Multi-COM

SORCUS

4. Auflage
14. Februar 2005
SORCUS Computer GmbH
Im Breitspiel 11
69126 Heidelberg

Inhaltsverzeichnis

13. Asynchrone serielle Kommunikation 13-1

Anhang

1. Das Multi-COM System

Multi-COM ist ein modulares Kommunikationssystem auf der Basis einer intelligenten PC-Zusatzkarte.

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

> Die Multi-COM Karte ist hardwaremäßig ein kompletter, unabhängiger 486-Computer auf einer PC-Zusatzkarte.

Durch die freie Programmierbarkeit ist die Multi-COM 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 Daten und 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 Multi-COM Karten um ein Vielfaches gesteigert werden: Der PC wird so zu einem Multi-Prozessor-System.

Hinzu kommt, dass die Multi-COM Karte hardwaremäßig durch 5 aufsteckbare S-Links an viele Kommunikationsaufgaben angepasst werden kann. Dies hat für den Entwickler den Vorteil, daß er sich nur in ein System "eindenken" muss. Die Karte ist außerdem praktisch 100% kompatibel zur MODULAR-4/486 Karte.

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 Multi-COM Basiskarte enthält bereits eine Reihe von Funktionen und Schnittstellen:

- 10 Timer (3 x 16-Bit Timer im Baustein 8254, 1 Taktgeber in der Uhr, 6 x 16-Bit Timer im Baustein SCC 8530 bzw. ESCC 85230), alle interruptfähig (die beiden Timer im SCC 8530 werden standardmäßig als Baudratengeneratoren verwendet).
- Watchdog Timer für die lokale 486-CPU
- 6 serielle Schnittstellen, davon 1 fest RS-232
- Uhrzeit/Datum, über externe Batterie pufferbar
- Cache-RAM on-board
- Arithmetik-Coprozessor (nur bei 486-DX CPUs)
- Interrupt-Eingänge
- LED on-board
- Ausgang für eine weitere LED
- 5 Steckplätze für S-Links (= Mikro-Module)
- Überwachung der Versorgungsspannung mit NMI-Auslösung bei Power-Fail.

Ebenfalls enthalten sind RAM (standardmäßig 512 KByte stat. RAM, alternativ auch mit 2 MByte, 10 MByte oder 34 MByte 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 (z.B. Kommunikationsprotokolle) 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 und PASCAL 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 (interruptfähig) zur Verfügung. Darüber können in beiden Richtungen gleichzeitig Daten und Programme ausgetauscht werden.

Zur Entwicklungsunterstützung für die Karte finden Sie auf den mitgelieferten Disketten das Test-, Service- und Debug-Programm SNW6 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 MODULAR-4 Karten

Es sind nur wenige Punkte zu beachten, wenn Sie schon mit einer MODULAR-4/486 (ML8) Karte gearbeitet haben und nun eine Multi-COM Karte (ML6) einsetzen wollen.

Die Multi-COM Karte verhält sich wie eine MODULAR-4/486 mit 3 M-COM-2 Modulen auf Steckplatz 1, 2 und 3 (serielle Schnittstellen A bis F). Die seriellen Schnittstellen A und B der MODULAR-4/486 können bei Multi-COM Karten auch als Modul M-COM-2 auf Steckplatz 1 angesprochen werden. Die Multi-COM enthält keine Jumper. Alle Einstellungen (z.B. Interrupt-Kanal, Watchdog, usw.) werden entsprechend den EEPROM-Inhalten per Software durchgeführt.

ser. Schnittstelle der MODULAR-4/486	**ser. Schnittstelle der Multi-COM**
Modul M-COM-2, Kanal A (Steckplatz 1)	A (S-Link), IRQ-D (7eh)
Modul M-COM-2, Kanal B (Steckplatz 1)	B (RS-232), IRQ-D (7eh)
Modul M-COM-2, Kanal A (Steckplatz 2)	C, (S-Link) IRQ-C (7dh)
Modul M-COM-2, Kanal B (Steckplatz 2)	D, (S-Link) IRQ-C (7dh)
Modul M-COM-2, Kanal A (Steckplatz 3)	E, (S-Link) IRQ-B (7ch)
Modul M-COM-2, Kanal B (Steckplatz 3)	F, (S-Link) IRQ-B (7ch)
RS-232 A (on-board)	A (S-Link), IRQ-SCC (90h), alternativ
RS-232 B (on-board)	B (RS-232), IRQ-SCC (90h), alternativ

Die Bedeutung der Wörter im EEPROM der Karten ist unterschiedlich.

1.2. Lieferumfang

Zum Lieferumfang gehört:

1. Die Multi-COM 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, Einstellungen, usw.
3. Dieses Handbuch
4. Diskette/n bzw. CD, die folgendes enthalten:
   - Hilfs- und Testprogramme
   - PC Programmbibliotheken, z.B. für Pascal, C und Basic (Plattform-abhängig)
   - PC Programmbeispiele, z.B. in Pascal und C
   - Beispiele für on-board Echtzeit-Programme in Pascal, C und Assembler
   - On-board-Programme zur gepufferten seriellen Kommunikation (CQ6)

1.3. CE-Kennzeichnung

Das Multi-COM System ist ein OEM-Produkt und für die Weiterverarbeitung durch Industrie, Handwerk oder sonstige auf dem Gebiet der elektromagnetischen Verträglichkeit fachkundigen Betriebe bestimmt. Im Sinne des EMVG vom 18. September 1998 §6 Abs. 9 besteht daher für das Multi-COM 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 das Multi-COM System eingesetzt wurde, 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 Multi-COM-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 Multi-COM Karte als auch der einzelnen S-Link Adapter.
2. Erstellen einer Arbeitskopie der mitgelieferten Diskette/n bzw. CD (siehe Seite 2-2).
3. Auspacken der Multi-COM Karte und der S-Links (Vorsicht vor elektrostatischen Aufladungen!).
4. Überprüfen (!) und Einstellen der gewünschten Konfiguration der Basiskarte (z.B. PC-I/O-Adresse, siehe Abschnitt 2.2.).
5. 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.
6. Aufstecken der S-Links.
7. 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 dass irgendwelche Kabel an die Karte angeschlossen sind), dann Einschalten des PC.

   Die Karte wird, wie andere Karten auch, in den PC eingesteckt und mit einer Schraube am Bügel gesichert. Der Platz, an dem die Karte eingesteckt wird, ist im Prinzip gleichgültig.

   Es ist besonders darauf zu achten, dass die Karte weder mechanisch noch elektrisch Kontakt zu einer benachbarten Karte hat. **!**

8. Starten des Programms "SNW6" oder "SNW32" und Überprüfen aller Schnittstellen.

Falls beim Einsatz mit der Karte einmal Probleme auftauchen sollten, ist das Programm SNW6 oder SNW32 auch geeignet, um einen möglichen Fehler auf der Karte oder auf den S-Links zu lokalisieren. Sind die einzelnen Tests erfolgreich verlaufen, können die gewünschten externen Anschlüsse aufgesteckt werden.

Die Karte ist nun betriebsbereit.

2.1. Erstellen von Arbeitsdisketten

Die beiliegenden Disketten (bzw. CD) sind nicht kopiergeschützt. Die Programme dürfen für Zwecke des Anwenders beliebig oft kopiert werden. Wiederverkäufer können die Programme auch verändern und weiter verkaufen. Eine besondere Genehmigung der SORCUS Computer GmbH ist dazu nicht erforderlich.

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

Da sich die Zahl der Programme auf den mitgelieferten Originaldisketten bzw. CD entsprechend dem aktuellen Entwicklungsstand ändert, ist an dieser Stelle kein Inhaltsverzeichnis angegeben. Sie können aber jederzeit gegen eine geringe Gebühr eine aktualisierte Version bei Ihrem Händler, direkt bei SORCUS Computer GmbH erhalten oder von der SORCUS Homepage (www.sorcus.com) herunterladen.

Alle Programme auf den Originaldisketten sind als Hilfsmittel gedacht, um Ihnen den Einsatz der Multi-COM 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 Multi-COM Karte sowie der mitgelieferten Software ergeben könnten, ausgeschlossen.

2.2. Einstellungen auf der Karte

Vor dem Einbau müssen Sie die PC I/O-Adresse der Karte (mit Drehschalter S1 oder 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 Multi-COM gespeichert (siehe Anhang L).

2.2.1. Einstellen der PC-I/O-Adresse (S1, J1)

Die Karte belegt 8 Adressen im I/O-Adreßbereich des PC (siehe Tabelle der reservierten I/O-Adressen des PC-AT auf der nächsten Seite). 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. Die eingestellte Adresse muss durch 8 teilbar sein.

Drehschalter S1 (bzw. Jumper J1) legt die Basisadresse (BA) 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

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. Netzkarten 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

2.2.2. 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 Bibliothek ML6BIB 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.

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

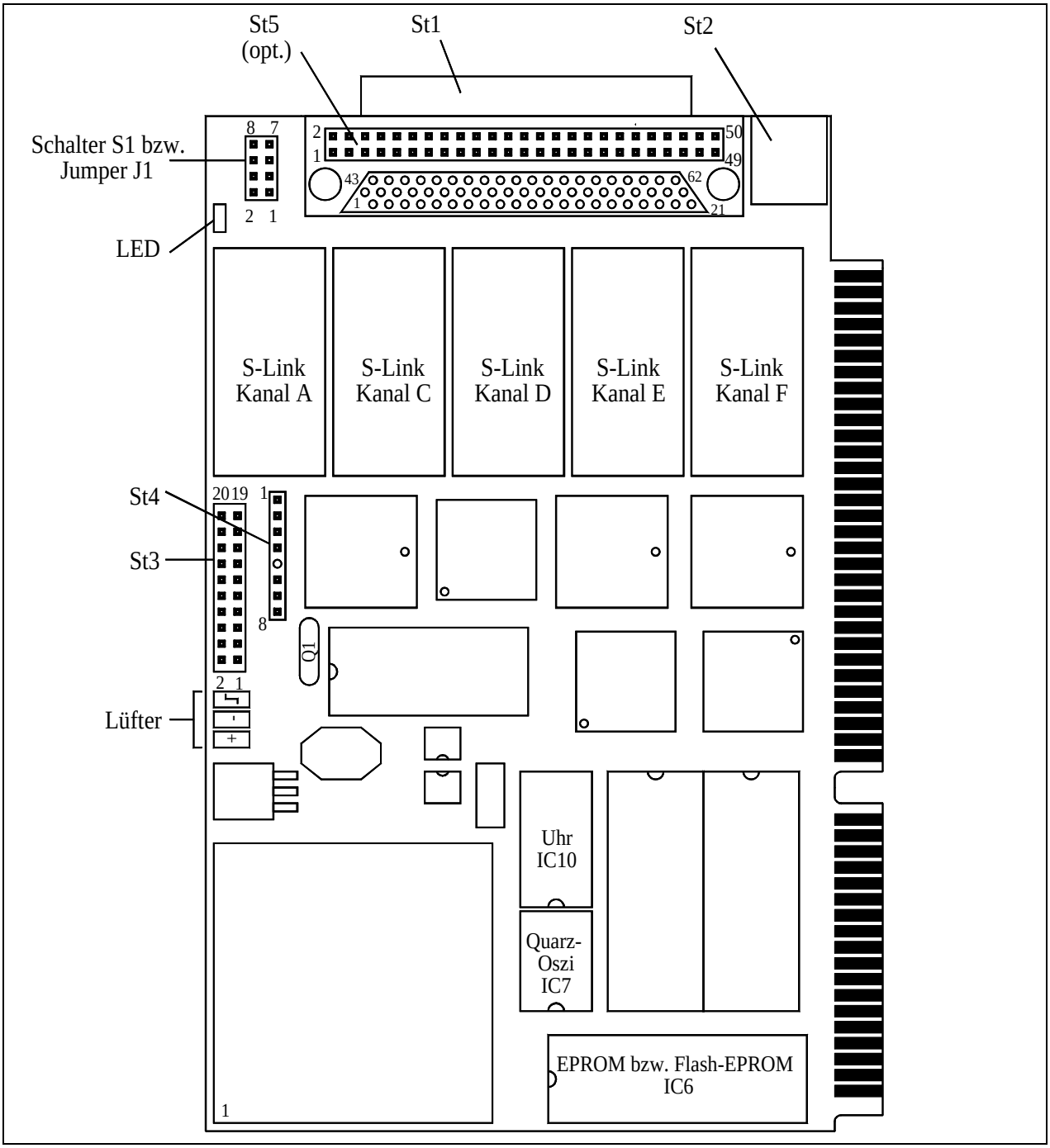
Diese Liste ist nicht unbedingt vollständig und kann sich an einigen Stellen auch von PC-Hersteller zu PC-Hersteller unterscheiden. Zusätzlich eingesteckte Karten, z.B. Netzkarten belegen evtl. die freien Interrupteingänge.

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

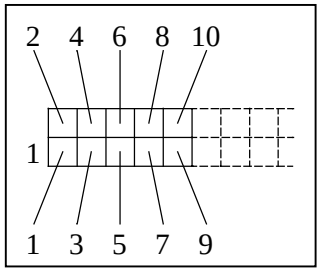
2.2.3. Anschluß einer externen Batterie (St3)

Zur Pufferung des statischen RAMs und der Uhr auf der Multi-COM Karte kann an St3 eine Batterie angeschlossen werden (siehe Kap. 3). Sie wird von der Karte automatisch abgeschaltet, sobald die 5 Volt Versorgungsspannung der Karte vorhanden ist. Es können Batterien mit 3 - 4 Volt eingesetzt werden.

2.3. Lageplan



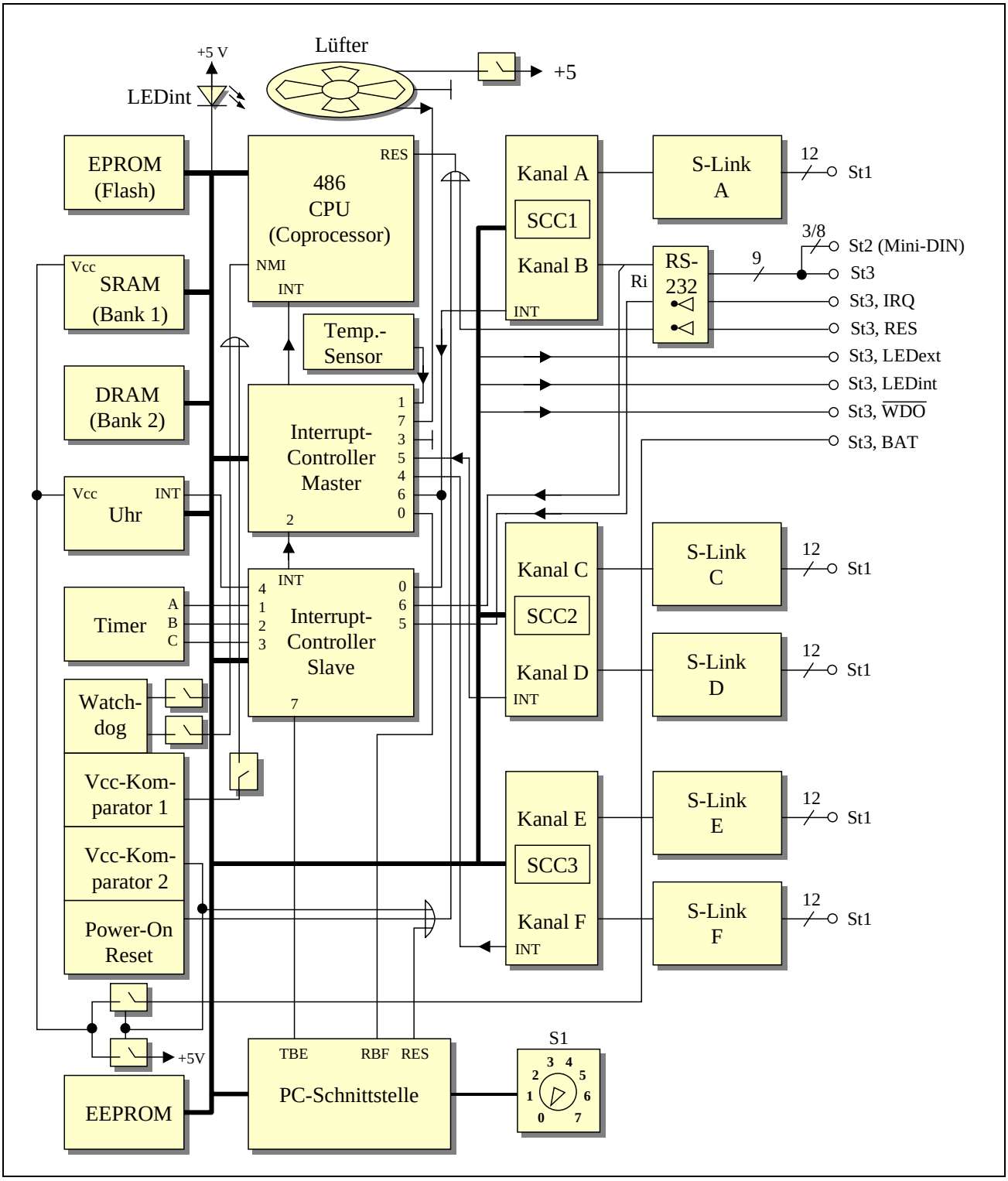
Auf der Abbildung der Karte sind die Stecker mit St1 bis St5 bezeichnet. Eine "1" oder "o" kennzeichnet Pin 1 eines Steckers oder eines ICs. Die nebenstehende Abbildung zeigt die Zählweise bei zweireihigen Pfostensteckern.



Notizen

3. Funktionseinheiten der Multi-COM

3.1. Blockschaltbild

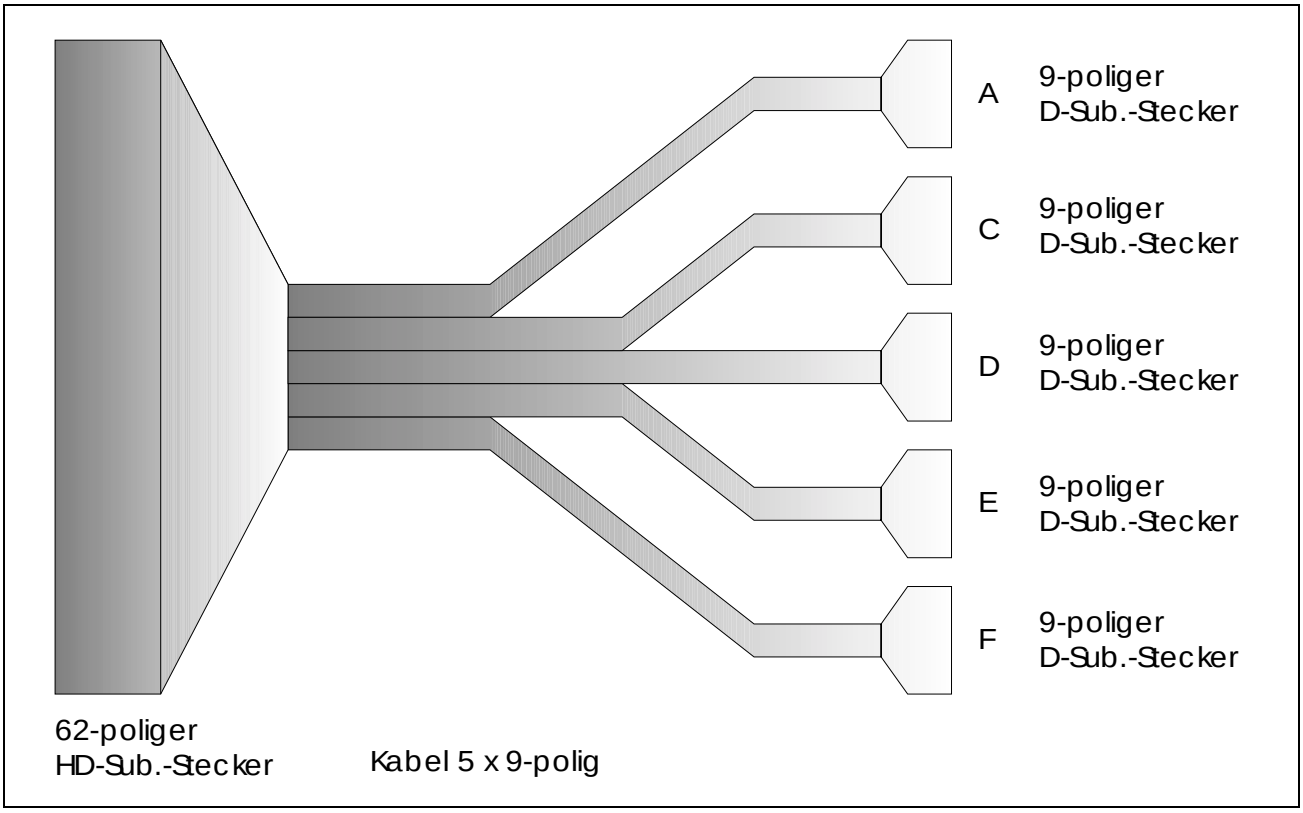


3.2. Stecker und Buchsen

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

Stecker / Buchse	**Typ (auf der Karte)**	**Funktion**
St1	62-pol. D-Sub. Buchse	Serielle Schnittstellen: Kanal A, C, D, E, F
St2	3-pol.oder 8-pol. Mini-DIN Buchse	Serielle Schnittstelle B (z.B. für Remote-Debugging)
St3	20-pol. Pfostenstecker mit Wanne (2x10)	5 Volt Versorgung Eingang/Ausgang ±12 Volt Versorgung Eingang/Ausgang Interrupt-Eingang IRQ-G Eingang für externe Batterie LED-Ausgänge (LEDext und LEDint) Hardware-Reset Eingang Watchdog-Ausgang (/WDO) Serielle Schnittstelle B
St4	8-pol. (1x8) Pfosten-stecker	Diagnose-System (nicht für Anwendungen verwendbar)
St5	50-pol. (2x25) Pfostenstecker	Serielle Schnittstellen: Kanal A, C, D, E, F (standardmäßig nicht bestückt)

3.2.1. Kontaktbelegung St1 und Kabel K2-6259 und K3-6260



S-Link	**Pin**		**62-pol. Buchse St1**				
Pin	**(9-polig)**	**Bedeutung**	**Kanal A**	**Kanal C**	**Kanal D**	**Kanal E**	**Kanal F**
A1	1	DCD	4	8	12	16	20
A3	2	RCV	45	49	53	57	61
A5	3	TMT	3	7	11	15	19
A20	4	DTR	44	48	52	56	60
A22	5	GND	43	47	51	55	59
A4	6	DSR	25	29	33	37	41
A6	7	RTS	24	28	32	36	40
A19	8	CTS	23	27	31	35	39
A21	9	RI	2	6	10	14	18
A23	–	EXT	22	26	30	34	38
A2	–	STB	46	50	54	58	62
A24	–	BSY	1	5	9	13	17

Anm.: Pin 21 und 42 sind reserviert und müssen unbeschaltet bleiben.

3.2.2. Serielle Schnittstelle B (St2)

An St2 (3-pol. oder 8-pol. Mini-DIN-Buchse) ist die serielle Schnittstelle B (z.B. für Remote-Debugging) herausgeführt. Für den Übergang von 3-pol. Mini-DIN auf 9-pol. D-Sub. ist ein Kabel lieferbar (Best.-Nr. K2-4003), das die Standardbelegung der seriellen Schnittstelle (bezüglich RCV, TMT und GND) wie beim IBM-PC/AT zur Verfügung stellt. Es kann direkt an eine serielle Schnittstelle des PC angeschlossen werden. Die serielle Schnittstelle B kann alternativ auch an Stecker St3 angeschlossen werden. Es darf zur selben Zeit aber immer nur eine der beiden Stecker bzgl. Schnittstelle B angeschlossen sein.

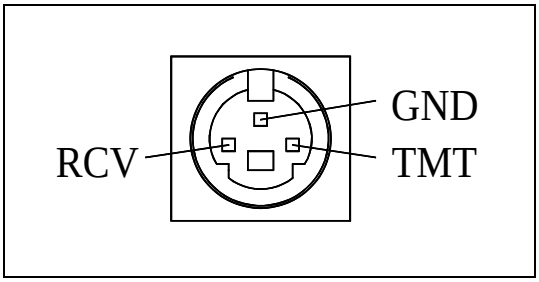


Abb. 3-1: *St2 (3-pol. Mini-DIN)*

Signal Bezeichnung	**St2, Mini-DIN (3-pol.)**	**(8-pol.)**	**St3, Pfostenstecker (20-pol.)**	**D-Sub. (9-pol.)**
DTR	-	1	7	4
RTS	-	2	4	7
TMT	1	3	5	3
CTS	-	4	6	8
RCV	2	5	3	2
DCD	-	6	1	1
GND	3	7	9	5
RI	-	8	8	9
DSR	-	-	2	6

3.2.3. Stecker St3

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	LEDext: Ausgang für externe LED (Kathode), Anode an +5 Volt anschließen.
11	Ausgang	Watchdog-Ausgang, TTL-Pegel, active low
12	Eingang	Hardware-Reset-Eingang, TTL, act. low, int. Pull-up-Widerstand
13	Ein-/Ausgang	+5 Volt
14	Ausgang	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

3.2.3.1. Eingang für Hardware-Reset der Karte (St3, 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.

3.2.3.2. LEDext (St3, 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.

3.2.3.3. LEDint (St3, 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.

3.2.3.4. Batterieanschluss (St3, 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 St3, Minuspol an Pin 9 oder 18 von St3. Wenn keine Batterie angeschlossen werden soll, bleibt Pin 17 unbeschaltet.

3.2.3.5. Watchdog-Timer Ausgang (St3, Pin 11)

Soll das Ablaufen des Watchdog-Timers (siehe Kap. 3.3.) einen Hardware-Reset der Multi-COM Karte auslösen, besteht die Möglichkeit, an Stecker St3 die Pins 11 (Watchdog-Ausgang) und 12 (Reset-Eingang) per Jumper zu verbinden (siehe auch Kapitel 3.3).

3.3. Watchdog-Timer und Spannungsüberwachung

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

3.3.1. Watchdog-Timer

Zur Überwachung der laufenden Echtzeit-Software (Tasks) stellt die Multi-COM 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 Multi-COM 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 Multi-COM 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
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 Multi-COM oder per Software aktiviert werden.
2. Nach der Aktivierung des Watchdogs wird dieser 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 **ml6rt_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 (**ml6rt_unmask_int(NMI)**).

Soll das Ablaufen des Watchdog-Timers einen Hardware-Reset der Multi-COM auslösen, besteht die Möglichkeit, an Stecker St3 die Pins 11 (Watchdog-Ausgang) und 12 (Reset-Eingang) per Jumper zu verbinden.

---

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

[2] Zustand nach Reset

3.3.2. Spannungsüberwachung

Die Multi-COM 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.

3.3.2.1. 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.

3.4. 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 Multi-COM und der S-Links.
- Durch Verbinden der Pins 14 und 16 an Stecker St3 wird nach einem Reset automatisch das ROM-OsX aktiviert. Die Multi-COM Karte und die seriellen Schnittstellen werden entsprechend den Einstellungen im EEPROM initialisiert.

3.5. Serielle Schnittstellen, Funktionsbeschreibung

Die Multi-COM Karte stellt 6 unabhängige serielle synchrone/asynchrone Schnittstellen, Kanal A bis F, zur Verfügung. Dazu ist die Karte mit 3 SCC-Bausteinen (SCC = Serial Communication Controller) vom Typ 85C30 bzw. 85230 (= verbesserte und erweiterte Version des SCC mit größeren FIFOs) ausgerüstet. Je SCC stehen 2 unabhängige Kanäle zur Verfügung. So können z.B. die Betriebsart, Übertragungsparameter und die Baudrate für jeden Kanal getrennt eingestellt werden.

Die Konfiguration der physikalischen Schnittstellen für Kanal A, C, D, E und F erfolgt über aufsteckbare S-Link-Adapter und ist damit an alle üblichen Pegel anpassbar. Jede dieser Schnittstellen ist über einen S-Link-Adapter getrennt konfigurierbar.

Außerdem enthält die Karte je SCC einen programmierbaren Quarzoszillator und 2 Baudratengeneratoren. Für besondere Einsatzbedingungen ist die Karte auch so konfigurierbar, dass die Baudraten für Senden und Empfangen bei einem Kanal je SCC unterschiedlich eingestellt werden können. Außerdem kann je nach S-Link der Sende- und/oder Empfangstakt von außen bzw. ein Takt nach außen geliefert werden. Die Richtung ist per Software umschaltbar, vorausgesetzt, der aufgesteckte S-Link-Adapter ist hierfür vorbereitet. Gleiches gilt für die automatische Umschaltung von Senden auf Empfangen, z.B. bei RS-485-Schnittstellen. Die Umschaltung kann per Software oder automatisch durch ein Steuersignal des SCC erfolgen.

Alle seriellen Kanäle sind interruptfähig.

Tab. 3-1: *Zuordnung der seriellen Schnittstellen zu den 3 SCCs*

Kanal	SCC	Quarzoszillator	Physikalische Schnittstelle
A	1	1	per S-Link konfigurierbar
B	1	1	RS-232 mit allen Modem-Steuerleitungen (TMT, RCV, RTS, CTS, DTR, DSR, RI, DCD)
C	2	2	per S-Link konfigurierbar
D	2	2	per S-Link konfigurierbar
E	3	3	per S-Link konfigurierbar
F	3	3	per S-Link konfigurierbar

3.5.1. Serielle Schnittstellen, Programmierung

Alle 6 seriellen Kanäle können unabhängig voneinander konfiguriert werden. Zusätzlich zu den in dieser Beschreibung angegebenen Konfigurationsmöglichkeiten bestehen noch weitere Möglichkeiten, die ggf. die werksseitige Umkonfiguration eines IC (CPLD) erforderlich machen. Falls Sie spezielle Konfigurationen benötigen, sollten Sie diese anfragen. Alle in dieser Beschreibung gemachten Angaben beziehen sich auf die Standard-Bestückung von Multi-COM Karte und S-Links. Es sind auch kundenspezifische S-Links möglich.

3.5.1.1. Anwahl einer Interrupt-Leitung

Die seriellen Schnittstellen A bis F sind interruptfähig. Die Interrupt-Leitung der SCCs ist fest mit je einem Interrupt der Multi-COM Karte verbunden.

Kanal	Interrupt-Leitung an Multi-COM
Kanal A und B (SCC1)	IRQ-SCC1 (= IRQ-Master 6 und IRQ-Slave 0)
Kanal C und D (SCC2)	IRQ-SCC2 (IRQ-Master 5)
Kanal E und F (SCC3)	IRQ-SCC3 (IRQ-Master 4)

3.5.1.2. Programmierung des SCC-Bausteins

Da dieser Baustein etwas komplex ist, wird hierzu auf die Literatur von Zilog (zum Baustein Z8530, Z85C30 und Z85230) bzw. Intel und AMD (82530) verwiesen. Dabei sind die Bausteine Z8530, Z85C30 und 82530 untereinander kompatibel, der Baustein Z85230 stellt eine kompatible und erweiterte Version dar (u. a. größere FIFOs).

Ein ausführliches Programmierhandbuch (in engl. Sprache) zu diesen Bausteinen ist bei SORCUS erhältlich (Best.-Nr.: MA-1529).

Bedingt durch die Konstruktion der Karte sind bei der Programmierung des SCC einige Dinge zu berücksichtigen:

1. Der Pin SYNC des SCC wird nicht für einen Quarzoszillator eingesetzt (siehe SCC-Beschreibung).

2. Der Pin RTxC des SCC muß als Eingang konfiguriert werden. Er wird nicht für einen Quarzoszillator eingesetzt (siehe SCC-Beschreibung). Das an diesem Pin anliegende Signal kann innerhalb des SCC für den jeweiligen Kanal für folgende Zwecke verwendet werden:

Pin	**In-/Output**	**Verwendbar als**
RTxC	Input	RCV-Clock TMT-Clock Eingang der DPLL Eingang von Baudratengenerator

3. Der Pin TRxC des SCC muß je nach verwendetem S-Link als Ein- oder Ausgang konfiguriert werden (Bit-2 in SCC-Write-Register 11). Als Ausgang können folgende Signale nach außen geliefert werden:

Pin	**In-/Output**	**Verwendbar als**
TRxC	Output	TMT-Clock Ausgang von Baudratengenerator Ausgang von RCV DPLL Signal an RTxC

Als Eingang kann TRxC folgendermaßen eingesetzt werden:

Pin	**In-/Output**	**Verwendbar als**
TRxC	Input	TMT-Clock RCV-Clock

4. Am Pin PCLK jedes SCC liegt der Takt eines diesem SCC zugeordneten Quarzoszillators. Dieses Signal kann z.B. als Eingangstakt für die Baudratengeneratoren dienen. Der Quarzoszillator ist per Software umschaltbar zwischen 7,3728 MHz und 4,9152 MHz. Auf Wunsch sind auch andere Frequenzen möglich.

3.5.1.3. Initialisierung eines SCC

Folgende Reihenfolge muß bei der Initialisierung eines SCC eingehalten werden:

Register	Data	Kommentar
1. Stufe: Betriebsarten und Konstanten definieren		
WR9	1100 0000	Hardware Reset
WR0	0000 00xx	Select Shift Mode (nur bei Z8030)
WR4	xxxx xxxx	Transmit/Receive Control: Asyn- oder Synchrone Betriebsart anwählen
WR1	0xx0 0x00	W/REQ anwählen (optional)
WR2	xxxx xxxx	Interrupt-Vektor programmieren
WR3	xxxx xxx0	Receiver Control, Bit D0 (Rx enable) muss hier auf 0 gesetzt werden.
WR5	xxxx 0xxx	Transmit Control, Bit D3 (Tx enable) muss hier auf 0 gesetzt werden.
WR6	xxxx xxxx	SYNC-Zeichen programmieren
WR7	xxxx xxxx	SYNC-Zeichen programmieren
WR9	000x 0xxx	Interrupt Control anwählen. Bit D3 (MIE) muss auf 0 gesetzt werden.
WR10	xxxx xxxx	Kontrollwort (optional)
WR11	xxxx xxxx	Clock Control
WR12	xxxx xxxx	Zeitkonstante, Lowbyte (optional)
WR13	xxxx xxxx	Zeitkonstante, Highbyte (optional)
WR14	xxxx xxx0	Kontrollwort, Bit 0 (BR enable) muss hier auf 0 gesetzt werden.
WR14	xxxS SSSS	Register kann mehrfach beschrieben werden, wenn mehr als ein Kommando geschickt werden soll
2. Stufe: Enables		
WR3	SSSS SSS1	Bit D0 (Rx Enable) auf 1 setzen.
WR5	SSSS 1SSS	Bit D3 (Tx Enable) auf 1 setzen
WR0	1000 0000	Reset TxCRC
WR14	000S SSS1	Baudraten Generator Enable, Bit D0 auf 1 setzen, Enable DPLL
WR1	xSS0 0S00	D7 auf 1 setzen, wenn DMA enabled werden soll.
Stufe 3: Interrupt Enable		
WR15	xxxx xxxx	Enable external interrupts
WR0	0001 0000	Reset EXT/STATUS zweimal
WR0	0001 0000	Reset EXT/STATUS zweimal
WR1	SSSx xSxx	Enable receive, transmit and external interrupt master.
WR9	000S xSSS	Enable Master Interrupt bit D3.

1 = Auf 1 setzen, 0 = auf 0 setzen, x = nach Wunsch setzen, S = so setzen wie bereits gesetzt

Arbeitsblatt:

Stufen	Register	Hex	Bit	Kommentar
1. Betriebsarten				
	WR9	_C_ _0_	1100 0000	Software Reset
	WR0	_0_ ___	0000 00__	_____
	WR4	___ ___	____ ____	_____
	WR1	___ ___	0__0 0_00	_____
	WR2	___ ___	____ ____	_____
	WR3	___ ___	____ ___0	_____
	WR5	___ ___	____ 0___	_____
	WR6	___ ___	____ ____	_____
	WR7	___ ___	____ ____	_____
	WR9	___ ___	000_ 0___	_____
	WR10	___ ___	____ ____	_____
	WR11	___ ___	____ ____	_____
	WR12	___ ___	____ ____	_____
	WR13	___ ___	____ ____	_____
	WR14	___ ___	____ ___0	_____
	WR14	___ ___	____ ___0	_____
2. Enables				
	WR3	___ ___	____ ___1	_____
	WR5	___ ___	____ 1___	_____
	WR0	_8_ _0_	1000 0000	Reset TxCRC
	WR14	___ ___	000_ ___1	_____
	WR1	___ ___	____ ____	_____
3. Interrupt				
	WR15	___ ___	____ ____	_____
	WR0	_1_ _0_	0001 0000	Reset Ext/Status
	WR0	_1_ _0_	0001 0000	Reset Ext/Status
	WR1	___ ___	____ ____	_____
	WR9	___ ___	000_ ____	_____

Initialisierungswerte nach Reset:

Register	**Hardware-Reset**	**Channel-Reset**
WR0	0000 0010	0000 0000
WR1	00_0 0_00	00_0 0_00
WR2	____ ____	____ ____
WR3	____ ___0	____ ___0
WR4	____ _1__	____ _1__
WR5	0__0 000_	0__0 0000
WR6	____ ____	____ ____
WR7	____ ____	____ ____
WR9	1100 00__	__0_ ____
WR10	0000 0000	0__0 0000
WR11	0000 1000	____ ____
WR12	____ ____	____ ____
WR13	____ ____	____ ____
WR14	__10 0000	__10 00__
WR15	1111 1000	1111 1000
RR0	01__ _100	01__ 100
RR1	0000 0111	0000 0111
RR3	0000 0000	0000 0000
RR10	0000 0000	0000 0000

3.5.2. Konfigurationsmöglichkeiten der seriellen Schnittstellen

Für jede der mit einem S-Link ausgerüsteten Schnittstellen A, C, D, E und F läßt sich per Software eine von 8 Betriebsarten einstellen (Mode 0, 1, 2, 3, 5 und 7 sind belegt, Mode 4 und 6 sind reserviert). Sie passen die Schnittstelle an den jeweils aufgesteckten S-Link-Adapter an. Bei einigen S-Links kann damit auch die Funktionalität verändert werden.

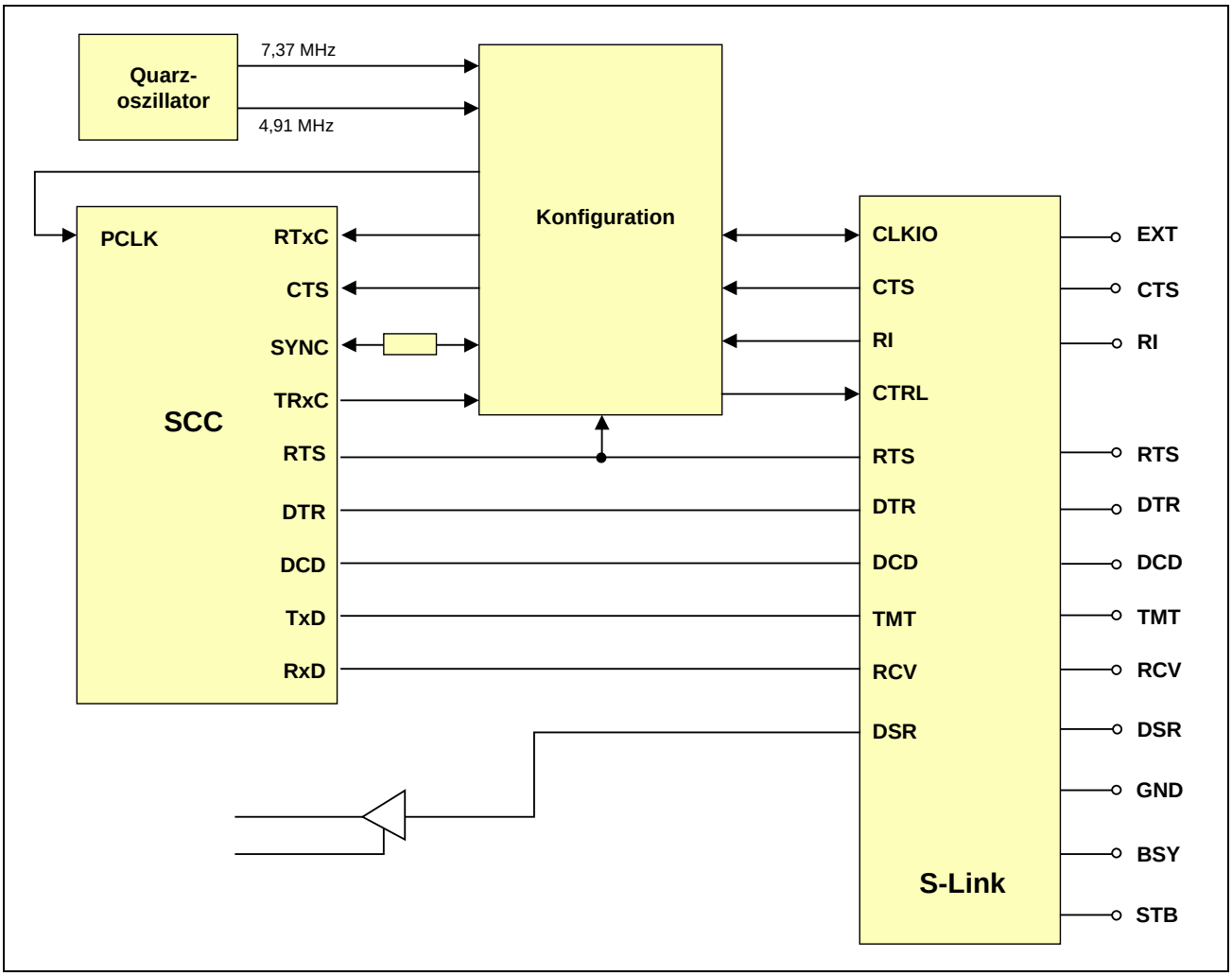


Abb. 3-2: *Prinzipschaltbild der Konfigurationsmöglichkeiten für die Schnittstellen A, C, D, E und F. PCLK eines SCC wird von beiden Kanälen des SCC genutzt. Die Frequenz ist je SCC programmierbar. Die DSR-Leitung des S-Link-Adapters kann keinen Interrupt auslösen, der Status dieser Leitung kann aber gelesen werden (Logik hier nicht gezeigt). Die Leitungen STB und BSY sind reserviert und im folgenden nicht beschrieben.*

Die EXT-Leitung (als Eingang) kann dazu verwendet werden, einen externen Takt an den Pin RTxC des SCC zu legen. Intern im SCC kann er als Empfangstakt, als Sendetakt, als Eingangstakt für die DPLL und/oder als Eingangstakt für den Baudratengenerator konfiguriert werden. Wenn er als Empfangs- und/oder Sendetakt verwendet wird, kann bei asynchronem Betrieb so die höchstmögliche Baudrate eingestellt werden, z.B. 460,8 KBaud mit 7,3728 MHz bzw. 1,152 MBaud mit 18,432 MHz bei Clock Mode x16.

Wird die EXT-Leitung als Ausgang konfiguriert, kann der Ausgangstakt von Pin TRxC des SCC nach außen geliefert werden.

Für jeden SCC steht ein programmierbarer Quarzoszillator zur Verfügung. Die Quarzoszillatoren sind per Software zwischen 4,9152 MHz und 7,3728 MHz umschaltbar.

3.5.3. Anschlussstecker St1

Die Karte wird über einen 62-poligen D-Sub.-Stecker St1 und ein entsprechendes Kabel mit der Außenwelt verbunden. Die Pinbelegung beim jeweiligen Kanal ist abhängig vom aufgesteckten S-Link-Adapter. Sie ist dort angegeben.

In den folgenden Tabellen ist meist auch die Belegung eines fertig konfektionierten Kabels angegeben. Dieses Kabel muss separat bestellt werden (Best.-Nr. K2-6259 oder K3-6260). Dabei ist an einem Ende des 1,5 bzw. 3 m langen, 60-poligen Kabels ein 62-poliger Stecker angebracht, der in St1 der Karte gesteckt wird. Am anderen Ende ist für jede der 5 Schnittstellen ein 9-poliger D-Submin.-Stecker angebracht (Belegung z.B. mit RS-232 S-Links entsprechend IBM-AT).

Die nachfolgende Beschreibung bezieht sich jeweils auf einen der 5 identisch aufgebauten Kanäle A, C, D, E und F. Bei allen Namen von Signalen und Pins ist der Zusatz /A, /C, /D, /E bzw. /F der Übersichtlichkeit halber weggelassen.

3.5.4. Übersicht der S-Link Adapter

Tab. 3-2: *Übersicht S-Link-Adapter*

S-Link-Adapter	Typ	Physikalische Schnittstelle	Kurzbeschreibung
SL-232S	1	RS-232 bis 220 KBaud	Modem-Steuerleitungen: TMT, RCV, RTS, CTS, DTR, DSR, RI, DCD RI als Clock-Eingang
SL-232A/o	4	RS-232 bis 220 KBaud	Modem-Steuerleitungen: TMT, RCV, RTS, CTS, DTR, DSR, RI, DCD Zusätzliche Funktionen: RS-232-Leitung EXT[1] als Clock-Ausgang RI als Clock-Eingang
SL-232A/i	3	RS-232 bis 220 KBaud	Modem-Steuerleitungen: TMT, RCV, RTS, CTS, DTR, DSR, RI, DCD Zusätzliche Funktionen: RS-232-Leitung EXT[1] als Clock-Eingang 1 RI als Clock-Eingang 2
SL-232i	16	RS-232 isol. bis 220 KBaud	Modem-Steuerleitungen, galvanisch getrennt: TMT, RCV, RTS, CTS Zusätzliche Funktionen: CTS als Clock-Eingang RTS als Clock-Ausgang
SL-422S	8	RS-422 bis 10 MBaud	Modem-Steuerleitungen: TMT, RCV, RTS, CTS Zusätzliche Funktionen: CTS als Clock-Eingang RTS als Clock-Ausgang

[1] EXT ist kein RS-232-Standardsignal. Es ist auch bei den üblichen 9-poligen oder 25-poligen D-Submin.-Steckern bzw. Buchsen nicht vorhanden. Die 9-polige Schnittstelle wurde hier um einen Pin erweitert, um bei RS-232 alle Modem-Steuersignale und zusätzlich einen Taktein- bzw. -ausgang zu ermöglichen. Bei jenen S-Links, die diesen Pin verwenden, kann er, wenn das entsprechende Signal nicht benötigt wird, frei bleiben (= nicht angeschlossen). Bei den Kabeln K2-6259 und K3-6360 ist dieser Pin nicht verfügbar.

S-Link-Adapter	**Typ**	**Physikalische Schnittstelle**	**Kurzbeschreibung**
SL-422i	24	RS-422 isol. bis 10 MBaud	Modem-Steuerleitungen, galvanisch getrennt: TMT, RCV, RTS, CTS Zusätzliche Funktionen: CTS als Clock-Eingang RTS als Clock-Ausgang
SL-485S	9	RS-485 bis 12 MBaud	Umschaltung von Senden auf Empfangen per Software oder automatisch (z.B. für SDLC/HDLC)
SL-485i	32	RS-485 isol. bis 12 MBaud	Auch für PROFIBUS bis 12 MBaud geeignet, zusätzlicher TTL-Ausgang zeigt Senden/Empfangen an, z.B. für ext. Transceiver.
SL-20MA	40	20 mA isol. bis 120 KBaud	20 mA Current Loop, passiv oder aktiv konfigurierbar (wenn passiv, dann galvanisch getrennt)
SL-LWL/P	48	Lichtwellen-leiter (Plastik)	Toshiba Plastik-Lichtwellenleiter APF (TODX297)
SL-LWL/G	49	Lichtwellen-leiter (Glas)	Toshiba Glas-Lichtwellenleiter PCS (TODX296)
SL-CANi	64	CAN-Interface, isoliert	Full-CAN, bis 1 MBit/s Übertragungsrate, 11- und 29-Bit Identifier
SL-DPSi	72	RS-485	PROFIBUS-Slave bis 12 MBaud
SL-IBSi	80	RS-485	InterBus S
SL-TEST	88	diverse	Test S-Link zum Testen aller anderen S-Links außer Typ 48 und 49

3.5.5. S-Link SL-232S: RS-232 mit allen Modem-Steuerleitungen

Für einen mit diesem S-Link-Adapter ausgerüsteten Kanal kann Mode 1, 3 oder 5 eingestellt werden. Die damit möglichen Funktionen der RS-232-Anschlüsse CTS, RI und RTS und der SCC-Pins RTxC und TRxC sind in folgender Tabelle angegeben.

Tab. 3-3: *Zusammenfassung der möglichen Modes mit SL-232S. Die mit * gekennzeichneten Signale RTS* und CTS* beziehen sich auf die RS-232-Anschluß-pins (siehe auch Abb. 3-3 bis Abb. 3-5).*

Mode	Funktion von CTS*	Funktion von RTS*	Funktion von RTxC	TRxC Ein-/Ausgang	Funktion von TRxC
1	CTS	RTS	= PCLK	Eingang	keine
3	CLK_{in} an RTxC	RTS	CLK_{in} von CTS*	Eingang	keine
5	CTS	CLK_{out} von TRxC	= PCLK	Ausgang	CLK_{out} an RTS*

Tab. 3-4: *Pinbelegung von St1 und Funktion der RS-232-Anschlüsse mit S-Link SL-232S in Mode 1, 3 und 5.*

RS-232 Signal	RS-232 Ein-/Ausgang	Funktion in Mode 1	3	5	Pin (St1): Kanal A	C	D	E	F	9-pol. D-Sub.
DCD	Eingang	DCD	DCD	DCD	4	8	12	16	20	1
DSR	Eingang	DSR	DSR	DSR	25	29	33	37	41	6
RCV	Eingang	RxD	RxD	RxD	45	49	53	57	61	2
RTS	Ausgang	RTS	RTS	CLK_{out}	24	28	32	36	40	7
TMT	Ausgang	TxD	TxD	TxD	3	7	11	15	19	3
CTS	Eingang	CTS	CLK_{in}	CTS	23	27	31	35	39	8
DTR	Ausgang	DTR	DTR	DTR	44	48	52	56	60	4
RI	Eingang	RI	RI	RI	2	6	10	14	18	9
GND	Ausgang	GND	GND	GND	43	47	51	55	59	5
EXT	-	-	-	-	22	26	30	34	38	–

- = Signal bzw. Pin ist nicht angeschlossen

SL-232S in Mode 1

Der Takt des Quarzoszillators liegt auch direkt am RTxC Pin eines Kanals des SCC.

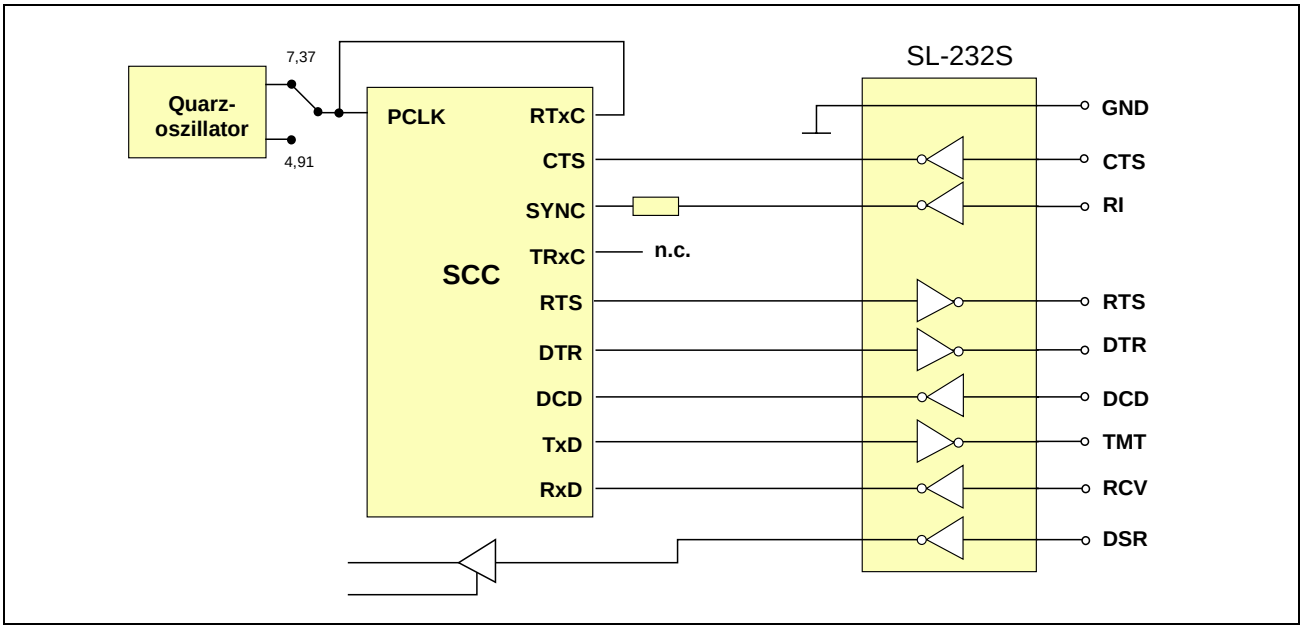


Abb. 3-3: *S-Link-Adapter SL-232S in Mode 1.*

SL-232S in Mode 3

Der RS-232-Anschluß CTS dient als Clock-Eingang und liegt am RTxC Pin des SCC des zugehörigen Kanals.

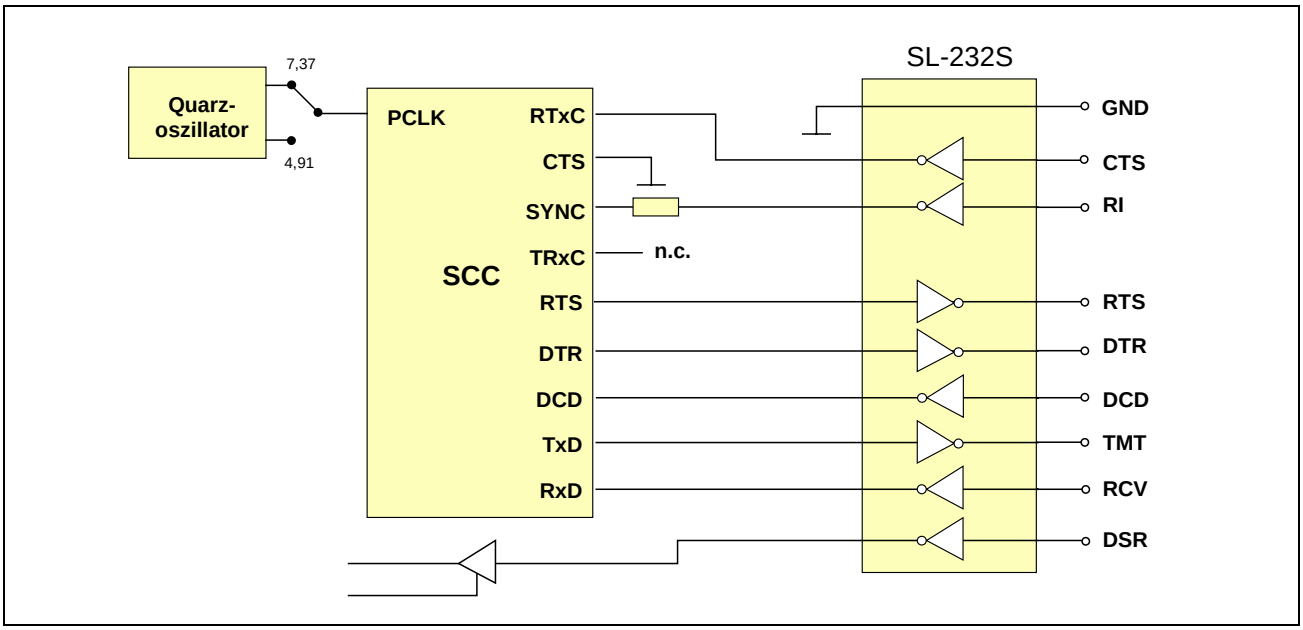


Abb. 3-4: *S-Link-Adapter SL-232S in Mode 3.*

SL-232S in Mode 5

Der Takt des Quarzoszillators liegt zusätzlich am RTxC Pin eines Kanals des SCC. Der RS-232-Anschluß RTS dient als Clock-Ausgang und liefert das Signal am TRxC Pin. Durch Programmierung des SCC wird festgelegt, ob das der Empfangstakt, der Sendetakt, der Ausgang der DPLL oder der Ausgang des Baudratengenerators dieses Kanals ist.

