



**DIN EN ISO 9001:2000  
certified**



**ADDI-DATA GmbH  
Dieselstraße 3  
D-77833 OTTERSWEIER**



**Technical support:  
+49 (0)7223 / 9493 – 0**

***Preliminary version***

**Function description**

**ADDICOUNT APCI-/CPCI-1710**

**BiSS-Master**

Edition: 01.02-09/2005

## Product information

This manual contains the technical installation and important instructions for correct commissioning and usage, as well as production information according to the current status before printing. The content of this manual and the technical product data may be changed without prior notice. ADDI-DATA GmbH reserves the right to make changes to the technical data and the materials included herein.

## Warranty and liability

The user is not permitted to make changes to the product beyond the intended use, or to interfere with the product in any other way.

ADDI-DATA shall not be liable for obvious printing and phrasing errors. In addition, ADDI DATA, if legally permissible, shall not be liable for personal injury or damage to materials caused by improper installation and/or commissioning of the board by the user or improper use, for example, if the board is operated despite faulty safety and protection devices, or if notes in the operating instructions regarding transport, storage, installation, commissioning, operation, thresholds, etc. are not taken into consideration. Liability is further excluded if the operator changes the board or the source code files without authorisation and/or if the operator is guilty of not monitoring the permanent operational capability of working parts and this has led to damage.

## Copyright

This manual, which is intended for the operator and its staff only, is protected by copyright. Duplication of the information contained in the operating instructions and of any other product information, or disclosure of this information for use by third parties, is not permitted, unless this right has been granted by the product licence issued. Non-compliance with this could lead to civil and criminal proceedings.

## ADDI-DATA software product licence

Please read this licence carefully before using the standard software. The customer is only granted the right to use this software if he/she agrees with the conditions of this licence.

The software must only be used to set up the ADDI-DATA boards.

Reproduction of the software is forbidden (except for back-up and for exchange of faulty data carriers). Disassembly, decompilation, decryption and reverse engineering of the software are forbidden. This licence and the software may be transferred to a third party if this party has acquired a board by purchase, has agreed to all the conditions in this licence contract and the original owner does not keep any copies of the software.

## Trademarks

- ADDI-DATA is a registered trademark of ADDI-DATA GmbH.
- Turbo Pascal, Delphi, Borland C, Borland C++ are registered trademarks of Borland Insight Company.
- Microsoft C, Visual C++, Windows XP, 98, Windows 2000, Windows 95, Windows NT, EmbeddedNT and MS DOS are registered trademarks of Microsoft Corporation.
- LabVIEW, LabWindows/CVI, DasyLab, Diadem are registered trademarks of National Instruments Corp.
- CompactPCI is a registered trademark of PCI Industrial Computer Manufacturers Group.
- VxWorks is a registered trademark of Wind River Systems Inc.

# WARNING

**The following risks result from improper implementation and from use of the board contrary to the regulations:**



- ◆ **Personal injury**
- ◆ **Damage to the MSX-Box, PC and peripherals**
- ◆ **Pollution of the environment**

◆ **Protect yourself, the others and the environment!**

◆ **Read carefully the safety precautions (yellow leaflet).**

If this leaflet is not with the documentation, please contact us and ask for it.

◆ **Observe the instructions of the manual.**

Make sure that you do not forget or skip any step. We are not liable for damages resulting from a wrong use of the board.

◆ **Used symbols:**



## **IMPORTANT!**

designates hints and other useful information.



## **WARNING!**

It designates a possibly dangerous situation.

If the instructions are ignored the board, PC and/or peripheral may be destroyed.

<b>1</b>	<b>DEFINITION OF APPLICATION .....</b>	<b>7</b>
1.1	Intended use .....	7
1.2	Usage restrictions.....	7
1.3	Technical description .....	7
1.4	Function description .....	8
1.5	Used abbreviations .....	8
<b>2</b>	<b>BISS-MASTER .....</b>	<b>9</b>
2.1	BISS protocol .....	9
2.2	Technical data .....	9
2.2.1	Limit values .....	9
2.2.2	Limits .....	9
2.3	Function description .....	10
2.3.1	Block diagram .....	10
2.3.2	Common functions.....	10
2.3.3	Types of communication.....	10
2.3.4	Relations between the functionalities .....	11
2.4	Used signals .....	14
2.5	Connector pin assignment for all modules with BiSS-Master 14	
2.6	Connection example .....	15
2.7	I/O mapping of the BiSS-Master interface.....	15
2.8	Description of the I/O functions.....	16
2.8.1	READ-REGISTER .....	16
2.8.2	WRITE-REGISTER.....	19
<b>3</b>	<b>STANDARD SOFTWARE .....</b>	<b>23</b>
3.1	Define values.....	23
3.2	BiSS initialisation .....	24
3.3	BiSS commands .....	31
3.4	BiSS status .....	33
3.5	Read BiSS data .....	37
3.6	BiSS – register communication.....	39
<b>4</b>	<b>TUTORIAL .....</b>	<b>41</b>
4.1	Example: Calculation of important parameters .....	42
4.1.1	Datasheet of the BiSS encoder .....	42
4.1.2	Calculation of the parameters for initialising the slave.....	43

4.1.3	Calculation of the parameters for initialising the multicycle communication .....	45
4.1.4	Calculation of the angle position from the sensor data .....	47
<b>4.2</b>	<b>Reading the angle position of a BiSS encoder .....</b>	<b>48</b>
4.2.1	Step 1: Initialising the board .....	49
4.2.2	Step 2: Finishing the current activities .....	49
4.2.3	Step 3: Initialising the Master .....	49
4.2.4	Step 4: Initialising the slave .....	50
4.2.5	Step 5: Concluding the initialisation .....	50
4.2.6	Step 6: Requesting the sensor data .....	50
4.2.7	Step 7: Reading validity messages .....	50
4.2.8	Step 8: Reading the sensor data .....	51
4.2.9	Step 9: Releasing the handle of the board .....	51
<b>4.3</b>	<b>Reading multicycle data of a BiSS encoder .....</b>	<b>52</b>
4.3.1	Step 1: Initialising the board .....	54
4.3.2	Step 2: Finishing the current activities .....	54
4.3.3	Step 3: Initialising the Master .....	54
4.3.4	Step 4: Initialising the slave .....	55
4.3.5	Step 5: Initialising the multicycle data .....	55
4.3.6	Step 6: Concluding the initialisation .....	55
4.3.7	Step 7: Requesting multicycle data .....	55
4.3.8	Step 8: Requesting the sensor data and reading the validity messages .....	56
4.3.9	Step 9: Reading multicycle data .....	56
4.3.10	Step 10: Releasing the handle of the board .....	56
<b>4.4</b>	<b>Reading register values of a BiSS encoder .....</b>	<b>57</b>
	Step 1: Initialising the board .....	58
	Step 1: Initialising the board .....	59
4.4.1	Step 2: Finishing the current activities .....	59
4.4.2	Step 3: Initialising the Master .....	59
4.4.3	Step 4: Concluding the initialisation .....	60
4.4.4	Step 5: Initialising the register communication .....	60
4.4.5	Step 6: Requesting the register data .....	60
4.4.6	Step 7: Reading register data .....	61
4.4.7	Step 8: Releasing the handle of the board .....	61
<b>4.5</b>	<b>Writing register values of a BiSS encoder .....</b>	<b>62</b>
4.5.1	Step 1: Initialisation of the board .....	64
4.5.2	Step 2: Finishing the current activities .....	64
4.5.3	Step 3: Initialisation of the master .....	64
4.5.4	Step 4: Concluding the initialisation .....	65
4.5.5	Step 5: Initialisation of the register communication .....	65
4.5.6	Step 6: Filling register data .....	65
4.5.7	Step 7: Starting the register communication .....	66
4.5.8	Step 8: Releasing the handle of the board .....	66

## Figures

Fig. 2-1: Block diagram .....	10
Fig. 2-2: Connector pin assignment of the 50-pin SUB-D connector X1 .....	14
Fig. 2-3: Connection of a BiSS sensor.....	15

## Tables

Table 1-1: Delivered function descriptions.....	8
Table 2-1: Used signals .....	14
Table 2-2: I/O mapping of the BiSS-Master interface .....	15
Table 2-3: Status register .....	16
Table 2-4: Initialisation register sensor-/MC-data.....	19
Table 2-5: Initialisation register „register communication“ .....	20
Table 2-6: Initialisation register BiSS-Master.....	20
Table 2-7: Frequency division of the master clock .....	21
Table 2-8: Command register .....	22
Table 3-1: Define value .....	23

# 1 DEFINITION OF APPLICATION

## 1.1 Intended use

The board **APCI-/CPCI-1710** must be inserted in a PC with PCI PCI 5V/32-bit slots or Compact PCI/PXI computer with COMPACT PCI 5V/32-bit slots, which is used as electrical equipment for measurement, control and laboratory pursuant to the norm IEC 61010-1.

## 1.2 Usage restrictions

The board **APCI-/CPCI-1710** must not be used as safety related part for securing emergency stop functions

The board **APCI-/CPCI-1710** must not be used in potentially explosive atmospheres.

## 1.3 Technical description

This manual refers to the **APCI-1710** as well as to the **CPCI-1710/-1711** board. Make sure that you have received the following items:

- The CD 1 “Standard Software Drivers” with the ADDISET parameterizing program and the required software drivers.
- The CD 2 “Technical Manuals”. This CD contains the following:
  - The technical description **ADDICOUNT APCI-1710 / CPCI-1710: Function-programmable counter board for the PCI bus** (containing general information on the operation of the board)
  - A function description for each function which you want to program on the board **APCI-/CPCI-1710**
- The yellow leaflet "Safety precautions"

According to the used function you will find the required assignment and programming functions in the different manuals for each function.

**Table 1-1: Delivered function descriptions**

Function	PDF file (CD2 technical manuals)		Function description SET1710	CFG file
	German	English		
Incremental counter	Inkr_zähler_d.pdf	incr_counter_e.pdf	Incremental counter	inc_cpt.cfg
SSI	SSI_d.pdf	SSI_e.pdf	SSI	ssi.cfg
SSI monitor	SSI-Monitor_d	SSIMonitor_e.pfd	SSI_Monitor	ssi_mon.cfg
Chronos	chronos_d.pdf	chronos_e.pdf	Chronos	chronos.cfg
Counter/timer	Zähler_timer_d.pdf	counter_timer_e.pdf	counter/timer	82x54.cfg
TOR	TOR_d.pdf	TOR_e.pdf	TOR	tor.cfg
PWM	PWM_d.pdf	PWM_e.pdf	Pulse width modulation	PWM.cfg
TTL	TTL_IO_d.pdf	TTL_IO_e.pdf	TTL I/O	ttl_io.cfg
Digital I/O	dig_EA_d.pdf	dig_IO_e.pdf	Digital I/O	dig_IO.cfg
Pulse counter	Impulszähler_d.pdf	pulse_counter_e.pdf	Pulse counter	imp_cpt.cfg
ETM	ETM_d.pdf	ETM_e.dpf	Edge time measurement	etm.cfg
BiSS-Master	BISS-Master_d.pdf	BISS-Master_e.pdf	BiSS-Master	BISS.cfg

**Please note:**

The board **CPCI-1710/1711** is compatible with the board **APCI-1710** as far as the installation of the software is concerned. The ADDIREG and SET1710 programs make no difference between PCI and CompactPCI boards.

The API functions of the standard software are also identical.

## 1.4 Function description

Besides a global description of the functions this manual contains:

- the pin assignment of the front connector
- a list of the used signals
- the I/O mapping
- a chapter about the API software functions of the standard software
- tutorials

## 1.5 Used abbreviations

The signals on the 50-pin SUB-D connector refer always to one function module.

Please note the used abbreviations:

- UAS: Interference signal
- CLK: Clock
- REF: Reference point logic
- ENA: Enable

**C1+** is a signal for **function module 1**.



## 2 BISS-MASTER

### 2.1 BISS protocol

Please find more detailed information under [www.biss-interface.com](http://www.biss-interface.com).

### 2.2 Technical data

#### 2.2.1 Limit values

Operation with the BISS-Master is only possible on the boards **APCI-1710-10k20** and **CPCI-1710-10k20**.

The following values must be kept:

Max.clock cycle: ..... 5 MHz

Max. number of slaves: ..... 1

Furthermore, the values of the BISS-Interface specifications must be kept (see [www.biss-interface.com](http://www.biss-interface.com)).

