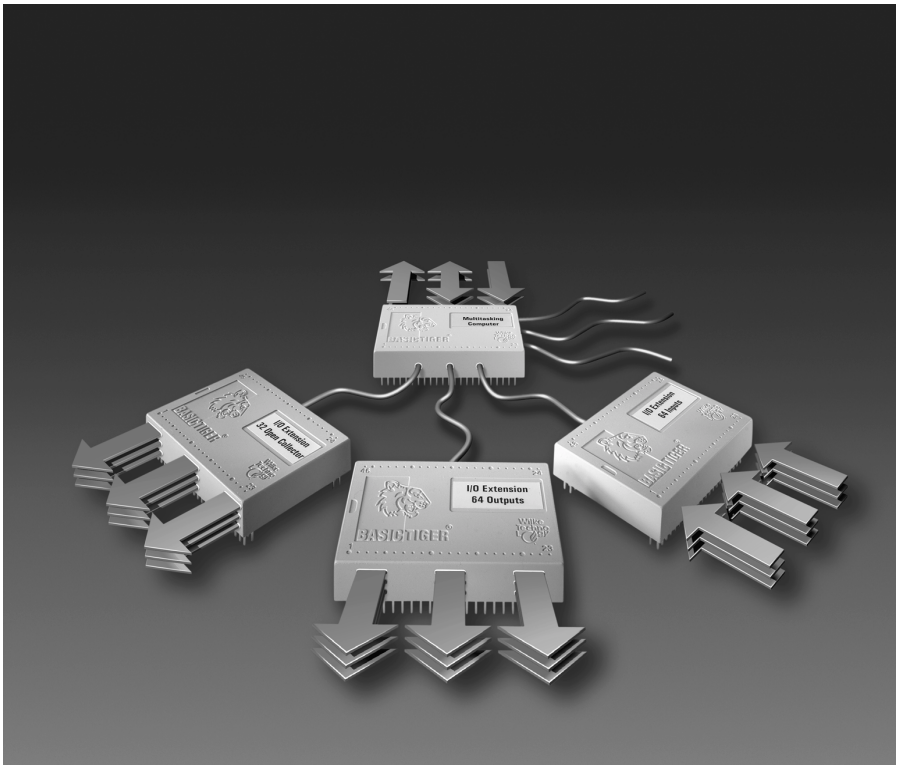


BASIC-TIGER®
Device-Treiber Handbuch

Leere Seite

Redaktion Klaus Hiltrop
Abbildungen Joji
Cover Werbeagentur Kordoni, Aachen
Druck Print Production, Aachen
5. Auflage - Version 5.0

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Inhaltsverzeichnis

Leere Seite

1 Zu diesem Buch

Tiger-BASIC® enthält neben den Instruktionen und Funktionen sogenannte Device-Treiber (Geräte-Treiber). Zu den Geräten zählen externe Einheiten wie LCD, Drucker, Encoder-Drehgeber, serielle Geräte. Geräte können sich aber auch im Modul befinden. So gibt es Device-Treiber für die Analogen Eingänge, die analoge Ausgabe mit Pulsweitenmodulation, Frequenzmessung über einen Eingangspin, Pulsausgabe und Pulslängenmessung, und viel mehr. Die Devicetreiber sind in einem eigenen Handbuch beschrieben. Dieses Handbuch enthält auch einige Applikation, die den Umfang eines Beispiels überschreiten. Nach der Installation finden Sie diese Applikationen im Verzeichnis APPLICAT.

Device-Treiber erweitern Tiger-BASIC® um Funktionen für die Ein- und Ausgabe, so daß externe Geräte komfortabel unterstützt werden. An Stelle zahlreicher neuer BASIC-Instruktionen für verschiedene I/O-Geräte zu verwenden, können alle Peripheriegeräte stets mit diesen 6 BASIC-Instruktionen gesteuert werden. In der Beschreibung der Device-Treiber ist festgelegt, welche Instruktionen der jeweilige Treiber unterstützt:

- PRINT
- PRINT_USING
- PUT
- INPUT
- INPUT_LINE
- GET

Diese Instruktionen arbeiten stets mit einem I/O-Device-Treiber. Der Device-Treiber kennt die gerätespezifischen Eigenschaften und stellt das Interface zum BASIC-Programm her. Unterschiedliche Geräte können ganz unterschiedlich auf eine PUT-Instruktion reagieren. Ein Robot-Arm verändert seine Position, ein Analog-Ausgang verändert einen Strom- oder Spannungswert, ein Modem erzeugt eine entsprechende Tonfolge. Die Unterschiedlichkeit der verschiedenen Geräte wird im Device-Treiber berücksichtigt, nicht in der BASIC-Syntax.

Einige Device-Treiber belegen bestimmte Pins oder Ports, die dann für IN- und OUT-Instruktionen gesperrt sind. Mehrere Device-Treiber können sich auch I/O-Pins teilen, z.B. in Form eines Busses.

Damit ein bestimmter Device-Treiber im Programm verfügbar ist, wird er zu Beginn der MAIN-Task installiert. Bei der Installation mit der Instruktion INSTALL_DEVICE wird dem Device-Treiber eine Zahl als Gerätenummer zugeordnet. Die oben genannten I/O-Instruktionen sprechen den Device-Treiber dann stets durch diese Gerätenummer an.

Tip

Die Gerätenummer in den I/O-Instruktionen kann auch eine Variable sein. Somit wird eine flexible I/O-Verwaltung ermöglicht. Die Datenströme können bei Bedarf leicht auf ander Geräte umgeleitet werden.

Ein- und Ausgabe-Instruktionen nutzen oft Sekundär-Adressen, Funktionscodes und Kommandos, um Geräte über die Treiber zu steuern.

Sekundär-Adressen wählen, ähnlich wie hausinterne Telefon-Durchwahlnummern, einen bestimmten Kanal eines Gerätes aus. Device-Treiber können, müssen aber nicht, Sekundär-Adressen unterstützen. Hat ein Device-Treiber z.B. 4 Kanäle, so wird durch die Sekundär-Adresse einer der 4 Kanäle spezifiziert. Device-Treiber können jedoch auch virtuelle Kanäle besitzen, die durch eine Sekundär-Adresse adressiert werden. In diese Kanäle können Steuer-Informationen geschrieben oder Status-Informationen gelesen werden. Die Funktionsweise ist abhängig von dem jeweiligen Device-Treiber. Einzelheiten sind in der Beschreibung des Device-Treibers geregelt.

Funktion-Codes können eingesetzt werden, um Steuerfunktionen auszuführen oder Status-Informationen vom Gerät zu erhalten. Die allgemeine Anwendung sowie einige allgemeine Funktion-Codes sind in diesem Kapitel beschrieben. Ob und welche Funktion-Codes unterstützt werden regelt die Beschreibung des Device-Treibers. Dort finden Sie auch die spezielleren Funktion-Codes beschrieben.

Neue Device-Treiber in der Version 5.0

Wenn Sie bereits erfahrener Tiger-BASIC®-Anwender sind, werden Sie sich sicher für Neuigkeiten interessieren. Sie finden hier eine kurze Übersicht über das Wichtigste, was auf dem Sektor Device-Treiber neu ist seit dem letzten Up-to-Date. Unter der nächsten Überschrift 'Was ist nicht ganz so neu?' finden Sie die Neuerungen des Vorletzten Up-to-Dates. Neue Instruktionen, Funktionen und Hardware-Neuerungen sind in den beiden anderen Handbüchern aufgeführt.

Hinweis: Der Treiber PLSO1 verwendet nun WORD-Variable für die Anzahl der Pulse. Bei bestehenden Projekten muß dies eventuell berücksichtigt werden.

Es sind weitere Device-Treiber hinzugekommen.

Einige der vorhandenen Device-Treiber haben erweiterete Funktionen bekommen:

- Der RTS0-Pin des SER1B_nn.TDD kann jetzt mit einem User-Function-Code gesetzt werden.
- Der Encoder erscheint dynamisch, wenn dem Device-Treiber eine Tabelle übergeben wird. Schnelles Drehen erzeugt dann mehr Schritte, als der Decoder wirklich erzeugt hat. Zusätzlich kann diese Geschwindigkeitsinformation ausgelesen werden.
- PLSO1 verwendet für die Anzahl der Pulse jetzt eine WORD-Variable (früher LONG).

1

Was ist nicht ganz so neu?

Die Device-Treiber, die in der Version 4 neu hinzugekommen sind:

Bis auf den Treiber TMEM.TDD arbeiten alle neuen Treiber mit dem TIMERA.TDD zusammen. Der Zeitbasis-Treiber TIMERA sorgt dafür, daß unabhängig vom BASIC im Hintergrund die angehängten Device-Treiber ihre Arbeit verrichten. Dabei können gleichzeitig mehrere Treiber an die Zeitbasis gekoppelt sein.

Einige der vorhandenen Device-Treiber haben erweiterete Funktionen bekommen:

- LCD1 ist mehrfach installierbar. Dazu können sowohl LCD als auch Tastaturscan einzel abgeschaltet werden.
- LCD1: es gibt jetzt ein Reset-Kommando zur Neu-Initialisierung.
- SER1 unterstützt nun RS-485-Betrieb, und das auch mit Adressen mit gesetztem 9. Bit.
- PWM2 akzeptiert neben Strings nun auch Adressen im Flash, so daß zur Sprachausgabe die Sprachdaten direkt aus dem Flash (ohne PEEK in String) ausgegeben werden können.
- Einige gepufferte Device-Treiber sind mit unterschiedlich großen Puffern vorhanden. Die Puffergrößenangabe steckt im Dateinamen.
- Die Ausgabe auf grafikfähige LC-Displays wird nun weitergehend unterstützt, allerdings durch Funktionen (Siehe Kapitel 'Grafik' im Programmierhandbuch). Außerdem ist ein Grafik-Toolkit in Vorbereitung, auf dem speziell Entwicklungen mit Grafik-LCD durchgeführt werden. Beachten Sie bitte auch die laufende Werbung sowie unsere Internetseiten (www.wilke.de).

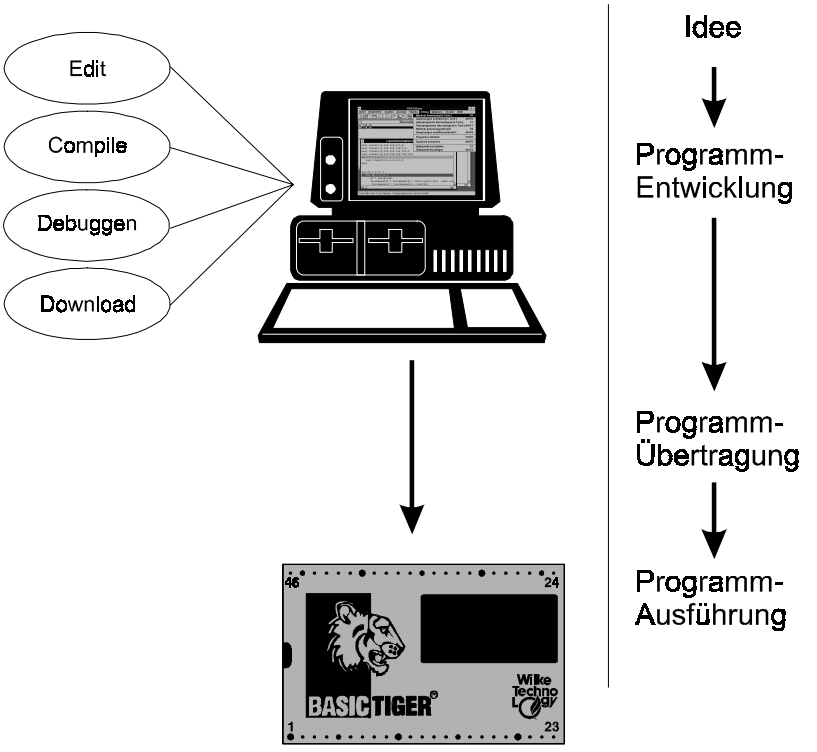
Typografische Konventionen und Symbole

Die folgenden Schriften und Symbole wurden verwendet, damit Sie wichtige Informationen schnell erkennen können:

Element	Bedeutung
TASTE	Tastenbezeichnungen, z.B. RETURN
`Programmlisting`	Tiger-BASIC-Programmlisting
Instruktion	Tiger-BASIC®-Instruktion
Variable	Platzhalter für Elemente, die Sie Ihrer Anwendung gemäß eingeben müssen.
[ ]	Elemente, die optional eingegeben werden können.
!	Wichtige Anmerkung: bitte beachten!
Tip	Tips und Hinweise, die Ihnen die Arbeit erleichtern.

1 *BASIC-Tiger®*

In diesem Kapitel erfahren Sie etwas über die Besonderheiten des BASIC-Tigers®.



Multitasking

Die auffälligste Besonderheit des BASIC-Tigers® ist seine Multitasking-Fähigkeit. Obwohl BASIC-Tiger®-Module nicht viel größer sind als ein CPU Chip, ist ein kompletter Multitasking Steuercomputer enthalten, der über eigenen Programmspeicher (FLASH), Arbeitsspeicher (SRAM + FLASH) sowie eine Reihe von Standard-I/Os verfügt. Mehrere Tiger-BASIC®-Programme (Tasks) können in den Programmspeicher der Tigers geladen werden und bleiben dort, ähnlich der Festplatte eines PC, dauerhaft gespeichert, bis sie durch neue Programme überschrieben werden. Der FLASH-Speicher läßt sich ferner auch als dauerhafter Speicher für Daten verwenden, der von BASIC-Programmen aus beschrieben, gelesen und gelöscht werden kann. Der Arbeitsspeicher kann bis zu mehreren MBytes SRAM betragen und läßt sich ebenfalls gegen Stromausfall sichern.

Der Vorteil von Multitasking wird sofort offenkundig, wenn man sich reale Aufgabenstellungen für einen Steuercomputer ansieht. Selten besteht eine Anwendung nur aus einer monolithischen, einzelnen Aufgabe, die linear in einer großen Schleife abgearbeitet wird. Schon bei kleinen Anwendungen gibt es meist 3, 4, 5 oder mehr eigenständige Aufgaben zu erledigen, die weitgehend unabhängig voneinander abzuarbeiten sind. Man denke nur an Ausgaben auf einen Drucker, Eingaben von Keyboards oder seriellen Eingängen etc., die Anwendungen oft „hängen“ lassen können. Um solche Zustände zu vermeiden ist oft einiger zusätzlicher Programmier- und Testaufwand erforderlich. Entsprechend schwerer verstehbar und pflegbar werden solche Programme.

Programmiert in Multitasking, bleibt nichts so leicht hängen. Eingaben, Ausgaben, abgeschlossene Steuerungen oder Auswertungen werden in eigenständigen Tasks abgearbeitet. Ist z.B. eine rechenintensive Auswertung noch nicht abgeschlossen, so werden dennoch erforderliche Steuersignale generiert, der Dialog mit einer Schnittstelle fortgeführt, Informationen auf Displays aufgefrischt und Bedienungstasten überwacht. Derartige Multitasking-Programme laufen nicht nur schneller und zuverlässiger ab, sie sind auch leichter zu pflegen und bieten mehr Übersicht. Zusätzliche Aufgaben können auch später noch leicht als neue Tasks hinzugefügt werden. Der Leistungsbedarf der einzelnen Aufgaben läßt sich durch Priorisierung der Tasks fein austarieren, Kontroll-Tasks können wichtige Funktionen im Auge behalten und ggf. Notprogramme starten und Alarme auslösen.

Die Programmierung in Multi-Tasking ist beim BASIC-Tiger® denkbar einfach und bereits mit wenigen BASIC-Zeilen zu realisieren. Beispiele finden Sie im Hardware/Installationshandbuch unter der Überschrift 'Blitzstart -Erste Schritte'.

1

Funktionen

Die ständig wachsende Bibliothek von Funktionen bildet ein mächtiges Tool, um Programmieraufgaben effektiv und mit wenigen Instruktionen zu realisieren. BASIC-Tiger®-Funktionen decken die Bereiche

- Integer Arithmetik
- Floating-Point Arithmetik
- String-Operationen
- Spezial-Funktionen

ab. Immer wiederkehrende Programmieraufgaben können auf fertige Funktionen zurückgreifen, die hohe Verarbeitungsgeschwindigkeit und kompakten Code erreichen.

Die 32-Bit-Integer-Arithmetik des BASIC-Tigers zeichnet sich durch hohe Geschwindigkeit und eine für viele Anwendungen bei weitem ausreichende Genauigkeit aus. Bei entsprechender Abbildung kann mit dem verfügbaren Wertebereich von -2.147.483.648 ... + 2.147.483.647 z.B. eine Skala von -20.000 ... +20.000 mit einer Auflösung von 0,000 01 (= 1/100.000) aufgebaut werden! Dies können Meßwerte wie Temperatur, Druck, Weg, Geschwindigkeit und vieles mehr sein, wie sie in Forschungsprojekten und Steuerungsaufgaben immer wieder vorkommen. Zu den Funktionen im Bereich Integer Arithmetik zählen u.a EXP, LD, MOD, SGN, ABS, RND, BIT, MASK, IMASK, LREAL, HREAL, LLTOR, LEN, LEN_FIFO, FREE_FIFO, READ_FIFO, etc.

Floating-Point-Arithmetik arbeitet im BASIC Tiger mit doppelter Genauigkeit (15-16 relevante Stellen), was auch hohe, wissenschaftliche Ansprüche abdeckt. Zudem stehen eine Reihe wichtiger Funktionen für komplexe Berechnungen zur Verfügung, u.a. SIN, COS, TAN, COT,ASIN, ACOS, ATAN, ACOT, SINH, COSH, TANH, COTH, LOG, LN, EXP, EXPE, SQRT, etc.

Bei den String-Funktionen wurde zu dem üblichen Funktionsumfang wie CHR$, LEFT$, RIGHT$, MID$, etc. eine Reihe starker Funktionen hinzugefügt, die eine sehr elegante und schnelle Programmierung anspruchsvoller Aufgaben erlauben. Hierdurch ist es möglich, Variablen vom Typ String in einem weit allgemeineren Zusammenhang einzusetzen, als dies traditionellerweise der Fall wäre. Insbesondere das Suchen, Selektieren, Ersetzen, Füllen, Fragmentieren und Konvertieren ist mit den neuen String-Funktionen ebenso schnell programmiert, wie diese Programme durch hohe Ablaufgeschwindigkeiten beeindrucken. Zu den neuen String-Funktionen zählen u.a. UPPER$, CONVERT$, NTOS$, RTOS$, STOS$, NFROMS, RFROMS, SELECT$, INDEX, REMOVE$, REMDOUBLE$, STRI$, etc.

Und schließlich gibt es spezielle Funktionen aus dem systemnahen Bereich die diverse Status-Informationen, Prozeßzeit, Versions-Nr, Fehler-Informationen etc. liefern.

Leere Seite

2 Device-Treiber

Durch die Verwendung von Device-Treibern, die die gerätespezifischen Eigenschaften von Peripheriegeräten berücksichtigen, erreicht der BASIC-Tiger® ein hohes Maß an Flexibilität und Leistungsfähigkeit bei einfacher Handhabung. Egal wie unterschiedlich verschiedene I/O-Geräte auch sein mögen, stets werden I/O-Kanäle, die mit Device-Treibern arbeiten, über die 6 standardmäßigen BASIC-Instruktionen **PRINT**, **PRINT USING**, **PUT**, **INPUT**, **INPUT LINE** und **GET** angesprochen. Die systematische Ansteuerung mit einer Device-Nr. (Device=Gerät) sowie optionaler Sekundär-Adresse und Funktions-Code gestattet einen systematischen, klar verstehbaren Programmaufbau. Die Umleitung von I/O auf alternative Kanäle, das Hinzufügen weiterer I/O-Kanäle und der Übergang zu anderer Hardware wird denkbar einfach. Eine I/O-Instruktion wie: „PUT #PUMPE, LEISTUNG4“ leitet den unformatierten Output der Variablen „LEISTUNG4“ auf das Gerät „PUMPE“, der in der physikalischen Realität aus ganz unterschiedlichen Kanälen bestehen kann: z.B. aus einem asynchronen seriellen Kanal, aus einem PWM-Ausgang, einer parallelen Schnittstelle oder einem ganz anderen Typ.

Die physikalischen Eigenschaften eines I/O-Devices werden weitgehend im Device-Treiber definiert und durch die Instruktion **INSTALL_DEVICE #Nr, Name** dem BASIC-Programm verfügbar gemacht. Um eine Ein- oder Ausgabe auf ein physikalisch anderes Gerät zu leiten, ist lediglich ein anderer Device-Treiber zu wählen oder ein Parameter in der INSTALL_DEVICE-Instruktion zu ändern.

Es ist die Aufgabe der Device-Treiber dem Programmierer das Leben zu erleichtern. An Stelle umfangreicher eigener Programmierungen zur Ansteuerung von I/O-Geräten kann sich der Programmierung auf die Bedienung des jeweiligen Gerätes konzentrieren. Übertragungsspezifische Details wie Puffer verwalten, Strobe-Signale erzeugen und auswerten, physikalische Adressen und Zeitverhalten handhaben, werden vom Device-Treiber übernommen. Der augenblickliche Satz von Device-Treibern wird laufend erweitert. Für spezielle Anforderungen können spezielle Treiber entwickelt werden.

Als ein Beispiel für die Vereinfachung durch Device-Treiber kann man sich den Treiber ‘LCD1.TDD’ ansehen, der die Ansteuerung eines LCD-Displays und einer Tastatur-Matrix mit bis zu 128 Tasten, Shift-LED und Piepser übernimmt. Der Treiber verwaltet nicht nur die normale Ansteuerung dieser Geräte einschließlich gepufferter Ein- und Ausgabe, es stehen auch eine Reihe leistungsfähiger ESC-Sequenzen zur Verfügung, mit denen man z.B. Tasten-Codes, Wiederholrate, Tasten-Klick, Attribute, Sonderzeichen etc. individuell einstellen kann. In seiner Gesamtheit ersetzt dieser Treiber mehrere 1000 Zeilen BASIC-Code.

Der Einsatz dieses Treibers wird u.a. in den Applikations-Programmen ‘ANA1_DEM.TIG’ und ‘LCD_SPC2.TIG’ veranschaulicht. (siehe 'Applikationen' im Device-Treiber-Handbuch).

Standard-Pinbenutzung der Device-Treiber

Folgende Tabelle zeigt die Pin- und Ressourcen-Benutzung der verschiedenen Device-Treiber. Der Datenbus (L6-0...L6-7) kann von mehreren Treibern gleichzeitig verwendet werden. Der Devicetreiber TIMERA ist die Zeitbasis für (gleichzeitig) eine Reihe weiterer Device-Treiber.

Einige Aufgaben, die von einem Device-Treiber erledigt werden könnten, sind unter den Funktionen zu finden, z.B. I²C und viele Aufgaben, die vielleicht bei dem Treiber LCD-6963 gesucht werden (Funktionen sind im Programmierhandbuch beschrieben).

	E_Ports	ANALOG	LCD1	LCD-6963	PRN1	PIN1	PLSIN1	PLSO1	PWMx	SERx	RTC1	CAN
E_PORTs			● k									
L6-0	●		●	●	●	●						
L6-1	●		●	●	●	●						
L6-2	●		●	●	●	●						
L6-3	●		●	●	●	●						
L6-4	●		●	●	●	●						
L6-5	●		●	●	●	●						
L6-6	●		●	●	●	●						
L6-7	●		●	●	●	●						
L7-0					●							
L7-1					●							
L7-2									●			
L7-3									●			
L8-0				●		●						
L8-1				●		●						
L8-2				●								
L8-3				●								
L8-4								●				
L8-5												
L8-6							●					●
L8-7												
L9-0										●		
L9-1										●		
L9-2										●		
L9-3										●		
L9-4										●		
L9-5										●		
L3-3	●											
L3-4	●											
L3-5	●											
L3-6			● L									
L3-7			● L									
L4-0												
L4-1(Modus)												
L4-2			● s									
An0		●										
An1		●										
An2		●										
An3		●										
Alarm											●	
Timer							●	●				

k=keyboard, L=LCD, s=sound

Device-Treiber Funktionen (User-Functions)

Neben der Device-Nummer und Kanalnummer kann als dritter Parameter mit ‚#' an den Device-Treiber ein User-Function-Code übergeben werden. Wenn der Device-Treiber keine Kanäle besitzt, muß aus syntaktischen der Kanal 0 angegeben werden. Mit der Instruktion GET werden Device-Treiber abgefragt, daß heißt es wird ein Wert zurückgeliefert. Mit der Instruktion PUT werden Steuerkommandos an den Device-Treiber abgesetzt. Oft ist dabei keine Ausgabe erforderlich. Aus syntaktischen Gründen muß der Ausgabe-Parameter der Instruktion PUT jedoch als eine Dummy-Ausgabe angegeben werden.

Die folgenden Funktionen werden von vielen Device-Treibern unterstützt, da sie allgemeiner Natur sind. Weiterhin gibt es spezielle Funktionen, die im Abschnitt der jeweiligen Device-Treiber beschrieben sind.

Die Abfrage- oder Steuerkommandos sind Bytes, die als 3. Parameter mit Prefix '#' nach der Device-Nummer und der Kanalnummer an den Device-Treiber übergeben werden. In der Include-Datei 'UFUNCn.INC' sind Symbole definiert, die im Quelltext besser verständlich sind als Nummern. Das 'n' in UFUNCn.INC steht für '1', '2' oder '3', da bei Änderungen in der UFUNC-Datei die Nummer eins weiter gezählt wird. Auf diese Weise bleiben ältere Programme compilier- und lauffähig. Für neue Applikationen verwenden Sie immer die UFUNC-Datei mit der höchsten Nummer.

Tip

Binden Sie die Datei UFUNCn.INC in Ihre Programme mit ein, damit dem Compiler die symbolischen Ausdrücke für die Kommandos an die Device-Treiber bekannt sind.

Einige User-Function-Codes für Abfragen (Instruktion GET), wie sie bei vielen Device-Treibern vorkommen:

Nr	Symbol Prefix UFCI_	Beschreibung
1	UFCI_IBU_FILL	Füllstand des Eingangspuffers (Byte)
2	UFCI_IBU_FREE	freier Platz im Eingangspuffer (Byte)
3	UFCI_IBU_VOL	Größe des Eingangspuffers (Byte)
33	UFCI_OBU_FILL	Füllstand des Ausgangspuffers (Byte)
34	UFCI_OBU_FREE	freier Platz im Ausgangspuffer (Byte)
35	UFCI_OBU_VOL	Größe des Ausgangspuffers (Byte)
65	UFCI_LAST_ERRC	letzter Error-Code
99	UFCI_DEV_VERS	Version des Treibers

Einige User-Function-Codes für Output (Instruktion PUT):

Nr	Symbol Prefix: UFCO_	Beschreibung
1	UFCO_IBU_ERASE	Eingangspuffer löschen
33	UFCO_OBU_ERASE	Ausgangspuffer löschen
65	UFCO_ERRC_RESET	setze letzten OK-/WARNING-/ERROR-Code zurück

Füllstand eines Puffers abfragen

Mit dem User-Function-Code UFCI_xxx_FILL werden gepufferte Device-Treiber nach den Füllstand der Puffer gefragt werden. ‚xxx' steht für IBU=Inputpuffer oder OBU=Outputpuffer. Aus einem leeren Puffer liest die Instruktion GET eine numerische Zahl mit dem Wert Null, die nicht von einer echten Null zu unterscheiden ist. Um einen gültigen Wert zu lesen, muss sichergestellt sein, daß genügend Bytes im Puffer waren.

Das folgende Beispielprogramm fragt den Füllstand des Eingangspuffers des sekundären Kanals 1 ab. Das Kommando 'UFCI_IBU_FILL' ist ein Byte, die gelesene Antwort des Treibers jedoch mindestens eine WORD-Zahl, da der Puffer größer ist als 256 Bytes. Es müssen mindestens 4 Bytes im Puffer sein, um eine gültige LONG-Zahl zu lesen. Das Beispielprogramm demonstriert die Konstruktion der Abfrage.

Programmbeispiel:

```
'------------------------------------------------------------------------
' Name: UFCI_FILL.TIG
' fragt vor dem Lesen den Fuellstand des Puffers ab
' hier: Sekundaer-Adresse 1
'------------------------------------------------------------------------
#INCLUDE UFUNC3.INC                     ' User Function Codes

TASK MAIN                               ' Beginn Task MAIN
  BYTE EVER
  LONG L                                ' LONG-Variable deklarieren
  WORD FILL_LEVEL                       ' Variable fuer den Fuellstand

  INSTALL_DEVICE #1, "SER1B_K1.TDD", &
  BD_38_400, DP_8N, YES, &              ' Einstellung SER0
  BD_19_200, DP_8N, YES                 ' Einstellung SER1

  FOR EVER = 0 TO 0 STEP 0
    GET #1, #1, #UFCI_IBU_FILL, 2, FILL_LEVEL
    IF FILL_LEVEL > 3 THEN              ' wenn min 1 LONG im Puffer
      GET #1, #1, 1, L                  ' lies ein Byte nach L
      PRINT #1, "read a LONG"           ' bestätigen
    ENDIF
  NEXT
END                                     ' Ende Task MAIN
```

Freien Platz eines Puffers abfragen

Wenn versucht wird, in einen Ausgabepuffer zu schreiben, der nicht genügend Platz für die auszugebenden Bytes hat, dann wartet die betroffene Task, bis wieder genügend Platz im Puffer frei ist. Manchmal ist das Warten unerwünscht oder es soll zumindest mit einem Time-Out verhindert werden, daß unendlich lang gewartet wird.

Vor der Ausgabe kann der freie Platz im Puffer erfragt werden. Das Kommando 'UFCI_OBU_FREE' ist ein Byte, die gelesene Antwort des Treibers jedoch mindestens eine WORD-Zahl, da der Puffer größer ist als 256 Bytes. Das Beispielprogramm demonstriert die Konstruktion der Abfrage. Die PRINT-Instruktion gibt zusätzlich zwei Zeichen aus: CR und LF. Das muss bei der Abfrage nach dem freien Platz im Puffer berücksichtigt werden.

Programmbeispiel:

```
'---------------------------------------------------------------------
' Name: UFCI_FREE.TIG
' fragt vor dem Schreiben den freien Platz des Puffers ab
'---------------------------------------------------------------------
#INCLUDE UFUNC3.INC                     ' User Function Codes

TASK MAIN                               ' Beginn Task MAIN
  LONG L                                ' LONG-Variable deklarieren
  WORD FREE_SPACE                       ' fuer den freien Platz

  INSTALL_DEVICE #1, "SER1B_K1.TDD", &
  BD_38_400, DP_8N, YES, &              ' Einstellung SER0
  BD_19_200, DP_8N, YES                 ' Einstellung SER1

  GET #1, #1, #UFCI_OBU_FREE, 2, FREE_SPACE
  IF FREE_SPACE > 10 THEN               ' wenn min 11 Bytes frei
    PRINT #1, #1, "brown fox"           ' schreibe den String + CR LF
  ENDIF
' ...
END                                     ' Ende Task MAIN
```

Hier nochmal, aber diesmal mit einer Konstruktion, die ein Time-Out behandelt.

Programmbeispiel:

```
'---------------------------------------------------------------------
' Name: TIME_OUT.TIG
' fragt vor dem Schreiben den freien Platz des Puffers ab
' und verzweigt nach einem Time-Out
'---------------------------------------------------------------------
#INCLUDE UFUNC3.INC                     ' User Function Codes

TASK MAIN                               ' Beginn Task MAIN
  LONG L                                ' LONG-Variable deklarieren
  WORD FREE_SPACE                       ' fuer den freien Platz
  LONG T, TIME_OUT                      ' Zeit in Ticks und Time-Out

  INSTALL_DEVICE #1, "SER1B_K1.TDD", &
  BD_38_400, DP_8N, YES, &              ' Einstellung SER0
  BD_19_200, DP_8N, YES                 ' Einstellung SER1

  TIME_OUT = 10000                      ' Tiem-Out in mSec

  FREE_SPACE = 0                        ' nehme an: kein Platz
  T = TICKS()                           ' Stand des Tick-Zaehlers
  WHILE FREE_SPACE < 11 &               ' solange nicht genug Platz
    AND DIFF_TICKS(T) < TIME_OUT        ' und Time-Out nicht erreicht
    GET #1, #1, #UFCI_OBU_FREE, 2, FREE_SPACE ' akt. Wert holen
  ENDWHILE
  IF DIFF_TICKS(T) < TIME_OUT THEN      ' war es ein Time-Out
    GOTO T_OUT                          ' dann verzweige
  ELSE                                  ' sonst
    PRINT #1, #1, "brown fox"           ' schreibe den String + CR LF
```

```
' ...

T_OUT:
' Hier Code um Time-Out zu behandeln

END                                     ' Ende Task MAIN
```


Pufferinhalt löschen

Manchmal ist es nötig, einen Puffer zu löschen, um ab einem definierten Zeitpunkt mit einem leeren Puffer zu beginnen. Das Kommando ‘UFCO_IBU_ERASE’ ist ein Byte. Das Beispielprogramm demonstriert lediglich die Schreibweise des Löschbefehls.

Programmbeispiel:

```
'-----------------------------------------------------------------------
' Name: UFC_ERA.TIG
' Löscht den Inhalt des Eingabepuffers
' hier: Sekundaer-Adresse 1
'-----------------------------------------------------------------------
#INCLUDE UFUNC3.INC                     ' User Function Codes

TASK MAIN                               ' Beginn Task MAIN
  INSTALL_DEVICE #1, "SER1B_K1.TDD", &
  BD_38_400, DP_8N, YES, &              ' Einstellung SER0
  BD_19_200, DP_8N, YES                 ' Einstellung SER1
' ...
  PUT #1, #1, #UFCO_IBU_ERASE, 0        ' 0 ist dummy
' ...
END                                     ' Ende Task MAIN
```

Versionsnummer des Treibers zur Laufzeit abfragen

Die Benutzeroberfläche gibt mit dem Kommando ‚Gerätetreiberliste' aus dem Menü ‚Ansicht' Aufschluß über die Versionsnummern der Device-Treiber. Auch zur Laufzeit können die Versionsnummern erfragt werden, zum Beispiel um sie zu Service-Zwecken anzuzeigen.

Die Versionsnummer ist als LONG-Zahl in 4 Bytes codiert. Es gilt folgender Aufbau:

Byte 3	**Byte 2**	**Byte 1**		**Byte 0**	
		high Nibble	low Nibble	high Nibble	low Nibble
immer 0	0=normal 1=beta	Vorkomm aziffer	1. Nachkom maziffer	2. Nachkom maziffer	Buchstabe
Beispiele					
00	00	10		00	
		1.00a			
00	00	10		42	
		1.04c			
00	00	56		78	
		5.67i			
00	01	10		00	
		beta 1.00a			

```
#INCLUDE UFUNCn.INC
GET #2,#0, #UFCI_DEV_VERS, 2, wVersion
```

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name: VERSION1.TIG
'--------------------------------------------------------------------
#INCLUDE UFUNC3.INC                     ' Definition USER-FUNCTIONS

TASK MAIN                               ' Beginn Task MAIN
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8

  STRING VRS$                           ' Variablendeklaration
  CALL GET_DEV_VERS ( 1, VRS$ )         ' hole Version von Device 1
  PRINT #1, "Version: "; VRS$
END                                     ' Ende Task MAIN


'--------------------------------------------------------------------
' Subroutine: Erzeuge Versions-String
'--------------------------------------------------------------------
SUB GET_DEV_VERS ( BYTE D; VAR STRING V$ )
  LONG V, V1                            ' Variablendeklaration
  STRING C$, V1$                        '

  V1$ = ""
        GET #D, #0, #UFCI_DEV_VERS, 4, V      ' Versionsnummer lesen
  IF V > 0FFFFh THEN                    ' pruefe ob beta-Version
    V1$ = "b"
  ENDIF
  V = V BITAND 0FFFFh                   ' maskiere Versionsnummer
  V$ = STRI$(V,"UH<4><4>   0.0.0.0.0")  ' wandele Nummer in String
  C$ = RIGHT$ ( V$, 1 )                 ' wandele rechte Ziffer
  V = ASC ( C$ ) + 49                   ' in Buchstabe
  V$ = V1$+LEFT$(V$,1)+"."+MID$(V$,1,2)+CHR$(V)
END                                     ' Ende subroutine
```

Device-Treiber-Warning-Codes

Die folgenden Warning-Codes werden von Device-Treibern bei der Abfrage nach dem letzten Error-Code zurückgeliefert. Eine Warnung bedeutet, daß eine Funktion nicht normal ausgeführt wurde. Zur besseren Lesbarkeit des BASIC-Programms sollte die Datei DEFINE_A.INC eingebunden werden und die definierten Abkürzungen verwendet werden.

Code	Symbol DEVW_	Beschreibung
1	DEVW_FAULT	Device ist beschäftigt (busy)
2	DEVW_INBU_OVL	Eingangspuffer-Überlauf
3	DEVW_ILL_CHR	Unerlaubtes Zeichen
4	DEVW_OBU_SPACE	Nicht genügend Platz im Ausgabepuffer
5	DEVW_NO_INPUT	Keine Input-Daten vorhanden

Device-Treiber-Error-Codes

Die folgenden Error-Codes werden von Device-Treibern bei der Abfrage nach dem letzten Error-Code zurückgeliefert. Zur besseren Lesbarkeit des BASIC-Programms sollte die Datei DEFINE_A.INC eingebunden werden und die definierten Abkürzungen verwendet werden.

Code	Symbol DEVE_	Beschreibung
128	DEVE_FAULT	Fehler - allgemein
129	DEVE_FATAL	Fatal Device-Error, schwerer, grundsätzlicher Device-Fehler, (z.B. Install war erfolglos)
130	DEVE_FUNC_NAV	Funktion nicht verfügbar
131	DEVE_SADR_NAV	Sekundär-ADR nicht verfügbar
132	DEVE_SFCODE_NAV	SYSTEM-Function-Code nicht verfügbar
133	DEVE_UFCODE_NAV	USER-Function-Code nicht verfügbar
134	DEVE_TOUT1	Timeout-1 ist eingetreten
135	DEVE_TOUT2	Timeout-2 ist eingetreten
136	DEVE_TVERS	Tiger-VERSION ist zu alt
137	DEVE_PARAM_WERT	Parameter-Wert nicht möglich
138	DEVE_MANY	Zu viele Parameter
139	DEVE_MISS	Parameter fehlt
140	DEVE_INT_NAV	INTERRUPT nicht verfügbar
141	DEVE_TIM_NAV	TIMER nicht verfügbar
142	DEVE_RAM_NAV	RAM nicht verfügbar
143	DEVE_RESO_NAV	RESOURCE nicht verfügbar
144	DEVE_FREQ_NAV	FREQUENZ nicht verfügbar
145	DEVE_SIZ	Parameter-Größen-Fehler
146	DEVE_PARTYP	Paramater-Typ Fehler
147	DEVE_IOINST_NAV	I/O-Instruktion nicht verfügbar
148	DEVE_NOW_DIS	Funktion-ist-JETZT-gesperrt
149	DEVE_NO_SPACE	kein Platz im Zielpuffer
150	DEVE_LOAD	zu hohe CPU-Leistung angefordert

151	DEVE_SIZE_NAV	Diese Variablengröße nicht verfügbar
152	DEVE_RESO_NINST	Ressource nicht installiert
153	DEVE_RESO_MISSIO	I/O-Channel nicht vorhanden
154	DEVE_ONLY_1HISP	Nur 1 High-Speed-Kanal verfügbar
155	DEVE_IO_VIOLAT	‚I/O-Pin exclusive'-Verletzung
156	DEVE_INP_TOF	Input-Data zum Flash nicht möglich
157	DEVE_NOFLADR	ungültige Flash-Adresse
158	DEVE_1DACC	Nur 1 High-Speed-Kanal verfügbar

Leere Seite

A/D-Eingänge mit Analog1

Der Device-Treiber ‘ANALOG1’ liest ungepuffert momentan anliegende Analogwerte ein.

Dateiname: ANALOG1.TDD

INSTALL DEVICE #*D*, "ANALOG1.TDD"

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers .

Der Device-Treiber ANALOG1.TDD liest die internen Analog-Eingänge ein. Es werden jeweils die aktuellen, gegenwärtig anliegenden Meßwerte gelesen. Den Sekundär-Adressen 0...3 entsprechen den 4 Eingangs-Kanäle. Unter Sekundär-Adresse 4 können alle 4 Kanäle auf einmal mit 8 Bit gelesen werden. Es wird nur eine Variable angegeben. Die 4 Bytes werden in einen String oder einen LONG-Wert eingelesen

Die Auflösung beträgt 8 Bit wenn BYTEs eingelesen werden (z.B.: GET #n,#sa,1,CHAR) bzw. 10 Bit, wenn WORDs oder LONGs eingelesen werden.

Mit Hilfe der FIFO-Puffer und des Befehls INTEGRAL_FIFO ist es möglich, die Auflösung zu steigern und Rauschen „herauszurechnen“.

Beispiele:

GET #4,#0,1,Wert liest vom Analog1-Treiber (hier: Kanal 4) vom A/D-Kanal 0 genau 1 Byte nach Wert (8-Bit-Auflösung). Wert ist vom Typ LONG, WORD oder BYTE:

GET #4,#3,2,Wert liest vom Analog1-Treiber vom A/D-Kanal 3 genau 2 Bytes nach Wert (10-Bit-Auflösung). Wert ist vom Typ LONG oder WORD.

GET #4,#4,4,Wert liest vom Analog1-Treiber von den A/D-Kanälen 0...3 genau je 1 Byte nach Wert (4x 8-Bit-Auflösung). Wert ist vom Typ LONG und das Byte von Kanal 0 ist das niederwertigste Byte.

**GET #4,#5,8,W$** lies 4 Kanaele à 10 Bit. Die Werte können z.B. so ausgegeben werden: liest vom Analog1-Treiber von den A/D-Kanälen 0...3 genau je 2 Byte nach Wert$ (4x 10-Bit-Auflösung). Wert$ muß eine String-Variable sein, um alle 8 Bytes aufzunehmen. Das niederwertige Byte von Kanal 0 ist das erste Byte. Die Funktion NFROMS (Number from String) liest die Bytes in richtiger Form aus dem String.

PRINT #1, NFROMS (W$, 0, 2) ‘ gebe Wert von Kanal 0 aus

Programmbeispiel:

```
'--------------------------------------------------------------------------
' Name: ANALOG1.TIG
'--------------------------------------------------------------------------
TASK Main                                ' Beginn Task MAIN
  BYTE EVER, K
  ARRAY Value(4) OF LONG                 ' LONG-Array deklarieren

' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  INSTALL DEVICE #4, "ANALOG3.TDD", &    ' Analog-Inputs installieren
    0, &       ' Port fuer Datenbus-Port = default
    0, &       ' Port fuer Steuerleitungen = default
    3, &    ' Pin-Nr. fuer -RD = default                                pin
    4, &    ' Pin-Nr. fuer -WR = default                                pin
    5, &    ' Pin-Nr. fuer HBEN = default                               pin
    7, &    ' Pin-Nr. fuer -CE = default                                pin
    22, &    ' speed reduction = default (keine)                         sp
    0, &       ' Reservierter Parameter immer 0
    3, &    ' Anzahl Adressleitungen = default                          no.
    8, &       ' Port fuer Adressleitungen = default
    0eeh &     ' Bitposition der Adressleitungen = default

  PUT #4, #0, "00 00 00 00 00 00 00 00"% ' ch0...7: alle 0...5V

  FOR EVER = 0 TO 0 STEP 0               ' Endlosschleife
    FOR K = 0 TO 7                       ' 8 Kanaele
      GET #4, #K, 2, Value(K)            ' Wert aus AD-Wandler lesen
    NEXT                                 ' naechster Kanal
    PRINT #1, "<1>";                     ' Bildschirm loeschen
    FOR K = 0 TO 7 STEP 2                ' 8 Kanaele anzeigen
      PRINT #1, "AD";K;":";Value(K)      ' Kanal-Nr. + Wert anzeigen
      PRINT #1, "AD";K+1;":";Value(K+1)  ' Kanal-Nr. + Wert anzeigen
    NEXT                                 ' naechster Kanal
    WAIT_DURATION 100                    ' 100 ms warten
  NEXT
END                                      ' Ende Task MAIN
```

A/D-Eingänge mit Analog2

Der Device-Treiber ‘ANALOG2’ liest gesteuert durch den Zeitbasis-Device-Treiber ‘TIMERA’ Analogwerte ein und legt diese in einem FIFO-Puffer ab (FIFO=First-In-First-Out).

Weitere Informationen zu ANALOG2.TDD:

- User-Function-Codes
- Messen in FIFO
- Messen in String
- Messungen mit 12-Bit
- Sample-Rate einstellen

Dateiname: ANALOG2.TDD

INSTALL DEVICE #*D*, "ANALOG2.TDD"

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Der Device-Treiber ANALOG2.TDD liest analoge Meßwerte von den internen Analogkanälen in einen FIFO-Puffer oder String ein. Die Messungen werden mit Hilfe des Zeitbasis-Treibers ‘TIMERA.TDD’ synchronisiert, so daß die Messungen unabhängig vom BASIC automatisch und bis zu hohen Geschwindigkeiten ausgeführt werden. Der Zeitbasis-Treiber stellt eine Basisfrequenz zur Verfügung, die durch den Pre-Scaler des Treibers ANALOG2 zur eigentlichen Meßrate heruntergeteilt wird. Die Einstellung des Pre-Scalers ist durch Kommandos (User-Function-Code) an den Treiber veränderbar.

Hinweis: TIMERA.TDD muß vorher eingebunden werden.

Der Treiber unterstützt die Auflösungen 8-Bit, 10-Bit und 12-Bit-interpoliert. Die Analogwerte können entweder in einen String oder in einen FIFO-Puffer eingelesen werden. Folgende Lesemodi werden unterstützt:

- von einem beliebigen Kanal
- von Kanal 0 und 1
- von Kanal 0, 1 und 2
- von Kanal 0, 1, 2 und 3

Es gibt also viele verschiedene Einstellungen, von welchem Kanal in welcher Auflösung wohin Analogwerte eingelesen werden. Dazu kann die Geschwindigkeit (Meß- oder Sample-Rate) in vielfältiger Weise verstellt werden. Zusätzlich können Optionen gewählt werden, die sich auf das Verhalten der Messung in Bezug auf Strings oder FIFO-Puffer beziehen. Es folgen daher einige Hinweise, was die

Unterschiede zwischen ‘Messung in String’ und ‘Messung in FIFO’ sind und was bei den verschiedenen Einstellungen zu beachten ist.

Zur Einstellung des analogen Meßsystems stehen User-Function-Codes zur Verfügung, die als symbolische Namen in UFUNCn.INC definiert sind. Einmal durchgeführte Einstellungen bleiben erhalten und müssen nicht vor jeder Messung wieder neu vorgenommen werden. Werden beim Start der Messung Optionen explizit angegeben (Offset im String, Anzahl der Messungen), dann gelten diese nur für diese eine Messung. Die zuvor mit mit Hilfe der User-Function-Codes vorgenommenen Einstellungen bleiben erhalten.

Die folgende Tabelle zeigt eine Übersicht über die besonderen Function-Codes dieses Treibers. Die Datei UFUNCx.INC muß eingebunden sein, damit dem Compiler die Kommando-Symbole bekannt sind.

User-Function-Codes des ANALOG2.TDD

User-Function-Codes des ANALOG2.TDD zum Setzen von Parametern (PUT):

Nr	Symbol Prefix: UFCO_	Beschreibung
46	UFCO_AD2_RESET	Setze alle Parameter auf Standardwerte
128	UFCO_AD2_CHAN	Setze Einzelkanal-Modus (FIFO, STRING): 0, 1, 2, 3 Dieser Kanal ist auch der Meßkanal im Modus mehrkanaliger Messung, wenn nur ein Kanal eingestellt ist.
129	UFCO_AD2_RESO	Setze Auflösung (FIFO, STRING): 8 = 8-Bit 10 = 10-Bit 12 = 12-Bit
130	UFCO_AD2_INTEG	Integrations-Weite bei 12-Bit (FIFO, STRING): 16, 32, 64, oder 128
132	UFCO_AD2_CNT	Anzahl von Messungen (je Kanal) (FIFO) 0 = endlos (nur für FIFO) n = Anzahl (LONG)
133	UFCO_AD2_PSCAL	Pre-Scaler, teilt die Basisfrequenz des Treibers "TIMERA.TDD" herunter (FIFO, STRING): 0,1 = ohne Prescaler n = Teiler (WORD)
134	UFCO_AD2_STOP	Stop AD-Sampling (FIFO, STRING): nur DUMMY-Parameter
135	UFCO_AD2_GROF	Set Growth-Flag (STRING) 0 = spontane Größen-Zuweisung am Ende der Messreihe sonst = kontinuierliche Erhöhung der String-Länge mit jeder Messung (wenn sich String verlängert!)
136	UFCO_AD2_SCAN	Setze Mehrkanal-Modus und Anzahl der Kanäle (FIFO, STRING): 1, 2, 3 oder 4 Kanäle gleichzeitig n = 1: der zuletzt mit UFC_AD2_CHAN eingestellte Kanal n = 2: 2-Kanal: Ch-0, Ch-1 n = 3: 3-Kanal: Ch-0, Ch-1, Ch-2 n = 4: 4-Kanal: Ch-0, Ch-1, Ch-2, Ch-3
137	UFCO_AD2_ISAMP	Integral-Samples (FIFO, STRING): gibt an,

Nr	Symbol Prefix: UFCO_	Beschreibung
		jede wievielte Messung in den ZIEL-Puffer geschrieben werden soll. Gilt nur, wenn INTEGRATION statt findet (nur bei 12-Bit) Werte: 1...65535 (WORD)
143	UFCO_AD2_PSCIMM	setzt den Pre-Scaler während der laufenden Messung

User-Function-Codes des ANALOG2.TDD zum Abfragen von Parametern (GET):

Nr	Symbol Prefix: UFCI_	Beschreibung
68	UFCI_CPU_LOAD	lese die durch diesen Treiber beanspruchte CPU-Leistung (100%=10.000)
99	UFCI_DEV_VERS	Version des Treibers

Messen in FIFO

Zunächst legen Sie die gewünschte Auflösung, die gewünschte (maximale) Meßrate sowie die Anzahl der Kanäle fest.

Wollen Sie einen unterbrechungsfreien Fluß von Meßdaten erzeugen, dann müssen Sie in einen FIFO-Puffer messen. Der Abtransport und die Weiterverarbeitung der Meßdaten bestimmt dabei die maximal mögliche Meßrate, die noch möglich ist, ohne daß der FIFO-Puffer überläuft und dadurch Meßdaten verloren gehen. Je größer der FIFO-Puffer ist, desto größer dürfen auch die Geschwindigkeitsschwankungen bei der Weiterverarbeitung der Meßdaten sein. Das Messen in einen FIFO-Puffer sowie das Abholen der Daten ist jedoch etwas langsamer als das Messen in einen String. Wenn der FIFO-Puffer voll ist, stoppt der Device-Treiber die Messung . Für weitere Messungen muß ein Neustart erfolgen.

Bei 12-Bit Auflösung kann die Integrationstiefe eingestellt werden (Größe des internen Integrationspuffers). Die Anzahl der Meßwerte kann reduziert werden, indem nicht jeder intern anfallende Meßwert in den String oder FIFO-Puffer übernommen wird. So entstehen zeitlich weiter auseinanderliegende Messungen, die jedoch durch eine gewisse Integrationstiefe bereinigt sind.

Nachdem alle Einstellungen mit Hilfe der User-Function-Codes vorgenommen worden sind, wird die Messung in einen FIFO-Puffer so gestartet:

PUT #D, *FIFO_Name*

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

FIFO_Name ist der FIFO-Puffer, in den die Meßwerte geschrieben werden. Der Puffer ist FIFO of Byte bei 8-Bit Messungen und FIFO of Word bei 10- oder 12-Bit Messungen. Der FIFO-Puffer wird automatisch bei Start auf LEER gesetzt.

Bitte beachten Sie, daß bei Integration (12-Bit) die gemessenen Werte erst stimmen, wenn der interne Integrationspuffer einmal gefüllt worden ist.

Die Messung kann mit dem User-Function-Code UFCO_AD2_STOP vorzeitig gestoppt werden.

Programmbeispiel:

```
'-------------------------------------------------------------------
' Name: ANALOG2F.TIG
'-------------------------------------------------------------------
#INCLUDE DEFINE_A.INC                   ' allgemeine Definitionen
#INCLUDE UFUNC3.INC                     ' User Function Codes

TASK MAIN                               ' Beginn Task MAIN
  FIFO SAMPLE (256) OF WORD             ' Sample-Puffer
  WORD A, B, C, D                       ' Var. fuer Analogwerte
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #LCD, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #LCD, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
' TIMER-A Treiber installieren (Zeitbasis-Timer: 1001Hz)
  INSTALL_DEVICE #TA, "TIMERA.TDD", 3, 156
' ANALOG-2 Treiber installieren
  INSTALL_DEVICE #AD2, "ANALOG2.TDD"

  PUT #AD2,#0,#UFCO_AD2_RESO, 10        ' Aufloesung
  PUT #AD2,#0,#UFCO_AD2_SCAN, 4         ' Anzahl Kanaele
  PUT #AD2,#0,#UFCO_AD2_STOVL, 0        ' stop on overflow
  PUT #AD2,#0,#UFCO_AD2_PSCAL, 5        ' Pre-Scaler: 1001/5=200S/sec
  PUT #AD2,SAMPLE                       ' Starte Messungen

  K = 0
  WHILE K < 127                         ' Ende wenn FIFO voll ist
    K = LEN_FIFO(SAMPLE)                '
    PRINT #LCD, "<1>Length=";K
  ENDWHILE
                                        ' ----------------------------
  WHILE LEN_FIFO(SAMPLE) > 4            ' FIFO anzeigen
    GET_FIFO SAMPLE, A
    PRINT #LCD, "<1bh>A<12><0><0f0h>";A;
    GET_FIFO SAMPLE, B
    PRINT #LCD, "<1bh>A<12><1><0f0h>";B;
    GET_FIFO SAMPLE, C
    PRINT #LCD, "<1bh>A<12><2><0f0h>";C;
    GET_FIFO SAMPLE, D
    PRINT #LCD, "<1bh>A<12><3><0f0h>";D;
  ENDWHILE
  PRINT #LCD, "<1Bh>A<0><3><0F0h>ready";
END                                     ' Ende Task MAIN
```

Messen in String

Zunächst legen Sie die gewünschte Auflösung, die gewünschte (maximale) Meßrate sowie die maximale Anzahl der Kanäle fest.

Wollen Sie aufeinanderfolgende Meßabschnitte erzeugen, so empfiehlt es sich in einen String zu messen. Vorteile: Die Messung verbraucht weniger CPU-Leistung und die Weiterverarbeitung ist aufgrund der Stringverarbeitungsfunktionen schneller als bei Messungen in FIFO-Puffer. Wenn der String z.B. seriell weitergegeben werden soll, kann dieser in Stücken à 240 Bytes (das ist die Begrenzung der Instruktionen PRINT und PUT) direkt weitergesendet werden. Die Messung wird automatisch gestoppt, wenn der String voll ist.

Um Analogwerte in einen String einzulesen, wird zunächst ein String in der passenden Länge deklariert. Der Zeitbasis-Treiber TIMERA.TDD wird eingebunden und auf die höchste in der Applikation benötigte Basisfrequenz eingestellt. Weitere Einstellungen wie Pre-Scaler, Auflösung, Anzahl der Kanäle sowie die Anzahl der Messungen werden festgelegt.

! Der Meß-String muß immer existieren! Nicht erlaubt sind also Variable, die nur vorübergehend leben, wie lokale Strings (in Unterprogrammen) oder temporäre Strings (Expressions). **Richtig: globale oder Task-lokale Strings**.

Die Messung muß nicht ab der Position 0 in den String geschrieben werden. Ebenso muß der String nicht leer sein. Es kann ein Offset angegeben werden, ab welcher Position in den String geschrieben wird. Werte vor der Schreibposition bleiben erhalten, wenn sie vorher definiert waren. War der String kürzer als als der Offset, dann bestehen undefinierte Werte vor der Schreibposition.

Die Messung wird beendet, wenn entweder die gesetzte Anzahl der Messungen erreicht ist oder wenn der String voll ist. Jedoch erreicht der String nicht unbedingt genau seine maximale Länge: wenn bei 4-Kanal-Messung mit 8-Bit Auflösung z.B. noch 2 Bytes im String frei sind, findet die Messung nicht mehr statt, weil bei jeder Messung 4 Bytes entstehen. Die Stringlänge bleibt also 2 Bytes unter der maximalen Länge.

Nachdem alle Einstellungen mit Hilfe der User-Function-Codes vorgenommen worden sind, wird die Messung in einen String so gestartet:

PUT #D, *String* [, *Offset*, *Anzahl*, *Growth_Flag*]

D	ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.
String	ist der String, in den die Meßwerte geschrieben werden. **Der String muß statisch, d.h. global oder task-lokal sein.** Beachte: der String wird bei Start nicht auf LEER gesetzt.
Offset	ist eine Variable, eine Konstante oder ein Ausdruck vom Typ BYTE, WORD oder LONG und gibt den Offset an, wenn die Meßwerte ab einer Position ungleich 0 in den String geschrieben werden sollen. Default = 0 (String-Anfang).
Anzahl	ist eine Variable, eine Konstante oder ein Ausdruck vom Typ BYTE, WORD oder LONG und gibt die Anzahl der Messungen an. Bei mehrkanaliger Messung werden entsprechend mehr Bytes pro Messung erzeugt. 10-Bit oder 12-Bit Messungen erzeugen 2 Bytes pro Messung und pro Kanal. Wenn Anzahl mit 0 angegeben wird, dann wird bis zum Ende des Strings gemessen.
Growth_Flag	ist eine Variable, eine Konstante oder ein Ausdruck vom Typ BYTE, WORD oder LONG und gibt an, ob die Größe des Strings kontinuierlich mit den Messungen wächst oder ob nach Beendigung der Messung die Größe gesetzt wird: 0: String wächst kontinuierlich. ungleich 0: Stringgröße wird nach der Messung gesetzt.

Wenn die Angaben Offset, Anzahl oder Growth_Flag fehlen, gelten die zuvor mit User-Function-Code gesetzten Einstellungen. Sind die Angaben Offset, Anzahl oder Growth_Flag vorhanden, so gelten sie nur für diese eine Messung und beeinflussen nicht die gesetzten Einstellungen.

Führt die angegebene Anzahl der Messungen dazu, daß die Stringgröße überschritten wird, so wird die Messung dann beendet, wenn kein Platz mehr für eine weitere Messung im Sting ist. Das kann vor Erreichen der maximalen Stringlänge der Fall sein, wenn z.B. bei 4-kanaliger 10-Bit-Messung (8 Bytes pro Messung) weniger als 8 Bytes im String zur Verfügung stehen.

Die Messung kann mit dem User-Function-Code UFCO_AD2_STOP vorzeitig gestoppt werden.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name: ANALOG2S.TIG
'--------------------------------------------------------------------
#INCLUDE DEFINE_A.INC                   ' allgemeine Definitionen
#INCLUDE UFUNC3.INC                     ' User Function Codes
STRING M$ (150)                         ' Messwert-String (global!)

TASK MAIN                               ' Beginn Task MAIN
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #LCD, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #LCD, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
' TIMER-A Treiber installieren (Zeitbasis Timer: 1001Hz)
  INSTALL_DEVICE #TA, "TIMERA.TDD", 3, 156
' ANALOG-2 Treiber installieren
  INSTALL_DEVICE #AD2, "ANALOG2.TDD"

  M$=""                                 ' Mess-String leer
  PUT #AD2,#0,#UFCO_AD2_PSCAL, 5        ' Pre-Scaler: 1001/5=200S/sec
  PUT #AD2,#0,#UFCO_AD2_RESO, 8         ' Aufloesung
  PUT #AD2,#0,#UFCO_AD2_SCAN, 4         ' Anzahl Kanaele
  PUT #AD2,M$,0,300,1                   ' Startpos., Anzahl Messungen
                                        ' groesser als String!
  K = 0                                 ' Ende wenn String voll ist
  WHILE K < 148                         ' String wird nicht 150 lang!
    K = LEN(M$)                         ' sondern 4Ch x 37 = 148
    PRINT #LCD, "<1>Length=";K
  ENDWHILE
                                        ' ----------------------------
  FOR I = 0 TO LEN(M$)-4 STEP 4         ' STRING anzeigen
    PRINT #LCD, "<1Bh>A<12><0><0F0h>0:";NFROMS(M$,I,1);
    PRINT #LCD, "<1Bh>A<12><1><0F0h>1:";NFROMS(M$,I+1,1);
    PRINT #LCD, "<1Bh>A<12><2><0F0h>2:";NFROMS(M$,I+2,1);
    PRINT #LCD, "<1Bh>A<12><3><0F0h>3:";NFROMS(M$,I+3,1);
    WAIT_DURATION 1000                  ' 1 Sek. warten
  NEXT
  PRINT #LCD, "<1Bh>A<0><3><0F0h>ready";

END                                     ' Ende Task MAIN
```

Messungen mit 12-Bit

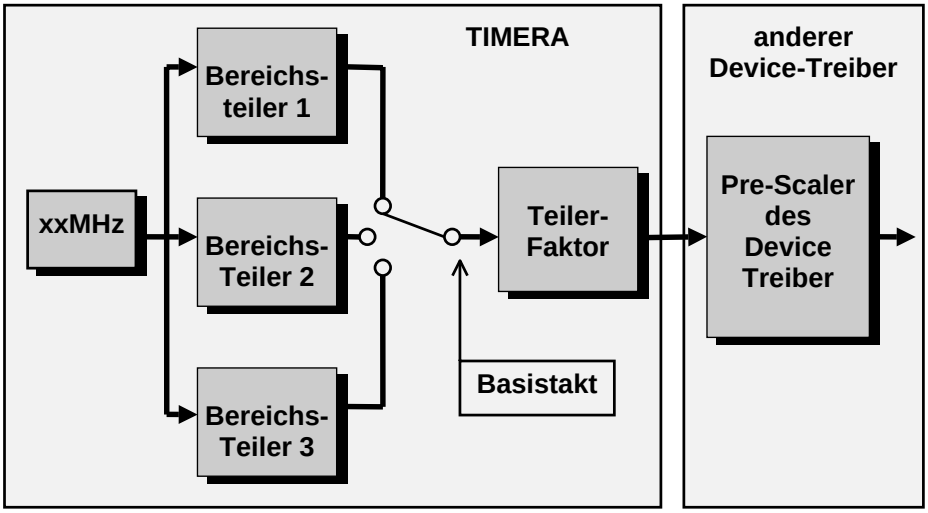
Bei 12-Bit Auflösung kann die Integrationstiefe eingestellt werden(Größe des internen Integrationspuffers). Die Anzahl der Meßwerte kann reduziert werden, indem nicht jeder intern anfallende Meßwert in den String oder FIFO-Puffer übernommen wird. So entstehen zeitlich weiter auseinanderliegende Messungen, die jedoch durch eine gewisse Integrationstiefe bereinigt sind.

```
PUT #7, #0, #UFCO_AD2_RESO, 12   ' 12-Bit Aufloesung einstellen
PUT #7, #0, #UFCO_AD2_INTEG, 32  ' 32 Bit Integrationspuffer
PUT #7, #0, #UFCO_AD2_ISAMP, 9   ' nur jede 9. Messung wird genommen
                                 ' die Sample-Rate wird durch 9 geteilt
```

Je größer der Integrationspuffer ist, um so genauer ist die Messung. Jedoch wird auch die Tiefpass-Wirkung größer, d.h. schnelle Signal-Änderungen werden herausgefiltert.

Sample-Rate einstellen

Die Meßrate oder Sample-Rate wird von der Basisfrequenz des Device-Treibers TIMERA abgeleitet. Der Pre-Scaler des Device-Treibers ANALOG2 teilt dazu die Basisfrequenz weiter herunter:



Nähere Informationen zum Einstellen des Device-Treibers TIMERA finden Sie ab Seite 349. Der Pre-Scaler wird mit dem User-Function-Code UFCO_AD2_PSCAL eingestellt. Beispiele finden Sie weiter oben unter ‘Messen in FIFO’ und ‘Messen in String’.

! Dieser Device-Treiber kann zusammen mit dem Treiber TIMERA bei ‘schneller’ Einstellung soviel CPU-Leistung beanspruchen, daß andere Tasks stark behindert werden. Mit dem User-Function-Code ‘UFCI_CPU_LOAD kann die Belastung der CPU, wie sie allein durch diesen Treiber verursacht wird, abgefragt werden.
Der Treiber kann bestimmte Einstellungen, die zur Überlastung des Systems führen würden, nicht annehmen.
Beachte: TIMERA muß **vor** ANALOG2 installiert werden.

Leere Seite

A/D-Eingänge mit Analog3

Der Device-Treiber ‘ANALOG3’ liest Analogwerte von dem externen A/D-Modul EP11 ein.

Weitere Informationen zu ANALOG3.TDD:

- Sekundär-Adressen des ANALOG3.TDD
- User-Function-Codes des ANALOG3.TDD
- Meßbereich skalieren
- Eingangsspannungsbereiche einstellen
- Kanalgruppen definieren
- Kanalgruppe löschen
- A/D-Wert lesen und Von einer Kanalgruppe lesen

Dateiname: ANALOG3.TDD

INSTALL DEVICE #*D*, "ANALOG3.TDD", P1,..., P11

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

P1...P11 sind weitere Parameter, die den Anschluß des EP11-Moduls an BASIC-Tiger® festlegen. Diese Parameter sind in der nachfolgenden Tabelle näher beschrieben.

Alle Parameter sind Bytes und können durch die Angabe von 0 oder 0EEH (=238) den Standardwert unverändert lassen.

	Standard	unverändert lassen	Beschreibung des Parameters
P1	6	0	Logische BUS-Adresse für EP11
P2	8	0	Logische Portadresse für: -RD, -WR, HBEN, -CE
P3	0	0eeh	Bitnummer für Signal '-RD'
P4	1	0eeh	Bitnummer für Signal '-WR'
P5	3	0eeh	Bitnummer für Signal '-HBEN'
P6	5	0eeh	Bitnummer für Signal '-CE'
P7	1	0eeh	I/O Access-Speed-Reduction (1=no,2...32)
P8	0	0eeh	Immer 0, reservierter Parameter
P9	0	0eeh	0 = keine Adreß-Signale verwendet 1 = eine Adreßleitung für 2 A/D-Ports 2 = zwei Adreßleitungen für 4 A/D-Ports 3 = drei Adreßleitungen für 8 A/D-Ports
P10	7	0	Logische Portadresse für Adreß-Signale des EP11
P11	0	0eeh	Erste Bit-Position für Adreß-Signale (0...7) z.B. zwei Adreß-Signale an Port 8, erste Position 6: L86 und L87 sind Adreß-Signale

I/O Access-Speed-Reduction wird bei der Kombination BASIC-Tiger® A, Tiny-Tiger® in Kombination mit dem Modul EP11 auf '1' gesetzt, d.h. ohne Geschwindigkeitsreduzierung. Ein anderer Wert kann notwendig werden, wenn kompatible A/D-Module auf den Markt kommen, die jedoch langsamer wandeln, oder wenn schneller BASIC-Tiger®-Module verfügbar werden.

Die **Steuerleitungen** können teilweise mit anderen Device-Treibern gemeinsam verwendet werden, insbesondere mit den Steuerleitungen des Grafik-LCDs.

Der Device-Treiber ANALOG3.TDD liest analoge Meßwerte von den externen Analogkanälen des analogen Erweiterungsmoduls EP11 ein. Die Messungen werden mit der GET-Instruktion ausgeführt und die Ergebnisse dann gelesen.

Die Auflösung beträgt 12 Bit. Der Treiber unterstützt die verschiedenen Modi der A/D-Eingänge des EP11-Moduls:

- Eingangsspannung 0...5V
- Eingangsspannung -5V...+5V
- Eingangsspannung 0...10V
- Eingangsspannung -10V...+10V

Zwei verschiedene Lesemodi stehen zur Verfügung:

- von einem beliebigen Kanal
- von einer Kanalgruppe, die zuvor zusammengestellt wurde

Die Einstellungen der Kanal-Modi sowie die Zusammenstellung der Gruppen erfolgt vor der Messung. Die Messung wird dann mit der Instruktion GET initiiert und die Ergebnisse direkt eingelesen.

Die Einstellungen des analogen Meßsystems werden durch Ausgabe von Strings auf bestimmte Sekundär-Adressen vorgenommen. Einmal durchgeführte Einstellungen bleiben erhalten und müssen nicht vor jeder Messung wieder neu vorgenommen werden.

Die folgende Tabelle zeigt eine Übersicht über die besonderen Funktionen des Treibers auf den Sekundär-Adressen.

Sekundär-Adressen des ANALOG3.TDD

Sekundär-Adresse	Beschreibung
0...63	PUT: Einstellung der Eingangsspannungsbereiche der Kanäle 0 bis 63 GET: Lesen von einem einzelnen Kanal
64...71	PUT: Kanalgruppe definieren (8 Gruppen sind möglich) GET: von Kanalgruppe alle definierten Kanäle lesen

User-Function-Codes des ANALOG3.TDD

User-Function-Codes des ANALOG3.TDD zum Setzen von Parametern (PUT):

Nr	Symbol Prefix: UFCO_	Beschreibung
144	UFCO_AD3_FACTOR	Setzt den Wert, der bei maximalem A/D-Meßwert des entsprechenden Kanals geliefert wird (WORD). 4096 entspricht Faktor 1

Meßbereich skalieren

Mit dem User-Function-Code UFCO_AD3_FACTOR kann ein Wert gesetzt werden, der zu einer Skalierung des Meßwertes herangezogen wird. Der Skalierungsfaktor kann für jeden Kanal einzeln gesetzt werden, indem auf die Sekundäradresse des Kanals geschrieben wird. Werden mehrere Werte auf eine Sekundäradresse geschrieben, dann werden die Faktoren ab der Sekundäradresse und für die folgenden vergeben. Nach einem Reset sind die Werte auf den Faktor 1 gesetzt, was dem Faktorwert 4096 entspricht.

Der A/D-Wandler liefert als Maximalwert 4095. Folgende Rechnung findet mit dem Faktor statt:

Meßergebnis = (Messung * Faktor + 2048) / 4096

Zur Rundung werden 2048 nach der Multiplikation dazu addiert. Ein Faktorwert von z.B. 100 führt zu folgender Rechnung bei Maximalwert des A/D-Wandlers:

Meßergebnis = (4095 * 100 + 2048) / 4096

was zu einem Meßergebnis von 100 führt. Der Faktorwert gibt also den gewünschten Wert für Vollausschlag an.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name: ANA3_8XV.TIG
'--------------------------------------------------------------------
TASK Main                                ' Beginn Task MAIN
  BYTE EVER, K
  REAL V, W
  ARRAY Value(8) OF WORD                 ' WORD-Array deklarieren

 DIR_PORT 8,0                            '

' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  INSTALL DEVICE #4, "ANALOG3.TDD", &   ' Analog-Inputs installieren
    6, &       ' Port fuer Datenbus-Port = default
    8, &       ' Port fuer Steuerleitungen = default
    3, &       ' Pin-Nr. fuer -RD = default
    4, &       ' Pin-Nr. fuer -WR = default
    5, &       ' Pin-Nr. fuer HBEN = default
    7, &       ' Pin-Nr. fuer -CE = default
    22, &      ' speed reduction = default (keine)
    0, &       ' Reservierter Parameter immer 0
    3, &       ' Anzahl Adressleitungen = default
    8, &       ' Port fuer Adressleitungen = default
    0 &        ' Bitposition der Adressleitungen = default

  PRINT #1,"EP11_001 - Test"
  PUT #4, #0, "00 00 00 00 00 00 00 00"% ' ch0...7: alle 0...5V
                                         ' setze Faktoren f. 4 Kanaele
  SET_LEN$ ( FACTOR$, 8 )                ' init. String f. 4 WORDs
  F4 = 100                               ' Faktor fuer Kanal 4
  FACTOR$ = NTOS$( FACTOR$, 0, 2, F4 )   ' einbauen
  F5 = 2550                              ' Faktor fuer Kanal 5
  FACTOR$ = NTOS$( FACTOR$, 0, 2, F5 )   ' einbauen
  F6 = 10000                             ' Faktor fuer Kanal 6
  FACTOR$ = NTOS$( FACTOR$, 0, 2, F6 )   ' einbauen
  F7 = 2550                              ' Faktor fuer Kanal 7
  FACTOR$ = NTOS$( FACTOR$, 0, 2, F7 )   ' einbauen
  PUT #4, #4, FACTOR$                    ' Faktor setzen Kanaele 4,5,6,7

  FOR EVER = 0 TO 0 STEP 0               ' Endlosschleife
    FOR K = 0 TO 7                       ' 8 Kanaele
      GET #4, #K, 2, VALUE(K)            ' Wert aus AD-Wandler lesen
    NEXT                                 ' naechster Kanal
    PRINT #1, "<1>";                     ' Bildschirm loeschen
    FOR K = 0 TO 7 STEP 2                ' 8 Kanaele anzeigen
      V = (LTR(Value(K))/4096.0 * 5000.0)/1000.0
      USING "UD<1><1>   0.0.0.0.1"
      PRINT_USING #1, K;":";             ' Kanal-Nr. + Wert
      USING "NF<1><1> <3> 0 0 0 0 0 0 0 1.3 0 0 0 0 0 0 0 0"
      PRINT_USING #1, V;                 ' Kanal-Nr. + Wert

      V = (LTR(Value(K+1))/4096.0 * 5000.0)/1000.0
      USING "UD<1><1>   0.0.0.0.1"
      PRINT_USING #1, "  ";K+1;":";     ' Kanal-Nr. + Wert
      USING "NF<1><1> <3> 0 0 0 0 0 0 0 1.3 0 0 0 0 0 0 0 0"
      PRINT_USING #1, V                  ' Kanal-Nr. + Wert
```

```
    NEXT                                ' naechster Kanal
    WAIT_DURATION 100                   ' 100 ms warten
  NEXT
END                                     ' Ende Task MAIN
```

Eingangsspannungsbereiche einstellen

Am Erweiterungsmodul EP11 läßt sich für jeden Kanal ein Eingangsspannungsbereich für den A/D-Wandler einstellen. Die Codierung des gewünschten Bereichs erfolgt in den unteren zwei Bits eines Bytes. Für jeden einzustellenden Kanal wird ein Byte mit PUT an den Device-Treiber übertragen. Durch Schreiben auf eine bestimmte Sekundär-Adresse wird der erste zu setzende Kanal ausgewählt. Die folgenden Bytes werden entsprechend für die nachfolgenden Kanäle verwendet. Die Tabelle zeigt die Codierung der Bereiche:

Codierung binär	Codierung HEX	Eingangsspannungsbereich
00000000b	0	unipolar 0V...+5V
00000001b	1	bipolar –5V...+5V
00000010b	2	unipolar 0V...+10V
00000011b	3	biipolar –10V...+10VV

Das folgende Beispiel setzt die Eingangsspannungsbereiche der Kanäle 17 bis 22:

```
'...
'               +---------------------- Kanal 17:   0V... +5V
'               !  +------------------- Kanal 18:   0V...+10V
'               !  !  +---------------- Kanal 19:  -5V... +5V
'               !  !  !  +------------- Kanal 20: -10V...+10V
'               !  !  !  !  +---------- Kanal 21: -10V...+10V
'               !  !  !  !  !  +--------Kanal 22:  -5V... +5V
PUT #AD3, #17, "00 02 01 03 03 01"%
'...
```

Kanalgruppen definieren

Wenn bestimmte A/D-Kanäle öfter gelesen werden und andere Kanäle wiederum nur selten, dann können Kanalgruppen gebildet werden, um das Lesen bestimmter Kanäle zusammenzufassen. Der Device-Treiber ANALOG3.TDD unterstützt bis zu 8 Gruppen. Jede Gruppe kann bis zu 64 Kanäle enthalten. Die Gruppen werden durch Ausgeben von Strings mit der Instruktion PUT auf die Sekundär-Adressen 64...71 definiert.

Sekundär-Adresse	Kanalgruppe
64	Kanalgruppe 0
65	Kanalgruppe 1
66	Kanalgruppe 2
67	Kanalgruppe 3
68	Kanalgruppe 4
69	Kanalgruppe 5
70	Kanalgruppe 6
71	Kanalgruppe 7

Die ausgegebenen Strings enthalten die Kanalnummern. Kanalnummern außerhalb des gültigen Bereichs werden ignoriert. Kanäle dürfen in einer Gruppe frei zusammengestellt werden. Auch doppelte Verwendung sowie Überlappungen mit anderen Gruppen sind erlaubt. Die Kanäle werden später in der Reihenfolge gelesen, wie sie definiert worden sind. Eine Gruppe kann jederzeit erneut zusammengestellt werden, indem sie einfach neu gesetzt wird.

```
...
PUT #AD3, #71, "3F 00 01 2B 03"%        ' Kanalgruppe 7: 5 Kanaele
PUT #AD3, #65, "<0><1><29><17>"    ' Kanalgruppe 1: 4 Kanaele
PUT #AD3, #66, FILL$(64,"<4>")     ' Kanalgruppe 2: 64 Kanaele
...
```

Kanalgruppe löschen

Um eine Gruppe zu löschen, wird ein Kanal außerhalb des Wertebereiches gesetzt, z.B. Kanal 99.

```
PUT #AD3, #64, "<99>"              ' Gruppe 0: leer
```

A/D-Wert einzeln und von einer Kanalgruppe lesen

Das Lesen eines A/D-Wertes von einem Kanal geschieht durch einfaches Lesen von der entsprechenden Sekundär-Adresse. Da die Auflösung 12 Bit beträgt, wird für jeden Analogwert ein WORD (2 Byte) benötigt.


```
GET #AD3, #10, 2, wVar            ' liest Wert von Kanal 10
```

Das Lesen mit der Instruktion GET auf den Sekundär-Adressen 64 bis 71 liest von allen Kanälen dieser Gruppe, jedoch maximal soviele Bytes, wie die Variable aufnehmen kann. Die Kanäle werden in der Reihenfolge gelesen, wie sie zuvor in der Gruppe definiert worden sind.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name: ANALOG3.TIG
' stellt EP11 Kanal 0...7 auf Bereich 0...5V
' gruppiert Kanaele 0...7, liest und zeigt Werte an
'--------------------------------------------------------------------
TASK Main                                ' Beginn Task MAIN
  BYTE EVER                              ' fuer Endlosschleife
  WORD W                                 ' Analogwert
  REAL V                                 ' Wert in Volt

 DIR_PORT 8,0                            '

' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  INSTALL DEVICE #4, "ANALOG3.TDD", &   ' Analog-Inputs installieren
    6, &       ' Port fuer Datenbus-Port = default
    8, &       ' Port fuer Steuerleitungen = default
    3, &       ' Pin-Nr. fuer -RD = default
    4, &       ' Pin-Nr. fuer -WR = default
    5, &       ' Pin-Nr. fuer HBEN = default
    7, &       ' Pin-Nr. fuer -CE = default
    22, &      ' speed reduction = default (keine)
    0, &       ' Reservierter Parameter immer 0
    3, &       ' Anzahl Adressleitungen = default
    8, &       ' Port fuer Adressleitungen = default
    0 &        ' Bitposition der Adressleitungen = default

 PRINT #1,"EP11_001 - Test"
  PUT #4, #0, "00 00 00 00 00 00 00 00"% ' ch0...7: alle 0...5V

  FOR EVER = 0 TO 0 STEP 0               ' Endlosschleife
    GET #4, #0, 2, W                     ' Wert aus AD-Wandler lesen
    V = (LTR(W)/4096.0 * 5000.0)/1000.0
    USING "NF<1><1> <2> 0 0 0 0 0 0 0 1.2 0 0 0 0 0 0 0"
    PRINT_USING #1, "<1Bh>A<0><0><F0h>Analog:";V;"V";' Analogwert in Vol
    WAIT_DURATION 100                    ' 100 ms warten
  NEXT
END                                      ' Ende Task MAIN
```

LC-Display, Tastatur und Beep mit LCD1

Der Device-Treiber ‘LCD1’ vereinigt drei der am häufigsten benötigten Geräte:

- Text-LC-Displays, die den populären Hitachi-Controller HD44780 besitzen oder mit diesem kompatibel sind. Das Display kann direkt angeschlossen werden, lediglich für die Kontrasteinstellung wird ggf. ein Potentiometer eingesetzt.
- Tastatur mit Shift- und ctrl- (Strg-)Tasten. Die Tastatur kann bis zu 128 Tasten haben, wobei einzelne Reihen der Tastaturmatrix auch als DIP-Schalter definierbar sind. Die externe Beschaltung besteht aus einigen preiswerten HC-MOS-IC und den Tastaturmatrix-Dioden. Ein Beispiel finden Sie im Schaltplan des Plug & Play Labs. Ein weiteres Beispiel zeigt eine Tastatur in Zusammenhang mit erweiterten Input-Pins im Hardware-Handbuch.
- Tonsignal-Ausgabe: Tastenklick, Glocke und Fehlersignal. Die Signale können durch ESC-Sequenzen verändert werden. Dazu wird ein entsprechendes Ausgabegerät, z.B. der Summer des Plug & Play Labs, an den Pin L42 (Pin-Nr. 35) angeschlossen. Stecken Sie eine Verbindung von L42 nach ‘beep’ auf der 9-poligen Stiftleiste im Analog-Teil.

Die Beschreibung des Device-Treibers LCD1 teilt sich entsprechend in die drei Unterkapitel 'LCD', 'Tastatur' und 'Ton'.

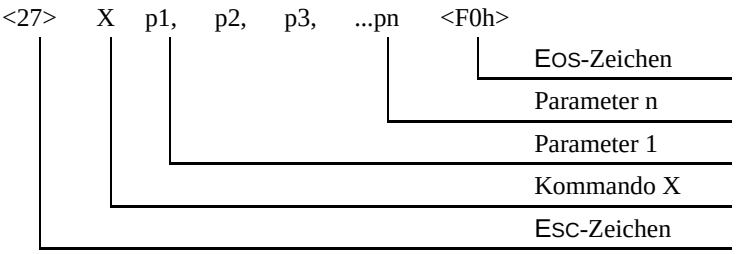
ESC-Kommandos

Kommandos an den LCD1-Device-Treiber können auch mit dem normalen Datenstrom in PRINT oder PUT-Instruktionen in Form von ESC-Sequenzen erfolgen. Die Kommandosequenz fängt dann mit <ESC>-Code an, es folgt ein Kommandozeichen und abhängig vom Kommando eine unterschiedliche Anzahl Parameter. Die Sequenz endet mit einem EOS-Zeichen.

ESC: <1Bh>

EOS: <F0h>

Kommandozeichen: die Groß- und Kleinschreibung wird hier beachtet!



ESC-Sequenzen sind stets in einer PRINT- oder PUT-Instruktion auszugeben. Wenn die Zeilenlänge es zuläßt, können mehrere ESC-Sequenzen in einer Instruktion ausgegeben werden.

Die Argumente der folgenden ESC-Kommandos (x, y, n) sind immer BYTES, wenn nicht anders angegeben. Groß- und Kleinschreibung werden in den Kommandos unterschieden.

LC-Display

Weitere Informationen zu LCD1.TDD, LCD-Teil:

- Typenliste des LCD1
- LC-Display anschließen
- User-Function-Codes
- Steuerzeichen des LC-Displays
- ESC-Kommandos LC-Display
- Cursor positionieren: ESC A
- Sonderzeichensatz einschalten: ESC S
- Sonderzeichensatz laden: ESC L
- Sonderzeichensatz zurücksetzen: ESC R
- Menü auf dem LC-Display: ESC M
- Cursor definieren: ESC c
- LC-Display – Sonderzeichensätze
- Vordefinierte Sonderzeichensätze

Dateiname: LCD1.TDD

INSTALL DEVICE #*D*, "LCD1.TDD" [, *P1*, *P2, ..., P14*]

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

P1 **LCD-Typ:** gibt an, welcher LCD-Modul-Typ angeschlossen ist. Um die in der folgenden Tabelle angegebenen Bezeichnungen (Spalte LCD-Typ) zu verwenden, binden Sie die Datei UFUNC.INC am Anfang Ihres Programms ein.

P2...P14 sind weitere Parameter, die die Standard-Anschlußbelegung des LC-Displays, der Shift-LED der Tastatur sowie der Tonausgabe für 'beep' und Tastenklick verändern. Diese Parameter sind in der nachfolgenden Tabelle näher beschrieben. In der Regel reicht die Angabe eines abweichenden LC-Display-Typs zur Anpassung aus.

In der Regel reicht die Angabe eines abweichenden LC-Display-Typs zur Anpassung aus. Mit den weiteren Parametern, die Sie beim Einbinden des Device-Treibers angeben können, ist die Benutzung der Ports zur Steuerung der externen Komponenten beeinflußbar.

Alle Parameter sind Bytes und können durch die Angabe von 0 oder 0EEH (=238) als Wert unverändert bleiben. Wenn Sie z.B. nur die logische Adresse für Ton von L42 (Standard) auf L87 verlegen wollen, dann geben Sie für die Parameter P1 bis P6 als Wert '0' an. P7 ist die Bitmaske (80H für das höchste Bit) und P8 die Portadresse (8 für den Port 8).

	unverändert lassen	Beschreibung des Parameters
P1	0	LCD-Display-Typ (siehe Extratabelle)
P2	0	Logische BUS-Adresse für LC-Display und Tastatur
P3	0	Bit-MASK für LC-Display-Signal ‘E’
P4	0	Logische Adresse LC-Display-Signal ‘E’
P5	0	Bit-MASK für LC-Display-Signal ‘RS’
P6	0	Logische Adresse LC-Display-Signal ‘RS’
P7	0EEH	Bit-Maske für Ton (Tastenklick, ‘beep’)
P8	0	Logische Adresse für Ton (Tastenklick, ‘beep’)
P9	0EEH	0: Ton wird erzeugt, wenn Bit=0 0FFH: Ton wird erzeugt, wenn Bit=1
P10	0EEH	Bit-Maske für Shift-LED
P11	0	Logische Port-Adresse Shift-LED
P12	0EEH	0: Shift-LED leuchtet, wenn Bit=0 0FFH: Shift-LED leuchtet, wenn Bit=1
P13	0EEh	0: Tastatur wird nicht gescannt <>0: Tastatur wird gescannt (Standard)
P14	-	0: LCD 0FFh: kein LCD

Anmerkung: Wird als Bitmaske 0 angegeben, dann ist der Pin für andere Anwendung frei. Sie sollten die Standard-Anschlußbelegung verwenden, wenn nicht besondere Gründe in Ihrer Schaltung dagegen sprechen. Beachten Sie bitte, daß Änderungen an der Anschlußbelegung leicht zu widersprüchlichen Bedingungen führen können, die manchmal nicht sofort erkennbar sind.

