S I X P A C K / Q U A D P A C K

Manual Version: 3.01
1. Oktober 2004

Sternstraße 67
D - 20357 Hamburg, Germany
Phone +49-40-51 48 06 - 0
FAX: +49-40-51 48 06 - 60
http://www.trinamic.com

SIXpack 6 Kanal 800mA
QUADpack 4 Kanal 1500mA

TRINAMIC Motion Control GmbH & Co KG
Alle Rechte vorbehalten. Kein Teil dieses Handbuches darf in irgendeiner Form (Fotokopie, Druck, Mikrofilm oder einem anderen Verfahren) ohne ausdrückliche Genehmigung der Trinamic Motion Control GmbH & Co KG reproduziert werden oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.
Der Inhalt dieses Handbuches wurde mit größter Sorgfalt erarbeitet und geprüft. TRINAMIC übernimmt jedoch keine Verantwortung für Schäden, die aus Fehlern in der Dokumentation resultieren könnten. Insbesondere Beschreibungen und technische Daten sind keine zugesicherten Eigenschaften im rechtlichen Sinne.
TRINAMIC hat das Recht, Änderungen am beschriebenen Produkt oder an der Dokumentation ohne vorherige Ankündigung vorzunehmen, wenn diese aus Gründen der Zuverlässigkeit oder Qualitätssicherung vorgenommen werden oder dem technischen Fortschritt dienen.

Sternstraße 67
D – 20357 Hamburg, Germany
Phone +49-40-51 48 06 - 0
FAX: +49-40-51 48 06 - 60
http://www.trinamic.com

Inhaltsverzeichnis:

1 Einleitung

1.1 Kurzbeschreibung

Bei dem QUAD/SIXpack (im folgenden PACK) handelt es sich um ein Produkt zum Betrieb von vier bzw. sechs bipolaren Schrittmotoren. Die von den integrierten Treiber-ICs gelieferten Spulenströme betragen 1500 mA beim QUAD- und 800 mA bei SIXpack.
Das robuste Gehäuse, die geprüfte elektromagnetische Verträglichkeit und das einfache Bedienungskonzept machen das PACK zur idealen Lösung für die Ansteuerung von Schrittmotoren in industriellen Umgebungen.
Das PACK wird über eine serielle Schnittstelle (RS232/485, CAN) an einen Host angeschlossen. Der Host parametrisiert das PACK und veranlasst Bewegungen der Motoren. Bei den Befehlen handelt es sich um Befehlsworte mit einer festen Länge von 9 Byte. Die Befehlsbytes können entweder manuell zusammengestellt oder mit Hilfe der von TRINAMIC zur Verfügung gestellten Windows -DLL generiert werden.

1.2 Was muss ich über meine Anwendung wissen

Bevor das PACK eingesetzt wird, sollten die Anforderungen der Anwendung an die Schrittmotoren und die Steuereinheit klar definiert sein.
Zunächst stellt sich die Frage nach den richtigen Motoren. Am PACK lassen sich grundsätzlich nur Zweiphasen-Schrittmotoren betreiben. Die erzielbare Spitzenleistung ergibt sich aus der Betriebsspannung des PACKs und den von den Treiber-ICs zur Verfügung gestellten Spulenströmen (max. 800mA beim Six- bzw. 1500mA beim QUADpack).
Direkt nach dem Einschalten erkennt das PACK die aktuellen Positionen der Motoren nicht. Hierzu muss erst eine Referenzfahrt durchgeführt werden. Eine grobe kontinuierliche Positionskontrolle ist auch über einen Drehwiderstand möglich. Bei manchen Anwendungen reicht auch ein mechanischer Nullpunkt ohne weitere Beschaltung. Die Möglichkeit einen Encoder anzuschließen, bietet das PACK nicht.
Zur Ansteuerung des PACKs stehen diverse Schnittstellen zur Verfügung (RS232, RS485, CANBus). Ob die Ablaufsteuerung nun auf einem Desktop-PC, der Bedieneinheit einer Maschine oder einer kleinen Schaltung mit Mikroprozessor läuft, ist für das PACK nicht relevant. Ein Standalone-Betrieb ist mit dem PACK allerdings nicht möglich.

1.2.1 Wahl der Motoren

Generell muss ein Motor ausgewählt werden, dessen Drehmoment für die maximale Spitzenlast an der Mechanik ausreicht. Zudem sollten Motoren mit möglichst geringer Induktivität, d.h. auch geringem Spulenwiderstand bevorzugt werden. Das garantiert eine hohe Laufruhe und maximal mögliche Drehzahlen. Mit dem Spulenwiderstand sinkt jedoch das Drehmoment. Daher sollte ein Motor mit der niedrigstmöglichen Induktivität gewählt werden, der bei einem Spulenstrom von ca. 600 mA bzw. 1000 mA das erforderliche Drehmoment erbringt. Eine möglichst hohe Betriebsspannung des PACKs erlaubt ebenfalls hohe Drehzahlen. Bei höherem Spulenwiderstand oder zu geringer Betriebsspannung steigt das Tastverhältnis der Choppertreiber an. Sobald dieses über 50% liegt, kann eine zirpende Geräuschbildung in den Spulen auftreten.

1.2.2 Was das PACK nicht kann

Standalone-Betrieb: Das PACK ist ohne einen Host, der über eine der seriellen Schnittstellen das PACK initialisiert und die Bewegungen der Motoren steuert nicht einzusetzen.
Closed-Loop-Betrieb: Das PACK verfügt über keine Encodereingänge und bietet auch keine andere Möglichkeit einen permanenten Soll-Ist-Abgleich die Motorposition betreffend zu realisieren. Es ist jedoch möglich den Wert des internen Positionszählers und den Status des Referenzschalters vergleichen zu lassen. Ergibt dieser Vergleich einen ungültigen Wert, kann eine automatische Referenzfahrt veranlasst werden.
Betrieb von externen Endstufen: Es gibt keine Möglichkeit externe Endstufen an das PACK anzuschließen oder anderweitig den maximalen Motorstrom zu erhöhen. Ist dies erforderlich, gibt es die Möglichkeit auf Basis des TRINAMIC-Moduls „MACMO406“ eine eigene Baugruppe mit frei wählbaren Ausgängen und Treibern aufzubauen. Der Befehlssatz ist zum PACK voll kompatibel.

1.3 Technische Daten

Fahrrampen (Profil):	automatische 3-Phasen-Rampen (±32 Bit Positionsauflösung). Für jeden Kanal programmierbare Parameter für Maximalfrequenz und Beschleunigung sowie Startfrequenz, auch benutzerdefinierte Rampen.
Schrittfrequenz:	je Fahrkurve 9 Bit Dynamik mit Vollschrittfrequenzen zwischen 0,3 Hz und 12,5 kHz.
Schritttyp:	16-fach Mikroschritt mit programmierbarer Kennlinie oder Sinustabelle.
Stromkontrolle:	beschleunigungsabhängige Motorbestromung; Standby-Stromabsenkung
Schnittstellen:	RS232, RS485 oder CAN. Über CAN Steuerung von Peripherie über RS232
Protokoll:	Properitäres Protokoll über 9-Byte-Befehle.
I/O-Leitungen:	Je Motor ein 10 Bit-Analogeingang für ratiometrische Widerstandsmessungen oder Stopp-Funktion und ein Digitaleingang für Referenzschalter. Separater Analogeingang, digitaler I/O, Ausgang, Ready Ausgang (Open Collector).
Stromversorgung:	15 bis 40 V DC; ca. 6W Aufnahme ohne Last; max. ca. 5A, je nach Motoren
Motorstrom SIXpack:	per Software konfigurierbar ca. 50-800 mA pro Kanal (Spulenspitzenstrom). Konstantstrom (Chopper, ca. 23 kHz), Motortreiber thermisch geschützt.
Motorstr. QUADpack:	per DIP-Schalter und Software konfigurierbar, ca. 100-1500 mA pro Kanal (Spulenspitzenstrom). Konstantstrom (Chopper, ca. 40 kHz), Motortreiber thermisch geschützt
Motoren:	bipolare 2-Phasen Motoren
Motorstecker:	8-polig single-in-line (Motor, Referenzschalter, A/D, 5V Versorgung (15 mA))
Temperaturbereich:	bis 85°C bei reduziertem Strom bzw. Zwangskühlung der Platine
Abmessungen:	Platine: B: 126, T: 180, H: 25 mm, Gehäuse: B: 152, T: 180, H: 36 mm

1.4 Hinweise zum Manual

1.4.1 Zum Umgang mit dem Manual

Das vorliegende Manual lässt sich grob in drei Abschnitte einteilen:

Im 2. Kapitel Grundlegende Funktionen werden alle für eine erste Inbetriebnahme des PACKs notwendigen Schritte beschrieben. Hierzu zählt auch die Benutzung der Windows -Software „SQPack". Zudem wird beschrieben, wie das PACK in seiner Grundkonfiguration schon als Steuereinheit eingesetzt werden kann.

Das 3. Kapitel Erweiterte Funktionen beschreibt eingehend alle notwendigen Schritte, um das PACK für den Großteil aller Anwendungsgebiete zu konfigurieren und zu programmieren. Der Schwerpunkt liegt hierbei auf der Beschreibung von Funktion und Einsatz der diversen Befehle, deren Aufbau ist in der Befehlsreferenz beschrieben.

Im 4. Kapitel Befehlsreferenz sind alle Befehle des PACKs, funktionsabhängig gegliedert, aufgelistet und detailliert beschrieben. Besonderes Augenmerk liegt auf dem Aufbau der Befehle und der Beschreibung der Parameter. Zudem werden auch Funktionalitäten beschrieben, die nur für Spezielanwendungen relevant sind oder deren Funktionsweise so einfach ist, dass sie keiner so eingehenden Beschreibung bedarf.

1.4.2 Namenskonventionen

Im vorliegenden Manual sind zum besseren Verständnis folgende Namenskonventionen definiert:

BefehlsName: Befehle sind durch einen eindeutigen Namen definiert und fett, kursiv geschrieben. Die Namen im Manual sind identisch mit den in der PACK_DLL benutzten. Für die Programmierung über Befehlsdatagramme ist der numerische Bezeichner (CMD xy) zu benutzen, welcher in der Befehlsreferenz angegeben ist.

`VariablenName:` Variablen sind mit dem Font „`Courier New`" geschrieben. Auch sie sind, soweit in dieser vorhanden, mit denen in der PACK_DLL benutzten identisch. In der Befehlsreferenz sind die Typen der Variablen angegeben. Handelt es sich z.B. um einen „32 bit, signed"- Wert, ist er bei Verwendung von Befehlsdatagrammen in vier Bytes im Zweier-Komplement zu übertragen.

FLAGNAME: Flags sind durchgehend groß geschrieben. Sie werden zum aktivieren bzw. deaktivieren von Funktionen benutzt und besitzen die Zustände 0 (nicht gesetzt) und 1 (gesetzt). Es können mehrere Flags in einem Datagrammbyte enthalten sein. Der Wert des Bytes ergibt sich durch verodern (bitweise Addition) der jeweiligen Flagbits nach ihrer Wertigkeit (siehe Kapitel 3.2.1).

$...: Sind numerische Werte im hexadezimalen Format angegeben, ist ihnen das Symbol „$" vorangestellt.

Mit PACK ist immer das Quad- bzw. SIXpack gemeint.

Sternstraße 67
D – 20357 Hamburg, Germany
Phone +49-40-51 48 06 - 0
FAX: +49-40-51 48 06 - 60
http://www.trinamic.com

2 Die Grundlegenden Funktionen

In diesem Teil der Dokumentation wird der einfache Einstieg in die Benutzung des PACKs beschrieben. Neben den notwendigen Arbeitsschritten auf Hardwareseite wird auf die Bedienung mit Hilfe der mitgelieferten Windows -Software eingegangen. Zudem werden grundlegende Anwendungen in der Standartkonfiguration vorgestellt.

2.1 Sicherheitshinweise

Um Schäden am PACK zu vermeiden, sollten Sie folgende Hinweise beachten:

- Falsche Anschlussbelegungen können zur Zerstörung des PACKs führen. Seien Sie deshalb sehr sorgfältig bei der Installation ihres PACKs.
- Auch ein Abziehen der Motorstecker während des Betriebes kann das PACK zerstören.

2.2 Einstellungen am Gerät

2.2.1 Die PACK-Adresse

Bevor das PACK in Betrieb genommen werden kann, müssen einige Einstellungen am Gerät geprüft bzw. vorgenommen werden.
Wichtig ist die Adresse unter der das PACK Befehle über die serielle Schnittstelle empfängt. Sie wird über die Kodierschalter CANHI und CANLO eingestellt und sollte jetzt auf null gesetzt sein. Siehe Abbildung 2-1.

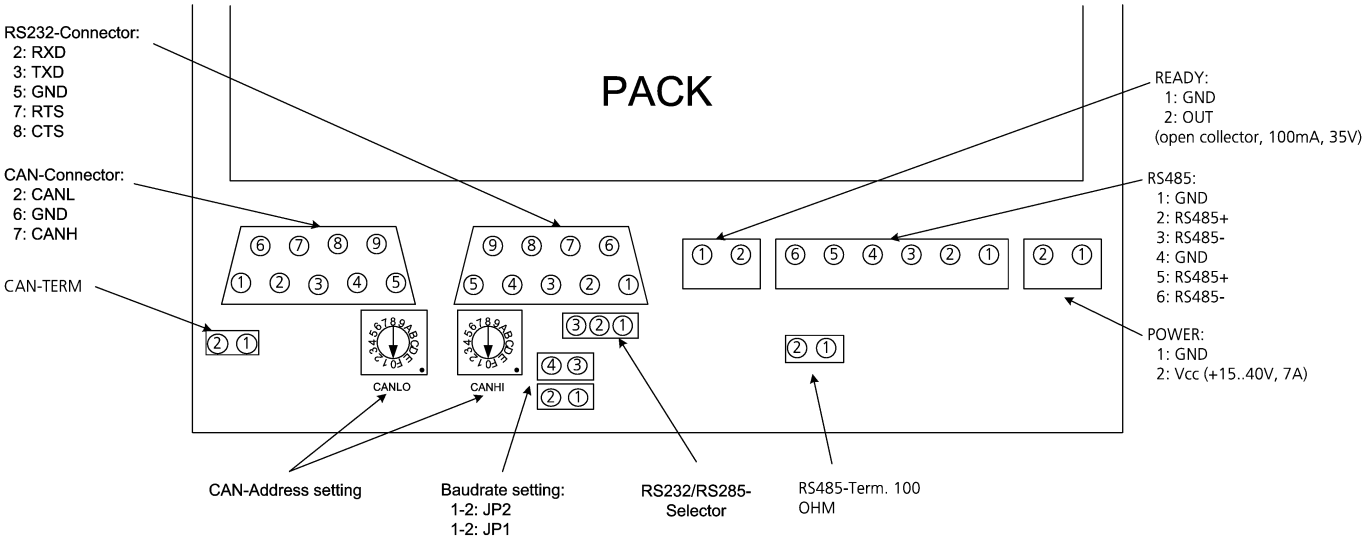


Abbildung 2-1: Jumper- und Steckerbelegung

2.2.2 Auswahl RS232/RS485

Die Ansteuerung des PACKs soll über die RS232-Schnittstelle geschehen. Deshalb muss der Jumper RS232/RS485 auf RS232 eingestellt sein (siehe Abbildung 2-1).
Die Benutzung der RS485-Schnittstelle wird in Kapitel 3.2 beschrieben.

2.2.3 Die Baudrate der seriellen Schnittstellen

Die Baudrate der seriellen Schnittstelle ist über die beiden Jumper JP1 und JP2 einstellbar (siehe Abbildung 2-1). Die jeweilige Jumperkonfiguration ist der nachfolgenden Tabelle zu entnehmen:

JP1	JP2	Baud rate RS232/RS485	Baud rate CAN
x	X	9600	1 Mbit/s (*)
-	X	57600	500 kbit/s (*)
x	-	38400	125 kbit/s (*)
-	-	19200	250 kbit/s (default)(*)

Tabelle 1: Baudrateneinstellung per Jumper

(*) Anmerkung: Das PACK hat einen internen Puffer für 16 CAN-Befehle, die interne Verarbeitung benötigt ca. 2ms je Befehl. Dies limitiert unter Umständen den maximalen Datendurchsatz.

Die Jumperkonfiguration kann über den Befehl *GetInputValues* abgefragt werden. Hierzu sind die in der Variablen `AllInputs` übergebenen Bits zu interpretieren (Bit6 = Jumper1, Bit7 = Jumper2).

Zudem ist es möglich, die Baudrate für die RS232/485 mit Hilfe des Befehls *SetPackBaudrate* per Software zu setzen.

2.2.4 Die Terminierung von CAN/RS485

Um die jeweiligen Busse zu terminieren, steht pro Schnittstelle je ein Terminator in Form eines Jumpers zur Verfügung (siehe Abbildung 2-1). Die Terminierung wird durch schließen der beiden Kontakte aktiviert.

2.2.5 Voreinstellung des maximalen Motorstroms am QUADpack

Das QUADpack hat als Besonderheit die Möglichkeit einer Voreinstellung des maximalen Motorstroms am Gerät. Die Abstufungen und nötigen Einstellungen an der DIP-Schalterleiste auf der Seite der Motorenstecker können Sie der Tabelle 2 entnehmen.

Motor	1		2		3		4	
Dip-SW	1	2	3	4	5	6	7	8
I =1500mA =100%	ON	ON	ON	ON	ON	ON	ON	ON
I =1000mA = 66%	OFF	ON	OFF	ON	OFF	ON	OFF	ON
I = 500mA = 33%	ON	OFF	ON	OFF	ON	OFF	ON	OFF
I = 0mA = 0%	OFF	OFF	OFF	OFF	OFF	OFF	OFF	OFF

Tabelle 2: Motorstromeinstellung am QUADpack

2.3 Die Steckverbindungen

2.3.1 Stromversorgung

Eine erste Inbetriebnahme des PACKs kann nach Anschluss der Stromversorgung und der seriellen Schnittstelle erfolgen.

