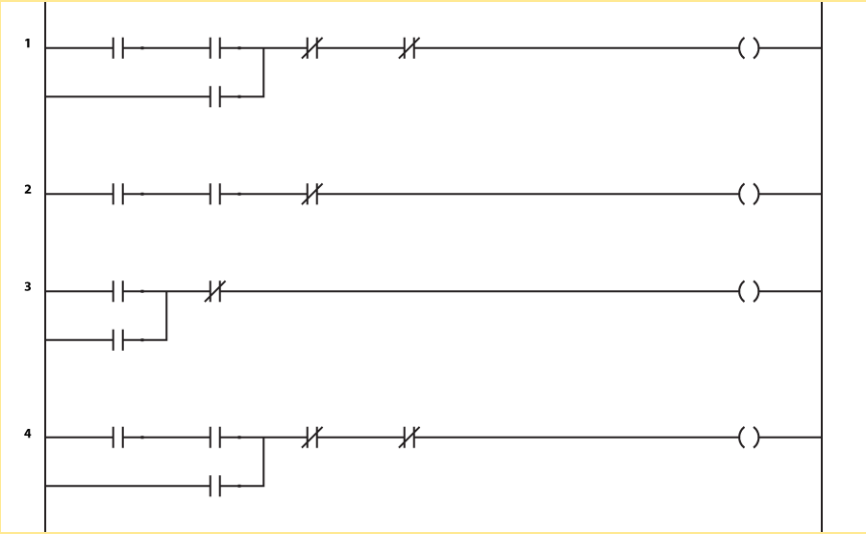
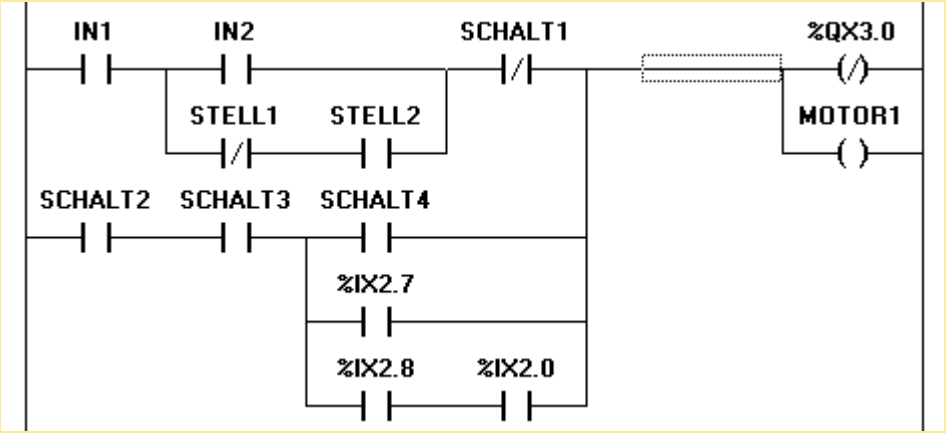
PLC, verschijnings-vormen

