1. Einführung

Abkürzungen **1-3**

Erstellen von Datenträgerkopien **1-4**

Analogeingänge **1-5**

Massebezogene Eingänge (single-ended) 1-5
Differenzeingänge 1-6
Einschwingzeit 1-7
Sample & Hold 1-7
Meßbereiche und Zahlendarstellung 1-8
Abgleich 1-10

Analogausgänge **1-11**

Abgleich 1-11
Setzen der Ausgänge 1-12

Digitaleingänge **1-13**

TTL-Eingänge 1-13
Schmitt-Trigger-Eingänge 1-14
Optoentkoppelte Eingänge 1-15
Lesen von Digitaleingängen 1-16
Interrupt-Eingänge 1-17
Trigger-Eingänge 1-17
Zähleingänge 1-17
Inkrementalgeberinterface 1-18
Pulsbreiten-, Periodendauer- und Frequenzmessung 1-19

Digitalausgänge **1-20**

TTL-Ausgänge 1-20
Optoentkoppelte Open-Collector-Ausgänge 1-21
Relais-Ausgänge 1-22
Watchdog 1-22

Jumper **1-23**

EEPROM **1-24**

Testen der Module **1-25**

Programmierung der Module **1-26**

Grundlagen zum Modul-Device-Treiber **1-27**

Treiberspezifische Begriffe 1-30
Verwaltung der Devices 1-34
Datentypen 1-35
Kanaleigenschaftsstrukturen (CPS) 1-36
Hinweise zur Verwendung der MDDs 1-39

Modul-Device-Treiber Bibliothek **1-40**

Dienste eines MDD 1-40
Dienste eines Kanals 1-49
Sonderdienste 1-56
Fehlermeldungen 1-63
Diagnosemeldungen 1-64

Hochsprachenbibliotheken **1-65**

Verwendung der Bibliotheken 1-65
Einbinden der Bibliotheken 1-67

Programmierung mit I/O-Zugriffen **1-70**

Abkürzungen

Für die Ein- und Ausgänge der SPB-Module werden in diesem Handbuch die folgenden Abkürzungen verwendet:

AGND	Masse für analogen Ein- oder Ausgang (= Analog Ground)
AIN	Analoger Eingang
AOUT	Analoger Ausgang
DGND	Masse für digitalen Ein- oder Ausgang (= Digital Ground)
DIN	Digitaler Eingang
DIO	Digitaler Ein- und Ausgang
DOUT	Digitaler Ausgang
GND	Masse (= Ground)
INT	Interrupt-Eingang
TRIG	Trigger-Eingang

Eine mittels Bindestrich angehängte Zahl steht für die Nummer des Ein- oder Ausgangs. Die Bezeichnung 'AOUT-3' bedeutet also: Analogausgang Nr. 3.

Erstellen von Datenträgerkopien

Die im Lieferumfang eines Moduls enthaltenen Datenträger sind nicht kopiergeschützt. Die Programme dürfen für Zwecke des Anwenders beliebig oft kopiert werden. Wiederverkäufer können die Programme auch verändern und weiter verkaufen. Eine besondere Genehmigung der SORCUS Computer GmbH ist dazu nicht erforderlich.

Eine Kopie der Datenträger wird, wie im Handbuch zum Betriebssystem Ihres PCs beschrieben, hergestellt. Verwahren Sie die Originale sicher vor schädigenden Einflüssen.

Da sich die Zahl der Programme auf den mitgelieferten Datenträgern entsprechend dem aktuellen Entwicklungsstand ändert, ist an dieser Stelle kein Inhaltsverzeichnis angegeben. Sie können aber jederzeit gegen eine geringe Gebühr eine aktualisierte Version bei Ihrem Händler oder direkt bei der Firma SORCUS Computer GmbH erhalten.

Alle Programme auf den Datenträgern sind als Hilfsmittel gedacht, um Ihnen den Einsatz der MODULAR-4 Karte zu erleichtern und zu verdeutlichen. Eine Garantie für die fehlerfreie Funktion dieser Programme übernimmt die Firma SORCUS Computer GmbH jedoch nicht. Insbesondere werden jede Haftung, Gewährleistung und eventuelle Schadenersatzansprüche, die sich aus dem Einsatz der MODULAR-4 Karte sowie der mitgelieferten Software ergeben könnten, ausgeschlossen.

Analogeingänge

Das System MODULAR-4 bietet Module mit massebezogenen Eingängen und mit Differenzeingängen. Je nach Modul liegt der Eingangstyp fest, oder er kann für alle Eingänge gemeinsam oder für jeden Eingang getrennt festgelegt werden.

Massebezogene Eingänge (single-ended)

Bei einem massebezogenen Eingang wird die Spannung an einem Eingang gegenüber einem Bezugspunkt gemessen, der für alle Eingänge gemeinsam ist. Meist ist dieser Bezugspunkt die Systemmasse. Wenn die Signalquellen nicht von vornherein gemeinsame Massen besitzen, muß darauf geachtet werden, daß beim Zusammenschalten der Quellen keine Kurzschlüsse über die Masseanschlüsse auf dem Modul entstehen. Aus Gründen der Störsicherheit wird empfohlen, die Masseleitungen möglichst nahe am Meßpunkt - also am Modul - zusammenzuführen. In diesem Fall werden die Signal- und Masseleitungen der einzelnen Quellen getrennt zum Eingang geführt und dort angeschlossen. Alle Analogeingänge haben eigene Masseeingänge, die auf dem Modul miteinander verbunden sind. Bei Modulen ohne galvanische Trennung sind alle Masseeingänge außerdem mit der Masse der Basiskarte und damit mit der PC-Masse verbunden.

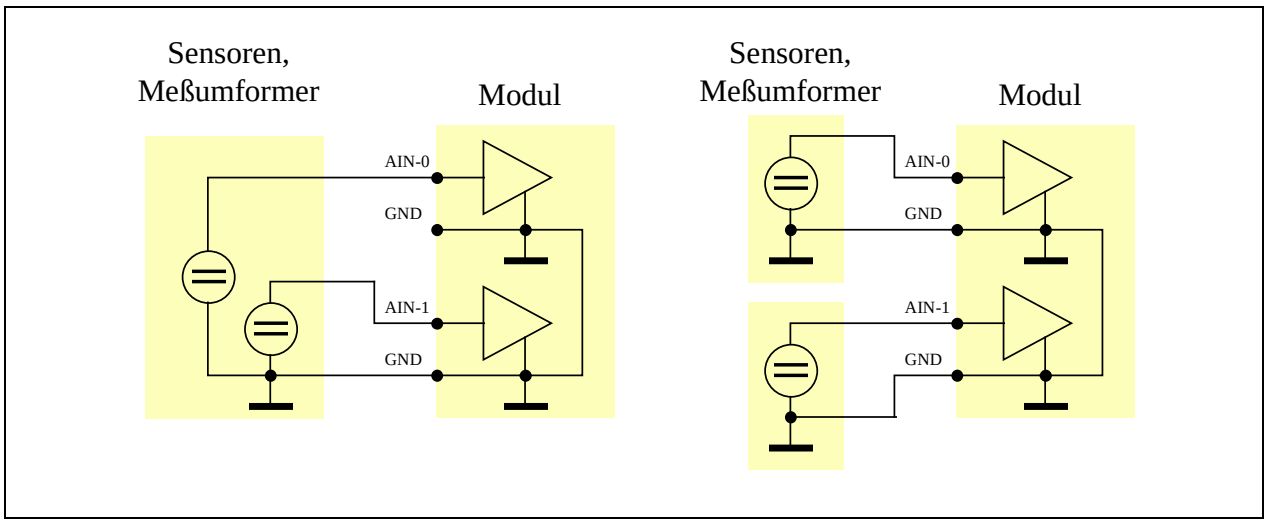


Abb. 1-1: Anschluß von zwei Signalquellen mit gemeinsamer Masse (links) oder getrennter Masse (rechts) an massebezogene Eingänge.

Differenzeingänge

Ein Differenzeingang wird aus zwei massebezogenen Eingängen gebildet, für deren Anschluß das im vorherigen Absatz Gesagte gilt. Der Unterschied ist lediglich, daß auf dem Modul die Differenz zweier Signale gemessen wird und somit 3-polige Signalquellen (2 Signalpole, 1 Pol für Bezugsmasse) angeschlossen werden können. Wichtig ist, daß auf jeden Fall die Bezugsmasse der beiden Signale angeschlossen werden muß.

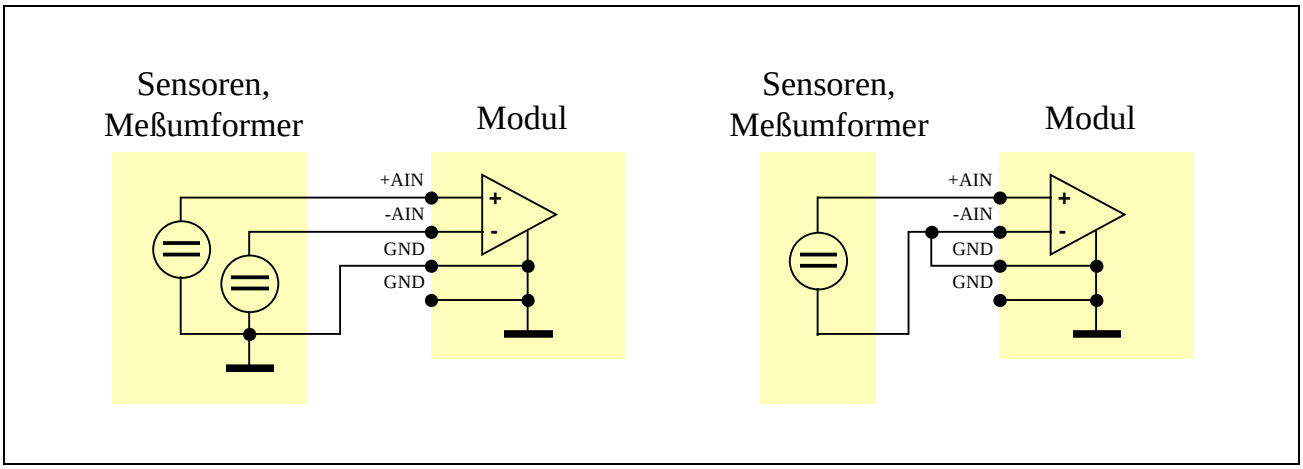


Abb. 1-2: Anschluß einer 3-poligen Signalquelle an einen Differenzeingang (links) und Anschluß einer 2-poligen Signalquelle an einen Differenzeingang (rechts)

- *Bei 2-poligen Signalquellen (z.B. einem Meßverstärker mit galvanischer Trennung oder einer Batterie) ist eine Differenzmessung eigentlich nicht erforderlich. Wenn sie trotzdem verwendet wird, muß einer der beiden Pole zusätzlich mit einem der beiden Masseeingänge verbunden werden (siehe Abbildung 1-2, rechts).*
- *Unabhängig von der Spannungsdifferenz der beiden Signale sind korrekte Messungen nur dann möglich, wenn keine der beiden massebezogenen Spannungen einen bestimmten Pegel (z.B. 10 Volt) überschreitet. Details dazu finden Sie in den technischen Daten der einzelnen Module.*

Einschwingzeit

Alle Module des MODULAR-4 Systems mit analogen Eingängen verfügen jeweils über einen Analog/Digital-Wandler (abgekürzt A/D-Wandler). Um mehrere Eingänge zu realisieren, ist dem A/D-Wandler ein Multiplexer vorgeschaltet, mit dem der zu wandelnde Eingang ausgewählt wird. Nach dem Umschalten auf einen Eingang dauert es eine gewisse Zeit, bis der Ausgang des Multiplexers auf die Spannung des Eingangs (= Signal) eingeschwungen ist. Diese Zeit wird als Einschwingzeit oder Settle-Time bezeichnet. Sie ist von Modul zu Modul verschieden und kann einige Mikrosekunden betragen. Bei einigen Modulen ist sie vom Ausgangswiderstand der Signalquelle abhängig.

Um korrekte Messungen zu erhalten, muß sichergestellt werden, daß die A/D-Wandlung erst nach Ablauf der Settle-Time gestartet wird. Einige Module verfügen zu diesem Zweck über einen sogenannten Settle-Timer. Dieser ist ein programmierbares Zeitglied, das verhindert, daß die A/D-Wandlung nach einer Eingangsumschaltung zu früh gestartet wird. Wenn ein solcher Timer nicht auf dem Modul vorhanden ist, muß die Software dafür sorgen, daß genügend Zeit zum Einschwingen bleibt.

Sample & Hold

Analog/Digital-Wandler benötigen pro Wandlung eine gewisse Zeit. In dieser Zeit darf sich die am A/D-Wandler anliegende Spannung nicht verändern. Dafür sorgt der sogenannte Sample-and-hold-Baustein (abgekürzt mit S&H-Baustein), der den analogen Spannungswert für die Zeit der Wandlung konstant hält.

Wenn mehrere Eingänge zu einem bestimmten Zeitpunkt abgetastet werden müssen, entsteht - bedingt durch die Wandlungs- und Einschwingzeit - ein geringer zeitlicher Versatz zwischen den einzelnen ermittelten Eingangsspannungen, der nicht bei jeder Anwendung toleriert werden kann. Mit einem S&H-Baustein pro Eingang kann dieses Problem gelöst werden. Alle Eingangsspannungen werden zeitgleich in S&H-Bausteinen gespeichert und dann nacheinander gewandelt. Ein Modul mit einem S&H-Baustein pro Eingang ist zum Beispiel das SPB-Modul M-SH12-8.

Meßbereiche und Zahlendarstellung

Analoge Signale können in verschiedenen Meßbereichen erfaßt werden. Die Meßverstärker dafür befinden sich vor dem A/D-Wandler. Die Einstellung des Meßbereichs ist je nach Modul per Software oder mit Jumpern möglich. Module mit per Software einstellbarem Meßbereich erlauben es, jeden Eingang mit unterschiedlichem Meßbereich zu erfassen. Beachten Sie, daß auch nach einer Meßbereichsumschaltung eine gewisse Zeit vergeht, bis die Verstärker neu eingeschwungen sind. Auch hier kann der Settle-Timer verwendet werden.

Die Zuordnung eines von einem A/D-Wandler gelieferten Digitalwertes zu der analogen Eingangsspannung ist vom gewählten Meßbereich abhängig. Die Zahlendarstellung wird von den Hochsprachenbibliotheken vereinheitlicht, so daß sie - unabhängig von der Hardware - für alle Module gleich ist. Den Tabellen auf den folgenden Seiten können Sie die Zuordnung der Spannungen zu den Digitalwerten entnehmen.

12-Bit-A/D-Wandler bei bipolarem Spannungsbereich:

Meßwert		
Dezimal	**Hexadezimal**	**Spannung**[1]
2047	07ffh	0.99976*FS
...	...	...
1	0001h	0,00048*FS
0	0000h	0 V
-1	ffffh	-0,00048*FS
...	...	...
-2048	f800h	-FS

[1] FS = Full-Scale, Meßbereichsendwert als Absolutwert.
Beispiele: Meßbereich ±5 V: FS = 5 V; Meßbereich 0 bis 10 V: FS = 10 V, Meßbereich 0 bis -5 V: FS = 5 V

12-Bit-A/D-Wandler bei unipolar positivem Spannungsbereich:

Meßwert		
Dezimal	**Hexadezimal**	**Spannung**
0	0000h	0 V
1	0001h	0,000244*FS
...	...	...
2047	07ffh	0,49975*FS
2048	0800h	0,5*FS
...	...	...
4095	0fffh	0,99975*FS

12-Bit-A/D-Wandler bei unipolar negativem Spannungsbereich:

Meßwert		
Dezimal	**Hexadezimal**	**Spannung**
0	0000h	0 V
-1	ffffh	...
...	...	...
-2048	f800h	-0,49975*FS
-2049	f7ffh	-0,5*FS
...	...	...
-4095	f001h	-0,99975*FS

16-Bit-A/D-Wandler bei bipolarem Spannungsbereich:

Meßwert		
Dezimal	**Hexadezimal**	**Spannung**
32767	7fffh	$(1-30{,}5*10^{-6})*FS$
...	...	...
1	0001h	$30{,}5*10^{-6}*FS$
0	0000h	0 V
-1	ffffh	$-30{,}5*10^{-6}*FS$
...	...	...
-32768	8000h	-FS

16-Bit-A/D-Wandler bei unipolar positivem Spannungsbereich:

Meßwert		
Dezimal	**Hexadezimal**	**Spannung**
0	0000h	0 V
1	0001h	$15{,}2*10^{-6}*FS$
...	...	...
32767	7fffh	$(0{,}5-15{,}2*10^{-6})*FS$
32768	8000h	0,5*FS
...	...	...
65535	ffffh	$(1-15{,}2*10^{-6})*FS$

! *Das bei direktem Ansprechen der I/O-Adressen des Moduls gelieferte Zahlenformat kann sich von dem der Bibliotheken unterscheiden, da diese eine Normalisierung der Werte durchführen.*

Abgleich

Die meisten Module mit Analogeingängen werden nicht mehr mit Potentiometern, sondern digital abgeglichen. Dabei gibt es Module, bei denen der Abgleich per Hardware durchgeführt wird, so daß man immer einen abgeglichenen Wert erhält, und Module, bei denen der Abgleich nach der Wandlung per Software vorgenommen werden muß. Sowohl die Einstellung der Hardware für den Abgleich als auch die Softwarekorrektur wird von den mitgelieferten Bibliotheken durchgeführt.

