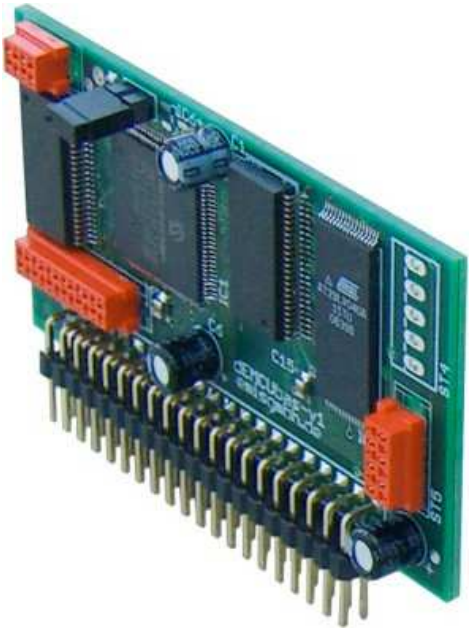
EMIS GmbH

Die Mikrocontrollereinheit EMCU



Version 1.0 vom 16.01.08

Inhaltsverzeichnis

1. Einführung

Das intelligente Schrittmotor-Interface EMCU dient zur Anbindung des modularen Schrittmotorsteuersystems SMCflex über die serielle Verbindung.

Das Interface EMCU wird direkt auf die Basiseinheit SMCflex-Basis aufgesteckt.

Das Interface enthält einen Mikrocontroller, welcher Ansteuersignale für die Motortreiber-Endstufen-Module SMCflex-ME generiert.

Im Lieferumfang des Schrittmotor-Interfaces EMCU sind enthalten:

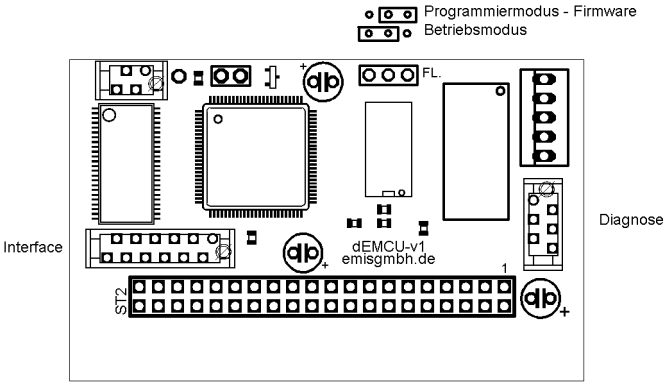
- Programm-CD mit ‚demcu-befehle' und ‚unismc2008'
- Dokumentation ‚**Die Mikrocontrollereinheit EMCU'**

- Adapterkabel zur Herstellung einer seriellen Verbindung

Das Adapterkabel hat folgende Belegung:

	SUBD-9pol
	Pin 1
TXD	Pin 2
RXD	Pin 3
	Pin 4
GND	Pin 5
	Pin 6
	Pin 7
	Pin 8
	Pin 9

2. Aufbau



Stiftleistenbelegung:

5V DC	1	2	GND
START	3	4	STOP
PAUSE	5	6	PARKEN
E1	7	8	A1
A2	9	10	A3
REFSW-X	11	12	REFANF
REFSW-Y	13	14	E2
REFSW-Z	15	16	E3
ETX	17	18	ETY
EDX	19	20	EDY
EEX	21	22	EEY
ETZ	23	24	PS1
EDZ	25	26	RG15
EEZ	27	28	USBVCC
RXD	29	30	DM
TXD	31	32	DP
GND	33	34	GND
FLASH	35	36	PS0
PS3	37	38	PS2
5V DC	39	40	GND

Bedeutung der Ein-/Ausgangsbezeichnungen:

Stromversorgung:

5V DC	Versorgungsspannung +5V DC
GND	Versorgungsspannung 0V
USBVCC	Externe USB-VCC (z.B. von PC-USB-Buchse)

Eingänge:

START	Programmstart bei Standalone-Anwendung
STOP	Programmabbruch bei Standalone-Anwendung
PAUSE	Programmpause bei Standalone-Anwendung
PARKEN	Parkposition anfahren bei Standalone-Anwendung
E1	Verknüpfungseingang
E2	reserviert
E3	reserviert
ETX	Taktsignal X-Motor
ETY	Taktsignal Y-Motor
ETZ	Taktsignal Z-Motor
EDX	Richtungssignal X-Motor
EDY	Richtungssignal Y-Motor
EDZ	Richtungssignal Z-Motor
EEX	Freigabe X-Motor
EEY	Freigabe Y-Motor
EEZ	Freigabe Z-Motor
REFSW-X	Referenzschalter X-Motor
REFSW-Y	Referenzschalter Y-Motor
REFSW-Z	Referenzschalter Z-Motor
REFANF	Referenzanforderung bei Standalone-Anwendung
PS0	Programm-Selekt-Eingang
PS1	Programm-Selekt-Eingang
PS2	Programm-Selekt-Eingang
PS3	reserviert
RG15	reserviert

Ausgänge:

A1	Schaltausgang
A2	Schaltausgang
A3	Schaltausgang

Kommunikation:

RXD	Empfangsdaten RS232
TXD	Sendedaten RS232
DM	USB-Data-Minus
DP	USB-Data-Plus

Alle Ein- und Ausgänge - mit Ausnahme von RXD und TXD - sind TTL-kompatibel.