PLC, programmeer-talen

KOP (kontakt-plan, ladder logic), lijkt op relais-logica

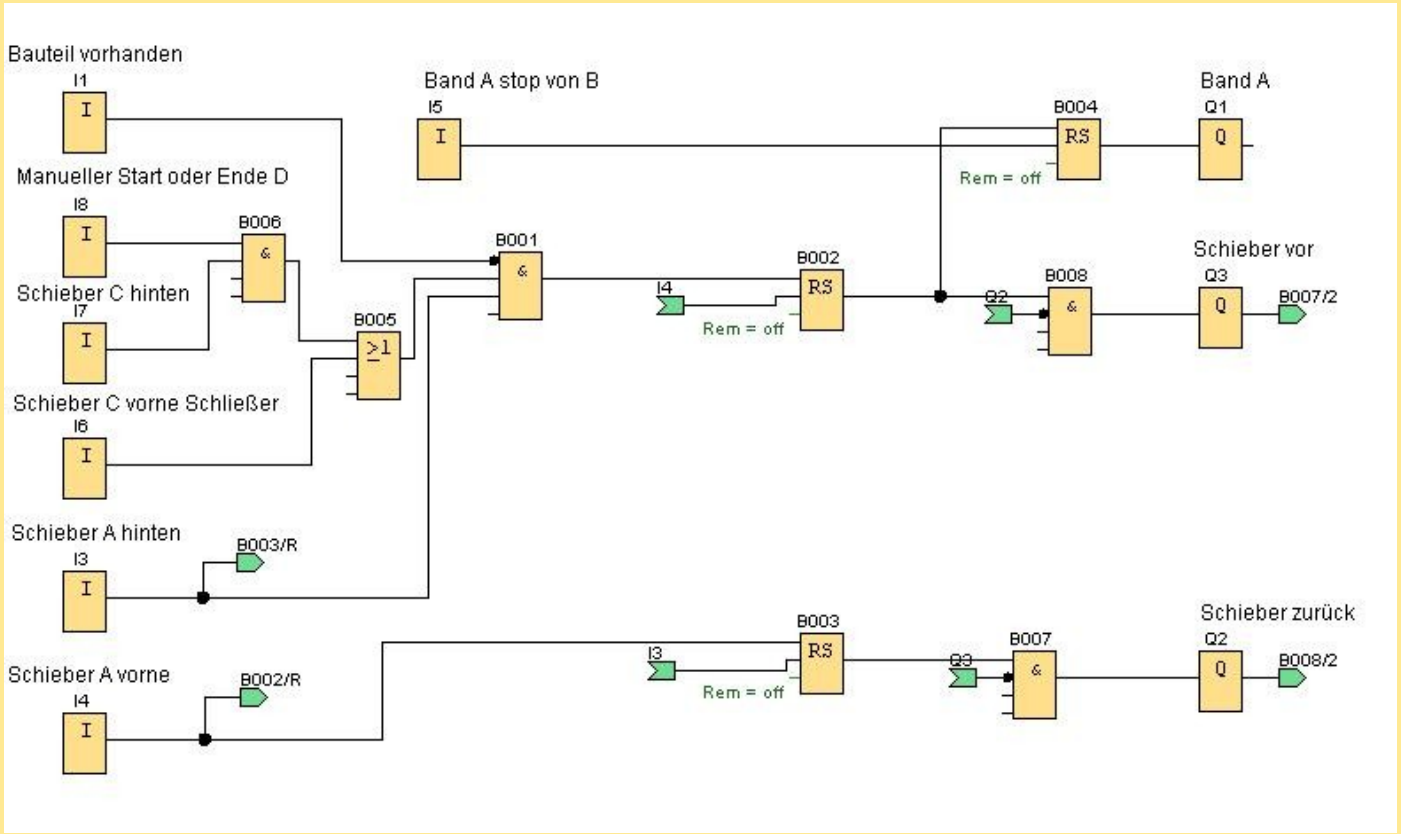


Toepassing: Kleine programma’s, door electrotechnici.



PLC, programmeer-talen

FUP (funktions plan), lijkt op halfgeleider-logica



Toepassing: Middelgrote programma’s, door electrotechnici.

PLC, programmeer-talen

AWL (anweisungs-liste), lijkt op assembly-language

```
      U(
      U(
      UN    "First_Scan"                M50.7
      SPBNB _00c
      L     100
      T     "MaxEngINT".Umid            DB5.DBW4
      SET
      SAVE
      CLR
_00c: U     BIE
      )
      SPBNB _00d
      L     "MaxEngINT".Umid            DB5.DBW4
      ITD
      T     "MaxEngDINT".Umid           DB6.DBD8
      SET
      SAVE
      CLR
_00d: U     BIE
      )
      SPBNB _00e
      L     "MaxEngDINT".Umid           DB6.DBD8
      DTR
      T     "MaxEngREAL".Umid           DB7.DBD8
_00e: NOP   0
```

```
1.  LD    "Start"     ;bit processor
2.  O     "Flag"
3.  A     "Stop"
4.  A     "BLK"
5.  AN    "Error"
5.  =     "Flag"
6.  =     FbB1        ;Fb to FbB1
7.  Read  FbB1        ;FbB1 to FB
8.  L     "Time1"     ;byte processor
9.  SD    "Timer1"
10. LD    "Timer1"    ;bit processor
11. =     "Start_Up"
12. L     "SP_cur"    ;byte processor
13. L     "SP_nom"
14. >=I
15. WriteFBb          ;FB to FBb
16. LD    "Start_Up"  ;bit processor
17. A     FBb         ;FBb to Fb
18. =     "Out"
19. LD    "Start_Up"
20. =     FbB2        ;Fb to FbB2
21. Read  FbB2        ;FbB2 to FB
22. L     "Time2"     ;byte processor
23. SD    "Timer2"
24. TH    "Timer2"    ;bit processor
25. R     "Flag"
26. S     "Error"
27. A     "Stop"
28. R     "Error"
```

