HEIDENHAIN

Benutzer-Handbuch

IK 320

VMEbus-Zählerkarte

6/2004

Gültigkeit

Dieses Handbuch gilt für die IK 320 ab Software-Version

246 118-08

Inhaltsübersicht

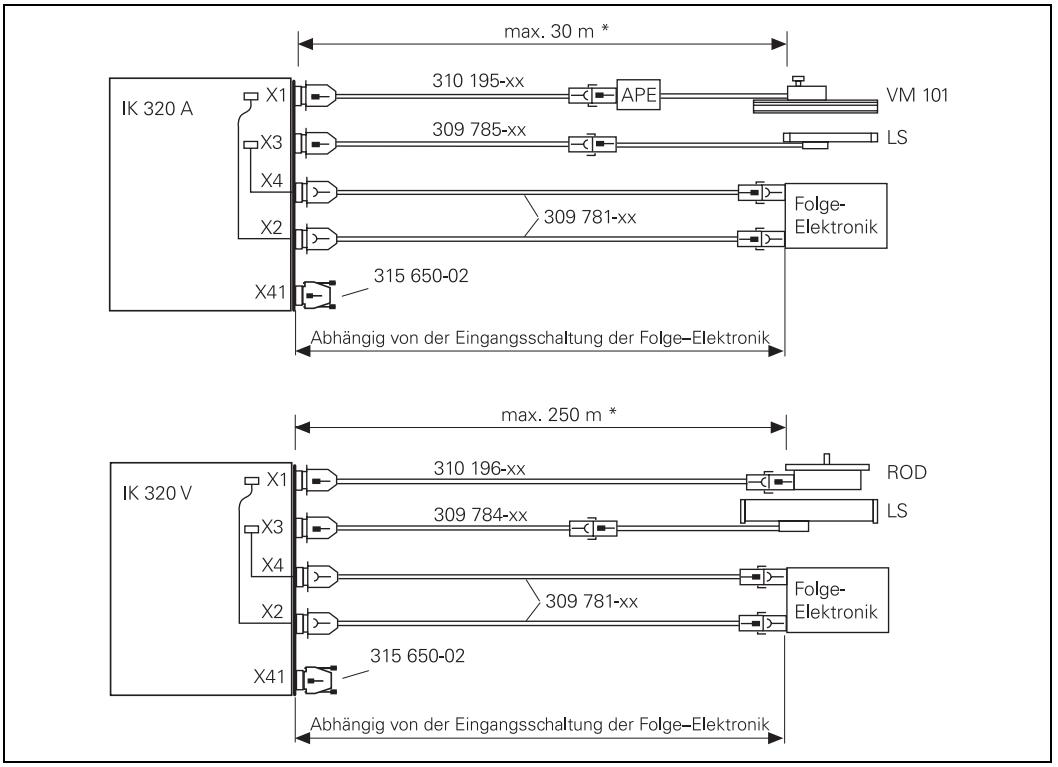
1. Lieferumfang

VMEbus-Zählerkarte IK 320, Treiber-Software und Benutzer-Handbuch

1.1 Ausführungen

IK 320 A	VMEbus-Zählerkarte mit Messsystem-Eingängen für sinusförmige
286 939 01	Stromsignale (11 μA_{SS}).
IK 320.1	**Sonderausführung:** VMEbus-Zählerkarte mit Messsystem-Eingängen für
286 839 11	sinusförmige Stromsignale (22 μA_{SS}).
IK 320 V	VMEbus-Zählerkarte mit Messsystem-Eingängen für sinusförmige
286 939 51	Spannungssignale (1 V_{SS}).

1.2 Zubehör



*Bei Versorgungsspannung 5,1 V und Stromaufnahme des Messsystems < 65 mA.

309 781-xx	Verbindungskabel vom Messsystem-Ausgang an eine weitere Anzeige oder Steuerung
315 650-02	Stecker für die externen Funktionen am Anschluss X41

IK 320 A/IK 320.1

309 785-xx 9/9polig	Kabeladapter mit Kupplung für HEIDENHAIN-Messsysteme; Standardlänge 0,5 m
310 195-xx 9/9polig	Kabeladapter mit Stecker für HEIDENHAIN-Messsysteme mit Flanschdose; Standardlänge 0,5 m

IK 320 V

309 784-xx 15/12polig	Kabeladapter mit Kupplung für HEIDENHAIN-Messsysteme; Standardlänge 0,5 m
310 196-xx 15/12polig	Kabeladapter mit Stecker für HEIDENHAIN-Messsysteme mit Flanschdose; Standardlänge 0,5 m

2. Wichtige Hinweise

Gefahr für interne Bauteile!
Die Vorsichtsmaßnahmen bei der Handhabung **elektrostatisch entladungsgefährdeter Bauelemente (ESD)** nach DIN EN 100 015 beachten. Als Transport-Verpackung nur antistatisches Material verwenden. Beim Einbau ausreichende Erdung des Arbeitsplatzes und der Person sicherstellen.

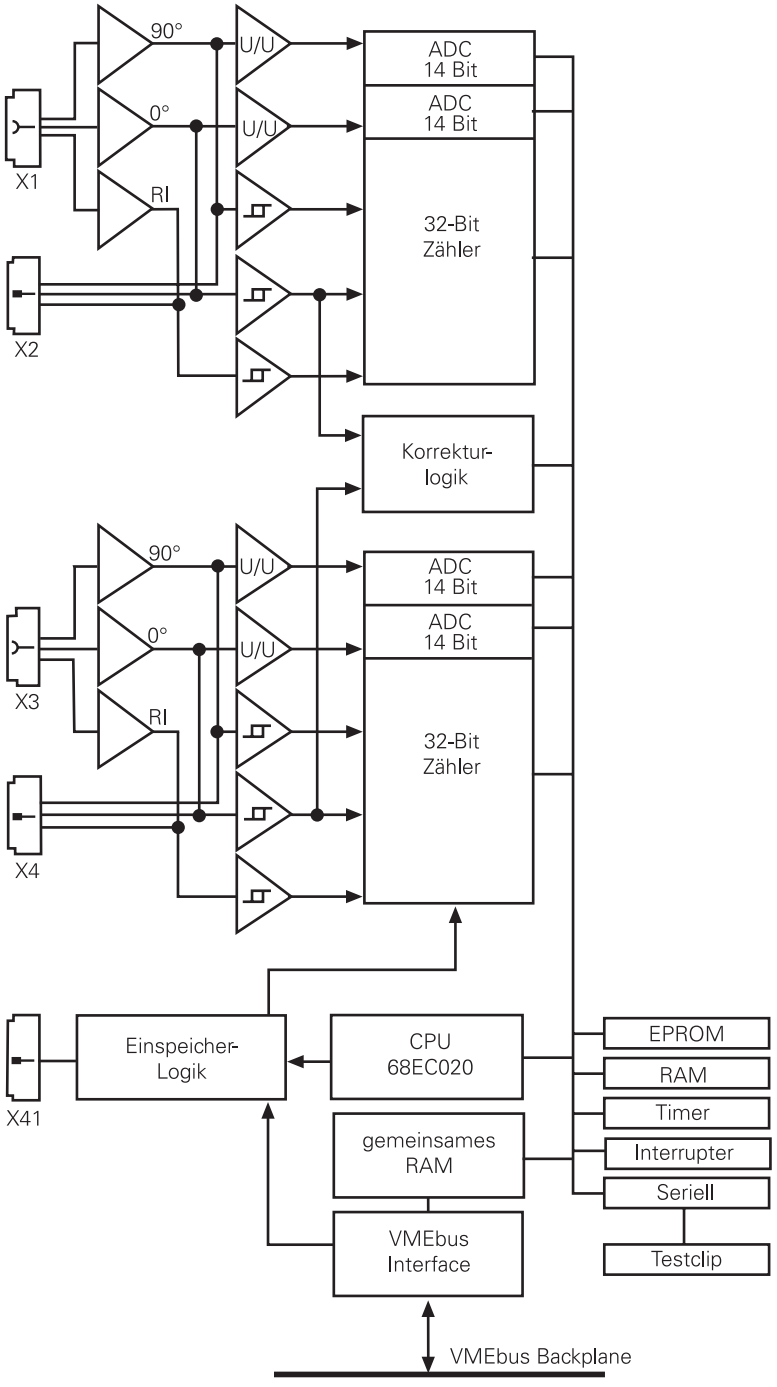
Einige Hinweise zu den verwendeten Begriffen:

- Zahlen in hexadezimaler Schreibweise werden mit $ gekennzeichnet, z.B. $FF.
- Invertierte Signale erhalten vor dem Signalnamen ein Minus-Zeichen, z.B. –IMPULS1.
- Die Messsystem-Eingänge werden als Achse 1 (entspricht Anschluss X1) und Achse 2 (entspricht Anschluss X3) bezeichnet.
- Der Begriff „**einspeichern**“ bedeutet, dass der Zählerwert im Daten-Register festgehalten wird. Dieser Zählerwert muss anschließend **abgerufen** werden, d.h. per Software gelesen und im Rechner gespeichert oder am Bildschirm angezeigt werden.
- Die Bedeutung der VMEbus-Signale und -Begriffe entnehmen Sie bitte der VMEbus-Fachliteratur.

3. Technische Beschreibung der IK 320

An die VMEbus-Zählerkarte IK 320 können zwei HEIDENHAIN-Messsysteme mit sinusförmigen Stromsignalen (IK 320 A) oder Spannungssignalen (IK 320 V) angeschlossen werden. Die Positionen der beiden Messsysteme werden mittels Software zum Rechner übertragen und weiterverarbeitet.
Die IK 320 ist ideal für Anwendungen, bei denen eine hohe Auflösung der Messsystem-Signale erforderlich ist.

Blockschaltbild:



Die Interpolations-Elektronik in der IK 320 unterteilt die Signalperiode des Eingangs-Signals 4096fach.
Der Interpolationswert (12 Bit) bildet zusammen mit dem Wert des Periodenzählers (32 Bit) den 44 Bit breiten Messwert. Die IK 320 speichert die Messwerte in 48 Bit breiten Daten-Registern, wobei die unteren 4 Bit nicht genutzt werden.
Die Messwerte können entweder über externe Eingänge, per Software oder durch das Überfahren der Referenzmarken eingespeichert und anschließend über den VMEbus zum Rechner übertragen werden.

Die Genauigkeit des Messergebnisses kann erhöht werden, wenn die Signalform der sinusförmigen Messsystem-Signale durch einen Korrekturlauf ermittelt wird. Die IK 320 analysiert die Signalform und speichert an bis zu 4096 Stützpunkten Korrekturwerte für die Signalform der analogen Signale.

Die Kommunikation zwischen dem Rechner (Master) und der IK 320 (Slave) erfolgt über Interrupts und ein gemeinsames RAM für Daten und Parameter (siehe „6. Daten, Parameter und Datenverkehr“).

3.1 Zugriffszeit auf Messwerte

Die Zugriffszeit auf die Messwerte beträgt ca. 400 µs pro Achse.

4. Hardware

4.1 VMEbus-Interface

Spezifikation:	ANSI/IEEE STD 1014-1987, IEC 821 and 297		
Größe:	double height board (233,4 mm x 160 mm x 20 mm), 1 Slot		
VME-Anschluss:	Stecker J1 (Steckerbelegung siehe VMEbus-Literatur)		
Interrupter:	D08(O) ROAK		
Adressraum A16:	Slave, D08(O) (Port)		
	Slave, ADO (synchron einspeichern)		
	Speicherbedarf: 16 kByte je Karte		
Adressraum A24:	Slave, D16, D08 (EO), 16 kByte		
Interruptleitungen:	7, über Jumper wählbar		
Spannungen:	+ 5 V :	+ 0,25 V / – 0,125 V	50mVss Noise
	+ 12 V:	+ 0,60V / – 0,36 V	20mVss Noise*
	– 12 V:	– 0,60 V / + 0,36 V	20mVss Noise*
	+ 5 V STDBY:	+ 0,25 V / – 1,70 V	

* nicht nach VME-Spezifikation

Die Störungen auf ± 12 V sind nicht nach VMEbus spezifiziert. Es muss darauf geachtet werden, daß Störungen auf ± 12 V möglichst niedrig gehalten werden. Störungen auf den Spannungsversorgungen beeinflussen die Messgenauigkeit.

Stromaufnahme (max):	+ 12 V	160 mA
	– 12 V	160 mA
	+ 5 V	1,2 A
	+ 5 V STDBY	100 µA

Über die Stand-By Spannung wird der Korrekturwertspeicher versorgt.

4.2 Kontroll-LED

Auf der Platine – in der Nähe des VMEbus-Steckers – befindet sich eine grüne Kontroll-LED. Die **Kontroll-LED** blinkt nach dem Einschalten mit einer Frequenz von ca. 1 Hz und zeigt damit an, dass die Karte arbeitet. Beim Abtasten der Messsystem-Signale – während der Korrekturwertaufnahme – blinkt die LED nicht (sie bleibt entweder ein- oder ausgeschaltet). Bei lokalen Fehlern auf der Karte blinkt sie mit etwa doppelter Frequenz. Diese Fehlermeldung kann nur durch Ausschalten des Systems oder durch einen RESET auf dem VMEbus gelöscht werden.

4.3 Messsystem-Eingänge IK 320 A

An die IK 320 A können HEIDENHAIN-Längenmesssysteme oder -Winkelmesssysteme mit sinusförmigen Stromsignalen I_1 und I_2 angeschlossen werden.

Signalamplituden:	I_1, I_2 (0°, 90°) I_0 (Referenzmarke)	7 μA_{SS} bis 16 μA_{SS} 3,5 µA bis 8 µA Nutzanteil
Signalpegel für Fehlermeldung I_1, I_2		≤ 2,5 μA_{SS}
Maximale Eingangsfrequenz		50 kHz
Kabellänge		max. 30 m; bei Versorgungsspannung 5,1 V und Stromaufnahme des Messsystems < 65 mA

Anschluss X1, X3 für Messsysteme
Sub-D-Anschluss mit Buchseneinsatz (9polig)

Anschluss-Nr.	Belegung
1	I_1 –
2	0 V (U_N)
3	I_2 –
4	Innenschirm
5	I_0 –
6	I_1 +
7	5 V (U_P)
8	I_2 +
9	I_0 +
Gehäuse	Außenschirm

4.4 Messsystem-Eingänge IK 320.1 (Sonderausführung)

Signalamplituden: I_1, I_2 (0°, 90°) I_0 (Referenzmarke)	14 μA_{SS} bis 30 μA_{SS} 10 µA bis 30 µA Nutzanteil
Signalpegel für Fehlermeldung I_1, I_2	≤ 6 μA_{SS}
Maximale Eingangsfrequenz	50 kHz
Kabellänge	max. 30 m

4.5 Messsystem-Eingänge IK 320 V

An die IK 320 V können Sie HEIDENHAIN-Längenmesssysteme oder -Winkelmesssysteme mit sinusförmigen Spannungssignalen A und B anschließen.

Signalamplituden: A, B (0°, 90°) R (Referenzmarke)	0,6 V_{SS} bis 1,2 V_{SS} 0,2 V bis 0,85 V Nutzanteil
Signalpegel für Fehlermeldung A, B	≤ 0,22 V_{SS}
Maximale Eingangsfrequenz	50 kHz
Kabellänge	max. 250 m: bei Versorgungsspannung 5,1 V und Stromaufnahme des Messsystems < 65 mA

Anschluss X1, X3 für Messsysteme
Sub-D-Anschluss mit Buchseneinsatz (15polig)

Anschluss-Nr.	Belegung
1	A +
2	0 V (U_N)
3	B +
4	+ 5 V (U_P)
5	nicht belegen
6	nicht belegen
7	R –
8	nicht belegen
9	A –
10	0 V (Fühlleitung)
11	B –
12	+ 5 V (Fühlleitung)
13	nicht belegen
14	R +
15	nicht belegen
Gehäuse	Außenschirm

4.6 Messsystem-Ausgänge

Die IK 320 gibt die Messsystem-Signale der Eingänge 1 (Anschluss X1) und 2 (Anschluss X3) zusätzlich an zwei 9polige Sub-D-Anschlüsse als **sinusförmige Stromsignale** (11 μA_{SS}) aus. Diese Ausgänge dürfen nur an Folge-Elektroniken angeschlossen werden, deren Eingangsverstärker mit ± 12 V versorgt werden, andere Eingangsverstärker werden übersteuert (U_0 der IK 320 liegt auf 0 V). Adapter-Kabel (Id.-Nr. 309 781-01) zum Anschluss an HEIDENHAIN-Positionsanzeigen sind lieferbar (siehe „1.2 Zubehör"). **Keine Interpolations-Elektroniken EXE anschließen!**

Messsystem-Ausgänge X2, X4
Sub-D-Anschluss mit Stifteinsatz (9polig)

Anschluss-Nr.	Belegung
1	I_1 –
2	0 V (U_N)
3	I_2 –
4	nicht angeschlossen
5	I_0 –
6	I_1 +
7	nicht angeschlossen
8	I_2 +
9	I_0 +
Gehäuse	Außenschirm

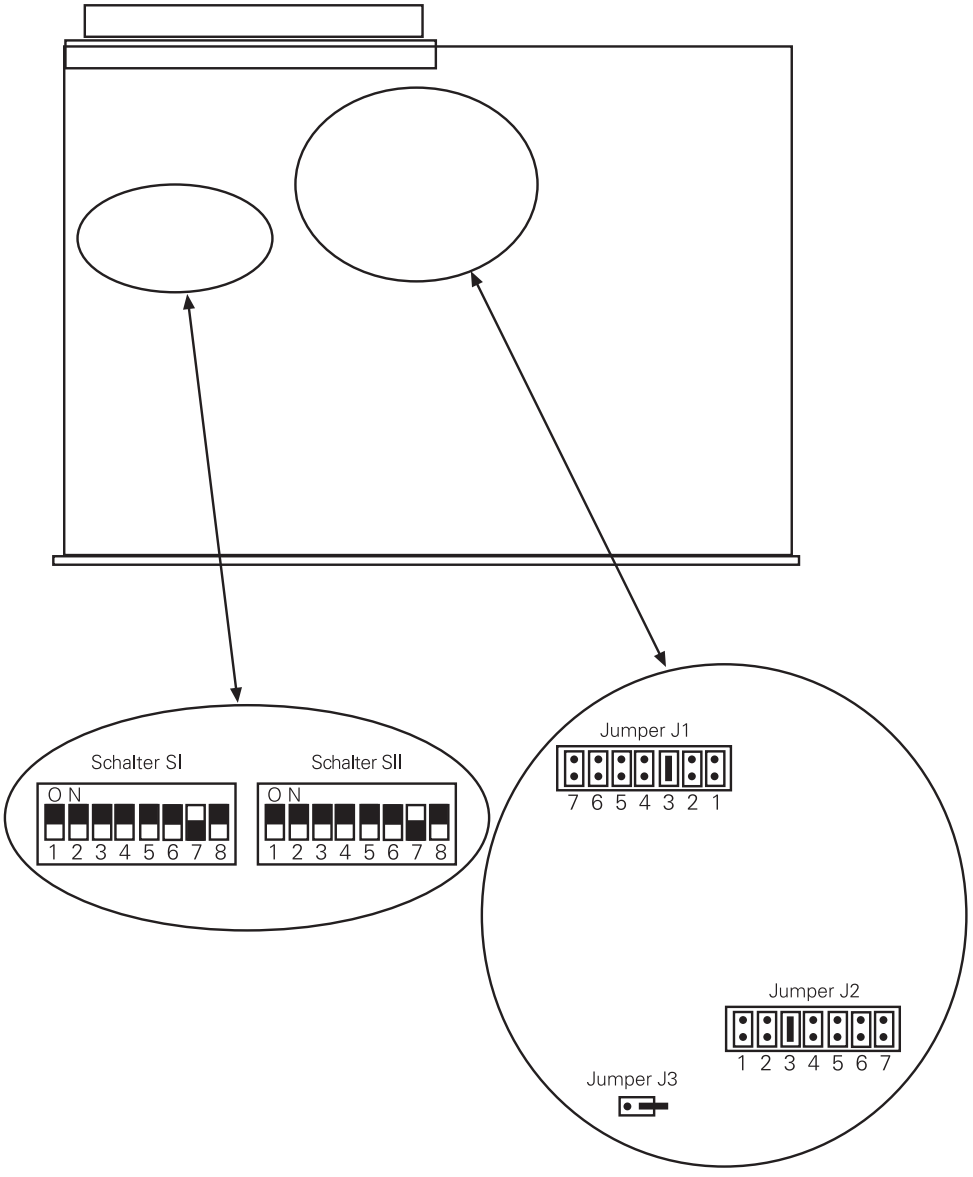
4.7 Externe Funktionen

Zum externen Einspeichern der Messwerte und zum synchronen Einspeichern mehrerer Karten ist ein 9poliger Sub-D-Anschluss vorhanden. Der dafür benötigte Stecker (Id.-Nr. 315 650-02) kann bei HEIDENHAIN bestellt werden. Weitere Beschreibung dieses Anschlusses siehe „8.1 Positionswert einspeichern".

