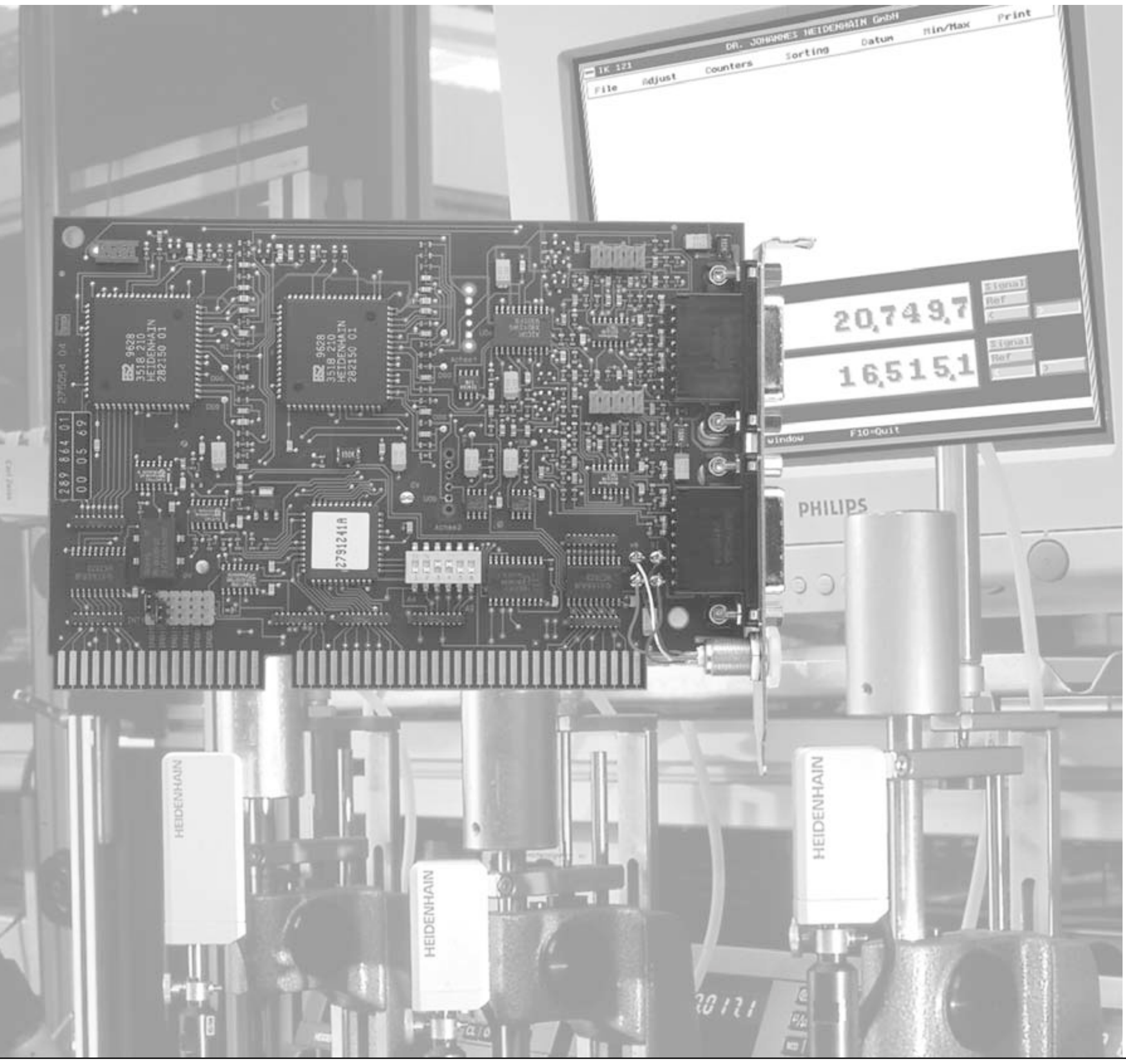


HEIDENHAIN

User's Manual IK 121

PC Counter Card for HEIDENHAIN Encoders

5/2000

Contents

Items Supplied

IK 121 counter card for PCs, programming examples, driver software and User's Manual

Versions

IK 121 A

290 155-xx	IK 121 counter card for PCs with encoder inputs for sinusoidal current signals (11 μA_{PP}).

IK 121 V

291 768-xx	IK 121 counter card for PCs with encoder inputs for sinusoidal voltage signals (1 V_{PP}).

Note on the EMC Guideline 89/336/EWG

Compliance with EMC Guideline 89/336/EWG was tested with a COMPAQ DESKPRO 386/20e computer.

Accessories

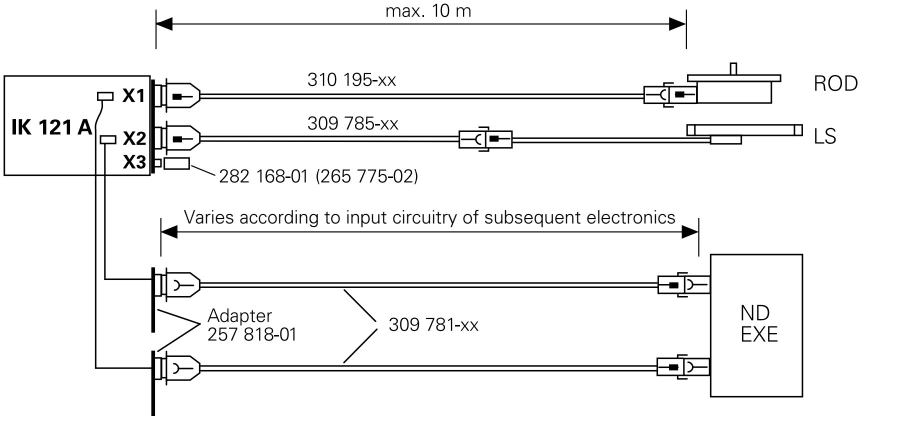
257 818-01	Additional D-sub connection for sending the encoder signals of input X1 or X2 to another display unit or control
309 781-xx	Connecting cables from additional D-sub connection to another display unit or control
282 168-01 (265 775-02)	Connectors for the external functions on input input X3 (two female, two male)

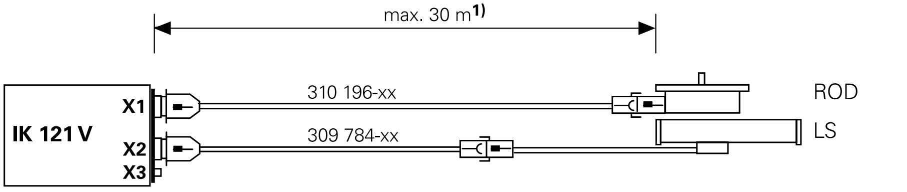
IK 121 A

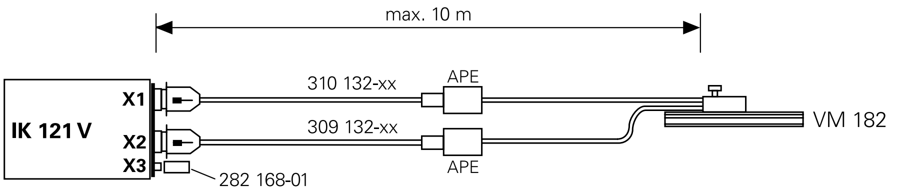
309 785-xx 9/9-pin	Adapter cables with coupling for HEIDENHAIN encoders; standard length 0.5 meters
310 195-xx 9/9-pin	Adapter cables with connector for HEIDENHAIN encoders with flange socket; standard length 0.5 meters

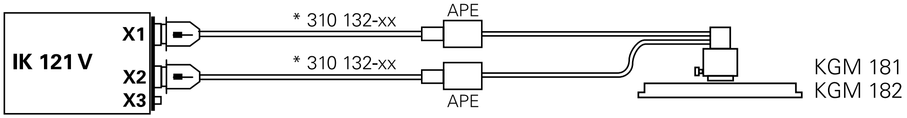
IK 121 V

309 784-xx 15/12-pin	Adapter cables with coupling for HEIDENHAIN encoders; standard length 0.5 meters
310 196-xx 15/12-pin	Adapter cables with connector for HEIDENHAIN encoders with flange socket; standard length 0.5 meters









1) Cable lengths up to 150 m (are possible, if it can be guaranteed that the encoder will be supplied by 5 V from an external power source.

Important Information

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

READ.ME

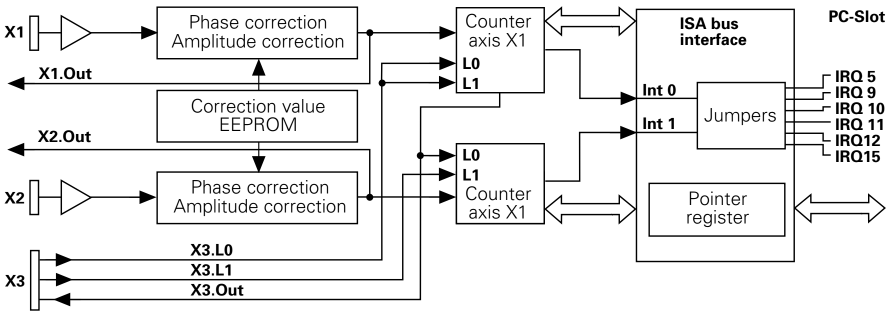
The READ.ME file contains important information on installation of the IK 121 and the supplied software. The program README.EXE displays this file on the screen.

Technical Description of the IK 121

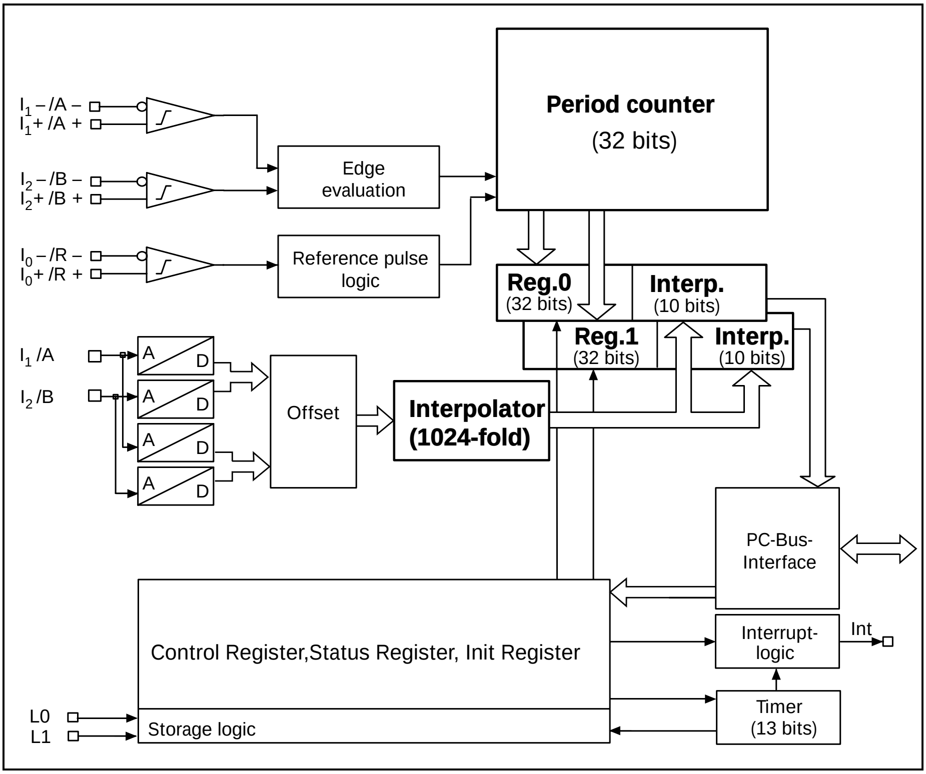
The IK 121 counter card is plugged directly into an expansion slot of an AT-compatible PC. The IK 121 accepts the signals from one or two HEIDENHAIN encoders with sinusoidal current signals (IK 121 A) or voltage signals (IK 121 V). The positions of the encoders are displayed on the PC screen and stored in the PC for further processing.

The IK 121 is ideal for applications in high-speed data acquisition requiring high resolution of the encoder signals.

Circuit diagram of the IK 121



Circuit diagram of the counter IC



The IK 121's interpolation circuitry subdivides the signal period of the input signal 1024-fold. The 42-bit measured value is formed from the interpolation value (10 bits) and the value of the period counter (32 bits). The measured values are stored in 48-bit 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 or with software or timers, as well as by crossing over the reference marks with port addressing.

Latching, as used in this manual, means that the count in Register 0 or 1 is retained (held). The count must then be read by the program and stored in the PC or displayed on the screen.

The phase and amplitude of the sinusoidal encoder signals can be adjusted with the software and electronic potentiometers; the offset can be adjusted with data registers in the counter.

Time to access measured values

The time required to access the measured values is approximately 35 µs.

Hardware

PC bus specification

The IK 121 can be installed in all IBM AT and 100% compatible PCs. HEIDENHAIN cannot guarantee proper functioning of the IK 121 with non-100% compatible PCs. The IK 121 conforms to the international IEEE P996 standard which specifies the AT and ISA bus (industry standard).

Bus width	**16 bits**
Power supply	+5 V ±5% +12 V ±5% –12 V ±5%
Power consumption	Approx. 1 watt without encoders
Slot specification	AT-compatible, short 16-bit slot

Encoder inputs IK 121 A

The IK 121 A accepts HEIDENHAIN linear or angle encoders that generate sinusoidal current signals I_1 and I_2. These encoders also generate a reference mark signal I_0.

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 messages	$\leq 2.5\ \mu A_{PP}$
Maximum input frequency	100 kHz
Cable length	Max. 10 m

Inputs X1 and X2 for encoders

D-sub connection with female insert (9-pin)

Connection no.	Assignment
1	I_1 –
2	0 V (U_N)
3	I_2 –
4	Internal shield
5	I_0 –
6	I_1 +
7	5 V (U_P)
8	I_2 +
9	I_0 +
Housing	External shield

Encoder inputs IK 121 V

The IK 121 V is connected with HEIDENHAIN linear or angle encoders with sinusoidal voltage signals A and B, plus the reference mark signal R.

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 messages	$\leq$ 0.22 V_{PP}
Maximum input frequency	400 kHz
Cable length[1)]	Max. 30 m

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

Inputs X1 and X2 for encoders

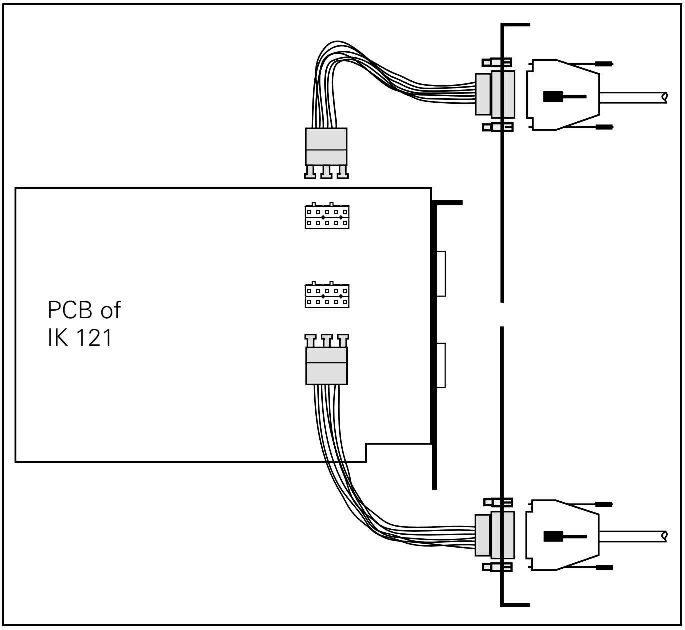
D-sub connection with female insert (15-pin)

Pin	Assignment
1	A +
2	0 V (U_N)
3	B +
4	+ 5 V (U_P)
5	*Do not use*
6	*Do not use*
7	R –
8	*Do not use*
9	A –
10	0 V (sensor line)
11	B –
12	+ 5 V (sensor line)
13	*Do not use*
14	R +
15	*Do not use*
Housing	External shield

Encoder outputs

The IK 121 also feeds the encoder signals from inputs X1 and X2 as **sinusoidal current signals** (11 μA_{PP}) to two 10-pin AMP connectors. These connections can be routed to 9-pin D-sub connections by means of additional cable assemblies with a PC slot cover (Id.-Nr. 257 818-01). Adapter cables (Id.-Nr. 309 78-xx) for connection to HEIDENHAIN position display units or interpolation electronics are available (see "Items Supplied, Accessories").

