Schnittstellenbeschreibung des Windows-95 Treibers

VPCIVMED.VXD

für das PCI-VME Interface von ARW-Elektronik

ARW - Elektronik
Schlehenweg 1
D-69168 Wiesloch
Germany
Fax+Tel. 06222/50816

Veränderungen

20.07.1998 Erstellung des Dokuments.
12.08.1998 Dokumentation zu Interrupt Behandlung hinzugefügt.
19.08.1998 Dokumentation zur Standard Initialisierung hinzugefügt.
11.03.1999 Veränderungen, um mehr Kompatibilität zu erreichen.
08.04.1999 Pfadstrukturen den veränderten Gegebenheiten angepasst.

Inhaltsverzeichnis

1. LIEFERUMFANG

Auf der mitgelieferten Diskette finden sich folgende Inhalte:
Im Verzeichnis WIN95\DRIVER:

vpcivmed.vxd	-	der Treiber
pcivme.inf	-	die 'INF' Datei zur Installation

Im Verzeichnis WIN95\DRIVER\SOURCE:

vpcivmed.h	-	die Header-Datei zur Schnittstelle zum Treiber.
vic.h	-	die Header-Datei zum VIC68A Baustein.
vme.h	-	die Header-Datei zu VME-Adreßbereichen.
	-	die Quellen zum vpcivmed.vxd Treiber

Im Verzeichnis WIN95\PVMON

pvmon.exe	-	ein nützliches Programm.

Im Verzeichnis WIN95\PVMON\SOURCE

	-	die Quellen zu pvmon.exe

2. INSTALLATION

Hinweis: Der Treiber arbeitet nur unter Windows95 und nur im 32-bit Modus. D.h. der Treiber kann von echten 32-bit Applikationen genutzt werden. Darunter fallen auch sogenannte Konsole-Applikationen, nicht jedoch MS-DOS Programme und WIN3.11 Programme.
Der Treiber kann unter WINDOWS-NT nicht genutzt werden. Hierzu ist ein anderer Treiber geplant. Die Schnittstelle zum Treiber ist jedoch Windows-NT kompatibel.

Installieren Sie die PCIVME-Hardware nach Anleitung. Lassen Sie Ihren Rechner unter WINDOWS-95 starten. Es erscheint nach der Installation beim ersten „Boot“ eine Meldung, daß eine neue Hardware erkannt wurde.
Es wird nach einer Diskette gefragt. Legen Sie die mitgelieferte CD in das Laufwerk (D): ein und suchen Sie nach der zugehörigen ‘INF’ Datei. Diese finden Sie unter „WIN95\DRIVER“ auf der Diskette.

Nach dem Bestätigen wird der Treiber nach „WINDOWS\SYSTEM\VPCIVMED.VXD“ kopiert und das Interface ist in der „Windows-Registry“ eingetragen. (Eintrag unter HKEY_LOCAL_MACHINE\ENUM\PCI\VEN10B5&DEV9050....).

Auch finden Sie das Interface unter „Start/Systemsteuerung/System/Geräte-Manager“ angezeigt.

3. INSTALLATIONSTEST

Mit Hilfe des Programms „pvmon“ können Sie einige einfache Tests auf dem VMEbus durchführen. Am Besten lassen sich diese Tests mit dem Test- und Diagnosewerkzeug VDIS von ARW durchführen.

Öffnen Sie eine DOS-Box unter Windwos95 und starten Sie dort die Konsole Applikation „pvmon.exe“. Die Abfrage „pvmon -?“ gibt eine Hilfe an der Kommandozeile. Die Eingabe von „?“ hilft dem Benutzer nach dem Aufruf weiter.

Der Treiberpfad muß, wenn der Treiber sich nicht in C:\WINDOWS\SYSTEM befindet, angegeben werden. Dann ist mit „c“ (wie configure) die Einstellung des pvmon zu setzen. Bitte beenden Sie pvmon nach der Einstellung und Sicherung der Konfiguration und starten Sie das Programm erneut.

4. ARBEITSWEISE DES TREIBERS

Der Treiber „vpcivmed.vxd“ ist ein dynamisch ladbarer Treiber. Dies bedeutet, daß der Treiber nicht beim Start des Betriebssystems angemeldet werden muß, sondern beim ersten Aufruf geladen wird und bei der letzten Freigabe wieder aus dem Speicher entfernt wird. Damit ist auch eine De-installation des Treibers einfach und für den Benutzer transparent: Der Treiber muß einfach gelöscht werden!

