Modular Computers ®

CP370

Multifunction Data Acquisition/Control Board for CompactPCI Applications

Manual ID 19705, Rev. Index 0200
Mar 99

The product described in this manual is in compliance with all applied CE standards.

This page was intentionally left blank.

Preface

Revision History ..... *0 - 4*
Trademarks ..... *0 - 4*
Explanation of Symbols ..... *0 - 5*
For Your Safety ..... *0 - 6*
High Voltage Safety Instructions ..... *0 - 6*
Special Handling and Unpacking Instructions ..... *0 - 6*
General Instructions on Usage ..... *0 - 7*
Two Years Warranty ..... *0 - 8*
Table of Contents ..... *0 - 9*

Revision History

Revision History			
Manual/Product Title:	**CP370**		
Manual ID Number:	**19705**		
Rev. Index	**Brief Description of Changes**	**Board Index**	**Date of Issue**
0100	Initial Issue	1.0	Aug. 98
0200	Electronic/paper ID; General revision of manual	1.0	Feb. 99

This document contains information proprietary of *PEP Modular Computers*. It may not be copied or transmitted by any means, disclosed to others or stored in any retrieval system or media, without the prior written consent of *PEP Modular Computers GmbH* or one of its authorized agents.

The information contained in this document is, to the best of our knowledge, entirely correct. However, *PEP Modular Computers* cannot accept liability for any inaccuracies, or the consequences thereof, nor for any liability arising from the use or application of any circuit, product, or example shown in this document.

PEP Modular Computers reserve the right to change, modify, or improve this document or the product described herein, as seen fit by *PEP Modular Computers* without further notice.

Trademarks

PEP Modular Computers, the *PEP* logo and “CXM” are trade marks owned by *PEP Modular Computers GmbH*, Kaufbeuren (Germany). In addition, this document may include names, company logos and trademarks, which are registered trademarks and, therefore, proprietary of their respective owners.

Explanation of Symbols



CE Conformity

This symbol indicates that the item described in this manual is in compliance with all applied CE standards. See also the section “Applied standards” of this manual.



Caution!

This symbol and title warn you of hazards due to electrical shocks (> 60 V) when touching products or parts of them. The non-observance of the measures indicated and/or prescribed by the law may cause harm to your product and/or life/health.

See also the section “High Voltage Safety Instructions”.



ESD-Sensitive Device!

This symbol and title inform you that electronic boards and their components are sensitive to static electricity. Therefore, care must be taken during all handling operations and inspections of this product, in order to ensure product integrity at all times.

Please read also the section “Special Handling and Unpacking Instructions” of this manual.



Attention!

This symbol and title emphasize aspects which, if not read through carefully by the reader, might cause hazards to health and/or damages to material.



Note:

This symbol and title emphasize aspects the reader should read through carefully for his or her own advantage.



PEP Advantage

This symbol and title emphasize advantages or positive aspects of a product and/or procedure.



Troubleshooting

This symbol and title characterize a message containing useful information on troubleshooting and problem solving.

For your safety

Your new *PEP* product was developed and tested carefully to provide all features necessary to ensure the renown electrical safety requirements. It was also designed for a long fault-free life. However, the life expectancy of your product can be drastically reduced by improper treatment during unpacking and installation. Therefore, in the interests of your own safety and of the correct operation of your new *PEP* product, you are requested to conform with the following guidelines.

High Voltage Safety Instructions



Warning!

All operations on this device must be carried out by sufficiently skilled personnel only.



Caution!

However, serious electrical shock hazards exist during all installation, repair and maintenance operations with this product. Therefore, always unplug the power cable to avoid exposure to hazardous voltage.
Before installing your new *PEP* product into a system always ensure that your mains power is switched off. This applies also to the installation of piggybacks.

Special Handling and Unpacking Instructions



ESD Sensitive Device!

Electronic boards and their components are sensitive to static electricity. Therefore, care must be taken during all handling operations and inspections of this product, in order to ensure product integrity at all times.

- Do not handle this product out of its protective enclosure while it is not used for operational purposes, unless it is otherwise protected.
- Whenever possible, unpack or pack this product only at EOS/ESD safe work stations. Where safe work stations are not guaranteed, it is important for the user to be electrically discharged before touching the product with his/her hands or tools. This is most easily done by touching a metal part of your system housing.
- It is particularly important to observe standard anti-static precautions when changing piggybacks, ROM devices, jumper settings etc. If the product contains batteries for RTC or memory back-up, ensure that the board is not placed on conductive surfaces, including anti-static plastics or sponges. They can cause short circuits and damage the batteries or tracks on the board.

General Instructions on Usage

- In order to maintain *PEP's* product warranty, this product must not be altered or modified in any way. Changes or modifications to the device, which are not explicitly approved by *PEP Modular Computers* and described in this manual or received from *PEP* Technical Support as a special handling instruction, will void your warranty.
- This device should only be installed in or connected to systems that fulfill all necessary technical and specific environmental requirements. This applies also to the operational temperature range of the specific board version, which must not be exceeded. If batteries are present, their temperature restrictions must be taken into account.
- In performing all necessary installation and application operations, please, follow only the instructions supplied by the present manual.
- Keep all the original packaging material for future storage or warranty shipments. If it is necessary to store or ship the board please re-pack it as nearly as possible in the manner in which it was delivered.
- Special care is necessary when handling or unpacking the product. Please, consult the special handling and unpacking instruction on the previous page of this manual.

Two Years Warranty

PEP Modular Computers grants the original purchaser of *PEP* products a ***TWO YEARS LIMITED HARDWARE WARRANTY*** as described in the following. However, no other warranties that may be granted or implied by anyone on behalf of *PEP* are valid unless the consumer has the express written consent of *PEP Modular Computers*.

PEP Modular Computers warrants their own products, excluding software, to be free from manufacturing and material defects for a period of 24 consecutive months from the date of purchase. This warranty is not transferable nor extendible to cover any other users or long-term storage of the product. It does not cover products which have been modified, altered or repaired by any other party than *PEP Modular Computers* or their authorized agents. Furthermore, any product which has been, or is suspected of being damaged as a result of negligence, improper use, incorrect handling, servicing or maintenance, or which has been damaged as a result of excessive current/voltage or temperature, or which has had its serial number(s), any other markings or parts thereof altered, defaced or removed will also be excluded from this warranty.

If the customer's eligibility for warranty has not been voided, in the event of any claim, he may return the product at the earliest possible convenience to the original place of purchase, together with a copy of the original document of purchase, a full description of the application the product is used on and a description of the defect. Pack the product in such a way as to ensure safe transportation (see our safety instructions).

PEP provides for repair or replacement of any part, assembly or sub-assembly at their own discretion, or to refund the original cost of purchase, if appropriate. In the event of repair, refunding or replacement of any part, the ownership of the removed or replaced parts reverts to *PEP Modular Computers*, and the remaining part of the original guarantee, or any new guarantee to cover the repaired or replaced items, will be transferred to cover the new or repaired items. Any extensions to the original guarantee are considered gestures of goodwill, and will be defined in the "Repair Report" issued by *PEP* with the repaired or replaced item.

PEP Modular Computers will not accept liability for any further claims resulting directly or indirectly from any warranty claim, other than the above specified repair, replacement or refunding. In particular, all claims for damage to any system or process in which the product was employed, or any loss incurred as a result of the product not functioning at any given time, are excluded. The extent of *PEP Modular Computers* liability to the customer shall not exceed the original purchase price of the item for which the claim exists.

PEP Modular Computers issues no warranty or representation, either explicit or implicit, with respect to its products' reliability, fitness, quality, marketability or ability to fulfil any particular application or purpose. As a result, the products are sold "as is," and the responsibility to ensure their suitability for any given task remains that of the purchaser. In no event will *PEP* be liable for direct, indirect or consequential damages resulting from the use of our hardware or software products, or documentation, even if PEP were advised of the possibility of such claims prior to the purchase of the product or during any period since the date of its purchase.

Please remember that no *PEP Modular Computers* employee, dealer or agent is authorized to make any modification or addition to the above specified terms, either verbally or in any other form, written or electronically transmitted, without the company's consent.

Table of Contents

Chapter 1

1. Introduction ..... 1 - 3
1.1 System Overview ..... 1 - 3
1.2 System Components ..... 1 - 4
1.3 Board Introduction ..... 1 - 8
1.4 Functional Block Diagram ..... 1 - 11
1.5 Technical Specifications ..... 1 - 12
1.6 Front Panel ..... 1 - 16
1.7 Board Layout ..... 1 - 16
1.8 Applied Standards ..... 1 - 17
1.9 Related Publications ..... 1 - 18

Chapter 2

2. Functional Description ..... 2 - 3
2.1 Electrical Isolation ..... 2 - 3
2.2 Channel Switching ..... 2 - 3
2.3 Operating Modes ..... 2 - 9

Chapter 3

3. Installation ..... 3 - 3
3.1 Board Installation ..... 3 - 3
3.2 Driver Installation ..... 3 - 4

3.3 HP VEE Component Installation ..... 3 - 5
3.4 Uninstall ..... 3 - 6
3.5 Measuring Points and Potentiometers ..... 3 - 7
3.6 Balancing ..... 3 - 8
3.7 Test Program ..... 3 - 9
3.8 PEP Technical Support ..... 3 - 9

Chapter 4

4. Configuration ..... 4 - 3
4.1 Jumper Settings ..... 4 - 3
4.2 Pinouts ..... 4 - 3

Chapter 5

5. Programming ..... 5 - 3
5.1 High-Level Language Programming ..... 5 - 3
5.2 HP VEE Programming ..... 5 - 4
5.3 General Rules for the Order of Operation ..... 5 - 8

Chapter 6

6. Software ..... 6 - 3
6.1 Functional Driver Overview ..... 6 - 3
6.2 Standards Used in this Section ..... 6 - 3
6.3 Definitions ..... 6 - 3
6.4 Description of API Functions ..... 6 - 4

Chapter 1

Introduction

1.1 System Overview ..... 1 - 3
1.1.1 Note on CompactPCI ..... 1 - 3
1.1.2 Main Features of CompactPCI Systems ..... 1 - 3
1.2 System Components ..... 1 - 4
1.3 Board Introduction ..... 1 - 8
1.3.1 CP370 Models ..... 1 - 9
1.3.2 Accessories ..... 1 - 10
1.3.3 Available Software ..... 1 - 10
1.4 Functional Block Diagram ..... 1 - 11
1.5 Technical Specifications ..... 1 - 12
1.6 Front Panel ..... 1 - 16
1.7 Board Layout ..... 1 - 16
1.8 Applied Standards ..... 1 - 17
1.8.1 CE Compliance ..... 1 - 17
1.8.2 Mechanical Compliance ..... 1 - 17
1.8.3 Environmental Tests ..... 1 - 17
1.9 Related Publications ..... 1 - 18
1.9.1 CompactPCI Systems/Boards ..... 1 - 18
1.9.2 PMC Add-On Modules/Carriers ..... 1 - 18

This page was intentionally left blank.

1. Introduction

1.1 System Overview

The *PEP Modular Computers* CompactPCI systems described in this chapter operate with the PCI bus architecture to support additional I/O and memory-mapped devices as required by various industrial applications. In the following you will find the most important information on all system relevant CompactPCI features. For more detailed information concerning the CompactPCI standard, please consult the complete Peripheral Component Interconnect (PCI) and CompactPCI Specifications. For further information regarding these standards and their use, visit the homepage of the *PCI Industrial Computer Manufacturers Group (PICMG)*.

1.1.1 Note on CompactPCI

CompactPCI is an extension of the PCI specification. It has been optimized for industrial and embedded applications that require a more robust mechanical form factor as compared to Desktop PCI. CompactPCI systems use industry standard mechanical components and high performance connector technologies to provide systems that are well suited for rugged applications. CompactPCI stands for systems that are electrically compatible with the PCI Specification, allowing low cost PCI components to be used. CompactPCI is an open specification supported by PICMG.

1.1.2 Main Features of CompactPCI Systems

Some of the outstanding features of the CompactPCI systems compliant with Specification 2.0, Release 2.1 are:

- PCI signalling
- 32 and 64 bit data transfer at 33 MHz
- up to 16 PCI slots per backplane
- industry standard software support
- 3U small form factor (100 mm by 160 mm)
- 6U form factor (233 mm by 160 mm)
- Eurocard packaging
- wide variety of available I/O functions
- industry support from over 350 members.
- possibility of PMC modules on some boards



PEP Advantage

PEP Modular Computers' CompactPCI systems are designed as open systems, able to be expanded at any time, so that their backplanes can be equipped with precisely the CPCI boards that correspond to a customer's specific needs. However, in order to supply you with an appropriate choice of workstations, the *PEP* basic equipment is divided into pre-configured sets and custom solutions.

1.2 System Components

PEP Modular Computers have devised their CompactPCI systems as a comprehensive open solution for industrial environments, offering different workstation configurations which are capable of including all the components necessary to fulfill the requirements of virtually all existing system functionalities.

CompactPCI Backplane(s)

- 4-slot backplane;
- 6-slot backplane;
- 8-slot backplane;
- 16-slot backplane.

CPU Function

PEP Modular Computers provides CPU boards corresponding in size and characteristics to the special features of the *PEP* systems.

Display Related Functions

Display related functions such as frame grabbing are supported by dedicated boards.

Communication

Communication boards are provided for the currently relevant industrial communication systems such as Fast Ethernet.

Fieldbus Control

Fieldbus control boards provide data exchange with field control and automation subsystems like *PEP Modular Computers'* SMART2 or VMEbus systems using up-to-date transmission standards for example:

- CAN fieldbus control
- PROFIBUS control

Industrial I/O Functions

PEP's CompactPCI systems support an ever increasing number of industrial I/O functionalities in the fields of:

- Digital I/O
- Analog I/O

Self-Testing

A special *PEP* self-testing board provides a trouble-spotting capability within your *PEP* system.

System Hardware

The *PEP* CompactPCI system hardware includes housings, storage devices, power supply units, network adapters etc. The most important system hardware elements with which to configure your CompactPCI system are as follows:

Multiprocessor

In a CompactPCI multiprocessor system, a system controller communicates with other CPU's through a non-transparent PCI / PCI bridge. *PEP Modular Computers'* CompactPCI system controllers, CP600/CP610, which are designed to work together with PEP CPU's, CP611/CP612, provide full multiprocessor solutions.

Housings

- Board cages for 19-inch cabinets or wall mounting
- PCI tower
- PCI desktop housing

Mass Storage Devices

- Hard-disk drive
- Floppy-disk drive
- CD-ROM drive

Power Supply Units

- 6U "M", 270 W, 120/230 V AC, 3.3V/30A, 5.1V/20A, 12V/4A, -12V/2A
- 3U "M", 180 W, 120/230 V AC, 3.3V/14A, 5.1V/20A, 12V/2A, -12V/1A
- "ATX", 235 W, 120/230 VAC, 3.3V/14A, 5V/22A, 12V/8A,-12V/1A, -5V/0.5A, 5VSB/0.1A

Form Factors

PEP Modular Computers CompactPCI boards are available in different form factors, mainly as 3U (single-height) and 6U (double-height) boards and as 4HP (single-width), 8HP (double-width), and 12HP (triple-width) boards.

Please note that the following units of measurement are used to express the dimensions of *PEP* CompactPCI card slots:

- Height: 1 U = 44.45 mm
- Width: 1 HP = 5.08 mm

On certain boards/carriers with PMC connectors, PMC add-on modules can also be installed. Please note that the form factor of *PEP Modular Computers* PMC modules is:

- 149mm x 74mm (single-size standard).

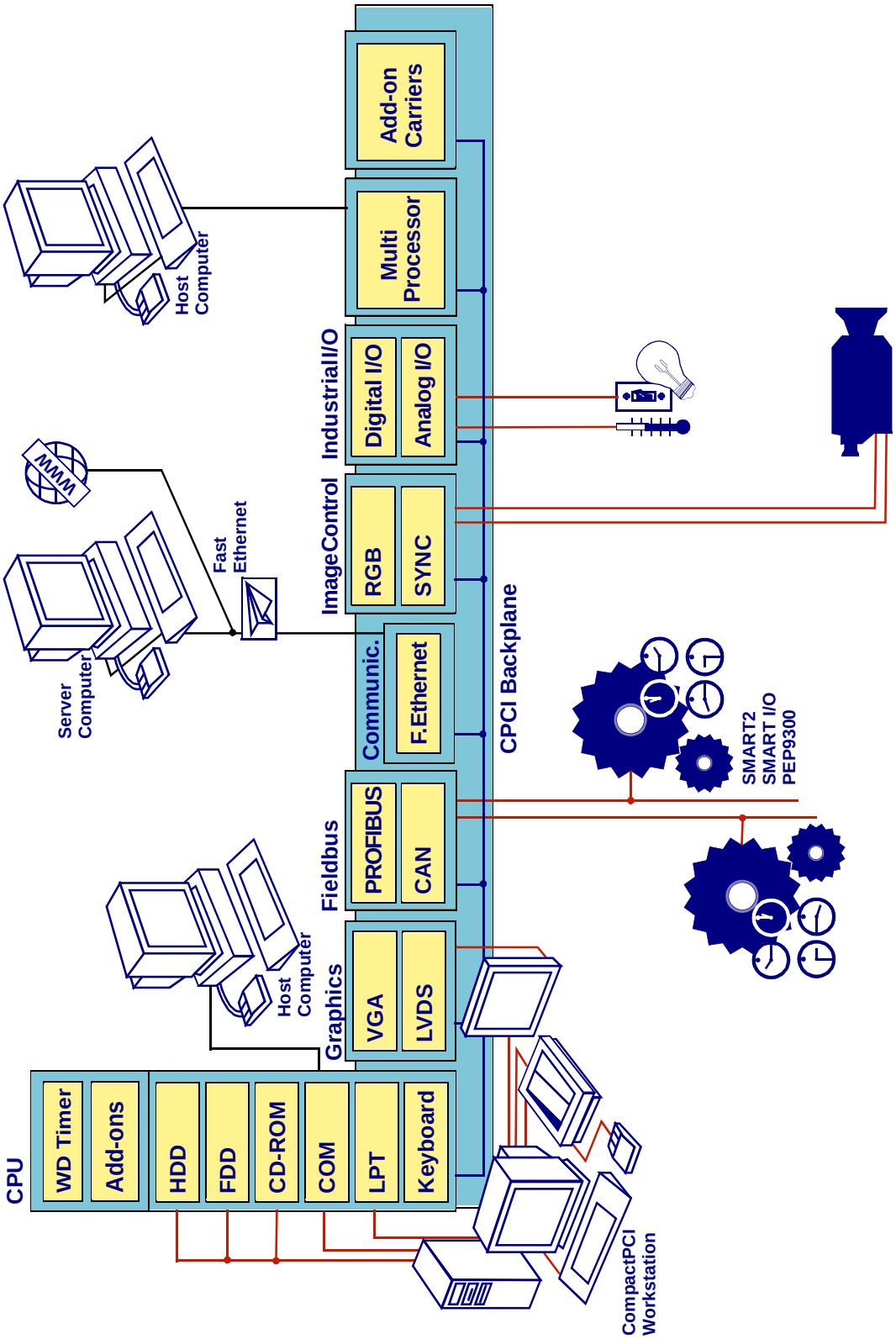
For a detailed description of the *PEP Modular Computers* CompactPCI modules please consult the specific component's manuals or data sheets.



PEP Advantage

As the *PEP Modular Computers* CompactPCI system provides comprehensive open solutions, new features and functionalities may be added to our range. To keep abreast of the latest developments, please contact your local *PEP Sales Office* or visit the ***PEP Web Site***.

Figure 1-1: Example of a *PEP Modular Computers* CompactPCI System



1.3 Board Introduction

The CP370 is a data acquisition and control board using innovative high technology. It consists of:

- CP370 CompactPCI Board
- Driver Kit (1,44 MB HD)
- 78-pin male DSUB connector

The CP370 provides full Plug&Play operation. There are no jumpers or switches to be set.

The CP370 is a high quality board for use in CompactPCI systems. The board is supported for operation under the Windows NT 4.0 operating systems. The hardware comes complete with the following components (function may vary depending in the CP370 model you have chosen):

- A/D converter with 12 bit A/D resolution
- D/A converter with 12 bit D/A resolution
- 32 digital I/O channels
- Pulse Width Modulation (PWM)
- Up/Down counter

Of particular interest on the CP370 is the “VEC” technology giving the possibility to change the performance characteristics of the board in order to suit specific needs without purchasing progressively new hardware. The abbreviation “VEC” is short for “virtual embedded circuitry”. You can choose from 3 different CP370 models, each with different functions and complexities:

- CP370: data acquisition
- CP370-SIM: data acquisition by simultaneous sampling
- CP370-SIG: data acquisition and signal generation



Note:

The CP370 and CP370-SIM board variants can be upgraded to the CP370-SIG, while no upgrading from the CP370 to the CP370-SIM is possible.