Toepassing: Historisch.

PLC, programmeer-talen

Structured Text, lijkt een beetje op de “klassieke” taal BASIC

```
IF    holdReq and
      (autoBed or holdRoute) THEN
   IF not routeWachtStap THEN
      holdPoging.one;
      holdRoute.lat;
      holdGeslaagd.lat;
   ELSE
      holdGeweigerd.lat;
   ENDIF
ENDIF
```

Toepassing: Kleine en grote programma’s, door programmeurs.

(1) Variable *holdPoging* is een zogenaamde “oneshot”, een variabele die precies één sweep lang *true* blijft en daarna op precies het zelfde punt in de sweep wordt gereset naar *false*.

Bovenstaande Structured Text is C++ met zwaar gebruik van de macro preprocessor. Door de keuze voor C++ is zowel simulatie als codegeneratie op elegante wijze mogelijk.

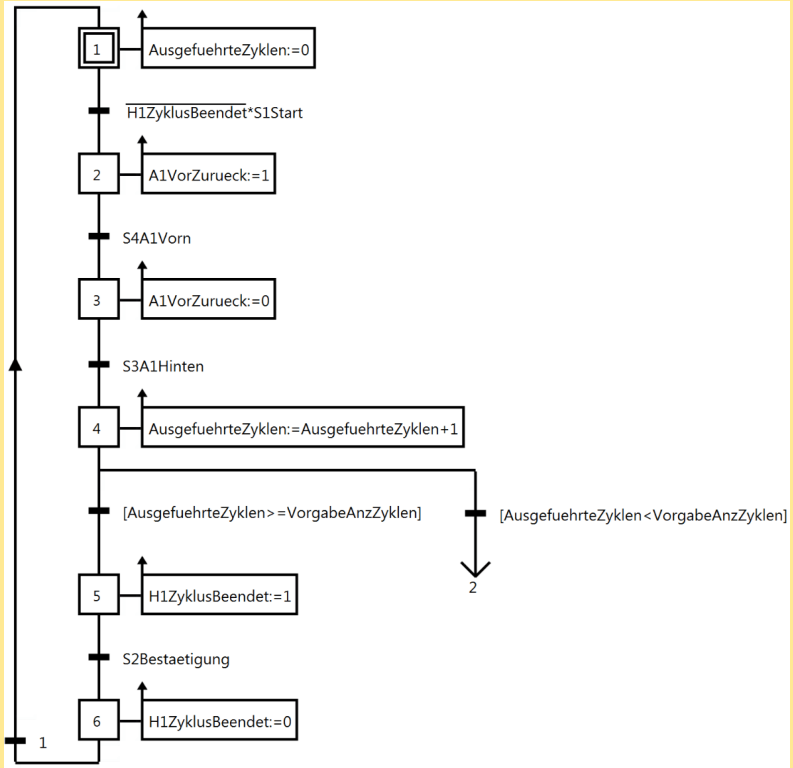
De simpele notatie van Structured Text is enigszins misleidend.
Zo wordt de boolean variabele *holdPoging* in dit code-fragment gereset (1) ongeacht of de expressie *not routeWachtStap* de waarde *true* of *false* heeft.

```
61  #define IF _if (
62  #define THEN );
63  #define ELSIF _elsif (
64  #define ELSE _else ();
65  #define ENDIF _endif ();
```

