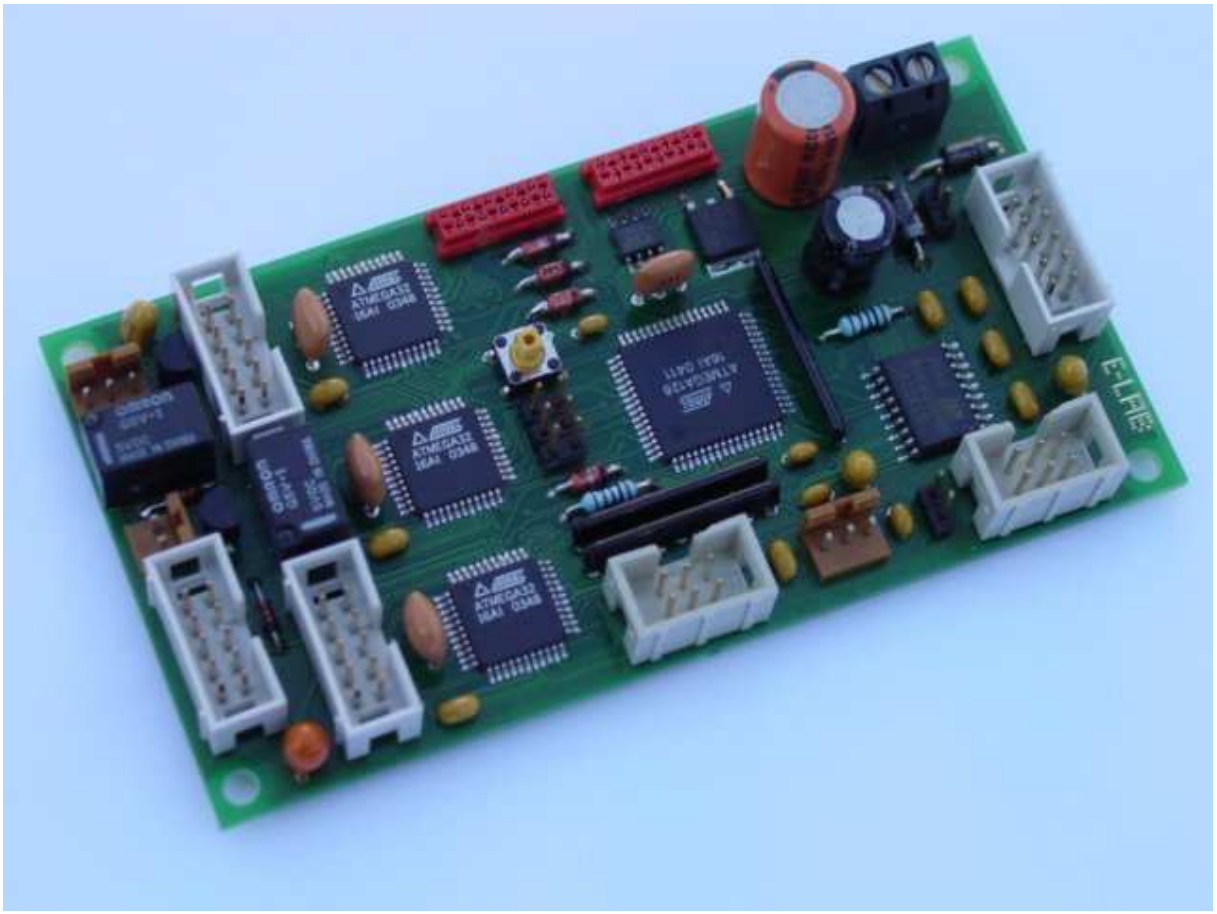
3-Achsen Stepper Controller

Hard- und Software Manual



E-LAB 3-Achsen Stepper Controller

Inhalt

E-LAB 3-Achsen Stepper Controller

Allgemeines und Aufbau

Mit dem Stepper-Controllerboard (ELAB2053) und dem Leistungstreiber (E-LAB2052 bzw. E-LAB2051) haben Sie die Möglichkeit drei Achsen unabhängig voneinander zu steuern.

Dies macht die Produkte in vielerlei Hinsicht zum Allrounder; es können z.B. Fräsen angesteuert werden.

Das Stepper-Controllerboard:

Das Stepper-Controllerboard dient zur Ansteuerung der Leistungstreiber. Mit diesem Board können Sie bis zu drei Achsen steuern. Die Steuerung wird von einem Masterprozessor (Mega128), und drei Coprozessoren (Mega32) übernommen. Die Kommunikation zwischen den Einheiten findet über einen I2C-Bus und Status Leitungen statt. Des weiteren befindet sich auf dem Board ein Temperatursensor, der zur Überwachung der Gehäuse Innentemperatur dienen kann.

> ***Wichtig:*** *Bitte beachten Sie bei der Installation der Boards, dass eine termische Verbindung mit Leistungstreiben bestehen muß. In den meisten fällen reicht dafür die Gehäuseluft aus.*

Die Kommunikation mit der Außenwelt findet über eine RS232 Verbindung statt. Dies gibt Ihnen die Möglichkeit, mit vielen Geräten eine Kommunikation aufzubauen, z.B. einen PC oder gar eine SPS.

Die Versorgungsspannung kann in einem gewissen Rahmen frei gewählt werden. Für eine 5Volt= Versorgung setzen Sie bitte den Jumper J1. Für Versorgungsspannungen über 5Volt entfernen Sie diesen Jumper. Die Versorgungsspannung sollte sich dann in einem Rahmen von 6 bis 9Volt= bewegen.

> ***Wichtig:*** *Auf eine saubere Trennung der Digital und Leistungselektronik ist in jedem Fall zu achten. Gerade bei falscher Projektierung ihres Netzteiles kann es hier zu großen Problemen kommen.*

Die Umgebungstemperaturen sind von 0 bis +40°Celsius nicht kondensierend. Andere Einsatzbedingungen klären Sie bitte mit dem Hersteller ab.

Für die Programmierung der Prozessoren sind alle JTAG-Ports über Pfostenstecker nach außen geführt.

Die Leistungstreiber E-LAB2051 und E-LAB2052:

Der Leistungstreiber E-LAB2051 dient zur Ansteuerung von einem 2-Phasen Schrittmotor im wave drive Betrieb. Die Grenzdaten sind 1 Ampere bei 35Volt. Bei Einsatztemperaturen über 40°Celsius ist u.U. der Strom zu reduzieren, damit der Leistungshalbleiter die entstehende Verlustleistung an die Umgebung ordentlich abführen kann. Der Leistungskreis ist Überstrom und Übertemperatur geschützt.

Der Leistungstreiber E-LAB2052 arbeitet nach dem gleichen Prinzip,
der maximal Zulässige Strom liegt hier bei 2Ampere.

Betriebsmodi

Die hier und im weiteren Verlauf aufgeführten Parameter und Kommando Namen (Mnemonics) sind alle aus der Sicht der übergeordneten Steuerung (Host), z.B. PC gesehen und beschrieben.

a. Der Arbeits-Mode (Feed)

Im Arbeits-Mode läuft der Schrittmotor unterhalb des Wertes der Start bzw. Stoprampe. Dies bedeutet, der Schrittmotor hat hier auch sein größtes Drehmoment. Er kann jederzeit sofort gestartet und gestoppt werden, ohne dass Schrittverluste auftreten.

Der Modus ist z.B. sinnvoll für Fräsen oder Plotter. Bei Fräsen muss mit einem geringen Vorschub, aber doch mit viel Kraft gearbeitet werden. Bei Plotten würden zu schnelle Bewegungen des Wagens zu einem Abriss der Tinte führen.

