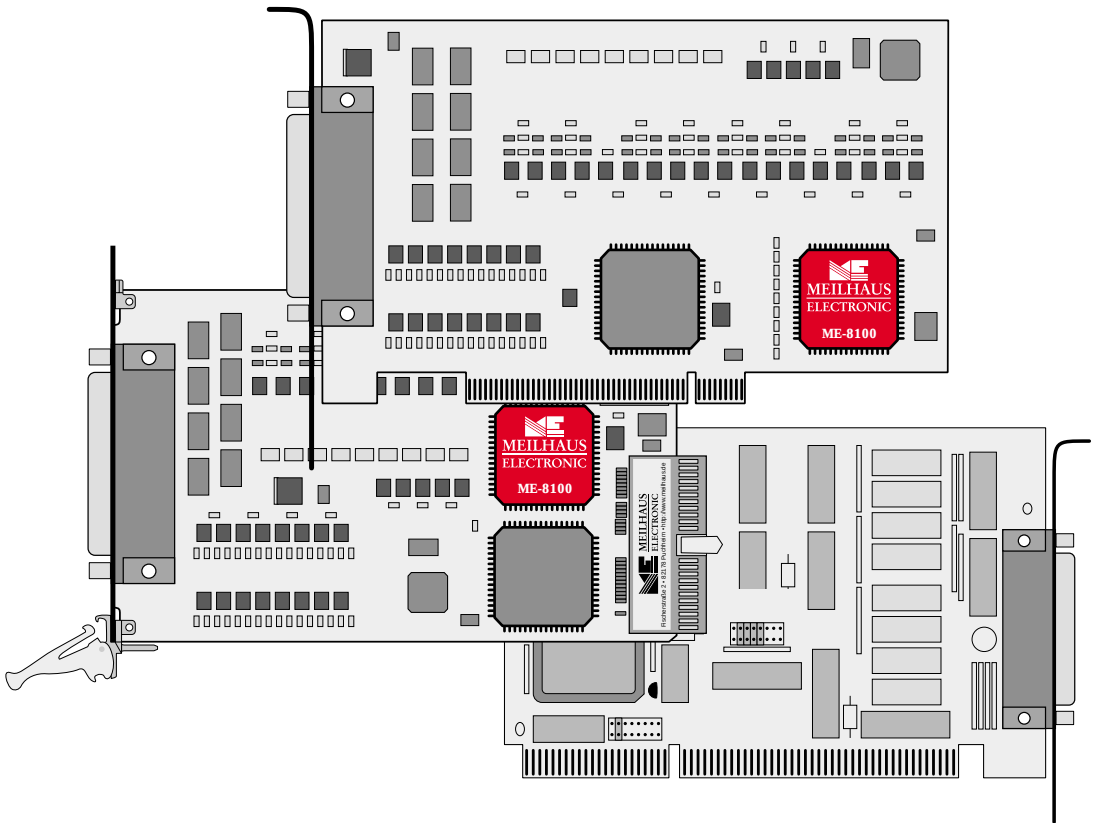
Meilhaus Electronic Handbuch

ME-81, ME-8100 1.6D

ISA, PCI- und CompactPCI-Varianten



Opto I/O-Karte mit Bitmuster-Vergleicher
und optionalem Zähler

Impressum

Handbuch ME-81, ME-8100

Revision 1.6D
Ausgabedatum: 8. Oktober 2002

Meilhaus Electronic GmbH
Fischerstraße 2
D-82178 Puchheim bei München
Germany
http://www.meilhaus.de

Wichtiger Hinweis:
Alle in diesem Handbuch enthaltenen Informationen wurden mit größter Sorgfalt und nach bestem Wissen zusammengestellt. Dennoch sind Fehler nicht ganz auszuschließen.

Aus diesem Grund sieht sich die Firma Meilhaus Electronic GmbH dazu veranlaßt, darauf hinzuweisen, daß sie weder eine Garantie (abgesehen von den im Garantieschein vereinbarten Garantieansprüchen) noch die juristische Verantwortung oder irgendeine Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen, übernehmen kann.

Für die Mitteilung eventueller Fehler sind wir jederzeit dankbar.

ISO 9001

EUROPE

Inhalt

1 Einführung

Sehr geehrte Kundin, sehr geehrter Kunde,

Mit dem Kauf einer PC-Einsteckkarte von Meilhaus Electronic haben Sie sich für ein technologisch hochwertiges Produkt entschieden, das unser Haus in einwandfreiem Zustand verlassen hat.

Überprüfen Sie trotzdem die Vollständigkeit und den Zustand Ihrer Lieferung. Sollten irgendwelche Mängel auftreten, bitten wir Sie, uns sofort in Kenntnis zu setzen.

Bevor Sie die Karte in Ihren Rechner einbauen, lesen Sie bitte aufmerksam diese Bedienungsanleitung, insbesondere Kapitel 2 zur Installation durch.

Beachten Sie bei den ISA-Versionen vor allem den Abschnitt zur Einstellung der Jumper. Dies erspart Ihnen das spätere, nochmalige Öffnen Ihres Rechners.

1.1 Lieferumfang

Wir sind selbstverständlich bemüht, Ihnen ein vollständiges Produktpaket auszuliefern. Um aber in jedem Fall sicherzustellen, daß Ihre Lieferung komplett ist, können Sie anhand nachfolgender Liste die Vollständigkeit Ihres Paketes überprüfen.

Ihr Paket sollte folgende Teile enthalten:

- Optoisolierte Digital-I/O-Karte der ME-81/8100 Familie für ISA-, PCI- oder CompactPCI-Bus
- Handbuch im PDF-Format auf CD-ROM
  (optional in gedruckter Form)
- Treiber-Software auf CD-ROM
- ME-81 ISA: 37poliger Sub-D Gegenstecker,
  ME-8100 PCI/cPCI: 78poliger Sub-D Gegenstecker

1.2 Leistungsmerkmale

Modell-Übersicht

Modell	Optoisolierte Ein-/Ausgänge	Zähler
ME-81 ISA	16 Ein- und 16 Ausgänge (24 V)	---
ME-8100A PCI **ME-8100A cPCI**	16 Ein- und 16 Ausgänge (24 V)	3 x 16 Bit (24 V)
ME-8100B PCI **ME-8100B cPCI**	32 Ein- und 32 Ausgänge (24 V)	3 x 16 Bit (24 V)

Tabelle 1: Modell-Übersicht ME-81/8100 Familie

Die Karten der ME-81/8100 Familie sind mit digitalen Ein-/Ausgabe-Ports und die 8100er-Modelle zusätzlich mit drei 16 Bit Zählern ausgestattet. Sowohl die Digital-Ports als auch die Zählersignale sind optoisoliert und für den in der Steuerungstechnik üblichen 24 V-Pegel ausgelegt.

Die ME-81 und ME-8100A besitzen 16 Ein- und 16 Ausgänge und einen 16 Bit breiten Bitmuster-Vergleicher. Die ME-8100B bietet 32 Ein- und 32 Ausgänge und zwei 16 Bit breite Bitmuster-Vergleicher. Die ME-8100 Modelle stellen zusätzlich drei 16 Bit Zähler zur Verfügung.

Auf den ME-8100 Modellen ist eine Umschaltung von „Source“- auf „Sink“-Treiber per Software möglich. Dies ermöglicht eine individuelle Anpassung an Ihre Anforderungen. Die Ausgänge aller Modelle sind sowohl im ausgeschalteten Zustand als auch nach dem Einschalten des Rechners zunächst hochohmig. D. h. der Spannungspegel, der sich am Ausgangs-Pin einstellt hängt von Ihrer externen Beschaltung ab. Erst nach Schreiben einer „1“ wird der Ausgang leitend.

Als Besonderheit bietet die ME-81/8100 die Betriebsarten „Bitmustervergleich“ und „Bitmusteränderung“. Bei Gleichheit eines 16 Bit breiten Bitmusters oder bei Bitmusteränderung eines über-

wachten Bits wird ein Interrupt ausgelöst. Außerdem kann die Karte zur Überwachung von Pegelzustandsänderungen eingesetzt werden.

Die Basisadresse wird bei den ISA-Modellen über einen DIP-Schalter eingestellt. Sie kann in einem weiten Bereich variiert werden. Bei den PCI-/cPCI-Modellen wird die Ressourcen-Vergabe vom BIOS bzw. Betriebssystem automatisch vergenommen (Plug&Play).

Die mitgelieferte Software ermöglicht das rasche Einbinden der Karten in eigene Applikationen der Meß- und Steuerungstechnik unter Windows (32 Bit Versionen). Es sind Treiber für Agilent VEE (früher HP VEE) und LabVIEW™ (National Instruments), verfügbar. Für die ISA-Versionen sind auf Anfrage Treiber für DOS und Windows 3.1 erhältlich.

1.3 Systemanforderungen

Die Karten können, je nach Version, in Computer mit Pentium Prozessor (empfohlen) und Kompatiblen eingesetzt werden, die über einen freien ISA-8 Bit bzw. Standard-PCI oder CompactPCI Steckplatz (je nach Version) verfügen.

1.4 Wichtiger Hinweis für die ISA-Versionen

Falls Sie einen PC mit PCI-Bus und einem BIOS mit Plug&Play-Funktionalität benutzen, müssen Sie für alle ISA-Einsteckkarten mit Interruptfunktion im BIOS Ihres Rechners den Interruptkanal dieser Karte für den ISA-Bus reservieren. Das entsprechende BIOS-Menü kann je nach BIOS-Hersteller etwas variieren (siehe Handbuch Ihres Motherboards). **Ansonsten ist die Interruptfunktion nicht gewährleistet!!!**

Beachten Sie, daß bei neueren Rechnern der ISA-Bus - abweichend von seiner Spezifikation - teilweise mit mehr als 8 MHz betrieben werden kann (siehe Einstellung im Setup Ihres PCs). In diesem Fall können wir jedoch eine einwandfreie Funktion der ISA-Karte nicht gewährleisten.

1.5 Softwareunterstützung

Den aktuellen Stand des Software-Lieferumfangs entnehmen Sie bitte den entsprechenden README-Dateien.

Systemtreiber	Für alle gängigen Betriebssysteme (siehe README-Dateien)
ME-Software-Developer-Kit (ME-SDK):	Beispiele für alle gängigen Programmiersprachen, sowie Tools und Testprogramme
Graphische Programmierumgebungen:	Meilhaus VEE-Treibersystem für HP VEE ab Version 4.0, HP VEE Lab, Agilent VEE Pro und Agilent VEE OneLab
	Treibersystem für LabView™ ab Version 4.0

2 Installation

Bitte lesen Sie **vor Einbau der Karte** das Handbuch Ihres Rechners bzgl. der Installation von zusätzlichen Hardwarekomponenten und das Kapitel „Hardware-Installation“ in diesem Handbuch (sofern zutreffend, z. B. für ISA-Karten).

• Installation unter Windows (Plug&Play)

Sie finden eine Anleitung in HTML-Form auf CD-ROM. Bitte **vor Installation lesen** und bei Bedarf ausdrucken!

Grundsätzlich gilt folgende Vorgehensweise:

Falls Sie die Treiber-Software in gepackter Form erhalten haben, entpacken Sie bitte **vor Einbau der Karte** die Software in ein Verzeichnis auf Ihrem Rechner (z. B. C:\temp).

Bauen Sie die Karte in Ihren Rechner ein und installieren Sie anschließend die Treiber-Software. Diese Reihenfolge ist wichtig, um die Plug&Play-Funktionalität unter Windows 95*/98/Me/ 2000/XP zu gewährleisten. Für Windows 95* und NT 4.0 gilt dies analog, beachten Sie jedoch die etwas andere Vorgehensweise bei der Treiberinstallation.

**Sofern Windows-Version von der betreffenden Karte unterstützt wird (siehe Readme-Dateien)*

• Installation unter Linux

Beachten Sie die Installationshinweise, die in der Archiv-Datei des jeweiligen Treibers enthalten sind.

Hinweis: Falls Sie PCI/cPCI-Modelle zusammen mit bereits vorhandener Applikationssoftware benutzen wollen, beachten Sie bitte die Hinweise in den entsprechenden README-Dateien.

Falls Sie eine ISA-Karte verwenden, nehmen Sie bitte zuerst die Jumpereinstellungen auf der Karte vor (siehe folgendes Kapitel).

2.1 Hardware-Installation ISA-Modelle

☛ Schalten Sie Ihren Rechner aus.

Achtung: Gefahr der Zerstörung hochempfindlicher Bauteile durch elektrostatische Entladung!

Deshalb: Entladen Sie Ihre Person vor Einbau der Karte indem Sie z. B. ein blankes Gehäuseteil Ihres Rechners berühren.

☛ Ziehen Sie das Netzkabel Ihres Rechners.

☛ Öffnen Sie das Gehäuse.

2.1.1 Position der Jumper

Bei der ME-81 ISA müssen vor dem Einbau Jumper-Einstellungen vorgenommen bzw. dahingehend überprüft werden, ob die werkseitigen Einstellungen für Ihren Rechner geeignet sind. Schalten Sie vor dem Einbau der Karte unbedingt Ihren Rechner aus.

Die Lage der Jumper auf der ME-81 können Sie der schematischen Darstellung der Karte entnehmen. Erläuterungen zu den Einstellungen finden Sie in den nachfolgenden Kapiteln.

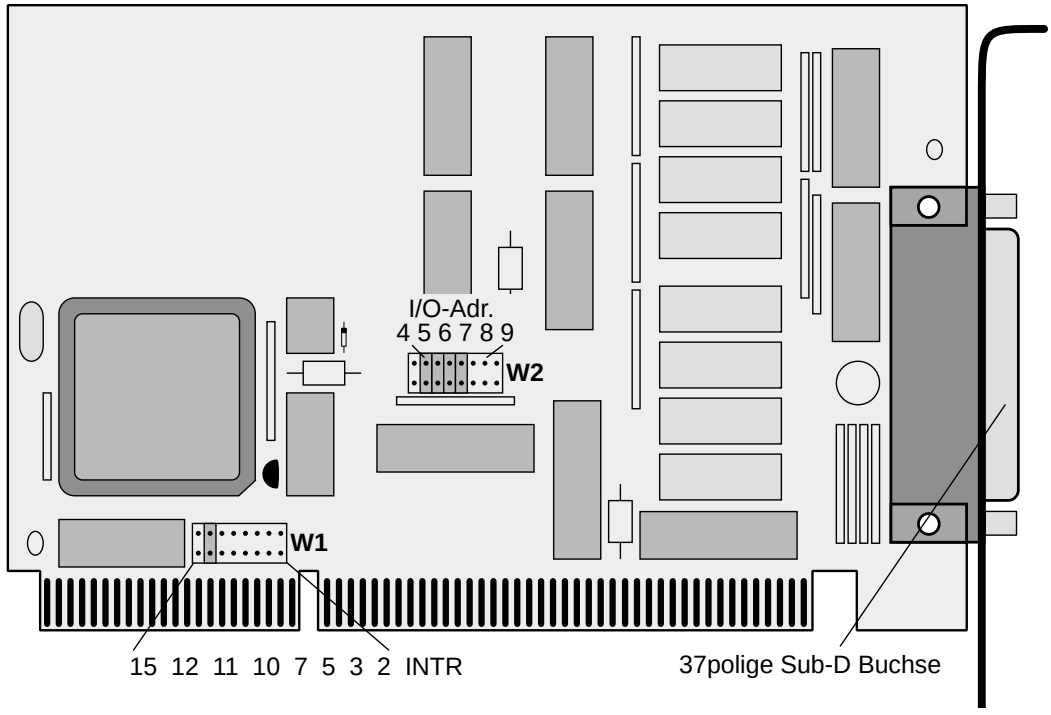


Abb. 1: Schematische Darstellung der ME-81 ISA

2.1.2 Einstellungen der Jumper

2.1.2.1 Basisadresse

Über die **Jumperreihe W2** läßt sich die **Basisadresse** (BA) der ME-81 im Bereich von 0Hex bis 3F0Hex in Schritten von 10Hex einstellen. Mit der Basisadresse beginnend, belegt die ME-81 12 Bytes des I/O-Adreßraumes. Vermeiden Sie bei der Einstellung Adreßkonflikte mit anderen Karten!

Ein gesteckter Jumper entspricht dem logischen Zustand „0“, ein abgezogener Jumper dem Zustand „1“ womit die entsprechende Adreßleitung ausgewählt wird. Die Basisadresse errechnet sich dann durch Aufsummierung der Wertigkeit der Steckplätze der nicht gesteckten Jumper. Das Beispiel erläutert die werksseitige Grundeinstellung der Karte (300Hex).

