7. Echtzeitprogrammierung

7.1. Einführung

Wie alle SORCUS-Karten besitzt die Multi-LAB/2 ein eigenes echtzeitfähiges Multi-Tasking-Betriebssystem. 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. Wenn Sie eine Multi-LAB/2i oder /2h Karte besitzen, können Sie auch den Turbo-Debugger verwenden, um Ihre Programme im Quelltext zu untersuchen.

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

In Kapitel 8 finden Sie eine Beschreibung zum Remote-Debuggen auf der Multi-LAB/2i oder /2h Karte.

In Kapitel 9 finden Sie eine Referenz aller Subroutinen der Echtzeitbibliothek ML2RTBIB für PASCAL und C.

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

Die im folgenden dargestellten Programme sind nur auf einer Multi-LAB/2 lauffähig. Diese Programme 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 aufgezeigt.

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 laufend 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 Programmes
	Datenbereich des Programmes
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 Programm-Nummer, Version und Revision des Programmes, die Lage (Adressen) der Prozeduren innerhalb des Programmes, usw. Jedes Programm muß dafür sorgen, daß die PDT eingerichtet, ausgefüllt und dem Betriebssystem bekanntgemacht wird (siehe Seite 7-11, '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	Programm-Nummer (z.B. 1)
6	Char	Programm-Version (z.B. '1')
7	Char	Programm-Revision (z.B. 'A')
8	Byte	Prozessor-Typ (0=8086, 1=V20/80186, 4=486)
9	Byte	Co-Prozessor-Typ (0 = kein Co-Prozessor)
10	Byte	Programmiersprache des Programmes
11	Byte	Programm-Typ
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	Minimale Größe des Datenbereichs in Byte
28	Long	Maximale Größe des Datenbereichs in Byte
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 Anwenderprozeduren)

7.3.1.1. Die Flags

Innerhalb der obenstehenden PDT finden Sie an der Position 12 den Eintrag "**Flags**". Dabei handelt es sich um ein Wort (2 Byte = 16 Bit), in dem jedes Bit 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
0..2	Tasktyp: 000: Nicht-Interrupt-Task (**_NI_TASK**) 001: Indirekte Interrupt-Task (**_II_TASK**) 010: Direkte Interrupt-Task (**_DI-TASK)**
3	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	Hypertext vorhanden (Bit=1) (**_HYPER_TEXT** = 128)
8	Der Datenbereich wird vom Betriebssystem (Bit=0) oder vom Programm selbst reserviert (Bit=1). (**_LOCAL_DATA** = 256)
9	Datenbereichsgröße variabel (Bit=0) oder fest (Bit=1) (**_FIXED_DATASIZE** = 512)
10	Der Parameterbereich wird vom Betriebssystem (Bit=0) oder vom Programm reserviert (Bit=1). (**_LOCAL_PARAMETER** = 1024)
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 Byte bestehen (z.B. Integerwerte), belegen mehrere Nummern. Um auf solch einen Parameter zuzugreifen, wird immer die erste Nummer angegeben.

Der Parameterbereich kann entweder Teil des Programmes sein oder vom Betriebssystem bei der Installierung des Programmes reserviert werden. Wenn das Programm den Paramterbereich 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 Programmes ist der Zugriff auf den Parameterbereich dann einfach ein Zugriff auf die deklarierte Datenstruktur.

Beispiel:

PASCAL

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

C

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

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

PASCAL

```
blinkrate := ml2rt_get_par_word(task,1);
led_status := ml2rt_get_par_byte(task,3);
```

C

```
blinkrate = ml2rt_get_par_word(task,1);
led_status = ml2rt_get_par_byte(task,3);
```

Beachten Sie die Numerierung der Parameter. **blinkrate** hat die Nummer 1 und ist ein Wort, **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 (ml2rt_get_par_address) und mit Pointern arbeiten oder mit Hilfe von Systemaufrufen Parameter lesen (ml2rt_read_par_xxxx) und schreiben (ml2rt_write_par_xxxx).

Üblicherweise werden im Parameterbereich Daten zur Konfiguration des Programmes 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 Programmes und kann innerhalb des Programmes 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 MLC bei M2INST oder von der PC-Bibliothek bei ml2_transfer_and_install übergeben wird) angegeben, ob einer der Einträge Größe, maximale Größe oder minimale Größe den Datenbereich bestimmt 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 7-10) und mit eigenen Zeigern arbeiten oder Systemaufrufe und die Betriebssystemzeiger benutzen (ml2rt_reset_x_pointer, ml2rt_move_x_pointer, ml2rt_read_data_xxxx, ml2rt_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 Programmes Verwendung finden. Sie werden genauso wie bei PC-Programmen eingesetzt. Im weiteren Verlauf dieses Kapitels 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 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 Byte, die der Funktion übergeben werden sollen.
4) Ein Pointer, der auf die Datenbyte zeigt, die an die Funktion übergeben werden sollen (Quellspeicher).
5) Anzahl Byte, 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 (ML2BIB) und auf der Karte (ML2RTBIB) zur Verfügung.

7.3.5. Task-Deskriptor-Tabelle (TDT)

Bevor ein Programm gestartet werden kann, muß es auf die Multi-LAB/2 geladen und dort installiert werden. Zur Erinnerung: Installieren bedeutet, daß Ihr Programm auf die Karte geladen und dem Betriebssystem bekanntgemacht wird.

Im Detail heißt dies, 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 Interrupt-Nummer, Tasknummer sowie die Schreib- und Lesepointer auf den Datenbereich in dieser Tabelle gehalten. Eine ausführliche Beschreibung der TDT finden Sie in Anhang J.

rel. Adr.	Feldbezeichner	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 Byte)
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) wird vom 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-LAB/2 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 Haupt-Prozedur wäre, wird vom Betriebssystem der Multi-LAB/2 Karte nur zum Zeitpunkt der Installierung einmal aufgerufen. Sie hat die Aufgabe, die PDT zu initialisieren und dem Betriebssystem bekanntzumachen (ml2rt_set_pdt_adr). Dieser Teil wird als PREPARE bezeichnet.

Im Unterschied zur DOS-Programmierung müssen Sie sich beim Erstellen von Echtzeitprogrammen selbst um die "Anmeldung" des Programmes kümmern, da die Compiler keine Informationen über das Betriebssystem der Multi-LAB/2 Karte haben, um die notwendigen Tabellen selbst anzulegen. Das erledigen Sie, wie schon erwähnt, im PREPARE-Teil des Programmes. Achten Sie unbedingt darauf, daß alle Einträge in der PDT zu Ihrem Programm passen.

7.5. Allgemeines zu den Beispielprogrammen

Im folgenden soll an einfachen Beispielprogrammen in Borland Pascal und C++ gezeigt werden, wie Programme in Hochsprachen erstellt werden. Alle Programme finden Sie auch auf den mitgelieferten Disketten, 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 die LED auf der Karte blinken. Einmal als NI-Task und einmal als DI-Task. Bei diesen Programmen ist der Aufbau des Parameterbereichs gleich. Es wird kein Datenbereich benutzt.

Parameter der Blink-LED-Programme:

Nr.	Typ	Erklärung
0	Byte	Status des Programmes: 0 = Programm bereit 2 = Programm läuft 4 = Programmlauf abgebrochen
1	Word	Blink-Rate der LED Bei der NI-Task wird hier die Anzahl der NI-Task-Aufrufe eingetragen, nach denen die LED umgeschaltet wird. Bei der DI-Task wird hier die Blink-Frequenz in Hertz angegeben.
3	Byte	Status der LED: 0 = LED ist aus 1 = LED ist an

Alle Programme benutzen die Echtzeitbibliothek "ML2RTBIB". Nähere Informationen zu dieser Bibliothek finden Sie in Kapitel 9 und in der Header-Datei "ML2RTBIB.H" bzw. in der Datei "ML2RTBIB.PAS".

