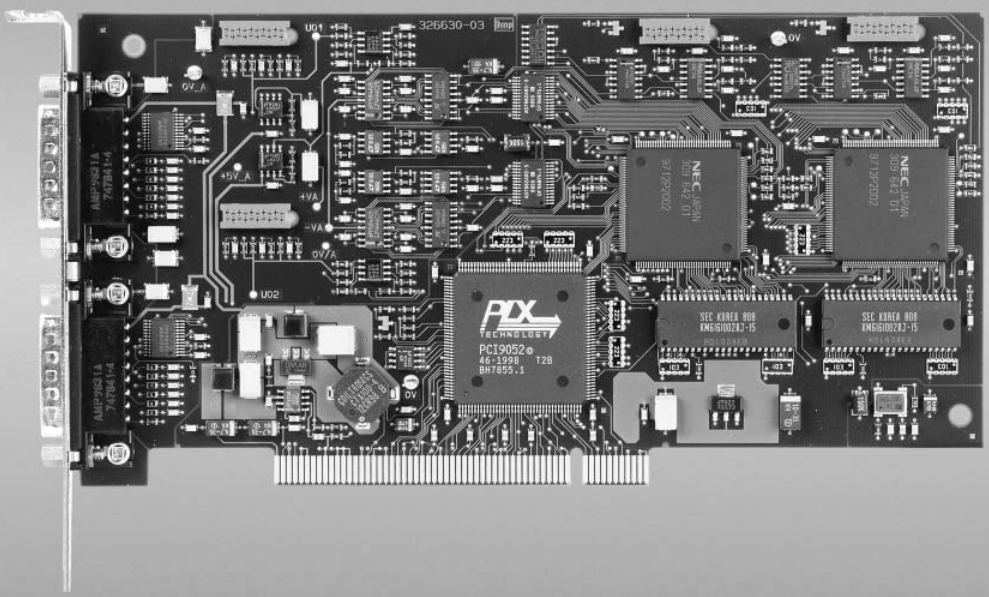


HEIDENHAIN

Benutzer-Handbuch
User's Manual
IK 220

PC-Zählerkarte zum Anschluss von HEIDENHAIN-Messgeräten

PC Counter Card for HEIDENHAIN encoders

5/2011

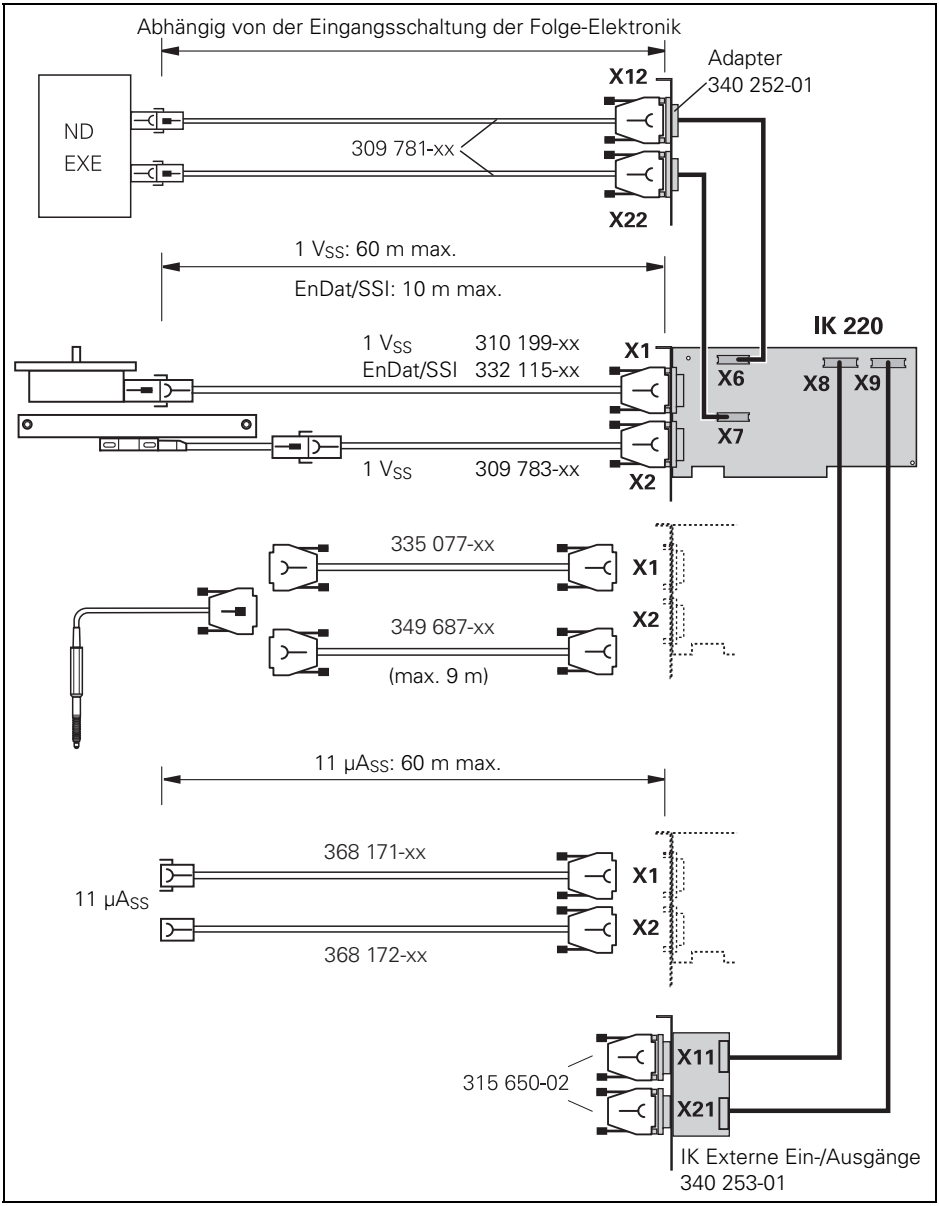
Inhalt

Lieferumfang

PC-Zählerkarte IK 220 — Id.-Nr. 337 481-01
Programmierbeispiele, Treiber-Software und Benutzer-Handbuch.

Zubehör

- IK Externe Ein-/Ausgänge — Id.-Nr. 340 253-01
- Stecker für externe Ein-/Ausgänge — Id.-Nr. 315 650-02
- Adapterkabel mit Stecker für HEIDENHAIN-Messgeräte
  - mit sinusförmigen Spannungssignalen 1 V_{SS} — Id.-Nr. 310 199-xx
  - mit sinusförmigen Stromsignalen 11 μA_{SS} — Id.-Nr. 368 171-xx
  - mit EnDat-Schnittstelle — Id.-Nr. 332 115-xx
- Adapterkabel mit Kupplung für HEIDENHAIN-Messgeräte
  - mit sinusförmigen Spannungssignalen 1 V_{SS} — Id.-Nr. 309 783-xx
  - mit sinusförmigen Stromsignalen 11 μA_{SS} — Id.-Nr. 368 172-xx
- Zusätzlicher Sub-D-Anschluss zur Weiterführung der Messgerät-Signale des Eingangs X1 und X2 an eine weitere Anzeige oder Steuerung — Id.-Nr. 340 252-01
- Verbindungskabel vom zusätzlichen Sub-D-Anschluss an eine weitere Anzeige oder Steuerung — Id.-Nr. 309 781-xx


Abhängig von der Eingangsschaltung der Folge-Elektronik
Adapter
340 252-01
X12
ND
EXE
309 781-xx
X22
1 VSS: 60 m max.
EnDat/SSI: 10 m max.
IK 220
1 VSS 310 199-xx
EnDat/SSI 332 115-xx
X1
X6
X8
X9
X7
1 VSS 309 783-xx
X2
335 077-xx
X1
X2
349 687-xx
(max. 9 m)
11 µASS: 60 m max.
368 171-xx
11 µASS
X1
X2
368 172-xx
X11
315 650-02
X21
IK Externe Ein-/Ausgänge
340 253-01

Wichtige Hinweise

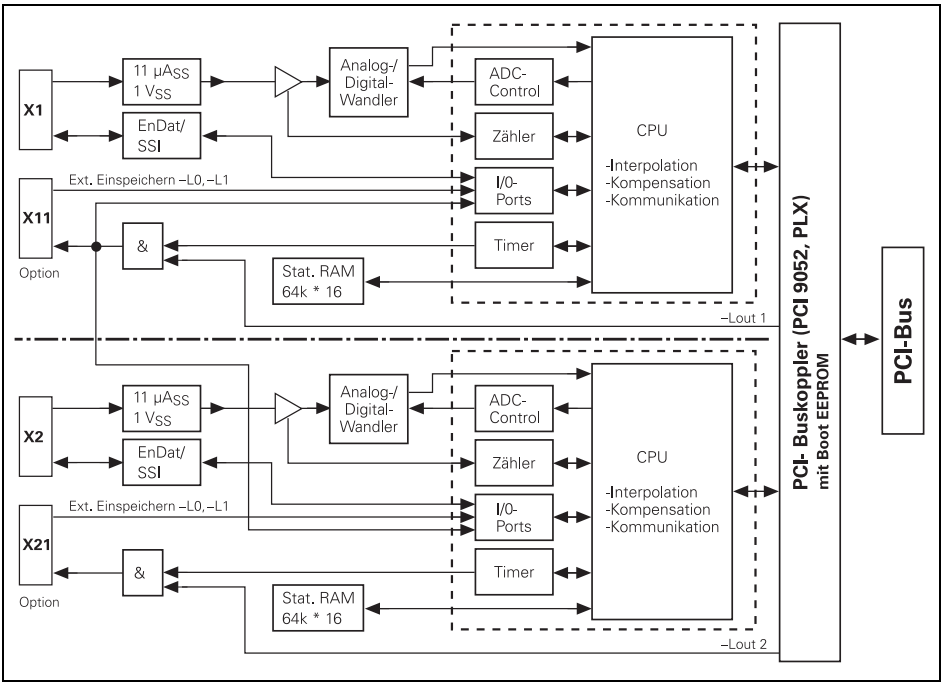
Das EnDat-Interface bietet die Möglichkeit im Speicherbereich des Kunden maschinen- oder anlagenspezifische Daten zu hinterlegen (z. b. Nullpunktverschiebung, OEM-Daten, …). Diese Daten können sicherheitsrelevante Informationen beinhalten. Bei fehlerhafter Behandlung dieser Daten können Maschinen- oder Personenschäden die Folge sein.
Informationen zur Schnittstelle EnDat und zum Messgerät entnehmen Sie bitte der EnDat-Spezifikation, der Montageanleitung des Messgerätes sowie der Produktbeschreibung (z. B. Katalog, Produktinformation). Für weitere Informationen wenden Sie sich bitte an Ihren Ansprechpartner im Vertrieb.

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.

Technische Beschreibung der IK 220

Die PC-Zählerkarte IK 220 wird direkt in einen Erweiterungs-Steckplatz eines Personal Computers mit PCI-Interface gesteckt. Es können zwei HEIDENHAIN-Messgeräte mit sinusförmigen Stromsignalen (11 μA_{SS}), Spannungssignalen (1 V_{SS}), EnDat- oder SSI-Schnittstelle angeschlossen werden. Die Positionen der beiden Messgeräte werden mittels Software am PC angezeigt, im PC gespeichert und weiterverarbeitet.
Die IK 220 ist ideal für Anwendungen, bei denen eine hohe Auflösung der Messgerät-Signale und eine schnelle Messwert-Erfassung erforderlich ist.

Blockschaltbild der IK 220



Die Interpolations-Elektronik in der IK 220 unterteilt die Signalperiode des Eingangs-Signals bis zu 4096fach.
Der Interpolationswert (12 Bit) bildet zusammen mit dem Wert des Periodenzählers (32 Bit) den 44 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, per Software oder Timer sowie durch das Überfahren der Referenzmarken abgerufen und gespeichert.

In den Zählerbausteinen befindet sich eine CPU, die zugehörige Firmware wird auf die Zählerkarte geladen.

Zykluszeit der Firmware: 25 µs

Offset, Phase und Amplitude der sinusförmigen Messgerät-Signale können per Software abgeglichen werden.

Zugriffszeit auf Messwerte

Inkrementale Messgeräte

- Ohne Kompensation der Messgerätsignale, ohne Ermittlung der Korrekturwerte:
  Max. 100 µs
- Mit Kompensation der Messgerätsignale, ohne Ermittlung der Korrekturwerte:
  Max. 110 µs
- Mit Kompensation der Messgerätsignale, mit Ermittlung der Korrekturwerte:
  Max. 160 µs

EnDat/SSI

Die Zugriffszeit ist abhängig vom Messgerät.
EnDat: 440 kHz Clock-Frequenz (One time call)
SSI: 360 KHz Clock-Frequenz

Hardware

Spezifikation des PCI-Bus

Die IK 220 kann in alle PCs mit PCI-Bus eingesetzt werden.

Spezifikation	PCI Local Bus Specification Rev. 2.1
Größe	ca. 100 * 190 mm
Stecker	PCI 5 V / 32 Bit (2*60) Steckverbinder
PCI-Baustein	PCI 9052 von PLX, Target Interface (Slave)
Stromaufnahme	+12 V: 22 mA[1)] -12 V: 22 mA +5 V: 620 mA
Leistungsaufnahme	< 4 Watt, ohne Messgeräte

[1)] ohne Stromaufnahme der angeschlossenen Messgeräte

Kennungen im Baustein PCI9052:
- Vendor ID = 0x10B5
- Device ID = 0x9050
- Sub Vendor ID = 0x10B5
- Sub Device ID = 0x1172

Anhand dieser 4 Kennungen ist die IK 220 eindeutig zu identifizieren. Die „Sub Device ID“ ist exklusiv der IK 220 zugeordnet.

Messgerät-Eingänge

An die IK 220 können Messgeräte mit folgenden Schnittstellen angeschlossen werden:

- 11 μA_{SS}
- 1 V_{SS}
- EnDat 2.1
- SSI

Die Spannungsversorgung der Messgeräte (typ. 5,12 V) wird aus den +12 V des PCI-Bus erzeugt. Aus der +5 V Versorgung für die Messgeräte dürfen max. 800 mA für beide Achsen entnommen werden.
Die zusätzliche Stromaufnahme aus den +12 V des PCI-Bus ergibt sich zu: $I_{(12V)} = I_{(5V)} * 5{,}12V * 1{,}35 / 12V$
Beispiel: $I_{(5V)} = 0.8A * 5{,}12\ V * 1{,}35 / 12V = 461mA$

Es ist zu beachten, dass die Grenzen der Spannungsversorgung am Messgerät eingehalten werden. Bei großen Kabellängen kann ein Spannungskonverter (370 225-xx) eingesetzt werden. Der Spannungskonverter hat einen Wirkungsgrad von ca. 72%. Dadurch reduziert sich die erlaubte Stromaufnahme für beiden Achsen auf 0,72 * 800 mA = 576 mA.

Spezifikation der 11 μA_{SS}-Schnittstelle

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	Standard: 33 kHz, umschaltbar auf 175 kHz
Kabellänge	max. 60 m

Spezifikation der 1 V_{SS}-Schnittstelle

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	Standard: 500 kHz, umschaltbar auf 33 kHz
Kabellänge[1)]	max. 60 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.

Spezifikation der EnDat 2.1-Schnittstelle

Die EnDat 2.1-Schnittstelle der absoluten Messgeräte ist bidirektional. Sie liefert die absoluten Positionswerte und ermöglicht den messgerätinternen Speicher zu lesen oder zu beschreiben. Zusätzlich stehen sinusförmige Spannungssignale (1 V_{SS}) zur Verfügung.

Kabellänge: max. 10 m
max. 50 m[2)]

[2)] Mit Original-HEIDENHAIN-Kabeln. Versorgungsspannungsgrenzen des Messgerätes beachten.

Spezifikation der SSI-Schnittstelle

Die SSI-Schnittstelle der absoluten Messgeräte ist unidirektional. Sie liefert die absoluten Positionswerte synchron zu einem von der Folge-Elektronik vorgegebenen Takt. Zusätzlich stehen sinusförmige Spannungssignale (1 V_{SS}) zur Verfügung. Kabellänge: max. 10 m

Anschluss X1, X2 für Messgeräte

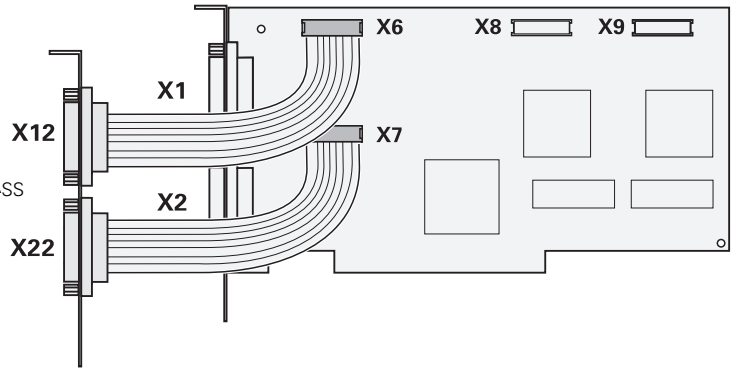
Sub-D-Anschluss mit Stifteinsatz (15-polig)

Anschluss-Nr.	Belegung		
	EnDat/SSI	**1 V_{SS}**	**11 μA_{SS}**
1	+5 V (U_P)	+5 V (U_P)	+5 V
2	0 V (U_N)	0 V (U_N)	0 V
3	A+	A+	I_1+
4	A-	A-	I_1-
5	+ Daten	nicht belegen	nicht belegen
6	B+	B+	I_2+
7	B-	B-	I_2-
8	- Daten	nicht belegen	nicht belegen
9	+5 V (Fühlleitung)	+5 V	+5 V
10	nicht belegen	R+	I_0+
11	0 V (Fühlleitung)	0 V	0 V
12	nicht belegen	R-	I_0-
13	Innenschirm	0 V	Innenschirm
14	+ Takt	nicht belegen	nicht belegen
15	- Takt	nicht belegen	nicht belegen
Gehäuse	Außenschirm	Außenschirm	Außenschirm

Messgerät-Ausgänge

Die IK 220 gibt die Messgerät-Signale der Eingänge X1 und X2 zusätzlich an zwei 10-polige MICROMATCH-Stecker (Buchse) auf der Platine als **sinusförmige Stromsignale** (11 μA_{SS}) aus. Über eine zusätzliche Kabelbaugruppe mit PC-Slot-Abdeckung (Id.-Nr. 340 252-01) 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“).

Messgerät-Ausgänge 11 μA_{SS}



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

Platinenstecker für Messgerät-Ausgänge
Anschluss X6, X7
MICROMATCH mit Buchseneinsatz 10-polig

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

1) Pin 1a befindet sich auf der Seite mit dem Kodierstift.

Messgeräte-Ausgänge (Ident-Nummer 340 252-01)
Sub-D-Anschluss mit Stifteinsatz (9-polig)

Anschluss-Nr.	Signal
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

Kompensation der Messgerät-Signale

Messgerät-Signale können automatisch – auch online kompensiert werden. In der mitgelieferten Software sind entsprechende Funktionen enthalten.

Externe Ein-/Ausgänge

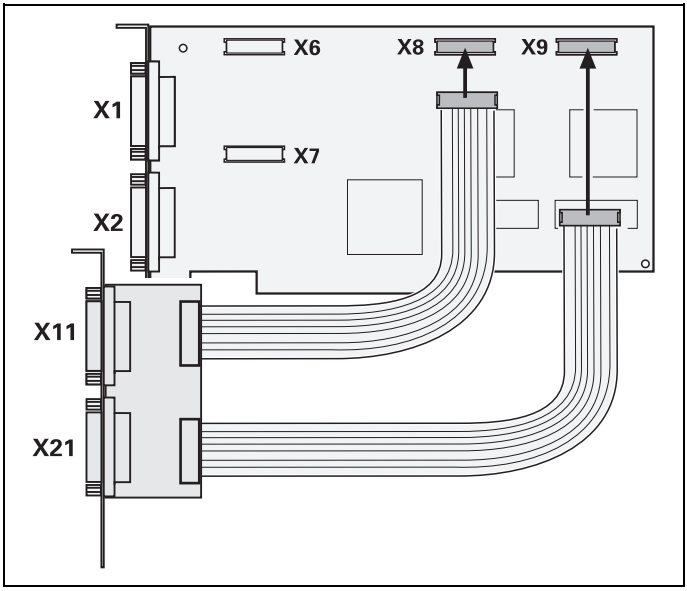
Für externe Ein-/Ausgänge ist eine zusätzliche Kabelbaugruppe mit PC-Slot-Abdeckung (IK externe Ein-/Ausgänge Id.-Nr. 340 253-01) lieferbar.

Anschluss X8, X9 für externe Ein-/Ausgänge

Anschluss-Nr.	Signal
1a	-L0
1b	0 V
2a	-L1
2b	0 V
3a	-I0
3b	0 V
4a	-I1
4b	-LOUT
5a	+5 V
5b	+5 V

Anschluss X11 und X21 für externe Ein-/Ausgänge (Option)
Sub-D-Anschluss mit Stifteinsatz (9-polig) auf PC-Slot-Abdeckung

Für externe Ein- und Ausgänge wird optional eine Baugruppe geliefert, bestehend aus einer Slot-Abdeckung mit zwei Sub-D-Anschlüssen, einer Entstörplatine und zwei Flachbandkabel zum Anschluss an 10-polige MICROMATCH-Stecker auf der Platine.



Anschluss-Nr.	Belegung
1	Ausgang: Messwert-Abruf (X11: -Lout 1; X21: -Lout 2)
2	Eingang: Messwert-Abruf -L0
3	Eingang: Messwert-Abruf -L1
4, 5	nicht belegen
6	Eingang: –I0 [1)]
7	Eingang: –I1 [1)]
8 , 9	0 V

[1)]zusätzliche Schalteingänge. Siehe Funktion IK220GetPort

Abruf der Messwerte über externe Eingänge
Die IK 220 hat zwei externe Eingänge zum Abrufen und Speichern der Messwerte.

Die Eingänge -L0 und -L1 sind low-aktiv; ein interner Pull-up-Widerstand (1,47 kΩ) hält sie auf High-Pegel. Sie können an TTL-Bausteine angeschlossen werden.

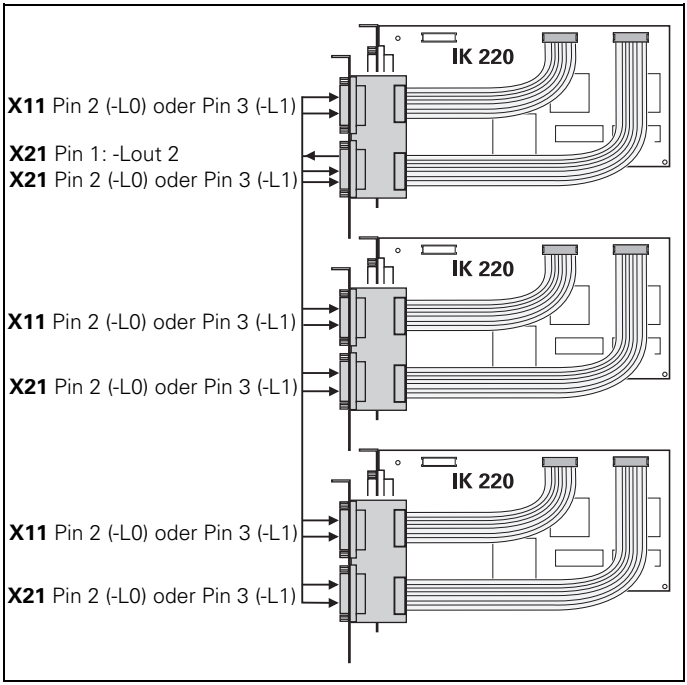
Die einfachste Art, die Eingänge zu aktivieren: Eine Brücke von 0 Volt (Anschlüsse 8, 9) auf den Eingang zum Abrufen.

Abruf-Ausgänge -Lout
Die IK 220 liefert zwei Ausgangssignale: -Lout 1 (an Sub-D-Anschluss X11) und -Lout 2 (an Sub-D-Anschluss X21).
-Lout 1/2 sind low-aktiv.
-Lout 1 liefert gleichzeitig mit dem synchronen Einspeichern der Messwerte (IK220LatchInt) oder mit dem Einspeichern per Timer einen low-aktiven Impuls.

