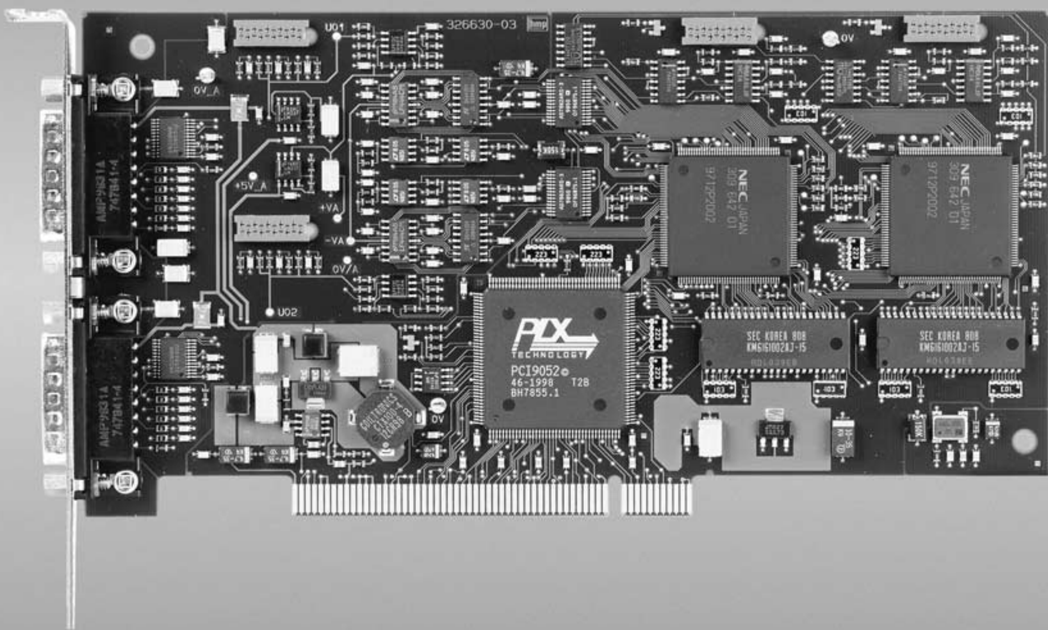


HEIDENHAIN

User's Manual

IK 220

PC Counter Card for
HEIDENHAIN encoders

7/2006

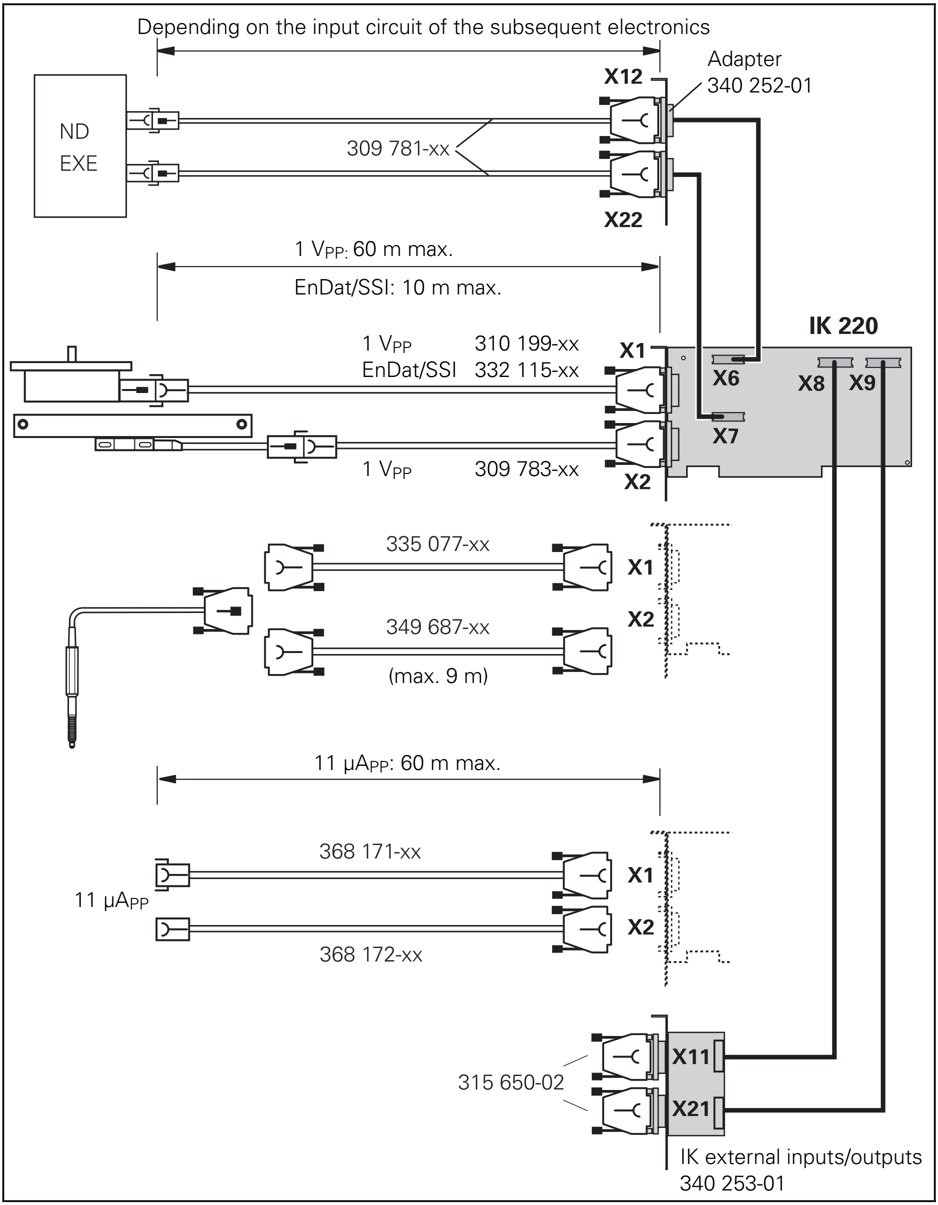
Content

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
X2
349 687-xx
(max. 9 m)
11 µAPP: 60 m max.
368 171-xx
X1
11 µAPP
X2
368 172-xx