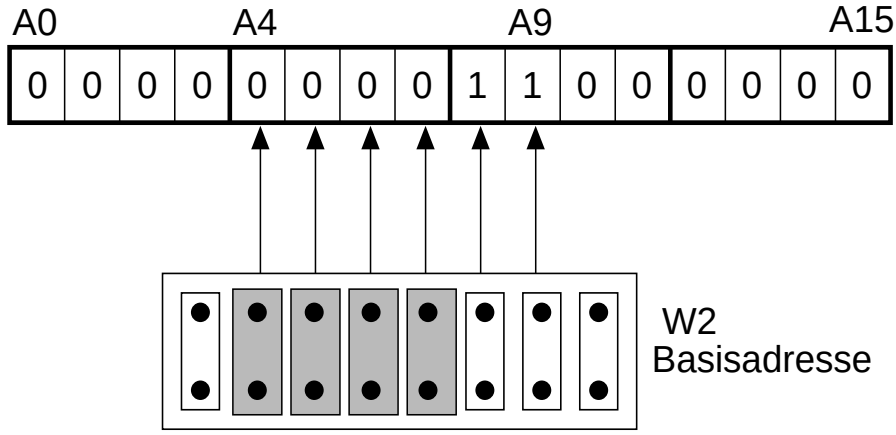


Abb. 2: Adreß-Jumper, hier Grundeinstellung: 300Hex

2.1.2.2 Interrupts

Für die Interruptsteuerung müssen Sie mittels **Jumperreihe W1** eine Interrupt-Request-Leitung (IRQ) von 2, 3, 5, 7, 10, 11, 12 oder 15 auswählen. Es ist jedoch zu beachten, daß der gewählte Interrupt von keiner anderen Interrupt-Quelle belegt wird.

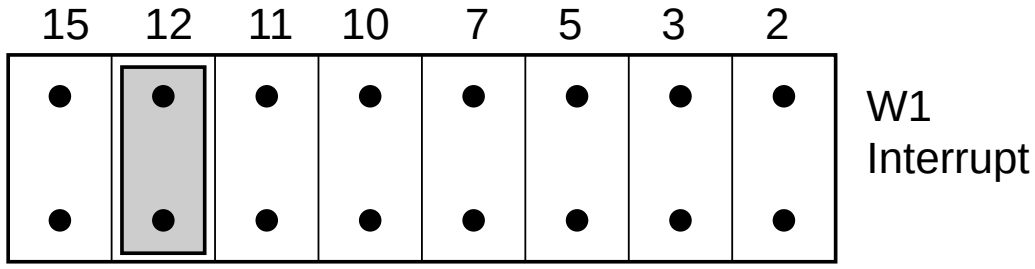


Abb. 3: Anordnung der Jumper

2.1.2.3 Standardeinstellungen

Funktion	Jumper/Schalter	Einstellung
Basisadresse	Jumperreihe W2	300Hex
Interruptkanal	Jumperreihe W1	IRQ 12

Tabelle 2: Standardeinstellungen der ME-81 ISA

3 Hardware

3.1 Blockschaltbild

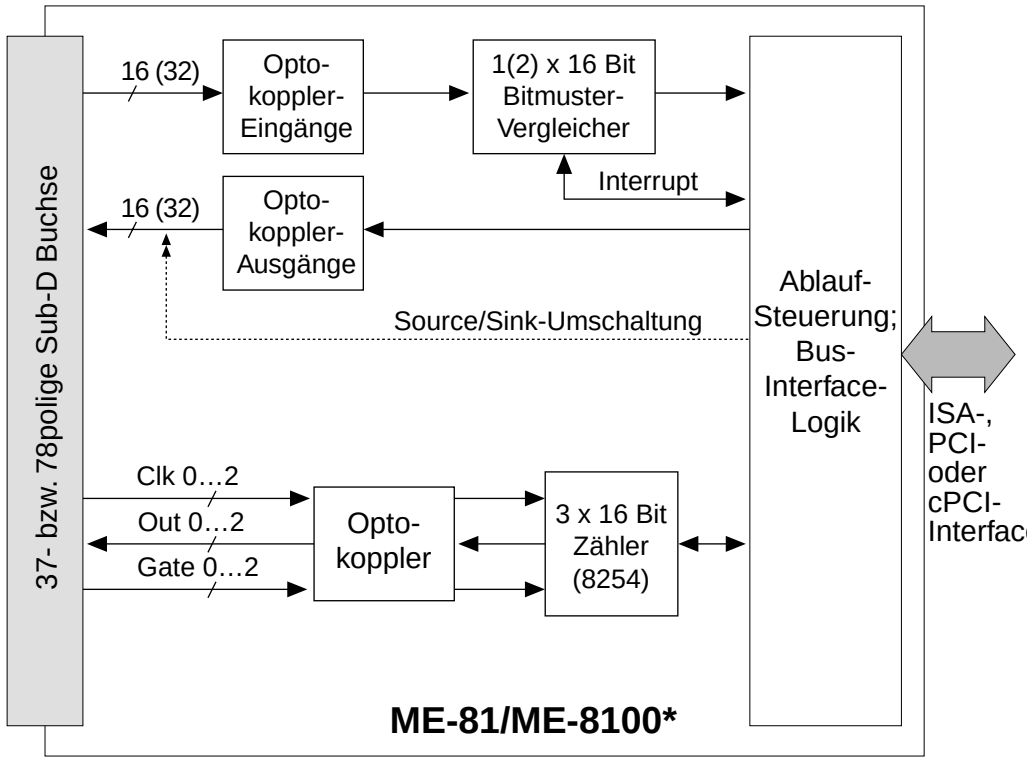


Abb. 4: Blockschaltbild der ME-81/8100

* Je nach Modell sind nicht alle der in obigem Blockschaltbild dargestellten Funktionsgruppen vorhanden:

ME-81: 16 Ein- und 16 Ausgängen, 1 x 16 Bit Bitmuster-Vergleicher (ohne Zähler und Source/Sink-Umschaltung).

ME-8100A: 16 Ein- und 16 Ausgänge, 1 x 16 Bit Bitmuster-Vergleicher, 3 x 16 Bit Zähler.

ME-8100B: 32 Ein- und 32 Ausgänge, 2 x 16 Bit Bitmuster-Vergleicher, 3 x 16 Bit Zähler.

ISA-Modelle: 37poliger Sub-D Buchse;
PCI-/cPCI-Modelle: 78poliger Sub-D Buchse

3.2 Generelle Hinweise

Achtung: Sämtliche Steckverbindungen der Karte sollten grundsätzlich nur im spannungslosen Zustand hergestellt bzw. gelöst werden.

Die Belegung der 37poligen (ME-81 ISA) bzw. 78poligen (ME-8100 PCI/cPCI) Sub-D Buchsen finden Sie im Anhang (siehe „Anschlußbelegungen" auf Seite 72).

3.3 Betriebsarten

Die Konfiguration der Karten erfolgt durch entsprechende Programmierung durch den Anwender. Verwenden Sie hierzu die mitgelieferte Treibersoftware, siehe Kap. 5.3 „Beschreibung der API-Funktionen" auf Seite 46 (wird auch für ISA-Modelle empfohlen).

Die Karten der ME-81/ME-8100 Familie können in folgenden 3 Betriebsarten für die digitale Ein-/Ausgabe konfiguriert werden:

3.3.1 Digitale Ein-/Ausgabe

ME-81/8100A: Ein 16 Bit breiter Eingangsport (optoisoliert) und ein 16 Bit breiter Ausgangsport (optoisoliert).

8100B: Zwei 16 Bit breite Eingangsports (optoisoliert) und zwei 16 Bit breite Ausgangsports (optoisoliert).

Die Ausgänge aller Modelle können per Software portweise hochohmig geschaltet werden. Zusätzlich können bei den ME-8100 Modellen, die Ausgänge von „Sink"-Treiber (low-aktiv) auf „Source"-Treiber (high-aktiv) umgeschaltet werden. Dies ermöglicht eine individuelle Anpassung an Ihre Anforderungen. Die Ausgänge aller Modelle sind sowohl im ausgeschalteten Zustand als auch nach dem Einschalten des Rechners zunächst hochohmig. D. h. der Spannungspegel, der sich am Ausgangs-Pin einstellt hängt von Ihrer externen Beschaltung ab. Erst nach Schreiben einer „1" wird der Ausgang leitend.

3.3.2 Bitmuster-Vergleich

In der Betriebsart „Bitmuster-Vergleich“ wird ein ins Vergleichsregister geschriebenes Bitmuster mit dem am korrespondierenden Eingangsport anliegenden Bitmuster verglichen. Bei Bitmuster-Gleichheit kann (falls freigegeben) ein Interrupt ausgelöst werden.

3.3.3 Bitmuster-Änderung

In der Betriebsart „Bitmuster-Änderung“ können ein oder mehrere ausgewählte Eingangsbits auf Zustandsänderung überwacht werden. Als Referenz dienen dabei die entsprechenden Bits des korrespondierenden Maskenregisters. Bei Zustandsänderung ($0 \rightarrow 1$ oder $1 \rightarrow 0$) von mindestens einem mit einer „1“ maskierten Bit kann (falls freigegeben) ein Interrupt ausgelöst werden.

3.3.4 Zähler (8254)

Dieser Abschnitt gilt für die Modelle ME-8100 A und B, **nicht** jedoch für die ME-81 ISA!

Als Zählerbaustein kommt der Standardtyp 82C54 zum Einsatz. Dies ist ein sehr vielseitig einsetzbarer Baustein, der über 3 unabhängige 16 Bit (Abwärts-) Zähler verfügt. Der Zählertakt von max. 1 MHz muß für jeden Zähler ext. zugeführt werden. Nach geeigneter Beschaltung des Gate-Eingangs (Freigabe durch Low-Pegel) zählt der entsprechende Zähler negativ flankengesteuert abwärts, wobei im BCD- oder Binärcode gezählt werden kann. Jeder Zähler kann unabhängig per Software für folgende 6 Betriebsarten konfiguriert werden:

- Modus 0: Zustandsänderung bei Nulldurchgang
- Modus 1: Retriggerbarer „One Shot“
- Modus 2: Asymmetrischer Teiler
- Modus 3: Symmetrischer Teiler
- Modus 4: Zählerstart durch Softwaretrigger
- Modus 5: Zählerstart durch Hardwaretrigger

Die Konfigurierung des 82C54 erfolgt per Software durch den Anwender. Benutzen Sie zur Programmierung die mitgelieferte Treibersoftware unter Windows (32 Bit) siehe Kap. 5.3.3 „Zählerfunktionen“ auf Seite 61. Die folgende Abbildung zeigt den Aufbau des Bausteins:

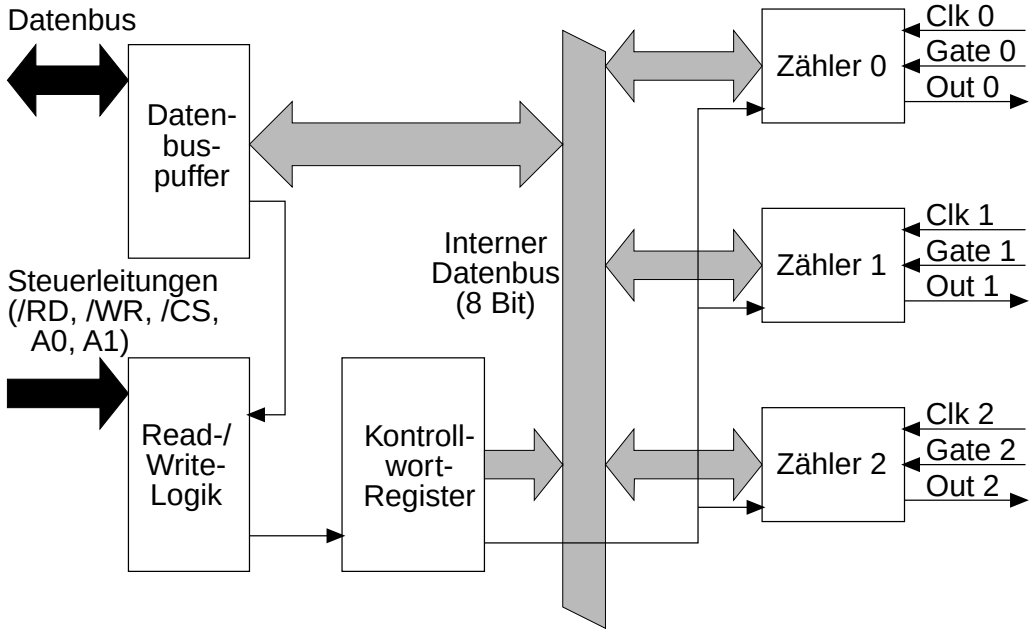


Abb. 5: Blockschaltbild des 82C54

3.4 Beschaltung

Beachten Sie, ...
daß sowohl die digitalen Ein-/Ausgangs-Ports als auch die Zählersignale optoisoliert und für die in der Steuerungstechnik üblichen 24 V ausgelegt sind.

3.4.1 Eingangsbeschaltung ME-81

Die 16 optoisolierten Eingangskanäle (DI 0…15) der ME-81 sind über die Vorwiderstände R_V auf Optokoppler geführt. Die Vorwiderstände sind für einen Spannungs-High-Pegel von typ. 24 V dimensioniert. Bei Bedarf, ist ein Betrieb mit TTL-Pegel durch Anpassung der Vorwiderstände R_V möglich. Ein Bezug zur Masse des digitalen Eingangsteils (ext. GND DI-Teil) muß stets hergestellt werden.

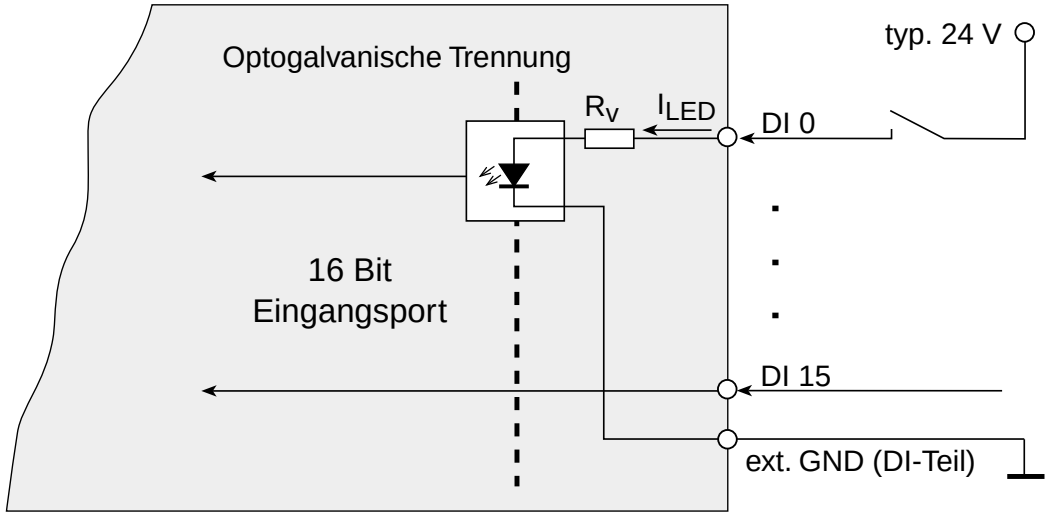


Abb. 6: Eingangsbeschaltung der ME-81

3.4.2 Ausgangsbeschaltung ME-81

Die 16 optoisolierten Ausgangskanäle (DO 0…15) der ME-81 sind mit zwei Treiberbausteinen vom Typ ULN2803 mit low aktiven Open-Collector Ausgängen (Sink-Treiber) ausgestattet. Ein Bezug zur Masse des digitalen Ausgangsteils (ext. GND DO-Teil) muß stets hergestellt werden.

