MODULAR-4®/486

SORCUS

Alle Angaben in diesem Handbuch sind ohne Gewähr und können ohne weitere Benachrichtigung geändert werden. Da sich trotz aller Bemühungen Fehler nie vollständig ausschließen lassen, sind wir für Hinweise dankbar. Eventuelle Erweiterungen und Korrekturen finden Sie auf der mitgelieferten CD bzw. im Internet unter **www.sorcus.com**.

9. neu bearbeitete und erweiterte Auflage
01.08. 2001
SORCUS Computer GmbH
Im Breitspiel 11
69126 Heidelberg

Inhaltsverzeichnis

3. Funktionseinheiten der MODULAR-4 3-1

4. Das Karten-Manager-Programm SNW32 4-1

5. Software 5-1

8. Remote-Debugging 8-1

9. Echtzeit-Bibliotheken 9-1

Anhang

Notizen

1. Das MODULAR-4 System

MODULAR-4 ist ein modulares Meßdatenerfassungs-, Steuerungs- und Kommunikationssystem auf der Basis einer intelligenten PC-Zusatzkarte.

Das System zeichnet sich durch eine hohe Flexibilität in der Anpassung an neue Aufgaben aus:

> Die MODULAR-4/486 Karte ist hardwaremäßig ein kompletter, unabhängiger 486-Computer auf einer PC-Zusatzkarte.

Durch die freie Programmierbarkeit ist die MODULAR-4 Karte prinzipiell nicht auf bestimmte Aufgaben festgelegt. Da sie außerdem auch unabhängig vom PC arbeiten kann, ergibt sich die Möglichkeit einer echten Parallelverarbeitung der beiden CPUs. Das bedeutet natürlich auch eine sehr hohe Verarbeitungsgeschwindigkeit und z.B. die Möglichkeit zur Vorverarbeitung und Vorauswertung von Meßdaten oder der kompletten Auslagerung von Kommunikationsprotokollen, ohne daß der PC eingreifen muß. Ein sehr leistungsfähiges Echtzeit-Multi-Tasking Betriebssystem auf der Karte unterstützt und erleichtert die Programmierung erheblich.

Die Leistungsfähigkeit eines PCs kann durch Einstecken mehrerer MODULAR-4 Karten um ein Vielfaches gesteigert werden: Der PC wird so zu einem Multi-Prozessor-System.

Hinzu kommt, daß die MODULAR-4 Karte hardwaremäßig durch bis zu 9 aufsteckbare Module an praktisch alle Meß-, Steuer- und Kommunikationsaufgaben angepaßt werden kann. Dies hat für den Entwickler den Vorteil, daß er sich nur in ein System "eindenken" muß.

Die Karte kann auch ohne irgendwelche Änderungen in praktisch allen mit dem IBM-kompatiblen PCs mit ISA- oder EISA-Bus eingesetzt werden. Da der Begriff "Kompatibilität" von einigen Herstellern solcher Computer sehr großzügig ausgelegt wird, sollte man zwischen Hardware- und Softwarekompatibilität unterscheiden:

Bezüglich der Softwarekompatibilität werden an den PC keinerlei Anforderungen gestellt. Als Hardwarevoraussetzungen sind lediglich die mechanischen Abmessungen der Karte und der Busanschluß (ISA- oder EISA-Bus) von Belang. Beides entspricht dem IBM-Standard.

Die MODULAR-4/486-Basiskarte enthält bereits eine Reihe von Funktionen und Schnittstellen:

- 6 Timer (3 x 16 Bit Timer im Baustein 8254, 1 Taktgeber in der Uhr, 2 x 16 Bit Timer im Baustein SCC 8530), alle interruptfähig (die beiden Timer im SCC 8530 werden standardmäßig als Baudratengeneratoren verwendet).
- Watchdog Timer für die lokale 486-CPU
- 2 serielle RS-232 Schnittstellen
- Uhrzeit/Datum, über externe Batterie pufferbar
- Cache-RAM on-board
- Arithmetik-Coprozessor (nur bei 486-DX CPUs)
- Interrupt-Eingänge
- LED on-board, auch als TTL-Ausgang verfügbar
- Ausgang für eine weitere LED
- Zwei bzw. vier Steckplätze für Erweiterungsmodule (SPB-Module)
- Überwachung der Versorgungsspannung mit NMI-Auslösung bei Power-Fail.

Ebenfalls enthalten sind RAM (je nach Version mit 256 KByte bis 4 MByte stat. RAM und bis zu 32 MByte dyn. RAM lieferbar) und EPROM bzw. Flash-EPROM (ausbaubar auf der Basiskarte bis 512 KByte). Im EPROM befindet sich das sehr schnelle Echtzeit-Multi-Tasking Betriebssystem 'OsX'. Anwenderprogramme können auf einfache Weise ins RAM der Karte geladen werden und laufen dann im Multi-Tasking Betrieb auf der Karte.

Die im Sourcecode mitgelieferten PC Programmbeispiele in C, Pascal, Visual Basic und Delphi zeigen, wie die Karte vom PC aus angesprochen wird. Hierzu stehen auch die entsprechenden PC Programm-Bibliotheken zur Verfügung, die im Lieferumfang enthalten sind.

Für die Kommunikation mit dem PC steht eine schnelle parallele 16-Bit-Schnittstelle (DMA- und Interruptfähig) zur Verfügung. Darüber können in beiden Richtungen gleichzeitig Daten und Programme ausgetauscht werden.

Zur Entwicklungsunterstützung für die Karte finden Sie auf der mitgelieferten CD bzw. den mitgelieferten Disketten das Test-, Service- und Debug-Programm SNW bzw. SNW32.

Eigene Echtzeit-Programme, die auf der Karte laufen, können mit den üblichen PC Programmiersprachen wie Turbo-Pascal und Borland C++ erstellt werden. Auch der Borland Turbo-Debugger (Remote Debugger) ist einsetzbar. Beispiele im Source-Code finden Sie in Kapitel 7. Weitere Informationen zum Remote-Debugging, zu den im Betriebssystem vorhandenen Assembler-Subroutinen (z.B. Intertask-Kommunikation) und der dazugehörigen Hochsprachen-Bibliothek finden Sie in den Kapiteln 8, 9 und 10.

1.1. Kompatibilität zu anderen MODULAR-4 Karten

Es sind nur wenige Punkte zu beachten, wenn Sie bisher mit einer MODULAR-4/Z80 (ML4) oder MODULAR-4/Z280 (ML5) Karte gearbeitet haben und nun eine leistungsfähigere MODULAR-4/486 Karte (ML7 oder ML8) einsetzen wollen.

1.1.1. Hardware

1. Die Programmierung der **Timer** ist bei der /486-Karte anders. Der Eingangstakt für die drei Timer des Timer-Chips 8254 beträgt bei der /486-Karte 1 MHz, 2,5 MHz oder 10 MHz (per Software einstellbar[1]), bei der /Z80-Karte 2,5 MHz. Die Timer-Ausgänge auf dem SP-Bus (= SORCUS-Prozeß-Bus) haben folgende Zuordnung:

bei /Z80 bzw. /Z280	entspricht bei /486	SP-Bus
TIMER-A	TIMER-A	A7
TIMER-B (Z80) bzw. Sio-Clock (Z280)	Sio-Clock	B7
TIMER-C	TIMER-B	B14
TIMER-D	TIMER-C	C7

[1] "kleine" MODULAR-4/486: per Software wählbar: 1 MHz, 2,5 MHz oder 10 MHz
"große" MODULAR-4/486, Rev. C: per Software wählbar: 2,5 MHz oder 10 MHz
"große" MODULAR-4/486, Rev. B: Bestückungsoption, siehe Prüfbericht der Endkontrolle

2. Die Programmierung der **Interrupt**-Controller ist bei der /486-Karte anders. Die Interrupt-Eingänge (ext. und auf dem SP-Bus) haben eine andere Bezeichnung: IRQ-x statt INT-n (x = A, B, C, ...; n = 0, 1, 2, ...). Die Eingänge sind im Prinzip gleichwertig, auch die aktive Flanke läßt sich programmieren. Es gilt folgende Zuordnung:

bei /Z80 und /Z280	entspricht bei /486	SP-Bus
INT-1	IRQ-A	A8
INT-2	IRQ-F	B8
INT-3	IRQ-B	A9
INT-4	IRQ-E	B9
INT-5	IRQ-C	A10
INT-7	IRQ-D	B10

3. Die **Echtzeituhr** ist bei der /Z280-Karte und der /486-Karte identisch, aber ein anderer Typ als bei der /Z80-Karte.

4. Die Programmierung der **seriellen Schnittstellen** ist bei der /486-Karte anders.

5. Alle **SPB-Module**, die auf der /Z80- und /Z280-Karte Verwendung finden, können auch auf der /486-Karte eingesetzt werden, mit folgenden Ausnahmen bzw. Anpassungen (siehe hierzu auch die Übersicht im Anhang):

   a) Das Modul **M-RU8-2** (8 Relaisausgänge, Typ 13) ist auf der /486-Karte erst ab Revision D des Moduls einsetzbar.

   b) Das Modul **M-5B-1/U** (Kombimodul und 5B-Ankopplung, Typ 20) muß per Jumper für die /486-Karte konfiguriert werden.

   c) Das Modul **M-iNC-3** (Inkrementalgeber/Zähler, Typ 28) ist ein Sonderfall. Der Timertakt auf der /486-Karte muß von 2,5 MHz auf 10 MHz geändert werden. Alternativ kann das Modul M-C16-3 (Typ 49) eingesetzt werden.

   d) Die Kommunikationsmodule **M-SiO-A/i** (Typ 4), **M-SiO-A/R** (Typ 5), **M-SiO-A/iR** (Typ 25), **M-422-2/A** (Typ 26), **M-422-2/S** (Typ 27) und **M-422-2** (Typ 36) enthalten spezielle Z80-Bausteine und können auf der /486 Karte nicht eingesetzt werden. Für Neuentwicklungen und für die /486-Karte steht ein verbessertes Kommunikationsmodul M-COM-2 (Typ 32 oder 33) zur Verfügung. Dieses Modul kann bei allen Karten eingesetzt werden. Die Programmierung ist auch sehr ähnlich wie bei den oben erwähnten Modulen.

1.1.2. Software

1. Die Makrobefehle der /486-Karte haben ein anderes Format und andere Codes als bei der /Z80- und /Z280-Karte, einige können aber identisch verwendet werden, z.B. die zur Identifikation der Karte. Von der PC Hochsprache (PC Bibliothek) aus betrachtet sind die Unterschiede aber gering, von einigen Erweiterungen bei der /486 Karte abgesehen.
2. Die Bedeutung der Wörter im EEPROM der Basiskarten ist unterschiedlich.
3. Für das Schreiben eigener Echtzeit-Programme gelten andere Konventionen (siehe Kapitel 7, 8, 9 und 10). Da das Prinzip des Betriebssystems aber bei allen Karten identisch ist, sind keine Strukturänderungen notwendig.

1.2. Lieferumfang

Zum Lieferumfang gehört:

1. Die MODULAR-4/486 Basiskarte.
2. Ein Prüf- und Bestückungsbericht, der die Spezifikationen zur Karte dokumentiert. Dort finden Sie die Seriennummer, Angaben zur Konfiguration des Speichers (RAM und EPROM), zu den Versionen der on-board Software, zu den Quarzfrequenzen, Jumpereinstellungen usw.
3. Dieses Handbuch
4. CD bzw. Diskette/n, mit
   - Hilfs- und Testprogrammen sowie
   - Bibliotheken, Treiberprogramme und Beispielprogramme für Echtzeit- und PC-Programmierung
5. Einige zusätzliche Kurzschlußstecker (Jumper), sofern erforderlich

1.3. CE-Kennzeichnung

Das MODULAR-4/486 System ist ein OEM-Produkt und als Zulieferteil für die Weiterverarbeitung durch Industrie, Handwerk oder sonstige auf dem Gebiet der elektromagnetischen Verträglichkeit fachkundigen Betriebe oder Personen bestimmt. Im Sinne des EMVG vom 18. September 1998 §6 Abs. 9 besteht daher für das MODULAR-4/486 System keine CE-Kennzeichnungspflicht.

Verkabelung, verwendeter PC und die Einsatzumgebung sind Faktoren, die sich auf die EMV eines Gerätes auswirken können. Ein Gerät, in das eine oder mehrere MODULAR-4/486-Karten bestückt mit SPB-Modulen eingesetzt wurden, muß in seiner Gesamtheit entsprechend den dafür gültigen Richtlinien bewertet werden, wenn mit dem CE-Kennzeichen Konformität dokumentiert werden soll oder muß.

Selbstverständlich wurden bei der Entwicklung des MODULAR-4/486-Systems alle möglichen Maßnahmen für einen EMV-gerechten Aufbau ergriffen.

2. Installierung und Einbau

Die Karte ist elektrostatisch geschützt verpackt. Beim Auspacken sollte unbedingt darauf geachtet werden, daß die Karte nicht elektrostatischen Entladungen ausgesetzt wird.

Nach dem Auspacken sollte die Lieferung zunächst auf Vollständigkeit und Unversehrtheit und die Karte auf Übereinstimmung mit dem beiliegenden Prüfbericht überprüft werden.

Bei Beschädigungen oder sonstigen Fehlern setzen Sie sich bitte umgehend mit Ihrem Lieferanten oder mit SORCUS Computer GmbH in Verbindung.

Es empfiehlt sich folgender Arbeitsablauf:

1. Überprüfen auf Vollständigkeit und Unversehrtheit der Lieferung, sowohl der Basiskarte als auch der einzelnen Module.
2. Erstellen einer Arbeitskopie der mitgelieferten Diskette/n bzw. CD (siehe Seite 2-3).
3. Auspacken der MODULAR-4 Karte und der Module (Vorsicht vor elektrostatischen Aufladungen!).
4. Überprüfen (!) und Einstellen der gewünschten Konfiguration der Basiskarte (z.B. PC I/O-Adresse, siehe Kapitel 2.2) und auf den Modulen (siehe Modulbeschreibungen)
5. Die beiden 9-poligen D-Submin.-Stecker auf der Basiskarte sind zwar am Befestigungsbügel angeschraubt, aber die Kontakte sind auf der Leiterplatte nicht angelötet. Dies ist kein Versehen. Wir wollen Ihnen alle Möglichkeiten offenlassen, die Anschlußkabel und Stecker für diese beiden Schnittstellen und gegebenenfalls auch für die Module frei zu konfigurieren. Wenn Sie die D-Submin.-Stecker so verwenden wollen, wie wir sie ausgeliefert haben, löten Sie bitte alle 18 (9+9) Kontakte auf der Lötseite der Leiterplatte an. Um Flachbandkabel aus dem PC herauszuführen, ist ein spezieller Befestigungsbügel mit Zugentlastung empfehlenswert, der bei SORCUS Computer GmbH erhältlich ist
6. Wenn Sie die Uhr oder das statische RAM auf der Karte puffern wollen, ist hierfür eine externe Batterie oder die Batterie des PC zu verwenden (Anschlußplan siehe Seite 2-12).

7. Nur für "große" MODULAR-4/486: Dokumentation aller Jumpereinstellungen in die dafür vorgesehene Tabelle auf Seite 2-17 in diesem Handbuch.

8. Wenn auf Steckplatz 1 der MODULAR-4/486 Karte ein Modul steckt, ist beim Einbau der Karte in einen 16-Bit-Slot des PC besonders sorgfältig vorzugehen. Zwischen der unteren IC-Reihe auf dem Modul und dem Slotstecker des PC ist nicht viel, aber doch genügend Platz. Schrauben Sie den unteren Abstandsbolzen vom Modul ab. Stecken Sie das Modul auf Steckplatz 1 nur so weit in die Karte wie die anderen Module mit Abstandsbolzen.

9. Ausschalten des PC, PC öffnen. Wie das gemacht wird, steht in den Handbüchern zu Ihrem PC (vorher unbedingt Netzstecker herausziehen!). Einstecken der Karte in den PC (ohne Module und ohne daß irgendwelche Kabel an die Karte angeschlossen sind), dann Einschalten des PC.

   Die Karte wird, wie andere PC-Karten auch, in den PC eingesteckt und mit einer Schraube am Bügel gesichert. Der 16-Bit ISA-Steckplatz, an dem die Karte eingesteckt wird, ist im Prinzip gleichgültig. Aus Gründen der Störsicherheit wird aber empfohlen, die Karte möglichst direkt neben einer Seitenwand und mit Abstand zu den Floppy-Disk- und Hard-Disk-Laufwerken und Kabeln zu plazieren. Wenn noch weitere Steckplätze frei sind, sollte zwischen der MODULAR-4 Karte und der nächsten Karte ein Steckplatz frei bleiben.

! *Es ist besonders darauf zu achten, daß die Karte weder mechanisch noch elektrisch Kontakt zu einer benachbarten Karte hat, ebenso, daß die Karte und die Module an der unteren Kante keinen Kontakt mit Bauteilen auf dem Motherboard des PC haben.*

10. Installieren des Treibers, siehe Kapitel 2.6.

11. Starten des Programms "SNW" oder "SNW32" zur Prüfung der Funktionsfähigkeit der Karte.

12. Ausschalten des PC, Aufstecken der Anschlußkabel auf die Module (das andere Ende der Kabel darf nirgendwo angeschlossen sein!), Aufstecken der Module auf die Basiskarte, Karte wieder in den PC einbauen und den PC einschalten.

13. Starten des Programms "SNW" oder "SNW32" und Überprüfen aller Schnittstellen.

Falls beim Einsatz mit der Karte einmal Probleme auftauchen sollten, ist das Programm SNW oder SNW32 auch geeignet, um einen möglichen Fehler auf der Karte oder auf den Modulen zu lokalisieren. Dazu empfiehlt es sich, zunächst alle Kabelverbindungen zur Karte zu unterbrechen und z.B. mit Hilfe eines Funktionsgenera-

tors, eines Oszillographen oder eines Voltmeters die Ein- und Ausgangssignale und einzelne Funktionsgruppen auf der Karte zu testen.

Sind die einzelnen Tests erfolgreich verlaufen, können die gewünschten externen Anschlüsse aufgesteckt werden.

Die Karte ist nun betriebsbereit.

2.1. Mitgelieferte Software

Die beiliegenden 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.

Da sich die Zahl der Programme auf den mitgelieferten Daternträgern entsprechend dem aktuellen Entwicklungsstand ändert, ist an dieser Stelle kein Inhaltsverzeichnis angegeben. Aktualisierte Versionen können jederzeit ohne zusätzliche Gebühr von der SORCUS Homepage **www.sorcus.com** heruntergeladen werden

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 ein fehlerfreies Funktionieren dieser Programme wird von SORCUS Computer GmbH aber nicht übernommen. Insbesondere werden jede Haftung, Gewährleistung und eventuelle Schadenersatzansprüche, die sich aus dem Einsatz der MODULAR-4 Karte sowie der mitgelieferten Software ergeben könnten, ausgeschlossen.

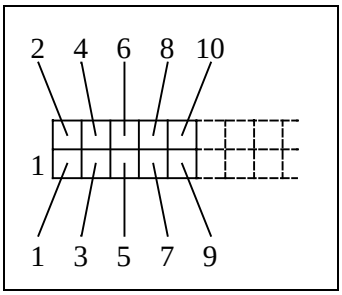
2.2. Einstellungen auf der "großen" MODULAR-4/486 Karte

Vor dem Einbau in den PC können Sie auf der MODULAR-4 Karte verschiedene Konfigurationen und Adressen einstellen. Hierzu sind Pfostenstecker vorgesehen. Durch Aufstecken oder Abziehen von Kurzschlußbrücken werden die Einstellungen vorgenommen. Werkseitig ist bereits die auf Seite 2-5 angegebene Konfiguration eingestellt.

! *Überprüfen Sie in jedem Fall alle Brücken.*

Wenn Sie Brücken ändern, empfiehlt es sich, die neue Konfiguration in der Tabelle auf Seite 2-17 zu **dokumentieren**.

Auf der Bestückungsseite der Karte (siehe Lageplan) sind die Jumper bzw. Jumperfelder mit J1 bis J8 bezeichnet, alle Stecker für externe Anschlüsse mit St1 bis St3, die Steckplätze für die vier SPB-Module mit M1 bis M4. Eine "1" kennzeichnet Pin 1 eines Jumperfeldes, einer Brücke, eines Steckers oder eines ICs. Die nebenstehende Abbildung zeigt die Zählweise bei zweireihigen Pfostensteckern.



Folgende Jumper sind zu überprüfen bzw. einzustellen: (siehe auch Lageplan auf Seite 2-16)

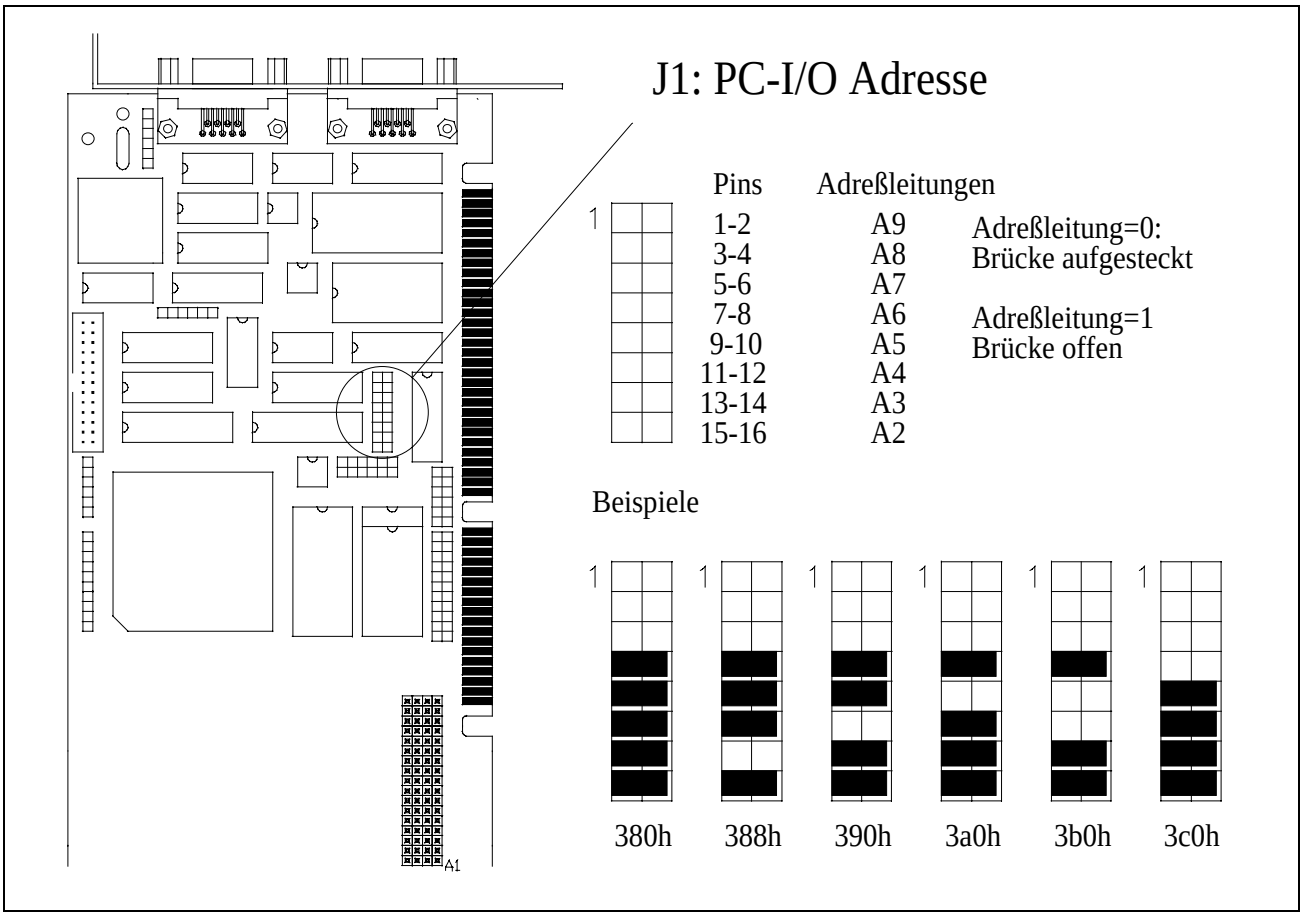
Brücke	Anzahl Pins	Standard-einstellung	Funktion
J1	2x8	7-8, 9-10, 11-12, 13-14, 15-16 (380h)	PC I/O Adresse der MODULAR-4 Karte
J2	2x6	4-6 (IRQ-7)	PC Interrupt-Kanal für die Karte
J3	2x11	3-5, 4-6 (DMA-1)	PC DMA-Kanal für die Karte
J4	1x6	1 2 offen (nicht aktiv)	Watchdog-Timer aktivieren
		3 4 offen (nein)	Power-Fail Signal an /NMI
		5 6 offen (200ms) (Rev. B: 1,6 s)	Watchdog-Timeout-Zeit: 100 ms (Rev. B) bzw. 200 ms (Rev. C) oder 1,6 s
J5	1x10	4-5, 6-7 (ohne Batterie)	Batterie konfigurieren, für die Uhr und gegebenenfalls für RAM (siehe J8)
J6	1x6	1-2 (DTR/A)	Funktion der DTR-Leitung der seriellen Schnittstelle A
		5-6 (TRXC/B)	Takt der seriellen Schnittstelle A (RTXC/A): Ri/A oder TRXC/B
J7	2x6	5-7, 1-2, 6-8 (27C512)	Konfiguration für den EPROM-Sockel
J8	1x6	2-3, 5-6 (+5 Volt)	Pufferung des statischen RAM: +5 Volt oder Batterie, für Bank 1 und Bank 2
St1	2x13	1-3	Eingang Spannungsüberwachung A an +5 Volt (nur bei Rev. B)

Im Lieferumfang der MODULAR-4/486 finden Sie einen Prüfbericht der Endkontrolle, der die Gesamtkonfiguration der Karte angibt, also auch die Versionsnummer des EPROM-Betriebssystems, etc. **!**

2.2.1. Einstellen der PC-I/O-Adresse (J1)

Die Karte belegt 4 Adressen im I/O-Adreßbereich des PC. Die eingestellte Adresse darf in Ihrem PC nicht von weiteren Komponenten verwendet werden. Wird eine der reservierten Adressen in Ihrem PC nicht verwendet, so kann auch diese eingestellt werden. Eine Übersicht über die Verwendung von I/O-Adressen im IBM-AT finden Sie in Kapitel 2.4.

J1 legt die Basisadresse (BA) fest, und zwar durch Setzen der I/O-Adreßleitungen A2 bis A9. Um eine Adreßleitung = 0 zu setzen, muß die angegebene Brücke aufgesteckt werden. Soll eine Adreßleitung = 1 gesetzt werden, darf die entsprechende Brücke nicht aufgesteckt sein.

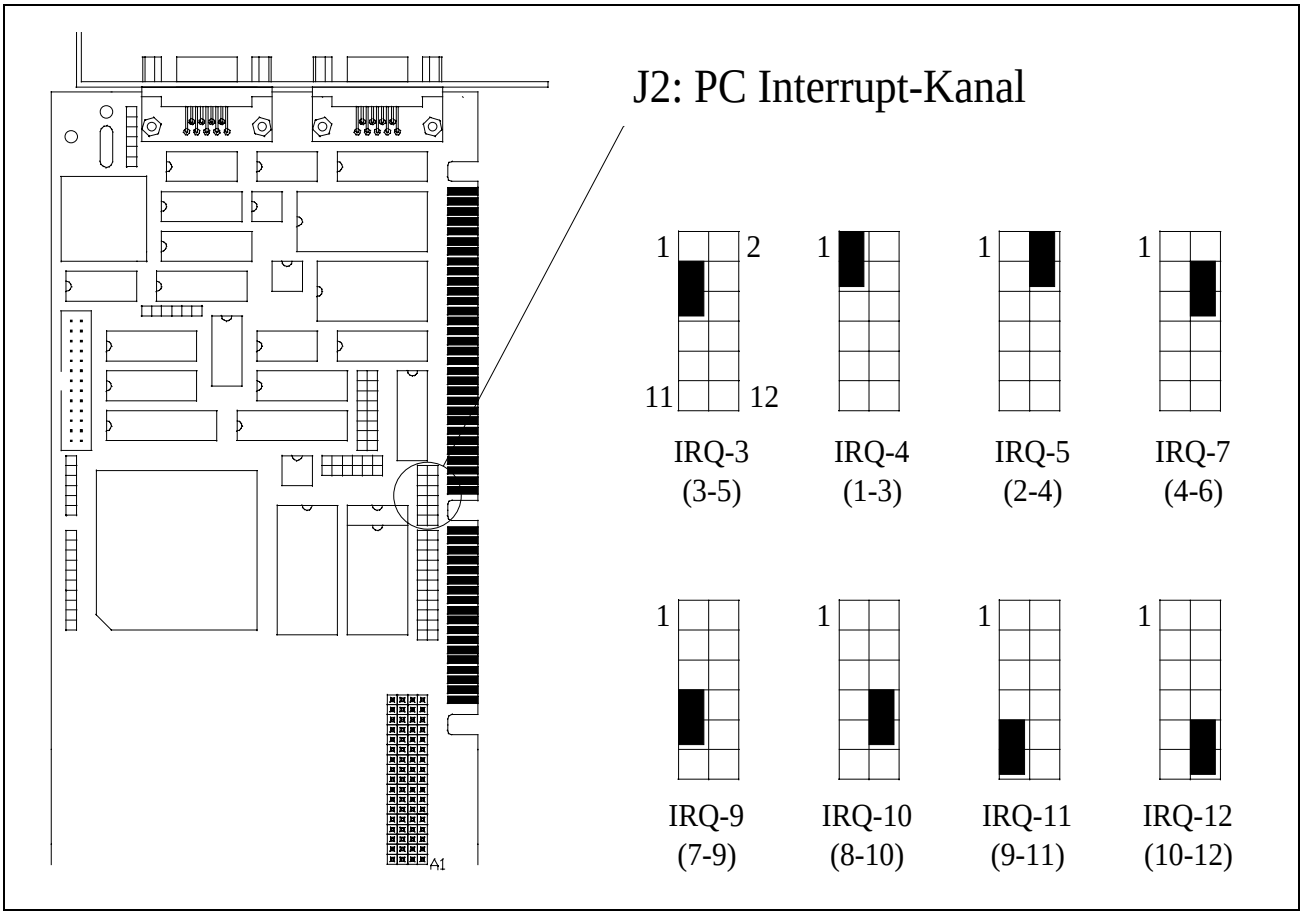


2.2.2. Einstellung eines PC Interrupt-Kanals (J2)

Wenn Daten oder Befehle von der Karte zum PC gesendet werden, so kann bei jedem übertragenen Byte bzw. Wort ein Interrupt auf dem PC ausgelöst werden. Bei der Kommunikation vom PC zur Karte kann der gleiche Interrupt-Kanal auch verwendet werden, um dem PC anzuzeigen, daß die Karte ein vom PC zur Karte gesendetes Byte bzw. Wort empfangen hat und bereit ist, das nächste Byte bzw. Wort zu empfangen.

Ob und unter welchen Bedingungen die Karte einen Interrupt-Request zum PC meldet, wird per Software eingestellt (z.B. durch Senden von Makrobefehlen vom PC zur Karte oder auch durch ein Programm auf der Karte selbst). Die Karte verwendet für alles denselben PC Interrupt-Kanal, der per Steckbrücke eingestellt werden muß. Bei Verwendung der mitgelieferten PC Bibliothek ML8BIB muß der hier eingestellte Interrupt-Kanal angegeben werden. Alles übrige erledigt dann die PC Bibliothek.

Es stehen die PC Interruptkanäle 3, 4, 5, 7, 9, 10, 11 und 12 zur Verfügung.

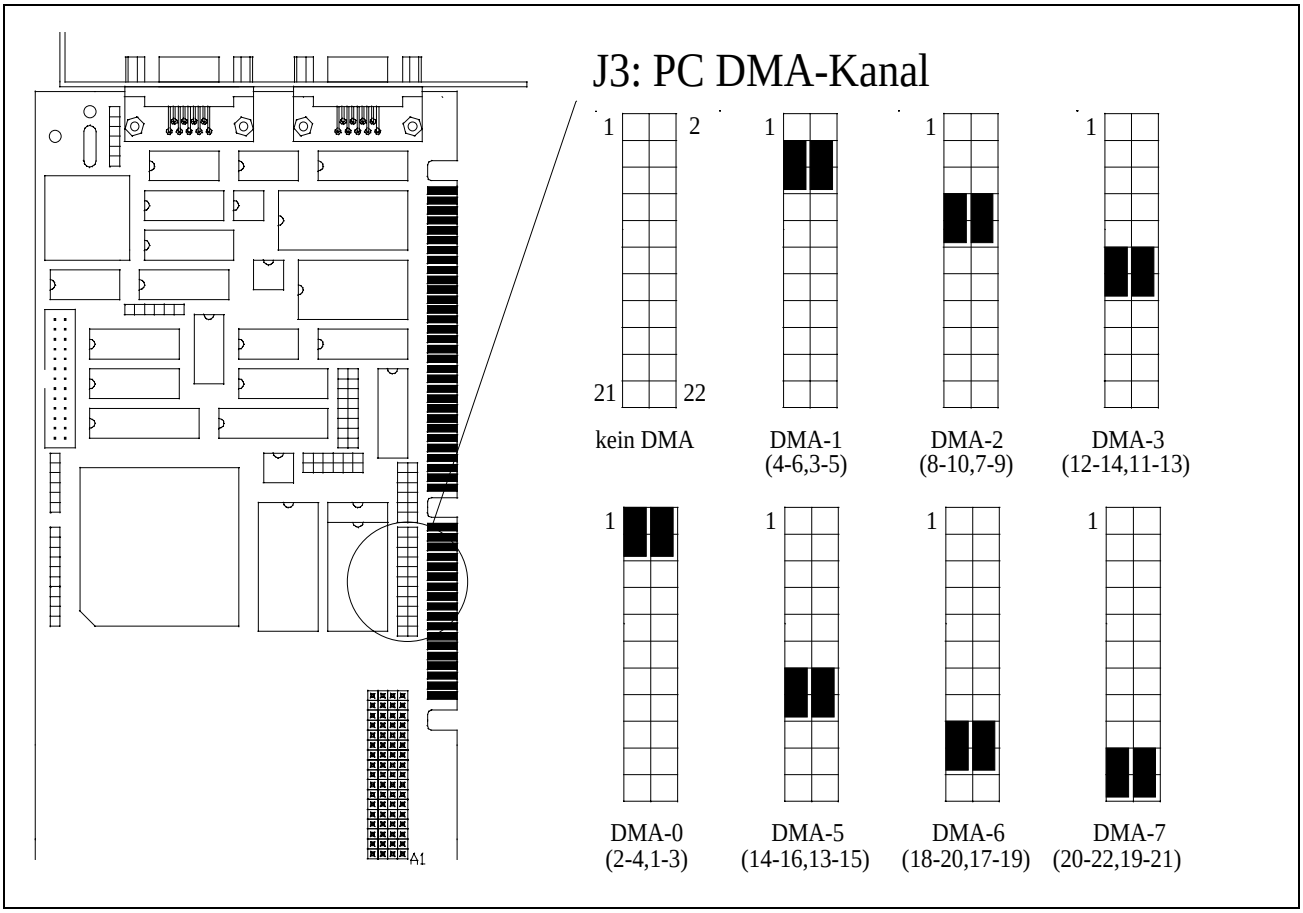


Eine Übersicht über die Verwendung von Hardware-Interrupts im IBM-AT (BIOS) finden Sie in Kapitel 2.5.

2.2.3. Einstellung eines PC-DMA-Kanals (J3)

Daten und Befehle können vom PC zur Karte und auch von der Karte zum PC per DMA übertragen werden. Der entsprechende Kanal muß per Steckbrücken eingestellt werden. Wenn Sie ohne DMA arbeiten, kann trotzdem ein Kanal eingestellt sein. Die Aktivierung des Kanals geschieht per Software, ebenso wie die Richtung der Kommunikation. Ob und unter welchen Bedingungen die Karte einen DMA-Request zum PC meldet, wird per Software eingestellt (z.B. durch Senden von Makrobefehlen vom PC zur Karte oder auch durch ein Programm auf der Karte selbst). Die Karte verwendet für alles denselben PC DMA-Kanal, der per Steckbrücke eingestellt werden muß. Die derzeit ausgelieferten PC-Bibliotheken unterstützen keinen DMA-Betrieb.

Es stehen die PC DMA-Kanäle 0, 1, 2, 3, 5, 6 und 7 zur Verfügung. Die Kanäle 0, 1, 2 und 3 übertragen ein Byte pro DMA-Zyklus, die Kanäle 5, 6 und 7 ein Wort (16 Bit).



2.2.4. Watchdog-Timer und Powerfail (J4)

Zur Überwachung eines ordnungsgemäßen Programmablaufs sind auf der Karte verschiedene Möglichkeiten vorhanden:

Einrichtung	**spricht an, wenn**	**bewirkt**
Watchdog	Watchdog-Timer nicht rechtzeitig nachgetriggert wird	**Rev. C:** NMI **Rev. B:** Hardware-Reset
Spannungs-überwachung A	Versorgungsspannung (5 V) unter Schwelle (4,8 V bei Rev. C, mit Poti einstellbar bei Rev. B) ist	NMI
Spannungs-überwachung B	Versorgungsspannung (5 V) unter 4,65 V (bei Rev. B: 4,4 V) ist	Hardware-Reset, optional RAM und Uhr auf Batterie-Pufferung

Der **Watchdog-Timer** dient zum Abfangen von Programmabstürzen auf der MODULAR-4 Karte. Dazu muß ein korrekt laufendes Programm den Watchdog regelmäßig innerhalb eines vorgegebenen Zeitintervalls ansprechen (nachtriggern) um anzuzeigen, daß das Programm noch einwandfrei läuft. Wenn das Nachtriggern über die eingestellte Zeit hinaus ausbleibt, löst der Watchdog bei MODULAR-4 Karten der Revision C einen NMI (nicht maskierbarer Interrupt), bei Rev. B einen Hardware-Reset aus. Um ein bestimmtes Zeitschema einzuhalten, sollte das Nachtriggern an der "zeitkritischsten" Stelle aller installierten Tasks erfolgen (nur an einer Stelle!).

In Anwendungen, in denen die Interrupt-Belastung der MODULAR-4/486 sehr hoch ist, kann es z.B. dazu kommen, daß NI-Tasks nur sehr selten vom Betriebssystem aufgerufen werden. Soll überwacht werden, daß eine bestimmte NI-Task trotzdem rechtzeitig an die Reihe kommt, muß die Re-Triggerung des Watchdog-Timers in der Hauptprozedur dieser NI-Task stattfinden.

Kommt das Betriebssystem nicht dazu, die Task, die die Re-Triggerung übernimmt, rechtzeitig aufzurufen, weil ein Programm abgestürzt ist, oder andere Tasks zu viel Zeit beanspruchen, bleibt die Re-Triggerung aus. Dadurch läuft der Watchdog-Timer ab. Die MODULAR-4/486 (Rev. C) reagiert darauf mit einem NMI-Interrupt. Wenn für diesen Interrupt eine Task installiert ist und der NMI freigegeben ist, ruft das OsX-Betriebssystem die Hauptprozedur dieser Task auf. Dort besteht die Möglichkeit, einen sicheren Betriebszustand herzustellen, Daten zu retten oder den Host-PC durch einen Service-Request zu informieren.

Bei Rev. B wird statt des NMI ein Hardware-Reset ausgelöst, damit werden alle laufenden Programme abgebrochen und alle Ausgänge in einen definierten Zustand gesetzt (entsprechend den Eintragungen in den EEPROMs).

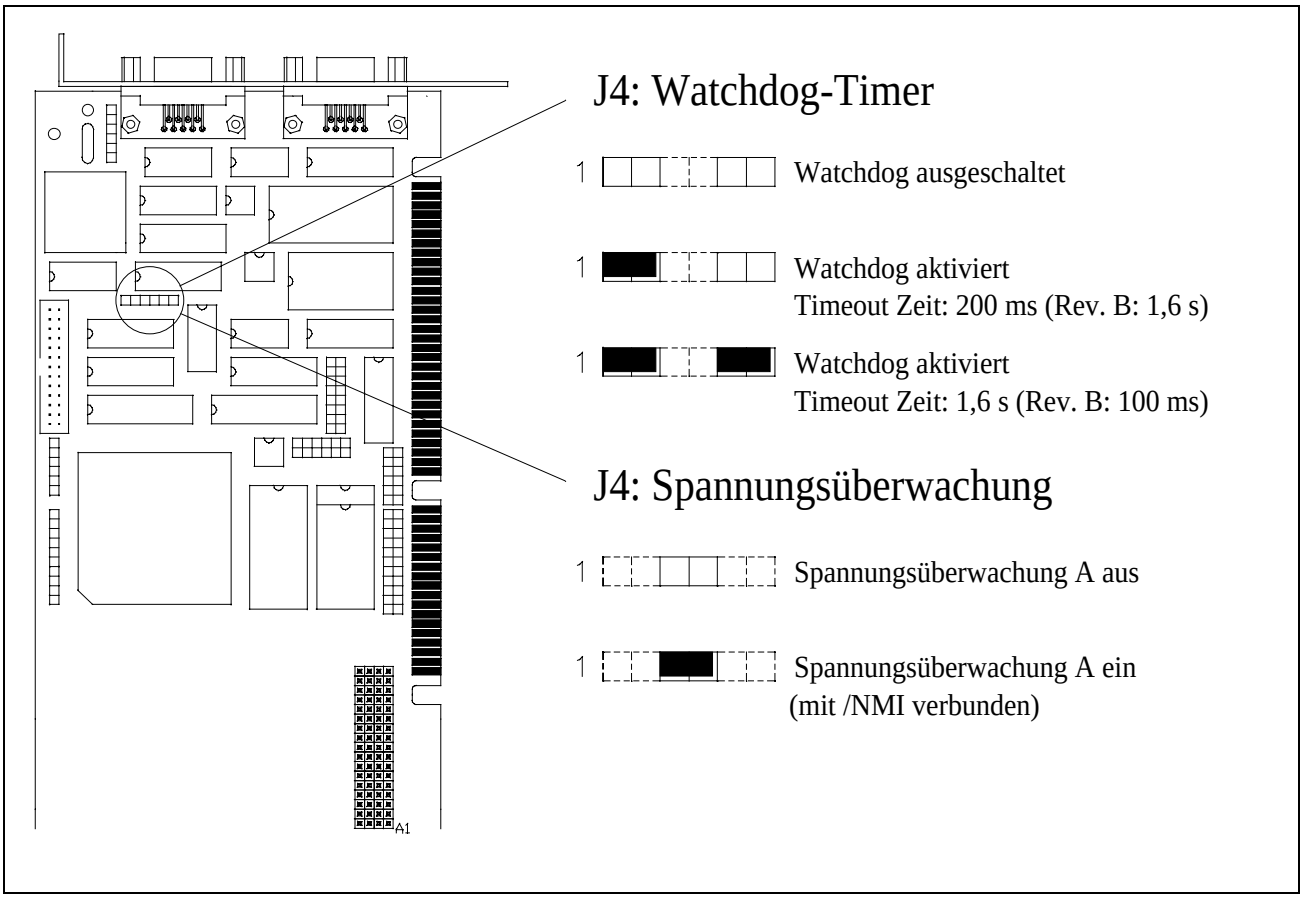
Der Watchdog kann durch Jumper J4, Pin 1 und 2 aktiviert werden (aktiviert = Jumper aufgesteckt). Die Watchdog-Timeout-Zeit, innerhalb der nachgetriggert werden muß, kann zu 1,6 s oder 100 ms (Rev. B) bzw. 200 ms (Rev. C) gewählt werden. Die erste Timeout-Zeit nach einem Hardware Reset ist - unabhängig von der Jumpereinstellung - immer 1,6 s. Das Nachtriggern kann mit einem Bibliotheksbefehl durchgeführt werden. Wenn der Watchdog nicht aktiviert ist (kein Jumper auf J4, Pin 1 und 2 aufgesteckt), hat er keine Funktion.

Watchdog	Deaktivieren		J4, Pin 1 und 2 offen
	Aktivieren		J4, Pin 1 u. 2 per Jumper verbunden
	Nachtriggern		Bibliotheksaufruf
Timeout	Rev. B: 100 ms,	Rev. C: 1,6 s	J4, Pin 5 u. 6 per Jumper verbunden
	Rev. B: 1,6 s,	Rev. C: 200 ms	J4, Pin 5 und 6 offen

Um die Funktionalität des Watchdogs nutzen zu können, sind folgende Schritte notwendig:

1. Der Watchdog muß per Jumper J4 aktiviert werden.
2. Nach der Aktivierung des Watchdogs wird dieser automatisch vom Betriebssystem nachgetriggert (Default-Einstellung). Um dies abzuschalten, ist der Betriebssystem-Parameter 214 (Byte) = 0 zu setzen (Auto-Retrigger = off).
3. Anstelle des Betriebssystems muß jetzt eine Anwender-Task die Nachtriggerung des Watchdog-Timers übernehmen. Das kann mit Hilfe der Bibliotheksfunktion **ml8rt_trigger_watchdog** geschehen. Zu beachten ist, daß jeder Zugriff auf die Echtzeituhr den Watchdog ebenfalls nachtriggert.
4. Eine II- oder DI-Task sollte unter dem NMI-Interrupt (= Interrupt Nr. 2) installiert werden, um bei Ablauf des Watchdog-Timers reagieren zu können. Zu beachten ist, daß der NMI demaskiert werden muß (**ml8rt_unmask_int(NMI)**).

Soll das Ablaufen des Watchdog-Timers einen Hardware-Reset der MODULAR-4/486 Karte auslösen, besteht die Möglichkeit, an Stecker St1 die Pins 13 (Watchdog-Ausgang) und 6 (Reset-Eingang) zu verbinden.



Die **Spannungsüberwachung A** kann dazu dienen, die 5 Volt Versorgungsspannung der Karte zu überwachen[1]. Im Fall, daß diese Spannung unter eine Schwelle (bei Rev. C ca. 4,8 V) abfällt, wird ein sog. "Nicht Maskierbarer Interrupt" (NMI) ausgelöst. Auf der MODULAR-4/486 Karte ist der NMI-Eingang der CPU aber per Software maskierbar. Durch einen Hardware-Reset oder einen entsprechenden Bibliotheksbefehl wird er maskiert, mit einem anderen Befehl wird er demaskiert. Um also einen NMI auszulösen, wenn die Schaltschwelle unterschritten ist, müssen folgende Voraussetzungen erfüllt sein:

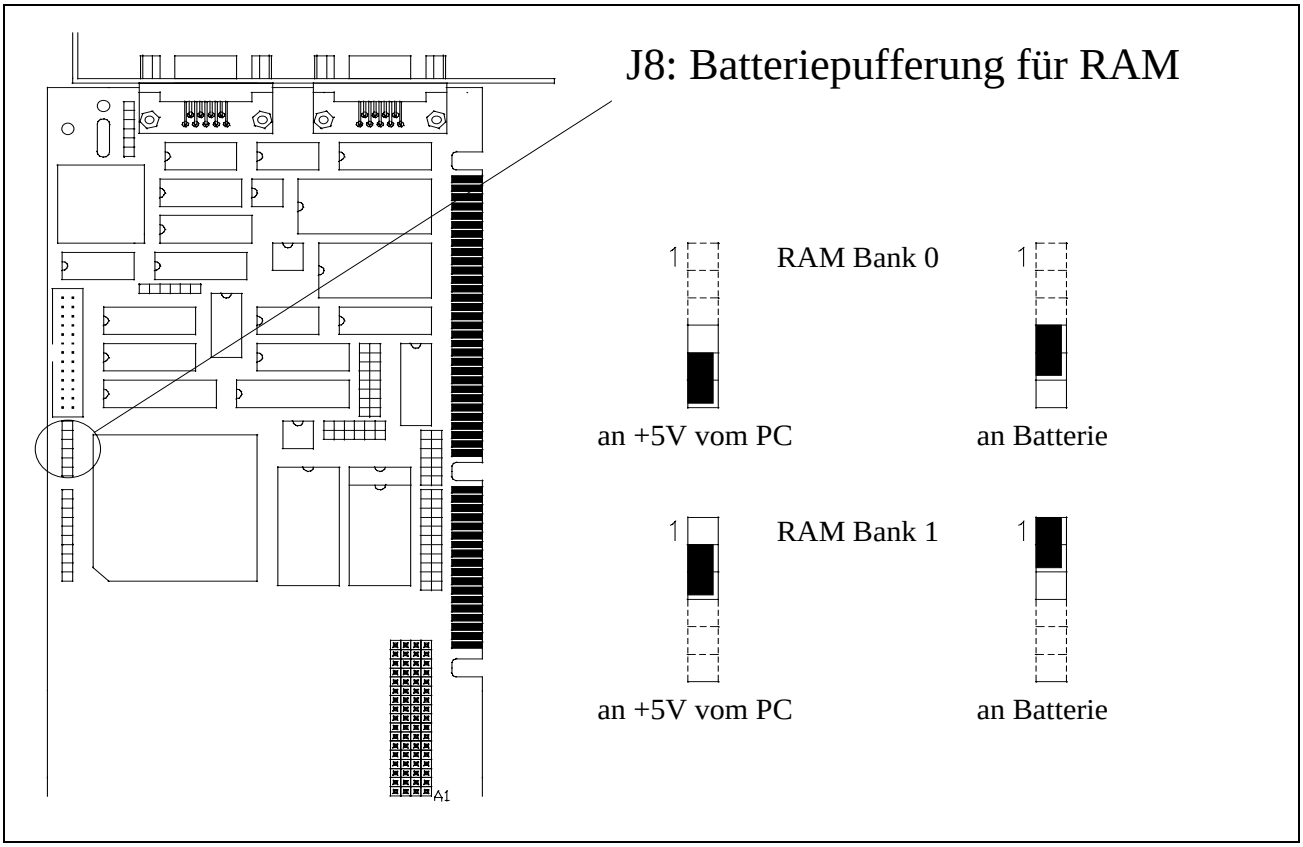
a) Ausgang von Spannungsüberwachung A an NMI legen: J4, Pin 3-4
b) NMI-Eingang aktivieren (Bibliotheksbefehl)

Die **Spannungsüberwachung B** hat eine Ansprechschwelle von ca. 4,65 Volt (bei Rev. B: 4,4 Volt). Bei Abfall der Versorgungsspannung von 5 Volt auf 4,8 Volt spricht dann also zunächst die Überwachungsschaltung A an und löst gegebenenfalls einen NMI aus. In der Interrupt-Service-Routine können von der CPU dann z.B.

[1] Bei Rev. B kann außer der Versorgungsspannung auch eine andere an St1 eingespeiste Spannung überwacht werden. Die Schwelle ist bei Rev. B mit einem Potentiometer einstellbar.

noch wichtige Daten ins RAM gerettet oder andere Dinge erledigt werden, bevor die Spannung bis auf ca. 4,65 Volt abgefallen ist. Danach ist das RAM auf Batteriepufferung geschaltet (Voraussetzung ist, daß das RAM mit J8 an die Batterieversorgung angeschlossen ist) und es erfolgt automatisch ein Hardware-Reset der Karte.

Wenn keine Batterie angeschlossen ist, sollte für RAM Bank 0 und RAM Bank 1 die Einstellung 'an +5 V vom PC' gewählt werden (siehe untenstehende Abbildung)



2.2.4.1. Auslesen der NMI-Interruptquelle

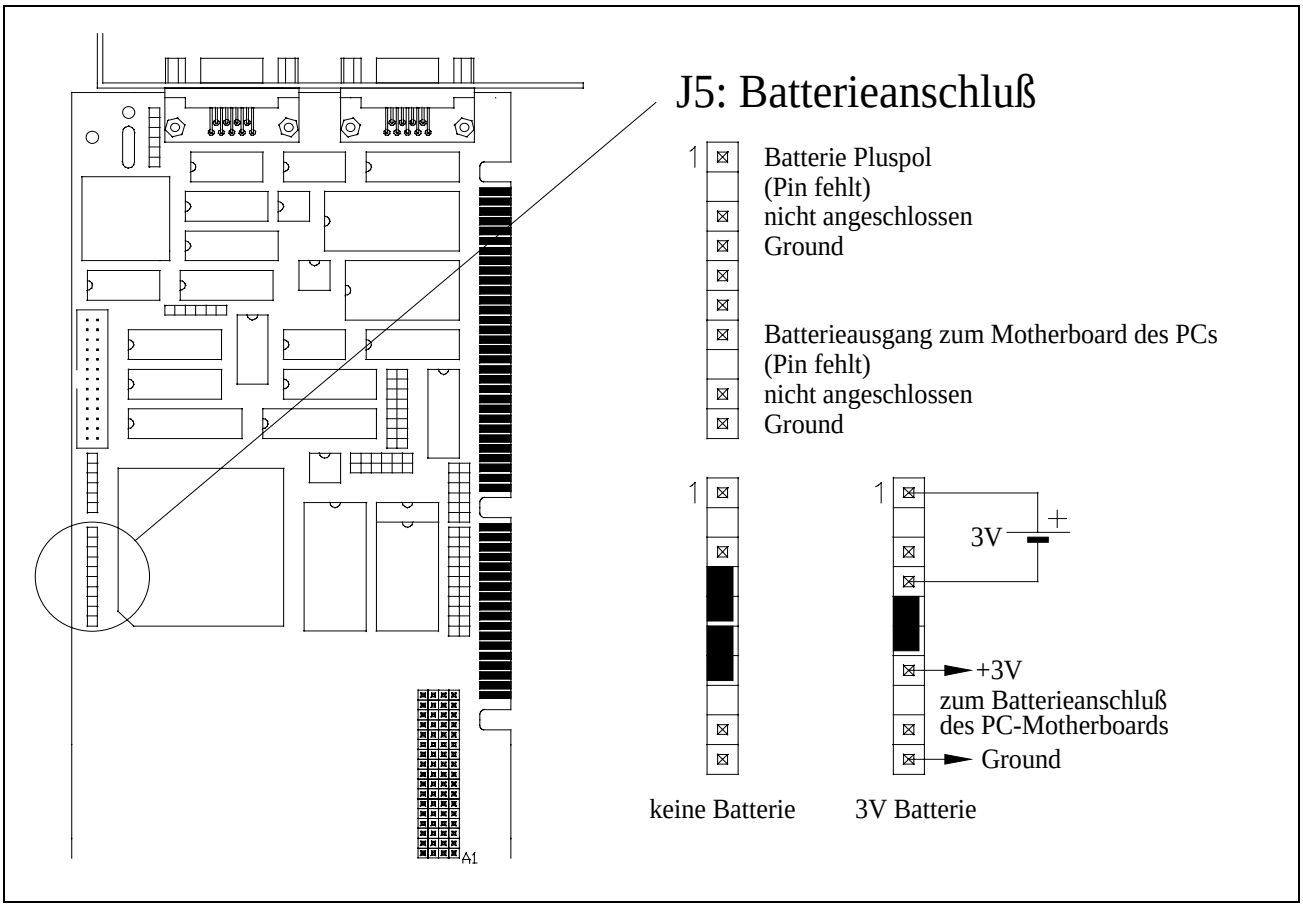
Werden mehrere NMI-Quellen freigegeben, kann die Ursache eines NMI durch einen 16-Bit Lesezugriff auf die lokale I/O-Adresse 28h herausgefunden werden. Darin ist:

- Bit 9 des gelesenen Statuswortes = 0, wenn die Spannungsüberwachung den NMI ausgelöst hat.
- Bit 10 = 1, wenn der Watchdog-Timer einen NMI verursacht hat.

2.2.5. Anschluß einer externen Batterie (J5)

Zur Pufferung des statischen RAMs und der Uhr auf der MODULAR-4 Karte kann an J5 eine Batterie angeschlossen werden. Sie wird von der Karte automatisch abgeschaltet, sobald die 5 Volt Versorgungsspannung der Karte vorhanden ist. Die Pinbelegung von J5 wurde so gewählt, daß eine übliche PC-Batterie oder auch die im PC eingebaute Batterie verwendet werden kann. Die an J5 eingespeiste Batteriespannung kann am gleichen Jumperfeld wieder entnommen werden. Eine Batterie, die sowohl die MODULAR-4 als auch das Motherboard des PCs versorgen soll, wird also direkt an die Karte angeschlossen und mit einem Kabel von J5 aus zum Motherboard weitergereicht. Es können Batterien mit 3 - 4 Volt eingesetzt werden.

Die Uhr der MODULAR-4 Karte wird immer gepuffert, wenn eine Batterie angeschlossen ist. Zur Pufferung des RAMs muß zusätzlich Jumperfeld J8 eingestellt werden. **!**

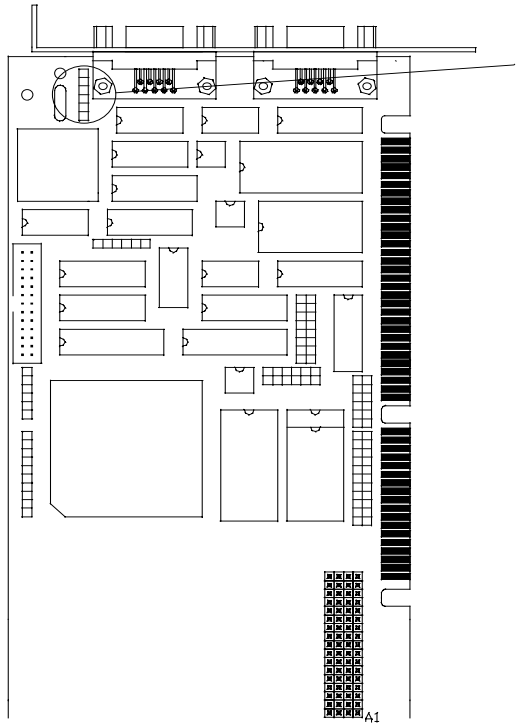


2.2.6. Konfiguration der seriellen Schnittstelle A (J6)

! *Für den auf der Basiskarte und auf den Modulen M-COM-2 und M-COM-8 eingesetzten seriellen Baustein SCC 8530 bzw. ESCC 85230 steht ein separates Handbuch zur Verfügung.*

Der Baudratentakt der seriellen Schnittstelle A kann über die DTR/A-Leitung nach außen geführt werden (RS-232 Pegel). Genauer betrachtet wird das Signal am Pin "TRXC/A" des SCC-8530 verwendet. Per Software muß dafür gesorgt werden, daß an diesem Pin dann auch der gewünschte Takt geliefert wird (siehe Beschreibung des SCC-8530, TRXC/A kann das Ausgangssignal des Baudratengenerators, den Sendetakt oder das Ausgangssignal der SCC-internen DPLL liefern). Die DTR/A-Leitung steht dann nicht mehr als Modemsteuerleitung zur Verfügung.

Der Baudratentakt für die Schnittstelle A wird immer über den Pin "RTXC/A" des SCC-8530 zur Verfügung gestellt. Das Signal an "RTXC/A" kann intern im SCC-8530 als Sendetakt, Empfangstakt, für die DPLL und für den Baudratengenerator verwendet werden. Dies ist per Software einstellbar. An den Pin "RTXC/A" kann per Jumper J6, Pin 4 bis 6, entweder das Ausgangssignal eines Quarzoszillators (Teil der Schnittstelle B, standardmäßig 7,3728 MHz) oder ein externes Signal über die Modem-Steuerleitung Ri/A eingespeist werden. Die Ri/A-Leitung kann dann nicht als Modemsteuerleitung eingesetzt werden.



J6: Serielle Schnittstelle A

Verwendung von DTR/A

1 als Modem-steuerleitung DTR/A

1 liefert Takt nach außen (von Pin "TRXCA" des SCC-8530)

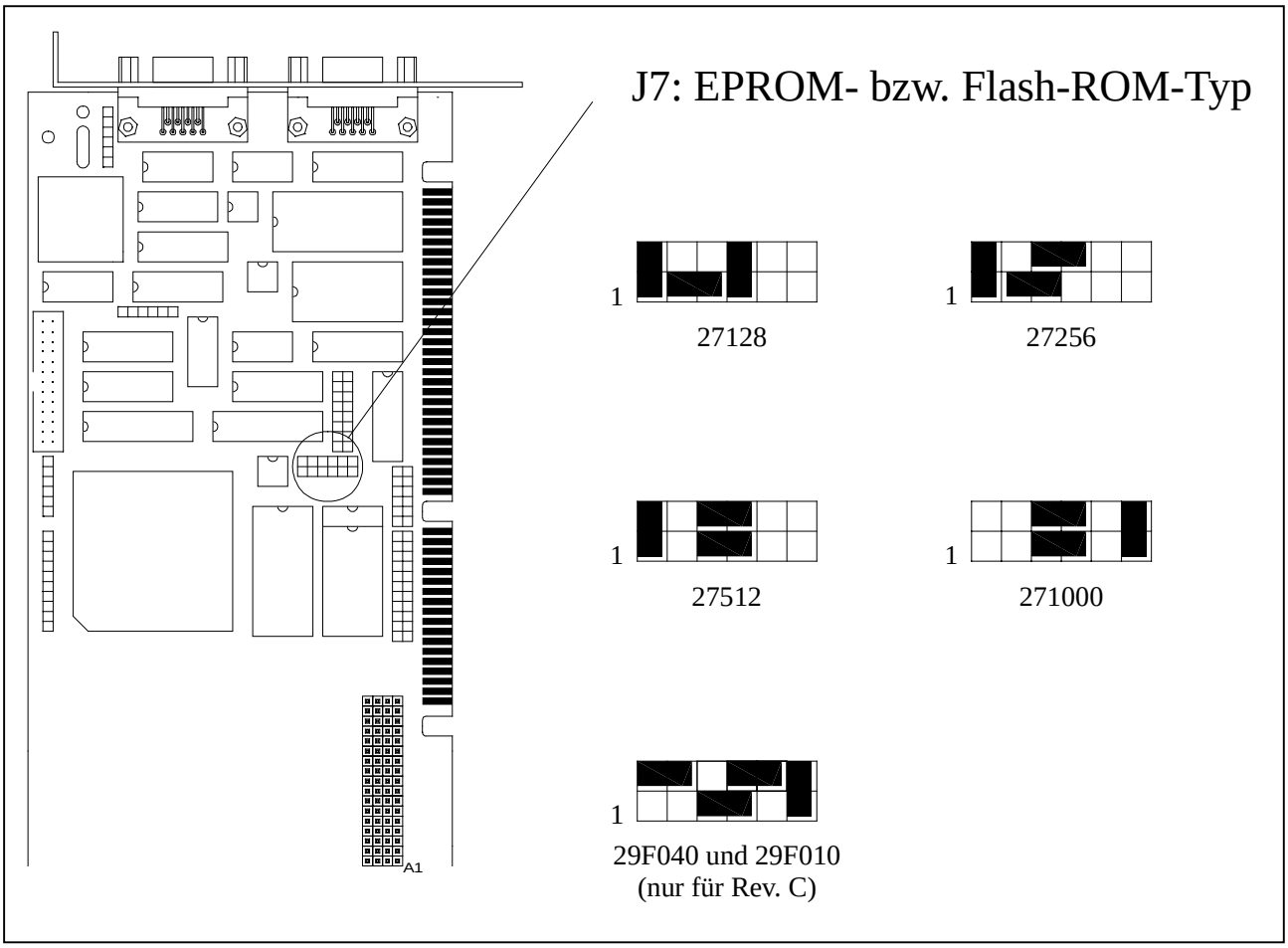
Baudratentakt

1 extern über Modemsteuer-leitung Ri/A

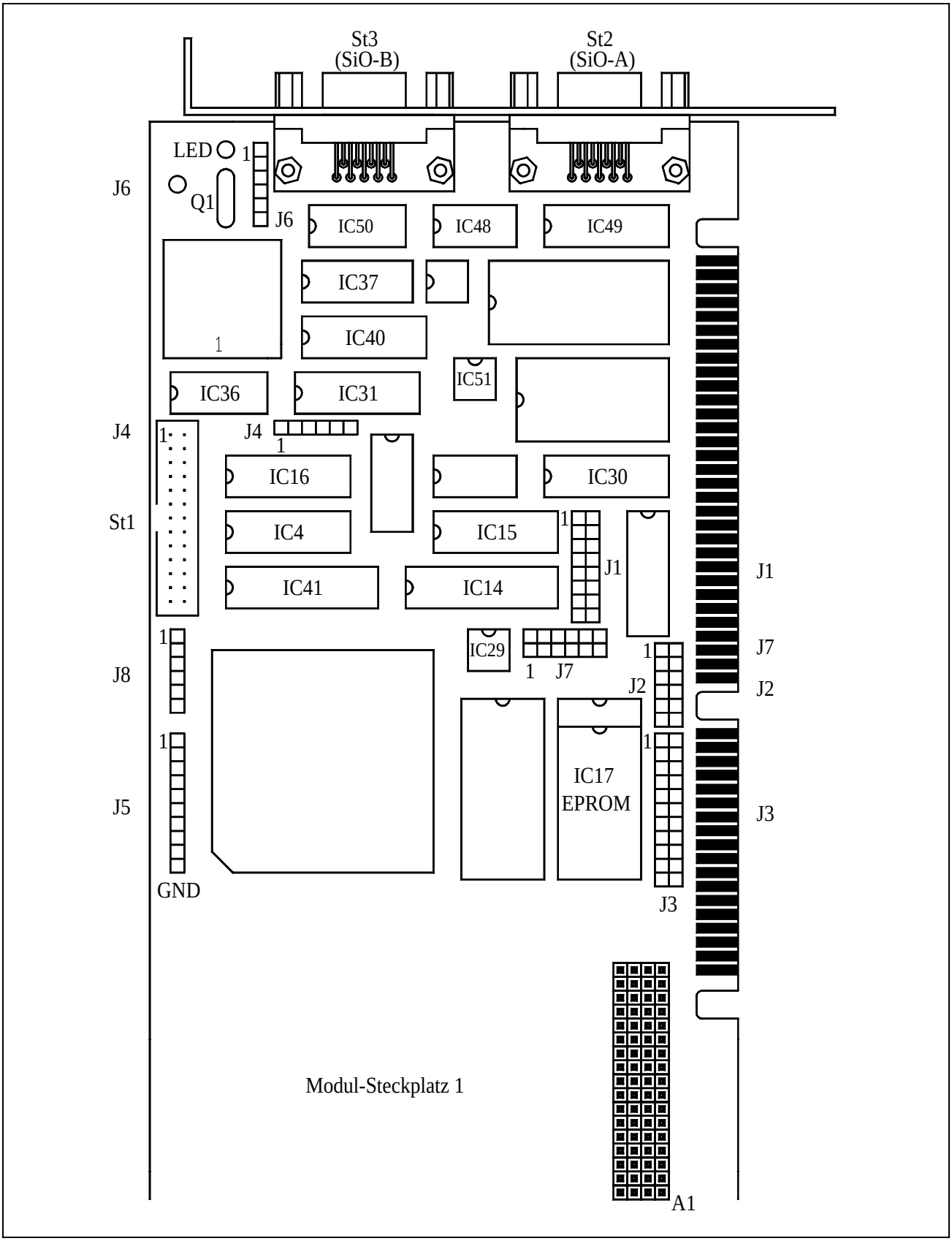
1 intern von Quarz-oszillator (Pin "RTXCA" an TRXCB des SCC-8530)

2.2.7. Konfiguration des EPROM-Sockels (J7)

Es können verschiedene EPROMs oder EEPROMs eingesetzt werden. Diese Einstellung darf nicht vom Anwender geändert werden.



2.2.8. Lageplan der "großen" MODULAR-4/486



2.2.9. Dokumentation

	werksseitig eingestellt	geändert am	von	Funktion
J1	7-8, 9-10, 11-12, 13-14, 15-16			PC I/O Adresse: 380h
J2	4-6			PC Interrupt-Kanal: IRQ-7
J3	3-5, 4-6			PC DMA-Kanal: DMA-1
J4	1 2 offen			Watchdog-Timer: nicht aktiv
	3 4 offen			Power-Fail Signal: nicht an /NMI
	5 6 offen			Watchdog-Timeout-Zeit: 200 ms (1,6 s bei Rev. B)
J5	4-5, 6-7			Batterie: keine angeschlossen
J6	1-2			Takt nach außen liefern: nein Funktion von DTR/A: DTR/A
	5-6			Takt für Kanal A: TRXC/B Funktion von Ri/A: Ri/A
J7	1-2, 5-7, 6-8			Konfiguration für das EEPROM: 27C512
J8	2-3, 5-6			Batteriepufferung RAM: aus
St1	1-3			Spannungsüberwachung A: an +5 Volt (nur bei Rev. B)

2.3. Einstellungen auf der "kleinen" MODULAR-4/486

Vor dem Einbau müssen Sie die PC I/O-Adresse der Karte (mit Drehschalter S1 bzw. Jumper J1) einstellen. Alle weiteren Einstellungen (z.B. der PC Interrupt-Kanal) werden per Software nach dem Einbau der Karte in den PC vorgenommen. Sämtliche Einstellungen werden im EEPROM der "kleinen" MODULAR-4/486 gespeichert (siehe Anhang L).

2.3.1. Einstellen der PC-I/O-Adresse (S1 bzw. J1)

Die Karte belegt 8 Adressen im I/O-Adreßbereich des PC. Die eingestellte Adresse darf in Ihrem PC nicht von weiteren Komponenten verwendet werden.

Drehschalter S1 bzw. Jumper J1 legt die PC-Basisadresse (PBA) fest (siehe Lageplan).

S1-Stellung	J1-Jumper	PC-I/O-Adresse
0	keiner	380h (werks. Einstellung)
1	1-2	388h
2	3-4	390h
3	1-2, 3-4	398h
4	5-6	280h
5	1-2, 5-6	288h
6	3-4, 5-6	290h
7	1-2, 3-4, 5-6	298h

Eine Übersicht über die Verwendung von I/O-Adressen im IBM-AT finden Sie in Kapitel 2.4.

2.3.2. Einstellung eines PC Interrupt-Kanals

Wenn Daten oder Befehle von der Karte zum PC gesendet werden, so kann bei jedem übertragenen Byte bzw. Wort ein Interrupt auf dem PC ausgelöst werden. Bei der Kommunikation vom PC zur Karte kann der gleiche Interrupt-Kanal auch verwendet werden, um dem PC anzuzeigen, daß die Karte ein vom PC zur Karte gesendetes Byte bzw. Wort empfangen hat und bereit ist, das nächste Byte bzw. Wort zu empfangen.

Ob und unter welchen Bedingungen die Karte einen Interrupt-Request zum PC meldet, wird per Software eingestellt (z.B. durch Senden von Makrobefehlen vom PC zur Karte oder auch durch ein Programm auf der Karte selbst). Die Karte verwendet für alles denselben PC-Interrupt-Kanal, der per Software eingestellt werden muß (siehe Anhang L, EEPROM-Wort 13). Bei Verwendung der mitgelieferten PC Bibliotheken ML7BIB und ML8BIB muß der im EEPROM eingetragene Interrupt-Kanal angegeben werden. Alles übrige erledigt dann die PC-Bibliothek.

Es stehen die PC-Interrupt-Kanäle 3, 4, 5, 7, 9, 10, 11 und 12 zur Verfügung. Eine Übersicht über die Verwendung von Hardware-Interrupts im IBM-AT (BIOS) finden Sie in Kapitel 2.5.

2.3.3. Watchdog-Timer und Powerfail

Zur Überwachung eines ordnungsgemäßen Programmablaufs sind auf der Karte verschiedene Möglichkeiten vorhanden:

Einrichtung	**spricht an, wenn**	**bewirkt**
Watchdog	Watchdog-Timer nicht rechtzeitig nachgetriggert wird	NMI
Spannungs-überwachung A	Versorgungsspannung (5 V) unter Schwelle 4,8 V ist	NMI
Spannungs-überwachung B	Versorgungsspannung (5 V) unter 4,65 V ist	Hardware-Reset, optional RAM und Uhr auf Batterie-Pufferung

Watchdog-Timer

Zur Überwachung der laufenden Echtzeit-Software (Tasks) stellt die "kleine" MODULAR-4/486 einen Watchdog-Timer zur Verfügung. Dieser muß durch eine Task in regelmäßigen Abständen neu gesetzt werden (re-triggern). Um ein gewünschtes Zeitschema einzuhalten, sollte die Re-Triggerung an der "zeitkritischsten" Stelle aller installierten Tasks erfolgen (nur an einer Stelle!).

In Anwendungen, in denen die Interrupt-Belastung der "kleinen" MODULAR-4/486 sehr hoch ist, kann es z.B. dazu kommen, daß NI-Tasks nur sehr selten vom Betriebssystem aufgerufen werden. Soll überwacht werden, daß eine bestimmte NI-Task trotzdem rechtzeitig an die Reihe kommt, muß die Re-Triggerung des Watchdog-Timers in der Hauptprozedur dieser NI-Task stattfinden.

Kommt das Betriebssystem nicht dazu, die Task, die die Re-Triggerung übernimmt, rechtzeitig aufzurufen, weil ein Programm abgestürzt ist, oder andere Tasks zu viel Zeit beanspruchen, bleibt die Re-Triggerung aus. Dadurch läuft der Watchdog-Timer ab. Die MODULAR-4/486 reagiert darauf mit einem NMI-Interrupt. Wenn für diesen Interrupt eine Task installiert ist und der NMI freigegeben ist, ruft das OsX-Betriebssystem die Hauptprozedur dieser Task auf. Dort besteht die Möglichkeit, einen sicheren Betriebszustand herzustellen, Daten zu retten oder den Host-PC durch einen Service-Request zu informieren.

Der Watchdog kann per Software aktiviert werden. Die Watchdog-Timeout-Zeit, innerhalb der nachgetriggert werden muß, beträgt ca. 200 ms. Die erste Timeout-Zeit nach einem Hardware-Reset ist immer ca. 1,6 s. Das Nachtriggern kann mit einem Bibliotheksbefehl durchgeführt werden. Wenn der Watchdog nicht aktiviert ist, hat er keine Funktion.

Lokale I/O-Adresse	**Zugriff**[1]	**Funktion**
94h	W8x	Watchdog-Timer disable[2]
95h	W8x	Watchdog-Timer enable
7fh	R8x	Watchdog-Timer wird nachgetriggert
44h	W8x	NMI abgeschaltet[2]
45h	W8x	NMI aktiv

Um die Funktionalität des Watchdogs nutzen zu können, sind folgende Schritte notwendig:

1. Der Watchdog muß im EEPROM (Wort-15) der "kleinen" MODULAR-4/486 oder per Software aktiviert werden.
2. Nach der Aktivierung des Watchdogs wird dieser automatisch vom Betriebssystem nachgetriggert (Default-Einstellung). Um dies abzuschalten, ist der Betriebssystem-Parameter 214 (Byte) = 0 zu setzen (Auto-Retrigger = off).
3. Anstelle des Betriebssystems muß jetzt eine Anwender-Task die Nachtriggerung des Watchdog-Timers übernehmen. Das kann mit Hilfe der Bibliotheksfunktion

[1] Zugriff auf lokale I/O-Adressen: W8x = 8-Bit Schreibzugriff, Datenbyte beliebig; R8x = 8-Bit Lesezugriff, Datenbyte ungültig.

[2] Zustand nach Reset

ml7rt_trigger_watchdog geschehen. Zu beachten ist, daß jeder Zugriff auf die Echtzeituhr den Watchdog ebenfalls nachtriggert.

4. Eine II- oder DI-Task sollte unter dem NMI-Interrupt (= Interrupt Nr. 2) installiert werden, um bei Ablauf des Watchdog-Timers reagieren zu können. Zu beachten ist, daß der NMI demaskiert werden muß (**ml7rt_unmask_int(NMI)**).

Soll das Ablaufen des Watchdog-Timers einen Hardware-Reset der MODULAR-4/486 auslösen, müssen an Stecker St1 die Pins 4 und 6, sowie die Pins 13 und 15 per Jumper verbunden werden.

Spannungsüberwachung

Die "kleine" MODULAR-4/486 stellt zwei Möglichkeiten zur Überwachung der 5-Volt-Versorgungsspannung zur Verfügung. Die Versorgungsspannung kann in zwei Stufen überwacht werden:

1. Spannungsüberwachung A:
   Auf ein Unterschreiten der Schwellspannung 4,8 V kann mit einem NMI reagiert werden (Aktivierung durch EEPROM-Einstellung).
   In einer unter dem NMI installierten Task kann versucht werden, wichtige Daten zu retten oder einen sicheren Betriebszustand herzustellen, bevor die Spannung soweit abgesunken ist, daß ein lokaler Reset ausgelöst wird.

2. Spannungsüberwachung B:
   Sinkt die Versorgungsspannung unter die Schwelle von ca. 4,65 V, wird das stat. RAM auf Batteriepufferung geschaltet und es erfolgt ein lokaler Hardware-Reset.

2.3.3.1. Auslesen der NMI-Interruptquelle

Werden mehrere NMI-Quellen freigegeben, kann die Ursache eines NMI durch einen 16-Bit Lesezugriff auf die lokale I/O-Adresse 28h herausgefunden werden. Darin ist:

- Bit 9 des gelesenen Statuswortes = 0, wenn die Spannungsüberwachung den NMI ausgelöst hat.
- Bit 10 = 1, wenn der Watchdog-Timer einen NMI verursacht hat.

2.3.4. Anschluß einer externen Batterie (St1 Pin 17)

Zur Pufferung des statischen RAMs und der Uhr auf der "kleinen" MODULAR-4/486 Karte kann an Pin 17 (+) und 18 (GND) von St1 eine Batterie angeschlossen werden. Sie wird von der Karte automatisch abgeschaltet, sobald die 5 Volt Versorgungsspannung der Karte vorhanden ist. Es können Batterien mit 3,0 - 3,6 Volt eingesetzt werden.

Wenn keine Batterie angeschlossen werden soll, bleibt Pin 17 unbeschaltet.

2.3.5. Konfiguration der seriellen Schnittstelle A

! *Für den auf der Basiskarte und auf den Modulen M-COM-2 und M-COM-8 eingesetzten seriellen Baustein SCC 8530 bzw. ESCC 85230 steht ein separates Handbuch zur Verfügung.*

Der Baudratentakt der seriellen Schnittstelle A kann über die DTR/A-Leitung nach außen geführt werden (RS-232 Pegel). Genauer betrachtet wird das Signal am Pin "TRXC/A" des SCC-8530 verwendet. Per Software muß dafür gesorgt werden, daß an diesem Pin dann auch der gewünschte Takt geliefert wird (siehe Beschreibung des SCC-8530, TRXC/A kann das Ausgangssignal des Baudratengenerators, den Sendetakt oder das Ausgangssignal der SCC-internen DPLL liefern). Die DTR/A-Leitung steht dann nicht mehr als Modemsteuerleitung zur Verfügung. Wenn TRXC/A als Ausgang konfiguriert ist, liefert dieser Pin das Signal an DTR/A.

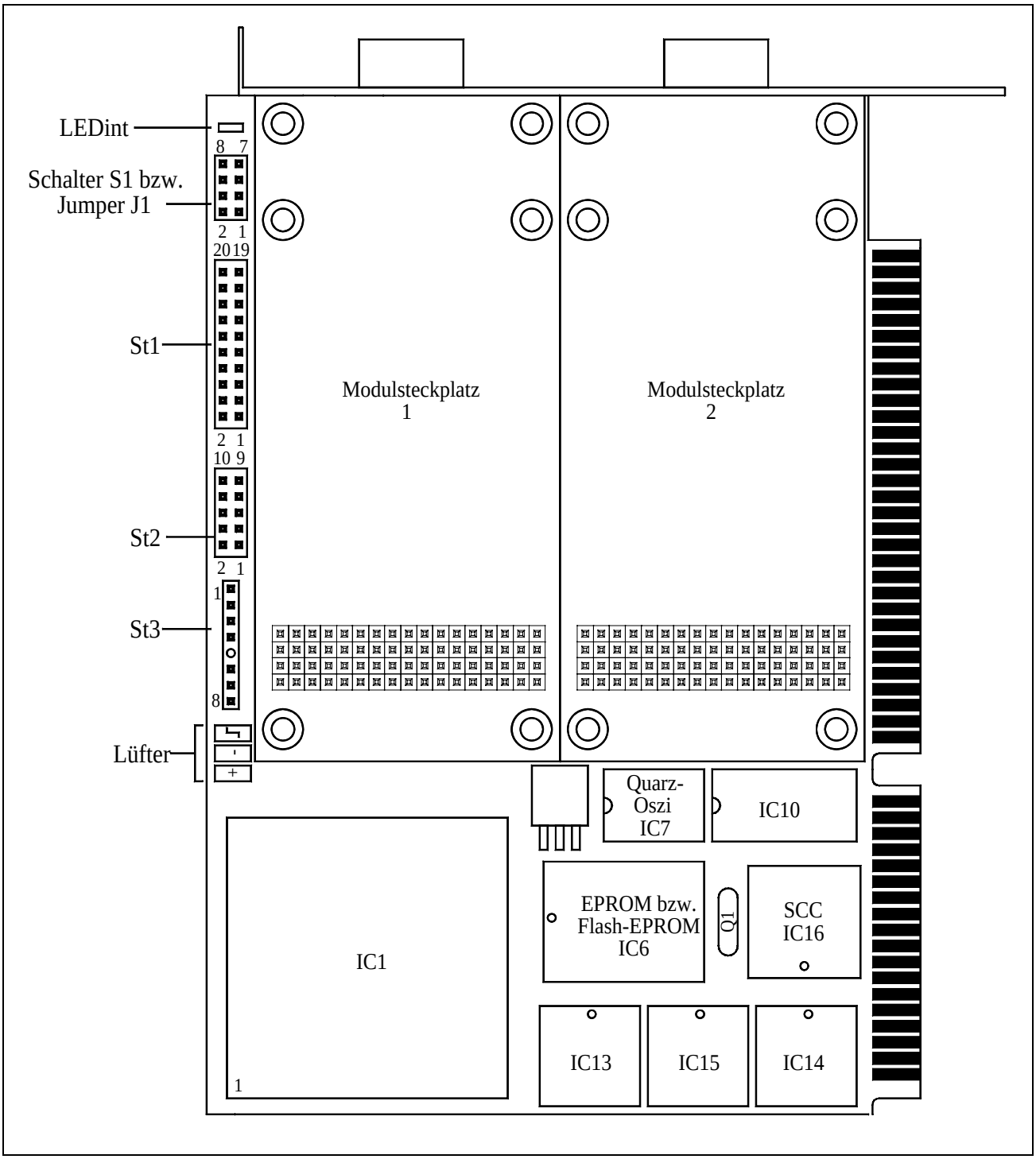
Der Baudratentakt für die Schnittstelle A wird immer über den Pin "RTXC/A" des SCC-8530 zur Verfügung gestellt. Das Signal an "RTXC/A" kann intern im SCC-8530 als Sendetakt, Empfangstakt, für die DPLL und für den Baudratengenerator verwendet werden. Dies ist per Software einstellbar. An den Pin "RTXC/A" kann entweder das Ausgangssignal eines Quarzoszillators (Teil der Schnittstelle B, standardmäßig 7,3728 MHz) oder ein externes Signal (über die Modem-Steuerleitung Ri/A) eingespeist werden. Die Ri/A-Leitung kann dann nicht als Modemsteuerleitung eingesetzt werden. Wenn TRXC/B als Ausgang konfiguriert ist, liefert dieser Pin den Takt an RTXC/A. Wenn er als Eingang geschaltet ist, wird der Takt über Ri/A geliefert.

Die Konfiguration der Schnittstelle erfolgt im EEPROM (siehe Anhang L).

2.3.6. Konfiguration des EPROM-Sockels

Es können verschiedene EPROMs oder EEPROMs (Flash) eingesetzt werden. Die Konfiguration erfolgt im EEPROM (siehe Anhang L).

2.3.7. Lageplan der "kleinen" MODULAR-4/486



2.4. Verwendung von I/O-Adressen im IBM-AT

Diese Liste ist nicht unbedingt vollständig und kann sich an einigen Stellen auch von PC-Hersteller zu PC-Hersteller unterscheiden. Zusätzlich eingesteckte Karten, z.B. Netzwerkkarten belegen evtl. andere, hier nicht aufgeführte I/O-Adressen.

Port (hex)	Verwendet für
00h - 1fh	DMA Controller 1 (8237)
20h - 3fh	Master Interrupt Controller 1 (8259)
40h - 5fh	Timer (8254)
60h - 6fh	Parallel I/O (8742)
70h - 7fh	Real Time Clock und DMA Maske
80h - 9fh	DMA Page Select Register
a0h - bfh	Interrupt Controller 2 (8259)
c0h - dfh	DMA Controller 2 (8237)
e0h - efh	Hardware Identification ROM
f0h - ffh	Co-Prozessor (f0h, f1h und f8h - ffh)
170h - 177h	Hard disk Controller (Secondary)
1f0h - 1f7h	Hard disk Controller (Primary)
200h - 20fh	Reserviert (game adapter)
278h - 27fh	Parallel Interface 2
2f0h - 2f7h	Reserviert
2f8h - 2ffh	Asynchrone Schnittstelle 2
300h - 31fh	Reserviert
338h - 33fh	45/60 MB Tape Drive
360h - 36fh	Reserviert
370h - 377h	Floppy Controller 2
378h - 37fh	Parallel Interface 1 (CGA)
380h - 38fh	DLC Interface und Bisynchron. Schnittstelle 2
3a0h - 3afh	Bisynchron. Schnittstelle 1
3b0h - 3bbh	Character Display Controller
3bch - 3bfh	Parallel Interface 1 (Hercules, MDA, VGA)
3c0h - 3cfh	Reserviert
3d0h - 3dfh	Graphik Display Controller
3f0h - 3f7h	Floppy Controller 1
3f8h - 3ffh	Asynchrone Schnittstelle 1

2.5. Verwendung der Hardware-Interrupts im IBM-AT (BIOS)

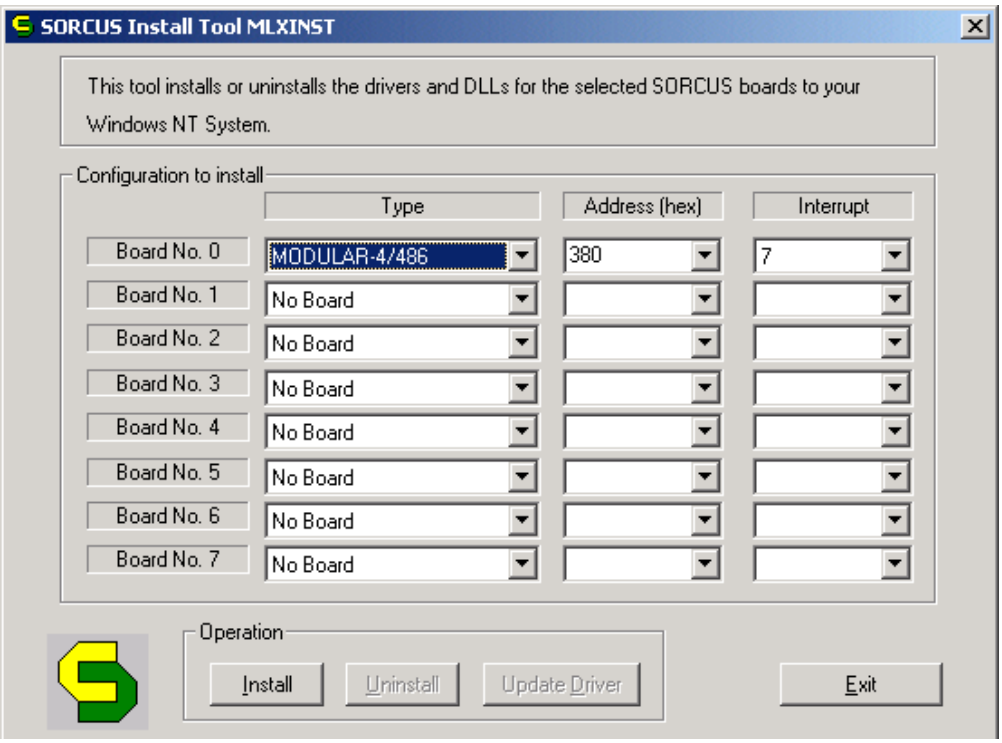
Diese Liste ist nicht unbedingt vollständig und kann sich an einigen Stellen auch von PC-Hersteller zu PC-Hersteller unterscheiden. Zusätzlich eingesteckte Karten, z.B. Netzwerkkarten belegen evtl. die freien Interrupteingänge.

IRQ-	INT #	Verwendet für
0	08h	Timer 0
1	09h	Keyboard
2	0ah	von Slave-Controller
3	0bh	Asynchrone Schnittstelle 1
4	0ch	Asynchrone Schnittstelle 2
5	0dh	Parallele Schnittstelle 2
6	0eh	Floppy Controller
7	0fh	Parallele Schnittstelle 1
8	70h	Echtzeituhr
9	71h	Umleitung auf INT 0AH
10	72h	Standardinterrupt
11	73h	Standardinterrupt
12	74h	Standardinterrupt
13	75h	Arithmetik Coprozessor, Umleitung auf INT 2H
14	76h	Festplatte
15	77h	Standardinterrupt

2.6. Treiberinstallation

2.6.1. Treiberinstallation unter Windows NT 4.0 und Windows 95

Bevor Sie die SORCUS-Karte unter Windows NT 4.0 bzw. 95 benutzen können, müssen Sie einen Treiber installieren. Entpacken Sie dazu die Datei NT4WIN95.ZIP und starten Sie das Programm MLXINST.EXE. Tragen Sie alle vorhandenen Karten ein und drücken Sie den *Install*-Button.



Beachten Sie dabei, daß die eingestellten Resourcen (Address, Interrupt) mit den Karteneinstellungen übereinstimmen und die Ressourcen nicht von anderen Baugruppen verwendet werden.

2.6.2. Treiberinstallation unter Windows 98, ME und 2000

Bevor Sie die SORCUS-Karte unter Windows 98, ME und 2000 benutzen können, müssen Sie einen Treiber installieren. Entpacken Sie dazu die Datei MLXWDM.ZIP und starten Sie das Programm MLXSETUP.EXE. Nachdem der Treiber installiert ist, können Sie mit dem „Hardware-Assistent“ in der Windows-Systemsteuerung SORCUS-Karten einfügen. Der Assistent zeigt Ihnen am Ende der Installation die Jumpereinstellungen bzw. EEPROM-Einträge an, die Sie auf der Karte einstellen müssen (Adresse, Interrupt).

2.7. Systemsteuerung

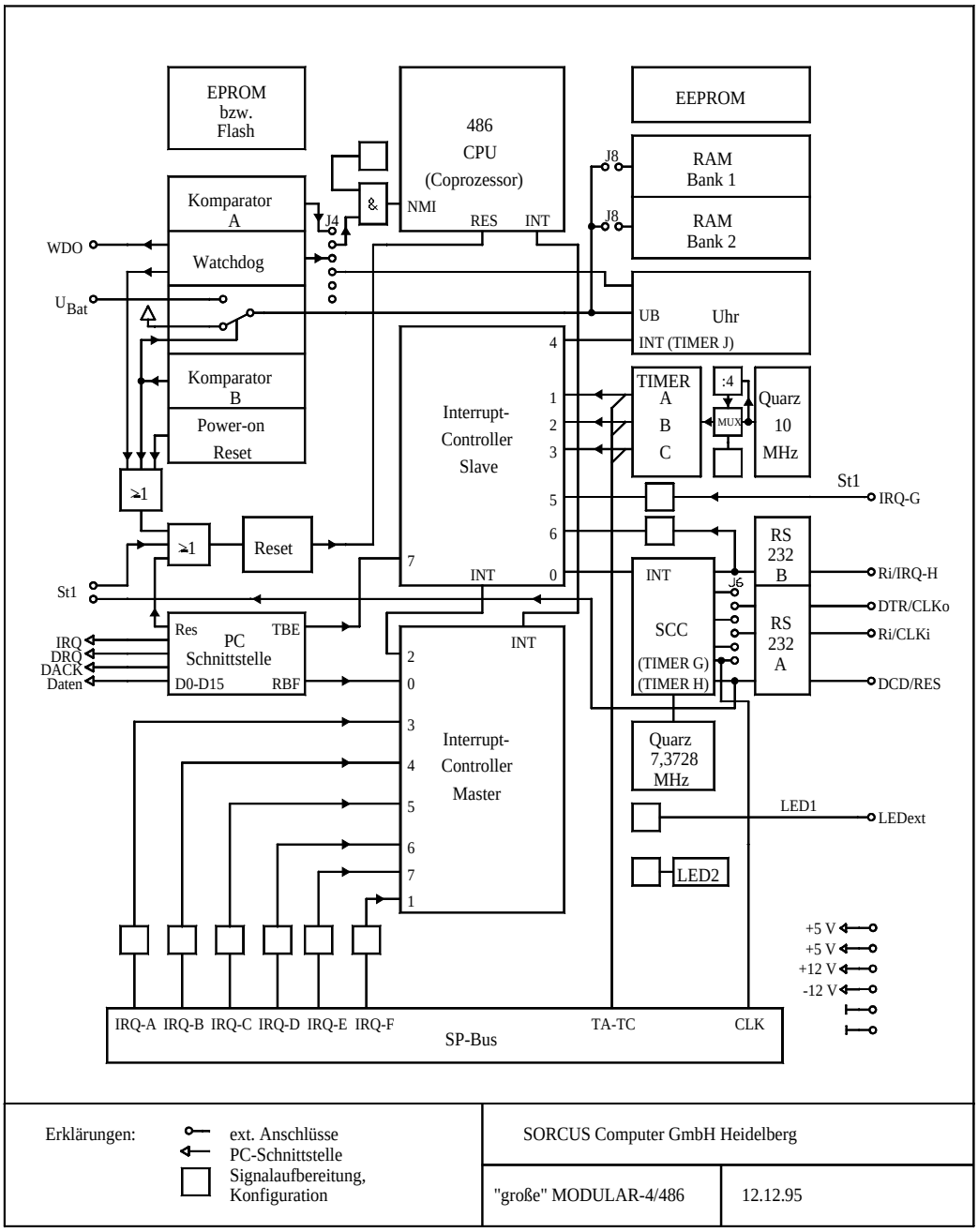
Nach der Treiberinstallation befindet sich in der Windows-Systemsteuerung das Symbol *SORCUS boards.* Der Startbildschirm zeigt eine Übersicht über alle installierten SORCUS-Karten. Mit diesem Programm können Eigenschaften der SORCUS-Karten angezeigt bzw. verändert werden. Außerdem können Remote-Verbindungen zu SORCUS-Karten erstellt werden.

Notizen

3. Funktionseinheiten der MODULAR-4

3.1. Funktionseinheiten der "großen" MODULAR-4/486

3.1.1. Blockschaltbild der "großen" MODULAR-4/486



Erklärungen: ext. Anschlüsse; PC-Schnittstelle; Signalaufbereitung, Konfiguration	SORCUS Computer GmbH Heidelberg	
	"große" MODULAR-4/486	12.12.95

3.1.2. Stecker und Buchsen der "großen" MODULAR-4/486

Einen Überblick über alle Stecker und Buchsen gibt folgende Tabelle und der Lageplan in Kapitel 2.

Stecker / Buchse	Typ (auf der Karte)	Funktion
St1	26-pol. Pfostenstecker[1] mit Wanne (2x13)	5 Volt Versorgung Eingang Spannungsüberwachung A Eingang Spannungsüberwachung A Ausgang Interrupt-Eingang IRQ-G Externe Batterie Eingang LED-Ausgänge Watchdog Ausgang Hardware-Reset Eingang Reset über DCD/A Serielle Schnittstelle A (alle Leitungen)[2] Serielle Schnittstelle B (TMT und RCV)[2]
St2 St2a[3]	9-pol. D-Sub. Stecker 10-pol. Buchsenleiste	Serielle Schnittstelle A Serielle Schnittstelle A
St3 St3a[3]	9-pol. D-Sub. Stecker 10-pol. Buchsenleiste	Serielle Schnittstelle B Serielle Schnittstelle B
J5	1x10 Pfostenstecker	Externe Batterie Ein- und Ausgang (Beschreibung siehe Kapitel 2)

[1] Der Stecker St1 ist bei den "großen" MODULAR-4/486 Rev. B Karten nur 14-polig

[2] Nicht bei "großer" MODULAR-4/486 Rev. B

[3] Die beiden Buchsenleisten St2a und St3a sind standardmäßig nicht bestückt, sie befinden sich unter St2 und St3.

3.1.3. Serielle Schnittstelle A (St2 und St2a), Serielle Schnittstelle B (St3 und St3a)

An St2 (9-pol. D-Submin. Stecker) ist die serielle Schnittstelle A, an St3 (9-pol. D-Submin. Stecker) die serielle Schnittstelle B der "großen" MODULAR-4/486 Karte herausgeführt. Die Belegung entspricht der Standardbelegung beim IBM-PC/AT. Bei beiden Schnittstellen sind alle Modem-Steuerleitungen vorhanden. Unter bestimmten Einbaubedingungen kann es sinnvoll sein, die beiden D-Sub. Stecker St2 und St3 nicht zu bestücken, um z.B. Flachbandkabel direkt aus dem PC herauszuführen. Hierzu ist ein spezielles Bracket mit Zugentlastung für die Flachbandkabel lieferbar. Für diesen Fall sind St2a und St3a vorgesehen. Dort werden dann sehr flache Buchsenleisten (je 2 x 10 pol.) für Flachbandkabel eingelötet. Der gesamte Aufbau ist dann so niedrig, daß die anderen Flachbandkabel darüber hinweggeführt werden können. Bei Rev. C sind die Schnittstellen auch am Stecker St1 herausgeführt (siehe dort).

Signal	St2, Pin (D-Sub. Stecker)	St2a, Pin (Buchsenleiste)
DCD/A, RESET-IN	1	1
Receive Data A	2	3
Transmit Data A	3	5
DTR/A oder CLKOUT/A	4	7
Ground	5	9
DSR/A	6	2
RTS/A	7	4
CTS/A	8	6
Ri/A oder CLKIN/A	9	8

Signal	St3, Pin (D-Sub. Stecker)	St3a, Pin (Buchsenleiste)
DCD/B	1	1
Receive Data B	2	3
Transmit Data B	3	5
DTR/B oder CLKOUT/B	4	7
Ground	5	9
DSR/B	6	2
RTS/B	7	4
CTS/B	8	6
Ri/B oder CLKIN/B	9	8

3.1.4. Stecker St1 der "großen" MODULAR-4/486

Dieser 26-pol. Stecker[1] (2 x 13 pol.) liefert verschiedene Signale bzw. stellt Eingänge zur Verfügung:

Pin	Eingang/Ausgang	Signal
1	Eingang	**Rev. C:** Pin muß unbeschaltet bleiben **Rev. B:** Spannungsüberwachung A, z.B. für Versorgungsspannung oder eine andere ext. Spannung
2	Ausgang	GND
3	Ausgang	+5 Volt
4	Ausgang	Invertierter (TTL-)Pegel des am DCD-Eingang der seriellen Schnittstelle A anliegenden Signals (kann zum Auslösen eines Hardware-Reset mit Pin 6 verbunden werden)
5	Ausgang	GND
6	Eingang	Hardware-Reset, TTL, int. pull-up-Widerstand[2]
7	Eingang	Interrupt IRQ-G, TTL, Flanke einstellbar
8	Ausgang	**Rev. C:** TTL-Ausgang für onboard-LED (Kathode), Anode an +5 Volt anschließen. **Rev. B:** Nicht angeschlossen
9	Ausgang	GND
10	Ein-/Ausgang	Batterie (direkt mit Pin 1 von J5 verbunden)
11	Ausgang	+5 Volt
12	Ausgang	TTL-Ausgang für externe LED (Kathode), Anode an +5 Volt anschließen.

[1] Bei Rev. B der "großen" MODULAR-4/486 Karte ist der Stecker nur 14 polig.

[2] Reset wird durch den Übergang vom Low- in den High-Pegel ausgelöst.

Pin	Eingang/Ausgang	Signal
13	Ausgang	Watchdog-Timer Ausgang, active low[1]
14	Ausgang	Spannungsüberwachung A, Ausgang
15[2]	Eingang	DCD/A, (=Pin 1 von St2)
16[2]	Eingang	DSR/A (=Pin 6 von St2)
17[2]	Eingang	Receive Data A (=Pin 2 von St2)
18[2]	Ausgang	RTS/A (=Pin 7 von St2)
19[2]	Ausgang	Transmit Data A (=Pin 3 von St2)
20[2]	Eingang	CTS/A (=Pin 8 von St2)
21[2]	Ausgang	DTR/A, (=Pin 4 von St2)
22[2]	Eingang	Ri/A (=Pin 9 von St2)
23[2]	Ausgang	GND
24[2]	Eingang	Receive Data B (=Pin 2 von St3)
25[2]	Ausgang	Transmit Data B (=Pin 3 von St3)
26[2]	Ausgang	GND

3.1.5. Eingang für Hardware-Reset der Karte (St1, Pin 6, 5, 4)

An den Eingang kann z.B. ein Taster (einpolig schließend, zwischen Pin 5 und 6) angeschlossen werden, um die Karte manuell zurückzusetzen. Alternativ ist es möglich, einen Hardware-Reset der Karte über den DCD-Eingang der seriellen Schnittstelle A auszulösen, wenn Pin 4 und 6 von St1 verbunden werden, z.B. per Jumper. In diesem Fall kann das DCD-Signal dann nicht mehr für die Schnittstelle A verwendet werden.

---

[1] I_{OUTMAX} bei log. 0 (<0,4 V): -2,7 mA
I_{OUTMAX} bei log. 1 (>3,5 V): 0,5 mA
Achtung: Externe CMOS-Logik nur über 4,7 kΩ Strombegrenzungswiderstand anschließen

[2] Bei Rev. B der "großen" MODULAR-4/486 nicht vorhanden

Wenn der Eingang (Pin 6 von St1) nicht verwendet werden soll, kann er unbeschaltet bleiben, er verfügt über einen internen Pull-up-Widerstand von 4,7 KOhm an + 5 Volt.

3.1.6. LED1 (St1, Pin 11 und 12) und LED2[1] (St1, Pin 8)

Hier können LEDs für Kontrollzwecke angeschlossen werden (max. Ausgangsstrom bei log. 0: 16 mA). LED1 wird als 'externe LED' bezeichnet, LED2[1] ist parallel zur Leuchtdiode auf der MODULAR-4 Karte (on-board LED) geschaltet.

3.1.7. Watchdog-Timer Ausgang (St1, Pin 13)

Soll das Ablaufen des Watchdog-Timers (siehe hierzu auch Kapitel 2) einen Hardware-Reset der MODULAR-4/486 Karte (Rev. C) auslösen, besteht die Möglichkeit, an Stecker St1, Pin 13 (Watchdog-Ausgang) mit Pin 15 (DCD-Eingang) und Pin 6 (Reset-Eingang) mit Pin 4 (inv. Pegel des Signals am DCD-Eingang) zu verbinden. Bei MODULAR-4/486 Karten Rev. B wird bei Ablaufen des Watchdog-Timers immer ein Reset der Karte durchgeführt.

Der Watchdog-Timer Ausgang liefert TTL-Pegel, der Ausgangstrom bei log. 0 (U< 0,4 V) beträgt max. 1,6 mA, bei log. 1 (U > 3,5 V) 1 µA.

Bedingung	Watchdog Ausgang
Versorgungsspannung der Karte unter 4,4 Volt	log. 1
Watchdog-Timer nicht aktiviert (siehe auch Beschreibung von J4)	log. 1
Watchdog Timer aktiviert:	
a) wird rechtzeitig nachgetriggert	log. 1
b) wird nicht rechtzeitig nachgetriggert	log. 0

[1] Bei Rev. B der "großen" MODULAR-4/486 nicht vorhanden.

3.1.8. Spannungsüberwachung A (Eingang: St1, Pin 1, Ausgang: St1, Pin 14)

Bei Karten der Rev. B besteht die Möglichkeit, die Versorgungsspannung der Karte oder eine extern eingespeiste Spannung zu überwachen. Die Spannung wird an Pin 1 von St1 angelegt. Zur Überwachung der Versorgungsspannung kann Pin 1 mit Pin 3 (= 5 Volt Versorgungsspannung) verbunden werden. Bei Karten der Revision C wird immer die Versorgungsspannung überwacht, Pin 1 muß unbeschaltet bleiben (siehe hierzu auch Kapitel 2).

Wenn nun die Versorgungsspannung unter eine bestimmte Schwelle[1] (4,8 Volt bei Rev. C) fällt, wird ein Nicht-Maskierbarer-Interrupt (NMI) auf der Karte ausgelöst, sofern dieser aktiviert wurde.

Die Schaltschwelle für die Auslösung eines NMI liegt geringfügig höher als die für die Umschaltung des RAM auf Batteriepufferung (4,65 V). Mit der Umschaltung auf Batteriepufferung ist das RAM dann auch nicht mehr vom Prozessor zugreifbar. Dadurch ist gewährleistet, daß der NMI zeitlich vor der Abschaltung des RAM erfolgt, so daß die CPU noch Zeit hat, wichtige Daten in das RAM zu retten.

3.1.9. Reset-Verhalten

Ein durch den Watchdog-Timer oder die Spannungsversorgung verursachter lokaler Reset bewirkt ebenso wie ein vom PC gesendetes Reset-Signal den Abbruch aller

---

[1] Bei "großer" MODULAR-4/486, Rev. B ist die Schaltschwelle ist mit einem Potentiometer einstellbar (4,4 bis 5,8 Volt, mit Vorwiderstand auch höhere Spannungen möglich)

Einstellen der Ansprechschwelle der Spannungsüberw. A bei "großer" MODULAR-4/486, Rev. B:

1. Ein Labornetzteil an Pin 2 (Minuspol) und Pin 1 (Pluspol) von St1 anschließen **(immer zuerst Pin 2 anklemmen und dann erst Pin 1 und immer zuerst Pin 1 wieder abklemmen und dann Pin 2!***)*.
2. Das Labornetzteil auf ca. 6 Volt einstellen und einschalten.
3. Während des Einstellvorgangs wird die Spannung an Pin 14 von St1 mit einem hochohmigen Meßgerät (Ri > 1 MOhm) beobachtet.
4. Die Spannung an Pin 14 muß jetzt größer 3 Volt sein.
5. Die Spannung des Labornetzteils wird nun per Hand langsam vermindert (nicht unter 0 Volt!), bis die Spannung an Pin 14 auf ca. 0 Volt springt.
6. Wenn die Spannung des Labornetzteils jetzt der gewünschten Ansprechschwelle von ca. 4,7 Volt entspricht, ist der Einstellvorgang beendet (immer zuerst Pin 1 von St1 wieder abklemmen, dann erst Pin 2!). Wenn nicht, muß Poti P1 verändert werden und der Vorgang ab Punkt 2. wiederholt werden.

Programme. Es kann eingestellt werden, wie sich die Karte im Anschluß daran verhalten soll:

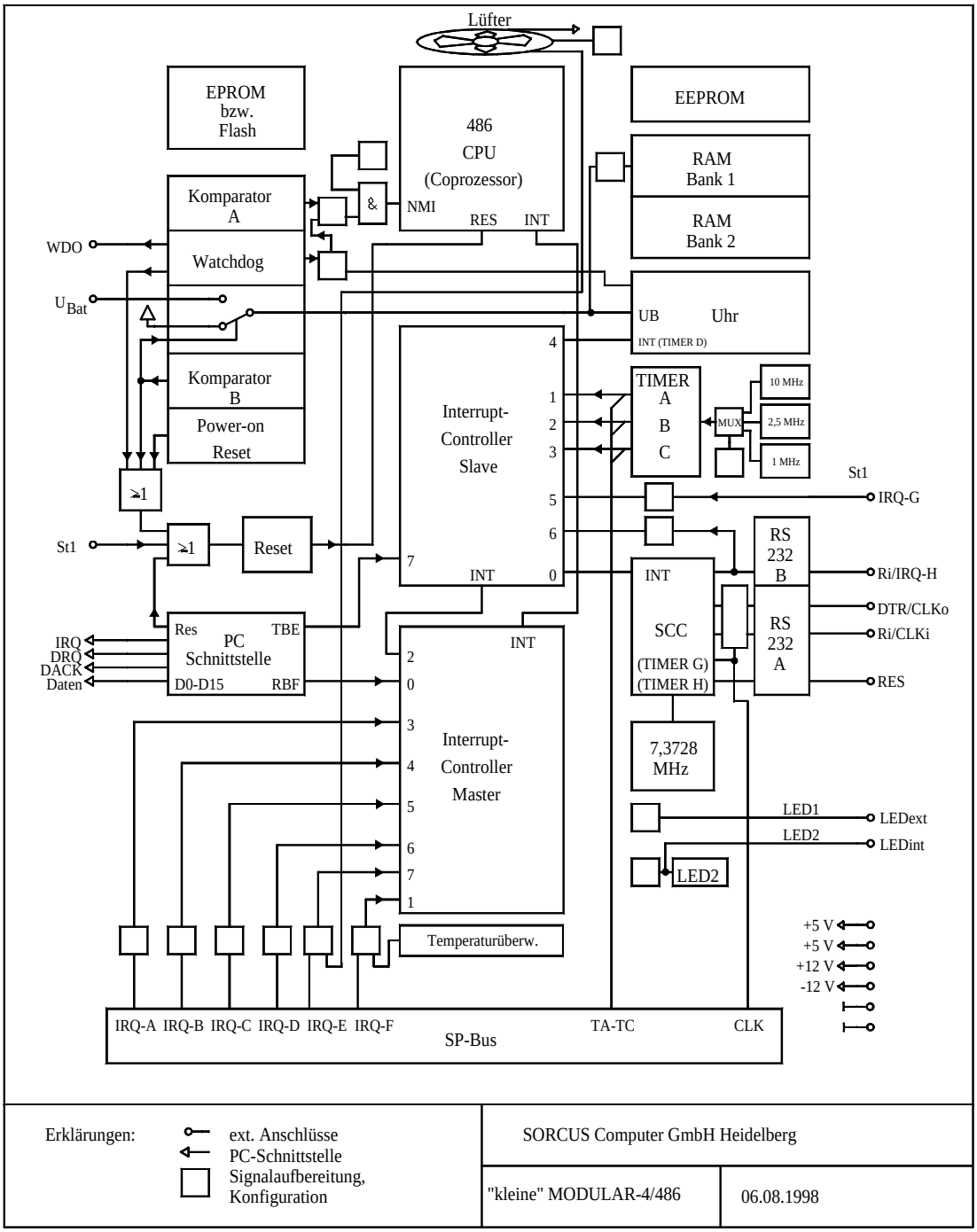
- Aktivierung des Mini-OS, welches nur einen sehr eingeschränkten Funktionsumfang beinhaltet. Es erfolgt z.B. keine Initialisierung der "großen" MODULAR-4/486 und der Module.
- Durch Verbinden der Pins 7 und 8 an Stecker St1 (per Jumper bei Rev. C) wird nach einem Reset automatisch das ROM-OsX aktiviert. Die MODULAR-4/486 Karte und die Module werden entsprechend den Einstellungen im EEPROM initialisiert.
  Bei MODULAR-4/486 (Rev. B) muß eine Drahtbrücke zwischen IC39, Pin 11 und Stecker St1, Pin 8 gelötet werden. Danach hat ein Jumper auf St1, Pin7 und 8 diesselbe Funktion wie bei Rev. C.

3.1.10. Batterieanschluß (St1, Pin 10)

Auf der Karte ist standardmäßig keine Batterie zur Versorgung der on-board Uhr und zur Pufferung von statischem RAM vorgesehen. Es kann aber eine externe Batterie angeschlossen werden. Hierfür ist Stecker bzw. Jumperfeld J5 vorgesehen, wie in Kapitel 2 beschrieben. Eine Batterie kann aber auch über St1, Pin 10 (Pluspol) und St1, Pin 9 (Minuspol) angeschlossen werden. Pin 1 und Pin 4 von J5 bleiben dann frei, Pin 5 und Pin 6 von J5 müssen per Jumper verbunden werden (siehe Kapitel 2.2.5., Anschluß einer 3V Batterie).

3.2. Funktionseinheiten der "kleinen" MODULAR-4/486

3.2.1. Blockschaltbild der "kleinen" MODULAR-4/486



Erklärungen: ext. Anschlüsse; PC-Schnittstelle; Signalaufbereitung, Konfiguration	SORCUS Computer GmbH Heidelberg	
	"kleine" MODULAR-4/486	06.08.1998

3.2.2. Stecker und Buchsen der "kleinen" MODULAR-4/486

Einen Überblick über alle Stecker gibt folgende Tabelle und der Lageplan in Kapitel 2.

Stecker / Buchse	Typ (auf der Karte)	Funktion
St1	20-pol. Steckerleiste (2x10)	5 Volt Versorgung Eingang/Ausgang ±12 Volt Versorgung Eingang/Ausgang Interrupt-Eingang IRQ-G Eingang für externe Batterie 2 LED-Ausgänge Hardware-Reset Eingang Watchdog Ausgang (/WDO) Serielle Schnittstelle B (RS-232)
St2	10-pol. Pfostenstecker mit Wanne (2x5)	Serielle Schnittstelle A (RS-232)
St3	8-pol. Pfostenstecker (1x8)	Diagnose-System (nicht für Anwendungen verwendbar)

3.2.3. Serielle Schnittstelle A (St2)

An St2 ist die serielle Schnittstelle A (RS-232) der "kleinen" MODULAR-4/486 Basiskarte herausgeführt. Die Schnittstelle verfügt über alle Modem-Steuerleitungen.

Schnittstelle A:

Signal (RS-232)	**St2, Pin (Buchsenleiste)**	**Standardbelegung (IBM-AT) 9-pol. D-Submin-Stecker**
DCD/A	1	1
DSR/A	2	6
RCV/A (Receive Data)	3	2
RTS/A	4	7
TMT (Transmit Data)	5	3
CTS/A	6	8
DTR/A oder CLKOUT/A	7	4
Ri/A oder CLKIN/A	8	9
Ground	9	5
-	10	-

3.2.4. Stecker St1 der "kleinen" MODULAR-4/486

Dieser 20-pol. Stecker (2 x 10 pol.) liefert verschiedene Signale bzw. stellt Eingänge zur Verfügung:

Pin	Eingang/Ausgang	Signal
1	Eingang	DCD/B, RS-232 Pegel
2	Eingang	DSR/B, RS-232 Pegel
3	Eingang	RCV/B, RS-232 Pegel
4	Ausgang	RTS/B, RS-232 Pegel
5	Ausgang	TMT/B, RS-232 Pegel
6	Eingang	CTS/B, RS-232 Pegel
7	Ausgang	DTR/B, RS-232 Pegel
8	Eingang	RI/B, RS-232 Pegel
9	Ein-/Ausgang	GND
10	Ausgang	LED1=LEDext: Ausgang für externe LED (Kathode), Anode an +5 Volt anschließen.
11	Ausgang	Watchdog-Ausgang, TTL-Pegel, active low
12	Eingang	Hardware-Reset-Eingang, active low, int. Pull-up-Widerstand
13	Ein-/Ausgang	+5 Volt
14	Ausgang	LED2=LEDint: Ausgang der on-board LED
15	Ein-/Ausgang	+5 Volt
16	Eingang	Interrupt IRQ-G, TTL, aktive Flanke einstellbar
17	Eingang	Pluspol Batterie 3,6 V (Minuspol an GND)
18	Ein-/Ausgang	GND
19	Ein-/Ausgang	-12 Volt
20	Ein-/Ausgang	+12 Volt

3.2.5. Eingang für Hardware-Reset der Karte (St1, Pin 12)

An den Eingang kann z.B. ein Taster (einpolig schließend, zwischen Pin 12 und 9 oder 18) angeschlossen werden, um die Karte manuell zurückzusetzen.

Wenn der Eingang nicht verwendet werden soll, kann er unbeschaltet bleiben, er verfügt über einen internen Pull-up-Widerstand.

Der Eingang ist ±25 V überspannungsfest.

3.2.6. Externe Leuchtdiode, LED1=LEDext (St1, Pin 10)

Hier kann eine LED für Kontrollzwecke angeschlossen werden (Anode an +5 V, max. Ausgangsstrom bei log. 0: 16 mA). Ein interner Vorwiderstand von 270 Ω dient zur Strombegrenzung.

3.2.7. On-board Leuchtdiode, LED2=LEDint (St1, Pin 14)

Hier kann eine LED parallel zur on-board LED für Kontrollzwecke angeschlossen werden (Anode an +5 V, max. Ausgangsstrom bei log. 0: 10 mA). Ein interner Vorwiderstand von 270 Ω dient zur Strombegrenzung.

3.2.8. Batterieanschluß (St1, Pin 17)

Auf der Karte ist standardmäßig keine Batterie zur Versorgung der on-board Uhr und zur Pufferung von statischem RAM vorgesehen. Es kann aber eine externe Batterie mit einer Spannung von 3,0 bis 3,6 Volt angeschlossen werden: Pluspol an Pin 17 von St1, Minuspol an Pin 9 oder 18 von St1. Wenn keine Batterie angeschlossen werden soll, bleibt Pin 17 unbeschaltet.

3.2.9. Watchdog-Timer Ausgang (St1, Pin 11)

Soll das Ablaufen des Watchdog-Timers (siehe hierzu auch Kapitel 2) einen Hardware-Reset der "kleinen" MODULAR-4/486 Karte auslösen, besteht die Möglichkeit, an Stecker St1 die Pins 11 (Watchdog-Ausgang) und 12 (Reset-Eingang) per Jumper zu verbinden.

3.2.10. Reset-Verhalten

Ein durch den Watchdog-Timer oder die Spannungsversorgung verursachter lokaler Reset bewirkt ebenso wie ein vom PC gesendetes Reset-Signal den Abbruch aller Programme. Es kann eingestellt werden, wie sich die Karte im Anschluß daran verhalten soll:

- Aktivierung des Mini-OS, welches nur einen sehr eingeschränkten Funktionsumfang beinhaltet. Es erfolgt z.B. keine Initialisierung der Basiskarte und der Module.
- Durch Verbinden der Pins 14 und 16 an Stecker St1 per Jumper wird nach einem Reset automatisch das ROM-OsX aktiviert. Die MODULAR-4/486 Karte und die Module werden entsprechend den Einstellungen im EEPROM initialisiert.

Notizen

4. Das Karten-Manager-Programm SNW32

4.1. Aufgabe

Das Programm „Schöne Neue Welt 32“ (SNW32) dient zum Steuern, Konfigurieren und Testen von SORCUS Karten und Modulen vom Host-PC aus. Es unterstützt alle Multi-LAB/2, Multi-COM, MODULAR-4/486, und MAX-Trägerkarten und MAX-Module. Das Programm arbeitet unter allen WIN32 Betriebssystemen.

4.2. Installation

Das Programm wird über ein Installationsprogramm auf Ihrem Rechner installiert. Beachten Sie, daß Sie auch einen entsprechenden Treiber für Ihre SORCUS Karte installiert haben (Siehe Kapitel Installation)!

4.3. Aufbau

Eine klar strukturierte Windows Bedienoberfläche zeigt die einzelnen Komponenten des SORCUS Systems in einer übersichtlichen Baumstruktur an. Über diese erhält der Anwender Zugriff auf sämtliche zur Verfügung stehenden Funktionseinheiten der jeweils installierten Karte(n). Neben dem Zugriff auf die Hardwarekomponenten wie Timer, Interrupt-Controller oder Speicher bietet das Programm auch die Möglichkeit zur Installation und Verwaltung von Softwarekomponenten auf der Karte. Ein Taskmanager kann beispielsweise alle auf der Karte installierten Echtzeitprogramme sowie deren Parameter anzeigen. Über Kommandozeilenparameter ist es auch möglich, mit Hilfe von Installationsdateien Echtzeitprogramme zu installieren, ohne SNW32 starten zu müssen.

4.4. Assistenten

Je Funktionseinheit steht dem Benutzer ein Assistent zur Verfügung, der einen komfortablen Zugriff auf das angewählte Device ermöglicht, und zwar sowohl für die Komponenten der MODULAR-4-Basiskarte wie auch für die aufgesteckten SPB-Module. Damit ist es z.B. möglich, eine Konfiguration sowie Ein- oder Ausgaben durchzuführen. Die Assistenten liegen in Form von 32-Bit DLLs vor und können auch problemlos in andere Windows-Applikationen eingebunden werden.

4.5. Installations-Dateien

Die SORCUS Karten verfügen über ein eigenes Echtzeit Multi-Tasking-Betriebssystem OsX. Eine zentrale Aufgabe im Umgang mit den Karten ist es, Echtzeit-Programme als Tasks zu installieren. Die einfachste Methode, Programme zu installieren, Parameter zu setzen und Prozeduren zu starten, ist die Verwendung von Installationsdateien. Installationsdateien sind Textdateien und bestehen aus einer Folge von Anweisungen, die die oben genannten Aktionen ausführen. Dafür sind einige Schlüsselwörter definiert, die in den Hilfetexten und im Anhang M ausführlich erklärt sind. Installationsdateien erhalten standardmäßig die Extension '.INS'. Das Programm enthält einen komfortablen Editor, mit dem solche INS-Dateien erstellt und getestet werden können. Um INS-Dateien auszuführen, kann SNW32 auch mit Komandozeilenparametern aufgerufen werden.

4.6. Flash Unterstützung

Das auf den SORCUS Karten vorhandene Flash-EEPROM kann neben dem Betriebssystem OsX auch Anwenderprogramme enthalten. Diese können nach einem Power-On oder nach einem Reset der Karte automatisch installiert werden. Der Flash-Assistent von SNW32 übernimmt dabei die gesamte Programmierung des Flash, so daß der Anwender keinen eigenen Programmieraufwand mehr hat.

4.7. Remote Verbindung

Neben einer lokal im Rechner eingebauten Karte kann SNW32 auch auf Remote-Karten zugreifen. Die Remote-Verbindung wird von einem Assistenten innerhalb des Programms vorgenommen und ermöglicht einen Zugriff auf seriell oder über ein Netzwerk angekoppelte Karten.