1.3.1 CP370 Models

Table 1-1: The CP370 Models in Overview

CP370	CP370-SIM Data Acquisition by Simultaneous Sampling	CP370-SIG Data Acquisition and Signal Generation
Single state machine for all sections	Single state machine for all sections	Several independent state machines for simultaneous data aquisition and data transfer
A/D Operation Modes: • Single value measurement • Block measurement • Conversion based on programmable channel list • Conversion can be triggered by: ext. Trigger, Counter (32 bit for SCAN-time and 16 bit for CHAN-time), or by software	A/D Operation Modes: • Single value measurement • Block measurement • Conversion based on programmable channel list • Conversion can be triggered by: ext. Trigger, Counter (32 bit for SCAN-time and 16 bit for CHAN-time), or by software • Simultaneous sampling for 4 single ended channels (and 12 “normal” single ended channels)	A/D Operation Modes: • Single value measurement • Block measurement • Conversion based on programmable channel list • Conversion can be triggered by: ext. Trigger, Counter (32 bit for SCAN-time and 16 bit for CHAN-time), or by software
D/A Operation Modes: • Single value output (transparent) • Simultaneous output to one or more channels (buffered)	D/A Operation Modes: • Single value output (transparent) • Simultaneous output to one or more channels (buffered)	D/A Operation Modes: • Single value output (transparent) • Simultaneous output to one or more channels (buffered) • “Continuous” mode: Data block loaded into the D/A FIFO; on FIFO half full the next block is read in • “Wraparound”-Mode: Data block loaded into the D/A FIFO and output repeatedly
Digital-I/O Operation mode: • 2 x 16 Bit I/O Ports (independently configurable)	Digital-I/O Operation mode: • 2 x 16 Bit I/O Ports (independently configurable)	Digital-I/O Operation mode: • 2 x 16 Bit I/O Ports (independently configurable) • Pulse Width Modulation

Table 1-1: The CP370 Models in Overview

CP370	CP370-SIM Data Acquisition by Simultaneous Sampling	CP370-SIG Data Acquisition and Signal Generation
Counter Functions: • up/down counter • loadable start and stop value • latched counter value readable	Counter Functions: • up/down counter • loadable start and stop value • latched counter value readable	Counter Functions: • up/down counter • loadable start and stop value • latched counter value readable

1.3.2 Accessories

Optionally the following products are available:

- CP-CBlock: 78-pin connector block for CP370
- CP-ADAP-CP370: board-to-connector block cable, 1 meter

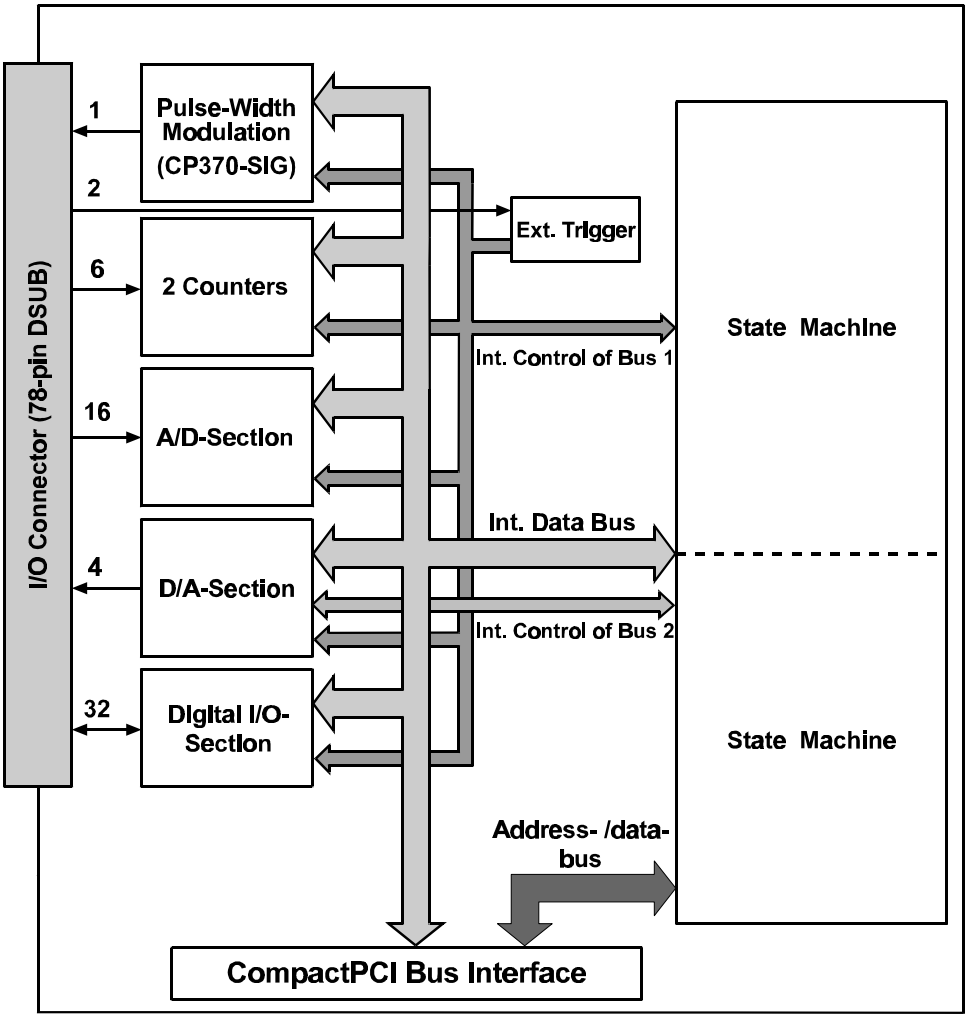
1.3.3 Available Software

Table 1-2: Available Software

Windows Support	Windows NT Driver Disk for CP370
High Level Language Support (32 bit)	Visual C++ V 4.0 or higher Delphi V 2.0 or higher Visual Basic V 4.0 or higher
Graphical Programming Tools Support	CP370 Driver System for HP VEE V 4.0 or higher CP370 Driver System for LabView (optional)
Test and Demo Software	CP370 Test program - HP VEE Demos

1.4 Functional Block Diagram

Figure 1-2: CP370 Functional Block Diagram



1.5 Technical Specifications

Table 1-3: CP370 Technical Specifications

CP370		Specification		
General Information	Plug & Play Operation	Fully supported (no switches or jumpers need to be set)		
	Bus Interface	PCI bus (32 Bit, 33 MHz)		
	Physical Size	3U CompactPCI board		
	Connector	78pin D-sub female;		
	Power Consumption	5V DC ±5% at 2.1 A min. (max. 2.7 A with full DIO load)		
	Weight	200 g approx.		
Analog Inputs	Number	16 single ended/8 differential channels; CP370-SIM with 4 single ended simultaneously sampled and 12 “normal” single ended channels		
	Input Amplifier	Programmable by software		
	Input Ranges	Depends on amplification factor and operation mode uni-polar or bipolar (software programmable):		
		Factor	Unipolar	Bipolar
		1	0...10V	±10V
		2	0...5V	±5V
		4	0...2.5V	±2.5V
		8	0...1.25V	±1.25V
	Input Impedance	typically 160 MΩ (min. 70MΩ) for all input ranges		
	Input Capacitance	3 pF for all input ranges		
	Resolution	12 Bit		
	Total Sampling Rate	Max. 500 kHz		
	Simultaneous Operation Sam-pling Rates (CP370-SIM)	The following table shows the relationship between the number of simultaneously operable channels and the maximum attainable sampling rate per channel.		
		Number of Channels	max. Sampling Rate	
		2	~150 kHz	
		3	~115 kHz	
		4	~93 kHz	
	Total Accuracy	±0.5 LSB at 500 kHz and constant input voltage		

Table 1-3: CP370 Technical Specifications

CP370		Specification		
Analog Inputs	Dynamic Data Acquisition	The following table shows the relationship between reachable accuracy and the max. sampling rate by taking into consideration the settling time of a dynamic signal with a full scale step of ±10 V (amplification factor = 1)		
		Sampling rate up to:	430kHz	500kHz
		Accuracy better than:	±3 LSB	±7 LSB
	Input Protection	±15 V		
	Electrical Isolation	up to 1000 V, A/D section from PC ground and from the rest of the board		
	DC/DC Converter	One for ±5 V and one for ±15 V		
	FIFO	2 k FIFO for A/D data		
	Channel List	Max. 1024 entries (channel number, amplification factor, uni-/bipolar, single ended or differential operation)		
	Triggering	Internal counter by software, external trigger		
	A/D Timer	Programmable counter for SCAN-time: channel list is processed on each pulse; Programmable counter for CHAN-time: the processing time between 2 consecutive channel list entries (sampling rate)		
	Start Frequency for SCAN-time	3 MHz		
	SCAN-time Range	4 µs ... ~ 23 minutes (32-bit divider)		
	Start Frequency for CHAN-time	3 MHz		
	CHAN-time range	2 µs ... ~ 22 ms (16-bit divider)		
	Operation Modes	All Board Variants: • Single value measurement • Block measurement • Conversion based on programmable channel list (CHAN- and SCAN-time programmable) CP370-SIM: • 4 simultaneously sampled input channels		

Table 1-3: CP370 Technical Specifications

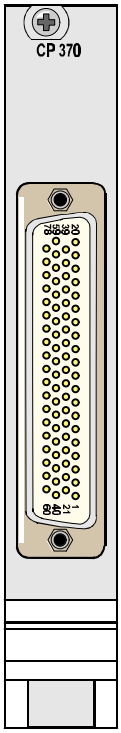
CP370		Specification
Analog Outputs	D/A Converter	Quad 12 Bit D/A converter
	Number of Outputs	4 independent channels with sense lines
	Sense Lines	1 sense input per D/A channel for automatic stabilization of output voltage depending on the load
	Resolution	12 bit
	Output Ranges	0...10 V, ±5 V, ±10 V Outputs are software-programmable independently
	Output Current	Max. 2 mA per channel
	Conversion Rate	CP370/CP370-SIM: • no FIFO operational mode • max. 100 kHz (at ±10 V, system dependent) CP370-SIG: • no FIFO operational mode • operational modes: continuous and wraparound • max. 100 kHz (at ±10 V, timer controlled, system independent)
	DC/DC Converter	±15 V
	Start Frequency for D/A Time	3 MHz
	D/A Output Timing Range	10 µs ... ~ 22 ms (16-bit divider)
	FIFO	4 k FIFO for D/A output
	Operation Modes	CP370/CP370-SIM: • Single value output (buffered or transparent); • Simultaneous output (buffered) CP370-SIG: • Single value output (buffered or transparent); • Simultaneous output (buffered) • “Continuous” mode • “Wraparound” mode

Table 1-3: CP370 Technical Specifications

CP370		Specification
Digital In/Outputs	Number	32 I/O channels (2 x 16 bit ports independently configurable)
	Driver Device	74ABT245
	Output Voltage (at max. current)	Low: max. 0.5 V at 24 mA High: min. 2 V at -15 mA
	Input Voltage	Low: max. 0.8 V High: min. 2 V
Pulse-Width Modulation	Pulse-Width Modulation (CP370 - SIG)	Channel number: • 1 output channel (pin 32) Frequency Range: • max. 30 kHz • min. 0.4 Hz by 16-bit divider Duty Cycle: • 0% – 99% in increments of 1% Output Voltage (at max. current) • Low: max. 0.33 V at 6 mA • High: min. 3.84 V at -6 mA
Counters	Functions	Load start value (start value < end value) Load end value Interrupt on zero-axis crossing Latched reading of current counter value
	Number	2 counters (user-accessable)
	Resolution	16 Bit
	External Clock	Max. 6 Mhz (if end value = 65535) Max. 90 kHz (if end value < 65535)
	Pulse Width	Min. 60 ns (high level)
	Input Voltage	Low: 0.8V max. High: 2.0V min.
Ext. Signals	IRQ Input, Trigger D/A, Trigger A/D	Low: 1.35V max. High: 3.85V min.
	Buffer Device	MM74HC125

1.6 Front Panel

Figure 1-3: CP370 Front Panel



1.7 Board Layout

Figure 1-4: CP370 Board Layout (Front Side)

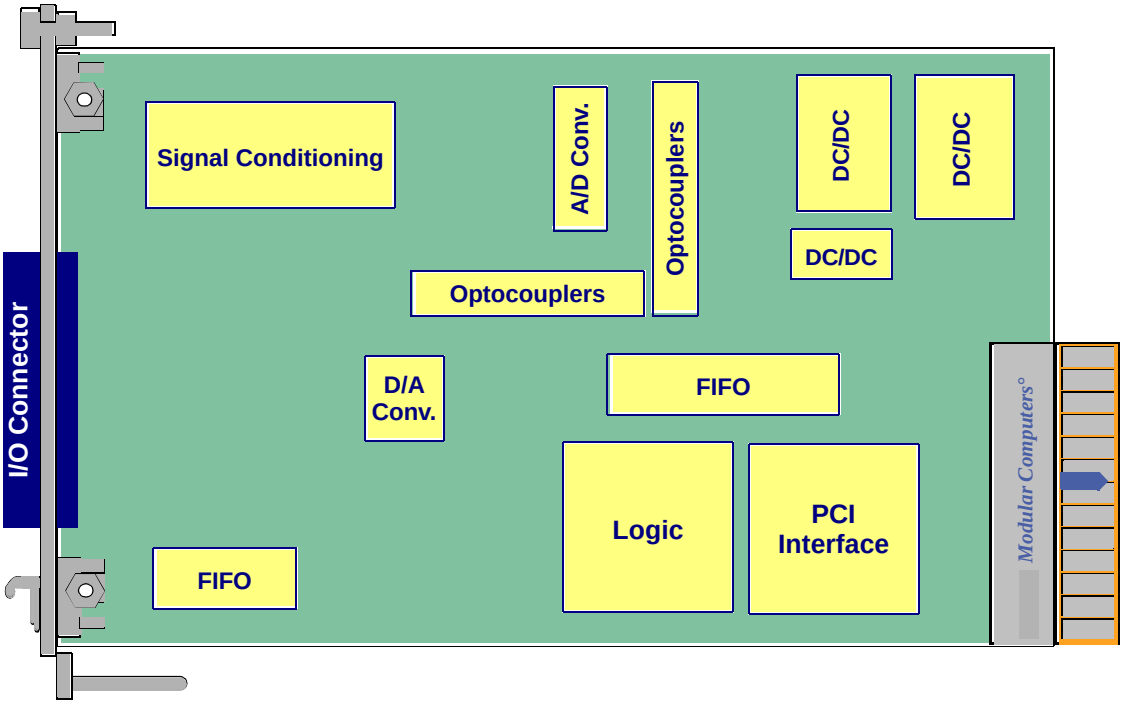
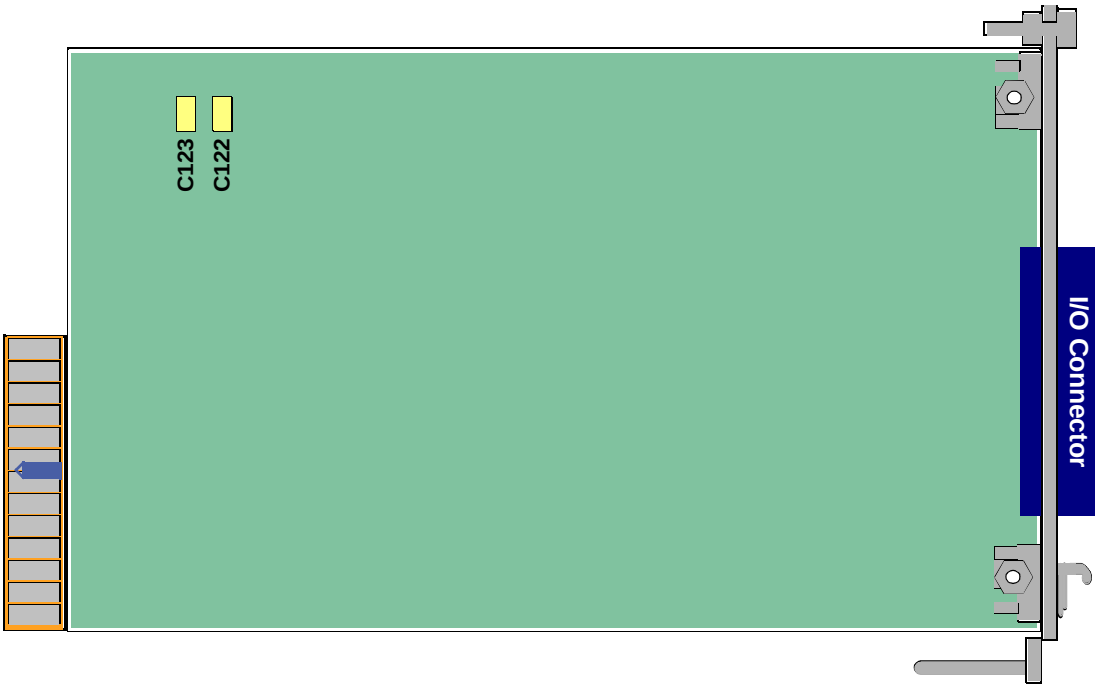


Figure 1-5: CP370 Board Layout (Reverse Side)



1.8 Applied Standards

1.8.1 CE Compliance

The *PEP Modular Computers'* CompactPCI systems comply with the requirements of the following CE-relevant standards:

- Emission EN50081-1
- Immission EN50082-2
- Electrical Safety EN60950

1.8.2 Mechanical Compliance

- Mechanical Dimensions IEEE 1101.10

1.8.3 Environmental Tests

- Vibration/Broad-Band IEC68-2-6
  Random Vibration IEC68-2-64 (3U boards)
- Permanent Shock IEC68-2-29
- Single Shock IEC68-2-27

1.9 Related Publications

1.9.1 CompactPCI Systems/Boards

CompactPCI Specification, V. 2.0, Rev. 2.1

1.9.2 PMC Add-On Modules/Carriers

- Draft Standard for a Common Mezzanine Card Family, P1386/Draft 2.0
- Draft Standard Physical and Environment Layers for PCI Mezzanine Cards, P1386.1/Draft 2.0

This page was intentionally left blank.

Chapter 2

Functional Description

2.1 Electrical Isolation ..... 2 - 3
2.2 Channel Switching ..... 2 - 3
2.2.1 A/D Channel Switching ..... 2 - 3
2.2.1.1 Single-Ended Operation ..... 2 - 4
2.2.1.2 Differential Operation ..... 2 - 5
2.2.2 D/A Channel Switching ..... 2 - 6
2.2.3 Switching the Digital I/O and PWM Lines ..... 2 - 8
2.2.4 Counter Switching ..... 2 - 8
2.3 Operating Modes ..... 2 - 9
2.3.1 A/D Operating Modes ..... 2 - 9
2.3.1.1 Single-Value Output ..... 2 - 9
2.3.1.2 Block Measurement ..... 2 - 9
2.3.1.3 Conversion Based on Channel List ..... 2 - 11
2.3.1.4 Simultaneous A/D Conversion ..... 2 - 12
2.3.2 D/A Operating Modes ..... 2 - 12
2.3.2.1 Single-Value Output ..... 2 - 12
2.3.2.2 Simultaneous Output ..... 2 - 12
2.3.2.3 Continuous Mode ..... 2 - 13
2.3.2.4 Wraparound Mode ..... 2 - 13
2.3.3 Digital I/O Operating Modes ..... 2 - 14
2.3.3.1 Digital I/O ..... 2 - 14
2.3.3.2 Pulse-Width Modulation ..... 2 - 14

Functional Description

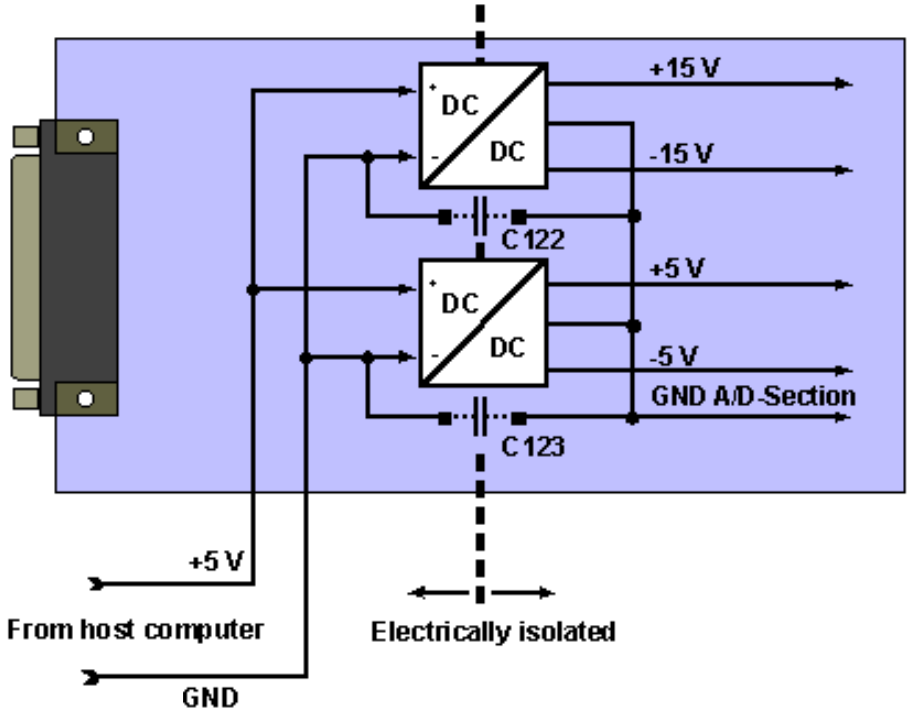
2.3.4 Counters ..... 2 - 14

2. Functional Description

2.1 Electrical Isolation

The A/D part of the CP370 is electrically isolated from the rest of the board and grounded behind the DC/DC converter. The standard accuracy of the A/D section is ±0,5 LSB. By bypassing the electrical isolation, greater accuracy can be achieved. This is done by placing two 1nF capacitors between the PC ground and the ground plane of the A/D section (see figures 2-1 and 1-5).