Der Abgleich wird mit zwei **Korrekturfaktoren** durchgeführt, einer für Verstärkungsfehler (*GAIN*) und einer für Offset-Fehler (*OFFSET*). Nach welchen Formeln der Meßwert korrigiert wird und wie die Korrekturfaktoren zu bestimmen sind, ist bei den einzelnen Modulen beschrieben. Die Korrekturfaktoren werden für jedes einzelne Modul im Werk bestimmt und in das EEPROM eingetragen.

Analogausgänge

Analogausgänge gibt es als Strom- und als Spannungsausgänge in verschiedenen Bereichen. In der Regel können die Ausgangsbereiche für alle Ausgänge getrennt mit Jumpern eingestellt werden. Die Zahlendarstellung beim Arbeiten mit Hochsprachenbibliotheken ist die gleiche wie bei analogen Eingängen (siehe Seite 1-8). Auch hier kann die Zahlendarstellung anders sein, wenn Sie das Modul direkt mit I/O-Zugriffen ansprechen oder einen Initialisierungswert in das EEPROM eintragen.

Vermeiden Sie eine Fehlbeschaltung eines Ausganges. Für Spannungsausgänge finden Sie in den technischen Daten die maximale Strombelastung. Für Stromausgänge ist die maximale Größe, die ein angeschlossener Lastwiderstand haben darf, angegeben. Die Stromausgänge werden bei zu großem Widerstand nicht zerstört, können aber den gewünschten Strom nicht mehr liefern.

Abgleich

Ebenso wie analoge Eingänge können analoge Ausgänge digital abgeglichen werden. Der Abgleich findet immer per Software statt. Die entsprechenden Formeln finden Sie in den Modulbeschreibungen. Auch hier werden die Korrekturwerte (*GAIN* und *OFFSET*) im EEPROM gespeichert.

Setzen der Ausgänge

Gleichzeitiges Setzen aller Ausgänge wird dadurch erreicht, daß die digitalen Werte zunächst nacheinander für jeden Ausgang in ein Zwischenregister eingetragen werden und dann gleichzeitig in die Ausgaberegister übernommen werden (siehe Abbildung 1-3).

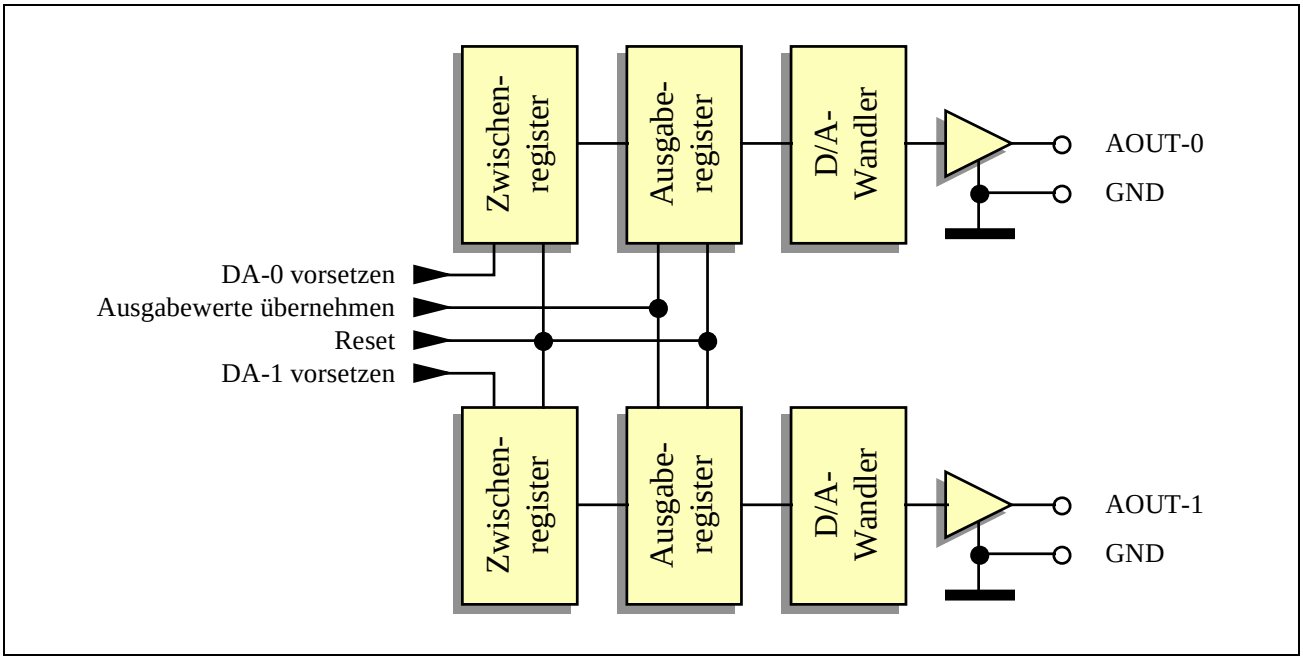


Abb. 1-3: Blockschaltbild für analoge Ausgänge (Beispiel für zwei Ausgänge)

Digitaleingänge

Im MODULAR-4 System stehen verschiedene Eingangstypen zur Verfügung: TTL-Eingänge, Schmitt-Trigger-Eingänge und optoentkoppelte Eingänge, Zähleingänge, Inkrementalgeberinterface, Interrupt-Eingänge und Trigger-Eingänge.

Wichtige Kenngrößen von Digitaleingängen sind die Eingangspegel. Sie geben an, welche Spannung bzw. welcher Strom für eine logische Eins oder eine logische Null stehen.

TTL-Eingänge

Bei TTL-Eingängen werden **Spannungen** für die Unterscheidung der logischen Zustände benutzt. 0 bis 0,8 Volt gelten als logische Null, 2,0 bis 5,0 Volt als logische Eins. Eingangsspannungen zwischen 0,8 und 2,0 Volt sind unzulässig und erzeugen keinen vorhersehbaren Logikzustand. In den technischen Daten zu den einzelnen Modulen können Sie nachlesen, welche Pegel für das jeweilige Modul gelten. Alle Spannungen beziehen sich auf eine gemeinsame Masse, die mit der Masse der Basiskarte verbunden ist.

Wenn z.B. ein Schalter an einen TTL-Eingang angeschlossen werden soll, ist zu beachten, daß der Eingang bei offenem Schalter nicht unbeschaltet sein darf. Unbeschaltete Eingänge verhalten sich undefiniert und sind deshalb nicht zulässig.

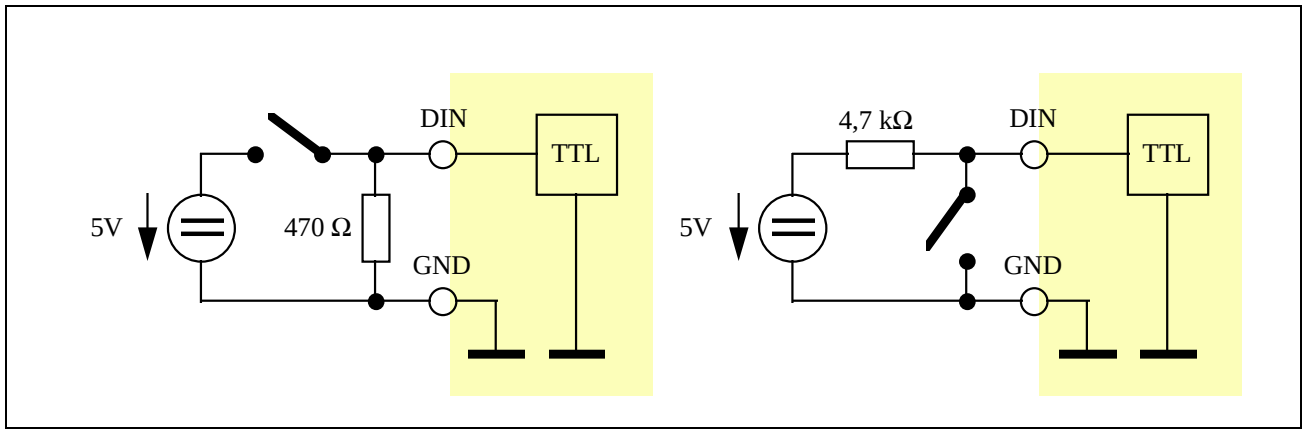


Abb. 1-4: Anschließen eines Schalters an einen TTL-Eingang

Abbildung 1-4 zeigt, wie ein Schalter an einen TTL-Eingang angeschlossen wird. In der linken Schaltung gilt ein geschlossener Schalter als logische Eins, in der rechten als logische Null. Die Widerstände sind unbedingt erforderlich, da der Eingang sonst

bei offenem Schalter nicht beschaltet ist (links) bzw. die Signalquelle bei geschlossenem Schalter kurzgeschlossen wird (rechts).

Schmitt-Trigger-Eingänge

Schmitt-Trigger-Eingänge verhalten sich ähnlich wie TTL-Eingänge. Der Unterschied besteht darin, daß es bei Schmitt-Trigger-Eingängen keinen unzulässigen Eingangsspannungsbereich gibt. Statt dessen werden zwei sogenannte Trigger-Schwellen angegeben, bei deren Über- bzw. Unterschreiten der Eingangspegel von einer logischen Null in eine logische Eins (bzw. umgekehrt) umschaltet. Die Schwellen für Über- und Unterschreiten haben unterschiedliche Werte. Anderenfalls würden bei einem Eingangssignal nahe der Trigger-Schwelle kleinste Störungen genügen, um zwischen logisch Null und Eins umzuschalten. Beim SPB-Modul M-AX-32 zum Beispiel wird beim Überschreiten von 1,6 Volt von logisch Null auf logisch Eins umgeschaltet. Erst wenn die Spannung unter 0,95 Volt fällt, wird wieder auf Null zurückgeschaltet.

Optoentkoppelte Eingänge

Dieser Eingangstyp unterscheidet die logischen Zustände nicht nach Spannungen, sondern danach, ob Strom fließt oder nicht. Je nach Modul werden Ströme innerhalb eines bestimmten Bereichs (z.B. von 5 bis 10 mA) als logische Eins, schwächere Ströme als logische Null gewertet. Die jeweilige Schwelle entnehmen Sie bitte den technischen Daten der einzelnen Module.

Auf den Modulen wird mit Vorwiderständen eingestellt, bei welcher Eingangsspannung der für eine logische Eins notwendige Strom fließt. Dabei handelt es sich in der Regel um Widerstandsnetzwerke, die die Stromanpassung für jeweils vier benachbarte Eingänge übernehmen. Widerstandsnetzwerke für die gebräuchlichen Eingangsspannungen (5 V, 12 V und 24 V) werden mitgeliefert. Standardbestückung sind Widerstandsnetzwerke für 5 V.

Zur Anpassung an andere Spannungen müssen Sie den benötigten Widerstandswert berechnen und ein entsprechendes Netzwerk einsetzen. Es handelt sich immer um Single-in-line-Widerstandsnetzwerke mit je vier getrennten Widerständen entsprechend nebenstehender Abbildung (SIL-8, z.B. Bourns 4608x).

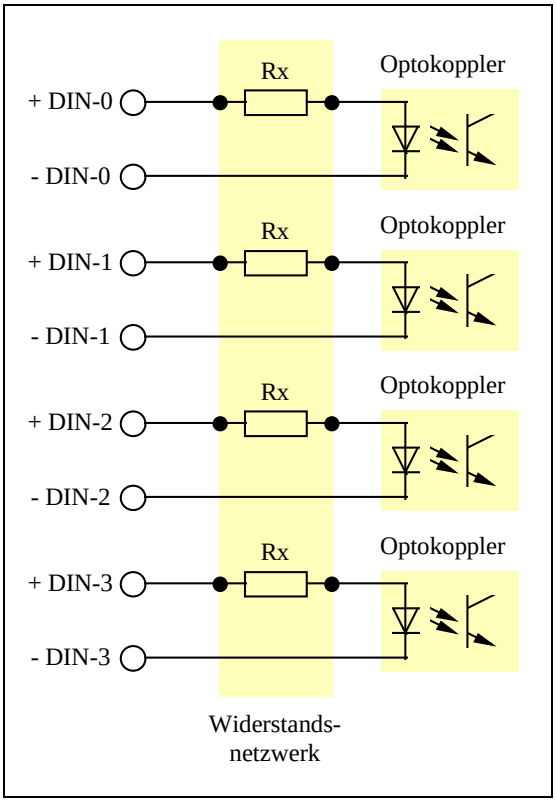


Abb. 1-5: Eingangsbeschaltung von vier benachbarten Eingängen

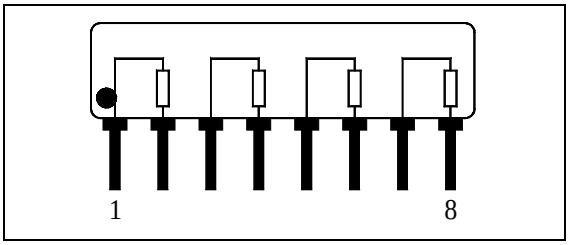


Abb. 1-6: Widerstandsnetzwerk

Die Berechnung des benötigten Widerstandswertes erfolgt mit folgender Formel:

$$R_X = \frac{U_E - 1{,}4\,V}{I_{MIN}}$$

U_E: Eingangsspannung = (Pegel an +DIN) - (Pegel an -DIN) in Volt
I_{MIN}: Stromschwelle für sicheres Erkennen einer logischen Eins in Ampere
R_X: Benötigter Widerstand in Ohm

Neben dem Strom ist auch die Verlustleistung an den Widerständen zu beachten, für die das Netzwerk ausgelegt sein muß. Die Gesamtverlustleistung berechnet sich nach folgender Formel:

$$P_V = 4\,\frac{(U_E - 1{,}4\,V)^2}{R_X}$$

U_E: Eingangsspannung = (Pegel an +DIN) - (Pegel an -DIN) in Volt
P_V: Gesamtverlustleistung des Widerstandsnetzwerks in Watt
R_X: Wert eines Widerstands des Widerstandsnetzwerkes in Ohm

Die mitgelieferten Widerstandsnetzwerke sind für einen Strom von etwa 10 mA ausgelegt. Dieser Strom genügt bei allen Modulen zum sicheren Erkennen einer logischen Eins. Die nebenstehende Tabelle zeigt, welche Widerstände zu den jeweilige Eingangsspannungen gehören. Mit zusätzlichen externen Vorwiderständen kann die Eingangsspannung bis 100 Volt betragen.

Eingangs-spannung (U_E)	Widerstands-wert (R_X)
5 V	270 Ω
12 V	1 kΩ
24 V	2,2 kΩ

Lesen von Digitaleingängen

Digitaleingänge werden in der Regel in einer Gruppe von 8 Bit gleichzeitig eingelesen. Die Bibliotheken bieten darüber hinaus Zugriff auf kleinere Einheiten (z.B. einen einzelnen Eingang). Um mehr als 8 Eingänge gleichzeitig abzutasten, enthalten die Module Zwischenspeicher (Latches), in die alle Eingangszustände zeitgleich übernommen werden können. Aus diesem Speicher können die Daten dann nacheinander ausgelesen werden. Die Übernahme der digitalen Werte in das Latch kann entweder per Software erfolgen (intern) oder von einem digitalen Eingang ausgelöst werden (extern) und wird als internes bzw. externes Triggern des Latches bezeichnet.

Interrupt-Eingänge

Diese Eingänge werden für Signale benutzt, auf die sofort reagiert werden muß (zum Beispiel wenn eine Maschine an einen Endschalter stößt oder ein Bimetallschalter anzeigt, daß eine Temperaturgrenze überschritten ist). Signale an Interrupt-Eingängen werden in der Regel direkt als Interrupts zum Prozessor weitergeleitet, der dann entsprechend darauf reagieren kann, z.B. mit dem Aufruf einer Interrupt Service Routine (ISR). Die Eingänge können so konfiguriert werden, daß sie bei positiver oder negativer Flanke einen Interrupt auslösen (z.B. mit der Bibliotheksfunktion mlxrt_set_int_edge). Wenn unter dem jeweiligen Interrupt eine DI- oder II-Task installiert ist, dann wird ihre Hauptprozedur (dies ist die ISR) aufgerufen, sobald eine entsprechende Flanke am Eingang auftritt.

Auf der MODULAR-4 Karte stehen über den SP-Bus an allen Modulen sechs Interrupt-Leitungen zur Verfügung. Je nach Modul kann je ein externer Interrupt-Eingang einer Interrupt-Leitung zugeordnet werden. Ebenfalls vom Modultyp abhängig ist, ob die Zuordnung mit Jumpern oder per Software vorgenommen wird.

Module, bei denen die Interrupt-Leitungen mit Jumpern aktiviert werden, werden in der Regel mit aufgesteckten Jumpern ausgeliefert. Die Interrupt-Eingänge sind also standardmäßig mit Interrupt-Leitungen verbunden. Achten Sie unbedingt darauf, daß es beim Einsatz mehrerer (vor allem gleichartiger) Module dadurch nicht zu Kurzschlüssen kommt. **!**

Trigger-Eingänge

Diese Eingänge werden für Signale benutzt, die auf dem Modul eine bestimmte Aktion auslösen sollen (z.B. externes Triggern eines Latches zum zeitgleichen Abtasten aller Eingänge). Die auslösende Flanke kann je nach Modul per Jumper oder per Software konfiguriert werden.

