HEIDENHAIN

User's Manual

IK 320

VMEbus Counter Card

7/2004

Validity

This manual describes the IK 320 beginning with software version

246 118-08

Table of Contents

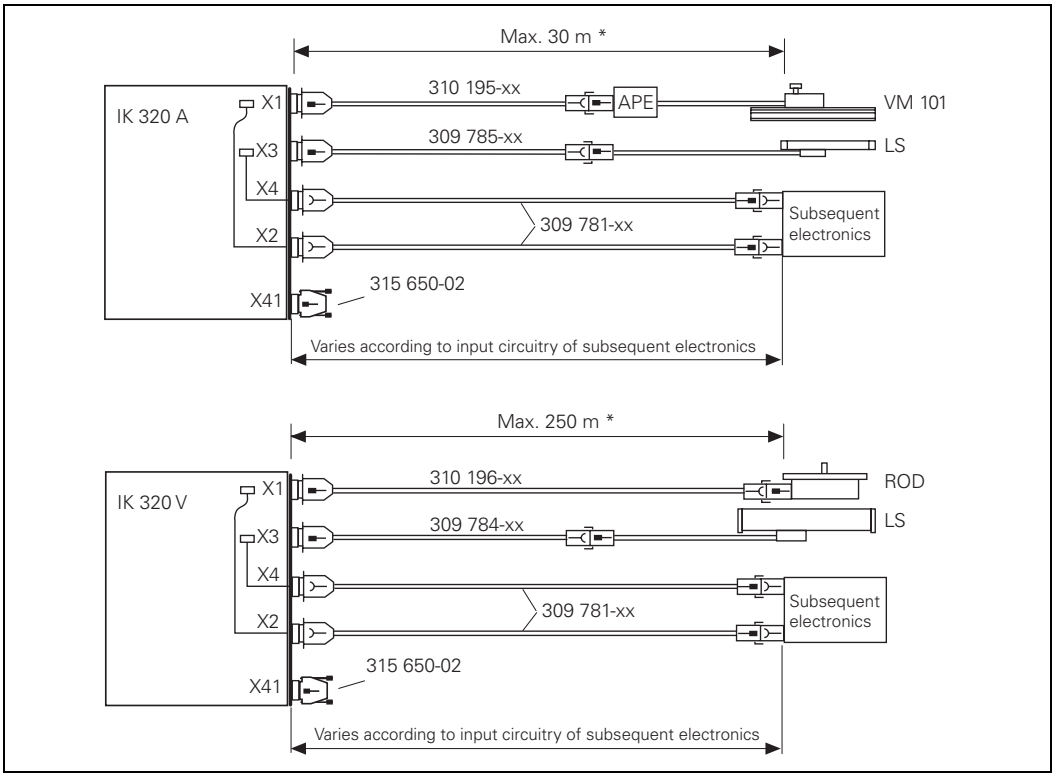
1. Items Supplied

IK 320 VMEbus counter card, driver software and User's Manual

1.1 Versions

IK 320 A	VMEbus counter card with encoder inputs for sinusoidal
286 939 01	current signals (11 μA_{PP}).
IK 320.1	**Special version:** VMEbus counter card with encoder inputs for
286 839 11	sinusoidal current signals (22 μA_{PP}).
IK 320 V	VMEbus counter card with encoder inputs for sinusoidal
286 939 51	voltage signals (1 V_{PP}).

1.2 Accessories



*For 5.1-V power supply and current consumption of the encoder < 65 mA.

309 781-xx	Connecting cable from encoder output to another display or control
315 650-02	Connector for external functions at connection X41

IK 320 A/IK 320.1

309 785-xx 9/9-pin	Cable adapter with coupling for HEIDENHAIN encoders; standard length 0.5 m
310 195-xx 9/9-pin	Cable adapter with connector for HEIDENHAIN encoders with flange socket; standard length 0.5 m IK 320 V

IK 320 V

309 784-xx 15/12-pin	Cable adapter with coupling for HEIDENHAIN encoders; standard length 0.5 m
310 196-xx 15/12-pin	Cable adapter with connector for HEIDENHAIN encoders with flange socket; standard length 0.5 m

2. Important Information

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

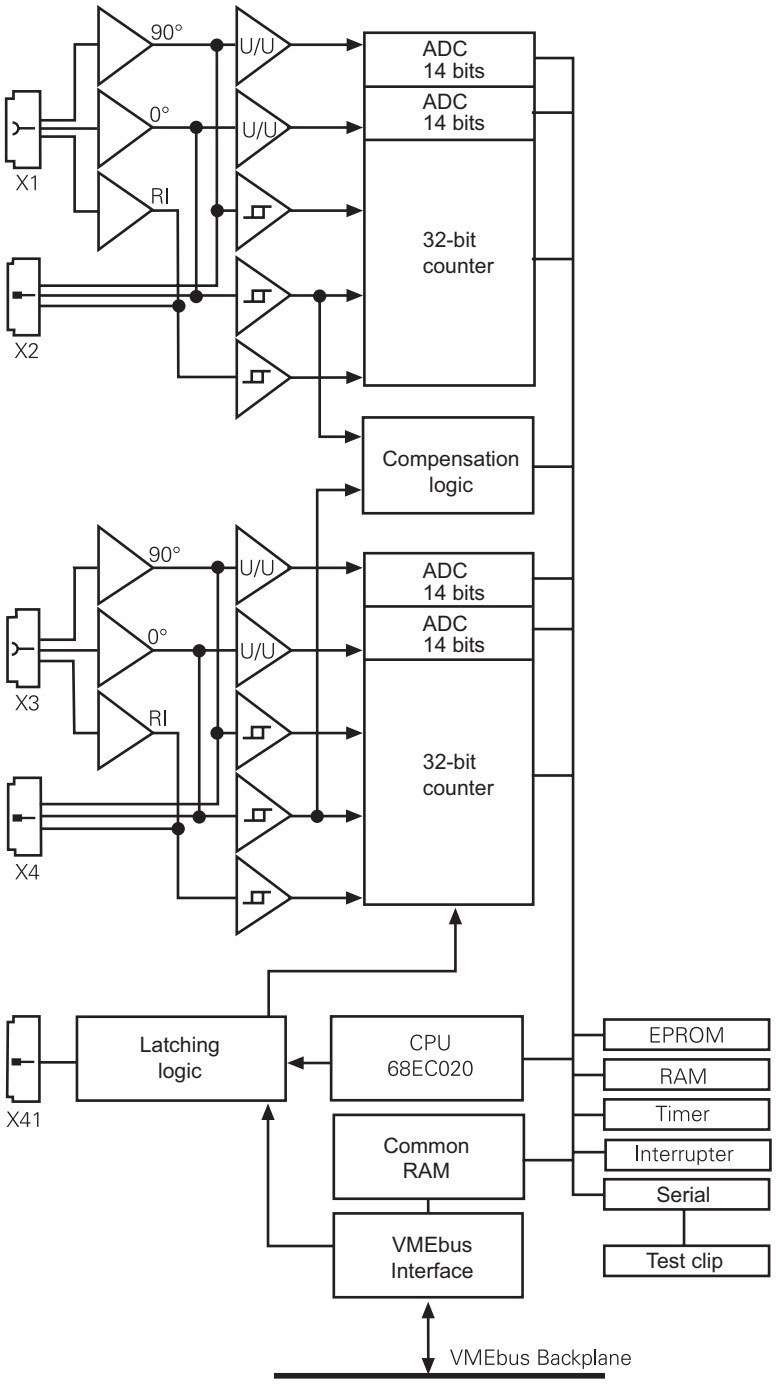
Notes on the terms used:

- Numbers in hexadecimal notation are identified by $, for example: $FF.
- Inverted signals contain a minus sign before the signal name, e.g. –PULSE1.
- The encoder inputs are designated Axis 1 (corresponding to connection X1) and Axis 2 (corresponding to connection X3).
- The term **"to latch"** means that the count value is entered in the data register. This count value must then be **interrogated,** i.e. read by the software and stored in the computer or displayed on the screen.
- Please refer to the technical literature on the VMEbus for the significance of the VMEbus signals and related terms.

3. Technical Description of the IK 320

You may connect two HEIDENHAIN encoders with sinusoidal current signals (IK 320 A) or voltage signals (IK 320 V) to the IK 320 VMEbus counter card. The positions of the two encoders are transmitted to the computer and further processed by the software.
The IK 320 is ideal for applications requiring high resolution of encoder signals.

Block diagram:



The IK 320's interpolation electronics subdivides the signal period of the input signal 4096-fold. The 44-bit wide measured value is formed from the interpolation value (12 bits) and the value of the period counter (32 bits). The IK 320 stores the measured values in the 48-bit data registers; it does not use the bottom 4 bits. The measured values can be latched either through external inputs, via software, or by reference-mark traverse. Then they can be transferred to the computer via the VMEbus.

You can improve the accuracy of the measurement result if you determine the signal shape of the sinusoidal encoder signals by means of a compensation run. The IK 320 analyzes the signal shape and stores compensation values at up to 4096 compensation points for the signal shape of the analog signals.

Communication between the computer (master) and the IK 320 (slave) takes place by means of interrupts and a common RAM for data and parameters (see Chapter 6 "Data, Parameters and Data Transfer").

3.1 Access Time to Measured Values

The time to access measured values is approximately 400 µs per axis.

4. Hardware

4.1 VMEbus Interface

Specification:	ANSI/IEEE STD 1014-1987, IEC 821 and 297		
Size:	Double height board (233.4 mm x 160 mm x 20 mm), 1 slot		
VME connection:	Connector J1 (for pin layout see VMEbus literature)		
Interrupter:	D08(O) ROAK		
Address space A16:	Slave, D08(O) (port)		
	Slave, ADO (synchronous latching)		
	Memory requirement: 16 kilobytes per card		
Address space A24:	Slave, D16, D08 (EO), 16 kilobytes		
Interrupt lines:	7, selectable with jumpers		
Voltages:	+ 5 V :	+ 0.25 V / – 0.125 V	50 mV_{PP} noise
	+ 12 V:	+ 0.60V / – 0.36 V	20 mV_{PP} noise*
	– 12 V:	– 0.60 V / + 0.36 V	20 mV_{PP} noise*
	+ 5 V STDBY:	+ 0.25 V / – 1.70 V	

* not according to VME specification

Noise levels on ± 12 V are not specified according to VMEbus. The noise levels on ± 12 V must be held as low as possible. Noise in the power supply can impair the measuring accuracy.

Current consumption (max):	+ 12 V	160 mA
	– 12 V	160 mA
	+ 5 V	1.2 A
	+ 5 V STDBY	100 µA

The stand-by voltage supplies the compensation value memory.

4.2 Monitor LED

There is a green monitor LED on the board — near the VMEbus connector. The **monitor LED** flashes on power-up at a frequency of approx. 1 Hz and indicates that the card is working. While the encoder signal is being scanned — during the compensation run — the LED does not flash (it remains either on or off). In the event of local faults on the card, it flashes at approximately double the frequency. You can clear this error message only by powering down the system or by a RESET on the VMEbus.

4.3 Encoder Inputs IK 320 A

You can connect HEIDENHAIN linear encoders or angle encoders with sinusoidal current signals I_1 and I_2 to the IK 320 A.

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 usable component
Signal threshold for error message I_1, I_2	≤ 2.5 μA_{PP}
Maximum input frequency	50 kHz
Cable length	Max. 30 m: for 5.1-V supply voltage and encoder current consumption < 65 mA

Connection X1, X3 for encoders
D-sub connection with female contact (9-pin)

Pin 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

4.4 Encoder Inputs IK 320.1 (Special Version)

Signal amplitudes: I_1, I_2 (0°, 90°) I_0 (reference mark)	14 μA_{PP} to 30 μA_{PP} 10 µA to 30 µA usable component
Signal threshold for error message I_1, I_2	≤ 6 μA_{PP}
Maximum input frequency	50 kHz
Cable length	Max. 30 m

4.5 Encoder Inputs IK 320 V

You can connect HEIDENHAIN linear encoders or angle encoders with sinusoidal voltage signals A and B to the IK 320 V.

