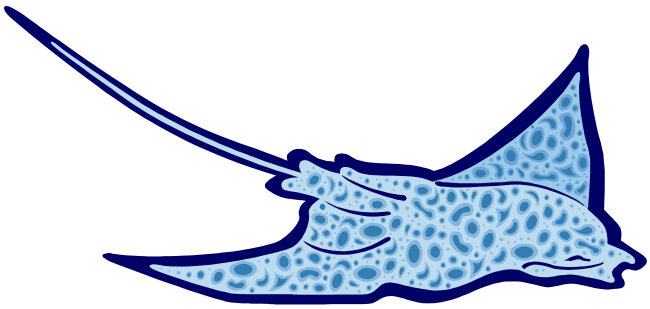
Notizen und Übersichten zur

TaktStrasse

Ulrich Müller

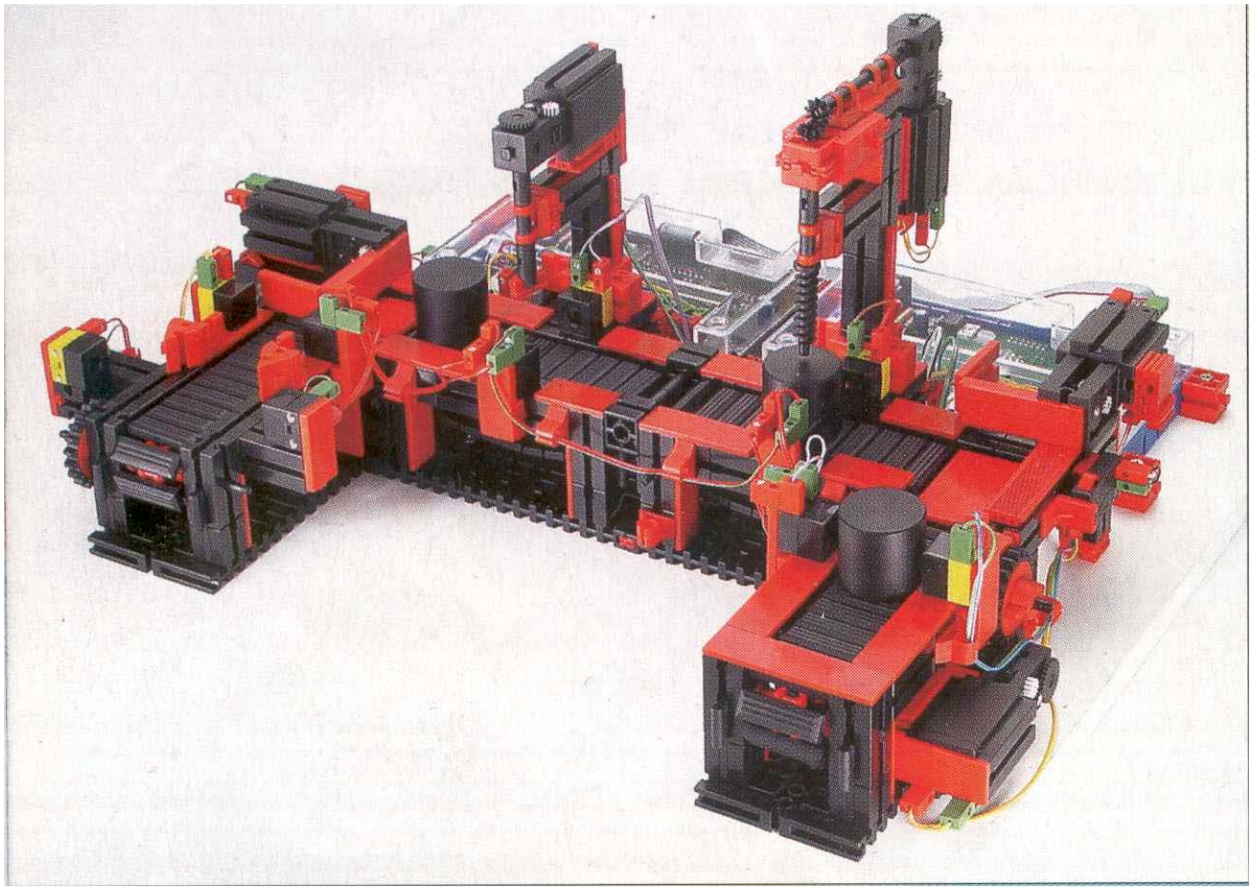


Inhaltsverzeichnis

 Dokumentname : TaktStrasse.doc. Druckdatum : 17.01.2005
Bild : einfügen | Grafik | AusDatei | Office | Rochen.WMF