Zähleingänge

Mit diesen Eingängen werden Impulse gezählt. Zählbeginn und -ende können sowohl per Software als auch von anderen digitalen Eingängen gesteuert werden. In der Regel kann nur der Zählerstand, nicht der logische Zustand des Eingangs ermittelt werden. Um während des Zählens den Zählerstand auslesen zu können und um mehrere Zählerstände gleichzeitig zu erfassen, verfügen auch die Zähler auf den Modulen über Latches, die per Software oder durch einen digitalen Eingang getriggert werden können.

Neben der Verwendung von Hardwarezählern gibt es auch die Möglichkeit, mit geeigneten Programmen die Impulse an digitalen Eingängen oder Interrupt-Eingängen zu zählen. Allerdings ist die maximale Zählrate dabei durch die Prozessorgeschwindigkeit begrenzt (ca. 200 kHz) und die Belastung des Prozessors je nach Frequenz sehr hoch.

Inkrementalgeberinterface

Ein besonderer Typ von Zähleingängen ist das Inkrementalgeberinterface, das unter anderem zur Positionsbestimmung benutzt werden kann. Inkrementalgeber sind Sensoren, die bei Positionsänderungen Impulse abgeben. Dabei hängt es vom Typ des Gebers ab, wie viele Impulse er pro Umdrehung bzw. pro Millimeter abgibt. Um die Richtung der Positionsänderung zu bestimmen, werden zwei um 90 Grad phasenverschobene Rechtecksignale (Phase A und Phase B) geliefert, die bei konstanter Bewegung die gleiche Frequenz haben. Je nach Bewegungsrichtung eilt Phase A oder Phase B voraus (siehe Abbildung 1-7). Dadurch kann das Inkrementalgeberinterface erkennen, ob aufwärts oder abwärts gezählt werden muß. Jeder detektierte Impuls auf dem Inkrementalgeber erzeugt eine Rechteckperiode. Die Kombination beider Signale ermöglicht es dem Interface, die Auflösung des Inkrementalgebers über die Impulszahl hinaus zu erhöhen, indem es innerhalb einer Rechteckperiode zwei (Zweifach-Auswertung) oder vier (Vierfach-Auswertung) Zählimpulse erzeugt (siehe Abbildung 1-8).

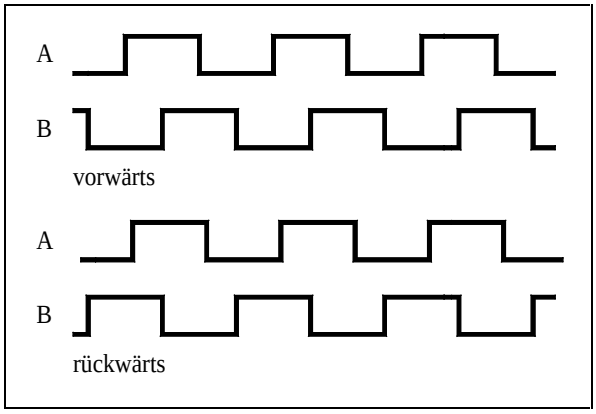


Abb. 1-7: Inkrementalgebersignale

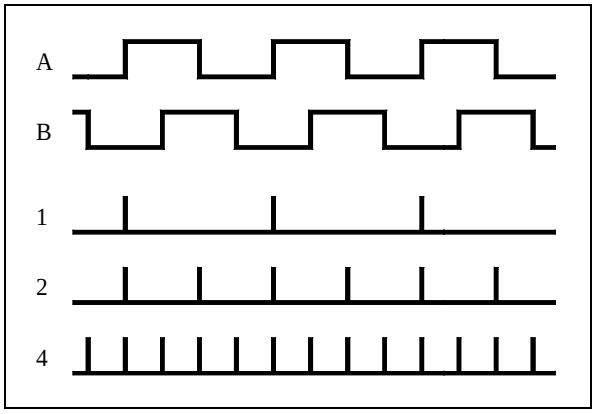


Abb. 1-8: Auswertung für Vorwärtsrichtung (ein-, zwei- oder vierfach)

Pulsbreiten-, Periodendauer- und Frequenzmessung

Durch die Kombination von Zähleingängen und zeitlich bekannten Referenzsignalen lassen sich Pulsbreiten-, Periodendauer- und Frequenzmessungen durchführen. Bei der Pulsbreitenmessung wird für die Dauer des Pulses ein von der Basiskarte vorgegebener oder ein am Modul eingespeister Referenztakt gezählt. Bei der MODULAR-4 Karte wird in der Regel der Timer-Eingangstakt verwendet, der per Software[1] auf 2,5 oder 10 MHz eingestellt werden kann. Die Periodendauermessung arbeitet nach dem gleichen Prinzip, nur daß innerhalb einer ganzen Periode gemessen wird.

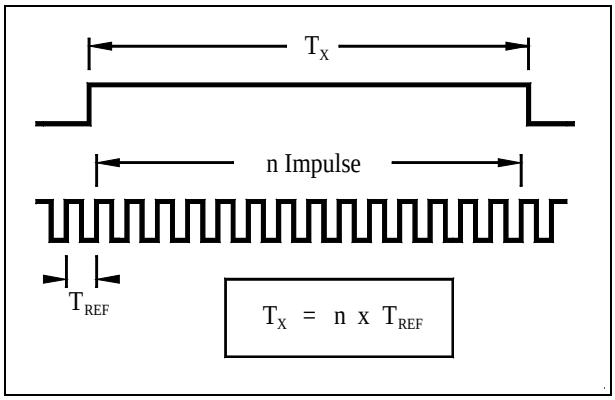


Abb. 1-9: Pulsbreitenmessung

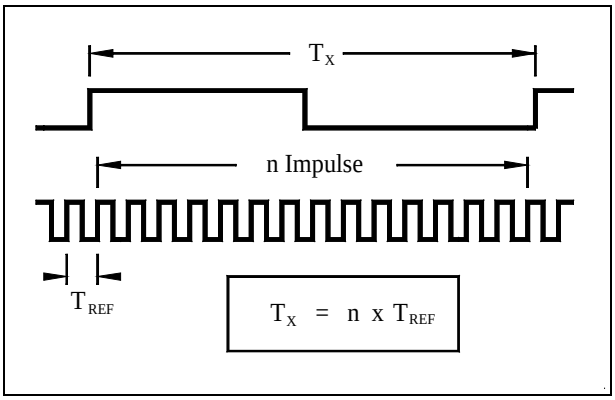


Abb. 1-10: Periodendauermessung

Bei der Frequenzmessung wird von der Basiskarte (z.B. von einem Timer) oder an einem Eingang des Moduls eine Zeit vorgegeben (Gate-Time), während der die Impulse des Eingangssignals gezählt werden. Je länger die vorgegebene Zeit ist, um so größer ist die Genauigkeit, um so länger dauert jedoch die Messung.

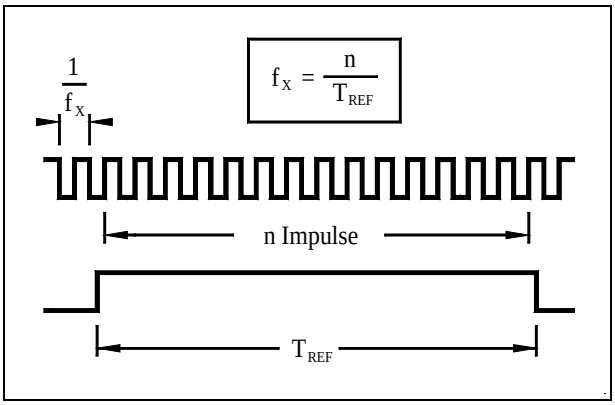


Abb. 1-11: Frequenzmessung

[1] Bei MODULAR-4/486 Karten der Revision B liegt der Takt durch die Bestückung fest. Seine Frequenz können Sie aus dem Prüfbericht entnehmen.

Digitalausgänge

Digitalausgänge gibt es im MODULAR-4 System in verschiedenen Typen: als TTL-Ausgänge, als optoentkoppelte Open-Collector-Ausgänge und als Relais.

TTL-Ausgänge

Der TTL-Standard sieht vor, daß der Ausgangspegel bei einer logischen Null maximal 0,4 Volt und bei einer logischen Eins minimal 2,4 Volt beträgt. Die genauen Werte für die einzelnen Module und ihre Abhängigkeit vom Ausgangsstrom entnehmen Sie den jeweiligen technischen Daten. TTL-Ausgänge sind nicht für hohe Ausgangsströme ausgelegt. In der Regel sind die maximalen Ströme kaum höher als 30 mA. Allerdings können TTL-Ausgänge sowohl Strom liefern (bei logisch Eins) als auch Strom aufnehmen (bei logisch Null). Abbildung 1-12 zeigt, wie TTL-Ausgänge beschaltet werden können. Im linken Bild fließt bei einer logischen Eins ein positiver Strom durch den Lastwiderstand R_L, im rechten Bild fließt bei einer logischen Null ein negativer Strom durch R_L.

Neben den logischen Pegeln Eins und Null können TTL-Ausgänge oft noch in einen dritten Zustand geschaltet werden, der als 'hochohmig' (Tri-State) bezeichnet wird. In diesem Fall verhält sich der Ausgang so, als wäre er modulseitig nicht angeschlossen (offen). Das Schalten in den hochohmigen Zustand wird auch als 'Disable' (Sperren), das Zurückschalten als 'Enable' (Aktivieren) bezeichnet.

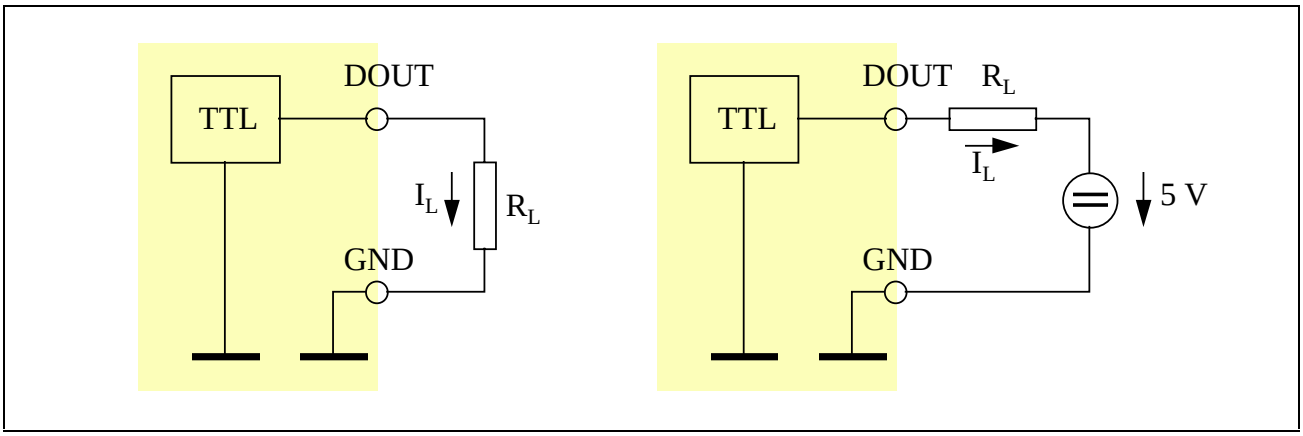


Abb. 1-12: Anschluß von TTL-Ausgängen

Optoentkoppelte Open-Collector-Ausgänge

Bei allen optoentkoppelten digitalen Ausgängen des MODULAR-4 Systems handelt es sich um Ausgänge, bei denen Collector und Emitter der Ausgangstransistoren für jeden Ausgang getrennt herausgeführt werden. Dadurch können alle Ausgänge vollständig unabhängig voneinander beschaltet werden. Der Ausgangstransistor kann idealisiert als Schalter betrachtet werden, wobei zwei Einschränkungen gelten:

1. Der Transistor leitet nur in einer Richtung Strom (siehe Pfeil an Emitter).
2. Im eingeschalteten (leitenden) Zustand ist die Ausgangsspannung nie exakt 0 Volt, sondern kann je nach Modul und Ausgangsstrom bis zu 1,4 Volt betragen (siehe technische Daten).

In der Regel gilt ein eingeschalteter (leitender) Transistor als logische Null, ein ausgeschalteter (nicht leitender) Transistor als logische Eins.

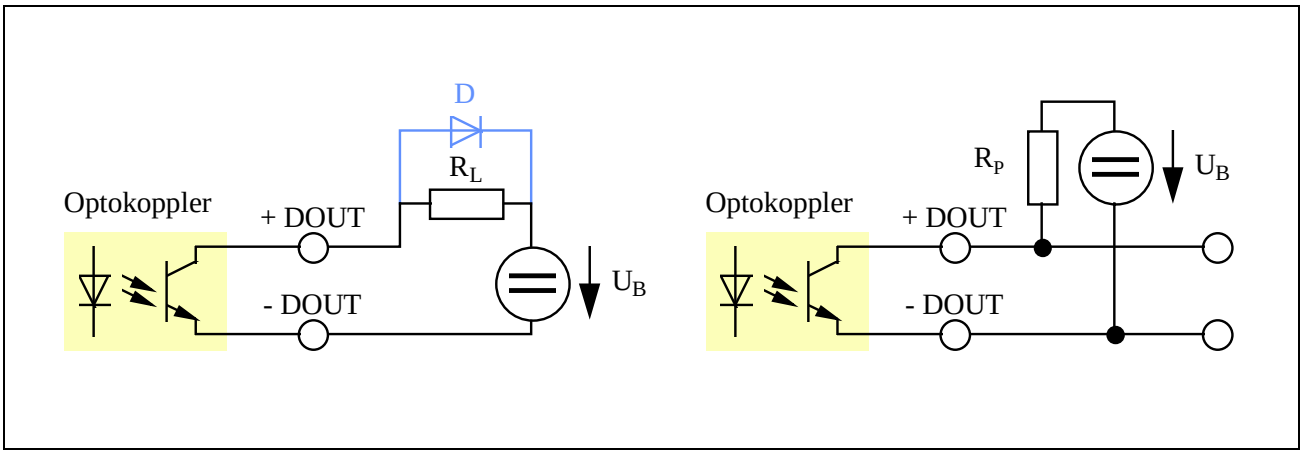


Abb. 1-13: Anschluß von optoentkoppelten Open-Collector-Ausgängen

Abbildung 1-13 zeigt den Anschluß optoentkoppelter Ausgänge. Das linke Schaltbild verdeutlicht, wie Lasten (z.B. Lampen oder Ventile) üblicherweise angeschlossen werden. Beachten Sie, daß beim Schalten von induktiven Lasten (z.B. Motoren und Relais) zusätzlich die punktiert gezeichnete Diode D zum Schutz des Ausganges notwendig ist. Die Last wird mit einer logischen Null eingeschaltet (z.B. Lampe leuchtet) und mit einer logischen Eins ausgeschaltet (z.B. Lampe aus). Beachten Sie stets, daß der maximale Ausgangsstrom nicht überschritten wird. Besonders beim Anschließen von Motoren ist zu beachten, daß der Einschaltstrom erheblich größer als der Nennstrom sein kann.

Wenn der Ausgang an einen digitalen Steuereingang eines anderen Gerätes (z.B. einer SPS) angeschlossen werden soll, kann die in Abbildung 1-13 rechts angegebene Schaltung verwendet werden. Der Ausgangspegel bei einer logischen Eins ist gleich der angelegten Spannung U_B. Bei einer logischen Null beträgt er je nach Modul zwi-

schen 0 und 1,4 Volt und ist vom Strom abhängig, der durch R_P vorgegeben ist. R_P liegt typischerweise im Bereich von 270 Ω bis 4,7 kΩ. Beachten Sie, daß die Einschaltzeit kürzer und die Flankensteilheit größer ist, je kleiner der Widerstand R_P gewählt wird.

Relais-Ausgänge

Relaisausgänge sind einfache Umschalter mit einem gemeinsamen Umschaltkontakt. Bei logisch Null befindet sich der Schalter auf dem 'Ruhekontakt', bei logisch Eins auf dem 'Arbeitskontakt'.