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 usable component
Signal threshold for error message A, B	≤ 0.22 V_{PP}
Maximum input frequency	50 kHz
Cable length	Max. 250 m: for 5.1-V supply voltage and encoder current consumption < 65 mA

Connection X1, X3 for encoders
D-sub connection with female contact (15-pin)

Pin No.	Assignment
1	A +
2	0 V (U_N)
3	B +
4	+ 5 V (U_P)
5	Do not assign
6	Do not assign
7	R –
8	Do not assign
9	A –
10	0 V (sensor line)
11	B –
12	+ 5 V (sensor line)
13	Do not assign
14	R +
15	Do not assign
Housing	External shield

4.6 Encoder Outputs

The IK 320 additionally transmits the encoder signals of inputs 1 (connection X1) and 2 (connection X3) to two 9-pin D-sub connectors in the form of **sinusoidal current signals** (11 μA_{PP}). Connect these outputs only to subsequent electronics with input amplifiers supplied with ± 12 V; other input amplifiers will be overloaded (U_0 on the IK 320 is 0 V). Adapter cables (Id. Nr. 309 781-01) for connecting HEIDENHAIN position display units are available (see "1.2 Accessories"). **Do not connect any EXE interpolation electronics!**

Encoder outputs X2, X4
D-sub connection with male contact (9-pin)

Pin No.	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

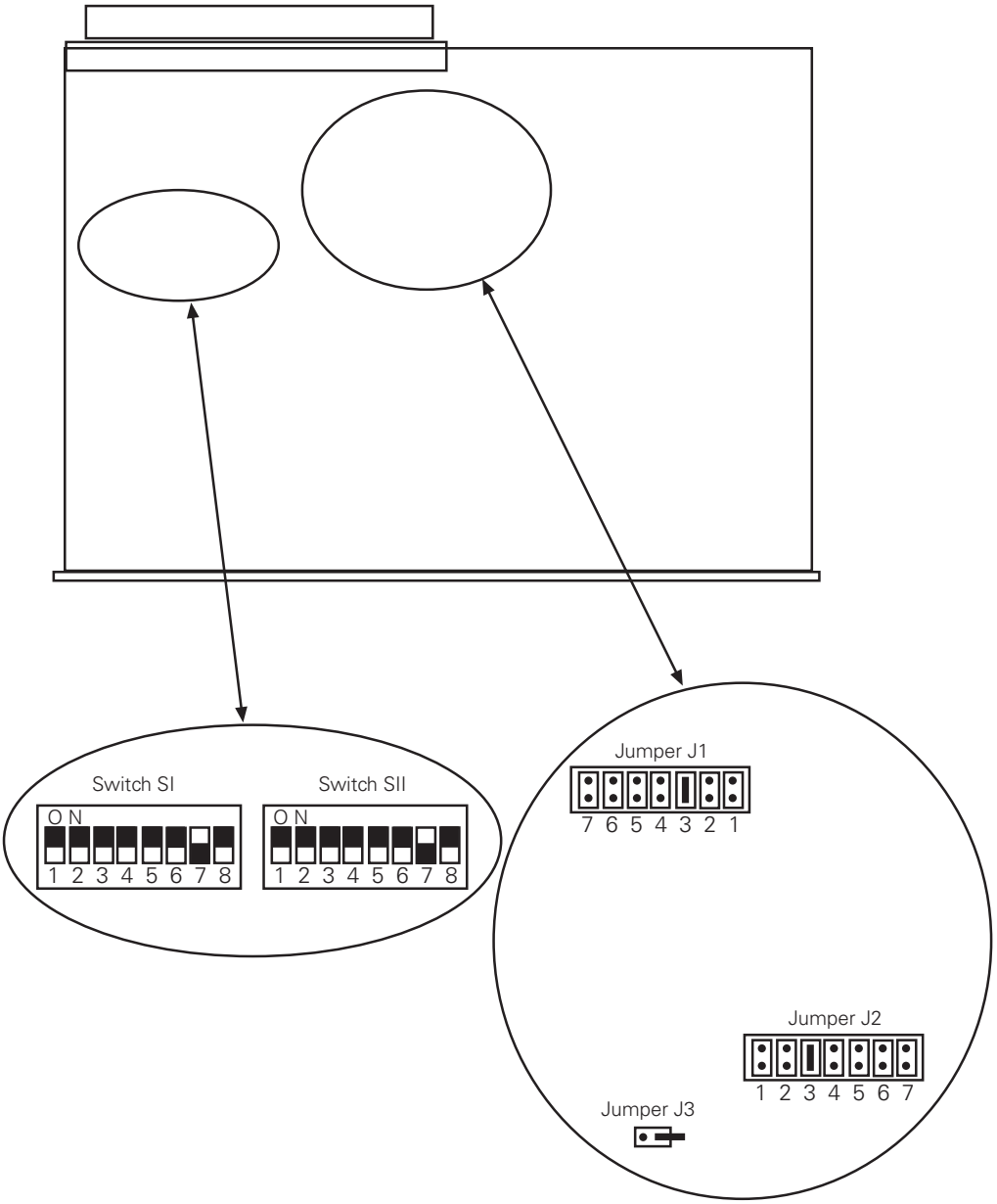
4.7 External Functions

A 9-pin D-sub connector is provided for external latching of measured values and for synchronous latching of multiple cards. The required connector (Id. Nr. 315 650-02) can be ordered from HEIDENHAIN. For a more detailed description of this connection, refer to "8.1 Latching the Position Value."

5. Addressing

5.1 Switches and Jumpers

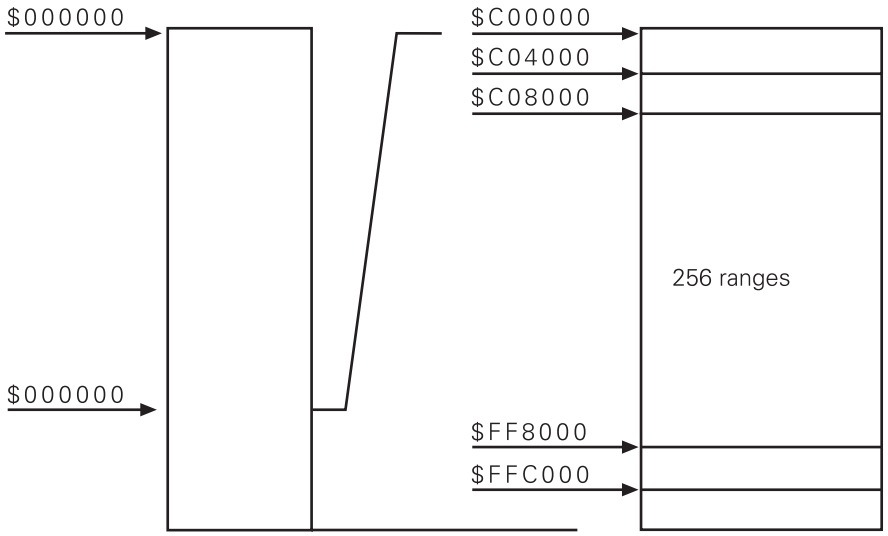
The IK 320 has two DIP switches and three jumpers. DIP switch SI determines the address of VME address space A16 and switch SII the address of VME address space A24.
Jumpers J1 and J2 determine the interrupt request and interrupt acknowledge line. Jumper J3 determines the behavior of the IK 320 during an "ADDRESS ONLY cycle" (ADO cycle).



5.2 VME Address Space A24 (Address Modifier AM: $39)

Data is exchanged between the master and IK 320 through a common 16-KB RAM (in VME address space A24) on the IK 320.
By means of the 8-pin DIP switch SII on the card, you set the IK address in the upper quarter of address space A24 (range $C00000 - $FFFFFF).
The basic address of the card is calculated as follows:
Basic address (BA) = $C00000 + (DIP switch SII*$4000)

Address space 24



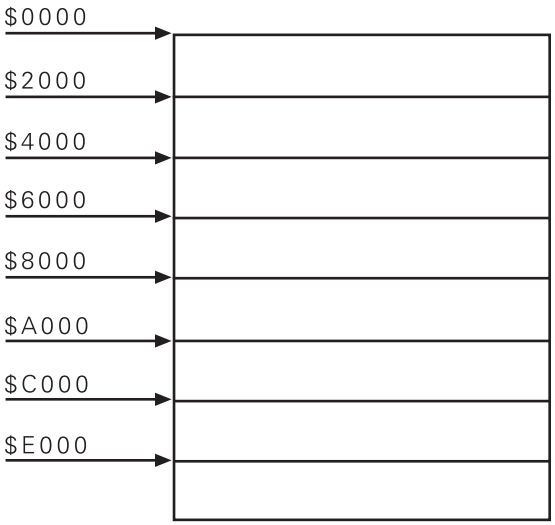
5.3 VME Address Space A16 (Address Modifier AM: $29)

The IK 320 provides two functions in address space A16:

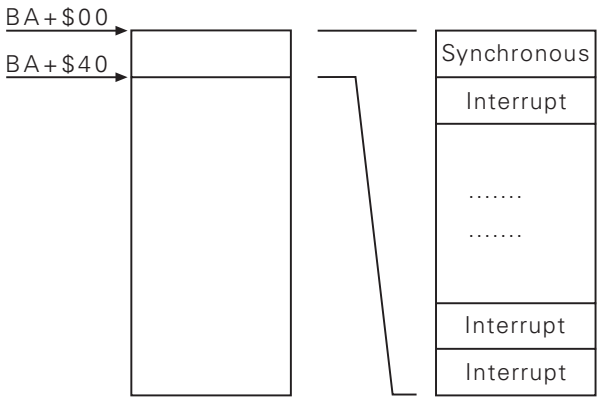
- Local interrupt on the IK 320.
- Latch signal (synchronous latching) and local interrupt. A latch signal can be generated on several cards simultaneously.

Address space A16 is divided into eight ranges (corresponding to groups) by S8, S7 and S6 (DIP switch SI).

Address space A16



The master can generate a **common** latch signal for all cards with the same group setting (DIP switch SI: S8, S7, S6) by carrying out an ADDRESS ONLY cycle at address $00 within this range (the ADDRESS ONLY cycle does not return –DTACK). If an ADO cycle is not possible, see "8.1.5 VMEbus Direct Latching within One Group."
Within one group, the cards are differentiated from the DIP switch S1 by the switches S5 to S1. The addresses for interrupt generation are in the lowest 32 words within the 8-KB range. Writing to a set interrupt address generates a local interrupt.



Do not set address 0 with S1 to S5 of DIP switch SI.

The set addresses are calculated as follows:

Group address = (DIP switch SI: S8, S7, S6) * $2000
Interrupt address = Group address + (DIP switch SI: S5-S1)* 2

5.4 Communication by Interrupt

5.4.1 From the IK 320 to the Master

The IK 320 generates an interrupt on the VMEbus for messages to the master. The master must have cleared the IK interrupt status register so that the IK 320 can send an interrupt to the master. Only then can the IK 320 place the message in the IK interrupt status register (16-bit word). The interrupt request line is selected with jumper J1 (possible lines: –IRQ1 to–IRQ7). The IK 320 stores an interrupt until the master performs an interrupt acknowledge cycle. If a valid IACK cycle is recognized, the IK 320 outputs the vector number (= switch setting of DIP switch SI) as a code for the card and clears the interrupt. The master can read the IK interrupt status register of the IK 320 in the interrupt service. Finally, the master must clear the IK interrupt status register to allow the IK 320 to output new interrupts.

Jumper 1: Interrupt request line

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

* factory-installed

Note: No jumper = Interrupt inhibited.

Jumper 2: Interrupt acknowledge line

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

* factory-installed

Note: The interrupt acknowledge line must correspond to the interrupt request line.

5.4.2 From the Master to the IK 320

The master writes the desired function number in parameter P81. Then the master generates an interrupt (ADO cycle) by writing the interrupt address to address space A16 on the IK 320. The IK 320 detects the interrupt, branches according to the entered function number and executes the function.