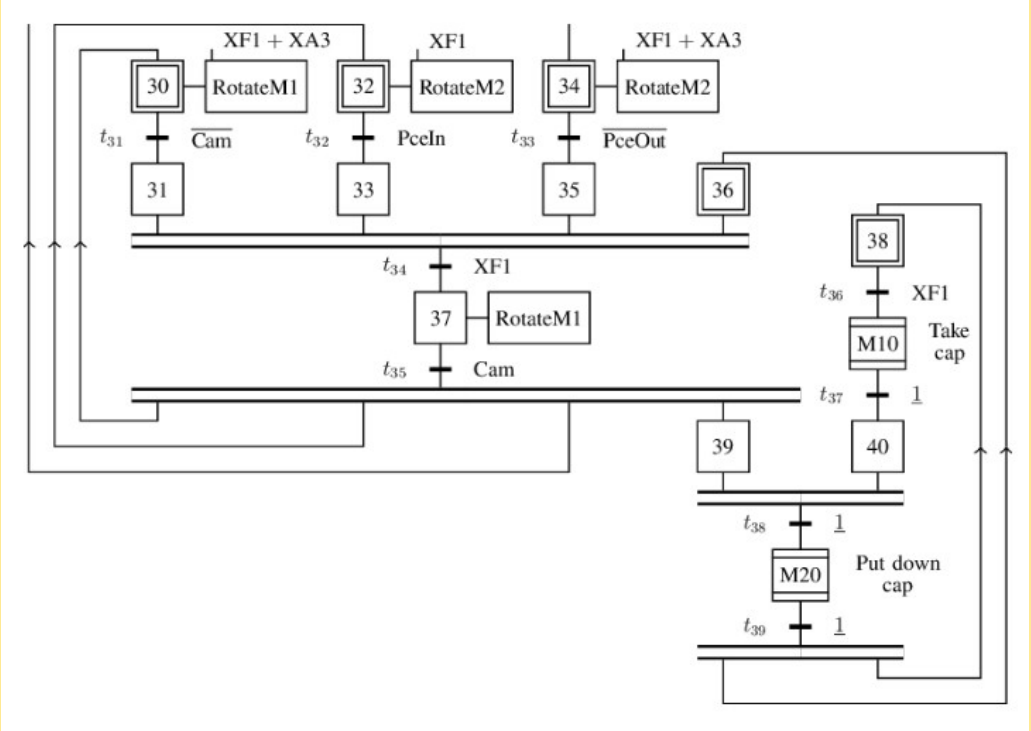
```
41  inline void _if (int boolExpres) {
42    boolLevel++;
43    boolContext [boolLevel] = boolContext [boolLevel - 1] && boolExpres;
44    boolMemo [boolLevel] = boolExpres;
45  }
```

PLC, programmeer-talen

Grafcet (state transition diagram), lijkt op Finite State Machine



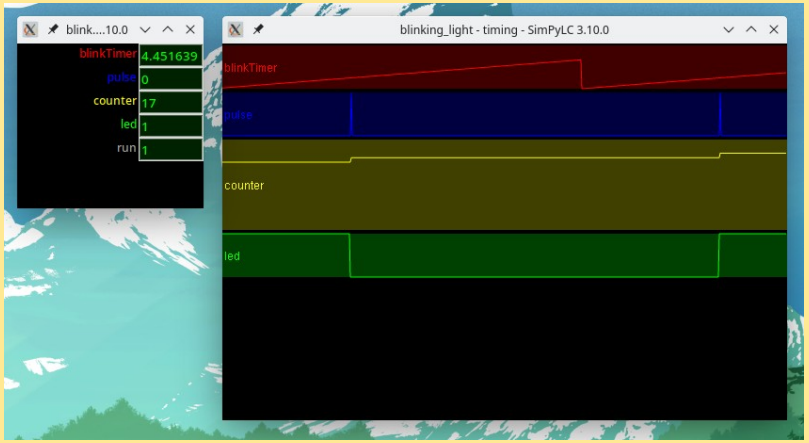
Toepassing: Programma's die zich als Finite State Machine gedragen, door proces-deskundigen.



