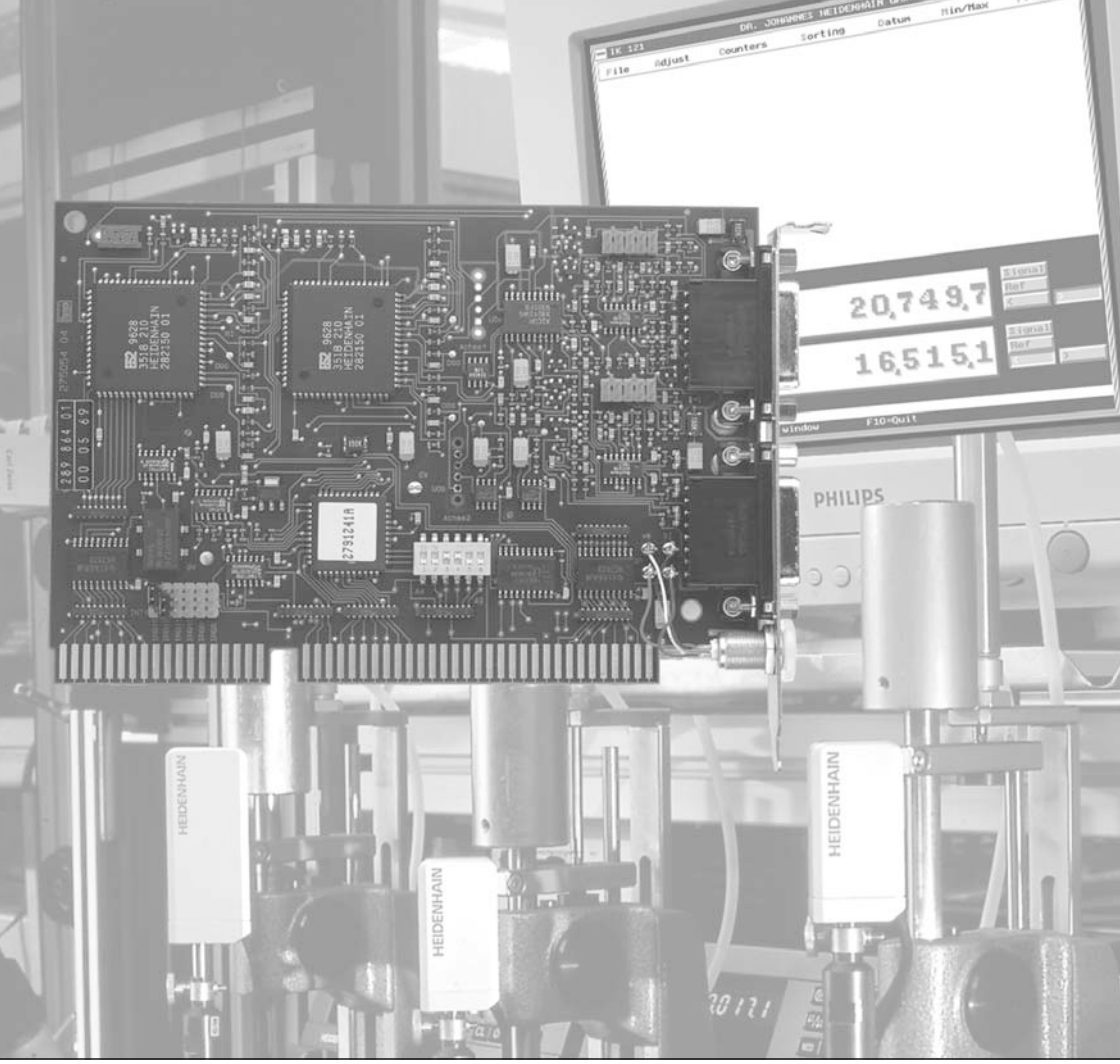


HEIDENHAIN

Benutzer-Handbuch

IK 121

PC-Zählerkarte zum Anschluss von HEIDENHAIN-Messgeräten

9/2000

Inhalt

Lieferumfang

PC-Zählerkarte IK 121, Programmierbeispiele,
Treiber-Software und Benutzer-Handbuch

Ausführungen

IK 121 A

290 155-xx PC-Zählerkarte mit Messgerät-Eingängen für sinusförmige Stromsignale (11 μA_{SS}).

IK 121 V

291 768-xx PC-Zählerkarte mit Messgerät-Eingängen für sinusförmige Spannungssignale (1 V_{SS}).

Hinweis zur EMV-Richtlinie 89/336/EWG

Die Bestimmungen der EMV-Richtlinie 89/336/EWG wurden mit dem COMPAQ-Rechner DESKPRO 386/20e geprüft.

Zubehör

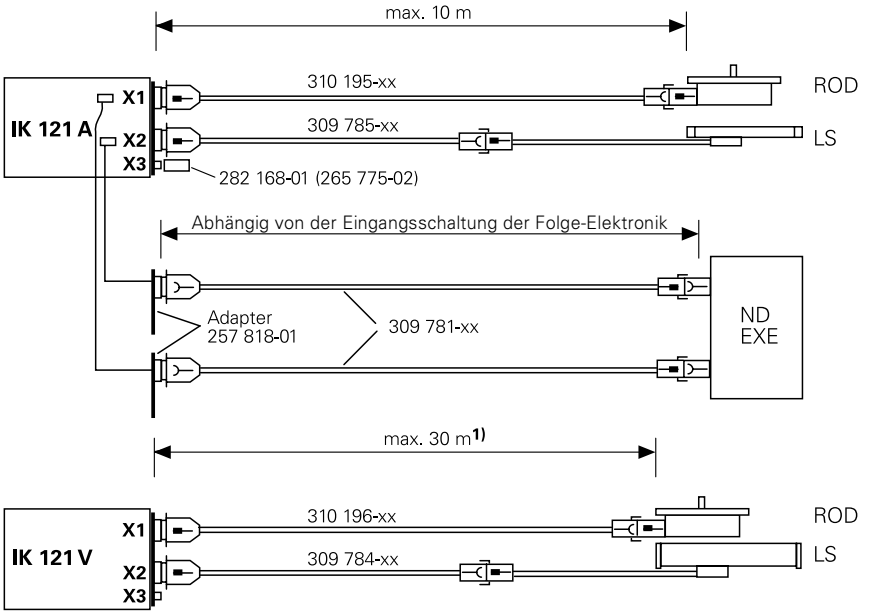
257 818-01	Zusätzlicher Sub-D-Anschluss zur Weiterführung der Messgerät-Signale des Eingangs X1 oder X2 an eine weitere Anzeige oder Steuerung
309 781-xx	Verbindungskabel vom zusätzlichen Sub-D-Anschluss an eine weitere Anzeige oder Steuerung
282 168-01 (265 775-02)	Stecker für die externen Funktionen am Anschluss X3 (zwei Buchsen, zwei Stifte)

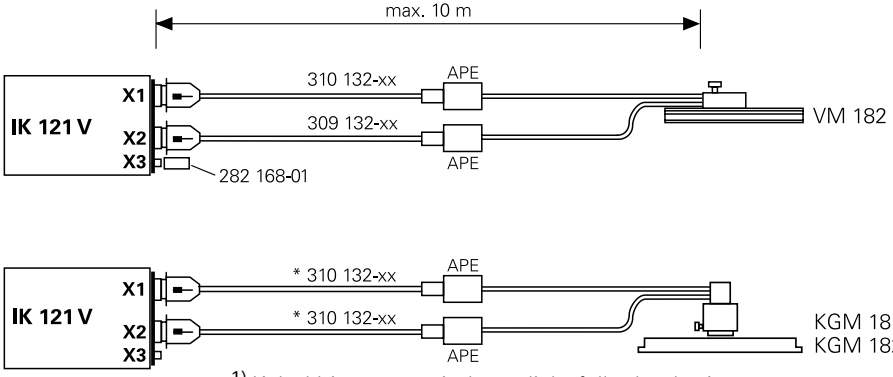
IK 121 A

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

IK 121 V

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





1) Kabel bis 150 m sind möglich, falls durch eine externe Versorgung gewährleistet ist, dass 5 V am Messgerät anliegen.

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.

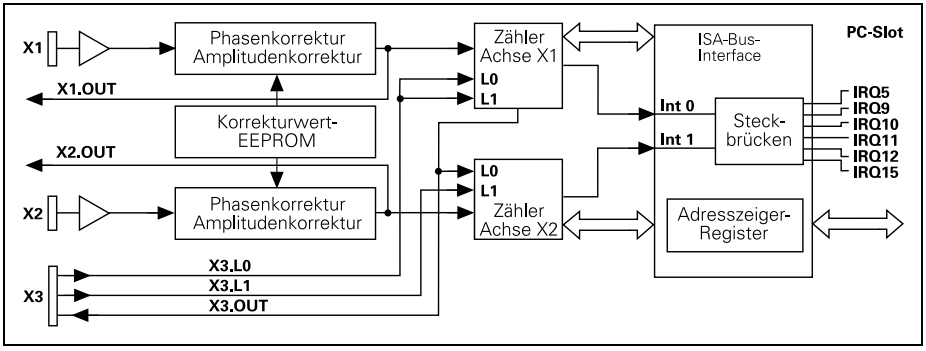
READ.ME

Die Datei READ.ME enthält die wichtigsten Informationen über die Installation der IK 121 und die mitgelieferte Software. Das Programm README.EXE stellt diese Datei seitenweise am Bildschirm dar.

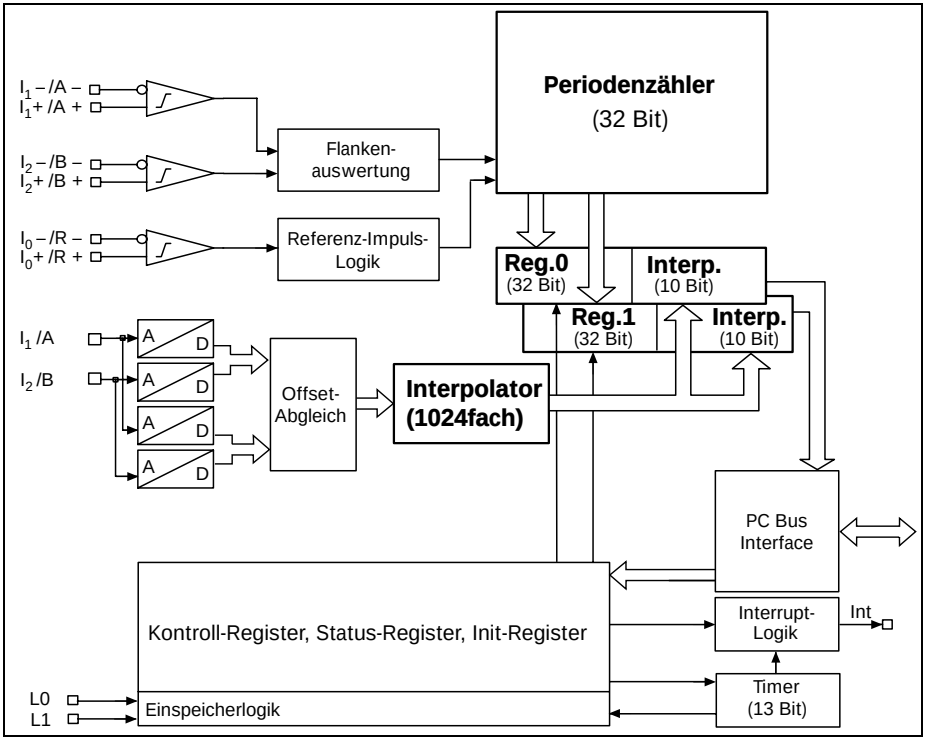
Technische Beschreibung der IK 121

Die PC-Zählerkarte IK 121 wird direkt in einen Erweiterungs-Steckplatz eines AT-kompatiblen Personal Computers gesteckt. Es können zwei HEIDENHAIN-Messgeräte mit sinusförmigen Stromsignalen (IK 121 A) oder Spannungssignalen (IK 121 V) angeschlossen werden. Die Positionen der beiden Messgeräte werden mittels Software am PC angezeigt, im PC gespeichert und weiterverarbeitet.
Die IK 121 ist ideal für Anwendungen, bei denen eine hohe Auflösung der Messgerät-Signale und eine schnelle Messwert-Erfassung erforderlich ist.

Blockschaltbild der IK 121



Blockschaltbild des Zählerbausteins



Die Interpolations-Elektronik in der IK 121 unterteilt die Signalperiode des Eingangs-Signals 1024fach.
Der Interpolationswert (10 Bit) bildet zusammen mit dem Wert des Periodenzählers (32 Bit) den 42 Bit breiten Messwert. Die Messwerte werden in 48 Bit breiten Daten-Registern gespeichert, wobei die oberen Bits entsprechend der Zweierkomplement-Darstellung vorzeichenrichtig erweitert werden.
Die Messwerte werden entweder über externe Abruf-Eingänge oder per Software oder Timer sowie durch das Überfahren der Referenzmarken über Port-Adressierung abgerufen und gespeichert.

Die Begriffe „Abruf" oder „abrufen" in diesem Handbuch bedeuten, dass der Zählerwert im Daten-Register 0 oder Daten-Register 1 festgehalten wird. Dieser Zählerwert muss anschließend per Software gelesen und im PC gespeichert oder am Bildschirm angezeigt werden.

Phase und Amplitude der sinusförmigen Messgerät-Signale können über elektronische Potentiometer per Software abgeglichen werden; der Offset über Datenregister im Zählerbaustein.

Zugriffszeit auf Messwerte

Die Zugriffszeit auf die Messwerte beträgt ca. 35 µs.

Hardware

Spezifikation des PC-Bus

Die IK 121 kann in alle IBM AT und 100% kompatiblen PCs eingesetzt werden. HEIDENHAIN garantiert nicht für die einwandfreie Funktion der IK 121 mit nicht 100% kompatiblen PCs. Die IK 121 entspricht der internationalen Norm IEEE P996, die den AT- und ISA-Bus spezifiziert (Industrie-Standard).

Busbreite	16 Bit
Spannungsversorgung	+5 V ±5% +12 V ±5% –12 V ±5%
Leistungsaufnahme	ca. 1 Watt ohne Messgeräte
Slot-Ausführung	AT-kompatibel, kurzer 16 Bit-Slot

Messgerät-Eingänge IK 121 A

An die IK 121 A werden HEIDENHAIN-Längenmessgeräte oder -Winkelmessgeräte mit sinusförmigen Stromsignalen I_1 und I_2 angeschlossen. Zusätzlich steht das Referenzmarken-Signal I_0 zur Verfügung.

Signalamplituden I_1, I_2 (0°, 90°) I_0 (Referenzmarke)	 7 μA_{SS} bis 16 μA_{SS} 3,5 µA bis 8 µA
Signalpegel für Fehlermeldung	≤ 2,5 μA_{SS}
Maximale Eingangsfrequenz	100 kHz
Kabellänge	max. 10 m

Anschluss X1, X2 für Messgeräte
Sub-D-Anschluss mit Buchseneinsatz (9-polig)

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

Messgerät-Eingänge IK 121 V

An die IK 121 V werden HEIDENHAIN-Längenmessgeräte oder -Winkelmessgeräte mit sinusförmigen Spannungssignalen A und B angeschlossen. Zusätzlich steht das Referenzmarken-Signal R zur Verfügung.

Signalamplituden A, B (0°, 90°) R (Referenzmarke)	 0,6 V_{SS} bis 1,2 V_{SS} 0,2 V bis 0,85 V
Signalpegel für Fehlermeldung	≤ 0,22 V_{SS}
Maximale Eingangsfrequenz	400 kHz
Kabellänge[1)]	max. 30 m

[1)] Kabel bis 150 m sind möglich, falls durch eine externe Versorgung gewährleistet ist, dass 5 V am Messgerät anliegen.
Die Eingangsfrequenz reduziert sich in diesem Fall auf max. 250 kHz.

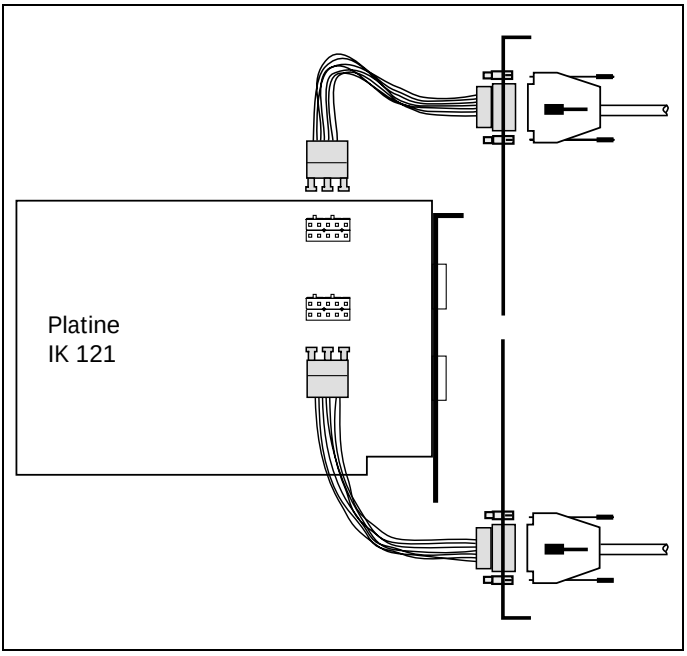
Anschluss X1, X2 für Messgeräte
Sub-D-Anschluss mit Buchseneinsatz (15-polig)

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

Messgerät-Ausgänge

Die IK 121 gibt die Messgerät-Signale der Eingänge X1 und X2 zusätzlich an zwei 10-polige AMP-Stecker als **sinusförmige Stromsignale** (11 μA_{SS}) aus. Über zusätzliche Kabelbaugruppen mit PC-Slot-Abdeckung (Id.-Nr. 257 818-01) können diese Anschlüsse nach außen zu 9-poligen Sub-D-Anschlüssen geführt werden. Adapterkabel (Id.-Nr. 309 781-xx) zum Anschluss an HEIDENHAIN-Positionsanzeigen oder Interpolations-Elektroniken sind lieferbar (siehe „Lieferumfang", „Zubehör").

Die maximale Kabellänge ist abhängig von der Eingangsschaltung der Folge-Elektronik.



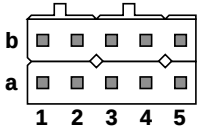
Messgerät-Ausgänge (Ident-Nummer 257 818-01)
Sub-D-Anschluss mit Stifteinsatz (9-polig)

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

Platinenstecker für Messgerät-Ausgänge
AMP mit Stifteinsatz (10-polig)

Anschluss-Nr.[1)]	Signal
1a	nicht angeschlossen
1b	nicht angeschlossen
2a	nicht angeschlossen
2b	0 V (U_N)
3a	I_0 –
3b	I_0 +
4a	I_2 –
4b	I_2 +
5a	I_1 –
5b	I_1 +

1) Die Seite mit den Verriegelungs-Stiften ist mit b bezeichnet. Anschlüsse 1a und 1 b befinden sich auf der Seite mit der Einkerbung.



Abgleich der Messgerät-Signale

Messgerät-Signale können wie folgt abgeglichen werden:
- Phase und Amplitude über elektronische Potentiometer
- Symmetrie (Offset) in den Zählerbausteinen mit Offset-Registern

Die Ansteuerung der Potentiometer erfolgt über I²C-Bus. Da die Erzeugung der Steuersequenzen aufwendig ist, soll zum Abgleich das Programm POTIS.EXE oder ADJUST.EXE benutzt werden.

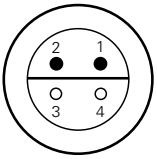
Wenn es für eine Anwendung doch erforderlich ist, die Potentiometer anzusteuern, dann können die Funktionen und Prozeduren in „TURBO PASCAL" aus der Datei IIC.PAS genutzt werden oder als Orientierungshilfe für eigene Funktionen dienen.

Die Korrekturwerte für Phase und Amplitude werden in den Bausteinen der elektronischen Potentiometer netzausfallsicher gespeichert. Die Offset-Register in den Zählerbausteinen sind nicht netzausfallsicher. Deshalb werden die Offset-Korrekturwerte in einem EEPROM im Baustein für die elektronischen Potentiometer gespeichert. Nach dem Einschalten müssen die Offset-Korrekturwerte vom EEPROM in die Offset-Register der Zählerbausteine geladen werden. In der Datei IIC. PAS sind zwei Prozeduren definiert, die diese Aufgaben erfüllen. Die Prozedur „store_offset" speichert die Offset-Korrekturwerte in das EEPROM. Die Prozedur „load_offset" liest die Offset-Korrekturwerte aus dem EEPROM in die Offset-Register der Zählerbausteine. „Load_offset" wird auch von der Prozedur „init_IK121" genutzt.

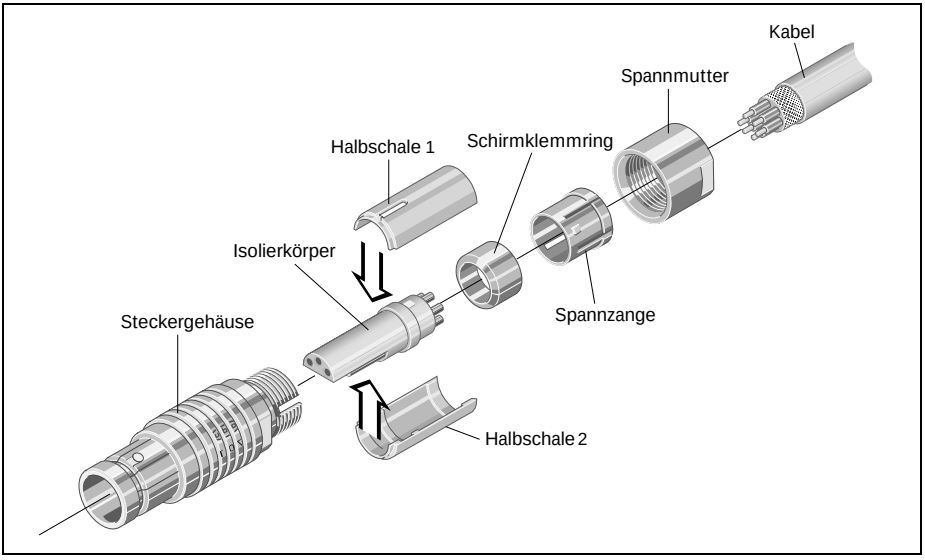
Externe Funktionen

Für externe Funktionen ist eine 4-polige Flanschdose vorhanden. Der dafür benötigte Stecker (Id.-Nr. 282 168-01) kann bei HEIDENHAIN bestellt werden.