Als Entwicklungsumgebung können Sie Borland Pascal für DOS oder Borland C++ für DOS verwenden. Ob diese integrierten Entwicklungsumgebungen unter DOS, WINDOWS oder OS/2 bzw. 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-LAB/2 ablegen können, z.B. "C:\SORCUS\ML2\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-LAB/2 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 Programmes 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 Echtzeitprogramme. 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\ML2\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-Programmes 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-LAB/2 erstellen. Beachten Sie dabei die folgenden Einschränkungen und Hinweise:

- *Es sind keine Bildschirm- (Grafik und Text) oder Tastaturfunktionen möglich.*
- *Es kann und darf keine Overlay-Technik verwendet werden.*
- *Es können keine Datei-Operationen durchgeführt werden.*
- *Es kann kein Speicher dynamisch reserviert bzw. freigegeben werden ("GETMEM", "NEW", "DISPOSE" etc.). Dafür stehen spezielle Bibliotheks-Routinen zur Verfügung.* ***Pointeroperationen können normal verwendet werden.***
- *Es darf kein Range-Checking aktiviert sein. Eine Zusammenstellung der Compilerschalter folgt auf Seite 7-16.*
- *Es dürfen keine DOS-spezifischen Funktionen benutzt werden (DOS-Interrupts).*
- *Das Einfügen von Debug-Informationen hat keinen Einfluß auf die Ausführung des Programmes.*
- *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 Programmier-Handbuch von Borland Pascal im Kapitel "Interne Details, Direkter Zugriff auf I/O-Adressen".*

Folgende Standardfunktionen 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-Checking ausschalten
{$S-}	Stack-Checking ausschalten
{$I-}	I/O-Checking ausschalten
{$G-}	8086-Code erzeugen
{$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

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 Compilerswitches 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 "M2P0300.PAS" auf den mitgelieferten Disketten.

```
{$F+}                                       { Prozeduren FAR deklarieren }
{$G-}                                       { 8086/88-Code erzeugen }
{$N-}                                       { Co-Prozessor aus }
{$E-}                                       { keine Emulation }
{$R-}                                       { kein Range-Checking }
{$S-}                                       { kein Stack-Checking }
{$I-}                                       { kein I/O-Checking }
{$V-}                                       { keine strikte Stringübergabe }
{$L+}                                       { Local Symbols für Turbo-Debugger }
{$M 1024, 0, 0}                             { Stack und Heap auf Minimum }

uses ml2rtbib;                              { Die Multi-LAB/2 on-board Bibliothek einbinden }

Const                          { ***** ALLGEMEINE ANGABEN ***** }
   Programm_Number      : Word = $300;      { Programm-Nummer = 300h }
   Version              : Char = '1';       { Version des Programmes }
   Revision             : Char = 'A';       { Revision des Programmes }

                     { ***** Die verschiedenen Statusmöglichkeiten des Programmes ***** }
   READY                = 0;                { Bereit }
   RUNNING              = 2;                { Programm läuft }
   STOPPED              = 4;                { Programm angehalten }

                     { ***** Zustände der LED ***** }
   EIN                  = 1;
   AUS                  = 0;

                     { ***** Aufbau der Programm-Deskriptor-Tabelle (PDT) ***** }
Type
   PDT_Type   = Record
            Typ           : Byte;     { Typ des PDT-Vorspanns }
            Lengt         : Byte;     { Länge des Vorspanns inkl. Prozedurangaben }
            Proc_Num      : Word;     { Anzahl der Prozeduren }
            Prg_Number    : Word;     { Nummer des Programmes }
            Version       : Char;     { Version des Programmes. Z.B.: A,B,C ... }
            Revision      : Char;     { Revision des Programmes. Z.B.: 1,2,3 ... }

            CPU_Type      : Byte;     { Prozessor-Typ }
            COProz_Type   : Byte;     { Co-Prozessor-Typ }
            Language      : Byte;     { Programmiersprache }
            Prg_Type      : Byte;     { Programm-Typ }

            Flags         : Word;     { Installierungsflags }
            Interrupt_Num : Word;     { Nummer des benutzten Interrupts }

            Data_Adr      : LongInt;  { Anfangsadresse des Datenbereichs (phys. Adresse) }
            Data_Num      : LongInt;  { Größe des Datenbereichs }
            Data_Min      : LongInt;  { Datenbereich, den das Programm mindestens benötigt }
            Data_Max      : LongInt;  { Maximaler Datenbereich, den das Prg. verarbeiten
                                        kann}

            Parameter_Adr : LongInt;  { Adresse des Parameterbereichs (phys. Adresse) }
            Parameter_Num : Word;     { Anzahl der Parameter }

            Hyper_Adr     : LongInt;  { Anfangsadresse des Hypertextes (phys. Adresse) }
            Res1          : Word;
            Res2          : LongInt;

            Main          : Pointer;  { Adresse der Haupt-Prozedur (Task-Prozedur) }
            Auto_Init     : Pointer;  { Adresse der Auto-Init-Prozedur }
            Start         : Pointer;  { Adresse der Start-Prozedur }
            Stop          : Pointer;  { Adresse der Stop-Prozedur }
         end;
```

```
            { ***** DEKLARATION DER GLOBALEN PROGRAMMDATEN ***** }
VAR
   PDT        : PDT_Type;              { Deklaration des Vorspanns }

            { ***** Parameterbereich des Programmes deklarieren. ***** }
            { Auf diese Daten kann später über das Betriebssystem zugegriffen werden. }
   Parameter  : Record
            TDT       : TDT_Type;     { Platz für die TDT reservieren }
            Status    : Byte;         { Status des Programmes }
            Blink_Rate: Word;         { Blinkrate }
            LED_Status: Byte;         { Zustand der LED }
         end;

   Pause      : Word;


Procedure Auto_Init;         { ***** AUTO-INITIALISIERUNG }
begin
   ml2rt_entry;                       { Register retten und vorbereiten }
   Parameter.Status := READY;         { Status des Programmes auf bereit }
   Parameter.Blink_Rate := 3000;      { Blinkrate defaultmäßig einstellen }
   Parameter.LED_Status := AUS;       { Status der LED auf "AUS" }
   ml2rt_led_Off;                     { On-Board LED ausschalten }
   ml2rt_exit;                        { Register wieder restaurieren }
end;


Procedure Start;             { ***** Start-Prozedur ***** }
begin
   ml2rt_entry;
   Parameter.Status := RUNNING;           { TDT enthält die Tasknummer, unter der das Programm
                                               installiert wurde}
   ml2rt_wakeup_task (Parameter.TDT.Task);
   Pause := 0;
   ml2rt_exit;
end;


Procedure Stop;              { ***** Stop-Prozedur ***** }
begin
   ml2rt_entry;
   ml2rt_sleep_task (Parameter.TDT.Task); { Task deaktivieren }
   Parameter.Status := STOPPED;           { Parameter auf "Programm angehalten". }
   ml2rt_exit;
end;


Procedure Main_Proc;         { ***** HAUPTPROZEDUR (NI-Task) ***** }
begin
   ml2rt_entry;
   Pause := Pause + 1;
   if Pause>=Parameter.Blink_Rate then       { Falls die Pause (Anzahl Aufrufe der Main-Proc)
                                               den angegebenen Wert erreicht hat, }
   begin                                  { ... die LED umschalten }
      if Parameter.LED_Status = EIN then
      begin                               { Falls die LED eingeschaltet ist, ... }
        ml2rt_led_off;                    { ... LED ausschalten und ... }
        Parameter.LED_Status := AUS;      { ... Zustand merken }
      end
      else                                { Falls die LED ausgeschaltet ist, ... }
      begin
        ml2rt_led_on;                     { ... LED einschalten und ... }
        Parameter.LED_Status := EIN;      { ... Zustand merken }
      end;
       Pause := 0;                         { Durchlaufzähler wieder auf Null setzen }
   end;
   ml2rt_Exit;
end;
```

```
Procedure Init_PDT;
begin                                    { Die Prozedur stellt alle Werte in der PDT ein }
   PDT.Typ := 1;                         { Programm-Deskriptor-Tabellen-Typ
                                              (z.Zt. nur Typ 1 unterstützt) }
   PDT.Lengt := _PDT_Length;             { Größe des PDT-Vorspanns
                                              (bei Typ 1 immer = 48) }
   PDT.Proc_Num := 4;                    { Anzahl der Prozeduren }
   PDT.Prg_Number := Programm_Number;    { Programm-Nummer }
   PDT.Version := Version;               { Programm-Version }
   PDT.Revision := Revision;             { Programm-Revision }
   PDT.CPU_Type := _186;                 { CPU-Typ }
   PDT.CoProz_Type := _NOCOPROZ;         { Co-Prozessor-Typ }
   PDT.Language := _TP70;                { Programmiersprache }
   PDT.Prg_Type := _User_Prg;            { Programm-Typ: Anwenderprogramm }
   PDT.Flags := $0;                      { Flags }
   PDT.Flags := _NI_Task + _PDT_Info_Enable + _Local_Data + _Local_Parameter+ _Fixed_Datasize;
   PDT.Interrupt_Num := 0;               { Interrupt-Nr., für die das Prg. gedacht ist
                                             (ungültig, da NI-Task) }
   PDT.Data_Adr := 0;                    { Adresse des Datenbereichs }
   PDT.Data_Num := 0;                    { Größe des Datenbereichs }
   PDT.Data_Min := 0;                    { Minimaler Datenbereich }
   PDT.Data_Max := 0;                    { Maximaler Datenbereich }
                                       { Anfangsadresse des Parameterbereichs }
   ml2rt_phys_adr (@Parameter, PDT.Parameter_Adr);
   PDT.Parameter_Adr := PDT.Parameter_Adr + SizeOf(TDT_Type);
                                       { Anzahl der Parameter }
   PDT.Parameter_Num := SizeOf(Parameter) - SizeOf(TDT_Type);
   PDT.Hyper_Adr := 0;                   { Adresse des Hypertextes (wird nicht benutzt) }
   PDT.Res1 := 0;
   PDT.Res2 := 0;
   PDT.Main := @Main_Proc;               { Adresse der Hauptprozedur }
   PDT.Auto_Init := @Auto_Init;          { Adresse der Auto-Init-Prozedur }
   PDT.Start := @Start;                  { Adresse der Start-Prozedur }
   PDT.Stop := @Stop;                    { Adresse der Stop-Prozedur }
end;
begin                          { ***** PREPARE ***** }
   Init_PDT;                             { PDT initialisieren }
   ml2rt_Set_PDT_Adr (PDT);              { Adresse der PDT im Rückgabe-Record
                                              dem Betriebssystem übergeben }
end.
```