Jumper 3:
Defines whether the card generates a DATCK pulse during an ADO (see also "8.1.5 VMEbus Direct Latching within One Group").
Jumper open: No DATCK
Jumper closed: DATCK is generated

5.5 Switch Settings (Example)

The entire system contains a total of five IK 320s divided into three groups:
Group 1: 2 * IK 320
Group 2: 2 * IK 320
Group 3: 1 * IK 320

DIP switch SI:

Group	No.	Switch S8 – S1	Address space A16	Interrupt	Common latching
Group 1	1	1010.0001	$A000-$BFFF	$A002	$A000
	2	1010.0010	$A000-$BFFF	$A004	$A000
Group 2	3	1100.0001	$C000-$DFFF	$C002	$C000
	4	1100.0010	$C000-$DFFF	$C004	$C000
Group 3	5	1110.0001	$E000-$FFFF	$E002	$E000

DIP switch SII:

Group	No.	Switch S8 – S1	Address space A24
Group 1	1	1000.0000	$E00000-$E03FFF
	2	1000.0001	$E04000-$E07FFF
Group 2	3	1100.0001	$F04000-$F07FFF
	4	1100.0010	$F08000-$F0BFFF
Group 3	5	1110.0001	$F84000-$F87FFF

Note:
Table entry "1" means the switch is set to "OFF." For address formation, the OFF setting means that the address bit has the value 1 (logic level = 5V).

6. Data, Parameters, and Data Transfer

Data is exchanged between the master and IK 320 through a common RAM. Interrupts and markers are used to coordinate the transfer. The common RAM is subdivided into the two functional groups **Data** and **Parameters.**

6.1 Data

Access to the data area is coordinated with transfer markers: when the corresponding transfer marker is **$00**, the IK 320 can write; if it is **$01**, the master can read. The transfer markers are cleared in POST (**P**ower **O**n **S**elf **T**est). If the IK 320 encounters a set marker while writing, an error message is sent to the master and the write procedure is delayed until the master deletes the marker.

BA+$00: Position value of axis 1
Size: 6 bytes, Motorola format (high byte first)
A 48-bit number is transmitted. The upper 32 bits contain the count value and the lower 16 bits the interpolation value. Since the interpolation value requires only 12 bits, the bottom 4 bits are not used.

BA+$06: Status of axis 1
Size: 1 byte

Bit	Contents = 0	Contents = 1
0	No signal compensation	Signal compensation
1	Reserved	
2	Stopped	Started
3	OK	Signal amplitude too low
4	OK	Frequency exceeded
5	–	Wait for REF
6	Reserved	
7	–	Compensation is being calculated

The status is updated with each measured value interrogation.

Meanings of the individual bits:

Bit 0: 0: The signal was not compensated during the most recent measured value interrogation, since:
- The position lies outside of the compensated range.
- P06 of this axis is not switched on.
- There are no valid compensation values in the memory.
- The axis has not yet traversed the reference mark (see Chapter 7.3).

1: The signal was compensated during the most recent measured value interrogation.

Bit 2: 0: The axis has stopped; it has not yet traversed the reference mark(s).
1: The axis is started; the bit is set after the referencing function (function number 08) has been called and the reference marks have been traversed

Bit 3: 0: The signal amplitude was OK during the most recent measured value interrogation.
1: The signal amplitude was too small during the last measured value interrogation.

Bit 4: 0: The frequency of the encoder signal did not exceed permissible values.
1: The frequency of the encoder signal exceeded permissible values.

Bit 5: 0: The axis is not in the "Waiting for reference mark" status.
1: The axis is in the "Waiting for reference mark" status; the bit is set after function call 08 and reset when the reference mark is traversed.

Bit 7: The bit is effective only during the compensation run.
0: The compensation run calculation has not yet started, or it is finished.
1: The compensation run is being calculated (see Chapter 7.4).

BA+$07: Transfer marker of axis 1
Size: 1 byte
Transfer marker = $00: IK 320 can write
Transfer marker = $01: Master can read

BA+$08: Position value of axis 2
Size: 6 bytes, Motorola format (high byte first).
A 48-bit number is transmitted. The upper 32 bits contain the count value and the lower 16 bits the interpolation value. Since the interpolation value requires only 12 bits, the bottom 4 bits are not used.

BA+$0E: Status of axis 2
Size: 1 byte

Bit	Contents = 0	Contents = 1
0	No signal compensation	Signal compensation
1	Reserved	
2	Stopped	Started
3	OK	Signal amplitude too low
4	OK	Frequency exceeded
5	–	Wait for REF
6	Reserved	
7	–	Compensation is being calculated

The status is updated with each measured value interrogation.

Meanings of the individual bits: see description **BA+$06: Status of axis 1**

BA+$0F: Transfer marker of axis 2
Size: 1 byte
Transfer marker = $00: IK 320 can write
Transfer marker = $01: Master can read

BA+$10: Common position value of combined axes 1 and 2
Size: 6 bytes, Motorola format (high byte first)
A 48-bit number is transmitted. The upper 32 bits contain the count value and the lower 16 bits the interpolation value. Since the interpolation value requires only 12 bits, the bottom 4 bits are not used.

BA+$16: Status for common position value of combined axes 1 and 2
Size: 1 byte
Reserved, not used

BA+$17: Transfer marker for common position value of combined axes 1 and 2
Size: 1 byte

BA+$18: IK interrupt status
Size: word
When the IK 320 sends the master an interrupt, the cause of the interrupt is shown in the interrupt status word. The master must read and then delete the status word in the interrupt service routine before the IK 320 can output a new interrupt.

$0000	Non-initialized interrupt
$01xx	Position value X1 was written: Lower byte request source: xx = $01: –IK1, external latching pulse $03: Synchronous latch command via VMEbus $04: –F1, external function input $05: –F2, external function input $06: Interrupt master (latching through software, function no. 1)
$01xx	Position value X2 was written: Lower byte request source: xx = $02: –IK2, external latching pulse $03: Synchronous latch command via VMEbus $04: –F1, external function input $05: –F2, external function input $06: Interrupt master (latching through software, function no. 2)
$03xx	Combined position value from X1 and X2 was written: Lower byte = request source xx = $01: –IK1, external latching pulse $03: Synchronous latch command via VMEbus $04: –F1, external function input $05: –F2, external function input $06: Interrupt master (latching through software, function no. 3)
$04xx	Error during latching: no latch signal: Lower byte = axis xx = $01: Axis 1 $02: Axis 2
$05xx	Error during latching: double latch: Lower byte = axis xx = $01: Axis 1 $02: Axis 2
$06xx	Position value cannot be written, since transfer marker set Lower byte = axis xx = $01: Axis 1 $02: Axis 2 $03: Combined axes 1 and 2
$07xx	POST (**P**ower **O**n **S**elf **T**est) concluded, Lower byte: error status xx = $00: no error Bit 0: CRC error compensation values 1 Bit 1: CRC error compensation values 2 Bit 2: Parameter error (see setting parameters) Bit 3: CRC error EPROM Bit 4: Checksum error EPROM 1 Bit 5: Checksum error EPROM 2 Bit 6: Hardware error
$08xx	Reference mark was traversed in axis 1 xx = $00: no error $01: Distance-coded reference marks: incorrect spacing

$02: Error in counter module:
Either traverse another reference mark and then call
the referencing run function again, or
reset the IK 320 (RESET).
This error occurs only if you interrupt or restart
the referencing run with distance-coded reference marks
after traversing the first reference mark.
$03: Abort by user

$09xx Reference mark was traversed in axis 2
xx =
$00: No error
$01: Distance-coded reference marks: incorrect spacing
$02: Error in counter module:
Either traverse another reference mark and then call
the referencing run function again, or
reset the IK 320 (RESET).
This error occurs only if you interrupt or restart
the referencing run with distance-coded reference marks
after traversing the first reference mark.
$03: Abort by user

$0Axx Parameters were updated
xx =
$00: Parameters were transferred unchanged
$01: Parameters contained errors; faulty parameters are overwritten with default values, which are also written in the common RAM. The number of the incorrect parameter can be found in the VME RAM in address BA+0xEA (see page 22).
$02: Error with parameter for rotary axis, P05 (signal periods) = 0 or incorrect value for P04 (REF spacing)

$0Bxx Compensation run for axis 1 completed. Lower byte = message
xx =
$00: OK
$01: Axis has no absolute reference (no REF)
$02: Speed error
$03: Wrong position (only with linear axis)
$04: Wrong direction
$05: Error within one scanning period, drive may be irregular
$06: Calculation error during compensation run
$07: Violation of internal memory area
$08: Wrong number of measuring interrupts, drive may be irregular
$03: Compensation run canceled by user
$10: Compensation run completed, axis drive can be stopped

$0Cxx Compensation run for axis 2 completed. Lower byte = message
xx =
$00: OK
$01: Axis has no absolute reference (no REF)
$02: Speed error
$03: Wrong position (only with linear axis)
$04: Wrong direction
$05: Error within one scanning period, drive may be irregular
$06: Calculation error during compensation run
$07: Violation of internal memory area
$08: Wrong number of measuring interrupts, drive may be irregular
$03: Compensation run canceled by user
$10: Compensation run completed, axis drive can be stopped

$0Exx	Error in background test for reference marks: wrong spacing xx = $01: Axis 1 $02: Axis 2 At each distance-coded reference mark the proper spacing is checked in the background.
$0Fxx	CRC and checksum error. Lower byte: error code xx = $01: EPROM $02: Compensation values 1 $03: Compensation values 2
$10xx	Preset set via VMEbus. Lower byte: Axis xx = $01: Axis 1 $02: Axis 2 $03: Combined axes 1 and 2
$11xx	Preset set via external function xx = Lower byte, **lower** nibble: axis $.1: Axis 1 $.2: Axis 2 $.3: Combined axes 1 and 2 Lower byte, **higher** nibble: source $0.: Function 1 $1.: Function 2
$19xx	Axis 1, start without REF xx = $01: Axis is not started (hardware may be defective) $02: 2. IK 320 waits for the 2nd reference mark being traversed[1)]
$1Axx	Axis 2, start without REF xx = $01: Axis is not started (hardware may be defective) $02: IK 320 waits for the 2nd reference mark being traversed[1)]
$1Bxx	Axes 1 + 2, start without REF xx = Bit 0 = 1: Axis 1 is not started (hardware may be defective) Bit 1 = 1: Axis 1, IK 320 waits for the 2nd reference mark being traversed[1)] Bit 4 = 1: Axis 2 is not started (hardware may be defective) Bit 5 = 1: Axis 2, IK 320 waits for the 2nd reference mark being traversed[1)] Bits 2, 3, 6, 7 have no meaning.

[1)] This IK 320 status may occur when you interrupt or cancel the referencing run with distance-coded reference marks after traversing the first reference mark. The error is corrected by traversing another reference mark or by a RESET on the IK 320.

$20xx	Compensation values for axis 1 read xx = $00: 0K $01: No valid compensation values in the memory $02: Incorrect compensation point number
$21xx	Compensation values for axis 1 written xx = $00: 0K $01: BCC error in transmission $02: Incorrect compensation point number $03: Incorrect axis
$22xx	Compensation values for axis 2 read xx = $00: 0K $01: No valid compensation values in the memory $02: Incorrect compensation point number
$23xx	Compensation value for axis 2 written xx = $00: 0K $01: BCC error in transmission $02: Incorrect compensation point number $03: Incorrect axis
$FDxx	Internal stack error xx = $01: Stack main program incorrect $02: Memory area for stack is exceeded
$FExx	Hardware error (non-used CPU exception has occurred)
$FFxx	Command not recognized. Lower byte: command (call of a non-existent function number)

Note:
If the card reports with the interrupt status $FDxx or $Fexx, this means that there is an internal error in the card. The IK 320 performs a local software RESET. Afterwards the card must be reinitialized (parameters, POST, REF run).

BA+$1A: CRC sum EPROM NOML
Size: word
Set in POST.

BA+$1C: CRC sum EPROM ACTL
Size: word
Set in POST.

BA+$1E: Checksum EPROM 1 NOML
Size: long word
Set in POST.

BA+$22: Checksum EPROM 1 ACTL
Size: long word
Set in POST.

BA+$26: Checksum EPROM 2 NOML
Size: long word
Set in POST.

