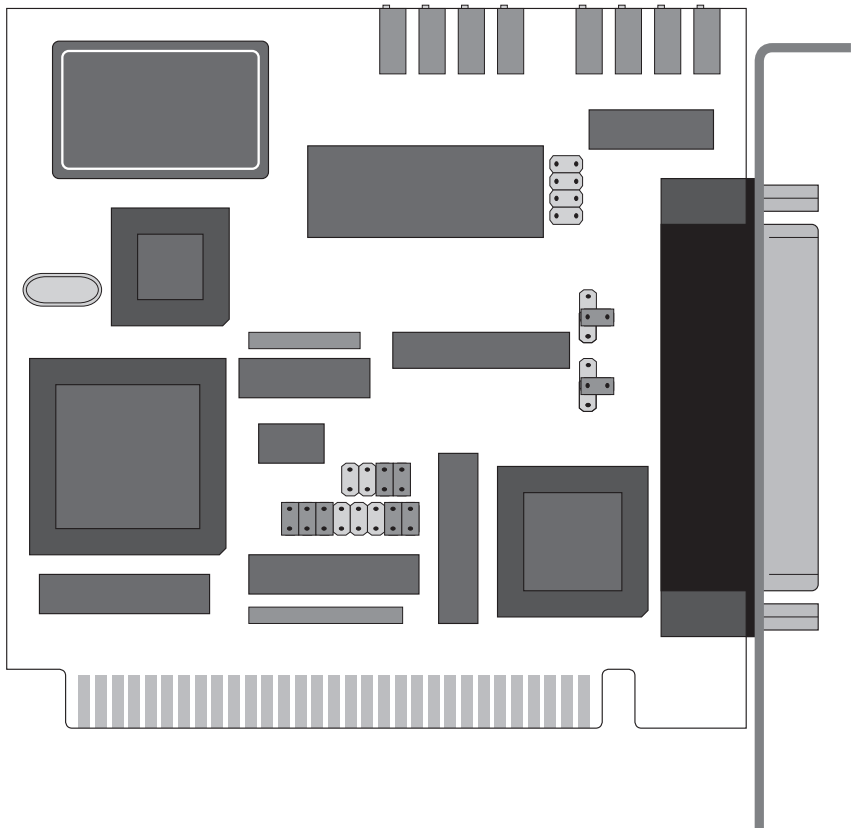
Meilhaus Electronic Handbuch

ME-80 2.0D



Digital-I/O-Karte
für ISA-Bus

Impressum

Handbuch ME-80

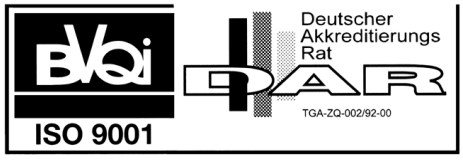
Revision 2.0D
Ausgabedatum: 6.12.99

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.



Inhalt

1 Einführung

Sehr geehrte Kundin, sehr geehrter Kunde,

Mit Ihrem Kauf 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 die Kapitel zur Installation durch. Beachten Sie 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:

- ME-80
- Dieses Handbuch
- ME-8x Treibersystem für Windows95/98/NT: 1 Diskette (3,5“ HD) „ME-8x Driver System“
- 37poliger Sub-D Gegenstecker

1.2 Leistungsmerkmale

Die ME-80 ist eine schnelle Digital I/O-Karte für den ISA-Bus. Die 32 Digital-I/Os können entweder als zwei 16 Bit breite Ports oder als ein 32 Bit breiter Port konfiguriert werden, die jeweils als Ein- oder Ausgangs-Port programmierbar sind. Ein als Eingang konfi-

gurierter Port kann die Daten entweder auf ein internes oder externes Triggersignal hin latchen.

Als Besonderheit bietet die ME-80 eine simultane 32 Bit breite Ein-/Ausgabe und einen programmierbaren, 32 Bit breiten Bitmustervergleicher, der bei Bitmustergleichheit einen Interrupt auslöst.

Als Interruptkanal sind die IRQs 2, 3, 5, 7, 10, 11, 12, und 15 über Jumper wählbar. Ein Interrupt kann auch durch einen externen Triggerimpuls ausgelöst werden.

Die mitgelieferte Software ermöglicht das rasche Einbinden der Karten in eigene Applikationen der Meß- und Steuerungstechnik unter Windows95/98/NT. Es sind Treiber für HP VEE (Hewlett-Packard) und LabVIEW™ (National Instruments), beide unter Windows95/98/NT, verfügbar. Auf Anfrage sind Treiber für DOS und Windows 3.1 erhältlich.

1.3 Systemanforderungen

Die ME-80 kann in jedem IBM-AT/386/486 bzw. Pentium und Kompatiblen eingesetzt werden, der über einen freien ISA-16 Bit Steckplatz verfügt.

1.4 Wichtige Hinweise

Falls Sie einen PC mit PCI-Bus und einem BIOS mit Plug&Play-Funktionalität benutzen, müssen Sie für alle 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 PC‘s). In diesem Fall können wir jedoch eine einwandfreie Funktion der Karte nicht gewährleisten.

Aus Gründen der Störsicherheit ist der Steckplatz mit größtmöglichem Abstand zur Videokarte vorzuziehen.

1.5 Verfügbare Software

Windows-Unterstützung	ME-80 Systemtreiber für Windows95/98/NT
Windows 3.x	auf Wunsch
MS-DOS	auf Wunsch

Hochsprachen-Unterstützung 32 Bit (im Lieferumfang)

Visual C++ ab Version 4.0
Delphi ab Version 2.0
Visual Basic ab Version 4.0

Graphische Programmierumgebungen (optional)

HP VEE Treibersystem für HP VEE ab Version 3.2
ME-80 Treibersystem für LabVIEW™ ab Version 4.0 (optional)

Test- und Demosoftware

Den aktuellen Stand des Software-Lieferumfangs ersehen Sie jeweils aus der README-Datei auf der (den) Diskette(n).

2 Installation

2.1 Hardware-Installation

2.1.1 Positionen der Jumper

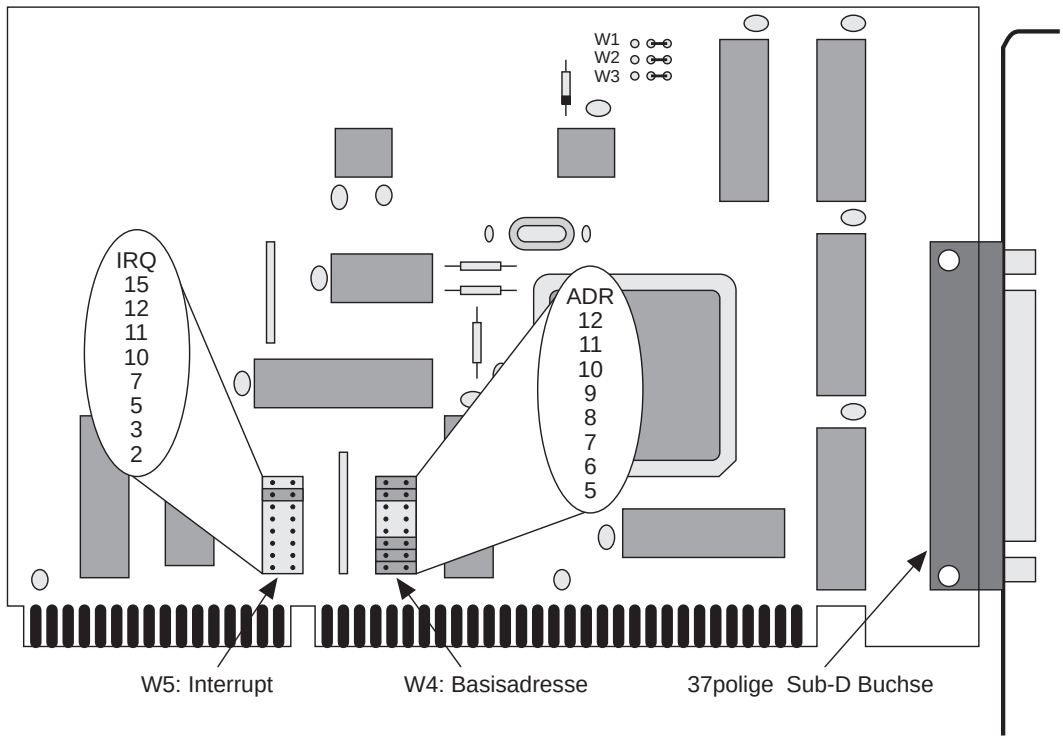


Abb. 1: Schematische Darstellung der Karte

Auf der ME-80 befinden sich Jumper für Basisadresse (W4) und Interruptkanal (W5). Die Positionen dieser Elemente sind in obiger Abbildung gekennzeichnet (vereinfachte Darstellung der ME-80). Erläuterungen zu den Funktionen und Einstellungen der Jumper finden Sie in den nachfolgenden Kapiteln.

2.1.2 Einstellungen der Jumper

Die Jumper W1, W2 und W3 sind reserviert. Pin 2 ist mit Pin 3 werkseitig gebrückt.