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

Als erstes wird die Unit "ML2RTBIB" 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. "ML2RTBIB.PAS" benötigt die Datei "ML2RTBIB.OBJ". Stellen Sie zur Compilierung auch den Pfad auf die "ML2RTBIB.OBJ"-Datei in "Options/ Directories/*.OBJ-Dateien" ein.

Anschließend wird die PDT deklariert. Sie sehen am Ende der PDT, daß das Programm vier Prozeduren enthält.

Außerdem wird noch ein Record vereinbart, der die Parameter enthält. Die Parameter werden also lokal im Programm gehalten, der Parameterbereich wird nicht vom Betriebssystem reserviert. Beachten Sie bitte, daß vor den eigentlichen Parametern Platz für die TDT reserviert wird.

Alle globalen (also in der PDT aufgeführten Prozeduren) beginnen mit **ml2rt_entry** und enden mit **ml2rt_exit**. Dadurch ist sichergestellt, daß alle Prozessor-Register gesichert und für die Prozedur richtig eingestellt werden.

In der "Auto-Init-Prozedur" werden die Parameter initialisiert und die LED ausgeschaltet. In der Prozedur "Init_PDT" wird die PDT initialisiert. Beachten Sie bitte die Einstellung der PDT-Flags.

Der Teil, der zwischen "**begin**" und "**end.**" steht, enthält in einem "normalen" Pascal-Programm das eigentliche Hauptprogramm. Dieser Teil wird während der Installierung abgearbeitet. Er sorgt dafür, daß die PDT initialisiert wird (Init_PDT) und dem Multi-LAB/2 Betriebssystem die Adresse der PDT mitgeteilt wird (ml2_set_pdt_adr).

7.6.5.2. Die Installierung der NI-Task

Nachdem das Programm "M2P0300.PAS" einwandfrei übersetzt und die Datei "M2P0300.EXE" erzeugt wurde, kann das Programm auf die Karte geladen und gestartet werden. Geben Sie dazu bitte folgende Installationsdatei mit Hilfe von MLC ein, und laden Sie diese Datei. Sie können auch die Beispielinstallationsdatei "NILED.INS" benutzen.

```
; Beispielinstallation: LED-Blinken als NI-Task (ohne Interrupt)
;
;
; Karte 0 anwählen, Adresse einstellen, Reset auslösen und
; Betriebssystem laden
M2DEVICE 0 0380 TIMEOUT=10 RESET ml2-??.??R
;
;
; Programm-Nummer 300h unter Tasknummer 1 installieren (nicht aktivieren)
M2INST M2P0300.EXE 0300 0001 01 000000 00000980
;
; Parameter der Task 1 setzen
; PAR-1,2 : Blinkrate der LED    => 1000h
; PAR-3   : Status der LED       => 0 = Aus
;
M2PAR 01 01 00 10 00
;
;
; Durch Aufruf der Start-Prozedur Programm starten (Proz. 2)
M2PROC 01 02
```

Diese Installationsdatei installiert die NI-Task, stellt die Blinkrate ein und startet das Programm durch Aufruf der Prozedur 2. Beachten Sie, daß die Art der Angabe des Betriebssystems (ml2-??.??R) erst ab der MLC Version 1.D unterstützt wird. Falls Sie ein älteres MLC benutzen, müssen Sie den kompletten Betriebssystemnamen inklusive Suchpfad angeben.

