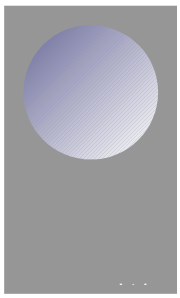
I332

HANDBUCH

ab Version 1.7

Stand 17.05.2008



Fa. Pfortner Ltd.

Krautstückerweg 13
76706 Dettenheim

Tel. +49 7247 906060
Fax +49 7247 906063

wopfo@t-online.de
http://www.pfortner.de

Inhaltsverzeichnis

Versions-Information

***Hinweise**: Wenn die jeweils aktuellen Dateien verwendet werden bzw. die Namen der Konstanten, sollten an bestehenden Programmen keine Änderungen erforderlich sein; ggf. ist eine Neu-Übersetzung notwendig.*

*Die Konstante i332Ver in der Datei I332.DEF sollte dem Highword der von **icGetVerNr** gelieferten Versionsnr. entsprechen; damit ist sichergestellt, daß die definierten Konstanten mit dem aktuellen Befehlssatz der I332 übereinstimmen.*

- automatische Baudraten-Erkennung bei seriellem Betrieb ab Version 1.4
- Version 1.5: neue Befehle
- Version 1.6: neue Befehle
- Version 1.7: Anpassung an die I332-DC2, einige neue Befehle
- Versionen 2.0, 2.1: Spezial-Versionen für bestimmte Anwendungen
- Version 2.2: Befehls-Erweiterungen
- Version 2.3: Spezieller SM-Modus
- Version 2.4: aktuelle ISA-Version
- Version 2.5: serielles ASCII-Interface
- Version 3.x: spezielle Hardware-Version
- Version 4.x: PCI-Version

1 Übersicht

Bei der Interpolatorkarte I332 handelt es sich um eine Schnittstellenkarte zur Steuerung von Motoren jeglicher Art; nicht integriert sind die Leistungsverstärker. Ausgelegt ist die Karte zur Ansteuerung von bis zu 8 Achsen.
Die Platine ist großteils in innovativer SMD-Technik aufgebaut, alle Ein- und Ausgänge sind zur optimalen Störsicherheit optoentkoppelt.

Die Karte enthält einen eigenen Prozessor, der viele Überwachungs- und Steuerungsaufgaben selbständig wahrnimmt (z.B. Endschalter, Geschwindigkeits-Poti, Rampengenerierung etc.). Damit ist der PC von den eigentlichen Steuerungsaufgaben entlastet.

Umfangreiche Routinensammlungen als Pascal-Quell-Code werden ebenso mitgeliefert wie Software zur Inbetriebnahme.

Die I332 steht in 2 Ausführungen zur Verfügung:

In der **ISA-Version** für IBM-PC bzw. Kompatible mit AT-Bus-Schnittstelle belegt sie vier 16-Bit-Portadressen im IO-Bereich, läßt sich aber auch stand-alone über eine integrierte serielle Schnittstelle betreiben.

In der Grundausführung sind bis zu 256-Byte RAM on board installierbar, ausreichend für ca. 1000 Fahrsätze-Sätze.

In der Grundausführung sind Schrittmotor-Treiber direkt anschließbar, als Zusatz-Option sind die Karten I332-DC und I332-IO verfügbar. Die Karte I332-DC stellt für jede Achse eine ±10V-Schnittstelle sowie die Drehgeber-Eingänge (Heidenhain-kompatibel) zum Anschluß von Servo-Verstärkern, die Karte I332-IO 30 Ein- und 30 Ausgabe-Kanäle zur Verfügung.

In der **PCI-Version** erfolgt die Anbindung über einen PCI-Baustein der über den PCI-Konfigurations-Mechanismus in den IO-Adressraum des PC eingeblendet wird. Der Anschluß der Maschine erfolgt über ein Maschinen-Modul, das direkt im Schaltschrank untergebracht werden kann. Die Verbindung geschieht über einen Lichtleiter mit einer Übertragungsrate von 1Mbit/sec. Optional stehen kaskadierbare IO-Moduln mit jeweils 16 Ein-/Ausgängen zur Verfügung, die über CAN-Bus angekoppelt werden. Maximal 4 solcher Moduln (= 64 Ein-/Ausgänge) werden unterstützt.

1.1 Leistungsmerkmale

Anzahl der Achsen: maximal 8

Fahrfunktionen:

- Lineare Positionierung
- Lineare Interpolation bis 8 Achsen gleichzeitig
- Kreis-Interpolation in 2 aus 8 Achsen (bel.), die übrigen Achsen linear zum Kreisbogen (Helix-Interpolation)
- Referenzfahrt komplett implementiert.
- Kontinuierliches Fahren.
- konstante Bahngeschwindigkeit
- Sonderfunktionen unabhängig von der Interpolation

Maschinen-funktionen:

- Unterstützung von Arbeitsspindeln
- frei programmierbare Ein-/Ausgänge (auch DAC-Ausgänge und ADC-Eingänge)
- Satzsynchrones Setzen / Löschen von Ausgängen

Fahrbereich: 32 Bit entsprechend ±2.147.483.647 Incrementen.

Koordinaten-system: Relatives Koordinatensystem (Kettenmaß-Programmierung)

Achsparameter: frei programmierbare Achsparameter, für alle Achsen getrennt:

- Start-, Stop- und Max.-Geschwindigkeit
- Beschleunigung / Verzögerung
- Modulo
- Schritte/mm in beliebigen Verhältnissen
- PID-Lageregel-Parameter für den integrierten Lageregel-Algorithmus (bei Einsatz der I332-DC)

Ausnahme-behandlung:

- Endschalter
- Schleppfehler (bei Einsatz der Zusatz-Karte I332-DC)
- Notaus

sonstiges:

- Satzzykluszeit < 3 ms
- Eigener Satzpuffer ca. 150 Fahr-Sätze, bis ca. 5000 Fahr-Sätze on Board ausbaubar, während der Abarbeitung nachladbar.
- DC-, AC-, Schrittmotor-Betrieb gemischt möglich (nur I332-ISA mit Zusatzkarte I332-DC)

1.2 Technische Daten I332-ISA

1.2.1 I332-1

Diese Platinen-Revision wird nicht mehr hergestellt und vertrieben.

1.2.2 I332-2

Format:	kurze Steckkarte (190mm x 100mm) für IBM-AT-kompatible Rechner
PC-Schnittstelle:	vier 16-BIT(!)-I/O-PORTS im Bereich 314h-317h Auf Wunsch sind andere Adressen durch den Austausch eines ICs möglich.
serielle Schnittstelle:	Stand-Alone-Betrieb: über serielle Schnittstelle (RS232) möglich (Anschlüsse TxD und RxD am SubD-Stecker)
Maschinen-Schnittstelle:	4 Achsen standard / 8 Achsen optional
Ausgänge:	SM - Antriebe: Richtung und Puls, optoentkoppelt Polarität frei wählbar max. Frequenz: achsabhängig, ca. 40 kHz bei 8 Achsen, Softwarebegrenzung bei 64 kHz, änderbar auf Kundenwunsch
Eingänge:	2 Endschaltereingänge pro Achse, optoentkoppelt.
Allg. Eingänge:	8 Analog-Eingänge (1 Poti-Eingang, weitere SW-Unterstützung nach Kundenwunsch) 1 Drehgeber-Eingang, optoentkoppelt (optional) 2 Eingänge, optoentkoppelt, potentialfrei (SW-Unterstützung nach Kundenwunsch) 1 Notaus-Eingang, optoentkoppelt
Steckverbinder:	Sub-D 37-pol: Achsen 0-3 Pfostenleiste 40-pol: Achsen 4-7 (optional) Pfostenleiste 64-pol für Zusatzkarten (optional)
Speicher:	Standardbestückung 64k-Byte, entspr. ca. 150 Fahr-Sätze On-Board ausbaubar bis 256k-Byte/ca. 1000 Fahr-Sätze
Stromversorgung:	+5V ±5% typ. 1,0 A bei PC-Betrieb über PC-Bus bzw. bei Stand-alone-Betrieb am SubD-Stecker extern: +12-24V zur Ansteuerung der Optokoppler optional ±12 V zur Versorgung der RS 422-Schnittstelle
Optionen:	Zusatzkarte I332-DC für DC-Antriebe Zusatzkarte I332-IO mit 30 Ein- und 30 Ausgabe-Kanälen

1.2.3 I332-3

Die I332-3 ist eine Weiterentwicklung der I332-2. Bis auf folgende Details ist sie identisch zur I332-2.

max. Speicher-Ausbau:	max. 1MByte anstatt 256kByte, je nach RAM-Typ

Übersicht

Firmware-Programm-speicher:	wiederbeschreibbares Flash-EPROM anstatt normales EPROM Programmupdate per Download-Software möglich
Allg. Eingänge:	8 Analog-Eingänge 1 Drehgeber-Eingang, optoentkoppelt (optional) 2 Eingänge, optoentkoppelt, potentialfrei, interruptfähig 1 Notaus-Eingang, optoentkoppelt 1 Interrupt-Eingang, optoentkoppelt

1.3 Technische Daten I332-PCI

Format:	kurze Steckkarte (190mm x 100mm) für IBM-AT-kompatible Rechner mit PCI-Bus
PC-Schnittstelle:	PCI-Schnittstellen-Baustein (AMCC)
serielle Schnittstelle:	Stand-Alone-Betrieb: über serielle Schnittstelle (RS232) möglich
Maschinen-Schnittstelle:	Lichtleiter-Ankopplung an ein Maschinen-Modul AM167/168 im Schaltschrank Je nach gewünschter Konfiguration ist ein Maschinen-Modul mit der entsprechenden Hardware notwendig: Anzahl der Achsen, Servo-/Schritt-Motoren (nicht gemischt!), zusätzliche AD/DA-Schnittstellen
Steckverbinder:	Lichtleiter-Anschlüsse Sub-D 9-pol: CAN-Bus, CAN-Open-kompatibel
Firmware-Programm-speicher:	wiederbeschreibbares Flash-EPROM, Programmupdate per Download-Software möglich
Speicher:	Standardbestückung 256k-Byte, entspr. ca. 1000 Fahr-Sätze On-Board ausbaubar bis 512k-Byte/ca. 5000 Fahr-Sätze
Stromversorgung:	+5V ±5% typ. 1,0 A Alle weiteren Spannungen werden über einen DC-DC-Wandler auf der Karte selbst bereitgestellt.

1.4 Zusatzoptionen für die I332-ISA-Version

Die Zusatzkarte wird von der I332-Software voll unterstützt und stellt lediglich die zur Ansteuerung von Servo-Leistungsteilen notwendige Hardware bereit.

1.4.1 I332-DC1

Diese Karte wird nicht mehr hergestellt und vertrieben

1.4.2 I332-DC2

Format:	kurze Steckkarte (190mm x 100mm) für IBM-AT-kompatible Rechner
PC-Schnittstelle:	nur Stromversorgung
I332-Schnittstelle:	62-pol Steckerleiste
Maschinen-Schnittstelle:	4 Achsen standard / 8 Achsen optional, je Achse: Eingänge: • 4 Drehgeber-Eingänge, optoentkoppelt Quadratur-Encoder mit 2 gegenphasigen Signalpaaren • Referenzpuls-Eingang, optoentkoppelt • Bereitschafts-Eingang vom Servo-Verstärker, optoentkoppelt Ausgänge: • Analog-Ausgang ±10V zur Ansteuerung von Leistungsteilen Auflösung 12 Bit, optoentkoppelt • Freigabe-Ausgang zum Servo-Verstärker, optoentkoppelt
Allg. Ausgänge:	keine
Steckverbinder:	Sub-D 37-pol: Achsen 0-3 Pfostenleiste 40-pol: Achsen 4-7 (optional) Pfostenleiste 64-pol: Anschluß I332
Stromversorgung:	+5V ±5% typ. 1.0A ±12 V zur Erzeugung der Analog-Spannung werden intern erzeugt (galvanisch getrennter DC-DC-Wandler). extern: +12-30V zur Versorgung der Optokoppler

1.4.3 Option Zusatz-Karte Ein-/Ausgänge: I332-IO

Für Ein-/Ausgabe-intensive Anwendungen steht die Zusatz-Karte I332-IO zur Verfügung, die von der Software ab Version 1.6 ebenfalls unterstützt wird.

Format:	kurze Steckkarte (190mm x 100mm) für IBM-AT-kompatible Rechner
PC-Schnittstelle:	nur Stromversorgung
I332-Schnittstelle:	64-pol Steckerleiste
Ausgänge:	30 Ausgänge, frei programmierbar, opto-entkoppelt
Eingänge	30 Eingänge, TTL-Pegel

Übersicht

Steckverbinder:	Sub-D 64-pol
Stromversorgung:	+5V ±5% typ. 1.0A bei PC-Betrieb über PC-Bus (5V) bzw. bei Stand-alone-Betrieb an der PC-Bus-Leiste

1.5 Übersicht: Zusatz-Hardware für die ISA-Version

Dieses Kapitel bietet lediglich eine Übersicht über lieferbare Zusatz-Hardware. Bei Bedarf fordern Sie bitte detaillierte Informationen an.

1.5.1 I332-MOT-AM

Motor-Anschaltmodul für Servomotoren an der I332-ISA / I332-DC-Karte

- bis zu 4 Achsen, für 5-8 Achsen sind zwei Module notwendig
- Anschlüsse achsweise auf Schraubklemmen
- Freigabe-Ausgänge optoentkoppelt (für I332-DC1)

1.5.2 I332-IO-AM

Anschaltmodul für die I332-IO-Karte:

- 30 optoentkoppelte Eingänge
- 30 mit Schaltrelais (24V) bestückte Ausgänge
- Anschlußmöglichkiet für die Handbedienbox
- durchgeschleifter Notaus-Kreis
- Alle Anschlüsse an Schraubklemmen

1.5.3 I332-UAM

Universelles Anschaltmodul für Servomotoren an der I332-ISA / I332-DC / I332-IO-Karte

- bis zu 6 Achsen
- Anschlüsse achsweise an SubD-Steckern
- Universelle Eingangsbeschaltung der Endschalter für 5 bis 30V
- Eingänge der I332-IO optoentkoppelt, Universalschaltung für 5 bis 30V
- Transistor-Ausgangstreiber für Relais an den Ausgängen der I332-IO
- Handbedienbox anschließbar
- Notauskreis durchgeschleift
- Weidmüller-kompatible Steckverbinder

1.6 Übersicht: Zusatz-Hardware für die PCI-Version

1.6.1 AM167/AM168-Anschaltmodule

Anschaltmodul für bis zu 8 Servo- oder Schrittmotoren

- Anschlüsse achsweise an SubD-Steckern
- Universelle Eingangsbeschaltung der Endschalter für 5 bis 30V
- Lichtleiteranschluß für I332-PCI

1.6.2 CAN-IO-Modul

CAN-Bus - Ein-/Ausgabe-Modul

- 16 Eingänge, optoentkoppelt, 5-24 Volt
- 16 Ausgänge, optoentkoppelt
- Ansteuerung über CAN-Bus
- 24V-Anschluß
- Anschlüsse mit Schraubklemmen
- Direkte Ansteuerung / Unterstützung durch die I332-PCI-Karte
- Auch unabhängig von der I332-PCI-Karte universell verwendbar

Sonderausführungen (DA-/AD-Wandler, Drehgeber-Eingänge u.a.) auf Anfrage.

1.7 Weitere Hardware

1.7.1 Hand-Bedienbox

Handbedien-Einheit für bis zu 6 Achsen mit Handrad bzw. Geschwindigkeitspoti und Notaus-Sicherheitstasten

1.7.2 LOGIPOS

Kleinsteuerung auf Basis der I332 / I332-DC für bis zu 8 Schrittmotor- oder Servo-Achsen, programmierbar nach DIN 66025.

1.7.3 CAN-PCI

PCI-Steckkarte zur Ansteuerung der CAN-IO-Module bzw. beliebiger CAN-Module von Drittanbietern, unabhängig von der I332-PCI-Karte.

1.8 PC-Software

Dieses Kapitel wird z.Zt. überarbeitet

2 Inbetriebnahme

Dieses Kapitel wird z.Zt. überarbeitet; zur Inbetriebnahme unter Windows 9x/ME/NT/2k/XP wird auf das Handbuch I332_DLL.PDF verwiesen.

2.1 ISA-Version

2.2 serielle Version

2.3 PCI-Version

2.4 USB-Version

3 Programmierschnittstelle der I332

Die folgende Übersicht beschreibt die Programmierung der I332.

Abkürzungen:

~	logisches Komplement (NOT)
^	Potenzieren (wie in BASIC)
AC	alternated current (Wechselstrom)
ADC	Analog-Digital-Wandler
DAC	Digital-Analog-Wandler
DC	direct current (Gleichstrom)
dir	direction
ccw	counter clockwise
CP	center point
cw	clockwise
ES	Endschalter
FE	following error (Schleppfehler)
GZS	Gegen-Uhrzeigersinn
IPL	Interpolation
LS	limit switch
LSB	least signifikant byte/bit (niederwertiges Byte oder Bit)
MP	Mittelpunkt
MSB	most signifikant byte/bit (höherwertiges Byte oder Bit)
PID-	Proportional / Integral / Differential - (Algorithmus / Regler /...)
RefS	Referenzschalter
Rtg	Richtung
SM	Schrittmotor
SV	Servo...
UZS	Uhrzeigersinn
VZ	Vorzeichen

Anmerkungen: *Für die Achsen 0..7 werden gelegentlich die Bezeichnungen XYZUVABC verwendet (s. Steckerbelegung); dies jedoch nur zur besseren Verständlichkeit.*

Die Befehle lassen sich nach verschiedenen Gesichtspunkten klassifizieren:

a) Datenformat: Schnittstelle zur I332 - Kap. 3.1
b) numerische Reihenfolge - Kap. 3.2, Datei I332.DEF
c) systematische Reihenfolge - Kap. 4, Befehlsgruppen nach Abb. 3.1

Die Programmierung der I332 wurde mit Blick auf die DIN 66 025 gestaltet. Aus diesem Grund wird gelegentlich auf diese Norm Bezug genommen. Ebenso orientiert sich die Verwendung der Termini an dieser Norm. Zum Verständnis des folgenden sollte die Kenntnis der Norm aber nicht notwendig sein.
Die Notation von Routinen oder Programmstücken folgt den Konventionen der Programmiersprache PASCAL. Die Beschreibung erfolgt nach folgendem Schema:

```
FUNCTION <Name> (<Parameter>): <Funktionsergebnis>;
```

Beschreibung	Beschreibung der Routine
Parameter	Beschreibung der Eingabeparameter
	Beschreibung der Ausgabeparameter
Funktionsergebnis	Beschreibung des Funktionsergebnis
Beispiel-Aufruf	(nicht bei allen Routinen)
Anmerkungen	(nicht bei allen Routinen)

3.1 Schnittstelle PC ←→ I332

Unter den 32-Bit-Windows-Versionen steht die I332.DLL zur Kommunikation zur Verfügung; im Rahmen dieses Handbuchs werden nur die Basis-Routinen

- `I332IFCInit`
- `I332IFCDone`
- `I332Reset`
- `I332Restart`
- `I332Write0`
- `I332Write1`
- `I332Write2`
- `I332Read1`
- `I332Read2`

benötigt. Diese und alle weiteren Funktionen sind im Handbuch I332_DLL.PDF beschrieben und in der Datei I332_IFC.PAS deklariert.

***Anmerkung** zu den folgenden Routinen:*

- *`icGet` bzw. `icSet` meint eine der in Kap. 4 beschriebenen Konstanten `ic...`*
- *Bis auf die Funktion `I332IFCInit` liefern alle Funktionen ggf. eine (negative) Fehlernummer oder den Wert 0 zurück*
- *die weitern hier verwendeten Konstantennamen (`imXXX, ieXXX ...`) sind im Handbuch I332_DLL.PDF beschrieben bzw. in der Datei I332_IFC.PAS deklariert*

3.1.1 I332IFCInit

```
FUNCTION I332IFCInit(ifcMode: WORD; ifcPara: LONGINT): INTEGER;
```

Beschreibung Diese Funktion initialisiert die Schnittstelle zur I332 und versucht eine Verbindung aufzubauen. Als Ergebnis wird ein Handle zurückgeliefert, das die I332-Karte identifiziert und bei allen anderen Funktionen der DLL als Parameter erwartet wird.

Parameter

`ifcMode` Schnittstelle: eine der Konstanten `imXXX` wie in I332_DLL.PDF beschrieben

`ifcPara` je nach Schnittstelle sind folgende Parameter erforderlich:

`imISA`	kein Parameter erforderlich, es sollte ifcPara = 0 übergeben werden
`imPCI`	`ifcPara` = lfd. Nummer der PCI-Karte, wenn mehrere I332-PCI im System vorhanden sind; `ifcPara` = 0: 1. Karte im System etc.
`imCOMx`	`ifcPara` = Baudrate; zulässig sind die Werte 9600, 19200, 38400 und 57600 `ifcPara` = 0: Standardbaudrate (57600)
`imHEXx`	`ifcPara` = Baudrate: wird (noch) nicht ausgewertet `ifcPara` = 0: Standardbaudrate (9600)

Funktionsergebnis Handle auf die I332-Schnittstelle bzw. Schnittstellen-Fehlercode `ieXXX`

***Hinweis**:* *Solange keine speziellen Anforderungen gestellt werden, kann immer `ifcPara` = 0 übergeben werden.*

3.1.2 I332IFCDone

```
FUNCTION I332IFCDone(I332Handle: INTEGER): INTEGER;
```

Beschreibung Beendet die Kommunikation mit der I332 und gibt die Schnittstelle wieder frei.

Parameter
`I332Handle` Handle auf die I332-Schnittstelle

Funktionsergebnis Schnittstellen-Fehlercode `ieXXX`

3.1.3 I332Reset

```
FUNCTION I332Reset(I332Handle: INTEGER): INTEGER;
```

Beschreibung Initialisiert die I332

Parameter
`I332Handle` Handle auf die I332-Schnittstelle

Funktionsergebnis Schnittstellen-Fehlercode `ieXXX`

3.1.4 I332Restart

```
FUNCTION I332Restart(I332Handle: INTEGER; icRestart: WORD): INTEGER;
```

Beschreibung Initialisiert die I332

Parameter
`I332Handle` Handle auf die I332-Schnittstelle
`icRestart` Ein Restart-Befehl (s. Kap. 4.1)

Funktionsergebnis Schnittstellen-Fehlercode `ieXXX`

***Hinweis**:* *Solange ein Restart-Befehl abgearbeitet wird (je nach Restart-Befehl und Hardware-Version bis zu 250 ms), nimmt die I332 keine weiteren Befehle an; die Funktion `I332Restart` trägt diesem Umstand Rechnung. Restart-Befehle sollten deshalb nur über diese Funktion, nicht über `I332Write0` aufgerufen werden.*

3.1.5 I332Write0

```
FUNCTION I332Write0(I332Handle: INTEGER; icSet: WORD): INTEGER;
```

Beschreibung Überträgt das Befehlswort `icSet` an die I332

Parameter
`I332Handle` Handle auf die I332-Schnittstelle
`icSet` Befehlswort vom Typ W0

Funktionsergebnis Schnittstellen-Fehlercode `ieXXX`

***Hinweis**:* *Restart-Befehle sollten über die Funktion `I332Restart`, nicht über `I332Write0` aufgerufen werden.*

3.1.6 I332Write1

```
FUNCTION I332Write1(I332Handle: INTEGER; icSet: WORD; Value: WORD): INTEGER;
```

Beschreibung Schreiben des Registers `icSet`: Überträgt das Befehlswort `icSet` und den dazugehörigen Wortparameter `Value` an die I332

Parameter

`I332Handle`	Handle auf die I332-Schnittstelle
`icSet`	Befehlswort vom Typ W1
`Value`	Befehlsparameter

Funktionsergebnis Schnittstellen-Fehlercode `ieXXX`

3.1.7 I332Write2

```
FUNCTION I332Write2(I332Handle: INTEGER; icSet: WORD; Value: LONGINT): INTEGER;
```

Beschreibung Schreiben des Registers `icSet`: Überträgt das Befehlswort `icSet` und den dazugehörigen Langwortparameter `Value` an die I332

Parameter

`I332Handle`	Handle auf die I332-Schnittstelle
`icSet`	Befehlswort vom Typ W1
`Value`	Befehlsparameter

Funktionsergebnis Schnittstellen-Fehlercode `ieXXX`

3.1.8 I332Read1

```
FUNCTION I332Read1(I332Handle: INTEGER; icGet: WORD; VAR Value: WORD): INTEGER;
```

Beschreibung Lesen des Registers `icGet`: Überträgt das Befehlswort `icGet` und liefert den Inhalt des Registers im Wortparameter `Value` zurück

Parameter

`I332Handle`	Handle auf die I332-Schnittstelle
`icGet`	Befehlswort vom Typ R1
`Value`	Ergebniswort von der I332

Funktionsergebnis Schnittstellen-Fehlercode `ieXXX`

3.1.9 I332Read2

```
FUNCTION I332Read2(I332Handle: INTEGER; icGet: WORD; VAR Value: LONGINT): INTEGER;
```

Beschreibung Lesen des Registers `icGet`: Überträgt das Befehlswort `icGet` und liefert den Inhalt des Registers im Langwortparameter `Value` zurück

Parameter

`I332Handle`	Handle auf die I332-Schnittstelle
`icGet`	Befehlswort vom Typ R2
`Value`	Ergebniswort von der I332

Funktionsergebnis Schnittstellen-Fehlercode `ieXXX`

3.2 Die Befehlsgruppen

Nach außen stellt sich die Interpolator-Karte als sehr komplexer IO-Baustein dar, der mit Hilfe einer Vielzahl von Registern programmierbar ist. Das Blockschaubild in Abb. 3.1 vermittelt einen ersten Überblick und ist gleichzeitig Grundlage für die Beschreibung der Befehle in den folgenden Kapiteln.

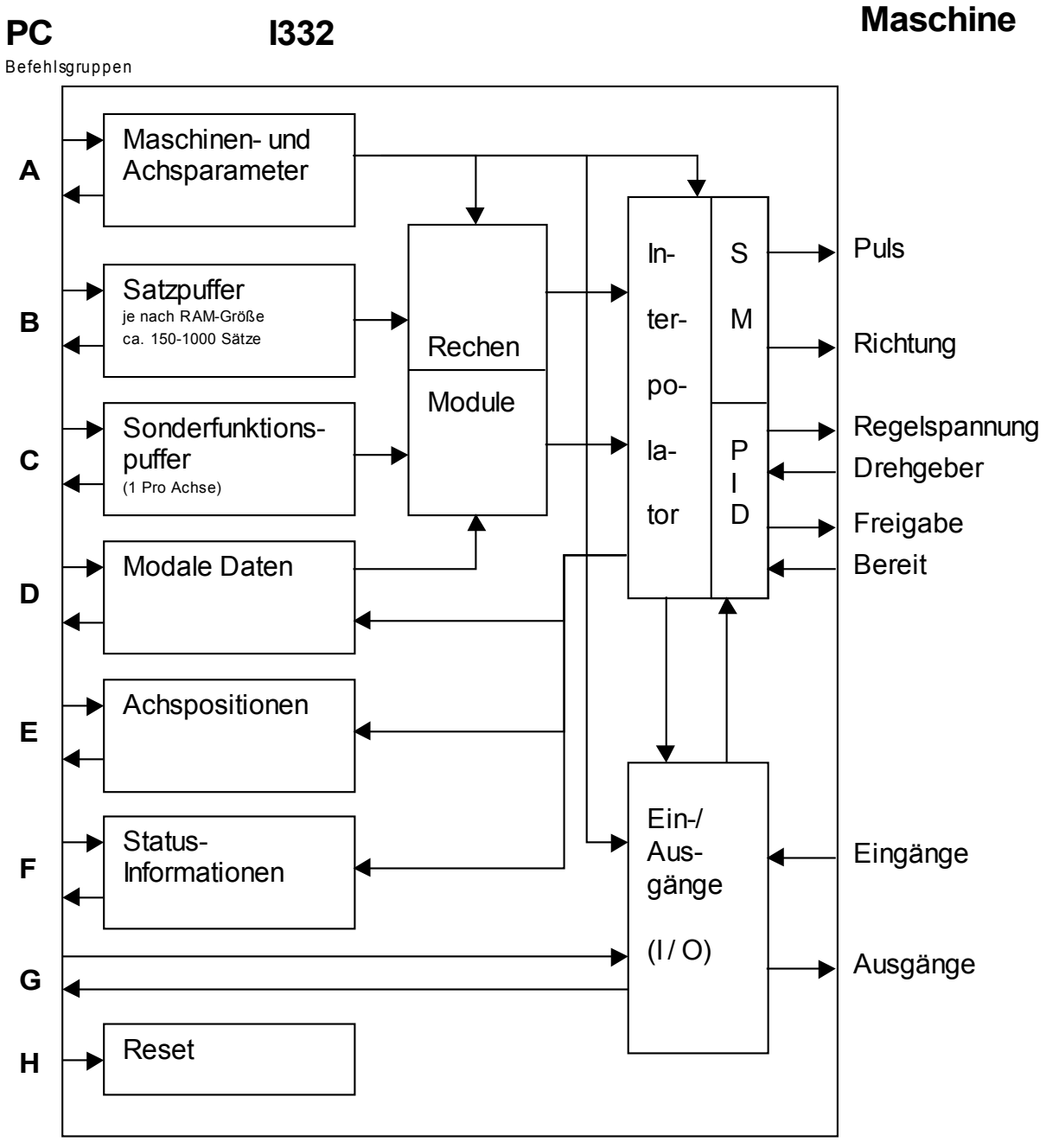


Abb. 3.1 Funktions-Blockschaubild der I332 (vereinfacht)

Programmierschnittstelle der I332

Gruppe A: Diese Befehle sprechen Register an, die die Eigenschaften der Maschine und der Achsen sowie die Anfangswerte für bestimmte Register festlegen.

Gruppe B: Diese Befehle schreiben und lesen die Register des aktuellen Satzes im Satzpuffer. Das Weiterschalten erfolgt durch einen Interpolationsbefehl. Dabei werden gleichzeitig die Modalen Daten (Gruppe D) in diesen Satz übernommen.

Gruppe C: Diese Befehle schreiben und lesen die Register im Puffer für die Sonderfunktionen

Gruppe D: Diese Befehle schreiben und lesen modale Register. Die Werte in diesen Registern bleiben erhalten, bis sie überschrieben werden. Beim Weiterschalten des Interpolationssatzes werden sie in den Satzpuffer übernommen.

Gruppe E: Diese Befehle schreiben und lesen die aktuelle Position und Geschwindigkeit der Achsen

Gruppe F: Diese Befehle schreiben und lesen interne Zustände

Gruppe G: Diese Befehle greifen auf Eingänge und Ausgänge zu und dienen auch zur Programmierung allgemeiner Steuerfunktionen mit Hilfe der IO-Bausteine.

Gruppe H: Diese Befehle setzen das System bzw. Teile davon zurück in einen Grundzustand.

***Anmerkungen**:*

- *Der Funktionsblock SM innerhalb des Interpolators befindet sich physikalisch auf der I332 (ISA-Version) bzw. auf dem AM167/AM168-Modul (PCI-Version).*
- *Der Funktionsblock PID innerhalb des Interpolators befindet sich physikalisch auf der Zusatzkarte I332-DC (ISA-Version) bzw. auf dem AM167/AM168-Modul (PCI-Version).*
- *Der Funktionsblock I/O ist physikalisch auf alle Karten im System aufgeteilt (s. a. Kap. 1):*
  Endschalter-Eingänge: I332-ISA bzw. AM167/AM168-Schraubklemmen
  ADC-Eingänge: I332-ISA bzw. AM167/AM168-Schraubklemmen
  Referenz-Eingänge: I332-DC bzw. AM167/AM168-Achsmodul
  Freigabe(-Enable)-Ausgänge: I332-DC bzw. AM167/AM168-Achsmodul
  Bereit(-Ready)-Eingänge: I332-DC bzw. AM167/AM168-Achsmodul
  IN01-IN30 / OUT01-OUT30: I332-IO bzw. CAN-IO-Modul

3.3 Befehls-Übersicht

Alle hier beschriebenen Befehle für die I332 sind in der Datei I332.DEF in Form von Konstanten verfügbar und kommentiert.

Andere als die hier definierten Befehle sollten nicht verwendet werden, da sie teilweise internen Zwecken dienen bzw. unvorhersehbare Reaktionen verursachen können!

3.3.1 Namenskonventionen

- alle Befehle beginnen mit dem Kürzel **ic..**
- Lese- bzw. Schreib-Befehle heißen **icGetXXX** bzw. **icSetXXX**, wenn sie das Register **XXX** ansprechen.
- einige Befehle erfordern zusätzliche Angaben im Befehlswort selbst, d.h. der Befehl stellt nur das Basiswort da, erforderliche weitere Angaben werden dazuaddiert. Die entsprechenden Konstanten sind anschließend an das jeweilige Befehlswort definiert und beschrieben. Diese Konstanten beginnen mit einem Kürzel, das sich am Basisbefehl orientiert; z.B. beginnen alle Bezeichner für die Achsparameter, die mit den Befehlen **icSetAP/icGetAP** geschrieben bzw. gelesen werden mit dem Kürzel **ap..**
- Konstanten, die einzelne Bits bezeichnen, enden mit ...B, zusätzlich sind zu allen Bitkonstanten auch Bit-Flags definiert (...F) in denen das entsprechende Bit gesetzt ist.
  z.B. **stErrB** = 1;
  stErrF = $2^{\mathbf{stErrB}}$ (= $0002: Bit 1 = **stErrB** ist gesetzt)
  Auch für die Bitkonstanten gilt eine entsprechende Namenskonvention, z.B. beginnen alle Bezeichner für die Statusbits mit dem Kürzel **st..**
- die Definitionen basieren auf englischen Bezeichnungen, die Datei I332.DEF ist englisch kommentiert. Sollten Unterschiede zu diesem Handbuch auftreten (nobody is perfect), so gilt die Datei I332.DEF.

3.3.2 Befehls-Parameter

Die Kennzeichnungen R1 / R2 bzw. W0 / W1 / W2 beziehen sich auf die Größe der zusätzlichen Daten. Langwort-Daten (2 Worte) sind immer in der Reihenfolge Lowword/Highword.

R1 / R2:	Lese-Befehl - es werden 1 bzw. 2 Worte von der I332 zurückgeliefert
W0 / W1 / W2:	Schreib-Befehl - die I332 erwartet keine bzw. 1 oder 2 weitere Datenworte

Die folgenden Kapitel enthalten eine Beschreibung der implementierten Befehle nach folgendem Schema:

ic.... **<Wert>** **<Typ> / <Gruppe>**
Beschreibung des Befehls

ic....	Name des Befehles
<Wert>	ist die numerische Darstellung in Hexadezimal-Schreibweise (im Turbo-PASCAL-Format: $XXXX, im C-Format entspräche dies der Schreibweise 0xXXXX, in BASIC &HXXXX)
<Typ >	ist der Befehlstyp R1 / R2 / W0 / W1 / W2 wie oben beschrieben
<Gruppe>	Befehlsgruppe nach Abb. 3.1

In den Befehlen für Register, die gelesen und geschrieben werden können, kennzeichnet das Bit 7 im Befehl in der Regel die Art des Zugriffs:

Bit 7 = 0 - Schreibzugriff
Bit 7 = 1 - Lesezugriff

Die beiden folgenden Kapitel beschreiben zwei Konventionen bei der Definition von Befehlen.