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



Encoder outputs (Id. Nr. 257 818-01)
D-sub connection with male insert (9-pin)

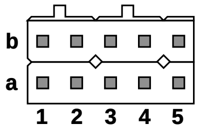
Pin	Assignment
1	I_1 –
2	0 V (U_N)
3	I_2 –
4	Not connected
5	I_0 –
6	I_1 +
7	Not connected
8	I_2 +
9	I_0 +
Housing	External shield

PCB connector for encoder outputs

AMP with male insert (10-pin)

Pin[*]	Signal
1a	Not connected
1b	Not connected
2a	Not connected
2b	0 V (U_N)
3a	I_0 –
3b	I_0 +
4a	I_2 –
4b	I_2 +
5a	I_1 –
5b	I_1 +

[*]The side with the locking pins is indicated as *b*.
Connections 1a and 1b are on the side with the notch.



Encoder signal adjustment

Encoder signals can be adjusted as follows:

- Phase and amplitude can be adjusted with electronic potentiometers
- Symmetry (offset) can be adjusted in the counters with offset registers

The potentiometer is I²C-bus-controlled. Since generation of the control sequences is complicated, the program POTIS.EXE or ADJUST.EXE should be used for adjustment.

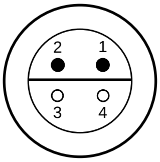
If an application does have to control the potentiometers, the functions and procedures in TURBO PASCAL from IIC.PAS can be used or they can serve as a basic orientation for your own functions.

The compensation values for phase and amplitude are stored in the ICs of the electronic potentiometers in nonvolatile memory. The offset registers in the counter ICs, however, are **volatile** (the information will be lost when the power is removed). For this reason, the offset compensation values are stored in an EEPROM in the IC for the electronic potentiometers. After switch-on, the offset compensation values must be loaded from the EEPROM into the offset registers of the counters. Two procedures in IIC.PAS fulfill these tasks. The store_offset procedure stores the offset compensation values in the EEPROM. The load_offset procedure copies the offset compensation values from the EEPROM into the offset register of the counters. Load_offset is also used by init_IK121.

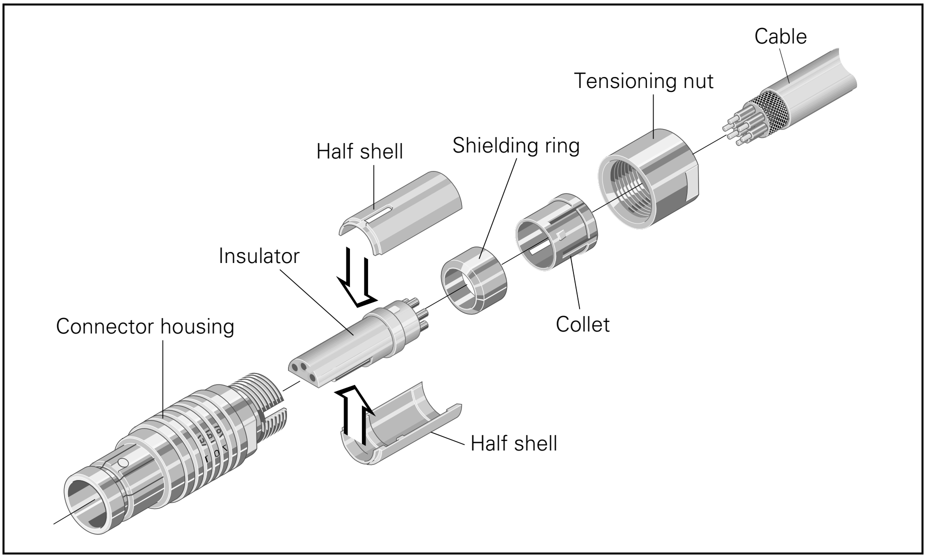
External functions

A 4-pin flange socket is available for external functions. The required connector (Id.-Nr. 282 168-01) is available from HEIDENHAIN.

Assembling the connector for the external functions



Rear view
o = female, • = male



Connection X3 for external functions

Flange socket with male/female insert (4-pin)

Connection no.	Assignment
1	Input: measured value latch X1 (X3.L0)
2	Input: measured value latch X2 (X3.L1)
3	Output: measured value latch (X3.Out)
4	0 V

Latching measured values via external inputs

The IK 121 has two external inputs on flange socket X3 for latching and storing measured values.

These inputs can also be used for interrupts.

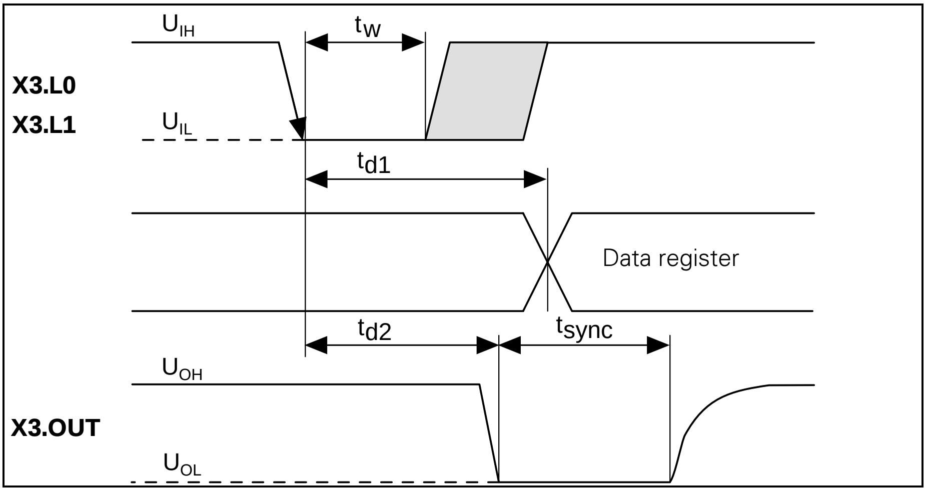
Inputs X3.L0 and X3.L1 are active low; they are kept at high level by a 10-kΩ internal pull-up resistor. They can be connected to TTL, LS or CMOS components.

Simplest way to activate the inputs: bridge from 0 volts (connection 4) to the input for latching.

Latching output X3.OUT
The output signal X3.OUT can be sent over flange socket X3 to further IK 121s (inputs X3.L0, X3.L1), for example to latch the measured values of multiple IKs.

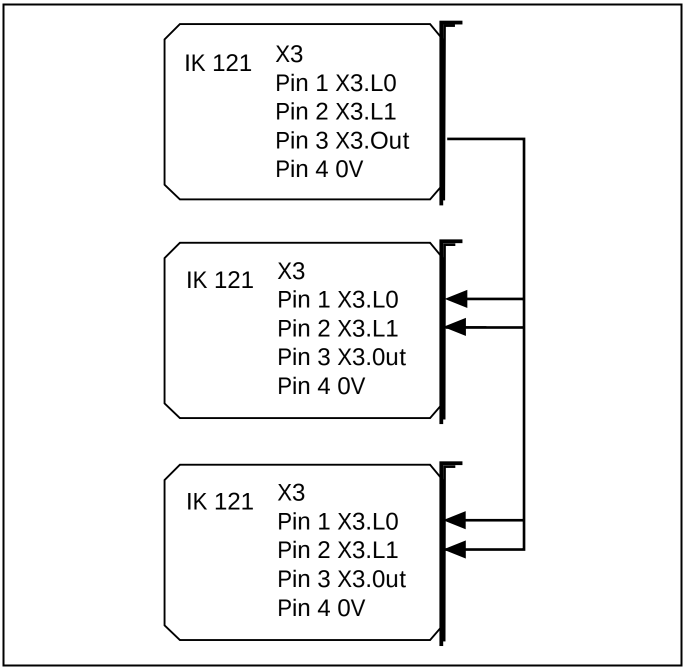
X3.OUT is an open collector output that switches to “Zero.”

Latching inputs X3.L0, X3.L1 and latching output X3.OUT: Time diagram and voltage level



Designation	MIN	MAX
U_{IL} (V)	– 3.0	0.9
U_{IH} (V)	3.15	30
t_{d1} (µs)	–	24
t_{d2} (ns)	–	500
t_W (ns)	250	–
t_{sync} (µs)	1	–
U_{OL} (V)	0	0.8 (U_{OH} = 4V – 12V) 1.0 (U_{OH} = 12V – 32V)
U_{OH} (V)	4	32
I_{OL} (mA)	–	40

Latching measured values from multiple IK 121s
Latching output X3.OUT can be used to latch the measured value from multiple IK 121 cards. This is done by connecting the latching output X3.OUT of one IK 121 to the latching inputs of further IK 121s.

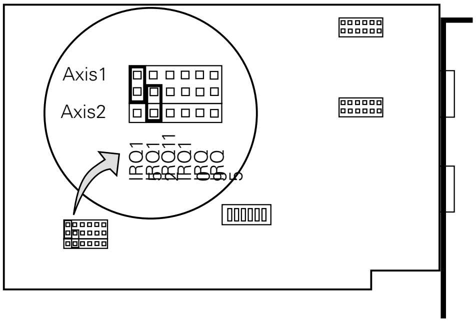


Interrupts

The IK 121 can use one of the following PC interrupts: IRQ5, IRQ9, IRQ10, IRQ11, IRQ12 or IRQ15.

The desired interrupt is selected with jumpers on the circuit board.

Axis 1 generates the internal signal **Int0** and axis 2 generates **Int1**. The arrangement of the pin connectors prevents **Int0** and **Int1** from being connected to the same IRQ line.



In the above setting, axis 1 is set to IRQ15 and axis 2 to IRQ12.

Assignment of the PC interrupts

Interrupt	Interrupt number	Interrupt address
IRQ5	0D	034 to 037
IRQ9	71	1C4 to 1C7
IRQ10	72	1C8 to 1CB
IRQ11	73	1CC to 1CF
IRQ12	74	1D0 to 1D3
IRQ15	77	1D7 to 1DF

Addressing

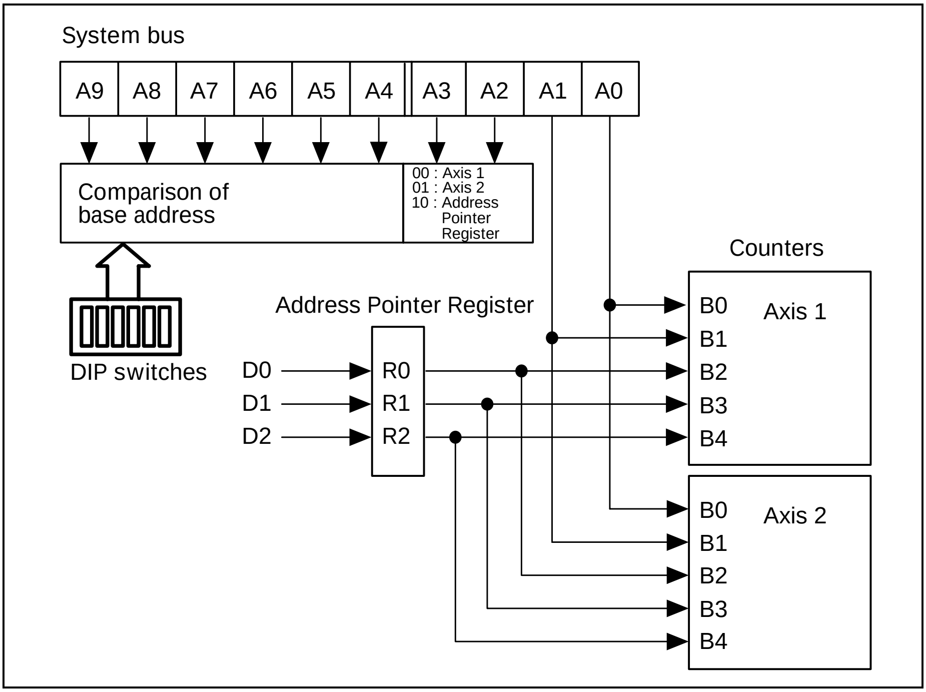
The IK 121 and the computer communicate over port addresses (I/O range) with a 16-bit data width. Since the vacant address space in the port address range is severely limited, the addresses of the counters were superimposed on each other. An Address Pointer Register is used to differentiate.

Each counter has five address lines B0 to B4. B0 and B1 are connected directly to the system bus. B2, B3 and B4 are generated in the Address Pointer Register. A2 and A3 of the system bus decode the **counters** for axes 1 or 2 or for the Address Pointer Register.

A4 to A9 of the system bus (base address) are set with the DIP switch on the circuit board. The "on" setting corresponds to logical zero.

Examples of switch settings

Address	**Switch setting**					
(Hex)	**A9**	**A8**	**A7**	**A6**	**A5**	**A4**
0110	**On**	**Off**	**On**	**On**	**On**	**Off**
0250	**Off**	**On**	**On**	**Off**	**On**	**Off**
0300	**Off**	**Off**	**On**	**On**	**On**	**On**
0330	**Off**	**Off**	**On**	**On**	**Off**	**Off**



Registers

The circuit diagram of the counters on the last fold-out page will help you in the following description.

Important note for programmers:
The IK 121 is accessed by reading and writing data words stored in registers. For this reason, only even port addresses may be addressed with word-write and word-read commands.

Overview of registers

Address (hex) B0 to B4	**Write mode**	**Read mode**
00 02 04	No function	Data Register 0, LS-Word Data Register 0 Data Register 0, MS-Word
06 08 0A	No function	Data Register 1, LS-Word Data Register 1 Data Register 1, MS-Word
0C Low Byte High Byte	Initializing Register 1 Initializing Register 2	Initializing Register 1 Initializing Register 2
0E Low Byte High Byte	Control Register 1 No function	Status Register 1 Status Register 2
10 Low Byte High Byte	Reference Mark Register No function	No function Amplitude Register
12 Low Byte High Byte	Enable Register for Measured Value Latching Axis Cascading	No function No function
14 Low Byte High Byte	Interrupt Enable Register No function	Interrupt Status Register 1 Interrupt Status Register 2
16 Low Byte High Byte	Offset Register for 0° Signal No function	Amplitude for 0°signal Amplitude for 0° signal
18 Low Byte High Byte	Offset Register for 90° Signal No function	Amplitude for 90° signal Amplitude for 90° signal
1A Low Byte High Byte	Timer Register, LSB Timer Register, MSB	No function No function
1C Low Byte High Byte	Control Register 2 No function	Status Register 3 Code Register
1E Low Byte High Byte	Control Register 3 No function	Status Register 4 No function

Data registers for the counters

The measured values are stored in 48-bit registers. Two data registers are available for each axis: Data Register 0 (00h to 04h) and Data Register 1 (06h to 0Ah). The measured values are composed of the 10-bit interpolation value and the 32-bit value of the period counter. Only 42 bits of the 48-bit registers are thus used for the measured value. The upper 6 bits are expanded with sign in two's complement form.

The data width of 48 bits can be shortened to 32 bits via Initializing Register 1 (0Ch), bit D6.

Initializing Register 1 (0Ch), bit D7 can also be used to define whether the measured value is formed only from the value of the period counter (D0 to D9 are not defined) or from the value of the period counter and the interpolation value.
The counter values can be stored in the data registers in the following ways:

- Software latch
- External inputs
- Timers
- Reference marks

The circuit diagram of the counters (see last fold-out page) illustrates the action of the various latch signals.

Bit D0 or D1 in Status Register 1 (0Eh) can be used to poll whether the measured value was stored in the data registers. As long as bit D0 or D1 is set, no further measured value can be stored until the most significant word of the measured value is read. (Exception: via Control Register 2, bit D6 or D7, the latch is enabled without the measured value having been read out.) In 48-bit mode these are the Data Registers 04h or 0Ah, and in 32-bit mode the Data Registers 02h or 08h. When the measured value has been read, bit D0 or D1 in Status Register 1 (0Eh) is reset.

If the counter is stopped, or stored through crossing over the reference marks, bits D0 to D9 will contain the fixed value 256.

0Ch: Initializing Register 1 (write mode)

Bit	Function
D0	**Operation with /without interpolation** 0 = Operation as period counter (w/o interpolation – data bits D0 to D9 are not defined) 1 = Measured value is formed from the value of the period counter and the interpolation value.
D1	0
D2	**Timer** 0 = Reset and stop timer 1 = Start timer
D3	No function
D4	
D5	
D6	**Latch enable** 0 = Mode: 32-bit Register Reading bits D24 to D31 resets status bit D0 or D1 in Status Register 1 (0Eh). 1 = Mode: 48-bit Register Reading bits D40 to D47 resets status bit D0 or D1 in Status Register 1 (0Eh).
D7	**Counting direction** The counting direction determines whether the counters count positive (normal) or negative (inverse) when the traverse direction is positive. 0 = Normal counting direction 1 = Inverse counting direction Inverse counting direction is permitted only in the period counter mode. In operation with interpolation, inverse counting direction will result in faulty gating of the interpolation value and the period counter value.