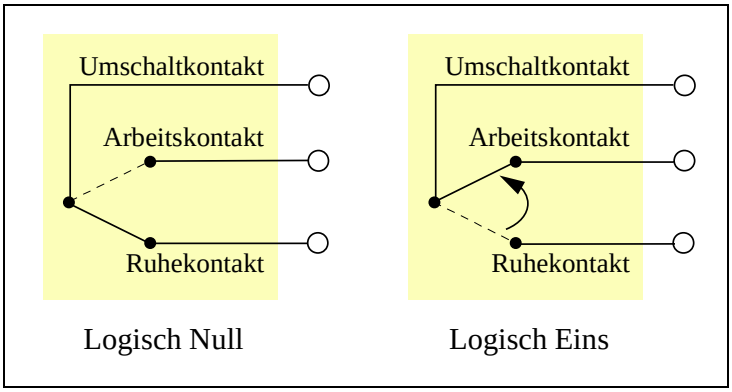


Abb. 1-14: Relaisausgang: links auf Ruhekontakt, rechts auf Arbeitskontakt

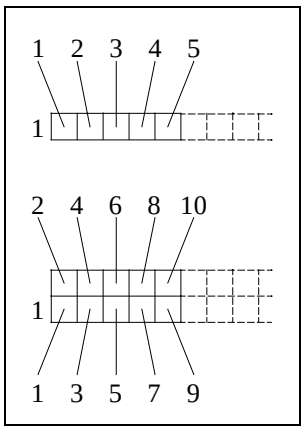
Watchdog

Digitale Ausgänge sind oft Prozeßsteuerausgänge, ihre einwandfreie Funktion ist häufig sicherheitsrelevant. Um das Sicherheitsrisiko bei einem Programmabsturz zu verringern, verfügen viele Module mit digitalen Ausgängen über einen Watchdog. Diese Einrichtung muß von der Software in regelmäßigen Intervallen angesprochen (getriggert) werden, um anzuzeigen, daß das Programm noch einwandfrei läuft. Wenn auf den Watchdog innerhalb einer festgelegten Zeit (Timeout-Zeit) nicht zugegriffen wird, schaltet er alle Ausgänge ab. Das heißt bei optoentkoppelten Ausgängen, daß alle Ausgangstransistoren stromlos (gesperrt) geschaltet werden, und bei Relais, daß alle Ausgänge auf Ruhekontakt geschaltet werden. Sie müssen eine Anlage also so auslegen, daß sie in diesem Zustand sicher ist.

Die Timeout-Zeit liegt in der Regel im Millisekundenbereich. Bei einigen Modulen ist sie fest, bei anderen ist sie einstellbar. Wenn Ihre Anwendung nicht sicherheitskritisch ist, können Sie den Watchdog auch abschalten.

Jumper

Vor dem Einbau eines Moduls, das Jumper (= Kurzschlußbrücken) enthält, sollten Sie die Einstellung aller Jumper kontrollieren und dokumentieren. Die Zählweise der einzelnen Pins in einreihigen und zweireihigen Jumperfeldern entnehmen Sie der nebenstehenden Abbildung, Pin 1 ist in den Lageplänen der Module jeweils gekennzeichnet.



Tragen Sie die Einstellung aller Jumper in das EEPROM des Moduls ein, da einige Programme (z.B. SNW) aus diesen Daten die Konfiguration des Moduls ermitteln.

EEPROM

Alle Module verfügen über einen nichtflüchtigen Speicher, ein sogenanntes EEPROM, in dem Initialisierungs-, Konfigurations- und Korrekturdaten abgelegt sind. Nach dem Laden und Starten des Betriebssystems OsX auf der Basiskarte werden die EEPROM-Inhalte immer in den Parameterbereich des Betriebssystems kopiert.

Werkseitig ist bereits eine Konfiguration im EEPROM voreingestellt.

Die Daten im EEPROM können auf zwei Arten gelesen und geschrieben werden:

1. Mit dem PC-Hilfsprogramm SNW bzw. SNW32, das im Lieferumfang jeder Basiskarte enthalten ist.
2. Aus einem PC-Programm heraus durch Aufruf der entsprechenden Bibliotheksroutinen. Die Beschreibung dieser Routinen finden Sie im Benutzerhandbuch Ihrer Basiskarte.

! *Wenn Daten des EEPROMs direkt gelesen oder geändert werden sollen, darf kein Anwendungsprogramm auf der Karte laufen (Empfehlung: vorher und hinterher sicherheitshalber einen Reset der Karte durchführen).*

Beachten Sie, daß das Eintragen von Konfigurationsdaten in das EEPROM ein Modul nicht automatisch konfiguriert. Das EEPROM speichert lediglich Werte, die per Software ausgelesen und auf dem Modul eingestellt werden müssen.

Die Bedeutung von WORT-0 ist bei allen Modulen gleich. Hier sind Modultyp und Revision eingetragen.

Das Bit-0 in WORT-1 legt bei allen Modulen fest, ob das Modul nach dem Einschalten des PCs und nach einem Hardware-Reset der MODULAR-4 Karte entsprechend den Eintragungen im EEPROM konfiguriert und initialisiert wird.

Testen der Module

SNW

Bis auf wenige Ausnahmen können alle Module mit dem Programm SNW getestet werden. Wählen Sie dazu einfach das gewünschte Modul im Menüpunkt 'Test/Module' aus oder doppelklicken Sie auf das entsprechende Modul.

Sofern ein Modul per Software konfigurierbar ist, werden beim Öffnen eines Testfensters alle Einstellungen entsprechend den EEPROM-Inhalten vorgenommen. Hierbei wird weder berücksichtigt, ob das Bit-0 in WORT-1 (Initialisierung nach Reset ein/aus) gesetzt ist, noch, ob andere Anwendungen (wie z.B. Echtzeitprogramme) das Modul gerade in einer anderen Konfiguration verwenden. Sie sollten also darauf achten, daß keine andere Anwendung das Modul benutzt, das Sie testen wollen.

Bei Modulen, die mit Jumpern konfiguriert werden, berücksichtigt SNW die EEPROM-Inhalte, die die Einstellung der Jumper widerspiegeln. Mit diesen Informationen wählt SNW das passende Meßverfahren (z.B. Spannungsbereich, Meßmodus, ...) aus. Wenn z.B. die Stellung der Jumper eines Moduls mit Analogausgängen korrekt im EEPROM eingetragen ist, können Sie in SNW direkt die Spannung oder den Strom angeben, der ausgegeben werden soll.

Unter dem Menüpunkt 'Test/Optionen' finden Sie für viele Module die Möglichkeit, die Konfiguration zu ändern. Das gilt natürlich für alle per Software vorgenommenen Einstellungen, aber auch für Jumper-Einstellungen, die nicht korrekt im EEPROM eingetragen sind. Geänderte Konfigurationen lassen sich mit 'Test/Optionen/Konfiguration speichern' im EEPROM oder in einer Datei ablegen. Bei einigen Modulen mit Analogausgängen bietet 'Test/Optionen' zusätzlich die Möglichkeit, einen Abgleich durchzuführen.

SNW32

Mit dem 32-Bit Programm SNW32 können ebenfalls bis auf wenige Ausnahmen alle Module getestet und konfiguriert werden. (Eine Beschreibung dazu finden Sie in der Programmhilfe).

Programmierung der Module

Grundsätzlich gibt es drei Möglichkeiten, um die Module anzusprechen:

- Verwenden der Modul-Device-Treiber
- Verwenden der Hochsprachenbibliotheken
- Direkte Programmierung durch Zugriffe auf die lokalen I/O-Adressen

Wir empfehlen Ihnen, bei neuen Projekten unbedingt die Modul-Device-Treiber zu verwenden, da Sie damit Hardware-unabhängig Programme erstellen können. Muß ein SPB-Modul modifiziert oder ersetzt werden, liefern wir einen kompatiblen Modul-Device-Treiber, so daß Ihre Software identisch bleibt. Nur der Modul-Device-Treiber muß ausgetauscht werden.

Die Hochsprachenbibliotheken demonstrieren lediglich, wie ein Modul angesprochen werden kann. Sie sind im Gegensatz zu den Modul-Device-Treibern nicht optimiert. Bei Änderung der Hardware, kann es sein, daß Sie Ihre Software ebenfalls ändern müssen.

Das Ansprechen der I/O-Adressen ist Spezialisten und Assembler-Programmierern vorbehalten. Auch hier ist es sehr wahrscheinlich, daß Sie Ihr Programm anpassen müssen, wenn sich die Hardware ändert.

Grundlagen zum Modul-Device-Treiber

Das MODULAR-4 System besteht aus einer Basiskarte und SP-Bus-Modulen, die auf die Basiskarte aufgesteckt werden. Je nach Basiskartentyp und Ausstattung stehen dafür bis zu 9 Steckplätze zur Aufnahme je eines Moduls zur Verfügung. Das Ansprechen der Module erfolgte bisher entweder mit Hilfe der Modulbibliotheken oder durch Zugriffe auf die lokalen I/O-Adressen der Module.

Das MODULAR-4 System basiert auf dem Multitasking-Betriebssystem OsX. Dies hat zur Konsequenz, daß mehrere Anwendungen das Modul ansprechen können. Folglich muß eine Modulbibliothek, die aus Code und Daten besteht, multitasking-fähig sein. Dies kann sie aber prinzipiell gar nicht sein, da in mehreren Anwendungen, in denen dieselbe Modulbibliothek eingebunden ist, zwar derselbe Code, aber nicht dieselben Daten eingebunden sind.

Als Beispiel sei hier das SP-Bus-Modul M-D40-2 angeführt: Die digitalen Ausgänge des Moduls können real immer nur in Gruppen zu jeweils 8 Bit gesetzt werden. Setzen zwei Anwendungen jeweils ein Bit innerhalb der gleichen Gruppe, so wird es mit sehr hoher Wahrscheinlichkeit zu Fehlern kommen, da die Bibliothek der einen Anwendung nicht weiß, auf welchen Wert die Bibliothek der anderen Anwendung die restlichen Bits der Gruppe gesetzt hat!

Wenn Sie die Modulbibliotheken nicht einsetzen und statt dessen mit Zugriffen auf die lokalen I/O-Adressen arbeiten, müssen Sie zur Sicherstellung der Multitasking-fähigkeit entsprechende Vorkehrungen in Ihrem Programm treffen. Dies ist allerdings mit relativ großem Aufwand verbunden und kann auch nur als Speziallösung angesehen werden, da bei einem Ausbau Ihrer Software das System neu überdacht werden muß!

Die Lösung der beschriebenen Problematik liegt in einem Treiberprogramm, das die Funktionseinheiten eines Moduls verwaltet und zum Ansprechen dieser Einheiten eine entsprechende Programmierschnittstelle zur Verfügung stellt. Dieses Treiberprogramm wird als **Modul-Device-Treiber (MDD)** bezeichnet.

Die Schnittstelle des Treibers muß sowohl für Zugriffe einer Task (On-Board) als auch für Zugriffe des PC verfügbar sein. Aus diesem Grunde muß der Treiber auf die Karte geladen werden. Jedes aufgesteckte SP-Bus-Modul erfordert hierbei die Installation eines zugehörigen MDD. Dieser muß nach jedem Hardware-Reset der MODULAR-4 Karte als Task auf der Karte installiert werden. Als Tasknummer wird die Nummer des zugehörigen Modulsteckplatzes verwendet.

Für die Devices der MODULAR-4 Karte muß der MDD für die Basiskarte als Task (Tasknummer 11) auf der Karte installiert werden.

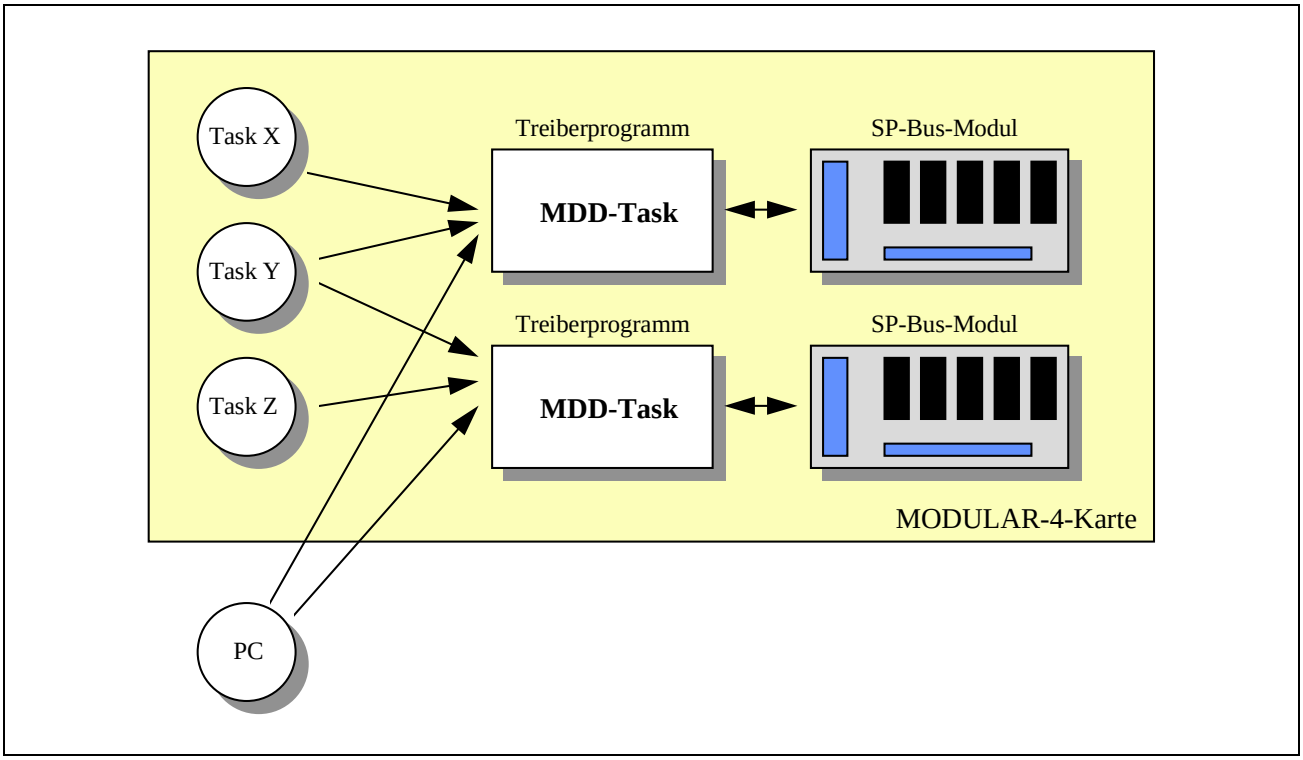


Abbildung 1-1: Für jedes Modul wird ein MDD installiert. Der Zugriff auf die Devices eines Moduls kann von verschiedenen Tasks und vom PC erfolgen.

Im folgenden wird das Prinzip der Kommunikation mit einem MDD beschrieben:

Die Verbindung zu dem Device wird als **Kanal,** der Vorgang des Verbindungsaufbaus als das **Öffnen eines Kanals** bezeichnet. Nur über einen geöffneten Kanal kann ein Device angesprochen werden. Das Öffnen eines Kanals erfolgt durch einen Aufruf der Funktion **mddx_open_channel**.[1]

Nach dem Öffnen ist ein Kanal in der Lage, Daten zu übertragen oder auch Sonderdienste auszuführen. Innerhalb eines Echtzeitprogramms muß für einen Kanalzugriff weder das Betriebssystem OsX noch der MDD herangezogen werden. Der Zugriff erfolgt über den Kanal direkt auf das entsprechende Device. Somit ist ein sehr schneller Zugriff auf das Device möglich. Bei einem PC-Programm ist dies anders: Hier teilt das PC-Programm den Kanal und die gewünschte Aktion dem Betriebssystem OsX mit, das daraufhin für den PC auf den Kanal zugreift.

Beim Beenden eines Anwenderprogrammes sollten alle bestehenden Verbindungen zu den Devices abgebaut werden. Der Abbau einer Verbindung wird als Schließen eines Kanals bezeichnet. Das Schließen eines Kanals erfolgt mittels eines Sonderdienstes mit **mddx_close_channel**.[1]

Abschließend werden nun einige wichtige Merkmale der MDDs stichpunktartig aufgelistet:

- Die Schnittstelle eines MDD wurde standardisiert und wird als **MDDI** (*Modul Device Driver Interface*) bezeichnet. Die MDDI ist damit die Anwender-Programmierschnittstelle der SP-Bus-Module (vergleichbar mit der WIN32API in Windows NT oder Windows 95).
- Aufgrund der standardisierten Schnittstelle ist es möglich, alle Devices mit einheitlichen Funktionen und somit einer einzigen Bibliothek anzusprechen. Die entsprechenden Funktionen sind in den hierfür erweiterten Bibliotheken, die bisher nur zum Programmieren der MODULAR-4 Karte verwendet wurden, enthalten**.** Die Funktionen zum Ansprechen der MDDs sind somit für alle Module gleich. Die Anzahl an Funktionen bleibt überschaubar. Dies erleichtert das Programmieren insbesondere beim Einsatz von unterschiedlichen Modultypen.
- Ein Device kann aus beliebig vielen Anwendungen angesprochen werden. Die Bibliothek ist multitaskingfähig.

---

[1] x = 7: Funktion für ″kleine″ MODULAR-4/486 Karte z.B. mdd7_open_channel
x = 8: Funktion für ″große″ MODULAR-4/486 Karte z.B. mdd8_open_channel

- MDDs sind objektorientiert aufgebaut. Interne Änderungen am MDD haben keine Auswirkungen auf bestehende Programme.
- Anwenderprogramme werden sicherer. Es besteht z.B. die Möglichkeit, einen Ausgang exklusiv für eine bestimmte Anwendung zu reservieren. Keine andere Anwendung kann dann diesen Ausgang beeinflussen.
- Falls bei einem Device Modifikationen vorgenommen werden, die das Ansprechverhalten des Device verändern, muß nur ein Update des MDD auf der MODULAR-4 Karte installiert werden. Bereits bestehende Anwenderprogramme müssen nicht verändert und somit nicht nochmals übersetzt werden. Gleiches gilt für den kompletten Ersatz eines Moduls durch einen kompatiblen Nachfolgetyp.
- Zum Erzielen einer hohen Performance sind die MDDs in Assembler geschrieben. Außerdem wird damit der kleinst mögliche Programmcode erzielt, so daß noch ausreichend freier Speicher für Anwenderprogramme und Daten auf der MODULAR-4 Karte verbleibt.
- Der Zugriff auf ein Device ist sehr schnell möglich. Der Zugriff erfolgt immer direkt, d.h. ohne Zwischenschaltung des Betriebssystems (Ausnahme: PC-Zugriff).