Figure 2-1: Electrical Isolation of the CP370



2.2 Channel Switching

2.2.1 A/D Channel Switching

The CP370 provides 16 single ended or 8 differential analog input channels. The CP370-SIM provides four simultaneously sampled inputs and twelve "normal" inputs. Depending on the CP370 model, your software will allow different modes of A/D-operation.

The channel list of the CP370 can hold up to 1024 entries. For each 8-bit entry, the following items can be set: channel number, amplification, uni/bi polar operation, and single-ended/differential operation. The bit which sets the single-ended/differential operating mode must be the same for all entries in the channel list. This means that the

board can operate either as single-ended or differential, but not both at the same time. The processing of the channel list can be started by a function call or by an external trigger pulse. The external trigger pulse pin (pin 51) is an input and is not electrically isolated from the system ground. The input can be set to trigger on positive or negative edges (set in software). Make sure that the 5V level is maintained and a reference to system ground (e.g. pin 13) is made.

Important!

For boards equipped with the simultaneous sample & hold components (CP370-SIM), only single-ended operation is possible!

The following amplification options are available:

Table 2-1: Programmable Gain : Amplification Options

Input Voltage Range		
Gain	**Unipolar**	**Bipolar**
1	0 – 10V	±10V
2	0 – 5V	±5V
4	0 – 2,5V	±2.5V
8	0 – 1,25V	±1.25V

When using dynamic amplification (differing amplifications within the same channel list) the amplifiers will need a short period of time to adjust. This will mean that the maximum sampling rate may not be achieved.

The following sections describe the switching of the analog inputs. The description will vary depending on whether single-ended or differential operation is chosen.

2.2.1.1 Single-Ended Operation

For single-ended operation, there are 16 input channels available. All inputs use the ground of the A/D section as reference. This means that all signal grounds of the input sources should be connected to the ground of the A/D section (pin 20 and 59). The signal lines of the input sources are connected to the chosen input lines.

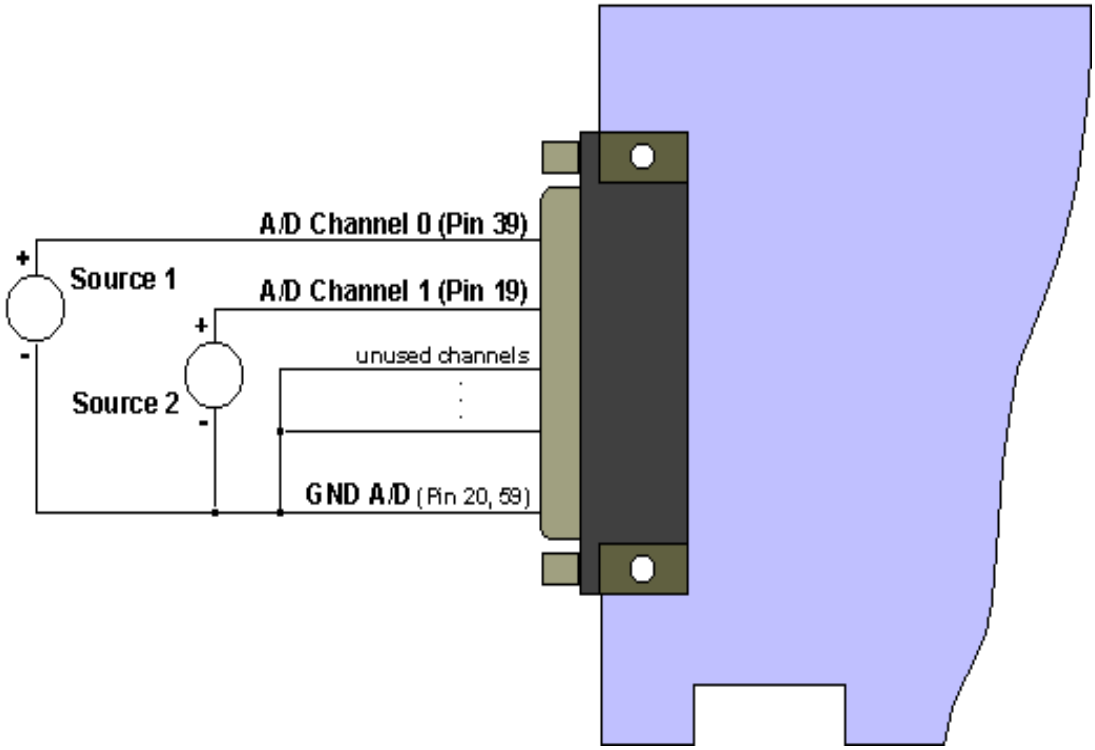
Important!

Please note that all input signal grounds must be of the same potential to avoid short circuits between the input signals.

We recommend that shielded input lines be used.

Unused input lines must be connected to the ground of the AID section.

Figure 2-2: Switching for Single-Ended Operation



2.2.1.2 Differential Operation

For differential operation, there are eight input channels available. For each input channel there is a positive and a negative input.

Please note that for differential operation a reference to analog ground is still required. This is achieved by referencing the negative input terminal to the ground of the A/D section (pin 20 and 59) with a resistor of approx. 10 kOhm. The positive signal input lines are numbered 0 to 7, and the negative input lines are numbered 8 to 15. Channel assignments are shown below:

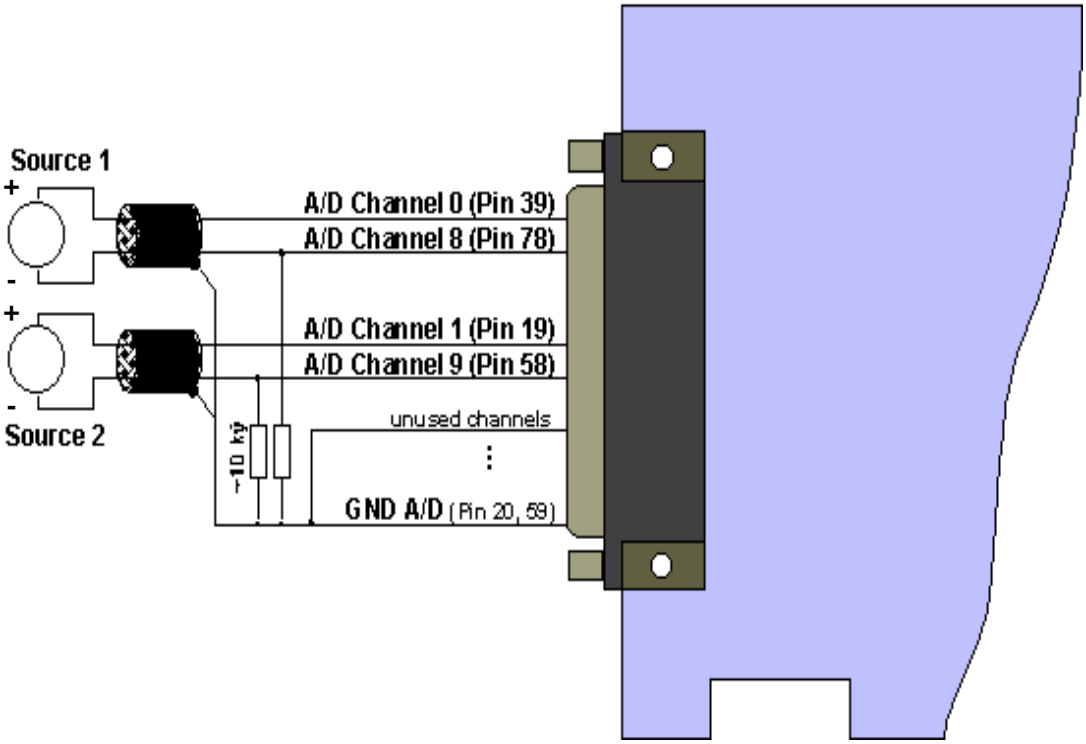
Table 2-2: : Channel assignments in differential operation

Positive signal	Pin No.	Negative signal	Pin No.
Channel 0	39	Channel 8	78
Channel 1	19	Channel 9	58
Channel 2	38	Channel 10	77
Channel 3	18	Channel 11	57
Channel 4	37	Channel 12	76
Channel 5	17	Channel 13	56
Channel 6	36	Channel 14	75
Channel 7	16	Channel 15	55

Important!

Unused input lines must be connected to the ground of the A/D section.

Figure 2-3: Switching for Differential Operation



2.2.2 D/A Channel Switching

The CP370 has 4 analog output lines with a conversion resolution of 12 bits. Depending on the CP370 model chosen, there are several different modes of operation for the D/A channels. The mode of operation is set by the software. The external trigger pulse pin (pin 52) can be set to trigger on positive or negative edges (set in software).

Note:

Please make sure the 5V level is maintained and a reference to system ground (e.g. pin 13) is made.

The D/A converter is a quad DAC type AD664. To maintain accuracy (especially when using longer lines), the CP370 provides an adjustment (D/A-Sense lines) to compensate for inaccuracies caused by signal loss over the output lines. Care should be taken to connect the sense lines as close as possible to the respective loads (see figure 2-4).

Where this is not done or impossible, the sense line must be connected directly to the output pin on the connector (see figure 2-5)

Important!

Connecting the D/A-Sense lines with the corresponding outputs is absolutely necessary for proper operation of the board!

For external switching keep in mind that the max. supply current of every D/A channel is 2 mA. After power up or a software reset all outputs are set 0 V.

Figure 2-4: D/A Output Switching With Sense Lines

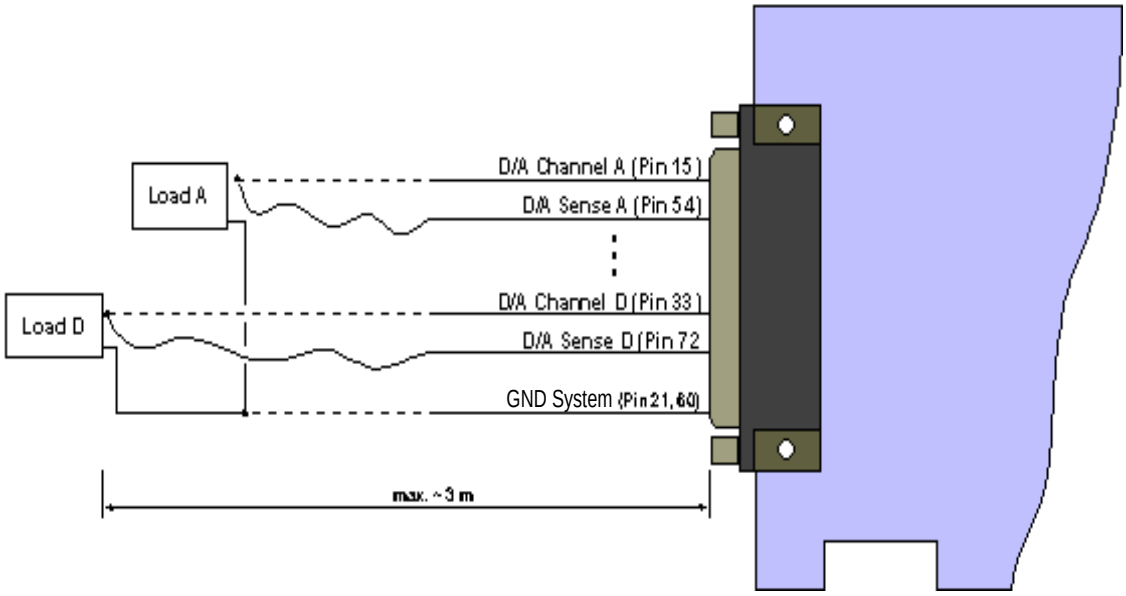
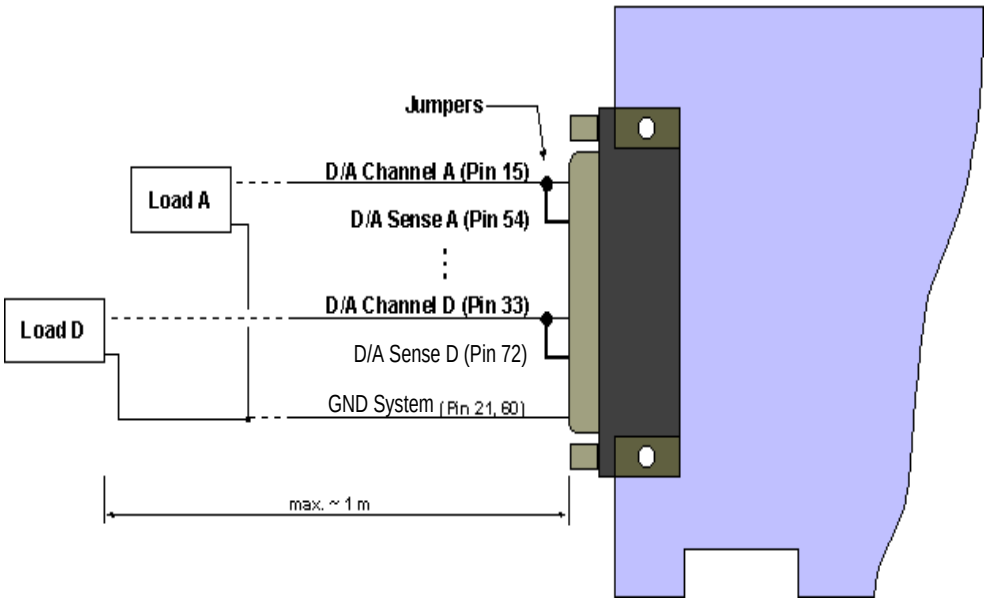


Figure 2-5: D/A Output Switching Without Sense Lines



2.2.3 Switching the Digital I/O and PWM Lines

The CP370 is provided with 32 digital I/O lines that are arranged as 2 x 16 bit ports. Each port can be independently configured as input or output. You can choose different modes of digital I/O operation. This is performed through the software (see section 2.3.3).

Using the PWM line (pin 32) you can output a square wave signal with adjustable frequency and duty cycle (see section 2.3.3). Please note that PWM is only possible with CP370 - SIG.

Important!

Make sure that the signals applied to the digital I/O stay within the TTL signal range (high level typ. 5 V, low level typ. 0 V) and that a reference to system ground is made (e. g. on pin 21)!
The maximum sink current of a digital output port may not exceed 24 mA per output bit; the maximum sink current of the PWM output line is 6 mA! See also the technical specifications in Chapter 1.

2.2.4 Counter Switching

On the CP370 two, 16-bit counters are accessible for the user. The configuration is performed by software (see section 2.3.4).

Every counter must be externally switched by the input signals “Gate”, “Clock” and “Up/Down”.

Table 2-3: CP370 Counter Switching

Clock:	Counter is triggered by positive edge
Gate	High Level “1”: enable counter; Low Level “0”:disable counter
Up/Down:	High Level “1” : count upwards; Low Level “0” count downwards

Important!

Make sure that the signals applied to the counter lines stay within the TTL signal range (high level typ. 5 V, low level typ. 0 V) and that a reference to system ground is made (e. g. on pin 21)! See also the Technical Specifications in Chapter 1.

2.3 Operating Modes

The operating modes of the CP370 depend on the model you have installed in your CompactPCI system. However, many functions are common to more than one or even to all CP370 models.

2.3.1 A/D Operating Modes

2.3.1.1 Single-Value Measurement

This operating mode is available to all CP370 models. A single value is read in from the chosen channel. The amplification can be set to 1, 2, 4, or 8, uni/bipolar operation and single-ended/differential operation can be chosen. There is no need to generate a channel list.

Use the following functions to implement the above described operation mode. Refer to chapter 6.4.3 for a full description of the functions:

`me3000AISingle`	Sampling a single value; `me3000AISingleEx` for background operation
`me3000AIStop`	Stops operation in infinite mode of background functions

2.3.1.2 Block Measurement

This operating mode is available to all CP370 models. In this mode of operation you can choose between two functions depending on the strategy required. While the function `me3000AIQuick` uses a single processing of the channel list as an interrupt event, the function `me3000AIMultiple` triggers an interrupt on “FIFO half full”. For both functions you must first enter a channel list consisting of at least one complete entry. Every entry

in the channel list consists of: the channel number, an amplification setting (1, 2, 4, 8), uni/bipolar operation and single ended or differential operation (must be the same for all channel list entries)!

Use the following functions to implement the above described operation mode. Refer to chapter 6.4.3 for a full description of the functions:

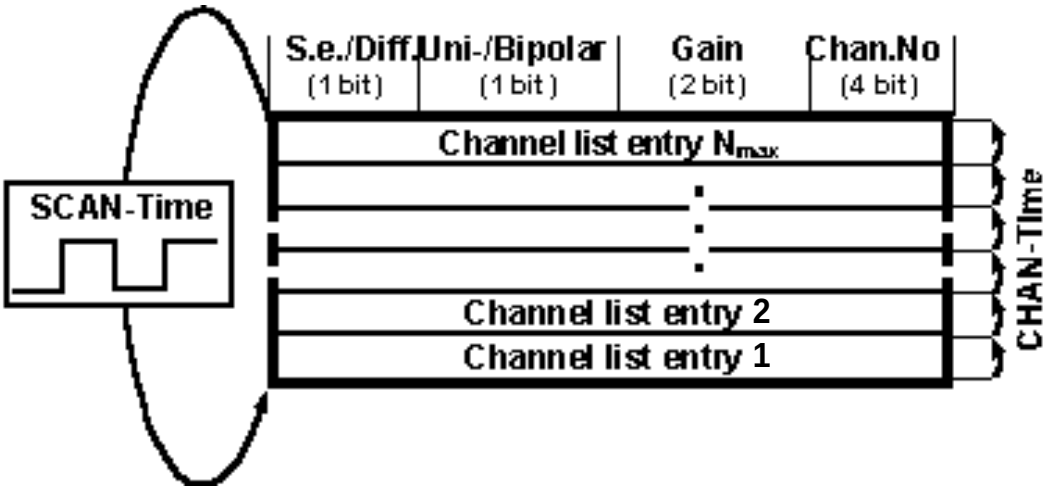
`me3000FrequencyToCounter`	Conversion of sample rate for passing to `me3000AISetTimer`
`me3000AISetTimer`	Setting the counters for A/D conversion
`me3000AISetTrigger`	Setting trigger edge and trigger source
`me3000AIClearChannelList`	Clear channel list FIFO
`me3000AIMakeChannelListEntry`	Prepare a channel list entry
`me3000AIAddToChannelList`	Writing a channel list entry to channel list FIFO
`me3000AIMultiple`	Sampling an array of values of defined size – first enter channel list with exactly one entry; `me3000AIMultipleEx` for background operation
`me3000AIQuick`	Sampling a few values from one channel – first enter channel list with at least one entry and all of the same channel; `me3000AIQuickEx` for background operation
`me3000AIStop`	Stops operation in infinite mode of background functions

2.3.1.3 Conversion Based on Channel List

This operating mode is available wit1h all CP370 models. A conversion based on the programmed channel list enables synchronous sampling of the analog input channels in a fixed time frame. It also allows separate parameters for each channel list entry consisting of the channel number, an amplification setting (1, 2, 4, 8)), uni/bipolar operation and single ended or differential operation (must be the same for all channel list entries)! The channel list can have a maximum size of N_{max} = 1024.

The programmable counters can be used as timers. Both counters have fixed functions and before conversion can begin, the timers must be programmed by calling the function `me3000AISetTimer`.

Figure 2-6: Channel List Diagram



- The SCAN-time represents the time difference to process 2 channel lists.
- The CHAN-time represents the sampling rate (the time difference between sampling of two consecutive channel list entries).
- Use the following functions to implement the above described operation mode. Refer to chapter 6.4.3 for a full description of the functions:

`me3000FrequencyToCounter`	Conversion of sample rate for passing to `me3000AISetTimer`
`me3000AISetTimer`	Setting the counters for A/D conversion
`me3000AISetTrigger`	Setting trigger edge and trigger source
`me3000AIClearChannelList`	Clear channel list FIFO
`me3000AIMakeChannelListEntry`	Prepare a channel list entry
`me3000AIAddToChannelList`	Writing a channel list entry to channel list FIFO
`me3000AIScan`	Sampling by channel list; `me3000AIScanEx` for background operation

me3000AIStop	Stops operation in infinite mode of background functions

2.3.1.4 Simultaneous A/D Conversion

CP370-SIM offers four simultaneous sample and hold components for simultaneous sampling. In this mode of operation channels 0 … 3 are sampled simultaneously when the trigger signal is received. Note that all channels must be configured as single ended.

Use the following function to implement the above described operation mode. Refer to chapter 6.4.3 for a full description of the function:

me3000AISetSimultaneous	Enable/disable simultaneous operating mode

2.3.2 D/A Operating Modes

2.3.2.1 Single-Value Output

This operating mode is available to all CP370 models. The value to be converted is written transparently onto the chosen D/A converter and immediately output.

Use the following functions to implement the above described operation mode. Refer to chapter 6.4.4 for a full description of the functions:

me3000AOSetMode	Configuring output range and operating mode “transparent”
me3000AOSingle	Loading single value to D/A converter and output immediately

2.3.2.2 Simultaneous Output

This operating mode is available to all CP370 models. The values to be sent out must be written into the D/A converter by calling the functions me3000AOSetMode and me3000AOSingle several times. By calling the function me3000AOUpdate all channels are updated simultaneously.