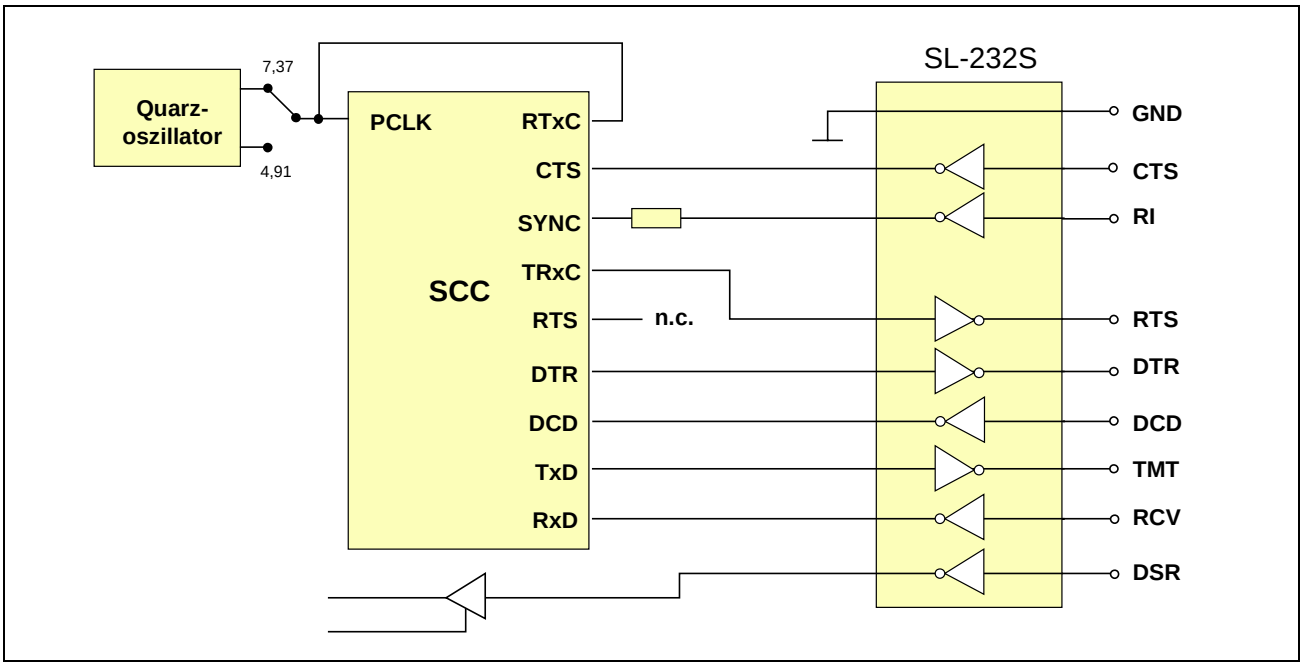


Abb. 3-5: *S-Link-Adapter SL-232S in Mode 5.*

3.5.6. S-Link SL-232A/i: RS-232 mit zusätzlichem CLK-Input

Für einen mit diesem S-Link-Adapter ausgerüsteten Kanal muss Mode 0 eingestellt und im SCC der zugehörige TRxC Pin als Eingang konfiguriert werden.

Über den RS-232-Anschluß EXT (Eingang) kann ein Takt an Pin RTxC des SCC gelegt werden. Der RS-232-Anschluß RI kann entweder als Modem-Steuerleitung RI dienen (liegt am zugehörigen SYNC-Pin des SCC und ist damit interruptfähig), oder es kann darüber ein weiterer Takt von außen an Pin TRxC des SCC gelegt werden.

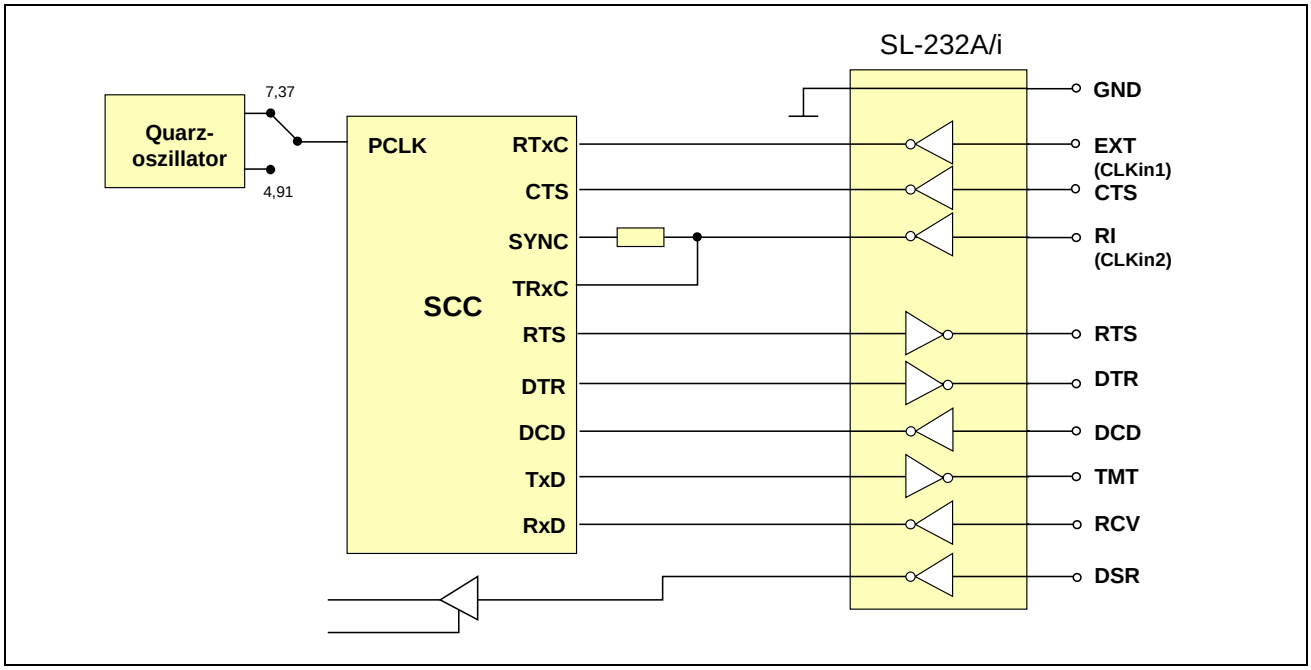


Abb. 3-6: *S-Link Adapter SL-232A/i in Mode 0.*

Tab. 3-5: *Pinbelegung von St1 mit S-Link SL-232A/i*

RS-232 Signal	RS-232 Ein-/Ausgang	Funktion (Mode 0)	Pin (St1): Kanal A	C	D	E	F	9-pol. D-Sub.
DCD	Eingang	DCD	4	8	12	16	20	1
DSR	Eingang	DSR	25	29	33	37	41	6
RxD	Eingang	RxD	45	49	53	57	61	2
RTS	Ausgang	RTS	24	28	32	36	40	7
TxD	Ausgang	TxD	3	7	11	15	19	3
CTS	Eingang	CTS	23	27	31	35	39	8
DTR	Ausgang	DTR	44	48	52	56	60	4
RI	Eingang	RI oder $CLK_{in}2$ (an SYNC u. TRxC)	2	6	10	14	18	9
GND	Ausgang	Ground	43	47	51	55	59	5
EXT	Eingang	$CLK_{in}1$ (an RTxC)	22	26	30	34	38	–

3.5.7. S-Link SL-232A/o: RS-232 mit zusätzlichem CLK-Output

Für einen mit diesem S-Link-Adapter ausgerüsteten Kanal muss Mode 5 eingestellt und im SCC der zugehörige TRxC Pin als Ausgang konfiguriert werden.

Über den RS-232-Anschluß EXT (Ausgang) kann das Signal des TRxC Pin dieses Kanals des SCC nach außen geliefert werden. Der Takt des Quarzoszillators liegt auch direkt am RTxC Pin dieses Kanals des SCC.

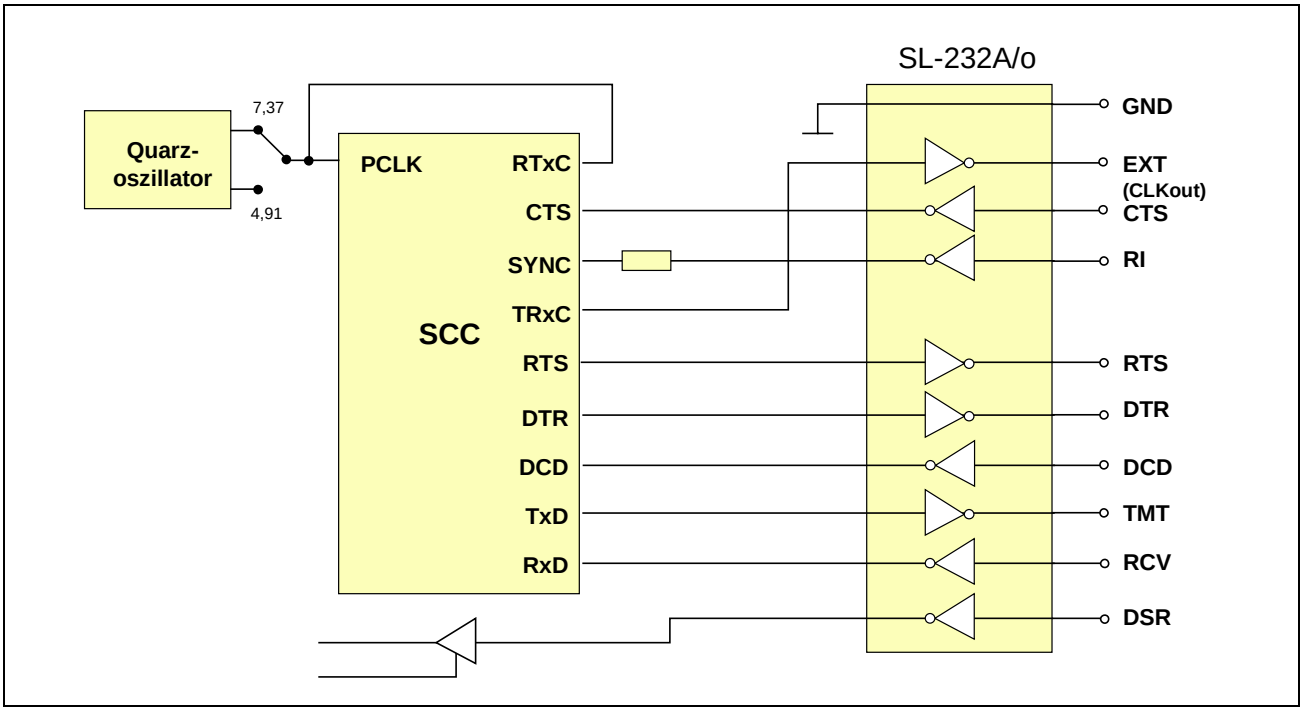


Abb. 3-7: *SL-232A/o in Mode 5.*

Tab. 3-6: *Pinbelegung von St1 mit S-Link SL-232A/o*

RS-232 Signal	RS-232 Ein-/Ausgang	Funktion (Mode 5)	Pin (St1): Kanal A	C	D	E	F	9-pol. D-Sub.
DCD	Eingang	DCD	4	8	12	16	20	1
DSR	Eingang	DSR	25	29	33	37	41	6
RxD	Eingang	RxD	45	49	53	57	61	2
RTS	Ausgang	RTS	24	28	32	36	40	7
TxD	Ausgang	TxD	3	7	11	15	19	3
CTS	Eingang	CTS	23	27	31	35	39	8
DTR	Ausgang	DTR	44	48	52	56	60	4
RI	Eingang	RI (an SYNC des SCC)	2	6	10	14	18	9
GND	Ausgang	Ground	43	47	51	55	59	5
EXT	Ausgang	CLK_{out} (von TRxC des SCC)	22	26	30	34	38	–

3.5.8. S-Link SL-232i: RS-232, galvanisch getrennt

Für einen mit diesem S-Link-Adapter ausgerüsteten Kanal kann Mode 1, 3 oder 5 eingestellt werden. Die damit möglichen Funktionen der RS-232-Anschlüsse CTS und RTS und der SCC-Pins RTxC und TRxC sind in folgender Tabelle angegeben.

Tab. 3-7: *Zusammenfassung der möglichen Modes mit SL-232i. Die mit * gekennzeichneten Signale RTS* und CTS* beziehen sich auf die RS-232-Anschlußpins.*

Mode	Funktion von CTS*	Funktion von RTS*	Funktion von RTxC	TRxC Ein-/Ausgang	Funktion von TRxC
1	CTS	RTS	= PCLK	Eingang	keine
3	CLK_{in} an RTxC	RTS	CLK_{in} von CTS*	Eingang	keine
5	CTS	CLK_{out} von TRxC	= PCLK	Ausgang	CLK_{out} an RTS*

Tab. 3-8: *Pinbelegung von St1 und Funktion der RS-232-Anschlüsse mit S-Link SL-232i in Mode 1, 3 und 5*

RS-232	RS-232	Funktion in Mode			Pin (St1): Kanal					9-pol.
Signal	Ein-/Ausgang	1	3	5	A	C	D	E	F	D-Sub.
-	-	-	-	-	4	8	12	16	20	1
-	-	-	-	-	25	29	33	37	41	6
RxD	Eingang	RxD	RxD	RxD	45	49	53	57	61	2
RTS	Ausgang	RTS	RTS	CLK_{out}	24	28	32	36	40	7
TxD	Ausgang	TxD	TxD	TxD	3	7	11	15	19	3
CTS	Eingang	CTS	CLK_{in}	CTS	23	27	31	35	39	8
-	-	-	-	-	44	48	52	56	60	4
-	-	-	-	-	2	6	10	14	18	9
GND	Ausgang	GND	GND	GND	43	47	51	55	59	5
-	-	-	-	-	22	26	30	34	38	–

- = Pin ist nicht angeschlossen

SL-232i in Mode 1

Der Takt des Quarzoszillators liegt auch direkt am RTxC Pin dieses Kanals des SCC.

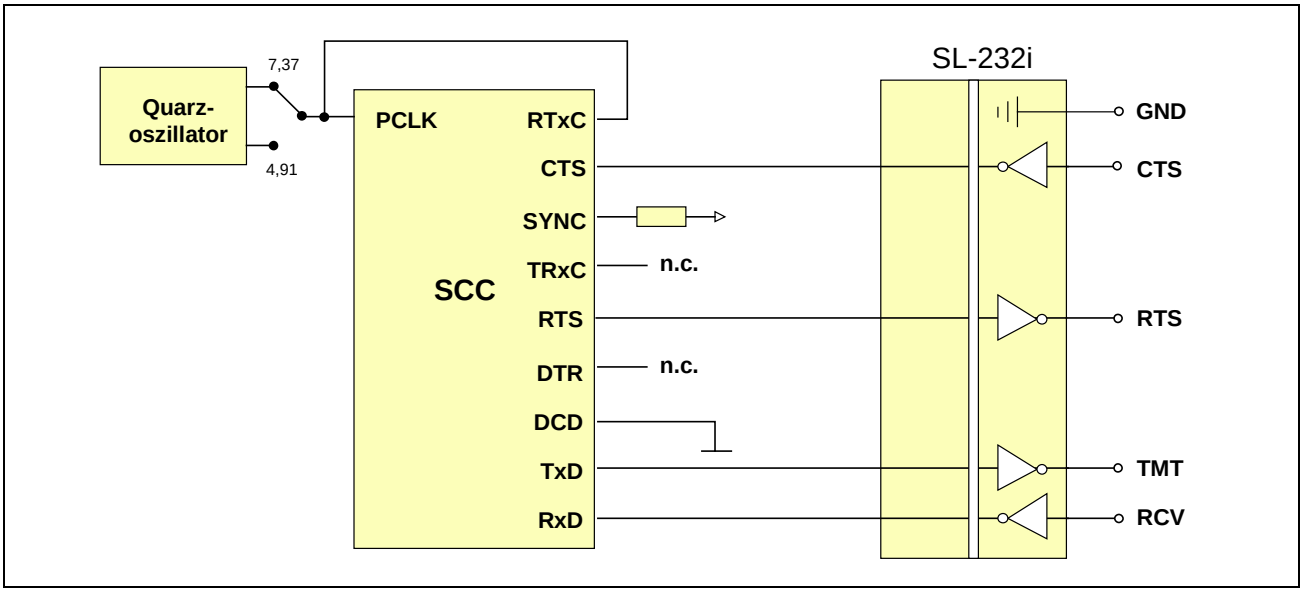


Abb. 3-8: *S-Link-Adapter SL-232i in Mode 1.*

SL-232i in Mode 3

Der RS-232-Anschluß CTS dient als Clock-Eingang und liegt am RTxC Pin des SCC des zugehörigen Kanals.

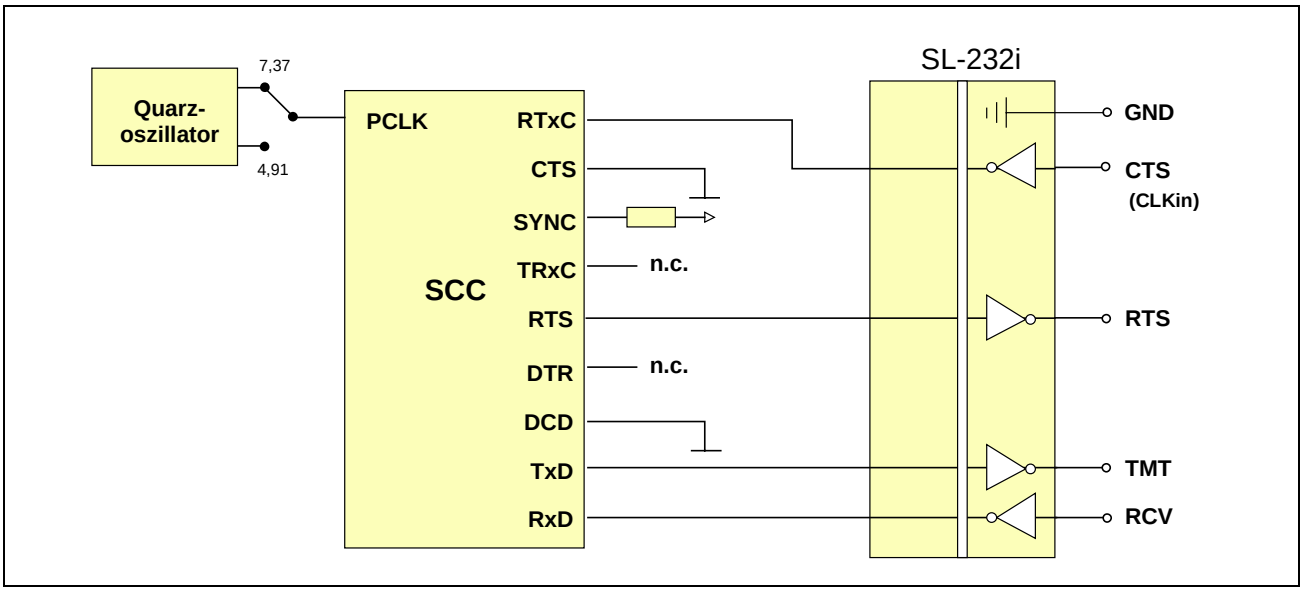


Abb. 3-9: *S-Link-Adapter SL-232i in Mode 3.*

SL-232i in Mode 5

Der Takt des Quarzoszillators liegt auch direkt am RTxC Pin dieses Kanals des SCC. Der RS-232-Anschluß RTS dient als Clock-Ausgang vom TRxC Pin des SCC des zugehörigen Kanals.

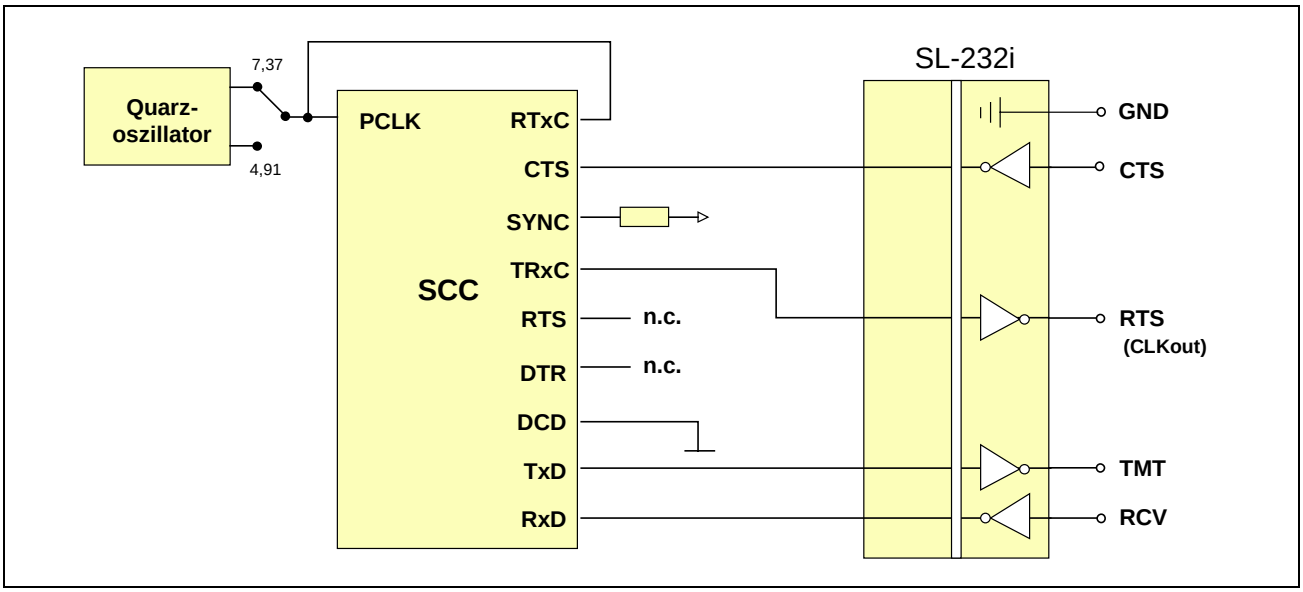


Abb. 3-10: *S-Link-Adapter SL-232i in Mode 5.*

3.5.9. S-Link SL-422S: RS-422-Schnittstelle

Verfügbar sind die Modem-Steuersignale RTS und CTS (Mode 1). Per Software kann die CTS-Leitung auch als Takteingang (Mode 3) oder die RTS-Leitung als Taktausgang (Mode 5) geschaltet werden. Die damit möglichen Funktionen der RS-422-Anschlüsse CTS und RTS und der SCC-Pins RTxC und TRxC sind in folgender Tabelle angegeben. Der DTR-Pin des zugehörigen Kanals muss im SCC auf log. 1 gesetzt werden. Damit wird der RS-422-Sendetreiber enabled. Beachten Sie bitte, dass bei RS-422-Verbindungen + mit + und - mit - der Gegenstation verbunden wird.

Tab. 3-9: *Zusammenfassung der möglichen Modes mit SL-422S und der Funktionen der RS-422-Leitungen (die mit * gekennzeichneten Signale CTS* und RTS* beziehen sich auf die RS-422-Anschlußpins).*

Mode	Funktion von CTS*	Funktion von RTS*	Funktion von RTxC	TRxC Ein-/Ausgang	Funktion von TRxC
1	CTS	RTS	= PCLK	Eingang	keine
3	CLK_{in} an RTxC	RTS	CLK_{in} von CTS*	Eingang	keine
5	CTS	CLK_{out} von TRxC	= PCLK	Ausgang	CLK_{out} an RTS*

Tab. 3-10: *Pinbelegung von St1 und Funktion der RS-422-Anschlüsse mit S-Link SL-422S in Mode 1, 3 und 5 (Angaben für 9-pol. D-Submin. Stecker gelten für SORCUS Kabel K2-6259 und K3-6260).*

RS-422 Signal	RS-422 Ein-/Ausgang	Funktion in Mode 1	3	5	Pin (St1): Kanal A	C	D	E	F	9-pol. D-Sub.
RTS-	Ausgang	RTS-	RTS-	CLK_{out}-	4	8	12	16	20	1
RTS+	Ausgang	RTS+	RTS+	CLK_{out}+	25	29	33	37	41	6
CTS-	Eingang	CTS-	CLK_{in}-	CTS-	45	49	53	57	61	2
CTS+	Eingang	CTS+	CLK_{in}+	CTS+	24	28	32	36	40	7
TMT-	Ausgang	TMT-	TMT-	TMT-	3	7	11	15	19	3
TMT+	Ausgang	TMT+	TMT+	TMT+	23	27	31	35	39	8
RCV-	Eingang	RCV-	RCV-	RCV-	44	48	52	56	60	4
RCV+	Eingang	RCV+	RCV+	RCV+	2	6	10	14	18	9
GND	Ausgang	GND	GND	GND	43	47	51	55	59	5
-	-	-	-	-	22	26	30	34	38	–

- = Pin ist nicht angeschlossen

Abschlusswiderstände bei S-Link SL-422S:

Das S-Link enthält Abschlusswiderstände für alle 4 Modemsteuerleitungen RCV, TMT, CTS und RTS. Über Schalter S1 bis S10 kann gewählt werden, ob mit oder ohne Busabschluss von 120 Ω (Termination) gearbeitet wird. Bei eingeschalteter Termination kann gewählt werden, ob bei offenem Eingang bzw. offenem Bus am Eingang des SCC eine log. 0 oder log. 1 liegt.

Funktion/Schalter	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**
Sender ohne Termination	o	-	-	-	-	-	-	-	-	-
Sender mit Termination	v	-	-	-	-	-	-	-	-	-
Empfänger ohne Termination	-	o	o	o	o	-	-	-	-	-
Empfänger mit Term., log. 0 am SCC-Pin	-	o	o	v	v	-	-	-	-	-
Empfänger mit Term., log. 1 am SCC-Pin	-	v	v	o	o	-	-	-	-	-
RTS ohne Termination	-	-	-	-	-	o	-	-	-	-
RTS mit Termination	-	-	-	-	-	v	-	-	-	-
CTS ohne Termination	-	-	-	-	-	-	o	o	o	o
CTS mit Termination, log. 0 am SCC-Pin	-	-	-	-	-	-	o	o	v	v
CTS mit Termination, log. 1 am SCC-Pin	-	-	-	-	-	-	v	v	o	o

- = keine Bedeutung, o = offen, v = verbunden

SL-422S für RS-422 in Mode 1

Der Takt des Quarzoszillators liegt auch direkt am RTxC Pin dieses Kanals des SCC.

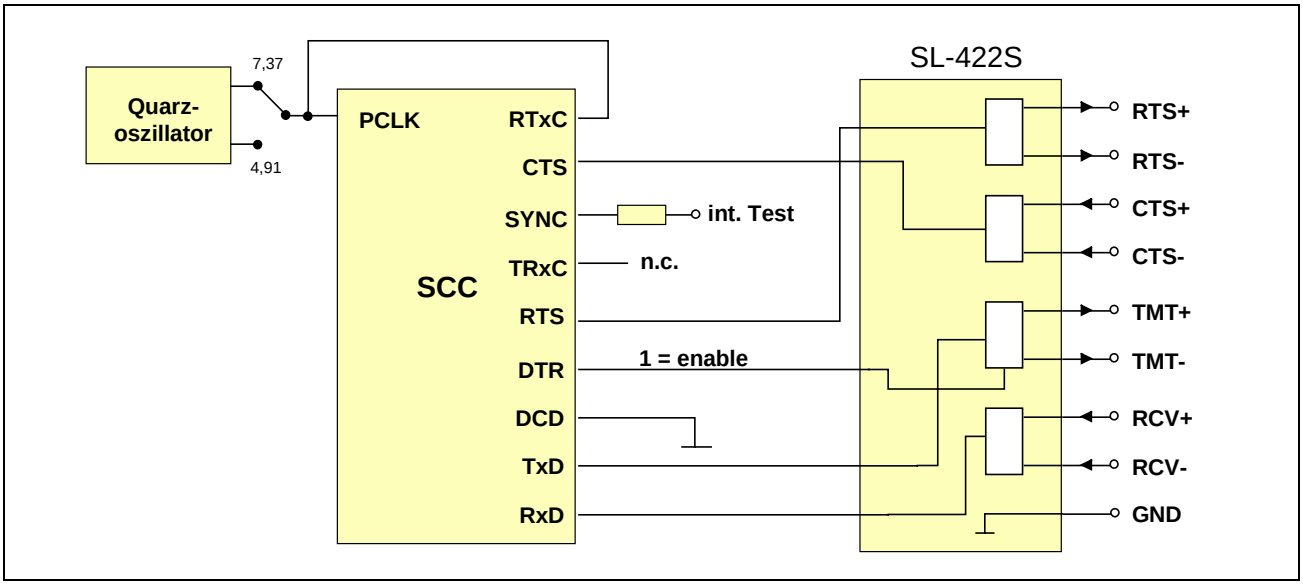


Abb. 3-11: *S-Link-Adapter SL-422S für RS-422 in Mode 1. Der Sendetreiber wird über die DTR-Leitung enabled.*

SL-422S für RS-422 in Mode 3

Der RS-422-Anschluß CTS dient als Clock-Eingang und liegt am RTxC Pin des SCC des zugehörigen Kanals.

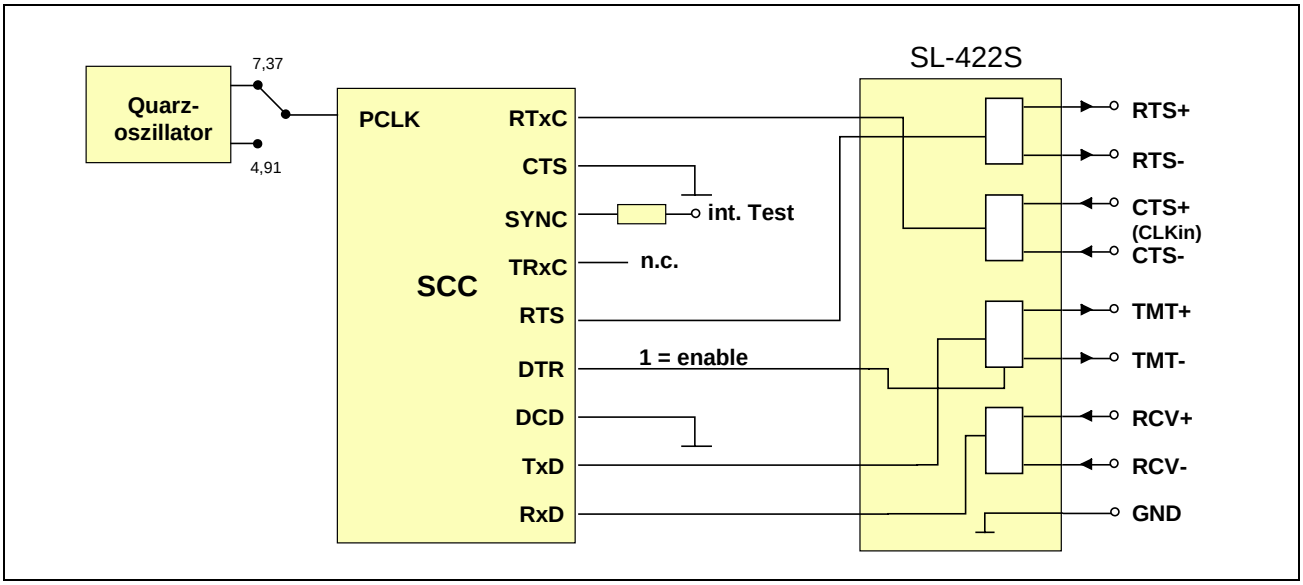


Abb. 3-12: *S-Link-Adapter SL-422S für RS-422 in Mode 3. Der Sendetreiber wird über die DTR-Leitung enabled.*

SL-422S für RS-422 in Mode 5

Der Takt des Quarzoszillators liegt auch direkt am RTxC Pin dieses Kanals des SCC. Der RS-422-Anschluß RTS dient als Clock-Ausgang vom TRxC Pin des SCC des zugehörigen Kanals.

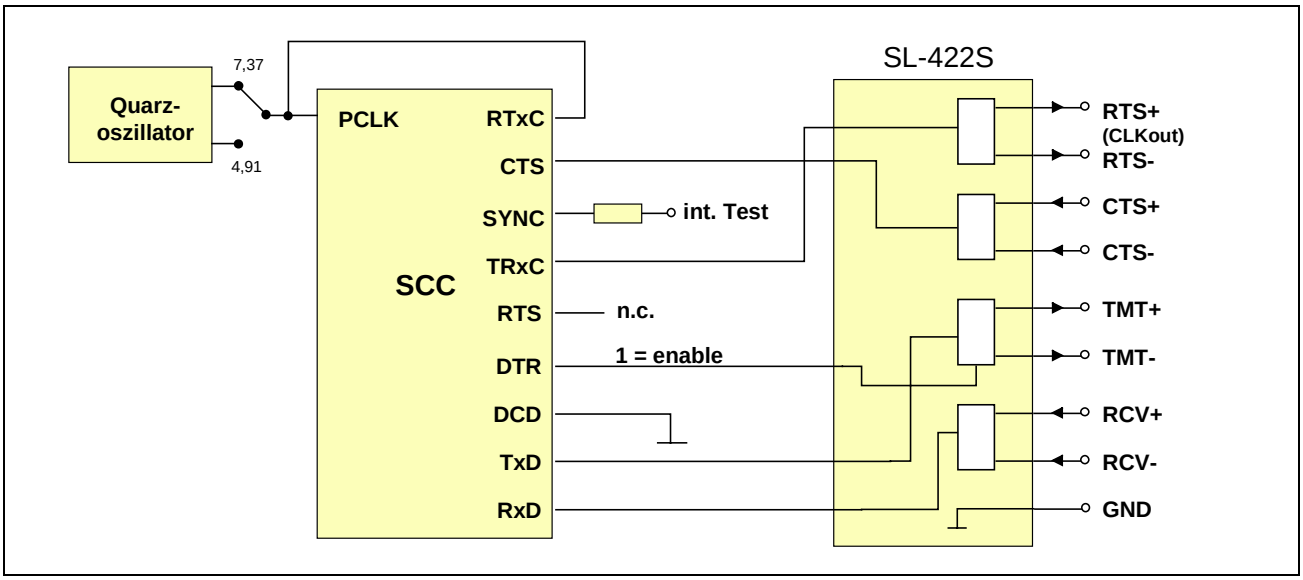


Abb. 3-13: *S-Link-Adapter SL-422S für RS-422 in Mode 5. Der Sendetreiber wird über die DTR-Leitung enabled.*

3.5.10. S-Link SL-422i: RS-422, galvanisch getrennt

Für einen mit diesem S-Link-Adapter ausgerüsteten Kanal kann Mode 1, 3 oder 5 eingestellt werden. Die damit möglichen Funktionen der RS-422-Anschlüsse CTS und RTS und der SCC-Pins RTxC und TRxC sind in folgender Tabelle angegeben. Beachten Sie bitte, dass bei RS-422-Verbindungen + mit + und - mit - der Gegenstation verbunden wird.

Tab. 3-11: *Zusammenfassung der möglichen Modes mit SL-422i. Die mit * gekennzeichneten Signale RTS* und CTS* beziehen sich auf die RS-422-Anschlußpins (siehe auch Abb. 3-14).*

Mode	Funktion von CTS*	Funktion von RTS*	Funktion von RTxC	TRxC Ein-/Ausgang	Funktion von TRxC
1	CTS	RTS	= PCLK	Eingang	keine
3	CLK_{in} an RTxC	RTS	CLK_{in} von CTS*	Eingang	keine
5	CTS	CLK_{out} von TRxC	= PCLK	Ausgang	CLK_{out} an RTS*

Tab. 3-12: *Pinbelegung von St1 und Funktion der RS-422-Anschlüsse mit S-Link SL-422i in Mode 1, 3 und 5*

RS-422 Signal	RS-422 Ein-/Ausgang	Funktion in Mode 1	3	5	Pin (St1): Kanal A	C	D	E	F	9-pol. D-Sub.
RTS-	Ausgang	RTS-	RTS-	CLK_{out}-	4	8	12	16	20	1
RTS+	Ausgang	RTS+	RTS+	CLK_{out}+	25	29	33	37	41	6
CTS-	Eingang	CTS-	CLK_{in}-	CTS-	45	49	53	57	61	2
CTS+	Eingang	CTS+	CLK_{in}+	CTS+	24	28	32	36	40	7
TMT-	Ausgang	TMT-	TMT-	TMT-	3	7	11	15	19	3
TMT+	Ausgang	TMT+	TMT+	TMT+	23	27	31	35	39	8
RCV-	Eingang	RCV-	RCV-	RCV-	44	48	52	56	60	4
RCV+	Eingang	RCV+	RCV+	RCV+	2	6	10	14	18	9
GND	Ausgang	GND	GND	GND	43	47	51	55	59	5
-	-	-	-	-	22	26	30	34	38	–

- = Pin ist nicht angeschlossen

Abschlusswiderstände:

Über Schalter S1 bis S10 kann gewählt werden, ob mit oder ohne Busabschluss von 120 Ω (Termination) gearbeitet wird. Bei eingeschalteter Termination kann gewählt werden, ob bei offenem Eingang bzw. offenem Bus am SCC-Pin eine log. 0 oder log. 1 liegt.

Funktion/Schalter	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**
Sender ohne Termination	o	-	-	-	-	-	-	-	-	-
Sender mit Termination	v	-	-	-	-	-	-	-	-	-
Empfänger ohne Termination	-	o	o	o	o	-	-	-	-	-
Empfänger mit Term., log. 0 am SCC-Pin	-	o	o	v	v	-	-	-	-	-
Empfänger mit Term., log. 1 am SCC-Pin	-	v	v	o	o	-	-	-	-	-
RTS ohne Termination	-	-	-	-	-	o	-	-	-	-
RTS mit Termination	-	-	-	-	-	v	-	-	-	-
CTS ohne Termination	-	-	-	-	-	-	o	o	o	o
CTS mit Termination, log. 0 am SCC-Pin	-	-	-	-	-	-	o	o	v	v
CTS mit Termination, log. 1 am SCC-Pin	-	-	-	-	-	-	v	v	o	o

- = keine Bedeutung, o = offen, v = verbunden

SL-422i in Mode 1

Der Takt des Quarzoszillators liegt auch direkt am RTxC Pin dieses Kanals des SCC.

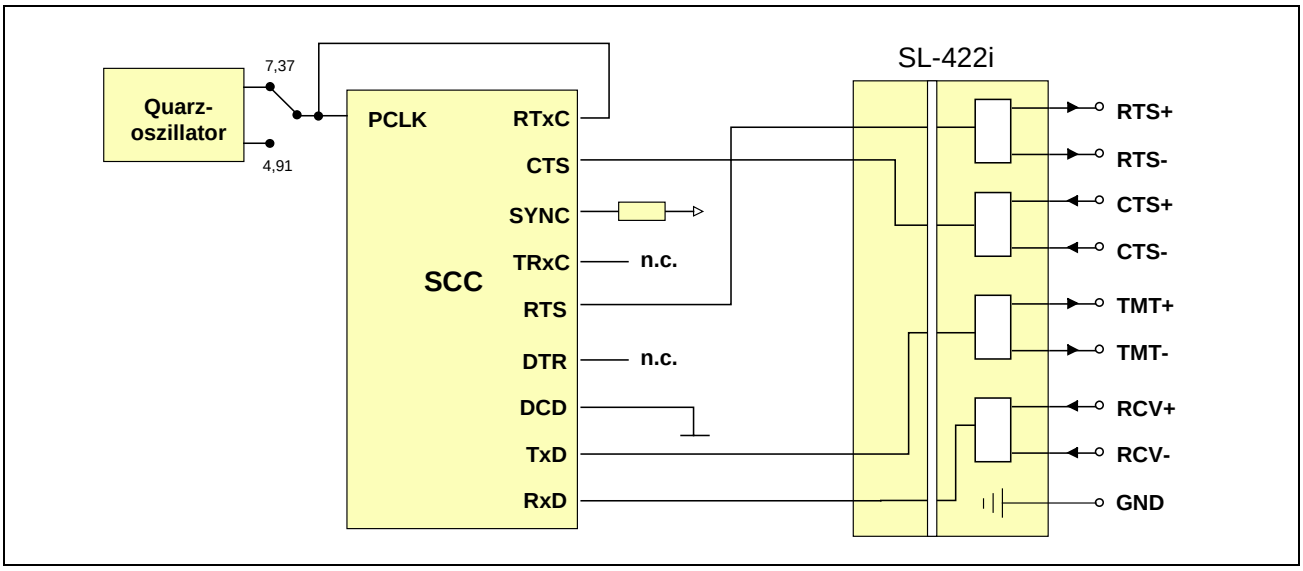


Abb. 3-14: *S-Link-Adapter SL-422i in Mode 1.*

SL-422i in Mode 3

Der RS-422-Anschluß CTS dient als Clock-Eingang und liegt am RTxC Pin des SCC des zugehörigen Kanals.

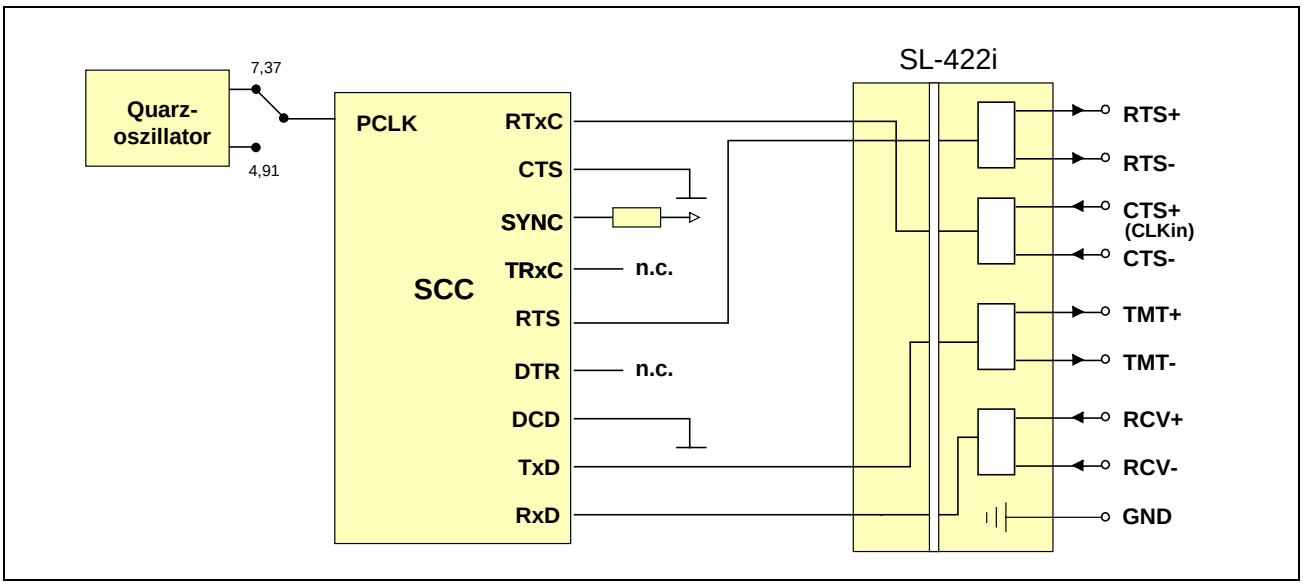


Abb. 3-15: *S-Link-Adapter SL-422i in Mode 3.*

SL-422i in Mode 5

Der Takt des Quarzoszillators liegt auch direkt am RTxC Pin dieses Kanals des SCC. Der RS-422-Anschluß RTS dient als Clock-Ausgang vom TRxC Pin des SCC des zugehörigen Kanals.

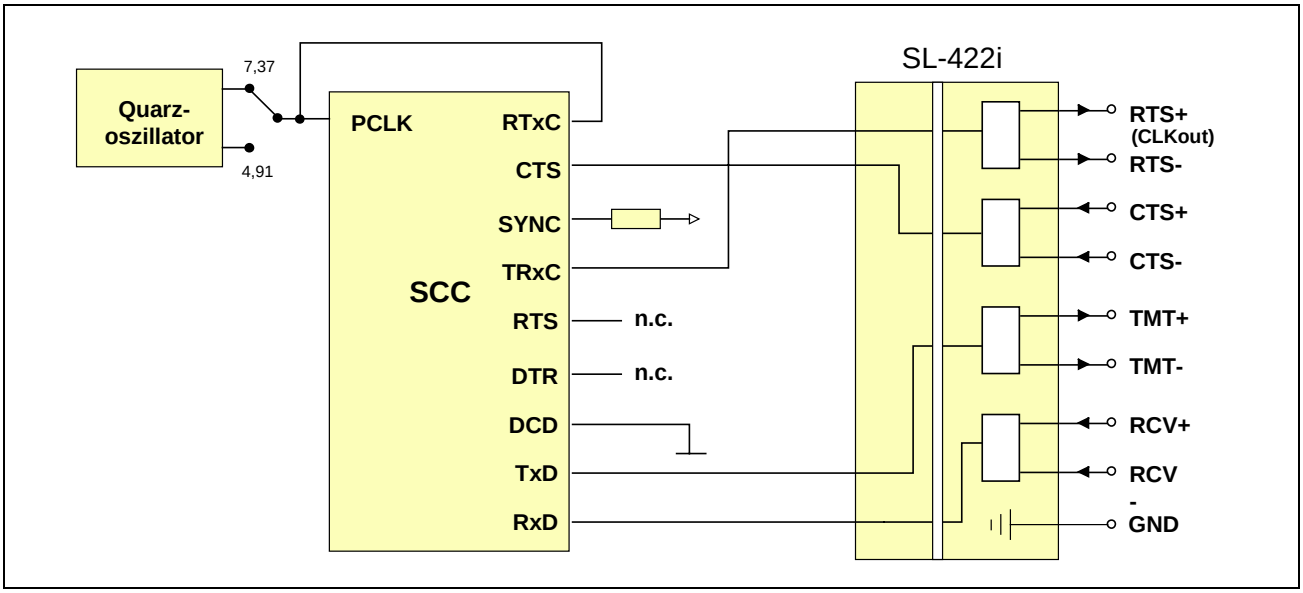


Abb. 3-16: *S-Link-Adapter SL-422i in Mode 5.*

3.5.11. S-Link SL-485S: RS-485-Schnittstelle

Die Umschaltung zwischen Senden und Empfangen geschieht per Software über den Pin RTS des SCC. Nach Reset ist der RS-485-Sendetreiber abgeschaltet. Der RS-485-Empfänger ist immer aktiviert, so dass auch ein von der Schnittstelle selbst gesendetes Signal wieder am Empfangseingang des SCC erscheint. Wenn das nicht gewünscht ist, muss der Empfänger im SCC disabled werden (Bit 0 in WR3 des SCC).

Zusätzlich verfügbar sind die Modem-Steuersignale RTS und CTS (Mode 1). Nach Reset ist der RTS-Pin = 1. Wenn in Mode 1, 3 oder 5 umgeschaltet wird, sollte also vorher RTS = 0 gesetzt werden (Bit 1 in WR5 des SCC = 1). Beachten Sie bitte, dass bei RS-485-Verbindungen + mit + und - mit - der Gegenstation verbunden wird.

Tab. 3-13: *Zusammenfassung der möglichen Modes mit SL-485S und der Funktionen der RS-485-Leitungen. RTS* und CTS* sind die RS-485-Leitungen, RTS und CTS die Signale (Pins) des SCC. Mode 0 ist mit angegeben, weil dieser Mode nach einem Hardware-Reset der Multi-COM Karte, z.B. auch nach Power-On, eingeschaltet ist.*

Mode	Funktion der RS-485-Leitungen CTS*	RTS*	Funktion von RTxC	TRxC Ein-/Ausgang	Funktion von TRxC	RS-485 Funktion
0	CTS	= 0	keine	Eingang	keine	Empfangen
1	CTS	RTS = 0	= PCLK	Eingang	keine	Empfangen
1	CTS	RTS = 1	= PCLK	Eingang	keine	Senden
3	CLK_{in}	RTS = 0	von CTS*	Eingang	keine	Empfangen
	CLK_{in}	RTS = 1	von CTS*	Eingang	keine	Senden
5	CTS	CLK_{out}	= PCLK	Ausgang	an RTS*	Empfangen (RTS = 0) Senden (RTS = 1)
7	keine	RTS = 0	= PCLK	Ausgang	keine	Senden
	keine	RTS = 1	= PCLK	Ausgang	keine	Empfangen

In Mode 7 liegt /RTS an CTS des SCC.

Abschlusswiderstände:

Das S-Link enthält Abschlusswiderstände für alle 4 Modemsteuerleitungen RCV, TMT, CTS und RTS. Über Schalter S1 bis S10 kann gewählt werden, ob mit oder ohne Busabschluss von 120 Ω (Termination) gearbeitet wird. Bei eingeschalteter Termination kann gewählt werden, ob bei offenem Eingang bzw. offenem Bus am Eingang vom SCC eine log. 0 oder log. 1 liegt.

Funktion/Schalter	**S1**	**S2**	**S3**	**S4**	**S5**	**S6**	**S7**	**S8**	**S9**	**S10**
Sender ohne Termination	o	-	-	-	-	-	-	-	-	-
Sender mit Termination	v	-	-	-	-	-	-	-	-	-
Empfänger ohne Termination	-	o	o	o	o	-	-	-	-	-
Empfänger mit Term., log. 0 am SCC-Pin	-	o	o	v	v	-	-	-	-	-
Empfänger mit Term., log. 1 am SCC-Pin	-	v	v	o	o	-	-	-	-	-
RTS ohne Termination	-	-	-	-	-	o	-	-	-	-
RTS mit Termination	-	-	-	-	-	v	-	-	-	-
CTS ohne Termination	-	-	-	-	-	-	o	o	o	o
CTS mit Termination, log. 0 am SCC-Pin	-	-	-	-	-	-	o	o	v	v
CTS mit Termination, log. 1 am SCC-Pin	-	-	-	-	-	-	v	v	o	o

- = keine Bedeutung, o = offen, v = verbunden

Tab. 3-14: *Pinbelegung von St1 mit S-Link SL-485S (Angaben für D-Submin. Stecker gelten für SORCUS Kabel K2-6259 bzw. K3-6260).*

RS-485			**Pin (St1): Kanal**			**9-pol.**
Signal	**A**	**C**	**D**	**E**	**F**	**D-Sub.**
RTS-	4	8	12	16	20	1
RTS+	25	29	33	37	41	6
CTS-	45	49	53	57	61	2
CTS+	24	28	32	36	40	7
TMT-	3	7	11	15	19	3
TMT+	23	27	31	35	39	8
RCV-	44	48	52	56	60	4
RCV+	2	6	10	14	18	9
GND	43	47	51	55	59	5
-	22	26	30	34	38	–

- = Pin ist nicht angeschlossen

SL-485S in Mode 1

Der Takt des Quarzoszillators liegt auch direkt am RTxC Pin dieses Kanals des SCC.

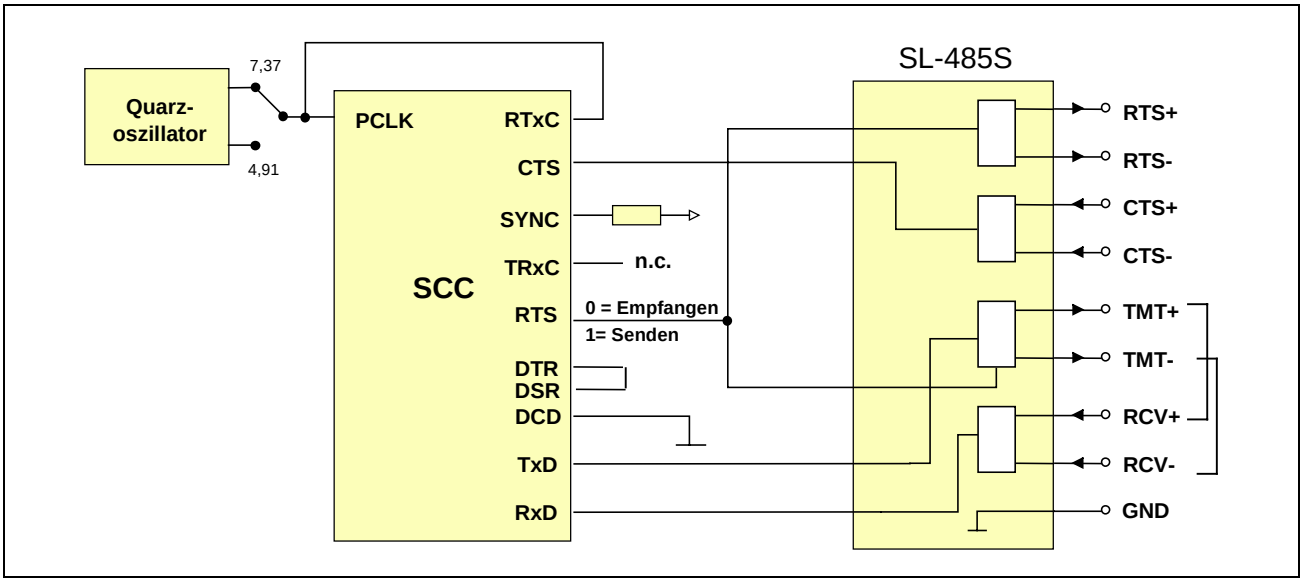


Abb. 3-17: *S-Link-Adapter SL-485S in Mode 1.*

SL-485S in Mode 3

Die RS-485-Anschlüsse CTS+ und CTS- können als Clock-Eingang dienen, das Signal liegt am RTxC Pin des SCC des zugehörigen Kanals.

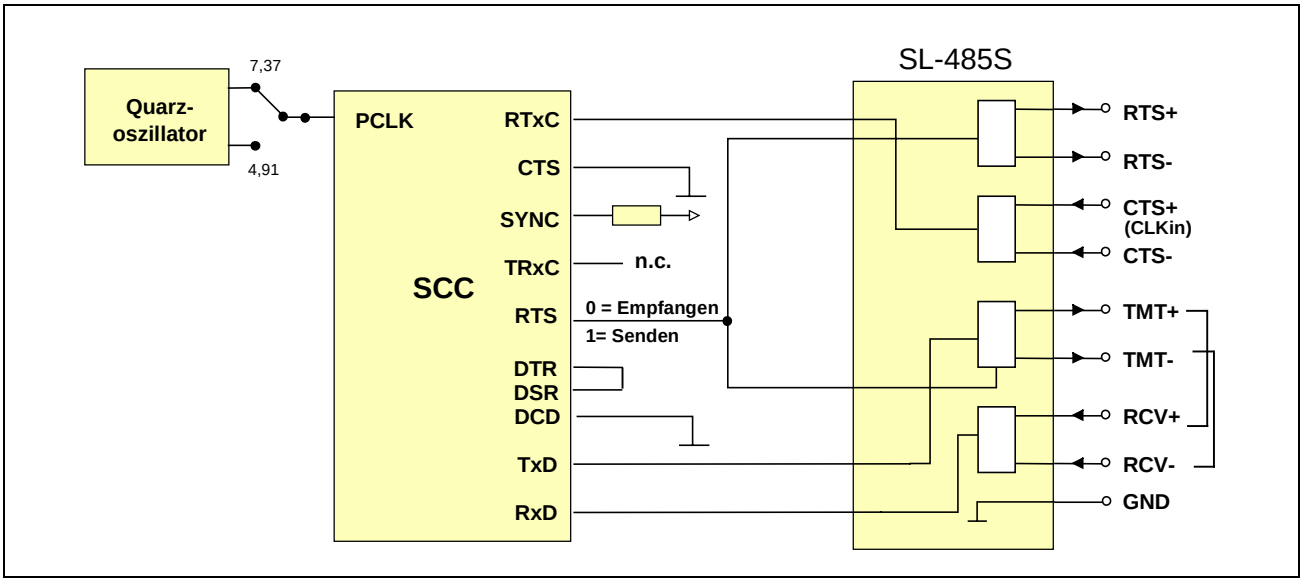


Abb. 3-18: *S-Link-Adapter SL-485S in Mode 3.*

SL-485S in Mode 5

Der Takt des Quarzoszillators liegt auch direkt am RTxC Pin dieses Kanals des SCC.