3.3.3 Achsregister

Die Befehle $0x00...$7xFF beziehen sich auf Register einzelner Achsen, wobei »x« die Achsnr. in den Bits 8..10 darstellt, d.h. die Achsnr. muß auf das Highbyte des Befehls addiert werden, um das gewünschte Register anzusprechen. Achsregister sind i.d.R. Langwort-Register, d.h. die zugehörigen Befehle sind vom Typ R2 bzw. W2.

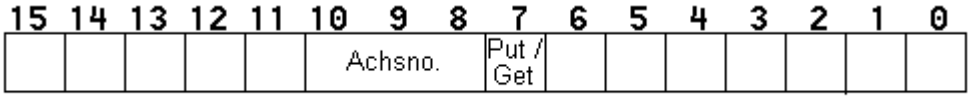


Abb. 3.2 Aufbau eines Achsregister-Befehls

Die folgenden Funktionen `I332SetAchsReg`, `I332GetAchsReg` und `I332AchsReg` erleichtern den Umgang mit diesen Registern. Details dazu s. I332_DLL.PDF.

3.3.4 Bitregister

Ein Bitregister ist ein Langwort-Register das sich in seiner Gesamtheit (32 Bit) manipulieren läßt. Zusätzlich besteht die Möglichkeit nur das Low- oder High-Wort anzusprechen oder einzelne Bits zu manipulieren. Je nach Art des Zugriffs wird eine der Konstanten brMode für die Bits 0..7 auf das Lowbyte des Befehls-Wortes addiert und bestimmt dadurch den Typ des Befehls (W0 / W1 / W2 / R1 / R2). Abb. 3.3 zeigt den Aufbau des Lowbytes in einem Bitregister-Befehl:

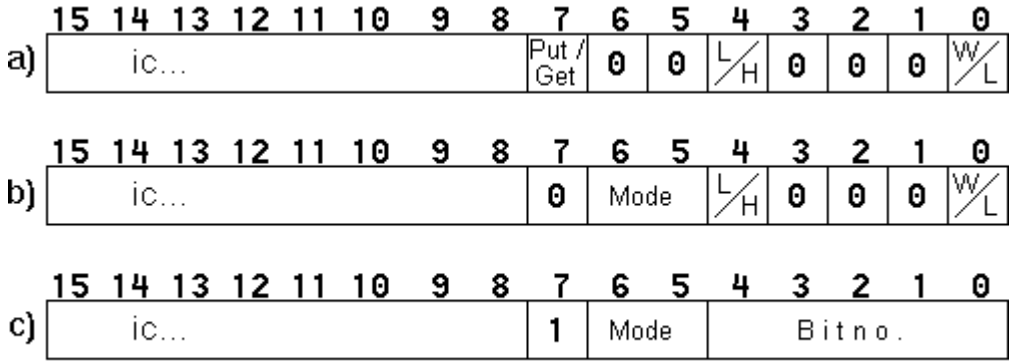


Abb. 3.3 Aufbau eines Bitregister-Befehls

Put / Get	=	0 / 1	Register schreiben / lesen
Mode	=	(00	Put bzw. Get)
		01	Clr
		10	Set
		11	Chg
L / H	=	0 / 1	Low- / Highwort des Registers
W / L	=	0 / 1	Wort / Langwort

folgende Kombinationen sind zulässig:

L/H	W/L	
0	0	Lowwort des Registers (16 Lowbits)
1	0	Highwort des Registers (16 Highbits)
0	1	ganzes Register (Langwort: 32 Bit)
1	1	nicht zuässig

Bitno	=	0..31	Bitnummer

Damit ergeben sich folgende Konstanten zur Ergänzung eines Registerbefehls:

a) Lesen / Schreiben des Registers

brModePutL	**$0000**	W1 - Lowwort des Registers schreiben
brModePutH	**$0010**	W1 - Higwort des Registers schreiben
brModePut		**$0001** W2 - High- und Lowwort des Registers schreiben
brModeGetL	**$0080**	R1 - Lowwort des Registers lesen
brModeGetH	**$0090**	R1 - Highwort des Registers lesen
brModeGet		**$0081** R2 - High- und Lowwort des Registers lesen

b) Bits mit einem Datenwort maskieren

alle Bits im Bitregister löschen, die im Datenwort = 0 sind:

brModeClrL	**$0020**	W1 - Bits im Lowwort löschen
brModeClrH	**$0030**	W1 - Bits im Highwort löschen
brModeClr	**$0021**	W2 - Bits im High- und Lowwort löschen

alle Bits im Bitregister setzen, die im Datenwort = 1 sind:

brModeSetL	**$0040**	W1 - Bits im Lowwort setzen
brModeSetH	**$0050**	W1 - Bits im Highwort setzen
brModeSet	**$0041**	W2 - Bits im High- und Lowwort setzen

alle Bits im Bitregister invertieren, die im Datenwort = 1 sind:

brModeChgL	**$0060**	W1 - Bits im Lowwort invertieren
brModeChgH	**$0070**	W1 - Bits im Highwort invertieren
brModeChg	**$0061**	W2 - Bits im High- und Lowwort invertieren

c) Einzelne Bits manipulieren

Die Bitnr. muß jeweils noch dazuaddiert werden (Bits 0..4 im Befehlswort), für die Bits 0..15 wird das Lowword, für Bits 16..31 das Highword des Registers zurückgeliefert

brModeClrB	**$00A0**	R1 - einzelnes Bit löschen
brModeSetB	**$00C0**	R1 - einzelnes Bit setzen
brModeChgB	**$00E0**	R1 - einzelnes Bit invertieren

3.3.5 Hinweise

- Es sollten immer die definierten Konstanten, nie die numerischen Werte selbst benutzt werden, um evtl. notwendige Änderungen auf einfache Weise durchführen zu können (s.a. Kap. 4).

- Falls nicht anders angegeben werden Distanzen in µm, Geschwindigkeiten in mm/min und Beschleunigungen/Verzögerungen in mm/sec² programmiert. Für Rundachsen sind die Werte als 1/1000 Grad, Grad/min bzw. Grad/sec² zu interpretieren.

- Einige Befehle sind nur sinnvoll / wirksam, wenn eine I332-DC-Karte bzw. die PCI-Version mit Servo-Modul oder eine I332-IO-Karte bzw. ein CAN-IO-Modul vorhanden sind (z.B. die PID-Parameter, Portzugriffe)!

4 Befehlsbeschreibung

Dieses Kapitel enthält eine systematische Beschreibung der implementierten Befehle und orientiert sich dabei an der Einteilung nach Befehlsgruppen (s. Abb. 3.1). In der Datei I332.DEF sind die Befehle i.W. in numerischer Reihenfolge gelistet. **Bei Druckfehlern bzw. Unterschieden, auch in dieser Beschreibung, gilt die Definition in der Datei I332.DEF.**

4.1 Reset-Befehle - Gruppe H

Die Befehlstabelle stellt mehrere Reset-Befehle zur Verfügung, die unterschiedliche Teile des I332-Systems neu initialisieren.

icRestart0 $FF00 W0 / H
(fast) kompletter Neustart der System-Software. Die Maschinendaten und Achsparameter werden auf ihre Defaultwerte gesetzt(!) und interne Variablen initialisert. Insbesondere wird auch die Baudrate zurückgesetzt. Auf seriellen Systemen hat dies zur Folge, daß die serielle Schnittstelle ab Version 1.4 neu synchronisiert werden muß (wegen der automatischen Baudraten-Erkennung)!!

icRestart1 $FF01 W0 / H
Löschen des Satzpuffers und der Positions-Register und aller internen Variablen, außer den Maschinendaten und Achsparametern, Neuinitialisieren des PID-Reglers (ein etwaiger Schleppfehler wird auf Null gesetzt, was sich u.U. durch eine Motorbewegung bemerkbar macht), insbesondere ist ein Neu-Referenzieren aller Achsen erforderlich. Alle Ausgänge werden auf die in den Maschinendaten programmierten Werte gesetzt (s. **mpXXOutDef**, **mpXXOutPAR**, bzw. **mpSP_....**, Kap. 4.2.1 und 4.8.2). Das Poti wird entsprechend dem Parameter **mpPotiInit** neu initialisiert. Die Satzpuffer für Interpolation und Sonderfunktionen werden gelöscht, die Statusbits **stSSB** und **stSFB** werden zurückgesetzt.

icRestart2 $FF02 W0 / H
wie **icRestart1**, Positionswerte bleiben erhalten; wegen der Neu-Initialiserung des PID-Reglers, geht jedoch die Referenzposition für die Servo-Achsen verloren.

icRestart3 $FF03 W0 / H
wie **icRestart2**, der PID-Regler wird nicht initialisiert, d.h. Servo-Achsen bleiben in der Regelung, die Referenzpositionen bleiben erhalten. Dieser Restart-Befehl sollte bevorzugt verwendet werden.

icRestart5 $FF05 W0 / H
wie **icRestart3**, der PID-Regler wird nicht initialisiert, d.h. Servo-Achsen bleiben in der Regelung, die Referenzpositionen bleiben erhalten, der Schleppfehler wird jedoch gelöscht.

icRestartFE $FF0A W0 / H
Schleppfehler löschen, sollte nur im Stillstand und bei leerem Interpolations- und Sonderfunktionsbuffer erfolgen!

icRestartPort $FF0C W0 / H
Ausgänge neu setzen entspr. Maschinenparameter **mpXXOutDef**.

icRestartSer $FF04 W0 / H
Dieser Befehl setzt nur die serielle Schnittstelle zurück; anschließend ist eine Neu-Synchronisation notwendig.

Ab Version 2.2:

icRestart11 $FF11 W0 / H
icRestart12 $FF12 W0 / H
icRestart13 $FF13 W0 / H
Diese 3 Restart-Befehle sind fast identisch zu den Befehlen **icRestart1**, **icRestart5**(!) und **icRestart3** mit dem Unterschied, daß zwar eine Neutinitialisierung der Register **MOutBuf** und **OutPAR** aus den entsprechenden Maschninendaten **mpXXXOutDef** bzw. **mpXXXOutPAR** aber keine Neuintialisierung des Registers **OutBuf** erfolgt, d.h. die als satzsynchron definierten Ausgänge werden auf ihren Defaultwert gesetzt, alle anderen bleiben unverändert. Details s.a. Kap. 4.2.1 und 4.8.2.
In allen Fällen wird die Ausgabe an die Motoren abgebrochen. Restartbefehle sollten deshalb nur bei Stillstand der Motoren eingesetzt werden.

icRestart14 $FF14 W0 / H
Abbruch aller Sonderfunktionen, Löschen des Interpolationsbuffers (Satz-Abbruch).

Befehlsbeschreibung

icRestart15 $FF15 W0 / H
wie **Restart14**, zusätzlich Schleppfehler löschen.

Zu den Restart-Befehlen sind auch die Befehle **icResetMP**, **icInitMP** und **icAPEnd** zu rechnen, die die Maschinendaten bzw. Achsparameter neu initialisieren. Sie werden im Kap. 4.2 erläutert.

***Hinweise**:* *Fahrfunktionen lassen sich auch ausführen, wenn die Achsen nicht referenziert sind (z.B. zum Freifahren von Endschaltern nach dem Einschalten o.ä.); dies ist besonders bei Restart-Befehlen zu beachten, bei denen die Referenzposition verloren geht! Informationen zum Referenzstatus der einzelnen Achsen liefert der Befehl **icGetRefInfo** (s. Kap. 4.7.2)*

Restart-Befehle werden quittiert noch bevor sie komplett abgearbeitet sind; während des Abarbeitens (je nach Restartbefehl und Hardware-Version bis zu 250ms) eines Restart-Befehls nimmt die I332 jedoch keine weiteren Befehle an. Dies kann dazu führen, daß der nachfolgende Befehl erst spät quittiert wird. Einige Restart-Befehle versuchen außerdem, die Achsen noch ordnungsgemäß über die Rampen zu stoppen, was zu weiteren Verzögerungen führen kann. Dies hat zur Folge, daß Aufrufe der DLL-Routinen u.U. mit einem Timeout-Fehler abbrechen. s.a. DLL-Funktion `I332Restart`, Kap. 3.1.4!

4.1.1 NOTSTOP-Routine

Diese Routine wird ausgelöst bei Aktivieren des Notaus-Eingangs (nur I332-ISA). Ab EPROM-Version 1.6 setzt die Routine lediglich die beiden Statusbits **stSSB** und **stSFB** zurück und stoppt so alle Achsbewegungen.
Ab Version 1.7 steht das Maschinendatum **mpEmStop** (s.u.) zur Verfügung.

4.2 Maschinendaten - Gruppe A

4.2.1 Globale Maschinendaten

4.2.1.1 Einleitung, Initialisierung

Zur allgemeinen Konfiguration der Maschine stehen (ab Version 1.6) eine Reihe von Parameter-Registern zur Verfügung. Einige der Register lassen sich nur lesen; diese enthalten Systeminformationen, die ein PC-Programm entsprechend auswerten kann.
Alle Register haben Langwort-Format, einige sind jedoch in einzelne Worte oder Bytes unterteilt. Es muß aber immer das ganze Register gelesen bzw. geschrieben werden!
Da diese Daten das grundlegende Verhalten der Maschine beeinflussen, ist NACH dem Ändern dieser Register ein spezieller Restart-Befehl notwendig:

icInitMP **$FF08** **W0 / H**
Der Befehl bewirkt wie **icRestart0** eine Neuinitialisierung des ganzen Systems. Die Maschinen- und Achsparameter werden jedoch nicht mit den Defaultwerten überschrieben, sondern bleiben erhalten, d.h. das System wird mit den zuletzt geschriebenen Maschinen- und Achsparametern initialisiert. Ist das Register **mpCode** = 0, so wird es nach 1 geändert, ansonsten bleibt es unverändert.

icResetMP **$FF0B** **W0 / H**
Der Befehl setzt sämtliche Achs- und Maschinendaten auf ihre Defaultwerte, führt aber keine interne Initialisierung aus. Anschließend können die Daten über **icSetMP** bzw. **icSetAP** geändert werden. Danach ist in ***jedem Fall*** der Befehl **icInitMP** auszuführen!

Zum Lesen und Schreiben der Register stehen zwei Basis-Befehle zur Verfügung; die Registernummer wird auf das Lowbyte addiert (Bits 0..6 des Befehlsworts, s. Abb. 4.4):

icSetMP **$B000** **W2 / A**
icGetMP **$B080** **R2 / A**

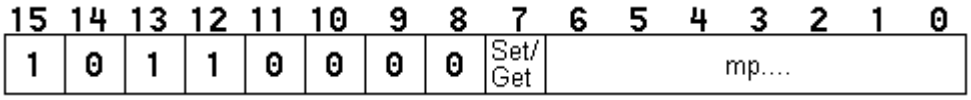


Abb. 4.4 Aufbau der Befehle **icXXXMP**

4.2.1.2 Allgemeine Maschinendaten

Folgende Register stehen zur Verfügung (R/O: nur Lese-Register, R/W: Schreib-/Lese-Register):

mpCode	**$0008**	**(R/W)**	**Kennung für die Maschinendaten**
Vorgabe	0	Kennung für die Defaultwerte von Maschinen- und Achsparametern	
	1	Der Befehl **icInitMP** wurde ausgeführt mit **mpCode** = 0. Hat das Register bei der Ausführung des Befehls **icInitMP** einen Wert <> 0, so wird er nicht verändert!	
	2..255	reserviert für spätere Erweiterungen alle anderen Werte können beliebig verwendet werden. Ein PC-Programm kann damit erkennen, ob Maschinendaten geändert wurden; außerdem ist es möglich, verschiedenen Maschinendatensätzen eine Kennung zuzuordnen.	
mpRAMSize	**$0010**	**(R/O)**	**RAM-Ausbau der I332**
mpAxis	**$001F**	**(R/O)**	**Achsversion**

Dieser Parameter besteht aus 4 Byte, die Anzahl der unterstützen Achsen steht im Lowbyte, die übrigen Byte enthalten interne Informationen

mpEmStop **$006F** **(R/W)** **Notstop-Reaktion**
Dieser Parameter konfiguriert die Notstop-Routine.
Bit 15 = 1 aktiviert die Notstop-Routine; +24V am Notaus-Eingang löst einen Interrupt aus
Lowbyte Notaus-Reaktion:
$00 **stSSB** und **stSFB** im Status-Register werden gelöscht
➔ alle Achsen werden normal gestoppt
$01 die Geschwindigkeit aller Achsen wird direkt(!) auf 0 gesetzt, zusätzlich werden die Freigaben weggenommen
➔ alle Achsen halten sofort an, es ist eine Referenzfahrt erforderlich

mpPotiInit **$005B** **(R/W)** **Initialisierung von Poti und Override**
Vorgabe $00008000
Dieser Parameter besteht aus 2 Worten:
Highword: Init für Override (s.a. **icSetOvr**), es wird nur das Highbyte ausgewertet:
➔ Override = 0: Poti ist aktiv
Lowword : Regelbereich für das Poti, es wird nur das Highbyte ausgewertet: $80 = 100%
➔ Potibereich 0..100%
Bei einem Restart-Befehl **icRestart..** wird das Poti bzw. der Override entsprechend diesem Parameter gesetzt.

mpFEFatal **$0049** **(R/W)** **max. zulässiger Schleppfehler**
Vorgabe 32000
Dieser Wert begrenzt den Schleppfehler auf den max. numerisch zulässigen Wert (in Schritt-Impulsen des Meß-Systems). Wird dieser Wert überschritten, so wird die Regelung von Servo-Motoren mit gesetztem **stFEB** im Statusregister abgebrochen und die Regler-Freigabe abgeschaltet.
Der Wert 32767 stellt die absolute Obergrenze dar und sollte nicht überschritten werden.

mpPID **$004A** **(R/W)** **Optionen für die PID-Regelung**
Vorgabe $000000FF
Bits 0..7 definieren den Zustand der Freigabe-Ausgänge nach einem Restart-Befehl für die Achsen 0..7, sofern für die Achse ein Freigabe-Ausgang definiert ist (**apDC**.**apEnB** = 1). Standardmäßig werden alle Freigaben gesetzt.

4.2.1.3 Arbeitsspindel

Die folgenden Parameter unterstützen den Einsatz einer Arbeitsspindel und die Funktionen M03/M04 (Spindel rechts/links) und M05 (Spindel aus) der DIN 66 025.

***Hinweis**:* *Zusammen mit der I332-ISA ist die Zusatz-Karte I332-DC erforderlich; falls keine freien Ausgänge vorhanden sind ist zusätzlich die I332-IO erforderlich.*
Für die I332-PCI-Version ist ggf. ein CAN-IO-Knoten erforderlich; für die Schrittmotor-Version der I332-PCI außerdem die Spindel-Option.

Es wird einer der ±10V-Sollwertausgänge zur Vorgabe der Spindeldrehzahl verwendet, ein oder zwei Ausgänge realisieren die beiden Schaltfunktionen.
Bei einem Restart-Befehl **icRestart..** werden die Ausgänge entsprechend "Spindel aus" gesetzt.

mpSP_Data **$0030** **(R/W)** **Definition der Spindel**
Vorgabe 0 keine Spindel
Dieses Register besteht aus einzelnen Bytes. Die beiden Bytes im Lowword definieren die Art der Spindel, das Highword wird nicht benutzt.
Lowbyte: Nr. des Sollwert-Ausgangs

0..7 entspricht den Ausgängen DC_X..DC_C. An einen hier definierten Ausgang darf keine Servo-Achse angeschlossen werden

Highbyte: Spindeltyp
- 0: keine Spindel
- 1: -10V..+10V = -SMax..+SMax, d.h. die Drehrichtung der Spindel wird über die Polarität der Sollwert-Spannung gesteuert
- 2: 0V..+10V = 0..SMax, d.h. die Spindel besitzt einen Schalt-Eingang für die Umschaltung der Drehrichtung

mpSP_MPorts **$0031** **(R/W)** **Ausgänge für Spindel ein/aus, rechts/links**

Vorgabe 0 kein Ausgang definiert

Dieses Register besteht aus zwei einzelnen Worten.

Highword: Ausgang für Spindel rechts/links

Lowword: Ausgang für Spindel ein/aus

Im Lowbyte jedes Wortes wird die Nr. des Ausgangs programmiert. Für die 30 Ausgänge OUT01..OUT30 der I332-IO sind die Nummern $60..$7F = 96..127 vorgesehen.

Das Highbyte enthält 3 Bitflags, die den Zustand des Ausgangs für die Funktionen Spindel rechts/links/aus definieren. Ein Wert von 0 bzw. 1 heißt: Am Ausgang liegt Masse-Pegel bzw. die Spannung UExt an.

Bitname	Bitnr.	
---	**0**	- unbelegt -
m03B	**1**	Ausgangspegel für Spindel rechts (M03)
m04B	**2**	Ausgangspegel für Spindel links (M04)
m05B	**3**	Ausgangspegel für Spindel aus (M05)

***Hinweis**: Für diese Ports sollten auch die zugehörigen Bits in den entsprechenden Maschinenparametern **mpXXXOutPAR** gelöscht werden, damit die Spindel über die modalen Spindelbefehle (s. Kap. 4.5.2) satzsynchron bedient werden kann.*

mpSP_SMax **$0033** **(R/W)** **Maximal-Drehzahl der Spindel in U/min**

In diesem Langwort-Register wird die Nominal-Drehzahl angegeben, die die Spindel bei einer Ausgangsspannung von +10V erreicht.

4.2.1.4 Ein-/Ausgänge

Die folgenden Register dienen der Initialisierung aller Ausgänge (I332-ISA, I332-DC und I332-IO bzw. I332-PCI und CAN-Knoten), soweit sie nicht für Achsfunktionen benötigt werden. Details zur Programmierung der Ein-/Ausgabe s. Kap. 4.8. Bei einem Restart-Befehl **icRestart..** werden die Ausgänge (bzw. die Register **OutBuf** und/oder **MOutBuf**, abhängig vom Restart-Befehl) entsprechend der Parameter **mpXXOutDef** gesetzt.

In Verbindung mit den Portbefehlen (s. Kap. 4.8.2) ist es möglich, Ausgänge entweder synchron zu einzelnen Fahrsätzen durch den Interpolator oder zu einem beliebigen Zeitpunkt durch den PC zu setzen. Die Register **mpXXOutPAR** initialisieren diese Zuordnung bei einem Restart-Befehl. Ein Wert von 0 bzw. 1 heißt Zugriff durch den Interpolator (satzsynchron) bzw. durch den PC (weitere Details s. Kap. 4.8.2).

mpSMOutDef **$003C** **(R/W)** **Initialisierung der Ausgänge $00..$1F**

Vorgabe 0

Das Highword des Registers besteht aus Bitpaaren, die jeweils den Zustand der Puls/Rtg-Ausgänge für die Achsen X bis C nach einem Restart-Befehl definieren, falls an diesen Ausgängen kein Schrittmotor angeschlossen ist (s. **apChar**). Das Lowword ist nicht belegt.

Ein Wert von 0 bzw. 1 heißt: Am Ausgang liegt Masse-Pegel bzw. die Spannung UExt an.

Highword: Bit 0/1..14/15 = PulsX/RtgX..PulsC/RtgC

Lowword: nicht belegt

mpEnOutDef **$003D** **(R/W)** **Initialisierung der Ausgänge $20..$3F**

Vorgabe 0

In diesem Register sind nur die Bits 16 bis 23 gültig. Sie definieren den Zustand der Freigabe-Ausgänge der Achsen X bis C nach einem Restart-Befehl, wenn für diese Achse keine programmgesteuerte Freigabe erfolgen soll (**apDC.dcEnOutB** = 0).

Ein Wert von 0 bzw. 1 heißt: Am Ausgang liegt Masse-Pegel bzw. die Spannung UExt an.

Highwort: Bit 24..31: nicht belegt
Bit 16..23: FreiX..FreiC

Lowwort: Bit 0..15: nicht belegt

mpIOOutDef **$003F** **(R/W)** **Initialisierung der Ausgänge $60..$7F**

Vorgabe 0

Das Register besteht aus zwei Worten mit je 16 Bitflags, die den Zustand des entsprechenden Ausgangs nach einem Restart-Befehl definieren. Ein Wert von 0 bzw. 1 heißt: Am Ausgang liegt Masse-Pegel bzw. die Spannung UExt an.

Highwort: Bit 0..15 = Ausgänge OUT01..OUT16

Lowwort: Bit 0..15 = Ausgänge OUT17..OUT32

mpSMOutPAR **$0040** **(R/W)** **Zuordnung der Ausgänge $00..$1F**

Vorgabe 0

Das Highword des Registers besteht aus Bitpaaren für die Puls/Rtg-Ausgänge der Achsen X bis C. Das Lowword ist nicht belegt.

Die Werte sind nur gültig, falls an diesen Ausgängen kein Schrittmotor angeschlossen wird (s. **apChar**).

Highword: Bit 0/1..14/15 = PulsX/RtgX..PulsC/RtgC

Lowword: nicht belegt

mpEnOutPAR **$0041** **(R/W)** **Zuordnung der Ausgänge $20..$3F**

Vorgabe 0

In diesem Register sind nur die Bits 16..23 für die Freigabe-Ausgänge der Achsen X bis C gültig. Alle anderen Bits sind nicht belegt.

Der Wert ist nur gültig, wenn für diese Achse keine programmgesteuerte Freigabe erfolgen soll (**apDC.dcEnOutB** = 0).

Highwort: Bit 24..31: nicht belegt
Bit 16..23: FreiX..FreiC

Lowwort: Bit 0..15: nicht belegt

mpIOOutPAR **$0043** **(R/W)** **Zuordnung der Ausgänge $60..$7F**

Vorgabe 0

Das Register besteht aus zwei Worten mit je 16 Bitflags.

Highwort: Bit 0..15 = Ausgänge OUT01..OUT16

Lowwort: Bit 0..15 = Ausgänge OUT17..OUT32

4.2.2 Achsparameter

Zur Anpassung an verschiedene Maschinen und Motoren erlaubt die I332 die Definition einer Reihe von Daten.
Der Anhang B enthält Hinweise zur Einstellung und Optimierung einiger Parameter.

Die achsbezogenen (**ap...**) Maschinendaten sind für jede Achse einzeln definierbar, die Befehle sind vom Typ »Achsregister«, d.h. die Achsnr. muß zum Highbyte addiert werden (Bits 8..10 des Befehlsworts, s. Abb. 4.5); alle Daten haben Langwort-Format.

Die Basisbefehle sind

icSetAP **$4000** **W2 / A**
Schreiben eines Achsparameter-Registers.

icGetAP **$4080** **R2 / A**
Lesen eines Achsparameter-Registers.

Die Nr. des gewünschten Achsparamters (**ap...**) wird zum Lowbyte addiert. Der gesamte Befehl setzt sich also aus 3 Teilen zusammen (Bits 0..6 des Befehlsworts, s. Abb. 4.5):
z.B.: Beschl. der Achse 4 setzen: **icSetAP** + 4*256 + **apAcc** = **$4403**
Modulo der Achse 2 lesen: **icGetAP** + 2*256 + **apMod** = **$4205**

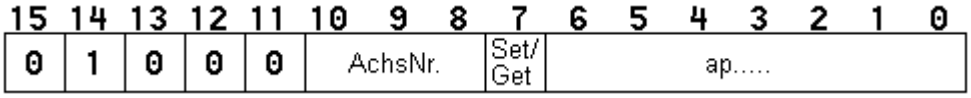


Abb. 4.5 Aufbau der Befehle icXXXAP

Für jeden Parameter sind in der I332 Vorgabe-Werte gespeichert, die sofort nach dem Einschalten zur Verfügung stehen. Bei einem ersten Test der Karte sind daher keine Definitionen erforderlich.
Die Achsparameter beschreiben die physikalischen Eigenschaften der Motoren und sie werden deshalb einmal zu Beginn (z.B. in einer Initialisierungs-Routine) gesetzt. Während der laufenden Arbeit sollten sie nicht verändert werden.
Sind Änderungen erforderlich, genügt es, nur die gewünschten Parameter neu zu setzen. Sind alle Änderungen erfolgt, MUSS zum Abschluß der Befehl **icAPEnd** oder der Befehl **icInitMP** ausgeführt werden:

icAPEnd **$FE00** **W0 / A**
Der Befehl initialisiert intern verwendete Variablen, die von den Achsparametern abhängen.

Bei einem **icRestart0**-Befehl werden alle Achsparameter wieder auf die Vorgabe-Werte gesetzt; alle anderen Restart-Befehle verändern die Achsparameter nicht (soweit nichts anderes angegeben ist) und führen auch keine interne Initialisierung aus.
Die angegebenen Werte-Bereiche sind die numerischen Eingabe-Grenzen; die tatsächlich verarbeiteten Werte hängen insbesondere auch von der mechanischen Auflösung (**apStep** bzw. **apmm**) ab (s.a. Kap. 6 - Technische Details)!

4.2.2.1 mechanische Achsparameter:

apVMax **$0000** **Maximalgeschwindigkeit in mm/min**
Vorgabe 1000
Wertebereich 0 .. +4.295.967.294 (unsigned long)
Hinweis: *u.U. wird die Maximalgeschwindigkeit durch die interne Tabelle zur Impulserzeugung (SM) bzw. (bei SV-Motoren) durch die Auflösung des Meß-Systems (=16 Bit) begrenzt!*

apVStart **$0001** **Startgeschwindigkeit in mm/min**
Mit dieser Geschwindigkeit kann der Motor sofort ohne Rampe losfahren
Vorgabe 200
Wertebereich 0 .. +4.295.967.294 (unsigned long)

apVStop **$0002** **Stopgeschwindigkeit in mm/min**
Aus dieser Geschwindigkeit kann der Motor sofort ohne Rampe stoppen
Vorgabe 200
Wertebereich 0 .. +4.295.967.294 (unsigned long)

apAcc **$0003** **Beschleunigung in mm/sec²**
Steigung der (linearen) Beschleunigungsrampe
Vorgabe 500
Wertebereich 0 .. +4.295.967.294 (unsigned long)
Anmerkung Interne Auflösung ca. 10mm/sec²

apDec **$0004** **Verzögerung in mm/sec²**
Steigung der (linearen) Bremsrampe
Vorgabe 500
Wertebereich 0 .. +4.295.967.294 (unsigned long)
Anmerkung Interne Auflösung ca. 10mm/sec²

apMod **$0005** **Modulo in µm**
z.B. für Rundachsen: die Istwertanzeige läuft von 0 bis apMod-1
Vorgabe 0 (kein Modulo)
Wertebereich 0 .. +4.295.967.294 (unsigned long)

apStep/ **$0006** **/ mechanische Auflösung der Schrittmotoren**
apmm **$0007** **\ bzw. des Meßsystems von Servomotoren**
Vorgabe apStep 1000 (Schritte oder Incremente)
Vorgabe apmm 1 (pro mm bzw. Grad)
Wertebereich 0 .. +4.295.967.294

apVRef **$0008** **Geschwindigkeit für die Referenzfahrt in mm/min**
Vorgabe 500
Wertebereich 0 .. +4.295.967.294 (unsigned long)

apVHys **$0009** **Geschwindigkeit für die ES-Hysterese in mm/min**
sollte nicht größer als apVStart bzw. apVStop sein
Vorgabe 200
Wertebereich 0 .. +4.295.967.294 (unsigned long)

apVRefP **$000A** **Referenzpuls suchen, in mm/min**
Mit dieser Geschwindigkeit wird ggf. ein am RefPuls-Eingang auf der I332-DC-Karte angeschlossener Referenz- oder Resolver-Impuls gesucht. Dazu müssen auch die Bits **dcRefIn** und **dcRefPol** im Register **apDC** (s.u.) entsprechend gesetzt sein.
Vorgabe 20
Wertebereich 0 .. +4.295.967.294 (unsigned long)

apHys **$000B** **Endschalter-Hysterese, in µm**
Um diese Strecke (in µm) wird bei einer Referenzfahrt mit der Geschwindigkeit **apVHys** vom Endschalter wegpositioniert (Schalt-Hysterese des ES)
Vorgabe 1000

Wertebereich 0 .. +4.295.967.294 (unsigned long)

apHysMax **$000C** **max. Endschalter-Hysterese, in µm**
Falls nach dieser Strecke (in µm) der Endschalter noch immer aktiv ist, wird mit einer Fehlermeldung abgebrochen.
Vorgabe 50000
Wertebereich 0 .. +4.295.967.294 (unsigned long)

apLS **$000D** **Endschalter-Flags**
Dieser Parameter enthält verschiedene Bitflags, die das Endschalter-Verhalten festlegen. Die Bits **lsP..** (Bits 0..6) betreffen den Eingang ES+, die Bits **lsM..** (Bits 8..14) den Eingang ES-

Bitname	Bitnr.	
lsPInB	**0**	Am Eingang ES+ der betreffenden Achse ist ein Endschalter angeschlossen
lsPPolB	**1**	Der angeschlossene Endschalter ist high aktiv (Spannung am Eingang)
lsPRefB	**2**	Der Eingang ES+ angeschlossene Endschalter dient nur als Referenzschalter (z.B. bei Rundachsen)
lsPHDirB	**4**	Der Endschalter wird in Minus-Rtg. freigefahren
lsPRDirB	**5**	Bei Ref.-Fahrt in Plus-Rtg. wird nach dem Freifahren des ES der Ref.-Puls in Minus-Rtg. gesucht (nur bei I332-DC)
lsMInB	**8**	Am Eingang ES- der betreffenden Achse ist ein Endschalter angeschlossen
lsMPolB	**9**	Der angeschlossene Endschalter ist high aktiv (Spannung am Eingang)
lsMRefB	**10**	Der am Eingang ES- angeschlossene Schalter dient nur als Referenzschalter (z.B. bei Rundachsen)
lsMHDirB	**12**	Der Endschalter wird in Minus-Rtg. freigefahren
lsMRDirB	**13**	Bei Ref.-Fahrt in Minus-Rtg. wird nach dem Freifahren des ES der Ref.-Puls in Minus-Rtg. gesucht (nur bei I332-DC)
lsChg	**7**	Ab Version 2.2: dieses Bit vertauscht die logische Zuordnung der Endschalter an den Eingängen ES+/ES- zu den Achsrichtungen +/-. Dies hat folgende Auswirkungen auf die Auswertung der Bits **lsP...** und **lsM...**: Die Bits **lsP/MInB** und **lsP/MPolB** gelten weiterhin für die tatsächlich am Eingang ES+/ES- angeschlossenen Schalter, während die übrigen Bits **lsP/M...** sich auf die Achsrichtung beziehen! ***Hinweis**:* *Dieses Bit sollte nur gesetzt werden, wenn aus irgendeinem Grund die Achsrichtungen einer schon konfigurierten Maschine geändert werden müssen. In den Maschinen-Unterlagen sollte dies deutlich dokumentiert werden!*

Vorgabe $131 = **lsPInF**+**lsMInF**+**lsPHDirF**+**lsPRDirF**
(Bits **lsPInB**, **lsMInB**, **lsPHDirB**, **lsPRDirB** = 1)
> beide ES angeschlossen
> low aktiv (Eingang offen)
> Freifahren und Ref.-Puls-Suchen jeweils in Gegenrtg.

apLSMax **$000F** **max. Bremsweg bei aktivem Endschalter, in µm**
Dieser Parameter definiert einen maximalen Bremsweg bei Erkennung eines Endschalters. Die Verzögerung wird ggf. so vergrößert, daß die Achse innerhalb dieses Weges zum Stillstand kommt.