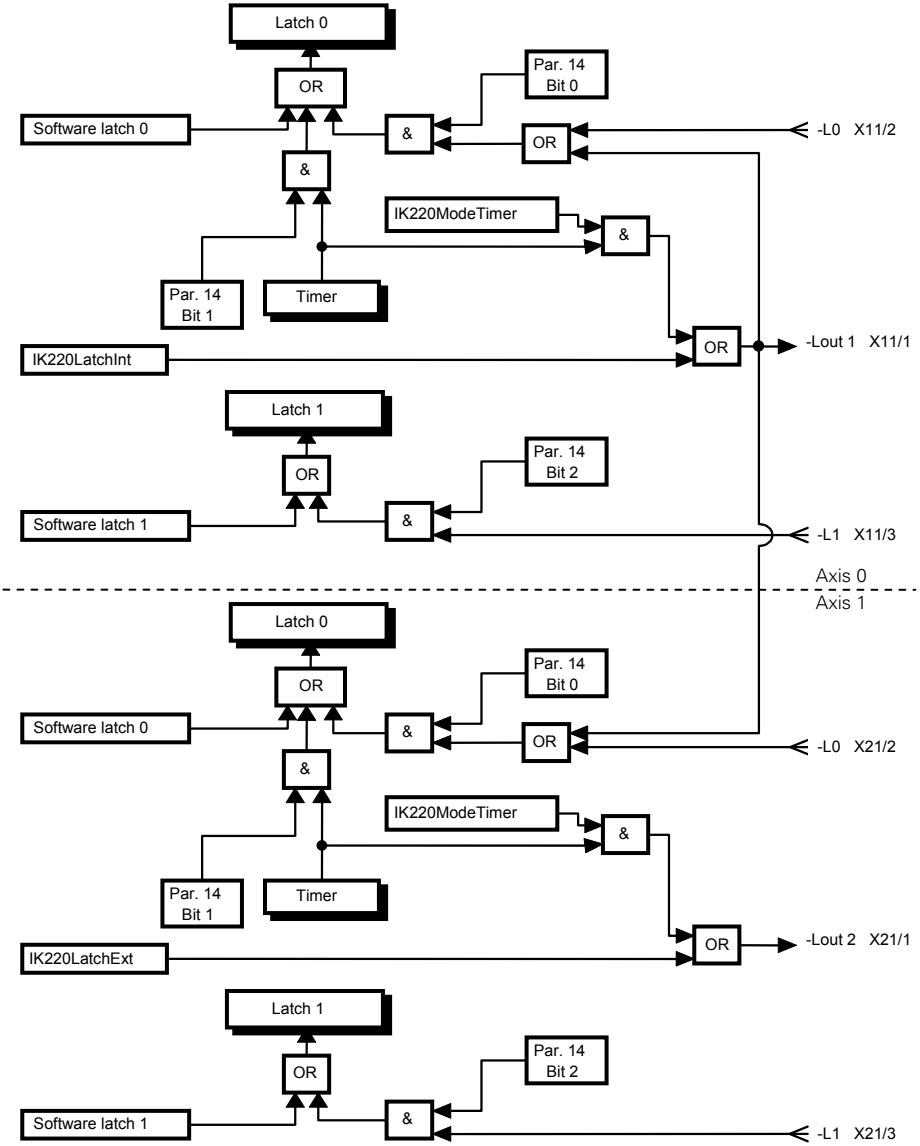
Um die Messwerte verschiedener IKs gleichzeitig einzuspeichern (IK220LatchExt) müssen Sie -Lout 2 verwenden (siehe nächste Seite).

Messwerte von mehreren IK 220 abrufen

Damit die Messwerte aller Achsen mehrerer IKs gleichzeitig gespeichert werden, muss das Ausgangs-Signal -Lout 2 zu allen beteiligten Messgerät-Eingängen (-L0 oder –L1) geführt werden – auch zu dem Eingang von dem -Lout 2 herausgeführt wird. Dadurch erfolgt das Einspeichern auf allen Achsen gleichzeitig – ohne Laufzeitunterschiede.



Ablaufdiagramm: Speichern von Messwerten



Betriebsparameter

Die IK 220 benötigt Betriebs-Parameter um die gewünschten Funktionen richtig ausführen zu können. Es sind Defaultwerte voreingestellt, die beim Download der mitgelieferten Betriebssoftware gesetzt werden. Die Defaultwerte sind in der folgenden Tabelle fett dargestellt. Parameterwerte lassen sich mit der Funktion IK220WritePar (Parameter schreiben) ändern und mit der Funktion IK220ReadPar (Parameter lesen) kann man die geänderten Werte wieder kontrollieren.

Folgende Parameter stehen zur Verfügung (die Defaultwerte sind fett gedruckt):

Param. Nummer	Format	Bedeutung
1	16 Bit	**0: inkrementales Messgerät** 1: EnDat 2.1-Messgerät 2: SSI-Messgerät
2	16 Bit	Parameter wirkt nur, wenn Parameter 1 = 0 0: 11 µA$_{SS}$ **1: 1 V$_{SS}$**
3	16 Bit	**0: Linearachse** [1)] 1: Winkelachse ± unendlich
4	16 Bit	**0: positive Zählrichtung** 1: negative Zählrichtung
5	32 Bit	0 bis 0xFFFFFFFF: Signalperioden/Umdrehung bei Winkelachse, wirkt nur wenn Parameter 3 = 1 [1)] **Defaultwert 0**
6	16 Bit	**0: einzelne REF-Marke** Abstandscodierte REF-Marken: [2)] Angabe des Grundabstandes als Festabstand in Teilungsperioden (TP) typische Beispiele 500: Festabstand 500TP 1000: Festabstand 1000TP 2000: Festabstand 2000TP 5000: Festabstand 5000TP

Param. Nummer	For-mat	Bedeutung
7	16 Bit	0 bis 12: Anzahl Interpolations-Bits **Defaultwert 12** Der Interpolationswert (16 Bit breit, max. 12 signifikante Bits, linksbündig) wird auf die Anzahl der eingestellten Bits gerundet und reduziert.
8	16 Bit	**0: Korrekturrechnung für Positionswert aus** 1: Korrekturrechnung für Positionswert ein
9	16 Bit	**0: Ermittlung Korrekturwerte aus** 1: Ermittlung Korrekturwerte ein
10	16 Bit	1 bis 8: Anzahl Messpunkte pro Oktant innerhalb einer Signalperiode für Berechnung der Korrekturwerte **Defaultwert 1**
11	16 Bit	16 bis 47: Zeitintervall für Timer **Defaultwert 33 entspricht 1 ms** 3)
12	16 Bit	Software-Teiler für Timer **Defaultwert 1** 3)
13	16 Bit	0 bis 8191: Anzahl der Werte pro Speicher-Zyklus die im internen RAM abgelegt werden **Defaultwert 8191**
14	16 Bit	Bit 0=1 (Integer 1): extern Latch L0 freigegeben Bit 1=1 (Integer 2): intern Timer Latch L0 freigegeben Bit 2=1 (Integer 4): extern Latch L1 freigegeben **Defaultwert 0**
15	16 Bit	1 bis 48: SSI Code-Länge in Bit **Defaultwert 19**
16	16 Bit	0: keine SSI-Parity **1: SSI-Parity (even)** 2: SSI-Parity (even) mit Vornullen
17	16 Bit	0: SSI disable Gray / Binär-Konvertierung **1: SSI enable Gray / Binär-Konvertierung**
18	16 Bit	0 bis 20: SSI Anzahl Vornullen **Defaultwert 0**
19	16 Bit	0 bis 20: SSI Anzahl Nachnullen **Defaultwert 0**

1) Parameter ist ohne Funktion

2) Abhängig vom angeschlossenen Messgerät (nur gerade 16-Bit-Werte möglich)

3) siehe nächste Seite

Parameter 11: Zeitintervall zwischen zwei Latches per Timer: Folgende Werte sind über die Timer der IK 220 direkt einstellbar:

Parameter-wert	Zeit-intervall	Parameter-wert	Zeit-intervall
17	100 µs[1)]	33	1000 µs
18	150 µs[2)]	34	1100 µs
19	200 µs	35	1200 µs
20	250 µs	36	1300 µs
21	300 µs	37	1400 µs
22	350 µs	38	1500 µs
23	400 µs	39	1600 µs
24	450 µs	40	1800 µs
25	500 µs	41	2000 µs
26	550 µs	42	2200 µs
27	600 µs	43	2400 µs
28	650 µs	44	2600 µs
29	700 µs	45	2800 µs
30	750 µs	46	3000 µs
31	800 µs	47	3200 µs
32	900 µs		

Parameter 12: Um ein Zeitintervall von mehr als 3,2 Millisekunden zu ermöglichen, wird mit Parameter 12 ein Zähler realisiert, der nur jeden n-ten Timer-Impuls zum Einspeichern verwendet. Um z.B. ein Einspeicherintervall von 9 Millisekunden zu ermöglichen, muss man Parameter 11 auf 46 (entspricht 3 Millisekunden) und Parameter 12 auf 3 setzen.

[1)] Nur möglich ohne Kompensation der Messgerätesignale und ohne Ermittlung der Korrekturwerte
[2)] Mit Kompensation; ohne Ermittlung der Korrekturwerte

Treiber-Software für WINDOWS

Allgemeines

Die Treiber-Software für die IK 220 ermöglicht es Anwendungen die IK 220 Zählerkarte von Windows 95/98, Windows NT/2000/XP, Linux und LabView aus anzusprechen.

Der Zugriff erfolgt bei Windows über eine Dynamic Link Library (DLL) und einen Windows 95/98, Windows NT oder Windows 2000/XP Device Treiber. Die Treiber und Anwendungsbeispiele befinden sich auf der mitgelieferten CD.

Der CD-Inhalt sowie Treiber für weitere Betriebssysteme (z. B. Windows Vista, Windows 7) befinden sich im Download-Bereich unter www.heidenhain.de. Für weitere Hinweise siehe „readme.txt“ des Download-Files.

Bei Fragen wenden Sie sich bitte an den HEIDENHAIN-Messgeräte-Support.

Inhalt Verzeichnis "Disk1":
- Installationsroutine Treiber NT und Win95/98
- DLL mit Sourcecode
- NT Treiber mit Sourcecode

Inhalt Verzeichnis "Disk2":
- Visual C++ Beispiel mit Sourcecode
- Konsolenbeispiel mit Sourcecode
- Visual Basic 5 Beispiel mit Sourcecode

Inhalt Verzeichnis "Disk3":
- Delphi 4 Beispiel mit Sourcecode

Inhalt Verzeichnis "Disk4":
- Installationsroutine Treiber Windows 2000/XP
- Windows 2000/XP Treiber (WDM) mit Sourcecode

Inhalt Verzeichnis "Disk5":
- LabView Bibliothek
- Beispielprogramme

Inhalt Verzeichnis "Disk6":
- Linuxtreiber für Kernel 2.4
- Beispielprogramme
- Beschreibung und Installationsanleitung
- Sourcecode

Installation des Treibers und der DLL unter Windows 2000 und XP

- Nachdem Sie die IK 220 Karte in den Rechner gesteckt haben, starten Sie Ihren Rechner neu.
- Folgen Sie den Anweisungen des automatischen Installationsassistenten
- Wählen Sie von der CD im Verzeichnis "Disk4" die Setup-Informationsdatei "IK220.inf" aus.
- Folgen Sie den Anweisungen des automatischen Installationsassistenten

Installation der Treiber und der DLL unter Windows NT und Windows 95/98

- Wählen Sie auf der mitgelieferten CD das Verzeichnis „Disk1\Install".
- Rufen Sie „Install.Bat" auf.

Device-Treiber für Windows 2000/XP (IK220DRV.SYS)

Der Windows 2000/XP Treiber ist ein WDM-Treiber für Windows 2000 und XP. Er ermöglicht den Zugriff auf die IK 220. Vom Treiber werden bis zu 8 IK 220 unterstützt. Zur Installation muss lediglich die Setupinformationsdatei (IK220.inf) im Verzeichnis "Disk4" ausgewählt werden. Der automatische Installationsassistent führt Sie Schritt für Schritt durch den Installationsprozess.

Device-Treiber für Windows NT (IK220DRV.SYS)

Der Windows NT Device Treiber ist ein Kernel Mode Treiber für Windows NT (Version 3.51 und 4.0). Er ermöglicht den Zugriff auf die IK 220. Vom Treiber werden bis zu 8 IK 220 unterstützt. Die Installation des Devicetreibers erledigt die Batch-Datei „Install.Bat" im Verzeichnis „Disk1\Install" im CD Verzeichniszweig "IK220".

Device-Treiber für Windows 95/98 (IK220VXD.VXD)
Der Windows 95/98 Device-Treiber ist ein virtueller Device-Treiber für Windows 95/98 der den Zugriff auf bis zu 8 IK 220 unterstützt. Die Installation des Devicetreibers erledigt die Batch-Datei „Install.Bat" im Verzeichnis „Disk1\Install" im CD Verzeichniszweig "IK220".

Die Windows DLL (IK220DLL.DLL)
Diese DLL ermöglicht es Anwendungsprogrammen die IK 220 anzusprechen. Es gibt jeweils eine DLL für Windows NT/2000/XP und eine für Windows 95/98. Unter Windows NT/2000/XP wird die IK 220 über den Device-Treiber für Windows NT/2000/XP angesprochen. Unter Windows 95/98 greift die DLL über den virtuellen Device-Treiber auf die Register der IK 220 zu.

Zur Installation des Gerätetreibers sind Administrationsrechte auf dem Zielrechner erforderlich!

Beispiele

Beispiel für Konsolen-Anwendung
Im Verzeichnis „\Disk2\IK220Con\Release“ auf der CD im Verzeichniszweig "IK 220" finden Sie eine einfache Konsolen-Anwendung: Starten Sie IK220Con.exe.
Achtung: Dieses Beispiel eignet sich nur für HEIDENHAIN-Messgeräte mit sinusförmigen Spannungssignalen 1 V_{SS}.

Beispiel für Visual C++
Im Verzeichnis „\Disk2\IK220App\Release“ auf der CD im Verzeichniszweig "IK 220" finden Sie eine Anwendung in Visual C++: Starten Sie IK220App.exe.
Wählen Sie die Schnittstelle (1 V_{SS}, 11 μA_{SS}, EnDat) und setzen Sie die Messgerät-Parameter unter „Setup“.

Beispiel für Visual Basic
Im Verzeichnis „\Disk2\IK220VB5“ auf der CD im Verzeichniszweig "IK 220" finden Sie eine Anwendung in Visual Basic: Starten Sie IK220App.exe.
Wählen Sie die Schnittstelle (1 V_{SS}, 11 μA_{SS}, EnDat) und setzen Sie die Messgerät-Parameter unter „Setup“.

Beispiel für Borland Delphi
Im Verzeichnis „\Disk3\Delphi“ auf der CD im Verzeichniszweig "IK 220" finden Sie eine Anwendung in Borland Delphi: Starten Sie IK220.exe.
Wählen Sie die Schnittstelle (1 V_{SS}, 11 μA_{SS}, EnDat, SSI) und setzen Sie die Messgerät-Parameter unter „Parameters/Encoder“.

Beispiele für LabView
Im Verzeichnis "\Disk5" auf der CD im Verzeichniszweig "IK 220" finden Sie Beispielapplikationen in LabView.

Beispiel für Linux
Im Verzeichnis "\Disk6" auf der CD im Verzeichniszweig "IK 220" finden Sie eine einfache Konsolen-Anwendung: Kompilieren und starten Sie ik220_read48.

Bei allen hier aufgezählten Applikationen handelt es sich um Beispiele, die Ihnen die Programmierung in der jeweiligen Sprache zeigen soll. Für einen produktiven Einsatz sind diese Beispiele nicht gedacht.

Aufruf der DLL-Funktionen aus einem Anwenderprogramm

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

Microsoft Visual C++

Wird das Anwenderprogramm mit Visual C++ erstellt, so ist die Datei „\IK220Dll\Release\IK220DLL.LIB" in das Library-Verzeichnis von Visual C++ zu kopieren (z.B.: „C:\MSDEV\LIB"). Außerdem muss diese Library mitgelinkt werden. Dazu ist ein Eintrag unter „Build/Settings/Link/ Object/library modules" nötig.
Das Header-File „\Include\DLLFunc.h", in dem die Funktions-Prototypen definiert sind, muss dem Projekt hinzugefügt werden.
Nachdem dies geschehen ist können die Funktionen wie „normale" C-Funktionen benutzt werden.

Microsoft Visual Basic

Für Microsoft Visual Basic sind die Funktionen im Modul „\Include\DLLFunc.bas" definiert. Diese Datei muss in das Projekt eingebunden werden.

Borland Delphi

In der Datei „\Include\DLLFunc.pas" sind die Funktionen und Typen definiert um die DLL-Funktionen mit Borland Delphi verwenden zu können.

Übersicht der DLL-Funktionen

Funktion	Kurzreferenz	
Installierte IK 220 feststellen	BOOL IK220Find	(ULONG* pBuffer16)
IK 220 initialisieren	BOOL IK220Init	(USHORT Axis)
Programm-Versionen lesen	BOOL IK220Version	(USHORT Axis, char* pVersCard, char* pVersDrv, char* pVersDll)
Zähler löschen	BOOL IK220Reset	(USHORT Axis)
Zähler starten	BOOL IK220Start	(USHORT Axis)
Zähler stoppen	BOOL IK220Stop	(USHORT Axis)
Frequenz- und Amplituden-Fehler löschen	BOOL IK220ClearErr	(USHORT Axis)
Zählerwert speichern	BOOL IK220Latch	(USHORT Axis, USHORT Latch)
Synchrones Zählerwert speichern intern	BOOL IK220LatchInt	(USHORT Card)

Funktion	Kurzreferenz	
Synchrones Zählerwert speichern extern	BOOL IK220LatchExt	(USHORT Card)
Zähler löschen mit nächster Referenzmarke	BOOL IK220ResetRef	(USHORT Axis)
Zähler starten mit nächster Referenzmarke	BOOL IK220StartRef	(USHORT Axis)
Zähler stoppen mit nächster Referenzmarke	BOOL IK220StopRef	(USHORT Axis)
Zählerwert speichern mit nächster Referenzmarke	BOOL IK220LatchRef	(USHORT Axis)
Abfrage ob Zählerwert gespeichert	BOOL IK220Latched	(USHORT Axis, USHORT Latch, BOOL* pStatus)
Warten bis Zählerwert gespeichert	BOOL IK220WaitLatch	(USHORT Axis, USHORT Latch)
Timeout-Zeit setzen	BOOL IK220SetTimeOut (ULONG TimeOut)	
Positionswert setzen	BOOL IK220Set	(USHORT Axis, double SetVal)
Presetwert setzen	BOOL IK220SetPreset	(USHORT Axis, double PresVal)
Presetwert lesen	BOOL IK220GetPreset	(USHORT AXIS, double* PresVal)
Zählerwert lesen (32 Bit)	BOOL IK220Read32	(USHORT Axis, USHORT Latch, LONG* pData)
Zählerwert lesen (48 Bit)	BOOL IK220Read48	(USHORT Axis, USHORT Latch, double* pData)
Eingespeicherten Zählerwert lesen (32 Bit)	BOOL IK220Get32	(USHORT Axis, USHORT Latch, LONG* pData)
Eingespeicherten Zählerwert lesen (48 Bit)	BOOL IK220Get48	(USHORT Axis, USHORT Latch, double* pData)
Messwert-Status lesen	BOOL IK220CntStatus	(USHORT Axis, USHORT Latch, USHORT* pRefSta, SHORT* pKorr00, SHORT* pKorr90, SHORT* pNKorr00, SHORT* pNKorr90, USHORT* pSamCnt)
Referenzpunkt-Fahren starten	BOOL IK220DoRef	(USHORT Axis)
Referenzpunkt-Fahren abbrechen	BOOL IK220CancelRef	(USHORT Axis)
Abfrage ob REF-Funktion aktiv	BOOL IK220RefActive	(USHORT Axis, BOOL* pStatus)
Warte bis REF-Funktion beendet	BOOL IK220WaitRef	(USHORT Axis)

Funktion	Kurzreferenz
Position der Referenzmarke ermitteln	BOOL IK220PositionRef (USHORT Axis, double* pData, LONG* pPeriod, USHORT* pIntpol)
Position der steigenden und fallenden Flanke der Referenzmarke ermitteln	BOOL IK220PositionRef2 (USHORT Axis, double*pData, LONG* pPeriod, USHORT* pIntpol)
Status IK 220 lesen	BOOL IK220Status (USHORT Axis, ULONG* pStatus)
Status der DLL-Funktionen lesen	BOOL IK220DllStatus (ULONG* pDLLStatus, ULONG* pDLLInfo)
REF-Status lesen	BOOL IK220RefStatus (USHORT Axis, LONG* pRef1, LONG* pRef2, LONG* pDiff, LONG* pCode, USHORT* pFlag)
Signal-Status lesen	BOOL IK220SignalStatus (USHORT Axis, USHORT* Freq, USHORT* pAmin, USHORT* pAact, USHORT* pAmax)
Angeglichene Korrekturwerte lesen	BOOL IK220GetCorrA (USHORT Axis,SHORT* pOfs0, SHORT* pOfs90, SHORT* Pha0, SHORT* pPha90,SHORT* Sym0, SHORT* pSym90, USHORT* pFlag1, USHORT* pFlag2)
Berechnete Korrekturwerte lesen	BOOL IK220GetCorrB (USHORT Axis,SHORT* pOfs0, SHORT* pOfs90,SHORT* Pha0, SHORT* Pha90,SHORT*pSym0, SHORT* pSym90, USHORT* pFlag1, USHORT* pFlag2)
Korrekturwerte laden	BOOL IK220LoadCorrA (USHORT Axis,SHORT Ofs0, SHORT Ofs90,SHORT Pha0, SHORT Pha90,SHORT Sym0, SHORT Sym90)
Oktanten-Status lesen	BOOL IK220OctStatus (USHORT Axis, USHORT* pOct0, USHORT* pOct1, USHORT* pOct2, USHORT* pOct3, USHORT* pOct4, USHORT* pOct5, USHORT* pOct6, USHORT* pOct7, USHORT* pSamCnt)

Funktion	Kurzreferenz
Prüfsumme der Parameter lesen	BOOL IK220ChkSumPar (USHORT Axis, USHORT* pChkSum)
Prüfsumme der Firmware lesen	BOOL IK220ChkSumPrg (USHORT Axis, USHORT* pChkSum1, USHORT* pChkSum2)
Parameter schreiben	BOOL IK220WritePar (USHORT Axis, USHORT ParNum, ULONG ParVal)
Parameter lesen	BOOL IK220ReadPar (USHORT Axis, USHORT ParNum, ULONG* pParVal)
Reset EnDat-Messgerät	BOOL IK220ResetEn (USHORT Axis, USHORT* pStatus)
Konfiguration des EnDat-Geber lesen	BOOL IK220ConfigEn (USHORT Axis, USHORT* pStatus, USHORT* pType, ULONG* pPeriod, ULONG* pStep, USHORT* pTurns, USHORT* pRefDist, USHORT* pCntDir)
Absoluten Zählerwert des EnDat-Geber lesen	BOOL IK220ReadEn (USHORT Axis, USHORT* pStatus, double* pData, USHORT* pAlarm)
Absoluten und inkrementalen Zählerwert des EnDat-Geber lesen	BOOL IK220ReadEnInc (USHORT Axis, USHORT Latch, USHORT* pStatus, double* pDataEn, USHORT* pAlarm, double* pDataInc)
Modus für durchlaufenden EnDat-Takt festlegen	BOOL IK220ModeEnCont (USHORT Axis, USHORT* Latch, USHORT Mode, USHORT* pStatus)
Absoluten und inkrementalen Zählerwert des mit EnDat-Geber durchlaufendem Takt lesen	BOOL IK220ReadEnIncCont (USHORT Axis, USHORT* pStatus, double* pDataEn, USHORT* pAlarm, double* pDataInc, USHORT* pSigStat)
Alarmwort EnDat-Geber lesen	BOOL IK220AlarmEn (USHORT Axis, USHORT* pAlarm)
Warnungswort EnDat-Geber lesen	BOOL IK220WarnEn (USHORT Axis, USHORT* pWarn)

Funktion	Kurzreferenz
Wert aus Speicherbereich des EnDat-Gebers lesen	BOOL IK220ReadMemEn(USHORT Axis, USHORT Range, USHORT MemAdr, USHORT* pMemData, USHORT* pStatus)
Wert in Speicherbereich des EnDat-Gebers schreiben	BOOL IK220WriteMemEn(USHORT Axis, USHORT Range, USHORT MemAdr, USHORT MemData, USHORT* pStatus)
Absoluten Zählerwert des SSI-Gebers lesen	BOOL IK220ReadSSI (USHORT Axis, USHORT* pStatus, double* pData)
Absoluten und inkrementalen Zählerwert des SSI-Gebers lesen	BOOL IK220ReadSsiInc (USHORT Axis, USHORT Latch, USHORT* pStatus, double* pDataSsi, double* pDataInc)
Wert für Timer festlegen	BOOL IK220SetTimer (USHORT Axis, ULONG SetVal, ULONG* pTimVal)
Modus für Timer festlegen	BOOL IK220ModeTimer (USHORT Axis, USHORT Mode)
Modus für RAM-Buffer festlegen	BOOL IK220ModeRam (USHORT Axis, USHORT Mode)
RAM-Buffer löschen	BOOL IK220ResetRam (USHORT Axis)
Zählerwert aus RAM-Buffer lesen	BOOL IK220GetRam (USHORT Axis, double* pData, USHORT* pRead, USHORT* pWrite, USHORT* pStatus)
Zählerwertblock aus RAM-Buffer lesen	BOOL IK220BurstRam (USHORT Axis, USHORT maxCount, double* pData, USHORT* pCount, USHORT* pStatus)
Amplitudenwerte aus RAM-Buffer lesen	BOOL IK220GetSig (USHORT Axis, USHORT* pPeriod, SHORT* pAmp0, SHORT* pAmp90, USHORT* pRead, USHORT* pWrite, USHORT* pStatus)

Funktion	Kurzreferenz	
Block von Amplituden-werten aus RAM-Buffer lesen	BOOL IK220BurstSig	(USHORT Axis, USHORT maxCount, USHORT* pPeriod, SHORT* pAmp0, SHORT* pAmp90, USHORT* pCount, USHORT* pStatus)
LED-Ansteuerung der Achsen	BOOL IK220Led	(USHORT Axis, USHORT Mode)
LED-Ansteuerung der Karte	BOOL IK220SysLed	(USHORT Card, USHORT Mode)
Externe Eingänge lesen	BOOL IK220GetPort	(USHORT Axis, USHORT* pPortInfo, USHORT* pRising, USHORT* pFalling)
Auswertung des Referenzmarken-Signals	IK220RefEval	(USHORT Axis, USHORT Mode)
Eingangsfrequenz für sinusförmige Inkremental-Signale	IK220SetBw	(USHORT Axis, USHORT Mode)

Folgende Funktionen werden von der Treiber-Software benutzt. In Anwendungs-Programmen sollten sie nicht verwendet werden.

