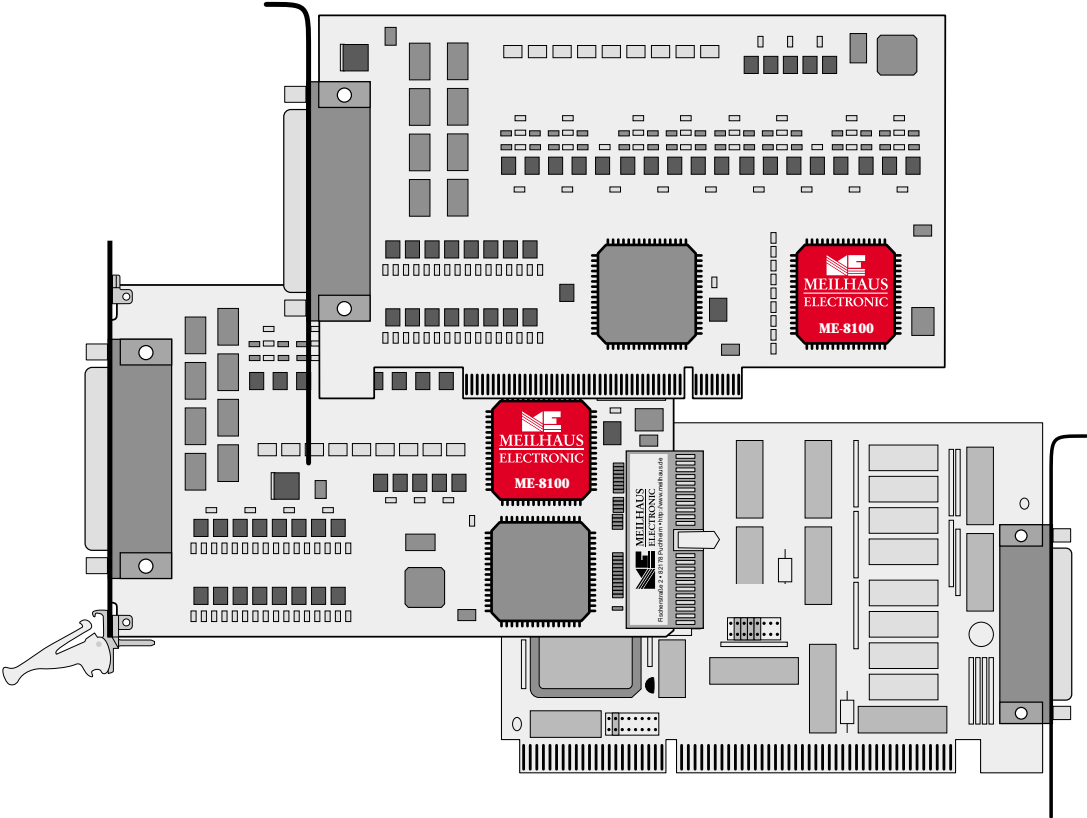


Meilhaus Electronic Manual

ME-81, ME-8100 1.6E ISA, PCI- and CompactPCI-Versions



**Opto isolated I/O-Board with Bit Pattern Comparator
and optional Counter**

Imprint

Manual ME-81, ME-8100

Revision 1.6E

Revised: 8. October 2002

Meilhaus Electronic GmbH
Fischerstraße 2
D-82178 Puchheim/Munich
Germany
<http://www.meilhaus.com>

© Copyright 2002 Meilhaus Electronic GmbH

All rights reserved. No part of this publication may be reproduced or distributed in any form whether photocopied, printed, put on microfilm or be stored in any electronic media without the expressed written consent of Meilhaus Electronic GmbH.

Important note:

The information contained in this manual has been reviewed with great care and is believed to be complete and accurate. Meilhaus Electronic assumes no responsibility for its use, any infringements of patents or other rights of third parties which may result from use of this manual or the product. Meilhaus Electronic assumes no responsibility for any problems or damage which may result from errors or omissions. Specifications and instructions are subject to change without notice.

IBM and IBM PC AT/XT are trademarks of the International Business Machines (IBM) Corporation

Borland Delphi is a trademark of Borland International Inc.

Turbo/Borland C is a trademark of Borland International Inc.

Visual C++ and Visual Basic are trademarks of the Microsoft Corporation.

VEE Pro and VEE OneLab are trademarks of Agilent Technologies.

ME-VEC is a trademark of Meilhaus Electronic GmbH

Other company names and product names found in the text of this manual are also trademarks of the companies involved.



Table of Contents

1	Introduction.....	7
1.1	Package contents.....	7
1.2	Performance Notes.....	8
1.3	System Requirements.....	9
1.4	Important Note for ISA Models	9
1.5	Software Support.....	10
2	Installation.....	11
2.1	Hardware Installation of ISA Models	12
2.1.1	Location of the Jumpers.....	12
2.1.2	Settings of the Jumpers.....	13
2.1.2.1	Base Address.....	13
2.1.2.2	Interrupts	13
2.1.2.3	Default Settings	14
3	Hardware	15
3.1	Block Diagram	15
3.2	General Notes	16
3.3	Operation Modes	16
3.3.1	Digital-I/O	16
3.3.2	Bit Pattern Compare	17
3.3.3	Bit Pattern Change	17
3.3.4	Counter (8254)	17
3.4	Switching	18
3.4.1	Input Switching ME-81.....	18
3.4.2	Output Switching ME-81.....	19
3.4.3	Input Switching ME-8100.....	20
3.4.4	Output Switching ME-8100	21
3.4.5	Counter Switching ME-8100	23
3.4.5.1	Switching Counter Inputs.....	24
3.4.5.2	Switching Counter Outputs.....	24
3.4.5.3	Counter Clock	25
3.4.5.4	Cascading the Counters.....	25

3.5	Register Description.....	26
3.5.1	Registers of ME-81 ISA.....	26
3.5.2	Register of the 82C54.....	28
3.5.2.1	Mode 0: Change state at zero.....	30
3.5.2.2	Mode 1: Retriggerable „One-Shot“.....	30
3.5.2.3	Mode 2: Asymmetric divider.....	31
3.5.2.4	Mode 3: Symmetric divider.....	31
3.5.2.5	Mode 4: Counter start by software trigger.....	32
3.5.2.6	Mode 5: Counter start by hardware trigger.....	32
3.6	Test Program.....	32
4	Programming.....	33
4.1	High Level Language Programming.....	33
4.1.1	Order of Operation.....	33
4.1.1.1	Interrupt on Bit Pattern Match.....	34
4.1.1.2	Interrupt on Bit Pattern Change.....	34
4.1.2	Example Programs.....	35
4.2	Agilent VEE Programming.....	35
4.2.1	User Objects.....	35
4.2.2	Example Programs.....	36
4.2.3	The "ME Board" Menu.....	36
4.3	LabVIEW™ Programming.....	36
4.3.1	Virtual Instruments.....	37
4.3.2	Example Programs.....	37
4.4	Register Programming (ISA versions).....	37
4.4.1	Initialisation.....	38
4.4.2	Simple Input.....	38
4.4.3	Simple Output.....	38
4.4.4	Interrupt by Bit Pattern Match.....	38
4.4.5	Interrupt on Bit Pattern Change.....	39

5	Function Reference	41
5.1	General	41
5.2	Naming Conventions	42
5.3	Description of the API Functions	43
5.3.1	General Functions	45
5.3.2	Digital I/O.....	48
5.3.3	Counter Functions.....	58
5.3.4	Interrupt Handling	60
5.3.5	Error Handling.....	64
Appendix	65
A	Specifications	65
B	Pinout	68
B1	ME-8100A/B PCI and cPCI.....	68
B2	ME-81 ISA	69
C	Accessories	70
D	Technical Questions	71
D1	Hotline	71
D2	Service address	71
D3	Driver Update	71
E	Index	73

1 Introduction

Valued customer,

Thank you for purchasing a Meilhaus data acquisition board. You have chosen an innovative high technology board that left our premises in a fully functional and new condition.

Take the time to carefully examine the contents of the package for any loss or damage that may have occurred during shipping. If there are any items missing or if an item is damaged, contact Meilhaus Electronic immediately.

Before you install the board in your computer, read this manual carefully, especially the chapter describing board installation.

On the ISA bus versions of the boards pay careful attention to the sections describing how to set the jumpers. This will save having to open the computer case again.

1.1 Package contents

We take great care to make sure that the package is complete in every way. We do ask that you take the time to examine the contents of the box. Your box should consist of:

- Electrically isolated Digital-I/O board of the board family ME-81/8100 for ISA-, PCI- or CompactPCI bus.
- Manual in PDF format on CD-ROM (optional as printed version)
- Driver software on CD-ROM
- ME-81 ISA: 37pin D-Sub male connector,
ME-8100 PCI/cPCI: 78pin D-Sub male connector

1.2 Performance Notes

Model Overview

Model	Opto isolated Digital-I/Os	Counter
ME-81 ISA	16 inputs and 16 outputs (24 V)	---
ME-8100A PCI ME-8100A cPCI	16 inputs and 16 outputs (24 V)	3 x 16 bit (24 V)
ME-8100B PCI ME-8100B cPCI	32 inputs and 32 outputs (24 V)	3 x 16 bit (24 V)

Table 1: Model overview ME-81/8100 family

The ME-81/8100 family of boards come with digital input and output ports and additional on the ME-8100 with three 16 bit counters. The digital ports and the counter signals are optically isolated and are designed for control applications requiring 24 V voltage level.

The ME-81 and ME-8100A boards have 16 inputs, 16 outputs, and a 16 bit wide bit pattern comparator. The ME-8100B has 32 inputs, 32 outputs and two bit pattern comparators each 16 bits. The ME-8100 boards have three 16-bit counters available.

The ME-8100 offers the option of switching from “source” drivers to “sink” drivers per software. This guarantees an individual adaption to your needs. First the outputs of all models are in a high impedance state if the PC is off or after power up. That means the voltage level at the output pin depends on your external switching. After a „1“ is set on the output there is current.

As a special feature the ME-81/8100 offers the operation modes „bit pattern compare“ and „bit pattern change“. If the bit pattern on the inputs matches a defined pattern or when a bit is changed from a defined bit, an interrupt occurs. The board can also be used to monitor changing of a level.

The base address for the ISA board version is set by DIP switches and can be set to a wide range of addresses. For the PCI versions of the board the resources are assigned automatically by the BIOS resp. the operating system (Plug&Play).

The software included in the package allows the board to be quickly used in control and measurement applications under Windows (32 bit versions). There are drivers available for Agilent VEE (formerly HP VEE) and LabVIEW™ (National Instruments). For the ISA board versions there are drivers available for DOS and Windows 3.1 on request.

1.3 System Requirements

The boards can be installed into computers with Pentium Prozessor (recommended) or compatibles with a free expansion slot for 8 bit ISA or PCI resp. CompactPCI (depending on the version).

1.4 Important Note for ISA Models

For computers with a PCI bus and a BIOS which supports Plug&Play make sure to reserve the interrupt lines for the ISA bus in the BIOS for any boards requiring the interrupt functions. The BIOS menu will vary depending on the manufacturer (consult the motherboard documentation on how to do this). **If the interrupt lines are not reserved in the BIOS, the interrupt function will not be guaranteed!!**

Also note that on some newer computers the frequency on the ISA bus will be more than the specified 8 MHz. Proper functioning of the boards is not guaranteed if this is the case. Check the BIOS setup of your computer to make sure.

1.5 Software Support

For the newest versions and latest software releases, please consult the README files.

System Drivers

For all common operating systems (see README files)

ME-Software-Developer-Kit (ME-SDK):

Support for all common programming languages, demos, tools und test programs

Graphical programming tools

Meilhaus VEE Driver System for HP VEE V 4.0 or higher, HP VEE Lab, Agilent VEE Pro and Agilent VEE OneLab

Driver System for LabVIEW 4.0 or higher

2 Installation

Please read your computer manual instructions on how to install new hardware components **before installing the board**. Note the chapter „Hardware Installation“ in this manual (if applicable, e. g. for ISA boards).

- **Installation under Windows (Plug&Play)**

An installation guide how to install the driver software can be found in HTML format on CD-ROM. Please read **before installation** and print it on demand!

Basically use the following procedure:

If you have got the driver software as an archive file please unpack the software **before installing the board**. Choose an directory on your computer (e. g. C:\temp).

Now insert the board into your computer and then install the driver software. This order of operation is important to guarantee the Plug&Play operation under Windows 95*/98/Me/2000/XP. Windows 95 and NT 4.0 need an analogous order of operation however the installation procedure differs slightly.

**If the Windows version is supported by the appropriate board type (see readme files).*

- **Installation under Linux**

Note the installation instructions included with archive file of the appropriate driver.

Note: If you want to run the PCI/cPCI versions with application software already written take care of the notes in the proper README file included with the driver software.

If you are using an ISA board please do the jumper settings on the board first (see the following chapters).

2.1 Hardware Installation of ISA Models

☞ Make sure that the computer is turned off.

➤ **Caution:** some of the more sensitive components can be damaged by static electricity!



That's why: Make sure to ground yourself by touching an exposed metal part of the PC case before handling the board.

☞ Unplug the power cable from your computer.

☞ Open the computer case.

2.1.1 Location of the Jumpers

The ISA versions ME-81 require that jumpers be checked/set before the board is installed into the computer. It must also be determined if the settings match the available resources in the computer before installation. **Make sure the computer is turned off before installing the board.**

The locations of the jumpers are shown in the diagrams below. The jumper settings are described in the next chapters.