4.8. Channel Manager

Der Zugriff auf Komponenten der Karten in eigenen PC- bzw. Echtzeit-Programmen erfolgt über sog. Modul-Device-Treiber Kanäle (siehe auch Anhang O). Der SORCUS Channel-Manager (CHM) dient zur Erstellung und Verwaltung von Modul-Device-Treiber (MDD) Kanälen. Der CHM ermöglicht es, ein komplettes SORCUS System, bestehend aus Karten, Modulen und MDD-Kanälen über eine komfortable Windows-Oberfläche zu konfigurieren. Die erstellte Konfiguration kann abgespeichert und mit Hilfe der SORCUS Bibliotheken in eigene Applikationen eingebunden werden. Hierzu wird jedem Modul-Device-Treiber Kanal ein eindeutiger

Name zugewiesen, der es ermöglicht, den Kanal zu öffnen. Die erstellte Konfiguration muß dabei nicht den tatsächlich vorhandenen Komponenten auf dem Rechner entsprechen, d.h. es kann z.B. eine MODULAR-4/486 Karte konfiguriert werden, obwohl diese nicht im Rechner vorhanden ist. So ist es z.B. möglich, mit einer komfortablen Windows-Oberfläche eine komplexe Konfiguration zu erstellen und abzuspeichern und die Datei anschließend auf einen DOS-Rechner zu übertragen und dort mit der DOS-Bibliothek zu nutzen.

4.9. Hotline File

Um der SORCUS-Hotline die Suche nach möglichen Fehlern in Ihrem System zu erleichtern, stellt das Programm SNW32 die Möglichkeit zur Verfügung, eine sog. Hotline-Datei zu erstellen. Diese Text-Datei enthält verschiedene Informationen, die die SORCUS Karten betreffen.

4.10. Hilfe

Das Programm bietet eine Online-Hilfe, die die Bedienung der einzelnen Komponenten erläutert.

Notizen

5. Software

PC-Zusatzkarten von SORCUS sind 'intelligent', d.h. sie sind eigenständige Computer, die durch ihre verschiedenen Schnittstellen und ihre 'on-board' Software für eine Vielzahl von Meß-, Steuer-, Regelungs- und Kommunikationsaufgaben eingesetzt werden können. Durch ihre freie Programmierbarkeit und durch die aufsteckbaren SPB-Module bieten MODULAR-4 Karten eine hohe Anpassungsfähigkeit. Hinzu kommt die sehr hohe Verarbeitungsgeschwindigkeit der 486-CPU, die mit bis zu 40 MHz extern getaktet wird (intern dann mit 33, 66, 100 oder 133 MHz, je nach Version) und das on-board Cache RAM.

Dieser hohen Leistungsfähigkeit der Hardware ist auch die Software auf der Karte angepaßt: Ein eigenes Echtzeit-Multi-Tasking-Betriebssystem, das auf der Karte läuft, verwaltet alle Meß-, Steuerungs- und Kommunikationsprogramme und auch die Kommunikation mit dem Hostrechner, also dem PC. Die Karten können mit minimaler Änderung der Software auch als 'Stand-alone'-Systeme eingesetzt werden. In diesem Fall kann die Kommunikation mit einem (auch räumlich weiter entfernten) Hostrechner über die serielle Schnittstelle der Basiskarte oder über eine andere Schnittstelle auf den Modulen erfolgen, z.B. über IEC-Bus, CAN-Bus, RS-232, RS-422, RS-485 oder Lichtwellenleiter. Die Beschreibung in diesem Kapitel bezieht sich auf den üblichen Einsatz in einem PC.

Der PC kommuniziert mit dem Betriebssystem der Karte auf unterster Ebene in einer Makrobefehlssprache. Darauf aufgebaut sind PC-Bibliotheken, die es ermöglichen, die Karte bequem in Hochsprachen anzusprechen. Diese PC-Bibliotheken sind in Kapitel 6 und Anhang O beschrieben, die zugrundeliegenden Makrobefehle in Kapitel 12.

Das Betriebssystem unterstützt alle Schnittstellen und Funktionseinheiten der Karte und ermöglicht dadurch, die Karte komplett vom PC aus zu betreiben. Für Anwendungen, bei denen es nicht auf hohe Geschwindigkeit ankommt, kann allein mit diesen Befehlen die Karte durchaus erfolgreich eingesetzt werden.

Die Möglichkeiten der Karte als Meß-, Steuerungs- und Kommunikationssystem kommen aber erst bei Ausnutzung der Multi-Tasking-Fähigkeiten des Betriebssystems voll zum Tragen. Damit ist es möglich, die gesamte Echtzeitprogrammierung auf die Karte zu verlagern. Der PC muß dann lediglich noch die Ansteuerung und Parameterübergabe vornehmen.

Das Echtzeit-Multi-Tasking-Betriebssystem unterstützt 1024 Tasks (task, engl. = Aufgabe, Arbeit), die interruptgesteuert (sog. I-Task) oder Nicht-Interrupt-gesteuert

(sog. NI-Task) sein können. Jeder Task kann ein Anwendungsprogramm zugeordnet werden. Diese Zuordnung wird im folgenden als "Installieren" bezeichnet. Es gibt viele Anwendungsprogramme, die mehr als eine Task benötigen und die deshalb aus mehreren Teilprogrammen bestehen. Jedes dieser Teilprogramme muß dann auch unter einer eigenen Task installiert werden. Die Anwendungsprogramme werden vom PC aus ins RAM der Karte geladen oder aus Programmen im ROM ausgewählt. Hierzu steht das Hilfsprogramm SNW bzw. SNW32 zur Verfügung, mit dem die Karte z.B. beim Starten des Systems automatisch wie gewünscht konfiguriert werden kann. Daneben bieten die PC-Bibliotheken die Möglichkeit, Anwendungsprogramme aus dem eigenen PC-Programm heraus auf die Karte zu laden und zu installieren.

5.1. Programmierebenen

5.1.1. Verwendung fertiger Softwarepakete

Der Anwender braucht keinerlei eigene Programmierarbeit zu leisten. Auf dem Markt erhältlich sind fertige Programmpakete für Meßdatenerfassung und Auswertung, Prüfstandssteuerung, Regelung, etc. Sie enthalten spezielle Treiber für die MODULAR-4 Karte und nutzen die Fähigkeiten der Karte optimal aus, z.B.:

Produkt	Hersteller	Lieferant
ARGUS	SORCUS	SORCUS
DIA/DAGO	NI resp. GfS, Aachen	National Instruments
DIA*dem*	NI resp GfS, Aachen	National Instruments
PDES	GIF, Aachen	GIF
MEVAS	R.I.A.S., Bottrop	R.I.A.S.
DASY*Lab*	NI resp. DASYTEC	National Instruments
LabVIEW	National Instruments (NI)	National Instruments

5.1.2. Verwendung fertiger Echtzeitprogramme

In diesem Fall wird die gesamte Echtzeitprogrammierung durch die für die Karte vorhandenen Programme zur Verfügung gestellt (vorausgesetzt, die Programme dekken die Problemstellung ab). Somit beschränkt sich der noch zu leistende Programmieraufwand auf die Ansteuerung der Karte. Dies ist im allgemeinen nicht mehr zeitkritisch, so daß es keine große Rolle spielt, in welcher Programmiersprache die PC-Programme geschrieben sind.

Ein typisches Beispiel für diesen Anwendungsfall ist die Verwendung der Kommunikationsprogramme (CQ7 bzw. CQ8) bzw. fertiger Protokolltasks, die auf der MODULAR-4 Karte laufen.

Da ständig neue Echtzeit-Anwendungsprogramme entwickelt werden, empfehlen wir Ihnen, sich über den aktuellen Stand bei Ihrem Händler oder direkt bei SORCUS zu informieren, wenn das für Ihren Anwendungsfall erforderliche Programm nicht bereits auf einem Datenträger mitgeliefert wurde.

5.1.3. Entwickeln eigener Echtzeitprogramme

Hierfür sind in Kapitel 7 Beispiele in Borland Pascal und Borland C angegeben. Für darüber hinausgehende Informationen, z.B. zu den auf der Karte eingesetzten ICs und ihrer Programmierung, wird auf die allgemein erhältliche Literatur verwiesen. In den Kapiteln 7 bis 10 wird ausführlich auf die Programmierung von eigenen on-board Echtzeitprogrammen eingegangen.

Außerdem wird auf die Seminare, die SORCUS hierzu anbietet, hingewiesen.

5.2. Das Multi-Tasking Betriebssystem "OsX"

Jede MODULAR-4 Karte enthält ein eigenes Betriebssystem im ROM (EPROM oder Flash-EPROM), das alle Schnittstellen und Funktionseinheiten der Karte unterstützt. Es wird vom PC aus über Makrobefehle und von Tasks auf der Karte über System-Subroutinen angesprochen. Die Makrobefehle und System-Subroutinen liegen den Bibliotheken zugrunde, mit denen Sie das Betriebssystem direkt in einer Hochsprache benutzen können.

5.2.1. Das Prinzip

Das Echtzeit-Multi-Tasking-Betriebssystem OsX der MODULAR-4 Karte geht von einem objektorientierten Ansatz aus. Es wurde bereits 1986 von SORCUS Computer für Z80 Systeme und 1990 für Intel i486 Prozessoren entwickelt und hat bis heute in Tausenden von Anwendungen seine hohe Effizienz und Benutzerfreundlichkeit unter Beweis gestellt. Es erlaubt dem Anwender, mehrere unabhängige Programme (theoretisch bis zu 1024) gleichzeitig auf einer MODULAR-4 Karte laufen zu lassen. Zur Zeit können Programme interruptgesteuert (DI-Task oder II-Task), Timer-initiiert (TI-Task) und Nicht-Interrupt-gesteuert (NI-Task) sein.

Um ein Programm auf der MODULAR-4 Karte laufen zu lassen, muß es zunächst im Speicher der Karte vorhanden sein. Dann muß das Programm unter einer Task installiert werden, z.B. mit einem Bibliotheksaufruf vom PC aus. Das Programm läuft aber erst dann los, wenn die Task aktiviert wird bzw. der zugehörige Interrupt demaskiert wird. Auch dies kann vom PC aus gesteuert werden. Bereits nach dem Installieren besitzt die Task folgende Strukturen (siehe auch Kapitel 7.3.):

Programm (bestehend aus Prozeduren)

Datenbereich

Parameterbereich

Die Wahl der Tasknummer bleibt dem Benutzer überlassen, sie hat keinen Einfluß auf die Priorität, mit der die Programme auf der Karte bearbeitet werden (siehe unten). Lediglich die Tasknummern 0 bis 20 sind derzeit für das Betriebssystem (Task 0) und die Modul-Device-Treiber reserviert (siehe auch Anhang O).

5.2.2. Die Echtzeitprogramme

Ein Echtzeitprogramm besteht aus einer Anzahl von Prozeduren (im folgenden wird einfach der Begriff Prozedur für eine Subroutine verwendet, unabhängig von der Zahl der hin- oder zurückgegebenen Parameter). Dabei wird unterschieden zwischen

Haupt-Prozedur,

Auto-Init-Prozedur und

Globalen Prozeduren.

Die **Haupt-Prozedur** ist die Prozedur, die vom Betriebssystem aufgerufen wird, wenn die Task an der Reihe ist. Sie stellt also das eigentliche Programm der Task dar.

Beispiel: Ein Programm zur Meßdatenerfassung wird unter einer interruptgesteuerten Task installiert. Der Interrupt wird durch einen Timer ausgelöst, der den Abtast-Trigger darstellt. In der Haupt-Prozedur, die bei jedem Interrupt aufgerufen wird, wird nun der A/D-Wandler angesteuert, die gewandelten Meßdaten vom A/D-Wandler in den Speicher geschrieben und ein Daten-Pointer weiterbewegt. Außerdem wird die Anzahl der Meßdaten mitgezählt und das Programm gegebenenfalls bei Ende der Messung abgebrochen.

Die **Auto-Init-Prozedur** eines Programms wird (optional), unmittelbar nachdem das Programm unter einer Task installiert wurde, einmalig vom Betriebssystem aufgerufen. Der Aufruf kann aber mit einem Flag bei der Installierung unterbunden werden.

Beispiel: Bei einem PID-Regler-Programm wird nach der Installierung der analoge Ausgang auf einen definierten Anfangswert gesetzt.

Globale Prozeduren sind Subroutinen, die von allen Tasks aus aufgerufen werden können, auch vom PC aus. Sie "gehören" zu einem Programm und stellen anderen Programmen bzw. Tasks bestimmte Funktionen zur Verfügung, die üblicherweise im Zusammenhang mit dem Programm stehen, aber nicht müssen.

Beispiel: Ein interruptgesteuertes Programm holt Daten von einer seriellen Schnittstelle ab und legt sie in einem Puffer ab. Das Programm stellt eine Prozedur zur Verfügung, mit der ein anderes Programm ein oder mehrere Zeichen aus dem Puffer auslesen kann.

Unbedingt erforderlich für ein Programm ist lediglich die Haupt-Prozedur, alle übrigen Prozeduren sind optional. Falls die Task, unter der das Programm installiert wurde, nie aktiviert wird, kann man sogar die Haupt-Prozedur weglassen. Eine solche Task könnte dann z.B. nur einen Daten- und/oder einen Parameterbereich haben, in dem globale Parameter für viele andere Tasks zur Verfügung gestellt werden. Aus Gründen der Übersichtlichkeit kann es in größeren Projekten auch durchaus sinnvoll sein, bestimmte Prozeduren, die von mehreren Tasks aus aufrufbar sein sollen, nur einmal auf der Karte zu halten und einer Task zuzuordnen, die selbst gar keine Hauptprozedur hat.

5.2.3. Die Tasktypen

Wann eine Task an der Reihe ist, also deren Haupt-Prozedur aufgerufen wird, hängt u.a. vom Tasktyp ab. Dieser wird durch das Programm selbst vorgegeben oder bei der Installierung der Task festgelegt. Es gibt drei Tasktypen:

Interrupt-Tasks (mit DI- bzw. II-Tasks)

Timer-Initiierte Tasks (TI-Tasks)

Nicht-Interrupt-Tasks (NI-Tasks)

5.2.3.1. Interrupt-Tasks (DI- bzw. II-Tasks)

Sie haben die höchste Priorität. Wenn das entsprechende Interrupt-Ereignis auftritt, wird die Haupt-Prozedur des unter der Interrupt-Task installierten Programms einmal aufgerufen. Das Programm, also die Haupt-Prozedur, bestimmt selbst, wann es die Kontrolle an das Betriebssystem zurückgibt.

Der Unterschied zwischen einer DI-Task (Direkte Interrupt-Task) und einer II-Task (Indirekte Interrupt-Task) besteht nur im Service, den das Betriebssystem dem Programmierer für die Haupt-Prozedur zur Verfügung stellt. Die Haupt-Prozedur einer II-Task ist eine ganz gewöhnliche Prozedur und hat den gleichen Aufbau wie alle anderen Prozeduren, auch wie die Hauptprozeduren der anderen Tasktypen. Die Haupt-Prozedur einer DI-Task dagegen muß als Interrupt-Service-Routine programmiert werden, d.h. der Programmierer muß sich um das Retten der Register und das korrekte Verlassen der Prozedur selbst kümmern. Dies kann für sehr zeitkritische Anwendungen ausgenutzt werden. Es ist aus vielen Gründen sinnvoll, die CPU durch interruptgesteuerte Programme so wenig wie möglich zu belasten.

Die Priorität der DI- bzw. II-Tasks untereinander wird durch die Hardware der MODULAR-4 Karte weitgehend festgelegt. Es besteht zwar von Hardwareseite

durch entsprechende Programmierung der Interrupt-Controller die Möglichkeit, die Priorität zu ändern, was aber bisher nicht genutzt wird. Da einige Interrupt-Quellen, z.B. die Timer und die externen Interrupt-Eingänge für die Module IRQ-A bis IRQ-F funktionell im übrigen gleichwertig sind, kann die Priorität auch durch die Wahl des Timers oder des Interrupt-Eingangs in gewissem Umfang beeinflußt werden.

Bei der Installierung einer Interrupt-Task kann die Interrupt-Nummer mit angegeben werden, wenn sie nicht durch das Programm fest vorgegeben ist. Sie entspricht nicht der Interrupt-Vektor-Nummer (siehe Anhang D). Im folgenden wird einfach von Interrupt xxx gesprochen, wenn die Interrupt-Nummer gemeint ist. Im Anhang D finden Sie eine Zusammenstellung aller lokalen Interrupts auf der MODULAR-4 Karte.

5.2.3.2. Timer-Initiierte Tasks (TI-Tasks)

Die Hauptprozedur einer TI-Task wird vom Betriebssystem in einstellbaren festen Zeitintervallen aufgerufen. Nach einer vom Benutzer angegebenen Anzahl von Aufrufen wird die Task automatisch deaktiviert, sofern die Anzahl der Aufrufe nicht auf "unendlich" gestellt wurde. Der erste Aufruf einer TI-Task kann sofort nach ihrer Aktivierung erfolgen oder erst nach einer einstellbaren Verzögerungszeit. Alle Zeitangaben, die zur Steuerung einer TI-Task benötigt werden, werden dem Betriebssystem beim Aktivieren als "Zeitplan" übergeben. Falls mehrere Aktivierungs-Befehle gegeben werden, während die zu aktivierende Task noch nicht aktiv ist, wird nur die letzte Aktivierung ausgeführt. Falls eine Task bereits aktiv ist und weitere Aktivierungs-Befehle gesendet werden, dann wird der letzte Aktivierungs-Befehl zwischengespeichert und nach Beendigung der Task ausgeführt.

Alle TI-Tasks werden von dem sogenannten TI-Task-Scheduler aufgerufen, der mit TIMER-C arbeitet und in festen Zeitintervallen (standardmäßig 1 ms) angesprochen wird. Dieses Zeitintervall wird als "Timer-Tic" bezeichnet, auf den sich alle TI-Tasks beziehen. Der Zeitplan der TI-Tasks, der beim Aktivieren übergeben wird, wird in Vielfachen des Timer-Tics angegeben. Um den Task-Scheduler müssen Sie sich im übrigen nicht weiter kümmern. Er wird automatisch eingerichtet, sobald die erste TI-Task installiert wird. Wenn Sie den Standard Timer-Tic von 1 ms verändern wollen, müssen Sie das vor Installierung der ersten TI-Task tun (durch Änderung von Betriebssystemparameter 316).

Die TI-Tasks haben untereinander eine einstellbare Priorität, die bestimmt, welche Task gestartet wird, falls der Aufruf zweier TI-Tasks zeitlich zusammenfällt. Es ist auch möglich, einer TI-Task einmalig die höchste Priorität zuzuordnen, so daß sie auf jeden Fall als nächste TI-Task an die Reihe kommt, danach aber regulär weiterbearbeitet wird. Wenn eine TI-Task nicht zum vorgesehenen Zeitpunkt aufgerufen

werden konnte, weil noch eine andere TI-Task lief oder eine TI-Task mit höherer Priorität zum selben Zeitpunkt gestartet werden sollte, wird der Aufruf später nachgeholt. Auch wenn mehrere Aufrufe hintereinander nicht bearbeitet werden konnten, geht keiner davon verloren. Sie werden alle nachgeholt, sobald die Tasks mit höherer Priorität ihre Aufrufe beendet haben. Wenn der Aufruf einer TI-Task aber über einen einstellbaren Zeitraum (Betriebssystemparameter 324) hinaus verzögert wird, löst das Betriebssystem einen Error-Request (Interrupt mit Übergabe eines Fehlercodes) zum PC aus.

! *Bitte beachten Sie, daß sich TI-Tasks gegenseitig nicht unterbrechen. Eine laufende TI-Task wird immer erst beendet, bevor eine andere an die Reihe kommt, unabhängig von der Priorität.*

TI-Tasks bestimmen ebenso wie DI- und II-Tasks selbst, wann sie die Kontrolle an das Betriebssystem zurückgeben. TI-Tasks haben eine niedrigere Priorität als DI- und II-Tasks und können von diesen unterbrochen werden.

5.2.3.3. Nicht-Interrupt Tasks (NI-Tasks)

Sie haben die niedrigste Priorität. Sie können von DI-, II- und TI-Tasks unterbrochen werden. NI-Tasks werden in einer vom Anwender festgelegten Reihenfolge aufgerufen. Die Häufigkeit, mit der eine NI-Task aufgerufen wird, kann dadurch erhöht werden, daß sie mehrfach aktiviert wird. Dadurch ändert sich aber nur ihre Aufruffrequenz relativ zu anderen NI-Tasks. Auch NI-Tasks bestimmen selbst, wann sie die Kontrolle an das Betriebssystem zurückgeben.

Eine Priorität im herkömmlichen Sinne von NI-Tasks untereinander existiert eigentlich nicht, weil NI-Tasks vom Betriebssystem einfach in einer bestimmten Reihenfolge aufgerufen werden, die durch die Reihenfolge ihrer Aktivierung gegeben ist. Da aber NI-Tasks auch mehrfach aktiviert werden können, kann man sagen, daß eine häufig aktivierte NI-Task statistisch auch häufiger aufgerufen wird und damit eine höhere Bedeutung hat.

Auch bei NI-Tasks besteht die Möglichkeit, eine bestimmte Task vorübergehend als die nächste Aufzurufende in der Reihenfolge ganz nach vorne zu schieben, so daß sie in jedem Fall als nächste NI-Task drankommt. Falls gerade eine NI-Task läuft, so wird damit gewartet, bis diese die Kontrolle an das Betriebssystem zurückgegeben hat.

5.2.4. Daten- und Parameterbereich

Bei der Installierung eines Programms unter einer Task bekommt diese Task einen Datenbereich und einen Parameterbereich zugewiesen. Die Größe dieser beiden Bereiche wird vom Programm selbst, die des Datenbereichs kann auch beim Installieren festgelegt werden. Beide Bereiche sind später nicht mehr änderbar. Sie können auch die Länge 0 haben. Sie sind voneinander unabhängige, lineare, durchgängige Bereiche. Sie unterscheiden sich durch ihre maximal erlaubte Größe und durch die Art, wie auf ihre Inhalte zugegriffen wird:

Der Zugriff auf den **Datenbereich** geschieht über Betriebssystemaufrufe. Dazu existiert für jede Task ein Schreib- und ein Lese-Pointer, die von den vom Betriebssystem zur Verfügung gestellten Subroutinen und von den Makrobefehlen verwendet werden. Der jeweilige Pointer wird nach jedem Zugriff um die Anzahl Byte inkrementiert, die gelesen bzw. geschrieben wurden. Der Datenbereich ist nicht segmentiert, so daß auch Speicherbereiche oberhalb 1 MB vom Betriebssystem genutzt werden können. Er eignet sich also besonders für fortlaufende Daten, wie sie zum Beispiel bei einem Meßdatenerfassungsprogramm anfallen.

Auf den **Parameterbereich** wird entweder direkt oder durch Betriebssystemaufrufe mit Angabe der Nummer des Parameters zugegriffen. Die Nummer ist die relative Adresse des ersten Byte des Parameters, bezogen auf den Anfang des Parameterbereichs (siehe auch Kapitel 7.3.2.).

	Datenbereich	Parameterbereich
Länge	0 bis max. verfügbarer Speicher	0 bis max. (64K - 256)
Zugriff	Indirekt über Pointer	Direkt

5.2.5. Datenpuffer

Ab Version ML7-1A.01x bzw. ML8-3A.xxy stellt das Betriebssystem ringförmig organisierte Datenpuffer zur Verfügung. Das Betriebssystem legt auf Anforderung einen Ringpuffer im Speicher der Karte an und liefert eine Puffernummer zurück, die für alle weiteren Zugriffe auf den Puffer benutzt wird. Zur Zeit können bis zu 256 voneinander unabhängige Puffer angelegt werden.

Ein Puffer arbeitet byteorientiert nach dem FIFO-Prinzip, d.h. die zuerst in den Puffer geschriebenen Byte werden auch als erste wieder ausgelesen.

Einige wichtige Punkte sind beim Arbeiten mit den Puffern zu beachten:

1. Alle Routinen, die die Puffer ansprechen, dürfen "gleichzeitig" von mehreren Tasks aufgerufen werden.
2. Derselbe Puffer darf "gleichzeitig" beschrieben und gelesen werden. So kann z.B. eine NI-Task (Nicht-Interrupt Task) einen Datenblock in einen Puffer schreiben. Dabei kann sie von einer Interrupt-Task unterbrochen werden, die dann einen (anderen, älteren) Datenblock aus dem Puffer auslesen kann.
3. Schreiben in einen Puffer und Lesen aus einem Puffer arbeiten blockorientiert, d.h.:
   a) Die mit einem Aufruf einer Schreibroutine geschriebenen Daten können erst dann ausgelesen werden, wenn der komplette Block im Puffer steht und der Aufruf der Schreibroutine beendet ist.
   b) Der durch das Auslesen eines Datenblocks aus dem Puffer frei gewordene Platz steht erst nach Beenden der Leseroutine wieder zur Verfügung.
   c) Die beim Schreiben und Lesen verwendeten Blockgrößen können unabhängig voneinander bei jedem Aufruf beliebig gewählt werden (max. Blockgröße ist 65520 Byte).
4. Ein Puffer kann auch von mehreren Tasks "gleichzeitig" beschrieben und/oder gelesen werden. Ein Puffer, der gerade beschrieben wird, ist aber während dieser Zeit für alle Tasks gesperrt, die in denselben Puffer schreiben wollen. Ebenso ist ein Puffer, aus dem gerade Daten gelesen werden, während dieser Zeit für alle anderen Tasks gesperrt, die ebenfalls Daten aus diesem Puffer lesen wollen. Diese Tasks würden die Fehlermeldung erhalten: "Puffer wird gerade beschrieben" bzw. "Puffer wird gerade ausgelesen". Sie müssen es später wieder versuchen.

Beim Anlegen eines Puffers sind für das Reservieren von Speicherplatz und für das Ausrichten des Speichers (Alignment) verschiedene Strategien möglich, die bei den Befehlen zum Einrichten eines Puffers (Kapitel 6, 9, 10 oder 12) beschrieben sind.

Es ist bei der MODULAR-4/486 Karte aus Geschwindigkeitsgründen sinnvoll, alle Puffer auf DWord (4 Byte) auszurichten. Wenn möglich, sollten auch die Zugriffe jeweils Vielfache von 2 oder möglichst 4 Byte lesen bzw. schreiben.

5.2.6. Fehlerbehandlung

Fehler und Service-Requests (sog. SRQs) werden als Wort (2 Byte) über eine Schnittstelle, z.B. über die PC-Schnittstelle gesendet. Eine Aufstellung dieser Fehler- und SRQ-Codes finden Sie im Anhang F.

Bei Fehlern und SRQs wird wie folgt unterschieden:

Typ	Beispiel
Systemfehler	CPU-Defekt nach Selbsttest
Spontaner Fehler	Nach Abschluß eines Makrobefehls
Provozierter Fehler	Während Ausführung eines Makrobefehls
System-SRQ	Überlauf eines Systempuffers
Anwender-SRQ	SRQ aus Anwendungsprogramm

Innerhalb des Betriebssystems auftretende Fehler werden direkt über die PC-Schnittstelle zum PC gesendet, z.B. solche Fehler, die nach dem Einschalten des Systems und beim anschließenden Selbsttest der Karte auftreten.

5.2.7. Einige betriebssysteminterne Tasktabellen

Bei der Installierung eines Programms unter einer Task werden außer Daten- und Parameterbereich auch einige Tabellen angelegt, die für die Verwaltung notwendig sind. Die beiden wichtigsten sind:

Programm-Deskriptor-Tabelle (PDT)

Task-Deskriptor-Tabelle (TDT)

Die Programm-Deskriptor-Tabelle (PDT) ist Teil des Programms, das auf die Karte geladen und unter einer Task installiert wird. Sie wird durch die Installierung nicht verändert, auch das Programm selbst darf sie zur Laufzeit nicht mehr verändern. Es könnte ja z.B. sein, daß dasselbe Programm nochmals unter einer anderen Task installiert wird, dann werden die Informationen auch nochmals benötigt. Außerdem greift das Betriebssystem auch zur Laufzeit auf die Tabelle zu, allerdings nur lesend. Der Aufbau und eine genaue Erklärung der PDT finden Sie in Anhang I.

Die Task-Deskriptor-Tabelle (TDT) ist der jeweiligen Task zugeordnet. Sie enthält praktisch alle Informationen über die Task, über das unter der Task installierte Programm, z.B. auch die Adresse der PDT, und weitere Parameter, die vorwiegend zur Laufzeit benutzt werden. Hier stehen auch die oben erwähnten Schreib- und Lese-Pointer für Zugriffe auf den Datenbereich. Den Aufbau und eine genaue Erklärung der TDT finden Sie in Anhang J.

5.2.8. Installierung von Programmen

Ein Anwendungsprogramm, das von SORCUS auf einem Datenträger geliefert wurde, muß zunächst in das RAM der Karte geladen und anschließend installiert werden. Der Name solcher Programmdateien ist:

MyPxxxx.LIB oder

MyPxxxx.EXE

Dabei bedeutet xxxx die Programmnummer zwischen 1 und ffffh (hexadezimal), y steht für den Kartentyp (7="kleine" MODULAR-4/486, 8="große" MODULAR-4/486.

Zur komfortablen Installierung der Programme finden Sie auf dem Originaldatenträger, der mit der MODULAR-4 Karte geliefert wurde, das Hilfsprogramm

"SNW" bzw. **"SNW32"**.

Das Programm "SNW" (**S**chöne **N**eue **W**elt) kann auch von der "AUTOEXEC.BAT" Datei des PC automatisch aufgerufen werden. Es sorgt dann für eine komplette Installierung der gewünschten Programme auf der Karte, beim Einschalten des Systems, ohne daß der Anwender eingreifen muß. Eine ausführliche Beschreibung des PC-Hilfsprogramms finden Sie in Kapitel 4 und Anhang M.

Wenn Sie die Programme aus Ihrem eigenen PC-Programm heraus installieren wollen, stehen dafür Funktionen in der PC-Bibliothek "ML7BIB" bzw. "ML8BIB" zur Verfügung. Mehr darüber erfahren Sie in Kapitel 6.

On-board-Echtzeitprogrammdateien mit der Erweiterung ".EXE" dürfen nicht auf dem PC gestartet werden und umgekehrt für den PC geschriebene Programme nicht auf die Karte geladen werden. In den Kapiteln 7, 8, 9 und 10 finden Sie Angaben dazu, wie Echtzeitprogramme für die MODULAR-4 erstellt werden. **!**

Programme im ROM bzw. Flash-EPROM können einfach mit den entsprechenden Bibliotheksbefehlen oder mit SNW32 bzw. SNW installiert werden. Außerdem kann eine Liste zu installierender ROM-Programme in ROM bzw. Flash-EPROM abgelegt werden, die nach dem Einschalten der MODULAR-4 Karte automatisch abgearbeitet wird.

5.3. Installieren eines neuen Betriebssystems

Nach einem Reset der Karte ist zunächst immer das sogenannte ‘Mini-Betriebssystem’ im ROM aktiv. Es dient im wesentlichen zum Laden und Aktivieren eines vollständigen Betriebssystems und zu Debugging-Zwecken (z.B. Post-mortem-Analyse nach Programmabsturz). Das Mini-Betriebssystem arbeitet ohne Verwendung von RAM und Interrupts und beherrscht nur einige Makrobefehle.

Neben dem Mini-Betriebssystem enthält das ROM der Karte ein vollständiges Betriebssystem (im folgenden ‘ROM-Betriebssystem’ genannt), das nach einem Reset erst aktiviert werden muß. Aus Geschwindigkeitsgründen wird dieses Betriebssystem vom ROM ins RAM kopiert und dann gestartet. Das Aktivieren geschieht in der Regel durch die Angabe eines entsprechenden Parameters im Resetbefehl der Bibliotheken bzw. von SNW32 bzw. SNW. Alternativ kann per Jumper an St1 eingestellt werden, ob nach einem Reset (z.B. bei Power-on) automatisch das ROM-Betriebssystem gestartet wird.

Das Betriebssystem der MODULAR-4 Karte kann statt aus dem ROM auch vom PC aus ins RAM der Karte geladen werden. Auf diese Weise sind z.B. Updates ohne Eingriff in die Hardware möglich. Auch dieser Vorgang kann sehr einfach und schnell mit dem Resetbefehl der Bibliotheken und von SNW32 bzw. SNW (unter Angabe des Namens einer Datei, die ein Betriebssystem enthält) erledigt werden. Wenn das Betriebssystem läuft, besteht kein Unterschied, ob es aus dem ROM oder vom PC geladen wurde. Die Bibliotheken enthalten Funktionen, mit denen ermittelt werden kann, welcher Typ und welche Version des Betriebssystems aktuell auf der Karte läuft.

Der Name der Betriebssystemdatei enthält Versionsinformationen und ist wie folgt aufgebaut:

MLx-za.nnR	Neues Betriebssystem x = 7: "kleine" MODULAR-4/486, x = 8: "große" MODULAR-4/486 z = Zahl 1 bis 9 (Version), a = Buchstabe A bis Z (Revision), nn = lfd. Nr. des Betriebssystems

Das Programm SNW32 bzw. SNW bietet die Möglichkeit, bei einem Reset in einer Installationsdatei (in der MxDEVICE-Zeile) oder einem interaktiven Reset (z.B. [CTRL]+[F10]) das jeweils aktuellste Betriebssystem zu laden. Dazu muß als Name "ML7-??.??R" bzw. "ML8-??.??R" angegeben werden. SNW32 bzw. SNW durchsucht dann beim Reset das "OSX"-Verzeichnis (wird beim Installieren der Original-Datenträger erstellt) und das Verzeichnis, in dem "SNW32.EXE" bzw. "SNW.EXE" steht, nach dem aktuellsten Betriebssystem. Ein neues Betriebssystem muß also nur

in eines dieser Verzeichnisse kopiert werden (vorzugsweise in das "OSX"-Verzeichnis).

5.4. Aufbau des RAM-Bereichs der Karte

Das Betriebssystem residiert immer in den unteren 64 KByte. Die Interrupt-Vektor-Tabelle beginnt ab Adresse 0 und belegt 1 KByte. Nach einem Hardware-Reset und der Aktivierung des Betriebssystems können Sie die Grenzen des noch freien RAM von der Karte lesen. Die Grenzen stehen im Parameterbereich des Betriebssystems, also der Task 0 (siehe Kapitel 5.5 oder Anhang K). Es werden immer physikalische 32-Bit-Adressen geliefert, als untere Grenze wird zur Zeit 0001 0000h gemeldet. Wenn ein Programm auf die Karte geladen und installiert wird, wird die untere bzw. obere Grenze des freien RAM entsprechend der Größe des Programms und gegebenenfalls der Größe des Parameter- und Datenbereichs der gerade installierten Task verändert, also nach oben verschoben. Auch bestimmte Aktionen des Betriebssystems belegen zusätzlich etwas Speicherplatz.

Bei der aktuellen Version des Betriebssystems, das meistens im Real Mode arbeitet, wird von den Anwendungsprogrammen erwartet, daß sie ebenfalls im Real Mode arbeiten. Nur die unteren 1 MByte des Speichers können also für Programme genutzt werden. Parameter- und Datenbereiche können auch im darüber hinausgehenden Bereich liegen (je nach Speicherausbau z.B. bis 34 MByte).

5.5. Parameterbereich des Betriebssystems

Das Betriebssystem selbst ist ebenfalls eine Task auf der Karte: Task 0. Damit stehen alle Taskbefehle auch für den Zugriff auf die Strukturen des Betriebssystems zur Verfügung. Das Betriebssystem hat dementsprechend auch einen eigenen Parameterbereich, der teilweise auch für den Anwender von Interesse sein kann. Auch hier nicht erklärte bzw. reservierte Parameter dürfen auf keinen Fall verändert werden.

In der folgenden Tabelle bedeutet in der Spalte "Typ":

B = Ein-Byte Parameter	D = Doppelwort Parameter (4 Byte)
nB = n-Byte Parameter	nS = n-Byte String (ASCII-Zeichen)
W = Wort Parameter (2 Byte)	P = Physikalische Adresse (4 Byte)

In Spalte "Zugr" bedeutet: R = Parameter darf nur gelesen werden
RW = Parameter darf gelesen und geschrieben werden

Parameter	Typ	Zugr	Bedeutung
0 (00h)	B	R	Version dieser Parameterdefinition: z.Zt. = 01h
1 (01h)	W	R	Anzahl gültiger Parameter des Betriebssystems
4 (04h)	B	R	Kartentyp (7 = "kleine", 8 = "große" MODULAR-4/486)
5 (05h)	B	R	Kartenrevision (1=A, 2=B, 3=C, ...)
10 (0ah)	2S	R	Jahrhundert der Herstellung des Betriebssystems, als Ergänzung zum Datum in Parameter 22
12 (0ch)	10S	R	Betriebssystem Name, Version und Revision, z.B.: "ML8-3B.05x" x = "R" für RAM-Version, x = "E" für (EP)ROM-Version x = "B" für RAM-Beta-Test-Version
22 (16h)	8S	R	Datum der Herstellung des Betriebssystems, z.B.: "10/08/98" (tt/mm/jj) (siehe auch Parameter 10)
30 (1eh)	8S	R	Uhrzeit der Herstellung des Betriebssystems, z.B.: "12:42:59" (hh:mm:ss)
38 (26h)	B	R	CPU-Typ: 01h = V20, 04h = i486, 05h = Pentium
39 (27h)	B	R	CPU-Revision (siehe Intel-/NEC-Spezifikation)
40 (28h)	B	R	CPU-Modell (z.Zt. nur gültig für Intel-Pentium)
41 (29h)	B	R	CPU-Hersteller: 1 = Intel, 2 = AMD, 3 = UMC, 4 = TI 5 = Cyrix, 6 = IBM, 7 = STM, 8 = NexGen, 9 = NEC, 10 = Transmeta, 255 = nicht ermittelt
42 (2ah)	D	R	Ergebnis des Hardware-Selbsttests der CPU (0 = ok, <> 0 = Fehler)
46 (2eh)	B	R	Co-Proz.-Typ: 0 = keiner, 4 = 487, 5 = Pentium
47 (2fh)	B	R	Co-Proz.-Hersteller: 1 = Intel, 255 = nicht ermittelt
48 (30h)	D	R	CPU-Features (z. Zt. nur gültig für Intel-Pentium)

Parameter	Typ	Zugr	Bedeutung
56 (38h)	W	R	Betriebssystem-Features: Bit-0: 0 = Datenzugriffe auf 1 MB beschränkt 1 = Datenzugriffe bis 4 GB erlaubt Bit-1: 0 = Datenbereiche werden von unten nach oben im Speicher reserviert 1 = von oben nach unten Bit-2: 0 = freies RAM wird nicht initialisiert 1 = freies RAM wird mit 0 initialisiert Bit-3: 0 = RAM-Größen Erkennung automatisch 1 = nicht automatisch, fix
64 (40h)	P	R	Physikal. RAM-Anfang (physikal. Adresse)
68 (44h)	P	R	Physikal. RAM-Ende (letzte Stelle + 1)
72 (48h)	P	R	Freies RAM-Anfang (erste freie Stelle)
76 (4ch)	P	R	Freies RAM-Ende (letzte Stelle + 1)
96 (60h)	W	R	Länge des Empfangs-Puffers für PC-Kommunikation, wenn = 0, dann max. 256
98 (62h)	W	R	Länge des Sende-Puffers für PC-Kommunikation, wenn = 0, dann max. 256
100 (64h)	W	R	EEPROM der Basiskarte: Länge (Anzahl Byte)
102 (66h)	W	R	Parameternr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 100 = 0
104 (68h)	W	R	EEPROM von Modul 1: Länge (Anzahl Byte)
106 (6ah)	W	R	Parameternr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 104 = 0
108 (6ch)	W	R	EEPROM von Modul 2: Länge (Anzahl Byte)
110 (6eh)	W	R	Parameternr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 108 = 0
112 (70h)	W	R	EEPROM von Modul 3: Länge (Anzahl Byte)
114 (72h)	W	R	Parameternr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 112 = 0
116 (74h)	W	R	EEPROM von Modul 4: Länge (Anzahl Byte)

Parameter	Typ	Zugr	Bedeutung
118 (76h)	W	R	Parameternr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 116 = 0
120 (78h)	W	R	EEPROM von Modul 5: Länge (Anzahl Byte)
122 (7ah)	W	R	Parameternr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 120 = 0
124 (7ch)	W	R	EEPROM von Modul 6: Länge (Anzahl Byte)
126 (7eh)	W	R	Parameternr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 124 = 0
128 (80h)	W	R	EEPROM von Modul 7: Länge (Anzahl Byte)
130 (82h)	W	R	Parameternr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 128 = 0
132 (84h)	W	R	EEPROM von Modul 8: Länge (Anzahl Byte)
134 (86h)	W	R	Parameternr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 132 = 0
136 (88h)	W	R	EEPROM von Modul 9: Länge (Anzahl Byte)
138 (8ah)	W	R	Parameternr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 136 = 0
140 (8ch)	W	R	EEPROM von Modul 10: Länge (Anzahl Byte)
142 (8eh)	W	R	Parameternr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 140 = 0
148 (94h)	W	R	CPU-Taktfrequenz (in Vielfachen von 1 MHz)
150 (96h)	W	R	Timer-Eingangstakt (in Vielfachen von 10 kHz)
168 (a8h)	W	R	Max. Anzahl Aktivierungen für NI-Tasks
170 (aah)	W	R	Max. Anzahl Tasks (alle Typen)
176 (b0h)	W	R	Max. Anzahl Puffer
200 (c8h)	B	RW	SRQ-Mode (PC-Schnittstelle): z.Zt. = 1
201 (c9h)	B	RW	SRQ-Delay (PC-Schnittstelle): z.Zt. = 64
202 (cah)	W	R	Größe des SRQ-Puffers in Bytes (Standardeinst. = 1024)
204 (cch)	W	R	Nummer des SRQ-Puffers
210 (d2h)	W	R	Zähler für Interrupts an IRQ-7 Master

Parameter	Typ	Zugr	Bedeutung
212 (d4h)	W	R	Zähler für Interrupts an IRQ-7 Slave
214 (d6h)	B	RW	Watchdog: 0 = Auto-Retrigger off, 1 = on
240 (f0h)	B	R	Status der externen LED (1 = an, 0 = aus)
241 (f1h)	B	R	Status der lokalen LED (1 = an, 0 = aus)
308 (134h)	W	R	Max. Anzahl TI-Tasks
310 (136h)	B	R	Typ des Timer-Chips (0 = 8254)
311 (137h)	B	RW[1]	Timer-Kanal für TI-Tasks (0 = Timer-A, 1=B, 2=C) (Standardeinstellung: Timer-C)
313 (139h)	B	RW[1]	Interrupt-Nr. für TI-Task Timer (Standardeinst.: 93h)
316 (13ch)	D	RW[1]	Timer-Tic für TI-Tasks in Mikrosekunden Es sind Werte zwischen 100 und 26000 zulässig. Sonderfall: 0 bedeutet 1 ms (=Standardeinstellung)
320 (142h)	W	R	Längste Verzögerungszeit einer TI-Task seit Reset (Tics)
322 (144h)	W	R	TI-Task, die das Maximum (Parameter 320) ausgelöst hat
324 (146h)	W	RW	Meldegrenze für Zeitverzögerung einer TI-Task (in Timer-Tics, ffffh = keine Meldung)
326 (148h)	W	RW	Error-Request-Wort bei Erreichen der Meldegrenze (Parameter 320 > Parameter 324)
398 (16eh)	W	R	Aktuell bearbeitete NI-Task
400 (170h)	W	R	Parameter-Nr., wo Fehler bei System-Subroutinen gemerkt werden
402 (172h)	W	R	Parameter-Nr., wo Fehler bei Makro-Befehlen gemerkt werden

[1] Die Parameter für TI-Tasks können nur vor der ersten Installierung einer TI-Task eingestellt werden, spätere Einstellungen sind wirkungslos.

Notizen

6. PC-Hochsprachenbibliotheken

Die Hochsprachenbibliotheken wickeln auf der PC-Seite die Kommunikation zwischen PC und der MODULAR-4 Karte ab. Sie enthalten Prozeduren und Funktionen, deren hauptsächliche Aufgabe es ist, das Multi-Tasking-Betriebssystem der MODULAR-4 Karte über Makrobefehle anzusprechen. Es stehen Bibliotheken für die PC-Betriebssysteme DOS und Windows (3.x, 95, 98, ME, 2000 und NT) zur Verfügung.

Alle Funktionen und Prozeduren der Bibliotheken beginnen mit **ml7_...** (ML7 = "kleine" MODULAR-4/486 Karte) bzw. **ml8_...** (ML8 = "große" MODULAR-4/486 Karte. Die Funktionen sind gleichwertig, aus Gründen der Übersichtlichkeit werden aber im folgenden nur die Funktionen der Bibliothek für die "große" MODULAR-4/486 Karte beschrieben.

Die Bibliotheken enthalten auch Funktionen für die Initialisierung der MODULAR-4 Karte und für die Fehlerbehandlung. Sie unterstützen bis zu acht MODULAR-4 Karten in einem PC und gestatten das Arbeiten mit PC-Interrupt-Auslösung durch die MODULAR-4 Karten.

Bei der Durchführung von Makrobefehlen werden unzulässige Zeitüberschreitungen (Timeout) festgestellt und die Integrität der Antworten auf die Makrobefehle wird überprüft. Die Fehlerbehandlung erfolgt in einer vom Benutzer programmierbaren Fehlerbehandlungsroutine.

6.1. Voraussetzungen für die Verwendung

Achten Sie darauf, daß Sie die MODULAR-4 Karte unter der richtigen Basisadresse (Werkseinstellung 380h) ansprechen. Wenn Sie mit Interrupts arbeiten wollen, achten Sie darauf, daß die Einstellung auf der Karte (Werkseinstellung IRQ-7) und die Einstellung in der Bibliothek bzw. im Setup-Programm übereinstimmen. Lesen Sie bitte auch die in den Plattform-spezifischen Unterverzeichnissen (DOS, Windows, Win95 und WinNT) befindlichen **README.TXT** Dateien.

6.2. PC-Betriebssysteme

In den folgenden Abschnitten werden Informationen zur Benutzung der Bibliothek unter verschiedenen PC-Betriebssystemen gegeben.

Wird im folgenden auf Compiler verwiesen, so wird dabei bewußt auf die Versionsangabe verzichtet. **Die jeweils aktuell unterstützten Compilerversionen finden Sie in den betriebssystemspezifischen README-Dateien.**
In diesen Dateien ist auch festgehalten, welche Dateien für die jeweiligen Compiler eingebunden werden müssen.

Beispielprogramme für die verschiedenen Betriebssysteme finden Sie auf der CD bzw. den mitgelieferten Disketten.

6.2.1. MS DOS

Programmiersprache C

Die Bibliothek wurde mit den C-Compilern von Borland, Microsoft und Watcom getestet.

Als Code-Modell muß **large** eingestellt sein. Alle Funktionen sind '**far pascal**' deklariert.

Die Benutzer-Fehlerbehandlungs-Funktion und die Benutzer-Interrupt-Service-Funktion, die von Ihnen deklariert werden, müssen ebenfalls mit den Modifizierern '**far pascal**' versehen werden.

Für die Programmierung der Benutzer-Service-Funktion und der Benutzer-Fehlerbehandlungs-Funktion beachten Sie bitte die Hinweise auf Seite 6-58.

Programmiersprache Pascal

Die Bibliothek wurde mit dem Pascal-Compiler von Borland getestet.

Für die Programmierung der Benutzer-Service-Funktion und der Benutzer-Fehlerbehandlungs-Funktion beachten Sie bitte die Hinweise auf Seite 6-58.

6.2.2. Windows 3.x

Für Windows 3.x stehen die Bibliothek WML7BIB.DLL und WML8BIB.DLL zur Verfügung. Die Aufrufkonvention der Funktionen ist **far pascal**. Zur Laufzeit eines Programms muß auf diese DLL zugegriffen werden können.

Die Bibliotheken wurden mit den C-Compilern von Borland, Microsoft und Watcom und den Compilern Borland Pascal und Borland Delphi getestet.

Für die Programmierung der Benutzer-Service-Funktion und der Benutzer-Fehlerbehandlungs-Funktion beachten Sie bitte die Hinweise auf Seite 6-58.

6.2.3. Windows 95

Für Windows 95 steht die Bibliothek **FMLXBIB.DLL** zur Verfügung. Sie setzt auf dem Gerätetreiber **MLXDRV.VXD** auf. Zur Laufzeit eines Programms muß der Treiber installiert sein und auf die DLL muß zugegriffen werden können.

Der Gerätetreiber muß in das Treiberverzeichnis von Windows 95 kopiert und in der Registrierdatenbank installiert werden. Dazu wird ein Installierungsprogramm mitgeliefert. Informationen über dieses Programm finden Sie im Verzeichnis '\ml8\win32\nt4win95' in der Datei 'win95_d.txt'. Unter Windows 95 kann derzeit nur mit einer Karte im PC gearbeitet werden!

Parallele Zugriffe auf die Karte können nur aus 32-Bit-Programmen erfolgen. Benutzen ein 16-Bit-Programm (DOS, Windows 3.x) und ein 32-Bit-Programm (Windows 95) gleichzeitig die Karte, so führt dies zu Kommunikationsfehlern. !

Die Bibliothek wurde mit den C-Compilern von Microsoft und Borland und mit Borland Delphi getestet.

6.2.4. Windows NT

Für Windows NT steht die Bibliothek **MLXW32.DLL** zur Verfügung. Sie setzt auf dem Gerätetreiber **MLXDRV.SYS** auf. Zur Laufzeit eines Programms muß der Gerätetreiber installiert sein und auf die DLL muß zugegriffen werden können.

Für DOS- und Windows-3.x-Anwendungen steht unter Windows NT der virtuelle Gerätetreiber **MLXVDD.DLL** zur Verfügung.

Beide Treiber müssen in das Treiberverzeichnis von Windows NT kopiert und in der Registrierdatenbank installiert werden. Dazu wird ein Installierungsprogramm mitgeliefert. Informationen über dieses Programm finden Sie im Verzeichnis '\ml8\win32\nt4win95' in der Datei 'winnt_d.txt'.

Die Bibliothek wurde mit den C-Compilern von Microsoft und Borland und mit Borland Delphi getestet.

Wenn Sie die MODULAR-4/486 Karte mit DMA betreiben wollen, fordern Sie bitte die entsprechende Application Note AN067 an oder laden Sie diese vom Internet.

6.2.5. Windows 98, ME, 2000

Für Windows 98, ME und 2000 steht die Bibliothek **MLXW32.DLL** zur Verfügung. Sie setzt auf dem Gerätetreiber **MLXWDM.SYS** auf. Zur Laufzeit eines Programms muß der Gerätetreiber installiert sein und auf die DLL muß zugegriffen werden können.

Für DOS- und Windows-3.x-Anwendungen steht unter Windows 98, ME und 2000 der virtuelle Gerätetreiber **MLXVDD.DLL** zur Verfügung.

Beide Treiber müssen in das Treiberverzeichnis von Windows 98, ME oder 2000 kopiert und in der Registrierdatenbank installiert werden. Dazu wird ein Installierungsprogramm mitgeliefert. Informationen über dieses Programm finden Sie im Verzeichnis '\ml8\win32\mlxwdm' in der Datei 'mlxwdm_d.txt'.

6.3. Modul-Device-Treiber

Zur Programmierung der I/O-Devices auf den Modulen und der MODULAR-4/486 Basiskarte können entweder die Modulbibliotheken oder die neueren Modul-Device-Treiber (MDD) verwendet werden. Beides finden Sie auf den mitgelieferten Disketten bzw. CD und im Internet unter **www.sorcus.com**. Die detaillierte Beschreibung zu den Modul-Device-Treibern finden Sie im Anhang O.

Die Modul-Device-Treiber gehen von einem Kanal-orientierten Ansatz aus. Der Anwender 'öffnet' einen Kanal zu einem oder mehreren Devices, z.B. einem Analog-Eingang. Beim Öffnen werden Kanal-spezifische Parameter wie z.B. der Meßbereich übergeben und ein Handle zurückgeliefert. Mit dem Handle kann nun sehr einfach auf das Device zugegriffen werden (z.B. Lesen eines Analogwertes). Wird ein Kanal nicht mehr benötigt, kann er geschlossen werden. Dabei wird der vorher belegte Speicherplatz wieder freigegeben.

Ein weiterer Vorteil der Modul-Device-Treiber ist ihre Multitasking-Tauglichkeit. Ein MDD verwaltet sämtliche Funktionseinheiten eines Moduls und stellt eine normierte Schnittstelle für den Zugriff auf die Funktionseinheiten zur Verfügung.

Modul-Device-Treiber können sowohl von PC-Anwendungen als auch von Echtzeitprogrammen auf der Karte **gleichzeitig** genutzt werden. Als Tasknummer des MDD wird die Nummer des zugehörigen Modulsteckplatzes bzw. 11 für die Basiskarte verwendet.

6.4. Bibliotheksfunktionen

Alle Funktionen und Prozeduren der Bibliotheken beginnen mit **ml7_...** (ML7) bzw. **ml8_...** (ML8). Die Funktionen sind gleichwertig, aus Gründen der Übersichtlichkeit werden aber im folgenden nur die Funktionen der Bibliothek für die "große" MODULAR-4/486 Karte beschrieben.

6.4.1. Funktionen zur Initialisierung

ml8_bib_startup — Initialisiere die Bibliothek

Pascal	PROCEDURE ml8_bib_startup (language: WORD);
C	void ml8_bib_startup (ushort language);
Funktion	Die Funktion initialisiert die Bibliothek.
Parameter	*language*: Für die Programmiersprache C und Visual Basic muß der Parameter *language* auf LANGUAGE_C gesetzt werden, für Pascal und Delphi auf LANGUAGE_PASCAL. Andere Werte sind zur Zeit nicht definiert.
Hinweis	Die Prozedur muß vor allen anderen Bibliotheksroutinen aufgerufen werden. Danach muß die Schnittstelle zur Karte mit **ml8_reset** oder **ml8_start** initialisiert werden. Erst dann dürfen die übrigen Bibliotheksroutinen verwendet werden.

ml8_reset Reset und Initialisieren der Kommunikation

Pascal

```
FUNCTION ml8_reset (card, adr, mode, irq_channel,
dma_channel: WORD; error_handler: POINTER; VAR osname: str80;
timeout: WORD): WORD;
```

C

```
ushort ml8_reset (ushort card, ushort adr, ushort mode,
ushort irq_channel, ushort dma_channel,
ERROR_PROC_P error_handler, char *osname, ushort timeout);
```

Funktion

Die Funktion **ml8_reset** wählt eine von maximal acht MODULAR-4 Karten an, initialisiert den zugehörigen Bibliotheksteil, stellt eine Betriebsart der Karte ein, führt einen Reset der Karte durch, aktiviert ein Betriebssystem und macht einen ersten Kommunikationstest.

Falls kein Reset der Karte durchgeführt und kein neues Betriebssystem auf der Karte aktiviert werden soll, so verwenden Sie anstelle von **ml8_reset** die Funktion **ml8_start** (s.u.).

Die Funktion liefert Null zurück, wenn die Initialisierung erfolgreich durchgeführt werden konnte und eine Fehlermeldung (siehe Anhang E), wenn ein Fehler bei der Initialisierung aufgetreten ist, z.B. ein Fehlerwert 259 (=103h) ist als Fehlerklasse 0, Fehlercode 3 zu interpretieren. Die Fehlerbehandlungsprozedur wird nicht aufgerufen.

Parameter

card: Anzuwählende MODULAR-4 Karte (0 bis 7).

adr: Basis-I/O-Adresse der MODULAR-4 Karte.

mode: Betriebsart der Kommunikation mit der MODULAR-4 Karte. Es stehen folgende Betriebsarten zur Verfügung:

0 = Betrieb ohne PC-Interrupt (wird unter Windows 95 und Windows NT nicht unterstützt)
1 = Betrieb mit PC-Interrupt

irq_channel: Verwendeter Interrupt-Kanal (nur für *mode* = 1). Mögliche Werte sind 3, 4, 5, 7, 9, 10, 11 und 12. Der Kanal muß mit dem auf der Karte per Steckbrücke bzw. im EEPROM-WORT-13 eingestellten Wert übereinstimmen. Unter Windows 95 und Windows NT wird dieser Parameter nicht ausgewertet. Der Interrupt-Kanal wird der Registrierdatenbank entnommen.

dma_channel: Wird nicht verwendet, = 0 setzen.

error_handler: Adresse der Fehlerbehandlungsprozedur. Weitere Informationen zur Fehlerbehandlung finden Sie ab Seite 6-58.

osname: Name des nach Reset zu aktivierenden Betriebssystems. Wenn als Name 'ROM' angegeben ist, wird das Betriebssystem aus dem EPROM aktiviert. Mit 'MINI' wird ein ebenfalls im EPROM befindliches Minimal-Betriebssystem aktiviert, das lediglich für Debugging-Zwecke benötigt wird. Alle anderen Namen werden als Name einer Betriebssystemdatei interpretiert, die geladen werden soll. Der String kann die Laufwerksbezeichnung (z. B. C) und einen vollständigen Pfadnamen enthalten.

timeout: Wartezeit in Zehntelsekunden. Wenn die MODULAR-4 innerhalb dieser Zeit nicht erwartungsgemäß reagiert, wird ein Fehler gemeldet.

ml8_start — Initialisiere die Kommunikation

Pascal
FUNCTION ml8_start (card, adr, mode, irq_channel, dma_channel: WORD; error_handler: POINTER; timeout: WORD): WORD;

C
ushort ml8_start (ushort card, ushort adr, ushort mode,
ushort irq_channel, ushort dma_channel,
ERROR_PROC_P error_handler, ushort timeout);

Funktion
Diese Funktion kann ebenso wie **ml8_reset** zur Initialisierung der Kommunikation verwendet werden, nur daß kein Reset durchgeführt und kein Betriebssystem geladen wird. Alle Programme und das zur Zeit aktivierte Betriebssystem bleiben also unverändert.

Parameter
siehe **ml8_reset**.

ml8_select_card — Wähle eine MODULAR-4 Karte an

Pascal PROCEDURE ml8_select_card (card: WORD);

C void ml8_select_card (ushort card);

Funktion Durch Aufruf der Prozedur wird eine von maximal acht MODULAR-4 Karten angewählt. Die Prozedur dient zum Umschalten zwischen verschiedenen MODULAR-4 Karten. Alle nachfolgend aufgerufenen Bibliotheksaufrufe beziehen sich dann auf die angewählte Karte. Die Anwahl einer Karte innerhalb der Benutzer-Service-Routine wird automatisch von der Bibliothek vorgenommen, **ml8_select_card** ist hier nicht erforderlich.

Parameter *card*: Nummer der Karte (0 bis 7).

ml8_exit_card — Melde eine Karte ab

Pascal PROCEDURE ml8_exit_card (card: BYTE);

C void ml8_exit_card (ushort card);

Funktion Diese Prozedur sollte aufgerufen werden, wenn die angegebene Karte innerhalb Ihres Programms nicht mehr angesprochen wird. Die Prozedur maskiert unter anderem den verwendeten PC-Interrupt-Kanal. Erst nach einem erneuten Aufruf von **ml8_start** oder **ml8_reset** kann die Karte wieder angesprochen werden. Vor Beendigung des Programms muß für jede mit **ml8_start** oder **ml8_reset** angesprochene Karte diese Prozedur aufgerufen werden (siehe auch **ml8_exit**).

Parameter *card*: Nummer der Karte (0 bis 7).

ml8_exit — Beende die Kommunikation

Pascal PROCEDURE ml8_exit;

C void ml8_exit (void);

Funktion Diese Prozedur kann alternativ zu **ml8_exit_card** am Programmende aufgerufen werden. Sie ruft für alle verwendeten (initialisierten) Karten einmal die Prozedur **ml8_exit_card** auf.

ml8_reacting Prüfe ob die MODULAR-4 Karte bereit ist

Pascal FUNCTION ml8_reacting: BOOLEAN;

C ushort ml8_reacting (void);

Funktion Diese Funktion testet, ob die MODULAR-4 Karte zur Kommunikation mit dem PC bereit ist. Wenn die Kommunikation in Ordnung ist, liefert die Funktion den Rückgabewert TRUE , sonst FALSE.

ml8_type_check Identifiziere das aktive Betriebssystem

Pascal FUNCTION ml8_type_check: BYTE;

C byte ml8_type_check (void);

Funktion Diese Funktion ermittelt, welches Betriebssystem aktiv ist. Der Rückgabewert kann drei Werte annehmen:
0 = Mini-Betriebssystem (läuft im ROM)
1 = ROM-Betriebssystem (aus dem ROM ins RAM geladen).
2 = RAM-Betriebssystem (vom PC ins RAM geladen).

ml8_prog_in_rom Prüfe, ob Programm im ROM ist

Pascal FUNCTION ml8_prog_in_rom (pgm: WORD): BOOLEAN;

C byte ml8_prog_in_rom (ushort pgm);

Funktion Diese Funktion prüft, ob ein Programm im ROM enthalten ist. Wenn das Programm im ROM ist, ist der Rückgabewert TRUE andernfalls FALSE.

Parameter *pgm*: Nummer des Programms.

ml8_get_selected_card Ermittle Nr. der angewählten Karte

Pascal FUNCTION ml8_get_selected_card: WORD;

C ushort ml8_get_selected_card (void);

Funktion Diese Funktion liefert die aktuell angewählte Kartennummer.

ml8_get_lib_version — Ermittle den Versions- und Datecode der Bibliothek

Pascal PROCEDURE ml8_get_lib_version (VAR version, date: LONGINT);

C VOID ml8_get_lib_version (ULONG* version, ULONG* date);

Funktion Die Prozedur liefert den Versionscode und den Datecode der verwendeten Bibliothek.

Parameter *version*: 32-Bit Versionscode[1] der Bibliothek.

date: 32-Bit Datecode[1] der Bibliothek.

ml8_get_osx_version — Ermittle den Versions- und Datecode des Betriebssystems

Pascal PROCEDURE ml8_get_osx_version (VAR version, date: LONGINT);

C VOID ml8_get_osx_version (ULONG* version, ULONG* date);

Funktion Die Prozedur liefert den Versionscode und den Datecode des auf der MODULAR-4/486 verwendeten Betriebssystems

Parameter *version*: 32-Bit Versionscode[1] des Betriebssystem.

date: 32-Bit Datecode[1] der Betriebssystem.

---

[1] Bedeutung der Codes siehe Kapitel 6.5.

ml8_init_io — Initialisiere Basiskarte und Module

Pascal PROCEDURE ml8_init_io (microslot, option: BYTE);

C void ml8_init_io (byte microslot, byte option);

Funktion Die Prozedur initialisiert alle I/O-Einheiten der angegebenen Hardware gemäß den Eintragungen im jeweiligen EEPROM.

Parameter *microslot*: Gibt an, welche Hardware initialisiert werden soll:
0: Basiskarte
1 bis 10: Modul auf dem entsprechenden Steckplatz.

option: Gibt an, ob in jedem Fall initialisiert wird (= 1), oder nur dann, wenn es laut Bit-0 im WORT-1 des EEPROMs erlaubt ist (Bit-0 = 1, *option = 0*).

ml8_change_timeout — Ändere den Timeout-Wert

Pascal PROCEDURE ml8_change_timeout (tenthsec: WORD);

C void ml8_change_timeout (ushort tenthsec);

Funktion Die Prozedur setzt die Timeout-Zeit für die angewählte Karte.

Parameter *tenthsec*: Timeout-Wert in Zehntelsekunden (s. **ml8_reset**).

6.4.2. Laden von Echtzeitprogrammen auf die MODULAR-4

ml8_transfer_and_install Installiere Programm auf der Karte

Pascal FUNCTION ml8_transfer_and_install (pgmname: str80; usage, tasknr, pgmnr, irqnr: WORD; flags, datasize: LONGINT): WORD;

C ushort ml8_transfer_and_install (char *pgmname, ushort usage, ushort tasknr, ushort pgmnr, ushort irqnr, ulong flags, ulong datasize);

Funktion Diese Funktion lädt ein Echtzeitprogramm (z.B. von der Festplatte), transferiert es zur MODULAR-4 und installiert es dort. Der Rückgabewert ist Null, wenn das Programm erfolgreich geladen und installiert wurde. Andernfalls gibt das höherwertige Byte die Fehlerklasse und das niederwertige Byte den Fehlercode an (siehe Anhang E).

Parameter

pgmname: Zeiger auf eine Variable, die den Dateinamen des zu ladenden Programms enthält. Dabei ist die Angabe eines Pfades zulässig. C-Programmierer müssen bei Angabe eines Pfades beachten, daß in C ein Backslash (\) verdoppelt werden muß.

usage: Reserviert, immer = 0 setzen.

tasknr: Tasknummer, unter der das Programm installiert werden soll.

pgmnr: Programmnummer des Programms. Dieser Wert muß mit der in der PDT eingestellten Nummer des Echtzeitprogramms übereinstimmen.

irqnr: Interrupt, den das Programm nutzen soll (nur für DI- und II-Tasks).

flags: Dieser Parameter spezifiziert Installationsoptionen, die der nachfolgenden Tabelle zu entnehmen sind. Das Flag kann einfach gebildet werden, indem die aus der Spalte Wert ermittelten Zahlen der gewünschten Optionen addiert werden.

datasize: Dieser Parameter gibt an, wie groß der Datenbereich der Task sein soll.

Hinweis Tasktyp (in *flags*), Interruptnummer und Datenbereichsgröße werden in der Regel von den Echtzeitprogrammen in der PDT mit Werten vorbesetzt. Ein Echtzeitprogramm kann durch entsprechende Flags in der PDT erzwingen, daß diese Einstellungen verwendet werden, unabhängig von den bei **ml8_transfer_and_install** übergebenen Werten. Sofern es das Echtzeitprogramm zuläßt, kann der Installierungsbefehl Tasktyp und Interruptnummer einstellen (*flags* Bit-3 = 1) oder die vom Programm voreingestellten Werte übernehmen (*flags* Bit-3 = 0). Die Größe des Datenbereichs kann der Installierungsbefehl - wiederum nur wenn es das Echtzeitprogramm zuläßt - aus drei Angaben in der PDT auswählen (minimaler Datenbereich, maximaler Datenbereich und Datenbereichsgröße) oder mit *datasize* frei bestimmen (*flags* Bits 9 und 10).

Die Bedeutung des Parameters *flags*:

Bit	**Wert**	**Bedeutung**
2-0		Tasktyp-Festlegung, wenn Bit 3 des Flags der PDT = 0 ist und Bit 3 des Parameters *flags* = 1 ist
	0	(= 000b): NI-Task (Nicht-Interrupt-Task)
	1	(= 001b): II-Task (Indirekte Interrupt-Task)
	2	(= 010b): DI-Task (Direkte Interrupt-Task)
	3	(= 011b): TI-Task (Timer-initiierte Task)
3		Wer legt Tasktyp und Interrupt-Nummer fest?
	0	(= 0b): PDT legt Tasktyp und Interrupt-Nummer fest
	8	(= 1b): wenn Bit 3 des Flags der PDT Null ist, dann wird der Tasktyp und die Interrupt-Nummer durch die Parameter von **ml8_transfer_and_install** festgelegt
5-4		Privilegstufe des Programms
	0	(= 00b): höchste Privilegstufe (Systemprogramme)
	16	(= 01b): zweithöchste Privilegstufe
	32	(= 10b): dritthöchste Privilegstufe
	48	(= 11b): niedrigste Privilegstufe (Anwendungsprogramme)

Bit	Wert	Bedeutung
8-6		Programmformat
		Für Ihre Hochsprachen-'EXE'-Dateien muß das Programmformat "EXE not relocated" (110b) angegeben werden.

	flagbits	Wo?	Format	Adresse	Typ
0	(= 000b):	RAM	PDT (tiny)	Anfang PDT	Assembler
256	(= 100b):	RAM	PDT (large)	Anfang PDT	Assembler
384	(= 110b):	RAM	EXE not reloc.	EXE-Header	C / Pascal
128	(= 010b):	RAM	EXE relocated	START_UP Code	Reserviert
64	(= 001b):	ROM	PDT	wird ignoriert	Reserviert

Bit	Wert	Bedeutung
10-9		Wie wird Größe von Datenbereich festgelegt? Die Einstellung ist nur wirksam, wenn Bit 9 im PDT-Flag = 0 ist

	flagbits	Größe richtet sich nach	fix/variabel
1536	(= 11b):	"Größe" in PDT	fix
1024	(= 10b):	"Minimum" in PDT	fix
512	(= 01b):	"Maximum" in PDT	fix
0	(= 00b):	Wert von *datasize*	variabel

Bit	Wert	Bedeutung
11		Auto-Init Prozedur aufrufen?
	0	(= 0b): Auto-Init Prozedur nicht aufrufen
	2048	(= 1b): Auto-Init Prozedur aufrufen
12		Task nach dem Installieren sofort aktivieren?
	0	(= 0b): Task nicht aktivieren
	4096	(= 1b): Task aktivieren
13-31		Reserviert

ml8_transfer_pgm — Lade Programm in den Speicher der MODULAR-4

Pascal

```
FUNCTION ml8_transfer_pgm (task, usage: WORD; pgmname: str80;
VAR loadadr:LONGINT; VAR prepadr: LONGINT; VAR flags:
LONGINT): WORD;
```

C

```
ushort ml8_transfer_pgm (ushort task, ushort usage, char *pgmname,
ulong *loadadr, ulong *prepadr, ulong *flags);
```

Funktion Diese Funktion lädt ein Programm in den Speicher der MODULAR-4 Karte. Bei erfolgreichem Programmtransfer ist der Rückgabewert der Funktion Null. Andernfalls liefert die Funktion die Fehlerursache zurück (siehe Anhang F). Wenn eine Programmdatei mit der Dateinamenerweiterung '.EXE' übergeben wird, so sorgt die Funktion beim Übertragen des Programms automatisch für die erforderliche Relokierung der Segmentadressen innerhalb des Programms.

Parameter

task: Nummer der Task, unter der das Programm installiert werden soll (nur zu Protokollzwecken).

usage: Reserviert, immer = 0 setzen.

pgmname: siehe **ml8_transfer_and_install**.

loadadr: Zeiger auf eine Variable, in die die physikalische Adresse, ab der sich das Programm im Speicher der MODULAR-4 befindet, eingetragen wird.

prepadr: Zeiger auf eine Variable, in die die physikalische Adresse, ab der sich der Prepare-Teil des Programms im Speicher der MODULAR-4 befindet, eingetragen wird.

flags: Zeiger auf eine Variable, die die Installationsoptionen enthält (siehe **ml8_transfer_and_install**).

Hinweis Verwenden Sie **ml8_transfer_pgm** nur dann, wenn Sie ein Programm mehrfach installieren wollen. Um ein Programm einfach zu installieren, verwenden Sie bitte **ml8_transfer_and_install**. !

ml8_install_task — Installiere Programm

Pascal PROCEDURE ml8_install_task (task, programnr, interruptnr: WORD; flags, datasize, adr: LONGINT);

C void ml8_install_task (ushort task, ushort programnr, ushort interruptnr, ulong flags, ulong datasize, ulong adr);

Funktion Diese Funktion installiert ein bereits im Speicher der MODULAR-4 Karte befindliches Echtzeitprogramm. Beim Installieren eines EXE-Programms, muß in *flags* als Programmformat **EXE relocated** eingestellt werden.

Parameter *tasknr, pgmnr, irqnr, flags, datasize*: siehe **ml8_transfer_and_install**

adr: Dieser Parameter gibt, abhängig von *flags***,** entweder die Adresse der PDT, die Start-Up-Adresse oder die Adresse des EXE-Headers des Programms an. In der Regel wird hier die von **ml8_transfer_pgm** im Parameter *prepadr* zurückgelieferte Adresse eingetragen.

! Hinweis Verwenden Sie **ml8_install_task** nur dann, wenn Sie ein Programm mehrfach installieren wollen. Um ein Programm einfach zu installieren, verwenden Sie besser **ml8_transfer_and_install**.

6.4.3. Taskbefehle

Die meisten der in den folgenden Kapiteln beschriebenen Funktionen enthalten den Parameter *task*. Dieser spezifiziert die Nummer, unter dem das Betriebssystem auf der MODULAR-4 Karte das zugehörige Echtzeitprogramm verwaltet. Er wird in den Parameterbeschreibungen nicht jedesmal aufgeführt.

6.4.3.1. Taskinformationen abfragen

ml8_get_task_info **Lies Systeminformationen einer Task**

Pascal PROCEDURE ml8_get_task_info (task, callnr: WORD;
VAR result: LONGINT);

C void ml8_get_task_info (ushort task, ushort callnr, ulong *result);

Funktion Diese Prozedur liefert Systeminformationen über eine Task.

Parameter *callnr*: Spezifikation der gewünschten Information (s. Anhang H)

result: Zeiger auf eine Variable, in die das Ergebnis eingetragen wird.

ml8_get_task_status — Ermittle, ob Task aktiviert ist

Pascal FUNCTION ml8_get_task_status (task: WORD): BYTE;

C byte ml8_get_task_status (ushort task);

Funktion Mit Hilfe dieser Prozedur können Sie die Anzahl der Aktivierungen einer Task ermitteln. Ist das Funktionsergebnis = 0, dann ist die Task nicht aktiviert.

ml8_get_task_number — Ermittle Tasknummer

Pascal FUNCTION ml8_get_task_number (prog_nr: WORD;
VAR install_nr: WORD): WORD;

C ushort ml8_get_task_number (ushort prog_nr, ushort *install_nr);

Funktion Die Funktion liefert die Tasknummer, unter der ein Programm installiert ist.

Parameter *prog_nr*: In der PDT eingetragene Nummer des Programms.

install_nr: Wenn ein Programm mehrfach installiert ist, dann kann in diesem Parameter angegeben werden, von welcher Instanz die Tasknummer ermittelt werden soll.

***install_nr* vor Aufruf**	***install_nr* nach Aufruf**	**Bedeutung**
0	0	Das Programm ist nicht installiert, der Rückgabewert ist ungültig.
0	>0	*install_nr* enthält die Anzahl der Installierungen, der Rückgabewert gibt die höchste Tasknummer an, unter der das Programm installiert ist.
1	0	Das Programm ist nicht installiert, der Rückgabewert ist ungültig.
1	1	Das Programm ist installiert, der Rückgabewert gibt die niedrigste Tasknummer an, unter der das Programm installiert ist.
2 bis 1024	0	Das Programm ist nicht installiert, der Rückgabewert ist ungültig.

***install_nr* vor Aufruf**	***install_nr* nach Aufruf**	**Bedeutung**
2 bis 1024	= *install_nr* vor Aufruf	Der Rückgabewert enthält die Tasknummer der Task, die sich, wenn man die Tasks nach ihrer Nummer ordnet, an der Stelle *install_nr* befindet.
2 bis 1024	< *install_nr* vor Aufruf	Das Programm wurde nur sooft installiert, wie in *install_nr* zurückgegeben wurde. Der Rückgabewert gibt die höchste Tasknummer an, unter der das Programm installiert ist.

ml8_get_int_task — Ermittle Tasktyp und -nummer

Pascal PROCEDURE ml8_get_int_task (inter: WORD; VAR task: WORD VAR dummy: BYTE);

C void ml8_get_int_task (ushort inter, ushort *task, byte *dummy);

Funktion Diese Prozedur ermittelt, welche Tasknummer die Task hat, die unter einem Interrupt installiert ist.

Parameter *inter*: Interruptnummer.

task: Zeiger auf eine Variable, in die die Nummer der Task eingetragen wird, die unter dem Interrupt *inter* installiert ist. Wenn der Interrupt unbenutzt ist, wird dieser Wert = -1 (ffffh) gesetzt.

dummy: Zur Zeit nicht benutzt.

ml8_get_pgm_installed — Ermittle Programmnummer einer Task

Pascal FUNCTION ml8_get_pgm_installed (task: WORD): WORD;

C ushort ml8_get_pgm_installed (ushort task);

Funktion Diese Funktion liefert die in der PDT eingetragene Nummer des Programms zurück, das unter der Tasknummer installiert ist. Wenn das Funktionsergebnis = 0 ist, dann ist kein Programm installiert.

6.4.3.2. Tasks aktivieren und deaktivieren

ml8_wakeup_task — Aktiviere eine Task

Pascal PROCEDURE ml8_wakeup_task (task: WORD);

C void ml8_wakeup_task (ushort task);

Funktion Diese Prozedur aktiviert eine Task. NI-Tasks können auch mehrfach aktiviert werden, wodurch sie gegenüber einfach aktivierten Tasks häufiger aufgerufen werden. Bei Interrupt-Tasks (DI- und II-Tasks) wird der zugehörige Interrupt demaskiert. Für TI-Tasks gibt es eine eigene Aktivierungsroutine (**ml8_wakeup_ti_task**).

ml8_sleep_task — Deaktiviere eine Task

Pascal PROCEDURE ml8_sleep_task (task: WORD);

C void ml8_sleep_task (ushort task);

Funktion Diese Prozedur deaktiviert eine Task. Eine mehrfach aktivierte NI-Task muß auch mehrfach wieder deaktiviert werden. Der Befehl bricht eine laufende Task ab, ein gerade laufender Aufruf einer Task wird aber ordnungsgemäß beendet. Bei Interrupt-Tasks (DI- und II-Tasks) wird der zugehörige Interrupt maskiert.

ml8_wakeup_ti_task — Aktiviere eine TI-Task

Pascal PROCEDURE ml8_wakeup_ti_task (task: WORD; priority: BYTE; count, interval, holdoff: LONGINT);

C void ml8_wakeup_ti_task (ushort task, byte priority, ulong count, ulong interval, ulong holdoff);

Funktion Mit dieser Funktion wird der Zeitplan einer TI-Task festgelegt und die Task aktiviert.

Parameter *priority*: Priorität der Task: Wertebereich 0 (höchste Priorität) bis 255 (niedrigste Priorität).

count: Gibt an, wie oft die TI-Task aufgerufen werden soll, bis sie deaktiviert wird. Wenn *count* = 0, läuft die Task so lange, bis sie deaktiviert wird.

interval: Zeitintervall zwischen zwei Aufrufen der Task, angegeben in Timer-Tics.

holdoff: Zeit (gemessen in Timer-Tics), nach der die Task zum erstenmal, nach ihren Aktivierung aufgerufen wird.

6.4.3.3. Taskfunktionen aufrufen

ml8_call_func — Aufruf einer Funktion einer Task

Pascal
PROCEDURE ml8_call_func (task, func, outsize: WORD;
VAR outdata_var; maxinsize: WORD; VAR insize: WORD;
VAR indata_var; VAR error: BYTE);

C
void ml8_call_func (ushort task, ushort func, ushort outsize,
void *outdata_var, ushort maxinsize, ushort *insize, void *indata_var,
byte *error);

Funktion
Mit dieser Prozedur kann eine beliebige Funktion aus einer auf der MODULAR-4 installierten Task aufgerufen werden. **Siehe auch Kapitel 9.3.1.**

Parameter

func: Nummer der Funktion.

outsize: Anzahl der Bytes, die an die Funktion der Task übergeben werden sollen (maximal 256 Byte).

outdata_var: Zeiger auf die Daten, die an die Funktion übergeben werden sollen.

maxinsize: Maximale Anzahl an Bytes, die die aufgerufene Echtzeitfunktion an den PC zurückliefern darf bzw. soll (maximal 256 Byte).

insize: Zeiger auf eine Variable, in die die tatsächlich zurückgelieferte Anzahl an Bytes eingetragen wird.

indata_var: Zeiger auf eine Variable, in die die Rückgabedaten der Funktion eingetragen werden.

error: Zeiger auf eine Variable, in die ein Fehlerstatus eingetragen wird. Ist dieser Wert = 0, ist kein Fehler bei der Ausführung der Funktion auf der MODULAR-4 Karte aufgetreten. Bei einem Wert größer Null handelt es sich entweder um eine Betriebssystemmeldung (siehe Anhang F) oder um eine spezielle Meldung der aufgerufenen Funktion.

ml8_call_proc — Aufruf einer Prozedur einer Task

Pascal PROCEDURE ml8_call_proc (task, procnr: WORD);

C void ml8_call_proc (ushort task, ushort procnr);

Funktion Diese Prozedur ist ein Sonderfall der Prozedur **ml8_call_func**: Es werden keine Daten an die aufgerufene Prozedur übergeben oder von ihr übernommen.

Parameter *procnr*: Nummer der Prozedur.

6.4.4. Zugriff auf die Parameterbereiche von Tasks

Zugriffe auf Variable im Parameterbereich einer Task erfolgen immer unter Angabe ihrer Adresse (relativ zum Anfang des Parameterbereichs, gezählt in Byte). Diese wird bei allen nachfolgenden Bibliotheksfunktionen im Parameter *start* angegeben. Sollen mehrere Parameter mit einem Befehl gelesen oder geschrieben werden, spezifiziert dieser Wert den ersten zu lesenden oder zu schreibenden Parameter.

ml8_get_par_sema
ml8_release_par_sema — Parameterbereichs-Semaphore

Pascal FUNCTION ml8_get_par_sema (task: WORD): WORD;

PROCEDURE ml8_release_par_sema (task: WORD);

C ushort ml8_get_par_sema (ushort task);

void ml8_release_par_sema (ushort task);

Funktion Mit diesen Funktionen wird der Zugriff auf den Parameterbereich einer Task gesteuert. Die Blockzugriffe auf den Parameterbereich einer Task sind durch Interrupts unterbrechbar. Damit also ein konsistenter Zugriff gewährleistet ist, muß ein Programm, das auf einen Parameterbereich zugreifen will, zunächst versuchen mit **ml8_get_par_sema** die Semaphore für den Parameterbereich zu bekommen. Gelingt dies, kann das Programm beliebig Daten in den Parameterbereich schreiben oder lesen. Hat sich bereits eine andere Anwendung die Semaphore geholt, liefert die Funktion eine Fehlermeldung und darf anschließend nicht auf den Parameterbereich zugreifen.

! Hinweis Das Betriebssystem prüft bei den Parameterbereichszugriffen selber nicht, ob die Semaphore frei ist. Um konsistente Daten sicherzustellen müssen alle Anwendungen vor einem Zugriff auf den Parameterbereich einer Task zunächst die Semaphore abprüfen!

Mit **ml8_release_par_sema** wird die Semaphore wieder frei gegeben.

Parameter *task* Nummer der Task

ml8_read_par_byte
ml8_read_par_word
ml8_read_par_dword Lies Parameter einer Task

Pascal FUNCTION ml8_read_par_byte (task, start: WORD): BYTE;

FUNCTION ml8_read_par_word (task, start: WORD): WORD;

PROCEDURE ml8_read_par_dword (task, start: WORD;
VAR data: LONGINT);

C byte ml8_read_par_byte (ushort task, ushort start);

ushort ml8_read_par_word (ushort task, ushort start);

void ml8_read_par_dword (ushort task, ushort start, ulong *data);

Funktion Die Funktionen lesen ein, zwei oder vier Byte aus dem Parameterbereich einer Task. Beachten Sie, daß der Wert eines Doppelwortes nicht als Funktionsergebnis sondern in *data* zurückgegeben wird. Die Zugriffe sind in sich konsistent. Daher braucht die Semaphore nicht vorher geholt zu werden.

Parameter *data*: Zeiger auf eine Variable, die das Ergebnis von **ml8_read_par_dword** aufnimmt.

ml8_read_par_block — Lies Parameterblock einer Task

Pascal
PROCEDURE ml8_read_par_block (task, start, size: WORD;
VAR data_var);

C
void ml8_read_par_block (ushort task, ushort start, ushort size,
void *data_var);

Funktion
Mit dieser Prozedur kann ein Block von Parametern einer Task gelesen werden.

Parameter *size*: Größe des Blocks (in Byte, z. B. mit sizeof (...))

data_var: Zeiger auf einen Variablenbereich, in den die gelesenen Daten eingetragen werden.

Hinweis
Während ein Block gelesen wird, sind auf der Karte alle Interrupts gesperrt, sofern nicht mit **ml8_set_macro_interruptible** die Makrobefehle unterbrechbar geschaltet wurden. Außerdem sollte vor jedem Zugriff mit **ml8_get_par_sema** die entsprechende Semaphore des Parameterbereichs geholt werden, um einen konsistenten Zugriff zu gewährleisten.

ml8_write_par_byte
ml8_write_par_word
ml8_write_par_dword — Schreibe Parameter einer Task

Pascal
PROCEDURE ml8_write_par_byte (task, start: WORD; data: BYTE);

PROCEDURE ml8_write_par_word (task, start: WORD;
data: WORD);

PROCEDURE ml8_write_par_dword (task, start: WORD;
data: LONGINT);

C
void ml8_write_par_byte (ushort task, ushort start, byte data);

void ml8_write_par_word (ushort task, ushort start, ushort data);

void ml8_write_par_dword (ushort task, ushort start, ulong data);

Funktion Mit diesen Prozeduren können ein, zwei oder vier Byte in den Parameterbereich einer Task geschrieben werden. Die Zugriffe sind in sich konsistent. Daher braucht die Semaphore nicht vorher geholt zu werden.

Parameter *data*: Wert, der geschrieben werden soll.

ml8_write_par_block Schreibe Parameterblock einer Task

Pascal PROCEDURE ml8_write_par_block (task, start, size: WORD;
VAR data_var);

C void ml8_write_par_block (ushort task, ushort start, ushort size,
void *data_var);

Funktion Mit dieser Prozedur kann ein Block von Parametern einer Task gesetzt werden.

Parameter *size*: Größe des Blocks (in Byte, z. B. mit sizeof (...))

data_var: Zeiger auf den Variablenbereich, in dem die Daten stehen, die geschrieben werden.

! Hinweis Während ein Block geschrieben wird, sind auf der Karte alle Interrupts gesperrt, sofern nicht mit **ml8_set_macro_interruptible** die Makrobefehle unterbrechbar geschaltet wurden. Außerdem sollte vor jedem Zugriff mit **ml8_get_par_sema** die entsprechende Semaphore des Parameterbereichs geholt werden, um einen konsistenten Zugriff zu gewährleisten.

6.4.5. Zugriff auf die Datenbereiche von Tasks

Der Zugriff auf den Datenbereich einer Task geschieht fast immer über den Schreib- bzw. Lesezeiger der Task. Diese Zeiger werden vom Betriebssystem verwaltet und bei jedem Zugriff automatisch um die Anzahl der gelesenen bzw. geschriebenen Bytes inkrementiert. Ausnahme: **ml8_get_data_offs** und **ml8_set_data_offs**, hier werden keine Zeiger verwendet.

ml8_get_data_sema
ml8_release_data_sema — Datenbereichs-Semaphore

Pascal
FUNCTION ml8_get_data_sema (task: WORD): WORD;
PROCEDURE ml8_release_data_sema (task: WORD);

C
ushort ml8_get_data_sema (ushort task);
void ml8_release_data_sema (ushort task);

Funktion
Mit diesen Funktionen wird der Zugriff auf den Datenbereich einer Task gesteuert. Diese Zugriffe sind durch Interrupts unterbrechbar. Damit ein konsistenter Zugriff gewährleistet ist, muß ein Programm, das auf einen Datenbereich zugreifen will, zunächst versuchen, mit **ml8_get_data_sema** die Semaphore für den Datenbereich zu bekommen. Gelingt dies, kann das Programm beliebig auf den Datenbereich zugreifen. Hat sich bereits eine andere Anwendung die Semaphore geholt, liefert die Funktion eine Fehlermeldung.

Hinweis
Das Betriebssystem prüft bei den Datenbereichszugriffen selber nicht, ob die Semaphore frei ist. Um konsistente Daten sicherzustellen, müssen alle Anwendungen vor einem Zugriff auf den Datenbereich einer Task zunächst die Semaphore abprüfen! !

Mit **ml8_release_data_sema** wird die Semaphore wieder frei gegeben.

Parameter
task Nummer der Task

ml8_get_data_offs
ml8_set_data_offs — Datenbereichszugriff

Pascal

PROCEDURE ml8_get_data_offs (task: WORD; ulstart: LONGINT; VAR size: WORD; VAR data);

PROCEDURE ml8_set_data_offs (task: WORD; ulstart: LONGINT; VAR size: WORD; VAR data);

C

void ml8_get_data_offs (ushort task, ulong ulstart, ushort *size, void *data);

void ml8_set_data_offs (ushort task, ulong ulstart, ushort *size, void *data);

Funktion

Mit diesen Funktionen wird auf den Datenbereich einer Task zugegriffen. Vor jedem Zugriff sollte mit **ml8_get_data_sema** die entsprechende Semaphore geholt werden, um einen konsistenten Zugriff zu gewährleisten.

Parameter

task Nummer der Task

ulstart Offset (in Bytes) bezogen auf den Anfang des Datenbereiches, ab dem die Daten gelesen bzw. geschrieben werden sollen.

size Zeiger auf eine Variable, in der steht wie viele Bytes (maximal) gelesen bzw. geschrieben werden sollen. Nach Beendigung der Funktion ist dort eingetragen, wie viele Bytes tatsächlich gelesen bzw. geschrieben worden sind. Die Größe des Datenbereiches kann aus der TDT mit Hilfe der Funktion **ml8_get_task_info** ausgelesen werden.

data Zeiger auf die Daten

ml8_reset_r_pointer — Setze Daten-Lesezeiger zurück

Pascal

PROCEDURE ml8_reset_r_pointer (task: WORD);

C

void ml8_reset_r_pointer (ushort task);

Funktion

Diese Prozedur setzt den Daten-Lesezeiger einer Task an den Anfang ihres Datenbereichs.

ml8_move_r_pointer Verschiebe Daten-Lesezeiger

Pascal PROCEDURE ml8_move_r_pointer (task: WORD;
offset: LONGINT);

C void ml8_move_r_pointer (ushort task, long offset);

Funktion Diese Prozedur verschiebt den Daten-Lesezeiger einer Task.

Parameter *offset*: Anzahl an Bytes, um die der Lesezeiger verschoben werden soll. Der Wert kann auch negativ sein.

ml8_reset_w_pointer Setze Daten-Schreibzeiger zurück

Pascal PROCEDURE ml8_reset_w_pointer (task: WORD);

C void ml8_reset_w_pointer (ushort task);

Funktion Diese Prozedur setzt den Daten-Schreibzeiger einer Task an den Anfang ihres Datenbereichs.

ml8_move_w_pointer Verschiebe Daten-Schreibzeiger

Pascal PROCEDURE ml8_move_w_pointer (task: WORD;
offset: LONGINT);

C void ml8_move_w_pointer (ushort task, long offset);

Funktion Diese Prozedur verschiebt den Daten-Schreibzeiger einer Task.

Parameter *offset*: Anzahl an Bytes, um die der Lesezeiger verschoben werden soll. Der Wert kann auch negativ sein.

ml8_read_data_byte
ml8_read_data_word
ml8_read_data_dword — Lies Datenbereich einer Task

Pascal

```
FUNCTION ml8_read_data_byte (task: WORD): BYTE;

FUNCTION ml8_read_data_word (task: WORD): WORD;

PROCEDURE ml8_read_data_dword (task: WORD;
VAR data: LONGINT);
```

C

```
byte ml8_read_data_byte (ushort task);

ushort ml8_read_data_word (ushort task);

void ml8_read_data_dword (ushort task, ulong *data);
```

Funktion

Die Funktionen lesen ein, zwei oder vier Byte aus dem Datenbereich einer Task. Es wird ab dem Lesezeiger der Task gelesen. Danach wird der Lesezeiger der Task um die entsprechende Anzahl Byte erhöht. Beachten Sie, daß der Wert eines Doppelwortes (vier Byte) nicht als Funktionsergebnis, sondern in *data* zurückgegeben wird. Die Zugriffe sind in sich konsistent. Daher braucht die Semaphore nicht vorher geholt zu werden.

Parameter

data: Zeiger auf eine Variable, die das Ergebnis von **ml8_read_data_dword** aufnimmt.

ml8_read_data_block — Lies Block aus Datenbereich

Pascal

```
PROCEDURE ml8_read_data_block (task, size: WORD;
VAR data_var);
```

C

```
void ml8_read_data_block (ushort task, ushort size, void *data_var);
```

Funktion

Diese Prozedur liest einen Datenblock aus dem Datenbereich einer Task. Es wird ab dem Lesezeiger der Task gelesen. Der Lesezeiger der Task wird anschließend um die gelesene Anzahl Bytes erhöht.

Parameter

size: Größe des Blocks (in Byte, z. B. mit sizeof (...))

data_var: Zeiger auf einen Variablenbereich, in den die gelesenen Daten eingetragen werden.

Hinweis Während des Lesens eines Blockes sind auf der Karte alle Interrupts gesperrt, sofern nicht mit **ml8_set_macro_interruptible** die Makrobefehle unterbrechbar geschaltet wurden. Außerdem sollte vor jedem Zugriff mit **ml8_get_par_sema** die entsprechende Semaphore des Datenbereichs geholt werden, um einen konsistenten Zugriff zu gewährleisten. !

ml8_read_data — Lies Block direkt aus Datenbereich

Pascal
PROCEDURE ml8_read_data (task: WORD; rel_ofs : LONGINT; size: WORD; VAR data_var);

C
void ml8_read_data (ushort task, ulong rel_ofs, ushort size, void* data_var);

Funktion
Diese Prozedur liest einen Datenblock aus dem Datenbereich einer Task. Diese Funktion kombiniert die Aufrufe von ml8_reset_r_pointer, ml8_move_r_pointer und ml8_read_data_block (schnellerer Zugriff).

Parameter
rel_ofs: Startadresse des Blocks (relativ zum Datenbereichsanfang)

size: Größe des Blocks (in Byte, z.B. mit sizeof (...))

data_var: Zeiger auf einen Variablenbereich, in den die gelesenen Daten eingetragen werden.

ml8_write_data_byte
ml8_write_data_word
ml8_write_data_dword — Schreibe Daten in den Datenbereich

Pascal
PROCEDURE ml8_write_data_byte (task: WORD; data: BYTE);

PROCEDURE ml8_write_data_word (task: WORD; data: WORD);

PROCEDURE ml8_write_data_dword (task: WORD; data: LONGINT);

C
void ml8_write_data_byte (ushort task, byte data);

void ml8_write_data_word (ushort task, ushort data);

void ml8_write_data_dword (ushort task, ulong data);

Funktion Diese Prozeduren schreiben ein, zwei oder vier Byte ab dem Schreibzeiger einer Task in ihren Datenbereich. Der Schreibzeiger wird danach automatisch um die entsprechende Anzahl von Byte erhöht. Die Zugriffe sind in sich konsistent. Daher braucht die Semaphore nicht vorher geholt zu werden.

Parameter *data*: Wert, der geschrieben werden soll.

ml8_write_data_block Schreibe Block in den Datenbereich

Pascal PROCEDURE ml8_write_data_block (task, size: WORD;
VAR data_var);

C void ml8_write_data_block (ushort task, ushort size, void *data_var);

Funktion Diese Prozedur schreibt einen Block von Daten ab dem Schreibzeiger einer Task in ihren Datenbereich. Der Schreibzeiger der Task wird danach automatisch um die entsprechende Anzahl von Byte erhöht.

Parameter *size*: Größe des Blocks (in Byte, z. B. mit sizeof (...))

data_var: Zeiger auf den Variablenbereich, in dem die Daten stehen, die geschrieben werden.

! Hinweis Während ein Block geschrieben wird, sind auf der Karte alle Interrupts gesperrt, sofern nicht mit **ml8_set_macro_interruptible** die Makrobefehle unterbrechbar geschaltet wurden. Außerdem sollte vor jedem Zugriff mit **ml8_get_par_sema** die entsprechende Semaphore des Datenbereichs geholt werden, um einen konsistenten Zugriff zu gewährleisten.

ml8_write_data Schreibe Block direkt in den Datenbereich

Pascal PROCEDURE ml8_write_data (task: WORD; rel_ofs : LONGINT; size: WORD; VAR data_var);

C void ml8_write_data (ushort task, ulong rel_ofs, ushort size, void* data_var);

Funktion Diese Prozedur schreibt einen Datenblock direkt in den Datenbereich einer Task. Diese Funktion kombiniert die Aufrufe von ml8_reset_w_pointer, ml8_move_w_pointer und ml8_write_-data_block (schnellerer Zugriff).

Parameter

rel_ofs: Startadresse des Blocks (relativ zum Datenbereichsanfang)

size: Größe des Blocks (in Byte, z. B. mit sizeof (...))

data_var: Zeiger auf den Variablenbereich, in dem die Daten stehen, die geschrieben werden.

6.4.6. Datenpuffer

Das Betriebssystem der MODULAR-4/486 stellt ringförmig organisierte Datenpuffer zur Verfügung. Jeder Puffer erhält bei der Erzeugung eine eindeutige Nummer. Unter Angabe dieser Nummer (wird jeweils im Parameter *buffer* an die Bibliotheksfunktionen übergeben) kann auf die Pufferdaten zugegriffen werden.

Das Betriebssystem übernimmt die gesamte Verwaltung des Ringpuffers. Die Schreib- und Lesezugriffe auf einen Datenpuffer können aus beliebigen Tasks auf der Karte genauso wie vom PC aus erfolgen. Das Betriebssystem stellt dabei sicher, daß ein Puffer in der Zeit, in der eine Task oder der PC daraus liest bzw. hinein schreibt, von keiner anderer Task oder vom PC gelesen bzw. beschrieben werden kann. In einem solchen Konflikt gibt die Lese- bzw. Schreibprozedur die Fehlermeldung zurück, daß der Puffer im Moment nicht verfügbar ist. Der gewünschte Aufruf muß zu einem späteren Zeitpunkt wiederholt werden.

Das Schreiben von Daten geschieht mit einem vom Betriebssystem verwalteten Schreibzeiger. Er wird nach jedem Schreibbefehl um die Anzahl geschriebener Byte weitergeschoben. Wenn mehr Daten geschrieben werden sollen, als Platz zur Verfügung steht, wird eine Fehlermeldung zurückgeliefert.

Das Lesen von Daten geschieht ebenfalls über einen Zeiger, der automatisch um die Anzahl der gelesenen Byte verschoben wird. Die Bereiche des Ringpuffers, die ausgelesen wurden, werden als frei markiert und stehen wieder für das Schreiben von Daten zur Verfügung.

ml8_create_buffer — Erzeuge einen Datenpuffer

Pascal
FUNCTION ml8_create_buffer (task: WORD; strategy, align: BYTE; usage: WORD; VAR size: LONGINT; VAR handle: LONGINT): BYTE;

C
byte ml8_create_buffer (ushort task, byte strategy, byte align, ushort usage, ulong *size, ulong *handle);

Funktion
Diese Funktion öffnet einen Datenpuffer. Der Rückgabewert ist ungleich Null, wenn ein Fehler aufgetreten ist (siehe Anhang F, High-Byte).

Parameter
task: Nummer der Task, die den Datenpuffer anfordert (nur zu internen Zwecken, kann = 0 gesetzt werden).

strategy: Speicherreservierungs-Strategie (siehe folgende Tabelle, die verwendeten Konstanten sind in ML8BIB.H definiert).

align: Ausrichtung des Anfangs des Datenpuffers. Die Ausrichtung wird in Zweierpotenzen angegeben: 0 = Byte, 1 = Wort, 2 = Doppelwort, 3 = 8 Byte ...

usage: Verwendung des Puffers (derzeit immer = 0 setzen)

size: Größe des Puffers in Byte. Nach dem Funktionsaufruf enthält die Variable die tatsächliche Größe des angelegten Puffers.

handle: Zeiger auf eine Variable, in die die Funktion eine Nummer eintragen kann, mit der auf den erzeugten Puffer zugegriffen werden kann.

Der Parameter *strategy* darf folgende Werte annehmen.

Wert	**Bedeutung**
ALLOC_UP_ABS	Der Speicher wird von der Untergrenze des freien Speichers beginnend reserviert. Wenn nicht genügend Speicher zur Verfügung steht, wird der Datenpuffer nicht angelegt.
ALLOC_UP_MAX	Der Speicher wird von der Untergrenze des freien Speichers beginnend reserviert. Wenn nicht genügend Speicher zur Verfügung steht, wird so viel wie möglich reserviert und die tatsächliche Größe des Puffers in *size* zurückgeliefert.
ALLOC_DOWN_ABS	Der Speicher wird an der Obergrenze des freien Speichers endend reserviert. Wenn nicht genügend Speicher zur Verfügung steht, wird der Datenpuffer nicht angelegt.
ALLOC_DOWN_MAX	Der Speicher wird an der Obergrenze des freien Speichers endend reserviert. Wenn nicht genügend Speicher zur Verfügung steht, wird so viel wie möglich reserviert und die tatsächliche Größe des Puffers in *size* zurückgeliefert.

ml8_clear_buffer — Lösche Inhalt eines Datenpuffers

Pascal PROCEDURE ml8_clear_buffer (buffer: LONGINT);

C void ml8_clear_buffer (ulong buffer);

Funktion Diese Prozedur löscht alle Daten in einem Datenpuffer.

ml8_get_buffer_status — Ermittle den Status eines Datenpuffers

Pascal PROCEDURE ml8_get_buffer_status (buffer: LONGINT;
VAR freesize: LONGINT; VAR usedsize: LONGINT);

C void ml8_get_buffer_status (ulong buffer, ulong *freesize,
ulong *usedsize);

Funktion Diese Prozedur ermittelt, wieviel Speicher im Puffer noch frei ist und wieviel derzeit belegt ist.

Parameter *freesize*: Zeiger auf eine Variable, in die die Funktion die Anzahl der unbenutzten Bytes des Puffers einträgt.

usedsize: Zeiger auf eine Variable, in die die Funktion die Anzahl der benutzten Bytes des Puffers einträgt.

ml8_get_prev_buffer_status — Ermittle den Status des zuletzt angesprochenen Datenpuffers

Pascal PROCEDURE ml8_get_prev_buffer_status (VAR freesize: WORD;
VAR usedsize: WORD);

C void ml8_get_prev_buffer_status (ushort *freesize, ushort *usedsize);

Funktion Diese geschwindigkeitsoptimierte Prozedur ermittelt, wieviel Speicher im zuletzt vom PC angesprochenen Puffer noch frei ist und wieviel Speicher derzeit belegt ist. Ist einer der Parameter ffffh, so ist der aktuelle Wert größer oder gleich ffffh. Dieser muß dann mit Hilfe von **ml8_get_buffer_status** ermittelt werden.

Parameter siehe **ml8_get_buffer_status**

ml8_write_buffer_byte
ml8_write_buffer_word
ml8_write_buffer_dword — Schreibe Daten in einen Puffer

Pascal
FUNCTION ml8_write_buffer_byte (buffer: LONGINT; data: BYTE): BYTE;

FUNCTION ml8_write_buffer_word (buffer: LONGINT; data: WORD): BYTE;

FUNCTION ml8_write_buffer_dword (buffer: LONGINT; data: LONGINT): BYTE;

C
byte ml8_write_buffer_byte (ulong buffer, byte data);

byte ml8_write_buffer_word (ulong buffer, ushort data);

byte ml8_write_buffer_dword (ulong buffer, ulong data);

Funktion
Diese Funktionen schreiben ein, zwei oder vier Byte in einen Puffer. Wenn das Schreiben erfolgreich war, wird eine Null zurückgeliefert. Andernfalls enthält der Rückgabewert eine Fehlermeldung (siehe Anhang F, High-Byte).

Parameter
data: Wert, der geschrieben werden soll.

ml8_set_buffer — Schreibe Block in einen Datenpuffer

Pascal
FUNCTION ml8_set_buffer (buffer: LONGINT; strategy: BYTE; VAR size: WORD; VAR data_var): BYTE;

C
byte ml8_set_buffer (ulong buffer, byte strategy, ushort *size, void *data_var);

Funktion
Mit dieser Funktion wird ein Datenblock in einen Datenpuffer geschrieben. Wenn Daten geschrieben werden konnten, ist der Rückgabewert = 0, andernfalls enthält er eine Fehlermeldung.

Parameter *strategy*: Legt die Strategie des Schreibens fest (s. unten)

size: Zeiger auf eine Variable, die die Größe des zu schreibenden Blocks enthält (maximal 256 Byte). Der Rückgabewert von *size* gibt die Anzahl der tatsächlich geschriebenen Byte an.

data_var: Zeiger auf die zu schreibenden Daten.

Die Bits im Parameter *strategy* haben folgende Bedeutung:

Bit	**Bedeutung**	
	Wenn Bit = 0	Wenn Bit = 1
0	Es wird entweder die in *size* angegebene Anzahl von Byte oder - falls im Puffer nicht genügend Platz ist nichts geschrieben.	Es werden *size* Byte geschrieben. Falls im Puffer nicht genügend Platz ist, werden so viele Byte wie möglich geschrieben.
1	reserviert, immer = 0 setzen	
2	Es wird nur einmal versucht, den Puffer zu beschreiben.	Falls der Puffer nicht sofort bereit ist, wird mehrfach versucht, den Schreibvorgang auszuführen.

ml8_read_buffer_byte
ml8_read_buffer_word
ml8_read_buffer_dword — Lies Daten aus einem Puffer

Pascal

```
FUNCTION ml8_read_buffer_byte (buffer: LONGINT;
VAR data_var: BYTE): BYTE;

FUNCTION ml8_read_buffer_word (buffer: LONGINT;
VAR data_var: WORD): BYTE;

FUNCTION ml8_read_buffer_dword (buffer: LONGINT;
VAR data_var: LONGINT): BYTE;
```

C

```
byte ml8_read_buffer_byte (ulong buffer, byte *data_var);

byte ml8_read_buffer_word (ulong buffer, ushort *data_var);

byte ml8_read_buffer_dword (ulong buffer, ulong *data_var);
```

Funktion Diese Funktionen lesen ein, zwei oder vier Byte aus einem Puffer Wenn das Lesen erfolgreich war, wird eine Null zurückgeliefert. Die gelesenen Daten sind gültig und werden im Puffer gelöscht. Andernfalls enthält der Rückgabewert eine Fehlermeldung (siehe Anhang F, High-Byte). Wenn nicht genügend Zeichen im Puffer sind, werden keine Daten gelesen.

Parameter *data_var*: Zeiger auf eine Variable, in die die gelesenen Daten eingetragen werden.

ml8_get_buffer Lies Datenblock aus einem Puffer

Pascal FUNCTION ml8_get_buffer (buffer: LONGINT; strategy: BYTE; VAR size: WORD; VAR data_var): BYTE;

C byte ml8_get_buffer (ulong buffer, byte strategy, ushort *size, void *data_var);

Funktion Mit dieser Funktion wird ein Datenblock aus einem Datenpuffer gelesen. Wenn Daten gelesen werden konnten, ist der Rückgabewert = 0, andernfalls enthält er eine Fehlermeldung (siehe Anhang F, High-Byte).

Parameter *strategy*: Legt die Strategie des Lesens fest (s. unten)

size: Zeiger auf eine Variable, die die Größe des zu lesenden Blocks enthält (maximal 256 Byte). Der Rückgabewert von *size* gibt die Anzahl der tatsächlich gelesenenen Byte an.

data_var: Zeiger auf einen Variablenbereich, in den die gelesenen Daten eingetragen werden.

Die Bits im Parameter *strategy* haben folgende Bedeutung:

Bit	**Bedeutung**	
	Wenn Bit = 0	Wenn Bit = 1
0	Es werden entweder die in *size* angegebene Anzahl von Byte oder - falls im Puffer nicht genügend vorhanden sind - keine Daten gelesen.	Es werden *size* Byte gelesen. Falls im Puffer nicht genügend Daten zur Verfügung stehen, werden so viele Byte wie möglich gelesen.
1	Die Daten werden aus dem Puffer gelesen, d. h. sie werden aus dem Puffer entfernt.	Die Daten werden aus dem Puffer kopiert und nicht entfernt.
2	Es wird nur einmal versucht, den Puffer zu lesen.	Falls der Puffer nicht sofort bereit ist, wird mehrfach versucht, den Lesevorgang auszuführen.

6.4.7. Zugriffe auf das RAM der MODULAR-4

ml8_allocate_ram Reserviere Speicher auf der MODULAR-4

Pascal
PROCEDURE ml8_allocate_ram (task, usage: WORD;
strategy, align: BYTE; VAR size: LONGINT; VAR adr: LONGINT);

C
void ml8_allocate_ram (ushort task, ushort usage, byte strategy,
byte align, ulong *size, ulong *adr);

Funktion
Diese Prozedur reserviert Speicherplatz auf der MODULAR-4 Karte.

Parameter

task: Tasknummer, für die Speicher reserviert werden soll. Die Angabe dient Protokollzwecken auf der MODULAR-4.

usage: Reserviert, immer = 0 setzen.

strategy: Reservierungsstrategie, mögliche Werte entsprechen denen des Parameters *strategy* in der Funktion **ml8_create_buffer** (siehe S. 6-33).

align: Ausrichtung des Anfangs des zu reservierenden Speicherbereichs in Potenzen zur Basis 2: 0 = Byte, 1 = Wort, 2 = Doppelwort, 3 = 8 Byte, ...

size: Größe des zu reservierenden Speicherbereichs (in Byte). Der Rückgabewert von *size* enthält die *tatsächlich* reservierte Speichergröße.

adr: Zeiger auf eine Variable, in die die physikalische Anfangsadresse des reservierten Speichers eingetragen wird. Wenn *size* = 0 ist, dann ist die Adresse nicht gültig.

ml8_get_free_ram_size — Ermittle den freien Speicher auf der MODULAR-4

Pascal PROCEDURE ml8_get_free_ram_size (align: BYTE;
VAR size: LONGINT);

C void ml8_get_free_ram_size (byte align, ulong *size);

Funktion Diese Prozedur ermittelt die Größe des freien Speichers auf der MODULAR-4 Karte.

Parameter *align*: Gewünschte Ausrichtung des Speichers in Potenzen zur Basis 2: 0 = Byte, 1 = Wort, 2 = Doppelwort, 3 = 8 Byte,...

size: Zeiger auf eine Variable, in die die Größe des freien Speichers (in Byte) eingetragen wird.

ml8_set_ram_pointer — Setze Pointer für Speicherzugriff

Pascal PROCEDURE ml8_set_ram_pointer (adr: LONGINT);

C void ml8_set_ram_pointer (ulong adr);

Funktion Diese Prozedur setzt einen Zeiger, über den alle Speicherzugriffe erfolgen.

Parameter *adr*: Physikalische Adresse, auf die der Zeiger gesetzt werden soll.

ml8_read_ram — Lies Daten aus dem Speicher der MODULAR-4

Pascal PROCEDURE ml8_read_ram (size: WORD; VAR data_var);

C void ml8_read_ram (ushort size, void *data_var);

Funktion Die Prozedur liest einen Datenblock (max. 64K) ab der Speicherstelle, auf die der mit **ml8_set_ram_pointer** gesetzte Pointer zeigt. Der vom Betriebssystem der MODULAR-4 verwaltete Zeiger wird nach der Operation automatisch um die entsprechende Anzahl von Byte erhöht.

Parameter *size*: Anzahl von Datenbyte die gelesen werden sollen.

data_var: Zeiger auf den Variablenbereich, in den die gelesenen Daten eingetragen werden.

ml8_write_ram Schreibe Daten in Speicher der MODULAR-4

Pascal PROCEDURE ml8_write_ram (size: WORD; VAR data_var);

C void ml8_write_ram (ushort size, void *data_var);

Funktion Die Prozedur schreibt einen Datenblock (max. 64K) ab der Speicherstelle, auf die der mit **ml8_set_ram_pointer** gesetzte Pointer zeigt. Der vom Betriebssystem der MODULAR-4 verwaltete Zeiger wird nach der Operation automatisch um die entsprechende Anzahl von Byte erhöht.

Parameter *size*: Anzahl von Datenbyte die geschrieben werden sollen.

data_var: Zeiger auf den Variablenbereich, der die zu schreibenden Daten enthält.

ml8_read_memory Lies Daten direkt aus dem Speicher der MODULAR-4

Pascal PROCEDURE ml8_read_memory (start_addr: LONGINT; size: WORD; VAR data_var);

C void ml8_read_memory (ulong start_addr, ushort size, void *data_var);

Funktion Die Prozedur liest einen Datenblock (max. 64K) ab der angegebenen Speicherstelle.

Parameter *start_addr*: Anfangsadresse (phys.) des zu lesenden Datenblocks.

size: Anzahl von Datenbyte die gelesen werden sollen.

data_var: Zeiger auf den Variablenbereich, in den die gelesenen Daten eingetragen werden.

ml8_write_memory — Schreibe Daten direkt in den Speicher der MODULAR-4

Pascal: PROCEDURE ml8_write_memory (start_addr: LONGINT; size: WORD; VAR data_var);

C: void ml8_write_memory (ulong start_addr, ushort size, void *data_var);

Funktion: Die Prozedur schreibt einen Datenblock (max. 64K) ab der angegebenen Speicherstelle.

Parameter:
- *start_addr*: Anfangsadresse (phys.) des zu schreibenden Datenblocks
- *size*: Anzahl von Datenbyte die geschrieben werden sollen.
- *data_var*: Zeiger auf den Variablenbereich, der die zu schreibenden Daten enthält.

6.4.8. Zugriff auf das Flash der MODULAR-4

ml8_get_flash_info — Flash-ROM-Informationen

Pascal: FUNCTION ml8_get_flash_info (slot, chip, mode, option: BYTE): WORD;

C: ushort ml8_get_flash_info (byte slot, byte chip, byte mode, byte option);

Funktion: Diese Funktion liefert Status-Informationen über den Flash-ROM-Speicher auf einer MODULAR-4 Karte.

Parameter:
- *slot*: Steckplatz des Moduls, auf dessen Flash zugegriffen werden soll (s. Aufdruck)
- *chip*: Angabe über welches IC Informationen eingeholt werden sollen. Dieser Parameter muß für die "große" MODULAR-4/486 = 17 und für die "kleine" MODULAR-4/486 = 6 gesetzt werden.

Die Statusinformation wird als Rückgabewert der Funktion gemäß der-folgenden Tabelle geliefert:

mode	*option*	**Bedeutung des Rückgabewertes**
FLASH_GET_CUSTOMER_-ID	0	Hersteller-ID des Flash-EPROM
FLASH_GET_DEVICE_ID	0	Kennung des EPROM-Typs
FLASH_GET_SIZE_INFO	0	Information über die Größe des Flash-EPROMs mit folgender Bedeutung: Aus L = Low-Byte des Rückgabewertes H = High-Byte des Rückgabewertes Folgt EPROM-Größe = 2^L Byte Sektor-Größe = 2^H Byte
FLASH_GET_ ORGANIZATION	0	Organisation des Flash-EPROM: x0h = Bit x1h = Byte x2h = Word 0xh = Kein Sektor geschützt lxh = Bottom 2xh = Top
FLASH_CHECK_ WRITE_PROTECTED	Sektor	Gibt an, ob der angegebene Sektor schreibgeschützt ist oder nicht: 1 = Sektor ist schreibgeschützt 0 = Sektor ist nicht schreibgeschützt
FLASH_GET_-ERASE_STATUS	Sektor	Gibt den Status nach einem Löschvor-gang an: 0 = Löschvorgang fehlerfrei beendet 1 = Löschvorgang noch nicht beendet 2 = Fehler beim Löschen aufgetreten
FLASH_GET_SOFT_STATE	0	Soft-State (Betriebsart) 0 = Standard 1 = Für Löschvorgang vorbereitet 2 = Löschvorgang aktiv 3 = Programmiervorgang aktiv

ml8_set_flash_mode Flash-Status setzen

Pascal PROCEDURE ml8_set_flash_mode (slot, chip, mode, option: BYTE);

C void ml8_set_flash_mode (byte slot, byte chip, byte mode, byte option);

Funktion Diese Funktion setzt den Status des Flash-ROM-Speicher auf der MODULAR-4/486.

Parameter *slot*: Steckplatz des Moduls, auf dessen Flash zugegriffen werden soll (0 = Basiskarte)

chip: Angabe, in welches IC Informationen geschrieben werden sollen. Dieser Parameter muß für die "große" MODULAR-4/486 = 17 und für die "kleine" MODULAR-4/486 = 6 gesetzt werden.

Die auszuführende Aktion wird durch die Parameter *mode* und *option* gesteuert:

mode	*option*	Aktion
FLASH_ERASE_ENABLE	0	Aktivieren den Löschmodus
FLASH_ERASE_DISABLE	0	Beenden des Löschmodus
FLASH_PROGRAM_ENABLE	0	Aktivieren des Programmiermodus
FLASH_PROGRAM_DISABLE	0	Beenden des Programmiermodus
FLASH_COPY_EPROM	Segment	Kopieren des EPROM ins Flash-EPROM

ml8_erase_flash_sector Flash-Sektoren löschen

Pascal FUNCTION ml8_erase_flash_sector (slot, chip: BYTE; first, last: LONGINT): WORD;

C ushort ml8_erase_flash_sector (byte slot, byte chip, ulong first, ulong last);

Funktion Diese Funktion löscht einen Bereich im Flash-ROM-Speicher auf der MODULAR-4/486 Karte. Es ist zu beachten, daß das Flash in Sektoren organisiert ist. Sektoren können nur als Ganzes gelöscht werden. Liegt die Anfangsadresse nicht am Anfang eines Sektors, so wird auch der Bereich zwischen Sektoranfang und Startadresse gelöscht.

Parameter

slot Steckplatz des Moduls, auf dessen Flash zugegriffen werden soll (s. Aufdruck auf der Trägerkarte)

chip Angabe, in welchem IC gelöscht werden soll. Dieser Parameter muß für die "große" MODULAR-4/486 = 17 und für die "kleine" MODULAR-4/486 = 6 gesetzt werden.

first Angabe der Adresse, ab der gelöscht werden soll

last Angabe der Adresse, bis zu der gelöscht werden soll

ml8_program_flash
ml8_read_flash — Flash-Zugriff

Pascal

```
PROCEDURE ml8_program_flash (slot, chip: BYTE; start:
LONGINT; size: WORD; VAR data);

PROCEDURE ml8_read_flash (slot, chip: BYTE; start: LONGINT;
size: WORD; VAR data);
```

C

```
void ml8_program_flash (byte slot, byte chip, ulong start, ushort size,
void *data);

void ml8_read_flash (byte slot, byte chip, ulong start, ushort size, void
*data);
```

Funktion Mit diesen Funktionen können Daten in das Flash-ROM auf der MODULAR-4/486 Karte geschrieben bzw. ausgelesen werden. Bevor Daten in das Flash geschrieben werden können, müssen die entsprechenden Sektoren mit **ml8_erase_flash_sector** gelöscht werden.

Parameter	*slot*	Steckplatz des Moduls, auf dessen Flash zugegriffen werden soll (s. Aufdruck auf der Trägerkarte)
	chip	Angabe, in welchem IC geschrieben/gelesen werden soll. Dieser Parameter muß für die "große" MODULAR-4/486 = 17 und für die "kleine" MODULAR-4/486 = 6 gesetzt werden.
	start	Relative Adresse (bezogen auf den Anfang des Flash), ab der Daten zu lesen bzw. zu schreiben sind
	size	Anzahl Bytes, die gelesen bzw. geschrieben werden sollen
	data	Zeiger auf eine Variable, in die die gelesenen Daten eingetragen werden, bzw. in der die zu schreibenden Daten stehen.

6.4.9. Zugriffe auf die I/O-Ports der MODULAR-4

ml8_in8_port
ml8_in16_port Lies Daten von I/O-Port der MODULAR-4

Pascal FUNCTION ml8_in8_port (port: WORD) : BYTE;

FUNCTION ml8_in16_port (port: WORD) : WORD;

C byte ml8_in8_port (ushort port);

ushort ml8_in16_port (ushort port);

Funktion Diese Funktionen liefern den 8-Bit- bzw. 16-Bit-Wert zurück, der von der lokalen MODULAR-4 I/O-Adresse gelesen wurde.

Parameter *port*: MODULAR-4 I/O-Adresse.

ml8_out8_port
ml8_out16_port Schreibe Daten an I/O-Port der MODULAR-4

Pascal PROCEDURE ml8_out8_port (port: WORD; data: BYTE);

PROCEDURE ml8_out16_port (port: WORD; data: WORD);

C void ml8_out8_port (ushort port, byte data);

void ml8_out16_port (ushort port, ushort data);

Funktion Diese Funktionen schreiben einen 8-Bit- bzw. 16-Bit-Wert an eine lokale MODULAR-4 I/O-Adresse.

Parameter *port*: MODULAR-4 I/O-Adresse.

data: Daten, die geschrieben werden sollen.

6

6.4.10. Befehle zur Systemsteuerung

ml8_set_macro_interruptible Mache Makrobefehle un-/unterbrechbar

Pascal PROCEDURE ml8_set_macro_interruptible (on_off: BOOLEAN);

C void ml8_set_macro_interruptible (byte on_off);

Funktion Die Prozedur dient dazu, Makrobefehle auf der MODULAR-4 Karte unterbrechbar oder ununterbrechbar zu machen. Wenn sie unterbrechbar sind, bewirkt ein Makrobefehl nur eine kurze Interrupt-Sperrung auf der MODULAR-4 Karte zu Beginn des Makrobefehls. Andernfalls ist der gesamte Makrobefehl nicht von lokalen Hardware-Interrupts unterbrechbar. In diesem Fall kann z. B. beim Austausch großer Datenmengen mit dem PC - die Reaktionszeit der auf der Karte laufenden Programme deutlich verlängert werden. Bei schneller Meßdatenerfassung können Meßwerte verloren gehen.

Parameter *on_off*: = TRUE (1): unterbrechbar
= FALSE (0): nicht unterbrechbar (Voreinstellung).

ml8_cache_control MODULAR-4 Cache-Kontrolle

Pascal PROCEDURE ml8_cache_control (mode: BYTE);

C void ml8_cache_control (byte mode);

Funktion Die Prozedur dient zum Ein- und Ausschalten des Prozessor-Caches auf der MODULAR-4 Karte.