Funktion	Kurzreferenz	
IK 220-Register lesen (16 Bit)	BOOL IK220InputW	(USHORT Axis, USHORT Adr, USHORT* pData)
IK 220-Register lesen (32 Bit)	BOOL IK220InputL	(USHORT Axis, USHORT Adr, ULONG* pData)
IK 220-Register schreiben (16 Bit)	BOOL IK220Output	(USHORT Axis, USHORT Adr, USHORT Data)
Wert aus RAM der IK 220 lesen	BOOL IK220RamRead	(USHORT Axis, USHORT Adr, USHORT* pData)
Wert ins RAM der IK 220 schreiben	BOOL IK220RamWrite	(USHORT Axis, USHORT Adr, USHORT Data)
Firmware in die IK 220 laden	BOOL IK220DownLoad	(USHORT Axis, USHORT* pPgmData, ULONG PgmSize)
EnDat Clock-Leitung setzen	BOOL IK220SetEnClock	(USHORT Axis, BOOL State, USHORT* pStatus)
EnDat Daten-Leitung setzen	BOOL IK220SetEnData	(USHORT Axis, BOOL State, USHORT* pStatus)
EnDat Daten-Leitung lesen	BOOL IK220ReadEnData	(USHORT Axis, BOOL State)

Referenz der DLL-Funktionen

Alle DLL-Funktionen liefern eine Boolsche Variable zurück. Ist diese Variable „Wahr" (=TRUE entspricht <>0) so war die Funktion erfolgreich. Enthält sie den Wert „Falsch" (=FALSE entspricht =0) so ist ein Fehler aufgetreten. Eingangswerte in die Funktionen werden als Zahlenwerte (by value) übergeben. Hat die Funktion Rückgabewerte, so wird die Adresse des Rückgabewertes (by reference) an die Funktion übergeben (Pointer auf Rückgabewert).

Es werden folgende Datentypen verwendet:

USHORT	:	unsigned 16 Bit
USHORT*	:	Pointer auf USHORT
SHORT	:	signed 16 Bit
SHORT*	:	Pointer auf SHORT
ULONG	:	unsigned 32 Bit
ULONG*	:	Pointer auf ULONG
LONG	:	signed 32 Bit
LONG*	:	Pointer auf LONG
double	:	Fließkomma 64 Bit
double*	:	Pointer auf Double
BOOL	:	Boolsche Variable 32 Bit
BOOL*	:	Pointer auf BOOL
char	:	Pointer auf String (mit 0x00 abgeschlossen)

IK220Find

Liefert die Adressen jeder Achse der installierten IK 220. Kann verwendet werden um festzustellen wie viele IK 220 installiert sind. Für jede IK 220 werden zwei Adressen an der entsprechenden Position in pBuffer16 abgelegt. Die unbenutzten Einträge werden auf 0 gesetzt. Für jede Achse muss anschließend IK220Init aufgerufen werden, um die Firmware zu laden und zu starten!

Prototyp: BOOL IK220Find (ULONG* pBuffer16);

pBuffer16: Zeiger auf 16 Langworte (16*4 Byte)

IK220Init

Lädt die Firmware in die IK 220 und startet sie. **Muss für jede Achse aufgerufen werden bevor weitere Funktionen benutzt werden können!**

Prototyp: BOOL IK220Init (USHORT Axis);

Axis: Nummer der Achse (0 bis 15)

IK220Version
Liest die Programm-Versionen der IK 220, des NT Device Treibers und der DLL. Die Programm-Versionen werden als ASCII-Zeichen abgelegt. Es muss jeweils Platz für mindestens 20 Zeichen reserviert werden. Die Zeichenketten werden mit einem Null-Byte abgeschlossen.

Prototyp: **BOOL IK220Version (USHORT Axis, char* pVersCard, char* pVersDrv, char* pVersDll)**

Axis: Nummer der Achse (0 bis 15)
pVersCard: Zeiger auf die Programm-Version der IK 220 Firmware
pVersDrv: Zeiger auf die Programm-Version des Windows NT Device Treibers (nur unter Windows NT)
pVersDll: Zeiger auf die Programm-Version der DLL

IK220Reset
Der Zähler wird auf Null gesetzt.

Prototyp: **BOOL IK220Reset (USHORT Axis);**
Axis: Nummer der Achse (0 bis 15)

IK220Start
Startet den Zähler.

Prototyp: **BOOL IK220Start (USHORT Axis);**
Axis: Nummer der Achse (0 bis 15)

IK220Stop
Stoppt den Zähler.

Prototyp: **BOOL IK220Stop (USHORT Axis);**
Axis: Nummer der Achse (0 bis 15)

IK220ClearErr
Löscht Amplituden- und Frequenz-Fehlerstatus.

Prototyp: **BOOL IK220ClearErr (USHORT Axis);**
Axis: Nummer der Achse (0 bis 15)

IK220Latch
Speichert den Zählerwert im angegebenen Register.

Prototyp: **BOOL IK220Latch (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
2=Zählerwert wird in Register 2 gespeichert (ohne Interpolation)

IK220LatchInt
Erzeugt ein Signal mit dem die Zählerwerte beider Achsen einer IK 220 synchron in Latch 0 gespeichert werden. Muss zuerst über Parameter 14 freigegeben werden.
Prototyp: BOOL IK220LatchInt (USHORT Card);
Card: Nummer der Karte (0 bis 7)

IK220LatchExt
Erzeugt einen Signal mit dem die Zählerwerte mehrerer IK 220 über eine externe Verbindung synchron in Latch 0/1 gespeichert werden. Muss zuerst über Parameter 14 freigegeben werden.
Prototyp: BOOL IK220LatchExt (USHORT Card);
Card: Nummer der Karte (0 bis 7)

IK220ResetRef
Der Zähler wird mit der nächsten Referenzmarke auf Null gesetzt.
Prototyp: BOOL IK220ResetRef (USHORT Axis);
Axis: Nummer der Achse (0 bis 15)

IK220StartRef
Der Zähler wird mit der nächsten Referenzmarke gestartet.
Prototyp: BOOL IK220StartRef (USHORT Axis);
Axis: Nummer der Achse (0 bis 15)

IK220StopRef
Der Zähler wird mit der nächsten Referenzmarke gestoppt.
Prototyp: BOOL IK220StopRef (USHORT Axis);
Axis: Nummer der Achse (0 bis 15)

IK220LatchRef
Mit der nächsten Referenzmarke wird der Zählerwert im Register 2 gespeichert. Der gespeicherte Wert ist ohne Interpolation und kann mit IKGet32 bzw. IKGet48 ausgelesen werden.

Prototyp: **BOOL IK220LatchRef (USHORT Axis);**
Axis: Nummer der Achse (0 bis 15)

IK220Latched
Stellt fest ob der Zählerwert gespeichert wurde.

Prototyp: **BOOL IK220Latched (USHORT Axis, USHORT Latch, BOOL* pStatus);**
Axis: Nummer der Achse (0 bis 15)
Latch: 0 = Abfrage für Register 0
1 = Abfrage für Register 1
2 = Abfrage für Register 2
pStatus: Zeiger auf Variable in welcher der Status abgelegt wird.
False (=0) = Wert nicht gespeichert
True (≠0) = Wert gespeichert

IK220WaitLatch
Wartet bis der Zählerwert gespeichert wurde. Wenn keine Timeout-Zeit definiert wurde wartet die Funktion bis ein Zählerwert im entsprechenden Latch gespeichert wurde.

Prototyp: **BOOL IK220WaitLatch (USHORT Axis, USHORT Latch);**
Axis: Nummer der Achse (0 bis 15)
Latch: 0 = Abfrage für Register 0
1 = Abfrage für Register 1
2 = Abfrage für Register 2

IK220SetTimeOut
Mit dieser Funktion kann eine Timeout-Zeit definiert werden. Wird keine Timeout-Zeit definiert so warten die Funktionen IK220WaitLatch, IK220WaitRef und IK220PositionRef bis das jeweilige Ereignis eintrifft.

Prototyp: **BOOL IK220SetTimeOut (ULONG TimeOut);**
TimeOut: 0 = kein Timeout
1.. = Timeout in ms

IK220Set
Setzt den Positionswert auf den angegebenen Wert. Benutzt Register 0 um die aktuelle Position zu ermitteln und berechnet daraus den Presetwert. Die Funktionen IK220Read48, IK220Get48, IK220ReadEnInc, IK220ReadEnIncCont, IK220ReadSsiInc, IK220GetRam und IK220BurstRam liefern dann inkrementale Positionswerte die sich auf den Presetwert beziehen (siehe IK220SetPreset und IK220GetPreset).

Prototyp: **BOOL IK220Set (USHORT Axis, double SetVal);**
Axis: Nummer der Achse (0 bis 15)
SetVal: Neuer Positionswert

IK220SetPreset
Setzt den Presetwert auf den angegebenen Wert. Der gesetzte Presetwert wird beim Aufruf von IK220Set immer zum Istwert der Achse addiert.

Prototyp: **BOOL IK220SetPreset (USHORT Axis, double PresVal);**
Axis: Nummer der Achse (0 bis 15)
PresVal: Neuer Presetwert

IK220GetPreset
Liefert den Presetwert der angegebenen Achse.

Prototyp: **BOOL IK220GetPreset (USHORT Axis, double* pPresVal);**
Axis: Nummer der Achse (0 bis 15)
pPresVal: Zeiger auf Variable in welcher der Presetwert abgelegt wird.

IK220Read32
Liefert den 32-Bit-Zählerwert.

Prototyp: **BOOL IK220Read32 (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 Variable in welcher der Positionswert abgelegt wird.

IK220Read48

Liefert den 48-Bit-Zählerwert.

Prototyp: **BOOL IK220Read48 (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 Variable in welcher der Zählerwert abgelegt wird.

IK220Get32

Liefert den 32-Bit-Zählerwert (20 Bit Vorkomma, 12 Bit Nachkomma (Interpolationswerte)). Bevor der Zählerwert ausgelesen werden kann muss er in Register 0, Register 1 oder Register 2 gespeichert werden (externes Signal, IKLatchInt, IK220LatchExt, Timer, IK220Latch oder IK220LatchRef) und dann eventuell abgefragt werden ob der Zählerwert schon gespeichert ist (IKLatched, IKWaitLatch).

Prototyp: **BOOL IK220Get32 (USHORT Axis, USHORT Latch, LONG* pData);**

Axis: Nummer der Achse (0 bis 15)

Latch: 0 = Auslesen aus Register 0
1 = Auslesen aus Register 1
2 = Auslesen aus Register 2

pData: Zeiger auf Variable in welcher der Zählerwert abgelegt wird.

IK220Get48

Liefert den 48-Bit-Zählerwert. Bevor der Zählerwert ausgelesen werden kann muss er in Register 0, Register 1 oder Register 2 gespeichert werden (externes Signal, IK220LatchInt, IK220LatchExt, Timer, IK220Latch oder IK220LatchRef) und dann eventuell abgefragt werden ob der Zählerwert schon gespeichert ist (IKLatched, IKWaitLatch).

Prototyp: **BOOL IK220Get48 (USHORT Axis, USHORT Latch, double* pData);**

Axis: Nummer der Achse (0 bis 15)

Latch: 0 = Auslesen aus Register 0
1 = Auslesen aus Register 1
2 = Auslesen aus Register 2

pData: Zeiger auf Variable in welcher der Zählerwert abgelegt wird (Floatwert: Nachkomma = Interpolationswert).

IK220CntStatus
Liefert zusätzliche Informationen über den letzten Zählerwert-Abruf des entsprechenden Registers.

Prototyp: **BOOL IK220CntStatus (USHORT Axis, USHORT Latch, USHORT* pRefSta, SHORT* pKorr00, SHORT* pKorr90, SHORT* pNKorr00, SHORT* pNKorr90, USHORT* pSamCnt);**

Axis: Nummer der Achse (0 bis 15)
Latch: 0 = Zähler-Register 0 auslesen
1 = Zähler-Register 1 auslesen
pRefSta: Zeiger auf Variable in welcher der REF-Status abgelegt wird.
pKorr00: Zeiger auf Variable in welcher der korrigierte 0°-Analogwert abgelegt wird.
pKorr90: Zeiger auf Variable in welcher der korrigierte 90°-Analogwert abgelegt wird.
pNKorr00: Zeiger auf Variable in welcher der nicht korrigierte 0°- Analogwert abgelegt wird.
pNKorr90: Zeiger auf Variable in welcher der nicht korrigierte 90°- Analogwert abgelegt wird.
pSamCnt: Zeiger auf Variable in welcher die aktuelle Anzahl der Messpunkte zur Korrekturwert-Ermittlung abgelegt wird.

IK220DoRef
Startet das Referenzpunkt-Fahren. Die REF-Marken werden wie in Parameter 6 festgelegt ausgewertet.

Prototyp: **BOOL IK220DoRef (USHORT Axis);**
Axis: Nummer der Achse (0 bis 15)

IK220CancelRef
Das Referenzpunkt-Fahren wird abgebrochen.

Prototyp: **BOOL IK220CancelRef (USHORT Axis);**
Axis: Nummer der Achse (0 bis 15)

IK220RefActive
Stellt fest ob eine REF-Funktion läuft (Reset, Start oder Stop mit REF bzw. REF-Fahren).

Prototyp: **BOOL IK220RefActive (USHORT Axis, BOOL* pStatus);**

Axis: Nummer der Achse (0 bis 15)
pStatus: Zeiger auf Variable in welcher der Status abgelegt wird.
False (=0) = REF-Funktion nicht aktiv
True (≠0) = REF-Funktion aktiv

IK220WaitRef
Wartet bis alle aktiven REF-Funktionen beendet sind (Reset, Start oder Stopp mit REF bzw. REF-Fahren). Wenn keine Timeout-Zeit definiert wurde wartet die Funktion bis die REF-Funktion beendet ist.

Prototyp: **BOOL IK220WaitRef (USHORT Axis);**
Axis: Nummer der Achse (0 bis 15)

IK220PositionRef
Wartet auf eine aktive Flanke des Referenzimpulses und führt dann einen Einspeicher-Befehl aus. Der eingespeicherte Wert entspricht der Position der Referenzmarke. Wenn keine Timeout-Zeit definiert wurde, wartet die Funktion bis ein Referenzimpuls erkannt wird. Ist der Referenzimpuls beim Aufruf der Funktion bereits aktiv, wird „FALSE" zurückgeliefert.

Prototyp: **BOOL IK220PositionRef (USHORT Axis, double* pData, LONG* pPeriod, USHORT* pIntpol)**
Axis: Nummer der Achse (0 bis 15)
pData: Zeiger auf Variable in welcher der Zählerwert abgelegt wird.
pPeriod: Zeiger auf Variable in welcher der Signalperiodenwert abgelegt wird.
pIntpol: Zeiger auf Variable in welcher der Interpolationswert abgelegt wird.

IK220PositionRef2
Wartet auf eine aktive Flanke des Referenzimpulses und speichert dann den aktuellen Positionswert. Anschließend wird auf die fallende Flanke des Referenzimpuls gewartet und ebenfalls der Positionswert gespeichert. Die gespeicherten Werte entsprechen der Position der steigenden und fallenden Flanke der Referenzmarke (siehe Technische Daten IK 220). Wenn keine Timeout-Zeit definiert wurde wartet die Funktion bis ein Referenzimpuls erkannt wird (siehe IK220SetTimeOut). Ist der Referenzimpuls beim Aufruf der Funktion bereits aktiv oder ist die Timeout-Zeit abgelaufen bevor eine Referenzmarke erkannt wurde wird „FALSE" zurück geliefert. Nach einem Timeout **muss** die Achse komplett neu initialisiert werden!

Prototyp: **BOOL IK220PositionRef2 (USHORT Axis, double* pData, LONG* pPeriod, USHORT* pIntpol)**
Axis: Nummer der Achse (0 bis 15)
pData1: Zeiger auf Variable in welcher der Positionswert der steigenden Flanke abgelegt wird.

pPeriod1: Zeiger auf Variable in welcher der Signalperiodenwert der steigenden Flanke abgelegt wird.
pIntpol1: Zeiger auf Variable in welcher der Interpolationswert der steigenden Flanke abgelegt wird.
pData2: Zeiger auf Variable in welcher der Positionswert der fallenden Flanke abgelegt wird.
pPeriod2: Zeiger auf Variable in welcher der Signalperiodenwert der fallenden Flanke abgelegt wird.
pIntpol2: Zeiger auf Variable in welcher der Interplationswert der fallenden Flanke abgelegt wird.

IK220Status
Liefert den Status der IK 220 zurück.
Prototyp: BOOL IK220Status (USHORT Axis, ULONG* pData);
Axis: Nummer der Achse (0 bis 15)
pData: Zeiger auf Variable in welcher der Status abgelegt wird.

Bitnummer	Bedeutung
0	1 = Zählerwert in Register 0 gespeichert
1	1 = Zählerwert in Register 1 gespeichert
2	1 = Zählerwert in Register 2 gespeichert
3	1 = EnDat-Abruf mit durchlaufendem Takt aktiv
4	1 = Verschmutzungswarnung/Fehler
5	1 = Zähler gestartet
6	1 = REF-Funktion aktiv (Start, Stopp, Reset oder Latch)
7	1 = Frequenzüberschreitung
8	1 = Korrekturwerte einmal berechnet
9	1 = Korrekturrechnung aktiv
10	1 = Speichern in RAM-Buffer läuft
11	
12	
13	
14 und 15	REF-Status: 0 = nicht REF gefahren 1 = warte auf 1. Referenzmarke 2 = warte auf 2. Referenzmarke 3 = REF-Fahren beendet
16 bis 31	

IK220DllStatus

Liefert den Status der DLL-Funktionen zurück.

Prototyp: **BOOL IK220DllStatus (ULONG* pDLLStatus, ULONG* pDLLInfo);**

Axis: Nummer der Achse (0 bis 15)

pDLLStatus: Zeiger auf Variable in welcher der Status der DLL-Funktionen abgelegt wird.

pDLLInfo: Zeiger auf Variable in welcher interne Status-Informationen abgelegt werden.

Der DLL-Status hat folgende Bedeutung:

Bitnummer	Bedeutung
0	Fehlermeldung von IK 220
1	Timeout-Fehler in DLL-Funktion
2	Falsche Befehls-Rückmeldung von IK 220
3	
4 bis 7	
8 bis 11	
12 bis 15	
16 bis 19	
20	Bereichsüberschreitung
21	REF-Signal ist schon aktiv
22	Timer-Nummer zu hoch
23	Fehler beim Device Treiber Aufruf
24	Falsche Datengröße beim Device Treiber Aufruf
25	Falscher Modus
26	Alarm von EnDat-Messgerät
27	Achs-Nummer zu hoch
28	Achse nicht installiert
29	Adresse zu hoch
30	Latch-Nummer zu hoch
31	Ungültige Speicheradresse

Die DLL-Info hat folgende Bedeutung:

Bitnummer	Bedeutung
0	Windows Timer nicht verfügbar
1	Windows Timer wird nicht verwendet
2	
3	
4 bis 7	
8 bis 11	
12 bis 15	
16 bis 19	
20 bis 23	
24 bis 27	
28 bis 31	

IK220RefStatus

Liefert den detaillierten REF-Status der IK 220 zurück.

Prototyp: **BOOL IK220RefStatus (USHORT Axis, LONG* pRef1, LONG* pRef2, LONG* pDiff, Long* pCode, USHORT* pFlag);**

Axis: Nummer der Achse (0 bis 15)

pRef1: Zeiger auf Variable in welcher die Position der 1. Referenzmarke abgelegt wird.

pRef2: Zeiger auf Variable in welcher bei abstandskodierten Messgeräten die Position der 2. Referenzmarke abgelegt wird.

pDiff: Zeiger auf Variable in welcher bei abstandskodierten Messgeräten die berechnete Differenz abgelegt wird.

pCode: Zeiger auf Variable in welcher bei abstandskodierten Messgeräten der berechnete Offset abgelegt wird.

pFlag: Zeiger auf Variable in welcher der REF-Status abgelegt wird.
0 = nicht REF gefahren
1 = warte auf 1. Referenzmarke
2 = warte auf 2. Referenzmarke
3 = REF-Fahren beendet

IK220SignalStatus

Liefert den Signal-Status der IK 220 zurück.

Prototyp: **BOOL IK220SignalStatus (USHORT Axis, USHORT* pFreq, USHORT* pAmin, USHORT* pAact, USHORT* pAmax);**

Axis: Nummer der Achse (0 bis 15)

pFreq: Zeiger auf Variable in welcher der Status der Frequenzüberschreitung abgelegt wird.
0 = o.k.
1 = Frequenzüberschreitung

pAmin: Zeiger auf Variable in welcher der Status der minimalen Amplitude abgelegt wird.

pAact: Zeiger auf Variable in welcher der Status der aktuellen Amplitude abgelegt wird.

pAmax: Zeiger auf Variable in welcher der Status der maximalen Amplitude abgelegt wird.

Status für Amplitude: 0 = Amplitude normal
1 = Amplitude klein
2 = Amplitude zu groß
3 = Amplitude zu klein

	11 µA$_{SS}$	**1 V$_{SS}$**	**Code**
Amplitude zu klein	2,5 µA$_{SS}$	0,22 V$_{SS}$	03
Amplitude klein	5 µA$_{SS}$	0,44 V$_{SS}$	01
Amplitude normal			00
Amplitude zu groß	16,25 µA$_{SS}$	1,40 V$_{SS}$	02

IK220GetCorrA

Liefert die angeglichenen Korrekturwerte der IK 220 zurück. Die Ermittlung von Korrekturwerten muss vorher über Parameter 9 freigegeben werden.