Typenliste des LCD1

Binden Sie die Include-Datei ‘UFUNC.INC’ ein, um die Bezeichner für den LCD-Typ zu verwenden. Paramter ‘LCD-Typ’:

Nr	LCD-Typ	Zeilen x Spalten	Refresh-Mode
1	LCD1_1_8	1 x 8	1:8
2	LCD1_1_12	1 x 12	1:8
3	LCD1_1_16	1 x 16	1:8
4	LCD1_1_20	1 x 20	1:8
5	LCD1_1_40	1 x 40	1:8
6	LCD1_2_8	2 x 8	1:8
7	LCD1_2_12	2 x 12	1:8
8	LCD1_2_16	2 x 16	1:8
9	LCD1_2_20	2 x 20	1:8
10	LCD1_2_40	2 x 40	1:8
11	LCD1_4_20	4 x 20	1:8
12	LCD2_2_8	2 x 8	1:16
13	LCD2_2_12	2 x 12	1:16
14	LCD2_2_16	2 x 16	1:16
15	LCD2_2_20	2 x 20	1:16
16	LCD2_2_40	2 x 40	1:16
17	LCD2_4_20	4 x 20 (default)	1:16
18	LCD2_2_24	2 x 24	1:16

Bei fehlender Angabe von **LCD-Typ** wird der Typ 17 angenommen.

Beispiel: setze LC-Display-Typ mit 2 Zeilen à 40 Zeichen (‘LCD1_2_40’ ist definiert in der Include-Datei ‘DEFINE_A.INC’):

```
INSTALL_DEVICE "LCD1.TDD", LCD1_2_40
```

Beispiel: lege den Sound-Pin auf L87. Lasse alle anderen Parameter unverändert:

```
INSTALL_DEVICE "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
```


Der Device-Treiber LCD1 läßt sich auch mehrfach einbinden, um zum Beispiel mehrere LCD oder auch mehrere Tastaturen anzuschließen. Dabei wird nach wie vor der gleiche Datenbus verwendet, jedoch weitere Pins als Steuerleitungen. Je nach Bedarf kann sowohl die Tastatur als auch das LCD abgeschaltet werden.

Beispiel: schalte Tastaturscanning ab. Lasse alle anderen Parameter unverändert:

```
INSTALL_DEVICE "LCD1.TDD", 0, 0, 0, 0, 0, 0, &
0eeh, 0, 0eeh, 0eeh, 0, 0eeh, 0
```

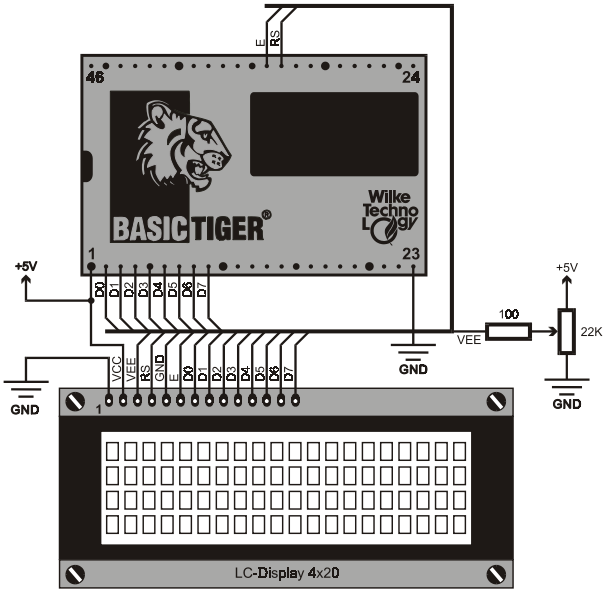
Beispiel: schalte LCD ab. Lasse alle anderen Parameter unverändert:

```
INSTALL_DEVICE "LCD1.TDD", 0, 0, 0, 0, 0, 0, &
0eeh, 0, 0eeh, 0eeh, 0, 0eeh, 0FFh, 0FFh
```

LC-Display anschließen

Ein LC-Display benötigt vom BASIC-Tiger®-Modul den Datenbus und zwei weitere I/O-Leitungen: ‘Enable-LCD’ (E) und ‘Register select’ (RS. Standard-Anschlußbelegung:

LCD-Pin-Funktion	Pin-Bezeichnung	Pin-Nr. Modul A	Pin-Nr. Tiny Tiger
D0...D7	L60...L67	2...9	1...8
E (Enable)	L36	33	32
RS (Register Select)	L37	34	33



Da LC-Displays eigene Controller besitzen, können Sie unabhängig von der steuernden Umgebung abstürzen oder außer Tritt geraten. Das passiert durch ststische Entladung in der Nähe des LC-Displays oder durch Einstreuung in das Datenkabel, insbesondere wenn dieses sehr lang ist. Bei langen Datenkabeln ist die Masseverbindung erfahrungsgemäß besonders kritisch. Als Abhilfe für solche Fälle können Sie vorsehen, daß das LC-Display vom steuernden Modul zurückgesetzt wird. Dazu schicken Sie ein Kommando (UFCO_LCD_RESET) an den Device-Treiber:

User-Function-Codes (LCD)

User-Function-Codes für Abfragen (Instruktion GET):

Nr	Symbol Prefix UFCI_	Beschreibung
33	UFCI_OBU_FILL	Füllstand des LCD-Ausgangspuffers (Byte)
34	UFCI_OBU_FREE	freier Platz im LCD-Ausgangspuffer (Byte)
35	UFCI_OBU_VOL	Größe des LCD-Ausgangspuffers (Byte)
65	UFCI_LAST_ERRC	letzter Error-Code
99	UFCI_DEV_VERS	Version des Treibers

User-Function-Codes des LCD1.TDD zum Setzen von Parametern (PUT):

Nr	Symbol Prefix: UFCO_	Beschreibung
33	UFCO_OBU_ERASE	Ausgangspuffer löschen
176	UFCO_LCD_RESET	Initialisiere LCD neu

Steuerzeichen des LC-Displays

Steuerzeichen werden direkt auf das LCD-Device geschrieben, ohne ESC und ohne EOS.

CLR	<01>	löscht LCD-Bildschirm
HOME	<02>	setzt den Cursor in die linke obere Ecke
FS	<05>	Cursor 1 Position nach rechts
BS	<08>	Cursor 1 Position nach links
LF, DO	<0Ah>	Cursor 1 Position nach unten
UP	<0Bh>	Cursor 1 Position nach oben
FF	<0Ch>	Form Feed, Seitenvorschub
CR	<0Dh>	Carriage Return, Wagenrücklauf

```
PRINT #LCD, "<1>";
```

löscht den LCD-Bildschirm und bringt den Cursor in die 'home'-Position (X=0, Y=0). Es kann direkt auch weiterer Text folgen:

```
PRINT #LCD, "<1>Hello World"
```

ESC-Kommandos LC-Display:

Übersicht:

ESC-Zeichen	Beschreibung
Esc, A, x, y, Eos	Absolute Cursor-Adressierung
Esc, S, n, Eos	Sonderzeichensatz n einschalten n=0...15
Esc, L, n, Daten, Eos	Sonderzeichensatz n laden n=0...15 64 Bytes Daten Beispiel: S. 372
Esc, R, n, Eos	Sonderzeichensatz n zurücksetzen wenn n>15: alle zurücksetzen
Esc, M, n, Eos Tz String	Menü-Auswahl n n ist Index im Menü Tz ist das Trennzeichen Beispiel: S. 68
Esc, c, n, Eos	Cursor definieren n=0: Cursor aus n=1: Cursor ein n>1: Cursor ein + blinken

Cursor positionieren: ESC A

PRINT #D, "<1Bh>A"; CHR$(x); CHR$(y); "<F0h>";

Positioniert den Cursor absolut auf dem Display.

D	ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.
x	x-Koordinate (Spalte), auf die der Cursor positioniert werden soll.
y	y-Koordinate (Zeile), auf die der Cursor positioniert werden soll.

Die Zählung der Zeilen und Spalten beginnt mit 0. Der mögliche Wertebereich hängt vom verwendeten LC-Display ab. Zu große Werte für **x** und **y** werden auf 0 gesetzt.

Folgende Zählweise ergibt sich bei dem 4x20-LC-Display:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0																				
1																				
2																				
3																				

Programmbeispiel:

```
'---------------------------------------------------------------------
' Name: GOTOXY.TIG
'---------------------------------------------------------------------
' Ein Stern ('*') wandert regelmaessig von links nach rechts
' ueber das Display und wieder zurueck.
'---------------------------------------------------------------------
TASK MAIN                               ' Beginn Task MAIN
  BYTE X                                ' BYTE-Variable deklarieren
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  PRINT #1, "<1Bh>c<0><F0h>";           ' Cursor ausschalten
  PRINT #1, "<1>";                      ' Bildschirm loeschen
  LOOP 9999999                          ' sehr viele Loops
    FOR X = 0 TO 19                     ' Schleife ueber alle Spalten
      CALL STARXY (X,1)                 ' "*" an Zeile 1, Spalte X
    NEXT                                ' naechste Spalte
    FOR X = 18 TO 1 STEP -1             ' Schleife ueber alle Spalten
      CALL STARXY (X,1)                 ' "*" an Zeile 1, Spalte X
    NEXT                                ' naechste Spalte
  ENDLOOP                               '
END                                     ' Ende Task MAIN

'---------------------------------------------------------------------
' Sub: Cursor an Position (x, y), "*" ausgeben und loeschen nach 200ms
'---------------------------------------------------------------------
SUB STARXY (BYTE X,Y)                   ' Beginn Subroutine STARXY
  PRINT #1, "<1Bh>A"; CHR$(X);&         ' Ausgabe "*" an Position
            CHR$(Y); "<F0h>*";          ' Spalte X, Zeile Y
  WAIT DURATION 200                     ' 200 ms warten
  PRINT #1, "<1Bh>A"; CHR$(X);&         ' Ausgabe "*" an Position
            CHR$(Y); "<F0h> ";          ' Spalte X, Zeile Y
END                                     ' Ende Subroutine STARXY
```

Sonderzeichensatz einschalten: ESC S

PRINT #D, “<1BH>S“; CHR$(n); “<F0H>“;

Schaltet einen Sonderzeichensatz ein.

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

n Nummer des Sonderzeichensatzes (zwischen 0 und 15).

Es gibt 16 Sonderzeichensätze, von denen stets genau einer aktiv sein kann und das Erscheinungsbild der 8 Sonderzeichen auf dem Display bestimmt. Das bedeutet, daß alle sichtbaren Sonderzeichen immer aus ein und demselben Sonderzeichensatz stammen.

Zeichen anderer Sonderzeichensätze, die gerade auf dem Display sichtbar sind, werden sofort entsprechend dem neuen Zeichensatz angezeigt.

Programmbeispiel:

```
'-----------------------------------------------------------------------
' Name: SEL_FNT.TIG
'-----------------------------------------------------------------------
' Zeigt wiederholt alle 16 Sonderzeichensaetze der Reihe nach
' fuer jeweils 2 Sekunden an.
'-----------------------------------------------------------------------
TASK MAIN                               ' Beginn Task MAIN
  BYTE X                                ' BYTE-Variable deklarieren
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  PRINT #1, "<1Bh>c<0><F0h>";           ' Cursor ausschalten
  PRINT #1, "<1>";                      ' Bildschirm loeschen
  FOR X=0 TO 7                          ' Schleife ueber alle Zeichen
    PRINT #1, CHR$(80h + X); "-";       ' Ausgabe des Zeichens
  NEXT                                  ' naechstes Zeichen
  LOOP 9999999                          ' sehr viele Loops
    FOR X=0 TO 15                       ' Schleife ueber alle Fonts
      PRINT #1,"<1Bh>S";CHR$(X);"<F0h>";' Font Nr. X einschalten
      WAIT DURATION 2000                ' 2 Sek warten
    NEXT                                ' naechster Font
  ENDLOOP                               '
END                                     ' Ende Task MAIN
```

Näheres zu den Sonderzeichen der LC-Displays finden Sie unter ‚LC-Display - Sonderzeichensätze’ auf den Seiten 52f.

Sonderzeichensatz laden: ESC L

PRINT #D, “<1BH>L“; CHR$(n); Datensatz; “<F0H>“;

Definiert bzw. lädt einen Sonderzeichensatz mit eigenen Daten.

D	ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.
n	Nummer des Sonderzeichensatzes (zwischen 0 und 15)
Datensatz	64 Bytes, die den Aufbau des Fonts bestimmen

Mit dieser ESC-Sequenz wird ein eigener Sonderzeichensatz geladen. Eigene Sonderzeichen werden dann benötigt, wenn ein besonderes Symbol dargestellt werden soll, oder wenn in fremdsprachlichen Textausgaben Zeichen nicht mit dem Standardzeichensatz darstellbar sind. Der Sonderzeichensatz wird in das LC-Display übertragen und dort gespeichert. Der Zeichensatz muß nach jedem Power-up neu geladen werden.

Programmbeispiel:

```
'---------------------------------------------------------------------
' Name: LOAD_FNT.TIG
'---------------------------------------------------------------------
' Die Zeichen des Sonderzeichensatz 7 werden auf dem Display an-
' gezeigt. In einer Schleife wird alle 2 Sekunden dieser Sonder-
' zeichensatz mit benutzerdefinierten Zeichen geladen und danach
' wieder auf den "Standard"-Sonderzeichensatz zurueckgeschaltet.
' Die Anzeige auf dem Display aendert sich entsprechend.
'---------------------------------------------------------------------
TASK MAIN                                ' Beginn Task MAIN
  STRING CSET$                           ' STRING-Variable deklarieren
  BYTE   X                               ' BYTE-Variable deklarieren
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8

  CSET$ = "&                             ' Daten fuer USER-Font
04 04 04 04 00 00 00 00&                 ' Daten Zeichen 0
01 02 02 04 00 00 00 00&                 ' Daten Zeichen 1
00 00 00 07 00 00 00 00&                 ' Daten Zeichen 2
00 00 00 04 02 02 01 00&                 ' Daten Zeichen 3
00 00 00 04 04 04 04 00&                 ' Daten Zeichen 4
00 00 00 04 08 08 10 00&                 ' Daten Zeichen 5
00 00 00 1C 00 00 00 00&                 ' Daten Zeichen 6
10 08 08 04 00 00 00 00"%                ' Daten Zeichen 7

  PRINT #1, "<1Bh>c<0><F0h>";            ' Cursor ausschalten
  PRINT #1, "<01h>";                     ' Bildschirm loeschen
  FOR X=0 TO 7                           ' Schleife ueber alle Zeichen
    PRINT #1, CHR$(80h + X); " ";        ' Zeichen Sonder-Font ausgeben
  NEXT                                   ' naechstes Zeichen
  PRINT #1, "<1Bh>S<7><F0h>";            ' Sonder-Zeichensatz Nr.7 ein
  LOOP 9999999                           ' sehr viele Loops
    WAIT DURATION 2000                   ' 2 sek. warten
    PRINT #1, "<1Bh>L<7>";CSET$;"<F0h>";' Sonder-Font benutzerdefiniert
    WAIT DURATION 2000                   ' 2 sek. warten
    PRINT #1, "<1Bh>R<7><F0h>";          ' Sonder-Font wieder Standard
  ENDLOOP                                '
END                                      ' Ende Task MAIN
```

Näheres zu den Sonderzeichen der LC-Displays finden Sie unter ‚LC-Display - Sonderzeichensätze' auf den Seiten 52f.

Siehe auch: Esc-Kommando S (Zeichensatz einschalten) und Esc-Kommando R (Zeichensatz zurücksetzen).

Sonderzeichensatz zurücksetzen: ESC R

PRINT #D, “<1BH>R“; CHR$(n); “<F0H>“;

Entfernt einen zuvor selbstdefinierten Sonderzeichensatz und setzt ihn zurück auf den Default-Sonderzeichensatz, wie im Device-Treiber-Handbuch auf den Seiten 73ff abgedruckt.

D	ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.
n	Nummer des Sonderzeichensatzes (zwischen 0 und 15)

Wird für **n** ein Wert größer als die max. Fontnummer (>15) angegeben, werden alle Sonderzeichensätze auf ihre Defaults zurückgesetzt.

Programmbeispiel:

```
'---------------------------------------------------------------------
' Name: RES_FNT.TIG
'---------------------------------------------------------------------
' Die Zeichen des Sonderzeichensatz 7 werden auf dem Display an-
' gezeigt. In einer Schleife wird alle 2 Sekunden dieser Sonder-
' zeichensatz mit benutzerdefinierten Zeichen geladen und danach
' wieder auf den "Standard"-Sonderzeichensatz zurueckgeschaltet.
' Die Anzeige auf dem Display aendert sich entsprechend.
'---------------------------------------------------------------------
TASK MAIN                                ' Beginn Task MAIN
  STRING CSET$                           ' STRING-Variable deklarieren
  BYTE   X                               ' BYTE-Variable deklarieren
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8

  CSET$ = "&                             ' Daten fuer USER-Font
04 04 04 04 00 00 00 00&                 ' Daten Zeichen 0
01 02 02 04 00 00 00 00&                 ' Daten Zeichen 1
00 00 00 07 00 00 00 00&                 ' Daten Zeichen 2
00 00 00 04 02 02 01 00&                 ' Daten Zeichen 3
00 00 00 04 04 04 04 00&                 ' Daten Zeichen 4
00 00 00 04 08 08 10 00&                 ' Daten Zeichen 5
00 00 00 1C 00 00 00 00&                 ' Daten Zeichen 6
10 08 08 04 00 00 00 00"%                ' Daten Zeichen 7

  PRINT #1, "<1Bh>c<0><F0h>";            ' Cursor ausschalten
  PRINT #1, "<01h>";                     ' Bildschirm loeschen
  FOR X=0 TO 7                           ' Schleife ueber alle Zeichen
    PRINT #1, CHR$(80h + X); " ";        ' Zeichen Sonder-Font ausgeben
  NEXT                                   ' naechstes Zeichen
  PRINT #1, "<1Bh>S<7><F0h>";            ' Sonder-Zeichensatz Nr.7 ein
  LOOP 9999999                           ' sehr viele Loops
    WAIT DURATION 2000                   ' 2 sek. warten
    PRINT #1, "<1Bh>L<7>";CSET$;"<F0h>";' Sonder-Font benutzerdefiniert
    WAIT DURATION 2000                   ' 2 sek. warten
    PRINT #1, "<1Bh>R<7><F0h>";          ' Sonder-Font wieder Standard
  ENDLOOP                                '
END                                      ' Ende Task MAIN
```

Näheres zu den Sonderzeichen der LC-Displays finden Sie unter ‚LC-Display - Sonderzeichensätze' auf den Seiten 52f.

Siehe auch Esc-Kommando S (Zeichensatz einschalten) und Esc-Kommando L (Zeichensatz laden).

Menü auf dem LC-Display: ESC M

PRINT #D, “<1BH>M“; CHR$(n); “<F0H>“; Menue$;

Das Kommando ‘M’ erlaubt es, aus einem großen String gezielt einen Teilstring durch Index zu selektieren. Dies kann z.B. dazu verwendet werden eine Menü-Auswahl anzubieten.

D	ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.
n	Index des aktuell anzuzeigenden Menüpunktes
Menue$	String, der die einzelnen darzustellenden Menue-Elemente enthält. Dieser String enthält zunächst das Trennzeichen (im Beispiel “:“), gefolgt von den Menü-Elementen. Die einzelnen Elemente sind durch das Trennzeichen getrennt, das erste Element wird durch Index=0 selektiert, die nachfolgenden durch den entsprechenden Index. Das Ende des Auswahlstrings wird durch das doppelte Auftreten des Trennzeichens markiert.

ACHTUNG: Wird der Index größer als die Anzahl der Elemente im String, werden leere Elemente ausgegeben! Im Beispielprogramm sind 4 Einträge (Index 0 bis 3) im String. Bei einem Index von n=5 würde hinter “Ihre Wahl: “ ein Leerstring gedruckt.

Programmbeispiel:

```
'-------------------------------------------------------------------------
' Name: MENU.TIG
'-------------------------------------------------------------------------
' Mit den Tasten <Up> und <Down> wird der aktuelle Index fuer den
' auszuwaehlenden Menuepunkt erhoeht oder verringert und der neue
' Menuepunkt angezeigt, mit <Return> wird der Index angezeigt und
' das Programm beendet.
'-------------------------------------------------------------------------
#INCLUDE KEYB_PP.INC                    ' Dt. Tastatur Plug & Play-Lab

TASK MAIN                               ' Beginn Task MAIN
  STRING KEY$                           ' STRING-Variable deklarieren
  BYTE   M, N, FLAG                     ' BYTE-Variablen deklarieren
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  CALL INIT_KEYB (1)                    ' Init: Tastatur deutsch

  M = 0                                 ' Menue-Index = 0
  FLAG = 1                              ' Flagge = 1 (TRUE)
  WHILE FLAG = 1                        ' Solange Flagge = TRUE
    CALL SHOWMENU (M)                   ' Menue anzeigen
    FOR N = 0 TO 0 STEP 0               ' Endlosschleife bis N=1(GET!)
      RELEASE_TASK                      ' Rest der Task-Zeit freigeben
      GET #1, #0, #1, 1, N              ' N=Zeichen im Tastatur-Buffer
    NEXT                                ' Ende Endlosschleife
    GET #1, 1, KEY$                     ' Tastatur-Buffer auslesen
    SWITCH KEY$                         ' Bedingte Auswahl nach Inhalt
      CASE _CR:                         ' KEY$ = Return:
        FLAG = 0                        ' Flagge = FALSE
      CASE _UP:                         ' KEY$ = Cursor Up:
        M = (M - 1) BITAND 3            ' Index verringern (min. 0)
      CASE _DO:                         ' KEY$ = Cursor Down:
        M = (M + 1) BITAND 3            ' Index erhoehen (max. 3)
    ENDSWITCH                           ' Ende der bedingten Auswahl
  ENDWHILE                              '
  PRINT #1, _CLR;                       ' Display loeschen
  PRINT #1, "Menu index is:"; M         ' Ausgabe auf LC-Display
END                                     ' Ende Task MAIN

'-------------------------------------------------------------------------
' Subroutine: Gibt aktuell gewaehlten Menuepunkt auf LC-Display aus
'-------------------------------------------------------------------------
SUB SHOWMENU (BYTE I)                   ' Beginn Subroutine SHOWMENU
  PRINT #1, _CLR;                       ' Display loeschen
  PRINT #1, "your choice: ";            ' Ausgabe Menupunkt Nr. I
  PRINT #1, "<1Bh>M";CHR$(I);&          '
            "<F0h>:AUTO   :MANUAL :ALARM  :STOP   ::";
END                                     ' Ende Subroutine SHOWMENU
```

Cursor definieren: ESC c

PRINT #D, “<1Bh>c“; CHR$(n); “<F0h>“;

Definiert das Aussehen des Cursors auf dem Display.

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

n Bestimmt die Form des Cursors
n=0: Cursor ist ausgeschaltet
n=1: Cursor ist eingeschaltet
n>1: Cursor ist eingeschaltet und blinkt

Programmbeispiel:

```
'---------------------------------------------------------------------
' Name: CURSOR.TIG
'---------------------------------------------------------------------
TASK MAIN                               ' Beginn Task MAIN
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8

  LOOP 9999999                          ' sehr viele Loops
    FOR X = 0 TO 2                      ' Parameter-Werte von 0 bis 2
      PRINT #1,"<1Bh>c";CHR$(X);"<F0h>";' Cursor-Modus einstellen
      SWITCHI X                         ' Index-Auswahl nach X
        CASE 0:                         ' Falls X=0:
          PRINT #1, "<1>Cursor off";    ' Ausgabe fuer "Cursor aus"
        CASE 1:                         ' Falls X=1:
          PRINT #1, "<1>Cursor on";     ' Ausgabe fuer "Cursor ein"
        CASE 2:                         ' Falls X=2:
          PRINT #1, "<1>Cursor blinks"; ' Ausgabe fuer "Cursor blinkt"
      ENDSWITCH                         ' Ende der Index-Auswahl
      WAIT DURATION 3000                ' 3 Sek warten
    NEXT                                ' naechster Parameter-Wert
  ENDLOOP                               '
END                                     ' Ende Task MAIN
```

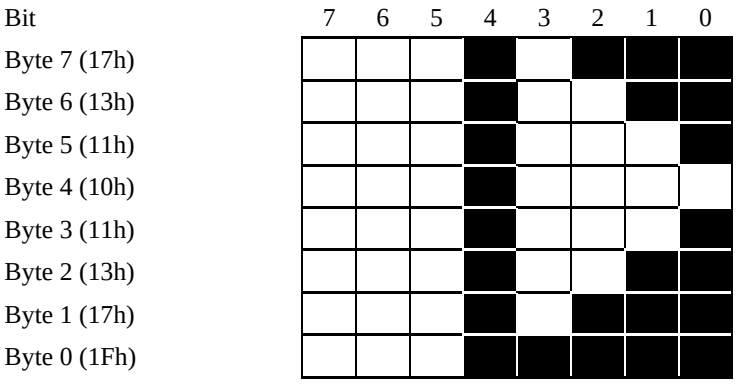
LC-Display - Sonderzeichensätze

Viele Applikationen benötigen Sonderzeichen auf dem LC-Display. In Deutschland werden die Umlaute verwendet, die im Standardzeichensatz nicht enthalten sind.

Der LCD-Device-Treiber unterstützt mit seinen ESC-Kommandosequenzen die Programmierung der Sonderzeichensätze des LC-Displays.

Jeder Sonderzeichensatz besteht aus 8 Zeichen, und jedes Zeichen besteht aus 8 Byte, deren niederwertigsten 5 Bits zur Darstellung verwendet werden. Jeder Zeichensatz besteht also 64 Byte, die bei der Definition gesendet werden.

Es gibt 16 Sonderzeichensätze, von denen stets genau einer aktiv sein kann und das Erscheinungsbild der 8 Sonderzeichen auf dem Display bestimmt. Das bedeutet, daß alle sichtbaren Sonderzeichen immer aus ein und demselben Sonderzeichensatz stammen.



Die Codes der Sonderzeichen beginnen bei 80h für Sonderzeichensatz 0.

Sonderzeichensatz	Codes	Sonderzeichensatz	Codes
0	80h...87h	8	C0h...C7h
1	88h...8Fh	9	C8h...CFh
2	90h...97h	10	D0h...D7h
3	98h...9Fh	11	D8h...DFh
4	A0h...A7h	12	E0h...E7h
5	A8h...AFh	13	E8h...EFh
6	B0h...B7h	14	F0h...F7h
7	B8h...BFh	15	F8h...FFh

Die Ausgabe schon eines Zeichens aus einem Sonderzeichensatz aktiviert automatisch genau diesen Zeichensatz. Zeichen anderer Sonderzeichensätze, die gerade auf dem Display sichtbar sind, werden sofort entsprechend dem neuen Zeichensatz angezeigt.

Alle 16 Sonderzeichensätze sind im LCD1-Treiber mit sinnvollen Zeichen vorbesetzt.

Im folgenden Beispiel werden Zeichen des Sonderzeichensatzes 0 ausgegeben (äöüÄÖÜ..). Nach einer kurzen Pause ändert sich die Darstellung, weil Zeichen eines anderen Sonderzeichensatzes ausgegeben werden.

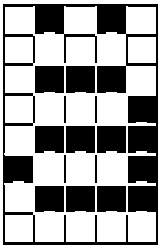
Programmbeispiel:

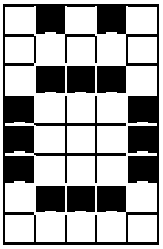
```
'------------------------------------------------------------------------
' Name: SPECCHR1.TIG
'------------------------------------------------------------------------
TASK MAIN                                ' Beginn Task MAIN
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  PRINT #1, "80818283"%                  ' Ausgabe aus Sonder-Font Nr.0
  WAIT_DURATION 2000                     ' 2 Sek warten
  PRINT #1, "8C8D8E8F"%                  ' Ausgabe aus Sonder-Font Nr.1
END                                      ' Ende Task MAIN
```

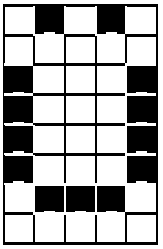
Vordefinierte Sonderzeichensätze

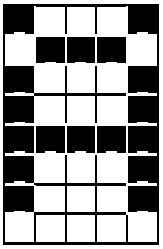
Auf den folgenden Seiten sind alle 16 Sonderzeichensätze des LCD1-Treibers dargestellt. Sie können sich diese Zeichensätze auch mit dem Programm ‚**LCD_SPCC.TIG'** auf Seite 372 ansehen.

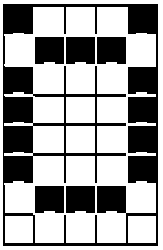
Sonderzeichensatz 0 (80h...87h)

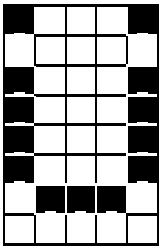

80h

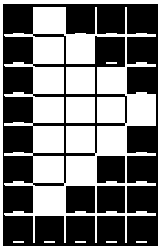

81h

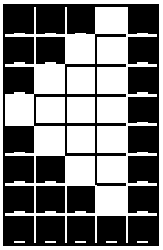

82h


83h

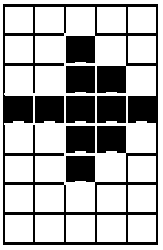

84h

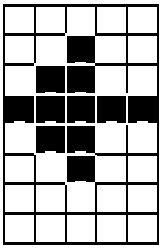

85h

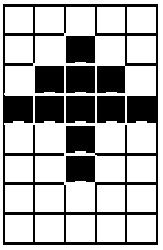

86h

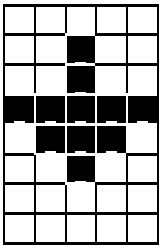

87h

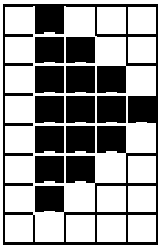
Sonderzeichensatz 1 (88h...8Fh)

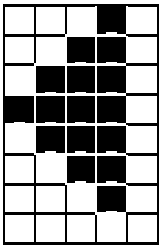

88h

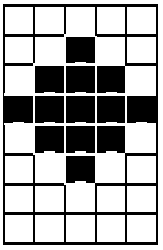

89h

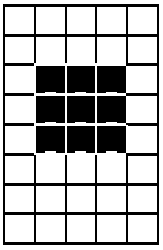

8Ah


8Bh

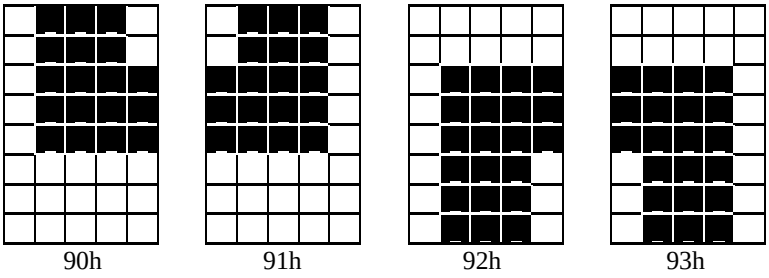

8Ch


8Dh

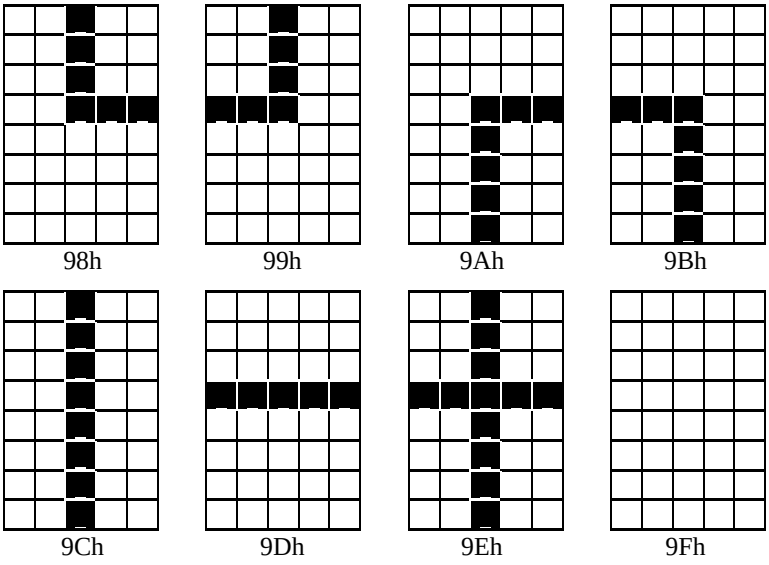

8Eh


8Fh

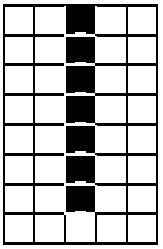
Sonderzeichensatz 2 (90h...97h)

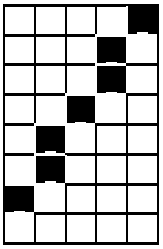


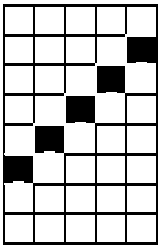
Sonderzeichensatz 3 (98h...9Fh)

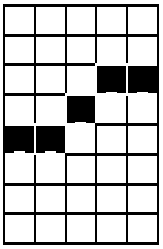


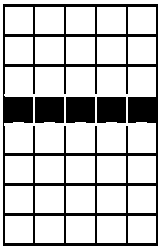
Sonderzeichensatz 4 (A0h...A7h)

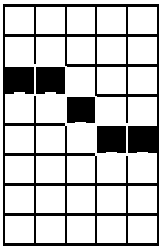

A0h

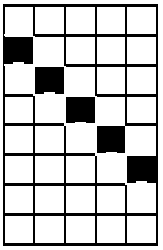

A1h

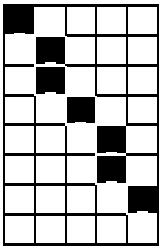

A2h


A3h

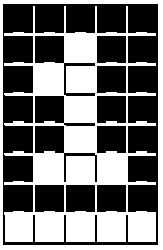

A4h

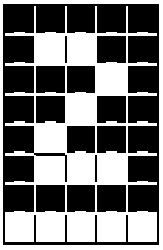

A5h

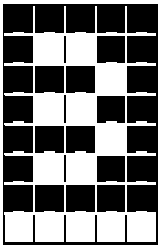

A6h

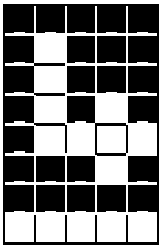

A7h

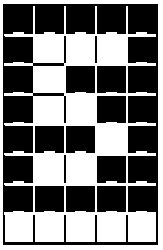
Sonderzeichensatz 5 (A8h...AFh)

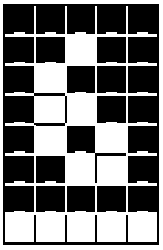

A8h

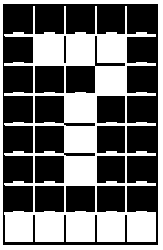

A9h

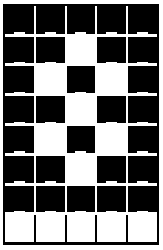

AAh


ABh

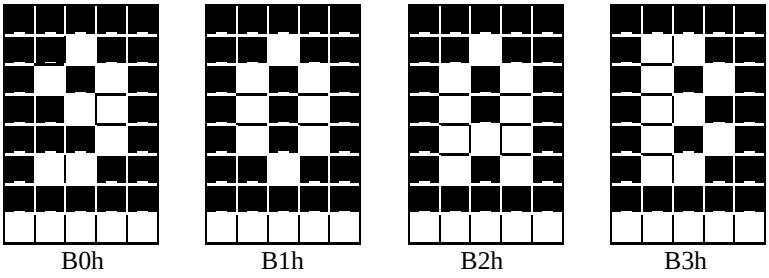

ACh


ADh

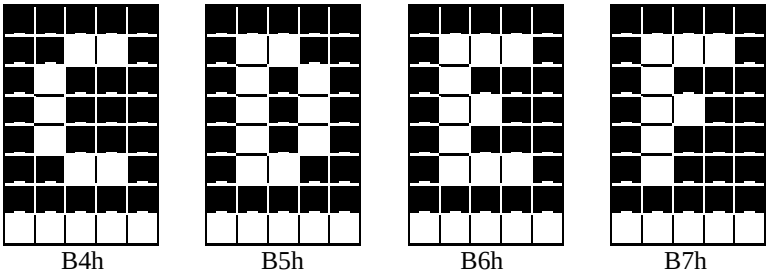

AEh


AFh

Sonderzeichensatz 6 (B0h...B7h)

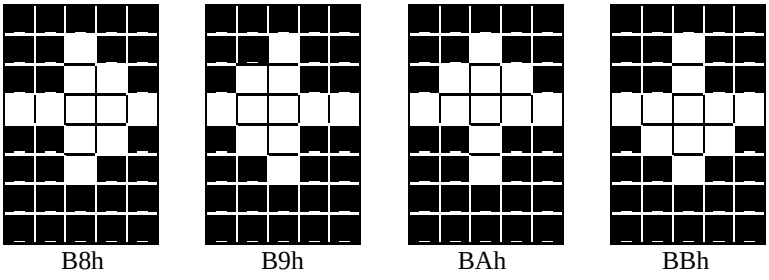


B0h B1h B2h B3h

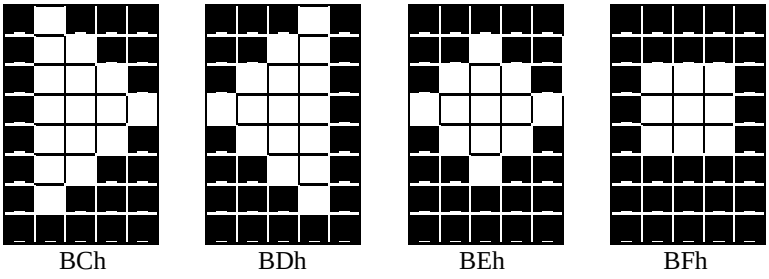


B4h B5h B6h B7h

Sonderzeichensatz 7 (B8h...BFh)

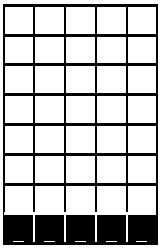


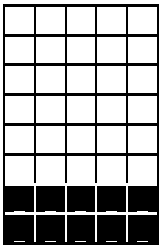
B8h B9h BAh BBh

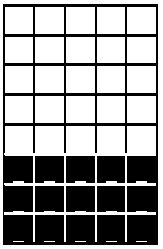


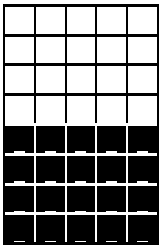
BCh BDh BEh BFh

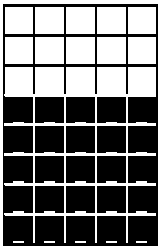
Sonderzeichensatz 8 (C0h...C7h)

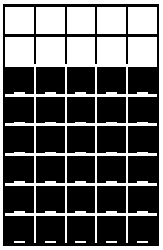

C0h

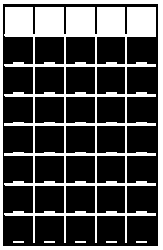

C1h

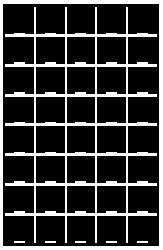

C2h


C3h

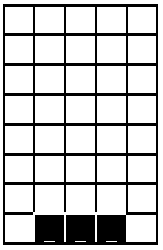

C4h

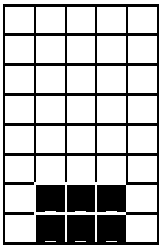

C5h

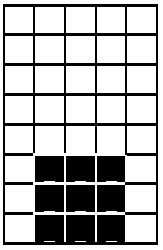

C6h

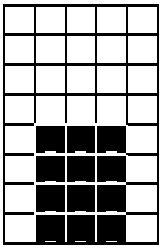

C7h

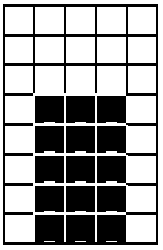
Sonderzeichensatz 9 (C8h...CFh)

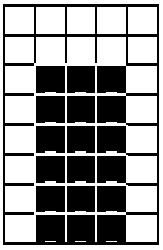

C8h

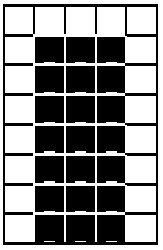

C9h

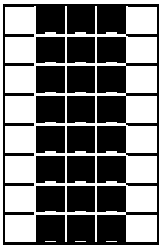

CAh


CBh

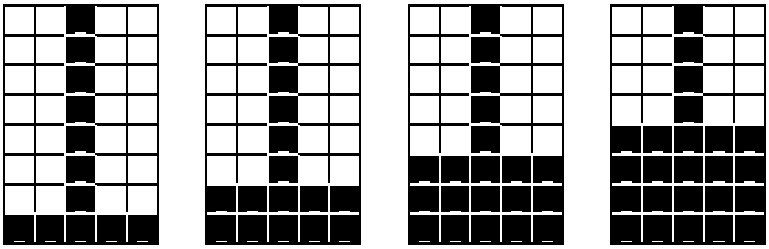

CCh


CDh

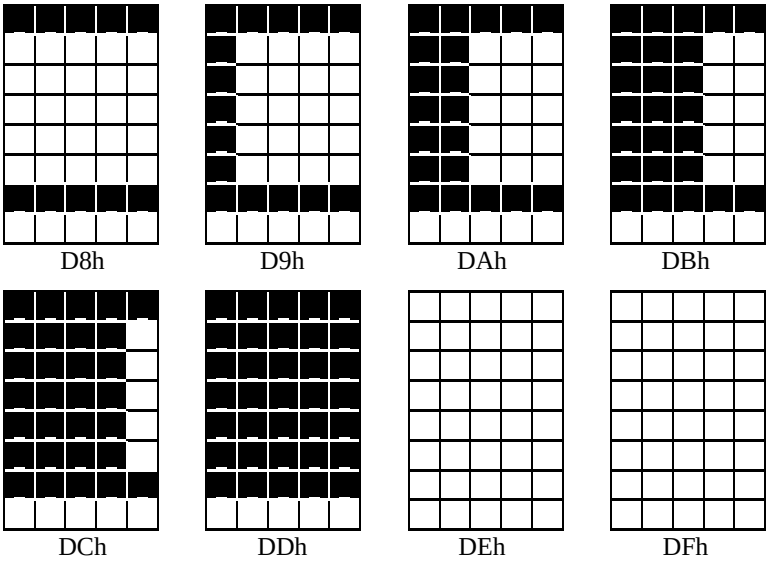

CEh


CFh

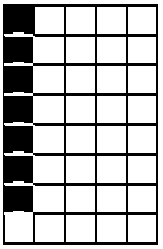
Sonderzeichensatz 10 (D0h...D8h)

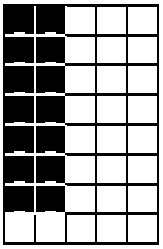


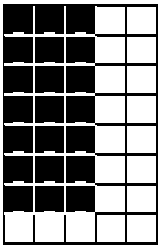
Sonderzeichensatz 11 (D8h...DFh)

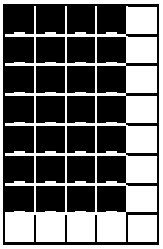


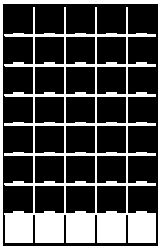
Sonderzeichensatz 12 (E0h...E7h)

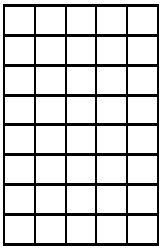

E0h

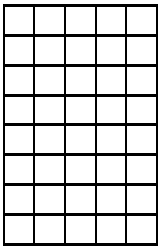

E1h

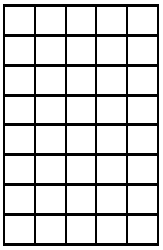

E2h


E3h

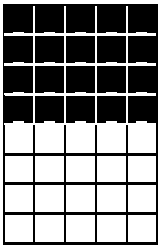

E4h

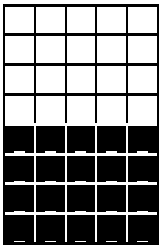

E5h

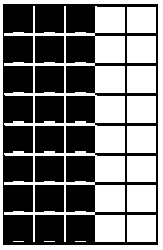

E6h

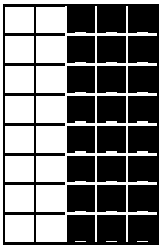

E7h

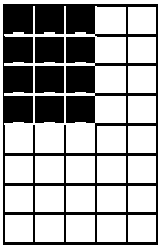
Sonderzeichensatz 13 (E8h...EFh)

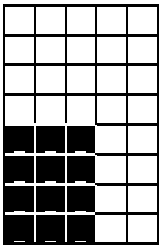

E8h

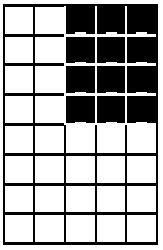

E9h

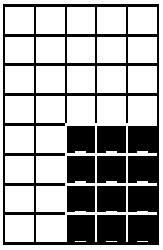

EAh


EBh

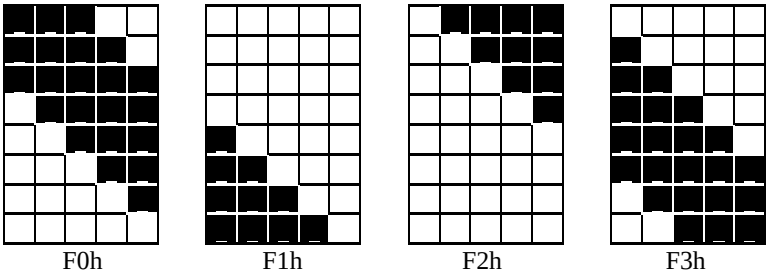

ECh


EDh

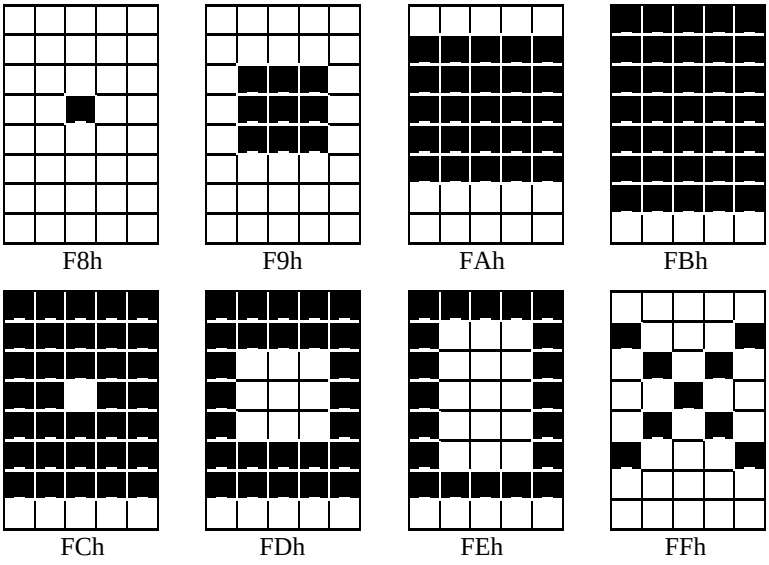

EEh


EFh

Sonderzeichensatz 14 (F0h...F7h)



Sonderzeichensatz 15 (F8h...FFh)



Leere Seite

LCD1-Tastatur

Es können sehr einfach Tastaturen mit bis zu 128 Tasten angeschlossen werden. Die Größe der Tastatur, die Tastencodes und Attribute wie ‘Shift’, ‘ctrl’, ‘shift-lock’ können angepasst werden. Einzelne Reihen der Tastaturmatrix sind auch als DIP-Schalter definierbar.

Weitere Informationen zu LCD1.TDD, Teil Tastatur:

- User-Function-Codes
- ESC-Kommandos Tastatur
- Tastatur-Auto-Repeat: ESC r
- Tasten-Codes: ESC Z bzw. ESC z
- Tastenattribute: ESC a
- DIP-Schalter: ESC D
- Einlesen von DIP-Schaltern
- Scan-Adressen: ESC k

Die Tastatur ist ein Teil der erweiterten Inputs. Die Tastatur benötigt den Datenbus L60...L67 und zwei weitere I/O-Leitungen ‘aclk’ und ‘keyb’ (‘keyb’=’In-enable’). Auf dem Plug & Play Lab müssen die Jumper auf J22 stecken. Da der Anschluß der Tastatur in engem funktionellem Zusammenhang mit den erweiterten Ein- und Ausgängen steht, finden Sie nähere Informationen zur Hardware der Tastatur im Hardware-Handbuch.

Tastatureingaben werden in einem Puffer gespeichert. Größe, Füllstand oder verbleibender Platz des Puffers können mit Hilfe der User-Function-Codes abgefragt werden (Seiten 16ff).

User-Function-Codes (Tastatur)

User-Function-Codes für Abfragen (Instruktion GET):

Nr	Symbol Prefix UFCI_	Beschreibung
1	UFCI_IBU_FILL	Füllstand des Tastaturpuffers (Byte)
2	UFCI_IBU_FREE	freier Platz im Tastaturpuffer (Byte)
3	UFCI_IBU_VOL	Größe des Tastaturpuffers (Byte)
65	UFCI_LAST_ERRC	letzter Error-Code
99	UFCI_DEV_VERS	Version des Treibers

User-Function-Codes des LCD1.TDD zum Setzen von Parametern (PUT):

Wert	Symbol Prefix: UFCO_	Beschreibung
1	UFCO_IBU_ERASE	Tastaturpuffer löschen
176	UFCO_LCD_RESET	Initialisiere LCD1-Treiber neu
128	UFCO_SET_ISEP	setze Separatoren für INPUT
129	UFCO_RES_ISEP	lösche Separatoren für INPUT
130	UFCO_KB_ECHO	erzeugt Echo auf LCD (JA/NEIN)

Für die Instruktion INPUT gelten standardmäßig KOMMA und RETURN als Separatorzeichen, die eine Eingabe abschließen. Mit Hilfe des User-Function-Codes UFCO_SET_ISEP lassen sich die Separatorzeichen verändern. Bevor neue Zeichen gesetzt werden, können die bereits gesetzten Zeichen gelöscht werden. Die zu setzenden oder zu löschenden Zeichen werden als Code-Bereiche angegeben:

PUT #D, #C, #UFCO_SET_ISEP, *Startcode*, *Endcode*, *Startcode*, *Endcode*

Wenn Sie die Standard-Separatoren löschen, jedoch keine neuen setzen, wird eine INPUT-Instruktion erst beendet, wenn der Input-Puffer voll ist.

!

Beispiel: setze neuen Separator LINE-FEED für die Instruktion INPUT auf dem Device-Treiber LCD1:

```
PUT #2,#0, #UFCO_RES_ISEP, 0, 255      ' loesche alle Separatoren
PUT #2,#0, #UFCO_SET_ISEP, 10, 10      ' setze Line-Feed als Separator
```

Beispiel: setze alle Steuerzeichen sowie Zeichen ab 7Fh als Separatorzeichen für die Instruktion INPUT auf dem Device-Treiber LCD1:

```
PUT #2, #0, #UFCO_RES_ISEP, 0, 255      ' loesche alle Separatoren
                                  ' setze neue Bereiche als Separatoren
PUT #2, #0, #UFCO_SET_ISEP, 0, 31, 127, 255
```

Beispiel: lösche Komma als Separatorzeichen für die Instruktion INPUT auf dem Device-Treiber LCD1:

```
PUT #2, #0, #UFCO_RES_ISEP, 2ch, 2ch    ' loesche Komma als Separatoren
```

```
PUT #2, #0, #UFCO_RES_ISEP, ‘,,’        ‘ loesche Komma als Separatoren
```

Ein weiteres Beispiel:


```
PUT #1, #0, #UFCI_ SET_ISEP, 'acXZ55'
‘ setzt als INPUT-Separatoren diese Zeichen:
‘               a, b, c, X, Y, Z, 5
```

Beispiel: erzeuge Echo auf dem LCD:

```
PUT #2, #0, #UFCO_SER_ECHO, JA
```

ESC-Kommandos Tastatur:

ESC-Zeichen	Beschreibung
Esc, Z, Daten, Eos	Tastatur-Codes; immer 128 Bytes Daten Beispiel: S. 366
Esc, z, Daten, Eos	Tastatur-Shift-Codes; immer 128 Bytes Daten Beispiel: S. 366
Esc, a, Daten, Eos	Tastatur-Attribut-Codes; immer 128 Bytes Daten 0 = normale Taste 1 = STRG-Taste (CTRL) 2 = Shift-Lock-Taste 3 = Shift-Taste 4 = Taste ohne Auto-Wiederholung Beisp.: S. 366
Esc, D, Anz, n1, nAnz, Eos	Anz = Anzahl Scan-Spalten (max. 16) n1...nAnz: 0 = DIP-Schalter-Spalte 1 = Tasten-Spalte
Esc, r, n1, n2, Eos	Auto-Repeat der Tastatur n1: Verzögerung in 16msec, 0=ohne n2: Frequenz in 16msec

Um auf dem Plug & Play Lab den Tastenklick zu hören, wird Pin L42 mit ‘beep’ verbunden.

Tastatur-Auto-Repeat: ESC r

PRINT #D, “<1Bh>r“; CHR$(n1); CHR$(n2); “<F0h>“;

Mit diesem Befehl werden Verzögerung und Wiederholrate der Tastatur eingestellt.

D	ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.
n1	Verzögerung (Zeitspanne zwischen Tastendruck und Erzeugung des 1. Codes) in 16 ms (0=nicht aktiv)
n2	Wiederholrate in 16 ms (alle n2*16 ms wird ein neuer Code durch die Tastatur erzeugt)

Beim PC hat sich eine Verzögerung von 250 ms und eine Wiederholrate von 30 Codes/Sekunde bewährt. Dies bedeutet für n1 =16 (16 * 16 = 256 ms) und für n2=2 (alle 32 ms ein Code = ca. 31 Codes/Sekunde). Wenn Tasten keine Auto-Repeat-Funktion besitzen sollen, haben diese ein besonderes Tastenattribut (Siehe Tastenattribute: ESC-a)

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name: REPEAT.TIG
'--------------------------------------------------------------------
' Liest Zeichen von der Tastatur ein. Durch Druecken von <F1> wird
' die Wiederholrate verdoppelt oder wieder halbiert. Mit der Taste
' <ESC> wird das Programm beendet.
'--------------------------------------------------------------------
#INCLUDE KEYB_PP.INC                    ' Dt. Tastatur Plug & Play-Lab

TASK MAIN                               ' Beginn Task MAIN
  STRING A$                             ' STRING-Variable deklarieren
  BYTE   Mode                           ' BYTE-Variable deklarieren
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  CALL INIT_KEYB(1)                     ' dt.Zeichensatz Plug&Play-Lab

  PRINT #1, "<1>";                      ' Bildschirm loeschen
  CALL TYPERATE (62,8)                  ' sehr langsame Wiederholrate
  A$ = ""                               ' A$ initialisieren
  Mode = 0                              ' Mode initialisieren
  WHILE A$ <> "<1Bh>"                   ' Solange nicht ESC gedrueckt
    FOR N = 0 TO 0 STEP 0               ' Endlosschleife bis N=1(GET!)
      RELEASE_TASK                      ' Rest der Task-Zeit freigeben
      GET #1, #0, #1, 1, N              ' N=Zeichen im Tastatur-Buffer
    NEXT                                ' Ende Endlosschleife
    GET #1, 1, A$                       ' Tastatur-Buffer auslesen
    IF A$ = "<F1h>" THEN                ' Falls <F1> gedrueckt:
      IF Mode = 0 THEN                  ' <- Falls Mode = 0:
        Mode = 1                        '    Mode umschalten
        CALL TYPERATE (31,4)            '    Wiederholrate verdoppeln
      ELSE                              '    Sonst:
        Mode = 0                        '    Mode umschalten
        CALL TYPERATE (62,8)            '    Wiederholrate halbieren
      ENDIF                             '
    ELSE                                ' Sonst:
      PRINT #1, A$;                     ' Zeichen ausgeben
    ENDIF                               '
  ENDWHILE                              ' Ende Schleife ueber Eingabe
  PRINT #1, ,"<1>Program end"           ' CR, CS & Ausgabe auf Display
END                                     ' Ende Task MAIN

'--------------------------------------------------------------------
' Subroutine: Stellt die Wiederholrate fuer die Tastatur ein
'--------------------------------------------------------------------
SUB TYPERATE (BYTE n1,n2)               ' Unterprg. fuer Wiederholrate
  PRINT #1, "<1Bh>r"; CHR$(n1);&        ' Einstellungen vornehmen
            CHR$(n2); "<F0H>";          '
END                                     ' Ende Unterprogramm
```

Tasten-Codes: ESC Z bzw. ESC z

PRINT #K, "<1Bh>Z"; Datensatz; "<F0h>";
PRINT #D, "<1Bh>z"; Datensatz; "<F0h>";

Mit diesen Befehlen können alle 128 Tasten mit einem Zeichen-Code belegt werden.

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Datensatz 128 Bytes, die den Zeichen-Code für die jeweilige Taste festlegen. Das erste Byte bestimmt dabei die Belegung für die Taste mit Scan-Code 0, das zweite die Belegung für die Taste mit Scan-Code 1 etc. Es müssen immer 128 Bytes angegeben werden.

Der Befehl **ESC Z** ermöglicht die Definition der Tastatur, d.h. welche Taste mit welchem Code belegt wird. Dabei werden nicht benutzte oder nicht vorhandene Tasten mit einem Dummy-Wert belegt, es werden also mit diesem Kommando immer genau 128 Daten-Byte übertragen. Es ist gestattet, beliebig viele Tasten mit dem gleichen Code zu belegen. So kann durch das Setzen des Codes 97 (61h) für die Bytes 15, 54 und 87 die jeweiligen Tasten mit dem Zeichen 'a' belegt werden. Die Definition durch den Befehl **ESC Z** betrifft nur den normalen, „ungeshifteten" Tastendruck. Für die Definition der Tasten in Verbindung mit der SHIFT-Taste wird der Befehl **ESC z** verwendet. Die Syntax dieser beiden Kommandos ist identisch, grundsätzlich sollten beide Definitionen direkt aufeinander folgen, um die Tastaturbelegung komplett zu definieren.

Ein Beispiel für die Verwendung dieses Befehls stellt die Datei **KEYB_PP.INC** dar, die sich im Unterverzeichnis **INC** befindet.

Tastenattribute: ESC a

PRINT #D, "<1Bh>a"; Datensatz; "<F0h>";

Mit diesem Befehl werden die Attribute aller 128 Tasten (normale Taste, SHIFT-Taste, mit einem Zeichen-Code belegt werden.

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Datensatz 128 Bytes, die das Attribut für die jeweilige Taste festlegen. Das erste Byte bestimmt dabei das Attribut für die Taste mit Scan-Code 0, das zweite das Attribut für die Taste mit Scan-Code 1 etc. Es müssen immer 128 Bytes angegeben werden.

Das a-Kommando legt für alle 128 Tasten das Attribut fest, d.h. ob eine Taste ein normales Zeichen erzeugt, oder ob die Taste eine SHIFT-, SHIFT-LOCK- oder CTRL-Taste ist. Insgesamt stehen 5 Attribute für die Tasten zur Verfügung:

Nr.	Tastenattribut
0	Taste erzeugt normales Zeichen
1	Taste ist CTRL-Taste
2	Taste ist SHIFT-LOCK-Taste
3	Taste ist SHIFT-Taste
4	Taste ohne Auto-Wiederholung

Eine Taste, die nicht benutzt wird, wird mit 0 geladen. Es werden also mit diesem Kommando immer genau 128 Daten-Byte übertragen. Es dürfen beliebig viele Tasten das gleiche Attribut besitzen (z.B. für mehrere SHIFT- oder CTRL-Tasten). So können durch die Angabe von Attribut 3 für die Bytes 29 und 73 die jeweiligen Tasten zu SHIFT-Tasten gemacht werden. Die Attribute 1, 2 und 3 überschreiben die für diese Tasten gesetzten Zeichen-Codes. Besitzt z.B. die Taste 49 den Zeichen-Code 69 und das Attribut 1, erzeugt sie nicht das Zeichen 'E', sondern wird als CTRL-Taste verwendet. Die Attribute 1 bis 4 haben einige Besonderheiten:

1 (CTRL) Wenn eine CTRL-Taste zusammen mit einer anderen Taste gedrückt wird, die einen Zeichen-Code generiert, erzeugt die CTRL -Taste einen Offset auf diesen Zeichen-Code. Dieser Offset beträgt -64 (-40h).

2 (SHIFT-LOCK) Setzt den SHIFT-LOCK. Ist dieser gesetzt, wird anschließend bei allen gedrückten Tasten, die einen Zeichen-Code generieren, dieser Zeichen-Code aus der Tabelle mit der „geshifteten" Belegung gelesen. Die Shift-LED wird eingeschaltet.

3 (SHIFT) Wenn eine SHIFT-Taste zusammen mit einer anderen Taste gedrückt wird, die einen Zeichen-Code generiert, wird dieser aus der „geshifteten“ Code-Tabelle genommen. Außerdem wird ein eventuell gesetzter SHIFT-LOCK zurückgesetzt. Die Shift-LED wird bei gedrückter Shift-Taste eingeschaltet.

4 Das Attribut 4 (keine Auto-Wiederholung) hat zur Folge, daß auch bei aktiver REPEAT-Funktion (siehe Befehl ESC r) bei dieser Taste keine automatische Zeichenwiederholung stattfindet. Dies kann sinnvoll sein für Tasten wie z.B. ESC, RETURN oder die Funktionstasten.

Ein Beispiel für die Verwendung dieses Befehls stellt die Datei **KEYB_PP.INC** dar, die sich im Unterverzeichnis **INC** befindet.

DIP-Schalter: ESC D

PRINT #D, “<1Bh>D“;CHR$(Anz);CHR$(n1);...;CHR$(n16)“<F0h>“;

Mit diesem Befehl wird festgelegt, welche Scan-Spalten der Tastatur DIP-Schalter bzw. Tastaturtasten enthalten.

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Anz Diese Angabe ist immer 16 (Scanspalten), unabhängig von der Anzahl der wirklich vorhandenen Spalten.

n1, ..., n16 BYTE-Angaben
0: an diese Spalte ist ein DIP-Schalter angeschlossen.
1: an dieser Spalte sind Tasten einer Tastatur angeschlossen.
Es müssen alle Parameter n1...n16 aufgeführt werden.

Tastaturspalten, die Tasten enthalten, werden gescannt, und erkannte entprellte Tastendrücke werden entsprechend ihrer Funktion weiterverarbeitet. Abgesehen von Sondertasten wie Shift werden die Tasten-Eingaben in dem Tastaturpuffer abgelegt.

Tastaturspalten, die DIP-Schalter oder sonstige feststehende Schalter enthalten, werden gescannt, und ihr Zustand kann über eine Sekundär-Adresse des Treibers gelesen werden. Es werden keine Scan-Codes in den Tastaturpuffer übernommen.

Programmbeispiel:

```
'-----------------------------------------------------------------------
' Name: DIP_SW.TIG
'-----------------------------------------------------------------------
TASK MAIN                               ' Beginn Task MAIN
  BYTE DIP                              ' BYTE-Variable definieren
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8

  PUT #1,&
  "<1BH>D<16><1><1><1><1><1><1><0><0><1><1><1><1><1><1><1><0F0H>"

  LOOP 9999999                          ' sehr viele Loops
    GET #1, #7, 1, DIP                  ' DIP-Schalter einlesen
    PRINT #1, "<1>DIP-switch 1:"; DIP   ' Ausgabe auf LC-Display
    GET #1, #8, 1, DIP                  ' DIP-Schalter einlesen
    PRINT #1, "DIP-switch 2:"; DIP      ' Ausgabe auf LC-Display
    WAIT DURATION 300                   ' 300 ms warten
  ENDLOOP                               '
END                                     ' Ende Task MAIN
```

Einlesen von DIP-Schaltern

DIP-Schalter werden an die Tastaturmatrix angeschlossen und belegen eine Tastaturspalte und damit eine Adresse.

Durch Angabe der Sekundär-Adresse können DIP-Schalter der Tastatur gelesen werden. Die Sekundäradresse gibt die Reihe der Tastaturmatrix an und liegt zwischen 1 und 16. Die Sekundär-Adresse 0 (Standard) veranlaßt den Treiber, Zeichen aus dem Tastaturpuffer zu lesen.

! Es wird stets das Ergebnis des letzten Tastaturscans gelesen. Da ein Tastaturscan ca 20msec dauert, können die DIP-Schalter nach einem Reset oder nach dem Einschalten frühestens nach 20msec richtig gelesen werden.

Sek.Adr.	Funktion
0	gescannte Zeichen werden aus dem Tastaturpuffer gelesen. Die Tasten-Eingaben werden entprellt, codiert und entsprechend der aktuellen Auto-Repeat-Einstellung bearbeitet .
1...16	Der Scanwert der angegebenen Reihe wird gelesen. Es können alle Reihen gelesen werden, auch solche, die normale Tasten enthalten.

Beispiel:

```
GET #1, #7, 1, DIP                    ' lies DIP-Schalter auf Reihe 7
```

Scan-Adressen: ESC k

PRINT #D, “<1Bh>k“; CHR$(n1);...; CHR$(n16); “<F0h>“;

Dieses Kommando legt die logischen Adressen fest, mit denen die Tastatur gescannt wird. Beachten Sie die Kleinschreibung.Scan-Adressen

D	ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.
n1, ..., n16	BYTE-Angaben Liste von 16 Scan-Adressen.

Die logischen Adressen, die der LCD1-Device-Treiber zum Scannen der Tastatur verwendet, werden mit dem ESC-k-Kommando festgelegt. Es müssen 16 Adressen angegeben werden, auch wenn die Tastatur nicht soviele Spalten hat.

Beispiel für das Plug & Play Lab:

```
PRINT #LCD, "<1Bh>k<98h><99h><9Ah><9Bh><9Ch><9Dh><9Eh><9Fh>&
<A0h><A1h><A2h><A3h><A4h><A5h><A6h><A7h><0F0h>"
```

LCD1-Ton

Der Device-Treiber ‘LCD1.TDD’ stellt eine Tonausgabe für folgende Zwecke zur Verfügung:

- Tastenklick
- Beep (Steuerzeichen-Code 7)
- Fehlerton

Weitere Informationen zu LCD1.TDD, Teil Tastatur:

- Ton ausschalten: ESC C
- Beep: ESC B
- Tastenklick: ESC K
- Steuerzeichen Ton

ESC-Kommandos Ton

Für die Tonausgabe wird ein Summer eingesetzt, der von Pin L42 gesteuert wird, ‘low’ auf Pin 42 schaltet den Ton ein. Es gibt 3 verschiedene Tonsignale: Klick, Beep und Alarm. Die Dauer und Tonabfolge für die 3 Tonsignale kann nach Belieben eingestellt werden.

ESC-Zeichen	Beschreibung
ESC, C, n, EOS	Stop Ton n = 1: Klick n = 2: Standard-Beep n = 3: Alarm-Ton
ESC, B, w1, ...w11, EOS	Setze Beep-Parameter (11x WORD) Tastatur-Klick w1 Dauer der 1. Pause (in 10msec) w2 Dauer der ON-Phase (in 10msec) w3 Dauer der 2. Pause (in 10msec) Standard-Beep: w4 Dauer der ON (in 10msec) w5 Dauer der PAUSE (in 10msec) w6 Anzahl der Pulse Alarm-Beep: w7 Dauer der ON (in 10msec) w8 Dauer der PAUSE (in 10msec) w9 Anzahl der Pulse in Gruppe w10 Pause zwischen den Gruppen w11 Anzahl der Gruppen
ESC, K, n, EOS	Tastenklick n=0: ON, n=255: OFF

Ton ausschalten: ESC C

PRINT #D, “<1Bh>C“; CHR$(n); “<F0h>“;

Dieser Befehl schaltet ein Tonsignal aus.

D	ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.
n	Bestimmt, welches Tonsignal ausgeschaltet werden soll n=1: Der Klick wird ausgeschaltet n=2: Der Standard-Beep wird ausgeschaltet n=3: Der Alarm-Ton wird ausgeschaltet

Der Befehl **ESC C** beendet einen Ton, der gerade ausgegeben wird. Alle weiteren Töne werden danach wieder normal ausgegeben. Der Alarm-Ton z.B. wird normalerweise über eine längere Zeitspanne ausgegeben. Behebt ein Benutzer die Ursache für den Alarm oder quittiert diesen über die Tastatur, kann die Ausgabe des Alarm-Tons mit diesem Befehl sofort abgebrochen werden.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name: STOP_SND.TIG
'--------------------------------------------------------------------
' Alle drei Tonsignale (Klick, Beep, Alarm) werden je zweimal
' ausgegeben. Beim ersten Mal vollstaendig, beim zweiten Mal wird
' die Ausgabe sofort oder nach kurzer Zeit abgebrochen
'--------------------------------------------------------------------
TASK MAIN                               ' Beginn Task MAIN
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8

  PRINT #1, "<1Bh>c<0><F0h>";           ' Cursor ausschalten
  PRINT #1, "<1>Click<0>"               ' Klick "<0>" ausgeben
  WAIT DURATION 2000                    ' 2 Sek warten
  PRINT #1, "<1>Beep<7>"                ' Beep "<7>" ausgeben
  WAIT DURATION 2000                    ' 2 Sek warten
  PRINT #1, "<1>Alarm (7 Sec)<20>"      ' Alarm "<20>" ausgeben
  WAIT DURATION 10000                   ' 10 Sek warten
  PRINT #1, "<1>Just a moment!"         ' Ausgabe auf LC-Display
  WAIT DURATION 2000                    ' 2 Sek warten
  PRINT #1, "<1>Click<0>"               ' Klick "<0>" ausgeben
  PRINT #1, "<1Bh>C";CHR$(1);"<F0h>";   ' Ausgabe "Klick" abbrechen
  WAIT DURATION 2000                    ' 2 Sek warten
  PRINT #1, "<1>Beep<7>"                ' Beep "<7>" ausgeben
  PRINT #1, "<1Bh>C";CHR$(2);"<F0h>";   ' Ausgabe "Beep" abbrechen
  WAIT DURATION 2000                    ' 2 Sek warten
  PRINT #1, "<1>Alarm (7 Sek)<20>"      ' Alarm "<20>" ausgeben
  WAIT DURATION 2000                    ' 2 Sek warten
  PRINT #1, "aborted"                   ' Ausgabe auf LC-Display
  PRINT #1, "<1Bh>C";CHR$(3);"<F0h>";   ' Ausgabe "Alarm" abbrechen
  WAIT DURATION 8000                    ' 8 Sek warten
  PRINT #1, "<1>Program end"            ' ClrScr & Ausgabe auf Display
END                                     ' Ende Task MAIN
```

Im ersten Durchgang werden alle drei Tonsignale einmal vollständig ausgegeben. Im zweiten Durchgang wird die Ausgabe für Klick und Beep sofort abgebrochen (d.h. der Ton ist gar nicht oder kaum wahrnehmbar), die Ausgabe des Alarm-Tons wird nach 2 Sekunden abgebrochen.

Beep: ESC B

PRINT #K, "<1Bh>B"; Datensatz; "<F0h>";

Dieser Befehl bestimmt die Ausgabeform für das BEEP-Steuerzeichen. Mit den Parametern werden die Anzahl und die Dauer der Töne sowie die Pausen zwischen den einzelnen Tönen festgelegt.

Parameter:

K Kanalnummer des Device-Treibers

Datensatz 11 WORD-Parameter mit folgender Bedeutung:
w1: Dauer der ersten Pause für Klick (in 10 ms)
w2: Dauer der ON-Phase für Klick (in 10 ms)
w3: Dauer der zweiten Pause für Klick (in 10 ms)
w4: Dauer der ON-Phase für Beep (in 10 ms)
w5: Pause zwischen zwei ON-Phasen des Beep (in 10 ms)
w6: Anzahl der ON-Phasen des Beep
w7: Dauer der On-Phase für Alarm (in 10 ms)
w8: Pause zwischen zwei ON-Phase des Alarm (in 10 ms)
w9: Anzahl der ON-Phasen einer Gruppe des Alarm
w10: Pause zwischen zwei Gruppen des Alarm (in 10 ms)
w11: Anzahl der Ausgabe der Gruppe des Alarm

Die WORD-Parameter werden als STRING zum Device-Treiber übertragen. Dieser String ist 22 Zeichen lang und enthält die Parameter in der Reihenfolge:

- Low-Byte w1, High-Byte w1,
- Low-Byte w2, High-Byte w2,
- ...
- Low-Byte w11, High-Byte w11.

Die Umwandlung kann mit der Funktion NTOS$ vorgenommen werden (siehe Beispiel).

Im ersten Durchgang werden alle drei Tonsignale einmal vollständig ausgegeben, wie sie durch den Device-Treiber vordefiniert sind. Im zweiten Durchgang werden die durch den **ESC B** Befehl modifizierten Tonsignale ausgegeben.

Programmbeispiel:


```
'--------------------------------------------------------------------
' Name: SET_SND.TIG
'--------------------------------------------------------------------
' Alle drei Tonsignale (Klick, Beep, Alarm) werden je zweimal ausge-
' geben. Beim ersten Mal so wie durch den Device-Treiber vorgegeben,
' beim zweiten Mal durch den ESC-B Befehl modifiziert.
'--------------------------------------------------------------------
TASK MAIN                               ' Beginn Task MAIN
  STRING A$                             ' STRING-Variable deklarieren
  ARRAY  W(12) OF WORD                  ' WORD-Array deklarieren
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8

  A$ = "00000000000000000000"           ' A$ initialisieren
  W(1) = 1                              ' Klick: Dauer 1. Pause
  W(2) = 10                             ' Klick: Dauer On-Phase
  W(3) = 1                              ' Klick: Dauer 2. Pause
  W(4) = 50                             ' Beep:  Dauer On-Phase
  W(5) = 10                             ' Beep:  Pause zw. On-Phasen
  W(6) = 1                              ' Beep:  Anzahl Pulse
  W(7) = 20                             ' Alarm: Dauer On-Phase
  W(8) = 20                             ' Alarm: Pause zw. On-Phasen
  W(9) = 3                              ' Alarm: Anzahl Pulse/Gruppe
  W(10) = 50                            ' Alarm: Pause zw. Gruppen
  W(11) = 3                             ' Alarm: Anzahl Gruppen
  PRINT #1, "<1Bh>c<0><F0h>";           ' Cursor ausschalten
  PRINT #1, "<1>Click<0>"               ' Klick "<0>" ausgeben
  WAIT DURATION 2000                    ' 2 Sek warten
  PRINT #1, "<1>Beep<7>"                ' Beep "<7>" ausgeben
  WAIT DURATION 2000                    ' 2 Sek warten
  PRINT #1, "<1>Alarm (7 Sec)<20>"      ' Alarm "<20>" ausgeben
  WAIT DURATION 10000                   ' 10 Sek warten
  FOR X = 0 TO 10                       ' Zaehlschleife fuer X
    A$ = NTOS$ ( A$, X*2, 2, W(X+1) )   ' W(X) in A$ einsetzen
  NEXT                                  ' naechster Wert
  PRINT #1, "<1Bh>B"; A$; "<F0h>";      ' SET_TON Befehl
  PRINT #1, "<1>Click<0>"               ' Klick "<0>" ausgeben
  WAIT DURATION 2000                    ' 2 Sek warten
  PRINT #1, "<1>Beep<7>"                ' Beep "<7>" ausgeben
  WAIT DURATION 2000                    ' 2 Sek warten
  PRINT #1, "<1>Alarm (7 Sec)<20>"      ' Alarm "<20>" ausgeben
  WAIT DURATION 7000                    ' 7 Sek warten
  PRINT #1, "<1>Program end"            ' CR, CS & Ausgabe auf Display
END                                     ' Ende Task MAIN
```

Tastenklick: ESC K

PRINT #D, “<1Bh>K“; CHR$(n); “<F0h>“;

Dieser Befehl schaltet den Tastenklick ein oder aus.

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

n Bestimmt, ob Tastenklick ein- oder ausgeschaltet werden soll
n=0: Der Tastenklick wird eingeschaltet
n=255: Der Tastenklick wird ausgeschaltet

Programmbeispiel:

```
'-------------------------------------------------------------------------
' Name: KEYCLICK.TIG
'-------------------------------------------------------------------------
' Liest Zeichen von der Tastatur ein. Durch Druecken von <F1> wird
' der Tastaturklick ein- oder ausgeschaltet. Mit der Taste <ESC>
' wird das Programm beendet.
'-------------------------------------------------------------------------
#INCLUDE KEYB_PP.INC                    ' Dt. Tastatur Plug & Play-Lab

TASK MAIN                               ' Beginn Task MAIN
  STRING A$                             ' STRING-Variable deklarieren
  BYTE   Mode, N                        ' BYTE-Variablen deklarieren
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  CALL INIT_KEYB(1)                     ' dt.Zeichensatz Plug&Play-Lab

  PRINT #1, "<1>";                      ' Bildschirm loeschen
  A$ = ""                               ' A$ initialisieren
  Mode = 0                              ' Mode initialisieren
  WHILE A$ <> "<1Bh>"                   ' Solange nicht ESC gedrueckt
    FOR N = 0 TO 0 STEP 0               ' Endlosschleife bis N=1(GET!)
      RELEASE_TASK                      ' Rest der Task-Zeit freigeben
      GET #1, #0, #1, 1, N              ' N=Zeichen im Tastatur-Buffer
    NEXT                                ' Ende Endlosschleife
    GET #1, 1, A$                       ' Tastatur-Buffer auslesen
    IF A$ = "<F1h>" THEN                ' Falls <F1> gedrueckt:
      IF Mode = 0 THEN                  ' <- Falls Mode = 0:
        Mode = 1                        '    Mode umschalten
        PRINT #1,"<1Bh>K";CHR$(255);"<F0H>"; ' Tastenklick ausschalten
      ELSE                              '    Sonst:
        Mode = 0                        '    Mode umschalten
        PRINT #1,"<1Bh>K";CHR$(0);"<F0H>";   ' Tastenklick einschalten
      ENDIF                             '
    ELSE                                ' Sonst:
      PRINT #1, A$;                     ' auf LC-Display ausgeben
    ENDIF                               '
  ENDWHILE                              ' Ende Schleife ueber Eingabe
  PRINT #1, ,"<1>End Program"           ' CR, CS & Ausgabe auf Display
END                                     ' Ende Task MAIN
```

Steuerzeichen Ton

Steuerzeichen werden direkt auf das LCD-Device geschrieben, ohne ESC und ohne EOS.

CLICK	<00>	Tastenklick
BELL	<07>	Standardglocke
ALARM	<14h>	Alarm-Beep

```
PRINT #1, "<0>"
PRINT #1, "<7>"
PRINT #1, "<14h>"
```

erzeugt einen Tastenklick. Dazu wird der Pin L42 (Tiger A Pin-Nr. 35) mit dem ‘beep’-Pin des Plug & Play Labs verbunden.

2

LCD - Grafik-Display

Dieser Device-Treiber ermöglicht die Ausgabe auf grafikfähige LC-Displays. Grafikanwendungen werden weitergehend durch Funktionen unterstützt (Siehe Kapitel 'Grafik' im Programmierhandbuch).

Hinweis: Ein Grafik-Toolkit steht zur Verfügung, auf dem speziell Entwicklungen mit Grafik-LCD durchgeführt werden. Die Applikationen zum Grafik-Toolkit finden Sie im Kapitel ‚Applikationen' in diesem Handbuch.

Weitere Informationen zum Device-Treiber LCD-6963:

- Typenliste für LCD-6963
- Grafik-LC-Display anschließen
- User-Function-Codes
- Steuerzeichen des LCD-6963
- ESC-Kommandos LCD-6963 (Text)
- LCD6963 Cursor positionieren: ESC A
- T6963-Modus: ESC m
- Grafikdisplay ein-/ausschalten: ESC G
- Textdisplay ein-/ausschalten: ESC T
- LCD6963 Cursor definieren: ESC c
- T6963 – Sonderzeichensatz
- Grafik des T6963
- Ausgabe auf den Grafikbildschirm
- Grafik-LCD Funktionen (Übersicht)

Dateiname: LCD-6963.TDD

INSTALL DEVICE #*D*, "LCD6963.TDD" [, *P1, ..., P6*]

D	ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.
P1...P6	sind weitere Parameter, die die Standard-Anschlußbelegung des LC-Displays verändern.

Alle Parameter P1...P6 sind Bytes und können durch die Angabe von 0 oder 0EEH (=238) als Wert unverändert bleiben. Wenn Sie die Bit-Zuordnung der Steuerleitungen verändern wollen, dann müssen die Parameter P3a bis P3d alle angegeben werden, während einmal '0EEH' als Wert für den 3. Parameter besagt, daß alle Steuerleitungen unverändert bleiben.

	Unverändert lassen	**Beschreibung des Parameters**
P1	0	Logische Datenbus-Adresse (6 oder 8)
P2	0	Logische Port-Adresse der Steuerleitungen (4, 6, 7, 8 oder 9)
P3a P3b P3c P3d	0EEH	Bit-Positionen (0...7) für die Steuerleitungen des LC-Displays (wenn unverändert, nur ein Parameter 0EEH, sonst müssen alle 4 Parameter angegeben werden): -WR -RD -CE C/D
P4	0EEH	LCD-Typ (siehe Tabelle)
P5	0	Übertragungsrate in kByte/Sek. (4...120) betrifft gewünschte CPU-Belastung.
P6	0EEH	Übertragungsrate (betrifft LCD-interne Eigenschaften. Nur verändern, wenn LCD zu langsam erscheint und Daten bei der Übertragung verloren gehen.

Es gibt eine Vielzahl unterschiedlicher Graphic-LCDs mit dem T6963 Controller in unterschiedlichen Abmessungen und mit unterschiedlicher Pixelzahl. Dabei kann es Unterschiede geben:

- bei dem Seitenverhältnis der Pixel. Üblich sind:
  quadartische Pixel (Bild wird 1:1 gezeigt),
  längliche Pixel (hochkantig).
  Das führt zu einer entsprechenden Verzerrung im Seitenverhältnis einer graphischen Darstellung.
- Beim Zeichengenerator. Es gibt Displays fester und einstellbarer Zeichengröße von 5x8...8x8 Dots.

Typenliste für LCD-6963

LC-Display-Typen mit Toshiba Prozessor: T6963

Nr	LCD-Typ	Pixelspalten x -zeilen	Font	Textspalten x - Zeilen	RAM
1	LCD_TGR_BW1	128 x 64	8 x 8	8 x 16	
2	LCD_TGR_BW2	128 x 128	8 x 8	16 x 16	
3	LCD_TGR_BW3	240 x 128	8 x 8	16 x 30	32k
4	LCD_TGR_BW4	240 x 128	8 x 8	16 x 30	8k
5	LCD_TGR_BW5	192 x 128	8 x 8	16 x 24	
6	LCD_TGR_BW6	240 x 128	6 x 8	16 x 40	8/32k
7	LCD_TGR_BW7	128 x 64	6 x 8	8 x 21	
8	LCD_TGR_BW8	192 x 128	6 x 8	16 x 32	

Anmerkungen: Die Fontgrößen müssen passend durch Beschaltung an einem Pin festgelegt werden.
Grafik-LCD 240x128 (Typ 4 und Typ 6) mit nur 8kByte haben nur einen Grafikpuffer, so daß laufende Ausgaben gesehen werden. Diese LCDs können jedoch auch als 'Typ 3' installiert werden, wenn entweder nur Grafikausgabe oder nur Textausgabe stattfindet.

Hinweis: Bei schnell bewegten Darstellungen ist die Trägheit der Anzeige beim jeweiligen LCD-Typ zu beachten. Manchmal kann man das sichtbare Ergebnis dadurch verbessern, daß man feine Strukturen während der Bewegungsphase größer ausführt als im Ruhezustand.

Die Ausgabe auf das LCD-Device ist je nach Funktion auf verschiedene Sekundäradressen verteilt:

Sekundär-Adresse	Funktion	Ausgabe-Instruktion
0	Text	PRINT, PUT
1	Grafik	PUT
2	Nicht verfügbar	
3	Zeichengenerator setzen (RAM)	PUT

Bevor die Datenausgabe beschrieben wird, folgen ein paar weitere grundlegende Eigenschaften des LCD-6963, die Auflistung der User-Function-Codes sowie der ESC-Sequenzen. Zur Datenausgabe siehe 'Ausgabe auf den Grafikbildschirm' ab Seite 124. Außerdem finden Sie im Programmierhandbuch ein ausführliches Kapitel

zu Grafikfunktion und im Device-Treiber-Handbuch unter den Applikationen viele Grafik-Applikationen.

Grafik-LC-Display anschließen

Ein Grafik-LC-Display benötigt vom BASIC-Tiger®-Modul den Datenbus und vier weitere I/O-Leitungen. Die folgende Tabelle zeigt die Standard-Anschlußbelegung:

LCD-Pin-Funktion	Pin-Bezeichnung	Pin-Nr. BASIC-Tiger®	Pin-Nr. Tiny-Tiger®
D0...D7	L60...L67	2...9	1...8
-WR	L80	14	13
-RD	L81	15	14
-CE	L82	16	15
C/D	L83	17	16

Es empfiehlt sich die 4 CTRL-Leitungen des LCDs mit einem Pull-UP Widerstand zu versehen, da die Tiger-Pins während der Power-On Phase hochohmig sind und das LCD in einen undefinierten Zustand geraten könnte. Mindestens <-CE> sollte man so verschalten.

Weiterhin ist neben der Stromversorgung und der Kontrasteinstellung ein Reset-Eingang am Grafik-LC-Display vorhanden. Das Display braucht beim Power-ON unbedingt einen RESET (siehe LCD Unterlagen) und wird daher üblicherweise an den System-RESET angeschlossen. (Beachte: der Reset-Pin des BASIC-Tiger® oder Tiny-Tiger® ist kein Ausgang.)

Der Pin zur Bestimmung der Fontgröße (in der Regel FS=Font Select) muß passend zum gewünschten Treiber verschaltet sein (z.B. bei LCD-Typ 4 auf low=8x8-Font, bei LCD-Typ 6 auf high=6x8-Font).

Da LC-Displays eigene Controller besitzen, können Sie unabhängig von der steuernden Umgebung abstürzen. Das passiert durch statische Entladung in der Nähe des LC-Displays oder durch Einstreuung in das Datenkabel, insbesondere wenn dieses sehr lang ist. Bei langen Datenkabeln ist die Masseverbindung erfahrungsgemäß besonders kritisch. Als Abhilfe für solche Fälle können Sie vorsehen, daß das LC-Display vom steuernden Modul zurückgesetzt werden kann. Dazu könnte ein freier Port-Pin mit dem RESET-Eingang des LCDs verbunden werden, so daß unter Kontrolle des BASIC-Programms das LCD zurückgesetzt werden kann. Als Software-Reset steht das Kommando UFCO_LCD_RESET zur Verfügung.