0Ch: Initializing Register 2 (write mode)

Bit	Function
D8 D9	**Only in operation as period counter:** **edge evaluation** The two incremental encoder signals (0°and 90° el.) provide a maximum of four edges for evaluation per signal period. The counters can be programmed to count one, two or four edges per signal period. D9 D8 0 0 = 1-fold 0 1 = 2-fold 1 1 = 4-fold In operation with interpolation value, 1-fold evaluation is automatically set.
D10	**Only in operation as period counter:** **Counting mode** 0 = Linear counting mode -2^{41} to $+2^{41} - 1$ 1 = Angle counting mode as defined below in D11 For angle encoders with 36 000 or 360 000 lines per revolution.
D11	**Only with angle display:** **Counting mode** 0 = 17 999 to –18 000 1 = 179 999 to –180 000
D12	0
D13	0
D14 D15	**Measured value latch with reference pulse** 0 = 1st reference mark stores in Data Register 0 1 = 1st reference mark stores in Data Register 1 0 = 2nd reference mark stores in Data Register 0 1 = 2nd reference mark stores in Data Register 1

0Ch: **Read mode:** Bits D0 to D15: Read back of Initializing Registers 1 and 2

0Eh: Control Register 1 (write mode)

Bit	Function
D0	1 = Software latch: measured value in Data Register 0
D1	1 = Software latch: measured value in Data Register 1
D2	1 = Software latch in all data registers (must be enabled in Latch Enable Register)
D3	1 = Start counter
D4	1 = Stop counter
D5	1 = Delete counter
D6	1 = Clear encoder error (frequency exceeded)
D7	Delete amplitude value register
D8 D9 D10 D11 D12 D13 D14 D15	No function

0Eh: Status Register 1 (read mode)

Bit	Function
D0	Status for software latch in Register 0 1 = Measured value ready
D1	Status for software latch in Register 1 1 = Measured value ready
D2	No function
D3	No function
D4	1 = Counter is stopped
D5	Counter 1: *No function* Counter 2: I^2C bus line SDA (input)
D6	1 = Encoder error (frequency exceeded)
D7	No function

0Eh: Status Register 2 (read mode)

Bit	Function
D8	1 = Reference mark traverse is active
D9 D10 D11 D12	No function
D13	Logic level for 0° signal
D14	Logic level for 90° signal
D15	Logic level for reference mark

10h: Reference Mark Register (write mode)

HEIDENHAIN linear and angle encoders can feature one or several reference marks. HEIDENHAIN recommends in particular measuring systems with distance-coded reference marks.

In the case of power interruption the link between the encoder position and the displayed position value is lost. This reference can be reestablished after power-on with the encoder reference marks.

When a reference mark is traversed a signal is generated which marks this position on the scale as reference point. This reference point serves to restore the link between the axis positions and the display values last defined. For linear encoders with distance-coded reference marks a reference can be reestablished after max. 20 mm traverse.

Bit	Function while traversing reference mark
D0	1 = Start counter
D1	1 = Stop counter
D2	1 = Clear counter
D3	1 = Latch measured value
D4	1 = Latch measured value when second reference mark is crossed over
D5	1 = Clear counter each time a reference mark is crossed over
D6 D7 D8 D9 D10 D11 D12 D13 D14 D15	No function

Evaluating distance-coded reference marks
Measuring systems with distance-coded reference marks have reference marks spaced at a regular interval over the entire length of the measuring range. Between each two reference marks is another, whose distance to each of the first two changes with each interval (see below). Each of these distances is a multiple of the grating period and each is unique. Thus only two consecutive reference marks need to be traversed following a power interruption for the link between axis positions and display values to be reestablished.

The example below illustrates how these distance-coded reference marks are evaluated:
If the nominal increment is 1000 signal periods, the reference marks are arranged in the following pattern:

```
   501       499        502       498       503       497
|------------|-------------|------------|------------|-------------|-------------|----
0         501      1000      1502      2000       2503       3000
```

First you need to initialize the reference mark register 10h as follows:

- Cancel and start counter with traverse of the first reference mark (Bit D0 = 1 and D2 =1).
- Latch counter with traverse of the second reference mark (Bit D4 = 1).

To calculate the absolute position only the distance between the reference marks in signal periods is required. This distance in increments, called DIFF in the description below, must be divided by 1024.

DIFF = DISTANCE_IN_INCR : 1024

To calculate the absolute position, the distance (OFFSET) between the 1st reference mark on the scale ("0" position in the drawing) and the 1st reference mark crossed over must be determined.

There are four possible cases:

1. Positive traverse direction and DIFF > 500
   OFFSET = (DIFF – 501) • 1000
2. Positive traverse direction and DIFF < 500
   OFFSET = (500 – DIFF) • 1000 – DIFF
3. Negative traverse direction and $|\text{DIFF}|$ > 500
   OFFSET = (DIFF – 501) • 1000 + DIFF
4. Negative traverse direction and $|\text{DIFF}|$ < 500
   OFFSET = (500 – DIFF) • 1000

The absolute position in increments can be calculated as follows:

ABS_POS_INCR = ACT_POS + PRESET + DISTANCE

ACT_POS: Distance (in increments) between the current position and the 1st reference mark crossed over.

PRESET: Position (in increments) assigned to the 1st reference mark of the scale on datum setting ("0" position in the drawing).

DISTANCE: Distance (in increments) between the first reference mark on the scale and the first traversed reference mark.

DISTANCE = OFFSET • 1024

The absolute position is calculated as follows – e.g. when using a scale with a signal period of 0.02 mm:

$$\text{ABS\_POS\_MM} = \frac{\text{ABS\_POS\_INCR} \bullet 360°}{1024}$$

With angle encoders:

$$\text{ABS\_POS\_DEGREE} = \frac{\text{ABS-POS\_INCR} \bullet 360°}{1024 \bullet \text{LINES\_PER\_REV.}}$$

You will find an application example in "TURBO PASCAL" for evaluating reference marks in the source code of TNC.EXE in the file CNT_2.PAS under (* Distance-coded ref. marks *).

10h: Amplitude Value Register (read mode)

Bit	Function
D0 D1 D2 D3 D4 D5 D6 D7	No function
D8 D9	**Current amplitude** A new amplitude value is determined with each measured value latch. D9 D8 **IK 121 A** **IK 121 V** 0 0 Normal amplitude $5\ \mu A < I_e < 15\ \mu A$ $0.47\ V_{PP} < U_e < 1.41\ V_{PP}$ 0 1 Low amplitude $2.5\ \mu A < I_e < 5\ \mu A$ $0.23\ V_{PP} < U_e < 0.47\ V_{PP}$ 1 0 High amplitude $I_e > 15\ \mu A$ $U_e > 1.41\ V_{PP}$ 1 1 Erroneously low amplitude $I_e < 2.5\ \mu A$ $U_e < 0.23\ V_{PP}$ Before being read, the amplitude value should be frozen by bit D4 in Control Register 2. The Amplitude Value Register is reset by bit D7 in Control Register 1.
D10 D11	**Minimum value of the amplitude** Coding and read mode: see bits D8 and D9. If the current amplitude value is lower than the stored minimum value more than four times in a row, the IK will replace the old minimum value with the current amplitude value.
D12 D13 D14 D15	No function

12h: Enable Register for measured value latch (write mode)

Bit	Function
D0	Enable L0 for Data Register 0 Axis 1: 0 Axis 2: 1 = Enable cascading with axis 1
D1	Enable L0 via delay circuit (125 ns) for Data Register 0 Axis 1: 1 = Enable external latch signal X3.L0 for Data Register 0 Axis 2: 0
D2	1 = Enable "software latch in all data registers" for Data Register 0
D3	1 = Enable "software latch via timers " for Data Register 0
D4	Enable L1 for Data Register 1 Axis 1: 1 = Enable external latch signal X3.L0 for Data Register 1 Axis 2: 1 = Enable external latch signal X3.L1 for Data Register 1
D5	Enable L1 via delay circuit (125 ns) for Data Register 1 0
D6	1 = Enable "software latch in all data registers " for Data Register 1
D7	1 = Enable "software latch via timers " for Data Register 1

The circuit diagram of the counters on the last foldout page shows the functions of the individual bits.

12h: Register for axis cascading and for I²C bus control (write mode)

Bit	Function
D8	Axis 1: 1 = Enable the external latch signal X3.L0 to 2nd axis (L0, X3.OUT) Axis 2: 0
D9	Axis 1: 1 = Enable "software latch in all data registers" to second axis (L0, X3.OUT) Axis 2: 0
D10	Axis 1: 1 = Enable timer strobe to second axis (L0, X3.OUT) Axis 2: 0
D11	Axis 1: 0 Axis 2: I²C bus line SDA: data (signals must be programmed inverted)
D12	No function
D13	
D14	
D15	Axis 1: 0 Axis 2: I²C bus line SCL: clock (signals must be programmed inverted)

The circuit diagram of the counters on the last foldout page shows the functions of the individual bits.

12h: Read mode has no function

14h: Interrupt Enable Register (write mode)

The interrupt logic is programmed through the Interrupt Enable Register. The interrupt can be caused by the measured value latch in Register 0, Register 1 or the Timer strobe. Each interrupt source can be programmed independently of the others. If several interrupts are received at the same time, the following priorities are used:

- Highest priority: measured value latch via Register 0
- Second-highest priority: measured value latch via Register 1
- Lowest priority: measured value latch via timer strobe

After an interrupt, no further interrupts can follow until Interrupt Status Register 2 has been read.

Bit	Function
D0	1 = Enable interrupt 0 with measured value latch via Register 0
D1	1 = Enable interrupt 1 with measured value latch via Register 1
D2	1 = Enable interrupt 2 for timer strobe
D3	1 = Interrupt is generated (D0 = D1 = D2 = 0)
D4 D5 D6 D7 D8 D9 D10 D11 D12 D13 D14 D15	No function

The circuit diagram of the **counters** on the last foldout page shows the functions of the individual bits.

14h: Interrupt Status Register 1 (read mode)

The currently active interrupt is displayed in this register (only one bit can be set). Reading this register resets the currently active interrupt and deletes the associated status bit so that the next interrupt can be triggered by a negative edge.

Bit	Function
D0	1 = Interrupt 0 is active, there was a measured value latch via Data Register 0
D1	1 = Interrupt 1 is active, there was a measured value latch via Register 1
D2	1 = Interrupt 2 is active; there was a measured value latch via the timer strobe
D3 D4 D5 D6 D7	No function

14h: Interrupt Status Register 2 (read mode)

In this register, all pending interrupts are shown (i.e., the active interrupt and the interrupts still to be executed). Several bits may therefore be set at the same time.

Bit	Function
D8	1 = Interrupt 0 is pending but not yet executed
D9	1 = Interrupt 1 is pending but not yet executed
D10	1 = Interrupt 2 is pending but not yet executed
D11 D12 D13 D14 D15	No function

16h: Offset Register for 0° signal (write mode)

This register contains the 7-bit offset value for the 0° signal in two’s complement form. From this there results a maximum compensation of ± 63.
The offset values can only be written if one of the status bits D5 or D6 in Status Register 3 has the value 0.

Functional principle:
Offset values are added to the digital values of the 0° signal (0 to 1023) and the 90° signal. In case of overflow the value is limited to 1023 or 0.

Bit	Function
D0 D1 D2 D3 D4 D5 D6	Offset value for the 0° signal in two’s complement form
D7 D8 D9 D10 D11 D12 D13 D14 D15	No function

The offset register in the counters is **volatile.** For this reason the offset values are stored in an EEPROM in the IC for the electronic potentiometers. After switch-on, the offset values must be loaded from the EEPROM into the offset registers of the **counters** (see procedures "store_offset" and "load_offset").

16h: Amplitude for 0°signal (read mode)

After each analog-digital conversion the result is stored. Before reading, the values should be frozen by bit D4 in Control Register 2.

Bit	Function
D0 D1 D2 D3 D4 D5 D6 D7 D8 D9	Amplitude for the 0° signal
D10 D11 D12 D13 D14 D15	No function

18h: Offset Register for 90° signal (write access)

This register contains the 7-bit offset value for the 90° signal. For a description of the function, see "16h: Offset register for 0° signal"

Bit	Function
D0 D1 D2 D3 D4 D5 D6	Offset value for the 90° signal in two’s complement form
D7 D8 D9 D10 D11 D12 D13 D14 D15	No function

18h: Amplitude for 90°signal (read access)

After each analog-digital conversion the result is stored. Before reading, the values should be frozen by bit D4 in Control Register 2.

Bit	Function
D0 D1 D2 D3 D4 D5 D6 D7 D8 D9	Amplitude for the 90° signal
D10 D11 D12 D13 D14 D15	No function

1Ah: Timer Register (write access)

The 13-bit timer value is stored in Registers 1Ah and 1Bh. The Timer Register can therefore hold values from 0 to 8191. The cycle time is indicated in µs; one µs must be subtracted from the desired cycle time.

Example

Desired cycle time: 1 ms (= 1000 µs)
Value to be programmed: 1000 – 1 = 999
Note that writing this register does not start the timer. The timer is started by setting bit D2 in Initializing Register 1 (0Ch). Furthermore, the corresponding bits must be set in Enable Registers 12h, 13h or 14h.

Bit	Function
D0 D1 D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12	Timer value
D13 D14 D15	No function

1Ah: Read mode has no function

1Ch: Control Register 2 (write access)

Bit	Function
D0 D1 D2 D3	Program a fixed value ≥ 8
D4	1 = Freeze amplitude for 0° signal (16h) and 90° signal (18h), as well as Amplitude Value Register (10h High Byte). This bit must be set to prevent the register values from being changed during readout. Whether the register values are frozen can be read in bit D4 in Status Register 3.
D5	0
D6	1 = New latch possible via Data Register 0 without having to fetch the measured value beforehand. In this mode it is possible to change the measured value while reading.
D7	1 = New latch possible via Data Register 1 without having to fetch the measured value beforehand. In this mode it is possible to change the measured value while reading.
D8 D9 D10 D11 D12 D13 D14 D15	No function

1Ch: Status Register 3 (read access)

Bit	Function
D0 D1 D2 D3	Cannot be read
D4	1 = Amplitude for 0° signal (16h) and 90° signal (18h) as well as Amplitude Value Register (10h High Byte) are frozen and can be read.
D5	0 = Offset Register for the 0° signal has been written.
D6	0 = Offset Register for the 90° signal has been written.
D7	No function

1Ch: Code Register (read access)

Bit	Function
D8 D9 D10 D11 D12 D13 D14 D15	IC code: 8 (counter chip G26) [1]

[1] Chip designation from HEIDENHAIN

1Eh: Control Register 3 (write access)

Bit	Function
D0 D1	Fixed value 0
D2 D3 D4 D5 D6 D7 D8 D9 D10 D11 D12 D13 D14 D15	No function

1Eh: Status Register 4 (read mode)

Bit	Function
D0	No function
D1	Logic level on pin L0
D2	Logic level on pin L1
D3 D4 D5 D6 D7 D8 D9 D10 D11 D12 D13 D14 D15	No function

The IK 121 in DOS-Applications

Quick Start

Do **not** yet install the IK 121.

➤ Insert the floppy disk with the driver software into drive A: of your PC.

➤ Enter the following DOS commands:

>A:

>INSTALL

After the initial screen the following information appears on your PC screen:

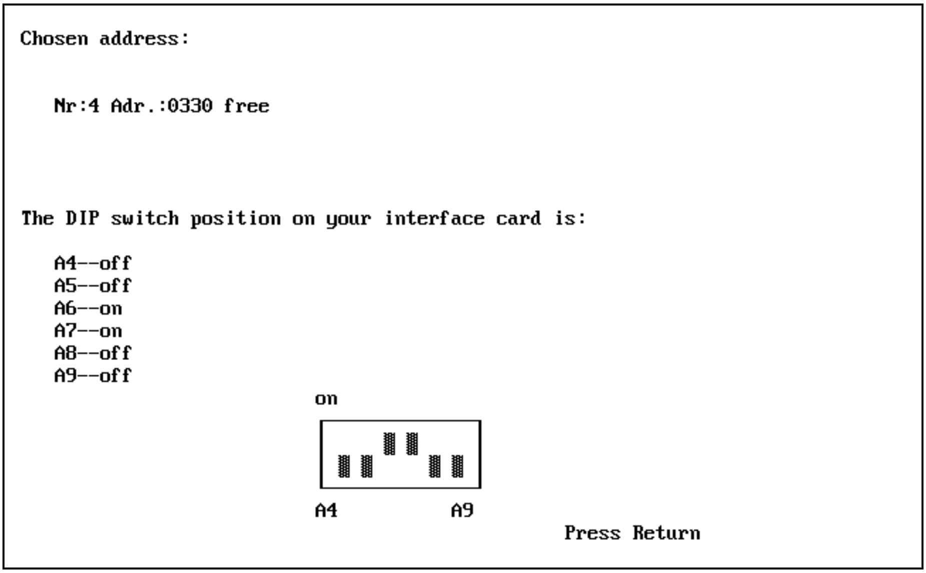
```
INSTALL IK 121
The IK 121 Interface Card should not be in
your PC!
Continue (y/n)?
```

➤ Enter "y".