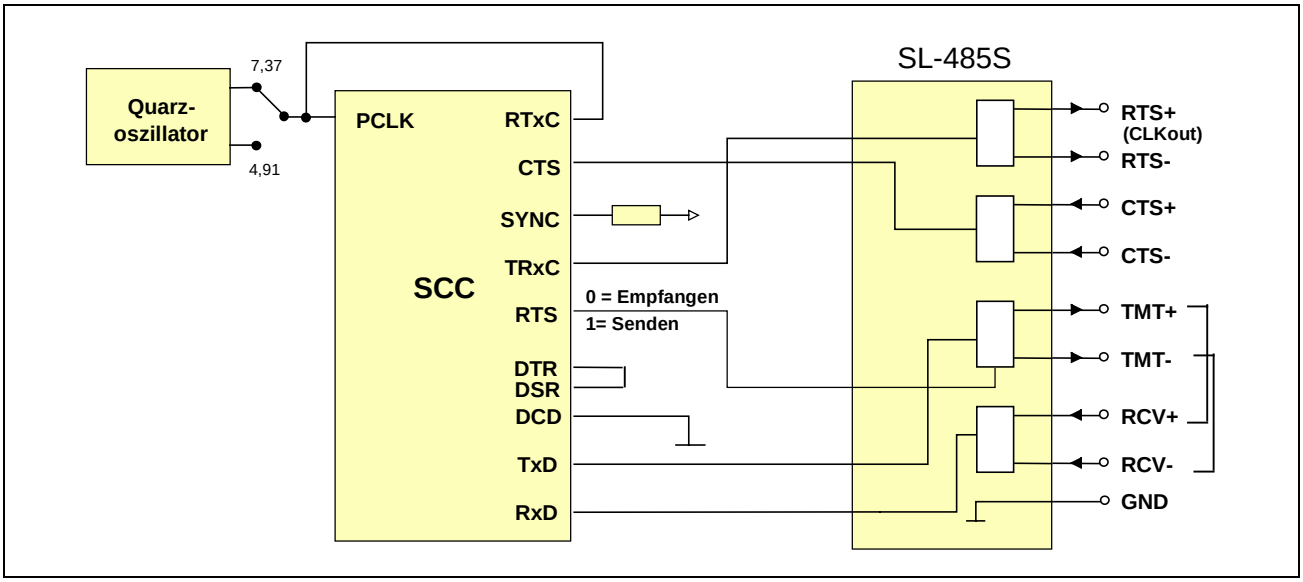


Abb. 3-19: *S-Link-Adapter SL-485S in Mode 5.*

SL-485S in Mode 7

Der Takt des Quarzoszillators liegt auch direkt am RTxC Pin dieses Kanals des SCC.

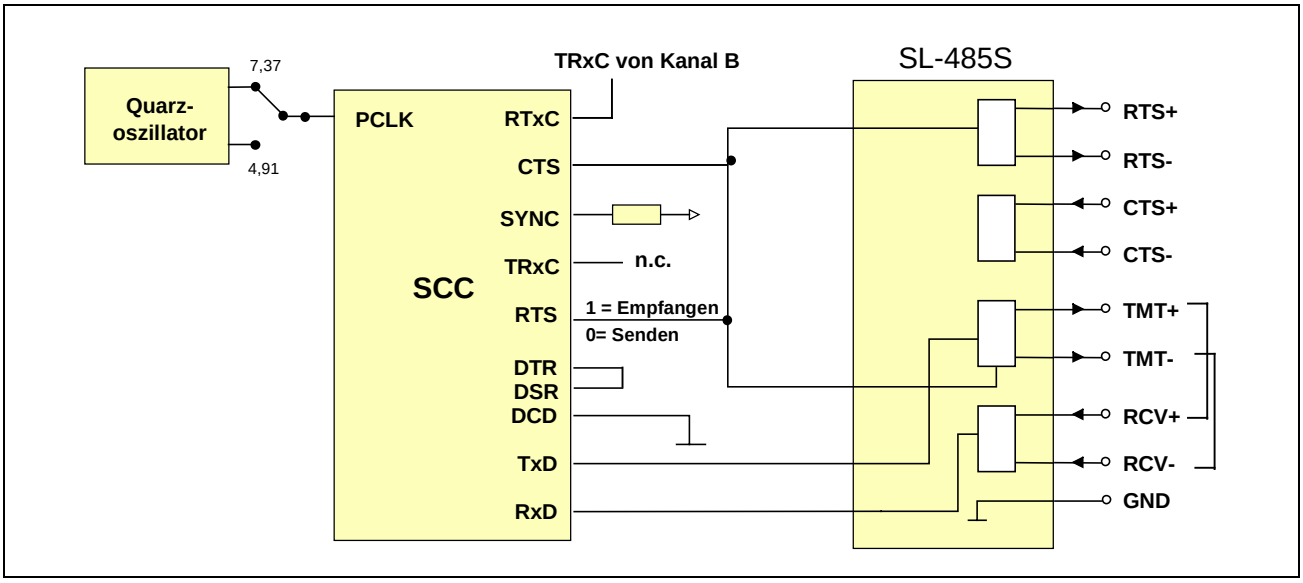


Abb. 3-20: *S-Link-Adapter SL-485S in Mode 7.*

3.5.12. S-Link SL-485i: RS-485, galvanisch getrennt

Dieser S-Link-Adapter stellt eine galvanisch getrennte RS-485-Schnittstelle für Baudraten bis 12 MBaud zur Verfügung und ist auch für PROFIBUS bis 12 MBaud geeignet.

Die Umschaltung zwischen Senden und Empfangen geschieht per Software durch Umschaltung zwischen Mode 2 und 3. In Mode 7 erfolgt die Umschaltung zwischen Senden und Empfangen über den Pin RTS des SCC.

Die zusätzliche Leitung DPRTS auf der galvanisch getrennten RS-485-Seite zeigt an, ob die RS-485-Schnittstelle Sender (DPRTS = log. 1) oder Empfänger (DPRTS = log. 0) ist. Diese Leitung wird von einem TTL-Buffer über einen Serienwiderstand von 300 Ω getrieben. Die Datenleitungen DPA und DPB sind überspannungsgeschützt und mit Pull-Up- (an DPB) bzw. Pull-Down-Widerständen (an DPA) von je 47 kΩ versehen. Der RS-485-Empfänger ist immer aktiv, beim Senden erscheint also das gesendete Signal wieder am Empfängereingang des SCC. Beachten Sie bitte, daß bei RS-485-Verbindungen + mit + und - mit - der Gegenstation verbunden wird.

Tab. 3-15: *Zusammenfassung der möglichen Modes mit SL-485i. Nach einem Reset des Systems ist immer Mode 0 eingestellt. Als Initialisierung sollte zunächst Mode 2 eingestellt werden (Empfangen) und danach der Pin TRxC des SCC als Ausgang geschaltet werden.*

		Pins des SCC							
Mode	**CTS**	**DCD**	**SYNC**	**RTS**	**Funktion von RTxC**	**TRxC Ein-/ Ausgang**	**Funktion von TRxC**	**DPRTS (RS-485)**	**Funktion von RS-485**
0	= 0	= 0	= 1	= x	keine	Eingang	keine	= 0	Empfangen
2	= 0	= 0	= 1	= x	= 0	Ausgang	keine	= 0	Empfangen
3	= 0	= 0	= 1	= 0	= 0	Ausgang	keine	= 1	Senden
	= 0	= 0	= 1	= 1	= 0	Ausgang	keine	= 0	Empfangen
7	= 0	= 0	= 1	= 0	= PCLK	Ausgang	keine	= 0	Empfangen
	= 0	= 0	= 1	= 1	= PCLK	Ausgang	keine	= 1	Senden

Abschlusswiderstände:

Das S-Link enthält keine Abschlusswiderstände.

Tab. 3-16: *Pinbelegung von St1 und Funktion der RS-485-Anschlüsse mit S-Link SL-485i in Mode 0, 2, 3 und 7 (Angaben für D-Submin. Stecker gelten für SORCUS Kabel K2-6259 und K3-6260).*

RS-485 Signal	RS-485 Ein-/ Ausgang	Funktion in Mode 0	2	3	7	Pin (St1) Kanal A	C	D	E	F	9-pol. D-Sub.
DPPE	Ausgang	GND	GND	GND	GND	4	8	12	16	20	1
DP5V	Ausgang	+5 V	+5 V	+5 V	+5 V	25	29	33	37	41	6
-	-	-	-	-	-	45	49	53	57	61	2
-	-	-	-	-	-	24	28	32	36	40	7
DPB	Ein-/Aus	-In	-In	-Out	-Out	3	7	11	15	19	3
DPA	Ein-/Aus	+In	+In	+Out	+Out	23	27	31	35	39	8
DPRTS	Ausgang	= 0	= 0	= RTS	= /RTS	44	48	52	56	60	4
-	-	-	-	-	-	2	6	10	14	18	9
DPGND	Ausgang	GND	GND	GND	GND	43	47	51	55	59	5
-	-	-	-	-	-	22	26	30	34	38	-

- = Pin ist nicht angeschlossen

SL-485i in Mode 2 und 3

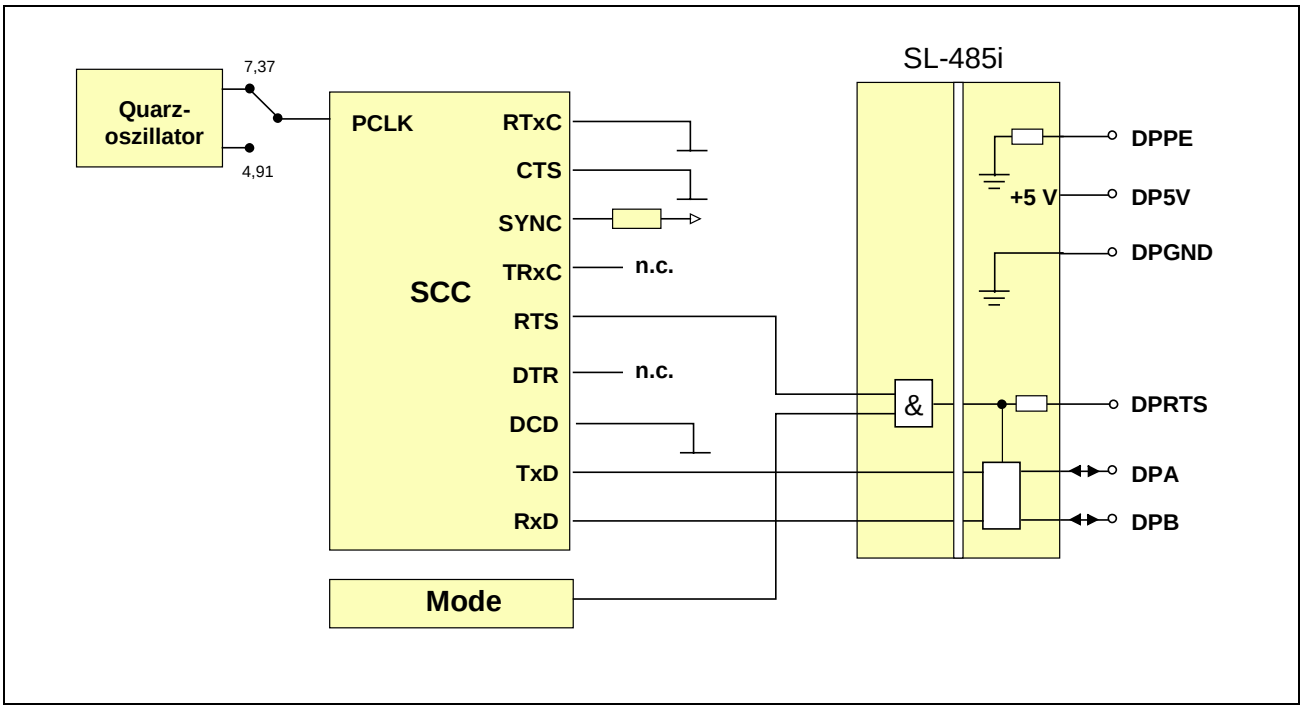


Abb. 3-21: *S-Link-Adapter SL-485i für RS-485 in Mode 2 und 3.*

SL-485i in Mode 7

Nach einem Reset des Systems ist immer Mode 0 eingestellt, der SCC-Pin RTS ist = 1 (= Empfangen). Nach der Einstellung von Mode = 7 muss der Pin TRxC des SCC als Ausgang geschaltet werden (TRxC hat im übrigen keine Funktion bei diesem S-Link). Mit dem SCC-Pin RTS kann per Software oder automatisch zwischen Empfangen und Senden umgeschaltet werden. RTS ist mit CTS des SCC verbunden, um einen Interrupt auslösen zu können, wenn die Sende-/Empfangsrichtung umgeschaltet wird.

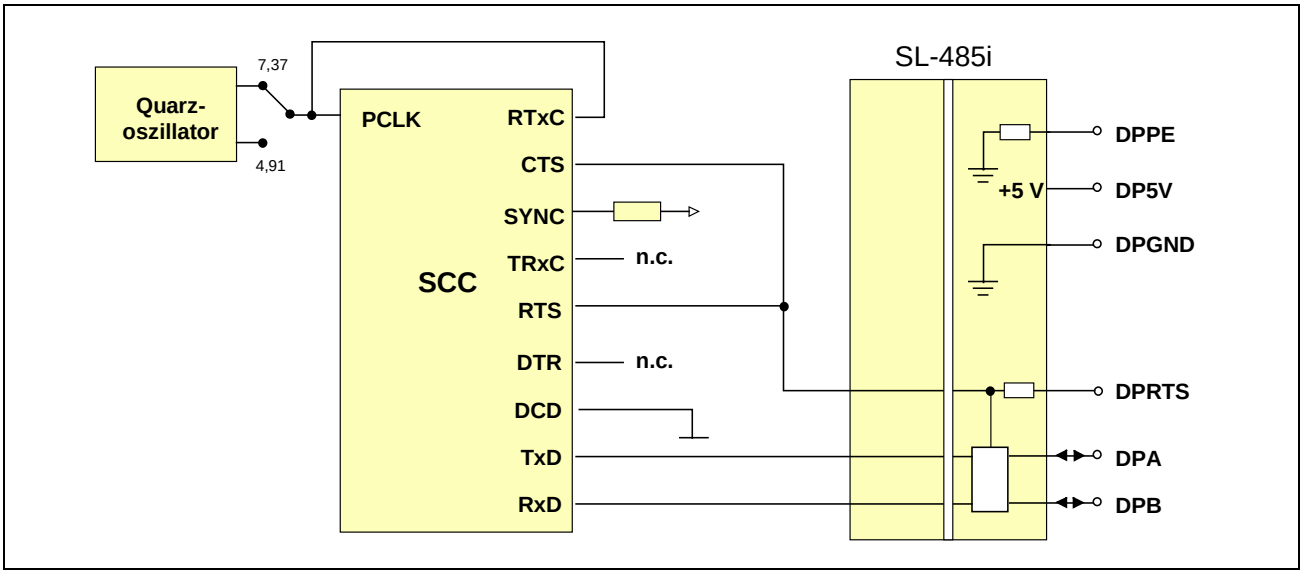


Abb. 3-22: *S-Link-Adapter SL-485i für RS-485 in Mode 7.*

3.5.13. S-Link SL-20MA: 20 mA Current Loop

Auf dem S-Link SL-20MA sind zwei 20 mA Konstantstromquellen vorhanden, je eine für Senden (TMT) und Empfangen (RCV). Ihre Verwendung ist optional. Wenn ein Teil der Schnittstelle (RCV oder TMT) mit Hilfe einer der beiden 20 mA Konstantstromquellen den Strom für die Verbindung liefert, wird er als aktiv bezeichnet. Wenn der Strom von der Gegenstation kommt, als passiv. Nur ein passiver Teil ist auf dem S-Link galvanisch getrennt. Die Konfiguration (aktiv/passiv) wird über Verbindungen am externen Anschlussstecker eingestellt, kann also auch nach dem Einbau der Karte noch geändert werden.

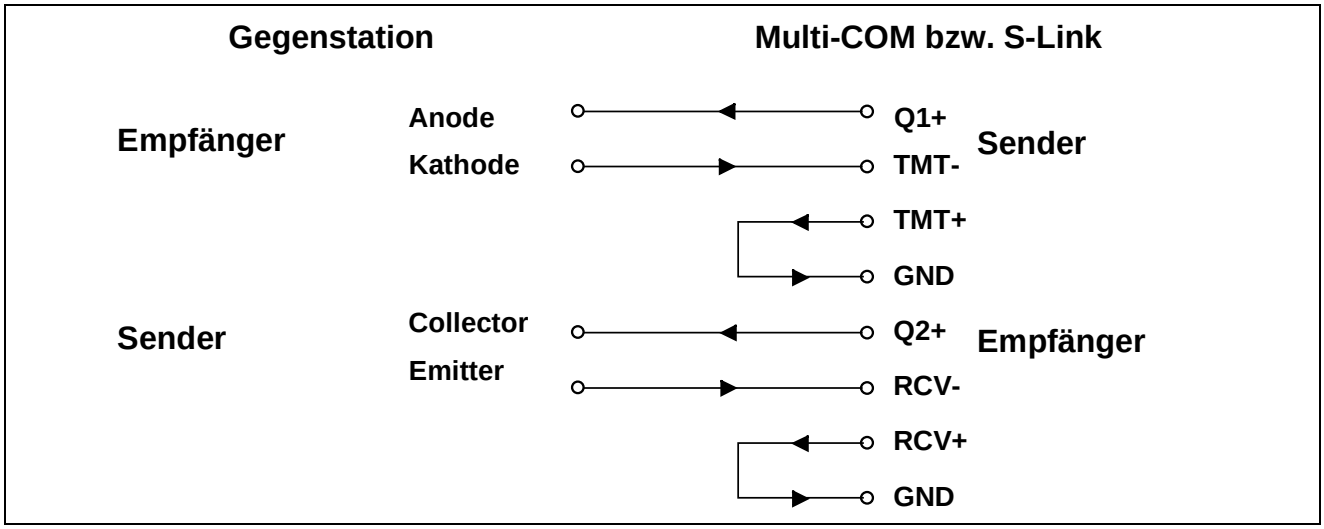
Beachten Sie bitte, dass bei 20 mA Verbindungen + mit - und - mit + der Gegenstation verbunden wird. Q1 und Q2 sind die Ausgänge der beiden 20 mA Konstantstromquellen.

Tab. 3-17: *20 mA Current Loop mit S-Link SL-20MA*

Signal	**Pin (St1) Kanal**					**RCV:**	**aktiv**	**aktiv**	**passiv**	**passiv**
S-Link	**A**	**C**	**D**	**E**	**F**	**TMT:**	**aktiv**	**passiv**	**aktiv**	**passiv**
-12 V	4	8	12	16	20		-	-	-	-
TMT-	2	6	10	14	18		TMT-	TMT-	TMT-	TMT-
TMT+	43	47	51	55	59			TMT+		TMT+
GND	24	28	32	36	40		*		*	-
Q1+	3	7	11	15	19		TMT+	-	TMT+	-
RCV-	23	27	31	35	39		* RCV-	* RCV-	RCV-	RCV-
RCV+	44	48	52	56	60				RCV+	RCV+
Q2+	45	49	53	57	61		RCV+	RCV+	-	-
-	22	26	30	34	38		-	-	-	-
-	1	5	9	13	17		-	-	-	-
n.c.	25	29	33	37	41		n.c.	n.c.	n.c.	n.c.
n.c.	46	50	54	58	62		n.c.	n.c.	n.c.	n.c.

\- = Pin ist nicht angeschlossen bzw. nicht vorhanden
* = Lötverbindungen am externen Anschlussstecker
n.c. = Pin darf nicht angeschlossen werden

Beispiel: Empfänger und Sender auf dem S-Link sind aktiv (nicht galvanisch getrennt) und mit den Optokopplern der Gegenstation verbunden:



Die Stromquellen auf dem S-Link werden von +12 Volt gespeist. Um den Spannungshub zu vergrößern, kann die Rückführung statt an Ground (wie in folgender Tabelle angegeben) auch an -12 Volt erfolgen:

Tab. 3-18: *Vergrößerter Spannungshub bei 20 mA Current Loop mit S-Link SL-20MA*

Signal S-Link	**Pin (St1): Kanal A**	**C**	**D**	**E**	**F**	**RCV: TMT:**	**aktiv aktiv**	**aktiv passiv**	**passiv aktiv**	**passiv passiv**
-12 V	4	8	12	16	20					-
TMT-	2	6	10	14	18		TMT- *	TMT-	TMT- *	TMT-
TMT+	43	47	51	55	59			TMT+		TMT+
GND	24	28	32	36	40		-	- *	-	-
Q1+	3	7	11	15	19		TMT+ *	-	TMT+	-
RCV-	23	27	31	35	39		RCV-	RCV-	RCV-	RCV-
RCV+	44	48	52	56	60				RCV+	RCV+
Q2+	45	49	53	57	61		RCV+	RCV+	-	-
-	22	26	30	34	38		-	-	-	-
-	1	5	9	13	17		-	-	-	-
n.c.	25	29	33	37	41		n.c.	n.c.	n.c.	n.c.
n.c.	46	50	54	58	62		n.c.	n.c.	n.c.	n.c.

- = Pin ist nicht angeschlossen bzw. nicht vorhanden
* = Lötverbindungen am D-Submin.-Stecker
n.c. = Pin darf nicht angeschlossen werden

Für einen mit dem S-Link-Adapter SL-20MA ausgerüsteten Kanal muss Mode 1 eingestellt werden. Der Takt des Quarzoszillators liegt damit auch direkt am RTxC Pin dieses Kanals des SCC.

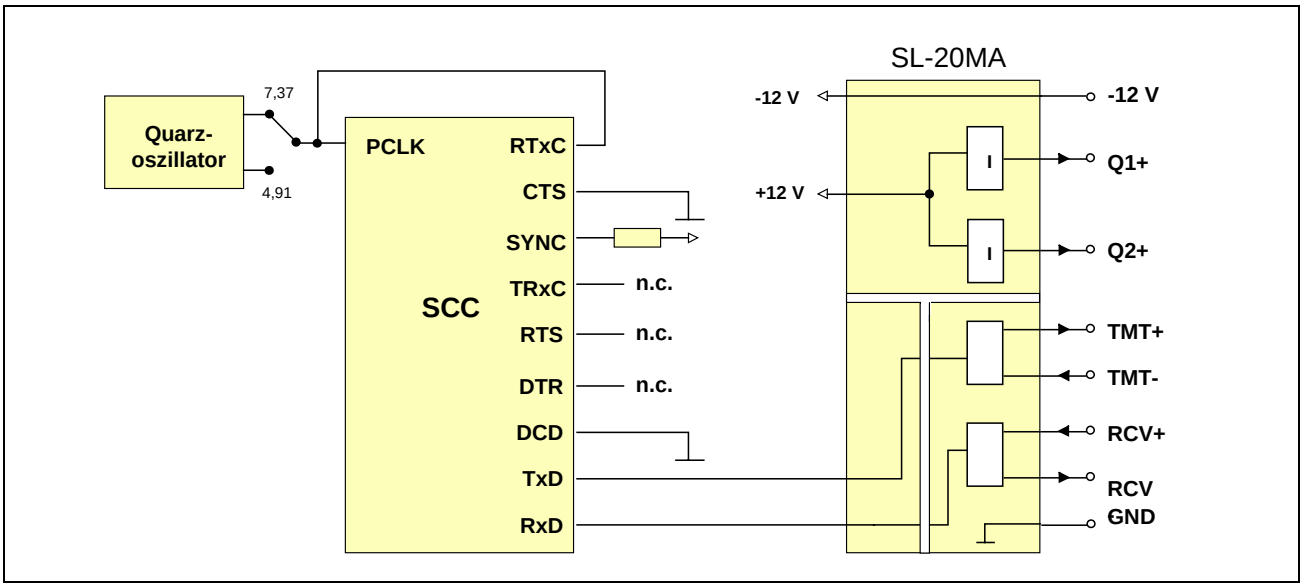


Abb. 3-23: *S-Link-Adapter SL-20MA in Mode 1.*

3.5.14. S-Link SL-LWL/P und /G: Lichtwellenleiter

Dieses S-Link für Lichtwellenleiter ist für Plastik- (/P) und Glasfaserkabel (/G) lieferbar. Eine Kontroll-LED auf dem S-Link zeigt den Pegel am RCV-Pin des SCC an: log. 0 = LED ein, log. 1 = LED aus.

Nur Mode 0 und 1 sind möglich. Damit kann eingestellt werden, ob die Sende- und Empfangspegel "invertiert" oder "nicht invertiert" sein sollen:

Tab. 3-19: *SL-LWL/P und SL-LWL/G Betriebsarten*

Mode	Funktion
0	Sende- und Empfangspegel "invertiert"
1	Sende- und Empfangspegel "nicht invertiert"

Bitte beachten Sie, dass der SCC-Pin TRxC im SCC als Eingang konfiguriert werden muss. Außerdem müssen die CTS-, DCD- und SYNC-Interrupts im SCC deaktiviert (disabled) sein.

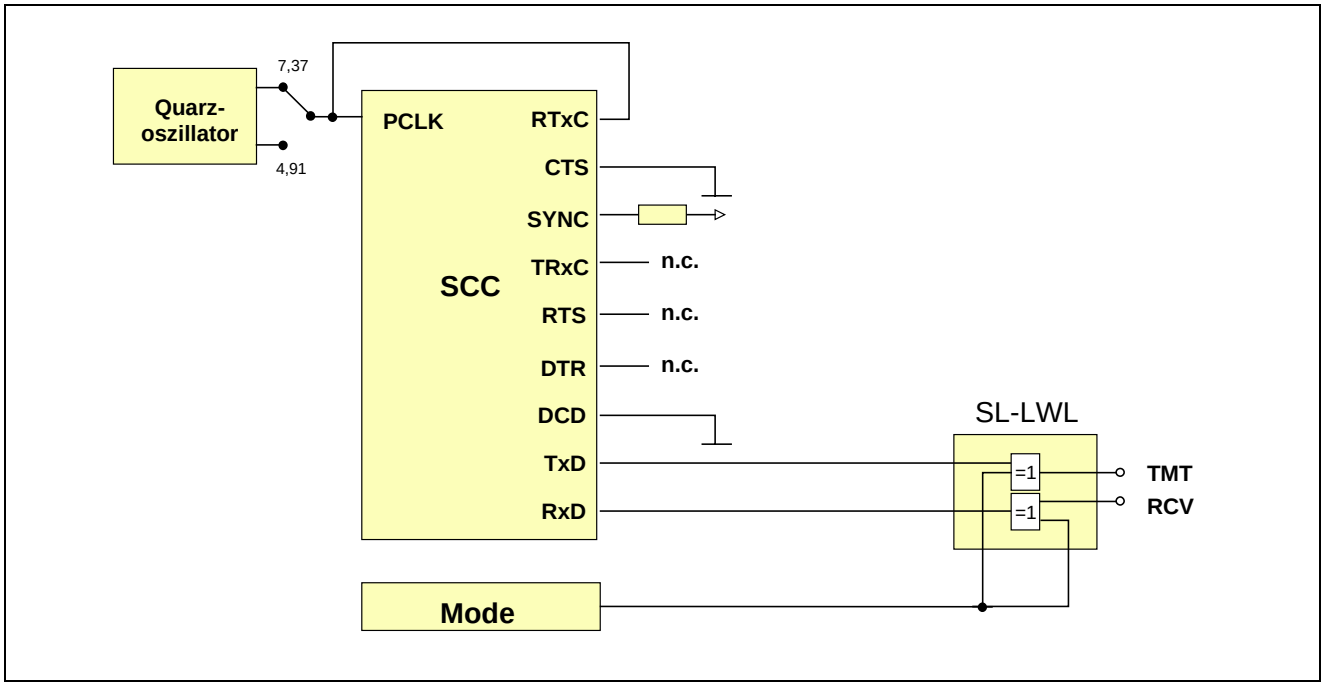


Abb. 3-24: *S-Link-Adapter SL-LWL/P und /G.*

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 Multi-COM-Karte wie auch für die aufgesteckten S-Links. 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 Multi-COM 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 Kommunikationsaufgaben eingesetzt werden können. Durch ihre freie Programmierbarkeit und durch die aufsteckbaren S-Links bieten Multi-COM 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 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 eine serielle Schnittstelle der Multi-COM erfolgen. 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 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 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 SNW6 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 Echtzeitprogramme

In diesem Fall wird die gesamte Echtzeitprogrammierung durch die für die Karte vorhandenen Programme zur Verfügung gestellt (vorausgesetzt, die Programme decken 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 (CQ6) bzw. fertiger Protokolltasks, die auf der Multi-COM 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 Diskette mitgeliefert wurde.

5.1.2. 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 Multi-COM Karte enthält ein eigenes Betriebssystem im ROM (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 Multi-COM 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 Multi-COM 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 Multi-COM 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:

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 Device-Treiber reserviert.

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: Über eine serielle Schnittstelle werden Daten ausgetauscht. Sobald ein Zeichen empfangen wurde, löst der Kommunikationsbaustein (SCC) einen Interrupt aus. Ein Kommunikationsprogramm wird unter diesem Interrupt installiert. Bei Auftreten eines Interrupts wird die Hauptprozedur dieser Task aufgerufen. Das empfangene Zeichen wird ausgelesen und zur Pufferung in den Speicher geschrieben.

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 Kommunikationsprogramm werden nach der Installierung die Übertragungsparameter (Baudrate, Anzahl Datenbits, etc.) im Kommunikationsbaustein (SCC) 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 Multi-COM Karte weitgehend festgelegt. Es besteht zwar von Hardwareseite durch ent-

sprechende 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 Interrupt-Eingänge der seriellen Schnittstellen 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 Multi-COM 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.

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-Tasks 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 über Betriebssystemaufrufe oder direkt durch Angabe der Nummer des Parameters zugegriffen. Die Nummer ist die relative Adresse des ersten Byte des Parameters, bezogen auf den Anfang des Parameterbereichs.

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

5.2.5. Datenpuffer

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 Multi-COM 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 Diskette geliefert wurde, muss zunächst in das RAM der Karte geladen und anschließend installiert werden. Der Name solcher Programmdateien ist:

M6Pxxxx.LIB oder

M6Pxxxx.EXE

Dabei bedeutet xxxx die Programmnummer zwischen 1 und ffffh (hexadezimal).

Zur komfortablen Installierung der Programme finden Sie auf einer der Originaldisketten, die mit der Multi-COM Karte geliefert wurden, das Hilfsprogramm

"SNW6" bzw. **"SNW32".**

Das Programm "SNW6" 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.

Wenn Sie die Programme aus Ihrem eigenen PC-Programm heraus installieren wollen, stehen dafür Funktionen in der PC-Bibliothek "ML6BIB" 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 Multi-COM erstellt werden.*

Programme im ROM (oder Flash-EPROM) können einfach mit den entsprechenden Bibliotheksbefehlen oder mit SNW6 bzw. SNW32 installiert werden. Außerdem kann eine Liste zu installierender ROM-Programme in ROM bzw. FLASH-EPROM abgelegt werden, die nach dem Einschalten der Multi-COM 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 SNW6. Alternativ kann (per Jumper auf St3) das Mini-Betriebssystem veranlasst werden, nach einem Reset automatisch das ROM-Betriebssystem zu starten.

Das Betriebssystem der Multi-COM 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 SNW6 (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:

ML6-za.nnR	Neues Betriebssystem (z = Zahl 2 bis 9 (Version), a = Buchstabe A bis Z (Revision), nn = lfd. Nr. des Betriebssystems)

Das Programm SNW6 bietet die Möglichkeit, bei einem Reset in einer Installationsdatei (in der M6DEVICE-Zeile) oder einem interaktiven Reset (z.B. [CTRL]+[F10]) das jeweils aktuellste Betriebssystem zu laden. Dazu muss als Name "ML6-??.??R" angegeben werden. SNW6 durchsucht dann beim Reset das "OSX"-Verzeichnis (wird beim Installieren der Originaldisketten erstellt) und das Verzeichnis, in dem "SNW6.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 z.Zt. für Programme genutzt werden. 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 (6 = Multi-COM)
5 (05h)	B	R	Kartenrevision (1 = A, 2 = B, 3 = C, ...)
12 (0ch)	10S	R	Betriebssystem Name, Version und Revision, z.B.: "ML6-1A.01x" 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.: "24/12/97" (tt/mm/jj)
30 (1eh)	8S	R	Uhrzeit der Herstellung des Betriebssystems, z.B.: "23:42:59" (hh:mm:ss)
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, 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 und Nexgen 586)

Parameter	**Typ**	**Zugr**	**Bedeutung**
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)
100 (64h)	W	R	EEPROM der Multi-COM Karte: Länge (Anzahl Byte)
102 (66h)	W	R	Parameter-Nr., ab der die EEPROM-Kopie steht oder Fehlercode, wenn Parameter 100 = 0
104 (68h)	W	R	EEPROM von Kanal A u. B (SCC1): 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 Kanal C u. D (SCC2): 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 Kanal E u. F (SCC3): 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	Reserviert
118 (76h)	W	R	Reserviert

Parameter	Typ	Zugr	Bedeutung
148 (94h)	W	R	CPU-Taktfrequenz (in Vielfachen von 100 kHz)
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
214 (d6h)	B	RW	Watchdog: 0=Auto-Retrigger off, 1=on
308 (134h)	W	R	Max. Anzahl TI-Tasks
311 (137h)	B	RW[1]	Timerkanal für TI-Tasks (0=Timer-A, 1=B, 2=C) (Standardeinstellung: Timer-C)
313 (139h)	B	RW[1]	Interrrupt-Nr. für TI-Task Timer (Standardeinst.: 93h = Timer-C)
316 (13ch)	D	RW[1]	Timer-Tic für TI-Tasks in Mikrosekunden 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 zum PC 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, spätere Einstellungen sind wirkungslos.

Notizen

6. PC-Hochsprachenbibliotheken

Die Hochsprachenbibliotheken wickeln auf der PC-Seite die Kommunikation zwischen PC und der Multi-COM Karte ab. Sie enthalten Prozeduren und Funktionen, deren hauptsächliche Aufgabe es ist, das Multi-Tasking-Betriebssystem der Multi-COM Karte über Makrobefehle anzusprechen. Es stehen Bibliotheken für die PC-Betriebssysteme DOS, Windows 3.x, Windows NT und Windows 95) zur Verfügung. Die Bibliotheken sind für die Programmiersprachen C (Borland und Microsoft), Pascal (Borland Pascal) und Delphi im Lieferumfang enthalten.

Die Bibliothek ML6BIB enthält auch Funktionen für die Initialisierung der Multi-COM Karte und für die Fehlerbehandlung. Sie unterstützt bis zu acht Multi-COM Karten in einem PC und gestattet das Arbeiten mit PC-Interrupt-Auslösung durch die Multi-COM 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 Multi-COM Karte unter der richtigen Basisadresse (Werkseinstellung 380h) ansprechen. Die Betriebssystemversion der Multi-COM muss größer oder gleich ML6-1A.01E (ROM-Betriebssystem) bzw. ML6-1A.01R (ladbares Betriebssystem) sein. Die Bibliothek (ML6BIB) muss Version 2.A.000 oder höher aufweisen. Bei Versionen größer 2.A.000 lesen Sie bitte auch die Datei **README.DOC**.

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 den mitgelieferten Disketten.

Für C-Compiler sind folgende Bezeichner (mit #define) definiert:

UCHAR	unsigned char
USHORT	unsigned short
ULONG	unsigned long
TRUE	1
FALSE	0

Als Zeiger für Fehlerbehandlung und Benutzer-Service-Routine sind die Typen **ERROR_PROC_P** und **SERVICE_PROC_P** deklariert.

Für Pascal-Compiler ist folgender Typ definiert:

str80	= STRING[80]

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 6-51.

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 6-51.

6.2.2. Windows 3.x

Für Windows 3.x steht die DLL WML6BIB.DLL zur Verfügung, welche von den verschiedenen Compilern benutzt wird. Die Aufrufkonvention der Funktionen ist **far pascal**. Zur Laufzeit eines Programms muß auf diese DLL zugegriffen werden können.

Die Bibliothek wurde 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 6-51.

6.2.3. Windows 95

Für Windows 95 steht die DLL **MLXW32.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 '\ml6\win95' in der Datei 'readme.wri'.

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 und 98

Für Windows NT bzw. 98 steht die DLL **MLXW32BIB.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 '\ml6\winnt' in der Datei 'readmed.wri'.

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 NT) gleichzeitig die Karte, so führt dies zu Kommunikationsfehlern.

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

6.3. Bibliotheksfunktionen

6.3.1. Funktionen zur Initialisierung

ml6_bib_startup **Initialisiere die Bibliothek**

Pascal	PROCEDURE ml6_bib_startup (language: WORD);
C	VOID ml6_bib_startup (USHORT language);
Funktion	Diese Funktion initialisiert die Bibliothek.
Parameter	*language*: Für die Programmiersprache C muß dieser Parameter auf LANGUAGE_C gesetzt werden, für Pascal und Delphi auf LANGUAGE_PASCAL.
Hinweis	Die Prozedur muss vor allen anderen Bibliotheksroutinen aufgerufen werden. Danach muss die Schnittstelle zur Karte mit **ml6_reset** oder **ml6_start** initialisiert werden. Erst dann dürfen die übrigen Bibliotheksroutinen verwendet werden.

ml6_reset — Reset und Initialisieren der Kommunikation PC-Karte

Pascal
FUNCTION ml6_reset (card, adr, mode, irq_channel, dma_channel: WORD; error_handler: POINTER; VAR osname: str80; timeout: WORD): WORD;

C
USHORT ml6_reset (USHORT card, USHORT adr, USHORT mode, USHORT irq_channel, USHORT dma_channel, ERROR_PROC_P error_handler, CHAR* osname, USHORT timeout);

Funktion
Diese Funktion wählt eine von maximal acht Multi-COM Karten an, stellt eine Betriebsart der Karte ein, führt einen Reset der Karte durch, aktiviert ein Betriebssystem und führt einen ersten Kommunikationstest durch.

Falls kein Reset der Karte durchgeführt und kein neues Betriebssystem auf der Karte aktiviert werden soll, so verwenden Sie stattdessen die Funktion **ml6_start** (s. u.).

Rückgabe
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. Die Fehlerbehandlungsprozedur wird nicht aufgerufen.

Parameter

card: Anzuwählende Multi-COM Karte (0 bis 7).

adr: Basis-I/O-Adresse der Multi-COM Karte (s. Kap. 2.2.1.).

mode: Betriebsart der Kommunikation mit der Multi-COM Karte. Es stehen folgende Betriebsarten zur Verfügung:

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

irq_channel: Verwendeter Interrupt-Kanal (nur für *mode* = 1). Mögliche Werte sind 3, 4, 5, 7, 9, 10, 11 und 12. Unter Windows 95 und Windows NT wird dieser Parameter nicht ausgewertet. Der Interrupt-Kanal wird der Registrierdatenbank entnommen (s. Kap. 2.2.2.).

dma_channel: Wird nicht verwendet, = 0 setzen.

error_handler: Adresse der Fehlerbehandlungsprozedur. Weitere Informationen zur Fehlerbehandlung finden Sie ab Seite 6-51.

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, dass 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 Multi-COM innerhalb dieser Zeit nicht erwartungsgemäß reagiert, wird ein Fehler gemeldet.

ml6_start — Initialisiere die Kommunikation PC-Karte

Pascal
FUNCTION ml6_start (card, adr, mode, irq_channel, dma_channel: WORD; error_handler: POINTER; timeout: WORD): WORD;

C
USHORT ml6_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 **ml6_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 **ml6_reset**.

ml6_select_card — Wähle eine Multi-COM Karte an

Pascal
PROCEDURE ml6_select_card (card: BYTE);

C
VOID ml6_select_card (USHORT card);

Funktion
Durch Aufruf der Prozedur wird eine von maximal acht Multi-COM Karten angewählt. Die Prozedur dient zum Umschalten zwischen ver-

schiedenen Multi-COM 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, **ml6_select_card** ist hier nicht erforderlich.

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

ml6_exit_card — Melde eine Karte ab

Pascal PROCEDURE ml6_exit_card (card: BYTE);

C VOID ml6_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 **ml6_start** oder **ml6_reset** kann die Karte wieder angesprochen werden. Vor Beendigung des Programms muß für jede mit **ml6_start** oder **ml6_reset** angesprochene Karte diese Prozedur aufgerufen werden (siehe auch **ml6_exit**).

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

ml6_exit — Beende die PC-Kommunikation mit der Karte

Pascal PROCEDURE ml6_exit;

C VOID ml6_exit (VOID);

Funktion Diese Prozedur kann alternativ zu **ml6_exit_card** am Programmende aufgerufen werden. Sie ruft für alle verwendeten (initialisierten) Karten einmal die Prozedur **ml6_exit_card** auf.

ml6_reacting — Prüfe, ob die Multi-COM Karte bereit ist

Pascal FUNCTION ml6_reacting: BOOLEAN;

C USHORT ml6_reacting (VOID);

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

ml6_type_check — Identifiziere das aktive Betriebssystem

Pascal FUNCTION ml6_type_check: BYTE;

C UCHAR ml6_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).

ml6_prog_in_rom — Prüfe, ob Echtzeit-Programm im ROM ist

Pascal FUNCTION ml6_prog_in_rom (pgm: WORD): BOOLEAN;

C UCHAR ml6_prog_in_rom (USHORT pgm);

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

Parameter *pgm*: Nummer des Programms.

ml6_get_selected_card — Ermittle Nr. der angewählten Karte

Pascal FUNCTION ml6_get_selected_card: WORD;

C USHORT ml6_get_selected_card (VOID);

Funktion Diese Funktion liefert die aktuell angewählte Kartennummer.

ml6_get_lib_version — Ermittle den Versions- und Datecode der Bibliothek

Pascal PROCEDURE ml6_get_lib_version (VAR version, date: LONGINT);

C VOID ml6_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.

ml6_get_osx_version — Ermittle den Versions- und Datecode des Betriebssystems

Pascal PROCEDURE ml6_get_osx_version (VAR version, date: LONGINT);

C VOID ml6_get_osx_version (ULONG* version, ULONG* date);

Funktion Die Prozedur liefert den Versionscode und den Datecode des auf der Multi-COM verwendeten Betriebssystems

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

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

ml6_init_io — Initialisiere Basiskarte und S-Links

Pascal PROCEDURE ml6_init_io (device, option: BYTE);

C VOID ml6_init_io (UCHAR device, UCHAR option);

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

Parameter *device*: Gibt an, welche Hardware initialisiert werden soll:
- 0: Multi-COM Karte
- 1: Serielle Schnittstellen A und B (SCC1)
- 2: Serielle Schnittstellen C und D (SCC2)
- 3: Serielle Schnittstellen E und F (SCC3)

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*).

[1] Bedeutung der Codes siehe Kapitel 6.4.

ml6_change_timeout — Ändere den Timeout-Wert

Pascal: PROCEDURE ml6_change_timeout (tenthsec: WORD);

C: VOID ml6_change_timeout (USHORT tenthsec);

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

Parameter: *tenthsec*: Timeout-Wert in Zehntelsekunden (s. **ml6_reset**).

6.3.2. Laden von Echtzeitprogrammen auf die Multi-COM

ml6_transfer_and_install — Installiere Programm auf der Karte

Pascal: FUNCTION ml6_transfer_and_install (pgmname: str80; usage, tasknr, pgmnr, irqnr: WORD; flags, datasize: LONGINT): WORD;

C: USHORT ml6_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 Multi-COM 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: Dient zu Protokollzwecken auf der Karte und muss für Anwendungsprogramme = 0 gesetzt werden.

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 **ml6_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 **ml6_transfer_and_install** festgelegt

Bit	Wert	Bedeutung
5-4		Privilegstufe des Programms
	0	(= 00b): höchste Privilegstufe (Systemprogramme)
	16	(= 01b): zweithöchste Privilegstufe
	32	(= 10b): dritthöchste Privilegstufe
	48	(= 11b): niedrigste Privilegstufe (Anwendungsprogramme)
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

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

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

ml6_transfer_pgm — Lade Programm in den Speicher der Multi-COM

Pascal
FUNCTION ml6_transfer_pgm (task, usage: WORD; pgmname: str80; VAR loadadr; VAR prepadr; VAR flags: LONGINT): WORD;

C
USHORT ml6_transfer_pgm (USHORT task, USHORT usage, CHAR* pgmname, ULONG* loadadr, ULONG* prepadr, ULONG* flags);

Funktion
Diese Funktion lädt ein Programm in den Speicher der Multi-COM 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 Dateinamenserweiterung '.EXE' übergeben wird, so sorgt die Funktion beim Übertragen des Programms automatisch für die erforderliche Relokierung 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 **ml6_transfer_and_install**.

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

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

flags: Zeiger auf eine Variable, die die Installationsoptionen enthält (siehe **ml6_transfer_and_install**).

Hinweis
Verwenden Sie **ml6_transfer_pgm** nur dann, wenn Sie ein Programm mehrfach installieren wollen. Um ein Programm einfach zu installieren, verwenden Sie bitte **ml6_transfer_and_install**.

ml6_install_task — Installiere Programm

Pascal
PROCEDURE ml6_install_task (task, programnr, interruptnr: WORD; flags, datasize, adr: LONGINT);

C
VOID ml6_install_task (USHORT task, USHORT programnr, USHORT interruptnr, ULONG flags, ULONG datasize, ULONG adr);

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

Parameter
tasknr, *pgmnr*, *irqnr*, *flags*, *datasize*: siehe **ml6_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 **ml6_transfer_pgm** zurückgelieferte Adresse eingetragen.

! Hinweis
Verwenden Sie **ml6_install_task** nur dann, wenn Sie ein Programm mehrfach installieren wollen. Um ein Programm einfach zu installieren, verwenden Sie besser **ml6_transfer_and_install**.

6.3.3. Taskbefehle

Die meisten der in den folgenden Kapiteln beschriebenen Funktionen enthalten den Parameter *task*. Dieser spezifiziert die Nummer, unter der das Betriebssystem auf der Multi-COM Karte das zugehörige Echtzeitprogramm verwaltet.

6.3.3.1. Taskinformationen abfragen

ml6_get_task_info **Lies Systeminformationen einer Task**

Pascal PROCEDURE ml6_get_task_info (task, callnr: WORD;
VAR result: LONGINT);

C VOID ml6_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.

ml6_get_task_status **Ermittle, ob Task aktiviert ist**

Pascal FUNCTION ml6_get_task_status (task: WORD): BYTE;

C UCHAR ml6_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.

ml6_get_task_number Ermittle Tasknummer

Pascal FUNCTION ml6_get_task_number (prog_nr: WORD;
VAR install_nr: WORD): WORD;

C USHORT ml6_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 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.

ml6_get_int_task — Ermittle Tasktyp und -nummer

Pascal PROCEDURE ml6_get_int_task (inter: WORD; VAR task: WORD VAR dummy: BYTE);

C void ml6_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.

ml6_get_pgm_installed — Ermittle Programmnummer einer Task

Pascal FUNCTION ml6_get_pgm_installed (task: WORD): WORD;

C USHORT ml6_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.

6.3.3.2. Tasks aktivieren und deaktivieren

ml6_wakeup_task — Aktiviere eine Task

Pascal PROCEDURE ml6_wakeup_task (task: WORD);

C VOID ml6_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 (**ml6_wakeup_ti_task**).

ml6_sleep_task — Deaktiviere eine Task

Pascal PROCEDURE ml6_sleep_task (task: WORD);

C VOID ml6_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.

ml6_wakeup_ti_task — Aktiviere eine TI-Task

Pascal PROCEDURE ml6_wakeup_ti_task (task: WORD; priority: BYTE; count, interval, holdoff: LONGINT);

C VOID ml6_wakeup_ti_task (USHORT task, UCHAR 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.

6.3.3.3. Taskfunktionen aufrufen

ml6_call_func Aufruf einer Funktion einer Task

Pascal
PROCEDURE ml6_call_func (task, func, outsize: WORD;
VAR outdata_var; maxinsize: WORD; VAR insize: WORD;
VAR indata_var; VAR error: BYTE);

C
VOID ml6_call_func (USHORT task, USHORT func, USHORT outsize, VOID* outdata_var, USHORT maxinsize, USHORT* insize, VOID* indata_var, UCHAR* error);

Funktion
Mit dieser Prozedur kann eine beliebige Funktion aus einer auf der Multi-COM installierten Task aufgerufen werden.

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 Multi-COM 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.

ml6_call_proc — Aufruf einer Prozedur einer Task

Pascal PROCEDURE ml6_call_proc (task, procnr: WORD);

C VOID ml6_call_proc (USHORT task, USHORT procnr);

Funktion Diese Prozedur ist ein Sonderfall der Prozedur **ml6_call_func**: Es werden keine Daten an die aufgerufene Prozedur übergeben oder von ihr übernommen.

Parameter *procnr*: Nummer der Prozedur.

6.3.4. Zugriff auf den Parameterbereich einer Task

Zugriffe auf Variablen 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 Parameter als Block gelesen oder geschrieben werden, spezifiziert dieser Wert den ersten zu lesenden oder zu schreibenden Parameter.

ml6_read_par_byte
ml6_read_par_word
ml6_read_par_dword — Lies Parameter einer Task

Pascal FUNCTION ml6_read_par_byte (task, start: WORD): BYTE;

FUNCTION ml6_read_par_word (task, start: WORD): WORD;

PROCEDURE ml6_read_par_dword (task, start: WORD;
VAR data: LONGINT);

C UCHAR ml6_read_par_byte (USHORT task, USHORT start);

USHORT ml6_read_par_word (USHORT task, USHORT start);

VOID ml6_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.

Parameter *data*: Zeiger auf eine Variable, die das Ergebnis von **ml6_read_par_dword** aufnimmt.

ml6_read_par_block — Lies Parameterblock einer Task

Pascal
PROCEDURE ml6_read_par_block (task, start, size: WORD;
VAR data_var);

C
VOID ml6_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 **ml6_set_macro_interruptible** die Makrobefehle unterbrechbar geschaltet wurden.

ml6_write_par_byte
ml6_write_par_word
ml6_write_par_dword — Schreibe Parameter einer Task

Pascal
PROCEDURE ml6_write_par_byte (task, start: WORD; data: BYTE);

PROCEDURE ml6_write_par_word (task, start: WORD;
data: WORD);

PROCEDURE ml6_write_par_dword (task, start: WORD;
data: LONGINT);

C
VOID ml6_write_par_byte (USHORT task, USHORT start, UCHAR data);

VOID ml6_write_par_word (USHORT task, USHORT start, USHORT data);

VOID ml6_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.

Parameter
data: Wert, der geschrieben werden soll.

ml6_write_par_block Schreibe Parameterblock einer Task

Pascal PROCEDURE ml6_write_par_block (task, start, size: WORD; VAR data_var);

C VOID ml6_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 **ml6_set_macro_interruptible** die Makrobefehle unterbrechbar geschaltet wurden.

6.3.5. Zugriff auf den Datenbereich einer Task

ml6_reset_r_pointer Setze Daten-Lesezeiger zurück

Pascal PROCEDURE ml6_reset_r_pointer (task: WORD);

C VOID ml6_reset_r_pointer (USHORT task);

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

ml6_move_r_pointer Verschiebe Daten-Lesezeiger

Pascal PROCEDURE ml6_move_r_pointer (task: WORD; offset: LONGINT);

C VOID ml6_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.

ml6_reset_w_pointer — Setze Daten-Schreibzeiger zurück

Pascal PROCEDURE ml6_reset_w_pointer (task: WORD);

C VOID ml6_reset_w_pointer (USHORT task);

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

ml6_move_w_pointer — Verschiebe Daten-Schreibzeiger

Pascal PROCEDURE ml6_move_w_pointer (task: WORD;
offset: LONGINT);

C VOID ml6_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.

6

ml6_read_data_byte
ml6_read_data_word
ml6_read_data_dword — Lies Datenbereich einer Task

Pascal FUNCTION ml6_read_data_byte (task: WORD): BYTE;

FUNCTION ml6_read_data_word (task: WORD): WORD;

PROCEDURE ml6_read_data_dword (task: WORD;
VAR data: LONGINT);

C UCHAR ml6_read_data_byte (USHORT task);

USHORT ml6_read_data_word (USHORT task);

VOID ml6_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.

Parameter *data*: Zeiger auf eine Variable, die das Ergebnis von **ml6_read_data_dword** aufnimmt.

ml6_read_data_block — Lies Block aus Datenbereich

Pascal
PROCEDURE ml6_read_data_block (task, size: WORD;
VAR data_var);

C
VOID ml6_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 **ml6_set_macro_interruptible** die Makrobefehle unterbrechbar geschaltet wurden.

ml6_read_data — Lies Block direkt aus Datenbereich

Pascal
PROCEDURE ml6_read_data (task: WORD, rel_ofs : LONGINT, size: WORD; VAR data_var);

C
VOID ml6_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 ml6_reset_r_pointer, ml6_move_r_pointer und ml6_read_data_block (schnellerer Zugriff).

Parameter
rel_ofs: Relative Startadresse des Blocks

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

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

ml6_write_data_byte
ml6_write_data_word
ml6_write_data_dword Schreibe Daten in den Datenbereich

Pascal PROCEDURE ml6_write_data_byte (task: WORD; data: BYTE);

PROCEDURE ml6_write_data_word (task: WORD; data: WORD);

PROCEDURE ml6_write_data_dword (task: WORD; data: LONGINT);

C VOID ml6_write_data_byte (USHORT task, byte data);

VOID ml6_write_data_word (USHORT task, USHORT data);

VOID ml6_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.

ml6_write_data_block Schreibe Block in den Datenbereich

Pascal PROCEDURE ml6_write_data_block (task, size: WORD; VAR data_var);

C VOID ml6_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. sizeof (*data_var*))

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 **ml6_set_macro_interruptible** die Makrobefehle unterbrechbar geschaltet wurden. !

ml6_write_data Schreibe Block direkt in den Datenbereich

Pascal PROCEDURE ml6_wr_data (task: WORD, rel_ofs : LONGINT, size: WORD; VAR data_var);

C VOID ml6_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 ml6_reset_w_pointer, ml6_move_w_pointer und ml6_write_data_block (schnellerer Zugriff).

Parameter *rel_ofs*: Relative Startadresse des Blocks

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

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

6.3.6. Datenpuffer

Das Betriebssystem der Multi-COM stellt ringförmig organisierte Datenpuffer zur Verfügung. Jeder Puffer erhält bei der Erzeugung eine eindeutige Nummer. Nur 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 anderen Task oder vom PC gelesen bzw. beschrieben werden kann. In einem solchen Konfliktfall 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 vom Betriebssystem verwalteten 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.

ml6_create_buffer — Erzeuge einen Datenpuffer

Pascal
FUNCTION ml6_create_buffer (task: WORD; strategy, align: BYTE; usage: WORD; VAR size: LONGINT; VAR handle: LONGINT): BYTE;

C
UCHAR ml6_create_buffer (USHORT task, UCHAR strategy, UCHAR 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 ML6BIB.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.

ml6_clear_buffer Lösche den Inhalt eines Datenpuffers

Pascal PROCEDURE ml6_clear_buffer (buffer: LONGINT);

C VOID ml6_clear_buffer (ULONG buffer);

Funktion Diese Prozedur löscht alle Daten in einem Datenpuffer.

ml6_get_buffer_status Ermittle den Status eines Datenpuffers

Pascal PROCEDURE ml6_get_buffer_status (buffer: LONGINT; VAR freesize: LONGINT; VAR usedsize: LONGINT);

C VOID ml6_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.

ml6_get_prev_buffer_status Ermittle den Status des zuletzt angesprochenen Datenpuffers

Pascal PROCEDURE ml6_get_prev_buffer_status (VAR freesize: WORD; VAR usedsize: WORD);

C VOID ml6_get_prev_buffer_status (USHORT* freesize, USHORT* usedsize);

Funktion Diese geschwindigkeitsoptimierte Prozedur ermittelt, wie viel Speicher im zuletzt vom PC angesprochenen Puffer noch frei ist und wie viel Speicher derzeit belegt ist. Ist einer der Parameter ffffh, so ist der aktuelle Wert größer oder gleich ffffh. Dieser muss dann mit Hilfe von **ml6_get_buffer_status** ermittelt werden.