User-Function-Codes des LCD-6963

User-Function-Codes des LCD-6963.TDD zum Abfragen von Parametern (Instruktion GET, Sekundäradresse 0, wenn nicht gesondert gekennzeichnet):

Nr	Symbol Prefix UFCI_	Beschreibung
33	UFCI_OBU_FILL	Füllstand des Ausgangspuffers (Byte)
34	UFCI_OBU_FREE	freier Platz im Ausgangspuffer (Byte)
35	UFCI_OBU_VOL	Größe des Ausgangspuffers (Byte)
99	UFCI_DEV_VERS	Version des Treibers

User-Function-Codes des LCD-6963.TDD zum Setzen von Parametern (PUT):

Nr	Symbol Prefix: UFCO_	Beschreibung
68	UFCI_CPU_LOAD	lese die durch diesen Treiber beanspruchte CPU-Leistung (100%=10.000)
99	UFCI_DEV_VERS	Version des Treibers
144	UFCO_LCD_TYPE	Setze LCD-Typ neu.
156	UFCO_LCD_TXTSIZ	Setzt Anzahl der Zeilen sowie Anzahl der Zeichen pro Zeile: 1. Parameterbyte: Anzahl Zeilen 2. Parameterbyte: Anzahl Zeichen/Zeile Danach muß das Resetkommando gegeben werden.
157	UFCO_LCD_LINE	2 Parameter bestimmen das Verhalten a) bei Erreichen des Zeilenendes: 0: weiter auf nächster Zeile 1: weiter auf dieser Zeile 2: stop b) bei Erreichen des Bildschirmendes: 0: weiter am Bildschirmanfang 1: weiter auf der letzten Zeile (Anfang) 2: stop
158	UFCO_LCD_PARAM	Setze alle LCD-Parameter wie in der INSTALL-Zeile
159	UFCO_LCD_PWR	setzt CPU-Power 4...128
159	UFCO_CPU_LOAD	Setze relative CPU-Leistung (4...88%)
176	UFCO_LCD_RESET	Initialisiere LCD neu

Insbesondere bei Grafikausgabe kann der Device-Treiber die CPU sehr stark belasten, so daß wenig CPU-Power für andere Tasks oder Treiber übrig bleibt. Je nach Anwendung kann der Benutzer selbst entscheiden, wie hoch die CPU-Belastung maximal sein soll. Mit dem User-Function-Code UFCO_CPU_LOAD kann eine maximale Belastung von 4...88% eingestellt werden.

Steuerzeichen des LCD-6963

Steuerzeichen werden direkt auf das LCD-Device geschrieben, ohne ESC und ohne EOS.

HOME	<02>	setzt den Cursor in die linke obere Ecke
FS	<05>	Cursor 1 Position nach rechts
BS	<08>	Cursor 1 Position nach links
LF, DO	<0Ah>	Cursor 1 Position nach unten
UP	<0Bh>	Cursor 1 Position nach oben
CR	<0Dh>	Carriage Return, Wagenrücklauf


```
PRINT #LCD, "<2>";                ' cursor home
```

bringt den Cursor in die 'home'-Position (X=0, Y=0). Es kann direkt auch weiterer Text folgen:

```
PRINT #LCD, "<2>Hello World"
```

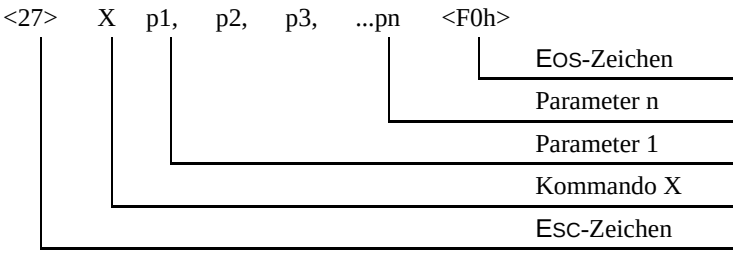
ESC-Kommandos LCD-6963 (Text)

Kommandos an den LCD1-6963-Device-Treiber können auch mit dem normalen Datenstrom in PRINT oder PUT-Instruktionen in Form von ESC-Sequenzen erfolgen. Die Kommandosequenz fängt dann mit <ESC>-Code an, es folgt ein Kommandozeichen und abhängig vom Kommando eine unterschiedliche Anzahl Parameter. Die Sequenz endet mit einem EOS-Zeichen.

ESC: <1Bh>

EOS: <F0h>

Kommandozeichen: die Groß- und Kleinschreibung wird hier beachtet!



Die Argumente der folgenden ESC-Kommandos (x, y, n) sind immer BYTES, wenn nicht anders angegeben. Groß- und Kleinschreibung werden in den Kommandos unterschieden.

!

Übersicht:

ESC-Zeichen	Beschreibung
ESC, A, x, y, EOS	Absolute Cursor-Adressierung
ESC, c, n, EOS	Cursor definieren n=0: Cursor = 1 Linie n=1: Cursor = 2 Linien n=7: Cursor = 8 Linien Bit 3: 0=blinken aus, 1=blinken an (n+10h) Bit 4: 0=mit Cursor, 1=ohne Cursor (n+20h)
ESC, m, n, EOS	Modus: Bits 4+3: Verknüpfung Text und Grafik 00 = Modus OR 01 = Modus XOR 10 = Modus XOR 11 = Modus AND Bit-0: 0 = interner Zeichensatz, 1 = Sonderzeichens.
ESC, G, n, EOS	Grafik n=0: aus n=1: ein
ESC, T, n, EOS	Text n=0: aus n=1: ein

ESC-Sequenzen sind stets in einer PRINT- oder PUT-Instruktion auszugeben. Wenn die Zeilenlänge es zuläßt, können mehrere ESC-Sequenzen in einer Instruktion ausgegeben werden.

LCD6963 Cursor positionieren: ESC A

PRINT #D, “<1Bh>A“; CHR$(x); CHR$(y); “<F0h>“;

Positioniert den Cursor absolut auf dem Display.

D	ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.
x	x-Koordinate (Spalte), auf die der Cursor positioniert werden soll.
y	y-Koordinate (Zeile), auf die der Cursor positioniert werden soll.

Die Zählung der Zeilen und Spalten beginnt mit 0. Der mögliche Wertebereich hängt vom verwendeten LC-Display ab. Zu große Werte für **x** und **y** werden auf den Maximalwert gesetzt.

Programmbeispiel:

```
'------------------------------------------------------------------------
' Name: T69ESC_A.TIG
'------------------------------------------------------------------------
TASK MAIN                               ' Beginn Task MAIN
  INSTALL DEVICE #1, "LCD-6963.TDD"     ' graf. Display installieren
                                        ' zeige verschiedene Pos.
  PRINT #1, "<1Bh>A<0><0><0F0h>X<3Ch>--0,0"
  PRINT #1, "0...15 column"
  PRINT #1, "0...7 line"
  PRINT #1, "<1Bh>A<1><5><0F0h>X<3Ch>--1,5"
  PRINT #1, "<1Bh>A<8><7><0F0h>15,7--<3Eh>X"
END                                     ' Ende Task MAIN
```

T6963-Modus: ESC m

PRINT #D, “<1BH>m“; CHR$(n); “<F0H>“;

Setzt den Modus des Grafik-Displays.

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

n Modus-Nummer.

Es gibt je 3 Modi mit dem internen Zeichensatz und mit dem Sonderzeichensatz, die sich durch die logische Verknüpfung der Text- und Grafikpixel unterscheiden. Ein gesetztes Pixel bedeutet, es erscheint ein dunkler Punkt auf dem hellen LCD-Hintergrund:

Modus	
0	interner Zeichensatz, OR: Text überdeckt Grafik (Text- und Grafikpixel, beide gesetzt)
1	Sonderzeichensatz, OR: Text überdeckt Grafik (Text- und Grafikpixel, beide gesetzt)
4	interner Zeichensatz, XOR: Text invertiert Grafik (Text- oder Grafikpixel gesetzt, nie beide)
5	Sonderzeichensatz, XOR: Text invertiert Grafik (Text- oder Grafikpixel gesetzt, nie beide)
8	Modus wie 4
9	Modus wie 5
12	interner Zeichensatz, AND: Text oder Grafikpixel erscheinen nur dort, wo beide gesetzt sind.
13	Sonderzeichensatz, AND: Text oder Grafikpixel erscheinen nur dort, wo beide gesetzt sind.

Programmbeispiel:

```
'-------------------------------------------------------------------------
' Name: T69ESC_M.TIG
'-------------------------------------------------------------------------
user_var_strict                    ' Vars deklarieren!
#include DEFINE_A.INC              ' allgemeine Definitionen
#include LCD_4.INC                 ' Definitionen fuer LCD Typ 4
#include GR_TK1.INC                ' Definitionen fuer Graphic Toolkit

DATALABEL T69MGR_M

TASK MAIN                          ' Beginn Task MAIN
T69MGR_M::
DATA FILTER "T69MGR_M.BMP", "GRAPHFLT", 0 ' lade Grafik
  BYTE I
  STRING GCLR$(130)                ' String loescht Display

  CALL Init_LCDpins                ' initialisiere LCD-Pins
                                   ' LCD-4=240x128, 150 KB/s
  INSTALL_DEVICE #1,"LCD-6963.TDD",0,0,0EEH,LCD_TYPE,150,11H

  GCLR$ = "<2>"                    ' erzeuge GCLR String
  LOOP 16                          '
    GCLR$ = GCLR$ + "         "    '
  ENDLOOP                          '
  GCLR$ = GCLR$ + "<2>"            '


  PRINT #1, "<1Bh>T<1><0F0h>"      ' Text einschalten
  PRINT #1, "<1Bh>G<1><0F0h>"      ' Grafik einschalten
  PUT #1, #1, T69MGR_M, 0, 0, GR_SIZE' Grafik ausgeben
  WAIT_DURATION 1000               ' 1 Sek. warten

  PRINT #1, "<1Bh>m<0><0F0h>";     ' Modus OR
  FOR I = 0 TO 7
    PRINT #1, "mode OR mode OR ";
  NEXT
  WAIT_DURATION 3000               ' 3 Sek. warten

  PRINT #1, GCLR$;
  PRINT #1, "<1Bh>m<1><0F0h>";     ' Modus XOR
  FOR I = 0 TO 7
    PRINT #1, "mode XORmode XOR";
  NEXT
  WAIT_DURATION 3000               ' 3 Sek. warten

  PRINT #1, GCLR$;
  PRINT #1, "<1Bh>m<3><0F0h>";     ' Modus AND
  FOR I = 0 TO 7
    PRINT #1, "mode ANDmode AND";
  NEXT
END                                ' Ende Task MAIN
```

Grafikdisplay ein-/ausschalten: ESC G

PRINT #D, “<1BH>G“; CHR$(n); “<F0H>“;

Schaltet die Grafik des Displays ein oder aus.

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

n 0 = aus, 1 = ein.

Mit dieser ESC-Sequenz wird die Grafik auf dem Display ein- oder ausgeschaltet. Der Textinhalt wird dadurch nicht beeinflußt.

Programmbeispiel:

```
'------------------------------------------------------------------------
' Name: T69ESC_G.TIG
'------------------------------------------------------------------------
user_var_strict                   ' Vars deklarieren!
#include DEFINE_A.INC             ' allgemeine Definitionen
#include LCD_4.INC                ' Definitionen fuer LCD Typ 4
#include GR_TK1.INC               ' Definitionen fuer Graphic Toolkit

DATALABEL T69MGR

TASK MAIN                         ' Beginn Task MAIN
T69MGR::
DATA FILTER "T69MGR.BMP", "GRAPHFLT", 0 ' lade Grafik

  CALL Init_LCDpins               ' initialisiere LCD-Pins
                                  ' LCD-4=240x128, 150 KB/s
  INSTALL_DEVICE #1,"LCD-6963.TDD",0,0,0EEH,LCD_TYPE,150,11H

  PRINT #1, "<1Bh>T<1><0F0h><1Bh>A<4><0><0F0h>Text ON"
  PRINT #1, "<1Bh>m<1><0F0h>";    ' Modus XOR
  PUT #1, #1, T69MGR, 0, 0, GR_SIZE' Grafik ausgeben
  PRINT #1, "<1Bh>G<1><0F0h><1Bh>A<3><3><0F0h>Graphic ON "
  WAIT_DURATION 3000              ' 3 Sek. warten
  PRINT #1, "<1Bh>G<0><0F0h><1Bh>A<3><3><0F0h>Graphic OFF"
  WAIT_DURATION 3000              ' 3 Sek. warten
  PRINT #1, "<1Bh>G<1><0F0h><1Bh>A<3><3><0F0h>Graphic ON "
END                               ' Ende Task MAIN
```

Näheres zu den Sonderzeichen der LC-Displays finden Sie unter ‚LC-Display - Sonderzeichensätze’ auf den Seiten 52f.

Siehe auch: Esc-Kommando S (Zeichensatz einschalten) und Esc-Kommando R (Zeichensatz zurücksetzen).

Textdisplay ein-/ausschalten: ESC T

PRINT #D, “<1BH>T“; CHR$(n); “<F0H>“;

Schaltet den Text des Displays ein oder aus.

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

n 0 = aus, 1 = ein.

Mit dieser ESC-Sequenz wird der Text auf dem Display ein- oder ausgeschaltet. Der Grafikinhalt wird dadurch nicht beeinflußt.

Programmbeispiel:

```
'------------------------------------------------------------------------
' Name: T69ESC_T.TIG
'------------------------------------------------------------------------
user_var_strict                   ' Vars deklarieren!
#include DEFINE_A.INC             ' allgemeine Definitionen
#include LCD_4.INC                ' Definitionen fuer LCD Typ 4
#include GR_TK1.INC               ' Definitionen fuer Graphic Toolkit

DATALABEL T69MGR

TASK MAIN                         ' Beginn Task MAIN
T69MGR::
DATA FILTER "T69MGR.BMP", "GRAPHFLT", 0 ' lade Grafik

  CALL Init_LCDpins               ' initialisiere LCD-Pins
                                  ' LCD-4=240x128, 150 KB/s
  INSTALL_DEVICE #1,"LCD-6963.TDD",0,0,0EEH,LCD_TYPE,150,11H

  PRINT #1, "<1Bh>G<1><0F0h>Graphic ON"
  PRINT #1, "<1Bh>m<1><0F0h>";    ' Modus XOR
  PUT #1, #1, T69MGR, 0, 0, GR_SIZE' Grafik ausgeben
  PRINT #1, "<1Bh>T<1><0F0h><1Bh>A<3><3><0F0h>Text ON "
  WAIT_DURATION 3000              ' 3 Sek. warten
  PRINT #1, "<1Bh>T<0><0F0h>"
  WAIT_DURATION 3000              ' 3 Sek. warten
  PRINT #1, "<1Bh>T<1><0F0h><1Bh>A<3><3><0F0h>&
For 3sec<1Bh>A<2><4><0F0h>Text was OFF"
END                               ' Ende Task MAIN
```

LCD6963 Cursor definieren: ESC c

PRINT #D, “<1Bh>c“; CHR$(n); “<F0h>“;

Definiert das Aussehen des Cursors auf dem Display.

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

n Bestimmt die Form des Cursors
n=0: Cursor ist eine Linie
n=1: Cursor ist zwei Linien
...
n=7: Cursor ist acht Linien
Bit-3: Cursor (0...7) + 8 oder -> Cursor blinkt
Bit-4: Cursor (0...7) + 16 oder +10h -> Cursor aus

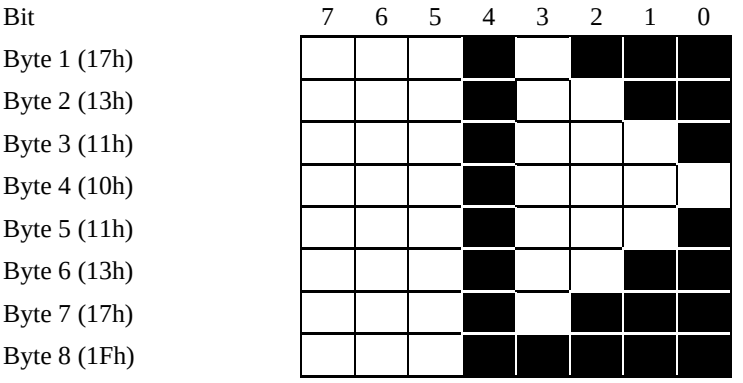
Programmbeispiel:

```
'------------------------------------------------------------------------
' Name: T69ESC_C.TIG
'------------------------------------------------------------------------
TASK MAIN                                ' Beginn Task MAIN
  INSTALL DEVICE #1, "LCD-6963.TDD"      ' graf. Display installieren

  PRINT #1,"<1>Cursor modes:"
  PRINT #1,"0...7: constant"
  PRINT #1,"8...15: blinking"
  FOR N=0 TO 15                          ' verschiedene Cursor Formen:
    PRINT #1,"<1BH>c";CHR$(N);"<0F0H>";
    PRINT #1,"<1BH>A<0><4><0F0H>n = ";N;" A";CHR$(8);
    WAIT DURATION 1000                   ' 1 Sek. warten
  NEXT
  PRINT #1,"<1BH>c";CHR$(16);"<0F0H>";  ' ---> ohne Cursor !
  PRINT #1,"<1BH>A<0><4><0F0H>n = 16    ";
  PRINT #1,"<1BH>A<0><6><0F0H>no Cursor: A<8>";
END                                      ' Ende Task MAIN
```

T6963 - Sonderzeichensatz

Viele Applikationen benötigen Sonderzeichen auf dem LC-Display. In Deutschland werden die Umlaute verwendet, die im Standardzeichensatz nicht enthalten sind.

Der LCD-Device-Treiber unterstützt mit seinen ESC-Kommandosequenzen die Programmierung des Sonderzeichensatzes des LC-Displays. Der RAM-Speicher des Zeichengenerators wird über die Sekundär-Adresse 3 angesprochen. Es kann ein Satz von bis 128 eigenen Text-Zeichen aufgebaut werden. Jedes TEXT-Zeichen wird aus 8 Bytes zusammengesetzt:



Die Programmierung erfolgt wie bei Graphikausgabe:

PUT #D, #3, data_string [, lcd_offset, src_offset, src_len]

D	ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.
#3	Sekundär-Adresse = 3: es werden Sonderzeichen definiert.
pixel_string	ist eine globale oder tasklokale Variable oder eine Konstante vom Typ STRING und enthält die Source-Daten der Sonderzeichen, die neu definiert werden.
lcd_offset	ist eine Variable, Konstante oder ein Ausdruck vom Typ BYTE, WORD oder LONG und gibt den Byte-Offset im Sonderzeichensatz-RAM an. Das Leerzeichen (ASCII 20H) hat den Offset 0.
src_offset	ist eine Variable, Konstante oder ein Ausdruck vom Typ BYTE, WORD oder LONG und gibt den Byte-Offset im **pixel_string** an. Es kann so auch nur ein Teil der Sonderzeichen neu definiert werden.

scr_len ist eine Variable, Konstante oder ein Ausdruck vom Typ BYTE, WORD oder LONG und gibt die Anzahl der Bytes an, die ab **src_offset** ausgegeben werden sollen.

Bei ‘lcd_offset’ = 0 beginnt der Datensatz für das Leerzeichen (ASCII-Code = 20H), nach 8 Byte folgt der Datensatz für ‘!’ (ASCII 21H), etc. Der komplette RAM-Zeichen-Generator umfaßt 128 Text-Zeichen zu je 8 Bytes, deren niederwertigsten 5 Bits zur Darstellung verwendet werden. Insgesamt beschreiben also 1024 Bytes den Sonderzeichensatz. Es ist sinnvoll stets den gesammten Zeichen-Generator zu setzen, damit alle Text-Zeichen im Code-Bereich von 20H...9FH zu definierten Ergebnissen führen. Sollen später im Programm-Ablauf einzelne Text-Zeichen oder Zeichengruppen verändert werden, so genügt es, wenn nur noch diese Bytes neu gesetzt werden.

Mit der ESC-Sequenz "ESC-m" kann zwischen internem und externem Zeichen-Generator umgeschaltet werden. Gleichzeitig bestimmt dieses Kommando den Modus der Verknüpfung von Text- und Grafikpixeln.

Programmbeispiel:

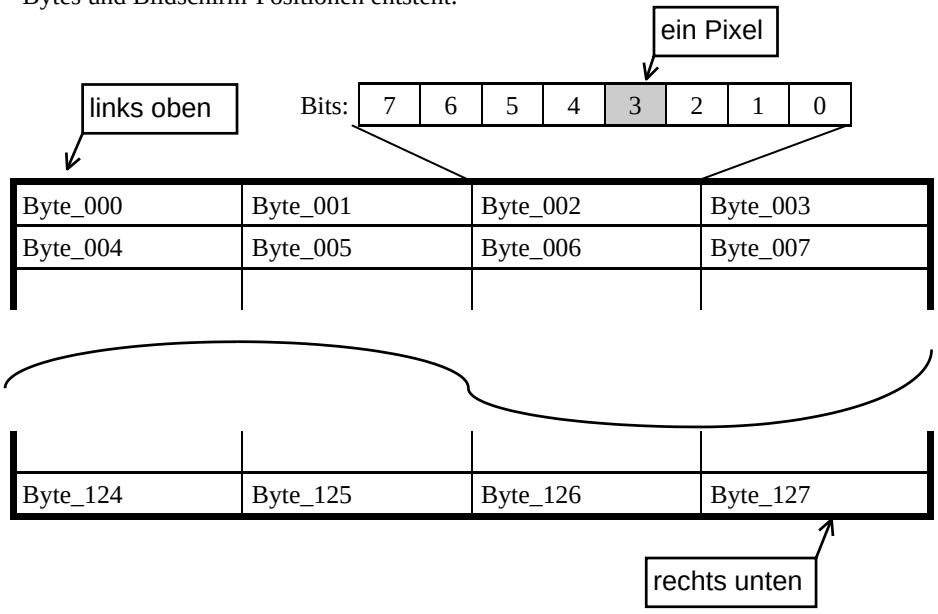
```
'------------------------------------------------------------------------
' Name: T69SPEC1.TIG
'------------------------------------------------------------------------
STRING M$(8), S$(8)                     ' String f. Sonderz.definition

TASK MAIN                               ' Beginn Task MAIN
  INSTALL DEVICE #1, "LCD-6963.TDD"     ' graf. Display installieren

  PRINT #1, "<1Bh>m<0><0F0h>"           ' interner Zeichensatz + '123'
  PRINT #1, " 1 1 1 1 1"                ' Ausgabe, die sich nach
  WAIT_DURATION 1000                    ' 1 Sek aendert

  S$ = "&
. . . . . . . . &                       ' Leerzeichen
. . . . . . . . &
. . . . . . . . &
. . . . . . . . &
. . . . . . . . &
. . . . . . . . &
. . . . . . . . &
. . . . . . . . "B

  M$ = "&
. . . . . * . . &                       ' Maennchen
. . . . * * * . &
. . . . . * . . &
. . . * * * * * &
. . . . . * . . &
. . . . * . * . &
. . . * . . . * &
. . . . . . . . "B

  PUT #1, #3, S$                        ' setze Sonderzeichen an ' '
  PUT #1, #3, M$, (31h-20h)*8           ' setze Sonderzeichen an '1'

  PRINT #1, "<1Bh>m<8><0F0h>"           ' externer Zeichensatz
END                                     ' Ende Task MAIN
```

Grafik des T6963

Der Grafikbildschirm der LC-Displays mit dem Controller T6963C von Toshiba oder mit einem kompatiblen Controller ist je nach Größe aus einer verschieden großen Anzahl von Grafikpunkten ausgestattet. Die Bildpunkte (auch Pixel oder Dots genannt) werden bitweise angesteuert:

Bit = 0	hell = Hintergrund = nicht gesetzter Punkt
Bit = 1	dunkel = gesetzter Punkt

Bytes bilden jeweils 8 waagerechte Dots auf dem LCD, wobei das höchstwertige Bit links liegt und das Bit-0 rechts. Der Graphic-Speicher des LCD beginnt in der linken oberen Ecke, so daß z.B. bei einem 32x32-Dots-Bildschirm folgende Zuordnung von Bytes und Bildschirm-Positionen entsteht:



Ausgabe auf den Grafikbildschirm

PUT #D, #1, *pixel_string* [, *lcd_offset*, *src_offset*, *src_len*]

D	ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.
#1	Sekundär-Adresse = 1: es wird Grafik ausgegeben
pixel_string	ist eine globale oder tasklokale Variable oder eine Konstante vom Typ STRING und enthält die Source-Daten der Grafik, die auf dem LCD gezeigt wird.
lcd_offset	ist eine Variable, Konstante oder ein Ausdruck vom Typ BYTE, WORD oder LONG und gibt den Byte-Offset im LCD-Grafik-RAM an. Die linke obere Bildschirm-Ecke hat den Offset 0.
src_offset	ist eine Variable, Konstante oder ein Ausdruck vom Typ BYTE, WORD oder LONG und gibt den Byte-Offset im **pixel_string** an. Es kann so auch nur ein Teil der Grafikdaten ausgegeben werden.
scr_len	ist eine Variable, Konstante oder ein Ausdruck vom Typ BYTE, WORD oder LONG und gibt die Anzahl der Bytes an, die ab **src_offset** ausgegeben werden sollen.

Einige Beispiele für Grafikausgabe-Instruktionen:

```
PUT #7, #1, A$          ' Dots beginnend linke obere Ecke ausgeben
PUT #7, #1, B$,64       ' Ueberspringe 64 x 8 Dots im LCD
PUT #7, #1, C$,0,0,128  ' 128 x 8 Dots oben auf LCD anzeigen
```

Die Ausgabe der Grafik erfolgt ausschließlich mit der Instruktion PUT, nie mit PRINT. Der Pixelstring muß immer existieren! Nicht erlaubt sind also Variable, die nur vorübergehend leben, wie lokale Strings (in Unterprogrammen) oder temporäre Strings (Expressions). **Richtig: globale oder Task-lokale Strings**.

!

Der Treiber schreibt Grafikdaten abwechselnd in einen von 2 internen Grafikpuffern. Das hat den Vorteil, daß auch bei schnellen Bildwechseln alle Pixel gleichzeitig sichtbar werden. Durch diese Strategie gibt es also stets 2 interne Bild-Speicher:

- der sichtbare und zuletzt mit PUT beschriebene Grafikspeicher,
- der vorletzte Grafikspeicher, der auch derjenige sein wird, der als nächstes angezeigt wird

Wird nicht der gesamte Bildschirm beschrieben, so enthalten jene Bildpunkte, die nicht neu geschrieben werden, noch die Werte wie bei der VOR-letzten Grafikausgabe und nicht die Punkte der gerade angezeigten Grafikausgabe. So zeigt die folgende Darstellung, daß nach dem Schreiben eines Teils der Ausgabe im Schritt 3 nicht die vorher sichtbaren ‘44444’ als ‘55444’ erscheinen, sondern ‘55333’:

Schritt	vor beschreiben unsichtbar	nach beschreiben sichtbar
		Puffer a: 22222
	Puffer b: 11111	
1. schreibe: 33333		Puffer b: 33333
	Puffer a: 22222	
2. schreibe: 44444		Puffer a: 44444
	Puffer b: 33333	
3. schreibe: 55		Puffer b: 55333
	Puffer a: 44444	

Wenn pixel_string=““ (leer) ist, dann wird nur der interne Puffer umgeschaltet.

Bei Graphikausgabe handelt es sich in der Regel um sehr viel mehr Daten als bei Textausgaben. Daher arbeitet dieser Device-Driver mit einer anderen Zugriffs-Methode als z.B. der Treiber ‘LCD1.TDD’.

Statt Daten-Bytes von einem Quellpuffer (im BASIC Programm) zu einen Zielpuffer (im Device-Treiber) zu transportieren, wird hier ein ‘direkter Zugriff’ ausgeführt. Bei dieser Methode werden mit der PUT-Instruktion keinerlei Datenbytes übertragen, sondern der Device-Treiber erhält einen **Zeiger** auf den Ort (Daten-String), an dem die Quelldaten vorliegen.

Diese Methode hat einige Vorteile, insbesondere wird Speicherplatz gespart und die Programmausführung ist viel schneller. Unmittelbar nachdem die PUT-Instruktion ausgeführt worden ist, ist der Zeiger zwar übergeben, aber es sind noch keine Daten gesendet worden. Dennoch wird bereits die nächste BASIC-Instruktion ausgeführt, während nun der Device-Treiber die Daten an das Grafik-LCD überträgt. Bei diesem Treiber ist also folgendes zu berücksichtigen:

Der Quellpuffer (String) muß immer existieren, d.h. global oder Task-lokal sein. Nicht erlaubt sind also Variable, die nur vorübergehend leben, wie:lokale Strings (in Unterprogrammen) oder temporäre Strings (Ausdrücke).

Verändert man den Quellstring bereits während der Device-Treiber noch Datenbytes überträgt, beeinflußt man den Output, ohne es zu wollen. Es muß also entweder

genügend Zeit vergehen oder die Output-Pufferfüllung des Device-Treibers abgefragt werden.

Programmbeispiel:


```
'------------------------------------------------------------------------
' Name: T69_GR1.TIG
'------------------------------------------------------------------------
STRING G$(1k)                           ' String f. Grafik 128x64 bit

TASK MAIN                               ' Beginn Task MAIN
#INCLUDE TIGHEAD.INC                    ' Include besetzt G$
  INSTALL DEVICE #1, "LCD-6963.TDD"     ' graf. Display installieren

  PRINT #1, "<1Bh>G<1><0F0h>"           ' Grafik einschalten
  PRINT #1, "<1Bh>T<0><0F0h>"           ' Text ausschalten
  PUT #1, #1, G$                        ' Grafik ausgeben
END                                     ' Ende Task MAIN
```

Grafik-LCD Funktionen

Die zahlreichen Funktionen, die speziell für die Programmierung des Grafik-LCD entwickelt wurden, sind in der folgenden Tabelle zusammen aufgeführt und kurz erläutert. Im Programmierhandbuch finden Sie ein Kapitel, welches nur der Grafik gewidmet ist..

Funktionsname	Pixel-grafik	Vektor-grafik	
OR2	•		2 Grafiken werden mit unbedingtem OR überlagert
OR3	•		3 Grafiken werden mit unbedingtem OR überlagert
OR4	•		4 Grafiken werden mit unbedingtem OR überlagert
AND2	•		2 Grafiken werden mit unbedingtem AND überlagert
AND3	•		3 Grafiken werden mit unbedingtem AND überlagert
AND4	•		4 Grafiken werden mit unbedingtem AND überlagert
XOR1	•		2 Grafiken werden mit unbedingtem XOR verknüpt
GRAPHIC_MASK_COPY	•		2 Grafiken werden über Maske verknüpft
GRAPHIC_MIRROR	•		Spiegeln einer Grafik an: X-, Y- odet X+Y Achsen
GRAPHIC_EXP_STRI	•		Expandiert Grafik-Bytes
GRAPHIC_COPY	•		Kopiert Grafik-Fenster
GRAPHIC_FILL_MASK	•		Füllt Rechteck-Maske in Grafik
INVERT	•		Invertiere Pixelwerte
DRAW_LINE		•	Beginnt Linie in Grafikbereich
DRAW_NEXT_LINE		•	Setzt Linie in Grafikbereich fort
CLOSE_LINE		•	Schließt Line
END_LINE		•	Beendet Line
SET_ROTATION		•	Setzt Rotationswinkel 0,01-Grad

Funktionsname	Pixel-grafik	Vektor-grafik	
SET_SCALE		•	Setzt Maßstab (100% = 1000)
SET_BASE		•	Setzt Basispunkt X/Y
SET_GRAREA		•	Setzt Grafikbereich: Destin + B x H
SET_DOT		•	Setzt DOT zu 0 or 1
FILL_AREA		•	Füllt Grafikbereich
Zubehör-Funktionen:			
DISTANCE			Berechnet Entfernung zweier Koordinaten
QUICK_WORD_SIN			Berechnet schnellen Sinus in WORD
QUICK_WORD_COS			Berechnet schnellen Cosinus in WORD

MF-II-PC-Tastatur

Der Device-Treiber ‘MF2_xxxx’ ermöglicht den Anschluß einer PC-Tastatur des Typs MF-II. Als externe Bauteile werden dazu neben der Tastaturbuchse lediglich 2 Widerstände benötigt.

Dateiname: MF2_8xpp.TDD

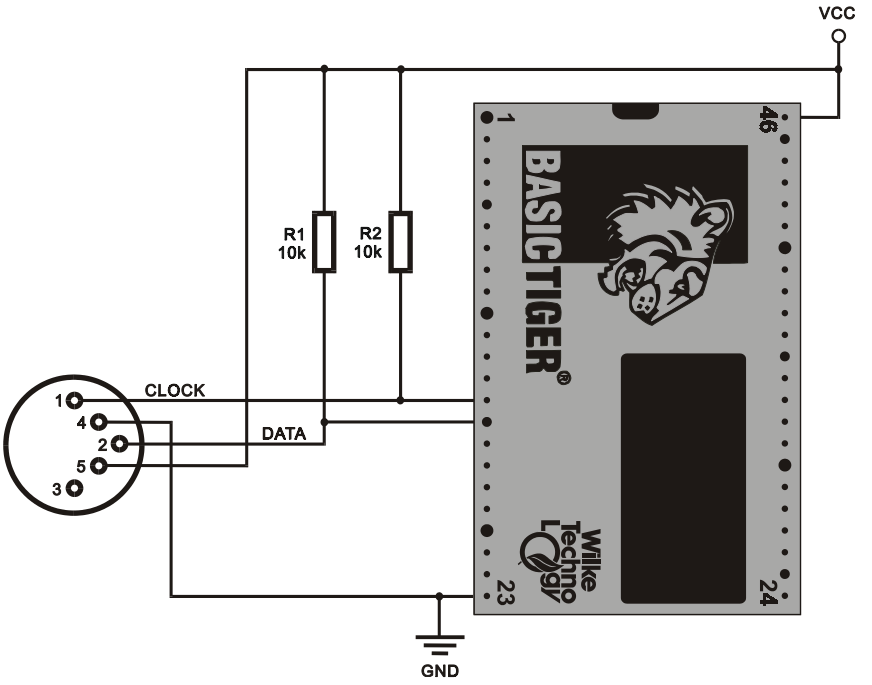
INSTALL DEVICE #*D*, "MF2_8xPp.TDD"

D	ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.
x	im Dateinamen gibt den Pin für den Anschluß der Clock-Leitung der Tastatur an.
Pp	im Dateinamen steht für: P: interner Port p: Pin für Data-Leitung der Tastatur.

Die Clock- und die Data-Leitung wird mit einem Pull-up-Widerstand gegen VCC versehen. Die Stromversorgung der Tastatur erfolgt an der Anschlußbuchse. Die Stromaufnahme entnehmen Sie den Datenblättern der Tastatur.

Größe und Füllstand des Eingangspuffers sowie die Version des Treibers können mit Hilfe der User-Function-Codes abgefragt werden.

Beispiel eines Anschlusses einer MF-II-Tastatur:



Da eine MF-II-Tastatur keine ASCII-Codes sendet, sondern weitere Umcodierungsschritte erfordert, sind recht umfangreiche Maßnahmen erforderlich, um die gewünschte Tastaturfunktion zu erhalten. Als Ausgangsbasis werden neben dem Beispielprogramm einige Include-Dateien mitgeliefert, die bei Bedarf an eigene Bedürfnisse angepaßt werden können. MF2_TR.INC wird als einzige Include-Datei in die Anwendung eingebunden. MF2_TR.INC bindet selbst alle weiteren Include-Datei ein.

Die Anwendung ruft Unterprogramme auf, die in den Dateien MF2_TR.INC, MF2_TR_D.INC stehen. Hier findet die Konversion zu ASCII statt. Im nächsten Layer MF2_PH.INC, MF2_PH_D.INC wird die (physikalische) Verbindung zum Treiber und damit zur Tastatur hergestellt.

Die Initialisierung ‚InitKeybTables' wird vor der Verwendung der Tastatur einmal aufgerufen. Als Argument wird die Nummer der Spache übergeben (1=englisch, 2=deutsch, 3=englisch und deutsch).

Das Unterprogramm ‚InitKeybDev' mit der Device-Nummer als Argument (WORD) wird ebenfalls einmal aufgerufen. Wenn der Treiber mehrfach eingebunden ist, dann wird auch ‚InitKeybDev' jeweils mit der Device-Nummer mehrmals aufgerufen.

Das Unterprogramm ‚**GetAsciiKey**' liefert in einem WORD:

- wenn kein Zeichen 0000h
- wenn ASCII-Zeichen im low-Byte das Zeichen, im high-Byte den Scan-Code
- wenn Taste mit extended Code im low-Byte 0, im high-Byte den Scan-Code

Das Unterprogramm ‚**CheckKeybFlags**' gibt Auskunft über den momentanen Zustand der Sondertasten wie STRG, ALT, SHIFT, etc.

Byte 0

Bit 0: rechte Shift-Taste gedrückt
Bit 1: linke Shift-Taste gedrückt
Bit 2: Strg-Taste gedrückt
Bit 3: ALT-Taste gedrückt
Bit 4: Scroll-Lock ist aktiv
Bit 5: Num-Lock ist aktiv
Bit 6: Caps-Lock ist aktiv
Bit 7: Einfügen ist aktiv

Byte 1

Bit 0: linke Strg-Taste gedrückt
Bit 1: linke ALT-Taste gedrückt
Bit 2: System-Request ist gedrückt
Bit 3: Pause-Taste ist getoggelt
Bit 4: Scroll-Lock-Taste gedrückt
Bit 5: Num-Lock-Taste gedrückt
Bit 6: Caps-Lock-Taste gedrückt
Bit 7: Einfügen-Taste gedrückt

Byte 2 (LED-Anzeige)

Bit 0: Scroll-Lock-Anzeige
Bit 1: Num-Lock-Anzeige
Bit 2: Caps-Lock-Anzeige
weitere Bits nicht verwendet.

Byte 3

Bit 0: letzter Code war der ‚E1 hidden code'
Bit 1: letzter Code war der ‚E0 hidden code'
Bit 2: rechte Strg-Taste gedrückt
Bit 3: rechte ALT-Taste gedrückt
weitere Bits nicht verwendet.

In ‚MF2_PH.INC' sind einige nützliche Unterprogramme enthalten:

Unterprogramm (Argumente)	Funktion
ResetKbd (WORD wDevId)	RESET Keyboard
SetKbdTypematicRate (WORD wDevId; BYTE bTpRate)	setzt Typematic rate der MF-II Tastatur
SetKbdIndicators (WORD wDevId; BYTE bLEDsMask)	setzt die LED-Anzeige der MF-II Tastatur (bLEDsMask, 0=aus, 1=an): Bit 0: Scroll-Lock Bit 1: Num-Lock Bit 2 Caps-Lock
ClearKbdBuffer (WORD wDevId)	löscht den MF-II Tastaturpuffer
GetKbdScanCode (WORD wDevId; VAR BYTE bCode)	holt ein Zeichen aus dem Tastaturpuffer. Wenn der Puffer leer ist, bleibt ‚bCode' unverändert.
SetKbdScanCodeTable (WORD wDevId; BYTE bTableId)	setzt in ‚bTableId' die Scan-Code-Tabelle für die Tastatur
GetKbdBufferFillSize (WORD wDevId;VAR LONG lBufSize)	liest den Füllstand des Tastaturpuffers

Alle Unterprogramme zur MF-II-Tastatur sind re-entrant, d.h. mehrere Tasks können sie gleichzeitig benutzen.

Anmerkung: Scan-Code-Sets: Die MF-II-Unterprogramme sind für den Scan-Code-Set 1 geschrieben.

Das folgende Beispielprogramm zeigt, daß die Verwendung der Tastatur durch die mitgelieferten Include-Dateien aus Anwendersicht einfach geworden ist.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name: MF2_1.TIG
' Zeigt die Verwendung einer MF-II-Tastatur am BASIC-Tiger
'--------------------------------------------------------------------
' 4 Leitungen der Tastatur anschliessen:
'               MF-II
'                 GND  <---->    GND
'     +5V  <---->  Vcc
'   CLOCK  <---->  L80 plus 10...22KOhm --> Vcc
'    DATA  <---->  L81 plus 10...22KOhm --> Vcc
'
'--------------------------------------------------------------------
user var strict                   ' unbedingte Var.deklaration
#include UFUNC3.INC               ' User Function Codes
#include DEFINE_A.INC             ' allg. Symbol-Definitionen
#include MF2_TR.INC               ' subroutines of the Transport Layer

WORD wKeybDevId1                  ' Tastatur-Device-Nummer
LONG lKeybExtFlags1               ' Tastatur-Flags
BYTE bKeybActLang1                ' Tastatur-Layout (Sprache)

'--------------------------------------------------------------------
TASK Main
  WORD wKey                       ' Taste (WORD)
  BYTE bIsActive                  '
  LONG lComplexMask               '

' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #LCD, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #LCD, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8

  INSTALL DEVICE #KEYB1, "MF2_8081.TDD" ' L80=clock,  L81=data

  wKeybDevId1 = KEYB1                                    '
initialisiere Tastaturvariable      init
  lKeybExtFlags1 = 0
  bKeybActLang1 = LANG_GERMAN
'  bKeybActLang1 = LANG_ENGLISH

  CALL InitKeybTables( bKeybActLang1 ) ' Initstufe 1
  CALL InitKeybDev( wKeybDevId1 )            ' Initstufe 2

  LOOP 9999999                    ' viele loops
    ' Lies ein Zeichen aus dem Puffer, nach ASCII uebersetzt
    CALL GetAsciiKey(wKeybDevId1, lKeybExtFlags1, bKeybActLang1, wKey)
    IF wKey <> 0 THEN             ' wenn gueltiges Zeichen
      PRINT #LCD, CHR$(wKey);     ' auf LCD anzeigen
    ENDIF
  ENDLOOP
END
```

Parallele Ausgabe: Drucker

Der Device-Treiber ‘PRN1’ ermöglicht den Anschluß eines parallelen Druckers ohne weitere Bauteile. Der Device-Treiber ‘PIN1’ realisiert mit Hilfe weniger Bauteile einen parallelen Eingang.

Dateiname: PRN1B_K8.TDD (mit 8K Puffern)
PRN1B_K1.TDD (mit 1K Puffern)
PRN1B_R1.TDD (mit 256 Byte Puffern)

INSTALL DEVICE #*D*, "PRN1.TDD" [, *P1*, ..., *P7*]

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Datenbus **L60...L67** ist der **Datenbus** (der gleiche Bus, der auch für LCD, erweiterte I/O-Pins und Tastatur verwendet wird).

Busy Pin **L70** ist der Input-Pin für **busy**.

Strobe Pin **L71** ist der Output-Pin für **strobe**.

P1...P7 sind weitere Parameter, die die Standard-Anschlußbelegung des PRN1-Treibers verändern. Die Reihenfolge der Parameterangabe muß eingehalten werden. Ebenso müssen Werte, die angegeben werden, auch stimmen. Es brauchen jedoch nicht alle 7 Parameter angegeben zu werden.

	Beschreibung des Parameters
P1	reserviert (immer 0 angeben)
P2	Anzahl der Burst-Zeichen (Standard ist 10, max. 20)
P3	Logische Port-Adresse des Datenbusses (Standard=6)
P4	Logische Portadresse für Signal ‘busy’ (Standard=7)
P5	True-Bitmaske für Signal ‘busy’, gibt Bit-Position an. (Standard=1)
P6	Logische Portadresse für Signal ‘strobe’ (Standard=7)
P7	True-Bitmaske für Signal ‘strobe’, gibt Bit-Position an. (Standard=2)

Falls die parallele Übertragung Schwierigkeiten bereitet, kann die Strobe-Länge experimentell verändert werden. Die Anzahl der Burst-Zeichen läßt sich für sehr schnelle Geräte einstellen. Es werden Zeichen sehr schnell in Gruppen (Burst) übertragen.

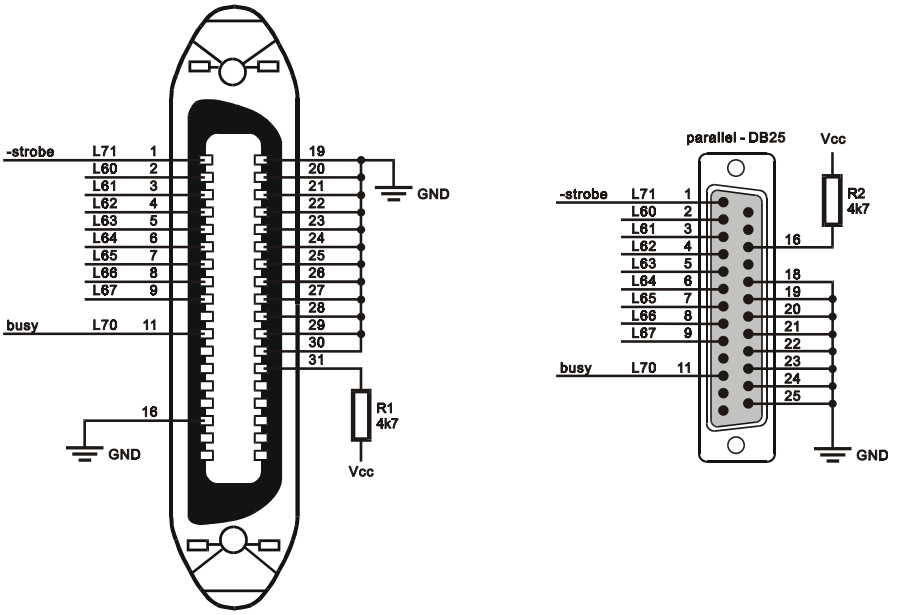
Der Treiber PRN1-Treiber und der LCD1-Treiber sowie die erweiterten I/Os lassen sich gleichzeitig betreiben. Daten, die auf diesen Device-Treiber ausgegeben werden, werden zunächst intern gepuffert. Die Ausgabe auf den Drucker erfolgt mit den Instruktionen PRINT, PRINT_USING oder PUT.

Tip

Um mehrere parallele Ausgänge zu realisieren, wird der Treiber PRN1.TDD mehr als einmal eingebunden. Die Pins für '-strobe' und 'busy' werden für jede Treiberinstanz getrennt eingestellt.

Größe, Füllstand oder verbleibender Platz des Ausgangspuffers sowie die Version des Treibers können mit Hilfe der User-Function-Codes abgefragt werden (Seiten 16ff).

Beispiel eines 36-poligen Centronics-Anschlusses:



Ziehen Sie Ihr Druckerhandbuch zu Rate, um festzustellen, ob der Drucker auf weiteren Leitungen bestimmte Signalpegel erwartet. Der Treiber bedient den Datenbus, 'strobe' und beachtet das 'busy'-Signal.

Programmbeispiel:

```
'-------------------------------------------------------------------------
' Name: PRN1.TIG
'-------------------------------------------------------------------------
TASK MAIN                                ' Beginn Task MAIN
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  INSTALL_DEVICE #5, "PRN1.TDD"          ' Drucker-Treiber installieren

  PRINT #1, "Attention: printing"        ' Auf LCD ausgeben
  PRINT #5, "Hello printer<12>"          ' Auf Drucker mit Formfeed
END                                      ' Ende Task MAIN
```

Leere Seite

Paralleler Eingang

Dateiname: PIN1.TDD (PIN1_.TDD mit kleineren Puffern)

INSTALL DEVICE #*D*, "PIN1.TDD" [, *P1*, ..., *P7*]

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Datenbus **L60...L67** ist der **Datenbus** (der gleiche Bus, der auch für LCD, erweiterte I/O-Pins und Tastatur verwendet wird).

Strobe Pin **L80** ist der Input-Pin für **Strobe**.

Read Pin **L81** ist der Output-Pin, der zum Lesen des Datenbytes auf low gelegt wird.

P1...P7 sind weitere Parameter, die die Standard-Anschlußbelegung des PRN1-Treibers verändern. Die Reihenfolge der Parameterangabe muß eingehalten werden. Ebenso müssen Werte, die angegeben werden, auch stimmen. Es brauchen jedoch nicht alle 7 Parameter angegeben zu werden.

In der externen Beschaltung des parallelen Eingangs wird das Datenbyte mit dem Strobe-Signal in ein Latch geschrieben. Gleichzeitig wird ein Flip-Flop gesetzt, welches dem Tiger-Modul anzeigt, daß ein Datenbyte im Latch ist. Aus dem gesetztem Flip-Flop wird ebenfalls das Busy-Signal für das externe Gerät erzeugt.

Das Tiger-Modul liest das Datenbyte und legt dazu die Read-Leitung auf ‘low’. Dadurch wird das Flip-Flop zurückgesetzt. Das Busy-Signal wird jedoch durch ein R-C-Glied noch eine Weile gehalten.

	Beschreibung des Parameters
P1	Anz. WAIT-Loops
P2	Anzahl der Burst-Zeichen
P3	Logische Port-Adresse des Datenbusses
P4	Logische Portadresse für Data-Valid (Stobe-Latch)
P5	True-Bitmaske für Data-Valid, gibt Bit-Position an.
P6	Logische Portadresse für Read-Signal
P7	True-Bitmaske für Read-Signal, gibt Bit-Position an.

Über den Parameter 1 wird eingestellt, wie lange noch in der Eingaberoutine gewartet wird, ob eventuell noch ein Zeichen kommt. Sonst dauert es 1msec bis das nächste

Zeichen geholt wird. Parameter 2 bestimmt, wieviele Zeichen in diesem Burst-Modus gelesen werden.

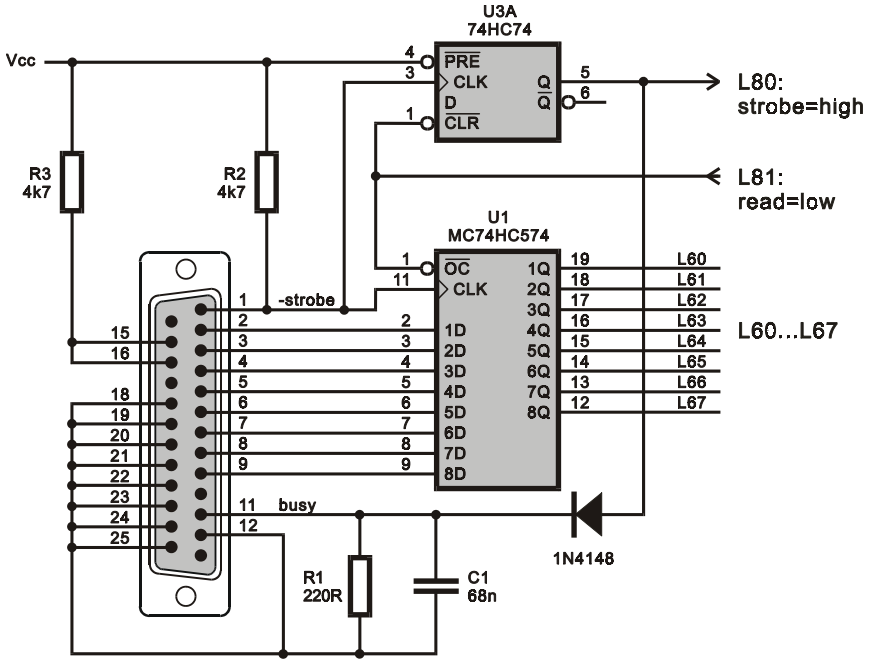
Die Treiber PIN1 (parallel-in), PRN1 (parallel-out), der LCD1-Treiber sowie die erweiterten I/Os lassen sich gleichzeitig betreiben. Daten, die auf dem parallelen Eingang eingelesen werden, speichert der Device-Treiber zunächst intern in einem Einganspuffer. Das Auslesen des Puffers erfolgt mit den Instruktionen GET, INPUT, oder INPUT_LINE.

Tip

Um mehrere parallele Eingänge zu realisieren, wird der Treiber PIN1.TDD mehr als einmal eingebunden. Die Pins für '-strobe' und 'busy' werden für jede Treiberinstanz getrennt eingestellt.

Größe, Füllstand oder verbleibender Platz des Eingangspuffers sowie die Version des Treibers können mit Hilfe der User-Function-Codes abgefragt werden (Seiten 16ff).

Beispiel eines 25-poligen parallelen Eingangs:



Die Haltezeit für 'busy' wird durch das R-C-Glied bestimmt und kann auch experimentell verändert werden.

Ziehen Sie das entsprechende Handbuch des sendenden Gerätes zu Rate, um festzustellen, ob auf weiteren Leitungen bestimmte Signalpegel erwartet werden. Der

Treiber benötigt den Datenbus und ‘strobe’. Die externe Beschaltung erzeugt das ‘busy’-Signal.

Die eingehenden Daten werden in einem Puffer gespeichert. Größe, Füllstand oder verbleibender Platz des Puffers können mit Hilfe der User-Function-Codes abgefragt werden (Seiten 16ff).

Neben den allgemeinen User-Function-Codes der I/O-Puffer gibt es für die Instruktion GET folgende Kommandos:

Nr	Symbol Prefix UFCI_	Beschreibung
1	UFCI_IBU_FILL	Füllstand des Eingangspuffers (Byte)
2	UFCI_IBU_FREE	freier Platz im Eingangspuffer (Byte)
3	UFCI_IBU_VOL	Größe des Eingangspuffers (Byte)
65	UFCI_LAST_ERRC	letzter Error-Code
99	UFCI_DEV_VERS	Version des Treibers

User-Function-Codes zum Setzen von Parametern (Instruktion PUT):

Nr	Symbol	Beschreibung
1	UFCO_IBU_ERASE	Eingangspuffer löschen
128	UFCO_SET_ISEP	setze Separatoren für INPUT
129	UFCO_RES_ISEP	lösche Separatoren für INPUT

Für die Instruktion INPUT gelten standardmäßig KOMMA und RETURN als Separatorzeichen, die eine Eingabe abschließen. Mit Hilfe des User-Function-Codes UFCO_SET_ISEP lassen sich die Separatorzeichen verändern. Bereits gesetzte Zeichen können gelöscht werden, bevor neue Zeichen gesetzt werden. Die zu setzenden oder zu löschenden Zeichen werden immer als **Code-Bereiche** angegeben:

PUT #D, #C, #UFCO_SET_ISEP, *Startcode*, *Endcode*, *Startcode*, *Endcode*

Wenn Sie die Standard-Separatoren löschen, jedoch keine neuen setzen, wird eine INPUT-Instruktion erst beendet, wenn der Input-Puffer voll ist.

Beispiel: setze neuen Separator LINE-FEED für die Instruktion INPUT auf dem Device-Treiber LCD1:

```
PUT #2,#0, #UFCO_RES_ISEP, 0, 255      ' loesche alle Separatoren
PUT #2,#0, #UFCO_SET_ISEP, 10, 10      ' setze Line-Feed als Separator
```

Beispiel: setze alle Steuerzeichen sowie Zeichen ab 7Fh als Separatorzeichen für die Instruktion INPUT auf dem Device-Treiber LCD1:

```
PUT #2, #0, #UFCO_RES_ISEP, 0, 255      ' loesche alle Separatoren
                                  ' setze neue Bereiche als Separatoren
PUT #2, #0, #UFCO_SET_ISEP, 0, 31, 127, 255
```

Beispiel: lösche Komma als Separatorzeichen für die Instruktion INPUT auf dem Device-Treiber LCD1:

```
PUT #2, #0, #UFCO_RES_ISEP, 2ch, 2ch    ' loesche Komma als Separatoren
' oder
PUT #2, #0, #UFCO_RES_ISEP, ',,'        ' loesche Komma als Separatoren
```

Ein weiteres Beispiel:

```
PUT #1, #0, #UFCI_ SET_ISEP, 'acXZ55'
' setzt als INPUT-Separatoren diese Zeichen:
'               a, b, c, X, Y, Z, 5
```

Programmbeispiel:

```
'-------------------------------------------------------------------------
' Name: PIN1.TIG
'-------------------------------------------------------------------------
TASK Main                                ' Beginn Task MAIN
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  INSTALL_DEVICE #2, "PIN1.TDD"          ' Parallel-In Treiber

  PRINT #1, "<1>Parallel In"
  LOOP 999999999                         ' sehr viele Loops
    GET #2, 1, A$                        ' einlesen von Parallel-Port
    IF A$ <> "" THEN                     ' Falls Inhalt,
      PRINT #1, A$;                      ' Ausgabe auf LC-Display
    ENDIF                                '
  ENDLOOP                                '
END                                      ' Ende Task MAIN
```

Leere Seite

Puls-Ein-/Ausgabe

Es gibt eine Reihe von Device-Treibern, die die Messung bzw. Zählung von Pulsen oder die Ausgabe von Pulsen betreffen. Einige dieser Device-Treiber verwenden gleichzeitig den Zeitbasis-Treiber TIMERA, ohne daß eine gegenseitige Behinderung auftritt. Es kann dann sogar der gleiche Treiber mehrmals eingebunden werden. Treiber, die mehrmals eingebunden werden, verwenden unterschiedliche Pins. Aus Gründen der Geschwindigkeit werden die Pins nicht durch Parameter gesetzt, sondern es gibt eine Serie von Treibern, die jeweils feste Pins verwenden. Aus dem Dateinamen des Device-Treibers geht hervor, welche Pins verwendet werden. Eine andere Gruppe von Device-Treibern belegen interne Ressourcen, so daß die gleichzeitige Nutzung anderer Device-Treiber, die dieselbe Ressource benötigen, ausgeschlossen ist. Folgende Tabelle schafft einen Überblick über den derzeitigen Stand:

Device-Treiber	**Funktion**	**Ressource**
ANALOG2	interner A/D-Wandler	TIMERA
CNT1_xxx	Pulszähler, Schnelligkeit und Auflösung nach TIMERA	TIMERA
ENC1_xxx	Encoder mit Richtungserkennung, Schnelligkeit und Auflösung nach TIMERA	TIMERA
FREQ1_xxx	Frequenzmessung, Schnelligkeit und Auflösung nach TIMERA	TIMERA
PLSIN1	Pulslängenmessung, schnell, hohe Auflösung	intern
PLS2_Pxx	Pulslängenmessung, Schnelligkeit und Auflösung nach TIMERA	TIMERA
PLSOUT1	Pulsausgabe, schnell, hohe Auflösung	intern
PO2_Pxx	Pulsausgabe, Schnelligkeit und Auflösung nach TIMERA	TIMERA
PWM2	interne PWM-Kanäle	TIMERA
zu Testzwecken		
SET1	setzt Pin auf high wenn TIMERA seine Arbeit beginnt.	TIMERA
RES1	setzt Pin auf low wenn TIMERA seine Arbeit beendet.	TIMERA

Device-Treiber

Der Zeitbasis-Treiber TIMERA kann auf maximal 12.5kHz eingestellt werden. Dadurch ergibt sich bei allen Puls-Device-Treibern, die den TIMERA verwenden, eine theoretische Obergrenze von 6.25kHz. Bei den Treibern, die Pulslängen oder Frequenzen messen, ist es nicht sinnvoll, den TIMERA während der Messung zu verstellen, da sonst in den Ergebnispuffern Werte entstehen, deren Einheit nicht bekannt ist. Nähere Informationen zum Treiber TIMERA finden Sie ab Seite 349.

Pulse zählen

Dieser Device-Treiber zählt die Anzahl der steigenden oder fallenden oder beider Flanken an einem oder zwei Pins. Der Zähler ist eine LONG-Zahl. Der momentane Wert kann jederzeit ausgelesen werden. Bei der Installation des Treibers wird durch den Dateinamen festgelegt, welcher Pin für die Zählung verwendet wird. Die Auflösung des Zählers ist durch die Einstellung des TIMERA festgelegt.

Dateiname: CNT1_**Ppp**.TDD

INSTALL DEVICE #*D*, "CNT1_Ppp.TDD", *P1, P2*

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Ppp im Dateinamen steht für:
P: interner Port
pp: erster Pin und zweiter Pin.

P1 ist ein Parameter, der die Benutzung der beiden Pins näher festlegt:

P1	Modus
0	nur Pulse am 1. Pin werden gezählt
1	Pulse an beiden Pins werden gezählt
2	Pulse am 1. Pin werden gezählt, der 2. Pin legt die Zählrichtung fest, positiver Pegel am 2. Pin bedeutet 'aufwärts zählen'.
3	Pulse am 1. Pin werden gezählt, der 2. Pin legt die Zählrichtung fest, negativer Pegel am 2. Pin bedeutet 'aufwärts zählen'.

P2 ist ein Parameter, der festlegt, welche Flanken der Zähler zählt:

P2	
0	nur steigende Flanken werden gezählt
1	nur fallende Flanken werden gezählt
2	beide Flanken werden gezählt

Hinweis: TIMERA.TDD muß vorher eingebunden werden.

Die Pulszählung wird durch Ausgabe eines Dummy-Wertes gestartet:

PUT #D, *1000*

Der Device-Treiber stellt bei zweikanaligem Zählen zwei LONG-Zählerstände zur Verfügung. Die Zählerstände können sowohl synchron als auch asynchron gelesen werden. Der Lesevorgang kann auf Wunsch die Zähler auf 0 zurücksetzen.

Die verschiedenen Lesemodi sind folgendermaßen den Sekundär-Adressen des Treibers zugeordnet:

Sek.-Adr.	Lesevorgang
0	Es wird der Zählerstand des ersten Zählers gelesen und eine Kopie des 2. Zählerstandes erzeugt.
1	Es wird die zuletzt erzeugte Kopie des zweiten Zählerstandes gelesen. Der laufende 2. Zählerstandes kann bereits weitergelaufen sein.
2	Es wird der Zählerstand des ersten Zählers gelesen und eine Kopie des 2. Zählerstandes erzeugt. Außerdem werden beide Zähler auf 0 gesetzt.
3	Es wird der Zählerstand des zweiten Zählers gelesen.

Beispiel:

```
GET #D, #0, 0, Z1              ' lies Zaehler 0 und erzeuge
                               ' Kopie des 2. Zaehlers
```

In der Include-Datei ‘UFUNCn.INC’ sind User-Function-Codes des Device-Treibers CNT1 definiert. Hier die UFCs für Input (Instruktion GET):

Nr	Symbol Prefix: UFCI_	Beschreibung
65	UFCI_LAST_ERRC	letzter Error-Code
99	UFCI_DEV_VERS	Version des Treibers

Die User-Function-Codes des Device-Treibers CNT1 für Output (Instruktion PUT):

Nr	Symbol Prefix: UFCO_	Beschreibung
131	UFCO_CNT_EDGE	setze Flankenzählung: 0: steigende Flanke 1: fallende Flanke 2: beide Flanken
149	UFCO_CNT_STOP	stoppe Zählung der Flanken

Beispiel:

```
PUT #D, #0, #UFCO_CNT_STOP, 0     ' stoppe Zaehlung der Flanken.
```

Das folgende Beispielprogramm zählt Pulse bei steigenden Flanken an Pin L80. Die Zählung erfolgt positiv, d.h. aufwärts, wenn Pin L81 auf 'high' liegt und negativ, d.h. abwärts, wenn Pin L81 auf 'low' liegt.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name: CNT1.TIG
'--------------------------------------------------------------------
TASK Main                                ' Beginn Task MAIN
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  INSTALL_DEVICE #2, "TIMERA.TDD",1,250 ' Zeitbasis 10kHz
  INSTALL_DEVICE #3, "CNT1_801.TDD",2,2 ' Pulszaehler

  PRINT #1, "<1>Pulses on L80"
  PRINT #1, "count direction L81"       ' low = aufwaerts
  PUT #3, 0                             ' beginne zu zaehlen
  LOOP 999999999                        ' sehr viele Loops
    GET #3, #0, 0, N                    ' Anz. Pulse einlesen
    PRINT #1, "<1BH>A<0><2><0F0H>pulses L80:";N;"  ";
    GET #3, #1, 0, N                    ' Anz. Pulse einlesen
    PRINT #1, "<1BH>A<0><3><0F0H>pulses L81:";N;"  ";
    WAIT_DURATION 100
  ENDLOOP                               '
END                                     ' Ende Task MAIN
```

Encoder

Dieser Device-Treiber ermöglicht den Anschluß eines Encoders. Encoder liefern bei Drehung 2 Pulsketten, in denen eine Drehrichtungsinformation steckt. Es gibt keinen Endanschlag und ebenso keine absolute Stellung. Der Device-Treiber ENC1 zählt die Pulse unter Berücksichtigung der Richtung in einem LONG-Zähler. Der momentane Wert kann jederzeit ausgelesen werden. Bei der Installation des Treibers wird durch den Dateinamen festgelegt, an welche Pins der Encoder angeschlossen ist. Die maximale Drehgeschwindigkeit des Encoders, ohne das Pulse verloren gehen, ist durch die Einstellung des TIMERA festgelegt. Es kann eine Dynamik in den Treiber eingebaut werden.

Weiter Informatione zum ENC1_xx.TDD

- Sekundär-Adressen des ENC1_xx.TDD
- User-Function-Codes des PLSOUT1.TDD
- Encoder-Drehgeber anschließen
- Dynamisches Bewerten der Drehung

Dateiname: ENC1_**Ppp**.TDD

INSTALL DEVICE #*D*, "ENC1_Ppp.TDD"

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Ppp im Dateinamen steht für:
P: interner Port
pp: erster Pin und zweiter Pin.

Hinweis: TIMERA.TDD muß vorher eingebunden werden.

Der Encoder-Device-Treiber stellt einen LONG-Zähler zur Verfügung, der bei Drehung in eine Richtung aufwärts, bei Drehung in die andere Richtung abwärts zählt. Der Anfangszählerstand des Encoders wird durch eine einfache PUT-Instruktion gesetzt. Gleichzeitig wird dadurch die Zählung gestartet:

```
PUT #17, 0            ' setze Resetwert 0
PUT #17, 12345        ' setze Resetwert 12345
PUT #17, L            ' L = LONG
```

Beim Auslesen des Zählerstandes bleibt der Zählerstand entweder erhalten oder er wird auf den zuvor gesetzten Anfangsstand zurückgesetzt. Auf den Sekundär-Adressen 2 und 3 wird der dynamisch entstandene Wert ausgelesen. Bei schneller Drehung des Drehgebers werden mehr Schritte gerechnet, als der Drehgeber wirklich erzeugt hat.

Sekundär-Adressen des ENC1_xx.TDD

Es gibt im Device-Treiber zwei Zähler, einen normalen und einen dynamischen, die auf verschiedene Weise gelesen werden:

Sek.-Adr.	Lesevorgang (GET)
0	Es wird der Zählerstand des normalen Zählers gelesen
1	Es wird der Zählerstand des normalen Zählers gelesen und beide Zähler auf den zuletzt gesetzten Resetwert zurückgesetzt.
2	Es wird der Zählerstand des dynamischen Zählers gelesen
3	Es wird der Zählerstand des dynamischen Zählers gelesen und beide Zähler auf den zuletzt gesetzten Resetwert zurückgesetzt.
4	Es werden in 8 Bytes beide Zählerstände in einen String gelesen: low 4 Bytes: Zählerstand des normalen Zählers high 4 Bytes: Zählerstand des dynamischen Zählers siehe Funktion: NFROMS
5	Es werden in 8 Bytes beide Zählerstände in einen String gelesen und beide Zähler auf den zuletzt gesetzten Resetwert zurückgesetzt: low 4 Bytes: Zählerstand des normalen Zählers high 4 Bytes: Zählerstand des dynamischen Zählers siehe Funktion: NFROMS
6	Es wird der Geschwindigkeits-Index der Drehung gelesen: 0: dreht sehr schnell ... 255: steht
Sek.-Adr.	**Schreibvorgang (PUT)**
0	Der Resetwert wird gesetzt und die Zählung (bei 0) gestartet. Der Resetwert wird in den Zähler gesetzt, wenn dieser mit Rücksetzen ausgelesen wird.
1	Ein neuer Resetwert wird gesetzt. Die Zählung wird dadurch weder gestartet noch gestoppt.
2	Ein neuer Wert des dynamischen Zählers wird gesetzt. Die Zählung wird dadurch weder gestartet noch gestoppt.

User-Function-Codes des ENC1.TDD

In der Include-Datei 'UFUNCn.INC' sind User-Function-Codes des Device-Treibers ENC1 für Input definiert (Instruktion GET):

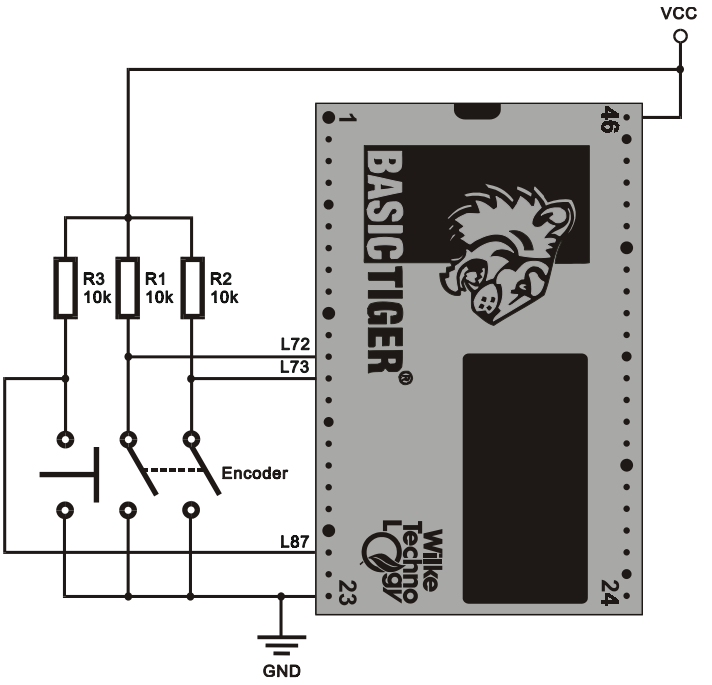
Nr	Symbol Prefix: UFCI_	Beschreibung
65	UFCI_LAST_ERRC	letzter Error-Code
99	UFCI_DEV_VERS	Version des Treibers

User-Function-Codes für Output (Instruktion PUT):

Nr	Symbol Prefix: UFCO_	Beschreibung
128	UFCO_WEIGHTTAB	Setzt Gewichtungstabelle. Es werden immer 256 Bytes neu gesetzt. Nicht übergebene Bytes werden auf 1 gesetzt.
129	UFCO_SCALEFCT	Setzt Zeit-Skalierungsfaktor. Ein größerer Wert ermöglicht langsame Drehgeber, indem Bei dem gesamten Ablauf die zeitlichen Vorgänge gestreckt werden.
133	UFCO_ENC_STOP	Stoppt die Zählung
134	UFCO_ENC_START	Startet die Zählung ohne einen Resetwert zu setzen
135	UFCO_ENC_DIR	Setzt die Drehrichtung des Drehgebers: 0: positiv <> 0: negativ

Encoder-Drehgeber anschließen

Die Hardware des Encoders besteht aus zwei Schaltern, die bei Drehung zeitversetzt schalten. Bei Druck auf die Achse kann bei vielen Decodern ein Taster betätigt werden. Der Device-Treiber ENC1 wertet jedoch nur die Drehung aus, der Drucktaster kann als Teil einer Tastatur verdrahtet werden.



Das folgende einfache Beispiel zeigt die grundlegende Benutzung des Encoder Device-Treibers.

Programmbeispiel:

```
'------------------------------------------------------------------------
' Name: ENC1.TIG
'------------------------------------------------------------------------
#INCLUDE UFUNC3.INC                      ' User-Function-Definitionen
TASK Main                                ' Beginn Task MAIN
  LONG P
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  INSTALL_DEVICE #2, "TIMERA.TDD",1,250 ' Zeitbasis 10kHz
  INSTALL_DEVICE #3, "ENC1_801.TDD"     ' Encoder

  PRINT #1, "<1>Encoder on L80/81"
  PUT #3, 0                              ' starte Encoder
  LOOP 999999999                         ' sehr viele Loops
    GET #3, #0, 4, P                     ' Encoderstellung einlesen
    PRINT_USING #1, "<1BH>A<0><1><0F0H>Encoder:";P;"  ";
    WAIT_DURATION 100
  ENDLOOP                                '
END                                      ' Ende Task MAIN
```

Dynamisches Bewerten der Drehung

Im ersten Beispielprogramm wird die Anzahl der Schritte ausgelesen, die der Encoder wirklich durchlaufen hat. Wenn mit Hilfe des Encoders große Wertebereiche eingestellt werden sollen, jedoch gleichzeitig bei gefundener grober Einstellung eine Feinjustierung vorgenommen werden muß, dann ist das reine Auslesen der stattgefundenen Schritte nicht ausreichend. Ohne Dynamik müßte man sehr lange drehen, bis ein großer Bereich durchfahren ist.

Dynamik bedeutet bei dem Drehgeber, daß aus einem physikalischen Schritt mehrere logische Schritte gemacht werden, wenn die Drehgeschwindigkeit hoch ist. Um sehr langsame und sehr schnelle Encoder anpassen zu können, werden die langen Zeiten des langsamen Encoders mit einem Teiler heruntergeteilt, so daß intern für die Benutzung der Dynamiktabelle wiederum entsprechend kurze Zeiten zur Verfügung stehen.

Die Anpassung eines Encoders wird durch Probieren nach Gefühl vorgenommen, da auch mechanische Gegebenheiten wie z.B. die Größe und Art des Drehknopfes eine Rolle spielen. In der folgenden Tabelle sind die Zeiten pro Schritt für zwei verschiedene handbediente Encoder dargestellt. Bei dem groben Encoder mit nur 6 Pulsen pro Umdrehung dauert ein Schritt selbst bei schneller Drehung noch länger als bei der langsamen Drehung des feinen Encoders. Der Teilerfaktor gleicht die Zeiten wieder ab:

	wenig Pulsen (6/Umdrehung)	vielen Pulsen (180/Umdrehung)
schnell	3 U/sec = 55.5 ms pro Step	3 U/sec = 1.8 ms pro Step
mittel	1 U/sec = 167 ms pro Step	1 U/sec = 5.6 ms pro Step
langsam	1/6 U/sec = 1000 ms pro Step	1/6 U/sec = 33.3 ms pro Step
Teiler	**10**	**1**
schnell	3 U/sec = 2,5 (10)Ticks pro Step	wie oben
mittel	1 U/sec = 16,7 (10)Ticks pro Step	wie oben
langsam	1/6 U/sec = 100 (10)Ticks pro Step	wie oben

Als Orientierungshilfe für eigene Gewichtungstabellen ist hier die Standardtabelle abgedruckt, die für einen Drehgeber mit 30 Schritten pro Umdrehung erstellt ist (dieser Drehgeber befindet sich auf dem Graphic Toolkit). Da die interne Tabelle 256 Bytes umfaßt, werden die restlichen Bytes automatisch mit 1 aufgefüllt:

```
WGHT$ = &
64 64 64 64 64 64 32 28 18 10 08 04 04 04 04 03&
03 03 03 03 03 03 02 02 02 02 02 02 02 02 02 02"%
```

Das Beispielprogramm kann auf dem Plug & Play Lab ausgeführt werden. Ein weiteres, beeindruckenderes Beispiel befindet sich im Verzeichnis LCD-Kit unter dem Namen ENC_WEIGHT.TIG. Es erfordert jedoch ein Grafik-LCD als Anzeige.

Programmbeispiel:


```
'-----------------------------------------------------------------------
' Name: ENC1_WGHT.TIG
' Zeigt die die Dynamik-Funktion des Encoder-Treibers
' Bei hoeherer Drehgeschwindigkeit zaehlt der Treiber
' in groesseren Schritten.
' Es werden auch aktuelle Speed-Index Werte angezeigt.
' Fuer Grafik-LCD siehe auch ENC_WEIGHT.TIG im Verzeichnis LCD-Kit
'-----------------------------------------------------------------------
user_var_strict                    ' erzwinge Var-Deklaration
#include DEFINE_A.INC              ' allgemeine Definitionen
#include UFUNC3.INC                ' User-Function-Codes

TASK MAIN
  LONG dwEnc, dwEnc_DYN, SPEED_NDX
  LONG N, SPEED
  REAL SPEED_REAL
  STRING DYN_WEIGHT$ (256)         ' Gewichts-Tabelle fuer DYN-Steps
weigh

' LCD-Treiber installieren (BASIC-Tiger)
  install_device #LCD, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  install_device #LCD, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  install_device #TA, "TIMERA.TDD",3,156 ' Zeitbasis 1 kHz
  install_device #ENC, "ENC1_801.TDD" ' Encoder driver

  PUT #LCD, "<1bh>c<0><0f0h>"      ' Cursor aus
  PRINT #LCD, "<1>Encoder on L80/81"

  DYN_WEIGHT$ = "&                 ' initialisiere Gewichtungstabelle
64 64 64 64 64 64 32 28 18 10 08 04 04 04 04 03&
03 03 03 03 03 03 02 02 02 02 02 02 02 02 02 02"%
  put #ENC, #0, #UFCO_WEIGHTTAB, DYN_WEIGHT$ ' dynamische Gewichtung
  put #ENC, #0, #UFCO_ENC_DIR, 1   ' setze Zaehlrichtung negativ
  put #ENC, 0                      ' starte Encoderzaehler

  while 0 = 0                      ' Endlosschleife
    get #ENC, #0, 4, dwEnc         ' Static-Drehschritte lesen
    get #ENC, #2, 4, dwEnc_DYN     ' DYNAMIC-Drehschritte lesen
    get #ENC, #6, 4, SPEED_NDX     ' lies Geschwindigkeits-Index: 00..FF
    SPEED_REAL = LOG (256000.0/(SPEED_NDX+1))
    SPEED      = INT (110*(SPEED_REAL-3))
    print #LCD, "<1Bh>A<0><1><0F0h>Steps:";dwEnc;"    ";
    print #LCD, "<1Bh>A<0><2><0F0h> dyn.:";dwEnc_DYN;"    ";
    print #LCD, "<1Bh>A<0><3><0F0h>Speed:";SPEED;"    ";
    wait_duration 100              ' Schleifengeschwindigkeit
loop spee
  endwhile
END
```

Leere Seite

Frequenzmesser

Dieser Device-Treiber mißt die Frequenz an einem Pin, indem die Pulslängen (high und low) ausgewertet werden. Durch die besondere Funktionsweise wird insbesondere bei niedrigen Frequenzen hohe Genauigkeit bei kurzer Torzeit erzielt. Bei der Installation des Treibers wird durch den Dateinamen festgelegt, an welchem Pin die Frequenzmessung stattfindet. Die maximal erfassbare Frequenz ist durch die Einstellung des TIMERA festgelegt.

Dateiname: FREQ1_**Pp**.TDD

INSTALL DEVICE #*D*, "FREQ1_Pp.TDD", P1

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Pp im Dateinamen steht für:
P: interner Port
p: Meß-Pin.

P1 Triggerung bei
0: steigender Flanke
1: fallender Flanke
2: nächster Flanke

Hinweis: TIMERA.TDD muß vorher eingebunden werden.

Die Messung wird gestartet, indem mit einer PUT-Instruktion die gewünschte Anzahl Messungen an den Treiber übergeben wird:

PUT #*D*, *Anzahl*

Zunächst wartet der Treiber FREQ1 auf die nächste passende Flanke. Dann werden maximal soviel Messungen durchgeführt, wie durch **Anzahl** vorgegeben ist. TIMERA gibt mit dem eingestellten Zeitraster vor, wann der Meßpin kontrolliert wird. Jeder Flankenwechsel an dem Pin leitet eine neue Messung ein.

Beachte: Wenn die TIMERA Frequenz geändert wird, während noch Messungen laufen, entstehen Werte, die nicht mehr nachvollziehbar sind.

User-Function-Codes für Input (Instruktion GET):

Nr	Symbol Prefix: UFCI_	Beschreibung
1	UFCI_IBU_FILL	Füllstand des Eingangspuffers (Byte)
2	UFCI_IBU_FREE	freier Platz im Eingangspuffer (Byte)
3	UFCI_IBU_VOL	Größe des Eingangspuffers (Byte)
65	UFCI_LAST_ERRC	letzter Error-Code
99	UFCI_DEV_VERS	Version des Treibers

User-Function-Codes für Output (Instruktion PUT):

Nr	Symbol Prefix: UFCO_	Beschreibung
1	UFCO_IBU_ERASE	Eingangspuffer löschen
129	UFCO_FRQ_GATE	Setzt Torzeit als maximale Meßzeit (6...65535)
131	UFCO_FRQ_EDGE	Setzt die Triggerflanke zum Start der Messung: 0: steigende Flanke 1: fallende Flanke 2: nächste Flanke
134	UFCO_FRQ_CAL	Setzt den Kalibrierfaktor
149	UFCO_FRQ_STOP	Stopt die Messung

Mit dem Kommando UFCO_FRQ_GATE wird die maximale Meßzeit in TIMERA-Einheiten gesetzt. Während dieser Zeit wird der Meßpin beobachtet und Flanken registriert. Die wirkliche Meßzeit ist stets kleiner-gleich dieser Torzeit.

