lose Kopplung, push-Betrieb

Praxis

Zunächst die lose Kopplung.
Die Auftragsvorgabe mittels der Fertigungsmatrix (den push-Betrieb) machen wir in einem zweiten Schritt.

Ich muß die Module in einer schnellen Schleife ständig nacheinander prüfen, bei Betriebsbereitschft starten und den Handshake ausführen. Ich mache das mit einer state-machine, würde wohl auch ein wenig einfacher klappen, aber ist übersichtlich. Das sieht ungefähr so aus :

```
if state1 == 1 :
        busy1 = sps.ReadOPCTag('module1','Busy')
        if busy1 == 1:
            sps.WriteOPCTag('module1','Order',1)
            sps.WriteOPCTag('module1','Start',1)
            state1=2

if state1 == 2 :
        ack1 = sps.ReadOPCTag('module1','Acknowledge')
        if ack1 == 1:
            sps.WriteOPCTag('module1','Start',0)
            state1=3  #(-> nächstes Modul)
```

Nun die Überwachung der Pufferspeicher.

Damit das Fertigstellen einer Bestückung genauer definiert wird, werte ich busy genauer aus :

```
if puffer_1_2 < 5 :          #nur starten, wenn Pufferplatz vorhanden

   if state1 == 1 :
      sps.WriteOPCTag('module1','Order',1)
      sps.WriteOPCTag('module1','Start',1)
      state1=2

   if state1 == 2 :
      ack1 = sps.ReadOPCTag('module1','Acknowledge')
      if ack1 == 1:
         sps.WriteOPCTag('module1','Start',0)
         state1=3

   if state1==3 :
      b1 = sps.ReadOPCTag('module1',"Busy")
      if b1 == 1 :
         state1=4

   if state1==4 :
      b1 = sps.ReadOPCTag('module1',"Busy")
      if b1 == 0 :
         puffer_1_2 = puffer_1_2 + 1
         state1=1
```

Damit die geamte Lösung :

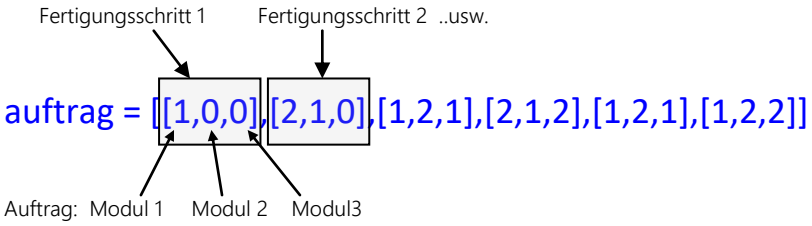
```
from twinLib2021 import *
sps = PlcModule()
nix = input("zum Start ENTER drücken")
sps.StartBand()

puffer_1_2 = 0
puffer_2_3 = 0

state1 = 1
state2 = 1
state3 = 1

while True :

   print("puffer1->2 : ",puffer_1_2," puffer2->3 : ",puffer_2_3)

   if puffer_1_2 < 5 :

      if state1 == 1 :
         sps.WriteOPCTag('module1','Order',1)
         sps.WriteOPCTag('module1','Start',1)
         state1=2

      if state1 == 2 :
         ack1 = sps.ReadOPCTag('module1','Acknowledge')
         if ack1 == 1:
            sps.WriteOPCTag('module1','Start',0)
            state1=3

      if state1==3 :
         b1 = sps.ReadOPCTag('module1',"Busy")
         if b1 == 1 :
            state1=4

      if state1==4 :
         b1 = sps.ReadOPCTag('module1',"Busy")
         if b1 == 0 :
            puffer_1_2 = puffer_1_2 + 1
            state1=1

   #------------

   if puffer_2_3 < 5 :

      if state2 == 1 :
         sps.WriteOPCTag('module2','Order',1)
```

```
sps.WriteOPCTag('module2','Start',1)
        state2=2

    if state2 == 2 :
        ack2 = sps.ReadOPCTag('module2','Acknowledge')
        if ack2 == 1:
            sps.WriteOPCTag('module2','Start',0)
            state2=3

    if state2==3 :
        b2 = sps.ReadOPCTag('module2',"Busy")
        if b2 == 1 :
            state2=4

    if state2==4 :
        b2 = sps.ReadOPCTag('module2',"Busy")
        if b2 == 0 :
            puffer_1_2 = puffer_1_2 - 1
            puffer_2_3 = puffer_2_3 + 1
            state2=1


 #------------


 if state3 == 1 :
    sps.WriteOPCTag('module3','Order',2)
    sps.WriteOPCTag('module3','Start',1)
    state3=2

 if state3 == 2 :
    ack3 = sps.ReadOPCTag('module3','Acknowledge')
    if ack3 == 1:
        sps.WriteOPCTag('module3','Start',0)
        state3=3

 if state3==3 :
    b3 = sps.ReadOPCTag('module3',"Busy")
    if b3 == 1 :
        state3=4

 if state3==4 :
    b3 = sps.ReadOPCTag('module3',"Busy")
    if b3 == 0 :
        puffer_2_3 = puffer_2_3 - 1
        state3=1
```

Wenn das läuft, erweitern wir jetzt auf den push-Betrieb.
Das ist, was die Programmentwicklung angeht, schon deutlich über Prüfungsniveau. Was dahintersteht, also der Unterschied zwischen push und pull, die andere Leitstruktur, ist natürlich für das Verständnis moderner Anlagen enorm wichtig.