315 650-02
X11
X21
IK external inputs/outputs
340 253-01

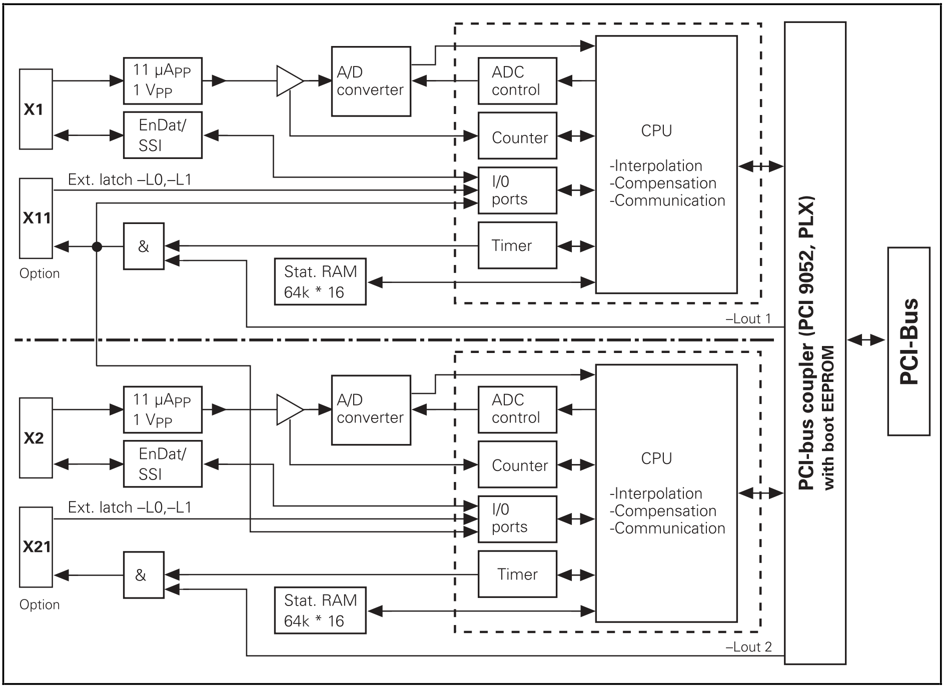
Important Information

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

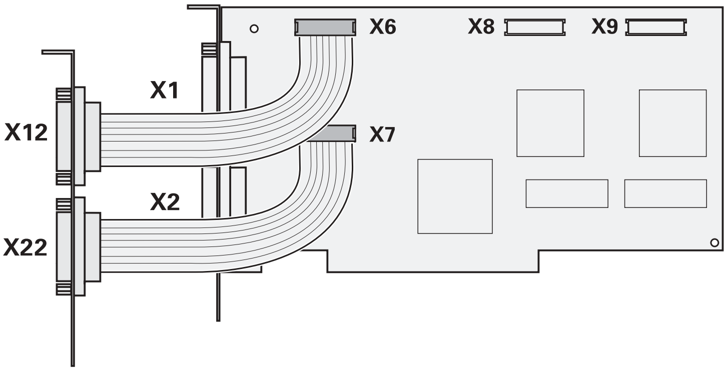
D-sub connection with male contacts (15-pin)

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).

