwinCat hat implizite Casts nicht so gerne.

6.3 Conditional Statements

6.3.1 If/Else

```
1 IF <cond1> THEN
2   ...
3 ELSIF <cond2> THEN
4   ...
5 ELSE
6   ...
7 END_IF
```

6.3.2 Switch-Case

```
1 CASE iVar OF
2   1:           // do something for 1
3   2..5:        // do something for range 2 - 5
4 ELSE
5   // do this as default-assignment
6 END_CASE;
```

Die Switch-Bedingung kann auch für Enum-Typen gebraucht werden:

```
1 CASE eState OF
2   Init:
3     // handle init state
4   Running:
5     // handle running state
6   Error:
7     // handle error state
8 ELSE
9   // it is beneficial to always have a default
10  statement
11 END_CASE;
```

6.4 Schleifen

Wie in C gibt es in TwinCat Schleifen. Dabei ist es von zentraler Bedeutung, dass diese Schleife nicht Endlos ist und eine Abbruchbedingung enthält. Ansonsten kann es vorkommen, dass auf die SPS nicht mehr zugegriffen werden kann. Die verschiedenen Schleifen werden hier aufgeführt.

6.4.1 WHILE

WHILE-Schleife

Sofern die Bedingung beim ersten Erreichen nicht zutrifft, wird der Inhalt der Schleife nicht ausgeführt.

```
1 WHILE iCounter <> 0 DO
2   iVar1 := iVar2;
3   iCounter := iCounter - 1;
4 END_WHILE;
```

6.4.2 REPEAT

REPEAT-Schleife

Diese Schleife wird mindestens einmal durchgeführt.

```
1 REPEAT
2   iVar1 := iVar2;
3   iCounter := iCounter - 1;
4 UNTIL
5   iCounter = 0;
6 END_REPEAT;
```

6.4.3 FOR

```
1 FOR iCounter = 1 TO 5 BY 1 DO
2   iVar1 := iVar2;
3 END_FOR;
```

6.4.4 Schleifen Modifikatoren

Mit diesen Modifikatoren kann wie in C beeinflusst werden, wie das Verhalten in der Schleife ist.

```
1 EXIT;          // Ausbruch aus aktuellem Loop
2 CONTINUE;      // Direkter Sprung zu Loop Anfang
```

7 TwinCat

TwinCat ist die Entwicklungsumgebung von Beckhoff für deren SPS Systeme. TwinCat liefert diverse Funktionen vorgefertigt für den Benutzer. In diesem Kapitel werden die relevantesten aufgelistet.

7.1 Bitshifting

Mit Bitshifting Operationen können sämtliche Bits innerhalb einer Variable nach links oder rechts geshiftet werden. Die Bits am Rand werden verworfen und neu entstandene Lücken mit 0 aufgefüllt.

```
1 res := SHL(bByte, 2);    // Shift Left
2 res := SHR(bByte, 2);    // Shift Right
```

Mit Bitrotate Befehlen können herausgeschobene Bits rotiert werden (herausgeschobene Bits werden am anderen Ende wieder hineingeschoben).

```
1 res := ROL(bByte,2);    // Circular Shift Left
2 res := ROR(bByte,3);    // Circular Shift Right
```

Die Shift-Operationen können mit allen gängigen Datentypen verwendet werden.

7.2 Limit

Die Limit-Funktion limitiert die Eingangsvariable innerhalb von einer Minimal- und Maximalwert. Die Funktion kann wie folgt verwendet werden:

```
1 rUsat := LIMIT(umin, rU, umax);
2 // rU wird von umin und umax limitiert
```

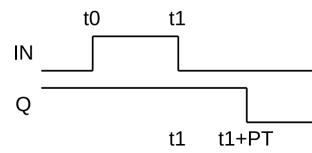
7.3 Timer

Verschiedene Timer Module können direkt aus TwinCat instanziert werden. Diese haben jeweils verschiedene Verhaltensmuster und sind für andere Anwendungen nützlich.

7.3.1 Timer Off-Delay (TOF)



In: Eingangssignal (1 zu 0 startet Timer)
 PT: Verzögerungszeit
 Q: Ausgang des Timers (1 wenn PT erreicht)
 ET: Abgelaufene Zeit



```

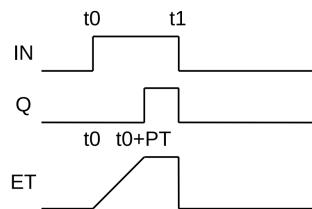
1 VAR
2     fbTOF : TOF;
3 END_VAR
4
5 fbTOF(IN := bTrig, PT := T#500ms);
6 bOut := fbTOF.Q;

```

7.3.2 Timer On-Delay (TON)



In: Eingangssignal (0 zu 1 startet Timer)
 PT: Verzögerungszeit
 Q: Ausgang des Timers (1 wenn PT erreicht)
 ET: Abgelaufene Zeit

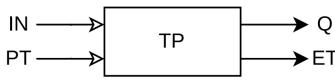


```

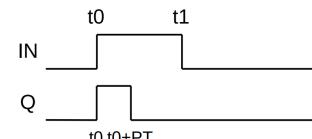
1 VAR
2     fbTON : TON;
3 END_VAR
4
5 fbTON(IN := bTrig, PT := T#500ms);
6 bOut := fbTON.Q;

```

7.3.3 Timer Pulse Generator (TP)



In: Eingangssignal (0 zu 1 startet Timer)
 PT: Pulslänge
 Q: Ausgang des Timers (1 wenn t < PT)
 ET: Abgelaufene Zeit



```

1 VAR
2     fbTP : TP;
3 END_VAR
4
5 fbTP(IN := bTrig, PT := T#500ms);
6 bOut := fbTP.Q;

```

8 Beispiel Code