Parameter *mode*: 0 = Cache ausschalten
1 = Cache einschalten
3 = Cache ungültig machen und einschalten.

6.4.11. Zugriffe auf die EEPROMs der MODULAR-4

ml8_read_eeprom_direct
ml8_read_eeprom_copy Lies ein Wort aus dem EEPROM

Pascal FUNCTION ml8_read_eeprom_copy (modul, reladr: BYTE) : WORD;

FUNCTION ml8_read_eeprom_direct(modul, reladr: BYTE) : WORD;

C ushort ml8_read_eeprom_copy (byte modul, byte reladr);

ushort ml8_read_eeprom_direct (byte modul, byte reladr);

Funktion Diese Funktionen lesen ein Wort direkt aus einem EEPROM bzw. aus dessen im RAM angelegter Kopie. Die Kopie wird beim Aktivieren eines Betriebssystems (also nach Aufruf von **ml8_reset**) automatisch erzeugt. Direkte Zugriffe sind langsamer und besonders in einer Echtzeitumgebung nicht zu empfehlen.

Parameter *modul*: Welches EEPROM gelesen werden soll:
0: EEPROM der MODULAR-4 Basiskarte
1 bis 10: EEPROM des Moduls auf dem entsprechenden Steckplatz.

reladr: relative Adresse des zu lesenden Wortes (0-31).

ml8_write_eeprom_direct
ml8_write_eeprom_copy Schreibe ein Wort ins EEPROM

Pascal PROCEDURE ml8_write_eeprom_copy (modul, reladr: BYTE; data: WORD);

PROCEDURE ml8_write_eeprom_direct (modul, reladr: BYTE; data: WORD);

C void ml8_write_eeprom_copy (byte modul, byte reladr, ushort data);

void ml8_write_eeprom_direct (byte modul, byte reladr, ushort data);

Funktion Diese Funktionen schreiben ein Wort direkt in ein EEPROM bzw. in dessen im RAM angelegte Kopie. In die Kopie geschriebene Daten gehen beim Abschalten der Karte verloren.

Parameter *modul, reladr*: siehe **ml8_read_eeprom_direct / copy**

data: Daten, die geschrieben werden sollen.

6.4.12. Zugriffe auf die Echtzeituhr der MODULAR-4

ml8_get_rtc_status — Lies Status der Uhr

Pascal FUNCTION ml8_get_rtc_status : BYTE;

C byte ml8_get_rtc_status (void);

Funktion Das Funktionsergebnis gibt den Status der Uhr an. Das Rückgabebyte hat folgende Bedeutung:

Bit	Bedeutung
0	0 = Uhr wurde nach Power-On bzw. nach einem Hardware-Reset des Systems noch nicht gesetzt 1 = Uhr ist gesetzt Dieses Bit ist erst ab Betriebssystemversion 3B.07 definiert.
1	0 = Uhr läuft, 1 = Uhr gestoppt
2	ist immer = 1
3	0 = Interruptausgang nicht maskiert, 1 = Interruptausgang maskiert
4,5	Bit-5 Bit-4 Frequenz am Impulsausgang: 0 0 1/15,625 ms 0 1 1/ sec 1 0 1/min 1 1 1/ h
6	Zustand des Impulsausgangs (invertiert)
7	ist immer = 0

ml8_set_rtc_mode — Setze Betriebsart der Uhr

Pascal PROCEDURE ml8_set_rtc_mode (mode : BYTE);

C void ml8_set_rtc_mode (byte mode);

Funktion Die Prozedur setzt die Betriebsart der Uhr.

Parameter *mode*: Die Bits haben folgende Bedeutung

Bit	Bedeutung
0	Reset des Subsekundenzählers: Um den Subsekundenzähler zurückzusetzen, muß die Funktion zweimal aufgerufen werden; Beim ersten Aufruf muß dieses Bit = 1, beim zweiten Aufruf = 0 gesetzt sein. Soll kein Reset durchgeführt werden, muß dieses Bit = 0 gesetzt werden.
1	0 = Uhr starten, 1 = Uhr stoppen
2	immer = 1 setzen
3	0 = Interruptausgang nicht maskieren, 1 = Interruptausgang maskieren
4,5	Bit-5 Bit-4 Frequenz am Impulsausgang: 0 0 1/15,625 ms 0 1 1/sec 1 0 1/min 1 1 1/h
6	immer = 1 setzen
7	immer = 0 setzen

ml8_get_rtc_date_code — Lies Datecode der Uhr

Pascal FUNCTION ml8_get_rtc_date_code : LONGINT;

C ulong ml8_get_rtc_date_code (void);

Funktion Diese Funktion liest das Datum der Echtzeituhr. Das Datum wird als Datecode übergeben (siehe Kapitel 6.5).

ml8_get_rtc_time_code — Lies Timecode der Uhr

Pascal FUNCTION ml8_get_rtc_time_code : LONGINT;

C ulong ml8_get_rtc_time_code (void);

Funktion Diese Funktion liest die Uhrzeit der Echtzeituhr. Die Uhrzeit wird als Timecode übergeben (siehe Kapitel 6.5).

ml8_get_time — Lies Uhrzeit

Pascal PROCEDURE ml8_get_time (VAR hour, min, sec: BYTE);

C void ml8_get_time (byte *hour, byte *min, byte *sec);

Funktion Die Prozedur liefert die aktuelle Uhrzeit der MODULAR-4.

Parameter
- *hour*: Zeiger auf eine Variable, in die die aktuelle Stunde eingetragen wird.
- *min*: Zeiger auf eine Variable, in die die aktuelle Minute eingetragen wird.
- *sec*: Zeiger auf eine Variable, in die die aktuelle Sekunde eingetragen wird.

ml8_get_date_and_time — Lies Datum und Uhrzeit

Pascal PROCEDURE ml8_get_date_and_time (VAR year, month, day, dow, hour, min, sec : BYTE);

C void ml8_get_date_and_time (byte *year, byte *month, byte *day, byte *dow, byte *hour, byte *min, byte *sec);

Funktion Die Prozedur liefert die aktuelle Uhrzeit und das aktuelle Datum der MODULAR-4. Das Jahr wird dabei 2 stellig (0 bis 255) übergeben.

Parameter *year*: Zeiger auf eine Variable, in die das aktuelle Jahr eingetragen wird (0 bis 255).[1]

month: Zeiger auf eine Variable, in die der aktuelle Monat eingetragen wird.

day: Zeiger auf eine Variable, in die der aktuelle Tag eingetragen wird.

dow: Zeiger auf eine Variable, in die der aktuelle Wochentag eingetragen wird (0 = Sonntag, 1 = Montag, ... 6 = Samstag).

hour, min, sec: siehe **ml8_get_time.**

ml8_set_date_and_time Setze Datum und Uhrzeit

Pascal PROCEDURE ml8_set_date_and_time (year, month, day, dow, hour, min, sec: BYTE);

C void ml8_set_date_and_time (byte year, byte month, byte day, byte dow, byte hour, byte min, byte sec);

Funktion Die Prozedur setzt die Echtzeituhr der MODULAR-4.

Parameter siehe **ml8_get_date_and_time**

[1] In Betriebssystem Versionen 3B.07 und höher entspricht der Wert 0 der Jahreszahl 1900, der Wert 255 der Jahreszahl 2155. Beachten Sie, daß die Funktion die richtige Jahreszahl nur über den gesamten Wertebereich liefert, wenn die Uhr nach Hardware-Reset und ml8_set-data_and_time einmal gestellt worden ist. Ansonsten reicht der Wertebereich nur von 80 bis 179 (entsprechend 1980 bis 2079). (Näheres siehe AN063)

6.4.13. Steuerung der LEDs auf der MODULAR-4

ml8_external_led_on
ml8_external_led_off
ml8_local_led_on
ml8_local_led_off — Schalte eine LED ein/aus

Pascal
```
PROCEDURE ml8_local_led_on;
PROCEDURE ml8_external_led_on;
PROCEDURE ml8_local_led_off;
PROCEDURE ml8_external_led_off;
```

C
```
void ml8_local_led_off (void);
void ml8_external_led_off (void);
void ml8_local_led_on (void);
void ml8_external_led_on (void);
```

Funktion: Diese Prozeduren schalten entweder die LED auf der MODULAR-4 Karte (local_led = LEDint) bzw. die extern an Stecker St1 angeschlossene LED (external_led = LEDext) ein (on) bzw. aus (off).

ml8_get_local_led_status — Lies den Zustand der LED

Pascal
```
FUNCTION ml8_get_local_led_status: BYTE;
```

C
```
byte  ml8_get_local_led_status (void);
```

Funktion: Die Funktion gibt als Funktionsergebnis den Zustand der LED auf der MODULAR-4 Karte zurück. Wenn Bit-0 = 0 ist, dann ist die LED ausgeschaltet, wenn Bit-0 = 1 ist, dann ist sie eingeschaltet.

6.4.14. Sonstige Funktionen

ml8_get_physical_address Ermittle physikalische Adresse

Pascal FUNCTION ml8_get_physical_address (seg_offs: LONGINT): LONGINT;

C ulong ml8_get_physical_address (ulong seg_offs);

Funktion Diese Funktion berechnet aus einer Adresse vom Typ Segment:Offset eine physikalische Adresse.

Parameter *seg_offs*: Umzuwandelnde Zeigervariable (Segment:Offset).

6.5. Versionscode, Datecode und Timecode

Die Bibliothek liefert Versionsdaten als sogenannte Versions- und Datecodes. Die Funktionen für die Echtzeituhr stellen Datum und Uhrzeit ebenfalls als Date- und Timecode zur Verfügung. Die Codes bieten die Möglichkeit, auf einfache Weise Versionen, Daten und Uhrzeiten zu vergleichen. Ist z.B. der Versionscode von Version A grösser als der von Version B, so ist Version A die neuere von beiden.

Die Codes (32-Bit) haben folgenden standardisierten Aufbau:

31 … 24	23 … 16	15 … 8	7 … 0
AAAAAAAA	BBBBBBBB	CCCCCCCC	DDDDDDDD

Format des Versionscodes:

AAAAAAAA	Versionsnummer (z.B. 01h)
BBBBBBBB	Revisionsbuchstabe als ASCII Zeichen (z.B. 'A')
CCCCCCCC	Laufende Revisionsnummer (z.B. 03h)
DDDDDDDD	Status der Version als ASCII Zeichen:
	'F' : Free Release (freigegebene Version) 'B' : Beta Version 'R' : RAM-Version (OsX) 'E' : EPROM-Version (OsX)

Format des Datecodes:

AAAAAAAA	Jahreszahl als Offset zu 1900 (0 bis 159)
BBBBBBBB	Monat (1 bis 12)
CCCCCCCC	Tag (1 bis 31)
DDDDDDDD	Wochentag (0 = Sonntag, 1 = Montag, ..., 6 = Samstag)

Format des Timecodes:

AAAAAAAA	Stunden (0 bis 23)
BBBBBBBB	Minuten (0 bis 59)
CCCCCCCC	Sekunden (0 bis 59)
DDDDDDDD	Hunderstel Sekunden (0 bis 100)

Die oben angeführten Codes können auch in Klartext (Strings) umgewandelt werden. Die Strings werden dabei folgendermaßen formatiert:

Version 01.A.003 (F)

für Version 1, Rev. A, Revisionszähler 3, Free Release

Datum MON 16/02/1998

für Montag, den 16.02.1998

Zeit 11:26:41.18

für 11 Uhr, 26 Minuten, 41 Sekunden und 18 Hundertstel

Zur Umwandlung in den entsprechenden String stehen folgende Prozeduren zur Verfügung:

ml8_create_version_string — Versionscode in String umwandeln
ml8_create_date_string — Datecode in String umwandeln
ml8_create_time_string — Timecode in String umwandeln

Pascal

```
PROCEDURE ml8_create_version_string  (VAR v_string: STRING;
code: LONGINT);

PROCEDURE ml8_create_date_string  (VAR d_string: STRING;
code: LONGINT);

PROCEDURE ml8_create_time_string  (VAR t_string: STRING;
code: LONGINT);
```

C

```
void ml8_create_version_string (char* v_string, ulong code);

void ml8_create_date_string (char* d_string, ulong code);

void ml8_create_time_string (char* t_string, ulong code);
```

Funktion Die Prozeduren liefern die Version der verwendeten Bibliothek bzw. deren Erstellungsdatum als String zurück.

Parameter

v_string: Zeigt auf Variable zum Empfang des Version-Strings

d_string: Zeigt auf Variable zum Empfang des Datum-Strings

t_string: Zeigt auf Variable zum Empfang des Uhrzeit-Strings

code Zu konvertierender Code

6.6. Fehlerbehandlung

6.6.1. Das Konzept der Fehlerbehandlung

Innerhalb der Bibliotheksfunktionen wird eine Fehlerprüfung vorgenommen. Wenn ein Fehler auftritt, ruft die jeweilige Bibliotheksfunktion direkt eine vom Benutzer programmierbare Fehlerbehandlung auf.

Innerhalb dieser Prozedur kann dann mit Hilfe der Funktion **ml8_get_error_info** auf Informationen über den aufgetretenen Fehler zugegriffen werden. Nun kann der Fehler analysiert werden und es können geeignete Aktionen (Fehlermeldung auf dem Bildschirm, Wiederholung eines Makrobefehls, Abbruch des Programms, etc.) eingeleitet werden.

Bei der Entwicklung einer eigenen Fehlerbehandlungsprozedur ist zu beachten, daß, wenn mit PC-Interrupt gearbeitet wird, die Fehlerbehandlungsprozedur auch aus der Interrupt-Service-Prozedur heraus aufgerufen werden kann. Es sind dann innerhalb der Fehlerbehandlungsprozedur die gleichen Einschränkungen bezüglich des Aufrufs von MS-DOS Funktionen (DOS ist nicht reentrant) zu beachten wie in der Benutzer-Service-Prozedur (siehe Kapitel 6.7). Unter Windows 95 und Windows NT gibt es keine Einschränkungen.

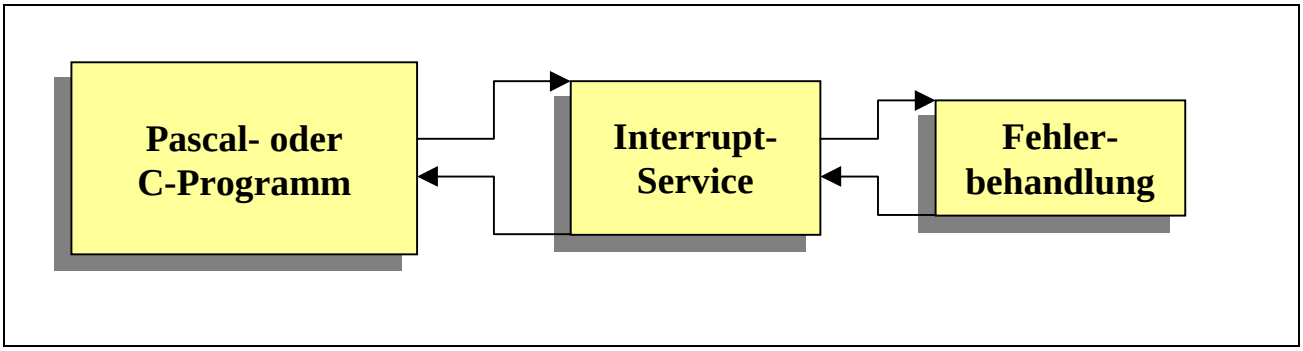


Abb. 6-1: Aufruf der Fehlerbehandlung in der Benutzer-Service-Routine

Die Fehlerbehandlung wird nicht aufgerufen, wenn mit **ml8_set_mark** der Wert Null eingestellt ist. Dies ist z. B. während der Ausführung von **ml8_reset** und **ml8_start** der Fall.

6.6.2. Prozeduren und Funktionen für die Fehlerbehandlung

ml8_set_error_handling — Übergib die Adresse der Fehlerbehandlungsprozedur

Pascal PROCEDURE ml8_set_error_handling (serv_proc: POINTER);

C void ml8_set_error_handling (ERROR_PROC_P serv_proc);

Funktion Die Funktionen **ml8_reset** und **ml8_start** erfordern bereits die Übergabe der Adresse einer Fehlerbehandlungsprozedur. Falls danach noch einmal die Fehlerbehandlungsprozedur gewechselt werden soll, muß diese Routine verwendet werden.

Parameter *serv_proc*: Adresse der Fehlerbehandlungsroutine.

ml8_get_error_info — Lies Fehlerinformation

Pascal PROCEDURE ml8_get_error_info
(VAR err_rec: ml8_error_info_type);

C void ml8_get_error_info (struct ml8_error_info_type *err_rec);

Funktion Diese Prozedur findet vor allem in der Fehlerbehandlungsprozedur Verwendung. Sie liefert eine Datenstruktur (Record) zurück, die Informationen über einen aufgetretenen Kommunikationsfehler zwischen MODULAR-4 Karte und PC enthält.

Das Datenfeld **ml8_error_info_type** hat folgenden Aufbau:

Feldbezeichner	Typ	Beschreibung
ok	byte	= FALSE: ein Fehler ist aufgetreten, sonst = TRUE. Solange diese Variable FALSE gesetzt ist, wird die MODULAR-4 Karte nicht mehr angesprochen (siehe **ml8_clear_error**).
repeatmacro	byte	= TRUE: der fehlgeschlagene Makrobefehl soll wiederholt werden. *ok* muß ebenfalls auf TRUE gesetzt werden. Bei jeder Makrobefehlsausführung wird *repeatmacro* wieder FALSE gesetzt!
errorcode	byte	Fehlercode (siehe Anhang E).
errorclass	byte	Fehlerklasse (siehe Anhang E).
statusport	byte	Wert des Status-Ports der MODULAR-4 - PC Schnittstelle (Basisadresse+0) unmittelbar nach Auftreten des Fehlers.
statusmap	byte	Zusätzliche Informationen über den Fehlerzustand. Die Bedeutung der einzelnen Bits steht in der nachfolgenden Tabelle.
errorrequest	ushort	reserviert. Die Fehlernachricht bei Auftreten eines Fehlers der Klasse 5 wird von der Funktion **ml8_get_message** geliefert.
sndcount	ushort	Anzahl der bereits gesendeten Byte eines Makrobefehls (nur Fehlerklasse 0 und 3).
rcvcount	ushort	Anzahl der bereits empfangenen Byte eines Makrobefehls (nur Fehlerklasse 0 und 3).
mark	ushort	Mit der Prozedur ml8_mark zuletzt gesetzter Markierungswert.

Feldbezeichner	Typ	Beschreibung
markmsgsize	char [1]	Anzahl gültiger Zeichen in *markmessage* (dieser Parameter existiert nicht in PASCAL!)
markmessage	char [80]	Textzeile für Tracing- und Debugging-Zwecke als Ergänzung der Variablen *mark*.
Outwords	ushort [6]	Auszugebende Byte (bis zu 12) des ausgeführten Makrobefehls (nur bei Fehlerklasse 0). Datenbyte der Makrobefehle **ml8_write_ram, ml8_write_data_block** und **ml8_write_par_block** sind nicht in diesem Datenfeld sondern in *datawords* gespeichert.
inwords	ushort [6]	Antwort (bis zu 12 Byte) der MODULAR-4 Karte (nur bei Fehlerklasse 0). Datenbyte der Makrobefehle **ml8_read_data_block, ml8_read_par_block** und **ml8_read_ram** sind nicht in diesem Datenfeld sondern in *datawords* gespeichert.
datawords	ushort [6]	Bis zu 12 Byte der Datenfelder der Makrobefehle **ml8_write_data_block, ml8_read_data_block, ml8_write_par_block, ml8_read_par_block, ml8_write_ram_block** und **ml8_read_ram_block** (nur bei Fehlerklasse 0).
errormsgsize	char [1]	Anzahl gültiger Zeichen in *errormessage* (dieser Parameter existiert nicht in PASCAL!)
errormessage	char [80]	Klartextmeldung zur Fehlerursache.
maxerrorretry	ushort	Festlegung, wie oft nach Auftreten eines Fehlers ein Makrobefehl wiederholt werden soll (siehe auch **ml8_set_max_retry** und **ml8_repeat_test**).
errorretry	ushort	Abwärtszähler für die Anzahl der durchgeführten Makrobefehlswiederholungen.

Die Bits im Parameter *statusmap* haben folgende Bedeutung:

statusmap Bit	**Bedeutung**
0	Ist gesetzt, wenn nach Aufruf der Fehlerbehandlungsprozedur ein weiterer Makrobefehl versucht, Daten zur MODULAR-4 Karte zu schreiben, ohne daß vorher *ok* = TRUE gesetzt wurde. Diese Aufgabe wird normalerweise durch Aufruf von **ml8_clear_error** in der Fehlerbehandlungsroutine erledigt.
1	Wie Bit 0, jedoch gesetzt, wenn Daten von der MODULAR-4 Karte gelesen werden sollen, ohne daß vorher *ok* = TRUE gesetzt wurde.
2-6	Reserviert
7	Ist gesetzt, wenn Routinen der Bibliothek ohne vorangegangenen Aufruf von **ml8_reset** oder **ml8_start** benutzt werden.

Unter Windows 95 und Windows NT lautet die Deklaration dieser Datenstruktur anders. Bitte sehen Sie in der Header-Datei (ML7BIB.H/ML8BIB.H) der verwendeten Bibliothek (je nach Betriebssystem) nach.

Wiederholung von Bibliotheksroutinen

Wenn eine Bibliotheksfunktion einen Fehler auslöst, wird die Fehlerbehandlungsroutine aufgerufen und anschließend das Programm fortgesetzt. Falls die Funktion einen Rückgabewert liefert, ist dieser nicht gesetzt, kann also falsche Daten liefern. Die Fehlerbehandlungsroutine muß also dafür sorgen, daß das Programm nicht sorglos weiterläuft, oder daß der letzte Befehl noch einmal ausgeführt wird und diesmal, wenn kein neuer Fehler auftritt, einen gültigen Rückgabewert liefert.

Ein Aufruf kann theoretisch beliebig oft wiederholt werden, die Anzahl wird jedoch durch *maxerrorretry* begrenzt. Das folgende Beispiel zeigt, wie die Wiederholung eines Befehls innerhalb der Fehlerbehandlungsroutine aussehen kann:

```
PROCEDURE errorhandler; FAR;
BEGIN
  {...}                              {Auswertung der Fehlerinformationen}
                                     {max. Anz. Wiederh. nicht erreicht?}
  IF ml8_error_repeat_test = TRUE THEN
    BEGIN
      IF ml8_clear=0 THEN            {Schnittstelle bereinigen und}
                                     {Fehlerrecord löschen}
        ml8_set_repeat_macro(TRUE); {Wiederholung veranlassen}
    END
  ELSE
    BEGIN                            {max. Anz. Wiederholungen überschritten}
      {... Programm anhalten oder Fehler melden}
    END;
END;
```

```
void far pascal errorhandler(void)
{
  /*...*/                           /*Auswertung der Fehlerinformationen*/
  if (ml8_error_repeat_test() == 1)/*max. Anz. Wiederh. nicht erreicht?*/
    {
     if (ml8_clear()==0)            /*Schnittstelle bereinigen und*/
                                    /*Fehlerrecord löschen*/
        ml8_set_repeat_macro(TRUE);/*Wiederholung veranlassen*/
    }
  else                              /*max. Anz. Wiederholungen überschritten*/
    {
      /* ... Programm anhalten oder Fehler melden */
    }
}
```

ml8_get_error_message — Lies die Fehler-Zeichenkette

Pascal PROCEDURE ml8_get_error_message (VAR errormessage: str80);

C void ml8_get_error_message (char *errormessage);

Funktion Mit dieser Prozedur erhält man eine kurze Klartextmeldung über die Fehlerursache.

Parameter *errormessage*: Zeiger auf eine Variable, in die die Nachricht eingetragen wird.

ml8_get_error_msg — Lies die Fehler-Zeichenkette

Pascal
PROCEDURE mlx_get_error_msg (VAR errstr: string;
errclass, errcode: byte);

C
void ml8_get_error_msg (char *errstr, unsigned char errclass, unsigned char errcode);

Funktion
Diese Routine liegt als Quellcode-Datei vor.

Parameter
errstr: Zeiger auf eine Variable, in die eine Fehlermeldung eingetragen wird.

errclass: Fehlerklasse

errcode: Fehlercode.

Hinweis
Die Funktion steht in verschiedenen Quellcode-Dateien zur Verfügung, die je nach Bedarf eingesetzt werden können.

ML8ERRD.C (Deutsch, C)

ML8ERRE.C (Englisch, C)

MLXERRD.PAS (Deutsch, Pascal)

MLXERRE.PAS (Englisch, Pascal)

ml8_set_error_message — Setze die Fehler-Zeichenkette

Pascal
PROCEDURE ml8_set_error_message (VAR errormsg: str80);

C
void ml8_set_error_message (char *errormsg);

Funktion
Diese Prozedur setzt den Inhalt einer Zeichenkette, die in der Fehlerbehandlungsprozedur mit der Funktion **ml8_get_error_message** angefordert werden kann.

Parameter
errormsg: Zeiger auf einen Variablenbereich, in dem die Fehlermeldung steht.

ml8_get_mark Lies die aktuelle Markierungszahl

Pascal FUNCTION ml8_get_mark: WORD;

C ushort ml8_get_mark (void);

Funktion Diese Funktion gibt den mit **ml8_set_mark** gesetzten Markierungswert zurück.

ml8_set_mark Setze Programmarke für Fehlerbehandlung

Pascal PROCEDURE ml8_set_mark (mark: WORD);

C void ml8_set_mark (ushort mark);

Funktion Mit dieser Prozedur wird eine Treibervariable gesetzt. Tritt danach ein Fehler auf, so kann in der Fehlerbehandlungsprozedur anhand der Markierung auf den Programmteil geschlossen werden, in dem der Fehler auftrat. Der mit **ml8_get_error_info** (siehe Seite 6-59) ermittelte Record enthält in der Variablen *mark* den zuletzt gesetzten Markierungswert. Er kann aber auch jederzeit mit **ml8_get_mark** ermittelt werden.

Parameter *mark*: Wert, auf den die Variable gesetzt werden soll.

ml8_get_mark_message Lies die Markierungs-Zeichenkette

Pascal PROCEDURE ml8_get_mark_message (VAR markmsg: str80);

C void ml8_get_mark_message (char *markmsg);

Funktion Mit dieser Prozedur erhält man die Zeichenkette, die mit **ml8_set_mark_message** für jede Karte gesetzt werden kann. Diese Prozedur ergänzt **ml8_get_mark**.

Parameter *markmsg*: Zeiger auf eine Variable, in die die Nachricht eingetragen wird.

ml8_set_mark_message — Setze die Markierungs-Zeichenkette

Pascal PROCEDURE ml8_set_mark_message (VAR markmsg: str80);

C void ml8_set_mark_message (char *markmsg);

Funktion Diese Prozedur setzt den Inhalt einer Zeichenkette, die in der Fehlerbehandlungsprozedur mit **ml8_get_mark_message** angefordert werden kann. Diese Prozedur ergänzt **ml8_set_mark**.

Parameter *markmsg*: Zeiger auf einen Variablenbereich, in dem die Zeichenkette steht.

ml8_set_max_error_retry — Setze zulässige Anzahl von Makrobefehlswiederholungen

Pascal PROCEDURE ml8_set_max_error_retry (value: WORD);

C void ml8_set_max_error_retry (ushort value);

Funktion Die Prozedur setzt die Anzahl von Makrobefehlswiederholungen (Variable *maxerrorretry* im Fehlerrecord). Gleichzeitig wird die Variable *errorretry* mit dem gleichen Wert initialisiert. *errorretry* wird von der Funktion **ml8_error_repeat_test** als Abwärtszähler verwendet. Die Werte von *maxerrorretry* und *errorretry* können mit Prozedur **ml8_get_error_info** ermittelt werden.

Parameter *value*: Wert, auf den *maxerrorretry* gesetzt werden soll.

ml8_error_repeat_test — Dekrementiere Fehlerzähler

Pascal FUNCTION ml8_error_repeat_test: BOOLEAN;

C byte ml8_error_repeat_test (void);

Funktion Diese Funktion dekrementiert die Variable *errorretry* um eins. Wenn die Variable gleich Null ist (max. Anzahl von Wiederholungen erreicht), wird FALSE zurückgegeben und die Variable wieder mit *maxerrorretry* initialisiert. Andernfalls wird TRUE zurückgegeben.

ml8_set_repeat_macro Erzwinge Makrobefehlswiederholung

Pascal PROCEDURE ml8_set_repeat_macro (rpm : BOOLEAN);

C void ml8_set_repeat_macro (byte rpm);

Funktion Die Prozedur darf nur innerhalb der Fehlerbehandlungsprozedur verwendet werden. Sie ist nur bei Fehlern der Klasse 0 sinnvoll einzusetzen. Durch diese Funktion wird erzwungen, daß der letzte Befehl nach Verlassen der Fehlerbehandlungsroutine noch einmal aufgerufen wird. Wenn dabei wieder ein Fehler auftritt, wird die Fehlerbehandlungsprozedur erneut aufgerufen.

Parameter *rpm*: Entscheidet, ob das Makro wiederholt wird (*rpm* = TRUE) oder nicht (*rpm* = FALSE).

ml8_clear_error Lösche die Fehlerinformation

Pascal PROCEDURE ml8_clear_error;

C void ml8_clear_error (void);

Funktion Die Prozedur löscht die Fehlerinformation der Bibliothek über einen vorangegangenen Fehler. Die Schnittstelle zur MODULAR-4 Karte wird nicht angetastet. Falls nach Abschluß der Fehlerbehandlungsprozedur das Programm fortgesetzt wird, *muß* die Fehlerinformation der Bibliothek gelöscht werden. Ansonsten kommt es beim nächsten Fehler zu einem Fehlerüberlauf. Bei Fehlerüberlauf ist in der Variablen *statusmap* Bit 0 oder Bit 1 gesetzt (zu ermitteln mit **ml8_get_error_info**).

ml8_clear Bereinige die Schnittstelle PC - MODULAR-4

Pascal FUNCTION ml8_clear: WORD;

C ushort ml8_clear (void);

Funktion Die Funktion entnimmt nicht abgeholte Zeichen aus der Schnittstelle und prüft, ob die Karte reagiert. Wenn **ml8_clear** den Wert 0 zurückliefert, so ist die Schnittstelle für weitere Kommunikation bereit. Die Fehlerbehandlungsprozedur wird nicht aufgerufen, sondern das Funktionsergebnis muß ausgewertet werden. Zu Beginn der Funktion **ml8_clear** wird die Fehlerinformation der Bibliothek über einen vorangegangenen Fehler gelöscht. **ml8_clear** enthält einen Aufruf von **ml8_clear_error.**

Die Tatsache, daß ein Kommunikationsfehler aufgetreten ist, bedeutet häufig, daß ein Makrobefehl nicht vollständig abgearbeitet worden ist. Es sollte daher in jedem Fall ergründet werden, warum der Kommunikationsfehler auftrat. **ml8_clear** dient dem Freimachen der Schnittstelle für Diagnosezwecke. Nach dem Aufruf von **ml8_clear** sollte das Programm nicht ohne Klärung der Fehlerursache fortgesetzt werden.

ml8_call_user_error Rufe Fehlerbehandlung auf

Pascal PROCEDURE ml8_call_user_error;

C void ml8_call_user_error (void);

Funktion Die Prozedur ruft die benutzereigene Fehlerbehandlung auf, die mit **ml8_reset**, **ml8_start** oder **ml8_set_error_handling** gesetzt worden ist. Die Funktion ist für die Erstellung von Bibliotheken gedacht, die bei einem Fehler die Fehlerbehandlungsroutine der **ML8BIB** aufrufen sollen.

6.7. Request-Behandlung

Die PC-Bibliothek für die MODULAR-4 Karte unterstützt die Behandlung von Service-Requests, die entweder vom Betriebssystem oder von Anwendungsprogrammen auf der Karte ausgelöst werden können. Die Service-Requests des Betriebssystems sind immer Fehlermeldungen und führen daher zu einem Aufruf der Fehlerbehandlungsprozedur. Die Behandlung der Service-Requests ist abhängig vom jeweiligen Betriebssystem:

6.7.1. MS-DOS und Windows 3.x

Wenn ein PC-Interrupt auf dem angewählten Kanal auftritt, wird das laufende Programm unterbrochen und zunächst eine bibliotheksinterne Interrupt-Service-Routine aufgerufen. Diese trifft einige Vorbereitungen und ruft dann die vom Benutzer in der Hochsprache (C, Pascal) geschriebene Benutzer-Service-Routine auf. Mit dem Ende dieser Benutzer-Service-Routine geht die Programmkontrolle wieder an die Bibliothek, die den Interrupt-Service abschließt. Alle interruptspezifischen Aktionen wie Register retten und Interrupt-Controller programmieren werden von der Bibliothek übernommen und dürfen in der Benutzer-Service-Routine nicht enthalten sein.

Folgende Besonderheiten und Restriktionen müssen beachtet werden:

Die Benutzer-Service-Routine muß **FAR** compiliert sein (DOS, Windows 3.x).

In der Interrupt-Service-Prozedur dürfen ohne besondere Vorkehrungen keine MS-DOS-Funktionen aufgerufen werden, da MS-DOS nicht reentrant ist (d. h. eine MS-DOS-Funktion darf während ihrer Ausführung nicht noch einmal aufgerufen werden). Daher sind keine Disketten- oder Festplattenoperationen zugelassen. Auch Bildschirmausgaben sind nur zugelassen, wenn die entsprechenden Routinen reentrant geschrieben sind und DOS und BIOS nicht verwenden (direktes Schreiben in Bildschirmspeicher ist möglich).

Zugriffe auf den PC-Hauptspeicher (Variablenzugriffe) und die Kommunikation mit der MODULAR-4 Karte mit den dafür vorgesehenen Bibliotheksfunktionen sind ohne Einschränkung möglich.

Es ist auch möglich, Service-Requests von einer MODULAR-4 Karte ohne die Verwendung eines PC-Interrupt-Kanals zu handhaben. Hierzu wird bei der Initialisierung mit **ml8_start** bzw. **ml8_reset** in Parameter *mode* die Betriebsart ohne Interrupt eingestellt (*mode* = 0). Wenn nun ein Request von der MODULAR-4 Karte generiert wird, dann wird dieser automatisch vor der Durchführung der nächsten Bibliotheksroutine erkannt und die Benutzer-Service-Routine aufgerufen. Um einen anstehenden

Interrupt zu erkennen und zu bearbeiten, ohne einen Makrobefehl zur Karte senden zu müssen, steht die Prozedur **ml8_poll_request** zur Verfügung. Es bleibt jedoch zu berücksichtigen, daß sich durch Verwendung dieses "Polled-Request" Modus die Reaktionszeit auf den Request auf der PC-Seite verlängert.

6.7.2. Windows 95 und Windows NT

Unter diesen Betriebssystemen werden Service-Requests in einem separaten Thread abgearbeitet. Dieser Thread wird bei der Initialisierung der Bibliothek (DLL) geöffnet. Er erhält eine hohe Priorität, so daß beim Auftreten eines Service-Requests dieser sofort behandelt wird. Zu Beginn der Service-Routine muß ml8_get_message aufgerufen werden.

ml8_set_service — Setze die Adresse der Serviceprozedur

Pascal: PROCEDURE ml8_set_service (servproc: POINTER);

C: void ml8_set_service (SERVICE_PROC_P servproc);

Funktion: Hiermit wird die Hochsprachen-Service-Routine bestimmt, die von der integrierten Interrupt-Service-Routine aufgerufen werden soll.

Parameter: *servproc*: Adresse der Hochsprachen-Service-Routine.

Beispiel:

```
ushort srq_word; /* globale Variable zum Aufnehmen des Requests */

/* Hochsprachen-Service-Routine */

void far pascal MyServiceFunction (void)
{
   srq_word = ml8_get_message();
   /*...*/             /* Anwenderdefinierte Request-Auswertung */
}

/* Hauptprogramm */

main()
{
   ...
   ml8_set_service (MyServiceFunction);
   ...
}
```

! Hinweis: In Borland Pascal darf die Prozedur nicht als vom Typ "Interrupt" deklariert werden. Stack Check {$S-} und Range Check {$R-} müssen abgeschaltet sein.

ml8_suspend_requests — Sperre die Schnittstelle

Pascal PROCEDURE ml8_suspend_requests;

C void ml8_suspend_requests (void);

Funktion Diese Prozedur unterbindet Service-Requests der MODULAR-4 Karte. Dies gilt sowohl für Requests mit PC-Interrupt-Auslösung als auch ohne.

Der Einsatz der Prozedur innerhalb der Benutzer-Service-Routine ist nicht sinnvoll, da der Treiber während des Interrupt-Service bereits selbst die Schnittstelle für Requests sperrt. Im Nicht-Interrupt-Teil des Hochsprachenprogrammes können dagegen Abfolgen von Prozeduren wie **ml8_move_r_pointer** und **ml8_read_data_block** gegen eine Unterbrechung geschützt werden.

Beispiel (Pascal):

```
ml8_suspend_requests;
ml8_reset_r_pointer (task);
ml8_move_r_pointer (task, 100);
ml8_read_data_block (task, size, destination);
ml8_allow_requests;
```

ml8_allow_requests — Gib die Schnittstelle frei

Pascal PROCEDURE ml8_allow_requests;

C void ml8_allow_requests (void);

Funktion Die Schnittstelle wird für Requests der MODULAR-4 Karte wieder freigegeben.

ml8_get_message — Lies aktuelle Interrupt-Meldung

Pascal FUNCTION ml8_get_message: WORD;

C ushort ml8_get_message (void);

Funktion Diese Funktion übergibt das von der MODULAR-4 Karte beim Interrupt zum PC übergebene Request-Wort, das im Treiber zwischengespeichert wurde. In der Regel wird diese Funktion zu Beginn der Benutzer-Service-Routine aufgerufen, um Aufschluß über die Ursache der Serviceanforderung zu erhalten.

ml8_message_available — Prüfe Interrupt-Status

Pascal FUNCTION ml8_message_available: BOOLEAN;

C byte ml8_message_available (void);

Funktion Mit dieser Funktion kann geprüft werden, ob tatsächlich ein Request-Wort vom Treiber zwischengespeichert wurde. Wird die Funktion **ml8_get_message** zu Beginn der Benutzer-Service-Routine aufgerufen, so ist es nicht nötig, mit **ml8_message_available** zu testen, ob ein Request-Wort vorliegt. Die Funktion ist jedoch für Hardware- und Software-Testzwecke verwendbar. Wenn ein Request-Wort vorhanden ist, liefert die Funktion den Wert TRUE zurück, ansonsten FALSE.

ml8_poll_request — Prüfe, ob ein Request vorliegt

Pascal FUNCTION ml8_poll_request: BOOLEAN;

C byte ml8_poll_request (void);

Funktion Diese Funktion prüft, ob ein Request der MODULAR-4 Karte ansteht. Sie wird nur im Betrieb ohne PC-Interrupt eingesetzt. Das Funktionsergebnis ist dann TRUE, wenn ein Benutzer-Service-Request ansteht, ansonsten ist das Funktionsergebnis FALSE.

7. Echtzeitprogrammierung

7.1. Einführung

Wie alle SORCUS Karten besitzt die MODULAR-4/486 ein eigenes, echtzeitfähiges Multi-Tasking-Betriebssystem: OsX. Die Programme dafür können in Borland-Pascal, Borland C++ oder in Assembler geschrieben werden. Vorausgesetzt, Sie sind mit Borland-Pascal oder Borland C++ vertraut, müssen Sie also weder eine neue Entwicklungsumgebung anschaffen noch sich neu einarbeiten. Außerdem können Sie den Turbo-Debugger verwenden, um Ihre Programme im Quelltext zu untersuchen.

In diesem Kapitel wird gezeigt, wie die Programme auszusehen haben, und worauf Sie bei der Programmierung achten müssen. Auf den mitgelieferten Disketten bzw. CD finden Sie die Beispielprogramme als Quelle und als compilierte Dateien.

Im Kapitel 8 finden Sie eine Beschreibung zum Remote-Debuggen auf der MODULAR-4/486 Karte.

Im Kapitel 9 finden Sie eine Referenz aller Subroutinen der Echtzeitbibliotheken ML7RTBIB bzw. ML8RTBIB für Pascal und C.

Kapitel 10 enthält Informationen und Aufrufkonventionen der Betriebssystem-Subroutinen auf Assembler-Ebene.

Die im folgenden dargestellten Programme sind auf einer MODULAR-4/486 lauffähig, sie sind also nicht auf einem PC verwendbar! **!**

7.2. Adressierung in Echtzeitprogrammen

In den folgenden Kapiteln werden Sie manchmal auf den Begriff der "physikalischen Adressen" und "Segment:Offset-Adressen" stoßen. Was damit gemeint ist, wird hier kurz erläutert:

Physikalische Adressen

Bei physikalischen Adressen besitzt jede Speicherstelle für ein Byte eine bestimmte Adresse. Der Speicher beginnt ab der Adresse 0, die Speicherstellen werden von dort aus fortlaufend durchnumeriert. Bei einem 1 MByte großen Speicher hat die letzte Speicherstelle also die Adresse fffffh (=1048575). Mit physikalischen Adressen können aber auch Speicherzellen, die sich oberhalb 1 MB befinden, adressiert werden.

Adressen vom Typ Segment:Offset

Diese Art der Adressierung wird von allen x86 CPUs im Real-Mode benutzt. Pointer in Pascal und in C sind so aufgebaut. Solche Adressen bestehen aus zwei Angaben, die "Segment" und "Offset" genannt werden. Das Segment einer Speicherstelle ist der ganzzahlige Anteil der physikalischen Adresse einer Speicherstelle geteilt durch 16. Der Offset ist der Rest dieser Division. Es gibt bei dieser Adressierung jedoch einen Haken: Eine physikalische Adresse kann durch verschiedene Adressen vom Typ Segment:Offset dargestellt werden. Weiterhin ist zu beachten, daß nur Speicherzellen, die sich unterhalb 1 MB befinden, adressiert werden können.

Beispiele:

Physikalische Adresse	Einige mögliche Segment:Offset-Adressen
12345h	1234:0005h 1230:0045h 1202:0325h

7.3. Elemente von Tasks

Jedes Programm, das unter dem Multi-Tasking-Betriebssystem OsX laufen soll, besteht aus den Elementen, die in der Tabelle aufgeführt sind. Zwingend müssen nur PDT und TDT vorhanden sein.

Tabellen	Programm-Deskriptor-Tabelle (PDT)
	Task-Deskriptor-Tabelle (TDT)
Daten	Parameter des Programms
	Datenbereich des Programms
Code	Prozedur 0 (Hauptprozedur)
	Prozedur 1 (Auto-Initialisierung)
	Prozedur 2
	...
	Prozedur n

Die PDT und die Prozeduren sind Bestandteil des Programms. Die TDT wird vom Betriebssystem bei der Installation angelegt. Parameter- und Datenbereich können sowohl Bestandteil des Programms sein als auch vom Betriebssystem angelegt werden.

In Kapitel 5 wurden die verschiedenen Tasktypen vorgestellt und erklärt, aus welchen Teilen sie bestehen und wann die einzelnen Prozeduren aufgerufen werden. In diesem Kapitel werden die einzelnen Teile nun näher vorgestellt und ihre Programmierung erklärt.

7.3.1. Programm-Deskriptor-Tabelle (PDT)

Die PDT ist so etwas wie ein Anmeldeformular, das das Betriebssystem braucht, um das Programm zu installieren. Sie enthält wichtige Informationen wie Programmnummer, Version und Revision des Programms, die Lage (Adressen) der Prozeduren innerhalb des Programms, usw. Jedes Programm muß dafür sorgen, daß die PDT eingerichtet, ausgefüllt und dem Betriebssystem bekanntgemacht wird (siehe Seite 7-13, 'PREPARE'). Eine ausführliche Beschreibung der PDT finden Sie im Anhang I.

Rel. Adr.	Typ	Erklärung
0	Byte	Typ der PDT (z.Zt. = 1)
1	Byte	Länge des Vorspanns der PDT (bei Typ 1 immer = 48)
2	Word	Anzahl der Prozeduren (z.B. = 3)
4	Word	Programmnummer (z.B. 1)
6	Char	Programmversion (z.B. '1')
7	Char	Programmrevision (z.B. 'A')
8	Byte	Prozessor-Typ (0 = 8086, 1 = V20/80186, 4 = 486)
9	Byte	Coprozessor-Typ (0 = kein Coprozessor, 4 = 486DX)
10	Byte	Programmiersprache des Programms
11	Byte	Programmtyp
12	Word	Flags (16 Bit)
14	Word	Interrupt-Nummer, für die das Programm gedacht ist
16	Long	Anfangsadresse des Datenbereichs
20	Long	Größe des Datenbereichs in Byte
24	Long	Minimal benötigte Datenbereichsgröße in Byte des Programms
28	Long	Maximal benötigte Datenbereichsgröße in Byte des Programms
32	Long	Anfangsadresse des Parameterbereichs
36	Word	Größe des Parameterbereichs in Byte
38	Long	Anfangsadresse des Hypertextbereichs
42	Word	Reserviert
44	Long	Reserviert
48	Long	Adresse der Haupt-Prozedur (Prozedur 0)
52	Long	Adresse der Auto-Init-Prozedur (Prozedur 1)
56	Long	Adresse der 1. Globalen Prozedur (Prozedur 2)
...	...	... (gegebenenfalls weitere Globale Prozeduren)

In den Bibliothek ML7RTBIB und ML8RTBIB wurden für die Verwendung in der PDT folgende Konstanten definiert:

Bezeichnung	Wert	Bedeutung
_PDT_LENGTH	48	Länge des PDT-Vorspanns
_486SX	3	Prozessor-Typ 80486 SX
_486DX	4	Prozessor-Typ 80486 DX
_NOCOPROZ	0	Kein Coprozessor benutzt
_487SX	3	Kein Coprozessor vorhanden (→ Emulation)
_487DX	4	80487 DX
_ASM	1	Programmiersprache Assembler
_TP70	7	Programmiersprache Turbo-Pascal 7.0 bzw. Borland-Pascal
_CPP31	6	Programmiersprache Borland C 3.1
_CPP40	8	Programmiersprache Borland C 4.0
_CPP45	9	Programmiersprache Borland C 4.5
_CPP50	10	Programmiersprache Borland C 5.0
_SYSTEM_PRG	0	Programmart: Systemprogramm
_USER_PRG	1	Programmart: Anwenderprogramm
_SOFT_INTERRUPT	2	Programmart: Software-Interrupt-Behandlung
_HARD_INTERRUPT	3	Programmart: Hardware-Interrupt-Behandlung
_TASK_MANAGER	4	Programmart: Taskmanager
_ERROR_TRAPS	5	Programmart: Fehler-Trap-Behandlung

7.3.1.1. Die Flags

Innerhalb der oben stehenden PDT finden Sie an der Position 12 den Eintrag "**Flags**". Dabei handelt es sich um ein Wort (2 Byte = 16 Bit), das Angaben über die Art der Installierung enthält. Es folgt eine Übersicht der einzelnen Bits und ihrer Bedeutung. Detaillierte Informationen dazu können Sie in der ausführlichen PDT-Beschreibung im Anhang I nachschlagen.

Bit-Nr.	Bedeutung	Bezeichnung in ML7RTBIB/ML8RTBIB
0..2	Tasktyp: 000: Nicht-Interrupt-Task 001: Indirekte Interrupt-Task 010: Direkte Interrupt-Task 011: Timer-Initiierte Task	 _NI_TASK _II_TASK _DI_TASK _TI_TASK
3	Task-Typ und Interrupt-Nummer werden durch PDT festgelegt (Bit=1)	_PDT_INFO_ENABLE
4	Real-Mode (Bit=0) oder Protected-Mode Programm (Bit=1)	_PROTECTED_MD
5	Code cacheable (Bit=0) oder nicht (Bit=1)	_NO_CACHE_USE
6	Reserviert	
7	Hypertext vorhanden (Bit=1)	_HYPER_TEXT
8	Der Datenbereich wird vom Betriebssystem (Bit=0) oder vom Programm selbst reserviert (Bit=1)	_LOCAL_DATA
9	Datenbereichsgröße variabel (Bit=0) oder fest (Bit=1)	_FIXED_DATASIZE
10	Der Parameterbereich wird vom Betriebssystem (Bit=0) oder vom Programm reserviert (Bit=1)	_LOCAL_PARAMETER
11..15	Reserviert	

7.3.2. Parameterbereich

Der Parameterbereich ist ein Variablenbereich, der einer Task zugeordnet, aber über verschiedene Betriebssystemfunktionen auch für andere Tasks zugänglich ist. Jede Task kann also auf die Parameter einer anderen Task zugreifen. Über entsprechende PC-Bibliotheksroutinen können auch vom PC aus Parameter gelesen oder verändert werden.

Die Parameter einer Task sind von Null beginnend byteweise durchnumeriert (relative Adresse). Der Zugriff auf die Parameter von einer anderen Task oder vom PC aus erfolgt immer über die Angabe dieser Nummer. Variablen des Parameterbereiches, die aus mehreren Bytes bestehen (z.B. Integer-Werte), belegen mehrere Nummern. Um auf solch einen Parameter zuzugreifen, wird immer die erste Nummer angegeben.

Der Parameterbereich kann entweder Teil des Programmes sein oder vom Betriebssystem bei der Installierung des Programms reserviert werden. Wenn das Programm den Parameterbereich selbst anlegt (Bit-10 in den PDT-Flags = 1, LOCAL_PARAMETER), dann geschieht das in der Regel mit einer Datenstruktur (**STRUCT** oder **RECORD**) im Variablenbereich. Die Adresse und die Größe dieser Struktur müssen in die PDT eingetragen werden (rel. Adresse 32 und 36). Innerhalb des Programms ist der Zugriff auf den Parameterbereich dann einfach ein Zugriff auf die deklarierte Datenstruktur.

Beispiel:

Pascal

```
Parameter: RECORD
               TDT:        TDT_Type;
               status:     BYTE;
               blink_rate: WORD;
               led_status: BYTE;
               led:        BYTE;
           END;
PDT.Flags := ... + _LOCAL_PARAMETER;
```

C

```
struct parameter_type
{ TDT_TYPE    tdt;
  byte        status;
  word        blink_rate;
  byte        led_status;
  byte        led;
} parameter;
PDT.Flags = ... + _LOCAL_PARAMETER;
```

Innerhalb der Task können Sie einfach mit **parameter.blinkrate** oder **parameter.led_status** auf einzelne Elemente zugreifen. Von anderen Tasks aus können Sie wie folgt auf diese Elemente zugreifen:

Pascal ("große" MODULAR-4/486)

```
blinkrate := ml8rt_read_par_word(task, 1);
led_status := ml8rt_read_par_byte(task, 3);
```

C ("kleine" MODULAR-4/486)

```
blinkrate = ml7rt_read_par_word(task, 1);
led_status = ml7rt_read_par_byte(task, 3);
```

Beachten Sie die Numerierung der Parameter. **blinkrate** hat die Nummer 1 und wird als Word zugegriffen, **led_status** hat die Nummer 3 und ist ein Byte.

Wenn der Parameterbereich beim Installieren des Programmes vom Betriebssystem angelegt wird (Bit-10 in den PDT-Flags = 0), muß nur die Größe des zu reservierenden Parameterbereichs in der PDT eingetragen sein. Der Eintrag 'Adresse des Parameterbereichs' in der PDT muß auf Null gesetzt werden. Um auf den Parameterbereich zugreifen zu können, muß das Programm die Adresse des Parameterbereichs ermitteln (ml8rt_get_par_address) und mit Pointern arbeiten oder mit Hilfe von Systemaufrufen Parameter lesen (ml8rt_read_par_xxxx) und schreiben (ml8rt_ write_par_xxxx).

Üblicherweise werden im Parameterbereich Daten zur Konfiguration des Programms abgelegt, z.B. Abtastrate, Anzahl der zu messenden Kanäle, usw.

7.3.3. Datenbereich

Im Datenbereich werden üblicherweise fortlaufende Daten (z.B. Meßdaten) gespeichert. Dieser Bereich ist genauso wie der Parameterbereich allen Tasks und dem PC zugänglich.

Anders als beim Parameterbereich wird von anderen Tasks oder vom PC aus über vom Betriebssystem verwaltete Zeiger (Pointer) auf die Daten zugegriffen. Für jede Task verwaltet das Betriebssystem einen Zeiger für Schreib- und einen für Lesezugriffe.

Der Datenbereich kann entweder im Programm selbst oder bei der Installierung vom Betriebssystem reserviert werden. Ist der Datenbereich im Programm enthalten, so ist seine Größe fest und kann nicht geändert werden (Bit-8 und Bit-9 in den PDT-Flags = 1, LOCAL_DATA, FIXED_DATASIZE). Üblicherweise ist der Datenbereich in diesem Fall als Array oder Struktur Teil der globalen Variablen des Programms und kann innerhalb des Programms mit normalen Variablenzugriffen benutzt werden. Seine Anfangsadresse und Größe müssen in die PDT eingetragen werden (rel. Adresse 16 und 20). Die Angaben für minimale und maximale Datenbereichsgröße (rel. Adresse 24 und 28) sind ohne Bedeutung und können auf Null gesetzt werden.

Wird der Datenbereich vom Betriebssystem reserviert, gibt es mehrere Möglichkeiten, seine Größe festzulegen:

- Das Programm legt die Größe des Datenbereichs auf den in der PDT als Größe eingetragenen Wert fest (Bit-9 in den PDT-Flags = 1, FIXED_DATASIZE)
- Die Größe wird erst beim Installieren angegeben (Bit-9 in den PDT-Flags=0). In diesem Fall wird im Installierungsbefehl (genauer gesagt im Installierungsflag, das von SNW bei M8INST oder von der PC-Bibliothek bei ml8_transfer_and_install übergeben wird) angegeben, ob einer der Einträge Größe, maximale Größe oder minimale Größe den Datenbereich bestimmen, oder ob die Datenbereichsgröße unabhängig von diesen drei Eintragungen im Installierungsbefehl übergeben wird. Dieses Verfahren wird zum Beispiel benutzt, um die Datenbereichsgröße dem aktuellen Meßproblem anzupassen.

Um auf den Datenbereich zuzugreifen, der vom Betriebssystem reserviert wurde, muß das Programm entweder die Adresse des Datenbereichs aus der TDT ermitteln (rel. Adresse 20, siehe Seite 7-11) und mit eigenen Zeigern arbeiten (nur möglich, wenn Adresse unterhalb von 1 Mbyte) oder Systemaufrufe und die Bibliotheksfunktionen benutzen (ml8rt_reset_x_pointer[1], ml8rt_move_x_pointer, ml8rt_read_data_xxxx, ml8rt_write_data_xxxx, etc.).

7.3.4. Prozeduren und Funktionen

Der eigentliche Programmcode besteht aus einer Anzahl unabhängiger (Task-) Prozeduren. Die Prozeduren sind von Null ausgehend durchnumeriert, wobei die Nummer durch die Position der Prozeduradresse in der PDT bestimmt wird. Neben den in der PDT eingetragenen, von außen aufrufbaren globalen Prozeduren, kann ein Programm beliebig viele lokale Prozeduren haben, die nur innerhalb des Programms Verwendung finden. Sie werden genauso wie bei PC-Programmen eingesetzt. Im weiteren Verlauf dieses Kapitel soll deshalb nur von globalen Prozeduren die Rede sein.

Die erste Prozedur (Nummer 0) ist die Hauptprozedur. Sie wird vom Betriebssystem immer dann aufgerufen, wenn das entsprechende Ereignis, unter dem die Task installiert wurde, aufgetreten ist (DI- oder II-Task) bzw. die TI- oder NI-Task an der Reihe ist.

[1] Für die "kleine" MODULAR-4/486 (ML7RTBIB) beginnen die Funktionsnamen mit ml7rt_...

Die zweite Prozedur (Nummer 1) ist die "Auto-Init-Prozedur". Sie wird vom Betriebssystem direkt nach der Installierung einmal aufgerufen. In dieser Prozedur können z.B. Parameter vorinitialisiert oder bestimmte Interrupts gesperrt werden. Der automatische Aufruf der Auto-Init-Prozedur kann durch das Setzen eines Installierungsflags unterbunden werden.

Alle folgenden Prozeduren sind optional und haben keine vordefinierte Funktion. Das können z.B. Start- oder Stop-Prozeduren sein. Alle Prozeduren sind auf der Karte global verfügbar. Das bedeutet, daß eine Task jede beliebige Prozedur einer beliebigen anderen Task aufrufen kann. Selbstverständlich lassen sich Prozeduren auch vom PC durch entsprechende PC-Bibliotheksfunktionen aufrufen. Prozeduren werden in der Regel mit Bibliotheksroutinen einfach durch Angabe von Task- und Prozedurnummer aufgerufen.

Eine besondere Gruppe von Prozeduren sind die **Funktionen**. Ihnen können (im Gegensatz zur Prozedur) beim Aufruf Daten übergeben werden, und sie können Daten an die aufrufende Task zurückgeben.

Beachten Sie bitte, daß die Prozeduren 0 und 1 keine Funktionen sein können!

Beim Aufruf einer Funktion müssen folgende Parameter übergeben werden:

1) Nummer der Task, deren Funktion aufgerufen werden soll.
2) Nummer der aufzurufenden Funktion.
3) Anzahl Bytes, die der Funktion übergeben werden sollen.
4) Ein Pointer, der auf die Datenbytes zeigt, die an die Funktion übergeben werden sollen (Quellspeicher).
5) Anzahl Bytes, die maximal von der Funktion zurückerwartet werden.
6) Ein Pointer, der auf den Speicherbereich zeigt, in den die Funktion ihre Antwort eintragen kann (Zielspeicher).

Die aufgerufene Funktion kann nach Aufruf mit diesen Parametern arbeiten und anschließend gegebenenfalls eine Antwort in den Zielspeicherbereich eintragen.

Um Prozeduren oder Funktionen aufzurufen, stehen entsprechende Routinen auf dem PC (ML7BIB/ML8BIB) und auf der Karte (ML7RTBIB/ML8RTBIB) zur Verfügung.

7.3.5. Task-Deskriptor-Tabelle (TDT)

Bevor ein Programm gestartet werden kann, muß es auf die MODULAR-4/486 geladen und dort installiert werden. Zur Erinnerung: Installieren bedeutet, daß ein Programm auf die Karte geladen und dem Betriebssystem bekanntgemacht wird.

Im Detail heißt das, daß das Betriebssystem die relevanten Informationen aus der PDT holt und in einer speziellen Tabelle, nämlich der Task-Deskriptor-Tabelle (TDT), ablegt. Die TDT wird für jede Task angelegt und steht direkt vor dem Parameterbereich der Task. Wenn der Parameterbereich im Programm enthalten ist, muß auch der Platz für die TDT im Programm reserviert werden. Dazu steht in den Echtzeitbibliotheken die Struktur **TDT_TYPE** zur Verfügung, mit deren Hilfe auch einfach auf die verschiedenen TDT-Einträge zugegriffen werden kann (siehe Beispielprogramme). Die TDT enthält unter anderem auch die Informationen, wo der Datenbereich beginnt. Das ist dann wichtig, wenn der Datenbereich vom Betriebssystem reserviert worden ist, und Sie die Anfangsadresse dieses Datenbereichs benötigen. Es werden außerdem weitere Informationen wie Interruptnummer, Tasknummer sowie die Schreib- und Lese-Pointer auf den Datenbereich in dieser Tabelle gehalten. Eine ausführliche Beschreibung der TDT finden Sie in Anhang J.

In den Bibliotheken ML7RTBIB und ML8RTBIB ist die TDT als RECORD (bzw. STRUCT in C) mit dem Namen TDT_Type definiert. Die Namen der einzelnen Strukturelemente entnehmen Sie der Spalte 'Feldbezeichner'.

Rel. Adr.	Feldbe-zeichner	Typ	Erklärung
0	typ	Byte	TDT-Typ (z. Zt. immer = 1)
1	length	Byte	Länge der TDT in Byte (bei Typ 1 = 36)
2	task	Word	Tasknummer
4	flags	Word	Flags
6	inter	Word	Interrupt-Nummer (bei II- und DI-Tasks) bzw. Zahl der Aktivierungen (bei NI-Tasks)
8	par	Word	Größe des Parameterbereichs (in Anzahl Bytes)
10	proc	Word	Anzahl der Prozeuren
12	pdt	Long	Adresse der PDT (physikalisch)
16	hyper	Long	Adresse der Hypertextinformationen (physikalisch)
20	datafirst	Long	Anfangsadresse des Datenbereichs (physikalisch)
24	datalast	Long	Endadresse+1 des Datenbereichs (physikalisch)
28	readptr	Long	Read-Pointer auf den Datenbereich (physikalisch). Dieser Pointer wird auch vom PC benutzt.
32	writeptr	Long	Write-Pointer auf den Datenbereich (physikalisch). Dieser Pointer wird auch vom PC benutzt.

7.4. Unterschiede zur PC-Programmierung unter DOS

Die wesentlichen Unterschiede zur PC-Programmierung ergeben sich aus der Struktur des Betriebssystems, die sich deutlich von der von DOS unterscheidet.

Auch unter DOS sind die meisten größeren Programme in Prozeduren (und Funktionen) aufgeteilt. Die Hauptprozedur (main in C oder das "Hauptprogramm" in Pascal) werden von Betriebssystem angesprungen, wenn das Programm gestartet wird. Von hier aus wird gesteuert, welche Prozeduren in welcher Reihenfolge aufgerufen werden.

In dem Echtzeit-Betriebssystem OsX der MODULAR-4 Karte wird der Aufruf der einzelnen Prozeduren nicht vom Programm selbst, sondern vom Betriebssystem gesteuert. Die Prozeduren werden beim Installieren aufgerufen (Auto-Init), wenn bestimmte Ereignisse eingetreten sind (Haupt-Prozedur) oder wenn sie von einer anderen Task oder vom PC aus gestartet worden sind (globale Prozeduren). Die Prozedur, die bei einem DOS-Programm die Hauptprozedur wäre (main) wird vom Betriebssystem der MODULAR-4 Karte nur zum Zeitpunkt der Installierung einmal aufgerufen. Sie hat die Aufgabe, die PDT zu initialisieren und dem Betriebssystem bekannt zu machen (ml7rt_set_pdt_adr bzw. ml8rt_set_pdt_adr). Dieser Teil wird als PREPARE bezeichnet.

Im Unterschied zur DOS Programmierung müssen Sie sich beim Erstellen von Echtzeitprogrammen selbst um die "Anmeldung" des Programmes kümmern, da die Compiler keine Informationen über das Betriebssystem der MODULAR-4 Karte haben, um die notwendigen Tabellen selbst anzulegen. Das erledigen Sie, wie schon erwähnt, im PREPARE-Teil des Programmes. Achten Sie unbedingt darauf, daß alle Einträge in der PDT zu Ihrem Programm passen.

7.5. Allgemeines zu den Beispielprogrammen

Im folgenden soll an einfachen Beispielprogrammen in Borland-Pascal und C++ gezeigt werden, wie Programme in Hochsprachen erstellt werden. Alle Programme finden Sie auch auf den mitgelieferten Disketten bzw. CD, ebenso wie weitere Beispielprogramme für die Echtzeitprogrammierung. Um erste Eindrücke zu sammeln, ist es sicherlich hilfreich, wenn Sie zuerst ein wenig mit diesen Programmen experimentieren.

Die hier beschriebenen Programme lassen lediglich eine Leuchtdiode - entweder die auf der Karte eingebaute oder eine extern angeschlossene - blinken, einmal als NI-Task und einmal als DI-Task. Bei diesen Programmen ist der Aufbau des Parameterbereichs gleich. Es wird kein Datenbereich benutzt.

Parameter der Blink-LED-Programme:

Nr.	Typ	Erklärung
0	Byte	Status des Programms: 0 = Programm bereit 2 = Programm läuft 4 = Programmlauf abgebrochen
1	Word	Blinkrate der LED Bei der NI-Task wird hier die Anzahl der NI-Task-Aufrufe eingetragen, nach denen die LED umgeschaltet wird. Bei der DI-Task wird hier die Blinkfrequenz in Hertz angegeben.
3	Byte	Status der LED: 0 = LED ist aus 1 = LED ist an
4	Byte	Benutzte LED: 1 = on-board LED 2 = Externe LED

Alle Programme für die "große" MODULAR-4/486 Karte benutzen die Echtzeitbibliothek "ML8RTBIB". Programme für die "kleine" MODULAR-4/486 verwenden die "ML7RTBIB". Nähere Informationen zu dieser Bibliothek finden Sie in Kapitel 9 und in den Header-Dateien "ML7RTBIB.H" und "ML8RTBIB.H" bzw. in den Dateien "ML7RTBIB.PAS und "ML8RTBIB.PAS".

Als Entwicklungsumgebung können Sie Borland-Pascal für DOS oder Borland C++ für DOS verwenden. Ob diese integrierten Entwicklungsumgebungen unter DOS, Windows 3.x , Windows 95 oder Windows NT laufen, ist hierfür ohne Bedeutung.

7.6. Programmierung in Borland-Pascal

7.6.1. Allgemeines

Bevor Sie mit der eigentlichen Programmierung beginnen können, sind einige Vorbereitungen notwendig. Zuerst sollten Sie sich ein Entwicklungsverzeichnis anlegen, in dem Sie Ihre Programme für die MODULAR-4/486 ablegen können, z.B. "C:\SORCUS\ML8\RT\Pascal". Danach muß die System-Unit von Borland-Pascal durch eine neue ersetzt werden.

7.6.2. Einbinden der neuen System-Unit

Die System-Unit enthält neben den Laufzeitbibliotheken auch den Initialisierungsteil von Borland-Pascal. Darin sind viele DOS-Aufrufe enthalten, die in einer speziellen System-Unit durch Aufrufe des MODULAR-4/486 Betriebssystems (OsX) ersetzt sind.

Die System-Unit wird automatisch in alle Programme eingebunden, ohne daß sie mit USES angegeben werden muß. Normalerweise liegt sie auch nicht explizit als SYSTEM.TPU vor, sondern ist mit den anderen Standard-Units von Borland-Pascal (z.B. PRINTER) in der Datei TURBO.TPL (TPL = Turbo-Pascal-Library) zusammengefaßt, wodurch sich die Ladezeiten bei der Compilierung deutlich verkürzen. Der Inhalt von TURBO.TPL wird mit dem Programm TPUMOVER verändert. Die genaue Beschreibung dieses Programms finden Sie im Borland-Pascal Benutzerhandbuch im Kapitel "Die Zusatzprogramme".

In Zukunft werden Sie mit zwei verschiedenen SYSTEM.TPUs arbeiten. Die Original Borland-Unit für PC-Programme und die SORCUS-Unit für Echtzeit-Programme. Um den Wechsel zwischen den Units möglich zu machen, muß zuerst die SYSTEM.TPU aus TURBO.TPL entfernt werden. Dazu wechseln Sie bitte in Ihr Borland-Pascal Verzeichnis und geben folgende Zeilen ein:

```
tpumover turbo * system
tpumover turbo - system
```

Mit der Ausführung der ersten Zeile wird die Datei SYSTEM.TPU erzeugt. Die zweite Zeile entfernt die System-Unit aus TURBO.TPL.

Nach wie vor müssen Sie die System-Unit nicht in der USES-Anweisung angeben, allerdings müssen Sie dem Compiler nun mitteilen, wo er die Datei SYSTEM.TPU findet. Das geschieht, wie für andere Units auch, unter 'Option/Verzeichnisse/Unit-Verzeichnisse'. Um Verwechslungen auszuschließen, geben Sie den Pfad, unter dem die SYSTEM.TPU zu finden ist, als ersten an. Für Echtzeitprogramme ist das normalerweise C:\SORCUS\ML7\RT\Pascal\TPU70 ("kleine" MODULAR-4/486) bzw. C:\SORCUS\ML8\RT\Pascal\TPU70 ("große" MODULAR-4/486), für PC-Programme der Pfad, in dem die SYSTEM.TPU extrahiert wurde.

Der Eintrag in die Verzeichnisliste unter 'Option' kann entfallen, wenn die SYSTEM.TPU in das Verzeichnis kopiert wird, in dem Sie Ihre Echtzeitprogramme entwickeln. Das aktuelle Verzeichnis wird immer vor den in der Liste angegebenen Verzeichnissen nach Units durchsucht.

Falls Sie die SYSTEM.TPU verwechseln, erhalten Sie beim Versuch, ein Echtzeitprogramm zu erstellen, die Meldung 'External Bezeichner **_wrong_startups_linked** nicht gefunden'. Im umgekehrten Fall, also beim Compilieren eines PC-Programms mit der SORCUS-System-Unit, ist die Fehlermeldung leider nicht so festgelegt. In der Regel wird die Meldung 'UNIT-Versionen stimmen nicht überein (*Name*)' lauten, wobei *Name* für eine beliebige, von Ihnen verwendete Unit stehen kann.

7.6.3. Programmierung

Nach der Einbindung der neuen SYSTEM.TPU können Sie Echtzeitprogramme mit Borland-Pascal für die MODULAR-4/486 erstellen. Beachten Sie dabei die folgenden Einschränkungen und Hinweise: **!**

- *Es sind keine Bildschirm- (Grafik und Text) oder Tastaturfunktionen möglich.*
- *Es kann und darf keine Overlay-Technik verwendet werden.*
- *Es können keine Datei-Operationen durchgeführt werden.*
- *Es kann kein Speicher dynamisch reserviert bzw. freigegeben werden ("GETMEM", "NEW", "DISPOSE" etc.). Dafür stehen spezielle Bibliotheksroutinen zur Verfügung.* ***Pointer-Operationen können normal verwendet werden.***
- *Es darf kein Range-Check aktiviert sein. Eine Zusammenstellung der Compilerschalter folgt auf Seite 7-18.*
- *Es dürfen keine DOS-spezifischen Funktionen benutzt werden (DOS-Interrupts)*
- *Das Einfügen von Debug-Informationen hat keinen Einfluß auf die Ausführung des Programms.*
- *Es dürfen keine Floating-Point-Operationen in DI- und II-Tasks, oder in Prozeduren bzw. Funktionen, die vom PC aus aufgerufen werden, vorgenommen werden.*
- *Die Befehle "Port" und "PortW", die in den auf Diskette bzw. CD mitgelieferten Programmen verwendet werden, beschreiben oder lesen ein Byte bzw. ein Wort einer I/O-Adresse. Nähere Informationen zu diesen Befehlen finden Sie im Programmierhandbuch von Borland-Pascal im Kapitel "Interne Details, Direkter Zugriff auf I/O-Adressen".*

Folgende Standardfunktionen von Borland-Pascal können ohne Einschränkungen verwendet werden:

Abs	Delete	Int	Pred	SSeg
Addr	DSeg	Length	Ptr	Str
ArcTan	Exp	Ln	Round	Succ
Chr	Exit	Lo	Seg	Swap
Concat	FillChar	Move	Sin	Trunc
Copy	Frac	Odd	SizeOf	Upcase
Cos	Hi	Ofs	SPtr	Val
CSeg	Inc	Ord	Sqr	
Dec	Insert	Pos	Sqrt	

Weiterhin kann selbstverständlich wie bisher das Unit-Konzept benutzt werden, um beispielsweise eigene Bibliotheken zu generieren.

7.6.4. Compiler- und Speichereinstellungen

{$F+}	Prozeduren mit FAR-CALLS aufgerufen
{$R-}	Range-Check ausschalten
{$S-}	Stack-Check ausschalten
{$I-}	I/O-Check ausschalten
{$M 1024,0,0}	Stack und Heap auf ein Minimum einstellen.

Wenn Fließkommazahlen (nicht vom Typ REAL) benutzt werden:

{$N+}	Befehle für den mathematischen Co-Prozessor erzeugen
{$E+}	Co-Prozessor-Emulation einbinden. Die Emulation muß auch bei MODULAR-4/486DX Karten eingeschaltet sein, der Co-Prozessor wird dann trotzdem benutzt.

Wenn keine Fließkommazahlen oder nur solche vom Typ REAL benutzt werden:

{$N-}	Keine Befehle für den mathematischen Co-Prozessor erzeugen
{$E-}	Keine Co-Prozessor-Emulation einbinden.

Das Einstellen der Compilerschalter können Sie entweder in der Entwicklungsumgebung unter 'Options' oder direkt im Quelltext mit Hilfe von Compiler-Switches vornehmen. Empfohlen wird die Einstellung im Quelltext.

7.6.5. Beispielprogramme für Borland-Pascal

7.6.5.1. NI-Task in Borland-Pascal

Das folgende Beispielprogramm (für die "große" MODULAR-4/486 Karte) zeigt das Blink-LED Programm als NI-Task. Sie finden das komplette Programm im Sourcecode unter den Namen "M8P0300.PAS" auf den mitgelieferten Disketten bzw. CD. Soll das Programm für die "kleine" MODULAR-4/486 Karte verwendet werden, müssen sämtliche "ml8..." Aufrufe durch "ml7..." ersetzt werden.

```
{$N-}                                   { Coprozessor aus }
{$E-}                                   { keine Emulation }
{$R-}                                   { kein Range-Check }
{$S-}                                   { kein Stack-Checking }
{$I-}                                   { kein IO-Checking }
{$V-}                                   { Keine strikte Stringübergabe }
{$L+}                                   { Local-Symbols für Turbo-Debugger }
{$F+}                                   { FAR Aufrufe erzwingen! }
{$M 1024, 0, 0}                         { Stack und Heap auf Minimum }

uses ml8rtbib;                          { Die on-board Bibliothek fuer die "große" MODULAR-4/486}
                                        { Karte (ML8) einbinden }

{ ***** ALLGEMEINE ANGABEN ***** }
Const
   Programm_Number      : Word = $300;    { Programmnummer = 300h }
   Version              : Char = '1';     { Version des Programms }
   Revision             : Char = 'D';     { Revision des Programms }

{ ***** Die verschiedenen Statusmöglichkeiten des Programms ***** }
   READY                = 0;              { Bereit }
   RUNNING              = 2;              { Programm läuft }
   ERROR                = 3;              { Falsche LED angewählt }
   STOPPED              = 4;              { Programmlauf angehalten }

{ ***** Zustände der LED ***** }
   EIN                  = 1;
   AUS                  = 0;
{ ***** Benutzte LED ***** }
   INTERN               = 1;              { Interne LED }
   EXTERN               = 2;              { Externe LED }



{ ***** Aufbau der Programm-Descriptor-Tabelle (PDT) ***** }
Type
   PDT_Type = Record
                PDT_HEAD : ML8_PDT_HEAD;  { PDT-Header }
                Main          : Pointer;  { Adresse der Haupt-Prozedur (Task-Prozedur) }
                Auto_Init     : Pointer;  { Adresse der Auto-Init-Prozedur }
                Start         : Pointer;  { Adresse der Start-Prozedur }
                Stop          : Pointer;  { Adresse der Stop-Prozedur }
       end;
```

```
{ ***** DEKLARATION DER GLOBALEN PROGRAMMDATEN ***** }
VAR
   PDT       : PDT_Type;                  { Deklaration der Kopftabelle }
{ ***** Parameterbereich des Programms deklarieren. ***** }
{ Auf diese Daten kann später über das Betriebssystem zugegriffen werden. }
   Parameter : Record
                 TDT       : TDT_Type;    { Platz für die TDT reservieren }
                 Status    : Byte;        { Status des Programms }
                 Blink_Rate: Word;        { Blinkrate }
                 LED_Status: Byte;        { Zustand der LED }
                 LED       : Byte;        { Welche LED 1=Intern, 2=Extern }
               end;
   Pause     : Word;                      { Ein Zähler, der die Durchläufe des Programms zählt }


{ ***** AUTO-INITIALISIERUNG ***** }
Procedure Auto_Init;
begin
   ml8rt_entry;                           { Register retten und vorbereiten }
   Parameter.Status := READY;             { Status des Programms auf bereit }
   Parameter.Blink_Rate := 3000;          { Blinkrate defaultmäßig einstellen }
   Parameter.LED_Status := AUS;           { Status der LED auf "AUS" }
   Parameter.LED := INTERN;               { Standardmäßig die interne LED benutzen }
   ml8rt_External_LED_Off;                { Externe LED ausschalten }
   ml8rt_Local_LED_Off;                   { On-Board LED ausschalten }
   Pause := 0;                            { Durchlaufzähler auf Null }
   ml8rt_exit;                            { Register wieder restaurieren }
end;

{ ***** Start-Prozedur ***** }
Procedure Start;
begin
   ml8rt_entry;
   if (Parameter.LED<>INTERN) AND (Parameter.LED<>EXTERN) then
   begin                                  { Falls eine falsche LED angewählt wurde, ... }
      Parameter.Status := ERROR;          { ... Fehlermeldung ausgeben }
   end
   else
   begin                                  { Falls alles OK, dann Task aktivieren }
      Parameter.Status := RUNNING;        { (TDT enthält die Tasknummer, unter der das Programm
                                             installiert wurde)}
      ml8rt_wakeup_task (Parameter.TDT.Task);
   end;
   ml8rt_exit;
end;

{ ***** Stop-Prozedur ***** }
Procedure Stop;
begin
   ml8rt_entry;
   ml8rt_sleep_task (Parameter.TDT.Task); { Task deaktivieren }
   Parameter.Status := STOPPED;           { Parameter auf "Programm angehalten". }
   ml8rt_exit;
end;
```

```
{ ***** HAUPTPROZEDUR (NI-Task) ***** }
Procedure Main_Proc;
begin
   ml8rt_entry;
   Pause := Pause + 1;                    { Durchlaufzähler um 1 erhöhen }
   if Pause>=Parameter.Blink_Rate then    { Wenn Pause (Anzahl Aufrufe der Main-Proc) angeg. Wert
                                            erreicht hat}
   begin                                  { ... die LED umschalten }
      if Parameter.LED_Status = EIN then
      begin                               { Falls die LED eingeschaltet ist, ... }
         if Parameter.LED = EXTERN then
            ml8rt_External_LED_Off        { ... LED ausschalten und ... }
         else
            ml8rt_Local_LED_Off;
         Parameter.LED_Status := AUS;     { ... Zustand merken }
      end
      else                                { Falls die LED ausgeschaltet ist, ... }
      begin
         if Parameter.LED = EXTERN then
            ml8rt_External_LED_On         { ... LED einschalten ... }
         else
            ml8rt_Local_LED_On;
         Parameter.LED_Status := EIN;     { ... Zustand merken }
      end;
      Pause := 0;                         { Durchlaufzähler wieder auf Null setzen }
   end;

   ml8rt_Exit;
end;

{ ***** PREPARE ***** }
begin                                     { Die Prozedur stellt alle Werte in der Kopftabelle ein }
   PDT.PDT_HEAD.ucType := 1;              { PDT-Typ (im Moment wird nur der Typ 1 unterstützt) }
   PDT.PDT_HEAD.ucSize := sizeof(ML8_PDT_HEAD); { Größe des Vorspanns der Kopftabelle (im
                                                  Moment immer = _PDT_Length = 48) }
   PDT.PDT_HEAD.usGlobalProc := 4;               { Anzahl der Prozeduren }
   PDT.PDT_HEAD.usProgNo := Programm_Number;     { Programm-Nummer }
   PDT.PDT_HEAD.cVersion := Version;             { Programm-Version }
   PDT.PDT_HEAD.cRevision := Revision;           { Programm-Revision }
   PDT.PDT_HEAD.ucCPU := _486SX;                 { CPU-Typ }
   PDT.PDT_HEAD.ucCoProc := _NOCOPROZ;           { Co-Prozessor-Typ }
   PDT.PDT_HEAD.ucLanguage := _TP70;             { Programmiersprache }
   PDT.PDT_HEAD.ucProgType := _User_Prg;         { Programm-Typ: Anwenderprogramm }
   PDT.PDT_HEAD.usFlags := $0;                   { Flags }
   PDT.PDT_HEAD.usFlags := _NI_Task + _PDT_Info_Enable + _Local_Data + _Local_Parameter +
                                                  _Fixed_Datasize;
   PDT.PDT_HEAD.usInt := 0;                      { Interrupt-Nummer für die das Programm gedacht ist
                                                   (ungültig, da NI-Task) }
   PDT.PDT_HEAD.ulDataAdr := 0;                  { Adresse des Datenbereichs }
   PDT.PDT_HEAD.ulDataSize := 0;                 { Größe des Datenbereichs }
   PDT.PDT_HEAD.ulDataSizeMin := 0;              { Minimaler Datenbereich }
   PDT.PDT_HEAD.ulDataSizeMax := 0;              { Maximaler Datenbereich }
                                                  { Anfangsadresse des Parameterbereichs }
   ml8rt_phys_adr (@Parameter, PDT.PDT_HEAD.ulGlobalParAdr);
   PDT.PDT_HEAD.ulGlobalParAdr:= PDT.PDT_HEAD.ulGlobalParAdr + SizeOf(TDT_Type);
                                                  { Anzahl der Parameter }
   PDT.PDT_HEAD.usGlobalParSize := SizeOf(Parameter) - SizeOf(TDT_Type);
   PDT.PDT_HEAD.ulHyperAdr := 0;                 { Adresse des Hypertextes (wird nicht benutzt) }
   PDT.PDT_HEAD.usRes1 := 0;
   PDT.PDT_HEAD.ulRes2 := 0;

   PDT.Main := @Main_Proc;                   { Adresse der Hauptprozedur }
   PDT.Auto_Init := @Auto_Init;              { Adresse der Auto-Init-Prozedur }
   PDT.Start := @Start;                      { Adresse der Start-Prozedur }
   PDT.Stop := @Stop;                        { Adresse der Stop-Prozedur }
   ml8rt_Set_PDT_Adr (PDT);                  { Adresse der PDT im Rückgabe-Record dem Betriebssystem
                                               übergeben }
end.
```

Da das Programm insgesamt umfangreich kommentiert ist, soll jetzt nur noch auf Besonderheiten eingegangen werden.

Als erstes wird die Unit "ML8RTBIB" eingebunden. Diese Unit enthält verschiedene Routinen, die Ihnen die Programmierung erleichtern sollen. Eine ausführliche Be-

schreibung der einzelnen Routinen finden Sie im Kapitel 9 dieses Handbuchs. Anschließend wird die PDT deklariert. Sie sehen am Ende der PDT, daß das Programm vier globale Prozeduren enthält.

Außerdem wird noch ein Record vereinbart, der die Parameter enthält. Die Parameter werden also lokal im Programm gehalten, der Parameterbereich wird nicht vom Betriebssystem reserviert. Beachten Sie bitte, daß vor den eigentlichen Parametern Platz für die TDT reserviert wird.

Alle globalen (also in der PDT aufgeführten Prozeduren) beginnen mit **ml8rt_entry** und enden mit **ml8rt_exit**. Dadurch ist sichergestellt, daß alle Prozessor-Register gesichert und für die Prozedur richtig eingestellt werden.

In der "Auto-Init-Prozedur" werden die Parameter initialisiert und die LED ausgeschaltet. In der Prozedur "Init_PDT" wird die PDT initialisiert. Beachten Sie bitte die Einstellung der PDT-Flags.

Der Teil, der zwischen "**begin**" und "**end.**" steht, enthält in einem "normalen" Pascal Programm das eigentliche Hauptprogramm. Dieser Teil wird während der Installierung abgearbeitet. Er sorgt dafür, daß die PDT initialisiert wird (Init_PDT) und dem MODULAR-4/486 Betriebssystem die Adresse der PDT mitgeteilt wird (ml8rt_set_pdt_adr).

7.6.5.2. Die Installierung der NI-Task

Nachdem das Programm "M8P0300.PAS" einwandfrei übersetzt und die Datei "M8P0300.EXE" erzeugt wurde, kann das Programm auf die Karte geladen und gestartet werden. Geben Sie dazu bitte folgende Installationsdatei mit Hilfe von SNW ein, und laden Sie diese Datei. Sie können auch die Beispielinstallationsdatei "NILED.INS" benutzen.

```
; Beispielinstallation: LED-Blinken als NI-Task (ohne Interrupt)
;
;
; "große" Karte anwählen, Adresse einstellen und Reset auslösen
M8DEVICE 0380 TIMEOUT=10 RESET
;
;
; Programmnummer 300h unter Tasknummer 21h installieren (nicht aktivieren)
M8INST M8P0300.EXE 0300 0021 01 000000 00000980
;
; Parameter der Task 21h setzen
; PAR-1,2 : Blinkrate der LED    => 1000h
; PAR-3   : Status der LED       => 0 = Aus
; PAR-4   : Benutzte LED         => 1 = On-board LED
;
M8PAR 21 01 00 10 00 01
;
;
; Durch Aufruf der Start-Prozedur Programm starten (Proz. 2)
M8PROC 21 02
```

Diese Installationsdatei installiert die NI-Task, stellt die Blinkrate ein und startet das Programm durch Aufruf der Prozedur 2.

7.6.5.3. DI-Task in Borland-Pascal

Das folgende Listing zeigt das Blink-LED Programm als DI-Task für die "kleine" MODULAR-4/486. Dieses Programm finden Sie auch unter dem Namen "M7P0301.PAS" auf der mitgelieferten Diskette bzw. CD. Das Programm wird unter dem Interrupt des TIMER-A installiert. Die Blinkrate kann in Hertz angegeben werden. Soll das Programm für die "große" MODULAR-4/486 Karte verwendet werden, müssen sämtliche "ml7..." Aufrufe durch "ml8..." ersetzt werden.

```
{$N-}                                   { Coprozessor aus }
{$E-}                                   { keine Emulation }
{$R-}                                   { kein Range-Check }
{$S-}                                   { kein Stack-Checking }
{$I-}                                   { kein IO-Checking }
{$V-}                                   { Keine strikte Stringübergabe }
{$L+}                                   { Local-Symbols für Turbo-Debugger }
{$F+}                                   { FAR Aufrufe erzwingen }
{$M 1024, 0, 0}                         { Stack und Heap auf Minimum }

uses ml7rtbib;                          { Echtzeit Bibliothek fuer die "kleine" MODULAR-4/486}
                                        { (ML7) einbinden}

{ ***** ALLGEMEINE ANGABEN ***** }
Const
   Programm_Number      : Word = $301;    { Programmnummer = 301h }
   Version              : Char = '1';     { Version des Porgramms }
   Revision             : Char = 'D';     { Revision des Programms }
   { ***** Die verschiedenen Statusmöglichkeiten des Programms ***** }
   READY                = 0;              { Bereit }
   RUNNING              = 2;              { Programm läuft }
   LED_ERROR            = 3;              { Falsche LED angewählt }
   STOPPED              = 4;              { Programm abgebrochen }
   { ***** Zustände der LED ***** }
   EIN                  = 1;
   AUS                  = 0;
   { ***** Benutzte LED ***** }
   INTERN               = 1;              { Interne LED }
   EXTERN               = 2;              { Externe LED }

   { ***** Timer Initialisierungen ***** }
   { Timer A 16-bit-> (65535 / 2), da Periodendauer T = t(LED_ON) + t(LED_OFF)}
   TIMER_VAL            : longint = 30000;
   { Timer A Basistakt 2.5 MHz }
   TIMER_FREQ           : longint = 2500000;

{ ***** Aufbau der Programm-Descriptor-Tabelle (PDT) ***** }
Type
   PDT_Type = Record
               PDT_HEAD : ML7_PDT_HEAD;  { PDT-Header }
               Main          : Pointer;   { Adresse der Haupt-Prozedur }
               Auto_Init     : Pointer;   { Adresse der Auto-Init-Prozedur }
               Start         : Pointer;   { Adresse der Start-Prozedur }
               Stop          : Pointer;   { Adresse der Stop-Prozedur }
            end;

{ ***** DEKLARATION DER GLOBALEN PROGRAMMDATEN ***** }
VAR
   PDT                  : PDT_Type;       { Deklaration der Kopftabelle }

{ ***** Parameterbereich des Programms deklarieren. ***** }
{ Auf diese Daten kann später über das Betriebssystem zugegriffen werden. }
   Parameter : Record
                TDT       : TDT_Type;     { Platz für TDT reservieren }
                Status    : Byte;         { Status des Programms }
                Blink_Rate: Word;         { Blinkrate in Hz }
                LED_Status: Byte;         { Zustand der LED }
                LED       : Byte;         { LED: 1=Intern, 2=Extern }
                Soft_Cnt  : Byte;         { Soft-Counter noetig, um mit Timer}
                                          { auch niedrige Freq. zu erzeugen! }
                Soft_Init : Byte;         { Zaehlerinitialisierungswert }
               end;
```

```
{ ***** AUTO-INITIALISIERUNG }
Procedure Auto_Init;
begin
   ml7rt_entry;                          { Register retten und vorbereiten }
   Parameter.Status := READY;            { Status des Programms auf bereit }
   Parameter.Blink_Rate := 1;            { Blinkrate Defaultwert 1 Hz }
   Parameter.LED_Status := AUS;          { Status der LED auf "AUS" }
   Parameter.LED := INTERN;              { Standardmaessig interne LED }
   ml7rt_External_LED_Off;               { Externe LED ausschalten }
   ml7rt_Local_LED_Off;                  { Interne LED ausschalten }
   ml7rt_exit;                           { Register wieder restaurieren }
end;

{ ***** Programm starten ***** }
Procedure Start;
VAR
   timer : Word;
begin
   ml7rt_entry;
   if (Parameter.LED=EXTERN) OR (Parameter.LED=INTERN) then
   begin
      {* Softzaehler ausrechnen (muss ganzzahlig sein, keine Fliesskommaop.!): *}
      parameter.soft_init:= TIMER_FREQ div (2* parameter.blink_rate * TIMER_VAL);

      {* falls Softzaehler auf 0 "abgerundet" wird, setze Zaehler auf 1 ! *}
      if parameter.soft_init = 0 then
         parameter.soft_init := 1;
      parameter.soft_cnt := parameter.soft_init;

      {* Timerwert ausrechnen: Fehler durch Softteiler (ganzzahlig!) wird durch
         Anpassung des maximalen Timerwerts kompensiert! *}
      timer := TIMER_FREQ div (2 * parameter.blink_rate * parameter.soft_init);

      ml7rt_set_timer (TIMER_A, timer, INT_MODE); { Timer-A starten }
      Parameter.Status := RUNNING;
      ml7rt_unmask_int (IRQ_TIMER_A);             { Timer-A demaskieren }
   end
   else
      Parameter.Status := LED_ERROR;
   ml7rt_exit;
end;


{ ***** Programmlauf abbrechen ***** }
Procedure Stop;
begin
   ml7rt_entry;
   ml7rt_mask_int (IRQ_TIMER_A);         { Timer-A Interrupt maskieren }
   Parameter.Status := STOPPED;          { Parameter setzen }
   ml7rt_exit;
end;
```

```
{ ***** HAUPTPROZEDUR (DI-Task) ***** }
Procedure Main_Proc;
begin
   ml7rt_entry;
   Parameter.Soft_Cnt := Parameter.Soft_Cnt -1;
   if Parameter.Soft_Cnt=0 then
   begin
     if Parameter.LED_Status = EIN then
     begin                                { Falls die LED eingeschaltet ist, }
       if Parameter.LED = EXTERN then
          ml7rt_External_LED_Off          { ... LED ausschalten und ... }
       else
       ml7rt_Local_LED_Off;
       Parameter.LED_Status := AUS;       { ... Zustand merken }
     end
     else                                 { Falls die LED ausgeschaltet ist, }
     begin
       if Parameter.LED = EXTERN then
          ml7rt_External_LED_On           { ... LED einschalten ... }
       else
       ml7rt_Local_LED_On;
       Parameter.LED_Status := EIN;       { ... Zustand merken }
     end;
     Parameter.Soft_Cnt := Parameter.Soft_Init;
   end;

   ml7rt_end_of_int (IRQ_TIMER_A);        { EOI an Interrupt-Contr. senden }
   ml7rt_exit_interrupt;
end;


{ ***** PREPARE ***** }
begin
   PDT.PDT_HEAD.ucType := 1;              { PDT-Typ (im Moment wird nur der Typ 1 unterstützt) }
   PDT.PDT_HEAD.ucSize := sizeof(ML7_PDT_HEAD); { Größe des Vorspanns der Kopftabelle (im
                                            Moment immer = _PDT_Length = 48) }
   PDT.PDT_HEAD.usGlobalProc := 4;              { Anzahl der Prozeduren }
   PDT.PDT_HEAD.usProgNo := Programm_Number;    { Programm-Nummer }
   PDT.PDT_HEAD.cVersion := Version;            { Programm-Version }
   PDT.PDT_HEAD.cRevision := Revision;          { Programm-Revision }
   PDT.PDT_HEAD.ucCPU := _486SX;                { CPU-Typ }
   PDT.PDT_HEAD.ucCoProc := _NOCOPROZ;          { Co-Prozessor-Typ }
   PDT.PDT_HEAD.ucLanguage := _TP70;            { Programmiersprache }
   PDT.PDT_HEAD.ucProgType := _User_Prg;        { Programm-Typ: Anwenderprogramm }
   PDT.PDT_HEAD.usFlags := $0;                  { Flags }
   PDT.PDT_HEAD.usFlags := _DI_Task + _PDT_Info_Enable + _Local_Data + _Local_Parameter +
                                                _Fixed_Datasize;
   PDT.PDT_HEAD.usInt := IRQ_TIMER_A;           { Interrupt-Nummer für die das Programm gedacht ist}
   PDT.PDT_HEAD.ulDataAdr := 0;                 { Adresse des Datenbereichs }
   PDT.PDT_HEAD.ulDataSize := 0;                { Größe des Datenbereichs }
   PDT.PDT_HEAD.ulDataSizeMin := 0;             { Minimaler Datenbereich }
   PDT.PDT_HEAD.ulDataSizeMax := 0;             { Maximaler Datenbereich }
                                                { Anfangsadresse des Parameterbereichs }
   ml7rt_phys_adr (@Parameter, PDT.PDT_HEAD.ulGlobalParAdr);
   PDT.PDT_HEAD.ulGlobalParAdr:= PDT.PDT_HEAD.ulGlobalParAdr + SizeOf(TDT_Type);
                                                { Anzahl der Parameter }
   PDT.PDT_HEAD.usGlobalParSize := SizeOf(Parameter) - SizeOf(TDT_Type);
   PDT.PDT_HEAD.ulHyperAdr := 0;                { Adresse des Hypertextes (wird nicht benutzt) }
   PDT.PDT_HEAD.usRes1 := 0;
   PDT.PDT_HEAD.ulRes2 := 0;
   PDT.Main := @Main_Proc;                { Adresse der Hauptprozedur }
   PDT.Auto_Init := @Auto_Init;           { Adresse der Auto-Init-Prozedur }
   PDT.Start := @Start;                   { Adresse der Start-Prozedur }
   PDT.Stop := @Stop;                     { Adresse der Stop-Prozedur }
   ml7rt_Set_PDT_Adr (PDT);               { Adresse der PDT im Rückgabe-Record dem Betriebssystem
                                            übergeben }
end.
```

Da auch hier das Listing ausführlich kommentiert ist, soll nur auf Besonderheiten eingegangen werden.

Die Start-Prozedur berechnet zuerst aus der angegebenen Frequenz die Timer-Werte. Beachten Sie bitte dabei, daß die Timer in der Regel mit einer Eingangsfrequenz von

2,5 MHz getaktet werden, und daß jeder Timer 16 Bit breit ist. Da mit diesen Werten "nur" Blinkfrequenzen zwischen 2,5 MHz und 38 Hz möglich sind, wurde der Hardware-Timer durch einen Softwarezähler erweitert. Erst wenn dieser Softwarezähler auf Null gelaufen ist, wird die LED umgeschaltet. Bei der Programmierung wurde darauf geachtet, daß der Timer-Wert möglichst groß ist, damit die Anzahl der Interrupts vom Timer pro Zeiteinheit möglichst klein ist.

Die Hauptprozedur unterscheidet sich ein wenig von der Hauptprozedur der NI-Task. Es handelt sich praktisch um eine Interrupt-Service Routine. Am Ende der Prozedur wird ein EOI zu den Interrupt-Controllern gesendet und anschließend die Prozedur mit "ml7rt_exit_interrupt" beendet.

*Die Hauptroutine darf nicht als '**interrupt**'-Prozedur deklariert werden. Das durch die **interrupt**-Anweisung bewirkte Retten der Register wird mit **ml7rt_entry** erledigt, das Beenden der Prozedur mit IRET mit **ml7rt_exit_interrupt**.* **!**

7.6.5.4. Installieren der DI-Task

Nachdem das Programm "M7P0301.PAS" einwandfrei übersetzt und die Datei "M7P0301.EXE" erzeugt wurde, kann das Programm auf die Karte geladen und gestartet werden. Geben Sie dazu bitte folgende Installationsdatei mit Hilfe von SNW ein und laden Sie diese Datei. Sie können auch die Beispielinstallationsdatei "DILED.INS" benutzen.

```
; Beispielinstallation: LED-Blinken mit Timer-Interrupt (TIMER-A)
;
; "kleine" MODULAR-4/486 Karte anwählen und Reset auslösen
M7DEVICE 0380 TIMEOUT=10 RESET
;
;
; Programm-Nummer 301h unter der Tasknummer 22h installieren
; (Interrupt 91h => TIMER-A)
M7INST M7P0301.EXE 0301 0022 91 000000 0982
;
;
; Parameter der Task 22h setzen
; PAR-0   : Status des Programms
; PAR-1,2 : Blinkrate der LED    => 5 Hz
; PAR-3   : Status der LED       => 0 = Aus
; PAR-4   : Benutzte LED         => 1 = On-Board LED
;
M7PAR 22 01 05 00 00 01
;
;
; Durch Aufruf der Start-Prozedur (Proz. 2) das Programm starten
M7PROC 22 02
```

Die Installationsdatei installiert die DI-Task und aktiviert sie anschließend.

Beachten Sie bitte, daß dieses Programm im Gegensatz zur NI-Task nur einmal (!) installiert werden kann, da es auch nur einen TIMER-A auf der Basiskarte gibt. Sie könnten als Übung ja versuchen, das vorhandene Programm unter einen anderen Namen zu kopieren und es so zu modifizieren, daß das neue Programm z.B. den TIMER-B benutzt.

Wenn Sie ausschließlich mit Pascal programmieren, lesen Sie bitte ab Seite 7-42 weiter.

7.7. Programmierung in Borland C

7.7.1. Allgemeines

Bevor Sie mit der eigentlichen Programmierung beginnen können, sind einige Vorbereitungen notwendig. Zuerst sollten Sie sich ein Entwicklungsverzeichnis anlegen, in dem Sie Ihre Programme für die MODULAR-4/486 ablegen können. Zum Beispiel C:\SORCUS\ML7\RT\BC. Als nächstes müssen die neuen Start-Up-Codes für Ihre C Version eingebunden werden.

7.7.2. Einbindung des neuen Start-Up-Codes

C enthält für jedes Speichermodell eine Datei mit Standardbibliotheksroutinen. Das Einbinden der Routinen erfolgt selbständig durch den Linker. Diese Dateien enthalten neben den Standardbibliotheken auch den sogenannten Start-Up-Code, der vor dem Aufruf der eigentlichen Main-Prozedur durchlaufen wird. Da dieser Start-Up-Code eine Vielzahl von DOS-Aufrufen beinhaltet, muß er durch den von SORCUS gelieferten Start-Up-Code für das MODULAR-4/486 Betriebssystem ersetzt werden.

In Zukunft werden Sie mit zwei verschiedenen Start-Up-Codes arbeiten: beim Erstellen von PC-Programmen mit dem Original-Borland-Start-Up-Code, beim Programmieren von Echtzeitprogrammen mit dem SORCUS Start-Up-Code. Welcher Code jeweils verwendet wird, wird unter den Menüpunkten "Options/Directories/Library Directory" und "Options/Directories/Include Directory" festgelegt. Wenn Sie also Echtzeitprogramme entwickeln, müssen Sie **vor (!)** dem Pfadnamen zu den Standardbibliotheken den Pfadnamen des Verzeichnisses angeben, in das Sie die SORCUS Start-Up-Codes (Dateinamen = C*.OBJ) kopiert haben.

Beispiel: Sie arbeiten in der folgenden Arbeitsumgebung:

Pfad zu den Bibliotheksroutinen von C++:
C:\BORLANDC\LIB

Pfad zu Ihren MODULAR-4/486 ("große" Karte) Start-Up-Codes:
C:\SORCUS\ML8\RT\BC\BC45

Der Eintrag in "Library Directories" müßte dann wie folgt aussehen:
C:\SORCUS\ML8\RT\BC\BC45;C:\BORLANDC\LIB;

C sucht jetzt bei dem Linkvorgang zuerst in Ihrem Entwicklungsverzeichnis nach den Start-Up-Codes und dann erst in dem Bibliotheksverzeichnis von C.

iese Einstellung gilt nur für die Entwicklung von Programmen, die auf der MODULAR-4/486 laufen. Jedes Projekt hat seine eigenen Optionen, wenn Sie mit der Projektentwicklung arbeiten. Das heißt, daß die Pfade in jedem Projekt eingestellt werden müssen. Für Ihre PC-Programme müssen Sie ein eigenes Verzeichnis erstellen. **!**

Wenn Sie mit Borland C 3.1 arbeiten, haben Sie die Wahl aus zwei verschiedenen Start-Up-Codes, die Sie in den Verzeichnissen 'BC31' bzw. 'BC31NOFP' finden. Letzteren sollten Sie verwenden, wenn Ihre Programme keine Floating-Point-Operationen enthalten, der erzeugte Code wird dann kleiner.

Falls beim Linken eines Echtzeitprogrammes die falschen Start-Up-Codes eingebunden werden, meldet der Linker 'undefined symbol **_wrong_startups_linked**'. Im umgekehrten Fall, also beim Linken der SORCUS Start-Up-Codes zu einem PC-Programm, lautet die Meldung in der Regel 'undefined symbol **SORCSPINIT** in Module xx'.

7.7.3. Programmierung

Nach der Einbindung der neuen Start-Up-Codes können Sie Programme mit C für die MODULAR-4/486 Karte erstellen. Beachten Sie bei der Programmierung die folgenden Hinweise:

!

- *Es sind keine Bildschirm- (Grafik oder Text) oder Tastaturfunktionen möglich.*
- *Es kann und darf keine Overlay-Technik verwendet werden.*
- *Es können keine Datei-Operationen durchgeführt werden.*
- *Es kann kein Speicher dynamisch über die Standardfunktionen von C reserviert oder freigegeben werden. Hierfür werden entsprechende Routinen von den Bibliotheken bereitgestellt.* ***Pointer-Operationen können jedoch ganz normal durchgeführt werden.***
- *Es dürfen keine DOS-spezifischen Funktionen bzw. Funktionen, die DOS-Interrupts verwenden, benutzt werden (muß im Zweifelsfall mit dem Turbo-Debugger überprüft werden).*
- *Das Einfügen von Debug-Informationen hat keinen Einfluß auf die Ausführung des Programms.*
- *Benutzen Sie das Speichermodell "LARGE".*
- *Bei Floating-Point-Operationen darf der Stack innerhalb der Prozedur nicht verändert worden sein. Durch* ***ml7rt_entry*** *bzw.* ***ml8rt_entry*** *wird der Stack verändert! Falls Sie also Fließkommaoperationen z.B. in der Hauptprozedur vornehmen möchten, so schreiben Sie bitte eine separate Prozedur, die dann innerhalb der Hauptprozedur aufgerufen wird.*
- *Verwenden Sie keine Registervariablen.*
- *Keine Floating-Point-Operationen in DI- und II-Tasks oder Prozeduren bzw. Funktionen, die von PC aus aufgerufen werden.*

Folgende Standardfunktionen von Borland C können ohne Einschränkungen verwendet werden:

fabs	cos	log	sizeof	isalpha	memcpy
exp	tan	log10	strlen	isascii	memmove
asin	ceil	poly	strcpy	isdigit	atoi
acos	floor	pow	strcmp	islower	itoa
atan	fmod	pow10	strcat	isupper	
sin	hypot	sqrt			

Dies ist nur ein Auszug der verwendbaren Routinen. Im Prinzip können alle Routinen zur Speichermanipulation und auch Stringfunktionen uneingeschränkt verwendet werden.

Alle Routinen, die unter die nachfolgenden Rubriken fallen, können bei der Echtzeitprogrammierung nicht verwendet werden: **!**

- *Verzeichnisroutinen*
- *I/O-Routinen*
- *Grafikroutinen*
- *Bildschirmroutinen*
- *Datum und Uhrzeit*
- *Dynamische Speicherverwaltung (z.B. Allokierung und Freigabe)*
- *Funktionen wie 'delay' und 'sound'*

7.7.4. Compilereinstellungen

Sie sollten die Compiler- und Linker-Optionen wie folgt einstellen:

Borland C 3.1:

Options:

- Compiler / Advanced-Code-Generation: Emulation
  80386-Code
  Generate Underbars
- Compiler / Entry-, Exit-Code: DOS standard
  Standard stack frame
- Compiler / Optimize / Optimize for: Speed
- Compiler / Optimize / Register Variables: None
- Compiler / Source-Options: BORLAND C++
- Linker / Libraries: keine zusätzlichen Libraries einbinden

Borland C 4.5 und 5.0:

TargetExpert:

- Zieltyp: Anwendung (.exe), Weiter Optionen: '.c Knoten'
- Umgebung: DOS (Standard)
- Zielmodell: Large
- Standardbibliotheken: Laufzeit
  'Emulation' oder 'keine Mathe-Unterstützung'

Projektoptionen:

- Compiler / Compiler-Ausgabe: Unterstriche erzeugen
- Compiler / Code-Generierung / Registervariablen: nicht verwenden
- 16-Bit Compiler / Prozessor: i486
- Optimierungen / Spezielle Optimierungen: hinsichtlich Geschwindigkeit (**nur bei Borland 4.5**)

7.7.5. Beispielprogramme für C++

7.7.5.1. NI-Task in C++

Das folgende Beispielprogramm für die "große" MODULAR-4/486 Karte zeigt das Blink-LED Programm als NI-Task. Sie finden dieses Programm auch unter dem Namen "M8P0300.C" auf der mitgelieferten Diskette bzw. CD. Zum Compilieren benutzen Sie bitte die ebenfalls auf der Diskette bzw. CD vorhandene Projektdatei "M8P0300.PRJ" (bzw. M8P0300.IDE). Bitte überprüfen Sie vor dem Compilieren die eingestellten Pfade ("Options/Directories"). Soll das Programm für die "kleine" MODULAR-4/486 Karte verwendet werden, müssen sämtliche "ml8..." Aufrufe durch "ml7..." ersetzt werden.

```
#include "ml8rtbib.h"                    /* Header für die Echtzeitbibliothek einbinden */
#include "dos.h"
#include "stdlib.h"
                                 /* ---- Allgemeine Angaben ---- */
#define PROGRAMM_NUMBER 0x300            /* Programmnummer = 300h */
#define VERSION         '1'              /* Version des Programms */
#define REVISION        'F'              /* Revision des Programms */
                                 /* ---- Die verschiedenen Statusmöglichkeiten des Programms ---- */
#define READY           0                /* Bereit */
#define RUNNING         2                /* Programm läuft */
#define ERROR           3                /* Falsche LED angewählt */
#define STOPPED         4                /* Programmlauf angehalten */
                                 /* ---- Zustände der LED ---- */
#define ON              1
#define OFF             0
                                 /* ---- Benutzte LED ---- */
#define INTERN          1                /* Interne LED */
#define EXTER           2                /* Externe LED */


struct pdt_type                  /* ---- Aufbau der Programm-Descriptor-Tabelle (PDT) ---- */
{
   ML8_PDT_HEAD           pdt_head;       /* PDT-Header */
   void* PFAR             main_proc;      /* Adresse der Haupt-Prozedur (Task) */
   void* PFAR             auto_init;      /* Adresse der Auto-Init Prozedur */
   void* PFAR             start;          /* Adresse der Start-Prozedur */
   void* PFAR             stop;           /* Adresse der Stop-Prozedur */
} pdt;

struct parameter_type            /* ---- Aufbau des Parameterbereichs ---- */
{
   tdt_type               tdt;            /* Platz für die Task-Descriptor-Tabelle */
   byte                   status;         /* Status des Programms (READY etc.) */
   ushort                 blink_rate;     /* Blinkrate */
   byte                   led_status;     /* Zustand der LED */
   byte                   led;            /* Welche LED 1=intern, 2=extern */
} parameter;
ushort                    pause;          /* Ein Zähler, der Durchläufe des Programms zählt*/

void PFAR auto_init (void);                /* Prototypen der benutzten Prozeduren */
void PFAR start (void);
void PFAR stop (void);
void PFAR main_task (void);
```

```
void PFAR auto_init (void)     /* ---- AUTO-INITIALISIERUNG ---- */
{
   ml8rt_entry();                          /* Register retten und vorbereiten */
   parameter.status = READY;               /* Status des Programms auf "BEREIT" */
   parameter.blink_rate = 10000;           /* Blinkrate auf einen Defaultwert einstellen */
   parameter.led_status = OFF;             /* Status der LED auf "AUS" */
   parameter.led = INTERN;                 /* Standardmäßig die interne LED benutzen */
   ml8rt_external_led_off();               /* Die externe LED ausschalten */
   ml8rt_local_led_off();                  /* Die On-Board-LED ausschalten */
   pause = 0;                              /* Durchlaufzähler auf Null */
   ml8rt_exit();                           /* Register wieder restaurieren */
};
void PFAR start (void)         /* ---- Start-Prozedur ---- */
{
   ml8rt_entry();
   if ((parameter.led != INTERN) && (parameter.led != EXTER))
                                           /* Als erstes die angewählte LED überprüfen */
      parameter.status = ERROR;            /* Fehlermeldung im Status-Parameter */
   else {
      parameter.status = RUNNING;          /* Falls alles OK, Task aktivieren (starten) */
      ml8rt_wakeup_task (parameter.tdt.task);
   }
   ml8rt_exit();
}

void PFAR stop (void)          /* ---- Stop-Prozedur ---- */
{
   ml8rt_entry();                          /* Programm deaktivieren (stoppen) */
   ml8rt_sleep_task (parameter.tdt.task);
   parameter.status = STOPPED;
   ml8rt_exit();
}

void PFAR main_task (void)     /* ---- HAUPTPROZEDUR DER TASK ---- */
{
   ml8rt_entry();
   if (++pause >= parameter.blink_rate) {
      if (parameter.led_status == ON) {    /* Falls die LED eingeschaltet ist, ... */
         if (parameter.led == EXTER)       /* ... LED ausschalten und ... */
            ml8rt_external_led_off();
         else
            ml8rt_local_led_off();
         parameter.led_status = OFF;       /* ... Zustand merken */
      }
      else {                               /* Falls die LED ausgeschaltet ist, ... */
         if (parameter.led == EXTER)
            ml8rt_external_led_on();       /* ... LED einschalten ... */
         else
            ml8rt_local_led_on();
         parameter.led_status = ON;        /* ... Zustand merken */
      }
      pause = 0;                           /* Durchlaufzähler wieder auf Null setzen */
   }
   ml8rt_exit();
}

void main ()                  /* ---- PREPARE ---- */
{
   pdt.pdt_head.ucType = 1;                    /* PDT-Typ (z.Zt. nur Typ 1 unterstützt)*/
   pdt.pdt_head.ucSize = sizeof(ML8_PDT_HEAD); /* Länge des PDT-Vorspanns (im Moment = 48) */
   pdt.pdt_head.usGlobalProc = 4;              /* Anzahl der Prozeduren */
   pdt.pdt_head.usProgNo = PROGRAMM_NUMBER;    /* Programm-Nummer */
   pdt.pdt_head.cVersion = VERSION;            /* Programm-Version */
   pdt.pdt_head.cRevision = REVISION;          /* Programm-Revision */
   pdt.pdt_head.ucCPU = _486SX;                /* CPU-Type */
   pdt.pdt_head.ucCoProc = _NOCOPROZ;          /* Co-Prozessor-Typ */
   pdt.pdt_head.ucLanguage = _CPP31;           /* Programmiersprache = C++ */
   pdt.pdt_head.ucProgType = _USER_PRG;        /* Programm-Typ: Anwenderprogramm */
   pdt.pdt_head.usFlags = 0;                   /* Flags */
   pdt.pdt_head.usFlags = _NI_TASK + _PDT_INFO_ENABLE + _LOCAL_DATA +
                          _LOCAL_PARAMETER+ _FIXED_DATASIZE;
   pdt.pdt_head.usInt = 0;                     /* Interrupt-Nr für die das Prg. gedacht ist
                                                  (nicht relevant, NI-Task) */
   pdt.pdt_head.ulDataAdr = 0;                 /* Adresse des Datenbereichs (kein Datenbereich) */
   pdt.pdt_head.ulDataSize = 0;                /* Größe des Datenbereichs */
   pdt.pdt_head.ulDataSizeMin = 0;             /* Minimale Datenbereichsgröße */
```

```
   pdt.pdt_head.ulDataSizeMax = 0;             /* Maximale Datenbereichsgröße */
                                               /* Anfangsadresse des Parameterbereichs errechnen */
   ml8rt_phys_adr (&parameter, &pdt.pdt_head.ulGlobalParAdr);
   pdt.pdt_head.ulGlobalParAdr = pdt.pdt_head.ulGlobalParAdr + sizeof(tdt_type);
                                               /* Anzahl der Parameter eintragen */
   pdt.pdt_head.usGlobalParSize = sizeof(parameter) - sizeof(tdt_type);
   pdt.pdt_head.ulHyperAdr = 0;                /* Adresse des Hypertextes (wird nicht verwendet)*/
   pdt.pdt_head.main_proc = main_task;         /* Adresse der Hauptprozedur */
   pdt.pdt_head.auto_init = auto_init;         /* Adresse der Auto_Init Prozedur */
   pdt.pdt_head.start = start;                 /* Adresse der Start-Prozedur */
   pdt.pdt_head.stop = stop;                   /* Adresse der Stop-Prozedur */
   ml8rt_set_pdt_adr (&pdt);                   /* Adresse der PDT dem Betriebssystem mitteilen */
}
```

Da das Programm insgesamt ausführlich kommentiert ist, soll nur auf Besonderheiten eingegangen werden.

Um dieses Programm zu compilieren, öffnen Sie bitte die Projektdatei "M8P0300.PRJ" (bzw. M8P0300.IDE). Beachten Sie bitte, daß die Datei "ML8RTBIB.LIB" in Ihre Projektdatei eingebunden werden muß. Nähere Informationen zu dieser Bibliothek finden Sie im Kapitel 9 dieses Handbuchs, wo alle Routinen der Bibliothek "ML8RTBIB" ausführlich erläutert sind.

Zuerst wird der Aufbau der PDT als Struktur deklariert. Das Programm enthält vier Prozeduren (Main-Proc, Auto-Init, Start und Stop).

Dann wird eine Struktur für die Parameter vereinbart. Der Speicher für die Parameter wird vom Programm bereitgestellt. Bei der Deklaration ist es wichtig, daß vor den eigentlichen Parametern Platz für die TDT reserviert wird.

Alle globalen (also in der PDT aufgeführten Prozeduren) beginnen mit **ml8rt_entry** und enden mit **ml8rt_exit**. Dadurch ist sichergestellt, daß alle Prozessor-Register gesichert und für die Prozedur richtig eingestellt werden.

In der Auto-Init-Prozedur werden die Parameter initialisiert und die LED ausgeschaltet. In der Prozedur "Init_PDT" wird die PDT initialisiert. Beachten Sie bitte die Einstellung der PDT-Flags.

Die Prozedur **main** enthält in einem "normalen" C-Programm das eigentliche Hauptprogramm. Dieser Teil, der hier als "PREPARE" bezeichnet wird, wird während der Installierung abgearbeitet. Er sorgt dafür, daß die PDT initialisiert wird (Init_PDT) und dem MODULAR-4/486 Betriebssystem die Adresse der PDT mitgeteilt wird (ml8rt_set_pdt_adr).

Die Datei "DOS.H" sollte in alle Programme eingebunden werden. Dadurch werden einige Prozeduren (z.B. die Port-Zugriffe **outportb**, **inportb**, ...) wesentlich schneller ausgeführt.

7.7.5.2. Installierung der NI-Task

Nachdem das Projekt "M8P0300.PRJ" einwandfrei übersetzt und die Datei "M8P0300.EXE" erzeugt wurde, kann das Programm auf die Karte geladen und gestartet werden. Geben Sie dazu bitte folgende Installationsdatei mit Hilfe von SNW ein, und laden Sie diese Datei. Sie können auch die Beispielinstallationsdatei "NI-LED.INS" benutzen.

```
; Beispielinstallation: LED-Blinken als NI-Task (ohne Interrupt)


; "große" Karte anwählen, Adresse einstellen und Reset auslösen
M8DEVICE 0380 TIMEOUT=10 RESET

; Programmnummer 300h unter Tasknummer 21h installieren (nicht aktivieren)
M8INST M8P0300.EXE 0300 0021 00 000000 00000980

; Parameter der Task 21h setzen
; PAR-1,2 : Blinkrate der LED    => 1000h
; PAR-3   : Status der LED       => 0 = Aus
; PAR-4   : Benutzte LED         => 1 = On-board LED

M8PAR 21 01 00 10 00 01

; Durch Aufruf der Start-Prozedur Programm starten (Proz. 2)
M8PROC 21 02
```

Diese Installationsdatei installiert die NI-Task, stellt die Blinkrate ein und aktiviert die Task durch Aufruf der Prozedur 2.