Der Treiber kann von mehreren Anwendungsprogrammen gleichzeitig genutzt werden. Er unterstützt keine asynchronen Aufrufe. (Asynchrone Aufrufe kehren sofort aus dem Treiber zurück und bieten das Ergebnis des gegebenen Dienstes zum späteren Zeitpunkt an.)

Der Treiber bietet Ihren Anwendungsprogrammen eine gesicherte Schnittstelle zur Hardware. Er arbeitet im Ring-0 des Prozessors und bietet die Resource „PCI-VME-Interface“ gleichberechtigt allen Programmen, die das Interface nutzen wollen, an.

Der Treiber benötigt keine Information über die Adreßeinstellung des Interface. Sämtliche Information erfragt der Treiber aus der „Windows-Registry“.

Der Treiber wird über Standard-IO-Funktionen von Windows95 angesprochen und ist daher unabhängig von jeder benutzten Programmiersprache. Die Strukturen und Konstanten der Header-Dateien sind jedoch nur für C oder C++ angeboten, können jedoch leicht auf andere Syntax angepaßt werden.

Der Treiber unterstützt gleichzeitig bis zu VPCIVMED_MAX_PCIADA Interface Einsteckkarten in den PC und VPCIVMED_MAX_VMEMM Einsteckkarten in den VME-BUS. (Die Konstanten VPCIVMED_MAX_PCIADA etc. sind in der Datei „vpcivemd.h“ definiert.) Die Interface werden alleinig durch die Modul-Nummer der angeschlossenen VMEMM unterschieden.

Eine Besonderheit des Treibers unterstützt den direkten Zugriff von Anwendungsprogrammen auf den VME-BUS ohne sich weiter um die Besonderheiten der PCI-VME Hardware kümmern zu müssen. Hierzu fordert das Anwendungsprogramm ein definiertes „Fenster“ in den VME-BUS an. Dieses Fenster besitzt eine festgelegte Größe, einen zugeordneten Adreß-Modifier und einen Offset in die Adressierung des VME-BUS. Der Treiber gibt einen Zeiger zurück von dessen Basis genau der angeforderte Bereich linear adressiert werden kann. Es können gleichzeitig mehrere solcher „Fenster“ aktiv sein und der wahlfreie Zugriff innerhalb der Fenster ist jederzeit möglich. Das hierzu notwendige Umschalten von Speicherseiten und Adreß-Modifier übernimmt ohne weiteres Zutun des Anwendungsprogramms der Treiber.

4.1. Bei dem Start des Treibers

„vxd_Handle = CreateFile(VxDpathName,0,0,NULL,0,
FILE_FLAG_DELETE_ON_CLOSE,NULL);“

erfaßt dieser die Anzahl und die Modul-Nummern der angeschlossenen VMEMM-Module. Nun können zu den PCIADA-Interfaces 0 bis (VPCIVMED_MAX_PCIADA - 1) die jeweiligen statischen Zustände erfragt werden. Mit Hilfe dieser Zustände kann die Zuordnung zwischen VMEMM Modul-Nummer und der PCIADA Interface-Nummer 0 bis (VPCIVMED_MAX_PCIADA - 1) festgestellt werden.

4.2. Alle weiteren Einstellzugriffe auf den Treiber werden mit Hilfe der Standard-Funktionen

„result = DeviceIoControl(vxd_Handle, ...);

durchgeführt. Der erste Übergabeparameter der Eingangsstruktur bezeichnet immer ein spezielles VMEMM-Interface (0 bis (VPCIVMED_MAX_VMEMM - 1)). D.h. der Treiber stellt selbst die Verbindung zwischen VMEMM-Interface und seiner zugehörigen PCIADA-Einsteckkarte her.

4.3. Nach Beendigung der Nutzung muß der Treiber mit

„CloseHandle(vxd_Handle);“

abgemeldet werden. Der Treiber entfernt sich aus dem Speicher, wenn das letzte Anwendungsprogramm den Pfad geschlossen hat.

