# Was ist eine SPS

eine speicherprogrammierbare Steuerung (Englisch PLC programmable logical control)

PLC triffts eigentlich sehr gut, weil viel mit logischen Verknüpfungen gearbeitet wird (in der Art „wenn Sensor x true ist und Sensor y false ist dann setze Motor z auf true“)

Prozessor, RAM, Taktgeber, digitale Ein und Ausgänge (neuere haben mehr Funktionen zB analoge Eingänge)

+ sehr **robust** (IP Klassen Gehäuse), zuverlässig, sehr flexibel auch in größeres System mit Feldbussen einbindbar, einfache Aktoren wie zB Lampen direkt ansteuerbar über 24V Ausgänge, **echtzeitfähig**.  
- relativ teurer (einige 100 Euro),

Alternativen zur SPS:

1. Generische Elektronikplatine:   
   + billig, wenn man sehr viele gleiche Schaltungen bauen würde  
   - Erweiterungen aufwändig, zB um es in eine übergeordnete Ampelsteuerung einzubinden,
2. Mikrokontrollerboard zB Arduino:  
   + billig, einfach zu programmieren, viele Schnittstellen erhältlich zB Funk GSM  
   - Stromversorgung der Lampen müsste extra erfolgen zB mittels Relais, robustes Gehäuse braucht man extra, Anbindung an Feldbusse oder ans Netz mit shields
3. IndustriePC (gibt es auch in kleinen Gehäuse, ist aber übertrieben für diese Aufgabe, Vorteil Netzwerk wäre schon dabei, Nachteil – digital out, Echtzeitfähigkeit?)

# SPS Programmierung

Ein- und Ausgänge sind einfache Boolesche Werte, also ein oder aus (5V bzw 24V, true oder false, wie man es halt betrachten will)

Wird in Umgekehrt Polnischer Notation (UPN) programmiert dh man legt boolesche Werte auf den Stack und verknüpft diese miteinander. Logik reagiert immer auf den obersten Stack-Wert dh ein Ausgang wird geschalten wenn dort wo der Schaltbefehl steht, der oberste Stackwert true ist  
zB

LD E0.0  
UN E0.1  
=A0.0

In C würde man sagen   
if(E0.0 && !E0.1)

{  
 A0.0=true;

}

Programmieren der SPS (zB Siemens Simatic):

Programmierung erfolgt mittels Anweisungsliste (AWL) oder Kontaktplan (sieht aus wie LabView, haben wir aber nicht gemacht, für Programmeirer ist die AWL intuitiver). Programmierung erfolgt stackbasierend in umgekehrter polnischer Notation: Werte auf Stack legen und am Stack mit logischen Operatoren verknüpfen. Werte sind als einzelne Bits zu sehen, die true oder false sein können. Es gibt Eingänge zb E0.0 bis E0.7 und Ausgänge A0.0 bis A0.5 Dabei bedeutet E0.1 das Bit 1 des Bytes 0, es geht also von E0.0 bis E0.7, dann ist Byte 0 fertig, dann weiter mit E1.0 bis E1.7 usw, je nach SPS gibt es mehr oder weniger IOs

Die AWL besteht aus „Netzwerken“ (NWs), das sind Programmabschnitte die jeweils einen eigenen Stack bekommen (wie Funktionen) und läuft in einer Endlosschleife, dh nach dem letzten Netzwerk geht’s von vorne los (wie die loop beim Arduino)  
Vor dem ersten NW werden alle logischen Zustände gebuffert, so dass jedes Netzwerk mit denselben Zuständen arbeitet – erst am Ende werden sie wieder aktualisiert und neu geschrieben, je nachdem was während der Schleife passiert ist: Man darf in einem Schleifendurchgang nicht sequentiell denken, ist etwas kniffelig und gewöhnungsbedürftig, weil Zustände erst beim nächsten Schleifendurchgang richtig aktiviert sind.

AWL zB   
LD E0.0 //Zustand des ersten Einganges auf den Stack laden, Stack hat danach Höhe 1  
LD E0.1 //Zustand des zweiten Einganges auf den Stack laden, Stack hat danach Höhe 2  
ULD //oberste Stackwerte mit logischem „Und“ zu einem verschmelzen, Stack hat wieder Höhe 1  
= A0.0 //Ausgang erhält den Wert des obersten Stacks

Was haben SPSen noch:  
-Timer (wie Eieruhren, man stellt eine Laufzeit ein, nach der wird der Timerausgang true. Ist wie eine Eieruhr, die nach der eingestellten Zeit klingelt)  
gibt es in unterschiedlicher Genauigkeit von 1ms, 10ms und 100ms  
  