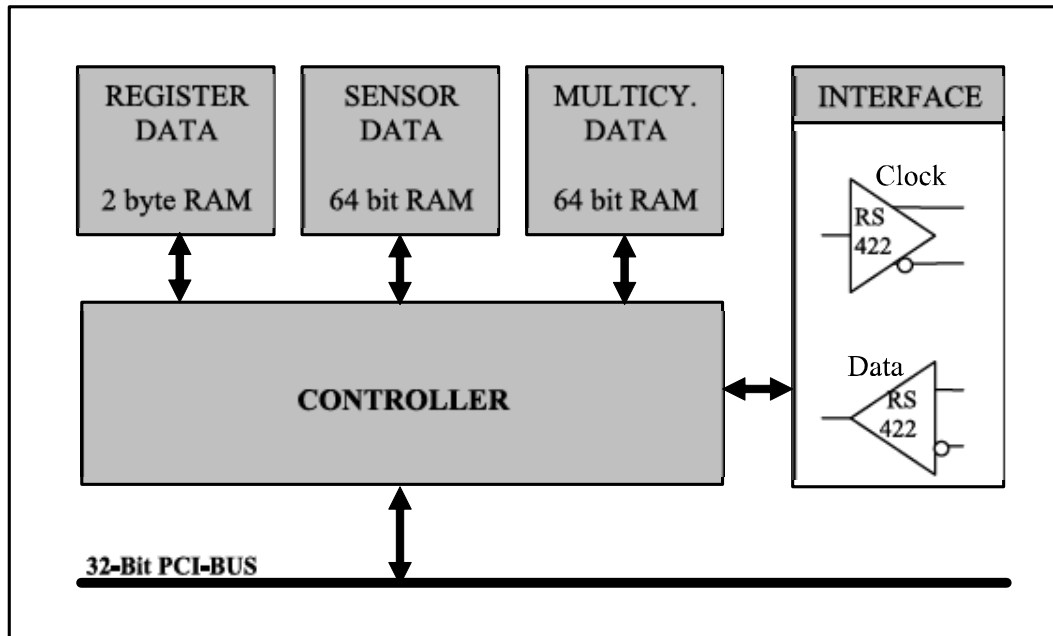
#### 2.2.2 Limits

- In this version of the BISS-Master only one channel is available, to which a BiSS-Sensor (slave) can be connected.
- The functionality “automatic sensor data request” (“automatische Sensordatenabfrage“), in which the sensor data is read out cyclically in a certain period of time, is not supported.
- In the register communication max. 8-bit registers can be written or read with one access.

## 2.3 Function description

### 2.3.1 Block diagram

Fig. 2-1: Block diagram



### 2.3.2 Common functions

Please find more detailed information under [www.biss-interface.com](http://www.biss-interface.com).

The following chapters contain only the most significant characteristics:

### 2.3.3 Types of communication

With the BiSS sensor interface either data can be read out very fast or registers can be written or read out. The switch between these both types of communication is realised by a time condition at the beginning of each communication cycle.

#### Sensor mode

In the sensor mode sensor data can be read out fast without the need of addressing.

In order not to interrupt the fast data transfer, if values shall be read out, which change slowly (e.g. from a temperature monitoring), there is the possibility to extend a data set of a communication with always one bit of the value that changes slowly. Therefore, several communication cycles are necessary (multicycle data, 'MCD'). In order to request a new multi-cycle-data value, the master finishes the communication cycle with a low level and not with a high level.

With the adequate initialisation of the BiSS-Master in the sensor mode, SSI sensors can be also operated.

## Register mode

In the register mode the registers can be written or read out in the several addressable sensors. When writing on registers, the data is transferred PWM coded, which allows a simple transfer of clock and data on the master line.

### 2.3.4 Relations between the functionalities

This chapter explains more detailed the relations of the several functions of the BiSS-Master. Please find more detailed examples in chapter “TUTORIAL (see chapter 4) and in the respecting supplied software samples. The respecting software functions are indicated in italics at the corresponding place. The chapter “Standard Software” contains detailed descriptions of the software functions with all parameters and return values.

## Initialisation

At the beginning all necessary initialisation must be realised.

At the initialisation of the master, the frequency of the master clock, as well as the type of sensor (BiSS or SSI) can be set. At the initialisation of the slave, the data length of the sensor, a possible sensor data adaptation and gray/binary conversion, as well as the polynomial and the transfer type of a possible CRC test are set.

The initialisation of the multicycle data „*i\_APCI1710\_BissInitMultiCycle*“ and of the register communication „*i\_APCI1710\_BissInitRegisterCommunication*“ is, according to the required operating mode, necessary. At the initialisation of the multicycle data, the data length of the multicycle data, a possible multicycle data adaptation and a gray/binary conversion, as well as the polynomial and the transfer type of a possible CRC-test are set.

During the initialisation of the register communication the start address, the access mode (read/write), the number of bytes to be transmitted (1, 2) and the slave-ID are set. As this version of the BiSS-Master is designed for only one slave, the slave-ID must always be set on the value '0'. According to the set number, 1 byte or 2 byte data are read or written through the adequate register.

After the successful initialisation of the mentioned functionalities, the initialisation must be completed over Bit4 of the command register

„*i\_APCI1710\_BissWriteCommandRegister*“. The completion through the command register is necessary only once at the beginning of the application. If during an application the register address is changed with the function „*i\_APCI1710\_BissInitRegisterCommunication*“, then the initialisation does not have to be completed once again.

## Commands

With the command register „*i\_APCI1710\_BissWriteCommandRegister*“ different commands can be executed.

The status of the commands must be checked with Bit0 of the status register „*i\_APCI1710\_BissReadSatusRegister*“ before the next command may be executed. Additionally, after each command the error bits of the status register should be read out in order to be able to react at possible errors in the correct manner (Example: See Tutorials).

### 1) Automatic sensor data request

With the command AGS (Bit0) an automatic sensor data request can be realised. However, this functionality is not available in the current version of the BiSS-Master. Therefore Bit 0 must always have the value ‘0’.

### 2) Sensor data request

With the commands GETSENS1 (Bit1) or GETSENS0 (Bit2) sensor data can be requested. The validity of the data and the status of the transfer can be read out through the status register „*i\_APCI1710\_BissReadSatusRegister*“.

With the command GETSENS1 the sensor data is requested once and can be read out after the successful transfer in the sensor data register „*i\_APCI1710\_BissReadSensorReceiveData*“. In addition to the sensor data, one bit of multicycle data is transferred. According to the length of the multicycle data, after a certain number of sensor data requests, the valid multicycle data are ready to be read out in the respecting register „*i\_APCI1710\_BissReadMultiCycleReceiveData*“. The validity of the data and the status of the transfer can be checked through the status register.

With the command GETSENS0 the sensor data are requested once, which can be read out after the successful transfer in the sensor data register „*i\_APCI1710\_BissReadSensorReceiveData*“. No multi cycle data is transferred additionally.

The status of the transfers must be checked with Bit0 of the status register „*i\_APCI1710\_BissReadSatusRegister*“ before the next command may be executed.

### 3) Register communication

With the command REG (Bit3) the register communication is started. Depending on the initialisation (see „Initialisation“) 1 byte (8 bit) or 2 byte (16 bit) are read or written. The data to write (16 bit value) must be filed in the corresponding register before the execution of the command.

„*i\_APCI1710\_BissWriteRegisterSendData*“. The data to be read (16 bit value) can be read in the correpsponding register after the execution of the command. „*i\_APCI1710\_BissReadRegisterReceiveData*“. The status of the register communication can be read through the status register.

In the BiSS sensor, register are available that are 1 byte broad. If during initialisation the communication is indicated with 2 byte. Then two registers can be read or written. The data are allocated as follows:

- **1 byte read access:**  
Bit0..7 of the read value is data of the register with the indicated start address. Bit8..15 have no meaning.
- **1 byte write access:**  
Bit0..7 of the value to be written, is written in the register with the indicated start address: Bit8..15 have no meaning.
- **2 byte read access:**  
Bit0..7 of the read values of the data of the register with the indicated start address. Bit 8...15 of the read value are the data from the register with the address: (start address + 1).
- **2 byte write access:**  
Bit0..7 of the value to be written is written into the register with the indicated start address. Bit8...15 of the value to be written is written into the register with the address: (start address +1).

The status of the transfers must be checked with Bit0 of the status register „*i\_APCI1710\_BissReadSatusRegister*“ before the next command may be executed.

#### 4) Initialisation

With the command INIT (Bit4) a previously executed initialisation is completed (see initialisation). If during an application the register communication is initialised newly („*i\_APCI1710\_BissInitRegisterCommunication*“), then the execution of this command is not required (see Sample08)

The status of the transfers must be checked with Bit0 of the status register „*i\_APCI1710\_BissReadSatusRegister*“, before the next command may be executed.

#### 5) Interrupt of the current activity

With the command BREAK (Bit7) the current activity of the BiSS-Master is interrupted and returns into the initial state.

The status of the transfers must be checked with Bit0 of the status register „*i\_APCI1710\_BissReadSatusRegister*“ before the next command may be executed.

### Status

With the status register „*i\_APCI1710\_BissReadSatusRegister*“ different information such as status and validity of the data can be requested.

### Validity messages

With the validity messages „*i\_APCI1710\_BissReadValidityRegister*“ the validity of the received sensor data and MultiCycle data can be checked.

## 2.4 Used signals

The function BiSS master occupies **one differential input** (channel C) and **one differential output** (channel A) of the respecting function module of the APCI-/CPCI-1710-10k20.

On one board you can operate max. 4 BiSS-Masters or one for each module.

**Table 2-1: Used signals**

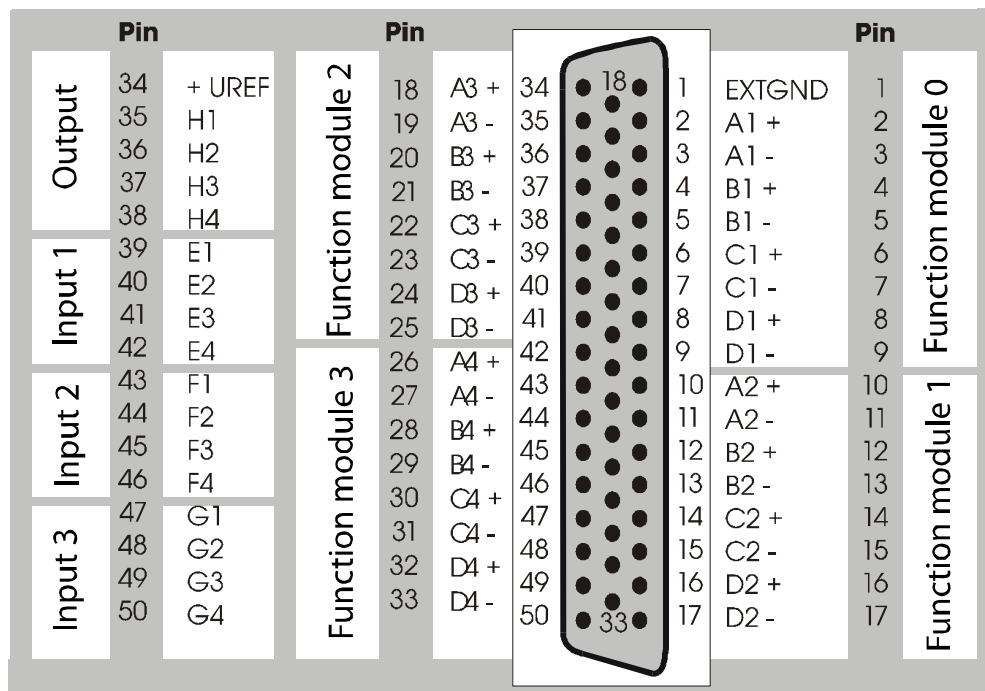
SIGNALS	ON THE CONNECTOR	POLARITY	FUNCTION
Input1_x	Cx +/-	Diff.	Digital input 1 (Data line from the slave to the master)
Output1_x	Ax +/-	Diff.	Digital output 1 (Clock line from the master to the slave)

x: Number of the function module.