Treiberspezifische Begriffe

MDDI

'MDDI' bedeutet *Modul-Treiber-Schnittstelle*. Sie ist die Programmierschnittstelle für den Anwender zum Programmieren eines SPB-Moduls oder der Basiskarte.

CDT

Unter 'CDT' versteht man eine Kanal-Deskriptor-Tabelle, die vom Treiber beim Öffnen für den Kanal angelegt wird. In der CDT sind alle Dienste, Eigenschaften und Parameter eines Kanals gespeichert. Sie wird vom Treiber ausschließlich intern verwaltet. Für Zugriffe auf die Einträge in der CDT stellt der Treiber entsprechende Funktionen zur Verfügung.

Handle

Ein 'Handle' wird vom Treiber beim Öffnen eines Kanals für den Kanal vergeben und ermöglicht den Zugriff auf den geöffneten Kanal. Das Handle eines geöffneten Kanals ist dabei immer eindeutig, d.h. kein anderer Kanal kann ein Handle mit demselben numerischen Wert haben. Das Handle ist ein 32-Bit-Wert. Bei einem geöffneten Kanal ist es immer $\neq 0$.

Device

Unter den 'Devices' eines Moduls versteht man die Funktionseinheiten eines SP-Bus-Moduls oder der Basiskarte. Ein Modul besitzt in den meisten Fällen gleichartige Devices, die vom Treiber gruppiert verwaltet werden (z.B. alle digitalen Eingänge des Moduls). Zur Zeit wird zwischen folgenden Devices unterschieden:

Device	**Bedeutung**
DIN	Digitaler Eingang
DOUT	Digitaler Ausgang
DIO	Digitaler Ein- und Ausgang
AIN_SE	Analoger massebezogener Eingang
AIN_DIFF	Analoger Differenzeingang
AOUT	Analoger Ausgang
TRIG	Triggereinheit
CTRL	Steuereinheit
LED	LED
TIMER	Timer
WATCHDOG	Watchdog
COUNTER	Zähler
CLOCK	Uhr
CALENDAR	Kalender (Datum)
SCC	Serieller Controller
SQRWAVE	Rechteckgenerator
EEPROM	EEPROM
HW_INT	Hardware-Interrupt

Damit sich innerhalb einer Gruppe ein bestimmtes Device gezielt ansprechen läßt, muß dessen Kennzahl, der sogenannte **Device-Index**, angegeben werden (bestehend aus erstem und letztem Device). Dies ist auch dann erforderlich, wenn eine Device-Gruppe nur aus einem einzigen Device besteht (IndexFirst = IndexLast). Bei einem Modul mit externen Devices (z.B. 5Bx64 Panel) muß zusätzlich zur Spezifikation des externen Device der **Extern-Device-Index** angegeben werden.

Ein Device wird immer nach folgendem Schema spezifiziert:

Device - Device-Index

Beispiel: Das Modul M-D40-2 besitzt 40 digitale Eingänge/Ausgänge. Der Ein- und Ausgang mit Nummer 2 wird somit durch DIO-2 spezifiziert.

Ein externes Device wird immer nach folgendem Schema spezifiziert:

Device - Device-Index - Extern-Device-Index

Beispiel: Werden an das Modul M-D40-2 5Bx64i Panels angeschlossen, so wird jedes der bis zu 8 Panels als externes Device betrachtet. Der digitale Eingang mit Nummer 2 auf dem Panel mit Nummer 1 wird somit durch DIN-2-1 spezifiziert.

Dienste

Die verfügbaren Funktionen eines Kanals werden als die Dienste des Kanals bezeichnet. Der zum Öffnen des Kanals aufgerufene Treiber verfügt über ein umfassendes Funktionsrepertoire und stellt dem Kanal daraus geeignete Funktionen als Dienste des Kanals zur Verfügung

Wenn z.B. ein Kanal zu einem Eingang geöffnet wird, so erhält der Kanal einen **Dateneingabedienst**, der bei einem entsprechenden Kanalzugriff genau diesen Eingang liest.

Wenn z.B. ein Kanal zu einem Ausgang geöffnet wird, so erhält der Kanal einen **Datenausgabedienst**, der bei einem entsprechenden Kanalzugriff diesen Ausgang nach Ihren Wünschen setzt. Ist der Ausgang zusätzlich rücklesbar, so erhält der Kanal auch einen **Dateneingabedienst**, der beim Abruf genau diesen Ausgang zurückliest.

Unabhängig davon, ob der Kanal zu einem Ein- oder Ausgang geöffnet wird, erhält er auch einige **Sonderdienste**. Mit Hilfe dieser Sonderdienste kann der Kanal gesteuert (z.B. Schließen des Kanals), parametriert oder diagnostiziert werden.

CSR

Unter 'CSR' versteht man das Kanal-Status-Register (32-Bit) eines Kanals. Jedes Bit des Registers registriert eine aktuelle Eigenschaft des Kanals. Das Register wird durch den Kanal selbständig verwaltet.

Auf das Register kann vom Anwender nur über einen Sonderdienst mit der Funktion **mddx_get_channel_info**[1] lesend zugegriffen werden. Bei der Beschreibung dieser Funktion werden die Bit des CSR erklärt.

---

[1] x = 7: Funktion für ″kleine″ MODULAR-4/486 Karte z.B. mdd7_get_channel_info
x = 8: Funktion für ″große″ MODULAR-4/486 Karte z.B. mdd8_get_channel_info

Caption

Unter 'Caption' versteht man die frei wählbare Bezeichnung eines Kanals. Die Caption hat keinen Einfluß auf die Funktionalität des Kanals und ist lediglich für Diagnosezwecke gedacht. Es besteht z.B. die Möglichkeit, daß der PC die Captions aller zur MODULAR-4 Karte geöffneten Kanäle liest und auf dem Bildschirm darstellt. Die Caption eines Kanals besteht aus maximal 80 Zeichen (Standard: 16 Zeichen, die Änderung der Länge muß vor dem Öffnen des 1. Kanals durchgeführt werden).

Das Setzen und Lesen der Caption erfolgt über Sonderdienste des jeweiligen Kanals. Das Setzen erfolgt mit der Funktion **mddx_set_channel_caption**[1], das Lesen mit **mddx_get_channel_caption**[1].

Infotext

Unter 'Infotext' versteht man eine Zeichenkette (String), der in Klartext Informationen zu einem MDD liefert. Ein MDD kann über maximal 256 Infotextseiten, die jeweils maximal 256 Infotexte enthalten können, verfügen. Das Lesen eines Infotextes erfolgt mit der Funktion **mddx_read_infotext**[1], der die Infotextnummer (Low-Byte = Seitennummer, High-Byte = Zeilennummer) des gewünschten Infotextes übergeben werden muß.

Zur Zeit sind folgende Seiten definiert:

Seite 0 enthält Angaben zum eingesetzten Modul:

Zeile	Eintrag
0	Modulbezeichnung
1	Revision des Moduls
ab 2	Zusätzliche Informationen zum Modul

Seite 1 enthält Angaben zum Hersteller:

Zeile	Eintrag
0	Firmenbezeichnung
1	Autor
ab 2	Zusätzliche Informationen zum Hersteller

[1] x = 7: Funktion für ″kleine″ MODULAR-4/486 Karte z.B. mdd7_set_channel_caption
x = 8: Funktion für ″große″ MODULAR-4/486 Karte z.B. mdd8_set_channel_caption

Verwaltung der Devices

Zur Verwaltung der Devices ist im zugehörigen MDD für jedes verfügbare Device ein **Device-Status-Zähler** reserviert. Dieser Zähler gibt an, ob zu dem Device bereits mindestens ein Kanal geöffnet ist (Zähler > 0) oder nicht (Zähler = 0).

Beim Öffnen eines Kanals kann der MDD durch Setzen des Flags *_CP_EXCLUSIVE* dazu angewiesen werden, das Device **exklusiv** für den Kanal zu reservieren. Dazu muß der zugehörige Device-Status-Zähler = 0 sein. Ist dies der Fall, so kann der Kanal geöffnet werden, und der Zähler wird = *DSC_IS_EXCLUSIVE* gesetzt. Anderenfalls kann der Kanal nicht geöffnet werden, und der Zähler bleibt unverändert.

Wenn der Kanal nicht exklusiv geöffnet wird, so prüft der MDD, ob der entsprechende Device-Status-Zähler ≠ *DSC_IS_EXCLUSIVE* ist. Ist dies der Fall, so kann der Kanal geöffnet werden und der Zähler wird um 1 inkrementiert. Anderenfalls kann der Kanal nicht geöffnet werden, da das Device bereits exklusiv für einen anderen Kanal reserviert ist, und der Zähler bleibt unverändert.

Beim Schließen eines Kanals ist die Strategie entsprechend umgekehrt: Beim Schließen eines exklusiv geöffneten Kanals wird der entsprechende Device-Status-Zähler wieder = 0 gesetzt. Bei einem nicht exklusiv geöffneten Kanal wird der entsprechende Device-Status-Zähler um 1 dekrementiert.

Werden mehrere Devices gruppiert, so wird die oben beschriebene Strategie auf alle Devices dieser Gruppe angewandt.

Der aktuelle Stand eines Device-Status-Zählers kann mit Hilfe der Funktion **mddx_get_device_status**[1] ermittelt werden. Diese Funktion kann folgende Werte zurückliefern:

Wert	Bedeutung
= 0	Zu dem Device ist kein Kanal geöffnet.
= *DSC_IS_EXCLUSIVE*	Das Device ist exklusiv für einen einzigen Kanal reserviert.
Sonst	Der Zählerstand gibt die Anzahl Kanäle an, die zu dem Device geöffnet sind.

[1] x = 7: Funktion für ″kleine″ MODULAR-4/486 Karte z.B. mdd7_get_device_status
x = 8: Funktion für ″große″ MODULAR-4/486 Karte z.B. mdd8_get_device_status

Datentypen

Ein Modul-Device-Treiber verwendet zur Kommunikation folgende Datentypen:

Datentyp	Bedeutung
BYTE (UCHAR)	Dieser Datentyp wird zur Kommunikation mit digitalen Devices verwendet. Er findet dann Einsatz, wenn die Anzahl der Devices zwischen 1 und 8 liegt. Bit-0 steht für das erste Device, Bit-1 für das nächste, usw.
WORD (USHORT)	Dieser Datentyp wird zur Kommunikation mit digitalen Devices verwendet. Er findet dann Einsatz, wenn die Anzahl der Devices zwischen 9 und 16 liegt. Bit-0 steht für das erste Device, Bit-1 für das nächste, usw.
DWORD (ULONG)	Dieser Datentyp wird zur Kommunikation mit digitalen Devices verwendet. Er findet dann Einsatz, wenn die Anzahl der Devices zwischen 17 und 32 liegt. Bit-0 steht für das erste Device, Bit-1 für das nächste, usw.
SHORT	Dieser Datentyp wird zur Kommunikation mit einem analogen Device verwendet. Der Zahlenwert wird immer im 2er-Komplement angegeben:

Auflösung		Unipolar pos.	Unipolar neg.	Bipolar
12 Bit	PV	4095	-	2047
	NP	0	0	0
	NV	-	-4095	-2048
16 Bit	PV	32767	-	32767
	NP	0	0	0
	NV	-	-32768	-32768

PV=Positiver Vollausschlag, NP=Nullpunkt, NV=Negativer Vollausschlag

BLOCK

Daten werden als Block übertragen, wenn die Größe der obigen Datentypen nicht mehr ausreicht.

- Bei einem digitalen Kanal ist dies der Fall, wenn mit *n* (> 32) Devices kommuniziert wird. In diesem Fall besteht der Block aus einer bestimmten Anzahl Datenbyte, die sich aus folgender Formel ergibt:

  Anzahl = (*n* DIV 8) + 1

- Bei einem analogen Kanal ist dies der Fall, wenn mit *n* (> 1) Devices kommuniziert wird. In diesem Fall besteht der Block aus einer bestimmten Anzahl Datenbyte (*n* SHORTs), die sich aus folgender Formal ergibt:

  Anzahl = $n \times 2$

VOID

Dieser Datentyp wird verwendet, wenn bei der Kommunikation mit einem Device keine Datenübertragung notwendig ist (z.B. Trigger-Kanal).

Kanaleigenschaftsstrukturen (CPS)

Eine CPS ist eine Datenstruktur, die alle erforderlichen Informationen enthält, die der Modul-Device-Treiber zum Öffnen eines Kanals benötigt. Für jeden MDD ist eine spezielle CPS definiert (siehe Dokumentation zum jeweiligen MDD). Alle definierten Kanaleigenschaftsstrukturen bestehen ausschließlich aus den im folgenden beschriebenen Strukturelementen.

Je nach Kanaltyp kann es vorkommen, daß nicht alle in der CPS enthaltenen Strukturelemente parametriert werden müssen. In der jeweiligen Dokumentation zum MDD ist für jeden Kanal angegeben, welche Strukturelemente parametriert werden müssen und auf welche Werte die Elemente gesetzt werden dürfen. Die restlichen Elemente werden nicht ausgewertet.

.usDevice (Word)
.usIndexFirst (Word)
.usIndexLast (Word)
.usIndexExtern (Word)

Diese Strukturelemente spezifizieren das Device (bzw. die Device-Gruppierung) zu dem bzw. zu der der Kanal geöffnet werden soll. Für das Strukturelement **.usDevice** können folgende Konstanten gesetzt werden:

Device	Bedeutung
DEVICE_DIN	Digitaler Eingang
DEVICE_DOUT	Digitaler Ausgang
DEVICE_DIO	Digitaler Ein- und Ausgang
DEVICE_AIN_SE	Analoger massebezogener Eingang
DEVICE_AIN_DIFF	Analoger Differenzeingang
DEVICE_AOUT	Analoger Ausgang
DEVICE_TRIG	Triggereinheit
DEVICE_CTRL	Steuereinheit
DEVICE_LED	LED
DEVICE_TIMER	Timer
DEVICE_WATCHDOG	Watchdog
DEVICE_COUNTER	Zähler
DEVICE_CLOCK	Uhr
DEVICE_CALENDAR	Kalender (Datum)
DEVICE_SCC	Serieller Controller
DEVICE_SQRWAVE	Rechteckgenerator
DEVICE_EEPROM	EEPROM
DEVICE_HW_INT	Hardware-Interrupt

Hinweise dazu, welche Werte für die Strukturelemente **.usIndexFirst**, **.usIndexLast** und ggf. **.usIndexExtern** zu setzen sind, finden Sie in der Dokumentation zum entsprechenden MDD.

.usFlags (Word)

Gibt sonstige Eigenschaften des Kanals an. Sollen keine sonstigen Eigenschaften angegeben werden, so muß **.usFlags** = 0 gesetzt werden. Anderenfalls können dem Kanal durch bitweises OR-Verknüpfen der folgenden Flags bestimmte Eigenschaften zugeordnet werden:

Flag	Bedeutung
_CP_EXCLUSIVE	Der Kanal soll exklusiv geöffnet werden.
_CP_HW_FORMAT	Die Daten sollen in dem Format übertragen werden, das die Hardware liefert bzw. benötigt. Diese Option sollte nur für Testzwecke verwendet werden, da die Bedeutung der Zahlenwerte sich hinsichtlich der Angaben im Abschnitt 'Datentypen' unterscheiden können.
_CP_UNCORRECTED	Dieses Flag kann bei analogen Kanälen gesetzt werden und bewirkt, daß die vom Device ermittelten Werte nicht einer Gain/Offset-Korrektur unterzogen werden. Diese Option sollte nur für Test- oder Abgleichzwecke verwendet werden.

.usRange (Word)

Gibt den physikalischen I/O-Bereich an. Der Bereich wird mit einer der folgenden Konstanten angegeben:

I/O Bereich	Unipoloar positiv	Unipolar negativ	Bipolar
312,5 mV	*RANGE_UPP_312MV5*	*RANGE_UPN_312MV5*	*RANGE_BIP_312MV5*
625 mV	*RANGE_UPP_625MV*	*RANGE_UPN_625MV*	*RANGE_BIP_625MV*
1,25 V	*RANGE_UPP_1V25*	*RANGE_UPN_1V25*	*RANGE_BIP_1V25*
2,5 V	*RANGE_UPP_2V5*	*RANGE_UPN_2V5*	*RANGE_BIP_2V5*
5 V	*RANGE_UPP_5V*	*RANGE_UPN_5V*	*RANGE_BIP_5V*
10 V	*RANGE_UPP_10V*	*RANGE_UPN_10V*	*RANGE_BIP_10V*
20 mA	*RANGE_20MA*		

.usOverSampling (Word)

Gibt den Faktor an, wie oft ein analoger Eingang abgetastet wird. Als Ergebnis wird der arithmetische Mittelwert der ermittelten Meßwerte übergeben. Der Faktor kann nur in 2er-Potenzen angegeben werden. Gültige Werte sind somit 1 (= kein Oversampling), 2, 4, 8, 16, usw.