2.1.2.1 Einstellung der Basisadresse

Über Jumper W4 läßt sich die Basisadresse (BA) der Karte in Schritten von 32 Byte einstellen. Durch Abziehen der Jumper wird die Adresse im Binärcode eingestellt. Mit der Basisadresse beginnend, belegt die ME-80 16 Byte des I/O-Adreßraumes. Vermeiden Sie bei der Einstellung Adreßkonflikte mit anderen Karten!

Ein gesetzter Jumper entspricht dem logischen Zustand '0', ein abgezogener Jumper dem Zustand '1' und wählt damit die entsprechende Adreßleitung aus. Die Basisadresse errechnet sich durch Aufsummierung der Wertigkeit der abgezogenen Jumper. Das folgende Beispiel erläutert die werksseitige Grundeinstellung der Karte (700Hex).

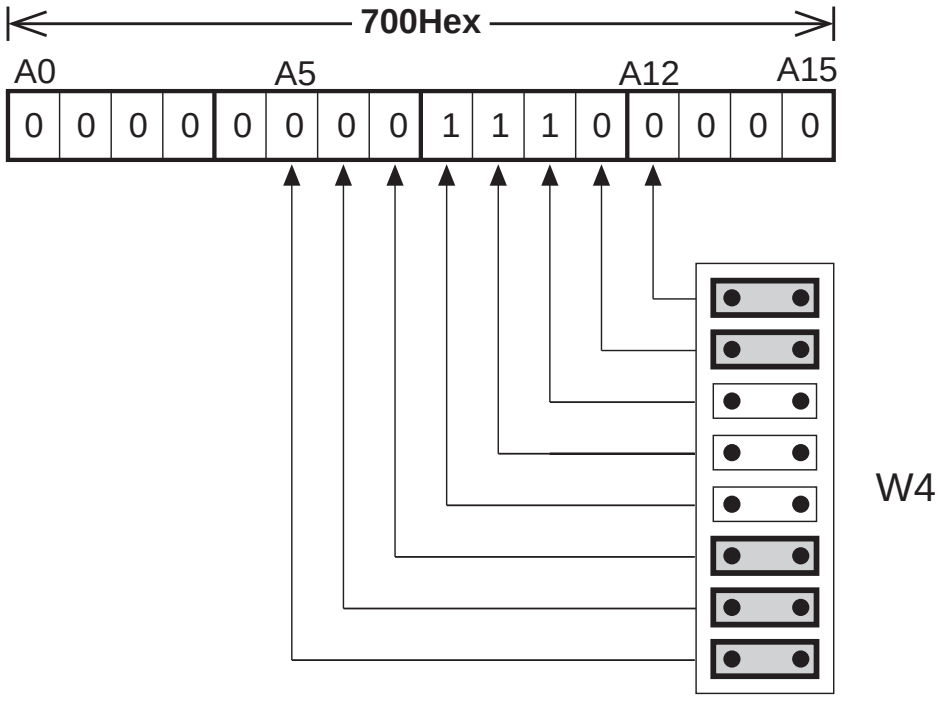


Abb. 2: Zuordnung Adressen ↔ Jumper, hier: Grundeinstellung 700Hex

2.1.2.2 Interrupts

Über den Jumper W5 stellen Sie die gewünschte Interruptleitung ein. Sie können die IRQ-Leitungen 2, 3, 5, 7, 10, 11, 12 oder 15 verwenden. Es ist jedoch zu beachten, daß der gewählte Interrupt nicht bereits von einer anderen Interruptquelle belegt ist.

Das folgende Bild zeigt, wie die Jumper gesteckt werden müssen, wenn beispielsweise die IRQ-Leitung 12 benutzt werden soll:

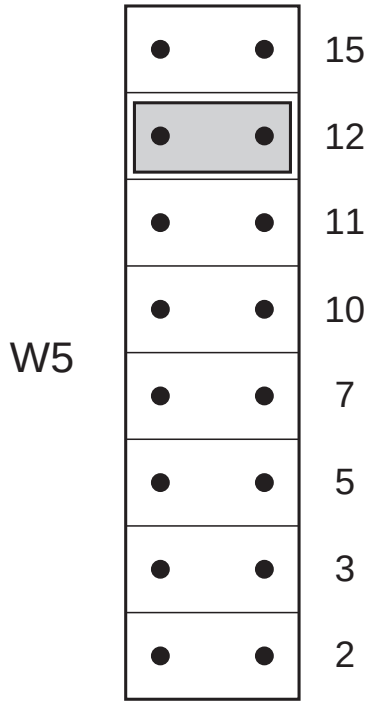


Abb. 3: Anordnung der Interrupt-Jumper
hier: Grundeinstellung IRQ 12

2.1.2.3 Standardeinstellungen

Funktion	Jumper/Schalter	Einstellung
Basisadresse	W4	700Hex
Interrupt	W5	IRQ 12

Tabelle 1: Werksseitige Standardeinstellung der ME-80

2.2 Treiberinstallation unter Windows 95/98/NT

2.2.1 Neuinstallation unter Windows95/98/NT

Für die Neuinstallation gilt unter Windows95/98 und WindowsNT 4.0 die gleiche Vorgehensweise:

- ☛ Legen Sie die Diskette „ME-8x Driver System“ ein.
- ☛ Wählen Sie im Start-Menü von Windows95/98/NT den Menüpunkt **Ausführen...** und geben Sie unter **Öffnen:** den Pfad und Dateinamen für die Datei `SETUP.EXE` auf der Installationsdiskette ein, oder klicken Sie auf **Durchsuchen...** Bestätigen Sie die Auswahl und achten Sie auf die Dialoge des Installationsprogrammes:

  ⇒ Das Installationsprogramm wird gestartet.
- ☛ Im Fenster „Install Options“ wählen Sie „Install a new board“ und klicken auf OK
- ☛ Beachten Sie die Dialoge und halten Sie die Werte für Basisadresse und Interruptkanal zur Eingabe bereit. **Achten Sie auf Übereinstimmung der Werte mit den Jumper-Stellungen** auf Ihrer Karte!

  ⇒ Die folgenden Dateien werden installiert:
  - ❑ Nur Windows 95/98: Kernel-Treiber `ME8x_32.VXD` im Pfad <*Windows-Verzeichnis*>`\SYSTEM`
  - ❑ Nur Windows NT: Kernel-Treiber `ME8x_32.SYS` im Pfad <*Windows-Verzeichnis*>`\SYSTEM32\DRIVERS`
  - ❑ API-DLL `ME8x_32.DLL` unter Windows95/98 im Pfad <*Windows-Verzeichnis*>`\SYSTEM`; unter Windows NT im Pfad <*Windows-Verzeichnis*>`\SYSTEM32`
  - ❑ Dialog-DLL `MEDLG32.DLL` unter Windows95/98 im Pfad <*Windows-Verzeichnis*>`\SYSTEM`; unter Windows NT im Pfad <*Windows-Verzeichnis*>`\SYSTEM32`

  ⇒ Diverse Dateien für die Programmierung unter Hochsprachen sowie Beispiel- und Testprogramme im Verzeichnis

<*Meilhaus-Arbeitsverzeichnis*>\ME-8x (siehe auch README-Datei auf Treiberdiskette)

⇒ Registry-Einträge werden vorgenommen.

☛ Führen Sie einen Neustart Ihres Rechners durch.

⇒ Der Systemtreiber wird automatisch geladen

⇒ In der Windows NT-Diagnose finden Sie alle korrekt installierten Karten unter **Ressourcen** aufgeführt. Die für Meilhaus-Karten relevanten Einträge finden Sie dort unter „IRQ“ und „I/O-Port“.

Hinweis:

Die ME-80 ist nicht Plug&Play-fähig und sind daher nicht im Geräte-Manager von Windows95/98 zu finden!

2.2.2 Aktualisierung der Systemtreiber

Zur Aktualisierung der Systemtreiber gilt unter Windows95/98 und WindowsNT 4.0 die gleiche Vorgehensweise.

☛ Legen Sie die Diskette „ME-8x Driver System“ ein.

☛ Wählen Sie im Start-Menü von Windows den Punkt **Ausführen...** Geben Sie unter **Öffnen:** Pfad und Dateiname für die Datei SETUP.EXE auf der Installationsdiskette ein, oder klikken Sie auf **Durchsuchen...** Bestätigen Sie die Auswahl und achten Sie auf die Dialoge des Installationsprogrammes:

⇒ Das Installationsprogramm wird gestartet.

☛ Im Fenster „Install Options“ wählen Sie „Update driver and language libraries“ und klicken auf OK.

⇒ Systemtreiber, API-DLL sowie Hochsprachenbibliotheken, Demo- und Testprogramme werden aktualisiert.

☛ Führen Sie einen Neustart Ihres Rechners durch.

2.2.3 Karteneinstellungen ändern

Verwenden Sie die hier beschriebene Vorgehensweise zum ändern der Einstellung für die Basisadresse in der Registry. Beachten Sie, daß diese Werte mit den Jumper-Stellungen auf Ihrer Karte übereinstimmen müssen. Es gilt unter Windows95/98 und WindowsNT 4.0 die gleiche Vorgehensweise.