The INSTALL program now shows an overview of the port addresses. Vacant port addresses are indicated as "free", occupied port addresses as "not free".

➤ Choose a vacant port address and enter the corresponding number (for example, 4).

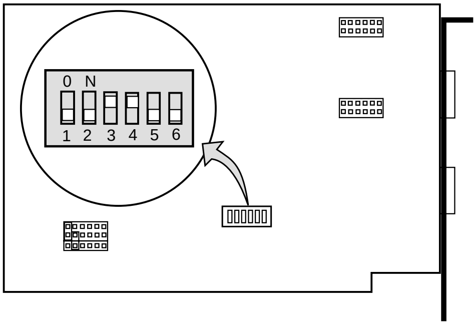
INSTALL now shows you how to set the DIP switches on the IK 121.



```
Chosen address:

   Nr:4 Adr.:0330 free



The DIP switch position on your interface card is:

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

```
                                                              Press Return
```

➤ Set the DIP switches on the IK 121.



➤ **Press "**Return" on the PC to store the selected port address in the file IK 121.INI. The address is available to all supplied example programs except SAMPLE32.PAS, SAMPLE48.PAS, SAMPLE32.C and SAMPLE48.C.

➤ Press "Return".
This ends the installation program.

If INSTALL does not run on your PC, use the standard setting (address 0330h). This is the factory setting of the IK 121 as well as the default setting for the software.

If this address is already occupied on your PC, use a text editor to write the address you selected (decimal) into IK121.INI. Or, start IK121.EXE and select "File", "Setup IK121", "Setup board", "Base address" and enter the desired address (decimal). Save the address with "Save setup".

➤ Remove the floppy disk from drive A, unplug the power cord of your PC and open the housing. Refer to the user's manual of your PC for information on how to install additional cards.

➤ Insert the IK 121 into your PC (a short slot is sufficient), secure the card and reinstall the PC housing.

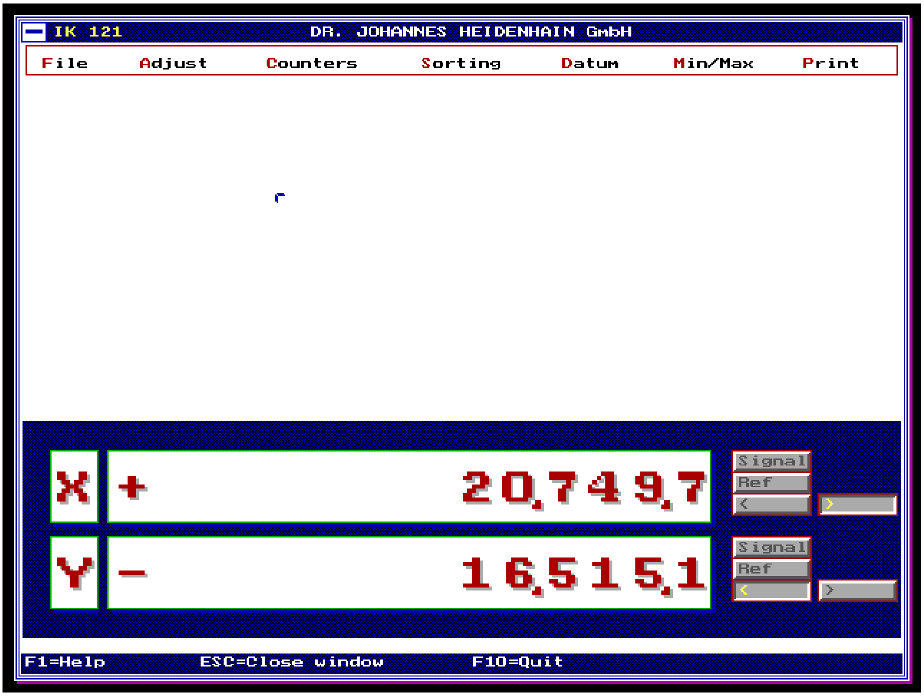
➤ Connect one or two linear encoders to input X1 or X2 using HEIDENHAIN adapter cable (Id.-Nr. 309 785 ..). The encoders must have sinusoidal output signals.

➤ Plug the PC's power cord back in. Switch on the PC.

➤ Insert the floppy disk with the driver software into drive A again. Enter the following DOS commands:

```
>A:
>IK121
```

➤ After the initial screen the PC displays the following:



IK 121.EXE displays the position values of the encoders.

The example program is dialog-driven. You can use either the mouse or the keyboard. You can call up additional explanations for the menu options by pressing F1.

The programming of the IK 121 in DOS applications is shown in this manual with examples in TURBO PASCAL and BORLAND C. The IK 121 uses port addresses and can be programmed with any programming language that can address port addresses.

IK121_0.* files: Basic functions for writing and reading the registers

The functions described in the following are on the supplied floppy disk. For TURBO PASCAL they are located under the directory **TP** in the file **IK121_0.PAS.** For BORLAND C they are located under the directory **BC** in the file **IK121_0.C** and the associated header file **IK121_0.H.**

The **counters** of the IK 121 are addressed through address registers (see "Addressing"). The address register must be set before a counter can be addressed. The function **write_g26** writes the data register, while **read_g26** reads it. These are the two basic functions for working with the IK 121.

Differences between TURBO PASCAL and BORLAND C

TURBO PASCAL understands *procedures* (functions without a return value) and *functions* (with a return value), while BORLAND C only understands functions (with or without a return value). This manual will use the TURBO PASCAL terms *procedure* and *function*.

Procedure for writing to the registers

The following procedure writes a value to the 16-bit register of a counter.

Procedure: **write_g26**[1]

Parameters

baseadr:	Base address of the IK 121 that was set with DIP switches
axis:	1 = axis 1; 2 = axis 2
address:	Addresses B0 to B4 of the register of the counter
datum:	Value to be written to the address

1) g26 is the HEIDENHAIN designation of the counter

Procedure in TURBO PASCAL

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

  BEGIN
       (*Discard the last four bits of card address*)
    baseadr:=baseadr and $0FF0;
       (*Address B0 to B4 of the counter*)
    address:=address and $001F;

    (*Load address pointer into address register*)
       (*Address of the address register*)
    adr_reg:=baseadr or $0008;
       (*Value of the address register R0 to R2 =
       address of the counter without B0 and B1*)
    adr_point:=address shr 2;
       (*Load address register*)
    portw[adr_reg]:=adr_point;

    (*Calculate port address*)
    if axes=1 then
      adr_gate:=baseadr or (address and $03)
    else
      adr_gate:=(baseadr or $0004) or (address and $03);

       (*Write <datum> to the counter*)
    portw[adr_gate]:=datum;
  END;
```

Procedure in BORLAND C

```
void write_g26 (unsigned int baseadr, unsigned char axis,
               unsigned int address, unsigned int datum)
{
unsigned int         adr_reg, adr_point, adr_gate;
   /*Discard the last four bits of the card address*/
baseadr = baseadr & 0x0FF0;
   /*Address B0 to B4 of the counter*/
address = address & 0x1F;

/*Load address pointer into address register*/
   /*Address of the address register*/
adr_reg = baseadr | 0x0008;
   /*Value of the address register R0 to R2 =
   address of the counter without B0 and B1*/
adr_point = address >> 2;
   /*Load address register*/
outpw (adr_reg, adr_point);

/*Calculate port address*/
switch (axis)
       {
       case 1:
       adr_gate = baseadr | (address & 0x03);
       break;
       case 2:
       adr_gate = (baseadr | 0x0004) | (address & 0x03);
       break;
       }

   /*Write <datum> to the counter*/
outpw (adr_gate, datum);
}
```

Function for reading the registers

The following function reads a value from a 16-bit register of a counter.

Function:	**read_g26**[1]

Parameters

baseadr:	Base address of the IK 121 that was set with DIP switches
axis:	1 = axis 1, 2 = axis 2
address:	Addresses B0 to B4 of the register of the counter
Result:	Value from the data register indicated under "axis" and "address" as a 16-bit variable

Function in TURBO PASCAL

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

  BEGIN
       (*Discard the last four bits of the card address*)
    baseadr:=baseadr and $0FF0;
       (*Address B0 to B4 of the counter*)
    address:=address and $001F;

    (*Load address pointer into address register*)
       (*Address of the address register*)
    adr_reg:=baseadr or $0008;
       (*Value of the address register R0 to R2 =
       address of the counter without B0 and B1*)
    adr_point:=address shr 2;
       (*Load address register*)
    portw[adr_reg]:=adr_point;

    (*Calculate port address*)
    if axes=1 then
      adr_gate:=baseadr or (address and $03)
    else
      adr_gate:=(baseadr or $0004) or (address and $03);

       (*Read datum from the counter*)
    read_g26:=portw[adr_gate];
  END;
```

1) g26 is the HEIDENHAIN designation of the counter

Function in BORLAND C

```
unsigned int read_g26 (unsigned int baseadr,
               unsigned char axis, unsigned int address)
{
unsigned int adr_reg, adr_point, adr_gate;
   /*Discard the last four bits of the card address*/
baseadr = baseadr & 0x0FF0;
   /*Address B0 to B4 of the counter*/
address = address & 0x1F;

/*Load address pointer into address register*/
   /*Form address of the address register*/
adr_reg = baseadr | 0x0008;
   /*Value of the address register R0 to R2 =
   address of the counter without B0 and B1*/
adr_point = address >> 2;
   /*Load address register*/
outpw (adr_reg, adr_point);

/*Calculate port address*/
switch (axis)
       {
       case 1:
       adr_gate = baseadr | (address & 0x03);
       break;
       case 2:
       adr_gate = (baseadr | 0x0004) | (address & 0x03);
       break;
       }
   /*Read datum from the counter*/
return(inpw(adr_gate));
}
```

Simple functions for measured value latch via software

Procedures for latching a measured value

The following procedures store the counter content of the desired axis in Data Register 0 (soft_l0) or Data Register 1 (soft_l1).

Procedure:	**soft_l0**
	soft_l1
Parameters	
baseadr:	Base address of the IK 121 that was set with DIP switches
axis:	1 = axis 1; 2 = axis 2

Procedures in TURBO PASCAL

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

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

Procedure in BORLAND C

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

Function for checking whether the measured value was stored

Function:	**latched**
Parameter	
baseadr:	Base address of the IK 121 that was set with DIP switches
axis:	1 = axis 1; 2 = axis 2
reg:	0 = Data Register 0; 1 = Data Register 1
Result:	False = no measured value was stored True = a measured value was stored

Function in TURBO PASCAL

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

Function in BORLAND C

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

Procedure for repeatedly polling whether the measured value was stored

The following procedure continues to poll whether a measured value was stored until a measured value is stored.

Procedure: **poll_latch**

Parameters

baseadr:	Base address of the IK 121 that was set with DIP switches
axis:	1 = axis 1; 2 = axis 2
reg:	0 = Data Register 0; 1 = Data Register 1

Function in TURBO PASCAL

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

Function in BORLAND C

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

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

Function for reading a 32-bit measured value

The following function reads a 32-bit measured value from a counter.

Function:	**read_count_value32**

Parameters

baseadr:	Base address of the IK 121 that was set with DIP switches
axis:	1 = axis 1; 2 = axis 2
reg:	0 = Data Register 0, 1 = Data Register 1
Result:	TURBO PASCAL: Value from the data register given under **axis** and **reg** as integer variable of the **comp** type. BORLAND C: as an integer variable of the **long** type.

Function in TURBO PASCAL

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

Function in BORLAND C

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

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

Function for reading a 48-bit measured value
The following function reads a 48-bit measured value from a counter.

Function: **read_count_value48**

Parameters

baseadr:	Base address of the IK 121 that was set with DIP switches
axis:	1 = axis 1; 2 = axis 2
reg:	0 = Data Register 0; 1 = Data Register 1
Result:	TURBO PASCAL: Value from the data register given under **axis** and **reg** as an integer variable of the **comp** type BORLAND C: As a floating point variable of the **double** type

Function in TURBO PASCAL

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

Function in BORLAND C

```
double read_count_value48 (unsigned int baseadr,
                  unsigned char axis, unsigned char reg)
{
unsigned int field[3];
double count_value48;
switch (reg)
       {
       case 0:
       field[0] = read_g26 (baseadr, axis, 0);
       field[1] = read_g26 (baseadr, axis, 2);
       field[2] = read_g26 (baseadr, axis, 4);
       break;
       case 1:
       field[0] = read_g26 (baseadr, axis, 6);
       field[1] = read_g26 (baseadr, axis, 8);
       field[2] = read_g26 (baseadr, axis, 10);
       break;
       }
if (field[2] && 0x8000)
       count_value48 = (double)((field[0]-65535) +
       65536.0*(field[1]-65535)+
       4294967296.0*(field[2]-65535)-1);
       else
       count_value48 = (double)(field[0] +
       65536.0*field[1] +
       4294967296.0*field[2]);
return (count_value48);
}
```

Simple program for measured value latching via software

The following examples employ the previously defined functions. The examples are included on the supplied floppy disk 1.

For TURBO PASCAL they are located under the directory **TP** in the files **SAMPLE32.PAS** and **SAMPLE48.PAS.**

For BORLAND C the examples are located in the directory **BC** in the file **SAMPLE32.C** and **SAMPLE48.C.**

In this example the counter is displayed on the screen in millimeters. The IK 121 can of course also display angular degrees. The conversion is shown by the following two formulas.

Converting the count into millimeters

$$\text{Value } [\text{mm}] = \text{Count} \bullet \frac{\text{Grating period } [\text{mm}]}{1024}$$

Example: Grating period = 20 µm

$$\text{Value } [\text{mm}] = \text{Count} \bullet \frac{0.020\ [\text{mm}]}{1024}$$

Converting the count into degrees

$$\text{Value } [^\circ] = \frac{\text{Count} \bullet 360\ [^\circ]}{1024 \bullet \text{Lines / rev.}}$$

Example: Lines/rev. = 36 000

$$\text{Value } [^\circ] = \frac{\text{Count} \bullet 360\ [^\circ]}{1024 \bullet 36\ 000}$$

Examples in TURBO PASCAL: Measured value latch via software

The following examples SAMPLE32.PAS and SAMPLE48.PAS illustrate the use of the previously described procedures and functions. These are declared and defined in the file **IK121_0.TPU** (all files are on the supplied floppy disk 1). Before compiling SAMPLE0.PAS, it is necessary to compile IK121_0.PAS to "unit" IK121_0.TPU. Since this example does not read the address of the IK 121 from IK121.INI, the address is defined as the constant "base_address".

