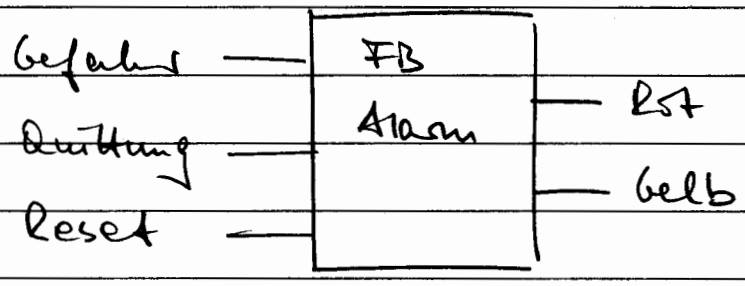
Anwendungsbsp.:

Funktionsbaustein für SPS (Speicherprogrammierbare Steuerung) zur Alarmverarbeitung



- Anforderungsbeschreibung:

1. Bei Eintritt von Gefahr muss Rst angehen.
2. ~~Nach Quittung durch Bediener wird Rst abgeschaltet und Gelb angeschaltet.~~
3. Nach Beseitigung der Gefahr kann Gelb durch Reset gelöscht werden.

- Vorschlag für Lösung (Sprache: AWL = Anweisungsliste)

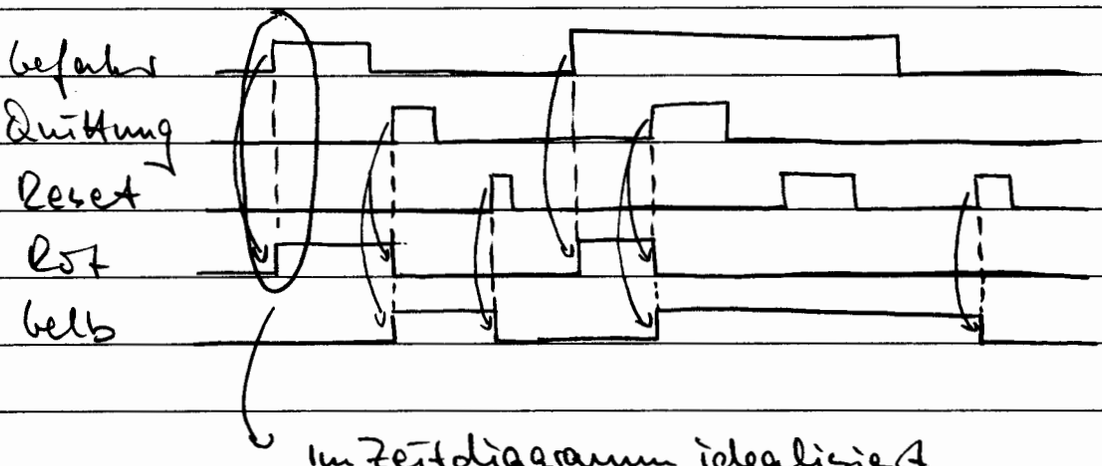
```
VAR_IN
   Gefahr, Quittung, Reset : BOOL;
END_VAR
VAR_OUT
   Rst, Gelb : BOOL := FALSE;
END_VAR

LD Gefahr
S Rst
LD Quittung
S Gelb
R Rst
LD Reset
ANDN Gefahr
R Gelb
```

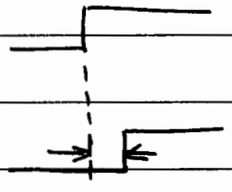
(A) Formalisieren der Anforderungen:

< ~~zuerst~~ probieren lassen, um Schwierigkeiten zu erkennen >

→ hilfreich für besseres Verständnis der Anforderungen: Zeitdiagramm



im Zeitdiagramm idealisiert.
In Wirklichkeit:



Reaktionszeit der Steuerung

Anforderungen:

1. $AG\,((Gefahr \wedge \neg Gelb) \rightarrow AF\,Rot)$ (*)

2. $AG\,((Rot \wedge Quittung) \rightarrow AF\,(\neg Rot \wedge Gelb))$

genauer: $A(Rot\ U\ (\neg Rot \wedge Gelb))$

3. $AG\,((Reset \wedge \neg Gefahr) \rightarrow AF\,(\neg Gelb))$

(*) AF in der Praxis häufig zu schwach.