BA+$2A: Checksum EPROM 2 ACTL
Size: long word
Set in POST.

BA+$2E: Hardware version
Size: word
Set in POST.

BA+$30: Software version
Size: 12 bytes, string
Set in POST.

BA+$3C: System status 1
Size: byte
Reserved, required only internally.

BA+$3D: System status 2
Size: byte
Reserved, required only internally.

BA+$3E: Current preset for axis 1
Size: 3 words

BA+$44: Current preset for axis 2
Size: 3 words

BA+$4A: Current preset for combined axes 1 and 2
Size: 3 words

Note:
The current preset is the value that was calculated during PRESET setting (externally or through VMEbus) and taken into account during the position value determination (see "8.2 Calculating the Position Value").

BA+$50 – BA+$6F: Data area for reading compensation value
Size: 16 words
Word 0: Number of compensation point
Words 1 to 8: Compensation values
Word 9: BCC sum via words 0-8
Words 10 to 15: *Reserved*

BA+$70 – BA+$8F: Data area for writing
Size: 16 words
Word 0: Number of compensation point
Words 1 to 8: Compensation values
Word 9: BCC sum via words 0-8
Words 10 to 15: *Reserved*

BA+$90: CRC sum compensation data for axis 1
Size: word

BA+$92: CRC sum compensation data for axis 2
Size: word

BA+$94 – BA+$E9: free memory area

BA+$EA: Number of incorrect parameters
Size: word
If POST or the Transfer parameter function reports an incorrect value, the number of the incorrect parameter can be taken from this memory area. In addition, the following numbers indicate the following errors:
100: Spacing between the reference marks (P04.x) is greater than the signal periods (P05.x)
101: Rotary axis (P02.x > 1) and signal periods = 0 (P05.x)
102: Distance-coded reference marks (P04.x is not equal to 0) and P19 is not equal to 0

BA+$EC (32 bits): Signal period counter X1
Size: long word

BA+$F0: Trigger status X1
Size: word
The trigger status indicates the level of the 90° trigger signal at the time of latching.
0x0000: Level is LOW
0x0001: Level is HIGH

BA+$F2 (32 bits): Signal period counter X2
Size: long word

BA+$F6: Trigger status X2
Size: word
For description, see above.

BA+$F8: Encoder input for axis 1: Analog value of 0 degree signal
Size: word

BA+$FA: Encoder input for axis 1: Analog value of 90 degree signal
Size: word

BA+$FC: Encoder input for axis 2: Analog value of 0 degree signal
Size: word

BA+$FE: Encoder input for axis 2: Analog value of 90 degree signal
Size: word

Note:
In order to determine the peak-to-peak value of the encoder current or voltage, the maximum and minimum values must be captured. The peak-to-peak values can then be calculated using the input amplification factors of the IK 320.
Amplification of the encoder input signals:
IK 320 A: 540 mV/µA
IK 320.1 A: 301 mV/µA
IK 320 V: Voltage amplification factor = 5.84
The analog values are rewritten only when a new measured value is interrogated.
Format: 14 bits, signed, left-aligned, bit 0 and bit 1 always at 0
Significance: One increment corresponds to 0.61 mV
Example: $4000 corresponds to 4096 x 0.61 mV = 2.499 volts

6.2 Parameters

Access to the parameter area is coordinated by way of interrupt: generally the master can always write to the parameter range. However, the IK 320 only reads the range when it is requested to do so. From this point until acknowledgment, the master is forbidden to write to the parameter range. Parameter P81 (Master Interrupt Functions) has a special status. This parameter is **always** read by the IK 320 after a master interrupt.

P01.1: Counting direction for axis 1
P01.2: Counting direction for axis 2
P01.3: Parameter is without meaning
Address: BA+$102, BA+$103, BA+$104
Valid values: 0, 1
Size: byte
0: normal, 1: inverse
In the setting **inverse** the position value to be output will be inverted.

P02.1: Axis definition for axis 1
P02.2: Axis definition for axis 2
P02.3: Axis definition for combined axes 1 and 2
Address: BA+$106, BA+$107, BA+$108
Valid values: 1, 2, 3, 4
Size: byte
1: linear, 2: angle 0° ... 360°, 3: angle ± ∞, 4: angle ± 180°
In the **Angle** setting, the position calculation is reduced to the position within one revolution, as is the assignment of compensation values.

P03: Number of bits for subdivision
Address: BA+$10A
Valid values: 0 – 16
Size: word
Internally, 16-bit subdivision is used. Before output to VMEbus the value is rounded off according to the indicated number of bits; the excess bits are deleted.

P04.1: Reference mark spacing for axis 1
P04.2: Reference mark spacing for axis 2
Address: BA+$10C, BA+$10E
Valid values: 0 and any even values from 64 to 8192
Size: word
Value = Basic spacing for distance-coded reference marks in signal periods;
0 = one reference mark or reference marks that are evenly spaced

P05.1: Signal periods for rotary axis 1
P05.2: Signal periods for rotary axis 2
P05.3: Signal periods for combined rotary axes 1 and 2
Address: BA+$110 (-$113), BA+$114 (-$117), BA+$118 (-$11B)
Valid values: any
Size: long word
Value = encoder signal periods per revolution.

P06.1: Compensation on/off for axis 1
P06.2: Compensation on/off for axis 2
Address: BA+$11C, BA+$11D
Valid values: 0, 1
Size: byte
0: Compensation off
1: Compensation on
P06 is significant only for measured value interrogation. Interrogation without compensation is faster.

P07.1: Compensation value range for start in axis 1
P07.2: Compensation value range for start in axis 2
Address: BA+$11E (-$121), BA+$122 (-$125)
Valid values: any
Size: long word
Absolute position in signal periods (without interpolation) for the compensated range.

P08.1: Number of compensation values for axis 1
P08.2: Number of compensation values for axis 2
Address: BA+$126 (-$127), BA+$128 (-$129)
Valid values: 1 – 4096
Size: word
Value = Number of compensation values

P09.1: Interval of compensation points for axis 1
P09.2: Interval of compensation points for axis 2
Address: BA+$12A (-$12B) , BA+$12C (-$12D)
Valid values: all except zero
Size: word
Value = Number of signal periods between compensation points

P10: Inhibit position value call for specific axes
Only output to the VMEbus is disabled.
Address: BA+$12E
Valid values: 0 – 3; 16 – 19
Size: byte
Bit 0: Disable transfer of value for axis 1 on the VMEbus
Bit 1: Disable transfer of value for axis 2 on the VMEbus
Bit 2: Reserved
Bit 3: Reserved
Bit 4: Disable external direct latching (with –IK1 / –IK2)
If, for example, the position values are interrogated through function number 03 (interrogation of combined axes), the card replies for P10 = 00 with the position values of each of the two axes and the position value of the combined axes, using an interrupt for each value: the master has to process 3 interrupts. If, however, only the combined position value is desired, the transmission of the individual axes can be disabled with P10 = 3 so that only one reply interrupt need be processed.
With function 01 (measured value interrogation of axis 1) and P10 = 01 there is no reply. P10 is effective with all other latching sources as well.

P19.1: Reserved for customer-specific use
P19.2: Reserved for customer-specific use
Address: BA+$130 (-$133), BA+$134 (-$137)
Size: long word
Value must be set to 0!

P21: Axis combination
Address: BA+$138
Valid values: 0 – 3
Size: byte

0:	no axis combination	common value is not output.
1:	X1 + X2	common value is output.
2:	X1 – X2	common value is output.
3:	(X1 + X2) / 2	common value is output

The result of the combination is stored in the "common position value" data range (see "6.1 Data").

Note:
If the counting direction of an individual axis is inverted with parameter 01.x, the position of that axis is calculated into the sum as a negative value.

P30.1 Direction (axis 1) and frequency (axes 1 and 2) of compensation run
Address: BA+$13A
Valid values: 1 – 7
Size: byte
Bit 0, bit 1: Frequency (axes 1 and 2)
1: 1.35 kHz – 65 Hz
2: 650 Hz – 35 Hz
3: 80 Hz – 5 Hz
Bit 2: Direction (axis 1)
0: positive
1: negative
Parameter P30.1 sets the frequency range for the compensation run for **both** axes. The 5-V power supply must be interrupted to set another frequency range (a RESET alone is not enough!). Then the correct value must be in P30.1 before the first POST to ensure that the corresponding frequency range is set. If only the direction bit is changed (bit 2), a parameter is transferred as usual (P81, function number $0A).
It is best to select a frequency range as close as possible to the middle of one of the 3 ranges. If the frequency range is exited during a compensation run, the IK 320 aborts the compensation run and outputs an error message.
P30.2 Direction of compensation run for axis 2
Address: BA+13B
Valid values: 0, 4
Size: byte
Bit 2: Direction for axis 2
0: positive
1: negative

P70.1: Value for external setting of axis 1
P70.2: Value for external setting of axis 2
P70.3: Value for external setting of combined axes 1 and 2
Address: BA+$13C (-$141), BA+$142 (-$147), BA+$148 (-$14D)
Valid values: any
Size: 3 words
The set value is used as a display value with external setting through the function inputs –F1, –F2; the corresponding function must be preset in P80.1 or P80.2. The card internally calculates a PRESET value that is added to the count in order to arrive at the desired value in the display. This makes it possible, for example, to reset to zero at any point along the scale. The calculated preset value is displayed in the status area (see "6.1 Data).

P71.1: Value for VMEbus setting for axis 1
P71.2: Value for VMEbus setting for axis 2
P71.3: Value for VMEbus setting for combined axes 1 and 2
Address: BA+$14E (-$153), BA+$154 (-$159), BA+$15A (-$15F)
Valid values: any
Size: 3 words
The values have the same function as P70 except that the PRESET setting is started through a VMEbus function call (function numbers 04,05,06).

P72.1: Axis offset for axis 1
P72.2: Axis offset for axis 2
P72.3: Axis offset for combined axes 1 and 2
Address: BA+$160 (-$165), BA+$166 (-$16B), BA+$16C (-$171)
Valid values: any
Size: 3 words
The set value is added unchanged into the actual value. With this parameter a position value shift is possible. The value is taken into account as soon as it is stored in local RAM with parameter input (P81, function number $0A). After RESET, P72 = 0.
P72 has no effect on the compensation value configuration.

P80.1: External function –F1
P80.2: External function –F2
Address: BA+$172, BA+$173
Valid values: 0 – 6
Size: byte

$00:	No function
$01:	Read axis 1: latching, calculation, and provision of the position value
$02:	Read axis 2: latching, calculation, and provision of the position value
$03:	Read combined axes 1 and 2: latching, calculation, and provision of the position values
$04:	Preset axis 1 with value from parameter P70.1
$05:	Preset axis 2 with value from parameter P70.2
$06:	Preset combined axes 1 and 2 with value from P70.3

Active edges at the –Fl or –F2 input of socket X41 generate an interrupt on the IK320. The latter carries out the function set in parameter P80.1 (or P80.2). When the function is completed or in case of an error, a message is returned to the master.

P81: Master interrupt function
Address: BA+$100
Valid values: 01 – $0C, $18, $19 – $1B, $20 – $23
Size: word

$0000:	No function
$0001:	Read axis 1: latching, calculation, and provision of the position value
$0002:	Read axis 2: latching, calculation, and provision of the position value
$0003:	Read combined axes 1 and 2: latching, calculation, and provision of the position values
$0004:	Preset axis 1 with value from parameter P70.1
$0005:	Preset axis 2 with value from parameter P70.2
$0006:	Preset combined axes 1 and 2 with value from P70.3
$0007:	Start POST
$0008:	Cross over reference mark in axis 1
$0009:	Cross over reference mark in axis 2
$000A:	Update parameters
$000B:	Compensation run in axis 1
$000C:	Compensation run in axis 2
$0018:	Move axis 1 and axis 2 over reference marks
$0019:	Axis 1, start without REF
$001A:	Axis 2, start without REF
$001B:	Axes 1+2, start without REF
$0020:	Read compensation values for axis 1
$0021:	Write compensation values for axis 1
$0022:	Read compensation values for axis 2
$0023:	Write compensation values for axis 2

If an interrupt is generated by the master on the IK 320 (writing the interrupt address in address space A16), the IK 320 performs the function set in parameter P81. When the function is completed or if an error occurs, a message is sent to the master.