5. Adressierung

5.1 Schalter und Jumper

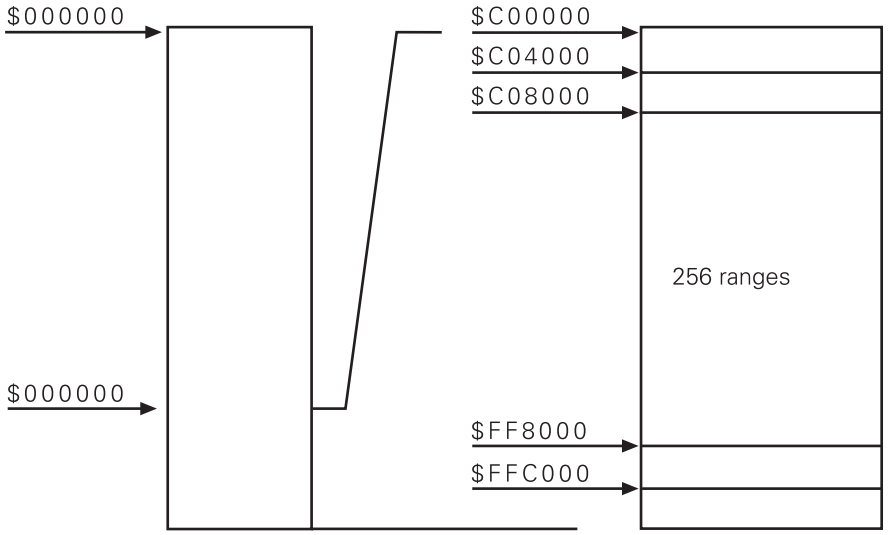
Auf der IK 320 befinden sich zwei DIP-Schalter und drei Steckbrücken. Der DIP-Schalter SI legt die Adresse des VME-Adressraums A16 und der Schalter SII die des VME-Adressraums A24 fest. Die Steckbrücken J1 und J2 bestimmen die Interrupt-Request- und Interrupt-Acknowledge-Leitung. Die Steckbrücke J3 legt das Verhalten der IK 320 bei einem „ADDRESS-ONLY-Zyklus" (ADO-Zyklus) fest.



5.2 VME-Adressraum A24 (Address Modifier AM: $39)

Der Datenaustausch zwischen Master und IK 320 erfolgt über ein gemeinsames 16 kByte großes RAM (im VME-Adressraum A24) auf der IK 320.
Mit dem auf der Karte vorhandenen 8poligen DIP-Schalter SII stellen Sie im oberen Viertel des Adressraums A24 (Bereich $C00000 - $FFFFFF) die IK-Adresse ein.
Die Basisadresse der Karte wird wie folgt berechnet:
Basisadresse (BA) = $C00000 + (DIP-Schalter SII*$4000)

Adress space 24



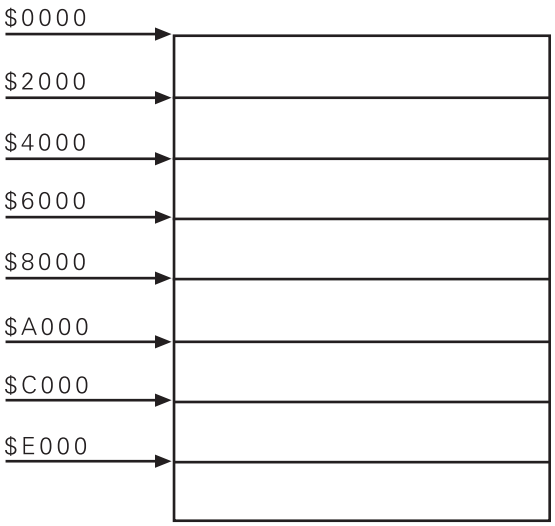
5.3 VME-Adressraum A16 (Address Modifier AM: $29)

Im Adressraum A16 stellt die IK 320 zwei Funktionen bereit:

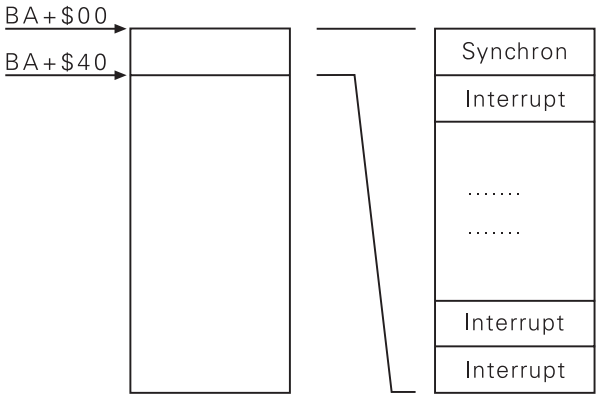
- Lokaler Interrupt auf der IK 320.
- Einspeicher-Signal (Synchron-Einspeichern) und lokaler Interrupt. Ein Einspeicher-Signal kann gleichzeitig auf mehrere Karten wirken.

Der Adressraum A16 wird durch S8, S7 und S6 (DIP-Schalter SI in acht Bereiche (entspricht Gruppen) geteilt.

Adressraum A16



Für alle Karten mit der gleichen Gruppeneinstellung (DIP-Schalter-SI: S8, S7, S6) kann der Master ein **gemeinsames** Einspeicher-Signal erzeugen. Dazu muss der Master auf Adresse $00 innerhalb dieses Bereichs einen ADRESS-ONLY-Zyklus durchführen (der ADRESS-ONLY-Zyklus liefert kein –DTACK zurück). Falls ein ADO-Zyklus nicht möglich ist: siehe „8.1.5 Direktes Einspeichern per VMEbus innerhalb einer Gruppe“.
Innerhalb einer Gruppe werden die Karten durch die Schalter S5 bis S1 von DIP-Schalter SI unterschieden. Die Adressen zur Interrupt-Generierung befinden sich in den untersten 32 Worten innerhalb des 8-kByte-Bereichs. Ein Schreiben auf eine eingestellte Interrupt-Adresse erzeugt einen lokalen Interrupt.



Mit S1 bis S5 von DIP-SCHALTER SI darf nicht Adresse 0 eingestellt werden.

Die eingestellten Adressen werden wie folgt berechnet:

Gruppenadresse = (DIP-Schalter SI: S8, S7, S6) * $2000
Interruptadresse = Gruppenadresse + (DIP-Schalter SI: S5-S1)* 2

5.4 Kommunikation über Interrupt

5.4.1 Von der IK 320 zum Master

Die IK 320 generiert für Nachrichten an den Master einen Interupt auf dem VMEbus. Damit die IK 320 einen Interrupt an den Master ausgeben kann, muss der Master das IK-Interrupt-Status-Register gelöscht haben. Erst dann kann die IK 320 die Nachricht im IK-Interrupt-Status-Register (16bit-Wort) ablegen. Mit dem Jumper J1 wählt man die Interrupt-Request-Leitung (mögliche Leitungen: –IRQ1 bis –IRQ7). Die IK 320 speichert einen Interrupt bis der Master einen Interrupt-Acknowledge-Zyklus durchführt. Wird ein gültiger IACK-Zyklus erkannt, so gibt die IK 320 die Vektor-Nummer (= Schalterstellung des DIP-Schalters SI) als Kennung der Karte aus und löscht den Interrupt. In der Interrupt-Service-Routine kann der Master das IK-Interrupt-Status-Register der IK 320 lesen. Zum Schluss muss der Master das IK-Interrupt-Status-Register löschen, um der IK 320 die Möglichkeit zu geben, neue Interrupts auszugeben.

Jumper 1: Interrupt-Request-Leitung

1	-IRQ1
2	-IRQ2
3	-IRQ3*
4	-IRQ4
5	-IRQ5
6	-IRQ6
7	-IRQ7

* Werkseitig bestückt

Anmerkung: keine Steckbrücke = Interrupt gesperrt.

Jumper 2: Interrupt-Acknowledge-Leitung

1	-INT1
2	-INT2
3	-INT3 *
4	-INT4
5	-INT5
6	-INT6
7	-INT7

* Werkseitig bestückt

Anmerkung: Die Interrupt-Acknowledge-Leitung muss der Interrupt-Request Leitung entsprechen.

5.4.2 Vom Master zur IK 320

Der Master schreibt die gewünschte Funktionsnummer in Parameter P81. Danach löst der Master durch Beschreiben der Interrupt-Adresse im Adressraum A16 auf der IK 320 einen Interrupt aus (ADO-Zyklus). Die IK 320 erkennt den Interrupt, verzweigt entsprechend der eingegebenen Funktionsnummer und führt die Funktion aus.

Jumper 3:

Legt fest, ob die Karte bei einem ADO einen DTACK-Impuls generiert oder nicht (siehe auch „8.1.5 Direktes Einspeichern per VMEbus innerhalb einer Gruppe").

Jumper offen: kein DATCK

Jumper geschlossen: DTACK wird generiert

5.5 Beispiel für Schaltereinstellungen

Ein Gesamtsystem beinhaltet insgesamt fünf IK 320 aufgeteilt auf drei Gruppen:

Gruppe 1: 2 * IK 320
Gruppe 2: 2 * IK 320
Gruppe 3: 1 * IK 320

DIP-Schalter SI:

Gruppe	Nr.	Schalter S8 – S1	Adressraum A16	Interrupt	Gemeinsames Einspeichern
Gruppe 1	1	1010.0001	$A000-$BFFF	$A002	$A000
	2	1010.0010	$A000-$BFFF	$A004	$A000
Gruppe 2	3	1100.0001	$C000-$DFFF	$C002	$C000
	4	1100.0010	$C000-$DFFF	$C004	$C000
Gruppe 3	5	1110.0001	$E000-$FFFF	$E002	$E000

DIP-Schalter SII:

Gruppe	Nr.	Schalter S8 – S1	Adressraum A24
Gruppe 1	1	1000.0000	$E00000-$E03FFF
	2	1000.0001	$E04000-$E07FFF
Gruppe 2	3	1100.0001	$F04000-$F07FFF
	4	1100.0010	$F08000-$F0BFFF
Gruppe 3	5	1110.0001	$F84000-$F87FFF

Hinweis:

Tabelleneintrag „1" bedeutet Schalterstellung „OFF" ,für die Adressbildung heißt Schalterstellung „OFF" die Wertigkeit dieses Adressbits = 1 (logischer Pegel = 5V).

6. Daten, Parameter und Datentransfer

Der Datenaustausch zwischen Master und IK 320 erfolgt über ein gemeinsames RAM. Zur Koordination werden Interrupts und Merker verwendet. Der gemeinsame RAM-Bereich wird in die beiden Bereiche **Daten** und **Parameter** unterteilt.

6.1 Daten

Der Zugriff auf den Datenbereich wird über Transfermerker koordiniert: Wenn der entsprechende Transfermerker = **$00** ist, darf die IK 320 schreiben; ist er **$01** ist, darf der Master lesen. Die Transfermerker werden in POST (**P**ower **O**n **S**elf **T**est) gelöscht. Findet die IK 320 beim Schreiben einen gesetzten Merker vor, so wird eine Fehlermeldung an den Master ausgegeben und mit dem Schreiben so lange gewartet, bis der Merker vom Master gelöscht wird.

BA+$00: Positionswert Achse 1
Größe: 6 Bytes, Motorola-Format (High Byte first)
Übertragen wird eine 48-Bit-Zahl, wobei die oberen 32 Bit den Zählerwert und die unteren 16 Bit den Interpolationswert enthalten. Da der Interpolationswert nur 12 Bit benötigt, werden die untersten 4 Bit nicht genutzt.

BA+$06: Status Achse 1
Größe: 1 Byte

Bit	Inhalt = 0	Inhalt = 1
0	keine Signalkorrektur	Signalkorrektur
1	reserviert	
2	gestoppt	gestartet
3	OK	Signalamplitude zu klein
4	OK	Frequenzüberschreitung
5	–	warten auf REF
6	reserviert	
7	–	Berechnung Korrektur läuft

Der Status wird mit jedem Messwertabruf aktualisiert.

Bedeutung der einzelnen Bits:

Bit0: 0: Beim letzten Messwertabruf wurde keine Signalkorrektur durchgeführt, da:
- die Position außerhalb des korrigierten Bereichs liegt
- P06 dieser Achse nicht eingeschaltet ist
- keine gültigen Korrekturwerte im Speicher sind
- die Achse noch nicht über die Referenzmarke gefahren ist (siehe Kapitel 7.3)

1: Beim letzten Messwertabruf wurde eine Signalkorrektur durchgeführt.

Bit2: 0: Die Achse ist gestoppt, sie hat die Referenzmarke(n) noch nicht überfahren.
1: Die Achse ist gestartet; das Bit wird nach Aufruf der Funktion Referenzpunktfahren (Funktionsnummer 08) und dem Überfahren der Referenzmarke(n) gesetzt.

Bit3: 0: Die Signalamplitude war beim letzten Messwert-Abruf in Ordnung.
1: Die Signalamplitude war beim letzten Messwert-Abruf zu klein.

Bit4: 0: Es ist keine Frequenzüberschreitung des Messsystemsignals aufgetreten.
1: Es ist eine Frequenzüberschreitung des Messsystemsignals aufgetreten.

Bit5: 0: Die Achse ist nicht im Status „Warten auf Referenzmarke".
1: Die Achse ist im Status „Warten auf Referenzmarke"; das Bit wird nach Funktionsaufruf 08 gesetzt und beim Überfahren der Referenzmarke gelöscht.

Bit7: Das Bit ist nur während der Korrekturwertaufnahme von Bedeutung.
0: Die Berechnung bei der Korrekturwertaufnahme läuft noch nicht oder ist abgeschlossen.
1: Die Berechnung bei der Korrekturwertaufnahme läuft (siehe Kapitel 7.4).

BA+$07: Transfermerker Achse 1
Größe: 1 Byte
Transfermerker = $00: IK 320 darf schreiben
Transfermerker = $01: Master darf lesen

BA+$08: Positionswert Achse 2
Größe: 6 Bytes, Motorola-Format (High Byte first).
Übertragen wird eine 48-Bit-Zahl, wobei die oberen 32 Bit den Zählerwert und die unteren 16 Bit den Interpolationswert enthalten. Da der Interpolationswert nur 12 Bit benötigt, werden die untersten 4 Bit nicht genutzt.

BA+$0E: Status Achse 2
Größe: 1 Byte

Bit	Inhalt = 0	Inhalt = 1
0	keine Signalkorrektur	Signalkorrektur
1	reserviert	
2	gestoppt	gestartet
3	OK	Signalamplitude zu klein
4	OK	Frequenzüberschreitung
5	–	warten auf REF
6	reserviert	
7	–	Berechnung Korrektur läuft

Der Status wird mit jedem Messwertabruf aktualisiert.

Bedeutung der einzelnen Bits: siehe Beschreibung **BA+$06: Status Achse 1**

BA+$0F: Transfermerker Achse 2
Größe: 1 Byte
Transfermerker = $00: IK 320 darf schreiben
Transfermerker = $01: Master darf lesen

BA+$10: gemeinsamer Positionswert der verknüpften Achsen 1 und 2
Größe: 6 Bytes, Motorola-Format (High Byte first)
Übertragen wird eine 48-Bit-Zahl, wobei die oberen 32 Bit den Zählerwert und die unteren 16 Bit den Interpolationswert enthalten. Da der Interpolationswert nur 12 Bit benötigt, werden die untersten 4 Bit nicht genutzt.

BA+$16: Status für gemeinsamen Positionswert der verknüpften Achsen 1 und 2
Größe: 1 Byte
Reserviert, nicht benützt

BA+$17: Transfermerker für gemeinsamen Positionswert der verknüpften Achsen 1 und 2
Größe: 1 Byte

BA+$18: IK-Interrupt-Status
Größe: Wort
Wenn von der IK 320 an den Master ein Interrupt ausgegeben wird, steht im Interrupt-Status-Wort die Interrupt-Ursache. In der Interrupt-Service-Routine muss der Master das Statuswort lesen und anschließend löschen. Erst danach kann die IK 320 einen neuen Interrupt ausgeben.

$0000	nicht initialisierter Interrupt
$01xx	Positionswert X1 wurde geschrieben: Lower Byte Anforderungsquelle: xx = $01: –IK1, externer Einspeicherimpuls $03: Synchron-Einspeicher per VMEbus $04: –F1, externer Funktionseingang $05: –F2, externer Funktionseingang $06: Interrupt Master (Einspeichern per Software, Funktionsnr. 1)
$02xx	Positionswert X2 wurde geschrieben: Lower Byte Anforderungsquelle: xx = $02: –IK2, externer Einspeicherimpuls $03: Synchron-Einspeicher per VMEbus $04: –F1, externer Funktionseingang $05: –F2, externer Funktionseingang $06: Interrupt Master (Einspeichern per Software, Funktionsnr. 2)
$03xx	Verknüpfter Positionswert X1 und X2 wurde geschrieben: Lower Byte = Anforderungsquelle xx = $01: –IK1, externer Einspeicherimpuls $03: Synchron-Einspeicher per VMEbus $04: –F1, externer Funktionseingang $05: –F2, externer Funktionseingang $06: Interrupt Master (Einspeichern per Software, Funktionsnr. 3)
$04xx	Fehler beim Einspeichern: kein Latch-Signal: Lower Byte = Achse xx = $01: Achse 1 $02: Achse 2
$05xx	Fehler beim Einspeichern: doppeltes Einspeichern: Lower Byte = Achse xx = $01: Achse 1 $02: Achse 2
$06xx	Positionswert kann nicht geschrieben werden, da Transfermerker gesetzt Lower Byte = Achse xx = $01: Achse 1 $02: Achse 2 $03: verknüpfte Achsen 1 und 2
$07xx	POST (**P**ower **O**n **S**elf **T**est) abgeschlossen, Lower Byte: Fehlerstatus xx = $00: kein Fehler Bit 0: CRC-Fehler Korrekturwerte 1 Bit 1: CRC-Fehler Korrekturwerte 2 Bit 2: Parameter Fehler (siehe Parameter setzen) Bit 3: CRC-Fehler Eprom Bit 4: Prüfsummenfehler Eprom 1 Bit 5: Prüfsummenfehler Eprom 2 Bit 6: Hardware-Fehler
$08xx	Referenzmarke Achse 1 wurde überfahren xx = $00: kein Fehler $01: abstandscodierte Referenzmarken: falscher Abstand $02: Fehler im Zählerbaustein: entweder: eine weitere Referenzmarke überfahren und dann erneut die

Funktion „Referenzpunktfahren“ aufrufen
oder: die IK 320 zurücksetzen (RESET)
Dieser Fehler tritt nur auf, wenn das Referenzpunktfahren bei abstandscodierten Referenzmarken nach dem Überfahren der 1. Referenzmarke abgebrochen bzw. neu gestartet wird.
$03: Abbruch durch Anwender

$09xx Referenzmarke Achse 2 wurde überfahren
xx =
$00: kein Fehler
$01: abstandscodierte Referenzmarken: falscher Abstand
$02: Fehler im Zählerbaustein:
entweder: eine weitere Referenzmarke überfahren und dann erneut die Funktion „Referenzpunktfahren“ aufrufen
oder: die IK 320 zurücksetzen (RESET)
Dieser Fehler tritt nur auf, wenn Sie das Referenzpunktfahren bei abstandscodierten Referenzmarken nach dem Überfahren der 1. Referenzmarke abgebrochen bzw. neu gestartet wird.
$03: Abbruch durch Anwender

$0Axx Parameter wurden aktualisiert
xx =
$00: Parameter wurden unverändert übernommen
$01: Parameter enthielten Fehler, fehlerhafte Parameter werden mit Default-Werten überschrieben, die auch ins gemeinsame RAM geschrieben werden. Die Nummer des fehlerhaften Parameters ist im VME-RAM unter Adresse BA+0xEA zu ersehen (siehe Seite 22).
$02: Fehler Parameter für Winkelachse, P05 (Signalperoden) = 0 oder falscher Wert für P04 (REF-Abstand)

$0Bxx Korrekturwertaufnahme Achse 1 beendet. Lower Byte = Meldung
xx =
$00: OK
$01: Achse hat keinen Absolut-Bezug (kein REF)
$02: Geschwindigkeitsfehler
$03: falsche Position (nur bei Linearachse)
$04: falsche Verfahrrichtung
$05: Fehler innerhalb Teilungsperiode, evtl. Antrieb ungleichmäßig
$06: Berechnungsfehler bei Korrekturwertaufnahme
$07: Überschreitung des internen Speicherbereichs
$08: falsche Anzahl Messinterrupts, evtl. Antrieb ungleichmäßig
$09: Korrekturwertaufnahme vom Anwender abgebrochen
$10: Messwertaufnahme beendet, Achsantrieb kann gestoppt werden

$0Cxx Korrekturwertaufnahme Achse 2 beendet. Lower Byte = Meldung
xx =
$00: OK
$01: Achse hat keinen Absolut-Bezug (kein REF)
$02: Geschwindigkeitsfehler
$03: falsche Position (nur bei Linearachse)
$04: falsche Verfahrrichtung
$05: Fehler innerhalb Teilungsperiode, evtl. Antrieb ungleichmäßig
$06: Berechnungsfehler bei Korrekturwertaufnahme
$07: Überschreitung des internen Speicherbereichs
$08: falsche Anzahl Messinterrupts, evtl. Antrieb ungleichmäßig
$09: Korrekturwertaufnahme vom Anwender abgebrochen
$10: Messwertaufnahme beendet, Achsantrieb kann gestoppt werden

$0Exx Fehler beim Hintergrundtest für die Referenzmarken: falscher Abstand
xx =
$01: Achse 1

$02: Achse2
An jeder abstandcodierten Referenzmarke wird im Hintergrund der zugehörige Abstand geprüft.

