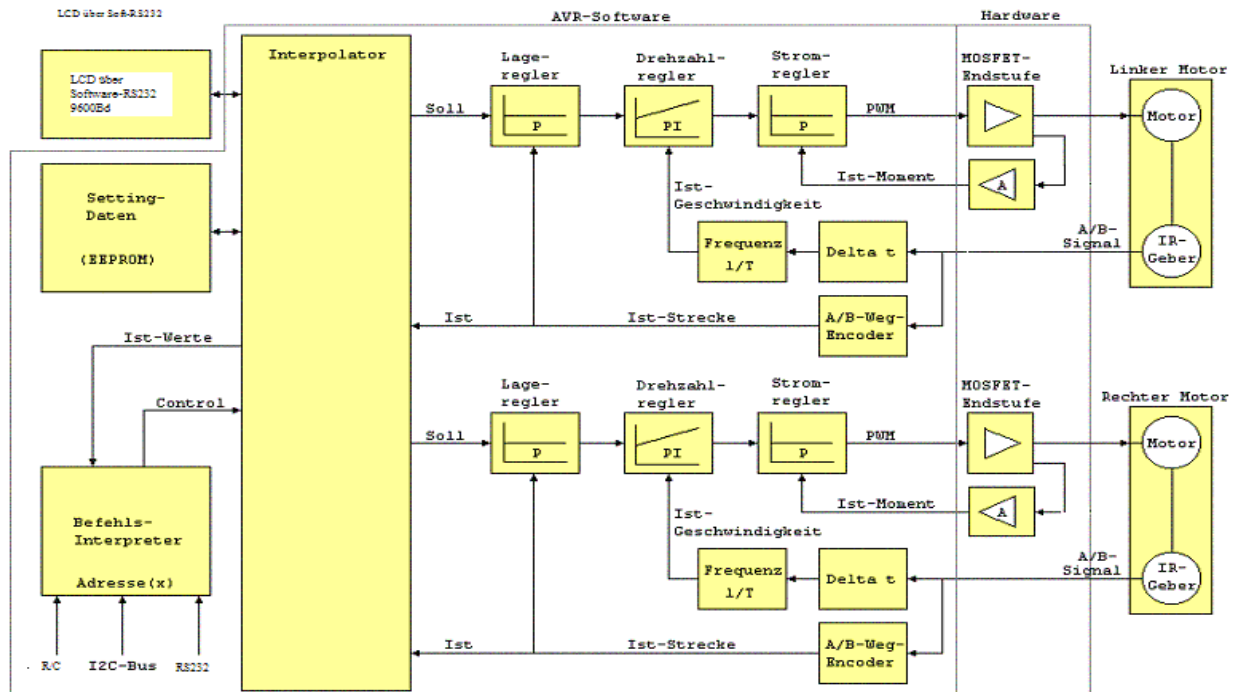


Software-Schema der kompletten Boards



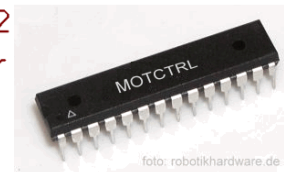
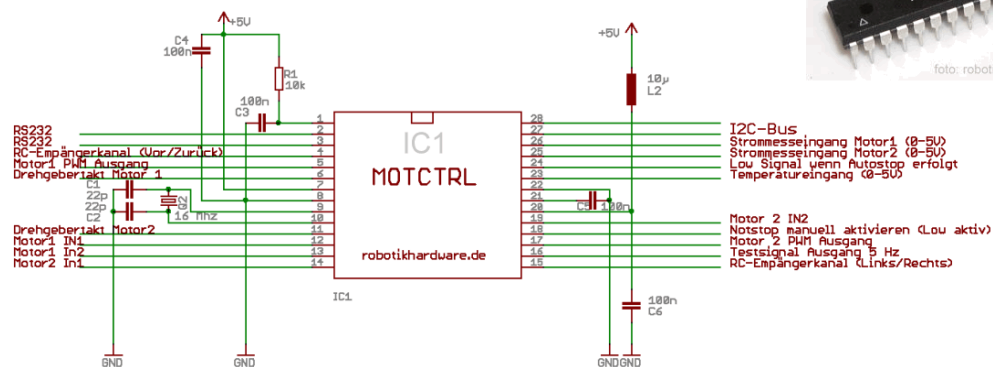
## Voraussetzungen an die intelligente und modulare Antriebseinheit