7. Functions

7.1 Interruptibility of the Functions

The following table contains the interruptible and non-interruptible functions:

Function or function number	Meaning	Interruptibility
Only hard latches –PULSE1, –PULSE2, X41, pins 4 and 5	External direct latching	Mutually interruptible. Otherwise interruptible only by the master interrupt with function number $0A. An interrupt of the same source is disabled until the inactive level is detected at the input pin.*
Only hard latches, –F1, –F2, X41, pins 6 and 7	External indirect latching	Interruptible only through external direct latching or by the master interrupt with function number $0A. An interrupt of the same source is disabled until the inactive level is detected at the input pin.*
$01	Soft latch X1	Interruptible only through hard latches
$02	Soft latch X2	Interruptible only through hard latches
$03	Soft latch X1/X2	Interruptible only through hard latches
$04	Preset X1	Not interruptible
$05	Preset X2	Not interruptible
$06	Preset X1/X2	Not interruptible
$07	POST	Not interruptible
$08	REF X1	Interruptible for hard latches, soft latches and REF X2; abortable if axis is in the status WAIT FOR REF
$09	REF X2	Interruptible for hard latches, soft latches and REF X1; abortable if axis is in the status WAIT FOR REF
$0A	Set parameter	Not interruptible
$0B	Compensation X1	Not interruptible; abortable
$0C	Compensation X2	Not interruptible; abortable
$18	REF X1/X2	After initialization interruptible for hard latches and soft latches; abortable
$19	Axis 1 Start without REF	Not interruptible
$1A	Axis 2 Start without REF	Not interruptible
$1B	Axes 1+2 Start without REF	Not interruptible
$20	Upload X1	Not interruptible
$21	Download X1	Not interruptible
$22	Upload X2	Not interruptible
$23	Download X2	Not interruptible

*Interruptibility with master interrupt and function number $0A is required to make a disable of the external functions via "Set parameter" possible.

Explanation of terms
Hard latches: Latching externally directly (via X41, –PULSE1/2, –CONTACT1/2), latching externally indirectly (via X41, –F1/2), VMEbus directly (latching via SYNCHRONOUS address)
Soft latches: VMEbus indirectly (function numbers $01, $02, $03)
Not interruptible: No new function calls are accepted until the IK 320 has completed a running function and has concluded it with a reply interrupt.
Abortable: Function can be aborted with function number 0 and master interrupt.

7.2 Power On Self Test (POST)

After switch-on, the master must start an initializing procedure on the IK 320. This is done by calling the POST, which takes about four seconds to run.
It is not possible for the IK 320 to access the common RAM before this initializing procedure has been performed; nor will it react to any other function call. The master can read and write to the common RAM as needed and can now, for example, set the parameters for the card.
When the POST is called the parameters are automatically transferred into the common RAM and checked for valid values. If there are incorrect values, the default values are set and an error message is generated. The default values are also written into the common RAM. If an error occurs, the number of the parameter concerned is written into the VME-RAM (BA+$EA).

Important:
After POWER ON, **before** the first POST call, the correct value must be written into parameter P30.1, which sets the frequency range of the compensation run, so that the hardware for the compensation run is initialized correctly.
While the POST is running, the control of the VME-RAM is handed over to the IK 320, and the master must not access the common RAM until the IK 320 has sent the master a reply interrupt.

During the POST the IK 320 performs the following routines for initializing and testing the hardware:

- Checksum and CRC sum test of the EPROM
- CRC sum test of the compensation value memory
- Test of local RAM
- Test of VME-RAM
- Test of BERR time-out
- Test of counter components for axes 1 and 2, counter components are stopped after POST

The IK 320 acknowledges with an interrupt, which the card sends to the master. In the interrupt status word the high byte contains the value $07 (= POST completed) and the low byte contains the error message (e.g. $00 = no error. See "6.1 Data," IK interrupt status).

Calling the POST:
The master erases the IK interrupt status byte in order to give the IK 320 the possibility of replying to the master.
The master then writes the value $07 in **P81 Master interrupt function** and causes a master interrupt on the IK 320 (see "5.4.2 From Master to IK 320").

7.3 Traversing the Reference Marks

With incremental encoders, the correlation between axis position and display value is lost after a power failure. After a power failure, traversing reference marks re-establishes the reference to the axis position. HEIDENHAIN encoders have one or more distance-coded reference marks. In parameter P04 you define which reference marks your encoder has.

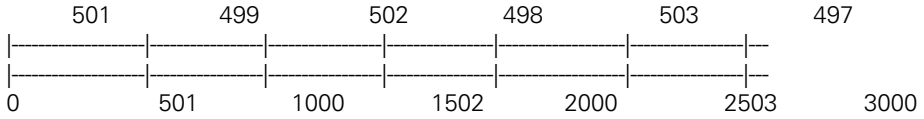
Procedure for one reference mark:
The master generates interrupt function $08 (traversing the reference mark for axis 1), $09 (traversing the reference mark for axis 2) or $18 (traversing both reference marks).
The IK 320 performs the following functions for the selected axis:

- Counter stop: the counter is stopped and the status is set in the register "status for axis 1."
- Counter reset: the counter is set to 0.
- Counter start with reference pulse: The counter starts with the next traversed reference mark. The counter status register is continually updated, i.e. the current states are stored in the data registers "status for axis 1" (BA+$06) or "status for axis 2" (BA+$0E) (see "6.1 Data").

After the reply interrupt the master reads the interrupt status register and recognizes through IK interrupt status $0800 or. $0900 that the reference mark was traversed. Position value interrogations during referencing always reply with the position value 0, i.e. the corresponding axis is not yet started.

Procedure for distance-coded reference marks:
In the case of encoders with distance-coded reference marks, reference marks are located at fixed distances on the entire measurement path. Between two of these reference marks there is a third, whose distance from the other two varies in such a way that each distance is a multiple of the grating period and occurs only once over the entire measurement path. This enables the IK 320 to re-establish the correlation between axis position and display value after a power failure simply by traversing two reference marks.

For a basic distance of 1000 signal periods, the following distribution results for the reference marks:



In parameter P04 you define the basic distance for the distance-coded reference marks in signal periods. The master starts traversing the reference marks with the interrupt function $08 (traversing reference mark in axis 1), $09 (traversing reference mark in axis 2) or $18 (traversing both reference marks).

The IK 320 performs the following functions for the selected axis:
- The counter is stopped and set to 0.
- The counter is started with traversing of the first reference mark.
- With traversing of the second reference mark, the IK 320 stores the position value and thereby determines the distance between the reference marks in increments. From this distance, the IK 320 calculates the absolute position in relation to the first reference mark on the scale (position "0" in the figure).

LIDA scale tapes with distance-coded reference marks, which are mounted on a circular band so that the reference interval is irregular at the butt joint, are also evaluated by the IK 320. In this case it is however necessary that the axis in P02 be defined as a rotary axis in order for the reference-mark evaluation at the butt joint to work.

Remark: Background tests (CRC sum EPROM, CRC sum compensation data, distances between coded REF marks) are switched off as soon as an axis goes into the WAIT FOR REF status.

Aborting traverse of reference marks
You can abort reference mark traverse only with function number 0 and a master interrupt.

Starting the axes without traversing the reference marks
For applications that require a speed signal immediately after switch-on, the axes can also be started without a referencing run.

The following interrupt functions are available for this purpose:
$19 (start axis 1 without REF),
$1A (start axis 2 without REF),
$1B (start axes 1+2 without REF).

7.4 Compensation Run to Compensate for Deviations in the Encoder Signals

The IK 320 can compensate for deviations in the encoder signals which are determined in a compensation run. A card can perform a compensation run for only **one** axis at a time. However, it is possible to start several cards at the same time for a compensation run. When both axes of a card are used, two separate compensation runs must be performed. The compensation run can go in the positive or negative direction (parameter 30.x, bit 2).
Parameters P07.x, P08.x, P09.x and P30.x must be set for each axis. P06.x plays no part in the compensation run.
Parameter P07.x indicates the smallest value of the compensation range: when the compensation run is in the positive direction it is the first value that is compensated; in negative direction it is the last value. P08.x indicates the number of compensation points. P09.x indicates the spacing of compensation points in terms of signal periods.
Parameters P07.x, P08.x and P09.x determine the compensation value range. Parameter P30.1 selects the speed interval for traversing the compensation value range. This interval applies for both axes. If the chosen speed is not maintained, the compensation run will be aborted and an error message results.

Important:
Parameters P07.x,P08.x and P09.x must not be changed after a compensation run. To do so would invalidate the assignment of compensation values to positions while interrogating measured values. Parameters P70.x, P71.x and P72.x, however, do not affect this assignment.

Operating sequence of the compensation run:
The axis must be active (see "7.3 Traversing the Reference Marks"). On **linear axes,** the position is checked before the actual beginning of the compensation run. It must be at least ten signal periods **before** the first value of the compensation run in accordance with the selected scanning direction.

On **rotary axes** there is no position checking: if the distance to the compensation range is not at least ten signal periods, the start of the compensation run is postponed until the next revolution. When the function is called, the axis can be stationary. It must then move in the proper direction and be at the correct speed when it reaches the compensation range.
If the axis moves in the wrong direction, the compensation run is aborted after 100 signal periods and the error status is set.
After the compensation range has been reached, the card begins scanning the encoder signals. This requires its complete processing capability, which is why the LED stops blinking (see "4.2 Monitor LED"). If an error occurs during scanning, the function is aborted and the error status is set. After scanning is completed, the card transmits the "compensation run completed" interrupt and the axis can be stopped. The LED starts blinking again and the card processes the scanning values.

Note:
For 4096 compensation points, the calculation of compensation values takes approximately 3 minutes and 30 seconds. During this time, bit 7 is set in the status byte of the respective axis (see "6.1 Data").

Errors may occur during calculation, for example from an uneven drive. In this case the calculation is aborted and the error status is set. If the compensation run and calculation are completed without error, the reply interrupt "compensation run OK" is transmitted.

Caution:
Acceleration should be kept as low and linear as possible during the compensation run.

Interrupting a compensation run
You can abort the compensation run only with function number 0 and a master interrupt.

7.5 Reading and Writing Compensation Values

A complete block of compensation values always contains P08x + 2 compensation points, i.e. when the values are being read or written the compensation points from 0 to P08x+1 must always be read or written. In the common RAM there are two data ranges reserved: one for reading (BA+$50 to BA+$6F) and one for writing (BA+$70 to BA+$8F). For a compensation point to be transmitted, the compensation point number, 8 compensation coefficients (K1 to K8) and the BCC sum of the compensation point number and the 8 coefficients are necessary. These are exclusively 16-bit data words.
The BCC sum is the result of an EXOR gating of the operands.

One coefficient pair contains the real and imaginary components of the respective vector of the function.
K1/K2 = Coefficients of the fundamental oscillation of the error function
K3/K4 = Coefficients of the 2nd harmonic error function
K5/K6 = Coefficients of the 3rd harmonic error function
K7/K8 = Coefficients of the 4th harmonic error function

Function numbers:
$20: Reading compensation values X1
$21: Writing compensation values X1
$22: Reading compensation values X2
$23: Writing compensation values X2

Status of reply interrupt:

$20xx	Compensation values for axis 1 read xx = $00: 0K $01: No valid compensation values in the memory $02: Incorrect compensation point number
$21xx	Compensation values for axis 1 written xx = $00: 0K $01: BCC error in transmission $02: Incorrect compensation point number $03: Incorrect axis
$22xx	Compensation values for axis 2 read xx = $00: 0K $01: No valid compensation values in the memory $02: Incorrect compensation point number
$23xx	Compensation value for axis 2 written xx = $00: 0K $01: BCC error in transmission $02: Incorrect compensation point number $03: Incorrect axis

7.5.1 Reading Compensation Values

It is only possible to read values when the compensation data stored in the memory is valid. You must write the compensation point number in the appropriate memory region of the common RAM (BA+$50). Then the function number (e.g. $20 for X1) is written in P81 and a master interrupt follows. The card writes the 8 coefficients for this compensation point in the common RAM, calculates the BCC sum of the compensation point numbers and the 8 coefficients and stores this in the common RAM (BA+$62). Then it answers with answer interrupt and status (e.g. $2000 for error-free cycle). Now the user can check the BCC sum, and the compensation point number together with the coefficients can be read and stored. In theory reading the compensation points is possible in any order. It is recommended, however, to keep the order 0 to P08x+1, as this order is necessary for the defining of the compensation value RAM (see 7.5.2).
In the common RAM the CRC sum can be read out via the compensation value RAM of an axis (BA+$90 for X1, BA+$92 for X2, 16-bit values). This CRC sum can also be saved as a control factor in a complete compensation value block of an axis, because the IK 320 must transmit the same CRC sum after the writing of the compensation values. It is also advisable to keep the corresponding parameter block.