3. Funktionsweise

Das intelligente Interface EMCU dient als Bindeglied zwischen Steuer-software und Schrittmotor-Endstufe. Das Interface enthält einen Mikro-controller, der die Erzeugung von Datensignalen für das modulare Schrittmotor-Steuersystem SMCflex übernimmt.

Es werden keine Software-Treiber benötigt. Die Ansteuerung erfolgt mit einfachen Steuerkommandos im ASCII-Format, die über die serielle Schnittstelle ausgegeben werden.

Diese Steuerkommandos können in jedem Betriebssystem, bzw. mit jeder Entwicklungsumgebung erzeugt werden, so dass hier völlig unabhängig vom Betriebssystem gearbeitet werden kann.

Es gibt prinzipiell zwei Möglichkeiten Daten über die EMCU an die Schrittmotor-Steuerkarte weiterzugeben.

Das übergeordnete Steuersystem sendet einfache Steuerbefehle im ASCII-Format an das Interface. Diese werden unmittelbar in Datensignale umgesetzt.

Das übergeordnete Steuersystem sendet einfache Steuerbefehle im ASCII-Format an das Interface. Das Interface legt diese im Festwertspeicher ab. Das übergeordnete Steuersystem wird nun nicht mehr benötigt. Das im Festwertspeicher abgelegte Programm kann per Steuertasten abgearbeitet werden.

Die beiliegende CD enthält ein einfaches Programm zur Konfiguration und Austestung des intelligenten Interfaces EMCU (demcu-befehle).

Außerdem enthält die CD ein tabellenartiges Programm (unismc2008) zur Eingabe von Vektoren bzw zur Ansteuerung der Ein-/ Ausgänge.

4. Befehlsübertragung

4.1. Datenübertragung

Die Datenübertragung zum Interface EMCU basiert auf einem ASCII- oder Textprotokoll. Alle Befehle sind aus ASCII-Zeichen aufgebaut.

Ein Handshake-Verfahren ist nicht vorgesehen. Zusätzliche Steuersignale werden für die Datenübertragung nicht verwendet.

Als Übertragungsparameter sind einzustellen:

Baudrate: 115200
Datenbit: 8
StopBit: 1
Parität: keine

4.2. Befehlsaufbau

Die Befehlssequenzen sind aus ASCII-Zeichen aufgebaut, wobei jeder Befehl in der Regel mit einem <CR> (carriage return = 13) abgeschlossen wird. Das Interface quittiert jeden Befehl entweder mit einem <READY> einem <BUSY> oder einem <ERROR>-Zeichen. Dem <ERROR>-Zeichen wird außerdem eine Fehlernummer vorausgestellt (siehe 4.3 Fehlercodes).

Abfragebefehle werden vom Interface mit einem Datenwert (ebenfalls ASCII-Zeichen) quittiert, wobei dieser Wert mit einem <READY> abgeschlossen ist.

Nach jeder Befehlsübertragung muss solange gewartet werden, bis das Interface den Befehl quittiert hat, erst dann darf ein neuer Befehl gesendet werden. Die Quittierung erfolgt in der Regel sehr schnell, kann aber je nach Befehl und Auslastung des Controllers einige Zeit in Anspruch nehmen (max. 25 ms.).

<READY> == ACK == 6 (dez)
<ERROR> == BEL == 7 (dez)
<BUSY> == NAK == 21 (dez)

Die Befehle zur Ansteuerung des Interfaces sind in zwei Gruppen unterteilt.

Zum einen gibt es so genannte Masterbefehle, welche vom Interface zu jedem Zeitpunkt verarbeitet werden können. Diese Befehle sind durch das @Zeichen gekennzeichnet.

Beispiele für **Masterbefehle** sind der „Reset“-Befehl, die „Statusabfrage“ oder der „Achsen sofort stoppen“-Befehl.

So kann z.B. während einer Vektorfahrt mittels „Achsenposition abfragen“ zwischen durch immer wieder die Position bestimmt werden.

Alle anderen Befehle werden zwar auch gleich abgearbeitet, können aber nicht zu jedem Zeitpunkt an das Interface geschickt werden. Erst wenn der vorhergehende Befehl abgearbeitet wurde, kann ein weiterer Befehl dieser Art geschickt werden.

Es empfiehlt sich, die Rückmeldungen zu überwachen. Sobald eine <READY> - Rückmeldung gesendet wurde, können neue Befehle an das Interface geschickt werden.

Für den ersten Einsatz des Interfaces muss dieses nicht konfiguriert werden, da bereits alle vorhandenen Parameter mit einem Wert voreingestellt sind.