Bsp.: $AG((befehl \wedge \neg belb) \rightarrow \underline{AF}\ Rst)$

eigentlich

Rst muss innerhalb einer Mindestzeit angehen.

Abhilfen:

(a) $AG((befehl \wedge \neg belb) \rightarrow \underline{AX}\ Rst)$

→ häufig zu stark (s. Modellierung, weiter unten)

(b) Echtzeitlogiken:

$AG((befehl \wedge \neg belb) \rightarrow AF_{<5sek}\ Rst)$

→ andere Modellklasse (hybride Systeme)

(c) Takte/Zyklen „zählen":

$AG((befehl \wedge \neg belb) \rightarrow \underbrace{AX(AX(\ldots AX\ Rst)\ldots)}_{\text{geeignete Zahl}})$

→ problematisch: Was ist ein Zyklus?

(2) Modell des Systems

Zwei Aspekte:

I. Wie wird Steuerungsprogramm abgearbeitet? (Berechnungsmodell)

II. Wie verhält sich die Strecke?

In beiden Fällen Expertenwissen notwendig.

Zu I:

Wiederum zwei Aspekte:

Ia: Wie ist die Semantik von AWL?

Ib: Wie funktioniert eine SPS?

Ia AWL:

- Anweisung: <Label>: <Operator> <Operand>
- ein Akkumulator (hier 1 Bit)
  Bezeichnung u.a.: AE ("aktuelles Ergebnis")
- Operationen:

LD x : AE := x

ANDN x : AE := AE ∧ ¬x

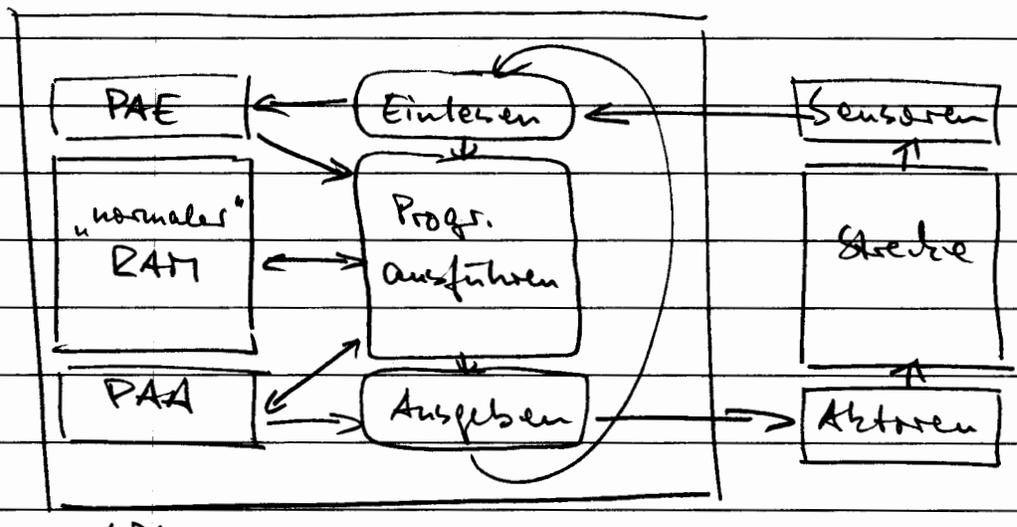
S x : $x := \begin{cases} 1 & \text{if } AE = 1 \\ x & \text{else} \end{cases}$

R x : $x := \begin{cases} 0 & \text{if } AE = 1 \\ x & \text{else} \end{cases}$

=> Zusammenfassung des AWL-Programms zu „Berechnungsschritten".