7.5.2 Writing Compensation Values

When writing compensation values it is vital to keep the order of the compensation point numbers from 0 to P08x+1. The axis values can only be transmitted in succession. If the order is not kept, a BCC error will occur and if the compensation point is defined for the other axis then the procedure is aborted and must begin again with compensation point 0.
You write the compensation point number, the 8 coefficients and the BCC sum of the compensation point number and the 8 coefficients in the common RAM (BA+$70 to $82). Then the function number (e.g. $21 for X1) is written in P81 and a master interrupt follows. The card checks the BCC sum of the coefficients, writes the 8 coefficients of this compensation point in the temporary memory and answers with the corresponding interrupt and status (e.g. $2100 for an error-free run). If all the necessary compensation points have been transmitted, the card writes the values from the temporary memory into the actual compensation value memory range and calculates again the CRC sum for this memory range. This CRC sum is also stored in the common RAM (BA+$90 for X1, BA+$92 for X2, 16-bit values). This sum must be the same as the CRC sum that was transmitted when the compensation values were being read.

8. Interrogating a Position Value

Interrogation of the position value is started by a latching procedure. The position value is calculated, stored in the "data" range and an interrupt is sent to the master. Additionally, the latching source of the measurement value is transferred in the interrupt status register. Depending on the parameter setting and the type of interrogation (external or via VMEbus), the card generates from zero to three reply interrupts (see P10, P21), which must all be processed before the card is again ready for new function calls.

8.1 Latching the Position Value

The position value can be latched directly or indirectly.

Note:

In the case of **direct** latching, a local interrupt is generated on the IK 320; in the case of **indirect** latching, the latching signal is generated by software via the local CPU of the IK 320. Consequently, different times result for latching the individual values.

The following latching methods are available:

- External direct: Signals –PULSE1, –PULSE2, –CONTACT1, –CONTACT2 in socket X41 (external functions).
- External indirect: Signals –F1, –F2 in socket X41 generate a local interrupt; latching by software.
- VMEbus direct: Setting the synchronous latch address.
- VMEbus indirect: Generation of a local interrupt; latching by software.
- By traversing the reference marks: Traversing a reference mark starts a latching process.
- With timer: Latching with local time reference (needed only for compensation run).

Every local latch pulse is output at X41 pin 1 (–LOUT), regardless of the latching method or the axis involved.

The current position values can only be interrogated if the reference marks have been traversed after POST (see "7.3 Traversing Reference Marks").

8.1.1 Pin Layout, Connection for External Functions (X41)

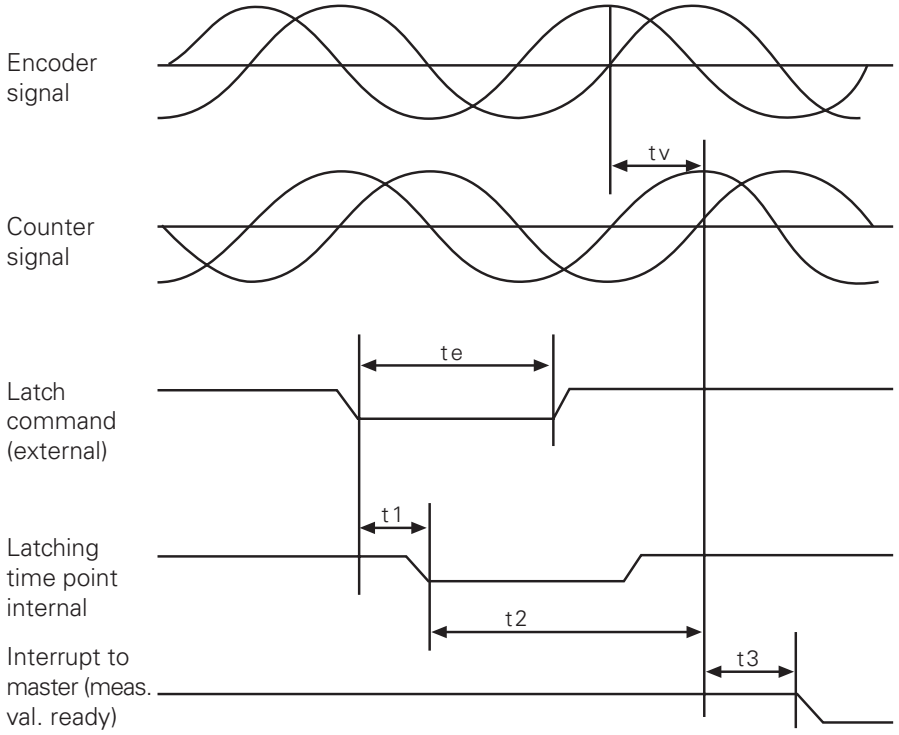
Meaning of signal	Designation	Pin number
Latching via pulse for axis 1	–PULSE1	4
Latching via pulse for axis 2	–PULSE2	5
Latching via contact for axis 1	–CONTACT1	2
Latching via contact for axis 2	–CONTACT2	3
Latching via interrupt for axis 1	–F1	6
Latching via interrupt for axis 2	–F2	7
0 V	0 V	8
0 V	0 V	9
Output for latching pulse	–LOUT	1

Signal levels:

Designation	Min.	Max.	Unit
U_{eH}	3	15	V
U_{eL}	– 0.5	1	V
I_{eL}		6	mA

The inputs are active LOW and are kept at HIGH level with internal pull-up resistors. Triggering is possible with TTL standard, LS, ALS or CMOS components.

Time diagram



Signal	Design.	Min.	Max.	Unit
Signal delay from encoder input to A/D converter or counter	t_v		14	µs
Width of latch command	t_e			*
Latch signal delay	t_1			*
Instant of stored measuring signal relative to instant of storage	t_2		100	ns
Measured value ready (per axis, compensated)	t3		400	µs
Measured value ready (per axis, uncompensated)	t_3		180	µs

* different for the various sources

8.1.2 External, Direct Latching

With make contact of the signals –PULSE1, –PULSE2 or –CONTACT1, –CONTACT2 against 0 V, a latch signal is generated via hardware. The inputs –PULSE and –CONTACT differ in their time response:

Pulse: $t_e \geq 1.2$ µs, $t_1 \leq 0.8$ µs **Contact:** $t_e \geq 7.0$ ms, $t_1 \leq 4.5$ ms

Caution:
With an axis combination (P21) the latch pulse at input –PULSE1/CONTACT1 is applicable for both axes. A pulse at –PULSE2/CONTACT2 is not evaluated.

8.1.3 External, Indirect Latching through Interrupt

With make contact of the signals –F1 and –F2 against 0 V, a local interrupt is generated on the card. The CPU performs the functions set in parameter P80.1 or P80.2; the parameter values $01, $02, $03 generate a latch signal in the corresponding channels.

Time response F1, F2:	te	≥ 25 µs
	t1	≤ 10 µs

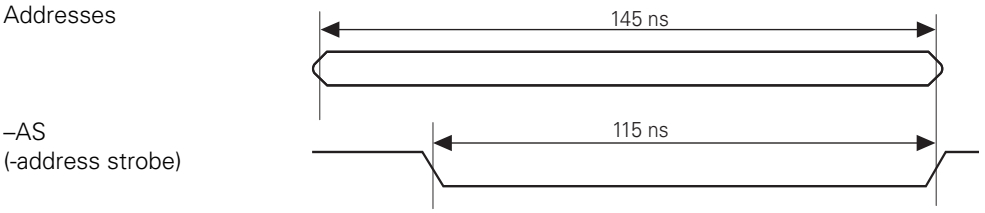
8.1.4 VMEbus Direct Latching

Latching can take place via the VMEbus by an address access in address space A16.

Time diagram

te = ADO access VME (ADDRESS ONLY access)
t1 ≤ 500ns from DS0/DS1

Timing:



8.1.5 VMEbus Direct Latching within One Group

A latching process is initiated simultaneously in all axes of this group (several cards) by an ADO access to the group address of address space A16. Individual axes can be disabled with parameter P10. Parameter P21 will be taken into account (axis combination). The higher-level processor must carry out an ADO cycle, i.e. none of the cards returns a –DTACK signal.

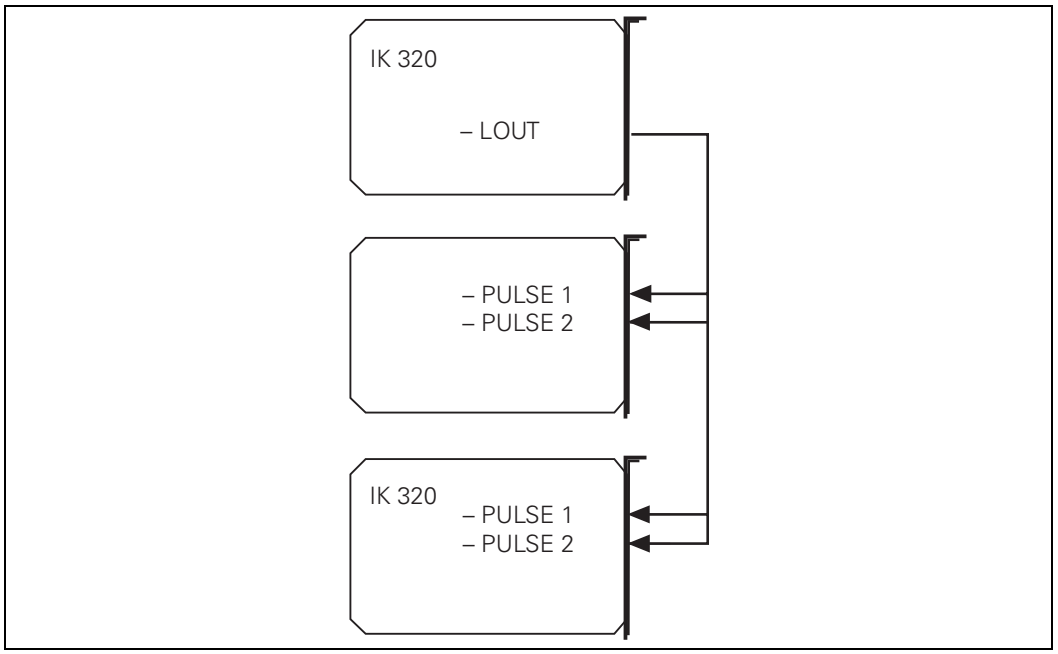
Note:
If the VMEbus processor is not capable of performing an ADO, one possible solution is to insert the jumper J3 in **one** of the cards in the group. This card will then return a –DTACK signal.

8.1.6 VMEbus Direct Latching over Several Groups

If you wish to synchronously latch the position values of cards which are set to different group addresses, proceed as follows:

- Connect the –LOUT outputs of the cards in the first group to the –PULSE1 and –PULSE2 inputs of the other cards.
- Store the position value of the cards of the first group with an ADO access to the group address.
- The cards of the other groups will be synchronously latched via the –LOUT outputs.