Montage des Steckers für die externen Funktionen



Rückansicht
o = Buchse, • = Stift



Anschluss X3 für externe Funktionen
Flanschdose mit Stift/Buchseneinsatz (4-polig)

Anschluss-Nr.	Belegung
1	Eingang: Messwert-Abruf X1 (X3.L0)
2	Eingang: Messwert-Abruf X2 (X3.L1)
3	Ausgang: Messwert-Abruf (X3.OUT)
4	0 V

Abruf der Messwerte über externe Eingänge
Die IK 121 hat zwei externe Eingänge an der Flanschdose X3 zum Abrufen und Speichern der Messwerte.

Über diese Eingänge können auch Interrupts ausgelöst werden.

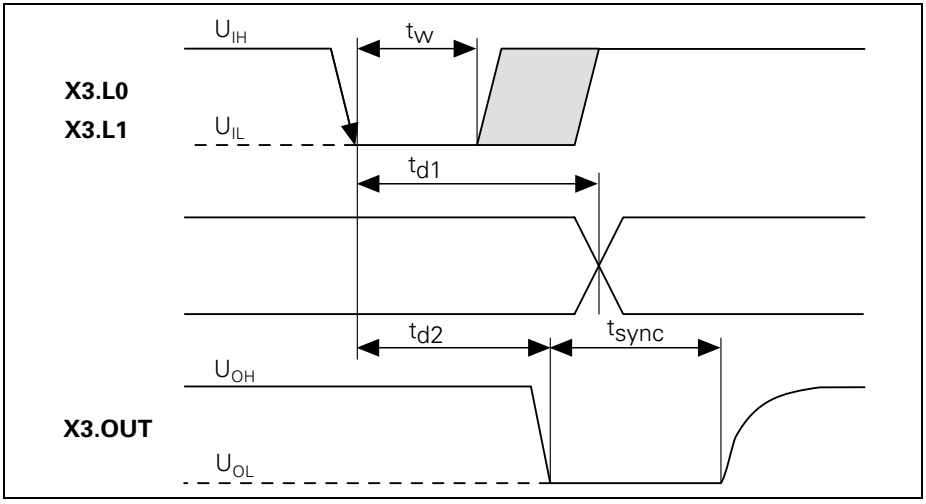
Die Eingänge X3.L0 und X3.L1 sind low-aktiv; ein interner Pull-up-Widerstand (10 kΩ) hält sie auf High-Pegel. Sie können an TTL-, LS- oder CMOS-Bausteine angeschlossen werden.

Die einfachste Art, die Eingänge zu aktivieren: Eine Brücke von 0 Volt (Anschluss 4) auf den Eingang zum Abrufen.

Abruf-Ausgang X3.OUT
Das Ausgangs-Signal X3.OUT kann über die Flanschdose X3 zu weiteren IK 121 geleitet werden (Eingänge X3.L0, X3.L1), um beispielsweise die Messwerte von verschiedenen IKs abzurufen.

X3.OUT ist ein Open-Collector-Ausgang, der auf „Null" schaltet.

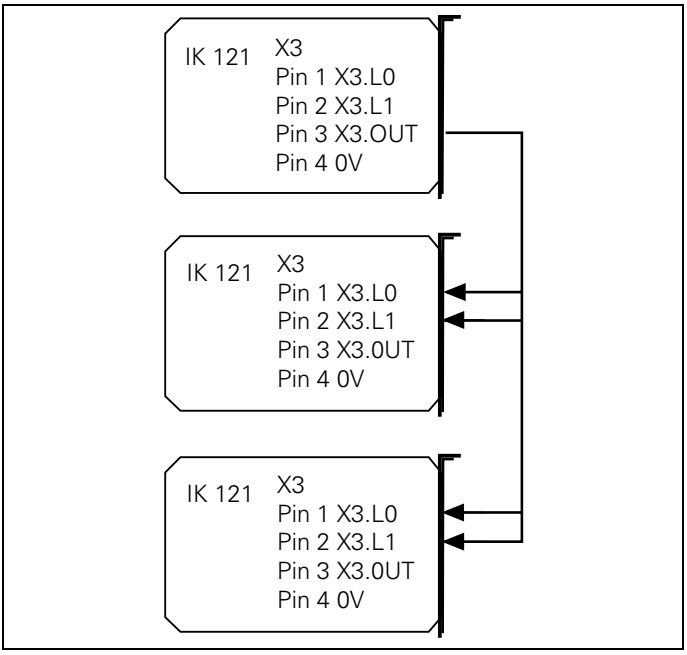
Abruf-Eingänge X3.L0, X3.L1 und Abruf-Ausgang X3.OUT: Zeitablauf und Spannungspegel



Bezeichnung	MIN	MAX
U_{IL} (V)	–3,0	0,9
U_{IH} (V)	3,15	30
t_{d1} (µs)	–	24
t_{d2} (ns)	–	500
t_w (ns)	250	–
t_{sync} (µs)	1	–
U_{OL} (V)	0	0,8 (U_{OH} = 4 V – 12 V) 1,0 (U_{OH} = 12 V – 32 V)
U_{OH} (V)	4	32
I_{OL} (mA)	–	40

Messwerte von mehreren IK 121 abrufen

Über den Abruf-Ausgang X3.OUT ist es möglich, die Messwerte von mehreren IK 121 abzurufen. Dazu ist der Abruf-Ausgang X3.OUT einer IK 121 mit den Abruf-Eingängen weiterer IK 121 zu verbinden.

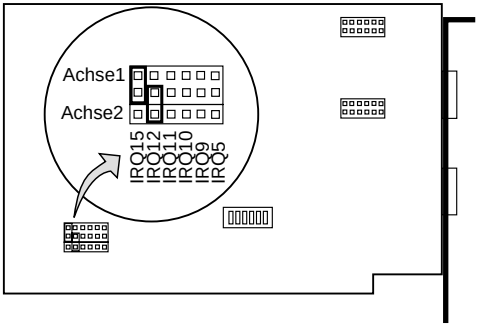


Interrupts

Die IK 121 kann einen der folgenden PC-Interrupts nutzen: IRQ5, IRQ9, IRQ10, IRQ11, IRQ12 oder IRQ15.

Der gewünschte Interrupt wird über Steckbrücken auf der Platine festgelegt.

Achse 1 erzeugt das interne Signal **Int0** und Achse 2 **Int1.** Durch die Anordnung der Stiftleisten können **Int0** und **Int1** nicht auf die gleiche IRQ-Leitung gelegt werden.



Bei der oben gezeigten Einstellung liegt Achse 1 auf IRQ15 und Achse 2 auf IRQ12.

Belegung der PC-Interrupts

Interrupt	Interrupt-Nummer	Interrupt-Adresse
IRQ5	0D	034 bis 037
IRQ9	71	1C4 bis 1C7
IRQ10	72	1C8 bis 1CB
IRQ11	73	1CC bis 1CF
IRQ12	74	1D0 bis 1D3
IRQ15	77	1D7 bis 1DF

Adressierung

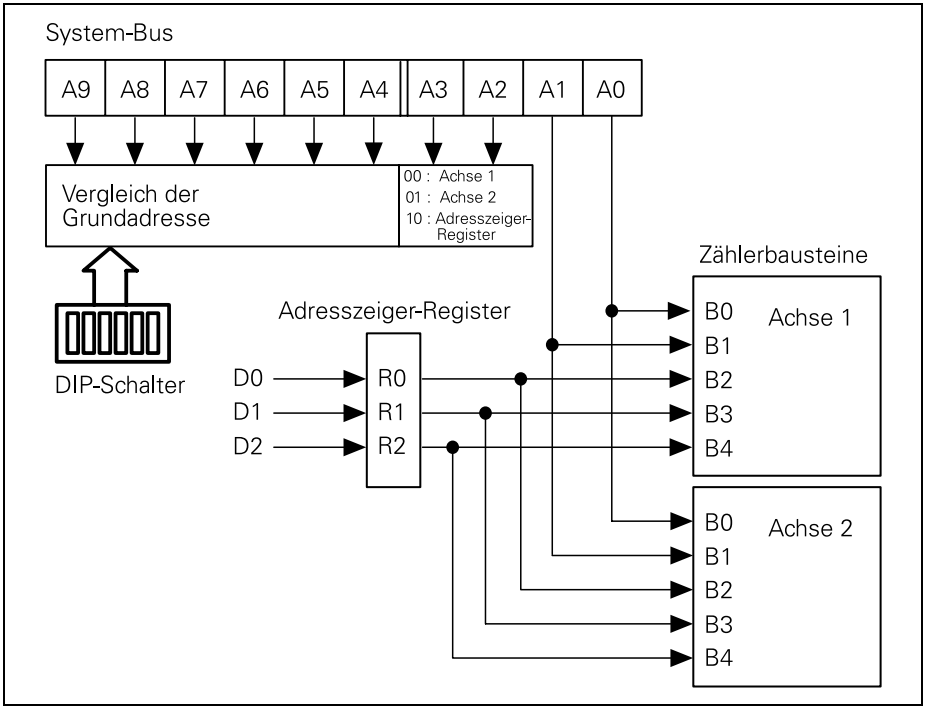
IK 121 und Rechner kommunizieren über Port-Adressen (I/O-Bereich) mit 16-Bit-Datenbreite. Da im Port-Adressbereich der freie Adressraum eng begrenzt ist, wurden die Adressen der Zählerbausteine übereinander gelegt. Zur Unterscheidung wird ein Adresszeiger-Register benutzt.

Jeder Zählerbaustein hat fünf Adressleitungen B0 bis B4. B0 und B1 sind direkt mit dem Systembus verbunden. B2, B3 und B4 werden im Adresszeiger-Register erzeugt. A2 und A3 des System-Bus dekodieren die Zählerbausteine für Achse 1, Achse 2 oder für das Adresszeiger-Register.

A4 bis A9 des Systembus (Grundadresse) werden über den DIP-Schalter auf der Platine eingestellt. Die Einstellung „on" entspricht einer logischen „Null".

Beispiele für Schaltereinstellungen

Adresse	**Schalterstellung**					
(Hex)	A9	A8	A7	A6	A5	A4
0110	on	off	on	on	on	off
0250	off	on	on	off	on	off
0300	off	off	on	on	on	on
0330	off	off	on	on	off	off



Register

Für die folgende Beschreibung ist das Prinzip-Schaltbild der Zählerbausteine auf der letzten Umschlagseite hilfreich.

Wichtiger Hinweis zur Programmierung:
Der Zugriff auf die IK 121 erfolgt durch Lesen von Datenworten, die in Registern abgelegt sind, und durch Schreiben von Datenworten in die Register. Deshalb dürfen nur gerade Port-Adressen mit Wort-Schreib- und Wort-Lesebefehlen adressiert werden.

Register-Übersicht

Adresse (Hex) B0 bis B4	Schreibzugriff	Lesezugriff
00 02 04	ohne Funktion	Daten-Register 0, LS-Word Daten-Register 0 Daten-Register 0, MS-Word
06 08 0A	ohne Funktion	Daten-Register 1, LS-Word Daten-Register 1 Daten-Register 1, MS-Word
0C Low Byte High Byte	Initialisierungs-Register 1 Initialisierungs-Register 2	Initialisierungs-Register 1 Initialisierungs-Register 2
0E Low Byte High Byte	Kontroll-Register 1 ohne Funktion	Status-Register 1 Status-Register 2
10 Low Byte High Byte	Referenzmarken-Register ohne Funktion	ohne Funktion Amplitudenwert-Register
12 Low Byte High Byte	Freigabe-Register für Messwert-Abruf Achs-Kaskadierung	ohne Funktion ohne Funktion
14 Low Byte High Byte	Interrupt-Freigabe-Register ohne Funktion	Interrupt-Status-Register 1 Interrupt-Status-Register 2
16 Low Byte High Byte	Offset-Register für 0°-Signal ohne Funktion	Amplitude für 0°-Signal Amplitude für 0°-Signal
18 Low Byte High Byte	Offset-Register für 90°-Signal ohne Funktion	Amplitude für 90°-Signal Amplitude für 90°-Signal
1A Low Byte High Byte	Timer-Register, LS-Byte Timer-Register, MS-Byte	ohne Funktion ohne Funktion
1C Low Byte High Byte	Kontroll-Register 2 ohne Funktion	Status-Register 3 Kennungs-Register
1E Low Byte High Byte	Kontroll-Register 3 ohne Funktion	Status-Register 4 ohne Funktion

Daten-Register für die Zähler

Die Messwerte werden in 48 Bit breiten Registern gespeichert. Pro Achse stehen zwei Daten-Register zur Verfügung: Daten-Register 0 (00h bis 04h) und Daten-Register 1 (06h bis 0Ah). Die Messwerte werden aus dem 10-Bit-Interpolationswert und dem 32 Bit breiten Wert des Periodenzählers zusammengesetzt. Von den 48 Bit breiten Registern werden also nur 42 Bit für den Messwert genutzt. Die oberen 6 Bits werden entsprechend der Zweier-Komplement-Darstellung vorzeichenrichtig erweitert.

Die Datenbreite von 48 Bit kann über das Initialisierungs-Register 1 (0Ch), Bit D6 auf 32 Bit verkürzt werden.

Über das Initialisierungs-Register 1 (0Ch), Bit D7 kann außerdem festgelegt werden, ob der Messwert nur aus dem Wert des Periodenzählers (Datenbits D0 bis D9 sind nicht definiert) oder aus dem Wert des Periodenzählers und aus dem Interpolationswert gebildet wird.
Das Speichern der Zählerwerte in die Daten-Register kann erfolgen über:

- Software-Abruf
- externe Eingänge
- Timer
- Referenzmarken

Die Wirkungsweise der verschiedenen Abruf-Signale zeigt das Prinzip-Schaltbild der Zählerbausteine auf der hinteren Umschlagseite.

Über das Status-Register 1 (0Eh), Bit D0 oder D1, kann abgefragt werden, ob der Messwert in den Daten-Registern gespeichert wurde. Solange Bit D0 oder D1 gesetzt sind, kann kein weiterer Messwert gespeichert werden, bis das oberste Wort des Messwerts gelesen wurde. (Ausnahme: über Kontroll-Register 2, Bit D6 oder D7, wird das Abrufen ohne vorheriges Auslesen des Messwerts freigegeben.) Im 48-Bit-Modus sind dies die Daten-Register 04h oder 0Ah, im 32-Bit-Modus die Daten-Register 02h oder 08h. Nach dem Lesen des Messwerts wird in Status-Register 1 (0Eh), Bit D0 oder D1, wieder rückgesetzt.

Wird der Zähler gestoppt oder durch Überfahren der Referenzmarke gespeichert, dann steht in D0 bis D9 der feste Wert 256.

0Ch: Initialisierungs-Register 1 (Schreibzugriff)

Bit	Funktion
D0	**Betrieb mit/ohne Interpolation** 0 = Betrieb als Periodenzähler (ohne Interpolation – Datenbits D0 bis D9 sind nicht definiert) 1 = Messwert wird aus dem Wert des Perioden-zählers und aus dem Interpolationswert gebildet.
D1	ohne Funktion
D2	**Timer** 0 = Timer nullen und stoppen 1 = Timer starten
D3	ohne Funktion
D4	
D5	
D6	**Einspeicher-Freigabe** 0 = Betriebsart: 32-Bit-Register Lesen der Bits D24 bis D31 setzt das Status-Bit D0 bzw. D1 im Status-Register 1 (0Eh) zurück. 1 = Betriebsart: 48-Bit-Register Lesen der Bits D40 bis D47 setzt das Status-Bit D0 bzw. D1 im Status--Register 1 (0Eh) zurück.
D7	**Zählrichtung** Die Zählrichtung legt fest, ob die Zähler bei positiver Verfahrrichtung positiv (normal) oder negativ (invers) zählen. 0 = Zählrichtung normal 1 = Zählrichtung invers Die Zählrichtung darf nur im Periodenzähler-Betrieb invers sein! Im Betrieb mit Interpolation ergibt die invertierte Zählrichtung eine fehlerhafte Verknüpfung von Interpolationswert und Periodenzählerwert.

0Ch: Initialisierungs-Register 2 (Schreibzugriff)

Bit	Funktion
D8 D9	**Nur im Betrieb als Periodenzähler:** **Flankenauswertung** Durch die beiden inkrementalen Messgerät-Signale (0°el. und 90° el.) stehen pro Signalperiode maximal vier Flanken zur Auswertung zur Verfügung. Die Zähler können so programmiert werden, dass sie entweder eine, zwei oder vier Flanken pro Signalperiode zählen. D9 D8 0 0 = 1fach 1 0 = 2fach 1 1 = 4fach Im Betrieb mit Interpolationswert ist automatisch die 1fache Auswertung eingestellt.
D10	**Nur im Betrieb als Periodenzähler:** **Zählweise** 0 = Linear-Zählweise -2^{41} bis $+2^{41}-1$ 1 = Winkel-Zählweise wie in D11 festgelegt Für Winkelmessgeräte mit 36 000 oder 360 000 Strichen pro Umdrehung.
D11	**Nur bei Winkel-Anzeige:** **Zählweise** 0 = 17 999 bis –18 000 1 = 179 999 bis –180 000
D12 **D13**	ohne Funktion
D14 **D15**	**Messwert-Abruf mit Referenzimpuls** 0 = 1. Referenzmarke speichert in Daten-Register 0 1 = 1. Referenzmarke speichert in Daten-Register 1 0 = 2. Referenzmarke speichert in Daten-Register 0 1 = 2. Referenzmarke speichert in Daten-Register 1

0Ch: Lesezugriff: Bits D0 bis D15: Rücklesen der Initialisierungs-Register 1 und 2

0Eh: Kontroll-Register 1 (Schreibzugriff)

Bit	Funktion
D0	1 = Software-Abruf: Messwert in Daten-Register 0
D1	1 = Software-Abruf: Messwert in Daten-Register 1
D2	1 = Software-Abruf in alle Daten-Register (muss im Abruf-Freigabe-Register freigegeben werden)
D3	1 = Zähler starten
D4	1 = Zähler stoppen
D5	1 = Zähler löschen
D6	1 = Messgerätfehler löschen (Frequenzüberschreitung)
D7	Amplitudenwert-Register löschen
D8 D9 D10 D11 D12 D13 D14 D15	ohne Funktion

0Eh: Status-Register 1 (Lesezugriff)

Bit	Funktion
D0	Status für Software-Abruf in Register 0 1 = Messwert ist bereit
D1	Status für Software-Abruf in Register 1 1 = Messwert ist bereit
D2	ohne Funktion
D3	ohne Funktion
D4	1 = Zähler ist gestoppt
D5	Zählerbaustein 1: ohne Funktion Zählerbaustein 2: I²C-Bus-Leitung SDA (Eingang)
D6	1 = Messgerätfehler (Frequenzüberschreitung)
D7	ohne Funktion

0Eh: Status-Register 2 (Lesezugriff)

Bit	Funktion
D8	1 = Anfahren der Referenzmarke ist aktiv
D9 D10 D11 D12	ohne Funktion
D13	Logik-Pegel für das 0°-Signal
D14	Logik-Pegel für das 90°-Signal
D15	Logik-Pegel der Referenzmarke

10h: Referenzmarken-Register (Schreibzugriff)
HEIDENHAIN-Längen- und Winkelmessgeräte können eine oder mehrere Referenzmarken haben. HEIDENHAIN empfiehlt insbesondere Messgeräte mit abstandscodierten Referenzmarken.

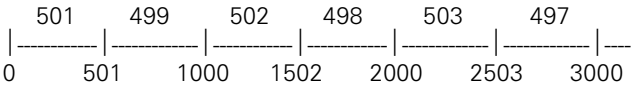
Bei einer Stromunterbrechung geht die Zuordnung zwischen der Position des Messgerätes und dem angezeigten Positionswert verloren. Mit den Referenzmarken der Messgeräte kann man die Zuordnung nach dem Einschalten wieder herstellen.

Beim Überfahren der Referenzmarken wird ein Signal erzeugt, das diese Maßstabs-Position als Referenzpunkt kennzeichnet. Mit diesem Referenzpunkt lassen sich die Zuordnungen zwischen den Achspositionen und Anzeigewerten, die zuletzt festgelegt wurden, wieder herstellen. Bei Längenmessgeräten mit abstandscodierten Referenzmarken braucht man dazu nur maximal um 20 mm zu verfahren.

Bit	Funktion beim Überfahren der Referenzmarke
D0	1 = Zähler starten
D1	1 = Zähler stoppen
D2	1 = Zähler löschen
D3	1 = Messwert abrufen
D4	1 = Messwert mit dem Überfahren der 2. Referenzmarke abrufen
D5	1 = Zähler mit dem Überfahren jeder Referenzmarke löschen
D6 D7 D8 D9 D10 D11 D12 D13 D14 D15	ohne Funktion

Auswerten abstandscodierter Referenzmarken
Bei Messgeräten 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 ganzen Messweg vorkommt. So können nach einer Stromunterbrechung nur durch Überfahren von zwei Referenzmarken die Zuordnungen zwischen Achspositionen und Anzeigewerten wieder hergestellt werden.

Die Auswertung dieser abstandscodierten Referenzmarken erläutert das folgende Beispiel:
Bei einem Grundabstand von 1 000 Signalperioden ergibt sich die folgende Verteilung in Inkrementen für die Referenzmarken:



Zunächst muss das Referenzmarken-Register 10h wie folgt initialisiert werden:
- Zähler mit dem Überfahren der 1. Referenzmarke löschen und starten (Bit D0 = 1 und D2 =1).
- Zähler mit dem Überfahren der 2. Referenzmarke abrufen (Bit D4 = 1).

Zum Berechnen der absoluten Position wird nur der Abstand zwischen den Referenzmarken in Signalperioden benötigt. Dieser Abstand in Inkrementen, in der folgenden Beschreibung mit DIFF bezeichnet, muss durch 1 024 geteilt werden.

DIFF = DISTANCE_IN_INCR : 1 024

Zum Berechnen der absoluten Position muss man den Abstand (OFFSET) zwischen der 1. Referenzmarke auf dem Maßstab (Position „0" auf der Zeichnung) und der 1. überfahrenen Referenzmarke ermitteln.