- Zielcontroller ist ein AVR vom Typ ATMEGA168-20 bei 16MHz
- Regelparameter anpassbar und speicherbar für alle Regler im EEPROM-Bereich des AVR's
- 
- Die Periodendaueremessung des A- und B-Signals soll mit internem Timer gelöst werden und zu einer Frequenz rekonstruiert werden.
- Die Takt-Signaldecodierung soll in einen 8-Bit Wert gewandelt werden der in der richtigen Richtung zählt und auch durch den Befehlsdecoder über den Interpolator gesetzt werden kann.
- Programmiersprache als Erstes in Assembler, vielleicht auch später mit BASCOM oder WINAVR
- Die Regler sollen kaskadiert sein. D.h. sie müssen nur einmal programmiert werden und werden als Funktionen zweimal nacheinander mit ausgetauschten Parameterbänken ParaA und ParaB wiederholt berechnet für beide Ketten im Wechsel. Dadurch wird ein identischer und kürzerer Code erhofft.
  
- Der Regeltakt soll per Interrupt (Timer) konstant ausgelöst werden in höchster Interrupt-Priorität, aber noch unter I2C-Interrupts.
- Das Settingdaten-EEPROM soll primär intern gelagert sein, nur bei nicht möglicher Realisierung soll es extern mit einem 24Cxx-Derivat ergänzt oder ersetzt werden.
- Die möglichen Interpolationen sollen die Grundfunktionen zum fahren in einer Richtung, dem exakten Wenden auf der Stelle in beliebigen Winkeln und dem Fahren auf einer beliebigen Kreisbahn ermöglichen ohne externe Eingriffe und Rechenleistung.
- Istpositionen und aktuelle Zustandsparameter sollen zu jeder Zeit und unmittelbar lesbar sein. Das soll einer übergeordneten CPU ermöglichen den Ablauf zu synchronisieren.
- Ebenfalls sinnvoll wäre eine Stromistwert-Diagnose um die aktuellen Drehmomente im Antrieb zu ermitteln, Verklemmen, Blockierung und andere schwerwiegende Probleme. Wahlweise per Parameter über I2C oder auch als digitalen Ausgang der parametrierbar sein kann mit einer Grenze (sofern noch ein Port frei ist). Dieser Port kann auch im erweiterten Fall eine andere Funktion bekommen welche wiederum durch die Parameter programmierbar sein kann.
- Antriebsfreigaben und Reglerfreigaben sowie eine Not-Aus-Funktion müssen ebenfalls vorhanden sein. Evtl. noch ein parametrierbares Lebenszeichen-Bit auf dem I2C-Bus zur Master-CPU oder per RS232 bis zum Steuerrechner das in einem definierten Zeitraum toggeln muß um die Kommunikationsaktivität zu überwachen und bei Fehler die Antriebe sofort stillsetzt mit einem Not-Aus.
- Evtl. noch Diagnose-LED's zur Anzeige der aktuellen Ausgangszustände (Vor / Zurück und Speed).
- Zum Schutz der Mechanik und zum verhindern von Überlastungen im Antrieb wäre eine Stromgrenze welche einem maximalen Momenten-Istwert entspricht denkbar. Diese sollte dann den Antrieb begrenzen.
- Fehler-Flags für den Fall eines Stromgrenzenfehlers und eines Drehzahlregleranschlags über der zulässigen Grenze. Diese sollen auch als Port zuweisbar sein und jederzeit per I2C abrufbar.
- Einfaches Dumpen der Settingdaten im EEPROM per I2C als Register in beiden Richtungen (lesen und Schreiben) soll möglich sein für die Parametrierung des Antriebsmoduls und der Erstellung einer Datensicherung.
- Einstellbare Koppelübersetzung des Gebers zur Motorwelle in Inkrementen zu Distanz in mm. Dient damit als Basis für Fahrten in der Wegberechnung auf normierte Einheiten. Die Antriebswellenübersetzung zur Kettenübersetzung muss auch definierbar sein.
- Eine aktuelle mitgerechnete X und Y-Koordinate wäre auch sinnvoll die sich aus den aktuellen interpolierten Strecken aufgrund der Odometrie herleitet. Eine Änderung der X- und -Werte soll über den I2C-Bus jederzeit möglich sein.
- Ein Busy-Flag für den Zustand der angeforderten Interpolation. Auch über I2C lesbar und auf einen Pin parametrierbar per Settingdaten. Gibt der Master-CPU Bescheid ob der Fahrauftrag abgeschlossen ist oder noch ausgeführt wird.
- Ein Ready-Flag zur Anzeige das der Antrieb empfangsbereit ist für einen Fahrauftrag und diesen unverzüglich ausführen kann.
- Alle Flags sollen in einem Statusregister als einzelne Bits festgelegt sein. Damit ist mit nur einem Byte der gesamte Zustand des Fahrantriebs lesbar von der Master-CPU.