Übersichten

Übersichtsbild



Modell 51 664A : 9V, Grundplatte 450 x 410 mm. Aus Katalog 2001/2002

Beschreibung aus Online Shop 2005

2 Bearbeitungsstationen
4 Transportbänder, U-förmig angeordnet
8 Motoren
4 Endtaster
5 Lichtschranken (Phototransistor und Linsenlampe)

Komponenten

Zufuhrstation

zfBandMotor	IF1 : M1	linksdrehend
zfPhotoEin	IF1 : I1	
zfPhotoAus	IF1 : I2	
zfLampe	sep.	direkt an Masse/+

Putzstation

ptBandMotor	IF1 : M2	linksdrehend
ptSchieberMotor	IF1 : M3	links : rückwärts
ptRuckTaster	IF1 : I3	Schließer
ptVorTaster	IF1 : I4	Schließer
ptPutzMotor	IF1 : M4	rechtsdrehend
ptPhotoEin	IF1 : I5	
ptLampe	sep.	direkt an Masse/+

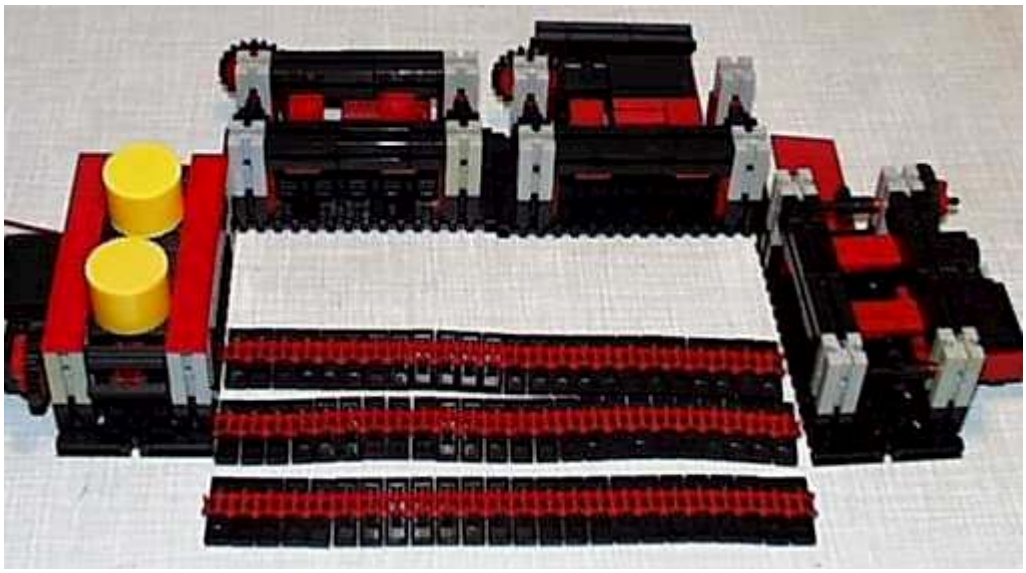
Bohrstation

bhBandMotor	EM1 : M1	linksdrehend
bhBohrMotor	EM1 : M2	rechtsdrehend
bhPhotoEin	EM1 : I1	
bhLampe	sep.	direkt an Masse/+