7.6.5.3. DI-Task in Borland Pascal

Das folgende Listing zeigt das Blink-LED-Programm als DI-Task. Dieses Programm finden Sie auch unter dem Namen "M2P0301.PAS" auf der mitgelieferten Diskette. Das Programm wird unter dem Interrupt des Timer-A installiert. Die Blinkrate kann in Hertz angegeben werden.

```
{$G-}                                   { 8086/88 Code erzeugen }
{$N-}                                   { Coprozessor aus }
{$E-}                                   { keine Emulation }
{$R-}                                   { kein Range-Check }
{$S-}                                   { kein Stack-Checking }
{$I-}                                   { kein IO-Checking }
{$V-}                                   { Keine strikte Stringübergabe }
{$L+}                                   { Local-Symbols für Turbo-Debugger }
{$M 1024, 0, 0}                         { Stack und Heap auf Minimum }

uses ml2rtbib;                          { Realtime Bibliothek einbinden }

{ ***** ALLGEMEINE ANGABEN ***** }
Const
   Programm_Number      : Word = $301;    { Programmnummer = 301h }
   Version              : Char = '1';     { Version des Porgramms }
   Revision             : Char = 'A';     { Revision des Programms }

   { ***** Die verschiedenen Statusmöglichkeiten des Programms ***** }
   READY                = 0;              { Bereit }
   RUNNING              = 2;              { Programm laeuft }
   STOPPED              = 4;              { Programm abgebrochen }

   { ***** Zustaende der LED ***** }
   EIN                  = 1;
   AUS                  = 0;

   { Timer A 12-bit-> (4096 / 2), da Periodendauer T = t(LED_ON) + t(LED_OFF)}
   TIMER_MAX            : longint = 2000;
   { Timer A Basistakt 500kHz }
   TIMER_FREQ           : longint = 500000;

{ ***** Aufbau der Programm-Descriptor-Tabelle (PDT) ***** }
Type
   PDT_Type = Record
                Typ           : Byte;     { Typ der Kopftabelle }
                Lengt         : Byte;     { Laenge der Kopftabelle }
                Proc_Num      : Word;     { Anzahl der Prozeduren }
                Prg_Number    : Word;     { Nummer des Programms }
                Version       : Char;     { Version des Programms }
                Revision      : Char;     { Revision des Programms }

                CPU_Type      : Byte;     { Prozessor-Typ }
                COProz_Type   : Byte;     { Co-Prozessor-Typ }
                Language      : Byte;     { Programmiersprache }
                Prg_Type      : Byte;     { Programm-Typ }

                Flags         : Word;     { Installierungsflags }
                Interrupt_Num : Word;     { Nummer des benutzten Interrupts }

                Data_Adr      : LongInt;  { Anfangsadresse des Datenbereichs }
                Data_Num      : LongInt;  { Größe des Datenbereichs }
                Data_Min      : LongInt;  { Min. Datenbereich fuer Programm }
                Data_Max      : LongInt;  { Max. Datenbereich fuer Programm }

                Parameter_Adr : LongInt;  { Adresse des Parameterbereichs }
                Parameter_Num : Word;     { Anzahl der Parameter }

                Hyper_Adr     : LongInt;  { Anfangsadresse des Hypertext }
                Res1          : Word;
                Res2          : LongInt;

                Main          : Pointer;  { Adresse der Haupt-Prozedur }
                Auto_Init     : Pointer;  { Adresse der Auto-Init-Prozedur }
                Start         : Pointer;  { Adresse der Start-Prozedur }
                Stop          : Pointer;  { Adresse der Stop-Prozedur }
              end;
```

```
{ ***** DEKLARATION DER GLOBALEN PROGRAMMDATEN ***** }
VAR
   PDT                  : PDT_Type;      { Deklaration der Kopftabelle }

   { ***** Parameterbereich des Programms deklarieren. ***** }
   { Auf diese Daten kann spaeter über das Betriebssystem zugegriffen werden. }
   Parameter  :Record
                TDT       : TDT_Type;     { Platz für die TDT reservieren }
                Status    : Byte;         { Status des Programms }
                Blink_Rate: word;         { Blinkrate in Hz }
                LED_Status: Byte;         { Zustand der LED }
                Soft_Cnt  : word;         { Soft-Counter, noetig um mit Timer }
                                          { auch niedrige Freq. zu erzeugen }
                Soft_Init : word;         { Soft-Counter Initialisierung }
              end;

{ ***** AUTO-INITIALISIERUNG ***** }
Procedure Auto_Init;
begin
   ml2rt_entry;                           { Register retten und vorbereiten }
   Parameter.Status := READY;             { Status des Programms auf bereit }
   Parameter.Blink_Rate := 1;             { Blinkrate defaultmaeßig 1 Hz }
   Parameter.LED_Status := AUS;           { Status der LED auf "AUS" }
   ml2rt_LED_Off;
   ml2rt_exit;                            { Register wieder restaurieren }
end;


{ ***** Programm starten ***** }
Procedure Start;
VAR
   timer                : word;
begin
   ml2rt_entry;

   { Softteiler ausrechnen (muss ganzzahlig sein, keine Fliesskommaop.!):  }
   parameter.soft_init := TIMER_FREQ div (2 * parameter.blink_rate * TIMER_MAX);
   {* falls Softzaehler auf 0 "abgerundet" wird, setze Zaehler auf 1 ! *}
   if parameter.soft_init = 0 then
      parameter.soft_init := 1;
   parameter.soft_cnt := parameter.soft_init;

   { Timerwert ausrechnen: Fehler durch Softteiler (ganzzahlig!) wird durch
     Anpassung des maximalen Timerwerts kompensiert!     }
   timer := TIMER_FREQ div (2 * parameter.blink_rate * parameter.soft_init);

   ml2rt_set_timer (TIMER_A, timer, TIMER_MODE); { Timer-A starten }
   Parameter.Status := RUNNING;
   ml2rt_unmask_int (IRQ_TIMER_A);               { Timer-A demaskieren }
   ml2rt_exit;
end;


{ ***** Programmlauf abbrechen ***** }
Procedure Stop;
begin
   ml2rt_entry;
   ml2rt_mask_int (IRQ_TIMER_A);                 { Timer-A Interrupt maskieren }
   Parameter.Status := STOPPED;                  { Parameter setzen }
   ml2rt_exit;
end;
```

```
{ ***** HAUPTPROZEDUR (DI-Task) ***** }
Procedure Main_Proc;
begin
   ml2rt_entry;

   Parameter.Soft_Cnt := Parameter.Soft_Cnt - 1;

   if Parameter.Soft_Cnt = 0 then         { Soft - Counter schon auf 0 ? }
   begin
      if Parameter.LED_Status = EIN then
      begin                               { Falls die LED eingeschaltet ist, }
         ml2rt_LED_Off;                   { ... LED ausschalten und ... }
         Parameter.LED_Status := AUS;     { ... Zustand merken }
      end
      else                                { Falls die LED ausgeschaltet ist, }
      begin
         ml2rt_LED_On;                    { ... LED einschalten und ... }
         Parameter.LED_Status := EIN;     { ... Zustand merken }
      end;
      Parameter.Soft_Cnt := Parameter.Soft_Init;
   end;

   ml2rt_end_of_int (IRQ_TIMER_A);        { EOI an den Interrupt-Contr. }
   ml2rt_exit_interrupt;                  { ISR verlassen }
end;


Procedure Init_PDT;
begin                                     { Prozedurtyp (Standard = 1) }
   PDT.Typ := 1;                          { Programm-Desriptor-Tabellen-Typ }
   PDT.Lengt := _PDT_Length;              { Größe des Vorspanns }
   PDT.Proc_Num := 4;                     { Anzahl der Prozeduren }
   PDT.Prg_Number := Programm_Number;     { Programm-Nummer }
   PDT.Version := Version;                { Programm-Version }
   PDT.Revision := Revision;              { Programm-Revision }
   PDT.CPU_Type := _186;                  { CPU-Typ }
   PDT.CoProz_Type := _NOCOPROZ;          { Co-Prozessor-Typ }
   PDT.Language := _TP70;                 { Programmiersprache }
   PDT.Prg_Type := _User_Prg;             { Programm-Typ: Anwenderprogramm }
   PDT.Flags := $0;                       { Flags }
   PDT.Flags := _DI_Task + _PDT_Info_enable + _Local_Parameter;
   PDT.Interrupt_Num := IRQ_TIMER_A;      { Interrupt-Nummer (Timer A) }
   PDT.Data_Adr := 0;                     { Adresse des Datenbereichs }
   PDT.Data_Num := 0;                     { Größe des Datenbereichs }
   PDT.Data_Min := 0;                     { Minimaler Datenbereich }
   PDT.Data_Max := 0;                     { Maximaler Datenbereich }
                                          { Anfangsadr. des Parameterbereichs}
   ml2rt_phys_adr (@Parameter, PDT.Parameter_Adr);
   PDT.Parameter_Adr := PDT.Parameter_Adr + SizeOf(TDT_Type);
                                          { Anzahl der Parameter }
   PDT.Parameter_Num := SizeOf(Parameter) - SizeOf(TDT_Type);
   PDT.Hyper_Adr := 0;                    { Adresse des Hypertextes }
   PDT.Res1 := 0;
   PDT.Res2 := 0;
   PDT.Main := @Main_Proc;                { Adresse der Hauptprozedur }
   PDT.Auto_Init := @Auto_Init;           { Adresse der Auto-Init-Prozedur }
   PDT.Start := @Start;                   { Adresse der Start-Prozedur }
   PDT.Stop := @Stop;                     { Adresse der Stop-Prozedur }
end;


begin                                     { ***** PREPARE ***** }
   Init_PDT;                              { PDT initialisieren }
   ml2rt_Set_PDT_Adr (PDT);               { Adr. der PDT fuer Betriebssystem }
end.
```

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

Die Start-Prozedur berechnet zuerst aus der angegebenen Frequenz die Timerwerte. Beachten Sie bitte dabei, daß die Timer mit einer Eingangsfrequenz von 500 kHz getaktet werden und daß Timer-A nur 12 Bit breit ist. Da mit diesen Werten "nur" Blinkfrequenzen zwischen 250 kHz und 122 Hz möglich sind, wurde der Hardware-Timer durch einen Software-Zähler erweitert. Erst wenn dieser Software-Zähler auf Null gelaufen ist, wird die LED umgeschaltet. Bei der Programmierung wurde darauf

geachtet, daß der Timerwert möglichst groß ist, damit die Anzahl der Interrupts vom Timer pro Zeiteinheit möglichst klein ist.

Die Hauptprozedur unterscheidet sich ein wenig von der Hauptprozedur der NI-Task. Es handelt sich praktisch um eine Interrupt-Service-Routine. Am Ende der Prozedur wird ein EOI zum Interrupt-Controller gesendet und anschließend die Prozedur mit "ml2rt_exit_interrupt" beendet.

! *Die Hauptroutine darf nicht als **'interrupt'**-Prozedur deklariert werden. Das durch die **'interrupt'**-Anweisung bewirkte Retten der Register wird mit **ml2rt_entry** erledigt, das Beenden der Prozedur mit IRET mit **ml2rt_exit_interrupt**.*

7.6.5.4. Installieren der DI-Task

Nachdem das Programm "M2P0301.PAS" einwandfrei übersetzt und die Datei "M2P0301.EXE" erzeugt wurde, kann das Programm auf die Karte geladen und gestartet werden. Geben Sie dazu bitte folgende Installationsdatei mit Hilfe von MLC ein und laden Sie diese Datei. Sie können auch die Beispielinstallationsdatei "DI-LED.INS" benutzen.

```
' Beispielinstallation: LED-Blinken mit Timer-Interrupt (Timer-A)
'
' Karte anwählen und Reset auslösen. Dabei wird gleichzeitig das angegebene
' Betriebssystem auf die Karte geladen.
M2DEVICE 0380 TIMEOUT=30 RESET \SORCUS\ML2\OSX\ML2-1D.10R
'
'
' Programm-Nummer 301h unter der Task-Nummer 2 installieren
' (Interrupt 83h => TIMER-A)
M2INST M2P0301.EXE 0301 0002 83 00000 0982
'
' Parameter der Task 1 setzen
' PAR-0   : Status des Programms
' PAR-1,2 : Blinkrate der LED    => 5 Hz
' PAR-3   : Status der LED       => 0 = Aus
M2PAR 02 01 0005 00
'
'
' Durch Aufruf der Start-Prozedur (Proz. 2) das Programm starten
M2PROC 02 02
```

Die Installationsdatei installiert die DI-Task und aktiviert sie anschließend.