LD Gefahr S Rot	Schritt 1	if Gefahr = 1 Rot := 1 (else Rot := Rot)
LD Quittung S Gelb R Rot	Schritt 2	if Quittung = 1 Gelb := 1 Rot := 0
LD Reset ANDN Gefahr R Gelb	Schritt 3	if Reset = 1 AND Gefahr = 0 Gelb := 0

Ib: SPS: „Permanent zyklischer Betrieb"



PAE = Prozessabbild der Eingänge
PAA = — " — der Ausgänge

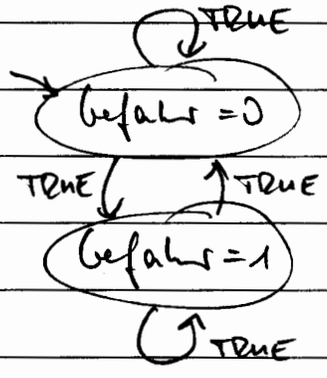
II. Wie verhält sich die Strecke?

Schlechtester Fall: Man weiß nichts.

Hier: Befehl, Quittung, Reset sind
freie Eingänge

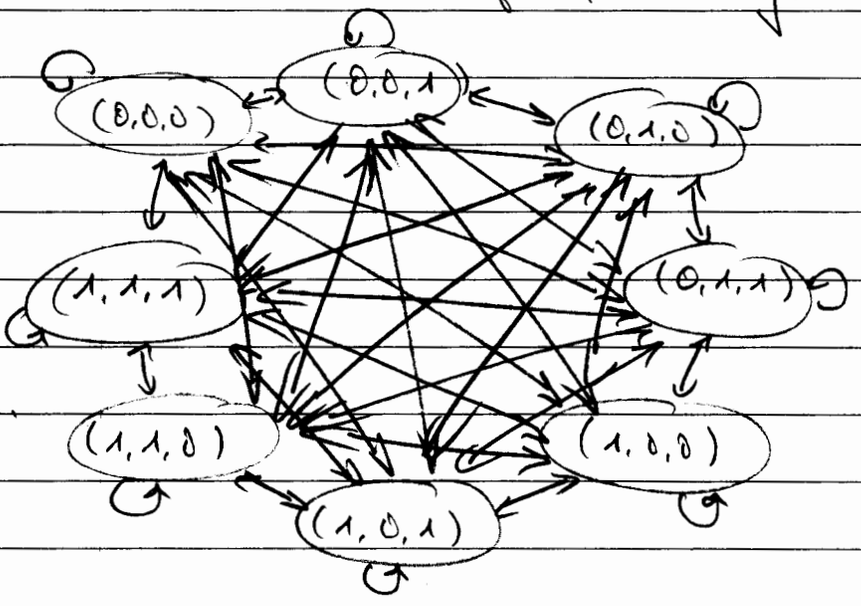
⇒ keine Einschränkung sinnvoll

Modell:



Dito für Befehl / Quittung: Bediener kann allen möglichen Mist machen.