$0Fxx CRC und Prüfsummenfehler. Lower Byte Fehlerkennung
xx =
$01: Eprom
$02: Korrekturwerte 1
$03: Korrekturwerte 2

$10xx Preset über VMEbus gesetzt. Lower Byte: Achse
xx =
$01: Achse 1
$02: Achse 2
$03: verknüpfte Achsen 1 und 2

$11xx Preset über externe Funktion gesetzt
xx =
Lower Byte, **Lower** Nibble: Achse
$.1: Achse 1
$.2: Achse 2
$.3: verknüpfte Achsen 1 und 2
Lower Byte, **Higher** Nibble: Quelle
$0.: Funktion 1
$1.: Funktion 2

$19xx Achse 1 Start ohne REF
xx =
$01: Achse nicht gestartet (evtl. Hardware defekt)
$02: 2. RI bleibt stehen[1)]

$1Axx Achse 2 Start ohne REF
xx =
$01: Achse nicht gestartet (evtl. Hardware defekt)
$02: 2. RI bleibt stehen[1)]

$1Bxx Achse 1 + 2 Start ohne REF
xx =
Bit0 = 1: Achse 1 nicht gestartet (evtl. Hardware defekt)
Bit1 = 1: Achse 1, 2. RI bleibt stehen[1)]
Bit4 = 1: Achse 2 nicht gestartet (evtl. Hardware defekt)
Bit5 = 1: Achse 2, 2. RI bleibt stehen[1)]
Bit2, 3, 6, 7 ohne Bedeutung

[1)] Dieser Zustand der IK 320 kann auftreten, wenn mit abstandscodierten REF-Marken Referenz gefahren wird und nach der 1.REF-Marke Referenzfahren abgebrochen wird. Fehlerbehebung durch das Überfahren einer weiteren REF-Marke bzw. ein RESET auf die IK 320.

$20xx Korrekturwerte Achse 1 gelesen
xx =
$00: 0K
$01: keine gültigen Korrekturwerte im Speicher

	$02: falsche Stützpunktnummer
$21xx	Korrekturwerte Achse 1 geschrieben xx = $00: 0K $01: BCC-Fehler bei Übertragung $02: falsche Stützpunktnummer $03: falsche Achse
$22xx	Korrekturwerte Achse 2 gelesen xx = $00: 0K $01: keine gültigen Korrekturwerte im Speicher $02: falsche Stützpunktnummer
$23xx	Korrekturwerte Achse 2 geschrieben xx = $00: 0K $01: BCC-Fehler bei Übertragung $02: falsche Stützpunktnummer $03: falsche Achse
$FDxx	interner Stack-Error xx = $01: Stack Hauptprogramm falsch $02: Speicherbereich für Stack wird überschritten
$FExx	Hardwarefehler (nicht benutzte CPU-Exception ist aufgetreten)
$FFxx	Befehl nicht erkannt. Lower Byte: Befehl (Aufruf einer nicht vorhandenen Funktionsnummer)

Hinweis:
Sollte sich die Karte mit dem Interrupt-Status $FDxx oder $FExx melden, so liegt ein interner Fehler der Karte vor. Die IK 320 führt einen lokalen Software-RESET durch. Anschließend muss die Karte wieder neu initialisiert werden (Parameter, POST, REF fahren).

BA+$1A: CRC-Summe Eprom SOLL
Größe: Wort
Wird in POST gesetzt.

BA+$1C: CRC-Summe Eprom IST
Größe: Wort
Wird in POST gesetzt.

BA+$1E: Prüfsumme Eprom 1 SOLL
Größe: Langwort
Wird in POST gesetzt.

BA+$22: Prüfsumme Eprom 1 IST
Größe: Langwort
Wird in POST gesetzt.

BA+$26: Prüfsumme Eprom 2 SOLL
Größe: Langwort
Wird in POST gesetzt.

BA+$2A: Prüfsumme Eprom 2 IST
Größe: Langwort
Wird in POST gesetzt.

BA+$2E: Hardware-Version
Größe: Wort
Wird in POST gesetzt.

BA+$30: Software-Version
Größe: 12 Byte, String
Wird in POST gesetzt.

BA+$3C: System-Status 1
Größe: Byte
Reserviert, wird nur intern benötigt.

BA+$3D: System-Status 2
Größe: Byte
Reserviert, wird nur intern benötigt.

BA+$3E: aktueller Preset Achse 1
Größe: 3 Worte

BA+$44: aktueller Preset Achse 2
Größe: 3 Worte

BA+$4A: aktueller Preset verknüpfte Achsen 1 und 2
Größe: 3 Worte

Hinweis:
Der aktuelle Preset ist der Wert, der beim PRESET-Setzen (extern oder über VMEbus) errechnet wurde und bei der Positionswertbestimmung (siehe „8.2 Positionswert berechnen“) verrechnet wird.

BA+$50 – BA+$6F: Datenbereich zum Korrekturwertlesen
Größe: 16 Worte
Wort 0: Stützpunkt-Nummer
Wort 1 bis 8: Korrekturwerte
Wort 9: BCC-Summe über Wort 0-8
Wort 10 bis 15: reserviert

BA+$70 – BA+$8F: Datenbereich zum Schreiben
Größe: 16 Worte
Wort 0: Stützpunkt-Nummer
Wort 1 bis 8: Korrekturwerte
Wort 9: BCC-Summe über Wort 0-8
Wort 10 bis 15: reserviert

BA+$90: CRC-Summe Korrekturdaten Achse 1
Größe: Wort

BA+$92: CRC-Summe Korrekturdaten Achse 2
Größe: Wort

BA+$94 – BA+$E9: freier Speicherbereich

BA+$EA: Nummer fehlerhafte Parameter
Größe: Wort
Meldet POST bzw. die Funktion Parameter übernehmen einen fehlerhaften Wert, so kann man in dieser Speicherstelle die Nummer des fehlerhaften Parameters entnehmen. Zusätzlich zeigen folgende Nummern folgende Fehler an:
100: Ref-Abstand (P04.x) ist grösser als Signalperioden (P05.x)

101: Winkelachse (P02.x > 1) und Signalperioden = 0 (P05.x)
102: Abstanscodierte REF-Marken (P04.x ungleich 0) und P19 ungleich 0

BA+$EC (32bit): Signalperiodenzähler X1
Größe: Langwort

BA+$F0: Trigger-Status X1
Größe: Wort
Der Triggerstatus zeigt den Pegel des 90°-Triggersignales zum Einspeicherzeitpunkt.
0x0000: Pegel ist low
0x0001: Pegel ist high

BA+$F2 (32bit): Signalperiodenzähler X2
Größe: Langwort

BA+$F6: Trigger-Status X2
Größe: Wort
Beschreibung s.o.

BA+$F8: Messsystem-Eingang Achse 1: Analogwert 0-Grad-Signal
Größe: Wort

BA+$FA: Messsystem-Eingang Achse 1: Analogwert 90-Grad-Signal
Größe: Wort

BA+$FC: Messsystem-Eingang Achse 2: Analogwert 0-Grad-Signal
Größe: Wort

BA+$FE: Messsystem-Eingang Achse 2: Analogwert 90-Grad-Signal
Größe: Wort

Hinweis:
Um den Spitze-Spitze-Wert von Messsystemstrom bzw. Messsystemspannung zu bestimmen, müssen Minimal- und Maximalwert erfasst werden. Mit den Eingangs-Verstärkungsfaktoren der IK 320 lassen sich dann die Spitze-Spitze-Werte berechnen.
Verstärkung der Messsystemeingangs-Signale:
IK 320 A: 540 mV/µA
IK 320.1: 301 mV/µA
IK 320 V: Spannungs-Verstärkungs-Faktor = 5,84
Die Analogwerte werden nur bei einem Messwertabruf neu beschrieben.
Format: 14 Bit, signed, linksbündig, Bit0 und Bit1 immer 0
Wertigkeit: Ein Inkrement entspricht 0,61 mV
Beispiel: $4000 entspricht 4096 x 0,61 mV = 2,499 Volt

6.2 Parameter

Der Zugriff auf den Parameterbereich wird über Interrupt koordiniert: Im allgemeinen darf der Master immer auf den Parameterbereich schreiben. Die IK 320 liest den Bereich jedoch nur, wenn sie dazu aufgefordert wird. Von diesem Zeitpunkt bis zur Rückmeldung ist das Schreiben des Parameterbereichs für den Master verboten. Eine Sonderstellung hat der Parameter P81 (Master-Interrupt-Funktionen). Er wird von der IK 320 **immer** nach einem Master-Interrupt gelesen.

P01.1: Zählrichtung Achse 1
P01.2: Zählrichtung Achse 2
P01.3: Parameter ist ohne Bedeutung
Adresse: BA+$102, BA+$103, BA+$104
Gültige Werte: 0, 1
Größe: Byte
0: normal, 1: invers
Bei Einstellung **invers** wird der auszugebende Positionswert invertiert.

P02.1: Achsdefinition Achse 1
P02.2: Achsdefinition Achse 2
P02.3: Achsdefinition verknüpfte Achsen 1 und 2
Adresse: BA+$106, BA+$107, BA+$108
Gültige Werte: 1, 2, 3, 4
Größe: Byte
1: linear, 2: Winkel 0° ... 360°, 3: Winkel ± ∞, 4: Winkel ± 180°
Bei Achsdefinition **Winkel** wird die Positionsberechnung auf die Position innerhalb einer Umdrehung reduziert, ebenso die Zuordung der Korrekturwerte.

P03: Anzahl der Bits für die Unterteilung
Adresse: BA+$10A
Gültige Werte: 0 – 16
Größe: Wort
Intern wird mit 16-Bit-Unterteilung gerechnet. Vor der Ausgabe auf VMEbus wird entsprechend der angegebenen Bit gerundet, und die überzähligen Bit werden gelöscht.

P04.1: Referenzmarken-Abstand Achse 1
P04.2: Referenzmarken-Abstand Achse 2
Adresse: BA+$10C, BA+$10E
Gültige Werte: 0 und alle geradzahligen Werte von 64 … 8192
Größe: Wort
Wert = Grundabstand bei abstandscodierten Referenzmarken in Signalperioden;
0 = eine Referenzmarke bzw. Referenzmarken mit gleichem Abstand

P05.1: Signalperioden für Winkelachse 1
P05.2: Signalperioden für Winkelachse 2
P05.3: Signalperioden verknüpfe Winkelachsen 1 und 2
Adresse: BA+$110 (-$113), BA+$114 (-$117), BA+$118 (-$11B)
Gültige Werte: beliebig
Größe: Langwort
Wert = Signalperioden des Winkelmessgeräts pro Umdrehung.

P06.1: Korrektur ein/aus Achse 1
P06.2: Korrektur ein/aus Achse 2
Adresse: BA+$11C, BA+$11D
Gültige Werte: 0, 1
Größe: Byte
0: Korrektur aus
1: Korrektur ein
P06 ist nur für den Messwertabruf von Bedeutung. Der Messwertabruf ohne Korrektur ist entsprechend schneller.

P07.1: Korrekturwertbereich Start Achse 1
P07.2: Korrekturwertbereich Start Achse 2
Adresse: BA+$11E (-$121), BA+$122 (-$125)
Gültige Werte: beliebig
Größe: Langwort
absolut in Signalperioden (ohne Interpolation) für den korrigierten Bereich

P08.1: Anzahl Stützwerte Achse 1
P08.2: Anzahl Stützwerte Achse 2
Adresse: BA+$126 (-$127), BA+$128 (-$129)
Gültige Werte: 1 – 4096
Größe: Wort
Wert = Anzahl der Korrekturstützpunkte

P09.1: Breite Korrekturstützpunkt Achse 1
P09.2: Breite Korrekturstützpunkt Achse 2
Adresse: BA+$12A (-$12B) , BA+$12C (-$12D)
Gültige Werte: alle Werte außer Null
Größe: Wort
Wert = Abstand der Korrekturstützpunkte in Signalperioden

P10: Abruf der Positionswerte für bestimmte Achsen sperren
Nur die Ausgabe auf den VMEbus wird unterdrückt.
Adresse: BA+$12E
Gültige Werte: 0 – 3; 16 – 19
Größe: Byte
Bit 0: Wertübergabe für Achse 1 auf dem VMEbus sperren
Bit 1: Wertübergabe für Achse 2 auf dem VMEbus sperren
Bit 2: Reserviert
Bit 3: Reserviert
Bit 4: Externes direktes Einspeichern (mit –IK1 / –IK2) sperren
Wird zum Beispiel ein Abruf der Positionswerte über die Funktionsnummer 03 (Abruf für verknüpfte Achsen) durchgeführt, so meldet die Karte bei P10 = 00 die Positionswerte der beiden Einzelachsen sowie den Positionswert der verknüpften Achse jeweils per Interrupt zurück, d.h. der Master muss 3 Interrupts abarbeiten. Benötigt man nur den verknüpften Positionswert, kann mit P10 = 3 der Positionswert der beiden Einzelachsen unterdrückt werden, und es ist nur ein Antwortinterrupt zu bearbeiten.
Bei Funktionsaufruf 01 (Messwertabruf Achse 1) und P10 = 01 kommt kein Antwort-Interrupt zurück. P10 wirkt auch bei allen anderen Einspeicher-Funktionen.

P19.1: Reserviert für kundenspezifische Anwendung
P19.2: Reserviert für kundenspezifische Anwendung
Adresse: BA+$130 (-$133), BA+$134 (-$137)
Größe: Langwort
Wert muss 0 gesetzt sein!

P21: Achsverknüpfung
Adresse: BA+$138
Gültige Werte: 0 – 3
Größe: Byte

0:	keine Verknüpfung	gemeinsamer Wert wird nicht ausgegeben
1:	X1 + X2	gemeinsamer Wert wird ausgegeben
2:	X1 – X2	gemeinsamer Wert wird ausgegeben
3:	(X1 + X2) / 2	gemeinsamer Wert wird ausgegeben

Das Ergebnis der Verknüpfung wird im Datenbereich „gemeinsamer Positionswert“ abgelegt (siehe „6.1 Daten“).

Hinweis:
Wenn mit Parameter 01.x die Zählrichtung einer Einzelachse invertiert wurde, so geht der Wert der Einzelachse mit negativem Vorzeichen in die Summenbildung ein.

P30.1 Richtung (Achse 1) und Geschwindigkeit (Achse 1 und 2) für die Korrekturwertaufnahme
Adresse: BA+$13A
Gültige Werte: 1 – 7
Größe: Byte
Bit 0,Bit1: Geschwindigkeit (Achse 1 und 2)

1: 1,35 kHz – 65 Hz
2: 650 Hz – 35 Hz
3: 80 Hz – 5 Hz

Bit 2: Richtung (Achse 1)
0: positiv
1: negativ

Mit Parameter P30.1 wird der Frequenzbereich für die Korrekturwertaufnahme **beider** Achsen festgelegt. Um einen anderen Frequenzbereich einzustellen, muss die 5-V-Versorgung unterbrochen werden (RESET reicht nicht aus!), anschließend muss vor dem ersten Aufruf von POST der richtige Wert in P30.1 stehen, damit der entsprechende Frequenzbereich eingestellt ist. Wird nur das Richtungsbit (Bit 2) verändert, muss lediglich eine normale Parameterübernahme (P81, Funktionsnummer $0A) durchgeführt werden.
Es wird empfohlen, den Geschwindigkeitsbereich so zu wählen, dass die Eingangsfrequenz möglichst in der Mitte eines der drei Bereiche liegt. Beim Verlassen des Geschwindigkeitsbereichs während einer laufenden Korrekturwertaufnahme bricht die IK 320 die Korrekturwertaufnahme ab und gibt eine Fehlermeldung aus.

P30.2 Richtung Korrekturwertaufnahme Achse 2
Adresse: BA+13B
Gültige Werte: 0, 4
Größe: Byte

Bit 2: Richtung Achse 2
0: positiv
1: negativ

P70.1: Wert für externes Setzen Achse 1
P70.2: Wert für externes Setzen Achse 2
P70.3: Wert für externes Setzen verknüpfte Achsen 1 und 2
Adresse: BA+$13C (-$141), BA+$142 (-$147), BA+$148 (-$14D)
Gültige Werte: beliebig
Größe: 3 Worte
Der eingestellte Wert wird beim externen Setzen über die Funktionseingänge –F1, –F2 als Anzeigewert verwendet (dabei muss in P80.1 bzw. P80.2 die entsprechende Funktion voreingestellt werden). Die Karte berechnet intern einen PRESET-Wert, der zum Zählerwert aufaddiert wird, um den gewünschten Wert in der Anzeige zu erhalten. Damit ist z.B. ein Nullen an einer beliebigen Stelle des Maßstabs möglich. Der berechnete Presetwert wird im Status-Bereich (siehe „6.1 Daten“) angezeigt.

P71.1: Wert für VMEbus-Setzen Achse 1
P71.2: Wert für VMEbus-Setzen Achse 2
P71.3: Wert für VMEbus-Setzen verknüpfte Achsen 1 und 2
Adresse: BA+$14E (-$153), BA+$154 (-$159), BA+$15A (-$15F)
Gültige Werte: beliebig
Größe: 3 Worte
Die Werte haben die gleiche Funktion wie P70, nur wird der Vorgang PRESET über einen VMEbus-Funktionsaufruf gestartet (Funktionsnummern 04, 05, 06).

P72.1: Achsoffset Achse 1
P72.2: Achsoffset Achse 2
P72.3: Achsoffset verknüpfte Achsen 1 und 2
Adresse: BA+$160 (-$165), BA+$166 (-$16B), BA+$16C (-$171)
Gültige Werte: beliebig
Größe: 3 Worte
Der eingestellte Wert wird bei der Istwertbildung unverändert aufaddiert. Mit diesem Parameter ist eine Verschiebung des Positionswerts möglich. Der Wert wird verrechnet, sobald er mit Parametereingabe (P81, Funktionsnummer $0A) im lokalen RAM gespeichert ist. Nach RESET ist P72 = 0.

P72 wirkt nicht auf die Korrekturwert-Zuordnung.