- ☛ Legen Sie die Diskette „ME-8x Driver System“ ein.
- ☛ Wählen Sie im Start-Menü von Windows den Punkt **Ausführen...** Geben Sie unter **Öffnen:** Pfad und Dateiname für die Datei `SETUP.EXE` auf der Installationsdiskette ein, oder klikken Sie auf **Durchsuchen...** Bestätigen Sie die Auswahl und achten Sie auf die Dialoge des Installationsprogrammes:

  ⇒ Das Installationsprogramm wird gestartet.
- ☛ Im Fenster „Install Options“ wählen Sie „Update settings of an installed board“ und klicken auf OK.
- ☛ Beachten Sie die Dialoge und halten Sie die Werte für Basisadresse und. Interruptkanal zur Eingabe bereit. **Achten Sie auf Übereinstimmung der Werte mit den Jumper-Stellungen** auf Ihrer Karte!
- ☛ Führen Sie einen Neustart Ihres Rechners durch.

  Unter WindowsNT können Sie alternativ im Start-Menü unter: **Einstellungen ➜ Systemsteuerung ➜ Geräte** den Treiber beenden und anschließend neu starten.

2.3 Deinstallation

2.3.1 Deinstallation einer einzelnen Karte

Mit Hilfe des Installations- und Deinstallationsprogramms auf Ihrer „ME-8x Driver System“-Diskette können Sie einzelne Karten der Kartenfamilie ME-8x aus der Windows-Registry entfernen. Die einzelnen Komponenten wie Systemtreiber, API-DLL und Hochsprachenbibliotheken werden auf diesem Wege nicht deinstalliert. Es gilt unter Windows95/98 und WindowsNT 4.0 die gleiche Vorgehensweise.

☛ Legen Sie bitte die Diskette „ME-8x Driver System“ ein.

☛ Wählen Sie im Start-Menü von Windows den Punkt **Ausführen...** Geben Sie unter **Öffnen:** Pfad und Dateiname für die Datei SETUP.EXE auf der Installationsdiskette ein, oder klikken Sie auf **Durchsuchen...** Bestätigen Sie die Auswahl und achten Sie auf die Dialoge des Installationsprogrammes:

⇒ Das Installationsprogramm wird gestartet.

☛ Im Fenster „Install Options“ wählen Sie „Uninstall a single board“ und bestätigen mit OK.

☛ Wählen Sie die Karte aus, die Sie aus der Windows-Registry entfernen möchten.

☛ Führen Sie einen Neustart Ihres Rechners durch.

⇒ Die gewählte Karte wird aus der Windows-Registry entfernt!

2.3.2 Deinstallation des Treibersystems

Beachten Sie, daß auf diesem Weg das ME-8x Treibersystem als ganzes deinstalliert werden kann. Dazu gehören neben dem Systemtreiber und der API-DLL (für **alle** installierten Karten vom Typ ME-63, ME-80 und ME-81) auch die Hochsprachenbibliotheken, Demos und Testprogramme, die sich standardmäßig im Unterverzeichnis „ME-8x“ des Verzeichnisses „C:\MEILHAUS“ befinden.

Für die Deinstallation des ME-8x Treibersystems gilt unter Windows95/98 und WindowsNT 4.0 die gleiche Vorgehensweise.

☛ Wählen Sie dazu im START-Menü von Windows unter **Einstellungen ➜ Systemsteuerung ➜ Software** im Register **Installieren/Deinstallieren** den Eintrag: „ME-8x Driver Uninstall“ und bestätigen Sie mit „OK“.

⇒ Das gesamte ME-8x Treibersystem wird deinstalliert!

3 Hardware

3.1 Blockschaltbild

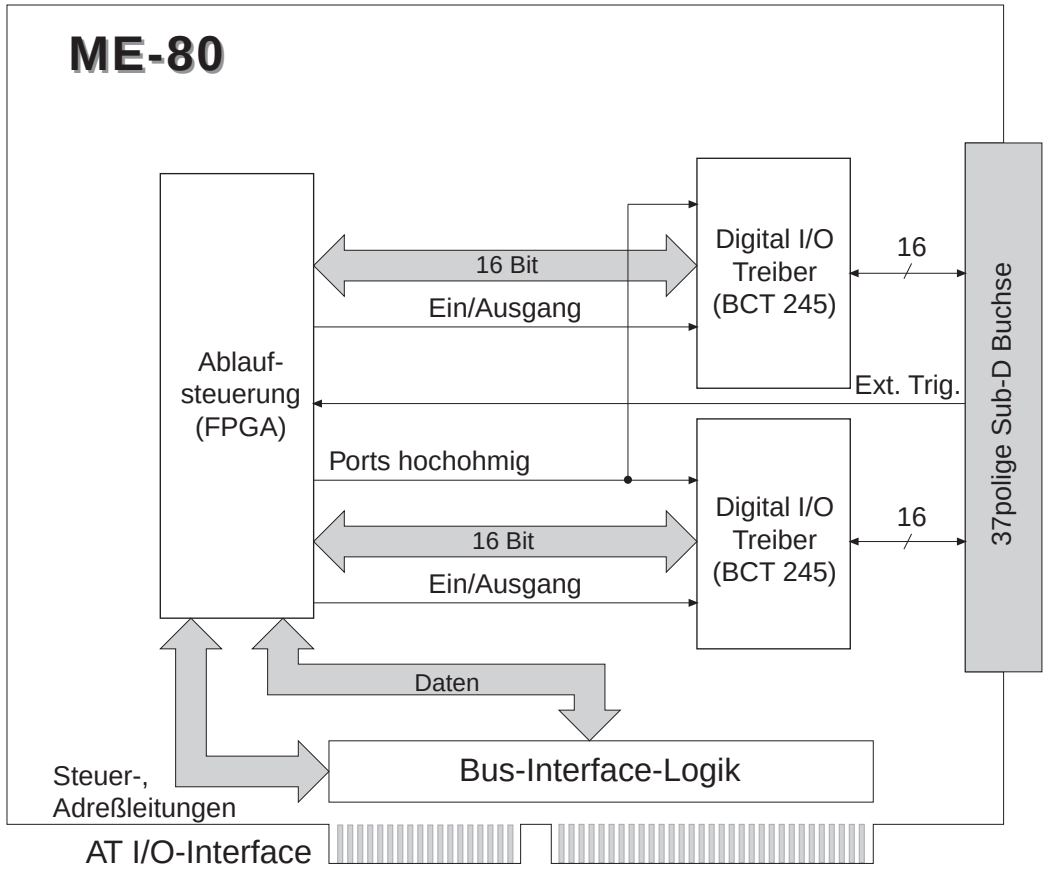


Abb. 4: Blockschaltbild der ME-80

3.2 Generelle Hinweise

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

3.3 Betriebsarten

Als besonderes Leistungsmerkmal bietet die ME-80 als 16 Bit ISA-Karte die Möglichkeit ein 32 Bit Wort simultan zu latchen. Das Einlesen bzw. Ausgeben muß jedoch systembedingt (16 Bit ISA-Bus) 16 Bit breit erfolgen. Dieser Sachverhalt erfordert eine geschickte Kombination der Betriebs- und Triggerarten.

Die Ausgänge aller Modelle können per Software portweise hochohmig geschaltet werden. Nach dem Einschalten der Versorgungsspannung sind die Ports als 32 Bit Eingangsport konfiguriert.

Die Konfiguration der Karte erfolgt durch entsprechende Programmierung durch den Anwender. Die Verwendung der mitgelieferten Treibersoftware unter Windows 95/98/NT wird empfohlen, siehe Kap. 5.3 „Beschreibung der API-Funktionen“ auf Seite 37.

3.3.1 Digitale Ein-/Ausgabe

Folgende Konfigurationmöglichkeiten stehen zur Verfügung (Port A = DIO15…DIO0, Port B = DIO31…DIO16):

- **32 Bit Eingangsport**

  Das anliegende 32 Bit Wort wird in Abhängigkeit von der verwendeten Triggerart (siehe Kap. 3.3.3 auf Seite 20) gelacht. Zum Einlesen verwenden Sie bitte die Funktion *_me80DIGetLong*.

- **32 Bit Ausgangsport**

  Das auszugebende 32 Bit-Wort wird simultan ausgegeben. Verwenden sie hierzu die Funktion *_me80DOSetLong*.

- **2 x 16 Bit Eingangsports**

  Die beiden Ports können unabhängig voneinander mit der Funktion *_me8xDIGetWord*.eingelesen werden. Zur Triggerart beachten Sie bitte Kap. 3.3.3 auf Seite 20.

- **2 x 16 Bit Ausgangsports**

  Die beiden Ports können unabhängig voneinander mit der Funktion *_me8xDOSetWord* ausgegeben werden.

- **16 Bit Eingangsport und 16 Bit Ausgangsport**

  In dieser Konfiguration ist immer Port A der Eingang und Port B der Ausgang. Verwenden Sie die Funktion *_me8xDIGetWord* um ein 16 Bit-Wort gemäß verwendeter Triggerart einzulesen bzw. Funktion *_me8xDOSetWord* um ein 16 Bit-Wort auszugeben.