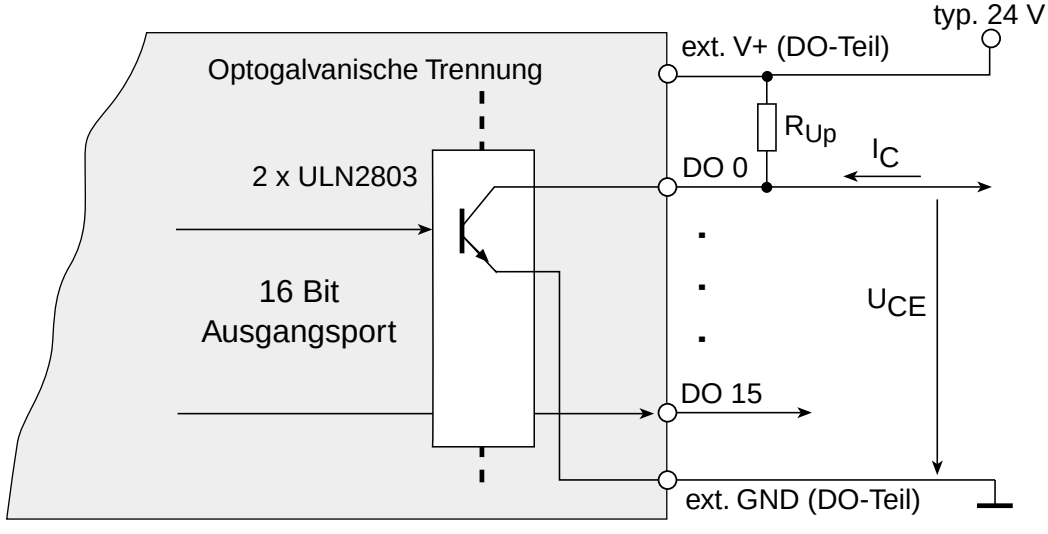


Abb. 7: Ausgangsbeschaltung der ME-81

Der maximale Strom pro Ausgang (I_C) hängt von der Sättingungsspannung U_{CE} ab und wird von der Verlustleistung der Summe der Kanäle auf P_{tot} = 1 W pro Baustein beschränkt (DO 0…7 =

Baustein 1, DO 8…15 = Baustein 2), siehe Abb. 8: "Sättigungsspannung ULN2803".

$$P_{tot} = P_0 + \ldots + P_7 \leq 1W \text{ (pro Baustein bei 70°C)}$$
$$\text{wobei } P_0 = I_{C0} \cdot U_{CE0}$$

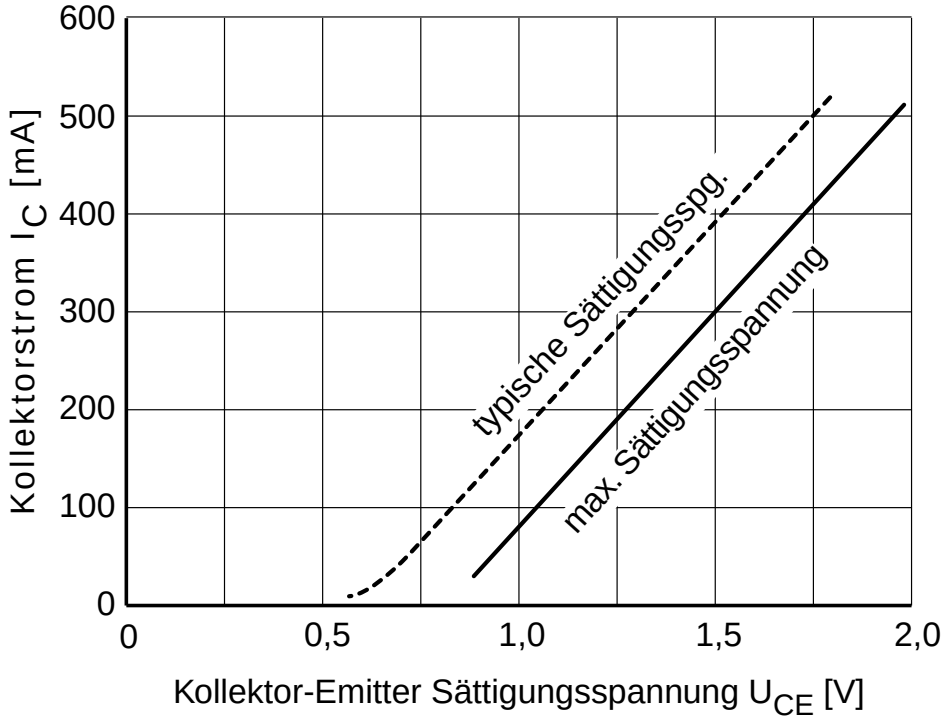


Abb. 8: Sättigungsspannung ULN2803

3.4.3 Eingangsbeschaltung ME-8100

Die 16 bzw. 32 optoisolierten Eingangskanäle (DI_A 0…15/ DI_B 0…15) der ME-8100A/B sind über die Vorwiderstände R_v auf Optokoppler geführt. Die Vorwiderstände sind für einen Spannungs-High-Pegel von typ. 24 V (R_v = 2,2kΩ) dimensioniert. Zum Schutz der Optokoppler wurde eine Schutzdiode zur Spannungsbegrenzung auf 26 V vorgesehen. Bei Bedarf, ist ein Betrieb mit TTL-Pegel durch Anpassung der Vorwiderstände R_v sowie der Schutzdioden möglich. Ein Bezug zur externen Masse (ext. GND) muß stets hergestellt werden.

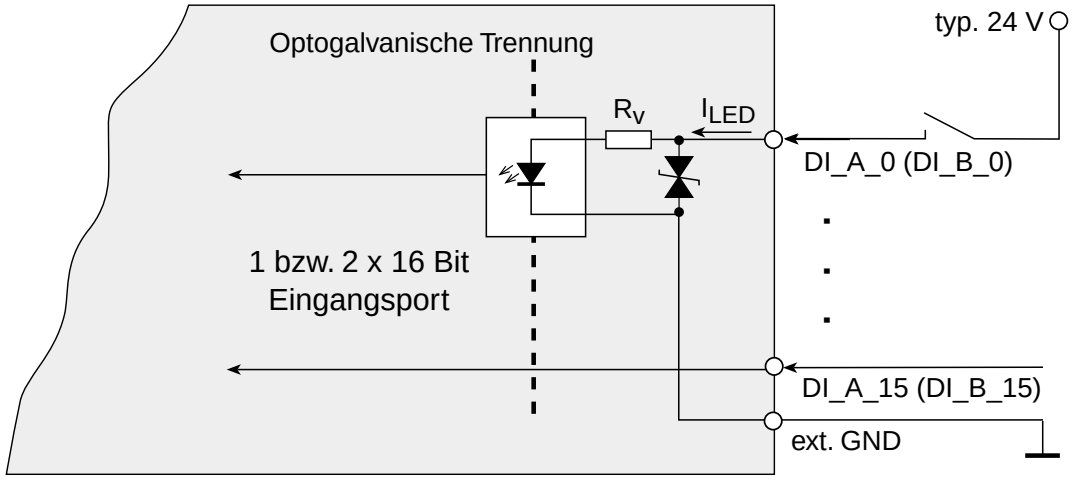


Abb. 9: Eingangsbeschaltung der ME-8100

3.4.4 Ausgangsbeschaltung ME-8100

Die 16 bzw. 32 optoisolierten Ausgangskanäle (DO_A 0…15/ DO_B 0…15) der ME-8100A/B sind mit 2 bzw. 4 Treiberbausteinen realisiert. Je nach Anwendungsfall hat der Anwender die Möglichkeit per Software zwischen low-aktiven Ausgängen (Sink-Treiber vom Typ ULN2803; Standard-Einstellung) und high-aktiven Ausgängen (Source-Treiber vom Typ UDN2982) zu wählen. Ein Bezug zur Masse des digitalen Ausgangsteils (ext. GND) muß stets hergestellt werden.

Sink-Treiber:

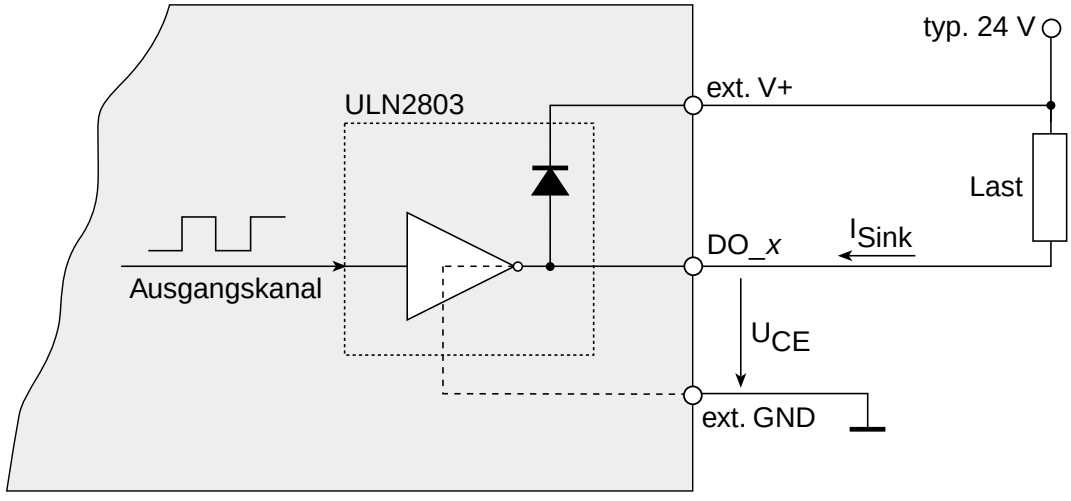


Abb. 10: Ausgänge der ME-8100 mit Sink-Treibern

Der maximale Strom pro Ausgang ($I_C = I_{Sink}$) hängt von der Sättingungsspannung U_{CE} ab und wird von der Verlustleistung der Summe der Kanäle auf P_{tot} = 1 W pro Baustein beschränkt (DO_*x* 0…7 = Baustein 1, DO_*x* 8…15 = Baustein 2, usw.), siehe Abb. 11: "Sättigungsspannung ULN2803".

$$P_{tot} = P_0 + \ldots + P_7 \leq 1W \text{ (pro Baustein bei 70°C)}$$
$$\text{wobei } P_0 = I_{C0} \cdot U_{CE0}$$

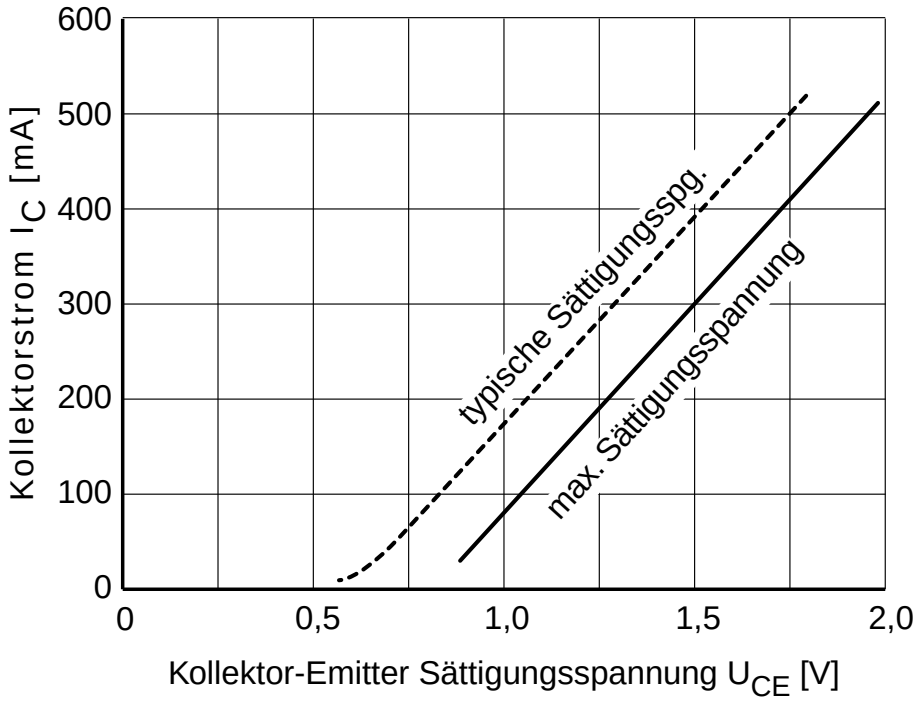


Abb. 11: Sättigungsspannung ULN2803

Source-Treiber:

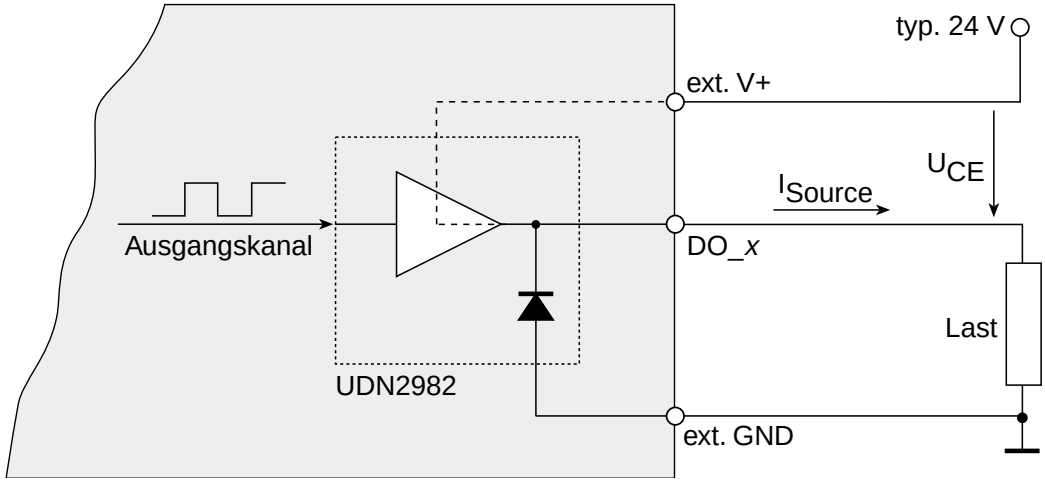


Abb. 12: Ausgänge der ME-8100 mit Source-Treibern

Den maximalen Strom pro Ausgang ($I_C = I_{Source}$) entnehmen sie bitte der folgenden Tabelle. Die Verlustleistung der Summe der Kanäle darf P_{tot} = 0,7 W pro Baustein nicht übersteigen (DO_*x* 0…7 = Baustein 1, DO_*x* 8…15 = Baustein 2, usw.).

$$P_{tot} = P_0 + \ldots + P_7 \leq 0{,}7 \text{ W (pro Baustein bei 70°C)}$$
$$\text{wobei } P_0 = I_{C0} \cdot U_{CE0} \text{ mit } U_{CE} = \text{typ. } 1{,}8 \text{ V}$$

	Anzahl der benutzten Kanäle							
	1	2	3	4	5	6	7	8
I_{Cmax} [mA]	350	175	115	85	70	55	50	40

Tabelle 3: Max. Strom der Source-Treiber

3.4.5 Zählerbeschaltung ME-8100

Die „Clk“-, „Gate“- und „Out“-Leitungen sind auf der ME-8100 bis 1000 V opto-galvanisch isoliert. Die Zählerausgänge sind mit Pull-Up Widerständen (R_{UP} = 2,2kΩ) bestückt. Sämtliche Zählersignale sind für die in der Steuerungstechnik üblichen 24 V (R_v = 2,2kΩ) ausgelegt.

Zur Freigabe des Zählers muß am Gate-Eingang Low-Pegel angelegt werden. Je nach Konfiguration wird der Gate-Eingang als Enable- oder Triggersignal verwendet.