```
program sample32;
{-------------------------------------------------------
 DR. JOHANNES HEIDENHAIN GmbH, Traunreut, Germany
 A simple program for the IK 121 to display
 two axes. Measured value with 32 bits.
 V 1.01
 April 1995
 -------------------------------------------------------}
{$N+,E+}
{$V+}
{$R+}
USES crt,ik121_0;
CONST
  base_address = $330;
VAR
  c_value_0, c_value_1  : comp;
BEGIN
clrscr;
   (*Initialise the board in interpolation mode, axis 1*)
write_g26 (base_address, 1, $0c, $0001);
   (*Initialise the board in interpolation mode, axis 2*)
write_g26 (base_address, 2, $0c, $0001);
   (*Reset error bit, start counter, axis 1*)
write_g26 (base_address, 1, $0e, $0048);
   (*Reset error bit, start counter, axis 2*)
write_g26 (base_address, 2, $0e, $0048);
   (*Write to control register 2, axis 1*)
write_g26 (base_address, 1, $1c, $0008);
   (*Write to control register 2, axis 2*)
write_g26 (base_address, 2, $1c, $0008);
REPEAT
   (*Software latch in register 0, axis 1*)
soft_l0 (base_address, 1);
   (*Software latch in register 0, axis 2*)
soft_l0 (base_address, 2);
   (*Poll whether latched in axis 1*)
poll_latch (base_address, 1, 0);
   (*Read axis 1*)
```

```
c_value_0:= read_count_value32 (base_address, 1, 0);
   (*Poll whether latched in axis 2*)
poll_latch (base_address, 2, 0);
 (*Read axis 2*)
c_value_1:= read_count_value32 (base_address, 2, 0);
(*Display measured values*)
gotoxy(1,10);
write(c_value_0*0.02/1024:16:4,
       c_value_1*0.02/1024:16:4);
UNTIL KEYPRESSED;
END.

program sample48;
{-------------------------------------------------------
 DR. JOHANNES HEIDENHAIN GmbH, Traunreut, Germany
 A simple program for the IK 121 to display
 two axes. Measured value with 48 bits.
 V 1.01
 April 1995
 -------------------------------------------------------}
{$N+,E+}
{$V+}
{$R+}
USES crt,ik121_0;
CONST
  base_address = $330;
VAR
  c_value_0, c_value_1  : comp;
BEGIN
clrscr;
   (*Initialise the board in interpolation mode, axis 1*)
write_g26 (base_address, 1, $0c, $0041);
   (*Initialise the board in interpolation mode, axis 2*)
write_g26 (base_address, 2, $0c, $0041);
   (*Reset error bit, start counter, axis 1*)
write_g26 (base_address, 1, $0e, $0048);
   (*Reset error bit, start counter, axis 2*)
write_g26 (base_address, 2, $0e, $0048);
   (*Write to control register 2, axis 1*)
write_g26 (base_address, 1, $1c, $0008);
   (*Write to control register 2, axis 2*)
write_g26 (base_address, 2, $1c, $0008);
REPEAT
   (*Software latch in register 0, axis 1*)
soft_l0 (base_address, 1);
   (*Software latch in register 0, axis 2*)
soft_l0 (base_address, 2);
   (*Poll whether latched in axis 1*)
poll_latch (base_address, 1, 0);
   (*Read axis 1*)
c_value_0:= read_count_value48 (base_address, 1, 0);
   (*Poll whether latched in axis 2*)
poll_latch (base_address, 2, 0);
 (*Read axis 2*)
c_value_1:= read_count_value48 (base_address, 2, 0);
```

```
(*Display measured values*)
gotoxy(1,10);
write(c_value_0*0.02/1024:16:4,
      c_value_1*0.02/1024:16:4);
UNTIL KEYPRESSED;
END.
```

Examples in BORLAND C:
Measured value latch via software

The following examples SAMPLE32.C and SAMPLE48.C illustrate the use of the previously described functions. These are declared and defined in the files IK121_0.H and IK121_0.C (all files are on the supplied floppy disk 1). For compiling, a project must be created with IK121_0.C and SAMPLE32.C or SAMPLE48.C. Since this example does not read the address of the IK 121 from IK121.INI, the address is defined as the constant "base_address".

```
/*----------------------SAMPLE32.C----------------------
 DR. JOHANNES HEIDENHAIN GmbH, Traunreut, Germany
 A simple program for the IK 121 to display
 two axes. Measured value with 32 bits.
 V 1.01
 April 1995
 Project files:     IK121_0.C, SAMPLE32.C
 Include file:      IK121_0.H
 -----------------------------------------------------*/
#include <stdio.h>
#include <conio.h>
#include "ik121_0.h"
#define base_address 0x0330
int main()
{
double c_value_0, c_value_1;
cls;
   /*Initialise the board in interpolation mode, axis 1*/
write_g26 (base_address, 1, 0x0c, 0x0001);
   /*Initialise the board in interpolation mode, axis 2*/
write_g26 (base_address, 2, 0x0c, 0x0001);
   /*Reset error bit, start counter, axis 1*/
write_g26 (base_address, 1, 0x0e, 0x0048);
   /*Reset error bit, start counter, axis 2*/
write_g26 (base_address, 2, 0x0e, 0x0048);
   /*Write to control register 2, axis 1*/
write_g26 (base_address, 1, 0x1c, 0x0008);
   /*Write to control register 2, axis 2*/
write_g26 (base_address, 2, 0x1c, 0x0008);
/*Cursor off*/
_setcursortype (_NOCURSOR);
while(!kbhit())
      {
           /*Software latch in register 0, axis 1*/
       soft_l0 (base_address, 1);
           /*Software latch in register 0, axis 2*/
       soft_l0 (base_address, 2);
           /*Poll whether latched in axis 1*/
       poll_latch (base_address, 1, 0);
          /*Read axis 1*/
       c_value_0 = (double)read_count_value32
                                 (base_address, 1, 0);
           /*Poll whether latched in axis 2*/
       poll_latch (base_address, 2, 0);
          /*Read axis 2*/
       c_value_1 = (double)read_count_value32
                                 (base_address, 2, 0);
        /*Display measured values*/
       printf("\r\t%16.4f\t%16.4f",c_value_0*0.02/1024,
                             c_value_1*0.02/1024);
       }
/*Cursor on*/
_setcursortype (_NORMALCURSOR);
return (0);
}

/*---------------------SAMPLE48.C---------------------
 DR. JOHANNES HEIDENHAIN GmbH, Traunreut, Germany
 A simple program for the IK 121 to display
 two axes. Measured value with 48 bits.
 V 1.01
 April 1995
 Project files:      IK121_0.C, SAMPLE48.C
 Include file:       IK121_0.H
 ------------------------------------------------------*/
#include <stdio.h>
#include <conio.h>
#include "ik121_0.h"
#define base_address 0x0330
int main()
{
double c_value_0, c_value_1;
cls;
   /*Initialise the board in interpolation mode, axis 1*/
write_g26 (base_address, 1, 0x0c, 0x0041);
  /*Initialise the board in interpolation mode, axis 2*/
write_g26 (base_address, 2, 0x0c, 0x0041);
   /*Reset error bit, start counter, axis 1*/
write_g26 (base_address, 1, 0x0e, 0x0048);
   /*Reset error bit, start counter, axis 2*/
write_g26 (base_address, 2, 0x0e, 0x0048);
   /*Write to control register 2, axis 1*/
write_g26 (base_address, 1, 0x1c, 0x0008);
   /*Write to control register 2, axis 2*/
write_g26 (base_address, 2, 0x1c, 0x0008);
/*Cursor off*/
_setcursortype (_NOCURSOR);
while(!kbhit())
```

```
      {
          /*Software latch in register 0, axis 1*/
      soft_l0 (base_address, 1);
          /*Software latch in register 0, axis 2*/
      soft_l0 (base_address, 2);
          /*Poll whether latched in axis 1*/
      poll_latch (base_address, 1, 0);
         /*Read axis 1*/
      c_value_0 = read_count_value48 (base_address, 1, 0);
          /*Poll whether latched in axis 2*/
      poll_latch (base_address, 2, 0);
         /*Read axis 2*/
      c_value_1 = read_count_value48 (base_address, 2, 0);