Startgeschwindigkeit:	200 Schritte/sec
Endgeschwindigkeit Referenzfahrt:	200 Schritte/sec
Endgeschwindigkeit Vektorfahrt:	600 Schritte/sec
Rampenlänge:	200 ms
Offset nach Referenzfahrt:	10 Schritte
E1-Verknüpfung:	deaktiviert

4.3. Fehlercodes

E1: unbekannter Befehl
E2: ungültige Programm-Nr.
E3: ungültiger FAT-Eintrag
E4: Speicherüberlauf; Aktion wurde rückgängig gemacht
E5: Programm-Nr. bereits vorhanden
E6: ungültiger Parameter
E7: Arbeitsbereich verlassen
E8: Programm-Header / Befehls-Länge von max 256 Bytes überschritten

5. Befehlsbeschreibung

Nachfolgend werden alle zur Verfügung stehenden Befehle genauer beschrieben. Die Masterbefehle sind besonders gekennzeichnet!

5.1 Reset durchführen (Masterbefehl)

Dieser Befehl führt einen Reset aus und setzt das Interface in einen definierten Zustand. Dabei werden alle Positionen auf Null gesetzt.

Laufende Bewegungsabläufe werden sofort, ohne Rampe, gestoppt.

Die Achsen-Position wird bei einer „Statusabfrage“ als unbekannt signalisiert (4. Zeichen = „1“). Aus diesem Grunde muss vor einer weiteren Bearbeitung unbedingt eine Referenzfahrt durchgeführt werden.

Die gültigen Konfigurationseinstellungen werden nicht verändert.

Befehl: @R<CR>
Quittierung: <READY>

5.2 Version abfragen (Masterbefehl)

Abfrage der Interface-Version. Als Quittierung sendet das Interface die aktuelle Versionsnummer.

Befehl: @V<CR>
Quittierung: @V dEMCU-v1.00<READY>

5.3 Status abfragen (Masterbefehl)

Den aktuellen Zustand des Interfaces abfragen. Als Quittierung liefert das Interface sechs Zeichen, welche durch die Darstellung einer Null (0) oder einer Eins (1) den aktuellen Zustand der Steuerung signalisieren.

Befehl: @X<CR>
Quittierung: @Xaaaaaa<READY>

Der Buchstabe a steht hier lediglich als Platzhalter für den Status.

Die sechs Zeichen der Quittierung haben folgende Bedeutung:

1. Zeichen: Achsenbewegung
0=Maschine steht / 1=Maschine läuft

2. Zeichen: Wartezeit
0=keine Wartezeit / 1=Wartezeit läuft ab

3. Zeichen: allgemeiner Fehler
0=kein Fehler / 1=Fehler aufgetreten

Im Fehlerfall sollte immer ein Reset und eine Referenzfahrt ausgeführt werden!

4. Zeichen: aktuelle Position
0=Position bekannt / 1=Position nicht bekannt

Ist die aktuelle Position nicht bekannt, so muss in jedem Falle eine Referenzfahrt durchgeführt werden, da ansonsten definierte Bewegungsabläufe nicht mehr möglich sind.

5. Zeichen: Referenzfahrt
0=keine Referenzfahrt / 1=Referenzfahrt läuft gerade

6. Zeichen: Standalone-Anwendung
0= ~ läuft nicht / 1= ~ läuft gerade

Beispiel: @X<CR> Statusabfrage
@X100110<READY> Quittierung

Es wird eine Referenzfahrt durchgeführt (5. Zeichen) und die Maschine läuft (1. Zeichen). Die aktuelle Position ist nicht bekannt (4. Zeichen), da diese erst durch die Referenzfahrt definiert wird. Welche der Motorachsen gerade in Bewegung ist, kann nicht festgestellt werden.

Beispiel: @X<CR> Statusabfrage
@X100001<READY> Quittierung

Die Quittierung in diesem Beispiel sagt aus, dass eine Standalone-Anwendung abläuft und dass eine Achsenbewegung durchgeführt wird (1. Zeichen). Welche der Motorachsen gerade in Bewegung ist, kann nicht festgestellt werden.

5.4 Achsenposition abfragen (Masterbefehl)

Die aktuelle Position einer Achse wird abgefragt. Als Parameter muss die jeweilige Achse durch die Buchstaben X, Y, Z angegeben werden.
Als Quittierung liefert das Interface die aktuelle Position in Schritten zurück. Die Position kann sowohl positiv, als auch negativ sein, je nach Stand der Achse zum Nullpunkt.

Befehl: @La<CR>
Quittierung: @Labbbb<READY>

Der Buchstabe a steht hier nur als Platzhalter. Im Befehl ist er durch die jeweilige Achse zu ersetzen, die mit X, Y oder Z bezeichnet werden muss. In der Quittierung steht der Buchstabe b für die momentane Position, die in Schritten zurückgeliefert wird.

Beispiel: @LX<CR> Positionsabfrage X-Achse
@LX1234<READY> Quittierung
Die X-Achse befindet sich 1234 Schritte in positiver Richtung vom Nullpunkt entfernt.

Beispiel: @LY<CR> Positionsabfrage Y-Achse
@LY-1234<READY> Quittierung
Die Y-Achse steht auf –1234 Schritte zum Nullpunkt.