Es sind mehrere Dienste zur Nutzung mit den Einstellzugriffen (DeviceIoControl(....)) festgelegt. Diese Dienste werden durch festgelegte Nummern unterschieden (z.B. #define VPCIVMED_GET_STATIC_STATUS ....). Zu jedem Dienst assoziieren in der Regel noch eine Eingangs- und eine Ausgangsstruktur. Zum Aufruf der jeweiligen Dienste muß dann die Dienstekennung, ein Zeiger auf die Eingangsstruktur, die Größe der Eingangsstruktur, ein Zeiger auf eine Ausgangsstruktur, die Größe der Ausgangsstruktur und einen Zeiger auf die Größe der zurückgegebenen Daten mitgegeben werden.
Der Aufruf von „DeviceIOControl(..)“ liefert einen Ergebniswert zurück. Ein Ergebniswert von 0 zeigt die ordentliche Abarbeitung an. Ansonsten kann die Nummer des aufgetretenen Fehlers mit der Funktion „GetLastError()“ erfragt werden. Die Fehlernummern sind in „winerror.h“ definiert.

Beispiel:

```
VPCIVMED_STANDARD_COMMAND sInterface;
VPCIVMED_VECTOR_LEVEL     sVectorLevel;
DWORD                     DIOC_count;
DWORD                     dwResult;
*
*
sInterface.dwInterface = 1;        // selection of 1st VMEMM

// poll if an interrupt is pending ----------
dwResult = DeviceIoControl(vxd_Handle, VPCIVMED_READ_VECTOR,
     &sInterface, sizeof(sInterface), &sVectorLevel, sizeof(sVectorLevel),
          &DIOC_count, NULL);
```

```
if (!dwResult)
  printf(„Error %d occured\n“, GetLastError());
else
  printf(„I have read a vector %d at level %d\n“, sVectorLevel.dwStatusID, wLevel);

*
*
```

Es wird der Dienst „VPCIVMED_READ_VECTOR“ aufgerufen.
Die Eingangsstruktur besteht hier nur aus einem einzigen DWORD, nämlich der VMEMM-Interface-Nummer.
Die Größe der Eingangsstruktur ist „sizeof(sInterface)“.
Als Ausgangsstruktur wird (wie erwartet) die Struktur „VPCIVMED_VECTOR_LEVEL sVectorLevel“ angeboten.
Die Ausgangsstruktur hat die Größe „sizeof(sVectorLevel)“.
In der Variablen „DIOC_count“ legt der Treiber die Größe der wirklich zurückgelieferten Daten ab.

5. DIE DIENSTE

Hinweis: Die verwendeten Konstanten sind in der Datei „vpcivmed.h“ definiert.

5.1. Der Dienst „VPCIVMED_INIT_HARDWARE“ initialisiert eine Hardware eines speziellen VMEMM Interface. In der Regel reicht die Standard Initialisierungsfolge des Treibers aus. Der Benutzer kann jedoch ein terminiertes Array von Initialisierungen (VPCIVMED_INIT_COMMAND) der Standard Initialisierung hinzufügen. Die Terminierung des Arrays muß mit der STOP Konstante enden.
Beispiel:

```
struct
{
        DWORD dwInterface;
        VPCIVMED_INIT_ELEMENT sVIC[3];
} sUserInitStruct  = {0, {{VIC,  BYTE_ACCESS,    0x57, 0xAA},
                          {VIC,  BYTE_ACCESS,    0x53, 0x00},
                          {STOP, WORD_ACCESS, 0x00,  0x00}}};
```

Hinweis: Keine zusätzliche Initialisierung besteht nur aus einem Element mit der STOP Konstante als Inhalt.
Ein VMEMM Interface wird nur zum ersten Aufruf initialisiert. Alle weiteren Aufrufe initialisieren die Hardware nicht mehr, es sein denn vorher wurde die Hardware deinitialisiert.

5.2. Mit Hilfe des „VPCIVMED_DEINIT_HARDWARE“ wird das spezielle PCIADA Interface mit seinem angeschlossenen VMEMM Interface deinitialisiert. Auch hier kann eine benutzerdefinierte der Standard Deinitialisierung hinzugefügt werden. Der Mechanismus ist der Gleiche wie bei der Initialisierung.