## Ports am verwendeten AVR

- 1 I2C-Anschluss zu Master-CPU (Hardware I2C)
- 1 RS232-Anschluß zum PC
- 2 Eingänge Takt oder A/B-Signale
- 2 Eingänge analog in ca. 8-Bit Auflösung oder auch 10-Bit für Stromistwerte (ev. 8Bit möglich)
- 8 E/A-s reserviert für :
  - 2 PWM-Ausgänge für die Endstufen
  - 4 digitale Ausgänge für das Richtungssignal der 2 Endstufen incl. Freilauf / gesteuerte Bremse
  - 2 Eingänge Takt eines Drehgebers
  - oder
  - 4 PWM-Ausgänge das Richtungssignal der 2 Endstufen incl. Freilauf / gesteuerte Bremse
  - 4 Eingänge für A/B-Signale eines Quadraturencoders
- 1 analogen Eingang für Temperaturfühler an der Endstufe
- 1 LED- Ausgang
- 1 STOP- Eingang (Freigabe)
- 1 STOP- Ausgang (NOTAUS)
- 1 RS232-Display-Ausgang

Endstufentreiber mit RC, I2C-Bus und RS232 Schnittstellen für Modelfahrzeuge / Roboter



# Motorbefehle

Die Befehls-Syntax ist im Dokument [RN-MotorControl](#) festgelegt; Die Datei ist im Download-Bereich vom Roboternetz erhältlich.

## Funktion des Blocks Stromregler mit Motorschutz

Kennung Byte1	Befehl Byte2	1 Parameter (Steuerwert) Byte3	2. Parameter (Auswirkung) Byte4	Startwert Byte5
60	1	0 – 150 in % des Motorstroms 100% entsprechen 1V am Eingang ADC2 "STROM" (RESET bei 250% Strom)	0 = Freilauf sofort 1 = Strombegrenzung 2 – 50 = Freilauf nach x*100ms	100
<b>Beispiel:</b>				

Am ADC Port "STROM" wird ständig die Spannung überwacht. Ein Wert von 0...1,5V lässt uneingeschränkten Betrieb des Controllers zu.

**Parameter 2 = 0; sofortiger Freilauf (Funktion einer Sicherung)**

Hier wird sofort bei Erreichen von 1,5V IN\_A und IN\_B auf 0 gesetzt. Statusbyte: Bit1 von ST = 1

**Parameter 2 = 1; kurzschlussfeste Strombegrenzung**

Hier wird die PWM soweit reduziert, das Parameter1 (max1,5V) nicht überschritten werden. Das Regelverhalten eines einfachen P-Reglers sollte ausreichend sein.. Bit2 von ST =1

**Parameter 2 = 2-50; toleranter Überstromschutz (Funktion ähnlich Motorschutzschalter)**

Hier wird die PWM soweit reduziert, das 2,2V nicht überschritten werden. Ein Timer wird mit Parameter 2 bei 1,5V gesetzt und bei <1,5V zurückgesetzt . Der Timer setzt nach Ablauf IN\_A und IN\_B auf 0.

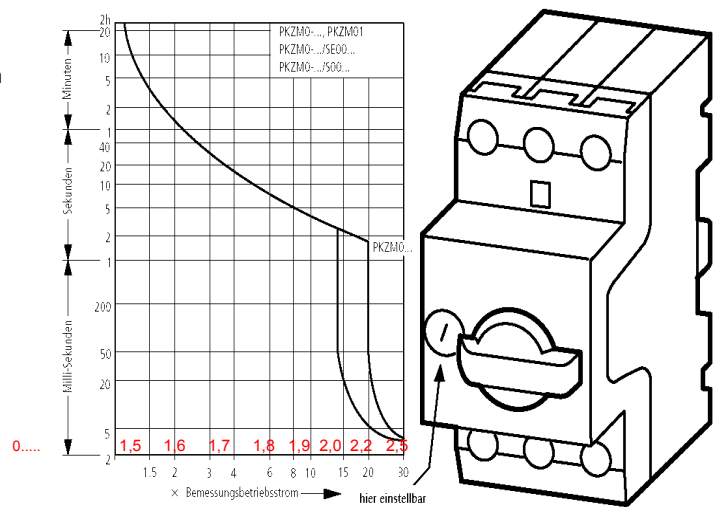
**Das Programm löst immer einen Software- RESET aus, wenn die Spannung an ADC2 2,2V überschreitet.** Es erfolgt ein Neustart des Controllers mit den Startwerten. Das ist die fest eingestellte „Sicherungsfunktion“ des Controllers.