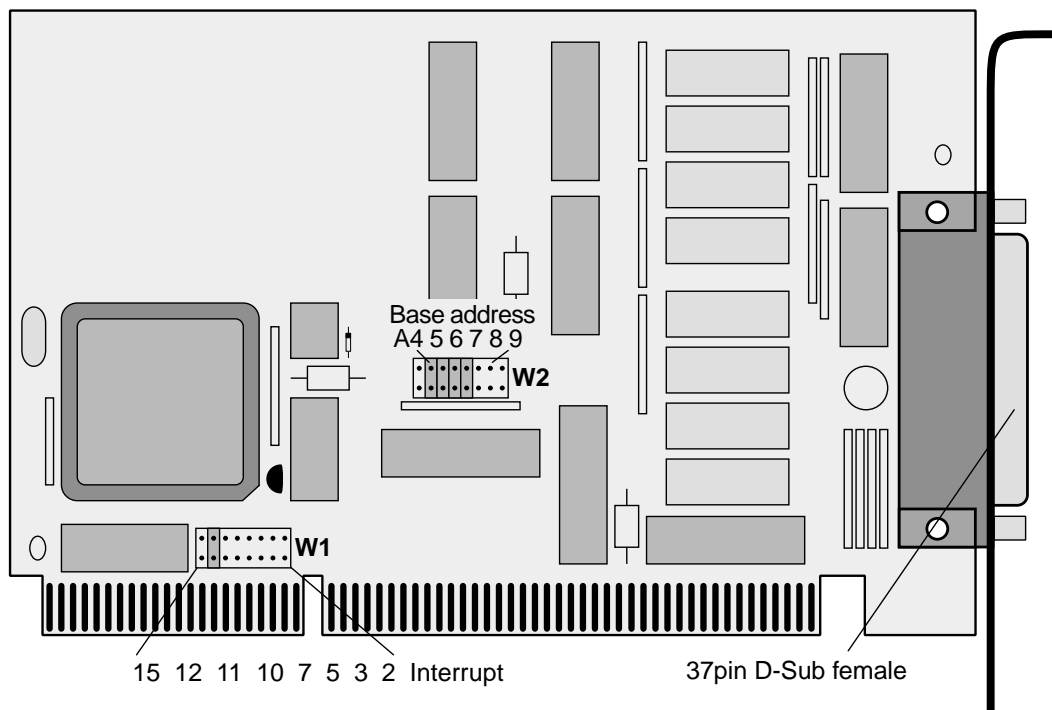


Diagram 1: Picture of the ME-81 ISA

2.1.2 Settings of the Jumpers

2.1.2.1 Base Address

The base address (BA) of the ME-81 is set by the **jumpers W2** in the range of 0Hex to 3F0Hex in 10Hex byte increments. Starting with the base address, the ME-81 occupies 12 bytes and the of I/O address space. Make sure that there are no address conflicts with other boards in the system before installing the board in the computer!

A jumper that is „on“ sets a logic "0" on the address line and a jumper that is „off“ sets a logic "1" on the address line. The base address is determined by summing the jumpers which are „off“. The following example shows the default setting on the board (300Hex):

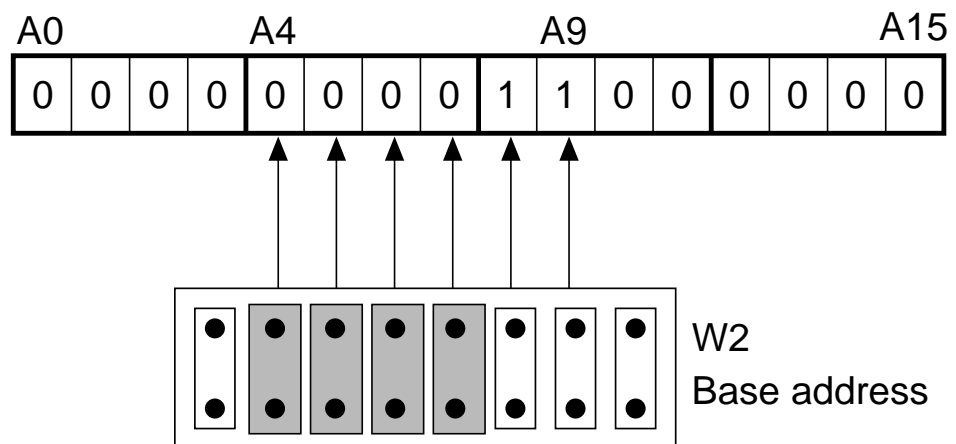


Diagram 2: Setting of base address (Default: 300Hex)

2.1.2.2 Interrupts

For interrupt operation, jumper settings are required. For the ME-81 use **jumper „W1“** to select an interrupt request line (IRQ) between 2, 3, 5, 7, 10, 11, 12 or 15. Make sure that the selected interrupt is not in use with other boards in the system!

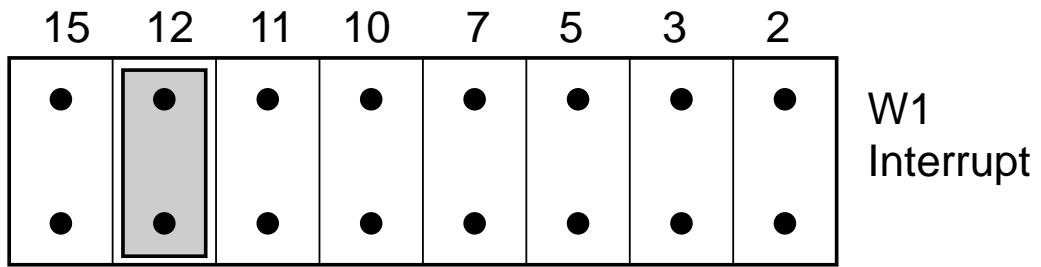


Diagram 3: Interrupt jumper setting

2.1.2.3 Default Settings

Function	Jumper	Settings
Base address	Jumper W2	300Hex
Interrupt	Jumper W1	IRQ 12

Table 2: Default settings of the ME-81 ISA by factory

3 Hardware

3.1 Block Diagram

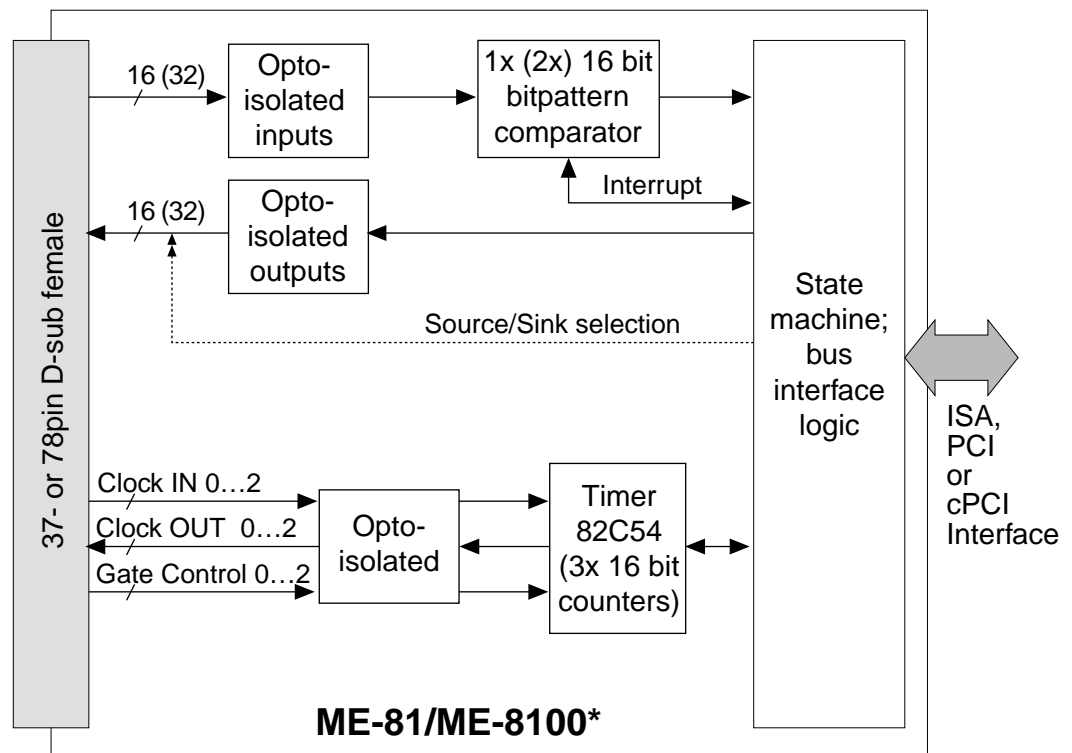


Diagram 4: Block diagram of ME-81/8100

* Depending on the version not all functional groups included in the block diagram above are available:

ME-81: 16 inputs and 16 outputs, 1 x 16 bit wide bitpattern comparator (without counters and source/sink selection).

ME-8100A: 16 inputs and 16 outputs, 1 x 16 bit wide bitpattern comparator, 3 x 16 bit counters.

ME-8100B: 32 inputs and 32 outputs, 2 x 16 bit wide bitpattern comparator, 3 x 16 bit counters.

ISA models: 37pin D-Sub female connector;

PCI-/cPCI models: 78pin D-Sub female connector.

3.2 General Notes

Important Note: The external connections to the board should only be made or removed in a powered down state.

For the pin configuration of the 37pin female D-Sub (ME-81 ISA) resp. 78pin female D-Sub (ME-8100 PCI/cPCI) see „Pinout“ on page 68).

3.3 Operation Modes

The board configuration is done with software by the user. Use the driver software shipped with the board; see chapter 5.3 "Description of the API Functions" (also recommended for ISA versions).

The ME-81/8100 boards can be used in three operational modes for digital-I/O:

3.3.1 Digital-I/O

ME-81/8100A: One 16 bit wide input port (opto isolated) and one 16 bit wide output port (opto isolated).

8100B: Two 16 bit wide input ports (opto isolated) and two 16 bit wide output ports (opto isolated).

The outputs of the ME-8100 models can be set as conductive or as tristate port by port. Additionally the ME-8100 offers the option of switching the outputs from “sink” (low active) to “source” (high active) drivers by software. This guarantees an individual adaption to your needs. First the outputs of all models are in a high impedance state if the PC is off or after power up. That means the voltage level at the output pin depends on your external switching. After a „1“ is set on the output there is current.

As a special feature the ME-81/8100 offers the operation modes „bit pattern compare“ and „bit pattern change“. If the bit pattern on the inputs matches a defined pattern or when a bit is changed from a defined bit, an interrupt occurs. The board can also be used to monitor changing of a level.

3.3.2 Bit Pattern Compare

In the „bit pattern compare“ mode, a bit pattern written to the comparison register is compared to the bit pattern on the corresponding input port(s). An interrupt (if enabled) can be generated when the bit patterns match.

3.3.3 Bit Pattern Change

In the „bit pattern change“ mode, selected input line(s) can be monitored for a change of state. The respective bits of the corresponding mask register serve as a reference. On a transition (0 → 1 or 1 → 0) of at least one, with „1“ masked bit, an interrupt (if enabled) can be generated.

3.3.4. Counter (8254)

This section applies to the ME-8100A and B board versions, **not** however for the ME-81 ISA.

The counter component is the standard 82C54 chip. This flexible component has 3 independent down counters, each 16 bit. The maximum counting frequency of 1 MHz must be provided by an external clock source. After proper switching of the gate input (Enabling by low level) the counter is counting downwards on negative edge of clock. It can be set to count in binary or BCD. Each counter can be configured by software to function independently in the following 6 operational modes:

- Mode 0: Change state at zero
- Mode 1: Retriggerable „One Shot“
- Mode 2: Asymmetric divider
- Mode 3: Symmetric divider
- Mode 4: Counter start by software trigger
- Mode 5: Counter start by hardware trigger

The configuration of the 82C54 is done by the user. For programming, use the driver software provided under Windows 95/98/NT; see chapter 5.3.3 „Counter Functions“ on page 58. The following block diagram shows the function of the component:

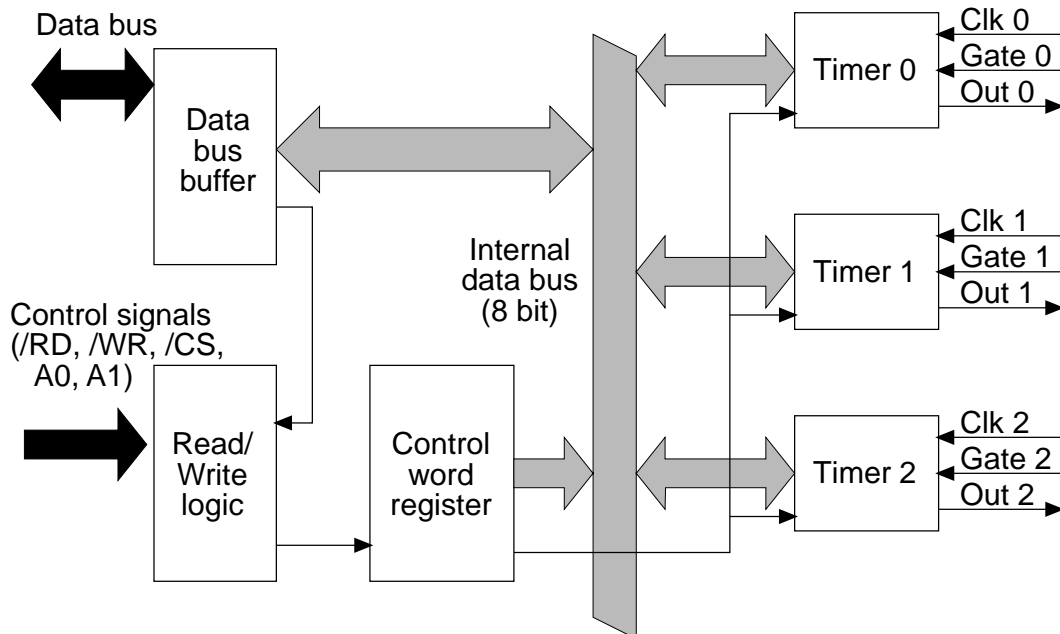


Diagram 5: Block diagram of the 82C54

3.4 Switching

Note, ...

the digital I/O-ports as well as the counter signals are optically isolated and designed for control applications requiring 24 V voltage level.

3.4.1 Input Switching ME-81

The 16 opto-isolated input channels (DI 0 ... 15) of the ME-81 are connected to the opto-couplers through the resistors R_v . These resistors are sized for inputs of 24 V. If required, these resistors can be sized for TTL signal level inputs. The digital lines must be referenced to the digital ground (ext GND DI-section).

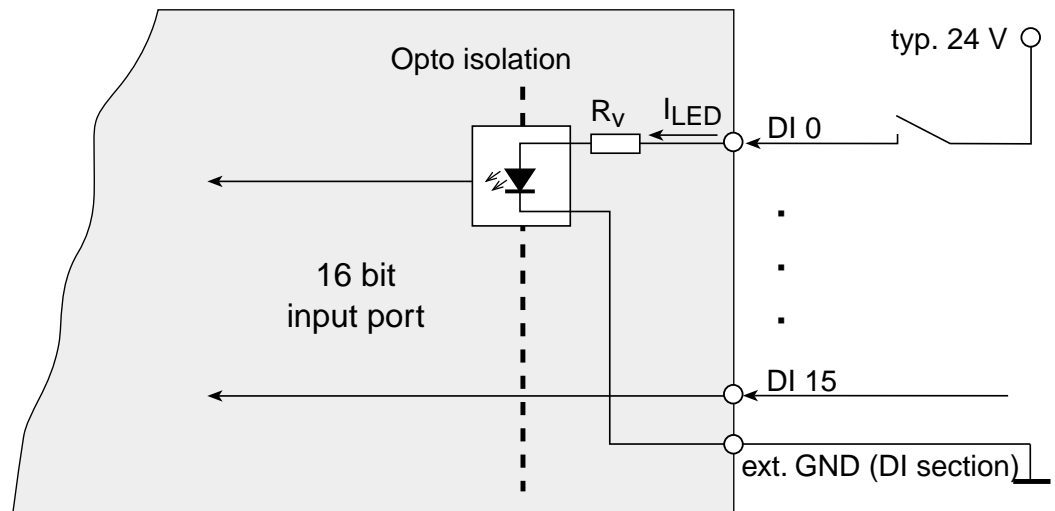


Diagram 6: Input switching of the ME-81

3.4.2 Output Switching ME-81

The 16 opto-isolated output channels (DO 0...15) are realised with 2 driver chips of the type ULN2803. These chips have active low open collector outputs (sink driver). The output levels must be referenced to the ground of the digital output section (ext GND DO-section).