Als Stromversorgung kann ein beliebiges Netzteil mit einer Ausgangsspannung von 15-40V genutzt werden. Der benötigte Strom richtet sich nach Verbrauch und Anzahl der Motoren.
Der Steckverbinder für den Anschluss der Stromersorgung ist mit POWER bezeichnet. Des Weiteren ist auf die korrekte Polung zu achten. Auch diese ist auf dem Gerät angegeben.
Ist die Stromversorgung korrekt angeschlossen, leuchten die LEDs „+5V“ und „+24V“ auf und auf der 7-Segment-Anzeige erscheint eine null. Bei Defekt können auf der 7-Segment-Anzeige „8“, „C“ oder „F“ erscheinen.

2.3.2 Serielle Schnittstelle

Die RS232-Schnittstelle am PACK wird über ein Nullmodemkabel mit einer der seriellen Schnittstellen am PC verbunden. Ein Nullmodemkabel besteht aus zwei über Kreuz verbundenen neunpoligen Sub-D-Buchsen.

2.3.3 Motorstecker

ACHTUNG: Vor dem Aufstecken oder Abziehen der Motorenstecker ist stets die Stromversorgung zu unterbrechen. Nichtbeachtung kann zur Zerstörung der Motortreiber führen!

Die Motoren werden über achtpolige Steckerleisten angeschlossen. Die Belegung der Stecker ist, ausgehend vom äußeren Platinenrand, folgende:

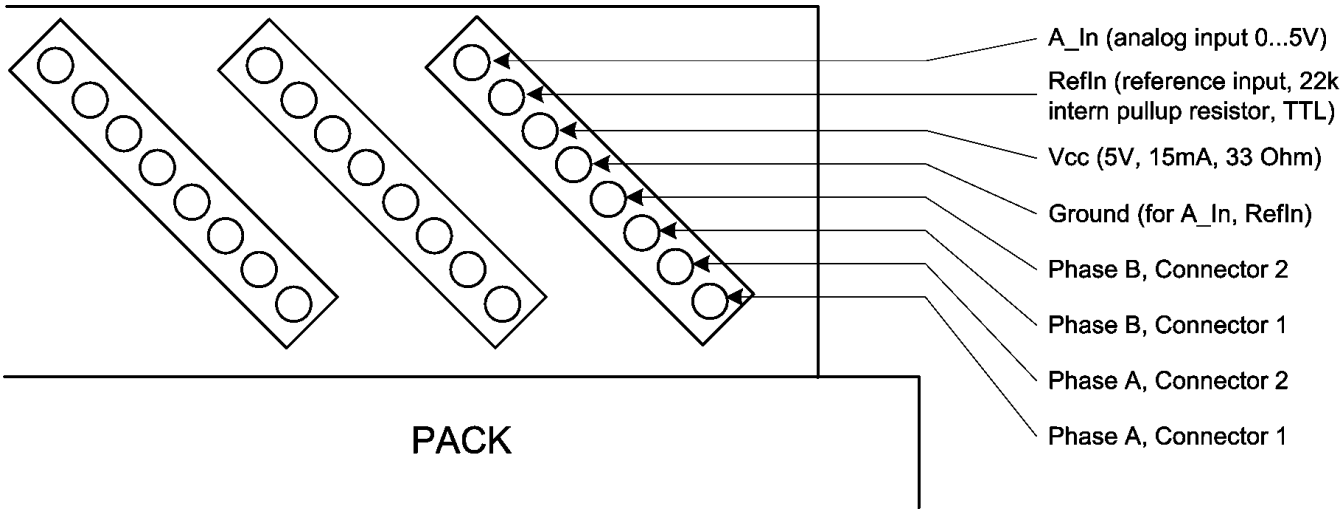


Abbildung 2-2: Belegung der Motorstecker

Für den Testbetrieb sind zunächst nur die Anschlüsse für die Motorenphasen interessant. Verbinden sie die Spulen des Motors wie in Abbildung 2-3 gezeigt mit der Steckerleiste am PACK.

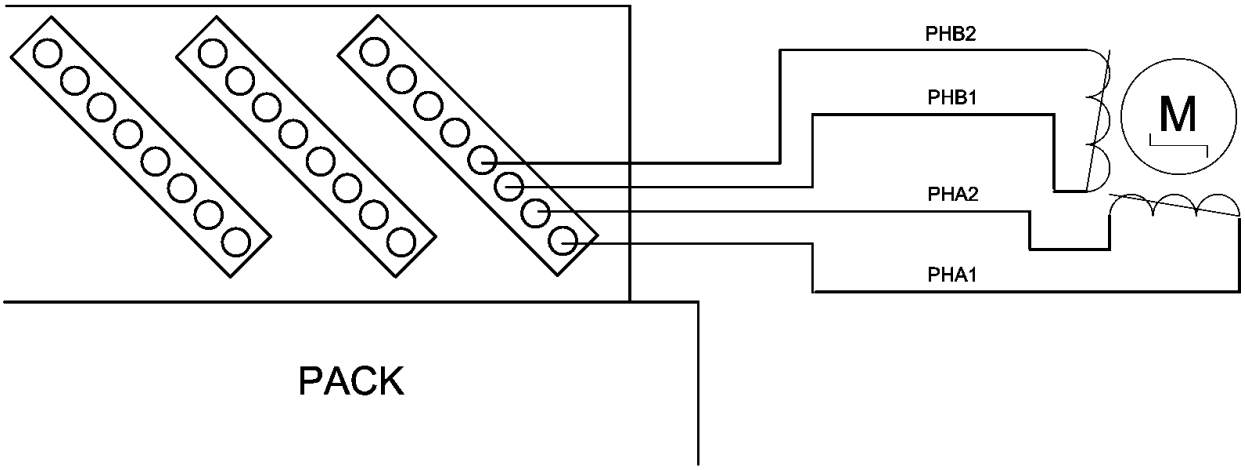


Abbildung 2-3: Anschluss der Motorspulen

2.3.4 Herstellerdaten der Steckverbinder

Motorstecker:	Hersteller:	AMP Connectors (www.amp.com)
	Artikelnummern:	0-0770602-8 (Gehäuse) 0-0770666-1 (Crimpkontakte) 0-0058517-1 (Crimpzange)
Spannungsversorgung/RS485:	Hersteller:	Weidmüller (www.weidmueller.com)
	Artikelnummer:	1716320000 (2-polig, VDC, Ready) 1716360000 (6-polig, RS485)

2.4 Inbetriebnahme mit Software SQPack

Sind die notwendigen Einstellungen an der Hardware des PACKs vorgenommen, können mit dem Windows™-Programm SQPack erste Funktionstest unternommen werden.

2.4.1 Die Installation

Das Programm SQPack wird durch kopieren der Datei „SQPack.exe“ an einen beliebigen Ort auf der Festplatte des PCs installiert. Das Programm erstellt beim ersten Start die Datei „SQPack.ini“, in der Einstellungen gespeichert werden. Um SQPack komplett zurückzusetzen, reicht es diese Datei zu löschen wenn das Programm nicht läuft.

2.4.2 Der Start

Durch einen Doppelklick auf die „SQPack.exe“ wird das Programm gestartet. Es erscheint ein Fenster mit neun Registern. Im Vordergrund befindet sich „Connection“. Hier wird die serielle Verbindung zum PACK hergestellt. Wählen Sie den korrekten COM-Port aus, die übrigen Einstellungen sollten schon auf den richtigen Werten stehen.

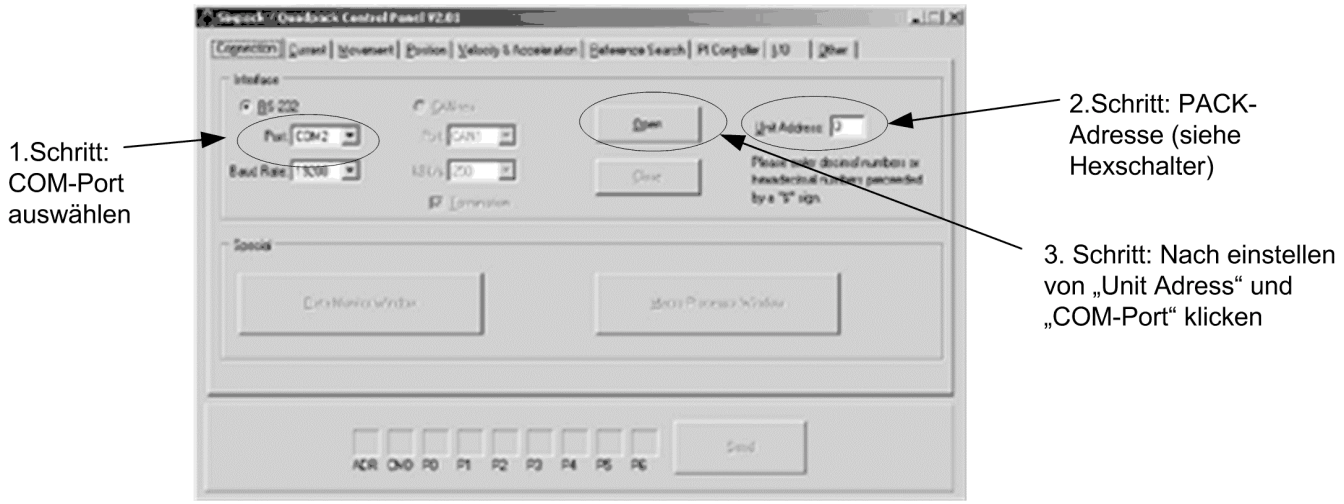


Abbildung 2-4: Setup von „SQPack“

2.4.3 Ein erster Funktionstest: Die Systeminformationen des PACK abfragen

Ob die Verbindung steht, lässt sich über den Befehl *GetUnitInformation* abfragen. Hierzu wählt man das Register „Others“ an und aktiviert im Kasten „Commands“ den Befehl „$43: Get Unit Information“.
Sind die Verbindungseinstellungen vorher korrekt vorgenommen worden, kann nun der Button „Send“ am unteren Rand des Fensters angeklickt und damit der Befehl zum PACK übertragen werden.
Steht die Verbindung zum PACK, sind nun die Werte für „Firmware Revision“, „Reset Flag“, „Temperature“ und „S/N“ im Kasten „Parameters“ eingetragen.

2.4.4 „Erste Schritte“: Einfache Bewegung des Motors

Der Vorteil von Schrittmotoren ist es, gezielt Positionen anfahren zu können. Hierzu beschleunigt das PACK den Motor auf eine Verfahrgeschwindigkeit und bremst ihn rechtzeitig vor erreichen der festgelegten Endposition wieder ab.
Wählen Sie hierzu das Register „Movement“ und wählen Sie den Befehl „Ramp“. Die Variable „Position“ gibt die absolute Zielposition an. Nach Klicken des „Send“-Buttons fährt der Motor die Position an.

2.4.5 Das Konzept der PACK-Steuerung

Das PACK wird generell mit Datagrammen einer Länge von 9 Byte gesteuert. Die Software SQPack zeigt diese Datagramme im unteren Teil des Fensters, direkt neben dem „Send“-Button an. Die neun Bytes werden hierbei hexadezimal dargestellt.
Zu Beginn ist es am einfachsten sich die Datagramme von der Software erzeugen zu lassen und diese dann ggf. in ein eigenes Programm einzufügen.

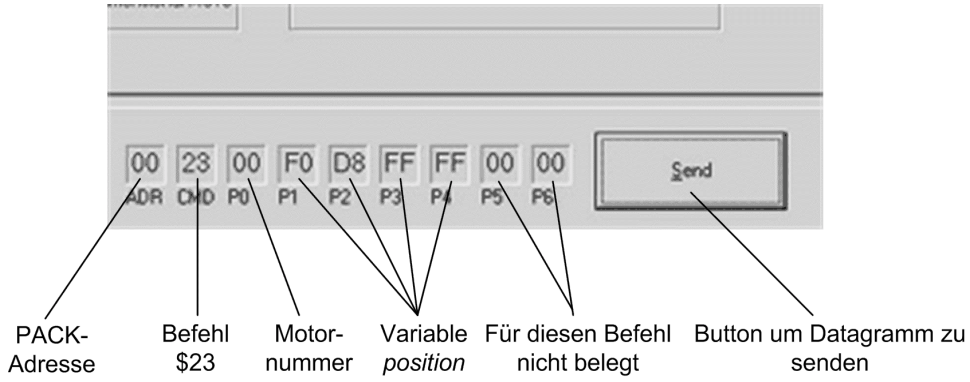


Abbildung 2-5: Darstellung der Befehlsdatagramme in „SQPack“

2.4.6 Die Makrofunktion von SQPack

Die Software verfügt auf dem Register „Connection“ über die Funktion „Macro Processor Window“. Hiermit lassen sich Befehlsfolgen in einer Datei ablegen. Diese Datei ist mit einem ASCII-Editor zu öffnen und enthält die Befehlssequenz in der folgenden Formatierung:

SendToPack (adr, cmd, p0, p1, p2, p3, p4, p5, p6)

„SendToPack“ steht hierbei für eine Funktion, die die neun Bytes über die serielle Schnittstelle an das PACK sendet. Die Parameter sind hierbei als Hexadezimalwerte angegeben.

Ein Makro wird erzeugt, in dem das „Macro Processing Window“ geöffnet und der Button „Record Macro“ anklickt wird. Es öffnet sich ein Dateidialog in dem die Datei anzugeben ist, in der das Makro gespeichert werden soll.
Nun können beliebige Befehle mit SQPack erzeugt und in der Makrodatei gespeichert werden. Beendet wird das Recording durch klicken von „End Recording“.
ACHTUNG: Manche Befehle werden nur bei stehendem Motor akzeptiert. Siehe hierzu die Beschreibung in der Befehlsreferenz.

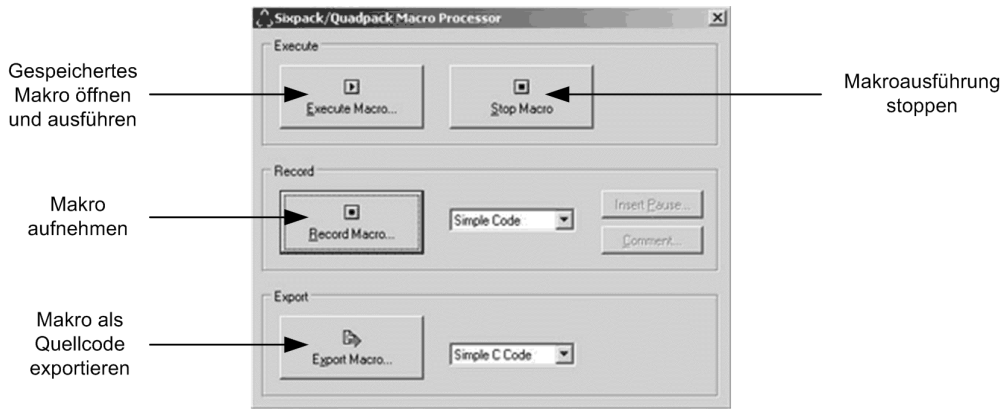


Abbildung 2-6: Die Makrofunktion von „SQPack“

2.5 Betrieb mit Referenz-/Endpunkten

Die meisten Anwendungen für Schrittmotoren benötigen einen Referenzpunkt von welchem ausgehend sich die aktuelle Position des Motors ermitteln lässt.
Mit Hilfe einer Referenzfahrt ermittelt das PACK den Referenzpunkt der jeweiligen Achse.
Die Skalierung der Positionen ist grundlegend bestimmt durch die Art der Anwendung. Bewegt der Motor eine Mechanik mit definiertem Start- und Endpunkt, so handelt es sich um eine **lineare Achse**. Jedem Punkt auf der Achse ist ein Positionswert zwischen dem des Start- und dem des Endpunktes zugeordnet.
Sind für die Achse kein Start- und Endpunkt definiert, so handelt es sich um eine **rotatorische Achse**. Die Positionswerte für die Achse sind für eine Motorumdrehung definiert. Nach einer kompletten Motorumdrehung beginnen die Werte wieder bei null.

2.5.1 Arten der Referenzpunktdefinition

Das PACK bietet zwei grundlegend verschiedenen Arten, den Referenzpunkt zu definieren:

- **Mechanischer Anschlag als Referenzpunkt (nur lineare Achsen).**
  Am Start- oder Endpunkt wird ein Anschlag für die Achsenmechanik vorgesehen. Startet nun eine Referenzfahrt, bewegt das PACK die jeweilige Achse um eine Distanz von `Poslimit` * 5/4 in Richtung des durch das Flag MT_NULLLEFT definierten Anschlages. Durch die Multiplikation von `Poslimit` mit 5/4 wird das PACK den Anschlag schon vor dem Ende der Referenzfahrt erreichen. Der Motor wird dann vom Anschlag festgehalten und dreht durch. Das stellt sicher, dass sich das PACK in jedem Fall am Ende dieser Referenzfahrt am Anschlagpunkt und damit am Referenzpunkt befindet.

- **Elektrischer Referenzschalter.**
  An einem beliebigen Punkt auf der Positionsskala der Achse befindet sich ein elektrischer Schalter, der durch den passierenden Motor ausgelöst wird. Das PACK weiß nun, dass sich an dieser Stelle der Referenzpunkt befindet.
  Befindet sich der Referenzschalter bei einer linearen Achse an einer der Achsenenden, so kann der Referenzschalter zugleich als Endabschalter genutzt werden. D.h. die Achse wird bei Aktivierung des Referenzschalters gestoppt.

2.5.2 Hardwareinstallation bei Nutzung eines Referenzschalters

2.5.2.1 Der Anschluss eines elektrischen Referenzschalters

Der Referenzschalter wird zwischen den Pins „RefIn“ und „GND“ angeschlossen. Optional kann ein Serien-Widerstand von etwa 2,2kΩ eingefügt werden um erhöhten EMV-Ansprüchen gerecht zu werden. In der Regel ist der Einsatz eines Öffners, also passiv geschlossenen Schalters zu empfehlen. So werden Kabelbrüche erkannt.