3.3.2 Bitmuster-Vergleich

In der Betriebsart „Bitmuster-Vergleich“ wird ein ins Vergleichsregister geschriebenes Bitmuster mit dem am korrespondierenden Eingangsport anliegenden Bitmuster verglichen. Zur Triggerung können Sie ein internes oder externes Triggersignal verwenden (siehe Kap. 3.3.3 auf Seite 20). Bei Bitmuster-Gleichheit kann (falls freigegeben) ein Interrupt ausgelöst werden. Der Vergleich kann 16 oder 32 Bit breit erfolgen. Dazu stehen zwei 16 Bit-Register zur Aufnahme eines Vergleichsbitmusters zur Verfügung. Vorraussetzung ist, daß mindestens ein Port als Eingang konfiguriert ist. Beim Schreiben der Vergleichs-Register sind folgende 3 Fälle zu unterscheiden:

- **32 Bit Eingangsport**

  Verwenden Sie die Funktion *_me8xDIOSetPattern* um das 32 Bit Vergleichsmuster zu schreiben.

- **2 x 16 Bit Eingangsports**

  In diesem Fall kann nur das Eingangswort an Port A mit dem korrespondierenden Vergleichsregister verglichen werden. Verwenden Sie die Funktion *_me8xDIOSetPattern* um das 16 Bit Vergleichsmuster für Port A zu schreiben.

- **16 Bit Eingangsport und 16 Bit Ausgangsport**

  In diesem Fall kann das Eingangswort an Port A mit dem korrespondierenden Vergleichsregister verglichen werden und Port B als 16 Bit Ausgangsport verwendet werden. Verwen-

den Sie die Funktion *_me8xDIOSetPattern* um das 16 Bit Vergleichsmuster für Port A zu schreiben und die Funktion *_me8xDOSetWord* um auf Port B auszugeben.

3.3.3 Triggerarten

Die ME-80 bietet folgende Trigger-Möglichkeiten:

- **Ohne Triggersignal**

  In dieser Triggerart werden bei 32 Bit breitem Einlesen die Eingänge nach dem **ersten** Lesezyklus gelatcht. Analog dazu wird bei 32 Bit breiter Ausgabe der Wert nach dem **zweiten** Schreibzyklus gelatcht. Bei Verwendung der Funktionen *_me80DIGetLong* bzw. *_me80DOSetLong* erledigt dies der Treiber für Sie und gewährleistet eine simultane 32 Bit breite Ein-/Ausgabe. Sobald ein 16 Bit breiter Port angesprochen wird, wird nach jedem Portzugriff gelatcht.

  Verwenden Sie die Konstante REGMODE_DIRECT in der Funktion *_me80DIOSetMode*.

- **Internes Triggersignal**

  Das interne Triggersignal (2,5 MHz) dient in erster Linie als Triggersignal in der Betriebsart Bitmuster-Vergleich.

  Verwenden Sie die Konstante REGMODE_INTERNAL in der Funktion *_me80DIOSetMode*.

- **Externes Triggersignal**

  In der Betriebsart Bitmuster-Vergleich kann der Vergleich auch über die steigende Flanke eines externen Triggersignals (Pin 10) gestartet werden. Das externe Triggersignal kann aber auch zum Latchen der Eingangsports verwendet werden.

  Verwenden Sie die Konstante REGMODE_EXTERNAL in der Funktion *_me80DIOSetMode*.

  Alternativ dazu kann der externe Triggereingang aber auch als Interrupt-Eingang verwendet werden, der direkt zum PC weitergeleitet wird. In diesem Fall darf jedoch die Konstante REGMODE_EXTERNAL in der Funktion *_me80DIOSetMode* nicht verwendet werden. In der Funktion

_me8xDIOSetIntMode verwenden Sie bitte die Konstante INT_EXTERN.

3.4 Beschaltung

Achtung: Der Anschlußstecker sollte grundsätzlich nur im spannungslosen Zustand auf die Sub-D Stecker der Karte aufgesteckt bzw. von dieser abgezogen werden.

Die Ein-/Ausgabe-Ports der ME-80 sind mit bidirektionalen Bustreibern vom Typ BCT245 ausgestattet. Bei der Beschaltung der digitalen Ein-/Ausgänge und des externen Triggereingangs ist lediglich darauf zu achten, daß der TTL-Pegel eingehalten wird und eine Verbindung zur Masse (Pin 20) hergestellt wird.

Die Belegung der 37poligen Sub-D Buchse finden Sie im Anhang (siehe “Anschlußbelegungen” auf Seite 55).

3.5 Register

Die ME-80 kann, z. B. unter DOS, auch auf Registerebene programmiert werden. Eine entsprechende Registerbeschreibung finden Sie in diesem Abschnitt. Wir empfehlen jedoch grundsätzlich die Verwendung der mitgelieferten Treibersoftware unter Windows 95/98 oder NT!

Die ME-80 belegt mit der über Jumper W4 eingestellten Basisadresse (BA) beginnend 16 aufeinanderfolgende Bytes im I/O-Adreßraum des PC. Die Register sind in den folgenden Tabellen beschrieben sind (R = Lesen, W = Schreiben):

Offset	16 Bit-Register	
BA+00H	**R**	**M8_FIDREG** Enthält Function-ID und INTR-Bit *b15…b9* . . 0 reserviert *b8* . . . . . *INTR_BIT* 1 wird gesetzt, bei Interrupt durch Bitmustergleichheit (EX_INT=0), oder durch steigende Flanke des ext. Triggers (EX_INT01) *b7* . . . . . *FID7* 1 Funktionsgruppe „Vergleichspattern“ *b6* . . . . . *FID6* 0 reserviert *b5* . . . . . *FID5* Funktionsgruppe „Analoge Eingabe“ (AI) *b4* . . . . . *FID4* Funktionsgruppe „Analoge Ausgabe“ (AO) *b3* . . . . . *FID3* Funktionsgruppe „Digitale Ein/Ausgabe“ (DIO) *b2…b0* . . *FID2…FID0* PROM-Version Daraus ergibt sich folgende Function-ID (z. B. für PROM-Version 1): 10001001 = 89Hex

Tabelle 2: Adreßraum ME-80

Offset		16 Bit-Register
BA+00H	**W**	**M8_CONTROL** Kontrollregister *b15…b8. .* 0 reserviert *b7 . . . . . ENIO* 1 Digitale Ports aktiv 0 Digitale Ports hochohmig *b6, b5 . . . EX_INT, EN_INT* 0 0 Interrupt deaktiviert 0 1 Interrupt bei Bitmuster-Gleichheit 1 0 Interrupt deaktiviert 1 1 Interrupt durch ext. Trigger *b4, b3 . . . INL_MOD1, INL_MOD0* 0 0 Daten werden durch Lesezyklus gelatcht 0 1 Daten werden mit der steigenden Flanke des internen Triggersignals gelatcht 1 0 Daten werden mit steigender Flanke des externen Triggers gelatcht 1 1 reserviert *b2…b0. . . CFG2…CFG0* 0 0 0 1 x 32 Bit Eingangsport 0 0 1 1 x 32 Bit Ausgangsport 0 1 0 2 x 16 Bit Eingangsports 0 1 1 2 x 16 Bit Ausgangsport 1 0 0 1 x 16 Bit Eingangsport (Port A) und 1 x 16 Bit Ausgangsport (Port B)
BA+02H	**R**	**M8_RSTINT** Interrupt-Bit INTR_BIT rücksetzen durch Lesen dieser Adresse. Solange INTR_BIT gesetzt ist, kann kein neuer Interrupt erzeugt werden.

Tabelle 2: Adreßraum ME-80

Offset	16 Bit-Register	
BA+04H	**R/W**	**M8_IOREQ0** Datenregister Port A *b15…b0 . . DIO15…DIO0* 16 Bit Datenregister Port A bzw. Low-Word für 32 Bit Port
BA+06H	**R/W**	**M8_IOREQ1** Datenregister Port B *b15…b0 . . DIO31…DIO16* 16 Bit Datenregister Port B bzw. High-Word für 32 Bit Port
BA+08H	**W**	**M8_COMP0** Vergleichsregister Port A *b15…b0 . .* Vergleichs-Bitmuster Port A bzw. Low-Word für 32 Bit Vergleichs-Bitmuster
BA+0AH	**W**	**M8_COMP1** Vergleichsregister Port B *b15…b0 . .* Vergleichs-Bitmuster Port B bzw. High-Word für 32 Bit Vergleichs-Bitmuster
BA+0CH	**–**	reserviert
BA+0EH	**–**	reserviert

Tabelle 2: Adreßraum ME-80

3.6 Testprogramm

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

4 Programmierung

4.1 Hochsprachenprogrammierung

Folgende Hochsprachen werden standardmäßig unterstützt:

- Visual C++ ab Version 4.0. Bitte beachten Sie die Hinweise in den entsprechenden README-Dateien.
- Delphi ab Version 2.0. Bitte beachten Sie die Hinweise in den entsprechenden README-Dateien.
- VisualBASIC ab Version 4.0. Bitte beachten Sie die Hinweise in den entsprechenden README-Dateien.
- für weitere Hochsprachen beachten Sie bitte die entsprechenden README-Dateien auf Ihrer Treiber-Diskette(n).