Es gibt vier mögliche Fälle:
1. Positive Verfahrrichtung und DIFF > 500
   OFFSET = (DIFF – 501) • 1 000
2. Positive Verfahrrichtung und DIFF < 500
   OFFSET = (500 – DIFF) • 1 000 – DIFF
3. Negative Verfahrrichtung und |DIFF| > 500
   OFFSET = (DIFF – 501) • 1 000 + DIFF
4. Negative Verfahrrichtung und |DIFF| < 500
   OFFSET = (500 – DIFF) • 1 000

Die absolute Position in Inkrementen berechnet sich wie folgt:

ABS_POS_INCR = ACT_POS + PRESET + DISTANCE

ACT_POS: Abstand zwischen der aktuellen Position und der 1. überfahrenen Referenzmarke (in Inkrementen).

PRESET: Position, die beim Bezugspunkt-Setzen auf die 1. Referenzmarke des Maßstabs (Position „0" auf der Zeichnung) gesetzt wird (in Inkrementen).

DISTANCE: Abstand zwischen der 1. Referenzmarke auf dem Maßstab und der 1. überfahrenen Referenzmarke (in Inkrementen).

DISTANCE = OFFSET • 1 024

Die absolute Position wird – z.B. bei einem Maßstab mit Signalperiode von 0,02 mm – wie folgt berechnet:

ABS_POS_MM = Fehler!

Bei Winkelmessgeräten:

ABS_POS_DEGREE = Fehler!

Ein Anwendungsbeispiel in „TURBO PASCAL" zum Auswerten von Referenzmarken finden Sie im Quellcode von TNC.EXE in der Datei CNT_2.PAS unter (* Distance-coded ref. marks *).

10h: Amplitudenwert-Register (Lesezugriff)

Bit	Funktion
D0 D1 D2 D3 D4 D5 D6 D7	ohne Funktion
D8 D9	**Aktuelle Amplitude** Mit jedem Messwert-Abruf wird ein neuer Amplitudenwert ermittelt. D9 D8 **IK 121 A** **IK 121 V** 0 0 normale Amplitude 5 µA < I_e < 15 µA 0,47 V_{SS} < U_e < 1,41 V_{SS} 0 1 kleine Amplitude 2,5 µA < I_e < 5 µA 0,23 V_{SS} < U_e < 0,47 V_{SS} 1 0 hohe Amplitude I_e > 15 µA U_e > 1,41 V_{SS} 1 1 fehlerhaft kleine Amplitude I_e < 2,5 µA U_e < 0,23 V_{SS} Der Amplitudenwert sollte vor dem Lesen durch Bit D4 im Kontroll-Register 2 eingefroren werden. Das Amplitudenwert-Register wird durch Bit D7 im Kontroll-Register 1 rückgesetzt.
D10 D11	**Minimalwert der Amplitude** Kodierung und Lesezugriff siehe Bits D8 und D9. Wenn der aktuelle Amplitudenwert mehr als viermal hintereinander den gespeicherten Minimalwert unterschreitet, ersetzt die IK den alten Minimalwert durch den aktuellen Amplitudenwert.
D12 D13 D14 D15	ohne Funktion

12h: Freigabe-Register für Messwert-Abruf (Schreibzugriff)

Bit	Funktion
D0	Freigabe L0 für Daten-Register 0 Achse 1: 0 Achse 2: 1 = Freigabe der Kaskadierung mit Achse 1
D1	Freigabe L0 über Verzögerungsglied (125 ns) für Daten-Register 0 Achse1: 1 = Freigabe des externen Abrufsignals m X3.L0 für Daten-Register 0 Achse 2: 0
D2	1 = Freigabe des „Software-Abrufs in alle Daten-Register“ für Daten-Register 0
D3	1 = Freigabe des „Software-Abrufs über Timer“ für Daten-Register 0
D4	Freigabe L1 für Daten-Register 1 Achse 1: 1 = Freigabe des externen Abrufsignals X3.L0 für Daten-Register 1 Achse 2: 1 = Freigabe des externen Abrufsignals X3.L1 für Daten-Register 1
D5	Freigabe L1 über Verzögerungsglied (125 ns) für Daten-Register 1 0
D6	1 = Freigabe des „Software-Abrufs in alle Daten-Register“ für Daten-Register 1
D7	1 = Freigabe des „Software-Abrufs über Timer“ für Daten-Register 1

Das Prinzip-Schaltbild der Zählerbausteine auf der hinteren Umschlagseite verdeutlicht die Funktion der einzelnen Bits.

12h: Register für Achs-Kaskadierung und zur I²C-Bus Ansteuerung (Schreibzugriff)

Bit	Funktion
D8	Achse 1: 1 = Freigabe des externen Abrufsignals zur 2. Achse (L0, X3.OUT) Achse 2: 0
D9	Achse1: 1 = Freigabe des „Software-Abrufs in alle Daten-Register" zur 2. Achse (L0, X3.OUT) Achse 2: 0
D10	Achse 1: 1 = Freigabe des Timerstrobes zur 2. Achse (L0, X3.OUT) Achse 2: 0
D11	Achse 1: 0 Achse 2: I²C-Bus-Leitung SDA: Daten (Signale müssen invertiert programmiert werden)
D12	ohne Funktion
D13	
D14	
D15	Achse 1: 0 Achse 2: I²C-Bus-Leitung SCL: Takt (Signale müssen invertiert programmiert werden)

Das Prinzip-Schaltbild der Zählerbausteine auf der hinteren Umschlagseite verdeutlicht die Funktion der einzelnen Bits.

12h: Lesezugriff ohne Funktion

14h: Interrupt-Freigabe-Register (Schreibzugriff)

Die Interrupt-Logik wird über das Interrupt-Freigabe-Register programmiert. Die Interrupt-Ursache kann der Messwert-Abruf in Register 0, Register 1 oder der Timer-Strobe sein. Alle drei Interrupt-Quellen können unabhängig voneinander programmiert werden. Bei gleichzeitigem Eintreffen mehrerer Interrupts besteht folgende Priorität:

- höchste Priorität: Messwert-Abruf über Register 0
- zweithöchste Priorität: Messwert-Abruf über Register 1
- niedrigste Priorität: Messwert-Abruf über Timer-Strobe

Nach einem Interrupt kann kein weiterer Interrupt erfolgen, bis das Interrupt-Status-Register 2 gelesen worden ist.

Bit	Funktion
D0	1 = Freigabe des Interrupts 0 beim Messwert-Abruf über Register 0
D1	1 = Freigabe des Interrupts 1 beim Messwert-Abruf über Register 1
D2	1 = Freigabe des Interrupts 2 für Timer-Strobe
D3	1 = Interrupt wird erzeugt (D0 = D1 = D2 = 0)
D4 D5 D6 D7 D8 D9 D10 D11 D12 D13 D14 D15	ohne Funktion

Das Prinzip-Schaltbild der Zählerbausteine auf der hinteren Umschlagseite verdeutlicht die Funktion der einzelnen Bits.

14h: Interrupt-Status-Register 1 (Lesezugriff)

In diesem Register wird der gerade aktive Interrupt angezeigt (es kann immer nur 1 Bit gesetzt sein). Durch Lesen dieses Registers wird der gerade aktive Interrupt rückgesetzt und das zugehörige Status-Bit gelöscht, so dass der nächste Interrupt durch eine negative Flanke ausgelöst werden kann.

Bit	Funktion
D0	1 = Interrupt 0 ist aktiv, es erfolgte ein Messwert-Abruf über Daten-Register 0
D1	1 = Interrupt 1 ist aktiv, es erfolgte ein Messwert-Abruf über Daten-Register 1
D2	1 = Interrupt 2 ist aktiv, es erfolgte ein Messwert-Abruf über den Timer-Strobe
D3 D4 D5 D6 D7	ohne Funktion

14h: Interrupt-Status-Register 2 (Lesezugriff)

In diesem Register werden alle Interrupts angezeigt, die anstehen, d.h. der aktive Interrupt und die Interrupts, die noch ausgeführt werden müssen. Es können also mehrere Bits gleichzeitig gesetzt sein.

Bit	Funktion
D8	1 = Interrupt 0 steht an, wird aber noch nicht ausgeführt
D9	1 = Interrupt 1 steht an, wird aber noch nicht ausgeführt
D10	1 = Interrupt 2 steht an, wird aber noch nicht ausgeführt
D11 D12 D13 D14 D15	ohne Funktion

16h: Offset-Register für 0°-Signal (Schreibzugriff)
Dieses Register enthält den 7-Bit-Offset-Korrekturwert für das 0°-Signal in Zweier-Komplement-Darstellung. Daraus folgt eine maximale Korrektur von ± 63.
Die Korrekturwerte können nur geschrieben werden, falls eines der Status-Bits D5 oder D6 im Status-Register 3 den Wert 0 hat.

Funktionsweise:
Zu den digitalen Werten des 0°-Signals (0 bis 1023) und 90°-Signals werden Offset-Korrekturwerte addiert. Bei einem Überlauf wird auf 1023 oder 0 begrenzt.

Bit	Funktion
D0 D1 D2 D3 D4 D5 D6	Offset-Korrekturwert für das 0°-Signal in Zweier-Komplement-Darstellung
D7 D8 D9 D10 D11 D12 D13 D14 D15	ohne Funktion

Das Offset-Register in den Zählerbausteinen ist **nicht** netzausfallsicher. Deshalb werden die Offset-Korrekturwerte in einem EEPROM im Baustein für die elektronischen Potentiometer gespeichert. Nach dem Einschalten müssen die Offset-Korrekturwerte vom EEPROM in die Offset-Register der Zählerbausteine geladen werden (siehe Prozeduren „store_offset“ und „load_offset“).

16h: Amplitude für das 0°-Signal (Lesezugriff)
Bei jedem Analog-Digital-Wandelvorgang wird das Ergebnis der Wandlung gespeichert. Vor dem Auslesen sollten durch Bit D4 im Kontroll-Register 2 die Werte eingefroren werden.

Bit	Funktion
D0 D1 D2 D3 D4 D5 D6 D7 D8 D9	Amplitude für das 0°-Signal
D10 D11 D12 D13 D14 D15	ohne Funktion

18h: Offset-Register für das 90°-Signal (Schreibzugriff)
Dieses Register enthält den 7-Bit-Offset-Korrekturwert für das 90°-Signal.
Die Beschreibung der Funktionsweise siehe „16h: Offset-Register für 0°-Signal“

Bit	Funktion
D0 D1 D2 D3 D4 D5 D6	Offset-Korrekturwert für das 90°-Signal in Zweier-Komplement-Darstellung
D7 D8 D9 D10 D11 D12 D13 D14 D15	ohne Funktion

18h: Amplitude für das 90°-Signal (Lesezugriff)
Bei jedem Analog-Digital-Wandelvorgang wird das Ergebnis der Wandlung gespeichert. Vor dem Auslesen sollten durch Bit D4 im Kontroll-Register 2 die Werte eingefroren werden.

Bit	Funktion
D0 D1 D2 D3 D4 D5 D6 D7 D8 D9	Amplitude für das 90°-Signal
D10 D11 D12 D13 D14 D15	ohne Funktion

1Ah: Timer-Register (Schreibzugriff)
In den Registern 1Ah und 1Bh wird der 13 Bit breite Timerwert abgelegt. Im Timer-Register können also Werte von 0 bis 8191 abgelegt werden. Die Zykluszeit wird in µs angegeben, wobei von der gewünschten Zykluszeit 1 µs abgezogen werden muss.

Beispiel:
Gewünschte Zykluszeit = 1 ms = 1 000 µs
Zu programmierender Wert = 1 000 – 1 = 999
Mit dem Beschreiben dieses Registers ist der Timer noch nicht gestartet. Dies erfolgt durch Setzen von Bit D2 im Initialisierungs-Register 1 (0Ch). Außerdem müssen die entsprechenden Bits in den Freigabe-Registern 12h, 13h oder 14h gesetzt werden.

Bit	Funktion
D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12	Timerwert
D13 D14 D15	ohne Funktion

1Ah: Lesezugriff ohne Funktion

1Ch: Kontroll-Register 2 (Schreibzugriff)

Bit	Funktion
D0 D1 D2 D3	Festen Wert ≥ 8 programmieren
D4	1 = Amplitude für 0°-Signal (16h) und 90°-Signal (18h) sowie Amplitudenwert-Register (10h High Byte) einfrieren. Um eine Änderung der Registerwerte während des Auslesens zu vermeiden, muss dieses Bit gesetzt werden. Ob die Registerwerte eingefroren sind, kann über Bit D4 im Status-Register 3 abgefragt werden.
D5	0
D6	1 = Erneuter Abruf über Daten-Register 0 möglich, ohne den Messwert vorher abzuholen. In dieser Betriebsart ist eine Änderung des Messwerts während des Lesens möglich.
D7	1 = Erneuter Abruf über Daten-Register 1 möglich, ohne den Messwert vorher abzuholen. In dieser Betriebsart ist eine Änderung des Messwerts während des Lesens möglich.
D8 D9 D10 D11 D12 D13 D14 D15	ohne Funktion

1Ch: Status-Register 3 (Lesezugriff)

Bit	Funktion
D0 D1 D2 D3	nicht lesbar
D4	1 = Amplitude für 0°-Signal (16h) und 90°-Signal (18h) sowie Amplitudenwert-Register (10h High Byte) sind eingefroren und können gelesen werden.
D5	0 = Das Offset-Register für das 0°-Signal ist geschrieben.
D6	0 = Das Offset-Registers für das 90°-Signal ist geschrieben.
D7	ohne Funktion

1Ch: Kennungs-Register (Lesezugriff)

Bit	Funktion
D8	
D9	
D10	
D11	Bausteinkennung: 8 (Zählerbaustein G26[1)])
D12	Id.-Nr. 282 150-01
D13	9 (Zählerbaustein G38[1)])
D14	Id.-Nr. 315 860-01
D15	13 (Zählerbaustein G49[1)])
	Id.-Nr. 333 033-01

[1)] Bausteinbezeichnung von HEIDENHAIN

1Eh: Kontroll-Register 3 (Schreibzugriff)

Bit	Funktion
D0	fester Wert 0
D1	
D2	ohne Funktion
D3	
D4	
D5	
D6	
D7	
D8	
D9	
D10	
D11	
D12	
D13	
D14	
D15	

1Eh: Status-Register 4 (Lesezugriff)

Bit	Funktion
D0	ohne Funktion
D1	logischer Pegel am Pin L0
D2	logischer Pegel am Pin L1
D3 D4 D5 D6 D7 D8 D9 D10 D11 D12 D13 D14 D15	ohne Funktion

Die IK 121 in DOS-Anwendungen

Schnell zur ersten Anzeige

Die IK 121 noch **nicht** einbauen.

- ➤ Legen Sie die Diskette mit der mitgelieferten Treiber-Software in das Laufwerk A Ihres PCs ein.
- ➤ Geben Sie folgende DOS-Befehle ein:
  >A:
  >INSTALL
  Nach dem Einstiegsbild zeigt der PC folgenden Hinweis an:

```
INSTALL IK 121
The IK 121-Interface-card should not be in
your PC!
Continue (y/n)?
```

- ➤ Geben Sie „y“ ein.
  Das Programm INSTALL zeigt nun eine Übersicht über die Port-Adressen.
  Freie Port-Adressen sind mit „free“ gekennzeichnet, belegte Port-Adressen mit „not free“.
- ➤ Wählen Sie eine freie Port-Adresse und geben Sie die zugehörige Nummer ein: z.B. „4“.
  INSTALL zeigt Ihnen nun, wie Sie die DIP-Schalter auf der IK 121 einstellen müssen.

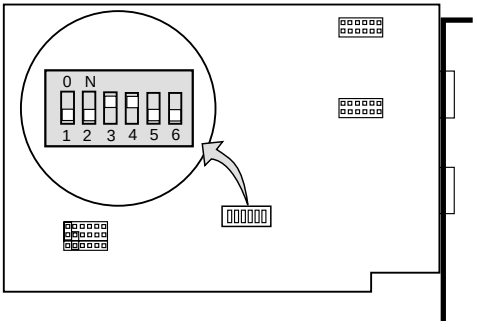
```
Chosen address:

   Nr:4 Adr.:0330 free


The DIP switch position on your interface card is:

   A4--off
   A5--off
   A6--on
   A7--on
   A8--off
   A9--off
                          on
                          A4          A9
                                        Press Return
```

➤ Stellen Sie die DIP-Schalter auf der IK 121 ein.



➤ Drücken Sie „Return“ am PC. Der PC speichert die gewählte Port-Adresse in der Datei IK 121.INI. Die Adresse steht für alle mitgelieferten Beispielprogramme – außer für die einfachen Beispiele SAMPLE32.PAS, SAMPLE48.PAS, SAMPLE32.C und SAMPLE48.C – zur Verfügung.

➤ Drücken Sie „Return“.
Das Programm INSTALL wird beendet.

Wenn INSTALL auf Ihren PC nicht läuft, dann gehen Sie wie folgt vor:

- Verwenden Sie die Standard-Einstellung (Adresse 0330h). Dies ist sowohl der Auslieferungszustand der Platine IK 121 als auch die Voreinstellung für die Software. Falls diese Adresse bei Ihrem PC belegt ist, dann
- schreiben Sie mit einem Text-Editor die von Ihnen gewählte Adresse (dezimal!) in die Datei IK121.INI oder
- starten Sie das Programm IK121.EXE und wählen Sie „File“, „Setup IK121“, „Setup board“, „Base address" und geben sie die gewünschte Adresse (dezimal!) ein. Speichern Sie die Adresse mit „Save setup“.

➤ Nehmen Sie die Diskette aus dem Laufwerk A heraus, stecken Sie den Netzstecker Ihres Computers aus und öffnen Sie das PC-Gehäuse. Wenn Sie nicht wissen, wie zusätzliche Karten in Ihren PC eingebaut werden, lesen Sie die Betriebsanleitung Ihres PCs.

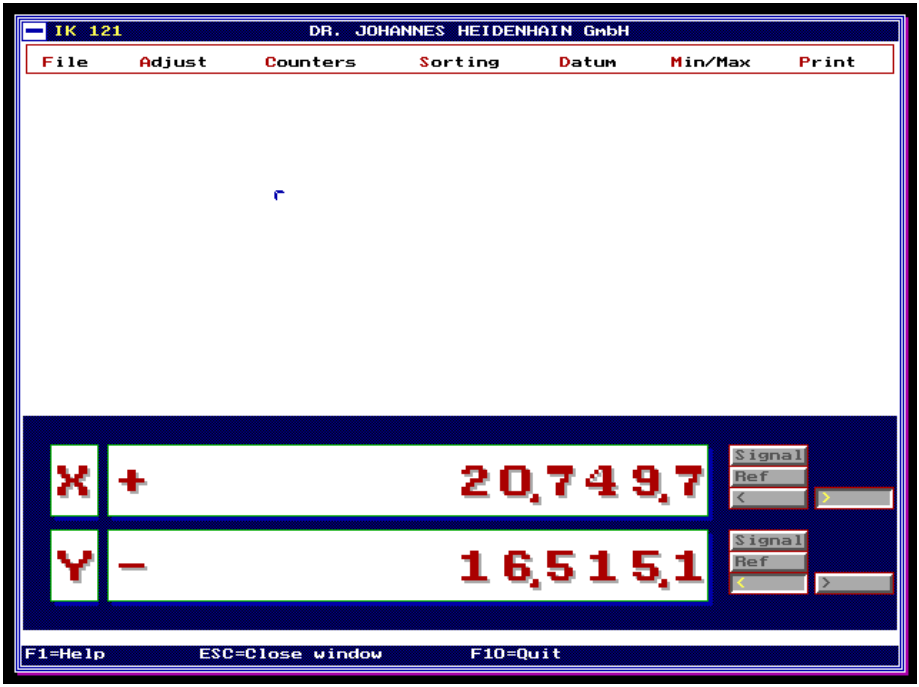
➤ Stecken Sie die IK 121 in Ihren PC ein (kurzer Slot ist ausreichend), befestigen Sie die Karte und montieren Sie das PC-Gehäuse wieder.

➤ Ein oder zwei Längenmessgeräte mit sinusförmigen Ausgangs-Signalen über HEIDENHAIN-Adapterkabel (Id.-Nr. 309 785-xx) an den Eingang „X1“ oder „X2“ anschließen.

➤ Netzkabel am PC anstecken und PC einschalten.

➤ Die Diskette mit der mitgelieferten Treiber-Software nochmals in das Laufwerk A Ihres PCs einstecken. Geben Sie folgende DOS-Befehle ein:
>A:
>IK121

➤ Der PC zeigt nach dem Einstiegsbild folgende Anzeige:



Das Programm IK 121.EXE zeigt die Positions-Istwerte der angeschlossenen Messgeräte am Bildschirm an.

Die Bedienung des Beispielprogramms erfolgt entweder über Maus oder Tastatur und ist dialoggeführt. Über Taste F1 können zusätzliche Erläuterungen für die einzelnen Menüpunkte angezeigt werden.

Die Programmierung der IK 121 in DOS-Anwendungen wird in dieser Beschreibung mit „TURBO PASCAL"- und „BORLAND C"-Beispielen gezeigt.
Die IK 121 verwendet Port-Adressen und kann mit jeder Programmiersprache programmiert werden, die Port-Adressen ansprechen kann.

Die Dateien IK121_0.*: Grundfunktionen zum Schreiben und Lesen der Register

Die im folgenden beschriebenen Funktionen finden Sie auf der mitgelieferten Diskette 1. Für „TURBO PASCAL“ unter dem Verzeichnis **\TP** in der Datei **IK121_0.PAS.** Für „BORLAND C" unter dem Verzeichnis **\BC** in der Datei **IK121_0.C** und der dazugehörenden Header-Datei **IK121_0.H.**
Die Zählerbausteine der IK 121 werden über Adress-Register adressiert (siehe „Adressierung“). Bevor ein Zählerbaustein angesprochen werden kann, muss das Adress-Register gesetzt werden. Die beiden Funktionen **write_g26** und **read_g26** schreiben und lesen die Daten-Register; sie sind die Grundfunktionen für das Arbeiten mit der IK 121.

Unterschied zwischen „TURBO PASCAL“ und „BORLAND C“

„TURBO PASCAL“ kennt Prozeduren (Funktionen ohne Rückgabewert) und Funktionen (mit Rückgabewert) während „BORLAND C“ nur Funktionen (mit oder ohne Rückgabewert) kennt. In dieser Beschreibung werden die „TURBO PASCAL“-Bezeichnungen „Prozedur“ und „Funktion“ verwendet.

Prozedur zum Schreiben in die Register

Die folgende Prozedur schreibt einen Wert in ein 16 Bit breites Register eines Zählerbausteins.