## 2.5 Connector pin assignment for all modules with BISS-Master

The figure below is a connection example: The function "BiSS-Master" is implemented on all function modules

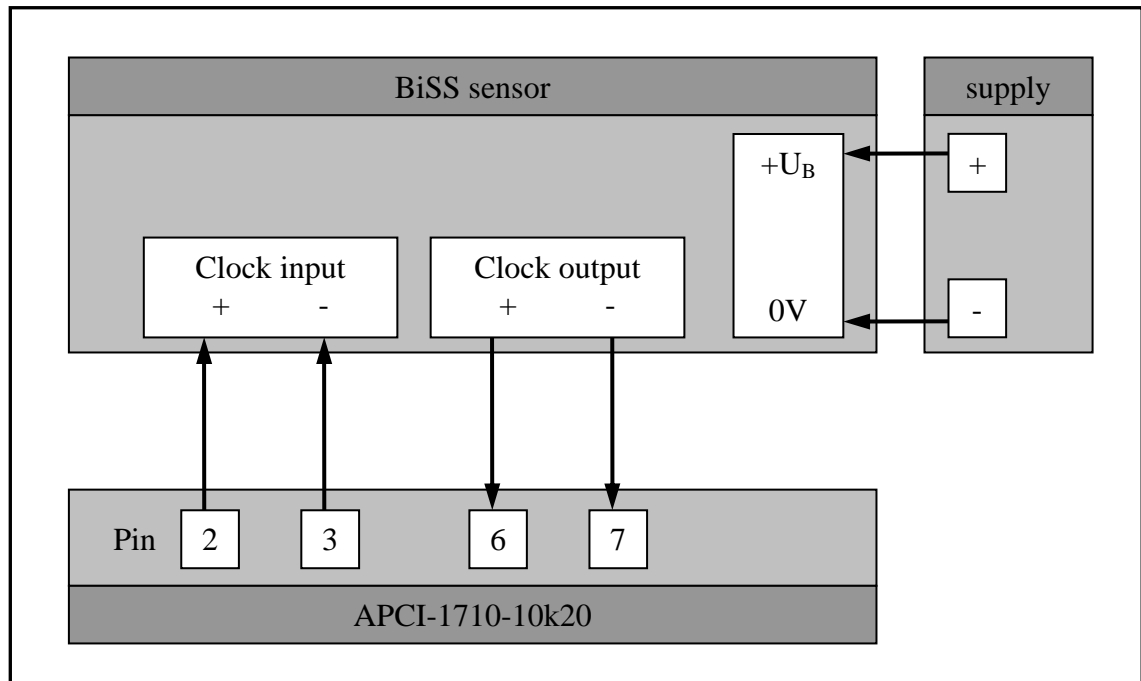
**Fig. 2-2: Connector pin assignment of the 50-pin SUB-D connector X1**



## 2.6 Connection example

A BiSS-Master is implemented on module 1 of an **APCI-1710-10k20** and controls the BiSS sensor. The sensor is supplied by an external voltage source.

**Fig. 2-3: Connection of a BiSS sensor**



## 2.7 I/O mapping of the BiSS-Master interface

**Table 2-2: I/O mapping of the BiSS-Master interface**

	IORD	IOWR
BASEx + 0	Sensor data register	-
BASEx + 4	Sensor data register	-
BASEx + 8	Multicycle data register	-
BASEx + 12	Multicycle data register	-
BASEx + 16	Register data	Register data
BASEx + 40	-	Initialisation register Sensor(Slave)-/ Multicycle- communication
BASEx + 44	-	Initialisation register communication
BASEx + 48	-	Initialisation register BISS-Master
BASEx + 52	Status register	-
BASEx + 56	Command register	Command register
BASEx + 60	Module identification	-

-: No function; **x**: Number of the function module.  
The accesses are always read or written in 32-bit

## 2.8 Description of the I/O functions

### 2.8.1 READ-REGISTER

#### Sensor data register (Base + 0)

In this register bit 0 – bit 31 of the sensor data is read out.

In addition to the sensor data and depending on the type of slave, error bits and data of the CRC-test are also available in the sensor data register. The information about such additional information can be obtained from the manufacturer of the slaves. Examples about the processing of the actual sensor data from the sensor data register can be seen in the delivered samples.

#### Sensor data register (Base + 4)

In this register bit 32 – bit 63 of the sensor data is read out.

#### Multicycle data register (Base + 8)

In this register bit 0 – bit 31 of the multicycle data is read out.

#### Multicycle data register (Base + 12)

In this register bit 32 – bit 63 of the multicycle data is read out.

#### Register data (Base + 16)

In this register the register data is read out. According to the setting in the initialisation register (Base + 44) 8 bit or 16 bit data is available.

#### Status register (Base + 52)

**Table 2-3: Status register**

BITS D0	0	Sensor/register data transmission not finished
	1	Sensor/register data transmission finished
BITS D1	0	Multicycle data transmission not finished
	1	Multicycle data transmission finished
BITS D2	0	Register data transmission finished
	1	Register data transmission not finished
BITS D3	0	CRC-error at register data transmission
	1	No CRC-error at register data transmission
BITS D4	0	CRC-error in the sensor data
	1	No CRC-error in the sensor data



BIT D5	0	CRC-error in the multicycle data
	1	No-CRC error in the multicycle data
BIT D6	0	Watchdog error at: - automatic sensor data transmission - Register data transmission
	1	No watchdog error
BIT D7	0	Error occurred
	1	No error occurred
BIT D8	0	Readable multicycle data not valid
	1	Readable multicycle data valid
BIT D9	0	Readable sensor data not valid
	1	Readable sensor data valid
BIT D15..D10		Not used
BIT 16	0	Level of the data line: „low“
	1	Level of the data line „high“
BIT 17		Current register data bit at slaves with BiSS model C
BIT 18		Fixed on „1“
BIT 19		Fixed on „0“
BIT 20		Fixed on „1“
BIT 21		Fixed on „0“
BIT 22		Fixed on „1“
BIT23		Fixed on „0“
BIT D24	0	No correctly transmitted byte at register data error
	1	One correctly transmitted byte at register data error
BIT D29..D25		Not used
BIT D30		No function in this version of the BiSS-Master
BIT D31	0	Multicycle data timeout has not run down
	1	Multicycle data timeout has run down

### Recognition register (Base + 56)

The command register can be read out. Description see „Write command register“. The reading of the command register is not used as software function. The reading is used only within the driver.

**Recognition register (Base + 60)**

The function and the revision are read out (read command, ASCII format)

**BASE + 60   "B"   "M"   "1"   "0"**

Meaning: BiSS-Master revision 1.0

## 2.8.2 WRITE-REGISTER

### Register data (Base + 16)

The register data to be written is filed in this register. Depending on the setting in the initialisation register (Base +44) 8 bit or 16 bit value are written.

### Initialisation register Sensor-/MC-data (Base + 40)

**Table 2-4: Initialisation register sensor-/MC-data**

BIT D5..D0	0	Bit length of the sensor data
BIT D6	0	No sensor data adaptation
	1	Sensor data adaptation
BIT D7	0	No gray/binary conversion for the sensor data
	1	Gray/binary conversion for the sensor data
BIT D14..D8		Sensor CRC polynomial
BIT D15	0	No inversion of the sensor-CRC-bit
	1	Inversion of the sensor-CRC-bits
BIT D21..D16		Bit length of the multicycle data
BIT D22	0	No multicycle data adaptation
	1	Multicycle data adaptation
BIT D23	0	No gray/binary conversion for the multicycle data
	1	Gray/binary conversion for the multicycle data
BIT D30..D24		Multicycle-CRC polynomial
BIT D31	0	No inversion of the multicycle-CRC-bits
	1	Inversion of the multicycle-CRC-bits

**Initialisation register „register communication“ (Base + 44)****Table 2-5: Initialisation register „register communication“**

BIT D6..D0		Register address
BIT D7	0	Registers are read at access
	1	Registers are written at access
BIT D9..D8		Number of bytes at write/read access 0: one byte 1: two bytes 2: Not available in this version of the BiSS-Master 3: Not available in this version of the BiSS-Master
BIT D10		Not used
BIT D13..D11		Slave-ID In this version of the BiSS-Master always value: „0“
BIT D14		REGVERS In this version of the BiSS-Master always value: „0“
BIT D15		MSEL In this version of the BiSS-Master always value: „0“
BIT D31..D16		Not used

**Initialisation register BiSS-Master (Base + 48)****Table 2-6: Initialisation register BiSS-Master**

BIT D3..D0		Frequency division (see Table 2-7)
BIT D7..D4		Not used
BIT D8	0	BiSS-protocol model: A or B
	1	BiSS-protocol model: C
BIT D9	0	Protocol type: BiSS
	1	Protocol type: SSI
BIT D31..D10		Not used

**Table 2-7: Frequency division of the master clock**

VALUE BIT D3..D0	Master frequency Sensor mode and SSI	Master frequency Register mode
0	16.5 MHz / 2	16.5 MHz / 64
1	16.5 MHz / 4	16.5 MHz / 128
2	16.5 MHz / 6	16.5 MHz / 192
3	16.5 MHz / 8	16.5 MHz / 256
4	16.5 MHz / 10	16.5 MHz / 320
5	16.5 MHz / 12	16.5 MHz / 384
6	16.5 MHz / 14	16.5 MHz / 448
7	16.5 MHz / 16	16.5 MHz / 512
8	16.5 MHz / 18	16.5 MHz / 576
9	16.5 MHz / 20	16.5 MHz / 640
10	16.5 MHz / 22	16.5 MHz / 704
11	16.5 MHz / 24	16.5 MHz / 768
12	16.5 MHz / 26	16.5 MHz / 832
13	16.5 MHz / 28	16.5 MHz / 896
14	16.5 MHz / 30	16.5 MHz / 960
15	16.5 MHz / 32	16.5 MHz / 1024

**CAUTION!**

The mentioned limit frequency of the board and the limit frequencies of the sensors must be kept.

**Command register (Base + 56)****Table 2-8: Command register**

BIT D0		Automatic sensor data request In this version of the BiSS-Master always set to value „0“
BIT D1	0	No sensor data enquiry
	1	Single sensor data enquiry with low-level at the end of the cycle (request of multicycle data)
BIT D2	0	No sensor data enquiry
	1	Single sensor data enquiry with high-level at the end of the cycle (no request of multicycle data)
BIT D3	0	No register access
	1	Execution of a register access
BIT D4	0	No sensor initialisation
	1	Sensor initialisation
BIT D6 - D5		Not used
BIT D7	0	No termination of the current activity
	1	Termination of the current activity
BIT D31..D8		Not used

## 3 STANDARDSOFTWARE

### 3.1 Define values

**WICHTIG!**

Please keep in mind the following style conventions:

Function: "i\_APCI1710\_SetBoardInformation"

Variable *ui\_Address*

**Table 3-1: Define value**

Define name	Decimal value	Hexadecimal value
DLL_COMPILER_C	1	1
DLL_COMPILER_VB	2	2
DLL_COMPILER_PASCAL	3	3
DLL_LABVIEW	4	4
APCI1710_DISABLE	0	0
APCI1710_ENABLE	1	1

## 3.2 BiSS initialisation

### 1) i\_APCI1710\_BissInitMaster (...)

#### Syntax:

```
<Return Wert> = i_APCI1710_BissInitMaster
                                (BYTE      b_BoardHandle,
                                BYTE      b_Module,
                                BYTE      b_FrequencyDivision,
                                BYTE      b_ModeOfOperation)
```

#### Parameter:

##### -Input:

BYTE	b_BoardHandle	Handle of the board <b>APCI-/CPCI-1710</b>
BYTE	b_Module	Number of the module to be configured (0 to 3)
BYTE	b_FrequencyDivision	Selection of the frequency division (0 to 15)
BYTE	b_ModeOfOperation	Selection of the operation mode (0 to 3)
	Bit0: BISSMOD	0: BiSS 1: SSI
	Bit1: SELSSI	0: BiSS model A or B 1: BiSS model C

##### -Output:

There is no output.

#### Task:

Configures the BiSS-Master of the selected module (*b\_Module*).

Call this function before calling any other functions that accesses the SSI monitor.