Prototyp: **BOOL IK220GetCorrA (USHORT Axis, SHORT* pOfs0, SHORT* pOfs90, SHORT* pPha0, SHORT* pPha90, SHORT* pSym0, SHORT* pSym90, USHORT* pFlag1, USHORT* pFlag2);**

Axis: Nummer der Achse (0 bis 15)

pOfs0: Zeiger auf Variable in welcher der Offset des 0°-Signals abgelegt wird.

pOfs90: Zeiger auf Variable in welcher der Offset des 90°-Signals abgelegt wird.

pPha0: Zeiger auf Variable in welcher die Phase des 0°-Signals abgelegt wird.

pPha90: Zeiger auf Variable in welcher die Phase des 90°-Signals abgelegt wird.

pSym0: Zeiger auf Variable in welcher die Symmetrie des 0°-Signals abgelegt wird.

pSym90: Zeiger auf Variable in welcher die Symmetrie des 90°-Signals abgelegt wird.
pFlag1: Zeiger auf Variable in welcher das Flag 1 abgelegt wird.
pFlag2: Zeiger auf Variable in welcher das Flag 2 abgelegt wird.

IK220GetCorrB

Liefert die berechneten Korrekturwerte der IK 220 zurück. Die Ermittlung von Korrekturwerten muss vorher über Parameter 9 freigegeben werden.

Prototyp: **BOOL IK220GetCorrB (USHORT Axis, SHORT* pOfs0, SHORT* pOfs90, SHORT* pPha0, SHORT* pPha90, SHORT* pSym0, SHORT* pSym90, USHORT* pFlag1, USHORT* pFlag2);**

Axis: Nummer der Achse (0 bis 15)
pOfs0: Zeiger auf Variable in welcher der Offset des 0°-Signals abgelegt wird.
pOfs90: Zeiger auf Variable in welcher der Offset des 90°-Signals abgelegt wird.
pPha0: Zeiger auf Variable in welcher die Phase des 0°-Signals abgelegt wird.
pPha90: Zeiger auf Variable in welcher die Phase des 90°-Signals abgelegt wird.
pSym0: Zeiger auf Variable in welcher die Symmetrie des 0°-Signals abgelegt wird.
pSym90: Zeiger auf Variable in welcher die Symmetrie des 90°-Signals abgelegt wird.
pFlag1: Zeiger auf Variable in welcher das Flag 1 abgelegt wird.
pFlag2: Zeiger auf Variable in welcher das Flag 2 abgelegt wird.

IK220LoadCorrA

Lädt die Korrekturwerte in die IK 220. Die Korrekturrechnung muss anschließend über Parameter 8 freigegeben werden.

Prototyp: **BOOL IK220LoadCorrA (USHORT Axis, SHORT Ofs0, SHORT Ofs90, SHORT Pha0, SHORT Pha90, SHORT Sym0, SHORT Sym90);**

Axis: Nummer der Achse (0 bis 15)
Ofs0: Korrekturwert für den Offset des 0°-Signals
Ofs90: Korrekturwert für den Offset des 90°-Signals
Pha0: Korrekturwert für die Phase des 0°-Signals
Pha90: Korrekturwert für die Phase des 90°-Signals
Sym0: Korrekturwert für die Symmetrie des 0°-Signals
Sym90: Korrekturwert für die Symmetrie des 90°Signal

IK220OctStatus

Liefert den Oktanten-Status der IK 220 zurück.

Prototyp: **BOOL IK220OctStatus (USHORT Axis, USHORT* pOct0, USHORT* pOct1, USHORT* pOct2, USHORT* pOct3, USHORT* pOct4, USHORT* pOct5, USHORT* pOct6, USHORT* pOct7, USHORT* pSamCnt);**

Axis: Nummer der Achse (0 bis 15)

pOct0: Zeiger auf Variable in welcher der Oktanten-counter 0 abgelegt wird.

pOct1: Zeiger auf Variable in welcher der Oktanten-counter 1 abgelegt wird.

pOct2: Zeiger auf Variable in welcher der Oktanten-counter 2 abgelegt wird.

pOct3: Zeiger auf Variable in welcher der Oktanten-counter 3 abgelegt wird.

pOct4: Zeiger auf Variable in welcher der Oktanten-counter 4 abgelegt wird.

pOct5: Zeiger auf Variable in welcher der Oktanten-counter 5 abgelegt wird.

pOct6: Zeiger auf Variable in welcher der Oktanten-counter 6 abgelegt wird.

pOct7: Zeiger auf Variable in welcher der Oktanten-counter 7 abgelegt wird.

pSamCnt: Zeiger auf Variable in welcher die aktuelle Anzahl der Messpunkte zur Korrekturwert-Ermittlung abgelegt wird.

IK220ChkSumPar

Liefert die aktuelle Prüfsumme der Parameter zurück.

Prototyp: **BOOL IK220ChkSumPar (USHORT Axis, USHORT* pChkSum);**

Axis: Nummer der Achse (0 bis 15)

pChkSum: Zeiger auf Variable in welcher die aktuelle Prüfsumme der Parameter abgelegt wird.

IK220ChkSumPrg

Liefert die Prüfsumme der IK 220 Firmware zurück.

Prototyp: **BOOL IK220ChkSumPrg (USHORT Axis, USHORT* pChkSum1, USHORT* pChkSum2);**

Axis: Nummer der Achse (0 bis 15)

pChkSum1: Zeiger auf Variable in welcher die Ist-Prüfsumme der Firmware abgelegt wird.

pChkSum2: Zeiger auf Variable in welcher die Soll-Prüfsumme der Firmware abgelegt wird.

IK220WritePar

Ändert einen Parameter der IK 220.

Prototyp: **BOOL IK220WritePar (USHORT Axis, USHORT ParNum, ULONG ParVal);**

Axis: Nummer der Achse (0 bis 15)
ParNum: Parameter-Nummer
ParVal: Parameter-Wert

IK220ReadPar

Liefert den Wert eines Parameter der IK 220.

Prototyp: **BOOL IK220ReadPar (USHORT Axis, USHORT ParNum, ULONG* pParVal);**

Axis: Nummer der Achse (0 bis 15)
ParNum: Parameter-Nummer
pParVal: Zeiger auf Variable in welcher der Parameter-Wert abgelegt wird.

IK220ResetEn

Setzt den angeschlossenen EnDat-Geber in den Grundzustand zurück.

Prototyp: **BOOL IK220ResetEn (USHORT Axis, USHORT* pStatus);**

Axis: Nummer der Achse (0 bis 15)
pStatus: Zeiger auf Variable in welcher der EnDat-Status abgelegt wird.
0 = o.k.
1 = Geber antwortet nicht bzw. nicht angeschlossen
2 = Übertragungsfehler
3 = Fehler Mode-Echo
4 = Fehler CRC-Summe
5 = Fehler Daten-Echo
6 = Fehler MRS-Code- / Adress-Echo

IK220ConfigEn

Liest die Konfiguration des angeschlossenen EnDat-Geber aus. Die genaue Bedeutung der einzelnen Werte ist der EnDat-Beschreibung zu entnehmen. Mit der Funktion IK220ReadMemEn können die Parameter des Messgerätherstellers ausgelesen werden um daraus weitere Informationen über das Messgerät zu erhalten. **Muss aufgerufen werden bevor weitere EnDat-Funktionen benutzt werden können!**

Prototyp: **BOOL IK220ConfigEn (USHORT Axis, USHORT* pStatus, USHORT* pType, ULONG* pPeriod, ULONG* pStep, USHORT* pTurns, USHORT* pRefDist, USHORT* pCntDir);**

Axis: Nummer der Achse (0 bis 15)

pStatus: Zeiger auf Variable in welcher der EnDat-Status abgelegt wird.

Low-Byte:
0 = o.k.
1 = Geber antwortet nicht bzw. kein Geber angeschlossen
2 = Übertragungsfehler
3 = Fehler Mode-Echo
4 = Fehler CRC-Summe
5 = Fehler Daten-Echo
6 = Fehler MRS-Code- / Adress-Echo

High-Byte:
0 = o.k.
1 = Fehler lesen Init-Parameter MRS-Code 0xA1
2 = Fehler lesen Bits pro Position
3 = Fehler lesen Gebertyp
4 = Fehler lesen LW Signalperiode
5 = Fehler lesen Init-Parameter MRS-Code 0xA3
6 = Fehler lesen HW Signalperiode
7 = Fehler lesen max. unterscheidbare Umdrehungen
8 = Fehler lesen Init Parameter MRS-Code 0xA5
9 = Fehler lesen unterstütze Alarme
10 = Fehler lesen unterstütze Warnungen
11 = Fehler lesen Grundabstand abstandskodiert REF
12 = Fehler lesen Messschritt LW
13 = Fehler lesen Messschritt HW
14 = Fehler lesen Messrichtung

pType: Zeiger auf Variable in welcher der Messgerättyp abgelegt wird.
pPeriod: Zeiger auf Variable in welcher die Signalperiode bzw. die Anzahl der Striche pro Umdrehung der inkremental Signale abgelegt wird.
pStep: Zeiger auf Variable in welcher der Messschritt bzw. die Anzahl der Messschritte pro Umdrehung des EnDat-Positionswertes abgelegt wird.
pTurns: Zeiger auf Variable in welcher die Anzahl unterscheidbarer Umdrehungen abgelegt wird.
pRefDist: Zeiger auf Variable in welcher der Grundabstand bei abstandskodierten REF-Marken abgelegt wird.
pCntDir: Zeiger auf Variable in welcher die Messrichtung abgelegt wird.

IK220ReadEn

Liefert den absoluten Zählerwert des angeschlossenen EnDat-Geber zurück. Der EnDat-Zählerwert hat die gleiche Wertigkeit wie der Inkremental-Wert, d.h. 1,0 entspricht einer Signalperiode!

Prototyp: **BOOL IK220ReadEn (USHORT Axis, USHORT* pStatus, double* pData, USHORT* pAlarm);**

Axis: Nummer der Achse (0 bis 15)
pStatus: Zeiger auf Variable in welcher der EnDat-Status abgelegt wird.
0 = o.k.
1 = Geber antwortet nicht bzw. kein Geber angeschlossen
2 = Fehler CRC-Summe
3 = Fehler Typ A
pData: Zeiger auf Variable in welcher der Zählerwert des EnDat-Geber abgelegt wird.
pAlarm: Zeiger auf Variable in welcher das Alarm-Bit abgelegt wird.
0 = o.k.
1 = Alarm ist aufgetreten

IK220ReadEnInc
Liefert den absoluten und inkrementalen Zählerwert des angeschlossenen EnDat-Geber zurück. Der EnDat-Zählerwert hat die gleiche Wertigkeit wie der Inkremental-Wert, d.h. 1,0 entspricht einer Signalperiode!

Prototyp: **BOOL IK220ReadEnInc (USHORT Axis, USHORT Latch, USHORT* pStatus, double* pDataEn, USHORT* pAlarm, double* pDataInc);**

Axis: Nummer der Achse (0 bis 15)
Latch: 0 = Auslesen Inkremental-Wert über Register 0
1 = Auslesen Inkremental-Wert über Register 1
pStatus: Zeiger auf eine Variable in welcher der EnDat-Status abgelegt wird.
0 = o.k.
1 = Geber antwortet nicht bzw. kein Geber angeschlossen
2 = Fehler CRC-Summe
3 = Fehler Typ A
pDataEn: Zeiger auf Variable in welcher der absolute Zählerwert des EnDat-Geber abgelegt wird.
pAlarm: Zeiger auf Variable in welcher das Alarm-Bit abgelegt wird.
0 = o.k.
1 = Alarm ist aufgetreten
pDataInc: Zeiger auf Variable in welcher der inkrementale Zählerwert des EnDat-Geber abgelegt wird.

IK220ModeEnCont
Startet und stoppt den durchlaufenden EnDat-Takt. Mit durchlaufendem EnDat-Takt werden ständig neue EnDat-Zählerwerte abgerufen und synchron dazu die Inkremental-Werte ermittelt. Die Zählerwerte können mit der Funktion IK220ReadEnIncCont ausgelesen werden. Andere Funktionen können in dieser Betriebsart nicht benutzt werden. Wird der durchlaufende EnDat-Takt ohne CRC-Prüfung gestartet wird nach der Datenübertragung die CRC-Summe nicht überprüft, dadurch wird die Abrufzeit verkürzt.

Prototyp: **BOOLIK220ModeEnCont (USHORT Axis, USHORT Latch, USHORT Mode, SHORT* pStatus)**

Axis: Nummer der Achse (0 bis 15)
Latch: 0 = Auslesen Inkremental-Wert über Register 0.

Mode: 0 = Auslesen mit durchlaufendem Takt beenden
1 = Auslesen mit durchlaufendem Takt mit CRC-Prüfung starten
2 = Auslesen mit durchlaufendem Takt ohne CRC-Prüfung starten

pStatus: 0 = o.k.
1 = Geber antwortet nicht bzw. kein Geber angeschlossen

IK220ReadEnIncCont

Liefert den absoluten und inkrementalen Positionswert des angeschlossenen EnDat-Messgerät zurück während die EnDat-Zählerwerte ständig mit durchlaufendem Takt ausgelesen werden. Bevor diese Funktion benutzt wird, muss mit IK220ModEnCont der durchlaufende EnDat-Takt gestartet werden. Der EnDat-Zählerwert hat die gleiche Wertigkeit wie der Inkremental-Wert, d.h. 1,0 entspricht einer Signalperiode.

Prototyp: BOOLIK220ReadEnIncCont (USHORT Axis, USHORT* pStatus, double* pDataEn, USHORT* pAlarm, double* pDataInc, USHORT* pSigStat)

Axis: Nummer der Achse (0 bis 15)

pStatus: Zeiger auf Variable in welcher der EnDat-Status abgelegt wird.
0: o.k.
Bit0=1: Geber antwortet nicht bzw. kein Geber angeschlossen.
Bit1=1: Fehler CRC-Summe
Bit2=1: Fehler Typ A

pDataEn: Zeiger auf Variable in welcher der absolute Zählerwert des EnDat-Messgerät abgelegt wird.

pAlarm: Zeiger auf Variable in welcher das Alarm-Bit abgelegt wird.
0 = o.k.
1 = Alarm ist aufgetreten

pDataInc: Zeiger auf Variable in welcher der inkrementale Zählerwert des EnDat-Messgerät abgelegt wird.

pSigStat: Zeiger auf Variable in welcher der Amplituden-Status der Inkremental Signale abgelegt wird.
Bit 0/1: Status minimale Amplitude
Bit 2/3: Status aktuelle Amplitude
Bit 4/5: Status maximale Amplitude
(Amplitude-Status: 00=normal / 01=klein / 10=groß / 11=zu klein)

IK220AlarmEn

Liefert das Alarmwort des EnDat-Geber und löscht alle aktiven Alarme.

Prototyp: **BOOL IK220AlarmEn (USHORT Axis, USHORT* pStatus, USHORT* pAlarm);**

Axis: Nummer der Achse (0 bis 15).

pStatus: Zeiger auf Variable in welcher der Status abgelegt wird.

Low-Byte: 0 = o.k.
1 = Geber antwortet nicht bzw. kein Geber angeschlossen
2 = Übertragungsfehler
3 = Fehler Mode-Echo
4 = Fehler CRC-Summe
5 = Fehler Daten-Echo
6 = Fehler MRS-Code- / Adress-Echo

High-Byte: 0 = o.k.
1 = Fehler lesen Init- Parameter MRS-Code 0xB9
2 = Fehler lesen/schreiben Alarmwort
3 = Fehler RESET auf Geber

pAlarm: Zeiger auf Variable in welcher das Alarmwort abgelegt wird.

IK220WarnEn

Liefert das Warnungswort des EnDat-Geber und löscht alle aktiven Warnungen.

Prototyp: **BOOL IK220WarnEn (USHORT Axis, USHORT* pStatus, USHORT* pWarn);**

Axis: Nummer der Achse (0 bis 15).

pStatus: Low-Byte: 0 = o.k.
1 = Geber antwortet nicht bzw. kein Geber angeschlossen
2 = Übertragungsfehler
3 = Fehler Mode-Echo
4 = Fehler CRC-Summe
5 = Fehler Daten-Echo
6 = Fehler MRS-Code- / Adress-Echo

High-Byte: 0 = o.k.
1 = Fehler lesen Init-Parameter MRS-Code 0xB9
2 = Fehler lesen/schreiben Warnungswort
3 = Fehler RESET auf Geber

pWarn: Zeiger auf Variable in welcher das Warnungswort abgelegt wird.

IK220ReadMemEn

Liest Werte aus dem Speicherbereich des EnDat-Gebers.

Prototyp: **BOOL IK220ReadMemEn (USHORT Axis, USHORT Range, USHORT MemAdr, USHORT* pMemData, USHORT* pStatus)**

Axis: Nummer der Achse (0 bis 15)

Range: Auswahl Speicherbereich
- 0: Betriebszustand
- 1: Parameter des Messgerät-Herstellers
- 2: Betriebsparameter
- 3: Parameter des OEM
- 4: Korrekturwerte

MemAdr: Wortadresse im gewählten Bereich.

pMemData: Zeiger auf Variable in welcher der Wert abgelegt wird.

pStatus: Zeiger auf Variable in welcher der EnDat-Status abgelegt wird.
- 0 = o.k.
- 1 = Geber antwortet nicht bzw. kein Geber angeschlossen
- 2 = Übertragungsfehler
- 3 = Fehler Mode-Echo
- 4 = Fehler CRC-Summe
- 5 = Fehler Daten-Echo
- 6 = Fehler MRS-Code- / Adress-Echo

IK220WriteMemEn

Schreibt Werte in den OEM-Parameterspeicher des EnDat-Gebers. Die Bedeutung der Werte wird vom OEM festgelegt.

Prototyp: **BOOL IK220WriteMemEn (USHORT Axis, USHORT Range, USHORT MemAdr, USHORT MemData, USHORT* pStatus)**

Axis: Nummer der Achse (0 bis 15)

Range: Auswahl Speicherbereich
- 0: Betriebszustand
- 1: Parameter des Messgerät-Herstellers
- 2: Betriebsparameter
- 3: Parameter des OEM
- 4: Korrekturwerte

MemAdr: Speicheradresse im gewählten Bereich

MemData: Wert der geschrieben wird

pStatus: Zeiger auf Variable in welcher der EnDat-Status abgelegt wird.
0 = o.k.
1 = Geber antwortet nicht bzw. kein Geber angeschlossen
2 = Übertragungsfehler
3 = Fehler Mode-Echo
4 = Fehler CRC-Summe
5 = Fehler Daten-Echo
6 = Fehler MRS-Code- / Adress-Echo

IK220ReadSSI
Liefert den absoluten Zählerwert des angeschlossenen SSI-Geber zurück. Die Übertragungsparameter des SSI-Geber müssen vorher in den Parametern festgelegt werden.

Prototyp: BOOL IK220ReadSSI (USHORT Axis, USHORT* pStatus, double* pData);

Axis: Nummer der Achse (0 bis 15)
pStatus: Zeiger auf Variable in welcher der SSI-Status abgelegt wird.
0 = o.k.
1 = Geber antwortet nicht bzw. kein Geber angeschlossen
2 = Parity-Error
3 = Parity-Error, Vornull-Bit nicht Null
pData: Zeiger auf Variable in welcher der Zählerwert des SSI-Geber abgelegt wird.

IK220ReadSsiInc
Liefert den absoluten und inkrementalen Zählerwert des angeschlossenen SSI-Geber zurück. Die Übertragungsparameter des SSI-Geber sind vorher in den Parametern festzulegen

Prototyp: BOOL IK220ReadSsiInc (USHORT Axis, USHORT Latch, USHORT* pStatus, double* pDataSsi, double* pDataInc)

Axis: Nummer der Achse (0 bis 15)
Latch: 0 = Auslesen Inkremental-Wert über Register 0
1 = Auslesen Inkremental-Wert über Register 1
pStatus: Zeiger auf Variable in welcher der SSI-Status abgelegt wird.
0 = o.k.
1 = Geber antwortet nicht bzw. kein Geber angeschlossen
2 = Parity-Error
3 = Parity-Error, Vornull-Bit nicht Null

pDataSsi: Zeiger auf Variable in welcher der absolute Zählerwert des SSI-Geber abgelegt wird.
pDataInc: Zeiger auf Variable in welcher der inkrementale Zählerwert des SSI-Geber abgelegt wird.

IK220SetTimer
Programmiert den Timer mit dem durch „SetVal" angegebenen oder nächst größeren möglichen Wert. Die Zeit wird durch den Parameter für das Zeitintervall des Timer und den Parameter für den Software-Teiler eingestellt.

Prototyp: BOOL IK220SetTimer (USHORT Axis, ULONG SetVal, ULONG* pTimVal);
Axis: Nummer der Achse (0 bis 15).
SetVal: Gewünschter Timerwert in Mikrosekunden.
pTimVal: Zeiger auf Variable in welcher der tatsächlich programmierte Timerwert in Mikrosekunden abgelegt wird.

IK220ModeTimer
Legt fest ob das Timer-Signal ausgegeben wird. Um die Achsen einer Karte bzw. mehrerer Karten über eine externe Verbindung speichern zu können muss das vom Timer erzeugte Signal ausgegeben werden. Die Periodendauer des Timer-Signal hängt nur vom Zeitintervall des Timer. Der im Software-Teiler programmierte Wert hat darauf keinen Einfluss.

Prototyp: BOOL IK220ModeTimer (USHORT Axis, USHORT Mode);
Axis: Nummer der Achse (0 bis 15).
Mode: 0 = Timer-Signale werden nicht ausgegeben
1 = Timer-Signale werden ausgegeben

IK220ModeRam
Gespeicherte Zählerwerte können in einem Speicher auf der IK 220 abgelegt werden. Die abgelegten Werte können dann mit den Funktionen IK220GetRam bzw. IK220BurstRam ausgelesen werden.

Prototyp: BOOL IK220ModeRam (USHORT Axis, USHORT Mode);
Axis: Nummer der Achse (0 bis 15)
Mode: 0 = Eingespeicherte Zählerwerte werden nicht abgelegt
1 = Eingespeicherte Zählerwerte aus Register 0 werden abgelegt[1)]
2 = Eingespeicherte Zählerwerte aus Register 1 werden abgelegt

[1)] Ringspeicherfunktion

3 = Eingespeicherte Zählerwerte aus Register 0 werden abgelegt bis maximale Anzahl erreicht ist (Single shot)
4 = Eingespeicherte Zählerwerte aus Register 1 werden abgelegt bis maximale Anzahl erreicht ist (Single shot)

IK220ResetRam

Der Schreib- und Lesezeiger des RAM-Buffers wird auf 0 gesetzt. Damit werden alle Werte im RAM-Buffer gelöscht.

Prototyp: **BOOL IK220ResetRam (USHORT Axis);**

Axis: Nummer der Achse (0 bis 15)

IK220GetRam

Ein vorher im RAM-Buffer abgelegter Zählerwert wird ausgelesen. Nach jedem Lesen wird der Offset des Lesezeigers erhöht bis alle Werte ausgegeben wurden.

Prototyp: **BOOL IK220GetRam (USHORT Axis, double* pData, USHORT* pRead, USHORT* pWrite, USHORT* pStatus);**

Axis: Nummer der Achse (0 bis 15)

pData: Zeiger auf Variable in welcher der Zählerwert abgelegt wird.

pRead: Zeiger auf Variable in welcher der Offset des Schreibzeigers im RAM-Buffer abgelegt wird.

pWrite: Zeiger auf Variable in welcher der Offset des Lesezeigers im RAM-Buffer abgelegt wird.

pStatus: Status des RAM-Buffers.
Bit0=1 : Buffer-Überlauf
Bit1=1 : kein Wert im Buffer
Bit2=1 : letzten Wert aus Buffer gelesen

IK220BurstRam

Vorher im RAM-Buffer abgelegte Zählerwerte werden ausgelesen. Der Lesezeiger wird anschließend um die Anzahl gelesener Werte erhöht.

Prototyp: **BOOL IK220BurstRam (USHORT Axis, USHORT maxCount, double* pData, USHORT* pCount, USHORT* pStatus)**

Axis: Nummer der Achse (0 bis 15)

maxCount: Maximale Anzahl der Werte die bei einem Aufruf gelesen werden.

pData: Zeiger auf ein Array von „Double-Variable“ (64 Bit) in welcher die Zählerwerte abgelegt werden. Es muss Platz für maxCount Zählerwerte reserviert werden!

pCount: Zeiger auf eine Variable in welcher die tatsächliche Anzahl gelesener Zählerwerte abgelegt wird.

pStatus: Status des RAM-Buffers.
Bit0=1 : Buffer-Überlauf
Bit1=1 : kein Wert im Buffer
Bit2=1 : letzten Wert aus Buffer gelesen
Bit15=1: Fehler beim Puffer auflesen

IK220GetSig

Ein im RAM-Buffer abgelegtes Amplituden-Wertepaar wird ausgelesen. Nach dem Lesen wird der Lesezeiger erhöht.