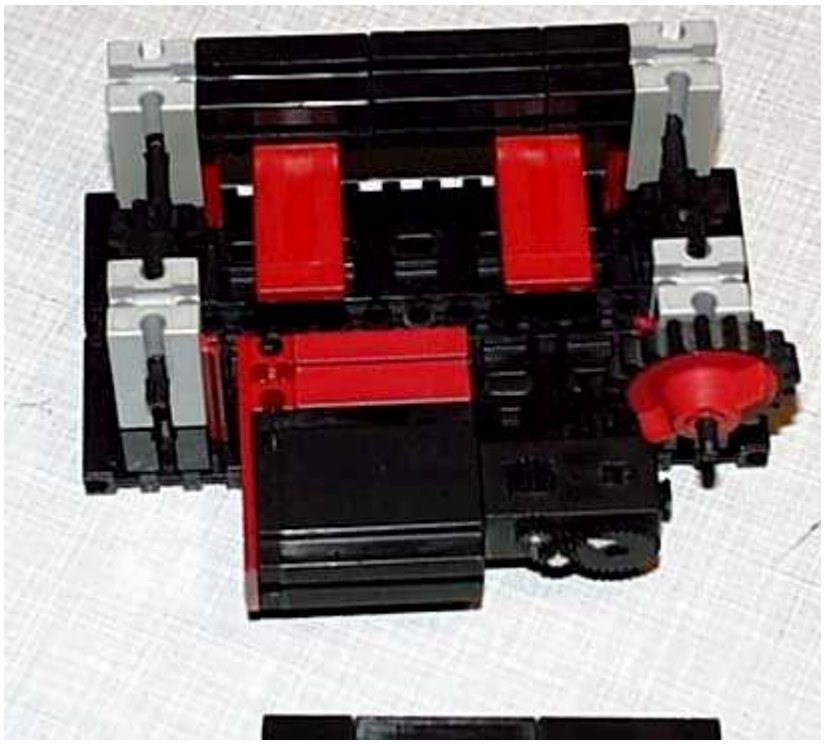
Abgabestation

agBandMotor	EM1 : M3	linksdrehend
agSchiebeMotor	EM1 : M4	links : rückwärts
agRuckTaster	EM1 : I2	Schließer
agVorTaster	EM1 : I3	Schließer
agPhotoEin	EM1 : I4	
agLampe	sep.	direkt an Masse/+

Detailbilder



Die Grundkonstruktion besteht aus vier gleichen Bandstationen, die U-förmig zusammengebaut und mit unterschiedlichen Anbauten versehen werden.



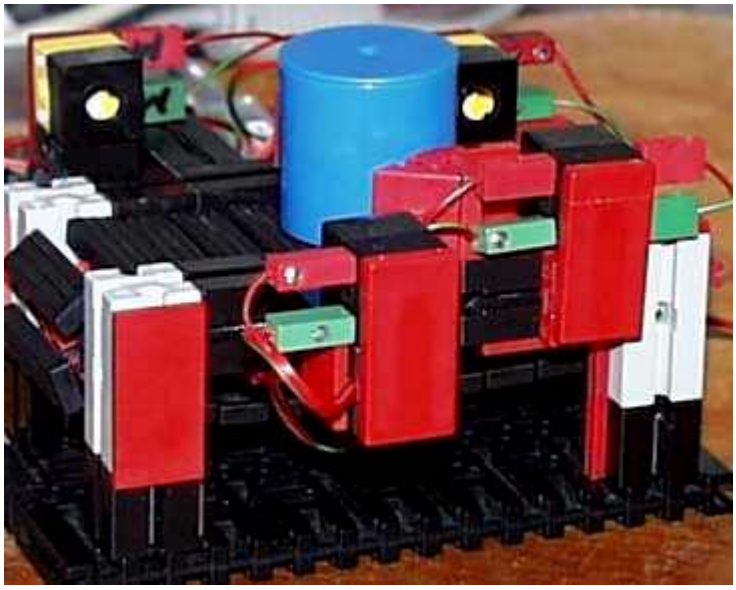
Eine Bandstation ist auf einer kleinen, schwarzen Grundplatte aufgebaut. Die Kette mit 24 Gliedern läuft auf 10er Zahnrädern mit schwarzen Steckachsen, die in 30er Grundbausteinen mit Mittelloch gelagert sind. Die Ketten werden von unten gestützt.

Die Verbindung der Bandstationen erfolgt durch Grundbausteine an denen auch die Ablagen (Bauplatte 45x30, 5mm tiefer) befestigt sind, Auf der einen Seite sind dann noch Zahnstangen für die Schiebemotoren aufgesteckt.