Auslösekennlinie des Systems:

Bild: Motorschutzschalter für Drehstrommotore

Diese Kennlinie dient auch als Vorgabe für den Schutz von DC-Motoren mit neuen Bemessungsstrom (rote x-Achse)

Auslösekennlinien Motorschutzschalter (Hochleistungs-)Kompaktstarter, I



## Funktion des Blocks Drehzahlregler:

<b>Drehzahlregler einstellen</b>				
<b>Es wird ein Drehzahl-Sollwert von 0...255 vorgegeben. Dieser Wert wird mit Der Drehzahlregler ist als PID-Regler ausgelegt.</b>				
<b>Parameter2: Diese Funktion stellt das Beschleunigungs- und Verzögerungsverhalten des Motors ein.</b>				
Wird eine Drehzahl vorgegeben, wird die PWM über eine Zeitschleife mit der Wartezeit erhöht bzw. reduziert, bis der Einstellwert erreicht ist.				
<b>Kennung</b> Byte1	<b>Befehl</b> Byt2	<b>1 Parameter</b> Byte3	<b>2. Parameter</b> Byte4	<b>Startwert</b> Byte5
60	<b>2</b>	0 = Rampe aus 1 = Rampe ein	0 bis 255 0 = schnell 1-255 = x *10ms	10
<b>Beispiel:</b>				

Verwendete Ports für Motor\_1 und Motor\_2

IN\_A1= PD6

IN\_B1= PD7

PWM1= PB3 OC2A

IN\_A2= PB0

IN\_B2= PB5

PWM2= PD3 OC2B

Funktionstabelle der Steuerogik:

IN_A	IN_B	PWM	Ergebnis
1	0	Puls	Vor
0	1	Puls	Rück
0	0	0	Freilauf
0	0	Puls	gesteuerte Bremse
1	1	X	Bremse_H

<b>Motor Drehen</b> Dieses ist die normale Motorsteuerungsfunktion				
<b>Parameter1:</b> Stellt die Betriebsart des Motors nach Funktionstabelle ein. <b>Parameter2:</b> Wird eine Drehzahl vorgegeben, wird die PWM über eine Zeitschleife mit der Wartezeit eingestellt				
Kennung	Befehl	1 Parameter	2. Parameter	Startwert
Byte1	Byte1	Byte1	Byte1	Byte1
60	3	0 – 3 0 = Freilauf 1 = vorwärts 2 = rückwärts 3 = Schnellstop	0 - 10 = Freilauf 10 – 255 Drehzahlstufen 255 = Vollgas	Freilauf mit PWM=0
<b>Beispiel:</b>				

<b>Motor Strom und Statusbyte auslesen</b> Für Kontroll- oder externe Regelzwecke kann der Motorstrom und das Statusbyte ausgelesen werden. So kann z. B. der Anschlag eines Stellmotors anhand der Überhöhung des Motorstroms erkannt werden.				
Kennung	Befehl	1 Parameter	2. Parameter	3. Parameter
Byte1	Byte2	Byte3	Byte4	Byte5
60	4			
<b>Beispiel</b>				
<b>I2C Adresse setzen (ändern)</b> Bei der Verwendung mehrerer Motorsteuerungen ist es notwendig, dass jeder Motor eine eigene I2C-Adresse bekommt.				
Kennung	Befehl	1 Parameter	2. Parameter	3. Parameter
Byte1	Byte2	Byte3	Byte4	Byte5
60	15	99	99	slaveid
<b>Beispiel:</b>				

<b>PWM-Frequenz</b> Die PWM-Frequenz ist einstellbar., z.B. um akustischen Resonanzen auszuweichen.				
Kennung	Befehl	1 Parameter	2. Parameter	3. Parameter
Byte1	Byte2	Byte3	Byte4	Byte5
60	5		2, 4, 8, 16, 32, 64 khz einstellbar	
<b>Beispiel</b>				
frei für weitere Parameter				
Kennung	Befehl	1 Parameter	2. Parameter	3. Parameter
Byte1	Byte2	Byte3	Byte4	Byte5
60	xx			
<b>Beispiel:</b>				

frei für weitere Parameter				
Kennung	Befehl	1 Parameter	2. Parameter	3. Parameter
Byte1	Byte2	Byte3	Byte4	Byte5
10	xx			

**Beispiel:**

Ziel dieser Software ist eine komfortable Bedienung der Motorsteuerung zusammen mit einer intelligenten Motor- und Geräteschutzfunktion. Alle Parameter sollten in weiten Grenzen manipulierbar sein, sofern sie den Geräteschutz nicht beeinflussen.

Diese geplanten Befehle sind sicherlich nicht vollständig. Falls sie sich als praxisfern herausstellen, sind sie praxisnah anzupassen. Weitere Befehle dürfen gern hinzugefügt werden.