Use the following functions to implement the above described operation mode. Refer to chapter 6.4.4 for a full description of the functions:

me3000AOSetMode	Configuring the output range and operating mode “buffered”
me3000AOSingle	Loading single value to D/A converter
me3000AOUpdate	Updating all D/A channels simultaneously
me3000AOReset	Reset all D/A channels

2.3.2.3 Continuous Mode

This operating mode is available with the model CP370-SIG. The D/A-FIFO is continuously loaded with data blocks. When the “FIFO half full” condition is reached, the next data block is requested.

Use the following functions to implement the above described operation mode. Refer to chapter 6.4.4 for a full description of the functions:

`me3000FrequencyToCounter`	Conversion of sample rate for passing to `me3000AOSetTimer`
`me3000AOSetTimer`	Setting the counter for D/A conversion
`me3000AOSetTrigger`	Setting trigger edge and trigger source
`me3000AOSetMode`	Configuring output range, other parameters are ignored
`me3000AOContinuous`	Continuously output with automatic reload of D/A FIFO; `me3000AOContinuousEx` for background operation
`me3000AOStop`	Stops operation of `me3000AOContinuous`
`me3000AOStopEx`	Stops background operation of `me3000AOContinuousEx`

2.3.2.4 Wraparound Mode

The wraparound mode is available with the CP370 - SIG model . The data block is loaded into the D/A-FIFO before the conversion. The individual values are repeatedly output in a cycle; e. g. this mode can be used to generate a periodic wave form.

Use the following functions to implement the above described operation mode. Refer to chapter 6.4.4 for a full description of the functions:

`me3000AOClearDACFIFO`	Clearing DAC FIFO
`me3000AOAddToDACFIFO`	Writing value to DAC FIFO
`me3000FrequencyToCounter`	Conversion of sample rate for passing to `me3000AOSetTimer`
`me3000AOSetTimer`	Setting the counter for D/A conversion
`me3000AOSetTrigger`	Setting trigger edge and trigger source
`me3000AOSetMode`	Configuring output range, other parameters are ignored
`me3000AOWraparound`	Repeated output of D/A FIFO
`me3000AOStop`	Stops operation of `me3000AOWraparound`

2.3.3 Digital I/O Operating Modes

2.3.3.1 Digital I/O

For all CP370 models two 16 bit digital I/O ports are available. For each port, the direction can be independently configured.

Use the following functions to implement the above described operation mode. Refer to chapter 6.4.5 for a full description of the functions:

`me3000DIOSetPortDirection`	Configuring port direction
`me3000DIGetBit`	1-bit input
`me3000DIGetByte`	8-bit wide input
`me3000DIGetWord`	16-bit wide input
`me3000DOSetBit`	1-bit output
`me3000DOSetByte`	8-bit wide output
`me3000DOSetWord`	16-bit wide output
`me3000DOReset`	Set output port to a predefined condition

2.3.3.2 Pulse-Width Modulation

This operating mode is available with model CP370-SIG. In this mode of operation a square wave with a variable duty cycle is output on pin 32. The duty cycle is programmable from 0% to 99% in increments of 1%. The maximum frequency is specified in Table 1-3: “Technical Specifications”.

Use the following functions to implement the operating modes described above. Refer to chapter 6.4.6 for a full description of the functions:

`me3000PWMSetMode`	Configuring duty cycle, frequency and polarity
`me3000PWMStart`	Start PWM output
`me3000PWMStop`	Stop PWM output

2.3.4 Counters

All CP370’s are provided with two counters (16-bit). The count range is user programmable by loading a start and end value into the counter. On the D-sub connector an external clock must be applied, the count direction must be set, and the counter must be enabled.The current count value can always be read out (latched). A typical application example would be a marker system which captures the exact counter value at the moment a marker pulse occurs.

Counting begins at the start value in the given direction. In operating mode “upwards” the current counter value is continuously compared with the end value. When the end

value is reached the counter resets to zero and continues counting. In operating mode “downwards” the current counter value is continuously compared with zero. When the value zero is reached the counter is set to the end value and then continues counting. Upon zero axis crossing an interrupt can be triggered in both directions.



Important!

The maximum frequency of the external clock may not exceed 90 kHz when using an end value less than 65535. If using an end value of exactly 65535 (counter range fully used) the maximum counter frequency is 6 MHz.

Use the following functions to implement the above described operation mode. Refer to chapter 6.4.7 for a full description of the functions:

`me3000CNTInterrupt`	Enable/disable interrupt
`me3000CNTRead`	Reading counter value
`me3000CNTReset`	Reset counter
`me3000CNTSetRange`	Write start and end value to counter
`me3000CNTStart`	Start counter

This page was intentionally left blank.

Chapter 3

Installation

3.1 Board Installation ..... 3 - 3
3.2 Driver Installation ..... 3 - 4
3.2.1 Installation under Windows NT 4.0 ..... 3 - 4
3.2.2 Updating the Board Driver ..... 3 - 5
3.2.3 Changing the CP370 Model ..... 3 - 5
3.3 HP VEE Component Installation ..... 3 - 5
3.4 Uninstall ..... 3 - 6
3.4.1 Uninstall a Single Board ..... 3 - 6
3.4.2 Uninstall the Driver System ..... 3 - 7
3.4.3 Uninstall the HP VEE Components ..... 3 - 7
3.5 Measuring Points and Potentiometers ..... 3 - 7
3.6 Balancing ..... 3 - 8
3.7 Test Program ..... 3 - 9
3.8 PEP Technical Support ..... 3 - 9

This page was intentionally left blank.

3. Installation

As a sophisticated electronic device, your CP370 must be set up and used correctly to maintain safe and proper functioning. Please follow the instructions in this chapter fully to ensure successful installation and long lasting, trouble-free operation.

If several boards are to be installed, please proceed board by board. After installing each board and registering the board with the operating system by running `SETUP.EXE`, run a complete system check before installing the next board. The test program `ME3000.EXE` can be used for this purpose. Once the test has been successfully completed, proceed with the installation of the next board in an analogous way.

3.1 Board Installation



Caution!

If your board type is not specifically qualified as hot-swap capable, please switch off the CompactPCI system before installing the board in a free CompactPCI slot. Failure to do so could endanger your life/health and may damage your board or system.



Note:

Certain CompactPCI boards require bus master and/or rear I/O capability. If you are in doubt whether such features are required for the board you intend to install, please check your specific board and/or system documentation to make sure your system is provided with an appropriate free slot to insert the board.



ESD Equipment!

Your CompactPCI board contains electrostatically sensitive devices. Please observe the necessary precautions to avoid damage to your board:

- Discharge your clothing before touching the assembly. Tools must be discharged before use.
- Do not touch components, connector-pins or traces.
- If working at an anti-static workbench with professional discharging equipment, please do not omit to use it.

The CP370 is factory set and balanced for all modes of operation. Setting and balancing will as a rule not be necessary for a new board. If balancing should be necessary, please refer to section 3.6 for details.

Section 3.8 contains a form. This form should be filled out with all the important information for one or more CP370 boards. The completed form will contain useful reference information such as licence and serial numbers (which may be needed if the CP370 model is changed in the future) as well as information about hardware configuration.

Also note (describe) the physical slot in which the card is installed because the logical board number has no relation to the physical location of the board.

The CP370 library functions use the logical board number in the parameter <BoardNumber> to identify individual CP370 boards. The <BoardNumber> is assigned automatically by the installation program for every CP370. It is shown at the end of the set-up process in the "BoardNumber" window. Make sure to note this on the information form.

Copy the attached form and keep the completed copy with the CP370 handbook. This keeps all the relevant board information available for future reference.

To check the <BoardNumber> after finishing the installation you can use the installation program as is described in the following:

- From the `Start` menu, choose the option `Run`. Under the `Open` option, enter the path for the `SETUP.EXE` file on the disk “Windows NT Driver Disk for CP370” or click `Browse...`
- In the window `Install Options`, choose `Update settings of an installed board` and confirm the selection with `OK`.
- In the window `Installed Boards` the logical board number is shown under `Logical <BoardNumber>`.
- To exit click `CANCEL`.

3.2 Driver Installation

3.2.1 Installation under Windows NT 4.0

To set up your CP370 under Windows NT 4.0 please proceed as follows:

1. Make sure you have your serial and license number ready and then insert the disk “Windows NT Driver Disk for CP370”.
2. Choose the option “Run” from the Windows NT start menu. Under the option “Open” enter the path for the `SETUP.EXE` file on the installation disk provided or click “Browse...”. Confirm your selection and follow the instructions of the setup program which now is being started.
3. From the "Install Options" window choose "Install new board" and confirm your selection with “OK”. The following files will be installed and the corresponding registry entries will be made:

Table 2-1: Installed Files under Windows NT 4.0

File(s)	File Type	Installation Path
`ME3000.SYS`	Kernel driver	`<Windows Directory>\SYSTEM32\DRIVERS>`
`ME3000.DLL`	API DLL	`<Windows Directory>\SYSTEM32`
Files for programming in Visual C++		`<PEP Working Directory>\ME-3000\VC`
Files for programming in Delphi		`<PEP Working Directory>\ME-3000\DELPHI`

Table 2-1: Installed Files under Windows NT 4.0

File(s)	File Type	Installation Path
Files for programming in Visual Basic		<PEP Working Directory>\ME-3000\VBASIC
ME3000.EXE	Test programm	<PEP Working Directory>\ME-3000\TEST

4. Reboot your computer.

3.2.2 Updating the Board Driver

1. Please insert the disk “Windows NT Driver Disk for CP370”.
2. Choose the option “Run” from the start menu. Under the option “Open” enter the path for the SETUP.EXE file on the installation disk provided or click “Browse...”. Confirm your selection and follow the instructions of the installation program which now is being started.
3. In the window "Install Options" choose “Updating driver and language libraries” and confirm your selection with “OK”.

   All system drivers, API DLL as well as high-level language libraries, demo and test programs will all be updated.
4. Reboot your computer.

3.2.3 Changing the CP370 Model

1. Make sure you have your serial and license number ready and then insert the disk “Windows NT Driver Disk for CP370”.
2. Choose the option “Run” from the Windows NT start menu. Under the option “Open” enter the path for the SETUP.EXE file on the installation disk provided or click “Browse...”. Confirm your selection and follow the instructions of the setup program which now is being started.
3. From the "Install Options" window choose "Update Installed board" and confirm your selection with “OK”.
4. Enter your serial number and your new licence key.
5. Reboot your computer

   The driver will check your licence number for validity and then load the appropriate CP370 model.

3.3 HP VEE Component Installation

The HP VEE components of the CP370 driver are on a separate disk. The CP370 is supported by HP VEE versions 4.x or higher.

1. Please insert the disk “HPVEE Driver Disk for CP370”.
2. Choose the option “Run” from the start menu. Under the option “Open” enter the

path for the SETUP.EXE file on the installation disk provided or click “Browse...”. Confirm your selection and follow the instructions of the setup program which now is being started.

The setup program searches your computer for valid HP VEE installations. If more HP VEE versions are found, you can choose to which HP VEE version the components of the CP370 driver should be installed. If at least one valid HP VEE installation was found, the following components will be installed and the corresponding registry entries will be made:

Table 2-2: Installed HP VEE Components

File(s)	File Type	Installation Path
ME3000.DLL	Driver DLL	<HP VEE Root> directory
ME3000VEE.H	Header file	<HP VEE Root> directory
VEE3000.MNU	Menu file	<HP VEE Root> directory (for all PEP boards supported for convenient use of the driver UOs)
ME3000*.VEE	Driver user objects	<HP VEE Root>\PEP\User Objects
XME3000*.VEE	Demo user objects	<HP VEE Root>\PEP\DEMOS

3. Restart HP VEE.

3.4 Uninstall

3.4.1 Uninstall a Single Board

With the help of the install and uninstall program included with the disk “Windows NT Driver Disk for CP370” you can remove individual boards from the Windows registry. The individual components such as the system driver, the application program interface DLL and the high-level language libraries are not removed by using this method.

1. Choose the option “Run” from the start menu. Under the option “Open” enter the path for the SETUP.EXE file on the installation disk provided or click “Browse...”. Confirm your selection and follow the instructions of the installation program which now is being started.
2. From the "Install Options" window, choose “Uninstall a single board” and confirm the selection with “OK”.
3. Select the board you want to remove from the Windows registry.
4. Reboot your computer. The selected board will be removed from the Windows registry.

3.4.2 Uninstall the Driver System

Using the following procedure, the entire CP370 driver system will be removed from your computer. All files will be removed, including the system driver, the application program interface DLL, the high level language libraries and the demo and test programs for all CP370 boards installed in the "CP370" subdirectory of the `C:\PEP` directory, if the default installation options were used.

1. From the Windows Start menu, under the subsequent options "Settings", “Control Panel”, “Add/Remove Programs” in the property page “Install/Uninstall”, choose the “ME-3000 Driver Uninstall” option and confirm the selection with "OK".

   This will remove the entire CP370 Driver System from your computer.

3.4.3 Uninstall the HP VEE Components

1. From the Windows Start menu, under the subsequent options "Settings", “Control Panel”, “Add/Remove Programs” in the property page “Install/Uninstall”, choose the appropriate option, e. g. “ME HP VEE Driver Uninstall 4.0”, and confirm your selection with "OK".

   This will remove the HP VEE components of all boards.

3.5 Measuring Points and Potentiometers

The location of the measuring points (MP) and potentiometers (POT) on the CP 370 are shown on the simplified diagram below:

Figure 3-1: Measuring Points on the CP370

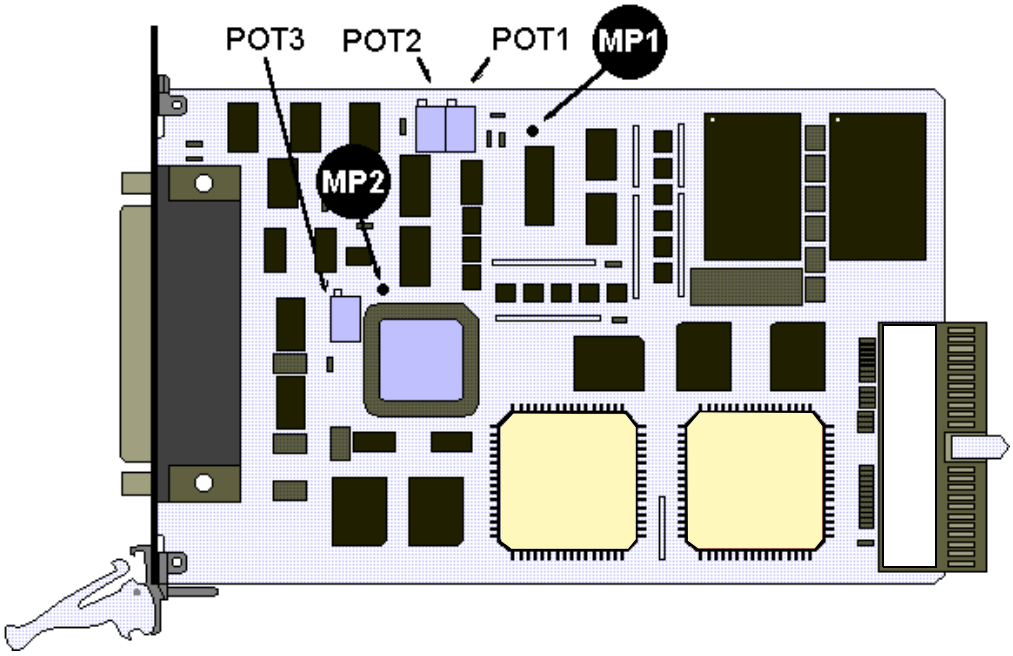


Table 3-3: Potentiometers and Measuring Points of the CP370

Potentiometer/ Measuring Point	Function
POT1	Offset balancing ADC
POT2	Balancing of max. voltage value ADC
POT3	Balancing of reference voltage DAC
MP1	Voltage directly at the A/D-Converter
MP2	Reference voltage of the DAC

3.6 Balancing

The CP 370 is delivered pre-balanced and no balancing should be required on a new board. If balancing of the board should be necessary (for example after use in a very noisy environment), please keep the following points in mind:

In the following description of the balancing procedure, non linear binary values are assumed.

Required Equipment

- DC calibrator;
- high accuracy digital multi-meter (resolution min 0.1 mV)

Balancing the A/D section:

The balancing is done on only one A/D channel (channel 0, pin 39). All other analog input channels (pins 16…19, 36…38, 55…58, 75…78) must be connected to ground of the A/D section (pins 20 or 59).

1. Offset balance:

   - Positive terminal of the DC calibrator is connected to channel 0 (pin 39) and the negative terminal of the DC calibrator is connected to ground of the A/D section (pin 20 or 59).
   - DC calibrator set to exactly 0.000 volts
   - Using the multimeter, measure the voltage between MP1 and the ground of the A/D section (e. g. pin 59)
   - Set the voltage to -0.61 V by adjusting POT1
   - The converted voltage value must be at the transition between 0x000 and 0xFFF.

2. Balancing the maximum voltage value:

   - Positive terminal of the DC calibrator is connected to channel 0 (pin 39) and the negative terminal of the DC calibrator is connected to the ground of the A/D section (pin 20 or 59).
   - Set the DC calibrator to exactly +10.0000 volts.
   - Using the multi-meter, measure the voltage between MP1 and the ground of the A/D section (e. g. pin 59).

- Set the voltage to +2.4981 V by adjusting POT 2.
- The converted voltage value must be at the transition between 0x7FE and 0x7FF.

Balancing the Reference Voltage for the D/A Converter

- Using the digital multi-meter, measure the voltage between MP2 and system ground (e. g. pin 13)
- Set the voltage to exactly +10.0000 V using POT3.

3.7 Test Program

For test issues an extensive test program is provided. The test program `ME3000.EXE` will be automatically installed to the directory `<PEP Working Directory>\ME-3000\TEST` and can be run by double clicking on the file.



Note:

The test utility installation will suceed only if the system driver is correctly installed.

3.8 *PEP* Technical Support

When invoking for technical support by *PEP Modular Computers*, please make sure you have the following information available before calling the support hotline:

To: PEP Modular Computers
CP370 Hotline Service

From: .....
.....

Model	Serial #	Licence #	Slot	Logical Board #
CP370				
CP370-SIM				
CP370-SIG				
.....				

This page was intentionally left blank.

Chapter 4

Configuration

4.1 Jumper Settings ..... 4 - 3
4.2 Pinouts ..... 4 - 3

This page was intentionally left blank.

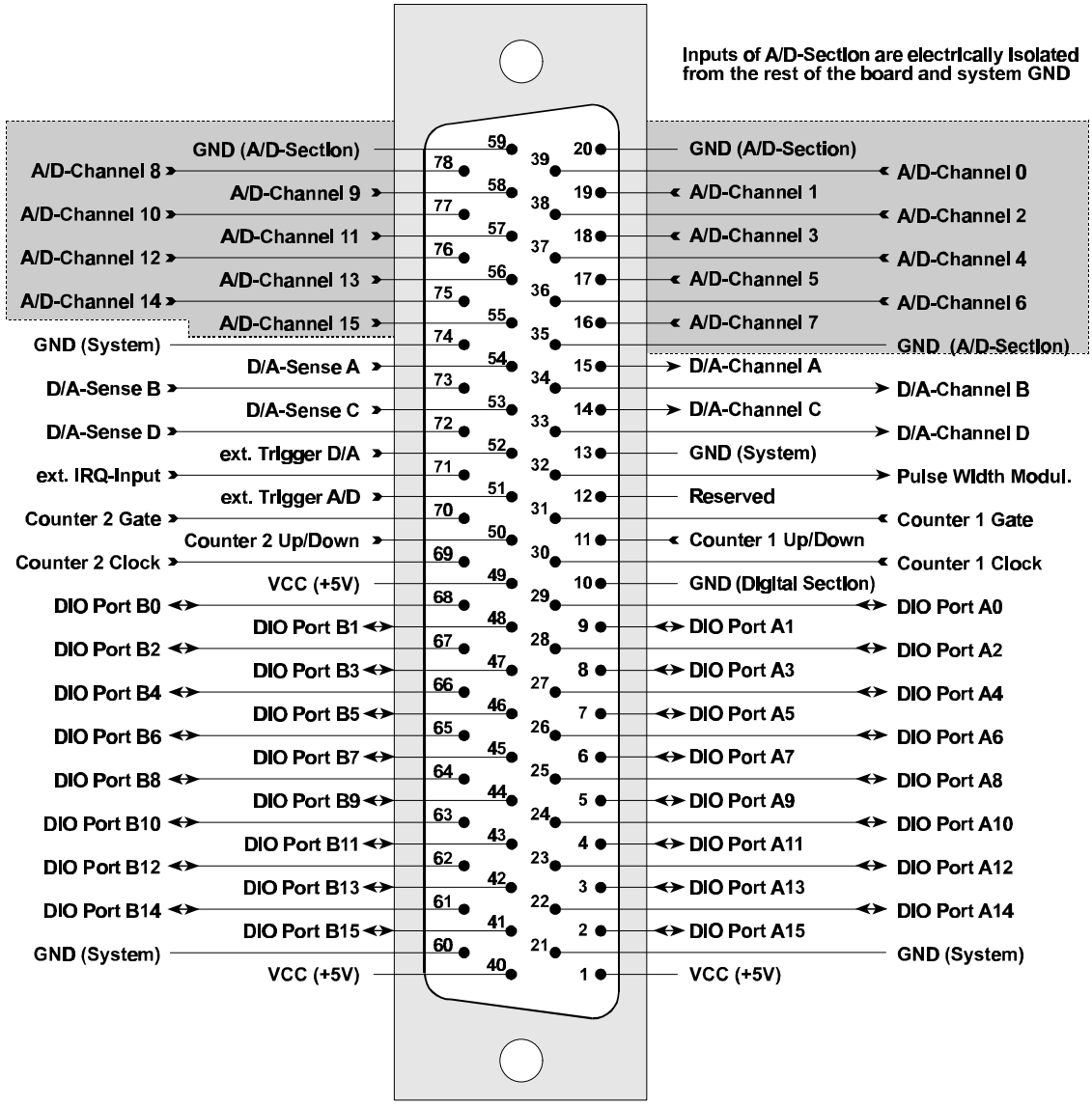
4. Configuration

4.1 Jumper Settings

The CP370 boards comply fully with the Plug and Play philosophy. Since they use Windows NT drivers they do not require any jumpers or other configuration of the boards themselves.