Prototyp: BOOL IK220GetSig (USHORT Axis, USHORT* pPeriod, SHORT* pAmp0, SHORT* pAmp90, USHORT* pRead, USHORT* pWrite, USHORT* pStatus)

Axis: Nummer der Achse (0 bis 15)

pPeriod: Zeiger auf Variable in welcher die unteren 16 Bit des Signalperiodenzählers abgelegt werden.

pAmp0: Zeiger auf Variable in welcher der 0°-Amplitudenwert abgelegt wird.

pAmp90: Zeiger auf Variable in welcher der 90°-Amplitudenwert abgelegt wird.

pRead: Zeiger auf eine Variable in welcher der Offset des Schreibzeigers im RAM-Buffer abgelegt wird.

pWrite: Zeiger auf Variable in welcher der Offset des Lesezeigers im RAM-Buffer abgelegt wird.

pStatus: Status des RAM-Buffers.
Bit0=1 : Buffer-Überlauf
Bit1=1 : kein Wert im Buffer
Bit2=1 : letzten Wert aus Buffer gelesen

IK220BurstSig

Vorher im RAM-Buffer abgelegte Amplituden-Wertepaare werden ausgelesen. Der Lesezeiger wird anschließend um die Anzahl gelesener Werte erhöht.

Prototyp: BOOL IK220BurstSig (USHORT Axis, USHORT maxCount, USHORT* pPeriod, SHORT* pAmp0, SHORT* pAmp90, USHORT* pCount, USHORT* pStatus)

Axis: Nummer der Achse (0 bis 15)

maxCount: Maximale Anzahl der Wertepaare die bei einem Aufruf gelesen werden.

pPeriod: Zeiger auf ein Array von Variablen, in welchem die Signalperiodenzähler-Werte abgelegt werden. Es muss Platz für maxCount Werte reserviert werden!

pAmp0: Zeiger auf ein Array von Variablen in welchem die 0°-Amplitudenwerte abgelegt werden. Es muss Platz für maxCount Werte reserviert werden!

pAmp90: Zeiger auf ein Array von Variablen in welchem die 90°-Amplitudenwerte abgelegt werden. Es muss Platz für maxCount Werte reserviert werden!
pCount: Zeiger auf eine Variable in welcher die tatsächliche Anzahl gelesener Wertepaare abgelegt wird.
pWrite: Zeiger auf Variable in welcher der Offset des Lesezeigers im RAM-Buffer abgelegt wird.
pStatus: Status des RAM-Buffers.
Bit0=1 : Buffer-Überlauf
Bit1=1 : kein Wert im Buffer
Bit2=1 : letzten Wert aus Buffer gelesen
Bit15=1: Fehler beim Buffer auslesen

IK220Led
Legt die Funktion der Achs-LED auf der IK 220 fest.
Prototyp: BOOL IK220Led (USHORT Axis, USHORT Mode);
Axis: Nummer der Achse (0 bis 15)
Mode: 0 = LED leuchtet nicht
1 = LED leuchtet
2 = LED blinkt

IK220SysLed
Legt die Funktion der System-LED auf der IK 220 fest.
Prototyp: BOOL IK220SysLed (USHORT Card, USHORT Mode);
Card: Nummer der Karte (0 bis 7)
Mode: 0 = LED leuchtet nicht
1 = LED leuchtet

IK220GetPort
Liefert den Zustand der Eingänge auf der IK 220 zurück.
Prototyp: BOOL IK220GetPort (USHORT Axis, USHORT* pPortInfo, USHORT* pRising, USHORT* pFalling);
Axis: Nummer der Achse (0 bis 15)
pPortInfo: Zeiger auf Variable in welcher der aktuelle Zustand der Eingänge abgelegt wird.
pRising: Zeiger auf Variable in welcher angezeigt wird ob „Set-Flanken" seit dem letzten Auslesen aufgetreten sind.
pFalling: Zeiger auf Variable in welcher angezeigt wird ob „Reset-Flanken" seit dem letzten Auslesen aufgetreten sind.

IK220RefEval
Legt die Art der Auswertung des Referenzmarken-Signals fest.

Prototyp: **BOOL IK220RefEval (USHORT Axis, USHORT Mode);**

Axis: Nummer der Achse
Mode: 0 = REF-Signal mit Inkremental-Signal maskiert
1 = REF-Signal nicht mit Inkremental-Signal maskiert.

IK220SetBw
Legt die Eingangsfrequenz für die sinusförmigen Inkremental-Signale fest. Wird über Parameter 2 von 11 μA_{SS} auf 1 V_{SS} umgeschaltet (oder umgekehrt), so ist die Standard-Einstellung wieder aktiv (bei 11 μA_{SS} 33 kHz, bei 1 V_{SS} 500 kHz). Wenn eine andere Eingangsfrequenz gewünscht ist, dann muss man diese über IK220SetBw setzen.

Prototyp: **BOOL IK220SetBw (USHORT Axis, USHORT Mode);**

Axis: Nummer der Achse
Mode: 0 = Hohe Eingangsfrequenz (bei 11 μA_{SS} 175 kHz, bei 1 V_{SS} 500 kHz)
1 = Niedrige Eingangsfrequenz (bei 11 μA_{SS} und bei 1 V_{SS} 33 kHz)

Folgende Funktionen werden von der Treiber-Software benutzt. In Anwendungs-Programmen sollten sie nicht verwendet werden.

IK220InputW
Liest einen 16-Bit-Wert von der angegebenen Adresse der Achse.

Prototyp: **BOOL IK220InputW (USHORT Axis, USHORT Adr, USHORT* pData)**

Axis: Nummer der Achse (0 bis 15)
Adr: Adresse des Registers (0 bis 15 bzw. 0 bis 0x0F)
pData: Zeiger auf Variable in welcher der gelesene Wert abgelegt wird.

IK220InputL
Liest einen 32-Bit-Wert von der angegebenen Adresse der Achse.

Prototyp: **BOOL IK220InputL (USHORT Axis, USHORT Adr, ULONG* pData)**

Axis: Nummer der Achse (0 bis 15)
Adr: Adresse des Registers (0 bis 14 bzw. 0 bis 0x0E)
pData: Zeiger auf Variable in welcher der gelesene Wert abgelegt wird.

IK220Output
Schreibt einen 16-Bit-Wert auf die angegebenen Adresse der Achse.

Prototyp: **BOOL IK220Output (USHORT Axis, USHORT Adr, USHORT Data)**

Axis: Nummer der Achse (0 bis 15)
Adr: Adresse des Registers (0 bis 15 bzw. 0 bis 0x0F)
Data: Wert der in das Register geschrieben wird

IK220RamRead
Aus dem RAM der IK 220 wird ein 16-Bit-Wert gelesen.

Prototyp: **BOOL IK220RamRead (USHORT Axis, USHORT Adr, USHORT* pData)**

Axis: Nummer der Achse (0 bis 15)
Adr: Adresse Im RAM (0 bis 65535)
pData: Zeiger auf Variable in welcher der Wert abgelegt wird.

IK220RamWrite
Ein 16-Bit-Wert wird in das RAM der IK 220 geschrieben.

Prototyp: **BOOL IK220RamWrite (USHORT Axis, USHORT Adr, USHORT Data)**

Axis: Nummer der Achse (0 bis 15)
Adr: Adresse im RAM (0 bis 65535)
Data: Wert (16 Bit) der ins RAM geschrieben wird

IK220DownLoad
Die Firmware der IK 220 wird in das RAM geladen.

Prototyp: **BOOL IK220DownLoad (USHORT Axis, USHORT* pPgmData, ULONG PgmSize)**

Axis: Nummer der Achse (0 bis 15)
pPgmData Zeiger auf ein Array in dem die Programm-Informationen abgelegt sind
PgmSize: Anzahl der Bytes im Array PgmData

IK220SetEnClock
Setzt die Clock-Leitung der EnDat-Schnittstelle.

Prototyp: **BOOL IK220SetEnClock (USHORT Axis, BOOL State, USHORT* pStatus)**

Axis: Nummer der Achse (0 bis 15)
State: False (=0): Clock-Leitung auf Low-Pegel setzen
True (≠0): Clock-Leitung auf High-Pegel setzen
pStatus: 0 = o.k.
1 = Fehler

IK220SetEnData

Setzt die Daten-Leitung der EnDat-Schnittstelle.

Prototyp: **BOOL IK220SetEnData (USHORT Axis, BOOL State, USHORT* pStatus)**

Axis: Nummer der Achse (0 bis 15)

State: False (=0): Daten-Leitung auf Low-Pegel setzen
True (≠0): Daten-Leitung auf High-Pegel setzen

pStatus: 0 = o.k.
1 = Fehler

IK220ReadEnData

Liest den Zustand der Datenleitung auf der EnDat-Schnittstelle.

Prototyp: **BOOL IK220ReadEnData (USHORT Axis, BOOL* pState)**

Axis: Nummer der Achse (0 bis 15)

pState: Zeiger auf Variable in welcher der Pegel der EnDat Daten-Leitung abgelegt wird.
False (=0): Daten-Leitung ist auf Low-Pegel
True (≠0): Daten-Leitung ist auf High-Pegel

Technische Daten

Mechanische Kennwerte	
Abmessungen	ca. 190 mm x 100 mm
Arbeitstemperatur **Lagertemperatur**	0° C bis 55° C –30° C bis 70° C

Elektrische Kennwerte	
Messgerät-Eingänge	zwei Sub-D-Anschlüsse (15-polig, Stift) Umschaltbar: • ∿ **11 µA$_{SS}$:** Sinusförmige Stromsignale Eingangsfrequenz: max. 33 kHz Kabellänge: max. 60 m[1)] • ∿ **1 V$_{SS}$:** Sinusförmige Spannungssignale Eingangsfrequenz: max. 500 kHz Kabellänge: max. 60 m[1)] • **EnDat 2.1** Kabellänge: max. 50 m[1)] • **SSI** Kabellänge: max. 10 m[1)]
Externe Abruf-Signale (Option) 2 Eingänge 1 Ausgang	Baugruppe mit zwei Sub-D-Anschlüsse (9-polig, Sift) TTL-Pegel TTL-Pegel
Messgerät-Ausgänge (Option)	sinusfömige Stromsignale (11 µA$_{SS}$) Baugruppe mit zwei Sub-D-Anschlüsse (9-polig, Stift)
Signal-Unterteilung	bis zu 4096fach

[1)] Mit Original-HEIDENHAIN-Kabeln, Versorgungsspannung des Messgerätes beachten!

Abgleich der Messgerät-Signale	Abgleich von Offset, Phase und Amplitude per Software – auch online
Datenregister für Messwerte	48 Bit, wobei für den Messwert 44 Bit genutzt werden
Schnittstelle	PCI-Bus (Plug and Play)
Interner Speicher	für 8191 Positionswerte
Leistungsaufnahme	ca. 4 W, ohne Messgerät

Software	
Treiber-Software und Demonstrations-Programm[2)]	für Windows NT/95/98/2000/XP Linux Kernel 2.4 LabView 7.1 in Visual C++, Visual Basic und Borland Delphi

2) Der CD-Inhalt sowie Treiber für weitere Betriebssysteme (z. B. Windows Vista, Windows 7) befinden sich im Download-Bereich unter www.heidenhain.de.

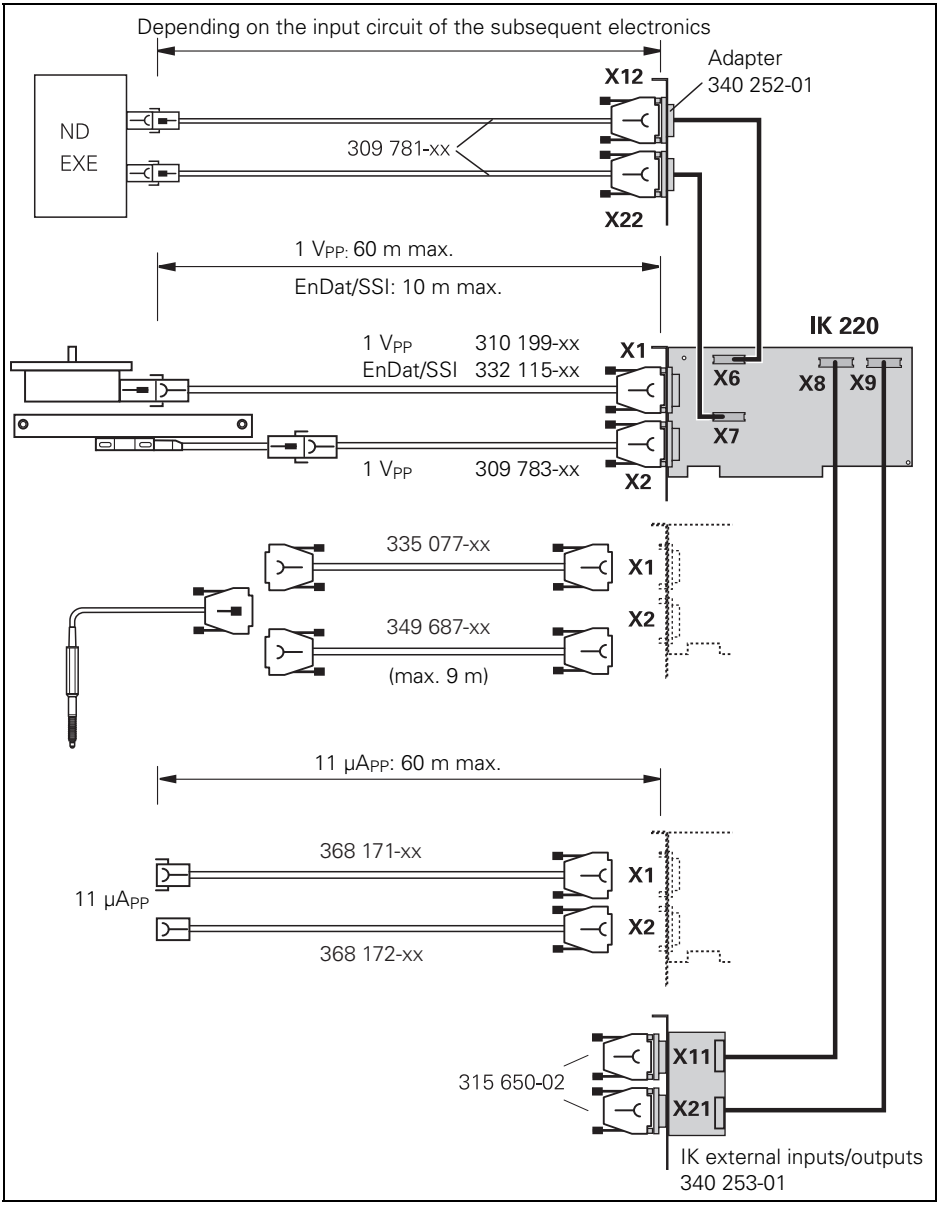
Content

Items Supplied

IK 220 Counter Card for PCs Programming examples, driver software and User's Manual.	Id. Nr. 337 481-01

Accessories

• IK external inputs/outputs	Id.-Nr. 340 253-01
• IK external inputs/outputs	Id.-Nr. 315 650-02
• Adapter cable with connector for HEIDENHAIN encoders	
- With sinusoidal voltage signals 1 V_{PP}	Id. Nr. 310 199-xx
- With sinusoidal current signals 11 μA_{PP}	Id. Nr. 368 171-xx
- With EnDat interface	Id. Nr. 332 115-XX
• Adapter cable with coupling for HEIDENHAIN encoders	
- With sinusoidal voltage signals 1 V_{PP}	Id. Nr. 309 783-xx
- With sinusoidal current signals 11 μA_{PP}	Id. Nr. 368 172-xx
• Additional D-sub connector for extending the encoder signals from inputs X1 and X2 to another display or control	Id. Nr. 340 252-01
• Connecting cable from the additional D-sub connection to another display or control	Id. Nr. 309 781-xx


Depending on the input circuit of the subsequent electronics
Adapter
340 252-01
X12
ND
EXE
309 781-xx
X22
1 VPP: 60 m max.
EnDat/SSI: 10 m max.
IK 220
1 VPP 310 199-xx
EnDat/SSI 332 115-xx
X1
X6
X8
X9
X7
1 VPP 309 783-xx
X2
335 077-xx
X1
349 687-xx
X2
(max. 9 m)
11 µAPP: 60 m max.
368 171-xx
11 µAPP
X1
X2
368 172-xx
X11
315 650-02
X21
IK external inputs/outputs
340 253-01

Important Information

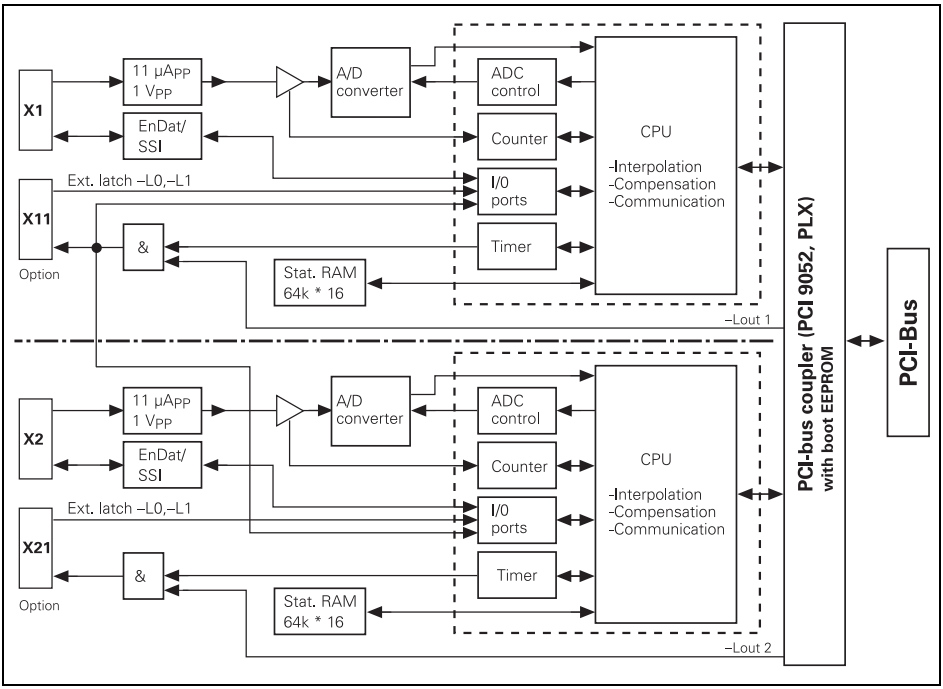
The EnDat interface offers the possibility of storing machine or system-dependent data in the customer memory area (e.g. datum shift, OEM data, ...). These data can contain safety-relevant information. Incorrect handling of these data can result in damage to the machine or personnel.
For information on the EnDat interface and the encoder, please refer to the EnDat specification, the mounting instructions of the encoder and the product description (e.g. catalog, product information). If you require more information, please get in touch with your contact person in the Sales department.

Danger to internal components!
When handling components that can be damaged by **electrostatic discharge (ESD),** follow the safety recommendations in EN 100 015. Use only antistatic packaging material. Be sure that the work station and the technician are properly grounded during installation.

Technical Description of the IK 220

The IK 220 counter card for PCs is plugged directly into an expansion slot of a personal computer with PCI interface. The card can support two HEIDENHAIN encoders with sinusoidal current signals (11 μA_{PP}), voltage signals (1 V_{PP}), EnDat or SSI interface. The positions of the two encoders are displayed on the PC, saved in the PC and further processed by the software. The IK 220 is ideal for applications requiring high resolution of encoder signals and fast measured value acquisition.

Block diagram of the IK 220



The IK 220's interpolation electronics subdivides the signal period of the input signal up to 4096-fold.
The 44-bit wide measured value is formed from the interpolation value (12 bits) and the value of the period counter (32 bits). The measured values are saved in 48-bit wide data registers, whereby the upper bits are expanded in two's complement representation with sign.
The measured values are called and latched either through external latch inputs, via software or timers, or by reference-mark traverse.

The counter components contain a CPU; the appropriate firmware is loaded to the counter.

Cycle time of the firmware: 25 µs

Offset, phase and amplitude of the sinusoidal encoder signals can be adjusted by software.

Time to access measured values

Incremental encoders

- Without compensation of the encoder signals, without calculation of compensation values:
  Max. 100 µs
- With compensation of the encoder signals, without calculation of compensation values:
  Max. 110 µs
- With compensation of the encoder signals, with calculation of compensation values:
  Max. 160 µs

EnDat/SSI

The access time varies depending on the encoder.
EnDat: 440 kHz clock frequency (one-time call)
SSI: 360 KHz clock frequency

Hardware

Specification of the PCI bus

The IK 220 can be installed in all PCs with PCI bus.

Specification	PCI local bus Spec. Rev. 2.1
Size	Approx. 100 * 190 mm
Connector	PCI 5 V / 32-bit (2*60) connecting element
PCI component	PCI 9052 from PLX, target interface (slave)
Current consumption	+12 V: 22 mA[1)] -12 V: 22 mA +5 V: 620 mA
Power consumption	< 4 watts, without encoders

[1)] Without current consumption of connected encoders

Identifier in component PCI9052:
Vendor ID = 0x10B5
Device ID = 0x9050
Subvendor ID = 0x10B5
Subdevice ID = 0x1172

The IK 220 can be unmistakably identified with these four identifiers. The "Sub Device ID" is assigned exclusively to the IK 220.

Encoder inputs

The IK 220 supports encoders with the following interfaces:

- 11 μA_{PP}
- 1 V_{PP}
- EnDat 2.1
- SSI

The power supply for the encoders (typ. 5.12 V) is generated from the +12 V of the PCI bus. Max. 800 mA of the +5 V power supply for the encoders may be used for both axes.
The additional current taken from the +12 V of the PCI bus is:
$I_{(12V)} = I_{(5V)} * 5.12\ V * 1.35 / 12\ V$
Example: $I_{(5\ V)} = 0.8\ A * 5.12\ V * 1.35 / 12\ V = 461\ mA$

It must be ensured that the power supply limits for the encoder are not exceeded. A voltage converter (370 225-xx) can be used for large cable lengths. The voltage converter has an efficiency of approx. 72 %, meaning that the permissible current consumption of both axes is reduced to 0.72 * 800 mA = 576 mA.

Specification of the 11-μA_{PP} interface

Signal amplitudes I_1, I_2 (0°, 90°) I_0 (reference mark)	 7 μA_{PP} to 16 μA_{PP} 3.5 µA to 8 µA
Signal levels for error message	≤ 2.5 μA_{PP}
Maximum input frequency	Standard: 33 kHz, switchable to 175 kHz
Cable length	Max. 60 m

Specification of the 1 V_{PP} interface

Signal amplitudes A, B (0°, 90°) R (reference mark)	 0.6 V_{PP} to 1.2 V_{PP} 0.2 V to 0.85 V
Signal levels for error message	≤ 0.22 V_{PP}
Maximum input frequency	Standard: 500 kHz, switchable to 33 kHz
Cable length[1)]	Max. 60 m

[1)] Cables up to 150 m are possible, if it can be guaranteed that the encoder is supplied with 5 V from an external power source.
In this case, the maximum input frequency is reduced to max. 250 kHz.

Specification of the EnDat 2.1 interface

The EnDat 2.1 interface of the absolute encoders is bidirectional. It supplies the position values and makes it possible to read from or write to the encoder's memory. Sinusoidal voltage signals (1 Vpp) are available as a complement.
Cable length: max. 10 m
max. 50 m[2)]

[2)] With genuine HEIDENHAIN cables. Ensure that the power supply limits for the encoder are not exceeded.