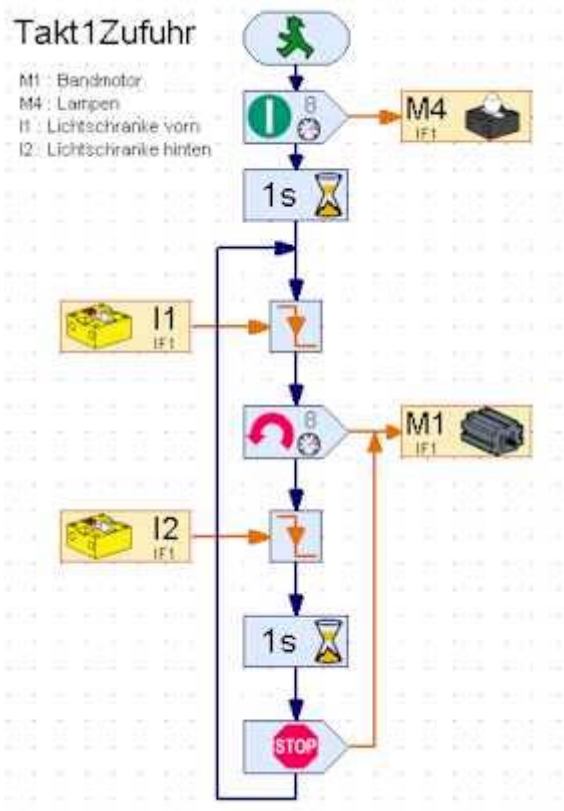
ROBO Pro Lösungen

Dienen gleichzeitig als Ablaufdiagramme der Lösungen in anderen Programmiersprachen und sollten deswegen vor der Lektüre der betreffenden Abschnitte durchgearbeitet werden.

Ablauf 1 : Einzelne Bandstation – Zufuhr



Separater Aufbau der Zufuhr-Bandstation mit Lichtschranken am Ein- und Ausgang.



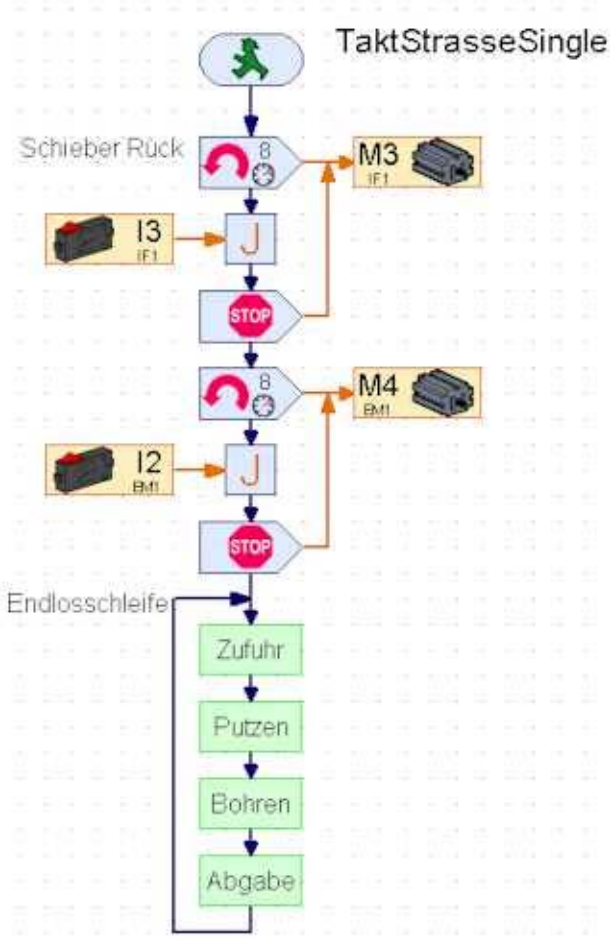
Einschalten der Lampen für die Lichtschranke, 1Sek. anwärmen.

Warten auf das Einlegen eines Teil, ein schon vorhandenes muß kurz entfernt werden.

Band einschalten bis Teil durch die zweite Lichtschranke durchgefahren ist, 1Sek. Nachlauf bis zum Bandend.

Achtung : die Interface-Belegung entspricht nicht der des großen Modells.

Ablauf 2 : Vollständige Bearbeitung jeweils eines Teils



Ausgangsstellung

SchieberMotoren : zurück
Alle Motoren : aus, Standard bei Start.
Alle Lampen : ein. Lampen sind an Masse/+ angeschlossen.

Takt 1 : Zufuhr

Warten auf Werkstück in zfPhotoEin
zfBandMotor : ein
Wenn Teil in zfPhotoAus :
zfBandMotor : aus mit xSek. Nachlauf

Takt 2 : Putzen

ptSchieberMotor :
vor bis ptVorTaster ein
ptBandMotor : ein bis Werkstück in ptPhotoEin
ptPutzMotor : ein für 5 Sek.
ptBandMotor : aus nach x Sek Nachlauf
ptSchieberMotor : zurück bisRuckTaster ein

Takt 3 : Bohren

bhBandMotor : ein bis Werkstück in bhPhotoEin
bhBohrMotor : 3 Sek. ein
bhBandMotor : ein für x Sek.