5.3. Mit Hilfe des Dienstes „VPCIVMED_ATTACH_WINDOW“ können bis zu „VPCIVMED_MAX_WINDOWS“ Zugriffsfenster in den VME-BUS angefordert werden. Es ist die Struktur „VPCIVMED_ADD_WINDOW“ assoziiert.
Dem angeforderten Fenster wird der zugehörige Adreß-Modifier, seine Basis in den VME-BUS und seine Größe mitgegeben. Die Basis und die Größe müssen auf einer 4-kByte Grenze beginnen bzw. enden. Hierzu werden die Macros PAGE_BASE(..) und PAGE_SIZE(..) zur verfügung gestellt.
Es wird dann vom Treiber ein Zeiger zurückgegeben. Über diesen Zeiger läßt sich nun der VME-BUS direkt erreichen. Dabei ist die komplette Größe des Fensters ausgehend von dem Basiszeiger erreichbar. Sämtliches Umschalten von Seiten Adressen oder Adreß-Modifiern geschieht transparent für den Zugreifenden. Zugriffe außerhalb des zugeordneten Fensters werden durch Zugriffsfehler abgewiesen.
Bus-Fehler des VME-BUS werden nicht als Windows-Zugriffsfehler interpretiert!
Die Größe eines angeforderten Fensters darf maximal 256 MByte betragen.

5.4. Der Dienst „VPCIVMED_DETATCH_WINDOW“ meldet ein zugeordnetes Zugriffsfenster wieder ab. Es ist die Struktur „VPCIVMED_REMOVE_WINDOW“ assoziiert.

5.5. Der Dienst „VPCIVMED_GET_STATIC_STATUS“ fordert statische Informationen zu einem speziellen VMEMM Interface an. Es ist die Struktur „VPCIVMED_STATIC_STATUS“ zugeordnet.

5.6. Der Dienst „VPCIVMED_GET_DYNAMIC_STATUS“ fordert im Betrieb sich ändernde Information an. Es ist die Struktur „VPCIVMED_DYNAMIC_STATUS“ zugeordnet.

5.7. Der Dienst „VPCIVMED_READ_VECTOR“ liest Informationen zu den Interrupts aus. Es ist die Struktur „VPCIVMED_VECTOR_LEVEL“ zugeordnet.

5.8. Mit Hilfe des Dienstes „VPCIVMED_ACCESS_VIC68A“ kann direkt der VIC68A Baustein eines VMEMM-Interface programmiert und ausgelesen werden. Es ist die Struktur „VPCIVMED_VIC68A_ACTION“ als Ein- und Ausgangsstruktur zugeordnet. Um die Vielfältigkeit des VIC68A-Bausteins nutzen zu können wurde der direkte Zugriff darauf freigegeben. Mit einem Zugriff kann der übergebene Inhalt gelesen, geschrieben, verodert oder verundet geschrieben und das Ergebnis erneut zurück gelesen werden. Es darf jedoch kein VIC68A Register verändert werden, welches direkten Einfluß auf das Adreß-Modifier Register des VIC68A hat. Dadurch kann der Mechanismus des VME Fenster Abbildung nachteilig beeinflußt werden.

5.9. Mit Hilfe des Dienstes „VPCIVMED_INSTALL_IRQ_HANDLER“ kann ein Interrupt-Handler auf Benutzer Ebene installiert werden. Beim Aufruf des Dienstes speichert der Treiber den TCB (Thread Control Block) des aufrufenden Threads. Der installierte Interrupt-Handler kann immer dann angesprungen werden wenn ein Interrupt Ereignis ausgelöst ist, der Interrupt zugelassen und der Thread „alarmierbar“ (alertable) ist.
Der Interrupt-Handler wird sowohl von PCIADA als auch von VMEMM Interrupts angesprungen. In einem Übergabeparameter in den Interrupt Handler wird die Interrupt Quelle kodiert: Im niederwerigsten Wort ist der Vector, im höherwertigsten Wort die Interrupt-Ebene kodiert.

Nutzung		unbenutzt	Interrupt-Ebene	unbenutzt	Interrupt-Vektor
Bits		31:19	18:16	15:8	7:0

Auch „Bus-Error“ werden in VMEMM als Interrupts gespiegelt. Interrupts von PCIADA kommend, werden auf eine virtuelle Interrupt Ebene „8“ abgebildet. Ein Interrupt von VMEMM kommend wird im Treiber gesperrt und muß vom Anwendungsprogramm erneut zugelassen werden.

5.10 Hierzu gibt es den Dienst „VPCIVMED_CONTROL_INTERRUPTS“. Damit können gleichzeitig Interrupts von VMEMM und PCIADA selektiv zugelassen oder gesperrt werden.

5.11 Mittels des Aufruf VPCIVMED_TAS kann ein ununterbrechbarer Lese-Schreib-Zyklus, ähnlich der TAS Instruktion der 68K Prozessoren auf dem VME-BUS ausgelöst werden.