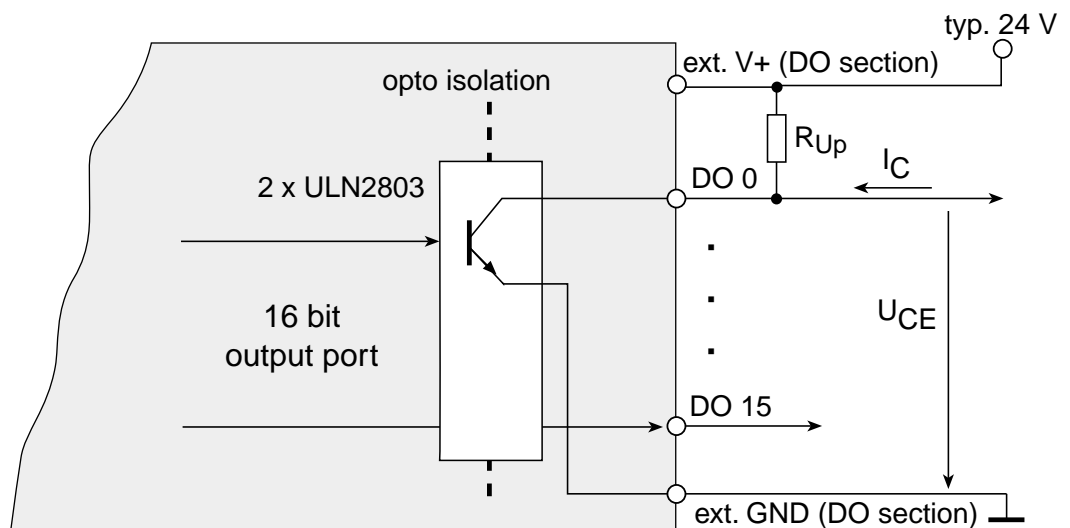


Diagram 7: Output switching of ME-81

The maximum current (I_C) depends on the saturation voltage U_{CE} and is limited by the power dissipation of the sum of the channels to $P_{tot} = 1 \text{ W}$ per chip (DO 0...7 = chip 1, DO 8...15 = chip 2), see Diagram 8: "Saturation voltage ULN2803".

$$P_{\text{tot}} = P_0 + \dots + P_7 \leq 1\text{W (pro Baustein bei } 70^\circ\text{C)}$$

$$\text{wobei } P_0 = I_{\text{C0}} \cdot U_{\text{CE0}}$$

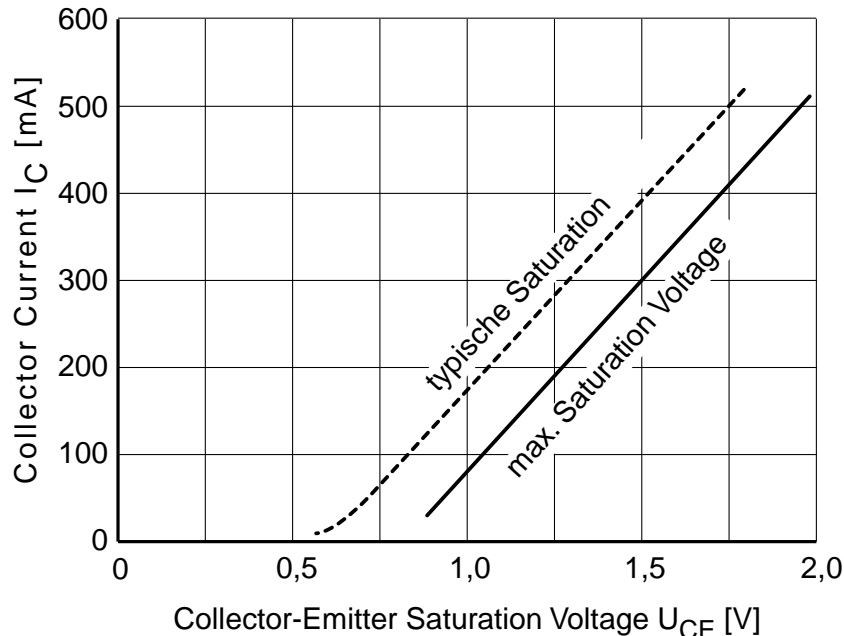


Diagram 8: Saturation voltage ULN2803

3.4.3 Input Switching ME-8100

The 16 resp. 32 opto-isolated input channels (DI_A 0...15/ DI_B 0...15) of the ME-8100A/B are connected to the opto-couplers through resistors R_v . These resistors are sized for inputs of typ. 24 V ($R_v = 2,2\text{k}\Omega$). For over voltage protection of the opto-couplers a protection diode (26 V) was considered. If required, the resistors R_v and the protection diode can be sized for TTL signal level inputs. The digital lines must be referenced to the digital ground (ext. GND).

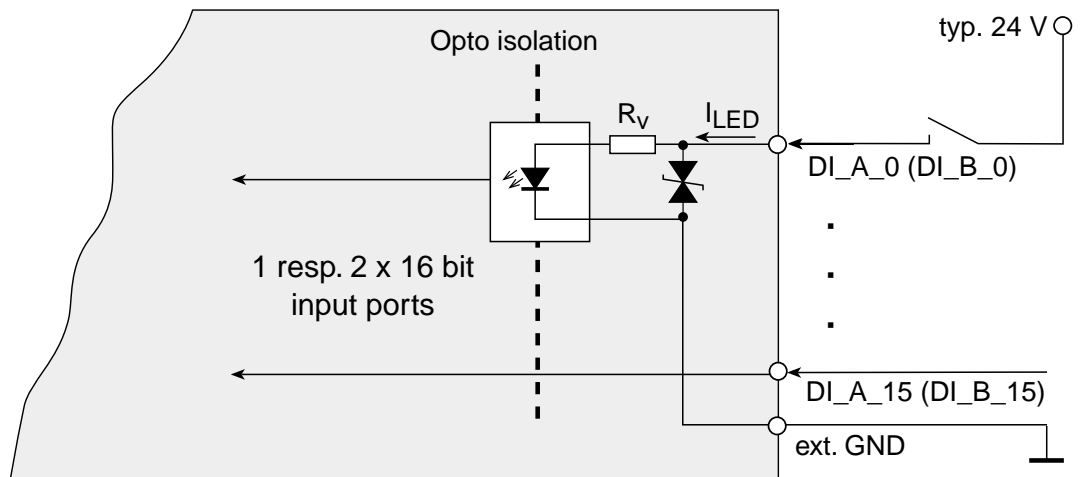


Diagram 9: Input switching of ME-8100

3.4.4 Output Switching ME-8100

The 16 resp. 32 opto-isolated output channels (DO_A 0...15/ DO_B 0...15) of the ME-8100A/B are assembled with 2 resp. 4 output driver chips. Depending on the application, the user can set the outputs as active low (sink driver chip ULN2803; Default) or active high (source driver chip UDN2982). The reference to external ground (ext. GND) must also be made.

Sink Driver:

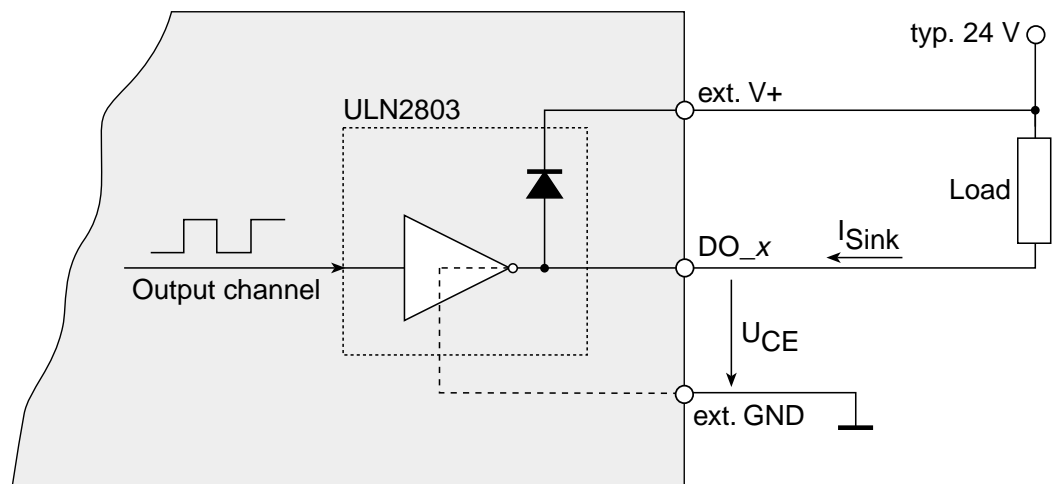


Diagram 10: Outputs of the ME-8100 with sink drivers

The maximum current ($I_C = I_{Sink}$) depends on the saturation voltage U_{CE} and is limited by the power dissipation of the sum of the channels to $P_{tot} = 1\text{ W}$ per chip ($DO_x\ 0\dots7 = \text{chip 1}$, $DO_x\ 8\dots15 = \text{chip 2}$, ...), see Diagram 11: "Saturation voltage ULN2803".

$$P_{tot} = P_0 + \dots + P_7 \leq 1\text{W (per chip at } 70^\circ\text{C)}$$

$$\text{with } P_0 = I_{C0} \cdot U_{CE0}$$

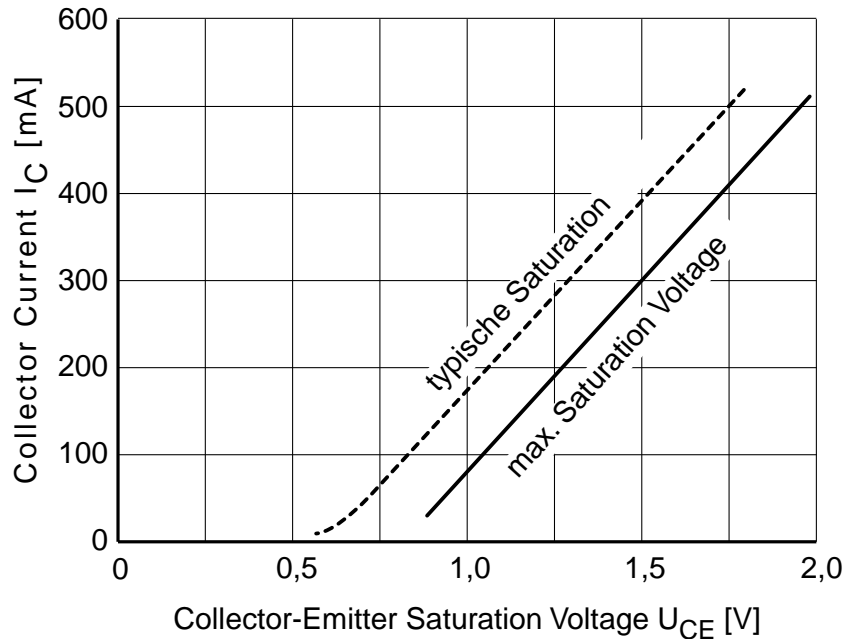


Diagram 11: Saturation voltage ULN2803

Source Driver:

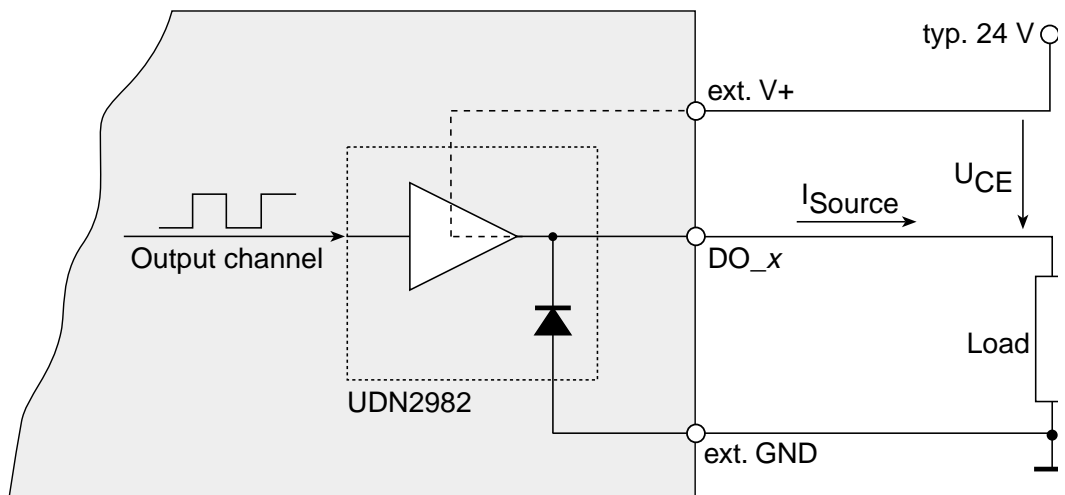


Diagram 12: Outputs of the ME-8100 with source drivers

Please see the following table for the maximum current per output ($I_C = I_{Source}$). The power dissipation of the sum of channels may not exceed $P_{tot} = 0,7 \text{ W}$ per chip (DO_x 0...7 = chip 1, DO_x 8...15 = chip 2, ...).

$$P_{tot} = P_0 + \dots + P_7 \leq 0,7 \text{ W (per chip at } 70^\circ\text{C)}$$

$$\text{with } P_0 = I_{C0} \cdot U_{CE0} \text{ and } U_{CE} = \text{typ. } 1,8 \text{ V}$$

Number of used channels								
	1	2	3	4	5	6	7	8
I_{Cmax} [mA]	350	175	115	85	70	55	50	40

Table 3: Max. current of the source drivers

3.4.5 Counter Switching ME-8100

The “Clk”, “Gate” and “Out” pins on the ME-8100 are electrically isolated up to 1000 V. The counter outputs have pull up resistors ($R_{UP} = 2,2\text{k}\Omega$). All the counter signals are designed for control applications requiring 24 V ($R_V = 2,2\text{k}\Omega$) voltage level.

For enabling the counter the Gate input must be connected to low level. The gate inputs are used as enable or trigger signals, depending on the configuration being used.