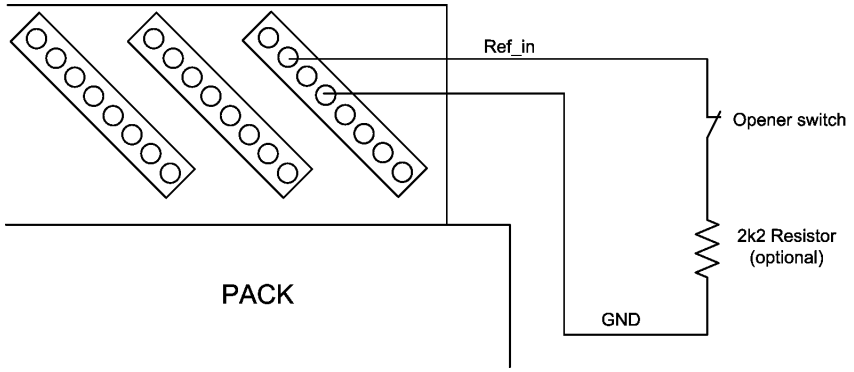


Abbildung 2-7: Anschluss eines Referenzschalters

2.5.2.2 Benutzung der Endpunktabschaltung

Um bei einer linearen Achse das Verfahren über die Endpunkte hinaus zu verhindern, können Endschalter eingesetzt werden. Angeschlossen werden diese am Pin „A_In“ am Motorstecker. Auf die genaue Funktionsweise dieses Eingangs wird später eingegangen, an dieser Stelle soll eine Standardbeschaltung beschrieben werden.
Als Endschalter sollten aus den oben genannten Gründen, Öffner verwendet werden.

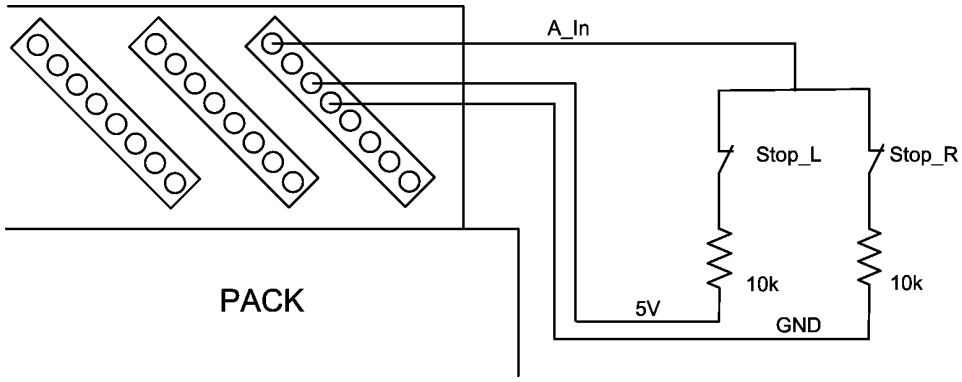


Abbildung 2-8: Anschluss zweier Endpunktschalter

2.5.2.3 Die Kombination von End- und Referenzschalter

Positioniert man den Referenzschalter an einem der Achsenenden, kann er als Endschalter genutzt werden. Auf Seiten der Hardware wird einfach der jeweilige Endschalter weggelassen.

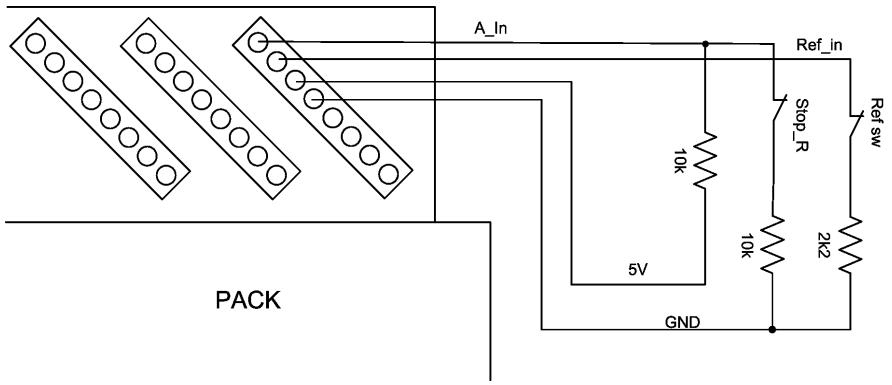


Abbildung 2-9: Kombination von Referenz- und Endpunktschaltern

Sternstraße 67
D – 20357 Hamburg, Germany
Phone +49-40-51 48 06 - 0
FAX: +49-40-51 48 06 - 60
http://www.trinamic.com

2.5.3 Softwarekonfiguration der Referenzfahrt

Die wesentlichen Einstellungen für die Referenzfahrt werden mit dem Befehl *SetMotorParameters* gemacht.

2.5.3.1 Berechnung des Wertebereichs

Die Positionswerte sind durch die Anzahl der Mikroschritte des Motors bestimmt. Das PACK betreibt den Motor mit einem festen Wert von 16 Mikroschritten pro Vollschritt. Über den Schrittwinkel eines Vollschritts gibt das Datenblatt des Motors Auskunft.
Der Wertebereich einer linearen Achse lässt sich am einfachsten experimentell ermitteln. Hierzu setzt man die Achsenmechanik manuell an die Anfangsposition und verfährt sie mit Hilfe der Software SQPack an das Achsenende. Hierzu benutzt man den Befehl „Ramp“ (Beschreibung s.o.) und setzt den Wert von *Position* langsam immer höher, bis man das Achsenende erreicht hat. *Position* gibt nun die Achsenlänge an.

2.5.3.2 Lineare Achse, mechanischer Referenzpunkt

Die für diese Konfiguration wichtigen Parameter sind:

- die Variable `Poslimit`,
- das Flag MT_NULLLEFT,
- das Flag MT_MECHREF.

In `Poslimit` wird der Wertebereich der Achse angegeben (Ermittlung s.o.). Ihn benötigt das PACK um die Länge für die Referenzfahrt zu berechnen (5/4 * `Poslimit`). `Poslimit` als MT_NULLLEFT legt fest, auf welcher Seite der Achse sich der Referenzanschlag befindet. Ist das Flag gesetzt (NullLeft=1), so ist der Anschlag auf der linken Seite der Achse.
Geht man von `Poslimit` = 10000 aus, so erzeugt SQPack (wie oben beschrieben) folgendes Befehlsdatagramm:

SendToPack($00, $15, $00, $10, $27, $00, $00, $10, $04).

Gestartet wir die Referenzfahrt mit dem Befehl *StartReferenceSearch*, in SQPack zu finden unter dem Register „Reference Search“. Das Datagramm dazu lautet:

SendToPack($00, $22, $00, $00, $00, $00, $00, $00, $00).

2.5.3.3 Lineare Achse, Referenzschalter am Achsenanfang

Ausgegangen wird vom der oben beschriebenen Hardwarekonfiguration. Für die Grundkonfiguration sind hier nur die Variable `Poslimit` (Ermittlung s.o.), MT_NULLLEFT und MT_NULLPOSITIVE zu beachten. MT_NULLLEFT und MT_NULLPOSITIVE werden auf 1 gesetzt. Das von QSPack ermittelte Datagramm für eine Achse der Länge `Poslimit` = 33333 lautet:

SendToPack($00, $15, $00, $20, $A1, $07, $00, $10, $40).

Start der Referenzfahrt wie beschrieben.

2.5.3.4 Lineare Achse, kombinierter End-/Referenzschalter links, Endschalter rechts

Wieder wird von der beschriebenen Konfiguration ausgegangen. Neben der Variable `Poslimit`, sind die Flags MT_NULLLEFT und MT_STOPNULL zu setzen.
Ausgehend von `Poslimit` = 500000 sieht ergibt sich folgendes Datagramm:

SendToPack($00, $15, $00, $20, $A1, $07, $00, $50, $40).

Start der Referenzfahrt wie beschrieben.

2.6 Grundlegende Konfigurationsschritte für den Betrieb

2.6.1 Einstellen des Motorstroms

Neben der Möglichkeit des QUADpack den maximalen Motorstrom manuell über den DIP Schalter am Gerät einzustellen, verfügt das PACK über viele Möglichkeiten den Motorstrom per Software an die Anforderungen des Betriebs anzupassen.

Zunächst ist der Maximalstrom zu ermitteln. Diesen können Sie aus dem Datenblatt des Motors entnehmen. Üblicherweise gibt der Hersteller den Strom im Vollschritt an. Für das PACK heißt dies, dass der 1,4-fache Spitzenstrom zulässig ist. Konfiguriert wird dieser Maximalstrom über den Befehl *PeakCurrent* (SQPack-Register „Current“).

ACHTUNG: Der Maximalstrom wird immer für zwei Motoren zusammengesetzt. Dabei gilt der Wert für einen ungeraden Motor n auch für den Motor n+1 !!!

Neben der Motornummer wird der ein Wert übergeben, der als Teiler für den Maximalstrom des PACKs fungiert. Dabei umfasst dieser Teiler die Werte 0-255. Die Formel zur Berechnung des benötigten Wertes ergibt sich wie folgt:

QUADpack: Teiler = (Maximalstrom*256) / (1500mA*DIPSchalter)

SIXpack: Teiler = (Maximalstrom*256) / 800mA

Beispiel-Datagramm für die Konfiguration eines Maximalstroms von 500mA am SIXpack (Teiler = 160) für die Motoren 1 und 2:

SendToPack($00, $10, $00, $A0, $00, $00, $00, $00, $00).

2.6.2 Konfiguration von Beschleunigung und Geschwindigkeit

Die genaue Berechnung und Konfiguration der Schrittfrequenz und eine Beschreibung aller die Schrittberechnung betreffenden Parameter folgt im zweiten Teil des Manuals. Hier soll nur die experimentelle Ermittlung und Konfiguration der Maximalwerte für Beschleunigung und Geschwindigkeiten beschrieben werden.

2.6.2.1 Konfiguration der Startgeschwindigkeit

ACHTUNG: Der Wert darf nur bei stehendem Motor geändert werden!!!

Die Startgeschwindigkeit bezeichnet die Geschwindigkeit zu Beginn einer Fahrrampe. Sie darf in der Standardeinstellung im Bereich 1-256 liegen.

Konfiguriert wird die Startgeschwindigkeit mit dem Befehl *StartVelocity* (SQPack-Register „Velocity & Acceleration“). Der Wert wird in der Variablen `VStart` angegeben.

ACHTUNG: Der Befehl verändert noch zwei weitere Variable, `VMin` und `ClkDiv`. Die Funktion dieser wird in der Befehlsreferenz erklärt. Zunächst reicht es, diese auf die Defaultwerte `VStart` = 5 und `Div` = 2 zu setzen

Beispiel-Datagramm `VStart` = 100, `VMin` = 5, `Div` = 2 für Motor 3 setzen:

SendToPack($00, $13, $02, $0A, $00, $0A, $00, $02, $00)

2.6.2.2 Beschleunigung und Maximalgeschwindigkeit

Beschleunigung und Maximalgeschwindigkeit hängen eng zusammen. Ausgehend von VStart wird die Geschwindigkeit in fester Frequenz um einen in AMax bestimmten Wert erhöht, bis VMax erreicht ist. Eine detaillierte Beschreibung des Verfahrens ist in der Befehlsreferenz zu finden.

Beide Werte werden mit dem Befehl *SetAMaxVMax* gesetzt. AMax muss im Bereich 1 – (VStart*64) liegen, VMax im Bereich 1 – 511.

Das Beispieldatagramm für AMax = 200, VMax = 400, setzen für Motor 4 lautet:

SendToPack($00, $14, $03, $C8, $00, $90, $01, $00, $00).

2.6.3 Bewegungssteuerung

Die verschiedenen Befehle für die Bewegung der Achsen sind gesammelt in Kapitel 3.6 beschrieben.

3 Erweiterte Funktionalitäten

3.1 Beschreibung der Ein-/Ausgänge

3.1.1 Die RS232-Schnittselle

Die RS232-Schnittstelle ist als bidirektionale 2-Draht Schnittstelle ausgelegt und erlaubt Voll-Duplex-Betrieb mit bis zu 255 Teilnehmern. Sie ist am gleichen UART-Bus wie die RS485-Schnittstelle angeschlossen, die Selektion der jeweiligen Schnittstelle erfolgt über den Jumper „RS232/RS485“. Die Funktion der beiden Schnittstellen ist weitestgehend analog, die RS232-Schnittstelle erlaubt jedoch nicht die Anschaltung mehrerer Sender an eine Empfangsleitung. Die `BaudRate` ist auf 19200 Baud vor eingestellt, kann jedoch per Befehl gewechselt werden.

3.1.2 Die CAN-Schnittselle

Die CAN-Schnittstelle ist mit einem Philips SJA1000 CAN-Controller realisiert. Er unterstützt die Full-CAN-Spezifikation 2.0B mit 29 Adressbits, es wird jedoch das Format mit 11 Adressbits verwendet.

Die CAN-Adresse des PACKs ergibt sich wie folgt:

CAN-Adresse = (CANLO + (CANHI*16)) * 8

Daraus ergibt sich der Adressbereich von $008 bis $7F8 in 8-er Schritten.
Beim Empfang des ersten gültigen Datagramms über CAN wird die Auswertung von Befehlen über RS 232 oder RS 485 beendet. Die CAN-Antwortadresse wird mit 8 Bit an das PACK übergeben, wie bei RS 232 / 485. Die Adresse wird für das Antwortdatagramm um 3 Stellen linksverschoben, so dass sich der gleiche Adressbereich wie oben ergibt. Bei anhaltenden Fehlerbedingungen wird CAN und RS 232 / 485 neu initialisiert.
Die Baudrateneinstellung zur CAN-Schnittstelle wird in Kapitel 2.2.2 beschrieben.

3.1.3 Die RS485-Schnittstelle

Die RS 485-Schnittstelle ist als bidirektionale 2-Draht Schnittstelle ausgelegt und erlaubt Halb-Duplex-Betrieb mit bis zu 32 Teilnehmern, mit Busrepeatern bis zu 255 Teilnehmer. Siehe auch Kapitel 3.1.1.

3.1.4 Der Ready-Ausgang

Der Ready-Ausgang kann aktiviert werden (low, open collector, mit internem Pullup auf 4,3V), sobald ein Motor aktiv ist (Geschwindigkeit über 0) oder eine Referenzfahrt macht. Der Ready-Ausgang wird innerhalb von 2 ms nach Start/Ende der Motorbewegung geschaltet. Die Wiederholbarkeit (Jitter) entspricht dabei in etwa der Mikroschrittrate bei Start bzw. Stop (siehe Befehl *SetVMinVStart*).

3.1.5 Multifunktionale Steckerwanne „RS232“

Die 10-polige Steckerwanne „RS232“ auf der Seite der Motoranschlüsse bietet diverse Ein-/Ausgänge, die für zusätzliche Funktionen des PACKs genutzt werden können.
Die Pins 1, 2, 3, 4 sind direkt mit dem Sub-D-Stecker an der RS232-Schnittstelle verbunden.
Pin 5 und 6 liegen an Masse, Pin 7 ist ein Analogeingang, Pin 8 ein Digitalausgang und Pin 9 ein Digitalein-/ausgang mit TTL-Pegel, Pin 10 liefert maximal 5V, 120mA.

Die RS232-Schnittstelle an den Pins 1 bis 4 ist elektrisch identisch mit der Schnittstelle am Sub-D-Stecker. Sie können also nie unabhängig von einander genutzt werden.

Der Analogeingang an Pin 7 ist für ratiometrische Messung eines Widerstandsteilers ausgelegt. Er wird über den Befehl *GetInputValues* ausgelesen.

Der Digitalausgang an Pin 8 arbeitet mit TTL-Pegel und hat einen internen Eingangswiderstand von 270Ω. Sein Wert wird mit dem Befehl *SetOutputs* in der Variablen „TTLOUT1“ ausgelesen. Durch Setzen von TTLOUT1_READY übernimmt der Ausgang die Funktion des Ready-Ausgangs.

Der digitale Ein-/Ausgang an Pin 9 arbeitet mit TTL-Pegel und besitzt einen internen 270Ω-Eingangswiderstand. Er wird konfiguriert und gesetzt mit dem Befehl *SetOutputs*. Setzt man die Variable `TTLIO1`_INPUT = 0 fungiert er als Ausgang und kann über `TtlIo1` gesetzt werden. Setzt man `TTLIO1` = 1 fungiert Pin 9 als Eingang und kann mit Hilfe des Befehls *GetInputValues* in der Variablen `TTLIO1` ausgelesen werden.