5.12 Der Aufruf von VPCIVMED_GET_PCIADA_STATUS liefert den Status aller PCIADA Interfaces in dem Rechner zurück. Damit kann vom Anwendungsprogramm festgestellt werden, welche VMEMM angeschlossen und betriebsbereit sind.

6. INTERRUPT VEKTOREN

Von VMEMM erzeugte oder weitergeleitete Interrupts müssen vektorisiert werden. Per Konvention werden die Interrupt Vektoren so aufgeteilt, daß die Vektoren 0x00..0x3f der internen (d.h. vom Treiber vorbelegten) Nutzung vorbehalten ist. Die Vektoren 0x40..0xff können von Peripherie auf dem VMEbus genutzt werden.
Hinweis: Der von PCIADA generierte Timeout Interrupt wird von der Software auf die Interrupt Vektor Nummer 1 gespiegelt.

Interrupt Ursache	**Vektor Nr.**
PCIADA erzeugter Interrupt (Timeout)	1 (aktiviert)
Clock Tick Interrupt Generator	2
Reset Taster an der Frontplatte	6 (aktiviert)
VMEbus Timeout (Bus-Error)	7 (aktiviert)
Interprocess communication global switch #0	8
Interprocess communication global switch #1	9
Interprocess communication global switch #2	10
Interprocess communication global switch #3	11
Interprocess communication module switch #0	12
Interprocess communication module switch #1	13
Interprocess communication module switch #2	14
Interprocess communication module switch #3	15
ACFAIL asserted	16
Write post Fail	17
Arbitration Timeout	18
SYSFAIL asserted	19
VMEbus Interrupter acknowledge	20

Hinweis: Der von dem Reset Taster an der Front des VMEMM Interface ausgelöste Interrupt kann dazu verwendet werden, eine benutzerdefinierte Reset- oder Initialisierungsfunktion auszulösen.

7. DIE STANDARD INITIALISIERUNG DES TREIBERS

Der Treiber nutzt folgendes Array von VPCIVMED_INIT_COMMANDs zur Initialisierung des Interface. Wie weiter oben gezeigt kann diese Initialisierung von dem Anwenderprogramm mit Hilfe des Dienstes VPCIVMED_INIT_HARDWARE ergänzt oder überschrieben werden.