PLC, programmeer-talen

Object-notatie in bijv. Python of C++, geschikt voor simulatie

Uit Object-notatie kan automatisch code worden gegenereerd voor KOP, FUP, AWL, Structured Text, Grafcet en alle andere talen, zoals bijvoorbeeld C(++).



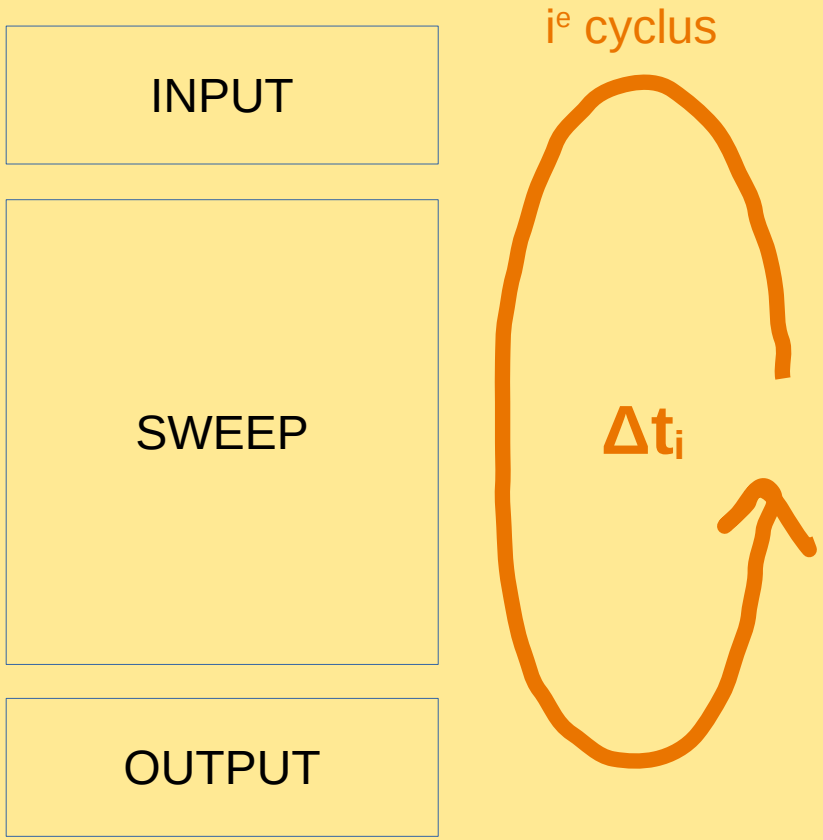
Toepassing: Kleine en grote en/of complexe programma's, door proces-deskundigen en programmeurs.

```
import simpylc as sp

class BlinkingLight (sp.Module):
    def __init__ (self):
        sp.Module.__init__ (self)

        self.blinkTimer = sp.Timer ()
        self.pulse = sp.Oneshot ()
        self.counter = sp.Register ()
        self.led = sp.Marker ()
        self.run = sp.Runner ()

    def sweep (self):
        self.blinkTimer.reset (self.blinkTimer > 8)
        self.pulse.trigger (self.blinkTimer > 3)
        self.counter.set (self.counter + 1, self.pulse)
        self.led.mark (not self.led, self.pulse)
```

PLC, essentie

Een PLC-besturing is in essentie een tijd-discrete besturing (volgende slide) die opeenvolgende cycli i ter lengte Δt_i doorloopt.

- In de INPUT fase worden alle sensors uitgelezen. (bedieningsorganen, camera's, radar, eindschakelaars etc.)
- In de SWEEP fase berekent het besturingsprogramma uit de input de output. Dit programma bevat geen lussen.
- In de OUTPUT fase worden alle actuators aangestuurd. (motoren, zuigers, lampen, geluidssignalen etc.)



Tijd-discrete besturingen

De tijd is verdeeld in tijdstappen Δt_i.