5.5 Startgeschwindigkeit einstellen

Festlegen der Startgeschwindigkeit für die Ansteuerung eines Schrittmotors in Schritte/sec. (Hz). Die Startgeschwindigkeit ist für alle Achsen gleich, kann also nicht für jede Achse einzeln definiert werden. Aus der Startgeschwindigkeit und der Rampenlänge ergibt sich die Startrampe und Bremsrampe.

Befehl: #Saaaa<CR>
Quittierung: <READY>

Der Buchstabe a steht als Platzhalter für die Startgeschwindigkeit.

Beispiel:	#S150<CR>	Startgeschwindigkeit auf 150 Hz einstellen
	<READY>	Quittierung

Voreinstellung: 200 Schritte/sec.

5.6 Endgeschwindigkeit einstellen

Konfiguration der Endgeschwindigkeit in Schritte/sec. Diese Einstellung gilt für alle Achsen. Die Endgeschwindigkeit kann mehrfach für verschiedene Bewegungsabläufe im Speicher des Interfaces hinterlegt werden. Es stehen max. 9 Tabellenfelder für die Endgeschwindigkeit zur Verfügung, auf die über einen Index [1-9] zugegriffen werden kann. Die Endgeschwindigkeit für die Referenzfahrt muss im Index 9 abgelegt werden.

Befehl: #Ea,bbbb
Quittierung: <READY>

Der Buchstabe a steht als Platzhalter für den Index im Tabellenfeld.
Der Buchstabe b dient als Platzhalter für die Endgeschwindigkeit.

Beispiel:	#E1,800<CR>	Es wird im Tabellenfeld 1 eine End-
	<READY>	geschwindigkeit von 800 Hz hinterlegt

Voreinstellung: Die Tabellenfelder[1..8] sind mit dem Wert 600 Hz gefüllt, die Referenzgeschwindigkeit im Tabellenfeld[9] mit 200 Hz.

Hinweis: Bei einer Vektorfahrt wird nicht mehr die Geschwindigkeit definiert, sondern nur noch der Index auf das Tabellenfeld, aus dem die Endgeschwindigkeit entnommen werden soll.

5.7 Rampenlänge einstellen

Festlegen der Rampenlänge in Millisekunden (ms). Die Rampenlänge gilt für alle Achsen und alle Bewegungsabläufe, eine Unterscheidung wird hier nicht getroffen.
Aus der Rampenlänge und der Startgeschwindigkeit errechnet sich die Startrampe und Bremsrampe, wobei Start- und Bremsrampe identisch sind.

Befehl: #Raaaa<CR>
Quittierung: <READY>

Der Buchstabe a steht als Platzhalter für die Rampenlänge.

Beispiel:	#R400<CR>	Rampenlänge auf 400 ms einstellen
	<READY>	Rückmeldung

Voreinstellung: 200 ms

5.8 Referenzreihenfolge festlegen

Wird der Eingang „Referenzanforderung“ aktiviert, erfolgt eine Referenzfahrt für die durch diesen Befehl festgelegte(n) Achse(n) und auch in der hier festgelegten Reihenfolge.

Befehl: #Habc<CR>
Quittierung: <READY>

Der Buchstaben a, b, c stehen als Platzhalter für die Motorachsen X, Y, Z.

Beispiel:	#HZXY<CR>	Vorbereitung einer Referenzfahrt
	<READY>	Rückmeldung Es soll zunächst die Z-Achse, dann die X-Achse und schließlich die Y-Achse referenziert werden. Die Referenzfahrt wird ausgeführt sobald der Eingang „Referenzanforderung“ aktiviert wird.

5.9 Offset nach Referenzfahrt einstellen

Dieser Parameter gibt die Anzahl der Schritte an, die nach einer Referenzfahrt vom Schalter weggefahren werden soll. Die Referenzfahrt besteht aus drei Bewegungsabläufen, der Suchfahrt hin zum Referenzschalter, der Freifahrt vom Schalter und der anschließenden Offsetfahrt. Für die Offsetfahrt wird hier die Schrittanzahl festgelegt.
Der Offset kann für jede Achse explizit definiert werden.

Befehl: #Oa,bbbb<CR>
Quittierung: <READY>

Der Buchstabe a steht als Platzhalter für die jeweilige Achse und ist durch X, Y, Z zu ersetzen. Der Buchstabe b steht für den Offset.

Beispiel: #OX,35<CR> die X-Achse bewegt sich nach dem Erreichen des Referenzschalters und der nachfolgenden Freifahrt vom Schalter anschließend noch 35 Schritte vom Schalter weg

Voreinstellung: 10 Schritte

5.10 Referenzfahrt ausführen

Führt eine Referenzfahrt für eine oder mehrere Achsen aus. Der Befehl selektiert die gewünschten Achsen und legt die Referenzreihenfolge fest. Die Referenzgeschwindigkeit wird dem Index 9 der Endgeschwindigkeiten entnommen. Zunächst wird die jeweilige Achse zum Referenzschalter hin bewegt, danach erfolgen Freifahrt und Offsetfahrt.

Befehl: $Habc<CR>
Quittierung: <BUSY> Referenzfahrt wird ausgeführt
<READY> Referenzfahrt ist beendet

Die Buchstaben a, b und c stehen als Platzhalter für die Achsen, die bewegt werden sollen und müssen durch X, Y oder Z ersetzt werden. Die Reihenfolge der Achsenbewegung ergibt sich aus der Reihenfolge der Achsendefinition.