Vorgabe 0 Es wird mit **apDec** auf Stillstand abgebremst

***Hinweis**:* *Bei großen Geschwindigkeiten kann es entsprechend zu hohen Verzögerungswerten kommen. Insbesondere bei Schrittmotoren stimmt dann u.U.*

die Position nicht mehr, da der Motor evtl. außerhalb seiner Spezifikation gebremst wird (Schrittverlust!). Bei Servomotoren kann es zusätzlich zu einem Schleppfehler kommen.

apChar **$0010** **Achs-Bitflags**
Dieser Parameter enthält ebenfalls verschiedene Bitflags, die eine bestimmte Achs-Charakteristik definieren.

Bitname	Bitnr.	
acPathB	**0**	Die Achse ist eine der Bahnachsen, d.h. die Bahngeschwindigkeit bezieht sich auf ein Koordinatensystem, das durch die Achsen mit gesetztem **acPathB** festgelegt wird.
acSMB	**8**	Antrieb: Schrittmotor
acDCB	**9**	Antrieb: Servomotor (nur bei Servo-Versionen)

Vorgabe $101 für die Achsen 0,1 und 2 (XYZ)
= **acPathF**+**acSMF** (Bits **acPathB**, **acSMB** = 1)
> Bahnkoordinaten XYZ
> Antrieb Schrittmotor
$100 für alle anderen = **acSMF** (**acSMB** = 1)
> Antrieb Schrittmotor

apSM **$0011** **Schrittmotor-Bitflags**
Dieser Parameter enthält Bitflags, die die Charakteristik der Schrittmotoren definieren:

Bitname	Bitnr.	
smPlsPolB	**0**	Polarität des Schrittmotor-Impulses ist aktiv low
smDirPolB	**1**	Polarität des Richtungs-Ausgangs
Vorgabe	$0000	für alle Achsen:
		> Puls-Ausgabe aktiv low
		> Richtungs-Ausgabe: high für positive Richtung

apDC **$0012** **Bitflags für Servomotoren**
Dieser Parameter enthält im Lowword Bitflags, die die Porteigenschaften bei den Servo-Versionen definieren; das nächst höhere Wort (Bit 16..31) dient der Skalierung der PID-Faktoren **apPFac**, **apIFac**, **apDFac** und **apVFac**.

Bitname	Bitnr.	
dcEnOutB	**0**	Enable-Ausgang zum Verstärker vorhanden
dcEnPolB	**1**	Enable Ausgang aktiv low
dcRdyInB	**2**	Ready-Eingang vom Verstärker vorhanden
dcRdyPolB	**3**	Ready-Eingang aktiv high
dcRefInB	**4**	Referenz-Puls-Eingang vorhanden
dcRefPolB	**5**	Refpuls-Eingang aktiv high
dcDGInvB	**8**	Auswertung der Drehgeber-Impulse wird invertiert
dcDACInvB	**9**	Sollwert-Ausgabe an die DA-Wandler wird invertiert
svIDScale	**16..23**	Skalierung für die I- und D-Faktoren; die Skalierung erfolgt mit dem Faktor $2^{svIDScale}$; dieser Faktor steht ab V. 2.4.22 zur Verfügung. Details dazu finden sich in der Datei PID332.PDF, die den implementierten PID-Algorithmus dokumentiert. Wertebereich: -31..+31
svPVScale	**24..31**	Skalierung für die P- und V-Faktoren; die Skalierung erfolgt mit dem Faktor $2^{svPVScale}$; dieser Faktor steht ab V. 2.4.22 zur Verfügung. Details dazu finden sich in der Datei PID332.PDF, die den implementierten PID-Algorithmus dokumentiert.

Wertebereich: -31..+31

Vorgabe	$00000000	für alle Achsen: ➔ Ein-/Ausgänge der I332-Karte werden vom Programm nicht automatisch ausgewertet (d.h. bei Einsatz von Servo-Motoren MUSS dieses Maschinendatum entsprechend gesetzt werden)! ➔ keine Skalierung der PID-Faktoren

4.2.2.2 Maschinenkinematik

Die folgenden Parameter definieren verschiedene maschinenabhängige Positionen. Sie werden während bzw. nach der Referenzfahrt in die entsprechenden Register übernommen.

apLSDeltaM $0014 Verfahrbereich in negativer Richtung, in µm
apLSDeltaP $0015 Verfahrbereich in positiver Richtung, in µm
Diese Parameter definieren den Verfahrbereich der Achse (in µm) und zwar bezogen auf den Referenzpunkt **apRefPos** der Maschine. Ist **apLSDeltaM** <> **apLSDeltaP**, so werden die Werte **apRefPos+apLSDeltaM** und **apRefPos+apLSDeltaP** nach der Referenzfahrt als Verfahrgrenzen der Achse berücksichtigt (Software-Endschalter).
Vorgabe 0
Wertebereich -2.147.483.646..+2.147.483.646 (signed long)

apRefOffs $0016 Offset des Referenzpunkts zum Referenz-Schalter, in µm
Ist **apRefOffs** <> 0, so wird nach dem Suchen des Referenz-Impulses diese Strecke zusätzlich verfahren. Damit ist es z.B. möglich, eine Achse parallel zu einer Arbeitsspindel auszurichten.
Vorgabe 0
Wertebereich -2.147.483.646..+2.147.483.646 (signed long)

apRefPos $0017 Referenzposition, in µm
Dieser Wert wird am Ende der Referenzfahrt als aktuelle Position in das Positionsregister übernommen.
Vorgabe 0
Wertebereich -2.147.483.646..+2.147.483.646 (signed long)

4.2.2.3 PID-Parameter

Die folgenden Parameter dienen zur Einstellung des PID-Algorithmus und der Digital-Analog-Wandler (DAC) für die Ansteuerung von Servo-Motoren. Die PID-Faktoren **apIFac** und **apDFac** lassen sich ab V. 2.2 mit Hilfe eines Faktors **svPIDScale** (s.o.) skalieren. Details dazu finden sich in der Datei PID332.PDF, die den implementierten PID-Algorithmus dokumentiert.

apPFac **$0018** **P-Faktor**
Vorgabe 20
Wertebereich 0 .. 32767 (unsigned int)

apIFac **$0019** **I-Faktor**
Vorgabe 0
Wertebereich 0 .. 32767 (unsigned int)

apDFac **$001A** **D-Faktor**
Vorgabe 0
Wertebereich 0 .. 32767 (unsigned int)

apVFac **$001B** **V-Faktor**
für die Geschwindigkeits-Vorsteuerung
Vorgabe 0
Wertebereich 0 .. 32767 (unsigned int)

apInPos **$001C** **In-Positions-Bereich in apSteps**
Ein Satzübergang erfolgt erst, wenn der Schleppfehler kleiner als dieser Wert ist.
Vorgabe 10
Wertebereich 0 .. 32767

apFEMax **$001D** **Maximaler Schleppfehler in apSteps**
Beim Überschreiten wird das Schleppfehler-Bit im PIDInfo-Register gesetzt. Überschreitet der Schleppfehler den Wert **mpFEFatal**, wird das Schleppfehler-Bit (**stFEB**) im Statusregister gesetzt und das Fahren abgebrochen.
Vorgabe 1600
Wertebereich 0 .. 32000

ap0Off **$001E** **Null-Offset des DAC**
Damit lassen sich evtl. Unsymmetrien am Verstärker-Eingang ausgleichen.
Vorgabe 0
Wertebereich -32767 .. +32767 (signed int), entspr. -10V..+10V

apAmpMax **$001F** **Begrenzung der Ausgangs-Spannung**
Vorgabe 32000
Wertebereich 0 .. +32767 (unsigned int), entspr. 0V .. 10V

apdFMax **$0020** **Überwachung der Istgeschwindigkeit**
Ähnlich wie **apFEMax** die Position überwacht dieser Parameter die Istgeschwindigkeit der Achsen. Der Unterschied der Istgeschwindigkeit (→ Geschwindigkeitssprung) zwischen 2 Abtast-Intervallen darf diesen Wert nicht überschreiten, ansonsten erfolgt eine Reaktion wie bei Schleppfehler.
Vorgabe 1600
Wertebereich 0..32000

4.3 Der Satzpuffer - Gruppe B

Der Satzpuffer ist ein zentraler Teil des Systems. Er kann (je nach Speicherausbau) ca. 150 bis 1000 Sätze aufnehmen. Dabei umfaßt ein Satz alle Daten, die zum Verfahren (Interpolieren) eines Bahnstücks notwendig sind. Außerdem können in einem Satz noch technologische Daten, z.B. das Ein- oder Ausschalten von Eingängen oder die Drehzahl einer Arbeitsspindel, programmiert werden. Die Struktur eines Satzes ist damit sehr ähnlich der Satz-Struktur der DIN 66 025.

Organisiert ist der Satzpuffer als FIFO-Speicher (First In - First Out, auch Warteschlange). Die Daten vom PC gelangen dabei in den Satz am einen Ende, während der Interpolator bereits die vorher eingegebenen Sätze verarbeitet (d.h. die Achsen verfährt). Damit lassen sich kontinuierliche Bahnbewegungen aus aufeinanderfolgenden Bahnstücken ohne weitere Maßnahmen realisieren.

Um zu verdeutlichen, welche Befehle welchen Satz beeinflussen, wurde die Gruppe B in die Untergruppe B(1) und B(2) aufgeteilt (s. Abb. 4.7)

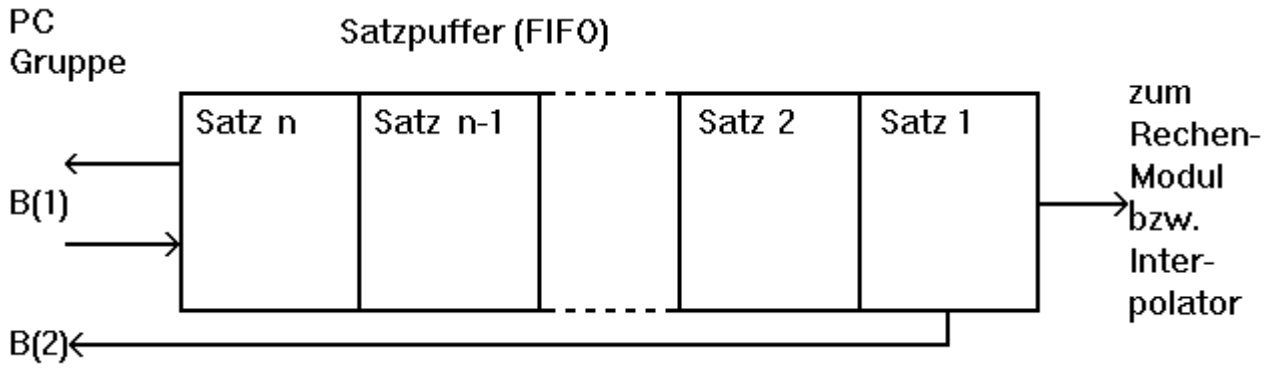


Abb. 4.6 Organisation des Satzpuffers für Interpolations-Sätze

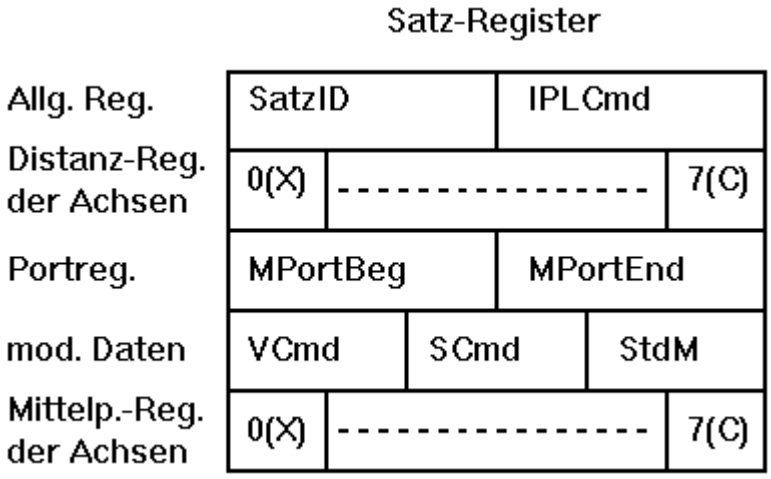


Abb. 4.7 Organisation eines Satzes

Die Programmierung eines kompletten Fahrsatzes / Interpolationssatzes besteht aus folgenden Schritten:

0 Programmierung der modalen Daten (s. Kap. 4.5, Gruppe-D-Befehle)
1 Programmierung der Endpunkte (relativ zum Satzanfang)
2 Programmierung weiterer Satzdaten
3 Programmierung der Interpolationsart (Interpolationsbefehl)

Die Reihenfolge der Punkte 0 bis 2 ist beliebig; Punkt 0 muß nur ausgeführt werden, wenn sich diese Daten ändern (s.Kap. 4.5). Der Interpolationsbefehl (Punkt 3) schließt die Programmierung eines

Satzes ab, übernimmt die modalen Daten in den Satz und schaltet den Satzpuffer um einen Satz weiter.
Details zur Programmierung s. Kap. 5.

4.3.1 Achsregister

icSetDist $0000 W2 / B(1)
Fahr-Distanz für diesen Interpolationssatz: Koordinaten des Endpunkts relativ zum Satzanfang.

icSetCP $0002 W2 / B(1)
Abstand zum Kreismittelpunkt setzen: Koordinaten des Kreismittelpunkts relativ zum Satzanfang.

icGetRest $0083 R2 / B(2)
Liest die noch zu fahrende Distanz im aktuellen Satz

Die I332 verarbeitet nur Angaben relativ zur Position am Satzanfang (Kettenmaß).

4.3.2 sonstige Satzregister

icSetID $8400 W2 / B(1)
Trägt eine Satzkennung in den nächsten Interpolations-Satz ein.

icGetID $8480 R2 / B(2)
Liest die Satzkennung des laufenden Satzes (d.h. der gerade verfahren wird).

Diese beiden Befehle gestatten die Identifizierung eines Satzes (z.B. zur Satzanzeige in einer Bedienoberfläche). Die Satzkennung wird von der I332 nicht verändert und auch in keiner Weise ausgewertet.

icSetWait $8401 W2 / B(1)
Verweilzeit setzen; die Programmierung erfolgt in ms, die interne Auflösung beträgt ca. 10ms.

icGetWait $8481 R2 / B(2)
restliche Verweilzeit im aktuellen Satz lesen (in ms)

icSetAccFac $8406 W1 / D
icSetDecFac $8407 W1 / D
Die in den Maschinendaten gesetzten Werte **apAcc** und **apDec** für Beschleunigung und Verzögerung können mit diesen beiden Befehlen modal im Bereich 0.003%..200% beeinflußt werden:

$0000	es gilt **apAcc** bzw. **apDec** (d.h. 100%)
$0001..$FFFF	Beschleunigung ist **apAcc** x **AccFac**/32768 bzw. Verzögerung ist **apDec** x **DecFac**/32768 → $8000 entspricht 100%

icSetMPort $D200 W1 / B(1)
Setzt oder löscht beliebige Ausgänge am Satzanfang oder am Satzende, wenn im Port-Assignment-Register **MOutPAR** das Bit für diesen Ausgang auf 0 gesetzt ist. Weitere Details s. Kap. 4.8.2.

4.3.3 Interpolations-Art

Der Befehl beendet die Übergabe von Satzdaten (Befehls-Gruppe B), übernimmt modale Daten (Befehlsgruppe D) in diesen Satz und schaltet den Satzpuffer um eins weiter. Die danach folgenden Satzdaten werden damit in den nächsten Satz übernommen.

Der Basisbefehl ist »**icIPL**«, der je nach Art der Interpolation durch Addition weiterer Konstanten und durch zusätzliche Datenworte ergänzt wird.

icIPL **$80xx** **B(1)**
Basisbefehl

iplModePos **$0080**
Addition der Konstanten **iplModePos** (d.h. Bit 7 der Befehle **icIPLxx** ist gesetzt) führt die Interpolation im Eilgang aus (d.h. Positionieren mit Geschwindigkeit VMax).

4.3.3.1 Linear-Interpolation

icIPLLin **$8001** **W0 / B(1)**
Linear-Interpolation

Die Bahngeschwindigkeit (**icSetVCmd**) bezieht sich auf die Bahn, die durch diejenigen Achsen definiert wird, deren **acPathB** im Achsparameter »**apChar**« gesetzt ist (Bahnachsen). Im Normalfall sind dies die Achsen 0-2 = XYZ → Bahn im 3D-Raum.
Die Geschwindigkeit dieser Achsen wird entsprechend berechnet; die Geschwindigkeit aller anderen Achsen wird so berechnet, daß sie gleichzeitig den Endpunkt erreichen. Falls dabei eine Achse ihre Maximalgeschwindigkeit **apVMax** überschreiten würde, wird die Bahngeschwindigkeit entsprechend reduziert.

Ist keine der Bahnachsen an der Interpolation beteiligt, so wird die Bahn durch die Achse mit der längsten Distanz definiert.

4.3.3.2 Kreis-Interpolation

icIPLcw **$8002** **W1 / B(1)**
Kreis-Interpolation im Uhrzeigersinn (UZS)

icIPLccw **$8003** **W1 / B(1)**
Kreis-Interpolation im Gegen-Uhrzeigersinn (GZS)

Ein zusätzliches Datenwort bei der Kreisinterpolation definiert die Kreis-Ebene: High- und Low-Byte bestimmen die Achsnr. für die X- bzw. Y-Koordinate

Die Bahngeschwindigkeit (**icSetVCmd**) ist die Geschwindigkeit auf der Kreisbahn; dies gilt auch dann, wenn keine der Bahnachsen (Achsen mit **apChar.acPathB** = 1) den Kreisbogen definiert.

Anmerkung: *Die getrennte Definition zweier Kreisbefehle (UZS und GZS) ist eigentlich überflüssig: eine Umkehr der Kreisrichtung ist auch durch Vertauschen der Koordinaten-Achsen im Datenwort von **icIPLcw** bzw. **icIPLccw** erreichbar.*

4.3.3.3 Verweilzeit

icIPLWait **$8004** **W0 / B(1)**
Verweilzeit

Das Einfügen einer Verweilzeit geschieht wie das Programmieren eines Interpolations-Satzes. Es dürfen jedoch keine Achsregister programmiert werden. Nach dem Setzen der Verweilzeit mit dem Befehl **icSetWait** und ggf. dem Programmieren von Ausgängen mit **icSetMPort** beendet der "Interpolationsbefehl" **icIPLWait** die Eingabe der Satzdaten.
Sobald der Satz ausgeführt wird, kann mit **icGetWait** die noch nicht verstrichene Verweilzeit gelesen werden.

4.4 Sonderfunktionen - Gruppe C

Sonderfunktionen lassen sich für alle Achsen unabhängig voneinander und parallel zur Interpolation ausführen. Für die Programmierung steht ein Puffer zur Verfügung mit einem Registersatz für jede Achse (Abb. 4.8, Abb. 4.9)

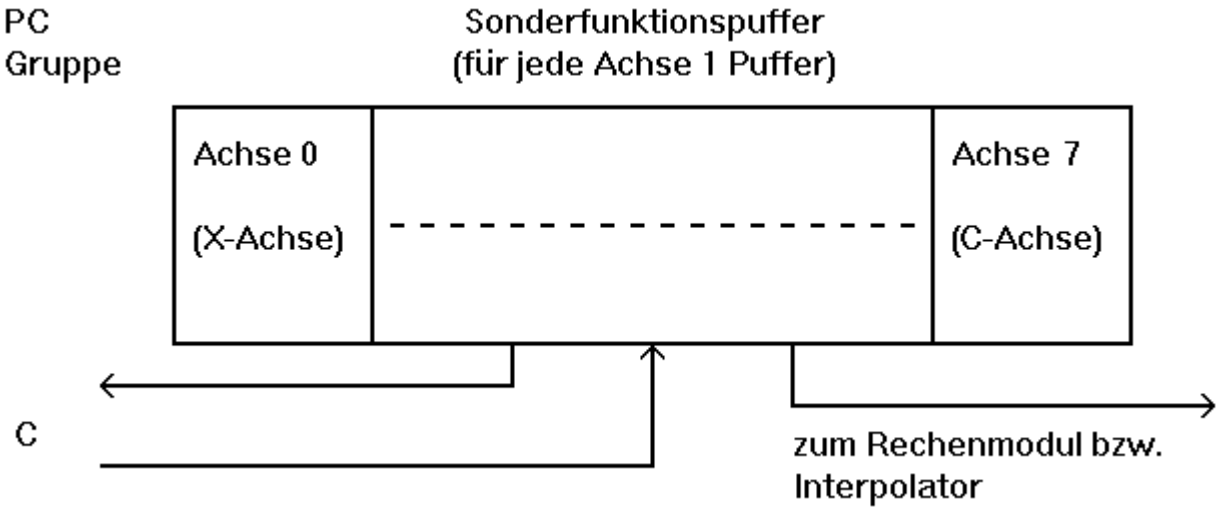


Abb. 4.8 Puffer für Sonderfunktionen: 1 Registersatz / Achse

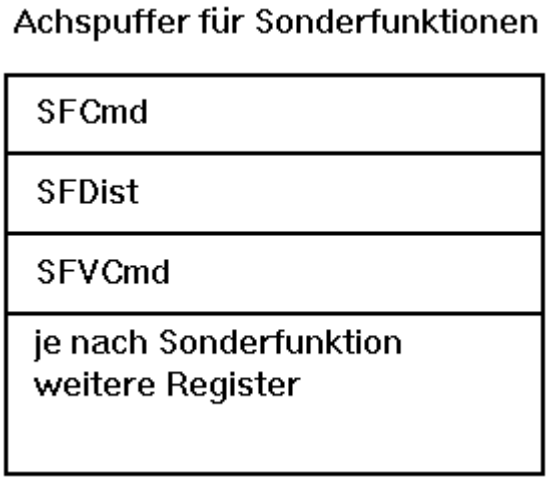


Abb. 4.9 Registersatz einer Achse für die Sonderfunktionen

Das Programmierschema ist das gleiche wie bei der Programmierung eines Interpolationssatzes, d.h. es werden zunächst die notwendigen Daten in die Achs-Register der gewünschten Achse geschrieben; das Schreiben des gewünschten Funktions-Befehls schließt die Programmierung dieser Achse ab.
Der Funktions-Befehl spezifiziert die gewünschte Funktion und bereitet das Verfahren der Achse vor. Die Funktions-Befehle benötigen die Angabe einer Achsnr. im Highbyte (Bits 8..10 des Befehlsworts) wie die Achsregister, sind jedoch W1-Befehle.
Im zusätzlichen Datenwort werden weitere (funktions-abhängige) Informationen übergeben. Für alle Sonderfunktionen gleich ist die Richtungsangabe (falls sie jeweils sinnvoll ist bzw. nicht anderweitig schon übergeben wird, z.B. als Vorzeichen eines anderen Datums) und die Definition der Geschwindigkeit in diesem Datenwort:

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

Je nach Funktion	sfDir	0	sfM49	sfLSOff	sfVxxx

Abb. 4.10 Aufbau Datenwort für Sonderfunktionen - Lowbyte

Richtungsbit Dir (Bit 7):

sfPos	**$0000**	positive Richtung
sfNeg	**$0080**	negative Richtung

Achs-/ Funktions-spezifische Geschwindigkeiten sfVxxx (codiert in Bit 0..3):

sfVStd	**$0000**	Standard-Geschwindigkeit für die ausgewählte Funktion z.B. **apVRef** für **icSFRef** (Referenzfahrt)
sfVCmd	**$0001**	Geschwindigkeit = **icSFVCmd**
sfVMax	**$0003**	Geschwindigkeit = **apVMax**
sfVSS	**$0004**	Geschwindigkeit = min (**apVStart**,**apVStop**)

folgende Konstanten (Bit 4 und 5) sind vor allem für den Joystick-Betrieb gedacht:

sfLSOff	**$0010**	ES-Behandlung abschalten, ab V2.2.31
sfM49	**$0020**	Poti abschalten: Fahren mit 100% (DIN66025: M49-Funktion)

Eine spezielle Funktion hat das Datenwort

sfBreak	**$FFFF**	Eine laufende Sonderfunktion wird abgebrochen.

Weitere Konstanten sind ggf. bei der jeweiligen Sonderfunktion definiert und werden ebenfalls dazuaddiert.

Da für jede der Achsen nur eine Funktion definiert werden kann, sollte vor der Programmierung einer Achse mit dem Befehl **icGetSFInfo** geprüft werden, ob die Achse bereits eine Sonderfunktion ausführt:

icGetSFInfo $C900 R1 / C
Liefert die derzeit mit einer Sonderfunktion aktiven Achsen als Bitflags. Gesetzte Bits im Lowbyte repräsentieren Achsen, bei denen irgendeine Sonderfunktion aktiv ist, d.h. diese Achse kann erst wieder neu programmiert werden, wenn die gerade aktive Funktion beendet ist.
Wird die Nr. einer Sonderfunktion (**icSetSF...**, s. Kap. 4.4.2) auf das Lowbyte des Befehls addiert, so sind im Highbyte des Ergebnisses die Bits für solche Achsen gesetzt, für die diese Sonderfunktion gerade aktiv ist, ansonsten ist das Highbyte undefiniert.

Weitere Details zur Programmierung s. Kap. 5.3

***Hinweis**:* *Wegen der internen Verwaltung der Positionsdaten von Interpolationssätzen und Sonderfunktionen in getrennten Registersätzen kann es vorkommen, daß die von den Registersätzen verwalteten Positionen nicht synchron sind; ein Restart-Befehl (**icRestart1..5**, **icRestart11..15**) synchronisiert die beiden Registersätze wieder neu. Dies ist immer dann notwendig, wenn eine Achse Sonderfunktionen ausgeführt hat und danach mit anderen Achsen zusammen wieder interpolierend verfahren soll.*
Weitere Hinweise zu diesem Thema im Kap. 5.4, „Interpolation und Sonderfunktionen“!

4.4.1 Achsregister für Sonderfunktionen

icSetSFDist $0010 W2 / C
Wegstrecke für die nächste Sonderfunktion setzen: Distanz zum nächsten Endpunkt

icGetSFDist $0090 R2 / C
Restweg der aktuellen Sonderfunktion: Distanz zum nächsten Endpunkt

icSetSFVCmd $0011 W2 / C
Geschwindigkeit für die nächste Sonderfunktion setzen

icGetSFVCur $0091 R2 / C
aktuelle Geschwindigkeit der Achse

4.4.2 Funktions-Befehle

Funktionsbefehle werden ins Register **SFCmd** geschrieben und beenden die Programmierung einer Achse. Im Highbyte (Bits 8..10) wird die Achsnr. übergeben.

4.4.2.1 Referenzfahrt

icSFRef $0050 W1 / C
Fährt die angegebene Achse auf den Referenz-Punkt. Die Richtung wird im Datenwort angegeben (**sfPos** oder **sfNeg**); das Highbyte hat folgenden Aufbau:

15	14	13	12	11	10	9	8
0	rmSWLS	rmPos	rmRef	rm			

Abb. 4.11 Aufbau Datenwort für Sonderfunktionen – Highbyte Referenzfahrt

Folgende Konstanten sind dafür definiert:

rm:	**$0F00**	normale Referenz-Fahrt
rmRef:	**$1000**	zusätzlich: Referenz-Puls suchen und anschließend die Achse um **apRefOffs** verfahren.
rmPos:	**$2000**	zusätzlich: neue Istposition aus dem Register **SFDist** setzen. Ist diese Konstante nicht angegeben, so wird die Istposition aus **apRefPos** übernommen.
rmSWLS	**$4000**	zusätzlich: Verfahrgrenzen **apLSDelta..** übernehmen.

Standard-Geschwindigkeit:
Wird als Geschwindigkeit **sfVStd** angegeben, so wird die Referenzfahrt mit der Geschwindigkeit **apVRef** ausgeführt.

Zur Programmierung und zum Ablauf der Referenzfahrt s. Kap. 5.3.1.

4.4.2.2 Asynchrones Fahren

icSFAsync $0051 W1 / C
Asynchrones Fahren - Die angegebene Achse wird während einer evtl. laufenden Interpolation zusätzlich um die in **SFDist** programmierte Distanz verfahren. Sie darf natürlich nicht selbst an dieser Interpolation beteiligt sein. Die Fahrtrichtung entspricht dem Vorzeichen von **SFDist**.

Standard-Geschwindigkeit:
Wird als Geschwindigkeit **sfVStd** angegeben, so wird die Bewegung mit der Geschwindigkeit **SFVCmd** ausgeführt.

Ab Version 2.4.67 kann zusätzlich im Highbyte die Nummer eines ADC-Eingangs (1..7) angegeben werden. Ein an diesem Eingang angeschlossenes Poti wird dann anstelle des normalen Potis (ADC0) verwendet.

Hinweis: *In diesem Fall ist darauf zu achten, daß im Lowword des Parameters **mpPotiInit** (Potibereich) ein Wert <> 0 steht!*

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

0	0	0	0	0	ADC-Nr	0	0	sfM49	0	sfVxxx

Abb. 4.12 Aufbau Datenwort für Sonderfunktion Asynchrones Fahren

Zur Programmierung s. Kap. 5.3.2.

4.4.2.3 Hand-Betrieb

icSFHand $0052 W1 / C
Hand-Betrieb - Die angegebene Achse wird während einer evtl. laufenden Interpolation zusätzlich mit der in **SFVCmd** programmierten bzw. mit der im Datenwort codierten Geschwindigkeit verfahren. Sie darf natürlich nicht selbst an dieser Interpolation beteiligt sein. Die Fahrtrichtung wird im Datenwort angegeben (**sfPos** oder **sfNeg**).

Standard-Geschwindigkeit:
Wird als Geschwindigkeit **sfVStd** angegeben, so wird die Bewegung mit der Geschwindigkeit **apVMax** ausgeführt.

Für diese Funktion gibt es eine Ausnahme der Regel, daß eine Achse erst dann wieder programmiert werden darf, wenn die aktuelle Sonderfunktion beendet ist:
Ist diese Funktion aktiv, so darf die Sonderfunktion erneut programmiert werden um die Geschwindigkeit zu ändern. Die neue Geschwindigkeit wird korrekt über die Rampe angefahren, auch bei Vorzeichenwechsel!

Beendet wird der Hand-Betrieb indem die Geschwindigkeit 0 vorgegeben wird oder über das Datenwort **sfBreak**.

Zur Programmierung s. Kap. 5.3.3.

4.4.2.4 Handrad

Durch den Anschluß eines Handrades / Encoders an den Drehgeber-Eingang der I332-ISA bzw. an unbenutzte Eingänge der I332-DC / AM167 können Achsen gezielt von Hand verfahren werden; ebenso ist ein elektronisches Getriebe realisierbar.

icSFHandrad $0054 W1 / C
Handrad-Betrieb - Im Datenwort wird im Highbyte die Nummer des Drehgeber-Eingangs angegeben, im Lowbyte der gewünschte Modus für die Geschwindigkeit (eine der Konstanten **sfV...**). Außerdem kann sfM49 (Bit 5) gesetzt sein (= Poti nicht berücksichtigen).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	Drehgeber-Nr				0	0	sfM49	sfLSOff	sfVxxx			

Abb. 4.13 Aufbau Datenwort für Sonderfunktion Handrad

Als Drehgeber sind die Nummern 0..7 für die Drehgeber auf der i332-DC / AM167 oder $0B für den Drehgeber auf der I332-ISA sowie $0E für das simulierte Drehgeber-Register (s.Kap. 4.8.3) zulässig.

Das Übersetzungsverhältnis wird in folgenden Achsregistern festgelegt:

icSetDG_my $0012 W2 / C 1000stel mm pro ...
icSetDG_pls $0013 W2 / C ... Anzahl Handrad-Impulse

Beendet wird der Handrad-Betrieb über das Datenwort **sfBreak**.

Ein Programmier-Beispiel findet sich im Kap. 5.3.4.

4.4.2.5 Joystick

Die Funktion ist ab Firmware-Version 2.2.30 verfügbar.

Der Anschluß von Joysticks erfolgt genauso wie ein Poti an den ADC-Eingängen der I332-Karte (s. Kap. 6): Mittelabgriff an den gewünschten ADC-Eingang, Spannungsversorgung (+5V/Gnd) an die beiden Widerstandsenden des Joysticks.

icSFJoystick $0056 W1 / C

Im Datenwort wird im Highbyte die Nummer des ADC-Eingangs (0..7) angegeben, im Lowbyte der gewünschte Modus für die Geschwindigkeit (eine der Konstanten **sfV...**). Außerdem MUSS sfM49 (Bit 5) gesetzt sein (= Poti nicht berücksichtigen).
Ab Version 2.2.31 KANN zusätzlich das Bit 4 gesetzt werden; dies bewirkt, daß die normale Endschalter-Behandlung unterdrückt wird, d.h. es wird keine Fehlermeldung erzeugt. Beim Überfahren eines Endschalters wird lediglich die Achse angehalten; durch entgegengesetzte Bewegung des Joysticks kann der Endschalter ganz normal freigefahren werden.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	ADC-Nr			0	0	sfM49	sfLSOff	sfVxxx			

Abb. 4.14 Aufbau Datenwort für Sonderfunktion Joystick

Beendet wird der Joystick-Betrieb über das Datenwort **sfBreak**.

Kennlinie:

Die Bestimmung der Geschwindigkeit erfolgt über eine Kennline mit 16 Intervallen = 17 Stützpunkten. Über diese Kennlinie wird die Spannung (0..5V) am AD-Wandler in eine Geschwindigkeit umgerechnet. Damit läßt sich das Geschwindigkeitsprofil, der Nullpunkt und die Maximalgeschwindigkeiten für die beiden Endlagen des Joysticks beliebig festlegen. Für jede Achse läßt sich eine eigene Kennlinie programmieren.

***NB**: d.h. die Kennlinie ist achsbezogen, nicht ADC-bezogen!*

Für jeden Stützpunkt läßt sich ein Wert zwischen -128..+127 entsprechend einer Geschwindigkeit -100%..+100% der programmierten Geschwindigkeit angeben.

Dazu dient der Befehl

icJSKennlinie $0016 W1 / C Stützpunkt für JS-Kennlinie

Im Highbyte des Befehls wird die Achsnr. übergeben; das Datenwort teilt sich ebenfalls in 2 Byte:
- Highbyte: Nr. des Stützpunkts (0..16)
- Lowbyte: Wert des Stützpunkts (-128..+127)

Hinweis *Für jede Achse ist die folgende Kennlinie bereits als Default vorgegeben; für einen ersten Test bzw. wenn dies den Erfordernissen genügt, ist es also nicht notwendig, die Kennlinie explizit zu programmieren:*

-128,-128,-112, -96, -64, -32, -16, 0, 0, 0, +16, +32, +64, +96,+112,+127,+127

Dies ergibt eine leicht Doppel-S-förmige Kennlinie –100% - +100% mit einem kleinen Plateau bei 0% in der Mitte des Stellbereichs für einen sicheren Stillstand der Achse in der Mittelstellung des Joysticks.