Parameter und Kommandos:

SetFeedSpeed	bestimmt die Steprate im Arbeits Mode
SetFeedStepsX	startet lineare X-Bewegung zur Zielposition im Arbeits Mode
SetFeedStepsY	startet lineare Y-Bewegung zur Zielposition im Arbeits Mode
SetFeedStepsZ	startet lineare Z-Bewegung zur Zielposition im Arbeits Mode
SetFeedStepsXYZ	startet beliebige 3D-Bewegung im Arbeits Mode
ForceFlushBuffer	der XYZ-Feed Buffer wird geleert
EndOfFeed	schliesst die 3D-Bewegung ab.
ForcePanicStop	stoppt sofort jegliche Bewegung, Position ist dann unbekannt

b. Der Positionier-Mode

Im Positionier-Mode (FastMove) wird die Startrampe bis zum Plateau hochgefahren und gehalten. Vor Erreichen der End-Position wird der Motor über die Stoprampe abgebremst. Wird die Position vor Erreichen der Maximalfrequenz erreicht, geht der Schrittmotor von der Startrampe in die Stoprampe über, damit die Position sauber angefahren werden kann. Dieser Modus dient zur
schnellen Positionierung der Motoren. Es wird nicht interpoliert und die Motoren werden unabhängig voneinander bewegt.

Parameter und Kommandos:

SetStartSpeed	bestimmt die Steprate beim Start einer Rampe zur Zielfahrt
SetAcceleration	bestimmt die Beschleunigung und Abbremsung bei einer Zielfahrt
SetMaxSpeed	bestimmt die maximale Steprate bei einer Zielfahrt
SetFastMoveXY	startet eine Zielfahrt mit Rampen und Plateau
SetFastMoveXYZ	startet eine Zielfahrt mit Rampen und Plateau
SetFastMoveZ	startet eine Zielfahrt mit Rampen und Plateau
ForcePanicStop	stoppt sofort jegliche Bewegung mit Schrittverlusten.
ForceStopMove	stoppt sofort jegliche Bewegung mit Bremsrampen ohne Schrittverluste

c. Der Jog-Modus

Der Jog-Mode dient hauptsächlich zur manuellen Positionierung. Der Schrittmotor fährt sehr langsam seine Position an. Er kann z.B. zum anfahren einer Null Position benutzt werden, der Benutzer steuert die Achsen z.B. über einen Joystick am PC.

Da hier die Stepraten für die Achsen vogegeben werden ohne dass dazu eine Rampe benutzt werden kann, muss der steuernde Host, z.B. der PC, diese Stepraten langsam erhöhen oder erniedrigen um Schrittverluste oder gar den Synchronisations Verlust der Motoren zu vermeiden.

Parameter und Kommandos:

SetJogSpeed	startet bzw. beschleunigt das Fahren der Motoren.
ForcePanicStop	stoppt sofort jegliche Bewegung mit Schrittverlusten.

d. Der Move-Modus

Hierzu gibt es zwei Kommandos die sich in erster Linie durch die Art des Stops unterscheiden

Der ReferenceMove fährt eine Rampe hoch in der gewählten Richtung und behält seine Steprate bei bis entweder ein Stop Befehl vom Host kommt oder der zu dieser Achse gehörende Referenz Schalter aktiviert wird. Dadurch wird ein Schnellstop eingeleitet.

Der VelocityMove fährt eine Rampe hoch in der gewählten Richtung und behält seine Steprate bei bis ein Stop Befehl vom Host kommt. Während dieser Fahrt kann die Steprate mit dem gleichen Befehl kontinuierlich geändert werden.

Allgemeine Hinweise zu den Kommandos

Sind Positionier- oder Arbeits Kommandos beim Controller in Arbeit, so muss der Host den Status des Controllers abfragen, bevor er weitere Kommandos absetzen kann.

RequestMachineState	gibt den allgemeinen Status der Maschine zurück.
RequestTemperature	gibt die Umgebungs Temperatur des Controllers zurück
RequestAxeState	gibt den Status der drei Motoren/Achsen zurück. Das ist die Haupt Poll Funktion, die benutzt werden muss bevor ein neues Kommando abgesetzt werden kann. Es muss solange gewartet werden, bis der Status = Ruhezustand ist.
RequestRelPosition	wird eine Ziel-Fahrt (Move) durch das Stop Kommando abgebrochen, können die restlichen, noch nicht abgearbeiteten Schritte hiermit abgefragt werden.
RequestSync	dient zum Neu-Synchronisieren, falls durch Übertragungsfehler die Synchronisation zwischen Host und Controller verloren gegangen ist.
RequestStart	dient zur Abfrage der optionalen Start Taste des Controllers.

E-LAB 3-Achsen Stepper Controller

FAQ und Wissenswertes zu Schrittmotoren

Warum wird der Schrittmotor mit steigender Drehzahl wärmer?
Der Schrittmotor hat nicht nur ohmsche Verluste, sondern auch magnetische. Diese steigen mit zunehmender Drehfeldfrequenz an, da durch die Umpolung die Verluste in dem magnetischen Wertstoffen steigen.

Was ist die beste Start- und Stoppfrequenz?
Diese Werte hängen stark von der Last ab, die der Motor treiben muss. Wird die Frequenz zu hoch gewählt, kann der Motor durch die Trägheit der Masse dem Drehfeld nicht folgen. Dies kann zu Schrittverlusten führen oder zum Stillstand des Motors.
Ist die Frequenz zu niedrig gewählt, kann es zu Resonanzen innerhalb des Systems kommen die unerwünscht sind. Letzteres kann durch Micro-Stepping Endstufen verhindert werden, wie z.B. die E-LAB Schritmotor Treiber Boards. Diese machen 8 Mikro-Schritte pro Motor Vollschritt.

Warum läuft der Schrittmotor am Schreibtisch (im uneingebauten Zustand) nicht hoch?
Auch hier spielen Resonanzen eine große Rolle. Der Schrittmotor arbeitet nach dem Prinzip eines Synchronmotors, dies hat eine hohe Steifigkeit zur Folge (große Kraftauswirkung bei kleiner Änderung der Lage). Die Folge der hohen Steifigkeit ist eine Tendenz zur Eigenresonanz die unter 100KHz liegt. Beim Betrieb in der Nähe der Eigenresonanz kann es zu einer Resonanzkatastrophe kommen und der Schrittmotor gerät außer Tritt. Die geringe Dämpfung des Einbaus oder der Last verhindert dies in den meisten Fällen.
Aber auch hier sorgt das MicroStepping weitgehend für Abhilfe.

Warum steht auf meinem Typenschild z.B. 6Volt, ich betreibe den Schrittmotor aber bei weit höheren Spannungen?
Diese Bezeichnungen stammen noch aus der Urzeit der Schrittmotortechnik. Damals wurden die Schrittmotoren im Konstantspannungsbetrieb betrieben, bei dem sich ein Strom einstellte. Heute werden die Schrittmotore im Konstantstrombetrieb angesteuert, welches eine sehr hohe Dynamik ermöglicht.

Die E-LAB Schrittmotor Endstufen arbeiten mit Konstantstrom.

Bei der Wahl der Motoren ist darauf zu achten, dass diese sehr niederohmig sind. Bei hoch ohmigen Motoren und hohen Schritt Frequenzen muss sonst mit extrem hohen Spannungen gearbeitet werden um den erforderlichen konstanten Stromwert einhalten zu können.

RS232-C

Zur Kommunikation mit der übergeordneten Steuereinheit dient eine RS232-C Verbindung. Die maximale Übertragungsstrecke liegt bei 15 Meter oder einer Gesamtleitungskapazität von 2,5nF. Durch den Einsatz guter Kabel lassen sich bis zu 30 Meter und mehr herausholen.
Die Logische 1 (Mark) entspricht einem Spannungspegel von –3 bis zu –15Volt, die Logische 0 (Space) entspricht einem Pegel von +3 bis +15Volt. Der Bereich von –3 bis +3Volt ist nicht definiert.