3.4.5.1 Switching Counter Inputs

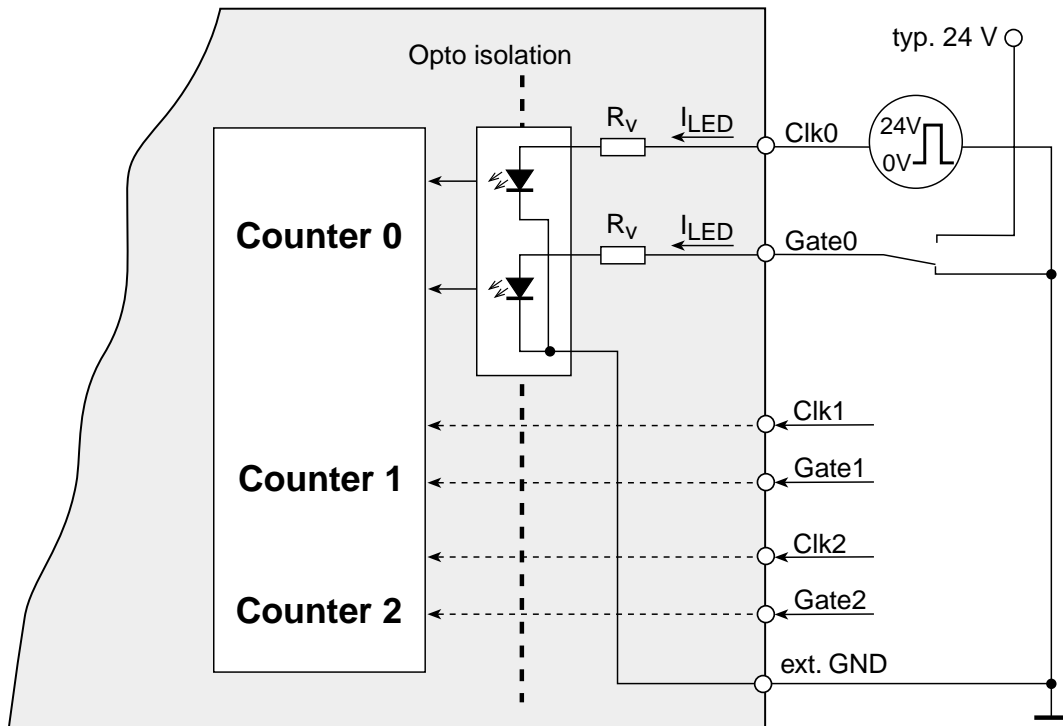


Diagram 13: Switching of counter inputs ME-8100A/B

3.4.5.2 Switching Counter Outputs

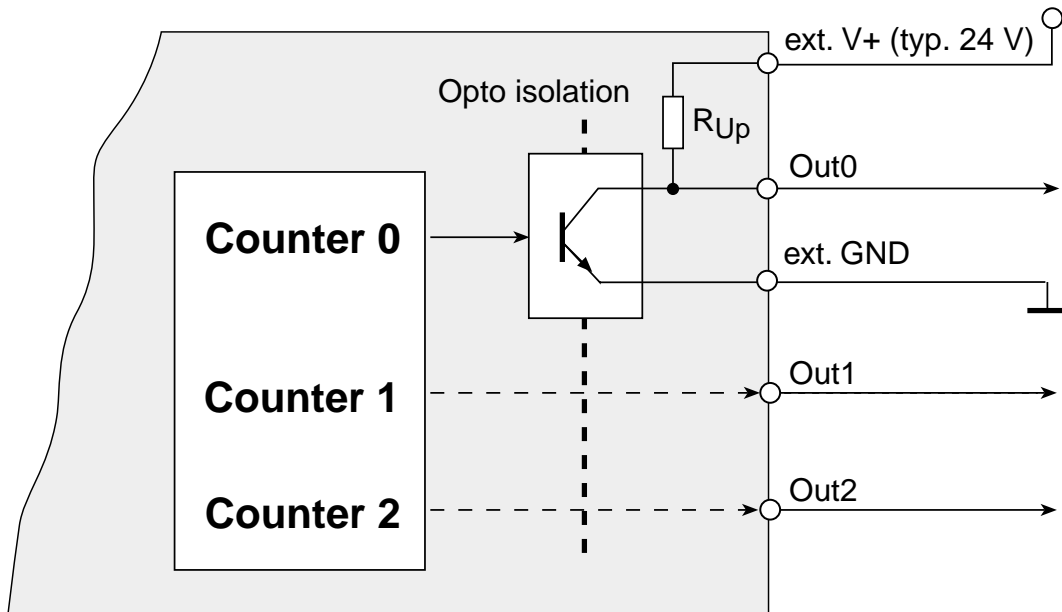


Diagram 14: Switching of counter outputs ME-8100A/B

3.4.5.3 Counter Clock

The input signals Clk 0...2 for the individual counters must be supplied with an external clock signal. The maximum frequency is 1 MHz.

3.4.5.4 Cascading the Counters

The outputs of the counter(s) can be cascaded in line by making the proper external connections. The following example explains how to cascade counters 0...2:

- The clock input of counter 0 (Clk 0) is connected to the source clock
- The output of counter 0 (Out 0) is connected to the clock input of counter 1 (Clk 1)
- The output of counter 1 (Out 1) is connected to the clock input of counter 2 (Clk 2)
- For enabling the counters the gate inputs (Gate 0...2) must be connected to 0 V
- At the output of counter 2 (Out 2) is the cascaded counter signal

3.5 Register Description

3.5.1 Registers of ME-81 ISA

The register set of the ME-81 requires 12 consecutive bytes of I/O address space starting at the base address „BA“. The registers are 16 bits wide and are described in the following tables (R = read, W = write):

Offset	16 Bit Registers	
BA+00H	R	<p>Function ID + INTR-BIT contains infos/ID of board:</p> <p><u>b15...b9</u> reserved</p> <p><u>b8 (INTR)</u> is set if interrupt condition is met (operation modes: bit pattern compare and bit pattern change, INTB0 = 1); but also if interrupt is disabled (e. g. in polling mode, INTB0 = 0)</p> <p><u>b7 (FID7)</u> reserved</p> <p><u>b6 (FID6)</u> function group bit pattern compare</p> <p><u>b5 (FID5)</u> function group AI (A/D-section)</p> <p><u>b4 (FID4)</u> function group AO (D/A-section)</p> <p><u>b3 (FID3)</u> function group DIO (Digital I/O-section)</p> <p><u>b2...b0 (FID2...FID0)</u> PROM version the following Function ID results in PROM version 1: 01001001: 49Hex</p>

Table 4: Address space of the ME-81

Offset		16 Bit Registers
BA+00H	W	<p>Control Register <u>b15...b8</u> reserved <i>Control of digital ports:</i> <u>b7.....ENIO</u> 1 Digital ports enabled 0 Digital ports disabled <i>Selection of interrupt source:</i> <u>b6, b5...INTB1, INTB0</u> 0 0 Interrupt disabled 0 1 Interrupt on bit pattern match 1 0 Interrupt disabled 1 1 Interrupt on bit pattern change <u>b4, b0</u> reserved</p>
BA+02H	R	<p>Reset INTR-Bit By reading from this address the INTR bit is reset. When the INTR bit is set, no new interrupt can occur</p>
BA+04H	R	<p>Input Register The inputs are available after approx. 100 ns</p>
BA+06H	W	<p>Output Register The ENIO = "1" register must be ready to write</p>
BA+08H	W	<p>Compare Register The input bit pattern is compared to the bit pattern in this register. For interrupt handling, the interrupt bits INTB1 and INTB0 in the control register must be previously set.</p>

Table 4: Address space of the ME-81

Offset	16 Bit Registers	
BA+0AH	W	Mask Register This register determines which input bits will be monitored for a change. The bit pattern in the input register is used as the reference. An interrupt occurs when the state of at least one input bit changes. For interrupt handling, the interrupt bits INTB1 and INTB0 in the control register must be previously set.
	R	The state of the input lines is stored in this register after an interrupt occurs. Its value can be read out from this address.

Table 4: Address space of the ME-81

3.5.2 Register of the 82C54

The counters are only available on the models ME-8100A/B. For programming we recommend the use of the software shipped with the board for Windows 95/98 and NT 4.0, which guarantees full functionality.

The control word sets the mode of operation and the counting system (BCD or binary) and controls the loading of the count register.

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

Table 5: Control word of the 82C54

The counter number is selected with bits SC1/0:

SC1	SC0	Function
0	0	Counter 0
0	1	Counter 1
1	0	Counter 2
1	1	not allowed

Table 6: Selection of the counter

The Read/Write mode is selected with bits RL1/0:

RL1	RL0	Function
0	0	Counter latching operation
1	0	only MSB
0	1	only LSB
1	1	first LSB, second MSB (2 x 8 bit)

Table 7: Selection of the read/write mode

The mode of operation for the individual counters is selected with bits M0 .. M2:

M2	M1	M0	Function
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

Table 8: Selection of the operation mode

The counting system for the individual counters is selected with the BCD bit:

BCD	Function
0	16 bit binary counter
1	BCD counter

Table 9: Selection of the counter mode

First, the counter is initialised by writing the control word to the appropriate register. This selects the counter, sets the read/write mode, counter system and mode of operation. The start value is then loaded into the appropriate count register. The counter is decremented by 1 on every falling edge of Clk-signal. The 6 available modes of operation are briefly described next.

3.5.2.1 Mode 0: Change state at zero

This mode of operation can be used e.g. to trigger an interrupt when the counter reaches zero. The counter output (Out 0...2) is set to low when the counter is initialised or when a new start value is loaded. To enable counting, the gate-input must be low. As soon as the start value is loaded and the counter is enabled, the counter is counting downwards and the output remains low.

Upon zero axis crossing, the output goes high and remains high until the counter is reloaded or initialised again. The counter continues to count down, even after zero is reached. If a counter register is loaded during a count in progress the following occurs:

1. when the first byte is written, the count process is stopped
2. when the second byte is written, the count process begins again

3.5.2.2 Mode 1: Retriggerable „One-Shot“

The counter output (Out 0...2) is set high when the counter is initialised. When a start value is loaded the output is set to low on the next trigger pulse (negative edge on the gate-signal). When the zero count is reached, the counter output goes high again.

With a negative edge at the gate-input, the counter can be reset (re-triggered) to the start value. The output remains low until the counter reaches zero.

The counter value can be read at any time without effecting the counting process.

3.5.2.3 Mode 2: Asymmetric divider

In this mode, the counter functions as a frequency divider. The counter output (Out 0...2) is set high after initialisation. When the counter is enabled with a low level at the gate-input, the counter counts down and the output remains high. When the count value reaches 0001Hex, the output goes low for one clock cycle. This process repeats as long as the gate-signal is low. If the gate-signal is not kept low, the output is immediately set to high.

If the counter is reloaded between two output pulses, the current counter state is not affected. The new value is used on the following period.

3.5.2.4 Mode 3: Symmetric divider

This mode of operation is similar to mode 2 with the difference that the divided frequency is symmetric (only for even count values). The counter output (Out 0...2) is set to high after initialisation. When the counter is enabled with a low level at the gate-signal, the counter counts down by 2. The output will cycle (change state), starting from high, every start value/2 periods of the input signal. As long as the gate-input remains low, the process is repeated, otherwise the output immediately is set to high.

If the counter is reloaded between two output pulses, the current counter state is not affected. The new value is used on the following period.

3.5.2.5 Mode 4: Counter start by software trigger

The counter output (Out 0...2) is set high when the counter is initialised. To enable the counter the gate-signal must be low. When the counter is loaded (software trigger) and enabled, the counter starts to count down, while the output remains high.

When zero is reached the output goes low for one clock period and then goes high again. The output remains high until the counter is initialised again and a new start value is loaded.

If the counter is reloaded during a count process, the new start value is used in the next cycle.

3.5.2.6 Mode 5: Counter start by hardware trigger

The counter output (Out 0...2) is set to high when the counter is initialised. When a start value is loaded, the count process will start with the cycle which follows the first trigger pulse (negative edge at the gate-input). When zero is reached, the output is set to low for one clock period. The output then is set to high and remains high until the next trigger pulse.

If the count register is reloaded between trigger pulses, the new start value is used after the next trigger pulse.

The counter can be set back to the start value (re-triggered) at any time by applying a negative edge to the gate-input. The output will remain high until the zero count is reached.

3.6 Test Program

For test issues an test program is provided. The appropriate test program could be found in a corresponding subdirectory of C:\MEILHAUS\ (Default) and can be run by double clicking on the file. (Condition: system driver correctly installed).

4 Programming

The driver concept of the ME-81 resp. the ME-8100 family under Windows (32 bit) offers you the possibility without changing your software using an equivalent PCI board instead of an „old“ ME-81 ISA board. This may be possible without recompilation because the functionality and syntax are identical. (Assumption: your „old“ ISA board and the new PCI board are using the same value in parameter <iBoardNumber>). Please note the appropriate README files on the installation disk of the ME-8100 driver system for the exact order of operation.

Replacing of ISA by PCI boards only works, if you used the function library of the ME-8x Driver System for programming your ISA board!

4.1 High Level Language Programming

The following high level languages are supported by standard:

- Visual C++ (version 4.0 or higher).
- Delphi (version 2.0 or higher).
- Visual Basic (version 4.0 or later).
- For further infos see the appropriate README files on the ME-Power-CD.

Note: The compilers and linkers require the correct paths to be set to the corresponding files in the high level languages.

By linking the high level language specific definition files into your project you can pass many macros and parameters in the form of predefined constants. As an alternative, you can pass the matching Hex value at any time.

4.1.1 Order of Operation

To implement the interrupt functions on the ME-8100 board the following order of operations should be followed:

Note: Interrupt functions are not supported under VisualBASIC.

4.1.1.1 Interrupt on Bit Pattern Match

1. Call *...DIOSetPattern*. The parameter <Pattern> will define the pattern desired. When this pattern appears on the inputs an interrupt will occur.
2. Call *...DIOSetTristateOFF*
3. Call *...SetSinkSourceMode* and set the outputs to source for example (not necessary on ME-81).
4. Call the *...SetIntMode* function. The mode must be set for INTERRUPT_ON_PATTERN_COMPARE.
5. Call *...EnableInt* to enable the interrupt. When the selected pattern appears on the inputs, an interrupt will occur and the service routine will be executed.
6. By calling the function *...GetIrqCnt*, the number of interrupts can be determined (not available on ME-81).
7. Call *...DisableInt*

4.1.1.2 Interrupt on Bit Pattern Change

1. Call *...DIOSetTristateOFF*
2. Call *...SetSinkSourceMode* and set the outputs to source for example (not necessary on ME-81).
3. Call the *...SetIntMode* function. The mode must be set for INTERRUPT_ON_BIT_CHANGE.
4. Call *...DIOSetMask*. The parameter <Mask> will define which bits are to be monitored. When one of these bits changes state, an interrupt will occur. Use FFFFHex to monitor all bits.
5. Call *...EnableInt* to enable the interrupt.
6. By calling the function *...GetIrqCnt*, the number of interrupts can be determined (not available on ME-81).
7. By calling the function *...DIGetIntStatus*, the bit that caused the interrupt can be determined from the value pointed to by the <BitValue> parameter.
8. Call *...DisableInt*

Note: A known pattern will exist on the input lines. This pattern defines the neutral state. An interrupt occurs when a bit changes state. The <Mask> parameter in the ...*DIOSetMask* function can be set to monitor all bits by setting it to FFFFHex. If for example, only selected bits are chosen to be monitored (example: set <Mask> to 000FHex to monitor the 4 lower bits), a bit change on any other bits (the 12 high bits in this example) will **not** cause an interrupt. Only a bit change on the bits set to „1“ in the <Mask> parameter will cause an interrupt.