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 Beispielprogramme

Zum leichteren Verständnis der Programmierung werden einfache Beispiele und kleine Projekte im Source-Code mitgeliefert. Die Beispielprogramme werden automatisch in entsprechende Unterverzeichnisse von `C:\Meilhaus\` (Default) installiert. Bitte beachten Sie die Hinweise in den entsprechenden README-Dateien.

4.2 HP VEE-Programmierung

Die HP VEE-Komponenten für die ME-80 erhalten Sie auf separater(n) Diskette(n) im Lieferumfang von HP VEE bei Meilhaus Electronic. Die ME-80 unterstützt die HP VEE-Vollversionen 3.2 oder höher unter Windows 95/98/NT. Zur Installation der HP VEE-Komponenten und für weitere Infos, beachten Sie bitte die PDF-Datei auf den mit dem HP VEE Treibersystem gelieferten Disketten. Zu den Grundlagen der HP VEE-Programmierung benutzen Sie bitte Ihre HP VEE-Dokumentation und die HP VEE Online-Hilfe.

4.2.1 User Objects

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

Die UOs sind weitgehend selbsterklärend und basieren auf den im Kap. „5 Funktionsreferenz“ auf Seite 35 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 UO finden Sie auch unter „Description“ indem Sie den Mauszeiger über das entsprechende UO bewegen und die rechte Maustaste drücken.

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

4.2.2 HP VEE-Demoprogramme

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

Die HP VEE-Demoprogramme enthalten teilweise auch Ergänzungen der „normalen“ UOs 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 HP VEE automatisch um den Eintrag „ME Board“. Dadurch ist eine komfortable Nutzung aller unter HP VEE zur Verfügung stehenden Treiberfunktionen möglich. Über das „ME Board“-Menü können Sie nach Kartenfamilien geordnet, die entsprechenden Treiber- und Demo-User Objects aufrufen.

Hinweis:

Der Installationsumfang der User Objects (UOs) richtet sich nach der von Ihnen gewählten Kartenfamilie zu Beginn der HP 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 HP VEE Komponenten jederzeit nachinstallieren. Verwenden Sie dazu die Diskette „Meilhaus HP VEE Driver System“.

4.3 LabVIEW™-Programmierung

Die LabVIEW™-Komponenten für die ME-80 erhalten Sie optional auf separater(n) Diskette(n) bei Meilhaus Electronic. Die ME-80 unterstützt die LabVIEW™ Vollversionen 4.x oder höher unter Windows 95/98/NT. Zur Installation der LabVIEW™-Komponenten und für weitere Infos beachten Sie bitte die entsprechende Liesmich-Datei auf den mit dem LabVIEW™-Treiber gelieferten Disketten. 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, welche intern API-Funktionen aufrufen. 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 VIs sind weitgehend selbsterklärend und basieren auf den im Kap. „5 Funktionsreferenz“ auf Seite 35 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 „ME-80 Function Tree“. Dieses VI dient nur der Dokumentation und kann über das Menü „Datei - Öffnen“ aufgerufen werden. Dort finden Sie unter „Description“ 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 LabVIEW™-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

Hinweis: Die Programmierung von Computer-Hochsprachen aus (falls Sie z. B. für die Kartenserie schon Software geschrieben haben) ist durch Port-Ein-/Ausgabe-Anweisungen möglich. 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 95/98 oder NT.

4.4.1 Initialisierung

- Function ID Register (M8_FIDREG) auslesen. Dabei müssen im niederwertigen Byte die höherwertigen 5 Bits (b7…b3) stets den Wert '10001' haben, während die niederwertigen 3 Bits (b2…b0) von der jeweiligen PROM-Version abhängen. So gilt z. B. für die PROM-Versionsnummer 1 (001) folgender Eintrag im Function ID Register:

  10001001Binär = 89Hex.

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

- Alle Aktionen auf der Karte stoppen: Kontrollregister (M8_CONTROL) auf 00Hex setzen.

- Interruptbehandlung auf der Karte rücksetzen, durch „Dummylesen“ des Registers M8_RSTINT.

4.4.2 Einfaches Einlesen

- Karte initialisieren (siehe 4.4.1)

- Festlegen der Konfiguration im Register M8_CONTROL:

  - Digitale Ports leitend schalten (ENIO); b7 = 1

  - Interrupt deaktivieren (EX_INT, EN_INT); b6, b5 = 0, 0)

  - Auswahl der Triggerart (INL_MOD1, INL_MOD0) (Dateneingänge werden unmittelbar nach Lesebefehl übernommen); b4, b3 = 0, 0

- Konfiguration der Betriebsart auf „Eingänge“ (CFG2..CFG0); b2…b0 = 0, 0, 0 oder 0, 1, 0

- Einlesen der jeweiligen Register (Erstes I/O-Word und/oder zweites I/O-Word)

4.4.3 Einfaches Ausgeben

- Karte initialisieren (siehe 4.4.1)
- Festlegen der Konfiguration (siehe 4.4.2) mit dem Unterschied, daß jetzt die Betriebsart auf „Ausgänge“ gesetzt wird (CFG2..CFG0); b2..b0 = 0, 0, 1 oder 0, 1, 1
- Setzen der Ausgänge durch Schreiben in die jeweiligen Register (Erstes I/O-Word und/oder zweites I/O-Word)

Hinweis:
Ein Port, der als Ausgang programmiert ist, kann auch rückgelesen werden.

4.4.4 Interrupt-Betrieb

(wahlweise durch Bitmuster-Vergleich oder externen Trigger)

- Karte initialisieren (siehe 4.4.1)
- INTR_BIT zurücksetzen durch „Dummylesen“ des Registers M8_RSTINT.
- Falls ein Interrupt über Bitmuster-Vergleich ausgelöst werden soll, Wert in Vergleichsregister schreiben (Erstes Vergleichs-WORD, zweites Vergleichs-WORD).
- Festlegen der Konfiguration im Register M8_CONTROL:
  - Digitale Ports leitend schalten (ENIO); b7 = 1
  - Auswahl der Interruptquelle (EX_INT, EN_INT) Bitmuster-Vergleich: b6, b5 = 0, 1; Ext. Trigger: b6, b5 = 1, 1
  - Auswahl der Triggerart (INL_MOD1, INL_MOD0)

Hinweis:
Bei Interrupt-Betrieb mittels Bitmuster-Vergleich muß einer der beiden folgenden Triggerarten (INL_MOD1, INL_MOD0) verwendet werden:

Interner Trigger (2,5 MHz):	b4, b3 = 0, 1
Ext. Trigger:	b4, b3 = 1, 0

- Konfiguration der Betriebsart (CFG2..CFG0)

(siehe auch Kap. 3.5 „Register“ auf Seite 21)

Hinweis:
Nach jedem Interrupt muß das INTR_BIT durch „Dummylesen“ des Registers M8_RSTINT zurückgesetzt werden, da sonst kein Interrupt mehr ausgelöst wird.

5 Funktionsreferenz

5.1 Funktionsweise des 32 Bit-Treibers

Der 32 Bit-Treiber für die ME-80 wurde für Windows95/98 und für WindowsNT erstellt und besteht aus folgenden Teilen:

- VxD-Treiber (`ME8x_32.VXD`) für Windows95/98, wird dynamisch geladen.
- Kernel-Treiber (`ME8x_32.SYS`) für WindowsNT wird automatisch beim Systemstart geladen.
- API-DLL (`ME8x_32.DLL`) mit den Treiber-Funktionen für die ME-80.
- Dialog-DLL (`MEDLG32.DLL`) mit Dialogfunktionen.

Durch das Installationsprogramm wird jede neu installierte Karte vom Typ ME-80 beim Betriebssystem registriert. Es werden bis zu 4 Karten aus derselben Kartenfamilie (iBoardNumber 0…3) und bis zu 12 Karten insgesamt vom Treiber unterstützt.

Mit jedem Treiberstart sucht dieser die registrierten Karten, überprüft jedoch nicht deren tatsächliches Vorhandensein (die ME-80 ist nicht Plug&Play kompatibel). Nachdem der Treiber erfolgreich geladen wurde, kann die Karte über die API-Funktionen angesprochen werden.

Die API-Funktionen ermöglichen den komfortablen Zugriff auf die Hardware. Jede Funktion, die auf eine Karte vom Typ ME-80 zugreifen soll, benötigt zur Identifizierung der Karte einen Integerwert. In der nun folgenden Beschreibung der Funktionen ist dieser Parameter mit `<BoardNumber>` bezeichnet. Die API-Funktionen werden im folgenden detailliert beschrieben.

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).