Parameter siehe **ml6_get_buffer_status**

ml6_write_buffer_byte
ml6_write_buffer_word
ml6_write_buffer_dword Schreibe Daten in einen Puffer

Pascal

FUNCTION ml6_write_buffer_byte (buffer: LONGINT; data: BYTE): BYTE;

FUNCTION ml6_write_buffer_word (buffer: LONGINT; data: WORD): BYTE;

FUNCTION ml6_write_buffer_dword (buffer: LONGINT; data: LONGINT): BYTE;

C

UCHAR ml6_write_buffer_byte (ULONG buffer, UCHAR data);

UCHAR ml6_write_buffer_word (ULONG buffer, USHORT data);

UCHAR ml6_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.

ml6_set_buffer Schreibe Block in einen Datenpuffer

Pascal FUNCTION ml6_set_buffer (buffer: LONGINT; strategy: BYTE; VAR size: WORD; VAR data_var): BYTE;

C UCHAR ml6_set_buffer (ULONG buffer, UCHAR 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.

ml6_read_buffer_byte
ml6_read_buffer_word
ml6_read_buffer_dword — Lies Daten aus einem Puffer

Pascal

```
FUNCTION ml6_read_buffer_byte (buffer: LONGINT;
VAR data_var: BYTE): BYTE;

FUNCTION ml6_read_buffer_word (buffer: LONGINT;
VAR data_var: WORD): BYTE;

FUNCTION ml6_read_buffer_dword (buffer: LONGINT;
VAR data_var: LONGINT): BYTE;
```

C

```
UCHAR ml6_read_buffer_byte (ULONG buffer, UCHAR* data_var);

UCHAR ml6_read_buffer_word (ULONG buffer,
USHORT* data_var);

UCHAR ml6_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.

ml6_get_buffer Lies Datenblock aus einem Puffer

Pascal FUNCTION ml6_get_buffer (buffer: LONGINT; strategy: BYTE; VAR size: WORD; VAR data_var): BYTE;

C UCHAR ml6_get_buffer (ULONG buffer, UCHAR 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 gelesenen 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.

6.3.7. Zugriffe auf das RAM der Multi-COM

ml6_allocate_ram Reserviere Speicher auf der Multi-COM

Pascal PROCEDURE ml6_allocate_ram (task, usage: WORD; strategy, align: BYTE; VAR size: LONGINT; VAR adr: LONGINT);

C VOID ml6_allocate_ram (USHORT task, USHORT usage, UCHAR strategy, UCHAR align, ULONG* size, ULONG* adr);

Funktion Diese Prozedur reserviert Speicherplatz auf der Multi-COM Karte.

Parameter

task: Tasknummer, für die Speicher reserviert werden soll. Die Angabe dient zu Protokollzwecken auf der Multi-COM.

usage: Reserviert, immer = 0 setzen.

strategy: Reservierungsstrategie, mögliche Werte entsprechen denen des Parameters *strategy* in der Funktion **ml6_create_buffer** (siehe S. 6-28).

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.

ml6_get_free_ram_size — Ermittle den freien Speicher auf der Multi-COM

Pascal PROCEDURE ml6_get_free_ram_size (align: BYTE;
VAR size: LONGINT);

C VOID ml6_get_free_ram_size (UCHAR align, ULONG* size);

Funktion Diese Prozedur ermittelt die Größe des freien Speichers auf der Multi-COM 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.

ml6_set_ram_pointer — Setze Pointer für Speicherzugriff

Pascal PROCEDURE ml6_set_ram_pointer (adr: LONGINT);

C VOID ml6_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.

ml6_read_ram Lies Daten aus dem Speicher der Multi-COM

Pascal PROCEDURE ml6_read_ram (size: WORD; VAR data_var);

C VOID ml6_read_ram (USHORT size, VOID* data_var);

Funktion Die Prozedur liest einen Datenblock ab der Speicherstelle, auf die der mit RAM-Pointer zeigt. Der vom Betriebssystem der Multi-COM verwaltete Zeiger wird nach der Operation automatisch um die entsprechende Anzahl von Byte erhöht. Der RAM-Pointer wird mit **ml6_set_ram_pointer** gesetzt.

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

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

ml6_write_ram Schreibe Daten in Speicher der Multi-COM

Pascal PROCEDURE ml6_write_ram (size: WORD; VAR data_var);

C VOID ml6_write_ram (USHORT size, VOID* data_var);

Funktion Die Prozedur schreibt einen Datenblock ab der Speicherstelle, auf die der RAM-Pointer zeigt. Der vom Betriebssystem der Multi-COM verwaltete Zeiger wird nach der Operation automatisch um die entsprechende Anzahl von Byte erhöht. Der RAM-Pointer wird mit **ml6_set_ram_pointer** gesetzt.

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

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

ml6_read_memory — Lies Daten direkt aus dem Speicher der Multi-COM

Pascal PROCEDURE ml6_read_memory (start_addr: LONG; size: WORD; VAR data_var);

C void ml6_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.

ml6_write_memory — Schreibe Daten direkt in den Speicher der Multi-COM

Pascal PROCEDURE ml6_write_memory (start_addr: LONGINT; size: WORD; VAR data_var);

C void ml6_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.

6.3.8. Zugriffe auf die I/O-Ports der Multi-COM

ml6_in8_port
ml6_in16_port — Lies Daten von I/O-Port der Multi-COM

Pascal FUNCTION ml6_in8_port (port: WORD): BYTE;

FUNCTION ml6_in16_port (port: WORD): WORD;

C UCHAR ml6_in8_port (USHORT port);

USHORT ml6_in16_port (USHORT port);

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

Parameter *port*: Multi-COM I/O-Adresse.

ml6_out8_port
ml6_out16_port — Schreibe Daten an I/O-Port der Multi-COM

Pascal PROCEDURE ml6_out8_port (port: WORD; data: BYTE);

PROCEDURE ml6_out16_port (port: WORD; data: WORD);

C VOID ml6_out8_port (USHORT port, UCHAR data);

VOID ml6_out16_port (USHORT port, USHORT data);

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

Parameter *port*: Multi-COM I/O-Adresse.

data: Daten, die geschrieben werden sollen.

6.3.9. Befehle zur Systemsteuerung

ml6_set_macro_interruptible — Mache Makrobefehle un-/unterbrechbar

Pascal PROCEDURE ml6_set_macro_interruptible (on_off: BOOLEAN);

C VOID ml6_set_macro_interruptible (UCHAR on_off);

Funktion Die Prozedur dient dazu, Makrobefehle auf der Multi-COM Karte unterbrechbar oder ununterbrechbar zu machen. Wenn sie unterbrechbar sind, bewirkt ein Makrobefehl nur eine kurze Interrupt-Sperrung auf der Multi-COM 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).

ml6_cache_control — Multi-COM Cache-Kontrolle

Pascal PROCEDURE ml6_cache_control (mode: BYTE);

C VOID ml6_cache_control (UCHAR mode);

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

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

6.3.10. Zugriffe auf das EEPROM der Multi-COM Karte

ml6_read_eeprom_direct
ml6_read_eeprom_copy — Lies ein Wort aus dem EEPROM

Pascal
FUNCTION ml6_read_eeprom_copy (device, reladr: BYTE):WORD;
FUNCTION ml6_read_eeprom_direct(device, reladr: BYTE):WORD;

C
USHORT ml6_read_eeprom_copy (UCHAR device, UCHAR reladr);
USHORT ml6_read_eeprom_direct (UCHAR device, UCHAR reladr);

Funktion
Diese Funktionen lesen ein Wort direkt aus dem EEPROM bzw. aus dessen im RAM angelegter Kopie. Die Kopie wird beim Aktivieren eines Betriebssystems (also nach Aufruf von **ml6_reset**) automatisch erzeugt. Direkte Zugriffe sind langsamer und besonders in einer Echtzeitumgebung nicht zu empfehlen.

Parameter
device: Zu lesender Teil des EEPROMs:
- 0: Multi-COM Karte
- 1: Schnittstelle A und B (SCC1)
- 2: Schnittstelle C und D (SCC2)
- 3: Schnittstelle E und F (SCC3)

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

ml6_write_eeprom_direct
ml6_write_eeprom_copy Schreibe ein Wort ins EEPROM

Pascal

PROCEDURE ml6_write_eeprom_copy (device, reladr: BYTE; data: WORD);

PROCEDURE ml6_write_eeprom_direct (device, reladr: BYTE; data: WORD);

C

VOID ml6_write_eeprom_copy (UCHAR device, UCHAR reladr, USHORT data);

VOID ml6_write_eeprom_direct (UCHAR device, UCHAR reladr, USHORT data);

Funktion

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

Parameter

device, reladr: siehe **ml6_read_eeprom_direct / copy**

data: Daten, die geschrieben werden sollen.

6.3.11. Zugriffe auf die Echtzeituhr der Multi-COM

ml6_get_rtc_status — Lies Status der Uhr

Pascal FUNCTION ml6_get_rtc_status : BYTE;

C UCHAR ml6_get_rtc_status (VOID);

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

Bit	Bedeutung
0	1 = Uhr gestellt
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

ml6_set_rtc_mode — Setze Status der Uhr

Pascal PROCEDURE ml6_set_rtc_mode (mode : BYTE);

C VOID ml6_set_rtc_mode (UCHAR 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

ml6_get_rtc_date_code — Lies Datecode der Uhr

Pascal FUNCTION ml6_get_rtc_date_code : LONGINT;

C ulong ml6_get_rtc_date_code (void);

Funktion Diese Funktion liest das Datum der Echtzeituhr. Das Datum wird als Datecode übergeben (siehe Kapitel 6.4).

ml6_get_rtc_time_code — Lies Timecode der Uhr

Pascal FUNCTION ml6_get_rtc_time_code : LONGINT;

C ulong ml6_get_rtc_time_code (void);

Funktion Diese Funktion liest die Uhrzeit der Echtzeituhr. Die Uhrzeit wird als Timecode übergeben (siehe Kapitel 6.4).

ml6_get_time — Lies Uhrzeit

Pascal PROCEDURE ml6_get_time (VAR hour, min, sec: BYTE);

C void ml6_get_time (byte *hour, byte *min, byte *sec);

Funktion Die Prozedur liefert die aktuelle Uhrzeit der Multi-COM.

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.

ml6_get_date_and_time — Lies Datum und Uhrzeit

Pascal PROCEDURE ml6_get_date_and_time (VAR year, month, day, dow, hour, min, sec : BYTE);

C void ml6_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 Multi-COM. Das Jahr wird dabei 2 stellig (0 bis 99) übergeben.

Parameter *year*: Zeiger auf eine Variable, in die das aktuelle Jahr eingetragen wird (0 bis 99).

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 **ml6_get_time.**

ml6_set_date_and_time — Setze Datum und Uhrzeit

Pascal: PROCEDURE ml6_set_date_and_time (year, month, day, dow, hour, min, sec: BYTE);

C: void ml6_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 Multi-COM.

Parameter: siehe **ml6_get_date_and_time**

*Die Funktionen **ml6_get_date_and_time** und **ml6_set_date_and_time** verwenden für die Jahreszahl den Zahlenbereich von 0 bis 159. Wird diese Zahl als Offset auf das Jahr 1900 verwendet, kommt es im Jahr 2000 zu einem Überlauf (Jahr-2000 Problem).*
*Für Neuentwicklungen sollten die Funktionen **ml6_get_date_and_time_2000** und **ml6_set_date_and_time_2000** verwendet werden (s.u.).*

ml6_get_date_and_time_2000 Lies Datum und Uhrzeit

Pascal PROCEDURE ml6_get_date_and_time_2000 (VAR year, month, day, dow, hour, min, sec : BYTE);

C void ml6_get_date_and_time_2000 (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 Multi-COM (Jahr-2000 kompatibel). Die Angabe *year* wird als Offset auf das Jahr 1900 interpretiert (Wertebereich 0 bis 159).

Parameter *year*: Zeiger auf eine Variable, in die das aktuelle Jahr als Offset auf 1900 eingetragen wird (0 bis 159).

month, day, dow, hour, min, sec: siehe **ml6_get_time.**

ml6_set_date_and_time_2000 Setze Datum und Uhrzeit

Pascal PROCEDURE ml6_set_date_and_time_2000 (year, month, day, dow, hour, min, sec: BYTE);

C void ml6_set_date_and_time_2000 (byte year, byte month, byte day, byte dow, byte hour, byte min, byte sec);

Funktion Die Prozedur setzt die Echtzeituhr der Multi-COM (Jahr-2000 kompatibel). Die Angabe year wird als Offset auf das Jahr 1900 interpretiert (Wertebereich 0 bis 159).

Parameter siehe **ml6_get_date_and_time_2000.**

6.3.12. Steuerung der LEDs auf der Multi-COM

ml6_external_led_on
ml6_external_led_off
ml6_local_led_on
ml6_local_led_off — Schalte eine LED ein/aus

Pascal
PROCEDURE ml6_local_led_on;
PROCEDURE ml6_external_led_on;
PROCEDURE ml6_local_led_off;
PROCEDURE ml6_external_led_off;

C
VOID ml6_local_led_off (VOID);
VOID ml6_external_led_off (VOID);
VOID ml6_local_led_on (VOID);
VOID ml6_external_led_on (VOID);

Funktion
Diese Prozeduren schalten entweder die LED auf der Multi-COM Karte (local_led, LEDint) bzw. die extern an Stecker St3 angeschlossene LED (external_led, LEDext) ein (on) bzw. aus (off).

ml6_get_local_led_status — Lies den Zustand der LED

Pascal
FUNCTION ml6_get_local_led_status: BYTE;

C
UCHAR ml6_get_local_led_status (VOID);

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

6.3.13. Sonstige Funktionen

ml6_get_physical_address — Ermittle physikalische Adresse

Pascal	FUNCTION ml6_get_physical_address (seg_offs: LONGINT): LONGINT;
C	ulong ml6_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).

6.4. 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:

ml6_create_version_string — Versionscode in String umwandeln
ml6_create_date_string — Datecode in String umwandeln
ml6_create_time_string — Timecode in String umwandeln

Pascal

```
PROCEDURE ml6_create_version_string  (VAR v_string: STRING;
code: LONGINT);

PROCEDURE ml6_create_date_string  (VAR d_string: STRING;
code: LONGINT);

PROCEDURE ml6_create_time_string  (VAR t_string: STRING;
code: LONGINT);
```

C

```
void ml6_create_version_string (char* v_string, ulong code);

void ml6_create_date_string (char* d_string, ulong code);

void ml6_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

6.5. Fehlerbehandlung

6.5.1. 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 **ml6_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 **6.6**). Unter Windows 95 und Windows NT gibt es keine Einschränkungen.

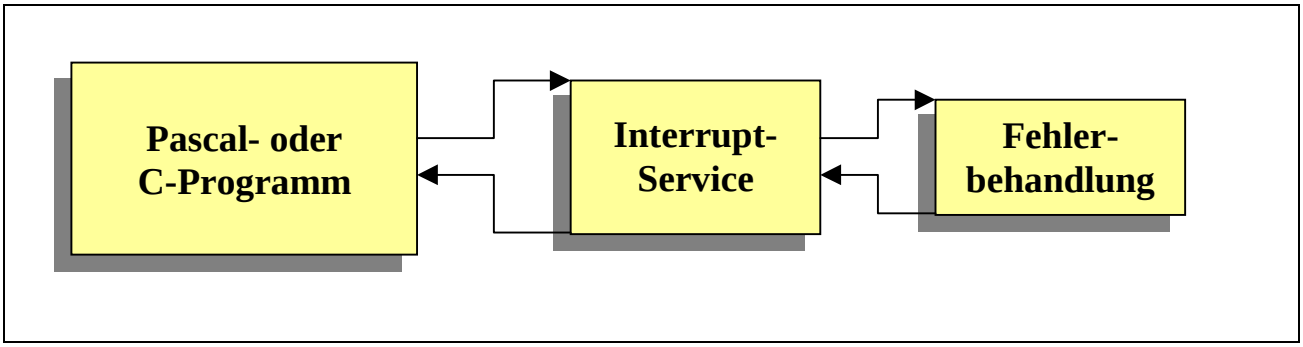


Abb. 6-1: Aufruf der Fehlerbehandlung in der Benutzer-Service-Routine

Die Fehlerbehandlung wird nicht aufgerufen, wenn mit **ml6_set_mark** der Wert Null eingestellt ist. Dies ist z. B. während der Ausführung von **ml6_reset** und **ml6_start** der Fall.

6.5.2. Prozeduren und Funktionen für die Fehlerbehandlung

ml6_set_error_handling — Setze die Adresse der Fehlerbehandlungsprozedur

Pascal PROCEDURE ml6_set_error_handling (serv_proc: POINTER);

C VOID ml6_set_error_handling (ERROR_PROC_P serv_proc);

Funktion Die Funktionen **ml6_reset** und **ml6_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.

ml6_get_error_info — Lies Fehlerinformation

Pascal PROCEDURE ml6_get_error_info
(VAR err_rec: ml6_error_info_type);

C VOID ml6_get_error_info (struct ml6_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 Multi-COM Karte und PC enthält.

Das Datenfeld **ml6_error_info_type** hat folgenden Aufbau:

Feldbezeichner	Typ	Beschreibung
ok	UCHAR	= FALSE: ein Fehler ist aufgetreten, sonst = TRUE. Solange diese Variable FALSE gesetzt ist, wird die Multi-COM Karte nicht mehr angesprochen (siehe **ml6_clear_error**).
repeatmacro	UCHAR	= TRUE: der fehlgeschlagene Makrobefehl soll wiederholt werden. *ok* muss ebenfalls auf TRUE gesetzt werden. Bei jeder Makrobefehlsausführung wird *repeatmacro* wieder FALSE gesetzt!
errorcode	UCHAR	Fehlercode (siehe Anhang E).
errorclass	UCHAR	Fehlerklasse (siehe Anhang E).
statusport	UCHAR	Wert des Status-Ports der Multi-COM - PC Schnittstelle (Basisadresse+0) unmittelbar nach Auftreten des Fehlers.
statusmap	UCHAR	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 **ml6_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).

Feldbezeichner	Typ	Beschreibung
mark	USHORT	Mit der Prozedur ml6_mark zuletzt gesetzter Markierungswert.
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 **ml6_write_ram, ml6_write_data_block** und **ml6_write_par_block** sind nicht in diesem Datenfeld sondern in *datawords* gespeichert.
inwords	USHORT [6]	Antwort (bis zu 12 Byte) der Multi-COM Karte (nur bei Fehlerklasse 0). Datenbyte der Makrobefehle **ml6_read_data_block, ml6_read_par_block** und **ml6_read_ram** sind nicht in diesem Datenfeld sondern in *datawords* gespeichert.
datawords	USHORT [6]	Bis zu 12 Byte der Datenfelder der Makrobefehle **ml6_write_data_block, ml6_read_data_block, ml6_write_par_block, ml6_read_par_block, ml6_write_ram** und **ml6_read_ram** (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 **ml6_set_max_retry** und **ml6_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 Multi-COM Karte zu schreiben, ohne dass vorher *ok* = TRUE gesetzt wurde. Diese Aufgabe wird normalerweise durch Aufruf von **ml6_clear_error** in der Fehlerbehandlungsroutine erledigt.
1	Wie Bit 0, jedoch gesetzt, wenn Daten von der Multi-COM 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 **ml6_reset** oder **ml6_start** benutzt werden.

Unter Windows 95 und Windows NT lautet die Deklaration dieser Datenstruktur anders. Bitte sehen Sie in der Header-Datei (ML6BIB.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}
  IF ml6_error_repeat_test THEN      {max. Anz. Wiederh. nicht erreicht?}
    BEGIN
      IF ml6_clear=0 THEN            {Schnittstelle bereinigen und}
                                     {Fehlerrecord löschen}
        ml6_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 (ml6_error_repeat_test() == 1)/*max. Anz. Wiederh. nicht erreicht?*/
     if (ml6_clear()==0)           /*Schnittstelle bereinigen und*/
                                   /*Fehlerrecord löschen*/
        ml6_set_repeat_macro(TRUE);/*Wiederholung veranlassen*/
  else                             /*max. Anz. Wiederholungen überschritten*/
    {
      /* ... Programm anhalten oder Fehler melden* /
    }
}
```

ml6_get_error_message — Lies die Fehler-Zeichenkette

Pascal PROCEDURE mlx_get_error_message (VAR errormessage: str80);

C VOID ml6_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.

ml6_get_error_msg — Lies die Fehler-Zeichenkette

Pascal PROCEDURE ml6_get_error_msg (VAR errstr: STRING; errclass, errcode: BYTE);

C VOID ml6_get_error_msg (CHAR* errstr, UCHAR errclass, UCHAR 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.

ML6ERRD.C	(Deutsch, C)
ML6ERRE.C	(Englisch, C)
MLXERRD.PAS	(Deutsch, Pascal)
MLXERRE.PAS	(Englisch, Pascal)

ml6_set_error_message — Setze die Fehler-Zeichenkette

Pascal PROCEDURE ml6_set_error_message (VAR errormsg: str80);

C VOID ml6_set_error_message (CHAR* errormsg);

Funktion Diese Prozedur setzt den Inhalt einer Zeichenkette, die in der Fehlerbehandlungsprozedur mit der Funktion **ml6_get_error_message** angefordert werden kann.

Parameter
- *errmsg*: Zeiger auf einen Variablenbereich, in dem die Fehlermeldung steht.

ml6_get_mark — Lies die aktuelle Markierungszahl

Pascal FUNCTION ml6_get_mark: WORD;

C USHORT ml6_get_mark (VOID);

Funktion Diese Funktion gibt den mit **ml6_set_mark** gesetzten Markierungswert zurück.

ml6_set_mark — Setze Programmarke für Fehlerbehandlung

Pascal PROCEDURE ml6_set_mark (mark: WORD);

C VOID ml6_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 **ml6_get_error_info** (siehe Seite 6-53) ermittelte Record enthält in der Variablen *mark* den zuletzt gesetzten Markierungswert. Er kann aber auch jederzeit mit **ml6_get_mark** ermittelt werden.

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

ml6_get_mark_message — Lies die Markierungs-Zeichenkette

Pascal PROCEDURE ml6_get_mark_message (VAR markmsg: str80);

C VOID ml6_get_mark_message (CHAR* markmsg);

Funktion Mit dieser Prozedur erhält man die Zeichenkette, die mit **ml6_set_mark_message** für jede Karte gesetzt werden kann. Diese Prozedur ergänzt **ml6_get_mark**.

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

ml6_set_mark_message Setze die Markierungs-Zeichenkette

Pascal PROCEDURE ml6_set_mark_message (VAR markmsg: str80);

C VOID ml6_set_mark_message (CHAR* markmsg);

Funktion Diese Prozedur setzt den Inhalt einer Zeichenkette, die in der Fehlerbehandlungsprozedur mit **ml6_get_mark_message** angefordert werden kann. Diese Prozedur ergänzt **ml6_set_mark**.

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

ml6_set_max_error_retry Setze zulässige Anzahl von Makrobefehlswiederholungen

Pascal PROCEDURE ml6_set_max_error_retry (value: WORD);

C VOID ml6_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 **ml6_error_repeat_test** als Abwärtszähler verwendet. Die Werte von *maxerrorretry* und *errorretry* können mit Prozedur **ml6_get_error_info** ermittelt werden.

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

ml6_error_repeat_test Dekrementiere Fehlerzähler

Pascal FUNCTION ml6_error_repeat_test: BOOLEAN;

C UCHAR ml6_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.

ml6_set_repeat_macro Erzwinge Makrobefehlswiederholung

Pascal PROCEDURE ml6_set_repeat_macro (rpm : BOOLEAN);

C VOID ml6_set_repeat_macro (UCHAR 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).

ml6_clear_error Lösche die Fehlerinformation

Pascal PROCEDURE ml6_clear_error;

C VOID ml6_clear_error (VOID);

Funktion Die Prozedur löscht die Fehlerinformation der Bibliothek über einen vorangegangenen Fehler. Die Schnittstelle zur Multi-COM 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 **ml6_get_error_info**).

ml6_clear Bereinige die Schnittstelle PC - Multi-COM

Pascal FUNCTION ml6_clear: WORD;

C USHORT ml6_clear (VOID);

Funktion Die Funktion entnimmt nicht abgeholte Zeichen aus der Schnittstelle und prüft, ob die Karte reagiert. Wenn **ml6_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 **ml6_clear** wird die Fehlerinformation der Bibliothek über einen vorangegangenen Fehler gelöscht. **ml6_clear** enthält einen Aufruf von **ml6_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. **ml6_clear** dient dem Freimachen der Schnittstelle für Diagnosezwecke. Nach dem Aufruf von **ml6_clear** sollte das Programm nicht ohne Klärung der Fehlerursache fortgesetzt werden.

ml6_call_user_error Rufe Fehlerbehandlung auf

Pascal PROCEDURE ml6_call_user_error;

C VOID ml6_call_user_error (VOID);

Funktion Die Prozedur ruft die benutzereigene Fehlerbehandlung auf, die mit **ml6_reset**, **ml6_start** oder **ml6_set_error_handling** gesetzt worden ist. Die Funktion ist für die Erstellung von Bibliotheken gedacht, die bei einem Fehler die Fehlerbehandlungsroutine der **ML6BIB** aufrufen sollen.

6.6. Request-Behandlung

Die PC-Bibliothek für die Multi-COM 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:

6.6.1. 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 Multi-COM Karte mit den dafür vorgesehenen Bibliotheksfunktionen sind ohne Einschränkung möglich.

Es ist auch möglich, Service-Requests von einer Multi-COM Karte ohne die Verwendung eines PC-Interrupt-Kanals zu handhaben. Hierzu wird bei der Initialisierung mit **ml6_start** bzw. **ml6_reset** in Parameter *mode* die Betriebsart ohne Interrupt eingestellt (*mode* = 0). Wenn nun ein Request von der Multi-COM 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 **ml6_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.

6.6.2. 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.

ml6_set_service — Setze die Adresse der Serviceprozedur

Pascal PROCEDURE ml6_set_service (servproc: POINTER);

C VOID ml6_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 = ml6_get_message();
   /*...*/              /* Anwenderdefinierte Request-Auswertung* /
}

/* Hauptprogramm* /

main()
{
   ...
   ml6_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. !

ml6_suspend_requests — Sperre die Schnittstelle

Pascal PROCEDURE ml6_suspend_requests;

C VOID ml6_suspend_requests (VOID);

Funktion Diese Prozedur unterbindet Service-Requests der Multi-COM 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 Hochsprachenprogramms können dagegen Abfolgen von Prozeduren wie **ml6_move_r_pointer** und **ml6_read_data_block** gegen eine Unterbrechung geschützt werden.

Beispiel (Pascal):

```
ml6_suspend_requests;
ml6_reset_r_pointer (task);
ml6_move_r_pointer (task, 100);
ml6_read_data_block (task, size, destination);
ml6_allow_requests;
```

ml6_allow_requests — Gib die Schnittstelle frei

Pascal PROCEDURE ml6_allow_requests;

C VOID ml6_allow_requests (VOID);

Funktion Die Schnittstelle wird für Requests der Multi-COM Karte wieder freigegeben.

ml6_get_message — Lies aktuelle Interrupt-Meldung

Pascal FUNCTION ml6_get_message: WORD;

C USHORT ml6_get_message (VOID);

Funktion Diese Funktion übergibt das von der Multi-COM 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.

ml6_message_available — Prüfe Interrupt-Status

Pascal FUNCTION ml6_message_available: BOOLEAN;

C UCHAR ml6_message_available (VOID);

Funktion Mit dieser Funktion kann geprüft werden, ob tatsächlich ein Request-Wort vom Treiber zwischengespeichert wurde. Wird die Funktion **ml6_get_message** zu Beginn der Benutzer-Service-Routine aufgerufen, so ist es nicht nötig, mit **ml6_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.

ml6_poll_request — Prüfe, ob ein Request vorliegt

Pascal FUNCTION ml6_poll_request: BOOLEAN;

C UCHAR ml6_poll_request (VOID);

Funktion Diese Funktion (nur für DOS und Windows 3.x verfügbar) prüft, ob ein Request der Multi-COM Karte ansteht. Sie wird nur im Betrieb ohne PC-Interrupt eingesetzt. Das Funktionsergebnis von **ml6_poll_request** ist TRUE, wenn ein Request ansteht, ansonsten ist das Funktionsergebnis FALSE.

Notizen

7. Echtzeitprogrammierung

7.1. Einführung

Wie alle SORCUS Karten besitzt die Multi-COM 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 der mitgelieferten CD finden Sie die Beispielprogramme als Quelle und als compilierte Dateien.

Im Kapitel 8 finden Sie eine Beschreibung zum Remote-Debuggen auf der Multi-COM Karte.

Im Kapitel 9 finden Sie eine Referenz aller Subroutinen der Echtzeitbibliothek ML6RTBIB für Pascal und C.

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

Die im folgenden dargestellten Programme sind auf einer Multi-COM 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

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 bekannt gemacht wird (s. 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	Min. Datenbereichsgröße in Byte, den das Prg. erfordert
28	Long	Max. Datenbereichsgröße in Byte, den das Prg. unterstützt
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 der Bibliothek ML6RTBIB 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 ML6RTBIB
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 Programms 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.pdt_head.usFlags := _LOCAL_PARAMETER
                        + ...;
```

C

```
struct parameter_type
{ TDT_TYPE   tdt;
  byte       status;
  word       blink_rate;
  byte       led_status;
  byte       led;
} parameter;
pdt.pdt_head.usFlags = _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

```
blinkrate := ml6rt_get_par_word(task,1)
led_status := ml6rt_get_par_byte(task,3)
```

C

```
blinkrate = ml6rt_get_par_word(task,1);
led_status = ml6rt_get_par_byte(task,3);
```

Beachten Sie die Nummerierung 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 (ml6rt_get_par_address) und mit Pointern arbeiten oder mit Hilfe von Systemaufrufen Parameter lesen (ml6rt_read_par_xxxx) und schreiben (ml6rt_ 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 SNW6 bei M6INST oder von der PC-Bibliothek bei ml6_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 oder Systemaufrufe und die Betriebssystemzeiger benutzen (ml6rt_reset_x_pointer, ml6rt_move_x_-pointer, ml6rt_read_data_xxxx, ml6rt_write_data_xxxx).

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.

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 (ML6BIB) und auf der Karte (ML6RTBIB) zur Verfügung.

7.3.5. Task-Deskriptor-Tabelle (TDT)

Bevor ein Programm gestartet werden kann, muss es auf die Multi-COM 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 der Bibliothek ML6RTBIB 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 Prozeduren
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 Multi-COM 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 Multi-COM Karte nur zum Zeitpunkt der Installierung einmal aufgerufen. Sie hat die Aufgabe, die PDT zu initialisieren und dem Betriebssystem bekannt zu machen (ml6rt_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 Programms kümmern, da die Compiler keine Informationen über das Betriebssystem der Multi-COM Karte haben, um die notwendigen Tabellen selbst anzulegen. Das erledigen Sie, wie schon erwähnt, im PREPARE-Teil des Programms. 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, 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 II-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 II-Task wird hier die Blinkfrequenz in Vielfachen von 1 ms angegeben.
3	Byte	Status der LED: 0 = LED ist aus 1 = LED ist an

Alle Programme benutzen die Echtzeitbibliothek "ML6RTBIB". Nähere Informationen zu dieser Bibliothek finden Sie in Kapitel 9 und in der Header-Datei "ML6RTBIB.H" bzw. in der Datei "ML6RTBIB.PAS".

Als Entwicklungsumgebung können Sie Borland-Pascal für DOS oder Borland C++ für DOS verwenden. Ob diese integrierten Entwicklungsumgebungen unter DOS, Windows 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 Multi-COM ablegen können, z.B. "C:\SORCUS\ML6\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 Multi-COM 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\ML6\RT\PASCAL\TPU70, 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 Multi-COM 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 ("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 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 Multi-COM DX 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 zeigt das Blink-LED Programm als NI-Task. Sie finden das komplette Programm im Sourcecode unter den Namen "M6P0300.PAS" auf den mitgelieferten Disketten.

```
{$N-}                                   {* Coprozessor aus                  *}
{$E-}                                   {* keine Emulation                  *}
{$R-}                                   {* kein Range-Check                 *}
{$S-}                                   {* kein Stack-Checking              *}
{$I-}                                   {* kein IO-Checking                 *}
{$V-}                                   {* Keine strikte Stringuebergabe    *}
{$L+}                                   {* Local-Symbols fuer Turbo-Debugger*}
{$F+}                                   {* FAR Aufrufe erzwingen!           *}
{$M 1024, 0, 0}                         {* Stack und Heap auf Minimum       *}

USES ml6rtbib;                          {* Echtzeitbibliothek einbinden     *}

                                        {* Allgemeine Angaben               *}
CONST
 PROG_NUMBER     : Word = $300;         {* Programmnummer = 300h            *}
 VERSION         : Char = '1';          {* Version des Programms            *}
 REVISION        : Char = 'A';          {* Revision des Programms           *}

                                        {* Statusmoeglichk. des Programms   *}
 READY           = 0;                   {* Bereit                           *}
 RUNNING         = 2;                   {* Programm laeuft                  *}
 ERROR           = 3;                   {* Falsche LED angewaehlt           *}
 STOPPED         = 4;                   {* Programm wurde angehalten        *}

                                        {* Zustaende der LED                *}
 ON              = 1;
 OFF             = 0;

VAR
   pdt         : RECORD                 {* Aufbau der PDT                   *}
                  pdt_head : ML6_PDT_HEAD;
                  main_proc: POINTER;   {* Adresse der Haupt-Prozedur       *}
                  auto_init: POINTER;   {* Adresse der Auto-Init-Prozedur   *}
                  start_proc: POINTER;  {* Adresse der Start-Prozedur       *}
                  stop_proc: POINTER;   {* Adresse der Stop-Prozedur        *}
                 END;
                                        {* Aufbau des Parameterbereichs     *}
   parameter   : RECORD
                  TDT       : TDT_TYPE;{* Platz fuer die TDT                *}
                  status    : Byte;     {* Status des Programms (READY, ...)*}
                  blink_rate: Word;     {* Blinkrate                        *}
                  led_status: Byte;     {* Zustand der LED                  *}
                  soft_cnt  : Word;     {* Soft-Counter                     *}
                 END;
```

7

```
PROCEDURE auto_init;                    {* AUTO-INITIALISIERUNG            *}
BEGIN
 ml6rt_entry;                           {* Register retten und vorbereiten  *}

 parameter.status := READY;             {* Status d. Programms auf "BEREIT" *}
 parameter.blink_rate := 50000;         {* Blinkrate auf Defaultwert        *}
 parameter.led_status := OFF;           {* Status der LED auf "AUS"         *}
 ml6rt_local_led_off;                   {* Die On-Board-LED ausschalten     *}

 ml6rt_exit;                            {* Register wieder restaurieren     *}
END;

PROCEDURE start;
BEGIN

 ml6rt_entry;
                                        {* Soft counter initialisieren      *}
 parameter.soft_cnt := parameter.blink_rate;
 parameter.status := RUNNING;

                                        {* Task aktivieren                  *}
 ml6rt_wakeup_task(parameter.tdt.task);

 ml6rt_exit;
END;

PROCEDURE stop;                         {* Task stoppen                     *}
BEGIN
 ml6rt_entry;
                                        {* Task deaktivieren                *}
 ml6rt_sleep_task(parameter.tdt.task);
 parameter.status := STOPPED;           {* Status auf "abgebrochen"         *}

 ml6rt_exit;
END;

PROCEDURE main_task;                    {* ---- HAUPTPROZEDUR DER TASK ---- *}
BEGIN
 ml6rt_entry;

 DEC(parameter.soft_cnt);               {* Soft-Counter dekrementieren      *}

 IF(parameter.soft_cnt = 0) THEN        {* Soft-Counter schon auf Null?     *}
   BEGIN                                {* Wenn ja, dann LED umschalten     *}
    IF(parameter.led_status = ON) THEN {* Falls die LED eingeschaltet ist, *}
      BEGIN
       ml6rt_local_led_off;
       parameter.led_status := OFF;     {* ... Zustand merken               *}
      END
    ELSE                                {* Falls die LED ausgeschaltet ist, *}
      BEGIN
       ml6rt_local_led_on;
       parameter.led_status := ON;      {* ... Zustand merken               *}
      END;
                                        {* Software-Teiler reinitialisieren *}
    parameter.soft_cnt := parameter.blink_rate;
   END;

 ml6rt_exit;
END;
```

```
BEGIN                                   {* PREPARE                          *}

                                        {* PTD-INITIALISIEREN               *}
 pdt.pdt_head.ucType := 1;              {* PDT-Typ einstellen               *}
                                        {* Laenge des PDT-Vorspanns         *}
 pdt.pdt_head.ucSize := sizeof(ML6_PDT_HEAD);
 pdt.pdt_head.usGlobalProc := 4;        {* Anzahl der Prozeduren            *}
 pdt.pdt_head.usProgNo := PROG_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 := _TP70;      {* Programmiersprache = Pascal      *}
 pdt.pdt_head.ucProgType := _USER_PRG;  {* Programm-Typ: Anwenderprogramm   *}
 pdt.pdt_head.usFlags    := 0;          {* Flags:                           *}
                                        {* Tasktyp wird durch PDT festgelegt*}
 pdt.pdt_head.usFlags    := _NI_TASK +  {* Tasktyp: NI-Task                 *}
                            _PDT_INFO_ENABLE +
                            _LOCAL_DATA +
                            _LOCAL_PARAMETER +
                            _FIXED_DATASIZE;

 pdt.pdt_head.usInt := 0;               {* Interrupt-Num. fuer das Programm *}
                                        {* Bei NI-Tasks ohne Bedeutung      *}
 pdt.pdt_head.ulDataAdr  := 0;          {* Adresse des Datenbereichs        *}
 pdt.pdt_head.ulDataSize := 0;          {* Groesse des Datenbereichs        *}
 pdt.pdt_head.ulDataSizeMin := 0;       {* Minimale Datenbereichsgroesse    *}
 pdt.pdt_head.ulDataSizeMax := 0;       {* Maximale Datenbereichsgroesse    *}
                                        {* Anfangsadresse Parameterbereich  *}
 ml6rt_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          *}
 pdt.main_proc  := @main_task;          {* Adresse der Hauptprozedur        *}
 pdt.auto_init  := @auto_init;          {* Adresse der Auto_Init Prozedur   *}
 pdt.start_proc := @start;              {* Adresse der Start Prozedur       *}
 pdt.stop_proc  := @stop;               {* Adresse der Stop Prozedur        *}

 ml6rt_set_pdt_adr(pdt);                {* Adresse der PDT bekanntgeben     *}
END.
```

Da das Programm insgesamt umfangreich kommentiert ist, soll jetzt nur noch auf Besonderheiten eingegangen werden.

Als erstes wird die Unit "ML6RTBIB" eingebunden. Diese Unit enthält verschiedene Routinen, die Ihnen die Programmierung erleichtern sollen. Eine ausführliche Beschreibung der einzelnen Routinen finden Sie im Kapitel 9 dieses Handbuchs.

Anschließend wird die PDT deklariert. Der Kopf der PDT ist bereits vordefiniert. 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 Be-

triebssystem 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 **ml6rt_entry** und enden mit **ml6rt_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.

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 (beachten Sie bitte die PDT-Flags) und dem Multi-COM Betriebssystem die Adresse der PDT mitgeteilt wird (ml6rt_set_pdt_adr).

7.6.5.2. Die Installierung der NI-Task

Nachdem das Programm "M6P0300.PAS" einwandfrei übersetzt und die Datei "M6P0300.EXE" erzeugt wurde, kann das Programm auf die Karte geladen und gestartet werden. Geben Sie dazu bitte folgende Installationsdatei mit Hilfe von SNW6 ein, und laden Sie diese Datei. Sie können auch die Beispielinstallationsdatei "NILED.INS" benutzen.

```
' Beispielinstallation: LED-Blinken (NI-Task)
' Mehrfachinst. muss abgeschaltet werden !
' (Naehere Informationen entnehmen Sie bitte dem Hand-
'  buch.)
'
' Karte anwaehlen und Reset ausloesen
M6DEVICE 0380 TIMEOUT=10 RESET
'
' Programm-Nummer 300h unter der Task-Nummer 20h in-
' stallieren (NI-Task)
' Auto-Init aufrufen und Task nicht aktivieren.
M6INST M6P0300.EXE 0300 0020 00 000000 0980

' Parameter der Task 20h setzen
' PAR-0   : Status des Programms
' PAR-1,2 : Blinkrate der LED    => Zaehlt die Aufrufe
' der Task mit
M6PAR 20 01 00 40
'     :  :  :  :
'     :  :  :  Parameter 2
'     :  :  Parameter 1
'     :  Parameteroffset
'     Tasknummer

' Aktivieren der Task durch Aufruf von Proz. 2
M6PROC 20 02
```

Diese Installationsdatei installiert die NI-Task, stellt die Blinkrate ein und startet das Programm durch Aufruf der Prozedur 2.

7.6.5.3. II-Task in Borland-Pascal

Das folgende Listing zeigt das Blink-LED Programm als II-Task. Dieses Programm finden Sie auch unter dem Namen "M6P0301.PAS" auf der mitgelieferten Diskette. Das Programm wird unter dem Interrupt des Timer-A installiert. Die Blinkrate kann in Vielfachen von 1 ms angegeben werden.

```
{$N-}                                   {* Coprozessor aus                  *}
{$E-}                                   {* keine Emulation                  *}
{$R-}                                   {* kein Range-Check                 *}
{$S-}                                   {* kein Stack-Checking              *}
{$I-}                                   {* kein IO-Checking                 *}
{$V-}                                   {* Keine strikte Stringuebergabe    *}
{$L+}                                   {* Local-Symbols fuer Turbo-Debugger*}
{$F+}                                   {* FAR Aufrufe erzwingen!           *}
{$M 1024, 0, 0}                         {* Stack und Heap auf Minimum       *}

USES ml6rtbib;                          {* Echtzeitbibliothek einbinden     *}

                                        {* Allgemeine Angaben               *}
CONST
 PROG_NUMBER     : Word = $301;         {* Programmnummer = 301h            *}
 VERSION         : Char = '1';          {* Version des Programms            *}
 REVISION        : Char = 'A';          {* Revision des Programms           *}

                                        {* Statusmoeglichk. des Programms   *}
 READY           = 0;                   {* Bereit                           *}
 RUNNING         = 2;                   {* Programm laeuft                  *}
 ERROR           = 3;                   {* Falsche LED angewaehlt           *}
 STOPPED         = 4;                   {* Programm wurde angehalten        *}

                                        {* Zustaende der LED                *}
 ON              = 1;
 OFF             = 0;

VAR
   pdt         : RECORD                 {* Aufbau der PDT                   *}
                  pdt_head : ML6_PDT_HEAD;
                  main_proc: POINTER;   {* Adresse der Haupt-Prozedur       *}
                  auto_init: POINTER;   {* Adresse der Auto-Init-Prozedur   *}
                  start_proc: POINTER;  {* Adresse der Start-Prozedur       *}
                  stop_proc: POINTER;   {* Adresse der Stop-Prozedur        *}
                 END;

                                        {* Aufbau des Parameterbereichs     *}
   parameter   : RECORD
                  TDT       : TDT_TYPE;{* Platz fuer die TDT                *}
                  status    : Byte;     {* Status des Programms (READY, ...)*}
                  blink_rate: Word;     {* Blinkrate                        *}
                  led_status: Byte;     {* Zustand der LED                  *}
                  soft_cnt  : Word;     {* Soft-Counter                     *}
                 END;
```

```
PROCEDURE auto_init;                    {* AUTO-INITIALISIERUNG            *}
BEGIN
 ml6rt_entry;                           {* Register retten und vorbereiten  *}
 parameter.status := READY;             {* Status d. Programms auf "BEREIT" *}
 parameter.blink_rate := 500;           {* Blinkrate auf Defaultwert 1 Hz   *}
 parameter.led_status := OFF;           {* Status der LED auf "AUS"         *}
 ml6rt_local_led_off;                   {* Die On-Board-LED ausschalten     *}
 ml6rt_exit;                            {* Register wieder restaurieren     *}
END;

PROCEDURE start;
VAR timer : Word;
BEGIN

 ml6rt_entry;
                                        {* Timer-Frequenz (1000 us = 1 kHz) *}
 timer := ml6rt_convert_timer_data(1000);
                                        {* Timer-A setzen                   *}
 ml6rt_set_timer(TIMER_A, timer, INT_MODE);

 parameter.soft_cnt := parameter.blink_rate;
 parameter.status := RUNNING;

                                        {* Task aktivieren                  *}
 ml6rt_wakeup_task(parameter.tdt.task);
 ml6rt_exit;
END;

PROCEDURE stop;                         {* Task stoppen                     *}
BEGIN
 ml6rt_entry;
                                        {* Task deaktivieren                *}
 ml6rt_sleep_task(parameter.tdt.task);
 parameter.status := STOPPED;           {* Status auf "abgebrochen"         *}
 ml6rt_exit;
END;

PROCEDURE main_task;                    {* ---- HAUPTPROZEDUR DER TASK ---- *}
BEGIN
 ml6rt_entry;

 DEC(parameter.soft_cnt);               {* Soft-Counter dekrementieren      *}

 IF(parameter.soft_cnt = 0) THEN        {* Soft-Counter schon auf Null?     *}
   BEGIN                                {* Wenn ja, dann LED umschalten     *}
    IF(parameter.led_status = ON) THEN {* Falls die LED eingeschaltet ist, *}
      BEGIN
       ml6rt_local_led_off;
       parameter.led_status := OFF;     {* ... Zustand merken               *}
      END
    ELSE                                {* Falls die LED ausgeschaltet ist, *}
      BEGIN
       ml6rt_local_led_on;
       parameter.led_status := ON;      {* ... Zustand merken               *}
      END;
                                        {* Software-Teiler reinitialisieren *}
    parameter.soft_cnt := parameter.blink_rate;
   END;

 ml6rt_exit;
END;
```

```
                                        {* PREPARE                          *}
BEGIN
                                        {* PTD-INITIALISIEREN               *}
 pdt.pdt_head.ucType := 1;              {* PDT-Typ einstellen               *}
                                        {* Laenge des PDT-Vorspanns         *}
 pdt.pdt_head.ucSize := sizeof(ML6_PDT_HEAD);
 pdt.pdt_head.usGlobalProc := 4;        {* Anzahl der Prozeduren            *}
 pdt.pdt_head.usProgNo := PROG_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 := _TP70;      {* Programmiersprache = Pascal      *}
 pdt.pdt_head.ucProgType := _USER_PRG; {* Programm-Typ: Anwenderprogramm   *}
 pdt.pdt_head.usFlags := 0;             {* Flags, Tasktyp wird beim         *}
 pdt.pdt_head.usFlags := _LOCAL_DATA + {* Installieren angegeben !         *}
                         _FIXED_DATASIZE +
                         _LOCAL_PARAMETER;

 pdt.pdt_head.usInt := IRQ_TIMER_A;     {* Interrupt-Num. fuer das Programm *}

 pdt.pdt_head.ulDataAdr  := 0;          {* Adresse des Datenbereichs        *}
 pdt.pdt_head.ulDataSize := 0;          {* Groesse des Datenbereichs        *}
 pdt.pdt_head.ulDataSizeMin := 0;       {* Minimale Datenbereichsgroesse    *}
 pdt.pdt_head.ulDataSizeMax := 0;       {* Maximale Datenbereichsgroesse    *}
                                        {* Anfangsadresse Parameterbereich  *}
 ml6rt_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          *}
 pdt.main_proc  := @main_task;          {* Adresse der Hauptprozedur        *}
 pdt.auto_init  := @auto_init;          {* Adresse der Auto_Init Prozedur   *}
 pdt.start_proc := @start;              {* Adresse der Start Prozedur       *}
 pdt.stop_proc  := @stop;               {* Adresse der Stop Prozedur        *}

 ml6rt_set_pdt_adr(pdt);                {* Adresse der PDT bekanntgeben     *}
END.
```

Da auch hier das Listing ausführlich kommentiert ist, soll nur auf Besonderheiten eingegangen werden.

Die Start-Prozedur setzt den Timer auf einen festen Takt von 1 kHz. Damit wird die Hauptprozedur der Task alle 1 ms aufgerufen. Der Hardware-Timer wird durch einen Softwarezähler erweitert. Erst wenn dieser Softwarezähler (zählt die Anzahl der Hauptprozeduraufrufe) 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 einer II-Task unterscheidet sich nicht von der einer NI-Task, obwohl es sich praktisch um eine Interrupt-Service Routine (ISR) handelt. Das OsX-Betriebssystem sorgt für die ordnungsgemäße Beendigung der ISR. Bei DI-Tasks muss dies vom Anwender durch entsprechende Programmierung durchgeführt werden: Es15 muss ein EOI (End Of Interrupt) zum Interrupt-Controller gesendet wer-

den (mit **ml6rt_end_of_int**). Die Hauptprozedur muß mit **ml6rt_exit_interrupt** beendet werden

*Die Hauptroutine darf nicht als '**interrupt**'-Prozedur deklariert werden. Das durch die **interrupt**-Anweisung bewirkte Retten der Register wird mit **ml6rt_entry** erledigt, das Beenden der Prozedur mit IRET mit **ml6rt_exit_interrupt** (bei DI-Tasks).* **!**

7.6.5.4. Installieren der II-Task

Nachdem das Programm "M6P0301.PAS" einwandfrei übersetzt und die Datei "M6P0301.EXE" erzeugt wurde, kann das Programm auf die Karte geladen und gestartet werden. Geben Sie dazu bitte folgende Installationsdatei mit Hilfe von SNW6 ein und laden Sie diese Datei. Sie können auch die Beispielinstallationsdatei "IILED.INS" benutzen.

```
' Beispielinstallation: LED-Blinken mit Timer-Interrupt
' (Timer-A)
' Mehrfachinst. muss abgeschaltet werden !
' (Naehere Informationen entnehmen Sie bitte dem Hand-
' buch.)
'
' Karte anwaehlen und Reset ausloesen
M6DEVICE 0380 TIMEOUT=10 RESET
'
' Programm-Nummer 301h unter der Task-Nummer 21h in-
' stallieren (II-Task)
' (Interrupt 91h => TIMER-A)
' Auto-Init aufrufen und Task nicht aktivieren.
M6INST M6P0301.EXE 0301 0021 91 000000 0989

' Task durch Aufruf von Proz. 2 starten:
M6PROC 21 02
```

Die Installationsdatei installiert die II-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.

Das Programm kann ohne Modifikationen auch unter einem anderen Interrupt oder als NI-Task installiert werden. Dazu muss lediglich die M6INST-Zeile in der INS-Datei geändert werden.

Für ein LED-Blinkprogramm bietet sich auch eine TI-Task an. Dazu muss lediglich die Aktivierungsroutine **ml6rt_wakeup_task** in der Start-Prozedur durch **ml6rt_wakeup_ti_task** ersetzt werden. Das Setzen des Timers kann entfallen.

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 Multi-COM ablegen können. Zum Beispiel C:\SORCUS\ML6\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 Multi-COM 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 Multi-COM Start-Up-Codes:
C:\SORCUS\ML6\RT\BC\BC45

Der Eintrag in "Library Directories" müsste dann wie folgt aussehen:
C:\SORCUS\ML6\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.

Diese Einstellung gilt nur für die Entwicklung von Programmen, die auf der Multi-COM 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 Echtzeitprogramms 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 Multi-COM 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* ***ml6rt_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), Weitere 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

7.7.5. Beispielprogramme für C++

7.7.5.1. NI-Task in C++

Das folgende Beispielprogramm zeigt das Blink-LED Programm als NI-Task. Sie finden dieses Programm auch unter dem Namen "M6P0300.C" auf der mitgelieferten Diskette. Zum Compilieren benutzen Sie bitte die ebenfalls auf der Diskette vorhanden Projektdatei "M6P0300.PRJ" (bzw. M6P0300.IDE). Bitte überprüfen Sie vor dem Compilieren die eingestellten Pfade ("Options/Directories").

```
#pragma option -r-                      /* Keine Registervariablen!!        */
#pragma option -k                       /* Standard Stack-Frame             */

#include "ml6rtbib.h"                   /* Echtzeitbibliothek einbinden     */
#include "dos.h"

                                        /* Allgemeine Angaben               */
#define PROG_NUMBER     0x300           /* Programmnummer = 300h            */
#define VERSION         '1'             /* Version des Programms            */
#define REVISION        'A'             /* Revision des Programms           */

                                        /* Statusmoeglichk. des Programms   */
#define READY           0               /* Bereit                           */
#define RUNNING         2               /* Programm laeuft                  */
#define ERROR           3               /* Falsche LED angewaehlt           */
#define STOPPED         4               /* Programm wurde angehalten        */

                                        /* Zustaende der LED                */
#define  ON             1
#define  OFF            0

typedef struct                          /* Aufbau der PDT                   */
{
 ML6_PDT_HEAD    pdt_head;
 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_type;


struct parameter_type                   /* Aufbau des Parameterbereichs     */
{
 TDT_TYPE              tdt;             /* Platz fuer die TDT               */
 unsigned char         status;          /* Status des Programms (READY, ...)*/
 unsigned int          blink_rate;      /* Blinkrate                        */
 unsigned char         led_status;      /* Zustand der LED                  */
 unsigned int          soft_cnt;        /* Soft-Counter                     */
} parameter;


pdt_type pdt;

void PFAR auto_init(void);              /* Prototypen der Prozeduren        */
void PFAR main_task(void);
void PFAR start(void);
void PFAR stop(void);
```

```
void PFAR auto_init(void)              /* AUTO-INITIALISIERUNG            */
{
 ml6rt_entry();                        /* Register retten und vorbereiten  */

 parameter.status = READY;             /* Status d. Programms auf "BEREIT" */
 parameter.blink_rate = 50000;         /* Blinkrate                        */
 parameter.led_status = OFF;           /* Status der LED auf "AUS"         */
 ml6rt_local_led_off();                /* Die On-Board-LED ausschalten     */

 ml6rt_exit();                         /* Register wieder restaurieren     */
}

void PFAR start(void)
{
 ml6rt_entry();
                                       /* Soft counter initialisieren      */
 parameter.soft_cnt = parameter.blink_rate;
 parameter.status = RUNNING;
                                       /* Task aktivieren                  */
 ml6rt_wakeup_task(parameter.tdt.task);

 ml6rt_exit();
}

void PFAR stop(void)                   /* Task stoppen                     */
{
 ml6rt_entry();
                                       /* Task deaktivieren                */
 ml6rt_sleep_task(parameter.tdt.task);
 parameter.status = STOPPED;           /* Status auf "abgebrochen"         */

 ml6rt_exit();
}


void PFAR main_task(void)              /* ---- HAUPTPROZEDUR DER TASK ---- */
{
 ml6rt_entry();

 if(parameter.soft_cnt-- == 0)         /* Soft-Counter schon auf Null?     */
   {                                   /* Wenn ja, dann LED umschalten     */
    if(parameter.led_status == ON)     /* Falls die LED eingeschaltet ist, */
      {
       ml6rt_local_led_off();
       parameter.led_status = OFF;     /* ... Zustand merken               */
      }
    else                               /* Falls die LED ausgeschaltet ist, */
      {
       ml6rt_local_led_on();
       parameter.led_status = ON;      /* ... Zustand merken               */
      }
                                       /* Software-Teiler reinitialisieren */
    parameter.soft_cnt = parameter.blink_rate;
   }

 ml6rt_exit();
}
```

```
void main (void)                        /* PREPARE                          */
{                                       /* PTD-INITIALISIEREN               */
 pdt.pdt_head.ucType = 1;               /* PDT-Typ einstellen               */
                                        /* Laenge des PDT-Vorspanns         */
 pdt.pdt_head.ucSize = sizeof(ML6_PDT_HEAD);
 pdt.pdt_head.usGlobalProc = 4;         /* Anzahl der Prozeduren            */
 pdt.pdt_head.usProgNo   = PROG_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                            */
                                        /* Tasktyp wird durch PDT festgelegt*/
 pdt.pdt_head.usFlags    = _NI_TASK +   /* Tasktyp: NI-Task                 */
                           _PDT_INFO_ENABLE +
                           _LOCAL_DATA +
                           _LOCAL_PARAMETER +
                           _FIXED_DATASIZE;

 pdt.pdt_head.usInt = 0;                /* Interrupt-Num. fuer das Programm */
                                        /* Bei NI-Tasks ohne Bedeutung      */

 pdt.pdt_head.ulDataAdr  = 0;           /* Adresse des Datenbereichs        */
 pdt.pdt_head.ulDataSize = 0;           /* Groesse des Datenbereichs        */
 pdt.pdt_head.ulDataSizeMin = 0;        /* Minimale Datenbereichsgroesse    */
 pdt.pdt_head.ulDataSizeMax = 0;        /* Maximale Datenbereichsgroesse    */
                                        /* Anfangsadresse Parameterbereich  */
 ml6rt_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          */
 pdt.main_proc  = main_task;            /* Adresse der Hauptprozedur        */
 pdt.auto_init  = auto_init;            /* Adresse der Auto_Init Prozedur   */
 pdt.start_proc = start;                /* Adresse der Start Prozedur       */
 pdt.stop_proc  = stop;                 /* Adresse der Stop Prozedur        */

 ml6rt_set_pdt_adr(&pdt);               /* Adresse der PDT bekanntgeben     */
}
```

Da das Programm insgesamt ausführlich kommentiert ist, soll nur auf Besonderheiten eingegangen werden.

Um dieses Programm zu compilieren, öffnen Sie bitte die Projektdatei "M6P0300.PRJ" (bzw. M6P0300.IDE). Beachten Sie bitte, dass die Datei "ML6RTBIB.LIB" in Ihre Projektdatei eingebunden werden muss. Nähere Informationen zu dieser Bibliothek finden Sie im Kapitel 9 dieses Handbuchs, wo alle Routinen der Bibliothek "ML6RTBIB" ausführlich erläutert sind.

Zuerst wird der Aufbau der PDT als Struktur deklariert. Der Kopf der PDT ist in der Headerdatei ML6RTBIB.H bereits vordefiniert. 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 **ml6rt_entry** und enden mit **ml6rt_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.

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 (beachten Sie bitte die Einstellung der PDT-Flags) und dem Multi-COM Betriebssystem die Adresse der PDT mitgeteilt wird (ml6rt_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 "M6P0300.PRJ" einwandfrei übersetzt und die Datei "M6P0300.EXE" erzeugt wurde, kann das Programm auf die Karte geladen und gestartet werden. Geben Sie dazu bitte folgende Installationsdatei mit Hilfe von SNW6 ein, und laden Sie diese Datei. Sie können auch die Beispielinstallationsdatei "NILED.INS" benutzen.