Prozedur: **write_g26**[1)]

Parameter

baseadr:	Grundadresse der IK 121, die über DIP-Schalter eingestellt wurde
axis:	1 = Achse 1, 2 = Achse 2
address:	Adresse B0 bis B4 des Registers des Zählerbausteins
datum:	Wert, der auf die Adresse geschrieben werden soll

[1)] g26 ist die HEIDENHAIN-Bezeichnung des Zählerbausteins

Die Prozedur in „TURBO PASCAL“

```
PROCEDURE
 write_g26(baseadr:word;axes,address:byte;datum:word);
  VAR adr_reg,adr_point,adr_gate : word;

  BEGIN
       (*Die letzten vier Bits der Karten-Adresse
       ausblenden*)
    baseadr:=baseadr and $0FF0;
       (*Adresse des Zähler-Bausteins B0 bis B4 bilden*)
    address:=address and $001F;

    (*Adress-Zeiger in Adress-Register laden*)
       (*Adresse des Adress-Registers bilden*)
    adr_reg:=baseadr or $0008;
       (*Inhalt des Adress-Registers R0 bis R2 bilden =
       Adresse des Zähler-Bausteins ohne B0 und B1*)
    adr_point:=address shr 2;
       (*Adress-Register laden*)
    portw[adr_reg]:=adr_point;

    (*Port-Adresse berechnen*)
    if axes=1 then
      adr_gate:=baseadr or (address and $03)
    else
      adr_gate:=(baseadr or $0004) or (address and $03);

       (* Daten in den Zähler-Baustein schreiben*)
    portw[adr_gate]:=datum;
  END;
```

Die Prozedur in „BORLAND C“

```
void write_g26 (unsigned int baseadr, unsigned char axis,
                unsigned int address, unsigned int datum)
{
unsigned int         adr_reg, adr_point, adr_gate;
   /*Die letzten vier Bit der Karten-Adresse ausblenden*/
baseadr = baseadr & 0x0FF0;
   /*Adresse des Zähler-Bausteins B0 bis B4 bilden*/
address = address & 0x1F;

/*Adress-Zeiger in Adress-Register laden*/
   /*Adresse des Adress-Registers bilden*/
adr_reg = baseadr | 0x0008;
   /*Inhalt des Adress-Registers R0 bis R2 bilden =
   Adresse des Zähler-Bausteins ohne B0 und B1*/
adr_point = address >> 2;
   /*Adress-Register laden*/
outpw (adr_reg, adr_point);

/*Port-Adresse berechnen*/
switch (axis)
       {
       case 1:
       adr_gate = baseadr | (address & 0x03);
       break;
       case 2:
       adr_gate = (baseadr | 0x0004) | (address & 0x03);
       break;
       }

   /* Daten in den Zähler-Baustein schreiben*/
outpw (adr_gate, datum);
}
```

Funktion zum Lesen der Register
Die folgende Funktion liest einen Wert von einem 16 Bit breiten Register eines Zählerbausteins.

Funktion: **read_g26**[1)]

Parameter

baseadr:	Grundadresse der IK 121, die über DIP-Schalter eingestellt wurde
axis:	1 = Achse 1, 2 = Achse 2
address:	Adresse B0 bis B4 des Registers des Zählerbausteins
Ergebnis:	Wert aus dem unter „axis“ und „address“ angegebenen Daten-Register als 16-Bit-Variable

Die Funktion in „TURBO PASCAL“

```
FUNCTION read_g26(baseadr:word;axes,address:byte):word;
  VAR adr_reg,adr_point,adr_gate : word;

  BEGIN
       (*Die letzten vier Bits der Karten-Adresse
       ausblenden*)
    baseadr:=baseadr and $0FF0;
       (*Adresse des Zähler-Bausteins B0 bis B4 bilden*)
    address:=address and $001F;

    (*Adress-Zeiger in Adress-Register laden*)
       (*Adresse des Adress-Registers bilden*)
    adr_reg:=baseadr or $0008;
       (*Inhalt des Adress-Registers R0 bis R2 bilden =
       Adresse des Zähler-Bausteins ohne B0 und B1*)
    adr_point:=address shr 2;
       (*Adress-Register laden*)
    portw[adr_reg]:=adr_point;

    (*Port-Adresse berechnen*)
    if axes=1 then
      adr_gate:=baseadr or (address and $03)
    else
      adr_gate:=(baseadr or $0004) or (address and $03);

       (*Daten vom Zähler-Baustein lesen*)
    read_g26:=portw[adr_gate];
  END;
```

[1)] g26 ist die HEIDENHAIN-Bezeichnung des Zählerbausteins

Die Funktion in „BORLAND C“

```
unsigned int read_g26 (unsigned int baseadr,
               unsigned char axis, unsigned int address)
{
unsigned int adr_reg, adr_point, adr_gate;
   /*Die letzten vier Bit der Karten-Adresse ausblenden*/
baseadr = baseadr & 0x0FF0;
   /*Adresse des Zähler-Bausteins B0 bis B4 bilden*/
address = address & 0x1F;

/*Adress-Zeiger in Adress-Register laden*/
   /*Adresse des Adress-Registers bilden*/
adr_reg = baseadr | 0x0008;
   /*Inhalt des Adress-Registers R0 bis R2 bilden =
   Adresse des Zähler-Bausteins ohne B0 und B1*/
adr_point = address >> 2;
   /*Adress-Register laden*/
outpw (adr_reg, adr_point);

/*Port-Adresse berechnen*/
switch (axis)
       {
       case 1:
       adr_gate = baseadr | (address & 0x03);
       break;
       case 2:
       adr_gate = (baseadr | 0x0004) | (address & 0x03);
       break;
       }
   /*Daten vom Zähler-Baustein lesen*/
return(inpw(adr_gate));
}
```

Einfache Funktionen für den Messwert-Abruf über Software

Prozeduren zum Speichern eines Messwerts
Die folgenden Prozeduren speichern den Zählerstand der gewünschten Achse im Daten-Register 0 (soft_l0) oder Daten-Register 1 (soft_l1).

Prozedur:	**soft_l0** **soft_l1**
Parameter	
baseadr:	Grundadresse der IK 121, die über DIP-Schalter eingestellt wurde
axis:	1 = Achse 1, 2 = Achse 2

Die Prozeduren in „TURBO PASCAL“

```
PROCEDURE soft_l0(baseadr:word;axis:byte);
  BEGIN
    write_g26(baseadr,axis,14,$0001);
  END;

PROCEDURE soft_l1(baseadr:word;axis:byte);
  BEGIN
    write_g26(baseadr,axis,14,$0002);
  END;
```

Die Prozedur in „BORLAND C“

```
void soft_l0 (unsigned int baseadr, unsigned char axis)
{
write_g26 (baseadr, axis, 0x0e, 0x0001);
}
void soft_l1 (unsigned int baseadr, unsigned char axis)
{
write_g26 (baseadr, axis, 0x0e, 0x0002);
}
```

Funktion zum Prüfen, ob der Messwert gespeichert wurde

Funktion: **latched**

Parameter

baseadr:	Grundadresse der IK 121, die über DIP-Schalter eingestellt wurde
axis:	1 = Achse 1, 2 = Achse 2
reg:	0 = Daten-Register 0, 1 = Daten-Register 1
Ergebnis:	false = Es wurde kein Messwert gespeichert true = Es wurde ein Messwert gespeichert

Die Funktion in „TURBO PASCAL“

```
FUNCTION latched(baseadr:word;axis,reg:byte):boolean;
  BEGIN
      case reg of
        0: latched:=
        (Read_g26(baseadr,axis,14) and $0001 ) = $0001;
        1: latched:=
        (Read_g26(baseadr,axis,14) and $0002 ) = $0002;
      end;
  END;
```

Die Funktion in „BORLAND C“

```
unsigned char latched (unsigned int baseadr,
                  unsigned char axis, unsigned char reg)
{
unsigned char result;
switch (reg)
       {
       case 0:
       result = (unsigned char)
                (read_g26 (baseadr, axis, 14) & 0x0001);
       break;
       case 1:
       result = (unsigned char)
                (read_g26 (baseadr, axis, 14) & 0x0002);
       break;
       }
return (result);
}
```

Prozedur zum wiederholten Prüfen, ob der Messwert gespeichert wurde

Die folgende Prozedur wiederholt solange die Abfrage nach einem Messwert, bis ein Messwert gespeichert wurde.

Prozedur: **poll_latch**

Parameter

baseadr: Grundadresse der IK 121, die über DIP-Schalter eingestellt wurde

axis: 1 = Achse 1, 2 = Achse 2

reg: 0 = Daten-Register 0, 1 = Daten-Register 1

Die Funktion in „TURBO PASCAL"

```
PROCEDURE poll_latch(baseadr:word;axis,reg:byte);
  BEGIN
      case reg of
        0: begin
             repeat
             until latched(baseadr,axis,0);
           end;
        1: begin
             repeat
             until latched(baseadr,axis,1);
           end;
      end;
  END;
```

Die Funktion in „BORLAND C"

```
void poll_latch (unsigned int baseadr,unsigned char axis,
                 unsigned char reg)
{
switch (reg)
       {
       case 0:
       while (latched (baseadr, axis, 0) == 0)
       ;
       break;

       case 1:
       while (latched (baseadr, axis, 1) == 0)
       ;
       break;
       }
}
```

Funktion zum Lesen eines 32-Bit-Messwerts
Die folgende Funktion liest einen 32 Bit breiten Messwert von einem Zählerbaustein.

Funktion: **read_count_value32**

Parameter

baseadr:	Grundadresse der IK 121, die über DIP-Schalter eingestellt wurde
axis:	1 = Achse 1, 2 = Achse 2
reg:	0 = Daten-Register 0, 1 = Daten-Register 1
Ergebnis:	„TURBO PASCAL“: Wert aus dem unter **axis** und **reg** angegebenen Daten-Registers als Integer-Variable des Typs **comp** „BORLAND C“: als Integer-Variable des Typs **long**

Die Funktion in „TURBO PASCAL“

```
FUNCTION
 read_count_value32(baseadr:word;axis,reg:byte):comp;
  TYPE
    vartype = (li,by);
    mapper = record
               case wert:vartype of
                 li : (field0:longint);
                 by : (field1:array[0..1] of word);
               end;
  VAR
    buffer : mapper;
  BEGIN
    case reg of
     0 : begin
           buffer.field1[0]:=read_g26(baseadr,axis,0);
           buffer.field1[1]:=read_g26(baseadr,axis,2);
         end;
     1 : begin
           buffer.field1[0]:=read_g26(baseadr,axis,6);
           buffer.field1[1]:=read_g26(baseadr,axis,8);
         end;
     end;
     read_count_value32:=buffer.field0;
  END;
```

Die Funktion in „BORLAND C“

```
long read_count_value32 (unsigned int baseadr,
                  unsigned char axis, unsigned char reg)
{
union  mapper
        {
        long field0;
        unsigned int field1[2];
        }buffer;

switch (reg)
       {
       case 0:
       buffer.field1[0] = read_g26 (baseadr, axis, 0);
       buffer.field1[1] = read_g26 (baseadr, axis, 2);
       break;
       case 1:
       buffer.field1[0] = read_g26 (baseadr, axis, 6);
       buffer.field1[1] = read_g26 (baseadr, axis, 8);
       break;
       }
return (buffer.field0);
}
```

Funktion zum Lesen eines 48-Bit-Messwerts
Die folgende Funktion liest einen 48 Bit breiten Messwert von einem Zählerbaustein.

Funktion: **read_count_value48**

Parameter

baseadr:	Grundadresse der IK 121, die über DIP-Schalter eingestellt wurde
axis:	1 = Achse 1, 2 = Achse 2
reg:	0 = Daten-Register 0, 1 = Daten-Register 1
Ergebnis:	„TURBO PASCAL": Wert aus dem unter **axis** und **reg** angegebenen Daten-Registers als Integer-Variable des Typs **comp** „BORLAND C": als Floating-Point-Variable des Typs **double**

Die Funktion in „TURBO PASCAL"

```
FUNCTION
 read_count_value48(baseadr:word;axis,reg:byte):comp;
  TYPE
    vartype = (li,by);
    mapper = record
               case wert:vartype of
                 li : (field0:comp);
                 by : (field1:array[0..3] of word);
               end;
  VAR
    buffer : mapper;
  BEGIN
    case reg of
     0 : begin
           buffer.field1[0]:=read_g26(baseadr,axis,0);
           buffer.field1[1]:=read_g26(baseadr,axis,2);
           buffer.field1[2]:=read_g26(baseadr,axis,4);
           if (buffer.field1[2] and $8000)=$8000 then
           buffer.field1[3]:=
                          $FFFF else buffer.field1[3]:=0;
         end;
     1 : begin
           buffer.field1[0]:=read_g26(baseadr,axis,6);
           buffer.field1[1]:=read_g26(baseadr,axis,8);
           buffer.field1[2]:=read_g26(baseadr,axis,10);
           if (buffer.field1[2] and $8000)=$8000 then
           buffer.field1[3]:=
                          $FFFF else buffer.field1[3]:=0;
         end;
     end;
     read_count_value48:=buffer.field0;
  END;
```

Die Funktion in „BORLAND C“

```
double read_count_value48 (unsigned int baseadr,
                  unsigned char axis, unsigned char reg)
{
unsigned int field[3];
double count_value48;

switch (reg)
       {
       case 0:
       field[0] = read_g26 (baseadr, axis, 0);
       field[1] = read_g26 (baseadr, axis, 2);
       field[2] = read_g26 (baseadr, axis, 4);
       break;
       case 1:
       field[0] = read_g26 (baseadr, axis, 6);
       field[1] = read_g26 (baseadr, axis, 8);
       field[2] = read_g26 (baseadr, axis, 10);
       break;
       }

if (field[2] && 0x8000)
       count_value48 = (double)((field[0]-65535) +
       65536.0*(field[1]-65535)+
       4294967296.0*(field[2]-65535)-1);
       else
       count_value48 = (double)(field[0] +
       65536.0*field[1] +
       4294967296.0*field[2]);


return (count_value48);
}
```

Einfaches Programm für den Messwert-Abruf über Software

In den folgenden Programm-Beispielen werden die vorher definierten Funktionen eingesetzt. Die Beispiele finden Sie auf der mitgelieferten Diskette 1.
Für „TURBO PASCAL“ unter dem Verzeichnis **\TP** in der Datei **SAMPLE32.PAS** und **SAMPLE48.PAS.**
Für „BORLAND C“ unter dem Verzeichnis **\BC** in der Datei **SAMPLE32.C** und **SAMPLE48.C.**
Der Zählerstand wird in diesem Beispiel in Millimetern am Bildschirm angezeigt. Selbstverständlich kann die IK 121 auch Winkelgrade anzeigen. Die Umrechnung zeigen die beiden folgenden Formeln.

Zählerstand in Millimeter umrechnen

$$\text{Wert [mm]} = \text{Zählerstand} \bullet \frac{\text{Teilungsperiode [mm]}}{1024}$$

Beispiel: Teilungsperiode = 20 µm

$$\text{Wert [mm]} = \text{Zählerstand} \bullet \frac{0{,}020\ \text{[mm]}}{1024}$$

Zählerstand in Grad umrechnen

$$\text{Wert [°]} = \frac{\text{Zählerstand} \bullet 360\ [°]}{1024 \bullet \text{Striche / Umdr.}}$$

Beispiel: Striche/Umdr. = 36 000

$$\text{Wert [°]} = \frac{\text{Zählerstand} \bullet 360\ [°]}{1024 \bullet 36000}$$

Beispiele in „TURBO PASCAL“: „Messwert-Abruf über Software“

Die folgenden Beispiele SAMPLE32.PAS und SAMPLE48.PAS zeigen die Anwendung der vorher beschriebenen Prozeduren und Funktionen. Diese sind in der Datei **IK121_0.TPU** deklariert und definiert (alle Dateien befinden sich auf der mitgelieferten Diskette 1).

Die Datei IK121_0.PAS muss vor dem Kompilieren von SAMPLE0.PAS zur „unit“ IK121_0.TPU kompiliert werden.

Da dieses einfache Beispiel die Adresse der IK 121 nicht aus der Datei IK 121.INI liest, wird die Adresse als Konstante „base_address“ definiert.

```
program sample32;
{-------------------------------------------------------
 DR. JOHANNES HEIDENHAIN GmbH, Traunreut, Germany
 Einfaches Programm für die IK 121 zur Anzeige
 von zwei Achsen. Breite des abgerufenen
 Messwertes: 32 Bit.
 V 1.01
 April 1995
 -------------------------------------------------------}
{$N+,E+}
{$V+}
{$R+}
USES crt,ik121_0;
CONST
  base_address = $330;
VAR
  c_value_0, c_value_1  : comp;
BEGIN
clrscr;
   (*Karte initialisieren im Betrieb mit Interpolation, Achse 1*)
write_g26 (base_address, 1, $0c, $0001);
   (*Karte initialisieren im Betrieb mit Interpolation, Achse 2*)
write_g26 (base_address, 2, $0c, $0001);
   (*Fehler löschen, Zähler starten, Achse 1*)
write_g26 (base_address, 1, $0e, $0048);
   (*Fehler löschen, Zähler starten, Achse 2*)
write_g26 (base_address, 2, $0e, $0048);
   (*Kontroll-Register 2 laden, Achse 1*)
write_g26 (base_address, 1, $1c, $0008);
   (*Kontroll-Register 2 laden, Achse 2*)
write_g26 (base_address, 2, $1c, $0008);
REPEAT
   (*Software-Abruf in Register 0, Achse 1*)
soft_l0 (base_address, 1);
   (*Software-Abruf in Register 0, Achse 2*)
soft_l0 (base_address, 2);
   (*Messwert von Achse 1 gespeichert?*)
poll_latch (base_address, 1, 0);
   (*Messwert lesen, Achse 1*)
c_value_0:= read_count_value32 (base_address, 1, 0);
```

```
   (*Messwert von Achse 2 gespeichert?*)
poll_latch (base_address, 2, 0);
 (*Messwert lesen, Achse 2*)
c_value_1:= read_count_value32 (base_address, 2, 0);
(*Messwert am Bildschirm anzeigen*)
gotoxy(1,10);
write(c_value_0*0.02/1024:16:4,
       c_value_1*0.02/1024:16:4);
UNTIL KEYPRESSED;
END.

program sample48;
{-------------------------------------------------------
 DR. JOHANNES HEIDENHAIN GmbH, Traunreut, Germany
 Einfaches Programm für die IK 121 zur Anzeige
 von zwei Achsen. Breite des abgerufenen
 Messwertes: 48 Bit.
 V 1.01
 April 1995
 -------------------------------------------------------}
{$N+,E+}
{$V+}
{$R+}
USES crt,ik121_0;
CONST
  base_address = $330;
VAR
  c_value_0, c_value_1  : comp;
BEGIN
clrscr;
   (*Karte initialisieren im Betrieb mit Interpolation, Achse 1*)
write_g26 (base_address, 1, $0c, $0041);
   (*Karte initialisieren im Betrieb mit Interpolation, Achse 2*)
write_g26 (base_address, 2, $0c, $0041);
   (*Fehler löschen, Zähler starten, Achse 1*)
write_g26 (base_address, 1, $0e, $0048);
   (*Fehler löschen, Zähler starten, Achse 2*)
write_g26 (base_address, 2, $0e, $0048);
   (*Kontroll-Register 2 laden, Achse 1*)
write_g26 (base_address, 1, $1c, $0008);
   (*Kontroll-Register 2 laden, Achse 2*)
write_g26 (base_address, 2, $1c, $0008);
REPEAT
   (*Software-Abruf in Register 0, Achse 1*)
soft_l0 (base_address, 1);
   (*Software-Abruf in Register 0, Achse 2*)
soft_l0 (base_address, 2);
   (*Messwert von Achse 1 gespeichert?*)
poll_latch (base_address, 1, 0);
   (*Messwert lesen, Achse 1*)
c_value_0:= read_count_value48 (base_address, 1, 0);
   (*Messwert von Achse 2 gespeichert?*)
poll_latch (base_address, 2, 0);
 (*Messwert lesen, Achse 2*)
c_value_1:= read_count_value48 (base_address, 2, 0);
```

```
(*Messwert am Bildschirm anzeigen*)
gotoxy(1,10);
write(c_value_0*0.02/1024:16:4,
       c_value_1*0.02/1024:16:4);
UNTIL KEYPRESSED;
END.
```