_me80… Funktion gültig für die Karte ME-80

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

Für die Funktionsbeschreibung gelten folgende Vereinbarungen:

Funktionsnamen	werden im Fließtext kursiv geschrieben z. B. *_me8xDIGetBit*
<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
me80…()	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		
_me8xGetDLLVersion	DLL-Versionsnummer ermitteln	38
_me8xPROMVersion	PROM-ID der Karte ermitteln	38
Digitale Ein-/Ausgabe		
_me8xDIGetBit	Bit einlesen	39
_me8xDIGetWord	Word (16 Bit) einlesen	40
_me80DIGetLong	32 Bit Wort einlesen	41
_me8xDIOSetIntMode	Interrupt-Ereignis auswählen	42
_me80DIOSetMode	Ports konfigurieren	43
_me8xDIOSetPattern	Vergleichsbitmuster schreiben	44
_me8xDIOSetTristateOFF	Ausgangsport leitend schalten	45
_me8xDIOSetTristateON	Ausgangsport hochohmig schalten	46
_me8xDOSetBit	Bit ausgeben	46
_me8xDOSetWord	Word (16 Bit) ausgeben	48
_me80DOSetLong	32 Bit Wort ausgeben	49
Interrupt-Handling		
_me8xDisableInt	Interruptsteuerung deaktivieren	50
_me8xEnableInt	Interruptsteuerung aktivieren	50
Fehler-Behandlung		
_me8xGetDrvErrMess	Fehlerstring gemäß Fehlercode	51

Tabelle 3: Übersicht der Bibliotheksfunktionen

5.3.1 Allgemeine Funktionen

_me8xGetDLLVersion

✎ Beschreibung

Funktion gilt für den Kartentyp ME-80.

Ermittelt die Versionsnummer der Karten-DLL für die Kartenfamilie ME-8x

● Definitionen

C: int _me8xGetDLLVersion();

Delphi: Function _me8xGetDLLVersion: integer;

Basic: Declare Function me8xGetDLLVersion Lib "me8x_32" Alias "_VBme8xGetDLLVersion@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: Rückgabewert 0x00010003 ergibt die Version 1.03

_me8xPROMVersion

✎ Beschreibung

Funktion gilt für den Kartentyp ME-80.

Ermittelt die karteninterne PROM-ID.

● Definitionen

C: int _me8xPROMVersion (int iBoardNumber, int *piVersion;)

Delphi: Function _me8xPROMVersion (iBoardNumber: integer; Var iVersion: integer): integer;

Basic: Declare Function me8xPROMVersion Lib "me8x_32" Alias "_VBme8xPROMVersion@8" (ByVal iBoardNumber As Long, ByRef iVersion As Long) As Long

➔ Parameter

`<BoardNumber>` Kartennummer für 1., 2., 3. oder 4. installierte Karte des Typs ME-63, ME-80 oder ME-81; mögliche Werte sind: 0…3

`<Version>` Zeiger auf Integerwert mit der Codierung der PROM-Version. Der Wert wird hexadezimal interpretiert. Es sind nur die niederwertigen 8 Bits signifikant. Die Bedeutung der einzelnen Bits entnehmen Sie bitte Registerbeschreibung (FID-Register, BA+00H).

Im Fehlerfall, d. h. wenn die Karte nicht korrekt installiert ist, wird die Funktion zwar ausgeführt, es wird jedoch eine ungültige Versionsnummer (FFHex) zurückgegeben.

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xGetDrvErrMess* ermittelt werden.

5.3.2 Digitale Ein-/Ausgabe

_me8xDIGetBit

✎ Beschreibung

Funktion gilt für den Kartentyp ME-80.

Liefert den Zustand einer einzelnen Eingangsleitung zurück.

☞ Wichtiger Hinweis:

Zur Konfiguration der Ports muß vorher die Funktion *_me80DIOSetMode* einmalig aufgerufen werden.

● Definitionen

C: int _me8xDIGetBit (int iBoardNumber, int iBitNo, int *piBitValue);

Delphi: Function _me8xDIGetBit (iBoardNumber, iBitNo: integer; Var iBitValue: integer): integer;

Basic: Declare Function me8xDIGetBit Lib "me8x_32" Alias "_VBme8xDIGetBit@12" (ByVal iBoardNumber As Long, ByVal iBitNo As Long, iBitValue As Long) As Long

➔ Parameter

`<BoardNumber>`	Kartennummer für 1., 2., 3. oder 4. installierte Karte des Typs ME-63, ME-80 oder ME-81; mögliche Werte sind: 0…3
`<BitNo>`	Nummer der Eingangsleitung, die abgefragt werden soll; möglich sind: 0…15: 16-Bit-Eingangsport A oder niederwertige 16 Bits des 32 Bit Ports 16…31: 16-Bit-Eingangsport B oder höherwertige 16 Bits des 32 Bit Ports
`<BitValue>`	Zeiger auf einen Integerwert, der dann dem Leitungszustand entsprechend gelesen wird: 0: Leitung auf logisch '0' gesetzt 1: Leitung auf logisch '1' gesetzt

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xGetDrvErrMess* ermittelt werden.

_me8xDIGetWord

✎ Beschreibung

Funktion gilt für den Kartentyp ME-80.

Liest ein Wort von einem 16 Bit Port der Karte.

☞ Wichtiger Hinweis:

Zur Konfiguration der Ports muß vorher die Funktion *_me80DIOSetMode* einmalig aufgerufen werden.

● Definitionen

C:	int _me8xDIGetWord (int iBoardNumber, int iPortNo, int *iValue);
Delphi:	Function _meDIGetWord (iBoardNumber, iPortNo: integer; Var iValue: integer): integer;
Basic:	Declare Function me8xDIGetWord Lib "me8x_32" Alias "_VBme8xDIGetWord@12" (ByVal iBoardNumber As Long, ByVal iPortNo As Long, iValue As Long) As Long

➔ Parameter

`<BoardNumber>`	Kartennummer für 1., 2., 3. oder 4. installierte Karte des Typs ME-63, ME-80 oder ME-81; mögliche Werte sind: 0…3
`<PortNo>`	Port-Name; möglich sind:

<PortNo>	*Beschreibung*
PORTA (00Hex)	16 Bit Eingangsport A
PORTB (01Hex)	16 Bit Eingangsport B

`<Value>`	Zeiger auf einen Integerwert, der das gelesene 16 Bit Wort enthält

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xGetDrvErrMess* ermittelt werden.

_me80DIGetLong

✎ Beschreibung

Funktion gilt für den Kartentyp ME-80.

Liest ein Langwort vom 32 Bit Eingangsport der Karte.

☞ Wichtiger Hinweis:

Zur Konfiguration der Ports muß vorher die Funktion *_me80DIOSetMode* einmalig aufgerufen werden.

● Definitionen

C:	int _me80DIGetLong (int iBoardNumber, int iPortNo, int *iValue);
Delphi:	Function _me80DIGetLong (iBoardNumber, iPortNo: integer; Var iValue: integer): integer;
Basic:	Declare Function me80DIGetLong Lib "me8x_32" Alias "_VBme80DIGetLong@12" (ByVal iBoardNumber As Long, ByVal iPortNo As Long, iValue As Long) As Long

➔ Parameter

`<BoardNumber>`	Kartennummer für 1., 2., 3. oder 4. installierte Karte des Typs ME-63, ME-80 oder ME-81; mögliche Werte sind: 0…3

`<PortNo>`	Port-Name; möglich sind:	
	<PortNo>	*Beschreibung*
	PORTA (00Hex)	32 Bit Eingangsport
`<Value>`	Zeiger auf einen Integerwert, der das gelesene 32 Bit Wort enthält	

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xGetDrvErrMess* ermittelt werden.

_me8xDIOSetIntMode

✎ Beschreibung

Funktion gilt für den Kartentyp ME-80.

Mit dieser Funktion kann das Interruptereignis ausgewählt werden. Diese Funktion wird von graphischen Programmiersprachen nicht unterstützt!

● Definitionen

C:	int _me8xDIOSetIntMode (int iBoardNumber, int iMode);
Delphi:	Function _me8xDIOSetIntMode (iBoardNumber: integer; iMode: integer): integer;
Basic:	Declare Function me8xDIOSetIntMode Lib "me8x_32" Alias "_VBme8xDIOSetIntMode@8" (ByVal iBoardNumber As Long, ByVal iMode As Long) As Long

➔ Parameter

`<BoardNumber>`	Kartennummer für 1., 2., 3. oder 4. installierte Karte des Typs ME-63, ME-80 oder ME-81; mögliche Werte sind: 0…3
`<Mode>`	Interrupt-Modi; mögliche Werte sind

<Mode>	*Beschreibung*
NO_INTERRUPT (00Hex)	kein Interruptbetrieb
INT_IF_PATTERN (20Hex)	Interrupt bei Bitmustergleichheit
INT_EXTERN (60Hex)	externer Interrupt

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xGetDrvErrMess* ermittelt werden.

_me80DIOSetMode

✎ Beschreibung

Funktion gilt für den Kartentyp ME-80.

Konfiguriert die Ports der ME-80 bezüglich Datenbreite, Richtung und Triggerart

☞ Wichtiger Hinweis:

Diese Funktion muß vor allen Bit-/Byte-Lese-/Schreib-Operationen einmalig aufgerufen werden.