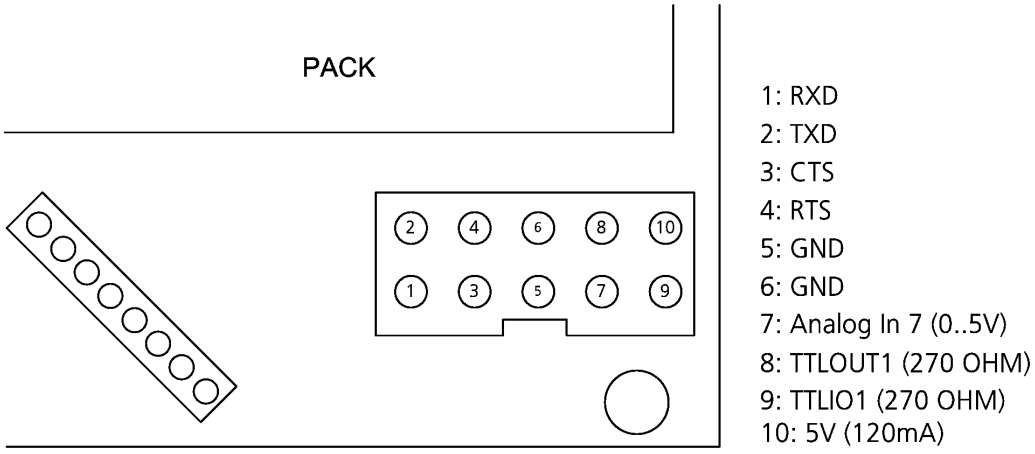


Abbildung 3-1: Belegung der multifunktionalen Steckerwanne

3.1.6 RS 232-Fernsteuerung über CAN-Schnittstelle

Die RS 232-Schnittstelle kann über CAN kontrolliert werden. Dazu wird über den Befehl *SetPackBaudrate* die `BaudRate` gesetzt. Es können nur 8 Bit, 1 Stoppbit und keine Parity verwendet werden. 7 Bit und Parity können natürlich durch den Benutzer simuliert werden. Die Antwortadresse für die auf RS 232 empfangenen Bytes, sowie die Paketgröße für das Weiterreichen von empfangenen Bytes werden über einen separaten Befehl konfiguriert.
Für den Empfang von über RS 232 zu sendenden Bytes wird die für das PACK eingestellte CAN - Adresse um eins inkrementiert, d.h. mit den Endbits 001. Alle auf dieser Adresse empfangenen Bytes werden an die RS 232-Schnittstelle weitergereicht. Pro Übertragung können dabei 1 bis 8 Bytes gesendet werden. Es ist zu beachten, dass der RS 232 Schnittstelle ausreichend Zeit gelassen wird, bevor der nächste Block übertragen wird. Zur Sicherheit kann per Befehl geprüft werden, ob der RS 232-Puffer leer ist. Es findet kein CTS - Handshake statt, die CTS - Leitung kann jedoch abgefragt werden (Befehl *SetRS232OverCan*).

Die über RS 232 empfangenen Bytes werden an die vor eingestellte Antwortadresse gesendet, sobald die eingestellte Anzahl an Bytes empfangen wurde. Fehlerhafte Messages werden jetzt ignoriert. Wenn das einstellbare RS 232-Timeout abgelaufen ist, werden die restlichen Bytes gesendet (Befehl *SetPacketTimeout*).

3.2 Die Programmierung

Das Konzept der PACK-Programmierung basiert auf Befehlsdatagrammen mit fester Länge. Um die Zuordnung der Befehle in einem Feldbus-Netzwerk zu ermöglichen, sind die Datagramme mit der jeweiligen PACK-Adresse versehen.
Wie die Adresse an das Datagramm gekoppelt wird, hängt von der verwendeten seriellen Schnittstelle ab. Das CAN-Protokoll verlangt eine Adressierung für jedes versandte Datenpaket. Bei RS232/RS485 wird jedes Befehlsdatagramm durch ein Adressbyte ergänzt.
ACHTUNG: Alle durch Programmierung gemachten Einstellungen werden nach einem Hardware-Reset zurückgesetzt. D.h. nach Unterbrechung der Stromversorgung oder senden des Befehls *ResetPack* müssen alle Einstellungen erneut zum PACK übertragen werden.

3.2.1 Zusammensetzung der Befehlsdatagramme

Das Befehlsdatagramm an sich besteht aus einem Befehls-Byte und sieben Bytes zur Parameter-Übergabe. Diese acht Bytes werden immer übermittelt, auch wenn sie keine Daten enthalten.

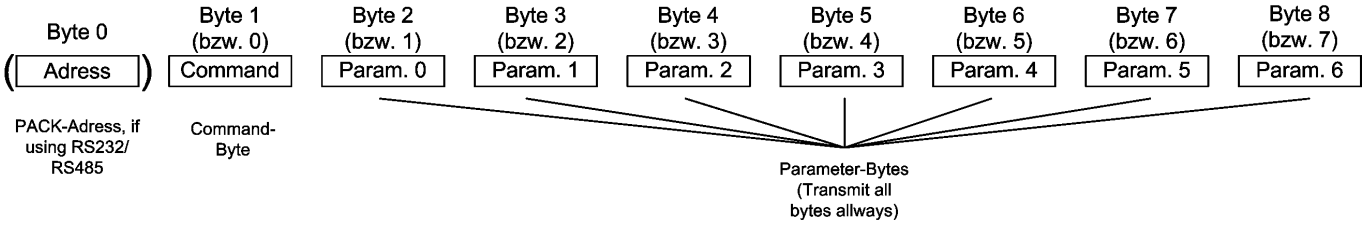


Abbildung 3-2: Zusammensetzung der Befehlsdatagramme

Das den Befehl repräsentierende Byte und der Inhalt der Parameter werden in der Befehlsreferenz (Kapitel 4) beschrieben.

Variablen die größer als ein Byte sind, werden im „least significant byte first"– Prinzip (LSB, niederwertigstes zuerst) übertragen. Negative Werte sind im 2er-Komplement anzugeben.
Anhand des Befehls *SetMotorParameters* hier die Zusammensetzung eines Datagramms:
Das erste Byte kodiert den Befehl, die 7 Parameter setzen sich wie folgt zusammen:
P1 gibt den Motor an, auf den der Befehl angewendet werden soll.
P2 bis P4 die Variable `Poslimit`. Wie beschrieben wird deren Wert nach dem LSB-Prinzip übertragen.
P5 und P6 werden zum Setzen von Flags genutzt. Welches Flag welche Bedeutung hat, wird in der Befehlsreferenz (Kapitel 4) beschrieben.

Enthalten die Bytes Flags, so sind diese als Bit[0-7] bezeichnet. Der Wert des zu übertragenen Parameter-Bytes ergibt sich hierbei aus der bitweisen Addition der Flags nach ihrer Wertigkeit:

$$Px = (2^{\text{Bitnummer des 1. Flags}} + \ldots + 2^{\text{Bitnummer des n. Flags}}).$$

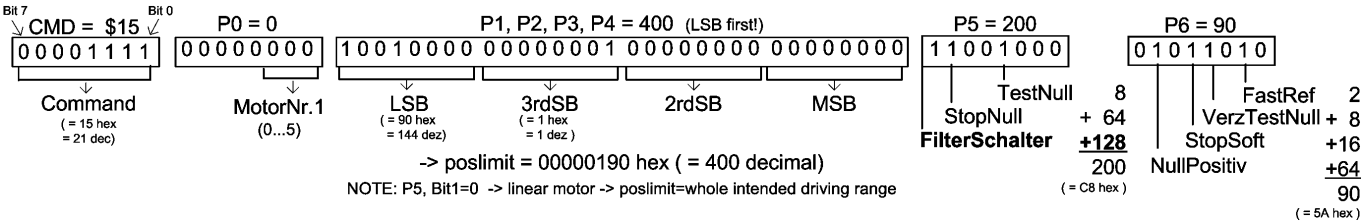


Abbildung 3-3: Beispiel eines Befehldatagramms

Sternstraße 67
D – 20357 Hamburg, Germany
Phone +49-40-51 48 06 - 0
FAX: +49-40-51 48 06 - 60
http://www.trinamic.com

3.2.2 Die Übertragung der Datagramme zum PACK

Das Datagramm muss immer vollständig innerhalb einer parametrierbaren Timeoutzeit (s. *SetPacketTimeout*), übertragen werden. Es wird abgebrochen und nicht ausgewertet, wenn zuvor ein Break-Code empfangen wird (Beschreibung s.u.).

Beim Auslesen von Parametern wird das Antwortdatagramm erst nach einer parametrierbaren Transmitterumschaltzeit (s. Befehl *SetPackBaudrate*), vor eingestellt auf 6 ms, übertragen. Dies ermöglicht dem Sender das Umschalten auf Empfang. Das gleiche gilt für die Umschaltung in entgegen gesetzter Richtung: Das PACK treibt die Leitung nach dem Aussenden eines Datagramms für die vor eingestellte Zeit weiter. Die Richtungsinformation ist an der RS 232-Schnittstelle auf der RTS-Leitung verfügbar (Negativ = PACK sendet). Die CTS-Leitung wird ignoriert.
Bei Empfang eines gültigen Datagramms blinkt die Status-LED kurz auf.

3.2.3 Beschreibung des Break-Codes

Wenn bei der Datenübertragung nach dem Startbit (Null) alle Datenbits "Null" sind und das Stoppbit auch "Null" ist (was normalerweise "Eins" ist) wird ein "Break" generiert und die Serielle Schnittstelle zurückgesetzt.

3.2.4 Die Programmierung über die PACK.DLL

Neben der Möglichkeit die Befehlsdatagramme selbst zu generieren, kann auf die von TRINAMIC™ angeboten PACK.DLL zurückgegriffen werden. Sie lässt sich einfach in C- bzw. C++-Projekte unter Windows™ einbinden und übernimmt die Generierung der Befehlsdatagramme.

Die Benutzung der Befehle wird in der Befehlsreferenz (Kapitel 4 ff.) beschrieben.

3.2.5 Verschachteln von Anfragen

Anfragen dürfen nicht verschachtelt werden. Nach jeder Anfrage sollte die Antwort vom *PACK* abgewartet werden, bevor ein neues Kommando abgesetzt wird. Dies ist besonders wichtig bei Steuerung über RS485, da es sonst zu Buskollisionen kommen kann. Bei Nutzung von RS232 darf jedoch eine verzögerte Rückmeldung (z.B. bei Befehl *GetActivitiesDelayed*) parallel zu einer weiteren Anfrage ausstehen.

3.3 Einstellungen zur Anpassung der Schrittberechnung für den Motor

3.3.1 Berechnung der Micro-Step-Frequenz, bzw. Mikroschrittfrequenz

Das PACK arbeitet mit einer festen relativen Mikroschrittfrequenz von 16 Mikroschritten je Vollschritt. Die Positionen bei der Programmierung sind in Mikroschritten in Relation zum Nullpunkt angegeben.
Wie viele Mikroschritte der eingesetzte Motor für eine Umdrehung benötigt, ergibt sich aus folgender Formel:

Anzahl Mikroschritte/Umdrehung = 360°/Vollschrittwinkel * 16

Der Vollschrittwinkel kann dem Datenblatt des Motors entnommen werden.

Die absolute Mikroschrittfrequenz, also die Anzahl der Schritte pro Sekunde, in Abhängigkeit von der jeweils aktuellen Geschwindigkeitseinstellung für den Motor ergibt sich aus folgender Formel:

$$f_{micro-step} = \frac{20MHz}{clkdiv+1} \cdot v_{akt} \Big/ 2^{14+div_i}$$

- Vollschrittfrequenz=1/16 Mikroschrittfrequenz
- *Clkdiv* (`ClockDivider`) wird für alle Motoren gemeinsam eingestellt (Bereich ist 0..31)
- v_{akt} (`VAkt`) ist die jeweilige Geschwindigkeit des Motors (Bereich: 511..-511)
- div_i (`Div`) ist für jeden Motor parametrierbar (Bereich 0..3)

ACHTUNG: Die Mikroschrittfrequenz darf 200kHz nicht überschreiten.

Beispiel: Das PACK wird veranlasst eine Rampenfahrt zu generieren. Alle für die Rampengenerierung relevanten Parameter sind wie unten angegeben konfiguriert.
Die Startposition (abfragen mit *GetPositionAndActivity*, setzten mit *SetActualPosition*) ist Null, Zielposition gleich 116666 (siehe Befehl: *StartRamp*).

Die aktuelle Vollschrittfrequenz ($f_{full-step}$) ergibt sich aus der oben genannten Formel für $f_{micro-step}$ durch Multiplikation mit 16. `VAkt` in den Beschleunigungsphasen berechnet sich über die Formel

$$\mathrm{VAkt} = \mathrm{VStart} + \frac{\mathrm{AMax}/64}{\mathrm{t}/2\,\mathrm{ms}}$$

Diese Formel ist für die Phase des Abbremsens nicht anwendbar. Um die aktuellen Funktionen des PACKs realisieren zu können, bremst das PACK etwas langsamer ab, als es beschleunigt. Nur in den Firmwareversionen ≤ 1.34 werden eine komplett symmetrische Beschleunigungs- und Abbremsrampe generiert.

Parameter:
`ClockDivider` = 5,
`Div` = 2,
`VStart` = 16,
`VMax` = 128,
`AMax` = 5

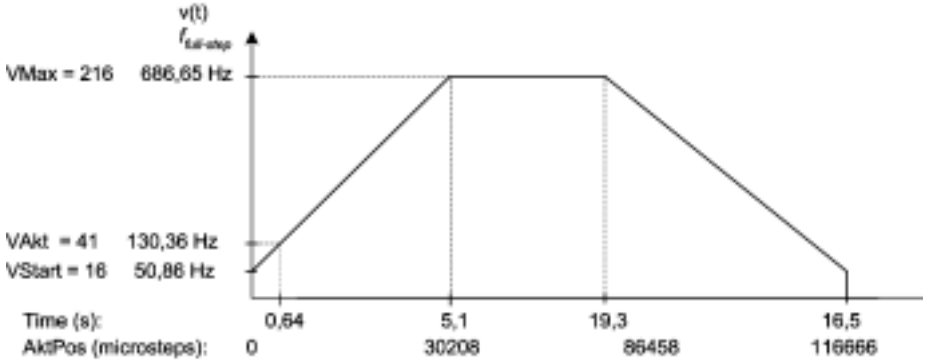


Abbildung 3-4: Beispiel einer Rampengenerierung

Sternstraße 67
D – 20357 Hamburg, Germany
Phone +49-40-51 48 06 - 0
FAX: +49-40-51 48 06 - 60
http://www.trinamic.com

3.3.2 Anpassen der Mikroschritt –Tabelle an die Motoreigenschaften

Das PACK entnimmt die Werte für die Bestromung der Spulen aus einer so genannten Mikroschritttabelle. In dieser Tabelle sind die Stromwerte für die 16 Mikroschritte pro Vollschritt abgelegt.

Grundeinstellung für die Mikroschritttabelle ist ein Viertel einer Sinusphase. Wie in Abbildung 3-1 zu sehen, sind die Stromwerte für die beiden Motorspulen durch eine Sinus- und Kosinusfunktion gegeben.

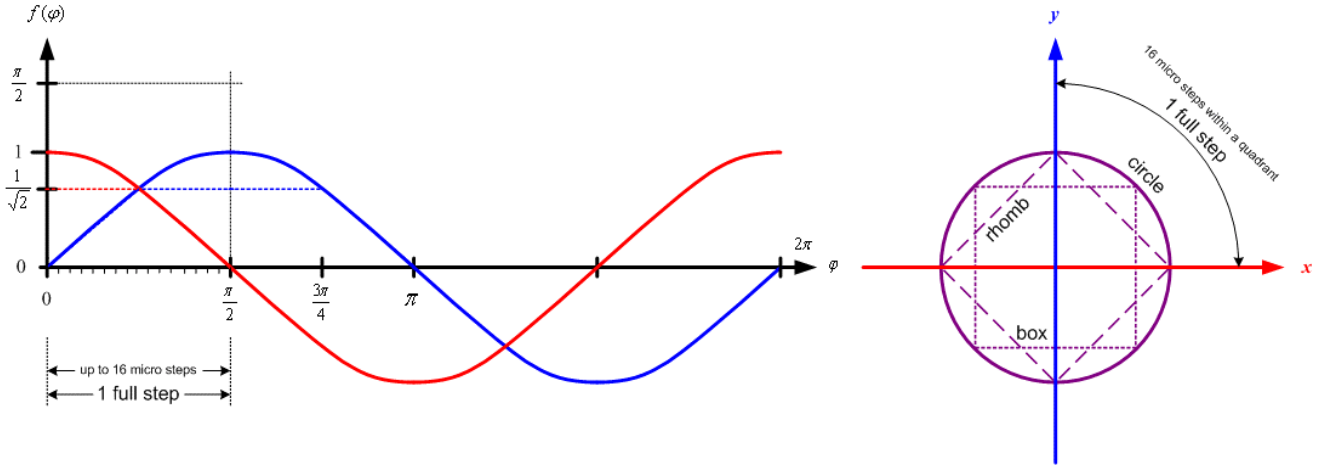


Abbildung 3-4: Generierung der Motorbestromung

Um das Laufverhalten des Motors zu verbessern kann es von Interesse sein, die Werte in der Mikroschritttabelle zu verändern. Hierzu können mit dem Befehl *WriteSineTable* alle Werte individuell verändern werden.

In Abbildung 3-6 sind Default-Mikroschritttabelle und zwei Beispiele für deren Veränderung aufgeführt. Die Werte der Tabelle errechnen sich aus der Formel

$$\text{Entries}[n] = 255 * \sin\left(\frac{I[n]\%}{32} * \pi\right)$$

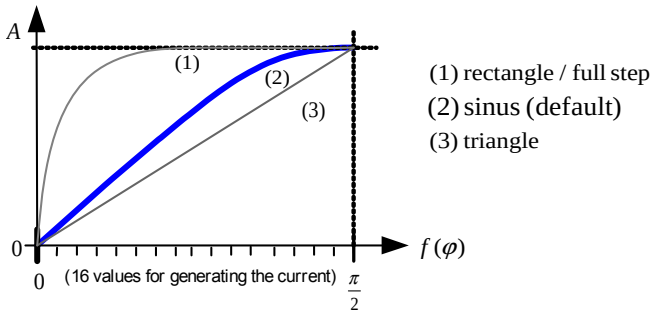


Abbildung 3-5: Die Mikroschritttabelle im PACK

3.4 Details zur Referenzpunkt-Einstellung

Wie bereits in Kapitel 2.5 beschrieben, bietet das PACK viele Möglichkeiten End-/Referenzschaltern anzuschließen und deren Zustände auszuwerten. Neben den beschriebenen Standardkonfigurationen sind auch viele Anpassungen an die individuelle Konfiguration möglich.

3.4.1 Das Koordinatensystem einer Achse

In der Standardkonfiguration steht der Positionszähler jeder Achse auf Null. Die in den Verfahrbefehlen angegebenen Werte für die Zielposition bezeichnen die absolute Anzahl von Mikroschritten zwischen dem Nullpunkt und dem Zielpunkt. Der maximale Betrag an Mikroschritten pro Achse wird über die Variable `Poslimit` im Befehl *SetMotorParameters* eingestellt.