7.7.5.3. DI-Task in C++

Das folgende Listing zeigt das Blink-LED Programm für die "kleine" MODULAR-4/486 Karte als DI-Task. Sie finden das Programm auch unter dem Namen "M7P0301.C" auf der mitgelieferten Diskette bzw. CD. Zum Compilieren benutzen Sie bitte die Projektdatei "M7P0301.PRJ" (bzw. M7P0301.IDE), die sich ebenfalls auf der Diskette bzw. CD befindet. Das Programm soll unter dem Interrupt von TIMER-A installiert werden. Die Blinkrate soll in Hertz angegeben werden. Soll das Programm für die "große" MODULAR-4/486 Karte verwendet werden, müssen sämtliche "ml7..." Aufrufe durch "ml8..." ersetzt werden.

```
#include "ml7rtbib.h"                  /* Echtzeitbibliothek einbinden */
#include "ml7makro.h"                  /* Makro-Definitionen; TASM notwendig!*/
#include "stdlib.h"                    /* (optional!) */
#include "dos.h"

/* ---- Allgemeine Angaben ---- */
#define PROGRAMM_NUMBER 0x301          /* Programmnummer = 301h */
#define VERSION         '1'            /* Version des Programms */
#define REVISION        'C'            /* Revision des Programms */

/* ---- Die verschiedenen Statusmöglichkeiten des Programms ---- */
#define READY           0              /* Bereit */
#define RUNNING         2              /* Programm läuft */
#define ERROR           3              /* Falsche LED angewählt */
#define STOPPED         4              /* Programm wurde angehalten */
/* ---- Zustände der LED ---- */
#define ON              1
#define OFF             0

/* ---- Benutzte LED ---- */
#define INTERN          1              /* Interne LED */
#define EXTER           2              /* Externe LED */

/* ---- Timer - Initialisierungen---- */
/* Timer A 16-bit-> (65535/ 2), da Periodendauer T = t(LED_ON) + t(LED_OFF) */
#define TIMER_VAL  30000UL
/* Timer A Basistakt Standard 2.5MHz (evtl. 10 MHz) */
#define TIMER_FREQ 2500000UL



/* ---- Aufbau der Programm-Descriptor-Tabelle (PDT) ---- */
struct pdt_type
{
   ML7_PDT_HEAD   pdt_head;                /* PDT-Header */
   void* PFAR     main_proc;               /* Adresse der Haupt-Prozedur */
   void* PFAR     auto_init;               /* Adresse der Auto-Init Prozedur */
   void* PFAR     start_proc;              /* Adresse der Start-Prozedur */
   void* PFAR     stop_proc;               /* Adresse der Stop-Prozedur */
} pdt;

/* ---- Aufbau des Parameterbereichs ---- */
struct parameter_type
{
   tdt_type              tdt;         /* Platz für Task-Descriptor-Tabelle */
   byte                  status;      /* Status des Programms (READY etc.) */
   ushort                blink_rate;  /* Blinkrate */
   byte                  led_status;  /* Zustand der LED */
   byte                  led;         /* Welche LED 1=intern, 2=extern */
   byte                  soft_cnt;    /* Soft-Counter noetig, um mit Timer */
                                      /* auch niedrige Freq. zu erzeugen!  */
   byte                  soft_init;   /* Soft-Counter Initialisierungswert */
} parameter;
```

```
/* Prototypen der benutzten Prozeduren */
void PFAR auto_init (void);
void PFAR main_proc (void);
void PFAR start (void);
void PFAR stop (void);


/* ---- AUTO-INITIALISIERUNG ---- */
void auto_init (void)
{
   ml7rt_entry();                      /* Register retten und vorbereiten */
   parameter.status = READY;           /* Status des Programms auf "BEREIT" */
   parameter.blink_rate = 1;           /* Blinkrate Defaultwert von 1 Hz */
   parameter.led_status = OFF;         /* Status der LED auf "AUS" */
   parameter.led = INTERN;             /* Standardmäßig interne LED */
   ml7rt_external_led_off();           /* Die externe LED ausschalten */
   ml7rt_local_led_off();              /* Die On-Board-LED ausschalten */
   ml7rt_exit();                       /* Register wieder restaurieren */
}


void PFAR start (void)
{
   unsigned short timer;

   ml7rt_entry();

   /* Falls nicht die richtige LED angegeben wurde, Fehler zurückgeben */
   if ((parameter.led != INTERN) && (parameter.led != EXTER))
      parameter.status = ERROR;
   else {
      /* Softzaehler ausrechnen (muss ganzzahlig sein, keine Fliesskommaop.!):  */
      parameter.soft_init = TIMER_FREQ / (2 * parameter.blink_rate * TIMER_VAL);
      /* falls Softzaehler auf 0 "abgerundet" wird, setze Zaehler auf 1 */
      if (parameter.soft_init == 0) parameter.soft_init = 1;
         parameter.soft_cnt = parameter.soft_init;
      /* Timerwert ausrechnen: Fehler durch Softteiler (ganzzahlig!) wird durch
         Anpassung des maximalen Timerwerts kompensiert! */
      timer = TIMER_FREQ / (2 * parameter.blink_rate * parameter.soft_init);
      /* Timer auf den berechneten Wert setzen und starten */
      ml7rt_set_timer (TIMER_A, timer, INT_MODE);
      parameter.status = RUNNING;
      ml7rt_unmask_int (IRQ_TIMER_A);  /* Timer-Interrupt freigeben! */
   }
   ml7rt_exit();
}


void PFAR stop (void)
{
   ml7rt_entry();
   ml7rt_mask_int (IRQ_TIMER_A);       /* Timer-Interrupt sperren */
   parameter.status = STOPPED;         /* Status auf "abgebrochen" */
   ml7rt_exit();
}
```

```
/* ---- HAUPTPROZEDUR DER TASK ---- */
void PFAR main_proc (void)
{
   ml7rt_entry();
   parameter.soft_cnt--;               /* Soft-Counter inkrementieren */
   if (parameter.soft_cnt == 0) {      /* Soft-Counter schon auf Null? */
      if (parameter.led_status == ON) {/* Wenn ja, dann LED umschalten */
         if (parameter.led == EXTER)   /* Falls die LED eingeschaltet ist, */
            ml7rt_external_led_off();  /* ... LED ausschalten ... */
         else
            ml7rt_local_led_off();
         parameter.led_status = OFF;   /* ... Zustand merken */
      }
      else {                            /* Falls die LED ausgeschaltet ist, */
         if (parameter.led == EXTER)
            ml7rt_external_led_on();   /* ... LED einschalten ... */
         else
            ml7rt_local_led_on();
         parameter.led_status = ON;    /* ... Zustand merken */
      }                                /* Software-Zaehler rücksetzen */
      parameter.soft_cnt = parameter.soft_init;
   }
   ml7rt_end_of_int (IRQ_TIMER_A);     /* EOI an den Interrupt-Contr. */
   ml7rt_exit_interrupt();
}
/* ---- PDT-INITIALISIEREN ---- */
void main (void)                       /* ---- PREPARE ---- */
   pdt.pdt_head.ucType = 1;                    /* PDT-Typ (z.Zt. nur Typ 1 unterstützt)*/
   pdt.pdt_head.ucSize = sizeof(ML7_PDT_HEAD); /* Länge des PDT-Vorspanns (im Moment = 48) */
   pdt.pdt_head.usGlobalProc = 4;              /* Anzahl der Prozeduren */
   pdt.pdt_head.usProgNo = PROGRAMM_NUMBER;    /* Programm-Nummer */
   pdt.pdt_head.cVersion = VERSION;            /* Programm-Version */
   pdt.pdt_head.cRevision = REVISION;          /* Programm-Revision */
   pdt.pdt_head.ucCPU = _486DX;                /* CPU-Type */
   pdt.pdt_head.ucCoProc = _NOCOPROZ;          /* Co-Prozessor-Typ */
   pdt.pdt_head.ucLanguage = _CPP31;           /* Programmiersprache = C++ */
   pdt.pdt_head.ucProgType = _USER_PRG;        /* Programm-Typ: Anwenderprogramm */
   pdt.pdt_head.usFlags = 0;                   /* Flags */
   pdt.pdt_head.usFlags = _DI_TASK + _PDT_INFO_ENABLE + _LOCAL_DATA +
                          _LOCAL_PARAMETER+ _FIXED_DATASIZE;
   pdt.pdt_head.usInt = IRQ_TIMER_A;           /* Interrupt-Nr für die das Prg. gedacht ist
                                                  (nicht relevant, NI-Task) */
   pdt.pdt_head.ulDataAdr = 0;                 /* Adresse des Datenbereichs (kein Datenbereich) */
   pdt.pdt_head.ulDataSize = 0;                /* Größe des Datenbereichs */
   pdt.pdt_head.ulDataSizeMin = 0;             /* Minimale Datenbereichsgröße */

   pdt.pdt_head.ulDataSizeMax = 0;             /* Maximale Datenbereichsgröße */
                                               /* Anfangsadresse des Parameterbereichs errechnen */
   ml8rt_phys_adr (&parameter, &pdt.pdt_head.ulGlobalParAdr);
   pdt.pdt_head.ulGlobalParAdr = pdt.pdt_head.ulGlobalParAdr + sizeof(tdt_type);
                                               /* Anzahl der Parameter eintragen */
   pdt.pdt_head.usGlobalParSize = sizeof(parameter) - sizeof(tdt_type);
   pdt.pdt_head.ulHyperAdr = 0;                /* Adresse des Hypertextes (wird nicht verwendet)*/
   pdt.pdt_head.main_proc = main_task;         /* Adresse der Hauptprozedur */
   pdt.pdt_head.auto_init = auto_init;         /* Adresse der Auto_Init Prozedur */
   pdt.pdt_head.start = start;                 /* Adresse der Start-Prozedur */
   pdt.pdt_head.stop = stop;                   /* Adresse der Stop-Prozedur */
   ml7rt_set_pdt_adr (&pdt);                   /* Adresse der PDT im Rückgabe-Record dem
                                                   Betriebssystem mitteilen */
}
```

Da das Listing ausführlich dokumentiert ist, soll nur auf die Besonderheiten eingegangen werden. Zum Compilieren benutzen Sie bitte die Projektdatei "M7P0301.PRJ" (bzw. M7P0301.IDE).

In der PDT werden Sie erkennen, daß das Programm vier Prozeduren enthält (Main-, Auto-Init-, Start- und die Stop-Prozedur).

Die Start-Prozedur berechnet zuerst aus der angegebenen Frequenz die Timer-Werte. Beachten Sie bitte dabei, daß die Timer mit einer Eingangsfrequenz von 2,5 MHz

getaktet werden, und daß jeder Timer 16 Bit breit ist. Da mit diesen Werten "nur" Blinkfrequenzen zwischen 2,5 MHz und 38 Hz möglich sind, wurde der Hardware-Timer durch einen Softwarezähler erweitert. Erst wenn dieser Softwarezähler auf Null gelaufen ist, wird die LED umgeschaltet. Bei der Programmierung wurde darauf geachtet, daß der Timer-Wert möglichst groß ist, damit die Anzahl der Interrupts vom Timer pro Zeiteinheit möglichst klein ist.

Die Hauptprozedur unterscheidet sich ein wenig von der Hauptprozedur der NI-Task. Es handelt sich praktisch um eine Interrupt-Service Routine. Am Ende wird deshalb den Interrupt-Controllern das Ende der Interrupt-Prozedur angezeigt (EOI = End of Interrupt). Anschließend wird die Prozedur mit **ml7rt_exit_interrupt** beendet.

Grundsätzlich könnte die Hauptprozedur auch als Interrupt-Prozedur deklariert werden, jedoch werden dabei die erweiterten Register des i486 nicht gerettet.

7.7.5.4. Installierung der DI-Task

Nachdem das Projekt "M7P0301.PRJ" einwandfrei übersetzt und die Datei "M7P0301.EXE" erzeugt wurde, kann das Programm auf die Karte geladen und gestartet werden. Geben Sie dazu bitte folgende Installationsdatei mit Hilfe von SNW ein, und laden Sie diese *.INS Datei. Sie können auch die Beispielinstallationsdatei "DILED.INS" benutzen.

```
; Beispielinstallation: LED-Blinken mit Timer-Interrupt (TIMER-A)
;
; "kleine" Karte anwählen, Adresse einstellen und Reset auslösen
M7DEVICE 0380 TIMEOUT=10 RESET
;
;
; Programm-Nummer 301h unter der Tasknummer 22h installieren
; (Interrupt 91h => TIMER-A)
M7INST M7P0301.EXE 0301 0022 91 000000 0982
;
;
; Parameter der Task 22h setzen
; PAR-0   : Status des Programms
; PAR-1,2 : Blinkrate der LED    => 5 Hz
; PAR-3   : Status der LED       => 0 = Aus
; PAR-4   : Benutzte LED         => 1 = On-Board LED
;
M7PAR 22 01 05 00 00 01
;
;
; Durch Aufruf der Start-Prozedur (Proz. 2) das Programm starten
M7PROC 22 02
```

Diese Installationsdatei installiert die DI-Task und aktiviert sie anschließend.

Sie könnten als Übung ja einmal versuchen, das vorhandene Programm unter einen anderen Namen zu kopieren und es so zu modifizieren, daß dieses neue Programm z.B. den TIMER-B benutzt.

7.7.6. Die Makrobibliotheken "ML7MACRO.H" und "ML8-MACRO.H"

Bislang wurden alle Funktionen aus der Bibliothek "ML7RTBIB" und "ML8RTBIB" ganz normal aufgerufen. Für die wichtigsten Funktionen (wie z.B. **ml8rt_entry**, **ml8rt_exit**) gibt es jedoch einen Weg, diese Methode zu optimieren. Die Header-Dateien "ML7MACRO.H" und "ML8MACRO.H" stellen diese Funktionen als Makros zur Verfügung. Der entsprechende Programmcode der Subroutine wird direkt an der entsprechenden Programmstelle eingefügt. Ein separater Aufruf einer Unterprozedur und das aufwendige Speichern der Parameter auf den Stack entfällt. Die Programme werden dadurch wesentlich schneller.

Sie können die Makros einfügen, indem Sie die Makrobibliothek direkt hinter dem Header der "ML7RTBIB" bzw. "ML8RTBIB" einbinden. Die Makros haben die gleichen Namen wie die jeweiligen Funktionen. Es ist lediglich ein '_' vorangestellt (z.B. **_ml8rt_entry**). Vorher ist im Menü "Options/Transfer" der Pfad zum Turbo-Assembler einzugeben (wird normalerweise defaultmäßig während der Installierung von Borland C eingestellt).

7.8. Allgemeine Hinweise zur Programmierung

7.8.1. Modul-Device-Treiber

Zur Programmierung der I/O-Devices auf den Modulen und der MODULAR-4/486 Basiskarte können entweder die Modulbibliotheken oder die neueren Modul-Device-Treiber (MDD) verwendet werden (siehe detaillierte Beschreibung in Anhang O). Beides finden Sie auf den mitgelieferten Disketten bzw. CD und im Internet unter **www.sorcus.com**.

Die Modul-Device Treiber gehen von einem Kanal-orientierten Ansatz aus. Der Anwender 'öffnet' einen Kanal zu einem oder mehreren Devices, z.B. einem Analog-Eingang. Beim Öffnen werden Kanal-spezifische Parameter wie z.B. der Meßbereich übergeben und ein gültiges Handle zurückgeliefert. Mit dem Handle kann nun sehr einfach auf das Device zugegriffen werden (z.B. Lesen eines Analogwertes). Wird ein Kanal nicht mehr benötigt, kann er geschlossen werden. Dabei wird der vorher belegte Speicherplatz wieder freigegeben.

Ein weiterer Vorteil der Modul-Device-Treiber ist ihre Multitasking-Tauglichkeit. Ein MDD verwaltet sämtliche Funktionseinheiten eines Moduls und stellt eine normierte Schnittstelle für den Zugriff auf die Funktionseinheiten zur Verfügung.

Modul-Device-Treiber können sowohl von PC-Anwendungen als auch von Echtzeitprogrammen auf der Karte **gleichzeitig** genutzt werden. Aus diesem Grunde müssen für die Basiskarte und für jedes aufgesteckte SP-Bus-Modul ein zugehöriger MDD geladen und installiert werden. Dieses muß nach jedem Hardware-Reset der MODULAR-4-Karte geschehen. Als Tasknummer des MDD wird die Nummer des zugehörigen Modulsteckplatzes bzw. 11 für die Basiskarte verwendet.

Nähere Informationen zur Verwendung der Modul-Device-Treiber finden Sie im Anhang O.

7.8.2. FAR-Compilierung der Prozeduren

Die Prozeduren, die dem Betriebssystem über die PDT bekannt gemacht werden, müssen immer als **FAR** deklariert werden. Stellen Sie dazu sicherheitshalber die entsprechenden Compilerschalter fest ein.

7.8.3. Lesen von Parametern und Daten (Datenaustausch zwischen Tasks)

Die Routinen innerhalb der ML7RTBIB bzw. ML8RTBIB ermöglichen eine einfache Intertask-Kommunikation auf der Karte. Besonders der Datenaustausch zwischen den einzelnen Tasks ist sehr einfach. Bedenken Sie aber, daß der Datenaustausch auch Zeit kostet. Vermeiden Sie deshalb die Übertragung von sehr großen Datenmengen. Müssen Datenblöcke wirklich ausgetauscht werden oder genügt es, wenn nur ein Zeiger auf diese Daten übergeben wird?

Zum Lesen eines Bytes, Wortes, oder Doppelwortes einer anderen Task benutzen Sie die entsprechenden Prozeduren. Die Benutzung von Byte-, Wort- oder Doppelwort-Zugriffen ist in diesen Fällen schneller als der Blockzugriff.

7.8.4. Geschwindigkeitsaspekte

Obwohl die Bibliotheken geschwindigkeitsoptimiert sind, kann es sein, daß sich eine direkte (Assembler-) Programmierung nicht immer vermeiden läßt. Meistens reicht es aber in solchen Fällen aus, nur eine bestimmte, sehr zeitkritische Routine zu optimieren. Oft hilft es auch, häufig benötigte Adressen (z.B. von Parametern oder Daten) einmal zu ermitteln, um dann direkt (ohne Aufruf einer Prozedur) auf diese Bereiche zuzugreifen.

7.8.5. Warten auf Ereignisse

Viele DOS-Programme verbringen den größten Teil der Zeit damit, auf Eingaben vom Benutzer zu warten. Unter DOS ist das kein Problem. Da die gesamte Rechenleistung ohnehin nur einem Programm zur Verfügung steht, kann dieses Programm die Tastatur einfach in einer Schleife abfragen und erst dann etwas tun, wenn eine Taste gedrückt ist.

Ganz anders sieht das im Multi-Tasking-Betrieb der MODULAR-4 aus. Wenn eine Task auf das Schließen eines Schalters oder auf das Eintreffen eines Zeichens an einer Schnittstelle warten muß, bevor sie fortfahren kann, dann darf das nicht in einer Schleife wie bei DOS geschehen. In diesem Fall würde nicht nur die eine Task, sondern alle anderen Tasks (zumindest alle NI-Tasks) blockiert und kämen nicht an die Reihe. Statt einer direkten Schleife sollten Sie für die Abfrage eine NI-Task verwenden. Prüfen sie, ob das erwartete Ereignis eingetreten ist. Wenn ja, können Sie entsprechend der Aufgabenstellung fortfahren. Wenn das Ereignis noch nicht eingetreten ist, sollten Sie die NI-Task beenden und beim nächsten Durchlauf erneut prüfen, ob das Ereignis, eingetreten ist.

Beispiel:

Eine Task soll Programmteil I abarbeiten und anschließend warten, bis ein Schalter geschlossen ist. Dann soll Programmteil II abgearbeitet werden und wieder von vorne begonnen werden. In einem klassischen DOS Programm sähe das dann so aus:

1. Programmteil I ausführen
2. Schleife bis Schalter geschlossen
3. Programmteil II ausführen
4. Springe zu 1.

Als **NI-Task** könnte das Programm wie folgt aussehen:

Start-Prozedur oder **Auto_Init**:

...

Statusvariable = 1 setzen

...

Haupt-Prozedur:

1. Statusvariable prüfen
   a) Status = 1: Weiter bei 2.1
   b) Status = 2: Weiter bei 3.1

2.1. Programmteil I ausführen
2.2. Statusvariable = 2 setzen
2.3. Hauptprozedur verlassen (oder bei 3.1 weiter)

3.1. Schalter prüfen
   a) Schalter offen: Hauptprozedur beenden
   b) Schalter geschlossen: Weiter bei 3.2.
3.2. Programmteil II ausführen
3.3. Statusvariable = 1 setzen
3.4. Hauptprozedur verlassen

Die Reaktionszeit auf das Schließen des Schalters ist in diesem Beispiel abhängig davon, wie stark die MODULAR-4 Karte mit anderen Tasks ausgelastet ist, also letztlich davon, wie oft die NI-Task aufgerufen wird. Wenn Sie sehr schnell (im Mikrosekundenbereich) auf das Schließen des Schalters reagieren müssen, sollten Sie dafür sorgen, daß der Schalter einen Interrupt auslöst und den Programmteil II in einer Interrupt-Task erledigen.

7.8.6. Ermitteln der eigenen Tasknummer

Es gibt Fälle, in denen eine Task bestimmen muß, unter welcher Tasknummer sie selbst installiert ist, zum Beispiel dann, wenn mit einem Systemaufruf Daten in den Datenbereich geschrieben werden sollen (ml8rt_write_data_xxxx). Je nach Installierung der Task gibt es eine oder zwei Möglichkeiten, die Tasknummer zu bestimmen.

Wenn der Parameterbereich Teil des Programmes ist, dann ist zwingend auch die Task-Deskriptor-Tabelle (TDT) im Programm enthalten. Die Tasknummer steht als Eintrag in der TDT. Wenn Sie für die Reservierung der TDT die vordefinierte Struktur (TDT_Type, siehe Kapitel 9) aus ml8rtbib verwendet haben, können Sie die Tasknummer direkt aus dem Eintrag **task** lesen.

In Fällen, wo der Parameterbereich nicht vom Programm selbst, sondern vom Betriebssystem vergeben wird, kann die Tasknummer nicht aus der TDT ermittelt werden, da die TDT in diesem Fall nur unter Angabe der Tasknummer zugreifbar ist. In diesem Fall verwenden Sie statt der Funktion **ml7rt_entry/ml8_entry** die Funktion **ml7rt_entry_task/ml8rt_entry_task**, die die Tasknummer als Funktionsergebnis zurückliefert.

7.8.7. Verwenden von Fließkommaoperationen

Die Verwendung von Fließkommaoperationen unterliegt bei der Erstellung von Echtzeitprogrammen zwei Einschränkungen.

Die Emulation des Coprozessors ist nicht reentrant. Das heißt, daß eine laufende Berechnung nicht von einer Funktion unterbrochen werden darf, die ebenfalls Fließkommaoperationen enthält. Sicherheitshalber sollten deshalb bei Karten ohne Coprozessor keine Fließkommaberechnungen in Interruptprozeduren bzw. in Prozeduren, die vom PC aus aufgerufen werden, enthalten sein. Wenn Sie einen Coprozessor besitzen, müssen Sie - falls die Gefahr besteht, daß eine Berechnung von einer anderen unterbrochen wird - dafür sorgen, daß der Coprozessor-Status gesichert wird. Die Bibliotheken ML7RTBIB und ML8RTBIB stellen dafür zwei Routinen zur Verfügung (ml7rt_store_80487/ml8rt_store_80487 und ml7rt_restore_80487/ml8rt_restore_80487).

In der Programmiersprache C gilt zusätzlich die Einschränkung, daß innerhalb von Prozeduren, die mit ml7rt_entry/ml8rt_entry oder ml7rt_entry_func/ml8rt_entry_func beginnen, keine Fließkommaoperationen erlaubt sind. Statt dessen müssen die Berechnungen in einer eigenen (lokalen) Prozedur durchgeführt werden, die nicht mit ml7rt_entry/ml8rt_entry beginnt (und auch nicht in die PDT eingetragen wird). Diese Prozedur kann dann beliebig aufgerufen werden.

Beispiel:

```
void fp_calculations (void)
{
  ... /* beliebige Berechnungen */
}

void xyz (void)
{
  ml8rt_entry();
  ...
  fp_calculations();
  ...
  ml8rt_exit();
}
```

Beachten Sie, daß bereits die Zuweisung einer Fließkommazahl eine Fließkommaoperation ist, für die alle oben beschriebenen Einschränkungen gelten. **!**

7.8.8. Objektorientierte Programmierung

Die Erstellung von objektorientierten Programmen mit C++ oder Pascal ist grundsätzlich möglich. Da aber die dynamische Speicherverwaltung von C++ bzw. Pascal auf der Karte nicht unterstützt wird, müssen alle Objekte statisch deklariert werden, dürfen also nicht auf dem Heap liegen. Achten Sie unbedingt darauf, daß auch die 'Vorfahren' eines Objektes keine dynamischen Speicherzugriffe (New, Dispose, ...) enthalten dürfen.

7.8.9. Mehrfachinstallation von Echtzeitprogrammen

Die mehrfache Installation eines Programmes bedeutet, daß der Programmcode nur einmal im RAM der Karte vorhanden ist. Jede Instanz eines mehrfach installierten Programms hat jedoch ihre eigene Tasknummer, sowie ihren eigenen Parameter- und Datenbereich. Dies hat den Vorteil, daß weniger RAM-Speicher benötigt wird. Bei der Programmierung müssen jedoch einige Punkte beachtet werden.

In den Flags der Programm-Deskriptor-Tabelle (PDT) des Echtzeitprogrammes müssen die Bits 8 und 10 = 0 gesetzt sein (s. Anhang I). Dies bewirkt, daß der Parameter- und Datenbereich nicht vom Programm selbst, sondern vom Betriebssystem reserviert wird. Da dann Parameter- bzw. Datenbereich vom Programmcode getrennt sind, kann im Programm nicht direkt auf den eigenen Parameter- bzw. Datenbereich zugegriffen werden. Statt dessen müssen die Betriebsystem-Routinen dazu verwendet werden (z.B. ml8_read_par_byte oder ml8_write_data_dword).

Die PDT-Angabe 'Anfangsadresse Parameterbereich' wird in diesem Fall nicht ausgewertet. Dadurch ist eine Vereinbarung einer Variablen, die den Parameterbereich bildet überflüssig.

Wenn in den globalen Prozeduren des Programms Zugriffe auf den eigenen Parameter- bzw. Datenbereich erfolgen (bei denen die Angabe der Tasknummer erforderlich ist), müssen diese Prozeduren deshalb mit **ml7rt_entry_task/ml8rt_entry_task** (statt ml7rt_entry/ml8rt_entry) begonnen werden. Dieses ist die einzige Möglichkeit zu erfahren welche Instanz des Programms die Prozedur aufgerufen hat.

Die Mehrfachinstallation eines Programms kann nicht mit ml8_transfer_and_install geschehen, sondern erfordert zwei Schritte:

1. Laden des Programmcodes in das RAM der Karte mit der Funktion **ml7_transfer_pgm** bzw. **ml8_transfer_pgm**. Die Funktion liefert die physikalische Prepare-Adresse und die Installations-Flags zurück.
2. Anschließend können mit der Funktion **ml7_install_task** bzw. **ml8_install_task** unter Angabe der Tasknummer mehrere Instanzen des Programmes erzeugt werden. Dazu geben Sie die Flags und die Prepare-Adresse an, die von ml7_transfer_pgm bzw. ml8_transfer_pgm geliefert wurde.

Bei der Installation aus SNW ist im Menüpunkt 'Linken/Laden' unter 'Optionen' der Punkt 'Mehrfach Installierung' anzuwählen.

8. Remote-Debugging

8.1. Remote-Debugging mit RTDS

8.1.1. Was ist RTDS?

SORCUS RTDS ("Real-Time Development Studio") ist eine Windows-Umgebung für die Entwicklung von Echtzeitprogrammen für die SORCUS-Karten Multi-LAB/2, Multi-COM, MODULAR-4/486 und MAX-PC.
Die Entwicklungsumgebung besteht aus einem komfortablen Editor, einem integrierten Debugger und einer Projektverwaltung zur automatischen Erzeugung von Make-Dateien. Im RTDS nicht enthalten sind allerdings Compiler, Linker und das Make-Tool selbst. Zur eigentlichen Programmerzeugung wird daher auf die entsprechenden Tools des Borland-C++-Compilers zurückgegriffen. Für Borland Pascal fehlt die Unterstützung zur Zeit noch.

Voraussetzungen:
PC mit Windows 98, NT, 2000
Borland C++

8.1.2. RTDS konfigurieren

Wenn Sie RTDS zum ersten Mal einsetzen, müssen Sie einige Einstellungen für die Anpassung an Ihr System vornehmen. Wählen Sie dafür aus dem Menü "Project" den Menüpunkt "Settings" (s.u.).
Nehmen Sie folgende Einstellungen vor:

- Unter "General" die Nummer der SORCUS-Karte, mit der Sie arbeiten wollen
- Unter "Connection" die Schnittstellen von PC und SORCUS-Karte, zwischen denen die Nullmodem-Verbindung zum Debuggen besteht
- Unter "Connection" das Verzeichnis, in dem sich die Programmdateien des SORCUS-Remote-Kernels befinden.
- Unter "Tools" beim Punkt "Tools Directory" das "bin"-Verzeichnis Ihres Borland-Compilers. In diesem Verzeichnis müssen sich die Dateien "bcc.exe", "tlink.exe" und "make.exe" befinden. Falls Sie den Turbo-Assembler verwenden wollen, muß sich auch die Datei "tasm.exe" in diesem Verzeichnis befinden.
- Unter "Directories" beim Punkt "Includes" das "Include"-Verzeichnis Ihres Borland-Compilers sowie das Verzeichnis, in dem sich die SORCUS-Header-Dateien befinden.

- Unter "Directories" beim Punkt "Libraries" das "Lib"-Verzeichnis Ihres Borland-Compilers sowie das Verzeichnis, in dem sich die SORCUS-Bibliotheken befinden.

8.1.3. Das Project-Menü

- **New Project**:
  Wählen Sie diesen Eintrag, um ein neues Projekt anzulegen.
- **Open Project**:
  Wählen Sie diesen Eintrag, um ein bestehendes Projekt zu öffnen.
- **Close Project:**
  Wählen Sie diesen Eintrag, um das aktuelle Projekt zu schließen. Falls Sie Änderungen am Projekt durchgeführt haben, werden Sie gefragt, ob Sie diese speichern möchten.
- **Insert File Into Project:**
  Wählen Sie diesen Eintrag, um dem aktuellen Projekt eine bestehende Datei hinzuzufügen. Falls Sie dem Projekt mehrere Dateien hinzufügen wollen, empfiehlt es sich, diesen Menüeintrag nicht zu verwenden sondern stattdessen die Dateien per drag-and-drop aus dem Windows-Explorer in den Project-Tree dem Projekt hinzuzufügen.
- **Build:**
  Wenn Sie diesen Eintrag anwählen, wird automatisch eine Make-Datei (<Name des Projekts>.mak) erzeugt und das Make-Programm mit dieser Datei aufgerufen. Falls Sie seit dem letzten Aufruf dieses Eintrags Änderungen an den Quelldateien Ihres Projekts durchgeführt haben, werden vom Make-Programm automatisch Compiler und Linker aufgerufen, um eine aktuelle Version des zum Projekt gehörenden Echtzeitprogramms zu erzeugen.
- **Install:**
  Beim Anwählen dieses Menüpunkts wird die zum Projekt gehörende Installationsdatei ausgeführt. Diese Datei wird nicht automatisch erzeugt, sie muß per Hand angelegt werden!

- **Settings:**
  Beim Anwählen dieses Eintrags wird ein Dialog angezeigt, in dem Einstellungen für das aktuelle Projekt durchgeführt werden können. Die Struktur der Projekteinstellungen entspricht der der Borland-C++-IDE. Schlagen Sie zur Information über die Einstellungen bitte in der Borland-Hilfe nach. Beim Anwählen des Buttons "Set Default" werden diese Einstellungen als Default-Einstellungen für alle neuen Projekte dieses Typs (d.h. dieser Art SORCUS-Karte) verwendet.

8.1.4. Neues Projekt anlegen

Wählen Sie aus dem "Project"-Menü (s.o.) den Eintrag "New Project", eine Dialogbox "New Project" erscheint.
Wählen Sie Im Feld "Type" den Typ des Projekts aus. Da im Moment nur die Entwicklung von Echtzeitprogrammen mit dem Borland C++-Compiler unterstützt wird, beschränkt sich die Auswahl auf den Typ der SORCUS-Karte, auf der Ihr Programm laufen soll.
Geben Sie im Feld "Location" einen Pfad an, in dem Ihr Projekt angelegt werden soll. Wenn Sie hier nichts eintragen, wird Ihr Projekt im Verzeichnis "c:\" angelegt.
Drücken Sie jetzt den Button "Create". Nun wird in dem unter "Location" angegebenen Pfad ein Verzeichnis mit dem Namen Ihres Projekts angelegt. In diesem Verzeichnis wird eine Datei <*Name des Projekts*>.rtp angelegt, in der die Projekteinstellungen gespeichert werden. Sollte bereits ein Verzeichnis mit dem Namen des Projekts vorhanden sein, so wird dieses nicht gelöscht.
Standardmäßig sind bereits die notwendigen SORCUS-Libraries, der Startup-Code sowie eine Installationsdatei namens <*Name des Projekts*>.ins im Projektverzeichnis in dem Projekt enthalten. Die Installationsdatei selbst wird zu dem Zeitpunkt allerdings noch nicht erzeugt. Ist in dem Projektverzeichnis bereits eine Installationsdatei <*Name des Projekts*>.ins vorhanden, so wird diese auch nicht gelöscht oder verändert.
Quellcodedateien werden beim Anlegen eines neuen Projekts nicht erzeugt, Sie müssen diese also per Hand anlegen und dem Projekt hinzufügen.

8.1.5. Debugger starten

Für das Debuggen von Echtzeitprogrammen mit RTDS gilt prinzipiell das gleiche wie für das Debuggen mit dem Turbo-Debugger von Borland (siehe Kapitel 8.2.).
So erfolgt auch bei der Arbeit mit RTDS die Kommunikation zwischen dem Debug-Kernel auf der Karte und dem Debugger über ein serielles Nullmodem-Kabel.
Da RTDS im Gegensatz zum Borland-Turbo-Debugger speziell für SORCUS-Karten entwickelt wurde, kann allerdings das Installieren des Debug-Kernels und des zu debuggenden Programms ohne Zuhilfenahme externer Programme direkt aus RTDS erfolgen. Der Debug-Kernel wird dabei von RTDS automatisch installiert und die Parameter korrekt eingestellt. Die Installation des Echtzeitprogramms, das debugged werden soll, kann auch aus RTDS erfolgen, die dafür benötigte Installationsdatei muß allerdings per Hand erstellt werden.
Um das zum aktuellen Projekt gehörende Echtzeitprogramm zu debuggen, muß aus dem Menü "Debug" (s.u.) der Eintrag "Start Debugging" angewählt werden. Daraufhin wird als erstes überprüft, ob der Debug-Kernel auf der Karte vorhanden ist und ggf. installiert. Dann wird versucht, über den Debug-Kernel das Echtzeitprogramm auf der Karte zu finden. Wird es gefunden, so wird geprüft, ob die Version des Programms auf der Karte älter ist als die auf der Festplatte. Ist das Programm auf der

Karte älter oder gar nicht vorhanden, so wird die zum Projekt gehörige Installationsdatei ausgeführt und nochmal geprüft, ob das Programm auf der Karte vorhanden und aktuell ist.
Ist jetzt das Programm in einer aktuellen Version auf der Karte vorhanden, so wird der Debugger gestartet. Dabei werden Register, Call Stack und zu beobachtende Variablen eingeblendet (falls nicht im Menü "Views" ausgeschaltet) und der Instruction Pointer (erkennbar am gelben Pfeil am linken Rand) auf den Anfang der main()-Funktion gesetzt.

8.1.6. Das Debug-Menü

- **Start Debugging:**
  Beim Anwählen dieses Eintrags wird der Debugger gestartet. Falls sie nicht im View-Menü deaktiviert wurden, erscheinen dann die Variablen- und Registeransicht sowie die Aufrufliste. Außderdem wird der Instruction Pointer auf den Anfang der main()-Funktion positioniert.
- **End Debugging:**
  Beim Anwählen dieses Eintrags wird der Debugger beendet. Waren Variablen- und Registeransicht oder Aufrufliste dargestellt, so werden sie geschlossen.
- **Break:**
  Mit diesem Eintrag sollte das Programm auf der Karte unterbrochen werden. Sollte, weil die momentane Version des Debug-Kernels dieses noch nicht unterstützt.
- **Go:**
  Nach Anwählen dieses Eintrags läuft das zu debuggende Programm weiter, solange bis es auf einen Breakpoint trifft oder das Programmende (d.h. der Rücksprung aus der main()-Funktion) erreicht ist.
- **Trace Into:**
  Nach Anwählen dieses Eintrags wird im zu debuggenden Programm eine Quellcodezeile bzw. bei aktivem Disassembly-Fenster ein Maschinenbefehl ausgeführt. Falls hierbei eine Funktion aufgerufen wird, so wird in der ersten Zeile der aufgerufenen Funktion angehalten.
- **Step Over:**
  Wie beim Trace Into wird bei diesem Eintrag eine Quellcodezeile bzw. ein Ma schinenbefehl ausgeführt. Der Unterschied besteht allerdings darin, daß bei Funktionsaufrufen erst dann angehalten wird, wenn zur aufrufenden Funktion zurückgekehrt wurde.

- **Step Out:**
  Wird dieser Eintrag gewählt, so läuft das zu debuggende so lange, bis aus der Funktion, in der sich der Instruction Pointer momentan befindet, zurückgekehrt wurde.
- **Run To Cursor:**
  Nach Anwählen dieses Menüpunkts läuft das Programm so lange, bis der die Quellcodezeile bzw. die Adresse, an der momentan der Cursor steht, erreicht wird.
- **Set IP To Cursor**:
  Mit diesem Eintrag wird der Instruction Pointer auf die momentane Cursorposition gesetzt, d.h. beim nächsten Go, Step Over, Trace Into oder Step Out-Befehl wird die Ausführung an der momentanen Cursorposition fortgesetzt.
- **Reset:**
  Hiermit wird das zu debuggende Programm wieder in den Anfangszustand versetzt, d.h. die Register werden restauriert und der Instruction Pointer wird auf den Anfang der main-Funktion() gesetzt.
- **Toggle Breakpoint:**
  Ist an der momentanen Cursorposition im Editor- oder Disassembly-Fenster kein Breakpoint gesetzt, so wird einer gesetzt. Ist ein Breakpoint gesetzt, so wird er entfernt.
- **Watch Expression:**
  Nach Anwählen dieses Menüpunkts wird ein Dialog dargestellt, in dem ein Ausdruck eingetragen werden kann. Dieser Ausdruck wird dann in die Variablenansicht aufgenommen.
- **Watch Selection:**
  Ist im aktuellen Editorfenster ein Ausdruck markiert (invertiert dargestellt), so wird er in die Variablenansicht aufgenommen.
- **Access I/O-Ports:**
  Ein Dialog wird dargestellt, in dem lesend und schreibend auf I/O-Adressen zugegriffen werden kann.

8.1.7. Unterstützte Compiler

In der gegenwärtigen Version von RTDS wird der Borland C++-Compiler der Versionen 4.5, 5.0 und 5.2, jeweils mit den zugehörigen make und tlink-Versionen, sowie der Borland-Turbo-Assembler (getestet nur in den Versionen 3.1 und 4.1), unterstützt.
Die Verwendung von Borland C++ 3.1 ist prinzipiell auch möglich, allerdings ist dabei das make-Tool problematisch: unter Windows NT 4.0 läuft es gar nicht, unter Windows 98 bleibt eine Prozeßleiche im Speicher. Falls Sie also auf Borland C++ 3.1 angewiesen sind, sollten Sie das make-Programm eines neueren Borland C++-Compilers verwenden.

8.2. Remote-Debugging mit dem Turbo-Debugger

8.2.1. Allgemeines

Der Turbo-Debugger ist ein Quelltext-Debugger, der entwickelt wurde, um eine leistungsfähige Testumgebung zur Verfügung zu stellen. Mit ihm läßt sich ein erstelltes Programm Schritt für Schritt austesten, so daß Fehler schnell und bequem eingegrenzt werden können. Der Turbo-Debugger wertet beliebige C-, Pascal- oder Assembler-Ausdrücke aus, hat umfangreiche Breakpoint-Funktionen und unterstützt die Rückwärts-Ausführung eines Programms.

Dieses Kapitel zeigt, wie Sie mit dem Turbo-Debugger auf der MODULAR-4/486 Karte arbeiten, und wie die Hardware vorbereitet werden muß. Änderungen am Turbo-Debugger sind nicht erforderlich.

Zufriedenstellend arbeiten läßt sich mit dem Turbo-Debugger nur unter DOS. In den Windows-Betriebssystemen funktioniert das Programme meistens nur sehr langsam.

8.2.1.1. Voraussetzungen

Folgende Voraussetzungen an Hard- und Software müssen erfüllt werden:

Hardware:

a) Der PC, auf dem der Turbo-Debugger laufen soll, braucht mindestens eine serielle Schnittstelle.

b) Eine MODULAR-4/486 Karte.

c) Ein Nullmodem-Kabel für die serielle Verbindung von PC und MODULAR-4/486.

Von SORCUS getestete Software:

a) OsX-Betriebssystem "ML7-1A.01x" oder "ML8-3A.03x" (x steht für E=EPROM Betriebssystem oder für R=ladbares Betriebssystem).

b) Turbo-Debugger Version 2.0 oder größer.

c) Turbo-Pascal 7.0. oder Borland C++ Versionen 3.1. bis 4.5, falls auf Quelltextebene gearbeitet werden soll. Turbo-Assembler (TASM) Version 3.1. für Assemblerprogrammierer.

d) SNW Version 3.K. oder größer bzw., falls die Echtzeitprogramme aus einem Anwenderprogramm heraus installiert werden, ML7BIB oder ML8BIB Version 2.A.000.

Das Betriebssystem und SNW können Sie, falls Sie ältere Versionen besitzen, bei SORCUS Computer von der SORCUS-Homepage herunterladen. Die anderen Produkte erhalten Sie gegebenenfalls bei Ihrem Softwarelieferanten.

8.2.1.2. Einschränkungen

Da auf der Karte ein völlig anderes Betriebssystem als auf einem PC läuft, werden einige Funktionen des Debuggers nicht unterstützt. Die Einschränkungen sind nicht drastisch und beziehen sich eher auf den Komfort als auf die Funktionalität.

- Sie können innerhalb des Turbo-Debuggers keine Tasks installieren. Die Tasks müssen vorher mit SNW oder ML7BIB bzw. ML8BIB installiert worden sein!
- Sie können keine Dateioperationen ausführen. So führt zum Beispiel die Funktion "Open" im Turbo-Debugger zu einem Fehler, wenn sich das angegebene Programm nicht schon auf der Karte befindet, und man versucht, dieses Programm mit dem Turbo-Debugger auf die Karte zu "laden". Um ein Echtzeitprogramm unter einer Task auf die MODULAR-4/486 Karte zu laden, müssen Sie entweder SNW oder die Bibliothek ML7BIB bzw. ML8BIB verwenden. Sie können jedoch mit der Funktion "Open" alle Programme anwählen, die sich auf der Karte befinden.
- Die Funktion "Get Info" ist nur teilweise implementiert.
- Falls Sie eine Prozedur vom Typ "Interrupt" tracen, so dürfen Sie den letzten Befehl der Interrupt-Prozedur (z.B. die Zeile "end;") nicht ausführen (!). Die Ausführung des Befehls würde zum Absturz der Karte führen.
- Falls Sie die Hauptprozedur einer DI-Task tracen möchten, so dürfen Sie nur bis zum Senden der EOIs tracen. Die EOIs und der iRET-Befehl (in ml7rt_exit_interrupt bzw. ml8rt_exit_interrupt enthalten) müssen in Echtzeit (z.B. durch Drücken der Taste F9) ausgeführt werden.

8.2.2. Die Hardwareinstallierung zum Remote-Debuggen

Der Turbo-Debugger ermöglicht das sogenannte "Remote-Debugging". Das bedeutet, daß der Turbo-Debugger auf einem anderen Rechner (Hostrechner) läuft als das zu testende Programm, das auf der MODULAR-4/486 (Remote-Rechner) läuft.

Verbunden werden beide Rechnersysteme über serielle Schnittstellen mit einem Null-Modem-Kabel (die Transmit- und Receive-Leitungen müssen gekreuzt sein). Um ein Programm zu debuggen, das auf der MODULAR-4/486 Karte unter dem Multi-Tasking-Betriebssystem läuft, müssen die seriellen Schnittstellen der Karte und des Hostrechners (COM1 oder COM2) verbunden werden. Die MODULAR-4/486 Karte ist der Remote-Rechner. Als Hostrechner kann entweder der PC verwendet werden, in dem die MODULAR-4/486 Karte eingesteckt ist (siehe **Abb. 8-1**), oder ein separater, zweiter PC (siehe **Abb. 8-2**). Sie können natürlich auch Programme auf Karten debuggen, die stand-alone betrieben werden, also in keinen PC eingesteckt sind (siehe **Abb. 8-3**).

In jedem Fall muß eine serielle Verbindung zwischen Hostrechner und MODULAR-4/486 Karte hergestellt werden. **Die parallele PC-Schnittstelle der Karte bleibt dabei voll funktionsfähig.** Das bedeutet, daß mit der Karte ganz normal über die parallele PC-Schnittstelle kommuniziert werden kann, während über die serielle Schnittstelle gleichzeitig ein Echtzeitprogramm getestet wird.

! *Die serielle Verbindung darf erst hergestellt werden, wenn die MODULAR-4/486 Karte im PC steckt.*

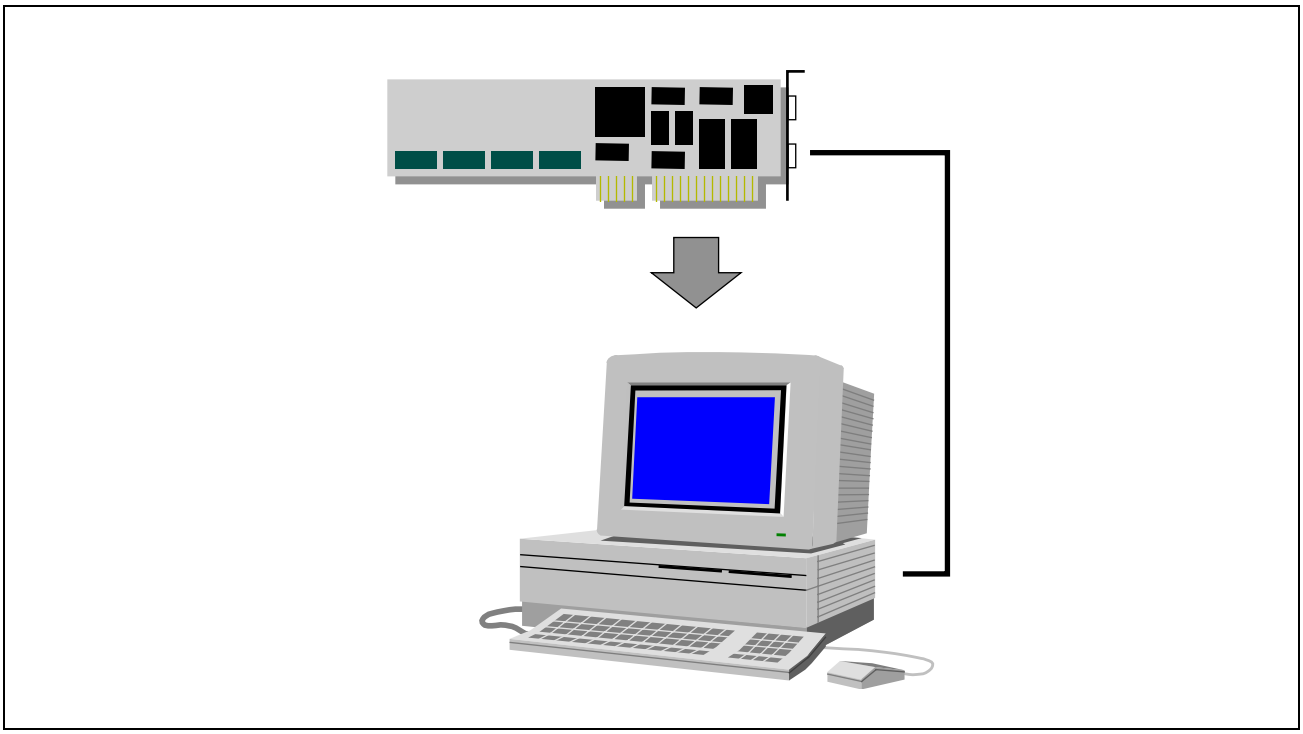


Abb. 8-1: Die Karte steckt in dem PC, auf dem auch der Turbo-Debugger läuft.

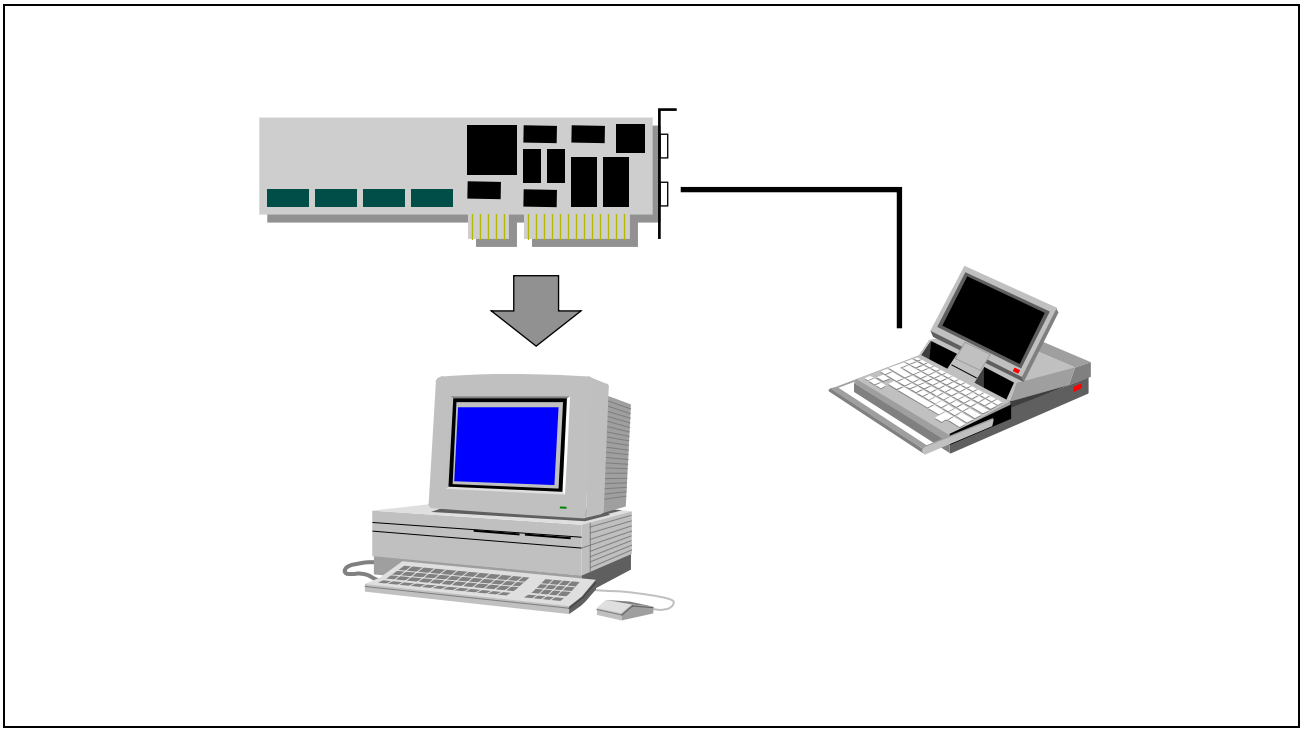


Abb. 8-2: Die Karte steckt in einem PC. Der Turbo-Debugger läuft auf einem zweiten PC und wird seriell mit der "großen" MODULAR-4/486 verbunden.

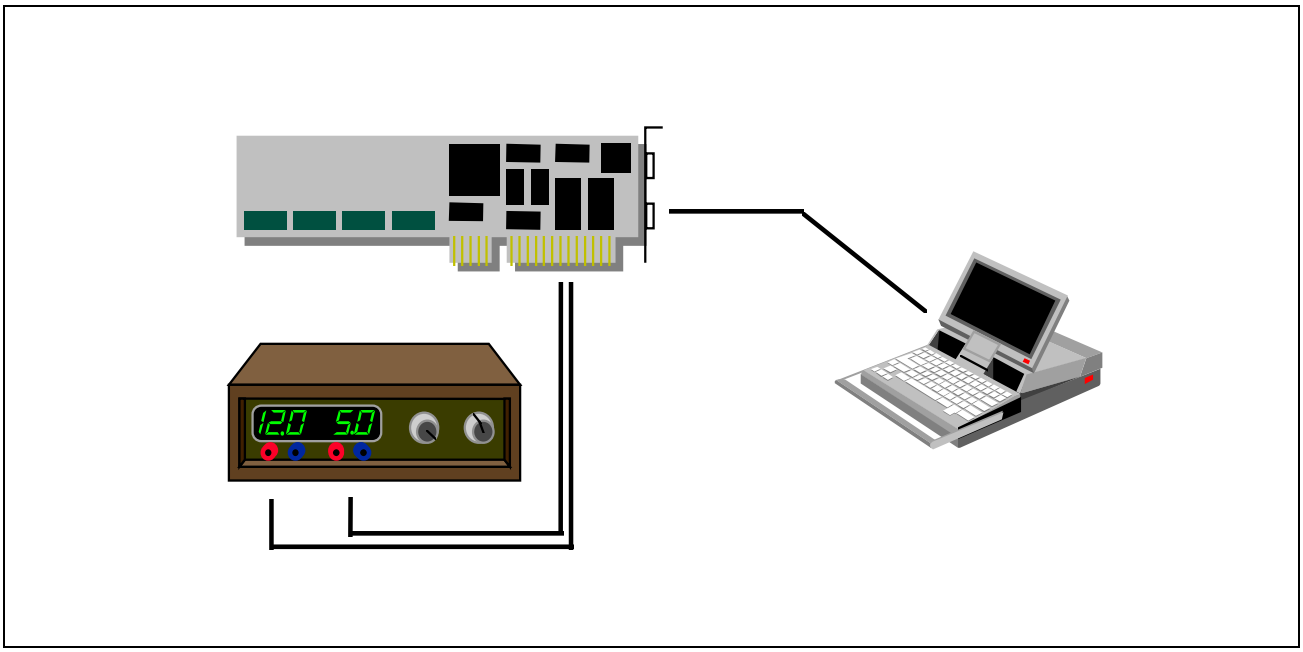


Abb. 8-3: Die Karte arbeitet 'stand-alone' und wird von einem Netzteil mit Spannung versorgt. Der Turbo-Debugger läuft auf einem PC und wird seriell angekoppelt.

Mit der Verbindung zwischen Hostrechner und MODULAR-4/486 Karte ist die Hardwareinstallation zum Remote-Debuggen abgeschlossen.

! *Beachten Sie, daß die D-Sub. Buchsen der seriellen Schnittstellen A und B der "großen" MODULAR-4/486 Karte standardmäßig nicht eingelötet sind. Bevor Sie mit dem Debuggen beginnen können muß die entsprechende Buchse eingelötet werden. Die "kleine" MODULAR-4/486 Karte verfügt über keine D-Sub. Buchsen - der Anschluß der serielle Schnittstellen erfolgt über den Pfostenstecker St1 bzw. St2.*

Weitere Informationen finden Sie im Kapitel "Remote-Debuggen" im Handbuch des Turbo-Debuggers (Borland).

8.2.3. Die Installierung der Software

Beim Remote-Debuggen eines Echtzeitprogramms läuft auf der MODULAR-4/486 Karte ein sogenannter "Remote-Kernel", während der Turbo-Debugger auf dem Hostrechner läuft.

! *Der Remote-Kernel ist ein Multi-Tasking-Echtzeitprogramm. Andere Tasks, die gleichzeitig auf der Karte laufen, werden nicht beeinträchtigt.*

Der Remote-Kernel läuft ab der Betriebssystemversion "ML7-1A.01x" und "ML8-3A.03x". Außerdem ist eine SNW-Version ab 3.K. notwendig. Sollten Sie ältere Versionen besitzen, so setzen Sie sich mit SORCUS wegen des Updates in Verbindung bzw. laden die aktuelle Version von der SORCUS-Homepage.

8.2.3.1. Installierung des Remote-Kernels auf der MODULAR-4

Vorbereitung der zu debuggenden Programme

Um ein Echtzeitprogramm auf der MODULAR-4/486 Karte debuggen zu können, muß dieses Echtzeitprogramm auf der Karte als Task installiert worden sein. Die Installierung kann entweder über "SNW" oder aus einem PC-Anwenderprogramm mit Hilfe der Bibliothek ML7BIB bzw. ML8BIB erfolgen. Wollen Sie das Echtzeitprogramm auf der Quelltextebene debuggen, so muß dieses Echtzeitprogramm vorher mit Debug-Informationen versehen worden sein (Compilereinstellungen siehe Seite 8-16).

Installierung des Kernels mit SNW

Der Remote-Kernel wird am einfachsten mit SNW installiert. Zuerst werden die zu debuggenden Programme auf die Karte geladen (nicht starten!). Anschließend wird der Remote-Kernel unter dem Menüpunkt "Tools/Turbo-Debugger" konfiguriert und gestartet. Überprüfen Sie die Einstellungen (Version des Turbo-Debuggers, Baudrate ...) und geben Sie den Pfad zu den Programmdateien des Remote-Kernels an (standardmäßig: \SORCUS\ML8\RT\ML8REMOT).

Installierung des Kernels mit einer Installationsdatei (*.INS)

Statt mit dem Dialogfenster in SNW können Sie den Kernel auch in einer Installationsdatei zusammen mit den zu testenden Programmen laden und installieren. Das ist der schnellere Weg, da Sie sich dann um den Turbo-Debugger nicht jedesmal gesondert kümmern müssen. Allerdings ist die Einstellung und Änderung der Parameter mit dem M8PAR-Befehl (Turbo-Debuggerversion, Baudrate ...) unübersichtlicher. Folgende Echtzeitprogramme müssen auf der Karte installiert werden:

ML8REMOT.EXE und ML8REMCO.EXE ("große" MODULAR-4/486)

ML7REMOT.EXE und ML7REMCO.EXE ("kleine" MODULAR-4/486)

Ergänzen Sie Ihre Installationsdatei um die folgenden Zeilen, oder benutzen Sie eine separate Installationsdatei, die die folgenden Zeilen enthält:

```
; Remote-Debugger-Kernel und Kommunikationsinterface
; installieren (Task-FF).

MxINST[1] MLxREMOT.EXE FF00 00FF 01 000000 980
MxINST MLxREMCO.EXE FF01 00FE 01 000000 980

; Gegebenenfalls können hier Anweisungen stehen, um
; Parameter des Remote-Kernels zu ändern.
; z.B.: MxPAR FF 06 04 um die Turbo-Debuggerversion 4.0 einzustellen

; Durch Aufruf der Prozedur 2 wird der Remote-Kernel gestartet.
MxPROC FF 02
```

Die Tasknummern der Programme (feh und ffh) können bei Bedarf geändert werden, falls die Tasknummern schon verwendet werden.

[1] x=7 bei "kleiner" MODULAR-4/486, x=8 bei "großer" MODULAR-4/486

8.2.3.2. Die Parameter des Remote-Kernels

In der Regel brauchen Sie sich um die Parameter des Remote-Kernels nicht zu kümmern, vor allem dann nicht, wenn Sie den Turbo-Debugger mit SNW unter dem Menüpunkt ‘Tools/Turbo-Debugger’ installieren. Die Parameter enthalten neben dem Status des Programms auch detaillierte Fehlermeldungen. Daneben kann die maximale Stackgröße festgelegt werden, die der Remote-Kernel für das zu debuggende Echtzeitprogramm reserviert. Die fett markierten Einträge sind die Standardeinstellungen.

Nr.	Init	Typ[2]	Erklärung
0	0	Byte (R)	Status des Programms: **0: Programm bereit.** 2: Kernel aktiviert, wartet auf Handshake vom Turbo-Debugger. 3: Fehler aufgetreten. Fehlermeldung in Parameter 1. 4: Programmlauf abgebrochen. 5: Initialisierungsphase. 6: Verbindung mit Turbo-Debugger hergestellt. 7: Turbo-Debugger hat die Verbindung unterbrochen. 8: Das zu debuggende Programm konnte auf der Karte gefunden werden. 9: Das zu debuggende Programm enthält keine Debug-Informa tionen bzw. es ist auf der Karte nicht installiert. 10: Suche zu debuggendes Programm. 11: Falsches bzw. unbekanntes Kommando vom Turbo-Debugger. Das unbekannte Kommando steht in Parameter 1. 12: Fehler im Kommunikationsinterface aufgetreten
1	0	Byte (R)	Fehlermeldung: **0: Kein Fehler aufgetreten.** 6: Falsche MODULAR-4 Betriebssystemversion 7: Kommunikationstask nicht vorhanden
2	**61440**	Word (R/W)	Stackgröße, die der Remote-Kernel für das zu debuggende Echtzeitprogramm reserviert. Es können maximal 65520 Byte für den Stack reserviert werden. Sollte nicht genug Speicher auf der Karte zur Verfügung stehen, so sendet der Kernel die Meldung "Not enough memory" zum Turbo-Debugger.
4	**254**	Word	Tasknummer des Kommunikationsinterfaces ‘ML7REMCO.EXE’

[2] R = Parameter darf nur gelesen werden

R/W = Parameter darf gelesen und geschrieben werden

Nr.	Init	Typ[2]	Erklärung
	(feh)	(R/W)	bzw. ‘ML8REMCO.EXE’
6	2	Byte (R/W)	Version des Turbo-Debuggers (Link-Version) **2: Turbo-Debugger 2.0 bis 3.1** 3: Turbo-Debugger 3.2 4: Turbo-Debugger 4.0 5: Turbo-Debugger 4.6 und 5.0

8.2.3.3. Prozeduren des Remote-Kernels

Der Remote-Kernel (ML7REMOT.EXE bzw. ML8REMOT.EXE) besitzt zwei Prozeduren:

Prozedur 2: Diese Prozedur aktiviert den Kernel und die eingestellten Parameter. Der Status wird auf "2" gesetzt. Der Kernel wartet jetzt auf das Handshake-Signal vom Turbo-Debugger.

Prozedur 3: Diese Prozedur deaktiviert den Kernel und bricht eine Verbindung mit dem Turbo-Debugger ab. Der Status geht auf "4" (Programm abgebrochen).

8.2.3.4. Parameter des Kommunikationsinterfaces

Für die Parameter des Kommunikationsinterfaces gilt das gleiche wie für den Remote Kernel: Wenn Sie den Turbo-Debugger mit SNW (Tools/Turbo-Debugger) installieren, brauchen Sie sich nicht darum zu kümmern.

Nr.	Init	Typ[3]	Erklärung
0	0	Byte (R)	Status des Programms: **0: Programm bereit.** 2: Programm läuft. 3: Fehler aufgetreten. Fehlermeldung in Parameter 1.
1	0	Byte (R)	Fehlermeldung: **0: Kein Fehler aufgetreten**. 1: Falscher Kommunikationskanal eingestellt! 2: Falsche Baudrate eingestellt! 3: SCC läßt sich nicht initialisieren 4: Transmit-Timeout! Verbindung mit Turbo-Debugger zusammengebrochen! 5: Receive-Timeout! Verbindung mit Turbo-Debugger zusammengebrochen! 8: Falscher Modulsteckplatz eingestellt.
2	0	Byte (R/W)	Typ des Kommunikationsmoduls (derzeit wird nur die Basiskarte unterstützt): **0: Basiskarte**
3	0	Byte (R/W)	Steckplatz des Kommunikationsmoduls: 0: Basiskarte
4	0	Byte (R/W)	Kommunikationsschnittstelle der Karte die verwendet werden soll: Typ 0: **0 = Schnittstelle-A der Basiskarte** 1 = Schnittstelle-B der Basiskarte
5	38400	Long (R/W)	Übertragungsrate in Baud: 9600, 19200, **38400**, 115200 (nur mit Quarzfreq. ≥7,3728 MHz)
11	0	Byte (R/W)	Quarzfrequenz für SCC (siehe Prüfbericht der Basiskarte) 0: **4,9152 MHz** 1: 7,3728 MHz

[3] R = Parameter darf nur gelesen werden

R/W = Parameter darf gelesen und geschrieben werden

8.2.4. Arbeiten mit dem Debugger

8.2.4.1. Vorbereitung

a) Compilieren Sie Ihre Programme mit Debug-Informationen.

Turbo-Pascal: In dem Menü "Option/Debugger" muß in der Option "Symbole" der Eintrag "Externer Debugger" eingeschaltet sein. Im Menü "Option/Compiler" sollten im Feld "Debugger" alle drei Einträge (Debug-Informationen, Lokale Symbole und Symbolinformationen) eingeschaltet sein.

Borland C 3.1: In dem Menü "Options/Debugger" muß in der Option "Source-Debugging" der Eintrag "Standalone" eingestellt werden. Außerdem muß im Menü "Options/Compiler/Advanced Code Generation" die Option "Debug Info in OBJs" eingeschaltet werden.

Borland C 4.5: In den Projektoption "Compiler/Debugger" muß "Debug Information in .OBJ-Dateien" angegeben werden und unter der Option "Linker/Allgemein" "Debug-Informationen mit aufnehmen".

Die Beispielprogramme "M8P0301.EXE (Turbo-Pascal) und "M8P0302.EXE" (Borland C) sind beide mit Debug-Informationen compiliert worden.

b) Stellen Sie die serielle Verbindung zwischen Hostrechner und MODULAR-4/486 Karte her. Im folgenden Beispiel wird davon ausgegangen, daß am PC die Schnittstelle COM1 verwendet wird (Default-Einstellungen der Parameter).

c) Installieren Sie alle zu debuggenden Tasks entweder mit SNW oder mit Ihrem PC-Programm. Verwenden Sie für das folgende Beispiel bitte die Installationsdatei "DI_DEB.INS", die Sie auf der mitgelieferten Diskette finden.

d) Rufen Sie in SNW den Menüpunkt "Tools/Turbo-Debugger" auf und stellen Sie die Parameter (Baudrate, Schnittstelle, ...) nach Ihren Bedürfnissen ein. Nach dem Betätigen des Startknopfes ist der Remote-Kernel einsatzbereit, wenn in der Statuszeile "Wartet auf Handshake" erscheint. Schließen Sie dann das Fenster und verlassen SNW.

8.2.4.2. Aufruf des Turbo-Debuggers auf dem Hostrechner

Nachdem alle Vorbereitungen abgeschlossen wurden, kann der Turbo-Debugger auf dem Host-PC aufgerufen werden. Um den Turbo-Debugger zu starten, geben Sie bitte folgende Zeile ein:

```
td -rp1 -rs3 m8p0301.exe
```

Die Option "-rp1" weist den Debugger an, COM1 als Schnittstelle zum Remote-Kernel zu benutzen (diese Angabe ist optional). Möchten Sie COM2 benutzen, so geben Sie bitte die Option "-rp2" ein. Die Option "-rs3" teilt dem Debugger mit, daß über die serielle Schnittstelle mit einer Übertragungsrate von 38400 Baud gearbeitet wird. "m8p0301.exe" ist der Name der Task, die debugged werden soll.

Im folgenden wird anhand des Beispielprogramms M8P0301.EXE (und dem Quelltext M8P0301.PAS) das Debuggen eines Programms auf der MODULAR-4/486 Karte gezeigt.

Falls alles in Ordnung ist, sollte sich der Debugger, wie in Abb. 8-4 dargestellt, mit dem Quellcode des Programmes M8P0301.PAS melden.

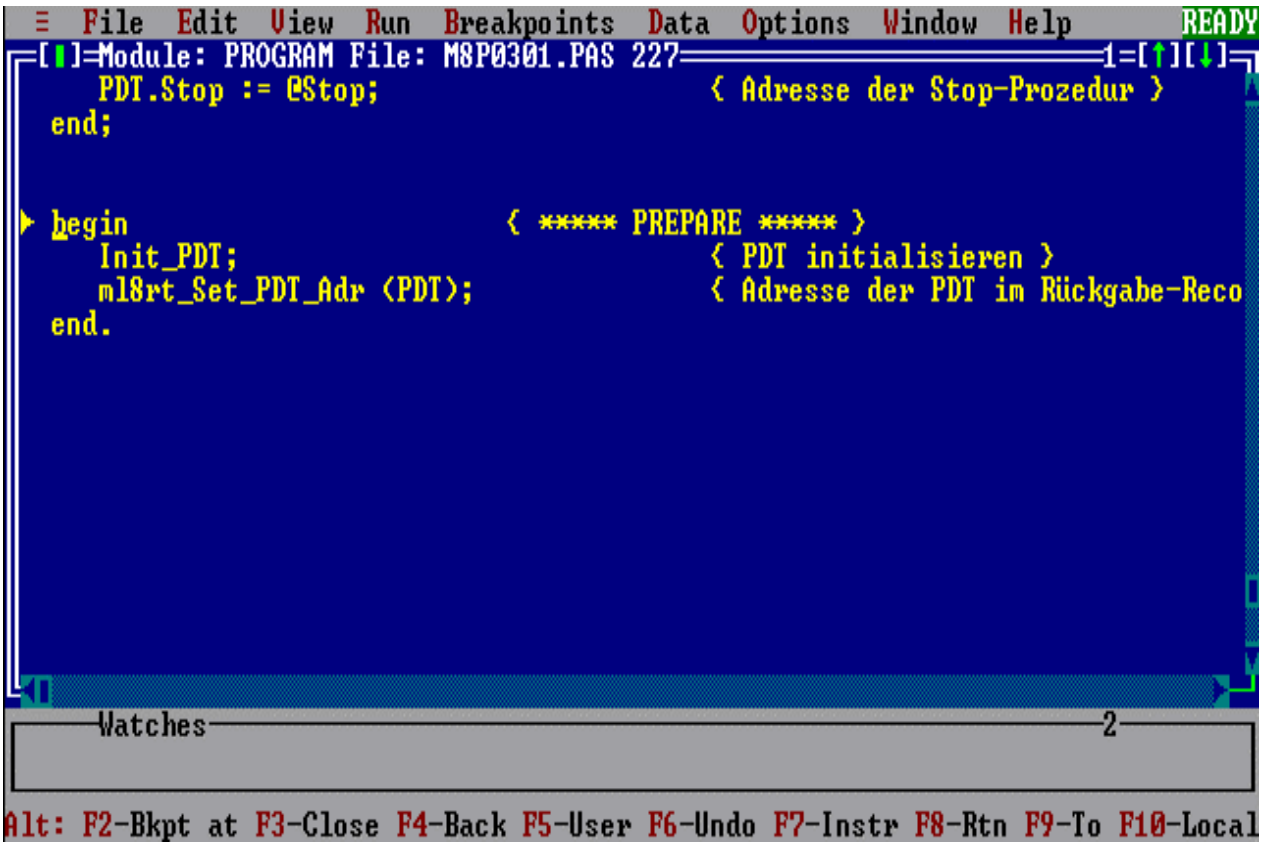


Abb. 8-4: Turbo-Debugger mit zu debuggendem Programm M8P0301.PAS

8.2.4.3. Anwählen einer Prozedur des Echtzeitprogramms

Wie in Kapitel 7 beschrieben, ist der Aufbau eines Echtzeitprogramms anders als der eines PC-Programms. Es besteht aus einer Anzahl verschiedener "Prozeduren" und dem Initialisierungsteil (hier "PREPARE" genannt). Nach dem Aufruf des Turbo-Debuggers zeigt der Programmzeiger (das ist der kleine Pfeil neben dem "begin") auf den Anfang des Prepare-Codes. Dieser Programmteil wurde während der Installierung schon abgearbeitet. Um eine spezielle Prozedur des Programms zu tracen, muß zuerst der Programmzeiger auf diese Prozedur eingestellt werden.

Gehen Sie dabei wie folgt vor:

a) Bewegen Sie den Cursor auf die entsprechende Prozedur im Quellcode bzw. im Assemblercode, falls Sie nicht auf Quelltextebene arbeiten. In diesem Beispiel bringen Sie bitte den Cursor auf die Zeile "PROCEDURE Start;" des Programms M8P0301.PAS, um diese Prozedur zu untersuchen.

b) Drücken Sie jetzt [ALT-V] und anschließend die Taste [C], um das CPU-Fenster zu öffnen.

c) Mit [ALT-F10] öffnen Sie nun das lokale Menü dieses Fensters und wählen anschließend den Menüpunkt [N] für "New cs:ip".

d) Danach können Sie das CPU-Fenster mit [ALT-F3] wieder schließen.

Der Programmzeiger befindet sich jetzt am Anfang der "Start"-Prozedur (siehe Abb. 8-5).

Der Turbo-Debugger bietet eine Möglichkeit, die oben aufgeführte Tastenfolge mit einem **Tastatur-Makro** zu automatisieren, so daß mit dem Aufruf des Makros der Programmzeiger direkt an die Stelle des Cursors gebracht werden kann. Dieses Makro brauchen Sie nicht selbst aufzuzeichnen, da es in der Datei "tdconfig.td" bereits enthalten ist. Kopieren Sie diese Datei in Ihr Entwicklungsverzeichnis, dann wird sie beim Starten des Debuggers automatisch geladen. Das Makro können Sie mit [ALT-P] abrufen, nachdem Sie den Cursor an den Anfang der Prozedur bewegt haben, die Sie untersuchen wollen.

8.2.4.4. Schrittweises Ausführen einer Prozedur

Nachdem Sie nun die Start-Prozedur angewählt haben, können Sie schrittweise durch die Startprozedur "tracen".

```
≡ File  Edit  View  Run  Breakpoints  Data  Options  Window  Help        READY
[■]=Module: PROGRAM File: M8P0301.PAS 128═══════════════════════1=[↑][↓]
  Procedure Start;              { ***** Programm starten ***** }
  VAR
     n                     : LongInt;
     Timer                 : Word;
▶ begin
     ml8rt_entry;
     if (Parameter.LED=EXTERN) OR (Parameter.LED=INTERN) then
     begin                                   { Aus der angegebenen Frequenz den
        n := 2500000 DIV Parameter.Blink_Rate;{ Die Timer werden mit 2,5 MHz g
        Parameter.Soft_Init := (n DIV 60000); { Die Zeiten für den Hardware-Ti
        Parameter.Soft_Cnt := Parameter.Soft_Init;
        Timer := Round (n/(Parameter.Soft_Init+1));
        ml8rt_set_timer (TIMER_A, Timer, INT_MODE); { Timer-A starten }
        Parameter.Status := RUNNING;
        ml8rt_unmask_int (IRQ_TIMER_A);     { Timer-A demaskieren }
     end
     else
        Parameter.Status := LED_ERROR;
Watches═══════════════════════════════════════════════════════════2
Alt: F2-Bkpt at F3-Close F4-Back F5-User F6-Undo F7-Instr F8-Rtn F9-To F10-Local
```

Abb. 8-5: Der Turbo-Debugger nach dem Anwählen der Prozedur ‘Start’

Zum Tracen können Sie z.B. die Taste [F8] benutzen. Dann wird die Zeile, in der sich der Programmzeiger befindet, ausgeführt, der Programmzeiger weitergeschoben und abgewartet. Wenn Sie das Ende der Prozedur erreicht haben, werden Sie die Meldung "Terminated, Exit-Code 0" erhalten. Falls Sie jetzt eine weitere Prozedur tracen möchten, so müssen Sie Ihr Programm durch "CTRL-F2" zurücksetzen und anschließend die zu tracende Prozedur anwählen.

Beachten Sie bitte, daß Sie immer zuerst Ihr Programm mit "CTRL-F2" zurücksetzen müssen, bevor Sie eine neue Prozedur anwählen. Dadurch werden die Programmregister neu gesetzt. Dies gilt insbesondere dann, wenn Sie eine Prozedur nicht bis zum Ende durchlaufen haben und eine andere Prozedur anwählen möchten.

Die Task, die Sie untersuchen, wurde als vollwertiges Programm auf der Karte installiert. Das heißt, daß nach dem Aufruf der Prozedur **ml7rt_unmask_int** bzw. **ml8rt_unmask_int** das Programm auf der Karte aktiviert wurde! Wenn Sie sich jetzt die LED ansehen, so sollte Sie nach der Aktivierung blinken.

```
≡ File Edit View Run Breakpoints Data Options Window Help        STATUS
[■]=Module: PROGRAM File: M8P0301.PAS 128═══════════════════1=[↑][↓]
  Procedure Start;            { ***** Programm starten ***** }
  VAR
     n                    : LongInt;
     Timer                : Word;
  begin
     ml8rt_entry;
     if (Parameter.LED=EXTERN) OR (Parameter.LED=INTERN) then
     begin                                { Aus der angegebenen Frequenz den
                                  er.Blink_Rate;{ Die Timer werden mit 2,5 MHz g
                                  n DIV 60000); { Die Zeiten für den Hardware-Ti
                                  rameter.Soft_Init;
                                  eter.Soft_Init+1));
                                  , Timer, INT_MODE); { Timer-A starten }
                                  ING;
                                  MER_A);     { Timer-A demaskieren }
     end
     else
        Parameter.Status := LED_ERROR;

[■]
   Terminated, exit code 0

      OK        Help

Watches═══════════════════════════════════════════════════2═══
```

Abb. 8-6: Das Ende der Start-Prozedur wurde erreicht.

Sie können nun den Debugger voll einsetzen und relevante Programmteile untersuchen oder andere Teile in Echtzeit ausführen lassen. Es ist möglich, andere Tasks im Hintergrund auf der Karte weiterlaufen zu lassen. Selbstverständlich können Sie Funktionen wie "conditional breakpoints", "backtracing" oder Beobachtung von Variablen frei benutzen.

8.2.5. Tips und Tricks

8.2.5.1. Arbeiten in der integrierten Entwicklungsumgebung

Die integrierten Entwicklungsumgebungen von Turbo-Pascal und Borland C erlauben es, andere Programme wie z.B. SNW oder den Turbo-Debugger aufzurufen, ohne die Entwicklungsumgebung zu verlassen. So ist es beispielsweise möglich, ein Programm zu schreiben, zu compilieren, das Programm mit SNW auf der Karte zu installieren und anschließend mit dem Turbo-Debugger das Programm auf der Karte zu debuggen, ohne die integrierte Entwicklungsumgebung zu verlassen. Diese Möglichkeit kann bei der Programmentwicklung sehr viel Zeit sparen. Nähere Informationen finden Sie in Turbo-Pascal unter dem Menüpunkt "TOOLS" bzw. "OPTION/TOOLS". In Borland C 3.1 finden Sie die entsprechenden Funktionen unter ALT-Leertaste bzw. "OPTIONS/TRANSFER". In Borland C 4.5 und 5.0 heißen die entsprechenden Menüpunkt "Tool" und "Optionen/Tools".

8.2.5.2. Daten von aktivierten Tasks untersuchen

Mit den Inspect- und Watchfunktionen des Turbo-Debuggers können auch Daten von aktivierten Tasks angezeigt werden. Dabei ist zu beachten, daß der Turbo-Debugger die Daten nicht ständig neu liest. Sie werden jedesmal neu ermittelt, wenn ein Programmschritt ausgeführt wird, wenn sie ins 'Watch'-Fenster eingetragen werden, oder wenn ein neues 'Inspect'-Fenster geöffnet wird. Das gilt auch für untergeordnete Inspect-Fenster.

8.2.5.3. Breakpoints

Die im Turbo-Debugger gesetzten Breakpoints werden nur dann aktiviert, wenn das zu debuggende Programm im Zustand **running** (Anzeige rechts oben am Bildschirm) ist. Es genügt also nicht, einen Breakpoint zum Beispiel in eine Interrupt-Service-Routine zu setzen, um dann, wenn der Interrupt auftritt, automatisch dorthin zu gelangen. Statt dessen muß man den Turbo-Debugger mit einem Trick in den Zustand **running** versetzen.

Programmieren Sie an einer Stelle Ihres Programmes, die gerade nicht getestet oder aufgerufen wird, eine Endlosschleife. Wenn Sie keinen unbenutzten Teil haben, müssen Sie dafür eventuell eine neue Prozedur vorsehen. Wenn das Programm compiliert und geladen ist, können Sie nun beliebig viele Breakpoints setzen. Positionieren Sie den Programmzähler anschließend in die Endlosschleife und starten Sie das Programm (z.B. mit Taste [F9]). Der Turbo-Debugger ist nun im Zustand **running** und bleibt das auch, bis einer der gesetzten Breakpoints angesprungen wird.

Sie dürfen natürlich nicht vergessen die Endlosschleife wieder zu entfernen!

9. Echtzeit-Bibliotheken

9.1. Einführung

Nachdem in Kapitel 7 Konzept und Struktur der Echtzeitprogramme erläutert wurde, sollen in den folgenden Abschnitten die Subroutinen vorgestellt werden, die Ihnen die Bibliothek ML7RTBIB bzw. ML8RTBIB zur Verfügung stellt. Verwenden Sie die ML7RTBIB zur Erstellung von Echtzeitprogrammen für die "kleine" MODULAR-4/486 (ML7) bzw. die ML8RTBIB für Programme, die auf der "großen" MODULAR-4/486 Karte (ML8) laufen sollen. Alle Funktionen und Prozeduren der Bibliotheken beginnen mit **ml7rt_...** (ML7) bzw. **ml8rt_...** (ML8). Die Funktionen sind gleichwertig, aus Gründen der Übersichlichkeit werden aber im folgenden nur die Funktionen der Bibliothek ML8RTBIB für die "große" MODULAR-4/486 Karte beschrieben.

Wenn Sie I/O-Devices auf Modulen programmieren möchten, sollten Sie die Modul-Device-Treiber verwenden. Bitte lesen Sie hierzu auch Anhang O.

Bevor Sie dieses Kapitel "durcharbeiten", sollten Sie die vorangegangenen Kapitel sorgfältig gelesen und mit den Beispielprogrammen ein wenig experimentiert haben. Insbesondere das Kapitel 7 "Echtzeitprogrammierung" sollten Sie mit besonderer Aufmerksamkeit gelesen haben. Es enthält wichtige Informationen zur Programmierung und Compilierung.

Die im folgenden dargestellten Funktionen und Prozeduren dienen zur Erstellung von Echtzeitprogrammen für die MODULAR-4/486 Karte! Diese Programme sind nicht auf einem PC verwendbar! **!**

Auf der Distributionsdiskette bzw. der CD zur Echtzeitprogrammierung finden Sie neben den abgedruckten Beispielprogrammen in C und Borland-Pascal auch die Header-Datei für C (**ML7RTBIB.H** bzw. **ML8RTBIB.H**).

Lesen Sie bitte die README Dateien auf den mitgelieferten Disketten bzw. CD. Sie enthalten wichtige Informationen, die nicht mehr rechtzeitig ins Handbuch aufgenommen werden konnten.

9.2. Bibliotheksverwaltung

ml8rt_get_lib_version — Ermittle den Versionscode der Bibliothek

Pascal PROCEDURE ml8rt_get_lib_version (VAR version, date: LONGINT);

C void ml8rt_get_lib_version (ulong* version, ulong* date);

Funktion Die Prozedur liefert den Versions- und den Datecode der verwendeten Echtzeit-Bibliothek.

Parameter *version*: 32-Bit Versionscode[1] der Bibliothek.

date: 32-Bit Datecode[1] der Bibliothek.

ml8rt_get_osx_version — Ermittle den Versionscode des Betriebssystems

Pascal PROCEDURE ml8rt_get_osx_version (VAR version, date: LONGINT);

C void ml8rt_get_osx_version (ulong* version, ulong* date);

Funktion Die Prozedur liefert den Versions- und den Datecode der verwendeten Bibliothek.

Parameter *version*: 32-Bit Versionscode[1] des Betriebssystem.

date: 32-Bit Datecode[1] des Betriebssystem.

---

[1] Bedeutung der Codes siehe Kapitel 9.8.

9.3. Prozeduren und Funktionen

9.3.1. Deklaration

Die globalen Prozeduren einer Task, also diejenigen, die von anderen Tasks, vom Betriebssystem oder vom PC aus aufgerufen werden können, werden grundsätzlich ohne Übergabewerte deklariert:

in C: **void name (void);**

in PASCAL: **PROCEDURE name;**

Einzige Ausnahme sind die Taskfunktionen. Sie haben folgenden vorgeschriebenen Aufbau:

in C: **void far pascal name (ushort task, ushort insize, void *inptr, ushort outsize, void *outptr);**

in PASCAL: **PROCEDURE name (task, insize : Word; VAR inptr; outsize: Word; VAR outptr);**

task: Nummer der eigenen Task.
insize: Anzahl der Daten, die der Funktion übergeben werden (in Byte).
inptr: Zeiger auf die übergebenen Daten.
outsize: Maximale Anzahl der Daten, die zurück erwartet werden (in Byte).
outptr: Zeiger auf den Bereich, in dem die Task ihre Daten zurückgibt.

Alle lokalen Prozeduren und Funktionen dürfen beliebig deklariert werden. **!**

9.3.2. Aufbau

Alle globalen Prozeduren einer Task beginnen mit einem Bibliotheksaufruf, der die Prozessor-Register sichert und für die Prozedur neu setzt. Am Ende jeder Prozedur steht ein Aufruf, der den ursprünglichen Zustand wiederherstellt. Je nachdem, ob es sich um eine globale Prozedur oder Funktion handelt und je nach Tasktyp, werden dabei unterschiedliche Funktionen verwendet.

ml8rt_entry — Beginne globale Prozedur

Pascal PROCEDURE ml8rt_entry;

C void ml8rt_entry (void);

Funktion Diese Prozedur rettet alle Register und setzt das DS-Register auf das Datensegment des Programms. Der Aufruf dieser Prozedur sollte immer als erste Anweisung in einer Prozedur stehen, die vom Betriebssystem aufrufbar sein soll, d.h. in allen Prozeduren, die in die PDT eingetragen wurden. Prozeduren, die nur innerhalb des Programms aufgerufen werden, müssen **ml8rt_entry** nicht aufrufen. In C kann auch alternativ das Makro _ml8rt_entry verwendet werden.

ml8rt_entry_task — Beginne globale Prozedur

Pascal FUNCTION ml8rt_entry_task: WORD;

C ushort ml8rt_entry_task (void);

Funktion Die Funktion arbeitet genau wie **ml8rt_entry**. In manchen Fällen, kann es erforderlich sein, in einer globalen Prozedur die eigene Tasknummer zu kennen. Das gilt vor allem dann, wenn Parameter- oder Datenbereich nicht im Programm enthalten sind und mit Betriebssystemaufrufen (z.B. ml8rt_read_par_xxx) zugegriffen werden muß. Für diesen Fall liefert diese Funktion die Tasknummer zurück. Für DI-Tasks ist diese Funktion nicht geeignet.

! Hinweis Lokale Variablen einer Taskprozedur dürfen erst nach dem Aufruf von ml8rt_entry bzw. ml8rt_entry_task initialisiert werden.

ml8rt_exit
ml8rt_exit_interrupt — Beende globale Prozedur

Pascal PROCEDURE ml8rt_exit;

PROCEDURE ml8rt_exit_interrupt;

C void ml8rt_exit (void);

void ml8rt_exit_interrupt (void);

Funktion Diese Prozeduren sind das Gegenstück zu **ml8rt_entry** und **ml8rt_entry_task**. Sie restaurieren die mit **ml8rt_entry** gesicherten Register und beenden die Prozedur. Der Aufruf dieser Prozeduren muß immer als letzter Aufruf in einer Prozedur stehen. **ml8rt_exit_interrupt** muß in Hauptprozeduren (Prozedur #0) von DI-Tasks verwendet werden! Sie beendet die Prozedur mit einem IRET (Return of Interrupt). In C kann auch alternativ das Makro _ml8rt_exit bzw. _ml8rt_exit_interrupt verwendet werden.

ml8rt_entry_function
ml8rt_entry_function_phys — Beginne globale Funktion

Pascal PROCEDURE ml8rt_entry_function;

PROCEDURE ml8rt_entry_function_phys;

C void ml8rt_entry_function (void);

void ml8rt_entry_function_phys (void);

Funktion Diese Prozeduren arbeitet genauso wie **ml8rt_entry,** jedoch sind sie für eine Task-*Funktion* bestimmt. Die Task-Funktion liefert im Gegensatz zur Task-Prozedur ein Ergebnis. **ml8rt_entry_function** sorgt dafür, daß die im Funktionskopf deklarierten Variablen gesetzt werden. In C kann auch alternativ das Makro _ml8rt_entry_function verwendet werden. **ml8rt_entry_function** verwendet die Adressen der Übergabedaten (in_data_var, out_data_var) als physikalische Adressen.

Hinweis Lokale Variablen einer Taskfunktion dürfen erst nach dem Aufruf von ml8rt_entry_function bzw. ml8rt_entry_function_phys initialisiert werden. !

ml8rt_exit_function
ml8rt_exit_function_phys — Beende globale Funktion

Pascal
PROCEDURE ml8rt_exit_function (leftsize, outsize, error: WORD);

PROCEDURE ml8rt_exit_function_phys (leftsize, outsize, error: WORD);

C
void ml8rt_exit_function (ushort leftsize, ushort outsize, ushort error);
void ml8rt_exit_function_phys (ushort leftsize, ushort outsize, ushort error);

Funktion
Diese Prozeduren beenden eine Funktion einer Task. Sie arbeiten genauso wie **ml8rt_exit**. Es wird jedoch zusätzlich eine Antwort zur aufrufenden Task zurückgesendet.

Parameter
leftsize: Anzahl Bytes, die nicht verwendet werden konnte.

outsize: Größe der Antwort (in Bytes).

error: Fehlermeldung für die aufrufende Task (0 = kein Fehler). Wenn die Funktion per Makrobefehl vom PC aus aufgerufen wird, wird nur das höherwertige Byte als Fehlermeldung zurückgegeben, das niederwertige Byte muß **e0h** sein. Wenn *error* also **xxe0h** ist, wird **xxh** als Fehlerbyte an den PC übergeben.

9.3.3. Aufrufe von Prozeduren und Funktionen

ml8rt_call_func — Rufe eine Funktion einer Task auf

Pascal
PROCEDURE ml8rt_call_func (task, funcnr, VAR outsize: WORD;
VAR outdata_var; maxinsize: WORD; VAR insize: WORD;
VAR indata_var; VAR error: WORD);

C
void ml8rt_call_func (ushort task, ushort funcnr,
ushort far *outsize, void far *outdata_var; ushort maxinsize,
ushort far *insize, void far *indata_var, ushort far *error);

Funktion
Mit dieser Prozedur kann eine beliebige Funktion einer auf der MODULAR-4 installierten Task aufgerufen werden. Ein möglicher Fehler wird nicht abgefangen (kein Aufruf der Fehlerbehandlung). Siehe auch Kapitel 9.3.1.

Parameter *task*: Nummer der Task, die die gewünschte Funktion enthält.

funcnr: Nummer der Funktion.

outsize: Anzahl der Bytes, die an die Funktion übergeben werden. Nach dem Aufruf der Funktion enthält *outsize* die Anzahl der nicht verwendeten Bytes.

outdata_var: Zeiger auf einen Variablenbereich, in dem die Daten stehen, die an die Funktion übergeben werden.

maxinsize: Maximale Anzahl an Bytes, die die aufgerufene Funktion zurückliefern darf bzw. soll.

insize: Zeiger auf eine Variable, in die die tatsächlich zurückgelieferte Anzahl an Bytes eingetragen wird.

indata_var: Zeiger auf eine Variable, in die die Rückgabedaten der Funktion eingetragen werden, kann gleich *outdata_var* sein.

error: Zeiger auf eine Variable, in die ein Fehlerstatus eingetragen wird. Ist dieser Wert = 0, ist kein Fehler bei der Ausführung der Funktion aufgetreten. Bei einem Wert größer Null handelt es sich entweder um eine Betriebssystemmeldung (siehe Anhang F) oder um eine spezielle Meldung der aufgerufenen Funktion.

ml8rt_call_proc — Rufe eine Prozedur einer Task auf

Pascal PROCEDURE ml8rt_call_proc (task, procnr : WORD);

C void ml8rt_call_proc (ushort task, ushort procnr);

Funktion Diese Prozedur ist ein Sonderfall der Prozedur **ml8rt_call_func**: Es werden keine Daten an die aufgerufene Prozedur übergeben oder von ihr übernommen.

Parameter *task*: Nummer der Task, die die gewünschte Prozedur enthält.

procnr: Nummer der Prozedur.

ml8rt_get_proc_address — Ermittle Adresse einer Prozedur einer Task

Pascal PROCEDURE ml8rt_get_proc_address (task: WORD; procnr: WORD; VAR adr: POINTER);

C void ml8rt_get_proc_address (ushort task, ushort procnr, void *adr);

Funktion Diese Prozedur ermittelt die Adresse einer globalen Taskprozedur. Mit Hilfe des Zeigers kann die globale Prozedur direkt aufgerufen werden (Aufruf ist schneller als mit ml8rt_call_proc).

Parameter *procnr*: Nummer der gesuchten Prozedur.

adr: Zeiger auf eine Variable, in die die Adresse (Segment: Offset) der Prozedur eingetragen wird.

! Hinweis Taskfunktionen können nicht direkt aufgerufen werden.

9.4. Taskbefehle

Die meisten der in diesem Abschnitt beschriebenen Funktionen enthalten den Parameter *task*. Dieser spezifiziert die Nummer, unter dem das Betriebssystem auf der MODULAR-4 Karte das zugehörige Echtzeitprogramm verwaltet. Er wird in den Parameterbeschreibungen nicht jedesmal aufgeführt.

9.4.1. Taskinformationen abfragen

ml8rt_get_task_status — Prüfe, ob eine Task aktiviert ist

Pascal FUNCTION ml8rt_get_task_status (task: WORD): BYTE;

C byte ml8rt_get_task_status (ushort task);

Funktion Diese Funktion liefert die Anzahl der Aktivierungen einer Task. Ist das Funktionsergebnis = 0, dann ist die Task nicht aktiviert. TI-, DI- und II-Tasks können maximal einmal aktiviert sein.

ml8rt_get_tdt_address — Ermittle Adresse der TDT

Pascal PROCEDURE ml8rt_get_tdt_address (task: WORD;
VAR adr: POINTER);

C void ml8rt_get_tdt_address (ushort task, void *adr);

Funktion Mit dieser Prozedur wird die Adresse der TDT einer Task ermittelt.

Parameter *adr*: Zeiger auf eine Variable, in die die Adresse (Segment: Offset) der TDT eingetragen wird.

ml8rt_get_task_info — Fordere Systeminformationen über eine Task an

Pascal PROCEDURE ml8rt_get_task_info (task: WORD; callnr: WORD;
VAR result: LONGINT);

C void ml8rt_get_task_info (ushort task, ushort callnr, ulong *result);

Funktion Diese Prozedur liefert Systeminformationen über eine Task.

Parameter: *callnr*: Spezifikation der gewünschten Information (s. Anhang H)

result: Zeiger auf eine Variable, in die das Ergebnis eingetragen wird.

ml8rt_get_int_task — Ermittle Tasktyp und -nummer

Pascal PROCEDURE ml8rt_get_int_task (inter: WORD; VAR task: WORD;
VAR typ: BYTE);

C void ml8rt_get_int_task (ushort inter, ushort *task, byte *typ);

Funktion Diese Prozedur ermittelt, welche Tasknummer und welchen Programmtyp die Task hat, die unter einem Interrupt installiert ist.

Parameter *inter*: Interruptnummer.

task: Zeiger auf eine Variable, in die die Nummer der Task eingetragen wird, die unter dem Interrupt *inter* installiert ist. Wenn der Interrupt unbenutzt ist, wird dieser Wert = -1 (ffffh) gesetzt.

typ: Zeiger auf eine Variable, in die der Tasktyp eingetragen wird. Der Wert entspricht einer der Bibliothekskonstanten **_NI_TASK, _II_TASK, _DI_TASK** oder **_TI_TASK.**

ml8rt_get_task_number — Ermittle Task-Nummer

Pascal
FUNCTION ml8rt_get_task_number (prog_nr: WORD;
VAR install_nr: WORD): WORD;

C
ushort ml8rt_get_task_number (ushort prog_nr, ushort *install_nr);

Funktion
Die Funktion liefert die Tasknummer, unter der ein Programm installiert ist.

Parameter
prog_nr: In der PDT eingetragene Nummer des Programms.

install_nr: Wenn ein Programm mehrfach installiert ist, dann kann in diesem Parameter angegeben werden, von welcher Instanz die Tasknummer ermittelt werden soll.

install_nr (vor Aufruf)	**install_nr** (nach Aufruf)	Bedeutung
0	0	Das Programm ist nicht installiert, der Rückgabewert ist ungültig
0	>0	*install_nr* enthält die Anzahl der Installierungen, der Rückgabewert gibt die höchste Tasknummer an, unter der das Programm installiert ist.
1	0	Das Programm ist nicht installiert, der Rückgabewert ist ungültig
1	1	Das Programm ist installiert, der Rückgabewert gibt die niedrigste Tasknummer an, unter der das Programm installiert ist.
2 bis 1024	0	Das Programm ist nicht installiert, der Rückgabewert ist ungültig
2 bis 1024	= *install_nr* vor Aufruf	Der Rückgabewert enthält die Tasknummer der Task, die sich, wenn man die Tasks nach ihren Nummern ordnet, an der Stelle *install_nr* befindet.

install_nr (vor Aufruf)	**install_nr** (nach Aufruf)	Bedeutung
2 bis 1024	< *install_nr* vor Aufruf	Das Programm wurde nur sooft installiert, wie in *install_nr* zurückgegeben wurde. Der Rückgabewert gibt die höchste Tasknummer an, unter der das Programm installiert ist.

9.4.2. Tasks aktivieren und deaktivieren

ml8rt_wakeup_task — Aktiviere eine Task

Pascal PROCEDURE ml8rt_wakeup_task (task: WORD);

C void ml8rt_wakeup_task (ushort task);

Funktion Diese Prozedur aktiviert eine Task. NI-Tasks können auch mehrfach aktiviert werden, wodurch sie gegenüber einfach aktivierten Tasks häufiger aufgerufen werden. Bei II- und DI-Tasks wird der zugehörige Interrupt demaskiert. TI-Tasks haben eine eigene Aktivierungsroutine (**ml8rt_wakeup_ti_task**).

ml8rt_wakeup_ni_task — Aktiviere eine NI-Task

Pascal PROCEDURE ml8rt_wakeup_ni_task (task, prior: WORD);

C void ml8rt_wakeup_ni_task (ushort task, ushort prior);

Funktion Die Prozedur wird zum Aktivieren von NI-Tasks verwendet. Gleichzeitig legt sie die Priorität für den ersten Aufruf der Task fest.

Parameter *prior*: Abhängig von diesem Parameter erfüllt diese Prozedur folgende Aufgaben:

prior	Funktion
1	Die Task wird aktiviert, also nach dem Aufruf der Funktion zyklisch durchlaufen. Diese Funktion entspricht der Prozedur **ml8rt_wakeup_task**.
2	Nach dem Aufruf der Funktion, wird die Task vorgezogen und kommt als nächste NI-Task an die Reihe, sobald die aktuell laufende NI-Task beendet ist. Wenn die Task zuvor nicht aktiviert wurde (*prior* = 1), dann wird sie nur ein einziges mal aufgerufen.
3	Der Aufruf entspricht dem mit *prior* = 2. Es wird zusätzlich geprüft, ob bereits eine NI-Task vorgezogen, aber noch nicht begonnen wurde. Wenn das so ist, wird die Fehlerbehandlungsroutine aufgerufen.

ml8rt_sleep_task — Deaktiviere eine NI-Task

Pascal PROCEDURE ml8rt_sleep_task (task: WORD);

C void ml8rt_sleep_task (ushort task);

Funktion Durch diese Prozedur wird eine Task deaktiviert. Mehrfach aktivierte NI-Tasks müssen auch mehrfach deaktiviert werden. Bei II- und DI-Tasks wird der zugehörige Interrupt maskiert.

ml8rt_wakeup_ti_task — Aktiviere eine TI-Task

Pascal PROCEDURE ml8rt_wakeup_ti_task (task: WORD; priority: BYTE; count, interval, holdoff: LONGINT);

C void ml8rt_wakeup_ti_task (ushort task, byte priority, ulong count, ulong interval, ulong holdoff);

Funktion Mit dieser Funktion wird der Zeitplan einer TI-Task festgelegt und die Task aktiviert. Folgende Parameter bestimmen den Zeitplan:

Parameter *priority*: Priorität der Task: Wertebereich 0 (höchste Priorität) bis 255 (niedrigste Priorität).

count: Gibt an, wie oft die Task aufgerufen werden soll, bevor sie automatisch wieder deaktiviert wird. Wenn *count* = 0, läuft die Task so lange, bis sie deaktiviert wird.

interval: Zeitintervall zwischen zwei Aufrufen der Task, angegeben in Timer-Tics.

holdoff: Zeit (gemessen in Timer-Tics), nach der die Task zum ersten Mal nach ihrer Aktivierung aufgerufen wird.

9.4.3. Zugriff auf die Parameterbereiche von Tasks

Zugriffe auf Variable im Parameterbereich einer Task erfolgen immer unter Angabe ihrer Adresse (relativ zum Anfang des Parameterbereichs, gezählt in Byte). Diese wird bei allen nachfolgenden Bibliotheksfunktionen im Parameter *start* angegeben. Sollen mehrere Parameter mit einem Befehl gelesen oder geschrieben werden, spezifiziert dieser Wert den ersten zu lesenden oder zu schreibenden Parameter.

Wenn auf bestimmte Parameter oft und schnell zugegriffen werden soll, kann die Speicheradresse des Parameters mit **ml8rt_get_par_address** einmal ermittelt werden und über die Adresse (Pointer) zugegriffen werden.

ml8rt_get_par_sema
ml8rt_release_par_sema — Parameterbereichs-Semaphore

Pascal
```
FUNCTION ml8rt_get_par_sema (task: WORD): WORD;
PROCEDURE ml8rt_release_par_sema (task: WORD);
```

C
```
ushort ml8rt_get_par_sema (ushort task);
void ml8rt_release_par_sema (ushort task);
```

Funktion Mit diesen Funktionen wird der Zugriff auf den Parameterbereich einer Task gesteuert. Die Blockzugriffe auf den Parameterbereich einer Task sind durch Interrupts unterbrechbar. Damit also ein konsistenter Zugriff gewährleistet ist, muß ein Programm, das auf einen Parameterbereich zugreifen will, zunächst versuchen mit **ml8rt_get_par_sema** die Semaphore für den Parameterbereich zu bekommen. Gelingt dies, kann das Programm beliebig Daten in den Parameterbereich schreiben oder lesen. Hat sich bereits eine andere Anwendung die Semaphore geholt, liefert die Funktion eine Fehlermeldung und darf anschließend nicht auf den Parameterbereich zugreifen.

! Hinweis Das Betriebssystem prüft bei den Parameterbereichszugriffen selber nicht, ob die Semaphore frei ist. Um konsistente Daten sicherzustellen müssen alle Anwendungen vor einem Zugriff auf den Parameterbereich einer Task zunächst die Semaphore abprüfen!

Mit **ml8rt_release_par_sema** wird die Semaphore wieder frei gegeben.

Parameter *task* Nummer der Task

ml8rt_read_par_byte
ml8rt_read_par_word
ml8rt_read_par_dword Lies Parameter einer Task

Pascal FUNCTION ml8rt_read_par_byte (task, start: WORD): BYTE;

FUNCTION ml8rt_read_par_word (task, start: WORD): WORD;

PROCEDURE ml8rt_read_par_dword (task, start: WORD;
VAR data: LONGINT);

C byte ml8rt_read_par_byte (ushort task, ushort start);

ushort ml8rt_read_par_word (ushort task, ushort start);

void ml8rt_read_par_dword (ushort task, ushort start, ulong *data);

Funktion: Die Funktionen lesen ein, zwei oder vier Byte aus dem Parameterbereich einer Task. Beachten Sie, daß der Wert eines Doppelwortes nicht als Funktionsergebnis sondern in *data* zurückgegeben wird. Die Zugriffe sind in sich konsistent. Daher braucht die Semaphore nicht vorher geholt zu werden.

Parameter: *data*: Zeiger auf eine Variable, die das Ergebnis von **ml8rt_read_par_dword** aufnimmt.

ml8rt_read_par_block Lies Parameterblock

Pascal PROCEDURE ml8rt_read_par_block (task, start, size: WORD; VAR data_var);

C void ml8rt_read_par_block (ushort task, ushort start, ushort size, void *data_var);

Funktion: Mit dieser Prozedur kann ein Block von Parametern einer Task gelesen werden.

Parameter: *size*: Größe des Blocks (in Byte, z. B. sizeof (...))

data_var: Zeiger auf einen Variablenbereich, in den die gelesenen Daten eingetragen werden.

Hinweis Das Lesen eines Blocks ist durch Interrupts unterbrechbar, falls vorher die Interrupts nicht gesperrt wurden, deshalb sollte vor jedem Zugriff mit **ml8rt_get_par_sema** die entsprechende Semaphore geholt werden, um einen konsistenten Zugriff zu gewährleisten. !

ml8rt_write_par_byte
ml8rt_write_par_word
ml8rt_write_par_dword Schreibe Parameter einer Task

Pascal PROCEDURE ml8rt_write_par_byte (task, start: WORD; data: BYTE);

PROCEDURE ml8rt_write_par_word (task, start: WORD; data: WORD);

PROCEDURE ml8rt_write_par_dword (task, start: WORD; data: LONGINT);

C void ml8rt_write_par_byte (ushort task, ushort start, byte data);

void ml8rt_write_par_word (ushort task, ushort start, ushort data);

void ml8rt_write_par_dword (ushort task, ushort start, ulong data);

Funktion: Mit diesen Prozeduren können ein, zwei oder vier Byte in den Parameterbereich einer Task geschrieben werden. Die Zugriffe sind in sich konsistent. Daher braucht die Semaphore nicht vorher geholt zu werden.

Parameter: *data*: Wert, der geschrieben werden soll.

ml8rt_write_par_block — Schreibe Parameterblock

Pascal PROCEDURE ml8rt_write_par_block (task, start, size: WORD;
VAR data_var);

C void ml8rt_write_par_block (ushort task, ushort start, ushort size,
void *data_var);

Funktion: Mit dieser Prozedur kann ein Block von Parametern einer Task gesetzt werden.

Parameter: *size*: Größe des Blocks (in Byte, z. B. sizeof (...))

data_var: Zeiger auf den Variablenbereich, in dem die Daten stehen, die geschrieben werden.

! Hinweis Das Schreiben eines Blocks ist durch Interrupts unterbrechbar, falls vorher die Interrupts nicht gesperrt wurden, deshalb sollte vor jedem Zugriff mit **ml8rt_get_par_sema** die entsprechende Semaphore geholt werden, um einen konsistenten Zugriff zu gewährleisten.

ml8rt_get_par_address — Ermittle Adresse eines Parameters einer Task

Pascal PROCEDURE ml8rt_get_par_address (task, start: WORD;
VAR adr: LONGINT);

C void ml8rt_get_par_address (ushort task, ushort start, ulong *adr);

Funktion Diese Prozedur ermittelt die Adresse eines Parameters.

Parameter *adr*: Zeiger auf eine Variable, in die die physikalische Adresse des Parameters eingetragen wird.

9.4.4. Zugriff auf die Datenbereiche von Tasks

Der Zugriff auf den Datenbereich einer Task geschieht fast immer über den Schreib- bzw. Lesezeiger der Task. Diese Zeiger werden vom Betriebssystem verwaltet und bei jedem Zugriff automatisch um die Anzahl der gelesenen bzw. geschriebenen Bytes inkrementiert. Ausnahme: **ml8rt_get_data_offs** und **ml8rt_set_data_offs**, hier werden keine Zeiger verwendet.

ml8rt_get_data_sema
ml8rt_release_data_sema — Datenbereichs-Semaphore

Pascal
```
FUNCTION ml8rt_get_data_sema (task: WORD): WORD;
PROCEDURE ml8rt_release_data_sema (task: WORD);
```

C
```
ushort ml8rt_get_data_sema (ushort task);
void ml8rt_release_data_sema (ushort task);
```

Funktion: Mit diesen Funktionen wird der Zugriff auf den Datenbereich einer Task gesteuert. Diese Zugriffe sind durch Interrupts unterbrechbar. Damit ein konsistenter Zugriff gewährleistet ist, muß ein Programm, das auf einen Datenbereich zugreifen will, zunächst versuchen, mit **ml8rt_get_data_sema** die Semaphore für den Datenbereich zu bekommen. Gelingt dies, kann das Programm beliebig auf den Datenbereich zugreifen. Hat sich bereits eine andere Anwendung die Semaphore geholt, liefert die Funktion eine Fehlermeldung.

Hinweis: Das Betriebssystem prüft bei den Datenbereichszugriffen selber nicht, ob die Semaphore frei ist. Um konsistente Daten sicherzustellen, müssen alle Anwendungen vor einem Zugriff auf den Datenbereich einer Task zunächst die Semaphore abprüfen! !

Mit **ml8rt_release_data_sema** wird die Semaphore wieder frei gegeben.

Parameter: *task* Nummer der Task

ml8rt_get_data_offs ml8rt_set_data_offs — Datenbereichszugriff

Pascal

PROCEDURE ml8rt_get_data_offs (task: WORD; ulstart: LONGINT; VAR size: WORD; VAR data);

PROCEDURE ml8rt_set_data_offs (task: WORD; ulstart: LONGINT; VAR size: WORD; VAR data);

C

void ml8rt_get_data_offs (ushort task, ulong ulstart, ushort *size, void *data);

void ml8rt_set_data_offs (ushort task, ulong ulstart, ushort *size, void *data);

Funktion

Mit diesen Funktionen wird auf den Datenbereich einer Task zugegriffen. Vor jedem Zugriff sollte mit **ml8rt_get_data_sema** die entsprechende Semaphore geholt werden, um einen konsistenten Zugriff zu gewährleisten.

Parameter

task Nummer der Task

ulstart Offset (in Bytes) bezogen auf den Anfang des Datenbereiches, ab dem die Daten gelesen bzw. geschrieben werden sollen.

size Zeiger auf eine Variable, in der steht wie viele Bytes (maximal) gelesen bzw. geschrieben werden sollen. Nach Beendigung der Funktion ist dort eingetragen, wie viele Bytes tatsächlich gelesen bzw. geschrieben worden sind. Die Größe des Datenbereiches kann aus der TDT mit Hilfe der Funktion **ml8rt_get_task_info** ausgelesen werden.

data Zeiger auf die Daten

ml8rt_reset_r_pointer — Setze Daten-Lesezeiger zurück

Pascal

PROCEDURE ml8rt_reset_r_pointer (task: WORD);

C

void ml8rt_reset_r_pointer (ushort task);

Funktion

Diese Prozedur setzt den Daten-Lesezeiger einer Task an den Anfang ihres Datenbereichs.

ml8rt_move_r_pointer — Verschiebe Daten-Lesezeiger

Pascal PROCEDURE ml8rt_move_r_pointer (task: WORD;
offset: LONGINT);

C void ml8rt_move_r_pointer (ushort task, long offset);

Funktion Diese Prozedur verschiebt den Daten-Lesezeiger der Task.

Parameter *offset*: Anzahl Byte, um die der Lesezeiger verschoben werden soll. Der Wert kann auch negativ sein.

ml8rt_read_data_byte / ml8rt_read_data_word / ml8rt_read_data_dword — Lies Einzelwert aus dem Datenbereich einer Task

Pascal FUNCTION ml8rt_read_data_byte (task: WORD): BYTE;

FUNCTION ml8rt_read_data_word (task: WORD): WORD;

PROCEDURE ml8rt_read_data_dword (task: WORD;
VAR data: LONGINT);

C byte ml8rt_read_data_byte (ushort task);

ushort ml8rt_read_data_word (ushort task);

void ml8rt_read_data_dword (ushort task, ulong *data);

Funktion Die Prozeduren lesen ein, zwei oder vier Bytes aus dem Datenbereich einer Task. Es wird ab dem Lesezeiger der Task gelesen. Danach wird der Lesezeiger der Task um die entsprechende Anzahl Byte erhöht. Beachten Sie, daß der Wert eines Doppelwortes (vier Byte) nicht als Funktionsergebnis, sondern in *data* zurückgegeben wird.

Parameter *data*: Zeiger auf eine Variable, die das Ergebnis von **ml8rt_read_data_dword** aufnimmt.

ml8rt_read_data_block — Lies Datenblock

Pascal PROCEDURE ml8rt_read_data_block (task, size: WORD;
VAR data_var);

C void ml8rt_read_data_block (ushort task, ushort size, void *data_var);

Funktion Diese Prozedur liest einen Datenblock aus dem Datenbereich einer Task. Es wird ab dem Lesezeiger der Task gelesen. Der Lesezeiger der Task wird anschließend um die gelesene Anzahl Bytes erhöht.

Parameter *size*: Größe des Blocks (in Byte, z. B. sizeof (...)).

data_var: Zeiger auf einen Variablenbereich, in den die gelesenen Daten eingetragen werden.

ml8rt_read_data — Lies Datenblock direkt

Pascal PROCEDURE ml8rt_read_data (task: WORD; rel_ofs: LONGINT; size: WORD; VAR data_var);

C void ml8rt_read_data (ushort task, ulong rel_ofs, ushort size, void* data_var);

Funktion Diese Prozedur liest einen Datenblock aus dem Datenbereich einer Task. Vorher wird der Lesezeiger auf den angegebenen Wert positioniert. Der Lesezeiger der Task wird anschließend um die gelesene Anzahl Bytes erhöht.

Parameter *size*: Größe des Blocks (in Byte, z.B. mit sizeof (...)).

rel_ofs: Blockanfangsadresse als Offset zum Anfang des Datenbereiches.

data_var: Zeiger auf einen Variablenbereich, in den die gelesenen Daten eingetragen werden.

! Hinweis Das Lesen eines Blocks ist durch Interrupts unterbrechbar, falls vorher die Interrupts nicht gesperrt wurden, deshalb sollte vor jedem Zugriff mit **ml8rt_get_data_sema** die entsprechende Semaphore geholt werden, um einen konsistenten Zugriff zu gewährleisten.

ml8rt_reset_w_pointer — Setze Daten-Schreibzeiger zurück

Pascal PROCEDURE ml8rt_reset_w_pointer (task: WORD);

C void ml8rt_reset_w_pointer (ushort task);

Funktion Diese Prozedur setzt den Daten-Schreibzeiger einer Task an den Anfang ihres Datenbereichs.

ml8rt_move_w_pointer — Verschiebe Daten-Schreibzeiger

Pascal PROCEDURE ml8rt_move_w_pointer (task: WORD; offset: LONGINT);

C void ml8rt_move_w_pointer (ushort task, long offset);

Funktion Diese Prozedur verschiebt den Daten-Schreibzeiger einer Task.

Parameter *offset*: Anzahl Byte, um die der Schreibzeiger verschoben werden soll. Der Wert kann auch negativ sein.

ml8rt_write_data_byte
ml8rt_write_data_word
ml8rt_write_data_dword — Schreibe Daten in den Datenbereich einer Task

Pascal PROCEDURE ml8rt_write_data_byte (task: WORD; data: BYTE);

PROCEDURE ml8rt_write_data_word (task: WORD; data: WORD);

PROCEDURE ml8rt_write_data_dword (task: WORD; data: LONGINT);

C void ml8rt_write_data_byte (ushort task, byte data);

void ml8rt_write_data_word (ushort task, ushort data);

void ml8rt_write_data_dword (ushort task, ulong data);

Funktion Diese Prozeduren schreiben ein, zwei oder vier Byte ab dem Schreibzeiger einer Task in ihren Datenbereich. Der Schreibzeiger wird danach automatisch um die entsprechende Anzahl von Byte erhöht.

Parameter *data*: Wert, der geschrieben werden soll.

ml8rt_write_data_block — Schreibe Datenblock

Pascal PROCEDURE ml8rt_write_data_block (task, size: WORD; VAR datavar);

C void ml8rt_write_data_block (ushort task, ushort size, void *datavar);

Funktion: Diese Prozedur schreibt einen Block von Daten ab dem Schreibzeiger einer Task in ihren Datenbereich. Der Schreibzeiger der Task wird danach automatisch um die entsprechende Anzahl von Byte erhöht.

Parameter: *size*: Größe des Blocks (in Byte, z. B. sizeof (...))

datavar: Zeiger auf den Variablenbereich, in dem die Daten stehen, die geschrieben werden.

ml8rt_write_data Schreibe Datenblock direkt

Pascal PROCEDURE ml8rt_write_data (task: WORD; rel_ofs: LONGINT; size: WORD; VAR data_var);

C void ml8rt_write_data (ushort task, ulong rel_ofs, ushort size, void* data_var);

Funktion Diese Prozedur schreibt einen Datenblock in den Datenbereich einer Task. Vorher wird der Schreibzeiger auf den angegebenen Wert positioniert. Der Schreibzeiger der Task wird anschließend um die geschriebene Anzahl Bytes erhöht.

Parameter *size*: Größe des Blocks (in Byte, z.B. mit sizeof (...)).

rel_ofs: Blockanfangsadresse als Offset zum Anfang des Datenbereiches.

datavar: Zeiger auf den Variablenbereich, in dem die Daten stehen, die geschrieben werden.

! Hinweis Das Schreiben eines Blocks ist durch Interrupts unterbrechbar, falls vorher die Interrupts nicht gesperrt wurden, deshalb sollte vor jedem Zugriff mit **ml8rt_get_data_sema** die entsprechende Semaphore geholt werden, um einen konsistenten Zugriff zu gewährleisten.

9.5. Ringpuffer

Das Betriebssystem der MODULAR-4/486 stellt ringförmig organisierte Datenpuffer zur Verfügung. Jeder Puffer erhält bei der Erzeugung eine eindeutige Nummer (Handle). Unter Angabe dieser Nummer (wird jeweils im Parameter *buffer* an die Bibliotheksfunktionen übergeben) kann von allen Tasks aus auf die Pufferdaten zugegriffen werden.

Das Betriebssystem übernimmt die gesamte Verwaltung des Ringpuffers. Es stellt sicher, daß ein Puffer in der Zeit, in der eine Task daraus liest bzw. hineinschreibt, von keiner anderer Task gelesen bzw. beschrieben werden kann. In einem solchen Konflikt gibt die Lese- bzw. Schreibprozedur die Fehlermeldung zurück, daß der Puffer im Moment nicht verfügbar ist. Der gewünschte Aufruf muß zu einem späteren Zeitpunkt wiederholt werden.

Das Schreiben von Daten geschieht mit einem vom Betriebssystem verwalteten Schreibzeiger. Er wird nach jedem Schreibbefehl um die Anzahl geschriebener Bytes weitergeschoben. Wenn mehr Daten geschrieben werden sollen, als Platz zur Verfügung steht, wird eine Fehlermeldung zurückgeliefert. Das Lesen von Daten geschieht ebenfalls über einen Zeiger, der automatisch um die Anzahl der gelesenen Bytes verschoben wird.

ml8rt_create_buffer — Erzeuge Ringpuffer

Pascal: FUNCTION ml8rt_create_buffer (task: WORD; strategy, align: BYTE; usage: WORD; VAR size: LONGINT): LONGINT;

C: long ml8rt_create_buffer (ushort task, byte strategy, byte align, ushort usage, ulong *size);

Funktion: Diese Funktion öffnet einen Ringpuffer und gibt die Nummer des neuen Puffers zurück.

Parameter:

task: Nummer der Task, die den Ringpuffer anfordert (nur zu Protokollzwecken).

strategy: Dieser Parameter legt die Speicherreservierungs-Strategie fest:

1 = Der Speicher wird von der Untergrenze des freien Speichers beginnend reserviert. Wenn nicht genügend Speicher zur Verfügung steht, wird der Ringpuffer nicht angelegt und ein Fehler gemeldet.

2 = Der Speicher wird von der Untergrenze des freien Speichers beginnend reserviert. Wenn nicht genügend Speicher zur Verfügung steht, wird so viel wie möglich reserviert und die tatsächliche Größe des Puffers in *size* zurückgeliefert.

3 = Der Speicher wird an der Obergrenze des freien Speichers endend reserviert. Wenn nicht genügend Speicher zur Verfügung steht, wird der Ringpuffer nicht angelegt und ein Fehler gemeldet.

4 = Der Speicher wird an der Obergrenze des freien Speichers endend reserviert. Wenn nicht genügend Speicher zur Verfügung steht, wird so viel wie möglich reserviert und die tatsächliche Größe des Puffers in *size* zurückgeliefert.

align: Ausrichtung des Anfangs des Ringpuffers. Die Ausrichtung wird in 2er Potenzen angegeben: 0 = Byte, 1 = Wort, 2 = Doppelwort, 3 = 8 Byte, ...

usage: Verwendung des Puffers (derzeit immer = 0 setzen).

size: Größe des Puffers in Byte. Nach dem Funktionsaufruf enthält die Variable die tatsächliche Größe des angelegten Puffers.

ml8rt_clear_buffer — Lösche Inhalt eines Ringpuffers

Pascal PROCEDURE ml8rt_clear_buffer (buffer: LONGINT);

C void ml8rt_clear_buffer (long buffer);

Funktion Diese Prozedur löscht alle Daten (bzw. den Inhalt) eines Ringpuffers.

ml8rt_get_buffer_status — Ermittle Status eines Ringpuffers

Pascal PROCEDURE ml8rt_get_buffer_status (buffer: LONGINT; freesize, usedsize: LONGINT);

C void ml8rt_get_buffer_status (long buffer, ulong *freesize, ulong *usedsize);

Funktion: Diese Prozedur ermittelt, wieviel Speicher im Puffer noch frei ist und wieviel derzeit belegt ist.

Parameter: *freesize*: Zeiger auf eine Variable, in die die Funktion die Anzahl der unbenutzten Bytes des Puffers einträgt.

usedsize: Zeiger auf eine Variable, in die die Funktion die Anzahl der benutzten Bytes des Puffers einträgt.

ml8rt_write_buffer_byte
ml8rt_write_buffer_word
ml8rt_write_buffer_dword Schreibe einzelne Daten in Puffer

Pascal FUNCTION ml8rt_write_buffer_byte (buffer: LONGINT; data: BYTE): WORD;

FUNCTION ml8rt_write_buffer_word (buffer: LONGINT; data: INTEGER): WORD;

FUNCTION ml8rt_write_buffer_dword (buffer: LONGINT; data: LONGINT): WORD;

C ushort ml8rt_write_buffer_byte (long buffer, byte data);

ushort ml8rt_write_buffer_word (long buffer, ushort data);

ushort ml8rt_write_buffer_dword (long buffer, ulong data);

Funktion: Diese Funktionen schreiben ein, zwei oder vier Byte in einen Puffer. Wenn das Schreiben erfolgreich war, ist der Rückgabewert = 0, andernfalls enthält er eine Fehlermeldung (siehe Anhang F).

Parameter: *data*: Wert, der geschrieben werden soll.

ml8rt_write_buffer_block Schreibe Datenblock in Puffer

Pascal FUNCTION ml8rt_write_buffer_block (buffer: LONGINT; size: WORD; VAR datavar);

C ushort ml8rt_write_buffer_block (long buffer, ushort size, void *datavar);

Funktion: Mit dieser Funktion wird ein Datenblock in einen Datenpuffer geschrieben. Wenn der freie Speicher im Puffer nicht ausreicht, werden keine Daten geschrieben. Wenn Daten geschrieben werden konnten, ist der Rückgabewert = 0, andernfalls enthält er eine Fehlermeldung.

Parameter: *size*: Größe des zu schreibenden Blocks (maximal 65520 Byte).

data_var: Zeiger auf die zu schreibenden Daten.

ml8rt_write_buffer_max Schreibe Datenblock in Puffer

Pascal
```
FUNCTION ml8rt_write_buffer_max (buffer: LONGINT;
VAR size: WORD; VAR datavar);
```

C
```
ushort ml8rt_write_buffer_max (long buffer, ushort *size,
void *datavar);
```

Funktion Diese Funktion entspricht der Funktion **ml8rt_write_buffer_block**. Allerdings wird, wenn der Puffer nicht ausreicht, um alle Daten zu schreiben, keine Fehlermeldung zurückgegeben, sondern so viele Daten geschrieben, wie Platz ist. In der Variablen *size* steht anschließend, wie viele Bytes nicht geschrieben worden sind, sie ist also Null, wenn der ganze Block übertragen werden konnte.

Parameter siehe **ml8rt_write_buffer_block.**

ml8rt_read_buffer_byte
ml8rt_read_buffer_word
ml8rt_read_buffer_dword Lies einzelne Daten aus Puffer

Pascal
```
FUNCTION ml8rt_read_buffer_byte (buffer: LONGINT;
VAR data_var: BYTE): WORD;

FUNCTION ml8rt_read_buffer_word (buffer: LONGINT;
VAR data_var: INTEGER): WORD;

FUNCTION ml8rt_read_buffer_dword (buffer: LONGINT;
VAR data_var: LONGINT): WORD;
```

C
```
ushort ml8rt_read_buffer_byte (long buffer, byte *data_var);

ushort ml8rt_read_buffer_word (long buffer, ushort *data_var);

ushort ml8rt_read_buffer_dword (long buffer, ulong *data_var);
```

Funktion: Diese Funktionen lesen ein, zwei oder vier Byte aus einem Puffer. Wenn das Lesen erfolgreich war, wird eine Null zurückgeliefert. Die gelesenen Daten sind gültig und werden im Puffer gelöscht. Der freigewordene Platz steht wieder für das Schreiben von Daten zur Verfügung. Wenn nicht genügend Zeichen im Puffer sind, werden keine Daten gelesen und der Rückgabewert enthält eine Fehlermeldung (siehe Anhang F).