● Definitionen

C:	int _me80DIOSetMode (int iBoardNumber, int iMode, int iRegMode);
Delphi:	Function _me80DIOSetMode (iBoardNumber, iMode, iRegMode: integer): integer;
Basic:	Declare Function me80DIOSetMode Lib "me8x_32" Alias "_VBme80DIOSetMode@12" (ByVal iBoardNumber As Long, ByVal iMode As Long, ByVal iRegMode As Long) As Long

➔ Parameter

`<BoardNumber>` Kartennummer für 1., 2., 3. oder 4. installierte Karte des Typs ME-63, ME-80 oder ME-81; mögliche Werte sind: 0…3

`<Mode>` Übernahme der Daten (Triggerart); möglich sind:

- REGMODE_DIRECT (00Hex): die Daten werden unmittelbar nach dem Lesebefehl übernommen
- REGMODE_INTERNAL (01Hex): die Daten werden periodisch mit der steigenden Flanke des internen Taktes übernommen.
- REGMODE_EXTERNAL (10Hex): die Daten werden mit der steigenden Flanke des externen Triggers übernommen

<RegMode> Datenbreite und Datenrichtung der Ports; möglich sind:

<RegMode>	*Beschreibung*
IN32 (00Hex)	32-Bit Eingang
OUT32 (01Hex)	32-Bit Ausgang
IN16 (02Hex)	2 x 16-Bit Eingang
OUT16 (03Hex)	2 x 16-Bit Ausgang
INOUT16 (04Hex)	16-Bit Eingang (Port A), 16-Bit Ausgang (Port B)

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xGetDrvErrMess* ermittelt werden.

_me8xDIOSetPattern

✎ Beschreibung

Funktion gilt für den Kartentyp ME-80.

Funktion schreibt ein 16 bzw. 32 Bit-Wort als Vergleichs-Bitmuster zur Karte. Bei Bitmuster-Gleichheit mit dem korrespondierenden Eingangsport kann ein Interrupt ausgelöst werden (falls freigegeben).

● Definitionen

C: int _me8xDIOSetPattern (int iBoardNumber, int iPattern);

Delphi: Function _me8xDIOSetPattern (iBoardNumber, iPattern: integer): integer;

Basic: Declare Function me8xDIOSetPattern Lib "me8x_32" Alias "_VBme8xDIOSetPattern@8" (ByVal iBoardNumber As Long, ByVal iPattern As Long) As Long

➔ Parameter

<BoardNumber> Kartennummer für 1., 2., 3. oder 4. installierte Karte des Typs ME-63, ME-80 oder ME-81; mögliche Werte sind: 0…3

<Pattern> Vergleichs-Bitmuster (16 bzw. 32 Bit breit), Register wird unabhängig von der Port-Breite durch **einen** Funktionsaufruf geschrieben.

Eingänge DIO0…15 entsprechen den Bits 0…15 des 1. Vergleichsregisters, Eingänge DIO16…31 entsprechen den Bits 0…15 des 2. Vergleichsregisters

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xGetDrvErrMess* ermittelt werden.

_me8xDIOSetTristateOFF

✎ Beschreibung

Funktion gilt für den Kartentyp ME-80.

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 _me8xDIOSetTristateOFF (int iBoardNumber);

Delphi: Function _me8xDIOSetTristateOFF (iBoardNumber: integer): integer;

Basic: Declare Function meDIOSetTristateOFF Lib "me8x_32" Alias "_VBme8xDIOSetTristateOFF@4" (ByVal iBoardNumber As Long) As Long

➔ Parameter

<BoardNumber> Kartennummer für 1., 2., 3. oder 4. installierte Karte des Typs ME-63, ME-80 oder ME-81; mögliche Werte sind: 0…3

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xGetDrvErrMess* ermittelt werden.

_me8xDIOSetTristateON

✎ Beschreibung

Funktion gilt für den Kartentyp ME-80.

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

● Definitionen

C: int _me8xDIOSetTristateON(int iBoardNumber);

Delphi: Function _me8xDIOSetTristateON(iBoardNumber: integer): integer;

Basic: Declare Function me8xDIOSetTristateON Lib "me8x_32" Alias "_VBme8xDIOSetTristateON@4" (ByVal iBoardNumber As Long) As Long

➔ Parameter

`<BoardNumber>` Kartennummer für 1., 2., 3. oder 4. installierte Karte des Typs ME-63, ME-80 oder ME-81; mögliche Werte sind: 0…3

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xGetDrvErrMess* ermittelt werden.

_me8xDOSetBit

✎ Beschreibung

Funktion gilt für den Kartentyp ME-80.

Setzt eine einzelne digitale Ausgangsleitung in den gewünschten Zustand.

☞ Wichtiger Hinweis:

Zur Konfiguration der Ports muß vorher die Funktion *_me80DIOSetMode* einmalig aufgerufen werden und zur Aktivierung der Ausgänge muß die Funktion *_me8xDIOSetTristateOFF* aufgerufen werden.

● Definitionen

C: int _me8xDOSetBit (int iBoardNumber, int iBitNo, int iBitValue);

Delphi: function _me8xDOSetBit (iBoardNumber: integer; iBitNo, iBitValue: integer): integer;

Basic: Declare Function me8xDOSetBit Lib "me8x_32" Alias "_VBme8xDOSetBit@12" (ByVal iBoardNumber As Long, ByVal iBitNo As Long, ByVal iBitValue As Long) As Long

→ Parameter

`<BoardNumber>` Kartennummer für 1., 2., 3. oder 4. installierte Karte des Typs ME-63, ME-80 oder ME-81; mögliche Werte sind: 0…3

`<BitNo>` Nummer der Ausgangsleitung, die gesetzt werden soll; möglich sind:

0…15: 16 Bit-Ausgangsport (Port A) bzw. niederwertige 16 Bits des 32 Bit Ausgangsports

16…31: 16 Bit-Ausgangsport (Port B) bzw. höherwertige 16 Bits des 32 Bit Ausgangsports

`<BitValue>` Mögliche Werte sind:

0: Bit wird auf logisch '0' gesetzt

1: Bit wird auf logisch '1' gesetzt

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xGetDrvErrMess* ermittelt werden.

_me8xDOSetWord

✎ Beschreibung

Funktion gilt für den Kartentyp ME-80.

Schreibt ein Wort an einen 16-Bit-Ausgangsport.

☞ Wichtiger Hinweis:

Zur Konfiguration der Ports muß vorher die Funktion *_me80DIOSetMode* einmalig aufgerufen werden und zur Aktivierung der Ausgänge muß die Funktion *_me8xDIOSetTristateOFF* aufgerufen werden.

● Definitionen

C:	int _me8xDOSetWord (int iBoardNumber, int iPortNo, int iValue);
Delphi:	Function _me8xDOSetWord (iBoardNumber, iPortNo, iValue: integer): integer;
Basic:	Declare Function me8xDOSetWord Lib "me8x_32" Alias "_VBme8xDOSetWord@12" (ByVal iBoardNumber As Long, ByVal iPortNo As Long, ByVal iValue As Long) As Long

➔ Parameter

`<BoardNumber>`	Kartennummer für 1., 2., 3. oder 4. installierte Karte des Typs ME-63, ME-80 oder ME-81; mögliche Werte sind: 0…3
`<PortNo>`	Portname; mögliche Werte sind:

<PortNo>	*Beschreibung*
PORTA (00Hex)	16-Bit Ausgangsport A
PORTB (01Hex)	16-Bit Ausgangsport B

`<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 *_me8xGetDrvErrMess* ermittelt werden.

_me80DOSetLong

✎ Beschreibung

Funktion gilt für den Kartentyp ME-80.

Schreibt ein Langwort an den 32 Bit Ausgangsport der Karte.

☞ Wichtiger Hinweis:

Zur Konfiguration der Ports muß vorher die Funktion *_me80DIOSetMode* einmalig aufgerufen werden und zur Aktivierung der Ausgänge muß die Funktion *_me8xDIOSetTristateOFF* aufgerufen werden.

● Definitionen

C:	int _me80DOSetLong (int iBoardNumber, int iPortNo, int iValue);
Delphi:	Function _me80DOSetLong (iBoardNumber, iPortNo, iValue: integer): integer;
Basic:	Declare Function me80DOSetLong Lib "me8x_32" Alias "_VBmeDOSetLong@12" (ByVal iBoardNumber As Long, ByVal iPortNo As Long, ByVal iValue As Long) As Long

➔ Parameter

`<BoardNumber>` Kartennummer für 1., 2., 3. oder 4. installierte Karte des Typs ME-63, ME-80 oder ME-81; mögliche Werte sind: 0…3

`<PortNo>` Port-Name; möglich sind:

<PortNo>	*Beschreibung*
PORTA (00Hex)	32 Bit Ausgangsport

`<Value>` 32 Bit Ausgabewert; mögliche Werte sind: 0Hex…FFFFFFFFHex (dezimal: 0…4294967295)

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xGetDrvErrMess* ermittelt werden.

5.3.3 Interrupt-Verarbeitung

_me8xDisableInt

✎ Beschreibung

Funktion gilt für den Kartentyp ME-80.

Deaktivierung der Interruptsteuerung. Beendet eine mit *_me8xEnableInt* gestartete Interruptsteuerung.

☞ Wichtiger Hinweis!

Diese Funktion wird nicht in HP VEE unterstützt!