Der Frequenz-Device-Treiber stellt das Ergebnis der Messung auf verschiedenen Sekundäradressen in verschiedenen Einheiten zur Verfügung:

Sek.-Adr.	Lesevorgang
0	Ergebnis in mHz (=1/1000Hz)
1	Ergebnis als durchschnittliche Periodenlänge in µsec

Um Quarzungenauigkeiten des Moduls auszugleichen, kann ein Kalibrierungsfaktor eingegeben werden. Der Wert 65536=10000H ist gleichbedeutend mit 100%, d.h. bei diser Angabe als Kalibrierfaktor wird der Meßwert unverändert weitergegeben. Abweichungen nach oben oder unten können entsprechend in Schritten à 1/65536 korrigiert werden.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name: FREQ1.TIG
' zeigt Puffergroesse an, misst dann mit Samplerate von 12.5kHz
' und kann Frequenzen bis theor. 6.25kHz messen
'--------------------------------------------------------------------
#INCLUDE UFUNC3.INC                     ' User-Function-Definitionen
TASK Main                               ' Beginn Task MAIN
  LONG F
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  INSTALL_DEVICE #2, "TIMERA.TDD",1,200' Zeitbasis 12,5kHz
  INSTALL_DEVICE #3, "FREQ1_80.TDD",0  ' Frequenzmesser

  GET #3, #0, #UFCI_IBU_VOL, 0, VOL     ' wie gross ist der Puffer
  PRINT #1, "buffer size:";VOL          ' anzeigen
  WAIT_DURATION 3000                    ' nur zur Information

  PRINT #1, "<1>Frequency on L80"
  PUT #3,#0,#UFCO_FRQ_GATE, 5000        ' Anzahl Samples/Messung
  PUT #3, 0                             ' starte endlose Messung
  LOOP 999999999                        ' sehr viele Loops
    GET #3,#0,#UFCI_IBU_FILL,0,FILLING
    IF FILLING > 0 THEN                 ' Liegen Messwerte vor?
      GET #3, #0, 4, F                  ' Ergebnis in mHz einlesen
      USING "UD<7><1>   3,3,3,3.3"      ' Dezimalpunkt richtig setzen
      PRINT_USING #1, "<1BH>A<0><2><0F0H> freq0:";F;"  ";
      GET #3,#0,#UFCI_IBU_FILL,0,FILLING
      WHILE FILLING = 0                 ' warte auf neuen Messwert
        GET #3,#0,#UFCI_IBU_FILL,0,FILLING
      ENDWHILE
      GET #3, #1, 4, F                  ' Ergebnis in Mikrosek.
      USING "UD<7><1>   3,3,3,3.3"      ' Dezimalpunkt richtig setzen
      PRINT_USING #1, "<1BH>A<0><3><0F0H>period:";F;"msec";
    ENDIF
  ENDLOOP
END                                     ' Ende Task MAIN
```

Pulslängen mit hoher Auflösung messen

Der Device-Treiber 'PLSIN1' mißt die Länge von High-Pulsen mit einer Auflösung bis zu 0,4 µsec und schreibt den Wert in einen Puffer, so daß auch kurz hintereinander folgende Pulse erfaßt werden können. Bei der Installation des Treibers wird der Meßbereich festgelegt. Der Bereich ist jedoch durch Kommandos an den Treiber auch später noch veränderbar.

Dateiname: PLSIN1.TDD

INSTALL DEVICE #*D*, "PLSIN1.TDD", *Bereich*

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Bereich ist ein Parameter zur Festlegung des Bereichs.

Bereich	Zeitbasis	Auflösung	Zeitbereich
1	2.500.000 kHz	0,400 µsec	0,0004...26,214 msec
2	625.000 kHz	1,600 µsec	0,0016...104,856 msec
3	156.250 kHz	6,400 µsec	0,0064...419,424 msec

Sekundär-Adresse 0 wählt den Kanal 0 des Pulse-in-Device-Treibers. In BASIC-Tiger®- oder Tiny-Tiger®-Modulen der Version 1.0xx steht nur dieser Kanal zur Verfügung. Der Eingabepin ist immer der **Pin L84**.

Der Device-Treiber PLSIN1 mißt sehr kurze Pulse-mit der Auflösung von bis herab zu 0,4µsec und beansprucht dafür Hardware-Resourcen des BASIC-Tiger® oder Tiny-Tiger®-Moduls. Da andere schnelle Treiber eventuell ebenfalls diese Hardware-Resourcen benötigen, ist die gleichzeitige Benutzung diverser Treiber ausgeschlossen.

Mögliche Nutzung der Treiber PLSIN1.TDD zusammen mit PLSOUT1.TDD in BASIC-Tiger®- und Tiny-Tiger®-Modulen:

PLSOUT1	PLSIN1
1 Kanal	—
—	1 Kanal

Die eingehende Meßwerte werden in einem Puffer von 256 Byte Größe gepuffert. Die gemessenen Werte sin WORD, so daß 128 Werte mit einer maximalen Pulslänge von 65535 Einheiten abgelegt werden. Eine Einheit ist identisch mit der Auflösung

des Bereichs, z.B. 1.6 µsec im Bereich 2. Die Zustände der Puffer sowie aufgetretene Fehler können durch ‘USER-FUNCTION-CODEs’, kurz ‘UFC’ abgefragt werden.

Beispiel: lies einen Pulslängen-Meßwert aus dem Puffer:


```
GET #11, 2, wVar
```

User-Function-Codes für Input (Instruktion GET):

Nr	Symbol	Beschreibung
1	UFCI_IBU_FILL	Füllstand des Eingangspuffers (Byte)
2	UFCI_IBU_FREE	freier Platz im Eingangspuffer (Byte)
3	UFCI_IBU_VOL	Größe des Eingangspuffers (Byte)
65	UFCI_LAST_ERRC	letzter Error-Code
99	UFCI_DEV_VERS	Version des Treibers
160	UFCI_IPL_OVL	Anzahl der Pufferüberläufe seit dem letzten Lesen dieses Zählers. Setzt den Zähler auf 0 zurück.

User-Function-Codes für Output (Instruktion PUT):

Nr	Symbol Prefix: UFCO_	Beschreibung
1	UFCO_IBU_ERASE	Eingangspuffer löschen
128	UFCO_IPL_RNG	setze Bereich

Beispiel: frage die Versionsnummer des Treibers ab:

```
GET #11,#0, #UFCI_DEV_VERS, 2, wVersion
```

Beispiel: setze Bereich 2:

```
PUT #11,#0, #UFCO_IPL_RNG, 2
```

Beispiel: frage den Füllstand des Pulslängenpuffers ab. Das Kommando 'UFCI_IBU_FILL' ist ein Byte, die gelesene Antwort des Treibers jedoch mindestens eine WORD-Zahl:

```
GET #11,#0, #UFCI_IBU_FILL, 2, wVar
```

Programmbeispiel:

```
'-----------------------------------------------------------------------
' Name: PLSIN1.TIG
'-----------------------------------------------------------------------
#INCLUDE UFUNC3.INC                      ' Definiere User-Function-Codes

TASK MAIN                                ' Beginn Task MAIN
  WORD I, PLEN
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  INSTALL DEVICE #8, "PLSIN1.TDD", 1    ' 1 = Bereich

  PRINT #1, "connect pin L8-0"           ' verbinde Pin L80
  PRINT #1, " with pin L8-4"             ' mit Pin L84
  OUT 8, 00000001b, 0                    ' Pin L8-0 low
  DIR_PIN 8, 0, 0                        ' Pin L8-0 output

  PUT #8,#0, #UFCO_IPL_RNG, 1            ' Bereich 1
  WAIT_DURATION 1000
  OUT 8, 00000001b, 1                    ' 1 Puls auf Pin L80
  OUT 8, 00000001b, 0
  GET #8, 2, PLEN                        ' Liest 0 wenn kein Puls
  PRINT #1, "<1>Pulse length:"
  PRINT #1, PLEN;" *0.4microsec"

  PUT #8,#0, #UFCO_IPL_RNG, 2            ' Bereich 2
  WAIT_DURATION 1000
  OUT 8, 00000001b, 1                    ' 1 Puls auf Pin L80
  OUT 8, 00000001b, 0
  GET #8, 2, PLEN                        ' Liest 0 wenn kein Puls
  PRINT #1, PLEN;" *1.6microsec"

  PUT #8,#0, #UFCO_IPL_RNG, 3            ' Bereich 3
  WAIT_DURATION 1000
  OUT 8, 00000001b, 1                    ' 1 Puls auf Pin L80
  OUT 8, 00000001b, 0
  GET #8, 2, PLEN                        ' Liest 0 wenn kein Puls
  PRINT #1, PLEN;" *6.4microsec"
END                                      ' Ende Task MAIN
```

Leere Seite

Pulslängen mit TIMERA messen

Dieser Device-Treiber mißt die Pulslängen (high und low) an einem Pin. Bei der Installation des Treibers wird durch den Dateinamen festgelegt, an welchem Pin die Pulslängenmessung stattfindet. Die Auflösung ist durch die Einstellung des TIMERA festgelegt.

Dateiname: PLSI2_**Pp**.TDD

INSTALL DEVICE #*D*, "PLSI2_Pp.TDD", P1, P2

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Pp im Dateinamen steht für:
P: interner Port
p: Meß-Pin.

P1 ist ein Parameter, der die Messung auf 16 oder 32 Bit einstellt
32: stellt 32-Bit-Messung ein
jeder andere Wert oder kein Wert: 16-Bit-Messung.

P2 ist ein Parameter, der bei 16-Bit-Messung die automatische Vorzeichenerweiterung von WORD nach LONG vornimmt.
0: Vorzeichen wird erweitert
1: Vorzeichen wird nicht erweitert

Hinweis: TIMERA.TDD muß vorher eingebunden werden.

Die Messung wird gestartet, indem mit einer PUT-Instruktion ein Wert an den Treiber übergeben wird. Der übergebene Wert bestimmt die Aktion des Treibers:

Wert	Lesevorgang
0	Stoppe die Messung
1	Starte die Messung mit der nächsten Flanke
2	Starte die Messung mit der nächsten steigenden Flanke
3	Starte die Messung mit der nächsten fallenden Flanke

Nachdem die Messung gestartet ist, wartet PLS2 im Takt der TIMERA-Ticks auf die passende Flanke. Ist die Flanke gekommen, dann wird in Zeiteinheiten, die TIMERA vorgibt, gemessen. Der 'high'-Teil des Pulses wird als positive Zahl, der 'low'-Teil als negative Zahl im Puffer abgelegt. Die Messung wird gestoppt, wenn der Puffer voll ist, oder wenn ein Stop-Kommando gegeben wurde.

Die 16-Bit-Messung hat einige Vorteile:

- Die CPU wird weniger belastet.
- Es passen mehr Meßwerte in den Puffer.

Da WORD-Variable kein Vorzeichen besitzen, sollten die Meßwerte mit eine LONG-Variablen ausgelesen werden. Der Treiber erweitert automatisch das Vorzeichen passend von WORD nach LONG. Würde man die Meßwerte mit WORD-Variablen auslesen oder die automatische Vorzeichenerweiterung abschalten, dann wären alle Meßwerte für den 'low'-Teil der Pulse 65536 minus der gemessenen Zeit.

Beachte: Wenn die TIMERA Frequenz geändert wird, während noch Messungen laufen, entstehen Werte, die nicht mehr nachvollziehbar sind. !

User-Function-Codes für Input (Instruktion GET):

Nr	Symbol Prefix: UFCI_	Beschreibung
1	UFCI_IBU_FILL	Füllstand des Eingangspuffers (Byte)
2	UFCI_IBU_FREE	freier Platz im Eingangspuffer (Byte)
3	UFCI_IBU_VOL	Größe des Eingangspuffers (Byte)
65	UFCI_LAST_ERRC	letzter Error-Code
99	UFCI_DEV_VERS	Version des Treibers

User-Function-Codes für Output (Instruktion PUT):

Nr	Symbol Prefix: UFCO_	Beschreibung
1	UFCO_IBU_ERASE	Eingangspuffer löschen
128	UFCO_PLS_SIGN	0: 16-Bit-Wert wird mit Vorzeichen ausgewertet (-32767...+32768) 1: 16-Bit-Wert wird ohne Vorzeichen ausgewertet (0...65535)
133	UFCO_PLS_STOP	Stoppt die Messung

Programmbeispiel:

```
'--------------------------------------------------------------------------
' Name: PLSI2.TIG
'--------------------------------------------------------------------------
user_var_strict
#include define_a.inc
#include UFUNC3.INC
TASK Main                                ' Beginn Task MAIN
LONG FILLING, F
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  INSTALL_DEVICE #2, "TIMERA.TDD",1,250 ' Zeitbasis 10kHz
  INSTALL_DEVICE #3, "PLS2_P80.TDD",0   ' Pulslaengenmessung
  INSTALL DEVICE #4, "SER1.TDD", BD_19_200, DP_8N, YES, BD_19_200, DP_8N
  USING "UD<8><1>   3,3,3,3.3"          ' Dezimalpunkt richtig setzen

  PRINT #1, "<1>pulses on L80";
  run_task disp
'         0 = Stoppe Messung sofort
'               1 = Starte Messung mit der naechsten Flanke
'               2 = Starte Messung mit der naechsten Steigenden Flanke
'               3 = Starte Messung mit der naechsten Fallenden Flanke
  PUT #3, 2                              ' starte mit naechster Flanke
  LOOP 999999999                         ' sehr viele Loops
    GET #3,#0,#UFCI_IBU_FILL,0,FILLING  ' Liegen Messwerte vor?
    IF FILLING > 1 THEN                  '
      GET #3, #0, 4, F                   ' Ergebnis einlesen
      F = ABS(F)
      PRINT_USING #1, "<1BH>A<0><2><0F0H>plsl0 L80:";F;
      PRINT_USING #4, "plsl0:";F;"<9>";
      GET #3, #0, 4, F                   ' Ergebnis einlesen
      F = ABS(F)
      PRINT_USING #1, "<1BH>A<0><3><0F0H>plsl1 L80:";F;
      PRINT_USING #4, "plsl1:";F
      WAIT_DURATION 10
    ENDIF
  ENDLOOP                                '
END                                      ' Ende Task MAIN

TASK disp
  BYTE i
  LONG f

  for i = 0 to 0 step 0
    get #3, #0, #UFCI_IBU_FILL, 0, f
    print #1, "<1bh>A<0><0><0f0h>fill: ";f;"  ";
    wait_duration 100
  next
END
```

Leere Seite

Pulse mit hoher Auflösung ausgeben

Der Device-Treiber ‘PLSOUT1’ ist in der Lage, Pulse mit einer Auflösung von bis zu 0,4 µsec zu erzeugen. Bei der Installation des Treibers wird der Bereich festgelegt. Der Bereich ist jedoch durch Kommandos an den Treiber auch später noch veränderbar.

Weitere Informationen zu PLSOUT1.TDD:

- Sekundär-Adressen des PLSOUT1.TDD
- User-Function-Codes des PLSOUT1.TDD

Dateiname: PLSOUT1.TDD

INSTALL DEVICE #*D*, "PLSOUT1.TDD", *Bereich*

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Bereich ist ein Parameter zur Festlegung des Bereichs.

Bereich	Zeitbasis	Auflösung	Zeitbereich
1	2.500.000 kHz	0,400 µsec	0,0004...26,214 msec
2	625.000 kHz	1,600 µsec	0,0016...104,856 msec
3	156.250 kHz	6,400 µsec	0,0064...419,424 msec

Sekundär-Adresse 0 wählt den Kanal 0 des Pulse-out-Device-Treibers. Die mögliche Anzahl der Kanäle ist Modulabhängig. In BASIC-Tiger®- oder Tiny-Tiger®-Modulen der Version 1.x steht ein Kanal zur Verfügung. Der Ausgabepin des Kanals 0 ist immer der **Pin L86**. Ab BASIC-Tiger® Modul Version 2.x wählt die Sekundär-Adresse 1 den zweiten Pulse-Ausgang auf dem Pin L82 aus.

Der Device-Treiber PLSOUT1 ermöglicht sehr schnelle Pulsausgabe bis 1,25MHz und beansprucht dafür Hardware-Resourcen des BASIC-Tiger® oder Tiny-Tiger®-Moduls. Da andere schnelle Treiber eventuell ebenfalls diese Hardware-Resourcen benötigen, ist teilweise die gleichzeitige Benutzung dieser Treiber ausgeschlossen.

Mögliche Nutzung der Treiber PLSOUT1.TDD zusammen mit PLSIN1.TDD in BASIC-Tiger®- und Tiny-Tiger®-Modulen:

PLSOUT1	PLSIN1
1 Kanal	—
—	1 Kanal

Pulsausgabe:

PUT #*D*, #0, *cnt*, *duty*, *cycle*

D	ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.
cnt	ist eine Variable oder ein Ausdruck vom Datentyp **WORD** und gibt die Anzahl der auszugebenden Pulse an.
duty	ist eine Variable oder ein Ausdruck vom Datentyp **WORD** im Bereich von 0...65535 und gibt die Zeit in Einheiten des eingestellten Bereichs an, die der Puls 'low' sein soll.
cycle	ist eine Variable oder ein Ausdruck vom Datentyp **WORD** im Bereich von 0..65535 und gibt die Gesamtdauer eines Pulses in Einheiten des eingestellten Bereichs an.

Hinweis: Der Treiber PLSO1 verwendet nun WORD-Variable für die Anzahl der Pulse. Bei bestehenden Projekten muß dies eventuell berücksichtigt werden.

Achtung:
Die Variablentypen für die Anzahl der Pulse, Cycle und Duty muß eingehalten werden. Andere Datentypen sind nicht erlaubt.

Wenn Pulse gezählt werden, dann gelten für die derzeitigen Tiger-Module folgende Einschränkungen für CYCLE und DUTY:
CYCLE, Range-1 min. 32, Range-2 min. 8, Range-3 min. 3
DUTY, Range-1 min. 31, Range-2 min. 7, Range-3 min. 2
Zu kleine Angaben führen bei COUNT <> 0 zu einem Runtime-Error.

User-Function-Codes des PLSOUT1.TDD

In der Include-Datei 'UFUNCn.INC' sind User-Function-Codes des Device-Treibers PLSOUT1 für Input definiert (Instruktion GET):

Nr	Symbol Prefix: UFCI_	Beschreibung
176	UFCI_OPL_STAT	lies aktuellen Count-Down-Wert 0: letzter Puls läuft gerade aus n: es folgen noch n komplette Pulse -1: ist bereits zum Stillstand gekommen

User-Function-Codes für Output (Instruktion PUT):

Nr	Symbol	Beschreibung
144	UFCO_OPL_RNG	setze Bereich
145	UFCO_OPL_CNT	setze neue Anzahl Pulse

Einstellen des Bereichs 2 während der Laufzeit:

```
PUT #10,#0, #UFCO_OPL_RNG, 2
```

Beispiel: lies die Anzahl der Pulse, die nach dem gerade laufenden noch folgen werden:

```
GET #10,#0, #UFCI_OPL_STAT, 4, CNT
```

Beispiel: bei laufender, vielleicht unendlicher Pulsausgabe, setze neue Anzahl der Pulse auf 1. Damit läßt sich die Ausgabe abbrechen, wobei das Tastverhältnis und die Frequenz erhalten bleiben, der letzte Puls wird noch vollständig ausgegeben:

```
PUT #10,#0, #UFCO_OPL_CNT, 1
```

Als Beispiel-Applikation zu dem Device-Treiber PLSOUT1.TDD finden Sie im Unterverzeichnis APPLICAT ein Programm welches über den Output-Pin eine Melodie ausgibt: PLSO1_JUKEB.TIG.

Programmbeispiel:

```
'-----------------------------------------------------------------------
' Name: PLSOUT1.TIG
'-----------------------------------------------------------------------
#INCLUDE UFUNC3.INC

TASK MAIN
  WORD  I, DUTY
  LONG CYCLE
  LONG N, COUNT
  INSTALL_DEVICE #9, "PLSOUT1.TDD", 3   ' Zeitbasis-Bereich = 3

  COUNT = 1                             ' 1 Puls der Länge
  CYCLE = 20                            ' 20x 6,4usec = 128usec
  DUTY = 10                             ' Duty 50%
  PUT #9, COUNT, DUTY, CYCLE            ' Pulsausgabe auf Pin 86

  COUNT = 1                             ' 1 langer Puls, dessen Ende
                                        ' abgewartet wird, duty 50%
  CYCLE = 65534                         ' 0,42sec
  DUTY = 32767                          ' Duty 50%
  PUT #9, COUNT, DUTY, CYCLE            ' Pulsausgabe auf Pin 86
                                        ' Count-Down-Loop wird beendet
  FOR N=999999999 TO 0 STEP -1          ' wenn N=0 gelesen wird
    GET #9,#0,#UFCI_OPL_STAT, 4, N      ' lies Treiber Count-Down
  NEXT

  COUNT = 200                           ' 200 Pulse der Länge
  CYCLE = 360                           ' 360x 6,4usec = 2,3msec
  DUTY = 18                             ' Duty 5%
  PUT #9, COUNT, DUTY, CYCLE            ' Pulsausgabe auf Pin 86
                                        ' Count-Down-Loop wird beendet
                                        ' Count-Down-Loop wird beendet
  FOR N=999999999 TO 0 STEP -1          ' wenn N=0 gelesen wird
    GET #9,#0,#UFCI_OPL_STAT, 4, N      ' lies Treiber Count-Down
  NEXT
END
```

Pulse mit TIMERA ausgeben

Dieser Device-Treiber gibt Pulse an einem Pin aus, deren Gesamtdauer (Cycle) und 'high'-Anteil (Duty) einstellbar ist. Bei der Installation des Treibers wird durch den Dateinamen festgelegt, an welchem Pin die Pulse ausgegeben werden. Die Auflösung ist durch die Einstellung des TIMERA festgelegt.

Weitere Informationen zu PLSO2_xx.TDD:

- Sekundär-Adressen des PLSO2_xx.TDD
- User-Function-Codes des PLSO2_xx.TDD

Dateiname: PLSO2_**Pp**.TDD

INSTALL DEVICE #*D*, "PLSO2_Pp.TDD", P1, P2

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Pp im Dateinamen steht für:
P: interner Port
p: Meß-Pin.

P1 ist ein Parameter, der den inaktiven Pegel einstellt.

Hinweis: TIMERA.TDD muß vorher eingebunden werden.

Die Pulsausgabe wird gestartet, indem mit einer PUT-Instruktion die Werte 'Anzahl der Pulse', Cycle und Duty an den Treiber übergeben werden.

PUT #*D*, *cnt*, *duty*, *cycle*

cnt ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...2147438647 und gibt die Anzahl der auszugebenden Pulse an.
cnt=0 bedeutet: unendlich.

duty ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...65535 und gibt die Zeit in Einheiten des TIMERA an, die der Puls ‘high’ sein soll. 'duty' ist immer kleiner als 'cycle'.

cycle ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0..65535 und gibt die Gesamtdauer eines Pulses in Einheiten des TIMERA an.

Sekundär-Adressen des PLSO2_xx.TDD

Auf die Sekundär-Adresse 1 geschriebene Werte sind 'Reload-Werte', die direkt übernommen werden, wenn die laufenden Pulse ausgegeben sind. Beispiel:

```
PUT #D, cnt, duty, cycle          ' starte Pulsausgabe
PUT #D, #1, cnt1, duty1, cycle1   ' hinterlege Werte f. folgende Ausgabe
```

Auf verschiedenen Sekundär-Adressen des Treibers können folgende Informationen gelesen werden:

Sekundär-Adr.	Information
0	wieviele Pulse sind noch auszugeben (ohne Reload)
1	wieviele Pulse sind laut Reload noch auszugeben
2	wieviele Pulse sind noch auszugeben (mit Reload)

User-Function-Codes des PLSO2_xx.TDD

User-Function-Codes für Input (Instruktion GET):

Nr	Symbol Prefix: UFCI_	Beschreibung
65	UFCI_LAST_ERRC	letzter Error-Code
99	UFCI_DEV_VERS	Version des Treibers
147	UFCI_PO2_REST	Liefert die Anzahl der Pulse, die noch auszugeben sind. Für 'unendlich wird 2147483647 (7FFFFFFF) zurückgegeben.

Beispiel: lese Anzahl der noch auszugebenden Pulse:

```
GET #10,#0, #UFCI_OPL_REST, 0, varREST
```

User-Function-Codes für Output (Instruktion PUT):

Nr	Symbol Prefix: UFCO_	Beschreibung
128	UFCO_PO2_LEVEL	Setzt den Ruhepegel am Pulsausgabepin
133	UFCO_PO2_STOP	Stoppt die Pulsausgabe 0: sofort (Pegel unbestimmt) 1: beim nächsten 'high'-Pegel 2: beim nächsten 'low'-Pegel
134	UFCO_PO2_ADJ	Setzt während der Pulsausgabe eine neue Anzahl und löscht dabei den Reload-Puffer. Damit ist ein Soft-Stop möglich.

Beispiel: setze neue Werte für Anzahl, Duty, Cycle bei laufender Pulsausgabe:

```
PUT #10,#0, #UFCO_OPL_ADJ, cnt, duty, cycle
```

Programmbeispiel:

```
'-----------------------------------------------------------------------
' Name: PLSO2.TIG
'-----------------------------------------------------------------------
#INCLUDE UFUNC3.INC

TASK MAIN
  LONG REST, REST0, REST1, REST2

' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  INSTALL_DEVICE #13, "TIMERA.TDD", 1,250 ' 10kHz
  INSTALL_DEVICE #20, "PLSO2_80.TDD"    ' Pulse auf Pin L80

  PUT #20, 400, 12, 24                  ' 400 Pulse 2,4microsec cycle
                                        ' in den Reload-Puffer
  PUT #20, #1, 800, 6, 12               ' 800 Pulse 1,2microsec cycle

  GET #20, #2, 4, REST2                 ' Pulse insgesamt noch
  WHILE REST2 > 0
    GET #20, #0, #UFCI_OPL_REST, 4, REST' Pulse noch auszugeben
    GET #20, #0, 4, REST0               ' werden gerade ausgegeben
    GET #20, #1, 4, REST1               ' sind im Reloadpuffer
    GET #20, #2, 4, REST2               ' insgesamt, wie UFCI_OPL_REST
    PRINT #1, "<1BH>A<0><0><0F0H>";REST;" total pulses";"    ";
    PRINT #1, "<1BH>A<0><1><0F0H>";REST0;" pulses";"    ";
    PRINT #1, "<1BH>A<0><2><0F0H>";REST1;" reload pulses";"    ";
    PRINT #1, "<1BH>A<0><3><0F0H>";REST2;" total pulses";"    ";
  ENDWHILE
                                        ' READY
END
```

Siehe auch: Applikation PLSO2_STEPPER.TIG

PWM1 (Pulsweitenmodulation)

Dateiname: PWM1.TDD

INSTALL DEVICE #*D*, "PWM1.TDD" [, *f0, r0, f1, r1*]

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

f0, f1 gibt die Frequenz für Kanal 0 an:
0: hohe Frequenz
1: mittlere Frequenz
2: niedrige Frequenz

r0, r1 gibt die Auflösung für Kanal 0 an:
0: 6-Bit
1: 7-Bit
2: 8-Bit

Dieser Device-Treiber ermöglicht pulsweitenmodulierte Analogausgabe auf 2 Pins des BASIC-Tiger®-Moduls mit der Instruktion PUT.

Funktion	Pin	Modul-A Pin-Nr.	Tiny-Tiger Pin-Nr.
PWM-Kanal 0	L72	12	11
PWM-Kanal 1	L73	13	12

Mit der Sekundär-Adresse in der Ausgabe-Instruktion wird die Kanalnummer angegeben. Bei 8-Bit Auflösung werden Werte von 0...255 an den jeweiligen PWM-Ausgang gegeben, bei den anderen Auflösungen entsprechend 0...127 oder 0...63. Der einmal gesetzte Wert bleibt am PWM-Ausgang so lange erhalten bis ein neuer Wert gesetzt wird.

In der Datei UFUNCn.INC sind Symbole für die Angaben der Frequenz und Auflösung definiert. Die Symbole und den Zusammenhang zwischen Frequenz und Auflösung stellt folgende Tabelle dar (ca.-Werte):

f0,f1	Symbol	Frequenz	6-Bit	7-Bit	8-Bit
0	PWM_SPEED_HI	Hoch	80kHz	40kHz	20kHz
1	PWM_SPEED_MED	Mittel	20kHz	10kHz	5kHz
2	PWM_SPEED_LO	Niedrig	5kHz	2.5kHz	1.2kHz

Weitere Einstellungen am PWM1-Device-Treiber können zur Laufzeit mit Hilfe der User-Function-Codes vorgenommen werden. Die Symbole sind in der Datei UFUNCn.INC definiert. Jeder User-Function-Code stoppt die PWM-Ausgabe. Nach dem Verändern der Auflösung, Frequenz etc. muß die Ausgabe neu gestartet werden.

User-Function Codes zum setzen von Parametern mit der Instruktion PUT:

Nr	Symbol	Beschreibung
144	UFCO_PWM_SPEED	Geschwindigkeit einstellen
		Argument des Kommandos:
0	PWM_SPEED_HI	hohe Geschwindigkeit
1	PWM_SPEED_MED	mittlere Geschwindigkeit
2	PWM_SPEED_LO	niedrige Geschwindigkeit
145	UFCO_PWM_RESO	Auflösung einstellen
		Argument des Kommandos:
0	PWM_RESO_6BIT	6-Bit-Auflösung
1	PWM_RESO_7BIT	7-Bit-Auflösung
2	PWM_RESO_8BIT	8-Bit-Auflösung

Die Frequenz des PWM1-Device-Treiber kann zur Laufzeit mit Hilfe von User-Function-Codes abgefragt werden:

Nr	Symbol	Beschreibung
144	UFCI_PWM_FREQU	Frequenz abfragen:
		Antwort des Treibers:
0	PWM_FREQU_HI	siehe SPEED-Tabelle
1	PWM_FREQU_MED	siehe SPEED-Tabelle
2	PWM_FREQU_LO	siehe SPEED-Tabelle

Programmbeispiel:

```
'-------------------------------------------------------------------------
' Name: PWM1.TIG
'-------------------------------------------------------------------------
TASK MAIN                                ' Beginn Task MAIN
  REAL PI, W                             ' REAL-Variablen deklarieren
  LONG I, P                              ' LONG-Variablen deklarieren
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  INSTALL_DEVICE #6, "PWM1.TDD"          ' PWM-Treiber installieren

  PI = 3.14159265359                     ' PI vorbesetzen
  PRINT #1, "PWM-Test an Pin L72"        ' Auf LCD ausgeben
  LOOP 9999999                           ' sehr viele Loops
    FOR I = 0 TO 360 STEP 5              ' 0 bis 360 Grad
      W = I * PI / 180                   ' in Bogenmass umrechnen
      P = 100 + 90 * SIN (W)             ' Wandlung in LONG
      PUT #6, #0, P                      ' PWM-Ausgabe auf Kanal 0
    NEXT                                 ' naechster Wert
  ENDLOOP                                '
END                                      ' Ende Task MAIN
```

Leere Seite

PWM2 (Pulsweitenmodulation)

Dieser Device-Treiber stellt zeit-synchrone Pulsweitenmoduliere Ausgabe zur Verfügung. Damit kann unter anderem Sprache und Musik ausgegeben werden.

Weiter Informatione zum PWM2.TDD

- PWM-Ausgabe
- User Function-Codes des PWM2.TDD
- Ausgabe-Rate einstellen
- Zwischenwerte ausgeben
- Reload-Puffer verwenden

Dateiname: PWM2.TDD

INSTALL DEVICE #*D*, "PWM2.TDD" [, *Ch*, *Reso*, *Frq*, *Ovs*, *decof*, *sh*, *PS*]

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Ch ist ein Parameter zur Festlegung des Kanals 0 oder 1.

Reso ist ein Parameter zur Festlegung der Auflösung:
0: 6-Bit
1: 7-Bit
2: 8-Bit

Frq ist ein Parameter zur Festlegung der PWM-Trägerfrequenz:
0: hoch
1: mittel
2: niedrig

Ovs ist ein Parameter zur Festlegung des Oversampling:
1: kein Oversampling
n=2...255: es werden n-1 Zwischenwerte erzeugt.

decof ist ein Parameter zur Festlegung des Decoder-Flags:
0: keine Decodierung
Zur Zeit kann hier nur 0 stehen.

sh ist ein Parameter zur Festlegung der Downshifts:
0: kein Downshift
1: ein Downshift (7-bit)
2: zwei Downshifts (6-bit).

PS ist ein Parameter zur Einstellung des Pre-Scalers:
0,1: kein Pre-Scaler
2...65535: Teilerfaktor des Pre-Scalers.

Der Device-Treiber PWM2.TDD gibt Werte aus einem String als PWM-Signal aus (PWM=Puls-Weiten-Modulation). Die Ausgabe wird mit Hilfe des Zeitbasis-Treibers 'TIMERA.TDD' synchronisiert, so daß die Ausführung unabhängig vom BASIC automatisch und bis zu hohen Geschwindigkeiten ausgeführt wird. Der Zeitbasis-Treiber stellt eine Basisfrequenz zur Verfügung, die durch den Pre-Scaler des Treibers PWM2 zur eigentlichen Ausgaberate heruntergeteilt wird. Bei der Installation können Einstellungen des Treibers vorgenommen werden. Später sind diese Einstellung mit Hilfe von User-Function-Codes vorzunehmen.

Hinweis: TIMERA.TDD muß vorher eingebunden werden.

Die Ausgabedaten sind als Bytes in einem String abgelegt.

Der Ausgabe-String muß immer existieren! Nicht erlaubt sind also Variable, die nur vorübergehend leben, wie lokale Strings (in Unterprogrammen) oder temporäre Strings (Expressions). **Richtig: globale oder Task-lokale Strings**.

Der String wird mit der Instruktion PUT mit der Sekundär-Adresse 0 auf dem Datenkanal die PWM-Treibers ausgegeben. Die Bytes des Strings erscheinen in der vorgegegebenen Ausgabegeschwindigkeit als puls-modulierte Werte am Ausgabepin. **Beachte:** Der Ausgabepin wird durch einen Parameter bei der Installation und nicht durch die Sekundär-Adresse bestimmt.

Auf die Sekundär-Adresse 1 werden Nachfolgedaten ("Reload-Data") geschrieben, die automatisch in den Ausgabekanal übergeben werden, sobald dort keine weiteren Daten mehr zur Ausgabe anstehen. Dadurch wird ein lückenloser Übergang von einem Datensatz zum nächsten ermöglicht. Die Übernahme aus dem Reload-Puffer in den Ausgabepuffer findet in dem Moment statt, wenn der Ausgabepuffer leer geworden ist. Wird der Reload-Puffer zu spät geladen, dann wird dieser erst am Ende der nächsten Ausgabe übernommen.

Über den User-Function-Code UFCI_PWM_RELOAD kann das BASIC-Programm abfragen, ob das Nachladen bereits stattgefunden hat und der Nachladepuffer wieder frei ist. Wenn das der Fall ist, kann bereits der nächste Datensatz auf die Sekundäradresse geschrieben werden.

Neben der Auflösung und dem Frequenzbereich der PWM-Trägerfrequenz ist ein 'Oversampling' einstellbar. Durch diese Option werden zwischen zwei Ausgabewerten weitere interpolierte Zwischenwerte ausgegeben, die den Signalverlauf glätten. Die Größe des Ausgabestrings kann dadurch ebenfalls klein gehalten werden.

Bei einer PWM-Auflösung von 7 oder 6 Bit kann der Treiber automatisch die Ausgabewerte durch schieben der Bits auf die Auflösung passend scalieren.

PWM-Ausgabe

Eine Ausgabe wird durch folgende Instruktion gestartet:

PUT #D [, #*Sek_Adr*] *Str$* [, *Offs*, *Anz*, *Wdh*, *Sh_dn*]

D	ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.
#Sek_Adr	ist Variable, Konstante oder Ausdruck vom Typ BYTE, WORD oder LONG und legt den Kanal innerhalb dieses Device-Treibers fest, mit dem kommuniziert werden soll. Ohne diese optionale Angabe wird die Sekundär-Adresse = 0 genommen. Auf der Sekundär-Adresse 1 wird der Nachladepuffer angesprochen.
Str$	ist der String, der die Ausgabedaten enthält. **Der String muß statisch, d.h. global oder task-lokal sein.** Anstelle des Strings kann auch ein DATALABEL als Adresse im Flash angegeben werden.
Offs	ist eine Variable, eine Konstante oder ein Ausdruck vom Typ BYTE, WORD oder LONG und gibt den Offset an, ab dem die Ausgabedaten bei einer Position ungleich 0 ausgegeben werden sollen. Default = 0 (String-Anfang).
Anz	ist eine Variable, eine Konstante oder ein Ausdruck vom Typ BYTE, WORD oder LONG und gibt die Anzahl der Bytes an, die ausgegeben werden sollen. Wenn Anzahl mit 0 angegeben wird, dann wird bis zum Ende des Strings ausgegeben bzw. entsprechend der Längenangabe im Flash, jedoch maximal bis zum Ende des Flashspeichers.
Wdh	ist eine Variable, eine Konstante oder ein Ausdruck vom Typ BYTE, WORD oder LONG und gibt die Anzahl der Wiederholungen an. Bei Angabe von 0 wird die Ausgabe endlos wiederholt.
Sh_dn	ist eine Variable, eine Konstante oder ein Ausdruck vom Typ BYTE, WORD oder LONG und gibt die Anzahl der Shifts nach rechts an. Damit kann die effektive Datengröße an die Auflösung (6,7,8-Bit) des PWM-Kanals angepaßt werden. Mögliche Werte sind 0, 1 oder 2.

Die pulsweitenmodulierte Analogausgabe findet auf folgendem Pin statt:

Funktion	Pin	Modul-A Pin-Nr.	Tiny-Tiger Pin-Nr.
PWM-Kanal 0	L72	12	11
PWM-Kanal 1	L73	13	12

In der Datei UFUNCn.TIG sind Symbole für die Angaben der Frequenz und Auflösung definiert. Die Symbole und den Zusammenhang zwischen Frequenz und Auflösung stellt folgende Tabelle dar (ca.-Werte):

Nr	Symbol	Frequenz	6-Bit	7-Bit	8-Bit
0	PWM_SPEED_HI	hoch	80kHz	40kHz	20kHz
1	PWM_SPEED_MED	mittel	20kHz	10kHz	5kHz
2	PWM_SPEED_LO	niedrig	5kHz	2.5kHz	1.2kHz

User Function-Codes des PWM2.TDD

Nach der Installation können weitere Einstellungen am PWM2-Device-Treiber zur Laufzeit mit Hilfe weiterer User-Function-Codes vorgenommen werden.

Die folgende Tabelle zeigt eine Übersicht über die besonderen Function-Codes dieses Treibers. Die Datei UFUNCn.INC muß eingebunden sein, damit dem Compiler die Kommando-Symbole bekannt sind.

Das Setzen neuer Parameter hat keine Auswirkung auf eine laufende PWM-Ausgabe. Die neuen Parameter werden erst beim nächsten Start wirksam. User-Function-Codes des PWM2.TDD zum Einstellen von Parametern (PUT):

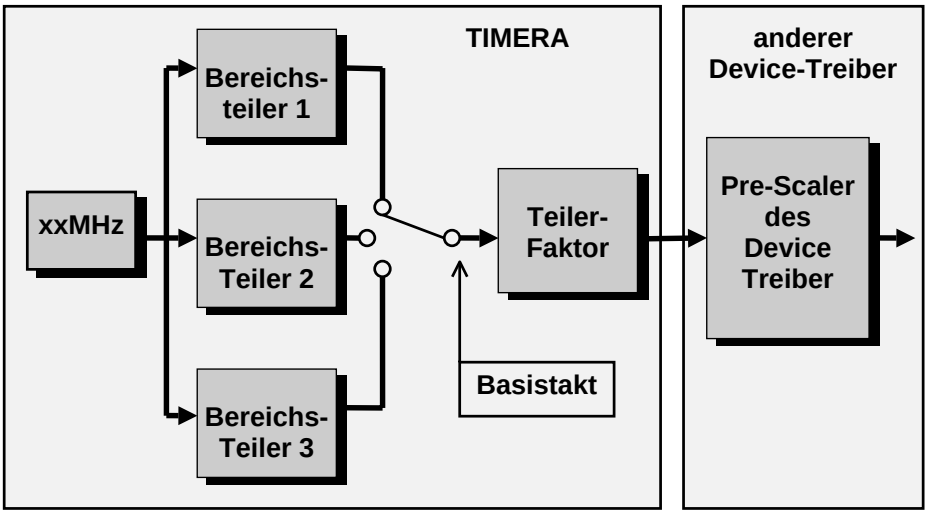
Nr	Symbol Prefix: UFCO_	Beschreibung
144	UFCO_PWM_FREQU	Setze PWM-Trägerfrequenz (siehe Tabelle) 0: hoch 1: mittel 2: niedrig
145	UFCO_PWM_RESO_	Setze PWM-Auflösung: 6: 6-Bit 7: 7-Bit 8: 8-Bit
146	UFCO_PWM_PSCAL	Setze Pre-Scaler: 0, 1: ohne Pre-Scaler n: Teiler (WORD)
147	UFCO_PWM_OSAMP	Oversampling: Anzahl der erzeugten Zwischenwerte: 1: keine Zwischenwerte 2...255: Intervall in so viele Teile aufteilen und Zwischenwerte erzeugen
154	UFCO_PWM_STOP	Stop: halte Ausgabe an.
46	UFCO_PWM_DFLT	Setze PWM-Standardeinstellung

User-Function-Codes des PWM2.TDD zur Abfrage (GET):

Nr	Symbol Prefix: UFCI_	Beschreibung
68	UFCI_CPU_LOAD	lese die durch diesen Treiber beanspruchte CPU-Leistung (100%=10.000)
144	UFCI_PWM_FREQU	lese tatsächliche PWM-Trägerfrequenz in Hz. Die Werte sind ungefähre Angaben.
145	UFCI_PWM_ACT	Aktivität der Ausgabe 0: Ausgabe findet statt. 0FFh: keine Ausgabe
146	UFCI_PWM_RELOAD	Zustand des Nachladepuffers: 0: leer sonst: enthält Daten.

Ausgabe-Rate einstellen

Die Ausgabe-Rate wird von der Basisfrequenz, die der Device-Treiber TIMERA zur Verfügung stellt, abgeleitet. Der Pre-Scaler des Device-Treibers PWM2 teilt dazu die Basisfrequenz weiter herunter:



Die effektive Ausgaberate berechnet sich:

Bereichsfrequenz / Teilerfaktor / Pre-Scaler

So wird zum Beispiel eine Ausgaberate von 500 Werten pro Sekunde eingestellt:

```
PUT #13, 2, 125              ' TIMERA Bereich 2, Teilerfaktor 125 = 5kHz
PUT #8, #0, #UFCO_PW2_PSCAL, 10 ' Pre-Scaler teilt durch 10
```

Nähere Informationen zum Einstellen des Device-Treibers TIMERA finden Sie ab Seite 349. Der Pre-Scaler wird mit dem User-Function-Code UFCO_PWM2_PSCAL eingestellt.

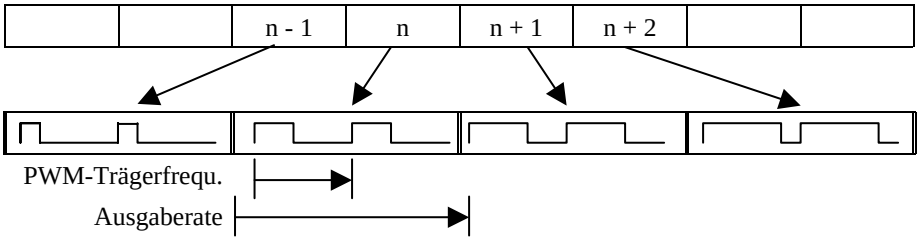
Dieser Device-Treiber kann zusammen mit dem Treiber TIMERA bei ‘schneller’ Einstellung soviel CPU-Leistung beanspruchen, daß andere Tasks stark behindert werden. Mit dem User-Function-Code ‘UFCI_CPU_LOAD kann die Belastung der CPU, wie sie allein durch diesen Treiber verursacht wird, abgefragt werden.
Der Treiber kann bestimmte Einstellungen, die zur Überlastung des Systems führen würden, nicht annehmen.
Beachte: TIMERA muß **vor** PWM2 installiert werden.

!

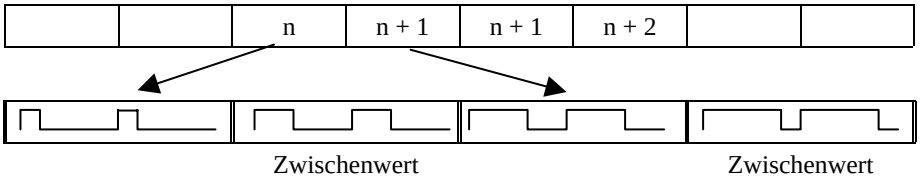
Nicht zu verwechseln mit der Ausgaberate ist die Trägerfrequenz des PWM-Signals. Während die Ausgaberate sehr flexible mit dem System ‚TIMERA und Pre-Scaler' abgestimmt werden kann, wird die Trägerfrequenz in drei möglichen Stufen festgelegt und ist zudem von der Auflösung abhängig.

Zwischenwerte ausgeben

Bei dem Device-Treiber PWM2.TDD bestimmt die Ausgaberate, in welcher Geschwindigkeit neue Ausgabewerte aus dem String geholt werden. Durch Interpolation können Zwischenwerte erzeugt werden, die mit der Ausgaberate ausgegeben werden, so daß die Daten aus dem String langsamer entnommen werden.



Ausgabe mit einem Zwischenwert:



Es folgt ein Beispiel, in dem der Ausgabestring nur zwei Zeichen enthält: 10h und 0C0h. Der Maximalwert von 0FFh kann zwar ausgegeben werden, wird aber auf dem Plug & Play Lab bei ca. 4V abgeschnitten, da der PWM-Filterverstärker nur bis 4V aussteuern kann. Beachten Sie auch, daß Frequenzen oberhalb 400Hz vom Filterverstärker nicht mehr verstärkt werden. Das Beispielprogramm erzeugt bei einer Ausgaberate von 612 Bytes pro Sekunde und zwei Ausgabebytes eine Ausgabeschwingung von 306Hz.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name: PWM2_1.TIG
'--------------------------------------------------------------------
USER VAR STRICT                          ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC                      ' User Function Codes
#INCLUDE DEFINE_A.INC                    ' allg. Symbol-Definitionen

TASK MAIN                                ' Beginn Task MAIN
  BYTE flag                              ' BYTE-Variable deklarieren
  WORD i, slen, rate                     ' WORD-Variablen deklarieren
  LONG rest                              ' LONG-Variable deklarieren
  STRING PWM$                            ' STRING-Variable deklarieren
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #LCD, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
' INSTALL DEVICE #LCD, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  INSTALL_DEVICE #TA, "TIMERA.TDD", 3, 255 ' 156250/255=612Hz
                                 ' ch reso freq ovs inib sh ps
  INSTALL_DEVICE #PWM2, "PWM2.TDD", 0,   8,   0,  0,   0, 1, 1

  PWM$ = "<10><0C0h>"                    ' setze PWM$
  slen = LEN (PWM$)                      ' bestimme Laenge von PWM$
  PRINT #LCD, "PWM2";                    ' Ausgabe auf LCD
'               src$, offs,  len, repeat, shift
  PUT #PWM2, #0, PWM$,    0, slen,      0,     0 ' starte PMW2 Ausgabe
  FOR i = 0 TO 0 STEP 0                  ' Endlosschleife
    GET #PWM2, #0, #UFCI_PW2_REST, 4, rest ' hole verbleibende Pulse
    PRINT #LCD, "<1Bh>A<0><1><0F0h>Rest";rest;"    ";
    WAIT_DURATION 100                    ' warte 100 ms
  NEXT                                   ' Ende Endlosschleife
END                                      ' Ende Task MAIN
```

Reload-Puffer verwenden

Es folgt ein Beispiel, in dem eine Sinusperiode mit 18 Stützpunkten in einem String berechnet wird. Dieser Sinus wird kontinuierlich durch ständiges Laden des Nachladepuffers ausgegeben. Bei einer Ausgaberate von 122Hz und einer Stringlänge von 18 Zeichen entsteht eine Sinus-Ausgabefrequenz von 6,7Hz.

Programmbeispiel:

```
'-----------------------------------------------------------------------
' Name: PWM2_2.TIG
'-----------------------------------------------------------------------
USER VAR STRICT                          ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC                      ' User Function Codes
#INCLUDE DEFINE_A.INC                    ' allg. Symbol-Definitionen

TASK MAIN                                ' Beginn Task MAIN
  BYTE C, flag                           ' BYTE-Variablen deklarieren
  WORD i, slen                           ' WORD-Variablen deklarieren
  LONG rest                              ' LONG-Variable deklarieren
  REAL PID180, S, W                      ' REAL-Variablen deklarieren
  STRING PWM$                            ' STRING-Variable deklarieren
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #LCD, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
' INSTALL DEVICE #LCD, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  INSTALL_DEVICE #TA, "TIMERA.TDD", 3, 128 ' 156250/128=1220Hz
                                 ' ch reso freq ovs inib sh ps
  INSTALL_DEVICE #PWM2, "PWM2.TDD", 0,   8,   0,  0,   0, 1, 10
  PID180 = MC_PI / 180                   ' Konstante errechnen
  PWM$ = ""                              ' PWM$ ist leer
  FOR i = 0 TO 359 STEP 20               ' Schleife fuer SIN-Argumente
    W = LTR ( i )                        ' wandle i in REAL um
    S = (1+SIN(W*PID180))*100            ' S ist SIN zwischen 0 und 2
    C = RTL ( S )                        ' wandle S in BYTE um
    PWM$ = PWM$ + CHR$(C)                ' fuege BYTE-Wert in PWM$ ein
  NEXT                                   ' naechster Wert fuer i
  slen = len ( PWM$ )                    ' bestimme Laenge von PWM$
  PRINT #LCD, "PWM2";                    ' Ausgabe auf LCD
'                src$, offs,  len, repeat, shift
  PUT #PWM2, #0, PWM$,    0, slen,      1,     0 ' Starte PWM2 Ausgabe
  FOR i = 0 TO 0 STEP 0                  ' Endlosschleife
    GET #PWM2, #0, #UFCI_PW2_REST, 4, rest ' hole verbleibende Pulse
    PRINT #LCD, "<1Bh>A<0><1><0F0h>Rest";rest;"    ";
    WAIT_DURATION 100                    ' warte 100 ms
    GET #PWM2, #0, #UFCI_PW2_RLD, 1, flag  ' Reload-Buffer leer ?
    IF flag = 0 THEN                     ' Falls Reload-Buffer leer
'                   src$, offs, len, repeat, shift
      PUT #PWM2, #1, PWM$,    0,slen,      1,     0 ' neue Ausgabe
    ENDIF                                '
  NEXT                                   ' Ende Endlosschleife
END                                      ' Ende Task MAIN
```

Soundausgabe mit PWM2

Mit PWM2 können beliebige Kurvenformen mit hoher Geschwindigkeit ausgegeben werden. Die Trägerfrequenz muß dazu mindestens doppelt so hoch sein wie die höchste auszugebende Kurvenfrequenz. Bei Sprache und Musik muß bei 8 Bit Auflösung die Trägerfrequenz von 20kHz gewählt werden. Ebenso muß die Ausgaberate doppelt so hoch sein wie die höchste wiederzugebende Frequenz. Die Ausgabefrequenz entspricht der Sample-Frequenz bei der Aufnahme. Ein externer

Filter unterdrückt bei der Wiedergabe die Trägerfrequenz und glättet die Sprünge, die durch das Sampling entstehen.

Da ein Sample ein Byte belegt, entstehen bei Mono-Aufnahme bei z.B. 8kHz Samplingfrequenz 8kByte an Daten. So ist nach 4 Sekunden die Aufnahmekapazität eines Strings erschöpft. Um mit BASIC-Tiger®-Modulen gute Sound-Ausgabe zu erzeugen, empfiehlt es sich, den Sound mit einem Sound-Bearbeitungsprogramm auf einem PC in die richtige Form zu bringen und dann in die Anwendung in den Flash-Speicher mit einzubinden. Für gute Qualität haben sich 8kHz-Mono bewährt. Um mehr Sound im gleichen Speicherraum unterzubringen, kann die Samplerate etwas heruntergesetzt werden. In jedem Fall sollte mit der hohen PC-Qualität aufgenommen (22kSamples, Stereo) und dann per Software auf 8kHz-Mono umgerechnet werden. Tiger-BASIC® benötigt schließlich die Rohdaten, d.h. ohne Datei-Header. Mit der DATA-Instruktion sind in Zusammnhang mit dem Filter WAVEFLT auch WAV-Dateien direkt einlesbar. Es können auch 16-Bit-Stereo-Daten gelesen werden, die direkt in 8-Bit-Mono gewandelt werden. Die Sample-Rate kann der Filter jedoch nicht verändern.

Interessante Effekte lassen sich erzielen, wenn die Ausgaberate von der Samplerate abweicht. Bei niedrigerer Ausgaberate werden die Töne tiefer, bei höherer Ausgaberate werden sie höher (Micky-Maus-Effekt).

Programmbeispiel:

```
'-----------------------------------------------------------------------
' Name: PWM2_SOUND1.TIG
' Gibt einen Sound aus von Sounddaten im Flash
'-----------------------------------------------------------------------
user_var_strict                    ' Vars deklarieren!
#include DEFINE_A.INC              ' allgemeine Definitionen
#include UFUNC3.INC                ' User-Function-Codes

DATALABEL Sound0, EndSound0        ' Sound Adresse im Flash
STRING pwm$(31k)                   ' PWM-String

TASK MAIN
Sound0::                           ' Sound im Flash
DATA FILE "Fifth.pcm"              ' Sound-Datei (mono, 8Bit, Rohdaten)
EndSound0::

' --------------------------------- Variables in MAIN
BYTE ever                          ' Endlosschleife
WORD slen, empty                   ' Stringlaenge, Flag
  install_device #TA, "TIMERA.TDD", 2,78 ' 8kHz
  install_device #PWM2,"PWM2.TDD",&
    0, &                           ' Kanal
    8, &                           ' Aufloesung
    0, &                           ' Frequenz
    0, &                           ' Oversample
    0, &                           ' reserviert, immer 0
    0, &                           ' Shifts
    1                              ' Pre-Scaler


  put #TA, 2,78                    ' lustig: verschiedene Geschwindigk.
  for ever = 0 to 0 step 0
    slen = EndSound0 - Sound0      ' berechne Laenge des Sounds
    put #PWM2, #0, sound0, &       ' Ausgabestring
                   0, &            ' Offset
                   slen, &         ' Laenge
                   1, &            ' Anzahl Ausgaben
                   0               ' Shift

    empty = 0                      ' nehme an: Ausgabe laeuft
    while empty = 0                ' warte solange Ausgabe laeuft
      get #PWM2, #0, #UFCI_PW2_ACT, 0, empty
    endwhile
    wait_duration 2000             ' warte 2 Sekunden
  next                             ' nochmal
END
```

Leere Seite

SER1B - Serielle Schnittstellen

Dieser Device-Treiber ermöglicht die serielle Ein- und Ausgabe auf den beiden internen seriellen Schnittstellen. Bei der Installation des Treibers können die Übertragungsparameter der seriellen Schnittstellen festgelegt. Diese sind jedoch durch Kommandos an den Treiber auch später noch veränderbar.

Weitere Informationen zu SER1B_xx.TDD:

- User-Function-Codes des SER1_xx.TDD
- Zeichen ausgeben
- Ausgabe mit Kontrolle des Puffers
- Zeichen einlesen
- Löschen eines Puffers
- RS-485-Betrieb
- RS-485 im 9.-Bit-Betrieb

Dateiname: SER1B_K8.TDD (mit 8K Puffern)
SER1B_K1.TDD (mit 1K Puffern)
SER1B_R1.TDD (mit 256 Byte Puffern)

INSTALL DEVICE #*D*, "SER1B_xx.TDD", *Baud, Data_Par, Rec.Flag, etc.*

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Baud ist ein Parameter zur Festlegung der Baudrate. Folgende Symbolischen Bezeichnungen sind in DEFINE_A.INC festgelegt:

Baudraten:

Nr.	Symbol	Bedeutung	Modul-Typ “A“
0	BD_50	50 Bd	
1	BD_75	75 Bd	
2	BD_110	110 Bd	
3	BD_150	150 Bd	
4	BD_200	200 Bd	
5	BD_300	300 Bd	verfügbar
6	BD_600	600 Bd	verfügbar
7	BD_900	900 Bd	
8	BD_1_200	1.200 Bd	verfügbar
9	BD_1_800	1.800 Bd	
10	BD_2_400	2.400 Bd	verfügbar
11	BD_3_600	3.600 Bd	
12	BD_4_800	4.800 Bd	verfügbar
13	BD_7_200	7.200 Bd	
14	BD_9_600	9.600 Bd	verfügbar
15	BD_14_400	14.400 Bd	
16	BD_19_200	19.200 Bd	verfügbar
17	BD_28_800	28.800 Bd	
18	BD_38_400	38.400 Bd	verfügbar
19	BD_57_600	57.600 Bd	
20	BD_76_800	76.800 Bd	verfügbar
21	BD_115_200	115.200 Bd	
22	BD_153_600	153.600 Bd	verfügbar
23	BD_230_400	230.400 Bd	

Data_Par ist ein Parameter zur Festlegung der Anzahl der Datenbits und der Parität. Folgende Symbolischen Bezeichnungen sind in DEFINE_A.INC festgelegt:

Anzahl Datenbits, Parität:

Nr.	Symbol	Bedeutung	Modul-Typ “A“
0	DP_7N	7 Data, No Parity	verfügbar
1	DP_7E	7 Data, Even	verfügbar
2	DP_7O	7 Data, Odd	verfügbar
3	DP_8N	8 Data, No Parity	verfügbar
4	DP_8E	8 Data, Even	verfügbar
5	DP_8O	8 Data, Odd	verfügbar
6	DP_9N	9 Data, No Parity	verfügbar
7	DP_9E	9 Data, Even	verfügbar
8	DP_9O	9 Data, Odd	verfügbar

Rec.Flag = NEIN es werden Zeichen unterdrückt, die von der Hardware als Fehlerhaft erkannt wurden.
= JA gibt eventuell fehlerhaft empfangene Zeichen weiter in den Empfangspuffer.

‘etc.’ steht für die Angaben des zweiten seriellen Kanals. Die Angaben wiederholen sich in der gleichen Form wie für den ersten Kanal.

Die Parameter des Treibers werden durch 3 Bytes pro Kanal übergeben. Diese können auf verschiedene Weise in der INSTALL_DEVICE-Instruktion angegeben werden.

Angabe der Parameter durch Kommata getrennt unter Verwendung der in ‘DEFINE_A.INC’ vordefinierten Ausdrücke:

```
#INCLUDE DEFINE_A.INC
INSTALL DEVICE #2, "SER1B_K1.TDD", BD_19_200, DP_8N, JA,&
BD_9600, DP_7E, NEIN
etc.
```

Übergabe in einem String:

```
SER$=CHR$(BD_300)+CHR$(DP_8N)+CHR$(JA),&
CHR$(BD_1200)+CHR$(DP_7E)+CHR$(NEIN)
```

```
INSTALL DEVICE #2, "SER1B_K1.TDD", SER$
```


Im Verlauf des Programms können die Schnittstellenparameter unter Angabe eines Kommandos jederzeit verändert werden, z.B.:

```
PUT #2,#0, #UFCO_SET_SERIAL, BD_300, DP_8N, JA
```

Sekundär-Adresse 0 wählt den seriellen Kanal 0, Sekundär-Adresse 1 wählt den seriellen Kanal 1 aus.

Die Unterstützung der in 'DEFINE_A.INC' definierten Schnittstellenparameter (siehe folgende Tabellen) ist abhängig von der eingesetzten Hardware (Modul-Typ, bzw. externe Bausteine).

Hinweis: BASIC-Tiger® unterstützt zunächst nur Datenübertragung mit einem Stopbit. Jedoch kann ein 2. Stopbit simuliert werden, indem 9-Bit-Betrieb gefahren wird mit gesetztem 9. Bit. Siehe 'RS-485 im 9.-Bit-Betrieb' ab Seite 209.

Sowohl eingehende als auch gesendete Daten werden in einem Puffer zwischengespeichert. Größe, Füllstand oder verbleibender Platz der Ein- und Ausgangspuffer sowie die Version des Treibers können mit Hilfe der User-Function-Codes abgefragt werden (Seiten 16ff).

User-Function-Codes des SER1_xx.TDD

User-Function-Codes für Abfragen (Instruktion GET):

Nr	Symbol Prefix UFCI_	Beschreibung
1	UFCI_IBU_FILL	Füllstand des Eingangspuffers (Byte)
2	UFCI_IBU_FREE	freier Platz im Eingangspuffer (Byte)
3	UFCI_IBU_VOL	Größe des Eingangspuffers (Byte)
33	UFCI_OBU_FILL	Füllstand des Ausgangspuffers (Byte)
34	UFCI_OBU_FREE	freier Platz im Ausgangspuffer (Byte)
35	UFCI_OBU_VOL	Größe des Ausgangspuffers (Byte)
65	UFCI_LAST_ERRC	letzter Error-Code
99	UFCI_DEV_VERS	Version des Treibers
144	UFCI_SER_STAT	gibt in WORD zurück: low Byte: Anzahl Empfangsfehler high Byte: Anzahl Pufferüberläufe
145	UFCI_SER_9STS	Status bei 9-Bit-Betrieb: 0: auf Adresse wartend 1: Daten empfangend
146	UFCI_SER_9ADR	zuletzt empfangen Adresse

Wenn nicht genügend Platz im Ausgabepuffer ist und trotzdem ausgegeben wird, dann wartet die Instruktion PUT oder PRINT (und damit die ganze Task) solange, bis wieder Platz im Puffer ist.

Beispiel: frage den Füllstand des Ausgangspuffers ab, um festzustellen, ob genug Platz für die Ausgabe ist.:

```
GET #2, #1, #UFCI_OBU_FILL, 0, wVarFill
IF wVarFill > (LEN(A$)+2) THEN     ' A$ + CR + LF
  PRINT #2, #1, A$
ENDIF
```

User-Function-Codes für die Instruktion PUT folgendes Kommando:

Nr	Symbol	Beschreibung
1	UFCO_IBU_ERASE	Eingangspuffer löschen
33	UFCO_OBU_ERASE	Ausgangspuffer löschen
94	UFCO_SET_SERIAL	setze serielle Parameter
128	UFCO_SET_ISEP	setze Separatoren für INPUT
129	UFCO_RES_ISEP	lösche Separatoren für INPUT
130	UFCO_SER_ECHO	erzeugt Echo in den Ausgabepuffer (JA/NEIN)
131	UFCO_SER_9BIT	bei 9-Bit-Betrieb: setze 9. Bit zu '0' oder '1'
132	UFCO_SER_9ADR	bei 9-Bit-Betrieb: setze Adresse dieses Moduls
133	UFCO_SER_RTS	setze RTS0 auf: 0 = nicht bereit 1 = bereit

Beispiel: setze neue Parameter auf seriellem Kanal 1. Die Parameter werden genauso angegeben wie in der INSTALL-Zeile, jedoch nur für einen Kanal:

```
PUT #2,#1, #UFCO_SET_SERIAL, BD_38_400, DP_8E, JA
```

Für die Instruktion INPUT gelten standardmäßig KOMMA und RETURN als Separatorzeichen, die eine Eingabe abschließen. Mit Hilfe des User-Function-Codes UFCO_SET_ISEP lassen sich die Separatorzeichen verändern. Bevor neue Zeichen gesetzt werden, können die bereits gesetzten Zeichen gelöscht werden. Die zu setzenden oder zu löschenden Zeichen werden als Code-Bereiche angegeben:

PUT #D, #C, UFCO_SET_ISEP, *Startcode*, *Endcode*, *Startcode*, *Endcode*

Wenn Sie die Standard-Separatoren löschen, jedoch keine neuen setzen, wird eine INPUT-Instruktion erst beendet, wenn der Input-Puffer voll ist.
Die Separatoren gelten für beide serielle Kanäle gleichermaßen.

!

Beispiel: setze neuen Separator LINE-FEED für die Instruktion INPUT auf dem seriellen Kanal 0:

```
PUT #2,#0, #UFCO_RES_ISEP, 0, 255      ' loesche alle Separatoren
PUT #2,#0, #UFCO_SET_ISEP, 10, 10      ' setze Line-Feed als Separator
```

Beispiel: setze alle Steuerzeichen sowie Zeichen ab 7Fh als Separatorzeichen für die Instruktion INPUT auf dem seriellen Kanal 0:

```
PUT #2, #0, #UFCO_RES_ISEP, 0, 255      ' loesche alle Separatoren
                                  ' setze neue Bereiche als Separatoren
PUT #2, #0, #UFCO_SET_ISEP, 0, 31, 127, 255
```

Beispiel: lösche Komma als Separatorzeichen für die Instruktion INPUT auf dem seriellen Kanal 0:

```
PUT #2, #0, #UFCO_RES_ISEP, 2ch, 2ch    ' loesche Komma als Separatoren
' oder
PUT #2, #0, #UFCO_RES_ISEP, ',,'        ' loesche Komma als Separatoren
```

Ein weiteres Beispiel:

```
PUT #1, #0, #UFCI_SET_ISEP, 'acXZ55'
' setzt als INPUT-Separatoren diese Zeichen:
'               a, b, c, X, Y, Z, 5
```

Beispiel: erzeuge Echo auf seriellem Kanal 0:

```
PUT #2, #0, #UFCO_SER_ECHO, JA
```

Zeichen ausgeben

Die Ausgabe erfolgt mit PUT, PRINT oder PRINT_USING.

Put gibt die Zeichen so aus, wie sie sind, daß heißt ohne Umwandlung in ASCII (bei Zahlen) und ohne Anfügen von CR und LF. Es ist nur ein Argument möglich.

PRINT oder PRINT_USING geben Zeichen formatiert aus. Zahlen werden in ASCII-Zeichen gewandelt. Es wird eine Sequenz CR+LF an die Ausgabe angehängt. mehrere Ausgabe-Argumente sind möglich. Semikolon und Komma haben dabei besondere Funktionen. PRINT_USING funktioniert wie PRINT, formatiert Zahlen jedoch nach vorgegebenem Format (siehe USING).

Unterschied der Ausgabe mit PUT und PRINT:

Befehlszeile	Ausgabe

PUT #SER, 255	1 Byte mit dem Wert 255
PUT #SER, 12345678h	4 Bytes (low 1st): 78h, 56h, 34h, 12h
PRINT #SER, 255	5 Bytes: ASCII 2, 5, 5, CR, LF
PRINT #SER, 12345678h	11 Bytes: “305419896”, CR, LF

Ausgabe mit Kontrolle des Puffers

Wenn nicht genügend Platz im Ausgabepuffer ist und trotzdem ausgegeben wird, dann wartet die Instruktion PUT oder PRINT (und damit die ganze Task) solange, bis wieder Platz im Puffer ist. Dieses Warten kann verhindert werden, indem vor der Ausgabe der freie Platz im Puffer abgefragt wird.

Beispiel: gebe nur aus, wenn noch genügend freier Platz im Ausgangspuffers ist:

```
GET #SER, #0, #UFCI_OBU_FREE, 0, FREE ' erfrage Platz in Puffer
IF FREE > 3 THEN                ' wenn mindestens 4 Bytes frei
  PUT #SER, A                   ' einmal mit PUT
ENDIF
```

Zeichen einlesen

Empfangene Zeichen werden vom Device-Treiber im Eingangspuffer abgelegt. Es gibt für jeden Kanal einen Puffer. Mit GET, INPUT oder INPUT_LINE werden die Zeichen gelesen.

INPUT und INPUT_LINE lesen (und warten) solange, bis ein Separatorzeichen gekommen ist. Der Separator trennt zwei Eingaben. Das gängigste Separatorzeichen für Eingaben ist das Zeichen RETURN.

GET wartet nicht und liest nur etwas, wenn auch etwas im Puffer gewesen ist. Wird in eine numerische Variable gelesen, dann erhält diese den Wert Null, wenn

- nichts im Puffer war
- wenn eine Null im Puffer war

Um die beiden Fälle zu unterscheiden, wird vor dem Lesen abgefragt, ob der Puffer (genügend) Zeichen enthält.

Beispiel: prüfe, ob mindestens ein Byte im Eingangspuffers ist:

```
GET #SER, #0, #UFCI_IBU_FILL, 0, FI ' lese Eingangsp.Fuellstand
IF FI > 0 THEN                ' wenn etwas im Puffer ist
  GET #SER, #0, 1, B          ' dann lies ein Byte
ENDIF
```


Das folgende Beispielprogramm zeigt die verschiedenen Fälle der Ausgabe sowie die Abfrage des Puffers vor der Eingabe. Schließen Sie ein Terminal an und steppen Sie das Programm durch.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name: SER1B_1.TIG
' sendet und empfängt Zeichen seriell und zeigt sie auf dem LCD an
' Terminal an SER0 anschliessen
'--------------------------------------------------------------------
#INCLUDE DEFINE_A.INC              ' allgemeine Definitionen
#INCLUDE UFUNC3.INC                ' User Function Codes

TASK MAIN
  BYTE EVER                        ' fuer Endlosschleife
  WORD FREE                        ' freier Platz im Puffer
  WORD FI                          ' Fuellstand des Puffers
  LONG A                           '

  INSTALL_DEVICE #SER, "SER1B_K1.TDD", & ' SER-Treiber installieren
    BD_9_600, DP_8N, YES, &        ' Einstellung SER0
    BD_9_600, DP_8N, YES           ' Einstellung SER1
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #LCD, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #LCD, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8

  PUT #SER, "abcd"                 ' gibt 4 Bytes auf Kanal 0 aus
  PRINT #SER, "abcd"               ' druckt auf Kanal 0 (6 Bytes)
  PRINT #SER, "abcd";              ' druckt auf Kanal 0 (4 Bytes)
  PRINT #SER, #0, "abcd"           ' druckt auch auf Kanal 0

  A = 44434241h                    ' A enthaelt ASCII "ABCD"
  GET #SER, #0, #UFCI_OBU_FREE, 0, FREE ' erfrage Platz in Puffer
  IF FREE > 3 THEN                 ' wenn mindestens 4 Bytes frei
    PUT #SER, A                    ' einmal mit PUT
  ENDIF

  PRINT #SER, A                    ' einmal mit PRINT
                                   ' 44434241h = dezimal 1145258561
  USING "UH<8><8>   0 0 0 0 8"     ' formatiere als HEX, 8 Ziffern
  PRINT_USING #SER, A

  FOR EVER = 0 TO 0 STEP 0         ' Endlosschleife
    GET #SER, #0, #UFCI_IBU_FILL, 0, FI ' lese Eingangsp.Fuellstand
    IF FI > 0 THEN                 ' wenn etwas im Puffer ist
      GET #SER, #0, 1, B           ' dann lies ein Byte
      PRINT #LCD, B;               ' zeige es als Tahl an
    ENDIF
  NEXT
END
```

RS-485-Betrieb

Beide Kanäle der seriellen Schnittstelle lassen sich auch als RS-485-Schnittstelle betreiben. Dazu wird ein Steuerpin benötigt, der den RS-485-Sender einschaltet, wenn gesendet werden soll, und den Senderbaustein wieder abschaltet, nachdem das letzte Bit gesendet worden ist.

Zusätzliche Parameter bei der Installation des Treibers legen die Steuerpins sowie die Polarität des Steuerpins fest:

INSTALL DEVICE #*D*, "SER1B_K1.TDD" *[, P1,..., P12]*

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

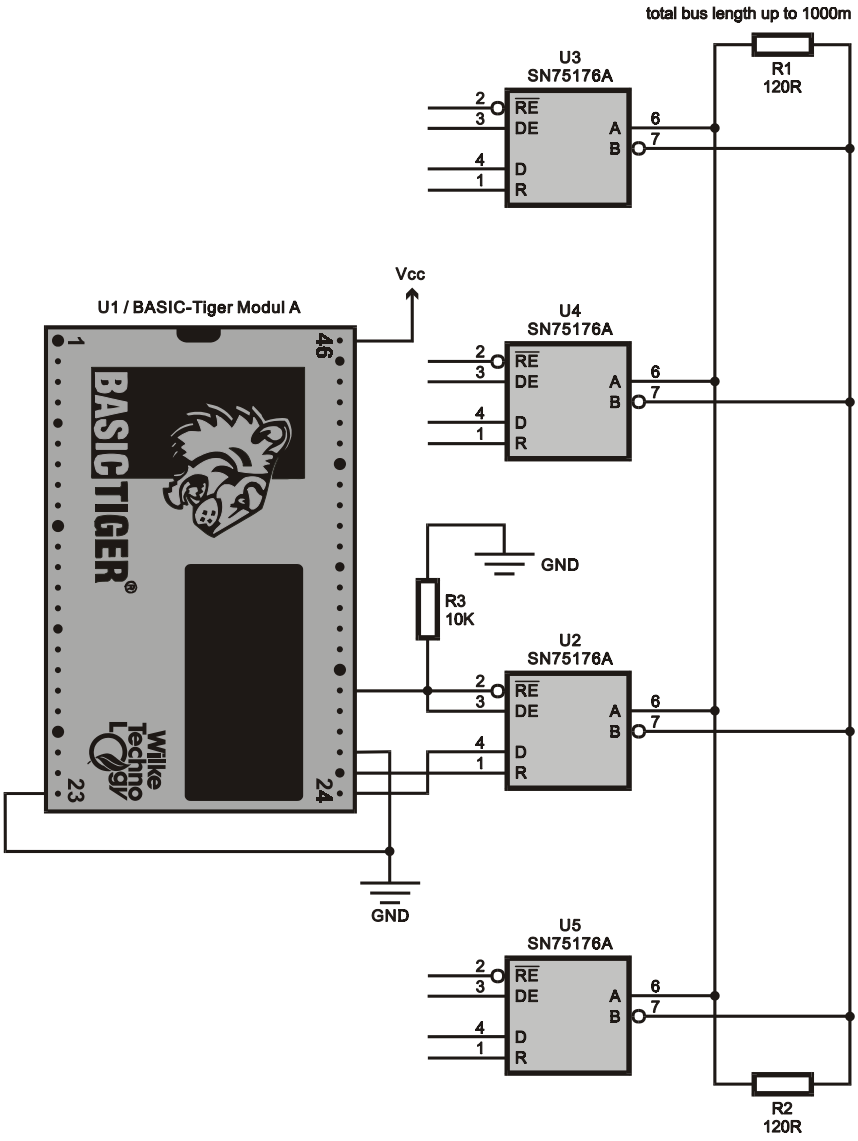
Für den seriellen Kanal 0 kann der bereits vorhandene RTS0-Pin als Steuerpin angegeben werden, für den seriellen Kanal 1 steht dieser Pin in keinem Fall zur Verfügung, es kann jedoch irgendein anderer freier interner I/O-Pin verwendet werden.

Im RS-485-Betrieb wird der Treibersteuerpin ca. 25µsec vor dem Senden eingeschaltet und ca. 25µsec nachdem das letzte Bit gesendet worden ist wieder ausgeschaltet.

	unverändert lassen	Beschreibung des Parameters
P1	0EEH	ist ein Parameter zur Festlegung der Baudrate, Kanal 0
P2	0EEH	ist ein Parameter zur Festlegung der Anzahl der Datenbits und der Parität, Kanal 0
P3	0EEH	= NEIN: werden Zeichen unterdrückt, die von der Hardware als Fehlerhaft erkannt wurden. = JA: gibt eventuell fehlerhaft empfangene Zeichen weiter in den Empfangspuffer, Kanal 0
P4	0EEH	ist ein Parameter zur Festlegung der Baudrate, Kanal 1
P5	0EEH	ist ein Parameter zur Festlegung der Anzahl der Datenbits und der Parität, Kanal 1
P6	0EEH	= NEIN: werden Zeichen unterdrückt, die von der Hardware als Fehlerhaft erkannt wurden. = JA: gibt eventuell fehlerhaft empfangene Zeichen weiter in den Empfangspuffer, Kanal 1
P7	0EEH	Bit-Maske für Transmit-Enable-Pin, Kanal 0
P8	0	Logische Adresse für Transmit-Enable, Kanal 0
P9	0EEH	0: Transmit-Enable ist high-Pegel 1: Transmit-Enable ist low-Pegel
P10	0EEH	Bit-Maske für Transmit-Enable-Pin, Kanal 1
P11	0	Logische Port-Adresse Transmit-Enable, Kanal 1
P12	0EEH	0: Transmit-Enable ist high-Pegel 1: Transmit-Enable ist low-Pegel

In der Hardware sollte ein entsprechender Widerstand die Steuerleitung vor der Initialisierung des Moduls auf dem Pegel für ‚nicht enabled' halten, da andernfalls ein nicht initialisiertes Modul den RS-485-Bus blockieren kann.

Der Handshake-Pin des Kanals 0 (CTS) muß im RS-485-Betrieb fest auf 'bereit' gelegt werden, also auf TTL-low.



Programmbeispiel:

```
'-----------------------------------------------------------------------
' Name: SER_485.TIG
'-----------------------------------------------------------------------
#INCLUDE DEFINE_A.INC                  ' allgemeine Definitionen

TASK MAIN
' installiere beide seriellen Kanaele fuer RS-485-Treiber-Chip
' setze Port 9, Pin 5 (RTS) als Transmit-Control fuer SER0
' setze Port 8, Pin 0 als Transmit-Control fuer SER1
' setze beide Steuerpins 1 => enabled=high
  INSTALL_DEVICE #2, "SER1B_K8.TDD",&
  BD_19_200, DP_8N, JA, BD_19_200, DP_8N, JA,00100000b,9,0,00000001b,8,0

  PRINT #2, "hello world"              ' drucke auf beiden Kanaelen
  PRINT #2, #1, "again hello world"
END
```

RS-485 im 9.-Bit-Betrieb

BASIC-Tiger® unterstützt RS-485-Betrieb im 9-Bit-Betrieb. In der Vernetzung erhalten alle angeschlossenen Teilnehmer eine Adresse (1 Byte). Auf dem Bus werden Adressen durch ein gesetztes 9. Bit gekennzeichnet, während Daten ein nicht-gesetztes 9. Bit erhalten. Diese Vorgehensweise erleichtert die Entscheidung, ob gesendete Daten auf dem Bus für das Modul bestimmt sind oder nicht, und entlastet so die CPU. Stimmt die Adresse auf dem Bus mit der Moduladresse überein, dann werden die folgenden Daten solange im Modul in den Empfangspuffer geschrieben, bis eine neue 'ungültige' Adresse auf dem Bus erscheint. Adressen werden nicht im Empfangspuffer gespeichert.

Beachte: im 9-Bit-Betrieb gibt es kein RTS Handshake, kein Error-Counting und kein Local-Echo.

Der Status, ob das Modul gerade auf eine Adresse wartet (nichts empfängt), oder ob gerade Daten empfangen werden, kann mit User-Function-Codes abgefragt werden:

```
GET #2,#0, #UFCI_SER_9STS, 0, STATUS
```

Adresse aufheben: das Setzen einer Adresse mit dem Wert 100h (256) löscht die 9-Bit-Adresse.

Das folgende Beispielprogramm zeigt lediglich, wie das 9. Bit mit den User-Function-Codes kontrolliert und wie eine Modul-Adresse gesetzt wird. Eine echte Anwendung erfordert zumindest einen Sender auf dem Bus und einen Empfänger. Eine Rahmenprogramm für eine solche Anwendung bestehend aus Senderprogramm und Empfängerprogramm finden Sie im Kapitel Applikationen unter der Überschrift SER1B9M.TIG (Master) und SER1B9S.TIG (Slave)

Programmbeispiel:

```
'-----------------------------------------------------------------------
' Name: SER1_9B.TIG
'-----------------------------------------------------------------------
#INCLUDE UFUNC3.INC

TASK MAIN
' installiere beide seriellen Kanaele fuer RS-485-Treiber-Chip
' setze Port 9, Pin 5 (RTS) als Transmit-Control fuer SER0
' setze Port 8, Pin 0 als Transmit-Control fuer SER1
' setze beide Steuerpins 1 => enabled=high
  INSTALL_DEVICE #2, "SER1B_K8.TDD",&
  BD_19_200, DP_8N, JA, BD_19_200, DP_8N, JA,00100000b,9,0,00000001b,8,0

  PUT #SER, #0, #UFCO_SER_9ADR, 85      ' setze Modul-Adresse = 85

  FOR I = 0 TO 0 STEP 0                 ' endlose Schleife
    PUT #SER, #0, #UFCO_SER_9BIT, 0     ' setze Bit 9 zurueck
    PUT #SER, #1, #UFCO_SER_9BIT, 0     ' Kanal 1 auch
    PUT #SER, #0, "A"
    PUT #SER, #1, "B"
    WAIT_DURATION 500                   ' warte eine halbe Sekunde
    PUT #SER, #0, #UFCO_SER_9BIT, 1     ' setze Bit 9
    PUT #SER, #1, #UFCO_SER_9BIT, 1     ' Kanal 1 auch
    PUT #SER, #0, "A"
    PUT #SER, #1, "B"
    WAIT_DURATION 500                   ' warte eine halbe Sekunde
  NEXT
END
```

Adresse aufheben: das Setzen einer Adresse mit dem Wert 100h (256) löscht die 9-Bit-Adresse.

Master und Slave mit 9-Bit-Adressen

Ein einfaches praktisches Beispiel zeigt der als 'Master' bezeichnete Sender eine Adresse mit gesetztem 9. Bit und anschliessend Daten sendet. Verbinden Sie einen ‚Slave' mit SER1_9B_SLAVE.TIG. Da das Slave-Programm die Daten mit PRINT anzeigt, sendet der Master in diesem Test nur ASCII-Zeichen.

Bei der Verwendung der 9-Bit-Adressen ist folgendes zu beachten: wenn Sie das Kommando geben, Bit 9 z.B. auf ‚high' zu setzen, dann wird das sofort ausgeführt (ein Bit in der UART). Davon betroffen sind dann auch Zeichen, die evtl. noch im Ausgabepuffer sind. Es muss daher vorher den Pufferstand abgefragt und gewartet werden. Umgekehrt gilt: das Senden von Daten dauert, wenn es sich um mehr als ein Zeichen handelt, evtl. länger als der PUT- oder PRINT-Befehl, der die Daten ja nur im Puffer ablegt. Der Treiber sendet dann die Zeichen so schnell wie die Interrupts von der UART kommen. Wenn dann das Bit 9 vorzeitig gesetzt wird, dann werden die Daten mit gesetztem 9. Bit gesendet.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name: SER1_9B_MASTER.TIG
' Master sendet eine Adresse mit gesetztem 9. Bit und anschliessend
' Daten. Verbinde einen Slave mit SER1_9B_SLAVE.TIG
'--------------------------------------------------------------------
user var strict                    ' unbedingte Var.deklaration
#include UFUNC3.INC                ' User Function Codes
#include DEFINE_A.INC              ' allg. Symbol-Definitionen

#define CH9 0                      ' benutzter Kanal fuer 9-Bit-485
#define MASTER 99                  ' Adresse des Masters
#define SLAVE1 1                   ' Adresse von Slave 1
#define SLAVE2 2                   ' Adresse von Slave 2

TASK MAIN
  BYTE ever

' installiere beide seriellen Kanaele fuer RS-485-Treiber-Chip
' setze Port 9, Pin 5 (RTS) als Transmit-Control fuer SER0
' setze Port 8, Pin 0 als Transmit-Control fuer SER1
' setze beide Steuerpins 1 => enabled=high
  install_device #SER, "SER1B_K1.TDD", &
    BD_19_200, DP_9N, YES, &       ' Einstellung SER0
    BD_19_200, DP_9N, YES          ' Einstellung SER1
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8

  put #SER, #CH9, #UFCO_SER_9ADR, MASTER ' setze Modul-Adresse

  for ever = 0 to 0 step 0             ' Endlosschleife
    put #SER, #CH9, #UFCO_SER_9BIT, 1  ' setze Bit 9
    put #SER, #CH9, SLAVE1             ' sende Adresse SLAVE1
    wait_duration 3                    ' warte bis gesendet

    put #SER, #CH9, #UFCO_SER_9BIT, 0  ' setze Bit 9 zurueck
    print #SER, #CH9, "hello slave 1"  ' sende Nachricht an SLAVE1
    print #LCD, "<1>sent to slave 1";  ' zeige es auf dem LCD an
    wait_duration 20                   ' warte bis gesendet
                                       ' ----------------------------
    put #SER, #CH9, #UFCO_SER_9BIT, 1  ' setze Bit 9
    put #SER, #CH9, SLAVE2             ' sende Adresse SLAVE2
    wait_duration 3                    ' warte bis gesendet

    put #SER, #CH9, #UFCO_SER_9BIT, 0  ' setze Bit 9 zurueck
    print #SER, #CH9, "hello slave 2"  ' sende Nachricht an SLAVE2
    print #LCD, "<1>sent to slave 2";  ' zeige es auf dem LCD an
    wait_duration 200                  ' Schleifengeschwindigkeit
  next
END
```

Das Modul, welches als ‚Slave' bezeichnet ist, setzt seine Adresse und erhält vom Master (ASCII-)Daten. Durch die 9-Bit-Adressen filtert der Treiber die Daten. Verbinden Sie einen Master mit SER1_9B_MASTER.TIG.

Programmbeispiel:


```
'---------------------------------------------------------------------
' Name: SER1_9B_SLAVE.TIG
' Slave setzt seine Adresse und erhaelt vom Master (ASCII-)Daten.
' Durch die 9-Bit-Adressen filtert der Treiber die Daten.
' Verbinde einen Master mit SER1_9B_MASTER.TIG
'---------------------------------------------------------------------
user var strict                  ' unbedingte Var.deklaration
#include UFUNC3.INC              ' User Function Codes
#include DEFINE_A.INC            ' allg. Symbol-Definitionen

#define CH9 0                    ' benutzter Kanal fuer 9-Bit-485
#define MASTER 99                ' Adresse des Masters
#define SLAVE1 1                 ' Adresse von Slave 1
#define SLAVE2 2                 ' Adresse von Slave 2

TASK MAIN
  BYTE ever
  WORD fi                        ' Pufferfuellstand
  STRING c$

' installiere beide seriellen Kanaele fuer RS-485-Treiber-Chip
' setze Port 9, Pin 5 (RTS) als Transmit-Control fuer SER0
' setze Port 8, Pin 0 als Transmit-Control fuer SER1
' setze beide Steuerpins 1 => enabled=high
  install_device #SER, "SER1B_K1.TDD", &
    BD_19_200, DP_9N, YES, &     ' Einstellung SER0
    BD_19_200, DP_9N, YES        ' Einstellung SER1
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8

  put #SER, #CH9, #UFCO_SER_9ADR, SLAVE2 ' setze Modul-Adresse

  for ever = 0 to 0 step 0       ' Endlosschleife
    get #SER, #CH9, #UFCI_IBU_FILL, 0, fi
    while fi > 0                 ' solange was im Puffer ist
      get #SER, #CH9, 1, c$      ' lies Byte
      print #LCD, c$;            ' zeige es auf dem LCD an
    endwhile                     ' ASCII and CRLF only
  next
END
```

SER2 - Serielle Schnittstellen durch Software

Dieser Device-Treiber ermöglicht die asynchrone serielle Ein- und Ausgabe auf internen I/O-Pins. Die Schnittsteelle ist rein softwaremäßig realisiert. Bei der Installation des Treibers wird durch den Dateinamen festgelegt, an welchen Pins die serielle Eingabe und Ausgabe stattfindet. Die Baudrate ist durch die Einstellung des TIMERA sowie den Prescaler des Device-Treibers festgelegt.

Der Treiber kann auf individuelle Anforderungen eingestellt werden:

- RxD + TxD: Kanäle einzeln zu-/abschaltbar.
- RxD + TxD: jeweils mit 256 Byte FiFo-Puffer ausgestattet.
- RxD + TxD: mit Flusskontrolle: RTS / CTS zu-/abschaltbar.
- TxD: RS-485 Buszugriffs-Kontrolle TE zu-/abschaltbar.
- Data-Bits: Datenformat: 1...8 Bits.
- Parity-Bit: No, Even, Odd, Mark, Space.
- Baudraten: quasi-stufenlose Baudraten über TIMERA und Prescaler im Bereich von 12 kBd-Senden (bei 1 x TxD) bis zu 3 Bd.
- Pegel: TRUE + INVERSE Pegel möglich für RS-232 mit/ohne Leitungs-Treiber.
- PINs: RxD, TxD, RTS, CTS und TE lassen sich auf fast jeden beliebigen I/O-Pin des Tigers legen.
- Kanäle: bis zu 8 serielle Kanäle (RxD / TxD in beliebiger Mischung).

Beachte: SER2_XX.TDD belastet die CPU um ein Vielfaches mehr als ein Treiber wie SER1B_xx.TDD, da für jedes einzelne Bit mehrere System-Task-Aufrufe erfolgen. Beim Einsatz dieses Treibers sollte daher berücksichtigt werden:

- SER2 nur einsetzen, wenn noch freie CPU Leistung verfügbar ist.
- Gesamt-Baudrate aller RxD und TxD Kanäle nicht zu hoch wählen:
- Bei derzeitigen Modulen: Summe = max. ca. 10 kBaud.
- z.B: 5 x TxD a 1200 Bd oder: 1 x RxD + TxD a 4800 Bd.
- Bei hoher CPU-Auslastung kann die Debug-Funktion beeinträchtigt werden.

Dateiname: SER2_pp.TDD

INSTALL DEVICE #*D*, "SER2_pp.TDD", *P1,...P7*

Hinweis: TIMERA.TDD muß vorher eingebunden werden.

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

pp im Dateinamen steht für die Position des Sendepins (Port,Pin). Eine Tabelle weiter unten im Text zeigt die Lage der Pins, die sich durch die Wahl des Device-Treibers ergibt.

P1...P7 folgende Tabelle zeigt die Bedeutung der Parameter P1 bis P7:

	unverändert lassen	**Beschreibung des Parameters**
P1	0EEH	ist ein Parameter zur Festlegung der Anzahl der Datenbits. Wert: 1...8
P2	0EEH	ist ein Parameter zur Festlegung der Parität: 0 = NO 1 = SPACE 2 = Even 3 = Odd 4 = MARK
P3	0EEH	0 = TRUE 1 = INVERS
P4	-	Transmitter Pre-Scaler 0 = kein Transmitter vorhanden 1 = ohne Prescaler 2...255 = Prescaler Faktor
P5	-	Receive Oversample 0,1,2 = kein Receiver vorhanden 3...255 = Oversample-Faktor
P6	-	Reserviert, immer 1.
P7	0EEH	Hardware-Hanshake Pins: untere 3 Bits: 000 = keine Handshake-Pins 001 = CTS-Pin (Eingang, steuert Sendeaktivität) 010 = RTS-Pin (Ausgang, zeigt an, ob RxD Platz im Puffer hat) 011 = RTS+CTS 100 = Transmitter-Enable f. RS-485 (Ausgang, zeigt an ob Data im TxD-Puffer)

Der Device-Treiber verwendet ein bis vier I/O-Pins, die fast beliebig auf die internen I/O Pins des Tiger-Moduls gelegt werden können. Folgende Tabelle zeigt, welche Zuordnungen durch die Wahl der passenden Treiberdatei möglich sind:

Treiber-Name	TxD (out)	RxD (in)	CTS (in) or TE (out)	RTS (out)
SER2_33.TDD	L33	L34	L35	L70
SER2_34.TDD	L34	L35	L70	L71
SER2_35.TDD	L35	L70	L71	L72
SER2_40.TDD	L40	L42	L33	L34
SER2_42.TDD	L42	L33	L34	L35
SER2_60.TDD	L60	L61	L62	L63
SER2_61.TDD	L61	L62	L63	L64
SER2_62.TDD	L62	L63	L64	L65
SER2_63.TDD	L63	L64	L65	L66
SER2_64.TDD	L64	L65	L66	L67
SER2_65.TDD	L65	L66	L67	L40
SER2_66.TDD	L66	L67	L40	L42
SER2_67.TDD	L67	L40	L42	L33
SER2_70.TDD	L70	L71	L72	L73
SER2_71.TDD	L71	L72	L73	L60
SER2_72.TDD	L72	L73	L60	L61
SER2_73.TDD	L73	L60	L61	L62
SER2_80.TDD	L80	L81	L82	L83
SER2_81.TDD	L81	L82	L83	L84
SER2_82.TDD	L82	L83	L84	L85
SER2_83.TDD	L83	L84	L85	L86
SER2_84.TDD	L84	L85	L86	L87
SER2_85.TDD	L85	L86	L87	L70
SER2_86.TDD	L86	L87	L70	L71
SER2_87.TDD	L87	L70	L71	L72

Sowohl eingehende als auch gesendete Daten werden in einem Puffer von 256 Byte zwischengespeichert. Größe, Füllstand oder verbleibender Platz der Ein- und Ausgangspuffer sowie die Version des Treibers können mit Hilfe der User-Function-Codes abgefragt werden.

User-Function-Codes des SER2_xx.TDD

User-Function-Codes für Abfragen (Instruktion GET):

Nr	Symbol Prefix UFCI_	Beschreibung
1	UFCI_IBU_FILL	Füllstand des Eingangspuffers (Byte)
2	UFCI_IBU_FREE	freier Platz im Eingangspuffer (Byte)
3	UFCI_IBU_VOL	Größe des Eingangspuffers (Byte)
33	UFCI_OBU_FILL	Füllstand des Ausgangspuffers (Byte)
34	UFCI_OBU_FREE	freier Platz im Ausgangspuffer (Byte)
35	UFCI_OBU_VOL	Größe des Ausgangspuffers (Byte)
65	UFCI_LAST_ERRC	letzter Error-Code
99	UFCI_DEV_VERS	Version des Treibers

Wenn nicht genügend Platz im Ausgabepuffer ist und trotzdem ausgegeben wird, dann wartet die Instruktion PUT oder PRINT (und damit die ganze Task) solange, bis wieder Platz im Puffer ist.