Beispiele in „BORLAND C“: „Messwert-Abruf über Software“

Die folgenden Beispiele **SAMPLE32.C** und **SAMPLE48.C** zeigen die Anwendung der vorher beschriebenen Funktionen. Diese sind in den Dateien IK121_0.H und IK121_0.C deklariert und definiert (alle Dateien befinden sich auf der mitgelieferten Diskette 1). Zur Kompilierung muss mit IK121_0.C und SAMPLE32.C oder SAMPLE48.C ein Projekt gebildet werden. Da dieses einfache Beispiel die Adresse der IK 121 nicht aus der Datei IK121.INI liest, wird die Adresse als Konstante „base_address“ definiert.

```
/*----------------------SAMPLE32.C----------------------
 DR. JOHANNES HEIDENHAIN GmbH, Traunreut, Germany
 Einfaches Programm für die IK 121 zur Anzeige
 von zwei Achsen. Breite des abgerufenen
 Messwertes: 32 Bit.
 V 1.01
 April 1995
 Projekt-Dateien:      IK121_0.C, SAMPLE32.C
 Include-Datei:        IK121_0.H
 ------------------------------------------------------*/
#include <stdio.h>
#include <conio.h>
#include “ik121_0.h"
#define base_address 0x0330
int main()
{
double c_value_0, c_value_1;
cls;
   /*Karte initialisieren im Betrieb mit Interpolation, Achse 1*/
write_g26 (base_address, 1, 0x0c, 0x0001);
   /*Karte initialisieren im Betrieb mit Interpolation, Achse 2*/
write_g26 (base_address, 2, 0x0c, 0x0001);
   /*Fehler löschen, Zähler starten, Achse 1*/
write_g26 (base_address, 1, 0x0e, 0x0048);
   /*Fehler löschen, Zähler starten, Achse 2*/
write_g26 (base_address, 2, 0x0e, 0x0048);
   /*Kontroll-Register 2 laden, Achse 1*/
write_g26 (base_address, 1, 0x1c, 0x0008);
   /*Kontroll-Register 2 laden, Achse 2*/
write_g26 (base_address, 2, 0x1c, 0x0008);
   /*Cursor ausschalten*/
_setcursortype (_NOCURSOR);
while(!kbhit())
       {
```

```
          /*Software-Abruf in Register 0, Achse 1*/
       soft_l0 (base_address, 1);
          /*Software-Abruf in Register 0, Achse 2*/
       soft_l0 (base_address, 2);
          /*Messwert von Achse 1 gespeichert?*/
       poll_latch (base_address, 1, 0);
         /*Messwert lesen, Achse 1*/
       c_value_0 = (double)read_count_value32
                              (base_address, 1, 0);
          /*Messwert von Achse 2 gespeichert?*/
       poll_latch (base_address, 2, 0);
         /*Messwert lesen, Achse 2*/
       c_value_1 = (double)read_count_value32
                              (base_address, 2, 0);
        /*Messwert am Bildschirm anzeigen*/
       printf("\r\t%16.4f\t%16.4f",c_value_0*0.02/1024,
                              c_value_1*0.02/1024);
       }
   /*Cursor wieder einschalten*/
_setcursortype (_NORMALCURSOR);
return (0);
}


/*----------------------SAMPLE48.C----------------------
 DR. JOHANNES HEIDENHAIN GmbH, Traunreut, Germany
 Einfaches Programm für die IK 121 zur Anzeige
 von zwei Achsen. Breite des abgerufenen
 Messwertes: 48 Bit.
 V 1.01
 April 1995
 Projekt-Dateien:      IK121_0.C, SAMPLE48.C
 Include-Datei:        IK121_0.H
 ------------------------------------------------------*/
#include <stdio.h>
#include <conio.h>
#include "ik121_0.h"
#define base_address 0x0330
int main()
{
double c_value_0, c_value_1;
cls;
   /*Karte initialisieren im Betrieb mit Interpolation, Achse 1*/
write_g26 (base_address, 1, 0x0c, 0x0041);
  /*Karte initialisieren im Betrieb mit Interpolation, Achse 2*/
write_g26 (base_address, 2, 0x0c, 0x0041);
   /*Fehler löschen, Zähler starten, Achse 1*/
write_g26 (base_address, 1, 0x0e, 0x0048);
   /*Fehler löschen, Zähler starten, Achse 2*/
write_g26 (base_address, 2, 0x0e, 0x0048);
   /*Kontroll-Register 2 laden, Achse 1*/
write_g26 (base_address, 1, 0x1c, 0x0008);
   /*Kontroll-Register 2 laden, Achse 2*/
write_g26 (base_address, 2, 0x1c, 0x0008);
   /*Cursor ausschalten*/
_setcursortype (_NOCURSOR);
while(!kbhit())
       {
```

```
       /*Software-Abruf in Register 0, Achse 1*/
       soft_l0 (base_address, 1);
           /*Software-Abruf in Register 0, Achse 2*/
       soft_l0 (base_address, 2);
           /*Messwert von Achse 1 gespeichert?*/
       poll_latch (base_address, 1, 0);
          /*Messwert lesen, Achse 1*/
       c_value_0 = read_count_value48 (base_address, 1, 0);
           /*Messwert von Achse 2 gespeichert?*/
       poll_latch (base_address, 2, 0);
          /*Messwert lesen, Achse 2*/
       c_value_1 = read_count_value48 (base_address, 2, 0);
       /*Messwert am Bildschirm anzeigen*/
       printf("\r\t%16.4f\t%16.4f",c_value_0*0.02/1024,
                                  c_value_1*0.02/1024);
       }
   /*Cursor wieder einschalten*/
_setcursortype (_NORMALCURSOR);
return (0);
}
```

Datei IK121_1.PAS: Funktionen für ein RAM-Speicher-Modell in „TURBO PASCAL“

Die im folgenden beschriebenen Datenstrukturen und Funktionen für „TURBO PASCAL“ finden Sie auf der mitgelieferten Diskette 1 unter dem Verzeichnis **\TP** in der Datei IK121_1.PAS und IIC.PAS (diese Datei wird beim Kompilieren in IK121_1.PAS eingebunden). Diese Dateien werden als **unit** mit Hilfe des Befehls **uses** in weitere Anwendungsprogramme eingebunden. In der Datei IK121_1.PAS wird mit Hilfe von Datenstrukturen (in „TURBO PASCAL“: record) ein RAM-Speicher-Modell der Register der IK 121 aufgebaut.
Die Daten des RAM-Speicher-Modells werden mit Hilfe der Prozeduren **init_handler** und **comm_handler** in die Port-Adressen der IK 121 geschrieben.
Außerdem sind nützliche Funktionen zum Initialisieren der Karte, zum Lesen der Messwerte, zur Interrupt-Programmierung und zum Abgleich der Messgerät-Signale enthalten.

Definition der Datentstrukturen

Record: softcommand

Diese Struktur ist das RAM-Speicher-Abbild von Kontroll-Register 1 (Schreibzugriff auf 0Eh).

Datenfelder

start:	Zähler starten
stop:	Zähler stoppen
clear:	Zähler löschen
latch0:	Software-Abruf: Messwert in Daten-Register 0
latch1:	Software-Abruf: Messwert in Daten-Register 1
latch2:	Software-Abruf mit Sonderfunktion
clrfrg:	Messgerätfehler löschen (Frequenzüberschreitung)
clrstat:	Amplitudenwert-Register löschen

Record: refcommand
Diese Struktur ist das RAM-Speicher-Abbild des Referenzmarken-Registers (Schreibzugriff auf 10h). Beim Überfahren der Referenzmarke können folgende Funktionen ausgeführt werden.
Datenfelder

ristart:	Zähler starten
ristop:	Zähler stoppen
riclear:	Zähler löschen
riclear2:	Zähler mit dem Überfahren jeder Referenzmarke löschen
rilatch:	Messwert abrufen
rilatch2:	Messwert mit dem Überfahren der 2. Referenzmarke abrufen

Record: initlatch
Diese Struktur ist das RAM-Speicher-Abbild des Freigabe-Registers für den Messwert-Abruf (Schreibzugriff auf 12h).
Datenfelder

en_L0_reg0:	Freigabe L0 für Daten-Register 0
en_latch2_reg0:	Freigabe des „Software-Abrufs in alle Daten-Register“ für Daten-Register 0
en_timer_reg0:	Freigabe des Software-Abrufs über Timer für Daten-Register 0
en_L1_reg1:	Freigabe L1 für Daten-Register 1
en_latch2_reg1:	Freigabe des „Software-Abrufs in alle Daten-Register“ für -Daten-Register 1
en_timer_reg1:	Freigabe des Software-Abrufs über Timer für Daten-Register 1

Record: initsync
Diese Struktur ist das RAM-Speicher-Abbild des Freigabe-Registers für Achs-Kaskadierung (Schreibzugriff auf 13h). Die folgenden Datenfelder sind nur für die Achse 1 sinnvoll.
Datenfelder

en_L0_axis2:	Freigabe des externen Abrufsignals X3.L0 zur 2. Achse
en_latch2_axis2:	Freigabe des Software-Abrufs mit Sonderfunktionen zur 2. Achse
en_timer_axis2:	Freigabe des Timerstrobes zur 2. Achse

Record: **initmain**

Diese Struktur ist das RAM-Speicher-Abbild der Initialisierungs-Register 1 und 2 (Schreibzugriff auf 0Ch und 0Dh).

Datenfelder

mode1024: 0 = Betrieb als Periodenzähler (ohne Interpolation, Datenbits D0 bis D9 sind nicht definiert).
1 = Messwert wird aus dem Wert des Periodenzählers und aus dem Interpolationswert gebildet.

en_timer: 0 = Timer nullen und stoppen
1 = Timer starten

en_48Bit: 0 = Betriebsart: 32-Bit-Zähler
1 = Betriebsart: 48-Bit-Zähler

onefold: Zähler zählen 1 Flanke pro Signalperiode

twofold: Zähler zählen 2 Flanken pro Signalperiode

fourfold: Zähler zählen 4 Flanken pro Signalperiode

arcmode: 0 = Linear-Zählweise
1 = Winkel-Zählweise

arc180000: 0 = Winkel-Zählweise 17 999 bis –18 000
1 = Winkel-Zählweise 179 999 bis –180 000

sel_ri_1st: 0 = 1. Referenzmarke speichert in Daten-Register 0
1 = 1. Referenzmarke speichert in Daten-Register 1

sel_ri_2nd: 0 = 2. Referenzmarke speichert in Daten-Register 0
1 = 2. Referenzmarke speichert in Daten-Register 1

Record: **initintrpt**

Diese Struktur ist das RAM-Speicher-Abbild des Interrupt-Freigabe-Registers (Schreibzugriff auf 14h).

Datenfelder

reg0: Freigabe des Interrupts 0 beim Messwert-Abruf über Daten-Register 0

reg1: Freigabe des Interrupts 1 beim Messwert-Abruf über Daten-Register 1

timer: Freigabe des Interrupts 2 für Timerstrobe

port: Interrupt wird erzeugt

Record: **intstate**

Diese Struktur ist das RAM-Speicher-Abbild der Interrupt-Status-Register 1 und 2 (Lesezugriff auf 14h und 15h).

Datenfelder

L0:	Interrupt 0 ist aktiv, es erfolgte ein Messwert-Abruf über Daten-Register 0
L1:	Interrupt 1 ist aktiv, es erfolgte ein Messwert-Abruf über Daten-Register 1
timer:	Interrupt 2 ist aktiv, es erfolgte ein Messwert-Abruf über den Timerstrobe
pendl0:	Interrupt 0 steht an, wird aber noch nicht ausgeführt
pendl1:	Interrupt 1 steht an, wird aber noch nicht ausgeführt
pendtimer:	Interrupt 2 steht an, wird aber noch nicht ausgeführt

Record: **countstate**

Diese Struktur ist das RAM-Speicher-Abbild der Statusregister 1 und 2 (Lesezugriff auf 0Eh und 0Fh).

Datenfelder

latch0:	Messwert in Daten-Register 0 ist bereit
latch1:	Messwert in Daten-Register 1 ist bereit
stop:	Zähler ist gestoppt
sda:	PC-Bus-Leitung SDA (Eingang)
error_frq:	Messgerätfehler (Frequenzüberschreitung)
ref_activ:	Anfahren der Referenzmarke ist aktiv

Record: **signalstate**

Diese Struktur ist das RAM-Speicher-Abbild des Amplitudenwert-Registers (Lesezugriff auf 11h), des Amplituden-Registers für das 0°-Signal (Lesezugriff auf 16h und 17 h) sowie des Amplituden-Registers für das 90°-Signal (Lesezugriff auf 18h und 19h).

Datenfelder

ad00:	Amplitude für das 0°-Signal
ad90:	Amplitude für das 90°-Signal
amp_act:	Aktuelle Amplitude
amp_min:	Minimalwert der Amplitude

Zeiger: **g26_pointr**

Zeiger auf eine Struktur **g26_record**

Record: **g26_record**
Diese Struktur enthält den kompletten Datensatz des RAM-Speicher-Abbilds der Register einer Achse.

Zeiger: **ik121_pointr**
Zeiger auf eine Struktur ik121_record

Record: **ik121_record**
Array von Zeigern **g26_pointr**

Record: **storage**
Array von Strukturen **g26_record** für die Datei IK121.INI zum Speichern der Initialisierungswerte

Prozeduren und Funktionen
Funktion: **look_for_IK121**
Diese Funktion testet, ob die Hardware der IK 121 vorhanden ist.
Ergebnis: true, falls IK121 vorhanden ist
false, falls IK 121 nicht vorhanden ist
Prototyp:
FUNCTION look_for_IK121 (board:IK121_pointr):boolean;

Prozedur: **init_IK121**
Diese Prozedur initialisiert die IK 121.
Prototyp:
PROCEDURE init_ik121 (board:IK121_pointr);

Prozedur: **init_handler**
Diese Prozedur schreibt die Initialisierungs-Daten vom RAM-Speicher-Modell in die Port-Adressen der IK 121 mit Hilfe folgender Funktionen: g26_main, g26_latch, g26_sync und g26_int.
Prototyp: PROCEDURE init_handler (pointr:g26_pointr);

Prozedur: **comm_handler**
Diese Prozedur schreibt Befehle vom RAM-Speicher-Modell in die Port-Adressen der IK 121 mit Hilfe folgender Funktionen: g26_soft, g26_ref.
Prototyp: PROCEDURE comm_handler
(pointr:g26_pointr);

Prozedur: **read_adr**
Diese Prozedur liest die Adresse der IK121 aus der Datei IK121.INI
Prototyp: PROCEDURE read_adr (board:ik121_pointr);

Prozedur: **write_adr**
Diese Prozedur schreibt die Adresse der IK 121 in die Datei IK121.INI
Prototyp: PROCEDURE write_adr (board:ik121_pointr);

Prozedur: **read_signal_status**
Diese Prozedur liest die Daten für die Struktur **signalstate** aus den Registern 11h, 16h, 17h, 18h und 19h.
Prototyp: PROCEDURE read_signal_status (pointr:g26_pointr);

Prozedur: **read_count_status**
Diese Prozedur liest die Daten für die Struktur **countstate** aus den Registern 0Eh und 0Fh.
Prototyp: PROCEDURE read_count_status (pointr:g26_pointr);

Prozedur: **read_int_status**
Diese Prozedur liest die Daten für die Struktur **intstate** aus den Registern 14h und 15h.
Prototyp: PROCEDURE read_int_status (pointr:g26_pointr);

Prozedur: **soft_latch0**
Diese Prozedur speichert einen Messwert in Daten-Register 0.
Prototyp: PROCEDURE soft_latch0 (pointr:g26_pointr);

Prozedur: **soft_latch1**
Diese Prozedur speichert einen Messwert in Daten-Register 1.
Prototyp: PROCEDURE soft_latch1 (pointr:g26_pointr);

Prozedur: **read_reg0**
Diese Prozedur liest einen Messwert vom Daten-Register 0.
Prototyp: PROCEDURE read_reg0 (pointr:g26_pointr);

Prozedur: **read_reg1**
Diese Prozedur liest einen Messwert vom Daten-Register 1.
Prototyp: PROCEDURE read_reg1 (pointr:g26_pointr);

Prozedur: **poll_reg0**
Diese Prozedur fragt solange das Statusbit D0 in Register 0Eh ab, bis ein Messwert in Daten-Register 0 gespeichert ist.
Prototyp: PROCEDURE poll_reg0 (pointr:g26_pointr);

Prozedur: **poll_reg1**
Diese Prozedur fragt solange das Statusbit D1 in Register 0Eh ab, bis ein Messwert in Daten-Register 1 gespeichert ist.
Prototyp: PROCEDURE poll_reg1 (pointr:g26_pointr);

Prozedur: **en_int**
Diese Prozedur gibt einen Interrupt frei.
Prototyp: PROCEDURE en_int (Intrpt:byte);

Prozedur: **dis_int**
Diese Prozedur sperrt einen Interrupt.
Prototyp: PROCEDURE dis_int (Intrpt:byte);

Prozedur: **clear_int**
Diese Prozedur setzt einen Interrupt zurück.
Prototyp: PROCEDURE clear_int;

Prozedur: **write_offset**
Diese Prozedur schreibt Offset-Korrekturwerte in die Offset-Register der Zählerbausteine (nicht netzausfallsicher!). Zum netzausfallsicheren Speichern der Offset-Korrekturwerte und zum Zurückspeichern in die Offset-Register eignen sich die Prozeduren „store_offset“ und „load_offset“.
Prototyp: PROCEDURE write_offset
(baseadr:word;axis:byte;offx,offy:integer);

Die folgenden Funktionen sind in der Datei IIC.PAS definiert. Nach Kompilieren zu einer „unit“ wird diese Datei in IK121_1.PAS eingebunden.

Prozedur: **load_offset**
Diese Prozedur liest den Offset aus den Offset-Registern.
Prototyp: PROCEDURE load_offset
(board:IK121_pointr;var error:boolean);

Prozedur: **store_offset**
Diese Prozedur speichert einen Korrekturwert in den Offset-Registern.
Prototyp: PROCEDURE store_offset
(board:IK121_pointr;var error:boolean)

Prozedur: **poti_default**
Diese Prozedur stellt die Potentiometer in die Null-Stellung
Prototyp: PROCEDURE poti_default
(pointr:ik121_pointr;var error:boolean);

Funktion: **read_phasepoti**
Diese Funktion liest die Stellung des Potentiometers für die Phasenlage.
Prototyp: FUNCTION read_phasepoti
(pointr:ik121_pointr;axis:byte;var error:boolean):byte;

Funktion: **read_sympoti**
Diese Funktion liest die Stellung des Potentiometers für die Symmetrie.
Prototyp: FUNCTION read_sympoti
(pointr:ik121_pointr;axis:byte;var error:boolean):byte;

Prozedur: **write_phasepoti**
Diese Prozedur stellt das Potentiometer für die Phasenlage auf einen vorgegebenen Wert.
Prototyp: PROCEDURE write_phasepoti
(pointr:ik121_pointr;axis,wert:byte;var error:boolean);

Prozedur: **write_sympoti**
Diese Prozedur stellt das Potentiometer für die Symmetrie auf einen vorgegebenen Wert.
Prototyp: PROCEDURE write_sympoti
(pointr:ik121_pointr;axis,wert:byte;var error:boolean);

Prozedur: **write_offset00**
Diese Prozedur schreibt einen Korrekturwert in das Offset-Register für das 0°-Signal.
Prototyp: PROCEDURE write_offset00
(pointr:ik121_pointr;axis,value:integer);

Prozedur: **write_offset90**
Diese Prozedur schreibt einen Korrekturwert in das Offset-Register für das 90°-Signal.
Prototyp: PROCEDURE write_offset90
(pointr:ik121_pointr;axis,value:integer);

Prozedur: **turn_phasepoti**
Diese Prozedur verstellt das Potentiometer für die Phasenlage.
Prototyp: PROCEDURE turn_phasepoti
(pointr:ik121_pointr;axis,turns:byte;updown:boolean;var error:boolean);

Prozedur: **turn_sympoti**
Diese Prozedur verstellt das Potentiometer für die Symmetrie.
Prototyp: PROCEDURE turn_sympoti (pointr:ik121_pointr;axis,turns:byte;updown:boolean;var error:boolean);

Prozedur: **turn_offsetdg00**
Diese Prozedur verändert den Korrekturwert für das Offset-Register für das 0°-Signal.
Prototyp: PROCEDURE turn_offsetdg00 (pointr:ik121_pointr;axis,turns:byte;updown:boolean)

Prozedur: **turn_offsetdg90**
Diese Prozedur verändert den Korrekturwert für das Offset-Register für das 90°-Signal.
Prototyp: PROCEDURE turn_offsetdg90 (pointr:ik121_pointr;axis,turns:byte;updown:boolean)

Prozedur: **store_potis**
Diese Prozedur speichert die Potentiometerstellungen.
Prototyp: PROCEDURE store_potis (pointr:ik121_pointr;var error:boolean);

Prozedur: **rom_write**
Diese Prozedur schreibt in den frei verfügbaren EEPROM-Speicher.
Prototyp: PROCEDURE rom_write (pointr:ik121_pointr;adr;data :byte;var error : boolean);

Funktion: **rom_read**
Diese Funktion liest den frei verfügbaren EEPROM-Speicher.
Prototyp: FUNCTION rom_read (pointr:ik121_pointr;adr:byte;var error:boolean) :byte;

Anwendungsprogramme mit dem RAM-Speicher-Modell in „TURBO PASCAL“

Die ausführbaren EXE-Dateien der folgenden Beispiele sind im Hauptverzeichnis der mitgelieferten Diskette 1 gespeichert. Die Quelldateien befinden sich im Unterverzeichnis **\TP.** Die Programme benötigen einen BORLAND-Grafik-Treiber (BORLAND Gaphics Interface = *.BGI). Im Lieferumfang der IK 121 ist der Grafik-Treiber EGAVGA.BGI enthalten. Dieser muss in dem gleichen Verzeichnis wie die Beispielprogramme gespeichert sein.

SAMPLE1.EXE

Das Beispiel SAMPLE1.EXE ist ein einfaches Anwendungsprogramm zur Anzeige der Inhalte der Daten-Register 0 von Achse 1 und Achse 2.

Quellcode:	SAMPLE1.PAS	
Units:	IK121_0.TPU	Grundfunktionen
	IK121_1.TPU	Funktionen für ein RAM-Speicher-Modell

SAMPLE2.EXE

Das Beispiel SAMPLE2.EXE zeigt die Anwendung der Interrupt-Programmierung. Dabei wird IRQ14 für Int1 und IRQ15 für Int0 verwendet. Ein Interrupt wird durch eine fallende Flanke an X3.L0 oder X3.L1 ausgelöst.

Quellcode:	SAMPLE2.PAS	
Units:	IK121_0.TPU	Grundfunktionen
	IK121_1.TPU	Funktionen für ein RAM-Speicher-Modell

SAMPLE3.EXE

Das Beispiel SAMPLE3.EXE zeigt die Anwendung der IK 121 zur Geschwindigkeitsmessung. Geschwindigkeit und Beschleunigung von Achse 1 werden angezeigt. Die Beschleunigung wird aus den Geschwindigkeiten ds(1) und ds(2) gebildet.

Quellcode:	SAMPLE3.PAS	
Units:	IK121_0.TPU	Grundfunktionen
	IK121_1.TPU	Funktionen für ein RAM-Speicher-Modell

SAMPLE4.EXE

Das Beispiel SAMPLE4.EXE zeigt, wie die elektronischen Potentiometer für Phasenlage und Amplitude sowie die Offset-Register gelesen werden können.

Quellcode:	SAMPLE4.PAS	
Units:	IK121_0.TPU	Grundfunktionen
	IK121_1.TPU	Funktionen für ein RAM-Speicher-Modell

SAMPLE5.EXE

Das Beispiel SAMPLE5.EXE zeigt für Achse 1 die Inhalte von Register 0 und Register 1. Außerdem wird vom Register 0 der Inhalt des Periodenzählers und der Interpolationswert separat angezeigt. Über die Tastatur können verschiedene Kommandos, die am Bildschirm erklärt sind, eingegeben werden.

Quellcode:	SAMPLE5.PAS	
Units:	IK121_0.TPU	Grundfunktionen
	IK121_1.TPU	Funktionen für ein RAM-Speicher-Modell

SAMPLE6.EXE

Das Beispiel SAMPLE6.EXE zeigt die Anwendung der IK121 als Periodenzähler für Achse 1 und Achse 2, d.h. der Interpolationswert wird nicht ausgewertet. Über die Tastatur können verschiedene Kommandos, die am Bildschirm erklärt sind, eingegeben werden.

Quellcode:	SAMPLE6.PAS	
Units:	IK121_0.TPU	Grundfunktionen
	IK121_1.TPU	Funktionen für ein RAM-Speicher-Modell

SCOPE.EXE

Das Beispiel SCOPE.EXE zeigt die sinusförmigen Signale der angeschlossenen Messgeräte entweder als Amplituden-Zeit-Diagramm oder in XY-Darstellung. Über Tastenbefehle, die am Bildschirm erklärt sind, können die Potentiometer eingestellt werden.