.usReadMode (Word)

Gibt den Lesemodus an. Zur Zeit sind folgende Modes möglich:

Mode	Bedeutung
IO_MODE_DIRECT	Es wird direkt vom Eingang gelesen.
IO_MODE_LATCH	Es wird aus dem Eingangspuffer (Hardware) des Moduls gelesen.
IO_MODE_RAM	Es wird aus dem RAM gelesen. Dieser Mode wird dazu verwendet, um Daten, die in den Kanal geschrieben wurden und im Treiber gepuffert sind, zurückzulesen.
IO_MODE_RAM_LATCH	Es wird aus dem RAM gelesen. Zum Generieren der Daten im RAM muß per Software ein Trigger ausgelöst werden. Dazu ist zusätzlich ein entsprechender Trigger-Kanal zu öffnen.

.usWriteMode (Word)

Gibt den Schreibmodus an. Zur Zeit sind folgende Modes möglich:

Mode	Bedeutung
IO_MODE_DIRECT	Es wird direkt in den Ausgang geschrieben.
IO_MODE_LATCH	Es wird in den Ausgangspuffer (Hardware) des Moduls geschrieben. Damit geschriebene Daten am Ausgang effektiv werden, muß ein Trigger ausgelöst werden.
IO_MODE_RAM_LATCH	Es wird in das RAM geschrieben. Damit geschriebene Daten am Ausgang effektiv werden, muß per Software ein Trigger ausgelöst werden. Dazu ist zusätzlich ein entsprechender Trigger-Kanal zu öffnen.

.ulSettleTime (Dword)

Gibt die Settle-Time (Einschwingzeit) als Vielfaches von 1 µs an.

.usMode (Word)

Gibt einen speziellen Mode für den Kanal an (siehe Dokumentation zum jeweiligen MDD).

Hinweise zur Verwendung der MDDs

- Der MDD muß auf der Karte installiert werden.
- Damit ein MDD fehlerfrei funktioniert, müssen alle Devices ausschließlich über den MDD angeprochen werden. Weder darf die Modulbibliothek des Moduls verwendet werden, noch darf von außen auf die lokalen I/O Adressen des Moduls zugegriffen werden.
- Die Tasknummern 0 bis 20 sind für Systemprogramme (Betriebssystem und MDDs) reserviert und dürfen nicht für Anwenderprogramme verwendet werden.
- Die maximale Länge für die Caption eines Kanals ist nach der Installation eines MDD auf 16 voreingestellt. Dieser Wert kann bei Bedarf durch Setzen des Task-Parameters (BYTE-Zugriff auf Parameter 1) auf Werte von 0 bis 80 eingestellt werden. Dies muß vor dem Öffnen des 1. Kanals geschehen.
- MDDs sind nicht mehrfach installierbar. Für jedes Modul muß ein MDD installiert werden.
- Die Bezeichnung von Datentypen, Funktionen, usw. enthalten oft ein „x“, dieses „x“ kann durch 7 (bei Verwendung einer „kleinen“ MODULAR-4/486) oder durch 8 (bei Verwendung einer „großen“ MODULAR-4/486) ersetzt werden, z.B. wird **mddx_open_channel** dann zu **mdd7_open_channel** bzw. **mdd8_open_channel** und **HMDDx** zu **HMDD7** bzw. **HMDD8**.

Modul-Device-Treiber Bibliothek

Dienste eines MDD

mddx_open_channel PC, RT

Funktion **mddx_open_channel** öffnet einen Kanal.

Die gewünschten Eigenschaften des zu öffnenden Kanals werden durch die Parameter der übergebenen **Kanaleigenschaftsstruktur (CPS)** spezifiziert. In den Bibliotheken ist für jeden MDD eine CPS definiert. Hinweise zum Parametrieren der CPS entnehmen Sie der Dokumentation zum jeweiligen MDD.

Die übergebenen Parameter der CPS werden vom MDD geprüft. Diese Prüfung bezieht sich sowohl auf die Plausibilität der gewünschten Kanaleigenschaften als auch auf die Verfügbarkeit des angegebenen Device.

Im fehlerfreien Fall erzeugt der MDD für den geöffneten Kanal ein Handle, das als Rückgabewert dieser Funktion übergeben wird. Dieses Handle muß bei allen Funktionen, die später auf diesen Kanal zugreifen, angegeben werden. Es muß deshalb in einer Variablen vom Typ HMDDx gespeichert werden.

Ein Handle ist immer eindeutig: Auch wenn **mddx_open_channel** mehrmals nacheinander mit denselben Parametern aufgerufen wird, so liefert die Funktion immer ein Handle mit einem anderen numerischen Wert.

Hinweis: Während der Zeit, in der ein Kanal geöffnet wird, werden auftretende Interrupts auf der Karte nicht bearbeitet. Es ist deshalb empfehlenswert, alle in Ihrer Anwendung benötigten Kanäle vorab beim Initialisieren Ihres Programms zu öffnen.

PASCAL FUNCTION mddx_open_channel
(usMDD, usStrucSize: USHORT; VAR pCPS): HMDDx;

C HMDDx mddx_open_channel
(USHORT usMDD, USHORT usStrucSize, void* pCPS);

Parameter

usMDD Tasknummer des MDD

usStrucSize Größe der CPS (in Byte)

pCPS Zeigt auf die Kanaleigenschaftsstruktur zur Spezifikation des gewünschten Kanals

Rückgabe Handle des geöffneten Kanals

Beispiel

```
/* MODULE M-D40-2 ON SLOT 1 */
#define SLOT_MD402 1

/* RESERVE CPS FOR MODULE M-D40-2 */
CPS_MD402 rcMD402;

/* RESERVE HANDLE FOR CHANNEL MACHINE-1 */
HMDDx hMachine1;

/* SPECIFY THE CHANNEL
     - OPEN TO DIO-0..DIO-7
     - OPEN CHANNEL EXCLUSIVE
     - WRITE DIRECT TO OUTPUT
     - READBACK DIRECT FROM OUTPUT */
rcMD402.usDevice       = DEVICE_DIO;
rcMD402.usIndexFirst   = 0;
rcMD402.usIndexLast    = 7;
rcMD402.usFlags        = _CP_EXCLUSIVE;
rcMD402.usReadMode     = IO_MODE_DIRECT;
rcMD402.usWriteMode    = IO_MODE_DIRECT;

/* OPEN THE CHANNEL AND STORE THE HANDLE
   IN hMachine1 */
hMachine1 = mddx_open_channel(
                 SLOT_MD402,
                 sizeof(rcMD402),
                 &rcMD402);
```

mddx_read_infotext PC, RT

Funktion **mddx_read_infotext** liest einen Infotext des angegebenen MDD.

Die verfügbaren Infotexte sind im MDD auf einer Seite (0 bis 255) und in einer Zeile (0 bis 255) gespeichert (siehe Dokumentation zum MDD). Der gewünschte Infotext wird durch die übergebene Infotextnummer (Low-Byte = Seite, High-Byte = Zeile) spezifiziert. Zur Zeit sind folgende Infotextnummern vordefiniert:

Infotextnummer	Bedeutung
ITNO_DEVICE_NAME	Bezeichnung des zugehörigen Device (Moduls oder Basiskarte)
ITNO_DEVICE_REVISION	Revision des zugehörigen Device (Moduls oder Basiskarte)
ITNO_FIRM_NAME	Herstellerfirma
ITNO_AUTHOR_NAME	Autor

Die Infotexte sind im MDD als nullterminierte Strings gespeichert. Die Umwandlung in das Stringformat des verwendeten Compilers erfolgt - falls erforderlich - in der Bibliothek.

PASCAL

```
PROCEDURE mddx_read_infotext
(usMDD, usTxtNr: USHORT;
ucMaxLen: UCHAR; VAR pacInfotext: STRING);
```

C

```
void mddx_read_infotext
(USHORT usMDD, USHORT usTxtNr,
UCHAR ucMaxLen, CHAR* pacInfotext);
```

Parameter

usMDD Tasknummer des MDD

usTxtNr Infotextnummer (siehe Dokumentation zum MDD)

ucMaxLen Maximal zulässige Länge des Infotextes + 1

pacInfotext Zeigt auf Variable zum Empfang des Infotextes (String)

Rückgabe Keine

mddx_get_device_status PC, RT

Funktion **mddx_get_device_status** ermittelt den Status eines Device.

Hinweise zur Spezifikation des gewünschten Device entnehmen Sie der Dokumentation zum jeweiligen MDD.

PASCAL
```
FUNCTION mddx_get_device_status
(usMDD, usDevice, usIndex, usExtIndex: USHORT): USHORT;
```

C
```
USHORT mddx_get_device_status
(USHORT usMDD, USHORT usDevice,
USHORT usIndex, USHORT usExtIndex);
```

Parameter

usMDD Tasknummer des MDD

usDevice Device-Typ (wie z.B. *DEVICE_DIN* für DIN-0)

usIndex Device-Index (z.B 0 für DIN-0)

usExtIndex Extern-Device-Index (wird nur dann ausgewertet, wenn Extern-Devices verfügbar sind, z.B 1 für DIN-0-1)

Rückgabe Siehe folgende Tabelle

Wert	**Bedeutung**
0 bis EFFFh	Anzahl der zu dem Device geöffneten Kanäle
FFF0h (= *DSC_SHARED_MAX*)	Anzahl der zu dem Device geöffneten Kanäle, maximale zulässige Anzahl damit erreicht
FFF1h (= *DSC_IS_NOT_AVAILABLE*)	Device ist nicht verfügbar, da Modul anders konfiguriert wurde
FFF2h (= *DSC_IS_LOCKED*)	Device ist (vorübergehend) gesperrt
FFFFh (= *DSC_IS_EXCLUSIVE*)	Device ist exklusiv mit einem einzigen Kanal verknüpft

mddx_scan_channel PC, RT

Funktion **mddx_scan_channel** ermöglicht die Diagnose eines Kanals.

Die Funktion ist besonders für Testzwecke und für Supervisor-Programme interessant. Die gewünschten Informationen des Kanals werden durch den Parameter *wInfo* bestimmt:

wInfo	**Information (Rückgabewert der Funktion)**
SCAN_CDT	Die Funktion liest den Header der Kanal-Deskriptor-Tabelle (CDT) des in *hChannel* angegebenen Kanals zurück. Die CDT (Details und Struktur dieser Tabelle finden Sie im Anhang) wird beim Öffnen des Kanals im freien Speicher der MODULAR-4 angelegt. Die Parameter der CDT werden im folgenden Format übergeben:
	Offset **Datentyp** **Bedeutung** 0 UCHAR Byte-0 der CDT 1 UCHAR Byte-1 der CDT : 59 UCHAR Byte-59 der CDT
	Hinweis: Es muß der gesamte Header gelesen werden. Fordern Sie deshalb *CDT_HEADER_SIZE* Datenbyte an.
SCAN _PARAMETER	Die Funktion liest die beim Öffnen des Kanals vom MDD errechneten Kanalparameter zurück. Diese Parameter sind nur für MDD-Entwickler interessant und werden im folgenden Format übergeben:
	Offset **Datentyp** **Bedeutung** 0 UCHAR Byte-0 der Kanalparameter : n-1 UCHAR Byte-(n-1) der Kanalparameter
	Hinweis: Die Größe der zu lesenden Datenstruktur muß mit der tatsächlichen Größe, die aus dem CDT-Header (mit *wInfo* = *SCAN_CDT*) ermittelt werden kann, übereinstimmen.
SCAN_CPS	Die Funktion liest die beim Öffnen des Kanals übergebene Kanaleigenschaftsstruktur (CPS) zurück. In *hChannel* muß das Handle des gewünschten Kanals angegeben werden. Die CPS des Kanals wird im folgenden Format übergeben:
	Offset **Datentyp** **Bedeutung** 0 USHORT Wort-0 der CPS : 2×n USHORT Wort-n (letztes Wort) der CPS
	Hinweis: Verwenden Sie zur Übernahme der CPS eine Variable desselben CPS-Typs, der auch zum Öffnen des Kanals verwendet wurde. Die angeforderte Anzahl Byte muß mit der Größe dieser CPS übereinstimmen.

SCAN_SERVICE _LIST

Die Funktion liest eine Auflistung der verfügbaren Dienste des in *hChannel* angegebenen Kanals zurück. Die zurückgelesene Liste enthält die Anzahl und die Kennungen der verfügbaren Dienste des Kanals und wird im folgenden Format übergeben:

Offset	**Datentyp**	**Bedeutung**
0	USHORT	Anzahl der verfügbaren Dienste
2	USHORT	Kennung des Datenausgabedienstes (0 = Dienst nicht verfügbar)
4	USHORT	Kennung des Dateneingabedienstes (0 = Dienst nicht verfügbar)
6	USHORT	Kennung des ersten verfügbaren Sonderdienstes
:		
2× (n+2)	USHORT	Kennung des n-ten verfügbaren Sonderdienstes

Hinweise: Die Kennung des Datenausgabe- und Dateneingabedienstes ist in der Liste auch dann enthalten, wenn der jeweilige Dienst zur Zeit gesperrt ist (siehe auch **mddx_send_channel_command**). Es findet keine Prüfung statt, ob die Liste mehr oder weniger Daten als angefordert enthält. Beachten Sie deshalb den Rückgabewert der Funktion.

SCAN_HANDLE

Die Funktion ermittelt die Handles der zum Zeitpunkt des Aufrufes vom MDD geöffneten Kanäle. Die Handles werden vom MDD in chronologischer Reihenfolge verwaltet: Beim Öffnen eines Kanals erhält der neue Kanal stets die Ordnungszahl 1, die Ordnungszahlen der bereits vom MDD geöffneten Kanäle erhöhen sich entsprechend um 1. Der Parameter *hChannel* bestimmt, ab welchem Kanal die Handles ermittelt werden sollen. Wird in *hChannel* das Handle eines vom MDD geöffneten Kanals übergeben, so beginnt die Liste mit dem Handle des Kanals mit der nächst höheren Ordnungsnummer. Ist *hChannel* = 0, so beginnt die Liste mit dem 'neuesten' vom MDD geöffneten Kanal.

Pro Handle müssen 4 Byte angefordert werden. Die ermittelten Handles werden im folgenden Format (Annahme: k ist die Ordnungszahl des in *hChannel* angegebenen Kanals, wenn *hChannel* = 0 ist, dann ist k = 0) übergeben:

Offset	Datentyp	Bedeutung
0	HMDDx	Handle des Kanals mit Ordnungszahl k+1
4	HMDDx	Handle des Kanals mit Ordnungszahl k+2
:		
4×(n-1)	HMDDx	Handle des Kanals mit Ordnungszahl k+n

Hinweise: Die Anzahl der von einem MDD geöffneten Kanäle kann mit **mlx_read_par_word(usMDD, 2)** ermittelt werden.

PASCAL

```
FUNCTION mddx_scan_channel
(usMDD, usInfo: USHORT; hChannel: HMDDx;
usInfoSize: USHORT; VAR pInfoData): USHORT;
```

C

```
USHORT mddx_scan_channel
(USHORT usMDD,  USHORT usInfo, HMDDx hChannel, USHORT
usInfoSize, void* pInfoData);
```

Parameter

usMDD Tasknummer des MDD

usInfo Spezifiziert die gewünschten Informationen (siehe oben)

hChannel Handle oder laufende Nummer des Kanals (siehe oben)

usInfoSize Anzahl angeforderter Informationen (in Byte)

pInfoData Zeigt auf Speicherbereich zum Empfang der Informationen

Rückgabe Anzahl der gültigen Informationen (in Byte)

mddx_get_version PC, RT

Funktion **mddx_get_version** ermittelt die Version und das Erstellungsdatum des angegebenen MDD.

Die Version wird als Dword in folgendem Format übergeben (zum Erzeugen einer Versionsangabe in Klartext kann der Wert der Funktion **mlx(rt)_create_version_string** übergeben werden:

Bitnummer	Bedeutung
0 bis 7	Versionsstatus
8 bis 15	Laufende Nummer (0 bis 99)
16 bis 23	Revisionsbuchstabe (z.B. 'A')
24 bis 31	Versionsnummer (z.B. 1)

Das Erstellungsdatum wird als Dword in folgendem Format übergeben (zum Erzeugen einer Datumangabe in Klartext kann der Wert der Funktion **mlx(rt)_create_date_string** übergeben werden):

Bitnummer	Bedeutung
0 bis 7	Wochentag (0 = *SUNDAY*, 1 = *MONDAY*, ..., 6 = *SATURDAY*)
8 bis 15	Tag (1 bis 31)
16 bis 23	Monat (1 bis 12)
24 bis 31	Jahr (z.B. 97)

PASCAL

```
PROCEDURE mddx_get_version
(usMDD: USHORT; VAR pulVersion: ULONG;
VAR pulDate: ULONG);
```

C

```
void mddx_get_version
(USHORT usMDD, ULONG* pulVersion,
ULONG* pulDate);
```

Parameter

usMDD Tasknummer des MDD

pulVersion Zeigt auf eine Variable (ULONG) zum Empfang der Version

pulDate Zeigt auf eine Variable (ULONG) zum Empfang des Erstellungsdatums

Rückgabe Keine

mddx_reset_driver PC, RT

Funktion **mddx_reset_driver** setzt den angegebenen MDD zurück.

Beim Zurücksetzen eines MDD werden alle Kanäle, die von ihm geöffnet wurden, geschlossen und die dazugehörigen Devices wie bei der Installation des MDD neu konfiguriert.

Hinweis: Die Handles der Kanäle, die durch das Zurücksetzen geschlossen werden, werden ungültig!