4.1.2 Example Programs

We have provided simple demo programs and small projects with source code to help understanding of the functions and how to include them into your project. These demo programs can be found within the ME Software Developer Kit (ME-SDK), which is installed to directory C:\Meilhaus\me-sdk by default. Please read the notes in the appropriate README files.

4.2 Agilent VEE Programming

The Agilent VEE components for your board are included with the „ME-Power-CD“ or for download under www.meilhaus.com.

The Meilhaus VEE Driver System supports the HP VEE full versions 4.x and 5.x, HP VEE Lab, Agilent VEE Pro and Agilent VEE OneLab. For installation of VEE components and for further informations please note the documentation included with the VEE driver system. For basics of VEE programming please use your VEE documentation and the VEE online help index.

4.2.1 User Objects

For convenient use of the driver, predefined „User Objects“ have been developed which internally call API functions. They can be called by the additional menu item „ME Board“ and be included in the VEE development environment. They can be placed and „wired“ in your application the same as standard VEE objects.

The User Objects are self descriptive and based on the API functions documented in the chapter „Function Reference“. Ad-

ditionally there are some „Expanded User Objects“ for making programming as easy as possible for you. A short description of every UO is also available under the item „Description“, if you move the cursor over the UO and push the right mouse button.

The UOs can be changed any time for user requirements and can be saved as a user specific object.

4.2.2 Example Programs

For demonstration purposes and for easier understanding, demo programs using the important UOs have been written. They can be called by the menu item „ME Board – Demos“.

The VEE demo programs contain partial additions to the „normal“ UOs and for differentiation from the „normal“ UOs the prefix "x..." in their file name is used.

4.2.3 The "ME Board" Menu

The installation program automatically expands the VEE menu by the „ME Board“ entry. It enables a convenient use of all driver functions available in VEE. From the „ME Board“ menu you can call the driver and demo User Objects sorted by board families.

Note:

The UOs installed, depend on the selected board family at the beginning of your VEE driver installation. If you call UOs under the „ME Board“ menu which are not installed, an error message occurs:

File '*filename*' was not found. Error number: 700

If necessary, you can install the additional VEE components any time (see „ME-Power-CD“).

4.3 LabVIEW™ Programming

The LabVIEW components for your board are included with the „ME-Power-CD“ or for download under www.meilhaus.com.

The Meilhaus LabVIEW drivers support the LabVIEW full versions 4.x or higher. For installation of LabVIEW driver components

please see the documentation delivered with the appropriate LabVIEW driver. For basics of LabVIEW programming please use your LabVIEW documentation and the LabVIEW online help.

4.3.1 Virtual Instruments

For convenient use of the driver, predefined „Virtual Instruments“ have been developed. They can be called by the additional menu item „File - Open“ and be included in the LabVIEW development environment. They can be placed and „wired“ in your application the same as standard LabVIEW objects.

The source VIs are self descriptive and based on the API functions documented in the chapter „Function Reference“. Additionally there are some „Expanded Virtual Instruments“ for making programming as easy as possible for you.

A short description of every VI is also available in the VI „...Function Tree“. This VI is only for documentation purposes and can be opened by the menu „File - Open“. Under „Description“ you find a short description of every Virtual Instrument.

The VIs can be changed any time for user requirements and can be saved as a user specific VI.

4.3.2 Example Programs

For demonstration purposes and for easier understanding, demo programs using the important virtual instruments (VIs) have been written. They can be called by the menu item „File - Open“.

4.4 Register Programming (ISA versions)

Note: The ME-81 ISA can also be programmed at the register level (e. g. in DOS). This can be done with high level language input and output port commands, see „Registers of ME-81 ISA“ on page 26. Consult the manuals for the high level language of your choice for the proper commands and syntax. We recommend, however, to use the driver software, supplied with the board, **else the software compatibility between ISA and PCI/cPCI versions is not given!**

Note the following order of operation:

4.4.1 Initialisation

- Reading the Function ID register (M81_FIDREG). The bits b7...b3 have to contain the value '01001' (binary) and the bits b2...b0 depend of the PROM version. E. g. for PROM version number 1 (001) the following entry is valid (see chapter 3.5 „Register Description“ on page 26):

01001001binary = 49hex.

If not, either the base address is wrong or the board is faulty.

- Stop all operations on the boards: set control register (BA+00hex) to 00Hex.
- Reset INTR bit, by „dummy reading“ of the register BA+02hex.

4.4.2 Simple Input

- Initialisation (see 4.4.1)
- Define the configuration of the control register (BA+00hex)
- Disable interrupt by setting INTB0 = 0
- Reading the input register (BA+04hex)

4.4.3 Simple Output

- Initialisation (see 4.4.1)
- Switching the digital outputs conductive ENIO = 1
- Writing the output register (BA+06hex)

4.4.4 Interrupt by Bit Pattern Match

- Initialisation (see 4.4.1)
- Writing comparison register (BA+08hex)
- Define the configuration of the control register (BA+00hex):

- Selecting the interrupt source:
Bit pattern compare: INTB1, INTB0 = 0, 1
- Interrupt occurs if comparison register matches the input bit pattern. Value which initiated the interrupt is written to the register BA+0AH and can be read in.
- Reset INTR bit, by „dummy reading“ of the register BA+02hex

4.4.5 Interrupt on Bit Pattern Change

- Initialisation (see 4.4.1)
- Reading the neutral value from input register BA+04hex (recommended).
- Define the configuration of the control register BA+00hex:
 - Selecting the interrupt source:
Bit pattern change: INTB1, INTB0 = 1, 1
- Writing bit mask to the mask register BA+0Ahex.
- Interrupt occurs on bit change (0 → 1 or 1 → 0) from at least one input bit relating to the neutral value in the input register. (condition: bit is released in the mask register, i. e. set to "1"). Value which initiated the interrupt is written to the register BA+0AH and can be read in.
- Reset INTR bit, by „dummy reading“ of the register BA+02hex

After every interrupt the INTR bit has to be reset by „dummy reading“ of the register BA+02hex. Else no more interrupt occurs.

5 Function Reference

5.1 General

The driver concept for the ME-81/8100 family intends two independent driver systems for ISA and for PCI/cPCI boards. Because of the driver concept basically ISA boards can be only accessed by the ME-14 driver system and PCI/cPCI boards only by ME-1400 driver system. To get a compatibility of PCI boards and even written application software, additional all functions of the ISA driver are declared in the PCI driver accessing to similar built functions of the the PCI driver. If you want to use this feature, please read the notes for the exact order of operation in the appropriate README files.

The functions of the API-DLL (ME8x_32.DLL) for the ME-81 are supported by the following 32 bit drivers:

- VxD driver (ME8x_32.VXD) for Windows 95/98/Me
- Kernel driver (ME8x_32.SYS) for Windows NT4.0/2000/XP
further on required: dialogue DLL (MEDLG32.DLL).

The functions of the API-DLL (ME8100.DLL) for the ME-8100 are supported by the following 32 bit drivers:

- VxD driver (ME8100.VXD) for Windows 95
- Kernel driver (ME8100.SYS) for Windows NT4.0
- WDM driver (ME8100.SYS) for Windows 98/Me/2000/XP

After the driver is successfully loaded, the API functions allow convenient access to the hardware. Every function that accesses a ME-81/8100 board requires an integer value for identification of the board. In the following description of the functions this parameter is referred to as <iBoardNumber>. Specifying the proper board keep the following in mind:

- Value range ISA versions: 0...3
- Value range PCI/cPCI versions: 0...31

- Value for the 1st board: 0
- Value for the 2nd board: 1
- Value for nth board : n-1

5.2 Naming Conventions

These functions were written specifically for the ME-81/8100 board family. For Visual C and Delphi (Pascal) every API function starts with an underscore „_“ (not so in Borland C and BASIC). The function names were selected to be as descriptive as possible. Each function name consists of a board type specific prefix and several elements which stand for the corresponding sections (e. g. "DIO" for "Digital In/Out").

_me81... Function only valid for ME-81
_me8x... Function only valid for ME-80 and ME-80
me8100... Function only valid for ME-8100A/B

For the description of the functions, the following standards will be used:

<i>function name</i>	will be italic in body text e. g. <i>me8100GetBoardVersion</i>
<parameters>	will be in brackets as shown and in font Courier
<variables>	will indicate a predefined constant and will be written in italic text and in brackets as shown
[square brackets]	variables in square brackets have to be used depending on board type.
FILE NAMES	or PATHS will be capitalized in font Courier
me8100... ()	parts of programs will be in Courier type

To identify data types, the following letters will be used:

i... or dw...	32 bit integer value
s... or w...	16 bit short value
c... or b...	8 bit character value
p...	pointer of data type (i, s, l or c)

5.3 Description of the API Functions

The functions will be described by functional groups as listed below. Within each functional group, the individual functions will be described in alphabetical order:

„5.3.1 General Functions“ on page 45

„5.3.2 Digital I/O“ on page 48

„5.3.3 Counter Functions“ on page 58

„5.3.4 Interrupt Handling“ on page 60

„5.3.5 Error Handling“ on page 64

Function	Short Description	Page
General Functions		
me8100GetBoardVersion	Determine board version	45
_me8xGetDLLVersion me8100GetDLLVersion	Determine DLL version number	45
_me8xPROMVersion me8100PROMVersion	Determine PROM-ID of board	46
me8100SetSinkSourceMode	Switching sink/source driver	47
Digital I/O		
_me8xDIGetBit me8100DIGetBit	Read bit	48
_me81DIGetIntStatus me8100DIGetIntStatus	Read bit pattern that initiated interrupt	49
_me8xDIGetWord me8100DIGetWord	Read word (16 bit)	50
_me8xDIOSetIntMode me8100DIOSetIntMode	Select interrupt event	51
_me81DIOSetMask me8100DIOSetMask	Write bit mask	52
_me8xDIOSetPattern me8100DIOSetPattern	Write bit pattern for comparison	53
_me8xDIOSetTristateOFF me8100DIOSetTristateOFF	Activate output port	54

Table 10: Overview of library functions

Function	Short Description	Page
_me8xDIOSetTristateON me8100DIOSetTristateON	Deactivate output port	55
_me8xDOSetBit me8100DOSetBit	Set bit	56
_me8xDOSetWord me8100DOSetWord	Write word (16 bit)	57
Counter Functions		
me8100CntRead	Read current counter state	58
me8100CntWrite	Configuration of counter and loading of start value	59
Interrupt Handling		
_me8xDisableInt me8100DisableInt	Disable interrupt control	60
_me8xEnableInt me8100EnableInt	Enable interrupt control	61
me8100GetIrqCnt	Determine number of interrupts	62
Error Handling		
me8100GetDrvErrMess	Error string corresponding to error code	64

Table 10: Overview of library functions

Note: In case of a function is valid for ISA boards (prefix *_me81* or *_me8x*) and for PCI boards (prefix *me8100*) the common prefix *_me8xxx* will be used in the following function description. For the valid prefixes look at the headline of each function. Most of the functions for the ME-8100 have been expanded by the parameter `<iRegisterSet>`. By this way you can access easily to the identical B-part of the ME-8100 (parameter not allowed for ISA versions).

5.3.1 General Functions

me8100GetBoardVersion

Description

Function is used for: ME-8100A/B.

Determines the board version number of an installed board of the board family ME-8100.

Definitions

C: int me8100GetBoardVersion (int iBoardNumber, int *piDevices)

Delphi: Function me8100GetBoardVersion (iBoardNumber: integer; Var iDevices: integer): integer;

Basic: Declare Function me8100GetBoardVersion Lib "me8100" Alias "_VBme8100GetBoardVersion@8" (ByVal iBoardNumber As Long, iDevices As Long) As Long

Parameters

<BoardNumber> Number of board to be accessed, ME-81 (0...3) resp. ME-8100 (0...31)

<Version> Pointer to an integer value where the board version is returned. Possible values:

810AHex:	ME-8100A
810BHex:	ME-8100B

Return value

If the function is successfully executed, a '1' is returned. If an error occurs, a '0' is returned. The cause of the error can be determined with the function *_me8xxxGetDrvErrMsg*.

_me8xGetDLLVersion me8100GetDLLVersion

Description

Function is used for: ME-81, ME-8100A/B.

Determines the version number of the driver DLL.

● Definitions

C: int _me8xxxGetDLLVersion();
 Delphi: Function _me8xxxGetDLLVersion: integer;
 Basic: Declare Function me8xxxGetDLLVersion Lib
 "me8xxx_32" Alias "_VBme8xxxGetDLLVersion@0" () As
 Long

→ Parameters none

< Return value

The version number is returned as a 32 bit value. The upper 16 bits contain the main version number and the lower 16 bits contain the sub version number. E. g.: 00020001Hex indicates the version number 2.01

_me8xPROMVersion **me8100PROMVersion**

🔪 Description

Function is used for: ME-81
 Returns the board identification and the PROM-ID.

● Definitions

C: int _me8xxxPROMVersion (int iBoardNumber, int
 *piVersion;)
 Delphi: Function _me8xxxPROMVersion (iBoardNumber: integer;
 Var iVersion: integer): integer;
 Basic: Declare Function me8xxxPROMVersion Lib "me8xxx_32"
 Alias "_VBme8xxxPROMVersion@8" (ByVal
 iBoardNumber As Long, ByRef iVersion As Long) As Long

→ Parameters

<BoardNumber> Number of board to be accessed,
 ME-81 (0...3) resp. ME-8100 (0...31)

<Version> Pointer to an integer value which contains the PROM-ID and the board version. The value is interpreted as a hexadecimal value:

<u><Version></u>	<u>Karte</u>
00810049Hex	ME-81
00008100Hex	ME-8100A/B

◀ Return value

If the function is successfully executed, a '1' is returned. If an error occurs, a '0' is returned. The cause of the error can be determined with the function *_me8xxxGetDrvErrMess*.

me8100SetSinkSourceMode

🔪 Description

Function is used for: ME-8100A/B.

This function activates either the sink or source driver of the outputs port by port (see „Output Switching ME-8100“ on page 21). After driver start the sink driver is selected. Moreover the function *me8100DIOSetTristateOFF* must be called once for each port after driver start.