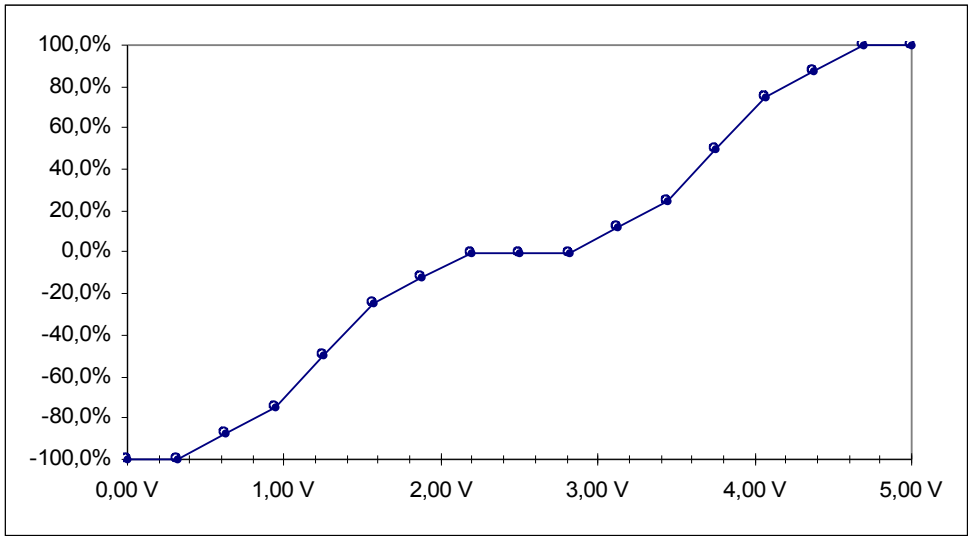


Abb. 4.15 Joystick-Kennlinie (Default)

4.5 Modale Daten - Gruppe D

Neben den Registern im Satzpuffer (Gruppe-B-Befehle) sind zur Programmierung eines Fahrsatzes weitere Register verfügbar. Diese Werte in diesen Registern müssen nicht für jeden Satz neu programmiert werden, sondern der jeweils aktuelle, zuletzt programmierte Wert wird beim Weiterschalten des Satzes automatisch in die entsprechenden Register des Satzpuffers übernommen.

Details zur Programmierung s. Kap. 5

4.5.1 Vorschub-Befehle

icSetVCmd $9000 W2 / D
Sollgeschwindigkeit für die nachfolgend eingegebenen Interpolationssätze. Sie gilt solange, bis sie durch erneute Eingabe überschrieben wird.

icGetVCmd $9080 R2 / D
aktuell gesetzte Sollgeschwindigkeit lesen

icGetVCur $9081 R2 / D
Ist-Geschwindigkeit der aktuellen Bahnbewegung

4.5.2 Spindel-Funktionen

Zur Unterstützung einer Arbeitsspindel stehen Befehle für die Drehzahl und die Schalteingänge der Spindel zur Verfügung. Die Parameter der Spindel werden in den Maschinendaten **mpSP_....** (s. Kap. 4.2.1.3) definiert.

icSetSCmd $9010 W2 / D
Spindeldrehzahl in U/min setzen; der Befehl unterstützt das S-Wort der DIN 66 025. Die Drehzahl (d.h. die entsprechende Spannung am DA-Ausgang) wird am Satzanfang gesetzt.

icGetSCmd $9090 R2 / D
aktuell gesetzte Spindeldrehzahl lesen - Solldrehzahl

icGetSCur $9091 R2 / D
aktuelle Spindeldrehzahl lesen - Istdrehzahl

icSetStdM $8410 W2 / D Standard-M-Funktionen setzen
Der Befehl unterstützt die Spindel-Funktionen M03, M04 und M05 der DIN 66 025. Im Highword bzw. Lowword wird in den beiden Lowbits die Funktion definiert, die am Satzanfang bzw. am Satzende wirksam sein soll. Der Befehl beeinflußt die in **mpSP_MPorts** definierten Ausgänge.

$0000	keine Änderung
$0001	Spindel ein, Rechtslauf (M03)
$0002	Spindel ein, Linkslauf (M04)
$0003	Spindel aus (M05)

Anmerkung: *Ist die Spindel vom Typ 1 (**mpSP_Data** = 1), dann erfolgt die Umschaltung der Drehrichtung über das Vorzeichen der Spindeldrehzahl (**icSetSCmd**). Die Werte $0001 bzw. $0002 bewirken dann beide nur das Einschalten der Spindel.*

icGetStdMCur $C885 R1 / D
Liefert die aktuell gültige Spindel-M-Funktion in den beiden Lowbits, Bits wie oben.

***Hinweis**:* *Spindeldrehzahl und Bedienung der Spindelausgänge sind voneinander unabhängig; das bedeutet z.B., daß mit M05 zwar die Ausgänge entsprechend gesetzt werden, die Ausgangsspannung für die Spindeldrehzahl jedoch nicht auf 0 gesetzt wird. Diese muß ggf. explizit programmiert werden!*

4.6 Achs-Register - Gruppe E

Die I332 besitzt eine Reihe von Registern, die Achspositionen enthalten. Sie dienen teilweise nur internen Zwecken und können daher nur gelesen werden (z.B. PID-Register).
Das wichtigste dieser Register ist das Positions-Register, das die aktuelle Position der Achsen enthält. Es kann beschrieben und gelesen werden.

4.6.1 Achsregister im Sollwertgenerator

icSetPos $0001 W2 / E
neue Achs-Position (Absolutposition) setzen, wird sofort wirksam, sollte daher nur bei stillstehenden Achsen aufgerufen werden

icGetPos $0081 R2 / E
momentane Achs-Position (Absolutposition, Sollposition) lesen

***Anmerkung**: Ist die Achse eine Schrittmotor-Achse, so ist die gelesene Position die Istposition; ist die Achse eine Servo-Achse, so weicht die tatsächliche Position von der gelesenen (Soll-)Position um den Schleppfehler **PIDFErr** (s.u.) ab!*

icGetPos0 $0085 R2 / E
momentane Achs-Position (Absolutposition, Istposition) lesen.

***Anmerkung**: Ist die Achse eine Schrittmotor-Achse, so ist die gelesene Position identisch zu der mit **icGetPos** gelesenen; ist die Achse eine Servo-Achse, so ist dies die tatsächliche (Ist-)Position incl. Schleppfehler **PIDFErr** (s.u.)!*

icGetIPLVCmd $00A7 R2 / E
aktuelle, vom Sollwertgenerator vorgegebene Geschwindigkeit

***Anmerkung**: Ist die Achse eine Schrittmotor-Achse, so ist dies die auch die Istgeschwindigkeit; ist die Achse eine Servo-Achse, so ist dies die Sollgeschwindigkeit für den PID-Lageregler.*

4.6.2 Achsregister für PID-Funktionen

Der PID-Algoritmus benutzt einen eigenen Registersatz, der aber nur gelesen werden kann. Diese Register müssen nicht notwendigerweise mit den Positionsregistern übereinstimmen! Mit Ausnahme des Schleppfehlerregisters sollten sie keinesfalls für irgendwelche Berechnungen herangezogen werden, sondern sind lediglich als Information zu betrachten.

icGetPIDCmdPos $00A0 R2 / E
Sollposition, Vorgabe für PID-Algorithmus

icGetPIDCurPos $00A1 R2 / E
Istposition, Rückmeldung der Drehgeber/Meß-System

icGetPIDFErr $00A2 R2 / E
momentaner Schleppfehler / Regelabweichung des PID-Algorithmus (Abweichung der Ist-Position von der mit **icGetPos** gelesenen Soll-Position).

icGetPIDVCmd $00A3 R2 / E
Sollgeschwindigkeit vom Sollwertgenerator

icGetPIDVCur $00A6 R2 / E
Istgeschwindigkeit, abgeleitet von der Rückmeldung der Drehgeber/Meß-System

4.7 Status-Informationen - Gruppe F

4.7.1 Status-Register

Das Status-Register ist ein Bitregister wie in Kap. 3 beschrieben und enthält diverse Bit-Flags, die verschiedene Betriebszustände anzeigen. Das High-Word (Bits 31..16) ist nicht belegt. Das Statusregister wird mit dem Befehl »**icStatus**« manipuliert.

icStatus **$C000** **Bitregister / F**

Anmerkung: *In der Datei I332.DEF sind zusätzlich zu den Bit-Nummern auch Bitflags als Wortkonstanten definiert nach folgendem Schema:*

in PASCAL-Schreibweise **stXXXF = 1 shl stXXXB**
oder
in C-Schreibweise **stXXXF = 1 << stXXXB**.

Diese Bits können nur gelesen werden:

stLSB **0** **Limit Switch Bit**
Eine der Achsen fuhr auf einen Endschalter, alle Achsen werden gestoppt.
Es wird auf einen Restart-Befehl (**icRestart..**) gewartet. Über den Befehl **icGetLSAktiv** können die betroffenen Achsen festgestellt werden.
Hinweis*:* *Der Befehl **icGetLSAktiv** liefert IMMER korrekte Werte; das Bit **stLSB** wird jedoch erst gesetzt, wenn versucht wird die entsprechende(n) Achse(n) zu fahren.*

stErrB **1** **Error Bit**
Ein Fehler ist aufgetreten; das Bit wird gelöscht durch das Lesen der Fehlernummer (Befehl **icGetErrNo**).

stOVB **2** **Overflow Bit**
Der Satzpuffer ist voll; es können keine weiteren Interpolations-Sätze mehr übergeben werden. Sobald ein Satz abgearbeitet ist, wird das Flag automatisch zurückgesetzt.

stMVB **3** **Move Bit**
mind. 1 Achse bewegt sich: Der Interpolator erzeugt für mindestens 1 Achse Schrittimpulse bzw. einen Regelsollwert <> 0.
Das Bit wird insb. auch dann zurückgesetzt, wenn alle Achsen per **stSSF** bzw. **stSFF** gestoppt sind.
Hinweis: *Dieses Bit sagt nichts darüber aus, ob noch unverarbeitete Fahrsätze im Interpolator vorhanden sind resp. ob alle Sätze verfahren wurden!*

stFEB **4** **Following Error Bit**
Der Schleppfehler einer Achse hat den Wert **apFEFatal** überschritten, alle Achsen werden gestoppt. Es wird auf einen Restart-Befehl (**icRestart..**) gewartet (nur I332-DC).

stBNEB **5** **Buffer not empty**
Der Satzpuffer ist nicht leer (d.h. enthält mindestens einen nicht bearbeiteten Satz).

stFEMaxB **6** **Following Error Max Bit**
Der Schleppfehler einer Achse hat den Wert **apFEMax** überschritten. Es wird nur dieses Bit (und das entsprechende Achsbit in **PIDInfo**) gesetzt. Es erfolgt keine weitere Aktion.

Diese Bits können geschrieben und gelesen werden:

stSSB **12** **Start/Stop Interpolation**
Die Abarbeitung von Interpolations-Sätzen wird freigegeben. Sind Interpolations-Sätze im Satzpuffer, so werden sie verfahren. Bei leerem Puffer passiert nichts. Neue Sätze werden dann sofort verfahren. Wird das Bit gelöscht, so wird eine evtl. laufende Interpolation unterbrochen, erneutes Setzen führt die Interpolation fort. Beim Starten und Stoppen werden die Beschleunigungs- und Verzögerungs-Rampen beachtet.

stSFB **8** **Start/Stop Sonderfunktion**
wie **stSSB**, jedoch für Sonderfunktionen

stSCB **13** **Single / Contingous Bit**
Einzelsatz-Betrieb aus/ein
Bit = 0: alle im Satzpuffer befindlichen Interpolations-Sätze werden nacheinander verfahren. Falls die Satzübergänge stetig sind, erfolgt kein Stop zwischen den Sätzen.
Bit = 1: nach jedem Satz löscht die I332 das **stSSB**, d.h. jeder Satz muß einzeln neu gestartet werden

Hinweis: *Bei der Abfrage von Bewegungszuständen sollten folgende Varianten genau unterschieden werden:*

1. *Stehen noch Interpolationssätze zur Verfügung bzw. wird eine Interpolation ausgeführt?*
   ➔ *In diesem Fall sollte das Bit stBNEB oder das Register icGetBlockCnt (s.u.) ggf. in Verbindung mit dem Bit stSSB abgefragt werden.*
2. *Wird eine Sonderfunktion ausgeführt?*
   ➔ *In diesem Fall sollte das Register icGetSFInfo (s.u.) ggf. in Verbindung mit dem Bit stSFB abgefragt werden.*
3. *Bewegen sich die Motoren?*
   ➔ *In diesem Fall sollte das Bit stMVB ggf. in Verbindung mit den Positionsbits im Register icGetPIDInfo (s.u.) abgefragt werden.*

4.7.2 sonstige Status-Informationen

icGetErrNo **$C100** **R1 / F**
liest die Nummer des zuletzt aufgetretenen Fehlers und setzt das **stErrB** im Status-Register zurück.

Mögliche Fehlernummern:

ieROMChkSum	= $0001:	Fehler im Programmspeicher
ieZeroDiv	= $0005:	Division durch Null
ieErr	= $0010:	unbekannter Fehler
ieRefHys	= $0011:	Fehler beim Referenzfahren
ieRefBack	= $0012:	Endschalter nicht gefunden
ieRefRel	= $0013:	Endschalter kommt nich frei
ieRefPls	= $0014:	Ref-Schalter nicht gefunden
ieEmergency0	= $0040:	Notstop-Routine aktiv, Modus 0
ieEmergency1	= $0041:	Notstop-Routine aktiv, Modus 1

Folgende Fehler werden nur durch Fehler in der Hardware verursacht (u.U. auch durch Störsignale) oder durch Fehlprogrammierung von Maschinendaten (insbesondere von nicht dokumentierten):

ieExc	= $xx02:	interner Prozessor-Fehler, xx = Nr. des Prozessorfehlers
ieIRQ	= $0003:	unbekannter Interrupt
ieTrace	= $0004:	Prozessortrace aktiv
ieTB	= $0008:	Zeitüberschreitung; dieser Fehler tritt auch bei der I332-PCI auf, wenn das Anschaltmodul AM167/AM168 nicht vorhanden ist bzw. nicht antwortet (Lichtleiter defekt / nicht angeschlossen).
iePID	= $0020:	Zeitüberschreitung PID-Regler
ieIPL	= $0021:	Zeitüberschreitung Interpolation

icGetPIDInfo $C800 R2 / F
In 4 Byte werden je 8 Bitflags für die Achsen 0..7 geliefert, die den Zustand des PID-Systems wiedergeben. In der Reihenfolge Highbyte..Lowbyte:

MSB:	Regelung der Achse ist freigegeben
. :	Achse ist in Position
. :	Achse meldet Schleppfehler
LSB :	Servo-Verstärker meldet »Bereit«

Für Achsen, die nicht als Servo-Achsen definiert sind, wird eine 0 als Flag geliefert.

icSetPIDEn $CC00 W0 / F
Bitflags für die Achsen 0..7 werden zum Lowbyte des Befehls addiert; die Regelung für diese Servo-Achsen wird aktiviert und die Verstärker erhalten das Freigabe-Signal, sofern das Bit **apDC.dcEnOutB** gesetzt ist.

icGetLSAktiv $D001 R1 / F
Liefert die aktiven Endschalter (unter Berücksichtigung der **apLS-Bits**!) in Bit 15..0 für die Endschalter -C..-X (Bit 15..8) und +C..+X (Bit 7..0)
d.h. Bit = 1: Endschalter aktiv, Bit = 0: Endschalter nicht aktiv oder nicht vorhanden
***Hinweis**:* *Die Werte sind bereits beim Einschalten korrekt; das Bit **stLSB** wird jedoch erst gesetzt, wenn versucht wird die entsprechende(n) Achse(n) zu fahren.*

icGetBlockCnt $B100 R2 / F
Liefert die Anzahl der noch im Interpolations-Puffer befindlichen Sätze.

icGetRefInfo $C883 R1 / F
Liefert im Lowbyte Bitflags für alle referenzierten Achsen

Den Status eventuell aktiver Sonderfunktionen liefert der Befehl
icGetSFInfo $C900 R1 / C
Liefert die derzeit aktiven Achsen einer gewünschten Sonderfunktion als Bitflags im Highbyte. Gesetzte Bits im Lowbyte repräsentieren Achsen, bei denen irgendeine Sonderfunktion aktiv ist. Details s. Kap. 4.4.

4.8 Ein-/Ausgänge - Gruppe G

Die folgenden Kapitel beschreiben die freie Programmierung der Ein- und Ausgänge zu allgemeinen Steuerungszwecken, d.h. soweit sie nicht durch Achsfunktionen oder andere Funktionen (z.B. Arbeitsspindel) belegt sind.

Übersicht über die unterstützten Ein-/Ausgänge

Interpolator-Karte I332-ISA
- 8 AD-Wandler ADC0 bis ADC7
- 16 Schrittmotor-Ausgänge
- 16 Endschalter-Eingänge

ISA-Zusatzkarte I332-DC
- 8 Drehgeber-Eingänge AX/AX*/BX/BX* bis AC/AC*BC/BC*
- 8 Sollwert-Ausgänge (DA-Wandler) DC_X..DC_C
- 8 Freigabe-Ausgänge FREIX bis FREIC
- 8 Referenzpuls-Eingänge REFX bis REFC
- 8 Bereit(Ready)-Eingänge BEREITX bis BEREITC

ISA-Zusatzkarte I332-IO
- 30 Einzelbit-Eingänge
- 30 Einzelbit-Ausgänge

Interpolator-Karte I332-PCI

alle Ein-/Ausgänge befinden sich auf den Zusatz-Moduln AM167 bzw. AM168:
- 16 Endschalter-Eingänge
- 4 AD-Wandler ADC0 bis ADC3

Schrittmotor-Version: bis zu
- 16 Schrittmotor-Ausgänge

Servo-Version: bis zu
- 8 Drehgeber-Eingänge AX/AX*/BX/BX* bis AC/AC*BC/BC*
- 8 Sollwert-Ausgänge (DA-Wandler) DC_X..DC_C
- 8 Freigabe-Ausgänge FREIX bis FREIC
- 8 Referenzpuls-Eingänge REFX bis REFC
- 8 Bereit(Ready)-Eingänge BEREITX bis BEREITC

Zusatz-Modul CAN-IO

je Modul:
- 16 Einzelbit-Eingänge
- 16 Einzelbit-Ausgänge

bis zu 4 Module einsetzbar, d.h. bis zu 64 Ein- und 64 Ausgänge

CAN-Module nach Kundenwunsch: weitere Drehgeber, AD/DA-Wandler etc.

***Achtung**: je nach Ausbau der Karten sind die Bausteine nur teilweise bestückt!*

4.8.1 Potentiometer

Die I332 stellt einen AD-Eingang zur Verfügung (ADC0), an den ein Geschwindigkeitspoti angeschlossen werden kann, mit dem die programmierte Geschwindigkeit von 0-100% geregelt werden kann. Wird dies nicht gewünscht oder ist dieser Eingang nicht beschaltet, so kann (bzw. muß!) durch den Befehl **icSetOvr** diese Eingangs-Information überschrieben werden. Alternativ kann das Maschinendatum **mpPotiInit** (s. Kap. 4.2) entsprechend gesetzt werden; dessen Highbyte intialisiert bei jedem Restart-Befehl **icRestart..** das Poti-Override.

icSetOvr $9800 W0 / G
Poti-Override: Ein Wert von 1..255 wird auf das Lowbyte addiert und repräsentiert einen Potiwert von 0,39..200(!)%. Der Wert 0 aktiviert den Poti-Eingang.

Die I332 stellt sicher, daß auch bei bei einem Override von 200% die in den Maschinendaten eingestellten Maximalwerte für die Geschwindigkeit nicht überschritten werden.

Das Poti bzw. dieser Override-Wert ist sowohl für Interpolations- als als auch für Sonderfunktionen wirksam.

Der Befehl ist sofort wirksam!
Ein Befehl **icRestart..** initialisiert den Override mit dem Maschinendatum **mpPotiInit**.

4.8.2 Port-Befehle: Einzelbit-Ein-/Ausgabe

Allgemeines:
Alle Port-Befehle sind R1- bzw. W1-Befehle, d.h. mit einem Befehl können 16 Portbits gleichzeitig angesprochen werden.
Mit den Port-Befehlen können die Schrittmotor- und Freigabe-Ausgänge (soweit sie nicht durch Achsfunktionen belegt sind), die Endschalter-, Referenz-, und Bereit-Eingänge sowie die Ports auf der I332-IO-Karte angesprochen werden; da die IO-Karte insgesamt 30 Eingänge (IN01 bis IN30) und 30 Ausgänge (OUT01 bis OUT30) enthält, wurden sie in 2 Portgruppen 1 und 2 aufgeteilt: Ports 1-16 und 17-32 (Port 31 und 32 sind am Stecker nicht verfügbar und deshalb auch nicht bestückt).

***Hinweis**:* *Bei der PCI-Version der I332 befinden sich die IOs 01..32 auf den beiden ersten CAN-Modulen.*

Um die Programmierung der Ausgänge flexibel zu halten, können sie auf zweierlei Weise angesprochen werden:

Zugriffsart a): Innerhalb eines Interpolations-Satzes (quasi als M-Funktion), s. Abb. 4.7
Zugriffsart b): direkt vom PC

Um zu verhindern, daß sich die beiden Zugriffsarten gegenseitig beeinflussen, existiert ein sog. Port-Assignment-Register (**OutPAR**), das für jedes Portbit die derzeitige Zugriffsart definiert. Dieses Register darf jederzeit (!) geändert werden. Damit ist es möglich, während einer laufenden Interpolation auf die Port-Ausgabe Einfluß zu nehmen. Abb. 4.16 verdeutlicht die Zusammenhänge.
Um die Ausgabe-Ports (**OutPort**) jederzeit auch lesen zu können, werden alle Ausgaben parallel in einen Ausgabe-Puffer (**OutBuf**) geschrieben.

Ein Wert von 0 bzw. 1 im Ausgabe-Register bedeutet Masse-Pegel bzw. eine positive Spannung entsprechend UExt am zugehörigen Ausgang.
Bei den TTL-Eingängen IN01 bis IN30 entspricht der gelesene Wert dem Eingangspegel (0/1 = Masse bzw. 5V).

Übersicht über die Register zur Programmierung der Ein- und Ausgänge:

Registername	Beschreibung
MPortBeg bzw. **MPortEnd**	Diese Register befinden sich in jedem Satz des Interpolations-Puffers und legen die Zustände der Ausgänge am Beginn bzw. am Ende eines Satzes fest (s. Abb. 4.7).

Die folgenden Register befinden sich im Block I / O (s. Abb. 3.1):

OutBuf	Dieses Register enthält eine Kopie von **OutPort**
MOutBuf	Dieses Register wird vom Interpolator entsprechend **MPortBeg** bzw. **MPortEnd** aktualisiert.
OutPAR	Die Bits in diesem Register legen fest, ob die in den Interpolations-Sätzen programmierten Werte (**MOutBuf**) nach **OutPort** geschrieben werden (Bit = 1) oder der PC direkt auf die Ausgänge zugreifen darf (Bit = 0).

Dies sind die physikalischen Register:

OutPort	Dies ist der physikalische Baustein, der sich nur beschreiben läßt
InPort	Dies ist der physikalische Nur-Lese-Baustein für die Eingänge

Initialwerte / Maschinendaten:

Beim Einschalten bzw. bei einem Restartbefehl **icRestart1X** wird das Register **MOutBuf** mit den Maschinen-Daten **mpOutDef**, das Register **OutPAR** mit **mpOutPAR** initialisert. Bei allen anderen Restart-Befehlen wird außerdem auch das Register **OutBuf** und das physikalische Register **OutPort** mit **mpOutDef** initialisiert.

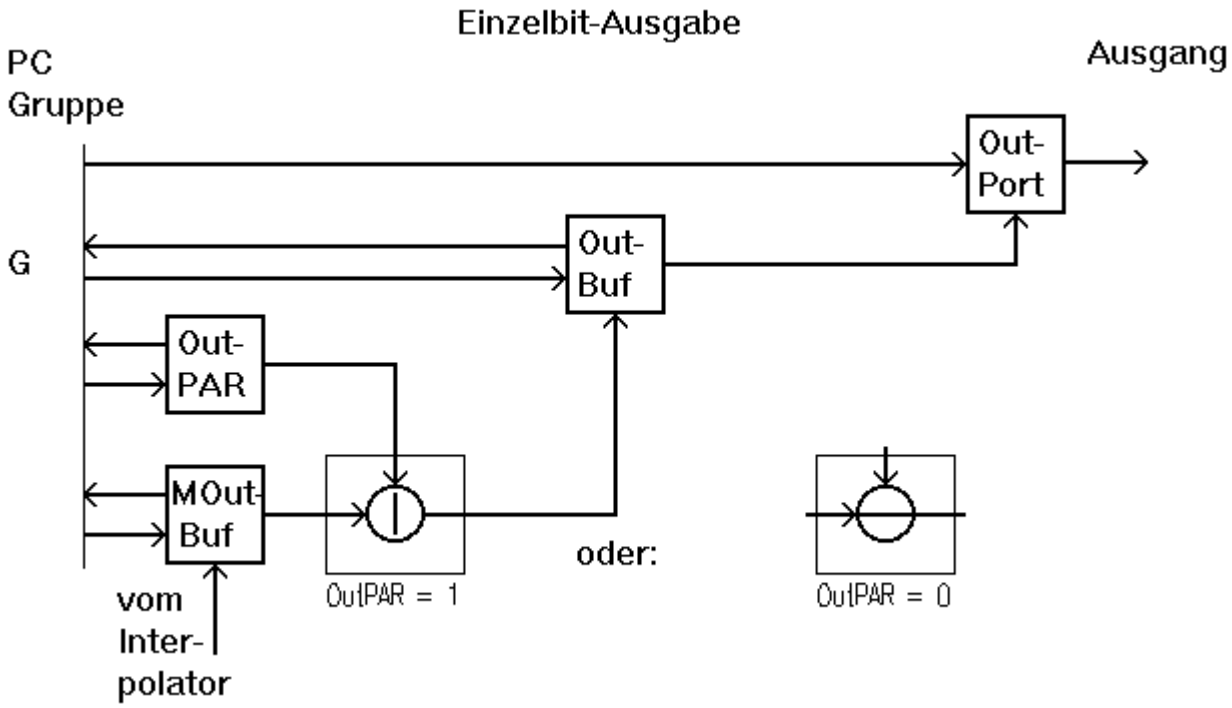


Abb. 4.16 Bedienung der Ausgänge (schematisch)

Die Port-Befehle setzen sich zusammen aus einem der beiden Basis-Befehle für die Zugriffsart plus Konstanten für die Auswahl der Portgruppe, Auswahl des Registers, Bits setzen, löschen etc....

allg. Konstanten für die Portbefehle:

Konstanten für die Portgruppen (Bits 4..6 des Befehlsworts)

pgSM	**$0000**	I332-SM-Ausgänge PulsX / RtgX .. PulsC / RtgC
pgLS	**$0010**	I332-Endschalter-Eingänge +X..+C/-X..-C
pgEn	**$0020**	I332-DC-Freigabe-Ausgänge X..C, Bits 8..15 undefiniert
pgRef	**$0020**	I332-DC-Referenz-Eingänge X..C, Bits 8..15 undefiniert
pgRdy	**$0030**	I332-DC-Bereit-Eingänge X..C, Bits 8..15 undefiniert
pgIO1	**$0060**	I332-IO-Karte bzw. CAN-IO: IN/OUT01..16
pgIO2	**$0070**	I332-IO-Karte bzw. CAN-IO: IN/OUT17..32

***Anmerkungen**:*

- *pgSM: An die als IO-Ports benutzen Ausgänge dürfen keine Schrittmotore angeschlossen sein; im Achsparameter **apChar** darf die Achse nicht als Schrittmotor definiert sein.*
- *pgEn: An die als IO-Ports benutzen Ausgänge dürfen keine Achsverstärker angeschlossen sein; im Achsparameter **apDC** darf das Bit **dcEnOutB** nicht gesetzt sein.*

Konstanten für die Manipulation (Bits 0..1 des Befehlsworts)

pmPut	**$0000**	Portbits entspr. Datenwort setzen
pmClr	**$0001**	für alle gelöschten Bits im Datenwort: Portbits löschen
pmSet	**$0002**	für alle gesetzten Bits im Datenwort: Portbits setzen
pmChg	**$0003**	für alle gesetzten Bits im Datenwort: Portbits invertieren nur für Zugriffsart b) und Register **OutPAR**

Basisbefehl für die Zugriffsart a)
icSetMPorts **$D200** **W1 / B**

Bei Zugriffsart a) läßt sich zusätzlich festlegen, ob die neue Portbelegung am Satzanfang oder am Satzende (s. Abb 4.5) wirksam werden soll (Bits 2..3 des Befehlsworts):
psBeg **$0000** wirksam am Satzanfang (Register **MPortBeg** schreiben)
psEnd **$0008** wirksam am Satzende (Register **MPortEnd** schreiben)

Ein Befehl für Zugriffsart a) setzt sich also aus folgenden Komponenten zusammen (s. Abb. 4.17):

icSetMPorts + pgIO... + ps... + pm...

Abb. 4.17 Format: icSetMPorts

Basis-Befehle für die Zugriffsart b) und den Zugriff auf das Port-Assignment-Registers (PAR)
icSetPorts **$D100** **W1 / G**
icGetPorts **$D180** **R1 / G**

Bei diesen Port-Befehlen muß zusätzlich das gewünschte Register angegeben werden (Bits 2..3 des Befehlsworts):

prOutBuf **$0004** Zugriff auf den Ausgabe-Puffer (**OutBuf**): Zugriffsart b)
Sind bei einem Schreibbefehl die entsprechenden Bits im Register **OutPAR** = 0 (Zugriffsart a)), so hat der Befehl auf diese Bits keinen Einfluß!

prOutPAR **$0008** Zugriff auf Port-Assignmet-Register **OutPAR**
Bit = 0/1: Zugriffsart a) / b)
Wird in diesem Register ein Bit von 1 nach 0 geändert (Wechsel der Zugriffsart von b) nach a)), so erscheint am zugehörigen Ausgang der im Register **MOutBuf** für diesen Ausgang programmierte Pegel.

prPort **$000C** direkter Zugriff auf den Port: Eingänge (**InPort**) lesen bzw. Ausgänge (**OutPort**) schreiben
Nur für Eingänge sinnvoll, Ausgänge werden praktisch sofort wieder mit dem Wert im Register **OutBuf** überschrieben!

Folgende Befehls-Kombinationen sind damit sinnvoll:

icSetOutBuf **$D104** **icSetPorts + prOutBuf**
icGetOutBuf **$D184** **icGetPorts + prOutBuf**

icSetOutPAR **$D108** **icSetPorts + prOutPAR**
icGetOutPAR **$D188** **icGetPorts + prOutPAR**

icSetOutPort **$D10C** **icSetPorts + prPort**
icGetInPort **$D18C** **icGetPorts + prPort**

Zu diesen Befehlen muß jeweils noch eine der Konstanten **pg...** und **pm...** addiert werden.

Ein Befehl für Zugriffsart b) bzw. für das Register **OutPAR** setzt sich also aus folgenden Komponenten zusammen (Abb. 4.18):

icSetPorts + pg... + pr... + pm...
icGetPorts + pg... + pr... + pm...

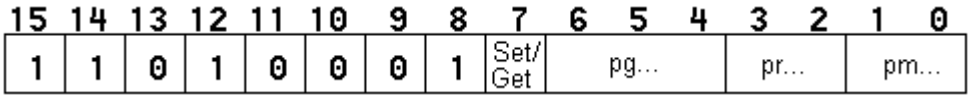


Abb. 4.18 Format icSetPorts **/** icGetPorts

4.8.3 IO-Bausteine

Alle IO-Bausteine, die auf den Karten I332, I332-DC und I332-IO bzw. auf dem Anschaltmodul AM167/168 vorhanden sind, lassen sich für allgemeine Steuerungszwecke verwenden, falls sie nicht durch Achsfunktionen belegt (oder auf den Karten nicht bestückt) sind. Auch wenn die Bausteine teilweise den Achsen zugeordnet sind, wurden die Befehle NICHT unter die Achsbefehle eingeordnet, um deutlich zu machen, daß damit keine Achsfunktionen programmiert werden sollten.

Basis-Befehle

icSetIODev	**$D800**	**W1 / G**	**Set IO-Device**
icGetIODev	**$D880**	**R1 / G**	**Get IO-Device**

icGetDG $D880 R1 / G
Lesen eines Drehgebers auf der I332-DC-Karte. Die Nummer des Drehgebers (0..7 entspricht den Drehgeber-Eingängen AX/AX*/BX/BX* bis AC/AC*/BC/BC* auf der I332-DC-Karte) wird auf das **Low-Byte**(!) addiert (Bit 0..2). Zusätzlich läßt sich unter der Nummer $0B der Drehgeber auf der I332-ISA-Karte abfragen; außerdem steht unter der Nummer $0E ein Wort-Register zur Verfügung, das der PC lesen und schreiben kann. Damit kann z.B. beim Handradbetrieb ein Drehgeber simuliert werden, der vom PC aus beschrieben wird:

icSetDG_PC	**$D80E**	**W1 / G**
icGetDG_PC	**$D88E**	**R1 / G**

Wertebereich der Drehgeber: 0..65535 (16 Bit)

icSetDAC	**$D840**	**W1 / G**
icGetDAC	**$D8C0**	**R1 / G**

Schreiben / Lesen eines DA-Wandlers auf der I332-DC-Karte. Die Nummer des DA-Wandlers (0..7 entspricht den Sollwert-Ausgängen DC_X..DC_C auf der I332-DC-Karte) wird auf das **Low-Byte**(!) addiert (Bit 0..2).

Anmerkung: *Der Befehl **icSetDAC** sollte nur für solche DA-Wandler benutzt werden, die NICHT in den Maschinendaten als Servo-Achsen (**apChar[acDCB]** = 1) oder als Spindel (**mpSP_Data** <> 0) definiert sind , da ansonsten der PID-Regler immer dazwischen funkt....!*

Wertebereich der DA-Wandler: 0..65535 (16 Bit)
die 12 Highbits werden an den DA-Wandler übergeben, die 4 Lowbits werden ignoriert. Es wird eine Spannung im Bereich -10V bis +10V erzeugt.

icGetADC $D890 R1 / G
Lesen eines AD-Wandlers auf der I332-Karte, Nummer des AD-Wandlers wird auf das **Low-Byte**(!) addiert (Bit 0..2).