Beispiel: $HZXY<CR> Referenzfahrt aller Achsen, in der
<BUSY> → <READY> Reihenfolge Z, dann X und zuletzt Y

$HY<CR> Referenzfahrt der Y-Achse,
<BUSY> → <READY> X- und Z-Achse werden nicht bewegt

5.11 Vektorfahrt ausführen

Führt eine Vektorfahrt in Schritten linear interpoliert aus. Die Schrittausgabe kann entweder relativ zur aktuellen Position oder absolut erfolgen. Die jeweilige Achse, sowie die Richtung werden ebenfalls als Parameter übergeben.

Befehl:	Lg,abbbb[,abbbbbb] [,abbbb]	
Quittierung:	<BUSY>	Vektorfahrt wird ausgeführt
	<READY>	Vektorfahrt ist abgeschlossen

Der Buchstabe g steht für als Platzhalter für die Geschwindigkeit und muss durch den Index auf das Tabellenfeld ersetzt werden.
Der Buchstabe a steht für die jeweilige Achse und muss durch X, Y, Z oder x, y, z ersetzt werden.
Große Buchstaben (X, Y, Z) stehen für eine Bewegung absolut zum Nullpunkt, kleine Buchstaben (x, y, z) für eine relative Bewegung zur aktuellen Position.
Der Buchstabe b steht für die Anzahl der Schritte. Eine positive Zahl bewegt die Achse in positive Richtung, eine negative Zahl in negative Richtung.

Beispiel:	L1,X200,Y500<CR>	Die Maschine fährt zur absoluten Schrittposition X=200, Y=500, mit der Geschwindigkeit aus dem Tabellenfeld 1.
Beispiel:	L1,x500,y1000<CR>	Die Maschine fährt von der aktuellen Position aus 500 Schritte in X-Richtung und 1000 Schritte in Y-Richtung (linear interpoliert), mit der Geschwindigkeit aus dem Tabellenfeld 1.
Beispiel:	L2,x-50,y-100<CR>	Die Maschine fährt von der aktuellen Position aus 50 Schritte auf der X-Achse und 100 Schritte auf der Y-Achse jeweils in negativer Richtung (linear interpoliert), mit der Geschwindigkeit aus dem Tabellenfeld 2.

Alle Beispiele haben folgende Rückmeldungen

<BUSY>	Rückmeldung, die anzeigt, dass die Verfahrbewegung ausgeführt wird
<READY>	Rückmeldung, dass das Ziel erreicht wurde

5.12 Achsen anhalten (Masterbefehl)

Mit diesem Befehl werden alle Achsen mit Rampenfahrt angehalten. Der aktuelle Verfahrbefehl wird beendet. Die Positionsinformationen bleiben erhalten.

Befehl: @B<CR>
Quittierung: <READY>

5.13 Achsen sofort stoppen (Masterbefehl)

Alle Achsen werden sofort, ohne Rampenfahrt gestoppt. Es gehen alle Positionsdaten verloren. Vor einer Weiterfahrt sollte deshalb eine Referenzfahrt durchgeführt werden, um die Achsen wieder in einen definierten Zustand zu setzen. Dieser Befehl kommt einem Reset gleich und sollte deshalb nur in Notfällen eingesetzt werden.

Befehl: @S<CR>
Quittierung: <READY>

5.14 Pause setzen (Masterbefehl)

Dieser Befehl gibt eine Pause aus und unterbricht den aktuellen Programmablauf. Alle Achsen werden mit Rampe gestoppt, die Positionen bleiben erhalten. Eine ablaufende Wartezeit ruht. Der Eingang E1 wird nicht mehr ausgewertet.
Der Programmablauf wird erst durch „Pause beenden“ fortgesetzt.
Ein „Reset“-Befehl oder „Achsen sofort stoppen“-Befehl hebt die Pause ebenfalls auf.

Befehl: @A<CR>
Quittierung: <READY>

5.15 Pause beenden (Masterbefehl)

Dieser Befehl beendet die mit dem „Pause setzen“-Befehl eingeleitete Unterbrechung und fährt mit der Abarbeitung der Befehlskette fort. Wurde vorher keine Pause gesetzt, so hat dies keine Auswirkungen; der Befehl wird einfach ignoriert. Da es sich um einen Masterbefehl handelt, wird dieser sofort ausgeführt.

Befehl: @C<CR>
Quittierung: <READY>

5.16 Ausgang setzen

Schaltet ein Ausgangssignal auf einer Datenleitung, welches zur Steuerung einer Bohrspindel oder Kühlmittelpumpe hergenommen werden kann. Es stehen insgesamt 3 Ausgänge zur Verfügung. Als Parameter wird der Zustand des Signals definiert. Eine 0 für Signal ‚aus‘ (Low), eine 1 für Signal ‚ein‘ (High).

Befehl: Aa,b<CR>
Quittierung: <READY>

Der Buchstabe a steht als Platzhalter für den Signalausgang [1, 2 oder 3].
Der Buchstabe b steht als Platzhalter für den Signalzustand und muss durch 0 oder 1 ersetzt werden.