P80.1: Externe Funktion –F1
P80.2: Externe Funktion –F2

Adresse: BA+$172, BA+$173
Gültige Werte: 0 – 6
Größe: Byte

$00:	keine Funktion
$01:	Abruf Achse 1: Einspeichern, Berechnen und Bereitstellen des Positionswerts
$02:	Abruf Achse 2: Einspeichern, Berechnen und Bereitstellen des Positionswerts
$03:	Abruf verknüpfte Achsen 1 und 2: Einspeichern, Berechnen und Bereitstellen der Positionswerte
$04:	Preset Achse 1 mit Wert aus Parameter P70.1
$05:	Preset Achse 2 mit Wert aus Parameter P70.2
$06:	Preset verknüpfte Achsen 1 und 2 mit Wert aus P70.3

Aktive Flanken am –F1 bzw. –F2 Eingang der Buchse X41 erzeugen einen Interrupt auf der IK 320. Diese führt die in Parameter P80.1 (bzw P80.2) eingestellte Funktion aus. Nach Beendigung der Funktion oder im Fehlerfall wird eine Meldung an den Master zurückgegeben.

P81: Master-Interrupt-Funktion
Adresse: BA+$100
Gültige Werte: 01 – $0C, $18, $19 – $1B, $20 – $23
Größe: Wort

$0000:	keine Funktion
$0001:	Abruf Achse 1: Einspeichern, Berechnen und Bereitstellen des Positionswerts
$0002:	Abruf Achse 2: Einspeichern, Berechnen und Bereitstellen des Positionswerts
$0003:	Abruf verknüpfte Achsen 1 und 2: Einspeichern, Berechnen und Bereitstellen der Positionswerte
$0004:	Preset Achse 1 mit Wert aus Parameter P70.1
$0005:	Preset Achse 2 mit Wert aus Parameter P70.2
$0006:	Preset verknüpfte Achsen 1 und 2 mit Wert aus P70.3
$0007:	POST starten
$0008:	Referenzmarke Achse 1 überfahren
$0009:	Referenzmarke Achse 2 überfahren
$000A:	Parameter aktualisieren
$000B:	Korrekturwerte aufnehmen Achse 1
$000C:	Korrekturwerte aufnehmen Achse 2
$0018:	Achse 1 und Achse 2 über Referenzmarken fahren
$0019:	Achse 1 Start ohne REF
$001A:	Achse 2 Start ohne REF
$001B:	Achse 1+2 Start ohne REF
$0020:	Korrekturwerte lesen Achse 1
$0021:	Korrekturwerte schreiben Achse 1
$0022:	Korrekturwerte lesen Achse 2
$0023:	Korrekturwerte schreiben Achse 2

Wird vom Master ein Interrupt auf der IK 320 generiert (Beschreiben der Interrupt-Adresse im Adressraum A16), führt die IK 320 die in Parameter P81 eingestellte Funktion aus. Nach Beendigen der Funktion oder im Fehlerfall wird eine Meldung an den Master zurückgegeben.

7. Funktionen

7.1 Unterbrechbarkeit der Funktionen

Die folgende Tabelle zeigt die unterbrechbaren und nicht unterbrechbaren Funktionen:

Funktion bzw. Funktionsnummer	**Bedeutung**	**Unterbrechbarkeit**
nur Hard-Latches –Impuls1, –Impuls2, X41, Pin 4 und 5	externes, direktes Einspeichern	gegenseitig unterbrechbar, sonst nur vom Master-Interrupt mit Funktionsnummer $0A unterbrechbar. Ein Interrupt der selben Quelle ist so lange gesperrt, bis wieder der inaktive Pegel am Eingangspin erkannt wird.*
nur Hard-Latches, –F1, –F2, X41, Pin 6 und 7	externes, indirektes Einspeichern	nur durch externes, direktes Einspeichern und vom Master-Interrupt mit Funktionsnummer $0A unterbrechbar. Ein Interrupt der selben Quelle ist so lange gesperrt, bis wieder der inaktive Pegel am Eingangspin erkannt wird.*
$01	Soft-Latch X1	nur von Hard-Latches unterbrechbar
$02	Soft-Latch X2	nur von Hard-Latches unterbrechbar
$03	Soft-Latch X1/X2	nur von Hard-Latches unterbrechbar
$04	Preset X1	nicht unterbrechbar
$05	Preset X2	nicht unterbrechbar
$06	Preset X1/X2	nicht unterbrechbar
$07	POST	nicht unterbrechbar
$08	REF X1	unterbrechbar für Hard-, Soft-Latches und REF X2; wenn Achse im Status WAIT FOR REF, abbrechbar
$09	REF X2	unterbrechbar für Hard-, Soft-Latches und REF X1, wenn Achse im Status WAIT FOR REF, abbrechbar
$0A	Setzen Parameter	nicht unterbrechbar
$0B	Kompensation X1	nicht unterbrechbar; abbrechbar
$0C	Kompensation X2	nicht unterbrechbar; abbrechbar
$18	REF X1/X2	nach Initialisierung unterbrechbar für Hard- und Soft-Latches; abbrechbar
$19	Achse 1 Start ohne REF	nicht unterbrechbar
$1A	Achse 2 Start ohne REF	nicht unterbrechbar
$1B	Achse 1+ 2 Start ohne REF	nicht unterbrechbar
$20	Upload X1	nicht unterbrechbar
$21	Download X1	nicht unterbrechbar
$22	Upload X2	nicht unterbrechbar
$23	Download X2	nicht unterbrechbar

*Die Unterbrechbarkeit mit Master-Interrupt und Funktionsnummer $0A ist nötig, um ein Disable der externen Funktionen über „Setzen Parameter“ sicher zu ermöglichen.

Begriffserklärungen
Hard-Latches: Einspeichern extern direkt (über X41, –IMPULS1/2, –KONTAKT1/2), Einspeichern extern indirekt (über X41, –F1/2), VMEbus direkt (Einspeichern über SYCHRON-Adresse)
Soft-Latches: VMEbus indirekt (Funktionsnummern $01, $02, $03)
Nicht unterbrechbar: Erst wenn die Funktion von der IK 320 vollständig ausgeführt und mit Antwort-Interrupt abgeschlossen ist, werden neue Funktionsaufrufe akzeptiert.
Abbrechbar: Funktion kann man mit Funktionsnummer 0 und Master-Interrupt abbrechen

7.2 Power On Self Test (POST)

Nach dem Einschalten muss der Master auf der IK 320 einen Initialisierungsvorgang starten. Dies geschieht durch den Aufruf von POST, der etwa vier Sekunden dauert.
Vorher kann die IK 320 nicht auf das gemeinsame RAM zugreifen und reagiert auch auf keinen anderen Funktionsaufruf. Der Master darf beliebig auf das gemeinsame RAM schreiben und lesen und kann jetzt zum Beispiel die Parameter für die Karte setzen.
Beim Aufruf von POST werden automatisch die Parameter vom gemeinsamen RAM übernommen und auf gültige Werte geprüft. Sind Werte fehlerhaft, werden Default-Werte für die Parameter gesetzt und eine Fehlermeldung generiert, diese Default-Werte werden auch ins gemeinsame RAM geschrieben. Im Fehlerfall wird die Nummer des Parameters, bei dem der Fehler aufgetreten ist, ins VME-RAM (BA+$EA) geschrieben.

Wichtig:
Parameter P30.1, der den Frequenzbereich der Korrekturwertaufnahme festlegt, muss nach POWER ON **vor** dem ersten Aufruf von POST mit dem richtigen Wert beschrieben werden, damit die Hardware für die Korrekturwertaufnahme richtig initialisiert wird.
Während POST abläuft wird die Kontrolle über das gemeinsame RAM an die IK 320 übergeben, und der Master darf erst wieder auf das gemeinsame RAM zugreifen, wenn sich die IK 320 durch einen Antwort-Interrupt an den Master zurückgemeldet hat.

Die IK 320 führt während POST folgende Routinen zur Initialisierung und zum Test der Hardware durch:
- Prüf- und CRC-Summentest der EPROMs
- CRC-Summentest des Korrekturwertspeichers
- Test lokales RAM
- Test gemeinsames RAM
- Test BERR-Timeout
- Test Zählerbausteine für Achse 1 und 2, Zählerbausteine sind nach POST gestoppt

Die Rückmeldung der IK 320 erfolgt über einen Interrupt, den die Karte an den Master ausgibt. Im Interrupt Status-Wort steht im High-Byte der Wert $07 (= POST beendet) und im Low-Byte die Fehlermitteilung (z.B. $00 = kein Fehler, siehe „6.1 Daten", IK-Interrupt Status).

Aufruf von POST:
Der Master löscht das IK-Interrupt-Status-Byte, um der IK 320 die Möglichkeit zu geben, Meldungen an den Master zurückzugeben.
Der Master schreibt als nächstes den Wert $07 in **P81 Master-Interrupt-Funktion** und löst einen Master-Interrupt auf der IK 320 aus (siehe „5.4.2 Vom Master zur IK 320").

7.3 Überfahren der Referenzmarken

Bei inkrementalen Messsystemen geht nach einer Stromunterbrechung die Zuordnung zwischen Achsposition und Anzeigewert verloren. Durch Überfahren von Referenzmarken wird nach einer Stromunterbrechung der Bezug zur Achsposition wieder hergestellt.HEIDENHAIN-Messsysteme besitzen entweder eine oder mehrere abstandscodierte Referenzmarken. In Parameter P04 legen Sie fest, welche Referenzmarken Ihr Messsystem besitzt.

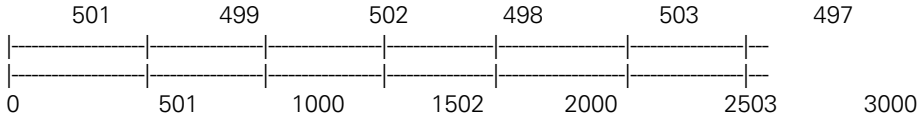
Vorgehensweise bei einer Referenzmarke:
Der Master gibt die Interrupt-Funktion $08 (Referenzmarke Achse 1 überfahren), $09 (Referenzmarke Achse 2 überfahren) oder $18 (beide Referenzmarken überfahren) aus.
Die IK 320 führt folgende Funktionen für die gewählte Achse aus:

- Zähler stopp: Zähler wird gestoppt und der Zustand im Register „Status Achse 1“gekennzeichnet.
- Zähler reset: Zählerstand wird auf 0 gesetzt.
- Zähler Start mit RI: Mit der nächsten Referenzmarke, die überfahren wird, startet der Zähler. Der Status der Zähler wird laufend aktualisiert, d.h. in den Daten-Registern „Status Achse 1“ (BA+$06) bzw. „Status Achse 2“ (BA+$0E) (siehe „6.1 Daten“) sind die aktuellen Zustände gespeichert.

Nach dem Antwort-Interrupt liest der Master das IK-Interrupt-Status-Register und erkennt über den IK-Interrupt-Status $0800 bzw. $0900, dass die Referenzmarke überfahren wurde. Abrufe von Positionswerten während des Überfahrens der Referenzmarken liefern immer den Positionswert 0 zurück, d.h. die entsprechende Achse ist noch nicht gestartet.

Vorgehensweise bei abstandscodierten Referenzmarken:
Bei Messsystemen mit abstandscodierten Referenzmarken befinden sich über den ganzen Messweg Referenzmarken in einem festen Abstand. Zwischen zwei dieser Referenzmarken befindet sich eine dritte, deren Abstand zu den anderen beiden so variiert, dass jeder Abstand ein Vielfaches der Teilungsperiode beträgt und nur einmal über den gesamten Messweg vorkommt. So kann die IK 320 nach einer Stromunterbrechung durch bloßes Überfahren von zwei Referenzmarken die Zuordnung zwischen Achsposition und Anzeigewert wieder herstellen.

Bei einem Grundabstand von 1 000 Signalperioden ergibt sich die folgende Verteilung für die Referenzmarken:



In Parameter P04 wird der Grundabstand für die abstandscodierten Referenzmarken in Signalperioden festgelegt. Der Master startet das Überfahren der Referenzmarken mit der Interrupt-Funktion $08 (Referenzmarke Achse 1 überfahren), mit $09 (Referenzmarke Achse 2 überfahren) oder mit $18 (beide Referenzmarken überfahren).

Die IK 320 führt folgende Funktionen für die gewählte Achse aus:
- der Zähler wird gestoppt und auf 0 gesetzt
- mit dem Überfahren der ersten Referenzmarke wird der Zähler gestartet
- mit dem Überfahren der zweiten Referenzmarke speichert die IK 320 den Positionswert und hält somit den Abstand zwischen den Referenzmarken in Inkrementen fest. Aus diesem Abstand berechnet die IK 320 die absolute Position bezogen auf die erste Referenzmarke auf dem Maßstab (Position „0“ auf der Zeichnung).

Auch LIDA-Bänder mit abstandscodierten Referenzmarken, die auf einem Ring montiert sind und somit an der Stoßstelle ein unregelmäßiges Referenzintervall haben, werden von der IK 320 ausgewertet. Allerdings muss die Achse in P02 als Winkelachse definiert sein, damit die Referenzmarken-Auswertung an der Stoßstelle funktioniert.

Anmerkung: Hintergrundtests (CRC-Summe EPROM, CRC-Summe Korrekturdaten, Abstände codierter REF-Marken) sind ausgeschaltet, sobald eine Achse im Status „WAIT FOR REF“ ist.

Überfahren der Referenzmarken abbrechen
Das Überfahren der Referenzmarken können Sie nur mit Funktionsnummer 0 und Master-Interrupt abbrechen.

Start der Achsen ohne Überfahren der Referenzmarken
Für Anwendungen, bei denen sofort nach dem Einschalten ein Geschwindigkeitssignal gebraucht wird, können Achsen auch ohne Referenzierung gestartet werden.

Hierzu können folgende Interrupt-Funktionen verwendet werden:
$19 (Achse 1 starten ohne Referenzmarke),
$1A (Achse 2 starten ohne Referenzmarke),
$1B (Achse 1+2 starten ohne Referenzmarke)

7.4 Korrekturwertaufnahme zur Kompensation von Abweichungen der Messsystem-Signale

Die IK 320 kann Abweichungen der Messsystem-Signale, die in einem Korrekturlauf ermittelt werden, kompensieren. Eine Karte kann nur jeweils für **eine** Achse die Korrekturwertaufnahme durchführen. Es ist aber möglich, mehrere Karten gleichzeitig für eine Korrekturwertaufnahme zu starten. Somit muss man, wenn beide Achsen einer Karte genutzt werden, zwei Korrekturwertläufe durchführen. Die Korrekturwertaufnahme kann in positiver und negativer Richtung erfolgen (Parameter 30.x, Bit 2).
Für jede Achse müssen die Parameter P07.x, P08.x, P09.x und P30.x gesetzt werden. P06.x spielt für die Korrekturwertaufnahme keine Rolle.
P07.x bezeichnet den Anfangswert des Korrekturbereiches: Bei positiver Richtung der Korrekturwertaufnahme ist es der Startwert, bei negativer der Endwert des korrigierten Bereiches. P08.x gibt die Anzahl der Korrekturstützpunkte an. P09.x gibt den Abstand der Korrekturstützpunkte in Signalperioden an.
Mit P07.x,P08.x und P09.x ist der Korrekturwertbereich bestimmt. Mit Parameter P30.1 wird ein Geschwindigkeitsintervall für das Überfahren des Korrekturwertbereiches gewählt. Dieses Intervall gilt für beide Achsen. Wird die gewählte Geschwindigkeit nicht eingehalten, so wird die Korrekturwertaufnahme abgebrochen und eine Fehlermeldung ausgegeben.

Wichtig:
Die Parameter P07.x,P08.x und P09.x dürfen nach der Korrekturwertaufnahme nicht mehr verändert werden, weil sonst für die Messwertabrufe die Zuordnung Positionswert zu Korrekturwert nicht mehr stimmt. Die Parameter P70.x, P71.x und P72.x beeinflussen dagegen diese Zuordnung nicht.

Ablauf der Korrekturwertaufnahme:
Die Achse muss gestartet sein (siehe „7.3 Überfahren der Referenzmarken"). Bei **Linearachsen** wird die Position vor dem eigentlichen Start der Korrekturwertaufnahme überprüft. Sie muss mindestens 10 Signalperioden **vor** dem Startwert des Korrekturwertbereichs liegen, entsprechend der eingestellten Abtast-Richtung.

Bei **Winkelachsen** wird keine Positionsprüfung vorgenommen, wenn aber der Abstand zum Korrekturbereich nicht mindestens 10 Signalperioden groß ist, wird der Start der Korrekturwertaufnahme in die nächste Umdrehung verschoben.
Beim Funktionsaufruf darf die Achse noch im Stillstand sein, muss sich aber dann in die richtige Richtung bewegen und am Beginn des Korrekturbereichs die richtige Geschwindigkeit erreicht haben.
Bewegt sich die Achse in die falsche Richtung, wird nach 100 Signalperioden die Korrekturwertaufnahme abgebrochen und der Fehlerstatus gesetzt.
Nach Erreichen des Bereichs beginnt die Karte mit dem Abtasten der Messsystem-Signale. Dies nimmt die volle Rechenleistung in Anspruch und die Kontroll-LED (siehe „4.2 Kontroll-LED") hört zu blinken auf. Tritt während des Abtastens ein Fehler auf, wird die Funktion abgebrochen und der Fehlerstatus gesetzt. Nach Beendigung des Abtastens sendet die Karte den Interrupt „Messwertaufnahme beendet", worauf die Achsbewegung gestoppt werden kann. Die LED beginnt wieder zu blinken, und die Karte bereitet die Abtastwerte rechnerisch auf.

Hinweis:
Bei 4 096 Korrekturstützpunkten dauert das Berechnen der Korrekturwerte etwa drei Minuten, 30 Sekunden. Während dieser Zeit ist im Status-Byte der jeweiligen Achse das Bit7 gesetzt (siehe „6.1 Daten").

Bei der Berechnung können Fehler auftreten, die eventuell auf einen ungleichmäßigen Antrieb zurückzuführen sind. In diesem Fall wird die Korrekturwertberechnung abgebrochen und der Fehlerstatus gesetzt. Verläuft die Korrekturwert-Aufnahme und Berechnung fehlerfrei, so wird zum Schluss der Antwort-Interrupt „Korrekturwertaufnahme OK" gesendet.

Achtung:
Die Beschleunigungen während der Korrekturwertaufnahme sollen möglichst gering oder konstant sein.

Korrekturwert-Aufnahme abbrechen
Die Korrekturwert-Aufnahme können Sie nur mit Funktionsnummer 0 und Master-Interrupt abbrechen.

7.5 Korrekturwerte lesen und schreiben

Ein kompletter Satz Korrekturwerte beinhaltet immer eine Stützpunktzahl von P08x + 2. D.h. beim Lesen und Schreiben der Werte müssen immer die Stützpunkte von 0 bis P08x+1 gelesen oder geschrieben werden. Im gemeinsamen RAM sind zwei Datenbereiche reserviert: einer zum Lesen (BA+$50 bis BA+$6F) und einer zum Schreiben (BA+$70 bis BA+$8F). Zum Übertragen eines Stützpunkts sind Stützpunktnummer, acht Korrekturkoeffizienten (K1 bis K8) und die BCC-Summe über Stützpunktnummer und acht Koeffizienten nötig. Es handelt sich dabei ausschließlich um 16-Bit-Datenworte.
Die BCC-Summe ist das Ergebnis einer EXOR-Verknüpfung der Operanden.