3.4.5.1 Beschaltung Zähler-Eingänge

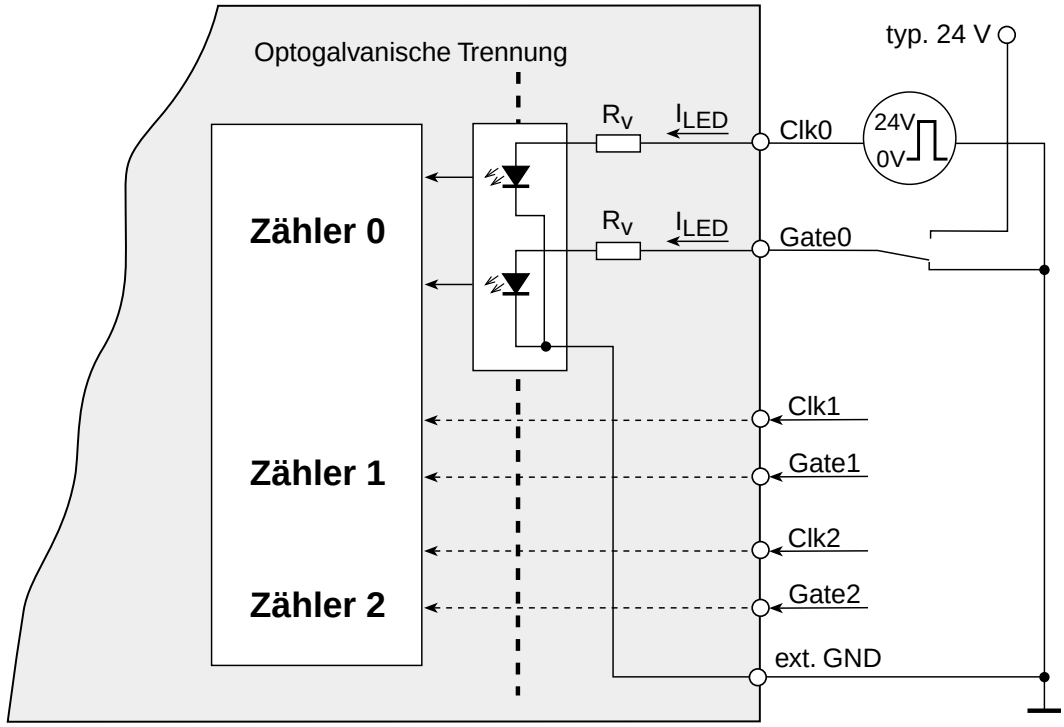


Abb. 13: Beschaltung der Zähler-Eingänge ME-8100A/B

3.4.5.2 Beschaltung Zähler-Ausgänge

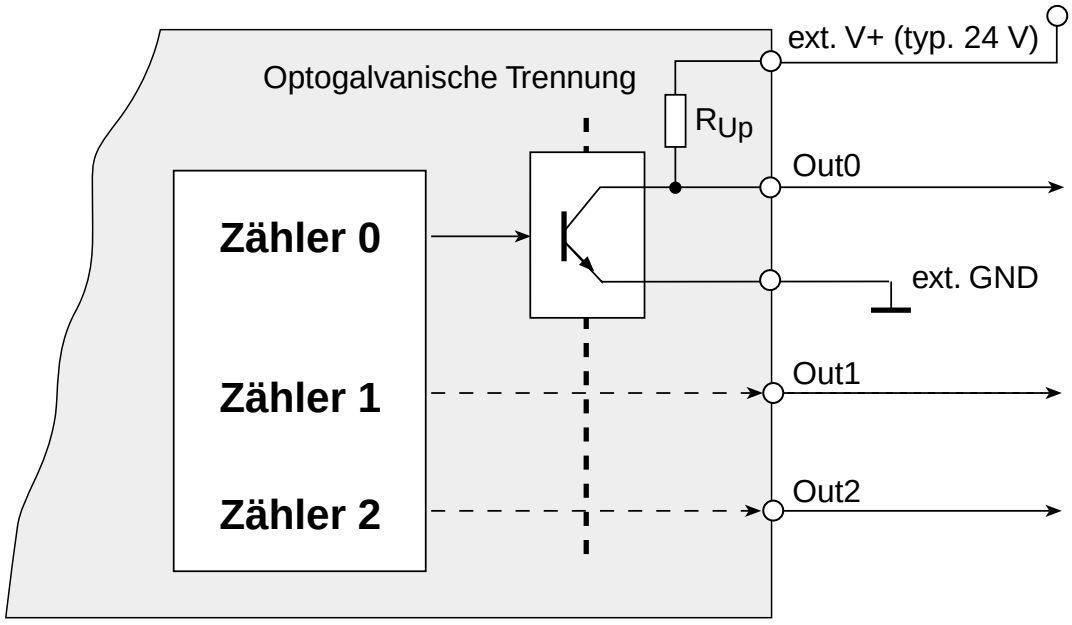


Abb. 14: Beschaltung der Zähler-Ausgänge ME-8100A/B

3.4.5.3 Zählertakt

Über die Eingangsleitungen Clk 0…2 muß ein externer Takt zur Speisung der einzelnen Zähler zugeführt werden. Die max. Zählfrequenz beträgt 1 MHz.

3.4.5.4 Kaskadierung der Zähler

Zur Kaskadierung der Zähler können die Ausgänge des/der Zählerbausteine(s) durch externe Verdrahtung „in Reihe“ geschaltet werden.

Sollen zum Beispiel die Zähler 0…2 kaskadiert werden, so sind durch geeignete Beschaltung folgende Verbindungen vorzunehmen:

- Den Takt-Eingang des Zählers 0 (Clk 0) mit dem Ausgangstakt verbinden
- Den Ausgang von Zähler 0 (Out 0) mit dem Takt-Eingang des Zählers 1 (Clk 1) verbinden
- Den Ausgang von Zähler 1 (Out 1) mit dem Takt-Eingang des Zählers 2 (Clk 2) verbinden
- Außerdem müssen die Gate-Eingänge (Gate 0…2) zur Freigabe der Zähler mit 0 V beschaltet werden werden.
- Am Ausgang des Zählers 2 (Out 2) steht das kaskadierte Zählersignal zur Verfügung.

3.5 Registerbeschreibung

3.5.1 Register der ME-81 ISA

Der Registersatz der ME-81 umfaßt 12 aufeinanderfolgende Bytes im I/O-Adreßraum des PCs. Sämtliche Register sind 16 Bit breit und in den folgenden Tabellen kurz beschrieben (R = Lesen, W = Schreiben):!

Offset	16 BitRegister	
BA+00H	**R**	**Function ID + INTR-BIT** enthält Informationen/ID zur Karte: *b15…b9* reserviert *b8 (INTR)* Wird gesetzt, falls Ereignis für Interrupt eingetreten ist (in den Betriebsarten Bitmuster-Vergleich und Bitmuster-Änderung, INTB0 = 1); aber auch falls Interupt deaktiviert (z. B. für Polling-Betrieb, INTB0 = 0) *b7 (FID7)* reserviert *b6 (FID6)* Funktionsgruppe Vergleichspattern *b5 (FID5)* Funktionsgruppe AI (A/D-Teil) *b4 (FID4)* Funktionsgruppe AO (D/A-Teil) *b3 (FID3)* Funktionsgruppe DIO (Digital I/O-Teil) *b2…b0 (FID2…FID0)* PROM-Version Daraus ergibt sich folgende Function-ID für PROM-Version 1: 01001001: 49Hex

Tabelle 4: Adreßraum der ME-81

Offset		16 BitRegister
BA+00H	**W**	**Kontroll-Register** *b15…b8* reserviert *Steuerung der digitalen Ports:* *b7 . . . . . .ENIO* 1 Digitale Ports aktiv 0 Digitale Ports hochohmig *Auswahl der Interruptquelle:* *b6, b5 . . . .INTB1, INTB0* 0 0 Interrupt deaktiviert 0 1 Interrupt bei Bitmuster-Gleichheit 1 0 Interrupt deaktiviert 1 1 Interrupt bei Bitmuster-Änderung *b4, b0* reserviert
BA+02H	**R**	**Reset INTR-Bit** Durch Lesen dieser Adresse wird das INTR-Bit rückgesetzt. Solange INTR-Bit gesetzt ist, kann kein neuer Interrupt erzeugt werden
BA+04H	**R**	**Eingangsregister** Die Eingänge stehen nach ca. 100 ns zur Verfügung (Entprellung)
BA+06H	**W**	**Ausgangsregister** Vorraussetznung: ENIO = „1“ Register bereits vorher beschreibbar
BA+08H	**W**	**Vergleichs-Register** Ein am Eingang anliegendes Bitmuster wird mit dem in diesem Register abgelegten Vergleichsbitmuster verglichen. Zur Interrupt-Behandlung müssen die Interrupt-Bits INTB1 und INTB0 im Kontroll-Register entsprechend gesetzt sein.

Tabelle 4: Adreßraum der ME-81

Offset	16 BitRegister	
BA+0AH	**W**	**Maskenregister** Durch Setzen der entsprechenden Bits im Maskenregister werden die Eingänge ausgewählt, die im Bitänderungsmodus berücksichtigt werden sollen. Als Bezug für eine Bitmuster-Änderung dient dabei der aktuelle Wert im Eingangsregister. Ein Interrupt wird ausgelöst, wenn sich der Zustand von mindestens einem Eingang ändert. Zur Interrupt-Behandlung müssen die Interrupt-Bits INTB1 und INTB0 im Kontroll-Register entsprechend gesetzt werden.
	R	Nach einem Interrupt wird der zum Interrupt-Zeitpunkt am Eingang anliegende digitale Wert gespeichert und kann unter dieser Adresse eingelesen werden.

Tabelle 4: Adreßraum der ME-81

3.5.2 Register des 82C54

Die Beschreibungen zum Zählerbaustein 82C54 in den folgenden Abschnitten gelten für die Modelle ME-8100A/B. Zur Programmierung verwenden Sie bitte die mitgelieferte Funktionsbibliothek.

Das Kontrollwort legt den Betriebsmodus und das Zählsystem (binär oder BCD-Code) des Zählerbausteins fest und kontrolliert das Laden der Zähler-Register.

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

Tabelle 5: Format des Kontrollwortes

Mit den Bits SC1/0 wird der Zähler ausgewählt:

SC1	SC0	Funktion
0	0	Zähler 0
0	1	Zähler 1
1	0	Zähler 2
1	1	nicht erlaubt

Tabelle 6: Auswahl des Zählers

Mit den Bits RL1/0 wird der Read-/Write-Modus ausgewählt:

RL1	RL0	Funktion
0	0	Zähler Latching Operation
1	0	nur MSB
0	1	nur LSB
1	1	erst LSB, dann MSB (2 x 8 Bit)

Tabelle 7: Auswahl des Read-/Write-Modus

Mit den Bits M0…M2 wird der Betriebsmodus für die einzelnen Zähler bestimmt:

M2	M1	M0	Funktion
0	0	0	Modus 0
0	0	1	Modus 1
X	1	0	Modus 2
X	1	1	Modus 3
1	0	0	Modus 4
1	0	1	Modus 5

Tabelle 8: Auswahl des Betriebs-Modus

Mit dem Bit BCD wird das Zählsystem jedes einzelnen Zählers festgelegt:

BCD	Funktion
0	16 Bit Binär-Zähler
1	BCD-Zähler (4 Dekaden)

Tabelle 9: Auswahl des Zählsystems

Zunächst wird der Zähler initialisiert, indem man das Kontrollwort mit Zählernummer, Read/Write Modus, Zählsystem und der Modusnummer schreibt. Danach lädt man den Startwert des Zählers ins entsprechende Zählregister. Der Zählerstand wird mit jeder negativen Flanke des Clk-Signals um 1 dekrementiert. Im folgenden sind die sechs zur Verfügung stehenden Modi kurz erklärt.

3.5.2.1 Modus 0: Zustandsänderung bei Nulldurchgang

Diese Betriebsart ist z. B. zur Signalisierung eines Interrupts geeignet. Der Zähler-Ausgang (Out 0…2) geht in den Low-Zustand sobald der Zähler initialisiert wird oder ein neuer Startwert in den Zähler geladen wird. Zur Freigabe des Zählers muß am Gate-Eingang Low-Pegel liegen. Sobald der Zähler geladen und freigegeben wurde beginnt er abwärts zu zählen, während der Ausgang gesperrt bleibt.

Bei Erreichen des Nulldurchganges geht der Ausgang in den High-Zustand und bleibt dort bis der Zähler neu initialisiert wird oder ein neuer Startwert geladen wird. Auch nach Erreichen des Nulldurchganges wird weiter abwärts gezählt. Sollte während des Zählvorganges ein Zählerregisters erneut geladen werden, hat dies zur Folge, daß

1. beim Schreiben des ersten Bytes der momentane Zählvorgang gestoppt wird
2. beim Schreiben des zweiten Bytes der neue Zählvorgang gestartet wird.

3.5.2.2 Modus 1: Retriggerbarer „One-Shot“

Der Zähler-Ausgang (Out 0…2) geht in den High-Zustand sobald der Zähler initialisiert wird. Nachdem ein Startwert in den Zähler geladen wurde geht der Ausgang mit dem auf den ersten Triggerimpuls (negative Flanke am Gate-Eingang) folgenden Takt in den Low-Zustand. Nach Ablauf des Zählers geht der Ausgang wieder in den High-Zustand.

Mit einer negativen Flanke am Gate-Eingang kann der Zähler jederzeit auf den Startwert zurückgesetzt („retriggered“) werden. Der Ausgang bleibt solange im Low-Zustand bis der Zähler den Nulldurchgang erreicht.

Der Zählerstand kann jederzeit ohne Auswirkung auf den momentanen Zählvorgang, ausgelesen werden.

3.5.2.3 Modus 2: Asymmetrischer Teiler

In diesem Modus arbeitet der Zähler als Frequenzteiler. Der Zähler-Ausgang (Out 0…2) geht nach der Initialisierung in den High-

Zustand. Nach Freigabe des Zählers durch einen Low-Pegel am Gate-Eingang wird abwärts gezählt, während der Ausgang noch im High-Zustand bleibt. Sobald der Zähler den Wert 0001Hex erreicht hat, geht der Ausgang für die Dauer einer Taktperiode in den Low-Zustand. Dieser Ablauf wiederholt sich periodisch, solange das Gate-Signal im Low-Zustand ist, ansonsten geht der Ausgang sofort in den High-Zustand.

Wird das Zählerregister zwischen zwei Ausgangs-Pulsen erneut geladen, so beeinflußt dies den momentanen Zählvorgang nicht, die folgende Periode arbeitet jedoch mit den neuen Werten.

3.5.2.4 Modus 3: Symmetrischer Teiler

Dieser Modus arbeitet ähnlich wie Modus 2 mit dem Unterschied, daß der geteilte Takt ein symmetrisches Tastverhältnis besitzt (nur für geradzahlige Zählerwerte geeignet). Der Zähler-Ausgang (Out 0…2) geht nach der Initialisierung in den High-Zustand. Nach Freigabe des Zählers durch einen Low-Pegel am Gate-Eingang wird in 2er-Schritten abwärts gezählt. Nun wechselt der Ausgang, beginnend mit High-Pegel alle Startwert/2-Perioden des Eingangstaktes seinen Zustand. Solange am Gate-Eingang ein Low-Pegel anliegt, wiederholt sich dieser Ablauf periodisch, ansonsten geht der Ausgang sofort in den High-Zustand.

Wird das Zählerregister zwischen zwei Ausgangs-Pulsen erneut geladen, so beeinflußt dies den momentanen Zählvorgang nicht, die folgende Periode arbeitet jedoch mit den neuen Werten.

3.5.2.5 Modus 4: Zählerstart durch Softwaretrigger

Der Zähler-Ausgang (Out 0…2) geht in den High-Zustand sobald der Zähler initialisiert wird. Zur Freigabe des Zählers muß am Gate-Eingang Low-Pegel liegen. Sobald der Zähler geladen (Softwaretrigger) und freigegeben wurde, beginnt er abwärts zu zählen, während der Ausgang noch im High-Zustand bleibt.

Bei Erreichen des Nulldurchganges geht der Ausgang für die Dauer einer Takt-Periode in den Low-Zustand. Danach geht der Ausgang wieder in den High-Zustand und bleibt dort bis der Zähler neu initialisiert wird und ein neuer Startwert geladen wird.

Wird das Zählerregister während eines Zählvorganges erneut geladen, so wird der neue Startwert mit dem nächsten Takt geladen.

3.5.2.6 Modus 5: Zählerstart durch Hardwaretrigger

Der Zähler-Ausgang (Out 0…2) geht in den High-Zustand sobald der Zähler initialisiert wird. Nachdem ein Startwert in den Zähler geladen wurde beginnt der Zählvorgang mit dem auf den ersten Trigger-impuls (negative Flanke am Gate-Eingang) folgenden Takt. Bei Erreichen des Nulldurchganges geht der Ausgang für die Dauer einer Takt-Periode in den Low-Zustand. Danach geht der Ausgang wieder in den High-Zustand und bleibt dort bis ein erneuter Triggerimpuls ausgelöst wird.

Wird das Zählerregister zwischen zwei Triggerimpulsen erneut geladen, so wird der neue Startwert erst nach dem nächsten Triggerimpuls berücksichtigt.

Mit einer negativen Flanke am Gate-Eingang kann der Zähler jederzeit auf den Startwert zurückgesetzt („retriggered“) werden. Der Ausgang bleibt solange im High-Zustand bis der Zähler den Nulldurchgang erreicht hat.

3.6 Testprogramm

Zum Test der Einsteckkarte wird ein einfaches Testprogramm mitgeliefert. Sie finden das jeweilige Testprogramm in einem entsprechenden Unterverzeichnis von `C:\Meilhaus` (Default). Das Testprogramm kann durch Doppelklick gestartet werden. (Vorraussetzung: Systemtreiber korrekt installiert).