Specification of the SSI interface

The SSI interface of the absolute encoders is bidirectional. It supplies the absolute position values in synchrony with a clock pulse from the subsequent electronics. Sinusoidal voltage signals (1 Vpp) are available as a complement.
Cable length: max. 10 m

Connection X1, X2 for encoders

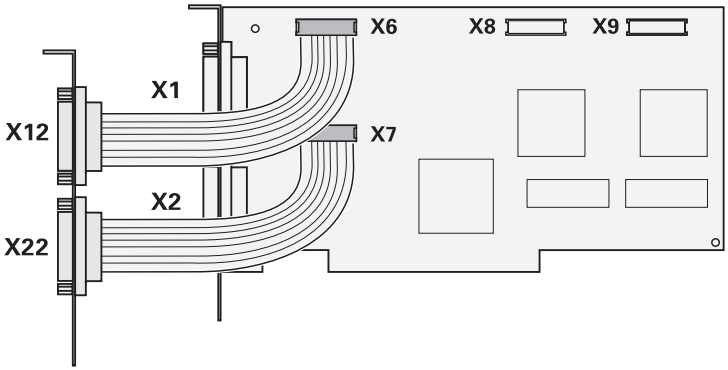
D-sub connection with male contacts (15-pin)

Pin No.	Assignment		
	EnDat/SSI	**1 V_{PP}**	**11 μA_{PP}**
1	+5 V (U_P)	+5 V (U_P)	+5 V
2	0 V (U_N)	0 V (U_N)	0 V
3	A+	A+	I_1+
4	A-	A-	I_1-
5	+ Data	Do not assign	Do not assign
6	B+	B+	I_2+
7	B-	B-	I_2 -
8	- Data	Do not assign	Do not assign
9	+5 V (sensor line)	+5 V	+5 V
10	Do not assign	R+	I_0+
11	0 V (sensor line)	0 V	0 V
12	Do not assign	R-	I_0-
13	Internal shield	0 V	Internal shield
14	+ Clock	Do not assign	Do not assign
15	- Clock	Do not assign	Do not assign
Housing	External shield	External shield	External shield

Encoder outputs

The IK 220 also feeds the encoder signals from inputs X1 and X2 as **sinusoidal current signals** (11 μA_{PP}) to two 10-pin MICROMATCH connectors (female) on the PCB. An additional cable assembly with PC slot cover (Id. Nr. 340 252-01) can be used to lead these connections out to 9-pin D-sub connectors. Adapter cables (Id. Nr. 309 781-xx) for connecting HEIDENHAIN position displays and interpolation units are available (see "Items supplied" and "Accessories").

Encoder outputs
11 μA_{PP}



The maximum cable length depends on the input circuit of the subsequent electronics.

Plug-in PCB for encoder outputs
Connections X6, X7
MICROMATCH with female contact 10-pin

Connection no.[1)]	Signal
1a	I_1-
1b	I_1+
2a	0 V (U_N)
2b	Not assigned
3a	I_2 -
3b	I_2+
4a	Not assigned
4b	I_0+
5a	I_0-
5b	Not assigned

1) Pin 1a is located on the side with the polarizing key.

Encoder outputs (Id. Nr. 340 252-01)
D-sub connection with male contacts (9-pin)

Pin No.	Signal
1	I_1-
2	0 V (U_N)
3	I_2 -
4	Not connected
5	I_0-
6	I_1+
7	Not connected
8	I_2+
9	I_0+
Housing	External shield

Encoder signal compensation

Encoder signals can be compensated automatically — even online. Corresponding functions are included in the software provided with the product.

External inputs/outputs

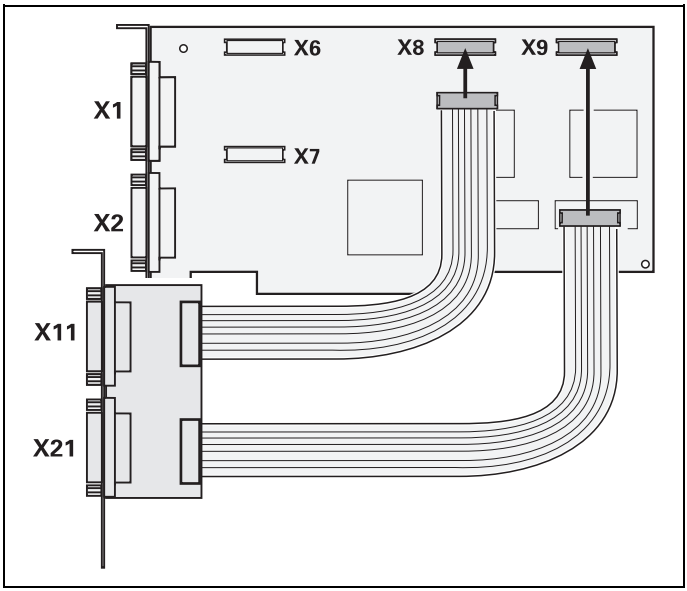
For external inputs/outputs, an additional cable assembly is available with PC slot cover (IK external inputs/outputs Id. Nr. 340 253-01).

Connections X8, X9 for external inputs/outputs

Pin No.	Signal
1a	-L0
1b	0 V
2a	-L1
2b	0 V
3a	-I0
3b	0 V
4a	-I1
4b	-LOUT
5a	+5 V
5b	+5 V

Connections X11 and X21 for external inputs/outputs (option)
D-sub connection with male contacts (9-pin) on PC slot cover

For external inputs/outputs, an optional assembly is available consisting of a slot cover with two D-sub connections, a noise-suppression PCB, and two ribbon cables for connection to 10-pin MICROMATCH connectors on the PCB.



Pin No.	Assignment
1	Output: Measured value latch (X11: -Lout 1; X21: -Lout 2)
2	Input: Measured value latch -L0
3	Input: Measured value latch -L1
4, 5	Do not assign
6	Input: –I0 [1)]
7	Input: –I1 [1)]
8 , 9	0 V

[1)]Additional switching inputs. See IK220GetPort function.

Latching measured values via external inputs
The IK 220 has two external inputs for latching and saving measured values.

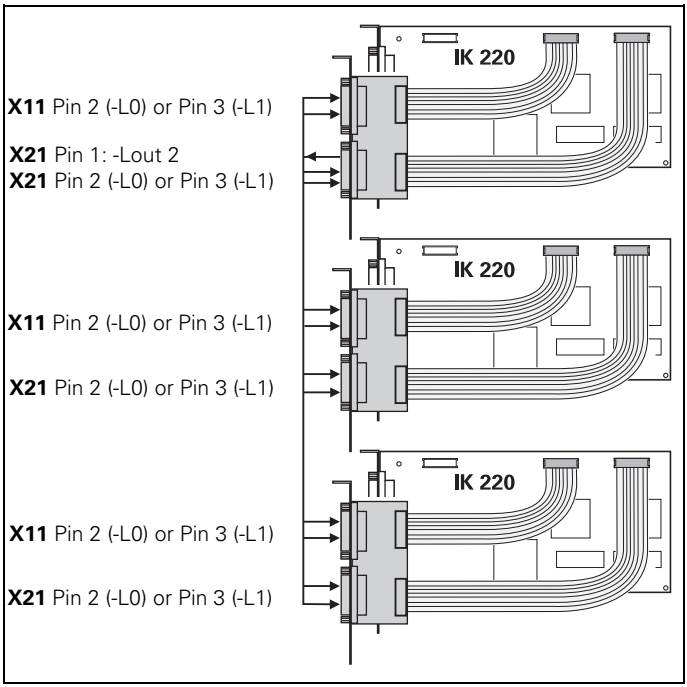
The inputs -L0 and -L1 are low-active; they are kept at high level by a 1.47-kΩ internal pull-up resistor. They can be connected to TTL components.

The simplest way to activate the inputs is to make a bridge from the 0-volt connection (terminals 8, 9) to the input for latching.

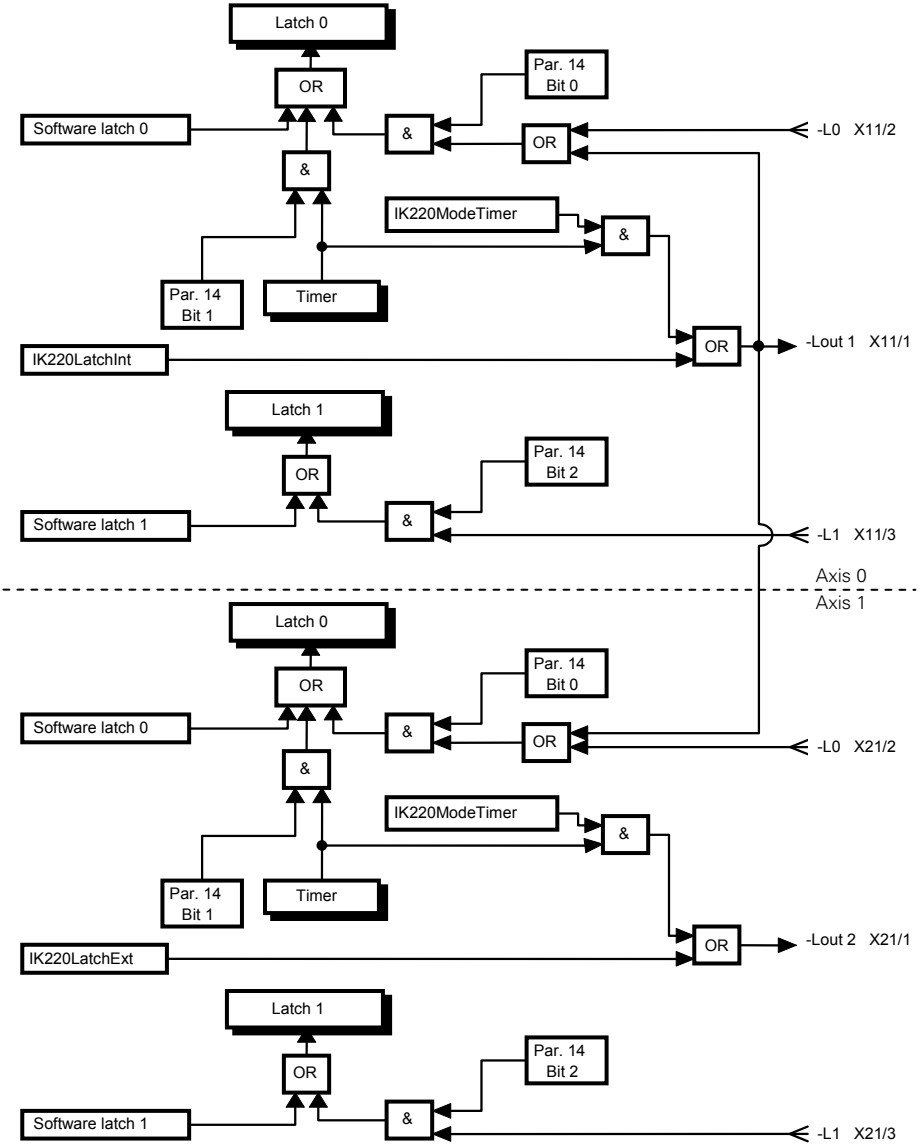
Latch outputs -Lout
The IK 220 supplies two output signals: -Lout 1 (to D-sub connection X11) and -Lout 2 (to D-sub connection X21).
-Lout 1/2 are low-active.
-Lout 1 supplies a low-level pulse simultaneously with synchronous latching of measured values (IK220LatchInt) or with latching by timer.

To latch the measured values of different IKs at the same time (IK220LatchExt), you must use -Lout 2 (see next page).

Latching the measured values of more than one IK 220
For the measured values of all axes of more than one IK to be saved simultaneously, the output signal -Lout 2 must be led to all corresponding encoder inputs (-L0 or -L1), even to the input from which -Lout 2 is led. This enables latching on all axes simultaneously — without differences in run time.



Flow chart: Saving measured values



Operating Parameters

The IK 220 requires operating parameters to properly execute the desired functions. Predetermined default values are set when downloading the supplied operating software. The default values are shown in bold typeface in the following table. You can change the parameter values with function IK220WritePar (Write Parameters), and then check your changes with function IK220ReadPar (Read Parameters).

The following parameters are available (default values in bold typeface):

Param. Number	For-mat	Meaning
1	16 bits	**0: incremental encoder** 1: EnDat 2.1 encoder 2: SSI encoder
2	16 bits	Parameter functions only if parameter 1 = 0 0: 11 µAPP **1: 1 VPP**
3	16 bits	**0: Linear axis** [1)] 1: Angular axis ± infinite
4	16 bits	**0: Positive counting direction** 1: Negative counting direction
5	32 bits	0 to 0xFFFFFFFF: Signal periods/revolution for angular axes, effective only if parameter 3 = 1 [1)] **Default value 0**
6	16 bits	**0:Single REF mark** Distance-coded REF marks: [2)] Indication of the nominal increment as fixed interval in grating periods (TP) Typical examples 500: Fixed interval 500TP 1000: Fixed interval 1000TP 2000: Fixed interval 2000TP 5000: Fixed interval 5000TP

Param. Number	For-mat	Meaning
7	16 bits	0 to 12: Number of interpolation bits **Default value 12** The interpolation value (16-bit width, max. 12 significant bits, left-aligned) is rounded off to the number of set bits.
8	16 bits	**0: Compensation for position value off** 1: Compensation for position value on
9	16 bits	**0: Acquisition of compensation value off** 1: Acquisition of compensation value on
10	16 bits	1 to 8: Number of measuring points per octant within a signal period for calculation of compensation values **Default value 1**
11	16 bits	16 to 47: Time interval for timer **Default value 33 corresponds to 1 ms** 3)
12	16 bits	Software divider for timer, **Default value 1** 3)
13	16 bits	0 to 8191: Number of values per memory cycle that are saved in the internal RAM **Default value 8191**
14	16 bits	Bit 0=1 (integer 1): Enable the external latch L0 Bit 1=1 (integer 2): Enable internal time latch L0 Bit 2=1 (integer 4): Enable external latch L1 **Default value 0**
15	16 bits	1 to 48: SSI code length in bits **Default value 19**
16	16 bits	0: No SSI parity **1: SSI parity (even)** 2: SSI parity (even) with leading zeros
17	16 bits	0: SSI disable Gray-to-binary conversion **1: SSI enable Gray-to-binary conversion**
18	16 bits	0 to 20: SSI no. of leading zeros **Default value 0**
19	16 bits	0 to 20: SSI no. of trailing zeros **Default value 0**

1) Parameter is without function

2) Depending on the connected encoder (only even 16-bit values possible)

3) See next page

Parameter 11: Interval between two latches per timer. The following values can be set directly by the timer of the IK 220:

Parameter value	Time interval	Parameter value	Time interval
17	100 µs[1)]	33	1000 µs
18	150 µs[2)]	34	1100 µs
19	200 µs	35	1200 µs
20	250 µs	36	1300 µs
21	300 µs	37	1400 µs
22	350 µs	38	1500 µs
23	400 µs	39	1600 µs
24	450 µs	40	1800 µs
25	500 µs	41	2000 µs
26	550 µs	42	2200 µs
27	600 µs	43	2400 µs
28	650 µs	44	2600 µs
29	700 µs	45	2800 µs
30	750 µs	46	3000 µs
31	800 µs	47	3200 µs
32	900 µs		

Parameter 12: To permit a time interval of over 3.2 milliseconds, parameter 12 can realize a counter that uses only every *n*th timer pulse for latching. To permit a latching interval of 9 milliseconds, for example, parameter 11 must be set to 46 (corresponds to 3 milliseconds) and parameter 12 set to 3.

[1)] Only possible without compensation of encoder signals and without calculation of compensation values.
[2)] With compensation; without calculation of compensation values.

Driver Software for WINDOWS

General information

The driver software for the IK 220 enables applications to access the IK 220 from Windows 95/98, Windows NT/2000/XP, Linux and LabView.

For Windows, access is made through a Dynamic Link Library (DLL) and a Windows 95/98, Windows NT or Windows 2000/XP device driver. The drivers and application examples are located on the CD that is among the items supplied.

The content of the CD as well as the drivers for other operating systems (e.g. Windows Vista, Windows 7) are available in the download area at www.heidenhain.de. For more information, see "readme.txt" of the downloaded file.

If you have any questions, please contact the HEIDENHAIN Encoder Support.

Content of "Disk1" directory:
- Installation routine for NT and Win95/98 driver
- DLL with source code
- NT driver with source code

Content of "Disk2" directory:
- Visual C++ example with source code
- Console application example with source code
- Visual Basic 5 example with source code

Content of "Disk3" directory:
- Delphi 4 example with source code

Content of "Disk4" directory:
- Installation routine for Windows 2000/XP driver
- Windows 2000/XP driver (WDM) with source code

Content of "Disk5" directory:
- LabView library
- Example programs

Content of "Disk6" directory:
- Linux driver for Kernel 2.4
- Example programs
- Description and installation instructions
- Source code

Installing the drivers and DLLs under Windows 2000 and Windows XP

- After inserting the IK 220 card into your computer, restart your computer.
- Follow the instructions of the automatic installation wizard.
- Select the "IK220.inf" setup information file in the "Disk4" directory on the CD.
- Follow the instructions of the automatic installation wizard.

Installing the Drivers and DLLs under Windows NT and Windows 95/98

- On the supplied CD, select the "Disk1\Install" directory.
- Call "Install.Bat."

Device driver for Windows 2000/XP (IK220DRV.SYS)

The Windows 2000/XP driver is a WDM driver for Windows 2000 and XP. It enables access to the IK 220. The driver supports up to eight IK 220s. To install the driver, simply select the setup information file (IK220.inf) in the "Disk4" directory. The automatic installation wizard will guide you through the installation process step by step.

Device driver for Windows NT (IK220DRV.SYS)

The Windows NT device driver is a kernel-mode driver for Windows NT (Versions 3.51 and 4.0). It enables access to the IK 220. The driver supports up to eight IK 220s. The installation of the device driver is taken care of by the "Install.Bat" batch file in the "Disk1\Install" subdirectory of the "IK220" directory on the CD.

Device driver for Windows 95/98 (IK220VXD.VXD)
The Windows 95/98 device driver is a virtual device driver for Windows 95/98 that supports access to up to eight IK 220. The installation of the device driver is taken care of by the "Install.Bat" file in the "Disk1\Install" directory of the "IK220" directory on the CD.

The Windows DLL (IK220DLL.DLL)
This DLL enables the IK 220 to access application programs. There is one DLL for Windows NT/2000/XP and one for Windows 95/98. Under Windows NT/2000/XP, the IK 220 is accessed through the device driver for Windows NT/2000/XP. Under Windows 95/98, the DLL accesses the registry of the IK 220 through the virtual device driver.

To install the device driver, you must have administrative rights on the target computer.

Examples

Example for console application
In the subdirectory "\Disk2\IK220Con\Release" of the "IK 220" directory on the CD you will find a simple console application: Start IK220Con.exe.
Note: This example is suited only for HEIDENHAIN encoders with 1 V_{PP} sinusoidal voltage signals.

Example for Visual C++
In the subdirectory "\Disk2\IK220App\Release" of the "IK220" directory on the CD you will find an application in Visual C++: Start IK220App.exe.
Select the interface (1 V_{PP}, 11 μA_{PP}, EnDat) and set the encoder parameters under "Setup."

Example for Visual Basic
In the subdirectory "\Disk2\IK220VB5" of the "IK 220" directory on the CD you will find an application in Visual Basic: Start IK220App.exe.
Select the interface (1 V_{PP}, 11 μA_{PP}, EnDat) and set the encoder parameters under "Setup."

Example for Borland Delphi
In the subdirectory "\Disk3\Delphi" of the "IK 220" directory on the CD you will find an application in Borland Delphi: Start IK220.exe.
Select the interface (1 V_{PP}, 11 μA_{PP}, EnDat, SSI) and set the encoder parameters under "Parameters/Encoder."

Examples for LabView
In the subdirectory "\Disk5" of the "IK 220" directory on the CD you will find example applications in LabView.

Example for Linux
In the subdirectory "\Disk6" of the "IK 220" directory on the CD you will find a simple console application: Compile and start ik220_read48.

All applications listed above are intended as programming examples in the respective language. The programming examples are not intended for use in production.

Calling the DLL functions from an application program

To be able to use the functions of the DLL they must be known by the application program.

Microsoft Visual C++

If the application program is written with Visual C++, the file "\IK220Dll\Release\IK220DLL.LIB" is to be copied into the library directory of Visual C++ (e.g.: C:\MSDEV\LIB). Moreover, this library must be linked. This requires an entry under "Build/Settings/Link/Object/library modules."
The header file "\Include\DLLFunc.h" in which the function prototypes are defined must be added to the project.
After this is done, the functions can be used as "normal" C functions.

Microsoft Visual C++

For Microsoft Visual Basic, the functions are defined in the module "\Include\DLLFunc.bas." This file must be included in the project.

Borland Delphi

The functions and types are defined in the file "\Include\DLLFunc.pas" to enable the DLL functions to be used with Borland Delphi.

Overview of DLL functions

Function	Short reference	
Determine installed IK 220	BOOL IK220Find	(ULONG* pBuffer16)
Initialize IK 220	BOOL IK220Init	(USHORT Axis)
Read program versions	BOOL IK220Version	(USHORT Axis, char* pVersCard, char* pVersDrv, char* pVersDll)
Clear counter	BOOL IK220Reset	(USHORT Axis)
Start counter	BOOL IK220Start	(USHORT Axis)
Stop counter	BOOL IK220Stop	(USHORT Axis)
Delete frequency and amplitude error	BOOL IK220ClearErr	(USHORT Axis)
Save counter value	BOOL IK220Latch	(USHORT Axis, USHORT Latch)
Save synchronous counter value internally	BOOL IK220LatchInt	(USHORT Card)
Save synchronous counter value externally	BOOL IK220LatchExt	(USHORT Card)

Function	Short reference	
Delete counter with next reference mark	BOOL IK220ResetRef	(USHORT Axis)
Start counter with next reference mark	BOOL IK220StartRef	(USHORT Axis)
Stop counter with next reference mark	BOOL IK220StopRef	(USHORT Axis)
Save counter with next reference mark	BOOL IK220LatchRef	(USHORT Axis)
Inquire whether counter value is saved	BOOL IK220Latched	(USHORT Axis, USHORT Latch, BOOL* pStatus)
Wait until counter value is saved	BOOL IK220WaitLatch	(USHORT Axis, USHORT Latch)
Set the timeout time	BOOL IK220SetTimeOut	(ULONG TimeOut)
Set position value	BOOL IK220Set	(USHORT Axis, double SetVal)
Set preset value	BOOL IK220SetPreset	(USHORT Axis, double PresVal)
Read preset value	BOOL IK220GetPreset	(USHORT AXIS, double* PresVal)
Read counter value (32 bits)	BOOL IK220Read32	(USHORT Axis, USHORT Latch, LONG* pData)
Read counter value (48 bits)	BOOL IK220Read48	(USHORT Axis, USHORT Latch, double* pData)
Read latched counter value (32 bits)	BOOL IK220Get32	(USHORT Axis, USHORT Latch, LONG* pData)
Read latched counter value (48 bits)	BOOL IK220Get48	(USHORT Axis, USHORT Latch, double* pData)
Read encoder status	BOOL IK220CntStatus	(USHORT Axis, USHORT Latch, USHORT* pRefSta, SHORT* pKorr00, SHORT* pKorr90, SHORT* pNKorr00, SHORT* pNKorr90, USHORT* pSamCnt)
Start reference point traverse	BOOL IK220DoRef	(USHORT Axis)
Interrupt reference point traverse	BOOL IK220CancelRef	(USHORT Axis)
Inquiry whether REF function is active	BOOL IK220RefActive	(USHORT Axis, BOOL* pStatus)
Wait until REF function is ended	BOOL IK220WaitRef	(USHORT Axis)
Find position of reference mark	BOOL IK220PositionRef	(USHORT Axis, double* pData, LONG* pPeriod, USHORT* pIntpol)

Function	Short reference
Find position of the rising and falling edge of the reference mark	BOOL IK220PositionRef2 (USHORT Axis, double*pData, LONG* pPeriod, USHORT* pIntpol)
Read status of IK 220	BOOL IK220Status (USHORT Axis, ULONG* pStatus)
Read status of DLL functions	BOOL IK220DllStatus (ULONG* pDLLStatus, ULONG* pDLLInfo)
Read REF status	BOOL IK220RefStatus (USHORT Axis, LONG* pRef1, LONG* pRef2, LONG* pDiff, LONG* pCode, USHORT* pFlag)
Read signal status	BOOL IK220SignalStatus (USHORT Axis, USHORT* Freq, USHORT* pAmin, USHORT* pAact, USHORT* pAmax)
Read adjusted compensation values	BOOL IK220GetCorrA (USHORT Axis,SHORT* pOfs0, SHORT* pOfs90, SHORT* Pha0, SHORT* pPha90,SHORT* Sym0, SHORT* pSym90, USHORT* pFlag1, USHORT* pFlag2)
Read calculated compensation values	BOOL IK220GetCorrB (USHORT Axis,SHORT* pOfs0, SHORT* pOfs90,SHORT* Pha0, SHORT* Pha90,SHORT*pSym0, SHORT* pSym90, USHORT* pFlag1, USHORT* pFlag2)
Read compensation values	BOOL IK220LoadCorrA (USHORT Axis,SHORT Ofs0, SHORT Ofs90,SHORT Pha0, SHORT Pha90,SHORT Sym0, SHORT Sym90)
Read octant status	BOOL IK220OctStatus (USHORT Axis, USHORT* pOct0, USHORT* pOct1, USHORT* pOct2, USHORT* pOct3, USHORT* pOct4, USHORT* pOct5, USHORT* pOct6, USHORT* pOct7, USHORT* pSamCnt)
Read checksum of parameters	BOOL IK220ChkSumPar (USHORT Axis, USHORT* pChkSum)