Ein Koeffizienten-Paar enthält den Real- und Imaginärteil des jeweiligen Vektors der Funktion.
K1/K2 = Koeffizienten der Grundschwindung der Fehlerfunktion
K3/K4 = Koeffizienten der 2. harmonischen Fehlerfunktion
K5/K6 = Koeffizienten der 3. harmonischen Fehlerfunktion
K7/K8 = Koeffizienten der 4. harmonischen Fehlerfunktion

Funktionsnummern:
$20: Lesen Korrekturwerte X1
$21: Schreiben Korrekturwerte X1
$22: Lesen Korrekturwerte X2
$23: Schreiben Korrekturwerte X2

Status-Antwortinterrupt:

$20xx	Korrekturwerte Achse 1 gelesen xx = $00: 0K $01: keine gültigen Korrekturwerte im Speicher $02: falsche Stützpunktnummer
$21xx	Korrekturwerte Achse 1 geschrieben xx = $00: 0K $01: BCC-Fehler bei Übertragung $02: falsche Stützpunktnummer $03: falsche Achse
$22xx	Korrekturwerte Achse 2 gelesen xx = $00: 0K $01: keine gültigen Korrekturwerte im Speicher $02: falsche Stützpunktnummer
$23xx	Korrekturwerte Achse 2 geschrieben xx = $00: 0K $01: BCC-Fehler bei Übertragung $02: falsche Stützpunktnummer $03: falsche Achse

7.5.1 Korrekturwerte lesen

Das Lesen der Werte ist nur möglich, wenn gültige Korrekturdaten im Speicher sind.
Die Stützpunktnummer muss in die vorgesehene Speicherstelle des gemeinsamen RAMs (BA+$50) eingeschrieben werden. Anschließend wird die Funktionsnummer (z.b. $20 für X1) in P81 geschrieben und ein Master-Interrupt ausgelöst. Die Karte schreibt die acht Koeffizienten dieses Stützpunkts in das gemeinsame RAM, bildet die BCC-Summe über Stützpunktnummer und acht Koeffizienten und legt sie im gemeinsamen RAM (BA+$62) ab. Anschließend antwortet sie mit Antwortinterrupt und Status (z.B. $2000 bei fehlerfreiem Ablauf). Nun kann der Anwender die BCC-Summe überprüfen und die Stützpunktnummer samt Koeffizienten auslesen und abspeichern. Im Prinzip ist das Lesen der Korrekturstützpunkte in beliebiger Reihenfolge möglich. Es wird aber empfohlen, die Reihenfolge 0 bis P08x+1 einzuhalten, weil beim Beschreiben des Korrekturwert-RAMs (siehe 7.5.2) diese Reihenfolge zwingend vorgeschrieben ist.
Im gemeinsamen RAM ist die CRC-Summe über das Korrekturwert-RAM einer Achse auslesbar (BA+$90 für X1, BA+$92 für X2, 16-Bit-Werte). Diese CRC-Summe kann zusätzlich zu einem kompletten Korrekturwertsatz einer Achse als Kontrollgröße gespeichert werden, denn nach dem Korrekturwert Schreiben muss die IK 320 wieder die gleiche CRC-Summe ermitteln. Ebenso ist es ratsam, den zugehörigen Parametersatz festzuhalten.

7.5.2 Korrekturwerte schreiben

Beim Beschreiben ist die Reihenfolge der Korrekturstützpunktenummern von 0 bis P08x+1 zwingend. Ebenso können die Werte einer Achse nur nacheinander übertragen werden. Wird die Reihenfolge nicht eingehalten, tritt ein BCC-Fehler auf oder wird ein Stützpunkt für die andere Achse beschrieben, wird der Vorgang abgebrochen und man muss wieder mit Stützpunktnummer 0 neu beginnen.
Die Stützpunktnummer, acht Koeffizienten und die BCC-Summe werden über Stützpunktnummer und acht Koeffizienten in das gemeinsame RAM (BA+$70 bis $82) eingeschrieben. Anschließend wird die Funktionsnummer (z.b. $21 für X1) in P81 geschrieben und ein Master-Interrupt ausgelöst. Die Karte prüft die BCC-Summe der Koeffizienten, schreibt die acht Koeffizienten dieser Stützpunkts in einen Zwischenspeicher und antwortet mit dem entsprechenden Interrupt und Status (z.B. $2100 bei fehlerfreiem Ablauf). Sind alle nötigen Stützpunkte übertragen, schreibt die Karte die Werte aus dem Zwischenspeicher in den eigentlichen Korrekturwertspeicher und bildet die CRC-Summe über diesen Speicherbereich neu. Diese CRC-Summe wird auch im gemeinsamen RAM abgelegt (BA+$90 für X1, BA+$92 für X2, 16-Bit-Werte). Sie muss mit der CRC-Summe übereinstimmen, die beim Lesen dieser Korrekturwerte ermittelt wurde.

8. Positionswert abrufen

Der Abruf des Positionswerts wird durch einen Einspeichervorgang gestartet. Der Positionswert wird berechnet, im Bereich „Daten“ abgelegt und ein Interrupt an den Master ausgegeben. Im Interrupt-Status-Register wird außerdem die Einspeicherquelle des Messwerts übergeben.
Je nach Parametereinstellung und Art des Abrufs (extern oder über VMEbus) liefert die Karte keinen oder bis zu drei Antwort-Interrupts (siehe P10, P21), die alle abgearbeitet werden müssen, damit die Karte für neue Funktionsaufrufe wieder bereit ist.

8.1 Positionswert einspeichern

Der Positionswert kann direkt oder indirekt eingespeichert werden.

Hinweis:
Beim **direkten** Einspeichern wird auf der IK 320 ein lokaler Interrupt erzeugt; beim **indirekten** Einspeichern wird das Einspeicher-Signal per Software über die lokale CPU der IK 320 erzeugt. Dadurch ergeben sich unterschiedliche Zeitpunkte für das Einspeichern der einzelnen Messwerte.

Es gibt folgende Einspeichermöglichkeiten:

- extern direkt: Signale –IMPULS1, –IMPULS2, –KONTAKT1, –KONTAKT2 an Buchse X41 (externe Funktionen).
- extern indirekt: Signale –F1, –F2 an Buchse X41 erzeugen einen lokalen Interrupt; Einspeichern per Software.
- VMEbus direkt: Anlegen der Synchron-Einspeicher-Adresse.
- VMEbus indirekt: Generierung eines lokalen Interrupts; Einspeichern per Software.
- durch Überfahren der Referenzmarken: Das Überfahren einer Referenzmarke löst einen Einspeichervorgang aus.
- per Timer: Einspeichern mit lokalem Zeitbezug, wird nur für die Korrekturwertaufnahme benötigt.

Jeder lokale Einspeicherimpuls wird auf X41 Pin1 (–LOUT) ausgegeben. Unabhängig davon, welche Einspeichermöglichkeit gewählt bzw. welche Achse eingespeichert wurde.

Aktuelle Positionswerte können nur abgefragt werden, wenn nach POST die Referenzmarken überfahren worden sind (siehe „7.3 Überfahren der Referenzmarken“).

8.1.1 Pinbelegung des Anschlusses für externe Funktionen (X41)

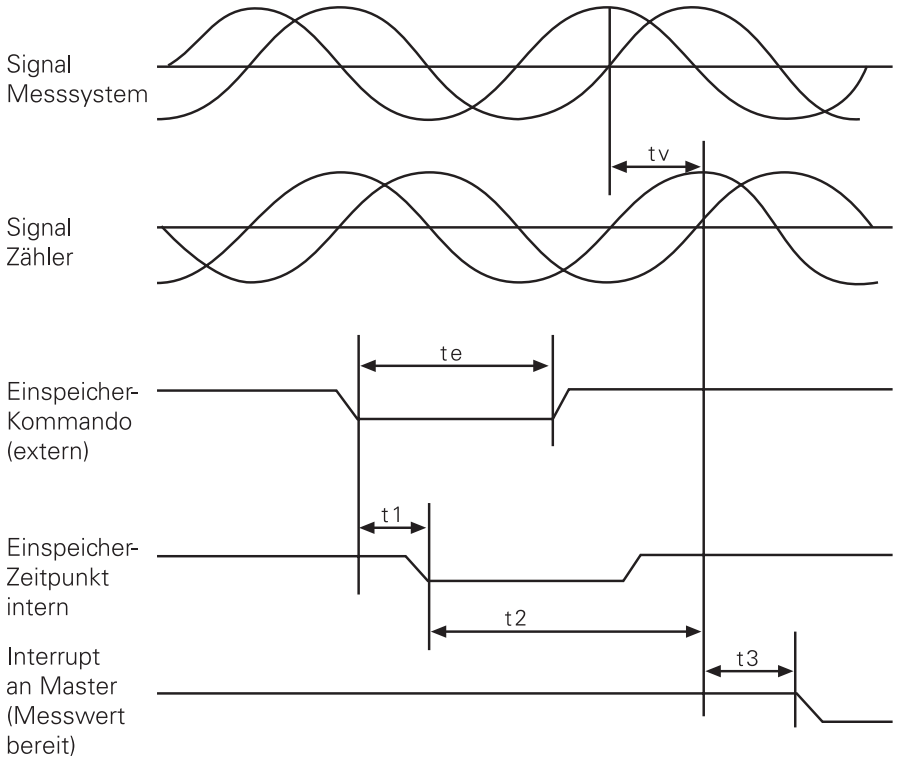
Signalbedeutung	Signalbezeichnung	Belegung Anschluss-Nr.
Einspeichern über Impuls für Achse 1	–IMPULS1	4
Einspeichern über Impuls für Achse 2	–IMPULS2	5
Einspeichern über Kontakt für Achse 1	–KONTAKT1	2
Einspeichern über Kontakt für Achse 2	–KONTAKT2	3
Einspeichern über Interrupt für Achse 1	–F1	6
Einspeichern über Interrupt für Achse 2	–F2	7
0 V	0 V	8
0 V	0 V	9
Ausgang für Einspeicher-Impuls	–LOUT	1

Signalpegel:

Bezeichnung	min.	max.	Einheit
U_{eH}	3	15	V
U_{eL}	– 0,5	1	V
I_{eL}		6	mA

Die Eingänge sind LOW-aktiv und werden durch interne Pull-Up Widerstände auf High-Pegel gehalten. Die Ansteuerung kann durch TTL-Standard, LS, ALS oder CMOS Bausteine erfolgen.

Zeitdiagramm



Signal	Bez.	min.	max.	Einheit
Signalverzögerung vom Messsystemeingang bis zum A/D-Wandler, bzw.Zähler	t_v		4	µs
Breite Einspeicherkommando	t_e			*
Verzögerung Einspeichersignal	t_1			*
Zeitpunkt des eingespeicherten Messsignalwertes in Bezug zum Einspeicherzeitpunkt	t_2		100	ns
Messwert bereit (pro Achse, korrigiert)	t3		400	µs
Messwert bereit (pro Achse, unkorrigiert)	t_3		180	µs

* für die einzelnen Quellen verschieden

8.1.2 Externes, direktes Einspeichern

Durch Kontaktschluss der Signale –IMPULS1, –IMPULS2 oder –KONTAKT1, –KONTAKT2 gegen 0 V wird ein Einspeichersignal per Hardware erzeugt. Die Eingänge –IMPULS und –KONTAKT unterscheiden sich in ihrem Zeitverhalten:

Impuls: $t_e \geq 1{,}2$ µs, $t_1 \leq 0{,}8$ µs **Kontakt:** $t_e \geq 7{,}0$ ms, $t_1 \leq 4{,}5$ ms

Achtung:
Bei einer Achsverknüpfung (P21) ist der Einspeicherimpuls am Eingang –IMPULS1/KONTAKT1 für beide Achsen maßgebend. Ein Impuls an –IMPULS2/KONTAKT2 wird nicht ausgewertet.

8.1.3 Externes, indirektes Einspeichern über Interrupt

Durch Kontaktschluss der Signale –F1 und –F2 gegen 0 V wird auf der Karte ein lokaler Interrupt erzeugt. Die CPU führt die in Parameter P80.1, bzw. P80.2 eingestellten Funktionen aus, die Parameterwerte $01, $02, $03 generieren ein Einspeichersignal in den entsprechenden Kanälen.

Zeitverhalten F1, F2:	te	≥ 25 µs
	t1	≤ 10 µs

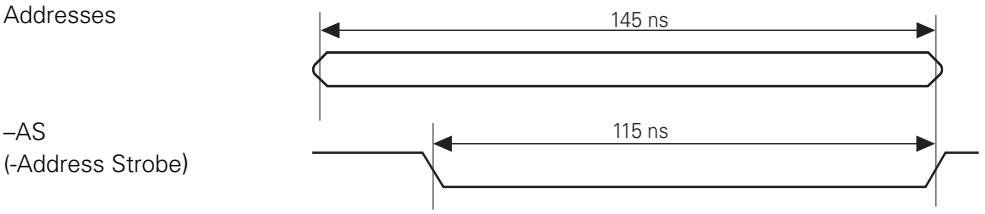
8.1.4 Direktes Einspeichern per VMEbus

Über den VMEbus kann das Einspeichern durch einen Adresszugriff im Adressraum A16 erfolgen.

Zeitdiagramm

te = ADO-Zugriff VME (ADDRESS-ONLY-Zugriff)
t1 ≤ 500ns ab DS0/DS1

Timing:



8.1.5 Direktes Einspeichern per VMEbus innerhalb einer Gruppe

Durch einen ADO-Zugriff auf die Gruppenadresse des Adressraums A16 wird in allen Achsen dieser Gruppe (mehrere Karten) gleichzeitig ein Einspeicher-Vorgang ausgelöst. Sollen einzelne Achsen ausgesperrt werden, so muss dies über Parameter P10 eingestellt werden. Parameter P21 wird berücksichtigt (Achsverknüpfung). Der übergeordnete Rechner muss einen ADO-Zyklus durchführen, d.h. keine der Karten liefert ein –DTACK-Signal zurück.

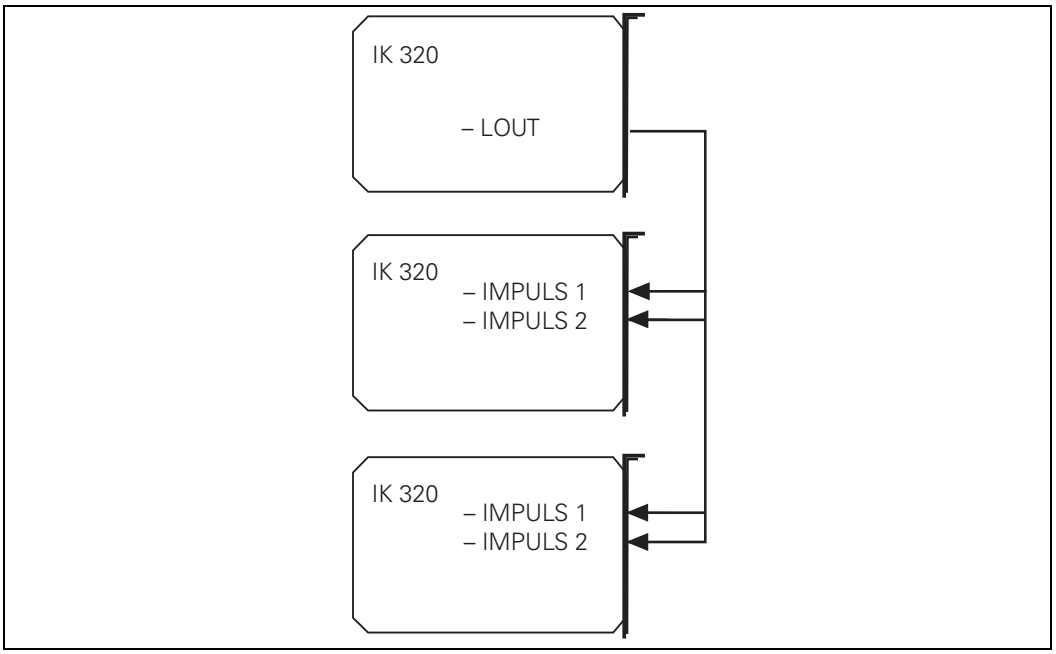
Hinweis:
Sollte der VMEbus-Rechner nicht die Fähigkeit besitzen, einen ADO durchzuführen, so kann alternativ bei **einer** Karte der Gruppe der Jumper J3 bestückt werden. Diese Karte liefert dann ein –DTACK-Signal.

8.1.6 Direktes Einspeichern per VMEbus über mehrere Gruppen

Wenn die Positionswerte von Karten, die auf unterschiedliche Gruppenadressen eingestellt sind, synchron eingespeichern werden, wird es wie folgt bearbeitet:

- Verbinden Sie die Ausgänge –LOUT der Karten der ersten Gruppe mit den Eingängen –IMULS1 und –IMPULS2 der weiteren Karten.
- Den Positionswert der Karten der ersten Gruppe speichern durch einen ADO-Zugriff auf die Gruppenadresse.
- Über die Ausgänge –LOUT werden die Karten der weiteren Gruppen synchron eingespeichert.

Laufzeit des Einspeicherimpulses bis –LOUT: < 10 ns

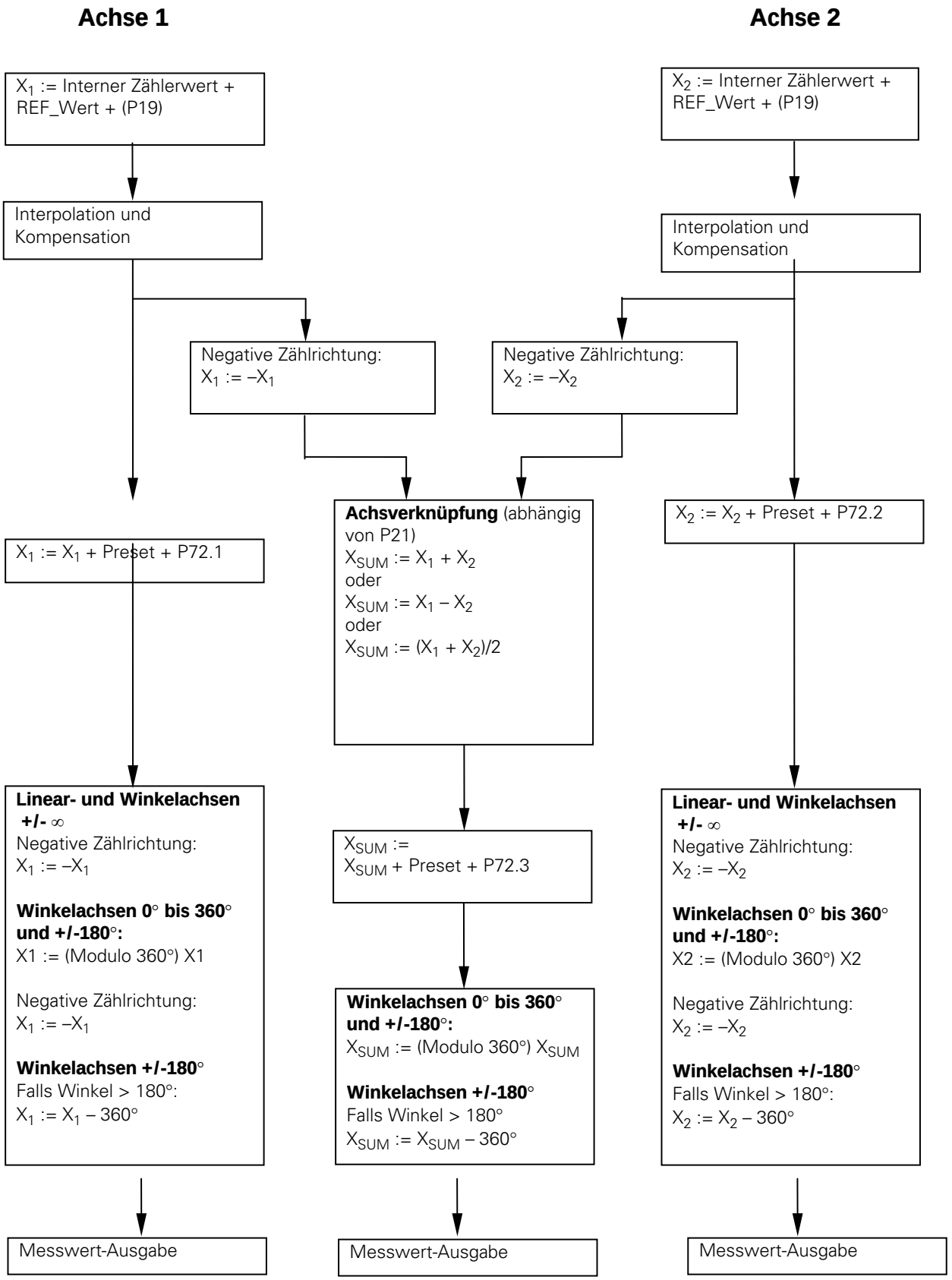


8.1.7 Indirektes Einspeichern per VMEbus

Durch einen ADO-Zugriff auf die Interrupt-Adresse des Adressraums A16 wird auf der IK 320 ein lokaler Interrupt generiert. Die CPU führt die in Parameter P81 eingestellte Funktion aus: Die Funktionen $01, $02, $03 generieren ein Einspeichersignal in den entsprechenden Achsen.

8.2 Positionswert berechnen

Das folgende Ablaufdiagramm zeigt, wie die IK 320 den Messwert in Inkrementen bildet. X1 entspricht dem Messwert der 1. Achse und X2 dem Messwert der 2. Achse.