Anmerkung: *Die ADC-Anschlüsse sind auf den Stecker herausgeführt, s. Steckerbelegung. ADC0 ist für das Poti reserviert, alle anderen werden bisher von der I332-Software nicht verwendet (was aber in Zukunft nicht so bleiben muß!).*

Außerdem steht unter der Nummer $0E ein Wort-Register zur Verfügung das der PC lesen und schreiben kann. Damit kann z.B. ein Poti simuliert werden, das vom PC aus beschrieben wird:

icSetADC_PC	**$D81E**	**W1 / G**
icGetADC_PC	**$D89E**	**R1 / G**

Wertebereich der AD-Wandler: 0..1023 (10 Bit)

4.8.4 sonstige Ein-/Ausgabe-Befehle

icGetLS $D000 R1 / G
Liefert den Zustand der Endschalter-Eingänge (direkter Zugriff auf die Hardware!) in Bit 15..0 für die Eingänge ES-C..ES-X (Bit 15..8) und ES+C..ES+X (Bit 7..0). Die Polarität der Endschalter wird berücksichtigt, nicht vorhandene Endschalter werden jedoch nicht ausmaskiert;
d.h. Bit = 1: Endschalter nicht vorhanden oder aktiv, Bit = 0: Endschalter nicht aktiv

4.9 sonstige Befehle

Hier sind einige Befehle aufgelistet die sich nicht nahtlos in die Gruppeneinteilung einfügen.

icAxisOvl $6000 W0
Ausgabe der Achse (A) erscheint auch am Ausgang des Motors (M) → beide Achsen fahren synchron. **apStep**/**apmm** müssen für beide Achsen gleich sein. Die Nr. der Achse (A) wird auf das Lowbyte, die Nr. der Achse (M) auf das Highbyte addiert, z.B. bedeutet **icAxisOvl**+$0102 = $6102: Die Ausgabe an den Motor der Achse 1 ist die gleiche wie für die Achse 2 programmiert, aber nicht umgekehrt!!! Um die Zuordnung aufzuheben, wird (A) = (M) programmiert. Im Beispiel wird durch **icAxisOvl**+$0101 = $6101 die Zuordnung wieder aufgehoben.
Dieser Befehl ist sofort wirksam!

icGetVerNo $F000 R2
Versions-/Revisionsnr. der Software in der Form [v v v v r r r r]
Die einzelnen Ziffer sind als Hexziffern zu interpretieren,
z.B.: $01060015 = Version 1.6, Revision 15
Die Hauptversionsnr. steht für die Hardware-Version:

01vvrrrr:	ISA-Version1 - wird nicht mehr gepflegt
02vvrrrr:	ISA-Version2
03vvrrrr:	VME-Bus
04vvrrrr:	PCI-Version
05vvrrrr:	serielle Version mit VG64-Anschluß

icGetVerDat $F001 R2
Versionsdatum der Software in der Form [d d m m j j j j]
z.B.: $08111993 = 8.11.1993

5 Programmierung der I332

In diesem Kapitel soll ein allgemeiner Überblick über das Prinzip der Programmierung und die Möglichkeiten gegeben werden. Die Code-Beispiele basieren auf der Programmiersprache Delphi unter Windows 9x/ME/NT/2000/XP und benutzen die I332.DLL.

***Hinweise**:* *Alle Code-Beispiele basieren auf den in Kap. 3.1 beschriebenen DLL-Funktionen; etliche der hier beschriebenen Beispiele stehen in der DLL aber bereits als fertige Funktionen zur Verfügung. Für Details wird auf das Handbuch der I332.DLL (I332_DLL.PDF) verwiesen.*
Fast alle Funktionen der I332.DLL erwarten als Parameter ein „I332Handle"; dieses wird als Ergebnis der Initialisierungsfunktion `I332IFCInit` *bei erfolgreicher Initialisierung zurückgeliefert.*
Fast alle Funktionen der DLL liefern als Rückgabewert einen (negativen) Fehlercode. In allen Beispielen wurde jedoch auf die Überprüfung des Funktionsergebnisses der Übersichtlichkeit halber verzichtet. In einer Anwendung sollte diese Überprüfung natürlich erfolgen.
*Soweit nicht anders vermerkt, haben alle Achs- und Maschinen-Parameter (**ap...** bzw. **mp...**) ihren Defaultwert.*

Prinzipiell werden 2 Arten von Fahrfunktionen unterschieden

1. Interpolationsfunktionen
2. Sonderfunktionen

Interpolationsfunktionen sind einzelne Fahrsätze, die in der Reihenfolge der Übergabe in einen Ringpuffer (Satzpuffer) geschrieben werden und nacheinander abgearbeitet werden.

Zusätzlich steht für jede Achse ein einzelner Puffer für Sonderfunktionen zur Verfügung. Eine Achse kann Sonderfunktionen parallel zu Interpolationsfunktionen ausführen, falls sie nicht selbst an der Interpolation beteiligt ist.

Zu beachten ist, daß sowohl bei Interpolationsfunktionen als auch bei Sonderfunktionen die Geschwindigkeit von einem externen Poti beeinflußt werden kann (ADC0-Anschluß am 37pol-SubD-Stecker). Wird dies nicht gewünscht, kann temporär per Befehl **icSetOvr** ein Poti-Wert von 100% fest vorgegeben werden (s.a. Kap. 4.8.1):

```
I332Write0(I332Handle,icSetOvr+$80);
```

Die Einstellung ist bis zum nächsten Restart-Befehl gültig. Ist kein Poti angeschlossen sollte statt dessen der Parameter **mpPotiInit** (s. Kap. 4.2.1) entsprechend gesetzt werden.

Die folgenden Code-Beispiele sind teilweise dem Demo-Programm „I332Demo" entnommen.

5.1 Initialisierung

Die Initialisierung der I332 besteht aus 2 Teilen

1. Initialisierung der Schnittstelle
2. Initialisierung der Maschinendaten

5.1.1 Initialisierung der Schnittstelle

Dies erledigt die DLL-Funktion `I332IFCInit` (s. Kap. 3.1.1).
Das von der Funktion als Ergebnis gelieferte I332Handle sollte in einer globalen Variablen zur Benutzung bei weiteren Funktionsaufrufen gespeichert werden.

5.1.2 Initialisierung der I332

Wenn die Schnittstelle zur I332 initialisiert wurde, sollten als nächstes die Maschinen- und Achsparameter initialisiert werden. Da diese Parameter die grundlegenden Eigenschaften der Maschine definieren, sollte dies nur ein einziges Mal, und zwar an dieser Stelle, passieren. Die Initialisierung muß mit dem Befehl **icInitMP** abgeschlossen werden.

```
      { Maschinendaten:}
      I332Write2(I332Handle,icSetMP+mp...,<Wert für mp...>);
      { weitere Maschinendaten }

      { Achsparameter:}
      I332Write2(I332Handle,icSetAP+ap...+<AchsNr>*256,<Wert      für
ap...>);
      { weitere Achsparameter }

      {Abschluß:}
      I332Write0(I332Handle,icInitMP);
```

***Hinweise**:* *Folgende Daten sollten auf jeden Fall programmiert werden:*

- ***mpInitPoti**, **apChar**, **apLS** und (falls Servomotoren zum Einsatz kommen) **apDC**.*
- *Für Achsen, die nicht bestückt sind, sollten die Achsparameter **apChar**, **apLS**, und **apDC** auf 0 gesetzt werden, um Fehler durch Einlesen nicht beschalteter Eingänge zu vermeiden.*

5.1.3 Beispiel: Initialisierung

```
VAR I332Handle: LONGINT;            { globale Variable }
BEGIN
  I332Handle = I332IFCInit(ifcMode,0);
  IF I332Handle < 0 THEN <...Fehlermeldung...>
  ELSE BEGIN
    I332Write2(I332Handle,icSetMP+mpInitPoti,$80008000); {Poti aus}

    {alle Endschalter inaktiv (keine ES angeschlossen)}
    I332Write2(I332Handle,icSetAP+apLS+0*256,0);
    I332Write2(I332Handle,icSetAP+apLS+1*256,0);
    I332Write2(I332Handle,icSetAP+apLS+2*256,0);
    I332Write2(I332Handle,icSetAP+apLS+3*256,0);
    I332Write2(I332Handle,icSetAP+apLS+4*256,0);
    I332Write2(I332Handle,icSetAP+apLS+5*256,0);
    I332Write2(I332Handle,icSetAP+apLS+6*256,0);
    I332Write2(I332Handle,icSetAP+apLS+7*256,0);

    {für die meisten Beispiele: keine weiteren Initialisierungen,
```

```
          d.h. es gelten die Defaultwerte,
          insb. alle Achsen Schrittmotoren}

        {für die Beispiele zur Port-Ausgabe:
         Initialisierung aus den entsprechenden
         Kapiteln einfügen (Kap. 5.2.5.2 bzw. Kap. 5.5)}

        {Ende der Initialisierung:}
        I332Write0(I332Handle,icInitMP);

            END;
          END;
```

***Hinweis**:* *Im Beispielprogramm I332DEMO wird zwar nach dem hier vorgestellten Schema verfahren, jedoch ist die Initialisierung der I332 etwas allgemeiner gehalten: Die Initialisierung übernimmt die DLL-Funktion I332XmtCfg(); die Maschinen-Daten werden als Datei I332.INI übergeben.*

5.2 Interpolation

Das Programmieren eines Interpolationssatzes erfordert die Übertragung der notwendigen Daten: Distanz zum nächsten Punkt, Geschwindigkeit, evtl. weitere Angaben (z.B. Kreismittelpunkt) sowie die Art der Interpolation (linear, zirkular, Verweilzeit). Diese Angaben werden automatisch in einen internen FIFO-Puffer geschrieben; dessen Verwaltung obliegt der I332. Vor dem Programmieren eines Interpolationssatzes sollte lediglich das Status-Bit »**stOVB**« getestet werden. Solange es gesetzt ist, ist der Satzpuffer gefüllt, und es sollten keine weiteren Sätze mehr übertragen werden.
Für die folgenden Beispiele wird von Standardachsparametern ausgegangen, d.h. insbesondere daß die Achsen 0-2 als Bahnachsen definiert sind, d.h. **apChar**.**acPathB** = 1; für die Achsen werden die Bezeichnungen XYZUVABC benutzt.

Schema:

0. ggf. Warten bis Platz im FIFO-Puffer frei ist
1. Übergabe der Satzdaten in beliebiger Reihenfolge
   Satzdaten können sein:
   - Bahngeschwindigkeit: nur erforderlich bei Änderung, d.h. wird in einem Satz keine Bahngeschwindigkeit programmiert, so gilt die zuletzt programmierte weiter
   - Satznummer (optional)
   - Bahnparameter: Distanzen der einzelnen Achsen bzw. Position des Mittelpunkts bzw.
   - Verweilzeit
   - Spindeldrehzahl
   - Ein-/Ausgabe-Funktionen
2. Befehl für die gewünschte Interpolationsart (**icIPL...**): Der Interpolationsbefehl beendet die Übergabe der Satzdaten und übernimmt diese in den internen Verfahrsatzpuffer.
3. ...nächster Satz...

***Anmerkungen**:*
- *Solange der FIFO-Puffer nicht voll ist, können weitere Sätze übertragen werden, auch während vorherige Sätze bereits verfahren werden.*
- *Abgearbeitete Sätze werden aus dem FIFO-Puffer gelöscht!*
- *Eine programmierte Geschwindigkeit gilt solange, bis ein neuer Wert programmiert wird.*
- *Der Befehl **icIPL**... kennzeichnet gleichzeitig das Ende der Satzdaten. Im übrigen können die Satzdaten in beliebiger Reihenfolge übertragen werden.*

- *Kontinuierliches Fahren:*
  Die I332 versucht, aufeinanderfolgende Sätze ohne Geschwindigkeitsreduzierung zu verfahren (kontinuierliches Fahren). Falls dies nicht möglich ist (Geschwindigkeitseinbrüche während des Verfahrens), kann dies folgende Ursachen haben:
  - *Die Satz-Übergänge sind zu "eckig": Die Ecke kann aufgrund der zulässigen Beschleunigungs- bzw. Verzögerungswerte (**apAcc** / **apDec**) nicht mit der gewünschten Geschwindigkeit gefahren werden.*
  - *Die Sätze sind zu klein: Innerhalb eines Satzes ist es nicht möglich von der gewünschten Geschwindigkeit auf die zulässige Geschwindigkeit des nächsten Satzes zu verzögern: Der komplette Satz wird dann bereits mit der geringeren Geschwindigkeit verfahren.*
  - *Die Sätze werden nicht schnell genug an die I332 übergeben: Am Ende eines Satzes steht der nächste Satz noch nicht zur Verfügung. D.h. die Sätze müssen so groß sein, daß während der Verfahrzeit eines Satzes der PC den folgenden Satz übergeben kann.*
  - *Die I332 hat eine Satz-Zykluszeit von ca. 3ms; d.h. wenn ein Satz vollständig an die I332 übergeben wurde, vergehen ca. 3ms bis dieser Satz auch tatsächlich verfahren werden kann; d.h. die Verfahrzeit der Sätze muß größer sein als 3ms.*
    Achtung: Die Kommunikation (z.B das Abfragen von Status- und/oder Positionsdaten) verzögert die Satz-Vorbereitungsphase in der I332! Das Anwendungsprogramm muß daher Sorge tragen, daß das Abfrageintervall groß genug ist (ca. 100ms); ggf. muß das entspr. dem Anwendungsfall angepaßt werden.
  - *Beim Einsatz von Servomotoren:*

*Ein Satzübergang bzw. der Start eines Satzes wird solange verzögert, bis am Satzende der Schleppfehler in allen Achsen kleiner als **apInPos** ist (Genauhalt). (s.a. Einstell-Hinweise zum PID-Regel-Algorithmus)*

- *Ähnliches gilt für den Fall, daß die programmierte Bahngeschwindigkeit nicht erreicht wird. Dabei ist zusätzlich folgendes zu beachten:*
  - *Die Auflösung für die Beschleunigung/Verzögerung (**apAcc** / **apDec**) beträgt ca. 10mm/sec². Kleinere Werte führen dazu, daß die Achsen nicht beschleunigen.*
  - *Wenn die Achsen zwar beschleunigen, aber die Bahngeschwindigkeit nicht erreicht wird, kommen prinzipiell ebenfalls die ersten 3 Ursachen von oben in Frage.*
  - *Es kann auch sein, daß der Achsparameter **apVMax** einer der beteiligten Achsen zu klein ist, und dadurch die Bahngeschwindigkeit begrenzt.*
  - *Da die I332 alle gegebenen Parameter auf die Bahn bezieht (Achsen mit **apChar.acPathB** = 1), können diese Werte sehr klein werden, wenn die Bahnlänge sehr klein im Verhältnis zu den Distanzen der einzelnen Achsen ist. Im Extremfall werden **apAcc** bzw. **apDec** zu 0, was ebenfalls zur Folge hat, daß die Achsen nicht beschleunigen.*

- *Ein drittes Kriterium, das sich u.U. in scheinbaren Wartezeiten zwischen zwei Sätzen bemerkbar macht, sind extrem unterschiedliche Werte für **apStop** und **apDec**. Insbesondere gilt dies, wenn **apStop** wenig größer ist als 0, **apDec** dagegen relativ groß.*

5.2.1 Allgemeine Funktionen

Warten, bis Platz im FIFO-Puffer frei:

```
REPEAT
  { Lowwort vom Status lesen: }
  I332Read1(I332Handle,icStatus or brModeGetL,I332Status);
UNTIL (I332Status and stOVF) <> 0;
```

***Hinweis**:* *Die Interpolationsfunktionen der I332-DLL überprüfen dieses Statusbit automatisch und kehren ggf. mit einer Fehlermeldung zurück.*

Das Status-Bit »**stSSB**« hat die Funktion einer Start/Stop-Taste. Damit kann die Achs-Bewegung beliebig angehalten bzw. fortgesetzt werden:

Interpolation starten bzw. weiterführen: **stSSB** im Statusregister setzen

```
I332Write1(I332Handle, icStatus or brModeSetL, stSSF);
```

Interpolation anhalten: **stSSB** im Statusregister löschen

```
I332Write1(I332Handle, icStatus or brModeClrL, (not stSSF));
```

***Hinweis**:* *Die drei DLL-Funktionen `I332ZyklusStart`, `I332ZyklusStop` und `I332DoZyklusStart` beeinflussen sowohl **stSSB** (Start/Stop Interpolation) als auch **stSFB** (Start/Stop Sonderfunktion).*

5.2.2 Linear-Interpolation

***Hinweis**:* *Die Linearinterpolation steht als DLL-Funktion `I332DoLin` zur Verfügung.*

5.2.2.1 Linear-Interpolation in der Ebene

- Bahngeschwindigkeit 1m/min
- Gerade in X und Y

```
{Bahng. 1000mm/min = 1m/min}
I332Write2(I332Handle, icSetVCmd,1000);
```

```
{Achse 0 um 10mm verfahren}
I332Write2(I332Handle, icSetDist or (0 shl 8), 10000);

{Achse 1 um 5mm verfahren}
I332Write2(I332Handle, icSetDist or (1 shl 8), 5000);

{Linear-Interpolation}
I332Write0(I332Handle, icIPLLin or iplModePath);
```

5.2.2.2 Linear-Interpolation im Raum + weitere Achsen

- Bahngeschwindigkeit 1m/min
- Gerade in X,Y und Z, jeweils 10mm
- Zusätzliche Bewegung in Achsen 3 und 4, jeweils 5mm

```
{Bahng. 1000mm/min = 1m/min}
I332Write2(I332Handle, icSetVCmd,1000);

{Achse 0,1,2 um jeweils 10mm verfahren}
I332Write2(I332Handle, icSetDist or (0 shl 8), 10000);
I332Write2(I332Handle, icSetDist or (1 shl 8), 10000);
I332Write2(I332Handle, icSetDist or (2 shl 8), 10000);

{Achse 3+4 um jeweils 5mm verfahren}
I332Write2(I332Handle, icSetDist or (3 shl 8), 5000);
I332Write2(I332Handle, icSetDist or (4 shl 8), 5000);

{Linear-Interpolation}
I332Write0(I332Handle, icIPLLin or iplModePath);
```

Hier ist es wichtig, sich über die jeweiligen Achsgeschwindigkeiten klar zu sein:

- In XYZ erfolgt eine Linearbewegung im Raum mit der programmierten **Bahngeschwindigkeit** von **1000mm/min**
  → da XYZ jeweils den gleichen Weg zurücklegen (10mm), ergeben sich für XYZ Geschwindigkeitskomponenten von jeweils ca. **577mm/min**.
- Die gesamte **Bahnlänge** in XYZ beträgt ca. **17,321mm**, die Achsen 3+4 bewegen sich dazu linear um jeweils 5mm
  → daraus ergibt sich eine Geschwindigkeit für diese beiden Achsen von jeweils ca. **289mm/min**.

5.2.3 Kreis-Interpolation

Für alle Kreis-Interpolationen gilt:

- Die Bahngeschwindigkeit bezieht sich IMMER auf die beiden Kreisachsen, unabhängig davon, was im Bit **apChar**.**acPathB** für die jeweiligen Achsen vorgegeben ist.

- Wird der Mittelpunkt nicht exakt programmiert, so fährt die I332 eine (archimedische) Spirale; die programmierte Geschwindigkeit vergrößert bzw. verkleinert sich dabei im Verlauf der Spirale, je nachdem sie Spirale von innen nach außen oder von außen nach innen verläuft.
  ***Warnung**: Dieser Effekt ergibt sich aufgrund der internen Algorithmen; im Extremfall können die zulässigen Achsbeschleunigungen bzw. die zulässige Achsgeschwindigkeit überschritten werden!*

***Hinweis**: Die Kreisinterpolation steht als DLL-Funktion `I332DoIPLCW` bzw. `I332DoIPLCCW` zur Verfügung.*

5.2.3.1 Einfache Kreis-Interpolation

- Bahngeschwindigkeit 1m/min
- XY-Ebene (Achsen 0 und 1)

- Halbkreis fahren im Uhrzeigersinn
- Durchmesser 10mm

```
{Bahng. 1000mm/min = 1m/min}
I332Write2(I332Handle, icSetVCmd,1000);

{Achse 0 um 10mm verfahren}
I332Write2(I332Handle, icSetDist or (0 shl 8), 10000);

{Achse 1 um 5mm verfahren}
I332Write2(I332Handle, icSetDist or (1 shl 8), 5000);

{Achse 0 Abstand zum Mittelpunkt 5mm}
I332Write2(I332Handle, icSetAbs or (0 shl 8), 5000);

{Achse 1 Abstand z. Mittelp. 0mm}
I332Write2(I332Handle, icSetAbs or (1 shl 8), 0);

{Kreis-Interpolation Achse 0/1, UZS}
I332Write1(I332Handle, icIPLcw or iplModePath,(0 shl 8) + 1);
```

5.2.3.2 Helix-Interpolation

- Bahngeschwindigkeit 1m/min
- XY-Ebene (Achsen 0 und 1)
- Vollkreis fahren im Uhrzeigersinn
- Durchmesser 10mm
- Zusätzliche Bewegung in Z um 5 mm

```
{Bahng. 1000mm/min = 1m/min}
I332Write2(I332Handle, icSetVCmd,1000);

{Achse 0 um 0mm verfahren}
I332Write2(I332Handle, icSetDist or (0 shl 8), 0);

{Achse 1 um 0mm verfahren}
I332Write2(I332Handle, icSetDist or (1 shl 8), 0);

{Achse 2 um 5mm verfahren}
I332Write2(I332Handle, icSetDist or (2 shl 8), 5000);

{Achse 0 Abstand zum Mittelpunkt 5mm}
I332Write2(I332Handle, icSetAbs or (0 shl 8), 5000);

{Achse 1 Abstand z. Mittelp. 0mm}
I332Write2(I332Handle, icSetAbs or (1 shl 8), 0);

{Kreis-Interpolation Achse 0/1, UZS}
I332Write1(I332Handle, icIPLcw or iplModePath,(0 shl 8) + 1);
```

Es entsteht ein Gang einer Schraubenlinie (Helix) mit einer Steigung von 5mm. Nach Bedarf können mehrere solcher Schraubenlinien ohne weiteres hintereinander programmiert werden um mehrgängige Helices zu erzeugen. Soll der 1. bzw. letzte Gang einer solchen Helix keine ganze Umdrehung umfassen, so muß der Wert der Linearachse entsprechend korrigiert werden: soll z.B. der 1. Gang nur ein Halbkreis sein, so muß für Z 0,5mm anstatt 1mm programmiert werden.

Auch hier sollte man sich über die resultierenden Geschwindigkeit klar werden:
Bei der Kreisinterpolation bezieht sich die Bahngeschwindigkeit IMMER auf die beiden Kreisachsen, unabhängig davon, was im Bit **apChar**.**acPathB** für die jeweiligen Achsen vorgegeben ist.
Für dieses Beispiel bedeutet dies:

- Der Kreis mit Radius 5mm wird mit 1000mm/min gefahren → Umfang = 31,416 mm
- Linear dazu erfolgt eine Bewegung in Z um 5mm

- Die Gesamtlänge der Kontur (= Helix) beträgt also ca. 31,811 mm
  → Es ergibt sich eine Geschwindigkeit auf der Kontur (= Helix) von 31,811 mm / 31,416 mm * 1000mm/min = 1013 mm/min
  Dieser Unterschied wird um so größer, je größer die Steigung im Verhältnis zum Radius ist!

5.2.4 Verweilzeit

Eine Verweilzeit wird intern behandelt wie ein Verfahrsatz, d.h. die Daten werden im Verfahrsatzpuffer gespeichert und entsprechend der Satzreihenfolge abgearbeitet.

- Beispiel: Verweilzeit von 10 sec

```
I332Write2(I332Handle, icSetWait, 10000);
I332Write0(I332Handle, icIPLWait);
```

***Hinweis**: Hier steht auch die DLL-Funktion `I332DoDelay` zur Verfügung.*

5.2.5 Spindel- und Ausgabefunktionen innerhalb von Interpolationssätzen

Zusätzlich zu den eigentlichen Verfahrbefehlen können innerhalb eines Satzes Spindel- und Ausgabefunktionen programmiert werden. Voraussetzung ist, daß die relevanten Maschinenparameter korrekt gesetzt wurden (s. Kap. 4.2.1.3 Arbeitsspindel und Kap. 4.2.1.4 Ein-/Ausgänge).

Die Funktionen werden zusammen mit den anderen Satzparametern in den Satzpuffer übernommen und an der richtigen Stelle innerhalb des Aublaufs ausgeführt. Die Programmierung erfolgt daher innerhalb der Satzprogrammierung an beliebiger Stelle, aber VOR der Übergabe des Interpolationsbefehls.

Hinweis: *In den folgenden Beispielen werden der Übersichtlichkeit halber die Interpolationsfunktionen der I332-DLL benutzt. Aus dem gleichen Grund unterbleibt auch die Überprüfung des Funktionsergebnisses auf Fehler. In einer konkreten Anwendung sollte dies natürlich geschehen!*
Für die Programmierung von Spindel- und Ausgabefunktionen stehen (noch) keine DLL-Funktionen zur Verfügung.

Die folgenden Beispiele demonstrieren die Programmierung und lassen sich bei Bedarf auch kombinieren!

5.2.5.1 Spindelfunktionen

Hinweis: *Die interne Behandlung von Spindeldrehzahl und den Schaltfunktionen (Spindel Ein/Aus) erfolgt unabhängig voneinander; wird nur die Spindel ausgeschaltet ohne die Drehzahl explizit auf 0 zu setzen, so bleibt die Spannung entspr. der Spindeldrehzahl am DA-Ausgang stehen! Außerdem erfolgt das Setzen der Drehzahl immer am Satzanfang.*

```
// Spindeldrehzahl 5000 U/min
I332Write2(I332Handle,icSetSCmd,5000);
// Spindel Ein / M04, am Satzanfang
I332Write2(I332Handle,icSetStdM,$00020000);
// Wartezeit 10sec
I332DoDelay(I332Handle,10000,0);

... Fahrfunktionen....

// Spindeldrehzahl 0 am Satzanfang!
I332Write2(I332Handle,icSetSCmd,0);
// Spindel aus / M05, am Satzende
I332Write2(I332Handle,icSetStdM,$00000003);
```

```
// Wartezeit 10sec
I332DoDelay(I332Handle,10000,0);
```

Natürlich kann anstatt einer Verweilzeit auch direkt eine Verfahrfunktion programmiert werden; da die I332 aber das Hochlaufen der Spindel nicht überwachen kann, sollte dies nur dann erfolgen, wenn sichergestellt ist, daß der Verfahrsatz lange genug ist. Ansonsten könnte schon während der Hochlaufphase die Spindel ins Material fahren.

5.2.5.2 Satzsynchrone Ausgabefunktionen

Das Setzen und Löschen von Ausgängen kann sowohl am Satzanfang als auch am Satzende erfolgen. Als Ausgänge stehen alle im System vorhandenen Ausgangsports zur Verfügung, soweit sie nicht durch Achs- oder Spindelfunktionen belegt sind.

***Hinweis**:* *Die Programmierung von Ausgängen (satzsynchron oder direkt vom PC) ist nur dann möglich, wenn sie nicht durch Achsfunktionen belegt sind (Freigabe- bzw. Schrittmotor-Ausgänge)!*

Das folgende Beispiel demonstriert die Vorgehensweise:

```
// Bit 4: SM-Puls, Achse 2, am Satzanfang löschen
I332Write1(I332Handle,icSetMPorts+pgSM+pmClr+psBeg,$FFEF);
// Bit 5: SM-Rtg., Achse 2, am Satzanfang setzen
I332Write1(I332Handle,icSetMPorts+pgSM+pmSet+psBeg,$0020);
// Bit 6: SM-Puls, Achse 3, am Satzende löschen
I332Write1(I332Handle,icSetMPorts+pgSM+pmClr+psEnd,$FFBF);
// Bit 7: SM-Rtg., Achse 3, am Satzende setzen
I332Write1(I332Handle,icSetMPorts+pgSM+pmSet+psEnd,$0080);

// Bit 4: Freigabe Achse 4, am Satzanfang löschen
I332Write1(I332Handle,icSetMPorts+pgEn+pmClr+psBeg,$FFEF);
// Bit 5: Freigabe Achse 5, am Satzanfang setzen
I332Write1(I332Handle,icSetMPorts+pgEn+pmSet+psBeg,$0020);
// Bit 6: Freigabe Achse 6, am Satzende löschen
I332Write1(I332Handle,icSetMPorts+pgEn+pmClr+psEnd,$FFBF);
// Bit 7: Freigabe Achse 7, am Satzende setzen
I332Write1(I332Handle,icSetMPorts+pgEn+pmSet+psEnd,$0080);

// Bit 4: OUT5, am Satzanfang löschen
I332Write1(I332Handle,icSetMPorts+pgIO1+pmClr+psBeg,$FFEF);
// Bit 5: OUT6, am Satzanfang setzen
I332Write1(I332Handle,icSetMPorts+pgIO1+pmSet+psBeg,$0020);
// Bit 6: OUT7, am Satzende löschen
I332Write1(I332Handle,icSetMPorts+pgIO1+pmClr+psEnd,$FFBF);
// Bit 7: OUT8, am Satzende setzen
I332Write1(I332Handle,icSetMPorts+pgIO1+pmSet+psEnd,$0080);

// z.B. Linear-Interpolation:
I332DoIPLLin(I332Handle,....)
```

Wird am Anfang eines Satzes ein Ausgang gesetzt, am Ende gelöscht und am Anfang des nächsten Satzes wieder gesetzt, so entsteht ein kurzer Impuls (ca. 2ms: Interpolationszyklus):

```
// Bit 5: OUT6, am Satzanfang setzen
I332Write1(I332Handle,icSetMPorts+pgIO1+pmSet+psBeg,$0020);
// Bit 5: OUT6, am Satzende löschen
I332Write1(I332Handle,icSetMPorts+pgIO1+pmClr+psEnd,$FFDF);

// z.B. Linear-Interpolation:
I332DoIPLLin(I332Handle,....)

      // nächster Satz:
```

```
// Bit 5: OUT6, am Satzanfang wieder setzen
I332Write1(I332Handle,icSetMPorts+pgIO1+pmSet+psBeg,$0020);

...weitere Angaben...
// z.B. Linear-Interpolation:
I332DoIPLLin(I332Handle,....)
```

Hinweis zur Initialisierung der Maschinendaten:

Innerhalb der Initialisierungsroutine sollten in den Maschinenparametern mpXXOutPAR die entsprechenden Ausgangsbits für die Bedienung durch den Interpolator (Bit = 0) konfiguriert werden:

```
I332Write2(I332Handle,icSetMP+mpSMOutPAR,$xxxxxxxx);
I332Write2(I332Handle,icSetMP+mpIOEnPAR,$xxxxxxxx);
I332Write2(I332Handle,icSetMP+mpIOOutPAR,$xxxxxxxx);
```

Optional können hier auch Defaultwerte nach einem Restart-Befehl gesetzt werden (z.B. alles auf 0):

```
I332Write2(I332Handle,icSetMP+mpSMOutDef,$xxxxxxxx);
I332Write2(I332Handle,icSetMP+mpEnOutDef,$xxxxxxxx);
I332Write2(I332Handle,icSetMP+mpIOOutDef,$xxxxxxxx);
```

Danach können alle Ausgänge deren mpXXOutPAR-Bits = 0 sind satzsynchron bedient werden.

***Hinweis**:* *Da dies für alle Ports der Default-Wert ist, ist diese Initialiserung für das Beispiel nicht unbedingt notwendig.*

5.3 Sonderfunktionen

Sonderfunktionen lassen sich parallel zur satzweisen Interpolation ausführen. Die übergebenen Daten werden nicht in den FIFO-Puffer der Interpolationsroutine geschrieben, sondern es existiert für jede Achse ein eigener Registersatz (Achsregister für Sonderfunktionen, s. Kap. 4.4.1).

Dies bedeutet:
1. Sonderfunktionen müssen für jede Achse getrennt programmiert werden
2. Bevor für eine Achse eine neue Sonderfunktion programmiert werden kann, muß die vorherige komplett bearbeitet sein. Die Verantwortung dafür liegt beim Programmierer!
3. Achsen, die an einer laufenden Interpolation beteiligt sind, können zwar Sonderfunktionen ausführen, es sind jedoch einige Einschränkungen bzw. Bedingungen zu beachten. Die Verantwortung dafür liegt ebenfalls beim Programmierer!
   Weitere Hinweise zu diesem Thema im Kap. 5.4, „Interpolation und Sonderfunktionen“!

Die Programmierung verläuft nach dem gleichen Schema wie bei der Interpolation:

0. Warten bis die entspr. Achse die letzte Sonderfunktion ausgeführt hat
1. Übergabe der Daten
2. Befehl für die gewünschte Sonderfunktion (**icSF...**)

Analog existiert das Statusbit »**stSFB**«, das dieselbe Aufgabe für die Sonderfunktionen hat, wie das Statusbit »**stSSB**« für die Interpolation. Wenn keine Sonderfunktionen mehr aktiv sind, sollte das Bit **stSFB** wieder zurückgesetzt werden:

```
I332Read1(I332Handle,icGetSFInfo+icSFRef,info);
IF info = 0
THEN I332Write0(I332Handle, icStatus or brModeClrB or stSFB);
```

Beenden von Sonderfunktionen:

1. Die Sonderfunktion beendet sich selbst, z.B. Referenzfahrt, Asynchrones Fahren

Funktionen, die nicht automatisch enden (z.B. Handbetrieb, Handrad, Joystick) müssen explizit abgebrochen werden:

2. Eine bestimmte Sonderfunktion wird gezielt abgebrochen:

   ```
   I332Write1(I332Handle,icSFHand+256*AchsNr,sfBreak);
   ```

   Dies bricht die Sonderfunktion „Handbetrieb“ der Achse „AchsNr“ ab; führt die Achse eine andere Sonderfunktion aus, so geschieht nichts.

3. Alle Sonderfunktionen werden abgebrochen:

   ```
   I332Write0(I332Handle,icSFBreak);
   ```

   Dies bricht alle Sonderfunktion in allen Achsen ab, unabhängig davon, welche Sonderfunktion gerade ausgeführt wird.

5.3.1 Referenzfahrt

Als Referenzpunkte dienen bei der Schrittmotorversion die Endschalter. Beim Einsatz der Zusatz-Karte I332-DC bzw. bei der I332-PCI steht zusätzlich je Achse ein echter Refpuls-Eingang (Nullimpuls vom Meß-System) zur Verfügung.

Programmierung:

Es genügt, den W1-Befehl »**icSFRef**« zu übergeben. Notwendige Parameter werden im Datenwort des Befehls definiert.

Beispiel: Achse 3 in negativer Richtung auf den Referenzpunkt fahren.

```
{Testen, ob SF-Puffer für Achse 3 frei:}
I332Read1(I332Handle, icGetSFInfo+icSFRef, info);
IF (info AND $0808) <> 0
THEN {Fehlermeldung}
ELSE I332Write1(I332Handle, icSFRef + 3*256, sfNeg+rm);
```

***Hinweis**:* *Die Referenzfahrt steht als DLL-Funktion `I332DoAxisRef` zur Verfügung.*

Die Referenzfahrt ist beendet, wenn der Befehl `icGetSFInfo` im entsprechenden Achs-Bit = 0 liefert:

```
{ Warten auf das Ende der Refernzfahrt: }
REPEAT
  I332Read1(I332Handle,icGetSFInfo+icSFRef,info);
UNTIL (info AND $0008) = 0;
```

***Anmerkung**:* *Es muß zumindest ein Endschalter/Referenzschalter am entsprechenden Eingang angeschlossen sein und der Achsparameter **apLS** korrekt gesetzt sein.*

Allgemeiner Ablauf der Referenzfahrt:

1. Die Achse fährt in der angegebenen Rtg (**sfNeg**) mit Geschwindigkeit **apVRef** los.
2. Wird der Endschalter für diese Rtg erkannt, so wird die Achse gestoppt.
3. Wurde der ES überfahren, so wird zunächst auf den ES zurück positioniert.
4. Anschließend wird der ES freigefahren, entspr. der in **apLS** definierten Richtung (**lsPHRtgB** bzw. **lsMHRtgB**)
5. Danach wird in die gleiche Richtung um den in **apLSHys** definierten Weg vom ES weg positioniert (Schalt-Hysterese des Endschalters).
   Kann der ES nach einer bestimmten Strecke nicht freigefahren werden (**apHysMax**), so wird mit der Fehlermeldung »**ieRefHys**« abgebrochen.
6. Ist der Parameter **apRefOffs** <> 0, so wird dieser Offset zusätzlich mit der Geschwindigkeit **apVRef** verfahren.
7. Zuletzt wird die Referenzposition aus dem Achsparameter **apRefPos** übernommen

Auswertung des Refpuls-Eingangs (mit I332-DC oder I332-PCI), setzen von Verfahrgrenzen:

Dazu müssen im Datenwort die Konstanten **rmRef** und **rmSWLS** addiert werden, z.B. für Achse 3:

```
I332Write1(I332Handle, icSFRef + 3*256, sfNeg+rm+rmRef+rmSWLS);
```

Außerdem ist zu beachten, daß im Maschinendatum **apDC** für diese Achse **dcRefInB** gesetzt ist. Die Verfahrfahrgrenzen werden aus **apLSDeltaP** und **apLSDeltaM** übernommen.