4 Programmierung

Das Treiberkonzept für die ME-81 bzw. ME-8100 Familie unter Windows (32 Bit) bietet Ihnen die Möglichkeit, ohne Änderung bereits vorhandener Applikationssoftware anstatt einer „alten“ ME-81 ISA-Karte eine äquivalente PCI-Karte zu verwenden. Da Funktionsumfang und Syntax identisch sind, ist dies im Idealfall ohne Neucompilierung möglich. (Voraussetzung: „alte“ ISA-Karte und neue PCI-Karte verwenden den gleichen Wert im Parameter <iBoardNumber>). Zur genauen Vorgehensweise beachten Sie bitte die entsprechenden README-Dateien auf der Installationsdiskette des ME-8100 Treibersystems.

Die Austauschbarkeit von ISA- gegen PCI-Karten gilt nur, wenn Sie zur Programmierung Ihrer ISA-Karte die Funktionsbibliothek des ME-81 Treibers verwendet haben!

4.1 Hochsprachenprogrammierung

Folgende Hochsprachen werden standardmäßig unterstützt:

- Visual C++ ab Version 4.0.
- Delphi ab Version 2.0.
- VisualBASIC ab Version 4.0.
- Für weitere Infos beachten Sie bitte die entsprechenden README-Dateien auf der ME-Power-CD.

Es ist darauf zu achten, daß für den Compiler und Linker die Pfade auf diese Dateien richtig gesetzt sind.

Durch Einbinden der hochsprachenspezifischen Definitionsdatei in Ihr Projekt können Sie viele Parameter in Form vordefinierter Konstanten und Makros übergeben. Alternativ ist die direkte Übergabe des entsprechenden Hex-Wertes jederzeit möglich.

4.1.1 Vorgehensweise

Zur Implementierung des Interrupt-Betriebs für die ME-8100 beachten Sie bitte folgende Vorgehensweise:

Hinweis: Interrupt-Service-Funktionen werden unter VisualBASIC nicht unterstützt.

4.1.1.1 Interrupt bei Bitmuster-Gleichheit

1. Rufen Sie die Funktion …*DIOSetPattern* auf. Der Parameter <Pattern> definiert das gewünschte Bitmuster. Sobald das gleiche Bitmuster am Eingang anliegt wird ein Interrupt ausgelöst.
2. Rufen Sie die Funktion …*DIOSetTristateOFF* auf.
3. Rufen Sie die Funktion …*SetSinkSourceMode* auf und schalten Sie z. B. die Source-Treiber aktiv (nicht nötig für ME-81).
4. Rufen Sie die Funktion …*SetIntMode* auf. Der Interrupt-Modus muß auf INTERRUPT_ON_PATTERN_COMPARE gesetzt werden.
5. Rufen Sie die Funktion …*EnableInt* um den Interrupt-Betrieb freizuschalten. Sobald das gewünschte Bitmuster am Eingang anliegt wird ein Interrupt ausgelöst und die Interrupt Service Routine ausgeführt.
6. Durch Aufruf der Funktion …*GetIrqCnt*, kann die Anzahl der aufgetretenen Interrupts ermittelt werden. (Funktion steht für ME-81 nicht zur Verfügung).
7. Rufen Sie die Funktion …*DisableInt* auf.

4.1.1.2 Interrupt bei Bitmuster-Änderung

1. Rufen Sie die Funktion …*DIOSetTristateOFF* auf.
2. Rufen Sie die Funktion …*SetSinkSourceMode* auf und schalten Sie z. B. die Source-Treiber aktiv (nicht nötig für ME-81).
3. Rufen Sie die Funktion …*SetIntMode* auf. Der Interrupt-Modus muß auf INTERRUPT_ON_BIT_CHANGE gesetzt werden.

4. Rufen Sie die Funktion …*DIOSetMask*. Der Wert im Parameter <Mask> definiert welche Bits überwacht werden sollen. Sobald eines dieser Bits seinen Zustand ändert, wird ein Interrupt ausgelöst. Verwenden Sie z. B. die Bitmaske 0xFFFF um alle Bits zu überwachen.

5. Rufen Sie die Funktion …*EnableInt* um den Interrupt-Betrieb freizuschalten.

6. Durch Aufruf der Funktion …*GetIrqCnt*, kann die Anzahl der aufgetretenen Interrupts ermittelt werden. (Funktion steht für ME-81 nicht zur Verfügung).

7. Durch Aufruf der Funktion …*DIGetIntStatus*, kann das Bitmuster, das den Interrupt ausgelöst hat über den Parameter <BitValue> eingelesen werden.

8. Rufen Sie die Funktion …*DisableInt* auf.

Hinweis: An den Eingangs-Pins sollte ein bekanntes Bitmuster anliegen. Dieses Bitmuster definiert den neutralen Zustand. Sobald sich der Zustand eines oder mehrerer Bits ändert wird ein Interrupt ausgelöst. Durch Übergabe des Wertes 0xFFFF im Parameter <Mask> der Funktion …*DIOSetMask* werden alle Bits überwacht. Falls nur einzelne Bits überwacht werden sollen (z. B. <Mask> = 0x000F für die 4 niederwertigen Bits) so löst die Zustandsänderung eines anderen Bits **keinen** Interrupt aus. Nur die Zustandsänderung eines Bits, das im Parameter <Mask> auf „1“ gesetzt wurde, wird einen Interrupt auslösen.

4.1.2 Beispielprogramme

Zum leichteren Verständnis der Programmierung werden einfache Beispiele und kleine Projekte im Source-Code mitgeliefert. Die Beispielprogramme finden Sie im ME Software Developer Kit (ME-SDK), das standardmäßig ins Verzeichnis `C:\Meilhaus\me-sdk` installiert wird. Bitte beachten Sie die Hinweise in den entsprechenden README-Dateien.

4.2 Agilent VEE-Programmierung

Die Agilent VEE-Komponenten für Ihre Karte finden Sie auf der „ME-Power-CD“ oder zum Download unter www.meilhaus.de.

Das Meilhaus VEE Treibersystem unterstützt die HP VEE Vollversionen 4.x und 5.x, HP VEE Lab, Agilent VEE Pro und Agilent VEE OneLab. Zur Installation der VEE-Komponenten und für weitere Infos beachten Sie bitte die Dokumentation, die Sie mit dem VEE Treibersystem erhalten. Zu den Grundlagen der VEE-Programmierung benutzen Sie bitte Ihre VEE Dokumentation und die VEE Online-Hilfe.

4.2.1 User Objects

Zur komfortableren Handhabung des Treibers wurden vordefinierte User Objects erstellt, welche intern API-Funktionen aufrufen. Diese sind über den zusätzlichen Menüpunkt „ME Board“ in der VEE-Entwicklungsumgebung aufrufbar und können – wie andere Standard-Funktionen von VEE auch – in der Entwicklungsumgebung plaziert und in einer Applikation „verdrahtet“ werden.

Die User Objects sind weitgehend selbsterklärend und basieren auf den im Kap. „Funktionsreferenz“ dokumentierten API-Funktionen. Zusätzlich gibt es noch sog. „Expanded User Objects“, um Ihnen das Programmieren so bequem wie möglich zu machen. Eine Kurzbeschreibung zum jeweiligen User Object finden Sie auch unter „Description“ indem Sie den Mauszeiger über das entsprechende UO bewegen und die rechte Maustaste drücken.

Die User Objects können für eigene Bedürfnisse jederzeit geändert, angepaßt und bei Bedarf als kundenspezifisches Objekt abgespeichert werden.

4.2.2 Demoprogramme

Zur Demonstration und zum leichteren Verständnis wurden kleine Demoprogramme erstellt, die alle wichtigen User Objects enthalten. Die Demoprogramme sind über den Menüpunkt „ME Board – Demos“ aufrufbar.

Die VEE-Demoprogramme enthalten teilweise auch Ergänzungen der „normalen“ User Objects und tragen zur leichteren Unterscheidung von diesen das Präfix "x…" im Dateinamen.

4.2.3 Das "ME Board"-Menü

Das Installationsprogramm erweitert die Menüleiste von VEE automatisch um den Eintrag „ME Board“. Dadurch ist eine komfortable Nutzung aller unter VEE zur Verfügung stehenden Treiberfunktionen möglich. Über das „ME Board“-Menü können Sie nach Kartenfamilien geordnet, die ent-sprechenden Treiber- und Demo-User Objects aufrufen.

Hinweis:
Der Installationsumfang der User Objects (UOs) richtet sich nach der von Ihnen gewählten Kartenfamilie(n) zu Beginn der VEE Treiber-Installation. Sollten Sie UOs im „ME Board“-Menü aufrufen, die jedoch nicht installiert wurden, so führt dies zur Fehlermeldung:

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

Bei Bedarf können Sie die benötigten VEE Komponenten jederzeit nachinstallieren (siehe „ME-Power-CD“).

4.3 LabVIEW™-Programmierung

Die LabVIEW™-Komponenten für Ihre Karte finden Sie auf der „ME-Power-CD“ oder zum Download unter www.meilhaus.de.

Die Meilhaus LabVIEW™-Treiber unterstützen die LabVIEW™ Vollversionen 4.x oder höher. Zur Installation der LabVIEW™-Komponenten und für weitere Infos beachten Sie bitte die Dokumentation, die Sie mit dem jeweiligen LabVIEW-Treiber erhalten. Zu den Grundlagen der LabVIEW™-Programmierung benutzen Sie bitte Ihre LabVIEW™ Dokumentation und die LabVIEW™ Online-Hilfe.

4.3.1 Virtual Instruments

Zur komfortableren Handhabung des Treibers wurden vordefinierte Virtual Instruments (VIs) erstellt. Diese sind über das Menü

„Datei - Öffnen“ in LabVIEW™ aufrufbar und können – wie andere Standard-VIs von LabVIEW™ auch – in der Entwicklungsumgebung plaziert und in einer Applikation „verdrahtet“ werden.

Die „Source VIs“ sind weitgehend selbsterklärend und basieren auf den im Kap. „Funktionsreferenz“ dokumentierten API-Funktionen. Zusätzlich gibt es noch sog. „Expanded Virtual Instruments“, um Ihnen das Programmieren so bequem wie möglich zu machen.

Eine Kurzbeschreibung zum jeweiligen VI finden Sie auch im VI „…Function Tree“. Dieses VI dient nur der Dokumentation und kann über das Menü „Datei - Öffnen“ aufgerufen werden. Unter „Description“ finden Sie eine Kurzbeschreibung zum jeweiligen VI.

Die VIs können für eigene Bedürfnisse jederzeit geändert, angepaßt und bei Bedarf als kundenspezifisches VI abgespeichert werden.

4.3.2 Demoprogramme

Zur Demonstration und zum leichteren Verständnis wurden kleine Demoprogramme erstellt, die alle wichtigen „Virtual Instruments“ (VIs) enthalten. Die Demoprogramme sind über das Menü „Datei – Öffnen“ aufrufbar.

4.4 Registerprogrammierung (ISA-Versionen)

Hinweis: Die ME-81 ISA kann, z. B. unter DOS, auch auf Registerebene programmiert werden. Dies ist von Computer-Hochsprachen aus durch Port-Ein-/Ausgabe-Anweisungen möglich (siehe Kap. 3.5 „Registerbeschreibung“ auf Seite 26). Informieren Sie sich bitte gegebenenfalls über die passende Befehlssyntax zur Portprogrammierung im Handbuch der Hochsprache Ihrer Wahl. Wir empfehlen jedoch die Verwendung der mitgelieferten Treibersoftware unter Windows (32 Bit), **ansonsten ist die Softwarekompatibilität zwischen ISA- und PCI/cPCI-Karten nicht gegeben!**

Beachten Sie bitte folgende Vorgehensweise:

4.4.1 Initialisierung

- Function ID Register (M81_FIDREG) auslesen. Dabei müssen die Bits b7…b3 stets den Wert '01001' (binär) haben, während die Bits b2…b0 von der jeweiligen PROM-Version abhängen. So gilt z. B. für die PROM-Versions-nummer 1 (001) folgender Eintrag (siehe Kap. 3.5 „Registerbeschreibung“ auf Seite 26):

  01001001Binär 49Hex.

  Wenn dies nicht der Fall ist, ist entweder die Basisadresse falsch oder die Karte defekt.

- Alle Aktionen auf der Karte stoppen: Kontrollregister (BA+00Hex) auf 00Hex setzen.
- INTR-Bit rücksetzen, durch „Dummylesen“ des Registers BA+02Hex.

4.4.2 Einfaches Einlesen

- Initialisierung (siehe 4.4.1)
- Festlegen der Konfiguration im Kontroll-Register (BA+00Hex)
- Interrupt deaktivieren INTB0 = 0
- Einlesen des Eingaberegisters (BA+04Hex)

4.4.3 Einfaches Ausgeben

- Initialisierung (siehe 4.4.1)
- Digitale Ausgänge leitend schalten ENIO = 1
- Beschreiben des Ausgangsregisters (BA+06Hex)

4.4.4 Interrupt durch Bitmuster-Vergleich

- Initialisierung (siehe 4.4.1)
- Wert in Vergleichsregister (BA+08Hex) schreiben.
- Festlegen der Konfiguration im Kontroll-Register (BA+00Hex)

- Auswahl der Interruptquelle
  Bitmuster-Vergleich: INTB1, INTB0 = 0, 1

- Interrupt-Auslösung erfolgt bei Übereinstimmung von Vergleichsregister und Bitmuster am Eingang. Der Wert, der den Interrupt ausgelöst hat wird im Register BA+0AH abgelegt und kann unter dieser Adresse eingelesen werden.
- INTR-Bit rücksetzen, durch „Dummylesen“ des Registers BA+02Hex.

4.4.5 Interrupt durch Bitmuster-Änderung

- Initialisierung (siehe 4.4.1)
- Es wird empfohlen, den Neutral-Wert vom Eingangsregister einzulesen
- Festlegen der Konfiguration im Kontroll-Register (BA+00Hex)
  - Auswahl der Interruptquelle
    Bitmuster-Änderung: INTB1, INTB0 = 1, 1
- Bitmaske in Maskenregister BA+0AHex schreiben.
- Interrupt-Auslösung erfolgt bei Zustands-Änderung ($0 \rightarrow 1$ oder $1 \rightarrow 0$) von mindestens einem Eingang der Karte bzgl. Neutral-Wert im Eingangsregister. (Vorraussetzung: Bit wurde im Maskenregister freigegeben, d. h. auf ˝1˝ gesetzt). Der Wert zum Interrupt-Zeitpunkt wird im Register BA+0AH abgelegt und kann unter dieser Adresse eingelesen werden.
- INTR-Bit rücksetzen, durch „Dummylesen“ des Registers BA+02Hex.

Nach jedem Interrupt muß das INTR-Bit durch „Dummylesen“ des Registers (BA+02Hex) zurückgesetzt werden, da sonst kein Interrupt mehr ausgelöst wird.

5 Funktionsreferenz

5.1 Allgemein

Das Treiberkonzept für die ME-81/8100 Familie sieht jeweils ein eigenständiges Treibersystem für ISA- und PCI/cPCI-Karten vor. Aufgrund des Treiberkonzepts können ISA-Karten grundsätzlich nur mit dem ME-8x Treibersystem angesprochen werden und PCI-Karten nur mit dem ME-8100 Treibersystem. Um bei Bedarf eine Kompatibilität von PCI-Karten zu bereits vorhandener Applikationssoftware zu erreichen sind im PCI-Treiber zusätzlich alle Funktionen des ISA-Treibers deklariert, die dann auf analog aufgebaute Funktionen des PCI-Treibers zugreifen. Wenn Sie von dieser Möglichkeit Gebrauch machen möchten, beachten Sie bitte die genaue Vorgehensweise in den entsprechenden README-Dateien.

Die Funktionen der API-DLL (`ME8x_32.DLL`) für die ME-81 werden von folgenden 32 Bit-Treibern unterstützt:

- VxD-Treiber (`ME8x_32.VXD`) für Windows 95/98/Me
- Kernel-Treiber (`ME8x_32.SYS`) für Windows 2000/XP/NT4.0

desweiteren benötigt: Dialog-DLL (`MEDLG32.DLL`).