Takt 4 : Abgabe

agSchieberMotor :
vor bis agVorTaster ein
agBandMotor : ein bis Werkstück in agPhotoEin, xSek Nachlauf
agSchieberMotor : zurück bis agRuckTaster ein

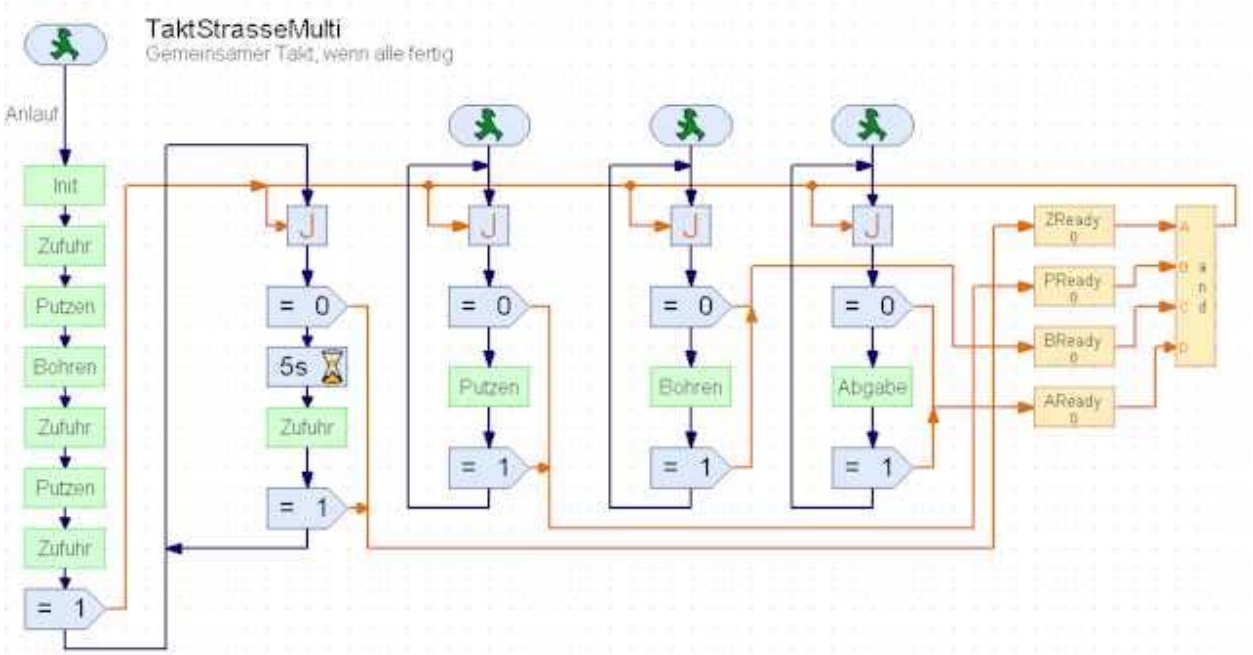
Zurück zu Takt 1

Es befindet sich immer nur ein Werkstück in der Anlage, ein neues Werkstück kann erst nachgelegt werden, wenn das vorherige der Ablagestation entnommen wurde.

Dieser Ablauf ist wenig effizient, aber instruktiv und sollte zum ersten Kennenlernen der Anlage unbedingt genutzt werden, hier läuft nicht soviel durcheinander.

Die Taktnamen sind auch gleichzeitig die Unterprogrammnamen des Programmes TaktStrasseSingle.RPP.