Beispiel:	A1,1<CR> <READY>	Ausgang 1 wird gesetzt
Beispiel:	A1,0<CR> <READY>	Ausgang 1 wird gelöscht

5.17 Wartezeit einfügen

Dieser Befehl fügt eine vorgegebene Zeitspanne in den aktuellen Arbeitsablauf ein. Die Zeit wird in Millisekunden angegeben. Die weitere Ausführung wird für eine gewisse Zeit unterbrochen, um z.B. ein Ventil zu schalten. Nach Ablauf der Zeit werden die nachfolgenden Befehle bearbeitet.
Die max. mögliche Wartezeit beträgt 3.600.000 ms (=1 Std).

Befehl: Waaaa<CR>
Quittierung: <BUSY>
<READY>

Beispiel:	W250<CR>	Wartezeit 250 ms
	<BUSY>	Rückmeldung zeigt an: Wartezeit läuft
	<READY>	Rückmeldung zeigt an: Wartezeit ist abgelaufen

5.18 E1-Verknüpfung

Befehl: &E1,b<CR>
Quittierung: <READY>

Der Buchstabe b steht als Platzhalter und muss durch 0 oder 1 ersetzt werden.

Beispiel:	&E1,0<CR>	Verknüpfung deaktiviert
	<READY>	Rückmeldung Die nachfolgenden Befehl werden ausgeführt, ohne Eingang E1 zu berücksichtigen
Beispiel:	&E1,1<CR>	Verknüpfung aktiviert
	<READY>	Rückmeldung Die nachfolgenden Befehl werden in Abhängig-keit von Eingang E1 ausgeführt. Sie werden ausgeführt sobald der Eingang E1 „High-Potential“ annimmt.

Voreinstellung: Verknüpfung deaktiviert

5.19 Eingang abfragen (Masterbefehl)

Befehl: @In
Quittierung: @In b<READY>

Bitte beachten Sie das Leerzeichen in der Rückmeldung nach @In.

Der Buchstabe n [0-F] steht als Platzhalter für die verschiedenen Eingänge:

0: Eingang FLASH
1: Eingang START
2: Eingang STOP
3: Eingang PAUSE
4: Eingang PARKEN
5: Eingang REFERENSCHALTER X
6: Eingang REFERENSCHALTER Y
7: Eingang REFERENSCHALTER Z
8: Eingang REFERENZANFORDERUNG
9: Eingang (ohne Bezeichnung)
A: Eingang (ohne Bezeichnung)
B: Eingang E1
C: Eingang PS0
D: Eingang PS1
E: Eingang PS2
F: Eingang (ohne Bezeichnung)

Der Buchstabe b[0,1] steht als Platzhalter den Zustand des Eingangs.

Beispiel:	@I1<CR>	Abfrage des START-Eingangs.
	@I1 0<READY>	Quittierung
		Der START-Eingang ist nicht gesetzt.

5.20 Programm schreiben

Dieser Befehl bewirkt, dass nachfolgende Befehle in einem der möglichen Programmspeicherplätze abgelegt werden.
Ingesamt stehen 7 solche Programmspeicherplätze mit jeweils 65536 Bytes zur Verfügung. Sollte ein Programm mehr als 65536 Bytes benötigen, reduziert sich die maximale Anzahl der Programmspeicherplätze.

Befehl:	*PWn<CR>	(ProgrammWrite...)
	*PWn<READY>	Rückmeldung
	Hh<STX>	Programm-Header+Programmstartkennzeichen
	<READY>	Rückmeldung
	C<CR>	Befehl
	<READY>	Rückmeldung
	c<CR>	Befehl
	<READY>	Rückmeldung
	C<ETX>	Befehl
	<READY>	Rückmeldung

Der Buchstabe n steht als Platzhalter für eine Programm-Nr. [1...7]
Die Buchstaben Hh stehen als Platzhalter für den Programm-Header.
Die Buchstaben CcC stehen als Platzhalter für die Programm-Befehlskette.

Beispiel:	*PW1<CR>	Vorbereitung zur Programm-Übertragung
	*PW1<READY>	Quittierung mit <READY> signalisiert Bereitschaft zum weiteren Datenempfang:

Es folgt zunächst der Programm-Header.
Siehe dazu unter **5.22 „Programm-Header lesen“**
Ein Programm-Header ist jedoch nicht zwingend nötig.

Beispiel für einen Programm-Header:
vektoren.etab | 01.01.2008 | 08:00:00 | 37<STX>

Das Zeichen **<STX>** bildet das so genannte Programmstart-kennzeichen. Es kennzeichnet einerseits das Ende des Programm-Headers als auch den Beginn der Befehlskette.

Auch wenn kein Programm-Header verwendet wird, muss das Programmstartkennzeichen <STX> gesendet werden!

Nun kann das eigentliche Programm gesendet werden.
Die einzelnen Befehle sind mit <CR> abzuschliessen.