Wie schon in Paket 21 (besonders Seite 4) besprochen, setzt die Fertigungsmatrix den MES-Kern, nämlich die Umwandlung von Produktaufttrag in Fertigungsschritte um. (In der Praxis wird das keine wirkliche Matrix sein, dazu sind es viel zu viele Daten, sondern eine geeignete Datenbank).

Nehmen wir dieses Beispiel :



Für die Ermittlung des akteullen Fertigungsschrittes an den einzelnen Modulen müssen wir für jedes Modul einen eigenen Zähler spendieren, weil diese ja verschieden schnell laufen :

```
schritt1=0
schritt2=0
schritt3=0
```

Die Kernoperation im Leitskript ist nun das Ansteuern der Module mit dem richtigen Auftrag :

```
b1 = sps.ReadOPCTag('module1',"Busy")
if b1 == 0 and puffer12 < 5 :
   sps.WriteOPCTag('module1','Order',auftrag[schritt1][0])
   sps.WriteOPCTag('module1','Start',1)
```

Das wär's schon, diese Struktur nun in den Kontext der losen Kopplung gesetzt ergibt die geasmte Lösung :

```
from twinLib2021 import *
sps = PlcModule()
nix = input("zum Start ENTER drücken")
sps.StartBand()

auftrag = [[1,0,0],[2,1,0],[1,2,1],[2,1,2],[1,2,1],[1,2,2]]

schritt1=0
schritt2=0
schritt3=0
state = 1
puffer12=0
puffer23=0

while True :

  if state == 1:
    b1 = sps.ReadOPCTag('module1',"Busy")
    if b1 == 0 and puffer12 < 5 :
      sps.WriteOPCTag('module1','Order',auftrag[schritt1][0])
      sps.WriteOPCTag('module1','Start',1)
      if auftrag[schritt1][0] != 0:
        puffer12=puffer12+1
      state=11
    else :
      state=2
```

```
if state == 11 :
    ack = sps.ReadOPCTag('module1','Acknowledge')
    if ack == 1:
      sps.WriteOPCTag('module1','Start',0)
      schritt1=schritt1+1
    state=2


  if state == 2 :
    b2 = sps.ReadOPCTag('module2',"Busy")
    if b2 == 0  and puffer23 < 5 :
      sps.WriteOPCTag('module2','Order',auftrag[schritt2][1])
      #print("auftrag modul2 : ",auftrag[schritt2][1])
      sps.WriteOPCTag('module2','Start',1)
      if auftrag[schritt2][1] != 0:
        puffer12=puffer12-1
        puffer23=puffer23+1
        #print("Puffer 1-2 fällt :",puffer12)
        #print("Puffer 2-3 steigt :",puffer23)
      state=21
    else :
      state=3
  if state == 21 :
    ack = sps.ReadOPCTag('module2','Acknowledge')
    if ack == 1:
      sps.WriteOPCTag('module2','Start',0)
      schritt2=schritt2+1
    state=3
```

```
if state == 3 :
    b3 = sps.ReadOPCTag('module3',"Busy")
    if b3 == 0 :
      sps.WriteOPCTag('module3','Order',auftrag[schritt3][2])
      #print("auftrag modul3 : ",auftrag[schritt3][2])
      sps.WriteOPCTag('module3','Start',1)
      if auftrag[schritt3][2] != 0:
        puffer23=puffer23-1
        #print("Puffer 2-3 fällt :",puffer23)
      state=31
    else :
      state=1
  if state == 31 :
    ack = sps.ReadOPCTag('module3','Acknowledge')
    if ack == 1:
      sps.WriteOPCTag('module3','Start',0)
      schritt3=schritt3+1
    state=1


  #damit es endlos läuft :----
  if schritt1==5 :
    schritt1=2
  if schritt2==5 :
    schritt2=2
  if schritt3==5 :
    schritt3=2
  #-------------------------------------
```

Die letzten 6 Zeilen sind natürlich an einer Anlage kompletter Unsinn, aber hier am Zwilling eignen sie sich gut, um das Verhalten ausführlicher studieren zu können.

Alternativ müßten wir in der Fertigungsmatrix den Produktionsausflauf anhängen (Nullaufträge Modul1 und Modul2)