Parameter: *data_var*: Zeiger auf eine Variable, in die die gelesenen Daten eingetragen werden.

ml8rt_read_buffer_block — Lies Datenblock aus Puffer

Pascal
```
FUNCTION ml8rt_read_buffer_block (buffer: LONGINT;
size: WORD; VAR data_var): WORD;
```

C
```
ushort ml8rt_read_buffer_block (long buffer, ushort size,
void *data_var);
```

Funktion: Mit dieser Funktion wird ein Datenblock aus einem Datenpuffer gelesen. Wenn nicht so viele Zeichen im Puffer sind wie angefordert, werden keine Daten gelesen. Die Bereiche des Ringpuffers, die ausgelesen wurden, werden als frei markiert und stehen wieder für das Schreiben von Daten zur Verfügung. Wenn Daten gelesen werden konnten, ist der Rückgabewert = 0, andernfalls enthält er eine Fehlermeldung (siehe Anhang F).

Parameter *size*: Größe des zu lesenden Blocks (maximal 65520 Byte).

data_var: Zeiger auf einen Variablenbereich, in den die gelesenen Daten eingetragen werden.

ml8rt_read_buffer_max — Lies Datenblock aus Puffer

Pascal
```
FUNCTION ml8rt_read_buffer_max (buffer: LONGINT;
VAR size: WORD; VAR data_var): WORD;
```

C
```
ushort ml8rt_read_buffer_max (long buffer, ushort *size,
void *data_var);
```

Funktion Diese Funktion entspricht der Funktion **ml8rt_read_buffer_block**. Allerdings wird, wenn nicht genügend Zeichen im Puffer sind, keine Fehlermeldung zurückgegeben, sondern so viele Daten gelesen, wie im Puffer sind. Wenn Daten gelesen werden konnten, ist der Rückgabewert Null, andernfalls wird eine Fehlermeldung zurückgegeben. In der Variablen *size* steht anschließend wie viele Byte nicht gelesen werden konnten.

Parameter siehe **ml8rt_read_buffer_block**.

ml8rt_view_buffer_block
ml8rt_view_buffer_max Kopiere Daten aus Puffer

Pascal

```
FUNCTION ml8rt_view_buffer_block (buffer: LONGINT;
size: WORD; VAR data_var): WORD;

FUNCTION ml8rt_view_buffer_max (buffer: LONGINT;
VAR size: WORD; VAR data_var): WORD;
```

C

```
ushort ml8rt_view_buffer_block (long buffer, ushort size,
void *data_var);

ushort ml8rt_view_buffer_max (long buffer, ushort *size,
void *data_var);
```

Funktion Diese Funktionen arbeiten genauso wie **ml8rt_read_buffer_block** und **ml8rt_read_buffer_max** mit dem Unterschied, daß die gelesenen Daten im Puffer nicht gelöscht sondern kopiert werden.

9.6. Funktionen zur Hardwarekontrolle

9.6.1. Interrupt-Controller

Alle Funktionen in diesem Abschnitt beziehen sich jeweils auf einen Interrupt (übergeben in der Variablen *hrdirq).* Dort ist die Interruptnummer des gewünschten Interrupts anzugeben. Dafür sind in ML8RTBIB.H folgende Konstanten definiert:

Bezeichnung	Wert	Bedeutung
NMI	02h	Nicht maskierbarer Interrupt
IRQ_RBF	78h	RBF-Interrupt von PC-Schnittstelle
IRQ_F	79h	Interrupt-Leitung F (SP-Bus)
bzw. IRQ_TEMP		bzw. Interrupt von Temperaturüberwachung[1]
IRQ_SLAVE	7ah	Interrupt vom Slave-Controller
IRQ_A	7bh	Interrupt-Leitung A (SP-Bus)
IRQ_B	7ch	Interrupt-Leitung B (SP-Bus)
IRQ_C	7dh	Interrupt-Leitung C (SP-Bus)
IRQ_D	7eh	Interrupt-Leitung D (SP-Bus)
IRQ_E bzw.	7fh	Interrupt-Leitung E (SP-Bus)
IRQ_FAN		bzw. Interrupt von Lüfterüberwachung[1]
IRQ_SCC	90h	Interrupt vom SCC (ser. Schnittstellen A und B)
IRQ_TIMER_A	91h	Interrupt von Timer-A
IRQ_TIMER_B	92h	Interrupt von Timer-B
IRQ_TIMER_C	93h	Interrupt von Timer-C
IRQ_RTC	94h	Interrupt von der Echtzeituhr (Timer-D)
IRQ_EXT	95h	Interrupt vom Stecker St1 (IRQ-G)
IRQ_COM_B	96h	Interrupt Ri der ser. Schnittstelle-B (IRQ-H)
IRQ_TBE	97h	TBE-Interrupt von der PC-Schnittstelle

ml8rt_unmask_int — Gib Hardware-Interrupt frei

Pascal PROCEDURE ml8rt_unmask_int (hrdirq: BYTE);

C void ml8rt_unmask_int (byte hrdirq);

Funktion Diese Prozedur gibt einen Interrupt frei.

[1] Nur bei "kleiner" MODULAR-4/486 Karte verfügbar.

ml8rt_mask_int — Sperre Hardware-Interrupt

Pascal PROCEDURE ml8rt_mask_int (hrdirq: BYTE);

C void ml8rt_mask_int (byte hrdirq);

Funktion Diese Prozedur sperrt (maskiert) einen Interrupt.

ml8rt_set_int_edge — Stelle Interrupt-Flanke ein

Pascal PROCEDURE ml8rt_set_int_edge (hrdirq: BYTE; edge: BYTE);

C void ml8rt_set_int_edge (byte hrdirq, byte edge);

Funktion Diese Prozedur stellt die aktive Flanke eines Interrupts ein.

Parameter *edge*: Spezifiziert, ob der Interrupt bei steigender (POS_EDGE) oder fallender (NEG_EDGE) Flanke ausgelöst werden soll (IRQ-A bis IRQ-H).

ml8rt_end_of_int — Beende Interrupt-Prozedur

Pascal PROCEDURE ml8rt_end_of_int (hrdirq: BYTE);

C void ml8rt_end_of_int (byte hrdirq);

Funktion Diese Prozedur sendet für den Interrupt ein "End of Interrupt" (EOI) zum Master- und zum Slave-Interrupt-Controller.

ml8rt_clear_int — Lösche Pending-Interrupt

Pascal PROCEDURE ml8rt_clear_int (hrdirq: BYTE);

C void ml8rt_clear_int (byte hrdirq);

Funktion Diese Prozedur löscht den Interrupt im Interrupt-Controller. Sie wird in der Regel vor **ml8rt_unmask_int** verwendet, um sicherzustellen, daß ein vorher aufgetretener Interrupt nicht zur Ausführung kommt.

ml8rt_disable_interruptible ml8rt_enable_interruptible — Programmabschnitt gegen Unterbrechung schützen

Pascal PROCEDURE ml8rt_disable_interruptible;

PROCEDURE ml8rt_enable_interruptible;

C void ml8rt_disable_interruptible (void);

void ml8rt_enable_interruptible (void);

Funktion Diese Prozeduren maskieren bzw. demaskieren alle Interrupts (außer NMI) auf den MODULAR-4/486 Karten. Durch Maskieren der Interrupts lassen sich Programmteile gegen Unterbrechung schützen. Anschließend muß die Prozedur **ml8rt_enable_interruptible** aufgerufen werden, um die Interrupts wieder freizugeben. Beide Routinen müssen innerhalb der selben Prozedur stehen.

Wenn in einer Prozedur **ml8rt_disable_interruptible** verwendet wird, muß in jedem Fall vor dem Verlassen der Prozedur **ml8rt_enable_interruptible** aufgerufen werden. Dies gilt auch, wenn z.B. bei einem Fehler eine Prozedur per exit o.ä. vorzeitig verlassen wird!
In C können alternativ auch die Makros _ml8rt_disable_interruptible und _ml8rt_enable_interruptible verwendet werden.

9.6.2. Speicherverwaltung

ml8rt_allocate_ram Reserviere Speicher auf der MODULAR-4

Pascal PROCEDURE ml8rt_allocate_ram (task, usage: WORD;
strategy, align: BYTE; VAR size: LONGINT; VAR adr: LONGINT);

C void ml8rt_allocate_ram (ushort task, ushort usage, byte strategy,
byte align, ulong *size, ulong *adr);

Funktion Diese Prozedur reserviert Speicherplatz auf der MODULAR-4 Karte.

Parameter *task*: Task, für die Speicher reserviert werden soll. Die Angabe dient Protokollzwecken auf der MODULAR-4 Karte.

usage: Reserviert, immer = 0 setzen.

strategy: Reservierungsstrategie. Folgende Werte sind zulässig:

1 = Es wird die in *size* angegebene Speichermenge (in Byte), von <u>unten</u> im freien Speicher beginnend, reserviert. Reicht der Speicher nicht, dann soll nichts reserviert und ein Fehler ausgelöst werden. In diesem Fall wird in *size* eine Null zurückgegeben.

2 = wie 1, jedoch soll, falls die in *size* angegebene Speichermenge (in Byte) nicht zur Verfügung steht, so viel wie möglich reserviert werden. Die tatsächlich reservierte Speichermenge wird in *size* zurückgegeben.

3 = wie 1, jedoch soll der Speicher von <u>oben</u> im freien Speicher beginnend reserviert werden.

4 = wie 2, jedoch soll der Speicher von <u>oben</u> im freien Speicher beginnend reserviert werden.

align: Ausrichtung des Anfangs des zu reservierenden Speicherbereichs in Potenzen zur Basis 2: 0 = Byte, 1 = Wort, 2 = Doppelwort, 3 = 8 Byte, ...

size: Größe des zu reservierenden Speicherbereichs (in Byte). Der Rückgabewert von *size* enthält die *tatsächlich* reservierte Speichergröße.

adr: Zeiger auf eine Variable, in die die physikalische Anfangsadresse des reservierten Speichers eingetragen wird. Der Wert darf daher nicht as C oder Pascal Pointer verwendet werden. Wenn *size* = 0 ist, dann ist die Adresse nicht gültig.

ml8rt_get_free_ram_size — Ermittle freien Speicher

Pascal
PROCEDURE ml8rt_get_free_ram_size (align: BYTE;
VAR size: LONGINT);

C
void ml8rt_get_free_ram_size (byte align, long *size);

Funktion: Diese Prozedur ermittelt die Größe des freien Speichers auf der MODULAR-4 Karte.

Parameter:

align: Gewünschte Ausrichtung des Speichers in Potenzen zur Basis 2: 0 = Byte, 1 = Wort, 2 = Doppelwort, 3 = 8 Byte, ...

size: Zeiger auf eine Variable, in die die Größe des freien Speichers (in Byte) eingetragen wird.

ml8rt_read_ram — Lies aus RAM

Pascal
PROCEDURE ml8rt_read_ram (size: WORD; source: LONGINT;
VAR dest);

C
void ml8rt_read_ram (ushort size, ulong source, void* dest);

Funktion
Diese Prozedur liest einen RAM-Bereich und kopiert ihn in einen Zielbereich. Der zu lesende Bereich kann auch oberhalb der 1 MB-Grenze liegen.

Parameter

size: Größe des Blocks (in Byte, max. 64K - 256)

source: Quelladresse (physikalisch)

dest: Zeigt auf den Zielbereich (Seg:Offs)

ml8rt_write_ram — Schreibe in RAM

Pascal: PROCEDURE ml8rt_write_ram (size: WORD; dest: LONGINT; VAR source);

C: void ml8rt_write_ram (ushort size, ulong dest, void* source);

Funktion: Diese Prozedur schreibt einen RAM-Bereich. Der zu beschreibende Bereich kann auch oberhalb der 1 MB-Grenze liegen.

Parameter:
- *size*: Größe des Blocks (in Byte, max. 64K - 256)
- *dest*: Zeigt auf den Zielbereich (physikalisch)
- *source*: Quelladresse (Seg:Offs)

ml8rt_copy_ram — Kopiere Speicher

Pascal: PROCEDURE ml8rt_copy_ram (size: WORD; VAR source; VAR dest);

C: void ml8rt_copy_ram (ushort size, void* source, void* dest);

Funktion: Diese Prozedur kopiert einen Speicherbereich.

Parameter:
- *size*: Größe des Blocks (in Byte, max. 64K - 256)
- *source*: Quelladresse (Seg:Offs)
- *dest*: Zeigt auf den Zielbereich (Seg:Offs)

ml8rt_get_flash_info — Flash-ROM-Informationen

Pascal: FUNCTION ml8rt_get_flash_info (slot, chip, mode, option: BYTE): WORD;

C: ushort ml8rt_get_flash_info (byte slot, byte chip, byte mode, byte option);

Funktion: Diese Funktion liefert Informationen über den Flash-ROM-Speicher auf einer MODULAR-4 Karte.

Parameter:
- *slot*: Steckplatz des Moduls, auf dessen Flash zugegriffen werden soll (0 = Basiskarte)

chip: Angabe über welches IC Informationen eingeholt werden sollen. Dieser Parameter muß für die "große" MODULAR-4/486 = 17 und für die "kleine" MODULAR-4/486 = 6 gesetzt werden.

Die Statusinformation wird als Rückgabewert der Funktion gemäß derfolgenden Tabelle geliefert:

mode	*option*	**Bedeutung des Rückgabewertes**
FLASH_GET_CUSTOMER_-ID	0	Hersteller-ID des Flash-EPROM
FLASH_GET_DEVICE_ID	0	Kennung des EPROM-Typs
FLASH_GET_SIZE_INFO	0	Information über die Größe des Flash-EPROMs mit folgender Bedeutung: Aus L = Low-Byte des Rückgabewertes H = High-Byte des Rückgabewertes Folgt EPROM-Größe = 2L Byte Sektor-Größe = 2H Byte
FLASH_GET_ ORGANIZATION	0	Organisation des Flash-EPROM: x0h = Bit x1h = Byte x2h = Word 0xh = Kein Sektor geschützt lxh = Bottom 2xh = Top
FLASH_CHECK_ WRITE_PROTECTED	Sektor	Gibt an, ob der angegebene Sektor schreibgeschützt ist oder nicht: 1 = Sektor ist schreibgeschützt 0 = Sektor ist nicht schreibgeschützt
FLASH_GET_ ERASE_STATUS	Sektor	Gibt den Status nach einem Löschvorgang an: 0 = Löschvorgang fehlerfrei beendet 1 = Löschvorgang noch nicht beendet 2 = Fehler beim Löschen aufgetreten
FLASH_GET_SOFT_STATE	0	Soft-State (Betriebsart) 0 = Standard 1 = Für Löschvorgang vorbereitet 2 = Löschvorgang aktiv 3 = Programmiervorgang aktiv

ml8rt_set_flash_mode Flash-Status setzen

Pascal PROCEDURE ml8rt_set_flash_mode (slot, chip, mode, option: BYTE);

C void ml8rt_set_flash_mode (byte slot, byte chip, byte mode, byte option);

Funktion Diese Funktion setzt den Status des Flash-ROM-Speicher auf der MODULAR-4/486.

Parameter *slot*: Steckplatz des Moduls, auf dessen Flash zugegriffen werden soll (0 = Basiskarte)

chip: Angabe, in welches IC Informationen geschrieben werden sollen. Dieser Parameter muß für die "große" MODULAR-4/486 = 17 und für die "kleine" MODULAR-4/486 = 6 gesetzt werden.

Die auszuführende Aktion wird durch die Parameter *mode* und *option* gesteuert:

mode	*option*	Aktion
FLASH_ERASE_ENABLE	0	Aktivieren den Löschmodus
FLASH_ERASE_DISABLE	0	Beenden des Löschmodus
FLASH_PROGRAM_ENABLE	0	Aktivieren des Programmiermodus
FLASH_PROGRAM_DISABLE	0	Beenden des Programmiermodus
FLASH_COPY_EPROM	Segment	Kopieren des EPROM ins Flash-EPROM

ml8rt_erase_flash_sector Flash-Sektoren löschen

Pascal FUNCTION ml8rt_erase_flash_sector (slot, chip: BYTE; first, last: LONGINT): WORD;

C ushort ml8rt_erase_flash_sector (byte slot, byte chip, ulong first, ulong last);

Funktion Diese Funktion löscht einen Bereich im Flash-ROM-Speicher auf der MODULAR-4/486 Karte. Es ist zu beachten, daß das Flash in Sektoren organisiert ist. Sektoren können nur als Ganzes gelöscht werden. Liegt die Anfangsadresse nicht am Anfang eines Sektors, so wird auch der Bereich zwischen Sektoranfang und Startadresse gelöscht.

Parameter

slot Steckplatz des Moduls, auf dessen Flash zugegriffen werden soll (0 = Basiskarte)

chip Angabe, in welchem IC gelöscht werden soll. Dieser Parameter muß für die "große" MODULAR-4/486 = 17 und für die "kleine" MODULAR-4/486 = 6 gesetzt werden.

first Angabe der Adresse, ab der gelöscht werden soll

last Angabe der Adresse, bis zu der gelöscht werden soll

ml8rt_program_flash
ml8rt_read_flash Flash-Zugriff

Pascal PROCEDURE ml8rt_program_flash (slot, chip: BYTE; start: LONGINT; size: WORD; VAR data);

PROCEDURE ml8rt_read_flash (slot, chip: BYTE; start: LONGINT; size: WORD; VAR data);

C void ml8rt_program_flash (byte slot, byte chip, ulong start, ushort size, void *data);

void ml8rt_read_flash (byte slot, byte chip, ulong start, ushort size, void *data);

Funktion Mit diesen Funktionen können Daten in das Flash-ROM auf der MODULAR-4/486 Karte geschrieben bzw. ausgelesen werden. Bevor Daten in das Flash geschrieben werden können, müssen die entsprechenden Sektoren mit **ml8rt_erase_flash_sector** gelöscht werden.

Parameter

slot Steckplatz des Moduls, auf dessen Flash zugegriffen werden soll (0 = Basiskarte)

chip Angabe, in welchem IC geschrieben/gelesen werden soll. Dieser Parameter muß für die "große" MODULAR-4/486 = 17 und für die "kleine" MODULAR-4/486 = 6 gesetzt werden.

start Relative Adresse (bezogen auf den Anfang des Flash), ab der Daten zu lesen bzw. zu schreiben sind

size Anzahl Bytes, die gelesen bzw. geschrieben werden sollen

data Zeiger auf eine Variable, in die die gelesenen Daten eingetragen werden, bzw. in der die zu schreibenden Daten stehen.

ml8rt_cache_control Kontrolliere den Cache

Pascal PROCEDURE ml8rt_cache_control (mode: BYTE);

C void ml8rt_cache_control (byte mode);

Funktion: Die Prozedur dient zum Ein- und Ausschalten des Prozessor-Caches auf der MODULAR-4 Karte.

Parameter: *mode*:
0 = Cache ausschalten und ungültig machen
1 oder 3 = Cache einschalten
4 = Cache-Status melden

9.6.3. Timer-Kontrolle

Für die Auswahl der Timer stehen die Konstanten TIMER_A, TIMER_B, TIMER_C in ML7RTBIB.H bzw. ML8RTBIB.H zur Verfügung. Alle Timer sind 16 Bit breit und werden mit jedem Takt um eins dekrementiert.

ml8rt_set_timer — Setze und starte einen Timer

Pascal: PROCEDURE ml8rt_set_timer (counter: BYTE; count: WORD; mode: BYTE);

C: void ml8rt_set_timer (byte counter, ushort count, byte mode);

Funktion: Diese Prozedur setzt einen Timer und startet ihn.

Parameter:

counter: Gibt an, welcher Timer gesetzt werden soll.

count: Initialisierungswert des Timers. Für eine Quarzfrequenz von 2,5 MHz beträgt die Auflösung 400 ns und die maximale Laufzeit 65535 * 400 ns = 26,2 ms. Für einen 10 MHz-Quarz sind dies 100 ns Auflösung und 6,5 ms maximale Laufzeit. Bei "kleinen" MODULAR-4/486 Karten kann zusätzlich 1 MHz als Quarzfreqeunz eingestellt werden. Die Auflösung beträgt dann 1 µs (max. Laufzeit = ca. 65 ms). Beachten Sie bitte die Worte 3 und 25 des EEPROMs der Basiskarte (s. Anhang L). In Parameter 150 des Betriebssystems (s. Anhang K) steht der tatsächliche Timer-Eingangstakt.

mode: Betriebsart des Timers. Derzeit ist dieser Parameter immer mit der Konstanten **INT_MODE** zu beschreiben**.**

ml8rt_get_timer — Lies Zählerstand eines Timers

Pascal: PROCEDURE ml8rt_get_timer (counter: BYTE; VAR count: WORD; VAR status: BYTE);

C: void ml8rt_get_timer (byte counter, ushort *count, byte *status);

Funktion: Diese Prozedur liest den aktuellen Zählerstand eines Timers.

Parameter *counter*: Gibt an, welcher Timer gelesen werden soll.

count: Zeiger auf eine Variable, in die der aktuelle Timerstand eingetragen wird.

status: Zeiger auf eine Variable, in die der Status des Timer-Bausteins eingetragen wird. Die Bits haben folgende Bedeutung:

Bit	Bedeutung
0	Zählermode: 0 = binär, 1 = dezimal
1 bis 3	Betriebsart
4, 5	immer = 0
6	Null-Count
7	Zustand des Output-Pins

Einzelheiten finden Sie im Datenblatt des Timer-Bausteins 8254 von Intel.

ml8rt_convert_timer_data Berechne einen Timerwert

Pascal FUNCTION ml8rt_convert_timer_data (time: LONGINT): WORD;

C ushort ml8rt_convert_timer_data (ulong time);

Funktion Diese Subroutine wandelt eine Zeit in den passenden Timerwert zum Programmieren eines Timers um. Da diese Routine die tatsächliche Frequenz des Timerquarzes berücksichtigt, können Sie ohne weiteren Rechenaufwand feste Zeitwerte realisieren. Das Ergebnis der Funktion kann direkt der Funktion **ml8rt_set_timer** übergeben werden. Im Fall eines ungültigen Parameters (Wertebereich s. **ml8rt_set_timer**) wird die Fehlerbehandlungsprozedur aufgerufen.

Parameter *time*: Zeit (in Mikrosekunden).

9.6.4. EEPROM-Zugriffe

ml8rt_read_eeprom_copy
ml8rt_read_eeprom_direct — Lies EEPROM-Wort

Pascal
FUNCTION ml8rt_read_eeprom_copy (modul, reladr: BYTE): WORD;

FUNCTION ml8rt_read_eeprom_direct(modul, reladr: BYTE): WORD;

C
ushort ml8rt_read_eeprom_copy (byte modul, byte reladr);

ushort ml8rt_read_eeprom_direct (byte modul, byte reladr);

Funktion
Diese Funktionen lesen ein Wort direkt aus einem EEPROM bzw. aus dessen im RAM angelegter Kopie. Die Kopie wird beim Aktivieren eines Betriebssystems automatisch erzeugt. Direkte Zugriffe sind langsam und in einer Echtzeitumgebung nicht zu empfehlen.

Parameter
modul: Welches EEPROM gelesen werden soll:
- 0: EEPROM der MODULAR-4 Basiskarte
- 1 bis 10: EEPROM des Moduls auf dem entsprechenden Steckplatz.

reladr: relative Adresse des zu lesenden Wortes (0 - 31).

ml8rt_write_eeprom_copy
ml8rt_write_eeprom_direct — Schreibe EEPROM-Wort

Pascal
PROCEDURE ml8rt_write_eeprom_copy (modul, reladr: BYTE; data: WORD);

PROCEDURE ml8rt_write_eeprom_direct (modul, reladr: BYTE; data: WORD);

C
void ml8rt_write_eeprom_copy (byte modul, byte reladr, ushort data);

void ml8rt_write_eeprom_direct (byte modul, byte reladr, ushort data);

Funktion:
Diese Funktionen schreiben ein Wort direkt in ein EEPROM bzw. in dessen im RAM angelegte Kopie. In die Kopie geschriebene Daten gehen beim Abschalten der Karte verloren.

Parameter: *modul, reladr*: siehe **ml8rt_read_eeprom_direct /copy**.

data: Daten, die geschrieben werden sollen.

9.6.5. Zugriffe auf die Echtzeituhr

ml8rt_get_rtc_status — Lies Status der Uhr

Pascal FUNCTION ml8rt_get_rtc_status : BYTE;

C byte ml8rt_get_rtc_status (void);

Funktion Das Funktionsergebnis gibt den Status der Uhr an. Das Rückgabebyte hat folgende Bedeutung:

Bit	Bedeutung
0	s.S.6-42
1	0 = Uhr läuft, 1 = Uhr gestoppt
2	1 = 24 Stundenanzeige, 0 = 12 Stundenanzeige
3	0 = Interruptausgang nicht maskiert, 1 = Interruptausgang maskiert
4,5	Bit-5 Bit-4 Frequenz am Impulsausgang: 0 0 64 Hz (1/15,625 ms) 0 1 1 Hz (1/1 sec) 1 0 0,0166 Hz (1/1 min) 1 1 0,000277 Hz (1/1 h)
6	Zustand des Impulsausgangs (invertiert)
7	ist immer = 0

ml8rt_set_rtc_mode Stelle Betriebsart der Uhr ein

Pascal PROCEDURE ml8rt_set_rtc_mode (mode: BYTE);

C void ml8rt_set_rtc_mode (byte mode);

Funktion Die Prozedur setzt die Betriebsart der Uhr. Indem der Interruptausgang demaskiert wird (Bit 3 = 0) und eine Task z.B. als DI-Task unter dem Interrupt IRQ_RTC (94h) installiert wird, erhält man einen weiteren Timer mit den in der Tabelle aufgeführten Zeiten.

Parameter *mode*: Die Bits haben folgende Bedeutung:

Bit	Bedeutung
0	Reset des Subsekundenzählers: Um den Subsekundenzähler zurückzusetzen, muß die Funktion zweimal aufgerufen werden; Beim ersten Aufruf muß dieses Bit = 1, beim zweiten Aufruf = 0 gesetzt sein. Soll kein Reset durchgeführt werden, muß dieses Bit = 0 gesetzt werden.
1	0 = Uhr starten, 1 = Uhr stoppen
2	1 = 24 Stundenanzeige, 0 = 12 Stundenanzeige
3	0 = Interruptausgang demaskieren, 1 = Interruptausgang maskieren
4,5	Bit-5 Bit-4 Frequenz am Impulsausgang: 0 0 64 Hz (1/15,625 ms) 0 1 1 Hz (1/1 sec) 1 0 0,0166 Hz (1/1 min) 1 1 0,000277 Hz (1/1 h)
6	immer = 1 setzen
7	immer = 0 setzen

ml8rt_get_date_and_time — Lies Datum und Uhrzeit

Pascal: PROCEDURE ml8rt_get_date_and_time (VAR year, month, day, day_of_week, hour, minute, second: WORD);

C: void ml8rt_get_date_and_time (ushort *year, ushort *month, ushort *day, ushort *day_of_week, ushort *hour, ushort *minute, ushort *second);

Funktion: Diese Prozedur liest Datum und Uhrzeit der Echtzeituhr.

Parameter:

- *year*: Jahreszahl, vierstellig (dezimal, z.B.: "1998").
- *month*: Monat
- *day*: Tag
- *dow*: Wochentag (0 = Sonntag, 1 = Montag, ..., 6 = Samstag).
- *hour*: Stunde
- *minute*: Minute
- *second*: Sekunde

ml8rt_set_date_and_time — Setze Datum und Uhrzeit

Pascal: PROCEDURE ml8rt_set_date_and_time (year, month, day, dow, hour, minute, second: WORD);

C: void ml8rt_set_date_and_time (ushort year, ushort month, ushort day, ushort dow, ushort hour, ushort minute, ushort second);

Funktion: Diese Prozedur setzt Datum und Uhrzeit der Echtzeituhr.

Parameter: siehe **ml8rt_get_date_and_time.**

ml8rt_get_rtc_date_code — Lies Datecode der Echtzeituhr

Pascal: FUNCTION ml8rt_get_rtc_date_code : LONGINT;

C: ulong ml8rt_get_rtc_date_code (void);

Funktion: Diese Funktion liest das Datum der Echtzeituhr. Das Datum wird als Datecode übergeben (siehe Kapitel 9.8).

ml8rt_get_rtc_time_code — Lies Timecode der Echtzeituhr

Pascal FUNCTION ml8rt_get_rtc_time_code : LONGINT;

C ulong ml8rt_get_rtc_time_code (void);

Funktion Diese Funktion liest die Uhrzeit der Echtzeituhr. Die Uhrzeit wird als Timecode übergeben (siehe Kapitel 9.8).

9.6.6. Coprozessor

ml8rt_store_80487 — Sichere Coprozessorstatus

Pascal PROCEDURE ml8rt_store_80487;

C void ml8rt_store_80487 (void);

Funktion Diese Prozedur sichert den kompletten Zustand des Coprozessors. Sie muß immer dann am Anfang einer DI- oder II-Task verwendet werden, wenn die Hauptprozedur der Task Floating-Point-Operationen veranlaßt.

Alle Register werden innerhalb der Bibliothek gespeichert. Innerhalb einer Task darf die Funktion deshalb erst dann wieder aufgerufen werden, wenn der Coprozessor mit **ml8rt_restore_80487** wieder hergestellt worden ist.

ml8rt_restore_80487 — Stelle Coprozessorstatus wieder her

Pascal PROCEDURE ml8rt_restore_80487;

C void ml8rt_restore_80487 (void);

Funktion Diese Prozedur stellt den Zustand des Coprozessor so wieder her, wie er mit **ml8rt_store_80487** gespeichert wurde. Sie wird in der Regel am Ende der Hauptprozedur einer DI- oder II-Task eingefügt, wenn mit Floating-Point-Werten gearbeitet wird.

9.6.7. Service-Requests

ml8rt_send_buffer_srq Sende gepufferten Service-Request

Pascal PROCEDURE ml8rt_send_buffer_srq (data, mode: WORD);

C void ml8rt_send_buffer_srq (ushort data, ushort mode);

Funktion Diese Prozedur sendet einen Service-Request zum PC. Sie löst auf dem PC einen Interrupt aus und schreibt ein Wort in die Schnittstelle des PC. Wenn der Request nicht sofort abgesetzt werden kann, z.B. weil die Schnittstelle belegt ist, dann wird die Anforderung in eine Warteschlange eingereiht und sobald wie möglich automatisch durch das Betriebssystem ausgeführt. Wenn ein Fehler auftritt, wird die Fehlerbehandlungsroutine aufgerufen.

Parameter *data*: Datenwort, das an den Host-PC gesendet wird.

! Hinweis Das Low Byte von 'data' muß innerhalb des Bereichs von 80h bis bfh liegen (siehe Anhang F).

mode: Spezifiziert den Host, an den der Service-Request gesendet wird. Derzeit ist nur *mode* = 1 erlaubt (PC-Bus).

ml8rt_send_host_srq Sende direkten Service-Request

Pascal FUNCTION ml8rt_send_host_srq (data: WORD): WORD;

C ushort ml8rt_send_host_srq (ushort data);

Funktion Diese Funktion arbeitet grundsätzlich wie **ml8rt_send_buffer_srq**. Allerdings wird der Interrupt, wenn er nicht abgesetzt werden konnte, nicht gespeichert. Statt dessen wird als Funktionsergebnis ein Fehlercode zurückgegeben. Das Echtzeitprogramm muß die Wiederholung des Interrupt-Sendeversuchs selbst vornehmen. Diese Funktion darf (im Gegensatz zu **ml8rt_send_buffer_srq**) nur in der Hauptprozedur einer NI-Task angewendet werden!

Wenn der Interrupt nicht ausgelöst wurde, gibt das Funktionsergebnis den Grund an, anderenfalls ist es Null. Die einzelnen Bits haben folgende Bedeutung:

Bit-Nummer	Bedeutung
Bit-8 = 1	RBF (PC-Schnittstelle)
Bit-11 = 1	Interruptleitung nicht angeschlossen
Bit-12 = 1	Angewählte Interruptleitung ungültig
Bit-13 = 1	Softlock der Schnittstelle durch eine andere Task
Bit-14 = 1	DLP war gesetzt
Bit-15 = 1	TBF war gesetzt (Schnittstelle voll)

Parameter *data*: siehe **ml8rt_send_buffer_srq.**

Hinweis Wenn der Versuch, einen Service-Request zum PC zu senden, mißglückt ist, muß die Kontrolle wieder an das Betriebssystem zurückgegeben werden. Der nächste Versuch sollte erst beim nächsten Aufruf der NI-Task unternommen werden. !

9.6.8. Zugriffe auf sonstige Hardware-Funktionseinheiten

ml8rt_init_io Initialisiere Basiskarte und Module

Pascal PROCEDURE ml8rt_init_io (microslot, options: BYTE);

C void ml8rt_init_io (byte microslot, byte options);

Funktion Die Prozedur initialisiert alle I/O-Einheiten der angegebenen Hardware gemäß den Eintragungen im jeweiligen EEPROM.

Parameter *microslot*: Gibt an, welche Hardware initialisiert werden soll:
0: Basiskarte
1 bis 10: Modul auf dem entsprechenden Steckplatz.

option: Gibt an, ob in jedem Fall initialisiert wird (= 1), oder nur dann, wenn es laut Bit-0 im WORT-1 des EEPROMs erlaubt ist (Bit-0 = 1, *option* = 0).

ml8rt_local_led_on
ml8rt_local_led_off
ml8rt_external_led_on
ml8rt_external_led_off — Schalte LED ein/aus

Pascal
```
PROCEDURE ml8rt_local_led_on;
PROCEDURE ml8rt_local_led_off;
PROCEDURE ml8rt_external_led_on;
PROCEDURE ml8rt_external_led_off;
```

C
```
void ml8rt_local_led_on (void);
void ml8rt_local_led_off (void);
void ml8rt_external_led_on (void);
void ml8rt_external_led_off (void);
```

Funktion: Die Prozeduren schalten die LED auf der Basiskarte (local_led) bzw. die an Stecker St1 angeschlossene LED (external_led) ein bzw. aus.

ml8rt_trigger_watchdog — Triggere Watchdog

Pascal
```
PROCEDURE ml8rt_trigger_watchdog;
```

C
```
void ml8rt_trigger_watchdog (void);
```

Funktion: Diese Prozedur triggert den Watchdog der MODULAR-4 Basiskarte. Nähere Informationen zum Watchdog finden Sie in Kapitel 2 und 3.

9.7. Fehlerbehandlung

Wenn beim Aufruf einer Bibliotheksfunktion ein Fehler auftritt, springt die Bibliothek in eine Fehlerbehandlungsprozedur, die vom Benutzer programmiert werden kann. Innerhalb der Fehlerbehandlungsprozedur kann die Fehlerursache mit **ml8rt_get_error_info** bestimmt werden. Wenn keine Fehlerbehandlungsprozedur installiert ist, geschieht beim Auftreten eines Fehlers nichts.

ml8rt_set_error_handler Lege Fehlerbehandlungsprozedur fest

Pascal PROCEDURE ml8rt_set_error_handler (errhandle: POINTER);

C void ml8rt_set_error_handler (void* errhandle);

Funktion Mit dieser Prozedur wird der Bibliothek mitgeteilt, welche Prozedur nach dem Auftreten eines Fehlers aufgerufen werden soll. Die Prozedur muß als **FAR** compiliert worden sein (d.h. **retf** als Rücksprung).

Parameter *errhandle*: Adresse der Fehlerbehandlungsprozedur.

ml8rt_get_error_info Fordere Fehlerinformation an

Pascal PROCEDURE ml8rt_get_error_info
(VAR err_rec: ml8rt_error_info_type);

C void ml8rt_get_error_info (ml8rt_error_info_type *err_rec);

Funktion Diese Prozedur findet vor allem in einer Fehlerbehandlungsprozedur Verwendung. Sie liefert folgende Datenstruktur mit Informationen über einen aufgetretenen Fehler zurück.

Feldbezeichner	Typ	Bedeutung
errorcode	Word	Fehlercode des Betriebssystems
subnum	Word	Nummer der Betriebssystem-Subroutine, die den Fehler ausgelöst hat.
mark	Word	Vom Benutzer gesetzte Fehlermarkierung.
erroverflow	Word	Wird auf 1 gesetzt, wenn ein weiterer Fehler auftritt, bevor der letzte Fehler mit **ml8rt_reset_error** zurückgesetzt wurde.

Nach dem Auftreten eines Fehlers wird der Fehlerrecord durch nachfolgende Fehler nicht verändert. Das heißt, daß nur der erste Fehler im Fehlerrecord eingetragen wird. Nachfolgende Fehler setzen *errorverflow* = 1. Erst nach einem Aufruf von **ml8rt_reset_error** wird der Fehlerrecord wieder neu beschrieben.

ml8rt_reset_error — Setze Fehlerrecord zurück

Pascal PROCEDURE ml8rt_reset_error;

C void ml8rt_reset_error (void);

Funktion Diese Prozedur setzt den Fehlerrecord zurück.

ml8rt_set_mark — Markiere eine Programmstelle für die Fehlerbehandlung

Pascal PROCEDURE ml8rt_set_mark (mark: WORD);

C void ml8rt_set_mark (ushort mark);

Funktion Mit dieser Prozedur wird eine Bibliotheksvariable gesetzt. Tritt danach ein Fehler auf, so kann in der Fehlerbehandlungsprozedur anhand der Markierung auf den Programmteil geschlossen werden, in dem der Fehler auftrat.

Parameter: *mark*: Wert, auf den die Variable gesetzt werden soll.

ml8rt_force_error — Erzeuge eine Fehlermeldung

Pascal PROCEDURE ml8rt_force_error;

C void ml8rt_force_error (void);

Funktion Diese Prozedur löst einen Fehler mit der Nummer 0FE0h aus. Sie kann dazu verwendet werden, aus anderen Bibliotheken die Fehlerbehandlungsprozedur der **ML7RTBIB** bzw. **ML8RTBIB** aufzurufen.

9.8. Versionscode, Datecode und Timecode

Die Bibliothek liefert Versionsdaten als sogenannte Versions- und Datecodes. Die Funktionen für die Echtzeituhr stellen Datum und Uhrzeit ebenfalls als Date- und Timecode zur Verfügung. Die Codes bieten die Möglichkeit, auf einfache Weise Versionen, Daten und Uhrzeiten zu vergleichen. Ist z.B. der Versionscode von Version A grösser als der von Version B, so ist Version A die neuere von beiden.

Die Codes (32-Bit) haben folgenden standardisierten Aufbau:

31 … 24	23 … 16	15 … 8	7 … 0
AAAAAAAA	BBBBBBBB	CCCCCCCC	DDDDDDDD

Format des Versionscodes:

AAAAAAAA	Versionsnummer (z.B. 01h)
BBBBBBBB	Revisionsbuchstabe (z.B. 'A')
CCCCCCCC	Laufende Revisionsnummer (z.B. 03h)
DDDDDDDD	Status der Version: 'F' : Free Release (freigegebene Version) 'B' : Beta Version 'R' : RAM-Version (OsX) 'E' : EPROM-Version (OsX)

Format des Datecodes:

AAAAAAAA	Jahreszahl als Offset zu 1900
BBBBBBBB	Monat (1 bis 12)
CCCCCCCC	Tag (1 bis 31)
DDDDDDDD	Wochentag (0 = Sonntag, 1 = Montag, ..., 6 = Samstag)

Format des Timecodes:

AAAAAAAA	Stunden (0 bis 23)
BBBBBBBB	Minuten (0 bis 59)
CCCCCCCC	Sekunden (0 bis 59)
DDDDDDDD	Hunderstel Sekunden (0 bis 100)

Die oben angeführten Codes können auch in Klartext (Strings) umgewandelt werden. Die Strings werden dabei folgendermaßen formatiert:

Version 01.A.003 (F)

für Version 1, Rev. A, Revisionszähler 3, Free Release

Datum MON 16/02/1998

für Montag, den 16.02.1998

Zeit 11:26:41.18

für 11 Uhr, 26 Minuten, 41 Sekunden und 18 Hundertstel

Zur Umwandlung in den entsprechenden String stehen folgende Prozeduren zur Verfügung:

ml8rt_create_version_string — Versionscode in String umwandeln
ml8rt_create_date_string — Datecode in String umwandeln
ml8rt_create_time_string — Timecode in String umwandeln

Pascal

```
PROCEDURE ml8rt_create_version_string  (VAR v_string: STRING;
code: LONGINT);

PROCEDURE ml8rt_create_date_string  (VAR d_string: STRING;
code: LONGINT);

PROCEDURE ml8rt_create_time_string  (VAR t_string: STRING;
code: LONGINT);
```

C

```
void ml8rt_create_version_string (char* v_string, ulong code);

void ml8rt_create_date_string (char* d_string, ulong code);

void ml8rt_create_time_string (char* t_string, ulong code);
```

Funktion Die Prozeduren liefern die Version der verwendeten Bibliothek bzw. deren Erstellungsdatum als String zurück.

Parameter

v_string: Zeigt auf Variable zum Empfang des Version-Strings

d_string: Zeigt auf Variable zum Empfang des Datum-Strings

t_string: Zeigt auf Variable zum Empfang des Uhrzeit-Strings

code: Zu konvertierender Code

9.9. Sonstige Funktionen

ml8rt_set_pdt_adr — Setze die PDT-Adresse

Pascal PROCEDURE ml8rt_set_pdt_adr (headadr: Pointer);

C void ml8rt_set_pdt_adr (void *headadr);

Funktion Diese Prozedur übergibt die Adresse der Programm-Deskriptor-Tabelle an das Betriebssystem und initialisiert die Bibliothek. Die PDT muß zu diesem Zeitpunkt vom Programm angelegt und ausgefüllt worden sein. Das Betriebssystem entnimmt der PDT Daten, die zur Installierung des Programmes nötig sind (Tasktyp, Interruptnummer ...). Deshalb muß diese Funktion immer im **Prepare**-Teil eines Echtzeitprogrammes aufgerufen werden (Siehe Kapitel 7).

Parameter *headadr*: PDT-Adresse (Segment:Offset).

ml8rt_real_adr — Rechne Adresse um: Physikalisch ⇒ SEG:OFFS

Pascal PROCEDURE ml8rt_real_adr (physadr: LONGINT; VAR realptr);

C void ml8rt_real_adr (long physadr, void *realptr);

Funktion Diese Prozedur berechnet aus einer physikalischen Adresse eine Adresse vom Typ Segment:Offset. Die umgewandelte Adresse hat immer einen Offset kleiner 16 (normalisiert).

Parameter *physadr*: Physikalische Adresse.

realptr: Zeiger auf eine Variable, in die das Ergebnis vom Typ Segment:Offset eingetragen wird. Diese Variable ist normalerweise ein Zeiger.

Beispiel:

```
VAR   BytePtr : ^BYTE;                { Pointer auf ein Byte }
...
ml8rt_real_adr($12345, BytePtr);      {BytePtr = $1234(SEG):$0005(OFFS)}
```

Derselbe Aufruf muß in C wie folgt aussehen:

```
char     *byteptr;                    /* Pointer auf ein Byte */
...
ml8rt_real_adr(0x12345, &byteptr);    /* byteptr = 0x1234(SEG):0x0005(OFFS)*/
```

ml8rt_phys_adr

Rechne Adresse um: SEG:OFFS ⇒ Physikalisch

Pascal
PROCEDURE ml8rt_phys_adr (realadr: POINTER;
VAR physadr: LONGINT);

C
void ml8rt_phys_adr (void *realadr, ulong *physadr);

Funktion
Diese Prozedur berechnet aus einer Adresse vom Typ Segment:Offset eine physikalische Adresse.

Parameter
realadr: Umzuwandelnde Zeigervariable (Segment:Offset).

physadr: Zeiger auf eine Variable, in die das Ergebnis der Konvertierung eingetragen wird.

10. Assembler-Programmierung

10.1. System-Subroutinen

Auf fast alle Strukturen des Betriebssystems und auf die Strukturen jeder Task, also deren Datenbereich, Parameterbereich und Prozeduren bzw. Funktionen kann von jeder Task aus über die folgenden Subroutinen zugegriffen werden (einschließlich PDT und TDT).

Die Subroutinen sind aus Geschwindigkeitsgründen in Assembler geschrieben, ebenso wie das gesamte Betriebssystem. Aus dem gleichen Grund wurden andere Aufrufkonventionen gewählt als Sie sie vielleicht von MS-DOS her kennen.

Hin- und Rückgabeparameter werden in Registern oder auch in einigen Fällen auf dem Stack übergeben. Gegebenenfalls wird ein Pointer übergeben, dann meist in eax. Adressen werden (bis auf wenige Ausnahmen) als physikalische 32-Bit-Adressen angegeben. Auch die internen Pointer des Betriebssystems werden als physikalische 32-Bit-Adressen gehalten. Die max. Blockgröße für Blocklese- und Blockschreibroutinen beträgt 64 kByte - 256 Byte = 65280 Byte, z.B. bei READ_ DATA_BLOCK. Es widerspricht der Philosophie des Betriebssystems, sehr große Blöcke "am Stück" zu übertragen. Deshalb wird auch hier empfohlen, nur kleinere Blöcke zu übertragen oder die entsprechende Routine mehrmals aufzurufen. Aus Geschwindigkeitsgründen wird in den Routinen auf die Überprüfung der Blockgröße verzichtet, es wird also kein Fehler gemeldet.

Flags (f): Nach Rückkehr aus einer Subroutine zeigt das Carry-Flag (CY) immer an, ob die Routine ohne Fehler ausgeführt werden konnte. Wenn CY = 1 gesetzt ist, ist ein Fehler aufgetreten. In Register ax steht dann eine Erklärung zum Fehler. Die Fehlercodes entsprechen denen, die auch bei Makrobefehlen auftreten können (siehe Anhang F).

Alle Subroutinen lassen den Status des maskierbaren CPU-Interrupts, also das Interrupt Enable Flag (IF), und das Direction Flag (DF) unverändert. Einige Subroutinen verändern diese Flags vorübergehend, hinterlassen sie aber so, wie sie sie vorgefunden haben. Keine der Subroutinen ändert den Status des Nicht Maskierbaren Interrupts (NMI).

Alle Aufrufe von Subroutinen geschehen indirekt über Pointer, die ab 00:400h am Anfang des Speicherbereichs stehen. Jeder Pointer besteht aus 4 Byte (Segment:Offset).

Bei den Subroutinen mit Parameterübergabe in Registern sieht der Aufruf aus einem Assembler-Programm, z.B. zum Setzen eines Parameterbyte einer beliebigen Task, folgendermaßen aus (Borland Turbo Assembler, Ideal Modus):

```
push  es                        ; ein Segmentregister,
xor   ax,ax                     ;  z.B. es, retten und
mov   es,ax                     ;  = 0 setzen

mov   dx,TASK_NR                ; Ziel-Task
mov   bx,PARAMETER_NR           ; rel. Adresse
mov   al,DATA                   ; neuer Wert (Byte)
call  [DWORD FAR es: 400h + SUBROUTINE_NR * 4]
                                ; = db 26h,0ffh,1eh,16*4,4
pop   es
jc    FEHLER                    ; Fehler?
....  .......                   ; nein
```

In Kapitel 10.2 finden Sie ein komplettes Beispielprogramm in Assembler.

Hinweis: Beachten Sie, daß Sie bei der direkten Programmierung von einigen IC's (z.B. SCC) zwischen einzelnen Zugriffen auf I/O-Adressen dieses IC's ein Delay (z.B. 1 Mikrosekunde) einfügen müssen. Berücksichtigen Sie bei der Dauer des Delays die Beschreibung des Herstellers.

Folgende Routinen sind vorhanden (Betriebssystemversion ML7-3B.11x/ ML8-3B.11x, x=E: ROM-Version, x=R: Download-Version, x=B: Beta-Version):

Nr.	Adr.	Name	Funktion (Kurzbeschreibung)	Seite
0	400h	FORCE_ERROR	Provoziert eine Fehlermeldung	10-5
1	404h	PROG_IN_ROM	Melde, ob Programm im ROM ist	10-6
2	408h	INSTALL_TASK	Installiere Programm unter Task	10-5
3	40ch	GET_TASK_INFO	Info über eine Task (z.B. TDT, PDT)	10-8
4	410h	GET_TASK_STATUS	Melde, ob Task aktiviert ist	10-8
5	414h	WAKEUP_TASK	Aktiviere Task	10-9
6	418h	SLEEP_TASK	Deaktiviere Task	10-10
7	41ch	-	Reserviert	
8	420h	GET_FUNC_ADDRESS	Melde Adresse einer Funktion	10-10
9	424h	CALL_PROC	Prozedur einer Task aufrufen	10-11
10	428h	CALL_FUNC	Funktion einer Task aufrufen	10-11
11	42ch	GET_PAR_ADDRESS	Melde Adresse eines Parameters	10-13
12	430h	READ_PAR_BYTE	Lies Parameterbyte	10-14
13	434h	READ_PAR_WORD	Lies Parameterwort	10-14
14	438h	READ_PAR_DWORD	Lies Parameterdoppelwort	10-14
15	43ch	READ_PAR_BLOCK	Lies Parameterblock	10-15
16	440h	WRITE_PAR_BYTE	Schreibe Parameterbyte	10-15
17	444h	WRITE_PAR_WORD	Schreibe Parameterwort	10-16
18	448h	WRITE_PAR_DWORD	Schreibe Parameterdoppelwort	10-16
19	44ch	WRITE_PAR_BLOCK	Schreibe Parameterblock	10-17
20	450h	RESET_R_POINTER	Setze Daten-Lesezeiger auf Anfang	10-18
21	454h	RESET_W_POINTER	Setze Daten-Schreibzeiger auf Anfang	10-18
22	458h	MOVE_R_POINTER	Verschiebe Lesezeiger +/- Offset	10-18
23	45ch	MOVE_W_POINTER	Verschiebe Schreibzeiger +/- Offset	10-19
24	460h	READ_DATA_BYTE	Lies Datenbyte	10-19
25	464h	READ_DATA_WORD	Lies Datenwort	10-19
26	468h	READ_DATA_DWORD	Lies Datendoppelwort	10-20
27	46ch	READ_DATA_BLOCK	Lies Datenblock	10-20
28	470h	WRITE_DATA_BYTE	Schreibe Datenbyte	10-21
29	474h	WRITE_DATA_WORD	Schreibe Datenwort	10-21
30	478h	WRITE_DATA_DWORD	Schreibe Datendoppelwort	10-22
31	47ch	WRITE_DATA_BLOCK	Schreibe Datenblock	10-22
32	480h	ALLOCATE_RAM	Reserviere freien Speicher	10-23
33	484h	MASK_INT	Maskiere einen Interrupt	10-25
34	488h	UNMASK_INT	Demaskiere einen Interrupt	10-25
35	48ch	CLEAR_INT	Lösche einen Pending Interrupt	10-26
36	490h	END_OF_INT	Beende Interrupt-Service-Routine	10-26
37	494h	SET_INT_EDGE	Setze aktive Interrupt-Flanke	10-27
38	498h	-	Reserviert	
39	49ch	TRIGGER_WATCHDOG	Watchdog nachtriggern	10-27
40	4a0h	CACHE_CONTROL	Cache steuern	10-27

Nr.	Adr.	Name	Funktion (Kurzbeschreibung)	Seite
41	4a4h	LOCAL_LED_ON	LED (auf Basiskarte) "ein"	10-28
42	4a8h	LOCAL_LED_OFF	LED (auf Basiskarte) "aus	10-28
43	4ach	EXTERNAL_LED_ON	LED (extern) "ein"	10-28
44	4b0h	EXTERNAL_LED_OFF	LED (extern) "aus"	10-28
45	4b4h	GET_RTC_STATUS	Lies Status der Uhr	10-29
46	4b8h	SET_RTC_MODE	Initialisiere Uhr	10-30
47	4bch	GET_DATE_AND_TIME	Lies Datum und Uhrzeit	10-31
48	4c0h	SET_DATE_AND_TIME	Setze Datum und Uhrzeit	10-31
49	4c4h	GET_TIMER	Lies Timer (Zähler/Status)	10-32
50	4c8h	SET_TIMER	Setze Timer (Mode/Anfang)	10-32
51	4cch	READ_EEPROM_DIRECT	Lies Wort aus EEPROM direkt	10-33
52	4d0h	WRITE_EEPROM_DIRECT	Schreibe Wort in EEPROM dir.	10-34
53	4d4h	SEND_HOST_SRQ	Sende SRQ-Wort zum PC	10-35
54	4d8h	GET_TDT_ADDRESS	Lies Adresse einer TDT	10-36
56	4e0h	READ_EEPROM_COPY	Lies Wort aus EEPROM-Kopie	10-33
57	4e4h	WRITE_EEPROM_COPY	Schreibe Wort in EEPROM-Kopie	10-34
58	4e8h	GET_INT_TASK	Melde Task, die Interrupt nutzt	10-36
59	4ech	CONVERT_TIMER_DATA	Berechne Timer-Wert	10-36
60	4f0h	SEND_BUFFER_SRQ	Sendet einen gepufferten SRQ	10-37
65	504h	WAKEUP_TI_TASK	TI-Task aktivieren	10-38
70	518h	CREATE_BUFFER	Ringpuffer erzeugen	10-39
71	51ch	DELETE_BUFFER	Ringpuffer entfernen	10-40
72	520h	CLEAR_BUFFER	Ringpuffer komplett leeren	10-40
73	524h	GET_BUFFER_STATUS	Anzahl Zeichen im Ringpuffer bestimmen	10-41
74	528h	WRITE_BUFFER_BYTE	Ein Byte in Ringpuffer schreiben	10-41
75	52ch	WRITE_BUFFER_WORD	Ein Wort in Ringpuffer schreiben	10-41
76	530h	WRITE_BUFFER_DWORD	Ein Doppelwort in Ringpuffer schreiben	10-41
77	534h	WRITE_BUFFER_BLOCK	Block in Ringpuffer schreiben	10-42
78	538h	WRITE_BUFFER_MAX	Block in Ringpuffer schreiben (max.)	10-42
79	53ch	READ_BUFFER_BYTE	Ein Byte aus Ringpuffer lesen	10-43
80	540h	READ_BUFFER_WORD	Ein Wort aus Ringpuffer lesen	10-43
81	544h	READ_BUFFER_DWORD	Ein Doppelwort aus Ringpuffer lesen	10-43
82	548h	READ_BUFFER_BLOCK	Block aus Ringpuffer lesen	10-43
83	54ch	READ_BUFFER_MAX	Block aus Ringpuffer lesen (max.)	10-44
84	550h	VIEW_BUFFER_BLOCK	Block aus Ringpuffer kopieren	10-44
85	554h	VIEW_BUFFER_MAX	Block aus Ringpuffer kopieren (max.)	10-45
90	568h	GET_TASK_NUMBER	Tasknummer zu Programmnummer ermitteln	10-45
93	574h	INIT_IO	Initialisieren eines Moduls oder einer Basiskarte	10-48
104	5a0h	FLASH_STATUS	Setze bzw. lies Status von Flash	10-49
105	5a4h	FLASH_ERASE	Lösche 1 oder mehrere Sektoren im Flash	10-50
106	5a8h	FLASH_PROGRAM	Programmiere Block im Flash	10-50

Nr.	Adr.	Name	Funktion (Kurzbeschreibung)	Seite
107	5ach	FLASH_READ	Lies Block aus Flash	10-50
112	5c0h	GET_PAR_SEMA	Semaphore für Parameterbereich einer Task holen	10-13
113	5c4h	RESET_PAR_SEMA	Semaphore für Parameterbereich einer Task freigeben	10-13
114	5c8h	GET_DATA_SEMA	Semaphore für Datenbereich einer Task holen	10-17
115	5cch	RESET_DATA_SEMA	Semaphore für Datenbereich einer Task freigeben	10-17
116	5d0h	READ_DATA_OFFS	Datenblock einer Task lesen	10-21
117	5d4h	WRITE_DATA_OFFS	Datenblock einer Task schreiben	10-23

FORCE_ERROR (Nr. 0)

Diese Subroutine dient Testzwecken und liefert nur eine Fehlermeldung zurück. Bei der Rückgabe ist CY immer = 1.

Entry:	-	
Changed:	f, ax	
Exit (CY=0):	-	CY=0 kommt nicht vor
Exit (CY=1):	ax	Fehlercode: 0fe0h = Subroutine nicht implementiert

PROG_IN_ROM (Nr. 1)

Diese Subroutine prüft, ob ein bestimmtes Programm im ROM enthalten ist.

Entry:	ax	Programm-Nr.
Changed:	f, ax	
Exit (CY=0):	ax	0: Programm nicht vorhanden unverändert: Programm ist im ROM
Exit (CY=1):	ax	Fehlercode: 0fe0h = Subroutine nicht implementiert

INSTALL_TASK (Nr. 2)

INSTALL_TASK installiert ein Programm unter einer Task. Die Installierung geschieht entsprechend einem Record, auf den das Register eax zeigt (eax = 32 Bit physikal. Adresse). Es sind die gleichen Installierungen und Optionen möglich wie mit dem Makrobefehl 40h. INSTALL_TASK darf nicht in einer Auto-Init-Routine aufgerufen werden.

Entry:	eax	Physikal. Adresse (32 Bit) des Records, muß im Real-Adress-Bereich (0...1 Mbyte) liegen.
Changed:	f, eax	
Exit (CY=0):	-	Programm installiert
Exit (CY=1):	ax	Fehlercode (ein Fehler kann auch von Auto-Init oder PREPARE kommen)

Der Aufbau des Installierungsrecords in Kurzform. Der Record hat eine Länge von 22 Byte. Die Bedeutung der Adresse *a* in diesem Record richtet sich nach der Art der zu installierenden Datei, was in den Flags *f* vermerkt ist.

Offset	**Typ**	**Bedeutung**
0	Byte	Länge des Records, z.Zt. = 22
1	Byte	Typ der TDT, z.Zt. = 1
2	Byte	Länge der TDT, z.Zt. = 36
3	Byte	Reserviert = 0
4	Wort	Tasknummer (1 bis 1023)
6	Wort	Programm-Nummer (1 bis 65534, siehe auch PDT)
8	Byte	Interrupt-Nummer (nur bei DI- bzw. II-Tasks)
9	Byte	Reserviert = 0
10	Doppelwort	Flags *f*, Erklärungen siehe Kapitel 6, ml8_transfer_and_install
14	Doppelwort	Größe des Datenbereichs. Die Größe wird in Anzahl Byte angegeben. Weitere Erklärungen finden sich bei Flags zu Bit 9 und 10 (s.o.).
18	Doppelwort	Adresse *a*, Format und Wert abhängig von Flags. Die Adresse wird als physikalische oder Segment:Offset-Adresse angegeben (siehe Flags, Bit 6 bis 8, nächste Tabelle). Bei Verwendung der mitgelieferten PC-Bibliothek wird die Formatanpassung automatisch übernommen.

Wo?	**Programmformat**	**Adresse a**	**Format Adresse**	**Flag-Bit* 8 7 6**	**Typ**
RAM	PDT tiny	Anfang PDT	physikal.	0 0 0	Assembler
RAM	PDT large	Anfang PDT	physikal.	0 0 0	Assembler
RAM	EXE not relocated	Anfang EXE-Header	physikal.	1 1 0	C / Pascal
RAM	EXE relocated	START_UP Code	Seg:Offs.	0 1 0	Reserviert
ROM	PDT	wird ignoriert	-	0 0 1	Reserviert

* alle anderen Kombinationen sind zur Zeit nicht erlaubt

GET_TASK_INFO (Nr. 3)

GET_TASK_INFO liefert Informationen über eine Task, z.B. über die Task-Deskriptor-Tabelle (TDT) oder die Programm-Deskriptor-Tabelle (PDT). Die Segment:Offset-Adresse einer Prozedur bzw. Funktion kann mit GET_FUNC_ ADDRESS (Nr. 8) ermittelt werden. Wenn kein Programm unter der Task installiert ist, erfolgt eine Fehlermeldung. Die angefragte Task kann auch deaktiviert sein.

Entry:	dx	Tasknummer
	ax	Bestimmt die Art der Information (siehe Anhang H)
	ah = 0:	Inhalt der PDT lesen: al = rel. Adresse des PDT-Eintrags
	ah = 1:	Inhalt der TDT lesen: al = rel. Adresse des TDT-Eintrags
	ah = 2:	Adresse der TDT: al = 0
	ah = 3:	Inhalt der DDT (Debug-Descriptor-Table): al = rel. Adresse des DDT-Eintrags
Changed:	f, eax	
Exit (CY=0):	eax	4 Byte Info (siehe Anhang H)
Exit (CY=1):	ax	Fehlercode: 08e0h = kein Programm installiert 19e0h = keine DDT angelegt

GET_TASK_STATUS (Nr. 4)

GET_TASK_STATUS meldet die Anzahl der Aktivierungen einer Task. Interrupt-Tasks (DI- und II-Tasks) können nur einmal aktiviert werden, NI-Tasks auch mehrfach.

Entry:	dx	Tasknummer
Changed:	f, ax	
Exit (CY=0):	ax	Anzahl der Aktivierungen
Exit (CY=1):	ax	Fehlercode: 08d2h = Task nicht installiert

WAKEUP_TASK (Nr. 5)

WAKEUP_TASK aktiviert eine NI-, DI- oder II-Task einmal. NI-Tasks können auch mehrfach aktiviert werden (max. 255). Sie werden dann vom Task-Scheduler des Betriebssystems entsprechend oft aufgerufen. TI-Tasks können mit dieser Subroutine nicht aktiviert werden (siehe hierzu WAKEUP_TI_TASK (Nr. 65)).

Eine NI-Task kann mit dieser Subroutine auch einmalig in der Bearbeitung durch den Scheduler vorgezogen werden (ax = 2 oder ax = 3). Die Task kommt dann automatisch als nächste NI-Task an die Reihe, wenn die aktuell laufende NI-Task beendet wurde. Mit "Prüfung" (ax = 3) wird, wenn bereits eine NI-Task vorgezogen, aber noch nicht ausgeführt wurde, eine Fehlermeldung zurückgeliefert (ax = 20d4h). Falls die vorgezogene NI-Task nicht aktiviert war, wird sie nur ein einziges mal aufgerufen und dann wieder deaktiviert.

Entry:	dx	Tasknummer
	ax = 1	Aktivieren der Task (NI-, DI- oder II-Task)
	ax = 2	Einmaliges Vorziehen einer NI-Task (in jedem Fall)
	ax = 3	Einmaliges Vorziehen einer NI-Task mit Prüfung, ob bereits eine NI-Task auf vorgezogene Abarbeitung wartet. Wenn ja, erfolgt eine Fehlermeldung.

Changed:	f, ax		
Exit (CY=0):	-		
Exit (CY=1)	ax	Fehlercode:	10e0h = Falscher Parameter
			20d4h = Warnung: schon benutzt
			09d2h = Nicht implementiert
			08d2h = Task nicht installiert
			0ad2h = NI-Task-Tabelle voll

SLEEP_TASK (Nr. 6)

SLEEP_TASK deaktiviert eine Task einmal. Wenn eine NI-Task mehrfach aktiviert wurde, muß sie auch entsprechend oft wieder deaktiviert werden, damit sie inaktiv ist.

Entry:	dx	Tasknummer	
	ax	Immer = 1 (dies ist für zukünftige Entwicklungen vorgesehen)	
Changed:	f, ax		
Exit (CY=0):	-		
Exit (CY=1):	ax	Fehlercode:	0bd2h = Systemfehler
			08d2h = Task nicht installiert
			0cd2h = Task nicht aktiv

GET_FUNC_ADDRESS (Nr. 8)

GET_FUNC_ADDRESS ermittelt die Adresse einer Funktion resp. Prozedur einer Task. Die Adresse wird als Segment:Offset-Adresse zurückgegeben.

Entry:	dx	Tasknummer	
	bx	Funktions- bzw. Prozedur-Nr. (0, 1, 2, ...)	
Changed:	f, eax, bx, fs, di		
Exit (CY=0):	fs:di	Segment:Offset-Adresse der Funktion	
Exit (CY=1):	ax	Fehlercode:	08e0h = Task nicht installiert
			18e0h = Funktionsnummer ungültig

CALL_PROC (Nr. 9)

CALL_PROC ruft eine Prozedur einer Task auf. Es werden keine Parameter an die Prozedur übergeben und keine zurückerwartet.

Entry:	dx	Tasknummer
	bx	Prozedur-Nr. (0, 1, 2, ...)
Changed:		Wenn die Prozedur nur retf macht, sind f, eax, bx und cx geändert.
Exit (CY=0):	-	Kein Fehler aufgetreten
Exit (CY=1):	ax	Fehlercode: 18e0h = "Prozedur bzw. Funktion nicht vorhanden" (tritt auch auf, wenn gar kein Programm unter der Task installiert ist). Im Fehlerfall wird die Prozedur nicht mehr aufgerufen.

Zu Beginn der aufgerufenen Prozedur enthält dx die Tasknummer. Alle in der aufgerufenen Prozedur verwendeten Register müssen am Anfang gerettet und am Ende wieder restauriert werden (nicht nötig für f, eax, bx, cx, dx, ds). Die Prozedur kann keine Parameter, auch keine Fehlermeldung, zurückliefern.

CALL_FUNC (Nr. 10)

CALL_FUNC ruft eine Funktion einer Task auf, gegebenenfalls mit Parameterübergabe an die Funktion und/oder auch mit Parameterrückgabe zum aufrufenden Programm. Das aufrufende Programm muß jeweils einen Puffer für die Parameter 'Hin' und 'Zurück' zur Verfügung stellen, wenn Parameter übergeben bzw. zurückerwartet werden.

Die Überprüfung, ob die aufgerufene Funktion auch die Erwartungen des aufrufenden Programmes bzgl. Übergabeparameter erfüllen kann, wird von der aufgerufenen Funktion selbst vorgenommen (anders als bei C mit Prototyp, dort übernimmt das der Compiler). Zu Beginn der aufgerufenen Funktion sind dx, di, si, eax und ecx so gesetzt, wie sie an CALL_FUNC übergeben werden. In der aufgerufenen Funktion müssen die verwendeten Register am Anfang gerettet und am Ende wieder restauriert werden (nicht nötig für f, eax, ecx, bx, di, si, ds). Alle Register (außer f, sp und ds) werden so zum Aufrufenden zurückgegeben, wie sie die aufgerufene Funktion am Ende hinterlassen hat. Die aufgerufene Funktion kann einen Fehler an den Aufrufenden zurückliefern. Hierzu muß sie am Ende CY = 1 und ax = Fehlermeldung setzen. Die aufgerufene Funktion kann im Fehlerfall keine Parameter 'Zurück' zum Aufrufenden zurückliefern.

Funktionen können auch vom PC aus per Makrobefehl aufgerufen werden. Wenn die Funktion einen Fehler zum PC melden will, kann sie dies nur durch eine 1-Byte-Fehlermeldung tun. Die von der aufgerufenen Funktion gelieferte Fehlermeldung muß vom Typ xxe0h sein, dann wird das Byte xxh zum PC geliefert. Bei allen anderen Fehlergruppen wird xxh = 1ah "unbekannter Fehler" zum PC geliefert.

Entry:	dx	Tasknummer
	bx	Funktionsnummer (2, 3, 4, ...)
	di	Anzahl Byte, die der Funktion zur Verfügung gestellt werden (Parameter 'Hin')
	si	Anzahl Byte, die die Funktion zurückliefern soll (Parameter 'Zurück')
	eax	Zeiger auf den Puffer für die Parameter 'Hin' (physikalische Adresse)
	ecx	Zeiger auf den Puffer für die Parameter 'Zurück' (physikalische Adresse)
Changed:		Alle Register (dx, bx, di, si, eax und ecx) werden so zum Aufrufenden zurückgegeben, wie sie die aufgerufene Funktion am Ende hinterlassen hat. Wenn die aufgerufene Funktion nur CY = 0, si = 0 setzt und dann retf macht, sind nur f, bx und ggf. si geändert. Wenn die aufgerufene Funktion zum Melden eines Fehlers CY = 1, ax = xxe0h setzt und dann retf macht, sind nur f, bx und ax geändert.
Exit (CY=0):	di	Rest (Anzahl Byte) der von der aufgerufenen Funktion nicht verwendeten Byte 'Hin' (die Parameter am Anfang des Puffers werden von der aufgerufenen Funktion zuerst verwendet)
	si	Anzahl der von der aufgerufenen Funktion tatsächlich zurückgelieferten Byte 'Zurück'
Exit (CY=1):	ax	Fehler aufgetreten, ax = Fehlercode Der Fehler kann entweder beim Aufruf der Funktion auftreten (z.B. Fehlercode = 18e0h: "Prozedur bzw. Funktion nicht vorhanden oder kein Programm unter der Task installiert"), oder die Funktion kann durch CY = 1 und ax = xxe0h (xx = Fehlertyp) einen Fehler zurückliefern. Der Aufrufende muß die Auswertung der Fehlermeldung selbst übernehmen.

GET_PAR_ADDRESS (Nr. 11)

GET_PAR_ADDRESS ermittelt die physikalische 32-Bit-Adresse eines Parameters einer Task. Wenn bx = 0 gesetzt wird, wird die Adresse des Anfangs des Parameterbereichs gemeldet.

Entry:	dx	Tasknummer
	bx	Parameternummer (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
Changed:	f, eax	
Exit (CY=0):	eax	Adresse des Parameters (physikal.)
Exit (CY=1):	ax	Fehlercode

GET_PAR_SEMA (Nr. 112)

Diese Funktion versucht, die Semaphore für den Parameterbereich der Task t zu bekommen.

Entry:	ax	Task-Nr. t des Parameterbereichs	
Changed:	f, eax		
Exit (CY=0):	ax	Ergebnis:	0 = OK, Semaphore bekommen
			<> 0 = Semaphore nicht bekommen
Exit (CY=1):	ax	Fehlercode: 15e1h = Device nicht vorhanden	

RESET_PAR_SEMA (Nr. 113)

Diese Funktion gibt die Semaphore für den Parameterbereich der Task t wieder frei.

Entry:	ax	Task-Nr. t des Parameterbereichs
Changed:	f, eax	
Exit (CY=0):	ax	kein Fehler
Exit (CY=1):	ax	Fehlercode: 15e1h = Device nicht vorhanden

READ_PAR_BYTE (Nr. 12)

READ_PAR_BYTE liest den Wert eines Parameterbyte einer Task.

Entry:	dx	Tasknummer
	bx	Parameternummer (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
Changed:	f, eax	
Exit (CY=0):	al	Data
Exit (CY=1):	ax	Fehlercode

READ_PAR_WORD (Nr. 13)

READ_PAR_WORD liest den Wert eines Parameterwortes einer Task.

Entry:	dx	Tasknummer
	bx	Parameternummer (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
Changed:	f, eax	
Exit (CY=0):	ax	Data
Exit (CY=1):	ax	Fehlercode

READ_PAR_DWORD (Nr. 14)

READ_PAR_DWORD liest den Wert eines Parameterdoppelwortes (4 Byte) einer Task.

Entry:	dx	Tasknummer
	bx	Parameternummer (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
Changed:	f, eax	
Exit (CY=0):	eax	Data
Exit (CY=1):	ax	Fehlercode

READ_PAR_BLOCK (Nr. 15)

READ_PAR_BLOCK kopiert einen Datenblock aus dem Parameterbereich einer Task an die durch den Pointer in eax gegebene physikalische Adresse. Der Parameterbereich ist während des Zugriffs nicht vor dem Zugriff durch eine andere Task geschützt. Das könnte z.B. durch Maskieren des CPU-Interrupts vor dem Aufruf geschehen.

Entry:	dx	Tasknummer (Quelle)
	bx	Parameternummer des ersten zu kopierenden Parameters (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
	cx	Anzahl zu kopierender Parameterbyte (max. 64K - 256 = 65280)
	eax	Zieladresse (physikalisch)
Changed:	f, eax,cx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_PAR_BYTE (Nr. 16)

WRITE_PAR_BYTE schreibt ein Parameterbyte.

Entry:	dx	Tasknummer
	bx	Parameternummer (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
	al	Data
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_PAR_WORD (Nr. 17)

WRITE_PAR_WORD schreibt ein Parameterwort.

Entry:	dx	Tasknummer
	bx	Parameternummer (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
	ax	Data
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_PAR_DWORD (Nr. 18)

WRITE_PAR_DWORD schreibt ein Parameterdoppelwort (4 Byte).

Entry:	dx	Tasknummer
	bx	Parameternummer (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
	eax	Data
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_PAR_BLOCK (Nr. 19)

WRITE_PAR_BLOCK kopiert einen Datenblock ab der durch den Pointer in eax angegebenen physikalischen Adresse in den Parameterbereich einer Task. Der Parameterbereich ist während des Zugriffs nicht vor dem Zugriff durch eine andere Task geschützt. Das könnte z.B. durch Maskieren des CPU-Interrupts vor dem Aufruf geschehen.

Entry:	dx	Ziel-Tasknummer
	bx	Parameternummer des ersten zu schreibenden Parameters (= rel. Adresse, bezogen auf den Anfang des Parameterbereichs)
	cx	Anzahl zu kopierender Byte (max. 64K - 256 = 65280)
	eax	Quelladresse (physikalisch)
Changed:	f, eax, cx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

GET_DATA_SEMA (Nr. 114)

Diese Funktion versucht, die Semaphore für den Datenbereich der Task t zu bekommen.

Entry:	ax	Task-Nr. t des Datenbereichs	
Changed:	f, eax		
Exit (CY=0):	ax	Ergebnis:	0 = OK, Semaphore bekommen
			<> 0 = Semaphore nicht bekommen
Exit (CY=1):	ax	Fehlercode: 15e1h = Device nicht vorhanden	

RESET_DATA_SEMA (Nr. 115)

Diese Funktion gibt die Semaphore für den Datenbereich der Task t wieder frei.

Entry:	ax	Task-Nr. t des Datenbereichs
Changed:	f, eax	
Exit (CY=0):	ax	kein Fehler
Exit (CY=1):	ax	Fehlercode: 15e1h = Device nicht vorhanden

RESET_R_POINTER (Nr. 20)

RESET_R_POINTER setzt den Lesezeiger einer Task auf den Anfang des Datenbereichs derselben Task.

Entry:	dx	Tasknummer
Changed:	f, eax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

RESET_W_POINTER (Nr. 21)

RESET_W_POINTER setzt den Schreibzeiger einer Task auf den Anfang des Datenbereichs derselben Task.

Entry:	dx	Tasknummer
Changed:	f, eax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

MOVE_R_POINTER (Nr. 22)

MOVE_R_POINTER verschiebt den Lesezeiger einer Task um den in eax angegebenen Betrag. Der Pointer kann vorwärts (zu höheren Adressen) und rückwärts (zu niedrigeren Adressen) bewegt werden.

Entry:	dx	Tasknummer
	eax	Verschiebung in Anzahl Byte (2er-Komplement)
Changed:	f, eax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

MOVE_W_POINTER (Nr. 23)

MOVE_W_POINTER verschiebt den Schreibzeiger einer Task um den in eax angegebenen Betrag. Der Pointer kann vorwärts (zu höheren Adressen) und rückwärts (zu niedrigeren Adressen) bewegt werden.

Entry:	dx	Tasknummer
	eax	Verschiebung in Anzahl Byte (2er-Komplement)
Changed:	f, eax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

READ_DATA_BYTE (Nr. 24)

READ_DATA_BYTE liest den Wert eines Datenbyte einer Task von der durch den Lesezeiger (R-Pointer) gegebenen Adresse und inkrementiert danach den Lesezeiger um 1.

Entry:	dx	Tasknummer
Changed:	f, eax	
Exit (CY=0):	al	Data
Exit (CY=1):	ax	Fehlercode

READ_DATA_WORD (Nr. 25)

READ_DATA_WORD liest den Wert eines Datenwortes einer Task von der durch den Lesezeiger (R-Pointer) gegebenen Adresse und inkrementiert danach den Lesezeiger um 2.

Entry:	dx	Tasknummer
Changed:	f, eax	
Exit (CY=0):	ax	Data
Exit (CY=1):	ax	Fehlercode

READ_DATA_DWORD (Nr. 26)

READ_DATA_DWORD liest den Wert eines Datendoppelwortes einer Task von der durch den Lesezeiger (R-Pointer) gegebenen Adresse und inkrementiert danach den Lesezeiger um 4.

Entry:	dx	Tasknummer
Changed:	f, eax	
Exit (CY=0):	eax	Data
Exit (CY=1):	ax	Fehlercode

READ_DATA_BLOCK (Nr. 27)

READ_DATA_BLOCK kopiert einen Datenblock aus dem Datenbereich einer Task von der durch den Lesezeiger (R-Pointer) dieser Task gegebenen Adresse an die durch den Pointer in eax gegebene physikalische Adresse. Danach wird der Lesezeiger der Task um die Anzahl kopierter Byte inkrementiert. Der Datenbereich ist während des Zugriffs nicht vor dem Zugriff durch andere Tasks geschützt. Das könnte z.B. durch Maskieren des CPU-Interrupts vor dem Aufruf geschehen.

Entry:	dx	Tasknummer (Quelle)
	cx	Anzahl zu kopierender Byte (max. 64K - 256 = 65280)
	eax	Zieladresse (physikalisch)
Changed:	f, eax, cx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

READ_DATA_OFFS (Nr. 116)

Diese Funktion liest einen Datenblock aus dem Datenbereich einer Task ohne Verwendung des R-Pointers der angesprochenen Task.

Entry:	dx	Task-Nr. des Datenbereichs, der gelesen werden soll
	ebx	Offset zum Anfang des Datenbereichs der Task
	eax	Ziel-Adresse, wo die gelesenen Daten hin sollen (physikalisch)
	cx	Anzahl Byte zu lesen
Changed:	f, eax, ebx, cx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_DATA_BYTE (Nr. 28)

WRITE_DATA_BYTE schreibt ein Byte in den Datenbereich einer Task an die durch den Schreibzeiger (W-Pointer) gegebene Adresse und inkrementiert danach den Schreibzeiger um 1.

Entry:	dx	Tasknummer
	al	Data
Changed:	f, eax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_DATA_WORD (Nr. 29)

WRITE_DATA_WORD schreibt ein Wort in den Datenbereich einer Task an die durch den Schreibzeiger (W-Pointer) gegebene Adresse und inkrementiert danach den Schreibzeiger um 2.

Entry:	dx	Tasknummer
	ax	Data
Changed:	f, eax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_DATA_DWORD (Nr. 30)

WRITE_DATA_DWORD schreibt ein Doppelwort (4 Byte) in den Datenbereich einer Task an die durch den Schreibzeiger (W-Pointer) gegebene Adresse und inkrementiert dann den Schreibzeiger um 4.

Entry:	dx	Tasknummer
	eax	Data
Changed:	f, eax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_DATA_BLOCK (Nr. 31)

WRITE_DATA_BLOCK kopiert einen Block von Daten von der durch den Pointer in eax gegebenen physikalische Adresse in den Datenbereich einer Task an die durch den Schreibzeiger (W-Pointer) dieser Task gegebenen Adresse. Danach wird der Schreibzeiger dieser Task um die Anzahl kopierter Byte inkrementiert. Der Datenbereich ist während des Zugriffs nicht vor dem Zugriff durch andere Tasks geschützt. Das könnte z.B. durch Maskieren des CPU-Interrupts vor dem Aufruf geschehen.

Entry:	dx	Tasknummer (Ziel)
	cx	Anzahl zu kopierender Byte (max. 64K - 256 = 65280)
	eax	Quelladresse (physikalisch)
Changed:	f, eax, cx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

WRITE_DATA_OFFS (Nr. 117)

Diese Funktion schreibt einen Datenblock in den Datenbereich einer Task ohne Verwendung des W-Pointers der angesprochenen Task.

Entry:	dx	Task-Nr. des Datenbereichs, in den geschrieben werden soll
	ebx	Offset zum Anfang des Datenbereichs der Task
	eax	Quell-Adresse, wo die zu schreibenden Daten stehen (physikalisch)
	cx	Anzahl Byte zu schreiben
Changed:	f, eax, ebx, cx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

ALLOCATE_RAM (Nr. 32)

ALLOCATE_RAM reserviert Speicher auf der Karte. Die Routine kann auch verwendet werden, um nur die Größe des freien RAM zu ermitteln, ohne Speicher zu reservieren.

Für das Reservieren sind verschiedene Strategien möglich:

0 = Größe des freien RAM ermitteln

1 = UP absolut:
Es soll soviel Platz wie angefordert von der unteren Grenze des freien RAM an aufwärts reserviert werden. Wenn nicht genug Platz ist, wird nichts reserviert.

2 = UP max.:
Es soll soviel Platz wie möglich aber nicht mehr als angegeben von der unteren Grenze des freien RAM an aufwärts reserviert werden.

3 = DOWN absolut:
Es soll soviel Platz wie angefordert von der oberen Grenze des freien RAM an abwärts reserviert werden. Wenn nicht genug Platz ist, wird nichts reserviert.

4 = DOWN max.:
Es soll soviel Platz wie möglich aber nicht mehr als angegeben von der oberen Grenze des freien RAM an abwärts reserviert werden.

Alignment:

Hiermit kann die Anfangsadresse des reservierten Bereichs so gewählt werden, daß sie ohne Rest durch n teilbar ist, wobei n = 0, 1, 2, 4, 8, 16, ... sein kann.

Die Angaben von Tasknummer und Art der Verwendung des Speichers dienen betriebssysteminternen Zwecken und können auch weggelassen werden. Sie sind für zukünftige Entwicklungen vorgesehen.

Entry:	eax	Größe bzw. Maximum des zu reservierenden Bereichs (in Anzahl Byte), wird bei bl = 0 ignoriert.
	bl	Strategie: 0: nur Größe freies RAM liefern (Alignment in bh wird berücksichtigt) 1: UP absolut 2: UP max. 3: DOWN absolut 4: DOWN max.
	bh	Byte-Alignment: 2^{bh} (bh max. <= 15) z.B. bh = 2: DWORD (32 Bit)
	cx	Verwendung des Speichers (Statistik)
	dx	Nr. der Task, die den Speicher nutzt (Statistik)
Changed:	f, eax, ebx	
Exit (CY=0):	eax	Größe des freien RAM (bl = 0) bzw. Anfangsadresse des reservierten Bereichs (bl = 1, 2, 3, 4)
	ebx	Unverändert (bl = 0) bzw. Größe des reservierten Bereichs (bl = 1, 2, 3, 4)
Exit (CY=1):	ax	Fehlercode: 13d7h = Nicht genügend Platz 10e0h = falsche Parameter bei Aufruf

MASK_INT (Nr. 33)

Diese Subroutine maskiert einen Interrupt.

Entry:	al	Interrupt-Nr.:		
		2	(= 02h):	NMI
		121	(= 79h):	IRQ-F bzw. IRQ-TEMP[1]
		123	(= 7bh):	IRQ-A
		124	(= 7ch):	IRQ-B
		125	(= 7dh):	IRQ-C
		126	(= 7eh):	IRQ-D
		127	(= 7fh):	IRQ-E bzw. IRQ-FAN[1]
		144	(= 90h):	SCC
		145	(= 91h):	Timer-A
		146	(= 92h):	Timer-B
		147	(= 93h):	Timer-C
		148	(= 94h):	Timer-D (Uhr)
		149	(= 95h):	IRQ-G
		150	(= 96h):	IRQ-H
Changed:	f, ax			
Exit (CY=0):	-			
Exit (CY=1):	ax	Fehlercode: 10e0h = falsche Interrupt-Nr.		

UNMASK_INT (Nr. 34)

Diese Subroutine demaskiert einen Interrupt.

Entry:	al	Interrupt-Nr.: siehe bei Subroutine MASK_INT
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode: 10e0h = falsche Interrupt-Nr.

[1] Nur auf der "kleinen" MODULAR-4/486 möglich.

CLEAR_INT (Nr. 35)

Diese Subroutine löscht das dem entsprechenden Interrupt-Eingang zugehörige Statusbit im Interrupt Controller, das anzeigt, daß ein Interrupt aufgetreten ist und auf Bedienung wartet.

Zur Erklärung: Die Information, daß ein Interrupt auslösendes Ereignis (also eine aktive Flanke) aufgetreten ist, wird im Interrupt Controller gespeichert, unabhängig davon, ob der Interrupt maskiert ist oder nicht. Wenn er nicht maskiert ist, wird er wie üblich von der CPU, abhängig von der Priorität und dem Interrupt-Flag in der CPU, bedient. Wenn er maskiert war und nun demaskiert wird, wird dieser Interrupt ebenfalls in jedem Fall noch von der CPU bedient, gleichgültig, wann zuvor die aktive Flanke aufgetreten war. Das ist in vielen Fällen unerwünscht. Meistens sollen Interrupts erst ab einem bestimmten Zeitpunkt registriert werden. Dies läßt sich erreichen, wenn das zugehörige Statusbit im Interrupt Controller vorsichtshalber unmittelbar vor dem Demaskieren gelöscht wird.

Sonderfall NMI: Wenn ein "Nicht Maskierbarer" Interrupt (NMI) auftritt, während er maskiert ist, dann wird diese Information nicht wie bei den normalen Interrupts gespeichert. Nach dem Demaskieren wird also kein NMI ausgelöst, erst wenn das Interrupt auslösende Ereignis nach der Demaskierung wieder auftritt.

Entry:	al	Interrupt-Nr.: siehe bei Subroutine MASK_INT (ausgenommen NMI)
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode: 10e0h = falsche Interrupt-Nr.

END_OF_INT (Nr. 36)

Diese Subroutine beendet eine Interrupt-Service-Routine, löscht also den Interrupt, der gerade 'In-Service' ist (das entspricht dem Senden von EOI an den Interrupt-Controller)

Entry:	al	Interrupt-Nr. (siehe Subroutine MASK_INT) (ausgenommen NMI)
Changed:	f, ax	
Exit (CY=0)	-	
Exit (CY=1)	ax	Fehlercode, z.B. 10e0h = falsche Interrupt-Nummer

SET_INT_EDGE (Nr. 37)

Diese Subroutine setzt die aktive Flanke der externen Interrupt-Eingänge IRQ-A bis -H. Die Interrupts sind flankengetriggert.

Entry:	al	Interrupt-Nr.:		
		123	(= 7bh):	IRQ-A
		124	(= 7ch):	IRQ-B
		125	(= 7dh):	IRQ-C
		126	(= 7eh):	IRQ-D
		127	(= 7fh):	IRQ-E bzw. IRQ-FAN
		121	(= 79h):	IRQ-F bzw. IRQ-TEMP
		149	(= 95h):	IRQ-G
		150	(= 96h):	IRQ-H
	ah	0 = positive Flanke, 1 = negative Flanke		
Changed:	f, eax			
Exit (CY=0):	-			
Exit (CY=1):	ax	Fehlercode:	10e0h = falsche Interrupt-Nummer	

TRIGGER_WATCHDOG (Nr. 39)

Diese Subroutine (re)triggert den Watchdog auf der Karte.

Entry:	-	
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

CACHE_CONTROL (Nr. 40)

Diese Subroutine schaltet den Cache ein bzw. aus.

Entry:	eax	0: Cache ausschalten und ungültig machen
		1 oder 3: Cache einschalten
		4: Cache-Status melden
	ebx	Reserviert (= 0 setzen)
	ecx	Reserviert (= 0 setzen)
Changed:	f, ax	
Exit (CY=0):	al	0=Cache off, 1= Cache on
Exit (CY=1):	ax	Fehlercode

LOCAL_LED_ON (Nr. 41)

Diese Subroutine schaltet die on-board LED auf der Karte ein.

Entry:	-	
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

LOCAL_LED_OFF (Nr. 42)

Diese Subroutine schaltet die on-board LED auf der Karte aus.

Entry:	-	
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

EXTERNAL_LED_ON (Nr. 43)

Diese Subroutine schaltet die externe LED ein. Das Steuersignal ist an Stecker St1 herausgeführt.

Entry:	-	
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

EXTERNAL_LED_OFF (Nr. 44)

Diese Subroutine schaltet die externe LED ein. Das Steuersignal ist an Stecker St1 herausgeführt.

Entry:	-	
Changed:	f,ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

GET_RTC_STATUS (Nr. 45)

Diese Subroutine liest den Status der Echtzeituhr. Ein unabhängiger Impulsausgang der Uhr ist mit IRQ-4 des Interrupt-Slave-Controllers verbunden (= Timer-D). Mögliche Impulsfrequenzen sind: 1/15,625 ms, 1/s, 1/min oder 1/h. Die Impulsdauer (1-0-1) ist immer 7,8125 ms. Der invertierte Zustand des Impulsausgangs kann über das Statusregister gelesen werden.

Entry:	-	
Changed:	f, ax	
Exit (CY=0):	al	Status, Erklärung s.u.
Exit (CY=1):	ax	Fehlercode

Die Bits des Statusregisters:

Bit	Bedeutung
0	s.S. 6-42
1	1 = Uhr gestoppt, 0 = Uhr läuft
2	1 = 24 Stundenanzeige, 0 = 12 Stundenanzeige
3	1 = Interrupt-Ausgang maskiert, 0 = Interrupt-Ausgang demaskiert
4, 5	Frequenz des Impulsausganges
	Bit-5 / Bit-4 / Frequenz
	0 / 0 / 64 Hz (1 Impuls je 15,625 ms)
	0 / 1 / 1 Hz (1 Impuls je Sekunde)
	1 / 0 / 0,0166 Hz (1 Impuls je Minute)
	1 / 1 / 0,000277 Hz (1 Impuls je Stunde)
6	Zustand des Interrupt-Ausgangs (invertiert)
7	Reserviert

SET_RTC_MODE (Nr. 46)

Die Betriebsart der Uhr wird entsprechend des Bitmusters in al gesetzt.

Entry:	al	Mode, siehe unten
Changed:	f, ax	
Exit (CY=0):		
Exit (CY=1):	ax	Fehlercode

Die Bits des Moderegisters (al):

Bit	Bedeutung
0	Resetbit des 1 Hz Zählers. Zum Zurücksetzen des Zählers dieses Bit zuerst = 1 und anschließend mit einem zweiten Aufruf = 0 setzen.
1	1 = Uhr stoppen, 0 = Uhr starten
2	1 = 24 Stundenanzeige, 0 = 12 Stundenanzeige
3	1 = Interrupt-Ausgang maskieren, 0 = Interrupt-Ausgang demaskieren
4, 5	Frequenz des Impulsausganges Bit-5 Bit-4 Frequenz 0 0 64 Hz (1/15,625 ms) 0 1 1 Hz (1/1 s) 1 0 0,0166 Hz (1/1 min) 1 1 0,000277 Hz (1/1 h)
6	Immer = 1 setzen
7	Immer = 0 setzen

GET_DATE_AND_TIME (Nr. 47)

Diese Subroutine liest das Datum, den Wochentag und die Uhrzeit aus der Uhr. Die Zeit wird immer in 24-Stunden-Anzeige geliefert. Jedes Byte der Register eax und ebx enthält eine Angabe in binärer Form.

Beispiele: Die Sekunden werden in al geliefert mit einem Wertebereich von 0 bis 59 (00h bis 3bh). Der Wochentag steht in bl mit einem Wertebereich von 0 (Sonntag) bis 6 (Samstag). Das Jahr steht in Bit 24 bis 31 von Register ebx mit einem Wertebereich von 00 bis 255 (00 bis ffh)

Entry:	-	
Changed:	f, eax, ebx	
Exit (CY=0):	eax	Zeit: 00 / Stunden / Minuten / Sekunden
	ebx	Datum: Jahr / Monat / Tag / Wochentag
		(bei Wochentag: 0 = Sonntag, 1 = Montag ...)
Exit (CY=1):	ax	Fehlercode

SET_DATE_AND_TIME (Nr. 48)

Diese Subroutine setzt Datum, Wochentag und Uhrzeit in der Uhr. Außerdem wird der Subsekundenzähler zurückgesetzt. Angaben zum Format siehe bei GET_DATE_AND_TIME.

Beispiel: Die Uhr soll auf Montag, den 21.12.1992 und 12:58 gestellt werden: eax = 000c3a00h, ebx = 5c0c1501

Entry:	eax	Zeit: 00 / Stunden / Minuten / Sekunden
	ebx	Datum: Jahr / Monat / Tag / Wochentag
		(bei Wochentag: 0 = Sonntag, 1 = Montag ...)
Changed:	f, eax, ebx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

GET_TIMER (Nr. 49)

Diese Subroutine liefert den aktuellen Timer-Wert und den Status von Timer A, B oder C. Einzelheiten zu den Timern finden Sie in der Beschreibung des Bausteins 8254 von Intel (das ist der gleiche Chip, der auch im PC eingesetzt ist).

Entry:	al	Timer: 0 = A, 1 = B, 2 = C
Changed:	f, ax, cl	
Exit (CY=0):	ax	Aktueller Zählerstand
	cl	Status:
		Bit 0: 0 = binär, 1 = dezimal
		Bit 1 bis 3: Mode
		Bit 4 und 5: = 0
		Bit 6: Null-Count
		Bit 7: Zustand des Output-Pins
Exit (CY=1):	ax	Fehlercode: 10e0h = Falscher Parameter bei Aufruf

SET_TIMER (Nr. 50)

Diese Subroutine setzt die Betriebsart (Mode) und den Zählerwert von Timer A, B oder C. Einzelheiten zu den Timern finden Sie in der Beschreibung des Bausteins 8254 von Intel (das ist der gleiche Chip, der auch im PC eingesetzt ist). Für die üblichen Anwendungen wird der Timer in Mode 2 betrieben.

Um einen Timer anzuhalten, kann er vorübergehend z.B. in Mode 5 gesetzt werden. Er kann dann definiert gestartet werden, in dem er in Mode 2 oder Mode 3 gesetzt wird.

Über einige Module, z.B. M-D40-2, kann das Ausgangssignal eines Timers gepuffert an einem Pin des Moduls nach außen geleitet werden. Puls- und Pausendauer des Ausgangssignals sind abhängig vom eingestellten Mode des Timers.

In Mode 2 beträgt die Pausendauer (Ausgang = log. 0) immer nur einen Takt. Wenn z.B. als Zählerwert 5 eingestellt wird, beträgt die Pulsdauer 4, die Pausendauer 1 Takt. Beim (Neu-)setzen eines Timers bleibt der Ausgang unverändert, der Timer startet mit der neuen Pulsdauer. Um ein Puls-/Pausenverhältnis von 1 zu erreichen, muß der Timer in Mode 3 betrieben werden. Bei ungeraden Zählerwerten ist die Pulsdauer um einen Takt länger als die Pausendauer.

Zu diesem Thema ist auch eine Application Note von SORCUS verfügbar (AN042).

Entry:	al	Timer: 0 = A, 1 = B, 2 = C
	ah	Mode: 0, 1, 2, 3, 4 oder 5
	cx	Zählerwert
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode: 10e0h: Falscher Parameter bei Aufruf

READ_EEPROM_DIRECT (Nr. 51)
READ_EEPROM_COPY (Nr. 56)

READ_EEPROM_DIRECT liest ein Wort direkt aus dem EEPROM der Basiskarte oder eines Moduls. Dies sollte nur nach einem Reset der Basiskarte gemacht werden. Wenn mehrere Wörter gelesen werden sollen, ist nur ein Reset zu Beginn erforderlich.

Das Auslesen direkt aus einem EEPROM ist nur in den seltensten Fällen erforderlich. Die Informationen aus den EEPROMs werden nach jedem Reset automatisch in den Parameterbereich des Betriebssystems übertragen und stehen dann dort für den Anwender zur Verfügung. Der Zugriff darauf kann auch über READ_EEPROM_COPY erfolgen.

Entry:	al	Wortnummer (Wort 0 bis Wort 31)
	ah	Modulsteckplatz: 0 = Basiskarte, 1 bis 4 = Modulsteckplatz auf der Basiskarte[1] 5 bis 10 = Modulsteckplatz auf dem Modulextender
Changed:	f, ax	
Exit (CY=0):	ax	Data (16 Bit)
Exit (CY=1):	ax	Fehlercode: 10e0h = Falscher Parameter bei Aufruf 15e1h = Device nicht vorhanden

[1] Auf der "kleinen" MODULAR-4/486 stehen nur die Steckplätze 1 und 2 zur Verfügung!

WRITE_EEPROM_DIRECT (Nr. 52)
WRITE_EEPROM_COPY (Nr. 57)

WRITE_EEPROM_DIRECT schreibt ein Wort direkt in ein EEPROM der Basiskarte oder eines Moduls. Dies sollte nur nach einem Reset der Basiskarte gemacht werden. Mit dem Schreiben in das EEPROM wird auch dessen Kopie im Parameterbereich des Betriebssystems mit dem neuen Wert überschrieben.

WRITE_EEPROM_COPY kann verwendet werden, wenn nur die Kopie des EEPROM-Inhaltes im Parameterbereich des Betriebssystems verändert werden soll, aber nicht die Inhalte der EEPROMs.

Entry:	al	Wortnummer (Wort 0 bis Wort 31)	
	ah	Modulsteckplatz: 0 = Basiskarte, 1 bis 4 = Modulsteckplatz auf der Basiskarte[1] 5 bis 10 = Modulsteckplatz auf dem Modulextender	
	bx	Data (16 Bit)	
Changed:	f, ax		
Exit (CY=0):	-		
Exit (CY=1):	ax	Fehlercode:	10e0h = Falscher Parameter bei Aufruf 15e1h = Device nicht vorhanden

[1] Auf der "kleinen" MODULAR-4/486 stehen nur die Steckplätze 1 und 2 zur Verfügung!

SEND_HOST_SRQ (Nr. 53)

SEND_HOST_SRQ sendet ein Wort als Service Request über die parallele PC-Schnittstelle zum Host. Auf dem Host wird dadurch ein Interrupt ausgelöst, sofern per Jumper bzw. Software ein Interrupt-Kanal gewählt ist. Diese Subroutine darf nur aus der Hauptprozedur von Nicht-Interrupt-Tasks (NI-Task) aufgerufen werden, nicht aus Interrupt-Tasks (DI- oder II-Task) oder Timer-Initiierten Tasks (TI-Tasks). Wenn Sie statt SEND_HOST_SRQ die Routine SEND_BUFFER_SRQ (Nr. 60) verwenden, gibt es diese Einschränkung nicht.

Auch Fehlermeldungen des Betriebssystems werden auf diese Weise zum PC gemeldet. Dabei wird im Low Byte des SRQ-Wortes die Fehlergruppe (c0h bis ffh) gemeldet, siehe Anhang F.

Entry:	bx	Das zu sendende SRQ-Wort: Für den Anwender ist im Low Byte nur 80h bis bfh erlaubt, im High Byte 0 bis ffh.
Changed:	f, ax	
Exit (CY=0):	-	ok, SRQ gesendet
Exit (CY=1):	ax	Fehlercode bzw. Warnung: xxd4h = Es konnte nicht gesendet werden, xx = Grund: Bit 15 = 1: TBF war voll (PC-Schnittstelle) Bit 14 = 1: DLP war gesetzt (PC-Schnittstelle) Bit 13 = 1: Softlock der Schnittstelle durch eine andere Task Bit 12 = 1: Angewählte Interrupt-Leitung ungültig Bit 11 = 1: Interrupt-Leitung nicht angeschlossen Bit 8 = 1: RBF (PC-Schnittstelle)

GET_TDT_ADDRESS (Nr. 54)

GET_TDT_ADDRESS liefert die physikalische Adresse des Anfangs der Task-Deskriptor-Tabelle (TDT) einer Task. Aus Geschwindigkeitsgründen findet bei dieser Subroutine ausnahmsweise keine Fehlerprüfung statt (das CY-Flag ist nach Verlassen der Subroutine immer = 0 gesetzt), es wird also nicht geprüft, ob unter der Task ein Programm installiert ist . Diese Überprüfung muß im aufrufenden Programm selbst vorgenommen werden (z.B. durch Prüfung der Flags in der TDT). Alternativ kann auch die Subroutine GET_TASK_INFO verwendet werden, die eine Fehlerprüfung beinhaltet, aber langsamer ist.

Entry:	dx	Tasknummer
Changed:	f, eax	
Exit (CY=0):	eax	32 Bit physikal. Adresse der TDT

GET_INT_TASK (Nr. 58)

GET_INT_TASK ermittelt Nummer und Typ der Task, die einen bestimmten Interrupt nutzt.

Entry:	al	Interrupt-Nr. (0 bis 255)	
Changed:	f, ax, bx		
Exit (CY=0):	al	Tasktyp	0 = NI-Task, 1 = II-Task, 2 = DI-Task, 3 = TI-Task, 7 = Systemtask, z.B. Taskmanager
	bx	Nr. der Task, die den Interrupt nutzt (bx = - 1, wenn der Interrupt nicht benutzt ist)	
Exit (CY=1):	ax	Fehlercode: 0bd7h = "Warnung vom System", z.B. wenn die betriebssysteminternen Tabellen Unsinn enthalten (= Absturz des Systems).	

CONVERT_TIMER_DATA (Nr. 59)

Diese Subroutine wandelt eine Zeit, die in Mikrosekunden angegeben wird, in den passenden Timer-Wert zum Programmieren eines Timers der Basiskarte (A, B oder C), z.B. für die Verwendung mit der Subroutine SET_TIMER (Nr. 50). Dadurch kann die Programmierung in absoluten Zeitwerten erfolgen, unabhängig von der Frequenz des Quarzoszillators auf der Basiskarte, der den Eingangstakt für die Timer liefert.

Entry:	eax	Zeit (in Mikrosekunden)
Changed:	f, eax	
Exit (CY=0):	ax	Timer-Wert für Timer A, B oder C
Exit (CY=1):	ax	Fehlercode: 21e0h = "unerlaubte Zeitangabe"

SEND_BUFFER_SRQ (Nr. 60)

SEND_BUFFER_SRQ sendet ein Wort über einen Puffer (FIFO-Prinzip) als Service-Request an einen Host. Als Host kann auch die parallele PC-Schnittstelle angegeben werden. Dann ist das weitere Verhalten wie bei SEND_HOST_SRQ (s.o.). Diese Subroutine kann aus allen Tasks heraus aufgerufen werden.

Die Größe des Puffers (in Anzahl Byte) kann vor dem ersten Aufruf dieser Routine in Systemparameter 202 (Wort) angegeben werden. Nach dem ersten Aufruf steht dort die tatsächliche Größe des Puffers und in Systemparameter 204 (Wort) die Puffernummer. Die Länge eines SRQ bzw. einer Fehlermeldung im Puffer beträgt 4 Byte.

Entry:	al	Host-Schnittstelle: al = 0: Reserviert 1: Parallele PC-Schnittstelle 2-7: Anwenderdefiniert
	ah	Protokoll: z. Zt. = 0 setzen
	bx	Das zu sendende SRQ-Wort: Im Low Byte ist für Anwender-SRQs nur 80h bis bfh erlaubt, im High Byte 0 bis ffh.
	cx	Reserviert (immer = 0 setzen)
	dx	Reserviert (immer = 0 setzen)
Changed:	f, eax, cx	
Exit (CY=0):	-	ok, SRQ in Puffer eingetragen
Exit (CY=1):	ax	Fehlercode bzw. Warnung: 22d7h = "Puffer wird gerade beschrieben" 24d7h = "Nicht genug Platz im Puffer" 26e0h = "Puffer-Nr. ungültig" 90e0h = "Host schon/noch installiert" 91e0h = "Host nicht installiert" 10e0h = Falscher Parameter bei Aufruf der Subroutine, z.B. "Host-Nr. nicht erlaubt"

WAKEUP_TI_TASK (Nr. 65)

WAKEUP_TI_TASK aktiviert eine TI-Task (timerinitiierte Task) mit einem als Parameter übergebenen Zeitplan. Er beschreibt, wann und wie oft die TI-Task aufgerufen wird (siehe Beschreibung Kapitel 5). Basis des Zeitplans für alle TI-Tasks ist ein sogenannter Timer-Tick, wofür standardmäßig Timer C mit 1 ms Takt verwendet wird. Die Taktrate kann geändert werden, indem vor der ersten Installierung einer TI-Task der Parameter 316 des Betriebssystems geändert wird (Doppelwort, Zeitangabe in µs).

Wenn eine bereits aktivierte TI-TASK noch einmal mit WAKEUP_TI_TASK aktiviert wird, werden die alten Parameter verworfen und die neu übergebenen verwendet.

WAKEUP_TI_TASK kann jederzeit aus allen Tasks heraus aufgerufen werden. Eine laufende TI-Task kann sich auch selbst neu aktivieren oder deaktivieren und damit ihren eigenen Zeitplan ändern.

Um einer TI-Task eine sehr hohe Priorität einzuräumen, müssen Priorität und Hold-Off-Zeit = 0 gesetzt werden. Die übrigen Parameter können wie gewünscht eingestellt werden. Sie kommt dann auf jeden Fall als nächste TI-Task (beim nächsten Timer-Tick) an die Reihe. Ein gerade laufender Aufruf einer TI-Task wird aber zunächst normal beendet.

Benötigt eine TI-Task mehr Zeit als zwischen zwei Timer-Ticks zur Verfügung steht, dann sorgt das Betriebssystem dafür, daß die "verlorene" Zeit wieder aufgeholt wird, also daß die nachfolgenden TI-Tasks wieder zum vorgesehenen Zeitpunkt aufgerufen werden. Wann das erreicht ist, hängt von der aktuellen CPU-Auslastung ab.

Entry:	dx	Tasknummer
	cl	Ordnungszahl für Priorität (cl = 0: höchste, cl = 255: niedrigste). Bei gleicher Ordnungszahl erhält die zuletzt aufgerufene TI-Task die höhere Priorität.
	ch	= 0 (für zukünftige Entwicklungen reserviert)
	eax	Anzahl von Aufrufen, nach denen die Task wieder deaktiviert wird (eax = 0: unendlich)
	esi	Intervall zwischen zwei Aufrufen der TI-Task in Anzahl Timer-Ticks
	edi	Hold-Off-Zeit: Verzögerung vom Aktivieren bis zum ersten Aufruf der Task, in Anzahl Timer-Ticks
Changed:	f, eax, ebx, ecx, edx, esi, edi	
Exit (CY=0):	-	-
Exit (CY=1):	ax	Fehlercode: 08e0h = "kein Programm installiert"

CREATE_BUFFER (Nr. 70)

Einige wichtige Punkte beim Arbeiten mit diesen Puffern sind in Kapitel 5 beschrieben. CREATE_BUFFER legt einen Ringpuffer im Speicher auf der Karte an und liefert eine Puffernummer, die für alle weiteren Zugriffe auf den Puffer benutzt wird. Es können max. 256 voneinander unabhängige Puffer angelegt werden. Nach dem Aufruf von CREATE_BUFFER ist der Puffer leer. Beim Anlegen eines Puffers sind für das Reservieren von Speicherplatz verschiedene **Strategien** möglich:

1. *UP absolut:* Der Puffer soll so groß sein wie angegeben und von der unteren Grenze des freien RAM an aufwärts reserviert werden. Wenn nicht genug Platz ist, wird kein Puffer angelegt.
2. *UP max.:* Der Puffer soll so groß wie möglich, aber nicht größer als angegeben sein und von der unteren Grenze des freien RAM an aufwärts reserviert werden.
3. *DOWN absolut:* Der Puffer soll so groß sein wie angegeben und von der oberen Grenze des freien RAM an abwärts reserviert werden. Wenn nicht genug Platz ist, wird kein Puffer angelegt.
4. *DOWN max.:* Der Puffer soll so groß wie möglich, aber nicht größer als angegeben sein und von der oberen Grenze des freien RAM an abwärts reserviert werden.

Alignment:
Hiermit kann die Anfangsadresse des Puffers so gewählt werden, daß sie ohne Rest durch n teilbar ist, wobei n = 0, 1, 2, 4, 8, 16, ... sein kann.

Die Angaben der Tasknummer und der Art der Verwendung des Puffers dienen betriebssysteminternen Zwecken und können auch = 0 gesetzt werden. Sie sind für zukünftige Entwicklungen vorgesehen.

Entry:	eax	Absolute bzw. maximale Größe des anzulegenden Puffers (in Anzahl Byte)
	bl	Strategie: 1: UP absolut 2: UP max. 3: DOWN absolut 4: DOWN max.
	bh	Byte-Alignment: n = 2**bh (bh max. <= 16) z.B.: 2^0= 1, 2^1 = 2, 2^2 = 4
	cx	Verwendung des Speichers (z. Zt. = 0)
	dx	Nr. der Task, die den Speicher nutzt (z. Zt. = 0)
Changed:	f, eax, ebx	
Exit (CY=0):	eax	Puffer-Nr.
	ebx	Größe des angelegten Puffers (in Anzahl Byte)
Exit (CY=1):	ax	Fehlercode: 13d7h = "Nicht genug Speicherplatz im RAM" 10e0h = "Falsche/r Parameter bei Aufruf"

DELETE_BUFFER (Nr. 71)

DELETE_BUFFER ist noch nicht implementiert.

Entry:	eax	Puffer-Nr.	
Changed:	f, eax		
Exit (CY=0):	-	kein Fehler	
Exit (CY=1):	ax	Fehlercode:	0fe0h = "Nicht implementiert"

CLEAR_BUFFER (Nr. 72)

CLEAR_BUFFER löscht den Inhalt eines Puffers. Der Puffer ist danach leer. Das Löschen entspricht logisch einem Lesezugriff, bei dem der gesamte Pufferinhalt ausgelesen wird.

Entry:	eax	Puffer-Nr.	
Changed:	f, eax		
Exit (CY=0):	-	kein Fehler	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig" 23d7h = "Puffer wird gerade gelesen"

GET_BUFFER_STATUS (Nr. 73)

GET_BUFFER_STATUS liefert die Anzahl gültiger Zeichen (Byte) und den freien Platz im Puffer. Beide Angaben addiert ergeben die Länge des Puffers. Die Routine kann unabhängig vom Lesen und Schreiben desselben Puffers durch andere Tasks aufgerufen werden.

Sie müssen diese Routine nicht vor jedem Schreiben in oder Lesen aus einem Puffer aufrufen. Es ist meistens einfacher (und auch genauso schnell), die entsprechende Schreib- oder Leseroutine direkt mit der gewünschten Blockgröße aufzurufen. Wenn nicht genug Platz ist bzw. nicht genug Zeichen im Puffer sind, erhalten Sie eine Fehlermeldung bzw. eine Rückmeldung und können entsprechend reagieren. **!**

Entry:	eax	Puffer-Nr.	
Changed:	f, eax, ebx		
Exit (CY=0):	eax	Anzahl gültiger Zeichen (Byte) im Puffer	
	ebx	Freier Platz im Puffer (in Anzahl Byte)	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig"

WRITE_BUFFER_BYTE (Nr. 74)
WRITE_BUFFER_WORD (Nr. 75)
WRITE_BUFFER_DWORD (Nr. 76)

WRITE_BUFFER_BYTE schreibt ein Byte, WRITE_BUFFER_WORD ein Wort (2 Byte) und WRITE_BUFFER_DWORD ein Doppelwort (4 Byte) in den Puffer (siehe auch WRITE_BUFFER_BLOCK und WRITE_BUFFER_MAX). Wenn nicht genug Platz im Puffer ist, wird nichts geschrieben und es erfolgt eine Fehlermeldung.

Entry:	eax	Puffer-Nr.	
	bl	Datenbyte (bei WRITE_BUFFER_BYTE)	
	bx	Datenwort (bei WRITE_BUFFER_WORD)	
	ebx	Datendoppelwort (bei WRITE_BUFFER_DWORD)	
Changed:	f, eax		
Exit (CY=0):	-	kein Fehler	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig"
			24d7h = "nicht genug Platz im Puffer"
			22d7h = "P. wird gerade beschrieben"

WRITE_BUFFER_BLOCK (Nr. 77)

WRITE_BUFFER_BLOCK schreibt eine Anzahl Zeichen (Byte) in den Puffer. Dabei wird nach dem "Alles-oder-nichts-Prinzip" verfahren, d. h. es werden alle Zeichen in den Puffer geschrieben, wenn genügend Platz im Puffer ist, oder gar nichts (siehe auch WRITE_BUFFER_BYTE und WRITE_BUFFER_MAX).

Entry:	eax	Puffer-Nr.	
	ebx	Adresse (physikal.) der zu schreibenden Daten	
	ecx	Anzahl Byte zu schreiben (0 bis 65520)	
Changed:	f, eax, ebx, ecx		
Exit (CY=0):	-	kein Fehler	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig"
			24d7h = "nicht genug Platz im Puffer"
			22d7h = "P. wird gerade beschrieben"

WRITE_BUFFER_MAX (Nr. 78)

WRITE_BUFFER_MAX schreibt eine Anzahl Zeichen (Byte) in den Puffer. Dabei wird nach dem "Soviel-wie-möglich-Prinzip" verfahren, d. h. es werden soviel Zeichen in den Puffer geschrieben wie Platz im Puffer ist, maximal aber nur soviel wie beim Aufruf angegeben. Wenn nicht genügend Platz im Puffer ist, wird der Puffer voll geschrieben und die Anzahl Zeichen, die nicht im Puffer untergebracht werden konnten, zurückgemeldet (siehe auch WRITE_BUFFER_BLOCK).

Entry:	eax	Puffer-Nr.	
	ebx	Anfangsadresse (physikal.) der in den Puffer zu schreibenden Daten	
	ecx	Anzahl Byte zu schreiben (0 bis 65520)	
Changed:	f, eax, ebx, ecx		
Exit (CY=0):	eax	Rest = Anzahl Byte, die nicht in den Puffer geschrieben werden konnten.	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig"
			22d7h = "P. wird gerade beschrieben"

READ_BUFFER_BYTE (Nr. 79)
READ_BUFFER_WORD (Nr. 80)
READ_BUFFER_DWORD (Nr. 81)

READ_BUFFER_BYTE liest ein Byte, READ_BUFFER_WORD ein Wort (2 Byte) und READ_BUFFER_DWORD ein Doppelwort (4 Byte) aus dem Puffer (siehe auch READ_BUFFER_BLOCK, READ_BUFFER_MAX, VIEW_BUFFER_BLOCK und VIEW_BUFFER_MAX). Wenn nicht genug Zeichen im Puffer sind, wird nichts gelesen und es erfolgt eine Fehlermeldung.

Entry:	eax	Puffer-Nr.	
Changed:	f, eax		
Exit (CY=0):	al	Datenbyte (bei READ_BUFFER_BYTE)	
	ax	Datenwort (bei READ_BUFFER_WORD)	
	eax	Datendoppelwort (bei READ_BUFFER_DWORD)	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig"
			25d7h = "nicht genug Zeichen im P."
			23d7h = "P. wird gerade ausgelesen"

READ_BUFFER_BLOCK (Nr. 82)

READ_BUFFER_BLOCK liest Zeichen (Byte) aus dem Puffer. Dabei wird nach dem "Alles-oder-nichts-Prinzip" verfahren, d. h. es wird die gewünschte Anzahl Zeichen aus dem Puffer gelesen, sofern genügend Zeichen im Puffer sind, oder gar nichts (siehe auch READ_BUFFER_MAX, VIEW_BUFFER_BLOCK und VIEW_BUFFER_MAX).

Entry:	eax	Puffer-Nr.	
	ebx	Adresse (32 Bit physikal.), wo die zu lesenden Daten hin sollen	
	ecx	Anzahl Byte zu lesen (0 bis 65520)	
Changed:	f, eax, ebx, ecx		
Exit (CY=0):	-	kein Fehler	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig"
			25d7h = "nicht genug Zeichen im P."
			23d7h = "P. wird gerade ausgelesen"

READ_BUFFER_MAX (Nr. 83)

READ_BUFFER_MAX liest eine Anzahl Zeichen (Byte) aus dem Puffer. Dabei wird nach dem "Soviel-wie-möglich-Prinzip" verfahren, d. h. es werden soviel Zeichen aus dem Puffer gelesen wie gewünscht, maximal aber nur soviel wie beim Aufruf angegeben. Wenn nicht so viele Zeichen wie gewünscht im Puffer sind, wird der Puffer leer gelesen und die Anzahl Zeichen, die nicht gelesen werden konnten, zurückgemeldet (siehe auch READ_BUFFER_BYTE, READ_BUFFER_BLOCK, VIEW_BUFFER_BLOCK und VIEW_BUFFER_MAX).

Entry:	eax	Puffer-Nr.	
	ebx	Anfangsadresse (32 Bit physikal.), wo die aus dem Puffer zu lesenden Daten hin sollen	
	ecx	Anzahl Byte zu lesen (0 bis 65520)	
Changed:	f, eax, ebx, ecx		
Exit (CY=0):	eax	Rest = Anzahl Byte, die nicht aus dem Puffer gelesen werden konnten.	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig"
			23d7h = "P. wird gerade ausgelesen"

VIEW_BUFFER_BLOCK (Nr. 84)

VIEW_BUFFER_BLOCK kopiert eine Anzahl Zeichen (Byte) aus dem Puffer, die Zeichen bleiben aber im Puffer unverändert enthalten.

Dabei wird nach dem "Alles-oder-nichts-Prinzip" verfahren, d. h. es wird die gewünschte Anzahl Zeichen aus dem Puffer kopiert, sofern genügend Zeichen im Puffer sind, oder gar nichts (siehe auch VIEW_BUFFER_MAX, READ_BUFFER_BYTE, READ_BUFFER_BLOCK und READ_BUFFER_MAX).

Entry:	eax	Puffer-Nr.	
	ebx	Anfangsadresse (32 Bit physikal.), wo die zu kopierenden Zeichen hin sollen	
	ecx	Anzahl Byte zu kopieren (0 bis 65520)	
Changed:	f, eax, ebx, ecx		
Exit (CY=0):	-	kein Fehler	
Exit (CY=1):	ax	Fehlercode:	26e0h = "Puffer-Nr. ungültig"
			25d7h = "nicht genug Zeichen"
			23d7h = "P. wird gerade ausgelesen"

VIEW_BUFFER_MAX (Nr. 85)

VIEW_BUFFER_MAX kopiert eine Anzahl Zeichen (Byte) aus dem Puffer, die Zeichen bleiben aber im Puffer unverändert enthalten.

Dabei wird nach dem "Soviel-wie-möglich-Prinzip" verfahren, d. h. es werden soviel Zeichen aus dem Puffer kopiert wie gewünscht, maximal aber nur soviel wie beim Aufruf angegeben. Wenn nicht so viele Zeichen im Puffer sind wie angegeben, werden alle Zeichen, die im Puffer sind, kopiert, und die Anzahl Zeichen, die nicht kopiert werden konnten, zurückgemeldet (siehe auch VIEW_BUFFER_BLOCK, READ_BUFFER_BLOCK und READ_BUFFER_MAX).

Entry:	eax	Puffer-Nr.
	ebx	Anfangsadresse (32 Bit physikal.), wo die aus dem Puffer zu kopierenden Daten hin sollen
	ecx	Anzahl Byte zu kopieren (0 bis 65520)
Changed:	f, eax, ebx, ecx	
Exit (CY=0):	eax	Rest = Anzahl Byte, die nicht aus dem Puffer kopiert werden konnten
Exit (CY=1):	ax	Fehlercode: 26e0h = "Puffer-Nr. ungültig" 23d7h = "P. wird gerade ausgelesen"

GET_TASK_NUMBER (Nr. 90)

Mit GET_TASK_NUMBER kann die Nummer einer Task ermittelt werden, unter der ein Programm installiert ist. In den meisten Anwendungsfällen ist ein Programm nur unter einer Task installiert. Es kann aber auch mehrmals unter verschiedenen Tasks installiert sein (der Programmcode muß aber nur einmal auf der Karte sein). Mit dieser Systemroutine können sowohl die Anzahl der Installierungen als auch alle zugehörigen Tasknummern ermittelt werden.

Entry:	ax	Programm-Nr.
	cx	Aufrufparameter n n = 1: melde die niedrigste Tasknummer, unter der das Programm installiert ist n > 1: melde die jeweils höhere Tasknummer, falls das Programm mindestens n mal installiert ist n = 0 oder -1 (= ffffh): melde die höchste Tasknummer, unter der das Programm installiert ist und die Anzahl der Installierungen
Changed:	f, ax, bx, cx, dx	

Exit (CY=0)	ax	Programm-Nr. (unverändert)
	cx	cx = 0 (für alle n): Programm ist nicht installiert, dx ungültig cx = n (n = 1 bis 1024): Programm ist mindestens n-mal installiert, dx = Tasknummer mit der n-höchsten Tasknummer aller Installierungen dieses Programmes. cx = 1 bis 1024 (n = -1): cx = Anzahl Installierungen, dx = höchste Tasknummer aller Installierungen dieses Programmes. cx < n (n = 2 bis 1024): Annahme, daß das Programm n-mal installiert ist, war falsch, das Programm ist nur cx-mal installiert. dx = Tasknummer mit der cx-höchsten Tasknummer aller Installierungen dieses Programmes.
	dx	Tasknummer, sofern gültig (s.o.)
Exit (CY=1)	ax	Fehler, z. Zt. kein Fehler definiert

Beispiel 1: Prüfen, ob und wie oft ein Programm installiert ist:

Entry:	cx = 0	
Exit (CY=0):	cx = 0	Programm ist nicht installiert
	cx > 0	Programm ist cx mal installiert, z.B.: cx = 1: Programm ist 1 mal installiert, dx = Tasknummer dieser Installierung cx = 2: Programm ist 2 mal installiert, dx = Tasknummer der Installierung mit der höchsten Tasknummer. Um die andere Tasknummer zu bekommen, muß diese Subroutine noch einmal mit cx = 1 aufgerufen werden. cx = 3: Programm ist 3 mal installiert, dx = Tasknummer der Installierung mit der höchsten - Tasknummer. aller 3 Installierungen. Um die anderen beiden Tasknummern zu bekommen, muß diese Subroutine noch 2 mal aufgerufen werden, mit cx = 1 und dann mit cx = 2 (ax ist nach dem Aufruf nicht verändert).

Beispiel 2 (Sonderfall): Wenn man weiß, daß ein Programm nur einmal installiert sein kann, kann man Zeit sparen:

Entry:	cx = 1	
Exit (CY=0):	cx = 0:	Programm ist nicht installiert.
	cx = 1:	Programm ist 1 mal installiert, dx = Tasknummer.