Hinweis: Die verwendeten Konstanten sind in der Datei „vpcivmed.h“ definiert.

```
{LCR,  WORD_ACCESS, 0x4c, 0x0009}            // disable interrupts
{LCR,  WORD_ACCESS, 0x50, 0x4180}            // enable interface

{VIC,  BYTE_ACCESS, (WORD)0x03, 0xf8+1}    // VIICR

{VIC,  BYTE_ACCESS, (WORD)0x07, 0x78+1}    // VICR1
{VIC,  BYTE_ACCESS, (WORD)0x0b, 0x78+2}
{VIC,  BYTE_ACCESS, (WORD)0x0f, 0x78+3}
{VIC,  BYTE_ACCESS, (WORD)0x13, 0x78+4}
{VIC,  BYTE_ACCESS, (WORD)0x17, 0x78+5}
{VIC,  BYTE_ACCESS, (WORD)0x1b, 0x78+6}
{VIC,  BYTE_ACCESS, (WORD)0x1f, 0x78+7}     // VICR7

{VIC,  BYTE_ACCESS, (WORD)0x23, 0xf8+0}     // DSICR

{VIC,  BYTE_ACCESS, (WORD)0x27, 0xf8+1}     // LICR1
{VIC,  BYTE_ACCESS, (WORD)0x2b, 0xf8+2}
{VIC,  BYTE_ACCESS, (WORD)0x2f, 0xf8+3}
{VIC,  BYTE_ACCESS, (WORD)0x33, 0xf8+4}
{VIC,  BYTE_ACCESS, (WORD)0x37, 0xf8+5}
{VIC,  BYTE_ACCESS, (WORD)0x3b, 0x38+6}
{VIC,  BYTE_ACCESS, (WORD)0x3f, 0x38+7}     // LICR7

{VIC,  BYTE_ACCESS, (WORD)0x43, 0xf8+2}     // ICGS
{VIC,  BYTE_ACCESS, (WORD)0x47, 0xf8+3}     // ICMS

{VIC,  BYTE_ACCESS, (WORD)0x4b, 0xe8+6}     // EGICR

{VIC,  BYTE_ACCESS, (WORD)0x4f, 0x08}        // ICGS-IVBR (!)
{VIC,  BYTE_ACCESS, (WORD)0x53, 0x0c}        // ICMS-IVBR (!)

{VIC,  BYTE_ACCESS, (WORD)0x57, 0x00}        // LIVBR (!)

{VIC,  BYTE_ACCESS, (WORD)0x5b, 0x10}        // EGIVBR (!)
```

```
    {VIC,  BYTE_ACCESS, (WORD)0x5f, 0x00}       // ICSR

    {VIC,  BYTE_ACCESS, (WORD)0x63, 0x00}       // ICR0
    {VIC,  BYTE_ACCESS, (WORD)0x67, 0x00}
    {VIC,  BYTE_ACCESS, (WORD)0x6b, 0x00}
    {VIC,  BYTE_ACCESS, (WORD)0x6f, 0x00}
    {VIC,  BYTE_ACCESS, (WORD)0x73, 0x00}       // ICR4

    {VIC,  BYTE_ACCESS, (WORD)0x83, 0xfe}       // VIRSR

    {VIC,  BYTE_ACCESS, (WORD)0x87, 0x0f}       // VIVR1
    {VIC,  BYTE_ACCESS, (WORD)0x8b, 0x0f}
    {VIC,  BYTE_ACCESS, (WORD)0x8f, 0x0f}
    {VIC,  BYTE_ACCESS, (WORD)0x93, 0x0f}
    {VIC,  BYTE_ACCESS, (WORD)0x97, 0x0f}
    {VIC,  BYTE_ACCESS, (WORD)0x9b, 0x0f}
    {VIC,  BYTE_ACCESS, (WORD)0x9f, 0x0f}       // VIVR7

    {VIC,  BYTE_ACCESS, (WORD)0xa3, 0x3c}       // TTR  - 16 usec

    {VIC,  BYTE_ACCESS, (WORD)0xb3, 0x40}       // ARCR
    {VIC,  BYTE_ACCESS, (WORD)0xb7, 0x29}       // AMSR
    {VIC,  BYTE_ACCESS, (WORD)0xd3, 0x00}       // RCR

    {IFR,  LONG_ACCESS, (WORD)ADRHL, 0xF0F0F0F0}   // ADR-H, ADR-L
    {IFR,  WORD_ACCESS, (WORD)CSR  , 0x0000}  // Contr-Reg

    {VIC,  BYTE_ACCESS, (WORD)0x7f, 0x80}       // ICR7

    {LCR,  WORD_ACCESS, 0x4c, 0x0009}           // disable interrupts

    {STOP, WORD_ACCESS, 0,     0}
```

8. DIE STANDARD DEINITIALISIERUNG DES TREIBERS

Die eingebaute Deinitialisierung des Treibers erfolgt in zwei Schritten. Zwischen diesen beiden Schritten wird die Deinitialisierungs Sequenz des Benutzers eingefügt.

Erste Deinitialisierungs Sequenz:

```
{VIC,  BYTE_ACCESS, (WORD)0x7f, 0x00},      // ICR7 - set SYSFAIL
{LCR,  WORD_ACCESS, 0x4c, 0x0009},          // disable interrupts
{STOP, WORD_ACCESS, 0,    0}};
```

Zweite Deinitialisierungs Sequenz:

```
{LCR,  WORD_ACCESS, 0x50, 0x4080},          // disable interface
{STOP, WORD_ACCESS, 0,    0}};
```

9. EINSCHRÄNKUNGEN

- Es werden bis zu VPCIVMED_MAX_WINDOWS Fenster in den VME-BUS unterstützt.
- Es ist nicht möglich den Verursacher eines Bus-Error Interrupts zu lokalisieren, wenn Fenster eines VMEMM Interfaces in den VME-BUS von mehreren Applikationen gleichzeitig genutzt werden. Es wird dann immer die eine registrierte Rückruffunktion angesprungen.
- Wird dem Interface von Windows kein Interrupt zugeordnet (Zu sehen unter „Start/Systemsteuerung/System/Geräte-Manager/VME/../Resourcen“), dann ist der Treiber auch ohne diese Zuordnung, jedoch ohne Interruptfunktionen, lauffähig.

10. WEITERE LITERATUR

1. Hardware Handbuch des PCI-VME Interface
2. Programmierhandbuch zum VIC068A der Firma Cypress