> ***Wichtig:*** *Die Ein- und Ausgänge der RS232 Schnittstelle sind nicht TTL oder CMOS kompatibel. Falsche Handhabung kann zur Zerstörung des Schnittstelleintreibers oder Ihrer Elektronik führen.*

Die Einstellung der Baudrate liegt bei 57600bits/sec. Desweiteren werden 8 Datenbits und 2 Stoppbit übertragen. Ein Paritycheck findet nicht statt.

> ***Wichtig:*** *Die Baudrate sollte möglichst genau eingehalten werden, die max. tolerierte Abweichung liegt bei 3% vom Wert.*

10 PIN PFOSTENSTECKER CONTROLLERBOARD	9 POL D-SUB STECKER PC
PIN 1 – N.C.	PIN 1 – N.C.
PIN 2 – N.C.	PIN 6 – N.C.
PIN 3 – Transmit Data	Auf PIN 2 – Receive Data
PIN 4 – N.C.	PIN 7 – N.C.
PIN 5 – Receive Data	Auf PIN 3 - Transmit Data
PIN 6 – N.C.	PIN 8 – N.C.
PIN 7 – External Reset	Auf PIN 4 – Request to Send
PIN 8 – N.C.	PIN 9 – N.C.
PIN 9 – GND	Auf PIN 5 – GND
PIN10 – N.C.	--

Anschlußbelegung Controllerboard <-> PC

Um den externen Reset benutzen zu können, müssen Sie den Jumper J7 setzen.
Der Reset wird durch eine Logische 1 auf der RTS-Leitung ausgelöst. Diese muss mindestens für 100msec im High Zustand verharren, um den Prozessoren einen sauberen Reset zu ermöglichen. Nachdem die Leitung auf Low gezogen wurde, sind weitere 100mSec zu warten.

> ***Wichtig:*** *Achten Sie bitte besonders auf die Einstellungen für das Handshake auf Ihrem PC. Ein Hardwarehandshake setzt vor jedem gesendeten Byte das Controllerboard in den Resetzustand.*

Die RS232 Schnittstellentreiber sind zwar bis zu 15kVolt ESD- und Kurzschluss geschützt, aber ein Hot Plug sollte trotzdem vermieden werden.

Die Abschirmung der RS232-Leitung ist am Gehäuse des Controllerboardes und am Gehäuse des PC anzubringen. Bei sehr rauen Bedingungen ist u.U. eine Potenzialtrennung, die optisch oder induktiv erfolgen kann, vorzusehen.

Da die meisten Laptops inzwischen nur noch mit einer USBSchnittstelle ausgerüstet sind, können Sie bei uns einen USB auf RS232 Wandler erwerben.

E-LAB 3-Achsen Stepper Controller

Aufbau der Befehle und Parameter

Die gesamte Kommunikation geht von der übergeordneten Steuereinheit (Host) aus.

Die Befehle setzen sich aus einem Steuerzeichen und den entsprechenden Parametern zusammen. Steuerzeichen werden durch ihr Mnemonic in spitzen Klammern dargestellt.

Boolesche Parameter werden durch ganze Bytes repräsentiert, d.h. eine Logische 1 ist gleichwertig mir einem $FF eine Logische 0 wird durch $00 dargestellt.

Zahlenwerte werden als Hexstring übergeben, d.h. die Zahl 28 wird als „1C“ dargestellt, dies ist ein zwei Zeichen langer String. Bitte beachten Sie bei der Datenübertragung die Stringlänge:

Ein Byte entspricht 2 Zeichen;
Ein Int entspricht 4 Zeichen;
Ein Long entspricht 8 Zeichen;

Parameter für die einzelnen Achsen (X/Y/Z) werden durch ein Komma, gefolgt von einem Semikolon und einem Punkt abgeschlossen (XX,YY;ZZ.).

Beispiel:

<DC4>1F1F1F1F,1E1E1E1E;FFFFFFFF.

Wird der Befehl von dem Controllerboard erkannt und kann es diesen auswerten, wird dies mit einem <ACK> beantwortet. Im Fehlerfall antwortet das Controllerboard mit einem <NAK>. Kommt ein NAK zurück so wurde hier (meistens) ein falscher Hexstring übertragen.

Bei Befehlen, bei denen Werte vom Controller an den PC übergeben werden, werden die Daten auch in Hexadezimaler Darstellung übergeben. Als erstes wird jedoch der Befehl wiederholt.

Beispiel:

PC -> Controller

<ETX>

Controller -> PC

<ETX>1F1F1F1F,1E1E1E1E;FFFFFFFF.

Zur Kenntlichmachung welche Daten vom Host kommen und welche vom Controller, werden Host Daten in rot und Controller Daten in blau dargestellt.

Die Zeichen **, : .** dienen nicht nur zur Trennung der Parameter sondern auch zur Überprüfung des Telegramms.

Allgemeine Befehle

RequestTemperature

Lese die aktuelle Temperatur der Controllerboards:

Dieser Befehl liest die aktuelle Temperatur des Controllerboardes. Der PC kann diese auswerten und z.B. eine Warnmeldung an den Benutzer ausgeben. Bitte beachten Sie, dass dies erfolgen muss bevor die Leistungsendstufen auf Übertemperatur abschalten. In diesem Fall würden Schritte verloren gehen und die aktuelle Position ist nicht mehr bekannt.

Der PC sendet ein <ETB> an den Controller. Dieser antwortet mit <ETB> gefolgt von einem 2Char Hexstring, der die Temperatur enthält und ein abschliessendes „#“.
Beispiel:

<ETB>
<ETB>14# *dies währen 20°Celsius*

RequestRelPosition

Lese die relative Position der Achsen:

Dieser Befehl dient zum Lesen der relativen Position der Schrittmotoren bezogen auf die letzten Koordinaten. Die Ergebnis Werte sind vom Typ Long Integer (32bits).
Ein Verlust von Schritten führt wegen der fehlenden Rückkopplung zu falschen Werten.
Dieses Kommando ist nur in Zusammenhang mit dem Befehl „ForceStopMove“ sinnvoll, denn nur danach ist die aktuelle Maschinen Position beim Host nicht bekannt.

Der PC sendet ein <ETX> an den Controller, dieser antwortet mit
<ETX>“XXXXXXXX,YYYYYYYY;ZZZZZZZZ.“.
Beispiel:

<ETX>
<ETX>FFFFFFFF,00000010;000000FF. *dies ist die Position (-1/16/255)*

RequestSync

Synchronisiere Controller mit PC:

Dieser Befehl synchronisiert den Controller mit dem PC. Der Controller akzeptiert den Befehl, schickt aber keine Antwort.
Der Zweck des <SYN> Befehl ist es die Kommunikation mit der RS232-Schnittstelle wieder zu erlangen. Bei Verlust der Synchronisation, senden Sie einfach eine grössere Zahl <SYN> Befehle zum Controller, gefolgt von einem RequestMachineState Kommando.

RequestStart

Lese die Start Taste:

Dieser Befehl fragt die optionale Start Taste des Controllers ab.