3.4.2 Der Referenzpunkt/Referenzschalter

Der Referenzpunkt ist durch den mechanischen Anschlag bzw. den Referenzschalter definiert.
Wird als Referenzschalter, im Gegensatz zur Beschreibung in Kapitel 2.5.1, ein aktiv geschlossener Schalter eingesetzt, ist das Flag MT_NULLPOSITIVE = 0 (Befehl *SetMotorParameters*) zu setzen.

3.4.3 Verschieben des Nullpunktes

In der Grundkonfiguration markiert der Referenzpunkt gleichzeitig den Nullpunkt. Mit dem Befehl *SetNullPointOffset* kann jedoch eine Nullpunktverschiebung vorgenommen werden. Die ist z.B. sinnvoll, wenn der Referenzschalter in der Mitte der Achse liegt, die Programmierung aber nur mit positiven Positionswerten arbeiten soll.

3.4.4 Automatische Referenzfahrt

Das PACK bietet weiterhin die Möglichkeit, bei erkanntem Schrittverlust eine automatische Referenzfahrt anzustoßen. Hierzu wird durch setzten des Flags MT_TESTNULL (Befehl *SetMotorParameters*) ständig der Wert des internen Positionszählers mit dem Status des Referenzschalters verglichen. Zeigt der Referenzschalter am Referenzpunkt keine Reaktion bzw. zeigt den Referenzpunkt nicht an der intern logischen Position an, wird eine Referenzfahrt gestartet und das Flag STOPFLAG (Befehl *GetPositionAndActivity*) gesetzt.

3.4.5 Einstellung des Aktivitätsbereiches des Referenzschalters

Mechanische Ungenauigkeiten können dazu führen, dass der Referenzschalter ein paar Schritte zu früh bzw. zu spät auslöst. Das kann bei Nutzung von MT_TESTNULL zu ungewollten Referenzfahrten führen. Mit dem Befehl *SetNullPointOffset* lässt sich die Variable `TestNullRange` setzen. Sie legt einen Bereich um den Nullpunkt fest, in dem der Status des Referenzschalters nicht überprüft wird.

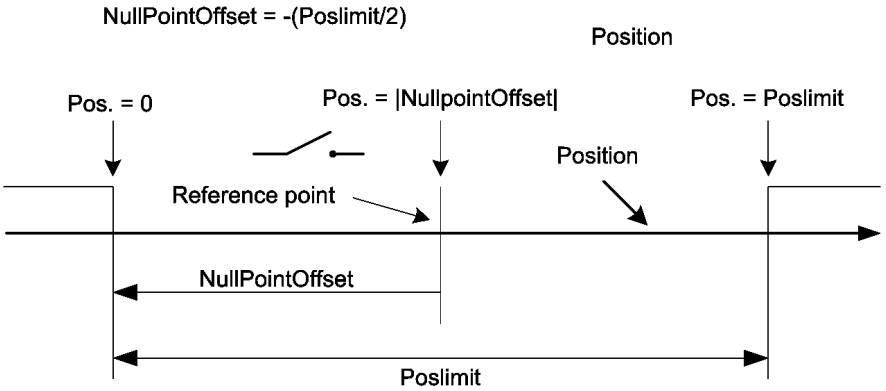


Abbildung 3-6: Koordinatentransformation über „NullPointOffset“

Sternstraße 67
D – 20357 Hamburg, Germany
Phone +49-40-51 48 06 - 0
FAX: +49-40-51 48 06 - 60
http://www.trinamic.com

3.4.6 Verzögerung des Referenzschalters ausgleichen

Bei mechanischen Referenzschaltern ergibt sich häufig das Problem, dass zwischen mechanischer und elektronischer Statusänderung einige Millisekunden verstreichen. D.h. das Signal über den aktuellen Status des Referenzschalters liegt leicht verzögert am Eingang des PACKs an. Damit dies nicht zu Störungen führt, lässt sich mit `PowerDownDelay` die Abfrage des Referenzschaltersignals verzögern.
`PowerDownDelay` wird im Befehl für die Stromsteuerung (Befehl *SetCurrentControl*) gesetzt und wird auch von dieser genutzt. MT_DELAYTESTNULL ist in der Grundkonfiguration gesetzt, `PowerDownDelay` ist auf 500 gesetzt.
ACHTUNG: `PowerDownDelay` wird ebenso für die „Power Down"-Verzögerung genutzt (Befehl *SetCurrentControl*).

3.4.7 Elimination von Störimpulsen auf der Leitung zum Referenzschalter

In der Grundeinstellung wird das Signal am Eingang RefIn über 22 ms entprellt. D.h. der Pegelwechsel an RefIn muss über diese Zeit Bestand haben, damit er als solcher interpretiert wird.
Die Entprellzeit wird in der Variablen `DebouncingTime` (Befehl *SetRefSearchParameters*) gesetzt.

3.4.8 Einstellung der Referenzfahrtgeschwindigkeit

In der Grundeinstellung ist die schnelle Referenzfahrt aktiviert. Zuständig hierfür ist das Flag MT_FASTREF (Befehl *SetMotorParameters*). In dieser Konfiguration wird die Achse bei der Referenzfahrt mit der in der Variablen *VRefMax* gesetzten Geschwindigkeit verfahren (Default *VRefMax*=100).

ACHTUNG: Wenn MT_FASTREF genutzt wird, muss zwingend die Achsenlänge in der Variablen `Poslimit` gegeben sein. Ansonsten kann das PACK keine ordnungsgemäße Referenzfahrt vollziehen. Wird MT_FASTREF nicht gesetzt, wird `Poslimit` nicht zwingend benötigt. Die Referenzfahrt wird dann mit der festen Geschwindigkeit 1 durchgeführt.

3.5 Details der Endschalterkonfiguration

Wie bereits in Kapitel 2.5 beschrieben, ist es sinnvoll den Fahrweg für lineare Achsen zu begrenzen. So können Schrittverluste erkannt und Beschädigungen an der Mechanik vermieden werden.

3.5.1 Die Funktionsweise von A_In als Endschaltereingang

Wie in Kapitel 2.5 beschrieben, können am Eingang A_In Endschalter zum begrenzen des Fahrbereiches angeschlossen werden.
Neben dem schon beschriebenen Anschluss von zwei öffnenden Endschaltern, sind auch andere Konfigurationen möglich:
A_In ist, wie bereits beschrieben, ein Analogeingang. Wird er als Endschaltereingang genutzt, werden Spannungswerte unter einer festgelegten Schwelle als Aktivität des linken, Spannungswerte über einer anderen Schwelle als Aktivität des rechten Endschalters interpretiert. Die Schwellen werden über den Befehl *SetStopSwitchLimits* in den Variablen `MinLeft` bzw. `MaxRight` eingestellt. Alle Werte zwischen diesen Schwellen als Inaktivität der Endschalter, also als gültiger Fahrbereich der Achse interpretiert.

Es ist also wichtig, dass durch die Beschaltung sichergestellt ist, dass die an A_In anliegende Spannung bei ausgelöstem linken Schalter kleiner und bei ausgelöstem rechten Schalter größer ist als die Spannung bei nicht ausgelösten Endschaltern. Das kann wie im Beispiel durch einen Spannungsteiler geschehen (z.B. Stop_L = GND, Stop_R = 5 V, weder Stop_L noch Stop_R = 2,5 V).

Die Spannungspegel können auch kontinuierlich, durch einen an die Achse gekoppelten Drehwiderstand erzeugt werden.

3.5.2 Kombination von End- und Referenzschaltern

Liegt der Referenzschalter an einem der Achsenenden, kann man ihn zugleich als Endschalter benutzen. Die Vorgehensweisen für die Installation der Hardware sowie die darauf bezogenen Konfigurationen sind analog zu den Beschreibungen oben.

Aktiviert wird diese Funktion durch Setzen des Flags MT_STOPNULL (Befehl *SetMotorParameters*). Das Flag MT_NULLLEFT (Befehl *SetMotorParameters*) bestimmt, auf welcher Seite des Fahrbereichs sich der kombinierte End-/Referenzschalter befindet (MT_NULLLEFT = 0 → End-/Referenzschalter rechts).

3.5.3 Konfiguration eines „security Margin" für einen kombinierten End-/Referenzschalter

Wie bereits in Kapitel 3.4.5 beschrieben, können Fehler durch zu früh bzw. zu spät auslösende Schalter entstehen. Deshalb empfiehlt es sich, einen kombinierten End-/Referenzschalter mit einem so genannten „security margin" zu versehen.
Der mit dem Befehl *SetMargin* gesetzte Wert legt die Anzahl an Schritten fest, die das PACK in einen ausgelösten Referenzschalter hinein fährt (Siehe auch Abbildung 3-4).

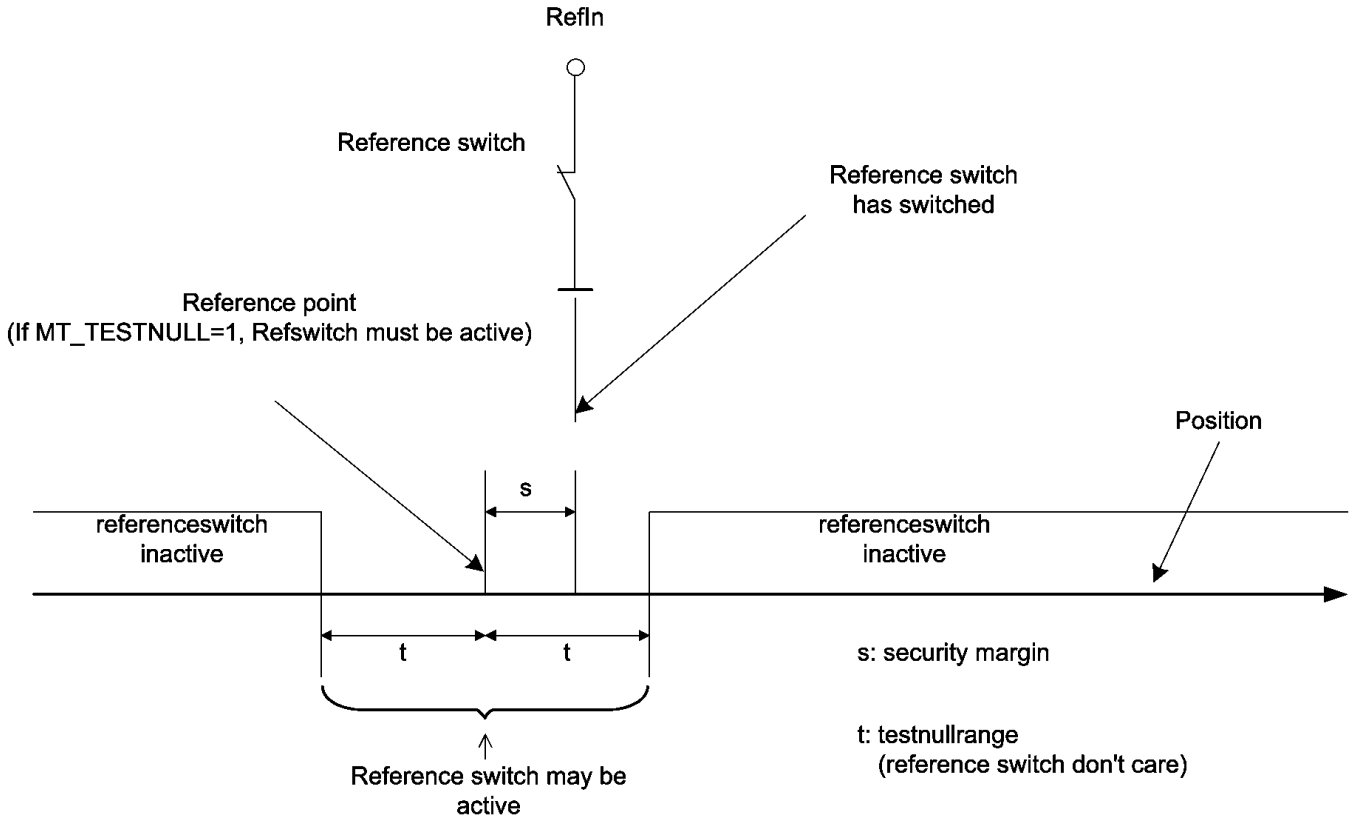


Abbildung 3-7: Feineinstellungen zur Referenzschalterfunktion

3.6 Die verschiedenen Befehle für die Bewegung der Achsen

Das PACK bietet verschiedene Möglichkeiten eine Achse zu verfahren. Welche Möglichkeit man wählt, hängt stark von der Anwendung ab.
Grundsätzlich gibt es die Möglichkeiten den Motor von a nach b fahren (Rampenbetrieb) oder mit einer festen Geschwindigkeit (Rotation) bis zur Eingabe eines neuen Befehls fahren zu lassen.

3.6.1 Einfache Rampenfahrt

Die Rampenfahrt setzt die vorhergehende Konfiguration der Referenzfahrt, den Werten für `Poslimit` Beschleunigung und Geschwindigkeit voraus.
Durch den Befehl *StartRamp* kann jede beliebige, durch `Poslimit` definierte Position angefahren werden. Das Ziel wird in der Variablen `TargetPosition` angegeben. Vor Erreichen der Zielposition wird der Motor abgebremst, so dass er die Zielposition genau erreicht.

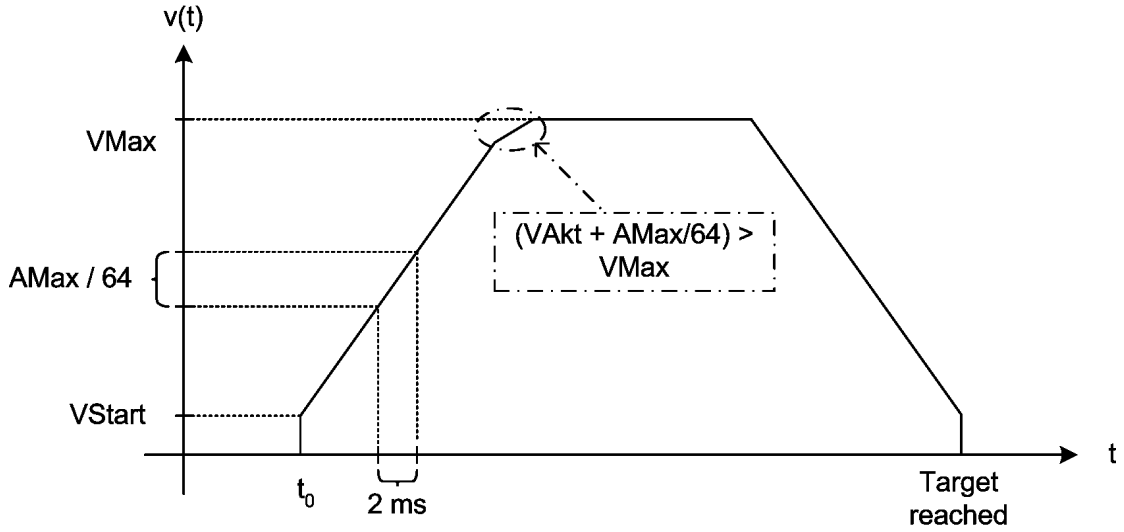


Abbildung 3-8: Schema der Rampengenerierung

3.6.2 Starten einer konstanten Rotation

Um den Motor einfach mit einer konstanten Geschwindigkeit rotieren zu lassen, gibt es den Befehl *ConstantRotation.* Als Parameter dieses Befehls wird die Variable `Velocity` übergeben, welche die Geschwindigkeit für die konstante Bewegung angibt (Berechnung siehe Kapitel 3.3).
Nach Übertragen des Befehls wird der jeweilige Motor mit der in der Variablen `AMax` (Kapitel 3.3) gegebenen maximalen Beschleunigung auf die durch `Velocity` gegebene Geschwindigkeit beschleunigt.
Zum Anhalten der Achse, wird erneut der Befehl *ConstantRotation* aufgerufen und `Velocity` auf 0 gesetzt.

3.6.3 Ändern der Zielposition einer Rampenfahrt

Um die Zielposition einer aktiven Rampenfahrt zu ändern, wird der Befehl *SetTargetPosition* genutzt. Als Parameter wird die Variable `TargetPosition` mit dem neuen Ziel der Fahrt übertragen.
Falls der Motor seine vorher definierte Zielposition schon erreicht hatte und steht, hat *SetTargetPosition* keine Wirkung. Um daraus resultierende Fehler zu vermeiden, empfiehlt es sich im Anschluss immer noch einmal den Befehl *StartRamp* mit der neuen Zielposition als Parameter aufzurufen.
War der Motor vorher im Modus *ConstantRotation* aktiv, wird er zunächst zum stehen gebracht und dann auf die neue Zielposition verfahren.

3.6.4 Synchrones Starten mehrerer Motoren

Zum synchronen starten mehrer Rampenfahrten, gibt es den Befehl *StartRampParallel.*
Bevor *StartRampParallel* aufgerufen wird, sollten die jeweiligen Zielpositionen für die Achsen mit dem Befehl *SetTargetPosition* gesetzt werden. Wenn dann alle Motoren inaktiv sind, kann die parallele Rampenfahrt gestartet werden.

Beispiel: In Abbildung 3-9 wird eine parallel gestartete Rampenfahrt von Motor 1 und Motor 2 dargestellt. Die Parameter für Geschwindigkeit und Beschleunigung (siehe Kapitel 3.6.1) sind für beide Motoren identisch. Die zu fahrende Strecke allerdings ist bei Motor 1 länger. Die Motoren starten beide gleichzeitig, beschleunigen synchron und fahren mit der gleichen Maximalgeschwindigkeit. Motor 2 erreicht sein Ziel früher und beendet deshalb seine Rampe früher als Motor 1.