Der REF-Wert ist der Codewert der ersten überfahrenen Referenzmarke bei abstandscodierten Referenzmarken; bei Messsystemen mit nur einer Referenzmarke ist REF-Wert = 0

Der Preset wird wie folgt berechnet:
Preset = Gewünschter Anzeigewert (P70/P71) – interner Zählerwert – P72 – REF-Wert

Es wird ein Wert berechnet, so dass der in den Parametern P70 bzw. P71 gewünschte Wert an dieser Stelle als Positionswert erscheint. Dadurch ist z.B. ein Nullen an einer beliebigen Stelle möglich, wenn man vor der Preset-Berechnung den Wert 0 als gewünschten Anzeigewert vorgibt.

9. Programmierung

Die Programmierung einer IK 320 mit zwei Achsen wird in dieser Beschreibung mit einem „BORLAND C"-Beispiel gezeigt. Das Programm wurde auf einem Industrie-Rechner (von Firma ROTEC, D-76411 Rastatt) mit einer INTEL 486 CPU (DOS-Version 6.0), VMEbus-Interface und BORLAND C++-Compiler (Version 4.0) erstellt und getestet.
Folgende Dateien auf der mitgelieferten Diskette dienen zur Anpassung des ISA-Bus an den VMEbus:

- VMEROTEC.H und
- VMEINIT.C

Die Daten- und Funktionsdefinitionen in diesen Dateien werden nicht weiter erläutert, da sie keine Funktionen der IK 320 beschreiben.

Die Dateien

- IK320.H und
- IK320.C

enthalten die wichtigsten Daten- und Funktionsdefinitionen, die bei der Arbeit mit der IK 320 benötigt werden

In den Dateien

- SAMPLE.H und
- SAMPLE.C

wird eine einfache Anwendung mit den Funktionen aus „IK320.C"gezeigt.

Ein lauffähiges Programm erhalten Sie, indem Sie die Dateien

- VMEINIT.C
- IK320.C
- SAMPLE.C

in ein „Projekt" einbinden.

Ein Programm für die IK 320 muss im wesentlichen die folgenden Funktionen ausführen:

- Karte initialisieren
- Referenzpunkte überfahren
- Positionswerte anzeigen, speichern und auswerten

Zusätzlich müssen bei der Inbetriebnahme Korrekturwerte zur Kompensation von Abweichungen der Messsystem-Signale aufgenommen werden. Die Korrekturwertaufnahme muss wiederholt werden

- nach einem Ausfall der „stand-by"-Stromversorgung oder
- nach dem Austausch eines Messsystems oder Abtastkopfes einer Achse

Die einzelnen Funktionen des Programms SAMPLE.C werden im folgenden beschrieben.

Das Herzstück dieses Beispiels ist die Interrupt-Funktion **NewInterruptRoutine()**. Diese Funktion behandelt alle Interrupt-Ursachen der IK 320. Die Funktion **Read-IK_Interrupt_Status()** liest den IK-Interrupt-Status (BA + $18). **Evaluate_IK_Interrupt-Status()** wertet die Interrupt-Ursache aus.

VmeInit()
Initialisiert den VMEbus. Diese Funktion ist angepasst an den Industrierechner von Firma ROTEC. Für eine andere kundenseitige Hardware müssen eigene Funktionen zum Initialisieren geschrieben werden.

InitIk320()
Initialisiert die IK 320. Zunächst wird die Interrupt-Adresse von **INT_NR** unter **pOriginalInterruptVector** gespeichert. Anschließend wird die neue Interrupt-Funktion **NewInterruptRoutine** installiert, die alle IK-Interrupts behandelt. Dann setzt diese Funktion über **InitParams()** die Parameter und führt den „Power On Self Test" (POST) aus (VMEbus-Interrupt-Funktion $07). Der POST wurde erfolgreich abgeschlossen, falls die IK 320 den Status $0700 meldet.

DisplayMessage() und DisplayError()
Zeigen Meldungen und Fehler an, die die IK 320 über den Interrupt-Status meldet.

TraverseOverReferencemark()
Aktiviert die Auswertung der Referenzmarken über die VMEbus-Interrupt-Funktion $0008 für Achse 1 und $0009 für Achse 2. Anschließend müssen die Achsen über die Referenzmarken gefahren werden. Die IK 320 meldet per Interrupt-Status zurück, ob die Referenzmarken überfahren wurden: $0800 ist der Status „Referenzmarke Achse 1 wurde überfahren" und $0900 ist der Status „Referenzmarke Achse 2 wurde überfahren".

DisplayPositionValue()
Zeigt die Positionen der beiden Achsen 1 und 2 am Bildschirm an. Die Funktion ruft **SynchroPosTrigger()** zum synchronen Einspeichern. Der Positionswert wird in **Evaluate_IK_Interrupt_Status()** ermittelt. Bei Linearachsen muss dieser Wert mit der Signalperiode multipliziert werden (z.B. mit 0,002 mm), damit man eine Anzeige in mm erhält. Bei Winkelachsen muss die Multiplikation mit 360°/Signalperioden pro Umdrehung erfolgen.

CompensationRun()
Ermittelt die Korrekturwerte zur Kompensation von Abweichungen der Messsystem-Signale. Die Aufnahme der Korrekturwerte wird mit **MasterInterrupt()** und der Master-Interrupt-Funktion $0Bxx (Achse 1) oder $0Cxx (Achse 2) gestartet. Anschließend müssen die Achsen mit möglichst konstanter Geschwindigkeit bewegt werden. Die IK 320 meldet die erfolgreiche Aufnahme der Korrekturwerte über einen Interrupt mit dem Status $0Bxx (Achse 1) und $0Cxx (Achse 2).

CompensationOnOff()
Aktiviert die Kompensation der Messsystem-Signale über Parameter P06.

RestoreOldInterruptVector()
Bevor das Programm verlassen wird, wird mit RestoreOldInterruptVector() die ursprüngliche Interrupt-Adresse wieder installiert.

9.1 Die Header-Datei SAMPLE.H

```
/*----------------------SAMPLE.H-------------------------------------

  DR. JOHANNES HEIDENHAIN GmbH, Traunreut, Germany

  Header File for SAMPLE.C

  V 1.00
  September 1995
  ------------------------------------------------------------------*/

/*------------------------------------------------------------------
  Address of the VME address space A16 (DIP switch SI) and of the
  VME address space A24 (DIP switch SII).
  ------------------------------------------------------------------*/

#define DIP_SWITCH_SI    0xA1
#define DIP_SWITCH_SII   0x80
/*------------------------------------------------------------------
  Prototypes of functions
  ------------------------------------------------------------------*/

void  MainMenu(void);
```

9.2 Das Programm-Beispiel SAMPLE.C

```
/*----------------------------SAMPLE.C-------------------------------

  DR. JOHANNES HEIDENHAIN GmbH, Traunreut, Germany

  Sample for IK 320

  V 1.00
  September 1995
 ------------------------------------------------------------------------*/
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include "vmerotec.h"//Header file for ROTEC VME interface
#include "ik320.h"
#include "sample.h"

/*------------------------------------------------------------------------
  Global variables
  ------------------------------------------------------------------------*/

unsigned char extucErrorCode = 0, extucMessage = 0;

/*------------------------------------------------------------------------
  Variables for Compensation run
  ------------------------------------------------------------------------*/

unsigned short usIntervalOfCompPoints, usNumberOfCompPoints;

/*------------------------------------------------------------------------
  Main
  ------------------------------------------------------------------------*/

void main()
{
  clrscr();

  //Initialize VME interface (ROTEC specific functions)
  VmeInit();
  //Initialize IK 320
  InitIK320(DIP_SWITCH_SI, DIP_SWITCH_SII);

  //Display the main menu
  MainMenu();
  exit(0);
}//End of main

/*------------------------------------------------------------------------
  MainMenu
  ------------------------------------------------------------------------*/
void MainMenu()
{
  char cCharacter;

  _setcursortype(_NOCURSOR);

  do
   {
   if (extucErrorCode)
    {
    DisplayError();
    }
   if (extucMessage)
    {
    DisplayMessage();
    }
   clrscr();
   fflush(stdin);
   printf("1: Traverse over Reference Mark axis 1\n");
   printf("2: Traverse over Reference Mark axis 2\n");
   printf("3: Display Position Values\n");
   printf("4: Compensation Run axis 1\n");
   printf("5: Compensation Run axis 2\n");
```

```
   printf("6: Compensation On/Off\n");
   printf("0: End");

   do
    {
    if (extucErrorCode)
     {
     DisplayError();
     }
    if (extucMessage)
     {
     DisplayMessage();
     }
    }
   while (!(kbhit()));

   cCharacter=getch();

   switch(cCharacter)
    {
    case '1':
     TraverseOverReferenceMark(DIP_SWITCH_SI, DIP_SWITCH_SII, 1);
     break;
    case '2':
     TraverseOverReferenceMark(DIP_SWITCH_SI, DIP_SWITCH_SII, 2);
     break;
    case '3':
     DisplayPositionValue(DIP_SWITCH_SI, DIP_SWITCH_SII);
     break;
    case '4':
     clrscr();
     fflush(stdin);
     printf("\nNumber of Compensation Points  ");
     scanf("%d",&usNumberOfCompPoints);
     fflush(stdin);
     printf("\nInterval of Compensation Points  ");
     scanf("%d",&usIntervalOfCompPoints);
     CompensationRun (DIP_SWITCH_SI, DIP_SWITCH_SII,1,0,
                      usNumberOfCompPoints,
                      usIntervalOfCompPoints,1);
     break;
    case '5':
     clrscr();
     fflush(stdin);
     printf("\nNumber of Compensation Points   ");
     scanf("%d",&usNumberOfCompPoints);
     fflush(stdin);
     printf("\nInterval of Compensation Points  ");
     scanf("%d",&usIntervalOfCompPoints);
     CompensationRun(DIP_SWITCH_SI,
DIP_SWITCH_SII,2,0,usNumberOfCompPoints,
                     usIntervalOfCompPoints,0);
     break;
    case '6':
     CompensationOnOff(DIP_SWITCH_SI, DIP_SWITCH_SII);
     break;
    }
   }
 while(cCharacter!='0');
  //Set normal cursor again
  _setcursortype (_NORMALCURSOR);

  //InitIK320() sets a new interrupt vector. Therefore the old
  //interrupt vector has to be reinstalled.
  RestoreOldInterruptVector();
}//End of MainMenu
```

9.3 Die Header-Datei IK320.H

```
/*----------------------IK320.H-----------------------------------------

  DR. JOHANNES HEIDENHAIN GmbH, Traunreut, Germany

  Header File for the Driver Unit IK320.C

  V 1.00
  September 1995
  ----------------------------------------------------------------------*/

#define SUBDIVISION        4096
#define IK_BASE_ADDRESS    0xC00000L
#define INTERPOLATION_BITS 12

/*The ROTEC VMEbus interface converts VME interrupts to DOS
  interrupt IRQ15. The following defines are addresses of
  the DOS interrupt controllers.*/
#define INTC1A0   0x20
#define INTC1A1   0x21
#define INTC2A0   0xa0
#define INTC2A1   0xa1
#define INT_NR             0x77        //Dos Interrupt IRQ15
#define INT_MASK          ~0x80        //Interrupt mask IRQ15
#define EOI                0x20        //End of Interrupt command

/*----------------------------------------------------------------------
  Macro to switch VME to A24 memory space;
  ROTEC specific code
  ----------------------------------------------------------------------*/
#define SWITCH_VME_TO_A24_ADDRESS_SPACE(switch) outport(ADR_REG,\
(short)((IK_BASE_ADDRESS + switch * 0x40001) >> 8) & 0xFF80)

/*----------------------------------------------------------------------
  Macro to switch VME to A16 memory space;
  ROTEC specific code
  ----------------------------------------------------------------------*/

#define SWITCH_VME_TO_A16_ADDRESS_SPACE(switch) outport(ADR_REG,\
((((short) ((switch & 0xE0) >> 5) * 0x2000) & 0x8000) >> 8 ) |0xFC00)

/*----------------------------------------------------------------------
  Macro to calculate the IK address.
  The code <& 0xFFFF) | 0x8000> is a ROTEC specific
  address modification
  ----------------------------------------------------------------------*/

#define CALCULATE_IK_ADDRESS(switch) \
 (short)(((IK_BASE_ADDRESS + switch * 0x40001)  & 0xFFFF) | 0x8000)

/*----------------------------------------------------------------------
  Macro to calculate the IK group address.
  The code <| 0x8000> is a ROTEC specific
  address modification
  ----------------------------------------------------------------------*/

#define  CALCULATE_BAS_ADR_GROUP(switch) \
 ((short)((switch & 0xE0) >> 5) * 0x2000) | 0x8000

/*----------------------------------------------------------------------
  Addresses of Parameters
  ----------------------------------------------------------------------*/
#define  PAR_01_1    0x0102  //Counting direction axis 1
#define  PAR_01_2    0x0103  //Counting direction axis 2
#define  PAR_01_3    0x0104  //Counting direction for axis comb.

#define  PAR_02_1    0x0106  //Definition axis 1
#define  PAR_02_2    0x0107  //Definition axis 2
#define  PAR_02_3    0x0108  //Definition for axis combination

#define  PAR_03      0x010A  //Bits of subdivision

#define  PAR_04_1    0x010C  //Ref. mark spacing axis 1
#define  PAR_04_2    0x010E  //Ref. mark spacing axis 2
```

```
#define  PAR_05_1    0x0110  //Signal periods per rev. axis 1
#define  PAR_05_2    0x0114  //Signal periods per rev. axis 2
#define  PAR_05_3    0x0118  //Signal periods per rev. for axis comb.

#define  PAR_06_1    0x011C  //Compensation on/off axis 1
#define  PAR_06_2    0x011D  //Compensation on/off axis 2

#define  PAR_07_1    0x011E  //Compensation start axis 1
#define  PAR_07_2    0x0122  //Compensation start axis 2

#define  PAR_08_1    0x0126  //Number of comp. points axis 1
#define  PAR_08_2    0x0128  //Number of comp. points axis 2

#define  PAR_09_1    0x012A  //Interval comp. points axis 1
#define  PAR_09_2    0x012C  //Interval comp. points axis 2

#define  PAR_10      0x012E  //Latch enable

#define  PAR_19_1    0x0130  //Ref offset axis 1
#define  PAR_19_2    0x0134  //Ref offset axis 2

#define  PAR_21      0x0138  //Axis combination

#define  PAR_30_1    0x013A  //Compensation run axis 1(axis 2)
#define  PAR_30_2    0x013B  //Compensation run axis 2

#define  PAR_70_1    0x013C  //Preset external setting axis 1
#define  PAR_70_2    0x0142  //Preset external setting axis 2
#define  PAR_70_3    0x0148  //Preset external setting for axis comb.

#define  PAR_71_1    0x014E  //Preset Master setting axis 1
#define  PAR_71_2    0x0154  //Preset Master setting axis 2
#define  PAR_71_3    0x015A  //Preset Master setting for axis comb.

#define  PAR_72_1    0x0160  //Axis offset axis 1
#define  PAR_72_2    0x0166  //Axis offset axis 2
#define  PAR_72_3    0x016C  //Axis offset for axis combination

#define  PAR_80_1    0x0172  //External function 1
#define  PAR_80_2    0x0173  //External function 2

#define  PAR_81      0x0100  // Master interrupt function


/*---------------------------------------------------------------------
  Status and Data of IK
  ---------------------------------------------------------------------*/
#define  POS_X1_1       0x0000 //Position value axis 1
#define  POS_X1_2       0x0002
#define  POS_X1_3       0x0004
#define  STAT_X1        0x0006 //Status axis 1
#define  TM_X1          0x0007 //Transfer marker axis 1

#define  POS_X2_1       0x0008 //Position value axis 2
#define  POS_X2_2       0x000A
#define  POS_X2_3       0x000C
#define  STAT_X2        0x000E //Status axis 2
#define  TM_X2          0x000F //Transfer marker axis 2

#define  POS_COMB_1     0x0010 //Pos. value comb. axis
#define  POS_COMB_2     0x0012
#define  POS_COMB_3     0x0014
#define  STAT_COMB      0x0016 //Status combined axis
#define  TM_COMB        0x0017 //Transfer marker comb. axis

#define  INTSTAT        0x0018 //IK interrupt status

#define  CRC_NOML       0x001A //Nominal CRC sum EPROM
#define  CRC_ACTL       0x001C //Actual CRC sum EPROM

#define  EPR1_NOML      0x001E //Nominal CRC sum EPROM 1
#define  EPR1_ACTL      0x0022 //Actual CRC sum EPROM 1
#define  EPR2_NOML      0x0026 //Nominal CRC sum EPROM 2
```

```
#define  EPR2_ACTL      0x002A //Actual CRC SUM EPROM 2

#define  HRDVERS        0x002E //Hardware version
#define  SFTVERS        0x0030 //Software version

#define  SYSST1         0x003C //System status 1
#define  SYSST2         0x003D //System status 2

#define  PRES1          0x003E //Preset axis 1
#define  PRES2          0x0044 //Preset axis 2
#define  PRES_COMB      0x004A //Preset axis combination


/*---------------------------------------------------------------------
  Prototypes for Functions
  ---------------------------------------------------------------------*/
void far interrupt NewInterruptRoutine(void);
void RestoreOldInterruptVector(void);
void Read_IK_InterruptStatus(unsigned char);
void Evaluate_IK_InterruptStatus(unsigned char);

void MasterInterrupt(unsigned char, unsigned char, unsigned short);

void InitIK320 (unsigned char, unsigned char);
void InitParams (unsigned char);
void SetParam (unsigned char, unsigned short, long, short);

void TraverseOverReferenceMark(unsigned char, unsigned char,
                               unsigned char);

void DisplayPositionValue(unsigned char, unsigned char);
void SynchroPosTrigger(unsigned char, unsigned char);

void CompensationRun(unsigned char, unsigned char,
                     unsigned char, unsigned char,
                     unsigned short, unsigned short, short);
void CompensationOnOff(unsigned char, unsigned char);

void DisplayMessage(void);
void DisplayError(void);
```