#### Calling convention:

ANSI C:

```
int          i_ReturnValue;
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_APCI1710_BissInitMaster    (b_BoardHandle,
                                              0,
                                              10,
                                              0);
```

#### Return value:

0: No error  
 -1: Handle parameter of the board is wrong  
 -2: The selected module is wrong  
 -3: The module is no BiSS-Master  
 -4: The selection of the frequency division is wrong  
 -5: The selection of the operation mode is wrong



## 2) i\_APCI1710\_BissInitSlave (...)

### Syntax:

```
<Return Wert> = i_APCI1710_BissInitSlave
                        (BYTE      b_BoardHandle,
                        BYTE      b_Module,
                        BYTE      b_SensorData,
                        BYTE      b_SensorCRC)
```

### Parameter:

#### -Input:

BYTE	b_BoardHandle	Handle of the board <b>APCI-/CPCI-1710</b>
BYTE	b_Module	Number of the module to be configured (0 to 3)
BYTE	b_SensorData	Selection of the sensor data (0 to 255) Bit0-5: SDLEN <sup>1</sup> (bit length of the sensor data 0 to 63) Bit6: ENSENS (adaptation for the sensor data) 0: Not available 1: Available Bit7: GRAYS (gray/binary conversion for the sensor data) 0: Disabled 1: Enabled (required for the SSI encoder)
BYTE	b_SensorCRC	Selection of the sensor-CRC (0 to 255) Bit0-6: SENSRCPPOLY <sup>2</sup> (CRC-polynomial for testing the sensor data 0-127) Bit7: INVCRCs (transmission of the sensor CRC-bits) 0: Not inverted 1: Inverted

#### -Output:

There is no output.

### Task:

Configures the BiSS slave of the selected module (*b\_Module*).

<sup>1</sup> The data length must be entered less 1, i.e. for 64 data bits it must be entered 63.

<sup>2</sup> If as CRC polynomial 000 0000b is entered, no CRC test is executed. As the last bit of a CRC polynomial always is 1, it is not registered in the polynomial register, but added automatically in the Master. Therefore a CRC polynomial of max 8 bit length is possible. If not the whole polynomial length is required, then the polynomial (without the last 1) must be entered right-justified and the leading places must be completed with 0. Example: The CRC polynomial 10 0011b is saved as 001 0001b.

**Calling convention:**

ANSI C:

```
int          i_ReturnValue;  
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_APCI1710_BissInitSlave (b_BoardHandle,  
0,  
90,  
161);
```

**Return value:**

0: No error

-1: Handle parameter of the board is wrong

-2: The selected module is wrong

-3: The module is no BiSS-Master

### 3) i\_APCI1710\_BissInitMultiCycle (...)

#### Syntax:

```
<Return Wert> = i_APCI1710_BissInitSlave
                        (BYTE      b_BoardHandle,
                        BYTE      b_Module,
                        BYTE      b_MultiCycleData,
                        BYTE      b_MCDCRC)
```

#### Parameter:

##### -Input:

BYTE	b_BoardHandle	Handle of the board APCI-/CPCI-1710
BYTE	b_Module	Number of the module to be configured (0 to 3)
BYTE	b_MultiCycleData	Selection of the multicycle data (0 to 225) Bit0-5: MDLEN <sup>1</sup> (bit length of the multicycle data 0 to 63) Bit6: ENMCD (adaptation for multicycle data) 0: Not available 1: Available Bit7: GRAYM (gray/binary conversion for the multicycle data) 0: Disabled 1: Enabled (required for SSI encoder)
BYTE	b_MCDCRC	Selection of the sensor CRC (0 to 255) Bit0-6: MCDCRCPOLY <sup>2</sup> (CRC polynomial for testing the multicycle data 0-127) Bit7: INVCRCM (transmission of the multicycle CRC bits) 0: Not inverted 1: Inverted

##### -Output:

There is no output.

#### Task:

Configures the multicycle data transmission of the selected module (*b\_Module*).

<sup>1</sup> The data length must be entered less 1, i.e. for 64 data bits, it must be entered 63.

<sup>2</sup> If as CRC polynomial 000 0000b is entered, no CRC test is executed. As the last bit of a CRC polynomial is 1, it is not registered in the polynomial register, but added automatically in the Master. Therefore a CRC polynomial of max 8 bit length is possible. If not the whole polynomial length is required, then the polynomial (without the last 1) must be entered right-justified and the leading places must be completed with 0. Example: The CRC polynomial 10 0011b is saved as 001 0001b.

**Calling convention:**

ANSI C:

```
int          i_ReturnValue;  
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_APCI1710_BissInitMultiCycle (b_BoardHandle,  
                                                0,  
                                                71,  
                                                137);
```

**Return value:**

0: No error

-1: Handle parameter of the board is wrong

-2: The selected module is wrong

-3: The module is no BiSS-Master

**4) i\_APCI1710\_BissInitRegisterCommunication (...)****Syntax:**

```
<Return Wert> = i_APCI1710_BissInitRegisterCommunication
                    (BYTE      b_BoardHandle,
                     BYTE      b_Module,
                     BYTE      b_StartAddress,
                     BYTE      b_WNR,
                     BYTE      b_CountOfBytes,
                     BYTE      b_SlaveID)
```

**Parameter:****-Input:**

BYTE	b_BoardHandle	Handle of the board <b>APCI-/CPCI-1710</b>
BYTE	b_Module	Number of the module to be configured (0 to 3)
BYTE	b_StartAddress	Selection of the start address (0 to 127)
BYTE	b_WNR	Operation mode (0 to 1) 0: Read access 1: Write access
BYTE	b_CountOfBytes	Selection of the number of bytes (0 to 1) 0: 1 byte is read/written 1: 2 byte are read/written
BYTE	b_SlaveID	Selection of the slave-ID (0 to 7)

**-Output:**

There is no output.

**Task:**

Initialised the register communication of the selected module (*b\_Module*).

**Calling convention:**

ANSI C:

```
int          i_ReturnValue;
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_APCI1710_BissInitRegisterCommunication
                    (b_BoardHandle,
                     0,
                     103,
                     0,
                     0,
                     0);
```

**Return value:**

0: No error

-1: Handle parameter of the board is wrong

-2: The selected module is wrong

-3: The module is no BiSS-Master

-4: The selection of the start address is wrong

-5: The selection of the operation mode is wrong

-6: The selection of the number of bytes is wrong

-7: The selection of the slave-ID is wrong

### 3.3 BiSS commands

#### 1) i\_APCI1710\_BissWriteCommandRegister (...)

##### Syntax:

```
<Return Wert> = i_APCI1710_BissWriteCommandRegister
                (BYTE      b_BoardHandle,
                 BYTE      b_Module,
                 BYTE      b_CommandRegister)
```

##### Parameter:

##### -Input:

BYTE	b_BoardHandle	Handle of the board <b>APCI-/CPCI-1710</b>
BYTE	b_Module	Number of the module to be configured (0 to 3)
BYTE	b_CommandRegister	Selection of the command register (0 to 255)
	Bit0:	AGS (automatic sensor data transmission) 0: Disabled 1: Enabled Is not available in this version of the BiSS-Master → value 0 transmitted
	Bit1:	GETSENS1 (single sensor data enquiry with low-level at the end of the cycle → request of new multicycle data) 0: No enquiry 1: New enquiry
	Bit2:	GETSENS0 (single sensor data request with high-level at the end of the cycle → no request of new multicycle data) 0: No enquiry 1: New enquiry
	Bit3:	REG (register access) 0: Execute no register access 1: Execute register access
	Bit4:	INIT (sensor initialisation) 0: Execute no initialisation 1: Execute initialisation
	Bit5:	UNUSED
	Bit6:	UNUSED
	Bit7:	BREAK (termination of the current activities) 0: No termination of the current activities 1: Termination of the current activities

**-Output:**

There is no output.

**Task:**

Gives the BiSS-Master of the selected module (*b\_Module*) different commands.

**Calling convention:**

ANSI C:

```
int          i_ReturnValue;  
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_APCI1710_BissWriteCommandRegister  
                (b_BoardHandle, 0, 128);
```

**Return value:**

- 0: No error
- 1: Handle parameter of the board is wrong
- 2: The selected module is wrong
- 3: The module is no BiSS-Master
- 4: Timeout
- 5: System error (when reading the system time)



### 3.4 BiSS status

#### 1) i\_APCI1710\_BissReadStatusRegister (...)

##### Syntax:

```
<Return Wert> = i_APCI1710_BissReadStatusRegister
                (BYTE      b_BoardHandle,
                 BYTE      b_Module,
                 PBYTE     pb_StatusInformation,
                 PBYTE     pb_ChannelStatus,
                 PBYTE     pb_RegisterMessages)
```

##### Parameter:

##### -Input:

BYTE	b_BoardHandle	Handle of the board <b>APCI-/CPCI-1710</b>
BYTE	b_Module	Number of the module to be configured (0 to 3)

##### -Output:

PBYTE	pb_StatusInformation	Status message (0 to 255)
	Bit0:	EOT (signing of the terminated senro and register data transfer) 0: Not terminated 1: Terminated
	Bit1:	MCDEND (End of the multicycle data transfer) 0: Not terminated 1: Terminated
	Bit2:	REGEND (End of the register data transfer) 0: Not terminated 1: Terminated
	Bit3:	nREGERR (CRC-error at register data transfer) 0: Error 1: No error
	Bit4:	nSENSERR (CRC-erro at sensor data transfer) 0: Error 1: No error
	Bit5:	nMCDERR (CRC-error at multicycle data transfer) 0: Error 1: No error
	Bit6:	nWDERR (watchdog error at automatic sensor data transfer register data transfer) 0: Error 1: No error

		Bit7:	nERR (general error message)
		0:	Error
		1:	No error
PBYTE	pb_ChannelStatus		Channel status (0 to 1)
		Bit0:	SL (Status of the data line from the slave to the master)
		0:	Low level
		1:	High level
PBYTE	pb_RegisterMessages		Register message (0 to 255)
		Bit0:	REGBYTES <sup>1</sup> (number of the correctly transferred register bytes in the case of error)
		0:	All bytes transferred flawlessly
		1:	Only one byte transferred flawlessly
		Bit1:	UNUSED
		Bit2:	UNUSED
		Bit3:	UNUSED
		Bit4:	UNUSED
		Bit5:	UNUSED
		Bit6:	REG (current register data bit at slave with BiSS model C)
		Bit7:	MCDTIMEOUT <sup>2</sup> (MCD-Timeout)
		0:	Not run down
		1:	Run down

**Task:**

Reads the status information of the selected module (*b\_Module*).

**Calling convention:**

ANSI C:

```
int          i_ReturnValue;
unsigned char b_BoardHandle;
unsigned char b_StatusInformation;
unsigned char b_ChannelStatus;
unsigned char b_RegisterMessages;
```

```
i_ReturnValue = i_APCI1710_BissReadStatusRegister (b_BoardHandle,
0,
```

<sup>1</sup> In the case of flawless transmission this bit is 0, in other case the number of register bytes is indicated that were transmitted flawlessly.

<sup>2</sup> A new MCD can be done firstly after a MCD timeout has run down; if a MCD request is started before, the slaves, which operate with the BiSS protocol model C, evaluate this as a register data transmission.

&b\_StatusInformation,  
&b\_ChannelStatus,

&b\_RegisterMessages);

**Return value:**

- 0: No error
- 1: Handle parameter of the board is wrong
- 2: The selected module is wrong
- 3: The module is no BiSS-Master

## 2) i\_APCI1710\_BissReadValidityRegister (...)

### Syntax:

```
<Return Wert> = i_APCI1710_BissReadValidityRegister
                (BYTE      b_BoardHandle,
                 BYTE      b_Module,
                 PBYTE     pb_VValidityMessages)
```

### Parameter:

#### -Input:

BYTE	b_BoardHandle	Handle of the board <b>APCI-/CPCI-1710</b>
BYTE	b_Module	Number of the module to be configured (0 to 3)

#### -Task:

PBYTE	pb_VValidityMessages	Gültigmeldung (0 bis 3)
	Bit0:	MVALID (validity of the read out multicycle data) 0: Not valid 1: Valid
	Bit1:	SVALID (validity of the read out sensor data) 0: Not valid 1: Valid

### Task:

Reads the validity information of the selected module (*b\_Module*).