```
' Beispielinstallation: LED-Blinken (NI-Task)
' Mehrfachinst. muss abgeschaltet werden !
' (Naehere Informationen entnehmen Sie bitte dem Hand-
'  buch.)
'
' Karte anwaehlen und Reset ausloesen
M6DEVICE 0380 TIMEOUT=10 RESET
'
' Programm-Nummer 300h unter der Task-Nummer 20h in-
stallieren (NI-Task)
' Auto-Init aufrufen und Task nicht aktivieren.
M6INST M6P0300.EXE 0300 0020 00 000000 0980

' Parameter der Task 20h setzen
```

```
' PAR-0   : Status des Programms
' PAR-1,2 : Blinkrate der LED => Zaehlt die Aufrufe der

' Task mit
M6PAR 20 01 00 40
'     :  :  :  :
'     :  :  :  Parameter 2
'     :  :  Parameter 1
'     :  Parameteroffset
'     Tasknummer

' Aktivieren der Task durch Aufruf von Proz. 2
M6PROC 20 02
```

Diese Installationsdatei installiert die NI-Task, stellt die Blinkrate ein und aktiviert die Task durch Aufruf der Prozedur 2.

7.7.5.3. II-Task in C++

Das folgende Listing zeigt das Blink-LED Programm als II-Task. Sie finden das Programm auch unter dem Namen "M6P0301.C" auf der mitgelieferten Diskette. Zum Compilieren benutzen Sie bitte die Projektdatei "M6P0301.PRJ" (bzw. M6P0301.IDE), die sich ebenfalls auf der Diskette befindet. Das Programm soll unter dem Interrupt von Timer-A installiert werden. Die Blinkrate soll in vielfachen von 1 ms angegeben werden.

```
#pragma option -r-                      /* Keine Registervariablen!!        */
#pragma option -k                       /* Standard Stack-Frame             */

#include "ml6rtbib.h"                   /* Echtzeitbibliothek einbinden     */
#include "dos.h"
                                        /* Allgemeine Angaben               */
#define PROG_NUMBER     0x301           /* Programmnummer = 301h            */
#define VERSION         '1'             /* Version des Programms            */
#define REVISION        'A'             /* Revision des Programms           */

                                        /* Statusmoeglichk. des Programms   */
#define READY           0               /* Bereit                           */
#define RUNNING         2               /* Programm laeuft                  */
#define ERROR           3               /* Falsche LED angewaehlt           */
#define STOPPED         4               /* Programm wurde angehalten        */

                                        /* Zustaende der LED                */
#define  ON             1
#define  OFF            0

typedef struct                          /* Aufbau der PDT                   */
{
 ML6_PDT_HEAD        pdt_head;
 void* PFAR          main_proc;         /* Adresse der Haupt-Prozedur       */
 void* PFAR          auto_init;         /* Adresse der Auto-Init-Prozedur   */
```

7

```
 void* PFAR          start_proc;       /* Adresse der Start-Prozedur       */
 void* PFAR          stop_proc;        /* Adresse der Stop-Prozedur        */
} pdt_type;

struct parameter_type                  /* Aufbau des Parameterbereichs     */
{
 TDT_TYPE              tdt;            /* Platz fuer die TDT               */
 unsigned long         status;         /* Status des Programms (READY, ...)*/
 unsigned int          blink_rate;     /* Blinkrate (in x10ms)             */
 unsigned char         led_status;     /* Zustand der LED                  */
 unsigned int          soft_cnt;       /* Soft-Counter                     */
} parameter;
pdt_type pdt;

void PFAR auto_init(void);             /* Prototypen der Prozeduren        */
void PFAR main_task(void);
void PFAR start(void);
void PFAR stop(void);

void PFAR auto_init(void)              /* AUTO-INITIALISIERUNG             */
{
 ml6rt_entry();                        /* Register retten und vorbereiten  */

 parameter.status = READY;             /* Status d. Programms auf "BEREIT" */
 parameter.blink_rate = 500;           /* Blinkrate auf Defaultwert 1 Hz   */
 parameter.led_status = OFF;           /* Status der LED auf "AUS"         */
 ml6rt_local_led_off();                /* Die On-Board-LED ausschalten     */

 ml6rt_exit();                         /* Register wieder restaurieren     */
}


void PFAR start(void)
{
 unsigned short timer;

 ml6rt_entry();

                                       /* Timer-Frequenz (1000 us = 1 kHz) */
 timer = ml6rt_convert_timer_data(1000L);
                                       /* Timer-A setzen                   */
 ml6rt_set_timer(TIMER_A, timer, INT_MODE);

 parameter.soft_cnt = parameter.blink_rate;
 parameter.status = RUNNING;

                                       /* Task aktivieren                  */
 ml6rt_wakeup_task(parameter.tdt.task);

 ml6rt_exit();
}


void PFAR stop(void)                   /* Task stoppen                     */
{
 ml6rt_entry();
                                       /* Task deaktivieren                */
 ml6rt_sleep_task(parameter.tdt.task);
 parameter.status = STOPPED;           /* Status auf "abgebrochen"         */

 ml6rt_exit();
}
```

```
void PFAR main_task(void)              /* ---- HAUPTPROZEDUR DER TASK ---- */
{
 ml6rt_entry();

 if(parameter.soft_cnt-- == 0)         /* Soft-Counter schon auf Null?     */
   {                                   /* Wenn ja, dann LED umschalten     */
    if(parameter.led_status == ON)     /* Falls die LED eingeschaltet ist, */
      {
       ml6rt_local_led_off();
       parameter.led_status = OFF;     /* ... Zustand merken               */
      }
    else                               /* Falls die LED ausgeschaltet ist, */
      {
       ml6rt_local_led_on();
       parameter.led_status = ON;      /* ... Zustand merken               */
      }
                                       /* Software-Teiler reinitialisieren */
    parameter.soft_cnt = parameter.blink_rate;
   }

 ml6rt_exit();
}

void main (void)                       /* PREPARE                          */
{
                                       /* PTD-INITIALISIEREN               */
 pdt.pdt_head.ucType = 1;              /* PDT-Typ einstellen               */
                                       /* Laenge des PDT-Vorspanns         */
 pdt.pdt_head.ucSize = sizeof(ML6_PDT_HEAD);
 pdt.pdt_head.usGlobalProc = 4;        /* Anzahl der Prozeduren            */
 pdt.pdt_head.usProgNo   = PROG_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, Tasktyp wird beim         */
 pdt.pdt_head.usFlags = _LOCAL_DATA +  /* Installieren angegeben !         */
                        _FIXED_DATASIZE +
                        _LOCAL_PARAMETER;

 pdt.pdt_head.usInt = IRQ_TIMER_A;     /* Interrupt-Num. fuer das Programm */

 pdt.pdt_head.ulDataAdr  = 0;          /* Adresse des Datenbereichs        */
 pdt.pdt_head.ulDataSize = 0;          /* Groesse des Datenbereichs        */
 pdt.pdt_head.ulDataSizeMin = 0;       /* Minimale Datenbereichsgroesse    */
 pdt.pdt_head.ulDataSizeMax = 0;       /* Maximale Datenbereichsgroesse    */
                                       /* Anfangsadresse Parameterbereich  */
 ml6rt_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          */
 pdt.main_proc  = main_task;           /* Adresse der Hauptprozedur        */
 pdt.auto_init  = auto_init;           /* Adresse der Auto_Init Prozedur   */
 pdt.start_proc = start;               /* Adresse der Start Prozedur       */
 pdt.stop_proc  = stop;                /* Adresse der Stop Prozedur        */

 ml6rt_set_pdt_adr(&pdt);              /* Adresse der PDT bekanntgeben     */
}
```

Da das Listing ausführlich dokumentiert ist, soll nur auf die Besonderheiten eingegangen werden. Zum Compilieren benutzen Sie bitte die Projektdatei "M6P0301.PRJ" (bzw. M6P0301.IDE).

In der PDT werden Sie erkennen, daß das Programm vier Prozeduren enthält (Main-, Auto-Init-, Start- und die Stop-Prozedur).

Die Start-Prozedur setzt den Timer auf einen festen Takt von 1 kHz. Damit wird die Hauptprozedur der Task alle 1 ms aufgerufen. Der Hardware-Timer wird durch einen Softwarezähler erweitert. Erst wenn dieser Softwarezähler (zählt die Anzahl der Hauptprozeduraufrufe) 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 einer II-Task unterscheidet sich nicht von der einer NI-Task, obwohl es sich praktisch um eine Interrupt-Service Routine (ISR) handelt. Das OsX-Betriebssystem sorgt für die ordnungsgemäße Beendigung der ISR. Bei DI-Tasks muss dies vom Anwender durch entsprechende Programmierung durchgeführt werden: Es muss ein EOI (End Of Interrupt) zum Interrupt-Controller gesendet werden (mit **ml6rt_end_of_int**). Die Hauptprozedur muss mit **ml6rt_exit_interrupt** beendet werden

*Die Hauptroutine darf nicht als **'interrupt'**-Prozedur deklariert werden. Das durch die **interrupt**-Anweisung bewirkte Retten der Register wird mit **ml6rt_entry** erledigt, das Beenden der Prozedur mit IRET mit **ml6rt_exit_interrupt** (bei DI-Tasks).*

Für ein LED-Blinkprogramm bietet sich auch eine TI-Task an. Dazu muss lediglich die Aktivierungsroutine **ml6rt_wakeup_task** in der Start-Prozedur durch **ml6rt_wakeup_ti_task** ersetzt werden. Das Setzen des Timers kann entfallen.

7.7.5.4. Installierung der II-Task

Nachdem das Projekt "M6P0301.PRJ" einwandfrei übersetzt und die Datei "M6P0301.EXE" erzeugt wurde, kann das Programm auf die Karte geladen und gestartet werden. Geben Sie dazu bitte folgende Installationsdatei mit Hilfe von SNW6 ein, und laden Sie diese *.INS Datei. Sie können auch die Beispielinstallationsdatei "IILED.INS" benutzen.

```
' Beispielinstallation: LED-Blinken mit Timer-Interrupt (Timer-A)
' Mehrfachinst. muss abgeschaltet werden !
' (Naehere Informationen entnehmen Sie bitte dem Handbuch.)
'
' Karte anwaehlen und Reset ausloesen
M6DEVICE 0380 TIMEOUT=10 RESET
'
' Programm-Nummer 301h unter der Task-Nummer 21h installieren
' (II-Task)
' (Interrupt 91h => TIMER-A)
' Auto-Init aufrufen und Task nicht aktivieren.
M6INST M6P0301.EXE 0301 0021 91 000000 0989

' Task durch Aufruf von Proz. 2 starten:
M6PROC 21 02
```

Die Installationsdatei installiert die II-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.

Das Programm kann ohne Modifikationen auch unter einem anderen Interrupt oder als NI-Task installiert werden. Dazu muss lediglich die M6INST-Zeile in der INS-Datei geändert werden.

7.7.6. Die Makrobibliothek "ML6MACRO.H"

Bislang wurden alle Funktionen aus der Bibliothek "ML6RTBIB" ganz normal aufgerufen. Für die wichtigsten Funktionen (wie z.B. **ml6rt_entry**, **ml6rt_exit**) gibt es jedoch einen Weg, diese Methode zu optimieren. Die Header-Datei "ML6MACRO.H" stellt 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 "ML6RTBIB" einbinden. Die Makros haben die gleichen Namen wie die jeweiligen Funktionen. Es ist lediglich ein '_' vorangestellt (z.B. **_ml6rt_entry**). Vor der Verwendung der Makro-Bibliothek 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. 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.

7.8.2. Lesen von Parametern und Daten (Datenaustausch zwischen Tasks)

Die Routinen innerhalb der ML6RTBIB 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.

7.8.3. 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.

7.8.4. 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 Multi-COM 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 Multi-COM 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.

7.8.5. 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 (ml6rt_write_data_xxxx). Je nach Installierung der Task gibt es eine oder zwei Möglichkeiten, die Tasknummer zu bestimmen.

Wenn der Parameterbereich Teil des Programms 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 ml6rtbib 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 **ml6rt_entry** die Funktion **ml6rt_entry_task**, die die Tasknummer als Funktionsergebnis zurückliefert (nicht bei DI-Tasks in der Hauptprozedur einsetzbar).

7.8.6. 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 Bibliothek ML6RTBIB stellt dafür zwei Routinen zur Verfügung (ml6rt_store_80487 und ml6rt_restore_80487).

In der Programmiersprache C gilt zusätzlich die Einschränkung, dass innerhalb von Prozeduren, die mit ml6rt_entry oder ml6rt_entry_func beginnen, keine Fließkommaoperationen erlaubt sind. Statt dessen müssen die Berechnungen in einer eigenen (lokalen) Prozedur durchgeführt werden, die nicht mit ml6rt_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)
{
  ml6rt_entry();
  ...
  fp_calculations();
  ...
  ml6rt_exit();
}
```

Beachten Sie, dass bereits die Zuweisung einer Fließkommazahl eine Fließkommaoperation ist, für die alle oben beschriebenen Einschränkungen gelten.

7.8.7. 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.

7.8.8. Mehrfachinstallation von Echtzeitprogrammen

Die mehrfache Installation eines Programms 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 Echtzeitprogramms 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. ml6_read_par_byte oder ml6_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 **ml6rt_entry_task** (statt ml6rt_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 ml6_transfer_and_install geschehen, sondern erfordert zwei Schritte:

1. Laden des Programmcodes in das RAM der Karte mit der Funktion **ml6_transfer_pgm**. Die Funktion liefert die physikalische Prepare-Adresse und die Installations-Flags zurück.
2. Anschließend können mit der Funktion **ml6_install_task** unter Angabe der Tasknummer mehrere Instanzen des Programms erzeugt werden. Dazu geben Sie die Flags und die Prepare-Adresse an, die von ml6_transfer_pgm geliefert wurde.

Bei der Installation aus SNW6 ist im Menüpunkt 'Linken/Laden' unter 'Optionen' der Punkt 'Mehrfach Installierung' anzuwählen.

Notizen

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 Pro-

gramms 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. Außerdem 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 Multi-COM Karte arbeiten, und wie die Hardware vorbereitet werden muss. Änderungen am Turbo-Debugger sind nicht erforderlich.

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 Multi-COM Karte.

c) Ein Nullmodem-Kabel für die serielle Verbindung von PC und Multi-COM (z.B. Kanal B), ein fertiges Kabel ist von SORCUS lieferbar.

Von SORCUS getestete Software:

a) Betriebssystem "ML6-1A.01x" für Rev. C (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 5.0, falls auf Quelltextebene gearbeitet werden soll. Turbo-Assembler (TASM) Version 3.1 für Assemblerprogrammierer.

d) SNW6 Version 3.K oder größer bzw., falls die Echtzeitprogramme aus einem Anwenderprogramm heraus installiert werden, ML6BIB Version 2.A.000 für Rev. C.

Das Betriebssystem und SNW6 können Sie, falls Sie ältere Versionen besitzen, bei SORCUS Computer updaten lassen. 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 SNW6 oder ML6BIB 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 Multi-COM Karte zu laden, müssen Sie entweder SNW6 oder die Bibliothek ML6BIB 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 ml6rt_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 Multi-COM (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 Multi-COM Karte unter dem Multi-Tasking-Betriebssystem läuft, muss eine der seriellen Schnittstellen der Karte mit einer des Hostrechners (COM1 oder COM2) verbunden werden. Die Multi-COM Karte ist der Remote-Rechner. Als Hostrechner kann entweder der PC verwendet werden, in dem die Multi-COM 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 Multi-COM 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 Multi-COM Karte im PC steckt.*

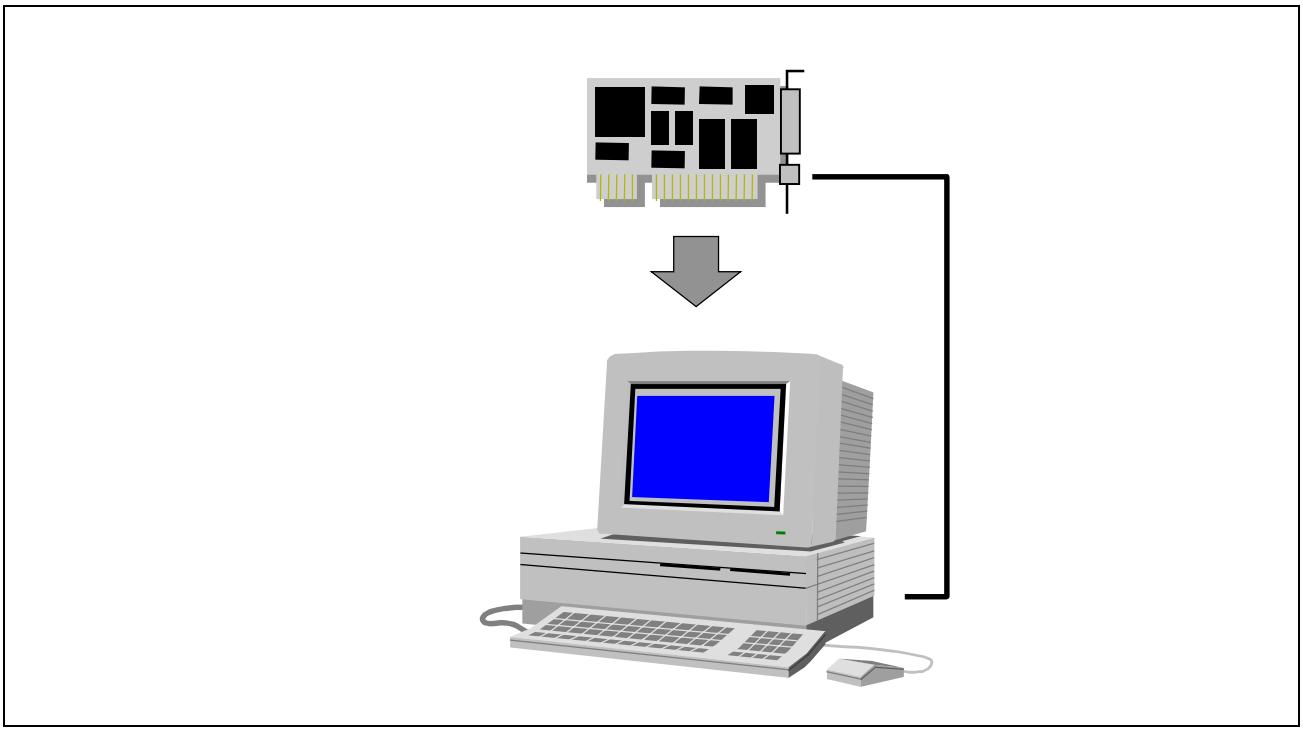


Abb. 8-1: Die Karte steckt in dem PC, auf dem auch der Turbo-Debugger läuft.

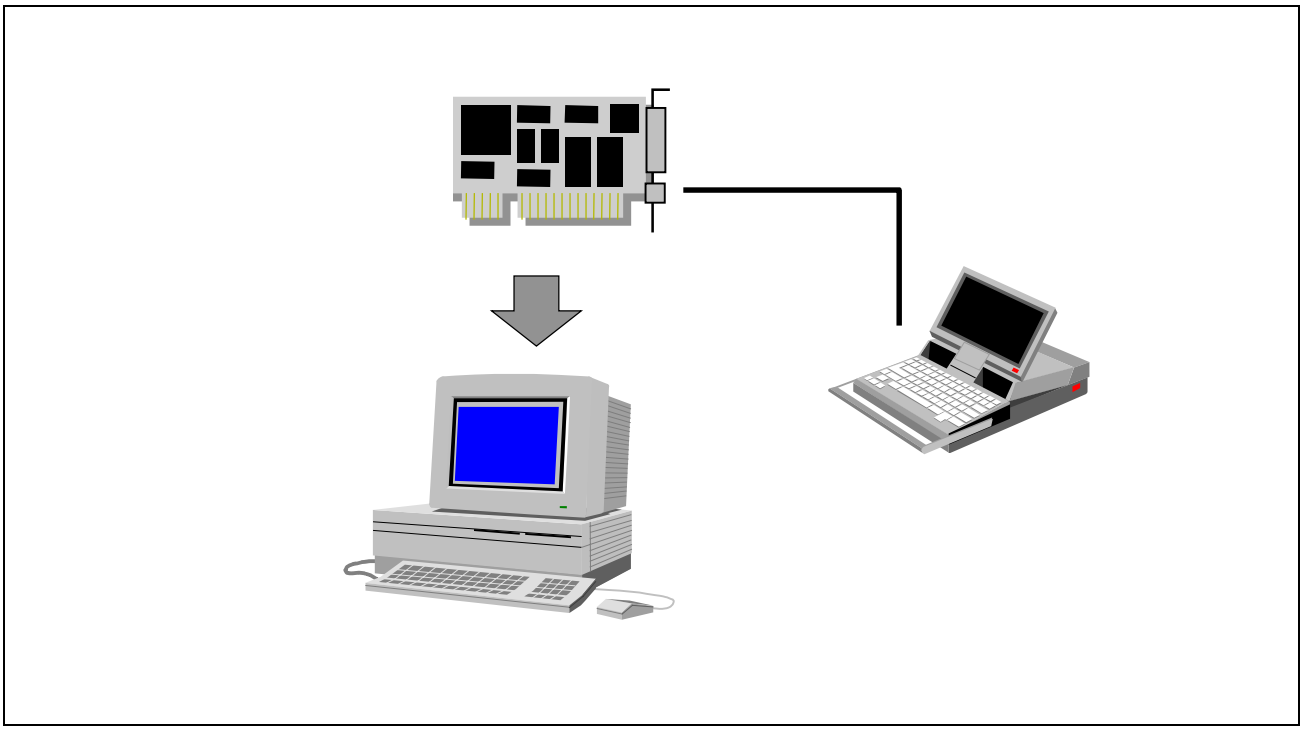


Abb. 8-2: Die Karte steckt in einem PC. Der Turbo-Debugger läuft auf einem zweiten PC und wird seriell mit der Multi-COM verbunden.

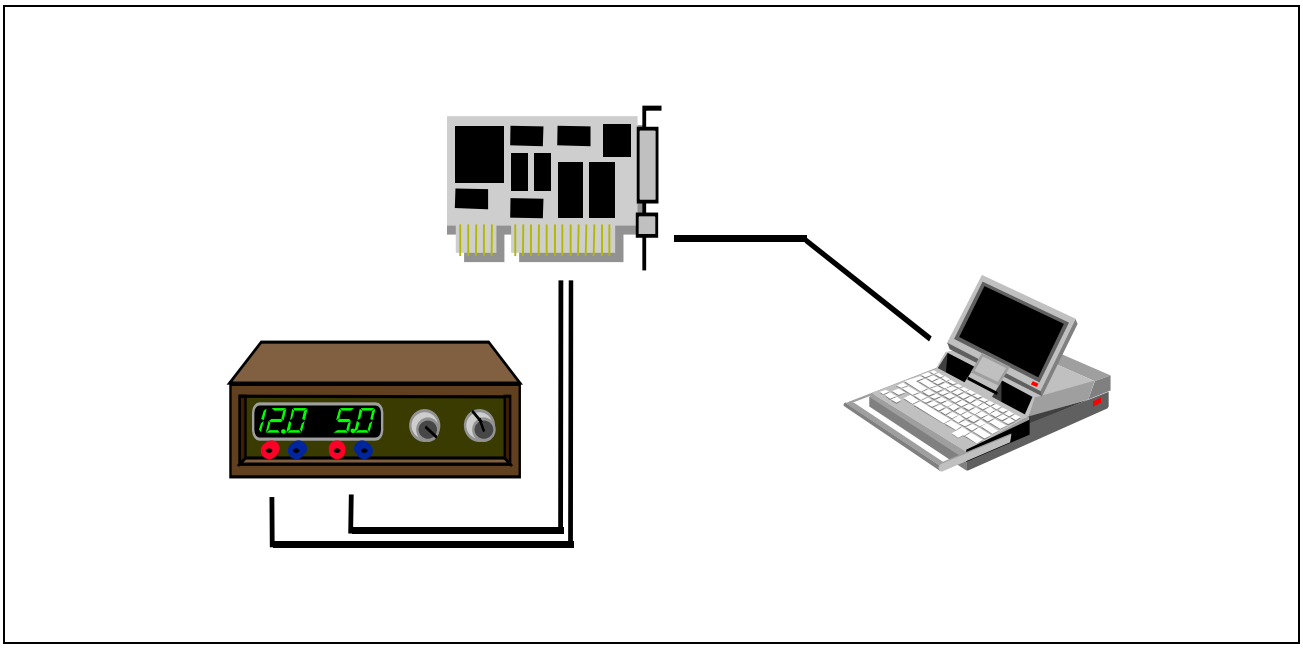


Abb. 8-3: Die Karte arbeitet 'stand-alone' und wird von einem Netzteil mit Spannung versorgt. Der Turbo-Debugger läuft auf einem PC und wird seriell angekoppelt.

Mit der Verbindung zwischen Hostrechner und Multi-COM Karte ist die Hardwareinstallation zum Remote-Debuggen abgeschlossen.

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 Multi-COM 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 "ML6-1A.01x". Außerdem ist eine SNW6-Version ab 3.K. notwendig. Sollten Sie ältere Versionen besitzen, so setzen Sie sich mit SORCUS wegen des Updates in Verbindung.

8.2.3.1. Installierung des Remote-Kernels auf der Multi-COM

Vorbereitung der zu debuggenden Programme

Um ein Echtzeitprogramm auf der Multi-COM Karte debuggen zu können, muss dieses Echtzeitprogramm auf der Karte als Task installiert worden sein. Die Installierung kann entweder über "SNW6" oder aus einem PC-Anwenderprogramm mit Hilfe der Bibliothek "ML6BIB" 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 SNW6

Der Remote-Kernel wird am einfachsten mit SNW6 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\ML6\RT\ML6REMOT).

Installierung des Kernels mit einer Installationsdatei (*.INS)

Statt mit dem Dialogfenster in SNW6 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 jedes Mal gesondert kümmern müssen. Allerdings ist die Einstellung und Änderung der Parameter mit dem M6PAR-Befehl (Turbo-Debuggerversion, Baudrate ...) unübersichtlicher. Folgende Echtzeitprogramme müssen auf der Karte installiert werden:

ML6REMOT.EXE und ML6REMCO.EXE

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).

M6INST ML6REMOT.EXE FF00 00FF 01 000000 980
M6INST ML6REMCO.EXE FF01 00FE 01 000000 980

; Gegebenenfalls können hier Anweisungen stehen, um
; Parameter des Remote-Kernels zu ändern.
; z.B.: M6PAR FF 06 04 um die Turbo-Debuggerversion 4.0 einzustellen

; Durch Aufruf der Prozedur 2 wird der Remote-Kernel gestartet.
M6PROC FF 02
```

Die Tasknummern der Programme (feh und ffh) können bei Bedarf geändert werden, falls die Tasknummern schon verwendet werden.

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 SNW6 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[1]	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-Informationen 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 Multi-COM 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.

[1] R = Parameter darf nur gelesen werden
R/W = Parameter darf gelesen und geschrieben werden

Nr.	Init	Typ[1]	Erklärung
4	**254** (feh)	Word (R/W)	Tasknummer des Kommunikationsinterfaces ‘ML6REMCO.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

8.2.3.3. Prozeduren des Remote-Kernels

Der Remote-Kernel (ML6REMOT.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 SNW6 (Tools/Turbo-Debugger) installieren, brauchen Sie sich nicht darum zu kümmern.

Nr.	Init	Typ[1]	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!
2	0	Byte (R/W)	Typ des Kommunikationsinterfaces: **0: Multi-COM**
3	0	Byte (R/W)	Auswahl des Kommunikationsinterfaces: **0: Multi-COM,** 1: SCC 1, 2: SCC 2, 3: SCC 3
4	0	Byte (R/W)	Kommunikationsschnittstelle der Karte, die verwendet werden soll: Multi-COM (**0 = Schnittstelle-A**, 1 = Schnittstelle B) SCC1 (0 = Kanal A, 1 = Kanal B) SCC2 (0 = Kanal C, 1 = Kanal D) SCC3 (0 = Kanal E, 1 = Kanal F)
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

[1] 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 und 5.0: 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 "M6P0300.EXE und "M6P0301.EXE" (Borland C) sind beide mit Debug-Informationen compiliert worden.

b) Stellen Sie die serielle Verbindung zwischen Hostrechner und Multi-COM 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 mit SNW6 oder mit Ihrem PC-Programm.

d) Rufen Sie in SNW6 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 SNW6.

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 m6p0301.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. "m6p0301.exe" ist der Name der Task, die debugged werden soll.

Im folgenden wird anhand des Beispielprogramms M6P0301.EXE (und dem Quelltext M6P0301.C) das Debuggen eines Programms auf der Multi-COM Karte gezeigt.

Falls alles in Ordnung ist, sollte sich der Debugger, wie in Abb. 8-4 dargestellt, mit dem Quellcode des Programms M6P0301.C melden.

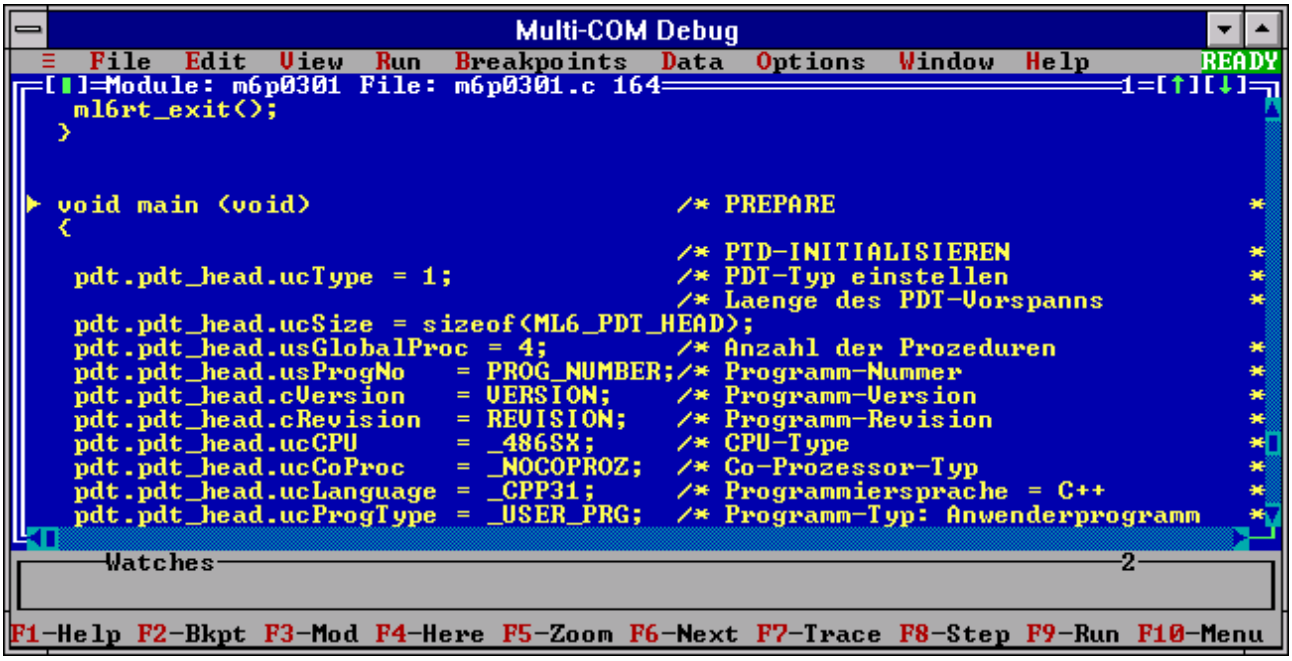


Abb. 8-4: Turbo-Debugger mit zu debuggendem Programm M6P0301.C

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 "void PFAR start(void)" des Programms M6P0301.C, 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".

```
Multi-COM Debug
≡ File Edit View Run Breakpoints Data Options Window Help READY
[■]=Module: m6p0301 File: m6p0301.c 108 ═══ 1=[↑][↓]
  ml6rt_local_led_off();                /* Die On-Board-LED ausschalten     *

  ml6rt_exit();                         /* Register wieder restaurieren     *
 }


▶ void PFAR start(void)
 {
  unsigned short timer;

  ml6rt_entry();

                                        /* Timer-Frequenz (1000 us = 1 kHz) *
  timer = ml6rt_convert_timer_data(1000L);
                                        /* Timer-A setzen                   *
  ml6rt_set_timer(TIMER_A, timer, INT_MODE);

  parameter.soft_cnt = parameter.blink_rate;
Watches ═══ 2
F1-Help F2-Bkpt F3-Mod F4-Here F5-Zoom F6-Next F7-Trace F8-Step F9-Run F10-Menu
```

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 **ml6rt_wakeup_task** das Programm auf der Karte aktiviert wurde! Wenn Sie sich jetzt die LED ansehen, so sollte Sie nach der Aktivierung blinken.

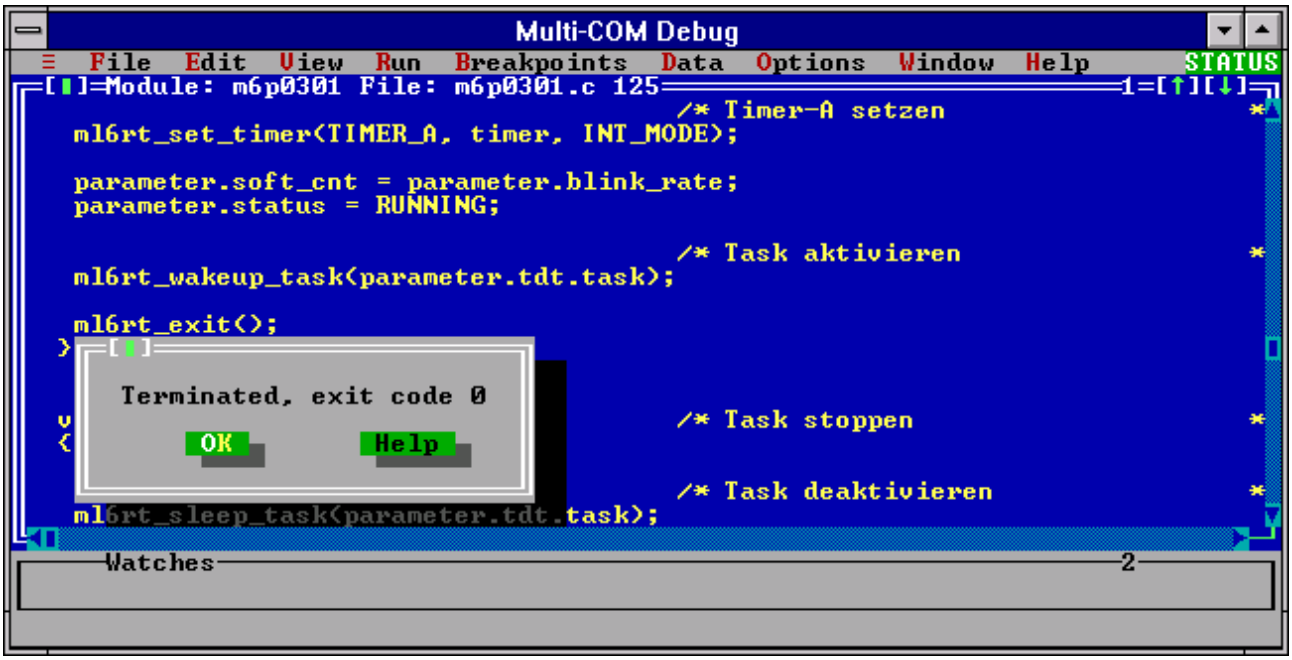


Abb. 8-6: Das Ende der Start-Prozedur wurde erreicht.

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. SNW6 oder den Turbo-Debugger aufzurufen, ohne die Entwicklungsumgebung zu verlassen. So ist es beispielsweise möglich, ein Programm zu schreiben, zu compilieren, das Programm mit SNW6 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 Programms, 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 ML6RTBIB zur Verfügung stellt.

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 Programme sind nur auf einer Multi-COM Karte lauffähig! Diese Programme sind also nicht auf einem PC verwendbar! **!**

Auf der Distributionsdiskette zur Echtzeitprogrammierung finden Sie neben den abgedruckten Beispielprogrammen in C und Borland-Pascal auch die Header-Datei für C (**ML6RTBIB.H**).

Lesen Sie bitte die README.DOC Dateien auf den mitgelieferten Disketten. Sie enthalten wichtige Informationen, die nicht mehr rechtzeitig ins Handbuch aufgenommen werden konnten.

9.2. Bibliotheksverwaltung

ml6rt_get_lib_version — Ermittle den Versionscode der Bibliothek

Pascal PROCEDURE ml6rt_get_lib_version (VAR version, date: LONGINT);

C VOID ml6rt_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.

ml6rt_get_osx_version — Ermittle den Versionscode des Betriebssystems

Pascal PROCEDURE ml6rt_get_osx_version (VAR version, date: LONGINT);

C VOID ml6rt_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 PFAR name (VOID);**

in Pascal: **PROCEDURE name;**

Einzige Ausnahme sind die Taskfunktionen. Sie haben folgenden vorgeschriebenen Aufbau:

in C: **VOID PFAR name (USHORT task, USHORT insize, VOID *inptr, USHORT outsize, VOID *outptr);**

in Pascal: **PROCEDURE name (task, insize : Word; VAR inptr; outsize: Word; VAR outptr);**

Parameter
- *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.

ml6rt_entry — Beginne globale Prozedur

Pascal PROCEDURE ml6rt_entry;

C VOID ml6rt_entry (VOID);

Funktion Diese Prozedur rettet alle Register und setzt das DS-Register auf das Datensegment des Programms. Der Aufruf dieser Prozedur muss 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 **ml6rt_entry** nicht aufrufen. In C kann auch alternativ das Makro _ml6rt_entry verwendet werden.

ml6rt_entry_task — Beginne globale Prozedur

Pascal FUNCTION ml6rt_entry_task: WORD;

C USHORT ml6rt_entry_task (VOID);

Funktion Die Funktion arbeitet genau wie **ml6rt_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. ml6rt_read_par_xxx) zugegriffen werden muss. Für diesen Fall liefert diese Funktion die Tasknummer zurück.

! Hinweis Lokale Variablen einer Taskprozedur dürfen erst nach dem Aufruf von ml6rt_entry bzw. ml6rt_entry_task initialisiert werden!

ml6rt_exit
ml6rt_exit_interrupt — Beende globale Prozedur

Pascal PROCEDURE ml6rt_exit;

PROCEDURE ml6rt_exit_interrupt;

C VOID ml6rt_exit (VOID);

VOID ml6rt_exit_interrupt (VOID);

Funktion Diese Prozeduren sind das Gegenstück zu **ml6rt_entry** und **ml6rt_entry_task**. Sie restaurieren die mit **ml6rt_entry** gesicherten Register und beenden die Prozedur. Der Aufruf dieser Prozeduren muss immer als letzter Aufruf in einer Prozedur stehen. **ml6rt_exit_interrupt** muss in Hauptprozeduren (Prozedur #0) von DI-Tasks verwendet werden! Sie beendet die Prozedur mit einem IRET (Return of Interrupt). In C können auch alternativ die Makros _ml6rt_exit bzw. _ml6rt_exit_interrupt verwendet werden.

ml6rt_entry_function — Beginne globale Funktion

Pascal PROCEDURE ml6rt_entry_function;

C VOID ml6rt_entry_function (VOID);

Funktion Diese Prozedur arbeitet genauso wie **ml6rt_entry,** jedoch ist sie für eine Task-*Funktion* bestimmt. Die Task-Funktion liefert im Gegensatz zur Task-Prozedur ein Ergebnis. **ml6rt_entry_function** sorgt dafür, dass die im Funktionskopf deklarierten Parameter gesetzt werden. In C kann auch alternativ das Makro _ml6rt_entry_function verwendet werden.

Hinweis Lokale Variablen einer Taskfunktion dürfen erst nach dem Aufruf von ml6rt_entry_function initialisiert werden!

ml6rt_exit_function — Beende globale Funktion

Pascal: PROCEDURE ml6rt_exit_function (leftsize, outsize, error: WORD);

C: VOID ml6rt_exit_function (USHORT leftsize, USHORT outsize, USHORT error);

Funktion: Diese Prozedur beendet eine Funktion einer Task. Sie arbeitet genauso wie **ml6rt_exit**. Es wird jedoch zusätzlich eine Antwort zur aufrufenden Task zurückgesendet. In C kann auch alternativ das Makro _ml6rt_exit_function verwendet werden.

Parameter:

leftsize: Anzahl Bytes, die nicht verwendet werden konnte (nur gültig, wenn *error* = 0).

outsize: Größe der Antwort (in Bytes, nur gültig, wenn *error* = 0).

error: Fehlermeldung für die aufrufende Task (0 = kein Fehler). Im Fehlerfall darf die Funktion weder Daten übernommen noch zurückgegeben haben. 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 globalen Prozeduren und Funktionen

ml6rt_call_func — Rufe eine Funktion einer Task auf

Pascal: PROCEDURE ml6rt_call_func (task, funcnr, VAR insize: WORD; VAR inptr; maxoutsize: WORD; VAR outsize: WORD; VAR outptr; VAR error: WORD);

C: VOID ml6rt_call_func (USHORT task, USHORT funcnr, USHORT* insize, VOID*inptr; USHORT maxoutsize, USHORT* outsize, VOID* outptr, USHORT* error);

Funktion: Mit dieser Prozedur kann eine globale Funktion einer auf der Multi-COM Karte installierten Task aufgerufen werden. Ein möglicher Fehler wird in *error* zurückgegeben (kein Aufruf der Fehlerbehandlung).

Parameter *task*: Nummer der Task, die die gewünschte Funktion enthält.

funcnr: Nummer der Funktion.

insize: Anzahl der Bytes, die an die Funktion übergeben werden. Nach dem Aufruf der Funktion enthält *insize* die Anzahl der nicht verwendeten Bytes.

inptr: Zeiger auf einen Variablenbereich, in dem die Daten stehen, die an die Funktion übergeben werden.

maxoutsize: Maximale Anzahl an Bytes, die die aufgerufene Funktion zurückliefern darf bzw. soll.

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

outptr: Zeiger auf eine Variable, in die die Rückgabedaten der Funktion eingetragen werden. *outptr* kann gleich *inptr* 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 Fehlermeldung der aufgerufenen Funktion.

ml6rt_call_proc — Rufe eine Prozedur einer Task auf

Pascal PROCEDURE ml6rt_call_proc (task, procnr : WORD);

C VOID ml6rt_call_proc (USHORT task, USHORT procnr);

Funktion Diese Prozedur ruft eine globale Prozedur einer auf der Multi-COM installierten Task auf. 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.

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 Multi-COM Karte das zugehörige Echtzeitprogramm verwaltet. Er wird in den Parameterbeschreibungen nicht jedes Mal aufgeführt.

9.4.1. Taskinformationen abfragen

ml6rt_get_task_status — Prüfe, ob eine Task aktiviert ist

Pascal: FUNCTION ml6rt_get_task_status (task: WORD): BYTE;

C: byte ml6rt_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-, II- und DI-Tasks können maximal einmal aktiviert sein.

ml6rt_get_tdt_address — Ermittle Adresse der TDT

Pascal:
```
PROCEDURE ml6rt_get_tdt_address (task: WORD;
VAR adr: POINTER);
```

C: VOID ml6rt_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.

ml6rt_get_proc_address — Ermittle Adresse einer Prozedur einer Task

Pascal PROCEDURE ml6rt_get_proc_address (task: WORD; procnr: WORD; VAR adr: POINTER);

C VOID ml6rt_get_proc_address (USHORT task, USHORT procnr, VOID* adr);

Funktion Diese Prozedur ermittelt die Adresse einer globalen Taskprozedur. Mit Hilfe dieses Zeigers kann die globale Prozedur direkt aufgerufen werden (Aufruf ist schneller als mit ml6rt_call_proc).

Parameter *procnr*: Nummer der gesuchten Prozedur.

adr: Zeiger auf eine Variable, in die die Adresse (Segment: Offset) der Funktion eingetragen wird.

Hinweis Taskfunktionen können nicht direkt aufgerufen werden.

ml6rt_get_task_info — Fordere Systeminformationen über eine Task an

Pascal PROCEDURE ml6rt_get_task_info (task: WORD; callnr: WORD; VAR result: LONGINT);

C VOID ml6rt_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.

ml6rt_get_int_task — Ermittle Tasktyp und -nummer

Pascal: PROCEDURE ml6rt_get_int_task (inter: WORD; VAR task: WORD; VAR typ: BYTE);

C: VOID ml6rt_get_int_task (USHORT inter, USHORT* task, UCHAR* type);

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.

type: 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.**

ml6rt_get_task_number — Ermittle Task-Nummer

Pascal: FUNCTION ml6rt_get_task_number (prog_nr: WORD; VAR install_nr: WORD): WORD;

C: USHORT ml6rt_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.
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

ml6rt_wakeup_task — Aktiviere eine Task

Pascal PROCEDURE ml6rt_wakeup_task (task: WORD);

C VOID ml6rt_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. Für TI-Tasks gibt es eine eigene Aktivierungsroutine (**ml6rt_wakeup_ti_task**).

ml6rt_wakeup_ni_task — Aktiviere eine NI-Task

Pascal PROCEDURE ml6rt_wakeup_ni_task (task, prior: WORD);

C VOID ml6rt_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 **ml6rt_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 nur dass geprüft wird, ob bereits eine NI-Task vorgezogen, aber noch nicht begonnen wurde. Wenn das so ist, wird die Fehlerbehandlungsroutine aufgerufen.

ml6rt_sleep_task — Deaktiviere eine Task

Pascal PROCEDURE ml6rt_sleep_task (task: WORD);

C VOID ml6rt_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.

ml6rt_wakeup_ti_task — Aktiviere eine TI-Task

Pascal PROCEDURE ml6rt_wakeup_ti_task (task: WORD; priority: BYTE; count, interval, holdoff: LONGINT);

C VOID ml6rt_wakeup_ti_task (USHORT task, UCHAR 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 den Parameterbereich einer Task

Zugriffe auf Variablen 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 **ml6_get_par_address** einmal ermittelt werden und über die Adresse (Pointer) zugegriffen werden.

ml6rt_read_par_byte
ml6rt_read_par_word
ml6rt_read_par_dword — Lies Parameter einer Task

Pascal

```
FUNCTION ml6rt_read_par_byte (task, start: WORD): BYTE;
FUNCTION ml6rt_read_par_word (task, start: WORD): WORD;
PROCEDURE ml6rt_read_par_dword (task, start: WORD;
VAR data: LONGINT);
```

C

```
byte ml6rt_read_par_byte (USHORT task, USHORT start);
USHORT ml6rt_read_par_word (USHORT task, USHORT start);
VOID ml6rt_read_par_dword (USHORT task, USHORT start,
ULONG* data);
```

Funktion: Die Funktionen lesen ein Byte, Wort oder Doppelwort aus dem Parameterbereich einer Task. Beachten Sie, dass der Wert eines Doppelwortes nicht als Funktionsergebnis sondern in *data* zurückgegeben wird.

Parameter: *data*: Zeiger auf eine Variable, die das Ergebnis von **ml6rt_read_par_dword** aufnimmt.

ml6rt_read_par_block — Lies Parameterblock

Pascal
PROCEDURE ml6rt_read_par_block (task, start, size: WORD; VAR data_var);

C
VOID ml6rt_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 (target))
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.

ml6rt_write_par_byte
ml6rt_write_par_word
ml6rt_write_par_dword — Schreibe Parameter einer Task

Pascal
PROCEDURE ml6rt_write_par_byte (task, start: WORD; data: BYTE);

PROCEDURE ml6rt_write_par_word (task, start: WORD; data: WORD);

PROCEDURE ml6rt_write_par_dword (task, start: WORD; data: LONGINT);

C
VOID ml6rt_write_par_byte (USHORT task, USHORT start, UCHAR data);

VOID ml6rt_write_par_word (USHORT task, USHORT start, USHORT data);

VOID ml6rt_write_par_dword (USHORT task, USHORT start, ULONG data);

Funktion: Mit diesen Prozeduren kann ein Byte, Wort oder Doppelwort in den Parameterbereich einer Task geschrieben werden.

Parameter:
data: Wert, der geschrieben werden soll.

ml6rt_write_par_block Schreibe Parameterblock einer Task

Pascal PROCEDURE ml6rt_write_par_block (task, start, size: WORD; VAR data_var);

C VOID ml6rt_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 (source))

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.

ml6rt_get_par_address Ermittle Adresse eines Parameters einer Task

Pascal PROCEDURE ml6rt_get_par_address (task, start: WORD; VAR adr: LONGINT);

C VOID ml6rt_get_par_address (USHORT task, USHORT start, ULONG* adr);

Funktion Diese Prozedur ermittelt die physikalische Adresse eines Parameters.

Parameter *adr*: Zeiger auf eine Variable, in die die physikalische Adresse des Parameters eingetragen wird.

9.4.4. Zugriff auf den Datenbereich einer Task

Der Zugriff auf den Datenbereich einer Task erfolgt 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.

ml6rt_reset_r_pointer — Setze Daten-Lesezeiger zurück

Pascal PROCEDURE ml6rt_reset_r_pointer (task: WORD);

C VOID ml6rt_reset_r_pointer (USHORT task);

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

ml6rt_move_r_pointer — Verschiebe Daten-Lesezeiger

Pascal PROCEDURE ml6rt_move_r_pointer (task: WORD;
offset: LONGINT);

C VOID ml6rt_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.

ml6rt_read_data_byte
ml6rt_read_data_word — Lies Einzelwert
ml6rt_read_data_dword — aus dem Datenbereich

Pascal
FUNCTION ml6rt_read_data_byte (task: WORD): BYTE;

FUNCTION ml6rt_read_data_word (task: WORD): WORD;

PROCEDURE ml6rt_read_data_dword (task: WORD;
VAR data: LONGINT);

C
byte ml6rt_read_data_byte (USHORT task);

USHORT ml6rt_read_data_word (USHORT task);

VOID ml6rt_read_data_dword (USHORT task, ULONG* data);

Funktion
Die Prozeduren lesen ein Byte, Wort oder Doppelwort 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 **ml6rt_read_data_dword** aufnimmt.

ml6rt_read_data_block — Lies Datenblock

Pascal
PROCEDURE ml6rt_read_data_block (task, size: WORD;
VAR data_var);

C
VOID ml6rt_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 (target)).

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

ml6rt_read_data — Lies Datenblock direkt

Pascal: PROCEDURE ml6rt_read_data (task: WORD; rel_ofs: LONGINT; size: WORD; VAR data_var);

C: VOID ml6rt_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 zurückgesetzt und 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 (target)).
- *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. !

ml6rt_reset_w_pointer — Setze Daten-Schreibzeiger zurück

Pascal: PROCEDURE ml6rt_reset_w_pointer (task: WORD);

C: VOID ml6rt_reset_w_pointer (USHORT task);

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

ml6rt_move_w_pointer — Verschiebe Daten-Schreibzeiger

Pascal: PROCEDURE ml6rt_move_w_pointer (task: WORD; offset: LONGINT);

C: VOID ml6rt_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.

ml6rt_write_data_byte
ml6rt_write_data_word
ml6rt_write_data_dword

Schreibe Daten in den Datenbereich

Pascal PROCEDURE ml6rt_write_data_byte (task: WORD; data: BYTE);

PROCEDURE ml6rt_write_data_word (task: WORD; data: WORD);

PROCEDURE ml6rt_write_data_dword (task: WORD; data: LONGINT);

C VOID ml6rt_write_data_byte (USHORT task, UCHAR data);

VOID ml6rt_write_data_word (USHORT task, USHORT data);

VOID ml6rt_write_data_dword (USHORT task, ULONG data);

Funktion Diese Prozeduren schreiben ein Byte, Wort oder Doppelwort 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.

ml6rt_write_data_block

Schreibe Datenblock

Pascal PROCEDURE ml6rt_write_data_block (task, size: WORD; VAR datavar);

C VOID ml6rt_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. mit sizeof (source))

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

ml6rt_write_data Schreibe Datenblock direkt

Pascal PROCEDURE ml6rt_write_data (task: WORD; rel_ofs: LONGINT; size: WORD; VAR data_var);

C VOID ml6rt_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 zurückgesetzt und auf den angegebenen Wert positioniert. Der Schreibzeiger der Task wird anschließend um die gelesene Anzahl Bytes erhöht.

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

9.5. Ringpuffer

Das Betriebssystem der Multi-COM Karte 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 anderen Task gelesen bzw. beschrieben werden kann. In einem solchen Konfliktfall 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 Verfü-

gung steht, wird eine Fehlermeldung zurückgeliefert. Das Lesen von Daten geschieht ebenfalls über einen Zeiger, der automatisch um die Anzahl der gelesenen Bytes verschoben wird.

ml6rt_create_buffer — Erzeuge Ringpuffer

Pascal
FUNCTION ml6rt_create_buffer (task: WORD; strategy, align: BYTE; usage: WORD; VAR size: LONGINT): HBUF6;

C
HBUF6 ml6rt_create_buffer (USHORT task, UCHAR strategy, UCHAR 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.

ml6rt_clear_buffer — Lösche Inhalt eines Ringpuffers

Pascal PROCEDURE ml6rt_clear_buffer (buffer: LONGINT);

C VOID ml6rt_clear_buffer (HBUF6 buffer);

Funktion Diese Prozedur löscht alle Daten eines Ringpuffers.

ml6rt_get_buffer_status — Ermittle Status eines Ringpuffers

Pascal PROCEDURE ml6rt_get_buffer_status (buffer: LONGINT; freesize, usedsize: LONGINT);

C VOID ml6rt_get_buffer_status (HBUF6 buffer, ULONG* freesize, ULONG* usedsize);

Funktion: Diese Prozedur ermittelt, wie viel Speicher im Puffer noch frei ist und wie viel 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.

ml6rt_write_buffer_byte
ml6rt_write_buffer_word
ml6rt_write_buffer_dword Schreibe einzelne Daten in Puffer

Pascal FUNCTION ml6rt_write_buffer_byte (buffer: LONGINT; data: BYTE): WORD;

FUNCTION ml6rt_write_buffer_word (buffer: LONGINT; data: INTEGER): WORD;

FUNCTION ml6rt_write_buffer_dword (buffer: LONGINT; data: LONGINT): WORD;

C USHORT ml6rt_write_buffer_byte (HBUF6 buffer, UCHAR data);

USHORT ml6rt_write_buffer_word (HBUF6 buffer, USHORT data);

USHORT ml6rt_write_buffer_dword (HBUF6 buffer, ULONG data);

Funktion: Diese Funktionen schreiben ein Byte, Wort oder Doppelwort 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.

ml6rt_write_buffer_block Schreibe Datenblock in Puffer

Pascal FUNCTION ml6rt_write_buffer_block (buffer: LONGINT; size: WORD; VAR datavar);

C USHORT ml6rt_write_buffer_block (HBUF6 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.

ml6rt_write_buffer_max Schreibe Datenblock in Puffer

Pascal FUNCTION ml6rt_write_buffer_max (buffer: LONGINT;
VAR size: WORD; VAR datavar);

C USHORT ml6rt_write_buffer_max (HBUF6 buffer, USHORT* size,
VOID* datavar);

Funktion Diese Funktion entspricht der Funktion **ml6rt_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 **ml6rt_write_buffer_block.**

ml6rt_read_buffer_byte
ml6rt_read_buffer_word
ml6rt_read_buffer_dword Lies einzelne Daten aus Puffer

Pascal FUNCTION ml6rt_read_buffer_byte (buffer: LONGINT;
VAR data_var: BYTE): WORD;

FUNCTION ml6rt_read_buffer_word (buffer: LONGINT;
VAR data_var: INTEGER): WORD;

FUNCTION ml6rt_read_buffer_dword (buffer: LONGINT;
VAR data_var: LONGINT): WORD;

C USHORT ml6rt_read_buffer_byte (HBUF6 buffer, CHAR* data_var);

USHORT ml6rt_read_buffer_word (HBUF6 buffer,
USHORT* data_var);

USHORT ml6rt_read_buffer_dword (HBUF6 buffer,
ULONG* data_var);

Funktion: Diese Funktionen lesen ein Byte, Wort oder Doppelwort 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.

ml6rt_read_buffer_block — Lies Datenblock aus Puffer

Pascal
FUNCTION ml6rt_read_buffer_block (buffer: LONGINT;
size: WORD; VAR data_var): WORD;

C
USHORT ml6rt_read_buffer_block (HBUF6 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.

ml6rt_read_buffer_max — Lies Datenblock aus Puffer

Pascal
FUNCTION ml6rt_read_buffer_max (buffer: LONGINT;
VAR size: WORD; VAR data_var): WORD;

C
USHORT ml6rt_read_buffer_max (HBUF6 buffer, USHORT* size,
VOID* data_var);

Funktion Diese Funktion entspricht der Funktion **ml6rt_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 **ml6rt_read_buffer_block**.

ml6rt_view_buffer_block
ml6rt_view_buffer_max — Kopiere Daten aus Puffer

Pascal