Function	Short reference
Read checksum of firmware	BOOL IK220ChkSumPrg (USHORT Axis, USHORT* pChkSum1, USHORT* pChkSum2)
Write parameters	BOOL IK220WritePar (USHORT Axis, USHORT ParNum, ULONG ParVal)
Read parameters	BOOL IK220ReadPar (USHORT Axis, USHORT ParNum, ULONG* pParVal)
Reset EnDat encoder	BOOL IK220ResetEn (USHORT Axis, USHORT* pStatus)
Read configuration of EnDat encoder	BOOL IK220ConfigEn (USHORT Axis, USHORT* pStatus, USHORT* pType, ULONG* pPeriod, ULONG* pStep, USHORT* pTurns, USHORT* pRefDist, USHORT* pCntDir)
Read EnDat encoder value	BOOL IK220ReadEn (USHORT Axis, USHORT* pStatus, double* pData, USHORT* pAlarm)
Read absolute and incremental counter value of the EnDat encoder	BOOL IK220ReadEnInc (USHORT Axis, USHORT Latch, USHORT* pStatus, double* pDataEn, USHORT* pAlarm, double* pDataInc)
Set mode for continuous EnDat clock	BOOL IK220ModeEnCont (USHORT Axis, USHORT* Latch, USHORT Mode, USHORT* pStatus)
Read absolute and incremental counter value of the EnDat encoder with continuous clock	BOOL IK220ReadEnIncCont (USHORT Axis, USHORT* pStatus, double* pDataEn, USHORT* pAlarm, double* pDataInc, USHORT* pSigStat)
Read EnDat encoder alarm word	BOOL IK220AlarmEn (USHORT Axis, USHORT* pAlarm)
Read EnDat encoder warning word	BOOL IK220WarnEn (USHORT Axis, USHORT* pWarn)

Function	Short reference
Read value from memory area of the EnDat encoder	BOOL IK220ReadMemEn (USHORT Axis, USHORT Range, USHORT MemAdr, USHORT* pMemData, USHORT* pStatus)
Write value from memory area of the EnDat encoder	BOOL IK220WriteMemEn (USHORT Axis, USHORT Range, USHORT MemAdr, USHORT MemData, USHORT* pStatus)
Read absolute counter value of the SSI encoder	BOOL IK220ReadSSI (USHORT Axis, USHORT* pStatus, double* pData)
Read absolute and incremental counter value of the SSI encoder	BOOL IK220ReadSsiInc (USHORT Axis, USHORT Latch, USHORT* pStatus, double* pDataSsi, double* pDataInc)
Determine value for timer	BOOL IK220SetTimer (USHORT Axis, ULONG SetVal, ULONG* pTimVal)
Determine mode for timer	BOOL IK220ModeTimer (USHORT Axis, USHORT Mode)
Determine mode for RAM buffer	BOOL IK220ModeRam (USHORT Axis, USHORT Mode)
Erase RAM buffer	BOOL IK220ResetRam (USHORT Axis)
Read counter value from RAM buffer	BOOL IK220GetRam (USHORT Axis, double* pData, USHORT* pRead, USHORT* pWrite, USHORT* pStatus)
Read counter value block from RAM buffer	BOOL IK220BurstRam (USHORT Axis, USHORT maxCount, double* pData, USHORT* pCount, USHORT* pStatus)
Read amplitude values from RAM buffer	BOOL IK220GetSig (USHORT Axis, USHORT* pPeriod, SHORT* pAmp0, SHORT* pAmp90, USHORT* pRead, USHORT* pWrite, USHORT* pStatus)

Function	Short reference	
Read amplitude values block from RAM buffer	BOOL IK220BurstSig	(USHORT Axis, USHORT maxCount, USHORT* pPeriod, SHORT* pAmp0, SHORT* pAmp90, USHORT* pCount, USHORT* pStatus)
LED control of axes	BOOL IK220Led	(USHORT Axis, USHORT Mode)
LED control of card	BOOL IK220SysLed	(USHORT Card, USHORT Mode)
Read external inputs	BOOL IK220GetPort	(USHORT Axis, USHORT* pPortInfo, USHORT* pRising, USHORT* pFalling)
Evaluation of the reference-mark signal	IK220RefEval	(USHORT Axis, USHORT Mode)
Input frequency for sinusoidal incremental signals	IK220SetBw	(USHORT Axis, USHORT Mode)

The following functions are used by the driver software.
They should not be used in application programs.

Function	Short reference	
Read IK 220 register (16 bits)	BOOL IK220InputW	(USHORT Axis, USHORT Adr, USHORT* pData)
Read IK 220 register (32 bits)	BOOL IK220InputL	(USHORT Axis, USHORT Adr, ULONG* pData)
Write IK 220 register (16 bits)	BOOL IK220Output	(USHORT Axis, USHORT Adr, USHORT Data)
Read value from RAM of the IK 220	BOOL IK220RamRead	(USHORT Axis, USHORT Adr, USHORT* pData)
Write value into RAM of the IK 220	BOOL IK220RamWrite	(USHORT Axis, USHORT Adr, USHORT Data)
Load firmware in the IK 220	BOOL IK220DownLoad	(USHORT Axis, USHORT* pPgmData, ULONG PgmSize)
Set EnDat clock line	BOOL IK220SetEnClock	(USHORT Axis, BOOL State, USHORT* pStatus)
Set EnDat data line	BOOL IK220SetEnData	(USHORT Axis, BOOL State, USHORT* pStatus)
Read EnDat data line	BOOL IK220ReadEnData	(USHORT Axis, BOOL State)

Reference of DLL functions

All DLL functions return a Boolean variable. If this variable is "true" (i.e.: <>0), then the function was successful. If it contains the value "false" (i.e., =0), then there has been an error. Input values in the functions are transferred as numerical values (transfer by value). If the function has a return code, then the address of the return code is transferred (transfer by reference) to the function (pointer to return code).

The following types of data are used:

USHORT	:	Unsigned 16-bit
USHORT*	:	Pointer to USHORT
SHORT	:	Signed 16-bit
SHORT*	:	Pointer to SHORT
ULONG	:	Unsigned 32-bit
ULONG*	:	Pointer to ULONG
LONG	:	Signed 32-bit
LONG*	:	Pointer to LONG
double	:	Floating comma 64-bit
double*	:	Pointer to double
BOOL	:	Boolean variable 32-bit
BOOL*	:	Pointer to BOOL
chat	:	Pointer to string (terminated with 0x00)

IK220Find

Supplies the address of each axis of the installed IK 220. Can be used to determine the number of installed IK 220s. For every IK 220, two addresses are saved at the corresponding position in pBuffer16. The unused entries are set to 0. For each axis, IK220Init must subsequently be called in order to load and start the firmware!

Prototype: BOOL IK220Find (ULONG* pBuffer16);

pBuffer16: Pointer to 16 long words (16*4 bytes)

IK220Init

Loads the firmware into the IK 220 and starts it. **Must be called for every axis before further functions can be used!**

Prototype: BOOL IK220Init (USHORT Axis);

Axis: Number of the axis (0 to 15)

IK220Version

Reads the program versions of the IK 220, the NT device driver and the DLL. The program versions are saved as ASCII characters. There must be room reserved for at least 20 characters. The character strings are concluded with a zero byte.

Prototype: **BOOL IK220Version (USHORT Axis, char* pVersCard, char* pVersDrv, char* pVersDll)**

Axis: Number of the axis (0 to 15)

pVersCard: Pointer to the program version of the IK 220 firmware

pVersDrv: Pointer to the program version of the Windows NT device drivers (only under Windows NT)

pVersDll: Pointer to the program version of the DLL

IK220Reset

The counter is set to zero.

Prototype: **BOOL IK220Reset (USHORT Axis);**

Axis: Number of the axis (0 to 15)

IK220Start

Starts the counter.

Prototype: **BOOL IK220Start (USHORT Axis);**

Axis: Number of the axis (0 to 15)

IK220Stop

Stops the counter.

Prototype: **BOOL IK220Stop (USHORT Axis);**

Axis: Number of the axis (0 to 15)

IK220ClearErr

Deletes the amplitude and frequency error status.

Prototype: **BOOL IK220ClearErr (USHORT Axis);**

Axis: Number of the axis (0 to 15)

IK220Latch

Saves the counter value in the indicated register.

Prototype: **BOOL IK220Latch (USHORT Axis, USHORT Latch);**

Axis: Number of the axis (0 to 15)

Latch: 0=Counter value is saved in register 0
1=Counter value is saved in register 1
2=Counter value is saved in register 2
(without interpolation)

IK220LatchInt
Generates a signal with which the counter values of both axes of an IK 220 are saved synchronously in Latch 0. Must first be enabled through parameter 14.
Prototype: BOOL IK220LatchInt (USHORT Card);
Card: Number of the card (0 to 7)

IK220LatchExt
Generates a signal with which the counter values of several axes of an IK 220 are saved synchronously in Latch 0/1 through an external connection. Must first be enabled through parameter 14.
Prototype: BOOL IK220LatchExt (USHORT Card);
Card: Number of the card (0 to 7)

IK220ResetRef
The counter is set to zero with the next reference mark.
Prototype: BOOL IK220ResetRef (USHORT Axis);
Axis: Number of the axis (0 to 15)

IK220StartRef
The counter is started with the next reference mark.
Prototype: BOOL IK220StartRef (USHORT Axis);
Axis: Number of the axis (0 to 15)

IK220StopRef
The counter is stopped with the next reference mark.
Prototype: BOOL IK220StopRef (USHORT Axis);
Axis: Number of the axis (0 to 15)

IK220LatchRef
With the next reference mark, the counter value is saved in register 2. The saved value is without interpolation and can be output with IKGet32 or IKGet48.
Prototype: BOOL IK220LatchRef (USHORT Axis);
Axis: Number of the axis (0 to 15)

IK220Latched
Determines whether the counter value was saved.
Prototype: BOOL IK220Latched (USHORT Axis, USHORT Latch, BOOL* pStatus);
Axis: Number of the axis (0 to 15)
Latch: 0 = Request for register 0.
1 = Request for register 1
2 = Request for register 2
pStatus: Pointer to a variable in which the status is saved.
False (=0) = value not saved
True (≠0) = value was saved

IK220WaitLatch
Waits until the counter value was saved. If no timeout time was defined, the function waits until a numerical value is saved in the corresponding latch.
Prototype: BOOL IK220WaitLatch (USHORT Axis, USHORT Latch);
Axis: Number of the axis (0 to 15)
Latch: 0 = Request for register 0.
1 = Request for register 1
2 = Request for register 2

IK220SetTimeOut
With this function you can define a timeout time. If no timeout time is defined, the IK220WaitLatch, IK220WaitRef and IK220PositionRef functions wait until the respective event occurs.
Prototype: BOOL IK220SetTimeOut (ULONG TimeOut);
TimeOut: 0 = No timeout
1.. = Timeout in ms

IK220Set

Sets the position value to the indicated value. Uses Register 0 to determine the current position, and calculates the preset value from that. The IK220Read48, IK220Get48, IK220ReadEnInc, IK220ReadEnIncCont, IK220ReadSsiInc, IK220GetRam and IK220BurstRam functions then deliver incremental position values which refer to the preset value (see IK220SetPreset and IK220GetPreset).

Prototype: BOOL IK220Set (USHORT Axis, double SetVal);

Axis: Number of the axis (0 to 15)
SetVal: New position value

IK220SetPreset

Sets the preset value to the indicated value. When IK220Set is called, the set preset value is always added to the actual value of the axis.

Prototype: BOOL IK220SetPreset (USHORT Axis, double PresVal);

Axis: Number of the axis (0 to 15)
PresVal: New preset value

IK220GetPreset

Supplies the preset value of the indicated axis.

Prototype: BOOL IK220GetPreset (USHORT Axis, double* pPresVal);

Axis: Number of the axis (0 to 15)
pPresVal: Pointer to a variable in which the preset value is saved

IK220Read32

Supplies the 32-bit counter value.

Prototype: BOOL IK220Read32 (USHORT Axis, USHORT Latch, LONG* pData);

Axis: Number of the axis (0 to 15)
Latch: 0 = Output from register 0
1 = Output from register 1
pData: Pointer to a variable in which the position value is saved.

IK220Read48

Supplies the 48-bit counter value.

Prototype: **BOOL IK220Read48 (USHORT Axis, USHORT Latch, double* pData);**

Axis: Number of the axis (0 to 15)

Latch: 0 = Output from register 0
1 = Output from register 1

pData: Pointer to a variable in which the counter value is saved.

IK220Get32

Supplies the 32-bit counter value (20 bits before and 12 bits after the decimal point (interpolation values)). Before the counter value can be output, it must be saved in register 0, register 1 or register 2 (external function, IKLatchInt, IK220LatchExt, Timer, IK220Latch or IK220LatchRef) and then you must perhaps inquire whether the counter value has already been saved (IKLatched, IKWaitLatch).

Prototype: **BOOL IK220Get32 (USHORT Axis, USHORT Latch, LONG* pData);**

Axis: Number of the axis (0 to 15)

Latch: 0 = Output from register 0
1 = Output from register 1
2 = Output from register 2

pData: Pointer to a variable in which the counter value is saved.

IK220Get48

Supplies the 48-bit counter value. Before the counter value can be output, it must be saved in register 0, register 1 or register 2 (external function, IK220LatchInt, IK220LatchExt, Timer, IK220Latch or IK220LatchRef) and then you must perhaps inquire whether the counter value has already been saved (IKLatched, IKWaitLatch).

Prototype: **BOOL IK220Get48 (USHORT Axis, USHORT Latch, double* pData);**

Axis: Number of the axis (0 to 15)

Latch: 0 = Output from register 0
1 = Output from register 1
2 = Output from register 2

pData: Pointer to a variable in which the counter value is saved (floating-point value: decimal places = interpolation value).

IK220CntStatus
Supplies additional information on the last counter value of the corresponding register.

Prototype: BOOL IK220CntStatus (USHORT Axis, USHORT Latch, USHORT* pRefSta, SHORT* pKorr00, SHORT* pKorr90, SHORT* pNKorr00, SHORT* pNKorr90, USHORT* pSamCnt);

Axis: Number of the axis (0 to 15)
Latch: 0 = Output counter register 0
1 = Output counter register 1
pRefSta: Pointer to a variable in which the REF status is saved
pKorr00: Pointer to a variable in which the compensated 0° analog value is saved.
pKorr90: Pointer to a variable in which the compensated 90° analog value is saved.
pNKorr00: Pointer to a variable in which the uncompensated 0° analog value is saved.
pNKorr90: Pointer to a variable in which the uncompensated 90° analog value is saved.
pSamCnt: Pointer to a variable in which the current number of measuring points is saved for determining the compensation value.

IK220DoRef
Starts reference-point traverse. The REF marks are evaluated as defined in Parameter 6.

Prototype: BOOL IK220DoRef (USHORT Axis);

Axis: Number of the axis (0 to 15)

IK220CancelRef
Interrupts reference-point traverse.

Prototype: BOOL IK220CancelRef (USHORT Axis);

Axis: Number of the axis (0 to 15)

IK220RefActive
Ascertains whether a REF function is running (Reset, Start or Stop with REF or REF traverse).

Prototype: BOOL IK220RefActive (USHORT Axis, BOOL* pStatus);

Axis: Number of the axis (0 to 15)
pStatus: Pointer to a variable in which the status is saved.
False (=0) = REF function not active
True (≠0) = REF function active

IK220WaitRef

Waits until all active REF functions are ended (Reset, Start or Stop with REF or REF traverse). If no timeout time was defined, the function waits until the REF function is ended.

Prototype: BOOL IK220WaitRef (USHORT Axis);

Axis: Number of the axis (0 to 15)

IK220PositionRef

Waits for an active edge of the reference pulse and then carries out a latch command. The latched value corresponds to the position of the reference mark. If no timeout time was defined, the function waits until a reference pulse is detected. If the reference pulse is already active when the function is called, "FALSE" is returned.

Prototype: BOOL IK220PositionRef (USHORT Axis, double* pData, LONG* pPeriod, USHORT* pIntpol)

Axis: Number of the axis (0 to 15)

pData: Pointer to a variable in which the counter value is saved.

pPeriod: Pointer to a variable in which the signal period value is saved.

pIntpol: Pointer to a variable in which the interpolation value is saved.

IK220PositionRef2

Waits for an active edge of the reference pulse and then saves the current position value. Then waits for the falling edge of the reference pulse and also saves that position value. The saved values correspond to the position of the rising and falling edges of the reference mark (see IK 220 Specifications). If no timeout time was defined, the function waits until a reference pulse is detected (see IK220SetTimeOut). If the reference pulse is already active when the function is called, or if the timeout time has expired before the reference mark was recognized, "FALSE" is returned. The axis **must** be completely newly initialized after a timeout!

Prototype: BOOL IK220PositionRef2 (USHORT Axis, double* pData, LONG* pPeriod, USHORT* pIntpol)

Axis: Number of the axis (0 to 15)

pData1: Pointer to a variable in which the position value of the rising edge is saved

pPeriod1: Pointer to a variable in which the signal-period value of the rising edge is saved

pIntpol1: Pointer to a variable in which the interpolation value of the rising edge is saved

pData2: Pointer to a variable in which the position value of the falling edge is saved
pPeriod2: Pointer to a variable in which the signal-period value of the falling edge is saved
pIntpol2: Pointer to a variable in which the interpolation value of the falling edge is saved

IK220Status

Reports the status of the IK 220.

Prototype: BOOL IK220Status (USHORT Axis, ULONG* pData);

Axis: Number of the axis (0 to 15)
pData: Pointer to a variable in which the status is saved.

Bit number	Meaning
0	1 = Counter value latched in register 0
1	1 = Counter value latched in register 1
2	1 = Counter value latched in register 2
3	1 = EnDat call with continuous clock
4	1 = Contamination warning / error
5	1 = Counter started
6	1 = REF function active (start, stop, reset or latch)
7	1 = Frequency exceeded
8	1 = Compensation values calculated once
9	1 = Calculation of compensation values active
10	1 = Values being latched in RAM buffer
11	
12	
13	
14 and 15	REF status: 0 = No REF traverse 1 = Waiting for 1st ref. mark 2 = Waiting for 2nd ref. mark 3 = REF traverse completed
16 to 31	

IK220DllStatus

Reports the status of the DLL functions.

Prototype: **BOOL IK220DllStatus (ULONG* pDLLStatus, ULONG* pDLLInfo);**

Axis: Number of the axis (0 to 15)

pDLLStatus: Pointer to a variable in which the status of the DLL functions is saved.

pDLLInfo: Pointer to a variable in which internal status information is saved.

The DLL status has the following meaning:

Bit number	Meaning
0	Error message from IK 220
1	Timeout error in DLL function
2	False command acknowledgment from IK 220
3	
4 to 7	
8 to 11	
12 to 15	
16 to 19	
20	Area limit violation
21	REF signal is already active
22	Timer number too high
23	Error while calling device driver
24	Incorrect data size while calling device driver
25	Incorrect mode
26	Alarm from EnDat encoder
27	Axis number too high
28	Axis not installed
29	Address too high
30	Latch number too high
31	Invalid memory address

The DLL info has the following meaning:

Bit number	Meaning
0	Windows timer is not available
1	Windows timer is not used
2	
3	
4 to 7	
8 to 11	
12 to 15	
16 to 19	
20 to 23	
24 to 27	
28 to 31	

IK220RefStatus

Reports the detailed REF status of the IK 220.

Prototype: BOOL IK220RefStatus (USHORT Axis, LONG* pRef1, LONG* pRef2, LONG* pDiff, Long* pCode, USHORT* pFlag);

Axis: Number of the axis (0 to 15)

pRef1: Pointer to a variable in which the position of the 1st reference mark is saved

pRef2: Pointer to a variable in which, with distance-coded reference marks, the position of the 2nd reference mark is saved

pDiff: Pointer to a variable in which, with distance-coded reference marks, the calculated difference is saved

pCode: Pointer to a variable in which, with distance-coded reference marks, the calculated offset is saved

pFlag: Pointer to a variable in which the REF status is saved
0 = No REF traverse
1 = Waiting for 1st ref. mark
2 = Waiting for 2nd ref. mark
3 = REF traverse completed

IK220SignalStatus

Reports the signal status of the IK 220.

Prototype: BOOL IK220SignalStatus (USHORT Axis, USHORT* pFreq, USHORT* pAmin, USHORT* pAact, USHORT* pAmax);

Axis: Number of the axis (0 to 15)

pFreq: Pointer to a variable in which the status of the excessive frequency is saved.
0 = OK
1 = Frequency exceeded

pAmin: Pointer to a variable in which the status of the minimum amplitude is saved.

pAact: Pointer to a variable in which the status of the current amplitude is saved.

pAmax: Pointer to a variable in which the status of the maximum amplitude is saved.

Status for amplitude: 0 = Amplitude normal
1 = Amplitude low
2 = Amplitude too high
3 = Amplitude too low

	11 μA_{PP}	1 V_{PP}	Code
Amplitude too small	2.5 μA_{PP}	0.22 V_{PP}	03
Amplitude low	5 μA_{PP}	0.44 V_{PP}	01
Amplitude normal			00
Amplitude too high	16.25 μA_{PP}	1.40 V_{PP}	02

IK220GetCorrA

Reports the adjusted compensation values of the IK 220. Ascertainment of the compensation must first have been enabled by parameter 9.

Prototype: BOOL IK220GetCorrA (USHORT Axis, SHORT* pOfs0, SHORT* pOfs90, SHORT* pPha0, SHORT* pPha90, SHORT* pSym0, SHORT* pSym90, USHORT* pFlag1, USHORT* pFlag2);

Axis: Number of the axis (0 to 15)

pOfs0: Pointer to a variable in which the offset of the 0° signal is saved

pOfs90: Pointer to a variable in which the offset of the 90° signal is saved

pPha0: Pointer to a variable in which the phase of the 0° signal is saved

pPha90: Pointer to a variable in which the phase of the 90° signal is saved

pSym0: Pointer to a variable in which the symmetry of the 0° signal is saved
pSym90: Pointer to a variable in which the symmetry of the 90° signal is saved
pFlag1: Pointer to a variable in which flag 1 is saved
pFlag2: Pointer to a variable in which flag 2 is saved

IK220GetCorrB

Reports the calculated compensation values of the IK 220. Ascertainment of the compensation must first have been enabled by parameter 9.

Prototype: BOOL IK220GetCorrB (USHORT Axis, SHORT* pOfs0, SHORT* pOfs90, SHORT* pPha0, SHORT* pPha90, SHORT* pSym0, SHORT* pSym90, USHORT* pFlag1, USHORT* pFlag2);

Axis: Number of the axis (0 to 15)
pOfs0: Pointer to a variable in which the offset of the 0° signal is saved
pOfs90: Pointer to a variable in which the offset of the 90° signal is saved
pPha0: Pointer to a variable in which the phase of the 0° signal is saved
pPha90: Pointer to a variable in which the phase of the 90° signal is saved
pSym0: Pointer to a variable in which the symmetry of the 0° signal is saved
pSym90: Pointer to a variable in which the symmetry of the 90° signal is saved
pFlag1: Pointer to a variable in which flag 1 is saved
pFlag2: Pointer to a variable in which flag 2 is saved

IK220LoadCorrA

Loads the compensation value into the IK 220.
The compensation calculation must then be released by Parameter 8.

Prototype: BOOL IK220LoadCorrA (USHORT Axis, SHORT Ofs0, SHORT Ofs90, SHORT Pha0, SHORT Pha90, SHORT Sym0, SHORT Sym90);

Axis: Number of the axis (0 to 15)
Ofs0: Compensation value for offset of the 0° signal
Ofs90: Compensation value for offset of the 90° signal
Pha0: Compensation value for phase of the 0° signal
Pha90: Compensation value for phase of the 90° signal
Sym0: Compensation value for symmetry of the 0° signal
Sym90: Compensation value for symmetry of the 90° signal.

IK220OctStatus

Reports the octant status of the IK 220.