Beispiel: frage den Füllstand des Ausgangspuffers ab, um festzustellen, ob genug Platz für die Ausgabe ist.:

```
GET #2, #0, #UFCI_OBU_FILL, 0, wVarFill
IF wVarFill > (LEN(A$)+2) THEN     ' A$ + CR + LF
  PRINT #2, #0, A$
ENDIF
```

User-Function-Codes für die Instruktion PUT folgendes Kommando:

Nr	Symbol	Beschreibung
1	UFCO_IBU_ERASE	Eingangspuffer löschen
33	UFCO_OBU_ERASE	Ausgangspuffer löschen
94	UFCO_SET_SERIAL	setze serielle Parameter Parameter 1...5 genau wie bei INSTALL Hier können keine I/O-Pins mehr hinzugefügt oder entfernt werden.
128	UFCO_SET_ISEP	setze Separatoren für INPUT
129	UFCO_RES_ISEP	lösche Separatoren für INPUT

Beispiel: setze neue Parameter auf seriellem Kanal. Die Parameter werden genauso angegeben wie in der INSTALL-Zeile, jedoch sind nur die ersten 5 Parameter zugelassen:

```
'       data,par,inv,TxPre,RxOvs,-,handshake
PUT #SER2, 8,  3,  1,    3,    3,1,   0
```

Für die Instruktion INPUT gelten standardmäßig KOMMA und RETURN als Separatorzeichen, die eine Eingabe abschließen. Mit Hilfe des User-Function-Codes UFCO_SET_ISEP lassen sich die Separatorzeichen verändern. Bevor neue Zeichen gesetzt werden, können die bereits gesetzten Zeichen gelöscht werden. Die zu setzenden oder zu löschenden Zeichen werden als Code-Bereiche angegeben:

PUT #D, #C, UFCO_SET_ISEP, *Startcode*, *Endcode*, *Startcode*, *Endcode*

! Wenn Sie die Standard-Separatoren löschen, jedoch keine neuen setzen, wird eine INPUT-Instruktion erst beendet, wenn der Input-Puffer voll ist.

Beispiel: setze neuen Separator LINE-FEED für die Instruktion INPUT auf dem seriellen Kanal 0:

```
PUT #2,#0, #UFCO_RES_ISEP, 0, 255      ' loesche alle Separatoren
PUT #2,#0, #UFCO_SET_ISEP, 10, 10      ' setze Line-Feed als Separator
```

Beispiel: setze alle Steuerzeichen sowie Zeichen ab 7Fh als Separatorzeichen für die Instruktion INPUT auf dem seriellen Kanal 0:

```
PUT #2, #0, #UFCO_RES_ISEP, 0, 255      ' loesche alle Separatoren
                                  ' setze neue Bereiche als Separatoren
PUT #2, #0, #UFCO_SET_ISEP, 0, 31, 127, 255
```

Beispiel: lösche Komma als Separatorzeichen für die Instruktion INPUT auf dem seriellen Kanal 0:

```
PUT #2, #0, #UFCO_RES_ISEP, 2ch, 2ch    ' loesche Komma als Separatoren
' oder
PUT #2, #0, #UFCO_RES_ISEP, ',,'        ' loesche Komma als Separatoren
```

Ein weiteres Beispiel:

```
PUT #1, #0, #UFCI_SET_ISEP, 'acXZ55'
' setzt als INPUT-Separatoren diese Zeichen:
'              a, b, c, X, Y, Z, 5
```

Beispiel: erzeuge Echo auf seriellem Kanal 0:

```
PUT #2, #0, #UFCO_SER_ECHO, YES
```

Das Beispielprogramm sendet auf dem Pin L80 (TxD und empfängt auf dem Pin L81 (RxD). Verbinden Sie beide Pins.

Programmbeispiel:

```
'-----------------------------------------------------------------------
' Name: SER2.TIG
' sendet Zeichen und zeigt empfangene Zeichen an
' Pins L80 (TxD) und L81 (RxD) verbinden
'-----------------------------------------------------------------------
user_var_strict                  ' Vars deklarieren!
#include DEFINE_A.INC            ' allgemeine Definitionen
#include UFUNC3.INC              ' User-Function-Codes

TASK MAIN
  BYTE I
  STRING A$

' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #lcd, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #lcd, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  INSTALL_DEVICE #TA, "TIMERA.TDD",2,173 ' 3612Hz fuer 1200baud
  INSTALL_DEVICE #SER2,"SER2_80.TDD", &
    8, &  ' Datenbits
    0, &  ' Paritaet 0=keine
    0, &  ' invert 0=true, 1=invers
    3, &  ' tx Prescaler
    3, &  ' rx Oversample
    1, &  ' reserviert, immer 1
    0     ' Handshake, 0=kein

  PUT #SER2, "hello world<13><10>"     ' Ausgabe mit PUT
  PRINT #SER2, "again hello world"     ' Ausgabe mit PRINT
  FOR I = 0 TO 0 STEP 0
    GET #SER2, 1, A$                   ' lies empfangene Zeichen
    PRINT #LCD, A$;                    ' zeige auf LCD
  NEXT
END
```

Das folgende Beispiel ist nützlich zum Experimentieren, da auf Tastendruck einzelne Zeichen gesendet sowie empfangene Zeichen angezeigt werden.

Programmbeispiel:


```
'--------------------------------------------------------------------
' Name: SER2A.TIG
' liest Zeichen von Tastatur des Plug & Play Labs
' sendet diese Zeichen auf SER2, und zeigt von SER2 empfangene
' Zeichen als ASCII-Code auf dem LCD an
' Pins L80 (TxD) und L81 (RxD) verbinden
'--------------------------------------------------------------------
user_var_strict                  ' Vars deklarieren!
#include DEFINE_A.INC            ' allgemeine Definitionen
#include UFUNC3.INC              ' User-Function-Codes
#include KEYB_PP.INC             ' fuer Tastatur des Plug & Play Labs

TASK MAIN
  BYTE ever
  STRING key$, s$                ' Taste und serielles Zeichen

' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #LCD, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #LCD, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  install_device #TA, "TIMERA.TDD",2,173 ' 3612Hz fuer 1200baud

  INSTALL_DEVICE #SER2,"SER2_80.TDD", &
    8, &  ' Datenbits
    0, &  ' Paritaet 0=keine
    0, &  ' invert 0=true, 1=invers
    3, &  ' tx Prescaler
    3, &  ' rx Oversample
    1, &  ' reserviert, immer 1
    0     ' Handshake, 0=kein

  call init_keyb ( LCD )              ' Tastaturtreiber initialisieren

  for ever = 0 to 0 step 0            ' Endlosschleife
    get #SER2, 1, s$                  ' lies empfangene Zeichen
    if s$ <> "" then                  ' wenn serielles Zeichen da
      print #LCD, asc(s$);            ' auf LCD als ASCII Code zeigen
    endif
    get #LCD, 1, key$                 ' lies Tastatur
    if key$ <> "" then                ' wenn Taste da
      put #SER2, key$                 ' auf (soft)seriell senden
    endif
    wait_duration 50                  ' Schleifengeschwindigkeit
  next
END
```

SER4 - Seriell direkt in Strings mit bis zu 614200baud

Dieser Device-Treiber ist weitgehend kompatibel zu SER1_xx.TDD, ermöglicht jedoch höhere Baudraten (bis 614200baud) sowie die serielle Ein- und Ausgabe direkt in Strings, die die Funktion der Ein-/Ausgangspuffer übernehmen. SER4 unterstützt dadurch einen hohen Datendurchsatz insbesondere in den höheren Baudraten.

Weitere Informationen zu SER1B_xx.TDD:

- Kontrolle des Datenflusses im DACC-Modus
- Datenausgabe im DACC-Modus
- Datenausgabe mit Reload im DACC-Modus
- Datenempfang im DACC-Modus
- Datenempfang mit Reload im DACC-Modus
- Besonderheiten des SER4 Device-Treibers

Dateiname: SER4_K1.TDD

INSTALL DEVICE #*D*, "SER4_K1.TDD", *P1,...P12*

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

P1...P12 folgende Tabelle zeigt die Bedeutung der Parameter P1 bis P12:

	unverändert lassen	Beschreibung des Parameters
P1	0EEH	ist ein Parameter zur Festlegung der Baudrate, Kanal 0
P2	0EEH	ist ein Parameter zur Festlegung der Anzahl der Datenbits und der Parität, Kanal 0
P3	0EEH	Einstellung Kanal 0: 0: intern gepuffert, gibt eventuell fehlerhaft empfangene Zeichen weiter in den Empfangspuffer 1: intern gepuffert, Zeichen, die von der Hardware als Fehlerhaft erkannt wurden, werden unterdrückt 2: DACC-gepuffert (direct access), Daten werden direkt in einen String geschrieben. Eventuell fehlerhaft empfangene Zeichen werden immer weitergegeben.
P4	0EEH	ist ein Parameter zur Festlegung der Baudrate, Kanal 1
P5	0EEH	ist ein Parameter zur Festlegung der Anzahl der Datenbits und der Parität, Kanal 1
P6	0EEH	Einstellung Kanal 1: 0: intern gepuffert, gibt eventuell fehlerhaft empfangene Zeichen weiter in den Empfangspuffer 1: intern gepuffert, Zeichen, die von der Hardware als Fehlerhaft erkannt wurden, werden unterdrückt 2: DACC-gepuffert (direct access), Daten werden direkt in einen String geschrieben. Eventuell fehlerhaft empfangene Zeichen werden immer weitergegeben.
P7	0EEH	Bit-Maske für Transmit-Enable-Pin, Kanal 0
P8	0	Logische Adresse für Transmit-Enable, Kanal 0
P9	0EEH	0: Transmit-Enable ist high-Pegel 1: Transmit-Enable ist low-Pegel
P10	0EEH	Bit-Maske für Transmit-Enable-Pin, Kanal 1
P11	0	Logische Port-Adresse Transmit-Enable, Kanal 1
P12	0EEH	0: Transmit-Enable ist high-Pegel 1: Transmit-Enable ist low-Pegel

Baud ist ein Parameter zur Festlegung der Baudrate. Folgende Symbolischen Bezeichnungen sind in DEFINE_A.INC festgelegt:

Baudraten:

Nr.	Symbol	Bedeutung	Modul-Typ “A“
0	BD_50	50 Bd	
1	BD_75	75 Bd	
2	BD_110	110 Bd	
3	BD_150	150 Bd	
4	BD_200	200 Bd	
5	BD_300	300 Bd	verfügbar
6	BD_600	600 Bd	verfügbar
7	BD_900	900 Bd	
8	BD_1_200	1.200 Bd	verfügbar
9	BD_1_800	1.800 Bd	
10	BD_2_400	2.400 Bd	verfügbar
11	BD_3_600	3.600 Bd	
12	BD_4_800	4.800 Bd	verfügbar
13	BD_7_200	7.200 Bd	
14	BD_9_600	9.600 Bd	verfügbar
15	BD_14_400	14.400 Bd	
16	BD_19_200	19.200 Bd	verfügbar
17	BD_28_800	28.800 Bd	
18	BD_38_400	38.400 Bd	verfügbar
19	BD_57_600	57.600 Bd	
20	BD_76_800	76.800 Bd	verfügbar
21	BD_115_200	115.200 Bd	
22	BD_153_600	153.600 Bd	verfügbar
23	BD_230_400	230.400 Bd	
24			
25			
26	BD_614_400	614.400 Bd	verfügbar

Data_Par ist ein Parameter zur Festlegung der Anzahl der Datenbits und der Parität. Folgende Symbolischen Bezeichnungen sind in DEFINE_A.INC festgelegt:

Anzahl Datenbits, Parität:

Nr.	Symbol	Bedeutung	Modul-Typ "A"
0	DP_7N	7 Data, No Parity	verfügbar
1	DP_7E	7 Data, Even	verfügbar
2	DP_7O	7 Data, Odd	verfügbar
3	DP_8N	8 Data, No Parity	verfügbar
4	DP_8E	8 Data, Even	verfügbar
5	DP_8O	8 Data, Odd	verfügbar
6	DP_9N	9 Data, No Parity	verfügbar
7	DP_9E	9 Data, Even	verfügbar
8	DP_9O	9 Data, Odd	verfügbar

Rec.Flag = NEIN es werden Zeichen unterdrückt, die von der Hardware als Fehlerhaft erkannt wurden.
= JA gibt eventuell fehlerhaft empfangene Zeichen weiter in den Empfangspuffer.

'etc.' steht für die Angaben des zweiten seriellen Kanals. Die Angaben wiederholen sich in der gleichen Form wie für den ersten Kanal.

Im gepufferten Betrieb werden Daten, die gesendet werden sollen, zunächst in den Ausgangspuffer des Device-Treibers geschrieben. Empfangene Daten gelangen ebenfalls zunächst im Empfangspuffer des Treibers. Die Puffer sind über die Instruktionen PUT, PRINT, PRINT_USING, GET, INPUT und INPUT_LINE zugänglich.

Im DACC-Betrieb (DACC = Diect ACCess) werden die Daten, die gesendet werden sollen, direkt in einem Tiger-BASIC®-String gespreichert. Da Strings unmittelbar zugänglich sind, erhöht dies die Effektivität und spart CPU-Zeit. Die Vorteile des DACC-Betriebs wirken sich besonders aus bei hohen Baudraten und hohem Datendurchsatz.

Der DACC.Modus unterstützt Blocktransfers über einen seriellen Kanal. Die Blockgröße kann während des Kommunikationsprozesses dynamisch gesteuert werden.

Der DACC-String muß immer existieren! Nicht erlaubt sind also Variable, die nur vorübergehend leben, wie lokale Strings (in Unterprogrammen) oder temporäre Strings (Expressions). **Richtig: globale oder Task-lokale Strings**.

!

2

Im DACC-Modus wird die Instruktion PUT sowohl zum Senden als auch zum Empfangen von Daten verwendet. Die Instruktionen PRINT, PRINT_USING, INPUT und INPUT_LINE können im DACC-Modus nicht verwendet werden. Die Instruktion GET dient nur zu Statusabfragen, nicht zum Lesen von Daten.

!

Im DACC-Modus werden Referenzen auf statische Variable übergeben, es werden keine Daten bewegt. Um einen flüssigen Datenstrom zu ermöglichen, werden sowohl beim Senden als auch beim Empfangen Reload-Strings an den Treiber übergeben, so daß dieser den String wechseln kann, sobald der erste String voll ist, bzw

Der Datenverkehr in SER4 wird über folgende Sekundär-Adressen abgewickelt:

Sek.-Adr.	Symbol (definiert in UFUNCn.INC)	Beschreibung
		Kanal 0
0	-	gepufferte Ein-/Ausgabe, Kanal 0
10H	DACC0_TX	Referenz auf Sende-String übergeben
12H	DACC0_TRLD	Referenz auf Sende-Reload-String übergeben
14H	DACC0_RX	Referenz auf Empfangs-String übergeben
16H	DACC0_RRLD	Referenz auf Empfangs-Reload-String übergeben
		Kanal 1
1	-	gepufferte Ein-/Ausgabe, Kanal 1
11H	DACC1_TX	Referenz auf Sende-String übergeben
13H	DACC1_TRLD	Referenz auf Sende-Reload-String übergeben
15H	DACC1_RX	Referenz auf Empfangs-String übergeben
17H	DACC1_RRLD	Referenz auf Empfangs-Reload-String übergeben

Wenn ein serieller Kanal im DACC-Modus installiert ist, dann ist er nur in diesem Modus erreichbar. Wenn zum Beispiel Kanal 0 im DACC-Modus installiert wurde, dann steht für den Datenverkehr die Sekundär-Adresse 0 nicht zur Verfügung. Jedoch werden über die Sekundär-Adresse 0 Statusinformationen abgefragt. Gleichzeitig ist in diesem Fall Kanal 1 automatisch im Modus ‚gepuffert', da nur ein Kanal gleichzeitig im DACC-Modus sein kann.

Kontrolle des Datenflusses im DACC-Modus

Im DACC-Modus greift der Device-Treiber auf Daten im String zu, während der String gleichzeitig auch vom weiterlaufenden BASIC-Programm zugänglich ist. Es muß daher verhindert werden, daß auf ein String vom BASIC-Programm zugegriffen wird, während gerade ein Sende- oder Empfangsvorgang den String verwendet.

Beim Senden wird die Stringlänge und der Inhalt des Strings nie durch den Device-Treiber verändert.

Beim Empfangen wird der String in verschiedener Weise modifiziert: Daten werden an die vorgegebenen Stellen im String geschrieben. Die Stringlänge kann sich je nach Einstellung verändern oder auch gleich bleiben.

Der Datenfluß kann durch Lesen des Status des Device-Treibers beobachtet und kontrolliert werden. Durch Lesen des Status auf verschiedenen Sekundär-Adressen werden Informationen über den Zustand der Sende-, Empfangs sowie der Reload-Strings erfragt.

User-Function-Codes für Abfragen (Instruktion GET):

Nr.	Symbol Prefix UFCI_	Sek.-Adr.	Beschreibung
147	UFCI_SER_STAT		Status
		0	4 Bytes geben Auskunft über den Sendestring low WORD: Anzahl noch ungesendeter Bytes im Sendestring high WORD: Sende-Reload-Stringlänge
		1	4 Bytes geben Auskunft über den Empfangsstring low WORD: Anzahl empfangener Bytes im Empfangsstring high WORD: Empfangs-Reload-Stringlänge
		2	8 Bytes geben Auskunft über den Sendestring und den Empfangsstring lowest WORD(1): Anzahl noch ungesendeter Bytes im Sendestring WORD(1): Sende-Reload-Stringlänge WORD(2): Anzahl empfangener Bytes im Empfangsstring high WORD(3): Empfangs-Reload-Stringlänge

Die folgenden Zeilen zeigen, wie der 8-Byte lange Status aufgesplittet wird:

```
GET #7, #2, #UFCI_SER4_STAT, 8, stat$  ' read DACC status (8 Byte)
PRINT #1, "    TxD=";NFROMS (stat$,0,2)' Transmit-Buffer status
PRINT #1, "Rel-TxD=";NFROMS (stat$,2,2)' Transmit-RELOAD buffer status
PRINT #1, "    RxD=";NFROMS (stat$,4,2)' Receive-Buffer status
PRINT #1, "Rel-RxD=";NFROMS (stat$,6,2)' Receive-RELOAD-Buffer status
```


User-Function-Codes für die Instruktion PUT folgendes Kommando:

Nr	Symbol	Beschreibung
1	UFCO_IBU_ERASE	Eingangspuffer löschen
33	UFCO_OBU_ERASE	Ausgangspuffer löschen
135	UFCI_SER_GROF	Growth-Flag: 2: Stringgröße ändert sich nicht. 3: Stringgröße wird beim Start des Empfangs durch die Parameter ‚Offset' und ‚Länge' gesetzt. String wird dabei nie kürzer. 4: Stringgröße wird beim Start des Empfangs durch die Parameter ‚Offset' und ‚Länge' auf genau die benötigte Länge gesetzt. String wird dabei länger oder kürzer.

Datenausgabe im DACC-Modus

Zur Datenausgabe im DACC-Modus werden die Daten in einen String gespeichert und dem Device-Treiber eine Referenz auf diesen String mit PUT übergeben. Wenn nicht anders angegeben, wird der String insgesamt ausgegeben.

Sobald der String an den Treiber übergeben wurde, beginnt der serielle Sendevorgang, der bei vorhandenen Handshake-Leitungen auch durch diese unterbrochen werden kann.

1) PUT #7, #10H, X$ ' pass X$ to be output in full length

2) PUT #7, #10H, X$, 17 ' pass X$ for output, starting at pos 17 to the end

3) PUT #7, #10H, X$, 8, 32 ' pass X$ to output 32 bytes beginning at pos 8

Das folgende Beispielprogramm gibt Daten im DACC-Modus aus, daß heißt statt in einen Puffer zu drucken, wird ein String von 5kByte an den Device-Treiber übergeben. Die Anzahl der noch auszugebenden Bytes wird angezeigt. Da das Handshake beachtet wird, muß CTS Bereit schaft signalisieren, damit ausgegeben werden kann. Verbinden Sie zum Beispiel ein Terminal an SER0.

Programmbeispiel:

```
'------------------------------------------------------------------------
' Name: SER4_TX.TIG
' sendet serielle Daten im DACC Modus
'------------------------------------------------------------------------
user_var_strict                  ' Vars deklarieren!
#include DEFINE_A.INC            ' allgemeine Definitionen
#include UFUNC3.INC              ' User-Function-Codes

STRING tx$(5k)                   ' global string

TASK Main
  BYTE ever                      ' fuer Endlosschleife
  WORD fo                        ' Anzahl noch zu sendender Bytes
' LCD-Treiber installieren (BASIC-Tiger)
  install_device #LCD, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  install_device #LCD, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  install_device #SER, "SER4_K1.TDD", &
    BD_19_200, DP_8N, DACC, &    ' Einstellung Kanal 0
    BD_19_200, DP_8N, YES        ' Einstellung Kanal 1

  tx$ = "this has been sent using SER4 DACC mode<13><10>"
  put #SER, #DACC0_TX, tx$       ' sende kurzen String

  tx$ = fill$ ( "x", 5k )        ' bilde einen langen String
  put #SER, #DACC0_TX, tx$       ' sende langen String

  for ever = 0 to 0 step 0       ' Endlosschleife
                                 ' zeige, wieviel Bytes noch
    get #SER, #0, #UFCI_SER4_STAT, 4, fo ' gesendet werden muessen
    fo = fo bitand 0FFFFh
    print #LCD, "<1Bh>A<0><0><0F0h>to be sent:";fo;"     ";
  next
END
```

Datenausgabe mit Reload im DACC-Modus

Datenblöcke können ohne Unterbrechung ausgegeben werden, wenn neben dem gerade aktiven Sendestring ein Reload-String an den Treiber übergeben wird. Sobald der aktive String fertig ausgegeben ist, wird der Reload-String an die Stelle übernommen und die Ausgabe fortgesetzt. Nachdem der Reload-String übernommen wurde, kann ein neuer Reload-String übergeben werden. Wann dieser Zeitpunkt erreicht ist, wird über Statusabfrage bestimmt.

1) PUT #7, #12H, A$ ' pass A$ as a reload value: full length

2) PUT #7, #12H, A$, 17 ' same as above, starting at pos 17 to the end

3) PUT #7, #12H, A$, 8, 32 ' same as above, 32 bytes starting at pos 8

Programmbeispiel:

```
'------------------------------------------------------------------------
' Name: SER4_TX_RELOAD.TIG
' sendet serielle Daten im DACC Modus
' benutzt Reload-Puffer, um eine kontinuierliche Ausgabe zu erhalten
'------------------------------------------------------------------------
user_var_strict                   ' Vars deklarieren!
#include DEFINE_A.INC             ' allgemeine Definitionen
#include UFUNC3.INC               ' User-Function-Codes

ARRAY tx$(2) OF STRINGS(1k)       ' globale Strings

TASK Main
  BYTE ever                       ' fuer Endlosschleife
  BYTE j                          ' Array-Index
  LONG fo                         ' Anzahl noch zu sendender Bytes
  LONG rest, rld                  ' im Reloadpuffer
' LCD-Treiber installieren (BASIC-Tiger)
  install_device #LCD, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  install_device #LCD, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  install_device #SER, "SER4_K1.TDD", &
    BD_19_200, DP_8N, DACC, &     ' Einstellung Kanal 0
    BD_19_200, DP_8N, YES         ' Einstellung Kanal 1

  tx$(0) = fill$ ( "a", 1k )      ' 1kByte mit 'a'
  tx$(1) = fill$ ( "b", 1k )      ' 1kByte mit 'b'
  put #SER, #DACC0_TX, tx$(0)     ' beginne zu senden

  j = 0
  for ever = 0 to 0 step 0        ' Endlosschleife
                                  ' zeige, wieviel Bytes noch
    get #SER, #0, #UFCI_SER4_STAT, 4, fo ' gesendet werden muessen
    rest = fo bitand 0FFFFh
    print #LCD, "<1Bh>A<0><0><0F0h>to be sent:";rest;"     ";
    rld = fo shr 16               ' und wieviele im Reload sind
    print #LCD, "<1Bh>A<0><1><0F0h> in reload:";rld;"     ";
    if rld = 0 then               ' wenn Reload leer
      put #SER, #DACC0_TRLD, tx$(j) ' dann nachladen
    endif
    if j = 0 then                 ' toggle array index
      j = 1
    else
      j = 0
    endif
  next
END
```

Datenempfang im DACC-Modus

Zum Datenempfang im DACC-Modus wird eine Referenz auf einen String an den Device-Treiber mit PUT übergeben. Die empfangenen Daten werden in dem String

gespeichert. Während des Empfangs ist der String von BASIC aus zugänglich. Wenn nicht anders angegeben, wird der String ab Adresse 0 bis zur maximalen Stringlänge gefüllt.

1) PUT #7, #14H, X$ ' pass X$ to input data until string is full

2) PUT #7, #14H, X$, 17 ' pass X$ to input start at pos 17, until string full

3) PUT #7, #14H, X$, 8, 32 ' pass X$ to input 32 Bytes, start at pos 8

Das folgende Beispielprogramm zeigt die Anzahl der empfangenen Zeichen an, die Länge des Empfangsstrings (in diesem Beispiel Konstant) sowie den Empfangsstring selbst. Es wird davon ausgegangen, daß nur druckbare ASCII-Zeichen empfangen werden. Auf dem Plug & Play Lab kann der Empangsvorgang mitverfolgt werden. Schließen Sie dazu ein Terminal an SER0 an und geben über die Terminaltastatur Zeichen ein. Die Anzahl der empfangenen Zeichen wächst von 0 bis 10 und der mit „xxxxxxxxxx“ vorbesetzte String enthält die empfangenen Zeichen.

Programmbeispiel:

```
'-------------------------------------------------------------------------
' Name: SER4_RX.TIG
' empfaengt serielle Daten im DACC Modus (ohne Reload)
'-------------------------------------------------------------------------
user_var_strict                   ' Vars deklarieren!
#include DEFINE_A.INC             ' allgemeine Definitionen
#include UFUNC3.INC               ' User-Function-Codes


STRING rx$(10)                    ' globaler String

TASK Main
  BYTE ever                       ' fuer Endlosschleife
  WORD ibu_fill                   ' Eingangspufferfuellung
' LCD-Treiber installieren (BASIC-Tiger)
  install_device #LCD, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  install_device #LCD, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  install_device #SER, "SER4_K1.TDD", &
    BD_19_200, DP_8N, DACC, &     ' Einstellung Kanal 0
    BD_19_200, DP_8N, YES         ' Einstellung Kanal 1

  rx$ = fill$ ( "x", 10 )
'  put #SER, #0, #UFCO_SER_GROF, 2 ' Empfangsstringlaenge konstant

  put #SER, #DACC0_RX, rx$        ' uebergebe Empfangsstring

  for ever = 0 to 0 step 0        ' Endlosschleife
                                  ' zeige, wieviel Bytes
    get #SER, #1, #UFCI_SER4_STAT, 4, fi ' empfangen wurden
    fi = len(rx$) - (fi bitand 0FFFFh)
    print #LCD, "<1Bh>A<0><0><0F0h>received:";fi;"     ";
    print #LCD, "<1Bh>A<0><1><0F0h>len(rx$):";len(rx$);"     ";
    print #LCD, "<1Bh>A<0><2><0F0h>     rx$:";rx$;"     ";
  next
END
```

Datenempfang mit Reload im DACC-Modus

Datenblöcke können ohne Unterbrechung empfangen werden, wenn neben dem gerade aktiven Empfangs-String ein Reload-String an den Treiber übergeben wird. Sobald der aktive String vollständig empfangen ist, wird der Reload-String an die Stelle übernommen und der Empfang fortgesetzt. Nachdem der Reload-String übernommen wurde, kann ein neuer Reload-String übergeben werden. Wann dieser Zeitpunkt erreicht ist, wird über Statusabfrage bestimmt.


```
1) PUT #7, #16H, A$        ' pass A$ as a reload value: full length

2) PUT #7, #16H, A$, 17    ' same as above, starting at pos 17 to the end

3) PUT #7, #16H, A$, 8, 32 ' same as above, 32 bytes starting at pos 8
```

Das folgende Beispielprogramm zeigt die Anzahl der empfangenen Zeichen an, wieviel Zeichen im Reloadpuffer sind sowie den Inhalt der beiden Strings, hier in Form von 2 Array-Elementen. Die empfangenen Daten werden jedesmal, wenn der Reloadpuffer neue Daten erhält, an den String ‚collect$' angehängt, um den String für den nächsten Reload-Vorgang freizugeben. Die Länge sowie der Inhalt des Empfangsstrings kann mit Hilfe des Befehls ‚Überwachte Ausdrücke' in dem Menü ‚Ansicht' betrachtet werden. Es wird davon ausgegangen, daß nur druckbare ASCII-Zeichen empfangen werden. Auf dem Plug & Play Lab kann der Empangsvorgang und die Übergabe in den Reload-String mitverfolgt werden. Schließen Sie dazu ein Terminal an SER0 an und geben über die Terminaltastatur Zeichen ein. Die Anzahl der empfangenen Zeichen wächst von 0 bis 10 und der mit „0000000000" vorbesetzte String enthält die empfangenen Zeichen. Wenn der Puffer voll ist, dann wird der Empfang im Reloadpuffer fortgesetzt. Kurzzeitig ist die Länge 10 in der Anzeige der Reloadpufferlänge zu sehen.

Programmbeispiel:

```
'-------------------------------------------------------------------------
' Name: SER4_RX_RELOAD.TIG
' empfaengt serielle Daten auf Kanal SER0 im DACC Modus
' benutzt Reload-Puffer und sammelt Daten in collect$ (hier: max 1k)
'-------------------------------------------------------------------------
user_var_strict                    ' Vars deklarieren!
#include DEFINE_A.INC              ' allgemeine Definitionen
#include UFUNC3.INC                ' User-Function-Codes

ARRAY  rx$(2) OF STRINGS(10)       ' Array mit 2 Strings a 10 Bytes
STRING collect$(1k)                ' sammelt empfangene daten ein

TASK Main
  BYTE ever                        ' fuer Endlosschleife
  BYTE j                           ' Array-Index
  BYTE r
  LONG fi                          ' Anzahl empfangene Bytes
  LONG rfill, rld                  ' im Reloadpuffer
  STRING c$(2)
' LCD-Treiber installieren (BASIC-Tiger)
  install_device #LCD, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  install_device #LCD, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  install_device #SER, "SER4_K1.TDD", &
    BD_19_200, DP_8N, DACC, &      ' Einstellung Kanal 0
    BD_19_200, DP_8N, YES          ' Einstellung Kanal 1

  rx$(0) = fill$ ( "0", 10 )       ' 10 Bytes mit '0'
  rx$(1) = fill$ ( "1", 10 )       ' 10 Bytes mit '1'
  put #SER, #DACC0_RX, rx$(0)      ' uebergebe Empfangsstring
  put #SER, #DACC0_RRLD, rx$(1)    ' uebergebe Empfangs-Reload-String
  j = 0
  collect$ = ""                    ' beginne mit leerem collect$
  for ever = 0 to 0 step 0         ' Endlosschleife
    get #SER, #1, #UFCI_SER4_STAT, 4, fi ' empfangen wurden
    rfill = len(rx$(0)) - (fi bitand 0FFFFh)' zeige, wieviel Bytes
    print #LCD, "<1Bh>A<0><0><0F0h>received:";rfill;"     ";
    rld = len(rx$(0)) - (fi shr 16)
    print #LCD, "<1Bh>A<0><1><0F0h>  reload:";rld;" ";
    print #LCD, "<1Bh>A<0><2><0F0h>  rx$(0):";rx$(0);" ";
    print #LCD, "<1Bh>A<0><3><0F0h>  rx$(1):";rx$(1);" ";
    if rld > 0 then                ' Hinweis, daß 1 String voll ist
      wait_duration 300            ' um Reloadwert zu sehen
      collect$ = collect$ + rx$(j)' sammle empfangene Daten
      c$ = chr$ ( j+30h )          ' 0 oder 1
      rx$(j) = fill$ ( c$, 10 )    ' bereite String f. Display vor
      put #SER, #DACC0_RRLD, rx$(j)' uebergebe Empfangs-Reload-String
      if j = 0 then                ' Array-Index umschalten
        j = 1
      else
        j = 0
      endif
    endif
  next
END
```

Besonderheiten des SER4 Device-Treibers

Folgende Einschränkungen sind abhängig vom Betriebsmodus des Device-Treibers SER4 zu beachten:

Nr.	Stichwort	Anmerkung
1	Empfangsfehler	• Zeichen werden im DACC-Modus immer übernommen, Unterdrückung nicht vorgesehen.
2	Intern gepufferter Betrieb	• Baudraten bis 153600 baud sind verfügbar • alle Einstellungsparameter sind möglich
3	Betrieb mit Strings (DACC)	• nur auf einem Kanal zur gleichen Zeit möglich, der andere Kanal steht währenddessen im gepufferten Betrieb zur Verfügung. • Alle Baudraten stehen zur Verfügung • nur 8 Datenbits (nicht 7 oder 9) • auf DACC-Betrieb angepasste Steuerfunktionen stehen zur Verfügung
4	Hohe Baudraten	• Module ohne eingebaute Treiberchips • externe Treiberchips müssen für die Baudrate geeignet sein

Leere Seite

CAN

Der Device-Treiber ‘CAN1_xx.TDD’ unterstützt die interne CAN-Schnittstelle des BASIC-Tiger®-CAN-Moduls.

In diesem Abschnitt finden Sie:

Beschreibung des Device-Treibers CAN1_xx.TDD

Dieser Device-Treiber ermöglicht in Zusammenhang mit dem BASIC-Tiger®-CAN-Modul die Ein- und Ausgabe auf dem CAN-Bus. Bei der Installation des Treibers können die Parameter der CAN-Schnittstelle festgelegt werden. Einige Parameter sind durch Kommandos an den Treiber auch zur Laufzeit veränderbar.

Dateinamen: CAN1_K8.TDD (mit 8K Puffern)
CAN1_K1.TDD (mit 1K Puffern)
CAN1_R1.TDD (mit 256 Byte Puffern)

INSTALL DEVICE #*D*, "CAN1_xx.TDD", "*Code, Mask, Bt0, Bt1, Mod, Oc*"

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Code ist ein Parameter zur Festlegung des Access-Codes. ‚Code’ ist immer 4 Bytes lang. Der Wertebereich für den Access-Code ist bei Standard-Frames 0...7FFh
und bei extended Frames 0...1FFF FFFF.
Standardwert: 0

Mask ist ein Parameter zur Festlegung des Akzeptanzfilters. ‚Mask' ist immer 4 Bytes lang.
Standardwert: 0FFFFFFFFh

Bt0 (Bustiming-Register-0) ist ein Parameter zur Festlegung des Baudraten-Prescalers sowie der Synchronisations-Sprungweite (1 Byte). Zusammen mit Bt1 wird dadurch die Übertragungsgeschwindigkeit festgelegt.
Standardwert: 0

Bt1 (Bustiming-Register-1) ist ein Parameter zur Festlegung des Bus-Timings und der Anzahl der Samples beim Empfang (1 Byte). Zusammen mit Bt0 wird dadurch die Übertragungsgeschwindigkeit festgelegt.
Standardwert: 2Fh (Tseg1=15, Tseg2=2)

Mod ist ein Parameter zur Festlegung des Modus (1 Byte).
Standardwert: 0

Bit	Symbol	wenn Bit gesetzt ('1')
2	CAN_LISTEN	Listen-Only-Mode
3	CAN_SELFTEST	Selftest-Mode
4	CAN_ACC_SINGLE	Single Acceptance-Filter-Mode (32-Bit-Filter)
5	CAN_SLEEP	Sleep-Mode
1,6,7		reserviert

Wird der Listen-Only-Mode installiert, dann versucht der Treiber, die Bitrate auf dem Bus anhand einer Tabelle mit vorgegebenen Bitraten automatisch zu erkennen.

Outctrl ist ein Parameter zur Festlegung der Ausgangsstufe der CAN-Hardware. Standardwert ist 1Ah zum Anschluß eines externen Treiberbausteins.

Beispiele für eine Installation für 500 kBit:

```
install_device #CAN, "CAN1_K1.TDD", &
0,0,0,0, &                      ' access code
0ffh,0ffh,0ffh,0ffh, &          ' access mask
0,2Fh, &                        ' bustim1, bustim2
0,1Ah                           ' mode, outctrl
```

CAN-Botschaften in den I/O-Puffern des Treibers

In den Eingangs und Ausgangspuffern des Tiger-BASIC®-CAN-Device-Treibers befinden sich immer vollständige CAN-Botschaften und keine weiteren Bytes. Eine CAN-Botschaft beginnt mit dem Frame-Info-Byte, welches bestimmt, ob es sich um eine Botzschaft mit 11-oder 29-Bit-Identifier handelt und wieviele Datenbytes enthalten sind. Außerdem befindet sich das RTR-Bit im Frame-Info-Byte. Je nach Frametyp folgen 3 Identifier-Bytes (Standardframe) oder 5 Identifier-Bytes (extended Frame) und anschliessend die Datenbytes. Eine CAN-Botschaft kann 0...8 Bytes als Nutzdaten übertragen.

Im Frame-Info-Byte befindet sich die Information über

- den Frametyp (11 oder 29 ID-Bits)
- die Anzahl der Datenbytes (0...8)
- ob es sich um ein Remote-Transmit-Request handelt

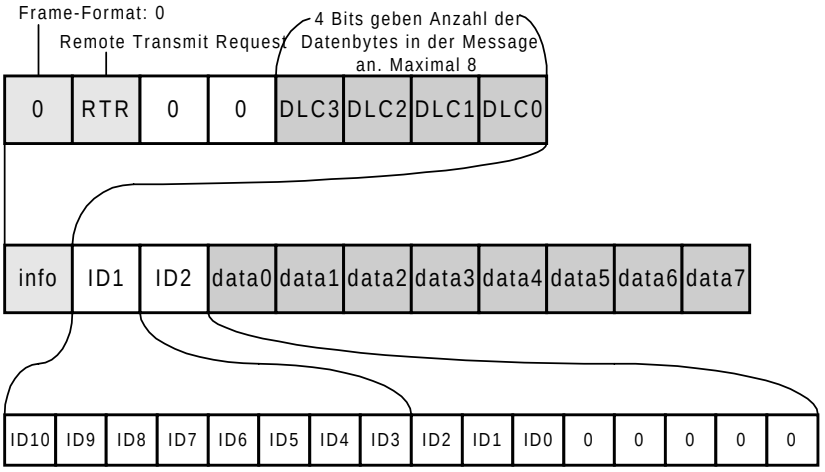
Der Identifier kann

- 29 Bits lang sein und belegt dann 4 Bytes im Puffer
- 11 Bits lang sein und belegt dann 2 Bytes im Puffer

Ein Standardframe belegt maximal 11 Bytes, ein extended Frame maximal 13 Bytes im Puffer. Wenn der Device-Treiber beim Empfang nicht mindestens noch 13 Bytes im Puffer frei hat, dann wird die Botschaft verworfen und ein Fehler ‚Pufferüberlauf' registriert. Je nach Länge der einzelnen empfangenen CAN-Nachrichten passen zwischen 341 Nachrichten (nur Standardframes ohne Daten) und 78 Nachrichten (nur extended Frames, alle mit 8 Datenbytes) in einen 1kByte großen Puffer.

Standardframe

Die Abbildung zeigt den Aufbau des Standardframes mit vergrößertem Frame-Info-Byte (oben) und den ID-Bytes (vergrößert unten). Die Länge der Botschaft wird vom Device-Treiber automatisch eingesetzt. Die 11 ID-Bits müssen sich mit dem höchstwertigen Bit zuerst linksbündig in den beiden Bytes befinden, wie in der Abbildung dargestellt.



Aufbau des ‚Standard Frame'

Standard Frame, Info-Bits:

FF	Frame-Format-Bit, hier FF=0. 0: Standard Frame 1: extended Frame
RTR	Remote Transmit Request, Sendeaufforderung. Botschaften mit gesetztem RTR-Bit werden vom Treiber direkt bearbeitet und erscheinen nicht im Puffer.
DLC	4 Bit geben die Anzahl der Datenbytes in der Message an (0...8). Diese Bits setzt der Device-Treiber.

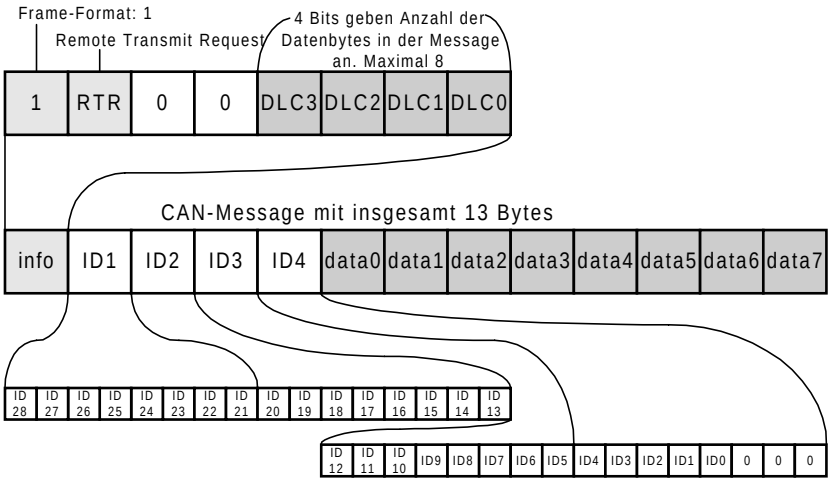
In beiden ID-Bytes zusammen befindet sich der um 5 Bits nach links verschobene 11-Bit-Identifier der CAN-Botschaft. Das Format ist hier ‚high-Byte first', im Gegensatz zu WORD-Variablen in Tiger-BASIC®, die ‚low-Byte first' sind.

Hinter den ID-Bytes folgen soviele Datenbytes, wie durch DLC angegeben.

Beispiel für das Erzeugen von Standardframes in Tiger-BASIC®:

```
t_id = 7FFh shl 5                   ' Transmit-ID, linksbuendig in WORD
' Standardframe mit Frame-Info-Byte, 2 leeren ID-Bytes, Daten
msg$ = "<0><0><0>" + data$
msg$ = ntos$ ( msg$, 1, -2, t_id ) ' ID einbauen mit high-Byte zuerst
                                    ' Laenge wird vom Treiber eingesetzt
print #CAN, msg$;                   ' PRINT, mit Semikolon!!
' oder
put #CAN, msg$
```

Extended Frame



Aufbau des ‚extended Frame'

Extended Frame, Info-Bits:

FF	Frame-Format-Bit, hier FF=1. 0: Standard Frame 1: extended Frame
RTR	Remote Transmit Request, Sendeaufforderung. Messages mit gesetztem RTR-Bit werden vom Treiber direkt bearbeitet und erscheinen nicht im Puffer.
DLC	4 Bit geben die Anzahl der Datenbytes in der Message an (0...8).

In den 4 ID-Bytes befindet sich der um 3 Bits nach links verschobene 29-Bit-Identifier der CAN-Botschaft. Das Format ist hier ‚high-Byte first', im Gegensatz zu LONG-Variablen, die ‚low-Byte first' sind.

Hinter den ID-Bytes folgen soviele Datenbytes, wie durch DLC angegeben.

Beispiel für das Erzeugen von extended Frames in Tiger-BASIC®:

```
t_id = 1FFFFFFFh shl 3              ' Transmit-ID, linksbuendig in LONG
' extended Frame mit Frame-Info-Byte, 4 leeren ID-Bytes, Daten
msg$ = "<80h><0><0><0><0>" + data$
msg$ = ntos$ ( msg$, 1, -4, t_id ) ' ID einbauen mit high-Byte zuerst
                                    ' Laenge wird vom Treiber eingesetzt
print #CAN, msg$;                   ' PRINT mit Semikolon!!
' oder
put #CAN, msg$
```

CAN User-Function-Codes

User-Function-Codes für Abfragen (Instruktion GET):

Nr	Symbol Prefix UFCI_	Beschreibung
1	UFCI_IBU_FILL	Füllstand des Eingangspuffers (Byte)
2	UFCI_IBU_FREE	freier Platz im Eingangspuffer (Byte)
3	UFCI_IBU_VOL	Größe des Eingangspuffers (Byte)
33	UFCI_OBU_FILL	Füllstand des Ausgangspuffers (Byte)
34	UFCI_OBU_FREE	freier Platz im Ausgangspuffer (Byte)
35	UFCI_OBU_VOL	Größe des Ausgangspuffers (Byte)
65	UFCI_LAST_ERRC	letzter Error-Code
152	UFCI_CAN_MODE	liest CAN register MODE
153	UFCI_CAN_STAT	liest CAN register STAT
154	UFCI_CAN_ALC	liest Kopie vom ‘arbitration lost register’
155	UFCI_CAN_ECC	liest Kopie vom ‘error code capture register’
156	UFCI_CAN_EWL	liest Kopie vom ‘error warning limit register’
157	UFCI_CAN_RXERR	liest Kopie vom ‘rx error counter register’
158	UFCI_CAN_TXERR	liest Kopie vom ‘tx error counter register’
159	UFCI_CAN_RMC	liest Kopie vom ‘rx message counter’
99	UFCI_DEV_VERS	Version des Treibers

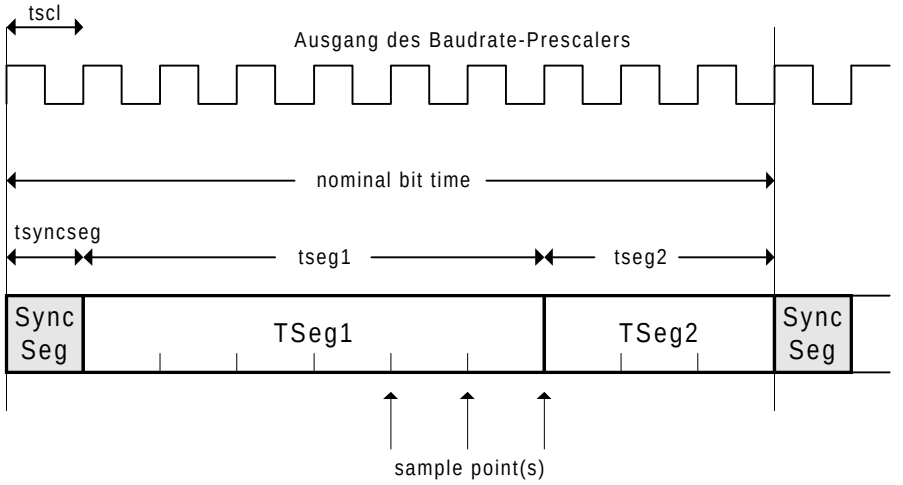
User-Function-Codes für Output (Instruktion PUT):

Nr	Symbol Prefix: UFCO_	Beschreibung
1	UFCO_IBU_ERASE	Eingangspuffer löschen
33	UFCO_OBU_ERASE	Ausgangspuffer löschen
65	UFCO_ERRC_RESET	setze letzten OK-/WARNING-/ERROR-Code zurück
136	UFCO_CAN_CODE	setzt CAN-Register CODE
137	UFCO_CAN_MASK	setzt CAN-Register MASK
138	UFCO_CAN_MODE	setzt CAN-Register MODE
139	UFCO_CAN_BUSTIM0	setzt CAN-Register BUSTIM0
140	UFCO_CAN_BUSTIM1	setzt CAN-Register BUSTIM1
141	UFCO_CAN_OUTCTRL	setzt CAN-Register OUTCTRL
176	UFCO_CAN_RESET	führt ein CAN-Soft-Reset durch

Bus-Timing und Übertragungsrate

Die Übertragungsrate wird durch die Länge eines Bits bestimmt. Ein Bit setzt sich aus drei Abschnitten zusammen, die wiederum aus einzelnen Zeitsegmenten bestehen:

- Sync-Segment, immer ein Zeitsegment lang.
- TSEG1 ist zwischen 5 und 15 Zeitsegmente lang. Innerhalb Tseg1 wird das Bit beim Empfang gesampelt.
- TSEG2 ist zwischen 2 und 7 Zeitsegmente lang.



Aufbau eines Bits

Die Einheit eines Zeitsegmentes wird im Bustiming-Register 0 festgelegt, die Anzahl Zeitsegmente, aus denen TSEG1 und TSEG2 bestehen im Bustiming-Register 1.

Bustiming-Register 0

Die Länge eines Zeitsegmentes ‚tscl' wird im **Bustiming-Register-0** durch den Baudrate-Prescaler **BRP** festgelegt. Der 6-Bit-Prescaler kann Werte zwischen 0 und 31 annehmen.

1 Zeitsegment: t_{scl} = 0,1 * (BRP+1) µsec

1 Bitzeit = Tsync + Tseg1 + Tseg2

Die oberen Bits in diesem Register legen die Synchronisations-Sprungweite fest. Der Wert **SJW** bestimmt die maximale Anzahl von Clockzyklen, um die ein Bit verkürzt oder verlängert werden darf, um Phasenunterschiede zwischen verschiedenen Buscontrollern durch Neusynchronisation auszugleichen.

Bustiming-Register 0							
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0

Bustiming-Register 1

Im **Bustiming-Register-1** wird festgelegt, aus wievielen Zeitsegmenten **Tseg1** und **Tseg2** bestehen und wie oft das empfangene Bit gesampelt wird (einmal oder dreimal).

Bustiming-Register 1							
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
SAM	TSEG2.2	TSEG2.1	TSEG2.0	TSEG1.3	TSEG1.2	TSEG1.1	TSEG1.0

SAM=1: Der Bus wird dreimal gesampelt. Empfohlen für langsame und mittelschnelle Busse, wenn Filterung von Spikes auf dem Bus Vorteile bringt.

SAM=0: Der Bus wird einmal gesampelt. Empfohlen für schnelle Busse.

Welche Werte für Tseg1 und Tseg2 Empfangssicherheit gewähren, hängt von den physikalischen Eigenschaften des Übertragungsmediums ab, inklusive Treiberbausteine, Optokoppler. Aus diesen Eigenschaften ergibt sich letztlich auch die erzielbare Baudrate und Leitungslänge.

Einige gängige Einstellungen finden Sie in folgender Tabelle (erreichbare Buslängen sind Anhaltspunkte):

Bitrate	Bustim0	Bustim1	Bt1 Tseg1	Bt1 Tseg2	Bus-länge
1 Mbit	0	43h	3	4	25m
500 kBit	0	5Ch	12	5	100m
250 kBit	1	5Ch	12	5	250m
125 kBit	3	5Ch	12	5	500m
100 kBit	4	5Ch	12	5	650m

Bei der Installation des Treibers kann durch Parameter die Bitrate festgelegt werden.

Während der Laufzeit können die Einstellungen des Bustimings mit Hilfe von User-Function-Codes verändert werden.
Anmerkung: der Ausgangspuffer sollte während des Setzens von Bustim0 oder Bustim1 leer sein, da sich der interen CAN-Chip vorübergehend im Resetmodus befindet. Er ist auch vorübergehend nicht empfangsbereit.

Beispiel: stelle 100kBit nach obiger Tabelle während der Laufzeit ein:

```
PUT #CAN, #0, #UFCO_CAN_BUSTIM0, 4
PUT #CAN, #0, #UFCO_CAN_BUSTIM1, 5CH
```

Error-Register

Sowohl der korrekte Empfang einer CAN-Botschaft als auch fehlerhafte Zustände auf dem CAN-Bus lösen einen Empfangs-Interrupt aus. In der Interrupt-Bearbeitung stellt der Device-Treiber fest, ob es sich um ein fehlerfrei empfangenes Paket handelt oder ob Fehler aufgetreten sind. In jedem Falle werden die Werte aufgefrischt, die mit Fehlerzuständen zu tun haben und für die nächste Fehlerabfrage mit einem User-Function-Code bereitgestellt. Treten vor der Fehlerabfrage weitere Fehler auf, dann wird der jeweils neueste Fehlercode abgelegt.

Folgende Fehlerabfragen sind möglich:

User-Function-Code	Bit(s)	Bedeutung
UFCI_CAN_STAT	0	Receive Buffer Status: 0: leer 1: voll
	1	Receive Overrun: 0: nein 1: ja Data-Overrun. Tritt ein, wenn eine neue CAN-Message bereits eintrifft, obwohl noch nicht genügend Platz im Empfangsbereich im CAN-Chip ist. Dies betrifft nicht den Puffer des Device-Treibers.
	2	Transmit Buffer: 0: blockiert 1: frei
	3	Senden: 0: läuft 1: fertig
	4	Empfangen: 0: frei 1: läuft
	5	Senden: 0: frei 1: läuft
	6	Fehler: 0: ok 1: ein Fehlerzähler oder beide (RXERR, TXERR) hat den gesetzten Wert für Error-Warning-Limit überschritten.
	7	Bus-Status: 0: ON 1: OFF Bei OFF beteiligt sich die CAN-Hardware nicht mehr an Aktivitäten auf dem Bus.
UFCI_CAN_ALC	0...4	Arbitration-Lost-Capture. Zeigt an bei welchem Bit incl. RTR-Bit der Buszugriff verloren ging.
	5...7	Reserviert
UFCI_CAN_ECC	0...4	Error-Code-Capture gibt an in welchem Segment der Fehler auftrat. Nähere Beschreibung weiter unten im Text.
	5	Richtung: 0: Tx 1: Rx
	6,7	Errortyp
UFCI_CAN_RXERR	0...7	Rx-Fehlerzähler. Zählt bei Empfangsfehler hoch und bei korrektem Empfang wieder runter bis 0. Siehe auch Error-Warning-Limit
UFCI_CAN_TXERR	0...7	Tx-Fehlerzähler. Zählt bei Sendefehler hoch und bei korrekter Sendung wieder runter bis 0. Siehe auch Error-Warning-Limit

Arbitration-Lost-Fehler

Die Abfrage des ALC-Registers kann näheren Aufschluß darüber geben, an welcher Bitposition der Buszugriff verloren ging. Auf dem CAN-Bus erscheint nach dem Startbit zunächst das höchstwertigste Identifier-Bit. Es folgen im Falle eines Standard-Frames 10 weitere Identifier-Bits. Da die ‚extended Frames' zu den Standard-Frames kompatibel sein müssen, folgt nach diesen 10 Identifier-Bits in jedem Fall ein RTR-Bit. Das nächste Bit entscheidet nun, ob es sich um ein Standard-Frame oder ein ‚extended Frame' handelt. Es heißt IDE-Bit, **Id**entifier **E**xtension. Nach einem reservierten Bit folgen im Falle des ‚extended Frame' die restlichen 18 Identifier-Bits. Das Arbitration-Lost-Register kann bis zum 31. Bit die Arbitrierung mitverfolgen, das ist bis zum RTR-Bit eines ‚extended Frame'.

Da alle Teilnehmer gleichzeitig auf den Bus zugreifen, zeigt das erste rezessive Bit, welches von einem dominanten Bit überschrieben wurde, den verlorenen Buszugriff an. Die Bitposition ist dabei ein Maß für die Priorität des Teilnehmers, der den Buszugriff verhindert.

Beachte: Bei jedem Interrupt wird der gepufferte Wert im Device aufgefrischt. Da das ALC-Register der CAN-Hardware zurückgesetzt wird, wenn es gelesen wird, wird ein einmal aufgetretener und registrierter Arbitration-Lost-Fehler beim nächsten korrekten Empfang überschrieben. Einzelne Arbitration-Lost-Zustände können daher nur erfaßt werden, wenn genügend Zeit besteht, um den Wert vom Treiber auszulesen. Immer wieder auftretende Arbitration-Lost-Zustände werden statistisch erfaßt.

ECC-Fehler-Register

Nachdem ein Bus-Error aufgetreten ist, rettet der Device-Treiber das ECC-Register (Error Code Capture) des CAN-Chips. Die Abfrage des ECC-Registers gibt Aufschluß über den Busfehler:

ECC-Code	Ursache des Busfehler
2	ID.28 bis ID.21
3	‚start of frame'
4	Bit SRTR (Substitute RTR)
5	Bit DIE (Identifier Extended)
6	ID.20 bis ID.18
7	ID.17 bis ID.13
8	CRC-Sequenz
9	reserviertes Bit 0
10	Datenfeld
11	DLC (Data Length Code)
12	Bit RTR
13	reserviertes Bit 1
14	ID.4 bis ID.0
15	ID.12 bis ID.5
17	Active Error Flag
18	Intermission
19	tolerate dominant bits
22	Passive Error Flag
23	Error Delimiter
24	CRC-Delimiter
25	Acknowledge-Slot
26	End of Frame
27	Acknowledge-Delimiter
28	Overload Flag

RXERR-Empfangsfehlerzähler

Bei jedem CAN-Interrupt wird im Device-Treiber der Empfangsfehlerzähler ausgelesen. Der letzte Wert kann mit einem User-Function-Code abgefragt werden. Die Abfrage verändert nicht den Zählerstand.

```
...
get #CAN, #0, #UFCI_CAN_RXERR, 1, rx_err
...
```

Wenn der Zählerstand den gesetzten Error-Warning-Limit (Standard: 96) überschreitet, wird das Bit 6 im Statusregister gesetzt.

Wenn der Zählerstand 127 überschreitet, geht der interne CAN-Chip in den Modus ‚Bus-Error-Passive' über und das Bit 7 im Statusregister wird gesetzt. In diesem Modus versendet die CAN-Hardware keine Fehlertelegramme mehr, sendet und empfängt seine Telegramme aber weiterhin. Fehlerfreie Datentelegramme auf dem Bus reduzieren den Fehlerzähler wieder.

TXERR-Sendefehlerzähler

Bei Fehler-Interrupts wird im Device-Treiber der Sendefehlerzähler ausgelesen. Der letzte Wert kann mit einem User-Function-Code abgefragt werden. Die Abfrage verändert nicht den Zählerstand.

```
...
get #CAN, #0, #UFCI_CAN_TXERR, 1, tx_err
...
```

Wenn der Zählerstand den gesetzten Error-Warning-Limit (Standard: 96) überschreitet, wird das Bit 6 im Statusregister gesetzt.

Wenn der Zählerstand 127 überschreitet, geht der interne CAN-Chip in den Modus ‚Bus-Error-Passive' über und das Bit 7 im Statusregister wird gesetzt. In diesem Modus versendet die CAN-Hardware keine Fehlertelegramme mehr, sendet und empfängt seine Telegramme aber weiterhin. Fehlerfreie Datentelegramme auf dem Bus reduzieren den Fehlerzähler wieder.

Wenn der Zählerstand 255 überschreitet, geht der CAN-Chip in den Buss-Off-Zustand. Dieser Zustand kann nur durch einen Hardware-Reset oder Software-Reset verlassen werden.

Empfangsfilter mit Code und Mask

Der gesetzte Access-Code bestimmt zusammen mit dem Access-Filter, welche CAN-Botschaften empfangen werden. Durch die Access-Maske werden Bits zu ‚don't care' gesetzt, wenn dies erforderlich ist. Die Bits des empfangenen Identifieres, die nicht ‚don't care' sind, müssen mit dem Code übereinstimmen, damit die Botschaft empfangen wird.

Es folgen die Beschreibungen:

- Setzen von Acess-Code und Access-Mask
- Standard-Frame mit Single filter configuration
- Extended Frame mit Single filter configuration
- Standard-Frame mit Dual-Filter-Konfiguration
- Extended Frame mit Dual-Filter-Konfiguration

Die empfangene CAN-Botschaft kann als Standard-Frame oder als Extended-Frame vorliegen. Ausserdem gibt es zwei Filtermodi: ‚Single filter configuration' oder ‚Dual filter configuration'.

Setzen von Acess-Code und Access-Mask

Acess-Code und Access-Mask sind als Register Bestandteil der CAN-Hardware und werden bei der Installation des Device-Treibers gesetzt. Wenn keine Parameter angegeben sind, wird Access-Code auf 0 und Access-Mask auf 0FFFFFFFh gesetzt, so daß alle Nachrichten den Filter passieren.

Man kann den Code und die Maske als einfaches Bitmuster oder als Zahl sehen. Zum Beispiel eignet sich eine LONG-Zahl, um die Bits des Access-Codes oder der Access-Mask zu speichern. Ein Problem ergibt sich dadurch, daß die CAN-Zahl mit dem höchstwertigen Byte beginnt, die Tiger-BASIC®-LONG-Zahl jedoch mit dem niedrigwertigsten:

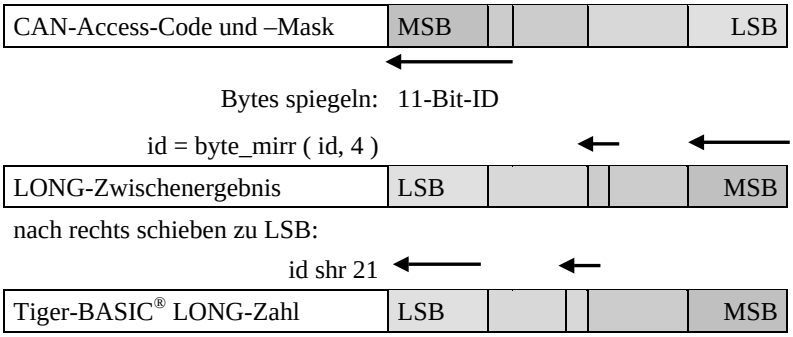
CAN-Access-Code und -Mask	MSB			LSB

Tiger-BASIC® LONG-Zahl	LSB			MSB

Hinzu kommt, daß je nach Frame-Typ die 11 Bits bzw. die 29 Bits linksbündig in den 32 Bit für den Identifier stehen. Zahlen beginnen jedoch rechts mit dem niedrigsten Bit und haben keine ‚don't care'-Bits rechts davon stehen. Links einer Zahl können jedoch Nullen stehen, die keine Rolle spielen.

Will man also den Identifier aus dem Access-Code als Zahl sehen, dann müssen die Bytes erst gespiegelt werden, und

- bei 11-Bit-Identifier der Wert von Access-Code um 21 Bit (5+16) nach rechts geschoben werden.
- bei 29-Bit-Identifier der Wert von Access-Code um 3 Bit nach rechts geschoben werden.



Umgekehrt: hat man eine Zahl vorliegen und will sie in ein CAN-Register Access-Code oder Access-Mask ablegen, dann

- müssen die Bits der Zahl zuerst nach links geschoben werden
- dann die Bytes der Zahl gespiegelt werden

Beachten Sie, daß die Funktion NTOS$ die Speigelung der Bytes vornehmen kann indem als Argument für die Anzahl der Bytes eine negativer Wert angegeben wird:

- **msg$ = ntos$ (msg$, 1, -2, t_id)** baut einen 11-Bit-Identifier, der als WORD-Zahl mit den ID-Bits bereits an der richtigen Stelle vorliegt, in einen String ein und spiegelt dabei die Bytes.
- **msg$ = ntos$ (msg$, 1, -4, t_id)** tut das gleiche für einen 29-Bit-Identifier, der als LONG-Zahl mit den ID-Bits bereits an der richtigen Stelle vorliegt.

In einem String verändert sich die Reihenfolge nicht:

id$ = “<1Fh><AAh><BBh><33h>“

oder

id$ = “1F AA BB 33“%

Steppen Sie das nachfolgende Beispeilprogramm, um in den ‚Überwachten Ausdrücken’ die Gegebenheiten nachzuvollziehen.

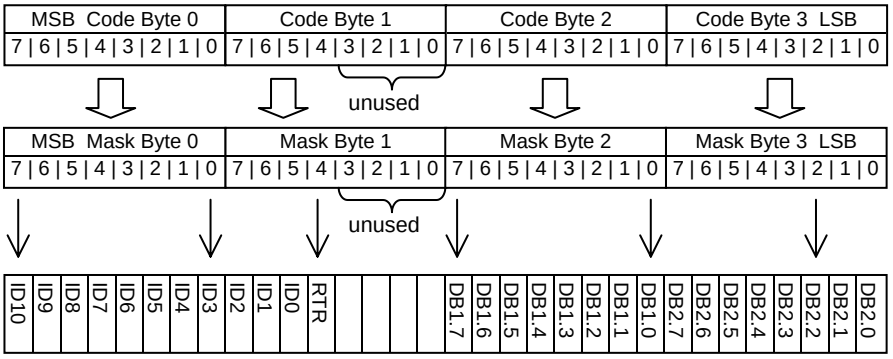
Programmbeispiel:

```
'--------------------------------------------------------------------
' Name:  CAN_SET_FILTER.TIG
' Setzt Filter-Konfiguration
' Demonstriert das Setzen von Access-Code und Access-Mask
' in verschiedenen Varianten
' Nur ein CAN-Tiger notwendig, da nicht gesendet oder empfangen wird
' Benutzen Sie das Kommando 'Ueberwachte Ausdruecke' aus
' dem Menue 'Anzeige'
'--------------------------------------------------------------------
user var strict                    ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC                ' User Function Codes
#INCLUDE DEFINE_A.INC              ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                   ' CAN-Definitionen

LONG ac_code, ac_mask
STRING id$

'--------------------------------------------------------------------
TASK MAIN
  install_device #LCD, "LCD1.TDD" ' LCD-Treiber installieren
  install_device #CAN, "CAN1_K1.TDD", & ' CAN-Treiber installieren
    "12 34 56 78 &                 ' access code
     EF FF FE FF &                 ' access mask
     10 45 &                       ' bustim1, bustim2
     08 1A"%                       ' single filter mode, outctrl

  using "UH<8><8>   0 0 0 4 4"     ' fuer ID Anzeige im gesamten Prg

' zeige Access-Code und Access-Mask wie installiert
  get #CAN, #0, #UFCI_CAN_CODE, 0, ac_code ' und access code lesen
  ac_code = byte_mirr ( ac_code, 4 ) ' Bytefolge gespiegelt fuer LONG
  print_using #LCD, "<1>ac_code:";ac_code
  get #CAN, #0, #UFCI_CAN_MASK, 0, ac_mask ' und access mask lesen
  ac_mask = byte_mirr ( ac_mask, 4 ) ' Bytefolge gespiegelt fuer LONG
  print_using #LCD, "ac_mask:";ac_mask
' dieselben Zeilen in show_codemask
  wait_duration 1000

' siehe Byte-Reihenfolge ('Ueberwachte Ausdruecke' id$ und ac_code)
  get #CAN, #0, #UFCI_CAN_CODE, 4, id$ ' Test: Access-Code lesen
  get #CAN, #0, #UFCI_CAN_CODE, 0, ac_code ' und in LONG lesen
  wait_duration 1000

' wenn der Code als Zahl vorliegt:
  ac_code = byte_mirr ( (1FFFFFFFh shl 3), 4 )' groesster Access-Code
  put #CAN, #0, #UFCO_CAN_CODE, ac_code ' und Access-Code setzen
  call show_codemask               ' und anzeigen
  wait_duration 1000

' das ist dasselbe:
  id$ = "FF FF FF F8"%             ' 1FFFFFFF linksbuendig
  put #CAN, #0, #UFCO_CAN_CODE, id$ ' und setzen
  call show_codemask               ' und anzeigen
  wait_duration 1000

' neuen Code setzen fuer nachfolgenden Lesetest
  ac_code = byte_mirr ( (12345678h shl 3), 4 ) ' wird 0C0B3A291h
  put #CAN, #0, #UFCO_CAN_CODE, ac_code ' und Access-Code setzen
```

```
  wait_duration 1000
' ab hier steppen
  get #CAN, #0, #UFCI_CAN_CODE, 0, ac_code ' siehe Byte-Reihenfolge
  ac_code = byte_mirr ( ac_code, 4 )       ' nach jedem Schritt
  ac_code = ac_code shr 3
  print_using #LCD, "<1>ac_code:";ac_code

END


'---------------------------------------------------------------------
' zeigt Access-Code und Access-Mask an
'---------------------------------------------------------------------
SUB show_codemask
  get #CAN, #0, #UFCI_CAN_CODE, 0, ac_code ' und access code lesen
  ac_code = byte_mirr ( ac_code, 4 ) ' Bytefolge gespiegelt fuer LONG
  print_using #LCD, "<1>ac_code:";ac_code
  get #CAN, #0, #UFCI_CAN_MASK, 0, ac_mask ' und access mask lesen
  ac_mask = byte_mirr ( ac_mask, 4 ) ' Bytefolge gespiegelt fuer LONG
  print_using #LCD, "ac_mask:";ac_mask
END
```

Standard-Frame mit Single-Filter-Konfiguration

Im Modus ‚single filter' werden beim **Standard-Frame** alle ID-Bits inclusive RTR-Bit und den ersten beiden Datenbytes durch den Access-Filter geleitet und mit dem gesetzten Code verglichen. Es können jedoch auch Botschaften mit weniger als 2 Datenbytes den Filter passieren, d.h. nicht vorhandene Datenbytes erfüllen immer die Filterbedingung. Die 4 untersten Bits zwischen RTR-Bit und Daten sollten aus Gründen der Kompatibilität ‚don't care' maskiert werden.



In dem Beispielprogramm CAN_FILTER_SS.TIG wird der Access-Code nach der Installation auf 4EE0 0000 gesetzt. Die Maske betimmt, welche Bits des gesetzten Codes relevant sind. Der Wert F11F FFFF hat im Bereich des Identifiers (die 11 Bits linksbündig) insgesamt 6 ‚0'-Bits, die besagen, daß dieses Bits in der Nachricht auf dem Bus mit dem Access-Code übereinstimmen müssen, damit die Nachricht empfangen wird. Der Test zeigt, daß die Werte durchkommen, die an zweiter Stelle ein ‚E' oder ‚F' haben und an dritter Stelle ein ‚E'. An dritter Stelle kann kein F stehen, da das 12. Bit das RTR-Bit ist. Es werden also genau die Nachrichten empfangen, deren Bits zu den relevanten Bits des Access-Code passen.

Die Abbildung zeigt den Access-Code, die Access-Maske und einen Identifier als Beispiel. Nur die ID-Bits sind gezeigt. Die anderen Bits sind im Beispiel sowieso ‚don't care':

	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR
Code: 4EEh	0	1	0	0	1	1	1	0	1	1	1	0
Maske: F11h	1	1	1	1	0	0	0	1	0	0	0	1
x=nicht relevant	x	x	x	x	1	1	1	x	1	1	1	x
ID: 0EEh	0	0	0	0	1	1	1	0	1	1	1	0
ID: 7FEh	0	1	1	1	1	1	1	1	1	1	1	0

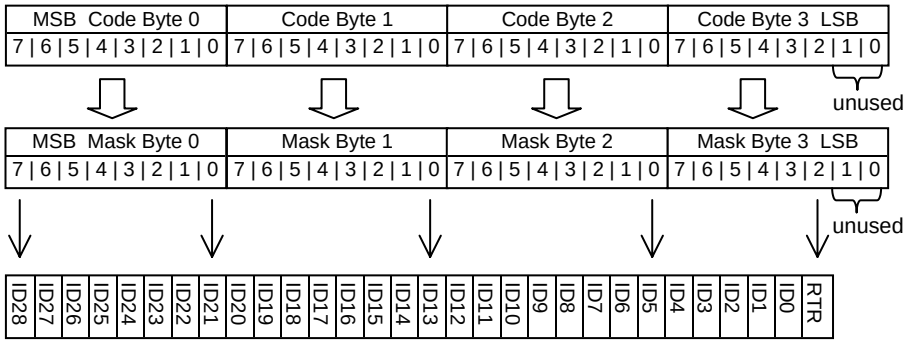
Programmbeispiel:

```
'-----------------------------------------------------------------------
' Name:  CAN_Filter_SS.TIG
' Single-Filter-Konfiguration
' Sendet Standard Frames mit verschiedenen IDs fuer Filtertest
' empfaengt gefiltert CAN-Botschaften und zeigt sie auf LCD an
' unterscheidet Standard und extended Frame
' Verbinde einen zweiten CAN-Tiger mit demselben Programm
'-----------------------------------------------------------------------
user var strict                    ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC                ' User Function Codes
#INCLUDE DEFINE_A.INC              ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                   ' CAN-Definitionen

BYTE frameformat, msg_len, can_stat
LONG ac_code, ac_mask
LONG r_id                          ' Rx ID
STRING id$(4), msg$(13), data$(8)

'-----------------------------------------------------------------------
TASK MAIN
  BYTE ever                        ' fuer Endlosschleife
  WORD ibu_fill                    ' Eingangspufferfuellung

  install_device #LCD, "LCD1.TDD" ' LCD-Treiber installieren
  install_device #CAN, "CAN1_K1.TDD", & ' CAN-Treiber installieren
    "4E E0 00 00 &                 ' access code
     F1 1F FF FF &                 ' access mask
     10 45 &                       ' bustim1, bustim2
     08 1A"%                       ' single filter mode, outctrl

' Code und Mask sind nun so gesetzt:
' 01001110111 RTR --data-- --data-- code (11 relevant Bits)
' 11110001000  1  11111111 11111111 mask (0-Bits zaehlen, 1=don't care)
' es passieren also alle Botschaften, die folgende Bitmuster haben:
' -----ID---- RTR --data-- --data-- code (11 relevant Bits)
' xxxx111x111  x  xxxxxxxx xxxxxxxx
' empfangen wird 0EEh, 0FEh, 1EEh, 1FEh, etc

  using "UH<8><8>   0 0 0 4 4"
  get #CAN, #0, #UFCI_CAN_CODE, 0, ac_code ' und access code lesen
  ac_code = byte_mirr ( ac_code, 4 ) ' Bytefolge gespiegelt fuer LONG
  print_using #LCD, "<1>ac_code:";ac_code

  get #CAN, #0, #UFCI_CAN_MASK, 0, ac_mask ' und access mask lesen
  ac_mask = byte_mirr ( ac_mask, 4 ) ' Bytefolge gespiegelt fuer LONG
  print_using #LCD, "ac_mask:";ac_mask

  run_task generate_frames             ' erzeuge aufsteigende IDs

' zeige nun IDs der empfangenen Botschaften an
  for ever = 0 to 0 step 0               ' Endlosschleife
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
'    print #LCD, "<1Bh>A<0><2><0F0h>IBU_FILL:";ibu_fill;"    ";
    if ibu_fill > 2 then           ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
```

```
        disable_tsw
        using "UH<4><4>   0 0 0 0 4"
      else                              ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        r_id = byte_mirr ( r_id, 4 )
        disable_tsw
        using "UH<8><8>   0 0 0 4 4"
      endif
      print_using #LCD, "<1Bh>A<0><2><0F0h>ID rcvd:";r_id;
      enable_tsw

      if msg_len > 0 then               ' wenn Daten
        get #CAN, #0, msg_len, data$    ' hole sie aus dem Puffer
      endif
    endif
'    get #CAN, #0, #UFCI_CAN_STAT, 0, can_stat
'    using "UH<2><2>   0 0 0 0 2"  ' HEX-Format fuer ein Byte
'    print_using #LCD, "<1Bh>A<1><1><0F0h>";can_stat;
  next
END


'--------------------------------------------------------------------
' Erzeugt Standardframes mit aufsteigender ID
'--------------------------------------------------------------------
TASK generate_frames
  BYTE ever                         ' fuer Endlosschleife
  WORD obu_free                     ' Platz im Ausgangspuffer
  LONG t_id                         ' Tx ID
  STRING msg$(13)

  t_id  = 0                         ' Standard Identifier
  for ever = 0 to 0 step 0          ' Endlosschleife
    get #CAN, #0, #UFCI_OBU_FREE, 0, obu_free
    if obu_free > 13 then
' Frame-Info 0 = standard, 2 ID-Bytes, keine Daten
      msg$ = "<0><0><0>"
      msg$ = ntos$ ( msg$, 1, -2, t_id ) ' ID high-Byte zuerst einb
      put #CAN, #0, msg$            ' sende Message im Standard-Frame
      disable_tsw
      using "UH<4><4>   0 0 0 0 4" ' fuer ID Anzeige
      print_using #LCD, "<1Bh>A<0><3><0F0h>ID sent:";t_id;
      enable_tsw
                                    ' dies zaehlt t_id um 1 hoch
                                    ' wenn der Shift um 5 des ID
                                    ' bruecksichtigt wird
      t_id = t_id + 100000b         ' naechste ID
      t_id = t_id bitand 0FFFFh     ' bleibe im Bereich Standardframe-ID
    endif
    wait_duration 30
  next
END
```

Extended Frame mit Single-Filter-Konfiguration

Beim **Extended-Frame** werden alle ID-Bits inklusive RTR-Bit durch den Filter geleitet. Die 2 untersten Bits sollten aus Gründen der Kompatibilität ‚don't care' maskiert werden.



Programmbeispiel:

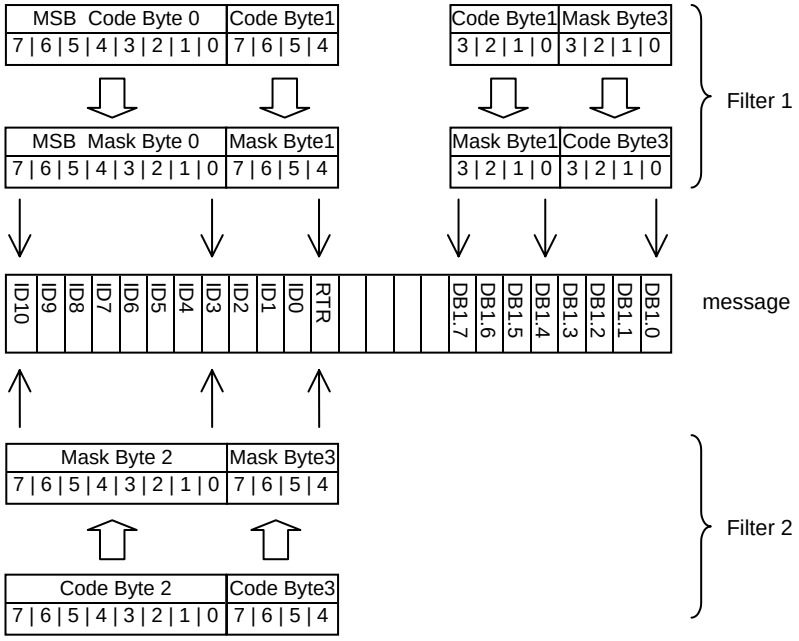
```
'--------------------------------------------------------------------
' Name:  CAN_Filter_ES.TIG
' Single-Filter-Konfiguration
' Sendet extended Frames mit verschiedenen IDs fuer Filtertest
' empfaengt gefiltert CAN-Botschaften und zeigt sie auf LCD an
' unterscheidet Standard und extended Frame
' Verbinde einen zweiten CAN-Tiger mit demselben Programm
'--------------------------------------------------------------------
user var strict                    ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC                ' User Function Codes
#INCLUDE DEFINE_A.INC              ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                   ' CAN-Definitionen

BYTE frameformat, msg_len, can_stat
LONG ac_code, ac_mask
LONG r_id
STRING id$(4), msg$(13), data$(8)

'--------------------------------------------------------------------
TASK MAIN
  BYTE ever                        ' fuer Endlosschleife
  WORD ibu_fill                    ' Eingangspufferfuellung

  install_device #LCD, "LCD1.TDD" ' LCD-Treiber installieren
  install_device #CAN, "CAN1_K1.TDD", & ' CAN-Treiber installieren
    "6D 55 D9 98 &                 ' access code
     EF FF FE FF &                 ' access mask
     10 45 &                       ' bustim1, bustim2
     08 1A"%                       ' single filter mode, outctrl

  using "UH<8><8>   0 0 0 4 4"     ' fuer ID Anzeige im gesamten Prg

  get #CAN, #0, #UFCI_CAN_CODE, 4, id$ ' Test: Access-Code lesen
  ' Bytefolge mit Anzeige - ueberwachte Ausdrücke ansehen
  get #CAN, #0, #UFCI_CAN_CODE, 0, ac_code ' und access code lesen
  ac_code = byte_mirr ( ac_code, 4 ) ' Bytefolge gespiegelt fuer LONG
  print_using #LCD, "<1>ac_code:";ac_code
  wait_duration 2000

' Code und Mask werden fuer extended Frames nun so gesetzt:
' 22222222 21111111 111
' 87654321 09876543 21098765 43210Rxx RTR, 2x don't care
' 01101101 01010101 11011001 10011000 code (29 relevante Bits+RTR)
' 11101111 11111111 11111110 11111111 mask (0-Bits sind relevant)
'                                     RTR und 2 nicht benutzte Bits
' es passieren also alle Botschaften, die folgende Bitmuster haben:
' xxx0xxxx xxxxxxxx xxxxxxx1 xxxxxxxx
' Bit 5 muss gesetzt und Bit 25 0 sein

' wenn der Code als Zahl vorliegt:
  ac_code = byte_mirr ( (0DAABB33h shl 3), 4 ) ' neuer Access-Code
  put #CAN, #0, #UFCO_CAN_CODE, ac_code ' und Access-Code setzen
' das ist dasselbe:
'  id$ = "6D 55 D9 98"%              ' neuer Access-Code
'  put #CAN, #0, #UFCO_CAN_CODE, id$ ' und Access-Code setzen

' Bytefolge wieder mit Anzeige - ueberwachte Ausdrücke ansehen
  get #CAN, #0, #UFCI_CAN_CODE, 4, id$ ' und Access-Code in String
```

```
  get #CAN, #0, #UFCI_CAN_CODE, 0, ac_code ' und in LONG lesen
  ac_code = byte_mirr ( ac_code, 4 )
  print_using #LCD, "<1>ac_code:";ac_code
  wait_duration 1000

  ac_mask = byte_mirr ( 0EFFFFEFFh, 4 ) ' Access-Maske
  put #CAN, #0, #UFCO_CAN_MASK, ac_mask ' und access mask setzen
  get #CAN, #0, #UFCI_CAN_MASK, 0, ac_mask ' und access mask lesen
  ac_mask = byte_mirr ( ac_mask, 4 ) ' Bytefolge gespiegelt fuer LONG
  print_using #LCD, "ac_mask:";ac_mask

  run_task generate_frames              ' erzeuge aufsteigende IDs

' zeige nun IDs der empfangenen Botschaften an
  for ever = 0 to 0 step 0               ' Endlosschleife
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
'    print #LCD, "<1Bh>A<0><2><0F0h>IBU_FILL:";ibu_fill;"    ";
    if ibu_fill > 2 then          ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        r_id = r_id shr 5
      else                              ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        r_id = byte_mirr ( r_id, 4 )
        r_id = r_id shr 3
        if msg_len > 0 then             ' wenn Daten
          get #CAN, #0, msg_len, data$  ' hole sie aus dem Puffer
        endif
      endif
      disable_tsw
      using "UH<8><8>   0 0 0 4 4"    ' fuer ID Anzeige               to
      print_using #LCD, "<1Bh>A<0><2><0F0h>ID rcvd:";r_id;
      enable_tsw

      if msg_len > 0 then               ' wenn Daten
        get #CAN, #0, msg_len, data$    ' hole sie aus dem Puffer
      endif
    endif
'    get #CAN, #0, #UFCI_CAN_STAT, 0, can_stat
'    using "UH<2><2>   0 0 0 0 2"  ' HEX-Format fuer ein Byte
'    print_using #LCD, "<1Bh>A<1><1><0F0h>";can_stat;
  next
END


'---------------------------------------------------------------------
' Erzeugt exteded Frames mit aufsteigender ID
'---------------------------------------------------------------------
TASK generate_frames
  BYTE ever
  WORD obu_free
  LONG t_id
  STRING msg$(13)

  using "UH<8><8>   0 0 0 4 4"    ' fuer ID Anzeige im gesamten Prg
  t_id  = 0AABB00h shl 3          ' extended Identifier
  for ever = 0 to 0 step 0        ' Endlosschleife
```

```
    if obu_free > 13 then
' Frame-Info 80h = extended, 4 ID-Bytes, keine Daten
      msg$ = "<80h><0><0><0><0>"
      msg$ = ntos$ ( msg$, 1, -4, t_id ) ' ID high-Byte zuerst einb
      put #CAN, #0, msg$           ' sende Message im Standard-Frame
      print_using #LCD, "<1Bh>A<0><3><0F0h>ID sent:";t_id shr 3;
                                  ' dies zaehlt um 1 im Byte 0 und 3
                                  ' wenn der Shift um 3 des ID
                                  ' bruecksichtigt wird
      t_id = t_id + 08000008h     ' naechste ID
    endif
    wait_duration 50
  next
END
```

Standard-Frame mit Dual-Filter-Konfiguration

Beim **Standard-Frame** werden alle ID-Bits inclusive RTR-Bit und dem ersten Datenbyte durch den ersten Access-Filter geleitet und mit dem gesetzten Code verglichen. Es können jedoch auch Botschaften mit weniger als 2 Datenbytes den Filter passieren. Ausserdem werden alle ID-Bits inclusive RTR-Bit durch den zweiten Access-Filter geleitet und mit dem gesetzten Code verglichen. Ist der Vergleich bei einem der beiden Filter erfolgreich, dann wird die CAN-Botschaft empfangen. Wenn das erste Datenbyte bei der Filterung keine Rolle spielen soll, dann werden die untersten 4 Bits der Filtermaske auf ‚don't care' gesetzt. Dann arbeiten beide Filter gleich.



In der Anzeige des Beispielprogramms auf dem LCD ist deutlich zu sehen, wie beim Doppelfilter eine Gruppe alles herausfiltert, was als dritte HEX-Ziffer ein ‚E' hat, während der zweite Filter alles durchläßt, was als zweite HEX-Ziffer ein ‚A' hat. Die meiste Zeit sind Zahlen nach dem Muster ‚xxE0' auf dem LCD zu sehen, also E ist fest und die Positionen xx zählen durch (Filter1). Wenn jedoch als zweite Ziffer das A erscheint, wird die dritte Position mit dem E durchgezählt.

