Encoder Auswertung

# Ausgangssituation

An den Motoren sind Lichtschranken und Schlitzscheiben zur Erfassung der Drehung angebracht. Jeder Motor hat dabei zwei Gabellichtschranken (A und B) die ein um 90° versetztes Signal liefern.

A

B

Phase

2

3

4

1

2

3

4

1

ISR

# Schematischer Ablauf der Auswertung

Die Auswertung eines Encoders soll durch einen Interrupt bei jeder steigenden und fallenden Flanke der Signale A und B erfolgen. Dadurch ergibt sich eine vierfache Genauigkeit, aber auch eine höhere Rechenlast. Die dadurch entstehende Anzahl der möglichen Interrupts beträgt 36.000-mal pro Sekunde.

Da der µC die Auswertung für zwei Motoren machen muss, müssen **24.000 ISR pro Sekunde** bearbeitet werden.

Die zwei Eingangs-Signale können vier Zustände (Phasen) annehmen und ergeben mit den Zuständen beim letzten Aufruf, insgesamt 16 Möglichkeiten. Da sich von einem auf den anderen Zustand nur ein Signal ändern darf, sind in den 16 Möglichkeiten einige nicht Gültige. Das KV-Diagramm zeigt diese Zustände, wobei A und B die aktuellen und a und b die Signale der vorhergehenden Messung sind. Die Signale mit Ausrufezeichen sind LOW und ohne HIGH.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | A | A | !A | !A |  |
| B | 0 | 1 | 0 | -1 | b |
| B | -1 | Inv | 1 | Inv | !b |
| !B | 0 | -1 | 0 | 1 | !b |
| !B | 1 | Inv | -1 | Inv | b |
|  | a | !a | !a | a |  |

Im zweiten Feld ist B HIGH und hat sich nicht geändert (kleines b ist auch HIGH), wobei A von LOW auf High gewechselt hat. Dies ergibt dann einen Schritt von 1. Bei negativen Schritten hat sich der Motor in die andere Richtung gedreht und bei 0 gar nicht.

# Programstruktur

|  |  |
| --- | --- |
| aSteps | |
| 0 | 0 |
| 1 | -1 |
| 2 | 1 |
| 3 | Inv |
| 4 | 1 |
| 5 | 0 |
| 6 | Inv |
| 7 | -1 |
| 8 | -1 |
| 9 | Inv |
| 10 | 0 |
| 11 | 1 |
| 12 | Inv |
| 13 | 1 |
| 14 | -1 |
| 15 | 0 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| bPhase | | | | | | | |
| 0 | 0 | 0 | 0 | A | B | a | b |

Die alten und neuen Signale werden in einer Variablen (bPhase) gespeichert und damit aus einer Liste (aSteps) der entsprechende Schritt entnommen.

Nach dieser Auswertung weiß man also, ob und in welche Richtung sich der Motor gedreht hat. Nun lassen sich diese Schritte aufaddieren und in einer globalen Variablen (lEncoder) für die Motordrehung speichern. Tritt bei der Auswertung ein Fehler auf, weil sich zwei Signale gleichzeitig geändert haben, wird dies in einer weiteren Variable (bError) gespeichert.

#DEFINE IS\_SET\_MOT1\_SIG\_A (PINC & (1<<PINC1)

#DEFINE IS\_SET\_MOT1\_SIG\_B (PINC & (1<<PINC2)

#DEFINE IS\_SET\_MOT2\_SIG\_A (PINC & (1<<PINC3)

#DEFINE IS\_SET\_MOT2\_SIG\_B (PINC & (1<<PINC4)  
#DEFINE INV -128

Encoder  
{  
// Member Variablen  
 long lEncoder\_R; // enthält die gezählten Flanken des rechten Motors  
 long lEncoder\_L; // enthält die gezählten Flanken des linken Motors  
 byte bError\_R; // enthält die gezählten Fehler des rechten Motors  
 byte bError\_L; // enthält die gezählten Fehler des linken Motors  
  
 byte bPhase\_R; // enthält das vergangene und neue Signal  
 byte bPhase\_L; // enthält das vergangene und neue Signal  
 byte barrSteps[] = {0,-1,1,INV,1,0,INV,-1,-1,INV,0,1,INV,1,-1,0}; // lookup table  
  
// Funktionen  
 void ISR(); // Interrupt Service Rutine  
 long GetEncoderR(); // gibt die Anzahl der rechten Flanken zurück  
 long GetEncoderL(); // gibt die Anzahl der linken Flanken zurück  
 long GetErrorR(); // gibt die Anzahl der Fehler der rechten Seite zurück   
 long GetErrorL(); // gibt die Anzahl der Fehler der linken Seite zurück  
 void Reset(); // setzt die Flanken und Fehler auf Null zurück  
}

## Wie viel passt in eine long (32bit)?