Der PC sendet ein <LF> an den Controller, dieser antwortet mit
<ACK> oder <NAK>
Beispiel:

<LF>
<NAK> *die Start Taste war nicht gedrückt*

RequestMachineState

Lese den Status des Controllers:

Dieser Befehl gibt den Status des Controllers zurück. (Word = 16bits)
In dem Status hat jedes Bit eine spezielle Information

tmDriveOn	Bit 0	Motoren stehen unter Spannung
tmSpindleOn	Bit 1	Spindle Motor ist eingeschaltet
tmBusy	Bit 2	Achsen sind in Bewegung
tmCOMfail	Bit 3	Comport Fehler (nicht implementiert)
tmTimeOut	Bit 4	interner Fehler (nicht implementiert)
tmParmError	Bit 5	Parameter Fehler (nicht implementiert)
tmOverTemp	Bit 6	interne Temperatur über 50grad
tmInvSetup	Bit 7	das Setup(StartSpeed, MaxSpeed, Acceleration) ist noch nicht komplett
tmLimSwX	Bit 8	ein Endschalter der X-Achse ist aktiv
tmLimSwY	Bit 9	ein Endschalter der Y-Achse ist aktiv
tmLimSwZ	Bit 10	ein Endschalter der Z-Achse ist aktiv
tmRefSwX	Bit 11	der Referenzschalter der X-Achse ist aktiv
tmRefSwY	Bit 12	der Referenzschalter der Y-Achse ist aktiv
tmRefSwZ	Bit 13	der Referenzschalter der Z-Achse ist aktiv
tmPanicStop	Bit 14	der Panic (Not-Aus) Schalter des Controllers ist aktiv

Das Bit 15 hat zur Zeit keine Bedeutung und ist Null.
Der PC sendet ein <ENQ> an den Controller, dieser antwortet mit
<ENQ>BBBB#
Beispiel:

<ENQ>
<ENQ>0080# *dies bedeutet dass das Setup nicht komplett ist (tmInvSetup = true)*

RequestAxeState

Lese den Status der Achsen:

Dieser Befehl gibt den Status der Achsen zurück. (Byte = 8bits)
In dem Status hat jedes Bit eine spezielle Information

taXMove	Bit 0	X-Achse ist im FastMove oder Feed Mode
taXJog	Bit 1	X-Achse ist im Jog Mode
taYMove	Bit 2	Y-Achse ist im FastMove oder Feed Mode
taYJog	Bit 3	Y-Achse ist im Jog Mode
taZMove	Bit 4	Z-Achse ist im FastMove oder Feed Mode
taZJog	Bit 5	Z-Achse ist im Jog Mode
taLimSw	Bit 6	Ein Endschalter ist aktiv
taRefSw	Bit 7	Ein Referenz Schalter ist aktiv

Der PC sendet ein <HT> an den Controller, dieser antwortet mit
<HT>BB#
Beispiel:

<HT>
<HT>05# *dies bedeutet dass die X und Y Achse in Bewegung sind Move oder Feed*

Im JogMode, wenn z.B. nur die X-Achse zur Zeit läuft sieht die Antwort so aus:
<HT>02# *dies bedeutet dass die X Achse in Bewegung ist, JOG Mode*

Diese Abfrage muss vor jedem Move oder Feed Kommando gemacht werden, um sicher zu stellen, dass die Maschine im Stillstand ist. Ausnahme ist das SetFeedXYZ Kommando. Hier muss nur vor dem ersten Kommando die Abfrage erfolgen, dann können beliebig viele SetFeedXYZ Kommandos folgen, bis zum Senden des EndOfFeed Kommandos.

Ist das Bit „taLimSw“ aktiv, so können keine weiteren Move Befehle mehr ausgeführt werden, ausgenommen die Jog-Befehle. Wenn eine Fahrt einen Endschalter erreicht hat, dann wird diese auch immer sofort abgebrochen, ausgenommen wiederum die Jog Bewegungen.

Dazu gilt auch generell für fast alle Kommandos, die ein <ACK> als Resultat erwarten, dass diese mit einem <NAK> zurückkehren, wenn ein Endschalter aktiv ist oder sogar der Panic bzw. Not-Aus Schalter betätigt ist.

Im Falle eines <NAK> ist es am besten man liest beide Status Informationen mit „RequestAxeState“ und „ReguestMachineState“ aus und trifft dann die entsprechenden Entscheidungen.

Setup Befehle

SetSwitchPolarity

Sende die End- oder Referenzschalter Polarität der Achsen:

Dieser Befehl schreibt die Polarität der End- oder Referenz Schalter in den Controller. Die Werte sind 8bit Bytes.
Um die Polarität der ***Limit*** Schalter zu schreiben, senden Sie ein <SO> plus die Polarität für die einzelnen Achsen z.B.:

<SO>FF, FF;00.

Der Controller antwortet mit einem <ACK> oder <NAK>.
Beispiel:

<SO>00,00;00.	*Limit Schalter sind Null-schaltend bzw. low-active*
<ACK>	*Kommando akzeptiert*

Um die Polarität der ***Referenz*** Schalter zu schreiben, senden Sie ein <DLE> plus die Polarität für die einzelnen Achsen z.B.:

<DLE>FF, FF;00.

Der Controller antwortet mit einem <ACK> oder <NAK>.
Beispiel:

<DLE>FF,FF;FF.	*Referenz Schalter sind 1-schaltend bzw. high-active*
<ACK>	*Kommando akzeptiert*

SetStartSpeed

Sende die Startgeschwindigkeit der Achsen:

Dieser Befehl schreibt die Startgeschwindigkeit (steps/sec) für die Move Befehle (FastMove, ReferenceMove, VelocityMove) in den Controller. Die Werte sind 16bit Words ohne Vorzeichen.
Um die Geschwindigkeitswerte zu schreiben, senden Sie ein <ESC> und ein „S“ plus die Werte für die einzelnen Achsen z.B.:

<ESC>S00FF,00FF;00FF.

Der Controller antwortet mit einem <ACK> oder <NAK>.
Beispiel:

<ESC> S0064,0064;0064.	*Alle Achsen bekommen die Startrate 100steps/sec*
<ACK>	*Kommando akzeptiert*

SetMaxSpeed

Sende die Maximalgeschwindigkeit der Achsen:

Dieser Befehl schreibt die Maximal Geschwindigkeit (steps/sec) für die Move Befehle (FastMove, ReferenceMove) in den Controller. Die Werte sind 16bit Words ohne Vorzeichen.
Um die Geschwindigkeitswerte zu schreiben, senden Sie ein <ESC> und ein „M“ plus die Werte für die einzelnen Achsen z.B.:

<ESC>M00FF,00FF;00FF.

Der Controller antwortet mit einem <ACK> oder <NAK>.
Beispiel:

<ESC> M1388,1388;1388.	*Alle Achsen bekommen die Maxrate 5000steps/sec*
<ACK>	*Kommando akzeptiert*

SetAcceleration

Sende die Beschleunigungswerte der Achsen:

Dieser Befehl schreibt die Beschleunigungswerte für die Rampen im Positioniermodus (FastMove) in den Controller. Die Werte sind 16bit Words ohne Vorzeichen. Je größer die Werte desto steiler ist die Rampe. Sinnvolle Werte liegen zwischen 500 und 2000.
Um die Beschleunigungswerte zu schreiben, senden Sie ein <ESC> und ein „A“ plus die Werte für die einzelnen Achsen z.B.:

<ESC>A00FF,00FF;00FF.

Die Controller antwortet mit einem <ACK> oder <NAK>.
Beispiel:

<ESC> A03E8,03E8;03E8. *Alle Achsen bekommen die Beschleunigung 1000*
<ACK> *Kommando akzeptiert*

SetFeedSpeed

Sende die Arbeitsgeschwindigkeit im Feedmode der Achsen:

Dieser Befehl schreibt die Arbeitsgeschwindigkeit im Feedmode in den Controller. Die Werte sind 16bit Words ohne Vorzeichen. Bitte beachten Sie, dass die Arbeitsgeschwindigkeit unterhalb der Startgeschwindigkeit des Motors sein muss, da hier keine Rampe gefahren wird.
Um die Arbeitsgeschwindigkeit zu schreiben, senden Sie ein <FS> und die Werte für die einzelnen Achsen z.B.:

<FS>00FF,00FF;00FF.

Der Controller antwortet mit einem <ACK> oder <NAK>.
Beispiel:

<FS> 0064,0064;0064. *Alle Achsen bekommen die Arbeitsrate 100steps/sec*
<ACK> *Kommando akzeptiert*

Dieser Parameter bestimmt die Steprate für die Arbeits (Feed) Befehle

SetFeedSpeedX
SetFeedSpeedY
SetFeedSpeedZ
SetFeedSpeedXYZ

Hilfs Funktionen

Die folgenden 3 Kommandos sind zusätzliche Funktionen die eine bestimmte weitere Hardware steuern und damit nicht in jedem Fall für alle Maschinen relevant sind.