Beispiel für eine Befehlskette:

A1,1<CR>	Ausgang 1 setzen
<READY>	Quittierung
L1,x100<CR>	Vektorfahrt x-Achse: 100 Schritte
<READY>	Quittierung
W250<CR>	Wartezeit 250ms
L1,y200<CR>	Vektorfahrt y-Achse: 200 Schritte
<READY>	Quittierung
A1,0<ETX>	Ausgang 1 zurücksetzen;
<READY>	

Das <ETX>-Zeichen definiert das Ende der Befehlskette und muss daher an den letzten Befehl statt des <CR> angehängt werden.

Im obigen Beispiel konnte das Programm 1 erfolgreich übertragen werden!

Um das Programm starten zu können, muss es zunächst durch Anlegen von Programm-Select-Signalen an PS0..PS2 ausgewählt werden. Es bestehen folgende Abhängigkeiten:

	PS2	**PS1**	**PS0**
Programm 1	0	0	1
Programm 2	0	1	0
Programm 3	0	1	1
Programm 4	1	0	0
Programm 5	1	0	1
Programm 6	1	1	0
Programm 7	1	1	1

Das Programm 1 kann demnach mit ‚high' an PS0 sowie ‚low' an PS1 und PS2 selektiert werden.

Der Start des Programms erfolgt über den „START"-Eingang.

5.21 Programm lesen

Der Programm-Header und die Programm-Befehlskette wird ausgelesen.

Befehl: *PRn<CR> (ProgrammRead...)
Quittierung: *PRn Hh<STX>C<CR>c<CR>C<ETX><READY>

Bitte beachten Sie das Leerzeichen in der Rückmeldung nach *PRn.

Der Buchstabe n steht als Platzhalter für eine Programm-Nr. [1...7].
Man kann mit n gleich [a] oder [A] den kompletten Festwertspeicher auslesen.
Die Buchstaben Hh stehen als Platzhalter für die Programm-Header.
Die Buchstaben CcC stehen als Platzhalter für die Programm-Befehlskette.

Beispiel: *PR1<CR> Programm 1 soll ausgelesen werden.

Die Abfrage könnte folgendermaßen quittiert werden:

*PR1 vektoren.etab | 01.01.2008 | 08:00:00 | 37<STX>
A1,0<CR>L1,x100<CR>W250<CR>L1,y200<CR><A1,0><ETX>
<READY>

Besonderheit:

Bei der Verwendung von *PRa erhält man ein Speicherabbild des kompletten Festwertspeichers (=458.752 Bytes).
Mittels der FAT ist eine Zuordnung der Programme zu den 7 Programmspeicherplätzen möglich.

5.22 Programm-Header lesen

Der Programm-Header umfasst alle Zeichen bis zum **Programmstart-kennzeichen <STX>**.
Der Programm-Header ist auf 256 Bytes begrenzt und bietet z.B. die Möglichkeit Windows-Datei-Informationen abzulegen, z.B. Dateiname - Dateidatum - Dateiuhrzeit – Dateigröße. So kann durch Vergleich die Aktualität der Daten im Programmspeicher festgestellt werden. Der Programm-Header kann natürlich auch für andere Zwecke verwendet werden.

Befehl: *PRnH<CR> (ProgrammRead...Header)
Quittierung: *PRnH Hh<READY>

Bitte beachten Sie das Leerzeichen in der Rückmeldung nach *PRnH.

Der Buchstabe n steht als Platzhalter für eine Programm-Nr. [1...7].
Die Buchstaben Hh stehen als Platzhalter für den Programm-Header.

Beispiel: *PR1H<CR>Programm-Header 1 soll ausgelesen werden.

Die Abfrage könnte folgendermaßen quittiert werden:

*PR1H vektoren.etab | 01.01.2008 | 08:00:00 | 37<READY>

Die Befehlskette vom Programm 1 entstammt der Datei vektoren.etab, die am 01.01.2008 um 08:00:00 erzeugt wurde und 37 Bytes d.h. 37 Zeichen enthält.

5.23 Programmgröße ermitteln

Dieser Befehl ermöglicht die benötigten Speicherbytes eines Programms zu ermitteln.

Es könnte z.B. die Notwendigkeit bestehen, im Vorfeld die Größe des Programms zu kennen, welches in einem Programmspeicher ablegt werden soll bzw. ob das Programm noch vergrößert werden kann.

Die Daten werden nicht im Festwertspeicher abgelegt!

Befehl:	*PS<CR>	(ProgrammSize)
	<READY>	Rückmeldung
	Hh<STX>	Programm-Header+Programmstartkennzeichen
	<READY>	Rückmeldung
	C<CR>	Befehl
	<READY>	Rückmeldung
	c<CR>	Befehl
	<READY>	Rückmeldung
	C<ETX>	letzter Befehl+Programmendekennzeichen
	aaaa<READY>	Rückmeldung incl. Programmgröße

Die Buchstaben Hh stehen als Platzhalter für den Programm-Header.
Die Buchstaben CcC stehen als Platzhalter für die Programm-Befehlskette.
Die Buchstaben aaaa stehen als Platzhalter für Programmgröße (in dezimaler Schreibweise).

Beispiel: siehe „Programm schreiben“
jedoch mit folgender letzter Rückmeldung

34<READY>	Das Programm 1 würde 34 Bytes im Programmspeicher belegen.