Propagation time of latch pulse to –LOUT : < 10 ns

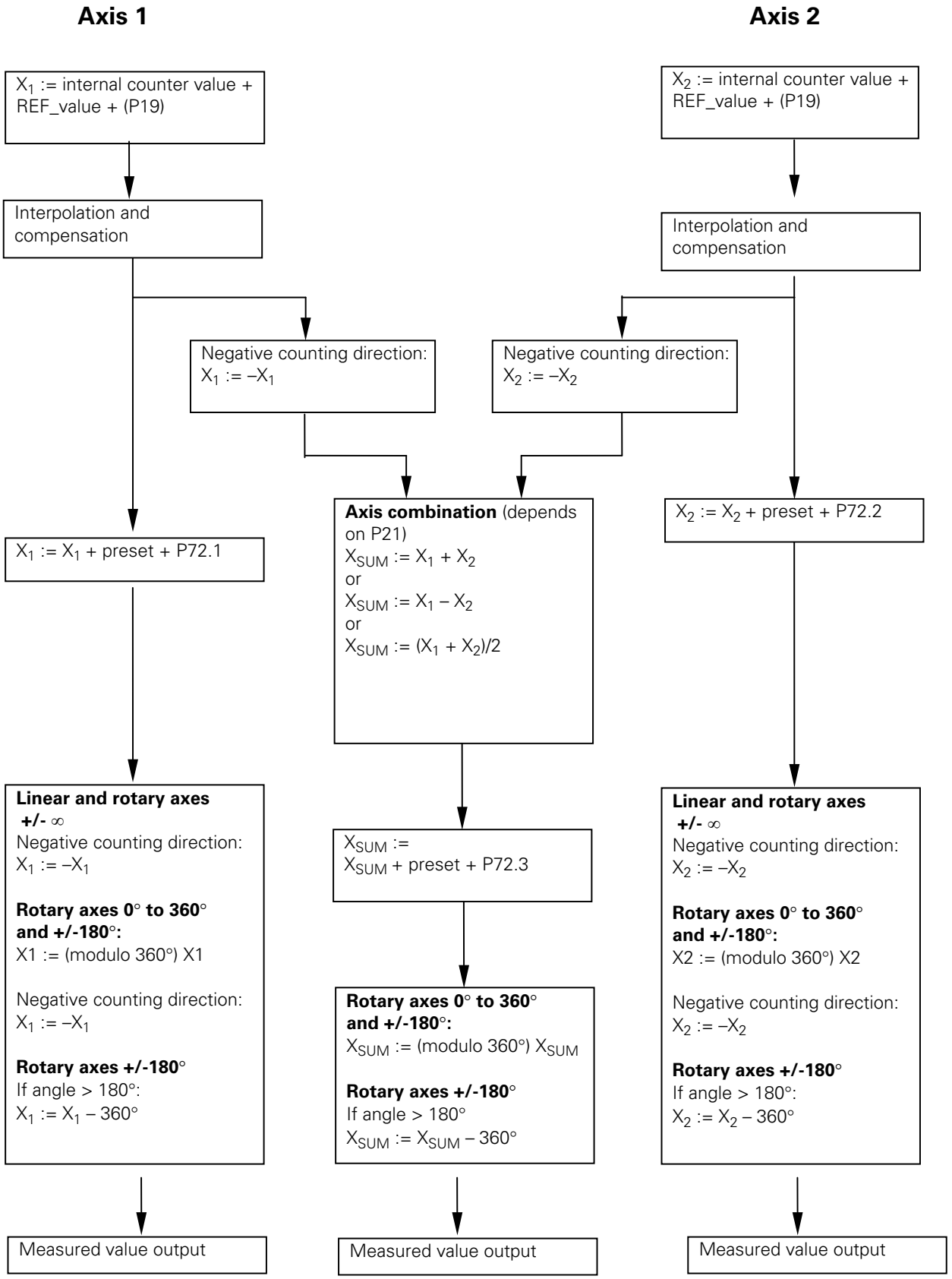


8.1.7 VMEbus Indirect Latching

A local interrupt on the IK 320 is generated by an ADO access to the interrupt address of address space A16. The CPU performs the function set in parameter P81: the functions $01, $02, $03 generate a latch signal in the corresponding axes.

8.2 Calculating the Position Value

The flowchart below shows how the IK 320 generates the measured value in increments.
X1 represents the measured value of the 1st axis and X2 the measured value of the 2nd axis.



The REF value is the absolute value of the first traversed reference mark on encoders with distance-coded reference marks. On encoders with only one reference mark the REF value = 0.

The preset is calculated as follows:
Preset = desired display value (P70/P71) – internal counter value – P72 – REF value

A value is calculated so that the value desired in parameters P70 or P71 appears at this location as the position value. This makes it possible, for example, to set the display to zero at any location, provided that before the preset calculation you define the value 0 as desired display value.

9. Programming

The programming of an IK 320 with two axes is shown in this description using a "BORLAND C" example. The program was written and tested on an industrial computer (manufactured by ROTEC, D-76411 Rastatt) with an INTEL 486 CPU (DOS Version 6.0), a VMEbus interface and BORLAND C++ compiler (Version 4.0).
The following files on the floppy disk provided are used to adapt the ISA bus to the VMEbus:

- VMEROTEC.H and
- VMEINIT.C

The data and function definitions in these files are not explained in greater detail, since they do not describe any functions of the IK 320.

The files

- IK320.H and
- IK320.C

contain the most important data and function definitions which you will need when working with the IK 320.

The files

- SAMPLE.H and
- SAMPLE.C

show a simple application with the functions from "IK320.C".

You can create an executable program by integrating the files

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

in a "project."

Essentially, a program for the IK 320 must perform the following functions:

- Initialize card
- Traverse reference points
- Display, store and evaluate position values

In addition, on commissioning compensation values must be entered to compensate for encoder signal deviations. You must repeat the compensation run after:

- A failure of the "stand-by" power supply or
- The replacement of an encoder or scanning head on an axis

The individual functions of the SAMPLE.C program are described below.

The heart of this example is the interrupt function **NewInterruptRoutine().** This function handles all interrupt causes of the IK 320. The function **Read-IK_Interrupt_Status()** reads the IK interrupt status (BA + $18). **Evaluate_IK_Interrupt-Status()** evaluates the interrupt origin.

VmeInit()
Initializes the VMEbus. This function is adapted to the industrial computer manufactured by ROTEC. You will have to write your own initialization function for the hardware you use.

InitIk320()
Initializes the IK 320. First the interrupt address of **INT_NR** is stored under **pOriginalInterruptVector.** Then the new interrupt function **NewInterruptRoutine,** which handles all IK interrupts, is installed. Then this function sets the parameters via **InitParams()** and performs the "Power On Self Test" (POST; VMEbus interrupt function $07). The POST has been completed successfully if the IK 320 returns the status $0700.

DisplayMessage() and DisplayError()
Display messages and errors which the IK 320 reports through the interrupt status.

TraverseOverReferencemark()
Activates the evaluation of the reference marks by means of the VMEbus interrupt function $0008 for axis 1 and $0009 for axis 2. Then you must traverse the axes over the reference marks. The IK 320 signals via interrupt status whether the reference marks have been crossed over: $0800 is the status "Reference mark of axis 1 was traversed" and $0900 is the status "Reference mark of axis 2 was traversed."

DisplayPositionValue()
Displays the positions of the two axes 1 and 2 on the screen. The function calls **SynchroPosTrigger()** for synchronous latching. The position value is determined in **Evaluate_IK_Interrupt_Status().** In the case of linear axes you must multiply this value by the signal period (e.g. by 0.002 mm), to obtain a display in mm. For rotary axes, multiply by 360°/signal periods per revolution.

CompensationRun()
Determines the compensation values to compensate for encoder signal deviations. The compensation run is started with **MasterInterrupt()** and the master interrupt function $0Bxx (axis 1) or $0Cxx (axis 2). Then you must move the axes at a speed which is as constant as possible. The IK 320 signals a successful compensation run via an interrupt with the status $0Bxx (axis 1) and $0Cxx (axis 2).

CompensationOnOff()
Activates compensation of the encoder signals via parameter P06.

RestoreOldInterruptVector()
Before you exit the program, RestoreOldInterruptVector() re-installs the original interrupt address.

9.1 The Header File SAMPLE.H

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

  DR. JOHANNES HEIDENHAIN GmbH, Traunreut, Germany

  Header File for SAMPLE.C

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

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

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

void  MainMenu(void);
```

9.2 Program Example SAMPLE.C

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

  DR. JOHANNES HEIDENHAIN GmbH, Traunreut, Germany

  Sample for IK 320
  V 1.00
  September 1995
 ------------------------------------------------------------------------*/
#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include "vmerotec.h"//Header file for ROTEC VME interface
#include "ik320.h"
#include "sample.h"

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

unsigned char extucErrorCode = 0, extucMessage = 0;

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

unsigned short usIntervalOfCompPoints, usNumberOfCompPoints;

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

void main()
{
  clrscr();

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

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

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

  _setcursortype(_NOCURSOR);

  do
   {
   if (extucErrorCode)
    {
    DisplayError();
    }
   if (extucMessage)
    {
    DisplayMessage();
    }
   clrscr();
   fflush(stdin);
   printf("1: Traverse over Reference Mark axis 1\n");
   printf("2: Traverse over Reference Mark axis 2\n");
   printf("3: Display Position Values\n");
   printf("4: Compensation Run axis 1\n");
   printf("5: Compensation Run axis 2\n");
   printf("6: Compensation On/Off\n");
```

```
   printf("0: End");

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

   cCharacter=getch();

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

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

9.3 The Header File IK320.H

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

  DR. JOHANNES HEIDENHAIN GmbH, Traunreut, Germany

  Header File for the Driver Unit IK320.C

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

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

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

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

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

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

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

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

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

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

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

#define  PAR_02_1    0x0106  //Definition axis 1
#define  PAR_02_2    0x0107  //Definition axis 2
#define  PAR_02_3    0x0108  //Definition for axis combination
#define  PAR_03      0x010A  //Bits of subdivision

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

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

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

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

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

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

#define  PAR_10      0x012E  //Latch enable

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

#define  PAR_21      0x0138  //Axis combination

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

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

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

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

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

#define  PAR_81      0x0100  // Master interrupt function


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

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

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

#define  INTSTAT        0x0018 //IK interrupt status

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

#define  EPR1_NOML      0x001E //Nominal CRC sum EPROM 1
#define  EPR1_ACTL      0x0022 //Actual CRC sum EPROM 1
#define  EPR2_NOML      0x0026 //Nominal CRC sum EPROM 2
#define  EPR2_ACTL      0x002A //Actual CRC SUM EPROM 2
```

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

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

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


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

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

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

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

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

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

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

9.4 The Functions in IK320.H

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

  DR. JOHANNES HEIDENHAIN GmbH, Traunreut, Germany

  Driver Unit for IK 320

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


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

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

/*---------------------------------------------------------------------
  Global variables
  ---------------------------------------------------------------------*/
static unsigned char staucDIP_Switch_II = DIP_SWITCH_SII;
```

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

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

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


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

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

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

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

  _enable();               //Enable DOS interrupt

  usAddress = CALCULATE_IK_ADDRESS(staucDIP_Switch_II);

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

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

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

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

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

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

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

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

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

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

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

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

  unsigned short usDummy, usAddress;
  long lDummy;

  usAddress = CALCULATE_IK_ADDRESS(ucDIP_Switch_II);

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

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

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

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

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

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

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

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

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

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

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

 case 0x0B:
  switch (stawbStatus.tb.uc0)
   {
   case 0x00:
```

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

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

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

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

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

   case 0x10:
    extucMessage = 0x08;
    break;

   case 0x11:
    extucMessage = 0x09;
    break;

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

   case 0xFE:
    extucErrorCode = 0x32;
    break;

   case 0xFF:
    extucErrorCode = 0x33;
    break;

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

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

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

  outport (usAddress + PAR_81,usFunction);

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

}//End MasterInterrupt

/*---------------------------------------------------------------------
  InitIk320
```

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

  //Disable hardware interrupts
  _disable();

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

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

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

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

  //Enable hardware interrupts
  _enable();

  SWITCH_VME_TO_A24_ADDRESS_SPACE(ucDIP_Switch_II);

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


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

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

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

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

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

  SetParam (ucDIP_Switch_II, PAR_05_1, 0x00008CA0,0);//Signal per. 1
  SetParam (ucDIP_Switch_II, PAR_05_2, 0x00000800,0);//Signal per. 2
  SetParam (ucDIP_Switch_II, PAR_05_3, 0x00000000,0);//Signal periods
```

```
                                                        //axis combination

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

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

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

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

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

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

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

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

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

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

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

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

}//End InitParams


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

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