Ablauf 3 : Simultane Bearbeitung mehrerer Teile



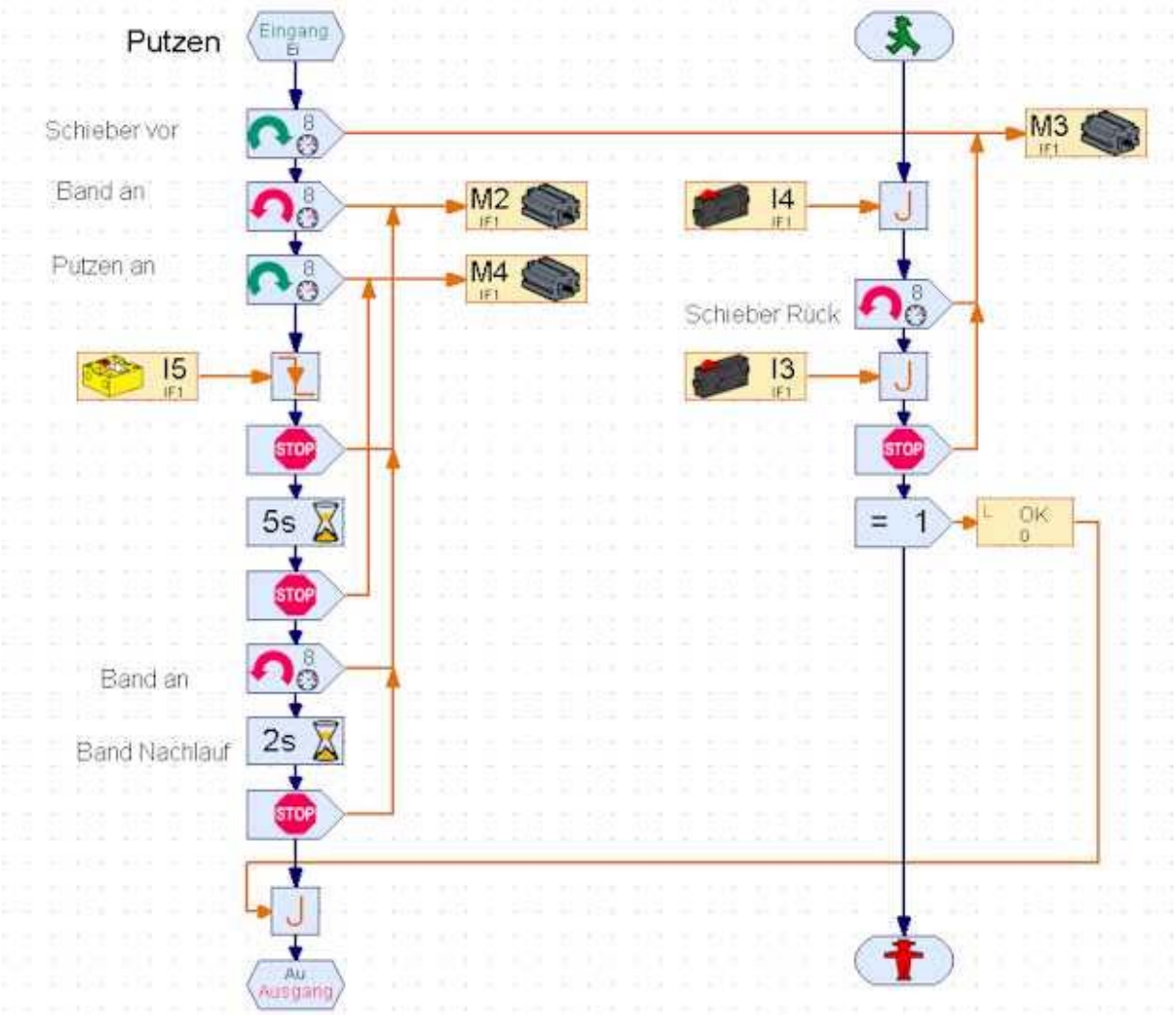
Prinzip des Programms ist der gemeinsame Start aller Takte, die dann simultan arbeiten. Die Takte Zufuhr – Putzen und Bohren – Abgabe haben gemeinsame Resourcen. Bei Zufuhr könnte es passieren, dass das Werkstück in den Schieber von Putzen läuft, deswegen wird der Start von Putzen um 5 Sek. verzögert.

Das Programm besteht aus einer ganzen Reihe von Prozessen (Threads). Eine Änderung der Standard-Einstellung von 5 auf 8 ist erforderlich, bei Download-Betrieb muß auch noch der Mindestspeicher pro Prozess halbiert werden. Einstellung erfolgt über Tab Eigenschaften des Hauptprogrammes

MainThread	**PutzenThread**	**BohrenThread**	**AblageThread**
Init und Anlaufphase	Warten auf OK	Warten auf OK	Warten auf OK
Warten auf OK	PReady = busy	BReady = busy	AReady = busy
ZReady = busy	Putzen	Bohren	Ablage
Zufuhr	PReady = ready	BReady = ready	AReady = ready
ZReady = ready	Zurück zu Warten	Zurück zu Warten	Zurück zu Warten
Zurück zu Warten			

Die einzelnen xReadys sind globale Variable, die durch das **and** auf ein gemeinsames OK gebündelt werden (alle sind fertig).