Die Funktionen der API-DLL (`ME8100.DLL`) für die ME-8100 werden von folgenden 32 Bit-Treibern unterstützt:

- VxD-Treiber (`ME8100.VXD`) für Windows 95
- Kernel-Treiber (`ME8100.SYS`) für Windows NT4.0
- WDM-Treiber (`ME8100.SYS`) für Windows 98/Me/2000/XP

Nachdem der Treiber erfolgreich geladen wurde, ermöglichen die API-Funktionen einen komfortablen Zugriff auf die Hardware. Jede Funktion, die auf eine Karte der ME-81/8100 Familie zugreifen soll, benötigt zur Identifizierung der Karte einen Integerwert. In der nun folgenden Beschreibung der Funktionen ist dieser Parameter mit `<BoardNumber>` bezeichnet. Er spezifiziert die anzusprechende Karte, wobei folgendes gilt:

- Wertebereich ISA-Modelle: 0…3
- Wertebereich PCI/cPCI-Modelle: 0…31
- Wert für die erste Karte: 0
- Wert für die zweite Karte: 1
- Wert für die x-te Karte: x-1

5.2 Nomenklatur

Die API-Funktionen sind kartenspezifisch gehalten. Jede API-Funktion beginnt für Visual C und Delphi (Pascal) mit einem Unterstrich „_“; nicht jedoch in Borland C und Basic. Das Präfix jeder Funktion bezeichnet unmittelbar die Karte(n), für die die Funktion zutrifft. Für die Funktionsnamen wurden weitgehend „selbstredende“ Bezeichner verwendet. Jeder Funktionsname besteht aus aus einem kartentypspezifischen Präfix und mehreren Bestandteilen für die entsprechende Funktionsgruppe (z. B.“DI“ für digitale Eingabe).

_me81… Funktion nur gültig für die Karte ME-81

_me8x… Funktion nur gültig für die Karten ME-80 und ME-81

me8100… Funktion nur gültig für die Karten ME-8100A/B

Für die Funktionsbeschreibung gelten folgende Vereinbarungen:

Funktionsnamen	werden im Fließtext kursiv geschrieben z. B. me8100*GetBoardVersion*
<`Parameter`>	werden in spitzen Klammern in der Schriftart `Courier` geschrieben
<*Variablen*>	als Platzhalter für vordefinierte Konstanten werden kursiv geschrieben und in spitze Klammern gesetzt
[eckige Klammern]	Variablen in eckigen Klammern sind fallweise bzw. alternativ zu verwenden (siehe Parameterbeschreibung der jeweiligen Funktion)
`DATEINAMEN`	oder `PFADE` werden in Großbuchstaben in der Schriftart `Courier` geschrieben
`me8100…()`	Programmausschnitte sind in der Schriftart `Courier` geschrieben

Zur Kennzeichnung des Datentyps werden folgende Kennbuchstaben verwendet:

i… oder dw…	32-Bit Integer-Wert
s… oder w…	16-Bit Short-Wert
c… oder b…	8-Bit Character-Wert
p…	Zeiger auf Datentyp (i, s oder c)

5.3 Beschreibung der API-Funktionen

Die Funktionsbeschreibung ist nach den folgenden Funktionsgruppen geordnet; innerhalb einer Funktionsgruppe gilt alphabetische Reihenfolge:

Funktion	Kurzbeschreibung	Seite
Allgemeine Funktionen		
me8100GetBoardVersion	Kartenversion ermitteln	48
_me8xGetDLLVersion me8100GetDLLVersion	DLL-Versionsnummer ermitteln	48
_me8xPROMVersion me8100PROMVersion	PROM-ID der Karte ermitteln	49
me8100SetSinkSourceMode	Umschaltung Sink/Source-Treiber	50
Digitale Ein-/Ausgabe		
_me8xDIGetBit me8100DIGetBit	Bit einlesen	51
_me81DIGetIntStatus me8100DIGetIntStatus	Bitmuster, das Interrupt ausgelöst hat einlesen	52
_me8xDIGetWord me8100DIGetWord	Word (16 Bit) einlesen	53
_me8xDIOSetIntMode me8100DIOSetIntMode	Interrupt-Ereignis auswählen	54
_me81DIOSetMask me8100DIOSetMask	Bitmaske schreiben	55
_me8xDIOSetPattern me8100DIOSetPattern	Vergleichsbitmuster schreiben	56
_me8xDIOSetTristateOFF me8100DIOSetTristateOFF	Ausgangsport leitend schalten	57

Tabelle 10: Übersicht der Bibliotheksfunktionen

Funktion	Kurzbeschreibung	Seite
_me8xDIOSetTristateON me8100DIOSetTristateON	Ausgangsport hochohmig schalten	58
_me8xDOSetBit me8100DOSetBit	Bit ausgeben	59
_me8xDOSetWord me8100DOSetWord	Word (16 Bit) ausgeben	60
Zählerfunktionen		
me8100CntRead	Aktuellen Zählerstand einlesen	61
me8100CntWrite	Zähler konfigurieren und Startwert laden	62
Interrupt-Handling		
_me8xDisableInt me8100DisableInt	Interruptsteuerung deaktivieren	63
_me8xEnableInt me8100EnableInt	Interruptsteuerung aktivieren	64
me8100GetIrqCnt	Anzahl der Interrupts ermitteln	66
Fehler-Behandlung		
_me8xGetDrvErrMess me8100GetDrvErrMess	Fehlerstring gemäß Fehlercode	67

Tabelle 10: Übersicht der Bibliotheksfunktionen

Hinweis: Stellvertretend für die Präfixe *_me81* und *_me8x* (für ME-81) und *me8100* (für ME-8100) wird in der folgenden Funktionsbeschreibung das Präfix *me8xxx* verwendet. Entnehmen Sie die jeweils gültigen Präfixe der Überschrift einer jeden Funktion (umrandeter Kasten). Die meisten Funktionen für die ME-8100 wurden um den Parameter <`iRegisterSet`> erweitert, um auf einfache Weise den identisch aufgebauten B-Teil der ME-8100B ansprechen zu können (dieser Parameter muß für die ISA-Variante ersatzlos gestrichen werden).

5.3.1 Allgemeine Funktionen

me8100me8100GetBoardVersion

✎ Beschreibung

Funktion gilt für die Modelle: ME-8100A/B.

Es wird die Kartenversion für eine installierte Karte der Kartenfamilie ME-8100 ermittelt.

● Definitionen

C:	int me8100GetBoardVersion (int iBoardNumber, int *piDevices)
Delphi:	Function me8100GetBoardVersion (iBoardNumber: integer; Var iDevices: integer): integer;
Basic:	Declare Function me8100GetBoardVersion Lib "me8100" Alias "_VBme8100GetBoardVersion@8" (ByVal iBoardNumber As Long, iDevices As Long) As Long

➔ Parameter

`<BoardNumber>`	Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)
`<Version>`	Zeiger auf Integer-Variable, in der die Device-ID zurückgegeben wird. Mögliche Werte sind: 810AHex: ME-8100A 810BHex: ME-8100B

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgege-- ben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über _me8xxxGetDrvErrMess ermittelt werden.)

_me8xGetDLLVersion
me8100GetDLLVersion

✎ Beschreibung

Funktion gilt für die Modelle: ME-81, ME-8100A/B.

Ermittelt die Versionsnummer der Treiber-DLL.

● Definitionen

C: int _me*8xxx*GetDLLVersion();

Delphi: Function _me*8xxx*GetDLLVersion: integer;

Basic: Declare Function me*8xxx*GetDLLVersion Lib "me*8xxx*_32" Alias "_VBme*8xxx*GetDLLVersion@0" () As Long

→ Parameter keine

‹ Rückgabewert

Versionsnummer. Der 32-Bit-Wert enthält in den höherwertigen 16 Bit die Hauptversion und in den niederwertigen 16 Bit die Unterversion. Beispiel: 0x00020001 ergibt die Version 2.01

_me8xPROMVersion
me8100PROMVersion

✎ Beschreibung

Funktion gilt für die Modelle: ME-81

Ermittelt die Kartenkennung und die karteninterne PROM-ID.

● Definitionen

C: int _me*8xxx*PROMVersion (int iBoardNumber, int *piVersion;)

Delphi: Function _me*8xxx*PROMVersion (iBoardNumber: integer; Var iVersion: integer): integer;

Basic: Declare Function me*8xxx*PROMVersion Lib "me*8xxx*_32" Alias "_VBme*8xxx*PROMVersion@8" (ByVal iBoardNumber As Long, ByRef iVersion As Long) As Long

→ Parameter

`<BoardNumber>` Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)

`<Version>` Zeiger auf Integer-Variable, in der die PROM-ID bzw. Kartenversion zurückgegeben wird. Der Wert wird hexadezimal interpretiert:

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

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xxxGetDrvErrMess* ermittelt werden.

me8100SetSinkSourceMode

✎ Beschreibung

Funktion gilt für die Modelle: ME-8100A/B.

Diese Funktion schaltet portweise entweder die Sink- oder Sourcetreiber an den Ausgängen ein (siehe „Ausgangsbeschaltung ME-8100” auf Seite 21). Nach Treiberstart ist der Sink-Treiber ausgewählt. Außerdem muß nach Treiberstart die Funktion *me8100DIOSetTristateOFF* pro Port einmal aufgerufen werden.

● Definitionen

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

→ Parameter

`<BoardNumber>` Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)

`<RegisterSet>` Auswahl des Registersatzes (für ME-8100A muß immer REGISTER_SET_A übergeben werden):

<A/B-Auswahl>	*Beschreibung*
REGISTER_SET_A (00Hex)	Registersatz für Teil A
REGISTER_SET_B (01Hex)	Registersatz für Teil B

`<Mode>` Auswahl der Ausgangstreiber-Bausteine:

<Sink/Source>	*Beschreibung*
SINK_MODE (00Hex)	Sink-Treiber aktiv
SOURCE_MODE (01Hex)	Source-Treiber aktiv

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xxxGetDrvErrMess* ermittelt werden.

5.3.2 Digitale Ein-/Ausgabe

_me8xDIGetBit
me8100DIGetBit

✎ Beschreibung

Funktion gilt für die Modelle: ME-81, ME-8100A/B

Liefert den Zustand einer einzelnen Eingangsleitung zurück.

● Definitionen

C: int _me*8xxx*DIGetBit (int iBoardNumber, [int iRegisterSet,] int iBitNo, int *piBitValue);

Delphi: Function _me*8xxx*DIGetBit (iBoardNumber, [iRegisterSet: integer;] iBitNo: integer; Var iBitValue: integer): integer;

Basic: Declare Function me*8xxx*DIGetBit Lib "me*8xxx*_32" Alias "_VBme*8xxx*DIGetBit@12" (ByVal iBoardNumber As Long, [ByVal iRegisterSet As Long,] ByVal iBitNo As Long, iBitValue As Long) As Long

➔ Parameter

`<BoardNumber>` Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)

`[RegisterSet]` (Parameter nicht in *_me8xDIGetBit*)

Auswahl des Registersatzes (für ME-8100A muß immer REGISTER_SET_A übergeben werden):

<A/B-Auswahl>	*Beschreibung*
REGISTER_SET_A (00Hex)	Registersatz für Teil A
REGISTER_SET_B (01Hex)	Registersatz für Teil B

`<BitNo>` Nummer der Eingangsleitung, die abgefragt werden soll; möglich sind:

0…15: 16-Bit-Eingangsport der ME-81 bzw. ME-8100

<BitValue> Zeiger auf einen Integerwert, der dann dem Leitungszustand entsprechend gelesen wird:
0: Eingang führt Low-Pegel
1: Eingang führt High-Pegel

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xxxGetDrvErrMess* ermittelt werden.

_me81DIGetIntStatus
me8100DIGetIntStatus

✎ Beschreibung

Funktion gilt für die Modelle: ME-81, ME-8100A/B

Mit dieser Funkiton kann das Eingangs-Bitmuster (16 Bit) eingelesen werden, das zum Auslösen eines Interrupts bei „Bitmustergleichheit“ bzw. „Bitmusteränderung“ geführt hat.

Diese Funktion wird nicht von Agilent VEE unterstützt!

● Definitionen

C: int _me*8xxx*DIGetIntStatus (int iBoardNumber, [int iRegisterSet,] int *iValue);

Delphi: Function _me*8xxx*DIGetIntStatus (iBoardNumber: integer; [iRegisterSet: integer;] Var iValue: integer): integer;

Basic: Declare Function me*8xxx*DIGetIntStatus Lib "me*8xxx*_32" Alias "_VBme*8xxx*DIGetIntStatus@8" (ByVal iBoardNumber As Long, [ByVal iRegisterSet As Long,] iValue As Long) As Long

→ Parameter

<BoardNumber> Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)

[RegisterSet] (Parameter nicht in *_me81DIGetIntStatus*)
Auswahl des Registersatzes (für ME-8100A muß immer REGISTER_SET_A übergeben werden):

<A/B-Auswahl>	*Beschreibung*
REGISTER_SET_A (00Hex)	Registersatz für Teil A
REGISTER_SET_B (01Hex)	Registersatz für Teil B

<Value> Zeiger auf gelesenen Integerwert; nur die niederwertigen 16 Bits signifikant.

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xxxGetDrvErrMess* ermittelt werden.

_me8xDIGetWord
me8100DIGetWord

✎ Beschreibung

Funktion gilt für die Modelle: ME-81, ME-8100A/B.

Liest ein Wort von einem 16-Bit Eingangsport der Karte.

● Definitionen

C: int _me*8xxx*DIGetWord (int iBoardNumber, [int iRegisterSet,] int iPortNo, int *iValue);

Delphi: Function _me*8xxx*DIGetWord (iBoardNumber: integer; [iRegisterSet: integer;] [iPortNo: integer;] Var iValue: integer): integer;

Basic: Declare Function me*8xxx*DIGetWord Lib "me*8xxx*_32" Alias "_VBme*8xxx*DIGetWord@12" (ByVal iBoardNumber As Long, [ByVal iRegisterSet As Long,] [ByVal iPortNo As Long,] iValue As Long) As Long

→ Parameter

`<BoardNumber>` Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)

`[RegisterSet]` (Parameter nicht in *_me8xDIGetWord*)

Auswahl des Registersatzes (für ME-8100A muß immer REGISTER_SET_A übergeben werden):

<A/B-Auswahl>	*Beschreibung*
REGISTER_SET_A (00Hex)	Registersatz für Teil A
REGISTER_SET_B (01Hex)	Registersatz für Teil B

[`PortNo`] (Parameter nicht in *me8100DIGetWord*)

hier muß PORTA (00Hex) für den 16 Bit-Eingangsport der ME-81 übergeben werden

<`Value`> Zeiger auf einen Integerwert mit gelesenem Wert; nur die niederwertigen 16 Bits signifikant.

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xxxGetDrvErrMess* ermittelt werden.

_me8xDIOSetIntMode
me8100DIOSetIntMode

✎ Beschreibung

Funktion gilt für die Modelle: ME-81, ME-8100A/B

Mit dieser Funktion kann die Interruptereignis ausgewählt werden. Diese Funktion wird nicht von Agilent VEE unterstützt!

● Definitionen

C: int _me*8xxx*DIOSetIntMode (int iBoardNumber, [int iRegisterSet,] int iMode);

Delphi: Function _me*8xxx*DIOSetIntMode (iBoardNumber: integer; [iRegisterSet: integer;] iMode: integer): integer;

Basic: Declare Function me*8xxx*DIOSetIntMode Lib "me*8xxx*_32" Alias "_VBme*8xxx*DIOSetIntMode@8" (ByVal iBoardNumber As Long, [ByVal iRegisterSet As Long,] ByVal iMode As Long) As Long

➔ Parameter

`<BoardNumber>` Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)

`[RegisterSet]` (Parameter nicht in *_me8xDIOSetIntMode*)
Auswahl des Registersatzes (für ME-8100A muß immer REGISTER_SET_A übergeben werden):

<A/B-Auswahl>	*Beschreibung*
REGISTER_SET_A (00Hex)	Registersatz für Teil A
REGISTER_SET_B (01Hex)	Registersatz für Teil B

<Mode> Interrupt-Modi;
mögliche Werte für die **ME-81** sind

<Modi ME-81>	Beschreibung
NO_INTERRUPT (00Hex)	kein Interruptbetrieb
INT_IF_PATTERN (20Hex)	Interrupt bei Bitmustergleichheit

INT_IF_MASK (60Hex)
Interrupt bei Bit-Änderung
mögliche Werte für die **ME-8100** sind

<Modi ME-8100>	Beschreibung
INTERRUPT_ON_PATTERN_COMPARE (00Hex)	Interrupt bei Bitmustergleichheit
INTERRUPT_ON_BIT_CHANGE (01Hex)	Interrupt bei Bitmusteränderung

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xxxGetDrvErrMess* ermittelt werden.

_me81DIOSetMask
me8100DIOSetMask

✎ Beschreibung

Funktion gilt für die Modelle: ME-81, ME-8100A/B.