Ablauf:
1-5 wie oben

6. entspr. der in **apLS** definierten Richtung (**lsPRRtgB** bzw. **lsMRtgB**) wird der Referenz-Puls mit der Geschwindigkeit **apVRef** gesucht. Sobald der entsprechende Eingang aktiv ist, wird die Achse gestoppt.

7. Ist der Parameter **apRefOffs** <> 0, so wird dieser Offset zusätzlich mit der Geschwindigkeit **apVRef** verfahren.
8. Zuletzt werden die Referenzposition und die Verfahrgrenzen aus den Achsparametern übernommen

***Anmerkungen**:*

- *Die Referenzfahrt startet erst, wenn das Statusbit **stSFB** gesetzt ist.*
- *Wird das **stSFB** gelöscht, so wird die Referenzfahrt ohne Fehlermeldung (**stErrB** im Statusregister) komplett abgebrochen!*
- *Wird während der Phasen 3-6 die in **apHysMax** definierte Distanz überschritten, so wird die Referenzfahrt mit gesetztem **stErrB** im Statusreg. und der Fehlernr. »**ieRefHys**« abgebrochen.*
- *Refpuls-Suchen: Wird als Referenz-Impuls der Null-Impuls eines Resolvers verwendet, so kann es passieren, daß der Puls innerhalb der mechanischen Toleranz des zuvor gesuchten Endschalters liegt.*
  Dabei kommt es vor, daß der Resolverpuls manchmal sofort, manchmal aber auch erst nach einer vollen Motorumdrehung erkannt wird.
  Zur Lösung dieses Problems bieten sich mehrere Möglichkeiten:
  - *Manche Verstärker bieten die Möglichkeit, die Lage des Resolver-Null-Impulses selbst zu bestimmen.*
  - *mechanisches Versetzen des Endschalters.*
  - *Einsatz des Parameters **apHys**: Diese Hysterese wird NACH dem Freifahren des Endschalters aber VOR dem Beginn der Ref-Pulssuche ausgeführt. Deshalb ist es möglich, mit diesem Parameter den Beginn der Ref-Pulssuche an einen Punkt zu legen, der garantiert zwischen zwei Ref-Pulsen liegt.*

5.3.2 Asynchrones Fahren – Positionierbetrieb

Das asynchrone Verfahren einer Achse (parallel zu einer laufenden Interpolation) ähnelt einer Linear-Interpolation in einer Achse:

Im Register **icSetSFDist** wird die gewünschte Strecke übergeben, evtl. in **icSetSFVCmd** die gewünschte Geschwindigkeit.
Abgeschlossen wird die Datenübergabe mit dem Befehl »**icSFAsync**«.

Beispiel1: Achse 3 um 100mm mit V = 200mm/sec asynchron verfahren

```
{Abfrage, ob Puffer frei:}
I332Read1(I332Handle, icGetSFInfo+icSFAsync, info);
IF (info AND $0808) <> 0
THEN {Fehlermeldung}
ELSE BEGIN

I332Write2(I332Handle,icSetSFDist+3*256,100000);
I332Write2(I332Handle,icSetSFVCmd+3*256,200);
I332Write1(I332Handle,icSFAsync+256*3,sfVCmd);

END;
```

Beispiel2: Achse 2 um 200mm im Eilgang (VMax) asynchron verfahren

```
{Abfrage, ob Puffer frei:}
I332Read1(I332Handle, icGetSFInfo+icSFAsync, info);
IF (info AND $0404) <> 0
THEN {Fehlermeldung}
ELSE BEGIN

  I332Write2(I332Handle,icSetSFDist+256*2,200000);
  I332Write1(I332Handle,icSFAsync+256*2,sfVMax);
  {Eine evtl. gesetzte Geschwindigkeit (icSetSFVCmd)
  wird nicht beachtet}
```

```
END;
```

Beispiel3 (ab V.x.4.67): Achse 2 um 200mm im Eilgang (VMax) asynchron verfahren, Poti an ADC-Eingang 1:

```
{Abfrage, ob Puffer frei:}
I332Read1(I332Handle, icGetSFInfo+icSFAsync, info);
IF (info AND $0404) <> 0
THEN {Fehlermeldung}
ELSE BEGIN

  sfMode = sfVMax+(1*256);        { ADCNr = 1 ins Highbyte }
  I332Write2(I332Handle,icSetSFDist+256*2,200000);
  I332Write1(I332Handle,icSFAsync+256*2,sfMode);
  {Eine evtl. gesetzte Geschwindigkeit (icSetSFVCmd)
  wird nicht beachtet}

END;
```

***Hinweis**:* *Das asynchrone Verfahren einer Achse steht als DLL-Funktion `I332DoAxisAsync` zur Verfügung.*

5.3.3 Hand-Betrieb

Unter **Hand-Betrieb** wird das Verfahren einer Achse (auch parallel zu einer laufenden Interpolation) ohne Vorgabe einer Distanz verstanden.

Dazu muß lediglich im Register **icSetSFVCmd** die gewünschte Geschwindigkeit übergeben werden. Abgeschlossen wird die Datenübergabe mit dem Befehl »**icSFHand**«.

Beispiel1: Achse 3 mit V = 200mm/sec in positiver Richtung verfahren, Stop bei Tastendruck

```
{Abfrage, ob Puffer frei:}
I332Read1(I332Handle, icGetSFInfo+icSFHand, info);
IF (info AND $0808) <> 0
THEN {Fehlermeldung}
ELSE BEGIN

  I332Write2(I332Handle,icSetSFVCmd+256*3,200);
  I332Write1(I332Handle,icSFHand+256*3,sfVCmd);

END;

WHILE NOT {Taste gedrückt} DO;  { Warten auf Tastendruck}

{ Handbetrieb Achse 3 abbrechen: }
I332Write1(I332Handle,icSFHand+256*3,sfBreak)
```

Die Übergabe der Geschwindigkeit kann entfallen, falls sie implizit mit dem Kommando **icSFHand** angegeben wird:

Beispiel2: Achse 3 im Eilgang in neg. Richtung verfahren:

```
I332Write1(I332Handle,isSFHand+256*3,sfVMax+sfNeg)
```

***Anmerkung**:* *Für diese Funktion gilt eine Ausnahme der Regel, daß eine Achse erst dann wieder programmiert werden darf, wenn die aktuelle Sonderfunktion beendet ist:*
Ist diese Funktion aktiv, so darf die Sonderfunktion erneut programmiert werden um die Geschwindigkeit zu ändern. Die neue Geschwindigkeit wird korrekt über die Rampe angefahren, auch bei Vorzeichenwechsel!

***Hinweis**:* *Das Hand-Verfahren einer Achse steht als DLL-Funktion `I332DoAxisHand` zur Verfügung.*

5.3.4 Handrad-Betrieb

Folgendes PASCAL-Fragment initialisiert den Handradbetrieb für die Achse "`AchsNr`", einem Handrad am DG-Eingang "`DGNr`" und der maximalen Geschwindigkeit "`VHandrad`"; das Übersetzungs-Verhältnis beträgt "`my`" µm / "`dgPulse`" Impulse:

```
{Abfrage, ob Puffer frei:}
I332Read1(I332Handle, icGetSFInfo+icSFHandRad, info);
IF (info AND ($0101 shl AchsNr)) <> 0
THEN {Fehlermeldung}
ELSE BEGIN
  { Geschwindigkeit, mm/min }
  I332Write2(I332Handle,icSetSFVCmd+AchsNr*256,VHandrad);

  { Übersetzungsverhältnis: }
  I332Write2(I332Handle,icSetDG_my+AchsNr*256,my);
  I332Write2(I332Handle,icSetDG_pls+AchsNr*256,dgPulse);

  { programmierte Geschwindigkeit (VHandrad) verwenden, }
  { Poti nicht berücksichtigen: }
  sfMode := (DGNr * 256)+sfVCmd+sfM49;

  { Sonderfunktion Handrad: }
  I332Write1(I332Handle,icSFHandrad+(AchsNr * 256),sfMode);

END;
```

Hinweis *Als Drehgeber-Eingänge stehen alle vorhandenen zur Verfügung soweit sie nicht durch Achs-Funktionen belegt sind (s.Kap. 4.8).*

5.3.5 Joystick-Betrieb

Folgendes PASCAL-Fragment initialisiert den Joystickbetrieb für die Achse "`AchsNr`", einem Joystick am ADC-Eingang "`ADCNr`" und der maximalen Geschwindigkeit "`VJoystick`" am Joystick-Anschlag:

```
{Abfrage, ob Puffer frei:}
I332Read1(I332Handle, icGetSFInfo+icSFJoystck, info);
IF (info AND ($0101 shl AchsNr)) <> 0
THEN { Fehlermeldung }
ELSE BEGIN
  { Geschwindigkeit, mm/min }
  I332Write2(I332Handle,icSetSFVCmd+AchsNr*256,VJoystick);

  { programmierte Geschwindigkeit (VJoystick) verwenden, }
  { Poti und ES unterdrücken }
  sfMode := (ADCNr * 256)+sfVCmd+sfM49;

  { Sonderfunktion Joystick }
  I332Write1(I332Handle,icSFJoystick+(AchsNr * 256),sfMode);

END;
```

Folgende PASCAL-Procedur übergibt eine neue Kennlinie für die Achse "`AchsNr`":

```
CONST KennLinie : ARRAY[0..16] OF ShortInt  {-128..+127)
                { 17 Kennlinien-Werte: }
                = (-128,-128,-112, -96, -64, -32, -16,   0,
{ Mittelstellung:}    0,
```

```
                        0, +16, +32, +64, +96,+112,+127,+127);

PROCEDURE SetJSKL(AchsNr: Byte);
VAR i : INTEGER;
BEGIN
  FOR i := 0 TO 16 DO
  I332Write1(I332Handle,
             icJSKennLinie+(AchsNr * 256),
             (i * 256) + Byte(KennLinie[i]));
END;
```

Hinweis *Für jede Achse ist DIESE Kennlinie bereits als Default vorgegeben; für einen ersten Test bzw. wenn dies den Erfordernissen entspricht, ist es also nicht notwendig, die Kennlinie explizit zu programmieren, s.a. Kap. 4.4.2.5.*
Als ADC-Eingänge stehen alle vorhandenen außer dem Poti-Eingang (ADC0) zur Verfügung (s.Kap. 4.8).

5.4 Interpolation und Sonderfunktionen

Die interne Verwaltung der Positionsdaten von Interpolationssätzen und Sonderfunktionen in getrennten Registersätzen erlaubt es, Interpolation und Sonderfunktionen unabhängig voneinander zu programmieren und zu fahren. Solange bestimmte Achsen nur interpolierend, andere dagegen nur mit Sonderfunktionen verfahren werden, ist nichts besonderes zu berücksichtigen.

Anders sieht es aus, wenn Achsen sowohl interpolierend als auch mit Sonderfunktionen verfahren werden sollen. Hier kann es vorkommen, daß die von den Registersätzen verwalteten Positionen nicht synchron sind; ein Restart-Befehl (**icRestart1..5**, **icRestart11..15**) synchronisiert die beiden Registersätze wieder neu.

Dieses Verhalten erlaubt es, sowohl Achsen während einer laufenden Interpolation zuzustellen, als auch zusätzliche Hilfsfunktionen weiterer Achsen interpolationsunabhängig auszuführen.

Beispiele:

- Zustellen einer Achse

  - Starten der Bewegung: **stSSB** im Statusregister setzen
  - Programmieren von Interpolationssätzen zum Abarbeiten einer Kontur
  - Zum Zustellen während der Bewegung: **stSSB** im Statusregister löschen
  - Starten der Zustellung: **stSFB** im Statusregister setzen
  - Sonderfunktion **icSFHand** für die zuzustellende(n) Achse(n) programmieren
    ***Hinweis**: Andere Sonderfunktionen sollten hier nicht eingesetzt werden!*
  - Nach Ende der Zustellbewegungen: **stSFB** im Statusregister löschen
  - Weiterbearbeiten der Kontur: **stSSB** im Statusregister wieder setzen
    Der Positionsversatz durch die Zustellbewegung(en) bleibt bis zum nächsten Restartbefehl erhalten.

- Handeingriff des Benutzers zwischen zwei Konturzügen (z.B. zum Werkzeugwechseln)

  - Zum Starten der Bewegung: **stSSB** im Statusregister setzen
  - Programmieren von Interpolationssätzen zum Abarbeiten des 1. Konturzuges
  - Konturzug komplett abarbeiten / verfahren, d.h. warten bis **stBNEB** im Status gelöscht ist.
  - Starten des Benutzereingriffs: **stSFB** im Statusregister setzen
  - Sonderfunktionen entsprechend der Benutzerinteraktion programmieren (z.B. als Reaktion auf die Bedienereingaben / Bedienoberfläche)
  - Nach Ende des Benutzereingriffs: **stSFB** im Statusregister löschen
  - *Restartbefehl (**icRestart1..5**, **icRestart11..15**) zum Synchronisieren der internen Positionsregister*
  - Zum Weiterbearbeiten der Kontur: **stSSB** im Statusregister wieder setzen
  - Ggf. Startpunkt des nächsten Konturzuges anfahren
  - Nächsten Konturzug programmieren.

5.5 Programmierung der Ausgänge

Wie in Kap. 4.8.2 beschrieben, kann die Bedienung der Ausgänge sauber auf I332 und PC aufgeteilt werden. Default-Einstellung ist die Bedienung durch die I332 bzw. deren Interpolationseinheit (s. Programmierbeispiel zur satzsynchronen Ausgabe, Kap. 5.2.5.2).

Soll der PC exklusiven Zugriff auf bestimmte Ausgänge haben, so sind die entsprechenden Bits in den Maschinenparametern **apXXOutPAR** innerhalb der Initialisierungs-Routine auf 1 zu setzen:

```
I332Write2(I332Handle,icSetMP+mpSMOutPAR,$xxxxxxxx);
I332Write2(I332Handle,icSetMP+mpEnOutPAR,$xxxxxxxx);
I332Write2(I332Handle,icSetMP+mpIOOutPAR,$xxxxxxxx);
```

Optional können hier auch Defaultwerte nach einem Restart-Befehl gesetzt werden (z.B. alles auf 0):

```
I332Write2(I332Handle,icSetMP+mpSMOutDef,$xxxxxxxx);
I332Write2(I332Handle,icSetMP+mpEnOutDef,$xxxxxxxx);
I332Write2(I332Handle,icSetMP+mpIOOutDef,$xxxxxxxx);
```

Danach können alle Ausgänge deren mpXXOutPAR-Bits = 1 sind ausschließlich vom PC bedient werden. Ausgänge deren mpXXOutPAR-Bits = 0 können weiterhin satzsynchron bedient werden!

***Hinweis**:* *Die Programmierung von Ausgängen (satzsynchron oder direkt vom PC) ist nur dann möglich, wenn sie nicht durch Achsfunktionen belegt sind (z.B. Freigabe- oder Schrittmotor-Ausgänge)!*

Beispiele:

Ausgänge OUT1..8 setzen, OUT9..16 löschen:

```
I332Write1(I332Handle,icSetOutBuf+pgIO1+pmPut,$00FF);
```

Einzelnen Ausgang OUT4:

```
I332Write1(I332Handle,icSetOutBuf+pgIO1+pmSet,$0008);  // setzen
I332Write1(I332Handle,icSetOutBuf+pgIO1+pmClr,$FFF7);  // löschen
I332Write1(I332Handle,icSetOutBuf+pgIO1+pmChg,$0008);  // invertieren
```

6 Technische Details

6.1 Numerische Einschränkungen

Verfahrbereich in jeder Achse:	± 2.147.483,647 mm
bzw.	± 134.217.727 Incremente
max. Bahnlänge für einen Interpolationssatz: (nicht zu verwechseln mit dem Verfahrbereich!)	ca. 71.000,000 mm
max. Verfahrwege für jede Achse bei	
3 Bahnachsen:	± 41.000,000 mm
2 Bahnachsen:	± 50.000,000 mm
1 Bahnachse:	± 71.000,000 mm
keine Bahnachse:	± 71.000,000 mm
max. Radius (Vollkreis):	± 11.000,000 mm
max. Radius:	71.000,000 mm

Anmerkung**: Bahnachsen sind Achsen mit **acPathB** = 1 im Achsparameter **apChar

max. Geschwindigkeit:	4.095.750 Incremente/sec
bei 1000 Inc/mm:	4.095 mm/sec = 245 m/min
Sonderfunktion "Asynchron Fahren"	
max. Bahnlänge:	± 2.147.483,647 mm
bzw.	± 134.217.727 Incremente

Alle Werte stellen *numerische* Grenzen dar. Ggf. werden sie durch die Achsparameter **apmm** und **apStep** entsprechend eingeschränkt!

6.2 I332-ISA

Die PC-Karte I332-ISA existiert aktuell in der Platinen-Revision I332-3; die Unterschiede sind unten beschrieben. Die Revisionen I332-1 und I332-2 wird nicht mehr hergestellt und vertrieben. Falls solche Karten ausgetauscht werden, so sind einige **Änderungen in der Steckerbelegung** zu beachten!

***Anmerkung** zu den Platinenversionen:*
Ab Version I332-2 bzw. I332-DC2 ist die Platinenversion auf der Rückseite (Lötseite) der Platinen eingedruckt.

***Anmerkung** zu den Steckerbelegungen:*
Die Nummerierung von SubD-Steckern erfolgt üblicherweise entlang der Stiftreihen, die Nummerierung von 2-reihigen Steckleisten dagegen abwechselnd links (= ungerade) und rechts (= gerade).

6.2.1 Steckerbelegung I332

Belegung des 37-poligen SubD-Steckers und der internen 40-poligen Steckleiste. Die interne 40-polige Steckleiste ist so beschaltet, daß ein 1:1-Flachbandkabel zu einer zweiten 37pol SUB-D-Buchse mit analoger Belegung für die Achsen 4 bis 7 angeschlossen werden kann.

37-polige SUB-D-Buchse

+5 V	(1)	
		(20) Ext+
ES+X	(2)	
		(21) ES-X
ES+Y	(3)	
		(22) ES-Y
ES+Z	(4)	
		(23) ES-Z
ES+U	(5)	
		(24) ES-U
PulsX	(6)	
		(25) RtgX
PulsY	(7)	
		(26) RtgY
PulsZ	(8)	
		(27) RtgZ
PulsU	(9)	
		(28) RtgU
+12 V	(10)	
		(29) +12 V
ADC0	(11)	
		(30) Ref1+
ADC2	(12)	
		(31) Ref1-
ADC4	(13)	
		(32) Notaus
ADC6	(14)	
		(33) I1_B*
I1_B	(15)	
		(34) I1_A*
I1_A	(16)	
		(35) RxD
TxD	(17)	
		(36) +U_EXT
GND_EXT	(18)	
		(37) Ext-
GND	(19)	

40-polige Steckleiste

+5 V	(1)	(2) Ext+
ES+V	(3)	(4) ES-V
ES+A	(5)	(6) ES-A
ES+B	(7)	(8) ES-B
ES+C	(9)	(10) ES-C
PulsV	(11)	(12) RtgV
PulsA	(13)	(14) RtgA
PulsB	(15)	(16) RtgB
PulsC	(17)	(18) RtgC
+12 V	(19)	(20) +12 V
ADC1	(21)	(22) Ref2+
ADC3	(23)	(24) Ref2-
ADC5	(25)	(26) Notaus
ADC7	(27)	(28) IRQ6
	(29)	(30)
	(31)	(32) RxD
TxD	(33)	(34) +U_EXT
GND_EXT	(35)	(36) Ext-
GND	(37)	(38)
	(39)	(40)

Die Belegung für die Achsen X, Y, Z und U entspricht dem 37pol-SubD-Stecker

Technische Details

Anschlüsse zur Spannungsversorgung:

+5V, GND	Versorgungsspannung der Platine Busversion: +5V, GND vom PC-Slot ser. Version: externe Spannungsversorgung
+12V	diese Spannung wird intern nicht benötigt Busversion: +12V vom PC-Slot, Belastung entspr. der Spezifikationen des verwendeten PC ser. Version: n.c.
+U_EXT	ext. Spannungsversorgung der Optokoppler für die Schrittmotor-Ausgänge, 12..24V
GND_EXT	gemeinsame Masse für Endschalter-Eingänge und Schrittmotor-Ausgänge
Ext+	Externe Spannung für die serielle Schnittstelle +12V, nur beschalten bei offenem Jumper J4
Ext-	Externe Spannung für die serielle Schnittstelle -12V, nur beschalten bei offenem Jumper J5

Signale:

ES+X..ES-C	Endschalter-Eingänge, optoentkoppelt, 12..24V gegen GND_EXT (gemeinsame Masse für alle Endschalter), Stromaufnahme ca. 10mA pro Eingang
PulsX..PulsC, RtgX..RtgC	Schrittmotor-Ausgänge, gemeinsame Masse (GND_EXT) Pulse und Richtungs-Signal, optoentkoppelt, max. Belastung ca. 200mA pro Ausgang
ADC0..ADC7 ADC0	Analog-Eingänge, 0..5V, max. 20mA Eingangsstrom, max. 10kOhm Poti-Eingang
Ref1+,Ref1-, Ref2+,Ref2-	allg. Eingänge (optional), optoentkoppelt, potential-frei Achtung: Bei Beschaltung ist ein Vorwiderstand vorzusehen, der den Strom auf max. 5 mA begrenzt!
I1_A,I1_A*, I1_B,I1_B*	Drehgeber-Eingänge (optional), optoentkoppelt, Heidenhein-kompatibel
Notaus	**I332-2**: Ein Pegel von 5V gegen GND_EXT aktiviert eine Notstop-Routine der I332 (s. Kap. 4.1.1) **I332-3**: Der Vorwiderstand wurde auf 3,3 kOhm erhöht. Ein Pegel von 12-24V gegen GND_EXT ist nun erforderlich, Stromaufnahme ca. 10mA
IRQ6	nur **I332-3**: Interrupt-Eingang, optoentkoppelt; eine Spannung von 12-24V gegen GND_EXT löst einen Interrupt der Priorität 6 aus (falls freigegeben), Stromaufnahme ca. 10mA Bei entsprechender Programmierung kann hier z.B. ein Messtaster-Signal angeschlossen werden.

Technische Details

TxD,RxD — serieller Ausgang bzw. Eingang, die Signale sind nach RS-422 galvanisch getrennt.

Verbindung mit dem PC:

9pol PC	25pol PC	Bezeichnung	37pol I332
3	2	TxD → RxD	3
2	3	RxD ← TxD	17
5	7	Gnd	19,37

ACHTUNG - Stand-Alone-Betrieb über die serielle Schnittstelle:
Über die entsprechenden Anschlüsse am PC-Bus muß die Karte mit +5V versorgt werden.

Stecker-Kompatibilität I332-1 / I332-2

Die beiden 40pol-Steckerleisten sind **nicht** kompatibel; für die 37pol SUB-D-Buchse gilt folgende Ausnahme-Tabelle:

PinNr	I332-1	I332-2	Anmerkung
20	+5V	Ext+	Bei geschlossenem Jmper J4 liegt hier die +12V-Spannung vom PC
37	GND	Ext-	Bei geschlossenem Jmper J5 liegt hier die -12V-Spannung vom PC
32	ADC5	Notaus	

Stecker-Kompatibilität I332-2 / I332-3

Die Stecker sind kompatibel.

6.2.2 Jumper

Die Platinen-Darstellung ist nicht maßstabgerecht.

Die Jumper J1, J2 und J3 sind nur zu Testzwecken vorhanden und sollten nicht verändert werden!

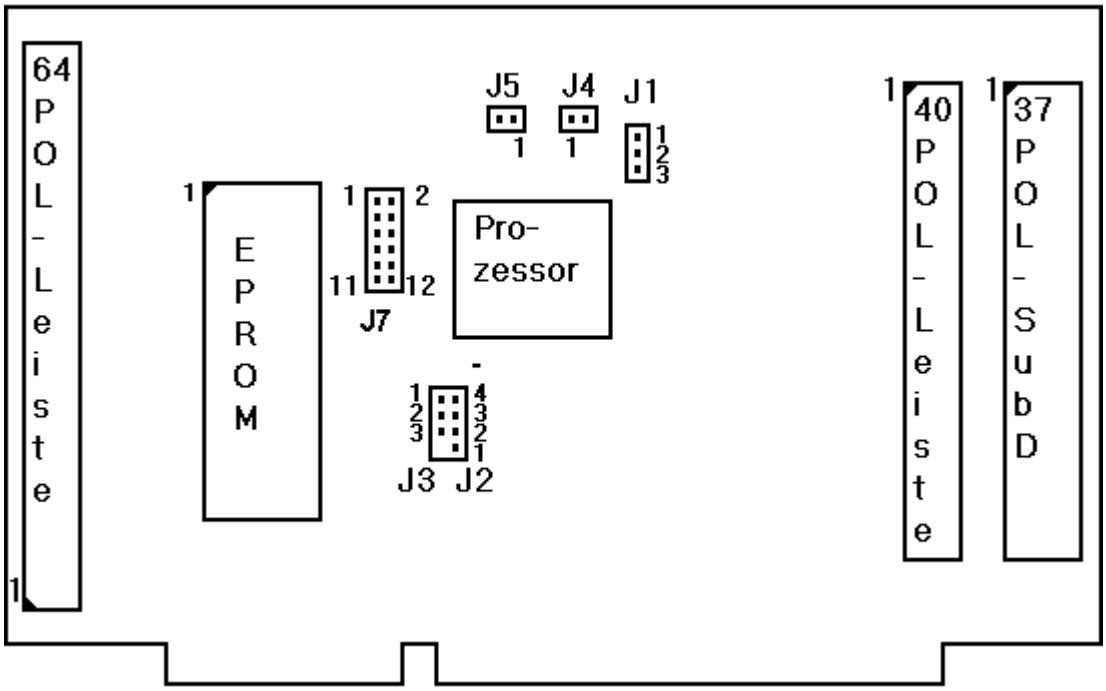


Abb. 6.19 Jumper auf der I332-2 (Bestückungsseite)

Jumper J1 - J3:

Jumper J1: 2-3
Jumper J2: 1-2
Jumper J3: 2-3

Stiftleiste J7:
Hier liegen u.a. noch einmal die Signale der seriellen Schnittstelle an:

Stift 3: GND
Stift 11: TxD
Stift 12: RxD

Alle anderen Stifte dürfen nicht beschaltet werden (Test-Ausgänge des Prozessors!)

Jumper J4,J5

offen: Die RS-422-Schnittstelle wird über die Anschlüsse Ext+ und Ext- mit +12V bzw. -12V versorgt.
Achtung: Die Masse-Leitung der seriellen Schnittstelle darf dann natürlich NICHT mit den Anschluß GND verbunden werden, sondern muß an die Masse der externen Spannungsversorgung gelegt werden!

geschlossen: Die RS-422-Schnittstelle wird vom PC mit +12V bzw. -12V versorgt; an den Anschlüsse Ext+ und Ext- liegt +12V bzw. -12V vom PC!

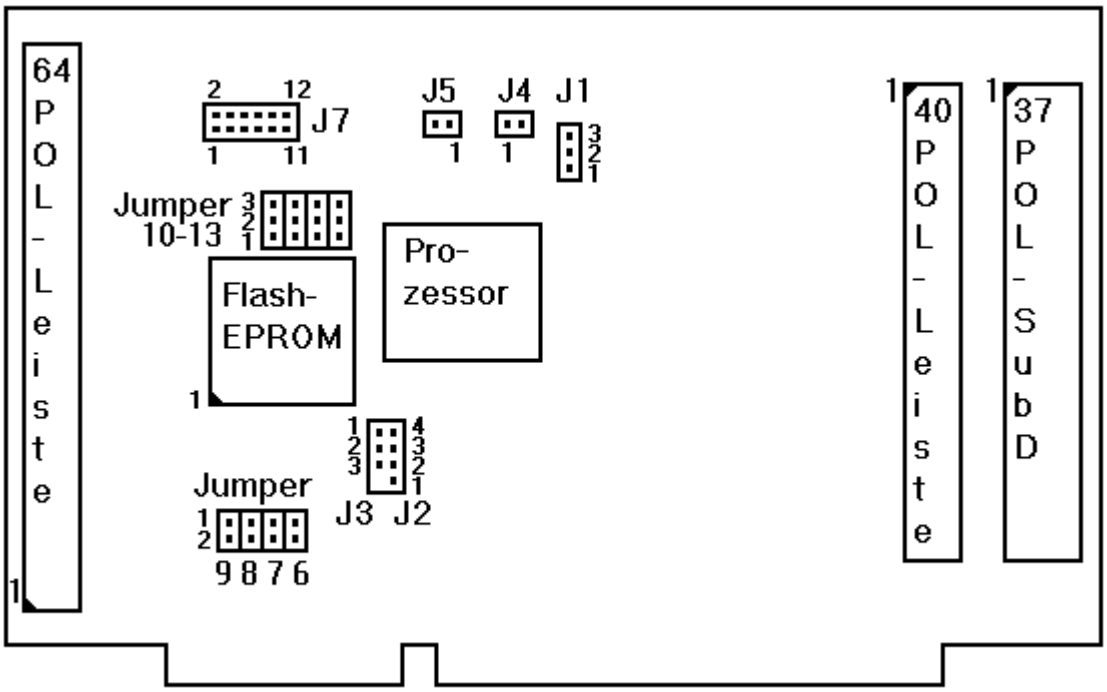


Abb. 6.20 Jumper auf der I332-3 (Bestückungsseite)

Die Jumper J1 bis J5 sind identisch zur I332-1/2. Dies gilt auch für die **Stiftleiste J7**; zu beachten ist jedoch, daß sie trotz ähnlicher Lage wie auf der I332-1 **umgekehrt orientiert** ist (Pin1 links unten anstatt rechts oben)!

Jumper 10 bis 13 dienen zur Auswahl des EPROM-Typs. Standardmäßig wird ein 128-kByte-Flash-EPROM bestückt, was folgende Konfiguration erfordert:

Jumper 10:	ganz offen (kein Jumper gesteckt!)
Jumper 11:	1-2
Jumper 12:	1-2
Jumper 13:	1-2

Jumper 6 bis 9 dienen der Auswahl eines PC-Interrupts; dies wird jedoch in der aktuellen Software-Version (V1.7) noch nicht unterstützt. Die Jumper MÜSSEN deshalb OFFEN bleiben. Im geschlossenen Zustand wählen sie folgende PC-Interrupts (es darf nur 1 Jumper geschlossen sein!):

Jumper 6:	INT5
Jumper 7:	INT7
Jumper 8:	INT10
Jumper 9:	INT12

6.2.3 Beschaltung der Ein-/Ausgänge

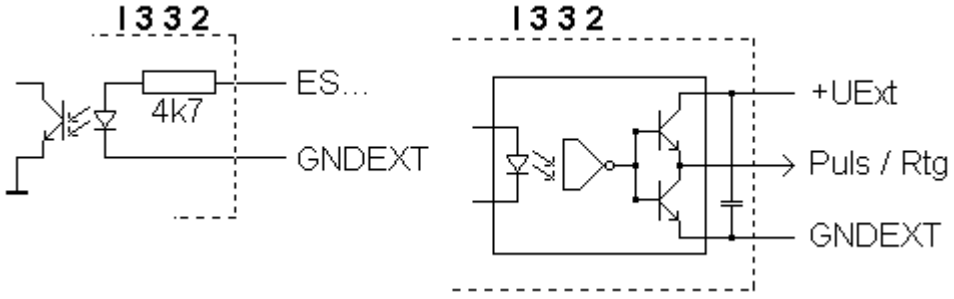


Abb. 6.21 a) Beschaltung: Endschalter- Eingänge b) Beschaltung: Puls/Rtg-Ausgänge

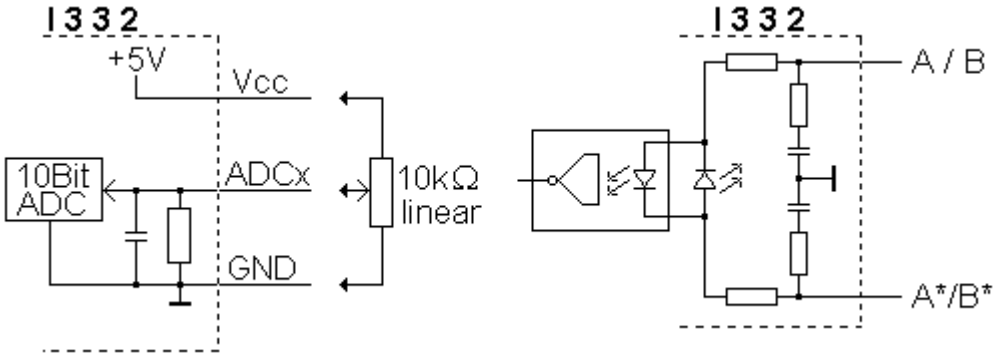


Abb. 6.22 a) Beschaltung der ADC-Eingänge b) Beschaltung der Drehgeber-Eingänge

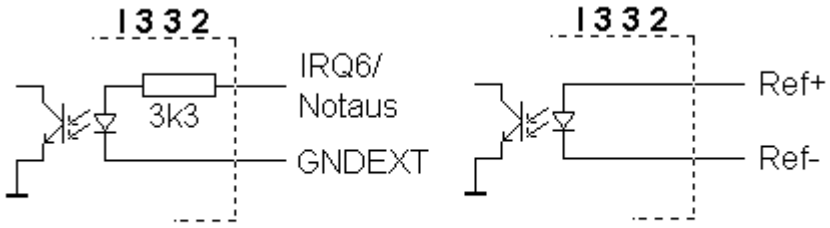


Abb. 6.23 Beschaltung der Eingänge IRQ6, Notaus, Ref1+/- und Ref2+/-

6.3 I332-DC

Die PC-Karte I332 existiert aktuell in der Platinen-Revision I332-DC2. Die Revision I332-DC1 wird nicht mehr hergestellt und vertrieben. Falls solche Karten ausgetauscht werden, so sind einige **Änderungen** zu beachten!

6.3.1 Steckerbelegung

Belegung des 37-poligen SubD-Steckers und der internen 40-poligen Steckleiste. Die interne 40-polige Steckleiste ist so beschaltet, daß ein 1:1-Flachbandkabel zu einer zweiten 37pol SUB-D-Buchse mit analoger Belegung für die Achsen 4 bis 7 angeschlossen werden kann.