Beispiel 3 (Sonderfall): Wenn ein Programm mehrfach installiert ist, sind alle Tasknummern verschieden. Mit cx wird die Ordnungszahl der Tasknummer angegeben, die man haben möchte:

cx = 1: niedrigste,
cx = 2: zweitniedrigste,
cx = 3: drittniedrigste, etc. bis
cx = -1 (ffffh): höchste Tasknummer

Fall 3a)

Entry:	cx = 1	
Exit (CY=0):	cx = 0:	Programm ist nicht installiert, dx ist ungültig
	cx = 1:	Programm ist 1 mal installiert, dx = Tasknummer

Fall 3b)

Entry:	cx = 2	
Exit (CY=0):	cx = 0:	Programm ist nicht installiert., dx = ungültig
	cx = 1:	Programm ist nur 1 mal installiert, dx = Tasknummer dieser Installierung
	cx = 2:	Programm ist 2 mal installiert, dx = Tasknummer der Installierung mit der höchsten Tasknummer

INIT_IO (Nr. 93)

INIT_IO initialisiert die Basiskarte bzw. ein Modul entsprechend den im zugehörigen EEPROM angegebenen Werten.

Entry:	al	Modulsteckplatz[1] (0 = Basiskarte)
	ah	ah = 0: nur initialisieren, wenn Bit 0 in WORT-1 des zugehörigen EEPROMs = 1 ist ah = 1: in jedem Fall initialisieren
Changed:	f, ax	
Exit (CY=0):	ax	ax = 0: ok, Initialisierung erfolgt
Exit (CY=1):	ax	Modul wurde nicht initialisiert wegen:
		Fehlercode: 1be0h: "Falscher Modulsteckplatz" 10e0h: "Falscher Parameter bei Aufruf" 1ce0h: "EEPROM-Info falsch" 15e1h: "Device nicht vorhanden"
		Warnungen: 80d4h: "Kein Modul auf dem Steckpl." 80d4h: "Modul braucht keine Initialis." 82d4h: "Bit-0 in EEPROM WORT-1=0 83d4h: "Keine Init-Subroutine in OsX"

Folgende Module können z.Zt. (ab Betriebssystemversion "ML7-3B.12x" bzw. "ML8-3B.12x") initialisiert werden:

Typ	Modul	Typ	Modul
1	MODULAR-4/486 Basiskarte	29	M-DA16-2
2	M-TRIG-1 (wie Typ 3)	32	M-COM-2
3	M-D40-2	34	M-DC15-2
6	M-OPT-1/A	37	M-AX-16
7	M-OPT-1/B und /Bx	38	M-AX-32
8	M-DA2-2	39	M-AD12-16
9	M-DA4-2	41	ML8-EX (Modul-Extender)
10	M-5B-1/M (wie Typ 20)	43	M-AD16-4
12	M-AD16-3	44	M-DPM-12
13	M-RU8-2	45	M-DPS-12
16	(M-SM-1)	47	M-C16-3

[1] Auf der "kleinen" MODULAR-4/486 stehen nur die Steckplätze 1 und 2 zur Verfügung!

Typ	Modul	Typ	Modul
19	M-IEC-1	48	M-COM-8
20	M-5B-1/U	49	M-CAN-1
22	M-SIO-8	50	M-DAS-A
24	M-SH12-8	51	M-C16-1
28	M-iNC-3	52	M-SSI-2

FLASH_STATUS (Nr. 104)

Setze bzw. lies Status von Flash.

Entry:	bl	Dienst (Erkl. siehe Makro-Befehl FLASH_STATUS)
	bh	Parameter
	cl	Steckplatz (0 = Basiskarte) des Flash
	ch	IC-Nr. (ML7 = IC6, ML8 = IC17)
Changed:	f, eax, bx, cx, dx	
Exit (CY=0):	ax	kein Fehler, Ergebnis
Exit (CY=1):	ax	Fehlercode: 84d4h: "Falscher Soft-State (Flash-EPROM) oder ein anderer Flash-EPROM Befehl läuft noch"
		85d4h: "Eingesetztes Flash-EPROM wird nicht unterstützt"
		86d4h: "Falscher Dienst (Flash-EPROM)"
		87d4h: "Kein Flash-EPROM lt. EEPROM"
		88d4h: "SP (Steckplatz) oder IC-Nr. falsch"
		89d4h: "Parameter nicht aligned, z.B. n ungerade bei Wort, auch bei Adresse a oder e nicht auf Sektorgrenze eines Flash-EPROM"
		8ad4h: "Adresse falsch"
		8bd4h: "Flash bzw. Sektor nicht gelöscht"
		8cd4h: "Flash nicht programmierbar / löschbar (Time-Out bzw. defekt)"

FLASH_ERASE (Nr. 105)

Lösche einen oder mehrere Sektoren im Flash.

Entry:	eax	Erste Adresse (auf Flash-Anfang bezogen und auf Sektoranfang aligned)
	ebx	Letzte Adresse (auf Flash-Anfang bezogen und auf Sektorende aligned)
	cl	Modul-Steckplatz (0=Basiskarte) des angespr. Flash
	ch	IC-Nr. (ML7 = IC6, ML8 = IC17)
Changed:	f, eax, ebx	
Exit (CY=0):	al	Einer der Sektoren, der gelöscht wurde
	ah	0
Exit (CY=1):	ax	Fehlercode (s. FLASH_STATUS)

FLASH_PROGRAM (Nr. 106)

Programmiere Block im Flash.

Entry:	eax	Pointer auf Datenblock
	ebx	Erste Adresse im Flash (rel. Adresse)
	cx	Anzahl Byte zu programmieren (max. 65535)
	dl	Modul-Steckplatz (0=Basiskarte) des angespr. Flash
	dh	IC-Nr. (ML7 = IC6, ML8 = IC17)
Changed:	f, eax, ebx, cx, dx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode (s. FLASH_STATUS)

FLASH_READ (Nr. 107)

Lies Block aus Flash.

Entry:	eax	Pointer, wo die Daten hin sollen
	ebx	Erste Adresse im Flash (rel. Adresse)
	cx	Anzahl Byte zu lesen (max. 65535)
	dl	Modul-Steckplatz (0=Basiskarte) des angespr. Flash
	dh	IC-Nr. (ML7 = IC6, ML8 = IC17)
Changed:	f, eax, ebx, cx, dx	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode (s. FLASH_STATUS)

10.2. Beispiel: LED-Blinkprogramm in Assembler

Zur Funktion des Programms M8P0380 (Programm 380h):

Mit Parameter 0 (Command/Status) des Programms wird das Programm und damit das Blinken der LED gestartet (Parameter 0 = 1 startet das Blinken). Danach wird Parameter 0 vom Programm selbst = 2 gesetzt und zeigt dadurch an, daß es läuft. Wenn Parameter 0 wieder = 0 gesetzt wird, stoppt das Programm und das Blinken hört auf. In diesem Zustand kann durch Setzen von Parameter 1 = 0 die LED aus- bzw. = 1 eingeschaltet werden. Den gleichen Effekt kann man auch durch Aufruf der Prozeduren 2 bzw. 3 dieses Programms erreichen.

Zur Programmierung:

Das Programm M8P0380 ist im IDEAL Mode in Borland Turbo-Assembler (Tiny Model) geschrieben. Der Turbo-Assembler muß mit folgenden Parametern aufgerufen werden:

/m3:	Erlaube 3 Phasen um Vorwärts-Referenzen zu finden
/mv32:	Maximale Länge von Symbolen auf 32 setzen
/q:	Fürs Linken nicht benötigte OBJ-Records unterdrücken

Der Turbo-Linker muß mit folgenden Parametern aufgerufen werden:

/3:	32-Bit processing ermöglichen
/t:	COM-Datei erzeugen

Das Programm M8P0380 ist möglichst einfach gehalten. Es soll die Programmierung einer einfachen NI-Task auf der MODULAR-4/486 Karte und deren Installierung zeigen. Zusätzlich sind die Prozeduren 2 und 3 vorgesehen, die mit dem eigentlichen Programm nichts zu tun haben, aber es sollte hier auch gezeigt werden, wie eine vom PC oder von einer anderen Task aufrufbare Prozedur aussehen kann.

Die Auto-Init Prozedur (Prozedur 1) sorgt hier für eine Initialisierung der Parameter. Beachten Sie aber, daß in der PDT alle Prozeduren fortlaufend durchnumeriert sind und keine Adresse fehlen darf (wenn noch weitere Prozeduren folgen). Wenn also die Auto-Init Prozedur nicht benötigt wird, muß in der PDT trotzdem eine gültige Adresse stehen. Es ist ratsam, dann auch eine Auto-Init Prozedur vorzusehen, die aber nichts macht, sie besteht also nur aus "retf". Die Auto-Init Prozedur kann wie alle anderen Prozeduren auch vom PC oder von einer anderen Task auf der Karte aufgerufen werden. Sie unterscheidet sich nicht von anderen Prozeduren, außer daß sie als Teil des Installierungsvorgangs zum Schluß der Installierung des Programms unter einer Task automatisch aufgerufen werden kann. Beachten Sie, daß die in der PDT angegebenen Adressen relative Adressen sind.

Nach dem Laden des Programms auf die Karte beim Installieren des Programms unter einer Task erfolgt eine Anpassung dieser Adressen (Relozierung). Dieses Programm hat keinen Datenbereich, der Parameterbereich wird lokal vom Programm selbst reserviert. In diesem Fall ist zu beachten, daß vor dem Parameterbereich immer Platz für die TDT frei gehalten werden muß (s.u.).

Zur Installierung:

Das Programm hat keinen Datenbereich, der Parameterbereich wird lokal vom Programm selbst reserviert. Beim Installieren wird auch der Task-Typ (0 = NI-Task) und das Programmformat angegeben.

Beispiel für Installierung dieses Programms mit SNW unter Task 10h und anschließender Aktivierung der Task und Start:

```
M8INST M8P0380.LIB 0380 0010 00 000000 00000000 ; Installiere Task 10h
M8CMD 41 02 10 00                               ; Aktiviere Task 10h
M8PAR 10 00 01                                  ; Setze Par. 0=1: Start
```

```
;-----------------------------------------------------------------------------
;M8P0380.ASM
;-----------------------------------------------------------------------------
;Programm 0380h: Blink LED als NI-Task
;-----------------------------------------------------------------------------
TITLE   M8P0380
;-----------------------------------------------------------------------------
MODEL TINY
.486
.CODE
IDEAL
;-----------------------------------------------------------------------------

VERS0380        equ     '1'             ; Programm-Version ASCII: 1234
REV0380         equ     'A'             ; Programm-Revision ASCII: ABCD
TYPPDT          equ     1               ; Typ der PDT (Konstante)
DEFLPDT         equ     48              ; Laenge des Vorspanns der PDT (Konst.)
DEFLTDT         equ     36              ; Laenge der TDT (Konstante)
LED_ON          equ     400h+41*4       ; System-Subroutine 41
LED_OFF         equ     400h+42*4       ; System-Subroutine 42

;-----------------------------------------------------------------------------
;M8P0380: Programm-Deskriptor-Tabelle (PDT)
;-----------------------------------------------------------------------------
M8P0380:
        DB      TYPPDT          ; 0     ; Typ der PDT
        DB      DEFLPDT         ; 1     ; Laenge des Vorspanns der PDT
        DW      (M8P0380E - M8P0380 - DEFLPDT)/4  ; Anzahl Prozeduren
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        DW      00380h          ; 4     ; Programm-Nr.
        DB      VERS0380                ; 6     ; Programm-Version
        DB      REV0380         ; 7     ; Programm-Revision
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        DB      4                       ; 8     ; Prozessor-Typ (1=186/V20,4=486)
        DB      0                       ; 9     ; Co-Prozessor-Typ (0=kein, 4=486DX)
        DB      1                       ; 10    ; Programmiersprache (1=ASM)
        DB      1                       ; 11    ; Programm-Typ (0=System, 1=Anwender)
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        DB      08h                     ; 12    ; Flag (Bit 0 bis 7): (ja=0)
                                ;       ;   Bit 0-2: Task-Typ (NI-Task=000)
                                ;       ;   Bit 3: Task-Typ, Int-Nr. von Install?
                                ;       ;   Bit 4: Real-Mode Programm?
                                ;       ;   Bit 5: Caching von Code erlaubt?
                                ;       ;   Bit 6: - (=0 setzen)
                                ;       ;   Bit 7: Kein Hypertext im Programm?
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        DB      4                       ; 13    ; Flag (Bit 8 bis 15): (ja=0)
                                ;       ;   Bit 8: Daten von OsX reservieren?
                                ;       ;   Bit 9: Groesse Daten variabel?
                                ;       ;   Bit 10: Parameter von OsX reservieren?
                                ;       ;   Bit 11-15: reserviert
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        DB      0                       ; 14    ; Interrupt-Nr. (entfaellt, hier = 0)
        DB      0                       ; 15    ; reserviert
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        DD      0                       ; 16    ; Anfangs-Adresse Datenbereich
        DD      0                       ; 20    ; Groesse Datenbereich
        DD      0                       ; 24    ; Minimum Datenbereich
        DD      10000h          ; 28    ; Maximum Datenbereich
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        DW      OFFSET PAR0380 ; 32     ; Anfangs-Adresse Parameterbereich
        DW      0                       ; 34    ;
        DW      PAR0380E - PAR0380; 36          ; Groesse Parameterbereich
        DD      0                       ; 38    ; Adresse Hypertextbereich
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        DW      0                       ; 42    ; reserviert
        DD      0                       ; 44    ; reserviert fuer SORCUS
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
        DW      OFFSET M8P0380_M; 48   ; Adresse der Main-Prozedur (Nr.0)
        DW      0
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        DW      OFFSET M8P0380_I; 52   ; Adresse der Auto-Init Prozedur (Nr.1)
        DW      0
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        DW      OFFSET M8P0380_2; 56   ; Adresse Prozedur LED_ON (Nr.2)
        DW      0
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        DW      OFFSET M8P0380_3; 60   ; Adresse Prozedur LED_OFF (Nr.3)
        DW      0
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
M8P0380E:

;-----------------------------------------------------------------------------
;Parameterbereich (ist Teil des Programms)
;-----------------------------------------------------------------------------
;vor dem Parameterbereich muss Platz fuer die TDT freigehalten werden

                db      DEFLTDT dup (0)        ; Laenge TDT

;-----------------------------------------------------------------------------

PAR0380:                                ; Anfang Parameterbereich
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
; Name                  Init    rel.Ad.   Bedeutung
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
STATUS:         db      0       ; 0     ; Command/Status:
                                ; Command: 0 = Stop
                                ;          1 = Start Blinking
                                ; Status:  2 = Programm laeuft
ZULED:          db      0       ; 1     ; Zustand der LED: 0 = off, 1 = on
RATE:           dw      40000   ; 2     ; Blink-Rate (ungefaehr)
COUNTER:        dw      1       ; 4     ; Counter: auf 1, damit sofort umgeschaltet wird
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
PAR0380E:                               ; Ende Parameterbereich


;-----------------------------------------------------------------------------
;Main-Prozedur (= Prozedur 0): Blink LED
;-----------------------------------------------------------------------------
;Vom Betriebssystem wird in dx bei Aufruf einer Prozedur (auch der
;Main-Prozedur) immer die Task-Nr uebergeben, unter der das Programm
;installiert ist, zu dem die Prozedur gehoert. Das wird aber hier nicht
;weiter ausgewertet.

PROC M8P0380_M          FAR                             ; alle Prozeduren muessen FAR sein

        pusha                                   ; es muessten nicht alle Register
        push    ds                              ;   gerettet werden, sondern nur die,
        push    es                              ;   die auch verwendet sind
        mov     ax,cs                           ; Tiny-Model: ds auf Anfang setzen
        mov     ds,ax
        xor     ax,ax                           ; es=0 fuer Aufruf von Systemroutinen
        mov     es,ax
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        mov     al,[BYTE ds:STATUS]     ; Status = 2 (Run) ?
        cmp     al,2
        jnz     P0380_X                 ; nein

        mov     cx,[WORD ds:COUNTER]    ; Programm laeuft
        dec     cx
        jnz     P0380_LP                        ; LED nicht umschalten

        mov     al,[BYTE ds:ZULED]      ; LED umschalten
        xor     al,1
P0380_ZX:       mov     [BYTE ds:ZULED],al      ; neuen Zustand der LED merken
        jz      P0380_OFF
        call    [DWORD es:LED_ON]
        jmp     P0380_LD
P0380_OFF:      call    [DWORD es:LED_OFF]

P0380_LD:       mov     cx,[WORD ds:RATE]               ; setze Zaehler neu
P0380_LP:       mov     [WORD ds:COUNTER],cx
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

```
P0380_END:      pop     es
        pop     ds
        popa
        ret                                     ; daraus macht TASM "retf"

P0380_X:        cmp     al,3                            ; Status = 3 (Fehler) ?
        jae     P0380_END                       ; ja, Ende
        and     al,al                           ; nein, Status = 0 (Programm Stop) ?
        jnz     P0380_MX
        mov     al,[BYTE ds:ZULED]      ; Setze LED auf gewuenschten Status
        cmp     al, 0
        jmp     P0380_ZX

P0380_MX:       mov     [BYTE ds:STATUS],2      ; setze Status immer = 2
        jmp     P0380_END

ENDP    M8P0380_M

;-----------------------------------------------------------------------------
;Auto-Init Prozedur (= Prozedur 1): Parameter auf Anfangs-Zustand
;-----------------------------------------------------------------------------
;Vom Betriebssystem wird in dx bei Aufruf einer Prozedur immer die
;Task-Nr uebergeben, das wird aber hier nicht ausgewertet.

PROC    M8P0380_I       FAR

        pusha
        push    ds
        mov     ax,cs                   ; Tiny-Model: ds auf Anfang setzen
        mov     ds,ax
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        mov     bx,OFFSET P0380_PI; Tabelle
        mov     di,[WORD ds:bx]         ; erster zu initialisierender Parameter
        add     di,OFFSET PAR0380
        mov     cx,[WORD ds:bx+2]       ; Anzahl Byte
        add     bx,4                    ; Pointer
M8P0380_I1:     mov     al,[BYTE ds:bx]
        mov     [BYTE ds:di],al
        inc     di
        inc     bx
        dec     cx
        jnz     M8P0380_I1
P0380_I2:
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        pop     ds
        popa
        ret
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
P0380_PI:       dw      0                       ; erster zu initialisierender Parameter
        dw      6                       ; Anzahl (Byte) Parameter zu initialisieren

        db      0                       ; STATUS
        db      0                       ; ZULED: Led-Zustand
        dw      4000                    ; RATE: Blink-Rate
        dw      1                       ; COUNTER

ENDP    M8P0380_I
```

```
;-----------------------------------------------------------------------------
;Prozedur 2: LED ON
;-----------------------------------------------------------------------------
;Vom Betriebssystem wird in dx bei Aufruf einer Prozedur immer die
;Task-Nr uebergeben, das wird aber hier nicht ausgewertet.

PROC    M8P0380_2      FAR

        pusha
        push    ds
        push    es
        mov     ax,cs                  ; Tiny-Model: ds auf Anfang setzen
        mov     ds,ax
        xor     ax,ax
        mov     es,ax
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        call    [DWORD es:LED_ON]
        mov     [BYTE ds:ZULED],1      ; neuen Zustand der LED speichern
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        pop     es
        pop     ds
        popa
        ret

ENDP    M8P0380_2


;-----------------------------------------------------------------------------
;Prozedur 3: LED OFF
;-----------------------------------------------------------------------------
;Vom Betriebssystem wird in dx bei Aufruf einer Prozedur immer die
;Task-Nr uebergeben, das wird aber hier nicht ausgewertet.

PROC    M8P0380_3      FAR

        pusha
        push    ds
        push    es
        mov     ax,cs                          ; Tiny-Model: ds auf Anfang setzen
        mov     ds,ax
        xor     ax,ax
        mov     es,ax
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        call    [DWORD es:LED_OFF]
        mov     [BYTE ds:ZULED],0              ; neuen Zustand der LED speichern
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
        pop     es
        pop     ds
        popa
        ret

ENDP    M8P0380_3

;-----------------------------------------------------------------------------
END M8P0380
;-------------------------- END OF M8P0380.ASM -------------------------------
```

11. Die PC-Schnittstelle

11.1. Funktion

Die folgenden Angaben gelten ohne Einschränkung für den PC, PC-XT, PC-AT, AT-386, AT-486, AT-Pentium (II, Pro) und kompatible Rechner. Sie sind nur für jene interessant, die eigene PC-Treiber schreiben wollen.

Die Kommunikation zwischen beiden Rechnern geschieht standardmäßig über eine parallele Schnittstelle mit den entsprechenden Steuersignalen, sowohl für die Übertragung vom PC zur MODULAR-4 Karte als auch für die umgekehrte Richtung. Somit können Befehle und Daten in beiden Richtungen ausgetauscht werden.

Nach dem Einschalten des PC und nach einem Reset ist die MODULAR-4 Karte bereit, Befehle und Daten vom PC zu empfangen. Die Kommunikation zwischen PC und Karte geschieht ausschließlich über Makro-Befehle. Dabei ist der PC grundsätzlich der Master, die Karte Slave. Das bedeutet, jede Kommunikation wird vom PC aus eingeleitet.

Die einzige Möglichkeit für die Karte, selbst aktiv zu werden, besteht darin, daß sie einen sog. Service-Request zum PC sendet. Dies tut sie, indem sie ein Wort (2 Byte) zum PC sendet (mit gesetztem DLP-Bit im Statusregister). Dieses Wort wird üblicherweise (aber nicht zwingend) per Interrupt auf dem PC in Empfang genommen. Aus dem Wort erkennt der PC die Ursache des Service-Requests der Karte und kann entsprechend reagieren.

Eine MODULAR-4/486 Karte belegt 4 aufeinanderfolgende Adressen im I/O-Adreßbereich des PC. Die Basisadresse (im folgenden mit "PBA" bezeichnet) wird auf der Karte mit Steckbrücken (Jumper) bzw. per Drehschalter eingestellt.

Die Übergabe von Daten vom PC an die MODULAR-4 Karte geschieht wortweise, nachdem durch Lesen des Status-Registers festgestellt wurde, daß die Schnittstelle frei ist. Das 16-Bit Wort wird in der Schnittstelle zwischengespeichert, gleichzeitig wird ein Bit im Status-Register gesetzt, dessen Zustand sowohl von der MODULAR-4 Karte als auch vom PC gelesen werden kann. Wenn das Wort von der MODULAR-4 Karte ausgelesen wird, wird dieses Status-Bit wieder zurückgesetzt. Die Schnittstelle ist damit wieder frei zur Übertragung des nächsten Wortes.

Das Betriebssystem auf der MODULAR-4 Karte versteht das Low Byte (= 1. Byte) des ersten Wortes einer Übertragung als Befehlscode. Daraus ergibt sich der Typ des Befehls. Das High Byte (= 2. Byte) des ersten Wortes ist ein Format-Byte und gibt einen Hinweis auf die Anzahl der noch folgenden Byte des Befehls und ob der Befehl eine Antwort von der Karte erwartet. Auch das Format der Antwort ist im Format-Byte codiert. Nachdem der gesamte Befehl übertragen ist, wird überprüft, ob er gültig ist. Ist das nicht der Fall, erfolgt eine Fehlermeldung durch Senden eines Codes (ein Wort) zurück an den PC. Ist der Befehl gültig, wird er ausgeführt. Falls bei der Ausführung ein Fehler auftritt, wird ebenfalls eine Fehlermeldung (ein Wort) zum PC zurückgesendet.

Wurden durch den Befehl Daten von der MODULAR-4 Karte angefordert, so ist der Ablauf der gleiche wie gerade beschrieben. Im Falle einer Fehlermeldung ist der Befehl damit abgeschlossen, es werden also keine evtl. fehlerhaften Daten von der MODULAR-4 Karte zurückgegeben. Konnte der Befehl erfolgreich ausgeführt werden, dann wird zunächst der ursprüngliche Befehlscode (das Low Byte des ersten Wortes des Befehls) und danach die angeforderten Daten zum PC zurückgesendet.

11.2. Die I/O-Adressen aus der Sicht des PC

Die Basisadresse (PBA) für die MODULAR-4 Karte wird mit Steckbrücken bzw. per Drehschalter eingestellt. Die Angaben in der Spalte "Zugriff" zeigen die erlaubten Zugriffe an, ein angehängtes "x" bedeutet, daß die gelesenen bzw. geschriebenen Daten ungültig bzw. ohne Bedeutung sind:

8 = nur Byte-Zugriff erlaubt
16 = nur Wort-Zugriff erlaubt
W = nur Schreibzugriffe erlaubt
R = nur Lesezugriffe erlaubt
RW = Schreib- und Lesezugriffe erlaubt
x = gelesene bzw. geschriebene Daten sind ohne Bedeutung

I/O-Adr.	Zugriff	Erklärung zu Data
PBA+0	R8	**Status der PC-Schnittstelle lesen** (Erklärung siehe nächste Seite)
PBA+0	W8x	**Rücksetzen von DLM** und **RESTART**[1] (DLM=0)
PBA+1	W8x	**Setzen von DLM** (DLM=1)
PBA+2	R16	**Lesen von Daten**, die von der MODULAR-4 Karte zum PC gesendet wurden. Gleichzeitig wird im Status-Register Bit 7 (RBF) = 0 gesetzt. DLM bleibt unverändert.
PBA+2	W16	**Schreiben von Daten an die MODULAR-4 Karte** Gleichzeitig wird im Status-Register Bit-0 (TBF) =1 gesetzt. Wenn die Karte das Datenwort liest, wird Bit 0 wieder = 0 gesetzt. DLM und DLP bleiben unverändert.
PBA+3	R8	Kartentyp[1] (Bit 4-0) und Revision (Bit 7-5)
PBA+3	W8x	**Hardware-Reset der MODULAR-4 Karte.** DLM und DLP werden = 0 gesetzt, RESTART = 1.

Die Beschreibung der Schnittstelle bezieht sich auf die Standardbestückung der MODULAR-4/486 Karten. Durch Umkonfigurierung bzw. Austausch verschiedener Bausteine kann die PC-Schnittstelle der MODULAR-4 Karten zusätzliche und andere Funktionen erhalten.

[1] Nur bei "kleiner" MODULAR-4/486

11.3. Das Status-Register (8 Bit)

Bit 0: **PC-TBF**	Status-Bit für die Schnittstelle vom PC zur MODULAR-4 Karte: 0 = Schnittstelle ist frei 1 = Schnittstelle nicht frei
Bit 1: **DLM**	Device Locking-Bit MODULAR-4: Flag, das vom PC aus gesetzt bzw. gelöscht werden kann (s.o.). Es kann bei Multi-Tasking Betrieb auf dem PC, z.B. unter UNIX, OS/2 oder Windows verwendet werden, um die MODULAR-4 Karte für andere PC-Tasks zu blockieren. Der Zustand dieses Bits kann auch von der MODULAR-4 aus abgefragt werden.
Bit 2: **RESET**	1 = Reset ist aktiv[1]
Bit 3: **BOOT**	0 = Karte hat kein EPROM[1]
Bit 4: **-**	Reserviert[1]
Bit 5: **RESTART**	1 = Zustand der Karte nach Reset[1]
Bit 6: **DLP**	Device Locking-Bit PC : Flag, das von der MODULAR-4 Karte aus gesetzt bzw. gelöscht werden kann. Es kann bei Multi-Tasking Betrieb auf der Karte verwendet werden, um die Schnittstelle zum PC für andere Tasks auf der Karte vorübergehend zu blockieren. Der Zustand dieses Bits kann also auch von der Karten-Seite abgefragt werden.
!	**Hinweis:** *Zur Zeit wird dieses Bit verwendet, um bei Meldungen von der Karte zwischen der Antwort auf einen Makro-Befehl (DLP = 0) und Service-Request (DLP = 1) oder Fehlermeldungen (DLP = 1) zu unterscheiden. Wenn DLP = 1 ist, wird das vom PC aus mit einem speziellen Makro-Befehl quittiert und dadurch wieder DLP = 0 gesetzt.*
Bit 7: **PC-RBF**	Status-Bit für die Schnittstelle von der MODULAR-4 Karte zum PC: 0 = Schnittstelle ist leer 1 = Schnittstelle enthält ein Byte/Wort
Bit 8 bis Bit 15: Zur Zeit nicht verwendet, undefinierter Zustand	

[1] Bei der "großen" MODULAR-4/486 Karte sind lediglich Bit 0, 1, 6 und 7 zur Zeit verwendet. Die übrigen Bit liefern beim Lesen ein nicht definiertes Ergebnis.

11.4. Beispiel für eine einfache Kommunikation

Die MODULAR-4/486 Karte soll den Zustand der Kontroll-LED (on-board LED) ermitteln und das Ergebnis an den PC übergeben.

(PBA = Basisadresse der MODULAR-4/486 Karte)

Schritt 1: Status-Registers lesen (I/O-Adresse = PBA+0).
Bit 0 = 1: Schritt 1 wiederholen.
Bit 0 = 0: die MODULAR-4 Karte ist bereit, ein Wort zu empfangen.

Schritt 2: Schreiben des Wortes 2033h an I/O-Adresse PBA+2
(Low Byte = 33h: Befehlscode für Lesen des Zustands der LED, High Byte = 20h: Formatbyte).

Schritt 3: Status-Register lesen und PC-RBF prüfen
Bit 7 = 0: es liegt noch keine Antwort vor, Schritt 3 wiederholen.
Bit 7 = 1: es liegt eine Antwort von der Karte vor.

Schritt 4: Bit 6 (DLP) im Status-Register = 1:
Es handelt sich um eine Fehlermeldung oder ein Service-Request von der Karte, also nicht um die Antwort auf den Makro-Befehl, weiter mit spezieller Fehlerbehandlungsroutine.

Bit 6 (DLP) im Status-Register = 0:
Es liegt die Antwort auf den Makro-Befehl vor. Weiter bei Schritt 5.

Schritt 5: Lesen und Auswerten der Antwort (I/O-Adresse = PBA+2)

Fall 1: Das Low Byte (gilt immer als erstes Byte) entspricht dem ursprünglichen Befehlscode 33h. Somit wurde der Befehl erfolgreich ausgeführt. Das High Byte der Antwort (gilt immer als zweites Byte) gibt die Antwort, also den Zustand der LED (wenn das Byte = 0 ist, leuchtet die LED nicht). Der Befehl und die Kommunikation sind damit beendet.

Fall 2: Das Byte entspricht weder dem ursprünglichen Befehlscode noch einem Fehlercode. Es liegt wahrscheinlich das Ergebnis eines früheren Befehls vor. Aus dem Byte kann der Ursprung der Daten erkannt werden.

Notizen

12. Makrobefehle

Einzelheiten über die Hardware der PC-Schnittstelle für die Kommunikation zwischen PC und MODULAR-4/486 sind in Kapitel 11 beschrieben. Alle im folgenden aufgeführten Makrobefehle sind im Betriebssystem der Karte enthalten. Sie sind nur für jene Anwender interessant, die eigene PC-Treiber schreiben wollen.

Die PC-Programm-Bibliotheken von SORCUS (im Lieferumfang enthalten, z.B. für Borland Delphi, Borland C++ oder Microsoft Visual C++) unterstützen alle Makrobefehle. Sie müssen nur noch die entsprechenden Prozeduren aufrufen und können damit die benötigten Makrobefehle sehr einfach in Ihr PC-Programm einbinden.

Makrobefehle können auch gesendet werden, während Anwenderprogramme auf der Karte laufen. Es ist Vorsicht geboten, wenn mit den Befehlen die gleichen Funktionseinheiten auf der Karte angesprochen werden, die auch von einem gerade auf der Karte laufenden Anwenderprogramm benutzt werden.

Die Kommunikation zwischen PC und Karte ist so organisiert, daß es nur die beiden folgenden Fälle geben kann:

Fall 1: Der PC fordert per Makrobefehl bestimmte Daten oder Aktionen der Karte an. Die Karte führt das aus und sendet ggfls. eine Antwort zurück. Wenn eine Antwort gefordert ist, sendet der PC erst dann den nächsten Befehl, wenn er die Antwort komplett empfangen hat. Das erste Byte des Makrobefehls und das erste Byte der Antwort sind identisch, wenn keine Fehler aufgetreten sind.

Fall 2: Die Karte kann unabhängig davon jederzeit selbst aktiv werden und per Service-Request (SRQ) den PC zu bestimmten Aktionen auffordern. Dies tut sie, indem sie ein Wort zum PC sendet (eventuell mit Interruptauslösung auf dem PC). Am gesetzten DLP-Bit im Status der Schnittstelle kann der PC unterscheiden, ob es sich um die Antwort auf einen zuvor gesendeten Makrobefehl oder um eine spontane Aktivität handelt. Das gleiche gilt, wenn Fehler auf der Karte auftreten.

Eine Kurzzusammenfassung aller Makrobefehle findet sich im Anhang G.

12.1. Das Format der Makrobefehle

Jeder Makrobefehl besteht aus mindestens 2 Byte. Das erste Byte enthält den Befehlscode, das zweite Byte enthält zwei Angaben: in den unteren 4 Bit steht ein Formatcode für den Befehl bzw. die Länge des Befehls (FCB), in den oberen 4 Bit steht ein Formatcode für die Antwort bzw. die Länge der Antwort (FCA).

Beim Formatcode für den Befehl bedeutet eine Angabe zwischen 0 und 11 die Anzahl Byte, die nach den ersten beiden Byte noch folgen. Angaben >= 12 sind Codierungen für das Format des Befehls (s.u.).

Beim Formatcode für die Antwort bedeutet eine Angabe zwischen 0 und 11 direkt die Länge der Antwort, und zwar die Gesamtzahl Byte, aus der die Antwort besteht. Bei 0 wird keine Antwort erwartet. Angaben >= 12 sind Codierungen für die Antwortlänge und das Format der Antwort. Dabei ist es auch möglich, die Länge der Antwort zunächst offen zu lassen, also nicht vorzugeben, sondern sie statt dessen von der Karte selbst ermitteln zu lassen. In einem solchen Fall wird die Länge der Antwort als Teil der Antwort zurückgegeben.

Wenn eine Antwort erwartet wird, hat sie folgenden Aufbau: Das erste Byte enthält wieder den Befehlscode, das zweite Byte kann schon das erste Byte der Nutzdaten oder die Längenangabe der Antwort sein.

Fehlermeldungen und spontane Aktivitäten der Karte (z.B. Service-Requests (SRQs) und Traps) bestehen immer nur aus zwei Byte. Eine Übersicht aller Fehlermeldungen (Fehlergruppe und Fehlertyp) finden Sie in Anhang F. Beide Bytes geben einen Hinweis auf den Grund der Aktivität, z.B. bei einem SRQ die Nummer der anfragenden Task.

Das erste Byte eines Makrobefehls wird immer im Low Byte, das zweite im High Byte übertragen, etc.

Das allgemeine Format der Makrobefehle (1 Kästchen = 1 Byte):

Befehl	CB	FCA	FCB	nL	nH	mL	mH	b1	:::	bn
Antwort	CA	s	wL	wH	vL	vH	a1	:::	ax	

Erklärung der Abkürzungen

CB	Makrobefehls-Code
FCA	Format-Code Antwort
FCB	Format-Code Befehl
n	Anzahl Nutzdatenbyte beim Befehl (nL, nH = Low Byte, High Byte von n)
m	Anzahl Nutzdatenbyte der Antwort (Vorgabe) (mL, mH = Low Byte, High Byte von m)
b1...bn	Nutzdatenbytes beim Befehl
CA	Makrobefehls-Code der Antwort
s	Statusmeldung
w	Anzahl verworfener Nutzdatenbyte des Befehls (wL, wH = Low Byte, High Byte von w)
v	Anzahl Nutzdatenbyte der Antwort (vL, vH = Low Byte, High Byte von v)
a1...ax	Nutzdatenbytes der Antwort

Hinweise zum Format:

1. Wenn bei der Ausführung des Makrobefehls kein Fehler aufgetreten ist und FCA ungleich 0 ist (also eine Antwort gefordert ist), dann ist CA = CB.
2. Wenn FCB = 0 ist, dann entfallen nL und nH und b1...bn.
3. Wenn 0 < FCB < 12 ist, dann entfallen nL und nH, n = FCB.
4. Wenn FCB = 0fh ist, dann kann n max. 65280 sein.
5. Wenn FCA = 0 ist, dann entfallen mL, mH und die gesamte Antwort (CA, s, wL, wH, vL, vH und a1...ax).
6. Wenn FCA = 1 ist, dann entfallen mL, mH, wL, wH, vL, vH und a1...ax.
7. Wenn 1 < FCA < 12 ist, dann entfallen mL, mH, s, wL, wH, vL und vH, x = FCA - 1.
8. Wenn FCA = 0fh ist, dann entfallen s, wL, wH, vL, vH, x = m.
9. Wenn FCA = 0eh ist, dann entfallen mL, mH, s, wL, wH, x = v.
10. Wenn FCB = 0dh ist, kann der Makrobefehl selbst festlegen, wie viele Nutzdatenbyte des Befehls er verwendet und wie viele er verwirft. Die Antwort enthält in jedem Fall die Felder wH, wL und s.
11. Wenn FCA = 0dh ist, entscheidet der Makrobefehl, wie viele Nutzdatenbyte er in der Antwort zurückgibt. Die Antwort enthält in jedem Fall die Felder vL, vH und s.
12. Die übrigen Werte für FCA und FCB sind reserviert.

Beispiel 1: Befehl ohne Nutzdatenbyte und ohne Antwort

CMD	0	0

Beispiel 2: Befehl mit n Nutzdatenbyte (0 < n < 12) ohne Antwort

CMD	0	n	b1	:::	bn

Beispiel 3: Befehl mit n Nutzdatenbyte (0 < n < 65280) ohne Antwort

CMD	0	0fh	nL	nH	b1	:::	bn

Beispiel 4: Befehl mit 2 Nutzdatenbyte, Antwort mit m Nutzdatenbyte (0 < m < 65280)

Befehl	CMD	0fh	2	mL	mH	b1	b2
Antwort	CMD	a1	:::	am			

Beispiel 5: Befehl ohne Nutzdatenbyte, die Karte bestimmt die Länge der Antwort mit x Nutzdatenbyte selbst (-1 < x < 65280)

Befehl	CMD	0eh	0			
Antwort	CMD	vL	vH	a1	:::	av

Beispiel 6: Befehl mit n Nutzdatenbyte (0 < n < 12),

Antwort mit m Nutzdatenbyte (0 < m < 11)

Befehl	CMD	m+1	n	b1	:::	bn
Antwort	CMD	a1	:::	am		

Beispiel 7: Befehl mit n Nutzdatenbyte (-1 < n < 65280), die Karte bestimmt die Länge der Antwort mit v Nutzdatenbyte selbst (-1 < v < 65280)

Befehl	CMD	0eh	0eh	nL	nH	b1	:::	bn
Antwort	CMD	vL	vH	a1	:::	av		

Beispiel 8: Befehl mit n Nutzdatenbyte (-1 < n < 260). Die Karte bestimmt selbst, wieviel sie davon nicht verwendet hat und meldet dies in w zurück. Bezüglich der Antwort wird im Befehl vorgegeben, wieviel Byte die Karte maximal zurückliefern darf (-1 < m < 256). Sie legt diese Anzahl selbst fest und liefert die Anzahl der tatsächlich zurückgelieferten Byte in v (v <= m). Zusätzlich erfolgt in der Antwort eine Statusmeldung (s), die dem Fehlertyp entspricht.

Befehl	CMD	0dh	0dh	nL	nH	mL	mH	b1	:::	bn
Antwort	CMD	s	wL	wH	vL	vH	a1	:::	av	

Die Befehlscodes von 00h bis 7fh sind für die Makrobefehle vorgesehen. Sie dienen dazu, das Multi-Tasking Betriebssystem bzw. die Strukturen der einzelnen Tasks anzusprechen.

Alle Parameter, wie z.B. Tasknummer, Parameter-Nummer, etc., werden in der maximal erforderlichen Länge angegeben. Bei einem Zugriff auf Parameter wird also z.B. die Parameter-Nummer als 16 Bit Zahl (2 Byte) angegeben, weil eine Task maximal 65280 Parameterbyte haben kann.

12.2. Kommunikationsbefehle PC - MODULAR-4/486

Bei "Code/Data" sind die einzelnen Byte in der Reihenfolge angegeben, wie sie zur Karte geschickt werden. Das gilt auch für solche Angaben, die aus zwei Byte bestehen, wie z.B. die Task-Nummer (tL und tH). Das erste Byte des Befehls wird im Low Byte, das zweite im High Byte, das dritte dann wieder im Low Byte, etc., gesendet.

Code/Data	Funktion
00h 20h	**Anforderung: Überprüfen, ob MODULAR-4/486 Karte bereit** Dieser Befehl sollte so oft gesendet werden, bis als Antwort 00h 00h zurückkommt. Anmerkung: Der Befehl kann sowohl bei 8 als auch bei 16 Bit breiter Schnittstelle gesendet werden. Bei 8 Bit Breite wird das zweite Byte (High-Byte) ignoriert (sowohl bei der Anforderung als auch bei der Antwort)
00h 00h	Antwort: MODULAR-4/486 Karte ist bereit für Kommunikation
01h 20h	**Anforderung: Typ der Karte und Betriebssystem melden**
01h 0xh	Antwort: Die MODULAR-4/486 Karte sendet 01h 0xh zurück, andere Karten (z.B. MODULAR-4/Z80) 04h. Das High-Byte kann bei der MODULAR-4/486 weiter ausgewertet werden. Es zeigt an, welches Betriebssystem aktiv ist: x = 0: Mini-Betriebssystem im EPROM (nach Power-On), es sind nur wenige Makrobefehle verfügbar. x = 1: Volles Betriebssystem (Kopie aus EPROM) x = 2: Volles Betriebssystem (vom aus PC geladen)

12.3. Konfigurationsbefehle

Mit diesem Makrobefehl werden verschiedene Optionen auf der Karte einstellbar sein, die den Programmablauf in Bezug auf PC-Makrobefehle regeln. Nur die beiden folgenden Formate sind z.Zt. verfügbar.

Code/Data	Funktion
1bh 02h 10h 10h	**PC-Makrobefehle unterbrechbar durch II-/DI-Tasks**
1bh 02h 00h 10h	**PC-Makrobefehle nicht unterbrechbar** (= Einstellung nach Reset)

12.4. Systemzugriffe: Speicher (privilegierte Befehle)

Code/Data	Funktion
20h 04h aE aF aG aH	**Pointer P setzen** (absolute physikal. Adresse)
25h f0h mL mH	**Anforderung: Block von (Pointer) lesen, P = P + m** m = Anzahl Byte (1 bis 65521)
25h b1...bm	Antwort
26h f4h mL mH pE p F pG pH	**Anforderung: Block lesen** m = Anzahl Byte (1 bis 65380) Pointer (32 Bit)
26h b1...bm	Antwort
21h 0fh nL nH b1...bn	**Block an (Pointer) schreiben, P = P + n** n = Anzahl Byte (1 bis 65521)
2eh 0fh nL nH pE pF pG pH b1...bn	**Block schreiben** n = Anzahl Byte - 4 (1 bis 65380) Pointer (32 Bit)
24h 02h 55h aah	**Start Programm ab Adresse des Pointers**

Code/Data	Funktion
22h 52h 00h z	**Anforderung: Freien Speicherplatz melden** Alignment: Der Anfang des freien Bereichs kann auf eine Grenze gelegt werden, deren Adresse durch eine 2er Potenz teilbar ist z=0: Byte, z=1: Wort, z=2: Doppelwort, ...
22h aE aF aG aH	Antwort: Größe des freien Speichers in Anzahl Byte
22h 9fh 0ah 00h	**Anforderung: Speicherplatz reservieren**
s	Strategie: s=0: nur Größe des freien Speichers melden s=1: von unten nach oben, soviel wie angegeben s=2: von unten nach oben, soviel wie möglich, aber nicht mehr als angegeben s=3: von oben nach unten, soviel wie angegeben s=4: von oben nach unten, soviel wie möglich, aber nicht mehr als angegeben
z	Alignment: Der Anfang des zu reservierenden Bereichs kann auf eine Grenze gelegt werden, deren Adresse durch eine 2er Potenz teilbar ist z=0: Byte, z=1: Wort, z=2: Doppelwort, ...
tL tH hL hH	t = Task-Nummer (nur für statistische Zwecke) h = Speichernutzung (nur für statistische Zwecke) (dyn., stat., Code, Daten, Parameter, etc.)
nE nF nG nH	Größe des zu reservierenden Bereichs in Anzahl Byte (s = 1, 2, 3, 4), sonst ohne Bedeutung
22h gE gF gG gH	Antwort: Größe des reservierten Bereichs (s = 1, 2, 3, 4) bzw. ohne Bedeutung (s = 0)
aE aF aG aH	Anfangsadresse des reservierten Bereichs (physikal. Adresse) (s = 1, 2, 3, 4) bzw. Größe des freien Speichers (s = 0)

12.5. Systemzugriffe: I/O (privilegierte Befehle)

Code/Data	Funktion
27h 22h pL pH	**Anforderung: Byte von 8-Bit breitem Port lesen** p = Port (I/O-Adresse)
27h b	Antwort: b = Byte
27h 32h pL pH	**Anforderung: Wort von 16-Bit breitem Port lesen** p = Port (I/O-Adresse)
27h wL wH	Antwort: w = Wort
23h 03h pL pH b	**Byte an 8-Bit breiten Port schreiben** p = Port (I/O-Adresse), b = Byte
23h 04h pL pH wL wH	**Wort an 16-Bit breiten Port schreiben** p = Port (I/O-Adresse), w = Wort

12.6. Die Taskbefehle

Erläuterung zu den Abkürzungen:

n	= Anzahl (0 bis 255)
nL nH	= Anzahl, Low Byte, High Byte (0 bis 65280)
aE aF aG aH	= 4-Byte Adresse, E = Low Byte, H = High Byte
b	= Byte
b1..bn	= n Byte (Byte 1 bis Byte n)
c	= Code für Makrobefehl
dE dF dG dH	= Datenbereich Länge, E = Low Byte, H = High Byte
f	= Flag
pL pH	= Programmnummer
(p)	= Parameter-Nummer (2 Byte): pL pH (0 bis 65280)
rL rH	= relative Adresse, Low Byte, High Byte
rE rF rG rH	= relative Adresse (4 Byte, E = Low, H = High Byte)
(t)	= Task-Nummer (2 Byte): tL tH (0 bis 1023)
vL vH	= Wort: Low Byte, High Byte
wL wH	= Wort: Low Byte, High Byte
z	= Prozedur-Nr. (0 bis 255)
sL sH	= Stations-Nr. (0 bis 65280)

12.6.1. Das Prinzip

Das Betriebssystem ordnet jeder Task, wenn ein Programm installiert wird, einen Parameter- und einen Datenbereich zu. Die Länge beider Bereiche kann auch = 0 sein. Die Länge des Parameterbereichs wird vom Programm selbst vorgegeben und ist damit konstant. Für die Angabe der Länge des Datenbereichs gibt es verschiedene Möglichkeiten, sie kann auch vom Anwender beim Installieren angegeben werden.

Das Programm selbst besteht aus allgemein aufrufbaren Prozeduren bzw. Funktionen, z.B. vom PC oder von einer anderen Task aus.

Vom PC aus kann mit Makrobefehlen auf die Strukturen (Prozeduren, Funktionen, Parameter, Daten) jeder Task zugegriffen werden: Es können Funktionen aufgerufen werden, Parameter gesetzt und gelesen werden und Daten geschrieben oder gelesen werden.

12.6.2. Datenbereich

Der Datenbereich einer Task ist immer ein durchgängiger Bereich ohne Lücken oder Überlappungen. Der Zugriff erfolgt immer indirekt über Pointer: einen Schreibpointer (W-Pointer) und einen Lesepointer (R-Pointer). Beide Pointer sind der Task, zu der der Datenbereich gehört, fest zugeordnet. Alle Tasks können Anfang und Ende des Datenbereichs jeder Task aber ermitteln und natürlich auch eigene Pointer halten.

Für das Lesen und Schreiben von Daten von anderen Tasks oder vom jeweiligen Host aus stehen folgende Funktionen zur Verfügung:

- R-Pointer auf Anfang setzen
- R-Pointer verschieben (vorwärts/rückwärts)
- W-Pointer auf Anfang setzen,
- W-Pointer verschieben (vorwärts/rückwärts)
- Daten aus Bereich lesen und R-Pointer um n inkrementieren
- Daten in Bereich schreiben und W-Pointer um n inkrementieren
- R-Pointer relativ zum Anfang des Datenbereichs setzen, Daten lesen und R-Pointer um n inkrementieren
- W-Pointer relativ zum Anfang des Datenbereichs setzen, Daten schreiben und W-Pointer um n inkrementieren

Bei einem linearen Puffer erfolgt nur beim Schreiben eine Bereichsüberprüfung.

12.6.3. Befehle zur Installierung und Taskverwaltung

12.6.3.1. Makrobefehl 40h zum Installieren

Vom PC aus muß ein Programm mit diesem Makrobefehl unter einer Task installiert werden. Hierbei sind verschiedene Optionen möglich.

Code/Data	Funktion
40h 0fh	**Installiere Programm** (Erklärung s. unten)
12h 00h	
tL th	t = Tasknummer
pL ph	p = Programmnummer
iL 00h	i = Interrupt-Nummer
fE fF fG fH	f = Flag
dE dF dG dH	d = Größe des zu reservierenden Datenbereichs
aE aF aG aH	a = Adresse

Erläuterungen zum Makrobefehl 40h zum Installieren

t = Task-Nummer (Wort)

Dies ist die Task-Nummer, unter der das Programm installiert werden soll. Erlaubt sind Angaben von 1 bis 1023. Task 0 ist immer für das Betriebssystem reserviert. Task 1 bis 15 sollten nicht für Anwendungsprogramme benutzt werden. Aus Geschwindigkeitsgründen ist es sinnvoll, zunächst Tasknummern kleiner 256 zu verwenden.

p = Programmnummer (Wort)

Die Programmnummer kann von 1 bis 65534 betragen. Die im Makrobefehl angegebene Nummer muß mit der Nummer in der PDT übereinstimmen. Programmnummer 0 ist reserviert für das Betriebssystem, Programmnummer ffffh (=-1) ist ein Dummy-Programm ohne Funktion. Sie wird gemeldet, wenn kein Programm unter einer Task installiert ist. Je nach Typ der Karte heißen die Programme M7P... ("kleine" MODULAR-4/486) oder M8P... ("große" MODULAR-4/486). Die Programmnummer erscheint auch als 4-stellige hexadezimale Zahl im Namen von Programmdateien und in dazugehörigen Dateien:

MxPnnnn.LIB = Anwenderprogramm (identisch mit .OBJ)
MxPnnnn.LAB = Relocated Anwenderprogramm
MxPnnnn.SEX = SORCUS EXE Datei (identisch mit .EXE)
MxPnnnn.SRX = SORCUS Relocated EXE Datei
MxPnnnn.SHX = SORCUS Hypertext Datei

i = Interrupt-Nummer (Byte), siehe Anhang

Bei NI-Tasks wird diese Angabe nicht ausgewertet, nur unter bestimmten Voraussetzungen bei Interrupt-Tasks (II- und DI-Tasks). Die Interrupt-Nummer legt fest, welcher Interrupt der Task, die gerade installiert wird, zugeordnet wird. Wenn dieser Interrupt auftritt, wird die Hauptprozedur des Programms aufgerufen, das unter der Task installiert ist.

Für die Festlegung des Interrupts gibt es zwei Möglichkeiten, durch die PDT (= Programm Deskriptor Tabelle) oder mit Makrobefehl 40h zum Installieren (siehe auch bei "Task-Typ"). Hierfür sind zwei Bit zuständig: Bit 3 im Flag in der PDT und Bit 3 im Flag im Makrobefehl. Dies sind die gleichen Bit, die auch für die Festlegung des Task-Typs verwendet werden.

f = Flag (Doppelwort = 32 Bit)

Die folgende Tabelle gibt eine Kurzerklärung für die Verwendung der 32 Bit im Flag.

Bit-Nr.	Anzahl Bit	Erklärung
0 - 2	3	Task-Typ (NI-, II-, DI-, TI-Task)
3	1	Wer legt Task-Typ und Interrupt-Nr. fest?
4, 5	2	Privilegstufe des Programms
6 - 8	3	Programm im RAM oder ROM?, in welchem Format?
9, 10	2	Wie wird Größe von Datenbereich festgelegt?
11	1	Auto-Init-Prozedur nach Installieren aufrufen?
12	1	Task sofort nach Installieren aktivieren?
13 - 31	19	Reserviert für SORCUS

Task-Typ: NI-, DI-, II-, TI-Task (Bit 0 bis 2)

Hiermit wird angegeben, welchen Typ die installierte Task haben soll:

0 (=000b): NI-Task (Nicht-Interrupt-Task)
1 (=001b): II-Task (Indirekte Interrupt-Task)
2 (=010b): DI-Task (Direkte Interrupt-Task)
3 (=011b): TI-Task (Timer-Initiierte-Task)

Für die Festlegung des Task-Typs (und der Interrupt-Nummer) gibt es zwei Möglichkeiten, entweder durch das zu installierende Programm selbst oder mit Makrobefehl 40h zum Installieren. Hierfür sind zwei Bit zuständig: Bit 3 im Flag in der PDT und Bit 3 im Flag im Makrobefehl. Diese Bit werden auch für die Festlegung der Interrupt-Nummer verwendet.

Wer legt Task-Typ und Interrupt-Nummer fest? (Bit 3)

Wenn Bit 3 im Flag der PDT = 1 gesetzt ist, wird dieses Bit ignoriert und Task-Typ und Interrupt-Nummer werden in jedem Fall aus der PDT verwendet. Die Angaben "Task-Typ" und "Interrupt-Nummer" im Makrobefehl werden verworfen.

Wenn Bit 3 im Flag der PDT = 0 gesetzt ist, erlaubt das Programm, beide Angaben auch per Makrobefehl festzulegen. Hierzu muß dieses Bit = 1 gesetzt werden. Natürlich müssen die Angaben von Task-Typ und Interrupt-Nummer gültige Werte aufweisen. Wenn dieses Bit = 0 ist, werden wiederum die Angaben für Task-Typ und Interrupt-Nummer aus der PDT verwendet.

Privilegstufe des Programms (Bit 4 und 5)

0 (=00b) ist die höchste Privilegstufe, 3 (=11b) die niedrigste. Anwendungsprogramme haben immer die Privilegstufe 3, Systemprogramme die höchste, also 0. Diese Flags werden bei MODULAR-4/486 nicht ausgewertet und können immer = 0 gesetzt werden.

Programm im ROM oder RAM und Programmformat (Bit 6 bis 8)

Wenn das zu installierende Programm im RAM steht, wird die Adresse a als absolute physikalische Adresse angesehen, ihre Bedeutung richtet sich danach, in welchem Format das Programm im RAM der Karte steht. Dies wird dem Betriebssystem über Bit 6 bis 8 im Flag des Makrobefehls mitgeteilt:

Wenn das zu installierende Programm im ROM vorhanden ist, wird die im Makrobefehl angegebene Adresse a ignoriert, entscheidend ist dann die im Makrobefehl angegebene Programmnummer.

Wo?	**Programmformat**	**Adresse a**	**Format Adresse**	**Flag-Bit* 8 7 6**
RAM	PDT (tiny)	Anfang PDT	physikal.	0 0 0
RAM	PDT (large)	Anfang PDT	physikal.	1 0 0
RAM	EXE nicht relocated	Anfang EXE-Header	physikal.	1 1 0
RAM	EXE relocated	PREPARE-Code	Seg:Offs.	0 1 0
ROM	PDT	wird ignoriert	-	0 0 1

* alle anderen Kombinationen sind zur Zeit nicht erlaubt

Wie wird die Größe des Datenbereichs festgelegt? (Bit 9 und 10)

Hierfür gibt es mehrere Möglichkeiten (siehe folgende Tabelle). Wenn Bit 9 im Flag der PDT = 1 gesetzt ist, kann die Größe durch den Makrobefehl nicht verändert werden. Der Normalfall ist der, daß die Größe durch d im Makrobefehl beim Installieren angegeben wird. Die maximal und minimal mögliche Größe kann durch die Vorgaben in der PDT eingeschränkt werden, falls ein Programm z.B. nur einen max. 64 K großen Datenbereich unterstützt.

		Flag in PDT	**Flag in Makrobefehl**	
fix/variabel	**Größe richtet sich nach**	**Bit 9**	**Bit 10**	**Bit 9**
fix	"Größe" in PDT	1	x	x
fix	"Größe" in PDT	0	1	1
fix	"Minimum" in PDT	0	1	0
fix	"Maximum" in PDT	0	0	1
variabel	d in Makrobefehl	0	0	0

Auto-Init aufrufen? (Bit 11)

Wenn dieses Bit = 1 gesetzt ist, wird unmittelbar nach dem Installieren sofort die Auto-Init Prozedur (Prozedur 1) aufgerufen. Wenn das Bit = 0 gesetzt ist, nicht.

Task nach Installieren sofort aktivieren? (Bit 12)

Wenn dieses Bit = 1 gesetzt ist, wird nach dem Installieren und ggfls. nach dem Aufruf der Auto-Init Prozedur die Task auch sofort aktiviert. Wenn das Bit = 0 gesetzt ist, nicht.

Reserve (Bit 13 bis 31)

d = Größe Datenbereich (Doppelwort)

Die Größe wird in Anzahl Byte angegeben. Weitere Erklärungen finden sich beim Flag zu Bit 9 und 10 (s.o.).

a = Adresse (Doppelwort)

Die Adresse wird als physikalische oder Segment:Offset Adresse angegeben, siehe Flag, Bit 6 bis 8. Bei Verwendung der mitgelieferten PC Bibliothek wird die Formatanpassung automatisch übernommen.

12.6.3.2. Makrobefehle zur Taskverwaltung

Code/Data	Funktion
38h 22h pL pH	**Anforderung: Melde, ob Programm im ROM vorhanden** pL, pH = Nr. des Programms
38h f	Antwort: Befehlscode, Flag Wenn f = 0, dann ist das Programm nicht im ROM.
3ah 31h i	**Anforderung: Melde die Nummer der Task, die den Interrupt i nutzt** i = Interrupt-Nr.
3ah tL tH	Antwort: Befehlscode, Task-Nr. (wenn t = -1 (ffffh), dann ist der Interrupt i nicht benutzt)
4eh 64h nL nH pL pH	**Melde die Task, unter der Programm p installiert ist** n = Ordnungszahl der gesuchten Installierung p = Programmnummer
4eh s mL mH tL tH	Antwort: Status: s=0: alles ok, s=-1 (ffh) Programm ist nicht installiert m = Ordnungszahl einer gefundenen Installierung (m ≤ n) t = Tasknummer der m. ten Installierung

Code/Data	Funktion
41h 02h tL tH	**Aktivieren einer Task** (außer TI-Task) tL, tH = Task-Nr. Eine NI-Task kann auch mehrfach aktiviert werden. Dadurch wird das installierte Programm häufiger aufgerufen. Bei II-Tasks und DI-Tasks wird der zugehörige Interrupt demaskiert. Dieser Befehl steht auch im ROM Mini-Betriebssystem zur Verfügung. Wenn als Task-Nr. 0 angegeben wird, wird ein voll funktionsfähiges Betriebsystem aus dem ROM der Karte ins RAM der Karte kopiert und aktiviert.
41h 0fh 10h 00h tL tH p 00h zE zF zG zH iE iF iG iH nE nF nG nH	**Aktivieren einer TI-Task** (andere Tasks s.o.) t = Task-Nr. p = Prioritätszahl (0 = höchst Priorität) z = Hold-Off Zeit vom Aktivieren bis zum 1. Aufruf der Task (in Anzahl Timer-Tics) i = Intervall zwischen 2 Aufrufen der Task (in Anzahl Timer-Tics) n = Anzahl Aufrufe, bis die Task automatisch wieder deakti viert wird (0 = unendlich)
42h 02h tL tH	**Deaktivieren einer Task** tL, tH = Task-Nr. (Task 0 ist nicht erlaubt) Eine mehrfach aktivierte NI-Task muß auch mehrfach wieder deaktiviert werden. Ein laufendes Programm wird abgebrochen, ein gerade laufender Aufruf wird ordnungsgemäß beendet. Bei II-Tasks und DI-Tasks wird der zugehörige Interrupt maskiert.
43h 22h tL tH	**Anforderung: Melde, ob Task t aktiviert ist** tL, tH = Task-Nr.
43h n	Antwort: n = Anzahl Aktivierungen (n = 0: Task ist nicht aktiviert)

12.6.4. Zugriff auf Parameter

Während des Zugriffs sind die Parameter vor dem gleichzeitigen Zugriff z.B. durch eine andere Task oder durch den PC geschützt. Um auf einzelne Parameter (Byte, Wort, Doppelwort) einer Task konsistent zugreifen zu können, können die bestehenden Funktionen Byte lesen/schreiben, Wort lesen/schreiben und Doppelwort lesen/schreiben eingesetzt werden. Die Funktionen zum Blocklesen (Makrobefehle 4ch und 4dh) dagegen sind durch Interrupts unterbrechbar, Konsistenz ist nicht mehr garantiert.

Mit dem Makrobefehl 3ch kann sich eine Task eine Semaphore eines Parameterbereichs einer anderen oder auch der eigenen Task holen, um einen konsistenten Zugriff zu gewährleisten. Mit Makrobefehl 3dh gibt sie die Semaphore wieder frei. Je Task steht eine Semaphore für den Parameterbereich zur Verfügung, die automatisch verwaltet wird. Alle Tasks, die auf den Parameterbereich derselben Task zugreifen wollen, müssen immer zunächst versuchen, sich die zugehörige Semaphore zu holen. Die Task, die die Semaphore bekommen hat, kann dann mit allen angegebenen Funktionen konsistent auf den Parameterbereich zugreifen. Danach muß sie sie wieder freigeben. Die einzelnen Zugriffe selbst kümmern sich um die Semaphore nicht, gleichgültig, ob auf einzelne Parameter, auf einen Block oder ob direkt über die Adresse von Parametern zugegriffen wird.

Code/Data	Funktion
3ch 22h tL tH	**Anforderung: Parameter-Semaphore anfordern** t = Task-Nummer des Parameterbereichs
3ch 04h sL sH	Antwort: s = 0: Semaphore bekommen, s <> 0: Semaphore nicht bekommen
3dh 02h tL tH	**Parameter-Semaphore freigeben** t = Task-Nummer des Parameterbereichs
3dh 02h	Antwort
58h 24h tL tH pL pH	**Anforderung: Parameter Byte lesen** t = Task-Nummer p = Nummer des zu lesenden Parameters
58h b	Antwort: b = gelesenes Parameter Byte

Code/Data	Funktion
59h 34h tL tH pL pH	**Anforderung: Parameter Wort lesen** t = Task-Nummer p = Nummer des zu lesenden Parameters
59h wL wH	Antwort: w = gelesenes Parameter Wort
5ah 54h tL tH pL pH	**Anforderung: Parameter Doppelwort lesen** t = Task-Nummer p = Nummer des zu lesenden Parameters
5ah dE dF dG dH	Antwort: d = gelesenes Parameter Doppelwort
4ch f4h mL mH tL tH pL pH	**Anforderung: Parameter Block lesen** m = Anzahl zu lesender Parameterbyte t = Task-Nummer p = Nummer des ersten zu lesenden Parameters
4ch b1...bm	Antwort: Data
5ch 05h tL tH pL pH b	**Parameter Byte schreiben** t = Task-Nummer p = Nummer des zu schreibenden Parameters b = Parameter Byte
5dh 06h tL tH pL pH wL wH	**Parameter Wort schreiben** t = Task-Nummer p = Nummer des zu schreibenden Parameters w = Parameter Wort
5eh 08h tL tH pL pH dE dF dG dH	**Parameter Doppelwort schreiben** t = Task-Nummer p = Nummer des zu schreibenden Parameters d = Parameter Doppelwort
4dh 0fh nL nH tL tH pL pH b1...bn-4	**Parameter Block schreiben** n = Anzahl zu schreibender Parameterbyte + 4 t = Task-Nummer p = Nummer des ersten zu schreibenden Parameters Data

12.6.5. Aufruf einer Prozedur bzw. Funktion

Beim Aufruf einer **Prozedur** einer Task werden keine Parameter an sie übergeben und keine von ihr zurückgeliefert.

Beim Aufruf einer **Funktion** können Parameter hin und zurück übergeben werden (Anzahl Hin und Rück werden immer als Anzahl Byte angegeben und können auch = 0 sein):

Bezüglich der an die Funktion übergebenen Parameter meldet die Funktion zurück, wieviel Byte sie davon verwerten konnte. Sie beginnt mit der Verwertung sequentiell ab dem zuerst übergebenen Parameterbyte.

Beim Aufruf kann außerdem angegeben werden, wieviele Parameter (Anzahl Byte) von der Funktion maximal zurückerwartet werden. Die Funktion meldet immer zurück, wieviel Byte sie tatsächlich liefern konnte.

Code/Data	Funktion
45h 04h tL tH fL fH	**Aufruf der Prozedur f einer Task** (es werden keine Parameter an die Prozedur übergeben oder von ihr zurückerwartet)
44h ddh nL nH mL mH tL tH fL fH b1...bn-4	**Aufruf der Funktion f einer Task** Anzahl Byte Hin (= Anzahl der an die Funktion übergebenen Parameterbyte + 4), max. 256 + 4 Anzahl der max. zurückerwarteten Byte, max. 256 Task-Nr. Funktions-Nr. Parameter, die an die Funktion übergeben werden
44h error wL wH vL vH b1...bv	Antwort: error = 0: kein Fehler[1] Anzahl der verworfenen Parameterbyte Hin Anzahl der tatsächlich zurückgelieferten Byte Parameter, die von der Funktion zurückgeliefert werden

12

[1] Die in error übergebenen Fehlercodes entsprechen den übrigen Fehlermeldungen, allerdings wird hier nur der Fehler-Typ gemeldet (siehe Anhang F), sofern die von der aufgerufenen Funktion gelieferte Fehlermeldung der Gruppe e0h ("Fehler bei Aufruf der Funktion") angehört, andernfalls wird error = 1ah ("unbekannter Fehler") gemeldet.

12.6.6. Zugriff auf Datenbereich

Der Zugriff auf den Datenbereich einer Task ist nur indirekt über Daten-Pointer möglich. Für den Zugriff vom PC aus werden die taskeigenen Pointer (je ein R-Pointer und W-Pointer pro Task) der angesprochenen Task verwendet. Dabei ist zu beachten, daß die gleichen Pointer auch von den lokalen System-Subroutinen auf der Karte verwendet werden (siehe Kapitel 9 und 10).

Der Datenbereich ist während des Zugriffs mit den Blockbefehlen nicht vor dem gleichzeitigen Zugriff durch eine andere Task geschützt, z.B. durch eine Interrupt-Task. Umgekehrt kann der PC-Makrobefehl eine andere Task unterbrechen, die ebenfalls gerade auf denselben Datenbereich oder einen der Pointer zugreift, Konsistenz ist nicht mehr garantiert.

Mit dem Makrobefehl 3eh kann sich eine Task eine Semaphore eines Datenbereichs einer anderen oder auch der eigenen Task holen, um einen konsistenten Zugriff zu gewährleisten. Mit Makrobefehl 3fh gibt sie die Semaphore wieder frei. Je Task steht eine Semaphore für den Datenbereich zur Verfügung, die automatisch verwaltet wird. Alle Tasks, die auf den Datenbereich derselben Task zugreifen wollen, müssen immer zunächst versuchen, sich die zugehörige Semaphore zu holen. Die Task, die die Semaphore bekommen hat, kann dann mit allen angegebenen Funktionen konsistent auf den Datenbereich zugreifen. Danach muß sie sie wieder freigeben. Die einzelnen Zugriffe selbst kümmern sich um die Semaphore nicht, gleichgültig, ob auf einzelne Daten, auf einen Block oder ob direkt über die Adresse von Daten zugegriffen wird.

Code/Data	Funktion
3eh 22h tL tH	**Anforderung: Datenbereichs-Semaphore anfordern** t = Task-Nummer des Datenbereichs
3eh 04h sL sHb	Antwort: s = 0: Semaphore bekommen, s <> 0: Semaphore nicht bekommen
3fh 02h tL tH	**Datenbereichs-Semaphore freigeben** t = Task-Nummer des Datenbereichs
3fh 02h	Antwort
46h 02h tL tH	**Setze R-Pointer auf den Anfang des Datenbereichs** t = Task-Nummer

Code/Data	Funktion
47h 06h tL tH nE nF nG nH	**Verschiebe R-Pointer um n Byte** t = Task-Nummer n = Anzahl Byte (2er Komplement)
48h 02h tL tH	**Setze W-Pointer auf den Anfang des Datenbereichs** t = Task-Nummer
49h 06h tL tH nE nF nG nH	**Verschiebe W-Pointer um n Byte** t = Task-Nummer n = Anzahl Byte (2er Komplement)
50h 22h tL tH	**Anforderung: Lies Datenbyte** von (R-Pointer) und inkrementiere danach R-Pointer um 1 t = Task-Nummer
50h b	Antwort: b = Datenbyte
51h 32h tL tH	**Anforderung: Lies Datenwort** von (R-Pointer) und inkrementiere danach R-Pointer um 2 t = Task-Nummer
51h wL wH	Antwort: w = Datenwort
52h 52h tL tH	**Anforderung: Lies Daten-Doppelwort** von (R-Pointer) und inkrementiere danach R-Pointer um 4 t = Task-Nummer
52h dE dF dG dH	Antwort: d = Daten-Doppelwort
4ah f2h mL mH tL tH	**Anforderung: Lies Datenblock** von (R-Pointer) und inkrementiere danach R-Pointer um m t = Task-Nummer
4ah b1...bm	Antwort: Datenbyte (Blockgröße = m Byte)

Code/Data	Funktion
53h f6h mL mH tL tH oE oF oG oH	**Anforderung: Setze R-Pointer und lies Datenblock** m = Blockgröße (0 bis 65535) t = Task-Nummer o = Offset zum Beginn des Datenbereichs Anschließend steht der R-Pointer auf o + m.
53h b1...bm	Antwort: Datenbyte (Blockgröße = m Byte)
54h 03h tL tH b	**Schreibe Datenbyte** an (W-Pointer) und inkrementiere danach W-Pointer um 1 t = Task-Nummer b = Datenbyte
55h 04h tL tH wL wH	**Schreibe Datenwort** an (W-Pointer) und inkrementiere danach W-Pointer um 2 t = Task-Nummer w = Datenwort
56h 06h tL tH dE dF dG dH	**Schreibe Daten-Doppelwort** an (W-Pointer) und inkrementiere danach W-Pointer um 4 t = Task-Nummer d = Daten-Doppelwort
4bh 0fh nL nH tL tH b1...bn-2	**Schreibe Datenblock** an (W-Pointer) und inkrementiere danach W-Pointer um (n - 2) n = Anzahl zu schreibender Bytes +2 t = Task-Nummer (Blockgröße = n - 2 Byte)
57h 0fh nL nH tL tH oE oF oG oH b1...bn-6	**Setze W-Pointer und schreibe Datenblock** Blockgröße = n - 6 Byte t = Task-Nummer o = Offset zum Anfang des Datenbereichs Der W-Pointer steht anschließend auf o + (n - 6).

12.7. System-Call: Taskinformationen (privilegierter Befehl)

Code/Data	Funktion
2fh 54h tL tH nL nH	**Anforderung: Task-Info** Task-Nr. (t = 0: Betriebssystem) Call-Nr. (Erklärung siehe Anhang H)
2fh aE aF aG aH	Antwort: 4 Byte Ergebnis, z.B. Adresse

12.8. Zugriff auf Puffer

Erläuterungen für das Arbeiten mit den Puffern finden Sie in Kapitel 5. Der Zugriff auf die Puffer vom PC aus ist gleichwertig mit dem Zugriff von irgendeiner Task aus.

Code/Data	Funktion
60h afh 0ah 00h s	**Lege einen Puffer im lokalen Speicher an** Strategie: s=1: von unten nach oben, soviel wie angegeben s=2: von unten nach oben, soviel wie möglich, aber nicht mehr als angegeben s=3: von oben nach unten, soviel wie angegeben s=4: von oben nach unten, soviel wie möglich, aber nicht mehr als angegeben
z	Alignment: Der Anfang des Puffers kann auf eine Grenze gelegt werden, deren Adresse durch eine 2er Potenz teilbar ist: z=0: Byte, z=1: Wort, z=2: Doppelwort, ...
tL tH hL hH nE nF nG nH	t = Task-Nummer (nur für statistische Zwecke) h = Speichernutzung (nur für statistische Zwecke) n = Größe des zu reservierenden Puffers (Sollgröße in Anzahl Byte)
60h s pE pF pG pH gE gF gG gH	Antwort: s = Status: s=0: alles ok, s<>0: Fehlermeldung p = Puffer-Nr. g = Größe des Puffers (Ist-Größe in Anzahl Byte)

Code/Data	Funktion
61h 04h pE pF pG pH	**Entferne einen Puffer aus dem Speicher der Karte** p = Puffer-Nr. ***(noch nicht implementiert!)***
62h 04h pE pF pG pH	**Lösche den Inhalt eines Puffers** p = Puffer-Nr.
63h a4h pE pF pG pH	**Melde den Status eines Puffers** p = Puffer-Nr.
63h s nE nF nG nH eE eF eG eH	Antwort: s = Status: s=0: alles ok, s<>0: Fehlermeldung n = Anzahl gültiger Zeichen im Puffer e = freier Platz im Puffer (in Anzahl Byte)
6ch 60h	**Melde Status des zuletzt vom PC angesprochenen Puffers**
6ch s nL nH eL eH	Antwort: s = Status: s=0: alles ok, s>0: Fehler z.B. s = 28h: bisher wurde noch kein Puffer vom PC aus angesprochen. n = Anzahl gültiger Zeichen im Puffer. Wenn n = ffffh, dann sind mindestens 65535 Byte im Puffer. e = freier Platz im Puffer (in Anzahl Byte). Wenn e = ffffh, dann sind mindestens 65535 Byte frei.
64h 34h pE pF pG pH	**Lies Byte aus Puffer** p = Puffer-Nr.
64h s b	Antwort: Status: s=0: kein Fehler, alles ok s=2: falscher Parameter bei Aufruf s=22h: Puffer wird gerade beschrieben s=23h: Puffer wird gerade gelesen s=25h: nicht genug Zeichen im Puffer b = Datenbyte
65h 44h pE pF pG pH	**Lies Wort aus Puffer** p = Puffer-Nr.
64h s wL wH	Antwort: Status: (siehe oben) w = Datenwort

Code/Data	Funktion
66h 64h pE pF pG pH	**Lies Doppelwort aus Puffer** p = Puffer-Nr.
66h s dE dF dG dH	Antwort: Status: (siehe oben) d = Daten-Doppelwort
67h d6h mL mH s 00h	**Lies bzw. kopiere Block aus Puffer** m = max. Größe des Blocks + 4 (in Anzahl Byte) s = Strategie: Bit 0 = 0: "alles oder nichts" Bit 0 = 1: "soviel wie möglich" Bit 1 = 0: Block aus Puffer auslesen Bit 1 = 1: Block kopieren (bleibt im Puffer) Bit 2 = 0: wenn es nicht sofort klappt, abbrechen Bit 2 = 1: mehrmals probieren bis es klappt
pE pF pG pH	p = Puffer-Nr.
67h s vL vH a1 ... av	Antwort: s = Status (s.o.) v = tatsächliche Größe des gelesenen/kopierten Blocks (Anzahl Byte) a1 bis av = Byte
68h 25h pE pF pG pH b	**Schreibe Byte in Puffer** p = Puffer-Nr. b = Datenbyte
68h s	Antwort: Status: s=00h: kein Fehler, alles ok s=02h: falscher Parameter bei Aufruf s=22h: Puffer wird gerade beschrieben s=23h: Puffer wird gerade gelesen s=24h: nicht genug freier Platz im Puffer
69h 26h pE pF pG pH wL wH	**Schreibe Wort in Puffer** p = Puffer-Nr. w = Datenwort
69h s	Antwort: Status: (siehe oben)

Code/Data	Funktion
6ah 28h pE pF pG pH dE dF dG dH	**Schreibe Doppelwort in Puffer** p = Puffer-Nr. d = Daten-Doppelwort
6ah s	Antwort: Status: (siehe oben)
6bh 4dh nL nH s 00h	**Schreibe Block in Puffer** n = Größe des Blocks + 6 (in Anzahl Byte) s = Strategie: Bit 0 = 0: "alles oder nichts" Bit 0 = 1: "soviel wie möglich" Bit 1: reserviert, immer = 0 setzen Bit 2 = 0: wenn es nicht sofort klappt, abbrechen Bit 2 = 1: mehrmals probieren bis es klappt
pE pF pG pH b1...bn-6	p = Puffer-Nr. b1 bis bn - 6 = Byte
6bh s wL wH	Antwort: s = Status (s.o.) w = Anzahl nicht verwendeter Byte des Blocks

12.9. EEPROM und Kopie davon

Wenn das EEPROM mit den unten angegebenen Makrobefehlen direkt angesprochen wird, dann darf keine Task auf der Karte aktiv sein. Deshalb muß vor einem direkten Zugriff auf das EEPROM ein Reset der Karte gemacht werden. Wenn das EEPROM neu beschrieben wurde, wird die neue Information erst nach einem Hardware-Reset der Karte vom Betriebssystem verwendet bzw. steht im Parameterbereich des Betriebssystems zur Verfügung, so daß nach dem Schreiben noch einmal ein Reset der Karte ausgeführt werden sollte.

Code/Data	Funktion
28h 32h	**Lesen eines Wortes direkt aus dem EEPROM**
m	Modulsteckplatz (m = 1 bis 10) oder Basiskarte (m = 0)[1]
a	Rel. Adresse des zu lesenden Wortes im EEPROM (a = 0 bis 31)
28h	Antwort:
wL wH	EEPROM-Wort
29h 04h	**Schreiben eines Wortes direkt in das EEPROM**
m	Modulsteckplatz (m = 1 bis 10) oder Basiskarte (m = 0)[1]
a	Rel. Adresse des zu lesenden Wortes im EEPROM (a = 0 bis 31)
wL wH	Zu schreibendes Wort

Die gesamte EEPROM-Information steht nach einem Reset im Parameterbereich des Betriebssystems zur Verfügung, sofern das Lesen des EEPROMs ohne Probleme erfolgte. Dafür ist ab Parameter 100 ein Block von 4 Byte vorgesehen, der die EEPROM-Information beschreibt. Die ersten beiden Byte enthalten die Anzahl der gültigen Wörter, die vom Betriebssystem aus dem EEPROM gelesen wurden, die zweiten enthalten die Parameter-Nummer, ab der der EEPROM-Inhalt selbst steht. Wenn die Anzahl = 0 ist, konnte das EEPROM nicht gelesen werden, statt der Parameter-Nr. wird dann eine Fehlerinformation geliefert (siehe Übersicht Fehlermeldungen im Anhang F).

[1] Bei der "kleinen" MODULAR-4/486 stehen nur die Modulsteckplätze 1 und 2 zur Verfügung

12.10. Echtzeit-Uhr

Die Uhr zeigt die Zeit in Sekunden, Minuten und Stunden an. Sie kann durch Setzen des Reset-Bit im Status auf 1 genau gestellt werden, dabei wird der Subsekundenzähler auf 0 gesetzt. Das Datum wird mit Tag, Monat, Jahr und Wochentag ausgegeben. Schaltjahre werden automatisch berücksichtigt. Der Wochentag wird mit 0 = Sonntag bis 6 = Samstag ausgegeben.

Zusätzlich verfügt die Uhr über einen Impulsausgang, der mit Interrupt-Eingang IRQ-4 des Slave-Interrupt-Controllers verbunden ist. Die Uhr kann also zusätzlich als Timer verwendet werden (Timer-D). Zwischen vier Impulsfrequenzen kann gewählt werden: 1/15,625 ms, 1/sec, 1/min oder 1/h. Die Impulsdauer (1-0-1) beträgt unabhängig von der eingestellten Frequenz immer 7,8125 msec. Im Statusregister zeigt Bit 6 (PULS) den invertierten Zustand des Impulsausgangs an.

Die Uhr muß mit einer externen Batterie gepuffert werden, wenn erforderlich. Hierzu kann z.B. auch die Batterie des PC verwendet werden (siehe Kapitel 2).

Code/Data	**Funktion**
34h 20h	**Lesen des Status der Uhr**
34h b	Antwort: b = Status der Uhr: Bit 0: s.S. 6-42 Bit 1: 1 = Uhr stop, 0 = Uhr läuft Bit 2: wird als 1 gelesen (24h-Mode) Bit 3: Impulsausgang: 1 = maskiert Bit 4 und 5: Frequenz am Impulsausgang 0 0 1/15,625 ms 1 0 1/sec 0 1 1/min 1 1 1/h Bit 6: Invert. Zustand des Impulsausgangs Bit 7: Reserviert, wird als 0 gelesen

Code/Data	**Funktion**
35h 01h s	**Setzen der Betriebsart der Uhr** s = Betriebsart der Uhr: Bit 0: Reset Subsekundenzähler: Wenn diese Funktion nicht aktiviert werden soll, muß das Bit = 0 gesetzt werden. Um den Zähler zurückzusetzen, muß das Bit zunächst = 1 und dann mit einem zweiten Aufruf des Makrobefehls = 0 gesetzt werden. Bit 1: 1 = Uhr stop, 0 = Uhr läuft Bit 2: Immer = 1 setzen (24h-Mode) Bit 3: Impulsausgang: 1 = maskieren Bit 4 und 5: Frequenz am Impulsausgang 0 0 1/15,625 ms 1 0 1/sec 0 1 1/min 1 1 1/h Bit 6: Interne Funktion (immer = 1 setzen) Bit 7: Reserviert (immer = 0 setzen)
36h 40h	**Lesen der Uhrzeit**
36h h m s	Antwort: h = Stunden (0 bis 23) m = Minuten (0 bis 59) s = Sekunden (0 bis 59)

Code/Data	Funktion
36h 80h	**Lesen von Datum und Uhrzeit** (Wenn ein Fehler beim Lesen der Uhr auftritt, sind j, m, t, w und h = 0, m enthält den Fehlertyp und s die Fehlergruppe).
36h	Antwort:
j	j = Jahr (0 bis 255)
m	m = Monat (1 bis 12)
t	t = Tag (1 bis 31)
w	w = Wochentag (0 bis 6, 0 = Sonntag)
h	h = Stunden (0 bis 23)
x	x = Minuten (0 bis 59)
s	s = Sekunden (0 bis 59)
37h 07h	**Datum und Uhrzeit stellen** (Wenn eine der Angaben außerhalb des erlaubten Bereichs liegt, wird nichts neu gesetzt).
j	j = Jahr (0 bis 255)
m	m = Monat (1 bis 12)
t	t = Tag (1 bis 31)
w	w = Wochentag (0 bis 6, 0 = Sonntag)
h	h = Stunden (0 bis 23)
x	x = Minuten (0 bis 59)
s	s = Sekunden (0 bis 59)

! *Bitte beachten Sie, daß die Echtzeituhr der MODULAR-4/486 Karte die Jahreszahl nur als 2 stelligen Wert (0..99) speichert.*

12.11. Kontroll-LEDs

Code/Data	Funktion
30h (00h)	**on-board LED2 ein (LEDint)**
31h (00h)	**on-board LED2 aus (LEDint)**
33h 20h	**Anforderung: Zustand der on-board LED2 (LEDint)**
33h b	Antwort: Zustand der LED2 (Bit 0: 0 = aus, 1 = ein)
23h 03h 50h 00h 00h	**externe LED1 ein (LEDext)**
23h 03h 51h 00h 00h	**externe LED1 aus (LEDext)**

12.12. Cache-Kontrolle

Code/Data	Funktion
39h 01h s	**Cache ein / aus** s = 0 : Cache aus s = 1 : Cache ein s = 3 : Cache ungültig machen und einschalten

12.13. Initialisierung von Basiskarte und Modulen

Code/Data	Funktion
3bh 22h	**Initialisiere Basiskarte bzw. Modul entsprechend dem Inhalt des zugehörigen EEPROMs**
m	Initialisierung von: m = 0: MODULAR-4/486 Basiskarte m = 1 bis 4: Steckplatz auf Basiskarte[1] m = 5 bis 10: Steckplatz auf Modul-Extender
f	Flag: f = 0: Initialisiere nur, wenn Bit 0 von WORT-1 des zugehörigen EEPROMs = 1 f = 1: Initialisiere in jedem Fall
3bh s	Antwort: Status s=00h: alles ok s=80h: kein Modul auf Steckplatz s=81h: Modul braucht keine Initialisierung s=82h: Modul wurde nicht initialisiert wg. EEPROM-WORT-1 Bit 0 = 0 s=83h: Initialisierungsroutine nicht implementiert

[1] Bei der "kleinen" MODULAR-4/486 stehen nur die Modulsteckplätze 1 und 2 zur Verfügung

12.14. Makro-Befehle für das Flash-EPROM

Code/Data	Funktion
2ah 44h	**FLASH_STATUS setzen bzw. lesen**
SP IC d p	SP=Modul-Steckplatz, IC=IC-Nr. d=Dienst (s.u.), p=Parameter

d	p	Funktion
0	0	Hersteller-ID des Flash-EPROMs ermitteln
1	0	Device-ID des Flash (Device-ID) ermitteln
2	0	Länge Flash (L) und Sektor (H) ermitteln: Länge (in Anzahl Byte) = 2^L bzw. 2^H
3	0	**Organisation** des Flash lesen (L=Org., H=0) L (untere 4 Bit) =0: Bit, =1: Byte, =2: Word L (obere 4 Bit) =0: kein Sektor geschützt, =1: Bottom, =2: Top, H=0
4	s	**Check, ob Sektor s Write-Protected** L=0: Write-Protected, L=1: Not-Protected, H=0
5	s	**Status nach Erase von Sektor/en s** L=0, H=0: Löschen fertig L=1, H=0: Löschen noch nicht fertig L=2, H=0: Fehler bei Löschen aufgetreten
10	0	**Lies Soft-State** (L = Soft-State, H=0)
11	0	**Setze "Erase enable"** (Soft-State 0 => 1)
-	-	Makro-Befehl: Erase Flash (Soft-State 1 => 2)
12	0	**Setze "Erase disable"** (Soft-State 0, 1 oder 2 => 0)
13	0	**Setze "Program enable"** (Soft-State 0 => 3)
-	-	Makro-Befehl "Programmiere Flash" (Soft-State bleibt => 3)
14	0	**Setze "Program disable"** (Soft-State 0 oder 3 => 0)

2ah err L H	Antwort: err=Error; L, H: Ergebnis je nach Dienst

Code/Data	Funktion
2bh 4ah	**FLASH_ERASE: Erase Flash-EPROM direkt**
SP IC	SP=Modul-Steckplatz, IC=IC-Nr.
aE aF aG aH	Erste zu löschende Adresse (bzg. auf Anfang Flash)
eE eF eG eH	Letzte zu löschende Adresse (bzg. auf Anfang Flash)
2bh err x 0	Anwort: err = Error, x = Nr. eines gelöschten Sektors
2ch 4dh	**FLASH_PROGRAM: Flash direkt programmieren**
nL nH	Anzahl Byte + 6 (n max. = 262)
SP IC	SP=Modul-Steckplatz, IC=IC-Nr.
rE rF rG rH	Relative Adresse des ersten zu progr. Bytes
d0...dn-6	Data 0 bis n-6
2ch err	Antwort: err = Error,
wL wH	w = Anzahl verworfener Datenbyte
2dh d6h	**FLASH_READ: Flash direkt lesen**
mL mH	Anzahl zu lesender Byte (m max. = 256)
SP IC	SP=Modul-Steckplatz, IC=IC-Nr.
rE rF rG rH	Rel. Adresse des ersten zu lesenden Bytes
2dh err	Antwort: err = Error,
iL iH	i = Anzahl tatsächlich gelesener Bytes,
d1...di	Data 1...i

12.15. Makro-Befehle für MDD-Dienste

Code/Data	Funktion
78h 24h hE h*F hG hH	**Sonderdienst ohne Parameter** h = Handle, h* = Handle mit Dienst
78h err	Antwort: err = Fehlermeldung
78h 25h hE h*F hG hH d	**Sonderdienst mit Parameter Byte Hin** h = Handle, h* = Handle mit Dienst d = Parameter Byte
78h err	Antwort: err = Fehlermeldung.
78h 26h hE h*F hG hH dL dH	**Sonderdienst mit Parameter Wort Hin** h = Handle, h* = Handle mit Dienst d = Parameter Wort
78h err	Antwort: err = Fehlermeldung
78h 28h hE h*F hG hH dE dF dG dH	**Sonderdienst mit Parameter Doppelwort Hin** h = Handle, h* = Handle mit Dienst d = Parameter Doppelwort
78h err	Antwort: err = Fehlermeldung
78h 34h hE h*F hG hH	**Sonderdienst mit Parameter Byte Rück** h = Handle, h* = Handle mit Dienst
78h err d	Antwort: err = Fehlermeldung d = Parameter Byte
78h 44h hE h*F hG hH	**Sonderdienst mit Parameter Wort Rück** h = Handle, h* = Handle mit Dienst
78h err dL dH	Antwort: err = Fehlermeldung d = Parameter Wort
78h 64h hE h*F hG hH	**Sonderdienst mit Parameter Doppelwort Rück** h = Handle, h* = Handle mit Dienst
78h err dE dF dG dH	Antwort: err = Fehlermeldung d = Parameter Doppelwort

Code/Data	Funktion
78h ddh nL nH 00h 00h hE h*F hG hH b1...bn-4	**Sonderdienst mit Stream Hin** n = Anzahl Byte Nutzdaten + 4 h = Handle, h* = Handle mit Dienst Nutzdaten
78h 00h wL wH 00h 00h 78h err n-4L n-4H 00h 00h	Antwort: w = Anzahl verworfener Nutzdaten (w ≤ n-4) oder Antwort: err = Fehlermeldung n = Anzahl Byte Nutzdaten
78h 0ddh 04h 00h mL mH hE h*F hG hH	**Sonderdienst mit Stream Rück** m = max. Anzahl Byte Nutzdaten Rück h = Handle, h* = Handle mit Dienst
78h 00h 00h 00h vL vH b1...bv 78h err 00h 00h 00h 00h	Antwort: v = tatsächliche Anzahl Nutzdaten Rück (v ≤ m) Nutzdaten oder Antwort: err = Fehlermeldung

Code/Data	Funktion
78h 0ddh nL nH mL mH hE h*F hG hH b1...bn-4	**Sonderdienst mit Stream Hin und Rück (noch nicht implementiert** n = Anzahl Nutzdaten hin + 4 m = max. Anzahl Byte Nutzdaten Rück h = Handle, h* = Handle mit Dienst Nutzdaten
78h 00h wL wH vL vH b1...bv 78h err n-4L n-4H 00h 00h	Antwort: w = Anzahl verworfener Nutzdaten (w ≤ n-4) v = tatsächliche Anzahl Nutzdaten Rück (v ≤ m) Nutzdaten oder Antwort: err = Fehlermeldung n = Anzahl Byte Nutzdaten
79h 34h hE h*F hG hH	**Lies Einzel-Data: Bit** h = Handle, h* = Modifiziertes Handle
79h err d	Antwort: err = Fehlermeldung d = Bit
79h 34h hE h*F hG hH	**Lies Einzel-Data: Byte** h = Handle, h* = Modifiziertes Handle
79h err d	Antwort: err = Fehlermeldung d = Byte
79h 44h hE h*F hG hH	**Lies Einzel-Data: Wort** h = Handle, h* = Modifiziertes Handle
79h err dL dH	Antwort: err = Fehlermeldung d = Wort
79h 64h hE h*F hG hH	**Lies Einzel-Data: Doppelwort** h = Handle, h* = Modifiziertes Handle
79h err dE dF dG dH	Antwort: err = Fehlermeldung d = Doppelwort
7ah 24h hE h*F hG hH	**Schreibe Einzel-Data: Trigger** h = Handle, h* = Modifiziertes Handle
7ah err	Antwort: err = Fehlermeldung

Code/Data	Funktion
7ah 25h hE h*F hG hH d	**Schreibe Einzel-Data: Bit** h = Handle, h* = Modifiziertes Handle d = Bit
7ah err	Antwort: err = Fehlermeldung
7ah 25h hE h*F hG hH d	**Schreibe Einzel-Data: Byte** h = Handle, h* = Modifiziertes Handle d = Byte
7ah err	Antwort: err = Fehlermeldung
7ah 26h hE h*F hG hH dL dH	**Schreibe Einzel-Data: Wort** h = Handle, h* = Modifiziertes Handle d = Wort
7ah err	Antwort: err = Fehlermeldung
7ah 28h hE h*F hG hH dE dF dG dH	**Schreibe Einzel-Data: Doppelwort** h = Handle, h* = Modifiziertes Handle d = Doppelwort
7ah err	Antwort: err = Fehlermeldung
7bh 0d4h mL mH hE h*F hG hH	**Lies Daten-Block bzw. Stream** m = max. Anzahl Byte Nutzdaten Rück + 4 h = Handle, h* = Handle mit Dienst
7bh err vL vH a1...av	Antwort: err = Fehlermeldung v = tatsächliche Anzahl Nutzdaten Rück (v ≤ m-4) Nutzdaten
7ch 4dh nL nH hE h*F hG hH d1...dn-4	**Schreibe Daten-Block bzw. Stream** n = Anzahl Byte Nutzdaten h = Handle, h* = Handle mit Dienst Nutzdaten
7ch err wL wH	Antwort: err = Fehlermeldung w = Anzahl verworfener Nutzdaten (w ≤ n-4)

13. Serielle Kommunikation

Das MODULAR-4 System bietet eine breite Palette serieller Schnittstellen. Die Intelligenz der MODULAR-4 Karte wird in diesem Aufgabenfeld zum einen für die Pufferung der Daten und zum anderen für die Abarbeitung beliebiger Protokolle benutzt. Beides läuft - wie Sie das inzwischen vom MODULAR-4 System kennen - komplett unabhängig vom PC. Der PC gibt die Nutzdaten zu beliebiger Zeit mit hoher Geschwindigkeit zur Karte, die sich von da ab allein um das Senden der Daten und um die Erfüllung aller Protokollanforderungen kümmert.

Die Echtzeit-Programme für die serielle Kommunikation lassen sich in zwei Teile zerlegen. Der eine Teil übernimmt das Senden und das Empfangen von Zeichen aus dem bzw. in den Speicher der MODULAR-4 Karte. Er setzt direkt auf die Hardware auf und stellt alle zeitkritischen Funktionen zur Verfügung. Er muß so programmiert sein, daß auch bei hohen Baudraten keine Zeichen verlorengehen. Dieser Teil wird als **Basiskommunikation** bezeichnet. Der zweite Teil ist eher zeitunkritisch. Er übernimmt das gesamte **Protokollhandling**, also die Erzeugung und Überprüfung von Steuerzeichen und -signalen, Anforderung von Wiederholungen und alle anderen Aktionen, die zur Erfüllung der Protokollspezifikationen nötig sind. Natürlich kann auch in diesem Teil des Programms die Zeit eine wichtige Rolle spielen, wenn zum Beispiel Reaktionstelegramme innerhalb einer bestimmten Zeitspanne versandt werden müssen.

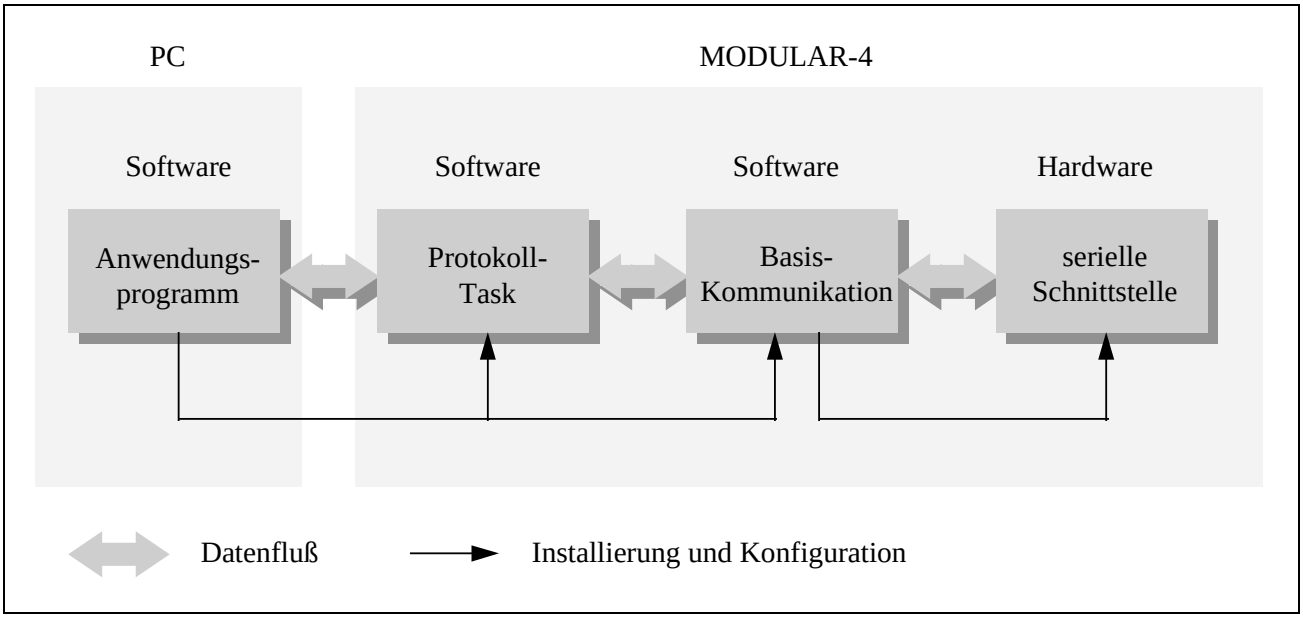


Abb. 13-1: Grundstruktur der seriellen Kommunikation

Der PC tauscht mit dem Teil bzw. mit der Task, die das Protokollhandling übernimmt, die Sende- und Empfangsdaten aus. Wenn ganz ohne Protokoll oder nur mit einem einfachen Handshake (RTS/CTS, XON/XOFF) kommuniziert wird, entfällt die Protokolltask, und der PC tauscht direkt mit den Tasks der Basiskommunikation Daten aus.

Um die mitgelieferten Kommunikationsprogramme verwenden zu können, muß auf der MODULAR-4 Karte ein Betriebssystem mit der Version ML7-1A.01x bzw. ML8-3A.00x (x=E: EPROM- oder x=R: RAM-Version) für oder größer laufen. Diese Beschreibung bezieht sich auf die Version 2.L der Kommunikationsprogramme oder spätere Versionen.

13.1. Basiskommunikation

Die Programme der Basiskommunikation lösen, wie bereits erwähnt, den zeitkritischen Teil der Kommunikation und arbeiten direkt mit der Hardware. Die Basiskommunikation für ein Modul bzw. für die Trägerkarte setzt sich aus mehreren Programmen bzw. Tasks zusammen. Die Installierung der Programme geschieht am einfachsten mit SNW.

13.1.1. Datenpuffer

Die Kommunikationstasks arbeiten mit zwei unterschiedlichen Datenpuffern.

Der **Sendepuffer** darf nur mit den entsprechenden Funktionen der Basiskommunikationstask beschrieben werden.

Der **Empfangspuffer** enthält die empfangenen Zeichen. Auch er darf nur über den Aufruf von Funktionen der Empfangstask ausgelesen werden.

Für beide Datenpuffer muß beim Installieren angegeben werden, wie groß sie sein sollen.

13.1.2. Handshake (asynchrone Kommunikation)

Für asynchrone Kommunikation bietet die Basiskommunikation zwei einfache Handshakemechanismen (RTS/CTS und XON/XOFF), mit denen verhindert werden kann, daß der Empfangspuffer überläuft. In beiden Fällen meldet die Empfangsstation, wenn der Empfangspuffer einen bestimmten (einstellbaren) 'Füllstand' erreicht hat. Der Sender sendet der Gegenstation dann vorübergehend keine Zeichen mehr,

bis gemeldet wird, daß der 'Füllstand' wieder unter eine (einstellbare) Grenze gefallen ist.

Beim **RTS/CTS**-Handshake muß die RTS-Leitung des Empfängers mit der CTS-Leitung des Senders verbunden werden. Wenn ein bestimmter Füllstand (Meldegrenze für 'Puffer voll') erreicht ist, setzt der Empfänger RTS auf Null, der Sender unterbricht dann den laufenden Sendevorgang. Sobald der Puffer wieder leer ist (Meldegrenze für 'Puffer leer'), wird RTS wieder auf eins gesetzt, und der Sender fährt mit dem Senden von Zeichen fort.

Der **XON/XOFF**-Handshake benötigt im Gegensatz zu RTS/CTS keine zusätzlichen Steuerleitungen. Der Empfänger sendet, sobald die obere Meldegrenze erreicht ist, das Steuerzeichen XOFF zum Sender. XOFF ist ein beliebiges Zeichen, das auf Empfänger- und Senderseite gleich sein muß (üblicherweise 13h). Sobald auf der Sendeseite XOFF empfangen wird, wird das Senden weiterer Zeichen unterbunden, bis der Empfänger durch das Senden von XON wieder Empfangsbereitschaft meldet. XON ist ebenfalls ein frei definierbares Zeichen (üblicherweise 11h). Beim Arbeiten mit XON/XOFF ist zu beachten, daß sowohl XON als auch XOFF reservierte Steuerzeichen sind und in den Nutzdaten nicht vorkommen dürfen. Sie werden immer als Steuerzeichen interpretiert und gelangen nie in den Empfangspuffer.

13.1.3. Senden und Empfangen

Die für Senden und Empfangen benutzten Parameter, Prozeduren und Funktionen sind für alle von SORCUS gelieferten Basiskommunikationsprogramme gleich. Je nachdem, ob die Sendedaten vom PC oder von einer übergeordneten Protokolltask kommen, müssen folgende Schritte mit Befehlen der Bibliotheken ML7BIB bzw. ML8BIB oder mit Betriebssystemaufrufen von ML7RTBIB bzw. ML8RTBIB durchgeführt werden. Alle in diesem Kapitel beschriebenen Aufrufe beziehen sich auf die Kommunikationstask des jeweiligen Kanals. Die Tasknummern können frei gewählt werden; empfohlen wird jedoch folgende Zuordnung, die standardmäßig auch beim Installieren mit SNW eingehalten wird:

Modulsteck-platz[1]	**Interruptkanal**	**Tasknummer des Interrupt-Managers**	**Tasknummer der Kommunikationstask**
0 (=Basiskarte)	SCC (90h)	300h	310h + Kanalnummer
1	IRQ-D (7eh)	301h	318h + Kanalnummer
2	IRQ-C (7dh)	302h	320h + Kanalnummer
3	IRQ-B (7ch)	303h	328h + Kanalnummer
4	IRQ-A (7bh)	304h	330h + Kanalnummer
5	IRQ-E (7fh)	305h	338h + Kanalnummer
6	IRQ-D (7eh)	306h	340h + Kanalnummer
7	IRQ-C (7dh)	307h	348h + Kanalnummer
8	IRQ-B (7ch)	308h	350h + Kanalnummer
9	IRQ-A (7bh)	309h	358h + Kanalnummer
10	IRQ-E (7fh)	30ah	360h + Kanalnummer

Tab. 13-1: Verteilung von Tasknummern und Interruptnummern bei Installierung der Basiskommunikation mit SNW. 'Kanalnummer' steht für die auf dem Modul verwendete serielle Schnittstelle, wobei folgende Zuordnung gilt: Kanal-A = 0, Kanal-B = 1, ... Kanal-H = 7.

! *Beachten Sie, daß mit den derzeitigen Basiskommunikationsprogrammen jedes Modul eine eigene Interruptleitung belegen muß. Es ist zur Zeit mit SNW also nicht möglich, z.B. auf Steckplatz 2 und Steckplatz 7 gleichzeitig Kommunikationsmodule zu installieren.*

[1] Bei der "kleinen" MODULAR-4/486 stehen nur die Modulsteckplätze 1 und 2 zur Verfügung

Senden

Um Daten zu senden, müssen Sie die Funktion 7 der Basiskommunikation aufrufen und dabei die Sendedaten (max. 256 Bytes) übergeben. Werten Sie unbedingt die von der Funktion zurückgegebene Fehlermeldung aus! Sie enthält zum Beispiel Informationen darüber, ob die Daten in den Puffer eingetragen werden konnten.

Das folgende Beispiel zeigt eine Funktion zum Senden eines Strings. Der Rückgabewert ist Null, wenn der String in den Sendepuffer eingetragen werden konnte.

Pascal:

```
FUNCTION send_string (modul, channel: BYTE; data: STRING): BYTE;
VAR
  dummyin, dummysize : BYTE;
  error : BYTE;
  task  : WORD;
BEGIN
  task := $310 + $08 * modul + channel;

  { Die Sendedaten werden mit data[1] übergeben, da bei einer }
  { Übergabe mit data auch das erste Byte der Stringvariablen }
  { (= Stringlänge) übertragen würde.                         }

  ml8_call_func(task,7,LENGTH(data),data[1],0,dummysize,dummyin,error);

  send_string := error;
END;
```

C:

```
byte send_string (byte modul, byte channel, char *data)
{
  void   *dummyin;
  ushort dummysize;
  byte   error;
  ushort task;

  task = 0x310 + 0x08 * modul + channel;

  ml8_call_func(task,7,strlen(data),data,0,&dummysize,dummyin,&error);

  return(error);
}
```

Empfangen

Die empfangenen Daten stehen im Empfangspuffer der Basiskommunikationstask, sie können mit der Funktion 17 (11h) der Kommunikationstask abgeholt werden. Dazu müssen Sie zuerst eine Datenstruktur reservieren, die die empfangenen Daten aufnehmen soll, und die Adresse dieser Struktur übergeben. Werten Sie unbedingt den von der Funktion zurückgelieferten Fehlercode aus. Er gibt zum Beispiel an, ob die übergebenen Daten gültig sind.

Das folgende Beispiel kopiert empfangene Zeichen in einen String. Wenn das Funktionsergebnis Null ist, sind die Daten gültig. Der beim Aufruf der Funktion übergebene String muß für mindestens 255 Zeichen Speicher reserviert haben.

Pascal:

```
FUNCTION rcv_string (modul,channel: BYTE; VAR data: STRING): BYTE;
VAR
  dummyout : BYTE;
  insize   : WORD;
  error    : BYTE;
  task     : WORD;
BEGIN
  task := $310 + $08 * modul + channel;

  { Die Empfangsdaten werden nicht an data, sondern an data[1] }
  { geschrieben, da das erste Byte von Data die Stringlänge    }
  { enthält.                                                   }

  ml8_call_func(task,17,0,dummyout,255,insize,data[1],error);

  data[0] := chr(insize);        {Stringlänge einstellen}
  rcv_string := error;
END;
```

C:

```
byte rcv_string (byte modul, byte channel, char *data)
{
  void   *dummyout;
  ushort insize;
  byte   error;
  ushort task;

  task = 0x310 + 0x08 * modul + channel;

  ml8_call_func(task,17,0,dummyout,255,&insize,data,&error);

  data[insize] = 0;  /* String mit 'Null' beenden */
  return(error);
}
```

13.2. Protokollhandling

Die von SORCUS lieferbaren Protokollprogramme unterstützen alle lieferbaren Kommunikationsmodule. Sie setzen auf der Basiskommunikation auf und können teilweise auch mit SNW installiert werden. Das Senden und Empfangen von Daten wird bei jedem Protokoll unterschiedlich gehandhabt.

13.3. Installieren mit SNW (asynchrone Kommunikation)

Mit Hilfe von SNW kann das Installieren der Basiskommunikationsprogramme und zum Teil auch der Protokollprogramme sehr einfach und komfortabel geschehen. Voraussetzungen dafür sind lediglich, daß sich die MODULAR-4 Karte mit den Kommunikationsmodulen im PC befindet und daß alle Kommunikationsprogramme in einem gemeinsamen Verzeichnis stehen.

Alle Hilfsmittel zur Installierung finden Sie unter dem Hauptmenüpunkt **COM**.

Kapitel 13.5 beschreibt die Installation der Basiskommunikation ohne SNW.

13.3.1. Gesamtkonfiguration

Die Gesamtkonfiguration wird in einem Fenster angezeigt, das mit **Konfiguration öffnen** geöffnet bzw. erstellt wird (siehe Abb. 13-2). Es enthält die Einstellungen aller verfügbaren Schnittstellen einer Karte und ihrer Module. Zum Konfigurieren und Installieren einzelner Kanäle muß immer eine Gesamtkonfiguration geöffnet sein. Die meisten Menüpunkte von COM lassen sich auch nur dann aktivieren, wenn ein Fenster mit einer Gesamtkonfiguration angewählt ist (doppelt umrandet).

Das Fenster enthält eine Tabelle, in der alle verfügbaren Schnittstellen aufgelistet sind. Wenn ein Kanal verwendet wird, steht in der Tabelle ein Name. Unbenutzte Kanäle werden mit einem waagerechten Strich gekennzeichnet. Innerhalb der Tabelle ist immer ein Kanal angewählt (invers dargestellt). In der Box links unterhalb des Fensters werden seine Konfigurationsdaten angezeigt. Die Kanalanwahl kann mit den Cursortasten oder mit der Maus vorgenommen werden.

Die Gesamtkonfiguration wird mit dem Menüpunkt **Konfiguration speichern** in einer Datei mit der Erweiterung '.SCF' (**S**erial **C**on**F**iguration) gespeichert. **Konfiguration öffnen** dient sowohl dem Laden einer vorhandenen Datei als auch dem Erstellen einer neuen. Wenn Sie eine neue Datei erstellen wollen, müssen Sie bei **Konfiguration öffnen** eine neuen Namen angeben.

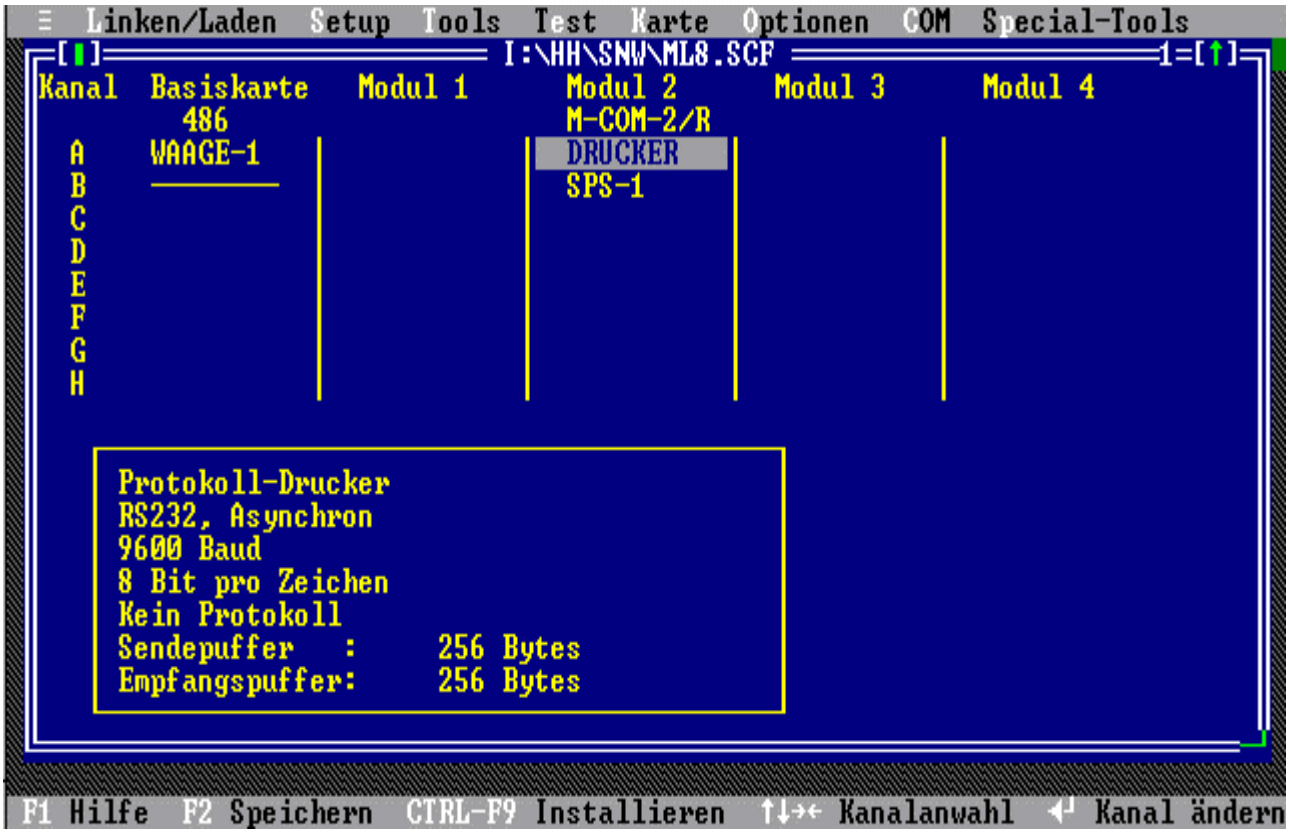


Abb. 13-2: Konfigurationsfenster in SNW

13.3.2. Kanaleinstellungen

Die für die Datenübertragung relevanten Parameter werden als *Kanaleinstellungen* bezeichnet. Sie werden für jeden Kanal getrennt eingestellt und gespeichert. Um die Einstellungen vorzunehmen, wählt man zunächst den gewünschten Kanal an, drückt die [Return]-Taste oder wählt den Menüpunkt **Kanaleinstellung ändern**. Alternativ genügt ein Doppelklick auf den gewünschten Kanal mit der Maus. Die Einstellungen werden für jeden Kanal in einer eigenen Datei mit der Bezeichnung <Name>.chc (**CH**annel-**C**onfiguration) gespeichert.

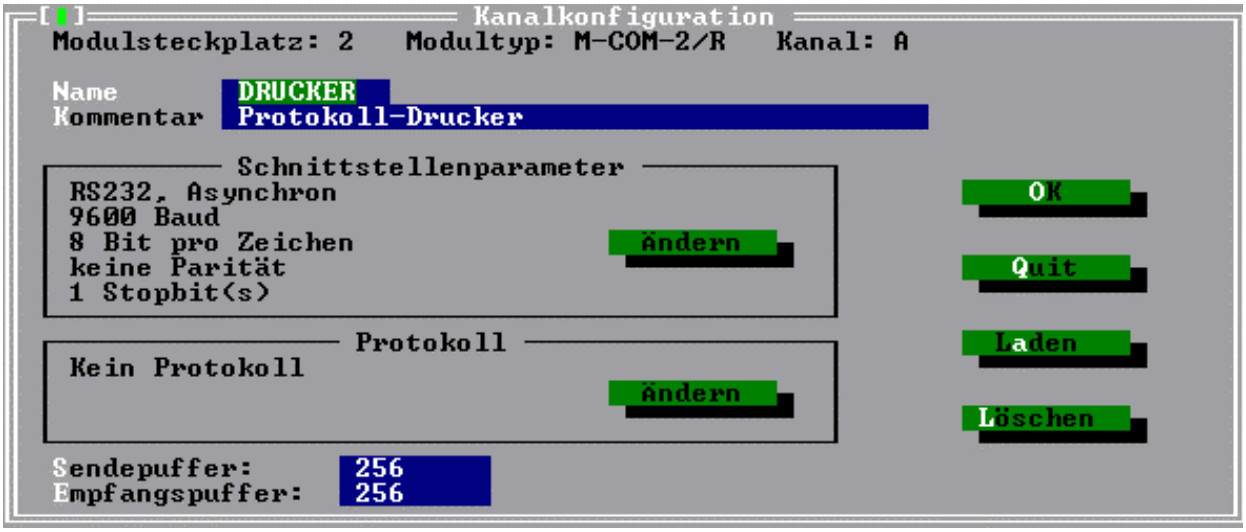


Abb. 13-3: Kanaleinstellungen in SNW

- **Name**
  Jeder Kanal-Parametersatz erhält einen Namen, der innerhalb einer Gesamtkonfiguration nur einmal vorkommen darf. Der Name dient zum einen der Übersichtlichkeit in der Gesamttabelle, wenn er eindeutig beschreibt, womit die Schnittstelle verbunden ist. Zum anderen dient er zum einfachen Austauschen und Kopieren von Konfigurationsdaten (siehe 'Laden'). Der in diesem Feld eingetragene Text muß den Ansprüchen eines Dateinamens (ohne Erweiterung) genügen. Vorsicht ist geboten, wenn mehrere Gesamtkonfigurationen denselben Namen (und damit dieselbe Parameterdatei) verwenden. In diesem Fall wirkt sich die Änderung der Parameter natürlich auch auf alle anderen Konfigurationen aus.

- **Kommentar**
  Das Kommentarfeld darf bis zu 40 beliebige Zeichen enthalten. Es dient der Beschreibung, z.B. der möglichst genauen Zuordnung der installierten Schnittstelle bzw. der Gegenstation.

- **Schnittstellenparameter**
  Als Schnittstellenparameter gelten die Parameter, die zur Übertragung eines Zeichens eingestellt werden müssen (Schnittstellentyp, Baudrate, Bit pro Zeichen, Paritätsprüfung und Stopbits). In der Box wird die aktuelle Einstellung gezeigt. Wenn sie geändert werden soll, muß der zugehörige 'Ändern'-Schalter betätigt werden. Dadurch wird eine Dialogbox zur Einstellung der Parameter geöffnet. Die einzige Besonderheit dieser Dialogbox ist die Eingabe der Baudrate. Neben dem Eingabefeld befindet sich ein Abwärtspfeil, der mit der Maus angeklickt oder mit der [↓]-Taste aktiviert werden kann. Damit erhalten Sie eine Liste aller für diesen Kanal einstellbaren Baudraten.

- **Protokoll**
  Unter diesen Punkt fallen sowohl die einfachen Handshakemechanismen XON/XOFF und RTS/CTS als auch komplexe, von SORCUS lieferbare Protokolle wie z.B. 3964/R. Die Box zeigt die aktuellen Einstellungen, die mit dem zugehörigen 'Ändern'-Schalter geändert werden können. Je nach eingestelltem Protokoll können die Dialogboxen zur Eingabe der Protokollparameter stark variieren.

- **Sendepuffer**
  In dieses Feld wird die Größe des Sendepuffers in Byte eingetragen.

- **Empfangspuffer**
  Hier wird die Größe des Empfangspuffers in Byte angegeben.

- **Laden**
  Laden dient zum Übernehmen der Parameter (Schnittstellen-, Protokollparameter, ...) anderer Schnittstellen. Nach dem Betätigen des Schalters muß der Name des Parametersatzes angegeben werden, der kopiert werden soll. Wenn es ein Parametersatz ist, der in der aktuellen Gesamtkonfiguration bereits enthalten ist, dann muß er anschließend umbenannt werden. Auf diese Weise können sehr schnell Installierungen zur Ansteuerung vieler gleichartiger Geräte durchgeführt werden. Beim Kopieren von Kanälen anderer Module kann es vorkommen, daß Einstellungen auf dem angewählten Modul unmöglich sind. In diesem Fall wird eine Warnung ausgegeben und für den fraglichen Wert eine Standardeinstellung vorgenommen.

- **Löschen**
  Dieser Schalter entfernt die Kanaleinstellung aus der Gesamtkonfiguration, der Kanal wird als unbenutzt markiert. Die Parameterdatei <Name>.chc wird aber nicht gelöscht, die Einstellungen können also später mit 'Laden' wieder zurückgeholt werden.

- **Sonderfunktionen**
  Unter diesem Punkt sind Funktionen und Parameter zusammengefaßt, die nicht für alle Module bzw. MODULAR-4 Karten zur Verfügung stehen (z.B. unterschiedliche Baudraten für Senden und Empfangen) und die üblicherweise nicht verwendet oder geändert werden. Die Dialogbox für Sonderfunktionen ist für jedes Modul und jede MODULAR-4 Karte unterschiedlich. Wenn keine Sonderfunktionen zur Verfügung stehen, fehlt der Schalter in der Dialogbox.

13.3.3. Installieren

Als Installieren wird das Laden und Parametrieren der Kommunikationsprogramme bezeichnet. Die MODULAR-4 Karte ist anschließend bereit zum Senden und Empfangen von Daten; alle Schnittstellen werden der Gesamtkonfiguration entsprechend eingerichtet. Für das Installieren mit SNW wird immer eine Installationsdatei (siehe Kapitel 4) benötigt. Sie kann mit dem Menüpunkt 'Installationsdatei erzeugen' erstellt werden. Wenn Sie 'Installieren' wählen, dann wird ebenfalls eine Installationsdatei erzeugt und anschließend installiert. Die Installationsdatei wird in dem Verzeichnis abgelegt, in dem sich die Gesamtkonfigurationsdatei befindet. SNW geht beim Laden davon aus, daß sich hier auch die Kommunikationsprogramme der MODULAR-4 Karte befinden. Die Installationsdatei erhält den gleichen Namen wie die Konfigurationsdatei, allerdings mit der Erweiterung '.INS'.

In der Installationsdatei sind alle Angaben über die Konfiguration der Schnittstellen enthalten, so daß sie auch alleine verwendbar ist. Sie kann also später auch direkt mit einem Kommandozeilenaufruf geladen werden, ohne daß die komplette SNW-Umgebung gestartet wird (z.B. in AUTOEXEC.BAT):

SNW /i:<name>

Unter dem Menüpunkt 'Optionen' können Sie auswählen, ob vor der Installierung der Kommunikationsprogramme ein Reset durchgeführt werden soll. Das ist dann sinnvoll, wenn außer den Kommunikationsprogrammen keine anderen Programme auf der Karte laufen sollen, oder die Kommunikationsprogramme immer zuerst installiert werden. **Standardmäßig wird ein Reset in die Installationsdatei eingefügt.**

13.3.4. Testen von Schnittstellen

Sobald die Kommunikationsprogramme für eine Schnittstelle installiert sind, können sie innerhalb von SNW getestet werden. Dazu muß der entsprechende Kanal im Konfigurationsfenster angewählt und im 'COM'-Untermenü 'Kanal testen' aktiviert werden. Getestet wird immer die Gesamtinstallierung eines Kanals. Wenn ein Protokoll installiert wurde, werden die Daten mit der übergeordneten Protokolltask und nicht mit der Basiskommunikation ausgetauscht. Von den Protokolltasks erzeugte Steuerzeichen und Telegrammköpfe gelangen nicht zum PC. Für jeden installierten Kanal kann ein Testfenster geöffnet werden, auch gleichzeitig mit anderen Fenstern.

Zeichen, die gesendet werden sollen, können direkt über die Tastatur eingegeben werden. Mit [Return] werden sie in den Sendepuffer der Karte übertragen. Das Fenster gibt allerdings keine Auskunft darüber, ob die Zeichen tatsächlich gesendet worden sind oder ob das Kommunikationsprogramm vielleicht noch auf ein Handshake-Signal zum Senden der Zeichen (z.B. XON) wartet. Die Sendedaten können entweder direkt als ASCII-Zeichen oder hexadezimal (ohne 'H', '0x' oder sonstige Kennung) eingegeben werden. Welches Format von beiden verwendet wird, können Sie in 'COM/Optionen' einstellen.

Etwa viermal pro Sekunde (bei jedem Blinken der linken oberen Fensterecke) überprüft SNW, ob Zeichen empfangen worden sind. Wenn das der Fall ist, werden bis zu achtzig Byte abgeholt und im unteren Fensterteil angezeigt. Das bedeutet, daß innerhalb des Testprogrammes von einer Schnittstelle maximal 320 Zeichen pro Sekunde zum PC übertragen werden können. Andere PC-Programme können natürlich mit wesentlich höheren Datenraten arbeiten.

13.3.5. Abhilfe bei Fehlern

In der Regel werden alle erkennbaren Fehler gemeldet. Solange eine Fehlermeldung auf dem Bildschirm angezeigt wird, erhalten Sie mit F1 weitere Informationen über die Fehlerursache und mögliche Abhilfen.

Wenn keine Zeichen gesendet oder empfangen werden, sollten Sie zunächst alle Kabel überprüfen. Denken Sie daran, daß die meisten seriellen Verbindungen ein sogenanntes Nullmodemkabel benötigen (mindestens TxD und RxD überkreuz miteinander verbunden). Stellen Sie sicher, daß Sender und Empfänger mit denselben Parametern (Baudrate ...), demselben Handshakemechanismus und demselben Protokoll arbeiten.

13.4. Die Grundstruktur der Basiskommunikation

Jedes Kommunikationsmodul hat mehrere serielle Schnittstellen, von denen wiederum jede aus mehreren Gründen Interrupts auslösen kann. Für jede Schnittstelle und jeden ihrer Interrupts wird eine eigene Task installiert. Da aber jedes Modul nur eine einzige Interruptleitung zur Basiskarte hat, muß zunächst entschieden werden, von welcher Schnittstelle und aus welchem Grund der Interrupt ausgelöst wurde, um zu bestimmen, welche Task den Interrupt bearbeiten muß. Diese Aufgabe wird von einer Task übernommen, die unter dem Interrupt des Moduls installiert und als **Interruptmanager** bezeichnet wird.

Die Basiskommunikationsprogramme bestehen aus einem Interruptmanager (Programm 500, M7P0500.LIB bzw. M8P0500.LIB[1]) und einem Kommunikationsprogramm (Programm 520, M7P0520.LIB bzw. M8P0520.LIB), die für die Schnittstellen der Basiskarte und für die Module M-COM-2 und M-COM-8 geeignet sind. Pro Modul muß eine Interruptmanager-Task installiert sein und für jeden verwendeten Kanal eine Kommunikations-Task.

Die Installierung aller notwendigen Tasks kann durch SNW erfolgen. Anschließend werden nur mit den Kommunikationstasks Daten ausgetauscht, der Interruptmanager wird nie aus der Anwendungsebene heraus angesprochen. Im folgenden werden die wichtigsten Parameter und Prozeduren des Kommunikationsprogramms dargestellt.

13.4.1. Das Kommunikationsprogramm 520

Das Kommunikationsprogramm 520 (M7P0520.LIB bzw. M8P0520.LIB) bedient **eine** serielle Schnittstelle des seriellen Schnittstellenbausteins auf der Basiskarte MODULAR-4/486 oder einem SPB-Modul M-COM-2 oder M-COM-8. Dies betrifft sowohl die Initialisierung der Schnittstelle als auch den laufenden Betrieb. Es ermöglicht den Zugriff auf alle für eine asynchrone serielle Schnittstelle relevanten Einstellungen und Zustände (Fehlerstatus, Statusleitungen, etc).

Das Basiskommunikationsprogramm enthält auch die für die beiden Datenfluß-Protokolle XON/XOFF und RTS/CTS nötigen Funktionen, die sich durch Parameter aktivieren lassen.

[1] "kleine" MODULAR-4/486: M7P0500.LIB und M7P0520.LIB,
"große" MODULAR-4/486: M8P0500.LIB und M8P0520.LIB

13.4.1.1. Initialisierung

Mit SNW wird das Basiskommunikationsprogramm den Einstellungen des Anwenders entsprechend parametriert und gestartet, so daß es zum Senden und Empfangen bereit ist (z.Zt. nur asynchrone serielle Kommunikation, die Initialisierung für synchrone Kommunikation muß vom Anwenderprogramm entsprechend der Parameterliste durchgeführt werden). Falls Sie aber Parameter (wie z.B. Baudrate) ändern oder die Kommunikation neu beginnen möchten, muß das Programm reinitialisiert werden. Nach dem Ändern von Parametern muß Prozedur 2 aufgerufen werden, damit die Änderungen wirksam werden. Mit den Prozeduren 5 und 15 können Sende- und Empfangsbereitschaft ein- und ausgeschaltet sowie Sende- und Empfangspuffer gelöscht werden.

13.4.1.2. Empfangen

Empfangene Zeichen stehen im Empfangspuffer und können mit der Funktion 17 ausgelesen werden (siehe Beispiel Seite 13-6). Dabei wird die gewünschte Anzahl an Zeichen und ein Pointer auf einen Puffer, in den die Zeichen geschrieben werden sollen, an die Funktion übergeben. Die Funktion überprüft dann den Status des Empfangspuffers, kopiert die empfangenen Zeichen in den angegebenen Puffer und liefert die Anzahl der tatsächlich gelesenen Zeichen zurück oder gegebenenfalls eine Fehlermeldung (siehe Tabelle der Fehlerrückgabecodes, Seite 13-23).

Um den Empfangsstatus zu ermitteln, liest man die zugehörigen Parameter (Nummern 262 und 264), in denen Fehlermeldungen sowie die Anzahl der Zeichen im Puffer enthalten sind.

13.4.1.3. Senden

Zeichen, die gesendet werden sollen, können mit der Funktion 7 an den Sendepuffer übergeben werden (siehe Beispiel auf Seite 13-5). Das Basiskommunikationsprogramm sendet dann die Zeichen. Der Funktion wird die Anzahl der zu sendenden Zeichen und der Zeiger auf einen Puffer, in dem die Zeichen stehen, übergeben. Eine Statusabfrage, ob die Zeichen gesendet oder in den Sendepuffer übernommen werden können, wird durch die Funktion vorgenommen. Im Fehlerfall wird ein Fehlercode zurückgeliefert (siehe Tabelle der Fehlerrückgabecodes, Seite 13-23).

Um den Sendestatus zu ermitteln (z.B. ob alle Zeichen gesendet wurden), liest man die zugehörigen Parameter (256 und 258), die Statusmeldungen (2 Byte) und die Anzahl der im Sendepuffer (4 Byte) enthaltenen Zeichen.

13.4.1.4. Die Parameter des Programmes 520

Alle Parameter, die in der folgenden Tabelle nicht beschrieben sind, sind reserviert und dürfen nicht geändert werden.

Nr.	Typ	Init	Zugr[1]	Bedeutung des Parameters
0	Byte	0	R	Genereller Programmstatus 0 = geladen, 1 = läuft, 2 = gestoppt, 3 = gestoppt nach Fehler
1	Byte	0	R	Fehlerinformation: gültig, wenn Parameter 0 = 3, siehe Fehlertabelle auf Seite 13-19
4	Word	0	R/W	Programmnummer des zugeh. Interruptmanagers
6*	Word	0	R/W	Tasknummer des zugehörigen Interruptmanagers
8*	Byte	0	R/W	Modulsteckplatz (0 = Basiskarte, 1 bis 10 = Modulsteckplatz)
9*	Byte	0	R/W	Kanalnummer auf Modulsteckplatz (0 bis max. 7)
10	Byte	1	R/W	Physikalische Verbindung (M-COM-2) 1 = RS-232 2 = 20 mA 3 = RS-422 4 = RS-485 5 = LWL invertierend 6 = LWL nicht invertierend 7 = RS-232 mit zusätzl. CLK-Input (CL-232A/i) 8 = RS-232 mit zusätzl. CLK-Output (CL-232A/o)
11*	Byte	1	R/W	Typ des Kommunikations-Controllers: 1 = SCC 2 = ESCC 3-255 = reserviert
16	Byte	1	R/W	Kommunikationstyp: 1 = asynchron 2 = Mono-Sync 3 = Bi-Sync 4 = external Sync 5 = SDLC[2] 6 = SDLC Loop Mode[2] 7-255 = reserviert

[1] Zugriff auf Parameter: R= Nur Lesen, R/W = Lesen und Schreiben

[2] Noch nicht implementiert

Nr.	Typ	Init	Zugr	Bedeutung des Parameters
17	Byte	1	R/W	Daten-Codierung Senden und Empfangen: 1 = NRZ (Non-Return to Zero) 2 = NRZI[1] (Non-Return to Zero Inverted) 3 = FM1[1] (Biphase Mark) 4 = FM0[1] (Biphase Space) 5 = Manchester[1] (Biphase Level) 6 bis 255 = reserviert
18*	Byte	1	R/W	Clock-Quelle Senden: 1 = Standard Baudratengenerator 2 = Von extern über Clock-Leitung an RTxC[2] 3 = Intern, Ausgabe auf Clock-Leitung[2] 4 = Autogeneration (DPLL) [1] (noch nicht implementiert) 5 = Von extern über Clock-Leitung an TRxC[2] 6-255 = reserviert
19*	Byte	1	R/W	Clock-Quelle Empfangen (Bedeutung siehe Parameter 18)
24*	Long (4)	9600	R/W	Baudrate für Senden[3]
28*	Byte	8	R/W	Zeichenlänge für Senden in Anzahl Bit: 5, 6, 7, 8
29*	Byte	1	R/W	Anzahl Stopbit (Senden und Empfangen) 0 = Sync-Mode Enable 1 = ein Stopbit, 2 = zwei Stopbits, 3 = 1,5 Stopbits
30*	Byte	0	R/W	Paritätsbit-Generierung (Senden und Empfangen): 0 = keine, 1 = ungerade, 2 = gerade
32	Byte	0	R/W	Sync Character 1 (Low-Byte, WR6)
33	Byte	0	R/W	Sync Character 2 (High-Byte, WR7)
34*	Long (4)	9600	R/W	Baudrate für Empfangen[3]

[1] Noch nicht implementiert

[2] Taktein- bzw. Ausgang entspricht dem Baudratentakt

[3] Es sind beliebige Baudraten einstellbar. Nach Aufruf der Prozedur 2 bzw. 4 wird hier die nächste realisierbare Baudrate eingetragen.

Nr.	Typ	Init	Zugr	Bedeutung des Parameters
38*	Byte	8	R/W	Zeichenlänge für Empfangen[1] in Anz. Bit: 5, 6, 7, 8
39	Byte	0	R	Bitmaske für Zeichenlänge[2]
40*	Long (4)	0	R/W	Sendepuffer-Größe (in Byte)
48*	Long (4)	0	R/W	Empfangspuffer-Größe (in Byte)
52	Word (2)	ffffh	R	Interrupt-Kontrolle (Bitmap, reserviert)
54	Byte	0	R/W	Empfangenes Byte bei Parity-Error verwerfen oder speichern: 0 = speichern, 1 = verwerfen
55	Byte	0	R/W	Sync-Features: Bit 0 = 1: TxCRC Enable Bit 1 = 1: SDLC/CRC-16 Bit 2 = 1: CRC Preset I/O Bit 3 = 1: 6-Bit/8-Bit Sync Bit 4 = 1: RxCRC Enable Bit 5 = 1: Sync Character Load Inhibit Bit 6 = 1: Enter Hunt Mode Bit 7 = 1: Address Search Mode (SDLC)
256	Word (2)	0	R	Sendestatus (Bitmap) Bit 0 = 1: Sendepuffer (Software) ist leer Bit 1 = 1: alle Zeichen gesendet (Hardware) Bit 2 bis 14: sind reserviert Bit 15 = 1: Sendeteil angehalten (durch PC oder Protokoll)
258	Long (4)	0	R	Anzahl Zeichen im Sendepuffer

[1] Standardmäßig liefert der Kommunikationsbaustein immer 8 Bit zurück, wobei bei Zeichenlängen kleiner 8 das höchstwertige Bit entsprechend der Parität gesetzt wird. Das Programm liefert normalerweise nur die gewünschte Anzahl von Bits/Zeichen. Alle anderen Bits werden = 0 gesetzt. Soll das Paritätsbit mitgeliefert werden, kann das über eine Bitmaske definiert werden (siehe Parameter 39)

[2] Bitmaske, die definiert, welche Bits zurückgeliefert werden sollen. Bit = 1 bedeutet: Bit wird nicht gelöscht.

Nr.	Typ	Init	Zugr	Bedeutung des Parameters
262	Word (2)	0	R	Empfangsstatus (Bitmap) Einmaliges Auftreten der Bedingung setzt das zugehörige Bit (Ausnahme Bit 15). Rücksetzung Bit 0 bis 4 durch Funktion 16 Bit 0 = Zeichenverlust wegen übergelaufenem Empfangspuffer Bit 1 = Zeichenverlust wegen Überlastung im ser. Controller Bit 2 = Parity-Fehler Bit 3 = Frame-Fehler Bit 4 = Break-Detection Bit 5 bis 14 sind reserviert Bit 15 = Empfangsteil angehalten (durch PC oder Protokoll)
264	Long (4)	0	R	Anzahl Byte im Empfangspuffer
268	Word (2)	0	R	Zustand der Eingangs-Steuerleitungen (Bitmap) Bit 0 = CTS (Clear To Send) Bit 1 = DCD (Data Carrier Detected) Bit 2 = RI (Ring Indicator)
290	Word (2)	0	R	Zähler, Sendepuffer war voll[1]
292	Word (2)	0	R	Zähler, Empfangspuffer übergelaufen[1]
294	Word (2)	0	R	Zähler, Zeichenverlust (Overrun)[1]
296	Word (2)	0	R	Zähler, Parity-Fehler[1]
298	Word (2)	0	R	Zähler, Framing-Fehler[1]
300	Word (2)	0	R	Zähler, Break-Fehler[1]
318	Word (2)	0	R/W	Protokolltyp 0 = kein Protokoll 1 = XON/XOFF 2 = RTS/CTS

[1] Parameter wird durch Aufruf der Funktion 2 auf Null gesetzt

Tabelle der möglichen Fehler für einen Programmabbruch

(Wenn Parameter 0 = 3 ist, dann steht in Parameter 1 die Fehlerursache.)

Fehlernummer in Parameter 1	**Erklärung**
0	Reserviert
1	Nicht genügend Platz für Sendepuffer-Reservierung
2	Falscher Parameter für Sendepuffer-Reservierung
3	Unbekannter Fehler während Sendepuffer-Reservierung
4	Falscher Parameter für Sendepuffer-Reservierung
7	Ungültige Sendepuffer-Nr.
8	Sendepuffer wird gerade benutzt (locked)
11	Nicht genügend Platz für Empfangspuffer-Reservierung
12	Falscher Parameter für Empfangspuffer-Reservierung
13	Unbekannter Fehler während Empfangspuffer-Reservierung
14	Falscher Parameter für Empfangspuffer-Rerservierung
17	Ungültige Puffernummer für Empfangspuffer
18	Empfangspuffer wird gerade benutzt (locked)
30	Fehler beim Aufruf einer externen Funktion (aufgerufen durch Aktions-Filter)
40	Fehler bei Interrupt-Service-Aufruf
50	Keine Quarzfrequenz in EEPROM eingetragen
99	Unbekannter Fehler

13.4.1.5. Die Funktionen und Prozeduren des Basis kommunikationsprogrammes

Das Basiskommunikationsprogramm umfaßt globale Prozeduren (ohne Übergabeparameter und Antwort) und globale Funktionen (mit Übergabe von Parametern und Anwort). In der folgenden Tabelle sind die Prozeduren in der Spalte 'Typ' mit P gekennzeichnet, die Funktionen mit F. Beim Aufruf von Funktionen müssen eine Reihe von Parametern übergeben werden. Sie sind in der folgenden Tabelle mit den Bezeichnern angegeben, mit der die Funktion in den Hochsprachen-Bibliotheken ML7BIB/ML8BIB und ML7RTBIB/ML8RTBIB aufgerufen wird.

Nr.	Typ	Bedeutung der Funktion
2	P	Start/Restart der Kommunikation (Konfiguration, Speicherreserv.)
4	P	Baudrate (Parameter 24 und 34) ändern
5	F	**Sendebereitschaft ein- und ausschalten,** optional Sendepuffer löschen Hin: outsize = 2 maxinsize = 0 data_out = *Kontroll-Wort* (s.u.) Rück: error = Fehlernummer *Kontroll-Wort:* 0 = Senden anhalten, Puffer nicht löschen 1 = Senden starten, Puffer nicht löschen 2 = Senden anhalten, Puffer löschen 3 = Senden starten, Puffer löschen
6	F	**Sendestatus melden** Hin: outsize = 0 maxinsize = 6 indata_var = *Rückgabe-Struktur* (s.u.) Rück: insize = 6 error = Fehlernummer *Rückgabe-Struktur:* Sendestatus (Word), Anzahl Zeichen im Sendepuffer (Long), Bedeutung wie Parameter 256 und 258

Nr.	Typ	Bedeutung der Funktion
7	F	**Zeichenkette an Sendepuffer übergeben** Hin: outsize = Anzahl zu übergebender Zeichen maxinsize = 0 data_out = *Übergabe-Puffer* (s.u.) Rück: error = Fehlernummer *Übergabe-Puffer:* Datenstruktur, die *outsize* zu sendende Bytes enthält.
15	F	**Empfangsbereitschaft ein- und ausschalten**, optional Empfangspuffer löschen Hin: outsize = 2 maxinsize = 0 data_out = *Kontroll-Wort* (s.u.) Rück: error = Fehlernummer *Kontroll-Wort:* 0 = Empfangen anhalten, Puffer nicht löschen 1 = Empfangen starten, Puffer nicht löschen 2 = Empfangen anhalten, Puffer löschen 3 = Empfangen starten, Puffer löschen
16	F	**Empfangsstatus melden** Hin: outsize = 2 maxinsize = 6 indata_var = *Rückgabe-Struktur* (s.u.) Rück: insize = 6 error = Fehlernummer *Rückgabe-Struktur:* Empfangsstatus (Word), Anzahl Zeichen im Empfangspuffer (Long), Bedeutung wie Parameter 262 und 264

Nr.	Typ	Bedeutung der Funktion
17	F	**Zeichenkette aus Empfangspuffer übernehmen** Hin: outsize = 0 maxinsize = Anzahl angeforderter Zeichen indata_var = *Rückgabe-Puffer* (s.u.) Rück: insize = Anzahl gelesener Zeichen error = Fehlernummer *Rückgabe-Puffer:* Datenstruktur, die die gelesenen Zeichen aufnehmen kann, also mindestens *maxinsize* Byte umfaßt. Nach dem Aufruf sind die ersten *insize* Byte gültig.
33	F	**Steuerleitungs-Ausgänge setzen** Hin: outsize = 4 maxinsize = 0 data_out = *Übergabe-Struktur* (s.u.) Rück: error = Fehlernummer *Übergabe-Struktur*: Wort 1: Maske für die betroffenen Steuerleitungen (Word), Bit = 1: Steuerleitung soll geändert werden Wort 2: Information, wie die Steuerleitung gesetzt werden soll Bitmap (Wort 1 und 2): Bit 0 = RTS Bit 1 = DTR Bit 2 = TMT-Break Bit 3 bis 15 reserviert
34	F	**Zustand der Steuerleitungs-Eingänge lesen** Hin: outsize = 0 maxinsize = 2 indata_var = *Rückgabe-Struktur* (s.u.) Rück: error = Fehlernummer *Rückgabe-Struktur:* Information, wie die Steuerleitung gesetzt wurde (Word) Bitmap: Bit 0 = CTS Bit 1 = DCD Bit 2 = RI (ab CQ8 Version 2.I) Bit 3 = DSR (ab CQ8 Version 2.I) Bit 4 bis 15 reserviert

Fehlerrückgabecodes von Funktionen des Programmes 520

Alle zurückgelieferten Fehler sind Meldungen vom Betriebssystem. Die folgende Tabelle zeigt, welche Fehler bei den einzelnen Funktionen auftauchen können und was sie bedeuten:

Funktion	Fehlernummer	Bedeutung
5	22h oder 23h	Sendepuffer gesperrt
	26h	Sendepuffer-Nummer ungültig
6	26h	Sendepuffer-Nummer ungültig
7	22h oder 23h	Sendepuffer gesperrt
	24h	Sendepuffer voll
	26h	Sendepuffer-Nummer ungültig
15	22h oder 23h	Empfangspuffer gesperrt
	26h	Empfangspuffer-Nummer ungültig
16	26h	Empfangspuffer-Nummer ungültig
17	22h oder 23h	Empfangspuffer gesperrt
	26h	Empfangspuffer-Nummer ungültig

Bei den Fehlermeldungen 'Puffer voll' bzw. 'Puffer gesperrt' können Sie den Funktionsaufruf zu einem späteren Zeitpunkt wiederholen. Damit sich der Pufferzustand ändern kann, müssen Sie die Kontrolle nach dem ersten Funktionsaufruf (der den Fehler gemeldet hat) zuerst wieder an das Betriebssystem zurückgeben. **!**

13.5. Installation der Basiskommunikation ohne SNW

Für die Installation der Basiskommunikation ohne SNW sind alle Parameter des Programms 520, die mit '*' gekennzeichnet sind (siehe Kapitel 13.4.1.4.), entsprechend zu setzen. Teilweise sind die Parameter vorinitialisiert.

Für synchrone Kommunikation sind zusätzlich die Parameter 16, 17, 32, 33 und 55 relevant.

Hinweise:

- Für die Schnittstellen der Basiskarte und für jedes Modul (M-COM-2 oder M-COM-8) muß jeweils ein Interrupt-Manager (Programm 500) unter dem von dem Modul bzw. der Basiskarte verwendeten Interrupt installiert (II-Task, Installationsflags: 0809h, kein Datenbereich benötigt) werden.
- Für jeden verwendeten Kanal ist jeweils eine Kommunikationstask (Programm 520, NI-Task, Installationsflags: 0808h, Datenbereich wird vom Programm selbst reserviert) zu installieren.
- Die Task-Nummer des zugehörigen Interruptmanagers muß in Parameter 6 jeder Kommunikationstask gesetzt werden. Es empfiehlt sich die Zuordnung wie in Tab. 13.1 vorzunehmen.
- Nach Setzen der gewünschten Parameter der Kommunikationstask und Aufruf der Prozedur 2 ist das Basiskommunikations-Programm konfiguriert und der Speicherplatz für die Puffer entsprechend der in den Parametern angegebenen Größe reserviert.
- Um Zeichen empfangen oder senden zu können, müssen noch die Funktionen 5 (Sendebereitschaft) und 15 (Empfangsbereitschaft) aufgerufen werden.
- Hinweise zum Empfangen: siehe Kapitel 13.1.3 und 13.4.1.2.
- Hinweise zum Senden: siehe Kapitel 13.1.3 und 13.4.1.3.

A. Technische Daten der Basiskarte

CPU:	5x86-P75 (133 MHz), 486DX4/120, 486DX2/66, 486DX/33, 486SX/25 oder /33	
RAM:	"kleine" MODULAR-4/486:	"große" MODULAR-4/486:
	512 KB (statisch) 2 MB (statisch) 10 MB (2 MB stat., 8 MB dyn.) 34 MB (2 MB stat., 32 MB dyn.)	256 KB (statisch) 1 MB (statisch) 4 MB (statisch)
	(statisches RAM batteriepufferbar)	
ROM:	EPROM oder Flash-EPROM, 64 KB bis 512 KB ausbaubar	
EEPROM:	32 Worte, seriell	
Timer:	3 (Baustein 8254), 16 Bit Breite, programmierbare Eingangsfrequenz 1 MHz (nur "kleine" MODULAR-4/486), 2,5 MHz oder 10 MHz, interruptfähig, 3 weitere Timer in SCC (2 Timer) und Echtzeituhr.	
Interrupts:	15 Eingänge, davon 6 an SP-Bus, 1 an Stecker St1, 2 an PC-Schnittstelle, 3 an Timer, 1 an serielle Schnittstellen, 1 an Uhr, 1 über RS-232 Eingang.	
	Zusätzlich bei "kleiner" MODULAR-4/486: Interrupts von Temperaturfühler und Lüfterüberwachung	
Spannungs-überwachung:	Zwei Ansprechschwellen (4,8 und 4,65 Volt), NMI-Auslösung, Pufferung von RAM und Uhr.	
PC-Schnittstelle:	16-Bit parallel, bidirektional, interruptfähig (lokal und PC-Seite), DMA-fähig von und zum PC, 1 MByte/s max. Datentransferrate unidirektional.	
Serielle Schnitt-stellen:	Zwei RS-232 Schnittstellen mit den üblichen Modem-Steuersignalen, 7,3728 MHz Quarzfrequenz (optional: 4,9152 MHz), Baustein SCC 85C30 oder ESCC 85230, Steckerbelegung wie IBM-AT (9-pol. D-Submin.) oder 10-pol. Pfostenstecker	

Echtzeituhr:	Datum (Tag, Monat, Jahr, Wochentag) und Uhrzeit (Std., Min., Sek.), batteriepufferbar, interruptfähig (1/64 sec, 1 sec, 1 min, 1 h)			
Multi-Tasking-Betriebssystem:	Voll echtzeitfähig, max. 1024 Tasks, Interrupt-, Timer-initiierte und Nicht-Interrupt-Tasks. Im EPROM der Karte enthalten, wird ins RAM kopiert, belegt 64 KB RAM.			
Stromaufnahme:	"kleine" MODULAR-4/486:		"große" MODULAR-4/486:	
	+5 V:	TBD A	+5 V:	ca. 1,7 A
	+12 V:	< 1 mA	+12 V	< 1 mA
	-12 V:	< 1 mA	-12 V:	< 1 mA
	-5 V:	nicht benutzt	-5 V:	nicht benutzt
	Werte gelten für Versionen mit 586-133 MHz CPU, serielle Schnittstellen nicht benutzt. Die Stromaufnahme enthält nicht den Strom des Lüfters der CPU.			
Abmessungen:	"kleine" MODULAR-4/486:		"große" MODULAR-4/486	
	160 mm x 106,68 mm		mit kurzem Slotstecker: 337,61 mm x 106,68 mm	
			mit langem Slotstecker: 337,61 mm x 121,92 mm	
	Alle Abmessungen ohne Slotblech und D-Sub-Stecker.			
Temperaturverträglichkeit:	0 bis 55°C			
Luftfeuchtigkeitsverträglichkeit:	5 bis 95% (nicht kondensierend)			

B. Modulübersicht

Die hier aufgeführten Module sind ohne Einschränkungen auf der MODULAR-4/486 und /586 Karte einsetzbar, sofern nichts anderes vermerkt ist. Bitte beachten Sie folgende Hinweise:

Konfiguration der Module

Einige Module müssen per Jumper für die MODULAR-4/486 und /586 Karte konfiguriert werden (z.B. Modul M-5B-1, ab Rev. D). Es ist in jedem Fall die Beschreibung der Module zu beachten.

Revisionsnummer des Moduls

Einige Module sind erst ab einer bestimmten Revisionsnummer des Moduls auf der MODULAR-4/486 Karte einsetzbar, z.B. Modul M-RU8-2 erst ab Rev. D.

Sonderfall Modul M-iNC-3

Dieses Modul erfordert einen 10 MHz Eingangstakt für die Timer A, B und C auf der Basiskarte. Bei "großen" MODULAR-4 Karten der Rev. C und bei den "kleinen" MODULAR-4/486 Karten kann die Umschaltung von 2,5 MHz auf 10 MHz per Software vorgenommen werden. Dazu müssen im EEPROM WORT-3 Bit 4 auf 1 und Bit 5 auf 0 gesetzt werden (siehe Anhang L). Bei Rev. B der "großen" MODULAR-4/486 müssen ein Quarzoszillator und ein IC auf der Basiskarte ausgetauscht werden. Der Umbau kann werkseitig kostenlos durchgeführt werden. Auf andere Module hat dieser Umbau keinen Einfluß, es ist lediglich bei der Programmierung der Timer der geänderte Eingangstakt zu berücksichtigen (10 MHz statt 2,5 MHz).

Typ	Modul	Beschreibung
03	M-D40-2	40 digital I/O, 4 Trigger-Eingänge, 2 Timer-Ausgänge
04[1]	M-SiO-A/i	2 ser. Schnittst., galv. getr. 20 mA (ersetzt durch Typ 32)
05[1]	M-SiO-A/R	2 serielle Schnittstellen, RS-232 (ersetzt durch Typ 32)
06	M-OPT-1/A	16 galvanisch getrennte digitale Eingänge
07	M-OPT-1/B M-OPT-1/Bx	16 galvanisch getrennte digitale Ausgänge wie M-OPT-1/B, aber mit 100 mA Ausgängen
08	M-DA2-2	2 galvanisch getrennte analoge Ausgänge, 12 Bit Auflösung
09	M-DA4-2	4 analoge Ausgänge, 12 Bit Auflösung
10[2]	M-5B-1/M	ersetzt durch Typ 20
12[2]	M-AD16-3	16 analoge Eingänge, 12 Bit, ersetzt durch Typ 39
13	M-RU8-2	8 Relaisausgänge (1 x Um). Für MODULAR-4/486 ab Modul-Revision D verwendbar
14[1,2]	M-SiO-i	ersetzt durch Typ 32
16[1]	M-SM-1	Schrittmotorsteuerung, 2 oder 4-Phasen
19	M-IEC-1	IEC-Bus Schnittstelle
20	M-5B-1/U	Ankopplung für 5B-Meßumformer (Ein- und Ausgänge), 4 Analog-In + 1 Analog-Out + 14 Digital-Out
21[2]	M-SCC-1	ersetzt durch Typ 32
22[2]	M-SiO-8	8 serielle RS-232 Schnittstellen (für MODULAR-4/486 nur Rev. E, für Neuentwicklungen Typ 48 verwenden)
24	M-SH12-8	8 analoge Eingänge mit Simultanabtastung, 1 analoger Ausgang, 2 Triggereingänge
25[1]	M-SiO-A/iR	2 ser. Schnittst.: RS-232 und 20 mA, ersetzt durch Typ 32
26[1]	M-422-2/A	2 x RS-422, asynchron, ersetzt durch Typ 32
27[1]	M-422-2/S	2 x RS-422, synchron, ersetzt durch Typ 32
28[2]	M-iNC-3	Inkrementalgeber/Zähler, 3 Kanäle, optoisoliert (Timertakt 10 MHz erforderlich, s.o.), ersetzt durch Typ 47
29	M-DA16-2	2 galvanisch getrennte analoge Ausgänge, 16 Bit Auflösung

[1] Nicht für MODULAR-4/486 verwendbar
[2] Verbesserte Version verfügbar (kompatibel)

Typ	Modul	Beschreibung
30[2]	M-AD16-8	8 galvanisch getrennte analoge Eingänge 16-Bit, Doppelmodul, ersetzt durch Typ 43 (Einfachmodul)
32	M-COM-2	2 serielle Schnittstellen (SCC 8530) mit Modem-Signalen. Durch C-Links je Kanal konfigurierbar als RS-232, RS-232 isoliert, RS-422, RS-485, RS-485 isoliert, 20 mA
33	M-COM-2/P M-COM-2/G	2 ser. Schnittstellen für Lichtwellenleiter (Plastik) 2 ser. Schnittstellen für Lichtwellenleiter (Glas)
34	M-DC15-2	4 Interrupt-Eingänge, DC/DC-Wandler für ±15 Volt, 8 Kontroll-LEDs
36[1]	M-422-2	2 x RS-422 synchron, asynchron, ersetzt durch Typ 32
37	M-AX-16	Programmierbares Gate-Array, 12 optoisolierte Eingänge, 4 optoisolierte Ausgänge
38	M-AX-32	Programmierbares Gate-Array, 32 digital-I/O (TTL)
39	M-AD12-16	16 analoge Eingänge, 12 Bit, 16 verschiedene Eingangsbereiche je Kanal per Software einstellbar
40[3]	M-X40-1	Modul-Adapter für X40-BUS Module
41	ML8-EX	Modulerweiterung auf insgesamt 9 Steckplätze
43	M-AD16-4	4 galvanisch getrennte analoge Eingänge, 16 Bit Auflösung
44	M-DPM-12	PROFIBUS-Master, 1 Kanal, bis 12 MBit/s
45	M-DPS-12	PROFIBUS-Slave, 2 Kanäle, bis 12 MBit/s
46	M-ETH-1	Ethernet-Anschluß (AUI und Twisted-Pair)
47	M-C16-3	3 kaskadierbare 16-Bit-Zähler, 12 Opto-Eingänge, 4 Opto-Ausgänge
48	M-COM-8	8 serielle RS-232 Schnittstellen (4x SCC 8530)
49	M-CAN-1	CAN-Bus-Interface
50	M-DAS-A	2 ser. Schnittstellen, Lichtwellenleiter und RS-232
51	M-C16-1	wie M-C16-3, aber 1 Kanal, 16 Bit
52	M-SSI-2	Synchron-serielles Interface (2 Kanäle) für Absolut-Weggeber mit RS-422-Signalen

[1] Nicht für MODULAR-4/486 verwendbar
[2] Verbesserte Version verfügbar (kompatibel)
[3] Noch nicht verfügbar

Notizen

C. Lokale I/O-Adressen

Die folgende Aufstellung soll als Übersicht dienen. Wenn Sie eigene Anwendungsprogramme in 486-Assembler oder in anderen Programmiersprachen schreiben wollen, finden Sie hierzu Hinweise und Beispiele in den Kapiteln 7 bis 13 dieses Handbuchs.

Die 486er CPU auf der Karte kann Programme, die für 8086, 80186, 80286, 80386 oder 80486 geschrieben sind, verarbeiten. Die CPU-internen Register sind hier nicht aufgeführt. Bitte benutzen Sie hierfür die umfangreiche Literatur, die zu diesem Thema auf dem Markt erhältlich ist. Das gleiche gilt auch für die Chips der Basiskarte, da es sich dabei ebenfalls um Standardchips handelt.

Auf alle Devices kann mit 8 Bit, auf einige darf auch mit 16 Bit (PC-Schnittstelle, Module und RAM) und 32 Bit (RAM) zugegriffen werden. Der Zugriff auf einige Devices, z. B. die Uhr und den SCC, ist komplexer und sollte nur über die System-Subroutinen erfolgen. Die Devices auf der Basiskarte werden alle mit I/O-Adressen von 00h bis ffh angesprochen.

Alle Adressenangaben sind hexadezimal. Die Angaben in der Spalte "Zugriff" zeigen die erlaubten Zugriffe an, ein angehängtes "x" bedeutet, daß die gelesenen bzw. geschriebenen Daten ungültig bzw. ohne Bedeutung sind:

W = nur Schreibzugriffe sind erlaubt
R = nur Lesezugriffe sind erlaubt
RW = Schreib- und Lesezugriffe sind erlaubt
8 = nur Bytezugriff erlaubt
16 = nur Wortzugriff erlaubt
x = gelesene bzw. geschriebene Daten sind ohne Bedeutung

Folgende Devices sind auf der Basiskarte verfügbar:

PC-Schnittstelle inkl. Konfiguration
Interrupt-Controller 1 und 2 (8259)
Einstellung der aktiven Flanken für einige Interrupts
Timer/Counter (8254): Timer A, Timer B und Timer C
Serielle Schnittstellen (SCC 8530): 2 serielle Kanäle mit Timer-E und -F
Uhr (RTC-72421) mit Timer-D
Watchdog
Überwachung der +5-Volt-Versorgungsspannung
Leuchtdioden LED1 (extern) und LED2 (on-board)
Serielles EEPROM (auch auf jedem Modul vorhanden)
Lüfter und Temperaturüberwachung der CPU[1]
SP-Bus (Module 1 und 2 bzw. 1 bis 4)

PC-Schnittstelle

(Funktion änderbar durch Bestückungsoption)

Adresse	Zugriff	Funktion	
28h	R8/R16	Status 1:	Bit 0 = RBF
			Bit 1 = DLM
			Bit 6 = DLP
			Bit 7 = TBF
28h	W8/W16	Sende Datenbyte/-wort an PC	
24h	R8/R16	Lies Datenbyte/-wort von PC	
42h	W8x	Setze Device Locking Bit DLP = 0[2]	
43h	W8x	Setze Device Locking Bit DLP = 1	
2ah	R8/R16	Status 2:	Bit 7 = RESTART[1]
			Bit 0 = ACTIVE[1]

[1] Nur bei "kleiner" MODULAR-4/486 verfügbar

[2] Zustand nach Reset

PC-Interrupt und PC-DMA

Die Funktionen von 5 Steuerleitungen sind durch Bestückungsoptionen änderbar. Die Standardfunktionen sind hier angegeben.

Adresse	Zugriff	Funktion
52h, 53h	W8x	PC-Interrupt-Leitung vom PC-Bus trennen[1]/verbinden.
55h, 54h	W8x	Interrupt-Request zum PC durch 2 aufeinanderfolgende Zugriffe (zuerst 55h, dann 54h) auslösen.
8nh	W8x	Anwahl einer PC-Interrupt-Leitung, n = Nr. der Interrupt-Leitung IRQ-... (erlaubt sind n = 3, 4, 5, 7, 9, 0ah (=10), 0bh (=11), 0ch (=12))
58h	W8x	DMA-Leitungen vom PC-Bus abtrennen[1].
59h	W8x	DMA-Leitungen an PC-Bus anschließen und DMA-Request zum PC entsprechend eingestellter Richtung (siehe 5ah / 5bh), sofern der DMA-Mechanismus auf der Karte enabled ist.
56h	W8x	DMA-Mechanismus auf der Karte enable.
57h	W8x	DMA-Mechanismus auf der Karte disable. Die DMA-Leitungen bleiben auf definiertem, inaktivem Pegel.
5ah	W8x	DMA-Richtung einstellen: DMA-Request zum PC, wenn Daten von ML8 an PC geschickt werden[1].
5bh	W8x	DMA-Richtung einstellen: DMA-Request zum PC, wenn Daten, die der PC an ML8 geschickt hat, von ML8 gelesen werden.

[1] Zustand nach Reset

Verwendung der on-board Interrupts

	IRQ-	Funktion
Master	0	von PC-Schnittstelle (Receive Buffer Full)
	1	= IRQ-F (SPBus), aktive/r Flanke/Pegel programmierbar bzw. IRQ-TEMP von Temperatursensor (alternativ)
	2	von Slave Controller (Interrupt-Controller 2)
	3	= IRQ-A (SPBus), aktive/r Flanke/Pegel programmierbar
	4	= IRQ-B (SPBus), aktive/r Flanke/Pegel programmierbar
	5	= IRQ-C (SPBus), aktive/r Flanke/Pegel programmierbar
	6	= IRQ-D (SPBus), aktive/r Flanke/Pegel programmierbar
	7	= IRQ-E (SPBus), aktive/r Flanke/Pegel programmierbar bzw. IRQ-FAN von Lüfterüberwachung (alternativ)
Slave	0	von Interrupt-Ausgang des SCC (Z8530)
	1	von OUT-Ausgang Timer A
	2	von OUT-Ausgang Timer B
	3	von OUT-Ausgang Timer C
	4	von Interrupt-Ausgang der Uhr (Timer D) ("große" MODULAR-4/486, Rev. B) bzw. invertierter Interrupt-Ausgang bei allen ander Versionen
	5	= IRQ-G: Externer Interrupt-Eingang (an St1), aktive/r Flanke/Pegel programmierbar
	6	= IRQ-H: von Ri der seriellen Schnittstelle B, aktive/r Flanke/Pegel programmierbar
	7	von PC-Schnittstelle (Transmit Buffer Empty)

Interrupt-Controller 1 (Master)

(siehe Datenblatt 8259)

Adresse	Zugriff	Funktion
00h	R8	Lies IRR (Interrupt Request Register) bzw. ISR (Interrupt Service Register)
01h	R8	Lies IMR (Interrupt Mask Register)
00h	W8	Schreibe IW1 (D4 = 1), PFCW (D3 = 0, D4 = 0) oder MCW (D3 = 1, D4 = 0)
01h	W8	Schreibe IW2, IW3 oder IW4 bzw. nach Init IMW
68h	W8x	IRQ-A: bei High bzw. pos. Flanke[1]
69h	W8x	bei Low bzw. neg. Flanke
6ah	W8x	IRQ-B: bei High bzw. pos. Flanke[1]
6bh	W8x	bei Low bzw. neg. Flanke
6ch	W8x	IRQ-C: bei High bzw. pos. Flanke[1]
6dh	W8x	bei Low bzw. neg. Flanke
6eh	W8x	IRQ-D: bei High bzw. pos. Flanke[1]
6fh	W8x	bei Low bzw. neg. Flanke
4eh	W8x	IRQ-E/IRQ-FAN: bei High bzw. pos. Flanke[1]
4fh	W8x	bei Low bzw. neg. Flanke
4ch	W8x	IRQ-F/IRQ-TEMP: bei High bzw. pos. Flanke[1]
4dh	W8x	bei Low bzw. neg. Flanke
98h	W8x	IRQ-E[1] verbinden mit Master IRQ-7
99h	W8x	IRQ-FAN[2] verbinden mit Master IRQ-7
9ch	W8x	IRQ-F[1] verbinden mit Master IRQ-1
9dh	W8x	IRQ-TEMP[2] verbinden mit Master IRQ-1

[1] Zustand nach Reset

[2] Nur bei "kleiner" MODULAR-4/486 verfügbar

Interrupt-Controller 2 (Slave)

(siehe Datenblatt 8259)

Adresse	Zugriff	Funktion	
08h	R8	Lies IRR (Interrupt Request Register) bzw. ISR (Interrupt Service Register)	
09h	R8	Lies IMR (Interrupt Mask Register)	
08h	W8	Schreibe IW1 (D4=1), PFCW (D3 = 0, D4 = 0) oder MCW (D3 = 1, D4 = 0)	
09h	W8	Schreibe IW2, IW3 oder IW4 bzw. nach Init IMW	
48h	W8x	Ext. Int (IRQ-G):	bei High bzw. pos. Flanke[1]
49h	W8x		bei Low bzw. neg. Flanke
4ah	W8x	IRQ-RiB (IRQ-H):	bei High bzw. pos. Flanke[1]
4bh	W8x		bei Low bzw. neg. Flanke

Timer/Counter A, B und C

(siehe Datenblatt 8254)

Adresse	Zugriff	Funktion
91h	W8x	Timereingangstakt auf 1 MHz stellen[2]
90h	W8x	Timereingangstakt auf 2,5 oder 10 MHz stellen[2] (siehe Adresse 40h bzw. 41h)
40h	W8x	Timereingangstakt auf 2,5 MHz bzw. 1 MHz stellen (siehe Adresse 90h bzw. 91h)
41h	W8x	Timereingangstakt auf 10 MHz stellen
10h	RW8	Counter/Timer A Data
11h	RW8	Counter/Timer B Data
12h	RW8	Counter/Timer C Data
13h	RW8	Control-Register

[1] Zustand nach Reset

[2] Nur bei "kleiner" MODULAR-4/486 verfügbar

Serielle Schnittstellen A und B

(siehe Datenblatt Z8530 SCC)

Für den seriellen Schnittstellenbaustein Z8530 SCC steht ein ca. 280 Seiten umfassendes Handbuch "SCC User's Manual" (engl.) mit Hinweisen zur Programmierung zur Verfügung.

Adresse	Zugriff	Funktion
32h	RW8	Kanal A: Control-Byte
33h	RW8	Data-Byte
30h	RW8	Kanal B: Control-Byte
31h	RW8	Data-Byte
28h	R8/R16	Status der Modem-Steuerleitungen[1] lesen: Kanal A: Bit 2 = Ri, Bit 3 = DSR Kanal B: Bit 4 = Ri, Bit 5 = DSR

LED 1 (ext.) und LED 2 (on-board)

Adresse	Zugriff	Funktion
50h	W8x	ext. LED 1 ein[2]
51h	W8x	ext. LED 1 aus
5ch	W8x	on-board LED 2 ein[2]
5dh	W8x	on-board LED 2 aus

[1] Die hier nicht angegebenen Modem-Steuerleitungen werden über Register des seriellen Controllers SCC (Z8530) angesprochen.

[2] Zustand nach Reset, kann aber nach Starten des Betriebssystems abhängig vom EEPROM-Inhalt geändert werden.

Uhr

(siehe Datenblatt RTC-72421)

Für den Zugriff ist eine besondere Prozedur erforderlich. Deshalb sollten hierfür nur die im Betriebssystem vorhandenen Subroutinen GET_RTC_STATUS, SET_RTC_MODE, GET_DATE_AND_TIME und SET_DATE_AND_TIME verwendet werden. Jeder Zugriff auf die Uhr triggert auch den Watchdog-Timer, sofern der Watchdog-Timer mit J4 (bei "großer" MODULAR-4/486) bzw. per Software (bei "kleiner" MODULAR-4/486) aktiviert ist.

Adresse	Zugriff	Funktion	
70h	spez.	Uhr:	Sekunden-Einer
71h	spez.		Sekunden-Zehner
72h	spez.		Minuten-Einer
73h	spez.		Minuten-Zehner
74h	spez.		Stunden-Einer
75h	spez.		Stunden-Zehner (+AM/PM)
76h	spez.	Datum:	Tag-Einer
77h	spez.		Tag-Zehner
78h	spez.		Monat-Einer
79h	spez.		Monat-Zehner
7ah	spez.		Jahr-Einer
7bh	spez.		Jahr-Zehner
7ch	spez.		Wochentag (0 bis 6)
7dh	RW8	Control-Register D (nur Bit 0 bis 3 gültig)	
7eh	RW8	Control-Register E (nur Bit 0 bis 3 gültig)	
7fh	RW8	Control-Register F (nur Bit 0 bis 3 gültig)	

Watchdog-Timer und NMI

"Große" MODULAR-4/486:

Der Watchdog-Timer wird mit Jumper J4 aktiviert. Wenn er aktiviert ist, muß er über einen I/O-Lesezugriff regelmäßig nachgetriggert werden, andernfalls erfolgt bei Revision B ein Hardware-Reset der Karte, bei Revision C ein NMI (sofern enabled). Ein NMI kann auch, sofern enabled, von den Modulen oder der on-board Spannungsüberwachung kommen (Einstellung über J4, siehe auch Kapitel 2).

Adresse	Zugriff	Funktion
7fh	R8	Watchdog-Timer nachtriggern
44h	W8x	NMI abgeschaltet[1]
45h	W8x	NMI enabled

"Kleine" MODULAR-4/486:

Der Watchdog-Timer wird per Software (s.u.) aktiviert. Wenn er aktiviert ist, muß er über einen I/O-Lesezugriff regelmäßig nachgetriggert werden, andernfalls erfolgt ein NMI (sofern enabled). Ein NMI kann auch, sofern aktiv, von den Modulen oder der on-board Spannungsüberwachung kommen (EEPROM-Einstellung, siehe auch Anhang L).

Adresse	Zugriff	Funktion
94h	W8x	Watchdog-Timer disable[2]
95h	W8x	Watchdog-Timer enable
7fh	R8	Watchdog-Timer nachtriggern
44h	W8x	NMI abgeschaltet[2]
45h	W8x	NMI enabled
28h	R16	NMI-Interruptquelle (nur Bit-9 und 10 sind relevant) Bit-10= 1: Watchdog-Timer hat NMI verursacht Bit-9 = 0: Spannungsüberwachung hat den NMI ausgelöst

[1] Zustand nach Reset

EEPROM-Anwahl-Adressen

Adresse	Zugriff	Funktion	
5eh	W8x	EEPROM Basiskarte:	Abwahl[1]
5fh	W8x		Anwahl
60h	W8x	EEPROM Modul 1:	Abwahl[1]
61h	W8x		Anwahl
62h	W8x	EEPROM Modul 2:	Abwahl[1]
63h	W8x		Anwahl
64h	W8x	EEPROM Modul 3:	Abwahl[1]
65h	W8x		Anwahl
66h	W8x	EEPROM Modul 4:	Abwahl[1]
67h	W8x		Anwahl
401h	W8x	EEPROM Modul 5:	Abwahl[1]
400h	W8x		Anwahl
403h	W8x	EEPROM Modul 6:	Abwahl[1]
402h	W8x		Anwahl
405h	W8x	EEPROM Modul 7:	Abwahl[1]
404h	W8x		Anwahl
407h	W8x	EEPROM Modul 8:	Abwahl[1]
406h	W8x		Anwahl
409h	W8x	EEPROM Modul 9:	Abwahl[1]
408h	W8x		Anwahl
40bh	W8x	EEPROM Modul 10:	Abwahl[1]
40ah	W8x		Anwahl
6ah	W8x	Schreibzugriff:	Aktivieren
6bh	W8x		Deaktivieren
68h	W8x	Schreibdaten:	= 0 setzen
69h	W8x		= 1 setzen
28h	R16	Lies EEPROM-Data (nur Bit 8 ist gültig)	

[1] Zustand nach Reset

FPGA-Versionen (nur bei "kleiner" MODULAR-4/486)

Adresse	Zugriff	Funktion
bnh (b0h - b7h)	R8	Lies FPGA-Version IC3, 8 Bit, je Zugriff liefert Bit 0 ein Bit der Versions-Nr., wobei n = Wertigkeit des Bit (aktuelle Version = 01)
cnh (c0h - c7h)	R8	Lies FPGA-Version IC4, 8 Bit, je Zugriff liefert Bit 0 ein Bit der Versions-Nr., wobei n = Wertigkeit des Bit (aktuelle Version = 01 bzw. 02)

Lüfter (nur bei "kleiner" MODULAR-4/486)

Adresse	Zugriff	Funktion
92h	W8x	Fan off
93h	W8x	Fan on

Der Lüfter ist unabhängig von dieser Funktion "on", wenn der Temperatursensor bei Karten ab Rev. B Übertemperatur anzeigt.

Modul-Basis-Adressen (MBA)

Auf der "kleinen" MODULAR-4/486 stehen nur die Steckplätze 1 und 2 zur Verfügung. Die Steckplätze 5 bis 10 befinden sich auf dem Modul-Extender, der nur bei der "großen" MODULAR-4/486 einsetzbar ist.

Modulsteckplatz	Adresse I/O-Bereich	Zugriff	Busbreite (Bit) (Modul-abhängig)
Modul 1	400h	RW8/RW16	8 bzw. 16
Modul 2	500h	RW8/RW16	8 bzw. 16
Modul 3	600h	RW8/RW16	8 bzw. 16
Modul 4	700h	RW8/RW16	8 bzw. 16
Modul 5	420h	RW8/RW16	8 bzw. 16
Modul 6	440h	RW8/RW16	8 bzw. 16
Modul 7	460h	RW8/RW16	8 bzw. 16
Modul 8	480h	RW8/RW16	8 bzw. 16
Modul 9	4a0h	RW8/RW16	8 bzw. 16
Modul 10	4c0h	RW8/RW16	8 bzw. 16

D. Lokale Interrupts der MODULAR-4/486 Karte

Interrupt-Nummer	Typ	Quelle/Erklärung
0 (00h)	Fault	CPU: Divide Error (z.B. / 0)
1 (01h)	Fault	CPU: Debug Exception (Trap/Fault)
2 (02h)	NMI	Ext.: INT 2 oder NMI
3 (03h)	Trap	CPU: Soft-Interrupt (INT 3) (1-Byte Opcode)
4 (04h)	Trap	CPU: Overflow (INT 0)
5 (05h)	Fault	CPU: Array Bounds Check
6 (06h)	Fault	CPU: Invalid Opcode
7 (07h)	Fault	CPU: Device Not Available
8 (08h)	Abort	CPU: Double Fault
9 (09h)	-	Reserviert (Intel)
10 (0ah)	Fault	CPU: Invalid TSS (Protected Mode)
11 (0bh)	Fault	CPU: Segment Not Present
12 (0ch)	Fault	CPU: Stack Fault
13 (0dh)	Fault	CPU: General Protection Fault
14 (0eh)	Fault	CPU: Page Fault
15 (0fh)	-	Reserviert (Intel)
16 (10h)	Fault	CPU: Floating Point Error (tritt bei i486SX nicht auf)
17 (11h)	Fault	CPU: Alignment Check
18 - 31 (12h - 1fh)	-	Reserviert (Intel)
32 - 119 (20h - 77h)	Trap	Soft-Interrupt (2-Byte Opcode), reserviert für SORCUS
120 - 127 (78h - 7fh)	Hardware	**Master-Interrupt-Controller** auf der Basiskarte: Master-0 bis Master-7 (siehe Tab. Seite D-2)
128 - 143 (80h - 8fH)	Hardware	Reserviert für weitere Slave-Interrupt-Controller
144 - 151 (90h - 97h)	Hardware	**Slave-Interrupt-Controller** auf der Basiskarte an Master-2 vom Master-Interrupt-Controller (siehe Tabelle Seite D-2)

Interrupt-Nummer	Typ	Quelle/Erklärung
152 - 191 (98h - bfh)	Hardware	Reserviert für weitere Slave-Interrupt-Controller
192 - 207 (c0h - cfh)	Hardware	**Z8530 SCC** auf der Basiskarte an IRQ-0 von Slave-Interrupt-Controller (= Slave 2-0)
208 - 239 (d0h - efh)	Trap	Soft-Interrupt (2-Byte Opcode), reserviert für SORCUS
240 - 255 (f0h - ffh)	Trap	Soft-Interrupt (2-Byte Opcode), frei für Anwender

Hardware Interrupts

Priorität	Interrupt Dez.	Hex	Quelle	Beschreibung	Interrupt-Eingang
Hoch	2	02h	NMI	Nicht maskierbarer Interrupt	NMI (CPU)
	120	78h	PC-RBF	Befehl vom PC	Master-0
	121	79h	IRQ-F/TEMP	Interrupt-Eingang (SP-Bus[1])	Master-1
	122	7ah	Slave	von Slave-Controller	Master-2
	144	90h	SCC	Serielle Schnittstellen A und B	Slave2-0
	145	91h	TIMER-A	Zeitgeber-A	Slave2-1
	146	92h	TIMER-B	Zeitgeber-B	Slave2-2
	147	93h	TIMER-C	Zeitgeber-C	Slave2-3
	148	94h	UHR	Taktgeber in der Uhr	Slave2-4
	149	95h	IRQ-G	Ext. Interrupt-Eingang	Slave2-5
	150	96h	IRQ-H	Ri/B Interrupt-Eingang	Slave2-6
	151	97h	PC-TBE	PC-Schnittstelle leer	Slave2-7
	123	7bh	IRQ-A	Interrupt-Eingang (SP-Bus)	Master-3
	124	7ch	IRQ-B	Interrupt-Eingang (SP-Bus)	Master-4
	125	7dh	IRQ-C	Interrupt-Eingang (SP-Bus)	Master-5
	126	7eh	IRQ-D	Interrupt-Eingang (SP-Bus)	Master-6
Niedrig	127	7fh	IRQ-E/FAN	Interrupt-Eingang (SP-Bus[2])	Master-7

[1] Kann per Software bzw. EEPROM-Konfiguration an den Ausgang des Temperatursensors gelegt werden.

[2] Kann per Software bzw. EEPROM-Konfiguration an die Überwachung des Lüfters gelegt werden.

E. Fehlermeldungen von PC-Bibliotheken

Beim Einsatz von MODULAR-4/486 Karten und den dazugehörigen Bibliotheken können die folgenden Fehlermeldungen auftreten.

Alle nicht erwähnten Fehlercodes sind reserviert.

Neue Fehlermeldungen, die zum Zeitpunkt der Drucklegung noch nicht festlagen, können Sie der Datei 'readme.doc' im Verzeichnis Ihrer PC-Bibliotheken entnehmen. **!**

Fehlerklasse 0: Kommunikationsfehler

Timeout-Fehler (Fehlercode 1 oder 2) treten auf, wenn eine Karte nicht innerhalb einer voreingestellten Zeit empfangs- oder sendebereit ist. Bei den Funktionen **ml7_start** bzw. **ml8_start** und **ml7_reset** bzw. **ml8_reset** wird ein Timeout (in Zehntelsekunden) eingestellt. Der Wert 10 (= 1 Sekunde) sollte in der Regel ausreichen. Mit **ml7_change_timeout** bzw. **ml8_change_timeout** kann der Timeout erhöht werden. Ein Timeout-Fehler entsteht auch, wenn sich unter der eingestellten Basisadresse keine Karte im PC befindet oder unter dieser Adresse schon eine andere Karte installiert ist.

Ein Plausibilitätsfehler (Fehlercode 3) entsteht, wenn das erste Byte der Antwort eines Makrobefehls nicht mit dem gesendeten Makrobefehls-Code übereinstimmt. Dabei ist zu berücksichtigen, daß diese Situation auch entstehen kann, wenn der vorangegangene Makrobefehl ein oder mehrere "überschüssige" Antwort-Byte in der Schnittstelle zurückgelassen hat, bzw. diese vom PC nicht aus der Schnittstelle entnommen wurden. Ein fehlerhaftes Echtzeitprogramm auf der Karte, das diese zum "Absturz" gebracht hat, kann ebenfalls zu einem Plausibilitätsfehler führen.

Fehlercode		Bedeutung des Fehlercodes der Fehlerklasse 0
0	0h	Kein Fehler
1	1h	TBF-Fehler Bei einem Sendeversuch des PC wurde die Schnittstelle zur Karte nicht innerhalb der Timeout-Zeit frei
2	2h	RBF-Fehler Bei einem Empfangsversuch konnte der PC innerhalb der Timeout-Zeit kein Byte von der Schnittstelle zur Karte lesen

Fehlercode		Bedeutung des Fehlercodes der Fehlerklasse 0
3	3h	Plausibilitätsprüfung gescheitert Das erste Antwort-Byte eines Makrobefehls war falsch
4	4h	DLP-Timeout zu Beginn eines Makrobefehls (DLP = 1)
5	5h	Unerwartetes Wort (RBF=1) zu Beginn eines Makrobefehls
6-7	6h-7h	Reserviert
8	8h	DLP konnte nicht auf Null gesetzt werden
9	9h	DLP konnte nicht auf Eins gesetzt werden
11	bh	TBF-Reset-Fehler Das Status-Bit TBF blieb nach Reset auf eins
14	eh	TBF-Reset-Fehler Das TBF-Status-Bit konnte nicht = 1 gesetzt werden
15	fh	DLP = 1 beim Empfangen der Daten
16	10h	DLP = 1 beim Senden der Daten
18	12h	Die Karte reagiert nicht
19	13h	RBF = 0 beim Empfangen eines Interrupts
20	14h	TBF = 1 beim Empfangen eines Interrupts
21	15h	Fehler am Anfang eines Makrobefehls

Fehlerklasse 1: Dateizugriffe und Betriebssystem laden

Fehlercode		Bedeutung des Fehlercodes der Fehlerklasse 1
0	0h	Kein Fehler
1	1h	Mini-Betriebssystem ist nicht geladen
2	2h	ROM-Betriebssystem konnte nicht aktiviert werden
3	3h	RAM Betriebssystemdatei nicht gefunden
4	4h	RAM Betriebssystemdatei: Lesefehler
5	5h	RAM Betriebssystemdatei: Fehler beim Öffnen/Schließen
6	6h	RAM Betriebssystem konnte nicht aktiviert werden
7-16	7h-10h	Reserviert
17	11h	Programmdatei nicht gefunden
18	12h	Speicher-Reservierungsfehler auf der Karte
19	13h	Zuwenig freier Speicher auf der Karte
20	14h	Programmdatei Lesefehler
21	15h	Programmdatei Fehler beim Öffnen/Schließen
22	16h	Falsche Linker-Signatur
23	17h	Karte läßt sich nicht initialisieren
24	18h	Debug-Tabelle konnte nicht eingerichtet werden
32	20h	Kein Modul auf dem Steckplatz vorhanden
33	21h	Modul braucht keine Initialisierung
34	22h	Modul wurde nicht initialisiert wegen EEPROM-Konfiguration
35	23h	Initialisierungsroutine ist nicht implementiert
64	40h	Bootfehler: Adresse konnte nicht eingestellt werden
65	41h	Bootfehler: Zugriffsmodus 'Wort' nicht möglich
66	42h	Bootfehler: Zugriffsmodus nicht wie erwartet
67	43h	Bootfehler: unbekannter Fehler bei Zugriffsmodus
68	44h	Bootfehler: Fehler beim Schreiben des OsX oder beim Starten
69	45h	Bootfehler: Fehler beim Rücksetzen des Befehlszeigers

Fehlerklasse 2: Unzulässige Übergabeparameter

Fehlercode		Bedeutung des Fehlercodes der Fehlerklasse 2
0	0h	Kein Fehler
1	1h	Kartennummer unzulässig (erlaubt sind 0 bis 7)
2	2h	Basisadresse unzulässig (entweder außerhalb 0 bis 3fch oder Wert nicht durch 8 teilbar)
3	3h	Betriebsart unzulässig
4	4h	PC-Interrupt-Kanal unzulässig
6	6h	Die Bibliothek ist für diese Karte nicht initialisiert. Verwenden Sie **ml7_reset/ml8_reset** oder **ml7_start/ml8_start**.
7	7h	Programmtyp und Installierungsflag (Bit 6-8) stimmen nicht überein (z. B. keine EXE-Datei).
8	8h	Falscher Kartentyp
16	10h	Unzulässiger Parameter (Datum/Zeit)
17	11h	Datenrückgabelänge überschreitet den Vorgabewert (**ml7_call_func/ml8_call_func**)
18	12h	Wortnummer für EEPROM ungültig
25	19h	Wortnummer für EEPROM ungültig
32	20h	Aufruf einer Bibliotheksfunktion ohne vorhergehenden Aufruf von **ml7_bib_startup** bzw. **ml8_bib_startup**
33	21h	Aufruf einer Multi-LAB/2 Funktion für eine MODULAR-4/486 Karte
34	22h	Aufruf einer MODULAR-4/486 Funktion für eine Multi-LAB/2 Karte
35	23h	Aufruf einer Funktion, die (noch) nicht unterstützt wird
36	24h	Multi-LAB/2a unterstützt Funktion nicht
37	25h	Multi-LAB/2d unterstützt Funktion nicht
38	26h	Parameter #1 ist falsch
39	27h	Parameter #2 ist falsch
40	28h	Parameter #3 ist falsch
41	29h	Funktion wird von Karte nicht unterstützt

Fehlercode		Bedeutung des Fehlercodes der Fehlerklasse 2
48	30h	Maximale Makro-Kommandolänge überschritten
49	31h	Maximale Makro-Kommandorückgabelänge überschritten
50	32h	Erwartete Funktions-Datenlänge (Hin) überschritten
51	33h	Erwartete Funktions-Datenlänge (Rückgabe) überschritten
54	36h	Der angegebene Makrobefehl ist unbekannt

Fehlerklasse 4: Fehler im PC-Interrupt-Service

Fehlercode		Bedeutung des Fehlercodes der Fehlerklasse 4
0	0h	Kein Fehler
1	1h	TBF-Fehler Bei einem Sendeversuch des PC wurde die Schnittstelle zur Karte nicht innerhalb der Timeout-Zeit frei
2	2h	RBF-Fehler Bei einem Empfangsversuch konnte der PC innerhalb der Timeout-Zeit kein Byte von der Schnittstelle zur Karte lesen
3	3h	Plausibilitätsprüfung gescheitert Das erste Antwort-Byte eines Makrobefehls war falsch
14	eh	Service Request ohne RBF
15	fh	DLP war nicht gesetzt in der PC-Interrupt-Service-Routine
16	10h	Rekursiver Aufruf der Interrupt Service Routine
17	11h	Message-Überlauf in der PC-Interrupt-Service-Routine
18	12h	TBF-Timeout bei DLP-Reset Makrobefehl 12h in der PC-Interrupt-Service-Routine

Fehlerklasse 5: Fehlerrequest der Karte

Bei dieser Fehlerklasse ist der Fehlercode nicht gültig. Die Fehlerinformation muß statt dessen mit der Funktion **ml7_get_message** bzw. **ml8_get_message** gelesen werden. Die Bedeutung des Request-Wortes entnehmen Sie bitte der Tabelle in Anhang F.

Fehlerklasse 7: Spezielle Fehler in Windows 95 und Windows NT

Fehlercode		Bedeutung des Fehlercodes der Fehlerklasse 7
1	1h	Systemfehler aufgetreten. Der Fehlercode wurde bereits von der Bibliothek aus dem System gelesen. Nach einem Aufruf von **ml7_get_error_info** bzw. **ml8_get_error_info** befindet sich der System-Fehlercode in <ErrorStruct>.system.error.
2	2h	Treiberfehler

Fehlerklasse 8: Gerätetreiberfehler

Fehlercode		Bedeutung des Fehlercodes der Fehlerklasse 8
1	1h	Treiber konnte nicht geöffnet werden
2	2h	Rückgabelänge des Treibers falsch
3	3h	SRQ-Thread konnte nicht aktiviert werden
4	4h	SRQ-Event konnte nicht aktiviert werden
5	5h	SRQ-Hold-Request konnte nicht installiert werden
6	6h	SRQ-Thread: Priorität kann nicht erhöht werden
7	7h	User-Service-Thread konnte nicht aktiviert werden
8	8h	SRQ-Synchronisation: Ereignis konnte nicht geöffnet werden
9	9h	SRQ-Synchronisation: Ereignis konnte nicht gesetzt werden
10	ah	SRQ-Synchronisation: Ereignis konnte nicht zurückgesetzt werden
11	bh	Prozeßspeicher im Treiber konnte nicht angelegt werden
12	ch	Prozeßspeicher im Treiber konnte nicht freigegeben werden
13	dh	Prozeß konnte nicht mit dem Treiber verbunden werden
14	eh	SRQ-Thread konnte nicht angehalten werden
16	10h	Falscher Parameter beim Lesen/Schreiben im Treiber
17	11h	Falscher Parameter beim Lesen im Treiber
18	12h	Falscher Parameter beim Schreiben im Treiber
19	13h	Parameter kann nicht geschrieben werden
20	14h	Parametertyp wurde falsch angegeben

Fehlercode		Bedeutung des Fehlercodes der Fehlerklasse 8
21	15h	Prozeß-ID ist nicht gültig
22	16h	Protokollpuffer kann nicht gelesen werden
23	17h	Kartennummer ungültig oder Karte nicht installiert
255	ffh	Allgemeiner Treiberfehler

Fehlerklasse 11: MODULAR-Device-Driver (MDD-)Fehler

Fehlercode		Bedeutung des Fehlercodes der Fehlerklasse 11
1	1h	Neuere OsX-Version erforderlich
64	40h	Handle ungültig
65	41h	Verwendeter Datentyp falsch
66	42h	Fehler in MDD-Dienst (Diagnose möglich)
67	43h	Fehler in Kanal-Dienst (Diagnose möglich)
68	44h	Dienst nicht verfügbar
69	45h	MDD nicht installiert/bereit

Fehlerklasse 12: Flash-Programmierung

Fehlercode		Bedeutung des Fehlercodes der Fehlerklasse 12
132	84h	Soft-State falsch oder anderer Befehl noch nicht beendet
133	85h	Eingesetztes Flash-EPROM wird nicht unterstützt
134	86h	Dienst falsch
135	87h	Kein Flash-EPROM (laut EEPROM, siehe Anhang L)
136	88h	Steckplatz oder IC-Nr. falsch
137	89h	Parameter nicht aligned
138	8ah	Adresse falsch
139	8bh	Flash-EPROM nicht gelöscht
140	8ch	Flash-EPROM nicht programmierbar/löschbar

Notizen

F. Fehlermeldungen und SRQ's des Betriebssystems

Fehlermeldungen und Service-Requests (SRQs) sind 2-Byte Meldungen. Sie werden mit gesetztem DLP-Bit entweder direkt zum PC geschickt oder in einem internen Fehlerpuffer auf der Karte gehalten. Das Low Byte gibt die Fehlergruppe an, das High Byte (Fehlertyp) gibt eine nähere Erklärung dazu oder z.B. die Tasknummer, den Makrobefehl, die Interrupt-Nummer, etc. Die Größe des Low Byte gibt einen Hinweis auf den Ernst der Lage:

Fehlergruppe (Low Byte)		**Bedeutung**	**Fehlertyp (High Byte)**
Dez	**Hex**		
255	ffh	CPU-Hardware-Defekt	Typ des Fehlers (s.u.)
254	feh	Hardwaredefekt im CPU-Teil der Karte	Typ des Fehlers bzw. Device
253	fdh	Hardwaredefekt auf Modul	Modul-Nr. (obere 4 Bit) und Typ (untere 4 Bit)
252	fch	reserviert	-
251	fbh	Systemabsturz	Hinweis auf Ursache
250	fah	Unerwarteter TRAP, FAULT, INT (s. dfh), nicht korrigierbar	Vektor-Nr.
249	f9h	Fehler aus PREPARE	Meldung aus PREPARE
248	f8h	reserviert	-
...	...	reserviert	-
245	f5h	reserviert	-
244	f4h	Hardware-Fehler von Device	siehe Device-
243	f3h	Systemabsturz	Ursache Task-Nr. 300h - 3ffh
242	f2h	Systemabsturz	Ursache Task-Nr. 200h - 2ffh
241	f1h	Systemabsturz	Ursache Task-Nr. 100h - 1ffh
240	f0h	Systemabsturz	Ursache Task-Nr. 000h - 0ffh
239	efh	Taskabsturz	Task-Nr. 300h - 3ffh
238	eeh	Taskabsturz	Task-Nr. 200h - 2ffh
237	edh	Taskabsturz	Task-Nr. 100h - 1ffh
236	ech	Taskabsturz	Task-Nr. 000h - 0ffh

Fehlergruppe (Low Byte)		**Bedeutung**	**Fehlertyp (High Byte)**
Dez	**Hex**		
235	ebh	Task deaktiviert	Task-Nr. 300h - 3ffh
234	eah	Task deaktiviert	Task-Nr. 200h - 2ffh
233	e9h	Task deaktiviert	Task-Nr. 100h - 1ffh
232	e8h	Task deaktiviert	Task-Nr. 000h - 0ffh
231	e7h	PC: Sende-Timeout Befehl abgebrochen	Makrobefehl
230	e6h	PC: Empfangs-Timeout Befehl abgebrochen	Makrobefehl
229	e5h	PC: Befehl unbekannt Befehl abgebrochen	Makrobefehl
228	e4h	PC: Fehler bei Befehl Befehl abgebrochen	Makrobefehl
227	e3h	PC: Fehler bei Befehl Befehl abgebrochen	Typ des Fehlers (s.u.)
226	e2h	Fehler bei Host-Kommunikation	
225	e1h	Fehler bei Zugriff auf Device	Typ des Fehlers (s.u.)
224	e0h	Fehler bei Subroutine	Typ des Fehlers (s.u.)
223	dfh	Unerwarteter TRAP, FAULT, INT, korrigiert (s. Gruppe 250 (=fah))	Vektor-Nr.
222	deh	Fehler: Parameter nicht erlaubt	Parameter-Nr.
...	...	Warnungen: reserviert	-
220	dch	Warnungen: reserviert	-
219	dbh	Warnung von Task	Task-Nr. 300h - 3ffh
218	dah	Warnung von Task	Task-Nr. 200h - 2ffh
217	d9h	Warnung von Task	Task-Nr. 100h - 1ffh
216	d8h	Warnung von Task	Task-Nr. 000h - 0ffh
215	d7h	Warnung von Betriebssystem	Grund (Typ des Fehlers)
214	d6h	Warnungen: reserviert	-
213	d5h	Warnungen: reserviert	-
212	d4h	Warnung: nicht gesendet	Grund (Bitmap)
211	d3h	Befehl komplett abgewickelt, aber nicht ausgeführt	Makrobefehl
210	d2h	Befehl komplett abgewickelt, aber nicht ausgeführt	Typ des Fehlers

Fehlergruppe (Low Byte)		**Bedeutung**	**Fehlertyp (High Byte)**
Dez	**Hex**		
209	d1h	PC: Warnung von Befehl, ausgeführt	Makrobefehl
208	d0h	PC: Warnung von Befehl, ausgeführt	Typ der Warnung
207	cfh	SRQ: Karte wurde geresettet	-
206	ceh	SRQ: reserviert für DMA	-
205	cdh	SRQ von Message-MDD	PC-Prozeß-Nr.
204	cch	SRQ von Clock-Master	Steckplatz des Clock-Masters
203	cbh	SRQ: Kommunikation wurde vom Initiator abgebrochen	-
202	cah	SRQs: reserviert	-
...	...	SRQs: reserviert	-
198	c6h	SRQs: reserviert	-
197	c5h	SRQ von anderem Host	Host-Identifikation
196	c4h	SRQ von Betriebssystem	Grund (s.u.)
195	c3h	SRQ von Task	Task-Nr. 300h - 3ffh
194	c2h	SRQ von Task	Task-Nr. 200h - 2ffh
193	c1h	SRQ von Task	Task-Nr. 100h - 1ffh
192	c0h	SRQ von Task	Task-Nr. 000h - 0ffh
191	bfh	SRQ-Codes für User	frei verfügbar
...	...	...	...
128	80h	SRQ-Codes für User	frei verfügbar

Fehlertypen und Warnungen

Der Fehlertyp und Erklärungen zu Warnungen wird im **High-Byte** einer Fehlermeldung (SRQ) und als Fehlermeldung bei bestimmten Makrobefehlen geliefert.

Nummer Dez	Hex	Bedeutung
1	01h	Timeout bei Kommunikation
2	02h	Falscher Parameter bei Makrobefehl
3	03h	Reserviert
4	04h	Makrobefehl nicht implementiert
5	05h	ROM-Programm nicht vorhanden
6	06h	Protected Mode nicht erlaubt
7	07h	Programmnummer bei Installierung und PDT verschieden
8	08h	Task bzw. Programm nicht installiert
9	09h	Feature (noch) nicht implementiert
10	0ah	NI-Task-Tabelle voll
11	0bh	Systemfehler (unlogisch)
12	0ch	Task ist nicht aktiv
13	0dh	Falsches Format
14	0eh	Kein Platz im RAM
15	0fh	Subroutine nicht implementiert
16	10h	Falscher Parameter bei Aufruf einer Subroutine
17	11h	Timeout aufgetreten
18	12h	Device läßt sich nicht beschreiben
19	13h	Nicht genug Speicher
20	14h	Falsche Kennung
21	15h	Device nicht vorhanden
22	16h	Anzahl Parameter Hin falsch (bei Funktionsaufruf)
23	17h	Anzahl Parameter Rück falsch (bei Funktionsaufruf)
24	18h	Funktionsnummer ungültig oder Task nicht installiert
25	19h	Debugging mit Turbo-Debugger nicht möglich, keine DDT
26	1ah	Unbekannter Fehler bzw. falsche Fehlergruppe von Funktion gemeldet
27	1bh	Falscher Modul-Steckplatz
28	1ch	EEPROM-Inhalt falsch
29	1dh	Erster zu schreibender Parameter/Data außerhalb Bereich

Nummer		**Bedeutung**
Dez	**Hex**	
30	1eh	Anzahl zu schreibender Parameter/Daten zu groß
31	1fh	Es läuft gerade ein PC-Makrobefehl
32	20h	Device wird vom Betriebssystem benutzt
33	21h	Unerlaubte Zeitangabe
34	22h	Puffer wird gerade beschrieben
35	23h	Puffer wird gerade gelesen
36	24h	Nicht genügend Platz im Puffer
37	25h	Nicht genügend Zeichen im Puffer
38	26h	Puffernummer ungültig
39	27h	Tasktyp ungültig
40	28h	Cache ungültig (Puffer)
41	29h	Unerlaubter Parameter
42	2ah	Modul nicht bereit
43	2bh	Device nicht bereit
44-47	2ch-2fh	Reserviert
48	30h	RAM (0 bis 64 KByte) defekt
49	31h	Interruptnummer falsch
50	32h	Reserviert
51	33h	Tasknummer schon benutzt
52	34h	TI-Task Timeout
53	35h	Linker-Signatur falsch
54	36h	Device noch nicht initialisiert
55	37h	Falscher Tasktyp
56-63	38h-3fh	Reserviert
64	40h	Fehler aus MDD: CDT-Kennung falsch
65-67	41h-43h	Reserviert
68	44h	Eintrag ungültig
69	45h	Reserviert
70	46h	Fehler aus MDD mit falscher Fehlergruppe (≠e0h)
71-95	47h-5fh	Reserviert

Nummer Dez	Hex	Bedeutung
96-127	**60h-7fh**	**Benutzerspezifische Fehler**
128	80h	Kein Modul auf Steckplatz vorhanden
129	81h	Modul benötigt keine Initialisierung
130	82h	Modul wird nicht initialisiert wegen EEPROM-Info in Wort 1
131	83h	Modul wird nicht initialisiert, weil Subroutine fehlt
132	84h	Falscher State (Flash-EPROM)
133	85h	Device-Typ wird nicht unterstützt (Flash-EPROM)
134	86h	Falscher Dienst (Flash-EPROM)
135	87h	Kein Flash lt. EEPROM-WORT-27 (Flash-EPROM)
136	88h	Falsche SP- oder IC-Nr. (Flash-EPROM)
137	89h	Adresse nicht aligned oder nicht auf Sektorgrenze (Flash-EPROM)
138	8ah	Adresse falsch (Flash-EPROM)
139	8bh	Flash bzw. Sektor nicht gelöscht (Flash-EPROM)
140	8ch	Fehler (Timeout) beim Programmieren/Löschen (Flash-EPROM)
141-143	8dh-8fh	Reserviert
144	90h	Host schon/noch installiert
145	91h	Host nicht installiert
146-223	92h-dfh	Reserviert
224-255	**e0h-ffh**	**Benutzerspezifische Warnungen**

G. Makrobefehle

Bei der Standardeinstellung und -betriebsart der Karte ist die PC-Schnittstelle 16 Bit breit. Bei der Anzahl Byte "Hin" und "Rück" wird vom kürzesten Format ausgegangen. Bei anderen Formaten, bestimmt durch das Formatbyte, müssen bei der Anzahl "Hin" bis zu 4 Byte, bei der Anzahl "Rück" bis zu 2 Byte hinzuaddiert werden. Die Angabe (s) in der Rubrik "Rück" bedeutet, daß optional von der Karte ein Service-Request zum PC erfolgen kann.

Format	Byte Hin	Byte Rück	Erklärung
Steuer- und Konfigurationsbefehle			
00h 20h	2	2	Überprüfung, MODULAR-4 bereit?
01h 20h	2	2	Lies Kartentyp / Betriebssystem
12h (00h)	2	0	DLP-Bit zurücksetzen
1bh 02h f s	4	0	Verhalten bei PC-Makrobefehlen setzen
I/O-Befehle (privilegiert)			
23h 03h pL pH b	5	0	Schreibe Byte b an 8-Bit-I/O-Adresse p
23h 04h pL pH wL wH	6	0	Schreibe Wort w an 16-Bit-I/O-Adresse p
27h 22h pL pH	4	2	Lies Byte von 8-Bit-I/O-Adresse p
27h 32h pL pH	4	3	Lies Wort von 16-Bit-I/O-Adresse p
Zugriff auf System-RAM (privilegiert)			
20h 04h aE aF aG aH	6	0	Setze Pointer auf absolute Adresse
21h 0fh nL nH b1..bn	4+n	0	Schreibe n Byte an (Pointer), P = P + n
24h 02h 55h aah	4	0	Starte Programm ab (Pointer)
25h f0h mL mH	4	m+1	Lies m Byte von (Pointer), P = P + m
22h 52h 00h 00h	4	5	Melde Größe freies RAM
22h 9fh 0ah 00h s a tL tH hL hH nE nF nG nH	14	9	RAM reservieren
26h f4h mL mH pE pF pG pH	8	m+1	Lies Datenblock aus Memory
2eh 0fh nL nH pE pF pG pH b1..bn	4+n	0	Schreibe Datenblock in Memory

G

Format	Byte Hin	Byte Rück	Erklärung
Zugriff auf EEPROM			
28h 32h s a	4	3	Lies EEPROM-Wort direkt
29h 04h s a wL wH	6	0	Schreibe EEPROM-Wort direkt
Zugriff auf Devices (Ausstattung) der MODULAR-4 Karte			
38h 22h pL pH	4	2	Melde ob Programm (pL pH) im ROM
39h 01h s	3	0	Cache ein/aus (s=0: aus, s=3: ein)
34h 20h	2	2	Lies Status der Uhr
35h 01h s	3	0	Setze Betriebsart der Uhr
36h 40h	2	4	Lies Uhrzeit
36h 80h	2	8	Lies Datum und Uhrzeit
37h 07h J M T W h m s	9	0	Setze Datum und Uhrzeit: Jahr, Monat, Tag, Wochentag, Stunden, Min., Sek.
30h 00h	2	0	LED 2 (on-board) ein
31h 00h	2	0	LED 2 (on-board) aus
33h 20h	2	2	Lies Status der LED 2 (on-board)
23h 03h 50h 00h 00h	5	0	LED 1 (ext.) ein
23h 03h 51h 00h 00h	5	0	LED 1 (ext.) aus
3bh 22h s f	4	2	Initialisiere Modul / Basiskarte entsprechend EEPROM
2ah 44h SP IC d p	6	4	Flash-Status lesen /schreiben
2bh 4ah SP IC	12	4	Flash löschen
aE aF aG aH			Anfangsadresse
eE eF eG eH			Endadresse
2ch 4dh nL nH SP IC	4+n	4	Flash programmieren
rE rF rG rH			Rel. Adresse 1. Byte zu programmieren
d0...dn-6			Data
2dh d6h mL mH SP IC	10	4+i	Flash direkt lesen
rE rF rG rH			Rel. Adresse 1. Byte zu lesen
Multi-Tasking: Installieren, Aktivieren, etc.			
40h 0fh 12h 00h	22	0	Installiere Programm:
tL tH			Task-Nr.
pL pH			Programm-Nr.
iL 00h			Interrupt-Nr.
fE fF fG fH			Flag

Format	Byte Hin	Byte Rück	Erklärung
dE dF dG dH			Größe Datenbereich
aE aF aG aH			Anfangsadresse
41h 02h tL tH	4	0	Aktivieren der Task t (NI-, DI-, II-)
41h 0fh 10h 00h tL tH	20	0	Aktivieren der TI-Task t
p 00h zE zF zG zH			Priorität, Hold-Off-Zeit
iE iF iG iH			Intervall
nE nF nG nH			Anzahl Aufrufe
42h 02h tL tH	4	0	Deaktivieren der Task t (SLEEP)
43h 22h tL tH	4	2	Melde ob Task t aktiviert (Anzahl)
3ah 31h i	3	3	Melde Task, die Interrupt i nutzt
4eh 64h nL nH pL pH	6	6	Melde Task, unter der Prg. p installiert ist
Multi-Tasking: Prozedur oder Funktion aufrufen			
44h ddh nL nH	10+n	6+v	Funktion f der Task t aufrufen
mL mH tL tH			(Antwort: 44h err wL wH vL vH a1...av)
fL fH b1..bn-4			
45h 04h tL tH	6	0	Prozedur f der Task t aufrufen
fL fH			
Multi-Tasking: Parameter lesen und schreiben			
3ch 22h tL tH	4	2	Semaphore anfordern
3dh 02h tL tH	4	0	Semaphore freigeben
58h 24h tL tH pL pH	6	2	Parameterbyte lesen
59h 34h tL tH pL pH	6	3	Parameterwort lesen
5ah 54h tL tH pL pH	6	5	Parameterdoppelwort lesen
4ch f4 mL mH tL tH	6	1+n	Parameterblock lesen
pL pH			
5ch 05h tL tH pL pH b	7	0	Parameterbyte schreiben
5dh 06h tL tH pL pH	8	0	Parameterwort schreiben
wL wH			
5eh 08h tL tH pL pH	10	0	Parameterdoppelwort schreiben
wE wF wG wH			
4dh 0f tL tH	6+n	0	Parameterblock schreiben
pL pH b1...bn			

Format	Byte Hin	Byte Rück	Erklärung
Multi-Tasking: Daten lesen und schreiben			
3eh 22h tL tH	4	2	Semaphore anfordern
3fh 02h tL tH	4	0	Semaphore freigeben
46h 02h tL tH	4	0	Setze R-Pointer auf Anfang
47h 06h tL tH nE nF nG nH	8	0	Verschiebe R-Pointer um ± n
48h 02h tL tH	4	0	Setze W-Pointer auf Anfang
49h 06h tL tH nE nF nG nH	8	0	Verschiebe W-Pointer um ± n
50h 22h tL tH	4	2	Lies Datenbyte, P = P + 1
51h 32h tL tH	4	3	Lies Datenwort, P = P + 2
52h 52h tL tH	4	5	Lies Datendoppelwort, P = P + 4
4ah f2h mL mH tL tH	6	1+m	Lies Datenblock, P = P + m
53h f6h mL mH tL tH oE oF oG oH	10	1+m	P=0, lies Datenblock, P = m
54h 03h tL tH b	5	0	Schreibe Datenbyte, P = P + 1
55h 04h tL tH wL wH	6	0	Schreibe Datenwort, P = P + 2
56h 06h tL tH wE wF wG wH	8	0	Schreibe Datendoppelwort, P = P + 4
4bh 0fh tL tH nL nH b1...bn	6+n	0	Schreibe Datenblock, P = P + (n - 2)
57h 0fh nL nH tL tH oE oF oG oH b1...bn	10+n	0	P=0, schreibe Datenblock, P = n - 6
Datenpuffer: Einrichten, Löschen, Statusmeldung, ...			
60h afh 0ah 00h s z tL tH hL hH nE nF nG nH	14	10	Lege Puffer an, Strategie, Alignment, Tasknummer, Verwendungszweck, Größe des Puffers
61h 04h pE pF pG pH	6	0	Entferne Puffer p aus dem Speicher
62h 04h pE pF pG pH	6	0	Lösche Inhalt von Puffer p
63h a4h pE pF pG pH	6	10	Melde Status des Puffers p
6ch 60h	2	6	Melde Status, des zuletzt vom PC angesprochenen Puffers

Format	Byte Hin	Byte Rück	Erklärung
Datenpuffer: Daten lesen und schreiben			
64h 34h pE pF pG pH	6	3	Lies Byte aus Puffer p
65h 44h pE pF pG pH	6	4	Lies Wort aus Puffer p
66h 64h pE pF pG pH	6	6	Lies Doppelwort aus Puffer p
67h d6h mL mH s 00h pE pF pG pH	10	4+v	Lies Block (max. m Bytes) aus Puffer p
68h 25h pE pF pG pH b	7	2	Schreibe Byte in Puffer p
69h 26h pE pF pG pH wL wH	8	2	Schreibe Wort in Puffer p
6ah 28h pE pF pG pH wE wF wG wH	10	2	Schreibe Doppelwort in Puffer p
6bh 4dh nL nH s 00h pE pF pG pH b1 ... bm	10+n	4	Schreibe Block (max. m Bytes) in Puffer p

Format	Byte Hin	Byte Rück	Erklärung
System-Call			
2fh 54h tL tH n 00h	6	5	System-Call (Taskinformationen PDT) n=0: Typ PDT, Länge PDT, Anzahl Prozeduren n=4: Programm-Nr., Vers., Rev. n=8: Prozessor-Typ, Sprache n=12: Flag und Interrupt-Nr. n=16: Anfang Datenbereich n=20: Größe Datenbereich n=24: Minimum Daten n=28: Maximum Daten n=32: Anfang Parameter n=36: Anzahl Parameter n=38: Anfang Hypertext
2fh 54h tL tH n 01h	6	5	System-Call (Taskinformationen TDT) n=0: Typ TDT, Länge TDT, Task-Nr. n=4: Flag, Interrupt-Nr., Anzahl n=8: Anzahl Parameter, Prozeduren n=12: Adresse PDT n=16: Adresse Hypertext n=20: Anfang Datenbereich n=24: Ende Datenbereich n=28: Daten R-Pointer n=32: Daten W-Pointer
2fh 54h tL tH 00h 02h	6	5	System-Call (Taskinformationen) Anfangsadresse der TDT der Task t

H. Taskinformationen

Die im folgenden beschriebenen System-Calls können mit folgenden Funktionen aufgerufen werden:

- PC-Bibliotheken (ML7BIB bzw. ML8BIB, Kapitel 6):
  ml7_get_task_info bzw.
  ml8_get_task_info
- Echtzeitbibliotheken (ML7RTBIB bzw. ML8RTBIB, Kapitel 9):
  ml7rt_get_task_info (Kapitel 9)
  ml8rt_get_task_info (Kapitel 9)
- Systemsubroutine 3: GET_TASK_INFO (Kapitel 10)
- Makrobefehl 2fh (Kapitel 12)

Alle System-Calls beziehen sich auf die beim Aufruf angegebene Task bzw. auf das darunter installierte Programm. Der Rückgabewert ist immer 4 Byte lang, die einzelnen Bytes werden mit aE, aF, aG und aH bezeichnet. Wenn mehrere Bytes zusammengefaßt werden, ist aE das niederwertigste und aH das höchstwertigste. Wenn kein Programm unter der Task installiert ist, wird immer 0 zurückgeliefert. Zurückgelieferte Adressen sind immer 32 Bit absolute physikalische Adressen, sofern nichts anderes vermerkt ist. Wenn nur Teile der Antwort angegeben sind, dann sind auch nur diese Teile gültig.

Die Funktions- bzw. Prozeduradressen können nur mit den oben angeführten Funktionen für die Funktionen bzw. Prozeduren 0 bis 31 ermittelt werden. Dabei muß für "x" die Länge des Vorspanns der Programm-Deskriptor-Tabelle (PDT) eingesetzt werden, die ebenfalls per System-Call ermittelt werden kann.

PDT-Informationen

callnr (lo)	callnr (hi)	Anforderung von
0	0	Typ PDT (aE), Länge (x) des Vorspanns der PDT (aF), Anzahl der Prozeduren (aG, aH)
4	0	Programm-Nr. (aE, aF), Version (aG) und Revision (aH) des installierten Programms: aG = Version : 30h (="0") bis 39h (="9") aH = Revision : 41h (="A") bis 5Ah (="Z")
8	0	Prozessor-Typ (aE), Co-Prozessor-Typ (aF), Programmiersprache (aG) und Programmtyp (aH)
12	0	Flag (aE, aF) (s. Anhang I), Interrupt-Nr. (aG), (aH = 0)
16	0	Anfangsadresse Datenbereich (physikalisch) (Entspricht dem Wert, der in der PDT eingetragen wurde. Die tatsächliche Anfangsadresse des Datenbereichs einer Task finden Sie in der TDT einer Task (siehe unten)!)
20	0	Größe Datenbereich (Anzahl Bytes)
24	0	Minimum Größe Datenbereich (Anzahl Bytes)
28	0	Maximum Größe Datenbereich (Anzahl Bytes)
32	0	Anfangsadresse des Parameterbereichs (physikalisch), nur gültig, wenn vom Programm selbst reserviert. Andernfalls muß das aus Anfang TDT + Länge TDT ermittelt werden. Der Parameterbereich beginnt immer direkt "nach" der TDT.
36	0	Größe des Parameterbereichs (aE, aF) (in Anzahl Bytes)
38	0	Anfangsadresse Hypertextbereich (physikalisch)
x+0	0	Adresse der Main-Prozedur (Proz. 0) (Segment:Offset) (x=48 bei PDT Typ 1)
x+4	0	Adresse der Auto-Init-Prozedur (Proz. 1) (Segment:Offset) (x=48 bei PDT Typ 1)
x+8	0	Adresse der 1. Anwenderprozedur (Proz. 2) (Segment:Offset) (x=48 bei PDT Typ 1)

TDT-Informationen

callnr (lo)	callnr (hi)	Anforderung von
0	1	Typ TDT (aE), Länge TDT (aF), Task-Nr. (aG, aH)
4	1	Flag (aE, aF), Interrupt-Nummer (I-Task) bzw. Anzahl der Aktivierungen (NI-Task) (aG) (aH = 0)
8	1	Anzahl Parameter (aE, aF) (in Anzahl Bytes), Anzahl Prozeduren (aG, aH)
12	1	Adresse PDT (physikalisch)
16	1	Adresse Hypertext (physikalisch)
20	1	Anfangsadresse Datenbereich (physikalisch)
24	1	Endadresse Datenbereich (physikalisch)
28	1	Datenbereich R-Pointer (physikalisch)
32	1	Datenbereich W-Pointer (physikalisch)

TDT-Adresse

callnr (lo)	callnr (hi)	Anforderung von
0	2	Anfangsadresse der TDT

Notizen

I. Programm-Deskriptor-Tabelle (PDT)

Die einzelnen Einträge der Tabelle werden im folgenden ausführlich erläutert. Die Spalte 'rel. Adr.' steht für die Position innerhalb der PDT, an der der beschriebene Eintrag beginnt (in Byte). Die Angaben gelten für PDT-Typ = 1.

rel. Adr.	**Typ**	**Erklärung**
0	Byte	Typ der PDT (z.Zt. = 1)
1	Byte	Länge des Vorspanns der PDT (bei Typ 1 immer = 48)
2	Word	Anzahl der Prozeduren (z.B. = 3)
4	Word	Programmnummer
6	Char	Programmversion (z.B. '1')
7	Char	Programmrevision (z.B. 'A')
8	Byte	Prozessor-Typ (4 = 486)
9	Byte	Coprozessor-Typ (4 = 486DX)
10	Byte	Programmiersprache des Programms
11	Byte	Programmtyp
12	Word	Flags (16 Bit)
14	Word	Interrupt-Nummer, für die das Programm gedacht ist
16	Long	Anfangsadresse des Datenbereichs (physikalisch)
20	Long	Größe des Datenbereichs in Byte
24	Long	Minimale Größe des Datenbereichs in Byte
28	Long	Maximale Größe des Datenbereichs in Byte
32	Long	Anfangsadresse des Parameterbereichs (physikalisch)
36	Word	Größe des Parameterbereichs in Byte
38	Long	Anfangsadresse des Hypertextbereichs (physikalisch)
42	Word	Reserviert
44	Long	Reserviert
48	Long	Adresse der Haupt-Prozedur (Prozedur 0)
52	Long	Adresse der Auto-Init-Prozedur (Prozedur 1)
56	Long	Adresse der 1. Anwenderprozedur (Prozedur 2)
...	...	... (gegebenenfalls weitere Anwenderprozeduren)

Typ der PDT (Byte)

Im Moment wird nur der PDT-Typ 1 unterstützt. Setzen Sie also in Ihren Programmen den Typ der PDT immer gleich 1.

Länge des Vorspanns der PDT (Byte)

Sie ist konstant und zur Zeit bei PDT-Typ 1 = 48 Byte. In den Bibliotheken "ML7RTBIB" und "ML8RTBIB" gibt es eine Konstante mit dem Namen "**_PDT_LENGTH**", die Sie in Ihren Programmen benutzen können.

Anzahl der Prozeduren (Word)

Sie beträgt mindestens 2 (Haupt-Prozedur und Auto-Init-Prozedur) und maximal 16320. Die Gesamtlänge der PDT in Byte errechnet sich aus der Länge des Vorspanns + (Anzahl der Prozeduren * 4).

Programmnummer (Word)

Sie kann von 1 bis 65534 gewählt werden. Programmnummer 0 ist reserviert für das Betriebssystem. Die Programmnummer 0ffffh ist ebenfalls reserviert, es handelt sich um ein Dummy-Programm ohne Funktion und wird gemeldet, wenn kein Programm unter einer Task installiert ist.

Programmversion und Programmrevision (2 Byte)

Beide Einträge werden als ASCII-Zeichen angegeben (Version von "1" bis "9", Revision von "A" bis "Z") und dienen nur der Versionsverwaltung des Programmierers. Bei Verwendung der Hypertext-Struktur werden sie auch dort verwendet.

Prozessor-Typ (Byte)

Diese Angabe beschreibt, für welchen Prozessor das Programm geschrieben wurde.

(Value)	Prozessor-Typ	Konstante in ML7RTBIB bzw. ML8RTBIB
0	8086/8088	**_86**
1	80186/V20	**_186**
2	80286	**_286**
3	80486 SX	**_486SX**
4	80486 DX	**_486DX**
5	80586	**_586**

Coprozessor-Typ (Byte)

Diese Angabe beschreibt, ob und welchen Coprozessor das Programm erwartet.

(Value)	Coprozessor-Typ	Konstante in ML7RTBIB bzw. ML8RTBIB
0	Keiner	**_NOCOPROZ**
1	8087	**_87**
2	80287	**_287**
3	80487 SX	**_487SX**
4	80487 DX	**_487DX**
5	80587	**_587**

Falls Sie Fließkommaoperationen verwenden wollen, so benutzen Sie **_487SX** bzw. **_487DX**. Falls Sie keine Fließkommaoperationen verwenden, so benutzen Sie **_NOCOPROZ**.

Falls Sie **_487SX** benutzen, sind die Programme sowohl auf einer MODULAR-4/486SX (Emulation wird benutzt) als auch auf einer MODULAR-4/486DX Karte (Coprozessor wird benutzt) lauffähig. Mit **_487DX** kompilierte Programme setzen voraus, daß eine MODULAR-4/486DX Karte verwendet wird.

Programmiersprache des Programms (Byte)

Durch die Angabe der Programmiersprache, in der das Programm geschrieben ist, wird das Betriebssystem in die Lage versetzt, bestimmte sprachspezifische Eigenheiten zu berücksichtigen.

(Value)	Programmiersprache	Konstante in ML7RTBIB bzw. ML8RTBIB
1	Assembler	**_ASM**
2	Turbo-Pascal 5.0	**_TP50**
3	Turbo-Pascal 5.5	**_TP55**
4	Turbo-Pascal 6.0	**_TP60**
5	Borland C 3.0	**_CPP30**
6	Borland C 3.1	**_CPP31**
7	Turbo-Pascal 7.0	**_TP70**
8	Borland C 4.0	**_CPP40**
9	Borland C 4.5	**_CPP45**
10	Borland C 5.0	**_CPP50**

Programmtyp (Byte)

Dieser Eintrag definiert die Aufgabe des Programms. Folgende Einträge sind möglich:

(Value)	Programmtyp	Konstante in ML7RTBIB bzw. ML8RTBIB
0	Systemprogramm	**_SYSTEM_PRG**
1	Anwenderprogramm	**_USER_PRG**
2	Soft-Interrupt	**_SOFT_INTERRUPT**
3	Interrupt	**_HARD_INTERRUPT**
4	Taskmanager	**_TASK_MANAGER**
5	Trap-/Errorhandler	**_ERROR_TRAPS**

Flags (Word = 16 Bit)

Die Angaben in Klammern sind die Namen der Konstanten, die für den jeweiligen Eintrag in ML7RTBIB bzw. ML8RTBIB definiert sind.

Bit-Nr.	Bedeutung
0..2	Tasktyp: Bits 210 000: Nicht-Interrupt-Task (**_NI_TASK**) 001: Indirekte Interrupt-Task (**_II_TASK**) 010: Direkte Interrupt-Task (**_DI_TASK**) 011: Timer-Initiierte Task (**_TI_TASK**)
3[1]	Tasktyp und Interrupt-Nummer werden durch PDT festgelegt (Bit = 1). (**_PDT_INFO_ENABLE** = 8)
4	Real-Mode- (Bit = 0) oder Protected-Mode Programm (Bit = 1). (**_PROTECTED_MD** = 16)
5	Code cacheable (Bit = 0) oder nicht (Bit = 1). (**_NO_CACHE_USE** = 32)
6	Reserviert.
7[1]	Hypertext vorhanden (Bit = 1). (**_HYPER_TEXT** = 128)
8[1]	Der Datenbereich wird vom Betriebssystem (Bit = 0) oder vom Programm selbst reserviert (Bit = 1). (**_LOCAL_DATA** = 256)
9[1]	Datenbereichsgröße variabel (Bit = 0) oder fest (Bit = 1). (**_FIXED_DATASIZE** = 512)
10[1]	Der Parameterbereich wird vom Betriebssystem (Bit = 0) oder vom Programm reserviert (Bit = 1). (**_LOCAL_PARAMETER** = 1024)
11..15	Reserviert.

[1] Erklärung siehe folgende Seiten

Flag Bit-3: Tasktyp und Interrupt-Nummer liegen fest?

Wenn dieses Bit = 0 gesetzt ist, können Tasktyp und Interrupt-Nummer beim Installieren festgelegt werden. Wenn das Bit = 1 gesetzt wird, werden die Angaben zum Tasktyp und zur Interrupt-Nummer auf jeden Fall aus der PDT entnommen. Natürlich müssen die Angaben von Tasktyp und Interrupt-Nummer in diesem Fall gültige Werte aufweisen.

Flag Bit-7: Angaben zu Hypertext im Programm enthalten?

Wenn dieses Bit = 0 ist, ist keine Hypertext-Information **als Teil des Programms** vorhanden, die Angabe "Adresse Hypertext" in dieser PDT ist ungültig. Es kann aber nach dem Installieren des Programms später trotzdem eine separate Datei mit Hypertext- Information auf die Karte geladen und installiert werden.

Wenn das Bit = 1 ist, dann enthält das Programm bereits Hypertext-Informationen und die Angaben "Adresse Hypertext" in dieser PDT ist gültig.

Flag Bit-8: Wer reserviert Datenbereich und liefert die Adresse?

- Bit-8 = 0
  Standardmäßig wird der Datenbereich vom Betriebssystem reserviert, und damit Anfangs- und Endadresse festgelegt, entsprechend dem zum Zeitpunkt der Installierung des Programms zur Verfügung stehenden Speicherplatz. Die Angabe "Anfangsadresse Datenbereich" in der PDT wird dann nicht verwendet. Ob die Angabe "Größe des Datenbereichs" ausgewertet wird, hängt von Bit-9 und von der Installierung ab (s.u.). Der Zugriff vom Programm aus auf den eigenen Datenbereich muß dann immer indirekt über Pointer oder Betriebssystemaufrufe geschehen. Durch diese Technik ist ein Programm, das nur einmal auf der Karte vorhanden ist, mehrfach installierbar, weil die Task bei der Installierung eines Programms einen eigenen Datenbereich (und Parameterbereich) erhält. Damit das Programm mehrfach installierbar ist, muß bei der Reservierung des Parameterbereichs genauso vorgegangen werden (siehe Flag Bit-10).

- Bit-8 = 1
  In diesem Fall, wird der Datenbereich vom Programm selbst reserviert oder zur Verfügung gestellt. Die Angaben in dieser PDT "Anfangsadresse Datenbereich" und "Größe Datenbereich" sind dann gültig und werden bei der Installierung des Programms verwendet. Das hat den Vorteil, daß dann der Datenbereich vom Programm aus direkt zugreifbar ist (ohne Pointer). Allerdings ist das Programm dann nicht mehrfach installierbar.

Flag Bit-9: Größe des Datenbereichs fest oder variabel?

- Bit-9 = 0
  Die Größe des Datenbereichs ist variabel und kann beim Installieren vorgegeben werden. Auch wenn die Größe als variabel deklariert ist, kann beim Installieren angegeben werden, daß eine der in der PDT vorgegebenen Größen ("Größe Datenbereich", "Minimum" oder "Maximum") verwendet werden soll.

- Bit-9 = 1
  Die Größe des Datenbereichs ist durch den in der PDT vorgegebenen Wert "Größe Datenbereich" fest vorgegeben, sie kann beim Installieren nicht geändert werden. Der bei der Installierung angegebene Wert für die Größe wird verworfen.

Flag Bit-10: Wer reserviert den Parameterbereich und legt die Anfangsadresse fest?

- Bit-10 = 0
  Standardmäßig reserviert das Betriebssystem den Platz für den Parameterbereich und legt die Anfangsadresse fest. Die Angabe in der PDT "Anfangsadresse Parameterbereich" ist dann ungültig, die Angabe "Größe Parameterbereich" in der PDT wird immer als Größe ausgewertet (s.u.). Der Zugriff vom Programm aus auf den eigenen Parameterbereich muß dann immer indirekt über Pointer geschehen. Durch diese Technik ist ein Programm, das nur einmal auf der Karte vorhanden ist, mehrfach installierbar, weil jede Task bei der Installierung eines Programms unter dieser Task einen eigenen Parameterbereich (und Datenbereich) erhält. Genauso muß dann auch bei der Reservierung des Datenbereichs vorgegangen werden (siehe Flag Bit-8).

- Bit-10 = 1
  Der Parameterbereich wird vom Programm selbst reserviert. Die Angabe in dieser PDT "Anfangsadresse Parameterbereich" ist dann gültig und wird bei der Installierung des Programms verwendet. Das hat den Vorteil, daß dann der Parameterbereich vom Programm aus direkt zugreifbar ist (ohne Pointer). Allerdings ist das Programm dann nicht mehrfach installierbar.

Interrupt-Nummer (Word)

Diese Angabe wird bei DI-, II- und TI-Tasks verwendet, allerdings auch bei solchen, die über einen Taskmanager bedient werden.

Anfangsadresse des Datenbereichs (DWord, physikal. Adresse)

Diese Angabe wird nur für die Installierung verwendet und muß nur dann gültig sein, wenn die Lage des Datenbereichs vom Programm vorgegeben wird. Soll der Datenbereich vom Betriebssystem reserviert werden, dann braucht hier nichts eingetragen werden.

Größe des Datenbereichs (DWord)

Auch diese Angabe wird nur für die Installierung verwendet. Wenn die Reservierung von Speicher und die Lage des Datenbereichs vom Programm selbst vorgegeben wird, dann muß hier die Größe stehen (Anzahl Byte). Wenn die Größe des Datenbereichs fest vorgegeben werden soll, steht hier ebenfalls die Größe des Datenbereichs.

Minimale Größe des Datenbereichs (DWord)

Hier sollte die minimale Größe in Byte angegeben werden, mit der das Programm arbeiten kann.

Maximale Größe des Datenbereichs (DWord)

Hier sollte die maximale Größe in Byte angegeben werden, die das Programm verwalten kann.

Anfangsadresse des Parameterbereichs (DWord, phys. Adresse)

Diese Angabe wird nur für die Installierung verwendet und muß nur gültig sein, wenn die Lage des Parameterbereichs vom Programm vorgegeben wird (siehe oben). Unmittelbar vor dem Parameterbereich liegt immer die TDT (Task-Deskriptor-Tabelle), unabhängig davon, wer den Parameterbereich reserviert und dessen Anfangsadresse festgelegt hat. Es muß die physikalische Adresse des Parameterbereichs angegeben werden.

Größe des Parameterbereichs (Word)

Diese Angabe ist immer zum Installieren erforderlich. Die Größe (in Anzahl Byte) wird fest vorgegeben und kann durch das Installieren nicht verändert werden. Die maximale Größe beträgt 65280. Unmittelbar vor dem Parameterbereich liegt immer die TDT. Wenn der Parameterbereich vom Programm selbst reserviert wird, muß das Programm vor dem Parameterbereich auch den Platz für die TDT reservieren.

Anfangsadresse des Hypertextes (DWord, physikal.)

Diese Adresse ist nur gültig, wenn sie vom Programm definiert wird und wenn Bit 7 im Flag der PDT = 1 ist. Es muß die physikalische Adresse des Hypertextanfangs angegeben werden.

Adressen der Prozeduren (DWord, Segment:Offset)

Die Adressen sind Segment:Offset Adressen, wenn das Programm für Real Mode geschrieben wurde. Die ersten beiden Prozeduren müssen immer vorhanden sein, also die Haupt-Prozedur (Prozedur #0) und die Auto-Init Prozedur (Prozedur #1). Die übrigen sind optional. Wenn mehrere Prozeduren vorhanden sind, müssen sie fortlaufend (ohne Lücken) durchnumeriert sein. Entsprechend der angegebenen Anzahl Prozeduren in dieser PDT müssen auch gültige Adressen vorhanden sein. Der Code einer Prozedur besteht mindestens aus einem "RET FAR".

Die Haupt-Prozedur wird bei NI-Tasks vom Scheduler bzw. bei DI- und II-Tasks bei Auftreten des zugehörigen Interrupts aufgerufen. Die Auto-Init-Prozedur wird immer nach Abschluß des Installierungsvorgangs des Programms unter einer Task aufgerufen, wenn im Flag beim Installieren Bit-11 = 1 gesetzt ist. In Zukunft ist geplant, Prozedur 2 und 3 auch für Betriebssystemzwecke zu benutzen.

Notizen

J. Task-Deskriptor-Tabelle (TDT)

rel.	Typ	Erklärung
0	Byte	TDT-Typ (z.Zt. immer = 1)
1	Byte	Länge der TDT in Byte (bei Typ 1 immer = 36)
2	Word	Task-Nummer
4	Word	Flags
6	Byte	Interrupt-Nummer (bei II- und DI-Tasks) bzw. Zahl der Aktivierungen (bei NI-Tasks)
7	Byte	Reserviert (=0)
8	Word	Größe des Parameterbereichs (Anzahl Byte)
10	Word	Anzahl der Prozeduren
12	Long	Adresse der PDT (physikalisch)
16	Long	Adresse der Hypertextinformationen (physik.)
20	Long	Anfangsadresse des Datenbereichs (physik.)
24	Long	Endadresse+1 des Datenbereichs (physik.)
28	Long	Read-Pointer auf den Datenbereich (physikalisch). Derselbe Pointer wird auch vom PC benutzt.
32	Long	Write-Pointer auf den Datenbereich (physikalisch). Derselbe Pointer wird auch vom PC benutzt.

Die Erklärung der einzelnen Einträge finden Sie auf den folgenden Seiten.

Typ der TDT (Byte)

Der Typ der TDT ist eine Konstante, die für zukünftige Entwicklungen vorgesehen ist. Sie ist zur Zeit immer = 1 gesetzt.

Länge der TDT (Byte)

Die Länge der TDT ist bei Typ 1 immer = 36 Byte.

Task-Nummer (Byte)

Die Tasknummer ist die Nummer, unter der das Programm installiert wurde.

Flags (Word)

Bit-Nr.	Bedeutung
0..2	Task-Typ: Bits 210 000: Nicht-Interrupt-Task 001: Indirekte-Interrupt-Task 010: Direkte-Interrupt-Task 011: Timer-Initiierte Task
3, 4	Privilegstufe des Programms 00: höchste Privilegstufe (z.B. Betriebssystem) 11: niedrigste Priorität (Anwendungsprogramme)
5	Programm läuft im Real-Mode (Bit=0) oder Protected-Mode (Bit=1).
6	Task ist aktiviert (Bit=1)
7	Programm installiert (Bit=1)
8	Angaben zu Hypertext vorhanden (Bit=1) oder nicht vorhanden (Bit=0)
9	Zugriff auf Daten erlaubt (Bit=0) oder nicht erlaubt (Bit=1)
10	Zugriff auf Parameter erlaubt (Bit=0) oder nicht erlaubt (Bit=1)
11..15	Reserviert

Interrupt-Nummer / Anzahl Aktivierungen (Word)

Bei Interrupt-Tasks (DI- und II-Tasks) steht hier die Interruptnummer, bei Nicht-Interrupt-Tasks (NI-Tasks) die Anzahl der Aktivierungen.

Anzahl Parameter (Word)

Dieser Eintrag ist eine Kopie aus dem entsprechenden Eintrag in der PDT des Programmes.

Anzahl Prozeduren (Word)

Dieser Eintrag ist eine Kopie aus dem entsprechenden Eintrag in der PDT des Programmes.

Adresse der PDT (DWord, physikalische Adresse)

Die Adresse der PDT ist nur gültig, wenn ein Programm unter der Task installiert wurde (Bit-7 in Flags =1).

Adresse des Hypertextes (DWord, physikalische Adresse)

Die Adresse ist nur gültig, wenn Hypertextinformationen zu dem Programm installiert wurden (Bit-8 in Flags =1).

Anfangsadresse des Datenbereichs (DWord, physikalische Adr.)

Diese Adresse gibt den Anfang des Datenbereichs an. Nach der Installierung ist diese Angabe immer gültig, unabhängig davon, ob Lage und Größe des Datenbereichs vom Betriebssystem oder vom Programm festgelegt wurden.

Endadresse des Datenbereichs (DWord, physikalische Adresse)

Diese Adresse gibt das Ende des Datenbereichs (letzte Adresse + 1) an. Nach der Installierung ist diese Angabe immer gültig, unabhängig davon, ob Lage und Größe des Datenbereichs vom Betriebssystem oder vom Programm festgelegt wurden.

Read-Pointer (DWord, physikalische Adresse)

Die hier eingetragene Adresse ist der Lesezeiger des Betriebssystems für diese Task. Er wird von den System-Subroutinen und Makrobefehlen zum Zugriff auf den Datenbereich benutzt, also z.B. dann, wenn der PC per Makrobefehl Daten aus dem Datenbereich der Task liest.

Write-Pointer (DWord, physikalische Adresse)

Die hier eingetragene Adresse ist der Schreibzeiger des Betriebssystems für diese Task. Er wird von den System-Subroutinen und Makrobefehlen zum Zugriff auf den Datenbereich benutzt, also z.B. dann, wenn der PC per Makrobefehl Daten in den Datenbereich der Task schreibt.

K. Parameter des Betriebssystems

Das Betriebssystem selbst ist ebenfalls eine Task auf der Karte: Task 0. Damit stehen alle Taskbefehle auch für den Zugriff auf die Strukturen des Betriebssystems zur Verfügung. Das Betriebssystem hat dementsprechend auch einen eigenen Parameterbereich, der teilweise auch für den Anwender von Interesse sein kann. Hier nicht erklärte bzw. reservierte Parameter dürfen auf keinen Fall verändert werden.

In der folgenden Tabelle bedeutet in der Spalte "Typ":

B = Ein-Byte Parameter
nB = n-Byte Parameter
W = Wort Parameter (2 Byte)
D = Doppelwort Parameter (4 Byte)
nS = n-Byte String (ASCII-Zeichen)
P = Physikalische Adresse (4 Byte)

In Spalte "Zugr" bedeutet: R = Parameter darf nur gelesen werden
RW = Parameter darf gelesen und geschrieben werden

Parameter	Typ	Zugr	Bedeutung
0 (00h)	B	R	Version dieser Parameterdefinition: z.Zt. = 01h
1 (01h)	W	R	Anzahl gültiger Parameter des Betriebssystems
4 (04h)	B	R	Kartentyp (7 = "kleine", 8 = "große" MODULAR-4/486)
5 (05h)	B	R	Kartenrevision (1=A, 2=B, 3=C, ...)
10 (0ah)	2S	R	Jahrhundert der Herstellung des Betriebssystems, als Ergänzung des Datums in Parameter 22.
12 (0ch)	10S	R	Betriebssystem Name, Version und Revision, z.B.: "ML8-3B.05x" x = "R" für RAM-Version, x = "E" für (EP)ROM-Version x = "B" für RAM-Beta-Test-Version
22 (16h)	8S	R	Datum der Herstellung des Betriebssystems, z.B.: "10/08/98" (tt/mm/jj), (siehe auch Parameter 10)
30 (1eh)	8S	R	Uhrzeit der Herstellung des Betriebssystems, z.B.: "12:42:59" (hh:mm:ss)

Parameter	Typ	Zugr	Bedeutung
38 (26h)	B	R	CPU-Typ: 01h = V20, 04h = 486, 05h = Pentium
39 (27h)	B	R	CPU-Revision (siehe Intel-/NEC-Spezifikation)
40 (28h)	B	R	CPU-Model (z.Zt. nur gültig für Intel-Pentium)
41 (29h)	B	R	CPU-Hersteller: 1=Intel, 2=AMD, 3=UMC, 4=TI, 5=Cyrix, 6=IBM, 7=STM, 8=NexGen, 9=NEC, 10=Transmeta, 255 = nicht ermittelt
42 (2ah)	D	R	Ergebnis des Hardware-Selbst-Tests der CPU (0 = ok, <> 0 = Fehler)
46 (2eh)	B	R	Co-Proz.-Typ: 0 = keiner, 4 = 487, 5 = Pentium
47 (2fh)	B	R	Co-Proz.-Hersteller: 1 = Intel, 255 = nicht ermittelt
48 (30h)	D	R	CPU-Features (z. Zt. nur gültig für Intel-Pentium)
56 (38h)	W	R	Betriebssytem-Features: Bit-0: 0 = Datenzugriffe auf 1 MB beschränkt 1 = Datenzugriffe bis 4 GB erlaubt Bit-1: 0 = Datenbereiche werden von unten nach oben im Speicher reserviert 1 = von oben nach unten Bit-2: 0 = freies RAM wird nicht initialisiert 1 = freies RAM wird mit 0 initialisiert Bit-3: 0 = RAM-Größen Erkennung automatisch 1 = nicht automatisch, fix
64 (40h)	P	R	Physikal. RAM-Anfang (physikal. Adresse)
68 (44h)	P	R	Physikal. RAM-Ende (letzte Stelle + 1)
72 (48h)	P	R	Freies RAM-Anfang (erste freie Stelle)
76 (4ch)	P	R	Freies RAM-Ende (letzte Stelle + 1)
96 (60h)	W	R	Länge des Empfangs-Puffers für PC-Kommunikation, wenn = 0, dann max. 256
98 (62h)	W	R	Länge des Sende-Puffers für PC-Kommunikation, wenn = 0, dann max. 256
100 (64h)	W	R	EEPROM der Basiskarte: Länge (Anzahl Byte)
102 (66h)	W	R	Parameter-Nr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 100 = 0

Parameter	Typ	Zugr	Bedeutung
104 (68h)	W	R	EEPROM von Modul 1: Länge (Anzahl Byte)
106 (6ah)	W	R	Parameter-Nr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 104 = 0
108 (6ch)	W	R	EEPROM von Modul 2: Länge (Anzahl Byte)
110 (6eh)	W	R	Parameter-Nr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 108 = 0
112 (70h)	W	R	EEPROM von Modul 3: Länge (Anzahl Byte)
114 (72h)	W	R	Parameter-Nr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 112 = 0
116 (74h)	W	R	EEPROM von Modul 4: Länge (Anzahl Byte)
118 (76h)	W	R	Parameter-Nr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 116 = 0
120 (78h)	W	R	EEPROM von Modul 5: Länge (Anzahl Byte)
122 (7ah)	W	R	Parameter-Nr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 120 = 0
124 (7ch)	W	R	EEPROM von Modul 6: Länge (Anzahl Byte)
126 (7eh)	W	R	Parameter-Nr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 124 = 0
128 (80h)	W	R	EEPROM von Modul 7: Länge (Anzahl Byte)
130 (82h)	W	R	Parameter-Nr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 128 = 0
132 (84h)	W	R	EEPROM von Modul 8: Länge (Anzahl Byte)
134 (86h)	W	R	Parameter-Nr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 132 = 0
136 (88h)	W	R	EEPROM von Modul 9: Länge (Anzahl Byte)
138 (8ah)	W	R	Parameter-Nr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 136 = 0
140 (8ch)	W	R	EEPROM von Modul 10: Länge (Anzahl Byte)
142 (8eh)	W	R	Parameter-Nr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 140 = 0

Parameter	Typ	Zugr	Bedeutung
148 (94h)	W	R	CPU-Taktfrequenz (in Vielfachen von 1 MHz)
150 (96h)	W	R	Timer-Eingangstakt (in Vielfachen von 10 kHz)
168 (a8h)	W	R	Max. Anzahl Aktivierungen für NI-Tasks
170 (aah)	W	R	Max. Anzahl Tasks (alle Typen)
176 (b0h)	W	R	Max. Anzahl Puffer
200 (c8h)	B	RW	SRQ-Mode (PC-Schnittstelle): z.Zt. = 1
201 (c9h)	B	RW	SRQ-Delay (PC-Schnittstelle): z.Zt. = 64
202 (cah)	W	R	Größe des SRQ-Puffers in Bytes (Standardeinst.=1024)
204 (cch)	W	R	Nummer des SRQ-Puffers
210 (d2h)	W	R	Zähler für Interrupts an IRQ-7 Master
212 (d4h)	W	R	Zähler für Interrupts an IRQ-7 Slave
214 (d6h)	B	RW	Watchdog: 0=Auto-Retrigger off, 1=on
240 (f0h)	B	R	Status der externen LED (1=on, 0=off)
241 (f1h)	B	R	Status der lokalen LED (1=on, 0=off)
308 (134h)	W	R	Max. Anzahl TI-Tasks
310 (136h)	B	R	Typ des Timer-Chips (0 = 8254)
311 (137h)	B	RW[1]	Timerkanal für TI-Tasks (0=Timer-A, 1=B, 2=C) (Standardeinstellung: Timer-C)
313 (139h)	B	RW[1]	Interruptnr. für TI-Task Timer (Standardeinst.: 93h)
316 (13ch)	D	RW[1]	Timer-Tic für TI-Tasks in Mikrosekunden Es sind Werte zwischen 100 und 26000 zulässig. Sonderfall: 0 bedeutet 1 ms (=Standardeinstellung)
320 (142h)	W	R	Längste Verzögerungszeit einer TI-Task seit Reset (Tics)
322 (144h)	W	R	TI-Task, die das Maximum (Parameter 320) ausgelöst hat
324 (146h)	W	RW	Meldegrenze für Zeitverzögerung einer TI-Task (in Timer-Tics, ffffh=keine Meldung)
326 (148h)	W	RW	Error-Requestwort bei Erreichen der Meldegrenze (Parameter 320 > Parameter 324)

[1] Die Parameter für TI-Tasks können nur vor der ersten Installierung einer TI-Task eingestellt werden, spä tere Einstellungen sind wirkungslos.

Parameter	Typ	Zugr	Bedeutung
398 (16eh)	W	R	Aktuell bearbeitete NI-Task
400 (170h)	W	R	Parameter-Nr., wo Fehler bei Sytsem-Subroutinen gemerkt werden
402 (172h)	W	R	Parameter-Nr., wo Fehler bei Makro-Befehlen gemerkt werden

Notizen

L. EEPROM-Inhalte

Alle Versionen der MODULAR-4/486 Basiskarte verfügen über ein EEPROM auf der Karte. Hier sind Konfigurations- und Initialisierungsdaten abgelegt, die auch vom Anwender benutzt werden können. Nach dem Laden und Starten des Betriebssystems *OsX* auf der Karte werden diese EEPROM-Inhalte immer in Parameter des Betriebssystems kopiert. In Parameter 100 (Wort) von Task 0 steht die Nummer des Parameters, ab dem die EEPROM-Inhalte der Basiskarte gelesen werden können.

Im EEPROM können in WORT-3 bis -8 Initialisierungsdaten angegeben werden. Sie legen fest, was nach einem Hardware-Reset der Karte passiert, z. B. welches Betriebssystem gestartet werden soll, und wie einige Hardwareausgänge der Karte gesetzt werden sollen.

Werkseitig ist bereits eine Konfiguration im EEPROM voreingestellt (s.u.). Alle nicht angegebenen Bits und Wörter sind reserviert und müssen = 0 gesetzt werden.

Eingabe der Konfigurationsdaten ins EEPROM

Es sind 32 16-Bit-Wörter im EEPROM für Konfigurations- und Initialisierungsdaten vorgesehen (WORT-0 bis WORT-31). Die Daten im EEPROM können auf zwei Arten gelesen und geändert werden:

1. Mit dem PC-Hilfsprogramm SNW bzw. SNW32 (von SORCUS), das im Lieferumfang jeder Karte enthalten ist.
2. Aus einem PC-Programm heraus durch Aufruf der entsprechenden Bibliotheksroutinen. Die Beschreibung dieser Routinen finden Sie in Kapitel 6.

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

Werkseitig ist bereits eine Konfiguration im EEPROM voreingestellt (hier dargestellt für MODULAR-4/486 DX2 mit 1 MB RAM und eine "kleine" MODULAR-4/586-133 mit 10 MB RAM):

Wort	Binär		Hex	Bedeutung (Kurzinfo)
0	0010 0011	0000 0001	2301h	Karte: "große" MODULAR-4/486[1], Rev. C
1	0000 0000	0000 0000	0000h	Init Basiskarte und Module
2	0000 0000	0000 0000	0000h	reserviert
Initialisierungsdaten				
3	0000 0000	0000 0000	0000h	Hardware, z. B. LEDs
4	0000 0010	0000 0100	0204h	Software: OsX, SRQ
5-6	0000 0000	0000 0000	0000h	reserviert
7	0000 0000	0000 0000	0000h	SRQ nach Init
8-12	0000 0000	0000 0000	0000h	reserviert
Jumper				
13	0000 0000	0000 1000	0008h	J2: PC-IRQ
14	0000 0000	0000 1100	000ch	J3: PC-DMA
15	0000 0000	0000 0000	0000h	Jx: diverse
Bestückung				
16	0000 0000	0000 0000	0000h	reserviert
17	0001 0001	0001 0001	1111h	GAL-Versionen
18	0001 0001	0001 0001	1111h	GAL-Versionen
19	0000 0000	0000 0001	0001h	Serielles EEPROM
20	0111 0011	0100 0001	7341h	CPU-Typ
21	0111 0011	0001 0010	7312h	CPU-Beschaltung
22	0000 0011	0011 0000	0330h	CPU-Quarz = 33,0 MHz
23	0000 0001	0001 0001	0111h	Serielle Schnittstelle
24	0000 0111	0011 0111	0737h	Baudraten-Quarz = 7,3728 MHz
25	0000 0001	0000 0000	0100h	Timer-Quarz = 10,0 MHz
26	0000 0100	0010 0001	0421h	Timer
27	0000 0010	0000 0010	0202h	ROM
28	0000 0001	0001 0000	0110h	RAM Bank 0
29	0000 0001	0001 0000	0110h	RAM Bank 1
30	0000 0001	0001 1001	0119h	Bestückung, div.
31	0000 0000	0000 0000	0000h	reserviert

[1] Der Kartentyp kann aus Parameter 4 des Betriebssystems ermittelt werden

Wort	Binär		Hex	Bedeutung (Kurzinfo)
0	0010 0011	0000 0001	2201h	Karte: "kleine" MODULAR-4/486[1], Rev. B
1	0000 0000	0000 0000	0000h	Init Basiskarte und Module
2	0000 0000	0000 0000	0000h	reserviert
Initialisierungsdaten				
3	0000 0000	0000 0000	0000h	Hardware, z. B. LEDs
4	0000 0010	0000 0100	0204h	Software: OsX, SRQ
5-6	0000 0000	0000 0000	0000h	reserviert
7	0000 0000	0000 0000	0000h	SRQ nach Init
8-9	0000 0000	0000 0000	0000h	reserviert
10	0000 0000	0000 0001	0001h	Gate-Array-Version IC3
11	0000 0000	0000 0001	0001h	Gate-Array-Version IC4
12	0000 0000	0000 0000	0000h	reserviert
Jumper				
13	0000 0000	0000 1000	0008h	J2: PC-IRQ
14	0000 0000	0000 1100	000ch	J3: PC-DMA
15	0000 0000	0000 0000	0000h	Jx: diverse
Bestückung				
16-18	0000 0000	0000 0000	0000h	reserviert
19	0000 0000	0000 0001	0001h	Serielles EEPROM
20	0111 0011	0100 0001	7341h	CPU-Typ
21	0111 0011	0001 0010	7312h	CPU-Beschaltung
22	0000 0011	0011 0000	0330h	CPU-Quarz = 33,0 MHz
23	0000 0001	0001 0001	0111h	Serielle Schnittstelle
24	0000 0111	0011 0111	0737h	Baudraten-Quarz = 7,3728 MHz
25	0000 0001	0000 0000	0100h	Timer-Quarz = 10,0 MHz
26	0000 0100	0010 0001	0421h	Timer
27	0000 0100	0001 0000	0410h	ROM
28	0000 0001	0001 0000	0110h	RAM Bank 0
29	0000 0001	0001 0000	0110h	RAM Bank 1
30	0000 0011	0001 1001	0319h	Bestückung, div.
31	0000 0000	0000 0000	0000h	reserviert

[1] Der Kartentyp kann aus Parameter 4 des Betriebssystems ermittelt werden

WORT-0: Typ und Version der Karte (darf nicht geändert werden)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	1	WORT-0: Kennung
								0	0	0	0	0	0	0	1	Modultyp: 1 = Basiskarte
				0	0	1	1									Revision: 1 = A, 2 = B, 3 = C
			0													Reserviert
0	0	1														Kennung

WORT-1: Init nach Reset (Basiskarte und Module)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	WORT-1: Init nach Reset (werks. Einst.)
																geändert am: von:
															0	Init Basiskarte: 0 = nein, 1 = ja
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		Reserviert

WORT-2: reserviert

WORT-3: Initialisierung Hardware Basiskarte

Neben den Anfangszuständen der Leuchtdioden enthält WORT-3 auch die Einstellungen für die Interrupt-Leitung (PC-IRQ) und DMA-Leitung (PC-DMA) zum PC. Dabei gilt folgende Zuordnung ('x' bedeutet: Zustand spielt keine Rolle):

IRQEN	IRQREN	PC-IRQ
0	x	nicht verbunden
1	0	1
1	1	0

IRQWEN	DMAEN	DMADIR	PC-DMA
x	0	x	nicht verbunden
0	1	0	freigeschaltet MODULAR-4 ⇒ PC
0	1	1	freigeschaltet PC ⇒ MODULAR-4
1	1	x	verbunden, nicht freigeschaltet

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	WORT-3: Init Basiskarte (werks. Einst.)
																geändert am: von:
														0	0	LED2 (on-board): 00 = off, 01 = on
												0	0			LED1 (ext.): 00 = off, 01 = on
										0	0					Timer-Eingangstakt[1]: 00 = 2,5 MHz 01 = 10 MHz 10 = 1 MHz[2]
								0	0							Reserviert
							0									Zustand von IRQEN
						0										Zustand von IRQREN
					0											Zustand von IRQWEN
				0												Zustand von DMAWEN
			0													Zustand von DMADIR
		0														Lüfter ROT[2] an IRQ-M7 (1 = verbunden)
	0															Temperatur-Sensor TEMP[2] an IRQ-M1 (1 = verbunden)
0																Reserviert (= 0)

[1] Umschaltung des Timer-Eingangstaktes bei Rev. B der "großen" MODULAR-4/486 nicht möglich

[2] Diese Einstellung ist nur bei der "kleinen" MODULAR-4/486 möglich

L

WORT-4: Initialisierung Software (nach Reset)

WORT-4 enthält diverse Angaben, wie nach Reset verfahren werden soll. Einige Angaben sind für zukünftige Entwicklungen vorgesehen.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	WORT-4: Init Software (werks. Einst.)
																geändert am: von:
												0	1	0	0	CPU-Verhalten: 0 = Stop 1 = Diagnose-Mode 1 2 = Diagnose-Mode 2 3 = vom PC aus booten 4 = Mini-OS starten 5 = wie 4, dann System-SRQ 6 = OsX im EPROM starten 7 = wie 6, dann System-SRQ (SRQ steht in WORT-7)
								0	0	0	0					Aktion nach Start von OsX: 0 = nichts tun
				0	0	1	0									System-SRQs: 0 = verwerfen 1 = nur intern puffern 2 = direkt zum PC 3 = über Puffer zum PC
0	0	0	0													Reserviert

WORT-5 und -6: reserviert

WORT-7: SRQ-Wort nach Init (siehe auch WORT-4)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	WORT-7: SRQ-Wort (werks. Einst.)
																geändert am: von:
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	SRQ-Wort nach Init

WORT-8 und -9 : reserviert

WORT-10 und -11: Gate-Array-Versionen[1]

Diese 2 Wörter dürfen nicht geändert werden!

Insgesamt ist die "kleine" MODULAR-4/486 mit zwei solcher ICs ausgerüstet. In gewissem Umfang können damit werkseitig spezielle Features auf der Karte eingestellt bzw. verändert werden.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WORT-10: IC3 Version (autom. Einst.)
												0	0	0	1	Revision (0 = unbekannt, 1 = A, 2 = B...)
								0	0	0	0					Version
0	0	0	0	0	0	0	0									Reserviert (=0)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WORT-11: IC4 Version (autom. Einst.)
												0	0	0	1	Revision (0 = unbekannt, 1 = A, 2 = B...)
								0	0	0	0					Version
0	0	0	0	0	0	0	0									Reserviert (=0)

L

[1] Nur bei "kleiner" MODULAR-4/486

WORT-13: Hardware-Einstellungen (Jumper)[1]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	WORT-13: Jumper J2 (IRQ) (werks. Einst.)
																geändert am: von:
															0	Pin 1-3: 1 = verb. (IRQ-4)
														0		Pin 3-5: 1 = verb. (IRQ-3)
													0			Pin 2-4: 1 = verb. (IRQ-5)
												1				Pin 4-6: 1 = verb. (IRQ-7)
											0					Pin 7-9: 1 = verb. (IRQ-9)
										0						Pin 9-11: 1 = verb. (IRQ-11)
									0							Pin 8-10: 1 = verb. (IRQ-10)
								0								Pin 10-12: 1 = verb. (IRQ-12)
0	0	0	0	0	0	0	0									Reserviert (= 0)

[1] Bei der "großer" MODULAR-4/486 **muß** ein Jumper aufgesteckt werden, der hier dokumentiert wird. Bei der "kleinen" MODULAR-4/486 wird hier der IRQ **definiert**, ein Jumperfeld ist **nicht** vorhanden..

WORT-14: Hardware-Einstellungen (Jumper)[1]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	WORT-14: Jumper J3 (DMA) (werks. Einst.)
																geändert am: von:
															0	Pin 1-3: 1 = verb. (DAK-0)
														0		Pin 2-4: 1 = verb. (DRQ-0)
													1			Pin 3-5: 1 = verb. (DAK-1)
												1				Pin 4-6: 1 = verb. (DRQ-1)
											0					Pin 7-9: 1 = verb. (DAK-2)
										0						Pin 8-10: 1 = verb. (DRQ-2)
									0							Pin 11-13: 1 = verb. (DAK-3)
								0								Pin 12-14: 1 = verb. (DRQ-3)
							0									Pin 13-15: 1 = verb. (DAK-5)
						0										Pin 14-16: 1 = verb. (DRQ-5)
					0											Pin 17-19: 1 = verb. (DAK-6)
				0												Pin 18-20: 1 = verb. (DRQ-6)
			0													Pin 19-21: 1 = verb. (DAK-7)
		0														Pin 20-22: 1 = verb. (DRQ-7)
0	0															Reserviert (= 0)

L

[1] Nur bei "großer" MODULAR-4/486

WORT-15: Hardware-Einstellungen (Jumper)[1]

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	WORT-15: Jumper, diverse (werks. Einst.)
																geändert am: von:
															0	J4, Pin 1-2: 1 = verb. (Watchdog enable)
														0		J4, Pin 5-6: 1 = verb. (Watchdog Timeout)
													0			J4, Pin 3-4: 1 = verb. (Powerfail an /NMI)
												0				St1, Pin 1: 0 = 1-3 (an +5V), 1 = ext. angeschlossen[2]
											0					J6, Pin 1, 2 und 3: 0 = 1-2 (DTR/A für Modem) 1 = 2-3 (DTR/A von TRxCA)
										0						J6, Pin 4, 5 und 6: 0 = 5-6 (RTxCA von Quarz) 1 = 4-5 (RTxCA von Ri/A)
								0	0							Reserviert (= 0)
							0									J8, Pin 1, 2 und 3: 0 = 2-3 (Bank 0 an +5V) 1 = 1-2 (Bank 0 an Ubat)
						0										J8, Pin 4, 5 und 6: 0 = 5-6 (Bank 1 an +5V) 1 = 4-5 (Bank 1 an Ubat)
					0											J5, Pin 1: 0 = frei, 1 = an Ubat+
				0												J5, Pin 4: 0 = 4-5, 1 = an Ubat-
			0													J5, Pin 5: 0 = 4-5, 1 = 5-6
		0														J5, Pin 6: 0 = 6-7, 1 = 5-6
	0															J5, Pin 7: 0 = 6-7, 1 = frei
0																Reserviert (= 0)

[1] Nur bei "großer" MODULAR-4/486

[2] Nur bei Rev. B der "großen" MODULAR-4/486, bei Rev. C reserviert (= 0)

WORT-16: reserviert

WORT-17 und -18: GAL-Versionen[1]

Insgesamt ist die MODULAR-4/486 Karte mit 8 solcher ICs ausgerüstet. In gewissem Umfang können damit spezielle Features auf der Karte eingestellt bzw. verändert werden. Deshalb ist es erforderlich, die Versionen hier einzutragen.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	WORT-17: GAL-Versionen (werks. Einst.)
																geändert am: von:
												0	0	0	1	IC4: 0 = unbekannt, 1 = A, 2 = B...
								0	0	0	1					IC14
				0	0	0	1									IC15
0	0	0	1													IC16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	WORT-18: GAL-Versionen (werks. Einst.)
																geändert am: von:
												0	0	0	1	IC30: 0 = unbekannt, 1 = A, 2 = B...
								0	0	0	1					IC31
				0	0	0	1									IC40
0	0	0	1													IC41

L

[1] Nur bei "großer" MODULAR-4/486

WORT-19: Bestückung: serielles EEPROM

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	WORT-19: serielles EEPROM (werks. Einst.)
																geändert am: von:
								0	0	0	0	0	0	0	1	Größe serielles EEPROM (Anzahl Wörter): 0 = kein, 1 = 32, 2 = 64,...
0	0	0	0	0	0	0	0									Reserviert (= 0)

WORT-20 und -21: Hardware: CPU-Label und CPU-Beschaltung

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	1	1	1	0	0	1	1	0	1	0	0	0	0	0	1	WORT-20: CPU (laut Label) (werks. Einst.)
																geändert am: von:
												0	0	0	1	CPU-Hersteller: 0 = unbekannt 1 = Intel 2 = AMD 3 = UMC 4 = TI 5 = Cyrix 6 = IBM 7 = STM 8 = Nexgen
								0	1	0	0					CPU-Typ: 0 = unbek. 1 = reserviert 2 = 486SX 3 = 486DX 4 = 486DX2 5 = 486DX4 6 = 586DX
				0	0	1	1									Maximal erlaubter externer Takt in MHz: 0 = unbekannt 1 = 20 2 = 25 3 = 33 4 = 40 5 = 50 6 = 60 7 = 66
0	1	1	1													Maximal erlaubter interner Takt in MHz: 0 = unbek. 1 = 20 2 = 25 3 = 33 4 = 40 5 = 50 6 = 60 7 = 66 8 = 80 9 = 90 10 = 100 11 = 120 12 = 133 13 = 150

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	1	1	1	0	0	1	1	0	0	0	1	0	0	1	0	WORT-21: CPU-Beschaltung (werks. Einst.)
																geändert am: von:
												0	0	1	0	CPU-Taktvervielfachung: 0 = unbekannt 1 = x1 2 = x2 3 = x2,5 4 = x3 5 = x4
								0	0	0	1					CPU-Spannungsversorgung 0 = unbekannt 1 = 5 V 2 = 3,3 V 3 = 3,45 V
				0	0	1	1									Externer Takt in MHz 0 = unbekannt 1 = 20 2 = 25 3 = 33 4 = 40 5 = 50 6 = 60 7 = 66
0	1	1	1													Interner Takt in MHz 0 = unbek. 1 = 20 2 = 25 3 = 33 4 = 40 5 = 50 6 = 60 7 = 66 8 = 80 9 = 90 10 = 100 11 = 120 12 = 133 13 = 150

WORT-22: Bestückung: CPU-Quarzfrequenz

Hier wird die externe Quarzfrequenz des CPU-Taktes in Form von 4 Ziffern eingetragen. Für 33,00 MHz wird z. B. die Hexadezimalzahl 0330 eingetragen, für 40,00 MHz entsprechend 0400.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	0	WORT-22: CPU-Quarzfrequenz (werks. Einst.)
																geändert am: von:
												0	0	0	0	1. Ziffer nach Komma
								0	0	1	1					3. Ziffer (danach Komma)
				0	0	1	1									2. Ziffer
0	0	0	0													1. Ziffer

L

WORT-23: Angaben zu seriellen Schnittstellen A und B

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	1	WORT-23: Serielle Schnittstellen (werks. Einst.)
																geändert am: von:
												0	0	0	1	Baustein: 0 = unbekannt 1 = SCC 2 = ESCC
								0	0	0	1					Maximal erlaubte Taktfrequenz in MHz: 0 = unbekannt 1 = 8 2 = 10 3 = 16 4 = 20
				0	0	0	1									Takt an PCLK: 0 = unbek. 1[1] = TCLK 2 = TCLK/2 3[1] = TCLK/4 4[1] = TCLK bzw. TCLK/4 5 = Software-mäßige Erzeugung[2] 6 = PCLK = 7,3728 MHz, Q1 bestückt[3]
0	0	0	0													Reserviert (= 0)

[1] Nur bei "großer" MODULAR-4/486

[2] Einstellung nur bei "kleiner" MODULAR-4/486 möglich, Frequenz siehe WORT-24

[3] Einstellung nur bei "kleiner" MODULAR-4/486 möglich, Takt an PCLK = 7,37 MHz

WORT-24: Bestückung: Quarzfrequenz SiOCLK für Baudrate

Hier wird die Quarzfrequenz für die Erzeugung der SCC-Baudraten in Form von 4 Ziffern eingetragen. Für 7,3728 MHz wird z. B. die Hexadezimalzahl 0737 eingetragen, für 4,9152 MHz entsprechend 0491. Bei der "kleinen" MODULAR-4/486 wird die Frequenz per Software eingestellt. Es kann 7,3728 MHz, 4,9152 MHz oder CPU-Quarzfrequenz/2 = 16,5 MHz eingestellt werden.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	1	1	1	0	0	1	1	0	1	1	1	WORT-24: Baudraten-Quarz (werks. Einst.)
																geändert am: von:
												0	1	1	1	2. Ziffer nach Komma
								0	0	1	1					1. Ziffer nach Komma
				0	1	1	1									2. Ziffer (danach Komma)
0	0	0	0													1. Ziffer

WORT-25: Bestückung: Quarzoszillator für Timer

Hier wird die Bestückung des Quarzoszillators (IC 51) angegeben. Die Taktfrequenz wird in Form von 4 Ziffern eingetragen. Für 2,5 MHz wird z. B. die Hexadezimalzahl 0025 eingetragen, für 10 MHz entsprechend 0100. Ab Rev. C der "großen" MODULAR-4/486 Karte ist die Karte standardmäßig mit einem 10 MHz Quarz ausgerüstet, die Eingangsfrequenz der Timer A, B und C kann aber aus Kompatibilitätsgründen mit Rev. B per Software auf 10 MHz oder 2,5 MHz umgeschaltet werden (siehe WORT-3, Bit 4 und 5). Dazu muß außerdem Bit-0 von WORT-1 = 1 gesetzt werden.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	WORT-25: Timer-Quarz (werks. Einst.)
																geändert am: von:
												0	0	0	0	1. Ziffer nach Komma
								0	0	0	0					3. Ziffer (danach Komma)
				0	0	0	1									2. Ziffer
0	0	0	0													1. Ziffer

WORT-26: Bestückung: Timer A, B und C

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	WORT-26: Timer (werks. Einst.)
																geändert am: von:
												0	0	0	1	Baustein: 0 = unbekannt 1 = 71054 2 = 8254
								0	0	1	0					Max. Taktfrequenz in MHz: 0 = unbekannt 1 = 8 2 = 10 3 = 16 4 = 20
				0	1	0	0									Eingangstakt für Timer: 0 = unbek. 1 = TCLK 2 = TCLK/2 3 = TCLK/4 4 = TCLK bzw. TCLK/4
0	0	0	0													Reserviert (= 0)

WORT-27: Bestückung: ROM (= IC 17)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	WORT-27: ROM (werks. Einst.)
																geändert am: von:
								0	0	0	0	0	0	1	0	Größe (in Vielfachen von 32 KByte): 0 = kein 1 = 32 KByte 2 = 64KByte 4 = 128 KByte, 16 = 512 KByte, etc.
				0	0	1	0									Typ ROM: 0 = unbekannt 1 = ROM 2 = EPROM 3 = EEPROM 4 = Flash-EPROM
0	0	0	0													Reserviert (= 0)

L

WORT-28: Bestückung RAM (Bank 0)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	WORT-28: RAM Bank 0 (werks. Einst.)
																geändert am: von:
								0	0	0	1	0	0	0	0	Größe (in Vielfachen von 32 KByte): 0 = kein 1 = 32 K 4 = 128 K 16 = 512 K, 64 = 2 MByte
					0	0	1									Typ: 0 = unbekannt 1 = statisch 2 = pseudostatisch 3 = dynamisch
				0												Refresh erforderlich: 0 = nein, 1 = ja
			0													Auto-Refresh möglich: 1 = ja
		0														Self-Refresh möglich: 1 = ja
	0															Adreß-Refresh möglich: 1 = ja
0																Reserviert (= 0)

WORT-29: Bestückung RAM (Bank 1)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	WORT-29: RAM Bank 1 (werks. Einst.)
																geändert am: von:
								0	0	0	1	0	0	0	0	Größe: (in Vielfachen von 32 KByte bzw. ab 2 MByte in Vielfachen von 2 MByte) 0 = kein 1 = 32 KByte 4 = 128 KByte 16 = 512 KByte, 64 = 2 MByte, 67 = 8 MByte, 79 =32 MByte
					0	0	1									Typ: 0 = unbekannt 1 = statisch 2 = pseudostatisch 3 = dynamisch
				0												Refresh erforderlich: 0 = nein, 1 = ja
			0													Auto-Refresh möglich: 1 = ja
		0														Self-Refresh möglich: 1 = ja
	0															Adreß-Refresh möglich: 1 = ja
0																Reserviert (= 0)

L

WORT-30: Bestückung: div.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	1	WORT-30: Bestückung, div. (werks. Einst.)
																geändert am: von:
														0	1	Uhrentyp: 0 = unbekannt 1 = 72421 2 = 62421 3 = 63421
													0			LED1 on-board: 1 = ja
												1				LED1 an St1, Pin 12: 1 = ja
											1					LED2 on-board: 1 = ja
										0						LED2 an St1, Pin 8: 1 = ja
									0							PC-RBF an NMI: 1 = ja
								0								Timer C an NMI: 1 = ja
							1									±12 V für RS-232 erforderlich: 1 = ja
					0	0										Temperatursensor: 00 = keiner 01 = TMP 03/04 10 = DS1721
0	0	0	0	0												Reserviert (= 0)

WORT-31: reserviert

M. Das Programm SNW (DOS-Version von SNW32)

Das Programm SNW (Schöne Neue Welt) stellt alles zur Verfügung, was Sie zum Arbeiten mit einer oder mehreren MODULAR-4 Karten benötigen. Es werden bis zu acht Karten unterstützt, die parallel (am PC-Bus) oder seriell (z.B. über COM1 bis COM4) angeschlossen sein können. Das Programm ist intuitiv zu bedienen und stellt zu jedem Zeitpunkt eine umfangreiche kontextsensitive Hilfefunktion zur Verfügung (überall Taste F1). Sie können deshalb gleich nach der Installation des Programms mit der Arbeit beginnen, ohne die weiteren Abschnitte dieses Kapitels lesen zu müssen. Die Beschreibung ist dazu gedacht, einen Überblick über SNW zu geben und die Fälle abzudecken, in denen die Intuition des Programmierers nicht mit der Ihren übereinstimmt.

M.1. Installation des Programmes SNW

Das Programm SNW besteht aus folgenden Dateien:

- SNW.EXE
- SNW.OVR
- SNW.LNK
- SNW.HCT
- SNW.MBF

Diese fünf Dateien müssen in ein gemeinsames Verzeichnis kopiert werden. Das Programm kann dann von jedem beliebigen Verzeichnis aus gestartet werden. So ist es auch möglich, daß mehrere Benutzer/-innen in einem Netz mit einem gemeinsamen Programm arbeiten. Am Ende des Programmes werden alle Einstellungen in der Datei SNW.CCF abgelegt. Diese Datei wird immer in das Verzeichnis geschrieben, von dem aus SNW gestartet wurde. Jeder hat somit seine eigene Konfigurationsdatei.

Die Installation ist mit dem Kopieren der Dateien schon abgeschlossen. Das Programm wird mit

SNW

gestartet.

M.2. Allgemeine Hinweise zur Bedienung von SNW

M.2.1. Menüleiste

Die Menüleiste bleibt während des Arbeitens mit SNW immer in der obersten Zeile des Bildschirms sichtbar. Sie enthält alle Programmoptionen. Viele Optionen sind nicht direkt dargestellt, sondern sind in 'Untermenüs' zu finden.

Abb. M-1: Menüleiste mit Untermenü ‘Tools’

Mit der Maus ist die Menüleiste sehr einfach zu bedienen. Sie müssen nur den Punkt anklicken, für den Sie sich interessieren, und die entsprechende Option wird gestartet oder das zugehörige Untermenü geöffnet.

Über die Tastatur ist die Menüleiste auf verschiedene Arten zu erreichen. Mit der Taste F10 wird ein Punkt der Menüleiste hervorgehoben (markiert). Diese Markierung wird mit den Cursortasten verschoben. Mit der RETURN-Taste wird die markierte Option gewählt. In den Untermenüs gilt das genauso, nur daß statt der 'horizontalen' die 'vertikalen' Cursortasten die Markierung steuern.

Ein schnellerer Weg zu den Optionen der Menüleiste führt über die hervorgehobenen Buchstaben der einzelnen Menüpunkte. Durch Drücken der Alt-Taste gemeinsam mit diesem Buchstaben wird die zugehörige Option sofort aktiviert oder ein Unterverzeichnis geöffnet. Das funktioniert in den Unterverzeichnissen fast genauso, nur daß die ALT-Taste nicht mehr gedrückt werden soll.

Beispiel: Mit [ALT-T] und [U] wird 'Uhr stellen' gestartet.

Den direkten Weg zu einigen Optionen in Untermenüs bieten die sogenannten 'Hotkeys'. Ein Hotkey ist eine Tastenkombination, die eine Option eines Untermenüs sofort startet, also ohne Umweg über die Menüleiste. Für welche Optionen Hotkeys definiert sind und wie sie lauten, wird in den Untermenüs angezeigt. Die Option 'Reset' im Untermenü 'Tools' kann zum Beispiel durch Drücken von CTRL-F10 sofort aktiviert werden (siehe Abb. M-1).

Grau angezeigte Optionen können nicht aktiviert werden. In der Regel deshalb, weil der so dargestellte Punkt nur für MODULAR-4 Karten anderen Typs gedacht ist.

M.2.2. Statuszeile

In den Fällen, wo das Programm nicht für Eingaben bereit ist, weil eine Aktion noch nicht beendet ist (z.B. Lesen der Daten der MODULAR-4 Karte) wird in der Statuszeile angezeigt, was das Programm gerade tut.

Außerdem werden Tastenkombinationen angezeigt, die gerade gültig sind. Im Gegensatz zur Menüleiste ändert sich der Inhalt der Statuszeile während des Programmlaufes. So wird zum Beispiel die Taste F2 je nach Stand des Programmes zum Aktualisieren der Basiskarte, zum Sichern einer Datei oder überhaupt nicht verwendet. Entsprechend taucht sie mit verschiedenen Bezeichnungen (oder gar nicht) in der Statuszeile auf.

Abb. M-2: Beispiel einer Statuszeile

Neben den Tastenkombinationen enthält die Statuszeile meistens eine kurze Hilfe zu dem, was das Programm von Ihnen erwartet, bzw. zu dem, was sich hinter einem Menüpunkt verbirgt.

M.2.3. Fenster

Ein Fenster ist ein Bildschirmbereich, in dem Informationen dargestellt und Dateien angezeigt und geändert werden können. Mehrere Fenster können gleichzeitig geöffnet sein. In der Regel hat ein Fenster folgendes Aussehen:

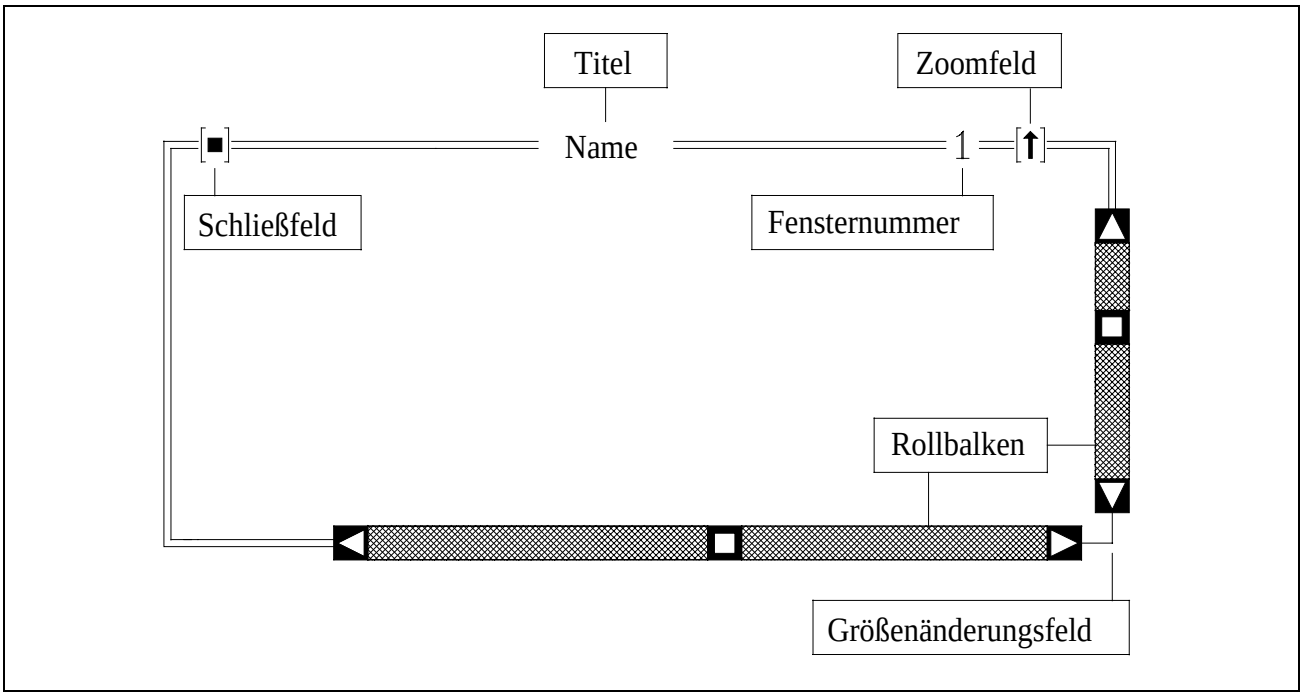


Abb. M-3: Elemente eines Fensters

Der Rahmen eines Fensters besteht aus einfachen oder doppelten Strichen. Doppelte Striche bedeuten, daß das Fenster aktiv ist und sich alle Tastatureingaben auf dieses Fenster beziehen. Der Rahmen enthält einige Elemente, die für die Bedienung des Fensters wichtig sind. Sie sind in Abbildung M-3 angegeben.

- **Fensternummer**
  Jedes Fenster hat eine Nummer, über die es angesprochen werden kann. Sie wird in der Regel in der rechten oberen Ecke des Fensters dargestellt und von SNW in der Reihenfolge des Erscheinens vergeben. Durch gleichzeitiges Drücken der ALT-Taste und der Fensternummer kann ein offenes Fenster angewählt werden. Es wird vor allen anderen dargestellt und mit einem doppelten Rand versehen. Alle Tastatureingaben beziehen sich dann auf dieses Fenster. Mit der Maus wird ein Fenster einfach dadurch angewählt, daß es an beliebiger Stelle angeklickt wird.

- **Titel**
  Jedes Fenster hat einen Titel, der Ihnen Auskunft über den Inhalt des Fensters gibt, zum Beispiel darüber, welche Datei dargestellt wird. Bei Bedienung mit einer Maus dient der Titel auch dazu, das Fenster zu verschieben. Der Mauszeiger wird auf den Titel positioniert und die linke Maustaste gedrückt. Solange die Taste gedrückt bleibt, folgt das Fenster den Bewegungen des Mauszeigers.

- **Schließfeld**
  Das Schließfeld dient dazu, das Fenster zu schließen. Dazu muß es nur mit der Maus angeklickt werden. Für die Bedienung mit der Tastatur hat das Schließfeld keine Bedeutung. Über die Tastatur kann ein Fenster mit ESC oder F4 geschlossen werden. Fenster ohne Schließfeld können weder per Tastatur noch mit der Maus geschlossen werden.

- **Zoom-Feld**
  Durch Anklicken des Zoom-Feldes wird das Fenster über den ganzen Bildschirmbereich ausgedehnt, bzw. wieder in seine ursprüngliche Größe gebracht. Über die Tastatur ist dieselbe Aktion mit F5 verfügbar. Fenster ohne Zoomfeld können nicht vergrößert werden.

- **Rollbalken**
  Der vertikale und der horizontale Rollbalken dienen dazu, das Fenster über einen Text zu bewegen, der zu groß für das Fenster ist. Eine Marke (Quadrat) zwischen den Pfeilen der Rollbalken gibt die relative Position des gezeigten Ausschnittes im Gesamttext an. Die Markierung kann direkt mit der Maus verschoben werden (Maustaste gedrückt halten). Durch Anklicken der Pfeile wird der Fensterinhalt um jeweils eine Zeile bzw. Spalte fortbewegt (Dauerfunktion bei gedrückter Maustaste). Statt mit der Maus können Sie den Fensterinhalt mit den Cursorsteuertasten (zeilen-/spaltenweise) und den Bild↑/Bild↓-Tasten (seitenweise) bewegen.

- **Größenänderungsfeld**
  Wenn das Größenänderungsfeld mit der Maus angeklickt wird, wird die Größe des Fensters mit dem Mauszeiger solange verändert, bis die Maustaste losgelassen wird. Über die Tastatur können Fenstergröße und -position verändert werden, indem F3 gedrückt wird und das Fenster anschließend mit den Cursortasten bewegt wird. Mit SHIFT und den Cursortasten wird die Größe des Fensters eingestellt. Die endgültige Position und Größe des Fensters wird mit RETURN festgelegt.

Der Sonderfall eines Fensters ist das Abbild der MODULAR-4 Karte. Es hat keinen Rahmen und kann weder bewegt noch geschlossen werden. Es hat die Fensternummer Null, so daß es durch Drücken von ALT-0 vor allen anderen Fenstern dargestellt werden kann.

M.2.4. Dialogbox

Eine Dialogbox ist ein Sonderfall eines Fensters. Solange eine Dialogbox geöffnet ist, kann kein anderes Fenster angewählt werden. Auch die Menüleiste kann nicht aktiviert werden.

Abb. M-4: Dialogbox

Dialogboxen kennen fünf verschiedene Objekte, mit denen Eingaben gemacht bzw. Optionen ausgewählt werden können. Innerhalb einer Dialogbox ist immer ein Objekt angewählt. Welches das ist, wird durch Hervorhebung der Bezeichnung dargestellt (in Abb. M-4 ist es 'Mode'). Wenn Sie eine Maus benutzen, ist die Hervorhebung für Sie nicht besonders interessant. Sie können die Objekte in beliebiger Reihenfolge anklicken und bearbeiten. Wenn Sie mit der Tastatur arbeiten, müssen Sie die Markierung mit der Tabulatortaste (vorwärts) und mit SHIFT-Tabulatortaste (rückwärts) bis zu dem Feld verschieben, das Sie bearbeiten wollen. Wenn ein Feld einen hervorgehobenen Buchstaben enthält, können Sie es mit der ALT-Taste und dem entsprechenden Buchstaben auch direkt anwählen.

- **Schalter**
  Jede Dialogbox enthält mindestens einen Schalter. Abb. M-4 enthält einen mit der Bezeichnung 'OK' und einen mit 'Quit'. Schalter dienen dazu, eine Dialogbox zu verlassen. Je nach dem, welchen Schalter Sie benutzen, geschehen beim Verlassen unterschiedliche Dinge. Der Schalter 'OK' sorgt dafür, daß die eingestellten Werte und Optionen der Dialogbox gespeichert und im weiteren Programmverlauf verwendet werden. Mit dem Schalter 'Quit' dagegen schließen Sie die Dialogbox, die

eingestellten Werte werden verworfen. Es gibt noch andere Schalter mit anderen Bedeutungen, die in der kontextsensitiven Hilfestellung ausführlich erläutert sind.
Um einen Schalter zu betätigen, müssen Sie ihn nur mit der Maus anklicken oder die Markierung mit TAB auf einen Schalter bewegen (Schalter blinkt) und RETURN oder Leertaste drücken. Schalter mit hervorgehobenem Buchstaben können auch mit ALT-Buchstabe direkt betätigt werden. Der Schalter 'Quit' kann immer mit der ESC-Taste betätigt werden.

- **Auswahlbox**
  Eine Auswahlbox zeigt eine Anzahl von Optionen, von denen immer nur eine gewählt werden kann (Beispiel: ‘Bit/Zeichen’ in Abb. M-4). Die gewählte Einstellung wird mit einem Punkt angezeigt. Der Punkt wird mit den Cursortasten verschoben oder mit einem Mausklick auf eine beliebige Option gesetzt. Optionen mit hervorgehobenem Buchstaben können, wenn die Auswahlbox selektiert ist, mit diesem Buchstaben direkt eingestellt werden.

- **Einstellungsbox**
  Einstellungsboxen unterscheiden sich von Auswahlboxen nur dadurch, daß mehrere Optionen gleichzeitig eingestellt werden können. Sie werden nicht mit einem Punkt sondern mit einem Kreuz markiert. Das Kreuz wird mit der Leertaste gesetzt bzw. gelöscht. Ansonsten entspricht die Bedienung der der Auswahlbox.

- **Eingabefeld**
  Eingabefelder nehmen Zahlen oder Texte entgegen (Beispiel: 'Baudrate' in Abb. M-4). Oft sind bereits Werte voreingestellt. Der voreingestellte Wert bleibt erhalten, wenn die RETURN-Taste gedrückt wird. Er kann aber auch einfach überschrieben werden. Die Prüfung des Inhaltes eines Eingabefeldes wird in der Regel erst bei Verlassen der Dialogbox mit 'OK' vorgenommen.
  Zum Teil befindet sich am Ende des Eingabefeldes ein Pfeil, der nach unten zeigt. In diesem Fall erhält man durch Anklicken dieses Symbols oder Drücken der Cursortaste-↓ eine Liste mit zuvor eingegebenen oder vom Programm vorgeschlagenen Einträgen.

- **Auswahlliste**
  Eine Auswahlliste dient wie die Auswahlbox zum Wählen aus mehreren Möglichkeiten (Beispiel: ‘phys. Ankopplung’ in Abb. M-4). Sie ist jedoch flexibler, weil mehr Auswahlmöglichkeiten angeboten werden können und sich die angebotenen Möglichkeiten während des Programmlaufes ändern können. Ein typisches Beispiel für eine Auswahlliste ist die Auswahl einer Datei aus den Dateien eines Verzeichnisses. Sie wird ähnlich wie die Auswahlbox bedient, nur daß der ausgewählte Begriff nicht mit einem Punkt gekennzeichnet wird, sondern insgesamt hervorgehoben wird.

M.3. Funktionen von SNW

M.3.1. Anwahl einer Karte

Auf dem Bildschirm wird nach Aufruf von SNW immer eine MODULAR-4 Karte dargestellt. Diese ist die 'aktuelle' Karte. Die meisten Funktionen von SNW beziehen sich auf diese Karte. Unter dem Menüpunkt 'Karte' wird angewählt, um welche der bis zu acht Karten es sich handelt, was für ein Typ es ist und unter welcher Adresse (bzw. Schnittstelle) sie anzusprechen ist.

Die Schnittstellenparameter werden für acht Karten gespeichert. Wenn sie einmal eingestellt sind, kann mit Hilfe der Auswahlbox einfach zwischen verschiedenen Karten gewechselt werden. Mit dem Schalter 'Ändern' können die Parameter der in der Auswahlbox markierten Karte eingestellt werden.

M.3.2. Installieren von Multi-Tasking-Programmen

Die MODULAR-4 Karte verfügt über ein eigenes Echtzeit Multi-Tasking-Betriebssystem (siehe Kapitel 5). Eine zentrale Aufgabe im Umgang mit der MODULAR-4 Karte ist es, Echtzeit-Programme als Tasks zu installieren.

Die eleganteste Methode, Programme zu installieren, Parameter zu setzen und Prozeduren zu starten, ist die Verwendung von Installationsdateien. Installationsdateien bestehen aus einer Folge von Anweisungen, die oben genannten Aktionen auszuführen. Dafür sind einige Schlüsselwörter (siehe Tab. M-1) definiert, die in den Hilfetexten und im Anhang N ausführlich erklärt sind. Installationsdateien erhalten standardmäßig die Extension '.INS'.

MxDEVICE[1]	Anwahl einer MODULAR-4 Karte (optional mit Reset und Laden eines Betriebssystems)
MxINST	Installieren eines Echtzeitprogrammes
MxPAR	Parameter einer Task setzen
MxPROC	Prozedur einer Task starten
MxFUNC	Funktion einer Task starten
MxCMD	Makrobefehl zur MODULAR-4 Karte senden
MxLOADMODUL	Programmierbares SPB-Modul laden

Tab. M-1: Einige Schlüsselwörter in Installationsdateien

[1] Das 'x' gibt den Kartentyp an: x=7 für "kleine" MODULAR-4/486 bzw. x=8 für "große" MODULAR-4/486 (z.B: M8DEVICE)

Das Installieren von Programmen geschieht in zwei Schritten:

- Erstellen einer Installationsdatei
- Laden gemäß der Installationsdatei

Der erste Schritt muß nur einmal ausgeführt werden. Später kann immer wieder auf die gleiche Installationsdatei zurückgegriffen werden.

M.3.2.1. Erstellen von Installationsdateien

Das Programm enthält einen Editor, der das Erstellen und Ändern von Installationsdateien besonders unterstützt. Er wird im Untermenü 'Linken/Laden' unter dem Punkt Datei 'Erstellen/Editieren' aufgerufen. Hilfe wird in zwei Stufen angeboten: eine permanente Kurzhilfe und eine ausführliche Hilfe, die mit F1 oder ALT-F1 angefordert werden.

Die Kurzhilfe erscheint am unteren Rand des Editorfensters. Hier wird angezeigt, ob in der Zeile des Cursors ein gültiges Schlüsselwort gefunden wurde und welches Format die Zeile haben muß. Die dabei verwendeten Kürzel werden im Fenster 'Befehlsformathilfe' kurz erklärt.

Mit ALT-F1 erhalten Sie ausführliche Informationen zum jeweiligen Schlüsselwort. Wenn die Formathilfe den Eintrag 'unbekannt' enthält, weil kein gültiges Wort gefunden wurde, erhalten Sie mit ALT-F1 eine Übersicht der möglichen Schlüsselwörter.

Der Editor kann mit CTRL-Tasten-Kombinationen im Stil von WordStar bedient werden. Besonders interessant sind dabei die Blockbefehle (siehe unten), die es erlauben, Blöcke zu löschen, zu verschieben und zu kopieren (auch von einer Datei in eine andere). Zum Verschieben von Blöcken zwischen unterschiedlichen Dateien (bzw. Editorfenstern) steht eine Zwischenablage zur Verfügung.

- **Cursorsteuerung**
  Der Cursor läßt sich in gewohnter Weise mit den Cursorsteuertasten und den Bild↑/Bild↓-Taste bedienen. Einige Sonderfunktionen sind im folgenden aufgelistet.

CTRL-Linkspfeil:	Wort links	CTRL-Rechtspfeil	Wort rechts
CTRL-Bild↑ (PgUp):	Dateibeginn	CTRL-Bild↓ (PgDn)	Dateiende
Pos1 (Home):	Zeilenbeginn	Ende (End)	Zeilenende
CTRL-T:	Wort löschen	CTRL-Y	Zeile löschen

- **Blockbefehle**
  Die Blockbefehle bestehen aus zwei Tastenkombinationen, die hintereinander eingegeben werden müssen. Blöcke können auf zwei Arten markiert werden. Mit CTRL-K/CTRL-B wird der Beginn eines Blockes gesetzt. Das Ende wird mit CTRL-K/CTRL-K eingestellt. Alternativ dazu können Sie den Block markieren, indem Sie die SHIFT-Taste zusammen mit den Cursorsteuertasten drücken.

  - CTRL-K/CTRL-B Beginn der Blockmarkierung
  - CTRL-K/CTRL-H Markierung ausschalten / einschalten
  - CTRL-K/CTRL-K Ende der Blockmarkierung
  - CTRL-K/CTRL-Y Block löschen
  - CTRL-K/CTRL-C Block an die Stelle des Cursors kopieren
  - CTRL-K/CTRL-V Block an die Stelle des Cursors verschieben

- **Zwischenablage**
  In die Zwischenablage können von jedem Editorfenster aus Blöcke kopiert werden. Ebenso kann ein Block aus der Zwischenablage in jedes Editorfenster eingefügt werden. Dadurch ist es möglich, Blöcke zwischen Dateien auszutauschen. Die Zwischenablage kann jeweils nur einen Block aufnehmen. Sie enthält immer den zuletzt einkopierten Block.

  - CTRL-Einfg (Ins) Block in die Zwischenablage kopieren
  - SHIFT-Einfg (Ins) Block aus der Zwischenablage einfügen

M.3.2.2. Laden gemäß einer Installationsdatei

Beim Laden werden Programmdateien auf die MODULAR-4 Karte geladen und als Tasks installiert. Gemäß der Installationsdatei werden Parameter gesetzt und Prozeduren gestartet.

Die Option 'Laden' muß nicht unbedingt zum Installieren von Programmen verwendet werden. Sie können sie auch nutzen, um (z.B. im Batchbetrieb) einen Reset der Karte durchzuführen, Makrobefehle zu senden oder ein programmierbares SPB-Modul zu laden.

Das Laden ist direkt von DOS aus und damit auch aus einer Batchdatei möglich. Die zugehörigen Kommandozeilenparameter sind '/i:' (für **I**nstallieren) und der Name der Installationsdatei. Wenn der Ladevorgang in 'AUTOEXEC.BAT' eingefügt wird, dann wird die MODULAR-4 Karte nach dem Einschalten automatisch konfiguriert.

Beispiel: **SNW /i:TEST.INS**

Gemäß der Installationsdatei TEST.INS werden Echtzeitprogramme auf der MODULAR-4 Karte installiert. Die Extension '.INS' muß nicht unbedingt angegeben werden.

M.3.3. Bearbeiten der EEPROM-Inhalte von Basiskarte und Modulen

Unter dem Menüpunkt 'Setup' können Sie den Inhalt der EEPROMs der angewählten Karte und ihrer Module ansehen und gegebenenfalls ändern. Weil kein Zugriff auf die EEPROMs erlaubt ist, während ein Echtzeitprogramm auf der MODULAR-4 Karte läuft, wird beim Starten von Setup automatisch ein RESET durchgeführt. Nachdem die EEPROMs beschrieben wurden, wird ein erneuter Reset gegeben, damit die neuen EEPROM Werte vom Betriebssystem verwendet werden.

M.3.4. Kommunikation PC - MODULAR-4

M.3.4.1. Kommunikation über Makrobefehle

Der Menüpunkt Tools/Makrobefehle bietet die Möglichkeit, Makrobefehle zur angewählten MODULAR-4 Karte zu senden und die Antwort zu empfangen. Für die Standardmakrobefehle ist die Anzahl von zu sendenden und zu empfangenden Bytes gespeichert. Diese Parameter können verändert und für neue (z.B. eigene) Makrobefehle eingegeben werden. Die veränderten Makrobefehle werden in einer Datei 'SNW.MBF' im aktuellen Verzeichnis gespeichert.

Die Makrobefehle und ihr Format sind in Kapitel 12 dieses Handbuches ausführlich beschrieben.

M.3.4.2. Kommunikation über PC-I/O-Ports

Mit der Option 'Tools/PC-Debug' können Bytes an die PC I/O-Ports geschrieben und von dort gelesen werden. Auf diese Weise kann die unterste Ebene der Kommunikation zwischen PC und MODULAR-4 Karte nachvollzogen werden.

M.3.5. MODULAR-4 Debug

Unter diesem Punkt können Sie den Speicherinhalt der MODULAR-4 ansehen und editieren. Für spätere Versionen ist ein Disassembler vorgesehen.

M.3.6. Hardware-Reset der MODULAR-4 Karte

Die Option 'Reset' ist im Untermenü 'Tools' zu finden. Nach jedem Reset wird überprüft, ob die Karte zur Kommunikation bereit ist.

Unter dem Punkt Optionen/Reset-Verhalten können Sie einstellen, was außer dem reinen Hardware-Reset noch getan wird (siehe Kapitel M.3.12.4)

M.3.7. Laden einer Datei auf die MODULAR-4 Karte

Diese Option befindet sich im Untermenü 'Tools'. Eine beliebige Datei kann damit von Diskette oder Platte an eine beliebige Stelle ins RAM der MODULAR-4 Karte geladen werden. Wenn es sich bei der Datei um ein lauffähiges Programm handelt, kann es innerhalb dieser Option auch gestartet werden.

M.3.8. Installieren des Turbo-Debuggers

Zum Betrieb des Turbo-Debuggers müssen Echtzeitprogramme auf die MODULAR-4 Karte geladen, parametriert und gestartet werden. Alle nötigen Einstellungen können in der Dialogbox unter ‘Tools/Turbo-Debugger’ vorgenommen werden. Mehr über die Einrichtung des Turbo-Debuggers erfahren Sie in Kapitel 8.

M.3.9. Liste installierter Tasks anzeigen

Unter dem Menüpunkt ‘Tools/Taskliste’ können Sie ein Fenster öffnen, in dem die aktuell auf der Karte installierten Task aufgelistet werden. Wenn mehrere Tasks installiert sind, können Sie eine davon auswählen, um Informationen über diese Task anzuzeigen. Derzeit werden die Programm-Deskriptor-Tabelle (PDT) und die Task-Deskriptor-Tabelle (TDT) angezeigt. Ausführliche Informationen über diese Tabellen finden Sie im Anhang und im Kapitel ‘Echtzeitprogrammierung’.

Das Fenster mit der Taskliste kann geöffnet bleiben. Neu installierte Tasks werden dann automatisch in die Liste aufgenommen.

M.3.10. Testen der Basiskarte und der Module

Unter der Option 'Test' finden sich Testmöglichkeiten für die Basiskarte und ihre Module. Für Module und für Funktionseinheiten der Basiskarte gibt es Testfenster, die die Zustände aller Eingänge (analog, digital, seriell ...) anzeigen und die Mög-

lichkeit bieten, den Zustand aller Ausgänge einzustellen. Der Zustand der Eingänge wird ständig ermittelt und angezeigt, es sei denn, der PC ist gerade mit anderen Dingen beschäftigt (z.B. Datei laden ...). Wenn Werte eingelesen werden, ist das am Blinken der linken oberen Ecke des Fensters erkenntlich.

Testfenster können geöffnet bleiben, während Sie ganz normal mit SNW weiterarbeiten. Insbesondere können auch mehrere Testfenster gleichzeitig geöffnet sein. Allerdings ist zu beachten, daß dann im Hintergrund ständig Kommunikation zwischen PC und MODULAR-4 Karte betrieben wird. Informationen zum aktuellen Testfenster erhalten Sie jeweils mit F1. Hilfe zur Einstellung des angewählten Ausganges erhalten Sie mit ALT-F1.

M.3.10.1. Testen von Einheiten der Basiskarte

Sie können die Leuchtdioden steuern, die seriellen Schnittstellen der Basiskarte testen und den Zustand der Interruptleitungen betrachten.

M.3.10.2. Testen von Modulen

Die Tests für alle gängigen Module sind in SNW integriert. Sollte ein neueres Modul nicht enthalten sein, dann werden die Tests entweder als eigenständige Programme nachgeliefert, oder sind in einem Update von SNW enthalten. Die eigenständigen Programme haben die Bezeichnung M?TMxxx.EXE (xxx=Modulnummer, dezimal) und können von SNW aus aufgerufen werden, wenn sie sich in dem Verzeichnis befinden, von dem aus SNW gestartet wurde.

Sofern ein Modul per Software konfigurierbar ist, wird es entsprechend der Eintragungen im EEPROM eingerichtet. Unabhängig davon, wie es im Augenblick konfiguriert ist und ob es gerade von einem Echtzeit-Programm benutzt wird!

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

M.3.11. Einrichten von Kommunikationsprogrammen (CQ7/CQ8)

In der Lieferung Ihrer MODULAR-4 Karte ist das Programmpaket CQ/ bzw. CQ8 zur gepufferten seriellen Kommunikation enthalten. Die darin gelieferten on-board Echtzeit-Programme übernehmen das zeitkritische Abholen von Zeichen, die an einer seriellen Schnittstelle ankommen und legen diese Zeichen in einem Puffer auf der Karte ab. Umgekehrt können Zeichen aus einem Puffer gesendet werden. Neben der einfachen Kommunikation kann die MODULAR-4 Karte auch komplexe Protokolle abhandeln. Für einige Protokolle können Sie fertige Echtzeit-Programme bei SORCUS Computer erwerben.

Um Einrichtung und Test dieser Kommunikationsprogramme zu erleichtern, kann die Installierung und Parametrierung der benötigten Echtzeitprogramme in SNW menügesteuert durchgeführt werden. Einzelheiten dazu finden Sie in Kapitel 13 "Synchrone und asynchrone serielle Kommunikation".

M.3.12. SNW Programmoptionen

M.3.12.1. Zugriffsmodus

Das Programm SNW greift in der Regel auf die angewählte MODULAR-4 Karte zu, um den Zustand der Karte zu prüfen und die Daten der dargestellten Basiskarte und ihrer Module zu erhalten. Diese Zugriffe können sich in einigen speziellen Anwendungen störend auswirken und können deshalb gesperrt werden. Drei verschiedene Zugriffsmodelle sind in 'Optionen/Zugriffe' einstellbar.

- **Zugriffe gesperrt**
  Das Programm SNW greift dann nicht mehr automatisch auf die MODULAR-4 Karte zu. Die Bildschirmfenster werden nur dann aktualisiert (bzw. initialisiert), wenn 'Aktualisieren' (F2) gewählt wird. Auf der dargestellten MODULAR-4 Karte erscheint die Meldung 'Basiskarte nicht identifiziert', wenn noch kein Zugriff auf die Karte stattgefunden hat, bzw. der Hinweis 'Daten nicht aktualisiert', wenn Anlaß zu der Vermutung besteht, daß sich die Daten im Fenster geändert haben.

- **einmaliger Zugriff (initialisieren)**
  Das Programm SNW greift nur dann auf die MODULAR-4 Karte zu, wenn:
  - eine Karte zum erstenmal (während einer Sitzung) angewählt wird.
  - die Zugriffsparameter einer Karte geändert wurden.
  - 'Aktualisieren' (F2) durchgeführt wird.

  Wenn auf eine Karte bereits zugegriffen wurde, werden die dabei ermittelten Daten dargestellt und nicht mehr aktualisiert. Auch hier wird der Hinweis 'Daten nicht aktualisiert' erzeugt.

- **freigegeben:**
  Immer wenn Anlaß zu der Vermutung besteht, daß sich die auf dem Bildschirm dargestellten Daten geändert haben, werden sie neu bestimmt. Das geschieht insbesondere dann, wenn
  - eine Karte neu angewählt wurde.
  - ein Reset der Karte durchgeführt wurde.

  Wenn die Zugriffe schon beim Starten des Programmes SNW gesperrt sein sollen (unabhängig von der letzten Einstellung, die in SNW.CCF gespeichert ist), dann muß das Programm mit dem Parameter '/Q' (für 'quiet') in der Kommandozeile gestartet werden.

M.3.12.2. Wahl des Monitors

Hier kann die Graphikdarstellung von SNW dem verwendeten Graphikadapter angepaßt werden. Wenn der Monitor unter DOS im richtigen Modus installiert ist, ist eine solche Anpassung in der Regel nicht nötig, da SNW beim ersten Aufruf den unter DOS eingestellten Modus verwendet. Die Option sollte nur in Sonderfällen benutzt werden, da einige Graphikadapter nicht mit allen Einstellungen zusammen arbeiten. Drei Bildschirmmodi stehen unter Optionen/Monitor zur Auswahl:

- **Monochromer Bildschirmmodus**
  Diese Einstellung ist diejenige mit den geringsten Anforderungen an den Monitor. Es werden nur Zeichen mit normaler und intensiver Helligkeit geschrieben. Unterschiedliche Graustufen gibt es nicht. Das äußert sich zum Beispiel darin, daß die Fenster keine Schatten haben. Wenn Sie einen Hercules-kompatiblen Grafiktreiber benutzen, sollten Sie nur mit 'Mono' arbeiten.

  Um den Bildschirmmodus 'Mono' gleich beim Starten des Programms zu wählen (unabhängig von der letzten Einstellung), wird das Programm mit dem Parameter '/M' (also mit 'SNW /M') in der Kommandozeile gestartet.

- **schwarz/weißer Bildschirmmodus**
  Diese Option setzt einen Schwarz/Weiß-Monitor und einen Grafikadapter voraus, der Farbe oder Graustufen darstellen kann. In diesem Modus werden Graustufen dargestellt, die Darstellung wird insgesamt übersichtlicher. Falls die Grautöne zu wenig Kontrast zeigen, können unter dem Menüpunkt 'Optionen/Farben' andere Grauwerte eingestellt werden. Für die meisten Laptops mit LCD-Bildschirmen liefert diese Einstellung die optimalen Ergebnisse.

- **farbiger Bildschirmmodus**
  Diese Einstellung setzt Farbadapter und Farbmonitor voraus. Ein Betrieb mit S/W-Monitor ist mit vielen Graphikadaptern zwar prinzipiell möglich, jedoch ergibt sich dabei teilweise ein sehr geringer Kontrast zwischen Text und Hintergrund bzw. zwischen normalem und hervorgehobenem Text.

Falls die voreingestellten Farben (bzw. Graustufen) der verschiedenen Modi für den verwendeten Monitor zu kontrastarm sind können die Farben (Graustufen) aller Elemente des Bildschirmes in 'Optionen/Farben' geändert werden.

M.3.12.3. Einschalten eines Warntones

Das Programm SNW kann zusätzlich zu jeder Fehlermeldung einen Warnton ausgeben. Dieser kann unter 'Optionen/Warnton' ein- oder ausgeschaltet werden. Nach dem Einschalten wird zur Kontrolle der Warnton einmal ausgegeben.

M.3.12.4. Reset-Verhalten

Hier wird bestimmt, welches Betriebssystem nach einem Reset der MODULAR-4 Karte auf der Karte aktiv werden soll und ob die Uhr automatisch gestellt wird (siehe auch Kapitel 3.1.8. und 3.2.9.).

- **Mini-OS**
  Dabei handelt es sich um ein Minimal-Betriebssystem, das direkt nach einem Reset aktiv ist. Es stellt nur wenige Makrobefehle zur Verfügung. Das führt unter anderem dazu, daß nicht alle Daten der MODULAR-4 Karte ermittelt werden können (z.B. der freie Speicher).

- **ROM-Betriebssystem**
  Mit dieser Option wird nach jedem Reset das im EPROM enthaltene Betriebssystem aktiviert.

- **Betriebssystem laden**
  Diese Einstellung sorgt dafür, daß ein Betriebssystem nach einem Reset von einer Festplatte oder Diskette auf die MODULAR-4 geladen und aktiviert wird. Diese Option ist insbesondere dann zu wählen, wenn Sie ein Betriebssystem-Update auf Diskette bekommen haben. Der Name des Betriebssystems wird in dem Eingabefeld neben der Option 'Betriebssystem laden' eingetragen. Die Angabe des kompletten Pfades ist zulässig.
- **DOS-Zeit nach Reset**
  Wenn diese Option angewählt wird, wird bei jedem Reset der Karte innerhalb von SNW die Uhrzeit des PCs gelesen und auf der MODULAR-4 eingestellt. Außerdem wird als Standardstatus 4ch eingestellt (Uhr läuft, Impulsausgang der Uhr maskiert).

M.3.13. Sonderfunktionen

M.3.13.1. Download-Datei erzeugen

Unter dem Menüpunkt 'Download-Datei erzeugen' sind Sonderfunktionen zusammengefaßt, die bei normaler Benutzung der MODULAR-4 Karte nicht benötigt werden. Es sind Optionen, die hauptsächlich für den SORCUS internen Gebrauch geschaffen wurden. In Fällen, wo sie doch benötigt werden, liegt eine ausführliche Beschreibung vor.

M.3.13.2. Betriebssystem laden

Mit dieser Option können Sie ein Betriebssystem von Festplatte oder Diskette auf die MODULAR-4 Karte laden und dort starten (siehe auch Kapitel M.3.12.4 Reset-Verhalten)

M.3.14. Kommandozeilenparameter

Diese Parameter können beim Aufruf von SNW angegeben werden.

/i:name Echtzeitprogramme gemäß der Datei name.INS laden. (Kapitel M.3.2.2). Dieser Parameter kann zum Abarbeiten mehrerer Installationsdateien auch mehrfach verwendet werden. Nach Ausführung der letzten Installationsdatei wird SNW beendet. Damit ist das Einbinden in eine Batchdatei möglich.

/h unterdrückt alle Bildschirmausgaben beim Laden von Echtzeitprogrammen.

/m SNW mit Monochrom-Monitor starten (Kapitel M.3.12.2)

/q Automatische Kartenzugriffe sperren (Kapitel M.3.12.1)

/r Beim Starten von SNW einen Reset der MODULAR-4 durchführen.

/a:adr Basisadresse vorwählen (adr: hexadezimale Basisadresse).

/t:typ Kartentyp vorwählen (typ=4: Z80, 5: Z280, 6: Multi/COM, 7= "kleine" MODULAR-4/486, 8: "große" MODULAR-4/486).

M.3.15. DOS-Beendigungscodes

Wenn das Programm SNW mit dem Kommandozeilenparameter '/i:' gestartet wurde, wird bei Beendigung des Programmes ein Fehlercode an DOS übergeben. Diesen Code können Sie zum Beispiel innerhalb einer Batch-Datei mit 'errorlevel' auswerten.

Fehlercode	Bedeutung
0	Alle Programme ordnungsgemäß geladen
1	Programmabbruch wegen Speichermangel des PC
2	Programmabbruch aus anderen Gründen

N. Befehle in Installationsdateien

Installationsdateien haben immer die Namenserweiterung '.INS'. Sie sind reine Textdateien, die mit jedem beliebigen Editor erstellt werden können. Jede Zeile beginnt mit einem Schlüsselwort (= Befehl) oder mit einem Kommentarzeichen, Leerzeilen sind zulässig. Alle zu einem Schlüsselwort gehörigen Parameter müssen in einer Zeile stehen.

Untenstehende Schlüsselwörter sind definiert und dürfen in beliebiger Reihenfolge und Häufigkeit verwendet werden. Der erste Befehl einer Installationsdatei muß aber immer ein MxDEVICE[1] Befehl sein.

MxDEVICE	Anwahl einer MODULAR-4 Karte (optional mit Hardware-Reset der Karte und Laden eines Betriebssystems)
MxINST	Installieren eines Echtzeitprogrammes
MxPAR	Parameter einer Task setzen
MxPROC	Prozedur einer Task aufrufen
MxFUNC	Funktion einer Task aufrufen
MxCMD	Makrobefehl zur MODULAR-4 Karte senden
MxLOADMODUL	Programmierbares SPB-Modul laden, z.B. M-AX-32

Für MODULAR-4/Z80 Karten existieren noch weitere Befehle (LINKTIME, IF, USES, M4ROM, M4RESET), die aber für MODULAR-4/486 Karten nicht verwendet werden sollten.

Alle Parameter sind hexadezimal anzugeben. Die maximale Länge ist in der Beschreibung jeweils angegeben. Führende Nullen können entfallen. In der folgenden Syntaxbeschreibung stehen eckige Klammern für optionale Angaben. Die kursiv gedruckten Bezeichner müssen durch Zahlenwerte oder Texte ersetzt werden.

[1] x = 7 für "kleine", x = 8 für "große" MODULAR-4/486 Karten

M7DEVICE bzw. M8DEVICE

Mit diesem Befehl wird eine Basiskarte angewählt, optional zurückgesetzt und optional ein Betriebssystem geladen. Alle darauffolgenden Befehle beziehen sich dann auf diese Karte, bis zum nächsten Befehl MxDEVICE, mit dem eine neue Karte gewählt wird. Wenn Sie die in SNW angewählte Karte eintragen wollen, dann können Sie im SNW-Editor die dafür nötige Befehlsfolge mit ALT-D an der Position des Cursors in die Installationsdatei einkopieren. Beachten Sie, daß im automatischen Eintrag von MxDEVICE immer RESET angegeben ist und gegebenenfalls gelöscht werden muß.

Syntax

MxDEVICE[1] *adr* [TIMEOUT=*time*] [RESET [*osname*]]

adr: vierstellige Basisadresse (I/O-Adresse, hexadezimal), unter der die Karte im PC ansprechbar ist.

TIMEOUT: Stellt den Timeout-Wert für die Kommunikation zwischen MODULAR-4 Karte und PC ein. Wenn 'TIMEOUT' fehlt, wird ein Timeout von einer Sekunde (entspricht TIMEOUT=10) eingesetzt.

time: Größe des Timeout in zehntel Sekunden (*time*=10 entspricht 1 s, erlaubt sind 1 bis 100, dezimal)

RESET: Wenn das Schlüsselwort RESET angegeben wird, dann wird die MODULAR-4 Karte beim Ausführen (Laden) der Installationsdatei mit einem Hardware-Reset zurückgesetzt.

osname: Nach einem RESET der MODULAR-4/486 ist standardmäßig das ROM-Betriebssystem aktiv. Soll statt dessen ein anderes Betriebssystem von Diskette oder Festplatte geladen werden, wird dessen Name nach RESET angegeben (Nur nach RESET möglich!). Pfadangaben sind zulässig.

[1] x = 7 für "kleine", x = 8 für "große" MODULAR-4/486 Karten

M7INST bzw. M8INST

Dieser Befehl dient zum Installieren von Anwenderprogrammen.

Die Programmdatei, des zu installierenden Programms, muß sich (sofern es sich nicht um ein ROM-Programm handelt) in einem der drei folgenden Verzeichnisse befinden:
- in dem Verzeichnis, von dem aus Sie SNW gestartet haben.
- in dem Verzeichnis, in dem die Installationsdatei steht.
- in dem Verzeichnis, in dem das Programm SNW steht.

Syntax:

MxINST[1] *filename number task interrupt datasize flags*

filename: Dateiname (max. 8 Zeichen) und Extension (max. 4 Zeichen incl. Punkt). Ein Suchpfad ist nicht zulässig. Die Datei muß sich in einem der oben genannten Verzeichnisse befinden.

number: Programmnummer (1 - ffffh, hexadezimal). Die Nummer muß mit der in der PDT eingetragenen Nummer des Programmes übereinstimmen (vierstellig, hexadezimal).

task: Tasknummer (1-3ffh, hexadezimal)

interrupt: Nummer des Interrupts, für den die Task installiert werden soll (bei NI-Tasks = 00, zweistellig, hexadezimal)

datasize: Länge des Datenbereichs der Task (sechsstellig, hexadezimal)

flags: Installierungsflags (achtstellig, hexadezimal).

Der Parameter *flags* spezifiziert Installationsoptionen, die der nachfolgenden Tabelle zu entnehmen sind. Beachten Sie bitte, daß für 'EXE'-Dateien (z.B. von Ihnen erstellte Hochsprachen-Programme) in *flags* das Programmformat "Exe not relocated" angegeben werden muß.

Die Parameter Tasktyp (in *flags*), *interrupt* und *datasize* werden in der Regel von den Echtzeit-Programmen mit Werten vorbesetzt (in der Programm-Deskriptor-Tabelle). Ein Echtzeit-Programm kann durch entsprechende Flags in der PDT erzwingen, daß diese Einstellungen verwendet werden, unabhängig von den bei **MxINST** übergebenen Werten. Dabei sind Tasktyp und Interrupt immer miteinander gekoppelt. Sofern es das Echtzeitprogramm zuläßt, kann der Installierungsbefehl

[1] x = 7 für "kleine", x = 8 für "große" MODULAR-4/486 Karten

beide Parameter einstellen (*flags* Bit-3 =1) oder die vom Programm voreingestellten Werte übernehmen (*flags* Bit-3=0). Die Größe des Datenbereichs kann der Installierungsbefehl - wiederum nur wenn es das Echtzeitprogramm zuläßt - aus drei Angaben in der PDT auswählen (minimaler Datenbereich, maximaler Datenbereich und Datenbereichsgröße) oder mit *datasize* frei bestimmen (*flags* Bits 9 und 10).

Im folgenden finden Sie die Bedeutung der einzelnen Bits in *flags*. Den Gesamtwert des Flags ermitteln Sie einfach, indem Sie die entsprechenden Zahlen der Spalte 'Wert' addieren:

Bit	**Wert**	**Bedeutung**
2-0		Tasktyp-Festlegung, wenn Bit 3 des Flags der PDT =0 ist und Bit 3 des Parameters *flags* =1 ist
	0	(=000b): NI-Task (Nicht-Interrupt-Task)
	1	(=001b): II-Task (Indirekte Interrupt-Task)
	2	(=010b): DI-Task (Direkte Interrupt-Task)
	3	(=011b): TI-Task (Timer-Initiierte Task)
3		Wer legt Tasktyp und Interrupt-Nummer fest?
	0	(=0b): PDT legt Tasktyp und Interrupt-Nummer fest
	8	(=1b): wenn Bit 3 des Flags der PDT Null ist, dann wird der Tasktyp und die Interruptnummer durch die Parameter von **MxINST** festgelegt
5-4		Privilegstufe des Programms
	0	(=00b): höchste Privilegstufe (Systemprogramme)
	10h	(=01b): zweithöchste Privilegstufe
	20h	(=10b): dritthöchste Privilegstufe
	30h	(=11b): niedrigste Privilegstufe (Anwendungsprogramme)
8-6		Programmformat

	flagbits	Wo?	Format	Adresse	Typ
0	(=000b):	RAM	PDT (tiny)	Anfang PDT	Assembler
100h	(=100b):	RAM	PDT (large)	Anfang PDT	Assembler
180h	(=110b):	RAM	Exe not reloc.	EXE-Header	C / Pascal
80h	(=010b):	RAM	EXE relocated	START_UP Code	Reserviert
40h	(=001b):	ROM	PDT	wird ignoriert	Reserviert

Bit	**Wert**	**Bedeutung**		
10-9		Wie wird Größe von Datenbereich festgelegt? Die Einstellung ist nur wirksam, wenn Bit 9 im PDT-Flag = 0 ist		
		flagbits	Größe richtet sich nach	fix/variabel
	600h	(=11b):	"Größe" in PDT	fix
	400h	(=10b):	"Minimum" in PDT	fix
	200h	(=01b):	"Maximum" in PDT	fix
	0	(=00b):	*datasize*	variabel
11		Auto-Init Prozedur nach dem Installieren aufrufen?		
	0	(=0b): Auto-Init Prozedur nicht aufrufen		
	800h	(=1b): Auto-Init Prozedur aufrufen		
12		Task nach dem Installieren sofort aktivieren?		
	0	(=0b): Task nicht aktivieren		
	1000h	(=1b): Task aktivieren		
13-31		reserviert		

Installieren bedeutet nicht unbedingt, daß der Programmcode auf die Karte geladen wird. Wenn Sie in SNW unter 'Linken-Laden/Optionen' die Option 'Mehrfach-Installierung' eingeschaltet haben, prüft SNW zunächst, ob bereits eine Task mit der gleichen Programmnummer installiert ist. Wenn ja, wird der Code nicht auf die Karte geladen, sondern der Code der bereits installierten Task verwendet. Das Programm muß natürlich so geschrieben sein, daß es eine Mehrfachnutzung erlaubt. **!**

M7PAR bzw. M8PAR

Mit diesem Befehl können ein oder mehrere Parameter einer Task gesetzt werden. Die Daten werden byteweise übergeben.

Syntax:

MxPAR[1] *task start b1* [*b2*] [*b3*] ... [*bm*]

task: Nummer der Task, deren Parameter gesetzt werden sollen (vierstellig, hexadezimal)

start: Nummer des ersten Parameters, der gesetzt werden soll (vierstellig, hexadezimal)

b1: Wert, auf den der in *start* angegebene Parameter gesetzt werden soll (zweistellig, hexadezimal)

b2: Wert, auf den der nächste (*start*+1) Parameter gesetzt werden soll (zweistellig, hexadezimal)

b3: ...

bm: Wert, auf den der Parameter mit der Nummer *start*+m-1 gesetzt werden soll (zweistellig, hexadezimal)

M7PROC bzw. M8PROC

Dieser Befehl ruft die Prozedur einer Task auf.

Syntax:

MxPROC[1] *task procnr*

task: Nummer der Task, deren Prozedur aufgerufen werden soll (vierstellig, hexadezimal)

procnr: Nummer der Prozedur, die aufgerufen werden soll (vierstellig, hexadezimal)

[1] x = 7 für "kleine", x = 8 für "große" MODULAR-4/486 Karten

M7FUNC bzw. M8FUNC

Dieser Befehl ruft die Funktion einer Task auf. Falls die Funktion eine Antwort zurückgibt, erscheint diese im Informationsfenster (in SNW die Echostufe auf 'jede Ausführung bestätigen' stellen).

Syntax:

MxFUNC[1] *task funcnr* [*p1*] [*p2*] ... [*pn*]

task: Nummer der Task, deren Funktion aufgerufen werden soll (vierstellig, hexadezimal)

funcnr: Nummer der Funktion, die aufgerufen werden soll (vierstellig, hexadezimal)

p1..pn: Parameterbyte, die der Funktion übergeben werden (zweistellig, hexadezimal).

M7CMD bzw. M8CMD

Mit diesem Befehl wird beim Laden ein Makrobefehl zur angewählten Karte gesendet. Falls die MODULAR-4 Karte eine Antwort auf den Makrobefehl zurückgibt, erscheint diese im Informationsfenster (in SNW die Echostufe auf 'jede Ausführung bestätigen' stellen).

Syntax:

MxCMD[1] *code format p1* [*p2*] .. [*pn*]

code: Makrobefehlscode (zweistellig, hexadezimal)

format: Formatbyte (siehe Kapitel 12, zweistellig, hexadezimal)

p1..pn: Parameter des Makrobefehls (zweistellig, hexadezimal, Angabe byteweise).

[1] x = 7 für "kleine", x = 8 für "große" MODULAR-4/486 Karten

M7LOADMODUL bzw. M8LOADMODUL

Dieser Befehl dient zum Laden eines programmierbaren SPB-Moduls. (z.B. M-AX-16, M-AX-32)

Die Ladedatei muß sich in einem der drei folgenden Verzeichnisse befinden:
- in dem Verzeichnis, von dem aus Sie SNW gestartet haben.
- in dem Verzeichnis, in dem die Installationsdatei steht.
- in dem Verzeichnis, in dem das Programm SNW steht.

Syntax:

MxLOADMODUL[1] *slot filename*

slot: Nummer des Modulsteckplatzes (1-10, dezimal)

filename: Dateiname (max. 8 Zeichen) und Extension (max. 4 Zeichen incl. Punkt). Ein Suchpfad ist nicht zulässig.

Kommentartext

Nach einem Strichpunkt (;) oder einem Hochkomma (') kann beliebiger Text stehen. Er wird komplett (bis zum Ende der Zeile) ignoriert. Eventuell enthaltene Befehls-Schlüsselwörter werden natürlich auch ignoriert.

[1] x = 7 für "kleine", x = 8 für "große" MODULAR-4/486 Karten

O. 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, multitaskingfähig 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 Multitaskingfähigkeit 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.

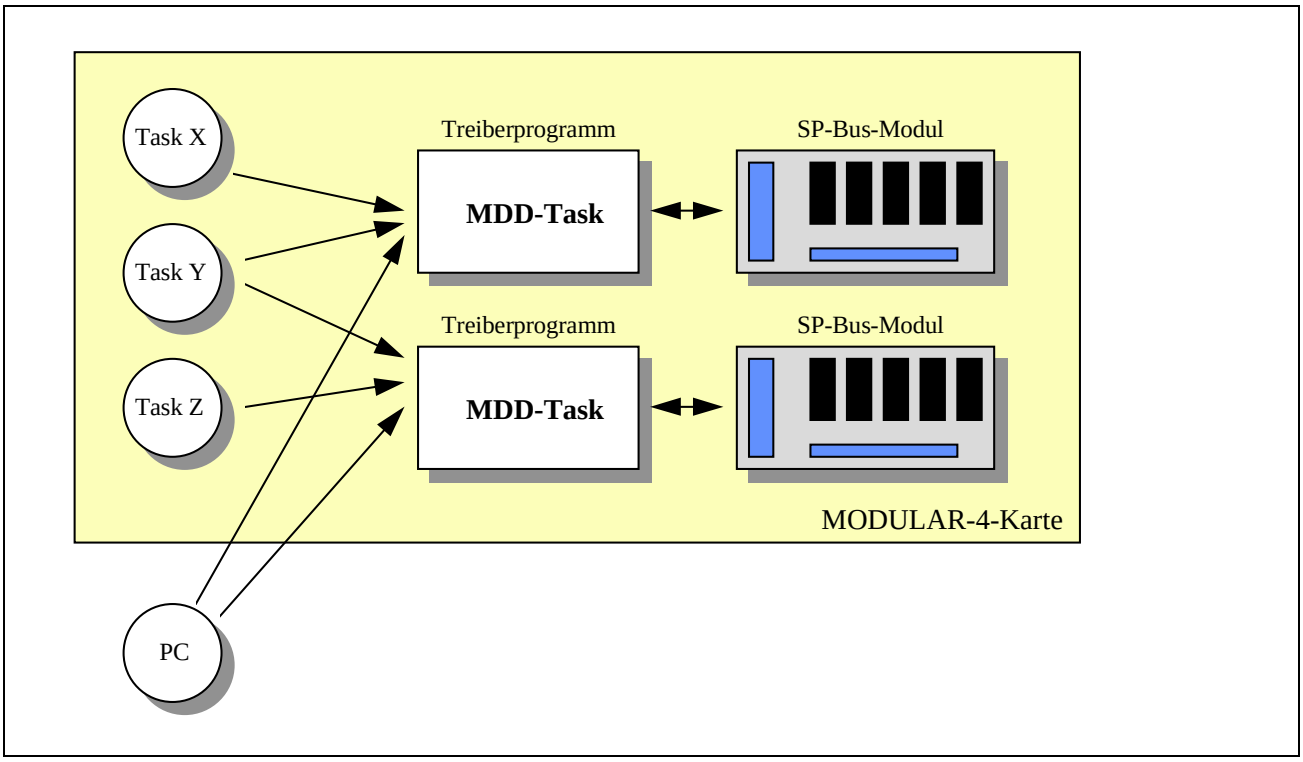


Abbildung O-1: Für jedes Modul wird ein MDD installiert. Der Zugriff auf die Devices eines Moduls kann von verschiedenen Tasks und vom PC erfolgen.

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 **mdd8_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 **mdd8_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] Funktion für ″kleine″ MODULAR-4/486 Karte: mdd7_open_channel bzw. mdd7_close_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: Zugriff vom PC aus) .

O.1. Treiberspezifische Begriffe

O.1.1. MDDI

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

O.1.2. 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.

O.1.3. 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 ≠ 0.

O.1.4. Device

Unter den 'Devices' eines Moduls versteht man die Funktionseinheiten eines SP-Bus-Moduls oder der Basiskarte. 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
BUS_IN	Bus-Empfänger
BUS_OUT	Bus-Sender
SSI	SSI-Empfänger

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.

O.1.5. 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.

O.1.6. 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 **mdd8_get_channel_info**[1] lesend zugegriffen werden. Bei der Beschreibung dieser Funktion werden die Bit des CSR erklärt.

[1] Funktion für ″kleine″ MODULAR-4/486 Karte: mdd7_get_channel_info

O.1.7. Kanalname

Dieser ist eine frei wählbare Bezeichnung eines Kanals. Der Kanalname hat keinen Einfluß auf die Funktionalität des Kanals und ist lediglich für Diagnosezwecke gedacht. Es besteht z.B. die Möglichkeit, daß der PC die Kanalnamen aller zur MODULAR-4 Karte geöffneten Kanäle liest und auf dem Bildschirm darstellt. Der Kanalname eines Kanals besteht aus maximal 80 Zeichen.

Das Setzen und Lesen des Kanalnamens erfolgt über Sonderdienste des jeweiligen Kanals. Das Setzen erfolgt mit der Funktion **mdd8_set_channel_caption**[1], das Lesen mit **mdd8_get_channel_caption**[1].

O.1.8. Infotext

Unter 'Infotext' versteht man eine Zeichenkette (String), die 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 **mdd8_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] Funktion für ″kleine″ MODULAR-4/486 Karte z.B. mdd7_set_channel_caption, mdd7_get_channel_caption

O.2. 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 **mdd8_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] Funktion für ″kleine″ MODULAR-4/486 Karte: mdd7_get_device_status

O.3. 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).

O.4. 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
DEVICE_BUS_IN	Bus-Empfänger
DEVICE_BUS_OUT	Bus-Sender
DEVICE_SSI	SSI-Empfänger

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	Unipolar 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).

O.5. 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 den Kanalnamen eines Kanals ist nach der Installation eines MDD auf 80 voreingestellt.
- MDDs sind nicht mehrfach installierbar. Für jedes Modul muß ein MDD installiert werden.
- Alle MDD-Funktionen und Prozeduren der Bibliotheken beginnen mit **mdd7_...** (für "kleine" MODULAR-4/486 Karte) bzw. **mdd8_...** (für "große" MODULAR-4/486 Karte). Die Funktionen sind gleichwertig, aus Gründen der Übersichtlichkeit werden aber im folgenden nur die Funktionen der Bibliothek für die "große" MODULAR-4/486 Karte beschrieben.

O.6. Modul-Device-Treiber Bibliothek

O.6.1. Dienste eines MDD

mdd8_open_channel — Öffnen eines MDD-Kanals

Pascal
FUNCTION mdd8_open_channel (usMDD, usStrucSize: USHORT; VAR pCPS): HMDD8;

C
HMDD8 mdd8_open_channel (USHORT usMDD, USHORT usStrucSize, void* pCPS);

Funktion
mdd8_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 HMDD8 gespeichert werden.

Ein Handle ist immer eindeutig: Auch wenn **mdd8_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.

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 */
HMDD8 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 = mdd8_open_channel(SLOT_MD402, sizeof(rcMD402), &rcMD402);
```