Beachten Sie bitte, daß dieses Programm im Gegensatz zur NI-Task nur einmal (!) installiert werden kann, da es auch nur einen Timer-A auf der Basiskarte gibt. Sie könnten als Übung ja versuchen, das vorhandene Programm unter einem anderen Namen zu kopieren und es so zu modifizieren, daß das neue Programm z.B. den Timer-B benutzt. Denken Sie daran, daß Timer-B im Gegensatz zu Timer-A 16 Bit breit ist und somit auch ohne Software-Teiler einen größeren Blink-Frequenz-Bereich abdecken kann.

Wenn Sie ausschließlich mit PASCAL programmieren, lesen Sie bitte ab Seite 7-39 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-LAB/2 ablegen können. Zum Beispiel C:\SORCUS\ML2\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-LAB/2 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:\BC\LIB

Pfad zu Ihren MODULAR-4/486 Start-Up-Codes:
C:\SORCUS\ML2\RT\BC\BC45

Der Eintrag in "Library Directories" müßte dann wie folgt aussehen:
C:\SORCUS\ML2\RT\BC\BC45;C:\BC\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 MultiLAB/2 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 zwischen zwei verschiedenen Start-Up-Codes, die Sie in den Verzeichnissen 'BC31' bzw. 'BC31NOFP' finden. Letzteren sollten Sie verwenden, wenn Ihre Programme keine Floating-Point-Operationen enthalten; der erzeugte Code wird dann kleiner.

Falls beim Linken eines Echtzeitprogrammes die falschen Start-Up-Codes eingebunden werden, meldet der Linker 'undefined symbol **_wrong_startups_linked**'. Im umgekehrten Fall, also beim Linken der SORCUS Start-Up-Codes zu einem PC-Programm, lautet die Meldung in der Regel 'undefined symbol **SORCSPINIT** in Module xx'.

7.7.3. Programmierung

Nach der Einbindung der neuen Start-Up-Codes können Sie Programme mit C++ für die Multi-LAB/2 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.* ***Pointeroperationen 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 Programmes.*
- *Benutzen Sie das Speichermodell "LARGE".*
- *Bei Floating-Point-Operationen darf der Stack innerhalb der Prozedur nicht verändert worden sein. Durch* ***ml2rt_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.*
- *Keine Registervariablen in Task-Funktionen.*
- *Keine Floating-Point-Operationen in DI-Tasks.*
- *Keine Floating-Point-Operationen in Prozeduren, die vom PC aus aufgerufen werden.*

7.7.4. Compilereinstellungen

Sie sollten die Compiler- und Linker-Optionen wie folgt einstellen:

Borland C 3.1:

Options:

- Compiler/Advanced-Code-Generation: Emulation
  80186-Code
  Generate Underbars
- Compiler/Entry-, Exit-Code: DOS standard
  Standard stack frame
- Compiler/Optimize: Optimize for Speed
- Compiler/Source-Options: BORLAND C++
- Linker/Options: keine zusätzlichen Libraries einbinden

Borland C 4.5:

TargetExpert:

- Zieltyp: Anwendung (.exe)
- Umgebung: DOS (Standard)
- Zielmodell: Large
- Standardbibliotheken: Laufzeit
  'Emulation' oder 'keine Mathe-Unterstützung'

Projektoptionen:

- Compiler / Compiler-Ausgabe: Unterstriche erzeugen
- 16-Bit Compiler / Prozessor: 80186
- 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 "M2P0300.C" auf der mitgelieferten Diskette. Zum Compilieren benutzen Sie bitte die ebenfalls auf der Diskette vorhandene Projektdatei "M2P0300.PRJ" (bzw. M2P0300.IDE). Bitte überprüfen Sie vor dem Compilieren die eingestellten Pfade ("Options/Directories").

```
#include "ml2rtbib.h"        /* Header für die Echtzeitbibliothek einbinden */
#include "dos.h"
#include "stdlib.h"

                             /* ---- Allgemeine Angaben ---- */
#define PROGRAMM_NUMBER 0x300            /* Programm-Nummer = 300h */
#define VERSION         '1'              /* Version des Programmes */
#define REVISION        'A'              /* Revision des Programmes */

                             /* ---- Die verschiedenen Statusmöglichkeiten des Programmes ---- */
#define READY           0                /* Bereit */
#define RUNNING         2                /* Programm läuft */
#define ERROR           3                /* Falsche LED angewählt */
#define STOPPED         4                /* Programmlauf angehalten */

                             /* ---- Zustände der LED ---- */
#define  ON             1
#define  OFF            0
                        /* ---- Aufbau der Programm-Deskriptor-Tabelle (PDT) ---- */
struct pdt_type
{
   unsigned char         typ;                        /* Typ der PDT */
   unsigned char         lengt;                      /* Länge des Vorspanns der PDT */
   unsigned int          proc_num;                   /* Anzahl der Prozeduren */
   unsigned int          prg_number;                 /* Nummer des Programmes */
   unsigned char         version;                    /* Version des Programmes */
   unsigned char         revision;                   /* Revision des Programmes */

   unsigned char         cpu_type;                   /* Prozessor-Typ */
   unsigned char         coproz_type;                /* Co-Prozessor-Typ */
   unsigned char         language;                   /* Programmiersprache */
   unsigned char         prg_type;                   /* Programm-Typ */

   unsigned int          flags;                      /* Installierungsflags */
   unsigned int          interrupt_num;              /* Nummer des benutzten Interrupts */

   unsigned long         data_adr;                   /* Anfangsadresse des Datenbereichs */
   unsigned long         data_size;                  /* Größe des Datenbereichs */
   unsigned long         data_min;                   /* Datenbereich, den das Programm mindestens
                                                          benötigt */
   unsigned long         data_max;                   /* Maximaler Datenbereich, den das
                                                        Programm verarbeiten kann */

   unsigned long         parameter_adr;              /* Adresse des Parameterbereichs */
   unsigned int          parameter_size;             /* Anzahl der Parameter */

   unsigned long         hyper_adr;                  /* Anfangsadresse des Hypertextbereichs */
   unsigned int          res1;
   unsigned long         res2;

   void                  (*main_proc)();             /* Adresse der Haupt-Prozedur (Task) */
   void                  (*auto_init)();             /* Adresse der Auto-Init Prozedur */
   void                  (*start)();                 /* Adresse der Start-Prozedur */
   void                  (*stop)();                  /* Adresse der Stop-Prozedur */
} pdt;
                                                     /* ---- Aufbau des Parameterbereichs ---- */
struct parameter_type
{
   tdt_type              tdt;                        /* Platz für die Task-Deskriptor-Tabelle */
   unsigned char         status;                     /* Status des Programmes (READY etc.) */
   unsigned int          blink_rate;                 /* Blinkrate */
   unsigned char         led_status;                 /* Zustand der LED */
} parameter;
```

```
unsigned int             pause;           /* Ein Zähler, der die Durchläufe des Programmes zählt */

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

void auto_init(void)         /* ---- AUTO-INITIALISIERUNG ---- */
{
   ml2rt_entry();                         /* Register retten und vorbereiten */
   parameter.status = READY;              /* Status des Programmes auf "BEREIT" */
   parameter.blink_rate = 10000;          /* Blinkrate auf einen Defaultwert einstellen */
   parameter.led_status = OFF;            /* Status der LED auf "AUS" */
   ml2rt_led_off();                       /* Die LED ausschalten */
   pause = 0;                             /* Durchlaufzähler auf Null */
   ml2rt_exit();                          /* Register wieder restaurieren */
}

void start(void)             /* ---- Start-Prozedur ---- */
{
   ml2rt_entry();
   parameter.status = RUNNING;            /* Status auf "Programm läuft" */
   ml2rt_wakeup_task (parameter.tdt.task);
   ml2rt_exit();
}

void stop(void)              /* ---- Stop-Prozedur ---- */
{
    ml2rt_entry();                        /* Programm deaktivieren (stoppen) */
    ml2rt_sleep_task (parameter.tdt.task);
    parameter.status = STOPPED;
    ml2rt_exit();
}

void main_task()             /* ---- HAUPTPROZEDUR DER TASK ---- */
{
   ml2rt_entry();
   if (++pause >= parameter.blink_rate)
   {                                      /* Falls die LED eingeschaltet ist, ... */
      if (parameter.led_status == ON)
      {                                   /* ... LED ausschalten und ... */
 ml2rt_led_off();
 parameter.led_status = OFF;              /* ... Zustand merken */
      }
      else                                /* Falls die LED ausgeschaltet ist, ... */
      {
 ml2rt_led_on();                          /* ... LED einschalten und ... */
 parameter.led_status = ON;               /* ... Zustand merken */
      }
      pause = 0;                          /* Durchlaufzähler wieder auf Null setzen */
   }
   ml2rt_exit();
}
```

```
void init_pdt()              /* ---- PDT-INITIALISIEREN ---- */
{
   pdt.typ = 1;                           /* PDT-Typ einstellen (im Moment wird Typ 1 unterstützt) */
   pdt.lengt = _PDT_LENGTH;               /* Länge des PDT-Vorspanns (im Moment immer = 48) */
   pdt.proc_num = 4;                      /* Anzahl der Prozeduren */
   pdt.prg_number = PROGRAMM_NUMBER;      /* Programm-Nummer */
   pdt.version = VERSION;                 /* Programm-Version */
   pdt.revision = REVISION;               /* Programm-Revision */
   pdt.cpu_type = _186;                   /* CPU-Typ */
   pdt.coproz_type = _NOCOPROZ;           /* Co-Prozessor-Typ */
   pdt.language = _CPP31;                 /* Programmiersprache = C++ */
   pdt.prg_type = _USER_PRG;              /* Programm-Typ: Anwenderprogramm */
   pdt.flags = _NI_TASK + _PDT_INFO_ENABLE + _LOCAL_DATA + _LOCAL_PARAMETER + _FIXED_DATASIZE;
   pdt.interrupt_num = 0;                 /* Int.-Nr für die das Prg. gedacht ist
                                             (nicht relevant, da NI-Task)*/
   pdt.data_adr = 0;                      /* Adr. des Datenbereichs (hier wird kein Datenbereich
                                              verwendet) */
   pdt.data_size = 0;                     /* Größe des Datenbereichs */
   pdt.data_min = 0;                      /* Minimale Datenbereichsgröße */
   pdt.data_max = 0;                      /* Maximale Datenbereichsgröße */
                                          /* Anfangsadresse des Parameterbereichs errechnen */
   ml2rt_phys_adr (&parameter, &pdt.parameter_adr);
   pdt.parameter_adr = pdt.parameter_adr + sizeof(tdt_type);
                                          /* Anzahl der Parameter eintragen */
   pdt.parameter_size = sizeof(parameter) - sizeof(tdt_type);
   pdt.hyper_adr = 0;                     /* Adresse des Hypertextes (wird nicht verwendet) */
   pdt.main_proc = main_task;             /* Adresse der Hauptprozedur */
   pdt.auto_init = auto_init;             /* Adresse der Auto_Init Prozedur */
   pdt.start = start;                     /* Adresse der Start-Prozedur */
   pdt.stop = stop;                       /* Adresse der Stop-Prozedur */
}

 void main ()                 /* ---- PREPARE ---- */
{
   init_pdt();                             /* PDT initialisieren */
   ml2rt_set_pdt_adr (&pdt);               /* Adresse der PDT im Rückgabe-Record dem Betriebssystem
mitteilen */
}
```