● Definitionen

C: int _me8xDisableInt (int iBoardNumber);

Delphi: Function _me8xDisableInt (iBoardNumber: integer):integer;

Basic: nicht realisiert

➔ Parameter

`<BoardNumber>` Kartennummer für 1., 2., 3. oder 4. installierte Karte des Typs ME-63, ME-80 oder ME-81; mögliche Werte sind: 0…3

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xGetDrvErrMess* ermittelt werden.

_me8xEnableInt

✎ Beschreibung

Funktion gilt für den Kartentyp ME-80.

Die Interruptsteuerung der Karte wird aktiviert. Bei auftretendem Interrupt wird eine vom Anwender definierte Interrupt-Service-Routine ausgeführt.

☞ Wichtiger Hinweis!

Diese Funktion wird nicht in HP VEE unterstützt!

● Definitionen

C: int _me8xEnableInt (int iBoardNumber, PSERVICE_PROC IrqFunc);

Delphi: Function _me8xEnableInt (iBoardNumber: integer; IrqFunc: Pointer): integer;

Basic: nicht realisiert

➔ Parameter

`<BoardNumber>` Kartennummer für 1., 2., 3. oder 4. installierte Karte des Typs ME-63, ME-80 oder ME-81; mögliche Werte sind: 0…3

`<IrqFunc>` Adresse einer anwenderdefinierten Funktion vom Typ (void PSERVICE_PROC (void)) für C bzw. vom Typ Pointer für Delphi, die bei einem Interrupt ausgeführt wird.

‹ Rückgabewert

Wurde die Funktion erfolgreich ausgeführt, so wird 1 zurückgegeben. Im Fehlerfall wird 0 zurückgegeben. Die genaue Fehlerursache kann dann über *_me8xGetDrvErrMess* ermittelt werden.

5.3.4 Fehler-Behandlung

_me8xGetDrvErrMess

✎ Beschreibung

Funktion gilt für den Kartentyp ME-80.

Falls bei der unmittelbar vorher aufgerufenen API-Funktion des ME-8x 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-DLL fehlerhaft (d. h. Funktionswert 0) ausgeführt wurde!

● Definitionen

C: int _me8xGetDrvErrMess (char *pcErrortext);

Delphi: Function _me8xGetDrvErrMess (Var errortext: errorstring): integer;

Basic: Declare Function me8xGetDrvErrMess Lib "me8x_32" Alias "_VBme8xGetDrvErrMess@4" (ByVal errortext As String) As Long

➔ Parameter

`<Errortext>` Zeiger auf Fehlertext; der Fehlercode wird als Funktionswert zurückgegeben.

‹ Rückgabewert

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

Anhang

A Spezifikationen

PC Interface

Typ	ISA-16 Bit
Basisadresse	Im Bereich 0000…1FE0Hex in Schritten von 20Hex einstellbar (Jumper)
Adreßraum	16 Byte
Interrupt	2, 3, 5, 7, 10, 11, 12, 15 wählbar (Jumper)

Digitale Ein-/Ausgänge

Portkonfiguration	2 x 16 Bit Eingangsports 2 x 16 Bit Ausgangsports 1 x 32 Bit Eingangsport 1 x 32 Bit Ausgangsport 1 x 16 Bit Eingangsport (Port A) und 1 x 16 Bit Ausgangsport (Port B)
Pufferung	Treiberbaustein 74BCT245
Eingangsspannung	Low: -0,5 V…+0,8 V (I_{ILmax} =±10 µA) High: +2,0 V…+5,5 V (I_{IHmax} =±10 µA)
Ausgangsspannung	Low: max. +0,45 V (I_{OL}=+2,5 mA) High: min. +2,4 V (I_{OH}=-2,5 mA)
Betriebsarten	Digitale Ein-/Ausgabe (je nach Portkonfiguration und Triggerart); Programmierbarer 16/32 Bit breiter Bitmustervergleicher (je nach Portkonfiguration und Triggerart)
Interrupt-Ereignis	bei Bitmuster-Gleichheit; oder über ext. Triggereingang direkt zum System weitergeleitet
Triggerarten	Ohne Trigger: Daten werden durch Lesezyklus gelatcht; Interner Trigger: Daten werden durch 2,5 MHz Takt gelatcht; Externer Trigger: Daten weden durch externes Triggersignal gelatcht
Ext. Triggerpegel	TTL-Pegel (siehe Digital-I/O Spezifik.)

Allgemeine Daten

Stromverbrauch	typ. 0,7 A (ohne Last)
Platinengröße	100 mm x 160 mm (ohne Slotblech und Busstecker)
Anschlüsse	37polige 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 Anschlußstecker (37polige Sub-D Buchse)

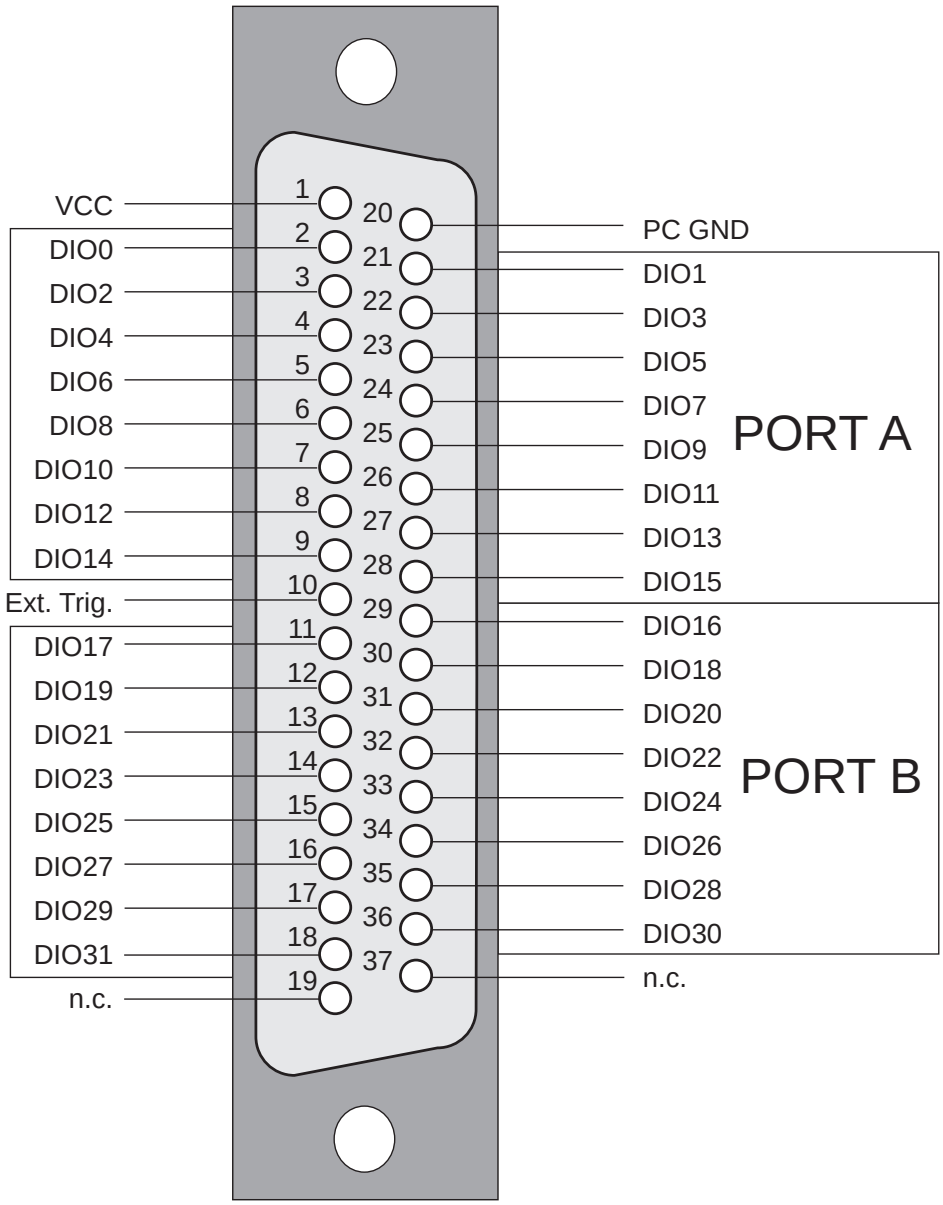


Abb. 5: 37polige Sub-D Buchse

C Zubehör

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

ME-AB-D37 M

Anschluß-Block mit 37poligem Sub-D Stecker

ME-AK-D37

37poliges Anschluß-Kabel mit Sub-D Steckverbinder (Stecker - Buchse), 2 m

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:

Meilhaus Electronic GmbH

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

Auf unserem FTP-Server stehen die aktuellen Treiberversionen rund um die Uhr für Sie bereit. Zuvor müssen Sie sich jedoch über unsere Homepage (http://www.meilhaus.de) anmelden, da der Zugang zum FTP-Server mit einem Paßwort geschützt ist. Dieses erhalten Sie dann umgehend per eMail oder Fax. In besonders dringenden Fällen auch telefonisch unter:
(++49) (0)89/ 89 01 66-0)

E Index