3.6.5 Starten einer linearen Interpolation über mehrere Achsen

Der Befehl *StartInterpolation* ermöglicht eine linear interpolierte Bewegung mehrerer Achsen. Alle Motoren erreichen das Ziel gleichzeitig. Die Zielposition muss zuvor für die einzelnen stehenden Motoren mit dem Befehl *SetTargetPosition* gesetzt werden.
Die Achse mit dem längsten Fahrweg, also die mit der längsten Fahrzeit bei maximaler Geschwindigkeit, fungiert als Führungsachse. An ihr orientieren sich die Geschwindigkeiten der anderen Achsen. Trotzdem kann es zu leichten Zeitunterschieden beim Erreichen der Zielpunkte kommen. Deshalb sollte vor dem Absetzen eines neuen Fahrbefehles der Status aller an der Bewegung beteiligten Motoren geprüft werden.

Beispiel: Abbildung 3-10 stellt eine linear interpolierte Rampenfahrt dar. Motor 1 hat einen längeren Fahrweg als Motor 2. Deshalb wird Motor 1 auf die in VMax vorgegebene Geschwindigkeit beschleunigt, für Motor 2 jedoch eine geringere Maximalgeschwindigkeit berechnet. So erreichen beide Motoren ihr Ziel gleichzeitig. Die Werte für die Beschleunigung bleiben weiterhin gültig, deshalb sind die Steigungen der Rampe während der Beschleunigung identisch.

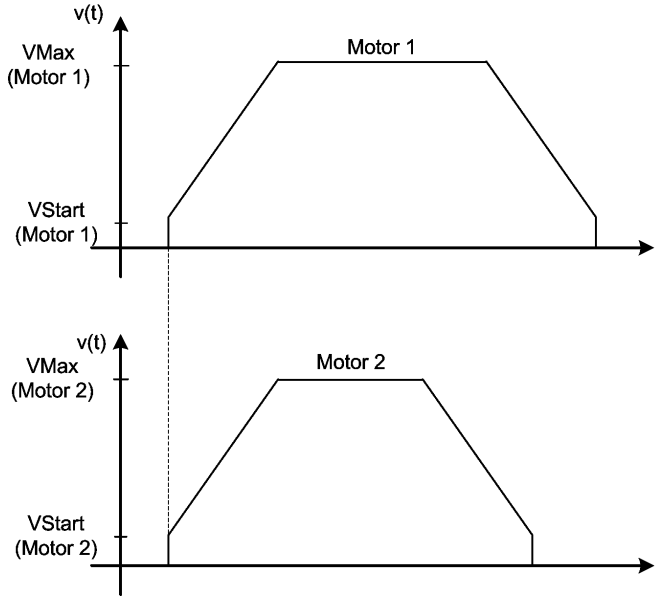


Abbildung 3-9: Synchroner Start von Motoren

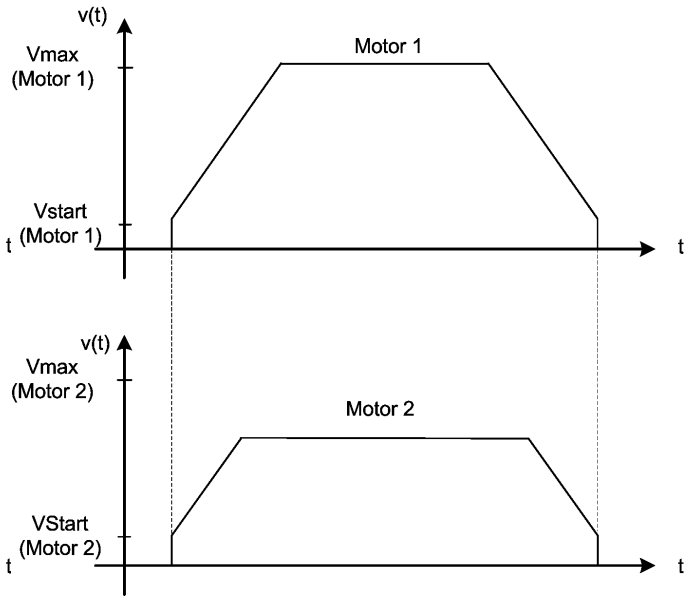


Abbildung 3-10: Linear interpolierte Fahrt

Sternstraße 67
D – 20357 Hamburg, Germany
Phone +49-40-51 48 06 - 0
FAX: +49-40-51 48 06 - 60
http://www.trinamic.com

3.6.6 Konfiguration für rotatorische Bewegungen

Wie in Kapitel 2.5 angedeutet, wird bei der Konfiguration grundsätzlich zwischen der Nutzung des PACKs für lineare und rotatorische Antriebe unterschieden. Der jeweilige Modus wird über das Flag MT_ROTARY (siehe Befehl *SetMotorParameters*) ausgewählt.
Wichtigster Unterschied ist die Interpretation der Positionswerte. Im Linearbetrieb wird die die Differenz zwischen aktueller und Zielposition als absolute Anzahl der zu fahrenden Mikroschritte interpretiert. Zielpositionen können auch außerhalb des durch `Poslimit` bestimmten Wertebereiches liegen.
Der Wertebereich bei rotatorischen Achsen ist streng auf den Wert von `Poslimit` begrenzt. Positionsangaben die über diesen Bereich hinausgehen, werden durch den Modulo-Operator um Vielfache bereinigt. Ob sich die Positionswerte im rotatorischen Betrieb im Positiven oder Negativen befinden, ist abhängig vom Flag MT_NULLLEFT (siehe Befehl *SetMotorParameters*):

MT_NULLLEFT = 0: 0 ≤ Positionswert < `Poslimit`
MT_NULLLEFT = 1: `-(Poslimit )` < Positionswert ≤ 0

Die Bewegungsrichtung des Motors ist in der Standartkonfiguration des rotatorischen Betriebs identisch mit dem linearen Betrieb. D.h. Bewegungen aufsteigender Koordinaten führen zu Rotation im, in absteigender Richtung gegen den Uhrzeigersinn.
Wenn das Flag MT_OPTIMIZEWAY = 1 (siehe Befehl *SetMotorParameters*) gesetzt wird, wählt das PACK den **kürzesten Weg** von der Start- zur Zielposition. D.h. wenn `Poslimit` = 1000, die Istposition gleich 900 und die Zielposition gleich Null, so wird der Motor 100 Schritte im Uhrzeigersinn auf die Position 0 = 1000 bewegt.

3.7 Steuerung des Motorstroms

Damit Motor und PACK nicht überhitzen, das Dreh- bzw. Haltemoment jedoch stets ausreichend ist, sollte eine individuelle Konfiguration der Motorströme vorgenommen werden.
Der maximale Motorstrom `IMax` wird mit dem Befehl *SetMaxCurrent* eingestellt. Beim QUADpack ist zusätzlich die Einstellung am Gerät notwendig (siehe Kapitel 2.2.5).
Der Befehl *SetCurrentControl* bietet zusätzlich die Möglichkeit, den Motorstrom an die aktuelle Motorbewegung anzupassen. In den Variablen können prozentuale Abstufungen, ausgehend vom Maximalstrom festgelegt werden.
Denkbar wäre eine maximale Bestromung während der Beschleunigung (`AccelerationCurrent` = 0), in der Phase konstanter Geschwindigkeit wird der Strom auf 75% abgesenkt (`RunningCurrent` = 1). Ist die Zielposition erreicht, wird der Strom auf 38% abgesenkt (`StandigCurrent` = 3) und nach Ablauf von `PowerDownDelay` sinkt der Strom auf 9% (`PowerDownCurrent` = 7).
Um die Stromaufnahme und damit auch die Wärmeentwicklung zu verringern, sollten für alle ungenutzten Motorausgänge der Wert für des PowerDown-Stroms auf Null gesetzt werden (`PowerDownCurrent` = 8).

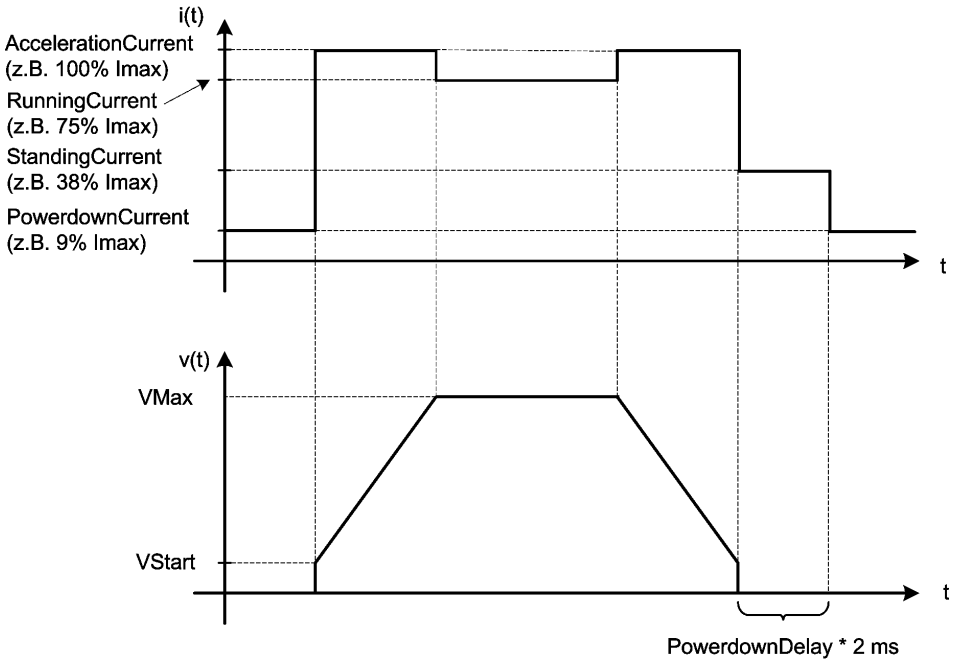


Abbildung 3-11: Rampenabhängige Rampenfahrt

3.8 Defaultwerte

Direkt nach dem Anschalten des PACKs bzw. nach Senden des Befehls *HardwareReset* werden alle Variablen und Einstellungen in einen Default Zustand zurückgesetzt. Alle hier nicht aufgeführten Werte sind im Default Zustand = 0.

Flags:
MT_DELAYTESTNULL = 1
MT_ FASTREF = 1
MT_FILTERSWITCH = 1
MT_NULLCENTER = 1

Variable:
`ActiveMask` = $3F (= bin111111 = alle Motoren aktiviert)
`AMax` = 128
`ClockDivider` = 5
`DebouncingTime` = $0FFF (= 22 ms)
`Div` = 2
`IntClip` = 129
`IntDiv` = 129
`IntIntpClip` = 1
`MaxCurrent` = 128
`Poslimit` = 6400 (= 400 Vollschritte)
`ReferenceSearchMask` = $3F (= bin111111 = alle Motoren aktiviert)
`PowerDownDelay` = 500 (= 1000 ms)
`TestNullRange` = 240 (= 15 Vollschritte)

Sternstraße 67
D – 20357 Hamburg, Germany
Phone +49-40-51 48 06 - 0
FAX: +49-40-51 48 06 - 60
http://www.trinamic.com

4 Befehlsreferenz

In der Befehlsreferenz sind alle für das PACK verfügbaren Befehle aufgeführt.
Die Überschriften stellen die Befehlsbezeichnungen dar. Sie sind als solche z.B. auch in der zum PACK angebotenen Windows™-DLL übernommen. Gleiches gilt für Variablen und Flags.
Mit CMD ist das den Befehl repräsentierende Byte für die Datagramm-Programmierung bezeichnet. Das vorgestellte $-Symbol gibt an, dass der Wert im Hexadezimalformat angegeben ist. P0 – P7 bezeichnen den Inhalt der Parameter-Bytes (siehe auch Kapitel 3.2).

Einige Befehle fassen mehrere Funktionen, manche Funktionen werden mit mehreren Befehlen konfiguriert. Dies liegt begründet in der Entstehung der PACK-Firmware. Beachten Sie hierzu die Hinweise in den Befehlsbeschreibungen.

Die Motoren sind von 0 bis 3 (QUADpack) bzw. 0 bis 5 (SIXpack) durchnummeriert.

4.1 Befehls-Seitenindex

4.2 Stromsteuerfunktionen

SetPeakCurrent

Befehl legt den Maximalstrom der jeweiligen Achse fest. Der Wert gilt für jeweils zwei Motoren. Gesetzt wird der Wert für den Motor 0, 2, 4 wobei die Werte auch für die Motoren 1, 3, 5 gelten.

CMD	$10
P0	`MotorNr` *(0, 2, 4)*
P1	`MaxCurrent` (0..255): SIXpack: Imax=0,8A * Wert/256 QUADpack: Imax=1,5A*Wert/256 * **DIPsetting** (0%, 33%, 66%, 100%)

SetCurrentControl

ACHTUNG: `PowerDownDelay` wird auch für das Testnull-Delay (Befehl SetMotorParameters, Flag MT_DELAYTESTNULL) benötigt!

Bietet die Möglichkeit den Motorstrom in Abhängigkeit der ausgeführten Aktion zu setzen. Hierbei können, in einer prozentualen Abstufung von 100%, 75%, 50%, 38%, 25%, 19%, 13%, 9% und 0%, die jeweiligen Stromwerte für die unten genannten Zustände gesetzt werden.
Die prozentualen Stromwerte werden in den unten genannten Variablen als Wert zwischen 0-8 gesetzt, wobei 0 = 100% und 8 = 0% bedeutet.

Hinweis zur Energieeinsparung**:** Eine deutliche Reduktion der gesamten Stromaufnahme kann erreicht werden, indem `PowerDownCurrent` für alle (ungenutzten) Motorausgänge auf 0% gesetzt wird.

CMD	$11
P0	`MotorNr` *(0...5)*
P1	`PowerDownCurrent`: Power Down- Strom (0..8).
P2	`StandingCurrent`*:* Strom bei stehendem Motor (vor Ablauf von `PowerDownDelay`).
P3	`RunningCurrent`*:* Strom während konstanter Geschwindigkeit
P4	`AccelerationCurrent`*:* Strom während Beschleunigung
P5,6 #	`PowerDownDelay`: (1..65535) Power Down- Verzögerungszeit. Einstellung in Schritten zu 2 ms (`Wartezeit = PowerDownDelay * 2 ms`).

4.3 Geschwindigkeit & Beschleunigung

SetStepFrequency

ACHTUNG: Einstellung gilt für alle Motoren gemeinsam, nur bei stehenden Motoren ändern!

Setzt die Variable `ClockDivider`, den Vorteiler für die Schrittfrequenzberechnung fest. Dieser Wert gilt für alle Motoren gemeinsam. Der achsenindividuelle Vorteiler `Div` wird im Befehl *SetVMinVStart* beschrieben. Siehe auch Kapitel 3.3.

CMD	$12
P0	`ClockDivider` (0..31): s. Berechnung der Schrittfrequenz (Default = 5)

SetVMinVStart

ACHTUNG: nur bei stehendem Motor ändern!

VMin ist nur für die Referenzfahrt relevant. Es setzt die Geschwindigkeit, etwa beim präzisen Suchen nach dem „Anschlagpunkt“ des Referenzschalters fest.

VStart gibt die Geschwindigkeit an mit der Fahrrampen gestartet bzw. beendet werden. Durch Einstellung eines möglichst hohen Wertes für VStart, erreicht der jeweilige Motor schnell die maximale Geschwindigkeit und muss diese auch erst spät vor Erreichen des Ziels abbremsen. Ein zu hoher Wert kann aber dazu führen, dass der Motor nicht anlaufen kann oder die genaue Zielposition verfehlt.

Div ist der für jede Achse individuell einstellbare Schrittfrequenz-Vorteiler (siehe auch Kapitel 3.3.1). Ein kleines Div ermöglicht hohe Schrittfrequenzen und damit maximale Achsenumdrehungen, ein großes Div hingegen erhöht die Auflösung der Geschwindigkeitsstufen und damit auch feinerer Beschleunigungsrampen. Dadurch lassen sich bei hohen Geschwindigkeiten die Zielpositionen bei Rampenfahrt präziser anfahren.

CMD	$13
P0	MotorNr *(0...5)*
P1,2 #	VMin (0..511): Minimalgeschwindigkeit für Referenzfahrt Bei Übergabe von 0 wird der optimale Wert für die Konfiguration berechnet. VMin <= 250Hz / fMikroschrittMin; VMin <= VStart (fMikroschrittMin ist die Mikroschrittfrequenz bei Geschwindigkeit 1)
P3,4 #	VStart (1..511): Startgeschwindigkeit bei Rampenfahrt. Zulässiger Maximalwert ist abhängig von Div: Div = 3: 0 < VStart < 512 Div = 2: 0 < VStart < 256 Div = 1: 0 < VStart < 128 Div = 0: 0 < VStart < 64
P5	Div (0..3): s. Berechnung der Schrittfrequenz (Default: Div =2)

SetAMaxVMax

ACHTUNG: AMax und VMax werden permanent überprüft! Änderungen werden sofort behandelt!

AMax gibt die Steigung der linearen Beschleunigungsrampe an. Ausgehend von VStart (Startgeschwindigkeit, siehe *SetVMinVStart*) wird VAkt (die aktuelle Geschwindigkeit, Abfrage über *GetVelocityAndActivity*) alle 2 ms um den festen Wert 1/64 * AMax inkrementiert. Wenn der nächste Erhöhungsschritt den Wert von VMax überschreiten würde, wird VAkt auf VMax gesetzt.
VMax gibt den Wert für die Maximalgeschwindigkeit an. Nach welcher Formel sich die Geschwindigkeit berechnet, wird in Kapitel 3.3.1 beschrieben.

CMD	$14
P0	MotorNr *(0...5)*
P1,2 #	AMax (1..32767): Maximale Beschleunigung des Motors. Funktionsweise: while (VAkt >= VMax) do VAkt = VAkt + (AMax/64); VAkt = VMax;
P3,4 #	VMax (1..511): Maximaler Wert für VAkt

Sternstraße 67
D – 20357 Hamburg, Germany
Phone +49-40-51 48 06 - 0
FAX: +49-40-51 48 06 - 60
http://www.trinamic.com

4.4 Motor- und Referenzfahrtparameter

SetMotorParameters

ACHTUNG: Hinweise auf der nächsten Seite!