PASCAL PROCEDURE mddx_reset_driver (usMDD: USHORT);

C void mddx_reset_driver (USHORT usMDD);

Parameter *usMDD* Tasknummer des MDD

Rückgabe Keine

Dienste eines Kanals

Datenausgabedienst (Daten schreiben)

mddx_write_channel_byte PC, RT

Funktion **mddx_write_channel_byte** schreibt ein Byte in den angegebenen Kanal.

Der Zugriff ist nur dann möglich, wenn der Kanal den Datentyp Byte (UCHAR) unterstützt (siehe Dokumentation zum MDD oder **mddx_get_channel_info**).

PASCAL
```
PROCEDURE mddx_write_channel_byte
(hChannel: HMDDx; ucData: UCHAR);
```

C
```
void mddx_write_channel_byte
(HMDDx hChannel, UCHAR ucData);
```

Parameter *hChannel* Handle des Kanals

ucData Zu schreibender Wert (Byte)

Rückgabe Keine

mddx_write_channel_word PC, RT

Funktion **mddx_write_channel_word** schreibt ein Wort in den angegebenen Kanal.

Der Zugriff ist nur dann möglich, wenn der Kanal den Datentyp Wort (USHORT) unterstützt (siehe Dokumentation zum MDD oder **mddx_get_channel_info**).

PASCAL
```
PROCEDURE mddx_write_channel_word
(hChannel: HMDDx; usData: USHORT);
```

C
```
void mddx_write_channel_word
(HMDDx hChannel, USHORT usData);
```

Parameter	*hChannel*	Handle des Kanals
	usData	Zu schreibender Wert (Wort)
Rückgabe	Keine	

mddx_write_channel_dword PC, RT

Funktion **mddx_write_channel_dword** schreibt ein Doppelwort in den angegebenen Kanal.

Der Zugriff ist nur dann möglich, wenn der Kanal den Datentyp Doppelwort (ULONG) unterstützt (siehe Dokumentation zum MDD oder **mddx_get_channel_info**).

PASCAL
```
PROCEDURE mddx_write_channel_dword
(hChannel: HMDDx; ulData: ULONG);
```

C
```
void mddx_write_channel_dword
(HMDDx hChannel, ULONG ulData);
```

Parameter	*hChannel*	Handle des Kanals
	ulData	Zu schreibender Wert (Doppelwort)
Rückgabe	Keine	

mddx_write_channel_short PC, RT

Funktion **mddx_write_channel_short** schreibt ein Wort mit Vorzeichen in den angegebenen Kanal.

Der Zugriff ist nur dann möglich, wenn der Kanal den Datentyp Wort mit Vorzeichen (SHORT) unterstützt (siehe Dokumentation zum MDD oder **mddx_get_channel_info**).

PASCAL
```
PROCEDURE mddx_write_channel_short
(hChannel: HMDDx; sData: SHORT);
```

C
```
void mddx_write_channel_short
(HMDDx hChannel, SHORT sData);
```

Parameter

hChannel	Handle des Kanals
sData	Zu schreibender Wert (Wort mit Vorzeichen)

Rückgabe Keine

mddx_write_channel_block PC, RT

Funktion **mddx_write_channel_block** schreibt einen Datenblock in den angegebenen Kanal.

Der Zugriff ist nur dann möglich, wenn der Kanal den Blocktransfer unterstützt (siehe Dokumentation zum MDD oder **mddx_get_channel_info**).

Die Größe des Datenblocks, die der Kanal übernehmen kann, wird durch die vom Kanal verwalteten Kanalparameter MINSIZE und MAXSIZE (siehe **mddx_get_channel_info**) bestimmt. Ist die Größe des zu schreibenden Datenblockes kleiner als MINSIZE oder größer als MAXSIZE, so übernimmt der Kanal keine Daten und erzeugt eine entsprechende Fehlermeldung.

PASCAL
```
FUNCTION mddx_write_channel_block
(hChannel: HMDDx; usSize: USHORT; VAR pData): USHORT;
```

C
```
USHORT mddx_write_channel_block
(HMDDx hChannel, USHORT usSize, void* pData);
```

Parameter	*hChannel*	Handle des Kanals
	usSize	Größe des zu schreibenden Datenblockes (in Byte)
	pData	Zeigt auf den zu schreibenden Datenblock
Rückgabe	Anzahl der vom Kanal nicht verwerteten Daten (in Byte)	

mddx_trigger_channel PC, RT

Funktion **mddx_trigger_channel** löst über den angegebenen Kanal (z.B. Watchdog-Kanal) per Software einen Trigger aus.

Der Zugriff ist nur dann möglich, wenn der Kanal triggerbar ist (siehe Dokumentation zum MDD oder **mddx_get_channel_info**).

PASCAL PROCEDURE mddx_trigger_channel (hChannel: HMDDx);

C void mddx_trigger_channel (HMDDx hChannel);

Parameter	*hChannel*	Handle des Kanals
Rückgabe	Keine	

Dateneingabedienst (Daten lesen)

mddx_read_channel_byte PC, RT

Funktion **mddx_read_channel_byte** liest ein Byte aus dem angegebenen Kanal.

Der Zugriff ist nur dann möglich, wenn der Kanal den Datentyp Byte (UCHAR) unterstützt (siehe Dokumentation zum MDD oder **mddx_get_channel_info**).

PASCAL FUNCTION mddx_read_channel_byte
(hChannel: HMDDx): UCHAR;

C UCHAR mddx_read_channel_byte (HMDDx hChannel);

Parameter *hChannel* Handle des Kanals

Rückgabe Gelesener Wert (Byte)

mddx_read_channel_word PC, RT

Funktion **mddx_read_channel_word** liest ein Word aus dem angegebenen Kanal.

Der Zugriff ist nur dann möglich, wenn der Kanal den Datentyp Word (USHORT) unterstützt (siehe Dokumentation zum MDD oder **mddx_get_channel_info**).

PASCAL FUNCTION mddx_read_channel_word
(hChannel: HMDDx): USHORT;

C USHORT mddx_read_channel_word (HMDDx hChannel);

Parameter *hChannel* Handle des Kanals

Rückgabe Gelesener Wert (Wort)

mddx_read_channel_dword PC, RT

Funktion **mddx_read_channel_dword** liest ein Doppelwort aus dem angegebenen Kanal.

Der Zugriff ist nur dann möglich, wenn der Kanal den Datentyp Doppelwort (ULONG) unterstützt (siehe Dokumentation zum MDD oder **mddx_get_channel_info**).

PASCAL FUNCTION mddx_read_channel_dword
(hChannel: HMDDx): ULONG;

C ULONG mddx_read_channel_dword (HMDDx hChannel);

Parameter *hChannel* Handle des Kanals

Rückgabe Gelesener Wert (Doppelwort)

mddx_read_channel_short PC, RT

Funktion **mddx_read_channel_short** liest ein Wort mit Vorzeichen aus dem angegebenen Kanal.

Der Zugriff ist nur dann möglich, wenn der Kanal den Datentyp Wort mit Vorzeichen (SHORT) unterstützt (siehe Dokumentation zum MDD oder **mddx_get_channel_info**).

PASCAL FUNCTION mddx_read_channel_short
(hChannel: HMDDx): SHORT;

C SHORT mddx_read_channel_short (HMDDx hChannel);

Parameter *hChannel* Handle des Kanals

Rückgabe Gelesener Wert (Wort mit Vorzeichen)

mddx_read_channel_block PC, RT

Funktion **mddx_read_channel_block** liest einen Datenblock aus dem angegebenen Kanal.

Der Zugriff ist nur dann möglich, wenn der Kanal den Blocktransfer unterstützt (siehe Dokumentation zum MDD oder **mddx_get_channel_info**).

Die Größe des Datenblocks, die der Kanal zur Verfügung stellen kann, wird durch die vom Kanal verwalteten Kanalparameter MINSIZE und MAXSIZE (siehe **mddx_get_channel_info**) bestimmt. Ist die Größe des zu lesenden Datenblockes kleiner als MINSIZE oder größer als MAXSIZE, so übergibt der Kanal keine Daten und erzeugt eine entsprechende Fehlermeldung.

PASCAL
```
FUNCTION mddx_read_channel_block
(hChannel: HMDDx; usSize: USHORT; VAR pData): USHORT;
```

C
```
USHORT mddx_read_channel_block
(HMDDx hChannel, USHORT usSize, void* pData);
```

Parameter

hChannel Handle des Kanals

usSize Größe des zu lesenden Datenblockes (in Byte)

pData Zeigt auf den zum Empfang des zu lesenden Datenblockes reservierten Speicherbereich

Rückgabe Anzahl der vom Kanal übergebenen Daten (in Byte)

Sonderdienste

mddx_send_channel_command PC, RT

Funktion **mddx_send_channel_command** sendet ein Steuerkommando über den angegebenen Kanal.

Zur Zeit sind folgende Steuerkommandos definiert:

usCmd	**Bedeutung**
CMD_RESET	Steuerkommando **RESET** (siehe Dokumentation zum jeweiligen MDD)
CMD_START	Steuerkommando **START** (siehe Dokumentation zum jeweiligen MDD)
CMD_STOP	Steuerkommando **STOP** (siehe Dokumentation zum jeweiligen MDD)
CMD_CONTINUE	Steuerkommando **CONTINUE** (siehe Dokumentation zum jeweiligen MDD)
CMD_LATCH_OUT	Die im Zwischenspeicher des verknüpften Device eingetragenen Werte werden am Ausgang effektiv. Weitere Details entnehmen Sie der Dokumentation zum jeweiligen MDD.
CMD_LATCH_IN	Die am Eingang des verknüpften Device anliegenden Werte werden in den Zwischenspeicher des Device übernommen. Weitere Details entnehmen Sie der Dokumentation zum jeweiligen MDD.
CMD_SUSP_WRITE	Schreibzugriffe auf Kanal vorübergehend sperren
CMD_CONT_WRITE	Schreibzugriffe auf Kanal wieder ermöglichen
CMD_SUSP_READ	Lesezugriffe auf Kanal vorübergehend sperren
CMD_CONT_READ	Lesezugriffe auf Kanal wieder ermöglichen

PASCAL
```
PROCEDURE mddx_send_channel_command
(hChannel: HMDDx; usCmd: USHORT);
```

C
```
void mddx_send_channel_command
(HMDDx hChannel, USHORT usCmd);
```

Parameter *hChannel* Handle des Kanals

usCmd Steuerkommando

mddx_send_channel_control PC, RT

Funktion **mddx_send_channel_control** sendet ein Steuerkommando und einen Steuerparameter über den angegebenen Kanal.

Zur Zeit sind folgende Steuerkommandos definiert:

usCtrl	**Bedeutung**
CTRL_DEVICE	Steuert das mit dem Kanal verknüpfte Device gemäß dem übergebenen Steuerparameter. Detaillierte Informationen zu den möglichen Steuerparametern entnehmen Sie der Dokumentation zum jeweiligen MDD.
CTRL_RW_PTR	Setzt den Lese-Schreib-Zeiger des Kanals auf die im Steuerparameter übergebene relative Adresse. Der Lese-Schreib-Zeiger des Kanals wird bei jedem Lese- oder Schreibzugriff auf den Kanal automatisch um die entsprechende Anzahl Byte erhöht. Die aktuelle Position des Lese-Schreib-Zeigers kann mit **mddx_get_channel_info** ermittelt werden.
CTRL_R_PTR	Setzt den Lesezeiger des Kanals auf die im Steuerparameter übergebene relative Adresse. Der Lesezeiger des Kanals wird bei jedem Lesezugriff auf den Kanal automatisch um die entsprechende Anzahl Byte erhöht. Die aktuelle Position des Lesezeigers kann mit **mddx_get_channel_info** ermittelt werden.
CTRL_W_PTR	Setzt den Schreibzeiger des Kanals auf die im Steuerparameter übergebene relative Adresse. Der Schreibzeiger des Kanals wird bei jedem Schreibzugriff auf den Kanal automatisch um die entsprechende Anzahl Byte erhöht. Die aktuelle Position des Schreibzeigers kann mit **mddx_get_channel_info** ermittelt werden.

PASCAL
```
PROCEDURE mddx_send_channel_control
(hChannel: HMDDx; usCtrl: USHORT; ulCtrlPar: ULONG);
```

C
```
void mddx_send_channel_control
(HMDDx hChannel, USHORT usCtrl, ULONG ulCtrlPar);
```

Parameter

hChannel Handle des Kanals

usCtrl Steuerkommando

ulCtrlPar Steuerparameter

mddx_send_channel_control_block PC, RT

Funktion **mddx_send_channel_control_block** sendet ein Steuerkommando und einen Steuerparameterblock über den angegebenen Kanal.

Diese Funktion wird zur Zeit nicht benötigt.

PASCAL
```
FUNCTION mddx_send_channel_control_block
(hChannel: HMDDx; usCtrl, usSize: USHORT;
VAR pCtrlData): USHORT;
```

C
```
USHORT mddx_send_channel_control_block
(HMDDx hChannel, USHORT usCtrl, USHORT usSize,
void* pCtrlData);
```

Parameter

hChannel	Handle des Kanals
usCtrl	Steuerkommando
usSize	Größe des Steuerparameterblocks
pCtrlData	Zeigt auf den zu sendenden Steuerparameterblock

Rückgabe Anzahl der vom Kanal nicht verwerteten Daten (in Byte)

mddx_get_channel_info PC, RT

Funktion **mddx_get_channel_info** ermittelt einen Kanalparameter des angegebenen Kanals.

Der gewünschte Kanalparameter wird durch *usInfo* ausgewählt:

usInfo	**Ermittelter Kanalparameter (Rückgabe der Funktion)**
INFO _STATUS	Status des Kanals, bestehend aus folgenden Flags:

Konstante	**Bit**	**Bedeutung**
_CSR_EXCLUSIVE	#0	Kanal ist exklusiv geöffnet
_CSR_HW_FORMAT	#1	Daten werden unformatiert übertragen
_CSR_UNCORRECTED	#2	Daten werden unkorrigiert übertragen
_CSR_OUT_SUSP	#16	Schreibzugriff gesperrt
_CSR_IN_SUSP	#17	Lesezugriff gesperrt

INFO _DATA_TYPE	Datentyp, der vom Kanal unterstützt wird:

Konstante	**Wert**	**Bedeutung**
DATA_VOID	01h	Kanal unterstützt TRIGGER
DATA_BYTE	03h	Kanal unterstützt UCHAR
DATA_WORD	05h	Kanal unterstützt USHORT
DATA_DWORD	07h	Kanal unterstützt ULONG
DATA_SHORT	06h	Kanal unterstützt SHORT
DATA_BLOCK	09h	Kanal unterstützt BLOCK

INFO _TRANSFER _MODE	Transfermode, der vom Kanal unterstützt wird:

Konstante	**Wert**	**Bedeutung**
R_MODE	01h	Nur Lesen möglich
W_MODE	02h	Nur Schreiben möglich
RW_MODE	03h	Lesen und Schreiben möglich

INFO _MINSIZE	Minimale Anzahl Datenbyte, die der Kanal bei einem Schreibzugriff erwartet bzw. bei einem Lesezugriff zur Verfügung stellt
INFO _MAXSIZE	Maximale Anzahl Datenbyte, die der Kanal bei einem Schreibzugriff übernehmen bzw. bei einem Lesezugriff zur Verfügung stellen kann
INFO _RESOLUTION	Bei analogem Kanal Auflösung eines Analogwertes in Bit
INFO_RW_PTR	Aktuelle Position des Lese-Schreib-Zeigers des Kanals
INFO_R_PTR	Aktuelle Position des Lesezeigers des Kanals
INFO_W_PTR	Aktuelle Position des Schreibzeigers des Kanals

INFO _DEVICE	Statusinformation des mit dem Kanal verknüpften Device. Detaillierte Hinweise zur Bedeutung der Statusinformation entnehmen Sie der Dokumentation zum jeweiligen MDD.		
INFO_RANGE	Physikalischer I/O-Bereich des Kanals		
	Konstante	**Wert**	**Bedeutung**
	RANGE_UPP_312MV5	21h	Bereich 0 bis +312,5 mV
	RANGE_UPP_625MV	22h	Bereich 0 bis +625 mV
	RANGE_UPP_1V25	23h	Bereich 0 bis +1,25 V
	RANGE_UPP_2V5	24h	Bereich 0 bis +2,5 V
	RANGE_UPP_5V	25h	Bereich 0 bis +5 V
	RANGE_UPP_10V	26h	Bereich 0 bis +10 V
	RANGE_UPN_312MV5	41h	Bereich 0 bis -312,5 mV
	RANGE_UPN_625MV	42h	Bereich 0 bis -625 mV
	RANGE_UPN_1V25	43h	Bereich 0 bis -1,25 V
	RANGE_UPN_2V5	44h	Bereich 0 bis -2,5 V
	RANGE_UPN_5V	45h	Bereich 0 bis -5 V
	RANGE_UPN_10V	46h	Bereich 0 bis -10 V
	RANGE_BIP_312MV5	61h	Bereich ±312,5 mV
	RANGE_BIP_625MV	62h	Bereich ±625 mV
	RANGE_BIP_1V25	63h	Bereich ±1,25 V
	RANGE_BIP_2V5	64h	Bereich ±2,5 V
	RANGE_BIP_5V	65h	Bereich ±5 V
	RANGE_BIP_10V	66h	Bereich ±10 V
	RANGE_20MA	81h	Strombereich 0 bis 20 mA

PASCAL
```
FUNCTION mddx_get_channel_info
(hChannel: HMDDx; usInfo: USHORT): ULONG;
```

C
```
ULONG mddx_get_channel_info
(HMDDx hChannel, USHORT usInfo);
```

Parameter *hChannel* Handle des Kanals

usInfo Gewünschter Kanalparameter (siehe Tabelle oben)

Rückgabe Ermittelter Kanalparameter (siehe Tabelle oben)

mddx_get_channel_info_block PC, RT

Funktion **mddx_get_channel_info_block** ermittelt einen Kanalparameterblock des angegebenen Kanals.