SetDrivePower

Sende den Motorstrom für alle drei Achsen:

Der Controller hat in seinen Steckverbindern für die Motor Endstufen zwei binär kodierte Bits (REF1/REF2) die den Strom der Endstufe einstellen können. Es sind 4 Stufen möglich:

1. AUS
2. 33%
3. 66%
4. 100%

Um den Motorstrom zu schreiben, senden Sie ein <STX> und einen Wert zwischen 0 und 100 in einem Hex-Byte:

<STX>FF

Der Controller antwortet mit einem <ACK> oder <NAK>. Der Übergabe Wert gilt für alle Achsen gleichzeitig. Der Parameter wird im Controller in die 4 Stufen aufgeschlüsselt:
<25 = off, <50 = Stufe1, <75 = Stufe2, >75 = stufe3
Beispiel:

<STX> 64	*Alle Achsen bekommen 100% Power*
<ACK>	*Kommando akzeptiert*

SetSpindleRPM

Sende die Drehzahl der Arbeitsspindel:

Der Controller kann keine Spindel Drehzahl direkt einstellen, sondern diese nur über das Relais RS1 ein oder ausschalten. Trotzdem muss hierzu ein 32bit unsigned word Parameter übergeben werden um auch zukünftige Erweiterungen abdecken zu können.

Um die Spindel Drehzahl zu schreiben, senden Sie ein <SOH> und einen Wert zwischen 0 und 1000 in einem long Hex-Word, z.B.:

<SOH>FFFFFFFF#

Der Controller antwortet mit einem <ACK> oder <NAK>. Ein Wert mit 0 schaltet das Relais aus, ein Wert > 0 schaltet das Relais ein.
Beispiel:

<SOH>000003E8#	*Spindel Einschalten*
<ACK>	*Kommando akzeptiert*

SetCoolOnOff

Schalte Kühlung ein oder aus:

Der Controller kann über das Relais RS2 eine Kühlmittel Pumpe ein oder ausschalten. Hier sind auch andere Peripherie Einheiten denkbar.
Um das Relais zu schalten, senden Sie ein <SI> und ein true oder false einem Hex-Byte:

<SI>FF

Der Controller antwortet mit einem <ACK> oder <NAK>. Ein true schaltet das Relais ein
Beispiel:

<SI>00	*Das Relais wird ausgechaltet*
<ACK>	*Kommando akzeptiert*

Stop Befehle

ForcePanicStop

Der „Panic Stop“ Befehl:

Der „Panic Stop“ Befehl dient zum Schnellstop aller Achsen bei Gefahr. Bei diesem Befehl wird keine Stopprampe gefahren. Beim Wiederanlauf muss das System neu initialisiert werden, da durch die fehlende Stoprampe Schritte verloren gehen.
Um den Befehl auszuführen senden Sie ein <BEL> an den Controller. Dieser antwortet mit einem <ACK>.
Beispiel:

<BEL>	*Alle Achsen sofort stoppen*
<ACK>	*Kommando akzeptiert*

ForceStopMove

Stoppe Achsenbewegung:

Wenn sich der Controller in einem Movemodus befindet, können hier alle Achsen gestoppt werden. Im Gegensatz zu dem Befehl „PanicStop“ wird hier eine Stoprampe gefahren d.h. es gehen keine Schritte verloren.
Um das System an zu halten, senden Sie ein <XOFF> an den Controller. Dieser antwortet mit einem <ACK>.
Beispiel:

<XOFF>	*Alle Achsen mittels RampDown sofort stoppen*
<ACK>	*Kommando akzeptiert*

Da der Controller alle Achsen kontrolliert mit einer Bremsrampe stoppt, ist diesem die noch verbleibended Schritte des vorausgegangenen FastMove Kommandos bekannt. Falls der Host diese Werte erfahren will, muss er solange den Controller pollen, bis alle Achsen wirklich zum Stillstand gekommen sind (RequestAxeState). Dann kann er mit „RequestRelPosition“ die jeweils noch verbleibenden Schritte abfragen.

Wurde der Stop Befehl während einer <DC2> Aktion „SetFeedStepsXYZ“ geschickt, enthält der Controller Buffer mit grosser Wahrscheinlichkeit noch Einzelschritte. Der Buffer kann jetzt mit folgendem Kommando geleert werden.

ForceFlushBuffer

Leere Buffer:

Wenn sich der Controller im XYZ Feed Modus befindet <DC2>, kann nach einem Stop Befehl der Buffer geleert werden. Um den Buffer zu leeren, senden Sie ein <GS> an den Controller. Dieser antwortet mit einem <ACK>.
Beispiel:

<GS>	*Buffer leeren*
<ACK>	*Kommando akzeptiert*

Befehle zur Positionierung

Zur Beachtung:
Alle Positionier Befehle bedingen als erstes ein korrektes Setup des Controllers und der Maschine. Dann muss, bevor ein solches Kommando abgesetzt werden kann, der Stillstand der Maschine sichergestellt sein. Deshalb muss vor jedem Positionier Kommando solange mit „RequestAxeState“ gepollt werden, bis der letzte Befehl komplett ausgeführt ist. Ausserdem darf das LimitSwitch Flag nicht aktiv sein (Endschalter).

Kehrt ein MOVE Befehl mit einem <NAK> zurück, dann ist entweder die Maschine auf einen Endschalter aufgelaufen oder der Operator hat den Not-Aus Knopf gedrückt (PanicState).
In Fall eines <NAK> müssen deshalb beide Status Informationen vom Controller gelesen werden um eine genaue Kenntniss von der Abweisung zu erhalten und dann entsprechend weiter zu verfahren.

Es werden keine absolute Koordinaten angefahren sondern immer nur relative. Die übergebenen Parameter bezeichnen die Anzahl der auszuführenden Schritte mit Vorzeichen. Damit wird immer eine Koordinate relativ zu momentanen Position angegeben.

SetFastMoveXY

Positioniere die X, Y-Achsen:

Dieser Befehl positioniert die Motoren der X und Y Achse. Senden Sie ein <FF> plus die Positionsdaten an den Controller. Die Werte sind 32bit Integer und beinhalten die Anzahl der Schritte für eine Richtung. Der Controller antwortet sofort mit einem <ACK/NAK>. Weitere Infos kommen nicht vom Controller, deshalb muss vor einem weiteren Kommando solange mit „RequestAxeState“ gepollt werden, bis der Befehl komplett ausgeführt wurde.
Um eine Position anzufahren senden Sie ein <FF> und die Positionen für die einzelnen Achsen z.B.:

<FF> 000003E8,FFFFFC18;	*X-Achse +1000steps,Y-Achse-1000 steps*
<ACK>	*Kommando akzeptiert*

SetFastMoveXYZ

Positioniere die X, Y, Z-Achsen:

Dieser Befehl positioniert die Motoren der X, Y und Z Achse. Senden Sie ein <EOT> plus die Positionsdaten an den Controller. Die Werte sind 32bit Integer und beinhalten die Anzahl der Schritte für eine Richtung. Der Controller antwortet sofort mit einem <ACK/NAK>. Weitere Infos kommen nicht vom Controller, deshalb muss vor einem weiteren Kommando solange mit „RequestAxeState“ gepollt werden, bis der Befehl komplett ausgeführt wurde.
Um eine Position anzufahren senden Sie ein <EOT> und die Positionen für die einzelnen Achsen z.B.:

<EOT> 000003E8,FFFFFC18;00000010.	*X-Achse +1000steps, Y-Achse -1000 step Z-Achse +16 steps.*
<ACK>	*Kommando akzeptiert*

SetFastMoveZ

Positioniere Z-Achse:

Dieser Befehl positioniert den Motore der Z Achse. Senden Sie ein <BS> plus die Positionsdaten an den Controller. Die Werte sind 32bit Integer und beinhalten die Anzahl der Schritte für eine Richtung. Der Controller antwortet sofort mit einem <ACK/NAK>. Weitere Infos kommen nicht vom Controller, deshalb muss vor einem weiteren Kommando solange mit „RequestAxeState“ gepollt werden, bis der Befehl komplett ausgeführt wurde.
Um eine Position anzufahren senden Sie ein <BS> und die Position für die Z Achse.
Beispiel:

<BS> 000003E8.	*Z-Achse +1000steps*
<ACK>	*Kommando akzeptiert*

Befehle im Arbeitsmodus

Zur Beachtung:
Alle Positionier Befehle, auch die Feed Typen, bedingen als erstes ein korrektes Setup des Controllers und der Maschine. Dann muss bevor ein solches Kommando abgesetzt werden kann, der Stillstand der Maschine sichergestellt sein. Deshalb muss vor jedem Feed Kommando solange mit „RequestAxeState“ gepollt werden, bis der letzte Befehl komplett ausgeführt ist.