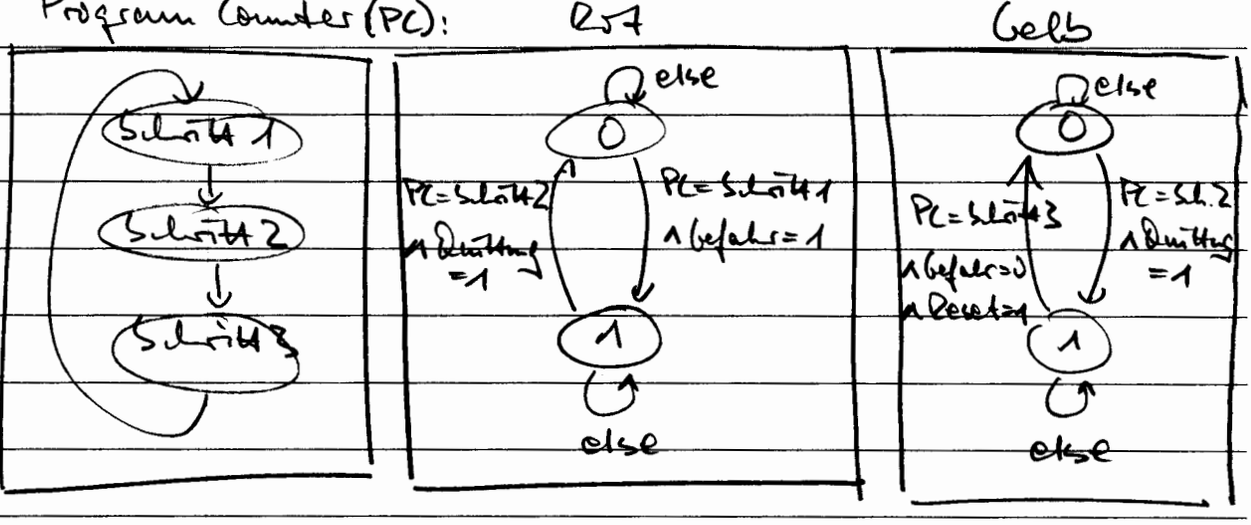
Synchrones
Produkt von Modellen für Befehl, Quittung, Reset:



Synchrone Kopplung:
Alle Teilsysteme machen gleichzeitig ihre Übergänge.

(nur zur Illustration, macht Werkzeug)

1. Versuch der Modellierung:



Warum funktioniert das nicht?

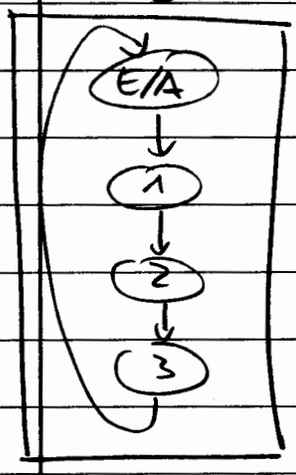
Wirkliches Verhalten wird nicht wiedergegeben.

Eingänge: Können sich hier von Schritt zu Schritt ändern.

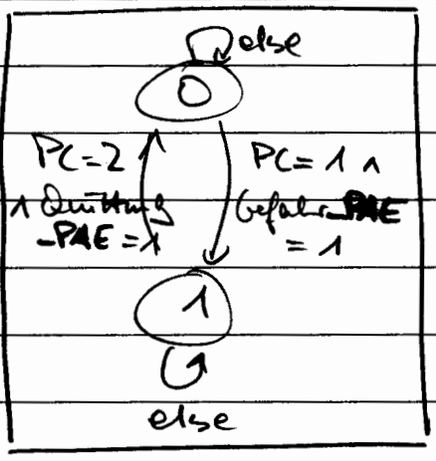
Ausgänge: Weist in Zwischenschritten ohne Relevanz für Ein-/Ausgangsverhalten.

2. Versuch: Modellierung von PAE und PAA

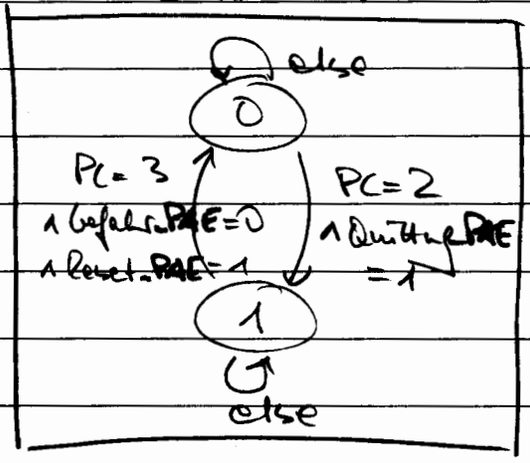
PC:



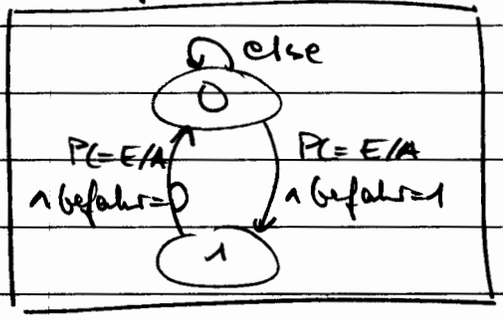
Rot_PAA



Gelb_PAA

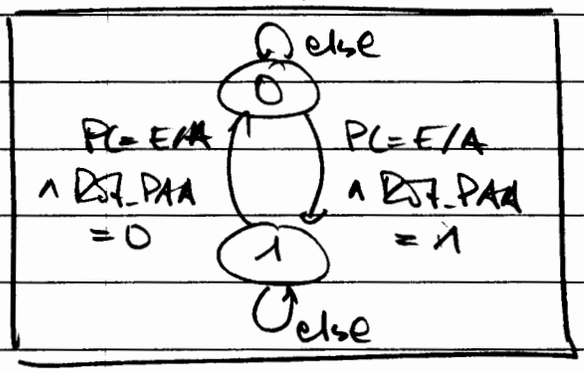


Befehl_PAE



analog für
Quittung_PAE
Reset_PAE

Rot



analog für
Gelb

→ - STV - Code

- Demo mit Rechner

alarm.smv

```
-- Beispiel Alarmverarbeitung
-- Stefan Kowalewski, 28.06.99
-- Überarbeitung 16.06.05

MODULE main

VAR
  -- Eingänge:
  Gefahr: boolean;
  Quittung: boolean;
  Reset: boolean;

  -- Speicherabbild der Eingänge:
  Gefahr_PAE: boolean;
  Quittung_PAE: boolean;
  Reset_PAE: boolean;

  -- Ausgänge:
  Rot: boolean;
  Gelb: boolean;

  -- Speicherabbild der Ausgänge:
  Rot_PAA: boolean;
  Gelb_PAA: boolean;

  -- Status der Programmbearbeitung:
  Status: {EA_Phase, Schritt1, Schritt2, Schritt3};

ASSIGN
  init(Rot) := 0;
  init(Gelb) := 0;
  init(Rot_PAA) := 0;
  init(Gelb_PAA) := 0;
  init(Status) := EA_Phase;

  -- Zyklischer Betrieb:
  next(Status) := case
                    Status=EA_Phase : Schritt1;
                    Status=Schritt1 : Schritt2;
                    Status=Schritt2 : Schritt3;
                    Status=Schritt3 : EA_Phase;
                  esac;

  -- Einlesen des PAE:
  next(Gefahr_PAE) := case
                        Status=EA_Phase : Gefahr;
                        1 : Gefahr_PAE;
                      esac;

  next(Quittung_PAE) := case
                          Status=EA_Phase : Quittung;
                          1 : Quittung_PAE;
                        esac;

  next(Reset_PAE) := case
                       Status=EA_Phase : Reset;
                       1 : Reset_PAE;
                     esac;

  -- Abarbeitung des Programms:
  next(Rot_PAA) := case
                     Status=Schritt1 & Gefahr_PAE : 1;
```

```
                    Status=Schritt2 & Quittung_PAE : 0;
                    1: Rot_PAA;
                  esac;

  next(Gelb_PAA) := case
                    Status=Schritt2 & Quittung_PAE : 1;
                    Status=Schritt3 & Reset_PAE & !Gefahr : 0;
                    1: Gelb_PAA;
                  esac;

  -- Auslesen des PAA:
  next(Rot) := case
                 Status=EA_Phase : Rot_PAA;
                 1 : Rot;
               esac;

  next(Gelb) := case
                 Status=EA_Phase : Gelb_PAA;
                 1 : Gelb;
               esac;

SPEC
AG((Gefahr & !Gelb) -> AF(Rot))
-- AG(!(Rot & Gelb))
```