5.24 Programm löschen

Das angegebene Programm wird aus dem Programmspeicher gelöscht.
Der Löschvorgang dauert ca. 0.7 sec pro Programmsektor (65536 Bytes).

Befehl: *PEn<CR> (ProgrammErase...)
Quittierung: *PEn<BUSY>
<READY>

Der Buchstabe n steht als Platzhalter für eine Programm-Nr. [1...7].
Man kann mit n gleich [a] oder [A] auch alle Programme gleichzeitig löschen.

Beispiel:	*PE1<CR>	Programm 1 soll gelöscht werden
	*PE1<BUSY>	Quittierung zunächst mit <BUSY>;
		nach erfolgtem Löschen wird dies durch
	<READY>	angezeigt

Beispiel:	*PEa<CR>	alle Programme sollen gelöscht werden
	*PEa<BUSY>	Quittierung zunächst mit <BUSY>;
		nach erfolgtem Löschen wird dies durch
	<READY>	angezeigt

5.25 FAT lesen

Die FAT (File Allocation Table) wird mit diesem Befehl gelesen.
Die Einträge der FAT enthalten die Start-Adressen und End-Adressen der Programme. Das Programm 1 kann irgendwo im Speicherbereich des Flash-Speichers abgelegt sein. Die FAT gibt Auskunft wo das Programm liegt.
Um die Speicherverwaltung braucht sich der Anwender nicht zu kümmern!

Befehl: *FRn<CR>
Quittierung: *FRn ssss,eeee<READY>

Bitte beachten Sie das Leerzeichen in der Rückmeldung nach *FRn.

Der Buchstabe n steht als Platzhalter für eine Programm-Nr. [1...7].
Die Buchstaben s stehen als Platzhalter für die Start-Adresse.
Die Buchstaben e stehen als Platzhalter für die End-Adresse.

Hinweis: Ein Strich als Rückgabewert einer Adresse bedeutet, dass kein Programm abgelegt wurde.
Zu beachten ist außerdem das Leerzeichen in der Rückmeldung nach *FRn.

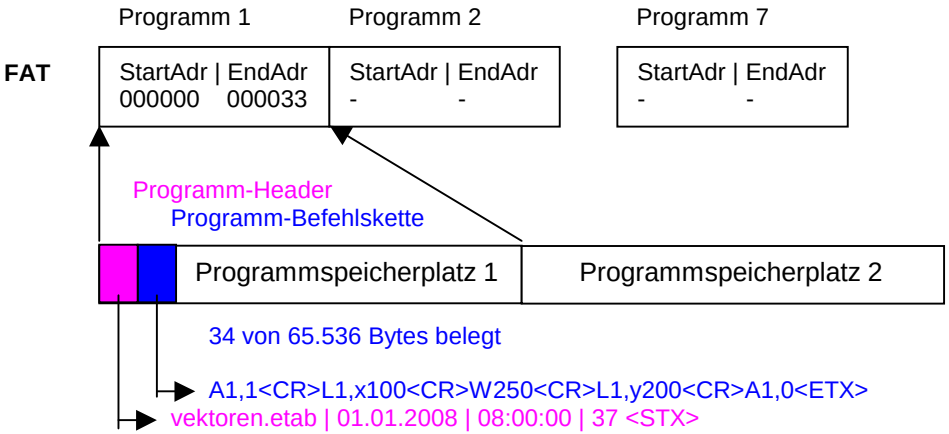
Beispiel: *FR1<CR> Der FAT-Eintrag des 1. Programms soll gelesen werden.

Die Abfrage könnte folgendermaßen quittiert werden:
*FR1 0,33<READY>
Das Programm 1 beginnt bei Adresse 0 und endet bei Adresse 33.

Beispiel: *FR2<CR> Der FAT-Eintrag des 2. Programms soll gelesen werden.

Die Abfrage könnte folgendermaßen quittiert werden:
*FR2 -,-<READY>
Das Programm 2 existiert (noch) nicht.

Symbolhafte Darstellung der Speicherverwaltung:



Insgesamt stehen 7 Programmspeicherplätze mit jeweils 65536 Bytes zur Verfügung.

Die Befehlskette eines Programms wird in einem freien Programm-speicherplatz abgelegt. Es kann somit durchaus vorkommen, dass die Befehlskette des 1. Programms im Progammspeicherplatz 3 abgelegt wird. In der FAT wird die Zuordnung von Programm zum Programm-speicherplatz verwaltet; in der FAT ist deswegen die StartAdresse des Programms hinterlegt.

Sollte die Befehlskette eines Programms mehr als 65536 Bytes benötigen, wird ein weiterer Programmspeicherplatz dafür herangezogen. Es können dann nicht mehr max. 7 Programme sondern nur noch max. 6 Programme abgelegt werden.

> Es ist auch möglich, dass ein einziges Programm mehr als 393.216 Bytes belegt. Es würde somit alle Programmspeicherplätze belegen.
> Es kann kein weiteres Programm im Festwertspeicher abgelegt werden!

Sollte die Befehlskette eines Programms einmal nicht im Festwertspeicher untergebracht werden können, so ist die Belegung des Festwertspeichers (FAT auslesen) zu überprüfen.