Es werden keine absolute Koordinaten angefahren sondern immer nur relative. Die übergebenen Parameter bezeichnen die Anzahl der auszuführenden Schritte mit Vorzeichen. Damit wird immer eine Koordinate relativ zu momentanen Position angegeben.

SetClearPosition

Starte Arbeitmodus (Feedmode):

Dieser Befehl setzt die internen Zähler zurück. Dieser Befehl sollte vor jedem Befehl im Feedmodus aufgerufen werden.
Durch ein <RS> wird der Befehl ausgeführt, die Controller antwortet mit <ACK>
Beispiel:

<RS>	*clear counter, starte Arbeits (Feed) Mode*
<ACK>	*Kommando akzeptiert*

SetFeedStepsX

Fahre zu Position X (Feedmode):

Dieser Befehl gibt die anzufahrende Endposition vor, d.h. der Motor fährt mit der FeedRateX zu einer vorgegebenen X-Koordinate. Die Y- und Z-Achse bleiben davon unberührt.
Um diesen Befehl aus zu führen senden Sie ein <EM> und die Positionsdaten der X-Achse an den Controller. Die Werte sind 32bit Integer und beinhalten die Anzahl der Schritte für eine Richtung. Der Controller antwortet sofort mit einem <ACK>.

Der Host darf bis zum Erreichen der Ziel Position keine weitere Kommandos schicken, sonder muss zyklisch den Achsen Status und evtl. den Maschinen Status Pollen.

<EM> 000003E8,	*X-Achse +1000steps*
<ACK>	*Kommando akzeptiert*

Position wird angefahren.

SetFeedStepY

Fahre zu Position Y (Feedmode):

Dieser Befehl gibt die an zu fahrende Endposition vor d.h. der Motor fährt mit der FeedRateY zu einer vorgegebenen Y-Koordinate. Die X- und Z-Achse bleiben davon unberührt.
Um diesen Befehl aus zu führen senden Sie ein <SUB> und die Positionsdaten der Y-Achse an den Controller. Die Werte sind 32bit Integer und beinhalten die Anzahl der Schritte für eine Richtung. Der Controller antwortet sofort mit einem <ACK>.

Der Host darf bis zum Erreichen der Ziel Position keine weitere Kommandos schicken, sonder muss zyklisch den Achsen Status und evtl. den Maschinen Status Pollen.

<SUB> 000003E8;	*Y-Achse +1000steps*
<ACK>	*Kommando akzeptiert*

Position wird angefahren.

SetFeedStepZ

Fahre zu Position Z (Feedmode):

Dieser Befehl gibt die anzufahrende Endposition vor d.h. der Motor fährt mit der FeedRateZ zu einer vorgegebenen Z-Koordinate. Die X- und Y-Achse bleiben davon unberührt.
Um diesen Befehl aus zu führen senden Sie ein <VT> und die Positionsdaten der Z-Achse an den Controller. Die Werte sind 32bit Integer und beinhalten die Anzahl der Schritte für eine Richtung. Der Controller antwortet sofort mit einem <ACK>.

Der Host darf bis zum Erreichen der Ziel Position keine weitere Kommandos schicken, sonder muss zyklisch den Achsen Status und evtl. den Maschinen Status Pollen.

<VT> 000003E8.	*Z-Achse +1000steps*
<ACK>	*Kommando akzeptiert*

Position wird angefahren.

SetFeedStepXYZ

Einzelschritt um die X/Y/Z-Achse (Feedmode):

Dieser Befehl ist das eigentliche Arbeitstier denn bei den o.a. Feed Befehlen werden immer nur lineare Strecken gefahren. Geometrische Formen waren bis jetzt nicht möglich. Das Kommando <DC2> ist der Einelschritt Befehl. Mit ihm werden solange einzelne Daten in den Buffer des Controllers geschreiben, bis die gewählte Figur abgefahren ist und der Einzelschritt Modus mit dem Kommando <CR> „EndOfFeed“ beendet wird.

Ein Schritt setzt sich immer aus zwei Bytes zusammen. Das erste ist das Kommando <DC2> und das zweite sind die x-y-z Steps. Die einzelnen Achsen werden Bitweise gesteuert d.h. für die X/Y/Z-Achsen werden immer 2Bit geschrieben,
Drehrichtung und der eigentliche Schritt.

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
False	***StepZ***	***StepY***	***StepX***	***False***	***DirZ***	***DirY***	***DirX***

Das Bit 7 und das Bit 3 müssen immer False sein. Die StepX, StepY und StepZ bits zeigen an, wenn sie gesetzt sind, dass die jeweilige Achse einen Schritt machen muss. Die zugehörige Richtung wird in den DirX, DirY und DirZ bits angegegeben.

Ist ein Stepbit Null, so führt diese Achse mit diesem Befehl keine Schritte aus. Ist ein Bit = 1 dann wird ein Schritt ausgeführt und zwar in der Richtung, die das Dir-Bit angibt.
Ist das Dir-Bit gesetzt so wird ein Step in die negative Richtung ausgeführt, ist das Bit = 0 so wird ein Schritt in die positive Richtung ausgeführt.

Das ganze Datenbyte besteht deshalb aus insgesamt 3 Vektoren x, y, z, wobei jeder Vektor nur den Wert +1, null oder -1 annehmen kann.

Step	=	0	Dir	=	?	Vektor => 0
Step	=	1	Dir	=	0	Vektor => +1
Step	=	1	Dir	=	1	Vektor => -1

Um z.B. einen Kreis zu fahren, füllt der Host den Buffer des Controller mit hunderten oder sogar tausenden von **<DC2>byte** Kommandos bis der Kreis geschlossen ist. Da der Buffer des Controllers meistens eine komplexe Figur nicht als ganzes aufnehmen kann, stoppt der Controller den Datenfluss des Hosts immer wieder mit <XOFF> und wenn wieder genügend Platz ist, wird die Sperrung mit <XON> wieder aufgehoben. Den Abschluss eines solchen Arbeitsvorgangs muss immer ein „EndOfFeed“ Kommando sein.

Der Host darf bis zum Erreichen der Ziel Position keine weitere Kommandos schicken, sonder muss zyklisch den Achsen Status und evtl. den Maschinen Status Pollen.

Man muss dabei beachten, dass der Controller sofort mit dem ersten <DC2> mit der Fahrt anfängt, und nicht erst wenn der Buffer voll ist oder gar das EndOfFeed vom Host geschickt wird. Das bedeutet, dass der Host möglichst schneller seine Daten schicken soll als die eingestellte Feedrate des Controllers ist um eine ruck- und stotterfreie Fahrt zu gewährleisten. Der Controller leert nämlich seinen XYZ Step Buffer mit der vorgegebenen Feedrate.