Quellcode:	SCOPE.PAS	
Units:	IK121_0.TPU	Grundfunktionen
	IK121_1.TPU	Funktionen für ein RAM-Speicher-Modell
	IK121_2.TPU	Funktionen für ADJUST.EXE, POTIS.EXE und SCOPE.EXE
	SCOPE_0.TPU	Funktionen für SCOPE.EXE
	CNT_0.TPU	Fenster-Funktionen
	LOGO.TPU	Einstiegsbild nach Programm-Start

POTIS.EXE

Das Beispiel POTIS.EXE zeigt die Stellung der elektronischen Potentiometer für Phasenlage und Amplitude sowie die Werte der Offset-Register an. Über Tastenbefehle, die am Bildschirm erklärt sind, können die Potentiometer eingestellt werden.

Quellcode:	POTIS.PAS	
Units:	IK121_0.TPU	Grundfunktionen
	IK121_1.TPU	Funktionen für ein RAM-Speicher-Modell
	IK121_2.TPU	Funktionen für ADJUST.EXE
	POTI_0.TPU	Funktionen für POTIS.EXE
	CNT_0.TPU	Fenster-Funktionen
	LOGO.TPU	Einstiegsbild nach Programm-Start

ADJUST.EXE

Das Beispiel ADJUST.EXE führt einen automatischen Abgleich der Achse 1 (Anwahl:1) oder Achse 2 (Anwahl:2) für Phasenlage (Anwahl:p), Amplitude (Anwahl:a) und Offset (Anwahl:o) der sinusförmigen Messgerät-Signale durch. Die Kompensationswerte werden durch langsames Bewegen des Messgerätes gebildet. Nach 30 Signalperioden wird durch einen Ton angezeigt, dass ein Kompensationswert gebildet wurde und durch Drücken der Taste S gespeichert werden kann.

Quellcode:	ADJUST.PAS	
Units:	IK121_0.TPU	Grundfunktionen
	IK121_1.TPU	Funktionen für ein RAM-Speicher-Modell
	IK121_2.TPU	Funktionen für ADJUST.EXE, POTIS.EXE und SCOPE.EXE
	ADJ_0.TPU	Funktionen für ADJUST.EXE
	CNT_0.TPU	Fenster-Funktionen
	LOGO.TPU	Einstiegsbild nach Programm-Start

IK121.EXE

Das Programm IK121.EXE ist ein Anwendungsbeispiel für ein Positionsanzeigen-Programm in „TURBO PASCAL“.

Quellcode:	IK121.PAS	
Units:	IK121_0.TPU	Grundfunktionen
	IK121_1.TPU	Funktionen für ein RAM-Speicher-Modell
	IK121_2.TPU	Funktionen für ADJUST.EXE, POTIS.EXE und SCOPE.EXE
	CNT_0.TPU	Fenster-Funktionen
	CNT_1.TPU	Positionsanzeige
	CNT_2.TPU	Zähler-Programm
	SCOPE_0.TPU	Funktionen für SCOPE.EXE
	ADJ_0.TPU	Funktionen für ADJUST.EXE
	LOGO.TPU	Einstiegsbild nach Programm-Start
Include-Dateien:	IK121.WIN	Fensterdefinitionen
	IK121.CNT	Initialisierungs-Datei
Hilfetexte:	IK121.HLP	Datei mit Texten zur Hilfestellung

Über Maus-Klick oder mit den horizontalen Pfeiltasten und Drücken der Eingabetaste kann in der Menüleiste ein Pull-Down-Menü geöffnet werden. Folgende Funktionen stehen zur Verfügung:

File	Einstellungen für die Karte, die Achsen und den Drucker ändern und speichern
Adjust	Sinusförmige Messgerät-Signale anzeigen und abgleichen
Counters	Achsen setzen, nullen oder Überfahren der Referenzmarken starten
Sorting	Betriebsart „Klassieren“: oberen und unteren Grenzwert eingeben
Datum	Bezugsebene wählen. Für beide Achsen stehen vier Bezugsebenen (L0 bis L3) zur Verfügung
MIN/MAX	Anzeige von minimalen oder maximalen Messwerten einer Messreihe aktivieren
Print	Messwert an den unter „File/setupIK121/printer“ eingestellten Drucker ausgeben. Wird als Drucker „File:on“ eingestellt, erfolgt die Ausgabe in die Datei IK121.DAT

Die Anzeigefelder neben den Positionsanzeigen haben die folgende Bedeutung:

signal	ein Messgerätfehler ist aufgetreten
Ref	das Anfahren der Referenzmarke ist angewählt
<	Messwert ist kleiner als der unter „Sorting“ eingegebene untere Grenzwert
>	Messwert ist größer als der unter „Sorting“ eingegebene obere Grenzwert

Wenn Sie bei der Bedienung des Programms IK 121 nicht mehr weiter wissen, dann drücken Sie einfach die Taste F1: am Bildschirm erscheinen Erklärungen zur angewählten Funktion.

Frei beschreibbares EEPROM

Auf der Platine der IK 121 steht ein EEPROM mit 512 Byte Speicher zur Verfügung, das über den I²C-Bus adressiert wird. Zum Schreiben und Lesen des EEPROMs sind die folgenden Programme auf der mitgelieferten Diskette 1:

RDROM.EXE

Das Programm RDROM.EXE liest den Inhalt des EEPROM-Speichers.

Quellcode:	RDROM.PAS	
Units:	IK 121_1.TPU	Funktionen für ein RAM-Speicher-Modell

WRROM.EXE

Das Programm WRROM.EXE schreibt die Datei IK121.TXT in den EEPROM-Speicher.

Quellcode:	RDROM.PAS	
Units:	IK 121_1.TPU	Funktionen für ein RAM-Speicher-Modell

Anwendungsbeispiele mit dem RAM-Speicher-Modell in „BORLAND C++“

Beispiele mit dem RAM-Speicher-Modell in „BORLAND C++“ befinden sich auf der mitgelieferten Diskette 1 im Verzeichnis **\BCPP.** Die verwendeten Datenstrukturen und Funktionen sind in folgenden Dateien deklariert und definiert:

IK121.H:	In dieser Header-Datei können Sie die Adresse von bis zu 16 IK 121 festlegen.
IK121_1.H:	Datenstrukturen für ein RAM-Speicher-Modell der Register der IK 121 und Funktions-deklarationen für die Funktionen in den Dateien IK121_1.CPP, IIC.CPP und POTI_1.CPP
IK121_1.CPP:	Basis-Funktionen für die IK 121
IIC.CPP:	Funktionen zum Datentransfer über den I²C-Bus.
POTI_1.CPP:	Funktionen zum Einstellen der elektronischen Potentiometer.

In der Datei IK121_1.H wird mit Hilfe von Datenstrukturen ein RAM-Speicher-Modell der Register der IK 121 aufgebaut. Die Daten des RAM-Speicher-Modells werden mit Hilfe der Prozeduren **InitHandler** und **CommHandler** in die Register der IK 121 geschrieben.

POTIS.EXE
Das Programm POTIS.EXE zeigt, wie Sie per Software die elektronischen Potentiometer der IK 121 über den I²C-Bus einstellen können.
Quellcode: POTIS.CPP, IK121_1.CPP
IIC.CPP, POTI_1.CPP
Header-Dateien: IK 121.H, IK121_1.H

RDROM.EXE
Das Programm RDROM.EXE liest den Inhalt des EEPROM-Speichers.
Quellcode: RDROM.CPP, IK121_1.CPP, IIC.CPP
Header-Dateien: IK121.H, IK 121_1.H

WRROM.EXE
Das Programm WRROM.EXE schreibt die Datei IK121.TXT in den EEPROM-Speicher.
Quellcode: WRROM.CPP, IK121_1.CPP, IIC.CPP
Header-Dateien: IK121.H, IK 121_1.H

DISPLAY.EXE
Das Programm DISPLAY.EXE zeigt die Inhalte aller Register der IK 121.
Quellcode: DISPLAY.CPP, IK121_1.CPP, IIC.CPP
Header-Dateien: IK121.H, IK 121_1.H

Die IK 121 in WINDOWS-Anwendungen

Auf den mitgelieferten Disketten 2 und 3 finden Sie die Treibersoftware, Dynamic Link Libraries (DLL) und Anwendungsbeispiele in VISUAL C++, VISUAL BASIC und BORLAND DELPHI für WINDOWS NT und WINDOWS 95.

Diskette 2
Auf der Diskette 2 finden Sie folgende Verzeichnisstruktur:

Install	**Installations-Dateien**
IK121Drv	**Device-Treiber-Source für WINDOWS NT**
Release	Release-Version
IK121Dll	**Windows-DLL-Source**
Release	Release-Version für Windows NT
Release95	Release-Version für Windows 95
IK121Con	**Beispiel-Source für Konsolen-Anwendung**
Release	Release-Version für Konsolen-Beispiel
IK121App	**Beispiel-Source für VISUAL C++**
Res	Resourcen für Windows-Beispiel
Release	Release-Version Windows-Beispiel
IK121VB5	**Beispiel-Source für VISUAL BASIC**
Include	**Include Dateien**
Install	**Installations Dateien**

Diskette 3
Auf der Diskette 3 finden Sie Beispiele für BORLAND DELPHI.

Device-Treiber für Windows NT (IK121DRV.SYS)

Auf Diskette 2 im Verzeichnis **\IK121Drv** finden Sie den Kernel-Mode-Device-Treiber für WINDOWS NT (Version 3.51 und 4.0). Er ermöglicht den Zugriff auf die IK 121.

Damit Windows NT den Treiber laden kann, muss er ins Windows-NT-Systemverzeichnis unter **\System32\Drivers** kopiert werden (z.B.: C:\WINNT\SYSTEM32\DRIVERS). Dies erledigt die Batch-Datei **Install.Bat.**

Registry-Eintrag

Die Information, auf welcher Portadresse die IK 121 installiert ist, erhält der Treiber aus der Registry. Folgender Registry Eintrag ist notwendig:

```
HKEY_LOCAL_MACHINE
  System
    CurrentControlSet
      Services
        IK121DRV
            ErrorControl      0x00000001
            Start             0x00000003
            Type              0x00000001
          Parameters
            IK_Base_10x00000330
            IK_Base_2   0x00000000
            IK_Base_3   0x00000000
            IK_Base_4   0x00000000
            IK_Base_5   0x00000000
            IK_Base_6   0x00000000
            IK_Base_7   0x00000000
            IK_Base_8   0x00000000
```

Auf Diskette 2 im Verzeichnis **\Install** finden Sie die Dateien **IK121Drv.Reg** und **SetReg.Bat.** Die Datei **IK121Drv.Reg** enthält die oben spezifizierten Daten, die Sie mit **Install.Bat** in die Registry schreiben.

In der Datei **IK121Drv.Reg** können Sie die Grundadresse der IK 121 ändern und/oder weitere Grundadressen hinzufügen. Der Treiber unterstützt bis zu acht IK 121. Der Eintrag in der Registry muss mit **SetReg.Bat** aktualisiert werden.

Die Windows DLL (IK121Dll.Dll)

Diese DLL ermöglicht es Anwendungsprogrammen, die IK 121 anzusprechen. Es gibt jeweils eine DLL für Windows NT und für Windows 95. Unter Windows NT wird die IK 121 über den Device Treiber für Windows NT angesprochen. Die DLL für Windows 95 greift direkt auf die Register der IK 121 zu. Oft benötigte Funktionen stehen direkt zur Verfügung (Start, Stopp, Zählerwert auslesen, REF-Funktionen usw.). Eine weitergehende Programmierung der IK 121 wird durch den Zugriff auf die Register der IK 121 ermöglicht (IKInputW, IKInputL, IKOutput usw.). Damit Awendungsprogramme die DLL laden können, muss sich unter Windows NT die Datei auf Diskette 2 **\IK121Dll\Release\IK121Dll.Dll** im Windows NT Systemverzeichnis unter **System32** befinden (z.B.: **C:\Winnt\System32**). Bei Windows 95 muss die Datei auf Diskette 2 **\IK121Dll\Release95\IK121Dll.Dll** im Systemverzeichnis unter **System** gespeichert sein (z.B.: **C:\Windows\System**). Die Batch-Datei **Install.Bat** kopiert die Dateien in das jeweilige Verzeichnis.

Beispiel für Konsolen-Anwendung

Auf Diskette 2 im Verzeichnis **\IK121Con\Release** finden Sie eine einfache Konsolen-Anwendung.

Beispiel für VISUAL C++

Auf Diskette 2 im Verzeichnis **\IK121App\Release** finden Sie eine Anwendung in VISUAL C++.

Beispiel für VISUAL BASIC

Auf Diskette 2 im Verzeichnis **\IK121VB5\Release** finden Sie eine Anwendung in VISUAL BASIC.

Beispiel für BORLAND DELPHI

Auf Diskette 3 finden Sie eine Anwendung in BORLAND DELPHI.

Installation der Treiber und der DLL unter WINDOWS NT und WINDOWS 95

- ➤ Wählen Sie auf der mitgelieferten Diskette 2 das Verzeichnis **\Install.**
- ➤ Tragen Sie in die Datei **IK121Drv.Reg** die Portadressen der installierten IK 121 ein.
- ➤ Rufen Sie **Install.Bat** auf.

Install.bat erzeugt den Eintrag in die Registry, kopiert den Treiber für WINDOWS NT aus dem Verzeichnis **\IK121Drv\Release** in das Systemverzeichnis (z. B. **C:\Winnt\System32\Drivers**) und die DLL für WINDOWS NT (oder WINDOWS 95) aus dem Verzeichnis **\IK121Dll\Release** (oder \IK121Dll\Release95) in das Systemverzeichnis (z. B. **C:\Winnt\System32**).

Aufruf der DLL-Funktionen aus Ihren eigenen Anwenderprogrammen

Um die Funktionen der DLL nutzen zu können, müssen sie dem Anwenderprogramm bekannt gemacht werden.

MICROSOFT VISUAL C++

Wenn Sie das Anwenderprogramm mit VISUAL C++ erstellen, dann muss sich die Datei **\IK121Dll\Release\IK121Dll.Lib** von Diskette 2 in dem Library-Verzeichnis von VISUAL C++ befinden (z.B. **C:\Msdev\lib**) und im Projekt eingebunden sein. Dazu erstellen Sie einen Eintrag in VISUAL C++ unter **Build, Settings, Link, Object/Library modules.**

In einem Header-File müssen Sie außerdem folgende Prototypen definieren:

```
#ifdef __cplusplus
  extern "C"
  {
#endif

WINUSERAPI BOOL WINAPI IKFind (ULONG* pBuffer8);
WINUSERAPI BOOL WINAPI IKInit (USHORT Axis, USHORT Mode);
WINUSERAPI BOOL WINAPI IKVersion (USHORT Axis, char* pVersCard, char* pVersDrv,
                                  char* pVersDll);

WINUSERAPI BOOL WINAPI IKReset (USHORT Axis);
WINUSERAPI BOOL WINAPI IKStart (USHORT Axis);
WINUSERAPI BOOL WINAPI IKStop (USHORT Axis);
WINUSERAPI BOOL WINAPI IKClear (USHORT Axis);
WINUSERAPI BOOL WINAPI IKLatch (USHORT Axis, USHORT Latch);

WINUSERAPI BOOL WINAPI IKResetREF (USHORT Axis);
WINUSERAPI BOOL WINAPI IKStartREF (USHORT Axis);
WINUSERAPI BOOL WINAPI IKStopREF (USHORT Axis);
WINUSERAPI BOOL WINAPI IKLatchREF (USHORT Axis, USHORT Latch);

WINUSERAPI BOOL WINAPI IKLatched (USHORT Axis, USHORT Latch, BOOL* pStatus);
WINUSERAPI BOOL WINAPI IKWaitLatch (USHORT Axis, USHORT Latch);

WINUSERAPI BOOL WINAPI IKStrtCodRef (USHORT Axis, USHORT Latch, ULONG RefDist);
WINUSERAPI BOOL WINAPI IKCodRef (USHORT Axis, BOOL* pStatus, double* pData);
WINUSERAPI BOOL WINAPI IKWaitCodRef (USHORT Axis, double* pData);
WINUSERAPI BOOL WINAPI IKStopCodRef (USHORT Axis);
```

```
WINUSERAPI BOOL WINAPI IKStatus (USHORT Axis, ULONG* pStatus);

WINUSERAPI BOOL WINAPI IKRead32 (USHORT Axis, USHORT Latch, LONG* pData);
WINUSERAPI BOOL WINAPI IKRead48 (USHORT Axis, USHORT Latch, double* pData);

WINUSERAPI BOOL WINAPI IKReadPhase (USHORT Axis, BYTE* pData);
WINUSERAPI BOOL WINAPI IKWritePhase (USHORT Axis, BYTE   Data);
WINUSERAPI BOOL WINAPI IKLoadPhase (USHORT Axis, BYTE* pData);

WINUSERAPI BOOL WINAPI IKReadAmp (USHORT Axis, BYTE* pData);
WINUSERAPI BOOL WINAPI IKWriteAmp (USHORT Axis,  BYTE   Data);
WINUSERAPI BOOL WINAPI IKLoadAmp (USHORT Axis, BYTE* pData);

WINUSERAPI BOOL WINAPI IKReadOffset (USHORT Axis, SHORT* Ofs0, SHORT* Ofs90);
WINUSERAPI BOOL WINAPI IKWriteOffset (USHORT Axis, SHORT  Ofs0, SHORT  Ofs90);
WINUSERAPI BOOL WINAPI IKLoadOffset (USHORT Axis, SHORT* Ofs0, SHORT* Ofs90);

WINUSERAPI BOOL WINAPI IKStore (USHORT Axis);
WINUSERAPI BOOL WINAPI IKDefault (USHORT Axis);

WINUSERAPI BOOL WINAPI IKRomRead (USHORT Card, BYTE Adr, BYTE* Data);
WINUSERAPI BOOL WINAPI IKRomWrite (USHORT Card, BYTE Adr, BYTE  Data);

WINUSERAPI BOOL WINAPI IKInputW (USHORT Axis, USHORT Adr, USHORT* pData);
WINUSERAPI BOOL WINAPI IKInputL (USHORT Axis, USHORT Adr, ULONG*   pData);
WINUSERAPI BOOL WINAPI IKOutput (USHORT Axis, USHORT Adr, USHORT    Data);

WINUSERAPI BOOL WINAPI IKSetI2C (USHORT Card, BOOL SCL, BOOL SDA)

WINUSERAPI BOOL WINAPI IKDefine (ULONG* pBuffer8);

WINUSERAPI BOOL WINAPI IKSetTimer (USHORT Axis, USHORT SetVal);
WINUSERAPI BOOL WINAPI IKSetEnableLatch (USHORT Axis, USHORT Latch, USHORT Source);
WINUSERAPI BOOL WINAPI IKSetEnableSync (USHORT Card, USHORT Source);
WINUSERAPI BOOL WINAPI IKLatchAll (USHORT Axis);
#ifdef __cplusplus
 }
endif
```

Anschließend können Sie die Funktionen wie „normale“ C-Funktionen nutzen.