9.4 Die Funktionen in IK320.C

```
/*----------------------------IK320.C---------------------------------

  DR. JOHANNES HEIDENHAIN GmbH, Traunreut, Germany

  Driver Unit for IK 320

  V 1.00
  September 1995
  ---------------------------------------------------------------------*/
#include <stdlib.h>
#include <conio.h>
#include <stdio.h>
#include <dos.h>
#include <process.h>
#include "vmerotec.h"//Header file for ROTEC VME interface
#include "ik320.h"
#include "sample.h"


/*---------------------------------------------------------------------
  Definition of an global union for IK interrupt status word
  ---------------------------------------------------------------------*/

static struct TWOBYTES  {unsigned char uc0, uc1;};
static struct ONEWORD   {unsigned short us;};
static union  WORDBYTE  {struct TWOBYTES tb;
                         struct ONEWORD ow;}stawbStatus;

/*---------------------------------------------------------------------
  Global variables
  ---------------------------------------------------------------------*/
```

```
static unsigned char staucDIP_Switch_II = DIP_SWITCH_SII;
static unsigned char staucAxis1WasRead, staucAxis2WasRead;
static unsigned char staucAxisComWasRead;
static unsigned char staucREF1Crossed, staucREF2Crossed;
static unsigned char staucInterruptFinished;
static double        stadPositionValue1, stadPositionValue2;
static double        stadPositionValueCom;

extern unsigned char extucErrorCode, extucMessage;
/*---------------------------------------------------------------------
  pOriginalInterruptVector

  Specifies a vector for the original interrupt.
  ---------------------------------------------------------------------*/
void (interrupt far *pOriginalInterruptVector)();


/*---------------------------------------------------------------------
  NewInterruptRoutine

  This function specifies the interrupt routine for IK interrupt
  ---------------------------------------------------------------------*/
void interrupt far NewInterruptRoutine(void)
{
  unsigned short usAddress;

  outp(INTC2A1,inp(INTC2A1) | ~INT_MASK);   //Disable IRQ15

  //Reset DOS-interrupt line
  outp(INTC2A0,EOI);
  outp(INTC1A0,EOI);

  _enable();               //Enable DOS interrupt

  usAddress = CALCULATE_IK_ADDRESS(staucDIP_Switch_II);

  SWITCH_VME_TO_A24_ADDRESS_SPACE(staucDIP_Switch_II);
  Read_IK_InterruptStatus(staucDIP_Switch_II);
  Evaluate_IK_InterruptStatus(staucDIP_Switch_II);

  //ROTEC specific code: Reset VME interrupt line
  int_eoi(IACK3);

  outp(INTC2A1,inp(INTC2A1) & INT_MASK);  //enable IRQ15

  usAddress = CALCULATE_IK_ADDRESS(staucDIP_Switch_II);
  outport (usAddress + INTSTAT, 0x0000);

  staucInterruptFinished = 1;                       //Interrupt finished
}//End NewInterruptRoutine

/*---------------------------------------------------------------------
  RestoreOldInterruptVector

  This function restores the old interrupt vector.
  ---------------------------------------------------------------------*/
void RestoreOldInterruptVector(void)
{
  //Disable hardware interrupts
  _disable();
  //Restore original interrupt vector
  _dos_setvect(INT_NR,pOriginalInterruptVector);
  //Enable hardware interrupts
  _enable();
}//End RestoreOldInterruptVector

/*---------------------------------------------------------------------
  Read_IK_InterruptStatus

  This function reads the interrupt status word of the IK.
  ---------------------------------------------------------------------*/
void Read_IK_InterruptStatus(unsigned char ucDIP_Switch_II)
{
  short usAddress;
```

```
  //Calculate address
  usAddress = CALCULATE_IK_ADDRESS(ucDIP_Switch_II);
  //Read status
  stawbStatus.ow.us = inport (usAddress + INTSTAT);
}//End Read_IK_InterruptStatus

/*---------------------------------------------------------------------
  Evaluate_IK_InterruptStatus

  When the IK sends an interrupt to the master, the cause of the
  interrupt is shown in the interrupt status word. This function
  evaluates the interrupt status word.
  ---------------------------------------------------------------------*/
void Evaluate_IK_InterruptStatus(unsigned char ucDIP_Switch_II)
{

  unsigned short usDummy, usAddress;
  long lDummy;

  usAddress = CALCULATE_IK_ADDRESS(ucDIP_Switch_II);

  switch (stawbStatus.tb.uc1)
    {
   case 0x00:
    break;

   case 0x01:
    usDummy=inport (usAddress + POS_X1_1);
    lDummy=(long)(usDummy)<<16;
    usDummy=inport (usAddress + POS_X1_2);
    lDummy+=usDummy;
    stadPositionValue1=(double)lDummy;
    usDummy=inport (usAddress + POS_X1_3);
    stadPositionValue1+=(double)usDummy/(SUBDIVISION*16.);
    outportb (usAddress + TM_X1, (char)0x00);
    usDummy=inportb  (usAddress + STAT_X1);
    staucAxis1WasRead = 1;
    break;

   case 0x02:
    usDummy=inport (usAddress + POS_X2_1);
    lDummy=(long)(usDummy)<<16;
    usDummy=inport (usAddress + POS_X2_2);
    lDummy+=usDummy;
    stadPositionValue2=(double)lDummy;
    usDummy=inport (usAddress + POS_X2_3);
    stadPositionValue2+=(double)usDummy/(SUBDIVISION*16.);
    outportb (usAddress + TM_X2, (char)0x00);
    usDummy=inportb  (usAddress + STAT_X2);
    staucAxis2WasRead = 1;
    break;

   case 0x03:
    usDummy=inport (usAddress + POS_COMB_1);
    lDummy=(long)(usDummy)<<16;
    usDummy=inport (usAddress + POS_COMB_2);
    lDummy+=usDummy;
    stadPositionValueCom=(double)lDummy;
    usDummy=inport (usAddress + POS_COMB_3);
    stadPositionValueCom+=(double)usDummy/(SUBDIVISION*16.);
    outportb (usAddress + TM_COMB, (char)0x00);
    usDummy=inportb  (usAddress + STAT_COMB);
    staucAxisComWasRead = 1;
    break;

   case 0x04:
    if (stawbStatus.tb.uc0 & 0x01)
     {
     extucErrorCode = 0x01;
     }
    if (stawbStatus.tb.uc0 & 0x02)
     {
     extucErrorCode = 0x02;
     }
```

```
 break;

case 0x05:
 if (stawbStatus.tb.uc0 & 0x01)
  {
  extucErrorCode = 0x03;
  }
 if (stawbStatus.tb.uc0 & 0x02)
  {
  extucErrorCode = 0x04;
  }
 break;

case 0x06:
 if (stawbStatus.tb.uc0 == 1)
  outportb (usAddress + TM_X1, (char)0x00);//Reset TM_X1
 else if (stawbStatus.tb.uc0 == 2)
  outportb (usAddress + TM_X2, (char)0x00);//Reset TM_X2
 else
outportb (usAddress + TM_COMB, (char)0x00);//Reset TM_COMB
 break;

case 0x07:
 extucMessage = 0x01;
 if (stawbStatus.tb.uc0 & 0x01)
  {
  extucErrorCode = 0x05;
  }
 if (stawbStatus.tb.uc0 & 0x02)
  {
  extucErrorCode = 0x06;
  }
 if (stawbStatus.tb.uc0 & 0x04)
  {
  extucErrorCode = 0x07;
  }
 if (stawbStatus.tb.uc0 & 0x08)
  {
  extucErrorCode = 0x08;
  }
 if (stawbStatus.tb.uc0 & 0x10)
  {
  extucErrorCode = 0x09;
  }
 if (stawbStatus.tb.uc0 & 0x20)
  {
  extucErrorCode = 0x10;
  }
 if (stawbStatus.tb.uc0 & 0x40)
  {
  extucErrorCode = 0x11;
  }
 break;

case 0x08:
 extucMessage = 0x02;
 staucREF1Crossed = 1;
 break;

case 0x09:
 extucMessage = 0x03;
 staucREF2Crossed = 1;
 break;

case 0x0A:
if (stawbStatus.tb.uc0 == 0x01)
   {
   extucErrorCode = 0x12;
   }
 break;

case 0x0B:
 switch (stawbStatus.tb.uc0)
  {
```

```
  case 0x00:
   extucMessage = 0x04;
   break;
  case 0x01:
   extucErrorCode = 0x13;
   break;
  case 0x02:
   extucErrorCode = 0x14;
   break;
  case 0x03:
   extucErrorCode = 0x15;
   break;
  case 0x04:
   extucErrorCode = 0x16;
   break;
  case 0x05:
   extucErrorCode = 0x17;
   break;
  case 0x06:
   extucErrorCode = 0x18;
   break;
   case 0x10:
   extucMessage = 0x05;
   break;
  default:
    extucErrorCode = 0x99;
  }
 break;

case 0x0C:
 switch (stawbStatus.tb.uc0)
  {
  case 0x00:
   extucMessage = 0x06;
   break;
  case 0x01:
   extucErrorCode = 0x19;
   break;
  case 0x02:
   extucErrorCode = 0x20;
   break;
  case 0x03:
   extucErrorCode = 0x21;
   break;
  case 0x04:
   extucErrorCode = 0x22;
   break;
  case 0x05:
   extucErrorCode = 0x23;
   break;
  case 0x06:
   extucErrorCode = 0x24;
   break;
  case 0x10:
   extucMessage = 0x07;
   break;
  default:
   extucErrorCode = 0x99;
  }
 break;

case 0x0E:
 if (stawbStatus.tb.uc0 == 0x01)
 {
 extucErrorCode = 0x25;
 }
 if (stawbStatus.tb.uc0 == 0x02)
 {
 extucErrorCode = 0x26;
 }
 break;

case 0x0F:
 if (stawbStatus.tb.uc0 == 0x01)
```

```
     {
     extucErrorCode = 0x27;
     }
    if (stawbStatus.tb.uc0 == 0x02)
     {
     extucErrorCode = 0x28;
     }
    if (stawbStatus.tb.uc0 == 0x03)
     {
     extucErrorCode = 0x29;
     }
    break;

   case 0x10:
    extucMessage = 0x08;
    break;

   case 0x11:
    extucMessage = 0x09;
    break;

   case 0xFD:
    switch (stawbStatus.tb.uc0)
     {
   case 0x01:
      extucErrorCode = 0x30;
      break;
     case 0x02:
      extucErrorCode = 0x31;
      break;
     default:
      extucErrorCode = 0x99;
     }

   case 0xFE:
    extucErrorCode = 0x32;
    break;

   case 0xFF:
    extucErrorCode = 0x33;
    break;

   default:
    extucErrorCode = 0x99;
   }//End switch - stawbStatus.tb.uc1
}//End Evaluate_IK_InterruptStatus

/*---------------------------------------------------------------------
  MasterInterrupt

  This function sets parameter P81 and generates a master interrupt
  ---------------------------------------------------------------------*/
void MasterInterrupt(unsigned char ucDIP_Switch_I,
                     unsigned char ucDIP_Switch_II,
                     unsigned short usFunction)
{
  short usAddress, sBasAdrGroup;
  //calculate synchronous latch interrupt address
  sBasAdrGroup = CALCULATE_BAS_ADR_GROUP(ucDIP_Switch_I);
  usAddress = CALCULATE_IK_ADDRESS(ucDIP_Switch_II);

  outport (usAddress + PAR_81,usFunction);

  _disable();
  SWITCH_VME_TO_A16_ADDRESS_SPACE(ucDIP_Switch_I);
  //Execute master interrupt
  outportb ((short)(sBasAdrGroup + ((ucDIP_Switch_I & 0x1F) * 2)),
           (char)0x00);
  SWITCH_VME_TO_A24_ADDRESS_SPACE(ucDIP_Switch_II);
  _enable();

}//End MasterInterrupt

/*---------------------------------------------------------------------
```

InitIk320

```
  This function initializes the IK.
  ------------------------------------------------------------------*/
void InitIK320 (unsigned char ucDIP_Switch_I,
                unsigned char ucDIP_Switch_II)
{
  printf("\nInitialize IK 320  %02x\n",ucDIP_Switch_II);

  //Disable hardware interrupts
  _disable();

  //Save old interrupt vector
  pOriginalInterruptVector = _dos_getvect(INT_NR);

  //Set new interrupt vector
  _dos_setvect(INT_NR,NewInterruptRoutine);

  //Set interrupt controller mask
  outp(INTC2A1,inp(INTC2A1) & INT_MASK);
  outp(INTC1A1,inp(INTC1A1) & ~0x04);

  //Set interrupt controller to End of Interrupt
  outp(INTC2A0,EOI);
  outp(INTC1A0,EOI);

  //Enable hardware interrupts
  _enable();

  SWITCH_VME_TO_A24_ADDRESS_SPACE(ucDIP_Switch_II);

  InitParams (ucDIP_Switch_II);//Set the parameters
  staucInterruptFinished = 0;
  MasterInterrupt(ucDIP_Switch_I, ucDIP_Switch_II, 0x0007);
  do
  {
   if (extucErrorCode)
    {
    DisplayError();
    return;
    }
   if (extucMessage)
    {
    DisplayMessage();
    }
  }
  while (staucInterruptFinished == 0);//Wait for interrupt
}//End InitIk320


/*------------------------------------------------------------------
  InitParams

  This function initializes the Parameters.
  ------------------------------------------------------------------*/
void InitParams (unsigned char ucDIP_Switch_II)
{
  SetParam (ucDIP_Switch_II, PAR_01_1, 0x00,0);//Count. direct. axis 1
  SetParam (ucDIP_Switch_II, PAR_01_2, 0x00,0);//Count. direct. axis 2
  SetParam (ucDIP_Switch_II, PAR_01_3, 0x00,0);//Counting direction
                                               //axis combination

  SetParam (ucDIP_Switch_II, PAR_02_1, 0x01,0);//Definition axis 1
  SetParam (ucDIP_Switch_II, PAR_02_2, 0x01,0);//Definition axis 2
  SetParam (ucDIP_Switch_II, PAR_02_3, 0x01,0);//Definition axis comb.

  //Number of bits for subdivision
  SetParam (ucDIP_Switch_II, PAR_03  , INTERPOLATION_BITS,0);

  SetParam (ucDIP_Switch_II, PAR_04_1, 0x0000,0);//Ref. mark spacing 1
  SetParam (ucDIP_Switch_II, PAR_04_2, 0x0000,0);//Ref. mark spacing 2

  SetParam (ucDIP_Switch_II, PAR_05_1, 0x00008CA0,0);//Signal per. 1
  SetParam (ucDIP_Switch_II, PAR_05_2, 0x00000800,0);//Signal per. 2
```

```
  SetParam (ucDIP_Switch_II, PAR_05_3, 0x00000000,0);//Signal periods
                                                      //axis combination

  SetParam (ucDIP_Switch_II, PAR_06_1, 0x00,0);//Compensation on/off 1
  SetParam (ucDIP_Switch_II, PAR_06_2, 0x00,0);//Compensation on/off 2

  SetParam (ucDIP_Switch_II, PAR_07_1, 0x00000000,0);//Comp. start 1
  SetParam (ucDIP_Switch_II, PAR_07_2, 0x00000000,0);//Comp. start 2

  SetParam (ucDIP_Switch_II, PAR_08_1, 0x0100,0);//Nr. of comp. pts. 1
  SetParam (ucDIP_Switch_II, PAR_08_2, 0x0100,0);//Nr. of comp. pts. 2

  SetParam (ucDIP_Switch_II, PAR_09_1, 0x0010,0);//Interv. of c. pts 1
  SetParam (ucDIP_Switch_II, PAR_09_2, 0x0010,0);//Interv. of c. pts 2

  SetParam (ucDIP_Switch_II, PAR_10  , 0x00,0);//Latch enable

  SetParam (ucDIP_Switch_II, PAR_19_1, 0x00000000,0);//Ref. offs. 1
  SetParam (ucDIP_Switch_II, PAR_19_2, 0x00000000,0);//Ref. offs. 2

  SetParam (ucDIP_Switch_II, PAR_21  , 0x00,0);//Axis combination

                             //Compensation run
  SetParam (ucDIP_Switch_II, PAR_30_1, 0x01,0);//Dir.(axis 1) &
                               //freq.(axis 1+2)
  SetParam (ucDIP_Switch_II, PAR_30_2, 0x00,0);//Direction axis 2

  SetParam (ucDIP_Switch_II, PAR_70_1, 0x00000000,0);//Ext. preset 1
  SetParam (ucDIP_Switch_II, PAR_70_2, 0x00000000,0);//Ext. preset 2
  SetParam (ucDIP_Switch_II, PAR_70_3, 0x00000000,0);//Ext. preset
                                 //axis combination

  SetParam (ucDIP_Switch_II, PAR_71_1, 0x00000000,0);//Bus preset 1
  SetParam (ucDIP_Switch_II, PAR_71_2, 0x00000000,0);//Bus preset 2
  SetParam (ucDIP_Switch_II, PAR_71_3, 0x00000000,0);//Bus preset
                                   //axis combination

  SetParam (ucDIP_Switch_II, PAR_72_1, 0x00000000,0);//Axis offset 1
  SetParam (ucDIP_Switch_II, PAR_72_2, 0x00000000,0);//Axis offset 2
  SetParam (ucDIP_Switch_II, PAR_72_3, 0x00000000,0);//Axis offset
                                   //axis combination

  SetParam (ucDIP_Switch_II, PAR_80_1, 0x00,0);//External function 1
  SetParam (ucDIP_Switch_II, PAR_80_2, 0x00,0);//External function 2

}//End InitParams


/*---------------------------------------------------------------------
  SetParam

  With this function the master sets the parameters.
  ---------------------------------------------------------------------*/
void SetParam (unsigned char ucDIP_Switch_II,
         unsigned short usOffsetAddress, long lData,short sData)
{
  unsigned short usBaseAddress;
  usBaseAddress = CALCULATE_IK_ADDRESS(ucDIP_Switch_II);
  switch (usOffsetAddress)
 {
  case PAR_01_1:
 outportb (usBaseAddress+usOffsetAddress, (char) lData);
 break;
  case PAR_01_2:
 outportb (usBaseAddress+usOffsetAddress, (char) lData);
 break;
  case PAR_01_3:
 outportb (usBaseAddress+usOffsetAddress, (char) lData);
 break;
  case PAR_02_1:
 outportb (usBaseAddress+usOffsetAddress, (char) lData);
 break;
  case PAR_02_2:
 outportb (usBaseAddress+usOffsetAddress, (char) lData);
```

```
break;
  case PAR_02_3:
outportb (usBaseAddress+usOffsetAddress, (char) lData);
break;
  case PAR_03  :
outport (usBaseAddress+usOffsetAddress, (short)  lData);
break;
  case PAR_04_1:
outport (usBaseAddress+usOffsetAddress, (short)  lData);
break;
  case PAR_04_2:
outport (usBaseAddress+usOffsetAddress, (short)  lData);
break;
  case PAR_05_1:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
break;
  case PAR_05_2:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
break;
  case PAR_05_3:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
break;
  case PAR_06_1:
outportb (usBaseAddress+usOffsetAddress, (char) lData);
break;
  case PAR_06_2:
outportb (usBaseAddress+usOffsetAddress, (char) lData);
break;
  case PAR_07_1:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
break;
  case PAR_07_2:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
break;
  case PAR_08_1:
outport (usBaseAddress+usOffsetAddress, (short)  lData);
break;
  case PAR_08_2:
outport (usBaseAddress+usOffsetAddress, (short)  lData);
break;
  case PAR_09_1:
outport (usBaseAddress+usOffsetAddress, (short)  lData);
break;
  case PAR_09_2:
outport (usBaseAddress+usOffsetAddress, (short)  lData);
break;
  case PAR_10  :
outportb (usBaseAddress+usOffsetAddress, (char) lData);
break;
  case PAR_19_1:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
break;
  case PAR_19_2:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
break;
  case PAR_21  :
outportb (usBaseAddress+usOffsetAddress, (char) lData);
break;
  case PAR_30_1:
outportb (usBaseAddress+usOffsetAddress, (char) lData);
break;
  case PAR_30_2:
outportb (usBaseAddress+usOffsetAddress, (char) lData);
break;
  case PAR_70_1:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
```

```
outport (usBaseAddress+usOffsetAddress+4, sData);
break;
  case PAR_70_2:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
outport (usBaseAddress+usOffsetAddress+4, sData);
break;
  case PAR_70_3:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
outport (usBaseAddress+usOffsetAddress+4, sData);
break;
  case PAR_71_1:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
outport (usBaseAddress+usOffsetAddress+4, sData);
break;
  case PAR_71_2:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
outport (usBaseAddress+usOffsetAddress+4, sData);
break;
  case PAR_71_3:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
outport (usBaseAddress+usOffsetAddress+4, sData);
break;
  case PAR_72_1:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
outport (usBaseAddress+usOffsetAddress+4, sData);
break;
  case PAR_72_2:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
outport (usBaseAddress+usOffsetAddress+4, sData);
break;
  case PAR_72_3:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
outport (usBaseAddress+usOffsetAddress+4, sData);
break;
  case PAR_80_1:
outportb (usBaseAddress+usOffsetAddress, (char) lData);
break;
  case PAR_80_2:
outportb (usBaseAddress+usOffsetAddress, (char) lData);
break;
  default:
   gotoxy(1,23);
   puts ("Error: Wrong parameter number");
  }//End switch (usOffsetAddress)
}//End SetParam

/*---------------------------------------------------------------------
  TraverseOverReferenceMark

  This function causes the operator to traverse over the reference
  mark.
  ---------------------------------------------------------------------*/
void TraverseOverReferenceMark(unsigned char ucDIP_Switch_I,
                      unsigned char ucDIP_Switch_II,
                                unsigned char ucAxis)
{
  clrscr();
  switch (ucAxis)
  {
  case 1:
   printf("\nCross over reference mark of axis 1\n");
   staucREF1Crossed = 0;
   MasterInterrupt(ucDIP_Switch_I, ucDIP_Switch_II, 0x0008);
   do
    {
    if (extucErrorCode)
```

```
     {
     DisplayError();
     break;
     }
    if (extucMessage)
     {
     DisplayMessage();
     }
    }
   while (!(staucREF1Crossed));
   break;
  case 2:
   printf("Cross over reference mark of axis 2\n");
   staucREF2Crossed = 0;
   MasterInterrupt(ucDIP_Switch_I, ucDIP_Switch_II, 0x0009);
   do
    {
     if (extucErrorCode)
      {
      DisplayError();
      break;
      }
     if (extucMessage)
      {
      DisplayMessage();
      }
     }
     while (!(staucREF2Crossed));
   break;
  default:
   gotoxy (1,23);
   puts ("Error: Wrong axis number");
  }
}//End TraverseOverReferenceMark

/*---------------------------------------------------------------------
  SynchroPosTrigger

  This function triggers axis 1 and axis 2 synchronously
  ---------------------------------------------------------------------*/
void SynchroPosTrigger(unsigned char ucDIP_Switch_I,
             unsigned char ucDIP_Switch_II)
{
  short sBasAdrGroup;
  //calculate synchronous latch interrupt address
  sBasAdrGroup = CALCULATE_BAS_ADR_GROUP(ucDIP_Switch_I);

  _disable();
  SWITCH_VME_TO_A16_ADDRESS_SPACE(ucDIP_Switch_I);
  outportb (sBasAdrGroup, (char)0x00);//Synchronous latch
  SWITCH_VME_TO_A24_ADDRESS_SPACE(ucDIP_Switch_II);
  _enable();
}//End SynchroPosTrigger
/*---------------------------------------------------------------------
  DisplayPositionValue

  This function displays the actual position
  ---------------------------------------------------------------------*/
void DisplayPositionValue(unsigned char ucDIP_Switch_I,
                 unsigned char ucDIP_Switch_II)
{
  clrscr();
  printf("\n\n");
  do
  {
  SynchroPosTrigger(ucDIP_Switch_I, ucDIP_Switch_II);

   do
    {
    if (extucErrorCode)
     {
     DisplayError();
     return;
     }
```

```
    }
   while (!(staucAxis1WasRead && staucAxis2WasRead));

   printf("\r\t%10.4f\t%10.4f",stadPositionValue1,stadPositionValue2);
   staucAxis1WasRead = 0;
   staucAxis2WasRead = 0;
  }
  while (!kbhit());getch();
}//End DisplayPositionValue

/*---------------------------------------------------------------------
  CompensationRun

  This function starts the compensation run
  ---------------------------------------------------------------------*/
void CompensationRun(unsigned char ucDIP_Switch_I,
             unsigned char ucDIP_Switch_II,
             unsigned char ucAxis,
             unsigned char ucCompensationStart,
             unsigned short usNumberOfCompPoints,
             unsigned short usIntervalOfCompPoints,
             short sDirectionAndFrequency)
{
  switch(ucAxis)
 {
 case 1: SetParam (ucDIP_Switch_II, PAR_06_1, 0x01,0);//Comp. on/off
  SetParam (ucDIP_Switch_II, PAR_07_1, ucCompensationStart,0);
  SetParam (ucDIP_Switch_II, PAR_08_1, usNumberOfCompPoints,0);
  SetParam (ucDIP_Switch_II, PAR_09_1, usIntervalOfCompPoints,0);
  SetParam (ucDIP_Switch_II, PAR_30_1, sDirectionAndFrequency,0);
  break;
 case 2: SetParam (ucDIP_Switch_II, PAR_06_2, 0x01,0);//Comp. on/off
  SetParam (ucDIP_Switch_II, PAR_07_2, ucCompensationStart,0);
  SetParam (ucDIP_Switch_II, PAR_08_2, usNumberOfCompPoints,0);
  SetParam (ucDIP_Switch_II, PAR_09_2, usIntervalOfCompPoints,0);
  SetParam (ucDIP_Switch_II, PAR_30_2, sDirectionAndFrequency,0);
  break;
 default:
    gotoxy (1,23);
  puts ("Error: The input value for axis must be 1 or 2\n");
    return;
 }

   staucInterruptFinished = 0;
   //Update parameter
   MasterInterrupt(ucDIP_Switch_I, ucDIP_Switch_II, 0x000a);

   do
   if (extucErrorCode)
    {
    DisplayError();
    return;
    }
    while (staucInterruptFinished == 0);//Wait for Interrupt

   clrscr();
  printf ("\nRecord compensation values axis %2d \n\n\
             - press any key to start\n\n\n\n", ucAxis);
  do
   {
   SynchroPosTrigger(ucDIP_Switch_I, ucDIP_Switch_II);

   do
     {
     if (extucErrorCode)
      {
      DisplayError();
      return;
      }
     }
    while (!(staucAxis1WasRead && staucAxis2WasRead));

    printf("\r\t%10.4f\t%10.4f",stadPositionValue1,stadPositionValue2);
    staucAxis1WasRead =0;
```

```
    staucAxis2WasRead =0;
    }
  while (!kbhit());getch();
   clrscr();

   switch(ucAxis)
   {
    //Compensation run axis 1
   case 1:
    MasterInterrupt(ucDIP_Switch_I, ucDIP_Switch_II, 0x000b);
    printf ("\nRecord of compensation points for axis 1 started\n\n");
    printf ("- Start axis\n");
   break;
   //Compensation run axis 2
   case 2:
    MasterInterrupt(ucDIP_Switch_I, ucDIP_Switch_II, 0x000c);
    printf ("\nRecord of compensation points for axis 2 started\n\n");
    printf ("- Start axis\n");
   break;
  }
  do
   {
   if (extucErrorCode)
    {
    DisplayError();
    return;
    }
   }
  while (!(extucMessage));
  clrscr();
  DisplayMessage();
  do
   {
   if (extucErrorCode)
    {
    DisplayError();
    return;
    }
   }
  while (!(extucMessage));
  clrscr();
  DisplayMessage();
  do
   {
   if (extucErrorCode)
    {
    DisplayError();
    return;
    }
   }
  while (!(kbhit()));
  getch();
}//End CompensationRun

/*---------------------------------------------------------------------
  CompensationOnOff

  This function switches the signal compensation on or off.
  ---------------------------------------------------------------------*/
void CompensationOnOff(unsigned char ucDIP_Switch_I,
                       unsigned char ucDIP_Switch_II)
{
  char cCharacter,cAxis;
  fflush(stdin);
  printf("\n Axis 1 or 2?   ");
  do
   {
   if (extucErrorCode)
    {
    DisplayError();
    return;
    }
   }
  while (!kbhit());
```

```
  cAxis=getche();

  fflush(stdin);
  printf("\n Compensation on = c / off = o   ");
  do
   {
   if (extucErrorCode)
    {
    DisplayError();
    return;
    }
   }
  while (!kbhit());

  cCharacter=getche();

  switch(cAxis)
   {
   //Switch compensation on/off axis 1
   case '1':
    if (cCharacter=='c')
     SetParam (ucDIP_Switch_II, PAR_06_1, 0x01,0);
    if (cCharacter=='o')
     SetParam (ucDIP_Switch_II, PAR_06_1, 0x00,0);
    break;
   //Switch compensation on/off axis 2
   case '2':
    if (cCharacter=='c')
    SetParam (ucDIP_Switch_II, PAR_06_2, 0x01,0);
    if (cCharacter=='o')
    SetParam (ucDIP_Switch_II, PAR_06_2, 0x00,0);
    break;
   default:
    gotoxy (1,23);
    puts ("Error: Wrong character");
   }
   //Update parameter
   MasterInterrupt(ucDIP_Switch_I, ucDIP_Switch_II, 0x000a);
}//End CompensationOnOff

/*---------------------------------------------------------------------
  DisplayMessage

  This function displays the messages of the IK interrupt status.
  ---------------------------------------------------------------------*/

void DisplayMessage(void)
{
  gotoxy (1,20);
  switch (extucMessage)
   {
   case 0x01:
    puts ("POST concluded\n");
    break;
   case 0x02:
    puts ("REF axis 1 was crossed over\n");
    break;
   case 0x03:
    puts ("REF axis 2 was crossed over\n");
    break;
   case 0x04:
    puts("Compensation run ended axis 1 - Press any key\n");
    break;
   case 0x05:
    puts ("Record of Compensation Points ended axis 1\
           - Please wait\n");
    break;
   case 0x06:
    puts ("Compensation run ended axis 2 - Press any key\n");
    break;
   case 0x07:
    puts ("Record of Compensation Points ended axis 2\
           - Please wait\n");
```

```
    break;
   case 0x08:
    puts ("Preset set via master\n");
    break;
   case 0x09:
    puts ("Preset set via external function\n");
    break;
   default:
    puts ("Unknown message code");
   }
  extucMessage = 0;
  delay (3000);
}//End DisplayMessage

/*---------------------------------------------------------------------
  DisplayError

  This function displays the error messages of the
  IK interrupt status.
  ---------------------------------------------------------------------*/

void DisplayError(void)
{
  gotoxy (1,23);
  switch (extucErrorCode)
   {
   case 0x01:
    puts ("Error: No latch on axis 1\n");
    break;
   case 0x02:
    puts ("Error: No latch on axis 2\n");
    break;
   case 0x03:
    puts ("Error: Double latch on axis 1\n");
    break;
   case 0x04:
    puts ("Error: Double latch on axis 2\n");
    break;
   case 0x05:
    puts ("Error: CRC correction value 1\n");
    break;
   case 0x06:
    puts ("Error: CRC correction value 2\n");
    break;
   case 0x07:
    puts("Error: parameter setting\n");
    break;
   case 0x08:
    puts("Error: CRC Eprom\n");
    break;
   case 0x09:
    puts ("Error: checksum EPROM 1\n");
    break;
   case 0x10:
    puts ("Error: checksum EPROM 2\n");
    break;
   case 0x11:
    puts ("Error: hardware\n");
    break;
   case 0x12:
    puts ("Error: wrong parameter setting\n");
    break;
   case 0x13:
    puts ("Error: axis 1 has no absolute reference\n");
    break;
   case 0x14:
    puts ("Error: axis 1 has wrong speed\n");
    break;
   case 0x15:
    puts ("Error: axis 1 has wrong position\n");
    break;
   case 0x16:
    puts ("Error: axis 1 has wrong direction\n");
    break;
```

```
    case 0x17:
     puts ("Error: axis 1 has wrong number of measuring\
            interrupts\n");
     break;
    case 0x18:
     puts ("Error: wrong calculation during compensation run\
            of axis 1\n");
     break;
    case 0x19:
     puts ("Error: axis 2 has no absolute reference\n");
     break;
    case 0x20:
     puts ("Error: axis 2 has wrong speed\n");
     break;
    case 0x21:
     puts ("Error: axis 2 has wrong position\n");
     break;
    case 0x22:
     puts ("Error: axis 2 has wrong direction\n");
     break;
    case 0x23:
     puts ("Error: axis 2 has wrong number of measuring\
            interrupts\n");
     break;
    case 0x24:
     puts ("Error: wrong calculation during compensation run\
            of axis 2\n");
     break;
    case 0x25:
     puts ("Error: REF distance: axis 1\n");
     break;
    case 0x26:
     puts ("Error: REF distance: axis 2\n");
     break;
    case 0x27:
     puts("Error: CRC - EPROM\n");
     break;
    case 0x28:
     puts ("Error: CRC - Compensation values axis 1\n");
     break;
    case 0x29:
     puts ("Error: CRC - Compensation values axis 2\n");
     break;
    case 0x30:
     puts ("Error: Stack of main program incorrect\n");
     break;
    case 0x31:
     puts ("Error: Memory area for stack is exceeded\n");
     break;
    case 0x32:
     puts ("Error:Hardware defective\n");
     break;
    case 0x33:
     puts ("Error:IK has received Unknown command\n");
     break;
    case 0x99:
     puts ("Error: Unknown code for interrupt status");
     break;
    default:
     puts ("Unknown error code");
    }
  extucErrorCode = 0;
  delay (3000);
}//End DisplayError
```