-Variablen:  
V0.0 … Bit 0 der Variablen V0  
VB0 …VariablenByte 0 also V0.0 bis V0.7, damit kann man also auch Zahlen machen  
VW0 … VariablenWord 16 Bit also die Varaiablen VB0 und VB1 zusammen  
VD0 … VariablenDouble 32 Bit VB0 bis VB3 oder VW0+VW1 (für sehr große Zahlen)  
Setzen von Variablen mit MOVB MOVD usw, vergleichen mit LDB=  
  
-Merker  
M0.0 behalten Infos auch nach Stromausfall

FlipFlops:

S A0.0,1 //Schaltet Ausgang A0.0 auf Dauer true, bis er mit R ausgeschaltet wird  
R A0.0,5 //Schaltet 5 Bits ab dem Ausgang A0.0 auf false, also A0.0, A0.1, …, A0.4 sind dann false

AWL läuft in einer Endlosschleife und in Echtzeit

Inkrement, Dekrement, Zuweisungs- und Vergleichsoperatoren:

NW0:  
INCB VB0 //wie VB0++

NW2:  
 LDB>= 1, VB0 //lädt true auf den Stack wenn der Wert im Byte 0 >=1 ist  
 LDB<= 10, VB0 VB0 //lädt true auf den Stack wenn der Wert im Byte <=10 ist  
 ULD // verknüpft die zwei obersten Stackwerte wirkt also wie: if(VB0>=1 && VB0<=10)  
 =A0.0 //Setzt den Ausgang = dem obersten Stack also gemäß dem if aus der vorigen Zeile

# Ampel Beispiel:

Ampelsteuerung Ablauf (siehe Impulsdiagramm, das sollte man zeichnen können):  
4 Sekunden Rot  
1 Sekunde Rot und Gelb  
5 Sekunden Grün danach 4 mal 0,5 Sekunden grün blinkend  
3 Sekunden Gelb

Könnte man auch mit S und R und Timern programmieren in dem man auf die vorhergehenden Zustand reagiert: zB wenn rot gesetzt ist und genug Zeit vergangen ist kommt orange dazu. Das ist aber extrem kompliziert zB beim Grün Blinken, weil man zusätzliche Merker braucht um zu wissen was vorher war. Einfacher ist es das Ganze in Zeitabschnitte einzuteilen, in dem man regelmäßige Takte erzeugt und bei jedem Takt einen Zähler erhöht. Der Zählerwert entspricht dann also der vergangenen Zeit.

Intervall: 0,5 Sekunden (kleinstes Intervall)

Rot: 1-10  
Gelb: 9-10 und 29-35  
Grün: 11-20 und bei 22, 24, 26 und 28

### Ausgänge

A0.0…Rot  
A0.1…Gelb  
A0.2…Grün

## Impulsdiagramm (Zeit-Schalt-Diagramm)

In einem Impulsdiagrmm sind in x-Richtung die Zeit und in Y-Richtung die Schaltzustände eingezeichnet

## AWL (die sollte man lesen können wenn die Kommentare entfernt sind)

NW0 //Takt erzeugen  
 LDN T40 //der oberste Stack ist true, wenn der timer noch nicht abgelaufen ist  
 TON T40, 5 //T40 ist ein 100ms Timer -> 5 bedeutet 500ms = 0.5s

NW1 //Takt zählen  
 LD T40 //T40 wird alle 500 ms ganz kurz true  
 INCB VB0 //wie VB0++

NW2 //Rot  
 LDB>= 1, VB0   
 LDB<= 10, VB0   
 ULD // if(VB0>=1 && VB0<=10)  
 =A0.0

NW3 //gelb  
 LDB>= 9, VB0  
 LDB<= 10, VB0  
 ULD  
 LDB>= 29, VB0  
 LDB<= 30, VB0  
 ULD  
 OLD  
 =A0.1

NW4 //grünes Licht (A0.2) ein vom 11 bis zum 20.ten Takt bei den Takten 22, 24, 26 und 28  
 LDB>= 11, VB0  
 LDB<= 20, VB0  
 ULD  
 LDB= 22, VB0  
 LDB= 24, VB0  
 LDB= 26, VB0  
 LDB= 28, VB0  
 OLD  
 OLD  
 OLD  
 OLD  
 =A0.2

NW5 //VB0 wieder auf 0 setzen , falls VB0 36 erreicht hat  
 LDB= 36, VB0 //ist wie if(VB===36)  
 MOVB 0, VBO //ist wie VB0=0