Der Controller gibt grundsätzlich keine Antwort auf diese Befehle, auch wenn der Buffer inzwischen leer sein sollte (Ausnahme: XOFF/XON).

Befehle im Jogbetrieb

Zur Beachtung:
Alle Positionier Befehle, auch der Jog Befehl, bedingen als erstes ein korrektes Setup des Controllers und der Maschine. Dann muss, bevor das erste Jog-Kommando abgesetzt werden kann, der Stillstand der Maschine sichergestellt sein. Deshalb muss vor dem ersten Jog-Kommando solange mit „RequestAxeState“ gepollt werden, bis der letzte Befehl komplett ausgeführt ist.

Nach dem Absetzen des ersten Jog-Kommandos braucht der Axen Status nicht mehr abgefragt zu werden, da der Controller jetzt im Jog Modus ist. Allerdings kann ein JOG Befehl mit einem <NAK> zurückkehren, wenn der Not-Aus manuell am Controller betätigt wurde.

Im Gegensatz zu allen anderen Move oder Feed Befehlen werden hier keine Schrittzahlen oder Steps vorgegeben, sondern Jog-Stepraten. Das heist, dass jeder Jog-Befehl nur die aktuell laufende Steprate ändert aber auch gleichzeitig die Motoren laufen lässt. Jeder Wert <> 0 lässt den jeweiligen Motor laufen. Nullwerte stoppen den betreffenden Motor.

Damit ist natürlich auch eiue Gefahr verbunden. Wird die Steprate gegenüber der aktuell laufenden zu stark verändert, kann der Motor ausser Synchronisation kommen was zum extremen Schütteln des Motors führen kann oder gar zum kraftlos vor sich hin summenden Stillstand. Der Host ist also hier selbst verantwortlich, Veränderungen über Beschleunigungs oder Brems Rampen zu erzeugen.

Der Modus wird automatisch wieder verlassen, wenn alle 3 Achsen eine Steprate mit dem Wert 0 erhalten.

SetJogSpeed

Sende Jog Geschwindigkeit:

Dieser Befehl dient in erster Linie zum Einrichten der Maschine, zum Beispiel das Anfahren des Nullpunkts oder der Refrenz Position. Es sind natürlich auch einige andere Anwendungen dafür denkbar.
Mit diesem Befehl <DC4> wird eine Joggeschwindigkeit vorgeben, mit der der Controller die Schrittmotoren antreiben soll, das Vorzeichen bestimmt die Richtung. Ist der Wert <> 0 dann läuft der Motor mit dieser Geschwindigkeit in diese Richtung. Ist der Wert 0 stoppt der Motor. Die Werte sind 16bit Integer mit Vorzeichen.

Um die Geschwindigkeitswerte zu schreiben, senden Sie ein <DC4> plus die Werte für die einzelnen Achsen z.B.:

<DC4>00FF,00FF;00FF.

Der Controller antwortet mit einem <ACK> oder <NAK>.
Beispiel:

<DC4> 0064,0064;FF9C.	*Alle Achsen fahren jetzt mit den Stepraten 100steps/sec X und Y Achse -> positive Richtung, Z-Achse negativ*
<ACK>	*Kommando akzeptiert*

E-LAB 3-Achsen Stepper Controller

Befehle zur einfachen Achsen Bewegung

Zur Beachtung:
Alle Move Befehle bedingen als erstes ein korrektes Setup des Controllers und der Maschine. Dann muss, bevor ein solches Kommando abgesetzt werden kann, der Stillstand der Maschine sichergestellt sein. Deshalb muss vor jedem Move Kommando solange mit RequestAxeState gepollt werden, bis der letzte Befehl komplett ausgeführt ist Ausnahme VelocityMove). Ausserdem darf das LimitSwitch Flag nicht aktiv sein (Endschalter).

Kehrt ein MOVE Befehl mit einem <NAK> zurück, dann ist entweder die Maschine auf einen Endschalter aufgelaufen oder der Operator hat den Not-Aus Knopf gedrückt (PanicState).
In Fall eines <NAK> müssen deshalb beide Status Informationen vom Controller gelesen werden um eine genaue Kenntniss von der Abweisung zu erhalten und dann entsprechend weiter zu verfahren.

Diese Move Befehle haben keine Ziel Koordinaten sondern es wird solange gefahren bis eine Abbruch Bedingung zutrifft. Die zwei Move Kommandos unterscheiden sich in erster Linie durch die Art des Stops.

SetReferenceMoveXYZ

Fahre die X, Y, Z-Achsen zu den Referenz Schaltern:
Der **ReferenceMove** fährt eine Rampe hoch in der gewählten Richtung und behält seine Steprate bei bis entweder ein Stop Befehl vom Host kommt oder der zu dieser Achse gehörende ***Referenz Schalter*** aktiviert wird. Dadurch wird ein Schnellstop eingeleitet.
Die Start-Steprate, Beschleunigung und End-Steprate werden durch die mit den Befehlen ***SetStartSpeed, SetAcceleration, SetMaxSpeed*** eingestellten Parameter bestimmt.

Senden Sie ein <US> plus die Richtungsdaten an den Controller. Die Werte sind 16bit Integer (-1/0/+1) und beinhalten die zu fahrende Richtung. Der Controller antwortet sofort mit einem <ACK/NAK>. Weitere Infos kommen nicht vom Controller, deshalb muss vor einem weiteren Kommando solange mit „RequestAxeState“ gepollt werden, bis der Befehl komplett ausgeführt wurde.
Um einen ReferenceMove zu starten senden Sie ein <US> und die Richtungen für die einzelnen Achsen z.B.:

<US> FFFF,0000;0001.	*X-Achse -DIR, Y-Achse no move, Z-Achse +Dir*
<ACK>	*Kommando akzeptiert*

Note:
Dieser Befehl macht nur Sinn, wenn ein funktionierender Referenz Schalter in der gewählten Richtung zu finden ist und die Schalter Polarität mit ***SetSwitchPolarity*** richtig eingestellt ist. Ist ein Achsen Parameter dieses Befehls “0“ dann erfolgt mit dieser Achse keine Bewegung.

Eine aktivierte Achse läuft solange, bis sie ihren Referenzschalter erreicht hat. Bei mehreren aktivierten Achsen bedeutet dies, dass solange kein neues Kommando abgesetzt werden kann, bis alle Achsen zum Stillstand gekommen sind. Der Stop Befehl bildet jedoch eine Ausnahme davon.

SetVelocityMoveXYZ

Fahre die X, Y, Z-Achsen bis auf Widerruf:

Der **VelocityMove** fährt eine Rampe hoch in der gewählten Richtung und behält seine Steprate bei bis ein Stop Befehl vom Host kommt oder der vorgegebene Wert 0 ist. Der übergebene Parameter bestimmt dabei die zu erreichende Schrittrate. Während dieser Fahrt kann die Steprate mit dem gleichen Befehl kontinuierlich geändert werden. Die Start-Steprate und Beschleunigung werden durch die mit den Befehlen ***SetStartSpeed*** und ***SetAcceleration*** eingestellten Parameter bestimmt.

Senden Sie ein <CAN> plus die Stepraten an den Controller. Die Werte sind 16bit Integer und beinhalten die zu fahrende Richtung im Vorzeichen. Der Controller antwortet sofort mit einem <ACK/NAK>. Weitere Infos kommen nicht vom Controller, deshalb muss vor einem weiteren Kommando solange mit „RequestAxeState“ gepollt werden, bis der Befehl komplett ausgeführt wurde.
Um einen VelocityMove zu starten senden Sie ein <CAN> und die Richtungen/Stepraten für die einzelnen Achsen z.B.:

<CAN> FC18,0000;03e9.	*X-Achse -Dir_1000stp/sec, Y-Achse no move, Z-Achse +Dir_1000stp/sec*
<ACK>	*Kommando akzeptiert*

Note:
Dieser Befehl macht nur Sinn, wenn er auch rechtzeitig manuell wieder abgebrochen wird. Ist ein Achsen Parameter dieses Befehls “0“ dann erfolgt mit dieser Achse keine Bewegung.