Details



Putzen ist die am längsten laufende Routine, deswegen wird hier Rückfahren des Schiebers in einen weiteren Thread (Prozess) ausgelagert. Der Prozeß wird mit dem Unterprogramm gestartet und beendet.Die gezeigte Synchronisation über die OK-Variable ist eigentlich überflüssig, da der Hauuptzweig deutlich länger läuft, aber man weiß ja nie.

Der Schieber schafft es nicht alleine ein Werksstück auf das Band zu schieben, deswegen läuft das Band parallell zum Schieber an. Der Putzer läuft ebenfalls gleich an : zum Warmlaufen.

VBA – vbaFish30 : Ablauf 2 – Singlethreading

```
Sub Main

  SetMotor     ptSchieberMotor, ftiLinks
  WaitForInput ptRuckTaster
  SetMotor     ptSchieberMotor, ftiAus
  SetMotor     agSchieberMotor, ftiLinks
  WaitForInput agRuckTaster
  SetMotor     agSchieberMotor, ftiAus

  Do
        Zufuhr
        Putzen
        Bohren
        Abgabe
  Loop Until Finish

End Sub


Sub Zufuhr

  WaitForLow zfPhotoEin
  SetMotor   zfBandMotor, ftiLinks
  WaitForLow zfPhotoAus
  Pause      1500
  SetMotor   zfBandMotor, ftiAus

End Sub
```

Der Ablauf entspricht dem von ROBO Pro, es kann deswegen auf eine weitere Besprechung verzichtet werden. Die verwendeten Befehle entsprechen nahezu 1:1 der bei ROBO Pro eingesetzten graphischen Symbolik. Im Unterschied zu ROBO Pro wird hier aber mit symbolischen Namen für die Ein- und Ausgänge der Inerfaces gearbeitet, da sie aussagekräftiger sind und leicht in der zentralen Definition zu Beginn des Programmes geändert werden können.

Eine simultane Lösung erscheint mit VBA nicht sinnvoll möglich.

C# - FishFa30.DLL : Ablauf 3 – Multithreading

Ablauf und Struktur entsprechen weitgehend der ROBO Pro-Lösung. Auf die zusätzlichen Prozesse (Threads) in Putzen und Ablage wurde verzichtet. Dafür wurde ein neuer "`KontrollThread`" zur Überwachung der "ArbeitsThreads" eingeführt. diese Arbeit wurde bei ROBO Pro durch den Datenfluß in den "gelben Linien" erledigt. Hinzugekommen ist eine eigene Bedien- und Anzeigeoberfläche, die einen Ablauf unabhängig von dem Entwicklungssystem als EXE-Datei erlaubt.

Das Programm besitzt ebenso wie die ROBO Pro-Lösung eine besondere Anlaufphase, auf eine Auslaufphase wurde auch hier verzichtet.

Threads

Genutzt werden die Threading Mechanismen von System.Threading. Zusätzlich zum Thread der Form existieren die fünf Threads : Der Überwachungsthread `KontrollThread` und die Arbeitsthreads `ZufuhrThread`, `PutzenThread`, `BohrenThread` und `AblageThread` mit jeweils einer zugehörenden Execute-Routine.

Die Arbeitsthreads werden über das zentrale Eventobjekt `AllReady` synchronisiert (`AllReady.Set` im KontrollThread und `AllReady.WaitOne` in den Arbeitsthreads) dazu melden sie *arbeit*Ready.Set, wenn sie ein Werkstück bearbeitet haben :

```
private void PutzenExecute() {
      do {
            AllReady.WaitOne();
            Putzen();
            PutzenReady.Set();
      } while(!Abschalten);
}
```

Die eigentliche Arbeit wird in der Putzen-Routine erledigt, hier erfolgt nur die Synchronisation.