      /*Display measured values*/
      printf("\r\t%16.4f\t%16.4f",c_value_0*0.02/1024,
                                   c_value_1*0.02/1024);
      }
/*Cursor on*/
_setcursortype (_NORMALCURSOR);
return (0);
}
```

IK121_1.PAS: Functions for a RAM memory map in TURBO PASCAL

The data structures and functions for TURBO PASCAL described below are on the supplied floppy disk 1 under the directory **\TP** in the files IK121_1.PAS and IIC.PAS (IIC.PAS is included in IK121_1.PAS while compiling). The command **uses** includes these files as a **unit** in further applications. In IK121_1.PAS a RAM memory map of the registers of the IK 121 is created with the help of data structures (in TURBO PASCAL: record).
The data of the RAM memory map are written to the port addresses of the IK 121 using the procedures **init_handler** and **comm_handler.**
In addition, useful functions are available for initializing the card, reading the measured values, interrupt programming and for adjustment of the encoder signals.

Definition of the data structures

Record: **softcommand**
This structure is the RAM memory image of Control Register 1 (write access to 0Eh).
Data fields

start:	Start counter
stop:	Stop counter
clear:	Clear counter
latch0:	Software latch: Measured value in Data Register 0
latch1:	Software latch**:** Measured value in Data Register 1
latch2:	Software latch with special functions
clrfrg:	Clear encoder error (frequency exceeded)
clrstat:	Clear Amplitude Value Register

Record: **refcommand**

This structure is the RAM memory image of the Reference Mark Register (write access to 10h). When the reference mark is traversed, the following functions can be executed:

Data fields

ristart:	Start counter
ristop:	Stop counter
riclear:	Clear counter
riclear2:	Clear counter when each reference mark is traversed
rilatch:	Latch measured value
rilatch2:	Latch measured value when the second reference mark is traversed

Record: **initlatch**

This structure is the RAM memory image of the Enable Register for the measured value latch (write access to 12h).

Data fields

en_L0_reg0:	Enable L0 for Data Register 0
en_latch2_reg0:	Enable "software latch in all data registers" for Data Register 0
en_timer_reg0:	Enable software latch via timers for Data Register 0
en_L1_reg1:	Enable L1 for Data Register 1
en_latch2_reg1:	Enable "software latch in all data registers" for Data Register 1
en_timer_reg1:	Enable software latch via timers for Data Register 1

Record: **initsync**

This structure is the RAM memory image of the Enable Register for axis cascading (write access to 13h). The following data fields apply only to axis 1:

Data fields

en_L0_axis2:	Enable external latch signal X3.L0 to the second axis
en_latch2_axis2:	Enable software latch with special functions to the second axis
en_timer_axis2:	Enable timer strobe to the second axis

Record: **initmain**

This structure is the RAM memory image of Initializing Registers 1 and 2 (write access to 0Ch and 0Dh).

Data fields

mode1024:	0 = Operation as period counter (without interpolation, data bits D0 to D9 are not defined).
	1 = Measured value is calculated from the value of the period counter the interpolation value.
en_timer:	0 = Reset and stop timer
	1 = Start timer
en_48Bit:	0 = Op. mode: 32-bit counter
	1 = Op. mode: 48-bit counter
onefold:	Counters count 1 edge per signal period
twofold:	Counters count 2 edges per signal period
fourfold:	Counters count 4 edges per signal period
arcmode:	0 = Linear counting mode
	1 = Angle counting mode
arc180000:	0 = Angle counting mode: 17 999 to –18 000
	1 = Angle counting mode: 179 999 to –180 000
sel_ri_1st:	0 = First reference mark stored in Data Register 0
	1 = First reference mark stored in Data Register 1
sel_ri_2nd:	0 = Second reference mark stored in Data Register 0
	1 = Second reference mark stored in Data Register 1

Record: **initintrpt**

This structure is the RAM memory image of the Interrupt Enable Register (write access to 14h).

Data fields

reg0:	Enable interrupt 0 with measured value latching via Data Register 0
reg1:	Enable interrupt 1 with measured value latching via Data Register 1
timer:	Enable interrupt 2 for timer strobe
port:	Interrupt is generated

Record: **intstate**

This structure is the RAM memory image of Interrupt Status Registers 1 and 2 (read access to 14h and 15h).

Data fields

L0:	Interrupt 0 is active, there was a measured value latch via Data Register 0
L1:	Interrupt 1 is active, there was a measured value latch via Data Register 1
timer:	Interrupt 2 is active, there was a measured value latch via the timer strobe
pendI0:	Interrupt 0 is pending but not yet executed
pendI1:	Interrupt 1 is pending but not yet executed
pendtimer:	Interrupt 2 is pending but not yet executed

Record: **countstate**

This structure is the RAM memory image of Status Registers 1 and 2 (read access to 0Eh and 0Fh).

Data fields

latch0:	Measured value in Data Register 0 is ready
latch1:	Measured value in Data Register 1 is ready
stop:	Counter is stopped
sda:	PC bus line SDA (input)
error_frq:	Encoder error (frequency exceeded)
ref_activ:	Reference mark traversing is active

Record: **signalstate**

This structure is the RAM memory image of the Amplitude Value Register (read access to 11h), of the Amplitude Register for the 0° signal (read access to 16h and 17h) and of the Amplitude Register for the 90° signal (read access to 18h and 19h).

Data fields

ad00:	Amplitude for the 0° signal
ad90:	Amplitude for the 90° signal
amp_act:	Current amplitude
amp_min:	Minimum value of the amplitude

Pointer: **g26_pointr**

Pointer to a structure **g26_record**

Record: **g26_record**
This structure contains the complete data set of the RAM memory image of the registers for one axis.

Pointer: **ik121_pointr**
Pointer to a structure ik121_record

Record: **ik121_record**
Array of pointers **g26_pointr**

Record: **storage**
Array of structures **g26_record** for the file IK121.INI for storing the initializing values.

Procedures and functions

Function: **look_for_IK121**
Tests whether the hardware of the IK 121 is present.
Result: True if IK121 is present
False if IK 121 is not present
Prototype: FUNCTION look_for_IK121
(board:IK121_pointr):boolean;

Procedure: **init_IK121**
Initializes the IK 121.
Prototype:
PROCEDURE init_ik121 (board:IK121_pointr);

Procedure: **init_handler**
Writes the initializing data from the RAM memory map to the port addresses of the IK 121 with the following functions: g26_main, g26_latch, g26_sync and g26_int.
Prototype: PROCEDURE init_handler
(pointr:g26_pointr);

Procedure: **comm_handler**
Writes the commands from the RAM memory map to the port addresses of the IK 121 using the following functions: g26_soft, g26_ref.
Prototype: PROCEDURE comm_handler
(pointr:g26_pointr);

Procedure: **read_adr**
Reads the address of the IK121 from the file IK121.INI
Prototype: PROCEDURE read_adr (board:ik121_pointr);

Procedure: **write_adr**
Writes the address of the IK 121 into the file IK121.INI
Prototype: PROCEDURE write_adr (board:ik121_pointr);

Procedure: **read_signal_status**
Reads the data for the structure **signalstate** from the registers 11h, 16h, 17h, 18h and 19h.
Prototype: PROCEDURE read_signal_status (pointr:g26_pointr);

Procedure: **read_count_status**
Reads the data for the structure **countstate** from the registers 0Eh and 0Fh.
Prototype: PROCEDURE read_count_status (pointr:g26_pointr);

Procedure: **read_int_status**
Reads the data for the structure **intstate** from the registers 14h and 15h.
Prototype: PROCEDURE read_int_status (pointr:g26_pointr);

Procedure: **soft_latch0**
Stores a measured value in Data Register 0.
Prototype: PROCEDURE soft_latch0 (pointr:g26_pointr);

Procedure: **soft_latch1**
Stores a measured value in Data Register 1.
Prototype: PROCEDURE soft_latch1 (pointr:g26_pointr);

Procedure: **read_reg0**
Reads a measured value from Data Register 0.
Prototype: PROCEDURE read_reg0 (pointr:g26_pointr);

Procedure: **read_reg1**
Reads a measured value from Data Register 1.
Prototype: PROCEDURE read_reg1 (pointr:g26_pointr);

Procedure: **poll_reg0**
Polls status bit D0 in register 0Eh until a measured value is stored in Data Register 0.
Prototype: PROCEDURE poll_reg0 (pointr:g26_pointr);

Procedure: **poll_reg1**
Polls status bit D1 in register 0Eh until a measured value is stored in Data Register 1.
Prototype: PROCEDURE poll_reg1 (pointr:g26_pointr);**Procedure: en_int**
Enables an interrupt.
Prototype: PROCEDURE en_int (Intrpt:byte);

Procedure: **dis_int**
Inhibits an interrupt.
Prototype: PROCEDURE dis_int (Intrpt:byte);

Procedure: **clear_int**
Resets an interrupt.
Prototype: PROCEDURE clear_int;

Procedure: **write_offset**
Writes offset values to the offset registers of the **counters** (volatile!). Use "store_offset" and "load_offset" for nonvolatile storage of the offset values and for storing them back in the offset registers.
Prototype: PROCEDURE write_offset
(baseadr:word;axis:byte;offx,offy:integer);

The following functions are defined in the file IIC.PAS. When compiling to a "unit", this file is included in IK121_1.PAS.

Procedure: **load_offset**
Reads the offset from the offset registers.
Prototype: PROCEDURE load_offset
(board:IK121_pointr;var error:boolean);

Procedure: **store_offset**
Stores a compensation value in the offset registers.
Prototype: PROCEDURE store_offset
(board:IK121_pointr;var error:boolean)

Procedure: **poti_default**
Sets the potentiometers to the zero position.
Prototype: PROCEDURE poti_default
(pointr:ik121_pointr;var error:boolean);

Function: **read_phasepoti**
Reads the setting of the potentiometer for the phase position.
Prototype: FUNCTION read_phasepoti
(pointr:ik121_pointr;axis:byte;var error:boolean):byte;

Function: **read_sympoti**
Reads the setting of the potentiometer for the symmetry.
Prototype: FUNCTION read_sympoti
(pointr:ik121_pointr;axis:byte;var error:boolean):byte;

Procedure: **write_phasepoti**
Sets the phase position potentiometer to a specific value.
Prototype: PROCEDURE write_phasepoti
(pointr:ik121_pointr;axis,wert:byte;var error:boolean);

Procedure: **write_sympoti**
Sets the potentiometer for the symmetry to a preset value.
Prototype: PROCEDURE write_sympoti
(pointr:ik121_pointr;axis,wert:byte;var error:boolean);

Procedure: **write_offset00**
Writes a compensation value to the offset register for the 0° signal.
Prototype: PROCEDURE write_offset00
(pointr:ik121_pointr;axis,value:integer);

Procedure: **write_offset90**
Writes a compensation value to the offset register for the 90° signal.
Prototype: PROCEDURE write_offset90
(pointr:ik121_pointr;axis,value:integer);

Procedure: **turn_phasepoti**
Adjusts the potentiometer for the phase position.
Prototype: PROCEDURE turn_phasepoti
(pointr:ik121_pointr;axis,turns:byte;updown:boolean;var error:boolean);

Procedure: **turn_sympoti**
Adjusts the potentiometer for the symmetry.
Prototype: PROCEDURE turn_sympoti
(pointr:ik121_pointr;axis,turns:byte;updown:boolean;var error:boolean);

Procedure: **turn_offsetdg00**
Changes the compensation value for the offset register for the 0° signal.
Prototype: PROCEDURE turn_offsetdg00
(pointr:ik121_pointr;axis,turns:byte;updown:boolean)

Procedure: **turn_offsetdg90**
Changes the compensation value for the offset register for the 90° signal.
Prototype: PROCEDURE turn_offsetdg90
(pointr:ik121_pointr;axis,turns:byte;updown:boolean)

Procedure: **store_potis**
Stores the potentiometer settings.
Prototype: PROCEDURE store_potis
(pointr:ik121_pointr;var error:boolean);

Procedure: **rom_write**
Writes to the vacant EEPROM memory.
Prototype: PROCEDURE rom_write
(pointr:ik121_pointr;adr;data :byte;var error : boolean);

Function: **rom_read**
Reads the vacant EEPROM memory.
Prototype: FUNCTION rom_read
(pointr:ik121_pointr;adr:byte;var error:boolean) :byte;

Applications programs with the RAM memory map in TURBO PASCAL

The executable EXE files in the following examples are located in the root directory of the supplied floppy disk 1. The source files are in subdirectory **\TP.** The programs require a BORLAND graphics interface (BORLAND Graphics Interface = *.BGI). The IK 121 includes the graphics interface EGAVGA.BGI. This interface must be located in the same directory as the example programs.

SAMPLE1.EXE

This is a simple application program for displaying the contents of Data Register 0 of axes 1 and 2.

Source code:	SAMPLE1.PAS	
Units:	IK121_0.TPU	Basic functions
	IK121_1.TPU	Functions for a RAM memory map

SAMPLE2.EXE

This program illustrates the use of interrupt programming. IRQ14 is used for Int1 and IRQ15 for Int0. An interrupt is triggered by a falling edge at X3.L0 or X3.L1.

Source code:	SAMPLE2.PAS	
Units:	IK121_0.TPU	Basic functions
	IK121_1.TPU	Functions for a RAM memory map

SAMPLE3.EXE

This program shows how the IK 121 can be used to measure speed. The speed and acceleration of axis 1 are displayed. The acceleration is calculated from speeds ds(1) and ds(2).

Source code:	SAMPLE3.PAS	
Units:	IK121_0.TPU	Basic functions
	IK121_1.TPU	Functions for a RAM memory map

SAMPLE4.EXE
Shows how the electronic potentiometers for phase position and amplitude as well as the offset registers can be read.

Source code:	SAMPLE4.PAS	
Units:	IK121_0.TPU	Basic functions
	IK121_1.TPU	Functions for a RAM memory map

SAMPLE5.EXE
This program shows the contents of Register 0 and Register 1 for axis 1. In addition, for Register 0 the contents of the period counter and the interpolation value are displayed separately. Various commands that you can enter on the keyboard are explained on the screen.

Source code:	SAMPLE5.PAS	
Units:	IK121_0.TPU	Basic functions
	IK121_1.TPU	Functions for a RAM memory map

SAMPLE6.EXE
SAMPLE6.EXE shows how the IK121 can be used as a period counter for axes 1 and 2 (i.e., the interpolated value is not evaluated). Various commands that you can enter on the keyboard are explained on the screen.

Source code:	SAMPLE6.PAS	
Units:	IK121_0.TPU	Basic functions
	IK121_1.TPU	Functions for a RAM memory map

SCOPE.EXE

This program displays the sinusoidal encoder signals either as in amplitude-time diagram or in XY representation. The potentiometers can be set with key commands that are explained on the screen.

Source code:	SCOPE.PAS	
Units:	IK121_0.TPU	Basic functions
	IK121_1.TPU	Functions for a RAM memory map
	IK121_2.TPU	Functions for ADJUST.EXE POTIS.EXE and SCOPE.EXE
	SCOPE_0.TPU	Functions for SCOPE.EXE
	CNT_0.TPU	Window functions
	LOGO.TPU	Initial image after program start

POTIS.EXE

This program shows the settings of the electronic potentiometers for phase position and amplitude as well as the values of the offset registers. The potentiometers can be set with key commands that are explained on the screen.

Source code:	POTIS.PAS	
Units:	IK121_0.TPU	Basic functions
	IK121_1.TPU	Functions for a RAM memory map
	IK121_2.TPU	Functions for ADJUST.EXE POTIS.EXE and SCOPE.EXE
	POTI_0.TPU	Functions for POTIS.EXE
	CNT_0.TPU	Window functions
	LOGO.TPU	Initial image after program start

ADJUST.EXE

ADJUST.EXE automatically adjusts axis 1 (selection:1) or axis 2 (selection:2) for phase position (selection:p), amplitude (selection:a) and offset (selection:o) of the sinusoidal encoder signals. The compensation values are calculated by slow movement of the encoder. After 30 signal periods, a tone signals that a compensation value is ready and can be stored by pressing the S key.

Source code:	ADJUST.PAS	
Units:	IK121_0.TPU	Basic functions
	IK121_1.TPU	Functions for a RAM memory map
	IK121_2.TPU	Functions for ADJUST.EXE, POTIS.EXE and SCOPE.EXE
	ADJ_0.TPU	Functions for ADJUST.EXE
	CNT_0.TPU	Window functions
	LOGO.TPU	Initial image after program start

IK121.EXE

This is an example of a position display program in TURBO PASCAL.

Source code:	IK121.PAS	
Units:	IK121_0.TPU	Basic functions
	IK121_1.TPU	Functions for a RAM memory map
	IK121_2.TPU	Functions for ADJUST.EXE, POTIS.EXE and SCOPE.EXE
	CNT_0.TPU	Window functions
	CNT_1.TPU	Position display
	CNT_2.TPU	Counter program
	SCOPE_0.TPU	Functions for SCOPE.EXE
	ADJ_0.TPU	Functions for ADJUST.EXE
	LOGO.TPU	Initial screen after program start
Include files:	IK121.WIN	Window definitions
	IK121-CNT	Initializing file
Help texts:	IK121.HLP	File with help texts

A pull-down menu can be opened by clicking with the mouse or by using the horizontal arrow keys and pressing the enter key. The following functions are then available:

File	Change and store settings for the card, the axes and the printer
Adjust	Display and adjust sinusoidal encoder signals
Counters	Set or reset axes, or start reference mark traverse
Sorting	Enter upper and lower limit values for sorting mode
Datum	Select reference plane. For both axes, four datum points available (datum 1 to datum 4)
Min/Max	Display minimum or maximum measured values of a measuring series
Print	Output measured value to the printer selected under "File/setupIK121/printer". If "File:on" is selected as the printer, the measured value will be output to the file IK121.DAT

The display fields next to the position displays have the following meanings:

signal	An encoder error has occurred
Ref	Reference mark traverse is active
<	Measured value is smaller than the lower limit value selected under "Sorting"
>	Measured value is larger than the upper limit value entered under "Sorting"

If you’re not sure how to proceed when using the IK 121 program, press F1 to display an explanation of the function.

Vacant EEPROM

The IK 121 board contains an EEPROM with 512 bytes of memory which is addressed via the I²C bus. The following programs are included on the supplied floppy disk 1 for writing and reading the EEPROM:

RDROM.EXE
Reads the contents of the EEPROM.

Source code:	RDROM.PAS	
Units:	IK 121_1.TPU	Functions for a RAM memory map

WRROM.EXE
Copies the file IK121.TXT into the EEPROM.

Source code:	RDROM.PAS	
Units:	IK 121_1.TPU	Functions for a RAM memory map

Applications examples with the RAM memory map in "BORLAND C++"

Examples with the RAM memory map in "BORLAND C++" can be found on the provided floppy disk 1 in the directory BCPP. The data structures and functions used are given and defined in the following files:

IK121.H:	You can set the addresses for up to 16 IK 121s in this header file.
IK121_1.H:	Data structures for a RAM memory map for the IK 121 registers and explanations of the functions in the files IK121_1.CPP, IIC.CPP and POTI_1.CPP
IK121_1.CPP:	Basic functions for the IK 121
IIC.CPP:	Functions for data transfer via the I²C-bus.
POTI_1.CPP:	Functions for setting the electronic potentiometer.

In the file IK121_1.H, a RAM memory map for the registers of the IK 121 is set up, with the help of data structures. The data for the map is written to the registers of the IK 121 with the help of the procedures **InitHandler** and **CommHandler.**

POTIS.EXE
The program POTIS.EXE shows how to set the electronic potentiometer of the IK 121 via the I²C-bus using software.

Source codes:	POTIS.CPP, IK121_1.CPP IIC.CPP, POTI_1.CPP
Header files:	IK 121.H, IK121_1.H

RDROM.EXE
The program RDROM.EXE reads the contents of the EEPROM.

Source codes:	RDROM.CPP, IK121_1.CPP, IIC.CPP
Headerfiles:	IK121.H, IK 121_1.H

WRROM.EXE
The program WRROM.EXE copies the file IK121.TXT into the EEPROM.

Source codes:	WRROM.CPP, IK121_1.CPP, IIC.CPP
Header files:	IK121.H, IK 121_1.H

DISPLAY.EXE
the program DISPLAY.EXE displays the contents of all the IK 121 registers.

Source codes:	DISPLAY.CPP, IK121_1.CPP, IIC.CPP
Header files:	IK121.H, IK 121_1.H

The IK 121 in WINDOWS Applications

Floppy disks 2 and 3 contain the driver software, Dynamic Link Libraries (DLL) and application examples in VISUAL C++, VISUAL BASIC and BORLAND DELPHI for WINDOWS NT and WINDOWS 95.

Disk 2

On floppy disk 2 you will find the following directory structure:

Install		**Installation files**
IK121Drv		**Device driver source for WINDOWS NT**
	Release	Release Version
IK121Dll		**Windows DLL source**
	Release	Release Version for Windows NT
	Release95	Release Version for Windows 95
IK121Con		**Example source for DOS applications**
	Release	Release Version for DOS applications
IK121App		**Example source for VISUAL C++**
	Res	Resources for Windows example
	Release	Release version for Windows example
IK121VB5		**Example source for VISUAL BASIC**
Include		**Include files**
Install		**Installation files**

Disk 3

Floppy disk 3 contains examples for BORLAND DELPHI.

Device driver for Windows NT (IK121DRV.SYS)

On disk 2 in directory **\IK121Drv** you will find the kernel mode device driver for WINDOWS NT (Versions 3.51 and 4.0). It enables access to the IK 121.

To enable Windows NT to load the driver, you must first copy it into the Windows NT system directory under **\System32\Drivers** (e.g.: C:\WINNT\SYSTEM32\DRIVERS). The batch file **Install.Bat** will copy the driver for you.

Registry entry

In the registry the driver finds the port address at which the IK 121 is installed. The following registry entry is required:

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

In the directory **\Install** on disk 2 are the files **IK121Drv.Reg** and **SetReg.Bat.** The file **IK121Drv.Reg** contains the data specified above, which you add to the registry with **Install.Bat.**

In the file **IK121Drv.Reg** you can change the base address of the IK 121 and/or add more base addresses. The driver supports up to eight IK 121. The entry in the registry must be updated with **SetReg.Bat.**

The Windows DLLs (IK121Dll.Dll)

This DLL enables applications programs to address the IK 121. There is one DLL each for Windows NT and for Windows 95. The DLL for Windows NT accesses the IK 121 indirectly through the device driver for Windows NT. The DLL for Windows 95 accesses the registry of the IK 121 directly. Frequently required functions are directly available (start, stop, read out counter value, REF functions, etc.). Additional programming of the IK 121 is enabled through the access to its registry (IKInputW, IKInputL, IKOutput etc.). To enable Windows NT application programs to load the DLL, the file **\IK121Dll\Release\IK121Dll.Dll** on disk 2 must be placed in the system directory under **System 32** (e.g. **C:\Winnt\System32**). For Windows 95, the file **\IK121Dll\Release95\IK121Dll.Dll** on disk 2 must be placed in the **System** directory (e.g.: **C:\Windows\System**). The batch file Install.Bat copies the required file into its respective directory.

Example for console application

In the directory **\IK121Con\Release** of floppy disk 2 you will find a simple console application.

Example for VISUAL C++

In the directory **\IK121App\Release** of floppy disk 2 you will find an application in VISUAL C++.

Example for VISUAL BASIC

In the directory **\IK121VB5\Release** of floppy disk 2 you will find an application in VISUAL BASIC.

Example for BORLAND DELPHI

On floppy disk 3 you will find an application in BORLAND DELPHI.

Installation of the driver and the DLL under WINDOWS NT and WINDOWS 95

➤ Select the directory **\Install** on the provided floppy disk 2.
➤ Enter the port addresses of the installed IK 121 in the file **IK121Drv.Reg.**
➤ Call **Install.Bat.**

Install.bat generates the entry in the registry, copies the driver for WINDOWS NT from the directory **\IK121Drv\Release** into the system directory (e.g. **C:\Winnt\System32\Drivers**), and copies the DLL for WINDOWS NT (or WINDOWS 95) from the directory **\IK121Dll\Release** (or \IK121Dll\Release95) into the system directory (e.g. **C:\Winnt\System32**).

Calling DLL functions from your own application programs

To be able to use the functions of the DLL you must identify them to the application program.

MICROSOFT VISUAL C++

If you write the application program in VISUAL C++, then the file **\IK121Dll\Release\IK121Dll.Lib** from floppy disk 2 must be copied into the library directory of VISUAL C++ (e.g. **C:\Msdev\lib**) and linked into the project. To link the file, make an entry in VISUAL C++ under **Build, Settings, Link, Object/Library modules.**

In a header file you must also define the following prototypes:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

WINUSERAPI BOOL WINAPI IKDefine (ULONG* pBuffer8);

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

Then you can use the functions like ”normal” C functions.

MICROSOFT VISUAL BASIC

In VISUAL BASIC you can define the functions in a module as follows:

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

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

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

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

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

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

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

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

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

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

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

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

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

Public Declare Function IKSetI2C          Lib "IK121DLL.DLL"
                                          (ByVal Card As Integer, ByVal SCL As Boolean, ByVal
                                          SDA As Boolean) As Boolean
Public Declare Function IKDefine          Lib "IK121DLL.DLL"
                                          (ByRef pBuffer8 As Long) As Boolean
```

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

BORLAND DELPHI

In BORLAND DELPHI you can link the functions into your program as follows:

Function IKFind	(pBuffer8: Long8Ptr) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKInit	(Axis: Word; Mode: Word) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKVersion	(Axis: Word; pVersCard: CharPtr; pVersDrv: CharPtr; pVersDll: CharPtr) : Boolean; StdCall; External 'IK121DLL.DLL';
Function IKReset	(Axis: Word) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKStart	(Axis: Word) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKStop	(Axis: Word) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKClear	(Axis: Word) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKLatch	(Axis: Word; Latch: Word) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKResetREF	(Axis: Word) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKStartREF	(Axis: Word) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKStopREF	(Axis: Word) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKLatchREF	(Axis: Word; Latch: Word) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKLatched	(Axis: Word; Latch: Word; pStatus: BoolPtr) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKWaitLatch	(Axis: Word; Latch: Word) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKStrtCodRef	(Axis: Word; Latch: Word; RefDist: LongInt) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKCodRef	(Axis: Word; pStatus: BoolPtr; pData: DoublePtr) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKWaitCodRef	(Axis: Word; pData: DoublePtr) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKStopCodRef	(Axis: Word) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKStatus	(Axis: Word; pStatus: LongPtr) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKRead32	(Axis: Word; Latch: Word; pData: LongPtr) : Boolean; StdCall;