### Calling convention:

ANSI C:

```
int          i_ReturnValue;
unsigned char b_BoardHandle;
unsigned char b_VValidityMessages;
```

```
i_ReturnValue = i_APCI1710_BissReadValidityRegister (b_BoardHandle,
                                                    0,
                                                    &b_VValidityMessages);
```

### Return value:

0: No error  
-1: Handle parameter of the board is wrong  
-2: The selected module is wrong  
-3: The module is no BiSS-Master

## 3.5 Read BiSS data

### 1) i\_APCI1710\_BissReadSensorReceiveData (...)

#### Syntax:

```
<Return Wert> = i_APCI1710_BissReadSensorReceiveData
                (BYTE      b_BoardHandle,
                 BYTE      b_Module,
                 PULONG    pul_SensorReceiveData_low,
                 PULONG    pul_SensorReceiveData_high)
```

#### Parameter:

##### -Input:

BYTE	b_BoardHandle	Handle of the board <b>APCI-/CPCI-1710</b>
BYTE	b_Module	Number of the module to be configured (0 to 3)

##### -Output:

PULONG	pb_SensorReceiveData_low	Sensor data (Bit31...0)
PULONG	pb_SensorReceiveData_high	Sensor data (Bit63...32)

#### Task:

Reads the received sensor data of the selected module (*b\_Module*).

#### Calling convention:

ANSI C:

```
int          i_ReturnValue;
unsigned char b_BoardHandle;
unsigned long b_SensorReceiveData_low;
unsigned long b_SensorReceiveData_high;
```

```
i_ReturnValue = i_APCI1710_BissReadSensorReceiveData
                (b_BoardHandle,
                 0,
                 &b_SensorReceiveData_low,
                 &b_SensorReceiveData_high);
```

#### Return value:

0: No error  
 -1: Handle parameter of the board is wrong  
 -2: The selected module is wrong  
 -3: The module is no BiSS-Master

## 2) i\_APCI1710\_BissReadMultiCycleReceiveData (...)

### Syntax:

```
<Return Wert> = i_APCI1710_BissReadMultiCycleReceiveData
                    (BYTE      b_BoardHandle,
                     BYTE      b_ModulNbr,
                     PULONG    pul_MultiCycleReceiveData_low,
                     PULONG    pul_MultiCycleReceiveData_high)
```

### Parameter:

#### -Input:

BYTE	b_BoardHandle	Handle of the board <b>APCI-/CPCI-1710</b>
BYTE	b_ModulNbr	Number of the module to be configured (0 to 3)

#### - Output

There is no output.

### Task:

PULONG	pb_MultiCycleReceiveData_low	Multicycle data (Bit31...0)
PULONG	pb_MultiCycleReceiveData_high	Multicycle data (Bit63...32)

### Calling convention:

ANSI C:

```
int          i_ReturnValue;
unsigned char b_BoardHandle;
unsigned long b_MultiCycleReceiveData_low ;
unsigned long b_MultiCycleReceiveData_high ;
```

```
i_ReturnValue = i_APCI1710_BissReadMultiCycleReceiveData
                    (b_BoardHandle,
                     0,
                     &b_MultiCycleReceiveData_low,
                     &b_MultiCycleReceiveData_high);
```

### Return value:

0: No error  
 -1: Handle parameter of the board is wrong  
 -2: The selected module is wrong  
 -3: The module is no BiSS-Master

### 3.6 BiSS – register communication

#### 1) i\_APCI1710\_BissReadRegisterReceiveData (...)

##### Syntax:

```
<Return Wert> = i_APCI1710_BissReadRegisterReceiveData
                    (BYTE      b_BoardHandle,
                     BYTE      b_Module,
                     PWORD     pw_ReceiveData)
```

##### Parameter:

##### -Input:

BYTE	b_BoardHandle	Handle of the board <b>APCI-/CPCI-1710</b>
BYTE	b_Module	Number of the module to be configured (0 to 3)

##### -Output:

PWORD	pw_ReceiveData	Register data (bit15...0)
-------	----------------	---------------------------

##### Task:

Reads the received register data of the selected module (*b\_Module*).

##### Calling convention:

ANSI C:

```
int                i_ReturnValue;
unsigned char      b_BoardHandle;
unsigned short int w_ReceiveData;
```

```
i_ReturnValue = i_APCI1710_BissReadRegisterReceiveData
                    (b_BoardHandle,
                     0,
                     &w_ReceiveData);
```

##### Return value:

0: No error  
 -1: Handle parameter of the board is wrong  
 -2: The selected module is wrong  
 -3: The module is no BiSS-Master

## 2) i\_APCI1710\_BissWriteRegisterSendData (...)

### Syntax:

```
<Return Wert> = i_APCI1710_BissWriteRegisterReceiveData
                    (BYTE      b_BoardHandle,
                     BYTE      b_Module,
                     WORD      w_SendData)
```

### Parameter:

#### -Input:

BYTE	b_BoardHandle	Handle of the board <b>APCI-/CPCI-1710</b>
BYTE	b_Module	Number of the module to be configured (0 to 3)
WORD	w_SendData	Register data (bit 15..0)

#### -Output:

There is no output.

### Task:

Saves the register data to be written of the selected module (*b\_Module*).

### Calling convention:

ANSI C:

```
int          i_ReturnValue;
unsigned char b_BoardHandle;
```

```
i_ReturnValue = i_APCI1710_BissWriteRegisterReceiveData
                    (b_BoardHandle,
                     0,
                     123);
```

### Return value:

0: No error  
 -1: Handle parameter of the board is wrong  
 -2: The selected module is wrong  
 -3: The module is no BiSS-Master



## 4 TUTORIAL

The tutorials show the different application possibilities of the BiSS-Master in examples. Here the application is created step by step together with the user in order to facilitate the first use of the function module. Of course, the scope of the presented tutorial can be extended with each BiSS-Master function that is available.

In order to present the tutorials as brief and clear as possible, the used functions will not be described more detailed. The contents of the technical description of the board **APCI-/CPCI-1710** and of the function description of the BiSS-Master should be known.

The tutorials refer to the programming language C and require a successful board installation, the loading of the function module and the connection of an adequate sensor.

The board installation and the loading of function module are described in the technical description of the board. A connection example of the BiSS-Master is shown in chapter 2.6 of the function description.

The first part of the tutorial shows with an example how the most important parameters of the initialisation must be calculated.

## 4.1 Example: Calculation of important parameters

When initialising the sensor, firstly sensor-specific parameters must be converted and then transmitted. When reading different information, these must be filtered and masked out of the received sensor data. The following example shows the different steps when calculating such parameters of a BiSS encoder.

### 4.1.1 Datasheet of the BiSS encoder

Resolution singleturn:	13 bit
Resolution multiturn:	12 bit
Transmitted additional information:	2 error-bits
Polynomial for sensor data test:	$1000011_b$
Size of the multicycle data:	8 bit
Polynomeal for multicycle data test:	$10011_b$

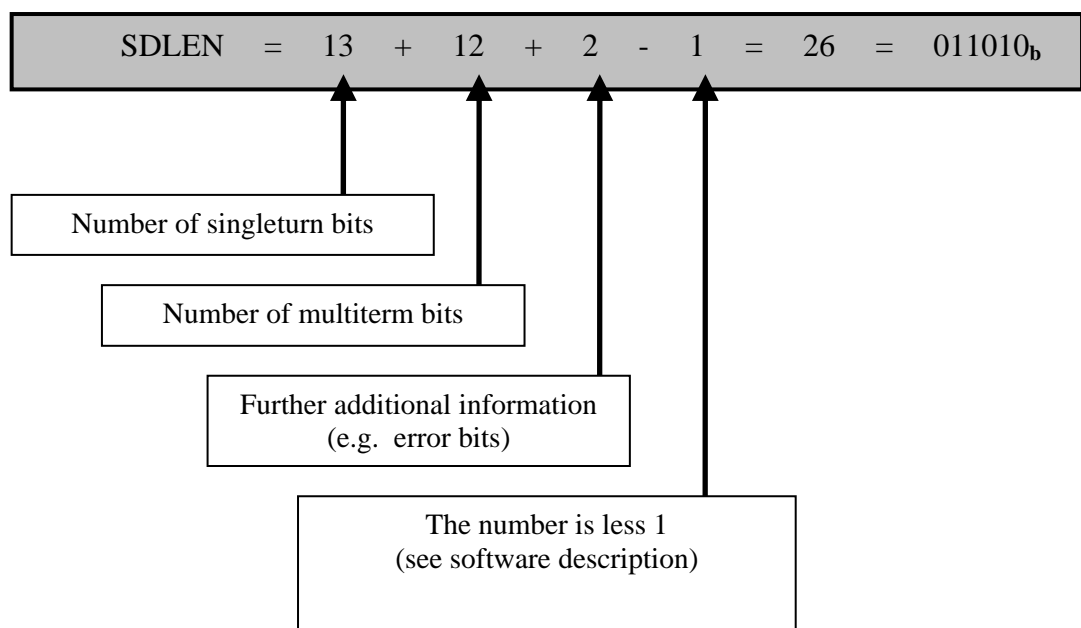
### 4.1.2 Calculation of the parameters for initialising the slave

#### 1) Parameter: b\_SensorData

The parameter b\_SensorData is a 8 bit value that is composed out of the following:

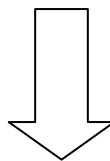
Bit0-5:	SDLEN	(bit length of the sensor data)
Bit6:	ENSENS	(adaptation for sensor data)
Bit7:	GRAYS	(gray-binary – conversion for sensor data)

#### Bit0-5: SDLEN



**Bit6:** ENSENS = 1 (adaptation for sensor data is available)

**Bit7:** GRAYS = 0 (gray-binary – conversion is disabled)



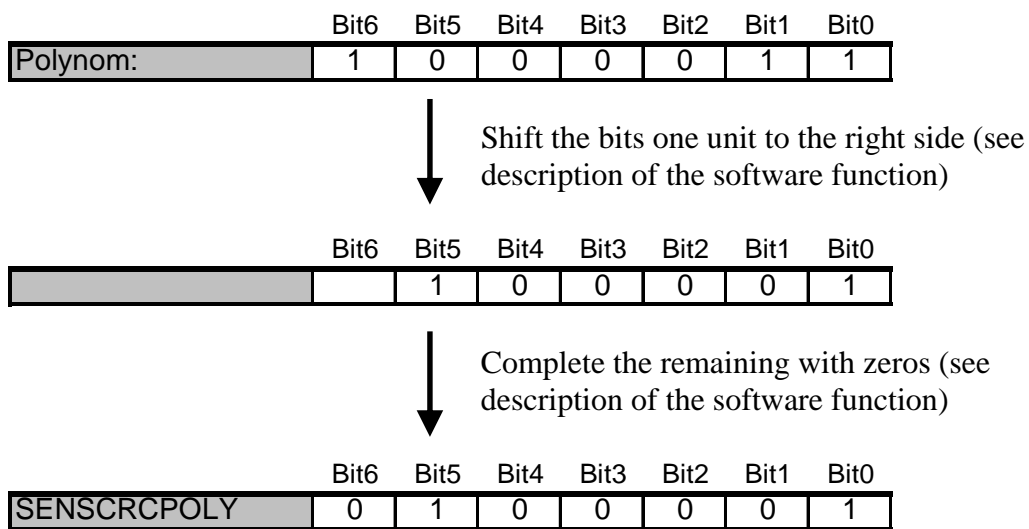
	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
b_SensorData	0	1	0	1	1	0	1	0

## 2) Parameter: b\_SensorCRC

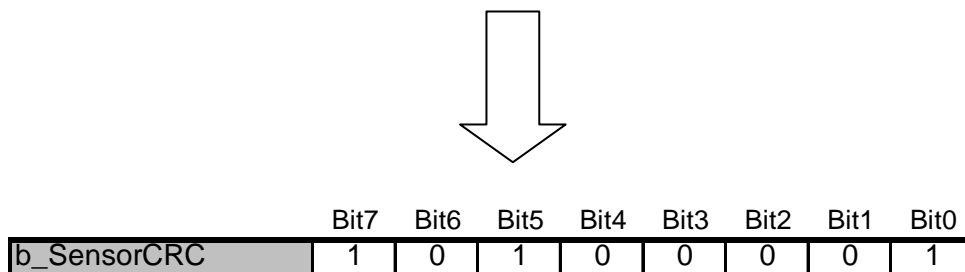
The parameter b\_SensorCRC is an 8 bit value, that is composed out of the following:

Bit0-6: SENSRCRPOLY (CRC polynomial for sensor data test)  
 Bit7: INVCRCs (Type of transmission of the sensor CRC bits)

### Bit0-6: SENSRCRPOLY



Bit7: INVCRCs = 1 (carry of the sensor CRC-bits inverted)



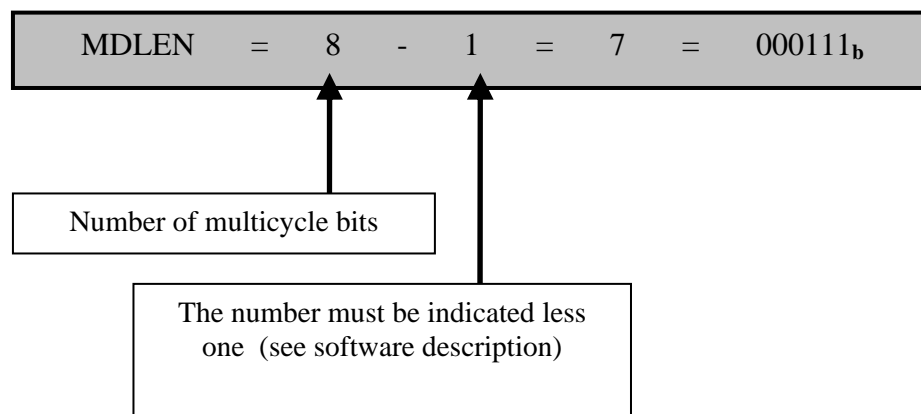
### 4.1.3 Calculation of the parameters for initialising the multicycle communication

#### 1) Parameter: b\_MultiCycleData

The parameter b\_MultiCycleData is a 8 bit value that is composed out of the following:

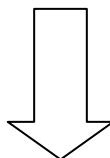
Bit0-5:	MDLEN	(bit length of the multicycle data)
Bit6:	ENMCD	(adaptation for multicycle data)
Bit7:	GRAYM	(gray-binary – conversion for multicycle data)

**Bit0-5:** MDLEN



**Bit6:** ENMCD = 1 (adaptation for multicycle data is available)

**Bit7:** GRAYM = 0 (gray-binary – conversion is disabled)

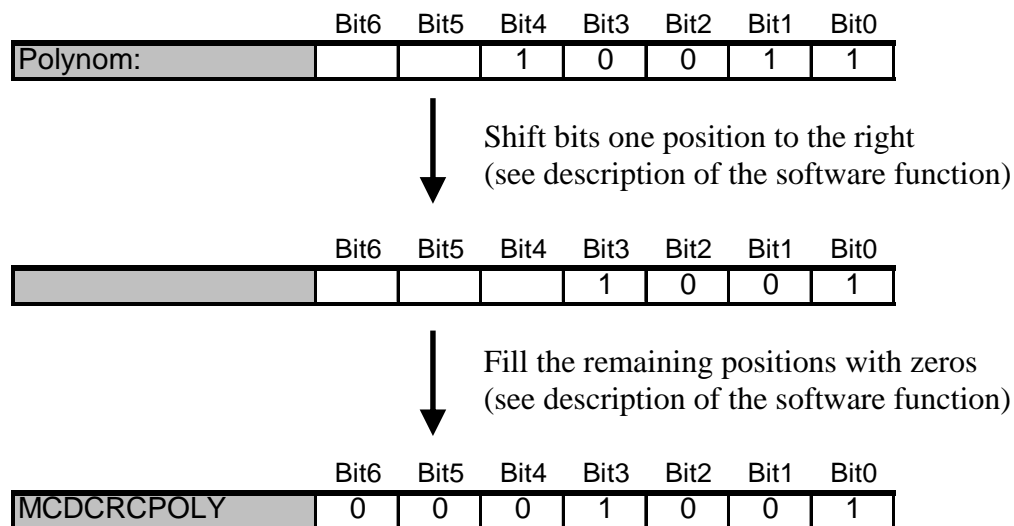


## 2) Parameter: b\_MCDCRC

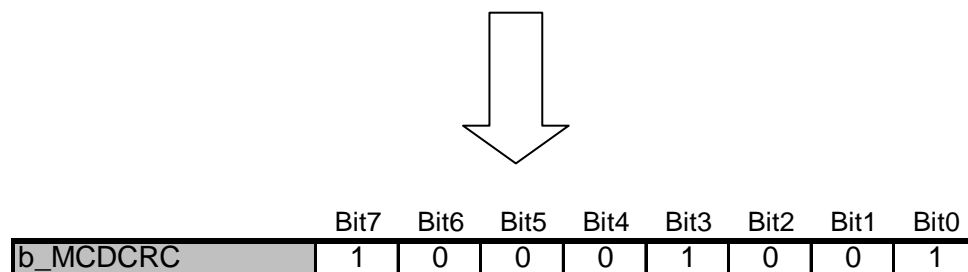
The parameter b\_MCDCRC is a 8-bit value that is composed of the following:

Bit0-6: MCDCRCPOLY (CRC-polynomial for MC-data test)  
 Bit7: INVCRCM (Type of transmission of the MC-CRC-bits)

### Bit0-6: MCDCRCPOLY



Bit7: INVCRCM = 1 (MultiCycle CRC-bits inverted)



#### 4.1.4 Calculation of the angle position from the sensor data

If a data request has been completed successfully, the respecting sensor data can be read out with the adequate software function

„i\_APCI1710\_BissReadSensorReceiveData“

The function returns two values:

1. pul\_SensorReceiveData\_low
2. pul\_SensorReceiveData\_high

Both return values correspond to the 64 bit sensor data value and are composed as follows.

Bit 63	<b>Sensordaten</b>		Bit 0
Bit 63	Bit 32	Bit 31	Bit 0
pul_SensorReceiveData_high		pul_SensorReceiveData_low	

From this sensor data value the sensor specific information can be read out. This specific example shows the singleturn and multiturn information.

Bit 63																Sensordaten																Bit 0																																																																																															
Bit 63																Bit 27																Bit 26																Bit 15																Bit 14																Bit 2																Bit 1																Bit 0															
Sonstiges z.B. CRC - Bits																																Multiturn																Singleturn																																Errorbits																																															

According to the sensor data manufacturer the CRC-bits and error bits can be transmitted or not transmitted. For further information about a CRC-test or possible error bits please refer to the datasheet of the sensor manufacturer.

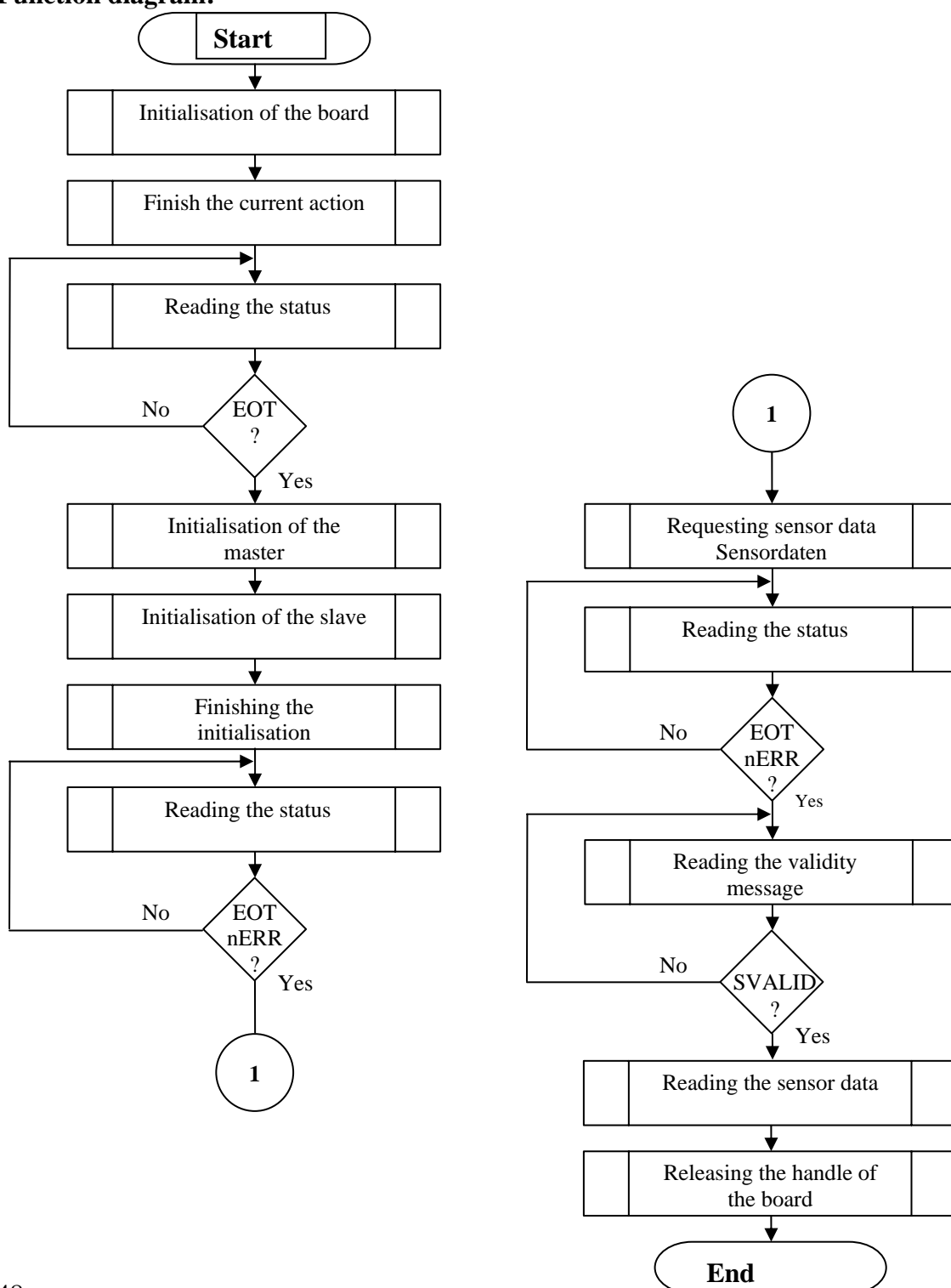
## 4.2 Reading the angle position of a BiSS encoder

BiSS encoders can be distinguished into sensor mode and register mode. In the sensor mode, the current angle position can be read out as follows.

The following description refers to the C-sample Sample04, in which the type of programming can be seen in detail.

The used functions are described in detail in the technical description of the manual or in the function description of the BiSS-Master.

### Function diagram:





### 4.2.1 Step 1: Initialising the board

Firstly, the board must be initialised. In order to show it more clearly, a function is created, which calls the functions that are provided by ADDI-DATA. In the C-samples this function “initialisation” is shown at the beginning. The following steps are required for the initialisation:

#### 1. Selection of the compiler:

With following functions the used compiler is indicated, in the following example a C-compiler.

```
i_ReturnValue = i_APCI1710_InitCompiler (DLL_COMPILER_C);
```

#### 2. Detecting the slot number:

```
i_ReturnValue = i_APCI1710_CheckAndGetPCISlotNumber  
(b_SlotNumberArray);
```

#### 3. Testing the slot number and detecting the handle of the board:

```
i_ReturnValue = i_APCI1710_SetBoardInformation (b_SlotNumberArray [0],  
                                                pb_BoardHandle);
```

### 4.2.2 Step 2: Finishing the current activities

At the beginning any current activity of the BiSS-Master, which could be possible at this time, is stopped:

```
i_ReturnValue = i_APCI1710_BissWriteCommandRegister (  
                                                        b_BoardHandle,  
                                                        b_Module,  
                                                        128);
```

The transmission status of commands must be checked after the execution over the bit EOT of the status register. The check of the nERR bit of the status register is not yet useful here as the error message could already be set by a former transmission (see Sample04).