mdd8_read_infotext MDD-Informationen

Pascal PROCEDURE mdd8_read_infotext (usMDD, usTxtNr: USHORT; ucMaxLen: UCHAR; VAR pacInfotext: STRING);

C void mdd8_read_infotext (USHORT usMDD, USHORT usTxtNr, UCHAR ucMaxLen, CHAR* pacInfotext);

Funktion **mdd8_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.

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

mdd8_get_device_status Status eines MDD-Device

Pascal FUNCTION mdd8_get_device_status (usMDD, usDevice, usIndex, usExtIndex: USHORT): USHORT;

C USHORT mdd8_get_device_status (USHORT usMDD, USHORT usDevice, USHORT usIndex, USHORT usExtIndex);

Funktion **mdd8_get_device_status** ermittelt den Status eines Device.

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

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

mdd8_scan_channel Diagnose eines MDD-Kanals

Pascal FUNCTION mdd8_scan_channel (usMDD, usInfo: USHORT; hChannel: HMDD8; usInfoSize: USHORT; VAR pInfoData): USHORT;

C USHORT mdd8_scan_channel (USHORT usMDD, USHORT usInfo, HMDD8 hChannel, USHORT usInfoSize, void* pInfoData);

Funktion **mdd8_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.

wInfo	**Information (Rückgabewert der Funktion)**
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 **mdd8_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.

wInfo	**Information (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	HMDD8	Handle des Kanals mit Ordnungszahl k+1
4	HMDD8	Handle des Kanals mit Ordnungszahl k+2
:		
4×(n-1)	HMDD8	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.

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)

mdd8_get_version MDD-Version

Pascal PROCEDURE mdd8_get_version (usMDD: USHORT; VAR pulVersion: ULONG; VAR pulDate: ULONG);

C void mdd8_get_version (USHORT usMDD, ULONG* pulVersion, ULONG* pulDate);

Funktion **mdd8_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 **ml8(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 **ml8(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)

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

mdd8_reset_driver — Reset eines MDD

Pascal PROCEDURE mdd8_reset_driver (usMDD: USHORT);

C void mdd8_reset_driver (USHORT usMDD);

Funktion **mdd8_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!

Parameter *usMDD* Tasknummer des MDD

Rückgabe Keine

O.6.2. Dienste eines Kanals

O.6.2.1. Datenausgabedienst (Daten schreiben)

mdd8_write_channel_byte — Schreiben in einen Kanal
mdd8_write_channel_word
mdd8_write_channel_dword
mdd8_write_channel_short
mdd8_write_channel_block

Pascal PROCEDURE mdd8_write_channel_byte (hChannel: HMDD8; data: UCHAR);

PROCEDURE mdd8_write_channel_word (hChannel: HMDD8; data: USHORT);

PROCEDURE mdd8_write_channel_dword (hChannel: HMDD8; data: ULONG);

PROCEDURE mdd8_write_channel_short (hChannel: HMDD8; data: SHORT);

FUNCTION mdd8_write_channel_block (hChannel: HMDD8; usSize: USHORT; VAR pData): USHORT;

C

void mdd8_write_channel_byte (HMDD8 hChannel, UCHAR data);

void mdd8_write_channel_word (HMDD8 hChannel, USHORT data);

void mdd8_write_channel_dword (HMDD8 hChannel, ULONG data);

void mdd8_write_channel_short (HMDD8 hChannel, SHORT data);

USHORT mdd8_write_channel_block (HMDD8 hChannel, USHORT usSize, void* pData);

Funktion

Diese Funktionen schreiben ein, zwei, vier oder *usSize* Bytes in den angegebenen Kanal.

Der Zugriff ist nur dann möglich, wenn der Kanal den jeweiligen Datentyp unterstützt (siehe Dokumentation zum MDD oder **mdd8_get_channel_info**).

Die Größe des Datenblocks, die der Kanal übernehmen kann, wird durch die vom Kanal verwalteten Kanalparameter MINSIZE und MAXSIZE (siehe **mdd8_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.

Parameter

hChannel Handle des Kanals

usSize Größe des zu schreibenden Datenblockes (in Byte)

data Zu schreibender Wert

pData Zeigt auf den zu schreibenden Datenblock

Rückgabe

Keine bzw. beim Schreiben eines Blocks dieAnzahl der vom Kanal nicht verwerteten Daten (in Byte).

mdd8_trigger_channel MDD-Kanal triggern

Pascal PROCEDURE mdd8_trigger_channel (hChannel: HMDD8);

C void mdd8_trigger_channel (HMDD8 hChannel);

Funktion **mdd8_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 **mdd8_get_channel_info**).

Parameter *hChannel* Handle des Kanals

Rückgabe Keine

O.6.2.2. Dateneingabedienst (Daten lesen)

mdd8_read_channel_byte Lesen aus einem Kanal
mdd8_read_channel_word
mdd8_read_channel_dword
mdd8_read_channel_short
mdd8_read_channel_block

Pascal FUNCTION mdd8_read_channel_byte (hChannel: HMDD8): UCHAR;

FUNCTION mdd8_read_channel_word (hChannel: HMDD8): USHORT;

FUNCTION mdd8_read_channel_dword (hChannel: HMDD8): ULONG;

FUNCTION mdd8_read_channel_short (hChannel: HMDD8): SHORT;

FUNCTION mdd8_read_channel_block (hChannel: HMDD8; usSize: USHORT; VAR pData): USHORT;

C UCHAR mdd8_read_channel_byte (HMDD8 hChannel);

USHORT mdd8_read_channel_word (HMDD8 hChannel);

ULONG mdd8_read_channel_dword (HMDD8 hChannel);

SHORT mdd8_read_channel_short (HMDD8 hChannel);

USHORT mdd8_read_channel_block (HMDD8 hChannel, USHORT usSize, void* pData);

Funktion

Diese Funktionen lesen ein, zwei, vier oder *usSize* Bytes aus dem angegebenen Kanal.

Der Zugriff ist nur dann möglich, wenn der Kanal den jeweiligen Datentyp unterstützt (siehe Dokumentation zum MDD oder **mdd8_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 **mdd8_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.

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

Gelesener Wert, bzw. Anzahl der vom Kanal übergebenen Daten (in Byte) beim Lesen eines Blocks.

O.6.3. Sonderdienste

mdd8_send_channel_command Steuerkommando

Pascal PROCEDURE mdd8_send_channel_command (hChannel: HMDD8; usCmd: USHORT);

C void mdd8_send_channel_command (HMDD8 hChannel, USHORT usCmd);

Funktion **mdd8_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

Parameter *hChannel* Handle des Kanals

usCmd Steuerkommando

mdd8_send_channel_control Steuerparameter

Pascal PROCEDURE mdd8_send_channel_control (hChannel: HMDD8; usCtrl: USHORT; ulCtrlPar: ULONG);

C void mdd8_send_channel_control (HMDD8 hChannel, USHORT usCtrl, ULONG ulCtrlPar);

Funktion **mdd8_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 **mdd8_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 **mdd8_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 **mdd8_get_channel_info** ermittelt werden.

Parameter *hChannel* Handle des Kanals

usCtrl Steuerkommando

ulCtrlPar Steuerparameter

mdd8_send_channel_control_block Steuerparameterblock

Pascal FUNCTION mdd8_send_channel_control_block (hChannel: HMDD8; usCtrl, usSize: USHORT; VAR pCtrlData): USHORT;

C USHORT mdd8_send_channel_control_block (HMDD8 hChannel, USHORT usCtrl, USHORT usSize, void* pCtrlData);

Funktion **mdd8_send_channel_control_block** sendet ein Steuerkommando und einen Steuerparameterblock über den angegebenen Kanal.

Diese Funktion wird zur Zeit nicht benötigt.

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)

mdd8_get_channel_info Kanalparameter lesen

Pascal FUNCTION mdd8_get_channel_info (hChannel: HMDD8; usInfo: USHORT): ULONG;

C ULONG mdd8_get_channel_info (HMDD8 hChannel, USHORT usInfo);

Funktion **mdd8_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

usInfo	**Ermittelter Kanalparameter (Rückgabe der Funktion)**		
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_TRANS-FER_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

usInfo	Ermittelter Kanalparameter (Rückgabe der Funktion)		
	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

Parameter *hChannel* Handle des Kanals

usInfo Gewünschter Kanalparameter (siehe Tabelle oben)

Rückgabe Ermittelter Kanalparameter (siehe Tabelle oben)

mdd8_get_channel_info_block Kanalparameterblock lesen

Pascal FUNCTION mdd8_get_channel_info_block (hChannel: HMDD8; usInfo, usSize: USHORT; VAR pInfoData): USHORT;

C USHORT mdd8_send_channel_control_block (HMDD8 hChannel, USHORT usInfo, USHORT usSize, void* pInfoData);

Funktion **mdd8_get_channel_info_block** ermittelt einen Kanalparameterblock des angegebenen Kanals.

Diese Funktion wird zur Zeit nicht benötigt.

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)

mdd8_set_channel_caption Kanalname definieren

Pascal PROCEDURE mdd8_set_channel_caption (hChannel: HMDD8; usLen: USHORT; pcCaption: STRING);

C void mdd8_set_channel_caption (HMDD8 hChannel, USHORT usLen, CHAR* pcCaption);

Funktion **mdd8_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 = 80).

Parameter *hChannel* Handle des Kanals

usLen Länge des zu setzenden Strings

pcCaption Zeigt auf den zu setzenden String

mdd8_get_channel_caption Kanalname ermitteln

Pascal PROCEDURE mdd8_get_channel_caption (hChannel: HMDD8; usLen: USHORT; VAR pcCaption: STRING);

C void mdd8_get_channel_caption (HMDD8 hChannel, USHORT usLen, CHAR* pcCaption);

Funktion **mdd8_get_channel_caption** liest die Kurzbeschreibung des angegebenen Kanals.

Parameter *hChannel* Handle des Kanals

usLen Maximale Länge des zu lesenden Strings

pcCaption Zeigt auf Variable zum Empfang des Strings

mdd8_close_channel MDD-Kanal schließen

Pascal PROCEDURE mdd8_close_channel (VAR phChannel: HMDD8);

C void mdd8_close_channel (HMDD8* phChannel);

Funktion **mdd8_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!

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

mdd8_check_handle_valid MDD-Handle prüfen

Pascal FUNCTION mdd8_check_handle_valid (hChannel: HMDD8): WORD;

C USHORT mdd8_check_handle_valid (HMDD8 hChannel);

Funktion **mdd8_check_handle_valid** prüft die Gültigkeit des übergebenen Handles.

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

O.6.4. 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 **ml8_error_info_type** (PC) bzw. **ml8rt_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

O.6.5. Diagnosemeldungen

Falls der Fehler *ERR_MDD_DRV_SERVICE* oder *ERR_MDD_CHANNEL_SERVICE* auftritt, so kann durch Aufruf der Funktion **mdd8_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 **mdd8_get_diagnosis** alle Fehlerinformationen mit **ml8_clear_error** gelöscht werden, da sonst kein Kartenzugriff zum Abholen der Diagnosemeldung möglich ist!

O.7. Modul-Device-Treiber für MODULAR-4/486

Ein Treiberprogramm verwaltet die Funktionseinheiten eines Moduls und stellt zum Ansprechen dieser Einheiten eine entsprechende Programmierschnittstelle zur Verfügung. 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.

Nachfolgend ist der Modul-Device-Treiber für die Basiskarte beschrieben.

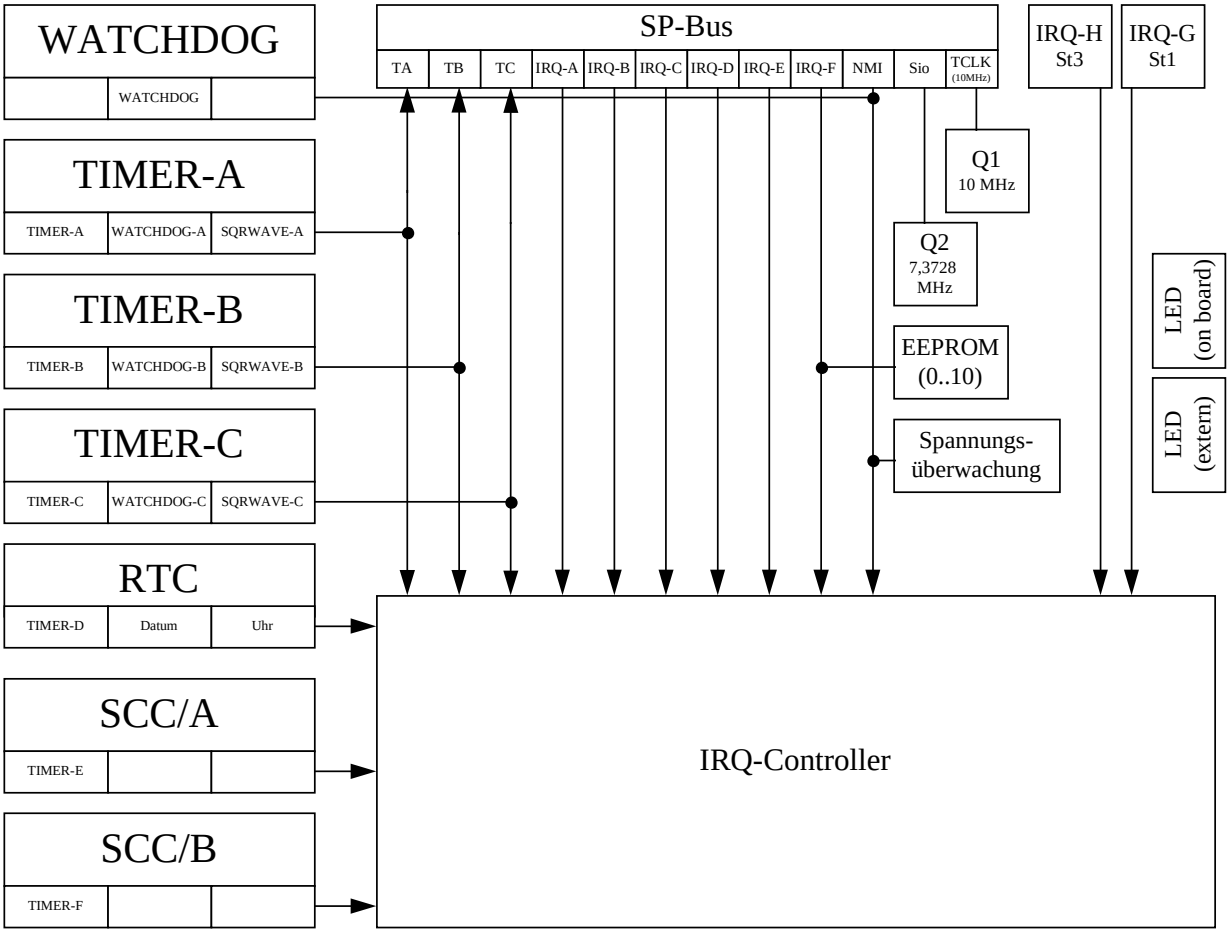
O.7.1. Installationsparameter

Parameter	Wert
Dateiname:	ML8D0000.LIB
Programmnummer:	4FFh
Tasknummer:	11
Interruptnummer:	90h
Länge des Datenbereichs:	0
Flags:	802h

O.7.2. Kanaleigenschaftsstruktur CPS_ML8

Offset	Strukturelement	Datentyp	Bedeutung
0	*.usDevice*	USHORT	Device-Typ
2	*.usIndexFirst*	USHORT	Index des ersten Device
4	*.usIndexLast*	USHORT	Index des letzten Device
6	*.usFlags*	USHORT	Flags für zusätzliche Kanaleigenschaften
8	*.usReadMode*	USHORT	Lesemodus
10	*.usWriteMode*	USHORT	Schreibmodus
12	*.ulTimeout*	DWORD	Timeoutzeit für Watchdog
16	*.usMode*	USHORT	Betriebsmodus

O.7.3. Device Ressourcen im Überblick



Die Devices sind zum Teil nicht unabhängig und verwenden gemeinsame Ressourcen. Wenn z.B. ein Kanal zu Timer-E geöffnet wurde (zur Generierung von Interrupts), steht die Kommunikationsfunktion des SCC/A nicht mehr zur Verfügung, da Timer-E den Baudratengenerator zur Intervallgenerierung benötigt.

O.7.4. Leuchtdioden (LED)

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_LED* = 0402h
2	*.usIndexFirst*	Device-Index: *ML8_EXTERNAL_LED* = 0, *ML8_LOCAL_LED*= 1
4	*.usIndexLast*	= *.usIndexFirst*
6	*.usFlags*	Mögliche Flags: Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv) Alle anderen Bits = 0.

Alle anderen Strukturelemente werden nicht ausgewertet.

Setzen einer Leuchtdiode (0 = aus, 1 = ein):

mdd8_write_channel_byte(handle, bLED)

Aktuelle Zustand einer Leuchtdiode lesen:

bLED = mdd8_read_channel_byte(handle)

O.7.5. Uhr

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_CLOCK* = 0701h
2	*.usIndexFirst*	= 0
4	*.usIndexLast*	= 0
6	*.usFlags*	Mögliche Flags: Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv) Alle anderen Bits = 0.
16	*.usMode*	Reserviert, = 0 setzen

Alle anderen Strukturelemente werden nicht ausgewertet.

Uhrzeit lesen:

ulTime = mdd8_read_channel_dword(handle);

Uhrzeit setzen:

mdd8_write_channel_dword(handle, ulTime);

Format von ulTime (32 Bit):

31 24	23 16	15 8	7 0
Stunden	Minuten	Sekunden	Sekunden/100

O.7.6. Kalender (Datum)

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_CALENDAR* = 0702h
2	*.usIndexFirst*	= 0
4	*.usIndexLast*	= 0
6	*.usFlags*	Mögliche Flags: Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv) Alle anderen Bits = 0.
16	*.usMode*	Reserviert, = 0 setzen

Alle anderen Strukturelemente werden nicht ausgewertet.

Datum lesen:

dDate = mdd8_read_channel_dword(handle);

Datum setzen:

mdd8_write_channel_dword(handle, dDate);

Format von dDate (32 Bit):

31 24	23 16	15 8	7 0
Jahr (0..99)	Monat (1..12)	Tag (1..31)	Wochentag (0=Sonntag, 1=Montag, ...)

O.7.7. Timer

Die MODULAR-4/486 Basiskarte verfügt über sechs Timer, die z.B. zur Generierung von Abtastraten verwendet werden können. Der Timer löst zyklisch Interrupts in einem programmierbaren Zeitraster aus.

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_TIMER* = 0501h
2	*.usIndexFirst*	Device-Index: *ML8_TIMER_A (16 Bit)* = 0 *ML8_TIMER_B (16 Bit)* = 1 *ML8_TIMER_C (16 Bit)* = 2 *ML8_TIMER_D* = 3 *ML8_TIMER_E (16 Bit)* = 4 *ML8_TIMER_F (16 Bit)* = 5
4	*.usIndexLast*	= *.usIndexFirst*
6	*.usFlags*	Mögliche Flags: Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv) Alle anderen Bits = 0.
16	*.usMode*	Reserviert, = 0 setzen

Alle anderen Strukturelemente werden nicht ausgewertet.

Das Intervall zwischen zwei Interrupts wird beim Setzen des Timers mit einem Schreibzugriff in µs angegeben. Der aktuelle Timerstand (in µs) kann mit einem Lesezugriff abgefragt werden:

Setzen: **mdd8_write_channel_dword(handle, ulInterval);**

Lesen: **ulTimer = mdd8_read_channel_dword(handle);**

Gestartet bzw. angehalten wird der Timer mit:

mdd8_send_channel_command(handle, CMD_START); bzw.

mdd8_send_channel_command(handle, CMD_STOP);

Nach dem Öffnen eines Timer-Kanals befindet sich dieser im STOP-Zustand.

! *Die Timer-D, -E und -F können nur zur Interruptgenerierung verwendet werden. Das Lesen des aktuellen Timerwertes ist nicht möglich. Timer-E und -F können nur verwendet werden, wenn die seriellen Schnittstellen der MODULAR-4/486 nicht verwendet werden. Timer-D unterstützt nur folgende Frequenzen: 64 Hz (15,625 ms), 1 Hz (1 Sekunde), 0,0166 Hz (1 Minute) und 0,000277 Hz (1 Stunde).*

O.7.8. Rechteckgenerator

Mit diesem Device ist es möglich, Rechtecksignale zu generieren. Das Puls- Pausen-Verhältnis kann programmiert werden. Das Rechtecksignal kann z.B. mit dem SPB-Modul M-D40-2 ausgegeben werden.

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_SQRWAVE* = 0901h
2	*.usIndexFirst*	Device-Index: *ML8_SQRWAVE_A (Timer-A)* = 0 *ML8_SQRWAVE_B (Timer-B)* = 1 *ML8_SQRWAVE_C (Timer-C)* = 2
4	*.usIndexLast*	= *.usIndexFirst*
6	*.usFlags*	Mögliche Flags: Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv) Alle anderen Bits = 0.
16	*.usMode*	Reserviert, = 0 setzen

Alle anderen Strukturelemente werden nicht ausgewertet.

Ausgabefrequenz (Periodendauer in µs) setzen:

mdd8_write_channel_dword(handle, ulInterval);

Aktiviert und deaktiviert wird der Kanal mit dem Sonderdienst **mdd8_send_channel_command(handle, CMD_START);**

bzw.

mdd8_send_channel_command(handle, CMD_STOP);

Nach dem Öffnen des Kanals befindet sich dieser im STOP-Zustand.

Zur Erzeugung des Rechtecksignals werden die Devices Timer-A, -B und -C der MODULAR-4/486 verwendet. Sind bereits Timer-Kanäle geöffnet, ist das Device nicht mehr als Rechteckgenerator einsetzbar. Das Puls/Pausenverhältnis beträgt bei MODULAR-4/486 immer 1:1. !

O.7.9. Watchdog

Ein Watchdog kann zur Überwachung der Software auf der MODULAR-4/486 verwendet werden. Ist der Watchdog aktiviert, muß er von der Software regelmäßig (vor Ablauf einer Timeout-Zeit) nachgetriggert werden, andernfalls erfolgt ein Interrupt. Unter dem Interrupt muß eine Interrupt Task installiert werden, die auf das Überschreiten der Timeoutzeit reagiert.

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_WATCHDOG* = 0503h
2	*.usIndexFirst*	Device-Index: *ML8_WATCHDOG (NMI)*[1] = 0 *ML8_WATCHDOG_A (Timer-A Interrupt)* = 1 *ML8_WATCHDOG_B (Timer-B Interrupt)* = 2 *ML8_WATCHDOG_C (Timer-C Interrupt)* = 3
4	*.usIndexLast*	= *.usIndexFirst*
6	*.usFlags*	Mögliche Flags: Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv) Alle anderen Bits = 0.
12	*.ulTimeout*	Timeoutzeit in µs
16	*.usMode*	Reserviert, = 0 setzen

Alle anderen Strukturelemente werden nicht ausgewertet.

Die Timeoutzeit des Watchdogs wird beim Öffnen des Kanals festgelegt (Angabe in µs).

Aktiviert und deaktiviert wird der Watchdog mit dem Sonderdienst **mdd8_send_channel_command(handle, CMD_START);**

bzw.

mdd8_send_channel_command(handle, CMD_STOP);

Das Nachtriggern des Watchdog erfolgt durch Aufruf der Funktion

mdd8_trigger_channel(handle);

Der aktuellen Zustand des Watchdogs (0 = deaktiviert, 1 = aktiviert, 2 = Timeout) wird mit

status = mdd8_get_channel_info(handle, *INFO_DEVICE* = 48h);

abgefragt.

[1] Die Timeout-Zeit wird mit Jumper J4 fest eingestellt. Möglich sind 100 ms oder 1,6 s. Der CPS-Parameter .ulTimeout hat keine Bedeutung.

O.7.10. Serielle Schnittstellen

Zur Verwendung der seriellen Schnittstellen der Basiskarte (RS-232, asynchron) öffnen Sie den Kanal wie folgt:

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_SCC* = 0801h
2	*.usIndexFirst*	Device-Index: *ML8_SCC_A* = serielle Schnittstelle A = 0 *ML8_SCC_B* = serielle Schnittstelle B = 1
4	*.usIndexLast*	= *.usIndexFirst*
6	*.usFlags*	Mögliche Flags: Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv) Alle anderen Bits = 0.
16	*.usMode*	Betriebsart, s.u.

Alle anderen Strukturelemente werden nicht ausgewertet.

Des Senden und Empfangen der Nutzdaten erfolgt mit den Datentransferdiensten des Kanals. Die Modemsteuerleitungen können mit den Sonderdiensten des Kanals gesetzt bzw. abgefragt werden.

Vor dem Senden von Zeichen ist es erforderlich, den Status des Sendepuffers zu ermitteln. Nur wenn dieser leer ist, darf ein neues Zeichen gesendet werden.

Ob Zeichen empfangen wurden, sollte vor dem Lesen des Empfangspuffers durch prüfen des Puffers ermittelt werden.

Den Status von Sende- und Empfangspuffer kann man ebenfalls mit Sonderdiensten des Kanals abfragen.

Das CPS-Strukturelement *.usMode* bestimmt die Betriebsart (Baudrate, Datenbits, Stopbits und Parität) des seriellen Kanals. Die Parameter werden oder-verknüpft:

Baudraten:

_ML8_COM_110	= 0000h	110 Baud
_ML8_COM_150	= 0020h	150 Baud
_ML8_COM_300	= 0040h	300 Baud
_ML8_COM_600	= 0060h	600 Baud
_ML8_COM_1200	= 0080h	1200 Baud
_ML8_COM_2400	= 00A0h	2400 Baud
_ML8_COM_4800	= 00C0h	4800 Baud
_ML8_COM_9600	= 00E0h	9600 Baud
_ML8_COM_19200	= 0100h	19200 Baud
_ML8_COM_38400	= 0200h	38400 Baud

Datenbits:

_ML8_COM_CHR7	= 0002h	7 Datenbits
_ML8_COM_CHR8	= 0003h	8 Datenbits

Stoppbits:

_ML8_COM_STOP1	= 0000h	1 Stoppbit
_ML8_COM_STOP2	= 0004h	2 Stoppbits

Parität:

_ML8_COM_NO_PARITY	= 0000h	ohne Parität
_ML8_COM_EVEN_PARITY	= 0018h	gerade Parität
_ML8_COM_ODD_PARITY	= 0008h	ungerade Parität

Soll z.B. ein Kanal mit den Einstellungen: 9600 Baud, 8 Datenbits, 1 Stoppbit, ohne Parität verwendet werden, setzen Sie

```
.usMode =   _ML8_COM_9600 + _ML8_COM_CHR8 + _ML8_COM_STOP1 + _ML8_COM_NO_PARITY;
```

Es stehen folgende Sonderdienste zur Verfügung:

ulStatus = mdd8_get_channel_info(handle, *INFO_DEVICE* = 48h);

liest den Status zurück. Die Bits haben folgende Bedeutung:

Bit 0: Sendepuffer-Status: 1 (TRUE) : Zeichen wurde gesendet, Sendepuffer leer

Bit 1: Empfangspuffer-Status: 1 (TRUE) : Zeichen wurden empfangen, Empfangspuffer voll

Bit 2: CTS-Status: Zustand der CTS Leitung

Bit 3: DCD-Status: Zustand der DCD Leitung

mdd8_send_channel_control(handle, *CTRL_DEVICE*, *ML8_SET_RTS* = 0);
RTS-Leitung setzen (=1)
mdd8_send_channel_control(handle, *CTRL_DEVICE*, *ML8_CLR_RTS* = 1);
RTS-Leitung löschen (=0)

mdd8_send_channel_control(handle, *CTRL_DEVICE*, *ML8_SET_DTR* = 2);
DTR-Leitung setzen (=1)
mdd8_send_channel_control(handle, *CTRL_DEVICE*, *ML8_CLR_DTR* = 3);
DTR-Leitung löschen (=0)

O.7.11. Hardware-Interrupts

Zur Programmierung von Interrupt-Tasks ist es notwendig, den verwendeten Interrupt und seine aktive Flanke festzulegen. Dazu verwenden Sie das Device HW_INT.

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_HW_INT* = 0B01h
2	*.usIndexFirst*	Device-Index: *ML8_INT_NMI (SP-Bus, Spannungsüberw., Watchdog)* = 0 *ML8_INT_IRQ_A (SP-Bus)* = 1 *ML8_INT_IRQ_B (SP-Bus)* = 2 *ML8_INT_IRQ_C (SP-Bus)* = 3 *ML8_INT_IRQ_D (SP-Bus)* = 4 *ML8_INT_IRQ_E (SP-Bus)* = 5 *ML8_INT_IRQ_F (SP-Bus)* = 6 *ML8_INT_IRQ_G (extern, St1)* = 7 *ML8_INT_IRQ_H (extern, St3)* = 8 *ML8_INT_TIMER_A* = 9 *ML8_INT_TIMER_B* = 10 *ML8_INT_TIMER_C* = 11 *ML8_INT_TIMER_D* = 12 *ML8_INT_TIMER_E* = 13 *ML8_INT_TIMER_F* = 14 *ML8_INT_SCC_A* = 15 *ML8_INT_SCC_B* = 16
4	*.usIndexLast*	= *.usIndexFirst*
6	*.usFlags*	Mögliche Flags: Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv) Alle anderen Bits = 0.
16	*.usMode*	Reserviert (=0)

Alle anderen Strukturelemente werden nicht ausgewertet.

Nur bei den Interrupts IRQ-A bis -H kann die aktive Flanke programmiert werden **!**

Der Sonderdienst CTRL_DEVICE stellt folgende Steuerbefehle zur Verfügung:

ML8_UNMASK_INT	= 0	Interrupt maskieren
ML8_MASK_INT	= 1	Interrupt demaskieren
ML8_POS_INT_EDGE	= 2	Positive Interruptflanke setzen
ML8_NEG_INT_EDGE	= 3	Negative Interruptflanke setzen:
ML8_CLR_INT	= 4	Interruptanforderung löschen
ML8_END_INT	= 5	Interrupt beenden (EOI)

Aufruf:

mdd8_send_channel_control(handle, *CTRL_DEVICE*, Steuerbefehle);

O.7.12. EEPROM

Ein Kanal zu einem EEPROM Device ermöglicht es, auf einen Eintrag im EEPROM der Basiskarte oder eines Moduls zuzugreifen. Der Zugriff auf die EEPROM-Inhalte kann direkt oder über eine Kopie im RAM der Karte erfolgen (schneller). In die Kopie geschriebene Daten gehen nach dem Abschalten der Karte verloren.

Ofs	Strukturelement	Wert
0	*.usDevice*	Device-Typ: *DEVICE_EEPROM* = 0A01h
2	*.usIndexFirst*	Device-Index = Modulsteckplatz (0 = Basiskarte)
4	*.usIndexLast*	= *.usIndexFirst*
6	*.usFlags*	Mögliche Flags: Bit 0: *_CP_EXCLUSIVE* = 1 (exklusiv) Alle anderen Bits = 0.
8	*.usReadMode*	Lesemodus: *IO_MODE_DIRECT* = 1 (direkt) *IO_MODE_RAM* = 4 (aus RAM)
10	*.usWriteMode*	Schreibmodus: *IO_MODE_DIRECT* = 1 (direkt) *IO_MODE_RAM* = 4 (aus RAM)

Alle anderen Strukturelemente werden nicht ausgewertet.

EEPROM-Wort anwählen:
mdd8_send_channel_control(handle, *CTRL_RW_PTR*, ulPointer);
EEPROM-Wort lesen bzw. schreiben:
wEEPROM = mdd8_read_channel_word(handle);
mdd8_write_channel_word(handle, wEEPROM);
Dabei wird der Lese-/Schreibzeiger automatisch um 1 inkrementiert!

O.7.13. Programmier-Beispiele

O.7.13.1. Timergesteuertes LED-Blinkprogramm (Echtzeit-Task)

Es sollen die on-board Leuchtdiode und Timer-B verwendet werden. Öffnen Sie die Kanäle wie folgt:

```
ML8CPS.usDevice   = DEVICE_LED;
ML8CPS.usIndexFirst     = ML8_LOCAL_LED;
ML8CPS.usIndexLast      = ML8_LOCAL_LED;
ML8CPS.usFlags    = 0;
LED_Handle = mdd8_open_channel(wTask, sizeof(ML8CPS), &ML8CPS);
ML8CPS.usDevice   = DEVICE_TIMER;
ML8CPS.usIndexFirst     = ML8_TIMER_B;
ML8CPS.usIndexLast      = ML8_TIMER_B;
ML8CPS.usFlags    = 0;
TimerB_Handle = mdd8_open_channel(wTask, sizeof(ML8CPS), &ML8CPS);
ML8CPS.usDevice   = DEVICE_HW_INT;
ML8CPS.usFlags    = 0;
ML8CPS.usIndexFirst     = ML8_INT_TIMER_B;
ML8CPS.usIndexLast      = ML8_INT_TIMER_B;
IRQ_TimerB_Handle = mdd8_open_channel(wTask, sizeof(ML8CPS), &ML8CPS);
```

Starten Sie den Timer und geben Sie den Interrupt frei:

```
mdd8_write_channel_dword(TimerB_Handle, 1000); // Intervall = 1000µs
mdd8_send_channel_command(TimerB_Handle, CMD_START);
mdd8_send_channel_control(IRQ_TimerB_Handle, CTRL_DEVICE, UNMASK_INT);
```

Damit wird die Task gestartet. Alle 1000 µs wird die Hauptprozedur der Task per Interrupt aufgerufen. In der Hauptprozedur der Task, die unter Timer-B Interrupt als II oder DI Task installiert werden muß, schaltet man die Leuchdiode ein bzw. aus.

```
mdd8_write_channel_byte(LED_Handle, 1); // LED einschalten
mdd8_write_channel_byte(LED_Handle, 0); // LED ausschalten
```

Wurde das Programm als direkte Interrupt-Task (DI) installiert, so muß in der Hauptprozedur der aufgetretene Interrupt gelöscht werden:

```
mdd8_send_channel_control(IRQ_TimerB_Handle, CTRL_DEVICE, END_OF_INT);
```

O.7.13.2. Verwendung der seriellen Schnittstellen (Echtzeit-Task)

Es sollen über die serielle Schnittstelle A der MODULAR-4/486 Daten empfangen und gesendet werden (asynchrone Kommunikation). Die Einstellungen für die Schnittstelle sollen sein: 4800 Baud, 7 Datenbits, 1 Stopbit, keine Parität.

Öffnen Sie den Kanal wie folgt:

```
ML8CPS.usDevice   = DEVICE_SCC;
ML8CPS.usIndexFirst     = ML8_SCC_A;
ML8CPS.usIndexLast      = ML8_SCC_A;
ML8CPS.usFlags    = 0;
ML8CPS.usMode     = _COM_4800 | _COM_CHR7 | _COM_STOP1 | _COM_NOPARITY;
SCC_Handle = mdd8_open_channel(wTask, sizeof(ML8CPS), &ML8CPS);
```

In der Hauptprozedur der Task, die als NI-Task zyklisch vom Betriebssystem aufgerufen wird, werden die empfangenen Zeichen eingelesen:

```
                              // prüfen, ob Receive Buffer Full
if(mdd8_get_channel_info(SCC_Handle, INFO_DEVICE) & CHECK_RBF)
bData = mdd8_read_channel_byte(SCC_Handle); // Zeichen einlesen
```

In einer weiteren Prozedur der Task, können Zeichen gesendet werden:

```
                              // prüfen, ob Transmit Buffer Empty
if(mdd8_get_channel_info(SCC_Handle, INFO_DEVICE) & CHECK_TBE)
mdd8_write_channel_byte(SCC_Handle, bData); // Zeichen senden
```

Sollen die Modemsteuerleitungen gesetzt werden, verwenden Sie folgende Aufrufe:

```
mdd8_send_channel_control(SCC1Handle, CTRL_DEVICE, SET_DTR);
mdd8_send_channel_control(SCC1Handle, CTRL_DEVICE, CLR_DTR);
mdd8_send_channel_control(SCC1Handle, CTRL_DEVICE, SET_RTS);
mdd8_send_channel_control(SCC1Handle, CTRL_DEVICE, CLR_RTS);
```

P. Stichwortverzeichnis

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z