```
FUNCTION ml6rt_view_buffer_block (buffer: LONGINT;
size: WORD; VAR data_var): WORD;

FUNCTION ml6rt_view_buffer_max (buffer: LONGINT;
VAR size: WORD; VAR data_var): WORD;
```

C

```
USHORT ml6rt_view_buffer_block (HBUF6 buffer, USHORT size,
VOID* data_var);

USHORT ml6rt_view_buffer_max (HBUF6 buffer, USHORT* size,
VOID* data_var);
```

Funktion

Diese Funktionen arbeiten genauso wie **ml6rt_read_buffer_block** und **ml6rt_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 ML6RTBIB.H folgende Konstanten definiert:

Bezeichnung	Wert	Bedeutung
NMI	02h	Nicht maskierbarer Interrupt
IRQ_RBF	78h	RBF-Interrupt von PC-Schnittstelle
IRQ_TEMP	79h	Interrupt-Leitung von Temperatursensor
IRQ_SLAVE	7ah	Interrupt vom Slave-Controller
-	7bh	Interrupt-Leitung, reserviert
IRQ_SCC3	7ch	Interrupt-Leitung von SCC3, Kanal E und F
IRQ_SCC2	7dh	Interrupt-Leitung von SCC2, Kanal C und D
IRQ_SCC1	7eh	Interrupt-Leitung von SCC1, Kanal A und B
IRQ_FAN	7fh	Interrupt-Leitung von Lüfterüberwachung
IRQ_SCC	90h	Interrupt vom SCC1, Kanal A und B (alternativ)
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
IRQ_EXT	95h	Interrupt vom Stecker St3 (IRQ-G)
IRQ_COM_B	96h	Interrupt Ri der ser. Schnittstelle-B (IRQ-H)
IRQ_TBE	97h	TBE-Interrupt von der PC-Schnittstelle

ml6rt_unmask_int — Gib Hardware-Interrupt frei

Pascal PROCEDURE ml6rt_unmask_int (hrdirq: BYTE);

C VOID ml6rt_unmask_int (UCHAR hrdirq);

Funktion Diese Prozedur gibt einen Interrupt frei.

ml6rt_mask_int — Sperre Hardware-Interrupt

Pascal PROCEDURE ml6rt_mask_int (hrdirq: BYTE);

C VOID ml6rt_mask_int (UCHAR hrdirq);

Funktion Diese Prozedur sperrt einen Interrupt.

ml6rt_set_int_edge — Stelle Interrupt-Flanke ein

Pascal PROCEDURE ml6rt_set_int_edge (hrdirq: BYTE; edge: BYTE);

C VOID ml6rt_set_int_edge (UCHAR hrdirq, UCHAR 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.

ml6rt_end_of_int — Beende Interrupt-Prozedur

Pascal PROCEDURE ml6rt_end_of_int (hrdirq: BYTE);

C VOID ml6rt_end_of_int (UCHAR hrdirq);

Funktion Diese Prozedur sendet für den Interrupt ein "End of Interrupt" (EOI) zum Master- und zum Slave-Interrupt-Controller.

ml6rt_clear_int — Lösche Pending-Interrupt

Pascal PROCEDURE ml6rt_clear_int (hrdirq: BYTE);

C VOID ml6rt_clear_int (UCHAR hrdirq);

Funktion Diese Prozedur löscht den Interrupt im Interrupt-Controller. Sie wird in der Regel vor **ml6rt_unmask_int** verwendet, um sicherzustellen, dass ein vorher aufgetretener Interrupt nicht zur Ausführung kommt.

ml6rt_disable_interruptible — Interrupts deaktivieren
ml6rt_enable_interruptible — Interrupts aktivieren

Pascal

```
PROCEDURE ml6rt_disable_interruptible;
PROCEDURE ml6rt_enable_interruptible;
```

C

```
VOID ml6rt_disable_interruptible (VOID);
VOID ml6rt_enable_interruptible (VOID);
```

Funktion

Diese Prozeduren maskieren bzw. demaskieren alle Interrupts (außer NMI) auf der Multi-COM Karte. Durch Maskieren der Interrupts lassen sich Programmteile gegen Unterbrechung schützen. Anschließend muss die Prozedur ml6rt_enable_interrupts aufgerufen werden, um die Interrupts wieder freizugeben. Beide Routinen müssen innerhalb der selben Prozedur stehen.
In C können alternativ auch die Makros _ml6rt_disable_interruptible und _ml6rt_enable_interruptible verwendet werden.

9.6.2. Speicherverwaltung

ml6rt_allocate_ram Reserviere Speicher auf der Multi-COM

Pascal PROCEDURE ml6rt_allocate_ram (task, usage: WORD;
strategy, align: BYTE; VAR size: LONGINT; VAR adr: LONGINT);

C VOID ml6rt_allocate_ram (USHORT task, USHORT usage,
UCHAR strategy, UCHAR align, ULONG* size, ULONG* adr);

Funktion Diese Prozedur reserviert Speicherplatz auf der Multi-COM Karte.

Parameter

task: Task, für die Speicher reserviert werden soll. Die Angabe dient Protokollzwecken auf der Multi-COM Karte.

usage: Reserviert, immer = 0 setzen.

strategy: Reservierungsstrategie. Folgende Werte sind zulässig:

1 = Es wird die in *size* angegebene Speichermenge (in Byte), von unten 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 oben im freien Speicher beginnend reserviert werden.

4 = wie 2, jedoch soll der Speicher von oben 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. Wenn *size* = 0 ist, dann ist die Adresse nicht gültig.

ml6rt_get_free_ram_size — Ermittle freien Speicher

Pascal
PROCEDURE ml6rt_get_free_ram_size (align: BYTE;
VAR size: LONGINT);

C
VOID ml6rt_get_free_ram_size (UCHAR align, ULONG* size);

Funktion:
Diese Prozedur ermittelt die Größe des freien Speichers auf der Multi-COM 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.

ml6rt_read_ram — Lies aus RAM

Pascal
PROCEDURE ml6rt_read_ram (size: WORD; source: LONGINT;
VAR dest);

C
VOID ml6rt_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).

source: Quelladresse (physikalisch).

dest: Zeigt auf den Zielbereich (Seg:Offs).

ml6rt_write_ram — Schreibe in RAM

Pascal: PROCEDURE ml6rt_write_ram (size: WORD; dest: LONGINT; VAR source);

C: VOID ml6rt_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).
- *dest*: Zeigt auf den Zielbereich (physikalisch).
- *source*: Quelladresse (Seg:Offs).

ml6rt_copy_ram — Kopiere Speicher

Pascal: PROCEDURE ml6rt_copy_ram (size: WORD; VAR source; VAR dest);

C: VOID ml6rt_copy_ram (USHORT size, VOID* source, VOID* dest);

Funktion: Diese Prozedur kopiert einen Speicherbereich.

Parameter:
- *size*: Größe des Blocks (in Byte).
- *source*: Quelladresse (Seg:Offs).
- *dest*: Zeigt auf den Zielbereich (Seg:Offs).

ml6rt_cache_control — Kontrolliere den Cache

Pascal: PROCEDURE ml6rt_cache_control (mode: BYTE);

C: VOID ml6rt_cache_control (UCHAR mode);

Funktion: Die Prozedur dient zum Ein- und Ausschalten des Prozessor-Caches auf der Multi-COM Karte.

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

9.6.3. Timer-Kontrolle

Für die Auswahl der Timer stehen die Konstanten TIMER_A, TIMER_B, TIMER_C in ML6RTBIB.H zur Verfügung. Alle Timer sind 16 Bit breit und werden mit jedem Takt um eins dekrementiert.

ml6rt_set_timer — Setze und starte einen Timer

Pascal: PROCEDURE ml6rt_set_timer (counter: BYTE; count: WORD; mode: BYTE);

C: VOID ml6rt_set_timer (UCHAR counter, USHORT count, UCHAR 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 1 MHz beträgt die Auflösung 1 µs, bei 2,5 MHz beträgt die Auflösung 400 ns und bei 10 MHz sind dies 100 ns. Die maximale Laufzeit ergibt sich aus Auflösung * 65535. Die Einstellung der Quarzfrequenz erfolgt in EEPROM-Wort 3.

mode: Betriebsart des Timers. Derzeit ist dieser Parameter immer mit der Konstanten **INT_MODE** zu beschreiben**.**

ml6rt_get_timer — Lies Zählerstand eines Timers

Pascal: PROCEDURE ml6rt_get_timer (counter: BYTE; VAR count: WORD; VAR status: BYTE);

C: VOID ml6rt_get_timer (UCHAR counter, USHORT* count, UCHAR* 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.

ml6rt_convert_timer_data Berechne einen Timerwert

Pascal FUNCTION ml6rt_convert_timer_data (time: LONGINT): WORD;

C USHORT ml6rt_convert_timer_data (ULONG time);

Funktion Diese Funktion 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 **ml6rt_set_timer** übergeben werden. Im Fall eines ungültigen Parameters (Wertebereich s. **ml6rt_set_timer**) wird die Fehlerbehandlungsprozedur aufgerufen.

Parameter *time*: Zeit (in Mikrosekunden).

9.6.4. EEPROM-Zugriffe

ml6rt_read_eeprom_copy
ml6rt_read_eeprom_direct — Lies EEPROM-Wort

Pascal

FUNCTION ml6rt_read_eeprom_copy (device, reladr: BYTE): WORD;

FUNCTION ml6rt_read_eeprom_direct (device, reladr: BYTE): WORD;

C

USHORT ml6rt_read_eeprom_copy (UCHAR device, UCHAR reladr);

USHORT ml6rt_read_eeprom_direct (UCHAR device, UCHAR 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

device: Zu lesender Teil des EEPROMs:
- 0: Multi-COM Karte
- 1: Schnittstelle A und B (SCC1)
- 2: Schnittstelle C und D (SCC2)
- 3: Schnittstelle E und F (SCC3)

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

ml6rt_write_eeprom_copy
ml6rt_write_eeprom_direct — Schreibe EEPROM-Wort

Pascal

PROCEDURE ml6rt_write_eeprom_copy (device, reladr: BYTE; data: WORD);

PROCEDURE ml6rt_write_eeprom_direct (device, reladr: BYTE; data: WORD);

C

VOID ml6rt_write_eeprom_copy (UCHAR device, UCHAR reladr, USHORT data);

VOID ml6rt_write_eeprom_direct (UCHAR device, UCHAR 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: *device, reladr*: siehe **ml6rt_read_eeprom_direct /copy**.

data: Daten, die geschrieben werden sollen.

9.6.5. Zugriffe auf die Echtzeituhr

ml6rt_get_rtc_status — Lies Status der Uhr

Pascal FUNCTION ml6rt_get_rtc_status : BYTE;

C UCHAR ml6rt_get_rtc_status (VOID);

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

Bit	Bedeutung
0	1 = Uhr gestellt
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

ml6rt_set_rtc_mode Stelle Betriebsart der Uhr ein

Pascal PROCEDURE ml6rt_set_rtc_mode (mode: BYTE);

C VOID ml6rt_set_rtc_mode (UCHAR 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

ml6rt_get_date_and_time_2000
ml6rt_get_date_and_time — Lies Datum und Uhrzeit

Pascal

PROCEDURE ml6rt_get_date_and_time (VAR year, month, day, day_of_week, hour, minute, second: WORD);

PROCEDURE ml6rt_get_date_and_time_2000 (VAR year, month, day, day_of_week, hour, minute, second: WORD);

C

VOID ml6rt_get_date_and_time (USHORT* year, USHORT* month, USHORT* day, USHORT *day_of_week, USHORT* hour, USHORT* minute, USHORT* second);

VOID ml6rt_get_date_and_time_2000 (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 bestehend aus vier Dezimalen (z.B.: "1997").

month: Monat

day: Tag

dow: Wochentag (0 = Sonntag, 1 = Montag, ..., 6 = Samstag).

hour: Stunde

minute: Minute

second: Sekunde

ml6rt_set_date_and_time_2000
ml6rt_set_date_and_time — Setze Datum und Uhrzeit

Pascal

PROCEDURE ml6rt_set_date_and_time_2000 (year, month, day, dow, hour, minute, second: WORD);

PROCEDURE ml6rt_set_date_and_time (year, month, day, dow, hour, minute, second: WORD);

C

VOID ml6rt_set_date_and_time_2000 (USHORT year, USHORT month, USHORT day, USHORT dow, USHORT hour, USHORT minute, USHORT second);

VOID ml6rt_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 **ml6rt_set_date_and_time.**

*Die Funktionen **ml6rt_get_date_and_time_2000** und **ml6rt_set_date_and_time_-2000** entsprechen den Funktionen **ml6rt_get_date_and_time** und **ml6rt_set_date_-and_time**. Bis zum Jahr 2079 liefern sie die richtige Jahreszahl. Um auf das Problem eines Überlaufs der Jahreszahl hinzuweisen, ruft ml6rt_get_date_and_-time_2000 vom Jahr 2060 an zusätzlich den Errorhandler auf.*

ml6rt_get_rtc_date_code — Lies Datum

Pascal FUNCTION ml6rt_get_rtc_date_code : LONGINT;

C ULONG ml6rt_get_rtc_date_code (VOID);

Funktion Diese Funktion liest das Datum der Echtzeituhr. Das Datum wird als Datecode übergeben (siehe Kapitel 9.8).

ml6rt_get_rtc_time_code — Lies Uhrzeit

Pascal FUNCTION ml6rt_get_rtc_time_code : LONGINT;

C ULONG ml6rt_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

ml6rt_store_80487 — Sichere Coprozessorstatus

Pascal PROCEDURE ml6rt_store_80487;

C VOID ml6rt_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 **ml6rt_restore_80487** wieder hergestellt worden ist.

ml6rt_restore_80487 — Stelle Coprozessorstatus wieder her

Pascal PROCEDURE ml6rt_restore_80487;

C VOID ml6rt_restore_80487 (VOID);

Funktion Diese Prozedur stellt den Zustand des Coprozessor so wieder her, wie er mit **ml6rt_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

ml6rt_send_buffer_srq Sende gepufferten Service-Request

Pascal PROCEDURE ml6rt_send_buffer_srq (data, mode: WORD);

C VOID ml6rt_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 Warteschlage 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 0bfh liegen (siehe Anhang F).

mode: Spezifiziert den Host, an den der Service-Request gesendet wird. Derzeit ist nur *mode* = 1 erlaubt (PC-Bus).

ml6rt_send_host_srq Sende direkten Service-Request

Pascal FUNCTION ml6rt_send_host_srq (data: WORD): WORD;

C USHORT ml6rt_send_host_srq (USHORT data);

Funktion Diese Funktion arbeitet grundsätzlich wie **ml6rt_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 sollte deshalb (im Gegensatz zu **ml6rt_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 **ml6rt_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

ml6rt_init_io — Initialisiere Multi-COM Karte

Pascal PROCEDURE ml6rt_init_io (device, options: BYTE);

C VOID ml6rt_init_io (UCHAR device, UCHAR options);

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

Parameter *device*: Gibt an, welche Hardware initialisiert werden soll:
- 0: Multi-COM Karte
- 1: Serielle Schnittstellen A und B (SCC1)
- 2: Serielle Schnittstellen C und D (SCC2)
- 3: Serielle Schnittstellen E und F (SCC3)

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).

ml6rt_local_led_on
ml6rt_local_led_off
ml6rt_external_led_on
ml6rt_external_led_off — Schalte LED ein/aus

Pascal
PROCEDURE ml6rt_local_led_on;
PROCEDURE ml6rt_local_led_off;
PROCEDURE ml6rt_external_led_on;
PROCEDURE ml6rt_external_led_off;

C
VOID ml6rt_local_led_on (VOID);
VOID ml6rt_local_led_off (VOID);
VOID ml6rt_external_led_on (VOID);
VOID ml6rt_external_led_off (VOID);

Funktion
Die Prozeduren schalten die LED auf der Basiskarte (local_led, LEDint) bzw. die an Stecker St3 angeschlossene LED (external_led, LEDext) ein bzw. aus.

ml6rt_trigger_watchdog — Triggere Watchdog

Pascal
PROCEDURE ml6rt_trigger_watchdog;

C
VOID ml6rt_trigger_watchdog (VOID);

Funktion
Diese Prozedur triggert den Watchdog der Multi-COM Karte. Nähere Informationen zum Watchdog finden Sie in Kapitel 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 **ml6rt_get_error_info** bestimmt werden. Wenn keine Fehlerbehandlungsprozedur installiert ist, geschieht beim Auftreten eines Fehlers nichts.

ml6rt_set_error_handler Lege Fehlerbehandlungsprozedur fest

Pascal PROCEDURE ml6rt_set_error_handler (errhandle: POINTER);

C VOID ml6rt_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.

ml6rt_get_error_info Fordere Fehlerinformation an

Pascal PROCEDURE ml6rt_get_error_info
(VAR err_rec: ML6RT_ERROR_INFO_TYPE);

C VOID ml6rt_get_error_info (ML6RT_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
subnum	Word	Nummer der 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 **ml6rt_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 *erroverflow* = 1. Erst nach einem Aufruf von **ml6rt_reset_error** wird der Fehlerrecord wieder neu beschrieben.

ml6rt_reset_error — Setze Fehlerrecord zurück

Pascal PROCEDURE ml6rt_reset_error;

C VOID ml6rt_reset_error (VOID);

Funktion Diese Prozedur setzt den Fehlerrecord zurück.

ml6rt_set_mark — Markiere eine Programmstelle für die Fehlerbehandlung

Pascal PROCEDURE ml6rt_set_mark (mark: WORD);

C VOID ml6rt_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.

ml6rt_force_error — Erzeuge eine Fehlermeldung

Pascal PROCEDURE ml6rt_force_error;

C VOID ml6rt_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 **ML6RTBIB** 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 als ASCII-Zeichen (z.B. 'A')
CCCCCCCC	Laufende Revisionsnummer (z.B. 03h)
DDDDDDDD	Status der Version als ASCII-Zeichen:

'F' : Freigegeben (Free Release)
'B' : Beta Version
'R' : RAM-Version (OsX)
'E' : EPROM-Version (OsX)

Format des Datecodes:

AAAAAAAA	Jahreszahl als Offset zu 1900 (0..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:

ml6rt_create_version_string — Versioncode in String umwandeln
ml6rt_create_date_string — Datecode in String umwandeln
ml6rt_create_time_string — Timecode in String umwandeln

Pascal

```
PROCEDURE ml6rt_create_version_string  (VAR v_string: STRING;
code: LONGINT);

PROCEDURE ml6rt_create_date_string  (VAR d_string: STRING;
code: LONGINT);

PROCEDURE ml6rt_create_time_string  (VAR d_string: STRING;
code: LONGINT);
```

C

```
VOID ml6rt_create_version_string (CHAR* v_string, ULONG code);

VOID ml6rt_create_date_string (CHAR* d_string, ULONG code);

VOID ml6rt_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

ml6rt_set_pdt_adr — Setze die PDT-Adresse

Pascal PROCEDURE ml6rt_set_pdt_adr (headadr: Pointer);

C void ml6rt_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 Programms nötig sind (Tasktyp, Interruptnummer ...). Deshalb muss diese Funktion immer im **Prepare**-Teil eines Echtzeitprogramms aufgerufen werden. (Siehe Kapitel 7)

Parameter *headadr*: PDT-Adresse (Segment:Offset).

ml6rt_real_adr — Rechne Adresse um: Physikalisch ⇒ SEG:OFFS

Pascal PROCEDURE ml6rt_real_adr (physadr: LONGINT; VAR realptr);

C VOID ml6rt_real_adr (ULONG physadr, VOID* realptr);

Funktion Diese Prozedur berechnet aus einer physikalischen Adresse eine Adresse vom Typ Segment:Offset. Die umgewandelte Adresse hat immer einen Offset, der kleiner 16 ist (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 }
...
ml6rt_real_adr($12345, BytePtr);       {BytePtr = $1234(SEG):$0005(OFFS)}
```

Derselbe Aufruf muß in C wie folgt aussehen:

```
UCHAR     *byteptr;              /* Pointer auf ein Byte */
...
ml6rt_real_adr(0x12345, &byteptr);   /* byteptr = 0x1234(SEG):0x0005(OFFS)*/
```

ml6rt_phys_adr — Rechne Adresse um: SEG:OFFS ⇒ Physikalisch

Pascal PROCEDURE ml6rt_phys_adr (realadr: POINTER;
VAR physadr: LONGINT);

C VOID ml6rt_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.

Namensgebung: In Assembler geschriebene Programme dürfen nicht die Dateierweiterung .EXE bekommen, andernfalls lassen sie sich nicht mit den mitgelieferten PC-Bibliotheken und Hilfsprogramme auf die Karte laden.

Folgende Routinen sind vorhanden (Betriebssystemversion ML6-1A.01x, x = R für Download-Version, x = E für EPROM-Version, x = B für Beta-Test-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	10-5
2	408h	INSTALL_TASK	Installiere Programm unter Task	10-5
3	40ch	GET_TASK_INFO	Info über eine Task (z.B. TDT, PDT)	10-7
4	410h	GET_TASK_STATUS	Melde, ob Task aktiviert ist	10-7
5	414h	WAKEUP_TASK	Aktiviere Task (1x)	10-8
6	418h	SLEEP_TASK	Deaktiviere Task (1x)	10-9
7	41ch	-	Reserviert	
8	420h	GET_FUNC_ADDRESS	Melde Adresse einer Funktion	10-9
9	424h	CALL_PROC	Prozedur einer Task aufrufen	10-10
10	428h	CALL_FUNC	Funktion einer Task aufrufen	10-10
11	42ch	GET_PAR_ADDRESS	Melde Adresse eines Parameters	10-12
12	430h	READ_PAR_BYTE	Lies Parameterbyte	10-12
13	434h	READ_PAR_WORD	Lies Parameterwort	10-12
14	438h	READ_PAR_DWORD	Lies Parameterdoppelwort	10-13
15	43ch	READ_PAR_BLOCK	Lies Parameterblock	10-13
16	440h	WRITE_PAR_BYTE	Schreibe Parameterbyte	10-14
17	444h	WRITE_PAR_WORD	Schreibe Parameterwort	10-14
18	448h	WRITE_PAR_DWORD	Schreibe Parameterdoppelwort	10-14
19	44ch	WRITE_PAR_BLOCK	Schreibe Parameterblock	10-15
20	450h	RESET_R_POINTER	Setze Daten-Lesezeiger auf Anfang	10-15
21	454h	RESET_W_POINTER	Setze Daten-Schreibzeiger auf Anfang	10-15
22	458h	MOVE_R_POINTER	Verschiebe Lesezeiger +/- Offset	10-16
23	45ch	MOVE_W_POINTER	Verschiebe Schreibzeiger +/- Offset	10-16
24	460h	READ_DATA_BYTE	Lies Datenbyte	10-16
25	464h	READ_DATA_WORD	Lies Datenwort	10-17
26	468h	READ_DATA_DWORD	Lies Datendoppelwort	10-17
27	46ch	READ_DATA_BLOCK	Lies Datenblock	10-17
28	470h	WRITE_DATA_BYTE	Schreibe Datenbyte	10-18
29	474h	WRITE_DATA_WORD	Schreibe Datenwort	10-18
30	478h	WRITE_DATA_DWORD	Schreibe Datendoppelwort	10-18
31	47ch	WRITE_DATA_BLOCK	Schreibe Datenblock	10-19
32	480h	ALLOCATE_RAM	Reserviere freien Speicher	10-19
33	484h	MASK_INT	Maskiere einen Interrupt	10-21
34	488h	UNMASK_INT	Demaskiere einen Interrupt	10-21
35	48ch	CLEAR_INT	Lösche einen Pending Interrupt	10-22
36	490h	END_OF_INT	Beende Interrupt-Service-Routine	10-22
37	494h	SET_INT_EDGE	Setze aktive Interrupt-Flanke	10-23
38	498h	-	Reserviert	
39	49ch	TRIGGER_WATCHDOG	Watchdog nachtriggern	10-23
40	4a0h	CACHE_CONTROL	Cache steuern	10-23

Nr.	Adr.	Name	Funktion (Kurzbeschreibung)	Seite
41	4a4h	LOCAL_LED_ON	LEDint (auf Karte) "ein"	10-24
42	4a8h	LOCAL_LED_OFF	LEDint (auf Karte) "aus	10-24
43	4ach	EXTERNAL_LED_ON	LEDext (extern) "ein"	10-24
44	4b0h	EXTERNAL_LED_OFF	LEDext (extern) "aus"	10-24
45	4b4h	GET_RTC_STATUS	Lies Status der Uhr	10-25
46	4b8h	SET_RTC_MODE	Initialisiere Uhr	10-26
47	4bch	GET_DATE_AND_TIME	Lies Datum und Uhrzeit	10-27
48	4c0h	SET_DATE_AND_TIME	Setze Datum und Uhrzeit	10-27
49	4c4h	GET_TIMER	Lies Timer (Zähler/Status)	10-28
50	4c8h	SET_TIMER	Setze Timer (Mode/Anfang)	10-28
51	4cch	READ_EEPROM_DIRECT	Lies Wort aus EEPROM direkt	10-29
52	4d0h	WRITE_EEPROM_DIRECT	Schreibe Wort in EEPROM direkt	10-30
53	4d4h	SEND_HOST_SRQ	Sende SRQ-Wort zum PC	10-31
54	4d8h	GET_TDT_ADDRESS	Lies Adresse einer TDT	10-32
56	4e0h	READ_EEPROM_COPY	Lies Wort aus EEPROM-Kopie	10-29
57	4e4h	WRITE_EEPROM_COPY	Schreibe Wort in EEPROM-Kopie	10-30
58	4e8h	GET_INT_TASK	Melde Task, die Interrupt nutzt	10-32
59	4ech	CONVERT_TIMER_DATA	Berechne Timer-Wert	10-32
60	4f0h	SEND_BUFFER_SRQ	Sendet einen gepufferten SRQ	10-33
65	504h	WAKEUP_TI_TASK	TI-Task aktivieren	10-34
70	518h	CREATE_BUFFER	Ringpuffer erzeugen	10-35
71	51ch	DELETE_BUFFER	Ringpuffer entfernen	10-36
72	520h	CLEAR_BUFFER	Ringpuffer komplett leeren	10-36
73	524h	GET_BUFFER_STATUS	Anzahl Zeichen im Ringpuffer bestimmen	10-37
74	528h	WRITE_BUFFER_BYTE	Ein Byte in Ringpuffer schreiben	10-37
75	52ch	WRITE_BUFFER_WORD	Ein Wort in Ringpuffer schreiben	10-37
76	530h	WRITE_BUFFER_DWORD	Ein Doppelwort in Ringpuffer schreiben	10-37
77	534h	WRITE_BUFFER_BLOCK	Block in Ringpuffer schreiben	10-38
78	538h	WRITE_BUFFER_MAX	Block in Ringpuffer schreiben (max.)	10-38
79	53ch	READ_BUFFER_BYTE	Ein Byte aus Ringpuffer lesen	10-39
80	540h	READ_BUFFER_WORD	Ein Wort aus Ringpuffer lesen	10-39
81	544h	READ_BUFFER_DWORD	Ein Doppelwort aus Ringpuffer lesen	10-39
82	548h	READ_BUFFER_BLOCK	Block aus Ringpuffer lesen	10-39
83	54ch	READ_BUFFER_MAX	Block aus Ringpuffer lesen (max.)	10-40
84	550h	VIEW_BUFFER_BLOCK	Block aus Ringpuffer kopieren	10-40
85	554h	VIEW_BUFFER_MAX	Block aus Ringpuffer kopieren (max.)	10-41
90	568h	GET_TASK_NUMBER	Tasknummer zu Programmnummer ermitteln	10-41
93	574h	INIT_IO	Init. der ser. Schnittstellen oder der Basiskarte	10-44

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.

Entry:	eax	Physikal. Adresse (32 Bit) des Records
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 (16 bis 1023, Task 0 bis 20 reserviert)
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 Anhang
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:	CY, 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-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)). Bei DI- und II-Tasks wird der zugehörige Interrupt demaskiert.

Außer dem Aktivieren kann eine NI-Task mit dieser Subroutine 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 der Aktivierung einer entsprechenden Prüfung (ax = 3) wird, wenn bereits eine Task vorgezogen, aber noch nicht ausgeführt wurde, eine Fehlermeldung zurückgeliefert (ax = 20d4h). Falls die vorgezogene Task nicht aktiviert war, wird sie nur ein einziges Mal aufgerufen und dann wieder deaktiviert.

Entry:	dx	Tasknummer
	ax = 1	Aktivieren der Task
	ax = 2	Einmaliges Vorziehen der Task (in jedem Fall)
	ax = 3	Einmaliges Vorziehen der Task mit Prüfung, ob bereits eine 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 bzw. eine NI-Task einmal. Wenn eine NI-Task mehrfach aktiviert wurde, muß sie auch entsprechend oft wieder deaktiviert werden, damit sie inaktiv ist. Bei Interrupt-Tasks (DI- und II-Tasks) wird der zugehörige Interrupt maskiert.

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 Programms 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 (physikalisch)
Exit (CY=1):	ax	Fehlercode

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

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

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

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-TEMP (Temperatursensor)
		124 (= 7ch): IRQ-SCC3 (ser. Schnittstelle E und F)
		125 (= 7dh): IRQ-SCC2 (ser. Schnittstelle C und D)
		126 (= 7eh): IRQ-SCC1 (ser. Schnittstelle A und B)
		127 (= 7fh): IRQ-FAN (Lüfterüberwachung)
		144 (= 90h): SCC (Kanal A und B)
		145 (= 91h): Timer-A
		146 (= 92h): Timer-B
		147 (= 93h): Timer-C
		148 (= 94h): Timer-D (Uhr)
		149 (= 95h): IRQ-G (ext. Eingang an St3)
		150 (= 96h): IRQ-H (ext. Eingang an Ri/B)
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.

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 Interrupt-Eingänge. Die Interrupts sind flankengetriggert.

Entry:	al	Interrupt-Nr.:
		121 (= 79h): IRQ-TEMP (Temperatursensor)
		127 (= 7fh): IRQ-FAN (Lüfterüberwachung)
		149 (= 95h): IRQ-G (ext. Eingang an St3)
		150 (= 96h): IRQ-H (ext. Eingang an Ri/B)
	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. Diese Subroutine kann durch Setzen von Betriebssystemparameter 424 deaktiviert werden.

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, eax	
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 (LEDint) auf der Karte ein. Das Steuersignal ist an Stecker St3 herausgeführt.

Entry:	-	
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

LOCAL_LED_OFF (Nr. 42)

Diese Subroutine schaltet die on-board LED (LEDint) auf der Karte aus. Das Steuersignal ist an Stecker St3 herausgeführt.

Entry:	-	
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

EXTERNAL_LED_ON (Nr. 43)

Diese Subroutine schaltet die externe LED (LEDext) ein. Das Steuersignal ist an Stecker St3 herausgeführt.

Entry:	-	
Changed:	f, ax	
Exit (CY=0):	-	
Exit (CY=1):	ax	Fehlercode

EXTERNAL_LED_OFF (Nr. 44)

Diese Subroutine schaltet die externe LED (LEDext) aus. Das Steuersignal ist an Stecker St3 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	1 = Uhr gestellt
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 (1/15,625 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	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 99 (00 bis 63h)

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.

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.

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 Multi-COM Karte (siehe Anhang L). 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	Device:	
		0 = Multi-COM (Basiskarte)	
		1 = ser. Schnittstelle A und B (SCC1)	
		2 = ser. Schnittstelle C und D (SCC2)	
		3 = ser. Schnittstelle E und F (SCC3)	
Changed:	f, ax		
Exit (CY=0):	ax	Data (16 Bit)	
Exit (CY=1):	ax	Fehlercode:	10e0h = Falscher Parameter bei Aufruf
			15e1h = Device nicht vorhanden

WRITE_EEPROM_DIRECT (Nr. 52)
WRITE_EEPROM_COPY (Nr. 57)

WRITE_EEPROM_DIRECT schreibt ein Wort direkt in ein EEPROM der Multi-COM Karte. 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	Device:	
		0 = Multi-COM (Basiskarte)	
		1 = ser. Schnittstelle A und B (SCC1)	
		2 = ser. Schnittstelle C und D (SCC2)	
		3 = ser. Schnittstelle E und F (SCC3)	
	bx	Data (16 Bit)	
Changed:	f, ax		
Exit (CY=0):	-		
Exit (CY=1):	ax	Fehlercode:	10e0h = Falscher Parameter bei Aufruf
			15e1h = Device nicht vorhanden

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 im EEPROM ein Interrupt-Kanal angegeben 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 Multi-COM (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 (1 MHz, 2,5 MHz oder 10 MHz), 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	Schnittstelle: al = 1: PC-Schnittstelle
	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, 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" 8490h = "Host schon/noch installiert" 8491h = "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-Tic, 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-Tic) 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-Tics 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-Tics
	edi	Hold-Off-Zeit: Verzögerung vom Aktivieren bis zum ersten Aufruf der Task, in Anzahl Timer-Tics
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 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_BYTE, 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 Multi-COM Karte bzw. die ser. Schnittstellen (SCC 1 bis 3) entsprechend den im EEPROM angegebenen Werten.

Entry:	al	Device: 0 = Multi-COM Karte 1 = ser. Schnittstelle A und B (SCC1) 2 = ser. Schnittstelle C und D (SCC2) 3 = ser. Schnittstelle E und F (SCC3)
	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	Device wurde nicht initialisiert wegen: Fehlercode: 1be0h: "Falsches Device" 10e0h: "falscher Parameter bei Aufruf" 1ce0h: "EEPROM-Info falsch" 15e1h: "Device nicht vorhanden" Warnungen: 80d4h: "Falsches Device" 80d4h: "Device braucht keine Initialis." 82d4h: "Bit-0 in EEPROM WORT-1=0 83d4h: "keine Init-Subroutine in OsX"

10.2. Beispiel: LED-Blinkprogramm in Assembler

Zur Funktion des Programms M6P0380 (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 M6P0380 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 M6P0380 ist möglichst einfach gehalten. Es soll die Programmierung einer einfachen NI-Task auf der Multi-COM 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:

```
M6INST M6P0380.LIB 0380 0010 00 000000 00000000 ; Installiere Task 10h
M6CMD 41 02 10 00                               ; Aktiviere Task 10h
M6PAR 10 00 01                                  ; Setze Par. 0=1: Start
```

```
;-----------------------------------------------------------------------------
;     M6P0380.ASM
;-----------------------------------------------------------------------------
;     Programm 0380h: Blink LED als NI-Task
;-----------------------------------------------------------------------------
;
;-----------------------------------------------------------------------------
      TITLE M6P0380
;-----------------------------------------------------------------------------
      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

;-----------------------------------------------------------------------------
;     M6P0380: Programm-Deskriptor-Tabelle (PDT)
;-----------------------------------------------------------------------------
M6P0380:
      DB    TYPPDT            ; 0   ; Typ der PDT
      DB    DEFLPDT           ; 1   ; Laenge des Vorspanns der PDT
      DW    (M6P0380E - M6P0380 - 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 M6P0380_M; 48    ; Adresse der Main-Prozedur (Nr.0)
      DW    0
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DW    OFFSET M6P0380_I; 52    ; Adresse der Auto-Init Prozedur (Nr.1)
      DW    0
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DW    OFFSET M6P0380_2; 56    ; Adresse Prozedur LED_ON (Nr.2)
      DW    0
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
      DW    OFFSET M6P0380_3; 60    ; Adresse Prozedur LED_OFF (Nr.3)
      DW    0
;- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

M6P0380E:

;-----------------------------------------------------------------------------
;     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 M6P0380_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, also
            jmp   P0380_END

ENDP  M6P0380_M
```

```
;-----------------------------------------------------------------------------
;     Auto-Init Prozedur (= Prozedur 1): Parameter auf Angangs-Zustand
;-----------------------------------------------------------------------------
;     Vom Betriebssystem wird in dx bei Aufruf einer Prozedur immer die
;     Task-Nr uebergeben, das wird aber hier nicht ausgewertet.

PROC  M6P0380_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
M6P0380_I1: mov   al,[BYTE ds:bx]
            mov   [BYTE ds:di],al
            inc   di
            inc   bx
            dec   cx
            jnz   M6P0380_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  M6P0380_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  M6P0380_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  M6P0380_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  M6P0380_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  M6P0380_3

;-----------------------------------------------------------------------------
      END M6P0380
;-------------------------- END OF M6P0380.ASM -------------------------------
```

Notizen

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 (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 Multi-COM 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 Multi-COM 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 Multi-COM Karte belegt 8 aufeinanderfolgende Adressen im I/O-Adreßbereich des PC. Die Basisadresse (im folgenden mit "BA" bezeichnet) wird auf der Karte mit einem Drehschalter (S1) oder Steckbrücken eingestellt.

Die Übergabe von Daten vom PC an die Multi-COM 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 Multi-COM Karte als auch vom PC gelesen werden kann. Wenn das Wort von der Multi-COM Karte ausgelesen wird, wird dieses Status-Bit wieder zurückgesetzt. Die Schnittstelle ist damit wieder frei zur Übertragung des nächsten Wortes.

Das Betriebssystem auf der Multi-COM 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 Multi-COM 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 Multi-COM 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 Multi-COM Karte wird mit Drehschalter S1 bzw. Jumper J1 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** (DLM=0) und **RESTART**
PBA+1	W8x	**Setzen von DLM** (DLM=1)
PBA+2	R16	**Lesen von Daten**, die von der Multi-COM 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 Multi-COM 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	Reserviert
PBA+3	W8x	**Hardware-Reset der Multi-COM Karte.** DLM und DLP werden = 0 gesetzt, RESTART = 1.

Die Beschreibung der Schnittstelle bezieht sich auf Multi-COM Karten mit Standardbestückung. Die PC-Schnittstelle der Multi-COM Karte kann im Betrieb umkonfiguriert werden, um weitere Funktionen zu erhalten (z.B. 'Plug and Play').

11.3. Das Status-Register (8 Bit)

Bit 0: **PC-TBF**	Status-Bit für die Schnittstelle vom PC zur Multi-COM Karte: 0 = Schnittstelle ist frei 1 = Schnittstelle nicht frei
Bit 1: **DLM**	Device Locking-Bit Multi-COM: 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 Multi-COM Karte für andere PC-Tasks zu blockieren. Der Zustand dieses Bits kann auch von der Multi-COM aus abgefragt werden.
Bit 2: **RESET**	Reset (1 = Reset ist aktiv)
Bit 3: **BOOT**	Boot (0 = Karte hat kein EPROM)
Bit 4:	Reserviert
Bit 5: **RESTART**	Restart
Bit 6: **DLP**	Device Locking-Bit PC : Flag, das von der Multi-COM 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 Multi-COM Karte zum PC: 0 = Schnittstelle ist leer 1 = Schnittstelle enthält ein Byte/Wort
Bit 8 bis Bit 15: Zur Zeit nicht verwendet, undefinierter Zustand	

11.4. Beispiel für eine einfache Kommunikation

Die Multi-COM Karte soll den Zustand der Kontroll-LED (on-board LED) ermitteln und das Ergebnis an den PC übergeben.

(PBA = Basisadresse der Multi-COM Karte)

Schritt 1: Status-Registers lesen (I/O-Adresse = PBA+0).
Bit 0 = 1: Schritt 1 wiederholen.
Bit 0 = 0: die Multi-COM 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 Multi-COM sind in Kapitel 11 beschrieben. Alle im folgenden aufgeführten Makrobefehle sind im Betriebssystem der Karte enthalten. Sie sind nur für jene 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, wie viel sie davon nicht verwendet hat und meldet dies in w zurück. Bezüglich der Antwort wird im Befehl vorgegeben, wie viel 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.

12.2. Kommunikationsbefehle PC - Multi-COM

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 Multi-COM 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: Multi-COM Karte ist bereit für Kommunikation
01h 20h	**Anforderung: Typ der Karte und Betriebssystem melden**
01h 0xh	Antwort: Die Multi-COM Karte sendet 01h 0xh zurück, andere Karten (z.B. MODULAR-4/Z80) 04h. Das High-Byte kann bei der Multi-COM 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): Startadresse phys.
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): Startadresse phys.
24h 02h 55h aah	**Start Programm ab Adresse des Pointers**

Code/Data	Funktion
22h 52h	**Anforderung: Freien Speicherplatz melden**
00h z	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 setzen, Daten lesen und R-Pointer um n inkrementieren
- W-Pointer relativ zum Anfang 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. Die Programmnummer erscheint auch als 4-stellige hexadezimale Zahl im Namen von Programmdateien und in dazugehörigen Dateien:

M6Pnnnn.LIB = Anwenderprogramm (identisch mit .OBJ)
M6Pnnnn.LAB = Relocated Anwenderprogramm
M6Pnnnn.SEX = SORCUS EXE Datei (identisch mit .EXE)
M6Pnnnn.SRX = SORCUS Relocated EXE Datei
M6Pnnnn.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?

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 Multi-COM 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 deaktiviert 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 mit den Blockbefehlen sind die Parameter nicht vor dem gleichzeitigen Zugriff z.B. durch eine andere Interrupt-Task geschützt. Umgekehrt kann der PC-Makrobefehl eine andere Task unterbrechen, die ebenfalls gerade auf dieselben Parameter zugreift, sofern diese Task das nicht unterbindet, z.B. durch vorübergehendes Sperren des maskierbaren Interrupts.

Code/Data	Funktion
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
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

Code/Data	**Funktion**
5dh 06h	**Parameter Wort schreiben**
tL tH	t = Task-Nummer
pL pH	p = Nummer des zu schreibenden Parameters
wL wH	w = Parameter Wort
5eh 08h	**Parameter Doppelwort schreiben**
tL tH	t = Task-Nummer
pL pH	p = Nummer des zu schreibenden Parameters
dE dF dG dH	d = Parameter Doppelwort
4dh 0fh	**Parameter Block schreiben**
nL nH	n = Anzahl zu schreibender Parameterbyte + 4
tL tH	t = Task-Nummer
pL pH	p = Nummer des ersten zu schreibenden Parameters
b1...bn-4	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, wie viel Byte sie davon verwerten konnte. Sie beginnt mit der Verwertung sequentiell ab dem zuerst übergebenen Parameterbyte.

Beim Aufruf kann außerdem angegeben werden, wie viele Parameter (Anzahl Byte) von der Funktion maximal zurückerwartet werden. Die Funktion meldet immer zurück, wie viel 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

[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. Eine einfache Methode, solche Konflikte zu verhindern, besteht darin, den maskierbaren Interrupteingang der CPU vorübergehend zu maskieren.

Code/Data	**Funktion**
46h 02h tL tH	**Setze R-Pointer auf den Anfang des Datenbereichs** t = Task-Nummer
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

Code/Data	Funktion
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)
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 m + o.
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) t = Task-Nummer (Blockgröße = n - 2 Byte)

Code/Data	Funktion
57h 0fh	**Setze W-Pointer und schreibe Datenblock**
nL nH	Blockgröße = n - 6 Byte
tL tH	t = Task-Nummer
oE oF oG oH	o = Offset zum Anfang des Datenbereichs
b1...bn-6	Der W-Pointer steht anschließend auf n - 6 + o.

12.7. System-Call: Taskinformationen (privilegierter Befehl)

Code/Data	Funktion
2fh 54h	**Anforderung: Task-Info**
tL tH	Task-Nr. (t = 0: Betriebssystem)
nL nH	Call-Nr. (Erklärung siehe Anhang H)
2fh	Antwort:
aE aF aG aH	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)
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)

Code/Data	Funktion
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
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

Code/Data	Funktion
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)
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)

Code/Data	Funktion
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	Multi-COM Karte (m=0), ser. Schnittstellen A/B (m=1), C/D (m=2) oder E/F (m=3)
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	Multi-COM Karte (m=0) , ser. Schnittstellen A/B (m=1), C/D (m=2) oder E/F (m=3)
a	Rel. Adresse des zu schreibenden 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).

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 Beschreibung zu J5, Kapitel 2).

Code/Data	Funktion
34h 20h	**Lesen des Status der Uhr**
34h b	Antwort: b = Status der Uhr: Bit 0: 1= Uhr gestellt 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 99)
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 99)
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)

12.11. Kontroll-LEDs

Code/Data	Funktion
30h (00h)	**on-board LED ein (LEDint)**
31h (00h)	**on-board LED aus (LEDint)**
33h 20h	**Anforderung: Zustand der on-board LED (LEDint)**
33h b	Antwort: Zustand der LED (Bit 0: 0 = aus, 1 = ein)
23h 03h 50h 00h 00h	**externe LED ein (LEDext)**
23h 03h 51h 00h 00h	**externe LED 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 der Multi-COM Karte

Code/Data	Funktion
3bh 22h	**Initialisiere Multi-COM Karte bzw. die ser. Schnittstellen entsprechend dem Inhalt des EEPROMs**
m	Initialisierung von: m = 0: Multi-COM Karte m = 1 bis 3: Serielle Schnittstellen A/B, C/D oder E/F
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: Schnittstelle nicht vorhanden s=81h: Schnittstellen benötigen keine Initialisierung s=82h: Schnittstellen wurden nicht initialisiert wg. EEPROM-WORT-1 Bit 0 = 0 s=83h: Initialisierungsroutine nicht implementiert

Notizen

13. Asynchrone serielle Kommunikation

Das Multi-COM System bietet 6 universelle serielle Schnittstellen. Die Intelligenz der Multi-COM Karte wird in diesem Aufgabenfeld zum einen für die Pufferung der Daten und zum anderen für die Abarbeitung beliebiger Protokolle benutzt. Beides läuft 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 Multi-COM 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.

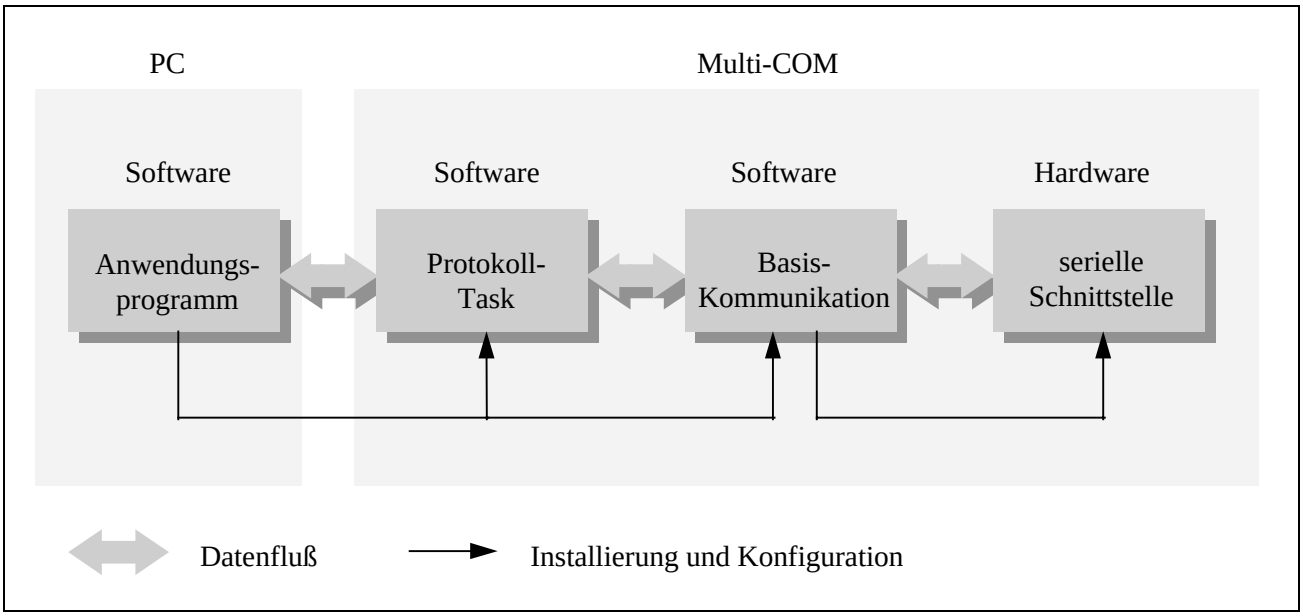


Abb. 13-1: Grundstruktur der asynchronen seriellen Kommunikation

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 die Daten direkt mit den Tasks der Basiskommunikation aus.

Um die mitgelieferten Kommunikationsprogramme verwenden zu können, muss auf der Multi-COM Karte ein Betriebssystem mit der Version ML6-1A.01E (bzw. ML6-1A.01R) oder größer laufen. Diese Beschreibung bezieht sich auf die Version 2.I 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 serielle Schnittstelle setzt sich aus mehreren Programmen bzw. Tasks zusammen. Die Installierung der Programme geschieht am einfachsten mit SNW6 oder SNW32.

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. Andere Protokolle

Die Basiskommunikation von SORCUS bietet sog. Aktions-Filter an, die zur Implementierung eigener Protokolle verwendet werden können. Bei bestimmten Zuständen von Steuerleitungen oder bei Empfang spezieller Zeichen können beliebige Funktionen aufgerufen werden. Das XON/XOFF-Handshaking basiert z.B. auf einem Aktions-Filter.

Kapitel 13.6 beschreibt die Verwendung von Aktions-Filtern.

13.1.4. 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 Bibliothek ML6BIB oder mit Betriebssystemaufrufen von ML6RTBIB durchgeführt werden. Alle in diesem Kapitel beschriebenen Aufrufe beziehen sich auf die Kommunikationstask des jeweiligen Kanals. Die Tasknummern können im Prinzip frei gewählt werden; empfohlen

wird jedoch folgende Zuordnung, die standardmäßig auch beim Installieren mit SNW6 bzw. SNW32 eingehalten wird:

Schnittstelle	**Interruptkanal**	**Tasknummer des Interrupt-Managers**	**Tasknummer der Kommunikationstask**
A	IRQ-D (7eh)	301h	318h
B			319h
C	IRQ-C (7dh)	302h	320h
D			321h
E	IRQ-B (7ch)	303h	328h
F			329h
A[1] (optional)	SCC (90h)	300h	310h
B (optional)			311h

Tab. 13-1: Verteilung von Tasknummern und Interruptnummern bei Installierung der Basiskommunikation mit SNW6 bzw. SNW32

Schnittstelle	**SCC-Baustein (Modul[1, 2])**	**SCC-Kanal**
A	1	0
B	1	1
C	2	0
D	2	1
E	3	0
F	3	1

Tab. 13-2: Zuordnung der Kanäle zu den Kommunikationsbausteinen.

[1] Die Basiskommunikation für die Schnittstellen A und B kann auch unter Interrupt 90h installiert werden (SCC0, Kompatibilität mit MODULAR-4/486).

[2] SNW6 verwendet die Bezeichnung 'Modul' für die Auswahl eines SCC-Bausteins.

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 (scc, scc_channel: BYTE; data: STRING): BYTE;
VAR
  dummyin, dummysize : BYTE;
  error : BYTE;
  task  : WORD;
BEGIN
  task := $310 + $08 * scc + scc_channel;

  { Die Sendedaten werden mit data[1] übergeben, da bei einer }
  { Übergabe mit data auch das erste Byte der Stringvariablen }
  { (= Stringlänge) übertragen würde.                         }

  ml6_call_func(task,7,LENGTH(data),data[1],0,dummysize,dummyin,error);

  send_string := error;
END;
```