	External ’IK121DLL.DLL’;
Function IKRead48	(Axis: Word; Latch: Word; pData: DoublePtr) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKReadPhase	(Axis: Word; pData: BytePtr) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKWritePhase	(Axis: Word; Data: Byte) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKLoadPhase	(Axis: Word; pData: BytePtr) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKReadAmp	(Axis: Word; pData: BytePtr) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKWriteAmp	(Axis: Word; Data: Byte) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKLoadAmp	(Axis: Word; pData: BytePtr) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKReadOffset	(Axis: Word; Ofs0: IntPtr; Ofs90: IntPtr) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKWriteOffset	(Axis: Word; Ofs0: Integer; Ofs90: Integer) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKLoadOffset	(Axis: Word; Ofs0: IntPtr; Ofs90: IntPtr) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKStore	(Axis: Word) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKDefault	(Axis: Word) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKRomRead	(Card: Word; Adr: Byte; Data: BytePtr) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKRomWrite	(Card: Word; Adr: Byte; Data: Byte) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKInputW	(Axis: Word; Adr: Word; pData: WordPtr) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKInputL	(Axis: Word; Adr: Word; pData: LongPtr) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKOutput	(Axis: Word; Adr: Word; Data: LongInt) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKSetI2C	(Card: Word; SCL: Boolean; SDA: Boolean) : Boolean; StdCall; External ’IK121DLL.DLL’;
Function IKDefine	(pBuffer8: Long8Ptr) : Boolean; StdCall; External 'IK121DLL.DLL';
Function IKSetTimer	(Axis: Word; SetVal: Word) : Boolean; StdCall; External 'IK121DLL.DLL';
Function IKEnableLatch	(Axis: Word; Latch: Word; Source: Word) : Boolean; StdCall; External 'IK121DLL.DLL';
Function IKEnableSync	(Card: Word; Source: Word) : Boolean; StdCall; External 'IK121DLL.DLL';
Function IKLatchAll	(Axis: Word) : Boolean; StdCall; External 'IK121DLL.DLL';

Overview of DLL functions

Function	Short Reference
Find the installed IK 121	BOOL IKFind (ULONG* pBuffer8);
Initialize the IK 121	BOOL IKInit (USHORT Axis, USHORT Mode);
Read program version	BOOL IKVersion (USHORT Axis, char* pVersCard, char* pVersDrv, char* pVersDll);
Reset the counter	BOOL IKReset (USHORT Axis);
Start the counter	BOOL IKStart (USHORT Axis);
Stop the counter	BOOL IKStop (USHORT Axis);
Erase the frequency and amplitude errors	BOOL IKClear (USHORT Axis);
Save the counter value	BOOL IKLatch (USHORT Axis, USHORT Latch);
Reset the counter with next reference mark	BOOL IKResetREF (USHORT Axis);
Start the counter with next reference mark	BOOL IKStartREF (USHORT Axis);
Stop the counter with the next reference mark	BOOL IKStopREF (USHORT Axis);
Save the counter value with the next reference mark	BOOL IKLatchREF (USHORT Axis, USHORT Latch);
Inquiry whether the counter value has been saved	BOOL IKLatched (USHORT Axis, USHORT Latch, BOOL* pStatus);
Wait until the counter value has been saved	BOOL IKWaitLatch (USHORT Axis, USHORT Latch);
Start a reference run with distance-coded reference marks	BOOL IKStrtCodRef (USHORT Axis, USHORT Latch, ULONG RefDist);
Inquiry whether the reference run with distance-coded reference mark has ended	BOOL IKCodRef (USHORT Axis, BOOL* pStatus, double* pData);
Wait until the reference run with distance-coded reference marks has ended	BOOL IKWaitCodRef (USHORT Axis, double* pData);
Stop the reference run with distance-coded reference marks	BOOL IKStopCodRef (USHORT Axis);
Status inquiry	BOOL IKStatus (USHORT Axis, ULONG* pStatus);
Read latched counter values (32 bits)	BOOL IKRead32 (USHORT Axis, USHORT Latch, LONG* pData);

Function	Short Reference
Read latched counter values (48 bits)	BOOL IKRead48 (USHORT Axis, USHORT Latch, double* pData);
Read phase compensation value	BOOL IKReadPhase (USHORT Axis, BYTE* pData);
Change phase compensation value	BOOL IKWritePhase (USHORT Axis, BYTE Data);
Read latched phase compensation value	BOOL IKLoadPhase (USHORT Axis, BYTE* pData);
Read amplitude compensation value	BOOL IKReadAmp (USHORT Axis, BYTE* pData);
Change amplitude compensation value	BOOL IKWriteAmp (USHORT Axis, BYTE Data);
Read latched amplitude compensation values	BOOL IKLoadAmp (USHORT Axis, BYTE* pData);
Read offset compensation value	BOOL IKReadOffset (USHORT Axis, SHORT* Ofs0, SHORT* Ofs90);
Change offset compensation value	BOOL IKWriteOffset (USHORT Axis, SHORT Ofs0, SHORT Ofs90);
Read offset compensation value	BOOL IKLoadOffset (USHORT Axis, SHORT* Ofs0, SHORT* Ofs90);
Save compensation values	BOOL IKStore (USHORT Axis);
Load and save neutral values	BOOL IKDefault (USHORT Axis);
Read values from EEPROM	BOOL IKRomRead (USHORT Card, BYTE Adr, BYTE* Data);
Write values to EEPROM	BOOL IKRomWrite (USHORT Card, BYTE Adr, BYTE Data);
Read IK121 register (16 bits)	BOOL IKInputW (USHORT Axis, USHORT Adr, USHORT* pData);
Read IK121 register (32 bits)	BOOL IKInputL (USHORT Axis, USHORT Adr, ULONG* pData);
Write IK121 register (16 bits)	BOOL IKOutput (USHORT Axis, USHORT Adr, USHORT Data);
Set I^{2}C lines	BOOL IKSetI2C (USHORT Card, BOOL SCL, BOOL SDA);
Define port addresses of the IK 121	BOOL IKDefine (ULONG* pBuffer8);
Set timer value	BOOL IKSetTimer (USHORT Axis, USHORT SetVal);
Enable latch signal	BOOL IKEnableLatch (USHORT Axis, USHORT Latch, USHORT Source);
Define cascading	BOOL IKEnableSync (USHORT Card, USHORT Source);
Latch position value of all axes	BOOL IKLatchAll (USHORT Axis);

Reference of the DLL functions

All DLL functions return a Boolean variable:
true (<> 0) if the function was successfully completed
false (= 0) if an error occurred

IKFind

This function supplies the port address of the installed IK 121. Unused entries are set to 0.

Prototype:	**BOOL IKFind (ULONG* pBuffer[8]);**
pBuffer:	Pointer to 8 long words (4 bytes)

IKInit

This function initializes the IK 121.

Prototype:	**BOOL IKInit (USHORT Axis, USHORT Mode);**
Axis:	Number of the axis (0 to 15)
Mode:	0=32-bit counter value 1=48-bit counter value

IKVersion

This function reads the program versions of 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 IKVersion (USHORT Axis, char* pVersCard, char* pVersDrv, char* pVersDll)**
Axis:	Number of the axis (0 to 15)
pVersCard:	Pointer to the version of the IK 121.
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

IKReset

This function sets the counter to zero.

Prototype:	**BOOL IKReset (USHORT Axis);**
Axis:	Number of the axis (0 to 15)

IKStart

This function starts the counter.

Prototype:	**BOOL IKStart (USHORT Axis);**
Axis:	Number of the axis (0 to 15)

IKStop

This function stops the counter.

Prototype:	**BOOL IKStop (USHORT Axis);**
Axis:	Number of the axis (0 to 15)

IKClear

This function deletes the error status.

Prototype:	**BOOL IKClear (USHORT Axis);**
Axis:	Number of the axis (0 to 15)

IKLatch

This function saves the counter value.

Prototype:	**BOOL IKLatch (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

IKResetREF

This function sets the counter to zero at the next reference mark.

Prototype:	**BOOL IKResetREF (USHORT Axis);**
Axis:	Number of the axis (0 to 15)

IKStartREF

This function starts the counter with the next reference mark.

Prototype:	**BOOL IKStartREF (USHORT Axis);**
Axis:	Number of the axis (0 to 15)

IKStopREF

This function stops the counter with the next reference mark.

Prototype:	**BOOL IKStopREF (USHORT Axis);**
Axis:	Number of the axis (0 to 15)

IKLatchREF

This function saves the counter value when the reference mark is passed over.

Prototype:	**BOOL IKLatchREF (USHORT Axis, USHORT Latch);**
Axis:	Number of the axis (0 to 15)

IKLatched

This function determines whether the counter value was saved.
Application: Before a counter value can be read out, the application program must inquire whether the counter value has been saved.

Prototype:	**BOOL IKLatched (USHORT Axis, USHORT Latch, BOOL* pStatus);**
Axis:	Number of the axis (0 to 15)
Latch:	0 = inquiry for register 0 1 = inquiry for register 1
pStatus:	Pointer to a Boolean variable (16 bits) false (= 0) = value not saved true (<> 0) = value saved

IKWaitLatch

This function waits until the counter value was saved.
Application: Before a counter value can be read out, the application program must inquire whether the counter value has been saved.

Prototype:	**BOOL IKWaitLatch (USHORT Axis, USHORT Latch);**
Axis:	Number of the axis (0 to 15)
Latch:	0 = inquiry for register 0 1 = inquiry for register 1

IKStrtCodRef

This function initializes the reference run with distance-coded reference marks. After initialization the application program must cyclically scan (function: IKCodRef) or wait (function: IKWaitCodRef), until the distance-coded reference mark run is completed.

Prototype:	**BOOL IKStrtCodRef (USHORT Axis, USHORT Latch, ULONG RefDist);**
Axis:	Number of the axis (0 to 15)
Latch:	0 = with register 0 1 = with register 1
RefDist:	fixed spacing of reference marks (e.g. 500, 1000, 2000, 5000)

IKCodRef

During a reference run with distance-coded reference marks this function ascertains whether the second reference mark has been traversed and reports the offset value. The offset value must be added to the counter value in order to obtain the absolute position. The application program must call this function cyclically after starting the reference mark run. (It can also, however, wait for the end. The function for waiting is IKWaitCodRef.)

Prototype:	**BOOL IKCodRef (USHORT Axis, BOOL* pStatus, double* pData);**
Axis:	Number of the axis (0 to 15)
pStatus:	Pointer to a Boolean variable (16 bits). False (= 0) = reference run not completed. True (<> 0) = reference run completed.
pData:	Pointer to a ”double variable” (64 bits), in which the offset value is saved (only if pStatus=TRUE).

IKWaitCodRef

This function waits until the reference run with distance-coded reference marks is completed. After the second reference mark has been traversed the offset value is returned.

Prototype:	**BOOL IKWaitCodRef (USHORT Axis, double* pData);**
Axis:	Number of the axis (0 to 15)
pData:	Pointer to a ”double variable” (64 bits), in which the offset value is filed.

IKStopCodRef

This function interrupts a reference run over distance-coded reference marks.

Prototype:	**BOOL IKStopCodRef (USHORT Axis);**
Axis:	Number of the axis (0 to 15)

IKStatus

This function reports the status of the IK 121.

Prototype:	**BOOL IKStatus (USHORT Axis, ULONG* pStatus);**
Axis:	Number of the axis (0 to 15)
pStatus:	Pointer to a long word (32 bits)

Bit	Function
D0	1 = counter value saved in register 0
D1	1 = counter value saved in register 1
D2	*No function*
D3	*No function*
D4	1 = counter stopped
D5	*No function*
D6	1 = maximum frequency exceeded
D7	*No function*
D8	1 = REF function active
D9	1 = counter is started with next reference mark[1)]
D10	1 = counter is stopped with next reference mark[1)]
D11	1 = counter is reset to 0 with next reference mark[1)]
D12	1 = counter is saved with next reference mark[1)]
D13	1 = counter is saved with the reference mark after the next one[1)]
D14	1 = counter is reset to 0 with every reference mark[1)]
D15	*No function*
D16	*No function*
D17 D18	Present amplitude 00 = normal amplitude (4.5 µA < U_e < 15 µA) 01 = low amplitude (U_e < 4.5 µA) 10 = high amplitude (U_e > 15 µA) 11 = unacceptably low amplitude (U_e < 2.5 µA)
D19 D20	Minimum amplitude 00 = normal amplitude (4.5 µA < U_e < 15 µA) 01 = low amplitude (U_e < 4.5 µA) 10 = high amplitude (U_e > 15 µA) 11 = unacceptably low amplitude (U_e < 2.5 µA)
D21 D22	Maximum amplitude [1)] 00 = normal amplitude (4.5 µA < U_e < 15 µA) 01 = small amplitude (U_e < 4.5 µA) 10 = high amplitude (U_e > 15 µA) 11 = unacceptably low amplitude (U_e < 2.5 µA
D23	*No function*
D24 to D31	IC code of the counter chip (08 or 09)

1) only when IC code = 09

IKRead32

This function supplies the 32-bit counter value. Before the counter value can be read out it must be saved in register 0 or register 1 (IKLatch, IKLatchREF), and the program must inquire whether the saving process has been completed (IKLatched, IKWaitLatch).

Prototype:	**BOOL IKRead32 (USHORT Axis, USHORT Latch, LONG* pData);**
Axis:	Number of the axis (0 to 15)
Latch:	0 = read out from register 0 1 = read out from register 1
pData:	Pointer to a long word (32 bits), in which the counter value is saved

IKRead48

This function supplies the 48-bit counter value. Before the counter value can be read out it must be saved in register 0 or register 1 (IKLatch, IKLatchREF), and the program must inquire whether the saving process has been completed (IKLatched, IKWaitLatch).

Prototype:	**BOOL IKRead48 (USHORT Axis, USHORT Latch, double* pData);**
Axis:	Number of the axis (0 to 15)
Latch:	0 = read out from register 0 1 = read out from register 1
pData:	Pointer to a ”double variable” (64 bits), in which the counter value is stored

IKReadPhase

This function reads the present setting of the phase-compensation potentiometer

Prototype:	**BOOL IKReadPhase (USHORT Axis, BYTE* pData);**
Axis:	Number of the axis (0 to 15)
pData:	Pointer to a ”byte variable” (8 bits) in which the phase compensation is saved

IKWritePhase

This function changes the instantaneous setting of the phase compensation.

Prototype:	**BOOL IKWritePhase (USHORT Axis, BYTE Data);**
Axis:	Number of the axis (0 to 15)
Data:	New value of the phase compensation (0 to 63)

IKLoadPhase

This function reports the nonvolatile saved value of the phase compensation.

Prototype:	**BOOL IKLoadPhase (USHORT Axis, BYTE* pData);**
Axis:	Number of the axis (0 to 15)
pData:	Pointer to a ”byte variable” (8 bits) in which the phase compensation is saved.

IKReadAmp

This function reports the instantaneous setting of the amplitude compensation.

Prototype:	**BOOL IKReadAmp (USHORT Axis, BYTE* pData);**
Axis:	Number of the axis (0 to 15)
pData:	Pointer to a ”byte variable” (8 bits) in which the amplitude compensation is saved.

IKWriteAmp

This function changes the instantaneous setting of the amplitude compensation.

Prototype:	**BOOL IKWriteAmp (USHORT Axis, BYTE Data);**
Axis:	Number of the axis (0 to 15)
Data:	New value of the amplitude compensation (0 to 63)

IKLoadAmp

This function reports the nonvolatile saved value of the amplitude compensation.

Prototype:	**BOOL IKLoadAmp (USHORT Axis, BYTE* pData);**
Axis:	Number of the axis (0 to 15)
pData:	Pointer to a ”byte variable” (8 bits) in which the amplitude compensation is saved.

IKReadOffset

This function reports the instantaneous setting of the offset compensation.

Prototype:	**BOOL IKReadOffset (USHORT Axis, SHORT* Ofs0, SHORT* Ofs90);**
Axis:	Number of the axis (0 to 15)
Ofs0:	Pointer to a ”short variable” (16 bits) in which the offset compensation of the 0° signal is saved.
Ofs90:	Pointer to a ”short variable” (16 bits) in which the offset compensation of the 90° signal is saved.

IKWriteOffset

This function changes the instantaneous setting of the offset compensation.

Prototype:	**BOOL IKWriteOffset (USHORT Axis, SHORT Ofs0, SHORT Ofs90);**
Axis:	Number of the axis (0 to 15)
Ofs0:	New value of the offset compensation of the 0° signal (–63 to +63)
Ofs90:	New value of the offset compensation of the 90° signal (–63 to +63)

IKLoadOffset

This function reports the nonvolatile saved value of the offset compensation.

Prototype:	**BOOL IKLoadOffset (USHORT Axis, SHORT* Ofs0, SHORT* Ofs90);**
Axis:	Number of the axis (0 to 15)
Ofs0:	Pointer to a ”short variable” (16 bits) in which the offset compensation of the 0° signal is saved.
Ofs90:	Pointer to a ”short variable” (16 bits) in which the offset compensation of the 90° signal is saved.

IKStore

This function transfers all instantaneous compensation values into a nonvolatile memory. The IK 121 activates the phase and amplitude compensation values automatically when the PC is switched on. The offset compensation values are activated when the IK 121 is initialized (function: IKInit).