Programmbeispiel:

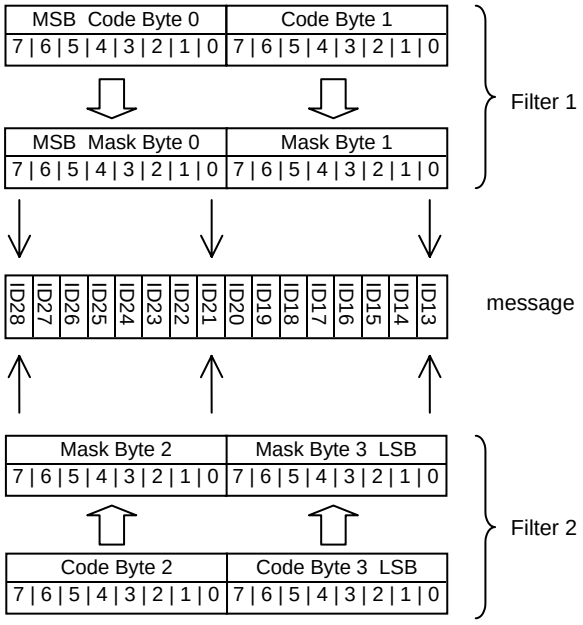
```
'--------------------------------------------------------------------
' Name:  CAN_Filter_SD.TIG
' Dual-Filter-Konfiguration
' Im Dual-Filter-Mode gibt es 2 Codes und 2 Masken mit 16 Bit
' Sendet Standard Frames mit verschiedenen IDs fuer Filtertest
' empfaengt gefiltert CAN-Botschaften und zeigt sie auf LCD an
' unterscheidet Standard und extended Frame
' Verbinde einen zweiten CAN-Tiger mit demselben Programm
'--------------------------------------------------------------------
user var strict                   ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC               ' User Function Codes
#INCLUDE DEFINE_A.INC             ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                  ' CAN-Definitionen

BYTE frameformat, msg_len, can_stat
LONG ac_code, ac_mask
LONG r_id                         ' Rx ID
STRING id$(4), msg$(13), data$(8)

'--------------------------------------------------------------------
TASK MAIN
  BYTE ever                       ' fuer Endlosschleife
  WORD ibu_fill                   ' Eingangspufferfuellung

  install_device #LCD, "LCD1.TDD" ' LCD-Treiber installieren
  install_device #CAN, "CAN1_K1.TDD", & ' CAN-Treiber installieren
    "4E E0 4A E0 &                ' access code
     FF 1F F0 FF &                ' access mask
     10 45 &                      ' bustim1, bustim2
     00 1A"%                      ' dual filter mode, outctrl

' In den ersten beiden Bytes sind Code und Mask nun so gesetzt:
' 01001110111 RTR --data-- --data-- code (11 relevant Bits)
' 11111111000  1  1111             mask (0-Bits zaehlen, 1=don't care)
' es passieren also alle Botschaften, die folgende Bitmuster haben:
' -----ID---- RTR --data-- --data-- code (11 relevant Bits)
' xxxxxxx0111  x  xxxx
' empfangen wird alles, was im ID eine 7 im low Nibble hat
' hier sichtbar als E im 3. Nibble

' In den zweiten beiden Bytes sind Code und Mask nun so gesetzt:
' 01001010111 RTR --data-- --data-- code (11 relevant Bits)
' 11110000111  1  1111             mask (0-Bits zaehlen, 1=don't care)
' es passieren also alle Botschaften, die folgende Bitmuster haben:
' -----ID---- RTR --data-- --data-- code (11 relevant Bits)
' xxx0101xxxx  x  xxxx
' empfangen wird alles, was im 2. Nibble ein A hat

  using "UH<8><8>   0 0 0 4 4"    ' fuer ID Anzeige im gesamten Prg
  get #CAN, #0, #UFCI_CAN_CODE, 0, ac_code ' und access code lesen
  ac_code = byte_mirr ( ac_code, 4 ) ' Bytefolge gespiegelt fuer LONG
  print_using #LCD, "<1>ac_code:";ac_code

  get #CAN, #0, #UFCI_CAN_MASK, 0, ac_mask ' und access mask lesen
  ac_mask = byte_mirr ( ac_mask, 4 ) ' Bytefolge gespiegelt fuer LONG
  print_using #LCD, "ac_mask:";ac_mask

  run_task generate_frames             ' erzeuge aufsteigende IDs
```

```
' zeige nun IDs der empfangenen Botschaften an
  for ever = 0 to 0 step 0                ' Endlosschleife
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
'    print #LCD, "<1Bh>A<0><2><0F0h>IBU_FILL:";ibu_fill;"    ";
    if ibu_fill > 2 then           ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        disable_tsw
        using "UH<4><4>   0 0 0 0 4"
      else                             ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id
        r_id = byte_mirr ( r_id, 4 )
        disable_tsw
        using "UH<8><8>   0 0 0 4 4"
      endif
      print_using #LCD, "<1Bh>A<0><2><0F0h>ID rcvd:";r_id;
      enable_tsw
      if msg_len > 0 then              ' wenn Daten
        get #CAN, #0, msg_len, data$    ' hole sie aus dem Puffer
      endif
    endif
  next
END

'---------------------------------------------------------------------
' Erzeugt Standardframes mit aufsteigender ID
'---------------------------------------------------------------------
TASK generate_frames
  BYTE ever                       ' fuer Endlosschleife
  WORD obu_free                   ' Platz im Ausgangspuffer
  LONG t_id                       ' Tx ID
  STRING msg$(13)


  t_id  = 0                       ' Standard Identifier
  for ever = 0 to 0 step 0        ' Endlosschleife
    get #CAN, #0, #UFCI_OBU_FREE, 0, obu_free
    if obu_free > 13 then
' Frame-Info 0 = standard, 2 ID-Bytes, keine Daten
      msg$ = "<0><0><0>"
      msg$ = ntos$ ( msg$, 1, -2, t_id ) ' ID high-Byte zuerst einb
      put #CAN, #0, msg$          ' sende Message im Standard-Frame
      disable_tsw
      using "UH<4><4>   0 0 0 0 4"
      print_using #LCD, "<1Bh>A<0><3><0F0h>ID sent:";t_id;
      enable_tsw
                                  ' dies zaehlt t_id um 1 hoch
                                  ' wenn der Shift um 5 des ID
                                  ' bruecksichtigt wird
      t_id = t_id + 100000b       ' naechste ID
    endif
    wait_duration 30
  next
END
```

2

Extended Frame mit Dual-Filter-Konfiguration

Beim **Extended-Frame** arbeiten beide Filter gleich. Es werden die ersten beiden ID-Bytes durch die beiden Access-Filter geleitet und mit dem gesetzten Code verglichen. Es werden nur ID13...ID28 ausgewertet. Signalisiert der Vergleich bei einem der beiden Filter Akzeptanz, dann wird die CAN-Botschaft empfangen.



Programmbeispiel:

```
'--------------------------------------------------------------------
' Name:  CAN_Filter_ED.TIG
' Dual-Filter-Konfiguration
' Im Dual-Filter-Mode gibt es 2 Codes und 2 Masken mit 16 Bit.
' Sendet extended Frames mit verschiedenen IDs fuer Filtertest,
' empfaengt gefiltert CAN-Botschaften und zeigt sie auf LCD an.
' Unterscheidet Standard und extended Frame.
' Verbinde einen zweiten CAN-Tiger mit demselben Programm.
'--------------------------------------------------------------------
user var strict                   ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC               ' User Function Codes
#INCLUDE DEFINE_A.INC             ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                  ' CAN-Definitionen

BYTE frameformat, msg_len, can_stat
LONG ac_code, ac_mask
LONG r_id                         ' Rx ID
STRING id$(4), msg$(13), data$(8)

'--------------------------------------------------------------------
TASK MAIN
  BYTE ever                       ' fuer Endlosschleife
  WORD ibu_fill                   ' Eingangspufferfuellung

  install_device #LCD, "LCD1.TDD" ' LCD-Treiber installieren
  install_device #CAN, "CAN1_K1.TDD", & ' CAN-Treiber installieren
    "10 55 10 AA &                ' access code
     FF 00 EF 00 &                ' access mask
     10 45 &                      ' bustim1, bustim2
     00 1A"%                      ' dual filter mode, outctrl

' In den ersten beiden Bytes sind Code und Mask nun so gesetzt:
' 22222222 21111111 111
' 87654321 09876543 21098765 43210Rxx RTR, 2x don't care
' 00010000 01010101                  code (29 relevante Bits+RTR)
' 11111111 00000000                  mask (0-Bits sind relevant)
'                                    RTR und 2 nicht benutzte Bits
' es passieren also alle Botschaften, die folgende Bitmuster haben:
' xxxxxxxx 01010101
' empfangen wird alles, was im ID eine 55h im 2. Byte hat              r

' In den zweiten beiden Bytes sind Code und Mask nun so gesetzt:
' 22222222 21111111 111
' 87654321 09876543 21098765 43210Rxx RTR, 2x don't care
' 00010000 10101010                  code (29 relevante Bits+RTR)
' 11101111 00000000                  mask (0-Bits sind relevant)
'                                    RTR und 2 nicht benutzte Bits
' es passieren also alle Botschaften, die folgende Bitmuster haben:
' xxx1xxxx 10101010
' empfangen wird, was mit 0AAh, 10AAh, 20AAh, etc. beginnt

  using "UH<8><8>   0 0 0 4 4"    ' fuer ID Anzeige im gesamten Prg

  get #CAN, #0, #UFCI_CAN_CODE, 0, ac_code ' und access code lesen
  ac_code = byte_mirr ( ac_code, 4 ) ' Bytefolge gespiegelt fuer LONG
  print_using #LCD, "<1>ac_code:";ac_code

  get #CAN, #0, #UFCI_CAN_MASK, 0, ac_mask ' und access mask lesen
```

```
  print_using #LCD, "ac_mask:";ac_mask

  run_task generate_frames              ' erzeuge aufsteigende IDs

' zeige nun IDs der empfangenen Botschaften an
  for ever = 0 to 0 step 0               ' Endlosschleife
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
'    print #LCD, "<1Bh>A<0><2><0F0h>IBU_FILL:";ibu_fill;"    ";
    if ibu_fill > 2 then          ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
      else                              ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        r_id = byte_mirr ( r_id, 4 )    ' ungeshiftet anzeigen
      endif
      print_using #LCD, "<1Bh>A<0><2><0F0h>ID rcvd:";r_id;

      if msg_len > 0 then               ' wenn Daten
        get #CAN, #0, msg_len, data$    ' hole sie aus dem Puffer
      endif
    endif
  next
END

'---------------------------------------------------------------------
' Erzeugt exteded Frames mit aufsteigender ID
'---------------------------------------------------------------------
TASK generate_frames
  BYTE ever                         ' fuer Endlosschleife
  WORD obu_free                     ' Platz im Ausgangspuffer
  LONG t_id                         ' Tx ID
  STRING msg$(13)

  t_id  = 10000020h                 ' extended Identifier
  for ever = 0 to 0 step 0          ' Endlosschleife
    get #CAN, #0, #UFCI_OBU_FREE, 0, obu_free
    if obu_free > 11 then
' Frame-Info 80h = extended, 4 ID-Bytes, keine Daten
      msg$ = "<80h><0><0><0><0>"
      msg$ = ntos$ ( msg$, 1, -4, t_id ) ' ID high-Byte zuerst einb
      put #CAN, #0, msg$            ' sende Message im Standard-Frame
      disable_tsw
      using "UH<8><8>   0 0 0 4 4"
      print_using #LCD, "<1Bh>A<0><3><0F0h>ID sent:";t_id;
      enable_tsw
                                    ' dies zaehlt um 1 im Byte 2 und 3
                                    ' wenn der Shift um 3 des ID
                                    ' bruecksichtigt wird
      t_id = t_id + 0008000h        ' naechste ID
      t_id = t_id bitand 0FFFFh     ' bleibe im Bereich Standardframe-ID
    endif
    wait_duration 50
  next
END
```

Versenden von CAN-Botschaften

Der CAN-Device-Treiber unterstützt folgende Methoden des Versands:

Versenden einzelner Botschaften, die 0...8 Zeichen enthalten und deren Identifier nach Bedarf einzeln festgelegt wird. Jede CAN-Botschaft wird mit einer PUT-oder PRINT-Instruktion ausgegeben. Bei der Print-Instruktion ist zu beachten daß die Ausgabe formatiert wird und eventuell zusätzliche Byte (CR, LF) angehängt werden.

Versenden von Daten, die auch mehr als 8 Zeichen enthalten können. Der Device-Treiber macht soviele CAN-Datenpakete daraus, wie zum Versand der gesamten Datenmenge erforderlich ist und verwendet den Identifier, der am Anfang des Strings angegeben wurde. Die Daten mit einer einzigen PUT- oder PRINT-Instruktion in den Puffer übergeben.

Antworten auf ein ‚Remote Transmission Request', indem eine Botschaft speziell für diesen Zweck im Device-Treiber bereitgestellt wird. Die bereitgestellte Nachricht wird vom Treiber automatisch versandt, wenn eine RTR-Message empfangen wird.

Der CAN-Device-Treiber erwartet als Argument eine CAN-Botschaft im vorgegebenen Format. Das erste Byte wird als Frame-Format-Byte interpretiert. Je nach Frame-Format sind die nächsten 2 oder 4 Bytes der Identifier der Nachricht. Eine typische CAN-Ausgabe als Standard Frame hat folgendes Aussehen:

PUT #CAN, #0, *"<Frame-Format><ID1><ID2>data"*

<Frame-Format>	enthält Information, daß es ein Standard-Frame ist.
<ID1>	enthält die oberen Bits 3...10 des Identifiers.
<ID2>	enthält die unteren Bits 0...2 des Identifiers an den Bitpositionen 5, 6 und 7. Die übrigen Bits in diesem Byte sind bedeutungslos.
data	sind Datenbytes, die in der Botschaft übertragen werden. 0...8 Datenbytes sind möglich.

Bei 0...8 Datenbytes entsteht daraus eine CAN-Botschaft. Wenn mehr als 8 Datenbytes enthalten sind, dann verpackt der Device-Treiber Die Daten in mehrere CAN-Botschaften und verwendet den gleichen Identifier.

Aus

PUT #CAN, #0, *"<Frame-Format><ID1><ID2>abcdefghijklmnopqrs"*

werden folgende CAN-Botschaften:

"<Frame-Format><ID1><ID2>abcdefgh"

“<Frame-Format><ID1><ID2>ijklmnop”

“<Frame-Format><ID1><ID2>qrs”

Werden die Daten über die Sekundär-Adresse 1 verschickt, dann wird in der Botschaft das RTR-Bit gesetzt und dadurch ein ‚Remote Transmission Request’ erzeugt.

Eine Einzel-Nachricht mit maximal 8 Datenbytes an Sekundär-Adresse 2 hinterlegt eine Antwort, die dann versandt wird, wenn der Device-Treiber selbst ein ‚Remote Transmission Request’ empfängt.

Sek.-Adr.	Funktion
0	Normaler Datenversand
1	Datenversand mit ‚Remote Transmission Request’
2	Hinterlegen einer Antwort-Botschaft, die dann versandt wird, wenn der Device-Treiber selbst ein ‚Remote Transmission Request’ empfängt.

Ein einfaches Sendebeispiel für **Standardframe**-CAN-Botschaften zeigt das folgende Programm.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name:  CAN_TXS.TIG
' sendet 'the quick brown fox' ueber den CAN-Bus in Standard-Frames
' Verbinde ein empfangendes CAN-Gerät, z.B. einen CAN-Tiger
'--------------------------------------------------------------------
user var strict                    ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC                ' User Function Codes
#INCLUDE DEFINE_A.INC              ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                   ' CAN-Definitionen

'--------------------------------------------------------------------
TASK MAIN
  BYTE ever, i_msg, can_stat
  WORD obu_free                    ' Platz im Output-Puffer
  WORD t_id                        ' Transmit ID
  STRING data$, msg$(11)

  install_device #LCD, "LCD1.TDD" ' LCD-Treiber installieren
  install_device #CAN, "CAN1_K1.TDD", & ' CAN-Treiber installieren
    "50 A0 00 00 &                 ' access code
     FF FF FF FF &                 ' access mask
     10 45 &                       ' bustim1, bustim2
     08 1A"%                       ' single filter mode, outctrl

  data$ = "the quick brown fox jumps over the lazy dog"
  i_msg = 0                        ' Index fuer laufenden Text
  t_id  = 155h shl 5               ' Standard Identifier

  for ever = 0 to 0 step 0         ' Endlosschleife
    get #CAN, #0, #UFCI_OBU_FREE, 0, obu_free
    print #LCD, "<1Bh>A<0><1><0F0h>OBU_FREE:";obu_free;"    ";
    if obu_free > 11 then
      msg$ = &  ' Frame-Info 0 = standard, 2 ID-Bytes, data
      "<0><0><0>" + mid$ ( data$, i_msg, 8 )' Frame-Info, ID     frame i
      msg$ = ntos$ ( msg$, 1, -2, t_id ) ' ID high-Byte zuerst einb
      print #CAN, #0, msg$;        ' sende Message im Standard-Frame
      i_msg = i_msg + 1            ' String-Index eins weiter
      if i_msg > len(data$)-8 then ' Achtung Limit
        i_msg = 0
      endif
    endif                          ' pruefe CAN Status
    get #CAN, #0, #UFCI_CAN_STAT, 0, can_stat
    using "UH<2><2>   0 0 0 0 2"  ' HEX-Format fuer ein Byte
    print_using #LCD, "<1Bh>A<0><0><0F0h>CAN-State:";can_stat;
    wait_duration 200
  next
END
```

Ein einfaches Sendebeispiel für **extended Frame**-CAN-Botschaften zeigt das folgende Programm.

Programmbeispiel:

```
'-------------------------------------------------------------------------
' Name:  CAN_TXE.TIG
' sendet 'the quick brown fox' ueber den CAN-Bus in extended Frames
' Verbinde ein empfangendes CAN-Gerät, z.B. einen CAN-Tiger
'-------------------------------------------------------------------------
user var strict                    ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC                ' User Function Codes
#INCLUDE DEFINE_A.INC              ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                   ' CAN-Definitionen

'-------------------------------------------------------------------------
TASK MAIN
  BYTE ever, i_msg, can_stat
  WORD obu_free                    ' Platz im Output-Puffer
  LONG t_id                        ' extended ID 4 Bytes
  STRING data$, msg$(13)

  install_device #LCD, "LCD1.TDD" ' LCD-Treiber installieren
  install_device #CAN, "CAN1_K1.TDD", & ' CAN-Treiber installieren
    "50 A0 00 00 &                 ' access code
     FF FF FF FF &                 ' access mask
     10 45 &                       ' bustim1, bustim2
     08 1A"%                       ' single filter mode, outctrl

  data$ = "the quick brown fox jumps over the lazy dog"
  i_msg = 0                        ' Index fuer laufenden Text
  t_id = 01733F055h shl 3          ' extended Identifier

  for ever = 0 to 0 step 0         ' Endlosschleife
    get #CAN, #0, #UFCI_OBU_FREE, 0, obu_free
    print #LCD, "<1Bh>A<0><1><0F0h>OBU_FREE:";obu_free;"    ";
    if obu_free > 13 then
      msg$ = &  ' Frame-Info 80h = extended, 4 ID-Bytes, data
      "<80h><0><0><0><0>" + mid$ ( data$, i_msg, 8 )
      msg$ = ntos$ ( msg$, 1, -4, t_id ) ' ID high-Byte zuerst einb
      print #CAN, #0, msg$;        ' sende Message im extended-Frame
      i_msg = i_msg + 1            ' String-Index eins weiter
      if i_msg > len(data$)-8 then ' Achtung Limit
        i_msg = 0
      endif
    endif                          ' pruefe CAN Status
    get #CAN, #0, #UFCI_CAN_STAT, 0, can_stat
    using "UH<2><2>   0 0 0 0 2"  ' HEX-Format fuer ein Byte
    print_using #LCD, "<1Bh>A<0><0><0F0h>CAN-State:";can_stat;
    wait_duration 200
  next
END
```

Empfangen von CAN-Botschaften

Der CAN-Device-Treiber empfängt CAN-Botschaften und legt diese im Empfangspuffer ab. Das Auslesen des Empfangspuffers bei dem CAN-Device-Treiber ist ein besonderer Vorgang und unterscheidet sich vom Auslesen anderer Puffer (z.B. des seriellen oder parallelen Treibers), da hier Botschaften im Puffer vorliegen, die außer den Daten weitere Informationen enthalten. Die Botschaften werden immer vollständig gelesen und entsprechend ihrem Messagetyp verarbeitet:

Zwei Lesemodi lesen unterschiedlich von den Sekundär-Adressen 0 und 1:

Sek.Adr.	
0	Die Bytes der CAN-Botschaften werden so gelesen, wie sie im Puffer vorliegen, inklusive Frame-Format- und ID-Bytes.
1	Nur Datenbytes werden gelesen. Frame-Format- und ID-Bytes werden ignoriert. Die Längeninformation von teilweise eingelesenen CAN-Botschaften wird im Puffer automatisch korrigiert.

Achtung: Von der Sekundär-Adresse 0 müssen die CAN-Botschaften vollständig gelesen werden, da sonst der nächste Lesevorgang nicht mit dem Frame-Info-Byte der nächtsen CAN-Botschaft beginnt.

Über die Skundär-Adresse 0 erfolgt das **Auslesen einzelner Botschaften**, die 0...8 Zeichen enthalten und deren Frameformatkennung und Identifier den Datenbytes vorangestellt sind. Zunächst wird das Frame-Info-Byte gelesen und festgestellt, ob es sich um einen ‚Standard-Frame' oder einen ‚extended Frame' handelt und wieviele Datenbytes enthalten sind. Dann werden die ID-Bytes gelesen, die den anwendungsspezifischen Botschaftstyp darstellen. Anschließend werden die Datenbytes eingelesen.

Das Beispielprogramm CAN_RX1.TIG liest die empfangenen Nachrichten aus dem Puffer, unterscheidet dabei Standardframes und extended Frames und zeigt diese in hexadezimaler Form an.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name:  CAN_RX1.TIG
' empfaengt ungefiltert CAN-Messages und zeigt sie auf LCD an
' unterscheidet Standard und extended Frame
' zeigt auch Status an
' Verbinde ein sendendes CAN-Gerät, z.B. einen CAN-Tiger
'--------------------------------------------------------------------
user var strict                   ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC               ' User Function Codes
#INCLUDE DEFINE_A.INC             ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                  ' CAN-Definitionen

'--------------------------------------------------------------------
TASK MAIN
  BYTE ever, frameformat, msg_len, can_stat
  WORD ibu_fill                   ' Eingangspufferfuellung
  LONG r_id                       ' empfangene ID
  STRING msg$(8), data$(8)

  install_device #LCD, "LCD1.TDD" ' LCD-Treiber installieren
  install_device #CAN, "CAN1_K1.TDD", & ' CAN-Treiber installieren
    "50 A0 00 00 &                ' access code
     FF FF FF FF &                ' access mask
     10 45 &                      ' bustim1, bustim2
     08 1A"%                      ' single filter mode, outctrl

  print #LCD, "<1Bh>A<0><0><0F0h>STAT LEN ID";

  for ever = 0 to 0 step 0              ' Endlosschleife
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
    print #LCD, "<1Bh>A<0><3><0F0h>IBU_FILL:";ibu_fill;"    ";
    if ibu_fill > 2 then          ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )     ' Bytefolge f. Tiger WORD
        r_id = r_id shr 5                ' rechtsbuendig schieben
        using "UH<8><3>   0 0 0 0 3"     ' fuer ID Anzeige
      else                               ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 4 )     ' low byte 1st in LONG
        r_id = r_id shr 3                ' rechtsbuendig schieben
        using "UH<8><8>   0 0 0 4 4"     ' fuer ID Anzeige
      endif
      print_using #LCD, "<1Bh>A<9><1><0F0h>";r_id;

      using "UH<1><1>   0 0 0 0 1"       ' zeige Laenge an
      print_using #LCD, "<1Bh>A<6><1><0F0h>";msg_len;
      if msg_len > 0 then                ' wenn Daten
        get #CAN, #0, msg_len, data$     ' hole sie und zeige an
        msg$ = "        "                ' 8 Leerzeichen
        msg$ = stos$ ( msg$, 0, data$, msg_len )' LCD-Feld vorbereiten
        print #LCD, "<1Bh>A<0><2><0F0h>data:";msg$;
      else
        print #LCD, ;"          ";
      endif
    endif

    get #CAN, #0, #UFCI_CAN_STAT, 0, can_stat ' CAN-Status
```

```
    print_using #LCD, "<1Bh>A<1><1><0F0h>";can_stat;
  next
END
```

Über die Skundär-Adresse 1 erfolgt das **Auslesen von Daten** ohne Berücksichtigung der Frame-Format- und Identifier-Bytes. Der Device-Treiber liest nur die Datenbytes und ignoriert die Identifier. Unvollständig gelesene CAN-Botschaften behalten das Frameformat und ihre ID-Bytes, die Länge wird vom Treiber entsprechen korrigiert, so daß der nächste Lesevorgang wieder eine intakte CAN-Nachricht im Puffer vorfindet.

Programmbeispiel:

```
'---------------------------------------------------------------------
' Name:  CAN_RX2.TIG
' empfaengt CAN-Daten und zeigt sie an, ignoriert IDs
' zeigt Daten als Text an (nur ASCII senden)
' zeigt auch Status an
' Verbinde ein sendendes CAN-Gerät, z.B. einen CAN-Tiger
'---------------------------------------------------------------------
user var strict                   ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC               ' User Function Codes
#INCLUDE DEFINE_A.INC             ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                  ' CAN-Definitionen

'---------------------------------------------------------------------
TASK MAIN
  BYTE ever, frameformat, msg_len, can_stat
  WORD ibu_fill                   ' Ausgangspufferfuellung
  LONG r_id
  STRING id$(4), data$, line$

  install_device #LCD, "LCD1.TDD" ' LCD-Treiber installieren
  install_device #CAN, "CAN1_K1.TDD", & ' CAN-Treiber installieren
    "50 A0 00 00 &                ' access code
     FF FF FF FF &                ' access mask
     10 45 &                      ' bustim1, bustim2
     08 1A"%                      ' single filter mode, outctrl

  print #LCD, "<1Bh>A<0><0><0F0h>STAT LEN ID";

  line$ = ""
  for ever = 0 to 0 step 0        ' Endlosschleife
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
    print #LCD, "<1Bh>A<0><3><0F0h>IBU_FILL:";ibu_fill;"    ";
    get #CAN, #1, 0, data$
    if data$ <> "" then
      line$ = line$ + data$
      if len(line$) > 20 then     ' wenn laenger als LCD-Zeile
        line$ = right$ ( line$, 20 )
      endif
      print #LCD, "<1Bh>A<0><2><0F0h>";line$;
    endif
    get #CAN, #0, #UFCI_CAN_STAT, 0, can_stat
    using "UH<2><2>   0 0 0 0 2"  ' HEX-Format fuer ein Byte
    print_using #LCD, "<1Bh>A<1><1><0F0h>";can_stat;
  next
END
```

Empfang eines ‚Remote Transmission Request', führt dazu, daß eine Botschaft, die speziell für diesen Zweck im Device-Treiber bereitgestellt wurde, verschickt wird. Die empfangen CAN-Botschaft wird ansonsten so behandelt, wie eine CAN-Botschaft ohne ‚Remote Transmission Request'.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name:  CAN_RTR.TIG
' bereitet eine RTR-Message vor und sendet dann CAN-Nachrichten
' in einer Schleife.
' RTR und Schleifen-Nachrichten haben unterschiedliche ID.
' Verbinde ein CAN-Gerät, welches mit einer RTR-Message die Antwort
' abruft, z.B. einen CAN-Tiger mit CAN_RTRS.TIG
'--------------------------------------------------------------------
user var strict                    ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC                ' User Function Codes
#INCLUDE DEFINE_A.INC              ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                   ' CAN-Definitionen

'--------------------------------------------------------------------
TASK MAIN
  BYTE ever                        ' Endlosschleife
  STRING rtr_msg$(13)

  install_device #LCD, "LCD1.TDD" ' LCD-Treiber installieren
  install_device #CAN, "CAN1_K1.TDD", & ' CAN-Treiber installieren
    "50 A0 00 00 &                 ' access code
     FF FF FF FF &                 ' access mask
     10 45 &                       ' bustim1, bustim2
     08 1A"%                       ' single filter mode, outctrl

  rtr_msg$ = "<0><0FFh><0E0h>RTR-resp"' RTR Antwort als Standard-Frame
  put #CAN, #2, rtr_msg$           ' in Treiber hinterlegen
  print #LCD, "RTR-message prepared"
                                   ' jetzt mache etwas anderes
  for ever = 0 to 0 step 0         ' Endlosschleife
    wait_duration 3000
    put #CAN, #0, "<0><0FFh><0C0h>abcdefgh"
    wait_duration 3000
    put #CAN, #0, "<0><0FFh><080h>ijklmnop"
  next
END
```

Ein- und Ausgangspuffer

CAN-Botschaften bestehen aus einem Frame-Format-Byte, einem Identifier und maximal 8 Datenbytes. Im Falle eines ‚Standard-Frames' belegt der Identifier 2 Bytes. Beim ‚extended Frame' ist der Identifier 4 Bytes lang. Jede Botschaft wird zusammen mit dem Frame-Format-Byte und dem Identifier im Puffer abgelegt. Wenn eine Botschaft nicht mehr in den Puffer paßt, dann wartet beim Senden die PUT-Instruktion, bis wieder Platz im Puffer ist. Beim Empfangen wird die Botschaft verworfen und ein Overflow-Fehler registriert.

Anzahl Datenbytes	**belegt im Puffer**	
	Standard Frame	extended Frame
0	3	5
8	11	13

Anmerkung: wird mit einer einzigen PUT-Instruktion ein String in den Puffer übertragen, der mehr als 8 datenbytes enthält, dann wird Platz für zusätzliche Identifier benötigt, da die Daten auf mehrere CAN-Botschaften verteilt werden.

Sowohl eingehende als auch gesendete Daten werden in einem Puffer zwischengespeichert. Größe, Füllstand oder verbleibender Platz der Ein- und Ausgangspuffer sowie die Version des Treibers können mit Hilfe der User-Function-Codes abgefragt werden.

Sowohl bei der Ausgabe als auch beim Empfang gilt ein Puffer als voll, sobald weniger als 13 Bytes frei sind. Eine CAN-Nachricht im Extended-Frame-Format ist 13 Bytes lang. Da keine halbe CAN-Nachrichten gespeichert werden können, gilt diese Grenze.

User-Function-Codes für Abfragen (Instruktion GET):

Nr	Symbol Prefix UFCI_	Beschreibung
1	UFCI_IBU_FILL	Füllstand des Eingangspuffers (Byte)
2	UFCI_IBU_FREE	freier Platz im Eingangspuffer (Byte)
3	UFCI_IBU_VOL	Größe des Eingangspuffers (Byte)
33	UFCI_OBU_FILL	Füllstand des Ausgangspuffers (Byte)
34	UFCI_OBU_FREE	freier Platz im Ausgangspuffer (Byte)
35	UFCI_OBU_VOL	Größe des Ausgangspuffers (Byte)

Wenn nicht genügend Platz im Ausgabepuffer ist und trotzdem ausgegeben wird, dann wartet die Instruktion PUT oder PRINT (und damit die ganze Task) solange, bis wieder Platz im Puffer ist. Dieses Warten kann verhindert werden, indem vor der Ausgabe der freie Platz im Puffer abgefragt wird.

Beispiel: gebe nur aus, wenn noch genügend freier Platz im Ausgangspuffers ist:

```
GET #CAN, #0, #UFCI_OBU_FREE, 0, wVarFree
IF wVarFree > (LEN(A$)) THEN
  PUT #CAN, #0, A$
ENDIF
```

Beispiel: pruefe, ob eine Nachricht im Eingangspuffers ist (die kürzeste mögliche Nachricht ist 3 bytes lang):

```
GET #CAN, #0, #UFCI_IBU_FILL, 0, wVarFill
IF wVarFill > 2 THEN
  ' lies die CAN-Nachricht
ENDIF
```

Automatische Bitratenerkennung

Wird der Treiber im Modus ‚listen-only' installiert, dann versucht er, die Bitrate automatisch zu erkennen. Durch den Modus ‚listen-only' kann der CAN-Chip selbst nichts senden, und so werden auch nicht die sonst üblichen Fehlertelegramme erzeugt, solange die Bitrate noch nicht erkannt ist. Welche Bitraten erkannt werden können, wird durch eine Tabelle vorgegegeben. Wird bei der Installation keine Tabelle übergeben, dann wird eine intern vorhandene Tabelle verwendet.

Um die Bitrate zu erkennen, sind folgende Voraussetzungen zu erfüllen:

- Es wird ein funktionierender Bus mit Datenverkehr vorausgesetzt, daß heißt, es müssen mindestens zwei aktive Teilnehmer vorhanden sein, die etwas senden.
- Die richtige Bitrate muß in der Tabelle enthalten sein.

Die Bitratenerkennung beginnt mit der ersten Einstellung aus der Tabelle, in der Regel die höchste mögliche Bitrate. Beim nächsten Datenpaket auf dem CAN-Bus tritt kein Empfangsfehler auf, wenn die Bitrate bereits stimmt. Tritt jedoch ein Empfangsfehler auf, dann schaltet der Treiber auf die nächste Bitrate der tabelle um, und wartet erneut auf ein CAN-Telegramm. Der Treiber wartet in jedem Fall bis genügend CAN-Telegramme entweder ein Erkennen der Bitrate ermöglicht haben, oder die Tabelle der möglichen Werte dreimal abgearbeitet ist. Wurde die Bitrate nicht erkannt, ist der CAN-Device-Treiber anschließend nicht installiert. Werden nur sehr selten CAN-Telegramme über den Bus geschickt und die richtige Bitrate befindet sich erst am Ende der Tabelle, dann dauert die Erkennung entsprechend lange. Wurde die Bitrate schließlich erkannt, verläßt der Device-Treiber den Modus ‚listen-only'.

Die Tabelle enthält die Einstellungen für die Register ‚bustim0' und ‚bustim1' im CAN-CHIP. Für jede Einstellung werden also 2 Bytes benötigt. Mindestens 4 Bytes müssen in der Tabelle enthalten sein, sonst wird auf die interne Tabelle zurückgegriffen, die folgende Werte enthält:

bustim0	bustim1	Bitrate
0	43h	1 Mbit
0	5Ch	500 kBit
1	5Ch	250 kBit
3	5Ch	125 kBit
4	5Ch	100 kBit
9	5Ch	50 kBit
10h	19h	45.2 kBit
0Fh	7Fh	20 kBit
1Fh	7Fh	12.5 kBit

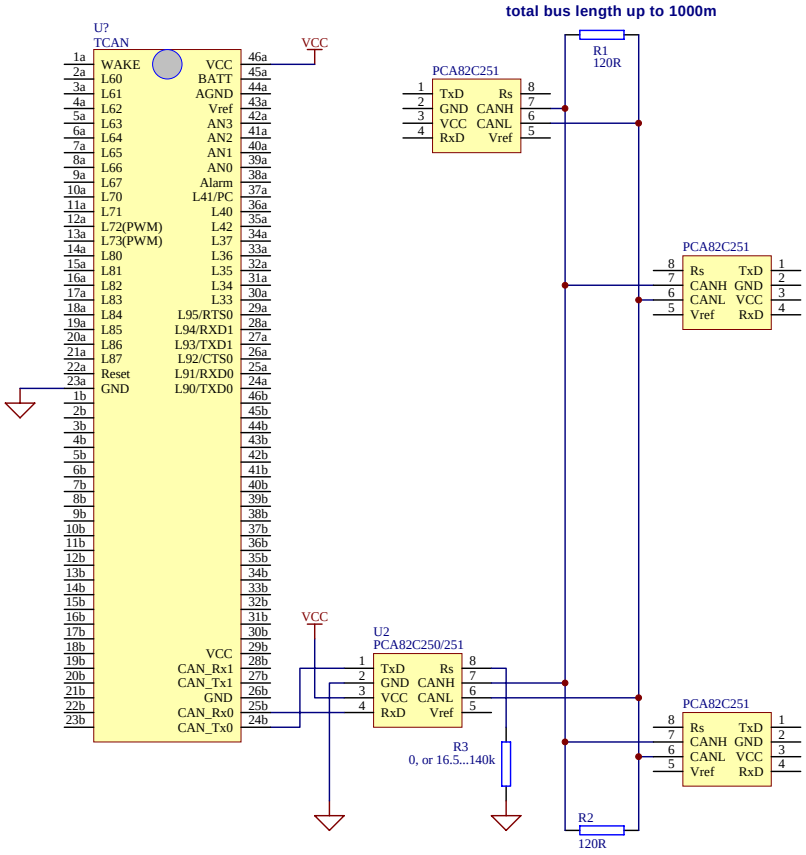
Programmbeispiel:

```
'---------------------------------------------------------------------
' Name:  CAN_ABR.TIG
' sucht automatisch die richtige Bitrate aus vorgegebener Tabelle
' sonst wie CAN_RX1.TIG
' Verbinde mit einem CAN-Bus mit Sendeverkehr
'---------------------------------------------------------------------
user var strict                   ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC               ' User Function Codes
#INCLUDE DEFINE_A.INC             ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                  ' CAN-Definitionen

'---------------------------------------------------------------------
TASK MAIN
  BYTE ever, frameformat, msg_len, can_stat
  WORD ibu_fill                   ' Eingangspufferfuellung
  LONG r_id                       ' empfangene ID
  STRING msg$(8), data$(8)

  install_device #LCD, "LCD1.TDD" ' LCD-Treiber installieren
  print #LCD, "trying to find <10><13>CAN bitrate.<10><13>Please wait...
  install_device #CAN, "CAN1_K1.TDD", & ' CAN-Treiber installieren
    "50 A0 00 00 & ' access code
     FF FF FF FF & ' access mask
     00 00 &       ' bustim1, bustim2
     0A 1A &       ' single filter + listen only, outctrl
     00 43 &       ' 1 Mbit     ab hier Tabelle mit Bytes
     00 5C &       ' 500 kbit    fuer bustim0 und bustim1
     01 5C &       ' 250 kbit           fuer automatische
     03 5C &       ' 125 kbit           Bitratenerkennung
     04 5C &       ' 100 kbit
     09 5C &       '  50 kbit
     10 45 &       '  49 kbit for SLIO: TSYNC + TSEG1 + TSEG2 = 10
     0F 7F &       '  25 kbit
     1F 7F"%       '  12.5 kbit

  print #LCD, "<1>STAT LEN ID";

  for ever = 0 to 0 step 0        ' Endlosschleife
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
    print #LCD, "<1Bh>A<0><3><0F0h>IBU_FILL:";ibu_fill;"    ";
    if ibu_fill > 3 then          ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat ' welches Frame-Format?
      msg_len = frameformat bitand 1111b
      if frameformat bitand 80h = 0 then  ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )     ' Bytefolge f. Tiger WORD
        r_id = r_id shr 5                ' rechtsbuendig schieben
        using "UH<8><3>   0 0 0 0 3"     ' fuer ID Anzeige
      else                               ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 4 )     ' low byte 1st in LONG
        r_id = r_id shr 3                ' rechtsbuendig schieben
        using "UH<8><8>   0 0 0 4 4"     ' fuer ID Anzeige
      endif
      print_using #LCD, "<1Bh>A<9><1><0F0h>";r_id;

      using "UH<1><1>   0 0 0 0 1"       ' zeige Laenge an
      print_using #LCD, "<1Bh>A<6><1><0F0h>";msg_len;
```

```
        get #CAN, #0, msg_len, data$     ' hole sie und zeige an
        msg$ = "        "                ' 8 Leerzeichen
        msg$ = stos$ ( msg$, 0, data$, msg_len )' LCD-Feld vorbereiten
        print #LCD, "<1Bh>A<0><2><0F0h>data:";msg$;
      else
        print #LCD, ;" RTR      ";
      endif
    endif

    get #CAN, #0, #UFCI_CAN_STAT, 0, can_stat ' CAN-Status
    using "UH<2><2>   0 0 0 0 2"  ' HEX-Format fuer ein Byte
    print_using #LCD, "<1Bh>A<1><1><0F0h>";can_stat;
  next
END
```

CAN-Bus Hardware-Anschlußbeispiel

In der Hardware sollte an jedem Leitungsende ein Abschlußwiderstand von 120 Ohm angebracht werden. Auf dem TCAN-Adapter befindet sich für diesen Zweck ein DIP-Schalter, der den Abschlußwiderstand zu- oder abschaltet.



Beachten Sie die Terminierung des Busses mit 120Ohm-Widerständen.

Eine kurze Einführung zu CAN

CAN ist eine Abkürzung für Controller Area Network. Ursprünglich wurde CAN als Kommunikationsprotokoll zum Informationsaustausch in Kraftfahrzeugen entwickelt. Mittlerweile ist CAN ebenso verbreitet in der Automatisierungstechnik und Haustechnik zu finden. Grundlage für den CAN-Bus ist eine Hardware, die den Anschluss an den CAN-Bus herstellt und den eigentlichen Nachrichtenversand und Nachrichtenempfang vornimmt, ähnlich einer UART bei der RS-232-Schnittstelle, jedoch bereits mit Prüfsummen, Fehlerkontrolle und Wiederholung der Nachrichten im Fehlerfalle sowie Bus-Arbitrierung und Bus-Priorisierung. Es gibt eine Vielzahl von Herstellern, die CAN-Schnittstellen auf ihren Prozessoren implementiert haben, und es gibt externe CAN-Chips, die an Prozessoren angeschlossen werden können, die keine CAN-Schnittstelle ‚on-board' haben.

Auf dem CAN-Bus werden kompakte Datenpakete versandt, im folgenden CAN-Botschaften genannt. Eine Botschaft besteht aus Anwendersicht aus einem Identifier und 0 bis 8 Datenbytes. Es gibt zwei Varianten des Bit-Protokolls auf dem Bus gemäß CAN 2.0A mit **11-Bit-Identifier** und gemäß CAN 2.0B mit **29-Bit-Identifier**. Beide Varianten existieren nebeneinander, beide haben jeweils ihre Vor- und Nachteile. Moderne Chips unterstützen entweder CAN2.0B oder akzeptieren zumindest das Vorhandensein von 29-Bit-Identifiern auf dem Bus (CAN2.0B passiv).

Buszugriffe und Zugriffs-Prioritäten sind durch die CAN-Spezifikation festgelegt und werden vollständig von der CAN-Hardware bewältigt. Die Anwendungs-Software legt die CAN-Botschaft mit einem ‚Aufkleber' in den CAN Sende-Briefkasten. Der Aufkleber, Identifier genannt, ist jedoch kein Adress-Aufkleber, sondern eine Identifikation des Inhalts der CAN-Botschaft, z.B. ‚die Temperaturinformation von Sensor A', oder ‚die Stellinformation für Druckregler X'. Jeder Busteilnehmer, für dessen Anwendung die Botschaft wichtig ist, wird darauf programmiert sein, sie aufzunehmen. Der Absender kann nicht feststellen, ob irgendein anderer Knoten die Botschaft aufgenommen hat.

Ein **Empfangsfilter** in der CAN-Hardware filtert die Botschaften nach bestimmten Kriterien vor, so dass nicht alle Botschaften zu der Anwendung gelangen. Im Empfangsteil liegen die größten Unterschiede bei den verschiedenen Implementationen von CAN-Hardware. Sowohl die Art der Filterung als auch die Anzahl der Botschaften, die im Empfangsbriefkasten gespeichert werden, sind sehr unterschiedlich. Es wird versucht, nur die Botschaften durch den Filter hindurchzulassen, welche für die Anwendung interessant sind.

Auf dem CAN-Bus können sogenannte ‚**Remote Transmission Request**s' verschickt werden. Damit werden die entsprechenden Busteilnehmer aufgefordert, mit einer spezifischen Botschaft zu antworten. So kann zum Beispiel die Aufforderung auf dem Bus erscheinen, die ‚Temperatur Kessel 2' zu melden. Die Anwendungen in den einzelnen CAN-Knoten legen fest, ob auf solche Sende-Aufforderungen geantwortet werden kann und welchen Inhalt die Antwort hat.

Die **Buszugriffe** finden in einem festen Zeitraster statt. Mit jedem Buszugriff synchronisieren sich alle Busteilnehmer. Die Zugriffe finden zur gleichen Zeit statt.

Der Ruhepegel auf dem Bus ist die ‚1'. Dieser Pegel ist der nicht dominante. Eine ‚1' kann durch eine ‚0' überschrieben werden, daher der Ausdruck ‚dominant' für die ‚0'. Ein Buszugriff wird mit einer **dominanten ‚0'** begonnen. Danach folgen die ‚1' und ‚0' Pegel des Identifiers, angefangen mit dem höchstwertigen Bit. Die niedriger priorisierten Busteilnehmer haben in den höherwertigen Bitstellen ‚1'-Bits und können daher von den priorisierteren Busteilnehmern mit einer ‚0' überschrieben werden. Sobald ein Teilnehmer bei einem Buszugriff seine ‚1' nicht platzieren konnte, bricht er den Buszugriff ab, um es später noch mal zu versuchen. Dieser erneute Versuch wird von der CAN-Hardware automatisch vorgenommen und braucht nicht in der Anwendung programmiert zu werden, die gar nichts davon weiß. Erst wenn ein Buszugriff über eine Anzahl Versuche hinweg nicht möglich ist, der Bus also scheinbar ständig von dominanteren Teilnehmern belegt ist, wird es der Anwendung möglich, über Abfrage von Fehlerregistern der CAN-Hardware diesen Zustand zu erkennen.

Hier noch einmal in einer Gegenüberstellung die prägnantesten Unterschiede zu den meisten anderen Netzen und Bussystemen:

Die meisten anderen Industrie-Bussysteme	**CAN-Bus**
Jeder Teilnehmer erhält eine Adresse und Nachrichten werden zusammen mit einer Zieladresse, manchmal auch einer Absenderadresse versehen.	Es gibt keine Adressen. Die Nachrichten sind mit einer Inhaltserklärung anstelle der Adresse versehen. Die Teilnehmer haben programmierbare Eingangsfilter, die bestimmte Nachrichten hindurchlassen.
Oft ist eine Empfangsbestätigung vorgesehen. Der Empfänger bestätigt dann den Korrekten Empfang der Sendung.	Die CAN-Hardware bestätigt am Ende eines Nachrichtenpaketes, dass dieses korrekt auf dem Bus erschienen ist (Acknowledge). Ob irgendein Teilnehmer die Nachricht auch aufgenommen hat, ist unbekannt.
Es existieren Regeln für den Buszugriff, so dass nie zwei Teilnehmer den Bus gleichzeitig benutzen.	Bei CAN können mehrere Teilnehmer gleichzeitig auf den Bus zugreifen. Im Laufe des Zugriffs verdrängt der priorisierteste Teilnehmer die anderen, die automatisch später noch mal auf den Bus zugreifen. Der Buszugriff wird vollständig von der CAN-Hardware bewältigt.

Besonderheiten des BASIC-Tiger®-CAN-Moduls

An das BASIC-Tiger®-CAN-Modul können andere Module oder Einheiten direkt angeschlossen werden, wenn es sich um kurze Entfernungen handelt, etwa auf der gleichen Platine oder zumindest im gleichen Gehäuse. Bei größeren Entfernungen ist ein externer Bustreiber erforderlich (z.B. PCA82C250).

Die CAN-Hardware wird durch den CAN-Device-Treiber unterstützt, der in Varianten vorliegt. Die Dateinamen haben folgende Bedeutung:

CAN_nn.TDD nn repräsentiert die Puffergröße
R1: 256 Byte
K1: 1 kByte
K8: 8 kByte

Bei der Installation können Parameter angegeben werden, die CAN-Botschaften-Filterungseigenschaften und das Bus-Timing festlegen. Zur Laufzeit können diese Parameter durch User-Function-Codes verändert werden.

Das BASIC-Tiger®-CAN-Modul unterstützt CAN2.0B, kann also Botschaften mit 29-Bit-Identifiern senden und empfangen. Botschaften werden in der regel zusammen mit dem Identifier in einem String vorbereitet und dann mit einer PUT- oder PRINT-Instruktion an den CAN-Treiber übergeben. Dieser puffert die Nachrichten, wenn nötig. Empfangene Botschaften werden im Empfangspuffer des Device-Treibers abgelegt, bis sie vom BASIC mit GET gelesen werden. Das spezielle Format der CAN-Botschaften ist in jedem Fall zu beachten.

Fehlersituationen

Im Folgenden werden einige Fehlersituationen aufgeführt, und es wird gezeigt, wie sie sich äußern.

Fehler-Erscheinung	Mögliche Ursache
Auf dem Scope ist zu sehen: ein Teilnehmer sendet ständig und unablässig auf dem Bus, obwohl die Anwendung nur eine einzelne Nachricht absetzen wollte.	Der sendende Teilnehmer, besser: dessen CAN-Hardware, bekommt kein Acknowledge von einem anderen Busteilnehmer. Deswegen sendet die CAN-Hardware die Nachricht immmer wieder erneut. Mögliche Gründe: Es befindet sich nur ein aktiver Teilnehmer am Bus. Die anderen sind entwerder nicht vorhanden, ausgeschaltet oder nicht initialisiert. Die Bitrate dieses Teilnehmers stimmt nicht mit der Bitrate der anderen Busteilnehmer überein.
Botschaften, die sicher gesendet werden, kommen nicht an.	Es treten Empfangsfehler auf. Lassen Sie die Fehlerregister anzeigen, um Rückschlüsse auf den Fehler ziehen zu können. Sind die Fehlerregister in Ordnung, dann könnte es sein, daß die Filter die Identifier nicht passieren lassen.
Beim Senden werden sofort die Fehlerregister gesetzt.	Möglicherweise ist der Bus durch einen höher-priorisierten Teilnehmer ständig belegt (Überlastung), oder die Bitrate ist falsch. Ist etwa kein anderer Teilnehmer aktiv? Mindestens ein Busteilnehmer muß das ACK-Bit setzen.

Literaturhinweise zu CAN

[1] Wolfgang Lawrenz: CAN Controller Area Network, Grundlagen und Praxis. Hüthig Verlag, 1994, ISBN 3-7785-2263-9

[2] Konrad Etschberger: CAN Controller Area Network, Grundlagen, Protokolle, Bausteine, Anwendungen. Verlag Hanser, 1994, ISBN 3-446-17596-2

[3] Bosch CAN Spezifikation Version 2.0 1991

[4] CiA: CAN in Automation e.V. Users Group, Am Weichselgarten 25, D-91058 Erlangen, Germany; Tel: +49 9131 601091, Fax: +49 9131 601092

[5] SJA1000 Datenbuch als PDF-datei im Internet: http://www-eu3.semiconductors.com/pip/SJA1000

[6] P82C150 CAN-SLIO Datenbuch als PDF-datei im Internet: http://www-eu3.semiconductors.com/pip/P82C150

In den Büchern finden Sie umfangreiche weitere Literaturangaben.

CAN-Board

Um die ersten Schritte in der CAN-Welt machen zu können, sind mindestens zwei, besser drei Busteilnehmer nötig. Sie haben also entweder

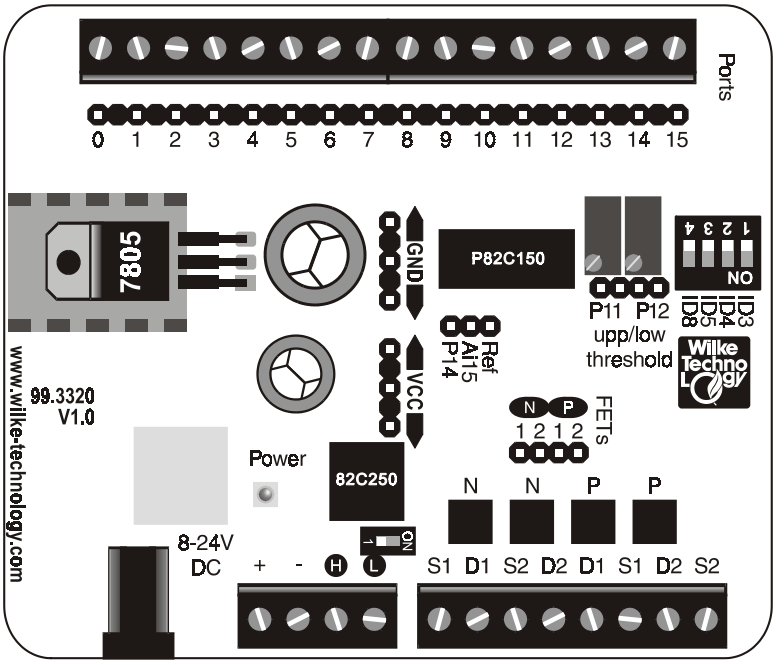
- sowieso schon Erfahrung mit CAN und andere Busteilnehmer greifbar. Sie kennen diese Geräte einigermaßen gut.
- Mindestens zwei TCAN-Module gekauft, einen Adapter für das Plug & Play Lab und eine andere Hardware-Plattform für das zweite TCAN-Modul aufbereitet.
- das TCAN-Entwicklungspaket gekauft, worin als weitere Busteilnehmer zwei Boards mit dem CAN-SLIO-Baustein von Philips enthalten sind.

In diesem Abschnitt finden Sie:

CAN-SLIO-Board

Das CAN-SLIO-Board ist eine I/O-Einheit, die über den CAN-Bus entfernt von der Steuereinheit seriell angebundenen ist. Das Kernstück ist der CAN-SLIO-Chip (SLIO=Serial-Linked-I/O), der in den folgenden Abschnitten ausführlich beschrieben ist. Das Board enthält:

- ein Netzteil mit Spannungsregler
- den CAN-Treiber-Chip
- einen zuschaltbaren Abschlußwiderstand für den CAN-Bus
- einen DIP-Schalter, um den Identifier einzustellen
- Zwei Potis, um 2 analoge Spannungen einzustellen
- eine einfache Filterschaltung für die quasi-analogen Ausgänge
- 2 N-FETs und 2 P-FETs zur beliebigen Verwendung
- Schraubklemmen für 16 digitale I/O
- Schraubklemmen für den CAN-Bus
- Schraubklemmen für die Stromversorgung, falls die Buchse nicht verwendet wird



Technische Daten des CAN-SLIO-Boards:

Außenmaße / Gewicht:	ca.100 x 85 x 27 mm / ca. 75g
Stromversorgung	8V...24V / ca. 22 mA (unbelastete I/O)
Belastbarkeit der I/O-Pins	±4mA, alle Pins zusammen < 200mW
Reset	Power-ON-Reset auf dem Board durch R-C-Glied
Temperaturbereich	-40...+85°C
A/D-Eingang	Auflösung 6...7-Bit, bis zu 6 Kanäle über Analogschalter
D/A-Ausgänge	2 DPM (distributed pulse modulated) Auflösung 10-Bit Wiederholfrequenz 1024 Bitzeiten

Um das Board in Betrieb zu nehmen, schließen Sie es an einen CAN-Bus mit einem quarzkontrollierten Teilnehmer an (Schraubanschlüsse ‚H' und ‚L'). Ein solcher Teilnehmer ist z.B. ein BASIC-Tiger®-CAN-Modul, welches auf dem CAN-Adapter des Plug & Play Labs Platz findet. Wenn sich das SLIO-Board am physikalischen Ende des Busses befindet, schlten Sie den DIP-Schalter neben der Schraubklemme auf ‚ON', um den Abschlußwiderstand zuzuschalten. Wenn sich das Board nicht am Bus-Ende befindet, schalten Sie den DIP-Schalter aus.

Verdrahten Sie weitere Hardware, die an I/O-Pins angeschlossen werden soll.

Stellen Sie an dem 4er-DIP-Schalter die gewünschten ID-Bits ein. Alle Busteilnehmer sollten einen eindeutigen Identifier haben, der nicht noch einmal im Bussystem vorkommt.

Laden Sie in das Modul TCAN-4/4 eines der Beispielprogramme. Das grundlegendste Beispielprogramm ist ‚SLIO_FIND1.TIG', welches das Finden des SLIO-Chips bewerkstelligt. Damit verbunden ist die automatische Bitratenerkennung und ständige Re-synchronisation des SLIO-Chips.

CAN-SLIO-Chip

Der CAN-SLIO-Chip P82C150 ist ein seriell angebundener I/O-Chip (SLIO=Serial-Linked-I/O), dessen Register über den CAN-Bus beschreibbar sind. Der SLIO-Chip unterstützt die Protokol-Spezifikationen 2.0A und 2.0B (passive). Abstriche bei dem Bit-Timing bestehen aufgrund der automatischen Bitraten-Erkennung, sofern diese genutzt wird. In einem System mit P82C150-Knoten mit automatischer Bitraten-Erkennung muß sich mindestens ein aktiver CAN-Knoten mit Quarz befinden.

Auf dem CAN-SLIO-Board befindet sich ein SLIO-Chip, der die Bitrate selbst erkennen muß. Dies ist möglich im Bereich von Bitzeiten von 8µsec bis ca. 50µsec. Die folgenden Beispiele sind auf eine Bitzeit von ca. 22µsec eingestellt.

Identifier des SLIO

Der CAN-SLIO verarbeitet 11-Bit-Identifier. 29-Bit-Identifier werden ignoriert. 4 Bits der Empfangsmaske werden nach dem Reset von den Portpins P0...P3 eingelesen. Diese Portpins sind danach als I/O-Pins verwendbar. Somit lassen sich bis zu 16 SLIO-Chips an einem Bus betreiben (bei Verwendung von externem Clock reduziert sich die Zahl auf 8). Jeder SLIO-Chip verwendet 2 Identifier, die sich im untersten Bit unterscheiden. Die höhere Priorität ist für Nachrichtenempfang reserviert (Bit ID.0=0).

ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3	ID.2	ID.1	ID.0
0	1	P3	1	0	P2	P1	P0	1	0	DIR

DIR	0: Nachrichten, die vom Host an den SLIO-CHIP gesendet werden (Schreibregister, Konfiguration). 1: Nachrichten, die vom SLIO-CHIP an den Host gesendet werden (Leseregister, RTR).
P0, P1, P2, P3	Programmierbare ID-Bits, die nach dem Reset von den Pins P0...P3 gelesen werden.

Bitrate automatisch erkennen

Nach dem Einschalten oder einem Reset befindet der SLIO-Chip sich im Sleep-Modus. Der interne Oszillator ist gestoppt und alle Output-Treiber sind de-aktiviert. Beim ersten dominanten Bit auf dem Bus geht er in den ‚differential mode'. Um den Chip auf dem CAN-Bus verfügbar zu machen, muß zunächst die Bitrate erkannt werden. Dazu benötigt der SLIO-Chip ein bestimmtes Bitmuster auf dem Bus. Eine bestimmte Nachricht mit festgelegtem Identifier und festgelegten Datenbytes erzeugt dieses Bitmuster. Die gleiche Nachricht wird später verwendet, um den SLIO-Chip zu re-synchronisieren und damit aktiv zu halten. Spätestens nach 8000 Bitzeiten muß erneut eine re-synchronisation erfolgen, sonst geht der SLIO in wieder in einen

inaktiven Zusatand über. Diese CAN-Nachricht wird in Tiger-BASIC® so zusammengesetzt:

```
calib_id = 0aah shl 5           ' ID der Kalibrier-Botschaften
calib$ = "<0><0><0><0aah><4>"   ' spezieller String fuer Init-SLIO
calib$ = ntos$ ( calib$, 1, -2, calib_id )' ID high-Byte zuerst
```


Der Identifier ist AAh und die Datenbytes sind AAh und 4. Da der Identifier linksbündig in einem WORD bzw. 2 Bytes stehen muß, werden die 11 Bits um 5 nach links geschoben. In Tiger-BASIC® steht in einem WORD jedoch das low-Byte zuerst, in der Message wird aber das high-Byte zuerst benötigt. Die Funktion NTOS$ dreht die Bytes beim Einbauen in den String durch die negativ angegebene Anzahl Bytes (-2). Diese Nachricht dient der Synchronisation, sie wird nicht von dem SLIO empfangen und sollte auch bei anderen Busteilnehmern nicht durch den Filter gelangen.

Die gewählte Bitrate sollte möglichst 10 Zeitsegmente enthalten oder dem nahe kommen, um die Bitratenerkennung zu erleichtern. Im Beispiel ist diese Bedingung erfüllt. Die Bitzeit sollte zwischen 8µsec und 50µsec liegen.

In den folgenden Beispielprogrammen ist der Filter der CAN-Hardware (Access-Code und Access-Mask) so eingestellt, daß möglicht nur die SLIO-Nachricht durchkommt. In einem System mit viel Datenverkehr würde dieses allererste einfache Beispielprogramm jede durchkommende Nachricht für eine SLIO-Nachricht halten.

Wenn die Bitrate erkannt ist, dann meldet sich der SLIO-Chip mit seiner ‚Sign-On'-Nachricht. Diese hat das gleiche Format wie alle SLIO-Nachrichten und meldet den Inhalt des Daten-Input-Registers.

SLIO-Nachrichtenformat

Im Identifier jeder SLIO-Nachricht ist das Richtungsbit ‚DIR' auf 1 gesetzt, die Variablen Bits sind von den Ports P0...P3 eingelesen worden. Sowohl gesendete als auch empfangene CAN-Nachrichten enthalten 3 Datenbytes. Das erste Byte enthält die Registeradresse sowie Statusinformationen, die beiden anderen Bytes den Inhalt des angegebenen Registers. Nach jeder erfolgreich versendeten Nachricht verzögert der SLIO-Chip seine nächste eventuell anstehende Nachricht um 3 Bitzeiten, um niedriger priorisierten Knoten eine Sendemöglichkeit einzuräumen. Dies ist wichtig im Falle von Wackelkontakten an einem Pin, der Flankengetriggert eine CAN-Nachricht auslöst.

Die Statusinformationen im ersten Byte werden bei empfangenen Nachrichten ignoriert. Jede empfangene Nachricht wird durch eine neue Nachricht mit dem Inhalt des adressierten Registers bestätigt. (Ausnahme: siehe Analog-Konfiguration). Wenn der SLIO-Chip eine CAN-Nachricht empfangen hat, dann sendet er als Antwort eine Nachricht, die 4 Statusbits, die zuvor empfangene Registeradresse sowie den Inhalt

des Registers enthält. (Ausnahmen siehe A/D-Konfiguration). So werden Leseregister gelesen, und bei Schreibregistern wird der Schreibvorgang bestätigt.

Statusbyte - Data-Byte 1

Status				Register-Adresse			
RSTD	EW	BM1	BM0	A3	A2	A1	A0

RSTD — 1: Sign-On-Nachricht
0: andere Nachrichten

EW — 1: ‚error warning limit' (32) ist erreicht. Bit wird gesetzt, wenn der Receive Error Counter oder der Transmit Error Counter den Wert 32 überschritten hat. Immer 1 in der ‚Sign-On'-Nachricht.

BM0, BM1 — Busmode Statusbits (siehe unten).

A0...A3 — Registeradresse. Bestimmt, welches interne SLIO-Register gelesen oder beschrieben wird.

Der Busmodus wird in den Statusbits BM0 und BM1 angezeigt:

Bus Mode	Bits		Reception Level		Transmission	
	BM1	BM0	recessive	dominant	Tx1	Tx0
0=differential	0	0	Rx0>Rx1	Rx0<Rx1	enabled	enabled
1=one wire Rx1	0	1	Rx1<REF	Rx1>REF	enabled	enabled
2=one wire Rx0	1	0	Rx0>REF	Rx0<REF	disabled	enabled
3=sleep	1	1	Rx0>REF and Rx1<REF	Rx0<REF and Rx1>REF	disabled	enabled

SLIOs auf dem Bus finden

Das Beispiel SLIO_FIND1.TIG geht davon aus, daß nur ein SLIO-Board angeschlossen ist. Es sollen erste Erkenntnisse über den SLIO gewonnen werden. Jede ankommende Nachricht wird für eine SLIO-Nachricht gehalten. Die Bytes werden in HEX so angezeigt, wie sie ankommen, so daß die oben erwähnten Bits leicht nachprüfbar sind. Die Adresse der SLIO ist an dem DIP-Schalter einstellbar. Nach dem Verstellen des DIP-Schalters muß das Board aus- und wieder eingeschaltet werden, denn der SLIO liest die ID-Bits nur beim Power-On bzw. nach einem Reset ein. Die Sign-On-Nachricht sieht z.B. so aus (alle DIP-Schalter aus):

03 50 A0 E0 80 00

03: Das Frame-Info-Byte zeigt an, daß es sich um eine Standard-Frame-Nachricht mit 3 datenbytes handelt.

50 A0: der Identifier enthält nur die festen SLIO-ID-Bits, die variablen Bits sind ‚0', da alle DIP-Schalter aus sind. Die 11 Bits sind rechtsbündig in den beiden Bytes enthalten: 0101 0000 101 – 0 0000 0000.

C0: RSTD=1 -> Sign-On-Nachricht, EW ist in Sign-On immer 1, BM=10 -> Busmodus ist ‚Sleep-Mode'. Die Registeradresse ist 0, es ist also das Data-Input-Register.

80 00: Inhalt des Data-Input-Registers. Bei offenen Pins ist der Inhalt eher zufällig.

Verstellen Sie den DIP-Schalter und Schalten Sie das SLIO-Board aus und wieder ein. Lassen Sie dann SLIO_FIND1 nochmal laufen.

Debuggen: wenn Sie SLIO-Beispielprogramme im Debugmodus per Einzelschritt untersuchen, dann geht wahrscheinlich der SLIO in den Sleep-Modus, da die Kalibrierungsnachrichten zu lange ausbleiben. Einzelschritte sollten am Besten gezielt nach einem Breakpoint durchgeführt werden. Danach muß ganz neu gestartet werden, damit der SLIO neu gefunden wird.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name:  SLIO_FIND1.TIG
' FIndet SLIO-Baustein(e) oder meldet Fehler.
' Verbinde CAN-SLIO-Board(s) mit unterschiedlichen Adressen
'--------------------------------------------------------------------
user var strict                   ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC               ' User Function Codes
#INCLUDE DEFINE_A.INC             ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                  ' CAN-Definitionen
#INCLUDE CANSLIO.INC              ' Definitionen zum CAN-SLIO-Chip

#define NOT_READY 0               ' SLIO-Status 'nicht bereit'
#define READY 1                   ' SLIO-Status 'bereit'

'                  3  210           ID-Bitpositionen
#define ID_SLIO1 00000000000b     ' ID-Bit-Einstellung dieses SLIO

BYTE slio_stat
WORD slio1_id, calib_id
LONG ac_code, ac_mask
STRING id$(4), calib$(5)
STRING slio1_dout$(6), slio1_doe$(6)    ' data out, data out enable

'--------------------------------------------------------------------
TASK MAIN
  BYTE ever
  WORD fi
  STRING id1$, c$

  install_device #LCD, "LCD1.TDD"

' warte auf SLIO-Code, aber filtere 'keep-alive'-Codes aus
' diese Bits stehen linksbuendig in den je 32-Bit Code u. Mask
' 01010000101 <- feste SLIO-Bits
' 00100111000 <- Filter: 1:don't care, 0: Bit muss stimmen

   install_device #CAN, "CAN1_K1.TDD", &
    "50 A0 00 00 &                ' access code
     27 1F FF FF &                ' access mask
     10 45 &                      ' Bitzeit 20usec (moegl.:8...50usec)
     08 1A"%                      ' single filter mode, outctrl

  calib_id = 0aah shl 5           ' ID der Kalibrier-Botschaften
  calib$ = "<0><0><0><0aah><4>"   ' spezieller String fuer Init-SLIO
  calib$ = ntos$ ( calib$, 1, -2, calib_id )' ID high-Byte zuerst

'--------------------------------------------------------------------
' Sendet Kalibrier-Botschaften und wartet auf
' (gefilterte) SLIO-Botschaften als Antwort
'
  print #LCD, "trying to find SLIO";
  slio_stat = NOT_READY
  while slio_stat = NOT_READY     ' pruefe SLIO-Status
    put #CAN, calib$              ' Kalibrierbotschaft auf den Bus
    wait_duration 50
    get #CAN, #0, #UFCI_IBU_FILL, 2, fi
    if fi > 5 then                ' wenn min 1 Botschaft da ist
      print #LCD, "<1bh>A<0><1><0f0h>"; ' Cursor setzen
```

```
        get #CAN, 1, c$
        using "UH<2><2>   0 0 0 0 2" ' und HEX anzeigen
        print_using #LCD, asc(c$);" ";
        get #CAN, #0, #UFCI_IBU_FILL, 2, fi ' mehr?
      endwhile
      print #LCD, "<1bh>A<0><2><0f0h>SLIO found";
      slio_stat = READY
    endif
  endwhile

  for ever = 0 to 0 step 0        ' Endlosschleife
  next
END
```


Für die später folgenden Beispiele ist ein etwas komplizierteres Unterprogramm ‚find_slios' geschrieben worden, welches weitere Gegebenheiten berücksichtigt:

Es können mehrer SLIOs am Bus sein.

Es könnten noch Busteilnehmer vorhanden sein, die keine SLIO sind.

Es könnte gar keine SLIO da sein, aber das Programm soll weiterlaufen.

Die SLIOs brauchen regelmäßig Kalibrier-Bitmuster auf dem CAN-Bus, sonst gehen sie wieder in den Sleep-Modus und die Ausgänge werden inaktiv.

Das Unterprogramm wartet eine Sekunde, um allen eventuell vorhandenen SLIOs die Möglichkeit zu geben, Ihre Sign-On-Nachricht zu schicken, und von der Anwendung auch berücksichtigt zu werden. Andererseits wird nicht länger gewartet, denn es könnte sein, daß sich keine SLIO meldet. Dieses Beispiel meldet diese Tastsache und beendet das Main-Programm.

Die Empfangenen Nachrichten werden unersucht, ob sie auch von einer SLIO stammen. Kein anderer Busteilnehmer darf eine der 16 möglichen SLIO-IDs haben. Andere als SLIO-Nachrichten werden in der Erkennungsphase nur aus dem Puffer gelesen und verworfen.

Die Task ‚keep_alive' sendet die Kalibrier-Nachrichten, um die SLIOs aufzuwecken und sie anschließend aktiv zu halten. Spätestens nach 8000 Bitzeiten soll eine Kalibrier-Nachricht auf dem Bus erscheinen. Kein Teilnehmer empfängt sie, nur ein für die Re-synchronisation der SLIOs notwendiges Bitmuster wird dadurch erzeugt. Während der Erkennungsphase wird die Kalibrier-Nachricht etwas öfter gesendet.

Die Task ‚keep_alive' sendet auch Nachrichten aus sofern dies nötig ist. In diesem Beispiel wird es nicht nötig, da nur die Erkennung demonstriert werden soll. Die anderen Beispiele benutzen jedoch die Ausgabe der Task ‚keep_alive' über die Variable ‚can_out$'.

Wenn Sie dieses Beispiel jetzt einzeln verstehen, können Sie in den folgenden Beispielen dieses Unterprogramm als ‚Black-Box' betrachten.

Debuggen: wenn Sie SLIO-Beispielprogramme im Debugmodus per Einzelschritt untersuchen, dann geht wahrscheinlich der SLIO in den Sleep-Modus, da die Kalibrierungsnachrichten zu lange ausbleiben. Einzelschritte sollten am Besten gezielt nach einem Breakpoint durchgeführt werden. Danach muß ganz neu gestartet werden, damit der SLIO neu gefunden wird.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name:  SLIO_FIND2.TIG
' Findet SLIO-Baustein(e) oder meldet Fehler.
' Verbinde CAN-SLIO-Board(s) mit unterschiedlichen Adressen
'--------------------------------------------------------------------
user var strict                    ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC                ' User Function Codes
#INCLUDE DEFINE_A.INC              ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                   ' CAN-Definitionen
#INCLUDE CANSLIO.INC               ' Definitionen zum CAN-SLIO-Chip

#define ALIVETIME 50               ' warte ca. 2200 * 22usec        wait
#define SLIOSLOT 10000             ' msec Zeit, sich 'bereit' zu melden

'                  3  210           ID-Bitpositionen
#define ID_SLIO1 00000000000b      ' ID-Bit-Einstellung dieses SLIO

                                   ' globale Variable
BYTE no_of_slios                   ' Zaehler SLIOs im System
WORD slio1_id, calib_id            ' ID der SLIO und der Kalib.-Botsch.
LONG ac_code, ac_mask              ' Access-Code und Access-Mask
LONG slio_equip                    ' ein Bit fuer jede SLIO
LONG alive_wait                    ' 1/2 Wartezeit in 'keep_alive'
STRING calib$(5)                   ' Kalibrierungsbotschaft
STRING can_out$(13)                ' Sendebotschaft

'--------------------------------------------------------------------
TASK MAIN
  BYTE ever                        ' Endlosschleife
  WORD fi                          ' Pufferfuellung
  STRING c$

  install_device #LCD, "LCD1.TDD" ' LCD-Treiber installieren

' warte auf SLIO-Code, aber filtere 'keep-alive'-Codes aus
' diese Bits stehen linksbuendig in den je 32-Bit Code u. Mask
' 01010000101 <- feste SLIO-Bits (50Ah)
' 00100111000 <- Filter: 1:don't care, 0: Bit muss stimmen (271h)

   install_device #CAN, "CAN1_K1.TDD", &
    "50 A0 00 00 &                 ' access code
     27 1F FF FF &                 ' access mask
     10 45 &                       ' Bitzeit 20usec (moegl.:8...50usec)
     08 1A"%                       ' single filter mode, outctrl

  call find_slios                  ' stellt Liste aller im System
                                   ' befindlichen SLIOs zusammen
  if slio_equip = 0 then           ' wenn keine SLIO gefunden
    goto no_slio_found             ' dann Programm abbrechen
  else
    print #LCD, "<1Bh>A<0><2><0F0h>";no_of_slios;" SLIOs found";
  endif

  for ever = 0 to 0 step 0         ' Endlosschleife
    print #LCD, "<1bh>A<0><3><0f0h>task main running"
  next

no_slio_found:
```

```
  print #LCD, "<1>no SLIO found"
  print #LCD, "program terminated"
END


'--------------------------------------------------------------------
' Sendet Kalibrier-Botschaften und wartet auf
' (gefilterte) SLIO-Botschaften als Antwort
' Innerhalb einer gesetzten Zeit muessen alle SLIOs sich gemeldet
' haben. Die Filterung muss im Hauptprogramm gesetzt sein.
' Nicht-SLIO-Botschaften und extended Frames werden verworfen,
' d.h. lediglich aus dem Puffer geräumt.
'--------------------------------------------------------------------
SUB find_slios ()
  BYTE ever, i                    ' Schleifenvariable
  WORD ibu_fill                   ' Eingangspufferfuellung
  BYTE frameformat, msg_len       ' Frame-Format, Botschaftslaenge
  LONG r_id                       ' Empfangs-IDreceive ID
  LONG t
  STRING msg$(13), data$(8)       ' Botschaft und Daten

  slio_equip = 0                  ' fange an mit 'keine SLIO'
  no_of_slios = 0
  calib_id = 0aah shl 5           ' ID der Kalibrier-Botschaften
  calib$ = "<0><0><0><0aah><4>"   ' spezieller String fuer Init-SLIO
  calib$ = ntos$ ( calib$, 1, -2, calib_id )' ID high-Byte zuerst
  can_out$ = ""                   ' muss initialisiert sein

  alive_wait = ALIVETIME/2        ' in der Initphase kuerzer
  run_task keep_alive             ' sendet Kalibrier-Botschaften
                                  ' und haelt SLIOs synchronisiert
  print #LCD, "trying to find SLIOs";
  t = ticks()                     ' Anfang des Zeitfensters
  while diff_ticks ( t ) < SLIOSLOT ' innerhalb des Zeitfensters
rx_cont:
    wait_duration 50
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
    if ibu_fill > 2 then                ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat      ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        r_id = r_id shr 5
                                        ' wenn keine SLIO-Botschaft
        if r_id bitand SLIO_ID_IMASK <> SLIO_FIX_ID then ' aus Puffer
          if msg_len > 0 then           ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto rx_cont                  ' warte auf naechste Botschaft
        endif
      else                              ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then             ' wenn Daten
          get #CAN, #0, msg_len, data$  ' hole sie aus dem Puffer
        endif
        goto rx_cont                    ' warte auf naechste Botschaft
      endif
'--------------------------------------------------------------------
' Hier: es war SLIO-Botschaft. Finde Adresse heraus und
```

```
      r_id = r_id bitand SLIO_ID_MASK    ' setzbare SLIO Adressbits
      if bit ( r_id, 8 ) = 1 then        ' wenn P8 gesetzt, dann
        r_id = ( r_id bitand 111000b ) + 40h
      endif
      r_id = r_id shr 3                  ' schiebe untere 3 ID-Bist weg
      set_bit slio_equip, r_id           ' setze Bitnummer in Liste
      print #LCD, "<1Bh>A<0><1><0F0h>SLIO ";r_id;" found"
      no_of_slios = no_of_slios + 1      ' zaehle die gefundene SLIO
                                         ' ignoriere hier Info in Daten
      if msg_len > 0 then                ' wenn Daten
        get #CAN, #0, msg_len, data$     ' hole sie aus dem Puffer
'        using "UH<2><2>   0.0.0.0.2"     ' Format HEX-Ausgabe
'        for i = 0 to len ( data$ ) - 1   ' alle Bytes des Strings
'          print_using #LCD, asc ( mid$ ( data$, i, 1 ) );' als HEX
'        next
      endif
    endif
  endwhile
  alive_wait = ALIVETIME               ' normale Wartezeit 'keep_alive'
END


'--------------------------------------------------------------------
' Sendet Kalibrier-Botschaften, um die SLIO synchronisiert zu halten.
' Spaetestens alle 8000 Bitzeiten muss eine Kalibrier-Botschaften
' gesendet werden.
' Wenn 'can_out$' nicht leer ist, wird es gesendet und anschliessend
' wieder auf leer gesetzt.
'--------------------------------------------------------------------
TASK keep_alive
  BYTE ever

  for ever = 0 to 0 step 0             ' Endlosschleife
    put #CAN, calib$                   ' sende Kalibrier-Botschaft
    wait_duration alive_wait           ' warte max 4000 Bitzeiten
    if can_out$ <> "" then
      put #CAN, can_out$               ' Sendebotschaft
      can_out$ = ""                    ' leer = gesendet
    endif
    wait_duration alive_wait           ' warte max 4000 Bitzeiten
  next
END
```

Einige Besonderheiten für Interressierte

Um die nicht ganz unkomplizierte Thematik in der Einführung nicht unnötig zu erschweren, wurden einige Besonderheiten nicht an Ort und Stelle erwähnt.

Remote Frames

Remote-Frames können eingesetzt werden, um entfernt über den CAN-Bus Nachrichten hervorzurufen, sprich Werte abzufragen und Informationen zu erhalten. Obwohl die SLIO auch ohne gesetztes RTR-Bit antwortet, darf das Bit natürlich gesetzt sein. Empfangene Remote Frames müssen den DLC (data length code) jedoch auf 3 gesetzt haben, sonst werden sie ignoriert. Die Antwort enthält immer die Bytes des Daten-Input-Registers.

Bit-Timing

Die nominale Bitzeit des CAN-SLIO-Chips ist unterteilt in 10 Bitsegmente. Das Synchronisations-Segment und das Propagation-Time-Segement leiten das Bit ein. Es folgt mit 4 Segmenten Phase 1, an deren Ende das Bit abgetastet wird. Phase 2 ist ebenfalls 4 Segment lang. Diese Timingstruktur ist nicht veränderbar. Der Quarz-kontrollierte Host sollte auch so eingestellt werden, daß die Bitzeit in 10 Segmente zerlegt wird.

1 Bitzeit									
BT1	BT2	BT3	BT4	BT5	BT6	BT7	BT8	BT9	BT10
Sync	Prop	phase_1				phase_2			

Oszillator und Kalibrierung

Der SLIO-Chip hat einen internen R-C-Oszillator, der automatisch durch die CAN-Nachrichten auf dem Bus kalibriert wird. Während des Starts wird jede Nachricht verwendet, um die Bitzeit einzustellen. Zur genauen Kalibrierung ist ein bestimmtes Bitmuster auf dem Bus notwendig. Beispiel einer geeigneten Kalibrier-Nachricht (| = Stuffbit, die wichtigen Bits sind unterstrichen):

SOF	**arbitration**	**control field**	**data byte 1**	**data byte 2**	**CRC field**
0	000101010100	000|010	10101010	0000|0100	000|01011100000|0

Identifier ist AAh, zwei Datenbytes mit den Werten AAh, 4 sind in der Nachricht enthalten.

Initialisierung

In der Resetphase (RST=high) sind alle Ausgänge des CAN_SLIO hochohmig. An den Pins P0...P3 werden die 4 ID-Bits eingelesen. Die anderen ID-Bits sind unveränderbar festgelegt. Gemäß der CAN-Definition dürfen nicht zwei Knoten den gleichen Identifier verwenden. Ein CAN-SLIO hat eine der 16 möglichen Bitkombinationen.

Reset-Zustand:

Status-Bits	Identifier-Bits
RSTD = 1	ID.3 gleich P0
EW = 1	ID.4 gleich P1
BM1 = 0	ID.5 gleich P2
BM0 = 0	ID.8 gleich P3

Der SLIO-Chip muss mindestens 3 Nachrichten auf dem Bus empfangen haben, bevor der Bus-Mode gewechselt wird. Die erste Botschaft wird benutzt, um die Bitzeit zu messen, deswegen sollte die Nachricht eine Sequenz ‚010101' enthalten. Die 2. und die 3. Nachricht wird trotz korrektem Empfang nicht mit einem ACK bestätigt. Nach 3 korrekt empfangenen Nachrichten sendet der SLIO-Chip seine ‚sign-on'-Nachricht.

Der CAN-SLIO-Chip wertet auch Nachrichten als korrekt empfangen, wenn diesen ein Error-Passive-Frame auf Grund des fehlenden ACK folgt. Diese Situation ist gegeben, wenn ein Host mit einem oder mehreren CAN-SLIOs zusammenarbeitet und die SLIOs noch nicht calibriert sind.

Sign-On-Message

Diese spezielle Botschaft wird einmal von jedem CAN-SLIO versandt, nachdem der Chip kalibriert wurde. Damit wird die Betriebsbereitschaft des Knotens angezeigt.

Die Sign-On-Nachricht meldet den Inhalt des data-Input-Registers und kann von anderen Nachrichten durch das gesetzte Bit RSTD unterschieden werden:

RSTD = 1: Sign-On-Nachricht
RSDT = 0: andere Nachrichten

Anmerkung: In der Sign-On-Nachricht ist das EW-Bit gesetzt. Trotzdem wird der Status und die Error-Zähler zurückgesetzt auf 0.

Registerübersicht

Register Adresse	Registername	Funktion
0	Data Input	Enthält die Input-Pegel der Pins P0 bis P15
1	Event Positive Edge	Aktiviert bei gesetztem Bit das Absetzen einer Nachricht bei positiver Flanke an den Pins P0...P15
2	Event Negative Edge	Aktiviert bei gesetztem Bit das Absetzen einer Nachricht bei negativer Flanke an den Pins P0...P15
3	Data Output	Enthält die Output-Pegel der Pins P0 bis P15
4	Output Enable	Aktiviert bei gesetztem Bit die Ausgangstreiber an den Pins P0...P15
5	Analog Configuration	Bits 0...4: ohne Funktion Bits 5,6,7: Position des Analogschalters Bits 8,9,10: Position der Monitoring-Schalter Bit 11: ohne Funktion Bits 12,13,14: Ergebnis der Analog-Komparatoren Bits 15: startet A/D-Wandlung, wenn gesetzt
6	DPM1	Distributed Pulse Modulation. Enthält den 10-Bit-Analogwert in den Bits 6...15. Quasi-Analog-Ausgang 1 am Pin P10
7	DPM2	Distributed Pulse Modulation. Enthält den 10-Bit-Analogwert in den Bits 6...15. Quasi-Analog-Ausgang 2 am Pin P4
8	A/D-Conversion	Enthält das 10-Bit-Ergebnis der A/D-Wandlung in den Bits 6...15

SLIO digitale Ein- und Ausgänge

Der SLIO-Chip stellt 16 Pins zur Verfügung, die als digitale Eingänge oder digitale Ausgänge arbeiten können. Einige Pins haben jedoch die Möglichkeit, andere Funktionen zu übernehmen, z.B. analoge Eingänge, analoge Ausänge sowie das Einlesen der ID-Bits nach dem Reset. Einige Pins sind daher auf dem CAN-SLIO-Board besonders beschaltet und nicht einfach auf die Stiftleiste geführt:

P0...P3	besitzen Pull-up-Widerstände von 47k und Pull-down-Widerstände von 4k7, um mit dem DIP-Schalter die ID-Bits einzustellen.
P4, P10	sind mögliche quasi-analoge Ausgänge und über einfache R-C-Filter geführt.
P15, P16	sind über einen 100k-Widerstand verbunden und an P15 befindet sich ein 3,3nF-Kondensator gegen Masse. Diese Beschaltung ist notwendig, um P15 als Analog-In zu verwenden.

Digitale Eingänge: Nach dem Reset oder Power-On sind alle Pins Eingänge mit ca. 500k Eingangswiderstand. Die Pegelzustände spiegeln sich im ‚Data-In'-Register (Adresse 0) wieder. Das Register ‚Output-Enable' (Adresse 4) hat für jeden Portpin ein Bit, welches auf ‚0' steht und damit den Output-Treiber de-aktiviert.

Digitale Ausgänge: Das Register ‚Output-Enable' (Adresse 4) hat für jeden Portpin ein Bit, welches auf ‚1' den Output-Treiber aktiviert. Das Register muß explizit beschrieben werden, um Ausgangspins zu aktivieren. Das Bitmutser für alle aktivierten Ausgänge wird in das Register ‚Data-Output' (Adresse 3) geschrieben.

Events: CAN-Nachrichten werden automatisch versandt, wenn in den Registern ‚Event Positive Edge' und/oder ‚Event Negative Edge' das Bit für den gewünschten Pin gesetzt ist und die dazu passende Flanke auftritt.

Das folgende Beispielprogramm setzt alle Portpins zu Ausgängen und schaltet sie abwechselnd auf high und low. Um die SLIO(s) zu finden, wird das Unterprogramm ‚**find_slios**' zusammen mit der Task ‚**keep_alive**' verwendet, so wie unter ‚SLIOs auf dem Bus finden' beschrieben. Der Identifier der zuletzt gefundenen SLIO wird verwendet. Die Ports werden zu Ausgängen, indem in dem ‚Output-Enable'-Register alle Bits auf ‚1' gesetzt werden. Dann wird in das ‚Data-Output'-Register abwechselnd 0000 und FFFFh geschrieben. Da der SLIO mit einer Bestätigungs-Nachricht antwortet, wenn er eine Nachricht empfangen hat, gibt die Task ‚**show_slio**' diese SLIO-Nachrichten und den Identifier auf dem LCD in HEX aus. Fremde Nachrichten, dazu zählen automatisch extended Frames, werden von ‚show_slio' verworfen, es wird nur der Empfangspuffer bereinigt.

Debuggen: wenn Sie SLIO-Beispielprogramme im Debugmodus per Einzelschritt untersuchen, dann geht wahrscheinlich der SLIO in den Sleep-Modus, da die Kalibrierungsnachrichten zu lange ausbleiben. Einzelschritte sollten am Besten gezielt nach einem Breakpoint durchgeführt werden. Danach muß ganz neu gestartet werden, damit der SLIO neu gefunden wird.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name:  SLIO_HIGH_LOW.TIG
' Blinkt alle Outputs des SLIO 'high-low'
' Verbinde CAN-SLIO-Board, alle DIP-Schalter aus
'--------------------------------------------------------------------
user var strict                    ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC                ' User Function Codes
#INCLUDE DEFINE_A.INC              ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                   ' CAN-Definitionen
#INCLUDE CANSLIO.INC               ' Definitionen zum CAN-SLIO-Chip

#define ALIVETIME 50               ' warte ca. 2500 * 20usec
#define SLIOSLOT 1000              ' msec Zeit, sich 'bereit' zu melden

#define NOT_READY 0                ' SLIO-Status 'nicht bereit'
#define READY 1                    ' SLIO-Status 'bereit'

'                  3  210           ID-Bitpositionen
#define ID_SLIO1 00000000000b      ' ID-Bit-Einstellung dieses SLIO

                                   ' globale Variable
BYTE no_of_slios                   ' Zaehler SLIOs im System
BYTE slio_stat                     ' SLIO status
WORD slio1_id, calib_id            ' ID der SLIO und der Kalib.-Botsch.
LONG ac_code, ac_mask              ' Access-Code und Access-Mask
LONG slio_equip                    ' ein Bit fuer jede SLIO
LONG alive_wait                    ' 1/2 Wartezeit in 'keep_alive'
STRING calib$(5)                   ' Kalibrierungsbotschaft
STRING slio1_dout$(6), slio1_doe$(6)' data out, data out enable
STRING can_out$(13)                ' Sendebotschaft

'--------------------------------------------------------------------
TASK MAIN
  BYTE ever                        ' Endlosschleife
  WORD fi                          ' Pufferfuellung

  install_device #LCD, "LCD1.TDD"

' warte auf SLIO-Code, aber filtere 'keep-alive'-Codes aus
' diese Bits stehen linksbuendig in den je 32-Bit Code u. Mask
' 01010000101 <- feste SLIO-Bits (50Ah)
' 00100111000 <- Filter: 1:don't care, 0: Bit muss stimmen (271h)

   install_device #CAN, "CAN1_K1.TDD", &
    "50 A0 00 00 &                 ' access code
     27 1F FF FF &                 ' access mask
     10 45 &                       ' Bitzeit 20usec (moegl.:8...50usec)
     08 1A"%                       ' single filter mode, outctrl

                                   ' ID dieses SLIO: 0=standard frame
```

```
'  slio1_id = (SLIO_FIX_ID bitor ID_SLIO1) shl 5 ' SLIO-Identifier
' slio1_id wird letzte gefundene SLIO
  calib_id = 0aah shl 5           ' ID der Kalibrier-Botschaften
  can_out$ = ""                   ' muss initialisiert sein

  call find_slios                 ' stellt Liste aller im System
                                  ' befindlichen SLIOs zusammen
  if slio_equip = 0 then          ' wenn keine SLIO gefunden
    goto no_slio_found            ' dann Programm abbrechen
  else
    print #LCD, "<1Bh>A<0><2><0F0h>";no_of_slios;" SLIOs found";
  endif
  wait_duration 1500
  print #LCD, "<1>";

  run_task show_slio              ' zeigt an, was die SLIO(s) senden

  slio1_doe$ = "<0><0><0><4><0FFh><0FFh>" ' SLIO_OUT_ENABLE
  slio1_doe$ = ntos$ ( slio1_doe$, 1, -2, slio1_id)'ID high-Byte zuerst
  can_out$ = slio1_doe$
  wait_duration 100
  for ever = 0 to 0 step 0        ' Endlosschleife ----------
    slio1_dout$ = "<0><0><0><3><0FFh><0FFh>" ' SLIO_DATA_OUT
    slio1_dout$ = ntos$ ( slio1_dout$, 1, -2, slio1_id )' ID high-Byte
    can_out$ = slio1_dout$
    wait_duration 100
    slio1_dout$ = "<0><0><0><3><0><0>" ' SLIO_DATA_OUT   SLIO_DATA_OUT
    slio1_dout$ = ntos$ ( slio1_dout$, 1, -2, slio1_id )' ID einbauen
    can_out$ = slio1_dout$
    wait_duration 100
  next                            ' Endlosschleife ----------

no_slio_found:                    ' keine SLIO gefunden
  stop_task keep_alive
  print #LCD, "<1>no SLIO found"
  print #LCD, "program terminated"
END


'--------------------------------------------------------------------
' Zeigt Inhalte der SLIO-Botschaften an
' Zeile 1 auf LCD: ID
' Zeile 2 auf LCD: Datenbytes in HEX
'--------------------------------------------------------------------
TASK show_slio
  BYTE ever, i                        ' Schleifenvariable
  WORD ibu_fill                       ' Eingangspufferfuellung
  BYTE frameformat, msg_len           ' Frame-Format, Botschaftslaenge
  LONG r_id                           ' Empfangs-ID
  STRING msg$(13), data$(8)           ' Botschaft und Daten

  for ever = 0 to 0 step 0            ' Endlosschleife
rx_cont:
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
    if ibu_fill > 2 then               ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat      ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
```

```
                                    ' wenn keine SLIO-Botschaft
        if r_id bitand SLIO_ID_IMASK <> SLIO_FIX_ID then ' aus Puffer
          if msg_len > 0 then       ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto rx_cont              ' warte auf naechste Botschaft
        endif
      else                          ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then         ' wenn Daten
          get #CAN, #0, msg_len, data$  ' hole sie aus dem Puffer
        endif
        goto rx_cont                ' warte auf naechste Botschaft
      endif                         ' ---------------------------
                                    ' Hier: es war SLIO-Botschaft
      using "UH<3><3>   0 0 0 0 3"      ' fuer ID Anzeige
      print_using #LCD, "<1Bh>A<0><1><0F0h>  ID:";r_id ' ID des SLIO
      print #LCD, "                     <13>DATA:";' loesche Datenanz.
      if msg_len > 0 then           ' wenn Daten
        get #CAN, #0, msg_len, data$    ' hole sie und zeige an
        using "UH<2><2>   0.0.0.0.2"    ' Format HEX-Ausgabe
        for i = 0 to len ( data$ ) - 1  ' alle Bytes des Strings
          print_using #LCD, asc ( mid$ ( data$, i, 1 ) );' als HEX
        next
      endif
    endif
  next
END


'---------------------------------------------------------------------
' Sendet Kalibrier-Botschaften und wartet auf
' (gefilterte) SLIO-Botschaften als Antwort
' Innerhalb einer gesetzten Zeit muessen alle SLIOs sich gemeldet
' haben. Die Filterung muss im Hauptprogramm gesetzt sein.
' Nicht-SLIO-Botschaften und extended Frames werden verworfen,
' d.h. lediglich aus dem Puffer geräumt.
'---------------------------------------------------------------------
SUB find_slios ()
  BYTE ever, i                  ' Schleifenvariable
  WORD ibu_fill                 ' Eingangspufferfuellung
  BYTE frameformat, msg_len     ' Frame-Format, Botschaftslaenge
  LONG r_id                     ' Empfangs-IDreceive ID
  LONG t
  STRING msg$(13), data$(8)     ' Botschaft und Daten

  slio_equip = 0                ' fange an mit 'keine SLIO'
  no_of_slios = 0
  calib_id = 0aah shl 5         ' ID der Kalibrier-Botschaften
  calib$ = "<0><0><0><0aah><4>"   ' spezieller String fuer Init-SLIO
  calib$ = ntos$ ( calib$, 1, -2, calib_id )' ID high-Byte zuerst

  alive_wait = ALIVETIME/2      ' in der Initphase kuerzer
  run_task keep_alive           ' sendet Kalibrier-Botschaften
                                ' und haelt SLIOs synchronisiert
  print #LCD, "trying to find SLIOs";
  t = ticks()                   ' Anfang des Zeitfensters
  while diff_ticks ( t ) < SLIOSLOT ' innerhalb des Zeitfensters
find_cont:
    wait_duration 50
```

```
    if ibu_fill > 2 then                ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat      ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        r_id = r_id shr 5
                                        ' wenn keine SLIO-Botschaft
        if r_id bitand SLIO_ID_IMASK <> SLIO_FIX_ID then ' aus Puffer
          if msg_len > 0 then           ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto find_cont                ' warte auf naechste Botschaft
        endif
      else                              ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then             ' wenn Daten
          get #CAN, #0, msg_len, data$  ' hole sie aus dem Puffer
        endif
        goto find_cont                  ' warte auf naechste Botschaft
      endif
'-------------------------------------------------------------------
' Hier: es war SLIO-Botschaft. Finde Adresse heraus und
' trage in Bit-Liste ein, verwende die Adresse im Testprogramm

      slio1_id = ( r_id bitand 7FEh) shl 5 ' nehme diese Adress f. Test
                                        ' maskiere DIR-Bit weg
      r_id = r_id bitand SLIO_ID_MASK   ' setzbare SLIO Adressbits
      if bit ( r_id, 8 ) = 1 then       ' wenn P8 gesetzt, dann
        r_id = ( r_id bitand 111000b ) + 40h
      endif
      r_id = r_id shr 3                 ' schiebe untere 3 ID-Bist weg
      set_bit slio_equip, r_id          ' setze Bitnummer in Liste
      print #LCD, "<1Bh>A<0><1><0F0h>SLIO ";r_id;" found"
      no_of_slios = no_of_slios + 1     ' zaehle die gefundene SLIO
                                        ' ignoriere hier Info in Daten
      if msg_len > 0 then               ' wenn Daten
        get #CAN, #0, msg_len, data$    ' hole sie aus dem Puffer
'        using "UH<2><2>   0.0.0.0.2"    ' Format HEX-Ausgabe
'        for i = 0 to len ( data$ ) - 1  ' alle Bytes des Strings
'          print_using #LCD, asc ( mid$ ( data$, i, 1 ) );' als HEX
'        next
      endif
    endif
  endwhile
  alive_wait = ALIVETIME               ' normale Wartezeit 'keep_alive'
END


'-------------------------------------------------------------------
' Sendet Kalibrier-Botschaften, um die SLIO synchronisiert zu halten.
' Spaetestens alle 8000 Bitzeiten muss eine Kalibrier-Botschaften
' gesendet werden.
'-------------------------------------------------------------------
TASK keep_alive
  BYTE ever

  for ever = 0 to 0 step 0             ' Endlosschleife
    put #CAN, calib$                   ' sende Kalibrier-Botschaft
    wait_duration alive_wait           ' warte max 4000 Bitzeiten
```

```
      put #CAN, can_out$                ' Sendebotschaft
      can_out$ = ""                     ' leer = gesendet
    endif
    wait_duration alive_wait            ' warte max 4000 Bitzeiten
  next
END
```

SLIO-Analog-Ausgänge

Der SLIO-Chip stellt zwei quasi-analoge Ausgänge zur Verfügung. Auf dem CAN-SLIO-Board sind diese Ausgänge über einfache R-C-Filter geführt.