● Definitions

C: int me8100SetSinkSourceMode (int iBoardNumber, int iRegisterSet, int iMode)

Delphi: Function me8100SetSinkSourceMode (iBoardNumber: integer; iRegisterSet: integer; iMode: integer): integer;

Basic: Declare Function me8100SetSinkSourceMode Lib "me8100" Alias "_VBme8100SetSinkSourceMode@12" (ByVal iBoardNumber As Long, ByVal iRegisterSet As Long, ByVal iMode As Long) As Long

➔ Parameters

<BoardNumber> Number of board to be accessed, ME-81 (0...3) resp. ME-8100 (0...31)

<RegisterSet> Selection of register set (for ME-8100A always REGISTER_SET_A must be passed):

<u><A/B-Selection></u>	<u>Description</u>
REGISTER_SET_A (00Hex)	register set for part A
REGISTER_SET_B (01Hex)	register set for part B

<Mode>	Selection of output driver devices:	
	<u><Sink/Source></u>	<u>Description</u>
	SINK_MODE (00Hex)	Sink driver active
	SOURCE_MODE (01Hex)	Source driver active

◀ Return value

If the function is successfully executed, a '1' is returned. If an error occurs, a '0' is returned. The cause of the error can be determined with the function `_me8xxxGetDrvErrMess`.

5.3.2 Digital I/O

`_me8xDIGetBit` `me8100DIGetBit`

🔪 Description

Function is used for: ME-81, ME-8100A/B
Returns the status of a single input line.

● Definitions

C: `int _me8xxxDIGetBit (int iBoardNumber, [int iRegisterSet,] int iBitNo, int *piBitValue);`

Delphi: `Function _me8xxxDIGetBit (iBoardNumber, [iRegisterSet: integer;] iBitNo: integer; Var iBitValue: integer): integer;`

Basic: `Declare Function me8xxxDIGetBit Lib "me8xxx_32" Alias "_VBme8xxxDIGetBit@12" (ByVal iBoardNumber As Long, [ByVal iRegisterSet As Long,] ByVal iBitNo As Long, iBitValue As Long) As Long`

➔ Parameters

<BoardNumber> Number of board to be accessed,
ME-81 (0...3) resp. ME-8100 (0...31)

[RegisterSet] (parameter not in `_me8xDIGetBit`)
Selection of register set (for ME-8100A always
REGISTER_SET_A must be passed):

<u><A/B-Selection></u>	<u>Description</u>
REGISTER_SET_A (00Hex)	register set for part A
REGISTER_SET_B (01Hex)	register set for part B

<BitNo>	The number the input line whose status is to be read in. Possible values are: 0...15: 16 bit input port of ME-81 resp. ME-8100
<BitValue>	Pointer to an integer value representing the status of the selected input line: 0: input line is set to low level 1: input line is set to high level

◀ Return value

If the function is successfully executed, a '1' is returned. If an error occurs, a '0' is returned. The cause of the error can be determined with the function *_me8xxxGetDrvErrMess*.

_me81DIGetIntStatus **me8100DIGetIntStatus**

🔪 Description

Function is used for: ME-81, ME-8100A/B

This function returns the input word (16 bit) of the last interrupt.

This function is not for use with Agilent VEE!

● Definitions

C: int *_me8xxxDIGetIntStatus* (int iBoardNumber, [int iRegisterSet,] int *iValue);

Delphi: Function *_me8xxxDIGetIntStatus* (iBoardNumber: integer; [iRegisterSet: integer;] Var iValue: integer): integer;

Basic: Declare Function *me8xxxDIGetIntStatus* Lib "me8xxx_32" Alias "_VBme8xxxDIGetIntStatus@8" (ByVal iBoardNumber As Long, [ByVal iRegisterSet As Long,] iValue As Long) As Long

➔ Parameters

<BoardNumber> Number of board to be accessed, ME-81 (0...3) resp. ME-8100 (0...31)

[RegisterSet] (parameter not in *_me8xDIGetIntStatus*)

Selection of register set (for ME-8100A always REGISTER_SET_A must be passed):

<u><A/B-Selection></u>	<u>Description</u>
------------------------------	--------------------

REGISTER_SET_A (00Hex)	register set for part A
REGISTER_SET_B (01Hex)	register set for part B

<Value> Pointer to an integer value; only the lower 16 bits are significant.

◀ Return value

If the function is successfully executed, a '1' is returned. If an error occurs, a '0' is returned. The cause of the error can be determined with the function `_me8xxxGetDrvErrMess`.

`_me8xDIGetWord` **`me8100DIGetWord`**

🔪 Description

Function is used for: ME-81, ME-8100A/B.

This function reads a word from a 16 bit port on the board.

● Definitions

C: `int _me8xxxDIGetWord (int iBoardNumber, [int iRegisterSet,] int iPortNo, int *iValue);`

Delphi: `Function _me8xxxDIGetWord (iBoardNumber: integer; [iRegisterSet: integer;] [iPortNo: integer;] Var iValue: integer): integer;`

Basic: `Declare Function me8xxxDIGetWord Lib "me8xxx_32" Alias "_VBme8xxxDIGetWord@12" (ByVal iBoardNumber As Long, [ByVal iRegisterSet As Long,] [ByVal iPortNo As Long,] iValue As Long) As Long`

➔ Parameters

<BoardNumber> Number of board to be accessed,
ME-81 (0...3) resp. ME-8100 (0...31)

[RegisterSet] (parameter not in `_me8xDIGetWord`)
Selection of register set (for ME-8100A always
REGISTER_SET_A must be passed):

<u><A/B-Selection></u>	<u>Description</u>
REGISTER_SET_A (00Hex)	register set for part A
REGISTER_SET_B (01Hex)	register set for part B

[PortNo] (Parameter not in `me8100DIGetWord`)

PORTA (00Hex) must be passed for the 16 bit input port of the ME-81

<Value> Pointer to an integer value representing the read in value, only the lower 16 bits significant.

◀ Return value

If the function is successfully executed, a '1' is returned. If an error occurs, a '0' is returned. The cause of the error can be determined with the function `_me8xxxGetDrvErrMess`.

_me8xDIOSetIntMode **me8100DIOSetIntMode**

🔪 Description

Function is used for: ME-81, ME-8100A/B

This function sets the interrupt mode of the board.

This function is not for use with Agilent VEE!

● Definitions

C: `int _me8xxxDIOSetIntMode (int iBoardNumber, [int iRegisterSet,] int iMode);`

Delphi: `Function _me8xxxDIOSetIntMode (iBoardNumber: integer; [iRegisterSet: integer;] iMode: integer): integer;`

Basic: `Declare Function me8xxxDIOSetIntMode Lib "me8xxx_32" Alias "_VBme8xxxDIOSetIntMode@8" (ByVal iBoardNumber As Long, [ByVal iRegisterSet As Long,] ByVal iMode As Long) As Long`

➔ Parameters

<BoardNumber> Number of board to be accessed,
ME-81 (0...3) resp. ME-8100 (0...31)

[RegisterSet] (parameter not in `_me8xDIOSetIntMode`)
Selection of register set (for ME-8100A always REGISTER_SET_A must be passed):

<u><A/B-Selection></u>	<u>Description</u>
REGISTER_SET_A (00Hex)	register set for part A
REGISTER_SET_B (01Hex)	register set for part B

<Mode> Interrupt modes;
possible values for the **ME-81** are:

<u><Modes ME-81></u>	<u>Description</u>
NO_INTERRUPT (00Hex)	no interrupt operation
INT_IF_PATTERN (20Hex)	interrupt on bit pattern match

INT_IF_MASK (60Hex)
 interrupt on bit pattern change
 possible values for the **ME-8100** are:

<Modes ME-8100>	Description
INTERRUPT_ON_PATTERN_COMPARE (00Hex)	interrupt on bit pattern match
INTERRUPT_ON_BIT_CHANGE (01Hex)	interrupt on bit pattern change

< Return value

If the function is successfully executed, a '1' is returned. If an error occurs, a '0' is returned. The cause of the error can be determined with the function `_me8xxxGetDrvErrMess`.

_me81DIOSetMask **me8100DIOSetMask**

🔪 Description

Function is used for: ME-81, ME-8100A/B.

This function writes a 16 bit value to the board which is linked as a mask to the digital inputs. If an input which has been masked to a logic „1“ changes its state, an interrupt is initiated (if enabled). For order of operation see 4.1.1.2 "Interrupt on Bit Pattern Change".

● Definitions

C: `int _me8xxxDIOSetMask (int iBoardNumber, [int iRegisterSet,] int iMask);`
 Delphi: `Function _me8xxxDIOSetMask (iBoardNumber: integer; [iRegisterSet: integer;] iMask: integer): integer;`
 Basic: `Declare Function me8xxxDIOSetMask Lib "me8xxx_32" Alias "_VBme8xxxDIOSetMask@8" (ByVal iBoardNumber As Long, [ByVal iRegisterSet As Long,] ByVal iMask As Long) As Long`

➔ Parameters

<BoardNumber> Number of board to be accessed,
 ME-81 (0...3) resp. ME-8100 (0...31)

[RegisterSet] (parameter not in *_me8xDIOSetMask*)

Selection of register set (for ME-8100A always REGISTER_SET_A must be passed):

<u><A/B-Selection></u>	<u>Description</u>
REGISTER_SET_A (00Hex)	register set for part A
REGISTER_SET_B (01Hex)	register set for part B
<Mask>	Mask value; the assignment to the inputs is bit by bit; possible values are: 0...65535 (0000Hex...FFFFHex)

◀ Return value

If the function is successfully executed, a '1' is returned. If an error occurs, a '0' is returned. The cause of the error can be determined with the function *_me8xxxGetDrvErrMess*.

_me8xDIOSetPattern **me8100DIOSetPattern**

🔪 Description

Function is used for: ME-81, ME-8100A/B

This function writes a 16 bit value as the comparison pattern to the corresponding register. If the pattern match those of the input lines, an interrupt is initiated (if enabled). For order of operation see 4.1.1.1 "Interrupt on Bit Pattern Match".

● Definitions

C: int *_me8xxxDIOSetPattern* (int iBoardNumber, [int iRegisterSet,] int iPattern);

Delphi: Function *_me8xxxDIOSetPattern* (iBoardNumber: integer; [iRegisterSet: integer;] iPattern: integer): integer;

Basic: Declare Function *me8xxxDIOSetPattern* Lib "me8xxx_32" Alias "_VBme8xxxDIOSetPattern@8" (ByVal iBoardNumber As Long, [ByVal iRegisterSet As Long,] ByVal iPattern As Long) As Long

➔ Parameters

<BoardNumber> Number of board to be accessed,
ME-81 (0...3) resp. ME-8100 (0...31)

[RegisterSet] (parameter not in *_me8xDIOSetPattern*)
 Selection of register set (for ME-8100A always REGISTER_SET_A must be passed):

<u><A/B-Selection></u>	<u>Description</u>
REGISTER_SET_A (00Hex)	register set for part A
REGISTER_SET_B (01Hex)	register set for part B

<Pattern> Comparison pattern (16 bit wide):
 Input lines 0...15 correspond to bits 0...15 of the comparison register

← Return value

If the function is successfully executed, a '1' is returned. If an error occurs, a '0' is returned. The cause of the error can be determined with the function *_me8xxxGetDrvErrMess*.

_me8xDIOSetTristateOFF ***me8100DIOSetTristateOFF***

✍ Description

Function is used for: ME-81, ME-8100A/B.

This function activates the digital outputs of the board. Values previously written into the registers are switched through to the port.

☞ Important Note:

After starting the driver, the function must be successfully completed before each output function call.

● Definitions

C: int *_me8xxxDIOSetTristateOFF* (int iBoardNumber, [int iRegisterSet]);

Delphi: Function *_me8xxxDIOSetTristateOFF* (iBoardNumber: integer; iRegisterSet: integer): integer;

Basic: Declare Function *me8xxxDIOSetTristateOFF* Lib "me8xxx_32" Alias "_VBme8xxxDIOSetTristateOFF@4" (ByVal iBoardNumber As Long[, ByVal iRegisterSet As Long]) As Long

→ Parameters

<BoardNumber> Number of board to be accessed,
 ME-81 (0...3) resp. ME-8100 (0...31)

[RegisterSet] (parameter not in *_me8xDIOSetTristateOFF*)

Selection of register set (for ME-8100A always REGISTER_SET_A must be passed):

<u><A/B-Selection></u>	<u>Description</u>
REGISTER_SET_A (00Hex)	register set for part A
REGISTER_SET_B (01Hex)	register set for part B

◀ Return value

If the function is successfully executed, a '1' is returned. If an error occurs, a '0' is returned. The cause of the error can be determined with the function *_me8xxxGetDrvErrMess*.

_me8xDIOSetTristateON **me8100DIOSetTristateON**

🔪 Description

Function is used for: ME-80, ME-81, ME-8100A/B.

This function deactivates the digital outputs on the board (tristate). When the driver is loaded, the outputs are set to the state described above.

● Definitions

C: int *_me8xxxDIOSetTristateON*(int iBoardNumber, [int iRegisterSet]);

Delphi: Function *_me8xxxDIOSetTristateON*(iBoardNumber: integer; iRegisterSet: integer): integer;

Basic: Declare Function *me8xxxDIOSetTristateON* Lib "me8xxx_32" Alias "_VBme8xxxDIOSetTristateON@4" (ByVal iBoardNumber As Long, [ByVal iRegisterSet As Long]) As Long

➔ Parameters

<BoardNumber> Number of board to be accessed,
ME-81 (0...3) resp. ME-8100 (0...31)

[RegisterSet] (parameter not in *_me8xDIOSetTristateON*)

Selection of register set (for ME-8100A always REGISTER_SET_A must be passed):

<u><A/B-Selection></u>	<u>Description</u>
REGISTER_SET_A (00Hex)	register set for part A
REGISTER_SET_B (01Hex)	register set for part B

◀ Return value

If the function is successfully executed, a '1' is returned. If an error occurs, a '0' is returned. The cause of the error can be determined with the function *_me8xxxGetDrvErrMess*.

_me8xDOSetBit **me8100DOSetBit**

🔪 Description

Function is used for: ME-81, ME-8100A/B

This function sets an output line to the chosen state.

👉 Important Note:

In order to activate the outputs after starting the driver, the function *_me8xxxDIOSetTristateOFF* must be successfully completed before this function can be called.