Prototype: **BOOL IK220OctStatus (USHORT Axis, USHORT* pOct0, USHORT* pOct1, USHORT* pOct2, USHORT* pOct3, USHORT* pOct4, USHORT* pOct5, USHORT* pOct6, USHORT* pOct7, USHORT* pSamCnt);**

Axis: Number of the axis (0 to 15)

pOct0: Pointer to a variable in which the octant counter 0 is saved

pOct1: Pointer to a variable in which the octant counter 1 is saved

pOct2: Pointer to a variable in which the octant counter 2 is saved

pOct3: Pointer to a variable in which the octant counter 3 is saved

pOct4: Pointer to a variable in which the octant counter 4 is saved

pOct5: Pointer to a variable in which the octant counter 5 is saved

pOct6: Pointer to a variable in which the octant counter 6 is saved

pOct7: Pointer to a variable in which the octant counter 7 is saved

pSamCnt: Pointer to a variable in which the current number of measuring points is saved for determining the compensation value.

IK220ChkSumPar

Reports the current check sum of the parameters.

Prototype: **BOOL IK220ChkSumPar (USHORT Axis, USHORT* pChkSum);**

Axis: Number of the axis (0 to 15)

pChkSum: Pointer to a variable in which the momentary checksum of the parameters is saved

IK220ChkSumPrg

Reports the check sum of the IK 220 firmware.

Prototype: **BOOL IK220ChkSumPrg (USHORT Axis, USHORT* pChkSum1, USHORT* pChkSum2);**

Axis: Number of the axis (0 to 15)

pChkSum1: Pointer to a variable in which the actual checksum of the firmware is saved

pChkSum2: Pointer to a variable in which the nominal checksum of the firmware is saved

IK220WritePar

Changes a parameter of the IK 220.

Prototype: **BOOL IK220WritePar (USHORT Axis, USHORT ParNum, ULONG ParVal);**

Axis: Number of the axis (0 to 15)
ParNum: Parameter number
ParVal: Parameter value

IK220ReadPar

Supplies the value of an IK 220 parameter.

Prototype: **BOOL IK220ReadPar (USHORT Axis, USHORT ParNum, ULONG* pParVal);**

Axis: Number of the axis (0 to 15)
ParNum: Parameter number
pParVal: Pointer to a variable in which the parameter value is saved

IK220ResetEn

Resets the connected EnDat encoder to its default status.

Prototype: **BOOL IK220ResetEn (USHORT Axis, USHORT* pStatus);**

Axis: Number of the axis (0 to 15)
pStatus: Pointer to a variable in which the EnDat status is saved.
0 = OK
1 = Encoder does not answer or is not connected
2 = Transmission error
3 = Error mode echo
4 = Error CRC sum
5 = Error data echo
6 = Error MRS code / address echo

IK220ConfigEn

Reads the configuration of the connected EnDat encoder. The exact meaning of the individual value is described in the EnDat Description. With the IK220ReadMemEn function, the parameters of the encoder manufacturer can be read out in order to receive further information on the encoder. **This function must be called before further EnDat functions can be used!**

Prototype: BOOL IK220ConfigEn (USHORT Axis, USHORT* pStatus, USHORT* pType, ULONG* pPeriod, ULONG* pStep, USHORT* pTurns, USHORT* pRefDist, USHORT* pCntDir);

Axis: Number of the axis (0 to 15)

pStatus: Pointer to a variable in which the EnDat status is saved.

Low byte:
0 = OK
1 = Encoder does not answer or no encoder connected
2 = Transmission error
3 = Error mode echo
4 = Error CRC sum
5 = Error data echo
6 = Error MRS code / address echo

High byte:
0 = OK
1 = Error reading init-parameter MRS code 0xA1
2 = Error reading bits per position
3 = Error reading encoder type
4 = Error reading low-word signal period
5 = Error reading init-parameter MRS code 0xA3
6 = Error reading high-word signal period
7 = Error reading max. distinguishable revolutions
8 = Error reading init-parameter MRS code 0xA5
9 = Error reading supported alarms
10 = Error reading supported warnings
11 = Error reading nominal increment distance-coded REF
12 = Error reading low-word measuring step
13 = Error reading high-word measuring step
14 = Error reading measuring direction

pType: Pointer to a variable in which the encoder type is saved.

pPeriod: Pointer to a variable in which the signal period of the incremental signals or the number of lines per revolution is saved.

pStep: Pointer to a variable in which the measuring step of the EnDat position value or the number of measuring steps per revolution is saved.

pTurns: Pointer to a variable in which the number of resolvable revolutions is saved.

pRefDist: Pointer to a variable in which the nominal increment of distance-coded REF marks is saved

pCntDir: Pointer to a variable in which the measuring direction is saved.

IK220ReadEn

Reports the absolute counter value of the connected EnDat encoder. The EnDat counter value has the same significance as the incremental value, i.e., 1.0 represents one signal period!

Prototype: BOOL IK220ReadEn (USHORT Axis, USHORT* pStatus, double* pData, USHORT* pAlarm);

Axis: Number of the axis (0 to 15)

pStatus: Pointer to a variable in which the EnDat status is saved.
0 = OK
1 = Encoder does not answer or no encoder connected
2 = Error CRC sum
3 = Error type A

pData: Pointer to a variable in which the counter value of the EnDat encoder is saved

pAlarm: Pointer to a variable in which the alarm bit is saved
0 = OK
1 = Alarm

IK220ReadEnInc

Reports the absolute and incremental counter value of the connected EnDat encoder. The EnDat counter value has the same significance as the incremental value, i.e., 1.0 represents one signal period!

Prototype: **BOOL IK220ReadEnInc (USHORT Axis, USHORT Latch, USHORT* pStatus, double* pDataEn, USHORT* pAlarm, double* pDataInc);**

Axis: Number of the axis (0 to 15)

Latch: 0 = Read out incremental values via register 0
1 = Read out incremental value via register 1

pStatus: Pointer to a variable in which the EnDat status is saved.
0 = OK
1 = Encoder does not answer or no encoder connected
2 = Error CRC sum
3 = Error type A

pDataEn: Pointer to a variable in which the absolute counter value of the EnDat encoder is saved

pAlarm: Pointer to a variable in which the alarm bit is saved
0 = OK
1 = Alarm

pDataInc: Pointer to a variable in which the incremental counter value of the EnDat encoder is saved

IK220ModeEnCont

Starts and stops the continuous EnDat clock. With a continuous EnDat clock, new EnDat counter values are continuously called, and incremental values are acquired in synchronism. The counter values can be read out with IK220ReadEnIncCont. Other functions cannot be used in this mode of operation. If the continuous EnDat clock is started without CRC check, the checksum will not be checked after data transmission. This shortens the latching time.

Prototype: **BOOLIK220ModeEnCont (USHORT Axis, USHORT Latch, USHORT Mode, SHORT* pStatus)**

Axis: Number of the axis (0 to 15)

Latch: 0 = Output incremental value via register 0.

Mode: 0 = End readout with continuous clock
1 = Start readout with CRC check and continuous clock
2 = Start readout with continuous clock and without CRC check.
pStatus: 0 = OK
1 = Encoder does not answer or no encoder connected

IK220ReadEnIncCont
Returns the absolute and incremental position value of the connected EnDat encoder while the counter values of the EnDat are read out continuously with continuous clock. Before this function is used, the continuous EnDat clock must be started with IK220ModEnCont. The EnDat counter value has the same significance as the incremental value, i.e., 1.0 represents one signal period.

Prototype: BOOLIK220ReadEnIncCont (USHORT Axis, USHORT* pStatus, double* pDataEn, USHORT* pAlarm, double* pDataInc, USHORT* pSigStat)

Axis: Number of the axis (0 to 15)
pStatus: Pointer to a variable in which the EnDat status is saved.
0: OK
Bit 0=1: Encoder does not answer or no encoder connected.
Bit 1=1: Error CRC sum
Bit 2=1: Error type A
pDataEn: Pointer to a variable in which the absolute counter value of the EnDat encoder is saved
pAlarm: Pointer to a variable in which the alarm bit is saved
0 = OK
1 = Alarm
pDataInc: Pointer to a variable in which the incremental counter value of the EnDat encoder is saved
pSigStat: Pointer to a variable in which the amplitude status of the incremental signals is saved.
Bit 0/1: Status minimum amplitude
Bit 2/3: Status present amplitude
Bit 4/5: Status maximum amplitude
(Amplitude status: 00=normal / 01=low / 10=high / 11=too low)

IK220AlarmEn

Supplies the alarm word of the EnDat encoder and cancels all active alarms.

Prototype: **BOOL IK220AlarmEn (USHORT Axis, USHORT* pStatus, USHORT* pAlarm);**

Axis: Number of the axis (0 to 15)

pStatus: Pointer to a variable in which the status is saved.

Low byte: 0 = OK
1 = Encoder does not answer or no encoder connected
2 = Transmission error
3 = Error mode echo
4 = Error CRC sum
5 = Error data echo
6 = Error MRS code / address echo

High byte: 0 = OK
1 = Error reading init-parameter MRS code 0xB9
2 = Error reading/writing alarm word
3 = RESET error on encoder

pAlarm: Pointer to a variable in which the alarm word is saved.

IK220WarnEn

Supplies the warning word of the EnDat encoder and cancels all active warnings.

Prototype: **BOOL IK220WarnEn (USHORT Axis, USHORT* pStatus, USHORT* pWarn);**

Axis: Number of the axis (0 to 15)

pStatus: Low byte: 0 = OK
1 = Encoder does not answer or no encoder connected
2 = Transmission error
3 = Error mode echo
4 = Error CRC sum
5 = Error data echo
6 = Error MRS code / address echo

High byte: 0 = OK
1 = Error reading init-parameter MRS code 0xB9
2 = Error reading/writing warning word
3 = RESET error on encoder

pWarn: Pointer to a variable in which the warning word is saved.

IK220ReadMemEn

Reads values from the memory range of the EnDat encoder.

Prototype: BOOL IK220ReadMemEn (USHORT Axis, USHORT Range, USHORT MemAdr, USHORT* pMemData, USHORT* pStatus)

Axis: Number of the axis (0 to 15)

Range: Selection of memory area
- 0: Operating status
- 1: Parameters of the encoder manufacturer
- 2: Operating parameters
- 3: OEM parameters
- 4: Compensation values

MemAdr: Word address in the selected range

pMemData: Pointer to a variable in which the value is saved.

pStatus: Pointer to a variable in which the EnDat status is saved.
- 0 = OK
- 1 = Encoder does not answer or no encoder connected
- 2 = Transmission error
- 3 = Error mode echo
- 4 = Error CRC sum
- 5 = Error data echo
- 6 = Error MRS code / address echo

IK220WriteMemEn

Writes values to the OEM parameter memory of the EnDat encoder. The meanings of the values are defined by the OEM.

Prototype: BOOL IK220WriteMemEn (USHORT Axis, USHORT Range, USHORT MemAdr, USHORT MemData, USHORT* pStatus)

Axis: Number of the axis (0 to 15)

Range: Selection of memory area
- 0: Operating status
- 1: Parameters of the encoder manufacturer
- 2: Operating parameters
- 3: OEM parameters
- 4: Compensation values

MemAdr: Memory address in the selected range

MemData: Value to be written.

pStatus: Pointer to a variable in which the EnDat status is saved.
0 = OK
1 = Encoder does not answer or no encoder connected
2 = Transmission error
3 = Error mode echo
4 = Error CRC sum
5 = Error data echo
6 = Error MRS code / address echo

IK220ReadSSI

Returns the absolute counter value of the connected SSI encoder. The transmission parameters of the SSI encoder must first be specified in the parameters.

Prototype: BOOL IK220ReadSSI (USHORT Axis, USHORT* pStatus, double* pData);

Axis: Number of the axis (0 to 15)

pStatus: Pointer to a variable in which the SSI status is saved
0 = OK
1 = Encoder does not answer or no encoder connected
2 = Parity error
3 = Parity error, pre-zero bit not equal to zero

pData: Pointer to a variable in which the counter value of the SSI encoder is saved

IK220ReadSsiInc

Reports the absolute and incremental counter value of the connected SSI encoder. The transmission parameters of the SSI encoder must first be specified in the parameters.

Prototype: BOOL IK220ReadSsiInc (USHORT Axis, USHORT Latch, USHORT* pStatus, double* pDataSsi, double* pDataInc)

Axis: Number of the axis (0 to 15)

Latch: 0 = Read out incremental values via register 0
1 = Read out incremental value via register 1

pStatus: Pointer to a variable in which the SSI status is saved
0 = OK
1 = Encoder does not answer or no encoder connected
2 = Parity error
3 = Parity error, pre-zero bit not equal to zero

pDataSsi: Pointer to a variable in which the absolute counter value of the SSI encoder is saved
pDataInc: Pointer to a variable in which the incremental counter value of the SSI encoder is saved

IK220SetTimer
Programs the timer with the value in "SetVal" or the next largest possible value. The time is defined by the parameter for the time interval of the timer and through the parameter for the software divider.

Prototype: BOOL IK220SetTimer (USHORT Axis, ULONG SetVal, ULONG* pTimVal);

Axis: Number of the axis (0 to 15)
SetVal: Required timer value in micro seconds.
pTimVal: Pointer to a variable in which the actually programmed timer value is saved in micro seconds.

IK220ModeTimer
Specifies whether the timer signal is output. To be able to save the axes of a card or several cards through an external connection, the signal generated by the timer must be output. The duration period of the timer signal depends only on the time interval of the timer. The value programmed in the software timer has no influence on this.

Prototype: BOOL IK220ModeTimer (USHORT Axis, USHORT Mode);

Axis: Number of the axis (0 to 15)
Mode: 0 = Timer signals are not output
1 = Timer signals are output

IK220ModeRam
Stored counter values can be transferred to a buffer memory on the IK 220. The values saved can then be read out with IK220GetRam or IK220BurstRam.

Prototype: BOOL IK220ModeRam (USHORT Axis, USHORT Mode);

Axis: Number of the axis (0 to 15)
Mode: 0 = Latched counter values are not transferred
1 = Latched counter values from register 0 are transferred[1)]
2 = Latched counter values from register 1 are transferred
3 = Latched counter values from register 0 are transferred until the max. number is reached (single shot)

[1)] Circular buffer function

4 = Latched counter values from register 1 are transferred until the max. number is reached (single shot)

IK220ResetRam

The write and read pointer of the RAM buffer is set to 0. All of the values in the RAM buffer are canceled.

Prototype: **BOOL IK220ResetRam (USHORT Axis);**

Axis: Number of the axis (0 to 15)

IK220GetRam

A counter value previously stored in the RAM buffer is output. After the value has been read, the offset of the read pointer is increased until all of the values are output.

Prototype: **BOOL IK220GetRam (USHORT Axis, double* pData, USHORT* pRead, USHORT* pWrite, USHORT* pStatus);**

Axis: Number of the axis (0 to 15)

pData: Pointer to a variable in which the counter value is saved.

pRead: Pointer to a variable in which the offset of the writing pointer is saved in the RAM buffer.

pWrite: Pointer to a variable in which the offset of the reading pointer is saved in the RAM buffer.

pStatus: Status of the RAM buffer.
Bit 0=1: Buffer overflow
Bit 1=1: No value in the buffer
Bit 2=1: Last value is read from the buffer

IK220BurstRam

Counter values previously stored in the RAM buffer are output. The read pointer is then increased by the number of read values.

Prototype: **BOOL IK220BurstRam (USHORT Axis, USHORT maxCount, double* pData, USHORT* pCount, USHORT* pStatus)**

Axis: Number of the axis (0 to 15)

maxCount: Maximum number of values that are read during latching.

pData: Pointer to an array of "double variables" (64 bits) in which the counter values are saved. Space must be reserved for maxCount counter values!

pCount: Pointer to a variable in which the actual number of read counter values is saved.

pStatus: Status of the RAM buffer.
Bit 0=1: Buffer overflow
Bit 1=1: No value in the buffer
Bit 2=1: Last value is read from the buffer
Bit 15=1: Error while reading buffer

IK220GetSig

An amplitude-value pair stored in the RAM buffer is output. The read counter is increased after reading.

Prototype: BOOL IK220GetSig (USHORT Axis, USHORT* pPeriod, SHORT* pAmp0, SHORT* pAmp90, USHORT* pRead, USHORT* pWrite, USHORT* pStatus)

Axis: Number of the axis (0 to 15)

pPeriod: Pointer to a variable in which the lower 16 bits of the signal-period counter are saved

pAmp0: Pointer to a variable in which the 0°-amplitude value is saved

pAmp90: Pointer to a variable in which the 90°-amplitude value is saved

pRead: Pointer to a variable in which the offset of the writing pointer is saved in the RAM buffer.

pWrite: Pointer to a variable in which the offset of the reading pointer is saved in the RAM buffer.

pStatus: Status of the RAM buffer.
Bit 0=1: Buffer overflow
Bit 1=1: No value in the buffer
Bit 2=1: Last value is read from the buffer

IK220BurstSig

Amplitude-value pairs previously stored in the RAM buffer are output. The read pointer is then increased by the number of read values.

Prototype: BOOL IK220BurstSig (USHORT Axis, USHORT maxCount, USHORT* pPeriod, SHORT* pAmp0, SHORT* pAmp90, USHORT* pCount, USHORT* pStatus)

Axis: Number of the axis (0 to 15)

maxCount: Maximum number of value pairs that are read during latching

pPeriod: Pointer to an array of variables in which the signal-period-counter values are saved Space must be reserved for maxCount values!

pAmp0: Pointer to an array of variables in which the 0° amplitude values are saved. Space must be reserved for maxCount values!

pAmp90: Pointer to an array of variables in which the 90° amplitude values are saved. Space must be reserved for maxCount values!
pCount: Pointer to a variable in which the actual number of read value pairs is saved.
pWrite: Pointer to a variable in which the offset of the reading pointer is saved in the RAM buffer.
pStatus: Status of the RAM buffer.
Bit 0=1: Buffer overflow
Bit 1=1: No value in the buffer
Bit 2=1: Last value is read from the buffer
Bit 15=1: Error while reading buffer

IK220Led
Defines the function of the axis LED on the IK 220.
Prototype: BOOL IK220Led (USHORT Axis, USHORT Mode);
Axis: Number of the axis (0 to 15)
Mode: 0 = LED does not light up
1 = LED lights up
2 = LED flashes

IK220SysLed
Defines the function of the system LED on the IK 220.
Prototype: BOOL IK220SysLed (USHORT Card, USHORT Mode);
Card: Number of the card (0 to 7)
Mode: 0 = LED does not light up
1 = LED lights up

IK220GetPort
Reports the status of the IK 220 inputs.
Prototype: BOOL IK220GetPort (USHORT Axis, USHORT* pPortInfo, USHORT* pRising, USHORT* pFalling);
Axis: Number of the axis (0 to 15)
pPortInfo: Pointer to a variable in which the current status of the inputs is saved.
pRising: Pointer to a variable showing whether "set edges" occurred since the last output.
pFalling: Pointer to a variable showing whether "reset edges" occurred since the last output.

IK220RefEval

Defines the type of evaluation of the reference-mark signal.

Prototype: BOOL IK220RefEval (USHORT Axis, USHORT Mode);

Axis: Number of the axis

Mode: 0 = REF signal masked with incremental signals
1 = REF signal not masked with incremental signals

IK220SetBw

Defines the input frequency for the sinusoidal incremental signals. Switchable from 11 μA_{PP} to 1 V_{PP} by parameter 2 (or vice versa) so that the standard setting is active again (33 kHz for 11 μA_{PP}, and 500 kHz for 1 V_{PP}). If you want to use another input frequency, you must define it with IK220SetBw.

Prototype: BOOL IK220SetBw (USHORT Axis, USHORT Mode);

Axis: Number of the axis

Mode: 0 = High input frequency (175 kHz for 11 μA_{PP}, 500 kHz for 1 V_{PP})
1 = Low input frequency (33 kHz for 11 μA_{PP} and 1 V_{PP})

The following functions are used by the driver software. They should not be used in application programs.

IK220InputW

Reads a 16-bit value from the given address of the axis.

Prototype: BOOL IK220InputW (USHORT Axis, USHORT Adr, USHORT* pData)

Axis: Number of the axis (0 to 15)

Adr: Address of the registers (0 to 15 or 0 to 0x0F)

pData: Pointer to a variable in which the read value is saved.

IK220InputL

Reads a 32-bit value from the given address of the axis.

Prototype: BOOL IK220InputL (USHORT Axis, USHORT Adr, ULONG* pData)

Axis: Number of the axis (0 to 15)

Adr: Address of the registers (0 to 14 or 0 to 0x0E)

pData: Pointer to a variable in which the read value is saved.

IK220Output

Writes a 16-bit value to the given address of the axis.

Prototype: **BOOL IK220Output (USHORT Axis, USHORT Adr, USHORT Data)**

Axis:	Number of the axis (0 to 15)
Adr:	Address of the registers (0 to 15 or 0 to 0x0F)
Data:	Value that is written to the register

IK220RamRead

Reads a 16-bit value from the RAM of the IK 220.

Prototype: **BOOL IK220RamRead (USHORT Axis, USHORT Adr, USHORT* pData)**

Axis:	Number of the axis (0 to 15)
Adr:	Address in RAM (0 to 65535)
pData:	Pointer to a variable in which the value is saved.

IK220RamWrite

Writes a 16-bit value to the RAM of the IK 220.

Prototype: **BOOL IK220RamWrite (USHORT Axis, USHORT Adr, USHORT Data)**

Axis:	Number of the axis (0 to 15)
Adr:	Address in RAM (0 to 65535)
Data:	Value (16-bit) that is written to RAM.

IK220DownLoad

Loads the firmware of the IK 220 into RAM.

Prototype: **BOOL IK220DownLoad (USHORT Axis, USHORT* pPgmData, ULONG PgmSize)**

Axis:	Number of the axis (0 to 15)
pPgmData	Pointer to an array in which the program information is saved
PgmSize:	Number of bytes in the PgmData array

IK220SetEnClock

Sets the clock line of the EnDat interface.

Prototype: **BOOL IK220SetEnClock (USHORT Axis, BOOL State, USHORT* pStatus)**

Axis:	Number of the axis (0 to 15)
State:	False (=0): Set clock line to low level. True (≠0): Set clock line to high level.
pStatus:	0 = OK 1 = Error

IK220SetEnData

Sets the data line of the EnDat interface.

Prototype: BOOL IK220SetEnData (USHORT Axis, BOOL State, USHORT* pStatus)

Axis: Number of the axis (0 to 15)

State: False (=0): Set data line to low level.
True (≠0): Set data line to high level.

pStatus: 0 = OK
1 = Error

IK220ReadEnData

Reads the condition of the data line on the EnDat interface.

Prototype: BOOL IK220ReadEnData (USHORT Axis, BOOL* pState)

Axis: Number of the axis (0 to 15)

pState: Pointer to a variable in which the level of the EnDat data line is saved
False (=0): Data line is on low level.
True (≠0): Data line is on high level.

Specifications

Mechanical Data	
Dimensions	Approx. 190 mm x 100 mm
Operating temperature	0° C to 55° C (32 °F to 131 °F)
Storage temperature	–30° C to 70° C (–22 °F to 158 °F)

Electrical data	
Encoder inputs	Two D-sub connections (15-pin male) switchable: • ∿ **11 µA$_{PP}$:** Sinusoidal current signals Input frequency: max. 33 kHz Cable length: max. 60 m[1)] • ∿ **1 V$_{PP}$:** Sinusoidal voltage signals Input frequency: max. 500 kHz Cable length: max. 60 m[1)] • **EnDat** Cable length: max. 50 m[1)] • **SSI** Cable length: max. 10 m[1)]
External latch signals (Option) 2 inputs 1 output	Assembly with two D-sub connections (9-pin, male) TTL levels TTL levels
Encoder outputs (Option)	Sinusoidal current signals (11 µA$_{PP}$) Assembly with two D-sub connections (9-pin, male)
Signal subdivision	Up to 4096-fold

[1)] With genuine HEIDENHAIN cables, please note the supply voltage for the encoder.

Adjustment of encoder signals	Adjustment of offset, phase and amplitude by software — also online
Data register for measured values	48 bits, where 44 bits are used for the measured value
Interface	PCI bus (plug and play)
Internal memory	For 8191 position values
Power consumption	Approx. 4 W, without encoder

Software	
Driver software and demonstration program[2)]	For Windows NT/95/98/2000/XP Linux Kernel 2.4 LabView 7.1 in Visual C++, Visual Basic and Borland Delphi

2) The content of the CD as well as the drivers for other operating systems (e. g. Windows Vista, Windows 7) are available in the download area at www.heidenhain.de.

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 32-1000
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 8669 31-3105
E-mail: service.lathe-support@heidenhain.de

www.heidenhain.de

785 183-90 · Ver00 · 15 · 5/2011 · H · Printed in Germany