C:

```
byte send_string (byte scc, byte scc_channel, char *data)
{
  void   *dummyin;
  ushort dummysize;
  byte   error;
  ushort task;

  task = 0x310 + 0x08 * scc + scc_channel;

  ml6_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 (scc, scc_channel: BYTE; VAR data: STRING): BYTE;
VAR
  dummyout : BYTE;
  insize   : WORD;
  error    : BYTE;
  task     : WORD;
BEGIN
  task := $310 + $08 * scc + scc_channel;

  { Die Empfangsdaten werden nicht an data, sondern an data[1] }
  { geschrieben, da das erste Byte von Data die Stringlänge    }
  { enthält.                                                   }

  ml6_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 scc, byte scc_channel, char *data)
{
  void   *dummyout;
  ushort insize;
  byte   error;
  ushort task;

  task = 0x310 + 0x08 * scc + scc_channel;

  ml6_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 seriellen Schnittstellen der Multi-COM Karte. Sie setzen auf der Basiskommunikation auf und können teilweise auch mit SNW6 bzw. SNW32 installiert werden. Das Senden und Empfangen von Daten wird bei jedem Protokoll unterschiedlich gehandhabt.

13.3. Installieren mit SNW6

Mit Hilfe von SNW6 kann das Installieren der Basiskommunikationsprogramme und zum Teil auch der Protokollprogramme sehr einfach und komfortabel geschehen. Voraussetzungen dafür sind lediglich, dass sich die Multi-COM Karte 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 SNW6 bzw. SNW32.

13.3.1. Gesamtkonfiguration

Die Gesamtkonfiguration wird in einem Fenster angezeigt, das mit **Konfiguration öffnen** geöffnet wird (siehe Abb. 13-2). Es enthält die Einstellungen aller verfügbaren Schnittstellen einer Karte. 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.

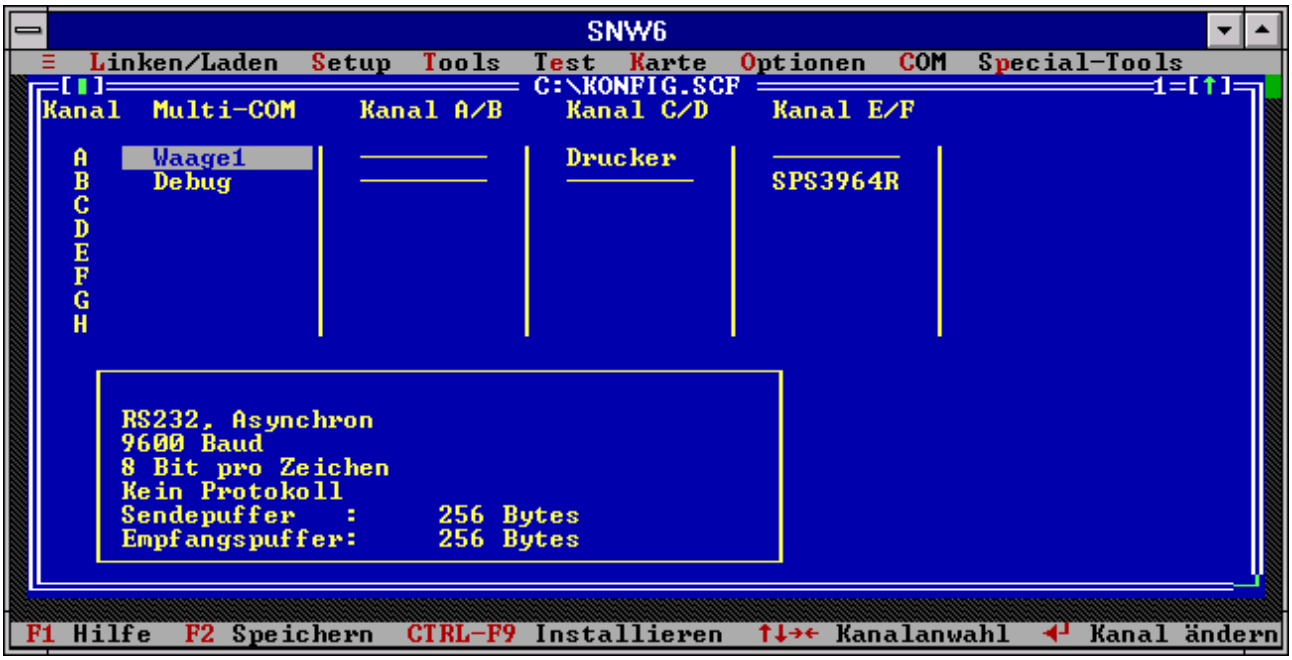


Abb. 13-2: Konfigurationsfenster in SNW6

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.

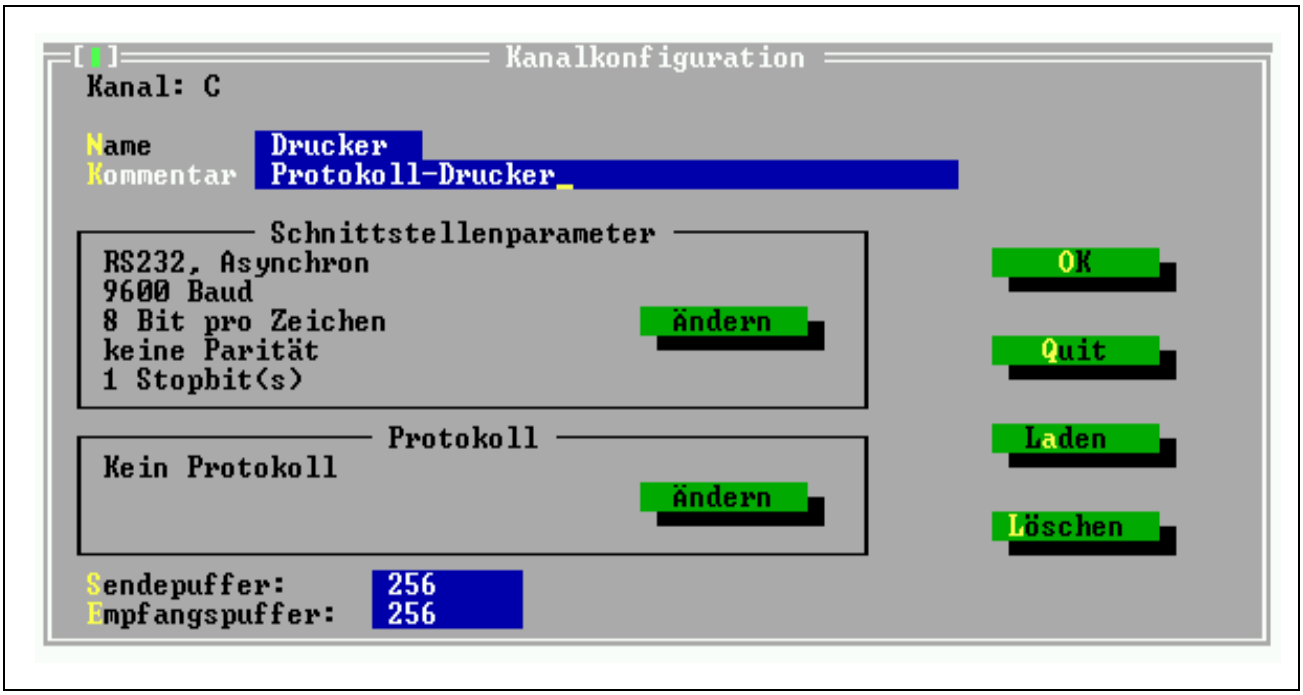


Abb. 13-3: Kanaleinstellungen in SNW6

- **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.

- **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 Schnittstellen bzw. Multi-COM Karten zur Verfügung stehen (z.B. unterschiedliche Baudraten für Senden und Empfangen) und die üblicherweise nicht verwendet oder geändert werden. 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 Multi-COM Karte ist anschließend bereit zum Senden und Empfangen von Daten; alle Schnittstellen werden der Gesamtkonfiguration entsprechend eingerichtet. Für das Installieren 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. SNW6 geht beim Laden davon aus, dass sich hier auch die Kommunikationsprogramme der Multi-COM 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 SNW6-Umgebung gestartet wird (z.B. in AUTOEXEC.BAT):

SNW6 /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 SNW6 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 SNW6, 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 Testprogramms 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

Jeder Kommunikationsbasutein (SCC) hat 2 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 jeder SCC nur eine einzige Interruptleitung zur Multi-COM 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 jeweiligen Interrupt des SCC installiert und als **Interruptmanager** bezeichnet wird.

Die Basiskommunikationsprogramme bestehen aus einem Interruptmanager (Programm M6P0500.LIB) und einem Kommunikationsprogramm (Programm M6P0520.LIB), die für die Schnittstellen der Multi-COM Karte geeignet sind. Pro SCC muss eine Interruptmanager-Task installiert sein und für jeden verwendeten Kanal eine Kommunikations-Task.

Die Installierung aller notwendigen Tasks nimmt SNW6 vor. 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 M6P0520.LIB

Das Kommunikationsprogramm M6P0520.LIB bedient **eine** serielle Schnittstelle des seriellen Schnittstellenbausteins auf der Multi-COM. Dies betrifft sowohl die Initialisierung der Schnittstelle als auch den laufenden Betrieb. Es ermöglicht den Zugriff auf alle für eine 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.

13.4.1.1. Initialisierung

Mit SNW6 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 Programms M6P0520.LIB

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	SCC (1 bis 3 = Schnittstellen A/B, C/D oder E/F, 0 = Basiskarte A/B, alternativ)
9*	Byte	0	R/W	SCC-Kanalnummer (0 oder 1)
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) 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 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 = Sendepuffer (Software) ist leer Bit 1 = alle Zeichen gesendet (Hardware) Bit 2 bis 14 sind reserviert Bit 15 = 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 Basiskommunikationsprogramms

Das Basiskommunikationsprogramm umfasst globale Prozeduren (ohne Übergabeparameter und Antwort) und globale Funktionen (mit Übergabe von Parametern und Antwort). 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 ML6BIB und ML6RTBIB aufgerufen wird.

Nr.	Typ	Bedeutung der Funktion
2	P	Start/Restart der Kommunikation (Konfiguration, Speicherreserv.)
4	P	Baudrate aus Parameter 24 und 34 übernehmen und einstellen
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 Bit 3 = DSR Bit 4 bis 15 reserviert

Fehlerrückgabecodes von Funktionen des Programms M6P0520

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 26h	Sendepuffer gesperrt Sendepuffer-Nummer ungültig
6	26h	Sendepuffer-Nummer ungültig
7	22h oder 23h 24h 26h	Sendepuffer gesperrt Sendepuffer voll Sendepuffer-Nummer ungültig
15	22h oder 23h 26h	Empfangspuffer gesperrt Empfangspuffer-Nummer ungültig
16	26h	Empfangspuffer-Nummer ungültig
17	22h oder 23h 26h	Empfangspuffer gesperrt 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 SNW6

Für die Installation der Basiskommunikation ohne SNW6 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 jeden verwendeten SCC der Multi-COM Karte muss jeweils ein Interrupt-Manager (Programm 500) unter dem von dem SCC 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 13.1.4 und 13.4.1.2.
- Hinweise zum Senden: siehe Kapitel 13.1.4 und 13.4.1.3.

13.6. Die Aktions-Filter des Programms M6P0520.LIB

Das Basis-Kommunikations-Programm M6P0520 kann unter bestimmten, einstellbaren Bedingungen Funktionen anderer Tasks aufrufen. Die Bedingungen stellen Filter für die Ausführung einer Aktion dar. Wir sprechen daher von Aktions-Filtern. Die Übergabe-Konvention für die aufgerufene Funktion entspricht der in Kapitel 10 (System-Subroutinen) bei CALL_FUNC und in Kapitel 9 bei ml6rt_call_func beschriebenen. Daher können Funktionen einer anderen Task abhängig von Ereignissen, die in der Basiskommunikation aufgetreten sind, aufgerufen werden. Dieser Mechanismus ist in erster Linie zur Implementierung von Protokollen vorgesehen. Zur Zeit sind 10 Filter-Aktionen definiert. Sie können mit dem Parameter "Aktions-Filter-Maske" vom Programm einzeln aktiviert oder unterdrückt werden. Jedem Aktions-Filter ist ein Parameter-Satz von 12 Byte Länge zugeordnet (Parameter siehe Kapitel 13.6.2). Dieser enthält die Nummer (2 Byte) der Task, die die aufzurufende Funktion enthält, und die Nummer der Funktion (2 Byte), die bei zutreffender Bedingung aufgerufen werden soll. Es folgt ein 4 Byte langes Argument, das von Filter-Aktion zu Filter-Aktion unterschiedliche Bedeutung hat. Danach kommt die Funktionsadresse der aufzurufenden Funktion im Format Segment:Offset (4 Byte). Diese Adresse wird beim Aufruf der Startfunktion (Prozedur 2) vom Programm M6P0520 berechnet, wenn das zugehörige Bit in der Aktions-Filter-Maske gesetzt ist. Die Task, die die aufzurufende Funktion enthält, muss zu diesem Zeitpunkt bereits installiert sein.

Der aufgerufenen Funktion wird eine Struktur mit diversen Status-Informationen übergeben (s.u.). Je nach Filter-Aktion muss eine Funktion einen Rückgabewert liefern. Der Aufbau dieser Funktion kann wie folgt aussehen (siehe auch Kapitel 9):

in C: **VOID PFAR name (USHORT task, USHORT insize, VOID *inptr, USHORT outsize, VOID *outptr);**

in Pascal: **PROCEDURE name (task, insize : Word; VAR inptr; outsize: Word; VAR outptr);**

Parameter
- *task*: Nummer der eigenen Task.
- *insize*: Anzahl der Daten, die der Funktion übergeben werden (= 19).
- *inptr*: Zeiger auf die Übergabedaten (Struktur *p0520_status*).
- *outsize*: Maximale Anzahl der Daten, die zurück erwartet werden (in Byte).
- *outptr*: Zeiger auf den Bereich, in dem die Task ihre Daten zurückgibt (Rückgabewert).

Bei Aufruf der Funktion aus einem Assembler-Programm sind die Register wie folgt vorbesetzt:

dx = Tasknummer der aufzurufenden Task
di = Länge der Übergabedaten (=19)
si = Maximale Anzahl der Daten, die zurück erwartet werden (in Byte).
eax = Adresse (physikalisch) der Übergabedaten (Struktur *p0520_status*)
ecx = Adresse (physikalisch) des Rückgabewertes
(Aktions-Filter 0 und Aktions-Filter 1)

Die Struktur *p0520_status* ist ein Ausschnitt aus dem Parameterbereich des Programms M6P0520.LIB. Die genaue Bedeutung der einzelnen Variablen entnehmen Sie bitte der Tabelle der Parameter des Programms M6P0520.LIB (Parameter 256 bis 271) . Hier die Struktur-Definition in C-Syntax:

```
struct _p0520_status {
   ushort sstatus;        // Sendestatus
   ulong  scount;         // Anzahl Zeichen im Sendepuffer
   ushort rstatus;        // Empfangsstatus
   ulong  rcount;         // Anzahl Zeichen im Empfangspuffer
   ushort control_status; // Zustand der Steuerleitungen
   byte   last_received;  // zuletzt empfangenes Zeichen(nur
                          // Filter-Aktion 1 + 2)
   ulong  s_ex_count;     // Anzahl Zeichen im Sende-Expreßpuffer
} p0520_status;
```

13.6.1. Aktions-Filter und zugehörige Filter-Argumente

In Parameter 56 von Programm 520 (siehe Kapitel 13.6.2) können z.Zt. 10 verschiedene Aktions-Filter ausgewählt werden.

Aktions-Filter 0

Bei jedem empfangenen Zeichen soll die zugeordnete Funktion aufgerufen werden.

Das Argument hat keine Bedeutung.

Im Rückgabewert (Byte) der Funktion wird festgelegt, ob das empfangene Zeichen nachträglich noch im Empfangs-Puffer gespeichert wird (= 1) oder nicht (= 0).

Aktions-Filter 1

Die zugeordnete Funktion wird aufgerufen wenn ein empfangenes Zeichen sich in einem Vergleichsstring befindet. Das erste Zeichen im Vergleichsstring enthält die Anzahl der nachfolgenden zu vergleichenden Zeichen.

Das Argument enthält den Zeiger (Format Segment:Offset) auf den Vergleichsstring.

Im Rückgabewert (Byte) der Funktion wird festgelegt, ob das empfangene Zeichen nachträglich noch im Empfangs-Puffer gespeichert wird (= 1) oder nicht (= 0).

Aktions-Filter 2

Die zugeordnete Funktion wird bei Empfangsfehlern aufgerufen.

Im Argument (untere 2 Byte) können Empfangsfehler maskiert werden. Die Bedeutung der Maske entspricht der Beschreibung des Parameters "Empfangsstatus". Ein gesetztes Bit (=1) führt zum Aufruf der Funktion, ein nicht gesetztes unterbindet den Aufruf.

Es gibt keine Rückgabewort.

Aktions-Filter 3

Funktionsaufruf bei Empfangs-Füllstands-Überschreitung.

Das Argument enthält die obere Empfangs-Füllstands-Grenze.

Es gibt keine Rückgabewort.

Wenn dieses Filter benutzt wird, muß auch Filter 4 benutzt werden.

Aktions-Filter 4

Funktionsaufruf bei Empfangs-Füllstands-Unterschreitung.

Das Argument enthält die untere Empfangs-Füllstands-Grenze.

Es gibt keine Rückgabewort.

Wenn dieses Filter benutzt wird, muß auch Filter 3 benutzt werden.

Aktions-Filter 5

Funktionsaufruf bei Sende-Füllstands-Überschreitung

Das Argument enthält die obere Sende-Füllstands-Grenze.

Es gibt keine Rückgabewort.

Wenn dieses Filter benutzt wird, muß auch Filter 6 benutzt werden.

Aktions-Filter 6

Funktionsaufruf bei Sende-Füllstands-Unterschreitung.

Das Argument enthält die untere Sende-Füllstands-Grenze.

Es gibt keine Rückgabewort.

Wenn dieses Filter benutzt wird, muß auch Filter 5 benutzt werden.

Aktions-Filter 7

Funktionsaufruf, wenn alle Zeichen physikalisch gesendet sind.

Das Argument hat keine Bedeutung.

Es gibt keine Rückgabewort.

Aktions-Filter 8

Funktionsaufruf, wenn sich eine Eingangs-Steuerleitung ändert. Abhängig von der verwendeten Hardware handelt es sich z.B. um die Steuerleitungs-Eingänge CTS, DCD, RI, DSR etc.

Das Argument enthält eine Maske für Steuerleitungs-Eingänge, die der Bedeutung des Parameters "Status der Steuerleitungen" entspricht. Ein gesetztes Bit (=1) führt zum Aufruf der Funktion, ein nicht gesetztes unterbindet den Aufruf.

Es gibt keine Rückgabewort.

Aktions-Filter 9

Funktionsaufruf bei nicht behebbarem Laufzeitfehler.

Die Fehlerursache muss durch Lesen des Parameter 1 des Programms M6P0520.LIB ermittelt werden.

Das Argument hat keine Bedeutung.

Es gibt kein Rückgabewort.

13.6.2. Erweiterte Parameterstruktur von Programm 520

Nr.	Typ	Init	Zugr	Bedeutung des Parameters
56	Long (4)	0	R/W	Aktions-Filter-Maske (Bitmap), Aufruf bei Bit 0 = 1: jedem empfangenen Zeichen Bit 1 = 1: empfangenem Zeichen im Vergleichs-string enthalten Bit 2 = 1: Fehler beim Empfang (Overflow, Parity, etc.) Bit 3 = 1: Empfangs-Füllstands-Überschreitung Bit 4 = 1: Empfangs-Füllstands-Unterschreitung Bit 5 = 1: Sende-Füllstands-Überschreitung Bit 6 = 1: Sende-Füllstands-Unterschreitung Bit 7 = 1: leerem Empfangspuffer Bit 8 = 1: Änderung der Steuerleitungs-Eingänge (z.B. CTS, DCD, RI, DSR). Die Verfügbarkeit der Steuerleitungen ist von der eingesetzten Hardware abhängig (siehe Hardwarebeschreibung). Bit 9 = 1: nicht behebbarem Laufzeitfehler Bit 10 bis 31 sind reserviert

Nr.	Typ	Init	Zugr	Bedeutung des Parameters
60				**Aktions-Filter 0**: bei jedem empfangenen Zeichen
	Word (2)	0	R/W	Aktions-Task
	Word (2)	0	R/W	Aktions-Funktion
	Long (4)	0	-	Argument: ohne Bedeutung
	Long (4)	0	R	Funktionsadresse (Segment:Offset) Rückgabewert (Byte): 0 = Zeichen nach Aufruf des Filters noch im Empfangspuffer speichern, 1 = nicht speichern
72				**Aktions-Filter 1**: empfangenes Zeichen ist im Vergleichsstring (siehe Parameter 320)
	Word (2)	0	R/W	Aktions-Task
	Word (2)	0	R/W	Aktions-Funktion
	Long (4)	0	R/W	Argument: Zeiger auf Vergleichsstring (Segment:Offset), Aufbau s.u.
	Long (4)	0	R	Funktionsadresse (Segment:Offset) Rückgabewert (Byte): 0 = Zeichen nach Aufruf des Filters noch im Empfangspuffer speichern, 1 = nicht speichern
84				**Aktions-Filter 2**: Empfangsfehler
	Word (2)	0	R/W	Aktions-Task
	Word (2)	0	R/W	Aktions-Funktion
	Long (4)	0	R/W	Argument: Fehlermaske (Bit 0 bis 15, Bitmap), Bedeutung wie Parameter "Empfangsstatus"
	Long (4)	0	R	Funktionsadresse (Segment:Offset)
96				**Aktions-Filter 3**: Empfangs-Füllstands-Überschreitung
	Word (2)	0	R/W	Aktions-Task
	Word (2)	0	R/W	Aktions-Funktion
	Long (4)	0	R/W	Argument: obere Empfangs-Füllstands-Grenze
	Long (4)	0	R	Funktionsadresse (Segment:Offset)
108				**Aktions-Filter 4**: Empfangs-Füllstands-Unterschreitung
	Word (2)	0	R/W	Aktions-Task
	Word (2)	0	R/W	Aktions-Funktion
	Long (4)	0	R/W	Argument: untere Empfangs-Füllstands-Grenze
	Long (4)	0	R	Funktionsadresse (Segment:Offset)

Nr.	Typ	Init	Zugr	Bedeutung des Parameters
120				**Aktions-Filter 5**: Sende-Füllstands-Überschreitung
	Word (2)	0	R/W	Aktions-Task
	Word (2)	0	R/W	Aktions-Funktion
	Long (4)	0	R/W	Argument: obere Sende-Füllstands-Grenze
	Long (4)	0	R	Funktionsadresse (Segment:Offset)
132				**Aktions-Filter 6**: Sende-Füllstands-Unterschreitung
	Word (2)	0	R/W	Aktions-Task
	Word (2)	0	R/W	Aktions-Funktion
	Long (4)	0	R/W	Argument: untere Sende-Füllstands-Grenze
	Long (4)	0	R	Funktionsadresse (Segment:Offset)
144				**Aktions-Filter 7**: Alle Zeichen gesendet
	Word (2)	0	R/W	Aktions-Task
	Word (2)	0	R/W	Aktions-Funktion
	Long (4)	0	R/W	Argument: ohne Bedeutung
	Long (4)	0	R	Funktionsadresse (Segment:Offset)
156				**Aktions-Filter 8**: Änderung einer Steuerleitung
	Word (2)	0	R/W	Aktions-Task
	Word (2)	0	R/W	Aktions-Funktion
	Long (4)	0	R/W	Argument: Maske für Steuerleitungen, Bedeutung wie bei Status der Steuerleitungen
	Long (4)	0	R	Funktionsadresse (Segment:Offset)
168				**Aktions-Filter 9**: nicht behebbarer Laufzeitfehler
	Word (2)	0	R/W	Aktions-Task
	Word (2)	0	R/W	Aktions-Funktion
	Long (4)	0	R/W	Argument: ohne Bedeutung
	Long (4)	0	R	Funktionsadresse (Segment:Offset)
180	6 * 12 = 72 Byte	0	R	**Aktions-Filter 10 bis 15**: reserviert Task, Funktion, Argument, Funktionsadresse (Aufbau wie Filter 0)

Nr.	Typ	Init	Zugr	Bedeutung des Parameters
318	Word (2)	0	R/W	**Protokolltyp** 0 = kein Protokoll 1 = XON/XOFF : Aktions-Filter 1, 3 und 4 werden verwendet, die Funktionen 38, 36 und 37 werden aufgerufen 2 = RTS/CTS: Aktions-Filter 3, 4 und 8 werden verwendet, die Funktionen 40, 41 und 42 werden aufgerufen
320	16 Byte	0	R/W	**Vergleichsstring für Filteraktion 1** Aufbau: Das erste Zeichen enthält die Anzahl der zu vergleichenden Zeichen im String, danach folgen die Zeichen (maximal 15), bei Protokolltyp = 1 werden automatisch die XON/XOFF Zeichen (11H, 13H) eingetragen.

A. Technische Daten der Multi-COM Karte

CPU:	5x86-P75 (133 MHz), 486DX4/120, 486DX2/66, 486DX/33, 486SX/25 oder /33
RAM:	Statisch: 512 KByte oder 2 MByte, batteriepufferbar Dynamisch: 2 MByte, 8 MByte oder 32 MByte
ROM:	EPROM oder Flash-EPROM, 64 KB bis 512 KB ausbaubar
EEPROM:	128 16-Bit Wörter, seriell
Timer:	3 (Baustein 8254), 16 Bit Breite, programmierbare Eingangsfrequenz: 1 MHz, 2,5 MHz oder 10 MHz, interruptfähig 7 weitere Timer in SCCs (je SCC 2 Timer) und Echtzeituhr
Interrupts:	15 Eingänge, davon 3 an den SCCs, einer an Stecker St3, zwei an PC-Schnittstelle, 3 an Timer, einer an Ri von serielle Schnittstelle B, einer an Uhr, einer an Temperatursensor, einer an Lüfterüberwachung
Spannungs-überwachung:	Zwei Ansprechschwellen (ca. 4,8 und 4,65 Volt), NMI-Auslösung, Umschaltung auf Batterie-Pufferung von RAM und Uhr
PC-Schnittstelle:	16-Bit parallel, bidirektional, interruptfähig (lokal und PC-Seite), 1 MByte/s max. Datentransferrate unidirektional
Serielle Schnitt-stellen:	6 ser. Schnittstellen mit allen Modem-Steuersignalen, 1 x RS-232, 5 per S-Link konfigurierbar, Baustein SCC 8530 oder ESCC 85230, programmierbare Quarzoszillatoren
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:	Echtzeitfähig, max. 1024 Tasks, Interrupt-, Timer-initiierte und Nicht-Interrupt-Tasks. Im EPROM der Karte enthalten, wird ins RAM kopiert

Stromaufnahme:	+5V:	1,5 A (486DX2/66MHz, 2 MB) 1,8 A (586-133MHz, 2 MB)
	+12V:	< 1 mA (ohne S-Links, ser. Schnittstelle B nicht benutzt)
	-12V:	< 1 mA (ohne S-Links, ser. Schnittstelle B nicht benutzt)
	-5V:	nicht benutzt
	Die Stromaufnahme enthält nicht den Strom für den Lüfter der CPU.	
Abmessungen:	Karte mit kurzem Slotstecker: 160 mm x 106,68 mm (ohne Slotblech und D-Sub-Stecker)	
Temperaturverträglichkeit:	0 bis 55°C	
Luftfeuchtigkeitsverträglichkeit:	5 bis 95% (nicht kondensierend)	

B. Übersicht S-Links

Die hier aufgeführten S-Links sind ohne Einschränkungen auf der Multi-COM Karte einsetzbar, sofern nichts anderes vermerkt ist. Einige S-Links müssen DIP-Switch konfiguriert werden. Es ist in jedem Fall die Beschreibung der S-Links zu beachten.

S-Link-Adapter	**Typ**	**Physikalische Schnittstelle**	**Kurzbeschreibung**
SL-232S	1	RS-232 bis 220 KBaud	Modem-Steuerleitungen: TMT, RCV, RTS, CTS, DTR, DSR, RI, DCD RI als Clock-Eingang
SL-232A/o	4	RS-232 bis 220 KBaud	Modem-Steuerleitungen: TMT, RCV, RTS, CTS, DTR, DSR, RI, DCD Zusätzliche Funktionen: RS-232-Leitung EXT als Clock-Ausgang, RI als Clock-Eingang
SL-232A/i	3	RS-232 bis 220 KBaud	Modem-Steuerleitungen: TMT, RCV, RTS, CTS, DTR, DSR, RI, DCD Zusätzliche Funktionen: RS-232-Leitung EXT als Clock-Eingang 1, RI als Clock-Eingang 2
SL-232i	16	RS-232 isol. bis 220 KBaud	Modem-Steuerleitungen, galvanisch getrennt: TMT, RCV, RTS, CTS Zusätzliche Funktionen: CTS als Clock-Eingang, RTS als Clock-Ausgang
SL-422S	8	RS-422 bis 10 MBaud	Modem-Steuerleitungen: TMT, RCV, RTS, CTS Zusätzliche Funktionen: CTS als Clock-Eingang, RTS als Clock-Ausgang

S-Link-Adapter	Typ	Physikalische Schnittstelle	Kurzbeschreibung
SL-422i	24	RS-422 isol. bis 10 MBaud	Modem-Steuerleitungen, galvanisch getrennt: TMT, RCV, RTS, CTS Zusätzliche Funktionen: CTS als Clock-Eingang, RTS als Clock-Ausgang
SL-485S	9	RS-485 bis 12 MBaud	Umschaltung von Senden auf Empfangen per Software oder automatisch (z.B. für SDLC/HDLC)
SL-485i	32	RS-485 isol. bis 12 MBaud	Auch für PROFIBUS bis 12 MBaud geeignet, zusätzlicher TTL-Ausgang zeigt Senden/Empfangen an, z.B. für ext. Transceiver.
SL-20MA	40	20 mA isol. bis 120 KBaud	20 mA Current Loop, passiv oder aktiv konfigurierbar (wenn passiv, dann galvanisch getrennt)
SL-LWL/P	48	Lichtwellenleiter (Plastik)	Toshiba Plastik-Lichtwellenleiter APF (TODX297)
SL-LWL/G	49	Lichtwellenleiter (Glas)	Toshiba Glas-Lichtwellenleiter PCS (TODX296)
SL-CANi	64	CAN-Interface, isoliert	Full-CAN, bis 1 MBit/s Übertragungsrate, 11- und 29-Bit Identifier
SL-DPSi	72	RS-485	PROFIBUS-Slave bis 12 MBaud
SL-IBSi	80	RS-485	InterBus S
SL-TEST	88	diverse	Test S-Link zum Testen aller anderen S-Links außer Typ 48 und 49

C. Lokale I/O-Adressen

Die folgende Aufstellung soll als Übersicht dienen. Wenn Sie eigene Anwendungsprogramme in 486-Assembler oder in anderen Sprachen schreiben wollen, finden Sie hierzu Hinweise und Beispiele in den Kapiteln 7, 8, 10, 11, 12 und 13 dieses Handbuchs.

Die 486er CPU auf der Karte kann alle 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 Karte, da es sich dabei ebenfalls um Standardchips handelt.

Auf alle Devices kann mit 8 Bit, auf einige darf auch mit 16 Bit (PC-Schnittstelle, RAM) und 32 Bit (RAM und ROM) zugegriffen werden. Der Zugriff auf einige Devices, z. B. den Temperatursensor, die Uhr und die SCCs, ist komplexer und sollte nur über die System-Subroutinen erfolgen.

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:

8 = nur Bytezugriff erlaubt
16 = nur Wortzugriff erlaubt
W = nur Schreibzugriffe sind erlaubt
R = nur Lesezugriffe sind erlaubt
RW = Schreib- und Lesezugriffe sind erlaubt
x = gelesene bzw. geschriebene Daten sind ohne Bedeutung

Folgende Devices sind auf der Multi-COM Karte verfügbar:

PC-Schnittstelle inkl. Konfiguration
Interrupt-Controller 1 und 2 (8259 bzw. 82SC59)
Einstellung der aktiven Flanke für einige Interrupts
Timer/Counter (8254): Timer A, Timer B und Timer C
6 serielle Schnittstellen, Kanal A bis F (3 x SCC 8530 bzw. ESCC 85230, SCC1 = Kanal A und B, SCC2 = Kanal C und D, SCC3 = Kanal E und F)
Uhr (RTC-72421) mit Timer-D
Watchdog-Timer
Überwachung der +5-Volt-Versorgungsspannung
Leuchtdioden LEDext (extern) und LEDint (on-board)
Serielles EEPROM (auch auf jedem S-Link vorhanden)
Temperatursensor/Thermostat
Lüftersteuerung und -überwachung (Drehzahl)

PC-Schnittstelle

Adresse	Zugriff	Funktion	
28h	R8/R16	Status 1:	Bit 0 = RBF
			Bit 1 = DLM
			Bit 2 = RESET (1=Reset aktiv)
			Bit 3 = BOOT (0= Karte hat kein EPROM)
			Bit 4 = Reserviert
			Bit 5 = RESTART
			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[1]	
43h	W8x	Setze Device Locking Bit DLP = 1	
2ah	R8/R16	Status 2:	Bit 7 = RESTART
			Bit 0 = ACTIVE

[1] Zustand nach Reset

PC-Interrupt

Adresse	Zugriff	Funktion
52h	W8x	PC-Interrupt-Leitung vom PC-Bus trennen[1]
53h	W8x	PC-Interrupt-Leitung an PC-Bus anschließen
55h, 54h	W8x	Interrupt-Request zum PC durch 2 aufeinanderfolgende Zugriffe (zuerst 55h, dann 54h)
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))

Verwendung der on-board Interrupts

	IRQ-	Funktion
Master	0	von PC-Schnittstelle (Receive Buffer Full)
	1	reserviert
	2	von Slave Controller (Interrupt-Controller 2)
	3	reserviert
	4	von SCC3 (Kanal E und F)
	5	von SCC2 (Kanal C und D)
	6	von SCC1 (Kanal A und B)
	7	von Lüfterüberwachung, aktive/r Flanke/Pegel programmierbar
Slave	0	von Interrupt-Ausgang des SCC1 (Kanal A und B)
	1	von Ausgang Timer A
	2	von Ausgang Timer B
	3	von Ausgang Timer C
	4	von Taktausgang der Uhr (Timer D)
	5	= IRQ-G: Externer Interrupt-Eingang (an St3), 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 (Master)

(siehe auch Datenblatt 8259)

Adresse	Zugriff	Funktion
00h	R8	Lies IRR (Interrupt Request 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
4ch	W8x	IRQ-TEMP: bei High bzw. pos. Flanke[2]
4dh	W8x	bei Low bzw. neg. Flanke
98h	W8x	Lüfterüberwachung von IRQ-FAN trennen
99h	W8x	Lüfterüberwachung mit IRQ-FAN verbinden
4eh	W8x	IRQ-FAN: bei High bzw. pos. Flanke[2]
4fh	W8x	bei Low bzw. neg. Flanke

Interrupt-Controller (Slave)

(siehe auch Datenblatt 8259)

Adresse	Zugriff	Funktion
08h	R8	Lies IRR (Interrupt Request 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	IRQ-G (ext. Int): bei High bzw. pos. Flanke[2]
49h	W8x	bei Low bzw. neg. Flanke
4ah	W8x	IRQ-H (RiB): bei High bzw. pos. Flanke[2]
4bh	W8x	bei Low bzw. neg. Flanke

[2] Zustand nach Reset

Timer/Counter A, B und C

(siehe Datenblatt 8254)

Adresse	Zugriff	Funktion
91h	W8x	Timer-Eingangstakt auf 1 MHz stellen
90h	W8x	Timer-Eingangstakt auf 2,5 oder 10 MHz stellen[3] (siehe Adresse 40h bzw. 41h)
40h	W8x	Timer-Eingangstakt auf 2,5 MHz stellen
41h	W8x	Timer-Eingangstakt 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

Leuchtdioden LEDext (extern) und LEDint (on-board)

Adresse	Zugriff	Funktion
50h	W8x	LEDext ein[4]
51h	W8x	LEDext aus
5ch	W8x	LEDint ein[4]
5dh	W8x	LEDint aus

I²C-Schnittstelle für Temperatursensor (nur Rev. D)

Adresse	Zugriff	Funktion
a2h	W8	SCL-Leitung (Ausgang) entsprechend Bit 0 setzen
2ch	R8	SDA-Eingang: lies Data an D0
a1h	W8	SDA-Ausgang entsprechend Bit 0 setzen
9ch	W8x	SDA-Mode = Eingang
9dh	W8x	SDA-Mode = Ausgang
c9h	R8	Lies Status von Thermostat an D0 (= 1 wenn Temperatur überschwellig)

[3] Zustand nach Reset

[4] Zustand nach Reset, kann aber nach Starten des Betriebssystems abhängig vom EEPROM-Inhalt (WORT-3) geändert werden.

Lüfter

Adresse	Zugriff	Funktion
92h	W8x	Fan off (sofern Temperatur unterschwellig)
93h	W8x	Fan on

Bei allen mit einem Lüfter ausgestatteten Multi-COM Karten ist der Lüfter nach Reset eingeschaltet. Wenn die Karte mit einem DS1721-Temperatursensor ausgestattet ist, verfügt sie neben der Temperaturmessung auch über einen Thermostatschalter, der den Lüfter zwangsweise einschaltet, sobald eine eingestellte Schwellentemperatur überschritten wird (unabhängig von obiger Einstellung).

Uhr

(siehe Datenblatt RTC-72421)

Für den Zugriff sind besondere Prozeduren erforderlich. Deshalb sollten hierfür nur die im EPROM vorhandenen Betriebssystem-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 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

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 ggf. ein NMI. Ein NMI kann auch, sofern aktiv, von der on-board Spannungsüberwachung kommen.

Adresse	Zugriff	Funktion
94h	W8x	Watchdog-Timer disable[3]
95h	W8x	Watchdog-Timer enable
7fh	R8x	Watchdog-Timer wird nachgetriggert
44h	W8x	NMI abgeschaltet[5]
45h	W8x	NMI aktiv

EEPROM-Anwahl-Adressen für die Basiskarte

Adresse	Zugriff	Funktion	
5eh	W8x	EEPROM Basiskarte:	Abwahl[5]
5fh	W8x		Anwahl
6ah	W8x	Schreibzugriff aktivieren	
6bh	W8x	Schreibzugriff deaktivieren	
68h	W8x	Schreibdaten = 0 setzen	
69h	W8x	Schreibdaten = 1 setzen	
28h	R16	Lies EEPROM-Data (nur Bit 8 ist gültig)	

[5] Zustand nach Reset bzw. siehe EEPROM-WORT-15, Bit 0

EEPROM-Anwahl-Adressen für die S-Links

Die EEPROMs der S-Links werden über Modem-Steuerleitungen DTR (= Clock), RTS (= Data schreiben) und DSR (= Daten lesen) des zugehörigen Kanals angesteuert, wenn das EEPROM angewählt ist.

Adresse	Zugriff	Funktion	
a0h	W8x	EEPROM S-Link A:	Abwahl[6]
a1h	W8x		Anwahl
a4h	W8x	EEPROM S-Link C:	Abwahl[6]
a5h	W8x		Anwahl
a6h	W8x	EEPROM S-Link D:	Abwahl[6]
a7h	W8x		Anwahl
a8h	W8x	EEPROM S-Link E:	Abwahl[6]
a9h	W8x		Anwahl
aah	W8x	EEPROM S-Link-F:	Abwahl[6]
abh	W8x		Anwahl

FPGA-Versionen

Adresse	Zugriff	Funktion
bnh (b0h - b7h)	R8	Lies FGPA-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 FGPA-Version IC4, 8 Bit, je Zugriff liefert Bit 0 ein Bit der Versions-Nr., wobei n = Wertigkeit des Bit (aktuelle Version = 01 bzw. 02 bei DRAM Versionen)
dnh (d0- d7h)	R8	Lies FGPA-Version IC5, 8 Bit, je Zugriff liefert Bit 0 ein Bit der Versions-Nr., wobei n = Wertigkeit des Bit (aktuelle Version = 01)

[6] Zustand nach Reset

Serielle Schnittstellen Kanal A und B

Adresse	Zugriff	Funktion	
9ah	W8x	Kanal A und B in MODULAR-4/486 Mode setzen[3]	
9bh	W8x	Kanal A und B in M-COM-2 Mode setzen	
9ah	R8	Mode lesen[7]:	0 = MODULAR-4/486 Mode 1 = M-COM-2 Mode
400h oder 30h	RW8	SCC1: Kanal B, Control	
401h oder 31h	RW8	Kanal B, Data	
402h oder 32h	RW8	Kanal A, Control	
403h oder 33h	RW8	Kanal A, Data	
40ah	W8	Konfiguration Schnittstelle A:	Port A1[8] (=CTRL A)
40bh	W8		Port A2[2]
40ch	W8		Port A3[2]
40dh	W8	Konfiguration Schnittstelle B:	Port B1[2] (=CTRL B)
40eh	W8		Port B2[2]
40fh	W8		Port B3[2]
408h	R8	Status lesen:	DSR/A[1]
40ah	R8		DSR/B[1]
404h oder 34h	W8	Anwahl PCLK für SCC1:	Data = 0: 7,37 MHz[2, 3] Data = 1: 4,91 MHz
405h oder 35h	W8	Reserviert	

[7] Beim Lesen steht das Ergebnis in Bit 0, die andere Bits sind ungültig

[8] Beim Schreiben wird der Port entsprechend Bit 0 gesetzt, Bit 1 bis Bit 7 sind ohne Bedeutung

[3] Zustand nach Reset

Serielle Schnittstellen Kanal C und D

Adresse	Zugriff	Funktion	
500h	RW8	SCC2: Kanal D, Control	
501h	RW8	Kanal D, Data	
502h	RW8	Kanal C, Control	
503h	RW8	Kanal C, Data	
50ah	W8	Konfiguration Schnittstelle C:	Port C1[1] (=CTRL C)
50bh	W8		Port C2[1]
50ch	W8		Port C3[1]
50dh	W8	Konfiguration Schnittstelle D:	Port D1[1] (=CTRL D)
50eh	W8		Port D2[1]
50fh	W8		Port D3[1]
508h	R8	Status lesen: DSR/C[2]	
50ah	R8	DSR/D[2]	
504h	W8	Anwahl PCLK für SCC2:	Data = 0: 7,37 MHz[1, 3] Data = 1: 4,91 MHz
505h	W8	Reserviert	

[1] Beim Schreiben wird der Port entsprechend Bit 0 gesetzt, Bit 1 bis Bit 7 sind ohne Bedeutung

[2] Beim Lesen steht das Ergebnis in Bit 0, die andere Bits sind ungültig

[3] Zustand nach Reset

Serielle Schnittstellen Kanal E und F

Adresse	Zugriff	Funktion	
600h	RW8	SCC3: Kanal F, Control	
601h	RW8	Kanal F, Data	
602h	RW8	Kanal E, Control	
603h	RW8	Kanal E, Data	
60ah	W8	Konfiguration Schnittstelle E:	Port E1[1] (=CTRL E)
60bh	W8		Port E2[1]
60ch	W8		Port E3[1]
60dh	W8	Konfiguration Schnittstelle F:	Port F1[1] (=CTRL F)
60eh	W8		Port F2[1]
60fh	W8		Port F3[1]
608h	R8	Status lesen: DSR/E[2]	
60ah	R8	DSR/F[2]	
604h	W8	Anwahl PCLK für SCC3:	Data = 0: 7,37 MHz[1, 3] Data = 1: 4,91 MHz
605h	W8	Reserviert	

[1] Beim Schreiben wird der Port entsprechend Bit 0 gesetzt, Bit 1 bis Bit 7 sind ohne Bedeutung

[2] Beim Lesen steht das Ergebnis in Bit 0, die andere Bits sind ungültig

[3] Zustand nach Reset

D. Lokale Interrupts der Multi-COM 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	TEMP	Temperatursensor	Master-1
	122	7ah	Slave	von Slave-Controller	Master-2
	144	90h	SCC1	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	-	Reserviert	Master-3
	124	7ch	SCC3	SCC3 (Kanal E und F)	Master-4
	125	7dh	SCC2	SCC2 (Kanal C und D)	Master-5
	126	7eh	SCC1	SCC1 (Kanal a und B)	Master-6
Niedrig	127	7fh	FAN	Lüfterüberwachung	Master-7

E. Fehlermeldungen von PC-Bibliotheken

Beim Einsatz von Multi-COM, MODULAR-4/486 und Multi-LAB/2 Karten und den dazugehörigen Bibliotheken können die folgenden Fehlermeldungen auftreten.

Alle nicht erwähnten Fehlercodes sind reserviert.

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 **ml6_start** und **ml6_reset** wird ein Timeout (in Zehntelsekunden) eingestellt. Der Wert 10 (= 1 Sekunde) sollte in der Regel ausreichen. Mit **ml6_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
3	3h	Plausibilitätsprüfung gescheitert Das erste Antwort-Byte eines Makrobefehls war falsch
4	4h	DLP-Timeout zu Beginn eines Makrobefehls (DLP = 1)

Fehlercode		Bedeutung des Fehlercodes der Fehlerklasse 0
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 (z.B. Adresskonflikt)
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 **ml6_reset** oder **ml6_start**.
7	7h	Programmtyp und Installierungsflag (Bit 6-8) stimmen nicht überein (z. B. keine EXE-Datei).
16	10h	Unzulässiger Parameter (Datum/Zeit)
17	11h	Datenrückgabelänge überschreitet den Vorgabewert (**ml6_call_ func**)
18	12h	Wortnummer für EEPROM ungültig
25	19h	Wortnummer für EEPROM ungültig
32	20h	Aufruf einer Bibliotheksfunktion ohne vorhergehenden Aufruf von **ml6_bib_startup**
33	21h	Aufruf einer Multi-LAB/2 Funktion für eine Multi-COM oder MODULAR-4/486 Karte
34	22h	Aufruf einer Multi-COM oder MODULAR-4/486 Funktion für eine Multi-LAB/2 Karte
35	23h	Aufruf einer Funktion, die (noch) nicht unterstützt wird
36	24h	Reserviert
37	25h	Reserviert
38	26h	Parameter #1 ist falsch
39	27h	Parameter #2 ist falsch
40	28h	Parameter #3 ist falsch
48	30h	Maximale Makro-Kommandolänge überschritten
49	31h	Maximale Makro-Kommandorückgabelänge überschritten

Fehlercode		Bedeutung des Fehlercodes der Fehlerklasse 2
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 **ml6_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 **ml6_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

Neue Fehlermeldungen, die zum Zeitpunkt der Drucklegung noch nicht festlagen, können Sie der Datei 'readme.doc' im Verzeichnis Ihrer PC-Bibliotheken entnehmen. **!**

Notizen

F. Fehlermeldungen 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 (serielle Schnittstellen)	SCC-Nr. (obere 4 Bit) und Typ (untere 4 Bit=32)
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	-
244	f4h	Reserviert	-
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
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

Fehlergruppe (Low Byte)		**Bedeutung**	**Fehlertyp (High Byte)**
Dez	**Hex**		
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	Warnungen: reserviert	-
221	ddh	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
209	d1h	PC: Warnung von Befehl, ausgeführt	Makrobefehl
208	d0h	PC: Warnung von Befehl, ausgeführt	Typ der Warnung

Fehlergruppe (Low Byte)		**Bedeutung**	**Fehlertyp (High Byte)**
Dez	**Hex**		
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
0	00h	Meldung: Kein Fehler
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 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
27	1bh	Falscher SCC angewählt
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)
41-47	29h-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
53-95	35h-5fh	Reserviert
96-127	60h-7fh	Benutzerspezifische Fehler
128	80h	SCC nicht vorhanden
129	81h	SCC benötigt keine Initialisierung
130	82h	SCC wird nicht initialisiert wegen EEPROM-Info in Wort 1
131	83h	SCC 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)

Nummer		**Bedeutung**
Dez	**Hex**	
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, Multi-COM 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	8+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 Multi-COM 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 Multi-COM/ser. Schnittstellen entsprechend EEPROM
Multi-Tasking: Installieren, Aktivieren, etc.			
40h 0fh 12h 00h	22	0	Installiere Programm:
tL tH			Tasknr.
pL pH			Programm-Nr.
iL 00h			Interrupt-Nr.
fE fF fG fH			Flag
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)

Format	Byte Hin	Byte Rück	Erklärung
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 mL mH tL tH fL fH b1..bn-4	10+n	6+v	Funktion f der Task t aufrufen (Antwort: 44h err wL wH vL vH a1...av)
45h 04h tL tH fL fH	6	0	Prozedur f der Task t aufrufen
Multi-Tasking: Parameter lesen und schreiben			
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 f4h tL tH pL pH	6	1+n	Parameterblock lesen
5ch 05h tL tH pL pH b	7	0	Parameterbyte schreiben
5dh 06h tL tH pL pH wL wH	8	0	Parameterwort schreiben
5eh 08h tL tH pL pH wE wF wG wH	10	0	Parameterdoppelwort schreiben
4dh 0fh tL tH pL pH b1...bn	6+n	0	Parameterblock schreiben

Format	Byte Hin	Byte Rück	Erklärung
Multi-Tasking: Daten lesen und schreiben			
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 d4h mL mH s 00h pE pF pG pH	10	4+m	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 mL mH s 00h pE pF pG pH b1 ... bm	10+ m	2	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:

- ML6BIB: ml6_get_task_info (Kapitel 6)
- ML6RTBIB: ml6rt_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) (Entspricht dem Wert, der in der PDT eingetragen wurde. 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 der Bibliothek "ML6RTBIB" 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 65535 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 ML6RTBIB
0	8086/8088	**_86**
1	80186/V20	**_186**
2	80286	**_286**
3	80486 SX	**_486SX**
4	80486 DX	**_486DX**
5	80586 (Pentium)	**_586**

Coprozessor-Typ (Byte)

Diese Angabe beschreibt, ob und welchen Coprozessor das Programm erwartet.

(Value)	Coprozessor-Typ	Konstante in ML6RTBIB
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 Multi-COM/486SX (Emulation wird benutzt) als auch auf einer Multi-COM/486DX Karte (Coprozessor wird benutzt) lauffähig. Mit **_487DX** kompilierte Programme setzen voraus, daß eine Multi-COM/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 ML6RTBIB
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 ML6RTBIB
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 ML6RTBIB 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) durchnummeriert 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 (Word)

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 Programms.

Anzahl Prozeduren (Word)

Dieser Eintrag ist eine Kopie aus dem entsprechenden Eintrag in der PDT des Programms.

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 (6 = Multi-COM)
5 (05h)	B	R	Kartenrevision (1 = A, 2 = B, 3 = C, ...)
12 (0ch)	10S	R	Betriebssystem Name, Version und Revision, z.B.: "ML6-1A.01x" 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.: "24/12/97" (tt/mm/jj)
30 (1eh)	8S	R	Uhrzeit der Herstellung des Betriebssystems, z.B.: "23:42:59" (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, 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 und Nexgen 586)
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)
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
104 (68h)	W	R	EEPROM von Kanal A u. B (SCC1): Länge (Anzahl Byte)
106 (6ah)	W	R	Parameter-Nr., ab der die EEPROM-Kopie steht oder

Parameter	Typ	Zugr	Bedeutung
			Fehlercode, wenn Parameter 104 = 0
108 (6ch)	W	R	EEPROM von Kanal C u. D (SCC2): 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 Kanal E u. F (SCC3): 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	Reserviert
118 (76h)	W	R	Reserviert
148 (94h)	W	R	CPU-Taktfrequenz (in Vielfachen von 100 kHz)
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
214 (d6h)	B	RW	Watchdog: 0=Auto-Retrigger off, 1=on

Parameter	Typ	Zugr	Bedeutung
308 (134h)	W	R	Max. Anzahl TI-Tasks
311 (137h)	B	RW[1]	Timerkanal 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 = Timer C)
316 (13ch)	D	RW[1]	Timer-Tic für TI-Tasks in Mikrosekunden 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)
424 (1a8h)	B	RW	System-Routine 39 (Watchdog Retrigger) enable (=0) oder disable (=1)
425 (1a9h)	B	RW	Zugriff über System Routinen auf Uhr enabled (=0) oder disabled (=1)

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

L. EEPROM-Inhalte

Alle Versionen der Multi-COM Karten verfügen über ein EEPROM. Hier sind Konfigurations- und Initialisierungsdaten für die Multi-COM (Basiskarte) und die seriellen Schnittstellen A bis F (SCC1 bis 3) abgelegt. Nach dem 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.

In WORT-3 bis -7 können 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 der Multi-COM Karte vorgesehen (WORT-0 bis WORT-31). Für die seriellen Schnittstellen sind zusätzlich je SCC 16 Wörter vorhanden. Die Daten im EEPROM können auf zwei Arten gelesen und geändert werden:

1. Mit dem PC-Hilfsprogramm SNW6 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). **!**

EEPROM der Multi-COM (z.B. DX2/66 mit 2 MB stat. RAM)

Wort	Binär	Hex	Bedeutung (Kurzinfo)	Einst.[1]
0	0010 010 0000 0001	2401h	Kartentyp: Multi-COM, Rev. D	W
Initialisierungsdaten				
1	0000 0000 0000 0000	0000h	Init. Multi-COM Karte	U
2	0000 0000 0000 0000	0000h	Reserviert	-
3	0000 0000 0000 0000	0000h	Init. Hardware, z.B. LEDs	U
4	0000 0010 0000 0100	0204h	Init. Software: OsX, SRQ	U
5, 6	0000 0000 0000 0000	0000h	Reserviert	-
7	0000 0000 0000 0000	0000h	SRQ nach Init	U
8, 9	0000 0000 0000 0000	0000h	Reserviert	-
10	0000 0000 0000 0001	0001h	Gate-Array-Version IC3	A
11	0000 0000 0000 0001	0001h	Gate-Array-Version IC4	A
12	0000 0000 0000 0001	0001h	Gate-Array-Version IC5	A
Einstellungen				
13	0000 0000 0000 1000	0008h	PC-IRQ	U
14	0000 0000 0000 0000	0000h	Reserviert	-
15	0000 0000 0000 0000	0000h	Diverse	U
Bestückung				
16-18	0000 0000 0000 0000	0000h	Reserviert	-
19	0000 0000 0000 0011	0003h	Serielles EEPROM	W
20	0111 0011 0100 0001	7341h	CPU-Typ	W
21	0111 0011 0001 0010	7312h	CPU-Beschaltung	W
22	0000 0011 0011 0000	0330h	CPU-Quarz = 33,0 MHz	W
23	0000 0000 0000 0000	0000h	Reserviert	-
24	0000 0000 0000 0000	0000h	Quarzfrequenz	W
25	0000 0001 0000 0000	0100h	Quarzosz. für Timer = 10,0 MHz	W
26	0000 0000 0010 0001	0021h	Timer	W
27	0000 0010 0000 0010	0202h	ROM	W
28	0000 0001 0001 0000	0140h	SRAM (Bank 1)	W
29	0000 0000 0000 0000	0000h	DRAM (Bank 2)	W
30	0000 0001 0010 1101	012Dh	Bestückung, div.	W
31	0000 0000 0000 0000	0000h	Reserviert	-

[1] Einstellung: A = automatisch, W = werkseitige Einstellung, U = Einstellung durch den Anwender

WORT-0: Typ und Version der Karte

Dieses Wort 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	1	0	0	0	0	0	0	0	0	0	1	WORT-0: Kennung
								0	0	0	0	0	0	0	1	Typ: 1 = Basiskarte
				0	1	0	0									Revision: 1 = A, 2 = B, 3 = C, 4 = D
			0													Reserviert (= 0)
0	0	1														Kennung

WORT-1: Init nach Reset (Multi-COM und ser. Schnittstellen)

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 Multi-COM: 0 = nein, 1 = ja
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		Reserviert (= 0)

WORT-3: Initialisierung Hardware Multi-COM Karte

Neben den Anfangszuständen der Leuchtdioden enthält WORT-3 auch die Einstellungen für die Interrupt-Leitung (PC-IRQ) zum PC (siehe hierzu auch WORT-13). Dabei gilt folgende Zuordnung ('x' bedeutet: Zustand spielt keine Rolle):

IRQEN	IRQREN	PC-IRQ
0	x	nicht verbunden
1	0	1
1	1	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	0	WORT-3: Init Multi-COM (werks. Einst.)
																geändert am: von:
														0	0	LEDint (on-board): 00 = off, 01 = on
												0	0			LEDext: 00 = off, 01 = on
										0	0					Timer-Eingangstakt (Timer-A, -B und -C): 0 = 2,5 MHz; 1 = 10 MHz; 2 = 1 MHz; 3 = Reserviert
								0	0							Reserviert
							0									Zustand von IRQEN
						0										Zustand von IRQREN
0	0	0	0	0	0											Reserviert (= 0)

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 (= 0)

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

L

WORT-10 bis -12: Gate-Array-Versionen

Diese 3 Wörter dürfen nicht geändert werden!

Insgesamt ist die Multi-COM Karte mit drei 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 (automat. 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 (automat. 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-12: IC5 Version (automat. 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)

WORT-13: Hardware-Einstellungen (PC-Interrupt)

Hier kann vom Anwender der einzustellende PC Interrupt-Kanal angegeben 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	1	0	0	0	WORT-13: PC-IRQ (werks. Einst.)
																geändert am: von:
															0	1 = IRQ-4
														0		1 = IRQ-3
													0			1 = IRQ-5
												1				1 = IRQ-7
											0					1 = IRQ-9
										0						1 = IRQ-11
									0							1 = IRQ-10
								0								1 = IRQ-12
0	0	0	0	0	0	0	0									Reserviert (= 0)

WORT-15: Hardware-Einstellungen (Diverse)

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: Diverse (werks. Einst.)
																geändert am: von:
															0	Watchdog: 1= enable, 0 = disable
														0		Reserviert (=0)
													0			Spannungsüberw. A: 1 = enable, 0 = disable
												0				Reserviert (= 0)
											0					DTR/A (ser. Schnittstelle A)[1]: 0 = DTR/A für Modem 1 = DTR/A von TRxCA
										0						RTxC/A (ser. Schnittstelle A)[2]: 0 = RTxC/A von Quarz 1 = RTxC/A von Ri/A
								0	0							Reserviert (= 0)
							0									SRAM: 0 = SRAM an +5V 1 = SRAM an Ubat
						0										Reserviert (= 0)
					0											Batterieanschluß: 0 = nein, 1 = ja
0	0	0	0	0												Reserviert (= 0)

[1] Dieses Bit wird nur für den Betrieb der seriellen Schnittstellen A und B im MODULAR-4/486-Mode benötigt. Siehe auch WORT-23.

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	1	1	WORT-19: serielles EEPROM (werks. Einst.)
																geändert am: von:
								0	0	0	0	0	0	1	1	Größe serielles EEPROM (Anzahl Wörter): 0 = kein, 1 = 32, 2 = 64, 3 = 128, ...
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

WORT-24: Bestückung: Quarz für Kanal B (SCC1)

Hier wird die Quarzfrequenz eingetragen. Dieser Takt kann zur Erzeugung der SCC-Baudraten verwendet werden. Standardmäßig ist dieser Quarz nicht bestückt. Für 7,3728 MHz wird die Hexadezimalzahl 0737 eingetragen, für 4,9152 MHz entsprechend 0491. Bei Multi-COM-Karten mit ESCC-20 Bausteinen kann auch 1650 eingetragen werden. Dann wird für PCLK der halbe CPU-CLK (=16,5 MHz) eingestellt.

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-24: Quarzfrequenz (werks. Einst.)
																geändert am: von:
												0	0	0	0	2. Ziffer nach Komma
								0	0	0	0					1. Ziffer nach Komma
				0	0	0	0									2. Ziffer (danach Komma)
0	0	0	0													1. Ziffer

WORT-25: Quarzoszillator TCLK für Timer

Hier wird die Frequenz des Quarzoszillators für den Timer angegeben. Die Taktfrequenz wird in Form von 4 Ziffern eingetragen. Für 2,5 MHz wird z.B. die Hexadezimalzahl 0025 eingetragen, für 10 MHz 0100. Die Karte ist standardmäßig mit 10 MHz ausgerüstet, die Eingangsfrequenz der Timer A, B und C kann per Software auf 1 MHz, 2,5 MHz oder 10 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: Angaben zu 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	0	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	0	0	0	0	0	0	0									Reserviert (= 0)

WORT-27: Bestückung: ROM

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, etc.
				0	0	1	0									Typ ROM: 0 = unbekannt 1 = ROM 2 = EPROM 3 = EEPROM 4 = Flash-EPROM
0	0	0	0													Reserviert (= 0)

WORT-28: Bestückung: SRAM (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	1	0	0	0	0	0	0	WORT-28: SRAM (werks. Einst.)
																geändert am: von:
								0	1	0	0	0	0	0	0	Größe (in Vielf. von 32 KByte): 0 = kein, 1 = 32 KByte, 4 = 128 KByte, 16 = 512 KByte, 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)

L

WORT-29: Bestückung: DRAM (Bank 2)

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-28: DRAM (werks. Einst.)
																geändert am: von:
								0	0	0	0	0	0	0	0	Größe (in Vielf. von 1 MByte):
					0	0	0									Typ:
				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)

Größe (in Vielf. von 1 MByte):

- 0 = kein
- 1 = 1 MByte
- 2 = 2 MByte
- 4 = 4 MByte
- 8 = 8 MByte
- 16 = 16 MByte
- 32 = 32 MByte

Typ:

0 = unbekannt bzw. kein DRAM
1 = statisch 2 = pseudostatisch
3 = dynamisch

WORT-30: Bestückung, diverse

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	1	0	1	1	0	1	WORT-30: Bestückung, div. (werks. Einst.)
																geändert am: von:
														0	1	Uhrentyp: 0 = unbekannt 1 = 72421 2 = 62421 3 = 63421
													1			LEDint on-board: 1 = ja
												1				LEDint an St3, 1 = ja
											0					LEDext on-board: 1 = ja
										1						LEDext an St3: 1 = ja
									0							PC-RBF an NMI: 1 = ja
								0								Timer-C an NMI: 1 = ja
							1									±12 V für Kanal B (RS-232) erforderlich: 1 = ja
					0	0										Bestückung: Temperatur-Sensor (Typ): 00: keiner 01: TMP04 10: DS1721
0	0	0	0	0												Reserviert (= 0)

L

EEPROM der Multi-COM (serielle Schnittstellen)

Werkseitig ist für jeden der drei SCCs (Kanal A/B, C/D und E/F) bereits eine Konfiguration im EEPROM voreingestellt:

Serielle Schnittstellen A und B (SCC1)

WORT	Binär		Hex.	Bedeutung (Kurzinfo)	Einst.[1]
0	0010 0111	0010 0000	2720h	Typ	W
1	0000 0000	0000 0001	0001h	Initialisierung	U
2	0000 0000	0001 0000	0010h	Bestückung	W
3	0000 0111	0011 0111	0737h	Frequenz	U
4	0000 0011	0000 0011	0303h	Bestückung (Kanal A)	W
5	0000 0101	0000 0100	0504h	Bestückung (Kanal B)	W
6	0000 0000	0000 0000	0000h	Reserviert	-
7	0000 0000	0000 0000	0000h	Reserviert	-
8	0000 0000	0000 0000	0000h	S-Link-Typ (Kanal A)	A
9	0000 0000	0000 0000	0000h	S-Link-Revision (Kanal A)	A
10	0000 0000	0000 0000	0000h	S-Link-Hersteller (Kanal A)	A
11	0000 0000	0000 0000	0000h	Reserviert	-
12	0000 0000	0000 0000	0000h	S-Link-Typ (Kanal B)	A
13	0000 0000	0000 0000	0000h	S-Link-Revision (Kanal B)	A
14	0000 0000	0000 0000	0000h	S-Link-Hersteller (Kanal B)	A
15	0000 0000	0000 0000	0000h	Reserviert	-

[1] Einstellung: A = automatisch, W = werkseitige Einstellung, U = Einstellung durch den Anwender

WORT-0: Typ und Version der Schnittstellen A und B

Dieses Wort 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	1	1	1	0	0	1	0	0	0	0	0	WORT-0: Kennung
								0	0	1	0	0	0	0	0	Typ: 32 = M-COM-2 kompatibel 1 = MODULAR-4/486 kompatibel
				0	1	1	1									Revision: 1 = A, 2 = B, 3 = C, ..., 7 = G bei M-COM-2: Rev. G bei MODULAR-4/486: Rev. C
			0													Reserviert
0	0	1														Kennung

WORT-1: Initialisierung

In diesem Wort kann eingestellt werden, ob die Schnittstellen A und B nach dem Einschalten und bei einem Hardware-Reset entsprechend den Eintragungen im EEPROM initialisiert werden (Bit-0 = 1) oder nicht (Bit-0 = 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-1: Initialisierung (werks. Einst.)
																geändert am: von:
															1	Init nach Hardware-Reset: 0 = nein, 1 = ja
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		Reserviert

WORT-2: Bestückung

Dieses Wort darf nicht geändert 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	1	0	0	0	0	WORT-2: Bestückung (werks. Einst.)
												0	0	0	0	Typ des SCC: 0 = SCC (85C30) 2 = ESCC (85230)
								0	0	0	1					Max. Takt (SCC): 0 = 6 MHz 1 = 8 MHz 2 = 10 MHz 3 = 16 MHz 4 = 20 MHz

L

WORT-3: Frequenz an SCC1 (wird an PCLK von SCC1 gelegt)

Die Angabe erfolgt mit 4 Ziffern in MHz mit 2 Vorkomma- und 2 Nachkommastellen. 4,9152 MHz würde also aufgerundet auf 4,92 und mit 4 Hexadezimalziffern 0491 (= 0491h) eingegeben. Der Eintrag 0737h bedeutet 7,3728 MHz. Bei Multi-COM-Karten mit ESCC-20 Bausteinen kann auch 1650 eingetragen werden. Dann wird für PCLK der halbe CPU-CLK (=16,5 MHz) eingestellt. Zusätzlich kann der Takt auch über TRxCB eingespeist werden. Dazu muss fffeh eingetragen 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-3: Frequenz (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
0	0	0	0													1. Ziffer

WORT-4: Bestückung Kanal A

Dieses Wort darf nicht geändert werden!

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	0	0	0	0	1	1	WORT-4: Hardware Kanal A (werks. Einst.)
																geändert am: von:
															1	1 = S-Link möglich 0 = nicht vorgesehen
														1		1 = 2 x 24-poliger Sockel 0 = nicht vorbereitet
													0			1 = S-Link eingelötet 0 = nicht eingelötet
								0	0	0	0	0				Reserviert
							1									1 = Interface möglich 0 = nicht vorgesehen
						1										1 = Sockel 0 = nicht vorbereitet
					0											1 = fest eingelötet 0 = nicht eingelötet
0	0	0	0	0												Reserviert

WORT-5: Bestückung Kanal B

Dieses Wort darf nicht geändert werden!

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	WORT-5: Hardware Kanal B (werks. Einst.)
															0	1 = S-Link möglich 0 = nicht vorgesehen
														0		1 = 2 x 24-poliger Sockel 0 = nicht vorbereitet
													1			1 = S-Link eingelötet 0 = nicht eingelötet
								0	0	0	0	0				Reserviert
							1									1 = Interface möglich 0 = nicht vorgesehen
						0										1 = Sockel 0 = nicht vorbereitet
					1											1 = fest eingelötet 0 = nicht eingelötet
0	0	0	0	0												Reserviert

L

WORT-8: S-Link-Typ Kanal A

In WORT-8 wird der Typ des S-Links für Kanal A automatisch eingetragen.

Typ		Name	Interface	isol.	Anmerkung
0	(0000h)	-	keines	-	-
1	(0001h)	SL-232S	RS-232	nein	-
2	(0002h)	SL-232A	-	-	Reserviert
3	(0003h)	SL-232A/i	RS-232	nein	-
4	(0004h)	SL-232A/o	RS-232	nein	-
8	(0008h)	SL-422S	RS-422	nein	-
9	(0009h)	SL-485S	RS-485	nein	-
10	(000Ah)	SL-485A	-	-	Reserviert
16	(0010h)	SL-232i	RS-232	ja	-
24	(0018h)	SL-422i	RS-422	ja	-
32	(0020h)	SL-485i	RS-485	ja	Für PROFIBUS-geeignet
40	(0028h)	SL-20MA	20 mA	ja	-
48	(0030h)	SL-LWL/P	LWL (Plastik)	ja	Plastikfaser
49	(0031h)	SL-LWL/G	LWL (Glas)	ja	Glasfaser
56	(0038h)	SL-SSIi	RS-422	ja	2 Kanäle Sync. Ser. Interface
64	(0040h)	SL-CANi	CAN-Bus	ja	CAN-Bus
72	(0048h)	SL-DPSi	RS-485	ja	PROFIBUS-Slave
80	(0050h)	SL-IBSi	RS-485	ja	Interbus-S

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-8: S-Link-Typ Kanal A
(automatische Einstellung)

WORT-9: Revision des S-Link Kanal A

Im Low-Byte von WORT-9 wird die Revision des aufgesteckten S-Links für Kanal A automatisch eingetragen. Kann der Typ, die Revision oder der Hersteller des S-Links nicht ermittelt werden, wird im High-Byte der Fehlercode eingetragen (siehe Anhang F).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	WORT-9: S-Link-Revision Kanal A (automatische Einstellung)
								0	0	0	0	0	0	0	0	Revision: 1 = A, 2 = B, 3 = C, ..., 7 = G
0	0	0	0	0	0	0	0									Fehlerinformation, siehe Anhang F

WORT-10: S-Link-Hersteller Kanal A

In WORT-10 wird der Herstellercode des verwendeten S-Links für Kanal A automatisch eingetragen.

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-10: S-Link-Hersteller Kanal A (automatische Einstellung) 0 = unbekannt, 1 = SORCUS, 2 = Complex International (Ci)

WORT-12: S-Link-Typ Kanal B

In WORT-12 steht Schnittstellen-Typ für das physikalische Interface für Kanal B (fest eingelötet).

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-12: Schnittstellen-Typ Kanal B (automatische Einstellung) 0 = unbekannt 1 = entspricht SL-232S in Mode 1

WORT-13: Revision des S-Links Kanal B

Im Low-Byte von WORT-13 wird die Revision des aufgesteckten S-Links für Kanal B automatisch eingetragen. Kann der Typ, die Revision oder der Hersteller des S-Links nicht ermittelt werden, wird im High-Byte der Fehlercode eingetragen (siehe Anhang F).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	WORT-13: S-Link-Revision Kanal B (automatische Einstellung)
								0	0	0	0	0	0	0	0	Revision: 1 = A, 2 = B, 3 = C, ..., 7 = G
0	0	0	0	0	0	0	0									Fehlerinformation, siehe Anhang F

WORT-14: S-Link-Hersteller Kanal B

In WORT-14 wird der Herstellercode des verwendeten S-Links für Kanal B automatisch eingetragen.

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-14: S-Link-Hersteller Kanal B (automatische Einstellung) 0 = unbekannt, 1 = SORCUS, 2 = Complex International (Ci)

Serielle Schnittstellen C und D (SCC2)

WORT	Binär		Hex.	Bedeutung (Kurzinfo)	Einst.[1]
0	0010 0111	0010 0000	2720h	Typ	W
1	0000 0000	0000 0001	0001h	Initialisierung	U
2	0000 0000	0001 0000	0010h	Bestückung	W
3	0000 0111	0011 0111	0737h	Frequenz	U
4	0000 0011	0000 0011	0303h	Bestückung (Kanal C)	W
5	0000 0011	0000 0011	0303h	Bestückung (Kanal D)	W
6	0000 0000	0000 0000	0000h	Reserviert	-
7	0000 0000	0000 0000	0000h	Reserviert	-
8	0000 0000	0000 0000	0000h	S-Link-Typ (Kanal C)	A
9	0000 0000	0000 0000	0000h	S-Link-Revision (Kanal C)	A
10	0000 0000	0000 0000	0000h	S-Link-Hersteller (Kanal C)	A
11	0000 0000	0000 0000	0000h	Reserviert	-
12	0000 0000	0000 0000	0000h	S-Link-Typ (Kanal D)	A
13	0000 0000	0000 0000	0000h	S-Link-Revision (Kanal D)	A
14	0000 0000	0000 0000	0000h	S-Link-Hersteller (Kanal D)	A
15	0000 0000	0000 0000	0000h	Reserviert	-

[1] Einstellung: A = automatisch, W = werkseitige Einstellung, U = Einstellung durch den Anwender

WORT-0: Typ und Version der Schnittstellen C und D

Dieses Wort 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	1	1	0	0	0	1	0	0	0	0	0	WORT-0: Kennung
								0	0	1	0	0	0	0	0	Typ: 32 = M-COM-2 kompatibel
				0	1	1	0									Revision: 1 = A, 2 = B, 3 = C, ..., 7 = G
			0													Reserviert
0	0	1														Kennung

WORT-1: Initialisierung

In diesem Wort kann eingestellt werden, ob die Schnittstellen C und D nach dem Einschalten und bei einem Hardware-Reset entsprechend den Eintragungen im EEPROM initialisiert werden (Bit-0 = 1) oder nicht (Bit-0 = 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-1: Initialisierung (werks. Einst.)
																geändert am: von:
															1	Init nach Hardware-Reset: 0 = nein, 1 = ja
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		Reserviert

WORT-2: Bestückung

Dieses Wort 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	0	0	0	0	1	0	0	0	0	WORT-2: Bestückung (werks. Einst.)
												0	0	0	0	Typ SCC: 0 = SCC (85C30) 2 = ESCC (85230)
								0	0	0	1					Max. Takt (SCC): 0 = 6 MHz 1 = 8 MHz 2 = 10 MHz 3 = 16 MHz 4 = 20 MHz

WORT-3: Frequenz für SCC2 (wird an PCLK von SCC2 gelegt)

Die Angabe erfolgt mit 4 Ziffern in MHz mit 2 Vorkomma- und 2 Nachkommastellen. 4,9152 MHz würde also aufgerundet auf 4,92 und mit 4 Hexadezimalziffern 0491 (= 0491h) eingegeben. Der Eintrag 0737h bedeutet 7,3728 MHz. Bei Multi-COM-Karten mit ESCC-20 Bausteinen kann auch 1650 eingetragen werden. Dann wird für PCLK der halbe CPU-CLK (=16,5 MHz) eingestellt.

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-3: Frequenz (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
0	0	0	0													1. Ziffer

WORT-4: Bestückung Kanal C

Dieses Wort darf nicht geändert werden!

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	0	0	0	0	1	1	WORT-4: Hardware Kanal C (werks. Einst.)
																geändert am: von:
															1	1 = S-Link möglich 0 = nicht vorgesehen
														1		1 = 2 x 24-poliger Sockel 0 = nicht vorbereitet
													0			1 = S-Link eingelötet 0 = nicht eingelötet
								0	0	0	0	0				Reserviert
							1									1 = Interface möglich 0 = nicht vorgesehen
						1										1 = Sockel 0 = nicht vorbereitet
					0											1 = fest eingelötet 0 = nicht eingelötet
0	0	0	0	0												Reserviert

L

WORT-5: Bestückung Kanal D

Dieses Wort darf nicht geändert werden!

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	0	0	0	0	1	1	WORT-5: Hardware Kanal D (werks. Einst.)
															1	1 = S-Link möglich 0 = nicht vorgesehen
														1		1 = 2 x 24-poliger Sockel 0 = nicht vorbereitet
													0			1 = S-Link eingelötet 0 = nicht eingelötet
								0	0	0	0	0				Reserviert
							1									1 = Interface möglich 0 = nicht vorgesehen
						1										1 = Sockel 0 = nicht vorbereitet
					0											1 = fest eingelötet 0 = nicht eingelötet
0	0	0	0	0												Reserviert

WORT-8: S-Link-Typ Kanal C

In WORT-8 wird der Typ des S-Links für Kanal C automatisch eingetragen.

Typ		Name	Interface	isol.	Anmerkung
0	(0000h)	-	keines	-	-
1	(0001h)	SL-232S	RS-232	nein	-
2	(0002h)	SL-232A	-	-	Reserviert
3	(0003h)	SL-232A/i	RS-232	nein	-
4	(0004h)	SL-232A/o	RS-232	nein	-
8	(0008h)	SL-422S	RS-422	nein	-
9	(0009h)	SL-485S	RS-485	nein	-
10	(000Ah)	SL-485A	-	-	Reserviert
16	(0010h)	SL-232i	RS-232	ja	-
24	(0018h)	SL-422i	RS-422	ja	-
32	(0020h)	SL-485i	RS-485	ja	PROFIBUS-geeignet
40	(0028h)	SL-20MA	20 mA	ja	-
48	(0030h)	SL-LWL/P	LWL (Plastik)	ja	Plastikfaser
49	(0031h)	SL-LWL/G	LWL (Glas)	ja	Glasfaser
56	(0038h)	SL-SSIi	RS-422	ja	2 Kanäle Sync. Ser. Interface
64	(0040h)	SL-CANi	CAN-Bus	ja	CAN-Bus
72	(0048h)	SL-DPSi	RS-485	ja	PROFIBUS-Slave
80	(0050h)	SL-IBSi	RS-485	ja	Interbus-S

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-8: S-Link-Typ Kanal C
(automatische Einstellung)

WORT-9: Revision des S-Link Kanal C

Im Low-Byte von WORT-9 wird die Revision des aufgesteckten S-Links für Kanal C automatisch eingetragen. Kann der Typ, die Revision oder der Hersteller des S-Links nicht ermittelt werden, wird im High-Byte der Fehlercode eingetragen (siehe Anhang F).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	WORT-9: S-Link-Revision Kanal C (automatische Einstellung)
								0	0	0	0	0	0	0	0	Revision: 1 = A, 2 = B, 3 = C, ..., 7 = G
0	0	0	0	0	0	0	0									Fehlerinformation, siehe Anhang F

WORT-10: S-Link-Hersteller Kanal C

In WORT-10 wird der Herstellercode des verwendeten S-Links für Kanal C automatisch eingetragen.

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-10: S-Link-Hersteller Kanal C (automatische Einstellung) 0 = unbekannt, 1 = SORCUS, 2 = Complex International (Ci)

WORT-12: S-Link-Typ Kanal D

In WORT-12 wird der Typ des S-Links für Kanal D automatisch eingetragen.

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-12: S-Link-Typ Kanal D (auto. Einstellung, siehe WORT-8)

WORT-13: Revision des S-Link Kanal D

Im Low-Byte von WORT-13 wird die Revision des aufgesteckten S-Links für Kanal D automatisch eingetragen. Kann der Typ, die Revision oder der Hersteller des S-Links nicht ermittelt werden, wird im High-Byte der Fehlercode eingetragen (siehe Anhang F).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	WORT-13: S-Link-Revision Kanal D (automatische Einstellung)
								0	0	0	0	0	0	0	0	Revision: 1 = A, 2 = B, 3 = C, ..., 7 = G
0	0	0	0	0	0	0	0									Fehlerinformation, siehe Anhang F

WORT-14: S-Link-Hersteller Kanal D

In WORT-14 wird der Herstellercode des verwendeten S-Links für Kanal D automatisch eingetragen.

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-14: S-Link-Hersteller Kanal D (automatische Einstellung) 0 = unbekannt, 1 = SORCUS, 2 = Complex International (Ci)

Serielle Schnittstellen E und F (SCC3)

WORT	Binär		Hex.	Bedeutung (Kurzinfo)	Einst.[1]
0	0010 0111	0010 0000	2720h	Typ	W
1	0000 0000	0000 0001	0001h	Initialisierung	U
2	0000 0000	0001 0000	0010h	Bestückung	W
3	0000 0111	0011 0111	0737h	Frequenz	U
4	0000 0011	0000 0011	0303h	Bestückung (Kanal E)	W
5	0000 0011	0000 0011	0303h	Bestückung (Kanal F)	W
6	0000 0000	0000 0000	0000h	Reserviert	-
7	0000 0000	0000 0000	0000h	Reserviert	-
8	0000 0000	0000 0000	0000h	S-Link-Typ (Kanal E)	A
9	0000 0000	0000 0000	0000h	S-Link-Revision (Kanal E)	A
10	0000 0000	0000 0000	0000h	S-Link-Hersteller (Kanal E)	A
11	0000 0000	0000 0000	0000h	Reserviert	-
12	0000 0000	0000 0000	0000h	S-Link-Typ (Kanal F)	A
13	0000 0000	0000 0000	0000h	S-Link-Revision (Kanal F)	A
14	0000 0000	0000 0000	0000h	S-Link-Hersteller (Kanal F)	A
15	0000 0000	0000 0000	0000h	Reserviert	-

[1] Einstellung: A = automatisch, W = werkseitige Einstellung, U = Einstellung durch den Anwender

WORT-0: Typ und Version der Schnittstellen E und F

Dieses Wort 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	1	1	0	0	0	1	0	0	0	0	0	WORT-0: Kennung
								0	0	1	0	0	0	0	0	Typ: 32 = M-COM-2 kompatibel
				0	1	1	0									Revision: 1 = A, 2 = B, 3 = C, ..., 7 = G
			0													Reserviert
0	0	1														Kennung

WORT-1: Initialisierung

In diesem Wort kann eingestellt werden, ob die Schnittstellen E und F nach dem Einschalten und bei einem Hardware-Reset entsprechend den Eintragungen im EEPROM initialisiert werden (Bit-0 = 1) oder nicht (Bit-0 = 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-1: Initialisierung (werks. Einst.)
																geändert am: von:
															1	Init nach Hardware-Reset: 0 = nein, 1 = ja
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0		Reserviert

WORT-2: Bestückung

Dieses Wort darf nicht geändert 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	1	0	0	0	0	WORT-2: Bestückung (werks. Einst.)
												0	0	0	0	Typ SCC: 0 = SCC (85C30) 2 = ESCC (85230)
								0	0	0	1					Max. Takt (SCC): 0 = 6 MHz 1 = 8 MHz 2 = 10 MHz 3 = 16 MHz 4 = 20 MHz

L

WORT-3: Frequenz für SCC3 (wird an PCLK von SCC3 gelegt)

Die Angabe erfolgt mit 4 Ziffern in MHz mit 2 Vorkomma- und 2 Nachkommastellen. 4,9152 MHz würde also aufgerundet auf 4,92 und mit 4 Hexadezimalziffern 0491 (= 0491h) eingegeben. Der Eintrag 0737h bedeutet 7,3728 MHz. Bei Multi-COM-Karten mit ESCC-20 Bausteinen kann auch 1650 eingetragen werden. Dann wird für PCLK der halbe CPU-CLK (=16,5 MHz) eingestellt.

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-3: Frequenz (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
0	0	0	0													1. Ziffer

WORT-4: Bestückung Kanal E

Dieses Wort darf nicht geändert werden!

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	0	0	0	0	1	1	WORT-4: Hardware Kanal E (werks. Einst.)
																geändert am: von:
															1	1 = S-Link möglich 0 = nicht vorgesehen
														1		1 = 2 x 24-poliger Sockel 0 = nicht vorbereitet
													0			1 = S-Link eingelötet 0 = nicht eingelötet
								0	0	0	0	0				Reserviert
							1									1 = Interface möglich 0 = nicht vorgesehen
						1										1 = Sockel 0 = nicht vorbereitet
					0											1 = fest eingelötet 0 = nicht eingelötet
0	0	0	0	0												Reserviert

WORT-5: Bestückung Kanal F

Dieses Wort darf nicht geändert werden!

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	0	0	0	0	1	1	WORT-5: Hardware Kanal F (werks. Einst.)
															1	1 = S-Link möglich 0 = nicht vorgesehen
														1		1 = 2 x 24-poliger Sockel 0 = nicht vorbereitet
													0			1 = S-Link eingelötet 0 = nicht eingelötet
								0	0	0	0	0				Reserviert
							1									1 = Interface möglich 0 = nicht vorgesehen
						1										1 = Sockel 0 = nicht vorbereitet
					0											1 = fest eingelötet 0 = nicht eingelötet
0	0	0	0	0												Reserviert

L

WORT-8: S-Link-Typ Kanal E

In WORT-8 wird der Typ des S-Links für Kanal E automatisch eingetragen.

Typ		Name	Interface	isol.	Anmerkung
0	(0000h)	-	keines	-	-
1	(0001h)	SL-232S	RS-232	nein	-
2	(0002h)	SL-232A	-	-	Reserviert
3	(0003h)	SL-232A/i	RS-232	nein	-
4	(0004h)	SL-232A/o	RS-232	nein	-
8	(0008h)	SL-422S	RS-422	nein	-
9	(0009h)	SL-485S	RS-485	nein	-
10	(000Ah)	SL-485A	-	-	Reserviert
16	(0010h)	SL-232i	RS-232	ja	-
24	(0018h)	SL-422i	RS-422	ja	-
32	(0020h)	SL-485i	RS-485	ja	PROFIBUS-geeignet
40	(0028h)	SL-20MA	20 mA	ja	-
48	(0030h)	SL-LWL/P	LWL (Plastik)	ja	Plastikfaser
49	(0031h)	SL-LWL/G	LWL (Glas)	ja	Glasfaser
56	(0038h)	SL-SSIi	RS-422	ja	2 Kanäle Sync. Ser. Interface
64	(0040h)	SL-CANi	CAN-Bus	ja	CAN-Bus
72	(0048h)	SL-DPSi	RS-485	ja	PROFIBUS-Slave
80	(0050h)	SL-IBSi	RS-485	ja	Interbus-S

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-8: S-Link-Typ Kanal E
(automatische Einstellung)

WORT-9: Revision des S-Link Kanal E

Im Low-Byte von WORT-9 wird die Revision des aufgesteckten S-Links für Kanal E automatisch eingetragen. Kann der Typ, die Revision oder der Hersteller des S-Links nicht ermittelt werden, wird im High-Byte der Fehlercode eingetragen (siehe Anhang F).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	WORT-9: S-Link-Revision Kanal E (automatische Einstellung)
								0	0	0	0	0	0	0	0	Revision: 1 = A, 2 = B, 3 = C, ..., 7 = G
0	0	0	0	0	0	0	0									Fehlerinformation, siehe Anhang F

WORT-10: S-Link-Hersteller Kanal E

In WORT-10 wird der Herstellercode des verwendeten S-Links für Kanal E automatisch eingetragen.

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-10: S-Link-Hersteller Kanal E (automatische Einstellung) 0 = unbekannt, 1 = SORCUS, 2 = Complex International (Ci)

WORT-12: S-Link-Typ für Kanal F

In WORT-12 wird der Typ des S-Links für Kanal F automatisch eingetragen.

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-12: S-Link-Typ Kanal F (auto. Einstellung, siehe WORT-8)

WORT-13: Revision des S-Link Kanal F

Im Low-Byte von WORT-13 wird die Revision des aufgesteckten S-Links für Kanal F automatisch eingetragen. Kann der Typ, die Revision oder der Hersteller des S-Links nicht ermittelt werden, wird im High-Byte der Fehlercode eingetragen (siehe Anhang F).

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	WORT-13: S-Link-Revision Kanal F (automatische Einstellung)
								0	0	0	0	0	0	0	0	Revision: 1 = A, 2 = B, 3 = C, ..., 7 = G
0	0	0	0	0	0	0	0									Fehlerinformation, siehe Anhang F

WORT-14: S-Link-Hersteller Kanal F

In WORT-14 wird der Herstellercode des aufgesteckten S-Links für Kanal F automatisch eingetragen.

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-14: S-Link-Hersteller Kanal F (automatische Einstellung) 0 = unbekannt, 1 = SORCUS, 2 = Complex International (Ci)

M. Das Programm SNW6

Das Programm SNW6 (Schöne Neue Welt) stellt alles zur Verfügung, was Sie zum Arbeiten mit der Multi-COM Karte benötigen. Es werden bis zu acht Karten unterstützt, die parallel (am PC-Bus) 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 SNW6 zu geben und die Fälle abzudecken, in denen die Intuition des Programmierers nicht mit der Ihren übereinstimmt.

M.1. Installation des Programms SNW6

Das Programm SNW6 besteht aus folgenden Dateien:

- SNW6.EXE
- SNW6.OVR
- SNW6.LNK
- SNW6.HCT
- SNW6.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 Programms werden alle Einstellungen in der Datei SNW6.CCF abgelegt. Diese Datei wird immer in das Verzeichnis geschrieben, von dem aus SNW6 gestartet wurde. Jeder hat somit seine eigene Konfigurationsdatei.

Die Installation ist mit dem Kopieren der Dateien schon abgeschlossen. Das Programm wird mit

SNW6

gestartet.

M.2. Allgemeine Hinweise zur Bedienung von SNW6

M.2.1. Menüleiste

Die Menüleiste bleibt während des Arbeitens mit SNW6 immer in der obersten Zeile des Bildschirms sichtbar. Sie enthält alle Programmoptionen. Viele Optionen sind nicht direkt dargestellt, sondern sind in 'Untermenüs' zu finden.

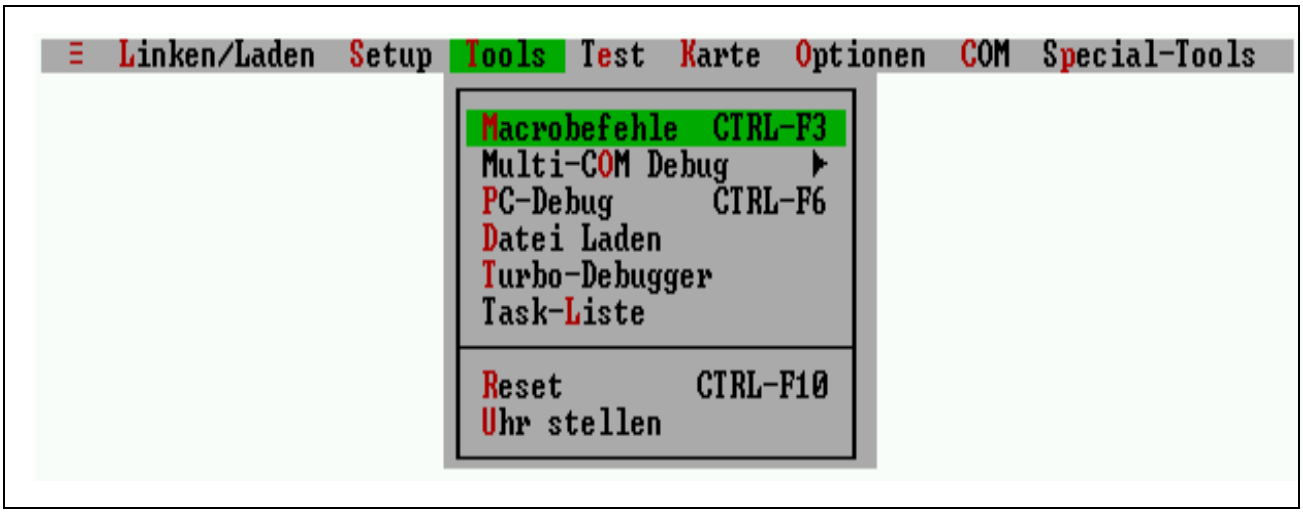


Abb. M-1: Menüleiste mit Untermenü ‘Tools’

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 SORCUS 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 Multi-COM 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 Programms zum Aktualisieren der Multi-COM Karte, 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:

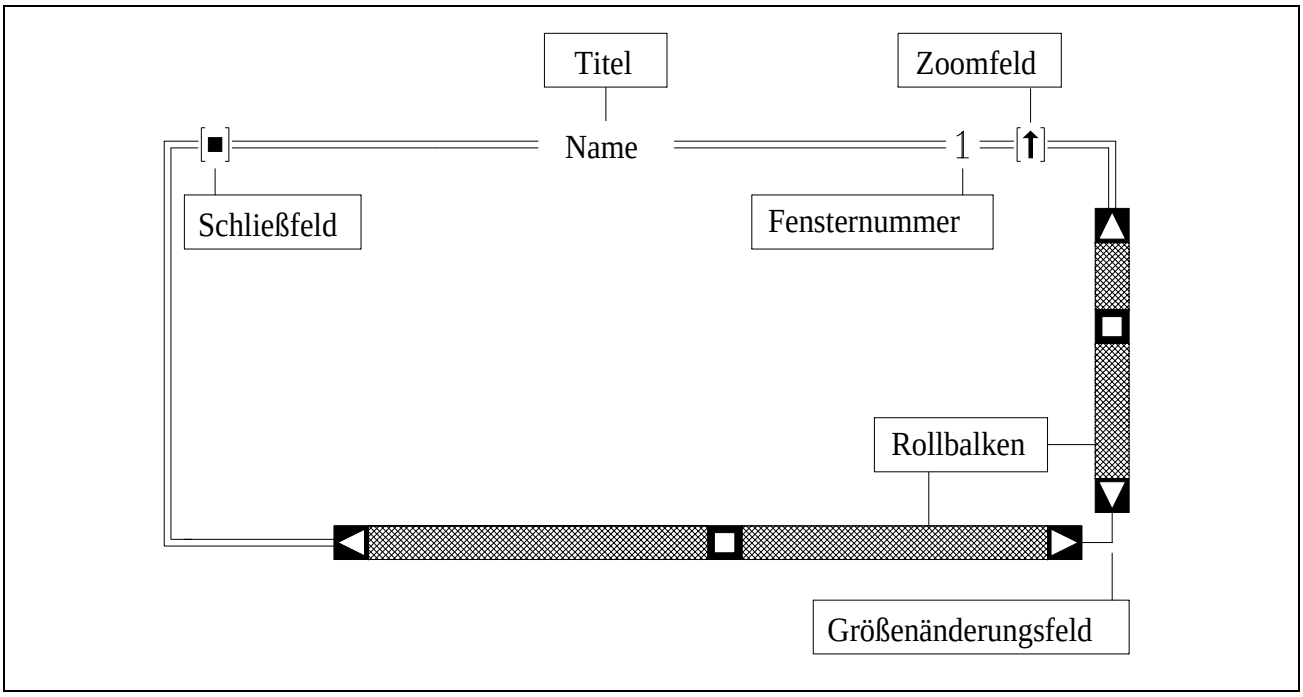


Abb. M-3: Elemente eines Fensters

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 SNW6 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 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. Das Fenster in 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 SNW6

M.3.1. Anwahl einer Karte

Auf dem Bildschirm wird nach Aufruf von SNW6 immer eine Multi-COM Karte dargestellt. Diese ist die 'aktuelle' Karte. Die meisten Funktionen von SNW6 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 Multi-COM Karte verfügt über ein eigenes Echtzeit Multi-Tasking-Betriebssystem (siehe Kapitel 5). Eine zentrale Aufgabe im Umgang mit der Multi-COM 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'.

M6DEVICE	Anwahl einer Multi-COM Karte (optional mit Reset und Laden eines Betriebssystems)
M6INST	Installieren eines Echtzeitprogramms
M6PAR	Parameter einer Task setzen
M6PROC	Prozedur einer Task starten
M6FUNC	Funktion einer Task starten
M6CMD	Makrobefehl zur Multi-COM Karte senden
M6LOADMODUL	Programmierbares S-Link laden

Tab. M-1: Einige Schlüsselwörter in Installationsdateien

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 Multi-COM 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 S-Link 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 Multi-COM Karte nach dem Einschalten automatisch konfiguriert.

Beispiel: **SNW6 /i:TEST.INS**

Gemäß der Installationsdatei TEST.INS werden Echtzeitprogramme auf der Multi-COM Karte installiert. Die Extension '.INS' muss nicht unbedingt angegeben werden.

M.3.3. Bearbeiten der EEPROM-Inhalte von Basiskarte und S-Links

Unter dem Menüpunkt 'Setup' können Sie den Inhalt der EEPROMs der angewählten Karte und der S-Links ansehen und gegebenenfalls ändern. Weil kein Zugriff auf die EEPROMs erlaubt ist, während ein Echtzeitprogramm auf der Multi-COM 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 - Multi-COM

M.3.4.1. Kommunikation über Makrobefehle

Der Menüpunkt Tools/Makrobefehle bietet die Möglichkeit, Makrobefehle zur angewählten Multi-COM 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 'SNW6.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 Multi-COM Karte nachvollzogen werden.

M.3.5. Multi-COM Debug

Unter diesem Punkt können Sie den Speicherinhalt der Multi-COM ansehen und editieren. Für spätere Versionen ist ein Disassembler vorgesehen.

M.3.6. Hardware-Reset der Multi-COM 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 Multi-COM Karte

Diese Option befindet sich im Untermenü 'Tools'. Eine Datei kann damit von Diskette oder Platte an eine beliebige Stelle ins RAM der Multi-COM 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 Multi-COM 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 S-Links

Unter der Option 'Test' finden sich Testmöglichkeiten für die Basiskarte und ihre S-Links. Für S-Links und für Funktionseinheiten der Basiskarte gibt es Testfenster, die die Zustände aller Eingänge der Kanäle anzeigen und die Möglichkeit 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 SNW6 weiterarbeiten. Insbesondere können auch mehrere Testfenster gleichzeitig geöffnet sein. Allerdings ist zu beachten, dass dann im Hintergrund ständig Kommunikation zwischen PC und Multi-COM 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 Karte

Sie können die Leuchtdioden steuern, die seriellen Schnittstellen der Multi-COM Karte testen und den Zustand der Interruptleitungen betrachten.

M.3.10.2. Testen von S-Links

Die Tests für alle gängigen S-Links sind in SNW6 integriert. Sollte ein neueres S-Link nicht enthalten sein, dann werden die Tests entweder als eigenständige Programme nachgeliefert, oder sind in einem Update von SNW6 enthalten. Die eigenständigen Programme haben die Bezeichnung M6TSxxx.EXE (xxx = S-Link-Nummer, dezimal) und können von SNW6 aus aufgerufen werden, wenn sie sich in dem Verzeichnis befinden, von dem aus SNW6 gestartet wurde.

Sofern ein S-Link per Software konfigurierbar ist, wird es entsprechend den Eintragungen im EEPROM eingerichtet. Unabhängig davon, wie es im Augenblick konfiguriert ist und ob es gerade von einem Echtzeit-Programm benutzt wird!

M.3.11. Einrichten von Kommunikationsprogrammen (CQ6)

In der Lieferung Ihrer Multi-COM Karte ist das Programmpaket CQ6 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 Multi-COM 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 SNW6 menügesteuert durchgeführt werden. Einzelheiten dazu finden Sie in Kapitel 13 "Asynchrone serielle Kommunikation".

M.3.12. SNW6 Programmoptionen

M.3.12.1. Zugriffsmodus

Das Programm SNW6 greift in der Regel auf die angewählte Multi-COM Karte zu, um den Zustand der Karte zu prüfen und die Daten der dargestellten Basiskarte und ihrer seriellen Schnittstellen 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 SNW6 greift dann nicht mehr automatisch auf die Multi-COM Karte zu. Die Bildschirmfenster werden nur dann aktualisiert (bzw. initialisiert), wenn 'Aktualisieren' (F2) gewählt wird. Auf der dargestellten Multi-COM 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 SNW6 greift nur dann auf die Multi-COM 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 Programms SNW6 gesperrt sein sollen (unabhängig von der letzten Einstellung, die in SNW6.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 SNW6 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 SNW6 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 'SNW6 /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 Warntons

Das Programm SNW6 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 Multi-COM Karte auf der Karte aktiv werden soll und ob die Uhr automatisch gestellt wird.

- **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 Multi-COM 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 Multi-COM 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 SNW6 die Uhrzeit des PCs gelesen und auf der Multi-COM 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 Multi-COM 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 Multi-COM Karte laden und dort starten (siehe auch Kapitel M.3.12.4 Reset-Verhalten)

M.3.14. Kommandozeilenparameter

Diese Parameter können beim Aufruf von SNW6 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 SNW6 beendet. Damit ist das Einbinden in eine Batchdatei möglich.
/h	unterdrückt alle Bildschirmausgaben beim Laden von Echtzeitprogrammen.
/m	SNW6 mit Monochrom-Monitor starten (Kapitel M.3.12.2)
/q	Automatische Kartenzugriffe sperren (Kapitel M.3.12.1)
/r	Beim Starten von SNW6 einen Reset der Multi-COM durchführen.
/a:adr	Basisadresse vorwählen (adr: hexadezimale Basisadresse).

M.3.15. DOS-Beendigungscodes

Wenn das Programm SNW6 mit dem Kommandozeilenparameter '/i:' gestartet wurde, wird bei Beendigung des Programms 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 M6DEVICE Befehl sein.

M6DEVICE	Anwahl einer Multi-COM Karte (optional mit Hardware-Reset der Karte und Laden eines Betriebssystems)
M6INST	Installieren eines Echtzeitprogramms
M6PAR	Parameter einer Task setzen
M6PROC	Prozedur einer Task aufrufen
M6FUNC	Funktion einer Task aufrufen
M6CMD	Makrobefehl zur Multi-COM Karte senden
M6LOADMODUL	Programmierbares S-Link laden

Für andere Karten existieren noch weitere Befehle (LINKTIME, IF, USES, M4ROM, M4RESET etc.), die aber für Multi-COM Karten nicht verwendet werden dürfen.

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.

M6DEVICE

Mit diesem Befehl wird eine Multi-COM Karte angewählt, optional zurückgesetzt und optional ein Betriebssystem geladen. Alle darauffolgenden Befehle beziehen sich dann auf diese Karte, bis zum nächsten Befehl M6DEVICE, mit dem eine neue Karte gewählt wird. Wenn Sie die in SNW6 angewählte Karte eintragen wollen, dann können Sie im SNW6-Editor die dafür nötige Befehlsfolge mit ALT-D an der Position des Cursors in die Installationsdatei einkopieren. Beachten Sie, dass im automatischen Eintrag von M6DEVICE immer RESET angegeben ist und gegebenenfalls gelöscht werden muss.

Syntax

M6DEVICE *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 Multi-COM 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 Multi-COM Karte beim Ausführen (Laden) der Installationsdatei mit einem Hardware-Reset zurückgesetzt.

osname: Nach einem RESET der Multi-COM 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.

M6INST

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 SNW6 gestartet haben.
- in dem Verzeichnis, in dem die Installationsdatei steht.
- in dem Verzeichnis, in dem das Programm SNW6 steht.

Syntax:

M6INST *filename number task interrupt datasize flags*

filename: Dateiname (max. 8 Zeichen) und Extension (max. 4 Zeichen inkl. 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 Programms übereinstimmen (vierstellig, hexadezimal).

task: Tasknummer (1-3ffh, hexadezimal)

interrupt: Nummer des Interrupts, für den die Task installiert werden soll (zweistellig, hexadezimal, bei NI-Tasks = 00)

datasize: Länge des Datenbereichs der Task (sechsstellig, hexadezimal)

flags: Installierungsflags (achtstellig, hexadezimal).

Der Parameter *flags* 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. 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 **M6INST** übergebenen Werten. Dabei sind Tasktyp und Interrupt immer miteinander gekoppelt. Sofern es das Echtzeitprogramm zulässt, kann der Installierungsbefehl beide Parameter einstellen (*flags* Bit-3 =1) oder die vom Programm voreingestellten Werte übernehmen (*flags* Bit-3=0). Die Größe des Datenbereichs kann der Installie-

rungsbefehl - 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 **M6INST** 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 SNW6 unter 'Linken-Laden/Optionen' die Option 'Mehrfach-Installierung' eingeschaltet haben, prüft SNW6 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. **!**

M6PAR

Mit diesem Befehl können ein oder mehrere Parameter einer Task gesetzt werden. Die Daten werden byteweise übergeben.

Syntax:

M6PAR *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)

M6PROC

Dieser Befehl ruft die Prozedur einer Task auf.

Syntax:

M6PROC *task procnr*

task: Nummer der Task, deren Prozedur aufgerufen werden soll (vierstellig, hexadezimal)

procnr: Nummer der Prozedur, die aufgerufen werden soll (vierstellig, hexadezimal)

M6FUNC

Dieser Befehl ruft die Funktion einer Task auf. Falls die Funktion eine Antwort zurückgibt, erscheint diese im Informationsfenster (in SNW6 die Echostufe auf 'jede Ausführung bestätigen' stellen).

Syntax:

M6FUNC *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).

M6CMD

Mit diesem Befehl wird beim Laden ein Makrobefehl zur angewählten Karte gesendet. Falls die Multi-COM Karte eine Antwort auf den Makrobefehl zurückgibt, erscheint diese im Informationsfenster (in SNW6 die Echostufe auf 'jede Ausführung bestätigen' stellen).

Syntax:

M6CMD *code format p1* [*p2*] .. [*pn*]

code: Makrobefehlscode (zweistellig, hexadezimal)

format: Formatbyte (siehe Kapitel 12, zweistellig, hexadezimal)

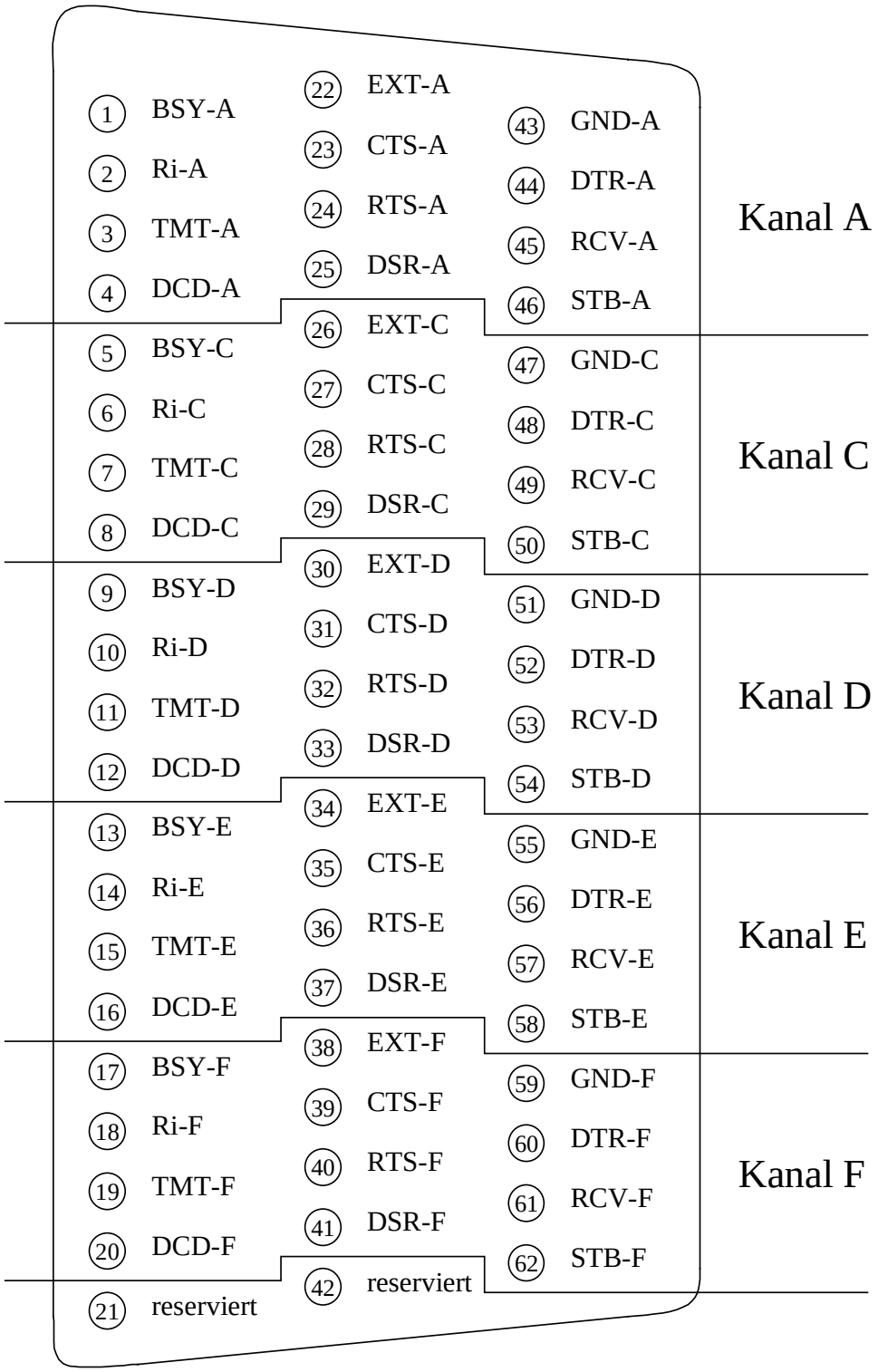
p1..pn: Parameter des Makrobefehls (zweistellig, hexadezimal, Angabe byteweise).

Kommentartext

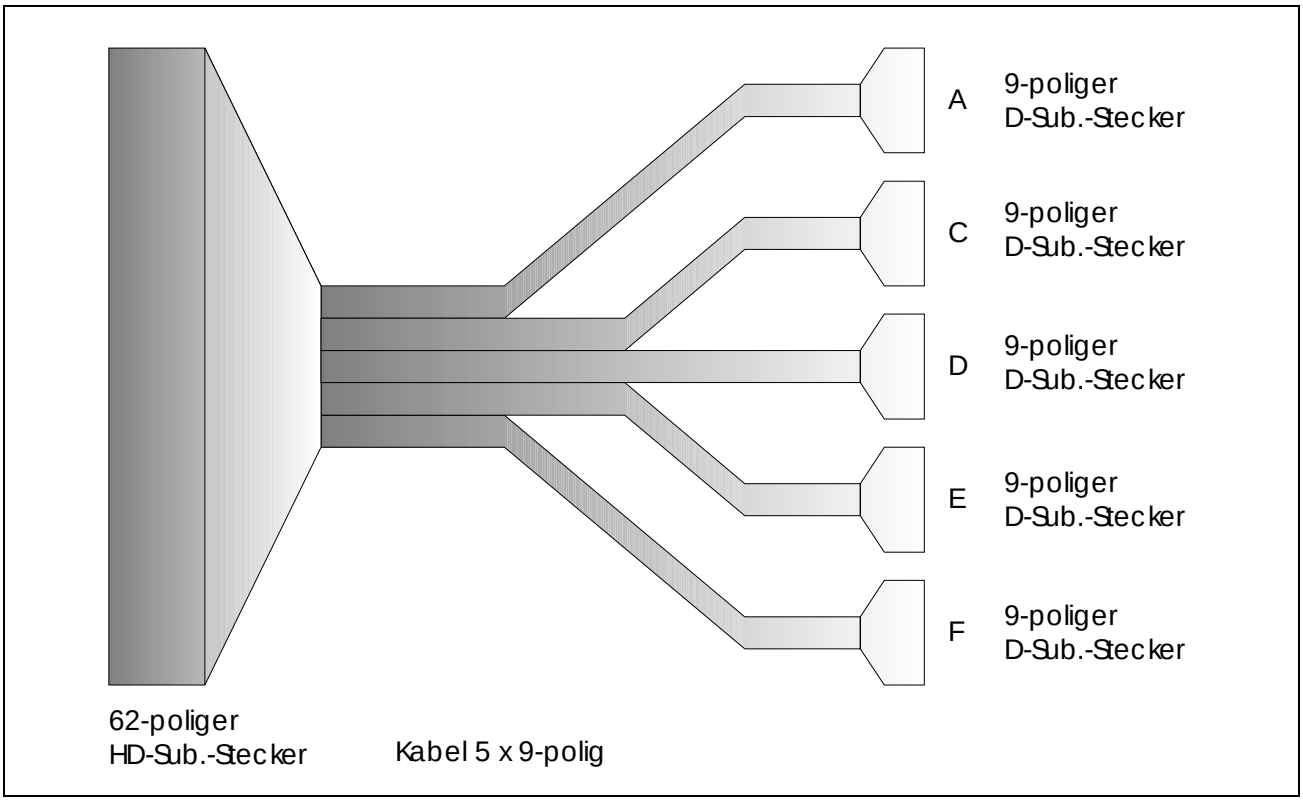
Nach einem Strichpunkt (;) oder einem Hochkomma (') kann beliebiger Text stehen. Er wird komplett (bis zum Ende der Zeile) ignoriert. Eventuell enthaltene Befehls-Schlüsselwörter werden natürlich auch ignoriert.

Notizen

O. inbelegung der 62-poligen Buchse (St1)



Fertig konfektioniertes Kabel (Best.-Nr. K2-6259 und K3-6260)



Signal-Name	S-Link Pin-Nr.	D-Sub. 9-pol.	62-pol. Buchse St1 Kanal A	Kanal C	Kanal D	Kanal E	Kanal F
DCD	A1	1	4	8	12	16	20
RCV	A3	2	45	49	53	57	61
TMT	A5	3	3	7	11	15	19
DTR	A20	4	44	48	52	56	60
GND	A22	5	43	47	51	55	59
DSR	A4	6	25	29	33	37	41
RTS	A6	7	24	28	32	36	40
CTS	A19	8	23	27	31	35	39
Ri	A21	9	2	6	10	14	18
EXT	A23	–	22	26	30	34	38
STB	A2	–	46	50	54	58	62
BSY	A24	–	1	5	9	13	17

Anm.: Pin 21 und 42 sind reserviert und müssen unbeschaltet bleiben.

Stecker St2, serielle Schnittstelle B

Bezeichnung	**St2, Mini-DIN, Pin**		**St3, Pin**
	(3-pol.)	**(8-pol.)**	**Pfostenstecker (20-pol.)**
DTR	-	1	7
RTS	-	2	4
TMT	1	3	5
CTS	-	4	6
RCV	2	5	3
DCD	-	6	1
GND	3	7	9
Ri	-	8	8
DSR	-	-	2

Stecker St3

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	LEDext-Ausgang für externe LED (Kathode), Anode an +5 Volt anschließen.
11	Ausgang	Watchdog Ausgang, TTL-Pegel, active low
12	Eingang	Hardware-Reset, TTL, act. low, int. Pull-up-Widerstand
13	Ein-/Ausgang	+5 Volt
14	Ausgang	LEDint-Ausgang für 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

Stecker St4

Der 8-pol. Diagnose-System-Stecker ist nicht für Anwendungen verwendbar.

Stecker St5

Dieser 50-pol. Stecker (2 x 25-pol.) wird nur alternativ zu St1 bestückt.

Signal-	**50-pol. Pfostenstecker**				
Name	**Kanal A**	**Kanal C**	**Kanal D**	**Kanal E**	**Kanal F**
DCD	1	11	21	31	41
DSR	2	12	22	32	42
RCV	3	13	23	33	43
RTS	4	14	24	34	44
TMT	5	15	25	35	45
CTS	6	16	26	36	46
DTR	7	17	27	37	47
Ri	8	18	28	38	48
GND	9	19	29	39	49
EXT	10	20	30	40	50

P. Stichwortverzeichnis

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z