Prototype:	**BOOL IKStore (USHORT Axis);**
Axis:	Number of the axis (0 to 15)

IKDefault

This function sets all compensation to neutral values (phase=31, Amplitude=31 and offset=0). This state is taken into the nonvolatile memory.

Prototype:	**BOOL IKDefault (USHORT Axis);**
Axis:	Number of the axis (0 to 15)

IKRomRead

This function reads an 8-bit value from the EEPROM.

Prototype:	**BOOL IKRomRead (USHORT Card, BYTE Adr, BYTE*pData);**
Card:	Number of the IK 121 (0 to 7)
Adr:	Address in the EEPROM (0 to 255)
pData:	Pointer to a "byte variable" (8 bit) in which the value is filed.

IKRomWrite

This function writes an 8-bit value into the EEPROM.

Prototype:	**BOOL IKRomWrite (USHORT Card, BYTE Adr, BYTE Data);**
Card:	Number of the IK 121 (0 to 7)
Adr:	Address in the EEPROM (0 to 255)
Data:	Value (8 bits) that is saved in the EEPROM.

IKInputW

This function reads a word in a register.

Prototype:	**BOOL IKInputW (USHORT Axis, USHORT Adr, USHORT* pBuffer);**
Axis:	Number of the axis (0 to 15)
Adr:	Address of the register (0 to 30 or 0 to 0x1E)
pBuffer:	Pointer to a word (16 bits) in which the read value is filed.

IKInputL

This function reads the long word of a register.

Prototype:	**BOOL IKInputL (USHORT Axis, USHORT Adr, ULONG* pBuffer);**
Axis:	Number of the axis (0 to 15)
Adr:	Address of the register (0 to 28 or 0 to 0x1C)
pBuffer:	Pointer to a long word (32 bits) in which the read value is saved.

IKOutput
This function writes a word in a register.

Prototype:	**BOOL IKOutput (USHORT Axis, USHORT Adr, USHORT Data);**
Axis:	Number of the axis (0 to 15)
Adr:	Address of the register (0 to 30 or 0 to 0x1E)
Data:	Value (16 bits) that is written into the register.

IKSetI2C
This function sets the lines of the I^{2}C bus, i.e. one can set or reset the DATA and CLOCK lines. This makes it possible to directly address the potentiometer and the EEPROM.

Prototype:	**BOOL IKSetI2C (USHORT Card, BOOL SCL, BOOL SDA);**
Card:	Number of the IK 121 (0 to 7)
SCL:	Status of the CLOCK line FALSE(=0)= Low TRUE(<>0)=High
SDA:	Status of the DATA line FALSE(=0)= Low TRUE(<>0)=High

IKDefine
This function defines the port addresses of the installed IK 121. For every IK 121, the port address must be saved at the corresponding position in pBuffer8. The unused entries must be set to 0. **This applies only to Win32s under Windows 3.1/3.11 because it has no registry!**

Prototype:	**BOOL IKDefine (ULONG* pBuffer8);**
pBuffer8:	Pointer to 8 long words (8*4 bytes)

IKSetTimer
This function defines the time interval of the timer.

Prototype:	**BOOL IKSetTimer (USHORT Axis, USHORT SetVal);**	
Axis:	Number of the axis (0 to 15)	
SetVal	0	= Timer is stopped
	1 to 8192	= Timer value in micro seconds

IKEnableLatch
This function defines which latch signal stores the position value.

Prototype:	**BOOL IKEnableLatch (USHORT Axis, USHORT Latch, USHORT Source);**
Axis:	Number of the axis (0 to 15)
Latch:	0 = Enable latch signal for register 0
	1 = Enable latch signal for register 1
Source:	0 = All latch signals are disabled
	1 = Enable external signal without delay (slave)
	2 = Enable external signal with delay (master)
	3 = Enable software latch
	4 = Enable timer

IKEnableSync
This function defines which latch signal is being transmitted to the 2nd axis.

Prototype:	**BOOL IKEnableSync (USHORT Card, USHORT Source);**
Card:	Number of the IK 121 (0 to 7)
Source:	0 = No cascading
	1 = Cascading external signal
	2 = Cascading software latch
	3 = Cascading timer

IKLatchAll
This function enables a software latch, which makes it possible to save the position values of several axes.

Prototype:	**BOOL IKLatchAll (USHORT Axis);**
Axis:	Number of the axis (0 to 15)

Specifications

Mechanical Data	
Dimensions	158 mm × 107 mm
Operating temp.	0 °C to 45 °C (32 °F to 113 °F)
Storage temp.	–30 °C to 70 °C (–22 °F to 158 °F)

Electrical Data	
Inputs/Outputs	
External latch signals:	X3: 4-pin flange socket
2 inputs:	U_H: 3.15 V to 30 V U_L:–3.0 V to 0.9 V
1 output:	U_H: 4.0 V to 32 V U_L: 0 V to 1.0 V
Encoder outputs:	Sinusoidal current signals (11 μA_{PP}) via additional assembly for each axis
IK 121 A	
Encoder inputs:	X1: Axis 1, 9-pin D-sub connection; sinusoidal signals: 7 μA_{PP} to 16 μA_{PP} X2: Axis 2, 9-pin D-sub connection; sinusoidal signals: 7 μA_{PP} to 16 μA_{PP}
Input frequency:	Max. 100 kHz
Cable length:	Max. 10 meters
IK 121 V	
Encoder inputs:	X1: Axis 1, 15-pin D-sub connection; sinusoidal signals: 0.6 V_{PP} to 1.2 V_{PP} X2: Axis 2, 15-pin D-sub connection; sinusoidal signals: 0.6 μV_{PP} to 1.2 μV_{PP}
Input frequency:	Max. 400 kHz
Cable length:	Max. 30 meters

Cables lengths of up to 150 m are possible if it can be guaranteed that the encoder will be supplied by 5 V from an external power source. In this case the input frequency is reduced to 250 kHz.

Signal interpolation	1024-fold

Adjustment of encoder signals	Phase and amplitude adjusted with electronic potentiometers Offset adjusted with registers in the counters
Data register for measured values	48 bits
Port addresses	Selected with DIP switches; the IK 121 occupies 16 addresses
Interrupts	IRQ5, IRQ9, IRQ10, IRQ11, IRQ12 or IRQ15
Power consumption	1 W approx., without encoders
Software	
Driver software and demonstration programs	As programming aid for: **DOS applications** in ”TURBO PASCAL” and ”BORLAND C++” **WINDOWS NT / 95** in VISUAL C++, VISUAL BASIC and BORLAND DELPHI

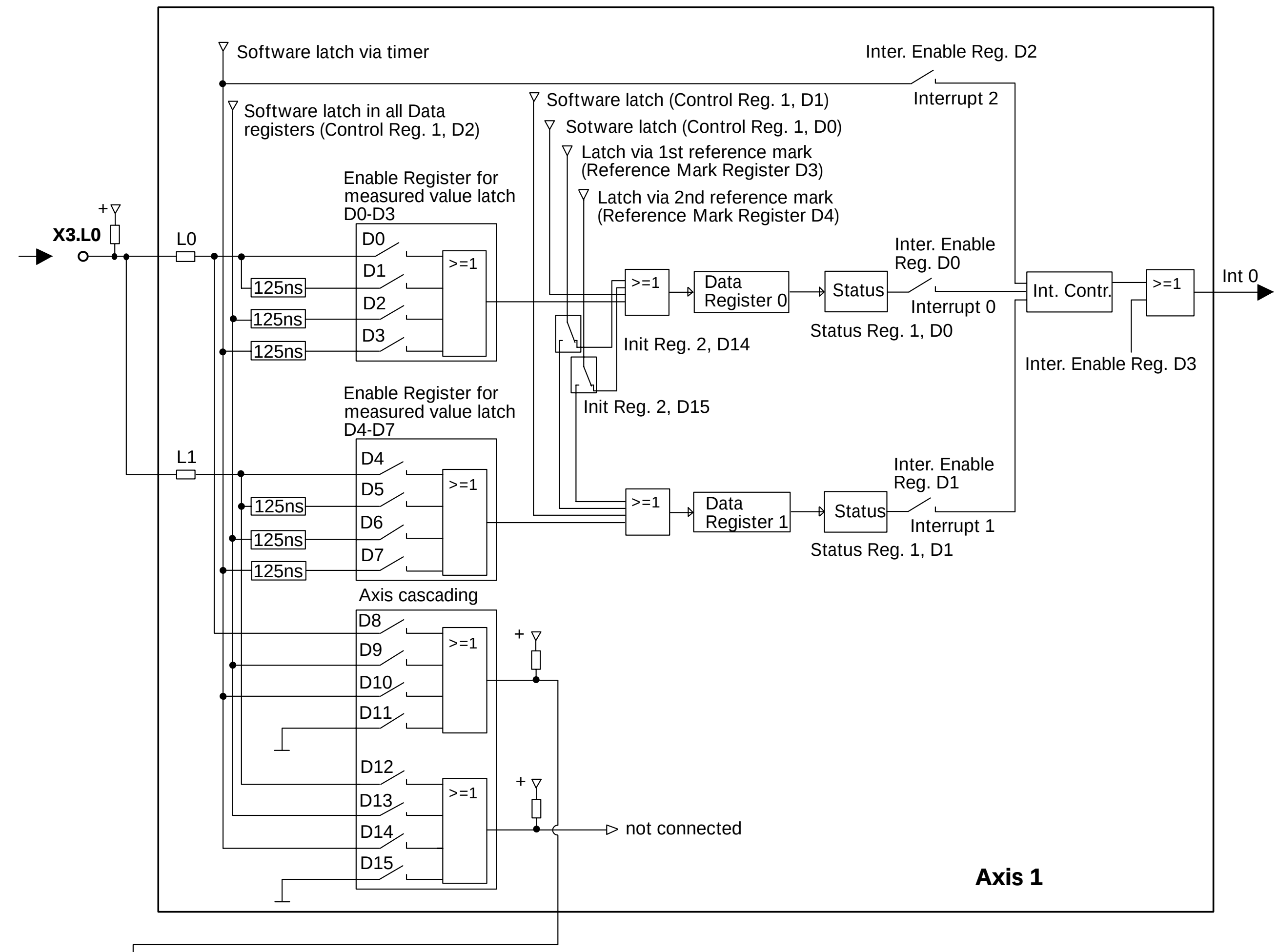
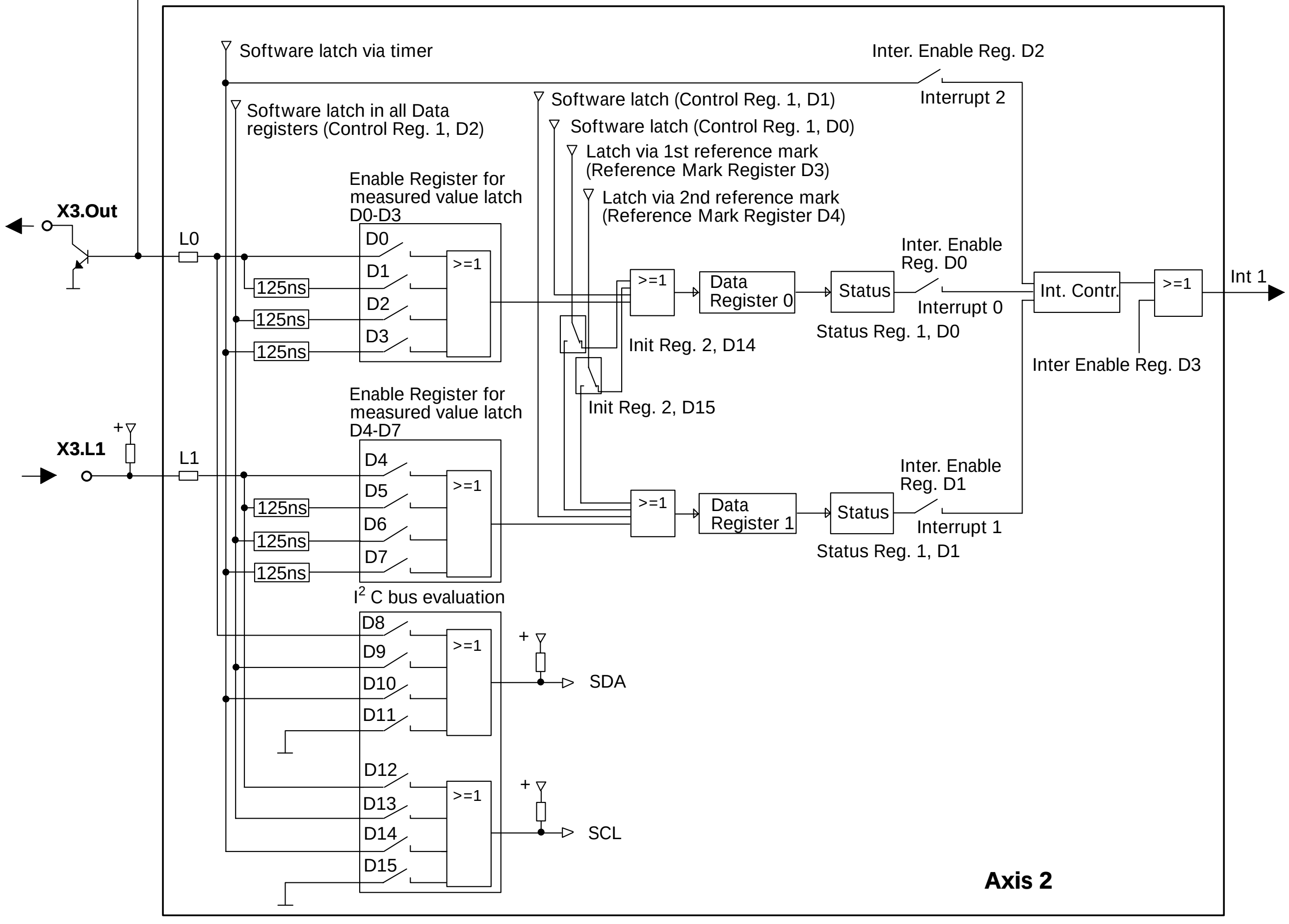
Subject Index

Basic circuit diagram of the latch paths in the counters

The circuit diagram on the following page shows:

- The effect of the latch signals on the data registers
- The function of the individual bits of the enable register for measured value latching
- The axis cascading with the associated bits of the register with the same name
- Formation of the interrupts
- The register for I²C bus control

The delay circuits with 125 nanoseconds delay time are used with synchronous latching of both axes to balance the signal transit times between axes 1 and 2. With synchronous latching, therefore, the signal for axis 1 should be routed through a delay circuit, while for axis 2 it should not. Since the same counter is used for axes 1 and 2, delay circuits are present in each axis (i.e., not all signal path combinations will result in a proper latch).


Software latch via timer
Inter. Enable Reg. D2
Interrupt 2
Software latch in all Data registers (Control Reg. 1, D2)
Software latch (Control Reg. 1, D1)
Sotware latch (Control Reg. 1, D0)
Latch via 1st reference mark (Reference Mark Register D3)
Latch via 2nd reference mark (Reference Mark Register D4)
Enable Register for measured value latch D0-D3
X3.L0
L0
D0
D1
D2
D3
125ns
>=1
Data Register 0
Status
Inter. Enable Reg. D0
Interrupt 0
Status Reg. 1, D0
Int. Contr.
Int 0
Inter. Enable Reg. D3
Init Reg. 2, D14
Init Reg. 2, D15
Enable Register for measured value latch D4-D7
L1
D4
D5
D6
D7
Data Register 1
Inter. Enable Reg. D1
Interrupt 1
Status Reg. 1, D1
Axis cascading
D8
D9
D10
D11
D12
D13
D14
D15
not connected
Axis 1

Software latch via timer
Inter. Enable Reg. D2
Interrupt 2
Software latch in all Data registers (Control Reg. 1, D2)
Software latch (Control Reg. 1, D1)
Software latch (Control Reg. 1, D0)
Latch via 1st reference mark (Reference Mark Register D3)
Latch via 2nd reference mark (Reference Mark Register D4)
Enable Register for measured value latch D0-D3
X3.Out
L0
D0
D1
D2
D3
125ns
>=1
Data Register 0
Status
Inter. Enable Reg. D0
Interrupt 0
Status Reg. 1, D0
Int. Contr.
Int 1
Inter Enable Reg. D3
Init Reg. 2, D14
Init Reg. 2, D15
Enable Register for measured value latch D4-D7
X3.L1
L1
D4
D5
D6
D7
Data Register 1
Inter. Enable Reg. D1
Interrupt 1
Status Reg. 1, D1
I2 C bus evaluation
D8
D9
D10
D11
D12
D13
D14
D15
SDA
SCL
Axis 2

HEIDENHAIN

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

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

www.heidenhain.de

Ve 00
291 019-25 · 4 · 5/2004 · F&W · Printed in Germany · Subject to change without notice