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

Um dieses Programm zu compilieren, öffnen Sie bitte die Projektdatei "M2P0300.PRJ" (bzw. M2P0300.IDE). Beachten Sie bitte, daß die Datei "ML2RTBIB.OBJ" in Ihre Projektdatei eingebunden werden muß. Nähere Informationen zu dieser Bibliothek finden Sie im Kapitel 9 dieses Handbuchs, wo alle Routinen der Bibliothek "ML2RTBIB" ausführlich erläutert sind.

Zuerst wird der Aufbau der PDT als Struktur deklariert. Das Programm enthält vier Prozeduren (Main-Proc, Auto-Init, Start und Stop).

Dann wird eine Struktur für die Parameter vereinbart. Der Speicher für die Parameter wird vom Programm bereitgestellt. Bei der Deklaration ist es wichtig, daß vor den eigentlichen Parametern Platz für die TDT reserviert wird.

Alle globalen (also in der PDT aufgeführten) Prozeduren beginnen mit **ml2rt_entry** und enden mit **ml2rt_exit**. Dadurch ist sichergestellt, daß alle Prozessor-Register gesichert und für die Prozedur richtig eingestellt werden.

In der Auto-Init-Prozedur werden die Parameter initialisiert und die LED ausgeschaltet. In der Prozedur "Init_PDT" wird die PDT initialisiert. Beachten Sie bitte die Einstellung der PDT-Flags.

Die Prozedur **main** enthält in einem "normalen" C-Programm das eigentliche Hauptprogramm. Dieser Teil, der hier als "Prepare" bezeichnet wird, wird während der Installierung abgearbeitet. Er sorgt dafür, daß die PDT initialisiert wird (Init_PDT) und dem Multi-LAB/2 Betriebssystem die Adresse der PDT mitgeteilt wird (ML2_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 "M2P0300.PRJ" einwandfrei übersetzt und die Datei "M2P0300.EXE" erzeugt wurde, kann das Programm auf die Karte geladen und gestartet werden. Geben Sie dazu bitte folgende Installationsdatei mit Hilfe von MLC ein, und laden Sie diese Datei. Sie können auch die Beispielinstallationsdatei "NILED.INS" benutzen.

```
; Beispielinstallation: LED-Blinken als NI-Task (ohne Interrupt)
;
;
; Karte 0 anwählen, Adresse einstellen, Reset auslösen und
; Betriebssystem laden
M2DEVICE 0 0380 TIMEOUT=10 RESET ml2-??.??R
;
;
; Programm-Nummer 300h unter Tasknummer 1 installieren (nicht aktivieren)
M2INST M2P0300.EXE 0300 0001 01 000000 00000980
;
; Parameter der Task 1 setzen
; PAR-1,2 : Blinkrate der LED    => 1000h
; PAR-3   : Status der LED       => 0 = Aus
;
M2PAR 01 01 00 10 00
;
;
; Durch Aufruf der Start-Prozedur Programm starten (Proz. 2)
M2PROC 01 02
```

Diese Installationsdatei installiert die NI-Task, stellt die Blinkrate ein und aktiviert die Task durch Aufruf der Prozedur 2. Beachten Sie, daß diese Art der Angabe des Betriebssystems (ml2-??-??R) erst ab der MLC-Version 1.D unterstützt wird. Falls Sie ein älteres MLC-Programm benutzen, müssen Sie den korrekten Betriebssystemnamen angeben (oder ein Update bei SORCUS Computer bestellen).