Der analoge Wert wird erzeugt, indem über einen festen Zeitraum mehr oder weniger ‚1'-Bits generiert werden (DMP=Distributed Pulse Modulation). Integriert man die ‚1'-Bits erhält man für wenige ‚1'-Bits pro Zeiteinheit eine niedrige Spannung, für viele ‚1'-Bits pro Zeiteinheit eine hohe Spannung (0...5V). Bei der Integration muß ein Kompromiß geschlossen werden zwischen Reaktionsgeschwindigkeit der analogen Spannung und Welligkeit. Die Erzeugung eines Analogwertes ist nach 1024 Bitzeiten abgeschlossen und wird anschließend wiederholt. Grundlage für die Anzahl der ‚1'-Bits ist der Inhalt des Registers ‚DPM1' für Pin P10 oder ‚DMP2' für Pin P4. Im Register befindet sich in einem WORD linksbündig der 10-Bit-Wert für die DPM-Ausgabe.

Die Ausgangstreiber der Pins, die zur Analogausgabe verwendet werden, müssen im Register ‚Output-Enable' aktiviert werden.

Das Beispielprogramm erzeugt auf dem Pin P10 eine Sägezahnkurve, indem aufsteigende Analogwerte ausgegeben werden. Die SLIO(s) werden mit Hilfe des Unterprogrammes ‚**find_slios**' zusammen mit der Task ‚**keep_alive**' gefunden, so wie unter ‚SLIOs auf dem Bus finden' beschrieben. Danach übernimmt die Hauptschleife des Programms die Aufgabe, die SLIOs zu re-kalibrieren, da hier sowieso eine schnelle und konstant sich wiederholende Ausgabe stattfindet. Die Task ‚**show_slio**' zeigt die ankommenden SLIO-Nachrichten und den Identifier auf dem LCD in HEX zur Kontrolle an. Fremde Nachrichten, dazu zählen automatisch extended Frames, werden von ‚show_slio' verworfen, es wird nur der Empfangspuffer bereinigt.

Debuggen: wenn Sie SLIO-Beispielprogramme im Debugmodus per Einzelschritt untersuchen, dann geht wahrscheinlich der SLIO in den Sleep-Modus, da die Kalibrierungsnachrichten zu lange ausbleiben. Einzelschritte sollten am Besten gezielt nach einem Breakpoint durchgeführt werden. Danach muß ganz neu gestartet werden, damit der SLIO neu gefunden wird.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name:  SLIO_DPM1.TIG
' erzeugt DPM-Outputs des SLIO (digital->analog)
' Verbinde CAN-SLIO-Board, alle DIP-Schalter aus
' messe am Pin P10
'--------------------------------------------------------------------
user var strict                   ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC               ' User Function Codes
#INCLUDE DEFINE_A.INC             ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                  ' CAN-Definitionen
#INCLUDE CANSLIO.INC              ' Definitionen zum CAN-SLIO-Chip

#define ALIVETIME 50              ' warte ca. 2500 * 20usec
#define SLIOSLOT 1000             ' msec Zeit, sich 'bereit' zu melden

#define NOT_READY 0               ' SLIO-Status 'nicht bereit'
#define READY 1                   ' SLIO-Status 'bereit'

'                  3  210          ID-Bitpositionen
#define ID_SLIO1 00000000000b     ' ID-Bit-Einstellung dieses SLIO

BYTE no_of_slios                  ' Zaehler SLIOs im System
BYTE slio_stat                    ' SLIO status
WORD slio1_id, calib_id           ' ID der SLIO und der Kalib.-Botsch.
LONG ac_code, ac_mask             ' Access-Code und Access-Mask
LONG slio_equip                   ' ein Bit fuer jede SLIO
LONG alive_wait                   ' 1/2 Wartezeit in 'keep_alive'
STRING calib$(5)                  ' Kalibrierungsbotschaft
STRING slio1_dpm1$(6), slio1_doe$(6)' DPM1 out, data out enable
STRING slio1_dout$
STRING can_out$(13)               ' Sendebotschaft

'--------------------------------------------------------------------
TASK MAIN
  BYTE ever                       ' Endlosschleife
  WORD fi                         ' Pufferfuellung
  WORD value, tmp                 ' Analogwert
  LONG t                          ' 'keep_alive' Zeit
  STRING tmp$(6)

  install_device #LCD, "LCD1.TDD" ' LCD-Treiber installieren

' warte auf SLIO-Code, aber filtere 'keep-alive'-Codes aus
' diese Bits stehen linksbuendig in den je 32-Bit Code u. Mask
' 01010000101 <- feste SLIO-Bits (50Ah)
' 00100111000 <- Filter: 1:don't care, 0: Bit muss stimmen (271h)

   install_device #CAN, "CAN1_K1.TDD", &
    "50 A0 00 00 &                ' access code
     27 1F FF FF &                ' access mask
     10 45 &                      ' Bitzeit 20usec (moegl.:8...50usec)
     08 1A"%                      ' single filter mode, outctrl

                                  ' ID dieses SLIO: 0=standard frame
                                  ' + 2 ID-Bytes um 5 geshiftet
  slio1_id = (SLIO_FIX_ID bitor ID_SLIO1) shl 5 ' SLIO-Identifier
  calib_id = 0aah shl 5           ' ID der Kalibrier-Botschaften
  can_out$ = ""                   ' muss initialisiert sein
```

```
  call find_slios                 ' stellt Liste aller im System
                                  ' befindlichen SLIOs zusammen
  if slio_equip = 0 then          ' wenn keine SLIO gefunden
    goto no_slio_found            ' dann Programm abbrechen
  else
    print #LCD, "<1Bh>A<0><2><0F0h>";no_of_slios;" SLIOs found";
  endif
  wait_duration 1500
  print #LCD, "<1>";

  run_task show_slio              ' zeigt an, was die SLIO(s) senden

  slio1_doe$ = "<0><0><0><4><0FFh><0FFh>" ' SLIO_OUT_ENABLE
  slio1_doe$ = ntos$ ( slio1_doe$, 1, -2, slio1_id)'ID high-Byte zuerst
  can_out$ = slio1_doe$
  wait_duration 1000

  slio1_dout$ = "<0><0><0><3><0FFh><0FFh>" ' SLIO_DATA_OUT
  slio1_dout$ = ntos$ ( slio1_dout$, 1, -2, slio1_id )' ID high-Byte
  can_out$ = slio1_dout$
  wait_duration 1000
  slio1_dout$ = "<0><0><0><3><0><0>" ' SLIO_DATA_OUT   SLIO_DATA_OUT
  slio1_dout$ = ntos$ ( slio1_dout$, 1, -2, slio1_id )' ID einbauen
  can_out$ = slio1_dout$
  wait_duration 1000

  slio1_dpm1$ = "<0><0><0><6><0><0>" ' SLIO_DPM1
  slio1_dpm1$ = ntos$ ( slio1_dpm1$, 1, -2, slio1_id )' ID high-Byte
  value = 0
  t = ticks()
  stop_task keep_alive            ' Hauptschleife uebernimmt das jetzt
  for ever = 0 to 0 step 0        ' Endlosschleife ----------
    tmp = value shl 5             ' 10-Bit rechtbuendig in WORD
    tmp$ = ntos$ ( slio1_dpm1$, 4, -2, tmp )' value high-Byte zuerst
    can_out$ = tmp$
    put #CAN, tmp$
    value = modulo_inc ( value, 0, 7FFh, 7Fh ) ' inc step 7Fh
    if diff_ticks ( t ) > 50 then      ' Wert haengt von Bitzeit ab
      put #CAN, calib$                 ' sende Kalibrier-Botschaft
      t = ticks()
    endif
  next                            ' Endlosschleife ----------

no_slio_found:                    ' keine SLIO gefunden
  stop_task keep_alive
  print #LCD, "<1>no SLIO found"
  print #LCD, "program terminated"
END


'--------------------------------------------------------------------
' Zeigt Inhalte der SLIO-Botschaften an
' Zeile 1 auf LCD: ID
' Zeile 2 auf LCD: Datenbytes in HEX
'--------------------------------------------------------------------
TASK show_slio
  BYTE ever, i                        ' Schleifenvariable
  WORD ibu_fill                       ' Eingangspufferfuellung
  BYTE frameformat, msg_len           ' Frame-Format, Botschaftslaenge
  LONG r_id                           ' Empfangs-IDreceive ID
```

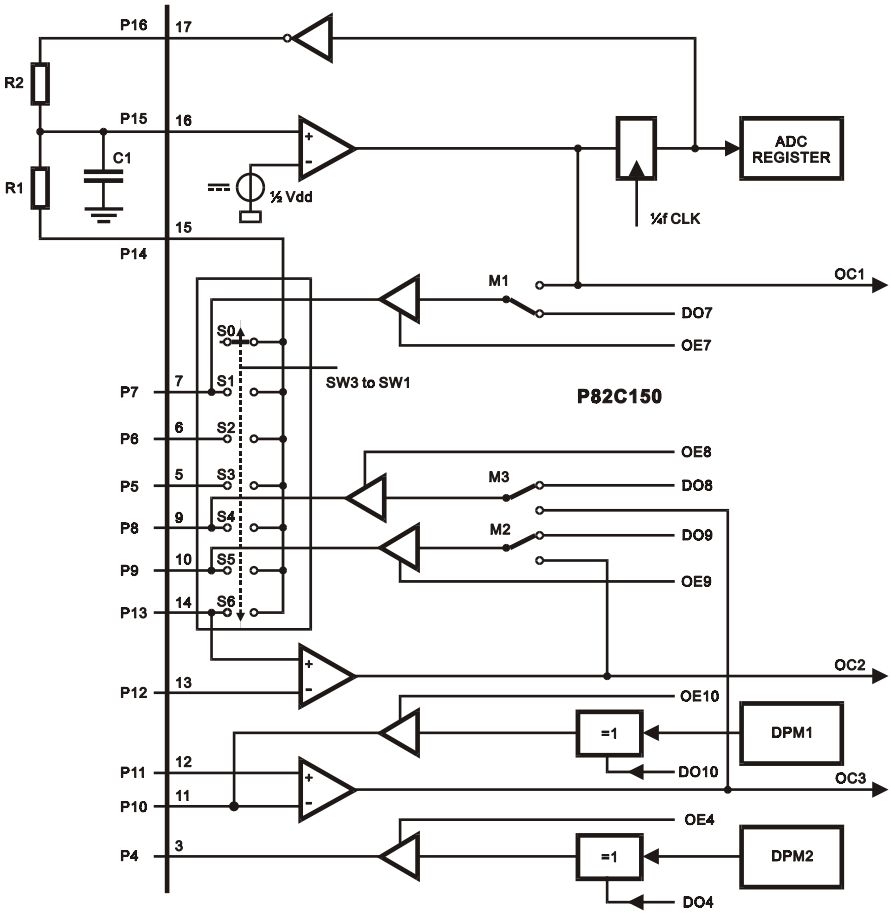
```
  for ever = 0 to 0 step 0              ' Endlosschleife
rx_cont:
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
    if ibu_fill > 2 then                ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat      ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        r_id = r_id shr 5
                                        ' wenn keine SLIO-Botschaft
        if r_id bitand SLIO_ID_IMASK <> SLIO_FIX_ID then ' aus Puffer
          if msg_len > 0 then           ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto rx_cont                  ' warte auf naechste Botschaft
        endif
      else                              ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then             ' wenn Daten
          get #CAN, #0, msg_len, data$  ' hole sie aus dem Puffer
        endif
        goto rx_cont                    ' warte auf naechste Botschaft
      endif                             ' ----------------------------
                                        ' Hier: es war SLIO-Botschaft
      using "UH<3><3>   0 0 0 0 3"      ' fuer ID Anzeige
      print_using #LCD, "<1Bh>A<0><1><0F0h>  ID:";r_id ' ID des SLIO
      print #LCD, "                     <13>DATA:";' loesche Datenanz.
      if msg_len > 0 then               ' wenn Daten
        get #CAN, #0, msg_len, data$    ' hole sie und zeige an
        using "UH<2><2>   0.0.0.0.2"    ' Format HEX-Ausgabe
        for i = 0 to len ( data$ ) - 1  ' alle Bytes des Strings
          print_using #LCD, asc ( mid$ ( data$, i, 1 ) );' als HEX
        next
      endif
    endif
  next
END


'--------------------------------------------------------------------
' Sendet Kalibrier-Botschaften und wartet auf
' (gefilterte) SLIO-Botschaften als Antwort
' Innerhalb einer gesetzten Zeit muessen alle SLIOs sich gemeldet
' haben. Die Filterung muss im Hauptprogramm gesetzt sein.
' Nicht-SLIO-Botschaften und extended Frames werden verworfen,
' d.h. lediglich aus dem Puffer geräumt.
'--------------------------------------------------------------------
SUB find_slios ()
  BYTE ever, i                    ' Schleifenvariable
  WORD ibu_fill                   ' Eingangspufferfuellung
  BYTE frameformat, msg_len       ' Frame-Format, Botschaftslaenge
  LONG r_id                       ' Empfangs-IDreceive ID
  LONG t
  STRING msg$(13), data$(8)       ' Botschaft und Daten

  slio_equip = 0                  ' fange an mit 'keine SLIO'
  no_of_slios = 0
  calib_id = 0aah shl 5           ' ID der Kalibrier-Botschaften
```

```
  calib$ = ntos$ ( calib$, 1, -2, calib_id )' ID high-Byte zuerst

  alive_wait = ALIVETIME/2        ' in der Initphase kuerzer
  run_task keep_alive             ' sendet Kalibrier-Botschaften
                                  ' und haelt SLIOs synchronisiert
  print #LCD, "trying to find SLIOs";
  t = ticks()                     ' Anfang des Zeitfensters
  while diff_ticks ( t ) < SLIOSLOT ' innerhalb des Zeitfensters
find_cont:
    wait_duration 50
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
    if ibu_fill > 2 then                ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat      ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        r_id = r_id shr 5
                                        ' wenn keine SLIO-Botschaft
        if r_id bitand SLIO_ID_IMASK <> SLIO_FIX_ID then ' aus Puffer
          if msg_len > 0 then           ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto find_cont                ' warte auf naechste Botschaft
        endif
      else                              ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then             ' wenn Daten
          get #CAN, #0, msg_len, data$  ' hole sie aus dem Puffer
        endif
        goto find_cont                  ' warte auf naechste Botschaft
      endif
'-------------------------------------------------------------------
' Hier: es war SLIO-Botschaft. Finde Adresse heraus und
' trage in Bit-Liste ein, verwende die Adresse im Testprogramm

      slio1_id = ( r_id bitand 7FEh) shl 5 ' nehme diese Adress f. Test
                                        ' maskiere DIR-Bit weg
      r_id = r_id bitand SLIO_ID_MASK   ' setzbare SLIO Adressbits
      if bit ( r_id, 8 ) = 1 then       ' wenn P8 gesetzt, dann
        r_id = ( r_id bitand 111000b ) + 40h ' erzeuge 4-Bit-Adr.
      endif
      r_id = r_id shr 3                 ' schiebe untere 3 ID-Bist weg
      set_bit slio_equip, r_id          ' setze Bitnummer in Liste
      print #LCD, "<1Bh>A<0><1><0F0h>SLIO ";r_id;" found"
      no_of_slios = no_of_slios + 1     ' zaehle die gefundene SLIO
                                        ' ignoriere hier Info in Daten
      if msg_len > 0 then               ' wenn Daten
        get #CAN, #0, msg_len, data$    ' hole sie aus dem Puffer
'        using "UH<2><2>   0.0.0.0.2"    ' Format HEX-Ausgabe
'        for i = 0 to len ( data$ ) - 1  ' alle Bytes des Strings
'          print_using #LCD, asc ( mid$ ( data$, i, 1 ) );' als HEX
'        next
      endif
    endif
  endwhile
  alive_wait = ALIVETIME               ' normale Wartezeit 'keep_alive'
END
```

```
' Sendet Kalibrier-Botschaften, um die SLIO synchronisiert zu halten.
' Spaetestens alle 8000 Bitzeiten muss eine Kalibrier-Botschaften
' gesendet werden.
'-------------------------------------------------------------------
TASK keep_alive
  BYTE ever
  LONG t

  t = ticks()
  for ever = 0 to 0 step 0              ' Endlosschleife
    if diff_ticks ( t ) > 50 then       ' Wert haengt von Bitzeit ab
      put #CAN, calib$                  ' sende Kalibrier-Botschaft
      t = ticks()
    endif
    if can_out$ <> "" then
      put #CAN, can_out$                ' Sendebotschaft
      can_out$ = ""                     ' leer = gesendet
    endif
  next
END
```

Analog-Konfiguration

Der SLIO-Chip stellt einen recht komplexen Aufbau zur Erfassung von analogen Eingangssignalen zur Verfügung. Es wird eine Auflösung von 7..8 Bit erreicht. Die Wandelrate ist genauso hoch wie die Ausgaberate bei der quasi-analogen Ausgabe. Eine Wandlung benötigt 1024 Bitzeiten.



Der A/D-Wandler im oberen Teil der Abbildung verwendet den Pin P15 als Analog-Eingang sowie P16 als Feedback-Ausgang. Die Eingangsbeschaltung besteht aus R1 und R2 (je 100k) und C1 (3,3nF). Auf dem CAN-SLIO-Board steht am Pin Ai15 ein solcher Eingang zur Verfügung. Der Pin P15 muß offen bleiben, wenn Ai15 verwendet wird.

Wird mehr als nur ein Analogeingang benötigt, dann können die Pins P5, P6 P7, P8, P9 und P13 über einen Analogschalter auf den Analogeingang geschaltet werden. Der Ausgang des Analogschalters P14 wird dazu auf den Pin Ai15 gebrückt.

Anmerkung: der Pin Vref auf der gleichen 3-poligen Stiftleiste hat nichts mit dem Analogteil des SLIO zu tun. Vref gehört zu der CAN-Schnittstelle.

Eine weitere Möglichkeit des Analogteils besteht darin, mit Hilfe von Komparatoren nur das Über- oder Unterschreiten eines Schwellwertes anzuzeigen und, wenn gewünscht, auch gleichzeitig auf einem Ausgangspin auszugeben. Der Aufbau kann so konfiguriert werden, daß eine CAN-Nachricht erzeugt wird, wenn der Schwellwert über- oder unterschritten wird, oder daß eine selbstätige Regelung läuft, ohne daß CAN-Nachrichten erzeugt werden. Folgende Komparatoren stehen zur Verfügung:

Eingangspins	Ausgang	schaltbar auf Ausgangspin	Bemerkung
P15(+)	OC1	P7	(-) an interner Vref=1/2 VCC (2.5V) Verwendet den Eingang des ADC
P12(+), P13(-)	OC2	P8	
P11(+), P10(-)	OC3	P9	P10 ist auch analog-Ausgang DMP1, der dann nicht mehr verfügbar ist.

Die Ausgänge der Komparatoren können im Register ‚Analog Configuration' gelesen werden (Bits 12, 13 und 14). Ob diese Signale auf Ausgänge durchgeschaltet werden, wird in den Bits 8, 9 und 10 des Registers ‚Analog Configuration' eingestellt. Außerdem müssen dazu natürlich die Ausgangstreiber der betroffenen Pins aktiviert sein. CAN-Nachrichten werden dann automatisch erzeugt, wenn in den Registern ‚Event Positive Edge' und/oder ‚Event Negative Edge' die entsprechenden Bits für P8, P9 und P10 gesetzt sind.

Das ‚Analog Configuration'-Register (Adresse 5):

ADC	OC3	OC2	OC1	0	M3	M2	M1	S3	S2	S1	0	0	0	0	0

Bit(s)			Funktion
0...4			ohne Funktion
S3	S2	S1	Position des Analogschalters
0	0	0	kein Schalter geschlossen
0	0	1	P7 an P14
0	1	0	P6 an P14
0	1	1	P5 an P14
1	0	0	P8 an P14
1	0	1	P9 an P14
1	1	0	P13 an P14
1	1	1	reserviert
			Position der Monitoringschalter
M1			1: OC1 an P7
M2			1: OC3 an P9
M3			1: OC2 an P8
11			ohne Funktion
			Ergebnis der Analog-Komparatoren
OC1			1: P15 > Vref-intern (2.5V)
OC2			1: P13 > P12
OC3			1: P11 > P10
ADC			1: startet A/D-Wandlung

Es ergeben sich einige Regeln (in jedem Fall Kurzschlußgefahr):

- Wenn der Analogschalter gesetzt ist, darf P14 nicht Ausgang sein.
- Wenn M1 gesetzt ist, d.h. OC1 auf P7 geschaltet ist, darf P7 kein Ausgang sein.
- Wenn M2 gesetzt ist, d.h. OC3 auf P9 geschaltet ist, darf P9 kein Ausgang sein.
- Wenn M3 gesetzt ist, d.h. OC2 auf P8 geschaltet ist, darf P8 kein Ausgang sein.

Starten der A/D-Wandlung

Eine Wandlung benötigt 1024 Bitzeiten und liefert eine Genauigkeit von 7...8 Bit.

Der A/D-Wandler wird gestartet, wenn

- das Register ‚A/D-Conversion' beschrieben wird. Die Antwort erfolgt nach der Conversion automatisch.
- das Register ‚Analog Configuration' mit gesetztem Bit ‚ADC' beschrieben wird. Die Antwort erfolgt nach der Conversion automatisch.

Das Beispielprogramm liest auf dem Pin P15 gemessene Analogwerte. Die SLIO(s) werden mit Hilfe des Unterprogrammes ‚**find_slios**' zusammen mit der Task ‚**keep_alive**' gefunden, so wie unter ‚SLIOs auf dem Bus finden' beschrieben. Die Task ‚**show_slio**' zeigt die ankommenden SLIO-Nachrichten und den Identifier auf dem LCD in HEX zur Kontrolle an. Fremde Nachrichten, dazu zählen automatisch extended Frames, werden von ‚show_slio' verworfen, es wird nur der Empfangspuffer bereinigt.

Debuggen: wenn Sie SLIO-Beispielprogramme im Debugmodus per Einzelschritt untersuchen, dann geht wahrscheinlich der SLIO in den Sleep-Modus, da die Kalibrierungsnachrichten zu lange ausbleiben. Einzelschritte sollten am Besten gezielt nach einem Breakpoint durchgeführt werden. Danach muß ganz neu gestartet werden, damit der SLIO neu gefunden wird.

Programmbeispiel:

```
'---------------------------------------------------------------------
' Name:  SLIO_ANA1.TIG
' Liest Analogeingang des SLIO an P15 und zeigt den Wert als
' HEX-Zahl an (0...7FFh).
' Verbinde CAN-SLIO-Board
'---------------------------------------------------------------------
user var strict                   ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC               ' User Function Codes
#INCLUDE DEFINE_A.INC             ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                  ' CAN-Definitionen
#INCLUDE CANSLIO.INC              ' Definitionen zum CAN-SLIO-Chip

#define ALIVETIME 50              ' warte ca. 2500 * 20usec
#define SLIOSLOT 1000             ' msec Zeit, sich 'bereit' zu melden

#define NOT_READY 0               ' SLIO-Status 'nicht bereit'
#define READY 1                   ' SLIO-Status 'bereit'

'                  3  210          ID-Bitpositionen
#define ID_SLIO1 00000000000b     ' ID-Bit-Einstellung dieses SLIO

                                  ' globale Variable
BYTE no_of_slios                  ' Zaehler SLIOs im System
BYTE slio_stat                    ' SLIO status
WORD slio1_id, calib_id           ' ID der SLIO und der Kalib.-Botsch.
LONG ac_code, ac_mask             ' Access-Code und Access-Mask
LONG slio_equip                   ' ein Bit fuer jede SLIO
LONG alive_wait                   ' 1/2 Wartezeit in 'keep_alive'
WORD acfg                         ' Analog-Konfigurationsbits
STRING calib$(5)                  ' Kalibrierungsbotschaft
STRING slio1_doe$(6)              ' data out enable
STRING slio1_acfg$(6), slio1_ain$(6)' analog config, analog in
STRING can_out$(13k)               ' Sendebotschaft zur SLIO

'---------------------------------------------------------------------
TASK MAIN
  BYTE ever                       ' Endlosschleife
  WORD fi                         ' Pufferfuellung
  STRING tmp$(6)

  install_device #LCD, "LCD1.TDD"

' warte auf SLIO-Code, aber filtere 'keep-alive'-Codes aus
' diese Bits stehen linksbuendig in den je 32-Bit Code u. Mask
' 01010000101 <- feste SLIO-Bits (50Ah)
' 00100111000 <- Filter: 1:don't care, 0: Bit muss stimmen (271h)

   install_device #CAN, "CAN1_K1.TDD", &
    "50 A0 00 00 &                ' access code
     27 1F FF FF &                ' access mask
     10 45 &                      ' Bitzeit 20usec (moegl.:8...50usec)
     08 1A"%                      ' single filter mode, outctrl

                                  ' ID dieses SLIO: 0=standard frame
                                  ' + 2 ID-Bytes um 5 geshiftet
  slio1_id = (SLIO_FIX_ID bitor ID_SLIO1) shl 5 ' SLIO-Identifier
  calib_id = 0aah shl 5           ' ID der Kalibrier-Botschaften
  can_out$ = ""                   ' muss initialisiert sein
```

```
  call find_slios                 ' stellt Liste aller im System
                                  ' befindlichen SLIOs zusammen
  if slio_equip = 0 then          ' wenn keine SLIO gefunden
    goto no_slio_found            ' dann Programm abbrechen
  else
    print #LCD, "<1Bh>A<0><2><0F0h>";no_of_slios;" SLIOs found";
  endif
  wait_duration 1500
  print #LCD, "<1>";

  run_task show_slio              ' zeigt an, was die SLIO(s) senden

  slio1_doe$ = "<0><0><0><4><0><0>" ' SLIO_OUT_ENABLE alles Inputs
  slio1_doe$ = ntos$ ( slio1_doe$, 1, -2, slio1_id)'ID high-Byte zuerst
  can_out$ = slio1_doe$           ' ausgeben (lassen)
  wait_duration 100
                                  ' bereite die Analogkonfiguration vor
  slio1_acfg$ = "<0><0><0><5><0><0>" ' SLIO_OUT_ENABLE alles Inputs
  slio1_acfg$ = ntos$ ( slio1_acfg$, 1, -2, slio1_id)'ID high-Byte erst
'        aooo-mmmsss-----
  acfg = 1000000000000000b        ' nur 'start conversion bit' gesetzt
                                  ' ooo = 0, mmm = 0: kein Comparator
                                  ' sss = 0, kein Analogschalter
  for ever = 0 to 0 step 0        ' Endlosschleife ----------
    tmp$ = ntos$ ( slio1_acfg$, 4, -2, acfg)' Konfig.wort einfuegen
    can_out$ = tmp$               ' ausgeben (lassen), startete A/D
    wait_duration 1000
  next                            ' Endlosschleife ----------

no_slio_found:                    ' keine SLIO gefunden
  stop_task keep_alive
  print #LCD, "<1>no SLIO found"
  print #LCD, "program terminated"
END


'--------------------------------------------------------------------
' Zeigt Inhalte der SLIO-Botschaften an
' Zeile 1 auf LCD: ID
' Zeile 2 auf LCD: Datenbytes in HEX
'--------------------------------------------------------------------
TASK show_slio
  BYTE ever, i                    ' Schleifenvariable
  BYTE frameformat, msg_len       ' Frame-Format, Botschaftslaenge
  WORD ibu_fill                   ' Eingangspufferfuellung
  WORD value                      ' Analogwert
  LONG r_id                       ' Empfangs-ID
  STRING msg$(13), data$(8)       ' Botschaft und Daten

  for ever = 0 to 0 step 0        ' Endlosschleife
rx_cont:
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
    if ibu_fill > 2 then              ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat    ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        r_id = r_id shr 5
                                      ' wenn keine SLIO-Botschaft
```

```
          if msg_len > 0 then          ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto rx_cont                 ' warte auf naechste Botschaft
        endif
      else                             ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then            ' wenn Daten
          get #CAN, #0, msg_len, data$ ' hole sie aus dem Puffer
        endif
        goto rx_cont                   ' warte auf naechste Botschaft
      endif                            ' ----------------------------
                                       ' Hier: es war SLIO-Botschaft
      using "UH<3><3>   0 0 0 0 3"     ' fuer ID Anzeige
      print_using #LCD, "<1Bh>A<0><1><0F0h>  ID:";r_id ' ID des SLIO
      if msg_len = 3 then              ' wenn Daten
        get #CAN, #0, msg_len, data$   ' hole sie und zeige an
        if nfroms ( data$, 0, 1 ) = 8 then ' wenn Analogregister if anal
          value = nfroms ( data$, 1, 2 )
          value = byte_mirr ( value, 2 ) shr 5
          using "UH<3><3>   0.0.0.0.3"   ' Format HEX-Ausgabe
          print_using #LCD, " P15:";value;
        endif
      endif ' msg_len
    endif ' ibu_fill
  next
END


'---------------------------------------------------------------------
' Sendet Kalibrier-Botschaften und wartet auf
' (gefilterte) SLIO-Botschaften als Antwort
' Innerhalb einer gesetzten Zeit muessen alle SLIOs sich gemeldet
' haben. Die Filterung muss im Hauptprogramm gesetzt sein.
' Nicht-SLIO-Botschaften und extended Frames werden verworfen,
' d.h. lediglich aus dem Puffer geräumt.
'---------------------------------------------------------------------
SUB find_slios ()
  BYTE ever, i                     ' Schleifenvariable
  WORD ibu_fill                    ' Eingangspufferfuellung
  BYTE frameformat, msg_len        ' Frame-Format, Botschaftslaenge
  LONG r_id                        ' Empfangs-IDreceive ID
  LONG t
  STRING msg$(13), data$(8)        ' Botschaft und Daten

  slio_equip = 0                   ' fange an mit 'keine SLIO'
  no_of_slios = 0
  calib_id = 0aah shl 5            ' ID der Kalibrier-Botschaften
  calib$ = "<0><0><0><0aah><4>"    ' spezieller String fuer Init-SLIO
  calib$ = ntos$ ( calib$, 1, -2, calib_id )' ID high-Byte zuerst

  alive_wait = ALIVETIME/2         ' in der Initphase kuerzer
  run_task keep_alive              ' sendet Kalibrier-Botschaften
                                   ' und haelt SLIOs synchronisiert
  print #LCD, "trying to find SLIOs";
  t = ticks()                      ' Anfang des Zeitfensters
  while diff_ticks ( t ) < SLIOSLOT ' innerhalb des Zeitfensters
find_cont:
    wait_duration 50
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
```

```
      get #CAN, #0, 1, frameformat      ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        r_id = r_id shr 5
                                         ' wenn keine SLIO-Botschaft
        if r_id bitand SLIO_ID_IMASK <> SLIO_FIX_ID then ' aus Puffer
          if msg_len > 0 then            ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto find_cont                 ' warte auf naechste Botschaft
        endif
      else                               ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then              ' wenn Daten
          get #CAN, #0, msg_len, data$  ' hole sie aus dem Puffer
        endif
        goto find_cont                   ' warte auf naechste Botschaft
      endif
'-------------------------------------------------------------------
' Hier: es war SLIO-Botschaft. Finde Adresse heraus und
' trage in Bit-Liste ein, verwende die Adresse im Testprogramm

      slio1_id = ( r_id bitand 7FEh) shl 5 ' nehme diese Adress f. Test
                                         ' maskiere DIR-Bit weg
      r_id = r_id bitand SLIO_ID_MASK    ' setzbare SLIO Adressbits
      if bit ( r_id, 8 ) = 1 then        ' wenn P8 gesetzt, dann
        r_id = ( r_id bitand 111000b ) + 40h
      endif
      r_id = r_id shr 3                  ' schiebe untere 3 ID-Bist weg
      set_bit slio_equip, r_id           ' setze Bitnummer in Liste
      print #LCD, "<1Bh>A<0><1><0F0h>SLIO ";r_id;" found"
      no_of_slios = no_of_slios + 1      ' zaehle die gefundene SLIO
                                         ' ignoriere hier Info in Daten
      if msg_len > 0 then                ' wenn Daten
        get #CAN, #0, msg_len, data$     ' hole sie aus dem Puffer
'        using "UH<2><2>   0.0.0.0.2"     ' Format HEX-Ausgabe
'        for i = 0 to len ( data$ ) - 1  ' alle Bytes des Strings
'          print_using #LCD, asc ( mid$ ( data$, i, 1 ) );' als HEX
'        next
      endif
    endif
  endwhile
  alive_wait = ALIVETIME               ' normale Wartezeit 'keep_alive'
END


'-------------------------------------------------------------------
' Sendet Kalibrier-Botschaften, um die SLIO synchronisiert zu halten.
' Spaetestens alle 8000 Bitzeiten muss eine Kalibrier-Botschaften
' gesendet werden.
'-------------------------------------------------------------------
TASK keep_alive
  BYTE ever

  for ever = 0 to 0 step 0             ' Endlosschleife
    put #CAN, calib$                   ' sende Kalibrier-Botschaft
    wait_duration alive_wait           ' warte max 4000 Bitzeiten
    if can_out$ <> "" then
```

```
      can_out$ = ""                    ' leer = gesendet
    endif
    wait_duration alive_wait           ' warte max 4000 Bitzeiten
  next
END
```

Zwei SLIOs an einem Bus

Das folgende Beispiel ist eine Kombination aus SLIO_DPM1.TIG und SLIO_HIGH_LOW.TIG. Es erzeugt DPM-Outputs auf SLIO1 (digital->analog) und digitale Rechteck-Ausgabe auf SLIO2. Stellen Sie auf einem CAN-SLIO-Board alle DIP-Schalter auf OFF, auf dem anderen CAN-SLIO-Board nur DIP-Schalter ID3 auf ON.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name:  SLIO_2.TIG
' 2 SLIOs im System
' erzeugt DPM-Outputs auf SLIO1 (digital->analog)
' erzeugt digitale Ausgabe auf SLIO2 (Rechteck)
' Verbinde
' 1 CAN-SLIO-Board, alle DIP-Schalter aus
' 1 CAN-SLIO-Board, nur DIP-Schalter ID3 an
' messe am Pin P10
'--------------------------------------------------------------------
user var strict                   ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC               ' User Function Codes
#INCLUDE DEFINE_A.INC             ' allg. Symbol-Definitionen
#INCLUDE CAN.INC                  ' CAN-Definitionen
#INCLUDE CANSLIO.INC              ' Definitionen zum CAN-SLIO-Chip

#define ALIVETIME 50              ' warte ca. 2500 * 20usec
#define SLIOSLOT 1000             ' msec Zeit, sich 'bereit' zu melden

#define NOT_READY 0               ' SLIO-Status 'nicht bereit'
#define READY 1                   ' SLIO-Status 'bereit'

'                  3  210          ID-Bitpositionen
#define ID_SLIO1 00000000000b     ' ID-Bit-Einstellung des SLIO1
#define ID_SLIO2 00000001000b     ' ID-Bit-Einstellung des SLIO2

BYTE no_of_slios                  ' Zaehler SLIOs im System
BYTE slio_12_found                ' Flag ueber gefundene SLIOs 1 und 2
BYTE slio_stat                    ' SLIO status
WORD slio1_id, slio2_id, calib_id ' ID der SLIOs und der Kalib.-Botsch.
LONG ac_code, ac_mask             ' Access-Code und Access-Mask
LONG slio_equip                   ' ein Bit fuer jede SLIO
LONG alive_wait                   ' 1/2 Wartezeit in 'keep_alive'
STRING calib$(5)                  ' Kalibrierungsbotschaft
STRING slio1_dpm1$(6), slio1_doe$(6)' DPM1 out, data out enable
STRING slio2_doe$(6)              ' data out enable SLIO2
STRING slio1_dout$
STRING slio2_dout$
STRING can_out$(13)               ' Sendebotschaft

'--------------------------------------------------------------------
TASK MAIN
  BYTE ever                       ' Endlosschleife
  WORD fi                         ' Pufferfuellung
  WORD value, tmp                 ' Analogwert
  LONG t                          ' 'keep_alive' Zeit
    LONG digi                     ' fuer digitalen Ausgang
  STRING tmp$(6)

  install_device #LCD, "LCD1.TDD" ' LCD-Treiber installieren

' warte auf SLIO-Code, aber filtere 'keep-alive'-Codes aus
' diese Bits stehen linksbuendig in den je 32-Bit Code u. Mask
' 01010000101 <- feste SLIO-Bits (50Ah)
' 00100111000 <- Filter: 1:don't care, 0: Bit muss stimmen (271h)

   install_device #CAN, "CAN1_K1.TDD", &
    "50 A0 00 00 &                ' access code
```

```
    10 45 &                         ' Bitzeit 20usec (moegl.:8...50usec)
    08 1A"%                         ' single filter mode, outctrl

                                    ' ID dieses SLIO: 0=standard frame
                                    ' + 2 ID-Bytes um 5 geshiftet
calib_id = 0aah shl 5               ' ID der Kalibrier-Botschaften
can_out$ = ""                       ' muss initialisiert sein

call find_slios                     ' stellt Liste aller im System
                                    ' befindlichen SLIOs zusammen
if slio_equip = 0 then              ' wenn keine SLIO gefunden
  goto no_slio_found                ' dann Programm abbrechen
else
  print #LCD, "<1Bh>A<0><2><0F0h>";no_of_slios;" SLIOs found";
endif
if slio_equip bitand 000000011b <> 3 then ' wenn SLIO1/2 n. gefunden
  print #LCD, "<1Bh>A<0><3><0F0h>SLIO1/2 not found";
endif
wait_duration 1500
print #LCD, "<1>";

slio1_id = (SLIO_FIX_ID bitor ID_SLIO1) shl 5 ' SLIO1-Identifier
slio2_id = (SLIO_FIX_ID bitor ID_SLIO2) shl 5 ' SLIO2-Identifier
run_task show_slio                  ' zeigt an, was die SLIO(s) senden

slio1_doe$ = "<0><0><0><4><0FFh><0FFh>" ' SLIO1_OUT_ENABLE
slio1_doe$ = ntos$ ( slio1_doe$, 1, -2, slio1_id)'ID high-Byte zuerst
can_out$ = slio1_doe$
wait_duration 1000

slio1_dout$ = "<0><0><0><3><0FFh><0FFh>" ' SLIO1_DATA_OUT
slio1_dout$ = ntos$ ( slio1_dout$, 1, -2, slio1_id )' ID high-Byte
can_out$ = slio1_dout$
wait_duration 1000
slio1_dout$ = "<0><0><0><3><0><0>" ' SLIO1_DATA_OUT
slio1_dout$ = ntos$ ( slio1_dout$, 1, -2, slio1_id )' ID einbauen
can_out$ = slio1_dout$
wait_duration 1000

slio2_doe$ = "<0><0><0><4><0FFh><0FFh>" ' SLIO2_OUT_ENABLE
slio2_doe$ = ntos$ ( slio2_doe$, 1, -2, slio2_id)'ID high-Byte zuerst
can_out$ = slio2_doe$
slio2_dout$ = "<0><0><0><3><0FFh><0FFh>" ' SLIO2_DATA_OUT
slio2_dout$ = ntos$ ( slio2_dout$, 1, -2, slio2_id )' ID einbauen
wait_duration 1000

slio1_dpm1$ = "<0><0><0><6><0><0>" ' SLIO1_DPM1
slio1_dpm1$ = ntos$ ( slio1_dpm1$, 1, -2, slio1_id )' ID high-Byte
value = 0
digi = 0                            ' Wert fuer digitalen Ausgang
t = ticks()
stop_task keep_alive                ' Hauptschleife uebernimmt das jetzt
for ever = 0 to 0 step 0            ' Endlosschleife ----------
  tmp = value shl 5                 ' 10-Bit rechtbuendig in WORD
  tmp$ = ntos$ ( slio1_dpm1$, 4, -2, tmp )' value high-Byte zuerst
  put #CAN, tmp$
  tmp$ = ntos$ ( slio2_dout$, 4, -2, digi )' value high-Byte zuerst
  put #CAN, tmp$
  digi = digi + 1                   ' dig. Ausgangswert aendern
  value = modulo_inc ( value, 0, 7FFh, 7Fh ) ' inc step 7Fh
```

```
      put #CAN, calib$                ' sende Kalibrier-Botschaft
      t = ticks()
    endif
  next                             ' Endlosschleife ----------

no_slio_found:                     ' keine SLIO gefunden
  stop_task keep_alive
  print #LCD, "<1>no SLIO found"
  print #LCD, "program terminated"
END


'--------------------------------------------------------------------
' Zeigt Inhalte der SLIO-Botschaften an
' Zeile 1 auf LCD: ID
' Zeile 2 auf LCD: Datenbytes in HEX
'--------------------------------------------------------------------
TASK show_slio
  BYTE ever, i                          ' Schleifenvariable
  WORD ibu_fill                         ' Eingangspufferfuellung
  BYTE frameformat, msg_len             ' Frame-Format, Botschaftslaenge
  LONG r_id                             ' Empfangs-IDreceive ID
  STRING msg$(13), data$(8)             ' Botschaft und Daten

  for ever = 0 to 0 step 0               ' Endlosschleife
rx_cont:
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
    if ibu_fill > 2 then                 ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat       ' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        r_id = r_id shr 5
                                         ' wenn keine SLIO-Botschaft
        if r_id bitand SLIO_ID_IMASK <> SLIO_FIX_ID then ' aus Puffer
          if msg_len > 0 then            ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto rx_cont                   ' warte auf naechste Botschaft
        endif
      else                               ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then              ' wenn Daten
          get #CAN, #0, msg_len, data$  ' hole sie aus dem Puffer
        endif
        goto rx_cont                     ' warte auf naechste Botschaft
      endif                              ' ----------------------------
                                         ' Hier: es war SLIO-Botschaft
      using "UH<3><3>   0 0 0 0 3"       ' fuer ID Anzeige
      print_using #LCD, "<1Bh>A<0><1><0F0h>  ID:";r_id ' ID des SLIO
      print #LCD, "                      <13>DATA:";' loesche Datenanz.
      if msg_len > 0 then                ' wenn Daten
        get #CAN, #0, msg_len, data$     ' hole sie und zeige an
        using "UH<2><2>   0.0.0.0.2"     ' Format HEX-Ausgabe
        for i = 0 to len ( data$ ) - 1   ' alle Bytes des Strings
          print_using #LCD, asc ( mid$ ( data$, i, 1 ) );' als HEX
        next
      endif
    endif
```

```
END


'--------------------------------------------------------------------
' Sendet Kalibrier-Botschaften und wartet auf
' (gefilterte) SLIO-Botschaften als Antwort
' Innerhalb einer gesetzten Zeit muessen alle SLIOs sich gemeldet
' haben. Die Filterung muss im Hauptprogramm gesetzt sein.
' Nicht-SLIO-Botschaften und extended Frames werden verworfen,
' d.h. lediglich aus dem Puffer geräumt.
'--------------------------------------------------------------------
SUB find_slios ()
  BYTE ever, i                    ' Schleifenvariable
  WORD ibu_fill                   ' Eingangspufferfuellung
  BYTE frameformat, msg_len       ' Frame-Format, Botschaftslaenge
  LONG r_id                       ' Empfangs-IDreceive ID
  LONG t
  STRING msg$(13), data$(8)       ' Botschaft und Daten

  slio_equip = 0                  ' fange an mit 'keine SLIO'
  no_of_slios = 0
  calib_id = 0aah shl 5           ' ID der Kalibrier-Botschaften
  calib$ = "<0><0><0><0aah><4>"   ' spezieller String fuer Init-SLIO
  calib$ = ntos$ ( calib$, 1, -2, calib_id )' ID high-Byte zuerst

  alive_wait = ALIVETIME/2        ' in der Initphase kuerzer
  run_task keep_alive             ' sendet Kalibrier-Botschaften
                                  ' und haelt SLIOs synchronisiert
  print #LCD, "trying to find SLIOs";
  t = ticks()                     ' Anfang des Zeitfensters
  while diff_ticks ( t ) < SLIOSLOT ' innerhalb des Zeitfensters
find_cont:
    wait_duration 50
    get #CAN, #0, #UFCI_IBU_FILL, 0, ibu_fill
    if ibu_fill > 2 then          ' wenn mindenstens eine Message
      get #CAN, #0, 1, frameformat' hole Frame-Info-Byte
      msg_len = frameformat bitand 1111b ' Laenge
      if frameformat bitand 80h = 0 then ' wenn Standard-Frame
        get #CAN, #0, CAN_ID11_LEN, r_id ' hole ID-Bytes
        r_id = byte_mirr ( r_id, 2 )
        r_id = r_id shr 5
                                  ' wenn keine SLIO-Botschaft
        if r_id bitand SLIO_ID_IMASK <> SLIO_FIX_ID then ' aus Puffer
          if msg_len > 0 then     ' wenn Daten: wegwerfen
            get #CAN, #0, msg_len, data$' hole sie aus dem Puffer
          endif
          goto find_cont          ' warte auf naechste Botschaft
        endif
      else                        ' sonst ist es extended frame
        get #CAN, #0, CAN_ID29_LEN, r_id' und damit nicht von SLIO
        if msg_len > 0 then       ' wenn Daten
          get #CAN, #0, msg_len, data$  ' hole sie aus dem Puffer
        endif
        goto find_cont            ' warte auf naechste Botschaft
      endif
'--------------------------------------------------------------------
' Hier: es war SLIO-Botschaft. Finde Adresse heraus und
' trage in Bit-Liste ein, verwende die Adresse im Testprogramm

      slio1_id = ( r_id bitand 7FEh) shl 5 ' nehme diese Adress f. Test
```

```
      r_id = r_id bitand SLIO_ID_MASK   ' setzbare SLIO Adressbits
      if bit ( r_id, 8 ) = 1 then       ' wenn P8 gesetzt, dann
        r_id = ( r_id bitand 111000b ) + 40h ' erzeuge 4-Bit-Adr.
      endif
      r_id = r_id shr 3                 ' schiebe untere 3 ID-Bist weg
      set_bit slio_equip, r_id          ' setze Bitnummer in Liste
'      print #LCD, "<1Bh>A<0><1><0F0h>SLIO ";r_id;" found"
      using "UH<1><1>   0 0 0 0 1"      ' fuer slio_equip Anzeige
      print_using #LCD, "<1Bh>A<0><1><0F0h>SLIO_equip ";slio_equip
      no_of_slios = no_of_slios + 1     ' zaehle die gefundene SLIO
                                        ' ignoriere hier Info in Daten
      if msg_len > 0 then               ' wenn Daten
        get #CAN, #0, msg_len, data$    ' hole sie aus dem Puffer
'        using "UH<2><2>   0.0.0.0.2"    ' Format HEX-Ausgabe
'        for i = 0 to len ( data$ ) - 1  ' alle Bytes des Strings
'          print_using #LCD, asc ( mid$ ( data$, i, 1 ) );' als HEX
'        next
      endif
    endif
  endwhile
  alive_wait = ALIVETIME               ' normale Wartezeit 'keep_alive'
END


'---------------------------------------------------------------------
' Sendet Kalibrier-Botschaften, um die SLIO synchronisiert zu halten.
' Spaetestens alle 8000 Bitzeiten muss eine Kalibrier-Botschaften
' gesendet werden.
'---------------------------------------------------------------------
TASK keep_alive
  BYTE ever
  LONG t

  t = ticks()
  for ever = 0 to 0 step 0             ' Endlosschleife
    if diff_ticks ( t ) > 50 then      ' Wert haengt von Bitzeit ab
      put #CAN, calib$                 ' sende Kalibrier-Botschaft
      t = ticks()
    endif
    if can_out$ <> "" then
      put #CAN, can_out$               ' Sendebotschaft
      can_out$ = ""                    ' leer = gesendet
    endif
  next
END
```

Touch-Memory

Der Device-Treiber ‘TMEM.TDD’ unterstützt die serielle Verbindung mit Touch-Memory-Modulen der Serie DS199x von Dallas Semiconductor. Bei der Installation des Treibers wird durch den Dateinamen festgelegt, an welchem Pin das Touch-Memory angeschlossen ist. Das Timing der Übertragung ist durch die Einstellung des TIMERA festgelegt.

Dateiname: TMEM_**Pp**.TDD

INSTALL DEVICE #*D*, "TMEM_Pp.TDD", *ResTim, PresTim, NCTim*

Hinweis: TIMERA.TDD muß vorher eingebunden werden.

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Pp im Dateinamen steht für:
P: interner Port
p: Meßpin.

ResTim ist ein Parameter zur Festlegung der Resetzeit in TIMERA-Ticks.

PresTim ist ein Parameter zur Festlegung der Zeit für 'Wait For Presence' in TIMERA-Ticks.

NCTim ist ein Parameter zur Festlegung der Zeit in TIMERA-Ticks, die nach einem 'Presence'-Signal bis zur Kommunikation gewartet wird (No-Communication-Time).

Das Lesen oder Schreiben eines Bytes findet immer in der Geschwindigkeit der TIMERA-Ticks statt. Das Timing für 'Reset', 'Wait for Presence' und 'No Communication after Presence' kann mit den Installationsparametern variiert werden.

Mit der Instruktion PUT können bis zu 256 Bytes in das Touch-Memory geschrieben werden:

PUT #D, *String$*

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

String$ enthält die Bytes, in den das Touch-Memory geschrieben werden. Die Daten werden zunächst in den Ausgangspuffer übertragen und dann vom Device-Treiber an den externen Chip gesendet.

Um Daten aus einem Touch-Memory zu lesen, wird mit der Instruktion PUT auf dem Sekundärkanal 1 die gewünschte Anzahl der zu lesenden Bytes ausgegeben:

PUT #D, #1, *Anzahl*

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Anzahl gibt an, wieviel Bytes aus dem Touch-Memory gelesen werden. Die Daten werden zunächst in den Eingangspuffer übertragen und werden dann mit der Instruktion GET aus dem Device-Treiber gelesen.

Nachdem das Lesekommando abgesetzt wurde, kann der Pufferfüllstand abgefragt werden, um festzustellen, ob die gewünschte Anzahl Zeichen gelesen wurde:

Das Touch-Memory wird zurückgesetzt (Reset), indem ein beliebiger Wert auf die Sekundär-Adresse 2 ausgegeben wird. Dabei wird auch gleichzeitig das 'Presence'-Signal geprüft:

PUT #D, #2, *Dummy*

Sowohl eingehende als auch gesendete Daten werden in einem Puffer zwischengespeichert. Größe, Füllstand oder verbleibender Platz der Ein- und Ausgangspuffer, Status-Informationen sowie die Version des Treibers können mit Hilfe der User-Function-Codes abgefragt werden.

User-Function-Codes des TMEM_xx.TDD

User-Function-Codes für Abfragen (Instruktion GET):

Nr	Symbol Prefix UFCI_	Beschreibung
1	UFCI_IBU_FILL	Füllstand des Eingangspuffers (Byte)
2	UFCI_IBU_FREE	freier Platz im Eingangspuffer (Byte)
3	UFCI_IBU_VOL	Größe des Eingangspuffers (Byte)
33	UFCI_OBU_FILL	Füllstand des Ausgangspuffers (Byte)
34	UFCI_OBU_FREE	freier Platz im Ausgangspuffer (Byte)
35	UFCI_OBU_VOL	Größe des Ausgangspuffers (Byte)
65	UFCI_LAST_ERRC	letzter Error-Code
99	UFCI_DEV_VERS	Version des Treibers
160	UFCI_TMEM_OVL	Overflow-Status: 0: ok n: Zahl der Buffer-Overflows
161	UFCI_TMEM_PRS	Presence-Status: 0: ok <>0: nicht Present

User-Function-Codes der I/O-Puffer für die Instruktion PUT:

Nr	Symbol	Beschreibung
1	UFCO_IBU_ERASE	Eingangspuffer löschen
33	UFCO_OBU_ERASE	Ausgangspuffer löschen

Das Touch-Memory wird mit einer einzigen Leitung an den Portpin angeschlossen. Die Leitung hat einen Pull-up-Widerstand von 4k7 gegen VCC.

Programmbeispiel:

```
'-----------------------------------------------------------------------
' Name: TMEM1.TIG
' DS1993
'-----------------------------------------------------------------------
user_var_strict
#include DEFINE_A.INC                    ' allgemeine Definitionen

#include UFUNC3.INC                      ' User Function Codes

TASK MAIN
  BYTE TMEM_PRES                         ' TMEM-Praesenz Flag
  BYTE FAM_CODE                          ' Family Code des Chips
  LONG SNR, I                            ' Seriennummer
  BYTE CHKSUM                            ' CRC-Prüfsummenbyte
  STRING A$

  INSTALL_DEVICE #TA,"TIMERA.TDD",2,125 ' 5 kHz
  INSTALL_DEVICE #LCD, "LCD1.TDD"        ' Text LCD 4x20
' Port 8 pin 0 is TouchMemory Bus      RESET, PRESENCE, NO-COMM
  INSTALL_DEVICE #TMEM, "TMEM_80.TDD",     3,        20,      5

                                         ' siehe ob TMEM praesent ist
  PUT #TMEM, #2, 0                       ' iBus zuruecksetzen
  WAIT_DURATION 25                       ' um Present-Flag zu setzen

  GET #TMEM, #0, #UFCI_TMEM_PRS, 1, TMEM_PRES
  PRINT #LCD, "<1>TMEM-Flg=";TMEM_PRES  ' 0=praesent, 255=nicht pr.
  WAIT_DURATION 2000

  PUT #TMEM, "<033H>"                    ' schreibe Kommando "READ ROM"
  WAIT_DURATION 10

  PUT #TMEM,#1, 8                        ' Device-Treiber vorbereiten
  WAIT_DURATION 25                       ' um 8 Bytes zu lesen

  GET #TMEM, #0, 1, FAM_CODE             ' das erste Byte lesen
  PRINT #LCD, "<1>Fam.Code:";FAM_CODE
  PRINT #LCD, "SNR:";
  GET #TMEM, #0, 6, A$                   ' ein Byte lesen
  FOR I = 5 TO 0 STEP -1                 ' 6 Bytes Seriennummer
    SNR = NFROMS(A$,I,1)                 ' MSB HEX auf LCD
    USING "UH<2><2>   0 0 0 0 0"
    PRINT_USING #LCD, SNR;
  NEXT
  GET #TMEM, #0, 1, CHKSUM               ' 1 Byte Pruefsumme
  USING "UH<2><2>   0 0 0 0 0"           ' Format f. HEX auf LCD
  PRINT_USING #LCD,"<10><13>CRC:";CHKSUM' und HEX anzeigen
                                         ' ----------------------------
  PUT #TMEM, "<0FH><26h><0>Hello 1-wire"' "write scratchpad"
  WAIT_DURATION 50                       ' Kommando schreiben
  PUT #TMEM, #2, 0                       ' ibus zurücksetzen
  WAIT_DURATION 25

  PUT #TMEM, "<033H>"                    ' schreibe Kommando "READ ROM"
  WAIT_DURATION 10

  PUT #TMEM,#1, 8                        ' 8 Bytes lesen
  WAIT_DURATION 20
```

```
  PUT #TMEM, "<0AAH>"                    ' "read scratchpad"
  WAIT_DURATION 30                       ' Kommando auf Bus schreiben

  PUT #TMEM,#1, 12                       ' lese 12 Bytes
  WAIT_DURATION 25
  GET #TMEM, #0, 12, A$                  ' lese 12 Bytes aus Buffer
  A$ = RIGHT$ (A$,5)
  PRINT #LCD, A$;                        ' und zeige an
END
```

Leere Seite

Uhr – RTC1

Der Device-Treiber ‘RTC1’ unterstützt die interne Echtzeituhr oder simuliert diese, wenn das Modul nicht mit Echtzeituhr ausgestattet ist.

Dateiname: RTC1.TDD

INSTALL DEVICE #*D*, "RTC1.TDD" [, *P1*, *P2*]

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

P1 und P2 sind weitere Parameter, die die Standard-Anschlußbelegung des RTC1-Treibers verändern.

	Beschreibung des Parameters
P1	Logische Portadresse für den Anschluß der Echtzeit-Uhr
P2	True-Bitmaske für den Anschluß der Echtzeit-Uhr, gibt Bit-Position an.

Die interne Uhr ist ein Zähler mit der Einheit Sekunden.

Die Uhr kann eine im Modul eingebaute, über den Batterie-Anschluß pufferbare Uhr sein. Diese Uhr läuft weiter, solange sie gepuffert wird. Ist keine gepufferte Uhr vorhanden, übernimmt Tiger-BASIC® selbst die Aufgabe des Sekundenzählers. Nach einem Reset beginnt dieser Zähler bei 0. Der Timer wird durch Ausgeben einer LONG-Zahl mit einer Ausgabe-Instruktion auf das Device gesetzt und mit einer Eingabe-Instruktion gelesen.

Die Datei ‘TIMECVT.TIG’ enthält Beispiel-Unterprogramme, die den Sekundenzähler in Zeitangaben von Minuten, Stunden und Datum umrechnen. Ähnliche Unterprogramme gibt es auch zum setzen des Sekundenzählers. Alle die Uhr betreffenden Beispiel-Unterprogramme in der Datei ‘TIMECVT.TIG’ gehen davon aus, daß um 0.00 Uhr am 1. Januar 1980 der Zähler mit 0 Sekunden begonnen hat. Sie können in Ihrem System einen beliebigen Anfangszeitpunkt zugrundelegen, jedoch dann die vorhandenen Unterprogramme nicht mehr nutzen.

Die Alarm-Funktion wird nur durch die Echtzeituhr unterstützt. Setzen der Alarmzeit geschieht über die Sekundär-Adresse 1: Das setzen der Alarmzeit bewirkt, daß die Uhr den Alarm-Pin nach einer kurzen Verzögerung auf ‘high’ schaltet. Bei erreichen der Alarmzeit setzt die Echtzeituhr den ‘Alarm’-Pin des BASIC-Tiger®-Moduls wieder auf ‘low’.

Sekundär-Adresse	Funktion
0	setzen und lesen der Uhrzeit
1	setzen der Alarmzeit (mit Echtzeituhr)


```
INSTALL DEVICE #RTC, "RTC1.TDD" ' Timer / Uhr

  LONG ALARMZEIT
  ALARMZEIT = 301234            ' spaeter als jetzt
  PUT #RTC, #1, ALARMZEIT       ' Alarm-Pin geht auf high
```

User-Function-Codes des RTC1.TDD

RTC1-User-Function-Codes und die zugehörigen Antworten des Treibers:

Nr	Symbol	Beschreibung
0A0H	UFCI_RTC_STAT0	Status des Uhrenchips
		Antwort des Treibers:
0	RTC_INITIAL	Zustand gleich nach Power-ON
1	RTC_INSTALLING	Installation dauert noch an
2	RTC_NO_RTC	keine RTC-Hardware vorhanden
3	RTC_PRESENT	OK, RTC-Hardware anwesend
4	RTC_RETRY	wiederholter Versuch, RTC zu finden
0A1H	UFCI_RTC_STAT1	Status des RTC-Device-Treibers
		Antwort des Treibers:
0	RTC_READY	bereit
1	RTC_BUSY	beschäftigt

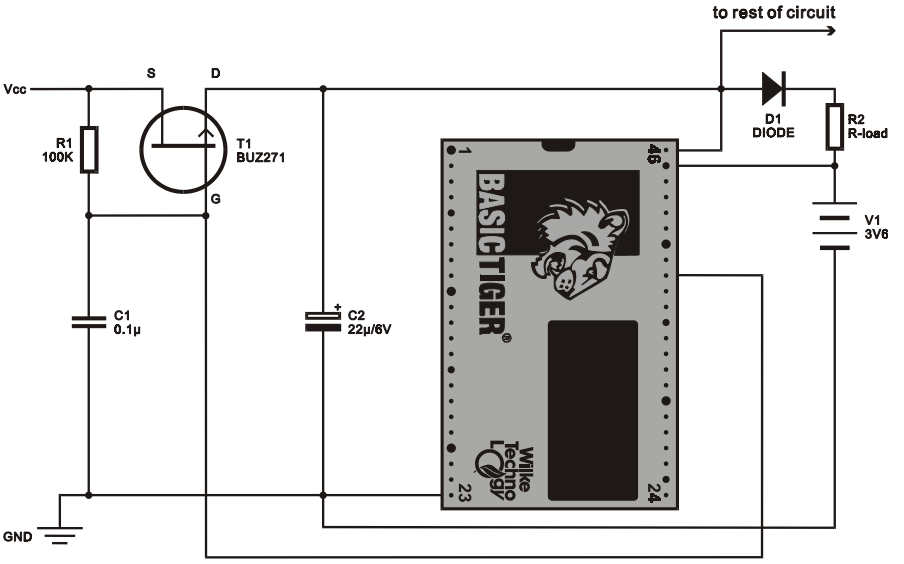
Die maximale Abweichung der Echtzeituhr ist maxiaml ca. ±4,3 Sekunden pro Tag. Das entspricht der Genauigkeit des Uhrenquarzes (±50 ppm).

Bei Abfragen auf die Uhrzeit sollte immer eine Formulierung ‘größer’, ‘kleiner’, ‘größer-gleich’ oder ‘kleiner-gleich’ gewählt werden, niemals ‘gleich’. Die Uhr kann durch interne Korrekturfaktoren gelegentlich eine Sekunde überspringen. !

Programmbeispiel:

```
'-------------------------------------------------------------------------
' Name: RTC1.TIG
'-------------------------------------------------------------------------
#INCLUDE UFUNC3.INC                      ' User Function Codes
TASK Main                                ' Beginn Task MAIN
  LONG Seconds, Prev_Sec                 ' LONG-Variablen deklarieren
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8
  INSTALL DEVICE #3, "RTC1.TDD"          ' RTC-Treiber installieren

  RTCSTAT = RTC_INITIAL                  '
  WHILE RTCSTAT < RTC_NO_RTC             ' Solange RTC gesucht wird
    GET #3, #0, #UFCI_RTC_STAT0, 1, RTCSTAT ' erfrage Status der Uhr
    PRINT #1,"<1>installing";            '
  ENDWHILE                               '
  IF RTCSTAT = RTC_PRESENT THEN          ' falls RTC vorhanden
    Seconds = 12345678                   ' Wert vorbesetzen
    PUT #3, Seconds                      ' setze Uhr in abs. Sekunden
    RTCSTAT = RTC_BUSY                   '
    WHILE RTCSTAT = RTC_BUSY             ' Solange RTC busy
      GET #3, #0, #UFCI_RTC_STAT1, 1, RTCSTAT ' lese Status der Uhr
      PRINT #1,"<1>busy";                '
    ENDWHILE                             '
    LOOP 9999999                         ' sehr viele Loops
      Prev_Sec = Seconds                 ' alte Zeit behalten
      WHILE Seconds = Prev_Sec           ' Solange aktuelle = alte Zeit
        GET #3,0, Seconds                ' lies Uhr
      ENDWHILE                           '
      PRINT #1,"<1>RTC-Time =<0>";Seconds; ' zeige an, wenn neue Zeit
    ENDLOOP                              '
  ELSE                                   ' falls keine RTC
    PRINT #1, "<1>No RTC found"          '
  ENDIF                                  '
END                                      ' Ende Task MAIN
```

Folgende Beispielschaltung zeigt, wie mit Hilfe der Echtzeituhr und eines FET das Modul abgeschaltet und bei erreichter Alarmzeit wieder eingeschaltet werden kann. Beachten Sie, daß das Modul eventuell nicht richtig abgeschaltet wird, wenn eine Versorgung über die I/O-Pins stattfindet. Der FET ist entsprechend dem gesamten Stromverbrauch der Schaltung zu bemessen.



Zeitbasis-Timer

Dieser Device-Treiber bildet eine interne einstellbare Zeitbasis, die von anderen Device-Treibern verwendet wird. Bei der Installation des Treibers werden Bereich und Teilerfaktor festgelegt. Die Einstellungen sind jedoch durch Ausgabe von Daten an den Treiber auch später noch veränderbar.

Dateiname: TIMERA.TDD

INSTALL DEVICE #*D*, "TIMERA.TDD", *Bereich, Teilerfaktor*

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

Bereich ist ein Parameter zur Festlegung des Bereichstaktes.

Teilerfaktor ist ein Parameter zur Festlegung des Faktors, mit dem der Bereichstakt heruntergeteilt wird.

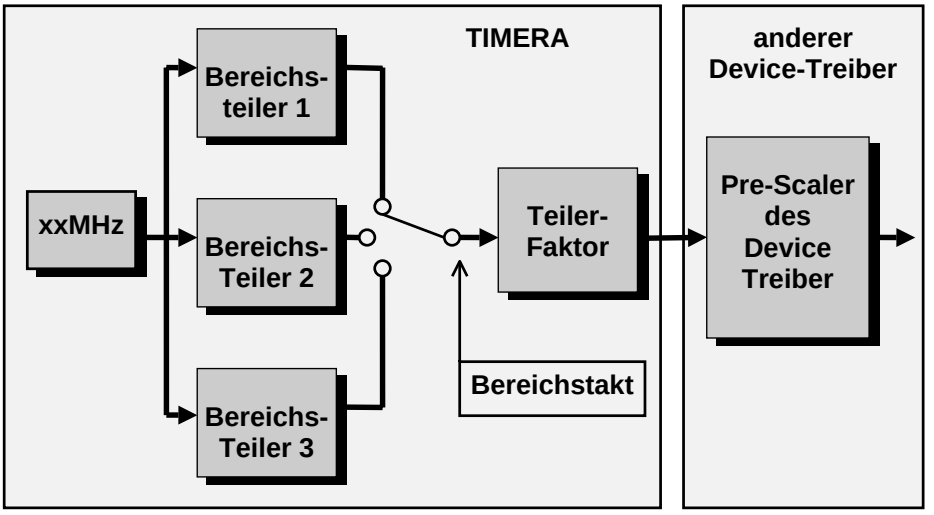
Der Device-Treiber TIMERA.TDD bildet eine Zeitbasis, die von anderen Device-Treibern genutzt wird (z.B. von dem schnellen Analog-Device-Treiber ANALOG2.TDD oder von PWM2.TDD). Es gibt immer nur eine Einstellung der Zeitbasis, die für alle Treiber gilt, die die Zeitbasis benötigen.

Beachte: TIMERA muß **vor** den abhängigen anderen Device-Treibern eingebunden werden.

Zunächst erzeugen drei Bereichsteiler aus der Grundfrequenz drei verschiedene Bereichsfrequenzen:

Bereich	Bereichstakt	Auflösung	Zeitbereich
0	Timer stoppen		
1	2.500.000 kHz	0,400 µsec	0,0004...26,214 msec
2	625.000 kHz	1,600 µsec	0,0016...104,856 msec
3	156.250 kHz	6,400 µsec	0,0064...419,424 msec
4	extern getriggert an L70		

Danach erfolgt die Feineinstellung der Zeitbasis-Frequenz, indem durch einen weiteren Faktor geteilt wird.



Während der Laufzeit wird die Zeitbasis eingestellt, indem der Bereich und der Teilerfaktor als **Datenbytes an den Device-Treiber** gesendet werden. Da der Treiber keine Datenausgabe hat, kommen hier nicht User-Function-Codes zum Einsatz. Die Werte für den Bereich und den Teilerfaktor sind BYTES, können jedoch auch als 2 Zeichen eines Strings, 2 Bytes einer WORD-Variablen oder WORD-Konstanten übergeben werden.

Beispiel: Einstellen der Zeitbasis während der Laufzeit auf 4960Hz (Bereich 2 hat den Bereichstakt 625.000Hz. Der Faktor 126 teilt den Basistakt auf 4960,3Hz:

```
BEREICH = 2                             ' BYTE-Variable BEREICH
FAKTOR = 126                            ' BYTE-Variable FAKTOR
PUT #13, BEREICH, FAKTOR                ' setze neue Zeitbasis
```


Wenn mehrere Device-Treiber die Zeitbasis benötigen, wird im Device-Treiber 'TIMERA' die höchste benötigte Frequenz eingestellt. Der Pre-Scaler des Device-Treibers, der einen langsameren Takt benötigt, teilt den Zeitbasistakt auf den gewünschen Wert herunter.

! Beachte: TIMERA kann zu erheblicher CPU-Belastung führen. Ein Beispiel: wenn TIMERA von vier anderen Device-Treiber benutzt wird und auf 5000Hz eingestellt ist, dann entstehen 4x5000=20000 kleine Systemtasks pro Sekunde. Stellen Sie TIMERA deswegen auf die niedrigste mögliche Frequenz ein.

Die Tabelle auf den folgenden Seiten zeigt die den Frequenzen zugehörigen Einstellungen für Bereich und Teilerfaktor des Device-Treibers 'TIMERA'.

Frequ.	Faktor
Bereich 1	
12.500	200
12.438	201
12.376	202
12.315	203
12.255	204
12.195	205
12.136	206
12.077	207
12.019	208
11.962	209
11.905	210
11.848	211
11.792	212
11.737	213
11.682	214
11.628	215
11.574	216
11.521	217
11.468	218
11.416	219
11.364	220
11.312	221
11.261	222
11.211	223
11.161	224
11.111	225
11.062	226
11.013	227

Frequ.	Faktor
10.965	228
10.917	229
10.870	230
10.823	231
10.776	232
10.730	233
10.684	234
10.638	235
10.593	236
10.549	237
10.504	238
10.460	239
10.417	240
10.373	241
10.331	242
10.288	243
10.246	244
10.204	245
10.163	246
10.121	247
10.081	248
10.040	249
10.000	250
9.960	251
9.920	252
9.881	253
9.842	254
9.803	255
9.765	0

Frequ.	Faktor
Frequ.	Faktor
Bereich 2	
12.500	50
12.255	51
12.019	52
11.792	53
11.574	54
11.364	55
11.161	56
10.965	57
10.776	58
10.593	59
10.417	60
10.246	61
10.081	62
9.920	63
9.765	64
9.615	65
9.469	66
9.328	67
9.191	68
9.057	69
8.928	70
8.802	71
8.680	72
8.561	73
8.445	74
8.333	75
8.223	76

Frequ.	Faktor
8.116	77
8.012	78
7.911	79
7.812	80
7.716	81
7.621	82
7.530	83
7.440	84
7.352	85
7.267	86
7.183	87
7.102	88
7.022	89
6.944	90
6.868	91
6.793	92
6.720	93
6.648	94
6.578	95
6.510	96
6.443	97
6.377	98
6.313	99
6.250	100
6.188	101
6.127	102
6.067	103
6.009	104
5.952	105

Frequ.	Faktor	Frequ.	Faktor	Frequ.	Faktor	Frequ.	Faktor
5.896	106	4.629	135	3.810	164	3.238	193
5.841	107	4.595	136	3.787	165	3.221	194
5.787	108	4.562	137	3.765	166	3.205	195
5.733	109	4.528	138	3.742	167	3.188	196
5.681	110	4.496	139	3.720	168	3.172	197
5.630	111	4.464	140	3.698	169	3.156	198
5.580	112	4.432	141	3.676	170	3.140	199
5.530	113	4.401	142	3.654	171	3.125	200
5.482	114	4.370	143	3.633	172	3.109	201
5.434	115	4.340	144	3.612	173	3.094	202
5.387	116	4.310	145	3.591	174	3.078	203
5.341	117	4.280	146	3.571	175	3.063	204
5.296	118	4.251	147	3.551	176	3.048	205
5.252	119	4.222	148	3.531	177	3.033	206
5.208	120	4.194	149	3.511	178	3.019	207
5.165	121	4.166	150	3.491	179	3.004	208
5.122	122	4.139	151	3.472	180	2.990	209
5.081	123	4.111	152	3.453	181	2.976	210
5.040	124	4.084	153	3.434	182	2.962	211
5.000	125	4.058	154	3.415	183	2.948	212
4.960	126	4.032	155	3.396	184	2.934	213
4.921	127	4.006	156	3.378	185	2.920	214
4.882	128	3.980	157	3.360	186	2.906	215
4.844	129	3.955	158	3.342	187	2.893	216
4.807	130	3.930	159	3.324	188	2.880	217
4.770	131	3.906	160	3.306	189	2.866	218
4.734	132	3.881	161	3.289	190	2.853	219
4.699	133	3.858	162	3.272	191	2.840	220
4.664	134	3.834	163	3.255	192	2.828	221

Frequ.	Faktor
2.815	222
2.802	223
2.790	224
2.777	225
2.765	226
2.753	227
2.741	228
2.729	229
2.717	230
2.705	231
2.693	232
2.682	233
2.670	234
2.659	235
2.648	236
2.637	237
2.626	238
2.615	239
2.604	240
2.593	241
2.582	242
2.572	243
2.561	244
2.551	245
2.540	246
2.530	247
2.520	248
2.510	249
2.500	250

Frequ.	Faktor
2.490	251
2.480	252
2.470	253
2.460	254
2.450	255
2.441	0

Frequ.	Faktor
Bereich 3	
13.020	12
12.019	13
11.160	14
10.416	15
9.765	16
9.191	17
8.680	18
8.223	19
7.812	20
7.440	21
7.102	22
6.793	23
6.510	24
6.250	25
6.009	26
5.787	27
5.580	28
5.387	29
5.208	30
5.040	31
4.882	32
4.734	33
4.595	34
4.464	35
4.340	36
4.222	37
4.111	38
4.006	39

Frequ.	Faktor
3.906	40
3.810	41
3.720	42
3.633	43
3.551	44
3.472	45
3.396	46
3.324	47
3.255	48
3.188	49
3.125	50
3.063	51
3.004	52
2.948	53
2.893	54
2.840	55
2.790	56
2.741	57
2.693	58
2.648	59
2.604	60
2.561	61
2.520	62
2.480	63
2.441	64
2.403	65
2.367	66
2.332	67
2.297	68

Frequ.	Faktor	Frequ.	Faktor	Frequ.	Faktor	Frequ.	Faktor
2.264	69	1.594	98	1.230	127	1.001	156
2.232	70	1.578	99	1.220	128	995	157
2.200	71	1.562	100	1.211	129	988	158
2.170	72	1.547	101	1.201	130	982	159
2.140	73	1.531	102	1.192	131	976	160
2.111	74	1.516	103	1.183	132	970	161
2.083	75	1.502	104	1.174	133	964	162
2.055	76	1.488	105	1.166	134	958	163
2.029	77	1.474	106	1.157	135	952	164
2.003	78	1.460	107	1.148	136	946	165
1.977	79	1.446	108	1.140	137	941	166
1.953	80	1.433	109	1.132	138	935	167
1.929	81	1.420	110	1.124	139	930	168
1.905	82	1.407	111	1.116	140	924	169
1.882	83	1.395	112	1.108	141	919	170
1.860	84	1.382	113	1.100	142	913	171
1.838	85	1.370	114	1.092	143	908	172
1.816	86	1.358	115	1.085	144	903	173
1.795	87	1.346	116	1.077	145	897	174
1.775	88	1.335	117	1.070	146	892	175
1.755	89	1.324	118	1.062	147	887	176
1.736	90	1.313	119	1.055	148	882	177
1.717	91	1.302	120	1.048	149	877	178
1.698	92	1.291	121	1.041	150	872	179
1.680	93	1.280	122	1.034	151	868	180
1.662	94	1.270	123	1.027	152	863	181
1.644	95	1.260	124	1.021	153	858	182
1.627	96	1.250	125	1.014	154	853	183
1.610	97	1.240	126	1.008	155	849	184

Frequ.	Faktor	Frequ.	Faktor	Frequ.	Faktor
844	185	730	214	643	243
840	186	726	215	640	244
835	187	723	216	637	245
831	188	720	217	635	246
826	189	716	218	632	247
822	190	713	219	630	248
818	191	710	220	627	249
813	192	707	221	625	250
809	193	703	222	622	251
805	194	700	223	620	252
801	195	697	224	617	253
797	196	694	225	615	254
793	197	691	226	612	255
789	198	688	227	610	0
785	199	685	228		
781	200	682	229		
777	201	679	230		
773	202	676	231		
769	203	673	232		
765	204	670	233		
762	205	667	234		
758	206	664	235		
754	207	662	236		
751	208	659	237		
747	209	656	238		
744	210	653	239		
740	211	651	240		
737	212	648	241		
733	213	645	242		

User-Function-Codes des TIMERA.TDD für Input (Instruktion GET):

Nr	Name	Beschreibung
<65>	UFCI_LAST_ERRC	letzter Error-Code
<99>	UFCI_DEV_VERS	Version des Treibers

Beispiel: frage die Versionsnummer des Treibers ab:

```
GET #2,#1, #UFCI_DEV_VERS, 2, wVersion
```

Leere Seite

SET1

Der Device-Treiber ‘SET1’ hilft während der Entwicklungsphase in Zusammenarbeit mit RES1.TDD, die Belastung der CPU durch die an den TIMERA angekoppelten Device-Treiber festzustellen.

Dateiname: SET1.TDD

INSTALL DEVICE #*D*, "SET1.TDD", P1

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

P1 ist ein Parameter, der den zur Testausgabe verwendeten Pin festlegt. Der Parameter besteht aus einer zweistelligen Zahl. Die Zehnerziffer gibt den (internen) Port an, die Einerziffer gibt die Pin-Nummer des Ports an.