Funktion schreibt ein 16-Bit-Wort zur Karte, das als Maske mit dem korrespondierenden Eingangsport verknüpft wird. Ändert ein mit '1' maskiertes Bit seinen Zustand, kann ein Interrupt ausgelöst werden (falls freigegeben). Beachten Sie zur Vorgehensweise Kap. 4.1.1.2 „Interrupt bei Bitmuster-Änderung“ auf Seite 36.

● Definitionen

C:	int _me*8xxx*DIOSetMask (int iBoardNumber, [int iRegisterSet,] int iMask);
Delphi:	Function _me*8xxx*DIOSetMask (iBoardNumber: integer; [iRegisterSet: integer;] iMask: integer): integer;
Basic:	Declare Function me*8xxx*DIOSetMask Lib "me*8xxx*_32" Alias "_VBme*8xxx*DIOSetMask@8" (ByVal iBoardNumber As Long, [ByVal iRegisterSet As Long,] ByVal iMask As Long) As Long

→ Parameter

`<BoardNumber>`	Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)

`[RegisterSet]` (Parameter nicht in *_me8xDIOSetMask*)
Auswahl des Registersatzes (für ME-8100A muß immer REGISTER_SET_A übergeben werden):

<A/B-Auswahl>	*Beschreibung*
REGISTER_SET_A (00Hex)	Registersatz für Teil A
REGISTER_SET_B (01Hex)	Registersatz für Teil B

`<Mask>` Maskenwert; die Zuordnung zu den Eingängen erfolgt bitweise; mögliche Werte: 0…65535 (0000Hex…FFFFHex)

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xxxGetDrvErrMess* ermittelt werden.

_me8xDIOSetPattern
me8100DIOSetPattern

✎ Beschreibung

Funktion gilt für die Modelle: ME-81, ME-8100A/B

Funktion schreibt ein 16-Bit-Wort als Vergleichs-Bitmuster zur Karte. Bei Bitmuster-Gleichheit mit dem korrespondierenden Eingangsport kann ein Interrupt ausgelöst werden (falls freigegeben). Beachten Sie zur Vorgehensweise Kap. 4.1.1.1 „Interrupt bei Bitmuster-Gleichheit“ auf Seite 36.

● Definitionen

C: int _me*8xxx*DIOSetPattern (int iBoardNumber, [int iRegisterSet,] int iPattern);

Delphi: Function _me*8xxx*DIOSetPattern (iBoardNumber: integer; [iRegisterSet: integer;] iPattern: integer): integer;

Basic: Declare Function me*8xxx*DIOSetPattern Lib "me*8xxx*_32" Alias "_VBme*8xxx*DIOSetPattern@8" (ByVal iBoardNumber As Long, [ByVal iRegisterSet As Long,] ByVal iPattern As Long) As Long

→ Parameter

`<BoardNumber>` Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)

`[RegisterSet]` (Parameter nicht in *_me8xDIOSetPattern*)
Auswahl des Registersatzes (für ME-8100A muß immer REGISTER_SET_A übergeben werden):

<A/B-Auswahl>	*Beschreibung*
REGISTER_SET_A (00Hex)	Registersatz für Teil A
REGISTER_SET_B (01Hex)	Registersatz für Teil B

`<Pattern>` Vergleichsmuster (16 Bit breit);
Eingänge DI_x 0…15 entsprechen den Bits 0…15 des Vergleichsregisters, mögliche Werte: 0…65535 (0000Hex…FFFFHex)

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xxxGetDrvErrMess* ermittelt werden.

_me8xDIOSetTristateOFF
me8100DIOSetTristateOFF

✎ Beschreibung

Funktion gilt für die Modelle: ME-81, ME-8100A/B.

Aktiviert die digitalen Ausgangsports der Karte. Vorher in die Register geschriebene Daten werden an die Ausgänge durchgeschaltet.

☞ Wichtiger Hinweis:

Nach einem Treiberstart muß diese Funktion vor allen Ausgabefunktionen einmalig aufgerufen werden.

● Definitionen

C: int _me*8xxx*DIOSetTristateOFF (int iBoardNumber, [int iRegisterSet]);

Delphi: Function _me*8xxx*DIOSetTristateOFF (iBoardNumber: integer[; iRegisterSet: integer]): integer;

Basic: Declare Function me*8xxx*DIOSetTristateOFF Lib "me*8xxx*_32" Alias "_VBme*8xxx*DIOSetTristateOFF@4" (ByVal iBoardNumber As Long[, ByVal iRegisterSet As Long]) As Long

➔ Parameter

`<BoardNumber>` Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)

`[RegisterSet]` (Parameter nicht in *_me8xDIOSetTristateOFF*)
Auswahl des Registersatzes (für ME-8100A muß immer REGISTER_SET_A übergeben werden):

<A/B-Auswahl>	*Beschreibung*
REGISTER_SET_A (00Hex)	Registersatz für Teil A
REGISTER_SET_B (01Hex)	Registersatz für Teil B

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xxxGetDrvErrMess* ermittelt werden.

_me8xDIOSetTristateON
me8100DIOSetTristateON

✎ Beschreibung

Funktion gilt für die Modelle: ME-80, ME-81, ME-8100A/B.

Schaltet die Ausgangsports der Karte in den hochohmigen Zustand. Nach einem Treiberstart befinden sich die Ausgänge ebenfalls im hochohmigen Zustand.

● Definitionen

C: int _me*8xxx*DIOSetTristateON(int iBoardNumber, [int iRegisterSet]);

Delphi: Function _me*8xxx*DIOSetTristateON(iBoardNumber: integer[; iRegisterSet: integer]): integer;

Basic: Declare Function me*8xxx*DIOSetTristateON Lib "me*8xxx*_32" Alias "_VBme*8xxx*DIOSetTristateON@4" (ByVal iBoardNumber As Long, [ByVal iRegisterSet As Long]) As Long

➔ Parameter

`<BoardNumber>` Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)

`[RegisterSet]` (Parameter nicht in *_me8xDIOSetTristateON*)
Auswahl des Registersatzes (für ME-8100A muß immer REGISTER_SET_A übergeben werden):

<A/B-Auswahl>	*Beschreibung*
REGISTER_SET_A (00Hex)	Registersatz für Teil A
REGISTER_SET_B (01Hex)	Registersatz für Teil B

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xxxGetDrvErrMess* ermittelt werden.

_me8xDOSetBit
me8100DOSetBit

✎ Beschreibung

Funktion gilt für die Modelle: ME-81, ME-8100A/B

Setzt eine einzelne digitale Ausgangsleitung in den gewünschten Zustand.

☞ Wichtiger Hinweis:

Zur Aktivierung der Ausgänge muß nach Treiberstart die Funktion *_me8xxxDIOSetTristateOFF* einmalig aufgerufen werden.

● Definitionen

C: int _me*8xxx*DOSetBit (int iBoardNumber, [int iRegisterSet,] int iBitNo, int iBitValue);

Delphi: function _me*8xxx*DOSetBit (iBoardNumber: integer; [iRegisterSet: integer;] iBitNo, iBitValue: integer): integer;

Basic: Declare Function me*8xxx*DOSetBit Lib "me*8xxx*_32" Alias "_VBme*8xxx*DOSetBit@12" (ByVal iBoardNumber As Long, [ByVal iRegisterSet As Long,] ByVal iBitNo As Long, ByVal iBitValue As Long) As Long

→ Parameter

`<BoardNumber>` Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)

`[RegisterSet]` (Parameter nicht in *_me8xDOSetBit*)
Auswahl des Registersatzes (für ME-8100A muß immer REGISTER_SET_A übergeben werden):

<A/B-Auswahl>	*Beschreibung*
REGISTER_SET_A (00Hex)	Registersatz für Teil A
REGISTER_SET_B (01Hex)	Registersatz für Teil B

`<BitNo>` Nummer der Ausgangsleitung, die gesetzt werden soll; möglich sind:
0…15: 16 Bit-Ausgangsport der ME-81 bzw. ME-8100

`<BitValue>` Mögliche Werte sind:
0: Ausgang wird gesperrt
1: Ausgang wird leitend

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xxxGetDrvErrMess* ermittelt werden.

_me8xDOSetWord
me8100DOSetWord

✎ Beschreibung

Funktion gilt für die Modelle: ME-81, ME-8100A/B
Schreibt ein Wort an einen 16-Bit-Ausgangsport.

☞ Wichtiger Hinweis:

Zur Aktivierung der Ausgänge muß nach Treiberstart die Funktion *_me8xDIOSetTristateOFF* vorher einmalig aufgerufen werden.

● Definitionen

C: int _me*8xxx*DOSetWord (int iBoardNumber, [int iRegisterSet,] int iPortNo, int iValue);

Delphi: Function _me*8xxx*DOSetWord (iBoardNumber: integer; [iRegisterSet: integer;] [iPortNo: integer;] iValue: integer): integer;

Basic: Declare Function me*8xxx*DOSetWord Lib "me*8xxx*_32" Alias "_VBme*8xxx*DOSetWord@12" (ByVal iBoardNumber As Long, [ByVal iRegisterSet As Long,] [ByVal iPortNo As Long,] ByVal iValue As Long) As Long

→ Parameter

`<BoardNumber>` Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)

`[RegisterSet]` (Parameter nicht in *_me8xDOSetWord*)
Auswahl des Registersatzes (für ME-8100A muß immer REGISTER_SET_A übergeben werden):

<A/B-Auswahl>	*Beschreibung*
REGISTER_SET_A (00Hex)	Registersatz für Teil A
REGISTER_SET_B (01Hex)	Registersatz für Teil B

[`PortNo`] (Parameter nicht in *me8100DOSetWord*)
hier muß PORTA (00Hex) für den 16 Bit Ausgangsport der ME-81 übergeben werden

<`Value`> Ausgabewert; mögliche Werte sind: 0000Hex…FFFFHex (dezimal: 0…65535)

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xxxGetDrvErrMess* ermittelt werden.

5.3.3 Zählerfunktionen

me8100CntRead

✎ Beschreibung

Funktion gilt für die Modelle: ME-8100A/B.

Puffert bei Funktionsaufruf den aktuellen Zählerstand des gewählten Zählers und liest den Wert ein.

● Definitionen

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

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

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

➔ Parameter

`<BoardNumber>` Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)

`<Counter>` Zähler, dessen Zählerstand eingelesen werden soll, mögliche Werte sind:

0, 1, 2: Zähler 0…2 der ME-8100A/B

`<Value>` 16-Bit Wert vom spezifizierten Zähler; es sind nur die niederwertigen 16 Bits signifikant.

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xxxGetDrvErrMess* ermittelt werden.

me8100CntWrite

✎ Beschreibung

Funktion gilt für die Modelle: ME-8100A/B.

Konfiguriert den gewählten Zähler in der gewünschten Betriebsart und lädt einen 16 Bit Startwert. Der Zähler startet mit Aufruf dieser Funktion (Vorraussetzung: Gate-Eingang mit 0 V beschaltet).

● Definitionen

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

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

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

➔ Parameter

`<BoardNumber>` Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)

<Counter>	Zähler, der konfiguriert und beschrieben werden soll, mögliche Werte sind:
	0, 1, 2: Zähler 0…2 der ME-8100A/B
<Mode>	Betriebsart des Zählers, mögliche Werte sind:

<Mode>	Zähler-Betriebsart
00Hex	Modus 0, "Zustandsänderung bei Nulldurchgang"
01Hex	Modus 1, "Retriggerbarer One-Shot"
02Hex	Modus 2, "Asymmetrischer Teiler"
03Hex	Modus 3, "Symmetrischer Teiler"
04Hex	Modus 4, "Zählerstart durch Softwaretrigger"
05Hex	Modus 5, "Zählerstart durch Hardwaretrigger"

<Value>	16-Bit Ladewert für spezifizierten Zähler; mögliche Werte sind: 0…65535 (0000Hex…FFFFHex)

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xxxGetDrvErrMess* ermittelt werden.

5.3.4 Interrupt-Handling

_me8xDisableInt
me8100DisableInt

✎ Beschreibung

Funktion gilt für die Modelle: ME-81, ME-8100.

Deaktivierung der Interruptsteuerung. Beendet eine mit *_me8xxxEnableInt* gestartete Interruptsteuerung.

☞ Wichtiger Hinweis!

Verwendung dieser Funktion in Agilent VEE nur in Verbindung mit *me8100GetIrqCnt* möglich!

● Definitionen

C: int _me*8xxx*DisableInt (int iBoardNumber, [int iRegisterSet]);

Delphi: Function _me*8xxx*DisableInt (iBoardNumber: integer; [iRegisterSet: integer;]):integer;

Basic: nicht realisiert

➔ Parameter

`<BoardNumber>` Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)

`[RegisterSet]` (Parameter nicht in *_me8xDisableInt*)

Auswahl des Registersatzes (für ME-8100A muß immer REGISTER_SET_A übergeben werden):

<A/B-Auswahl>	*Beschreibung*
REGISTER_SET_A (00Hex)	Registersatz für Teil A
REGISTER_SET_B (01Hex)	Registersatz für Teil B

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xxxGetDrvErrMess* ermittelt werden.

_me8xEnableInt
me8100EnableInt

✎ Beschreibung

Funktion gilt für die Modelle: ME-81, ME-8100A/B.

Die Interruptsteuerung der Karte wird aktiviert. Bei auftretendem Interrupt wird eine vom Anwender definierte Callback-Routine ausgeführt. Wird diese Funktion in Verbindung mit me8100GetIrqCnt benutzt, so wird anstatt der Interruptfunktion ein Null-Pointer übergeben.

☞ Wichtiger Hinweis!

Verwendung dieser Funktion in Agilent VEE nur in Verbindung mit *me8100GetIrqCnt* möglich!

● Definitionen

C: int _me*8xxx*EnableInt (int iBoardNumber, [int iRegisterSet,] [ME8100_]PSERVICE_PROC IrqFunc[, int iContext]);

Delphi: Function _me*8xxx*EnableInt (iBoardNumber: integer; [iRegisterSet: integer;] IrqFunc: Pointer[; iContext: integer]): integer;

Basic: nicht realisiert

→ Parameter

`<BoardNumber>` Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)

`[RegisterSet]` (Parameter nicht in *_me8xEnableInt*)

Auswahl des Registersatzes (für ME-8100A muß immer REGISTER_SET_A übergeben werden):

<A/B-Auswahl>	*Beschreibung*
REGISTER_SET_A (00Hex)	Registersatz für Teil A
REGISTER_SET_B (01Hex)	Registersatz für Teil B

<IrqFunc> Adresse einer anwenderdefinierten Callback-Routine, die bei einem Interrupt ausgeführt wird. Für die ME-81 ist diese vom Typ:
(void PSERVICE_PROC (void))
und für die ME-8100 vom Typ:
(void ME8100_PSERVICE_PROC (void))
bzw. vom Typ Pointer für Delphi.

`[Context]` (Parameter nicht in *_me8xEnableInt*)

Bei der ME-8100 haben Sie die Möglichkeit über diesen Parameter einen Integerwert zur Identifikation der Interruptquelle zu übergeben. Z. B. für die zweite installierte ME-8100 mit der <`Board-Number`> = „1“ und Registersatz B den Wert „4“.

← Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xxxGetDrvErrMess* ermittelt werden.

.

me8100GetIrqCnt

✎ Beschreibung

Funktion gilt für die Modelle: ME-8100A/B.

Diese Funktion ermittelt die Anzahl aller Interrupts seit dem Starten des Gerätes. Zweck dieser Funktion ist es, auch unter grafischen Programmieroberflächen wie AgilentVEE oder LabView™ Interruptfunktionen zur Verfügung stellen zu können. Wie üblich, muß auch hier die Interruptfunktionalität mit den Funktionen *…EnableInt* und *…DisableInt* aktiviert bzw. deaktiviert werden. Durch Abfrage des Zahlenwertes im Parameter `piCnt` ist es möglich, relativ zu einer vorherigen Abfrage festzustellen, ob ein Interrupt eingetroffen ist oder nicht.

● Definitionen

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

Delphi: Function me8100GetIrqCnt (iBoardNumber: integer; iRegisterSet: integer; Var iIrqCnt: integer): integer;

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

➔ Parameter

`<BoardNumber>` Nummer der anzusprechenden Karte vom Typ ME-81 (0…3) bzw. ME-8100 (0…31)

`<RegisterSet>` Auswahl des Registersatzes (für ME-8100A muß immer REGISTER_SET_A übergeben werden):

<A/B-Auswahl>	*Beschreibung*
REGISTER_SET_A (00Hex)	Registersatz für Teil A
REGISTER_SET_B (01Hex)	Registersatz für Teil B

`<IrqCnt>` Anzahl der seit dem Gerätestart eingetroffenen Interrupts.