4.2 Pinouts

Figure 4-1: 78-Pin D-Sub Pinouts



This page was intentionally left blank.

Chapter 5

Programming

5.1 High-Level Language Programming ..... 5 - 3
5.1.1 Visual C++ ..... 5 - 3
5.1.2 Delphi ..... 5 - 3
5.1.3 Visual Basic ..... 5 - 3
5.1.4 Example of a Callback Routine ..... 5 - 3
5.1.5 Example Programs for High Level Languages ..... 5 - 4
5.2 HP VEE Programming ..... 5 - 4
5.2.1 The "ME Board" Menu ..... 5 - 4
5.2.2 HP VEE Example Programs ..... 5 - 4
5.2.3 User Objects ..... 5 - 4
5.2.4 Expanded User Objects ..... 5 - 5
5.3 General Rules for the Order of Operation ..... 5 - 8
5.3.1 Example for Analog and Digital I/O Operation ..... 5 - 8

This page was intentionally left blank.

5. Programming

5.1 High-Level Language Programming

For programming in high level languages, the following are required:

- `ME3000.DLL` (Driver DLL in Windows NT system directory)
- Definition and library files appropriate to whatever high level language is used



Important!

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

5.1.1 Visual C++

For programming in Visual C++ (version 4.0 or higher) the header files `COMMON.H`, `ME3000DLL.H` and `ME3000DEFS.H` are required. In these files you find the definitions of the API function calls, predefined constants and macros. Be sure to read the instructions within these files.

5.1.2 Delphi

For programming in Delphi (version 2.0 or higher) the TPU source files `COMMON.PAS`, `ME3000.PAS` and `ME3000DEFS.PAS` are required. In these files you find the definitions of the API function calls and predefined constants. Be sure to read the instructions within these files.

5.1.3 Visual Basic

For programming in Visual Basic (version 4.0 or later) the included file `ME3000.BAS` is required. In this file you will find the definitions of the API function calls and predefined constants. Be sure to read the instructions within this file.

5.1.4 Example of a Callback Routine

Background processes require callback routines for passing data respectively to start an application specific event if an interrupt has been triggered. Such a background process could be started, for example, by the function `me3000AIScanEx`. For examples of callback routines look in the high level language specific subdirectories of `<PEP Working Directory>\ME-3000\.`

- `VC\ME3000EX.C` for Visual C++
- `DELPHI\ME3000EX.PAS` for Delphi
- `VBASIC\ME3000EX.BAS` for Visual Basic

5.1.5 Example Programs for High Level Languages

We have provided simple demo programs with source codes to assist with understanding the functions and how to incorporate them into your work. These demo programs are installed automatically. There are demos for Visual C++, for Delphi and for Visual Basic. These programs will be installed in high level language specific subdirectories of <PEP Working Directory>\ME-3000\. If required, parts of these programs can be easily copied into your project by clipboard.

5.2 HP VEE Programming

HP VEE components for the CP370 are delivered on separate disks. The CP370 is supported by HP VEE full versions 4.x or higher. For basics of HP VEE programming please use your HP VEE documentation and the HP VEE online help index.

5.2.1 The "ME Board" Menu

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

Important!

The UOs delivered depend on the purchased board. Not all UOs for all board families will be available under “ME Board” menu.

5.2.2 HP VEE 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 HP VEE demo programs contains partial additions to the “normal” UOs and for differentiation from the “normal” UOs the prefix "x…" in their file name is used.

5.2.3 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 HP VEE development environment. They can be placed and “wired” in your application the same as standard HP VEE objects.

The User Objects are self descriptive and documented in the chapter “Description of the API functions”; with the exception of the “Expanded User Objects”, described in the next chapter. Additionally you can find a short description of every UO under the item “Edit Description” if you move the cursor over the UO and press the right mouse button.

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

5.2.4 Expanded User Objects

The following is a list of expanded user objects which are described in the sections below:

- me3000DeleteDLL
- me3000DigitToVolt
- me3000AIFormatArray
- me3000LoadDLL
- me3000AICreateChannelList
- me3000StopOnError
- me3000VoltToDigit

5.2.4.1 me3000DeleteDLL

Description

This UO deletes the `ME3000.DLL` from the HP VEE programming environment. This function must be called at the end of every program that uses a driver function from the DLL.

Input

None.

Output

None.

5.2.4.2 me3000DigitToVolt

Description

Converts the measured value from the A/D converter to a voltage value. It can be used for a single value or an array of values.

Input

<Digit>	Measured value from the A/D converter. Must be “wired” to the output terminals 'Value' or 'Buffer' on the UOs *me3000AISingle*, *me3000AIMultiple*, *me3000AIQuick* or *me3000AIScan*
<Input Range>	Menu for selection of input voltage range

Output

<Volt>	Calculated voltage value(s)

5.2.4.3 me3000AIFormatArray

Description

User object is used for all CP370 models.

This user object reformats the array of measured values for easy use, for example, to make a graph. The input array has individual channel list entry measured values one after another. The output array has all values for a given channel list entry after each other. An example of its use can be found in the demo program “Analog In Scan”.

Input

<Buffer>	Array of values for all channels measured. This terminal is “wired” to the output terminal <Buffer> og the UO me3000AIScan
<Size>	Number of the channel list entries
<Count>	Number of channel list processings (input terminal). To be “wired” with the input terminal <Count> from me3000AIScan

Output

<Array>	Array of output values for a particular channel.

5.2.4.4 me3000LoadDLL

Description

Loads the ME3000.DLL. After loading, all driver UOs will be available under the “ME Board” menu.

Input

None.

Output

None.

5.2.4.5 me3000AICreateChannelList

Description

User Object is used for all CP370 models.

The User Object is used to create a channel list on the user panel screen. The function goes into a loop to allow the user to create a channel list and write it to the channel list FIFO. The user panel entries are set to the desired values and then the “Write entry to channel list” button is pushed. The loop repeats “channel list entry” times.



Note!

`This user object calls the functions me3000AIAddToChannelList and me3000AIMakeChannelListEntry so the inputs to these functions must also be set.`

Input

`<Channellist Entries>`	Number of entries in the channel list

Output

None

5.2.4.6 me3000StopOnError

Description

When an error occurs, this function will return an error message in plain text, end the current program, and delete the driver DLL.

Input

`<Error?>`	To be “wired” with the output terminal <Error> of previous user objects

Output

None.

5.2.4.7 me3000VoltToDigit

Description

Converts a voltage value to the required value range of the D/A converter.

Input

<Volt>	Voltage value to be output in Volts
<Output Range>	Menu for selection of output voltage range

Output

<Digit>	Calculated value for output to the D/A converter

5.3 General Rules for the Order of Operation

All analog and digital input and output operations can be used in any order. It is important to note that the configuration for input and output operations must be completed before any I/O operation can begin. In order to optimize speed, logical operation checking is not performed by the driver functions.

5.3.1 Example for Analog and Digital I/O Operation

```
...
//In the following example all functions refer to the board with the
//logical <BoardNumber> “0”
iBoardNumber = 0;
//Configure port A as output
return = me3000DIOSetPortDirection(iBoardNumber, PORTA, MEOUTPUT);
//Basically after every function call an error check by the function
//me3000ErrorMessage should be performed
if (return == 0) then
me3000ErrorMessage("ME3000-Test");
endif;
//Configure port B as input
return = me3000DIOSetPortDirection(iBoardNumber, PORTB, MEINPUT);
//Start A/D conversion by software
return = me3000AISetTrigger(iBoardNumber, AI3000_TRIGGER_TIMER);
//Set scan frequency to 1 kHz
iScan = me3000FrequencyToCounter(1000);
//Set chan frequency to 10 kHz
iChan = me3000FrequencyToCounter(10000);
//Programming timer
return = me3000AISetTimer(iBoardNumber, iScan, iChan);
//Clear complete channel list FIFO;
```

```
return = me3000AIClearChannelList(iBoardNumber);
//Generate channel list with 10 entries

//Channels 1 to 10 will be sampled

for i = 1 to 10
//Generate channel list entry

return = me3000AIMakeChannelListEntry(i, AI3000_PLUS_10,
AI3000_SINGLE_ENDED, Entry);
//Write channel list entry to channel list FIFO

return = me3000AIAddToChannelList(iBoardNumber, Entry);

endfor;

//Settings for D/A conversion: D/A channel number: 2, output level
//range: 0-10V, buffered output

return = me3000AOSetMode(iBoardNumber, 2, AO3000_PLUS_10,
AO3000_BUFFERED);
//Write output value 0xFFF (=10V) to D/A converter

return = me3000AOSingle(iBoardNumber, 2, 0xFFF);
//Now you can run e. g. the following operations in any order:
//Set bit no. 8 of port A to high level

return = me3000DOSetBit(iBoardNumber, PORTA, 8, 1);
//Read the lower byte of port B

return = me3000DIGetByte(iBoardNumber, PORTB, LOW_BYTE, iByteValue);

//Process the channel list with 10 entries two times. Values are
//written to the array <Buffer> and fetched after interrupt

return = me3000AIScan(iBoardNumber, Buffer, 10, 2, AI3000_INTERRUPT);
//Update all D/A channels simultaneously

return = me3000AOUpdate(iBoardNumber);

...
```

This page was intentionally left blank.

Chapter 6

Software

6.1 Functional Driver Overview ..... 6 - 3
6.2 Standards Used in this Section ..... 6 - 3
6.2.1 High-Level Language Functions ..... 6 - 3
6.2.2 HP VEE User Objects ..... 6 - 3
6.3 Definitions ..... 6 - 3
6.4 Description of API Functions ..... 6 - 4
6.4.1 List of Functions ..... 6 - 4
6.4.2 General Functions ..... 6 - 7
6.4.3 Analog Input ..... 6 - 16
6.4.4 Analog Output ..... 6 - 42
6.4.5 Digital I/O ..... 6 - 57
6.4.6 Pulse Width Modulation ..... 6 - 67
6.4.7 Counter ..... 6 - 70
6.4.8 Interrupt Handling ..... 6 - 76
6.4.9 Error Handling ..... 6 - 78

This page was intentionally left blank.

6. Software

6.1 Functional Driver Overview

The 32 bit driver for the CP370 is written for Windows NT 4.0. It consists of the following components:

- Kernel driver `ME3000.SYS` for Windows NT which is automatically loaded upon system start.
- API DLL `ME3000.DLL` with the driver functions for the CP370.

The installation program for the CP370 automatically registers every new CP370 board with the operating system. Upon every start, the driver searches for a registered board and checks the physical existence. After the driver is successfully loaded, the board can be accessed via the API functions.

The API functions allow convenient access to the hardware. Every function that accesses the CP370 requires an integer value for identification of the board. This integer value will be indicated by `<iBoardNumber>` in the following function descriptions

6.2 Standards Used in this Section

6.2.1 High-Level Language Functions

These functions were written specifically for the CP370 board, and they will as a rule begin with the prefix “me3000...”. The function names were selected to be as descriptive as possible. Every function name incorporates several components representing the different function groups.

6.2.2 HP VEE User Objects

User Objects directly calling an API function use the same spelling. Expanded user objects with integrated parameter preprocessing or with auxiliary functions have different spellings. The expanded user objects are described in chapter 5.2.4.

To describe the parameters in the user objects no identifiers for data types have been used.

6.3 Definitions

All predefined constants and macros for Visual C++ will be found in the file `ME3000DEFS.H`, for Delphi in the file `ME3000DEFS.PAS` and for Visual Basic in the file `ME3000.BAS`. By linking the appropriate definition file into your project you can pass many parameters in the form of predefined constants (e. g. MEINPUT, AO_PLUS_10). As an alternative, you can pass the matching Hex value at any time.

6.4 Description of API Functions

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

- General Functions
- Analog Input (including background operation)
- Analog Output (including background operation)
- Digital I/O
- Pulse Width Modulation
- Counter
- Interrupt Handling
- Error Handling

6.4.1 List of Functions

General Functions

me3000FileName ..... 6 - 7
me3000FrequencyToCounter ..... 6 - 8
me3000GeneralReset ..... 6 - 9
me3000GetDLLVersion ..... 6 - 10
me3000GetDriverVersion ..... 6 - 11
me3000GetSerialNumber ..... 6 - 12
me3000GetVECModel ..... 6 - 13
me3000Initialize ..... 6 - 14
me3000SetPriority ..... 6 - 15

Analog Input

me3000AIAddToChannelList ..... 6 - 16
me3000AIClearChannelList ..... 6 - 18
me3000AIMakeChannelListEntry ..... 6 - 19
me3000AIMultiple ..... 6 - 21
me3000AIMultipleEx ..... 6 - 23
me3000AIQuick ..... 6 - 25
me3000AIQuickEx ..... 6 - 27
me3000AIScan ..... 6 - 29
me3000AIScanEx ..... 6 - 31
me3000AISetSimultaneous ..... 6 - 33

me3000AISetTimer ..... 6 - 34
me3000AISetTrigger ..... 6 - 35
me3000AISingle ..... 6 - 37
me3000AISingleEx ..... 6 - 39
me3000AIStop ..... 6 - 41

Analog Output

me3000AOAddToDACFIFO ..... 6 - 42
me3000AOClearDACFIFO ..... 6 - 43
me3000AOContinuous ..... 6 - 44
me3000AOContinuousEx ..... 6 - 46
me3000AOReset ..... 6 - 48
me3000AOSetMode ..... 6 - 49
me3000AOSetTimer ..... 6 - 50
me3000AOSetTrigger ..... 6 - 51
me3000AOSingle ..... 6 - 52
me3000AOStop ..... 6 - 53
me3000AOStopEx ..... 6 - 54
me3000AOUpdate ..... 6 - 55
me3000AOWraparound ..... 6 - 56

Digital I/O

me3000DIOSetPortDirection ..... 6 - 57
me3000DIGetBit ..... 6 - 58
me3000DIGetByte ..... 6 - 60
me3000DIGetWord ..... 6 - 61
me3000DOReset ..... 6 - 62
me3000DOSetBit ..... 6 - 63
me3000DOSetByte ..... 6 - 64
me3000DOSetWord ..... 6 - 66

Pulse Width Modulation

me3000PWMSetMode ..... 6 - 67
me3000PWMStart ..... 6 - 68

me3000PWMStop ..... 6 - 69

Counter

me3000CNTInterrupt ..... 6 - 70
me3000CNTRead ..... 6 - 71
me3000CNTReset ..... 6 - 72
me3000CNTSetRange ..... 6 - 73
me3000CNTStart ..... 6 - 75

Interrupt Handling

me3000DisableExternalInt ..... 6 - 76
me3000EnableExternalInt ..... 6 - 77

Error Handling

me3000ErrorMessage ..... 6 - 78
me3000GetDriverErrorString ..... 6 - 79



6.4.2 General Functions

6.4.2.1 me3000FileName

Description

This function is used for all CP370 models.

This function saves the acquired data to a file. You can use this feature in combination with the functions me3000AIScan and me3000AIScanEx in the operation mode AI3000_SCAN_TO_DISK.

Definitions

C	int me3000FileName(int iBoardNumber, int iType, char *pFile-Name);
Delphi	Function me3000FileName (iBoardNumber, iType: integer; Var ptFileName: tmestring): integer;
Basic	Declare Function me3000FileName Lib "me3000" Alias "_VBme3000FileName@12" (ByVal lBoardNumber As Long, ByVal lType As Long, ByVal pFileName As String) As Long

Parameters

<BoardNumber>	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
<Type>	Reserved; to be set to "0"
<FileName>	String, consisting of the path and file name. Default file name: C:\ME3000_<BoardNumber>.DAT

Return Value

If the function is successfully executed, a "1" is returned. If an error occurs a "0" is returned. The exact cause of the error can be determined by the function me3000ErrorMessage.

6.4.2.2 me3000FrequencyToCounter

Description

This function is used for all CP370 models.

Converts the frequency value [Hz] to an integer value for passing to the function me3000AISetTimer resp. me3000AOSetTimer. Must be called for every parameter (<Scan>, <Chan> resp. <Time>).

Definitions

C	`int me3000FrequencyToCounter(double dFreq);`
Delphi	`Function me3000FrequencyToCounter (dFreq: double): integer;`
Basic	`Declare Function me3000FrequencyToCounter Lib "me3000" Alias "_VBme3000FrequencyToCounter@8" (ByVal dFreq As Double) As Long`
VEE	User object in panel view with a dialog box to edit the `Frequency` and the output terminal `Counter` for the divide factor as the return value.

Parameters

<Freq>	Conversion rate in [Hz]

Return Value

Returns the calculated divide factor as an integer value.

6.4.2.3 me3000GeneralReset

Description

Function used for all CP370 models.

The CP370 hardware will be set to a defined reset condition. The firmware defining the CP370 model stays loaded.

Definitions

C	`int me3000GeneralReset(int iBoardNumber);`
Delphi	`Function me3000GeneralReset (iBoardNumber: integer): integer;`
Basic	`Declare Function me3000GeneralReset Lib "me3000" Alias _VBme3000GeneralReset@4 (ByVal lBoardNumber As Long) As Long`
VEE	User object in panel view with menu to select the `BoardNumber` and the output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.2.4 me3000GetDLLVersion

Description

This function is used for all CP370 models.

Returns the version number of the API-DLL of the CP370.

Definitions

C	`int me3000GetDLLVersion(void);`
Delphi	`Function me3000GetDLLVersion: integer;`
Basic	`Declare Function me3000GetDLLVersion Lib “me3000” Alias "_VBme3000GetDLLVersion@0" ()As Long`
VEE	User object in panel view with output terminal `Version` which returns the DLL version.

Parameters

None.

Return Value

The 32 bit value has the main version in the upper 16 bits and the sub version in the lower 16 bits. For example, a returned value: 0x00010003 for the version 1.03

6.4.2.5 me3000GetDriverVersion

Description

This function is used for all CP370 models.

Returns the version number of the CP370 driver.

Definitions

C	`int me3000GetDriverVersion(char *pBuffer);`
Delphi	`Function me3000GetDriverVersion (Var pBuffer: string): integer;`
Basic	`Declare Function me3000GetDriverVersion Lib "me3000" Alias "_VBme3000GetDriverVersion@4" (ByVal pBuffer As String) As Long`
VEE	User object in panel view with the output terminal `Version` for driver version and `Error` for the return value.

Parameters

`<Buffer>`Pointer to the string containing the driver version

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.2.6 me3000GetSerialNumber

Description

This function is used for all CP370 models.

Returns the serial number of the chosen CP370 board.

Definitions

C	`int me3000GetSerialNumber (int iBoardNumber, int *piSerialNumber)`
Delphi	`Function me3000GetSerialNumber (iBoardNumber: integer; Var piSerialNumber: integer): integer;`
Basic	`Declare Function me3000GetSerialNumber Lib "me3000" Alias "_VBme3000GetSerialNumber@8" (ByVal lBoardNumber As Long, ByRef lSerialNumber As Long) As Long`
VEE	User object in panel view with a menu to select the `BoardNumber` and output terminals for `SerialNumber` and the return value `Error`.

‘Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
`<SerialNumber>`	Pointer to an integer value, containing the serial number.

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.2.7 me3000GetVECModel

Description

This function is used for all CP370 models.

Returns the currently loaded model of the CP370 board.

Definitions

C	`int me3000GetVECModel (int iBoardNumber, int *piMEVECModel)`
Delphi	`Function me3000GetVECModel (iBoardNumber: integer; Var piMEVECModel: integer): integer;`
Basic	`Declare Function me3000GetVECModel Lib "me3000" Alias "_VBme3000GetVECModel@8" (ByVal lBoardNumber As Long, ByRef lMEVECModel As Long) As Long`
VEE	User object in panel view with a menu to select the `BoardNumber` and output terminals for `MEVECModel` and the return value `Error`.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
`<MEVECModel>`	Pointer to an integer value, containing the CP370 model number. Only the lower 2 bits are significant.

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.2.8 me3000Initialize

Description

This function is used for all CP370 models.

Internal variables of the driver and control registers of the CP370 are reset. The firmware defining the CP370 model stays loaded.

Definitions

C	`int me3000Initialize(int iBoardNumber);`
Delphi	`Function me3000Initialize (iBoardNumber: integer): integer;`
Basic	`Declare Function me3000Initialize Lib “me3000” Alias "_VBme3000Initialize@4" (ByVal lBoardNumber As Long) As Long`
VEE	User object in panel view with menu to select the `BoardNumber` and the output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.2.9 me3000SetPriority

Description

This function is used for all CP370 models.