```
 case PAR_02_3:
outportb (usBaseAddress+usOffsetAddress, (char) lData);
break;
 case PAR_03  :
outport (usBaseAddress+usOffsetAddress, (short)  lData);
break;
 case PAR_04_1:
outport (usBaseAddress+usOffsetAddress, (short)  lData);
break;
 case PAR_04_2:
outport (usBaseAddress+usOffsetAddress, (short)  lData);
break;
 case PAR_05_1:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
break;
 case PAR_05_2:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
break;
 case PAR_05_3:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
break;
 case PAR_06_1:
outportb (usBaseAddress+usOffsetAddress, (char) lData);
break;
 case PAR_06_2:
outportb (usBaseAddress+usOffsetAddress, (char) lData);
break;
 case PAR_07_1:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
break;
 case PAR_07_2:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
break;
 case PAR_08_1:
outport (usBaseAddress+usOffsetAddress, (short)  lData);
break;
 case PAR_08_2:
outport (usBaseAddress+usOffsetAddress, (short)  lData);
break;
 case PAR_09_1:
outport (usBaseAddress+usOffsetAddress, (short)  lData);
break;
 case PAR_09_2:
outport (usBaseAddress+usOffsetAddress, (short)  lData);
break;
 case PAR_10  :
outportb (usBaseAddress+usOffsetAddress, (char) lData);
break;
 case PAR_19_1:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
break;
 case PAR_19_2:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
break;
 case PAR_21  :
outportb (usBaseAddress+usOffsetAddress, (char) lData);
break;
 case PAR_30_1:
outportb (usBaseAddress+usOffsetAddress, (char) lData);
break;
 case PAR_30_2:
outportb (usBaseAddress+usOffsetAddress, (char) lData);
break;
 case PAR_70_1:
outport (usBaseAddress+usOffsetAddress,   (short) (lData >> 16));
outport (usBaseAddress+usOffsetAddress+2, (short) (lData & 0xffff));
outport (usBaseAddress+usOffsetAddress+4, sData);
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    printf("\r\t%10.4f\t%10.4f",stadPositionValue1,stadPositionValue2);
    staucAxis1WasRead =0;
    staucAxis2WasRead =0;
```

```
    }
  while (!kbhit());getch();
   clrscr();

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

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

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

```
  cAxis=getche();

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

  cCharacter=getche();

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

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

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

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

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

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

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

void DisplayError(void)
{
  gotoxy (1,23);
  switch (extucErrorCode)
   {
   case 0x01:
    puts ("Error: No latch on axis 1\n");
    break;
   case 0x02:
    puts ("Error: No latch on axis 2\n");
    break;
   case 0x03:
    puts ("Error: Double latch on axis 1\n");
    break;
   case 0x04:
    puts ("Error: Double latch on axis 2\n");
    break;
   case 0x05:
    puts ("Error: CRC correction value 1\n");
    break;
   case 0x06:
    puts ("Error: CRC correction value 2\n");
    break;
   case 0x07:
    puts("Error: parameter setting\n");
    break;
   case 0x08:
    puts("Error: CRC Eprom\n");
    break;
   case 0x09:
    puts ("Error: checksum EPROM 1\n");
    break;
   case 0x10:
    puts ("Error: checksum EPROM 2\n");
    break;
   case 0x11:
    puts ("Error: hardware\n");
    break;
   case 0x12:
    puts ("Error: wrong parameter setting\n");
    break;
   case 0x13:
    puts ("Error: axis 1 has no absolute reference\n");
    break;
   case 0x14:
    puts ("Error: axis 1 has wrong speed\n");
    break;
   case 0x15:
    puts ("Error: axis 1 has wrong position\n");
    break;
   case 0x16:
    puts ("Error: axis 1 has wrong direction\n");
    break;
   case 0x17:
```

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

10. Specifications of the IK 320

Mechanical data			
Dimensions	Double-height VME board, size B External dimensions: 262 mm x 187 mm x 20 mm		
Operating temperature **Storage temperature**	0 °C to 55 °C (32 °F to 131 °F) –40 °C to 75 °C (–40 °F to 167 °F)		
Electrical data			
VMEbus specification	ANSI/IEEE STD1014–1987, IEC 821 and 297 Double height board with J1 connector 1 slot Interrupter: D08(O) ROAK		
Addresses	Address space A16: Slave, D08(O) (port) Slave, ADO (synchronous latching) Memory requirement: 16 kilobytes per card Address space A24: Slave, D16, D08 (EO), 16 kilobytes Address space selectable via DIP switch		
Inputs/outputs			
Encoder inputs	**IK 320 V:** X1, X3: D-sub connector 15-pin, sinusoidal signals: 1 V_{PP} **IK 320 A:** X1, X3: D-sub connector 9-pin, sinusoidal signals: 11 μA_{PP} Input frequency: 50 kHz **IK 320.1:** X1, X3: D-sub connector 9-pin, sinusoidal signals 22 μA_{PP}		
Encoder outputs	**IK 320 V, IK 320 A:** X2, X4: D-sub connector 9-pin, sinusoidal signals 11 μA_{PP} (reference voltage = 0 V; connection of EXEs impossible) **IK 320.1:** X2, X4: D-sub connector 9-pin, sinusoidal signals 22 μA_{PP}		
External latch signals	X41: D-sub connector 9-pin 6 inputs –PULSE1, –PULSE2 –CONTACT1, –CONTACT2 –F1, –F2	U_{high}: 3 – 15 V	U_{low}: –0.5 – 1 V
	Output –LOUT	U_{high}: 2.4 – 5 V	U_{low}: 0 – 0.4 V
Signal interpolation	4096-fold		
Compensation of encoder signals	**Via 4096 compensation points**		
Data register for measured values	48-bit; only 44 bits are used for the measured value		
Interrupts	–IRQ1 to –IRQ7: selectable by jumper		
Power consumption	+12 V –12 V +5 V + 5 $V_{STANDBY}$	160 mA 160 mA 1.2 A 100 µA	

HEIDENHAIN

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

www.heidenhain.de

AT **HEIDENHAIN**
Dr.-Johannes-Heidenhain-Straße 5
83301 Traunreut, Deutschland
☎ (08669) 31 13 37
FAX (08669) 50 61

BE **HEIDENHAIN NV/SA**
Pamelse Klei 47,
1760 Roosdaal-Pamel, Belgium
☎ (054) 34 31 58
FAX (054) 34 31 73

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

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

CH **HEIDENHAIN (SCHWEIZ) AG**
Post Box
Vieristrasse 14
8603 Schwerzenbach, Switzerland
☎ (044) 806 27 27
FAX (044) 806 27 28

CN **HEIDENHAIN** (Tianjin)
Optics and Electronics Co. Ltd.
Room 808, The Exchange Beijing Tower 4
No. 118, Jian Guo Lu Yi
Chaoyang District
100022 Beijing, China
☎ (86) 10 6567 32 38
FAX (86) 10 6567 27 89

CZ **HEIDENHAIN s.r.o.**
Stremchová 16
SK 106 00 Praha 10, Czech Republic
☎ 272 658 131
FAX 272 658 724

DK **TP TEKNIK A/S**
Korskildelund 4
2670 Greve, Denmark
☎ (70) 10 09 66
FAX (70) 10 01 65

ES **FARRESA ELECTRONICA S.A.**
Les Corts, 36-38 bajos
08028 Barcelona, Spain
☎ 934 092 491
FAX 933 951 17

FI **HEIDENHAIN AB**
Mikkelänkallio 3
02770 Espoo, Finland
☎ (09) 86 76 476
FAX (09) 86 76 47 40

FR **HEIDENHAIN FRANCE sarl**
2, Avenue de la Cristallerie
92316 Sèvres, France
☎ 01 41 14 30 00
FAX 01 41 14 30 30

GB **HEIDENHAIN (G.B.) Limited**
200 London Road, Burgess Hill
West Sussex RH15 9RD, Great Britain
☎ (01444) 247 711
FAX (01444) 870 024

GR **MB Milionis Vassilis**
38. Scoufa Str.
St. Dimitrios
17341 Athens, Greece
☎ (0210) 933 66 07
FAX (0210) 934 96 60

HK **HEIDENHAIN LTD**
Unit 2, 15/F, APEC Plaza
49 Hoi Yuen Road
Kwun Tong
Kowloon, Hong Kong
☎ (852) 27 59 19 20
FAX (852) 27 59 19 61

HU **HEIDENHAIN Kereskedelmi Képviselet**
Hrivnák Pál utca 13.
RO 1237 Budapest, Hungary
☎ (1) 421 09 52
FAX (1) 421 09 53

IL **NEUMO VARGUS**
Post Box 57057
34-36, Itzhak Sade St.
Tel-Aviv 61570, Israel
☎ (3) 537 32 75
FAX (3) 537 21 90

IN **ASHOK & LAL**
Post Box 5422
12 Pulla Reddy Avenue
Chennai – 600 030, India
☎ (044) 26 15 12 89
FAX (044) 26 47 82 24

IT **HEIDENHAIN ITALIANA S.r.l.**
Via Asiago 14
20128 Milano, Italy
☎ 02 27 075-1
FAX 02 27 075-210

JP **HEIDENHAIN K.K.**
Kudan Center Bldg. 10th Floor
Kudankita 4-1-7, Chiyoda-ku
Tokyo 102-0073 Japan
☎ (03) 32 34-77 81
FAX (03) 32 62-25 39

KR **HEIDENHAIN LTD.**
Suite 1415, Family Tower Building
958-2 Yeongtong-Dong
Paldal-Gu, Suwon
442-470 Kyeonggi-Do, South Korea
☎ (82) 31 201 15 11
FAX (82) 31 201 15 10

MX **HEIDENHAIN CORPORATION MEXICO**
Av. Las Américas 1808
Fracc. Valle Dorado
20235, Aguascalientes, Ags., Mexico
☎ (449) 913 08 70
FAX (449) 913 08 76

NL **HEIDENHAIN NEDERLAND B.V.**
Post Box 92, 6710 BB EDE
Copernicuslaan 34, 6716 BM EDE
The Netherlands
☎ (0318) 581 800
FAX (0318) 581 870

NO **KASPO MASKIN AS**
Hoeggvn. 66
7036 Trondheim, Norway
☎ (073) 96 96 00
FAX (073) 96 96 01

PL **APS**
Popularna 56
02-473 Warszawa, Poland
☎ (22) 863 97 37
FAX (22) 863 97 44

PT **FARRESA ELECTRÓNICA LDA.**
Rua do Outeiro, 1315 1º M
4470 Maia, Portugal
☎ (22) 947 81 40
FAX (22) 947 81 49

SE **HEIDENHAIN AB**
Fittjavägen 23
14553 Norsborg, Sweden
☎ (08) 531 933 50
FAX (08) 531 933 77

SG **HEIDENHAIN PACIFIC PTE LTD.**
51, Ubi Crescent
Singapore 408593,
Republic of Singapore
☎ (65) 67 49-32 38
FAX (65) 67 49-39 22

TR **ORSEL LTD.**
Kusdili Cad. No. 43
Toraman Han, Kat 3
81310 Kadiköy/Istanbul, Turkey
☎ (216) 347 83 95
FAX (216) 347 83 93

TW **HEIDENHAIN Co., Ltd.**
No. 12-5, Gong 33rd Road
Taichung Industrial Park
Taichung 407, Taiwan, R.O.C.
☎ (886-4) 23 588 977
FAX (886-4) 23 588 978

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

ZA **MAFEMA SALES SERVICES C.C.**
107 - 16th Road Unit B3
Tillbury Business Park, Randjespark
Midrand, 1685, South Africa
☎ (011) 314 44 16
FAX (011) 314 22 89

286 941-24 · 1 · 7/2004 · S · Printed in Germany · Subject to change without notice