KontrollThread sammelt diese Ready-Meldungen :

```
private void KontrollExecute() {
  while(!Abschalten) {
        AllReady.Reset();
        ZufuhrReady.WaitOne();
        PutzenReady.WaitOne();
        BohrenReady.WaitOne();
        AblageReady.WaitOne();
        AllReady.Set();
        Thread.Sleep(200);
  }
}
```

AllReady ist von der Klasse ManualResetEvent und muß deswegen expliziet zurückgesetzt werden, die anderen sind von der Klasse AutoResetEvent und werden bei WaitOne automatisch zurückgesetzt. Das Sleep wurde eingefügt, um einen sicheren Anlauf der Arbeitsthreads (vor dem Reset) zu gewährleisten.

Bedieneroberfläche

Die Bedieneroberfläche enthält Anzeigefelder für die Aktivitäten der einzelnen Arbeitsthreads und ein allgemeines Statusfeld. Die Felder werden farblich codiert.

Das Programm wird über die Buttons START / ENDE bzw. HALT gesteuert. Sie sind während des Programmablaufes zeitweise verriegelt um ein geordnetes Beenden des Programmes zu gewährleisten.

VB.NET – FishFa30.DLL : Ablauf 3 - Multithreading

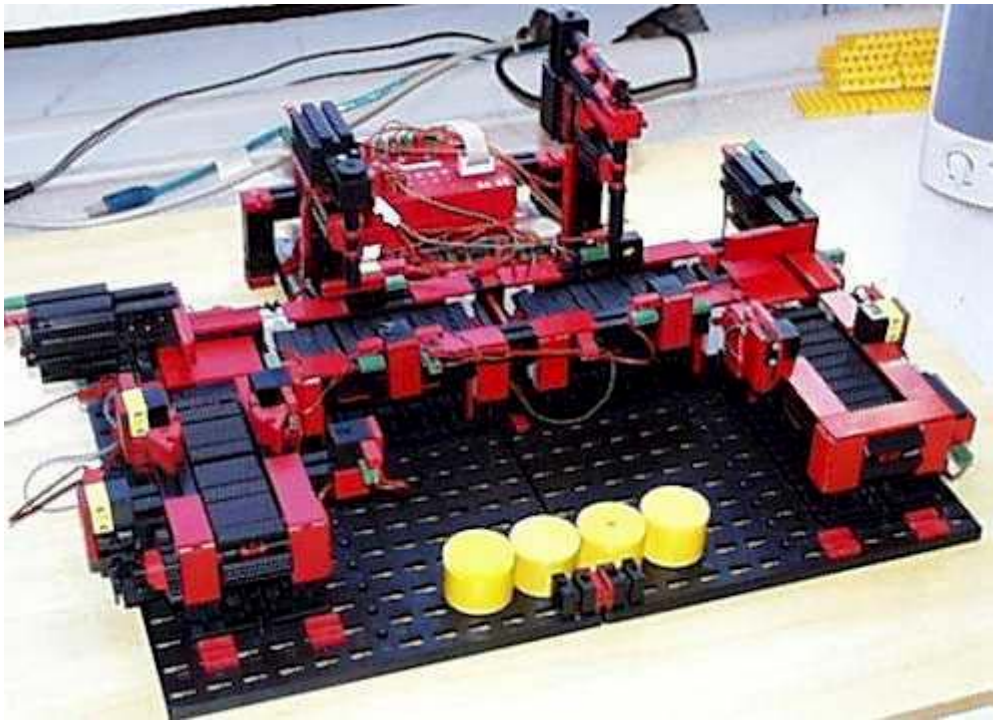


Bild des freien Nachbaus der Taktstraße (Katalogbild siehe C#). Eingesetzt wurde das ROBO Interface mit I/O-Extension im Kompatiblitätmode am COM-Port.

Die VB.NET-Lösung wurde nahezu 1:1 von der C#-Lösung "abgeschrieben", die Struktur des Programmes ist deswegen nahezu gleich. Nur die Notation variiert ein wenig z.B. fehlen die ";", "{}" und die Kommentare beginnen mit "'"

Threads

Wie C#, die Thread-Routine PutzenExecute schreibt sich jetzt so :

```
Private Sub PutzenExecucte()
      Do
            AllReady.WaitOne()
            Putzen()
            PutzenReady.Set()
      Loop Until Abschalten
End Sub
```

und der KontrollThread so :

```
Private Sub KontrollExecute()
      Do Until Abschalten
            AllReady.Reset()
            ZufuhrReady.WaitOne()
            PutzenReady.WaitOne()
            BohrenReady.WaitOne()
            AblageReady.WaitOne()
            AllReady.Set()
            Thread.Sleep(200)
      Loop
End Sub
```