MICROSOFT VISUAL BASIC

In VISUAL BASIC können Sie die Funktionen in einem Modul auf folgende Weise definieren:

```
Public Declare Function IKFind     Lib "IK121DLL.DLL"
                                   (ByRef pBuffer8 As Long) As Boolean
Public Declare Function IKInit     Lib "IK121DLL.DLL"
                                   (ByVal Axis As Integer, ByVal Mode As Integer) As Boolean
Public Declare Function IKVersion  Lib "IK121DLL.DLL"
                                   (ByVal Axis As Integer, ByRef pVersCard As Byte, ByRef
                                   pVersDrv As Byte, ByRef pVersDll As Byte) As Boolean
```

```
Public Declare Function IKReset Lib "IK121DLL.DLL"
                                (ByVal Axis As Integer) As Boolean
Public Declare Function IKStart Lib "IK121DLL.DLL"
                                (ByVal Axis As Integer) As Boolean
Public Declare Function IKStop  Lib "IK121DLL.DLL"
                                (ByVal Axis As Integer) As Boolean
Public Declare Function IKClear Lib "IK121DLL.DLL"
                                (ByVal Axis As Integer) As Boolean
Public Declare Function IKLatch Lib "IK121DLL.DLL"
                                (ByVal Axis As Integer, ByVal Latch As Integer)
                                As Boolean

Public Declare Function IKResetREF Lib "IK121DLL.DLL"
                                   (ByVal Axis As Integer) As Boolean
Public Declare Function IKStartREF Lib "IK121DLL.DLL"
                                   (ByVal Axis As Integer) As Boolean
Public Declare Function IKStopREF  Lib "IK121DLL.DLL"
                                   (ByVal Axis As Integer) As Boolean
Public Declare Function IKLatchREF Lib "IK121DLL.DLL"
                                   (ByVal Axis As Integer, ByVal Latch As Integer)
                                   As Boolean

Public Declare Function IKLatched Lib "IK121DLL.DLL"
                                  (ByVal Axis As Integer, ByVal Latch As Integer, ByRef pS
                                  tatus As Boolean) As Boolean
Public Declare Function IKWaitLatch Lib "IK121DLL.DLL"
                                    (ByVal Axis As Integer, ByVal Latch As Integer)
                                    As Boolean

Public Declare Function IKStrtCodRef Lib "IK121DLL.DLL"
                                     (ByVal Axis As Integer, ByVal Latch As Integer, ByVal
                                     RefDist As Long) As Boolean
Public Declare Function IKCodRef Lib "IK121DLL.DLL"
                                 (ByVal Axis As Integer, ByRef pStatus As Boolean,
                                 ByRef pData As Double) As Boolean
Public Declare Function IKWaitCodRef Lib "IK121DLL.DLL"
                                     (ByVal Axis As Integer, ByRef pData As Double)
                                     As Boolean
Public Declare Function IKStopCodRef Lib "IK121DLL.DLL"
                                     (ByVal Axis As Integer) As Boolean

Public Declare Function IKStatus Lib "IK121DLL.DLL"
                                 (ByVal Axis As Integer, ByRef pStatus As Long) As
                                 Boolean

Public Declare Function IKRead32 Lib "IK121DLL.DLL"
                                 (ByVal Axis As Integer, ByVal Latch As Integer, ByRef
                                 pData As Long) As Boolean
Public Declare Function IKRead48 Lib "IK121DLL.DLL"
                                 (ByVal Axis As Integer, ByVal Latch As Integer, ByRef
                                 pData As Double) As Boolean
```

```
Public Declare Function IKReadPhase   Lib "IK121DLL.DLL"
                                      (ByVal Axis As Integer, ByRef pData As Byte) As Boolean
Public Declare Function IKWritePhase  Lib "IK121DLL.DLL"
                                      (ByVal Axis As Integer, ByVal Data As Byte) As Boolean
Public Declare Function IKLoadPhase   Lib "IK121DLL.DLL"
                                      (ByVal Axis As Integer, ByRef pData As Byte) As Boolean

Public Declare Function IKReadAmp     Lib "IK121DLL.DLL"
                                      (ByVal Axis As Integer, ByRef pData As Byte) As Boolean
Public Declare Function IKWriteAmp    Lib "IK121DLL.DLL"
                                      (ByVal Axis As Integer, ByVal Data As Byte) As Boolean
Public Declare Function IKLoadAmp     Lib "IK121DLL.DLL"
                                      (ByVal Axis As Integer, ByRef pData As Byte) As Boolean

Public Declare Function IKReadOffset  Lib "IK121DLL.DLL"
                                      (ByVal Axis As Integer, ByRef Ofs0 As Integer, ByRef
                                      Ofs90 As Integer) As Boolean
Public Declare Function IKWriteOffset Lib "IK121DLL.DLL"
                                      (ByVal Axis As Integer, ByVal Ofs0 As Integer, ByVal Ofs90
                                      As Integer) As Boolean
Public Declare Function IKLoadOffset  Lib "IK121DLL.DLL"
                                      (ByVal Axis As Integer, ByRef Ofs0 As Integer, ByRef
                                      Ofs90 As Integer) As Boolean

Public Declare Function IKStore       Lib "IK121DLL.DLL" (ByVal Axis) As Boolean
Public Declare Function IKDefault     Lib "IK121DLL.DLL" (ByVal Axis) As Boolean

Public Declare Function IKRomRead     Lib "IK121DLL.DLL"
                                      (ByVal Card As Integer, ByVal Adr As Byte, ByRef Data
                                      As Byte) As Boolean
Public Declare Function IKRomWrite    Lib "IK121DLL.DLL"
                                      (ByVal Card As Integer, ByVal Adr As Byte, ByVal Data
                                      As Byte) As Boolean

Public Declare Function IKInputW      Lib "IK121DLL.DLL"
                                      (ByVal Axis As Integer, ByVal Adr As Integer, ByRef pData
                                      As Long) As Boolean
Public Declare Function IKInputL      Lib "IK121DLL.DLL"
                                      (ByVal Axis As Integer, ByVal Adr As Integer, ByRef pData
                                      As Long) As Boolean
Public Declare Function IKOutput      Lib "IK121DLL.DLL"
                                      (ByVal Axis As Integer, ByVal Adr As Integer, ByVal Data
                                      As Long) As Boolean

Public Declare Function IKSetI2C      Lib "IK121DLL.DLL"
                                      (ByVal Card As Integer, ByVal SCL As Boolean, ByVal SDA
                                      As Boolean) As Boolean

Public Declare Function IKDefine      Lib "IK121DLL.DLL"
                                      (ByRef pBuffer8 As Long) As Boolean
```

```
Public Declare Function IKSetTimer      Lib "IK121DLL.DLL"
                                        (ByVal Axis As Integer, ByVal SetVal As Integer)
                                        As Boolean
Public Declare Function IKEnableLatch   Lib "IK121DLL.DLL"
                                        (ByVal Axis As Integer, ByVal Latch As Integer, ByVal
                                        Source As Integer) As Boolean
Public Declare Function IKEnableSync    Lib "IK121DLL.DLL" (
                                        ByVal Card As Integer, ByVal Source As Integer)
                                        As Boolean
Public Declare Function IKLatchAll      Lib "IK121DLL.DLL"
                                        (ByVal Card As Integer) As Boolean
```

BORLAND DELPHI

In BORLAND DELPHI binden Sie die Funktionen auf folgende Weise in Ihr Programm ein:

```
Function IKFind          (pBuffer8: Long8Ptr) : Boolean; StdCall; External 'IK121DLL.DLL';
Function IKInit          (Axis: Word; Mode: Word) : Boolean; StdCall; External
                         'IK121DLL.DLL';
Function IKVersion       (Axis: Word; pVersCard: CharPtr; pVersDrv: CharPtr;
                         pVersDll: CharPtr) : Boolean; StdCall; External 'IK121DLL.DLL';

Function IKReset         (Axis: Word) : Boolean; StdCall; External 'IK121DLL.DLL';
Function IKStart         (Axis: Word) : Boolean; StdCall; External 'IK121DLL.DLL';
Function IKStop          (Axis: Word) : Boolean; StdCall; External 'IK121DLL.DLL';
Function IKClear         (Axis: Word) : Boolean; StdCall; External 'IK121DLL.DLL';
Function IKLatch         (Axis: Word; Latch: Word) : Boolean; StdCall; External
                         'IK121DLL.DLL';

Function IKResetREF      (Axis: Word) : Boolean; StdCall; External 'IK121DLL.DLL';
Function IKStartREF      (Axis: Word) : Boolean; StdCall; External 'IK121DLL.DLL';
Function IKStopREF       (Axis: Word) : Boolean; StdCall; External 'IK121DLL.DLL';
Function IKLatchREF      (Axis: Word; Latch: Word) : Boolean; StdCall; External
                         'IK121DLL.DLL';

Function IKLatched       (Axis: Word; Latch: Word; pStatus: BoolPtr) : Boolean; StdCall;
                         External 'IK121DLL.DLL';
Function IKWaitLatch     (Axis: Word; Latch: Word) : Boolean; StdCall; External
                         'IK121DLL.DLL';

Function IKStrtCodRef    (Axis: Word; Latch: Word; RefDist: LongInt) : Boolean; StdCall;
                         External 'IK121DLL.DLL';
Function IKCodRef        (Axis: Word; pStatus: BoolPtr; pData: DoublePtr) : Boolean;
                         StdCall; External 'IK121DLL.DLL';
Function IKWaitCodRef    (Axis: Word; pData: DoublePtr) : Boolean; StdCall; External
                         'IK121DLL.DLL';
Function IKStopCodRef    (Axis: Word) : Boolean; StdCall; External 'IK121DLL.DLL';

Function IKStatus        (Axis: Word; pStatus: LongPtr) : Boolean; StdCall; External
                         'IK121DLL.DLL';
```

```
Function IKRead32       (Axis: Word; Latch: Word; pData: LongPtr) : Boolean; StdCall;
                        External 'IK121DLL.DLL';
Function IKRead48       (Axis: Word; Latch: Word; pData: DoublePtr) : Boolean; StdCall;
                        External 'IK121DLL.DLL';

Function IKReadPhase    (Axis: Word; pData: BytePtr) : Boolean; StdCall; External
                        'IK121DLL.DLL';
Function IKWritePhase   (Axis: Word;   Data: Byte) : Boolean; StdCall; External
                        'IK121DLL.DLL';
Function IKLoadPhase    (Axis: Word; pData: BytePtr) : Boolean; StdCall; External
                        'IK121DLL.DLL';

Function IKReadAmp      (Axis: Word; pData: BytePtr) : Boolean; StdCall; External
                        'IK121DLL.DLL';
Function IKWriteAmp     (Axis: Word;   Data: Byte) : Boolean; StdCall; External
                        'IK121DLL.DLL';
Function IKLoadAmp      (Axis: Word; pData: BytePtr) : Boolean; StdCall; External
                        'IK121DLL.DLL';

Function IKReadOffset   (Axis: Word; Ofs0: IntPtr;   Ofs90: IntPtr) : Boolean; StdCall;
                        External 'IK121DLL.DLL';
Function IKWriteOffset  (Axis: Word; Ofs0: Integer; Ofs90: Integer) : Boolean; StdCall;
                        External 'IK121DLL.DLL';
Function IKLoadOffset   (Axis: Word; Ofs0: IntPtr;   Ofs90: IntPtr) : Boolean; StdCall;
                        External 'IK121DLL.DLL';

Function IKStore        (Axis: Word) : Boolean; StdCall; External 'IK121DLL.DLL';
Function IKDefault      (Axis: Word) : Boolean; StdCall; External 'IK121DLL.DLL';

Function IKRomRead      (Card: Word; Adr: Byte; Data: BytePtr) : Boolean; StdCall; External
                        'IK121DLL.DLL';
Function IKRomWrite     (Card: Word; Adr: Byte; Data: Byte) : Boolean; StdCall; External
                        'IK121DLL.DLL';

Function IKInputW       (Axis: Word; Adr: Word; pData: WordPtr) : Boolean; StdCall;
                        External 'IK121DLL.DLL';
Function IKInputL       (Axis: Word; Adr: Word; pData: LongPtr) : Boolean; StdCall;
                        External 'IK121DLL.DLL';
Function IKOutput       (Axis: Word; Adr: Word;   Data: LongInt) : Boolean; StdCall;
                        External 'IK121DLL.DLL';

Function IKSetI2C       (Card: Word; SCL: Boolean; SDA: Boolean) : Boolean; StdCall;
                        External 'IK121DLL.DLL';

Function IKDefine       (pBuffer8: Long8Ptr) : Boolean; StdCall; External 'IK121DLL.DLL';

Function IKSetTimer     (Axis: Word; SetVal: Word) :
                        Boolean; StdCall; External 'IK121DLL.DLL';
Function IKEnableLatch  (Axis: Word; Latch: Word; Source: Word) :
                        Boolean; StdCall; External 'IK121DLL.DLL';
Function IKEnableSync   (Card: Word; Source: Word) : Boolean; StdCall;
                        External 'IK121DLL.DLL';
Function IKLatchAll     (Axis: Word) : Boolean; StdCall; External 'IK121DLL.DLL';
```

Übersicht der DLL-Funktionen

Funktion	Kurzreferenz
Installierte IK 121 feststellen	BOOL IKFind (ULONG* pBuffer8);
IK 121 initialisieren	BOOL IKInit (USHORT Axis, USHORT Mode);
Programmversionen lesen	BOOL IKVersion (USHORT Axis, char* pVersCard, char* pVersDrv, char* pVersDll);
Zähler löschen	BOOL IKReset (USHORT Axis);
Zähler starten	BOOL IKStart (USHORT Axis);
Zähler stoppen	BOOL IKStop (USHORT Axis);
Frequenz- und Amplituden-Fehler löschen	BOOL IKClear (USHORT Axis);
Zählerwert speichern	BOOL IKLatch (USHORT Axis, USHORT Latch);
Zähler löschen mit nächster Referenzmarke	BOOL IKResetREF (USHORT Axis);
Zähler starten mit nächster Referenzmarke	BOOL IKStartREF (USHORT Axis);
Zähler stoppen mit nächster Referenzmarke	BOOL IKStopREF (USHORT Axis);
Zählerwert speichern mit nächster Referenzmarke	BOOL IKLatchREF (USHORT Axis, USHORT Latch);
Abfrage, ob Zählerwert gespeichert	BOOL IKLatched (USHORT Axis, USHORT Latch, BOOL* pStatus);
Warten, bis Zählerwert gespeichert	BOOL IKWaitLatch (USHORT Axis, USHORT Latch);
Startet Referenzpunkt-Fahren mit abstandscodierten Referenzmarken	BOOL IKStrtCodRef (USHORT Axis, USHORT Latch, ULONG RefDist);
Abfrage, ob Referenzpunkt-Fahren mit abstandscodierten Referenzmarken beendet	BOOL IKCodRef (USHORT Axis, BOOL* pStatus, double* pData);
Wartet, bis Referenzpunkt-Fahren mit abstandscodierten Referenzmarken beendet	BOOL IKWaitCodRef (USHORT Axis, double* pData);
Stoppt Referenzpunkt-Fahren mit abstandscodierten Referenzmarken	BOOL IKStopCodRef (USHORT Axis);
Statusabfrage	BOOL IKStatus (USHORT Axis, ULONG* pStatus);
Gespeicherten Zählerwert lesen (32 Bit)	BOOL IKRead32 (USHORT Axis, USHORT Latch, LONG* pData);

Funktion	Kurzreferenz
Gespeicherten Zählerwert lesen (48 Bit)	BOOL IKRead48 (USHORT Axis, USHORT Latch, double* pData);
Phasenkorrekturwert lesen	BOOL IKReadPhase (USHORT Axis, BYTE* pData);
Phasenkorrekturwert ändern	BOOL IKWritePhase (USHORT Axis, BYTE Data);
Gespeicherten Phasenkorrekturwert lesen	BOOL IKLoadPhase (USHORT Axis, BYTE* pData);
Amplitudenkorrekturwert lesen	BOOL IKReadAmp (USHORT Axis, BYTE* pData);
Amplitudenkorrekturwert ändern	BOOL IKWriteAmp (USHORT Axis, BYTE Data);
Gespeicherten Amplitudenkorrekturwert lesen	BOOL IKLoadAmp (USHORT Axis, BYTE* pData);
Offsetkorrekturwert lesen	BOOL IKReadOffset (USHORT Axis, SHORT* Ofs0, SHORT* Ofs90);
Offsetkorrekturwert ändern	BOOL IKWriteOffset (USHORT Axis, SHORT Ofs0, SHORT Ofs90);
Gespeicherten Offsetkorrekturwert lesen	BOOL IKLoadOffset (USHORT Axis, SHORT* Ofs0, SHORT* Ofs90);
Korrekturwerte abspeichern	BOOL IKStore (USHORT Axis);
Neutrale Werte laden und abspeichern	BOOL IKDefault (USHORT Axis);
Wert aus EEPROM lesen	BOOL IKRomRead (USHORT Card, BYTE Adr, BYTE* Data);
Wert in EEPROM schreiben	BOOL IKRomWrite (USHORT Card, BYTE Adr, BYTE Data);
IK121-Register lesen (16 Bit)	BOOL IKInputW (USHORT Axis, USHORT Adr, USHORT* pData);
IK121-Register lesen (32 Bit)	BOOL IKInputL (USHORT Axis, USHORT Adr, ULONG* pData);
IK121-Register schreiben (16 Bit)	BOOL IKOutput (USHORT Axis, USHORT Adr, USHORT Data);
I^2C-Leitungen setzen	BOOL IKSetI2C (USHORT Card, BOOL SCL, BOOL SDA);
Portadressen der IK 121 festlegen	BOOL IKDefine (ULONG* pBuffer8);
Wert für Timer festlegen	BOOL IKSetTimer (USHORT Axis, USHORT SetVal);
Einspeicher-Signal freigeben	BOOL IKEnableLatch (USHORT Axis, USHORT Latch, USHORT Source);
Kaskadierung festlegen	BOOL IKEnableSync (USHORT Card, USHORT Source);
Positionswert speichern alle Achsen	BOOL IKLatchAll (USHORT Axis);

Referenz der DLL-Funktionen

Alle DLL-Funktionen liefern eine Boolsche Variable zurück:
true (<> 0), falls die Funktion erfolgreich ausgeführt wurde, und
false (= 0), falls ein Fehler auftrat.

IKFind

Diese Funktion liefert die Portadressen der installierten IK 121. Unbenutzte Einträge werden auf 0 gesetzt.

Prototyp: **BOOL IKFind (ULONG* pBuffer[8]);**
pBuffer: Zeiger auf 8 Langworte (4 Byte)

IKInit

Diese Funktion initialisiert die IK 121.

Prototyp: **BOOL IKInit (USHORT Axis, USHORT Mode);**
Axis: Nummer der Achse (0 bis 15)
Mode: 0=32-Bit-Zählerwert
1=48-Bit-Zählerwert

IKVersion

Diese Funktion liest die Programmversionen des NT Device Treibers und der DLL. Die Programmversionen werden als ASCII-Zeichen abgelegt. Es muss jeweils Platz für mindestens 20 Zeichen reserviert werden. Die Zeichenketten werden mit jeweils einem Null-Byte abgeschlossen.

Prototyp: **BOOL IKVersion (USHORT Axis, char* pVersCard, char* pVersDrv, char* pVersDll)**
Axis: Nummer der Achse (0 bis 15)
pVersCard: Zeiger auf die Version der IK 121
pVersDrv: Zeiger auf die Programmversion des Windows NT Device Treibers (nur unter Windows NT)
pVersDll: Zeiger auf die Programmversion der DLL

IKReset

Diese Funktion setzt den Zähler auf Null.

Prototyp: **BOOL IKReset (USHORT Axis);**
Axis: Nummer der Achse (0 bis 15)

IKStart

Diese Funktion startet den Zähler.

Prototyp: **BOOL IKStart (USHORT Axis);**
Axis: Nummer der Achse (0 bis 15)

IKStop
Diese Funktion stoppt den Zähler.
Prototyp: **BOOL IKStop (USHORT Axis);**
Axis: Nummer der Achse (0 bis 15)

IKClear
Diese Funktion löscht den Fehlerstatus.
Prototyp: **BOOL IKClear (USHORT Axis);**
Axis: Nummer der Achse (0 bis 15)

IKLatch
Diese Funktion speichert den Zählerwert.
Prototyp: **BOOL IKLatch (USHORT Axis, USHORT Latch);**
Axis: Nummer der Achse (0 bis 15)
Latch: 0=Zählerwert wird in Register 0 gespeichert
1=Zählerwert wird in Register 1 gespeichert

IKResetREF
Diese Funktion setzt den Zähler mit der nächsten Referenzmarke auf Null.
Prototyp: **BOOL IKResetREF (USHORT Axis);**
Axis: Nummer der Achse (0 bis 15)

IKStartREF
Diese Funktion startet den Zähler mit der nächsten Referenzmarke.
Prototyp: **BOOL IKStartREF (USHORT Axis);**
Axis: Nummer der Achse (0 bis 15)

IKStopREF
Diese Funktion stoppt den Zähler mit der nächsten Referenzmarke.
Prototyp: **BOOL IKStopREF (USHORT Axis);**
Axis: Nummer der Achse (0 bis 15)

IKLatchREF
Diese Funktion speichert beim Überfahren der Referenzmarke den Zählerwert.
Prototyp: **BOOL IKLatchREF (USHORT Axis, USHORT Latch);**
Axis: Nummer der Achse (0 bis 15)

IKLatched
Diese Funktion stellt fest, ob der Zählerwert gespeichert wurde. Anwendung: Bevor ein Zählerwert ausgelesen wird, muss abgefragt werden, ob der Zählerwert gespeichert ist.

Prototyp: **BOOL IKLatched (USHORT Axis, USHORT Latch, BOOL* pStatus);**

Axis: Nummer der Achse (0 bis 15)
Latch: 0 = Abfrage für Register 0
1 = Abfrage für Register 1
pStatus: Zeiger auf eine Boolsche Variable (16 Bit)
false (= 0) = Wert nicht gespeichert
true (<> 0) = Wert gespeichert

IKWaitLatch
Diese Funktion wartet, bis der Zählerwert gespeichert wurde. Anwendung: Bevor ein Zählerwert ausgelesen wird, muss abgefragt werden, ob der Zählerwert gespeichert ist.

Prototyp: **BOOL IKWaitLatch (USHORT Axis, USHORT Latch);**

Axis: Nummer der Achse (0 bis 15)
Latch: 0 = Abfrage für Register 0
1 = Abfrage für Register 1

IKStrtCodRef
Diese Funktion initialisiert das Referenzpunkt-Fahren mit abstandscodierten Referenzmarken. Danach muss man zyklisch abfragen (Funktion: IKCodRef) **oder** warten (Funktion: IKWaitCodRef)., bis das Überfahren der abstandscodierten Referenzmarken beendet ist

Prototyp: **BOOL IKStrtCodRef (USHORT Axis, USHORT Latch, ULONG RefDist);**

Axis: Nummer der Achse (0 bis 15)
Latch: 0 = mit Register 0
1 = mit Register 1
RefDist: fester Abstand der Referenzmarken
(z.B. 500, 1 000, 2 000, 5000)

IKCodRef

Diese Funktion stellt fest, ob beim Referenzpunkt-Fahren mit abstandscodierten Referenzmarken die zweite Referenzmarke überfahren wurde und liefert den Offsetwert zurück. Der Offsetwert muss zum Zählerwert addiert werden, um die absolute Position zu erhalten. Diese Funktion muss man nach dem Starten des Referenzpunkt-Fahrens zyklisch aufrufen. (Man kann aber auch auf das Ende warten – Funktion: IKWaitCodRef).

Prototyp: **BOOL IKCodRef (USHORT Axis, BOOL* pStatus, double* pData);**

Axis: Nummer der Achse (0 bis 15)

pStatus: Zeiger auf eine Boolsche Variable (16 Bit).
False (= 0) = Referenzpunkt-Fahren nicht beendet.
True (<> 0) = Referenzpunkt-Fahren beendet.

pData: Zeiger auf eine „double-Variable“ (64 Bit), in welcher der Offsetwert abgelegt wird (nur wenn pStatus=TRUE ist).

IKWaitCodRef

Diese Funktion wartet, bis das Referenzpunkt-Fahren mit abstandscodierten Referenzmarken beendet ist. Nachdem die zweite Referenzmarke überfahren wurde, wird der Offsetwert zurückgegeben.

Prototyp: **BOOL IKWaitCodRef (USHORT Axis, double* pData);**

Axis: Nummer der Achse (0 bis 15)

pData: Zeiger auf eine „double-Variable“ (64 Bit), in welcher der Offsetwert abgelegt wird.

IKStopCodRef

Diese Funktion bricht das Überfahren von abstandscodierten Referenzmarken ab.

Prototyp: **BOOL IKStopCodRef (USHORT Axis);**

Axis: Nummer der Achse (0 bis 15)

IKStatus

Diese Funktion liefert den Status der IK 121 zurück.