### 4.2.3 Step 3: Initialising the Master

In this step the transmission clock and the type of the transmission (BiSS or SSI) are set.

```
i_ReturnValue = i_APCI1710_BissInitMaster ( b_BoardHandle,  
                                              b_Module,  
                                              b_FrequencyDivision,  
                                              b_ModeOfOperation);
```

#### 4.2.4 Step 4: Initialising the slave

In this step the sensor is initialised. The composition of the single parameters is described more detailed under „Standardsoftware“ and in the first part of the tutorial.

```
i_ReturnValue = i_APCI1710_BissInitSlave (    b_BoardHandle,  
                                              b_Module,  
                                              b_SensorData,  
                                              b_SensorCRC);
```

#### 4.2.5 Step 5: Concluding the initialisation

In this step the initialisation is concluded with the respecting command

```
i_ReturnValue = i_APCI1710_BissWriteCommandRegister (    b_BoardHandle,  
                                                         b_Module,  
                                                         16);
```

The transmission status of command must be check after the execution with the Bit EOT of the status register. Additionally, the Bit nERR must be checked in order to react to possible error (see Sample04).

The conclusion is only once at the beginning of a sample necessary in order to ensure the readiness of the slave.

#### 4.2.6 Step 6: Requesting the sensor data

With the following command the new sensor data is requested and the transmission between master and slave is started.

```
i_ReturnValue = i_APCI1710_BissWriteCommandRegister (    b_BoardHandle,  
                                                         b_Module,  
                                                         4);
```

The transmission status of the command must be checked after the execution with the bit EOT of the status register. Additionally, the Bit nERR must be checked, in order to react to possible errors (see Sample04).

#### 4.2.7 Step 7: Reading validity messages

After the successful transmission the validity of the sensor data must be checked.

```
i_ReturnValue = i_APCI1710_BissReadValidityRegister ( b_BoardHandle,  
                                                       b_Module,  
                                                       pb_VValidityMessages);
```

### 4.2.8 Step 8: Reading the sensor data

After the successful transmission and check of the validity messages, the sensor data received by the master can be read.

```
i_ReturnValue = i_APCI1710_BissReadSensorReceiveData (  
                                                    b_BoardHandle,  
                                                    b_Module,  
                                                    pul_SensorReceiveData_low,  
                                                    pul_SensorReceiveData_high);
```

### 4.2.9 Step 9: Releasing the handle of the board

Before the user program is finished, the handle of the board must be released again with the following function:

```
i_ReturnValue = i_APCI1710_CloseBoardHandle (b_BoardHandle);
```

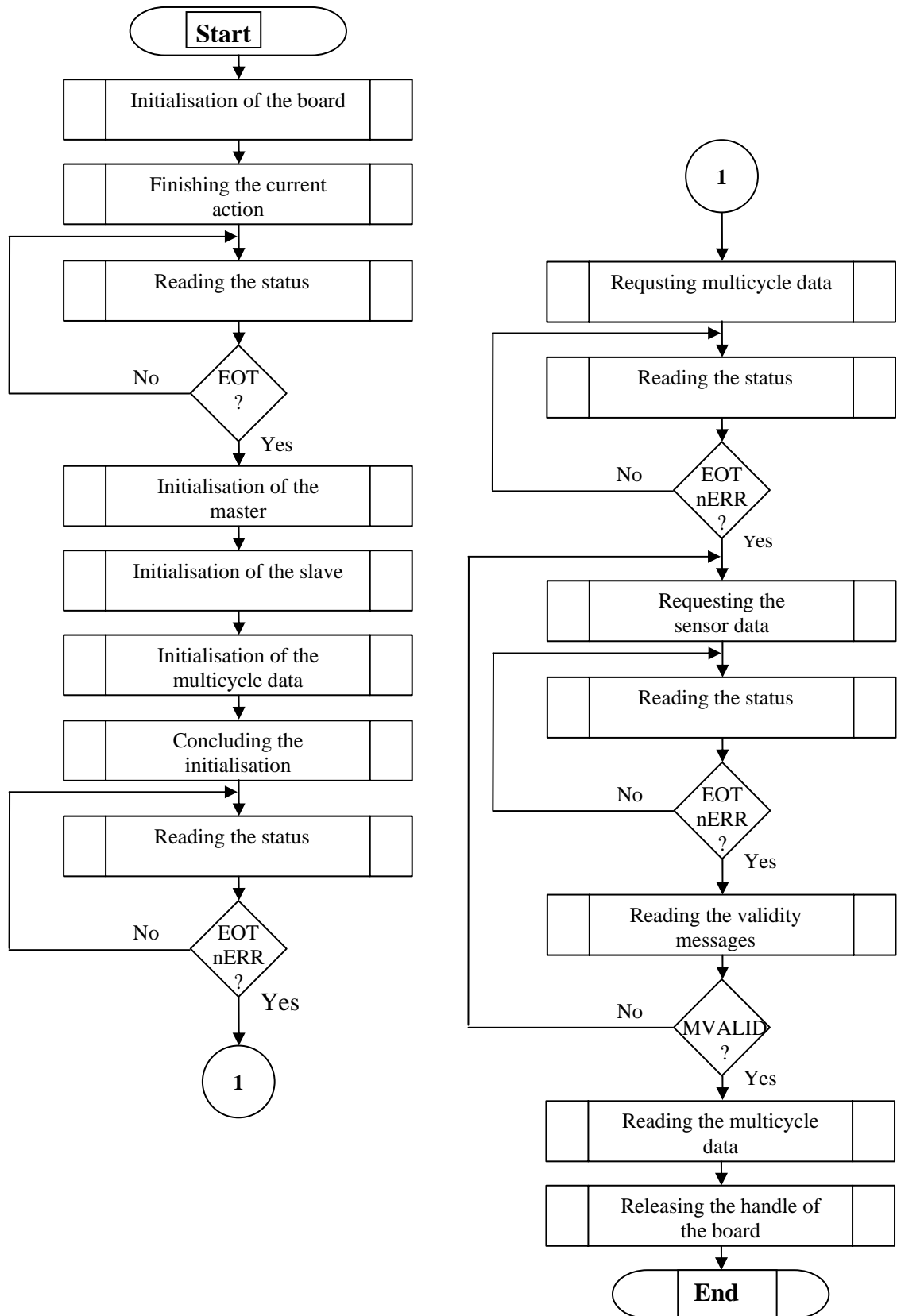
## 4.3 Reading multicycle data of a BiSS encoder

In the following example a multicycle value is read out. The example refers to the C-sample Sample06, in which the type of programming can be seen in detail.

At the multicycle data transfer always one bit of multicycle data is added to a sensor data transfer. According to the length of the multicycle data a certain number of sensor data transfers are required in order to obtain valid multicycle data. The first sensor data transfer must be started with the command “2”, then with the command “4” further sensor data transfers are executed. After each sensor data transfer the validity register is read out. If new and valid multicycle data are available, the transfer of sensor data is stopped and the multicycle data is read with the respecting software function.

The used functions are described detailed in the technical description of the board or in the function description of the BiSS-Master.

Function diagram:



### 4.3.1 Step 1: Initialising the board

Firstly, the board must be initialised. In order to show this more clearly, a function is created, which calls the functions that are provided by ADDI-DATA. In the C-samples this function “initialisation” is shown at the beginning. The following steps are required for the initialisation:

**1. Selecting the compiler:**

With the following function the used compiler is indicated, in this example a C-compiler.

```
i_ReturnValue = i_APCI1710_InitCompiler (DLL_COMPILER_C);
```

**2. Detecting the slot number:**

```
i_ReturnValue = i_APCI1710_CheckAndGetPCISlotNumber  
(b_SlotNumberArray);
```

**3. Testing the slot number and detecting the handle of the board:**

```
i_ReturnValue = i_APCI1710_SetBoardInformation (b_SlotNumberArray  
[0] ,  
pb_BoardHandle);
```

### 4.3.2 Step 2: Finishing the current activities

At the beginning an activity that is possible at the current time is terminated. Hereto the following command is executed:

```
i_ReturnValue = i_APCI1710_BissWriteCommandRegister (  
                    b_BoardHandle,  
                    b_Module,  
                    128);
```

The transmission status of command must be checked after the execution with the Bit EOT of the status register. The check of the bit nERR is here not yet useful, because from a previous transmission the error message could be set (see Sample06).

### 4.3.3 Step 3: Initialising the Master

In this step the transmission clock and the type of transmission (BiSS or SSI) are set.

```
i_ReturnValue = i_APCI1710_BissInitMaster (     b_BoardHandle,  
                                            b_Module,  
                                            b_FrequencyDivision,  
                                            b_ModeOfOperation);
```

#### 4.3.4 Step 4: Initialising the slave

In this step the sensor is initialised. The composition of the single parameters is described more detailed under „standardsoftware“ and in the first part of the tutorial.

```
i_ReturnValue = i_APCI1710_BissInitSlave (    b_BoardHandle,  
                                              b_Module,  
                                              b_SensorData,  
                                              b_SensorCRC);
```

#### 4.3.5 Step 5: Initialising the multicycle data

In this step the multicycle data transmission is initialised. The composition of the single parameters is described more detailed under „Standardsoftware“ and in the first part of the tutorial.

```
i_ReturnValue = i_APCI1710_BissInitMultiCycle (    b_BoardHandle,  
                                                    b_Module,  
                                                    b_MultiCycleData,  
                                                    b_MCDCRC);
```

#### 4.3.6 Step 6: Concluding the initialisation

In this step the initialisation is concluded with the respecting command.

```
i_ReturnValue = i_APCI1710_BissWriteCommandRegister (  
                                                    b_BoardHandle,  
                                                    b_Module,  
                                                    16);
```

The transmission status of the command must be checked after the successful execution with the Bit EOT of the status register. Additionally, the bit nERR must be checked in order to be able to react to possible errors (see Sample06).

This conclusion of an initialisation is only once at the beginning of a sample necessary in order to ensure the readiness of the slave.

#### 4.3.7 Step 7: Requesting multicycle data

With the following command the new multicycle data are requested and the transmission between master and slave is started.

```
i_ReturnValue = i_APCI1710_BissWriteCommandRegister (  
                                                    b_BoardHandle,  
                                                    b_Module,  
                                                    2);
```

The transmission status of the command must be checked after the execution of the bit EOT of the status register. Additionally, the bit nERR must be checked in order to be able to react to possible errors (see Sample06).

#### 4.3.8 Step 8: Requesting the sensor data and reading the validity messages

In this step sensor data are requested in a loop as long as new and valid multicycle data are signaled by the validity messages.

```
i_ReturnValue = i_APCI1710_BissWriteCommandRegister (
    b_BoardHandle,
    b_Module,
    4);
```

The transmission status of the command must be checked after the execution with the bit EOT of the status register. Additionally, the bit nERR must be checked in order to be able to react to possible errors (see Sample06).

With the following function the validity of multicycle data is requested:

```
i_ReturnValue = i_APCI1710_BissReadValidityRegister ( b_BoardHandle,
    b_Module,
    pb_VValidityMessages);
```

#### 4.3.9 Step 9: Reading multicycle data

After the successful transmission and checking of the validity messages, the multicycle data received by the master can be read.

```
i_ReturnValue = i_APCI1710_BissReadMultiCycleReceiveData (
    b_BoardHandle,
    b_Module,
    pul_MultiCycleReceiveData_low,
    pul_MultiCycleReceiveData_high);
```

#### 4.3.10 Step 10: Releasing the handle of the board

Before the user program is finished the handle of the board must be released again with the following function:

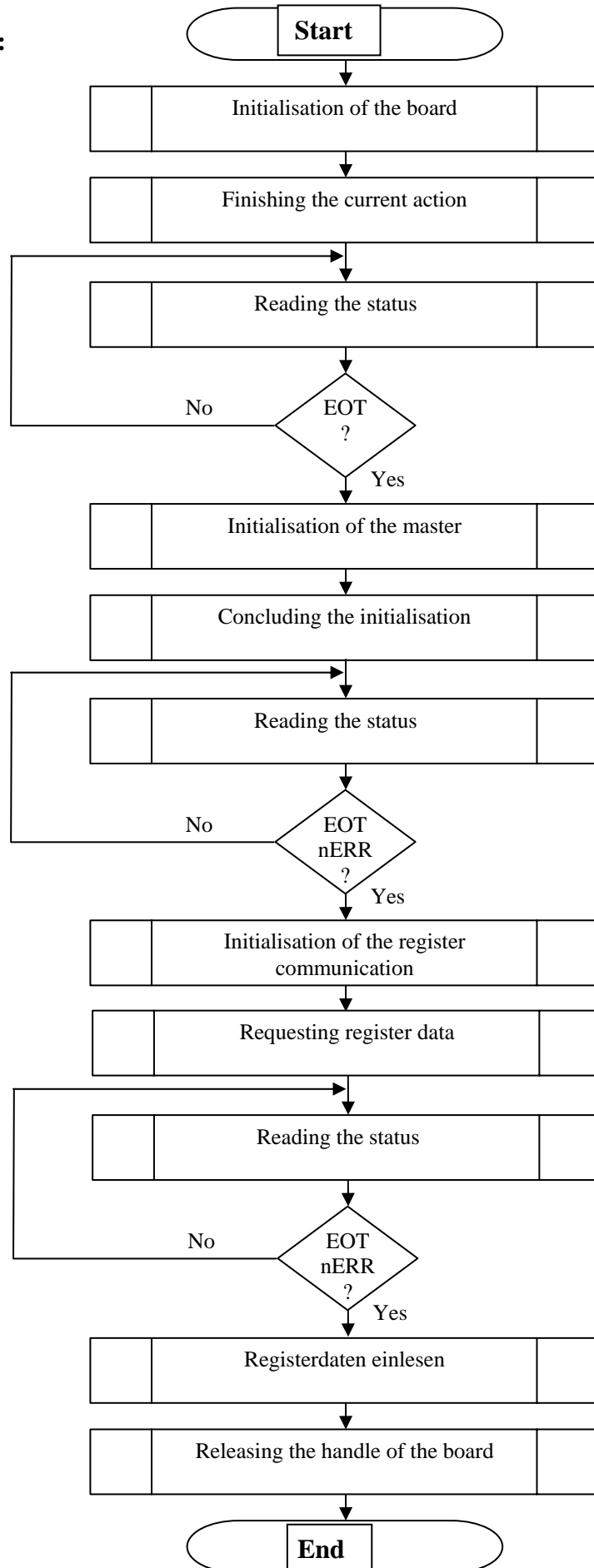
```
i_ReturnValue = i_APCI1710_CloseBoardHandle (b_BoardHandle);
```



## 4.4 Reading register values of a BiSS encoder

In the following example a 8 bit register value is read out. The example refers to the C-sample Sample07 in which the method of programming can be seen in detail.

The used functions are described in detail in the technical description of the board or in the function description of the BiSS-Master.

**Function diagram:**

## Step 1: Initialising the board

Firstly, the board must be initialised. In order to show this more clearly, a function is created, which calls the functions that are provided by ADDI-DATA. In the C-samples this function “initialisation” is shown at the beginning. The following steps are required for the initialisation:

### 1. Selecting the compiler:

With the following function the used compiler is indicated, in this example a C-compiler.

```
i_ReturnValue = i_APCI1710_InitCompiler (DLL_COMPILER_C);
```

### 2. Detecting the slot number:

```
i_ReturnValue = i_APCI1710_CheckAndGetPCISlotNumber  
(b_SlotNumberArray);
```

### 3. Testing the slot number and detecting the handle of the board:

```
i_ReturnValue = i_APCI1710_SetBoardInformation (b_SlotNumberArray [0],  
                                                pb_BoardHandle);
```

## 4.4.1 Step 2: Finishing the current activities

At the beginning an activity of the BiSS-Master that would be possible at the current time is terminated. Hereto the following command is executed:

```
i_ReturnValue = i_APCI1710_BissWriteCommandRegister (  
                                                    b_BoardHandle,  
                                                    b_Module,  
                                                    128);
```

The transmission status of the command must be checked after the execution with the bit EOT of the status register. The check of the bit nERR of the status register is at this time not yet useful, as a former transmission could have set the error message (see Sample07).

## 4.4.2 Step 3: Initialising the Master

In this step the transmission clock and the type of transmission (BiSS or SSI) are set.

```
i_ReturnValue = i_APCI1710_BissInitMaster      (  
                                                    b_BoardHandle,  
                                                    b_Module,  
                                                    b_FrequencyDivision,  
                                                    b_ModeOfOperation);
```

#### 4.4.3 Step 4: Concluding the initialisation

In this step the initialisation is concluded with the respecting command.

```
i_ReturnValue = i_APCI1710_BissWriteCommandRegister (  
                                                    b_BoardHandle,  
                                                    b_Module,  
                                                    16);
```

The transmission status of the command must be checked after the execution with the bit EOT of the status register. Additionally, the bit nERR must be checked in order to be able to react to possible errors (see Sample07).

The conclusion of an initialisation is only required once at the beginning of a sample in order to ensure the readiness of the slave.

#### 4.4.4 Step 5: Initialising the register communication

```
i_ReturnValue = i_APCI1710_BissInitRegisterCommunication (  
                                                    b_BoardHandle,  
                                                    b_Module,  
                                                    b_StartAddress,  
                                                    b_WNR,  
                                                    b_CountOfBytes,  
                                                    b_SlaveID);
```

The value „0“ must be transferred to the parameter „*b\_WNR*“ for a reading access (for a writing access the value „1“).

„*b\_SlaveID*“ must have the value „0“ as to this version of the BiSS-Master only one slave can be connected. If one register shall be read, then „0“ must be transferred to „*b\_CountOfBytes*“. If two registers shall be read, then a „1“ must be transferred. If two registers shall be read, then firstly the register of the transferring start address is read and then the register with the address (start address + 1)

#### 4.4.5 Step 6: Requesting the register data

With the following command a new register value is requested and the transmission between master and slave is started.

```
i_ReturnValue = i_APCI1710_BissWriteCommandRegister (  
                                                    b_BoardHandle,  
                                                    b_Module,  
                                                    8);
```

The transmission status of the command must be checked after the execution with the bit EOT of the status register. Additionally, the bit nERR must be checked in order to be able to react to possible errors (see Sample07).

#### 4.4.6 Step 7: Reading register data

After the successful transmission the register data received by the master can be read.

```
i_ReturnValue = i_APCI1710_BiSSReadRegisterReceiveData (b_BoardHandle,  
                                                         b_Module,  
                                                         pw_ReceiveData  
                                                         a);
```

#### 4.4.7 Step 8: Releasing the handle of the board

Before the user program will be terminated, the handle of the board must be released again with the following function:

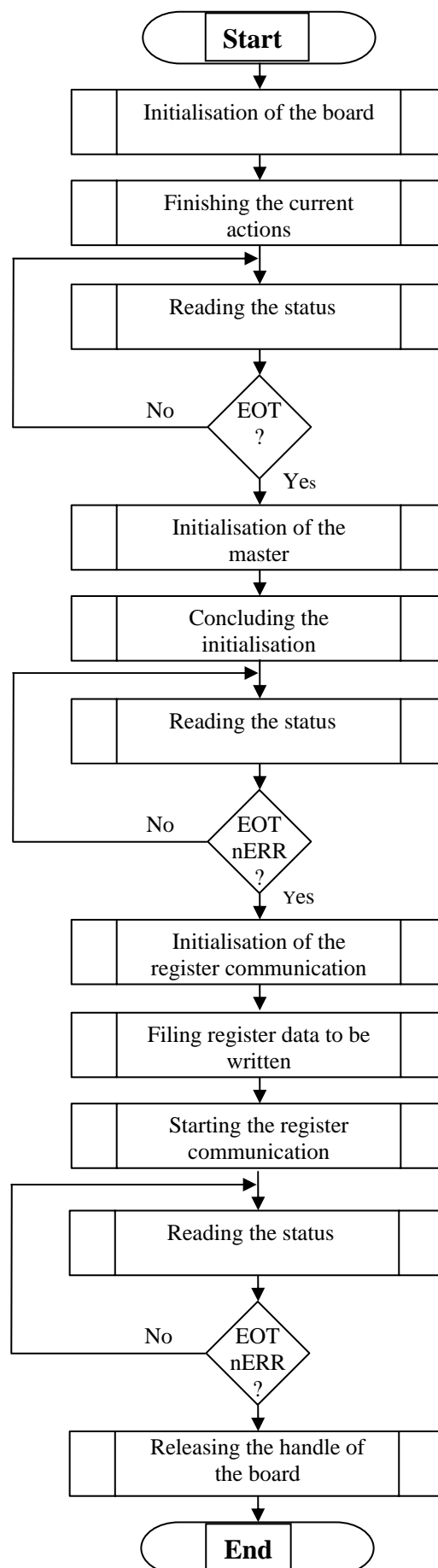
```
i_ReturnValue = i_APCI1710_CloseBoardHandle (b_BoardHandle);
```

## 4.5 Writing register values of a BiSS encoder

In the following example a 8 bit register value is written. The example refers to the C-sample Samle09, in which the type of programming is shown in detail.

The used functions are described in detail in the technical description of the board or in the function description of the BiSS-Master.

## Function diagram:



### 4.5.1 Step 1: Initialisation of the board

Firstly, the board must be initialised. In order to show this more clearly, a function is generated that calls the functions supplied by ADDI-DATA. In the C-samples this function „initialisation“ is shown at the beginning. The following steps are required for the initialisation:

#### 1. Selecting the compiler:

With the following functions the used compiler is indicated, in this example a C-compiler.

```
i_ReturnValue = i_APCI1710_InitCompiler (DLL_COMPILER_C);
```

#### 2. Detecting the slot number:

```
i_ReturnValue = i_APCI1710_CheckAndGetPCISlotNumber  
(b_SlotNumberArray);
```

#### 3. Testing the slot number and detecting the handle of the board:

```
i_ReturnValue = i_APCI1710_SetBoardInformation (b_SlotNumberArray [0],  
                                                pb_BoardHandle);
```

### 4.5.2 Step 2: Finishing the current activities

At the beginning an activity of the BiSS-Master that would be possible at the current time is terminated. Hereto the following command is executed:

```
i_ReturnValue = i_APCI1710_BissWriteCommandRegister (  
                                                        b_BoardHandle,  
                                                        b_Module,  
                                                        128);
```

The transmission status of the command must be checked after the execution with the bit EOT of the status register. The checking of the bit nERR of the status register is not yet useful at this time because an error message could have been set by a former transmission (see Sample09).

### 4.5.3 Step 3: Initialisation of the master

In this step the transmission clock and the type of transmission are set (BiSS or SSI).

```
i_ReturnValue = i_APCI1710_BissInitMaster ( b_BoardHandle,  
                                              b_Module,
```



```
b_FrequencyDivision,  
b_ModeOfOperation);
```

#### 4.5.4 Step 4: Concluding the initialisation

In this step the initialisation is concluded with the respectin command.

```
i_ReturnValue = i_APCI1710_BissWriteCommandRegister (  
    b_BoardHandle,  
    b_Module,  
    16);
```

The transmission status of the command must be checked after the execution with the bit EOT of the status register. Additionally, the bit nERR must be checked in order to be able to react to possible errors (see Sample09).

The conclusion of an initialisation is only necessary once at the beginning of a sample in order to ensure the readiness of the slave.

#### 4.5.5 Step 5: Initialisation of the register communication

```
i_ReturnValue = i_APCI1710_BissInitRegisterCommunication (  
    b_BoardHandle,  
    b_Module,  
    b_StartAddress,  
    b_WNR,  
    b_CountOfBytes,  
    b_SlaveID);
```

The value „1“ must be transferred to the parameter “*b\_WNR*” at the writing access (for reading access the value „0“).

„*b\_SlaveID*“ must have the value „0“, as to this version of the BiSS master only one slave can be connected. If one register shall be written, then for „*b\_CountOfBytes*“ a „0“ must be transferred. If two registers shall be written, then a „1“ must be transferred. If two registers are written, then firstly the register of the transferring start address is written and then the register with the address (start address +1).

#### 4.5.6 Step 6: Filing register data

Before the actual communication between master and slave, the register data to be transferred, must be filed in the master with the following function:

```
i_ReturnValue = i_APCI1710_BissWriteRegisterSendData (b_BoardHandle,  
    b_Module,  
    w_SendData);
```

### 4.5.7 Step 7: Starting the register communication

With the following command the transmission between master and slave is started.

```
i_ReturnValue = i_APCI1710_BissWriteCommandRegister (  
                                                    b_BoardHandle,  
                                                    b_Module,  
                                                    8);
```

The transmission status of the command must be checked after the execution with the bit EOT of the status register. Additionally, the bit nERR must be checked in order to be able to react to possible errors (see Sample09).

### 4.5.8 Step 8: Releasing the handle of the board

Before the user program is finished, the handle of the board must be released again with the following function:

```
i_ReturnValue = i_APCI1710_CloseBoardHandle (b_BoardHandle);
```