With this function you can set an active background operation to higher or lower priority.

Definitions

C	`int me3000SetPriority (int iBoardNumber, int iPriority);`
Delphi	`Function me3000SetPriority (iBoardNumber, iPriority: integer): integer;`
Basic	`Declare Function me3000SetPriority Lib “me3000” Alias "_VBme3000SetPriority@8" (ByVal lBoardNumber As Long, ByVal lPriority As Long) As Long`

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
`<Priority>`	priority value of the background process on the chosen board

Priority	Priority
`PRIORITY_HIGH`	higher priority
`PRIORITY_LOW`	lower priority

Return Value

Function always returns “1”.

6.4.3 Analog Input

6.4.3.1 me3000AIAddToChannelList

Description

This function is used for all CP370 models.

Generates a channel list entry and writes it into the channel list FIFO. The function `me3000AIMakeChannelListEntry` offers a parameter preparation. The function must be called for each channel list entry.

Definitions

C	`int me3000AIAddToChannelList (int iBoardNumber, int iEntry);`
Delphi	`Function me3000AIAddToChannelList (iBoardNumber, iEntry: integer): integer;`
Basic	`Declare Function me3000AIAddToChannelList Lib "me3000" Alias "_VBme3000AIAddToChannelList@8" (ByVal lBoardNumber AsLong, ByVal lEntry As Long) As Long`
VEE	User object in panel view with a menu to select the `Board No`, the input terminal `Entry` and an output terminal for the return value `Error`.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
`<Entry>`	A channel list entry is passed with this parameter. An entry is created by OR-linking the three variables <Channel>, <Range> and <Mode> (see note below).

`<Channel>`	A/D channel number. Possible values are: • single ended operation: 0..15; • differential operation: 0…7

`<Range>`	Input Range
`AI3000_MINUS_10`	+/- 10V
`AI3000_MINUS_5`	+/- 5V
`AI3000_MINUS_2`	+/-2.5V
`AI3000_MINUS_1`	+/-1.25V

AI3000_PLUS_10	0-10V
AI3000_PLUS_5	0-5V
AI3000_PLUS_2	0-2.5V
AI3000_PLUS_1	0-1.25V
<Mode>	Operating Mode
AI3000_SINGLE_ENDED	single ended mode
AI3000_DIFFERENTIAL	differential mode

The variable <Mode> must be the same for all channel list entries in one channel list!



Note:

Syntax is as shown:

```
Entry = (<Channel> + <Range> + <Mode>);
me3000AIAddToChannelList (BoardNumber, Entry)
```

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function me3000ErrorMessage.

6.4.3.2 me3000AIClearChannelList

Description

This function is used for all CP370 models.

Clears all entries in the channel list FIFO.

Definitions

C	`int me3000AIClearChannelList (int iBoardNumber);`
Delphi	`Function me3000AIClearChannelList (iBoardNumber: integer): integer;`
Basic	`Declare Function me3000AIClearChannelList Lib "me3000" Alias "_VBme3000AIClearChannelList@4" (ByVal lBoardNumber As Long) As Long`
VEE	`User object in panel view with a menu to select the BoardNumber and an output terminal for the return value Error`

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.3.3 me3000AIMakeChannelListEntry

Description

This function is used for all CP370 models.

Prepares and checks the channel list entry, but does not write it into the FIFO. To write the channel list entry into the FIFO, pass the parameter <Entry> to the function me3000AIAddToChannelList. Both functions must be called for each channel list entry.

Definitions

C

```
int me3000AIMakeChannelListEntry (int iChannel,
int iRange, int iMode, short *psEntry);
```

Delphi

```
Function me3000AIMakeChannelListEntry (iChan-
nel: integer; iRange: integer; iMode: integer;
Var psEntry: smallint): integer;
```

Basic

```
Declare Function me3000AIMakeChannelListEntry
Lib “me3000” Alias
"_VBme3000AIMakeChannelListEntry@16" (ByVal
lChannel AsLong, ByVal lRange AsLong, ByVal
lMode AsLong, ByRef iEntry As Integer) As Long
```

Parameters

<Channel> A/D channel number; possible values for single ended operation are 0…15; for differential operation 0…7

<Range> Selection of input voltage range:

<Range>	Input range
AI3000_MINUS_10	± 10V
AI3000_MINUS_5	± 5V
AI3000_MINUS_2	± 2.5V
AI3000_MINUS_1	± 1.25V
AI3000_PLUS_10	0-10V
AI3000_PLUS_5	0-5V
AI3000_PLUS_2	0-2.5V
AI3000_PLUS_1	0-1.25V

<Mode>	Operating mode: single ended or differential; the mode of operation must be the same for all entries!

<Mode>	Operating mode
AI3000_SINGLE_ENDED	single ended mode
AI3000_DIFFERENTIAL	differential mode

<Entry>	With this parameter the content of a channel list entry is passed

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function me3000ErrorMessage.

6.4.3.4 me3000AIMultiple

Description

This function is used for all CP370 models.

This function acquires samples into a buffer of defined size. You can choose between “polling” or “interrupt” operation mode. As soon as “FIFO half full” is reached, an interrupt could be triggered (see also function me3000AIQuick).



Note:

This function may only be called after the functions `me3000AISetTrigger`, `me3000AISetTimer` and `me3000AIClearChannelList` have been successfully run at least once. As well, the functions `me3000MakeChannelListEntry` and `me3000AIAddToChannelList` must have generated a channel list with exactly one entry. Example:

```
if(me3000AISetTrigger(...) && me3000AISetTimer(...) &&
me3000AIClearChannelList(...) &&
me3000AIMakeChannelListEntry(...) &&
me3000AIAddToChannelList(...)) then

me3000AIMultiple(...);
endif;
```

Definitions

C	`int me3000AIMultiple (int iBoardNumber, short *psBuffer, int iCount, int iMode);`
Delphi	`Function me3000AIMultiple (iBoardNumber: integer; Var psBuffer: smallint; iCount, iMode: integer): integer;`
Basic	`Declare Function me3000AIMultiple Lib “me3000” Alias "_VBme3000AIMultiple@16" (ByVal lBoardNumber As Long, ByRef piBuffer As Integer, ByVal lCount As Long, ByVal lMode As Long,) As Long`
VEE	User object in panel view with menus to select `BoardNumber` and `Mode`, a dialog box `Count` for the number of channel list processings, as well the output terminal `Buffer` for the array of sampled values and `Error` for the return value.

Parameters

<BoardNumber>	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
<Buffer>	Pointer to the array of measured values. Only the lowest 12 bits are significant and contain the linearized measured values. A decimal value of 0 is the lowest value and a decimal value of 4095 is the maximum value. Note that the voltage value represented by <Buffer> depends on the input range selected. The buffer size is defined by: <Count>
<Count>	Number of values to be acquired.
<Mode>	Operating mode:

<Mode>	Operating mode
AI3000_POLLING	Continuously request whether FIFO is half full
AI3000_INTERRUPT	Interrupt on FIFO half full

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function me3000ErrorMessage.

6.4.3.5 me3000AIMultipleEx

Description

This function is used for all CP370 models.

Version of me3000AIMultiple for background operation. If <Loops> is passed with AI3000_INFINITE, the background operation is stopped by calling the function me3000AIStop.



Note:

This function may only be called after the functions me3000AISetTrigger, me3000AISetTimer and me3000AIClearChannelList have been successfully run at least once. As well, the functions me3000MakeChannelListEntry and me3000AIAddToChannelList must have made a channel list with exactly one entry. Example:

```
if (me3000AISetTrigger(...) &&
me3000AISetTimer(...) &&
me3000AIClearChannelList(...) &&
me3000AIMakeChannelListEntry(...) &&
me3000AIAddToChannelList(...)) then

me3000AIMultipleEx(...);
endif;
```

Definitions

C

```
int me3000AIMultipleEx (int iBoardNumber, short
*psBuffer, int iCount, int iMode, int iLoops,
int iDelay, P_1_PROC pCallback);
```

Delphi

```
Function me3000AIMultipleEx (iBoardNumber,
iCount, iMode, iLoops, iDelay: integer; Var
psBuffer: smallint; pCallback: P_1_PROC): inte-
ger;
```

Basic

```
Declare Function me3000AIMultipleEx Lib
“me3000” Alias "_VBme3000AIMultipleEx@26"
(ByVal lBoardNumber As Long, ByVal lCount As
Long, ByVal lMode As Long, ByVal lLoops As
Long, ByVal lDelay As Long, ByRef piBuffer As
Integer, By Val pCallback As Long) As Long
```

Parameters

<BoardNumber>	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
<Buffer>	Pointer to the array of measured values. Only the lowest 12 bits are significant and contain the linearized measured values. A decimal value of 0 is the lowest value and a decimal value of 4095 is the maximum value. Note that the voltage value represented by <Buffer> depends on the input range selected. The buffer size is defined by: <Count>
<Count>	Number of values to be acquired
<Mode>	Operating mode:

<Mode>	Operating mode
AI3000_POLLING	Continuously request whether FIFO is half full
AI3000_INTERRUPT	Interrupt on FIFO half full

<Loops>	Number of repeats. For an infinite loop use the constant AI3000_INFINITE
<Delay>	Delay time between 2 repeats in [ms]
<Callback>	Pointer to the user defined call back routine

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function me3000ErrorMessage.

6.4.3.6 me3000AIQuick

Description

This function is used for all CP370 models.

This function acquires samples into a buffer of defined size. You can choose between “polling” or “interrupt” operation mode. This function utilizes the end of channel list processing as an interrupt event. This makes sense if only a few values are acquired. It gives you the ability to react quickly to the end of A/D conversion. In comparison the functions me3000AIMultiple and me3000AIScan have to wait for “FIFO half full”. In me3000AIQuick the number of values is restricted by the size of the channel list FIFO.



Note:

This function may only be called after the functions me3000AISetTrigger, me3000AISetTimer and me3000AIClearChannelList have been successfully run at least once. As well, the functions me3000MakeChannelListEntry and me3000AIAddToChannelList must have made a channel list with at least one entry (always sampling the same channel). Example

```
if(me3000AISetTrigger(...) &&
me3000AISetTimer(...) &&
me3000AIClearChannelList(...)!=0) then

for(i=0; i<Count; i++)
me3000AIMakeChannelListEntry(...)
me3000AIAddToChannelList(...)
endfor;

me3000AIQuick(...);
endif;
```

Definitions

C	`int me3000AIQuick (int iBoardNumber, short *psBuffer, int iCount, int iMode);`
Delphi	`Function me3000AIQuick (iBoardNumber: integer; Var psBuffer: smallint; iCount, iMode: integer): integer;`
Basic	`Declare Function me3000AIQuick Lib “me3000” Alias "_VBme3000AIQuick@16" (ByVal lBoardNumber As Long, ByRef piBuffer As Integer, ByVal lCount As Long, ByVal lMode As Long) As Long`

VEE

User object in panel view with menus to select `BoardNumber` and `Mode`, a dialog box for `BufferSize`, which is calculated by number of channel list entries `x Count`, a dialog box `Count` for the number of channel list processings and the output terminal `Buffer` for the sampled values and `Error` for the return value.

Parameters

`<BoardNumber>`

Board number for the addressed CP370 among boards 1...16. Possible values: 0…15

`<Buffer>`

Pointer to the array of measured values. Only the lowest 12 bits are significant and contain the linearized measured values. A decimal value of 0 is the lowest value and a decimal value of 4095 is the maximum value. Note that the voltage value represented by `<Buffer>` depends on the input range selected. The buffer size is defined by: `<Count> x` number of channel list entries

`<Count>`

Number of times to process the channel list
(for <Count> = 1 no polling operation possible)

`<Mode>`

Operating mode:

`<Mode>`	Operating mode
`AI3000_POLLING`	Continuously request whether channel list processing ended
`AI3000_INTERRUPT`	Interrupt at the end of channel list processing

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage.`

6.4.3.7 me3000AIQuickEx

Description

This function is used for all CP370 models.

Version of me3000AIQuick for background operation. If <Loops> is passed with AI3000_INFINITE, the background operation is stopped by calling the function me3000AIStop.



Note:

This function may only be called after the functions me3000AISetTrigger, me3000AISetTimer and me3000AIClearChannelList have been successfully run at least once. As well, the functions me3000MakeChannelListEntry and me3000AIAddToChannelList must have made a channel list with at least one entry (always sampling the same channel). Example:

```
if(me3000AISetTrigger(...) &&
me3000AISetTimer(...) &&
me3000AIClearChannelList(...)!=0) then

for(i=0; i<Count; i++)
me3000AIMakeChannelListEntry(...)
me3000AIAddToChannelList(...)
endfor;

me3000AIQuickEx(...);
endif;
```

Definitions

C

```
int me3000AIQuickEx (int iBoardNumber, short
*psBuffer, int iCount, int iMode, int iLoops,
int iDelay, P_1_PROC pCallback);
```

Delphi

```
Function me3000AIQuickEx (iBoardNumber: inte-
ger; Var psBuffer: smallint; iCount, iMode,
iLoops, iDelay: integer; pCallback: P_1_PROC):
integer;
```

Basic

```
Declare Function me3000AIQuickEx Lib “me3000”
Alias "_VBme3000AIQuickEx@26" (ByVal lBoardNum-
ber As Long, ByRef piBuffer As Integer, ByVal
lCount As Long, ByVal lMode As Long, ByVal
lLoops As Long, ByVal iDelay As Long, ByVal
pCallback As Long) As Long
```

Parameters

<BoardNumber>	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
<Buffer>	Pointer to the array of measured values. Only the lowest 12 bits are significant and contain the linearized measured values. A decimal value of 0 is the lowest value and a decimal value of 4095 is the maximum value. Note that the voltage value represented by <Buffer> depends on the input range selected. The buffer size is defined by: <Count> x number of channel list entries
<Count>	Number of times to process the channel list (for <Count> = 1 no polling operation is possible)
<Mode>	Operating mode:

<Mode>	Operating mode
AI3000_POLLING	Continuously request whether channel list processing ended
AI3000_INTERRUPT	Interrupt at the end of channel list processing

<Loops>	Number of repeats. For an infinite loop use the constant AI3000_INFINITE
<Delay>	Delay time between 2 repeats in [ms]
<Callback>	Pointer to the user defined call back routine

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function me3000ErrorMessage.

6.4.3.8 me3000AIScan

Description

This function is used for all CP370 models.

Starts processing of the channel list according to the operation mode passed by the <Mode> parameter. The values are passed after the conversion process is complete. If you wish to write the data into a file use the mode AI3000_SCAN_TO_DISK.



Note:

This function may only be called after the functions me3000AISetTrigger, me3000AISetTimer and me3000AIClearChannelList have been successfully run at least once. As well, the functions me3000MakeChannelListEntry and me3000AIAddToChannelList must generate and load the channel list. Example:

```
if(me3000AISetTrigger(...) &&
me3000AISetTimer(...) &&
me3000AIClearChannelList(...)!=0) then

for(i=0; i<Count; i++)
me3000AIMakeChannelListEntry(...)
me3000AIAddToChannelList(...)
endfor;

me3000AIScan(...);
endif;
```

Definitions

C	`int me3000AIScan (int iBoardNumber, short *psBuffer, int iSize, int iCount, int iMode);`
Delphi	`Function me3000AIScan (iBoardNumber: integer; Var psBuffer: smallint; iSize, iCount, iMode: integer;): integer;`
Basic	`Declare Function me3000AIScan Lib “me3000” Alias "_VBme3000AIScan@20" (ByVal lBoardNumber As Long, ByRef piBuffer As Integer, ByVal lSize As Long, ByVal lCount As Long, ByVal lMode As Long,) As Long`
VEE	User object in panel view with menus to select `BoardNumber` and `Mode`, a dialog box `Count` for the number of channel list processings, a dialog box `Size` for the number of channel list entries and the output terminals `Buffer` for the array of sampled values and `Error` for the return value.

Parameters

<BoardNumber>	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
<Buffer>	Pointer to the array of measured values. Only the lowest 12 bits are significant and contain the linearized measured values. A decimal value of 0 is the lowest value and a decimal value of 4095 is the maximum value. Note that the voltage value represented by <Buffer> depends on the input range selected. The buffer size is defined by: <Count> x <Size>
<Size>	Number of channel list entries
<Count>	Number of times to process the channel list(if <Count> = 1 no polling operation possible)
<Mode>	Operating mode:

<Mode>	Operating mode
AI3000_POLLING	Continuously request whether channel list processing ended or FIFO is half full
AI3000_INTERRUPT	Interrupt at the end of channel list processing or at A/D-FIFO half full
AI3000_SCAN_TO_DISK	By default writes data to the file C:\ME3000_ <Board-Number>.DAT, which will be automatically generated (see also me3000FileName). Make sure that there is enough space on your hard drive for all the data. This constant must be logically ORed with one of the constants listed above.

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function me3000ErrorMessage.

6.4.3.9 me3000AIScanEx

Description

This function is used for all CP370 models.

Version of me3000AIScan for background operation. In operation mode AI3000_SCAN_INFINITE, or if <Loops> is passed with AI3000_INFINITE, the background operation is stopped by calling the function me3000AIStop.



Note:

This function may only be called after the functions me3000AISetTrigger, me3000AISetTimer and me3000AIClearChannelList have been successfully run at least once. As well, the functions me3000MakeChannelListEntry and me3000AIAddToChannelList must generate and load the channel list. Example:

```
if(me3000AISetTrigger(...) &&
me3000AISetTimer(...) &&
me3000AIClearChannelList(...)!=0) then

for(i=0; i<Count; i++)
me3000AIMakeChannelListEntry(...)
me3000AIAddToChannelList(...)
endfor;

me3000AIScanEx(...);
endif;
```

Be aware that using the mode AI3000_SCAN_INFINITE can cause data to accumulated very quickly and could surpass the available memory in your system. In order to optimize the speed of the routine, there is no check for this built in. (e. g.: a sampling rate of 20 kHz will require 24 MB of memory if left running for 10 minutes)

Definitions

C

```
int me3000AIScanEx (int iBoardNumber, short
*psBuffer, int iSize, int iCount, int iMode,
int iLoops, int iDelay, P_1_PROC pCallback);
```

Delphi

```
Function me3000AIScanEx (iBoardNumber: integer;
Var psBuffer: smallint; iSize, iCount, iMode,
iLoops, iDelay: integer; pCallback: P_1_PROC):
integer;
```

Basic

```
Declare Function me3000AIScanEx Lib "me3000" Alias
"_VBme3000AIScanEx@30" (ByVal lBoardNumber As Long,
ByRef piBuffer As Integer, ByVal iSize As Long, ByVal iCount
As Long, ByVal iMode As Long, ByVal iLoops As Long, ByVal
iDelay As Long, ByVal pCallback As Long) As Long
```

Parameters

<BoardNumber>	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
<Buffer>	Pointer to the array of measured values. Only the lowest 12 bits are significant and contain the linearized measured values. A decimal value of 0 is the lowest value and a decimal value of 4095 is the maximum value. Note that the voltage value represented by <Buffer> depends on the input range selected. The buffer size is defined by: <Count> x <Size>
<Size>	Number of channel list entries
<Count>	Number of times to process the channel list (if <Count> = 1 no polling operation is possible)
<Mode>	Operating mode:

<Mode>	Operating mode
AI3000_POLLING	Continuously request whether channel list processing ended or FIFO is half full
AI3000_INTERRUPT	Interrupt at the end of channel list processing or at A/D-FIFO half full
AI3000_SCAN_INFINITE	The channel list is processed until the function me3000AIStop is called
AI3000_SCAN_TO_DISK	By default writes data to the file C:\ME3000_<BoardNumber>.DAT, which will be automatically generated (see also me3000FileName). Make sure that there is enough space on your hard drive for all the data. This constant must be logically ORed with one of the constants listed above.

<Loops>	Number of repeats. For an infinite loop use the constant AI3000_INFINITE
<Delay>	Delay time between 2 repeats in [ms]
<Callback>	Pointer to the user defined call back routine



Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function me3000ErrorMessage.

6.4.3.10 me3000AISetSimultaneous

Description

This function is used for the CP370-SIM boards provided with the optional “sample & hold” device.

Function to configure the board for simultaneous sampling of the A/D channels 0…3.

Definitions

C	`int me3000AISetSimultaneous (int iBoardNumber, int iMode);`
Delphi	`Function me3000AISetSimultaneous (iBoardNumber: integer; iMode: integer): integer;`
Basic	`Declare Function me3000AISetSimultaneous Lib “me3000” Alias "_VBme3000AISetSimultaneous@8" (ByVal lBoardNumber As Long, lMode As Long) As Long`
VEE	User object in panel view with menus to select `BoardNumber` and `Mode` and the output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
`<Mode>`	Configuring the board for simultaneous sampling

`<Mode>`	Operating mode
`AI3000_ENABLE`	enable simultaneous mode
`AI3000_DISABLE`	disable simultaneous mode

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.3.11 me3000AISetTimer

Description

This function is used for all CP370 models.

Configures the counters used as timers for the A/D conversion. The function me3000FrequencyToCounter allows easy conversion to pass the timer values to the function me3000AISetTimer.