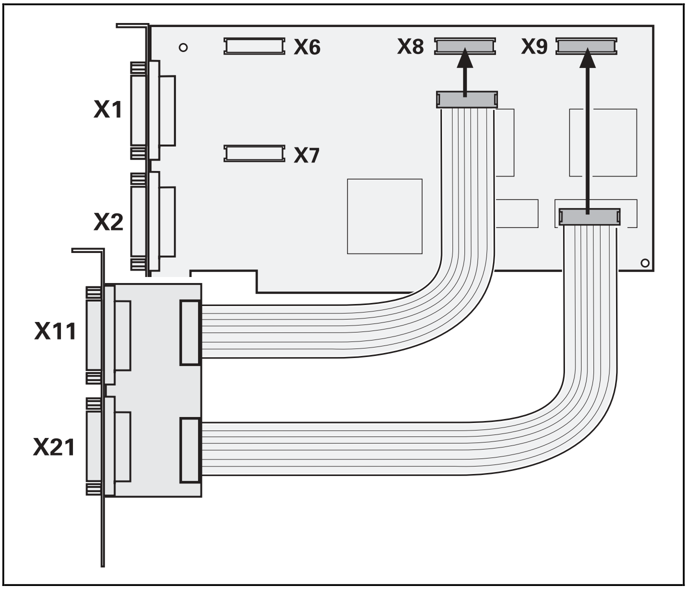
Connections X8, X9 for external inputs/outputs

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.

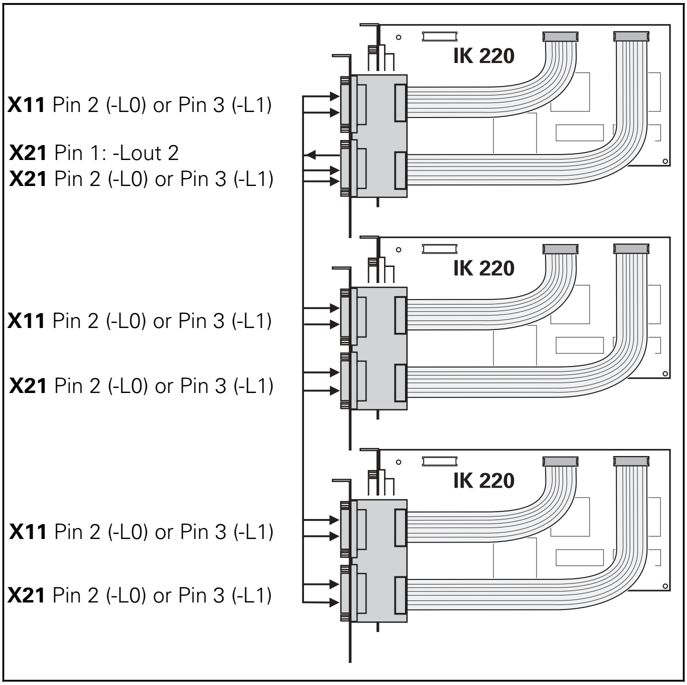
The simplest way to activate the inputs is to make a bridge from the 0-volt connection (terminals 8, 9) to the input for latching.

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.