7.7.5.3. DI-Task in C++

Das folgende Listing zeigt das Blink-LED-Programm als DI-Task. Sie finden das Programm auch unter dem Namen "M2P0301.C" auf der mitgelieferten Diskette. Zum Compilieren benutzen Sie bitte die Projektdatei "M2P0301.PRJ" (bzw. M2P0301.IDE), die sich ebenfalls auf der Diskette befindet. Das Programm soll unter dem Interrupt von Timer-A installiert werden. Die Blinkrate soll in Hertz angegeben werden.

```
#pragma option -1            /* Code für 80186 erzwingen! */
#pragma option -r-           /* Keine Registervariablen verwenden!! */

#include "ml2rtbib.h"        /* Echtzeitbibliothek einbinden */
#include "ml2makro.h"        /* Die Makro-Datei benötigt den Turbo-Assembler*/
#include "stdlib.h"
#include "dos.h"

/* ---- Allgemeine Angaben ---- */
#define PROGRAMM_NUMBER 0x301          /* Programmnummer = 301h */
#define VERSION         '1'            /* Version des Programms */
#define REVISION        'B'            /* Revision des Programms */

/* ---- Die verschiedenen Statusmöglichkeiten des Programms ---- */
#define READY           0              /* Bereit */
#define RUNNING         2              /* Programm läuft */
#define ERROR           3              /* Falsche LED angewählt */
#define STOPPED         4              /* Programm wurde angehalten */

/* ---- Zustände der LED ---- */
#define  ON             1
#define  OFF            0

/* ----- Timer-Initialisierungen ------ */
/* Timer A 12-bit-> (4096 / 2), da Periodendauer T = t(LED_ON) + t(LED_OFF) */
#define TIMER_VAL  2000UL
/* Timer A Basistakt 500kHz */
#define TIMER_FREQ 500000UL

/* ---- Aufbau der Programm-Descriptor-Tabelle (PDT) ---- */
struct pdt_type
{
   unsigned char  typ;                 /* Typ der PDT */
   unsigned char  lengt;               /* Länge des Vorspanns PDT */
   unsigned short proc_num;            /* Anzahl der Prozeduren */
   unsigned short prg_number;          /* Nummer des Programms */
   unsigned char  version;             /* Version des Programms */
   unsigned char  revision;            /* Revision des Programms */

   unsigned char  cpu_type;            /* Prozessor-Typ */
   unsigned char  coproz_type;         /* Co-Prozessor-Typ */
   unsigned char  language;            /* Programmiersprache */
   unsigned char  prg_type;            /* Programm-Typ */

   unsigned short flags;               /* Installierungsflags */
   unsigned short interrupt_num;       /* Nummer des benutzten Interrupts */

   unsigned long  data_adr;            /* Anfangsadresse des Datenbereichs */
   unsigned long  data_size;           /* Größe des Datenbereichs */
   unsigned long  data_min;            /* Minimaler Datenbereich f. Programm*/
   unsigned long  data_max;            /* Maximaler Datenbereich f. Programm*/

   unsigned long  parameter_adr;       /* Adresse des Parameterbereichs */
   unsigned short parameter_size;      /* Anzahl der Parameter */

   unsigned long  hyper_adr;           /* Anfangsadresse Hypertext-Bereich */
   unsigned short res1;
   unsigned long  res2;

   void (*main_proc)();                /* Adresse der Haupt-Prozedur */
   void (*auto_init)();                /* Adresse der Auto-Init Prozedur */
   void (*start_proc)();               /* Adresse der Start-Prozedur */
   void (*stop_proc)();                /* Adresse der Stop-Prozedur */
} pdt;
```

```
/* ---- Aufbau des Parameterbereichs ---- */
struct parameter_type
{
   tdt_type              tdt;          /* Platz für Task-Descriptor-Tabelle */
   unsigned char         status;       /* Status des Programms (READY etc.) */
   unsigned short        blink_rate;   /* Blinkrate in Hertz */
   unsigned char         led_status;   /* Zustand der LED */
   unsigned char         soft_cnt;     /* Soft-Counter noetig, um mit Timer */
                                       /* auch niedrige Freq. zu erzeugen!  */
   unsigned char         soft_init;    /* Soft-Counter Initialisierungswert */
} parameter;

/* Prototypen der benutzten Prozeduren */
void auto_init(void);
void main_proc(void);
void start(void);
void stop(void);
void init_pdt(void);

/* ---- AUTO-INITIALISIERUNG ---- */
void auto_init()
{
   ml2rt_entry();                      /* Register retten und vorbereiten */
   parameter.status = READY;           /* Status des Programms auf "BEREIT" */
   parameter.blink_rate = 1;           /* Blinkrate defaultmaessig auf 1 Hz */
   parameter.led_status = OFF;         /* Status der LED auf "AUS" */
   ml2rt_led_off();                    /* LED ausschalten */
   ml2rt_exit();                       /* Register wieder restaurieren */
}


void start()
{
   unsigned short timer;

   ml2rt_entry();

   /* Softteiler ausrechnen (muss ganzzahlig sein, keine Fliesskommaop.!):  */
   parameter.soft_init = TIMER_FREQ / (2 * parameter.blink_rate * TIMER_VAL);

   /* falls Softzaehler auf 0 "abgerundet" wird, setze Zaehler auf 1 */
   if (parameter.soft_cnt == 0)
      parameter.soft_cnt = 1;
   parameter.soft_cnt = parameter.soft_init;

   /* Timerwert ausrechnen: Fehler durch Softteiler (ganzzahlig!) wird durch
   Anpassung des maximalen Timerwerts kompensiert! */
   timer = TIMER_FREQ / (2 * parameter.blink_rate * parameter.soft_init);

   /* Timer auf den berechneten Wert setzen und starten */
   ml2rt_set_timer (TIMER_A, timer, TIMER_MODE);
   parameter.status = RUNNING;
   ml2rt_unmask_int (IRQ_TIMER_A);     /* Timer-Interrupt freigeben */
   ml2rt_exit();
}


void stop()
{
   ml2rt_entry();
   ml2rt_mask_int (IRQ_TIMER_A);       /* Timer-Interrupt sperren */
   parameter.status = STOPPED;         /* Status auf "abgebrochen" */
   ml2rt_exit();
}


/* ---- HAUPTPROZEDUR DER TASK ---- */
void main_proc()
{
   ml2rt_entry();

   parameter.soft_cnt--;               /* Soft-Counter dekrementieren */
   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, */
      {
         ml2rt_led_off();              /* ... LED ausschalten und ... */
         parameter.led_status = OFF;   /* ... Zustand merken */
      }
      else                             /* Falls die LED ausgeschaltet ist, */
      {
         ml2rt_led_on();               /* ... LED einschalten und ... */
```

```
         parameter.led_status = ON;    /* ... Zustand merken */
      }                                /* Software-Teiler ruecksetzen */
      parameter.soft_cnt = parameter.soft_init;
   }

   ml2rt_end_of_int(IRQ_TIMER_A);       /* EOI an den Interrupt-Contr. */
   ml2rt_exit_interrupt();              /* ISR verlassen */
}

/* ---- PDT-INITIALISIEREN ---- */
void init_pdt(void)
{
   pdt.typ = 1;                         /* PDT-Typ einstellen, Standard=1 */
   pdt.lengt = _PDT_LENGTH;             /* Länge des PDT-Vorspanns */
   pdt.proc_num = 4;                    /* Anzahl der Prozeduren */
   pdt.prg_number = PROGRAMM_NUMBER;    /* Programm-Nummer */
   pdt.version = VERSION;               /* Programm-Version */
   pdt.revision = REVISION;             /* Programm-Revision */
   pdt.cpu_type = _186;                 /* CPU-Type */
   pdt.coproz_type = _NOCOPROZ;         /* Co-Prozessor-Typ */
   pdt.language = _CPP31;               /* Programmiersprache = C++ */
   pdt.prg_type = _USER_PRG;            /* Programm-Typ: Anwenderprogramm */
   pdt.flags = 0;                       /* Flags */
   pdt.flags = _DI_TASK + _PDT_INFO_ENABLE + _LOCAL_PARAMETER;
   pdt.interrupt_num = IRQ_TIMER_A;     /* Interrupt-Nummer für Programm */
   pdt.data_adr = 0;                    /* Adresse des Datenbereichs (hier 0)*/
   pdt.data_size = 0;                   /* Größe des Datenbereichs */
   pdt.data_min = 0;                    /* Minimale Datenbereichsgröße */
   pdt.data_max = 0;                    /* Maximale Datenbereichsgröße */
   /* Anfangsadresse des Parameterbereichs errechnen */
   ml2rt_phys_adr (&parameter, &pdt.parameter_adr);
   pdt.parameter_adr = pdt.parameter_adr + sizeof(tdt_type);
   /* Anzahl der Parameter eintragen */
   pdt.parameter_size = sizeof(parameter) - sizeof(tdt_type);
   pdt.hyper_adr = 0;                   /* Adresse des Hypertextes */
   pdt.main_proc = main_proc;           /* 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 */
}

/* ---- PREPARE ---- */

void main ()
{
   init_pdt();                             /* PDT initialisieren */
   ml2rt_set_pdt_adr (&pdt);               /* Adresse der PDT fuer System */
}
```

Da das Listing ausführlich dokumentiert ist, soll nur auf die Besonderheiten eingegangen werden. Zum Compilieren benutzen Sie bitte die Projektdatei "M2P0301.PRJ" (bzw. M2P0301.IDE).

In der PDT werden Sie erkennen, daß das Programm vier Prozeduren enthält (Main-, Auto-Init-, Start- und Stop-Prozedur). Die "Start-Prozedur" berechnet zuerst aus der angegebenen Frequenz die Timerwerte. Beachten Sie bitte dabei, daß die Timer mit einer Quarzfrequenz von 500 kHz getaktet werden und daß Timer-A 12 Bit breit ist. Da bei diesen Werten "nur" eine Blinkfrequenz zwischen 250 kHz und 122 Hz möglich ist, wurde der Hardware-Timer durch einen Software-Zähler erweitert. Erst wenn dieser Software-Zähler auf Null gelaufen ist, wird die LED umgeschaltet. Bei der Programmierung wurde darauf geachtet, daß der Timerwert möglichst groß ist, damit die Anzahl der Interrupts pro Zeiteinheit möglichst klein ist.