CMD	$15
P0	`MotorNr` *(0...5)*
P1,2,3,4	`Poslimit`: Bei rotatorischen Achsen: Mikroschritte pro Umdrehung (Vollschritte * 16). Bei linearen Achsen (MT_ROTARY = 0): Mindestens der gesamte Fahrbereich, um ungewollte oder unterbrochene Referenzfahrten zu vermeiden!
P5	`MotorType` (untere acht Bits): BitNr. Flagbezeichnung Funktionsbeschreibung Bit0 MT_PI 0 = Rampenfahrt (Default) 1 = PI-Regler Bit1 MT_ROTARY 0 = lineare Achse (Default) 1 = *rotatorische* Achse Bit2 MT_AUTONULLCMD 1 = nach Senden des Befehls wird Referenzfahrt durchgeführt (siehe auch *ReferenceSearch*) Bit3 MT_TESTNULL 1 = Wenn interner Positionszähler auf Position des Referenzpunktes, Referenzschalter jedoch nicht aktiv wird STOPFLAG (siehe *GetPositionAndActivity*) gesetzt. Es folgt eine Referenzfahrt wenn MT_STOPNOREF nicht gesetzt. Wird nur bei stehendem Motor überprüft! Bit4 MT_NULLLEFT 0 = Referenzpunkt rechts vom Fahrbereich 1 = Referenzpunkt links vom Fahrbereich Bit5 MT_NULLCENTRE 1 = Referenzpunkt wird in der Mitte des Aktivitätsbereiches des Referenzschalters

	gesucht. Ist für rotatorische Achsen besonders sinnvoll. (Default) Bit6 MT_STOPNULL 1 = Definiert den Referenzschalter als Endschalter (Achsenseite wird durch MT_NULLEFT definiert). Wenn Referenzschalter aktiviert, wird STOPFLAG gesetzt (siehe *GetPositionAndActivity*). Zudem wird überprüft, ob Status des Referenzschalters und Wert des internen Positionszählers gemeinsam gültig sind. Wenn nicht und MT_STOPNOREF=0 wird eine Referenzfahrt durchgeführt Bit7 MT_FILTERSWITCH 1 = Referenzschalter wird per Software entprellt (siehe *SetRefSearchParameters*, Variable `DebouncingTime`) (Default)

P6	`MotorType` (obere acht Bits): BitNr. Flagbezeichnung Funktionsbeschreibung Bit0 MT_OPTIMIZEWAY 1 = Wegoptimierung für rotatorische Achsen durch Auswahl von Links-/Rechtslauf Bit1 MT_FASTREF 1 = Referenzfahrt mit hoher Geschwindigkeit (siehe *SetRefSearchParameters*) (default) Bit2 MT_MECHREF 1 = Mechanischer Anschlag für Referenzfahrt (siehe Kapitel 2.5.1) Bit3 MT_DELAYTESTNULL 1 = Zustand des Referenzschalters und interner Positionszähler werden um die in der Variablen `PowerDownDelay` (siehe *SetCurrentControl*) gesetzte Zeit verzögert verglichen. Dies verhindert Fehler durch Ungenauigkeiten in der Mechanik in Verbindung mit MT_TESTNULL. (default) Bit4 MT_STOPSOFT 0 = Motor wird bei Eintreten von Stopp-Bedingung (Endschalter aktiviert) abrupt gestoppt. Da es hierbei zu Schrittverlusten kommen kann, schließt sich eine Referenzfahrt an. Die kann durch setzen von MT_STOPNOREF verhindert werden. 1= Motor wird bei Eintreten von Stopp-Bedingung mit einer negativen Beschleunigung vom Betrag `AMax` abgebremst. Sollte die Stopp-Bedingung vor dem Stillstand aufgehoben werden, wird der vorangegangene Fahrbefehl weiter ausgeführt. Bit5 MT_STOPNOREF

	1 = Es wird keine automatische Referenzfahrt ausgeführt, wenn Stopp-Bedingung eingetreten. Bit6 MT_NULLPOSITIVE 0 = Referenzeingang ist low aktiv (Schalter ist Schließer) 1 = Referenzschalter ist high aktiv (Schalter ist Öffner) Bit7 MT_STOPFULLSTEPS 1 = Fahrrampen werden so berechnet, dass sie nur an Vollschrittpositionen enden

ACHTUNG: Einige Flags sind in der Standarteinstellung schon gesetzt (siehe „Default“ in der Beschreibung hier oder Kapitel 3.7). D.h. diese Funktionen sind von Beginn an aktiviert. Wird nun *SetMotorParameters* aufgerufen, werden alle Flags so wie in der Variablen `MotorType` angegeben, gesetzt.

PACK_DLL: Der Parameter `MotorType` sollte aus den Konstanten MT_xxx aus dem Headerfile durch verodern gebildet werden.

SetRefSearchParameters

ACHTUNG: `VRefMax` gilt nur für die schnelle Referenzfahrtrelevant. nur bei stehendem Motor ändern!

PACK_DLL: Das Flag STOPAFTERREF ist in der Version 1.0 der DLL nicht enthalten.
Die Variable `DebouncingTime` gibt direkt die Zeit an (maximal 30, nur gerade Zahlen erlaubt). `Mask` (P3/P4) wird dann automatisch erzeugt.

CMD	$16
P0	`MotorNr` *(0...5)*
P1,2 #	`VRefMax` (1..511): Referenzfahrt- Maximalgeschwindigkeit `VMax` >= `VRefMax` >= `VStart`; `VRefMax` <= 511
P3,4 #	`DebouncingTime`: Maske für Referenzschalterentprellung ($0001=0ms, $0003=2ms, ...$FFFF=30ms) Default = $0FFF (=22ms)
P5 #	STOPAFTERREF: 0 = macht nach Referenzfahrt mit akt. Aktion weiter 1 = Stopp nach Referenzfahrt

WriteSineTable

Die Motoren werden positionsabhängig mit analogen Werten bestromt. Die unteren fünf Bit der Position jedes Motors bilden einen Zeiger in die symmetrische Kennlinientabelle und geben den Strombetrag für Spule A des Motors vor. Spule B wird um 16 Schritte verschoben gemäß den Tabelleneinträgen bestromt. Zur verbesserten Anpassung an die Motorcharakteristik kann die Tabelle für alle Motoren gemeinsam verändert werden. Dazu wird nur die untere Hälfte (16 Einträge) beschrieben.

Die Werte der Standard-Tabelle ergeben sich aus der Formel 255 * SIN([0.5..15.5]/32*PI).
Genaueres beschreibt Kapitel 3.2.2.

PACK_DLL: Variable `Entries` ist als Zeiger auf ein Array einer Länge 4 zu realisieren.

CMD	$17
P0	`TableStart`: Zeiger auf Tabelle (Start = 0,4,8 oder 12)
P1..P4 #	`Entries`: 4 Tabellenwerte (0..255) ab der angegebenen Position (z.B. Tabelle=255, 255, ..., 255 -> Vollschritt)

SetNullPointOffset

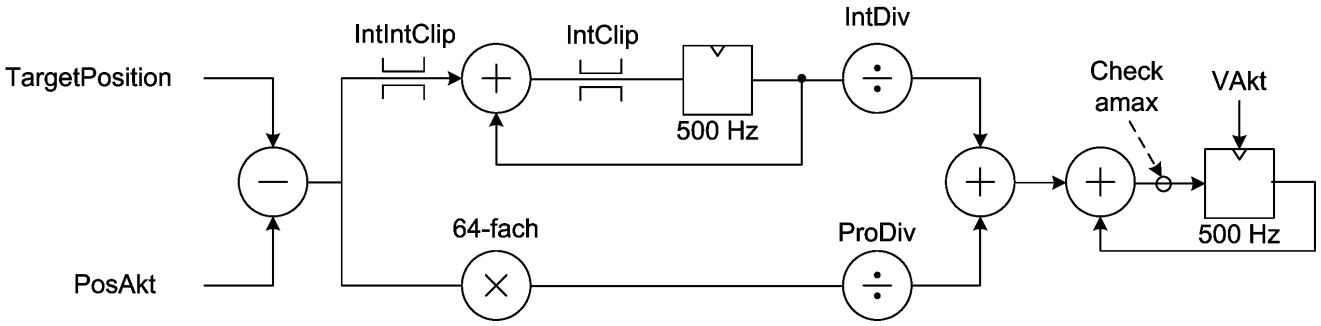
Der Referenzpunkt-Offset legt den Referenzpunkt auf die angegebene Position. Wird dieser Befehl nicht verwendet ist der Referenzpunkt identisch mit dem Nullpunkt. Es handelt sich nur um eine Koordinatentransformation, die am Verhalten der Referenzfahrten nichts ändert. Sie kann z.B. genutzt werden, um den Nullpunkt an das Ende des Fahrbereichs zu verlegen, etwa bei Referenzpunkten in der Mitte einer Fahrstrecke, um dann nur mit positiven Koordinaten arbeiten zu müssen. Siehe auch Kapitel 3.4.3.

Wenn gleichzeitig der Test auf zu frühes Unterbrechen des Referenzschalters (siehe Befehl *SetMotorParameters*, Flag *MT_STOPNULL*) eingeschaltet ist, kann über die Variable `TestNullRange` ein kleiner Bereich um den Referenzpunkt herum vom Test ausgenommen werden. Außerhalb dieses Bereiches werden bei Fahrt auf den Referenzpunkt über den Referenzschalter ein Nothalt und eine anschließende Referenzfahrt ausgelöst. Siehe auch Kapitel 3.4.5.

CMD	$18
P0	`MotorNr` *(0...5)*
P1,2,3,4 #	`NullOffset` (signed long): Distanz zwischen Nullpunkt und Referenzschalter
P5,6 #	`TestNullRange` (0..65535): Bereich in dem Nullschalter aktiv sein darf

SetPIParameters

Der PI-Regler ermöglicht es, alleine durch häufige Positionsvorgaben über den Hostrechner eine gemittelte Fahrkurve zu erzeugen. die Faktoren für Integral- und Proportionalteil bestimmen dabei, wie scharf der Regler reagiert. Der Regler arbeitet mit 500 Hz. Der Proportionalteil ist 64*Positionsdifferenz/`PropDiv`. Der Integralteil integriert jeweils nur den durch `IntIntpClip` begrenzten Anteil der Positionsdifferenz. Der Einfluss des Integralteils wird durch den `Divisor` `IntDiv` bestimmt.
(Vorteil: Vermeiden von ruckartigen Bewegungen!)



CMD	$19
P0	`MotorNr` *(0...5)*
P1	`PropDiv` (1..255)
P2,3 #	`IntDiv` (1..32767)

P4,5 #	`IntClip` (1..32767)
P6	`IntIntpClip` (0..255)

SetMargin

Dieser Befehl ermöglicht es, bei linearen Antrieben ein Spiel des Referenzschalters auszugleichen. Es wird bei der Referenzfahrt um `Margin` weiter in den Referenzschalter hinein gefahren und der Referenzpunkt erst dort gesetzt. Dies ist sinnvoll, wenn die Referenzpunktüberprüfung (siehe *SetMotorParameters*, MT_TESTNULL) verwendet wird.

CMD	$1A
P0	`MotorNr` *(0...5)*
P1,2 #	`Margin` (0..65535)

4.5 Bewegungsfunktionen

GetPositionAndActivity
Dieser Befehl gibt als Antwort die aktuelle Position des abgefragten Motors und Angaben über die ausgeführte Aktion zurück. Darüber hinaus gibt das STOPFLAG an, ob seit der letzten Benutzung dieses Befehls eine Stopp-Bedingung für diese Achse eingetreten ist.

CMD	$20
P0	`MotorNr` *(0...5)*
P1	`ResponseAdress`: Adresse des Host zum Empfangen der Antwort

Antwort	
CMD	$20
P0	`MotorNr` *(0...5)*
P1,2,3,4 #	`PosAkt` (signed long): aktuelle Position
P5	`Activity`: momentane Aktion (0: inaktiv, 5: Rampe, 10: PI-Regler, 15: Rotation, ≥ 20: Referenzfahrt
P6	STOPFLAG*:* 1=Stop Bedingung über Schalter ist eingetreten. Wird bei Abfrage zurückgesetzt.(=0)

GetVelocityAndActivity
Befehl gibt als Antwort die aktuelle Geschwindigkeit und, wie auch *GetPositionAndActivity*, die momentan ausgeführte Aktion zurück.

CMD	$21
P0	`MotorNr` *(0...5)*
P1	`ResponseAdress`: Adresse des Host zum Empfangen der Antwort

Antwort	
CMD	$21
P0	`MotorNr` *(0...5)*
P1,2 #	`VAkt` (integer): aktuelle Geschwindigkeit
P3	`Activity`: momentane Aktion (0: inaktiv, 5: Rampe, 10: PI-Regler, 15: Rotation, ≥ 20: Referenzfahrt

ReferenceSearch
Der Motor fährt in der vorkonfigurierten Richtung mit der Geschwindigkeit `VMin`, bis der Referenzpunkt über den Referenzpunktschalter eindeutig identifiziert wurde. Die Position wird dann auf null gesetzt.
Wenn das Flag STOPAFTERREF = 0 (siehe *SetRefSearchParameters*), fährt der Motor nach Ende der Referenzfahrt wieder an die Position, von der er die Referenzfahrt gestartet hat.

CMD	$22
P0	`MotorNr` *(0...5)*

StartRamp
ACHTUNG: Bei rotatorischen Achsen sind bei MT_NULLLEFT = 0 nur negative, bei MT_NULLLEFT = 1 nur positive Werte definiert!

Der Motor fährt von der aktuellen Position auf die Zielposition. Das Kommando wird nicht ausgeführt, wenn der Motor noch in der Rampenfahrt ist.

Um die Zielposition für einen sich bewegenden Motor zu ändern, siehe Kapitel 3.6.3

CMD	$23
P0	`MotorNr` *(0...5)*
P1,2,3,4 #	`TargetPosition` (32 Bit signed long)

PITargetPosition
Die Motorposition wird über einen PI-Regler auf die Sollposition geregelt. Die Umschaltung auf den PI-Regler findet sofort statt, auch wenn der Motor noch aktiv ist. Ist der PI-Regler bereits aktiv, wird die Zielposition sofort programmiert.

CMD	$24
P0	`MotorNr` *(0...5)*
P1,2,3,4 #	`TargetPosition` (32 Bit signed long)

ConstantRotation
Über diesen Befehl können Achsen mit der jeweils angegebenen Geschwindigkeit rotiert werden. Die Maximalbeschleunigung (Variable `AMax`) wird bei Änderung der Geschwindigkeit beachtet. Die Umschaltung auf Rotation findet sofort statt, auch wenn der Motor noch aktiv ist.
Abhängig von der eingestellten `DebouncingTime` kann es passieren, dass ab einer bestimmten Drehzahl keine Überprüfung des Referenzschalters mehr stattfindet. Die Zeit seiner Aktivität wird dann durch die Filterung komplett überlagert.

CMD	$25
P0	`MotorNr` *(0...5)*
P1,2 #	`Velocity` (-511..511)

SetTargetPosition
Verändert die Zielposition. Kann z.B. auch zur Verkürzung / Verlängerung einer Fahrrampe verwendet werden. Die Änderung der Zielposition und Fahrt auf das neue Ziel findet sofort statt, wenn der Motor aktiv ist. Bei stehendem Motor erfolgt eine Speicherung der Zielposition, es wird jedoch keine Motorbewegung ausgelöst.

CMD	$26
P0	`MotorNr` *(0...5)*
P1,2,3,4 #	`TargetPosition`: (32 Bit signed long)

SetActualPosition

ACHTUNG: Wird ein Referenzschalter verwendet, kann die Benutzung dieses Befehls zu einer Referenzfahrt mit anschließender Zurücksetzung des Positionszählers führen.
Der Befehl verändert den Wert des internen Positionszählers.

CMD	$27
P0	`MotorNr` *(0...5)*
P1,2,3,4 #	`PosAkt`: neue aktuelle Position (32 Bit signed long)

GetActivitiesDelayed

ACHTUNG: Bei RS 485-Betrieb mit mehreren Teilnehmern kann dies zu Buskollisionen führen! Um dies zu vermeiden, darf während einer ausstehenden Anfrage deshalb keine weitere Busnutzung erfolgen.

PACK_DLL: Dieser Befehl ist in der Version 1.0 der DLL nicht enthalten.
Dieser Befehl ermöglicht es, sämtliche Achsen auf ihre Aktivität abzufragen. Wird eine verzögerte Rückmeldung angefordert, sendet das PACK diese erst, nachdem alle nicht ausmaskierten Motoren inaktiv geworden sind und liefert die Aktion aller Motoren zurück.

CMD	$28
P0	`ResponseAdress`: Adresse des Host zum empfangen der Antwort
P1	Maske für verzögerte Rückmeldung (Bit 0=Motor 0, Bit 5=Motor 5) (0: Motor ausmaskiert, 1: erst antworten, sobald Motor inaktiv)

Antwort	
CMD	$28
P0..P5 #	`PosAkt`: aktuelle Aktion des jew. Motors (0..5) 0: inaktiv, 5: Rampe, 10: PI-Regler, 15: Rotation, 20 - 29: Referenzfahrt, 30: mechanische Referenzfahrt

StartRampParallel

Dieses Kommando ermöglicht einen synchronen Start, von mehreren Achsen.
Die Zielpositionen der jeweiligen Achsen sind vorher mit dem Befehl *SetTargetPosition* zu setzen.
Wird der Befehl *StartRampParallel* nun zum PACK übertragen, werden alle durch die unten angegebenen Maskierung adressierten Motoren veranlasst, die in der Variablen `TargetPosition` gesetzte Position anzufahren.
Eine neue parallele Rampenfahrt darf erst wieder gestartet werden, wenn alle beteiligten Motoren inaktiv sind.

PACK_DLL: Die Variable `Mask` sollte durch verodern der Konstanten MOTOR0..MOTOR5 aus dem Headerfile gebildet werden.