Diese Funktion wird zur Zeit nicht benötigt.

PASCAL
```
FUNCTION mddx_get_channel_info_block
(hChannel: HMDDx; usInfo, usSize: USHORT;
VAR pInfoData): USHORT;
```

C
```
USHORT mddx_send_channel_control_block
(HMDDx hChannel, USHORT usInfo, USHORT usSize,
void* pInfoData);
```

Parameter

hChannel Handle des Kanals

usInfo Gewünschter Kanalparameterblock (siehe Tabelle oben)

usSize Maximal zulässige Größe des Kanalparameterblocks (bei String mindestens Stringlänge + 1)

pInfoData Zeigt auf den zum Empfang des zu lesenden Kanalparameterblockes reservierten Speicherbereich

Rückgabe Anzahl der vom Kanal übergebenen Daten (in Byte)

mddx_set_channel_caption PC, RT

Funktion **mddx_set_channel_caption** trägt eine Kurzbeschreibung als String in den angegebenen Kanal ein.

Beim Öffnen des Kanals wird für den Kanal ein Zeichenpuffer reserviert. Die Puffergröße entnimmt der öffnende MDD seinem Parameterbereich (Byte, relativer Offset 1, Voreinstellung = 16, maximaler Wert = 80).

PASCAL
```
PROCEDURE mddx_set_channel_caption
(hChannel: HMDDx; usLen: USHORT; pcCaption: STRING);
```

C
```
void mddx_set_channel_caption
(HMDDx hChannel, USHORT usLen, CHAR* pcCaption);
```

Parameter *hChannel* Handle des Kanals

usLen Länge des zu setzenden Strings

pcCaption Zeigt auf den zu setzenden String

mddx_get_channel_caption PC, RT

Funktion **mddx_get_channel_caption** liest die Kurzbeschreibung des angegebenen Kanals.

PASCAL PROCEDURE mddx_get_channel_caption
(hChannel: HMDDx; usLen: USHORT; VAR pcCaption: STRING);

C void mddx_get_channel_caption
(HMDDx hChannel, USHORT usLen, CHAR* pcCaption);

Parameter *hChannel* Handle des Kanals

usLen Maximale Länge des zu lesenden Strings

pcCaption Zeigt auf Variable zum Empfang des Strings

mddx_close_channel PC, RT

Funktion **mddx_close_channel** schließt den angegebenen Kanal.

Im fehlerfreien Fall wird der Kanal beim verknüpften Device abgemeldet und das Handle = 0 gesetzt.

Hinweis: Beim Öffnen des Kanals wurde für den Kanal eine bestimmter Speicherbereich (ca. 100 Byte) auf der MODULAR-4 reserviert. Dieser Speicherbereich kann zur Zeit mit dem Schließen des Kanals nicht wieder freigegeben werden!

PASCAL PROCEDURE mddx_close_channel (VAR phChannel: HMDDx);

C void mddx_close_channel (HMDDx* phChannel);

Parameter *phChannel* Zeigt auf das Handle des zu schließenden Kanals

mddx_check_handle_valid RT

Funktion **mddx_check_handle_valid** prüft die Gültigkeit des übergebenen Handles.

PASCAL FUNCTION mddx_check_handle_valid (hChannel: HMDDx): WORD;

C USHORT mddx_check_handle_valid (HMDDx hChannel);

Parameter *hChannel* Zu prüfendes Handle

Rückgabe > 0: Handle ist gültig, der Rückgabewert gibt die Tasknummer des MDD an, der den Kanal geöffnet hat

= 0: Handle ist ungültig

Fehlermeldungen

Falls beim Zugriff auf einen MDD oder einen Kanal ein Fehler auftritt, so wird die Fehlerbehandlungsroutine der Bibliothek aufgerufen. In die Strukturen vom Typ **mlx_error_info_type** (PC) bzw. **mlxrt_error_info_type** (Echtzeit) werden folgende Parameter eingetragen:

Bei einem **PC-Programm** wird das Strukturelement **.errorclass = 11** gesetzt. In **.errorcode** sind folgende Fehlercodes möglich:

Fehlercode	Bedeutung
ERR_MDD_HANDLE_INVALID = 40h	Angegebenes Handle ungültig
ERR_MDD_WRONG_DATA_TYPE = 41h	Falscher Zugriff: Kanal erwartet anderen Datentyp
ERR_MDD_DRV_SERVICE = 42h	Fehler in MDD-Dienst, Diagnose möglich (siehe unten)
ERR_MDD_CHANNEL_SERVICE = 43h	Fehler in Kanal-Dienst, Diagnose möglich (siehe unten)
ERR_MDD_SERVICE_NOT _AVAILABLE = 44h	Dienst nicht verfügbar
ERR_MDD_DRV_NOT_INSTALLED = 45h	Treiber nicht korrekt installiert
ERR_MDD_UNKNOWN_ERROR = 46h	Unbekannter MDD-Fehler

Bei einem **Echtzeit-Programm** wird das Strukturelement **.subnum = 1nnnh** gesetzt (nnn = Tasknummer des zugehörigen MDD; 1000h = keinem MDD zuordnenbar). In **.errorcode** sind folgende Fehlercodes möglich:

Fehlercode	Bedeutung
ERR_MDD_HANDLE_INVALID = 40E0h	Angegebenes Handle ungültig
ERR_MDD_WRONG_DATA_TYPE = 41E0h	Falscher Zugriff: Kanal erwartet anderen Datentyp
ERR_MDD_DRV_SERVICE = 42E0h	Fehler in MDD-Dienst, Diagnose möglich (siehe unten)
ERR_MDD_CHANNEL_SERVICE = 43E0h	Fehler in Kanal-Dienst, Diagnose möglich (siehe unten)
ERR_MDD_SERVICE_NOT _AVAILABLE = 44E0h	Dienst nicht verfügbar
ERR_MDD_DRV_NOT_INSTALLED = 45h	Treiber nicht korrekt installiert

Diagnosemeldungen

Falls der Fehler *ERR_MDD_DRV_SERVICE* oder *ERR_MDD_CHANNEL_SERVICE* auftritt, so kann durch Aufruf der Funktion **mddx_get_diagnosis** (keine Übergabeparameter) eine zusätzliche Diagnosemeldung angefordert werden. Die Funktion sollte nur innerhalb der Fehlerbehandlungsroutine aufgerufen werden.

Innerhalb der Fehlerbehandlungsroutine eines PC-Programms müssen vor dem Aufruf von **mddx_get_diagnosis** alle Fehlerinformationen mit **mlx_clear_error** gelöscht werden, da sonst kein Kartenzugriff zum Abholen der Diagnosemeldung möglich ist!

Hochsprachenbibliotheken

Verwendung der Bibliotheken

Die Hochsprachenbibliotheken für die Module sind geeignet für die Basiskarten MODULAR-4/486, MODULAR-4/Z80 und MODULAR-4/Z280. Die MODULAR-4/Z80 und /Z280 Karten werden in dieser Beschreibung nicht behandelt. Bei der MODULAR-4/486 Karte können die Bibliotheken sowohl für den PC (unter **DOS** und **Windows 3.x**) als auch für Echtzeitprogramme, die lokal auf der Karte laufen, benutzt werden.

Die Bibliotheken sind für den Einsatz der Hochsprachen **Borland Pascal (Version 7.0), Turbo Pascal (Version 6.0 und 7.0)** und **Borland C++ (Version 3.1)** geeignet. Getestet wurden sie mit Borland Pascal Version 7.0 und Borland C++ Version 3.1. Die Windows-DLLs können natürlich auch mit anderen Programmiersprachen (z.B. Visual Basic) verwendet werden. Sie sind vorwiegend dazu gedacht, das Ansprechen der Module zu demonstrieren, und sind nicht bezüglich ihrer Geschwindigkeit optimiert.

Es wird keine Überprüfung der Übergabeparameter vorgenommen.

Allgemeine Hinweise:

1. Der Parameter **micro_slot** wird in allen Routinen verwendet und gibt den Modulsteckplatz an. Er kann bei der MODULAR-4/486 Karte die Werte 1 bis 4 annehmen, bei der MODULAR-4/486 Karte mit Modulextender die Werte 2 bis 10.

2. Vor der Verwendung von Bibliotheksroutinen vom **PC** aus müssen folgende Routinen einmal aufgerufen werden:

   ml8_bib_startup und
   ml8_reset oder **ml8_start**

Für **Echtzeitprogramme**, die auf der MODULAR-4/486 Karte laufen sollen, ist kein solcher Aufruf erforderlich.

3. Bei vielen Modulbibliotheken muß außerdem eine eigene Initialisierungsroutine aufgerufen werden. Ob solch ein Aufruf nötig ist und welche Funktion das ist, entnehmen Sie jeweils dem Kapitel 'Hochsprachenbibliotheken' der einzelnen Modulbeschreibungen.

4. Die Angabe **EXPORT** bei der Beschreibung der Routinen für C bedeutet:

bei DOS-Programmen:	far pascal
bei Windows-Programmen (DLL):	_export far pascal

! *Wenn ein Modul aus verschiedenen Tasks oder vom PC und von einer Task angesprochen wird, kann es zu Problemen kommen.Insbesondere bei Digitalausgängen ist das Setzen einzelner Bits in der Regel nicht möglich. Sehr kritisch ist auch die Verwendung der Bibliotheken in Tasks, die sich gegenseitig unterbrechen können. Dazu gehört auch die Programmierung vom PC aus, da der PC auf der Karte laufende Tasks unterbrechen kann.*
Das Problem läßt sich lösen, in dem man alle Modulzugriffe in eine zentrale Steuertask verlegt. Alle anderen Tasks und der PC benutzen dann keine Modulbibliotheks-Befehle mehr, sondern rufen Prozeduren und Funktionen der Steuertask auf, um das Modul anzusprechen (siehe auch vorigen Abschnitt).

Einbinden der Bibliotheken

Im folgenden wird beschrieben, wie Sie die Bibliotheken in Ihre Programme einbinden können. Dabei werden zwei Bezeichner benutzt, die als Platzhalter für modulspezifische Namen stehen. Die richtigen Namen für die einzelnen Module entnehmen Sie jeweils dem Kapitel 'Hochsprachenbibliotheken'.

MODULE: Angabe des Unterverzeichnisses, in dem sich alle Programme und Bibliotheken zum jeweiligen Modul befinden.

libname: Name der Bibliothek. Dieser Name wird für alle Dateien, die zur Bibliothek gehören, verwendet. Er setzt sich aus der Modulkennummer (dreistellige Zahl) und der Bezeichnung '_LIB' zusammen (z.B. libname = 'M039_LIB' für M-AD12-16)

MODULAR-4/486: Pascal-Programme unter DOS

Hierfür werden folgende Dateien mitgeliefert:

Unterverzeichnis	Dateiname	Kommentar
MODULE\PASCAL	*libname*.PAS	Source-Datei PASCAL
MODULE\ML8\DOS\PASCAL	*libname*.OBJ	.OBJ-Datei
MODULE\ML8\DOS\PASCAL	*libname*.TP	Konfigurationsdatei

Verwenden Sie die Datei *libname*.PAS und geben Sie in der USES-Anweisung ML8BIB und *libname* an.

MODULAR-4/486: C-Programme unter DOS

Hierfür werden folgende Dateien mitgeliefert:

Unterverzeichnis	Dateiname	Kommentar
MODULE\C	*libname*.C	Source-Datei in C
MODULE\C	*libname*.H	Header-Datei für C
MODULE\ML8\DOS\C	*libname*.PRJ	Projektdatei für C

Binden Sie die Include-Dateien ML8BIB.H und *libname*.H in Ihr C-Programm ein (#include "Dateiname"). Linken Sie ML8BIB.LIB und *libname*.OBJ zu Ihrem C-Programm dazu. Das Define "ML8" muß gesetzt sein.

MODULAR-4/486: Pascal-Programme unter Windows (DLL)

Hierfür werden folgende Dateien mitgeliefert:

Unterverzeichnis	Dateiname	Kommentar
MODULE\PASCAL	*libname*.PAS	Source-Datei PASCAL
MODULE\ML8\WINDOWS\PASCAL	*libname*.DLL	.DLL-Datei
MODULE\ML8\WINDOWS\PASCAL	*libname*.LIB	.LIB-Datei
MODULE\ML8\WINDOWS\PASCAL	*libname*.DEF	.DEF-Datei
MODULE\ML8\WINDOWS\PASCAL	*libname*.TPW	.TPW-Datei
MODULE\ML8\WINDOWS\PASCAL	*libname*.TP	Konfigurationsdatei

Geben Sie in der USES-Anweisung WML8BIB und *libname* an. Die Dateien WML8BIB.DLL, WML8BIB.TPW, *libname*.DLL und *libname*.TPW müssen im Default-Verzeichnis sein. Die Import-Unit *libname*.TPW wurde aus der Datei *libname*.PAS erzeugt.

MODULAR-4/486: C-Programme unter Windows (DLL)

Hierfür werden folgende Dateien mitgeliefert:

Unterverzeichnis	Dateiname	Kommentar
MODULE\C	*libname*.C	Source-Datei in C
MODULE\C	*libname*.H	Header-Datei für C
MODULE\ML8\WINDOWS\C	*libname*.PRJ	Projektdatei
MODULE\ML8\WINDOWS\C	*libname*.DLL	.DLL-Datei
MODULE\ML8\WINDOWS\C	*libname*.LIB	.LIB-Datei
MODULE\ML8\WINDOWS\C	*libname*.DEF	.DEF-Datei

Binden Sie die Include-Dateien ML8BIB.H und *libname*.H in Ihr C-Programm ein (#include "Dateiname"). Linken Sie WML8BIB.LIB und *libname*.LIB zu Ihrem C-Programm dazu. Die Dateien WML8BIB.DLL und *libname*.DLL müssen zur Laufzeit im Default-Verzeichnis sein.

MODULAR-4/486: Pascal-Programme auf der Karte

Hierfür werden folgende Dateien mitgeliefert:

Unterverzeichnis	Dateiname	Kommentar
MODULE\PASCAL	*libname*.PAS	Source-Datei in PASCAL
MODULE\ML8\RT\PASCAL	*libname*.OBJ	.OBJ-Datei für PASCAL
MODULE\ML8\RT\PASCAL	*libname*.TP	Konfigurationsdatei

Verwenden Sie die Datei *libname*.PAS und geben Sie in der USES-Anweisung ML8RTBIB und *libname* an.

MODULAR-4/486: C-Programme auf der Karte

Hierfür werden folgende Dateien mitgeliefert:

Unterverzeichnis	Dateiname	Kommentar
MODULE\C	*libname*.C	Source-Datei in C
MODULE\C	*libname*.H	Header-Datei für C
MODULE\ML8\RT\C	*libname*.PRJ	Projektdatei für C

Binden Sie die Include-Dateien ML8RTBIB.H und *libname*.H in Ihr C-Programm ein (#include "Dateiname"). Linken Sie ML8RTBIB.OBJ und *libname*.OBJ zu Ihrem C-Programm dazu. Die Defines "ML8" und "ML8RT" müssen gesetzt sein.

Weitere Hinweise zu den Hochsprachenbibliotheken finden Sie im Benutzerhandbuch zur MODULAR-4 Karte in den Kapiteln "PC-Hochsprachenbibliotheken" und "Echtzeitbibliotheken".

Programmierung mit I/O-Zugriffen

Wenn für Ihr PC-Betriebssystem oder für Ihre Programmiersprache keine Modulbibliotheken vorliegen oder wenn Sie geschwindigkeitsoptimierte Echtzeitprogramme schreiben wollen, müssen Sie die Funktionseinheiten der Module direkt programmieren. Das geschieht mit I/O-Zugriffen des Prozessors auf das jeweilige Modul.

Pro Modul stehen maximal 32 I/O-Adressen zur Verfügung. Die anzusprechende I/O-Adresse ergibt sich jeweils als Summe aus der steckplatzabhängigen Modul-Basis-Adresse (MBA, siehe folgende Tabelle) und einem Adreß-Offset.

Steckplatz	1	2	3	4	5	6	7	8	9	10
MBA	400h	500h	600h	700h	420h	440h	460h	480h	4a0h	4c0h

Die I/O-Adressen und ihre Funktion sind bei den Modulen jeweils im Kapitel 'I/O-Adressen' zu finden. Dort ist auch angegeben, ob auf die jeweilige Adresse mit 8- oder mit 16-Bit-Befehlen zugegriffen werden darf. In den Tabellen gelten folgende Abkürzungen:

R: nur Lesezugriffe erlaubt
W: nur Schreibzugriffe erlaubt
R/W: Schreib- und Lesezugriffe erlaubt
8: nur 8-Bit-Zugriffe erlaubt
16: nur 16-Bit-Zugriffe erlaubt
x: die gelesenen bzw. geschriebenen Daten sind ohne Bedeutung