Note:

The start frequency for SCAN- and CHAN-time is fixed to 3 MHz. Note that SCAN-Time must be equal to or greater than CHAN-time x number of channel list entries. For the max. sampling rate, please refer to section 1.5.

Definitions

C	`int me3000AISetTimer (int iBoardNumber, int iScan, int iChan);`
Delphi	`Function me3000AISetTimer (iBoardNumber, iScan, iChan: integer): integer;`
Basic	`Declare Function me3000AISetTimer Lib “me3000” Alias "_VBme3000AISetTimer@12" (ByVal lBoard-Number As Long, ByVal lScan As Long, ByVal lChan As Long) As Long`
VEE	User object in panel view with menus to select `BoardNum-ber`, dialog boxes for the divide factors `Scan` and `Chan` and the output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
`<Scan>`	Divide factor for the 32 bit SCAN-Timer that determines the time between two channel list processings. The start frequency is fixed to 3 MHz. With divide factors between 0x0C…0xFFFFFFFF you can realize SCAN-times between 4 µs and approx. 23 minutes. If you intend to use only the CHAN time, set the value to zero.
`<Chan>`	Divide factor for the 16 bit CHAN-timer that determines the sampling rate. The start frequency is fixed to 3 MHz. With divide factors between 0x06…0xFFFF you can realize CHAN-times between 2 µs and approx. 22 ms.



Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.3.12 me3000AISetTrigger

Description

This function is used for all CP370 models.

Determines the condition for the conversion start. Trigger source and rising or falling edge can be set. This function has to be called before the me3000AIMultiple, `me3000AIQuick` or `me3000AIScan` and the corresponding background functions.

Definitions

C	`int me3000AISetTrigger (int iBoardNumber, int iModePolarity);`
Delphi	`Function me3000AISetTrigger (iBoardNumber, iModePolarity: integer): integer;`
Basic	`Declare Function me3000AISetTrigger Lib “me3000” Alias "_VBme3000AISetTrigger@8" (ByVal lBoardNumber As Long, ByVal lModePolarity As Long) As Long`
VEE	User object in panel view with menus to select `BoardNumber` and `ModePolarity` as well as the output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
`<ModePolarity>`	Used to set the condition for conversion start by configuring trigger source and trigger edge.

The trigger modes depend on the function used and the parameters `<Mode>` and `<Count>`. See the following table for the valid trigger modes and note the OR-linking, if continuous conversion is used.

Table 5-1: Trigger Modes

Functions	me3000AIScan	me3000AIScan	me3000AIScan	me3000AIScan	me3000AIScanEx	me3000AIScanEx	me3000AIScanEx	me3000AIScanEx	me3000AIMultiple	me3000AIMultipleEx	me3000AIQuick	me3000AIQuickEx
<Mode>	AI3000_INTERRUPT	AI3000_POLLING	AI3000_INTERRUPT	AI3000_POLLING	AI3000_INTERRUPT	AI3000_POLLING	AI3000_INTERRUPT	AI3000_POLLING				
<Count>	= 1	= 1	> 1	> 1	= 1	= 1	> 1	> 1				
Start by software: <ModePolarity> = AI3000_TRIGGER_TIMER	X	X	X	X	X	X	X	X	X	X	X	X
Single start by ext. Trigger, falling edge: <ModePolarity> = AI3000_TRIGGER_EXT_LOW	X	—	—	—	X	—	—	—	—	—	X	X
Single start by ext. Trigger, rising edge: <ModePolarity> = AI3000_TRIGGER_EXT_HIGH	X	—	—	—	X	—	—	—	—	—	X	X
Continuous start by ext. trigger, falling edge: <ModePolarity> = AI3000_TRIGGER_EXT_LOW + AI3000_TRIGGER_CONTI	—	—	X	X	—	—	X	X	X	X	—	—
Continuous start by ext. trigger, rising edge: <ModePolarity> = AI3000_TRIGGER_EXT_HIGH + AI3000_TRIGGER_CONTI	—	—	X	X	—	—	X	X	X	X	—	—

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.3.13 me3000AISingle

Description

This function is used for all CP370 models.

Does a single conversion and delivers the result. The function call directly starts the conversion process. There is no interrupt or external trigger support from this function. Configuration is not required before the function is called.

Definitions

C	`int me3000AISingle (int iBoardNumber, int iChannel, int iMode, short *psValue);`
Delphi	`Function me3000AISingle (iBoardNumber, iChannel, iMode: integer; Var psValue: smallint): integer;`
Basic	`Declare Function me3000AISingle Lib “me3000” Alias "_VBme3000AISingle@16" (ByVal lBoardNumber As Long, ByVal lChannel As Long, ByVal lMode As Long, ByRef piValue As Integer) As Long`
VEE	User object in panel view with menus to select `BoardNumber`, `ChannelNumber`, `Range` and `Mode` as well as the output terminals `Value` for the measured value and `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
`<Channel>`	A/D channel number; possible values for single ended operation are 0…15; for differential operation 0…7
`<Mode>`	Input voltage range; the variable <Range> has to be logically ORed with the variable <Mode>.

<Range>	Input range
`AI3000_MINUS_10`	± 10V
`AI3000_MINUS_5`	± 5V
`AI3000_MINUS_2`	± 2.5V
`AI3000_MINUS_1`	± 1.25V
`AI3000_PLUS_10`	0-10V
`AI3000_PLUS_5`	0-5V

	AI3000_PLUS_2	0-2.5V
	AI3000_PLUS_1	0-1.25V
	<Mode>	Operating mode
	AI3000_SINGLE_ENDED	Single ended mode
	AI3000_DIFFERENTIAL	Differential mode
<Value>	Pointer to a 16-bit value. Only the lowest 12 bits are significant and contain the linearized measured value. A decimal value of 0 is the lowest value and a decimal value of 4095 is the maximum value.	



Note:

Syntax is as shown:

```
me3000AISingle (BoardNumber, Channel, <Range> + <Mode>, Value);
```

Example: channel 5 of board 2 with input range ±5 V should be sampled in differential mode. The syntax is:

```
me3000AISingle (0x02, 0x05, AI3000_MINUS_5 +
AI3000_DIFFERENTIAL, pValue);
```

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.3.14 me3000AISingleEx

Description

This function is used for all CP370 models.

Version of me3000AISingle for background operation. If <Loops> is passed with AI3000_INFINITE, the background operation is stopped by calling the function me3000AIStop.

Definitions

C

```
int me3000AISingleEx (int iBoardNumber, int iChannel, int iMode, int iLoops, int iDelay, short *psValue, PSHORT_PROC pCallback);
```

Delphi

```
Function me3000AISingleEx (iBoardNumber, iChannel, iMode, iLoops, iDelay: integer; Var psValue: smallint; pCallback: PSHORT_PROC): integer;
```

Basic

```
Declare Function me3000AISingleEx Lib “me3000” Alias "_VBme3000AISingleEx@26" (ByVal lBoardNumber As Long, ByVal lChannel As Long, ByVal lMode As Long, ByVal lLoops As Long, ByVal lDelay As Long, ByRef piValue As Integer, ByVal pCallback As Long) As Long
```

Parameters

<BoardNumber> Board number for the addressed CP370 among boards 1...16. Possible values: 0…15

<Channel> A/D channel number; possible values for single ended operation are 0…15; for differential operation 0…7

<Mode> Input voltage range; the variable <Range> has to be logically ORed with the variable <Mode>.

<Range>	Input range
AI3000_MINUS_10	± 10V
AI3000_MINUS_5	± 5V
AI3000_MINUS_2	± 2.5V
AI3000_MINUS_1	± 1.25V
AI3000_PLUS_10	0-10V
AI3000_PLUS_5	0-5V
AI3000_PLUS_2	0-2.5V

	AI3000_PLUS_1	0-1.25V
	<Mode>	Operating mode
	AI3000_SINGLE_ENDED	single ended mode
	AI3000_DIFFERENTIAL	differential mode
<Loops>	Number of repeats. For an infinite loop use the constant AI3000_INFINITE	
<Delay>	Delay time between 2 repeats in [ms]	
<Value>	Pointer to a 16-bit value. Only the lowest 12 bits are significant and contain the linearized measured value. A decimal value of 0 is the lowest value and a decimal value of 4095 is the maximum value.	
<Callback>	Pointer to the user defined call back routine	

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function me3000ErrorMessage.

6.4.3.15 me3000AIStop

Description

This function is used for all CP370 models.

Calling this function will terminate operation of the following background functions: me3000AIMultipleEx, me3000AIQuickEx, me3000AIScanEx and me3000AISingleEx.

Definitions

C	`int me3000AIStop (int iBoardNumber);`
Delphi	`Function me3000AIStop (iBoardNumber: integer): integer;`
Basic	`Declare Function me3000AIStop Lib “me3000” Alias "_VBme3000AIStop@4" (ByVal lBoardNumber As Long) As Long`

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.4 Analog Output

6.4.4.1 me3000AOAddToDACFIFO

Description

This function is used for the CP370-SIG only.

Calling this function will write a single value directly into the D/A FIFO.

Definitions

C	`int me3000AOAddToDACFIFO (int iBoardNumber, int iEntry);`
Delphi	`Function me3000AOAddToDACFIFO (iBoardNumber, iEntry: integer): integer;`
Basic	`Declare Function me3000AOAddToDACFIFO Lib "me3000" Alias "_VBme3000AOAddToDACFIFO@8" (ByVal lBoardNumber As Long, ByVal lEntry As Long) As Long`
VEE	User object in panel view with menus to select `BoardNumber` and `ChannelNumber`, a dialog box `Value` to pass the output value as well as an output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15	
`<Entry>`	The 14 bit wide entry consists of a 12 bit voltage value and 2 bits for the D/A channel number. The voltage value occupies bits 0...11, and the channel number bits 12 and 13. To create a valid entry the channel number value must be shifted left 12 times and then logically ORed with the voltage value. (see also the note).	
	`<Value>`	12 bit value in the range 0x000 (min) to 0xFFF (maximum value of the output range); see also `me3000AOSetMode`.
	`<Channel>`	D/A channel number; possible values: 0…3



Note:

The syntax is:

```
Entry = ((Channel << 12) + (Value & 0xFFF))
me3000AOAddToDACFIFO (BoardNumber, Entry);
```

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.4.2 me3000AOClearDACFIFO

Description

This function is used for the CP370-SIG model only.

Calling this function will delete the entire content of the D/A FIFO.

Definitions

C	`int me3000AOClearDACFIFO (int iBoardNumber);`
Delphi	`Function me3000AOClearDACFIFO (iBoardNumber: integer): integer;`
Basic	`Declare Function me3000AOClearDACFIFO Lib “me3000” Alias "_VBme3000AOClearDACFIFO@4" (ByVal lBoardNumber As Long) As Long`
VEE	User object in panel view with menue to select `BoardNumber` and an output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.4.3 me3000AOContinuous

Description

This function is used for the CP370-SIG model only.

Calling this function will continuously output analog voltage values. The D/A-FIFO is automatically reloaded on “FIFO half full”.



Note:

This function may only be called after the functions `me3000AOSetTrigger`, `me3000AOSetTimer` and `me3000AOSetMode` have been successfully run at least once. Additionally, the user must write the values in an output buffer before running. The functions `me3000AOAddToDACFIFO` and `me3000AOClearDACFIFO` are not necessary in this operation mode. Example:

```
if(me3000AOSetTrigger(...) &&
me3000AOSetTimer(...) &&
me3000AOSetMode(...)!=0) then

me3000AOContinuous(...)
endif;
```

Definitions

C	`int me3000AOContinuous (int iBoardNumber, int iMode, int iSize, short *psBuffer);`
Delphi	`Function me3000AOContinuous (iBoardNumber, iMode, iSize: integer; psBuffer: smallint): integer;`
Basic	`Declare Function me3000AOContinuous Lib “me3000” Alias "_VBme3000AOContinuous@16" (ByVal lBoardNumber As Long, ByVal lMode As Long, ByVal lSize As Long, ByVal piBuffer As Integer) As Long`
VEE	User object in panel view with menus to select `BoardNumber` and `Mode`, a dialog box `Size` for the size of the output buffer, an input terminal `Buffer` to the array of values to be output and the output terminal `Error` for the return value.

Parameters

<BoardNumber>	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15	
<Mode>	Operating mode:	
	<Mode>	Operating mode
	AO3000_POLLING	Permanent request whether FIFO is half full
	AO3000_INTERRUPT	Interrupt if FIFO is half full
<Size>	Size of the output buffer in words	
<Buffer>	Pointer to the output buffer with the values to be output	

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function me3000ErrorMessage.

6.4.4.4 me3000AOContinuousEx

Description

This function is used for the CP370-SIG model only.

Version of me3000AOContinuous for background operation. If <Loops> is passed with AO3000_INFINITE, the background operation is stopped by calling the function `me3000AOStopEx`.



Note:

This function may only be called after the functions `me3000AOSetTrigger`, `me3000AOSetTimer` and `me3000AOSetMode` have been successfully run at least once. As well the user must write the values in an output buffer before running. The functions `me3000AOAddToDACFIFO` and `me3000AOClearDACFIFO` are not necessary in this operation mode. Example:

```
if(me3000AOSetTrigger(...) &&
me3000AOSetTimer(...) &&
me3000AOSetMode(...)!=0) then

me3000AOContinuousEx(...)
endif;
```

Definitions

C

```
int me3000AOContinuousEx (int iBoardNumber, int
iMode, int iSize, short *psBuffer, int iLoops,
int iDelay, PVOID_PROC pCallback);
```

Delphi

```
Function me3000AOContinuousEx (iBoardNumber,
iMode, iSize: integer; psBuffer: smallint;
iLoops, iDelay: integer; pCallback:
PVOID_PROC): integer;
```

Basic

```
Declare Function me3000AOContinuousEx Lib
“me3000” Alias "_VBme3000AOContinuousEx@28"
(ByVal lBoardNumber As Long, ByVal lMode As
Long, ByVal lSize As Long, ByVal piBuffer As
Integer, ByVal lLoops As Long, ByVal lDelay As
Long, ByVal pCallback As Long,) As Long
```

Parameters

<BoardNumber>	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15	
<Mode>	Operating mode:	
	<Mode>	Operating mode
	AO3000_POLLING	Continuously request whether FIFO is half full
	AO3000_INTERRUPT	Interrupt if FIFO is half full
<Size>	Size of the output buffer in words	
<Buffer>	Pointer to the output buffer with the values to be output	
<Loops>	Number of repeats. For an infinite loop use the constant AO3000_INFINITE	
<Delay>	Delay time between 2 repeats in [ms]	
<Callback>	Pointer to user defined callback routine	

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function me3000ErrorMessage.

6.4.4.5 me3000AOReset

Description

This function is used for all CP370 models.

Calling it sets the analog outputs to ground (logical “0”).

Definitions

C	`int me3000AOReset (int iBoardNumber);`
Delphi	`Function me3000AOReset (iBoardNumber: integer): integer;`
Basic	`Declare Function meAO3000Reset Lib “me3000” Alias "_VBme3000AOReset@4" (ByVal lBoardNumber As Long) As Long`
VEE	User object in panel view with menu to select `BoardNumber` and an output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.4.6 me3000AOSetMode

Description

This function is used for all CP370 models.

Configuration of operating modes for analog output. When using the functions me3000AOContinuous and me3000AOWraparound the parameters <Channel> and <Mode> will be ignored.

Definitions

C	`int me3000AOSetMode (int iBoardNumber, int iChannel, int iRange, int iMode);`
Delphi	`Function me3000AOSetMode (iBoardNumber, iChannel, iRange, iMode: integer): integer;`
Basic	`Declare Function me3000AOSetMode Lib "me3000" Alias "_VBme3000AOSetMode@16" (ByVal lBoardNumber As Long, ByVal lChannel As Long, ByVal lRange As Long, ByVal lMode As Long) As Long`
VEE	User object in panel view with menus to select `BoardNumber`, `ChannelNumber`, `Range` and `Mode` as well as an output terminal `Error` for the return value.

Parameters

<BoardNumber>	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
<Channel>	D/A channel number; possible values: 0…3
<Range>	Output voltage range; possible values: `<Range>` — Output range `AO3000_PLUS_10` — 0-10V `AO3000_MINUS_5` — ±5V `AO3000_MINUS_10` — ±10V
<Mode>	Output operating mode. For buffered output use the function `me3000AOUpdate`, for tranparent output use the function `me3000AOSingle`. `<Mode>` — Operating mode `AO3000_BUFFERED` — Buffered output `AO3000_TRANSPARENT` — Transparent output

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.4.7 me3000AOSetTimer

Description

This function is used for the CP370-SIG model only.

Calling it will configure the counter used as a timer for the functions `me3000AOContinuous`, `me3000AOContinuousEx` and `me3000AOWraparound`. The function `me3000FrequencyToCounter` offers a convenient conversion alternative for passing parameters to the function `me3000AOSetTimer`.

Definitions

C	`int me3000AOSetTimer (int iBoardNumber, int iTime);`
Delphi	`Function me3000AOSetTimer (iBoardNumber, iTime: integer): integer;`
Basic	`Declare Function me3000AOSetTimer Lib “me3000” Alias "_VBme3000AOSetTimer@8" (ByVal lBoardNumber As Long, ByVal lTime As Long) As Long`
VEE	User object in panel view with menus to select `BoardNumber`, the dialog box `Time` for the divide factor and an output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
`<Time>`	Divide factor for the 16 bit counter that controls the conversion rate of the D/A converter in the "continuous" and "wrap-around" modes. The start frequency is fixed to 3 MHz. The time value range is between 10µs and 22ms approx. (0x001E…0xFFFF).

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.4.8 me3000AOSetTrigger

Description

This function is used for the CP370 - SIG model only.

Determines the type of conversion start for the functions `me3000AOContinuous`, `me3000AOContinuousEx` and `me3000AOWraparound`. It defines the trigger source and the trigger edge.

Definitions

C	`int me3000AOSetTrigger (int iBoardNumber, int iModePolarity);`
Delphi	`Function me3000AOSetTrigger (iBoardNumber, iModePolarity: integer): integer;`
Basic	`Declare Function me3000AOSetTrigger Lib "me3000" Alias "_VBme3000AOSetTrigger@8" (ByVal lBoardNumber As Long, ByVal lModePolarity As Long) As Long`
VEE	User object in panel view with menus to select `BoardNumber` and `ModePolarity` as well as an output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
`<ModePolarity>`	Condition for conversion start by setting source and rising or falling edge

`<ModePolarity>`	Conversion start
`AO3000_TRIGGER_TIMER`	By software
`AO3000_TRIGGER_EXT_LOW`	External trigger, falling edge
`AO3000_TRIGGER_EXT_HIGH`	External trigger, rising edge

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.4.9 me3000AOSingle

Description

This function is used for all CP370 models.

The function can do transparent output of a single voltage value to a specific D/A channel and can load the D/A converter for buffered voltage output. Within the function me3000AOSetMode the mode must be set to either AO3000_TRANSPARENT or AO3000_BUFFERED.



Note:

This function may only be called after the function me3000AOSetMode has been successfully run at least once. Example:

```
if (me3000AOSetMode(...)!=0) then
me3000AOSingle(...)
endif;
```

Definitions

C	`int me3000AOSingle (int iBoardNumber, int iChannel, short sValue);`
Delphi	`Function me3000AOSingle (iBoardNumber, iChannel: integer; sValue: smallint): integer;`
Basic	`Declare Function me3000AOSingle Lib “me3000” Alias "_VBme3000AOSingle@12" (ByVal lBoardNumber As Long, ByVal lChannel As Long, ByVal iValue As Integer) As Long`
VEE	User object in panel view with menus to select `BoardNumber` and `ChannelNumber`, a dialog box `Value` for the output value and an output terminal `Error` for the return value.

Parameters

<BoardNumber>	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
<Channel>	D/A channel number; possible values: 0…3
<Value>	12 bit output value ranging between a minimum of 0x000 and a maximum of 0xFFF

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function me3000ErrorMessage.

6.4.4.10 me3000AOStop

Description

This function is used for the CP370-SIG model only.

Calling it ends an analog output started with the functions `me3000AOContinuous` or `me3000AOWraparound`.

Definitions

C	`int me3000AOStop (int iBoardNumber);`
Delphi	`Function me3000AOStop (iBoardNumber: integer): integer;`
Basic	`Declare Function me3000AOStop Lib “me3000” Alias "_VBme3000AOStop@4" (ByVal lBoardNumber As Long) As Long`
VEE	User object in panel view with menu to select `BoardNumber` and an output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.4.11 me3000AOStopEx

Description

This function is used for the CP370-SIG model only.

Calling it ends a background process started with the function `me3000AOContinuousEx`.

Definitions

C	`int me3000AOStopEx (int iBoardNumber);`
Delphi	`Function me3000AOStopEx (iBoardNumber: integer): integer;`
Basic	`Declare Function me3000AOStopEx Lib “me3000” Alias "_VBme3000AOStopEx@4" (ByVal lBoardNumber As Long) As Long`

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.4.12 me3000AOUpdate

Description

This function is used for all CP370 models.

Function for buffered voltage output. All D/A channels are updated simultaneously. For loading individual D/A converters please use the function me3000AOSingle in combination with the function me3000AOSetMode in operation mode AO3000_BUFFERED.