10. Technische Daten IK 320

Mechanische Kennwerte	
Abmessungen	Doppel-Europakarten-Format, Größe B Außenabmessungen: 262 mm x 187 mm x 20 mm
Arbeitstemperatur **Lagertemperatur**	0 °C bis 55 °C –40 °C bis 75 °C
Elektrische Kennwerte	
VMEbus-Spezifikation	ANSI/IEEE STD1014–1987, IEC 821 und 297 Double Height Board mit J1-Stecker 1 Slot Interrupter:D08 (O) ROAK
Adressen	Adressraum A16: Slave, D08(O) (Port) Slave, ADO (synchron einspeichern) Speicherbedarf: 16 kByte je Karte Adressraum A24: Slave, D16, D08 (EO), 16 kByte Adressraum über DIP-Schalter einstellbar
Eingänge/Ausgänge	
Messsystem-Eingänge	**IK 320 V**: X1, X3: Sub-D-Anschluss 15polig, Sinus-Signale: 1 V_{SS} **IK 320 A**: X1, X3: Sub-D-Anschluss 9polig, Sinus-Signale: 11 μA_{SS} Eingangs-Frequenz: 50 kHz **IK 320.1**: X1, X3: Sub-D-Anschluss 9polig, Sinus-Signale 22 μA_{SS}
Messsystem-Ausgänge	**IK 320 V**, **IK 320 A:** X2, X4: Sub-D-Anschluss 9polig, Sinus-Signale 11 μA_{SS} (Bezugsspannung = 0 V; keine EXEn anschließbar) **IK 320.1**: X2, X4: Sub-D-Anschluss 9polig, Sinus-Signale 22 μA_{SS}
externe Abruf-Signale	X41: Sub-D-Anschluss 9polig 6 Eingänge –IMPULS1, –IMPULS2 U_{high}: 3 – 15 V U_{low}: –0,5 – 1 V –KONTAKT1, –KONTAKT2 –F1, –F2 Ausgang –LOUT U_{high}: 2,4 – 5 V U_{low}: 0 – 0,4 V
Signal-Interpolation	4 096fach
Abgleich der Messsystem-Signale	**über 4 096 Stützpunkte**
Datenregister für Messwerte	48 Bit, wobei für den Messwert nur 44 Bit genutzt werden
Interrupts	–IRQ1 bis –IRQ7: über Jumper wählbar
Stromaufnahme	+12 V 160 mA –12 V 160 mA +5 V 1,2 A + 5 $V_{STANDBY}$ 100 µA

HEIDENHAIN

DR. JOHANNES HEIDENHAIN GmbH
Dr.-Johannes-Heidenhain-Straße 5
83301 Traunreut, Germany
☎ +49 (8669) 31-0
FAX +49 (8669) 5061
e-mail: info@heidenhain.de

www.heidenhain.de

HEIDENHAIN Technisches Büro Nord
Rhinstraße 134
12681 Berlin, Deutschland
☎ (030) 54705-240
FAX (030) 54705-200

HEIDENHAIN Technisches Büro Mitte
Kaltes Feld 22
08468 Heinsdorfergrund, Deutschland
☎ (03765) 69544
FAX (03765) 69628

HEIDENHAIN Technisches Büro West
Bandstahlstraße 2
58093 Hagen, Deutschland
☎ (02331) 9579-0
FAX (02331) 9579-49

HEIDENHAIN Technisches Büro Südwest
Eichachstraße 20
72131 Ofterdingen, Deutschland
☎ (07473) 22733
FAX (07473) 21764

HEIDENHAIN Technisches Büro Südost
Dr.-Johannes-Heidenhain-Straße 5
83301 Traunreut, Deutschland
☎ (08669) 311345
FAX (08669) 5061

AT **HEIDENHAIN**
Dr.-Johannes-Heidenhain-Straße 5
83301 Traunreut, Deutschland
☎ (08669) 311337
FAX (08669) 5061

BE **HEIDENHAIN NV/SA**
Pamelse Klei 47,
1760 Roosdaal-Pamel, Belgium
☎ (054) 343158
FAX (054) 343173

BR **DIADUR Indústria e Comércio Ltda.**
Rua Servia, 329, Santo Amaro
04763-070 – São Paulo – SP, Brazil
☎ (011) 5523 – 6777
FAX (011) 5523 – 1411

CA **HEIDENHAIN CORPORATION**
Canadian Regional Office
11-335 Admiral Blvd., Unit 11
Mississauga, Ontario L5T 2N2, Canada
☎ (905) 670-8900
FAX (905) 670-4426

CH **HEIDENHAIN (SCHWEIZ) AG**
Post Box
Vieristrasse 14
8603 Schwerzenbach, Switzerland
☎ (044) 8062727
FAX (044) 8062728

CN **HEIDENHAIN** (Tianjin)
Optics and Electronics Co. Ltd.
Room 808, The Exchange Beijing Tower 4
No. 118, Jian Guo Lu Yi
Chaoyang District
100022 Beijing, China
☎ (86) 10656732 38
FAX (86) 10656727 89

CZ **HEIDENHAIN s.r.o.**
SK Stremchová 16
106 00 Praha 10, Czech Republic
☎ 272658131
FAX 272658724

DK **TP TEKNIK A/S**
Korskildelund 4
2670 Greve, Denmark
☎ (70) 100966
FAX (70) 100165

ES **FARRESA ELECTRONICA S.A.**
Les Corts, 36-38 bajos
08028 Barcelona, Spain
☎ 934092491
FAX 933395117

FI **HEIDENHAIN AB**
Mikkelänkallio 3
02770 Espoo, Finland
☎ (09) 8676476
FAX (09) 86764740

FR **HEIDENHAIN FRANCE sarl**
2, Avenue de la Cristallerie
92316 Sèvres, France
☎ 0141143000
FAX 0141143030

GB **HEIDENHAIN (G.B.) Limited**
200 London Road, Burgess Hill
West Sussex RH15 9RD, Great Britain
☎ (01444) 247711
FAX (01444) 870024

GR **MB Milionis Vassilis**
38. Scoufa Str.
St. Dimitrios
17341 Athens, Greece
☎ (0210) 9336607
FAX (0210) 9349660

HK **HEIDENHAIN LTD**
Unit 2, 15/F, APEC Plaza
49 Hoi Yuen Road
Kwun Tong
Kowloon, Hong Kong
☎ (852) 27591920
FAX (852) 27591961

HU **HEIDENHAIN Kereskedelmi Képviselet**
Hrivnák Pál utca 13.
RO 1237 Budapest, Hungary
☎ (1) 4210952
FAX (1) 4210953

IL **NEUMO VARGUS**
Post Box 57057
34-36, Itzhak Sade St.
Tel-Aviv 61570, Israel
☎ (3) 5373275
FAX (3) 5372190

IN **ASHOK & LAL**
Post Box 5422
12 Pulla Reddy Avenue
Chennai – 600 030, India
☎ (044) 26151289
FAX (044) 26478224

IT **HEIDENHAIN ITALIANA S.r.l.**
Via Asiago 14
20128 Milano, Italy
☎ 0227075-1
FAX 0227075-210

JP **HEIDENHAIN K.K.**
Kudan Center Bldg. 10th Floor
Kudankita 4-1-7, Chiyoda-ku
Tokyo 102-0073 Japan
☎ (03) 3234-7781
FAX (03) 3262-2539

KR **HEIDENHAIN LTD.**
Suite 1415, Family Tower Building
958-2 Yeongtong-Dong
Paldal-Gu, Suwon
442-470 Kyeonggi-Do, South Korea
☎ (82) 312011511
FAX (82) 312011510

MX **HEIDENHAIN CORPORATION MEXICO**
Av. Las Américas 1808
Fracc. Valle Dorado
20235, Aguascalientes, Ags., Mexico
☎ (449) 9130870
FAX (449) 9130876

NL **HEIDENHAIN NEDERLAND B.V.**
Post Box 92, 6710 BB EDE
Copernicuslaan 34, 6716 BM EDE
The Netherlands
☎ (0318) 581800
FAX (0318) 581870

NO **KASPO MASKIN AS**
Hoeggvn. 66
7036 Trondheim, Norway
☎ (073) 969600
FAX (073) 969601

PL **APS**
Popularna 56
02-473 Warszawa, Poland
☎ (22) 8639737
FAX (22) 863 9744

PT **FARRESA ELECTRÓNICA LDA.**
Rua do Outeiro, 1315 1º M
4470 Maia, Portugal
☎ (22) 9478140
FAX (22) 9478149

SE **HEIDENHAIN AB**
Fittjavägen 23
14553 Norsborg, Sweden
☎ (08) 53193350
FAX (08) 53193377

SG **HEIDENHAIN PACIFIC PTE LTD.**
51, Ubi Crescent
Singapore 408593, Republic of Singapore
☎ (65) 6749-3238
FAX (65) 6749-3922

TR **ORSEL LTD.**
Kusdili Cad. No. 43
Toraman Han, Kat 3
81310 Kadiköy/Istanbul, Turkey
☎ (216) 3478395
FAX (216) 3478393

TW **HEIDENHAIN Co., Ltd.**
No. 12-5, Gong 33rd Road
Taichung Industrial Park
Taichung 407, Taiwan, R.O.C.
☎ (886-4) 23588977
FAX (886-4) 23588978

US **HEIDENHAIN CORPORATION**
333 State Parkway
Schaumburg, IL 60173-5337, U.S.A.
☎ (847) 490-1191
FAX (847) 490-3931

Zum Abheften hier falzen!

286 941-04 · 1 · 6/2004 · S · Printed in Germany · Änderungen vorbehalten