☞ Beispiel

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

Sleep(1000);           //auf Interrupts warten

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

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xxxGetDrvErrMess* ermittelt werden

5.3.5 Fehler-Behandlung

_me8xGetDrvErrMess
me8100GetDrvErrMess

✎ Beschreibung

Funktion gilt für die Modelle: ME-81, ME-8100A/B

Falls bei der unmittelbar vorher aufgerufenen API-Funktion des Treibers ein Fehler aufgetreten ist, liefert diese Funktion den entsprechenden Fehlercode mit Fehlertext zurück.

☞ Wichtiger Hinweis!

Diese Funktion darf nur aufgerufen werden, wenn die unmittelbar vorher aufgerufene API-Funktion der `ME8x_32.DLL` bzw. `ME8100.DLL` fehlerhaft (d. h. Funktionswert 0) ausgeführt wurde!

● Definitionen

C:	int _me*8xxx*GetDrvErrMess (char *pcErrortext[, int iBufferSize]);
Delphi:	Function _me*8xxx*GetDrvErrMess (Var errortext: errorstring[; iBufferSize: integer]): integer;
Basic:	Declare Function me*8xxx*GetDrvErrMess Lib "me*8xxx*_32" Alias "_VBme*8xxx*GetDrvErrMess@4" (ByVal errortext As String[, ByVal iBufferSize As Long]) As Long

→ Parameter

<Errortext>	Zeiger auf Fehlertext; der Fehlercode wird als Funktionswert zurückgegeben.
[BufferSize]	(Parameter nicht in *_me8xGetDrvErrMess*) Puffergröße in Anzahl der Zeichen für Fehlertext wird reserviert (empfohlen: max. 128 Zeichen).

‹ Rückgabewert

0, falls kein Fehler aufgetreten ist oder Fehlercode entsprechend aufgetretenem Fehler

Anhang

A Spezifikationen

PC-Interface

Bus-System (je nach Modell)	ISA-Bus (8 Bit); Standard PCI (32 Bit, 33 MHz); CompactPCI (32 Bit, 33 MHz)
Basisadresse	**ISA** 000Hex…3F0Hex in Schritten von 10Hex einstellbar (Jumper) **PCI, cPCI:** Automatische Zuweisung der Basisadresse unter Windows (32 Bit)
Interrupts	**ISA:** IRQ2, 3, 5, 7, 10, 11, 12, 15 (Jumper) **PCI, cPCI:** Automatische Zuweisung des Interrupt-Kanals unter Windows (32 Bit)

Digitale Eingänge

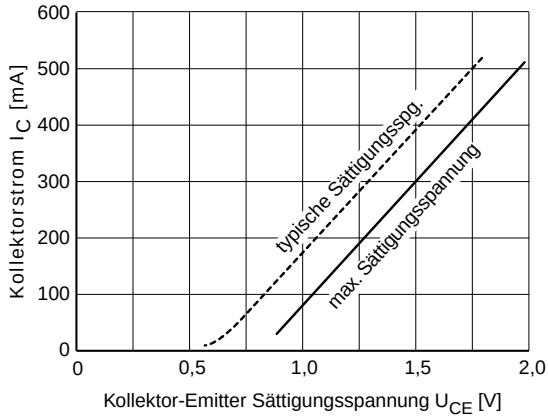
Anzahl	ME-81, ME-8100A: 16, optoentkoppelt ME-8100B: 32, optoentkoppelt
Schaltfrequenz	max. 1 kHz (abhängig von Betriebssystem und Anwendersoftware)
Eingangs-Pegel	typ. 24 V ±2 V
Eingangsstrom	10 mA pro Kanal
Betriebsarten	Einfaches Einlesen; Bitmuster-Vergleich; Bit-Änderung
Interrupt-Ereignis	Bitmustergleichheit oder bei Bit-Änderung eines ausmaskierten Eingangs-Bits

Digitale Ausgänge

Anzahl	ME-81, ME-8100A: 16, optoentkoppelt ME-8100B: 32, optoentkoppelt
Schaltfrequenz	max. 1 kHz (abhängig von Betriebssystem und Anwendersoftware)
Ausgangs-Pegel	typ. 24 V (von ext. Versorgung abhängig)

Ausgangstreiber	ME-81: Sink-Treiber (ULN2803) ME-8100A/B: portweise per Software Sink-Treiber (ULN2803) oder Source-Treiber (UDN2982) aktivierbar
Ausgangsstrom	Der maximale Strom pro Ausgang (I_C) hängt von der Sättingungsspannung U_{CE} ab und wird von der Verlustleistung der Summe der Kanäle auf P_{tot} = 1 W pro Baustein beschränkt: $P_{tot} = P_0 + \ldots + P_7 \leq 1W$ (bei 70°C)

ULN2803:



UDN2982:

Den maximalen Strom pro Ausgang (I_{Source}) entnehmen sie bitte der folgenden Tabelle. Die Verlustleistung der Summe der Kanäle darf P_{tot} = 0,7 W pro Baustein nicht übersteigen:
$P_{tot} = P_0 + \ldots + P_7 \leq 0,7$ W (bei 70°C)
wobei $P_0 = I_{C0} \cdot U_{CE0}$ mit U_{CE} = typ. 1,8 V

Anzahl der benutzten Kanäle								
	1	2	3	4	5	6	7	8
I_{Source} [mA]	350	175	115	85	70	55	50	40

Zähler/Zeitgeber

Anzahl	ME-8100A/B: 3 voneinander unabhängig
Typ	82C54
Auflösung	16 Bit
Takt-Signal (Clk)	optoisoliert, Eingangsspg. typ. 24 V

Gate-Signal (Gate)	optoisoliert, low-aktiv, Eingangsspg. typ. 24 V
Zähler-Ausgang (Out)	optoisoliert, Ausgangsspg. typ. 24 V (von ext. Versorgung abhängig)
Zählertakt	extern bis max. 1 MHz

Allgemeine Daten

Stromverbrauch bei +5 V	ME-81: typ. 0,8 A (ohne ext. Last) ME-8100A/B: typ. 1,3 A (ohne ext. Last)
Kartenabmessungen (ohne Slotblech und Stecker)	ME-81 ISA: 100 mm x 180 mm ME-8100 PCI: 174 mm x 98 mm ME-8100 cPCI: CompactPCI-Karte mit 3 HE
Anschlüsse	ME-81 ISA: 37polige Sub-D Buchse ME-8100 PCI und cPCI: 78polige Sub-D Buchse
Betriebstemperatur	0…70°C
Lagertemperatur	0…50 °C
Luftfeuchtigkeit	20…55% (nicht kondensierend)

CE-Zertifizierung

EG-Richtlinie	89/336/EMC
Emission	EN 55022
Störfestigkeit	EN 50082-2

B Anschlußbelegungen

B1 ME-8100A/B PCI und cPCI

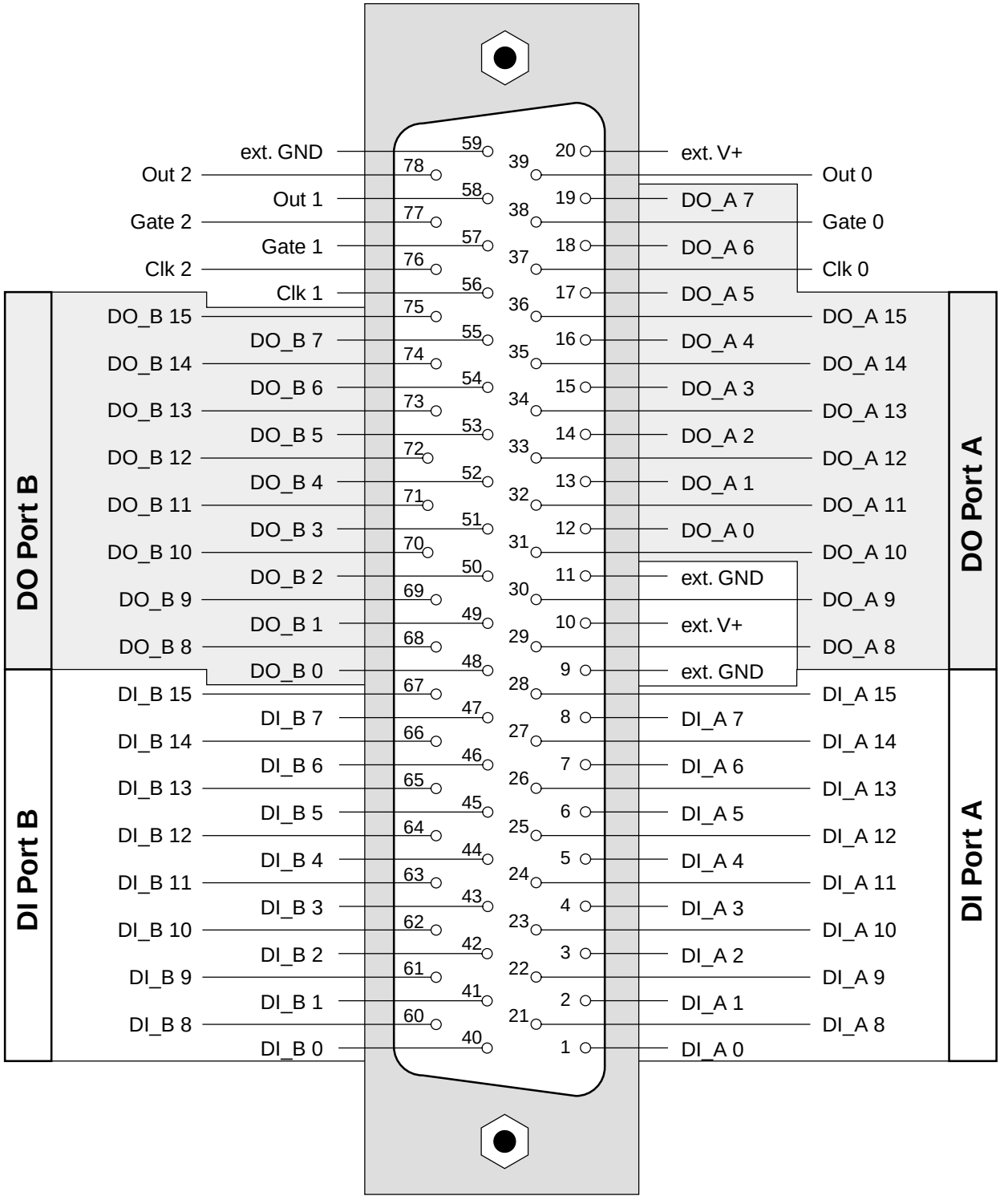


Abb. 15: Belegung der 78poligen Sub-D-Buchse ME-8100A/B

B2 ME-81 ISA

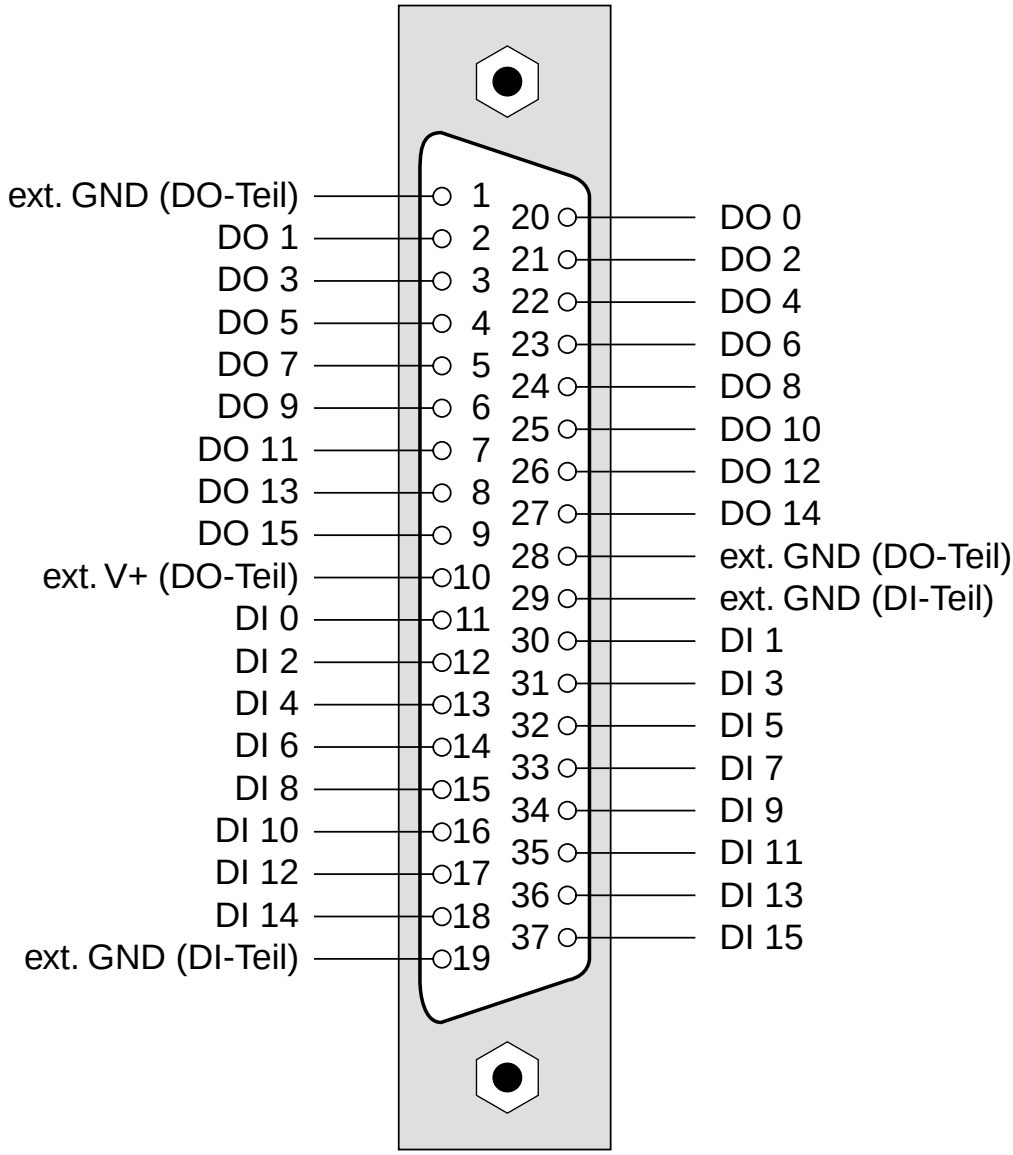


Abb. 16: Belegung der 37poligen Sub-D-Buchse ME-81 ISA

C Zubehör

Als Optionen sind folgende Produkte erhältlich (weitere Informationen über Zusatzprodukte entnehmen Sie bitte dem Meilhaus Electronic Gesamtkatalog)

ME-AB-D37M

37poliger Sub-D Anschluß-Block (Stecker) für ME-81 ISA

ME-AB-D78M

78poliger Sub-D Anschluß-Block (Stecker) für ME-8100A, ME-8100B PCI und cPCI

ME-AK-D37

37poliges Sub-D Anschluß-Kabel (Stecker-Buchse), 2 m, für ME-81 ISA

ME-AK-D78

78poliges Sub-D Anschluß-Kabel (Stecker-Buchse), 2 m, für ME-8100A, ME-8100B PCI und cPCI

D Technische Fragen

D1 Fax-Hotline

Sollten Sie technische Fragen oder Probleme haben, die auf die Karte zurückzuführen sind, dann schicken Sie bitte eine ausführliche Problembeschreibung an unsere Hotline:

Fax-Hotline: (++49) (0)89 - 89 01 66-28
eMail: support@meilhaus.de

D2 Serviceadresse

Wir hoffen, daß Sie diesen Teil des Handbuches nie benötigen werden. Sollte bei Ihrer Karte jedoch ein technischer Defekt auftreten, wenden Sie sich bitte an:

Meilhaus Electronic GmbH

Abteilung Reparaturen
Fischerstraße 2
D-82178 Puchheim

Falls Sie Ihre Karte zur Reparatur an uns zurücksenden wollen, legen Sie bitte unbedingt eine ausführliche Fehlerbeschreibung bei, inkl. Angaben zu Ihrem Rechner/System und verwendeter Software!

D3 Treiber-Update

Unter www.meilhaus.de stehen Ihnen stets die aktuellen Treiber für Meilhaus-Karten sowie unsere Handbücher im PDF-Format zur Verfügung.

E Index

W

Z