Die Hauptprozedur unterscheidet sich ein wenig von der Hauptprozedur der NI-Task. Es handelt sich praktisch um eine Interrupt-Service Routine. Am Ende wird deshalb dem Interrupt-Controller die Abarbeitung der Interrupt-Prozedur angezeigt (EOI = End of Interrupt). Anschließend wird die Prozedur mit **ml2rt_exit_interrupt** beendet.

7.7.5.4. Installierung der DI-Task

Nachdem das Projekt "M2P0301.PRJ" einwandfrei übersetzt und die Datei "M2P0301.EXE" erzeugt wurde, kann das Programm auf die Karte geladen und gestartet werden. Geben Sie dazu bitte folgende Installationsdatei mit Hilfe von MLC ein, und laden Sie diese *.INS Datei. Sie können auch die Beispielinstallationsdatei "CDILED.INS" benutzen.

```
; Beispielinstallation: LED-Blinken mit Timer-Interrupt (Timer-A)
;
; Karte 0 anwählen, Adresse einstellen, Reset auslösen und
; Betriebssystem laden
M2DEVICE 0 0380 TIMEOUT=10 RESET ML2-??.??R
;
;
; Programm-Nummer 301h unter der Tasknummer 2 installieren
M2INST M2P0301.EXE 0301 0002 83 000000 0982
;
; Parameter der Task 2 setzen
; PAR-0   : Status des Programmes
; PAR-1,2 : Blinkrate der LED    => 5 Hz
; PAR-3   : Status der LED       => 0 = Aus
;
M2PAR 02 01 05 00 00
;
;
; Durch Aufruf der Start-Prozedur (Proz. 2) das Programm starten
M2PROC 02 02
```

Diese Installationsdatei installiert die DI-Task und aktiviert sie anschließend.

Sie könnten als Übung ja einmal versuchen, das vorhandene Programm unter einem anderen Namen zu kopieren und es so zu modifizieren, daß dieses neue Programm z.B. den Timer-B benutzt. Beachten Sie dabei, daß Timer-B im Gegensatz zu Timer-A 16 Bit breit ist und somit auch ohne Software-Teiler einen größeren Bereich abdecken kann.

7.7.6. Die Makrobibliothek "ML2MAKRO.H"

Bislang wurden die Funktionen aus der Bibliothek "ML2RTBIB" ganz normal aufgerufen. Für die wichtigsten Funktionen (wie z.B. **ml2rt_entry**, **ml2rt_exit**) gibt es jedoch einen Weg, diese Methode zu optimieren. Wenn Sie die Header-Datei "ML2MAKRO" einbinden, wird der entsprechende Programmcode der Subroutine direkt an der entsprechenden Programmestelle 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 "ML2RTBIB" einbinden und anschließend Ihr Programm nochmals compilieren. Vorher ist im Menü "Options/Transfer" der Pfad zum Turbo-Assembler einzugeben (wird normalerweise defaultmäßig während der Installierung von Borland C++ eingestellt).

7.8. Allgemeine Hinweise zur Programmierung

7.8.1. FAR-Kompilierung 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

Die Routinen innerhalb der ML2RTBIB 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 Byte, 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 Sie trotz allem nicht immer um eine direkte (Assembler-) Programmierung herumkommen. Meistens reicht es aber in solchen Fällen aus, nur eine bestimmte, sehr zeitkritische Routine zu optimieren. Vielleicht 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-LAB/2 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 wie bei DOS in einer Schleife geschehen. In diesem Fall würde nicht nur die eine Task, sondern auch 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-LAB/2 Karte mit anderen Tasks ausgelastet ist, also letztlich davon, wie oft die NI-Task pro Zeiteinheit 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 (ml2rt_write_data_xxxx). Je nach Installierung der Task gibt es eine oder zwei Möglichkeiten, die Tasknummer zu bestimmen.

Wenn der Parameterbereich Teil des Programmes ist, dann ist zwingend auch die Task-Deskriptor-Tabelle (TDT) im Programm enthalten. Die Tasknummer steht als Eintrag in der TDT. Wenn Sie für die Reservierung der TDT die vordefinierte Struktur (TDT_Type, siehe Kapitel 9) aus ml2rtbib 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 auf die TDT in diesem Fall nur unter Angabe der Tasknummer zugegriffen werden kann. In diesem Fall verwenden Sie statt der Funktion **ml2rt_entry** die Funktion **ml2rt_entry_task**, die die Tasknummer als Funktionsergebnis zurückliefert.

7.8.6. Verwenden von Fließkommaoperationen

Die Verwendung von Fließkommaoperationen unterliegt bei der Erstellung von Echtzeitprogrammen zwei Einschränkungen.

Die Emulation des Co-Prozessors ist nicht reentrant. Das heißt, daß eine laufende Berechnung nicht von einer Funktion unterbrochen werden darf, die ebenfalls Fließkommaoperationen enthält. Es dürfen deshalb keine Fließkommaberechnungen in Interrupt-Prozeduren bzw. in Prozeduren, die vom PC aus aufgerufen werden, enthalten sein.

7.8.6.1. Hinweise für C-Programmierer

Bei der Programmierung mit C müssen folgende Dinge beachtet werden:

1. Die Datei MATH.H muß in die Quelldatei eingebunden werden:

   ```
   #include <math.h>
   ```

2. Bei der Initialisierung der PDT muß als Coprozessor '_87' eingestellt werden:

   ```
   pdt.coproz_type = _87;
   ```

3. Die Compiler-Optimierungen müssen ausgeschaltet werden (Menü 'Options/Compiler/Optimizations').
4. Die Emulation muß eingeschaltet sein (Menü 'Options/Compiler/Advanced Code Generation').
5. Der Standard-Stackframe muß erzeugt werden (Menü 'Options/Compiler/Entry-Exit Code').
6. Innerhalb von Taskfunktionen dürfen keine Fließkommaoperationen ausgeführt werden. Unter 'Taskfunktion' werden sowohl Prozeduren (ohne Übergabeparameter) als auch Funktionen (mit Übergabeparametern) verstanden, die in der PDT eingetragen sind und daher mit **ml2rt_entry** beginnen und mit **ml2rt_exit** enden.

!
- *Auch einfache Zuweisungen wie '`variable = 0.56`' sind Fließkommaoperationen und dürfen nicht in Taskfunktionen ausgeführt werden.*
- *Die Auto-Init-Prozedur ist eine Taskfunktion. Darin dürfen keine Fließkommaoperationen ausgeführt werden.*

1. Es dürfen keine Fließkommaoperationen in der Hauptprozedur einer DI-Task ausgeführt werden und auch nicht in Funktionen (Taskfunktionen *und* anderen Funktionen), die diese Prozedur aufruft.

   Die Hauptprozedur einer DI-Task ist eine Interrupt-Service-Routine. In einer Interrupt-Service-Routine dürfen prinzipiell keine Fließkommaoperationen ausgeführt werden.

2. In Taskfunktionen, die vom PC aus aufgerufen werden (mit **ml2_call_proc** oder **ml2_call_func**), sind ebenfalls keine Fließkommaoperationen erlaubt, da ein PC-Zugriff auf die Karte innerhalb einer Interrupt-Service-Routine abgearbeitet wird.

3. Benutzen mehrere NI-Tasks Fließkommaoperationen, so darf während der Installierung einer solchen Task keine andere ausgeführt werden.

   Es muß daher so vorgegangen werden, daß zunächst alle Tasks auf die Karte geladen werden, ohne diese zu starten (Installierungsflag 12 = 0). Erst wenn alle NI-Tasks auf die Karte geladen worden sind, dürfen sie mit **ml2_wakeup_task** gestartet werden.

7.8.6.2. Programmbeispiel in C

Im folgenden ist ein Ausschnitt aus einem C-Beispielprogramm abgedruckt:

```
#include <math.h>
#include "ml2rtbib.h"

....

/* Funktion, in der Fließkommaoperationen ausgeführt werden */
void doublecalc(void)
{
   double var1, var2, var3;

   var1 = 1.5;
   var2 = sin(var1);
   var3 = var1 * var2;
}

/* Taskfunktion einer NI-Task */
void MainProc(void)
{
   ml2rt_entry();
   ...
   doublecalc();
   ...
   ml2rt_exit();
}

...

void init_pdt()                          /* PDT initialisieren */
{
   ...
   pdt.cpu_type = _186;                  /* CPU-Type */
   pdt.coproz_type = _87;                /* Co-Prozessor-Typ */
   ...
   pdt.flags = _NI_TASK + _PDT_INFO_ENABLE + _LOCAL_PARAMETER;
   ...
}

void main ()                             /* PREPARE */
{
   init_pdt();                           /* PDT initialisieren */
   ml2rt_set_pdt_adr (&pdt);
}
```

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.