● Definitions

C: int *_me8xxxDOSetBit* (int iBoardNumber, [int iRegisterSet,] int iBitNo, int iBitValue);

Delphi: function *_me8xxxDOSetBit* (iBoardNumber: integer; [iRegisterSet: integer;] iBitNo, iBitValue: integer): integer;

Basic: Declare Function *me8xxxDOSetBit* Lib "me8xxx_32"
 Alias "_VBme8xxxDOSetBit@12" (ByVal iBoardNumber As Long, [ByVal iRegisterSet As Long,] ByVal iBitNo As Long, ByVal iBitValue As Long) As Long

➔ Parameters

<BoardNumber> Number of board to be accessed,
ME-81 (0...3) resp. ME-8100 (0...31)

[RegisterSet] (parameter not in *_me8xDOSetBit*)
 Selection of register set (for ME-8100A always REGISTER_SET_A must be passed):

<u><A/B-Selection></u>	<u>Description</u>
REGISTER_SET_A (00Hex)	register set for part A
REGISTER_SET_B (01Hex)	register set for part B

<BitNo> Number of the output line whose status is to be set; possible values are:
 0...15: 16 bit output port of ME-81
 resp. ME-8100

<BitValue> Possible values are:
 0: output line is set off-state
 1: output line is set on-state

◀ Return value

If the function is successfully executed, a '1' is returned. If an error occurs, a '0' is returned. The cause of the error can be determined with the function *_me8xxxGetDrvErrMess*.

_me8xDOSetWord ***me8100DOSetWord***

🔪 Description

Function is used for: ME-81, ME-8100A/B
 This function writes a word to a 16 bit output port.

👉 Important Note:

In order to activate the outputs after starting the driver, the function *_me8xxxDIOSetTristateOFF* must be successfully completed before this function can be called.

● Definitions

C: int *_me8xxxDOSetWord* (int iBoardNumber, [int iRegisterSet,] int iPortNo, int iValue);

Delphi: Function *_me8xxxDOSetWord* (iBoardNumber: integer; [iRegisterSet: integer;] [iPortNo: integer;] iValue: integer): integer;

Basic: Declare Function `me8xxxDOSetWord` Lib "me8xxx_32"
 Alias "_VBme8xxxDOSetWord@12" (ByVal
 iBoardNumber As Long, [ByVal iRegisterSet As Long,]
 [ByVal iPortNo As Long,] ByVal iValue As Long) As Long

→ Parameters

<BoardNumber> Number of board to be accessed,
 ME-81 (0...3) resp. ME-8100 (0...31)

[RegisterSet] (parameter not in *_me8xDOSetWord*)
 Selection of register set (for ME-8100A always
 REGISTER_SET_A must be passed):

<u><A/B-Selection></u>	<u>Description</u>
REGISTER_SET_A (00Hex)	register set for part A
REGISTER_SET_B (01Hex)	register set for part B

[PortNo] (Parameter not in *me8100DOSetWord*)
 PORTA (00Hex) must be passed for the 16 bit out-
 put port of the ME-81

<Value> Output value, possible values are:
 0000Hex...FFFFHex (dezimal: 0...65535)

← Return value

If the function is successfully executed, a '1' is returned. If an error occurs, a '0' is returned. The cause of the error can be determined with the function *_me8xxxGetDrvErrMess*.

5.3.3 Counter Functions

me8100CntRead

🔪 Description

Function is used for: ME-8100A/B.
 Upon calling this function, the actual timer value is buffered and the value is read in.

● Definitions

C: `int me8100CntRead(int iBoardNumber, int iCounter, int *piValue);`

Delphi: `Function me8100CntRead(iBoardNumber, iCounter: integer; Var iValue: integer): integer;`

Basic: Declare Function me8100CntRead Lib "me8100" Alias
 "_VBme8100CntRead@12" (ByVal iBoardNumber As
 Long, ByVal iCounter As Long, iValue As Long) As Long

→ Parameters

<BoardNumber> Number of board to be accessed,
 ME-81 (0...3) resp. ME-8100 (0...31)

<CounterNo> Defines the counter whose counter value is to be
 read in, possible values are:
 0, 1, 2: Counter 0...2 of ME-8100A/B

<Value> 16 bit value from specified counter

← Return value

If the function is successfully executed, a '1' is returned. If an error
 occurs, a '0' is returned. The cause of the error can be determined
 with the function *_me8xxxGetDrvErrMess*.

me8100CntWrite

🔪 Description

Function is used for: ME-8100A/B.

This function configures a counter with a mode of operation and
 loads a 16 bit start value. The counting function starts automatically
 upon calling this function (condition: Gate input enabled by 0 V).

● Definitions

C: int me8100CntWrite (int iBoardNumber, int iCounter, int
 iMode, int iValue);

Delphi: Function me8100CntWrite(iBoardNumber, iCounter,
 iMode, iValue: integer): integer;

Basic: Declare Function me8100CntWrite Lib "me8100" Alias
 "_VBme8100CntWrite@16" (ByVal iBoardNumber As
 Long, ByVal iCounter As Long, ByVal iMode As Long,
 ByVal iValue As Long) As Long

→ Parameters

<BoardNumber> Number of board to be accessed,
 ME-81 (0...3) resp. ME-8100 (0...31)

<CounterNo> Defines the counter that is to be configured. Pos-
 sible values are:
 0, 1, 2: Counter 0...2 of ME-8100A/B

<Mode>	Operation mode of counter, possible values are: <u><Mode></u> <u>Counter Operation Mode</u>
00Hex	Mode 0, "Change of state if zero axis crossing"
01Hex	Mode 1, "Retriggerable One-Shot"
02Hex	Mode 2, "Asymmetrical divider"
03Hex	Mode 3, "Symmetrical divider"
04Hex	Mode 4, "Counter start by software trigger"
05Hex	Mode 5, "Counter start by hardware trigger"
<Value>	16 bit load value for the specified counter; possible values are: 0...65535 (0000Hex...FFFFHex)

◀ Return value

If the function is successfully executed, a '1' is returned. If an error occurs, a '0' is returned. The cause of the error can be determined with the function `_me8xxxGetDrvErrMess`.

5.3.4 Interrupt Handling

`_me8xDisableInt` `me8100DisableInt`

✍ Description

Function is used for: ME-81, ME-8100.

The interrupt control on the board previously started by `_me8xxxEnableInt` is disabled.

👉 Important Note!

In Agilent VEE, use this function only for the ME-8100 in connection with `me8100GetIrqCnt!`

● Definitions

C:	<code>int _me8xxxDisableInt (int iBoardNumber, [int iRegisterSet]);</code>
Delphi:	<code>Function _me8xxxDisableInt (iBoardNumber: integer; [iRegisterSet: integer;]):integer;</code>
Basic:	not realized

➔ Parameters

<BoardNumber> Number of board to be accessed,
ME-81 (0...3) resp. ME-8100 (0...31)

[RegisterSet] (parameter not in *_me8xDisableInt*)
Selection of register set (for ME-8100A always
REGISTER_SET_A must be passed):

<u><A/B-Selection></u>	<u>Description</u>
REGISTER_SET_A (00Hex)	register set for part A
REGISTER_SET_B (01Hex)	register set for part B

◀ Return value

If the function is successfully executed, a '1' is returned. If an error occurs, a '0' is returned. The cause of the error can be determined with the function *_me8xxxGetDrvErrMess*.

_me8xEnableInt **me8100EnableInt**

🔪 Description

Function is used for: ME-81, ME-8100A/B.

The interrupt control is enabled. When an interrupt occurs an user defined callback routine is processed. If this function is used with *me8100GetIrqCnt*, a null pointer is passed instead of the callback routine.

👉 Important Note!

In Agilent VEE, use this function only for the ME-8100 in connection with *me8100GetIrqCnt*!

● Definitions

C: `int _me8xxxEnableInt (int iBoardNumber, [int iRegisterSet,] [ME8100_]PSERVICE_PROC IrqFunc[, int iContext]);`

Delphi: `Function _me8xxxEnableInt (iBoardNumber: integer; [iRegisterSet: integer;] IrqFunc: Pointer; iContext: integer): integer;`

Basic: not realized

➔ Parameters

<BoardNumber> Number of board to be accessed,
ME-81 (0...3) resp. ME-8100 (0...31)

[RegisterSet] (parameter not in *_me8xEnableInt*)
 Selection of register set (for ME-8100A always REGISTER_SET_A must be passed):

<u><A/B-Selection></u>	<u>Description</u>
REGISTER_SET_A (00Hex)	register set for part A
REGISTER_SET_B (01Hex)	register set for part B

<IrqFunc> Address of a user defined callback routine, processed if interrupt occurs.
 For ME-81 of type:
 (void SERVICE_PROC (void))
 and for ME-8100 of type:
 (void ME8100_PSERVICE_PROC (void))
 resp. of type pointer in Delphi.

[Context] (Parameter not in *_me8xEnableInt*)
 For the ME-8100 you can pass an integer value in this parameter to identify the interrupt source.
 E. g. for the second installed ME-8100 with <BoardNumber> = „1“ and register set B pass the value „4“.

◀ Return value

If the function is successfully executed, a '1' is returned. If an error occurs, a '0' is returned. The cause of the error can be determined with the function *_me8xxxGetDrvErrMess*.

me8100GetIrqCnt

✍ Description

Function is used for: ME-8100A/B.

This function acquires the number of interrupts since the device is running. Use this function to provide interrupt control for graphic programming languages like Agilent VEE or LabView™. As usual the interrupt control must be enabled resp. disabled by the functions *...EnableInt* and *...DisableInt*. By reading the value of the parameter *piCnt* it is possible to determine whether an interrupt was initiated or not, relative to a previous request.

● Definitions

C: int me8100GetIrqCnt (int iBoardNumber, int iRegisterSet, int* piCnt);

Delphi: Function me8100GetIrqCnt (iBoardNumber: integer; iRegisterSet: integer; Var iIrqCnt: integer): integer;
 Basic: Declare Function me8100GetIrqCnt Lib "me8100" Alias "_VBme8100GetIrqCnt@8" (ByVal iBoardNumber As Long, ByVal iRegisterSet As Long, ByRef iIrqCnt As Long) As Long

→ Parameters

<BoardNumber> Number of board to be accessed, ME-81 (0...3) resp. ME-8100 (0...31)
 [RegisterSet] Selection of register set (for ME-8100A always REGISTER_SET_A must be passed):

<u><A/B-Selection></u>	<u>Description</u>
REGISTER_SET_A (00Hex)	register set for part A
REGISTER_SET_B (01Hex)	register set for part B

<IrqCnt> Number of interrupts since the device is running

☞ Example

```
iErrorCode = me8100SetSinkSourceMode(0, REGISTER_SET_A, SINK_MODE);
iErrorCode = me8100EnableInt(0, REGISTER_SET_A, 0, 0);
iErrorCode = me8100GetIrqCnt(0, REGISTER_SET_A, &iIrqCntBefore);

Sleep(1000);           //waiting for interrupts

iErrorCode = me8100GetIrqCnt(0, REGISTER_SET_A, &iIrqCntAfter);
iErrorCode = me8100DisableInt(0, REGISTER_SET_A);
IrqCnt = (iIrqCntAfter-iIrqCntBefore);
```

← Return value

If the function is successfully executed, a '1' is returned. If an error occurs, a '0' is returned. The cause of the error can be determined with the function *_me8xxxGetDrvErrMess*.

5.3.5 Error Handling

_me8xGetDrvErrMess
me8100GetDrvErrMess

✍ Description

Function is used for: ME-81, ME-8100A/B

If an error occurs during the processing of the previously called API function of the driver, this routine returns the matching error code and text.

☞ Important Note!

This function may only be called, if the previously called API function of the ME8x_32.DLL or ME8100.DLL returned an error code (i. e. error code 0)!

● Definitions

C: int _me8xxxGetDrvErrMess (char *pcErrortext[, int iBufferSize]);

Delphi: Function _me8xxxGetDrvErrMess (Var errortext: errorstring; iBufferSize: integer): integer;

Basic: Declare Function me8xxxGetDrvErrMess Lib "me8xxx_32" Alias "_VBme8xxxGetDrvErrMess@4" (ByVal errortext As String[, ByVal iBufferSize As Long]) As Long

➔ Parameters

<Errortext> Pointer to a string; the return value is the error code.

[BufferSize] (Parameter not in *_me8xGetDrvErrMess*)
 Buffer size in number of characters for <Errortext> will be allocated (recommended: max. 128 characters).

< Return value

The function always returns the DLL global variable <DLLErrorCode>

Appendix

A Specifications

PC Interface

Bus system
(depends on model)

ISA bus (16 bit);
Standard PCI (32 Bit, 33 MHz);
CompactPCI (32 bit, 33 MHz)

Addressing

ISA
000Hex...3F0Hex in 10Hex steps (Jumper)

PCI, cPCI:

Automatic assignment of base address
under Windows (32 bit)

Interrupts

ISA:
IRQ2, 3, 5, 7, 10, 11, 12, 15 (Jumper)

PCI, cPCI:

Automatic assignment of interrupt channel
under Windows (32 bit)

Interrupt events

Bit pattern match or bit change of a
masked input bit

Digital input

Number

ME-81, ME-8100A: 1 x 16 bit port,
opto isolated

Switching frequency

ME-8100B: 2 x 16 bit ports, opto isolated
max. 1 kHz (depends on operating system
and application software)

Input level

typ. 24 V \pm 2 V

Input current

10 mA per channel

Operation modes

Simple input;
Bit pattern match
Bit pattern change

Digital output

Number

ME-81, ME-8100A: 1 x 16 bit port,
opto isolated

Switching frequency

ME-8100B: 2 x 16 bit ports, opto isolated
max. 1 kHz (depends on operating system
and application software)

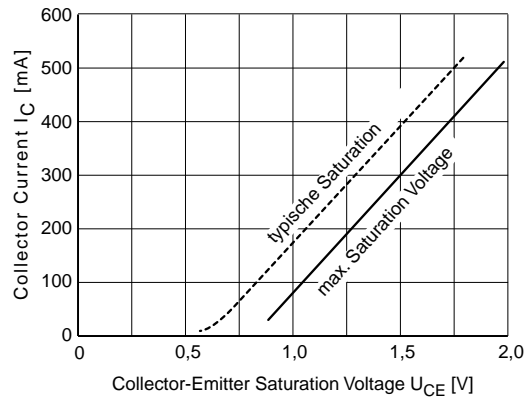
Output level

typ. 24 V (depends on ext. power supply)

Output driver ME-81: sink driver (ULN2803)
 ME-8100A/B: sink (ULN2803) or source (UDN2982) driver selectable by software port by port

Output current The max. current per output (I_C) depends on the saturation voltage U_{CE} and is limited by the power dissipation of the sum of channels to $P_{tot} = 1$ W per chip:
 $P_{tot} = P_0 + \dots + P_7 \leq 1W$ (at 70°C)

ULN2803:



UDN2982:

Please see the table below for the max. current per output (I_{Source}). The power dissipation of the sum of channels is limited to $P_{tot} = 0,7$ W per chip:
 $P_{tot} = P_0 + \dots + P_7 \leq 0,7$ W (at 70°C)
 with $P_0 = I_{C0} \cdot U_{CE0}$ and $U_{CE} = \text{typ. } 1,8$ V