Note:

This function may only be called after the function me3000AOSetMode has been successfully run at least once. Example:

```
for(i=0; i<4; i++)
me3000AOSetMode(...)
me3000AOSingle(...)
endfor;

me3000AOUpdate(...);
```

Definitions

C	`int me3000AOUpdate (int iBoardNumber);`
Delphi	`Function me3000AOUpdate (iBoardNumber: integer): integer;`
Basic	`Declare Function me3000AOUpdate Lib “me3000” Alias "_VBme3000AOUpdate@4" (ByVal lBoardNumber As Long) As Long`
VEE	User object in panel view with menu to select `BoardNumber` and an output terminal `Error` for the return value.

Parameters

<BoardNumber>	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.4.13 me3000AOWraparound

Description

This function is used for the CP370-SIG model only.

Function for continuous output of voltage values (function generator). The D/A FIFO must be loaded once before calling this function.



Note:

This function may only be called after the functions `me3000AOSetTrigger`, `me3000AOSetTimer` and `me3000AOSetMode` have been successfully run at least once. Example:

```
if(me3000AOSetTrigger(...) &&
me3000AOSetTimer(...) &&
me3000AOSetMode(...)!=0) then

me3000AOWraparound(...)
endif;
```

Definitions

C	`int me3000AOWraparound (int iBoardNumber);`
Delphi	`Function me3000AOWraparound (iBoardNumber: integer): integer;`
Basic	`Declare Function me3000AOWraparound Lib “me3000” Alias "_VBme3000AOWraparound@4" (ByVal lBoardNumber As Long) As Long`
VEE	User object in panel view with menu to select `BoardNumber` and an output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.5 Digital I/O

6.4.5.1 me3000DIOSetPortDirection

Description

This function is used for all CP370 models.

Configures a digital port as input or output.



Note:

This function must be called for each port before any digital I/O operation can take place.

`Definitions.`

C	`int me3000DIOSetPortDirection (int iBoardNumber, int iPortNo, int iDir);`
Delphi	`Function me3000DIOSetPortDirection (iBoardNumber, iPortNo, iDir: integer): integer;`
Basic	`Declare Function me3000DIOSetPortDirection Lib "me3000" Alias "_VBme3000DIOSetPortDirection@12" (ByVal lBoardNumber As Long, ByVal lPortNo As Long, ByVal lDir As Long) As Long`
VEE	User object in panel view with menus to select `BoardNumber`, `Port` and `Direction` as well as an output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15	
`<PortNo>`	Selection of I/O port:	
	`<Port>`	Digital port
	`PORTA`	16 bit port A
	`PORTB`	16 bit port B
`<Dir>`	Port direction:	
	`<Dir>`	Input/Output
	`MEINPUT`	Port as input
	`MEOUTPUT`	Port as output

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function me3000ErrorMessage.

6.4.5.2 me3000DIGetBit

Description

This function is used for all CP370 models.

Calling it returns the state of the selected input line..



Note:

To properly configure the port the function *me3000DIOSetPortDirection* must first be called for the selected I/O port.

Definitions

C	`int me3000DIGetBit (int iBoardNumber, int iPortNo, int iBitNo, int *piBitValue);`
Delphi	`Function me3000DIGetBit (iBoardNumber, iPortNo, iBitNo: integer; Var piBitValue: integer): integer;`
Basic	`Declare Function me3000DIGetBit Lib “me3000” Alias "_VBme3000DIGetBit@16" (ByVal lBoardNumber As Long, ByVal lPortNo As Long, ByVal lBitNo As Long, ByRef lBitValue As Long) As Long`
VEE	User object in panel view with menus to select `BoardNumber`, `Port`, and `BitNumber` as well as the output terminals `BitValue` and `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15	
`<PortNo>`	Selection of input port:	
	`<Port>`	Input port
	`PORTA`	16 bit port A
	`PORTB`	16 bit port B

<BitNo>	Number of the selected input line to be read; possible values: 0…15
<BitValue>	Pointer to an integer value passing the state of the selected input line; possible values: 0: line set to low level 1: line 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 exact cause of the error can be determined by the function me3000ErrorMessage.

6.4.5.3 me3000DIGetByte

Description

This function is used for all CP370 models.



Note:

To properly configure the port the function *me3000DIOSetPortDirection* must first be called for the selected I/O port.

Definitions

C	`int me3000DIGetByte (int iBoardNumber, int iPortNo, int iByteNo, int *piByteValue);`
Delphi	`Function me3000DIGetByte (iBoardNumber, iPortNo, iByteNo: integer; Var piByteValue: integer): integer;`
Basic	`Declare Function me3000DIGetByte Lib “me3000” Alias "_VBme3000DIGetByte@16" (ByVal lBoardNumber As Long, ByVal lPortNo As Long, ByVal lByteNo As Long, ByRef lByteValue As Long) As Long`
VEE	User object in panel view with menus to select `BoardNumber`, `Port` and `ByteNumber` as well as output terminals `ByteValue` and `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15	
`<PortNo>`	Select input port:	
	`<Port>`	Input port
	`PORTA`	16 bit port A
	`PORTB`	16 bit port B
`<ByteNo>`	Select between upper or lower byte:	
	`<ByteNo>`	Byte
	`LOW_BYTE`	Lower byte
	`HIGH_BYTE`	Upper byte
`<ByteValue>`	Pointer to an integer value where the byte is read into. Only the lower 8 bits are significant.	

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.5.4 me3000DIGetWord

Description

This function is used for all CP370 models.

Calling it reads a word (16 bit) from a port defined as input.



Note:

To properly configure the port the function `me3000DIOSetPortDirection` must first be called for the selected I/O port.

Definitions

C	`int me3000DIGetWord (int iBoardNumber, int iPortNo, int *piValue);`
Delphi	`Function me3000DIGetWord (iBoardNumber, iPortNo: integer; Var piValue: integer): integer;`
Basic	`Declare Function me3000DIGetWord Lib “me3000” Alias "_VBme3000DIGetWord@12" (ByVal lBoardNumber As Long, ByVal lPortNo As Long, ByRef lValue As Long) As Long`
VEE	User object in panel view with menus to select `BoardNumber` and `Port` as well as output terminals `WordValue` and `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15	
`<PortNo>`	Select input port:	
	`<Port>`	Input port
	PORTA	16 bit port A
	PORTB	16 bit port B
`<Value>`	Pointer to an integer value where the word is read into.	

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function me3000ErrorMessage.

6.4.5.5 me3000DOReset

Description

This function is used for all CP370 models.

Calling it sets the specified port to the original condition and configures it as an input port.

Definitions

C	`int me3000DOReset (int iBoardNumber, int iPortNo);`
Delphi	`function me3000DOReset (iBoardNumber, iPortNo: integer): integer;`
Basic	`Declare Function me3000DOReset Lib “me3000” Alias "_VBme3000DOReset@8" (ByVal lBoardNumber As Long, ByVal lPortNo As Long) As Long`
VEE	User object in panel view with menus to select `BoardNumber` and `Port` as well as an output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15	
`<PortNo>`	Select output port:	
	`<Port>`	Output port
	`PORTA`	16 bit port A
	`PORTB`	16 bit port B

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.5.6 me3000DOSetBit

Description

This function is used for all CP370 models.

Calling it sets the state of the selected output line.



Note:

To properly configure the port the function `me3000DIOSetPortDirection` must first be called for the selected I/O port.

Definitions

C	`int me3000DOSetBit (int iBoardNumber, int iPortNo, int iBitNo, int iBitValue);`
Delphi	`Function me3000DOSetBit (iBoardNumber, iPortNo, iBitNo, iBitValue: integer): integer;`
Basic	`Declare Function me3000DOSetBit Lib “me3000” Alias "_VBme3000DOSetBit@16" (ByVal lBoardNumber As Long, ByVal lPortNo As Long, ByVal lBitNo As Long, ByVal lBitValue As Long) As Long`
VEE	User object in panel view with menus to select `BoardNumber`, `Port` and `BitNumber` as well as a dialog box for the `BitValue` and an output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
`<PortNo>`	Select output port:

`<Port>`	Output port
PORTA	16 bit port A
PORTB	16 bit port B

`<BitNo>`	Number of the selected output line to be set. Possible values are 0…15
`<BitValue>`	Possible values are: = 0: bit is set to low level > 0: bit 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 exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.5.7 me3000DOSetByte

Description

This function is used for all CP370 models.

Calling it writes a Byte to a port defined as output.



Note:

To properly configure the port the function `me3000DIOSetPortDirection` must first be called for the selected I/O port.

Definitions

C	`int me3000DOSetByte (int iBoardNumber, int iPortNo, int iByteNo, int iByteValue);`
Delphi	`function me3000DOSetByte (iBoardNumber, iPortNo, iByteNo, iByteValue: integer): integer;`
Basic	`Declare Function me3000DOSetByte Lib “me3000” Alias "_VBme3000DOSetByte@16" (ByVal lBoardNumber As Long, ByVal lPortNo As Long, ByVal lByteNo As Long, ByVal lByteValue As Long) As Long`
VEE	User object in panel view with menus to select `BoardNumber`, `Port` and `ByteNumber` as well as a dialog box for the `ByteValue` and an output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15	
`<PortNo>`	Select output port:	
	`<Port>`	Output port
	`PORTA`	16 bit port A
	`PORTB`	16 bit port B



<ByteNo>	Select between lower or upper byte:	
	<ByteNo>	Byte
	LOW_BYTE	Lower byte
	HIGH_BYTE	Upper byte
<ByteValue>	Output value; possible values are: 0…255 (0x00…0xFF).	

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function me3000ErrorMessage.

6.4.5.8 me3000DOSetWord

Description

This function is used for all CP370 models.

Calling it writes a word (16 bit) to a port defined as output.



Note:

To properly configure the port the function `me3000DIOSetPortDirection` must first be called for the selected I/O port.

Definitions

C	`int me3000DOSetWord (int iBoardNumber, int iPortNo, int iValue);`
Delphi	`function me3000DOSetWord (iBoardNumber, iPortNo, iValue: integer): integer;`
Basic	`Declare Function me3000DOSetWord Lib "me3000" Alias "_VBme3000DOSetWord@12" (ByVal lBoardNumber As Long, ByVal lPortNo As Long, ByVal lValue As Long) As Long`
VEE	User object in panel view with menus to select `BoardNumber`, and `Port` as well as a dialog box for the `WordValue` and an output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15	
`<PortNo>`	Select output port:	
	`<Port>`	Output port
	`PORTA`	16 bit port A
	`PORTB`	16 bit port B
`<Value>`	Output value; possible values are: 0…65535 (0x0000…0xFFFF).	

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.6 Pulse Width Modulation

6.4.6.1 me3000PWMSetMode

Description

This function is used for the CP370-SIG model only.

This function configures the functional group “Pulse Width Modulation” (PWM) on the CP370 board. The modulated signal is on pin 32 on the interface connector.

Definitions

C	`int me3000PWMSetMode (int iBoardNumber, double dFreq, int iDutyCycle, int iInverted);`
Delphi	`function me3000PWMSetMode (iBoardNumber: integer, dFreq: double; iDutyCycle, iInverted: integer): integer;`
Basic	`Declare Function me3000PWMSetMode Lib “me3000” Alias "_VBme3000PWMSetMode@20" (ByVal lBoardNumber As Long, ByVal dFreq As Double, ByVal lDutyCycle As Long, ByVal lInverted As Long) As Long`
VEE	User object in panel view with menus to select `BoardNumber` and `Polarity`, a slider for `DutyCycle`, a dialog box for `Frequency` as well as an output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
`<Freq>`	Frequency of the PWM signal in [Hz]. Range is: 0.4….30000 Hz
`<DutyCycle>`	Duty cycle of the PWM signal (can be set from 0% to 99% in increments of 1%)
`<Inverted>`	Polarity of the PWM signal:

`<Polarity>`	Inverted/not inverted
`PWM3000_NORMAL`	signal not inverted
`PWM3000_INVERTED`	signal inverted

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.6.2 me3000PWMStart

Description

This function is used for the CP370-SIG model only.

Calling it starts the pulse-width modulation. The modulated signal is directed to pin 32 of the interface connector.

Definitions

C	`int me3000PWMStart (int iBoardNumber);`
Delphi	`function me3000PWMStart (iBoardNumber: integer): integer;`
Basic	`Declare Function me3000PWMStart Lib “me3000” Alias "_VBme3000PWMStart@4" (ByVal lBoardNumber As Long) As Long`
VEE	User object in panel view with menu to select `BoardNumber` and an output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.6.3 me3000PWMStop

Description

This function is used for the CP370-SIG model only.

Calling it ends the output of the pulse-width modulated signal.

Definitions

C	`int me3000PWMStop (int iBoardNumber);`
Delphi	`function me3000PWMStop (iBoardNumber: integer): integer;`
Basic	`Declare Function me3000PWMStop Lib “me3000” Alias "_VBme3000PWMStop@4" (ByVal lBoardNumber As Long) As Long`
VEE	User object in panel view with menu to select `BoardNumber` and an output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.7 Counter

6.4.7.1 me3000CNTInterrupt

Description

This function is used for all CP370 models.

Calling it enables or disables the interrupt function of the counters. Upon zero axis crossing an interrupt is triggered.

Definitions

C	`int me3000CNTInterrupt (int iBoardNumber, int iCounter, int iFlag, PVOID_PROC pCallback);`
Delphi	`function me3000CNTInterrupt (iBoardNumber, iCounter, iFlag: integer; pCallback: PVOID_PROC): integer;`
Basic	`Declare Function me3000CNTInterrupt Lib "me3000" Alias "_VBme3000CNTInterrupt@16" (ByVal lBoardNumber As Long, ByVal lCounter As Long, ByVal lFlag As Long, ByVal pCallback As Long) As Long`

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15	
`<Counter>`	Selection of counter:	
	`<Counter>`	Counter A/B
	`COUNTER3000_A`	Select Counter A
	`COUNTER3000_B`	Select Counter B
`<Flag>`	Will enable or disable the interrupt mode for both counters:	
	`<Flag>`	Enable/disable
	`COUNTER3000_ENABLE`	Interrupt enable
	`COUNTER3000_DISABLE`	Interrupt disable
`<Callback>`	Pointer to user defined callback routine	

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.7.2 me3000CNTRead

Description

This function is used for all CP370 models.

Calling it reads the current counter value.

Definitions

C	`int me3000CNTRead (int iBoardNumber, int iCounter, short *psCounterValue);`
Delphi	`function me3000CNTRead (iBoardNumber, iCounter: integer; psCounterValue: short): integer;`
Basic	`Declare Function me3000CNTRead Lib "me3000" Alias "_VBme3000CNTRead@12" (ByVal lBoardNumber As Long, ByVal lCounter As Long, ByRef piCounterValue As Integer) As Long`
VEE	User object in panel view with menus to select `BoardNumber` and `Counter` as well as output terminals for the `Counter-Value` and `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15	
`<Counter>`	Selection of counter:	
	`<Counter>`	Counter A/B
	`COUNTER3000_A`	Select counter A
	`COUNTER3000_B`	Select counter B
`<CounterValue>`	Returns the current counter value.	

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.7.3 me3000CNTReset

Description

This function is used for all CP370 models.

Calling it sets the selected counter to zero and disables it.

Definitions

C	`int me3000CNTReset (int iBoardNumber, int iCounter);`
Delphi	`function me3000CNTReset (iBoardNumber, iCounter: integer): integer;`
Basic	`Declare Function me3000CNTReset Lib "me3000" Alias "_VBme3000CNTReset@8" (ByVal lBoardNumber As Long, ByVal lCounter As Long) As Long`
VEE	User object in panel view with menu to select `BoardNumber` and `Counter` as well as an output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15	
`<Counter>`	Selection of counter:	
	`<Counter>`	Counter A/B
	`COUNTER3000_A`	Select counter A
	`COUNTER3000_B`	Select counter B

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.7.4 me3000CNTSetRange

Description

This function is used for all CP370 models.

Calling it allows the loading of the start and end values for the user accessible counters. Each of the 16 bit counters must be loaded separately. Calling the function me3000CNTStart starts counting at the <StartValue> in the given direction. In operation mode “upwards” the current counter value is continuously compared to the <EndValue>. When the end value is reached the counter resets to zero and continues counting. In operation mode “downwards” the current counter value is continuously compared to zero. When the value zero is reached the counter is set to the <EndValue> and then continues counting. Upon zero axis crossing an interrupt can be triggered in both directions.



Note:

The maximum frequency of the external clock must not exceed 90kHz when using an end value minor than 65535. When using an end value equal to 65535, which corresponds to the full use of the counter range, the maximum counter frequency is 6MHz.

Definitions

C	`int me3000CNTSetRange (int iBoardNumber, int iCounter, int iEndValue, int iStartValue);`
Delphi	`function me3000CNTSetRange (iBoardNumber, iCounter, iEndValue, iStartValue: integer): integer;`
Basic	`Declare Function me3000CNTSetRange Lib “me3000” Alias "_VBme3000CNTSetRange@16" (ByVal lBoardNumber As Long, ByVal lCounter As Long, ByVal lEndValue As Long, ByVal IStartValue As Long) As Long`
VEE	User object in panel view with menu to select `BoardNumber` and `Counter`, dialog boxes for `EndValue` and `StartValue` as well as an output terminal `Error` for the return value.

Parameters

<BoardNumber>	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15	
<Counter>	Selection of counter:	
	<Counter>	Counter A/B
	COUNTER3000_A	Select counter A
	COUNTER3000_B	Select counter B
<EndValue>	End value for the counter. Possible values are: 0…65535 (0x0000…0xFFFF)	
<StartValue>	Start value for the counter. Possible values are: 0…65535 (0x0000…0xFFFF)	

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function me3000ErrorMessage.

6.4.7.5 me3000CNTStart

Description

This function is used for all CP370 models.

Calling it starts the selected counter.

Definitions

C	`int me3000CNTStart (int iBoardNumber, int iCounter);`
Delphi	`function me3000CNTStart (iBoardNumber, iCounter: integer): integer;`
Basic	`Declare Function me3000CNTStart Lib “me3000” Alias "_VBme3000CNTStart@8" (ByVal lBoardNumber As Long, ByVal lCounter As Long) As Long`
VEE	User object in panel view with menu to select `BoardNumber` and `Counter` as well as an output terminal `Error` for the return value.

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15	
`<Counter>`	Selection of counter:	
	`<Counter>`	Counter A/B
	`COUNTER3000_A`	Select counter A
	`COUNTER3000_B`	Select counter B

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.8 Interrupt Handling

6.4.8.1 me3000DisableExternalInt

Description

This function is used for all CP370 models.

Calling it disables the external interrupt.

Definitions

C	`int me3000DisableExternalInt (int iBoardNumber);`
Delphi	`function me3000DisableExternalInt (iBoardNumber: integer): integer;`
Basic	`Declare Function me3000DisableExternalInt Lib "me3000" Alias "_VBme3000DisableExternalInt@4" (ByVal lBoardNumber As Long) As Long`

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.8.2 me3000EnableExternalInt

Description

This function is used for all CP370 models.

Calling it enables the external interrupt.

Definitions

C	`int me3000EnableExternalInt (int iBoardNumber, PVOID_PROC pCallback);`
Delphi	`function me3000EnableExternalInt (iBoardNumber: integer; pCallback: PVOID_PROC): integer;`
Basic	`Declare Function me3000EnableExternalInt Lib "me3000" Alias "_VBme3000EnableExternalInt@8" (ByVal lBoardNumber As Long, ByVal pCallback As Long) As Long`

Parameters

`<BoardNumber>`	Board number for the addressed CP370 among boards 1...16. Possible values: 0…15
`<Callback>`	Pointer to user defined callback routine

Return Value

If the function is successfully executed, a “1” is returned. If an error occurs a “0” is returned. The exact cause of the error can be determined by the function `me3000ErrorMessage`.

6.4.9 Error Handling

6.4.9.1 me3000ErrorMessage

Description

This function is used for all CP370 models.

If the previously called API function returned an error code, this function displays a corresponding description of the error in a message box.



Note:

This function can only be called if the previously called API function of `ME3000.DLL` returned an error (error code “0”)

Definitions

C	`int me3000ErrorMessage (char *pcAppName);`
Delphi	`Function me3000ErrorMessage (Var pcAppName: tmestring): integer;`
Basic	`Declare Function me3000ErrorMessage Lib “me3000” Alias "_VBme3000ErrorMessage@4" (ByRef pcAppName As String) As Long`
VEE	User object in panel view with input terminal `AppName` and the output terminal `DLLErrorCode` passing the return value.

Parameters

<AppName>	Pointer to a string that is used for the title in the Message Box

Return Value

The function always returns the DLL global variable <DLLErrorCode>

6.4.9.2 me3000GetDriverErrorString

Description

This function is used for all CP370 models.

If the previously called API function returned an error code, the parameter <ErrorText> passes a corresponding description of the error as a string.



Note:

This function can only be called if the previously called API function of ME3000.DLL returned an error (error code “0”)!

Definitions

C	`int me3000GetDriverErrorString (char *pcErrorText);`
Delphi	`Function me3000GetDriverErrorString (Var pcErrorText: string): integer;`
Basic	`Declare Function me3000GetDriverErrorString Lib “me3000” Alias "_VBme3000GetDriverErrorString@4" (ByRef pcErrorText As String) As Long`
VEE	User object in panel view with output terminal `ErrorText` and the output terminal `DLLErrorCode` passing the return value.

Parameters

`<ErrorText>`	Pointer to a string giving a text description of the error corresponding to the error code

Return Value

The function always returns the DLL global variable <DLLErrorCode>.

This page was intentionally left blank.