CMD	$29
P0	`Mask`: Maske für Rampenstart (Bit0=Motor0, Bit 5=Motor5) (0: Motor ausmaskiert, 1: Motor starten)

StopMotors
Mehrere Motoren können gemeinsam abgebremst werden, indem über dieses Kommando die Zielposition mit der aktuellen Position gleichgesetzt wird. Die Motoren werden dennoch mit der in AMax festgelegten negativen Beschleunigung abgebremst.

PACK_DLL: Die Variable Mask sollte durch verodern der Konstanten MOTOR0..MOTOR5 aus dem Headerfile gebildet werden.

CMD	$2A
P0	Mask: Maske für Abbremsen (Bit0=Motor0, Bit5=Motor5) (0: Motor ausmaskiert, 1: Zielposition auf aktuelle Position setzen)

StartInterpolation
Koordinierte synchrone Rampenfahrt mehrerer Motoren. Alle Motoren erreichen ihr jeweiliges Ziel gleichzeitig. Die Zielposition muss zuvor für die einzelnen (stehenden) Motoren gesetzt werden (Befehl *SetTargetPosition*).

Die Achse mit dem längsten Fahrweg, also die mit der längsten Fahrzeit bei maximaler Geschwindigkeit, fungiert als Führungsachse. An ihr orientieren sich die Geschwindigkeiten der anderen Achsen. Trotzdem kann es zu leichten Zeitunterschieden beim Erreichen der Zielpunkte kommen. Deshalb müssen vor dem absetzen eines neuen Fahrbefehles der Status aller an der Bewegung beteiligten Motoren abgefragt werden.

Mit Mask werden die an der interpolierten Fahrt beteiligten Motoren ausgewählt. Es können auch mehrere interpolierte Rampenfahrten mit unterschiedlichen Motoren gleichzeitig stattfinden.

PACK_DLL: Die Variable Mask sollte durch verodern der Konstanten MOTOR0..MOTOR5 aus dem Headerfile gebildet werden.

CMD	$50
P0	Mask: Maske für die Motoren (Bit0=Motor0, Bit5=Motor5) (0: Motor ausmaskiert, 1: Motor an Bewegung beteiligt)

Sternstraße 67
D – 20357 Hamburg, Germany
Phone +49-40-51 48 06 - 0
FAX: +49-40-51 48 06 - 60
http://www.trinamic.com

4.6 I/O-Funktionen

GetInputValues

Die Analogkanäle sind für ratiometrische Messungen von Widerstandsteilern ausgelegt. Kanal 0 bis 5 bezeichnen die Eingänge an den Motorsteckern (A_In), Kanal 6 ist der externe Eingang, Kanal 7 misst die Versorgungsspannung des PACKs (1V ergibt Wert 22). Die Referenzeingänge (RefIn) geben den anliegenden Pegel invertiert aus.

Die Variable `AllInputs` enthält die Werte aller digitalen Eingänge. Die Bits 0-5 enthalten die invertierten Pegel der Referenzeingänge (RefIn), Bit 6 und 7 geben die Jumperkonfiguration für die CAN-Adressierung zurück (siehe auch Kapitel 3.1.2).

CMD	$30
P0	`Channel`*:* (0=kanal0 ... 7=kanal7)
P1	`ResponseAdress`: Adresse des Host zum empfangen der Antwort

Antwort	
CMD	$30
P0	`Channel`*:* (0=kanal0 ... 7=kanal7)
P1,2 #	`AnalogueValue`*:* (unsigned 0..1023)
P3	`RefInput` : (Bit 0)
P4	`AllInputs`: alle Referenzeingänge/Jumper (Bit0 = Motor 0, ..., Bit6=Jumper1, Bit7 = Jumper2)
P5	`TtlIo1`: Pegel an `TTLIO1`

SetStopSwitchLimits

Über ein Potentiometer oder eine Widerstandsbeschaltung von zwei Endschaltern, kann über den Analogeingang jedes Motors ein Stopp (hart oder weich) ausgelöst werden. Die Spannung am Analogeingang soll in Fahrtrichtung rechts ansteigen. Der gemessene Wert wird in Abhängigkeit von der Fahrtrichtung überprüft. Wenn eine Stopp-Bedingung eintritt, wird der Motor sofort angehalten. Die Werte für `MinLeft` bzw. `MaxRight` errechnen sich über die Formel

Wert = (Max.|Min.)-Spannung * (5V / 1024).

Ist das Flag MT_STOPSOFT gesetzt, so wird mit der durch `AMax` eingestellten Beschleunigung abgebremst.

(Achtung: Falls vor dem Stillstand die Stopp–Bedingung nicht mehr gilt, wird weitergefahren) Wenn MT_STOPSOFT nicht gesetzt ist hält der Motor abrupt an, so dass die genaue Motorposition für weitere Fahrten verloren gehen kann. Deshalb wird zusätzlich eine Referenzfahrt gestartet, es sei denn, das Flag MT_STOPNOREF ist gesetzt. Ist das Flag MT_STOPNULL gesetzt, fungiert der Nullschalter selbst gleichzeitig auch als Endabschalter an seiner Seite.

CMD	$31
P0	`Channel`*: (0...7)*
P1,2 #	`MinLeft`*:* Analogwert Minimum für Stopp links (0=keine Funktion, d.h. linker Endschalter deaktiviert)
P3,4 #	`MaxRight`*:* Analogwert Maximum für Stopp rechts (≥1023=keine Funktion, d.h. rechter Endschalter deaktiviert)

SetOutputs
Befehl zur Konfiguration und setzten der digitalen Ein-/Ausgänge.

PACK_DLL: Parameter TtlOut1: muss TTLOUT1_HIGH, TTLOUT1_LOW oder TTLOUT1_READY sein; Parameter `TtlIo1`: muss `TTLIO1`_HIGH, `TTLIO1`_LOW oder `TTLIO1`_INPUT sein (Konstanten im Headerfile). Die Kombinationen der Parameter des Befehls $32 werden durch diese Funktion entsprechend gebildet.

CMD	$32	
P0	TTLOUT1:	Pegel an TTLOUT1 auf Zusatzschnittstelle setzen (Wert = 0 oder 1)
P1	`TTLIO1_INPUT`:	Input Enable für `TTLIO1` (1=Input, 0=Output)
P2	`TTLIO1`:	Pegel an `TTLIO1` setzen (Wert = 0 oder 1)
P3	`TTLOUT1_READY`:	1 = `TTLOUT1` übernimmt Funktion des Ready-Ausgangs.

SetReadyOutput
Der Ready-Ausgang kann aktiviert werden (low, open collector, mit internem Pullup auf 4,3V), sobald ein Motor aktiv ist (Geschwindigkeit über 0) oder eine Referenzfahrt macht. Der Ready-Ausgang wird innerhalb von 2 ms nach Start/Ende der Motorbewegung geschaltet. Die Wiederholbarkeit (Jitter) entspricht dabei in etwa der Mikroschrittrate bei Start bzw. Stop (siehe Befehl *SetVMinVStart*).

PACK_DLL: Die `Mask`-Variablen werden durch verodern der MOTOR0...MOTOR5 Konstanten gebildet werden.

CMD	$33
P0	`ActiveMask`: Maske für aktive Motoren (Bit 0=Motor 0, Bit 5=Motor 5)
P1	`ReferenceSearchMask`: Maske für Referenzfahrt eines Motors (Bit 0=Motor 0, Bit 5=Motor 5)

4.7 Sonstige Einstellungen

SetPackBaudrate
Neben der Baudratenkonfiguration über die Jumper am Gerät (siehe Kapitel 2.2.3), ist die Einstellung im Betrieb per Software möglich.

PACK_DLL: Die Variable `BaudRate` gibt direkt die `BaudRate` (z.B. 19200) an.

CMD	$40
P0,P1 #	`BaudRate`: Baudratedivisor (16 Bit): Baudratedivisor = 20MHz/ (16*`BaudRate`)
P2,P3 #	`Delay`: Transmitter-Umschaltzeit in Vielfachen von 2ms (1..1000) *(default: 6ms)*

SetPacketTimeout
Beschreibung siehe Kapitel 3.1.6.

CMD	$41
P0,P1 #	`Timeout`: in Vielfachen von 2ms (2..65535)

SetUnitAdress
Beschreibung siehe Kapitel 2.2.1.

CMD	$42
P0	`NewAddress`: Neue RS 232 / RS 485 –Adresse

GetUnitInformation

Dieser Befehl ermöglicht die Abfrage der Firmware-Revision, Reset-Flag, Temperatur und Seriennummer.

CMD	$43
P0	`ResponseAdress`: Adresse des Host zum empfangen der Antwort

Antwort	
CMD	$43
P0	`FirmwareRevision` *:* (z.B. 164=V1.6.4)
P1	RESETFLAG: Ist beim ersten Auslesen 1, hinterher 0
P2	`Temperature`: Temperatur auf dem *PACK* in °C (8 Bit signed)
P3,P4 #	`SerialNumber`: Seriennummer des PACK

SetRs232OverCan

PACK_DLL: Dieser Befehl ist in der Version 1.0 der DLL nicht enthalten.

CMD	$44
P0	Adresse für Antwort
P1	Antwortadresse bei RS 232-Empfang über CAN (obere 8 Bit)
P2	Anzahl sofort weiter zureichender Bytes bei RS232-Empfang (1..8, 0=disabled)

Antwort	
CMD	$44
P0	Anzahl Bytes im RS 232-Sendepuffer (0=leer)
P1	Bit 0: Zustand der CTS-Leitung (invertiert)

SafetyMode
(Steht bei Firmwareversion 1.46 und ab 1.49 zur Verfügung).

PACK_DLL: Dieser Befehl ist in der Version 1.0 der DLL nicht enthalten.

Diese Funktion erlaubt es, sobald die Betriebsspannung unter 13V sinkt, die Motorpositionen im internen EEPROM abzuspeichern, und über einen Befehl wieder zu setzen. Ebenfalls werden die Motoren bei Spannungsabfall abgeschaltet, damit die Versorgungsspannung weniger schnell absinkt. Bei Benutzung dieses Features ist daher im normalen Betrieb sicherzustellen, dass die Versorgung niemals nennenswert unter 15V absinkt.
Über Bit0 gesteuert, werden gültige im EEPROM beim letzten Powerdown gespeicherte Positionen wieder ausgelesen und ins Positionsregister geschrieben. Wenn Bit0 und Bit2 gesetzt sind, bleibt gleichzeitig die Powerdown-Funktion aktiviert. Über Bit1 kann die gespeicherte Position im EEPROM auf ungültig gesetzt werden. Ist Bit2 gelöscht, fahren die Motoren unabhängig vom gerade geltenden Powerdown Level und es wird auch keine Position ins EEPROM abgespeichert. Wird es gesetzt, werden bei (schon bestehender) Unterspannung die Motoren (sofort) abgeschaltet und die Position ins EEPROM geschrieben. Der Powerdown Status wird unabhängig von Bit2 überwacht und mit Bit3 zurückgesetzt. Ebenso ist der Powerdown Status beim Auslesen unabhängig von der Stop-& Save-Aktivierung mit Bit2.

CMD	$45
P0	Adresse für Antwort
P1	• Bit0: 1 = Position aus EEPROM lesen, und falls gültig, setzen • Bit1: 1 = Position im EEPROM als ungültig setzen • Bit2: 1 = Powerdown aktivieren (stoppt Motoren & autosave position) • Bit3: 1 = Powerdown state zurücksetzen (geht nur, wenn die Spannung über Powerdown Level liegt)

Antwort	
CMD	$45
P0	• Bit0: 1 = gültige Position aus EEprom geladen • Bit1: *1* = gültige Position im EEprom gefunden • Bit2: *1* = *Powerdown* – Zustand vor Kommando • Bit3: *1* = *Powerdown* – Zustand nach Kommando

ResetPack
ACHTUNG: Alle eingestellten Parameter gehen verloren und werden durch Defaultwerte ersetzt! Dieses Kommando dauert ungefähr 1 Sekunde.
Anmerkung: Über RS232 wird währenddessen ein 0-Byte empfangen.

CMD	$CC

4.8 Service-Funktionen

Diese Funktionen sind nicht zur Verwendung durch den Benutzer gedacht und können bei unsachgemäßer Verwendung die Einheit unbrauchbar machen.

PACK_DLL: Sämtliche Befehle dieses Abschnitts sind in der Version 1.0 der DLL nicht enthalten.

Flash-Speichers Löschen und Beschreiben ermöglichen

CMD	$F2
P0	Adresse für Antwort
P1,2,3,4 #	Magic Code

Antwort	
CMD	$F2
P0	1=Erase OK

Flash-Speicher programmieren

Diese Funktion wird nur nach einem Löschvorgang ausgeführt. Sie sollte nicht durch andere Funktionen unterbrochen werden, bis der Speicher vollständig programmiert wurde.

CMD	$F3
P0-P6 #	7 Datenbytes

Flash-Speicher Prüfsumme abfragen und ggf. Programmierung beenden

CMD	$F4
P0	Adresse für Antwort

Antwort	
CMD	$F4
P0,P1 #	`Prüfsumme` (Wert hängt von SW-Version ab)

Flash-Speicher auslesen

Vor Erstaufruf dieser Funktion, muss die Prüfsumme abgefragt werden, um den auto-dekrementierenden Adresszähler auf Null zu setzen.

CMD	$F5
P0	Adresse für Antwort

Antwort	
CMD	$F5
P0-P6 #	7 gelesene Datenbytes

EEPROM schreiben

CMD	$F6
P0	Adresse für Antwort
P1,2 #	Magic Code 2
P3,P4 #	Adresse
P5,P6 #	Wert

Antwort	
CMD	$F6
P0	1=Schreiben erfolgreich

EEPROM lesen

CMD	$F7
P0	Adresse für Antwort
P1,2 #	*Magic Code* 2
P3,P4 #	Adresse
P5,P6 #	Wert

Antwort	
CMD	$F7
P0	1=Schreiben erfolgreich

Sternstraße 67
D – 20357 Hamburg, Germany
Phone +49-40-51 48 06 - 0
FAX: +49-40-51 48 06 - 60
http://www.trinamic.com

5 Testberichte

EMV Services GmbH	Test report	Reference	Date	Page
Emission Immunity	No. 00/9091-5	EMV-00/9091-5	May. 09, 00	2 / 26

Customer: Trinamic
Electronic System Design GmbH
Silemstraße 76a
D-20257 Hamburg

Equipment under test: Quadpack Vers.1.0, S/N: Prototype

Date of test: April 13, and May 09, 2000

Test site: EMV Services GmbH
Harburger Schloßstr. 6-12
D-21079 Hamburg

Test personnel:	**Tel.**	**Fax**	**E-mail**
Dipl.-Ing. H. Meisel	040/766293432	040/76629506	meisel@maz-hh.de
Dipl.-Ing. Z. Wang	040/766293431	040/76629506	wang@maz-hh.de

Applied standards:

EN 50 081-2 (1993): **Generic emission standard; Part 2: Industrial environment,**

FCC (1997): **Limit for digital devices, Class A,**

EN 61000-6-2 (2000): **Generic immunity standard, Industrial environment:**

- EN 61000-4-2 (1995): Electrostatic discharge immunity test,
- EN 61000-4-3 (1996): Radiated, radio-frequency electromagnetic field - immunity test,
- EN 61000-4-4 (1995): Electrical fast transient/burst immunity test,
- EN 61000-4-5 (1995): Surge immunity tests,
- EN 61000-4-6 (1996): Immunity to conducted disturbances, induced by radio frequency fields.

Test results:

Emission:
The device complies with the limits for conducted emission.
The device complies with the limits for radiated emission.

Immunity:
The requirements of Immunity:

- To electrostatic discharge are fulfilled,
- To radiated, radio-frequency electromagnetic field are fulfilled,
- To electrical fast transient/burst are fulfilled,
- To surge are fulfilled,
- To conducted disturbances, induced by radio frequency fields are fulfilled.

Dr. E. Sauer
Lab manager

p.p. Dipl.-Ing. Z. Wang

EMV Services GmbH
Ein Unternehmen der TÜV Nord Gruppe
Harburger Schloßstraße 6-12
D-21079 Hamburg

EMV Services GmbH	Test report	Reference	Date	Page
Emission Immunity	No. 00/9091-6	EMV-00/9091-6	Nov. 1, 00	2 / 23

Customer: Trinamic
Electronic System Design GmbH
Silemstraße 76a
D-20257 Hamburg

Equipment under test: Sixpack Rev.1.3, S/N: Prototype

Date of test: October 17, 2000

Test site: EMV Services GmbH
Harburger Schloßstr. 6-12
D-21079 Hamburg

Test personnel:	**Tel.**	**Fax**	**E-mail**
Dipl.-Ing. J. Plambeck	040/766293431	040/76629506	plambeck@emv-services.de

Applied standards:

EN 50 081-2 (1993): **Generic emission standard; Part 2: Industrial environment,**

FCC (1997): **Limit for digital devices, Class A,**

EN 61000-6-2 (2000): **Generic immunity standard, Industrial environment:**

- EN 61000-4-2 (1995): Electrostatic discharge immunity test,
- EN 61000-4-3 (1996): Radiated, radio-frequency electromagnetic field - immunity test,
- EN 61000-4-4 (1995): Electrical fast transient/burst immunity test,
- EN 61000-4-6 (1996): Immunity to conducted disturbances, induced by radio frequency fields.

Test results:

Emission:
The device complies with the limits for conducted emission.
The device complies with the limits for radiated emission.

Immunity:
The requirements of Immunity:

- To electrostatic discharge are fulfilled,
- To radiated, radio-frequency electromagnetic field are fulfilled,
- To electrical fast transient/burst are fulfilled,
- To conducted disturbances, induced by radio frequency fields are fulfilled.

Dr. E. Sauer
Lab manager

p.p. Dipl.-Ing. J. Plambeck

EMV Services GmbH
Ein Unternehmen der TÜV Nord Gruppe
Harburger Schloßstraße 6-12
D-21079 Hamburg