Number of channels								
	1	2	3	4	5	6	7	8
I_{Source} [mA]	350	175	115	85	70	55	50	40

Counter/Timer

Number ME-8100A/B: 3 independent
 Type 82C54
 Resolution 16 bit
 Clock signal (Clk) opto isolated, input voltage typ. 24 V
 Gate signal (Gate) opto isolated, low-active, input voltage typ. 24 V

Counter output (Out)	opto isolated, output voltage typ. 24 V (depends on ext. power supply)
Counter clock	external up to max. 1 MHz

General Information

Power consumption at +5 V	ME-81: typ. 0,8 A (without ext. load) ME-8100A/B: typ. 1,3 A (without ext. load)
Physical size (without mounting bracket and connector)	ME-81 ISA: 100 mm x 180 mm ME-8100 PCI: 174 mm x 98 mm ME-8100 cPCI: CompactPCI-Karte mit 3 HE
Physical size	100 mm x 180 mm (ME-81 ISA) 174 mm x 98 mm (PCI models) 100 mm x 160 mm (cPCI models)
Connectors	ME-81 ISA: 37pin D-Sub female ME-8100 PCI and cPCI: 78pin D-Sub female
Operating temperature	0...70°C
Storage temperature	0...50°C
Relative humidity	20...55% (non condensing)

CE Certification

EMC Directive	89/336/EMC
Emission	EN 55022
Noise immunity	EN 50082-2

B Pinout

B1 ME-8100A/B PCI and cPCI

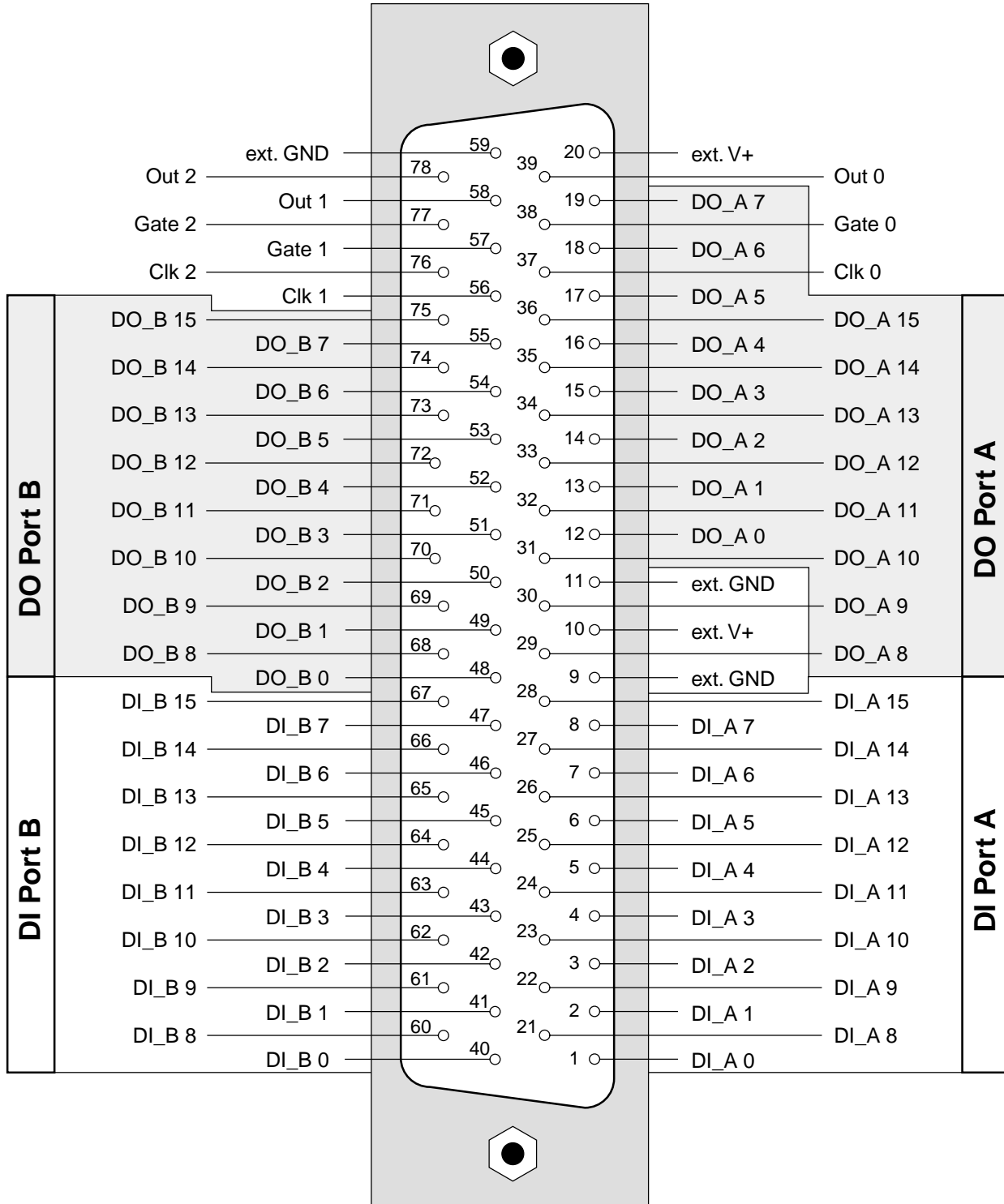


Diagram 15: Pinout of the 78pin female D-Sub on ME-8100A/B

B2 ME-81 ISA

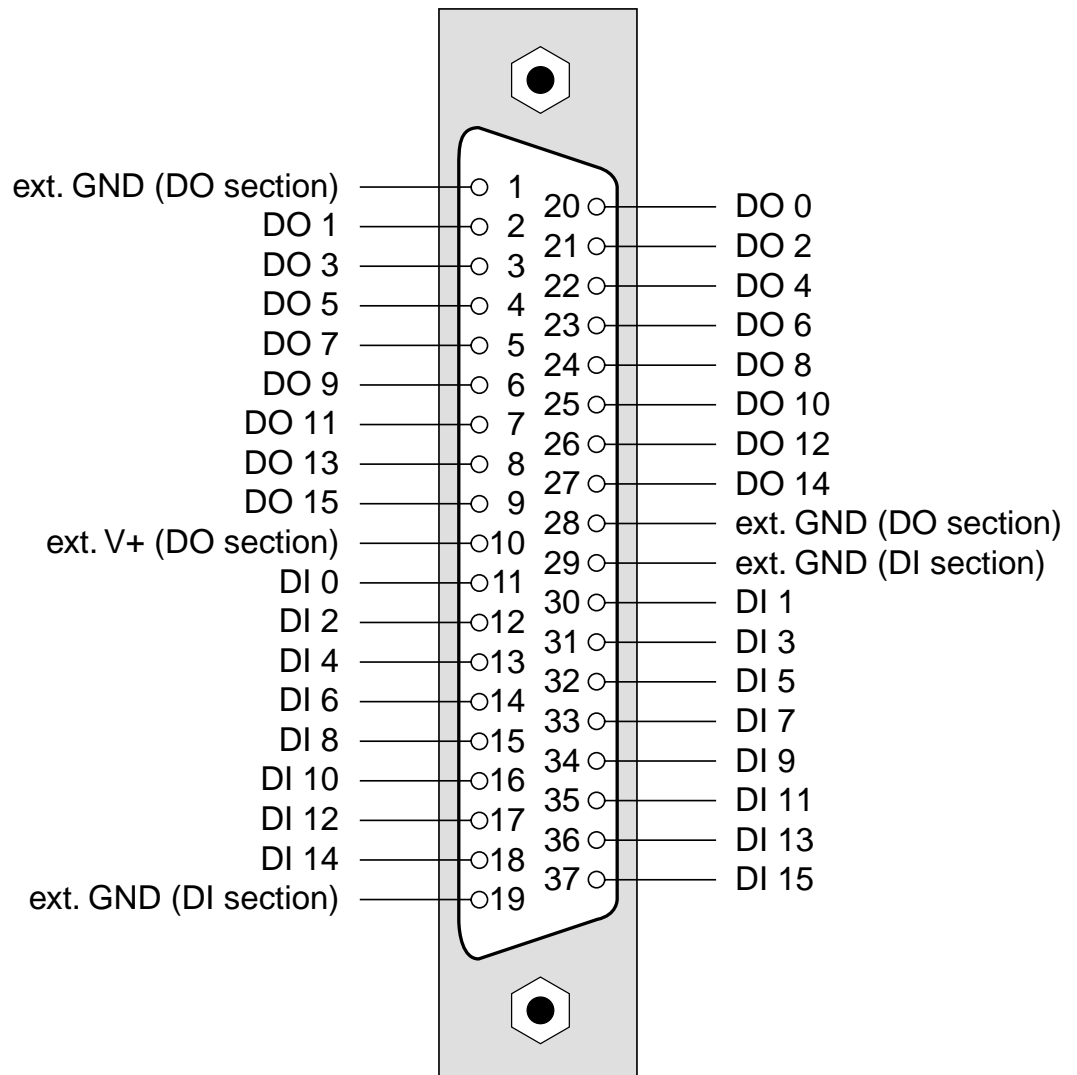


Diagram 16: Pinout of the 37pin female D-Sub on ME-81 ISA

C Accessories

Optionally the following products are available:

ME-AB-D37M

37pin D-Sub connector block (male) for ME-81 ISA

ME-AB-D78M

78pin D-Sub connector block (male) for ME-8100A, ME-8100B
PCI and cPCI

ME-AK-D37

37pin D-Sub cable (male - female), 2 m, for ME-81 ISA

ME-AK-D78

78pin D-Sub cable (male - female), 2 m, for ME-8100A, ME-8100B
PCI and cPCI

D Technical Questions

D1 Hotline

If you should have any technical questions or problems with the board hardware or the driver software, please send a fax to our hotline:

Fax hotline: ++ 49 (0) 89/89 01 66 28

eMail: support@meilhaus.de

Please give a full description of the problems and as much information as possible, including operating system information.

D2 Service address

We hope that your board will never need to be repaired. If this should become necessary please contact us at the following address:

Meilhaus Electronic GmbH

Service Department

Fischerstraße 2

D-82178 Puchheim/Germany

If you would like to send a board to Meilhaus Electronic for repair, please do not forget to add a full description of the problems and as much information as possible, including operating system information.

D3 Driver Update

The current driver versions for Meilhaus boards and our manuals in PDF format are available under www.meilhaus.com.

E Index

Function Reference

- me8100CntRead 58
- me8100CntWrite 59
- me8100DIGetBit 48
- me8100DIGetIntStatus 49
- me8100DIGetWord 50
- me8100DIOSetIntMode 51
- me8100DIOSetMask 52
- me8100DIOSetPattern 53
- me8100DIOSetTristateOFF 54
- me8100DIOSetTristateON 55
- me8100DisableInt 60
- me8100DOSetBit 56
- me8100DOSetWord 57
- me8100EnableInt 61
- me8100GetBoardVersion 45
- me8100GetDLLVersion 45
- me8100GetDrvErrMess 64
- me8100GetIrqCnt 62
- me8100PROMVersion 46
- me8100SetSinkSourceMode 47
- _me81DIGetIntStatus 49
- _me81DIOSetMask 52
- _me8xDIGetBit 48
- _me8xDIGetWord 50
- _me8xDIOSetIntMode 51
- _me8xDIOSetPattern 53
- _me8xDIOSetTristateOFF 54
- _me8xDIOSetTristateON 55
- _me8xDisableInt 60
- _me8xDOSetBit 56
- _me8xDOSetWord 57
- _me8xEnableInt 61
- _me8xGetDLLVersion 45
- _me8xGetDrvErrMess 64
- _me8xPROMVersion 46

A

- Accessories 70
 - Cable 70
 - Connector block 70
- Appendix 65

B

- Base address 13
- Bit Pattern Change 17
- Bit Pattern Compare 17
- Block Diagram 15

C

- Cascading the Counters 25
- Connectors 68
- Counter Functions

- me8100CntRead 58
- me8100CntWrite 59
- Counter Register 28
- Counter Switching 23
- D**
- Default Settings 14
- Description of the API Functions 43
- Digital I/O 16
 - _me81DIGetIntStatus 49
 - _me81DIOSetMask 52
 - _me8xDIGetBit 48
 - _me8xDIGetWord 50
 - _me8xDIOSetIntMode 51
 - _me8xDIOSetPattern 53
 - _me8xDIOSetTristateOFF 54
 - _me8xDIOSetTristateON 55
 - _me8xDOSetBit 56
 - _me8xDOSetWord 57
- me8100DIGetBit 48
- me8100DIGetIntStatus 49
- me8100DIGetWord 50
- me8100DIOSetIntMode 51
- me8100DIOSetMask 52
- me8100DIOSetPattern 53
- me8100DIOSetTristateOFF 54
- me8100DIOSetTristateON 55
- me8100DOSetBit 56
- me8100DOSetWord 57
- DIO Modes
 - Bit Pattern Change 17
 - Bit Pattern Compare 17
 - Simple I/O 16
- Driver General 41
- Driver Update 71
- E**
- Error Handling
 - _me8xGetDrvErrMess 64
 - me8100GetDrvErrMess 64
- Example Programs 35
- F**
- Function reference 41
- G**
- General Functions
 - _me8xGetDLLVersion 45
 - _me8xPROMVersion 46
 - me8100GetBoardVersion 45
 - me8100GetDLLVersion 45
 - me8100PROMVersion 46
 - me8100SetSinkSourceMode 47
- H**
- Hardware Description 15
- I**
- Important Note for ISA Models 9
- Interrupt Handling

- _me8xDisableInt 60
 - _me8xEnableInt 61
 - me8100DisableInt 60
 - me8100EnableInt 61
 - me8100GetIrqCnt 62
- Interrupts 13
- Introduction 7
- J**
 - Jumper location 12
 - Jumper settings 13
- K**
 - Kernel driver 41
- L**
 - LabVIEW™
 - Example programs 37
 - Programming 36
 - Virtual Instruments 37
- M**
 - ME Board Menu 36
 - Model Overview 8
- N**
 - Naming Conventions 42
- O**
 - Operation Modes 16
- P**
 - Package contents 7
 - Performance Notes 8
- Pinout 68
- Programming
 - Order of Operation 33
 - Register Level 37
 - under High Level Languages 33
 - under LabVIEW 36
 - under VEE 35
- R**
 - Register Description 26
 - Register Programming 37
- S**
 - Service and Support 71
 - Software Support 10
 - Specifications 65
 - Switching
 - of Counters 23
 - of Inputs ME-81 18
 - of Inputs ME-8100 20
 - of Outputs ME-81 19
 - of Outputs ME-8100 21
 - System requirements 9
- T**
 - Technical Questions 71
 - Test Program 32
- V**
 - VEE
 - Example Programs 36

ME Board Menu 36

Programming 35

User Objects 35

VxD driver 41

W

WDM Driver 41