37-polige-SubD-Buchse

AX	(1)	
		(20) AX*
BX	(2)	
		(21) BX*
AY	(3)	
		(22) AY*
BY	(4)	
		(23) BY*
AZ	(5)	
		(24) AZ*
BZ	(6)	
		(25) BZ*
AU	(7)	
		(26) AU*
BU	(8)	
		(27) BU*
DC_X	(9)	
		(28) DC_Y
DC_Z	(10)	
		(29) DC_U
GND_DAC	(11)	
		(30) VCC_EXT
FREIX	(12)	
		(31) FREIY
FREIZ	(13)	
		(32) FREIU
REFX	(14)	
		(33) REFY
REFZ	(15)	
		(34) REFU
BEREITX	(16)	
		(35) BEREITY
BEREITZ	(17)	
		(36) BEREITU
GND_EXT	(18)	
		(37) +5V
GND	(19)	

40-polige Steckerleiste (intern)

AV	(1)	(2) AV*
BV	(3)	(4) BV*
AA	(5)	(6) AA*
BA	(7)	(8) BA*
AB	(9)	(10) AB*
BB	(11)	(12) BB*
AC	(13)	(14) AC*
BC	(15)	(16) BC*
DC_V	(17)	(18) DC_A
DC_B	(19)	(20) DC_C
GND_DAC	(21)	(22) VCC_EXT
FREIV	(23)	(24) FREIA
FREIB	(25)	(26) FREIC
REFV	(27)	(28) REFA
REFB	(29)	(30) REFC
BEREITV	(31)	(32) BEREITA
BEREITB	(33)	(34) BEREITC
GND_EXT	(35)	(36) +5V
GND	(37)	(38)
	(39)	(40)

Die Belegung entspricht dem 37pol-SubD-Stecker für die Achsen X, Y, Z und U.

Anschlüsse zur Spannungsversorgung:

+5V, GND	Versorgungsspannung der Platine Busversion: +5V, GND vom PC-Slot serielle Version: externe Spannungsversorgung
GND_DAC	Masse vom DA-Wandler (DC-Ausgänge) bei der seriellen Version muß der DA-Wandler über die entsprechenden ±12V-PC-Slot-Anschlüsse mit ±11,4-16,5V versorgt werden. **I332-DC1**: Dieser Anschluß ist intern mit GND verknüpft **I332-DC2**: Dieser Anschluß ist intern **nicht** mit GND verknüpft
GND_EXT	ext. Masse der Optokoppler für Referenz- und Bereitschafts-Eingänge. Es besteht keine Verbindung zum entsprechenden Anschluß auf der Interpolatorkarte I332!

Signale:

AX/AX*/BX/BX* ... AC/AC*/BC/BC*	Drehgeber-Signale, optoentkoppelt, Heidenhein-kompatibel (antivalent, 90grd phasenverschoben) gemeinsame Masse für alle 16 Signalpaare **I332-DC1**: GND **I332-DC2**: GND_EXT
DC_X..DC_C	Analog-Ausgänge ±10V für DC-Endstufen. externe Spannungs-Versorgung für die DA-Wandler: **I332-DC1**: ±12V-Spannungs-Versorgung über den PC-Bus **I332-DC2**: nicht notwendig
REFX..REFC	Referenzpuls-Eingänge, optoentkoppelt, max. 5V gegen GND_EXT, Stromaufnahme ca. 10mA pro Eingang
BEREITX.. ..BEREITC	Bereitschafts-Eingänge von der Endstufe optoentkoppelt, max. 5V gegen GND_EXT Stromaufnahme ca. 10mA pro Eingang
FREIX..FREIC	Endstufe-Freigabe-Ausgänge, optoentkoppelt max. Belastung ca. 20mA

ACHTUNG - Stand-Alone-Betrieb über serielle Schnittstelle:
Zwischen der Interpolator-Karte und der DC-Karte bestehen keinerlei Verbindungen zur Stromversorgung. Es müssen also BEIDE Karten mit den entsprechenden Spannungen versorgt werden (DC-Karte über die entsprechenden Pins am PC-Bus). Insbesondere muß zwischen beiden Karten eine Masse-Verbindung gelegt werden!

Stecker-Kompatibilität I332-DC1 / I332-DC2

Pin	I332-DC1	I332-DC2	Anmerkung
30 SUB-D	OUT8	VCC_EXT	Versorgung für die Optokoppler der Freigabe-
22 40pol	OUT9	VCC_EXT	Ausgänge +12-30 V gegen GND_EXT

6.3.2 Jumper

Auf der Karte sind keine Jumper vorhanden.

6.3.3 Beschaltung der Ein-/Ausgänge

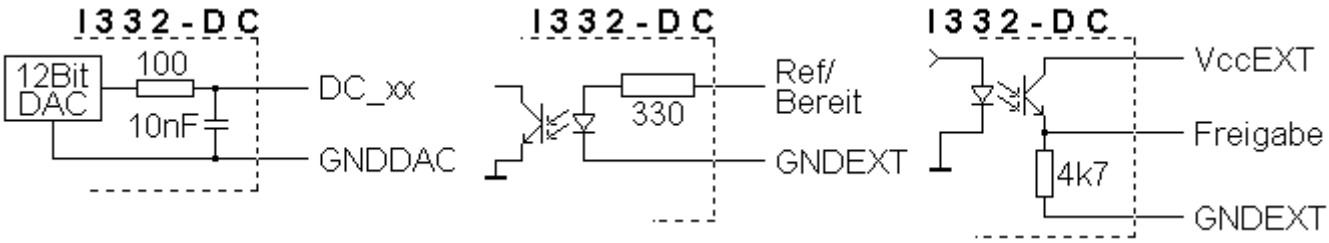


Abb. 6.24 a) DC-Ausgänge **b) Referenz-/Bereit-Eingänge** **c) Freigabe-Ausgänge**

Anmerkung: *zu Abb. 6.24a: Die Ansteuerung des DAC erfolgt über Optokoppler. Die Beschaltung der Drehgeber-Eingänge (Ax/Ax*/Bx/Bx*) ist identisch zur I332 (s. Abb. 6.3b), der Masse-Anschluß liegt jedoch nicht an GND sondern an GND_EXT.*

6.4 I332-IO

6.4.1 Steckerbelegung

Der Anschluß erfolgt über einen 62-poligen, dreireihigen High-Density-Sub-D-Stecker

		(22)	IN21		
(1)	GND			(43)	OUT12
		(23)	IN22		
(2)	IN1			(44)	OUT13
		(24)	IN23		
(3)	IN2			(45)	OUT14
		(25)	IN24		
(4)	IN3			(46)	OUT15
		(26)	IN25		
(5)	IN4			(47)	OUT16
		(27)	IN26		
(6)	IN5			(48)	OUT17
		(28)	IN27		
(7)	IN6			(49)	OUT18
		(29)	IN28		
(8)	IN7			(50)	OUT19
		(30)	IN29		
(9)	IN8			(51)	OUT20
		(31)	IN30		
(10)	IN9			(52)	OUT21
		(32)	OUT1		
(11)	IN10			(53)	OUT22
		(33)	OUT2		
(12)	IN11			(54)	OUT23
		(34)	OUT3		
(13)	IN12			(55)	OUT24
		(35)	OUT4		
(14)	IN13			(56)	OUT25
		(36)	OUT5		
(15)	IN14			(57)	OUT26
		(37)	OUT6		
(16)	IN15			(58)	OUT27
		(38)	OUT7		
(17)	IN16			(59)	OUT28
		(39)	OUT8		
(18)	IN17			(60)	OUT29
		(40)	OUT9		
(19)	IN18			(61)	OUT30
		(41)	OUT10		
(20)	IN19			(62)	GND_EXT
		(42)	OUT11		
(21)	IN20				

Anschlüsse zur Spannungsversorgung:

GND Bezugsmasse für die TTL-Eingänge IN1..IN30

GND_EXT Bezugsmasse für die optoentkoppelten Ausgänge OUT1..OUT30

Signale:

IN1..IN30 30 TTL-Eingänge

OUT1..OUT30 30 optoentkoppelte Ausgänge, Open-Collector, max. Belastung ca. 50mA

6.4.2 Jumper

Auf der Karte sind keine Jumper vorhanden.

6.4.3 Beschaltung der Ein-/Ausgänge

Die Eingänge der I332-IO sind einfache TTL-Eingänge, die Beschaltung der Ausgänge ist als optoentkoppelter Open-Collector ausgeführt (s.Abb. 6.25).

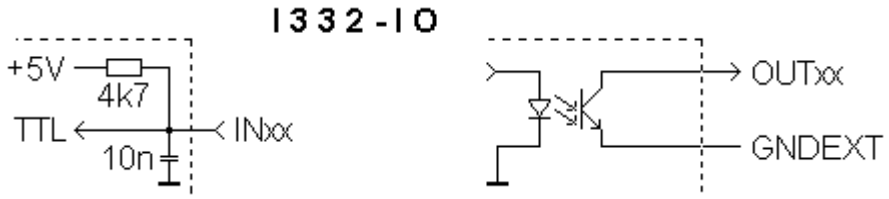


Abb. 6.25 Ein-und Ausgänge der I332-IO

6.5 Anmerkungen zum Anschluß der ISA-Zusatz-Karten

- Auf der Interpolator-Platine muß die 64polige Buchsenleiste (gegenüber des 37pol SubD-Steckers) bestückt sein. Der Anschluß der Zusatz-Platinen erfolgt mit dem beiliegenden Flachbandkabel.

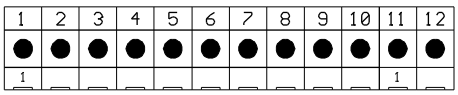
- Für den Stand-Alone-Betrieb:
  - Außer den obligatorischen 5V müssen DC1-Karten mit ±12-15V extern versorgt werden (über die entsprechenden ±12V-Anschlüsse an PC-Bus-Messerleiste)
  - Zwischen den Platinen muß eine Masse-Verbindung gelegt werden (Anschluß GND, ggf. auch GND_EXT)

6.6 I332-PCI

***Allgemeiner Hinweis zur Anschluss-Belegung bzw. zur internen Beschaltung der Ein- und Ausgänge**:*
Die interne Beschaltung der Ein-/Ausgänge entspricht, wenn nicht anders vermerkt, denjenigen der ISA-Version. Dies betrifft insbesondere die Analog-Eingänge, die wie bei der ISA-Version mit 5V versorgt werden.

6.6.1 AM167/AM168-Anschaltmodule, allgemeine Pinbelegung

Schraubklemme 1:

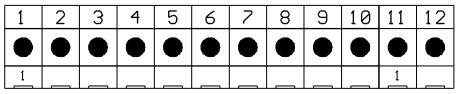


12polige PHOENIX- Schraubklemme

PIN:

1 Endschalter +X
2 Endschalter +Y
3 Endschalter +Z
4 Endschalter +U
5 GND-extern (GND- Endschalter)
6 Endschalter -X
7 Endschalter -Y
8 Endschalter -Z
9 Endschalter -U
10 GND-extern (GND- Endschalter)
11 VCC-extern (+24V)
12 GND-extern

Schraubklemme 2:



12polige PHOENIX- Schraubklemme

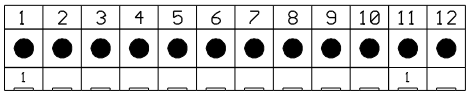
PIN:

1 VCC-extern (+24V)*
2 GND-extern*
3 +5V-intern
4 Analog-Eingang 1
5 Analog-Eingang 2
6 Analog-Eingang 3
7 Analog-Eingang 4
8 GND-intern
9 Handrad A
10 Handrad B
11 Notaus
12 GND-extern (GND- Notaus)

Abb. 6.26 Schraubklemmen 1+2 - AM-Modul

***Hinweis** zu Schraubklemme 2: Zur Versorgung von Potentiometern an den Analog-Eingängen (Pin4-7) bzw. einem Handrad (Pin9/10) steht an Pin3 eine 5V-Spannung zur Verfügung; die zugehörige Masse an Pin8 ist NICHT mit GND-extern verbunden.*

Schraubklemme 3:



12polige PHOENIX- Schraubklemme

PIN:

1	Endschalter +V
2	Endschalter +A
3	Endschalter +B
4	Endschalter +C
5	GND-extern (GND- Endschalter)
6	Endschalter -V
7	Endschalter -A
8	Endschalter -B
9	Endschalter -C
10	GND-extern (GND- Endschalter)
11	VCC-extern (+24V)
12	GND-extern

Abb. 6.27 Schraubklemme 3, AM-Zusatzmodul

6.6.2 AM167/AM168-Anschaltmodul – Schrittmotor-Version

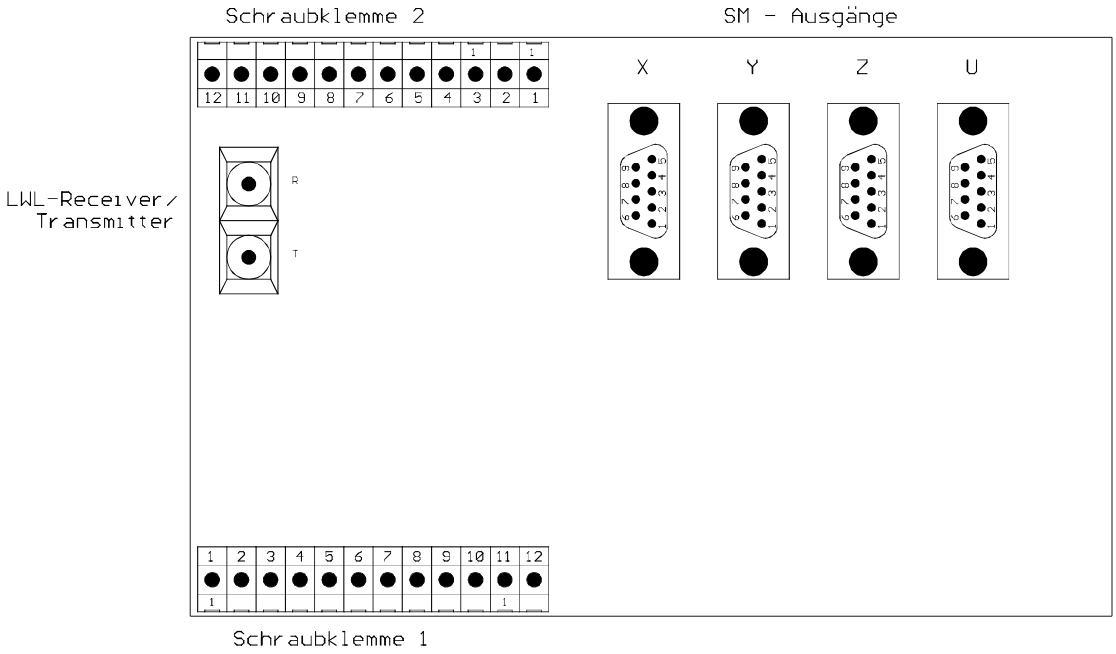


Abb. 6.28 -Modul - Anordnung der Anschlüsse - SM-Modul

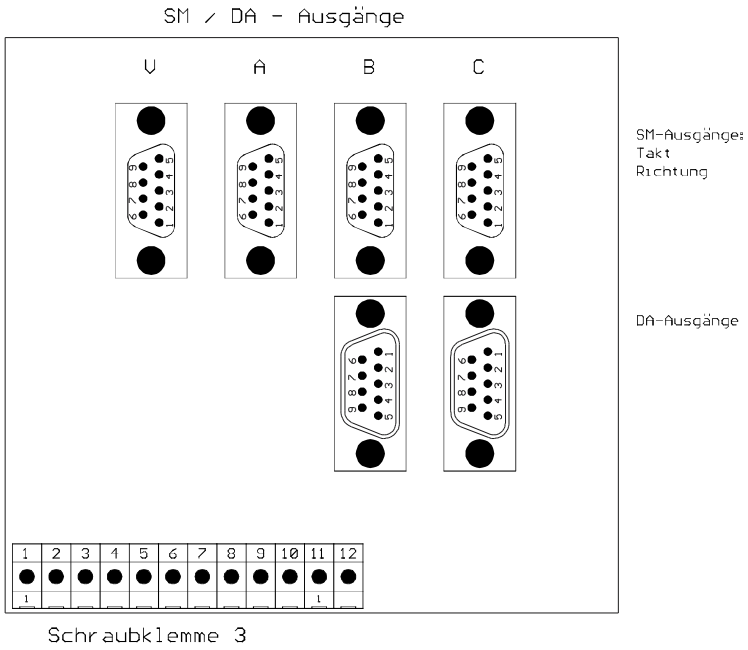
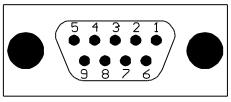


Abb. 6.29 Anordnung der Anschlüsse – SM-Zusatzmodul

(DA-Anschlüsse optional)

SM - Ausgang



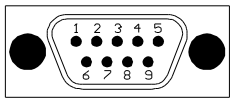
9pol. SubD Buchse

PIN:

1	VCC-extern (+24V)*
2	Takt
3	Richtung
4	nc.
5	nc.
6	nc.
7	nc.
8	nc.
9	GND-extern*

DA - Ausgang

(Analog-Ausgang: -10V bis +10V)



9pol. SubD Stecker

PIN:

1	nc.
2	nc.
3	nc.
4	Analog-Ausgang (-10V bis +10V)
5	Analog-GND (GND Analog-Ausgang)
6	nc.
7	nc.
8	nc.
9	nc.

Abb. 6.30 SM-Modul - Steckerbelegung (DA-Ausgang optional)

6.6.3 AM167/AM168-Anschaltmodul – Servo-Version

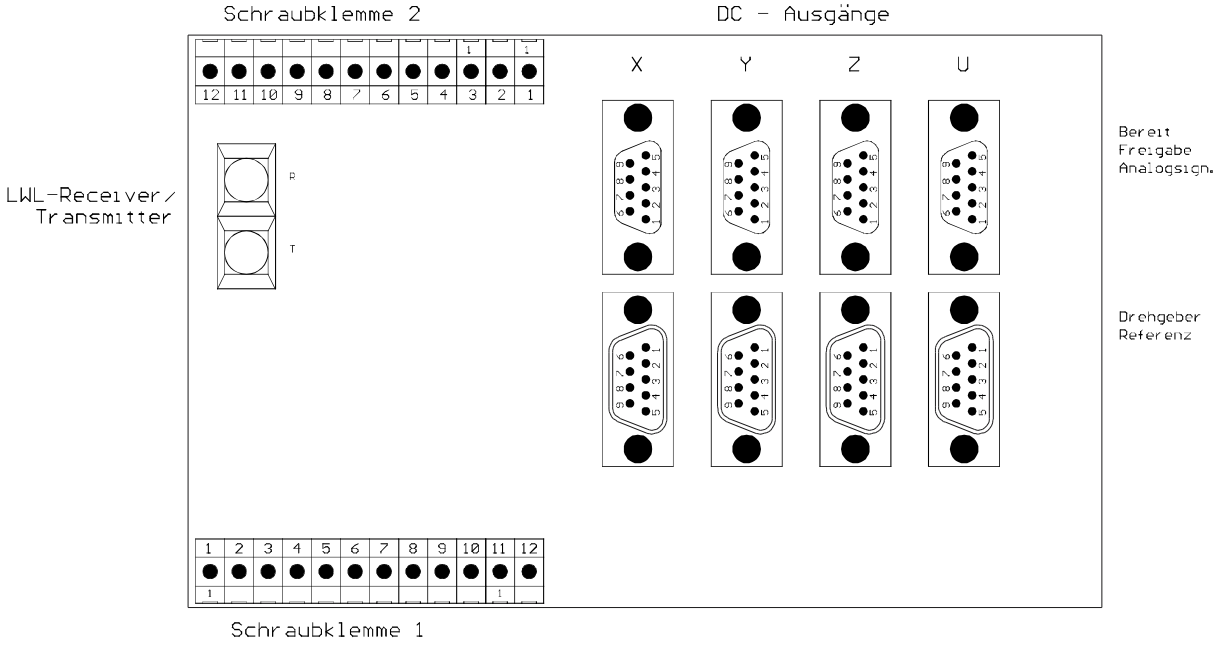


Abb. 6.31 Anordnung der Anschüsse – Servo-Modul

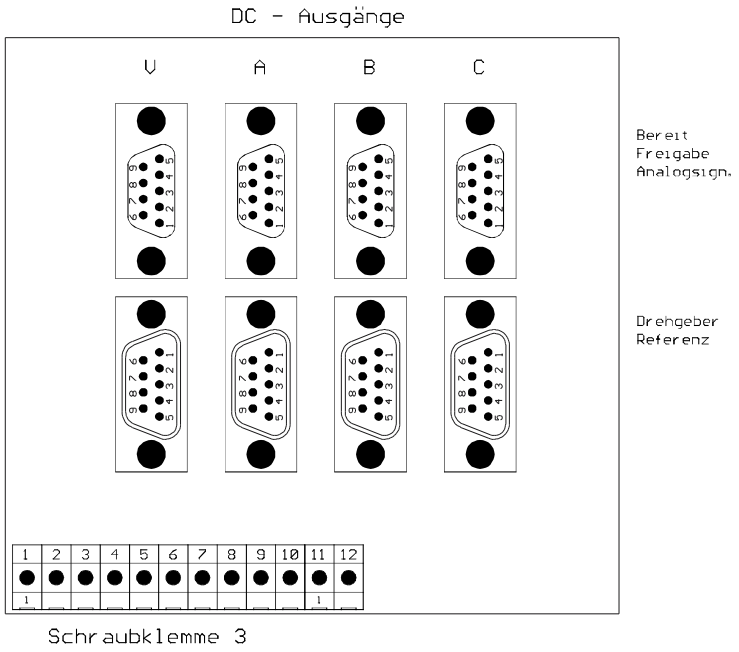
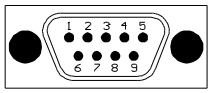


Abb. 6.32 Anordnung der Anschlüsse – Servo-Zusatzmodul

DC - Ausgang
(Drehgeber / Referenz)

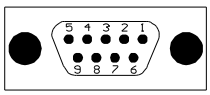


9pol. SubD Stecker

PIN:

1	A
2	A\
3	B (Messtaster- Eingang)
4	B\ (Messtaster- GND)
5	- Referenz-Eingang (TTL- Sign.)
6	nc.
7	nc.
8	+ Referenz-Eingang (TTL- Sign.)
9	GND-extern*

DC - Ausgang
(Bereit / Freigabe / Analogsign.)



9pol. SubD Buchse

PIN:

1	VCC-extern (+24V)*
2	nc.
3	nc.
4	Analog-Ausgang (-10V bis +10V)
5	Analog-GND (GND Analog-Ausgang)
6	Freigabe-Ausgang
7	nc.
8	Bereit-Eingang
9	GND-extern*

Abb. 6.33 Servo-Modul - Steckerbelegung (Messtastereingang optional)

6.6.4 CAN-IO-Modul

Weitere technische Details s. Handbuch CAN-IO.PDF.

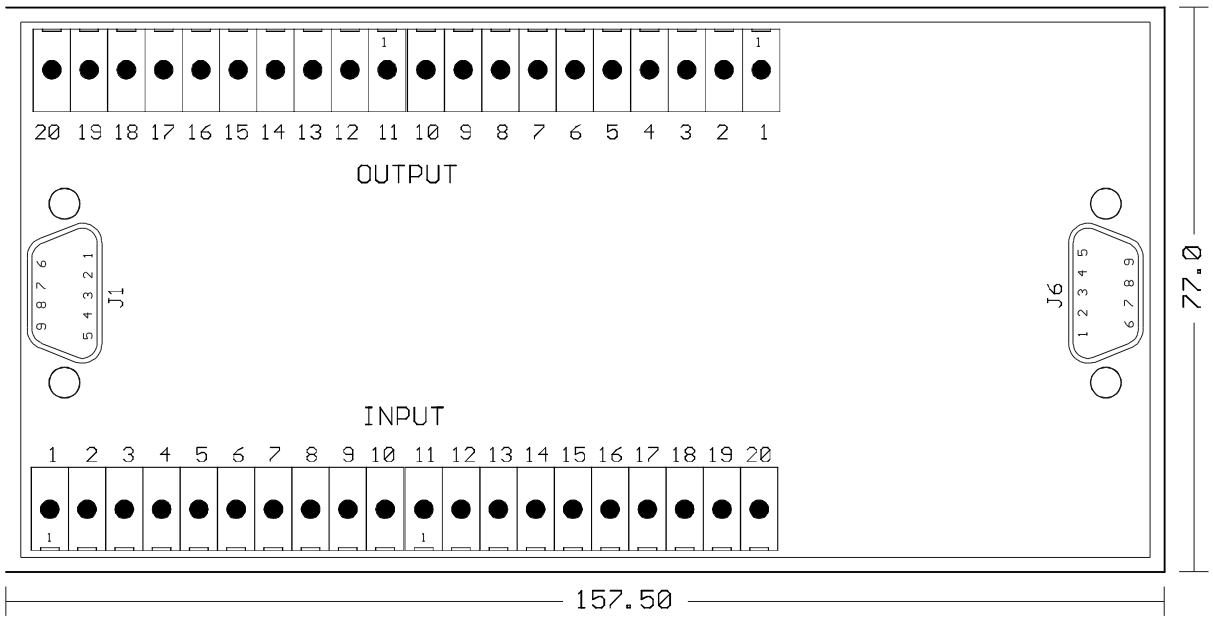


Abb. 6.34 CAN-IO-Modul - Anordnung der Anschlüsse

OUTPUT Gruppe A:

Pin:		
	1	+24V extern (A)
	2	GND extern (A)
	3	Output 1
	4	Output 2
	5	Output 3
	6	Output 4
	7	Output 5
	8	Output 6
	9	Output 7
	10	Output 8

OUTPUT Gruppe B:

Pin:		
	11	+24V extern (B)
	12	GND extern (B)
	13	Output 9
	14	Output 10
	15	Output 11
	16	Output 12
	17	Output 13
	18	Output 14
	19	Output 15
	20	Output 16

INPUT:

Pin:		
	1	GND Input
	2	Input 1
	3	Input 2
	4	Input 3
	5	Input 4
	6	Input 5
	7	Input 6
	8	Input 7
	9	Input 8
	10	GND Input
	11	GND Input
	12	Input 9
	13	Input 10
	14	Input 11
	15	Input 12
	16	Input 13
	17	Input 14
	18	Input 15
	19	Input 16
	20	GND Input

J1 und J6:
CAN-BUSSCHNITTSTELLE
9poliger SubD Stecker

Pin:		
	1	frei
	2	CANL
	3	GND-intern
	4	frei
	5	frei
	6	GND-intern
	7	CANH
	8	frei
	9	frei

120 Ohm

Abb. 6.35 CAN-IO-Modul – Steckerbelegung

***Hinweis**:* *Die beiden CAN-Bus-Anschlüsse J1/J6 sind intern verbunden und erlauben so das Aneinanderreihen mehrerer CAN-IO-Module.*
Um beide OUTPUT-Gruppen (A+B) nutzen zu können, müssen sowohl Pin1/2 als auch Pin11/12 an die Spannungsversorgung angeschlossen werden.

7 Beispielprogramme

Dieses Kapitel wurde aus dem Handbuch entfernt. Statt dessen wird auf Demo-Programm „I332Demo.EXE“ verwiesen, das die Verwendung der I332.DLL demonstriert und deshalb im Handbuch I332_DLL.PDF beschrieben wird.

A Anhang - Inbetriebnahme / Fehlerdiagnose

Allgemeines

Hinweise: *Es wird dringend empfohlen, vor der Inbetriebnahme von Maschinen einen* ***Notaus-Schalter*** *zu installieren,* ***der sämtliche Komponenten der Anlage stromlos schaltet!***
Allgemein gültige Sicherheitsregeln sind unbedingt zu beachten.
Inbetriebnahme / Fehlerbehebung darf nur von geschultem Fachpersonal vorgenommen werden.
Für Schäden kann keinerlei Haftung übernommen werden.

Zur Kontrolle der Signale ist ein Vielfach-Meßinstrument (Sollwert-Spannung, Richtungs-, Freigabe- und Bereit-Signal) sowie ein Oszilloskop nützlich (Drehgeber-Signale, Schritt-Impulse).

Bei den Anschluß-Bezeichnungen steht im folgenden "xx" für die jeweilige Achsbezeichnung am Stecker (XYZUVABC).

Die Signale für die Achsen VABC können über ein 1:1 Flachbandkabel an der 40pol. Steckleiste auf der I332-ISA bzw. der I332-DC ebenfalls auf eine 37pol-SubD-Buchse geführt werden und liegen dann analog zu den Signalen der Achsen XYZU.

Hinweis

Es wird darauf hingewiesen, daß bei allen Einstellarbeiten an dynamischen Systemen unvorhergesehene Reaktionen und/oder Systemzustände enstehen können. Beim Einstellen von Servo-Motoren kann es insbesondere zu starkem Schwingen der Motoren kommen (am schnellsten passiert das durch zu große P- und D-Faktoren!) oder zu unvorhergesehenen, auch ruckartigen Achsbewegungen.

Für Personenschäden oder Schäden an den Systemen kann daher keine Haftung übernommen werden !!!!!!!!

A.1 Inbetriebnahme von Maschinen

A.1.1 Spannungsversorgung

Wegen der optoentkoppelten Signale wird zum Betrieb eine externe Spannung von 12-30V benötigt. Es sei nocheinmal auf folgendes hingwiesen:

- Wird die I332 im PC betrieben, so besteht über den PC-Bus eine Verbindung zwischen den GND-Leitungen sowie zwischen den Anschlüssen zur +5V-Versorgung, **nicht jedoch zwischen den Anschlüssen GND_EXT**. An den Stecker-Pins +5V bzw. GND liegt die 5V-Versorgung des PCs. Im Stand-alone-Betrieb müssen diese Stecker-Pins extern beschaltet werden.
- Die externe Spannungs-Versorgung muß ebenfalls an alle I332-Platinen (I332, I332-DC und I332-IO) angeschlossen werden und zwar an die Anschlüsse U_Ext bzw. GND_EXT.
- Zwischen den Platinen besteht ansonsten keine direkte Verbindung der verschiedenen Spannungs- und Masse-Leitungen, insbesondere nicht zu GND_DAC (Bezugs-Masse der Sollwertspannung).
- Die für die Sollwert-Erzeugung notwendige +/-10V-Spannung wird intern über einen galvanisch getrennten DC-DC-Wandler aus der 5V-Spannung gewonnen.

Falls erforderlich muß der Potentialausgleich zwischen den einzelnen Stromversorgungs-Kreisen (+5V/GND, U_Ext/GND_EXT sowie GND_DAC) durch geeignete Maßnahmen im Schaltschrank sichergestellt werden.

***Hinweis**:* *An der Platine I332-DC ist der Anschluß für die externe Spannung mit VCC_EXT anstatt U_Ext bezeichnet.*

Stromaufnahme:

Die Optokoppler in den Eingangskreisen (Endschalter ESxx, Bereit-Eingänge Bereitxx, Referenz-Eingänge Refxx) benötigen ca. 10mA an jedem beschalteten Eingang. Der Strombedarf der Ausgänge hängt vom nachgeschalteten Eingang ab. Die Schrittmotor-Ausgänge Pulsxx bzw. Rtgxx sind in der Lage ca. 200mA, die Freigabe-Ausgänge Freixx ca. 20mA zu treiben

***Hinweis**:* *Wird als externe Spannung U_Ext eine an manchen Verstärkern zur Verfügung stehende Hilfsspannung benutzt, so ist zu beachten, daß diese Spannung nur an **einem** der Verstärker entnommen werden darf. Außerdem ist sicherzustellen, daß die Spannungsquelle den nötigen Strom liefern kann.*

A.1.2 Schrittmotoren

Verkabelung:

Der Anschluß von Schrittmotoren ist nach Abb. A.36 sehr einfach. Es sind lediglich Verbindungen zwischen den Puls- und Richtungssignalen von I332 und Schritt-Motor-Verstärker notwendig.

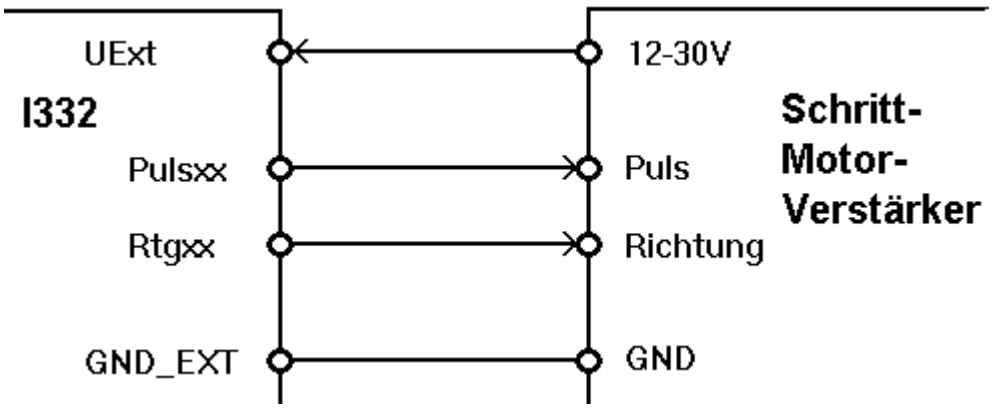


Abb. A.36 - Anschluß eines Schrittmotor-Verstärkers

Die Polarität der Signale kann über die Bits **apSM.smPlsPolB** bzw. **apSM.smDirPolB** ggf. angepaßt werden. Viele Verstärker bieten alternativ die Möglichkeit, diese Anpassung per Jumper vorzunehmen.

I332, 37pol-SubD-Buchse	
Bezeichnung	Pin-Nr.
UExt	36
GND_EXT	18
PulsX	6
RtgX	25
PulsY	7
RtgY	26
PulsZ	8
RtgZ	27
PulsU	9
RtgU	28

A.1.3 Servomotoren

Hinweise: *Es wird empfohlen, die Motoren soweit als möglich in nicht eingebautem Zustand in Betrieb zu nehmen.*
*Ebenso sollten die Motoren **einzeln** in Betrieb genommen werden; am einfachsten wird dazu der Achsparameter **apChar** aller nicht benötigten Achsen auf 0 gesetzt sowie der zugehörige Verstärker stromlos geschaltet.*

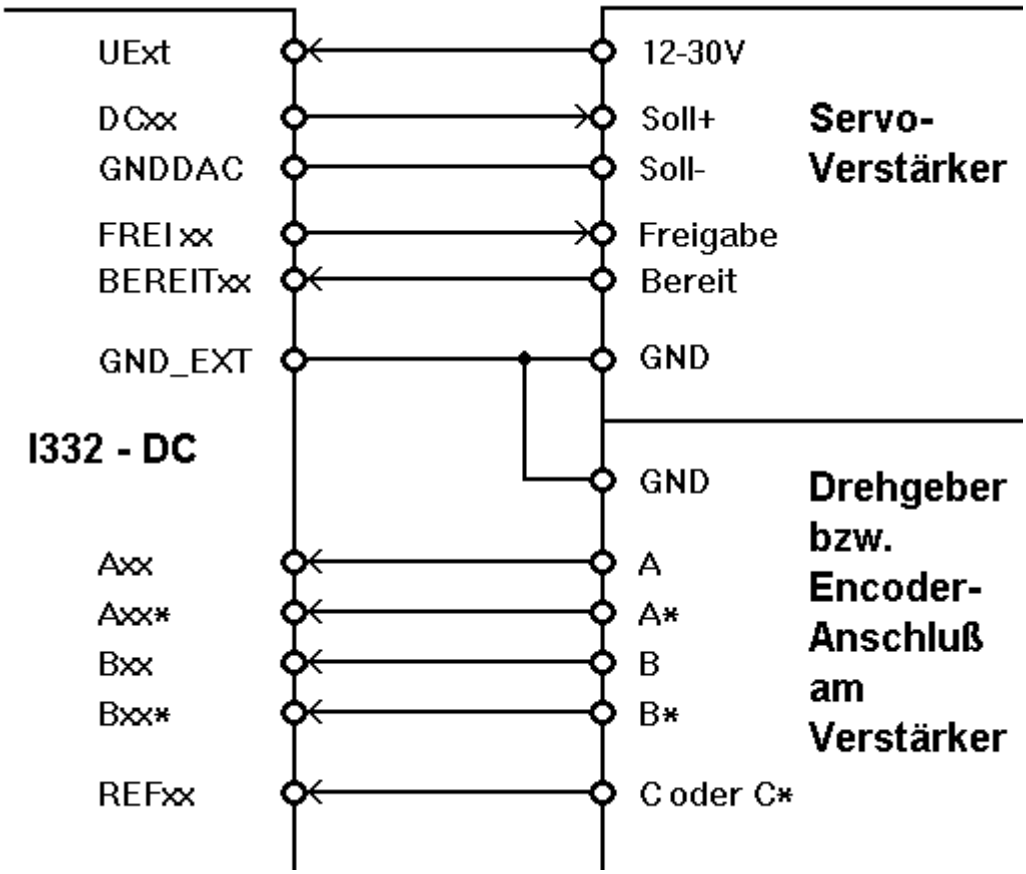


Abb. A.37 - Anschluß eines Servo-Verstärkers

Zur Kontrolle der Verkabelung sind vier Funktionen wichtig (s.a. Kap. 4.8):

1. Beschreiben des DA-Wandlers mit dem Befehl **icSetDAC**
2. Schreiben des Freigabe-Ausgangs mit dem Portbefehl **icSetOutBuf+pgEn**
3. Lesen des Drehgebers mit dem Befehl **icGetDG**
4. Bereit- und Referenz-Signal lesen mit den Portbefehlen **icGetInPort+pgRef** und **icGetInPort+pgRdy**

Es ist ein kleines Programm erforderlich, das es erlaubt, den DA-Wandler und den Freigabe-Ausgang einer Achse zu schreiben sowie das Drehgeberregister und die Referenz- und Bereit-Eingänge einer Achse zu lesen.