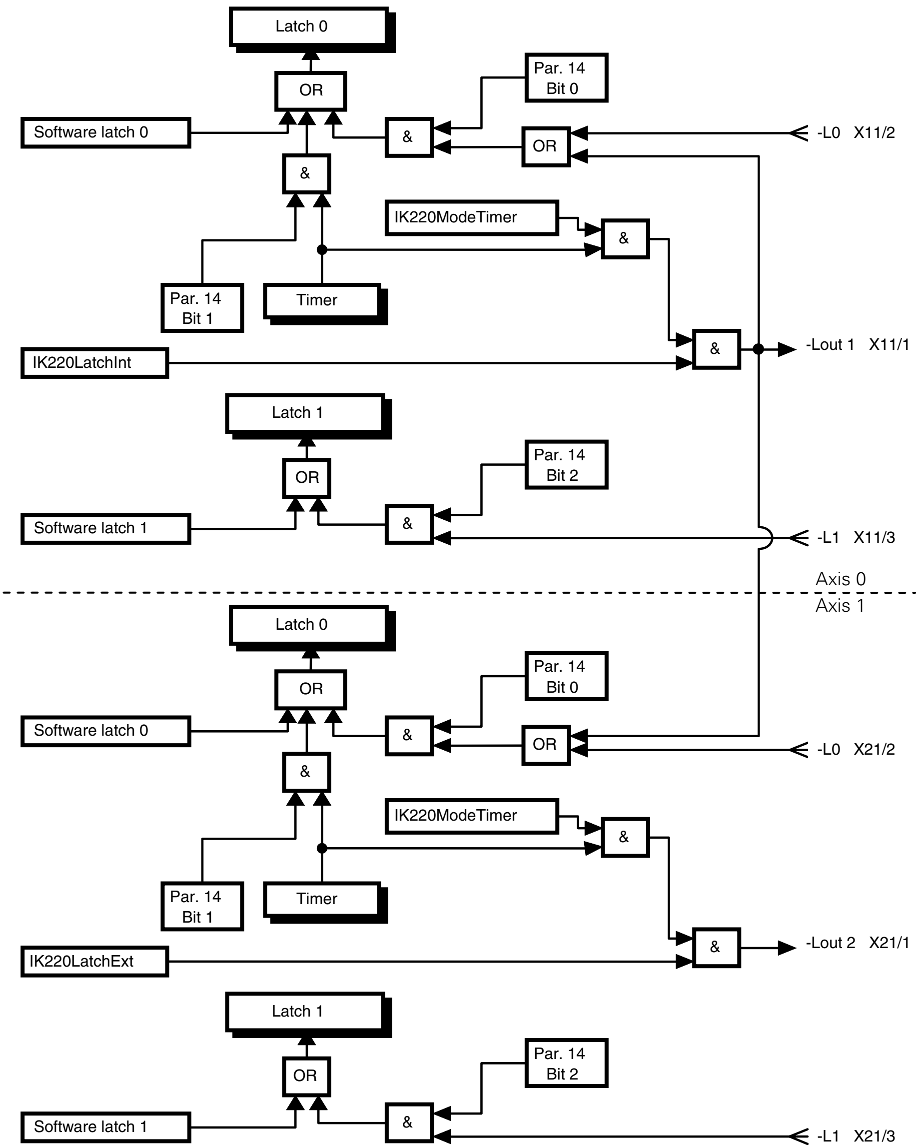
To latch the measured values of different IKs at the same time (IK220LatchExt), you must use -Lout 2 (see next page).

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)] 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** 500: Distance-coded REF marks, Fixed interval 500 x GP 1000: Distance-coded REF marks, Fixed interval 1000 x GP 2000: Distance-coded REF marks, Fixed spacing 1000 x GP 5000: Distance-coded REF marks, fixed spacing 5000 x GP

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** [2)
12	16 bits	Software divider for timer, **Default value 1** [2)
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) 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 milli-seconds, 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 supplied with the card or can be downloaded from the IK 220 directory at www.heidenhain.de.

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 DDL 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 ($\neq$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 µAPP to 1 VPP by parameter 2 (or vice versa) so that the standard setting is active again (33 kHz for 11 µAPP, and 500 kHz for 1 VPP). 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 µAPP, 500 kHz for 1 VPP)
1 = Low input frequency (33 kHz for 11 µAPP and 1 VPP)

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 **Storage temperature**	0° C to 55° C (32 °F to 131 °F) –30° C to 70° C (–22 °F to 158 °F)

Electrical data	
Encoder inputs	Two D-sub connections (15-pin male) switchable: • **~ 11 µAPP:** Sinusoidal current signals Input frequency: max. 33 kHz Cable length: max. 60 m[1] • **~ 1 VPP:** 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 µAPP) 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	For Windows NT/95/98/2000/XP Linux Kernel 2.4 LabView 7.1 in Visual C++, Visual Basic and Borland Delphi

HEIDENHAIN

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

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

www.heidenhain.de

Ve 03
339 904-26 · 10 · 7/2006 · S · Printed in Germany · Subject to change without notice