Prototyp: **BOOL IKStatus (USHORT Axis, ULONG* pStatus);**

Axis: Nummer der Achse (0 bis 15)

pStatus: Zeiger auf ein Langwort (32 Bit)

Bit	Funktion
D0	1 = Zählerwert in Register 0 gespeichert
D1	1 = Zählerwert in Register 1 gespeichert
D2 D3	ohne Funktion
D4	1 = Zähler gestoppt
D5	ohne Funktion
D6	1 = Frequenzüberschreitung
D7	ohne Funktion
D8	1 = REF-Funktion aktiv
D9	1 = Zähler wird mit nächster Referenzmarke gestartet[1)]
D10	1 = Zähler wird mit nächster Referenzmarke gestoppt[1)]
D11	1 = Zähler wird mit nächster Referenzmarke auf Nullgesetzt[1)]
D12	1 = Zähler wird mit nächster Referenzmarke gespeichert[1)]
D13	1 = Zähler wird mit übernächster Referenzmarke gespeichert[1)]
D14	1 = Zähler wird mit jeder Referenzmarke auf Null gesetzt[1)]
D15 D16	ohne Funktion
D17 D18	Aktuelle Amplitude 00 = normale Amplitude (4,5µA < U_e < 15µA) 01 = kleine Amplitude (U_e < 4,5µA) 10 = hohe Amplitude (U_e > 15µA) 11 = fehlerhaft kleine Amplitude (U_e < 2,5µA)
D19 D20	Minimale Amplitude 00 = normale Amplitude (4,5µA < U_e < 15µA) 01 = kleine Amplitude (U_e < 4,5µA) 10 = hohe Amplitude (U_e > 15µA) 11 = fehlerhaft kleine Amplitude (U_e < 2,5µA)
D21 D22	Maximale Amplitude [1)] 00 = normale Amplitude (4,5µA < U_e < 15µA) 01 = kleine Amplitude (U_e < 4,5µA) 10 = hohe Amplitude (U_e > 15µA) 11 = fehlerhaft kleine Amplitude (U_e < 2,5µA
D23	ohne Funktion
D24 bis D31	IC-Code des Zählerbausteins (08 oder 09)

1) nur bei IC-Code = 09

IKRead32

Diese Funktion liefert den 32-Bit-Zählerwert. Bevor der Zählerwert ausgelesen werden kann, muss er in Register 0 oder Register 1 gespeichert (IKLatch, IKLatchREF) und abgefragt werden, ob die Speicherung erfolgte (IKLatched, IKWaitLatch).

Prototyp: **BOOL IKRead32 (USHORT Axis, USHORT Latch, LONG* pData);**

Axis: Nummer der Achse (0 bis 15)

Latch: 0 = Auslesen aus Register 0
1 = Auslesen aus Register 1

pData: Zeiger auf ein Langwort (32 Bit), in dem der Zählerwert abgelegt wird.

IKRead48

Diese Funktion liefert den 48-Bit-Zählerwert. Bevor der Zählerwert ausgelesen werden kann, muss er in Register 0 oder Register 1 gespeichert (IKLatch, IKLatchREF) und abgefragt werden, ob die Speicherung erfolgte (IKLatched, IKWaitLatch).

Prototyp: **BOOL IKRead48 (USHORT Axis, USHORT Latch, double* pData);**

Axis: Nummer der Achse (0 bis 15)

Latch: 0 = Auslesen aus Register 0
1 = Auslesen aus Register 1

pData: Zeiger auf ein „double-Variable“ (64 Bit), in welcher der Zählerwert abgelegt wird.

IKReadPhase

Diese Funktion liest die aktuelle Einstellung des Phasenkorrektur-Potentiometers.

Prototyp: **BOOL IKReadPhase (USHORT Axis, BYTE* pData);**

Axis: Nummer der Achse (0 bis 15)

pData: Zeiger auf eine „Byte-Variable“ (8 Bit), in der die Phasenkorrektur abgelegt wird.

IKWritePhase
Diese Funktion ändert die momentane Einstellung der Phasenkorrektur.
Prototyp: BOOL IKWritePhase (USHORT Axis, BYTE Data);
Axis: Nummer der Achse (0 bis 15)
Data: Neuer Wert der Phasenkorrektur (0 bis 63)

IKLoadPhase
Diese Funktion liefert den nichtflüchtig gespeicherten Wert der Phasenkorrektur zurück.
Prototyp: BOOL IKLoadPhase (USHORT Axis, BYTE* pData);
Axis: Nummer der Achse (0 bis 15)
pData: Zeiger auf eine „Byte-Variable“ (8 Bit), in welcher die Phasenkorrektur abgelegt wird.

IKReadAmp
Diese Funktion liefert die momentane Einstellung der Amplitudenkorrektur zurück.
Prototyp: BOOL IKReadAmp (USHORT Axis, BYTE* pData);
Axis: Nummer der Achse (0 bis 15)
pData: Zeiger auf eine „Byte-Variable“ (8 Bit), in welcher die Amplitudenkorrektur abgelegt wird.

IKWriteAmp
Diese Funktion ändert die momentane Einstellung des Amplitudenkorrektur.
Prototyp: BOOL IKWriteAmp (USHORT Axis, BYTE Data);
Axis: Nummer der Achse (0 bis 15)
Data: Neuer Wert der Amplitudenkorrektur (0 bis 63)

IKLoadAmp
Diese Funktion liefert den nichtflüchtig gespeicherten Wert der Amplitudenkorrektur zurück.
Prototyp: BOOL IKLoadAmp (USHORT Axis, BYTE* pData);
Axis: Nummer der Achse (0 bis 15)
pData: Zeiger auf eine „Byte-Variable“ (8 Bit), in welcher die Amplitudenkorrektur abgelegt wird.

IKReadOffset
Diese Funktion liefert die momentane Einstellung der Offsetkorrektur zurück.

Prototyp: **BOOL IKReadOffset (USHORT Axis, SHORT* Ofs0, SHORT* Ofs90);**

Axis: Nummer der Achse (0 bis 15)
Ofs0: Zeiger auf eine „Short-Variable“ (16 Bit), in welcher die Offsetkorrektur des 0-Grad-Signals abgelegt wird.
Ofs90: Zeiger auf eine „Short-Variable“ (16 Bit), in welcher die Offsetkorrektur des 90-Grad-Signals abgelegt wird.

IKWriteOffset
Diese Funktion ändert die momentane Einstellung der Offsetkorrektur.

Prototyp: **BOOL IKWriteOffset (USHORT Axis, SHORT Ofs0, SHORT Ofs90);**

Axis: Nummer der Achse (0 bis 15)
Ofs0: Neuer Wert der Offsetkorrektur des 0-Grad-Signals (–63 bis +63)
Ofs90: Neuer Wert der Offsetkorrektur des 90-Grad-Signals (–63 bis +63)

IKLoadOffset
Diese Funktion liefert den nichtflüchtig gespeicherten Wert der Offsetkorrektur zurück.

Prototyp: **BOOL IKLoadOffset (USHORT Axis, SHORT* Ofs0, SHORT* Ofs90);**

Axis: Nummer der Achse (0 bis 15)
Ofs0: Zeiger auf eine „Short-Variable“ (16 Bit), in welcher die Offsetkorrektur des 0-Grad-Signals abgelegt wird.
Ofs90: Zeiger auf eine „Short-Variable“ (16 Bit), in welcher die Offsetkorrektur des 90-Grad-Signals abgelegt wird.

IKStore
Diese Funktion überträgt alle momentan eingestellten Korrekturwerte in einen nichtflüchtigen Speicher.
Die Phasen- und Amplituden-Korrekturwerte aktiviert die IK 121 automatisch beim Einschalten des PCs. Die Offset-Korrekturwerte werden beim Initialisieren der IK 121 (Funktion: IKInit) aktiviert.

Prototyp: **BOOL IKStore (USHORT Axis);**

Axis: Nummer der Achse (0 bis 15)

IKDefault
Diese Funktion setzt alle Korrekturwerte auf neutrale Werte (Phase=31, Amplitude=31 und Offset=0). Dieser Zustand wird in den nichtflüchtigen Speicher übernommen.

Prototyp: **BOOL IKDefault (USHORT Axis);**
Axis: Nummer der Achse (0 bis 15)

IKRomRead
Diese Funktion liest aus dem EEPROM einen 8 Bit-Wert.

Prototyp: **BOOL IKRomRead (USHORT Card, BYTE Adr, BYTE*pData);**
Card: Nummer der IK 121 (0 bis 7)
Adr: Adresse im EEPROM (0 bis 255)
pData: Zeiger auf eine „Byte-Variable“ (8 Bit), in welcher der Wert abgelegt wird.

IKRomWrite
Diese Funktion schreibt einen 8 Bit-Wert in das EEPROM.

Prototyp: **BOOL IKRomWrite (USHORT Card, BYTE Adr, BYTE Data);**
Card: Nummer der IK 121 (0 bis 7)
Adr: Adresse im EEPROM (0 bis 255)
Data: Wert (8 Bit) welcher im EEPROM abgespeichert wird.

IKInputW
Diese Funktion liest ein Wort eines Registers.

Prototyp: **BOOL IKInputW (USHORT Axis, USHORT Adr, USHORT* pBuffer);**
Axis: Nummer der Achse (0 bis 15)
Adr: Adresse des Registers (0 bis 30 bzw. 0 bis 0x1E)
pBuffer: Zeiger auf ein Wort (16 Bit) in welchem der gelesene Wert abgelegt wird.

IKInputL
Diese Funktion liest ein Langwort eines Registers.

Prototyp: **BOOL IKInputL (USHORT Axis, USHORT Adr, ULONG* pBuffer);**
Axis: Nummer der Achse (0 bis 15)
Adr: Adresse des Registers (0 bis 28 bzw. 0 bis 0x1C)
pBuffer: Zeiger auf ein Langwort (32 Bit), in dem der gelesene Wert abgelegt wird.

IKOutput

Diese Funktion schreibt ein Wort in ein Register.

Prototyp: **BOOL IKOutput (USHORT Axis, USHORT Adr, USHORT Data);**

Axis: Nummer der Achse (0 bis 15)

Adr: Adresse des Registers (0 bis 30 bzw. 0 bis 0x1E)

Data: Wert (16 Bit), der in das Register geschrieben wird.

IKSetI2C

Diese Funktion setzt die Leitungen des I^2C-Bus, d.h. man kann die Daten- und Clockleitung setzen oder zurücksetzen. Damit lassen sich die Potentiometer und das EEPROM direkt ansprechen.

Prototyp: **BOOL IKSetI2C (USHORT Card, BOOL SCL, BOOL SDA);**

Card: Nummer der IK 121 (0 bis 7)

SCL: Zustand der Clockleitung
FALSE(=0)= Low
TRUE(<>0)=High

SDA: Zustand der Datenleitung
FALSE(=0)= Low
TRUE(<>0)=High

IKDefine

Diese Funktion legt die Portadressen der installierten IK 121 fest. Für jede IK 121 muss die Portadresse an der entsprechenden Position in pBuffer8 abgelegt werden Unbenutzte Einträge müssen auf 0 gesetzt werden. **Nur für Win32s unter Windows 3.1/3.11, da dort keine Registry existiert!!!**

Prototyp: **BOOL IKDefine (ULONG* pBuffer8);**

pBuffer8: Zeiger auf 8 Langworte (8*4 Byte)

IKSetTimer

Diese Funktion legt das Zeitintervall für den Timer fest.

Prototyp: **BOOL IKSetTimer (USHORT Axis, USHORT SetVal);**

Axis: Nummer der Achse (0 bis 15)

SetVal 0 = Timer gestoppt
1 bis 8192 = Timerwert in Mikrosekunden

IKEnableLatch
Diese Funktion legt fest, welches Einspeicher-Signal den Positionswert speichert.

Prototyp: **BOOL IKEnableLatch (USHORT Axis, USHORT Latch, USHORT Source);**

Axis: Nummer der Achse (0 bis 15)
Latch: 0 = Einspeicher-Signal freigeben für Register 0
1 = Einspeicher-Signal freigeben für Register 1
Source: 0 = Alle Einspeicher-Signale gesperrt
1 = Freigabe externes Signal, ohne Verzögerung (Slave)
2 = Freigabe externes Signal, mit Verzögerung (Master)
3 = Freigabe Software-Abruf
4 = Freigabe Timer

IKEnableSync
Diese Funktion legt fest, welches Einspeicher-Signal zur 2. Achse weitergeleitet wird.

Prototyp: **BOOL IKEnableSync (USHORT Card, USHORT Source);**

Card: Nummer der IK 121 (0 bis 7)
Source: 0 = Keine Kaskadierung
1 = Kaskadierung externes Signal
2 = Kaskadierung Software-Abruf
3 = Kaskadierung Timer

IKLatchAll
Diese Funktion erzeugt einen Software-Abruf mit dem die Positionswerte mehrerer Achsen gespeichert werden können.

Prototyp: **BOOL IKLatchAll (USHORT Axis);**

Axis: Nummer der Achse (0 bis 15)

Technische Daten

Mechanische Kennwerte	
Abmessungen	158 mm • 107 mm
Arbeitstemperatur	0 °C bis 45 °C
Lagertemperatur	–30 °C bis 70 °C

Elektrische Kennwerte	
Eingänge/Ausgänge	
externe Abruf-Signale:	X3: Flanschdose, 4-polig
2 Eingänge:	U_H: 3,15 V bis 30 V U_L: –3,0 V bis 0,9 V
1 Ausgang:	U_H: 4,0 V bis 32 V U_L: 0 V bis 1,0 V
Messgerät-Ausgänge:	sinusförmige Stromsignale (11 μA_{SS}) über zusätzliche Baugruppe für jede Achse
IK 121 A	
Messgerät-Eingänge:	X1: Achse 1, Sub-D-Anschluss 9-polig; Sinus-Signale: 7 μA_{SS} bis 16 μA_{SS} X2: Achse 2, Sub-D-Anschluss 9-polig; Sinus-Signale: 7 μA_{SS} bis 16 μA_{SS}
Eingangs-Frequenz:	max. 100 kHz
Kabellänge:	max. 10 m
IK 121 V	
Messgerät-Eingänge:	X1: Achse 1, Sub-D-Anschluss 15-polig; Sinus-Signale: 0,6 V_{SS} bis 1,2 V_{SS} X2: Achse 2, Sub-D-Anschluss 15-polig; Sinus-Signale: 0,6 μV_{SS} bis 1,2 μV_{SS}
Eingangs-Frequenz:	max. 400 kHz
Kabellänge:	max. 30 m
Kabel bis 150 m sind möglich, falls durch eine externe Versorgung gewährleistet ist, dass 5 V am Messgerät anliegen. Die Eingangsfrequenz reduziert sich in diesem Fall auf 250 kHz.	
Signal-Interpolation	1024fach

Abgleich der Messgerät-Signale	**Phase und Amplitude über elektronische Potentiometer Offset über Register in den Zählerbausteinen**
Datenregister für Messwerte	48 Bit
Port-Adressen	über DIP-Schalter einstellbar; die IK 121 belegt 16 Adressen
Interrupts	IRQ5, IRQ9, IRQ10, IRQ11, IRQ12 oder IRQ15
Leistungsaufnahme	ca. 1 W, ohne Messgeräte

Software	
Treiber-Software und Demonstrationsprogramme	zur Unterstützung der Programmierung für: **DOS-Anwendungen** in „TURBO PASCAL“ und „BORLAND C++“ **WINDOWS NT / 95** in VISUAL C++, VISUAL BASIC und BORLAND DELPHI

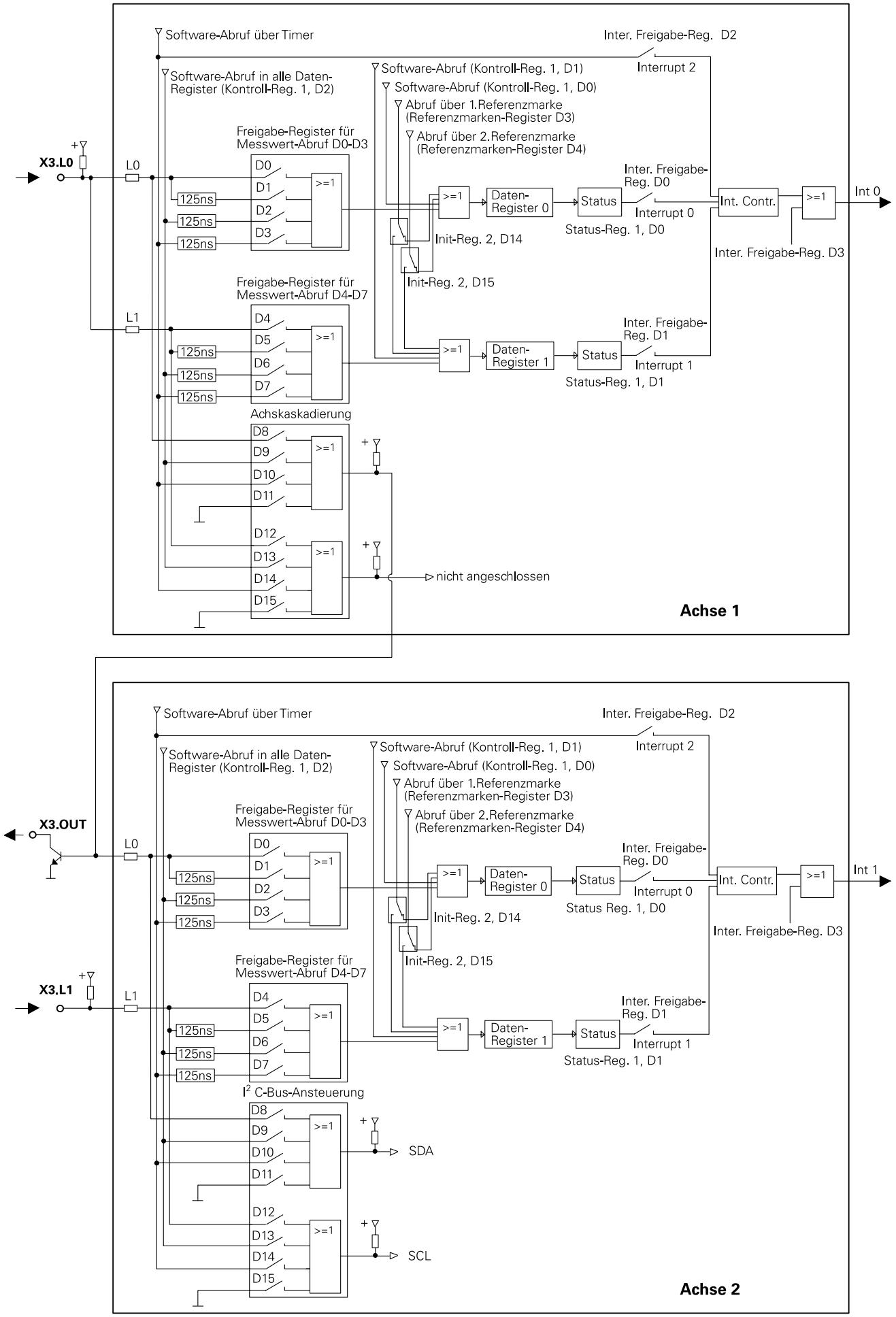
Stichwortverzeichnis

Prinzip-Schaltbild der Abruf-Wege in den Zählerbausteinen

Das folgende Prinzip-Schaltbild zeigt:

- die Wirkungsweise der Abruf-Signale auf die Daten-Register
- die Funktion der einzelnen Bits des Freigabe-Registers für den Messwert-Abruf
- die Achs-Kaskadierung mit den zugehörigen Bits des gleichnamigen Registers
- die Bildung der Interrupts
- das Register zur I²C-Bus-Ansteuerung

Die Verzögerungsglieder mit 125 ns Verzögerungszeit werden beim synchronen Einspeichern von beiden Achsen zum Ausgleich der Signallaufzeit zwischen Achse 1 und Achse 2 genutzt. Deshalb sollte beim synchronen Einspeichern für Achse 1 ein Signalweg über ein Verzögerungsglied und für Achse 2 ohne Verzögerungsglied gewählt werden. Da für Achse 1 und Achse 2 der gleiche Zählerbaustein verwendet wird, sind in beiden Achsen Verzögerungsglieder vorhanden, d.h. nicht alle Signalweg-Kombinationen ergeben einen sinnvollen Abruf!


Software-Abruf über Timer
Inter. Freigabe-Reg. D2
Software-Abruf in alle Daten-Register (Kontroll-Reg. 1, D2)
Software-Abruf (Kontroll-Reg. 1, D1)
Interrupt 2
Software-Abruf (Kontroll-Reg. 1, D0)
Abruf über 1.Referenzmarke (Referenzmarken-Register D3)
Abruf über 2.Referenzmarke (Referenzmarken-Register D4)
Freigabe-Register für Messwert-Abruf D0-D3
X3.L0
L0
D0
D1
D2
D3
125ns
>=1
Daten-Register 0
Status
Inter. Freigabe-Reg. D0
Interrupt 0
Status-Reg. 1, D0
Int. Contr.
Int 0
Init-Reg. 2, D14
Init-Reg. 2, D15
Inter. Freigabe-Reg. D3
Freigabe-Register für Messwert-Abruf D4-D7
L1
D4
D5
D6
D7
Daten-Register 1
Inter. Freigabe-Reg. D1
Interrupt 1
Status-Reg. 1, D1
Achskaskadierung
D8
D9
D10
D11
D12
D13
D14
D15
nicht angeschlossen
Achse 1
Software-Abruf über Timer
Inter. Freigabe-Reg. D2
Software-Abruf in alle Daten-Register (Kontroll-Reg. 1, D2)
Software-Abruf (Kontroll-Reg. 1, D1)
Interrupt 2
Software-Abruf (Kontroll-Reg. 1, D0)
Abruf über 1.Referenzmarke (Referenzmarken-Register D3)
Abruf über 2.Referenzmarke (Referenzmarken-Register D4)
Freigabe-Register für Messwert-Abruf D0-D3
X3.OUT
L0
D0
D1
D2
D3
125ns
>=1
Daten-Register 0
Status
Inter. Freigabe-Reg. D0
Interrupt 0
Status Reg. 1, D0
Int. Contr.
Int 1
Init-Reg. 2, D14
Init-Reg. 2, D15
Inter. Freigabe-Reg. D3
Freigabe-Register für Messwert-Abruf D4-D7
X3.L1
L1
D4
D5
D6
D7
Daten-Register 1
Inter. Freigabe-Reg. D1
Interrupt 1
Status-Reg. 1, D1
I2 C-Bus-Ansteuerung
D8
D9
D10
D11
D12
D13
D14
D15
SDA
SCL
Achse 2

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

Technical support FAX +49 (8669) 31-1000
e-mail: service@heidenhain.de
Measuring systems ☏ +49 (8669) 31-3104
e-mail: service.ms-support@heidenhain.de
TNC support ☏ +49 (8669) 31-3101
e-mail: service.nc-support@heidenhain.de
NC programming ☏ +49 (8669) 31-3103
e-mail: service.nc-pgm@heidenhain.de
PLC programming ☏ +49 (8669) 31-3102
e-mail: service.plc@heidenhain.de
Lathe controls ☏ +49 (711) 952803-0
e-mail: service.hsf@heidenhain.de

www.heidenhain.de

Ve00
291019-05· 4·5/2004· F&W· Printed in Germany· Änderungen vorbehalten