Für die korrekte Funktion des Regelkreises ist es zunächst wichtig, daß sowohl die Polarität der Sollwertspannung als auch die Zählrichtung des Drehgebers korrekt eingestellt sind. Ist das nicht der Fall, so läuft der Motor beim Einsetzen der Regelung in die falsche Richtung los und schaltet dann mit Schleppfehler ab! In diesem Fall muß entweder die Polarität der Sollwertspannung ODER die Zählrichtung der Drehgeber geändert werden – s. die beiden folgenden Kapitel.

A.1.3.1 Sollwertspannung und Freigabe

Verkabelung:

Die Sollwert-Spannung wird von der I332 an den Ausgängen DAC_GND und DC_xx ausgegeben und muß an die entsprechenden Eingänge des Verstärkers geführt werden.

Das Freigabe-Signal steht an den Ausgängen GND_EXT und FREIxx zur Verfügung und wird ebenfalls auf die entsprechenden Eingänge des Verstärkers geführt. Die Spannungsversorgung des Ausgangs-Treibers erfolgt über UExt.

Wird eine **negative** Sollwertspannung ausgegeben, so sollte sich der Motor so drehen, daß sich die Maschinen-Achse in **positiver** Richtung bewegt und umgekehrt.

Anmerkung: *Dieses Verhalten widerspricht (leider) den üblichen Konventionen (pos. Spannung → pos. Zählrichtung) und ist bedingt durch invertierende Optokoppler bei der Ansteuerung der DA-Wandler.*

Sollte dies nicht der Fall sein, so müssen zwei Fälle unterschieden werden:

a) Die Sollwert-Eingänge am Verstärker sind **potentialfrei** (s. Verstärker-Handbuch):
In diesem Fall können die Signale GND_DAC und DC_xx ***am Verstärker*** getauscht werden.

b) Die Sollwert-Eingänge am Verstärker sind **nicht potentialfrei** (s. Verstärker-Handbuch):
In diesem Fall muß die Stromversorgung des Motors entsprechend geändert werden.

Ab Version 2.2.0 der I332-Firmware besteht die Möglichkeit über das Bit **apDC.dcDACInv** die Sollwertspannung für die DA-Wandler ***im Regelkreis*** zu invertieren und damit an die Verkabelung anzupassen.
NB*: Beim direkten Auslesen bzw. Beschreiben der DAC-Register mit den Befehle* ***icGetDAC*** *und* ***icSetDAC*** *hat dieses Bit keine Wirkung!*

I332-DC, 37pol-SubD-Buchse	
Bezeichnung	Pin-Nr.
UExt	30(!)
GND_EXT	18
GND_DAC	11
DC_X	9
DC_Y	28
DC_Z	10
DC_U	29
FREIX	12
FREIY	31
FREIZ	13
FREIU	32

A.1.3.2 Richtung der Drehgeberimpulse

Verkabelung:

Die Drehgeber-Eingänge Axx/Axx* und Bxx/Bxx* werden mit den entsprechenden Ausgängen des Drehgebers / Resolvers bzw. den Encoder-Ausgängen am Verstärker verbunden. Der Masse-Anschluß ist GND_EXT.

Neben den Bezeichnungen A/A*/B/B* finden sich gelegentlich auch die folgenden Bezeichnungen an den Drehgebern:

Ua1 ←→ A
Ua1* ←→ A*
Ua2 ←→ B
Ua2* ←→ B*

Wird der Motor so gedreht (von Hand oder mit einer kleinen Sollwert-Spannung am Verstärker), daß sich die Maschinen-Achse in die **positive** Richtung bewegt, so muß auch der zughörige Zähler in **positiver** Richtung zählen und umgekehrt.
Hinweis: *Der Zähler läuft von 0 bis 65535 und springt dann zurück auf 0.*

Ist dies nicht der Fall, so können alternativ folgende Anschlüsse am Drehgeber / Encoder vertauscht werden:

Axx ←→ Axx*
oder
Bxx ←→ Bxx*
oder
Axx ←→ Bxx
und
Axx* ←→ Bxx*

Ab Version 2.2.0 der I332-Firmware besteht die Möglichkeit über das Bit **apDC.dcDGInv** die Auswertung der Drehgeber ***im Regelkreis*** zu invertieren und damit an die Verkabelung anzupassen.
***NB**: Beim direkten Auslesen der Drehgeber-Register mit dem Befehl **icGetDG** hat dieses Bit keine Wirkung!*

Drehgeber mit 1-phasigen Signalen:

Einige (wenige) Drehgeber liefern nur die Signale A und B, nicht aber die invertierten Signale A* und B*. In diesem Fall müssen die Eingänge Axx* und Bxx* auf Masse-Potential gelegt werden. Es wird jedoch ausdrücklich betont, daß diese Anschlußvariante nicht empfohlen wird, da sie aufgrund der unsymmetrischen Ansteuerung störanfälliger ist. Ggf. sollte beim Hersteller des Drehgebers nach Alternativen gefragt werden.

I332-DC, 37pol-SubD-Buchse			
Bezeichnung	Pin-Nr.	Bezeichnung	Pin-Nr.
GND_EXT	18	(GND	19)
AX	1	AX*	20
BX	2	BX*	21
AY	3	AY*	22
BY	4	BY*	23
AZ	5	AZ*	24
BZ	6	BZ*	25
AU	7	AU*	26
BU	8	BU*	27

A.1.3.3 Referenz-Signal

Verkabelung:

Falls notwendig / gewünscht wird das Referenz-Signal (Null-Impuls) des Drehgebers / Encoders oder ein Referenz-Schalter mit den Eingängen REFxx und GND_EXT verbunden.

Der Null-Impuls ist bei Drehgebern oft mit C bzw. C* (invertiertes Signal) gekennzeichnet. Daneben finden sich gelegentlich folgenden Bezeichnungen:

Ua0 ←→ N ←→ C
Ua0* ←→ N* ←→ C*

Welches der beiden Signale mit dem Eingang REFxx verbunden wird ist unkritisch, da die Polarität im Bedarfsfall auch über **apDC.dcRefPolB** festgelegt werden kann.

Wird der Referenz-Schalter separat mit Spannung versorgt und wird dazu eine Spannung aus dem PC verwendet, so müssen die Anschlüsse GND und GND_EXT verbunden werden! Aus diesem Grund wird diese Variante nicht empfohlen!

Wird als Referenz-Signal der Null-Impuls eines Drehgebers / Encoders verwendet, so ist zur Kontrolle ein Oszilloskop sinnvoller als die Abfrage des entsprechenden Eingangs durch ein Programm, da der Impuls i.d.R. nur einen Strich des Drehgebers / Resolvers breit ist!

I332-DC, 37pol-SubD-Buchse	
Bezeichnung	Pin-Nr.
GND_EXT	18
(GND	19)
REFX	14
REFY	33
REFZ	15
REFU	34

A.1.3.4 Bereit-Signal

Verkabelung:

Falls notwendig / gewünscht wird das Bereit-Signal des Verstärkers mit den Eingängen BEREITxx und GND_EXT verbunden.

I332-DC,37pol-SubD-Buchse	
Bezeichnung	Pin-Nr.
GND_EXT	18
BEREITX	16
BEREITY	35
BEREITZ	17
BEREITU	36

A.2 Endschalter

Der Anschluß der Endschalter für positive bzw. negative Achsrichtung erfolgt an den Eingängen ES+xx bzw. ES-xx der I332-Karte. Es ist darauf zu achten, daß der Endschalter für die positive Achsrichtung an den Anschluß ES+xx, der Endschalter für die negative Achsrichtung an den Anschluß ES-xx geführt wird.

Es wird empfohlen, Öffner als Endschalter zu verwenden, da sonst ein offener Eingang bzw. Kabelbruch nicht als Fehler erkannt werden kann!

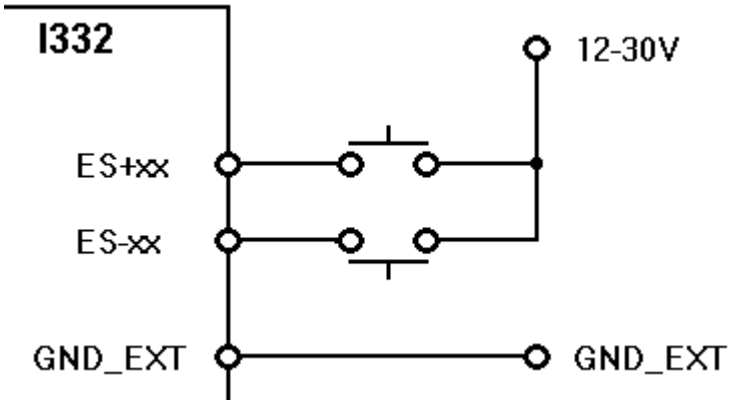


Abb. A.38 - Anschluß von Endschaltern

***Hinweis**:* *Abb. A.38 zeigt den Anschluß von mechanischen Endschaltern; selbstverständlich sind aber auch Transistor-Schalter möglich, die eine Schaltspannung an die ES-Eingänge liefern!*

Ab Version 2.2.0 der I332-Firmware besteht die Möglichkeit über das Bits **apLS.lsChg** die Zuordnung ES+/ES- zur Achsrichtung zu vertauschen. Es wird jedoch empfohlen, von dieser Möglichkeit nur Gebrauch zu machen, wenn bei einer schon verkabelten / konfigurierten Maschine eine solche Änderung dauerhaft notwendig sein sollte.

I332, 37pol-SubD-Buchse	
Bezeichnung	Pin-Nr.
GND_EXT	18
ES+X	2
ES-X	21
ES+Y	3
ES-Y	22
ES+Z	4
ES-Z	23
ES+U	5
ES-U	24

A.3 Fehlerbehandlung

Fehler	**Überprüfen**	**Behebung**
Achsen bewegen sich nicht und Positionszähler **icGetPos** zählt nicht	Poti angeschlossen	Poti aufdrehen
	Anschluß prüfen	korrekt anschließen
	Maschinendaten prüfen	**mpPotiInit** korrekt setzen (z.B $00008000) oder Poti abschalten mit **icSetOvr**
	kein Poti angeschlossen	**mpPotiInit** korrekt setzen (z.B $80008000) oder Poti abschalten mit **icSetOvr**
- Servomotoren:	Bereit-Eingänge prüfen	korrekt anschließen
	Achsparameter prüfen	**apDC.dcRdyInB** / **apRdyPolB** korrekt setzen
Achsen bewegen sich nicht, aber Positionszähler **icGetPos** zählt	I332-Version/Achsanzahl	Distributor konsultieren
- Schrittmotoren:	Verkabelung / Treiber	korrekt verkabeln, s. Handbuch des Treiberherstellers
	Achsparameter prüfen	- **apChar.acSMB** = 1 - alle anderen = 0 - **apSM** korrekt?
- Servomotoren:		
kein Schleppfehler	Achsparameter prüfen	- **Char.acDCB** = 1 - alle anderen = 0 - **dcRdyInB** / **dcRdyPolB** korrekt?
bei Schleppfehler	Verkabelung / Treiber	- korrekt verkabeln, s. Handbuch des Treiberherstellers - s.o. Inbetriebnahme-Hinweise
	Achsparameter prüfen	- **apDC.dcEnOutB** = 1 - **apDC.dcEnPolB** korrekt?
Servomotoren laufen sofort bei Einsetzen der Regelung los und schalten danach mit „Schleppfehler“ ab.	Motor dreht in die falsche Richtung bzw. Zählrichtung der Drehgeber falsch	Sollwertspannung oder Zählrichtung der Drehgeber ändern (s. A.1.3.1 bzw. A.1.3.2)
Endschalter-Fehler wird für beide Richtungen gemeldet, obwohl die Achse nicht auf einem Endschalter steht	Endschalter-Anschluß	korrekt anschließen
	Achsparameter prüfen	**ApLS**-Bits korrekt?
	***Achtung**: beim Initialisieren der Maschinendaten müssen die Endschalter-Eingänge bereits gültige Werte liefern; d.h. die Versorgungs-Spannung der Endschalter sollte bereits beim Einschalten des Hauptschütz zur Verfügung stehen.*	
Interpolation ist nicht kontinuierlich	Details / Lösungen	s. Kap. 5.1
	mögliche Ursachen	- Satzübergänge unstetig - Sätze zu klein - Satzübergabe zu langsam - bei Servo-Antrieben: **apInPos** zu klein
Bahngeschwindigkeit wird nicht erreicht	Details / Lösungen	s. Kap. 5.1
	mögliche Ursachen	- **apAcc**/**apDec** zu klein - **apVMax** einer Achse zu klein - Sätze zu klein - Verhältnis Bahn / Nichtbahnachse zu klein

Ein Wegstück wird programmiert, die Bewegung startet jedoch nicht	Korrekte Programmierung überprüfen	s. Kap. 5
	Startbit(s) nicht gesetzt	s. Kap. 4.7.1
- Servomotor	Mind. eine der beteiligten Achsen steht außerhalb des In-Positionsfensters	Servo-/Regeleinstellungen korrigieren: - Servoregler korrekt einstellen (s. Handbuch des Servo-Herstellers) - Parameter des Lageregelkreises korrekt einstellen (**apPFac**, **apIFac**, **apDFac**, **apVFac**), s. Kap. B.2 - In-Positionsfenster vergrößern (**apInPos**) - Weitere Details s. Kap. B.3

A.4 Problemlösungen

Problem	Lösung
Richtung einer Achse ändern	
- Schrittmotor	Invertieren des Bits **apSM.smDirPol**
	Alternativ bieten viele Schrittmotor-Verstärker die Möglichkeit, die Richtung des Motors per Jumper festzulegen. Sind die Verstärker-Eingänge potentialfrei, so können auch die Anschlüsse getauscht werden.
- Servomotor	Bei Einsatz von Servo-Motoren ist die Dreh-Richtung des Motors durch seine Anschlußbelegung festgelegt.
	Besitzt der eingesetzte Servo-Verstärker einen **potentialfreien** Sollwert-Eingang (s. Handbuch des Verstärker-Herstellers), so kann durch folgende Maßnahmen (**und nur in diesem Fall!**) die Motorrichtung geändert werden: • Vertauschen der beiden Leitungen für den Sollwert am Verstärker (von den I332-DC-Anschlüssen GND_DAC und DC_xx) ***und*** • Vertauschen der Signalpaare Axx ←→ Bxx und Axx* ←→ Bxx* des Drehgeber / Resolvers xx steht für die jeweilige Achse
	Ist der Sollwert-Eingang **nicht potentialfrei**, so bleibt als Alternative die Stromversorgung des Motors entsprechend zu ändern; ggf. müssen auch in diesem Fall die Signalpaare Axx/Axx* und Bxx/Bxx* wie oben getauscht werden, je nachdem ob ein Drehgeber auf der Motorachse oder ein simulierter Encoder im Verstärker das Positionssignal liefert
	Ab Version 2.2.0 der I332-Firmware lassen sich diese Vertauschungen auch per Software vornehmen (**apDC.dcDGInv** bzw. **apDC.dcDACInv**).
Achtung: *Die Endschalter müssen in beiden Fällen ebenfalls vertauscht werden!! Ab Version 2.2.0 der I332-Firmware läßt sich diese Vertauschung auch per Software vornehmen (**apLS.IsChg**).*	

B Anhang - Einstellung von Maschinendaten

Allgemeines

Auf einen wichtigen Unterschied sei an dieser Stelle besonders hingewiesen:

Die **Rampenparameter** (**apVMax**, **apVStart**, **apVStop**, **apAcc** und **apDec**) bestimmen das Verhalten des Interpolators, d.h. wie groß die Maximalgeschwindigkeit ist und auf welche Weise sie erreicht wird oder, anders ausgedrückt, welche Folge von Sollpositionen erzeugt wird.
Dagegen bestimmen die **Regelparameter** (**apPFac**, **apIFac**, **apDFac** und **apVFac**) das Verhalten des nachgeschalteten PID-Reglers, d.h. auf welche Weise versucht wird, die Achsen auf die vom Interpolator erzeugten Sollpositionen zu fahren.

Hinweis

Es wird darauf hingewiesen, daß bei allen Einstellarbeiten an dynamischen Systemen unvorhergesehene Reaktionen und/oder Systemzustände enstehen können. Beim Einstellen von Servo-Motoren kann es insbesondere zu starkem Schwingen der Motoren kommen (am schnellsten passiert das durch zu große P- und D-Faktoren!) oder zu unvorhergesehenen, auch ruckartigen Achsbewegungen.

Für Personenschäden oder Schäden an den Systemen kann daher keine Haftung übernommen werden !!!!!!!!

B.1 Einstellen der Rampenparameter

Nicht immer sind die Motorparameter bekannt, aus denen sich die Achsparameter ableiten lassen. Deshalb folgen hier einige praktische Hinweise, die sich vor allem auf Schrittmotoren und symmetrische Start/Stop-Rampen (**apVStart** = **apVStop** und **apAcc** = **apDec**) beziehen.

Erforderlich ist ein kleines Test-Programm, das eine Achse um eine bestimmte Strecke im Eilgang hin- und her bewegt.
Achten Sie darauf, daß die Achsauflösung (**apStep** und **apmm**) korrekt definiert ist.

Für die Parameter **apVMax**, **apVStart**, **apVStop**, **apAcc** und **apDec** hat sich folgendes Vorgehen als praktikabel erwiesen:

- In einem 1. Schritt werden die Start- und die Stopgeschwindigkeit optimal eingestellt.
  - Beginnen Sie mit kleinen Werten für Start-, Stop- und Maximalgeschwindigkeit und zwar so, daß **apVStart** = **apVStop** <= **apVMax** und **apAcc** = **apDec** = 0 ist.
    Damit ist gewährleistet, daß die Beschleunigungs- und Verzögerungswerte **apAcc** und **apDec** keinen Einfluß haben.
    Beim Starten des Testprogramms sollte die Achse nun ohne Probleme mit der eingestellten Start/Stop-Geschwindigkeit ohne Rampe(!) verfahren.
  - Nun werden die drei Parameter sukzessive erhöht, bis die Grenzen des Motors gefunden sind (immer darauf achten, daß **apVStart** = **apVStop** <= **apVMax**).

- Im 2. Schritt wird die Maximalgeschwindigkeit optimiert.
  - Dazu wird für die Beschleunigung und die Verzögerung (**apAcc** = **apDec**) ein so kleiner Wert gewählt, den der Motor mit Sicherheit verkraftet.
  - Nun wird sukzessive die Maximalgeschwindigkeit **apVMax** erhöht, bis die Grenzen des Motors erreicht sind.
    Achten Sie darauf, daß die eingestellte Maximalgeschwindigkeit beim Verfahren des Motors auch tatsächlich erreicht wird, d.h. im Test-Programm ist die Anzeige der Bahngeschwindigkeit erforderlich! Ggf. muß der Verfahrweg vergrößert werden.

- Im 3. Schritt werden die Beschleunigung und die Verzögerung optimiert.
  Dazu wird analog zum bisherigen Vorgehen sukzessive **apAcc** und **apDec** erhöht.

Um diese gefundene Start/Stoprampe bei Bedarf asymmetrisch zu gestalten, kann ausgehend von diesen Werten in einem 4. Schritt systematisch jeder der 4 Parameter **apVStart**, **apVStop**, **apAcc** und **apDec** getrennt optimiert werden.

Die folgende Abbildung B.1 verdeutlicht die Zusammenhänge:
Zum Zeitpunkt ***t1*** wird die Bewegung mit der Geschwindigkeit **apVStart** begonnen, danach wird mit dem Beschleunigungswert **apAcc** auf die Maximalgeschwindigkeit **apVMax** (bzw. auf eine im Verfahrsatz programmierte Geschwindigkeit) beschleunigt. Diese Geschwindikeit wird (abhängig von **apAcc**) zum Zeitpunkt ***t2*** erreicht.
Zum Zeitpunkt ***t3*** beginnt die Verögerungsphase, in der von der momentanen Geschwindigkeit mit dem Wert **apDec** auf die Geschwindigkeit **apVStop** verzögert wird. Diese Geschwindigkeit wird (abhängig von **apDec**) zum Zeitpunkt ***t4*** erreicht. Danach steht die Achse wieder.

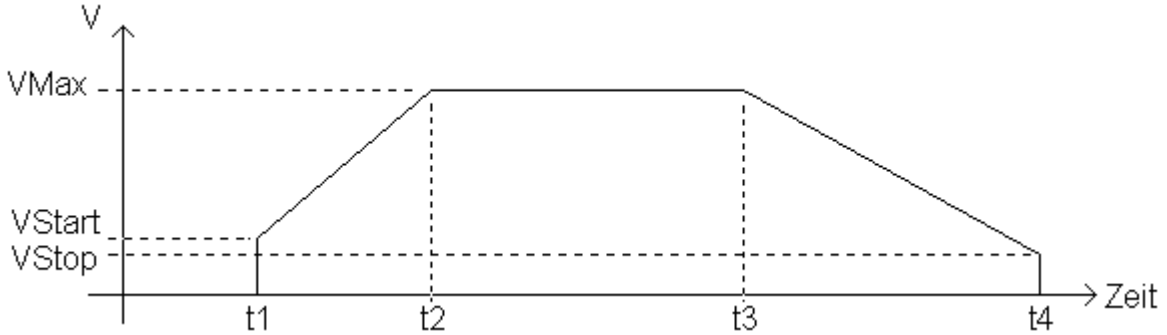


Abb. B.39 Geschwindigkeitsprofil einer Achsbewegung

Die Abb. B.39 verdeutlicht auch, daß **apVStart** und **apVStop** sowie **apAcc** und **apDec** nicht notwendigerweise gleich groß sein müssen (im Beispiel ist **apVStart** > **apVStop** und **apAcc** > **apDec**: hohe Beschleunigung - niedrigere Verzögerung).
Ist statt der Beschleunigung **apAcc** die Hochlaufzeit (= ***t2*** – ***t1***) bekannt, so kann nach der entsprechenden physikalischen Formel die Beschleunigung bestimmt werden:

apAcc = (**apVMax** - **apVStart**) / (60 * (***t2***-***t1***))

Analog läßt sich ggf. auch **apDec** bestimmen:

apDec = (**apVMax** - **apVStop**) / (60 * (***t4***-***t3***))

B.2 Einstellen der PID-Regel-Parameter

Beim Einsatz der Zusatzkarte I332-DC zum Anschluß von Servo-Motoren sollten zunächst die Regelparameter optimiert werden, und zwar bevor die Rampen bestimmt werden (es sollten dafür kleine Werte eingesetzt werden). Dazu ist wiederum ein kleines Testprogramm zum Bewegen einer Achse hilfreich.

Folgende Vorgehensweise ist dabei empfehlenswert:

- Wichtigster Punkt:
  Einstellung des Servo-Verstärkers nach Herstellerangaben
  (Nullpunktabgleich, Stromverstärkung, Drift etc...)

- Die Schleppfehler-Überwachungen **mpFEFatal** und **apFEMax** sowie die In-Position-Überwachung **apInPos** sollten auf ihren Maximalwert gesetzt werden, um Fehlinterpretationen auszuschließen.

- Als nächstes wird der P-Anteil (**apPFac**) des Reglers soweit erhöht, bis der Schleppfehler ein Minimum erreicht, ohne daß die Achse zu schwingen beginnt.

- Danach erfolgt die Einstellung des I-Anteils (**apIFac**), so daß während der Achsbewegung der Schleppfehler minimal wird. Besonderes Augenmerk sollte auf das Bewegungsende gelegt werden, da ein Regler mit zu großem I-Anteil zum Überschwingen (über die Soll-Position hinaus → VZ-Wechsel des Schleppfehlers) neigt.

- Zuletzt wird mit Hilfe des D-Anteils (**apDFac**) das Impulsverhalten (Abbremsen / Beschleunigen) optimiert (minimaler Schleppfehler).

- Diese Schritte können nun mehrfach wiederholt werden.

Das Einstellen der Rampen kann nun entspr. Kap. B.1 erfolgen. Bei Servomotoren ist besonders auf das Anfahr- und Stop-Verhalten (**apVStart**, **apVStop**) zu achten. Die Parameter **apAcc** und **apDec** sind im Vergleich zum Schrittmotor wesentlich unkritischer. Sie sind zu groß, wenn der Schleppfehler bei Erreichen der Sollgeschwindigkeit über- bzw. beim Abbremsen unterschwingt.

Eine weitere Einstell-Möglichkeit bietet der Parameter **apVFac**. Dieser erzeugt einen zusätzlichen Spannungsoffset zum Sollwert, der proportional zur Sollgeschwindigkeit ist. Eine optimale Einstellung dieses Parameters führt ebenfalls zum Verschwinden des Schleppfehlers und ist damit in seiner Wirkung mit dem I-Anteil des Regler vergleichbar. Ein Regler mit den Parametern P und V hat jedoch nicht den Nachteil des Überschwingens, da der V-Anteil mit dem Verringern der Geschwindigkeit ebenfalls kleiner und schließlich zu 0 wird. Dafür ist diese Variante in höherem Maße lastabhängig als ein PI- oder PID-Regler.

***Hinweise**: Es wird empfohlen, entweder nur die P-, I- und D-Parameter oder nur die Parameter P und V zu verwenden.*
*Für den Einstellvorgang sollten die Parameter **apInPos** und **apFEMax** groß (max. 32000) gewählt werden, um dadurch bedingte Einflüsse auszuschalten.*

B.3 Anmerkungen zum Ablauf des PID-Algorithmus:

B.3.1 Maschinenparameter

a) Achsparameter **apChar**:
   acDCB muß gesetzt sein
b) Achsparameter **apDC**:
   Es muß explizit angegeben werden ob der Servo-Verstärker einen Freigabe-Eingang (i.d.R. immer) bzw. einen Ready-Ausgang besitzt (Bits **dcEnOutB** bzw. **dcRdyInB**). Außerdem muß die Polarität korrekt angegeben werden (Bits **dcEnPolB** bzw. **dcRdyPolB**).
c) PID-Parameter (**apPFac**, **apIFac**, **apDFac**):
   Standardmäßig ist nur **apPFac** = 20 programmiert (reiner P-Regler)
   ===>>> damit allein kann der Schleppfehler nicht auf 0 geregelt werden. Dazu müssen zusätzlich **apIFac** und **apDFac** oder **apVFac** programmiert werden (s. Kap B.2).

B.3.2 Ausgabe-Kontrolle

Die Ausgabe-Kontrolle erfolgt mehrstufig:

a) Überschreitet der Schleppfehler den Wert **apInPos**, so wird das entsprechende InPos-Bit im Register **PIDInfo** GELÖSCHT (Bit 23..16 = Achse 7..0). Am Satzende wird gewartet, bis der Schleppfehler diesen Wert unterschreitet, erst dann erfolgt der Satzübergang (Genau-Halt), dann erst wird eine neue Achsbewegung dann gestartet.
   Wird diese Funktion nicht gewünscht, so ist **apInPos** = **apFEMax** oder **apInPos** = **mpFEFatal** zu programmieren.
b) Überschreitet der Schleppfehler den Wert **apFEMax**, so wird das Bit **stFEMaxB** im **Status**-Register und das entsprechende Schleppfehler-Bit im Register **PIDInfo** GESETZT (Bit 15..8 = Achse 7..0).
   Eine weitere Reaktion erfolgt nicht, d.h. ggf. muß das PC-Programm geeignete Maßnahmen ergreifen!
c) Überschreitet der Schleppfehler den Wert **mpFEFatal** bzw. 32000 (numerische Grenzen des Einlesebausteins bzw. des implemtierten PID-Algorithmus), so wird das entsprechende Freigabe-Bit im Register **PIDInfo** GELÖSCHT (Bit 31..24 = Achse 7..0), der Freigabe-Ausgang zur Servo-Endstufe und der Regelalgoritmus selbst deaktiviert. Zusätzlich wird das **stFEB** im Statusregister gesetzt und die Interpolation abgebrochen.
   DIESER ZUSTAND KANN NUR DURCH EINEN Restart-BEFEHL (**icRestart..**) VERLASSEN WERDEN!

B.3.3 Ausgangsspannungsbegrenzung

Der Achsparameter **apAmpMax** begrenzt die maximale Ausgangsspannung.
Der Defaultwert 32000 entspricht ca. ±10V.

B.4 Referenzpuls-Einstellung

Zum allgemeinen Ablauf der Referenzfahrt wird auf das entsprechende Kapitel 5.3.1 verwiesen. Hier sollen nur Hinweise zur Positionierung auf einen Referenz-Impuls gegeben werden (Feinreferenz). Diese Einstellungen sind erforderlich, wenn im Verlauf der Referenzfahrt nach dem Positionieren auf den Endschalter nicht auf die Referenzposition z.B. eines Glasmaßstabs sondern auf den periodisch von einem Drehgeber (Resolver) erzeugten Nullimpuls (Refpuls, Resolverpuls, Indexpuls) positioniert werden soll.

Generell ist es dazu erforderlich, vor dem Positionieren auf diesen Resolverpuls sicherzustellen, daß die Achse ungefähr zwischen zweien dieser Impulse steht. Zunächst ist also die Lage des Endschalters relativ zum Resolverpuls zu bestimmen und bei der Referenzfahrt nach dem Freifahren vom Endschalter die Achse um eine geeignete Strecke so zu verfahren, daß diese Bedingung erfüllt ist. Dazu dient der Achsparameter **apHys**.
Das Freifahren, das Positionieren zwischen zwei Resolverpulse und das Positionieren auf den Resolverpuls sollte immer in Richtung "weg vom Endschalter" erfolgen (d.h. bei Referenzfahrt in positiver Richtung nach Minus bzw. bei Referenzfahrt in negativer Richtung nach Plus).

Die Einstellung der relevanten Achsparameter erfolgt in vier Schritten:

1. Programmierung der Parameter:
   a) **apHys** = 0
   b) apLS:
   - **lsPIn**, **lsMIn**, **lsPPol** und **lsMPol** entsprechend der angeschlossenen Endschalter
   - **lsPRef** und **lsMRef** = 0**!**
   - **lsPHDir** und **lsPRDir** = 1
   - **lsMHDir** und **lsMRDir** = 0

   c) apDC:
   - **dcRefIn** = 1
   - **dcRefPol** entsprechend der Polarität des Ref-Pulses

   d) Die Parameter für die Geschwindigkeiten **apVRef**, **apVHys** und **apVRefP** sollten sinnvoll gesetzt sein, ebenso **apHysMax**; die anderen Parameter, insbesondere **apRefPos**, sollten auf 0 gesetzt sein (am einfachsten beläßt man es bei den Default-Werten).

2. Mit diesen Einstellungen wird für die Achse eine Referenzfahrt in der gewünschten Richtung durchgeführt. Danach steht die Achse auf dem Referenz-Impuls vor dem Endschalter, die Ist-Position (Register **icGetPos**) wird auf 0 gesetzt.

3. Anschließend wird nun langsam bis zum Endschalter zurück positioniert (→ Endschalter-Fehlermeldung). Im Register **icGetPos** läßt sich nun der Abstand Endschalter-Refpuls direkt ablesen.

4. Für die Bestimmung des Parameter **apHys** muß nun der Abstand zwischen zwei Resolverpulsen (in µm) bekannt sein. Er ergibt sich aus der Strichzahl des eingesetzten Resolvers, ggf. der Getriebe-Übersetzung und der Steigung der Antriebsspindel. Normalerweise erscheint der Resolverpuls 1x pro Motor-Umdrehung. Die folgenden Formeln zeigen die Berechnung (alle Angaben in µm, ohne Vorzeichen: die Richtung ergibt sich aus **lsXHDir** bzw. **lsXRDir**):

   ER Abstand Endschalter-Refpuls
   U Abstand zwischen zwei Resolverpulsen

   apHys = ER - U/2 falls ER > U/2
   apHys = ER + U/2 falls ER < U/2

   Beispiel
   Getriebe: 3:2 (d.h. 3 Umdr. am Motor → 2 Umdr. an der Achse)
   Spindelsteigung: 5mm

   ==> U = 5000µm * 2 / 3 = 3333µm

Abstand Endschalter-Refpuls nach der Referenzfahrt wie oben dargestellt:

ER = 2300µm > U/2

==> **apHys** = 2300µm - 1666µm = 633µm

5. Zum Schluß können noch die übrigen Parameter entsprechend den Erfordernissen eingestellt werden. Insbesondere sollten erst jetzt ggf. die Parameter **lsPRef** bzw. **lsMRef** gesetzt werden, falls der entsprechende Endschalter nur als Grobreferenz und nicht als mechanischer Endschalter dienen soll.

Hinweis *zum Einstellen der Geschwindigkeit **apVRefP**: Die zeitliche Auflösung beim Suchen des Referenz-Impulses beträgt 1ms. Da der Resolver-Impuls eines Drehgebers normalerweise 1 Impuls breit ist, sollte die Geschwindigkeit geringer als 1 Impuls/ms sein.*

Beispiel: ***apSteps*** *= 4096*
apmm *= 5*

→ 1 Impuls/ms = 1.22 mm/ms = 73 mm/min
*d.h. es sollte **apVRef** < 73 mm/min programmiert werden*

Wird innerhalb einer Motor-Umdrehung der Refpuls nicht gefunden, so kann dies zwei Ursachen haben:

1. *Es liegt ein Verkabelungsfehler vor; möglicherweise wird der Referenzpuls separat mit Spannung versorgt und diese ist nicht mit GND_EXT verbunden (s.a. Anhang A).*
2. *Die Geschwindigkeit **apVRefP** ist zu groß. Sie sollte nur so groß sein, daß innerhalb eines Interpolations-Intervalls von 1ms der Puls sicher erkannt werden kann.*

Index

Abbildungsverzeichnis