Eine aktivierte Achse läuft solange, bis sie ihren Endschalter erreicht hat oder ein allgemeiner Stop Befehl abgesetzt wurde. Bei mehreren aktivierten Achsen bedeutet dies, dass solange kein neues Kommando abgesetzt werden kann, bis alle Achsen zum Stillstand gekommen sind. Der Stop Befehl und der VelocityMove Befehl selbst bilden die Ausnahme davon. Jeder Stop einer Achse führt zu einer Bremsrampe.

Der erste VelocityMove Befehl bedingt wie bei allen anderen Move Befehlen, das die Achsen im Stillstand sind. Nach dem Absetzen des ersten VelocityMove kann dieser Befehl mit veränderten Parametern beliebig wiederholt werden, allerdings wird hier dann das Vorzeichen nicht mehr beachtet. Eine Drehrichtungs Umkehr ist bei laufendem Motor hier nicht möglich. Dazu muss die jeweilige Achse zuerst die Velocity “0“ erhalten, dann muss gewartet werden, bis diese Achse zum Stillstand gekommen ist (pollen des Achsen Status) und dann kann die Achse neu gestartet werden mit einem geänderten Vorzeichen und neuen Velocity Werten.

Achtung:
Velocity Werte kleiner als die eingestellte StartStopFrequenz werden als “0“ interpretiert und führen zum Stop der Achse mit Bremsrampe.

Zusammenfassung der Befehle

TBD.

Rot bedeutet PC -> Controller.
Blau bedeutet Controller -> PC.

E-LAB 3-Achsen Stepper Controller

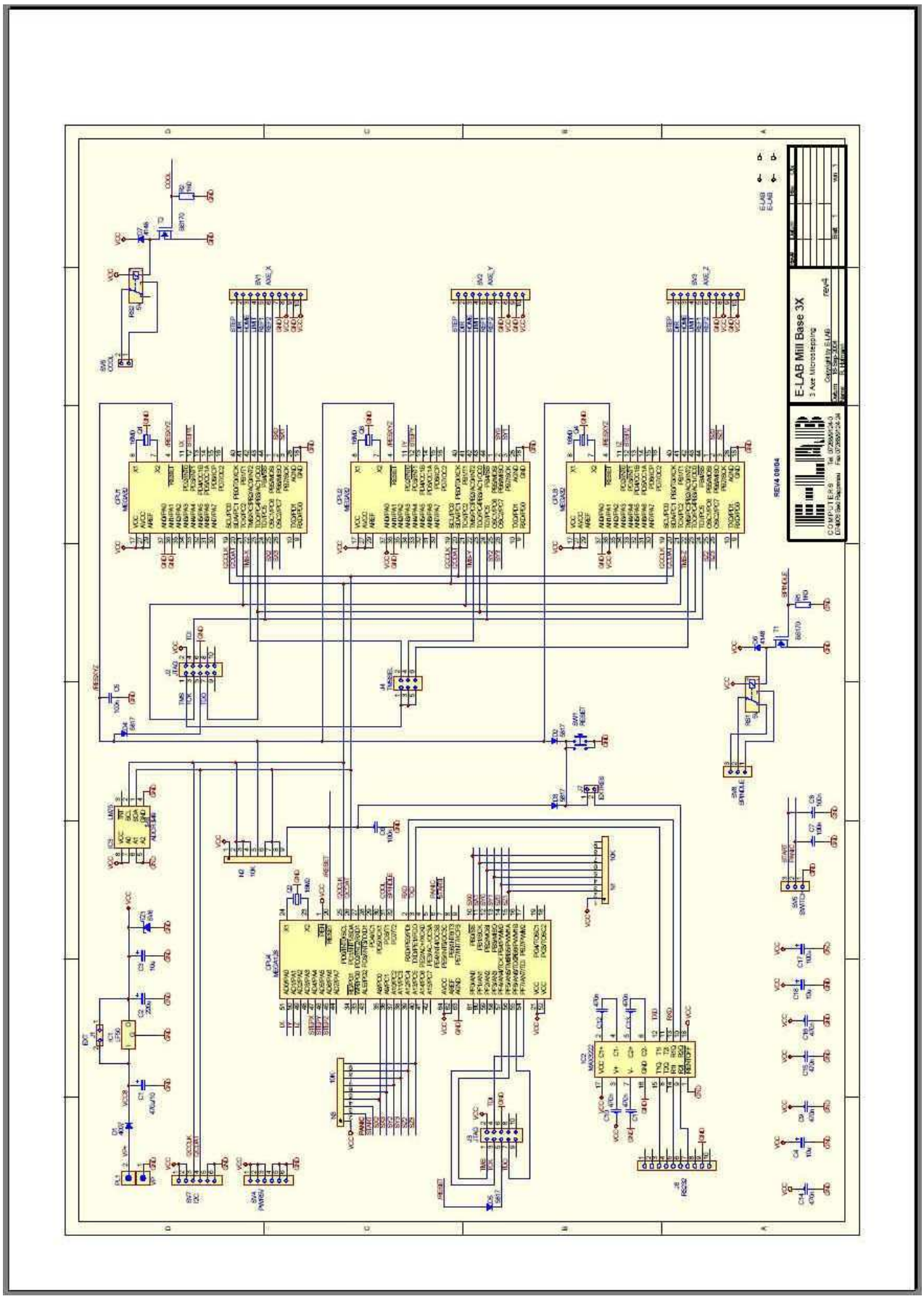
Steuerzeichen Tabelle und Kommando Liste

Mnemonic	Character	Hex	Controller CMD	Note
<SOH>	#1	$01	Set Spindle RPM	2
<STX>	#2	$02	Set Drive Power	2
<ETX>	#3	$03	Request relative Position	
<EOT>	#4	$04	Set FastMove XYZ	1, 2
<ENQ>	#5	$05	Request Machine State	
<ACK>	#6	$06	Request Absolute Position	
<BEL>	#7	$07	Set Panic Stop	
<BS>	#8	$08	Set FastMove Z	1, 2
<HT>	#9	$09	Request Axe State	
<LF>	#10	$0A	Request Start Info	1, 2
<VT>	#11	$0B	Set FeedSteps Z	1, 2
<FF>	#12	$0C	Set FastMove XY	1, 2
<CR>	#13	$0D	End Of Feed	1, 2
<SO>	#14	$0E	Set Limit Switch Polarity	
<SI>	#15	$0F	Set Cooling ON/OFF	2
<DLE>	#16	$10	Set Reference Switch Polarity	
<XON>	#17	$11		
<DC2>	#18	$12	Set FeedSteps XYZ	1, 2
<XOFF>	#19	$13	Force Stop Move	
<DC4>	#20	$14	Set Jog Speed	2
<NAK>	#21	$15		
<SYN>	#22	$16	Request Synchronisation	
<ETB>	#23	$17	Request Temperatur	
<CAN>	#24	$18	Set Velocity Move XYZ	1, 2
<EM>	#25	$19	Set FeedSteps X	1, 2
<SUB>	#26	$1A	Set FeedSteps Y	1, 2
<ESC>	#27	$1B	Set Rampe: Start, Max, Acceleration	
<FS>	#28	$1C	Set Standard Feed Rate	
<GS>	#29	$1D	Force Flush Buffer	
<RS>	#30	$1E	Set Clear Position	1, 2
<US>	#31	$1F	Set Reference Move XYZ	1, 2

Notes:

1. Dieses Kommando kehrt mit einem <NAK> zurück, wenn ein Endschalter aktiv ist.
2. Dieses Kommando kehrt mit einem <NAK> zurück, wenn der Not-Aus aktiv ist

Schaltplan



Bestückungsplan