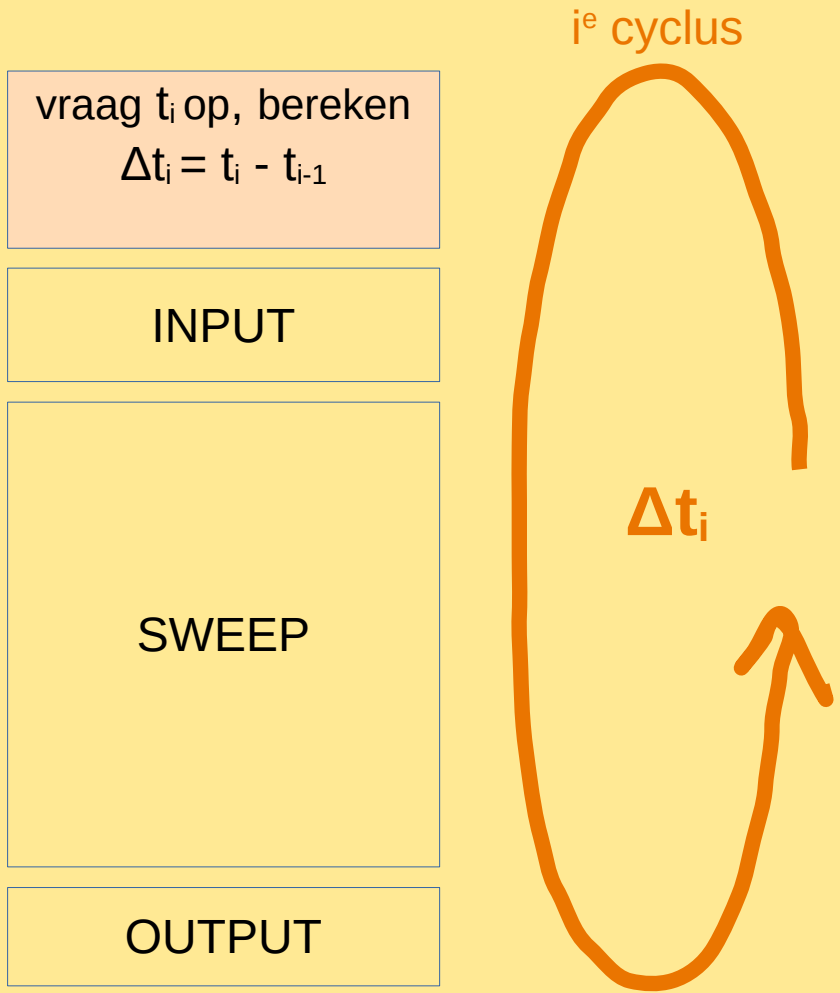
- Deze hoeven niet van gelijke lengte te zijn!
- Ze mogen een bepaalde lengte $\Delta t_{i,\,max}$ niet overschrijden vanwege de reactiesnelheid van de besturing.
- Binnen één tijdstap is het overal even laat, er zijn geen “tussenliggende” tijden.

Voor elke cyclus wordt de klok één 1x opgevraagd

Dit eenmalig opgevraagde tijdstip t_i geldt gedurende de hele i^e cyclus, dus gedurende INPUT, SWEEP en OUTPUT.

Ook geldt:

$$\Delta t_i = t_i - t_{i-1}$$



Een PLC lijkt te multitasken

Alle outputs worden “in 1 klap” berekend uit alle inputs.

Alle outputs reageren binnen $\Delta t_{i,\,max}$ (bijv. 10 ms) op de inputs.

Dit wekt de indruk van multitasking, maar zonder de complexiteit en kwetsbaarheid hiervan.

Timers worden **nooit** gemaakt met delays, maar altijd door de starttijd van de timer t_{i_start} te onthouden en deze in elke cyclus van de huidige tijd t_i af te trekken.

Een timer “loopt af” als in cyclus i geldt: $t_i - t_{i\_start} >= t_{threshold}$
Een timer wordt gestart door eenmalig de starttijd op te bergen.

Omdat de sweep geen lussen of delays bevat blijft de besturing nooit “hangen”.
Dit alles maakt een PLC-besturingen eenvoudig en effectief.