Der Device-Treiber SET1.TDD dient in Zusammenarbeit mit RES1.TDD während der Entwicklungsphase dazu, die Belastung der CPU durch die an den TIMERA angekoppelten Device-Treiber festzustellen. Am Anfang jedes Zeit-Ticks des TIMERA setzt der Treiber SET1 den bei der Installation festgelegten Pin auf 'high'. Am Ende des Zeit-Ticks, d.h. wenn alle angekopplelten Treiber ihre Aufgabe erledigt haben, setzt RES1 den Pin wieder auf 'low'. Mit einem Oszilloskop kann nun die Tätigkeit der TIMERA-abhängigen Device-Treiber beobachtet werden. Die Zeit, die das Signal an dem Pin 'high' ist, wird von den betroffenen Device-Treibern verbraucht. Die Zeit, die das Signal 'low' ist, steht den anderen Treibern und BASIC zur Verfügung.

Hinweis:
a) der verwendete Pin muß im BASIC-Programm als Ausgang initialisiert werden.
b) SET1.TDD wird nach TIMERA.TDD, jedoch vor den abhängigen Device-Treibern eingebunden.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name: SET1.TIG
'--------------------------------------------------------------------
#INCLUDE UFUNC3.INC

TASK MAIN
  LONG REST
  INSTALL_DEVICE #13, "TIMERA.TDD", 1,250 ' 10kHz
  INSTALL_DEVICE #19, "SET1.TDD", 70    ' Setze Pin L70 auf high
  INSTALL_DEVICE #20, "PO2_P80.TDD"     ' Pulse auf Pin L80
  INSTALL_DEVICE #21, "PO2_P81.TDD"     ' Pulse auf Pin L81
  INSTALL_DEVICE #23, "RES1.TDD", 70    ' Setze Pin L70 auf low

  DIR_PIN 7,0,0                         ' setze Testpin als Ausgang

  PUT #20, 400, 12, 24                  ' 400 Pulse 2,4microsec cycle
  PUT #21, #2, 800, 6, 12               ' 800 Pulse 1,2microsec cycle
  REST = 1
  WHILE REST > 0
    GET #20, #0, #UFCI_PO2_REST, 4, rest' Pulse noch auszugeben
  ENDWHILE
                                        ' fertig
END
```

RES1

Der Device-Treiber ‘RES1’ hilft während der Entwicklungsphase in Zusammenarbeit mit SET1.TDD, die Belastung der CPU durch die an den TIMERA angekoppelten Device-Treiber festzustellen.

Dateiname: RES1.TDD

INSTALL DEVICE #*D*, "RES1.TDD"

D ist eine Konstante, Variable oder ein Ausdruck vom Datentyp BYTE, WORD, LONG im Bereich von 0...63 und steht für die Gerätenummer des Treibers.

P1 ist ein Parameter, der den zur Testausgabe verwendeten Pin festlegt. Der Parameter besteht aus einer zweistelligen Zahl. Die Zehnerziffer gibt den (internen) Port an, die Einerziffer gibt die Pin-Nummer des Ports an.

Der Device-Treiber RES1.TDD dient in Zusammenarbeit mit SET1.TDD während der Entwicklungsphase dazu, die Belastung der CPU durch die an den TIMERA angekoppelten Device-Treiber festzustellen. Am Anfang jedes Zeit-Ticks des TIMERA setzt der Treiber SET1 den bei der Installation ausgewählten Pin auf 'high'. Am Ende des Zeit-Ticks, d.h. wenn alle angekopplelten Treiber ihre Aufgabe erledigt haben, setzt RES1.TDD den Pin wieder auf 'low'. Mit einem Oszilloskop kann nun die Tätigkeit der TIMERA-abhängigen Device-Treiber beobachtet werden. Die Zeit, die das Signal an dem Pin 'high' ist, wird von den betroffenen Device-Treibern verbraucht. Die Zeit, die das Signal 'low' ist, steht den anderen Treibern und BASIC zur Verfügung.

Hinweis:
a) der verwendete Pin muß im BASIC-Programm als Ausgang initialisiert werden.
b) RES1.TDD wird nach den von TIMERA.TDD abhängigen Device-Treibern eingebunden.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name: RES1.TIG
'--------------------------------------------------------------------
#INCLUDE UFUNC3.INC

TASK MAIN
  LONG REST
  INSTALL_DEVICE #13, "TIMERA.TDD", 1,250 ' 10kHz
  INSTALL_DEVICE #19, "SET1.TDD", 70    ' Setze Pin L70 auf high
  INSTALL_DEVICE #20, "PO2_P80.TDD"     ' Pulse auf Pin L80
  INSTALL_DEVICE #21, "PO2_P81.TDD"     ' Pulse auf Pin L81
  INSTALL_DEVICE #23, "RES1.TDD", 70    ' Setze Pin L70 auf low

  DIR_PIN 7,0,0                         ' setze Testpin als Ausgang

  PUT #20, 400, 12, 24                  ' 400 Pulse 2,4microsec cycle
  PUT #21, #2, 800, 6, 12               ' 800 Pulse 1,2microsec cycle
  REST = 1
  WHILE REST > 0
    GET #20, #0, #UFCI_PO2_REST, 4, rest' Pulse noch auszugeben
  ENDWHILE
                                        ' fertig
END
```

Leere Seite

3 Applikationen

In diesem Kapitel finden Sie Beispielapplikationen, die Ihnen den Einstieg in die Programmierung des BASIC-Tigers® erleichtern sowie Anregungen für eigene Programme geben. Die aufgeführten Programme befinden sich alle auf der mitgelieferten Diskette.

Dieses Kapitel enthält folgende Beispiele:

Plug & Play Lab Tastatur-Anpassung

Dateiname: KEYB_GR2.INC

Includes: keine

Die Anpassung der Plug & Play Lab-Tastatur wird in dem Unterprogramm in der Include-Datei ‘KEYB_GR2.INC’ vorgenommen.

Sie können diese Include-Datei auch in eigenen Anwendungen verwenden. Dazu wird die Datei mit in Ihre Anwendung eingebunden (#INCLUDE) und das Unterprogramm unter Angabe der Device-Nummer ‘INIT_KEYB (Device-Nr)’ aufgerufen . Das Unterprogramm nimmt daraufhin eine Anpassung der Tastatur des Plug & Play Labs an den Device-Treiber vor.

Programmbeispiel:

```
'------------------------------------------------------------------------
' BASIC-Tiger Include Datei:    KEYB_GR2.INC    27.03.1997    v1.1
'------------------------------------------------------------------------
' Einstellungen fuer das  "PLUG & PLAY Lab" des BASIC-Tigers:
'
'               Tastatur-Codes + Tasten-Attribute
'
' Beachte:  --> INSTALL_DEVICE ...  erfolgt im BASIC-Programm
'           --> Aufruf:    CALL INIT_KEYB (Device-Nr)
'------------------------------------------------------------------------

'------------------------------------------------------------------------
' CTRL-Codes mit Bedeutung fuer LCD Device-Driver:   LCD1.TDD
'------------------------------------------------------------------------
#DEFINE _CLR   "<01H>" ' CTRL-'A' = Clear Screen + Cursor Home
#DEFINE _HOME  "<02H>" ' CTRL-'B' = Cursor Home
#DEFINE _FS    "<05H>" ' CTRL-'E' = Cursor rechts
#DEFINE _BS    "<08H>" ' CTRL-'H' = Cursor links
#DEFINE _DO    "<0AH>" ' CTRL-'J' = Cursor runter = <LF>
#DEFINE _UP    "<0BH>" ' CTRL-'K' = Cursor hoch

#DEFINE _LF    "<0AH>" ' CTRL-'J' = Line-Feed = <DO>
#DEFINE _CR    "<0DH>" ' CTRL-'M' = Carriage Return
#DEFINE _FF    "<0CH>" ' CTRL-'L' = Form-Feed

#DEFINE _CLICK "<00H>" ' CTRL-'@' = Tasten-Klick (Prioritaet hoch)
#DEFINE _BELL  "<07H>" ' CTRL-'G' = Standard Ton (Prioritaet mittel)
#DEFINE _ALARM "<14H>" ' CTRL-'T' = Alarm Ton    (Prioritaet niedrig)

#DEFINE _ESC   "<1BH>" ' ESCAPE

#DEFINE _KEIN_CURSOR  "<1BH>c<0><F0H>"  ' <- Cursor-AUS
#DEFINE _KONST_CURSOR "<1BH>c<1><F0H>"  ' <- Cursor-EIN (Underline)
#DEFINE _BLINK_CURSOR "<1BH>c<2><F0H>"  ' <- blinkender Cursor (Block)


'------------------------------------------------------------------------
' dt. Umlaute fuer LCD Device-Driver:   LCD1.TDD
'------------------------------------------------------------------------
#DEFINE _a      "<80H>" ' Umlaut "a" auf LC-Display
#DEFINE _o      "<81H>" ' Umlaut "o"       "
#DEFINE _u      "<82H>" ' Umlaut "u"       "
#DEFINE _UA     "<83H>" ' Umlaut "A"       "
#DEFINE _UO     "<84H>" ' Umlaut "O"       "
#DEFINE _UU     "<85H>" ' Umlaut "U"       "


'------------------------------------------------------------------------
' ESC-Sequenzen mit Bedeutung fuer LCD Device-Driver:   LCD1.TDD
'------------------------------------------------------------------------
#DEFINE _MENU_X "<1BH>M";CHR$(X);"<F0H>";         ' Menue-Selektion
#DEFINE _POS_XY "<1BH>A";CHR$(X);CHR$(Y);"<F0H>"; ' Cursor Position


'------------------------------------------------------------------------
' Subroutine:   setze Tasten-Codes + Attribute + div.
'------------------------------------------------------------------------
'
```

```
'                = Device-Nr des LCD-/Keyboard Device-Treibers
'-------------------------------------------------------------------------
SUB INIT_KEYB (LONG DEV_NR)                '
STRING A$ (128)                            ' String mit max 128 Zeichen

A$="&                                      ' Tasten-Codes UN-Shifted
6B6C8180230890916D2C2E2D00000A20&          ' 00..0F
100D00000000000000000000000000000&         ' 10..1F
00000000000000000000000000000000&          ' 20..2F
00000000000000000000000000000000&          ' 30..3F
1BF1F2F3F4F5F6F75E31323334353637&          ' 40..4F
1D71776572747A750F6173646667686A&          ' 50..5F
003C79786376626EF8F9FA00001AFBFC&          ' 60..6F
38393086277FFDFE696F70822B080B05"%         ' 70..7F
PRINT #DEV_NR, "<1BH>Z";A$;"<F0H>";        ' Key-Codes UN-Shifted setzen

A$="&                                      ' Tasten-Codes Shifted
4B4C84832708B0B14D3B3A5F00000A20&          ' 00..0F
100D00000000000000000000000000000&         ' 10..1F
00000000000000000000000000000000&          ' 20..2F
00000000000000000000000000000000&          ' 30..3F
1BF1F2F3F4F5F6F7B42122862425262F&          ' 40..4F
1D51574552545A550F4153444647484A&          ' 50..5F
003E59584356424EF8F9FA00001AFBFC&          ' 60..6F
28293D3FB37FFDFE494F50852A080B05"%         ' 70..7F
PRINT #DEV_NR, "<1BH>z";A$;"<F0H>";        ' Key-Codes Shifted setzen

A$="&                                      ' Tasten-Attribute
00000000000000000000000003000000&          ' 00..0F
00000000000000001000000000000000&          ' 10..1F
00000000000000000000000000000000&          ' 20..2F
00000000000000000000000000000000&          ' 30..3F
04000000000000000000000000000000&          ' 40..4F
00000000000000002000000000000000&          ' 50..5F
03000000000000000000000000000000&          ' 60..6F
00000000000000000000000000000000"%         ' 70..7F
PRINT #DEV_NR, "<1BH>a";A$;"<F0H>";        ' Tasten-Attribute setzen

PRINT #DEV_NR, "<1BH>K<0><F0H>";           ' Tasten-Klick: 0 = ON
PRINT #DEV_NR, "<1BH>r<20><5><F0H>";       ' Tasten-Repeat
PRINT #DEV_NR,&                            ' Tasten/DIP-Schalter
"<1BH>D<16><1><1><1><1><1><1><0><0><1><1><1><1><1><1><1><1><F0H>";
END                                        ' = RETURN from Sub


'-------------------------------------------------------------------------
' for subroutine 'get_string1'
'-------------------------------------------------------------------------
#define FLD_BS 08h                  ' back space = delete char
#define FLD_CAN 15h                 ' cancel field = clear + repeat
#define FLD_RET 0dh                 ' return = enter
#define FLD_ESC 1bh                 ' ESC = quit, return empty or curr_ent
#define FLD_MF 1ch                  ' move forward
#define FLD_MB 1dh                  ' move backward
#define FLD_INS 1eh                 ' toggle insert/overwrite
```

Systemparameter abfragen „VERSION.TIG“

Dateiname: VERSION.TIG

Includes: DEFINE_A, UFUNC3

Device-Treiber: LCD1.TDD

Dieses Programm fragt einige Systemvariable und zeigt sie auf dem LC-Display an:

- BASIC-Tiger®-Modul-Version
- BASIC-Tiger®-Modul-Typ
- Größe des Flash-Speichers insgesamt
- Größe des Flash-Speichers für Daten

Programmbeispiel:

```
'-----------------------------------------------------------------------
' Name: VERSION.TIG
'-----------------------------------------------------------------------
USER_VAR_STRICT
#INCLUDE DEFINE_A.INC                   ' Allgemeine Definitionen
#INCLUDE UFUNC3.INC

TASK MAIN                               ' Beginn Task MAIN
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #LCD, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #LCD, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8

  STRING VRS$                           ' Variablendeklaration
  CALL GET_DEV_VERS ( VRS$ )            ' Rufe Unterprogramm
'
' Modul-Version, -Typs, FLASH- und User-FLASH-Groesse ausgeben
'
  PRINT #LCD, "Version: "; VRS$
  PRINT #LCD, " Module:"; SYSVARN ( TIGER_MODULE, 2 )
  PRINT #LCD, "  Flash:"; SYSVARN ( FLASH_GSIZE, 9 ) / 1024; " KB"
  PRINT #LCD, "  Flash:"; SYSVARN ( FLASH_DSIZE, 9 ) / 1024; " KB"
END                                     ' Ende Task MAIN

'-----------------------------------------------------------------------
' Subroutine: Erzeuge Versions-String
'-----------------------------------------------------------------------
SUB GET_DEV_VERS ( VAR STRING V$ )      ' Beginn Subroutine
  LONG V                                ' Variablendeklaration
  STRING C$                             '
  V = SYSVARN ( TIGER_VERS, 0 )         ' Lies Version als LONG
  V$ = STRI$ (V,"UH<4><4>   0.0.0.0.0") ' Unwandeln in String
  C$ = RIGHT$ ( V$, 1 )                 ' Bestimme Versions-
  V = ASC ( C$ ) + 42                   ' Buchstaben
  V$ = "V" + LEFT$ ( V$, 1 ) + "." +&   ' Setze Versions-String
MID$ ( V$, 1, 2 ) + CHR$ ( V )          ' zusammen
END                                     ' Ende Subroutine
```

Scan-Codes „KEY_NO.TIG“

Dateiname: KEY_NO.TIG

Includes: keine

Device-Treiber: LCD1.TDD

Dieses Programm zeigt die gescannten Tastatur-Codes als dezimale und als hexadezimale Zahl auf dem LC-Display an.

Die Codes entsprechen also der Position in den Code-Tabellen. Eigene Tastaturen lassen sich so leicht an den im Device-Treiber LCD1.TDD enthaltenen Tastaturtreiber anpassen.

Programmbeispiel:

```
'-----------------------------------------------------------------------
'  Name: KEY_NR.TIG
' Stand: 05.12.1996
' Zweck: zeigt Tasten  N u m m e r n  wie sie vom Device-Driver
'        "LCD1.TDD" gescannt werden.
'-----------------------------------------------------------------------
TASK MAIN                                ' Beginn Task MAIN
' LCD-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #1, "LCD1.TDD"
' LCD-Treiber installieren (TINY-Tiger)
'  INSTALL DEVICE #1, "LCD1.TDD", 0, 0, 0, 0, 0, 0, 80h, 8

  USING "UD<4><1>   0.0.0.0.4UH<2><2>   0.0.0.0.2"   ' Format-String
  PRINT #1, "<1>==== KEY_NR.TIG ====";  '
  FOR X=0 TO 0 STEP 0                    ' Endlosschleife
    FOR N=0 TO 0 STEP 0                  ' Endlosschleife bis N=1(GET!)
      RELEASE_TASK                       ' Rest der Task-Zeit freigeben
      GET #1, #0, #1, 1, N               ' N=Zeichen in Tastatur-Buffer
    NEXT                                 ' Ende Endlosschleife
    GET #1, 1, A$                        ' Tastatur-Buffer auslesen
    PRINT #1, "<2><10>Key-Nr =";         ' Ausgabe auf LC-Display
    PRINT USING #1, ASC(A$);" ($";ASC(A$);")"   ' zeige Tasten-Nr
  NEXT                                   ' Ende Endlosschleife
END                                      ' Ende Task MAIN
```

Sonderzeichen „LCD_SPCC.TIG“

Dateiname: LCD_SPCC.TIG

Includes: keine

Device-Treiber: LCD1.TDD

Alle 13 Jumper der Stiftleiste J22 auf dem Plug & Play Lab müssen stecken.

Dieses Programm stellt auf interessante Weise die momentan definierten Sonderzeichensätze auf dem LC-Display dar. In diesem Fall sind es die im Device-Treiber vordefinierten 16 Zeichensätze.

Die Task ‘Main’

- startet die Task ‘LIVING_CODE’.
- zeigt welcher Sonderzeichensatz gerade ausgewählt.
- stellt in der Zeile 3 die 8 Zeichen des Zeichensatzes dar.
- ruft das Unterprogramm ‘WARTE_TASTE_SWITCH (SONDER)’ auf, um ein Umschalten auf den nächsten Zeichensatz zu ermöglichen.

Die Task ‘LIVING_CODE’ druckt in einer Endlosschleife nacheinander die 8 Sonderzeichen an die gleiche Stelle des Displays. Diejenigen Zeichensätze, die für eine fließende Darstellung entworfen wurden, beginnen dadurch ‘zu leben’. Das trifft auf alle Zeichensätze für Balkendiagramme zu und auf den rotierenden Strich.

Das Unterprogramm ‘WARTE_TASTE_SWITCH (SONDER)’ inkrementiert oder dekrementiert entsprechend der Eingabe die Variable ‘Sonder’ jeweils um 8 und schaltet so einen Sonderzeichensatz weiter oder zurück.

‘WAIT_FOR_1CHAR’ **wartet** auf die Eingabe einer Taste.

Zeichensätze „LCD_SPC4.TIG“

Dateiname: LCD_SPC4.TIG

Includes: KEYB_PP.INC

Device-Treiber: LCD1.TDD

Alle 13 Jumper der Stiftleiste J22 auf dem Plug & Play Lab müssen stecken.

Dieses Programm demonstriert einige Anwendungen der im Device-Treiber vordefinierten 16 Zeichensätze des LC-Displays. Um die Tastatur des Plug & Play Labs zu verwenden, wird zunächst die Datei KEYB_PP.INC eingebunden. In dieser Datei sind die Initialisierungs-Strings für den Device-Treiber sowie das Unterprogramm ‘INIT_KEYB (Gerätenr.)’ definiert. Die Task ‘Main’ ruft ‘INIT_KEYB (LCD)’ auf, wodurch der LCD1-Treiber an die Tastur des Plug & Play Labs angepasst wird.

Die Variable ‘MAIN_SEL$’ enthält das Menü, welches in diesem Programm verteilt über 3 Zeilen dargestellt wird. Mit den Pfeiltasten wird das Menü nach oben oder unten verschoben, während der Auswahlzeiger fest in Zeile 2 bleibt.

Das Unterprogramm ‘SELECT’ erzeugt zusammen mit dem Unterprogramm ‘WARTE_TASTE_SEL’ diese Art der Menü-Darstellung und liefert nach gedrückter Auswahltaste die Nummer des ausgewählten Menüpunktes an die Task ‘MAIN’ zurück. Das Unterprogramm ‘WRAP’ achtet darauf, daß der Wertebereich der Auswahl eingehalten wird (hier 0...5). Die Auswahl wird in einer IF...ELSE IF-Kette weiterverarbeitet. Das Programm verzweigt in Unterprogramme, die jeweils die Demonstartions-Task starten und auch selbst bei Tastendruck wieder beenden.

Oszilloscope

‘OSZILLOSCOPE’ startet die Task ‘OSCILLO’, die in einer Endlosschleife eine Oscilloscope-Anzeige auf dem LC-Display demonstriert. Währenddessen wartet ‘OSZILLOSCOPE’ durch den Aufruf von ‘WAIT_FOR_1CHAR’ auf Tasteneingabe. Nach erfolgter Eingabe wird die Demonstrations-Task beendet, und die Task ‘MAIN’ baut das Menü wieder auf.

‘OSCILLO’ selbst berechnet 20 Sinuswerte und weist jedem Wert abhängig von der Zeile des Displays ein Sonderzeichen zu. Der benutzte Sonderzeichensatz ist im Device-Treiber ‘LCD1’ bereits enthalten.

Spektrum

‘SPECKT’ funktioniert prinzipiell genauso wie ‘OSCILLO’, jedoch wird ein anderer Sonderzeichensatz verwendet und die angezeigten Werte sind Zufallswerte.

Dünne Linien

Die Task ‘DUENN’ zeigt, wie der Sonderzeichensatz verwendet werden kann. Das Unterprogramm ‘DRAW_BOX’ zeichnet einen Rahmen aus dünnen Linien.

Dann läßt ‘LAUF_TEXT’ einen Text in dem Rahmen laufen, indem der String in A$ immer wieder neu zusammengesetzt und der linke Teil mit der passenden Laenge in die Box gedruckt wird.

Dicke Linien

Die Task ‘DICK’ zeichnet mehrere Rahmen aus dicken Linien und läßt dann ebenfalls einen Text in einer Box laufen.

Pfeile

Die Task ‘ARR’ zeigt rechts und links des angezeigten Textes große Pfeile, die blinkend mit einem grafisch gemusterten Feld abwechseln. Der Text wird mit zwei PRINT-Instruktionen ausgegeben und der auf dem LC-Display stattfindende Zeilenumbruch ausgenutzt. Sowohl die Pfeile als auch das gemusterte Feld bestehen aus einem Feld von 4 Zeichen Breite über die 4 Zeilen des LC-Displays. Wie im folgenden Beispiel werden auch hier grafische Elemente aus einem Sonderzeichensatz des Device-Treibers ‘LCD1’ verwendet.

Das Unterprogramm ‘PRINT_DIGIT’ druckt das gesamte Feld an der angegebenen Spalte auf das LC-Display.

Big Numbers (Große Ziffern)

Die Task ‘BIG_NUM’ verwendet ein Feld von 4 Zeichen Breite über die 4 Zeilen des LC-Displays, um große Ziffern anzuzeigen. Jede Ziffer ist in dem String ‘CHGEN$’ vorbereitet und besteht aus grafischen Elementen, die in einem Sonderzeichensatz des Device-Treibers ‘LCD1’ zur Verfügung stehen. In ‘CHGEN$’ sind die im Unterprogramm ‘PRINT_DIDGIT’ zu verwendenden Sonderzeichen 0...7 oder ‘Leerzeichen’ aufgeführt. Jeweils 16 Zeichen in CHGEN$ beschreiben also ein 4x4-Feld einer großen Ziffer. Das Unterprogramm ‘PRINT_DIGIT’ druckt das gesamte Feld an der angegebenen Spalte auf das LC-Display.

Seriell-I/O „SER1_DEM.TIG“

Dateiname: SER1_DEM.TIG

Includes: KEYB_PP.INC, DEFINE_A.INC, DEFINE_I.INC

Device-Treiber: LCD1.TDD, SER1B_K1.TDD

Alle 13 Jumper der Stiftleiste J22 auf dem Plug & Play Lab müssen stecken.

Dieses Programm verwendet in 7 Tasks die beiden seriellen Schnittstellen sowohl zur Ausgabe als auch zur Eingabe, zeigt auf dem LC-Display die seriell empfangenen Zeichen sowie Tastatureingaben und einen Zähler an. Nebenbei blinken die Leuchtdioden an Port 8.

Zunächst werden die Definitionen zur Anpassung des Device-Treibers an die Tastatur de Plug & Play Labs, allgemeine Definitionen außerhalb und innerhalb der Task ‘MAIN’ und User-Funktionen eingebunden. Bei der Installierung des seriellen Treibers werden die Schnittstellenparameter festgelegt. Als Kommentar ist im Listing ist eine alternative Schreibweise zur Installation des Treibers angegebn. Ebenfalls auskommentiert ist die mögliche Veränderung der Übertragungsparameter der seriellen Schnittstellen während des Programmlaufs mit der Instruktion ‘PUT’. In jedem Fall werden die leichter lesbaren in den INCLUDE-Dateinen vordefinierten Parameter anstelle der nichts-aussagenden Nummern verwendet (‘BD_19_200’ statt ‘16’).

Die Task ‘MAIN’ startet die anderen Tasks und sorgt dann für das Blinken der LEDs an Port 8 und zählt dabei ‘N’ als Variable der FOR...NEXT-Schleife hoch.

Die Task ‘LCD_DISPLAY’ zeigt den Wert der globalen Variablen ‘N’ an.

‘SER1_OUT’ sendet laufend den Text „Hallo Welt“.

‘SER0_OUT’ berechnet zwei Sinuswerte, die mit 39 multipliziert eine Tabulatorposition vorgeben, an der dann ‘*’ bzw. ‘#’ ausgegeben wird. Da der Sinuswert auch negativ wird, ist der Nullpunkt der TAB-Funktion an die Stelle 40 verlegt. Schließen Sie ein Terminal oder einen PC mit Terminal-Programm an SER0 an, und Sie sehen als Ergebnis zwei endlos wandernde Sinuskurven, die sich in Schwebung befinden.

Die Tasks ‘SER0_IN’ und ‘SER1_IN’ benutzen beide (!) das Unterprogramm APPEND, um Zeichen aus den Empfangspuffern in Zeile 1 und Zeile 2 anzuzeigen.

Das Unterprogramm ‘APPEND’ kann von beiden Tasks verwendet werden, weil in Tiger-BASIC® Unterprogramme für jeden Aufruf einen getrennten Satz von Variablen anlegen.

Auch die Ausgabe auf das LC-Display von mehreren Tasks muß gut durchdacht sein. Entweder verwendet jede Task nur **eine** PRINT-Instruktion zur Ausgabe, um bei der Ausgabe nicht Unterbrochen zu werden, oder jede PRINT-Instruktion muß den

Cursor am Anfang auf die richtige Position setzen. Nicht-Beachten dieser Regeln wird durch Ausgabe-Salat auf dem Display bestraft.

Die Task ‘KEY_IN’ liest ebenfalls mit Hilfe von ‘APPEND’ Zeichen aus dem Tastaturpuffer und zeigt sie auf dem LC-Display in Zeile 3 an.

Aufgabe des Unterprogrammes ‘APPEND’ ist neben der Zusammensetzung des Ausgabestrings auch, darauf zu achten, daß keine Steuerzeichen in den String gelangen.

Analogkanäle zeigen „ANA1_DEM.TIG“

Dateiname: ANA1_DEM.TIG

Includes: KEYB_PP.INC

Device-Treiber: LCD1.TDD, ANALOG1.TDD

Alle 13 Jumper der Stiftleiste J22 auf dem Plug & Play Lab müssen stecken. Pin L42 wird mit dem Pin ‘beep’ und Analogeingang ’An3’ mit dem Pin ‘micro’ auf der 9-poligen Stiftleiste im Ananlogbereich verbunden. Auf J13 steckt der Jumper ganz links, so daß das Mikrofon auf dem Eingang des Mikrofonverstärkers liegt. Stellen Sie das Poti vor J13 auf Maximum. Das Programm ‘ANA1_DEM.TIG’ zeigt ein Analogsignal auf verschiedene Weise auf dem LC-Display an. Nach dem Start des Programms wählen Sie zunächst den gewünschten Analog-Eingang aus.

Im folgenden Durchroll-Menü haben Sie folgende Wahlmöglichkeiten:

Auswahl	Funktion
Slow Sampling	Messen mit 6S/sec, Darstellung als Oszilloskop
Fast Sampling	Messen mit 333S/sec, Darstellung als Oszilloskop
Numerisch	Darstellung in großen Ziffern
Linear-Test	Meßwerte werden integriert
Akustik-Pegel	Darstellung als laufendes Balkendiagramm

Die Task ‘MAIN’ bindet zunächst die Device-Treiber ein und initialisiert diverse Variable sowie den LCD1-Treiber für die deutsche Tastatur des Plug & Play Labs. Dann findet die Auswahl des Analogkanals statt (außer Funktion ‘Akustik-Pegel: fest auf ‘An3’). Das Unterprogramm ‘SELECT1’ sorgt dafür, daß die Auswahlindex immer zwischen 0 und dem für das entsprechende Menü geltenden Maximalwert bleibt. Auch die folgende Auswahl der Funktion bedient sich dieses Unterprogramms. Die Verzweigung in das Unterprogramm, welches die ausgewählte Funktion ausführt, findet in einer IF...ELSE-Kette statt.

Slow Sample

‘CONT_OSZILLOSCOPE’ startet die Task ‘CONT_OSCILLO’, die in einer Endlosschleife eine Oscilloscope-Anzeige auf dem LC-Display realisiert. Währenddessen wartet ‘CONT_OSZILLOSCOPE’ durch den Aufruf von ‘WAIT_FOR_1CHAR’ auf Tasteneingabe. Nach erfolgter Eingabe wird die Demonstrations-Task beendet, und die Task ‘MAIN’ baut das Menü wieder auf.

‘CONT_OSCILLO’ selbst initialisiert LC-Display und das Zeitraster und liest dann einen Meßwert ein. Der Meßwert wird auf 31 begrenzt, da das LC-Display 32 Zeilen hat. Der Meßwert wird an die Position 20 in die Array-Variable ‘OSZ’ eingefügt. Die

Spalten des LC-Displays zählen von 0 bis 19. Das Unterprogramm ‘PRINT_OSZ’ schiebt den Inhalt von ‘OSZ’ um eins nach links und sorgt für die Anzeige auf dem LC-Display.

‘PRINT_OSZ’ weist jeder Position des Displays ein Sonderzeichen abhängig vom Wert in ‘OSZ’ zu. Der benutzte Sonderzeichensatz ist im Device-Treiber ‘LCD1’ bereits enthalten. Dann wird der gesamte Inhalt des LC-Displays in einer PRINT-Instruktion gedruckt.

Fast Sample

‘SAMPLE_OSZILLOSCOPE’ startet die Task ‘SAMPLE_OSCILLO’, die in einer Endlosschleife eine Oscilloscope-Anzeige auf dem LC-Display realisiert. Währenddessen wartet ‘SAMPLE_OSZILLOSCOPE’ durch den Aufruf von ‘WAIT_FOR_1CHAR’ auf Tasteneingabe. Nach erfolgter Eingabe wird die Demonstrations-Task beendet, und die Task ‘MAIN’ baut das Menü wieder auf.

‘SAMPLE_OSCILLO’ selbst initialisiert LC-Display und das Zeitraster und liest dann 20 Meßwerte ein. Die Meßwerte werden auf 31 begrenzt, da das LC-Display 32 Zeilen hat. Die Meßwerte werden an die Positionen 1 bis 20 in die Array-Variable ‘OSZ’ eingefügt. Die Spalten des LC-Displays zählen von 0 bis 19. Das Unterprogramm ‘PRINT_OSZ’ schiebt den Inhalt von ‘OSZ’ um eins nach links und sorgt für die Anzeige auf dem LC-Display.

‘PRINT_OSZ’ weist jeder Position des Displays ein Sonderzeichen abhängig vom Wert in ‘OSZ’ zu. Der benutzte Sonderzeichensatz ist im Device-Treiber ‘LCD1’ bereits enthalten. Dann wird der gesamte Inhalt des LC-Displays in einer PRINT-Instruktion gedruckt.

Numerisch

‘ANALOG_BIG_NUMBERS’ startet die Task ‘AN_BIG_NUM’, die in einer Endlosschleife eine numerische Anzeige auf dem LC-Display realisiert. Währenddessen wartet ‘ANALOG_BIG_NUMBERS’ durch den Aufruf von ‘WAIT_FOR_1CHAR’ auf Tasteneingabe. Nach erfolgter Eingabe wird die Task ‘AN_BIG_NUM’ beendet, und die Task ‘MAIN’ baut das Menü wieder auf.

‘AN_BIG_NUM’ selbst initialisiert LC-Display und das Zeitraster sowie den String, mit dessen Hilfe später die großen Ziffern generiert werden. Es wird ein Meßwert eingelesen und die einzelnen Ziffern nacheinander an ‘PRINT_DIGIT’ zusammen mit der Position, an der die Ziffer auf dem Display erscheinen soll, übergeben.

‘PRINT_DIGIT’ verwendet einen der Sonderzeichensätze des Device-Treibers ‘LCD1’, um die großen Ziffern aufzubauen. Dazu geben je 16 Ziffern aus dem Zeichengenerator-String CHGEN$ die Indize innerhalb des Sonderzeichensatzes vor, oder es wird ein Leerzeichen benötigt. In einem Feld von 4 Zeichen Breite über die 4 Zeilen des LC-Displays werden die Ziffern mit 16 Sonderzeichen und Leerzeichen aufgebaut.

Linear-Test

Diese Test-Funktion erzeugt einen Meßwertsatz von 20 Werten, der linear ansteigt und angezeigt wird. Es wird nicht von einem Analog-Eingang eingelesen.

Akustik-Pegel

‘AKUSTIK_LEVEL’ startet die Tasks ‘AKUSTIK_LEV’ und ‘SHOW_OSZ’, die jeweils in Endlosschleifen die Meßwertaufnahme und die Anzeige auf dem LC-Display realisieren. Währenddessen wartet ‘AKUSTIK_LEVEL’ durch den Aufruf von ‘WAIT_FOR_1CHAR’ auf Tasteneingabe. Nach erfolgter Eingabe werden die Tasks ‘AKUSTIK_LEV’ und ‘SHOW_OSZ’ beendet, und die Task ‘MAIN’ baut das Menü wieder auf.

Die Task ‘AKUSTIK_LEV’ holt zunächst 8 Meßwerte vom Eingang ‘An3’ und stellt die Amplitude als Differenz zwischen minimalem und maximalem Wert fest. Der Mittelwert von 256 Amplitudenwerten stellt den momentan anliegenden Schallpegel dar. Der so ermittelte Wert wird in der globalen Variablen _NEXT_OSZ_WERT ablegt.

Die Task ‘SHOW_OSZ’ initialisiert LC-Display und das Zeitraster und fügt 8 mal in der Sekunde den Wert _NEXT_OSZ_WERT in die Array-Variable OSZ an der Position 20 ein. Das Unterprogramm ‘PRINT_OSZ’ schiebt den Inhalt von ‘OSZ’ um eins nach links und sorgt für die Anzeige auf dem LC-Display.

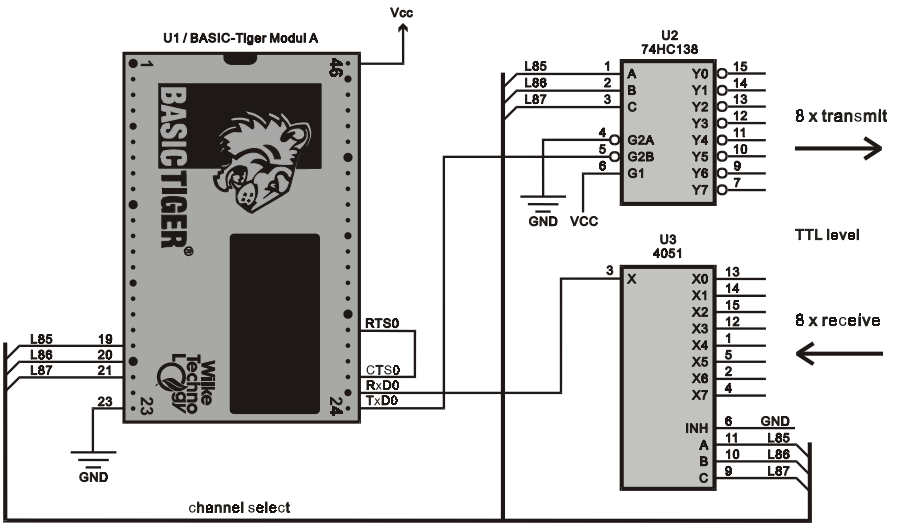
Serielle Kanäle umschalten „8X_SER1.TIG“

Dateiname: 8X_SER1.TIG

Includes: UFUNCn.INC

Device-Treiber: SER1B_K1.TDD

Um aus einer seriellen Schnittstelle 8 umschaltbare serielle Kanäle zu erhalten, benötigen Sie lediglich 2 preiswerte externe integrierte IC für die Sende- und Empfangsleitung.



Auf die Empfangsleitung wird über einen Analogschalter einfach eine der 8 externen Leitungen durchgeschaltet. Die Sendeleitung aktiviert die selektierte Leitung des 74HC138 sobald ein Bit mit TTL-Level ‘0’ gesendet wird. Die selektierte Leitung geht daraufhin auch auf ‘0’, während die nicht selektierten Kanäle im Ruhezustand (TTL ’1’) verbleiben. In der Beispielschaltung sind RTS und CTS gebrückt. Sofern die Handshake-Leitungen benötigt werden, ist die gleiche Schaltung noch einmal erforderlich.

Die folgende Applikation geht davon aus, über den seriellen Kanal 1 die Aufforderung zu erhalten, mit einem bestimmten Kanal auf der seriellen Schnittstelle 0 Kontakt aufzunehmen und einen Datenverkehr abzuwickeln. Zum Beispiel könnten sich an den 8 seriellen Anschlüssen Waagen oder ähnliche Meßinstrumente befinden, die nach Ihrem Meßwert befragt werden. Die Meßdaten werden daraufhin über die serielle Schnittstelle 1 weitergegeben.

Programmbeispiel:

```
'--------------------------------------------------------------------
'  Name: 8XSER1.TIG
' Stand: 03.12.1997
'   Typ: Tiger-BASIC Source Code
' Zweck: Beispielanwendung fuer 8 serielle Kanaele an SER0,
'        ueber SER1 werden Messwerte von einem der 8 Geraete,
'        die an der erweiterten SER0 angeschlossen sind,
'        abgefragt.
'
' sonst: Device-Driver "SER1.TDD"
'--------------------------------------------------------------------
'
' Benutzen Sie z.B. das "Plug + Play Lab" und verbinden Sie:
'
'               SER1 mit Terminal A oder einem PC im Terminal-Mode
'                    (Applikation laeuft nur im RUN-Mode)
'               SER0 ueber die Beispielschaltung mit 8 externen
'                    Terminals oder sonstige Geraete.
'                    Die Protokolle muessen ggfls. angepasst werden.
'
'
' Compilieren und laden Sie dieses Programm in den Tiger.
' Schalten Sie PC-Mode aus, RESET-Taste (Programm startet).
' Geben Sie am Terminal A die Aufforderung ein, z.B.: "read 3".
' Auf dem externen Terminal 3 erscheint nun die Abfrageaufforderung.
' Simulieren Sie die Antwort mit der Eingabe eines Messwertes.
' Der Messwert wird auf SER 1 an Terminal A weitergegeben.
'
' Die Protokolle sind hier Klartexte:
' Terminal A an Tiger: "ask 3<CR>"      <-- frage Geraet 3
' Tiger an Geraet:     "Wert 3<CR>"     <-- Aufforderung: sende Wert
' Geraet an Tiger:     " 23456<CR>"     <-- Wert wird gesendet
' Tiger an Terminal A: " 3: 23456<CR>"  <-- Wert an Terminal A
'
'--------------------------------------------------------------------
USER VAR STRICT                         ' Forciere Var-Deklaration

#INCLUDE DEFINE_A.INC                   ' allgemeine Definitionen
#INCLUDE UFUNC2.INC                     ' User Function Codes

#DEFINE SCOM 1                          ' serieller Kanal "common"
#DEFINE S8CH 0                          ' serieller Kanal "8 channels"

'--------------------------------------------------------------------
' globale Variablen
'--------------------------------------------------------------------
WORD M                                  ' Messwert

'--------------------------------------------------------------------
' Haupttask
'--------------------------------------------------------------------
TASK MAIN                               ' Beginn Task MAIN
  LONG I, R1                            ' einige lokale Variablen
  STRING R1$, A$
  INSTALL_DEVICE #SER, "SER1.TDD",&
BD_19_200, DP_8N, JA, BD_19_200, DP_8N, JA

  DIR_PORT 8, 00011111b
```

```
  FOR I = 0 TO 0 STEP 0                  ' Endlosschleife
    INPUT_LINE #SER, #SCOM, R1$          ' warte auf Abfrageanforderung
    PRINT #SER, #SCOM, R1$               ' ??? sende Anforderung an Term.
    IF LEFT$ (R1$, 4) = "ask " THEN      ' Falls Anforderung = "ask"
      R1$ = RIGHT$ (R1$, LEN (R1$) - 4) ' Extrahiere Device-Nummer
      R1 = VAL_NUM (R1$)                 ' nach R1
    ELSE                                 ' sonst:
      PRINT #SER, #SCOM, "f:";R1$        ' sende Fehler an Terminal A
      R1 = -1                            ' setze Device-Nummer = -1
    ENDIF                                '
    IF R1 > 0 AND R1 < 9 THEN            ' Falls Device-Nummer = 1..8
      R1 = R1 SHL 5                      '
      OUT 8, 11100000b, R1               '
      PRINT #SER, #S8CH, "Wert";R1       ' Sende Wertanfrage an Device
      INPUT_LINE #SER, #S8CH, A$         ' Empfange Wert von Device
      PRINT #SER, #SCOM, R1;":";A$       ' Sende Wert an Terminal A
    ENDIF                                '
    R1$ = ""                             ' loesche Transport-String
  NEXT                                   ' Ende der Endlosschleife
END                                      ' Ende Task MAIN
```

Schrittmotor mit PLSO2

Dateiname: PLSO2_STEPPER.TIG

Includes: -

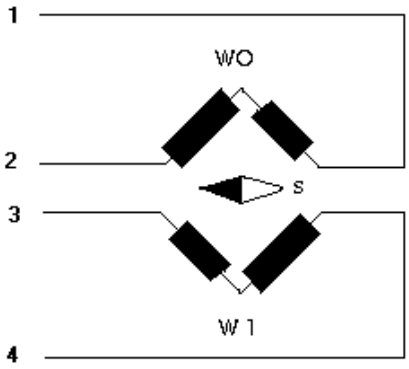
Device-Treiber: TIMERA.TDD, PLSO2_xx.TDD(, SER1B_K1.TDD)

Viele Maschinen und Produktionseinrichtungen verlangen heute einen leistungsfähigen Antrieb für das Verfahren bestimmter Einrichtungen und Werkstücke. In diesen Bereichen setzt sich der Schrittmotor immer weiter durch, unterstützt auch durch das vermehrte Aufkommen von Robotereinrichtungen, die ohne solche Antriebe nicht denkbar wären.

Der Schrittmotor bietet den Vorteil der exakten Positionierbarkeit, der bestimmbaren Geschwindigkeit und der regelbaren Stärke (selbst bei Stillstand). Dies alles läßt sich ohne Rückmeldeeinrichtung realisieren.

Durch die Zerlegung der gewünschten Bewegung in einzelne Schritte lassen sich dynamische Verhaltensweisen programmieren, die mit keinem anderen Antriebssystem realisierbar sind. Man gibt immer sinnvoller Weise die Anzahl der Schritte für eine Umdrehung an. Typisch sind 36 bzw. 100 Schritte pro Umdrehung, was einem Schrittwinkel von 10° bzw. 3,6° zur Folge hat.

Grundsätzlich kann man sagen, dass ein bipolarer Schrittmotor aus einen beweglichen Magnetkern und aus zwei feststehende Spulen besteht. Nun kann man die Ausrichtung des Kerns mit Hilfe der Polarität des Stromes in den Spulen beeinflussen.



Die Polung der Wicklungen wird für jeden Schritt geändert, so daß Ströme in unterschiedlichen Richtungen fließen und im Motor entsprechende magnetische Kräfteverhöltnisse herschen:

	W0		W1	
Schritte	1	2	3	4
1	+	-	+	-
2	-	+	+	-
3	-	+	-	+
4	+	-	-	+

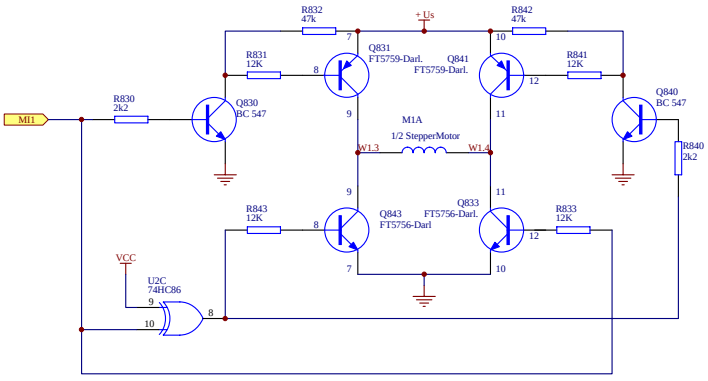
Wenn ein Motor wie oben dargestellt diese 4 Schritte durchlaufen hat, dann hat er eine komplette Umdrehung gemacht. Der Schrittwinkel würde also 90° betragen. Je nach Wicklungsaufbau brauchen gängige bipolare Schrittmotoren 36...100 Schritte, um eine volle Umdrehung zu machen.

Schaltung

Diese Steuerschaltung hat nun die Aufgabe die Spulen des Schrittmotors mit Strom in der richtigen Richtung zu versoregn. Sie besteht aus einem Zähler, zwei Endstufen und einer Strombegrenzung vor allem für den Ruhezustand des Motors, wenn nur die Gleichstromwiderstände der Spulen wirksam sind.

Treiberstufe für die Wicklungen

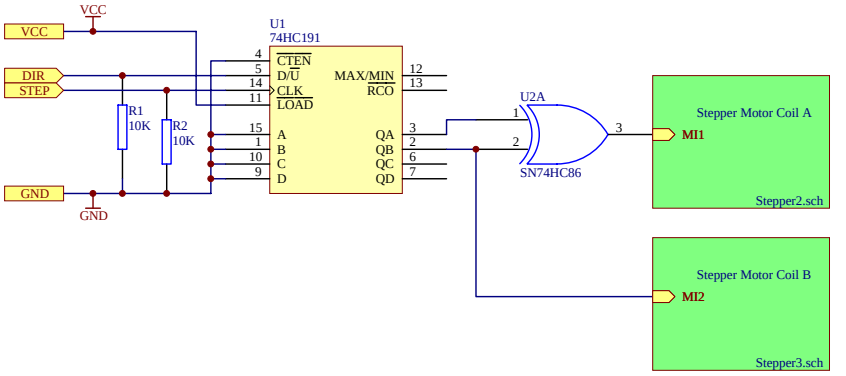
Die Endstufe versorgt die Wicklungen in der richtigen Richtung mit Strom. Der 74HC86 wirkt hier als Inverter. Wenn der Eingang der Schaltung ‚low' ist, dann sperren die Transistoren Q830, Q831 und Q833. Die anderen Transistoren leiten, so daß der Strom im Schaltbild von rechts nach links durch die Wicklung fließt. Wenn der Eingang der Schaltung ‚high' ist, dann ergibt sich genau der umgekehrte Zustand. Die Stromrichtung wird durch die zwei logischen Zuständen beeinflusst.



Steuert man die andere Wicklung ebenfalls auf diese Weise, dann braucht man sich nur noch über die richtige Reihenfolge Gedanken machen. Gehen wir doch einfach mal alle Kombinationen durch:

Zähler 74HCT191

Der Zähler hat die Aufgabe, bei eingehenden Impulsen an den Ausgängen die Schrittmuster für die beiden Endstufen und damit für die Wicklungen zu sorgen. Über Pin5 wird die Zählrichtung vorgeben. Damit kann das Bitmuster an den Ausgängen vorwärts und rückwärts durchlaufen werden.

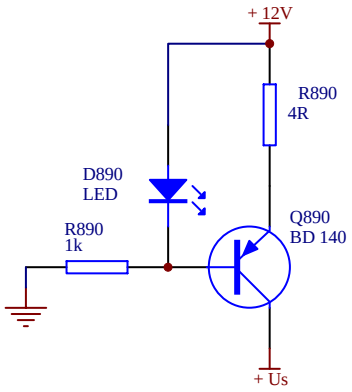


Mit dem XOR-Gatter ergeben sich folgende ‚high'-‚low'-Zustände an den Eingängen der beiden Treiberstufen:

QB	QA	MI1	MI2
0	**0**	0	0
0	**1**	1	0
1	**0**	1	1
1	**1**	0	1

Strombegrenzung

Wenn der Motor sich dreht, gibt es eine Gegeninduktion, die den fließenden Spulenstrom mit ansteigender Drehzahl verkleinert. Steht der Motor jedoch, so wirkt nur noch der ohmische Widerstand. Die Folge ist ein wesentlich höherer Strom beim Stillstand. Daher ist es sinnvoll oder sogar notwendig, den Strom zu begrenzen. Der Transistors muß entsprechend dem Motor genügend Strom liefern können und entsprechend gekühlt werden. Steigt der Spulenstrom an, so steigt die Spannung an dem Begrenzungswiderstand (hier 4Ohm) ebenfalls. Wenn an dem Widerstand soviel Spannung abfällt, wie die Leuchtdioden-Flußspannung, dann fängt der Transistor an zu sperren und begrenzt so den Strom.



Dieser Widerstand wird so gewählt, dass die Schaltung genug Strom zum Laufen liefert, aber deutlich den Strom beim Stillstand begrenzt.

Software

Diese Applikation zeigt, wie ein Bi-Polarer Schrittmotor mit einem Richtungs-Auswahlpin (normaler I/O-Pin) und einem Pulsausgabepin gesteuert wird. Gefordert ist ein langsames Anfahren mit einer Beschleunigungsphase in mehreren Stufen sowie gleichartige Abbremsphasen. Die Pulse werden von PLSO2 Device-Treiber erzeugt. Die Pulsausgabe ist auf Pin L80 eingestellt, die Richtungsauswahl auf Pin L81. Das Beispielprogramm besteht aus dem Programm ‚Main', welches stellvertretend für jede beliebige Anwendung die Schrittmotorsteuerung verwendet.

Unterprogramm ‚init_stepper'

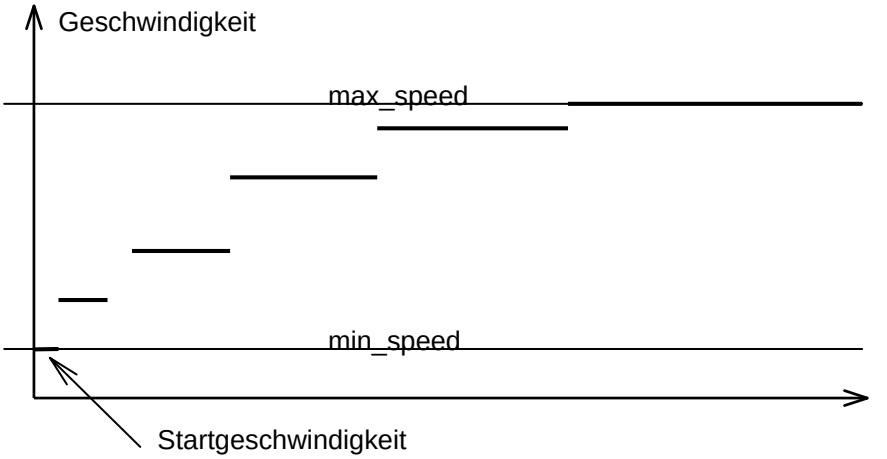
Zunächst wird im Unterprogramm ‚init_stepper' eine Beschleunigungstabelle anhand der definierten Vorgaben berechnet. In einem Array mit werden für die geforderte Anzahl von Beschleunigungsphasen Werte abgelegt für:

- die Anzahl TIMERA-Zyklen für einen Schritt des Schrittmotors
- wie viele Schritte lang jede Beschleunigungsphase dauern soll.

Der Schrittmotor soll einige langsame Schritte machen, um sich in Bewegung zu setzen. Die nächsten Schritte können bereits schneller sein. So steigert sich die Schrittgeschwindigkeit, bis die maximale Geschwindigkeit erreicht ist. Die ersten Phasen sind kürzer als die späteren, wenn der Motor bereits recht schnell ‚schreitet'. Um den Motor anzuhalten wird der gleiche Vorgang umgekehrt durchlaufen.

Unterprogramm ‚stepper'

Diese Unterprogramm sorgt dafür, daß die richtigen Beschleunigungs- und Bremsfolgen ausgeführt werden. Zunächst wird geprüft, ob nur wenige Schritte gemacht werden sollen. Dann brauchen die höheren Beschleunigungsstufen gar nicht erreicht werden. Beschleunigt wird nur soweit, daß die Anzahl der Beschleunigungsschritte jeder Phase mal zwei – wegen der Bremsschritte – plus ca. 20% durchfahren werden. Die höchstgeschwindigkeit wird also erst bei entsprechend hoher auszuführender Schrittzahl erreicht.



Report

Die Reportfunktion besteht aus einigen PRINT-Zeilen in dem Unterprogramm ‚stepper' sowie aus dem Unterprogramm ‚report', welches mehrach aufgerufen wird. Die eingebaute Reportfunktion dient Testzwecken und ist für echte Anwendungen auszukommentieren. Das Report-Unterprogramm zeigt zu jeder Beschleunigungs- bzw. Bremsphase die Daten an:

- welche Phase (Array-Index) sich im Reloadpuffer befindet
- wie viele Schritte in dieser Phase ausgeführt werden
- wie viele TIMERA-Einheiten eine Schrittzeit ist (hoher Wert=langsam)
- wie viele Schritte noch auszuführen sind
- wie viele Bremsschritte auszuführen sind

Hinweis: Wenn die Reportfunktion des Beispiels verwendet wird, sollte ein Terminal an SER0 angeschlossen sein. Wenn kein Terminal angeschlossen ist, müssen die PRINT-Instruktionen auskommentiert werden, da sonst das Programm stehen bleibt, wenn der Ausgabepuffer voll ist. Alle Reportstellen sind mit ‚@report' gekennzeichnet.

Programmbeispiel:

```
'--------------------------------------------------------------------
' Name: PLSO2_STEPPER.TIG
' Demonstriert, wie PLSO2 einen Schrittmotor steuert.
' Das Main-Programm gibt die Werte für die Steuerung vor,
' und das Unterprogramm 'stepper' steuert den Motor.
' Vorgegeben werden:
' maxiamle Geschwindigkeit in Schritten/Sek
' in welcher Zeit und
' in wieviel Beschleunigungsphasen
' die Geschwindigkeit erreicht werden soll
'--------------------------------------------------------------------
user var strict                    ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC                ' User Function Codes
#INCLUDE DEFINE_A.INC              ' allg. Symbol-Definitionen

                                   ' Motor-Richtungspin
#define P_DIR 8                    ' Port fuer Richtungssteuerpin
#define M_DIR 00000010b            ' Bitmaske fuer Richtungssteuerpin
#define N_DIR 1                    ' Pin-Nummer des Richtungssteuerpin
#define RIGHT 0FFh                 ' Pinpegel fuer rechtsherum
#define LEFT  0                    ' Pinpegel fuer linksherum

                                   ' Motordaten
                                   ' folgende Werte als REAL angeben!
#define MAX_SPEED 300              ' in Schritten pro Sekunde
#define MIN_SPEED 20               ' in Schritten pro Sekunde
#define T_ACCEL 1000               ' Beschleunigungszeit in msec
#define ACCEL_STEPS 8              ' Anzahl Beschleunigungphasen
#define MIN_SLOW_STEPS 3           ' minimale Anzahl langsamster steps
#define ISPD 0                     ' Array-Index Geschwindigkeit
#define ISTEP 1                    ' Array-Index no_of_steps
#define ILIM 2                     ' Array-Index min. steps

                                   ' Globale Variable
'LONG no_of_steps                   ' Anzahl auszufuehrender Schritte
BYTE direction                     ' Drehrichtung
REAL ta_unit                       ' TIMERA-Zykluszeit
ARRAY speeds(ACCEL_STEPS,3) of LONG' Serie von Geschwindigkeitsvorgaben

' Unterprogramm Stepper
LONG nsteps, step_cnt              ' interne Step-Einstellung
LONG brk_steps                     ' soviel werden z. Bremsen gebraucht
LONG phases, phase                    ' Phasen
WORD cycle, duty                   ' PLSO2-Parameter
LONG reload, rest                  ' Abfrage des Device-treibers


'--------------------------------------------------------------------
' main
' ruft Initialisier-Unterprogramm auf und verwendet Unterprg. Stepper
'--------------------------------------------------------------------
TASK MAIN
  BYTE ever                        ' Endlosschleife

  INSTALL_DEVICE #TA, "TIMERA.TDD", 1,250 ' 10 kHz
  ta_unit = 0.0001                 ' TIMERA-Zykluszeit
  INSTALL_DEVICE #OPL2, "PLSO2_80.TDD"   ' Pulse auf Pin L80
```

```
  direction = LEFT                ' linksherum
  call stepper ( 5 )              ' Anzahl Schritte
  call stepper ( 8 )
  call stepper ( 20 )
  call stepper ( 55 )
  call stepper ( 128 )
  call stepper ( 200 )
  call stepper ( 400 )
  call stepper ( 2000 )
END


'--------------------------------------------------------------------
' Name: stepper
' verwendet PLSO2 um den Motor zu steuern
' Werte sind in den globalen Variablen vorgegeben:
' TIMERA bestimmt global die Geschwindigkeit
'
' uebergebener Wert:
' LONG no_of_steps                  ' Anzahl auszufuehrender Schritte
'
' global deklarierte Variable:
' BYTE direction                    ' Drehrichtung
' WORD cycle, duty                  ' Geschwindigkeit
' LONG reload, rest
' REAL ta_unit                      ' TIMERA-Zykluszeit
' ARRAY speeds(ACCEL_STEPS,3) of LONG' Serie von Geschwindigkeitsvorg.
' LONG nsteps, step_cnt             ' interne Step-Einstellung
' LONG brk_steps                    ' soviel werden z.Bremsen gebraucht
' LONG phases, phase                ' Phasen

'--------------------------------------------------------------------
SUB stepper ( no_of_steps )

  out P_DIR, M_DIR, direction     ' setze Drehrichtung
                                  ' wieviel Beschl.phasen?
  for phases = ACCEL_STEPS - 1 to 1 step -1
    if no_of_steps > speeds ( phases, ILIM ) then
      goto do_phases
    endif
  next

do_phases:                        ' --------------------------------
                                  ' wenn nur wenige Schritte
  if no_of_steps < speeds ( 0, ILIM ) then
    phase = 0
    cycle = speeds ( phase, ISPD )    ' niedrigste Geschwindigkeit
    duty = cycle / 2
    put #OPL2, #0, no_of_steps, duty, cycle ' starte Pulsausgabe
    goto stepper_stop             ' warte bis fertig
  endif
                                  ' sonst
                                  ' --------------------------------
  step_cnt = no_of_steps          ' beschleunigen
  brk_steps = 0

  cycle = speeds ( 0, ISPD )      ' Startgeschwindigkeit
  duty = cycle / 2
  nsteps = speeds ( 0, ISTEP )
```

```
  brk_steps = brk_steps + nsteps  ' Bremsschritte mitzaehlen
  step_cnt = step_cnt - nsteps    ' getane Schritte abziehen

  for phase = 1 to phases
    cycle = speeds ( phase, ISPD )
    duty = cycle / 2
    nsteps = speeds ( phase, ISTEP )
    put #OPL2, #1, nsteps, duty, cycle ' reload Pulsausgabe
    brk_steps = brk_steps + nsteps' Bremsschritte mitzaehlen
    step_cnt = step_cnt - nsteps  ' getane Schritte abziehen

    reload = 1
    while reload > 0              ' warte bis Reload-Puffer frei
      get #OPL2, #1, 0, reload
    endwhile
  next
  phase = phase - 1               ' in gueltigen Bereich stellen
                                  ' --------------------------------
  nsteps = step_cnt - brk_steps   ' laufen
  cycle = speeds ( phase, ISPD )
  duty = cycle / 2
  put #OPL2, #1, nsteps, duty, cycle ' reload Pulsausgabe
  step_cnt = step_cnt - nsteps    ' getane Schritte abziehen

  reload = 1
  while reload > 0                ' warte bis Reload-Puffer frei
    get #OPL2, #1, 0, reload
  endwhile
                                  ' --------------------------------
  for phase = phases to 0 step -1 ' bremsen
    cycle = speeds ( phase, ISPD )
    duty = cycle / 2
    nsteps = speeds ( phase, ISTEP )
    put #OPL2, #1, nsteps, duty, cycle ' reload Pulsausgabe
    brk_steps = brk_steps - nsteps' Bremsschritte mitzaehlen
    step_cnt = step_cnt - nsteps  ' getane Schritte abziehen

    reload = 1
    while reload > 0              ' warte bis Reload-Puffer frei
      get #OPL2, #1, 0, reload
    endwhile
  next

stepper_stop:                     ' --------------------------------
  rest = 1                        ' warte bis alles steht
  while rest > 0
    get #OPL2, #2, 0, rest        ' Pulse noch auszugeben
  endwhile
END


'-----------------------------------------------------------------------
' init_stepper
' berechnet Beschleunigingsdaten in ein Array.
' zum Bremsen wird das gleiche Array verwendet.
' im Array steht:
' ( phase, Anz. Schritte, Cycle-Zeit = Geschwindigkeit )
' hier wird eine lineare Anfahrrampe vorgegeben
'-----------------------------------------------------------------------
SUB init_stepper
```

```
  REAL t_acc, acc_steps, min_cyc, max_cyc
  REAL nsteps, step_cyc, t_phase, ta_cyc, nlim

  t_acc = T_ACCEL                 ' wandele Werte nach REAL
  acc_steps = ACCEL_STEPS
  min_cyc = 1.0 / MAX_SPEED
  max_cyc = 1.0 / MIN_SPEED

  dir_pin P_DIR, N_DIR, 0         ' mache Richtungspin zum Ausgang
                                  ' Anfahr- und Bremsrampe

  for ap = 0 to ACCEL_STEPS - 1
                                  ' Dauer dieser Beschl.phase
    t_phase = t_acc/(acc_steps-ap)-t_acc/(acc_steps-ap+1)
                                  ' Zykluszeit eines Steps in d. Phase
    step_cyc = &
         min_cyc + ((max_cyc-min_cyc)/(ap+1)-(max_cyc-min_cyc)/(ap+2))
    ta_cyc = step_cyc / ta_unit   ' in TIMERA-Einheiten
                                  ' Anzahl Zyklen fuer diese Phase
    nsteps = t_phase / step_cyc / 1000
    if nsteps < MIN_SLOW_STEPS then
      nsteps = MIN_SLOW_STEPS
    endif
                                  ' min Schritte f. diese Geschw.
    nlim = (2 * nsteps) + (nsteps * 0.2)
    speeds ( ap, ISPD ) = RTL(ta_cyc)  ' Cycle-Zeit
    speeds ( ap, ISTEP ) = RTL(nsteps) ' Anzahl Schritte
    speeds ( ap, ILIM ) = RTL(nlim) ' min Schritte f. diese Geschw.
    if ap > 0 then
      speeds(ap,ILIM) = speeds(ap,ILIM) + speeds(ap-1,ILIM)
    endif
  next
END
```

Musik mit PLSO1

Dateiname: PLSO1_JUKEBOX.TIG

Includes: DEFINE_A, UFUNC, MUSIC_POPCORN

Device-Treiber: PLSO1.TDD, LCD1.TDD

Der Pulstreiber PLSO1.TDD wird in dieser Applikation verwendet, um ein Musikstück abzuspielen. Die Töne sind rein digital erzeugt und haben im Ausgang eine rechteckige Kurvenform. Eine Filterung und damit Abrundung der Kurvenform findet nur im Verstärker und durch den Lautsprecher statt. Der maximal mögliche Frequenzbereich reicht bei PLSO1.TDD von ca. 38Hz bis 1,25Mhz. Wenn die Ton-Zeitdauer begrenzt ist und durch die Anzahl der Pulse bestimmt wird, dann reduziert sich die Obergrenze etwas, da PLSO1 die Zyklen zählen muß. Die Zykluslängen werden mit einer Auflösung von 0,4µsec berechnet. Die gewünschten Töne liegen im Frequenzbereich:von ca. 100Hz bis 5kHz.

Der Wert für DUTY bestimmt das Tastverhältnis und damit die Klangfarbe (Oberwellen-Gehalt). Hier wird ein Tastverhältnis von 50% gewählt.

Die Töne (vereinfacht): Kammerton "A" = 440 Hz Jeder Halbton-Schritt wird hier durch Multiplikation bzw. Division mit der 12. Wurzel aus 2 (=1.05946309...) erzeugt:

A' = 440 Hz
B' = 440 Hz * 1.05946.. = 466,16 Hz
H' = 466,16 * 1.05946.. = 493,88 Hz
usw.

Die Oktaven bestehen mit den Halbtönen aus 12 Tönen. Die Basisfrequenzen für die Oktaven sind:

1. Oktave: ab 130,8 Hz
2. Oktave: ab 261,6 Hz
3. Oktave: ab 523,2 Hz
4. Oktave: ab 1046,5 Hz
5. Oktave: ab 2093,0 Hz
6. Oktave: ab 4186,0 Hz

Die Töne werden in einem Unterprogramm berechnet und als WORD-Werte in einem String abgelegt. Auf einen Ton wird per **Index in den String TONE$** zugegriffen. Der Index hinter dem letzten Ton hat die Bedeutung einer Pause (Index = 63(3Fh). Folgende Tabelle zeigt die Indice zu den Tönen:

	1.Okt	2.Okt	3.Okt	4.Okt	5.Okt	6.Okt
C	0 (00h)	12 (0Ch)	24 (18h)	36 (24h)	48 (30h)	60 (3Ch)
cis	1 (01h)	13 (0Dh)	25 (19h)	37 (25h)	49 (31h)	61 (3Dh)
D	2 (02h)	14 (0Eh)	26 (1Ah)	38 (26h)	50 (32h)	62 (3Eh)
dis	3 (03h)	15 (0Fh)	27 (1Bh)	39 (27h)	51 (33h)	- pause
E	4 (04h)	16 (10h)	28 (1Ch)	40 (28h)	52 (34h)	
F	5 (05h)	17 (11h)	29 (1Dh)	41 (29h)	53 (35h)	
fis	6 (06h)	18 (12h)	30 (1Eh)	42 (2Ah)	54 (36h)	
G	7 (07h)	19 (13h)	31 (1Fh)	43 (2Bh)	55 (37h)	
gis	8 (08h)	20 (14h)	32 (20h)	44 (2Ch)	56 (38h)	
A	9 (09h)	21 (15h)	33 (21h)	45 (2Dh)	57 (39h)	
B	10 (0Ah)	22 (16h)	34 (22h)	46 (2Eh)	58 (3Ah)	
H	11 (0Bh)	23 (17h)	35 (23h)	47 (2Fh)	59 (3Bh)	

In die beiden höchsten Bits jedes Index wird die Tonlänge codiert:

Bit 7 und 6	Pausenlänge	addiert zum Index
00	1-fach	+00h
01	2-fach	+40H
10	3-fach	+80H
11	4-fach	+C0H

Applikationen

Das Beispielprogramm PLSO1_JUKEBOX.TIG lädt das abzuspielende Musikstück in einer Include-Zeile in der Task Main. In der auskommentierten Zeile wird ein ‚Musikstück' eingebunden, welches das Abspielen der ersten Oktave mit verschiedenen Tonlängen demonstriert. Kommentieren Sie dazu lediglich die entsprechende Include-Zeile ein, und die andere aus.

Programmbeispiel:

```
'---------------------------------------------------------------------
' Name: PLSO1_JUKEBOX.TIG
' Spielt Popcorn mit PLSO1.TDD am Pin L86
' Verbinde L86 mit PA-in (Verstärker-Eingang) und schliesse
' Lautsprecher an. Durch Veraendern der frequenz- und
' zeitbestimmenden Variablen, koennen entsprechende andere Tonlagen
' und Klangfarben sowie andere Tonfolgen/Melodien erzeugt werden.
'---------------------------------------------------------------------
user var strict                    ' unbedingte Var.deklaration
#INCLUDE UFUNC3.INC                ' User Function Codes
#INCLUDE DEFINE_A.INC              ' allg. Symbol-Definitionen

STRING tone$ (144)                 ' Cycle-Werte fuer alle Toene

TASK MAIN
  LONG  count
  WORD  duty, cycle
  STRING MUSIC1$ (1000)            ' enthaelt ein Stueck Musik

  INSTALL_DEVICE #1, "LCD1.TDD"
  INSTALL_DEVICE #OPL1, "PLSOUT1.TDD", 1 ' Zeitbasis-Bereich = 1

  PRINT #1,"<1>==PLSO1-JUKEB.TIG===";
  PRINT #1,   " PulseOut <<Musik>"
  PRINT #1,   "  L86 <<--> PA-in" ' Eingangs-Meldung auf LCD
  PRINT #1,   "+ connect speaker !";

  CALL INIT_TONES                  ' initialisiere Toene f. 6 Oktaven

#INCLUDE MUSIC_POPCORN.INC         ' hier wird MUSIC1$ mit den
                                   ' Musikdaten initialisiert

  CALL PLAY_MUSIC (MUSIC1$, 360, 25, 1)

  PRINT #1,"<1>====PO-JUKEB.TIG====";
  PRINT #1,   "                     ";
  PRINT #1,   "Ende !"             ' Abschieds-Meldung auf LCD
END


'---------------------------------------------------------------------
' Spiele ein Stueck Musik ab:
'
' M$ = String mit Noten + Pausen des Musikstuecks
'  SPEED = Taktrate, z.B. 136 --> 136 Schlaege pro Min (1/4 Note)
' TP_REL = Ton/Pause Verhaeltnis in %. pro 1/8 Note.
' Also:  80 --> 80% Ton, dann 20% Pause (bei 1/8 Note)
' REPEAT = Anzahl der Wiederholungen dieses Musik-Stueckes:
' 0 = endlos, 1...nnnnnnn = Anzahl
'---------------------------------------------------------------------
SUB PLAY_MUSIC (STRING M$; LONG SPEED, TP_REL, REPEAT)
LONG MUS_LEN, THIS_TONE, DACAPO, ST
LONG TONE_CNTS, PAUSE_MS

  MUS_LEN=LEN(M$)                  ' Laenge des Musikstuecks
  IF REPEAT=0 THEN
    ST=0                           ' setze Schrittweite des FOR-Loops
  ELSE
```

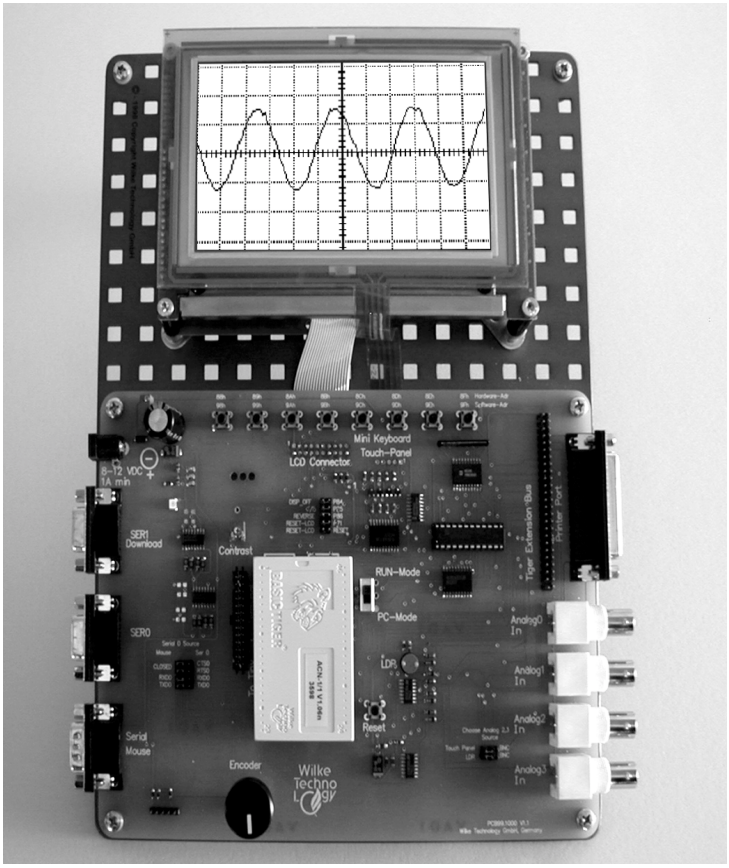
```
  ENDIF
  TONE_CNTS=2500000*60/(SPEED*2)  ' 0,4 usec Counts pro 1/8 Note
  PAUSE_MS=(1000*60*(100-TP_REL))/(SPEED*100)' in ms nach jedem Ton

  FOR DACAPO=0 TO REPEAT-1 STEP ST' <---------- Loop --------->
    FOR THIS_TONE=1 TO MUS_LEN
      CALL PLAY_TONE (M$, TONE_CNTS, PAUSE_MS, THIS_TONE)
    NEXT                          ' TONE
  NEXT                            ' DACAPO
END


'---------------------------------------------------------------------
' Spiele einen Ton ab in der gewuenschten Art:
'
' MUS$  = Musik     - String mit Noten + Pausen des Musikstuecks
' TCNTS = Ton-Dauer - fuer 1/8 Note bei der gewuenschten Taktrate
'         in 0,4 usec Einheiten
' PAUSE = nach Ton  - das hoerbare Tone-Ende, Angabe in:  ms
' TNDX  = Index+1   - in den Ton-String MUS$ --> dieser Ton
'
'---------------------------------------------------------------------
SUB PLAY_TONE (STRING MUS$; LONG TCNTS, PAUSE, TNDX)
  LONG A, COUNT, N
  WORD CYCLE, DUTY

  A=NFROMS (MUS$, TNDX-1, 1)      ' hole 1 Byte aus String
  COUNT=(A BITAND 0C0H) SHR 6     ' Ton-Dauer-Ndx:  0, 1, 2, 3

  A=A BITAND 3FH                  ' Ton (Note): 00...3E, 3F
  CYCLE=NFROMS (TONE$, A*2, 2)    ' Ton (Note): in CYCLE-Laenge
  COUNT=((COUNT+1)*TCNTS)/CYCLE   ' Ton-Dauer in COUNTS

  IF A=3FH THEN                   ' Unterscheide: Pause ?
    CYCLE=2500000/COUNT*CYCLE/1000' Hier: nur eine Pause ausgeben
    WAIT DURATION CYCLE           ' Dauer der Pause in (ms)
  ELSE
    DUTY=CYCLE/2                  ' Hier: Ton ausgeben
    PUT #OPL1, COUNT, DUTY, CYCLE ' Sende TON/PAUSE
    N = 2
    WHILE N > 1                   ' --> Loop wartet auf Ton-Ende
      RELEASE TASK                ' andere Tasks dran lassen
      GET #OPL1, #0, #0B0H, 4, N  ' hole akt. Count-Down ab und
      N = SIGNEXT ( N, 16 )       ' GET bekommt nur WORD
    ENDWHILE
    WAIT DURATION PAUSE           ' noch eine kleine Pause
  ENDIF
END


'---------------------------------------------------------------------
' Initialisiert die Toene fuer 6 Oktaven
'---------------------------------------------------------------------
SUB INIT_TONES
  LONG N, K
  REAL  ROOT12                    ' 12. Wurzel aus 2
  REAL  FREQU                     ' Frequenz

  ROOT12 = EXPE (LN(2.0)/12.0)    ' berechne 12. Wurzel 2
  TONE$ = FILL$ (CHR$(0),12*6*2)  ' init TONE$ String
```

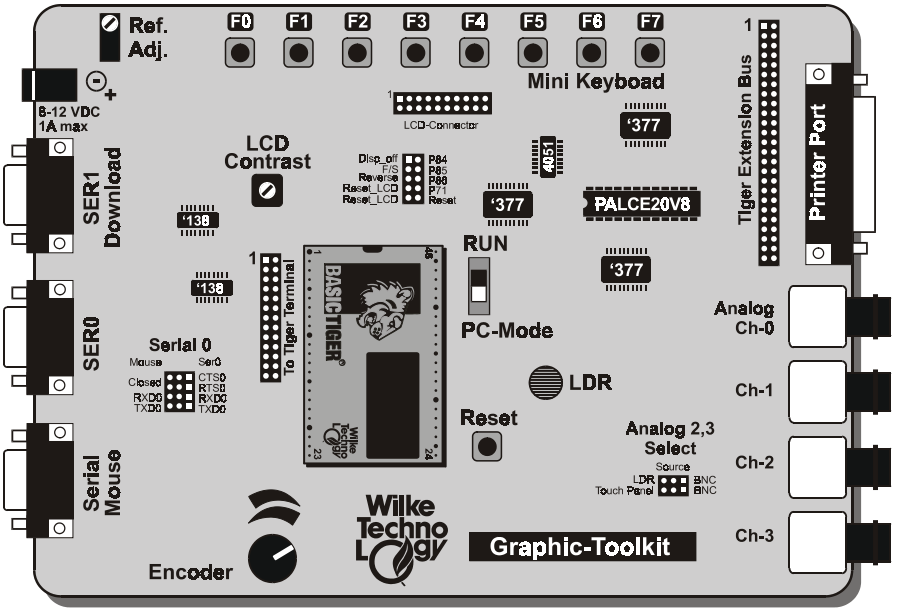
```
  FOR N=0 TO 62*2 STEP 2          ' 6 Oktaven berechnen
    K=((1250000/FREQU)+0.5)
    TONE$ = NTOS$ (TONE$, N, 2, K)
    FREQU = FREQU * ROOT12        ' naechste Frequenz
  NEXT
END
```

Leere Seite

Leere Seite

4 BASIC-TIGER® Graphic-Toolkit





Gesamtansicht des BASIC-Tiger® Graphic-Toolkits

Das BASIC-Tiger® Graphic-Toolkit ist eine Hardwareplattform zur schnellen Entwicklung von Anwendungen mit Grafik-LCD mit dem BASIC-Tiger®. Es wird Vorausgesetzt, daß Sie das Entwicklungssystem BASIC-Tiger® Version 4.0 oder neuer besitzen und mit Tiger-BASIC® vertraut sind.

Das BASIC-Tiger® Graphic-Toolkit stellt folgende Peripherie-Komponenten zur Verfügung.

- Grafik LC-Display mit 240 x 128 Pixel und CFL Hintergrundbeleuchtung
- analoges Touchpanel
- 2 serielle RS232 Schnittstellen, davon eine optional als PC-Mausanschluß
- Minitastatur mit 8 Tasten
- Druckerport
- 4 analoge Eingänge mit BNC-Buchsen und Eingangsspannungsbegrenzung
- Drehimpulsgeber für z.B. Menü-Scrolling
- Netzteileingang für 8-12 V DC
- Fotowiderstand

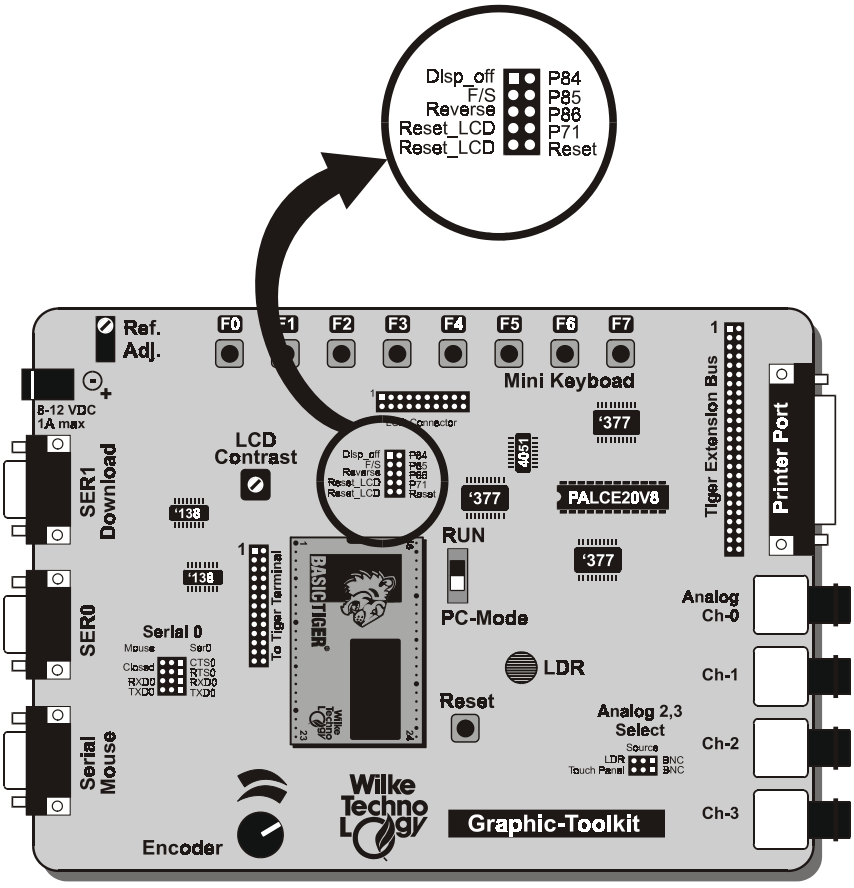
Das BASIC-Tiger® Graphic-Toolkit kann sowohl mit BASIC-Tiger® A-Modulen als auch (über Adapter) mit BASIC-Tiger® T-Modulen bestückt werden. Das BASIC-Tiger®-Modul darf jedoch nicht mit RS-232-Treiber ausgestattet sein.

Die lange Jumperreihe links neben dem Modul ist in der Regel mit Jumpern bestückt, um eine Verbindung zur Mini-Tastatur des BASIC-Tiger® Graphic-Toolkits herzustellen. Wenn alle Jumper gezogen werden, dann ist an diesem Steckverbinder das BASIC-Tiger®-Terminal anschließbar.

Folgende Komponenten sind auf dem Bord an das Modul angeschlossen. Die mit ‚Jumper' gekennzeichneten Anschlüsse teilen sich den Pin mit anderen Geräten und sind entsprechend umschaltbar bzw. trennbar:

Komponente	Pin (logische Bezeichnung)	Bemerkung
Grafik-LCD	L60...L67 L80...L83 L84 L85 L86 L71	Datenbus Steuerleitungen (fest verbunden) DISP_OFF (über Jumper) Font-Select (über Jumper) Invers (über Jumper) LCD-Reset (über Jumper)
Erweiterte Ein- und Ausgänge	L60...L67 L33,34,35	Datenbus und Steuerleitungen. Erw. I/O werden für Tastatur und Touchpanel gebraucht. Jumper (sonst BASIC-Tiger®-Terminalanschluß)
Touchpanel	erw. I/O, AN2	schalten VCC, GND, analogen Meßpunkt analoger Meßeingang (Jumper)
Tastatur	=erw. I/O	8 Tasten
Drehimpulsgeber	L72,73, L87	2x Impuls und Achse des Drehimpulsgebers
Drucker	L60...L67 L70,71	Datenbus Steuerleitungen '-strobe', 'busy'
DB9-SER0	L90,91,92,95	über RS-232-Treiber (Jumper: DB9-Maus)
DB9-SER1	L93,94	über RS-232-Treiber
BNC AN0 BNC AN1 BNC AN2 BNC AN3	AN0 AN1 AN2 AN3	Analogeingang AN0 Analogeingang AN1 Analogeingang AN2 (Jumper Touchpanel) Analogeingang AN3 (Jumper LDR)
Summer	L42	Tastenklick, Beep
LDR	AN3	Lichtsensor (Jumper)
PC/RUN-Mode	PC	Schiebeschalter
Vref einstellbar	Vref	
Resettaster	RESET	
nicht verwendete Pins	L36, L37, L40	(verwendet von BASIC-Tiger®-Terminal) frei

Grafik-LC-Display



Steuerpins des Grafik-LCD

Zur Ansteuerung des Grafik-LCDs steht ein Device-Treiber zur Verfügung: LCD-6963. Genauere Informationen zum Device-Treiber des Grafik-Displays sind im Abschnitt „LCD-Grafik-Display“ des BASIC-Tiger® des Device Treiberhandbuchs zu finden.

Neben verschiedenen LCD-Größen gibt es bei den Grafik-LCDs mit Prozessor T6963 zwei Einstellungen, die einer verschiedenen Ansteuerung durch den Device-Treiber bedürfen. Der Logikpegel am Steuerpin ‚Font-Select‘ bewirkt, daß

- bei low-Pegel die Zeichen in einer Matrix **8x8-Pixel** dargestellt sind und alle 8 Bits von jedem Grafikbyte als Pixel erscheinen. **LCD-Typ: 4**
- bei high-Pegel die Zeichen in einer Matrix **6x8-Pixel** dargestellt sind und nur die niederen 6 Bit von jedem Grafikbyte als Pixel erscheinen. **LCD-Typ: 6**

Die Steuerpins des LCD müssen passend gesetzt werden, und es muß bei der Installation des Device-Treibers ‘LCD-6963.TDD‘ der passende LCD-Typ entsprechend der verwendeten Einstellung des Font-Select-Pins angegeben werden (Typ 4 oder Typ 6). Dies wird in den Beispielen durch das Einbinden der Include-Dateien in der richtigen Reihenfolge sowie durch den Aufruf von ‚init_LCDpins‘ gewährleistet. ‚init_LCDpins‘ ist in GR_TK1.INC enthalten.

Für den LCD-Typ des BASIC-Tiger® Graphic-Toolkit gibt es je eine Include-Datei für die beiden Modi ‚6x8-Pixel-Zeichensatz‘ und ‚8x8-Pixel-Zeichensatz‘. Die beiden Include-Dateien **LCD_4.INC** und **LCD_6.INC** enthalten die passenden Definitionen für den LCD-Typ 4 bzw. LCD-Typ 6. In den Beispielen zum BASIC-Tiger® Graphic-Toolkit wird ein I/O-Pin des BASIC-Tiger®-Moduls als Steuerpin für den Font-Select-Pin am LCD verwendet (siehe LC-Display-Jumperblock).

Die Include-Datei zum BASIC-Tiger® Graphic-Toolkit **GR_TK1.INC** enthält Definitionen zum Toolkit und das Unterprogramm ‚init_LCDpins‘. Sie wird auf jeden Fall nach der Include-Datei des LCD-Typs eingebunden.

Für eigene Projekte mit Hardware, die vom BASIC-Tiger® Graphic-Toolkit abweicht, sowie andere LCD-Typen können die vorhandenen Include-Dateien als Vorlage dienen.

Der Font-Select-Pin beeinflußt nicht nur den Zeichensatz, sondern ebenfalls die Pixelverteilung des Grafikbereiches. Im Modus ‚6x8-Zeichensatz‘ werden von jedem Grafikbyte nur die niedersten 6 Bit für die Grafik verwendet. Damit eine Grafik richtig im LCD erscheint, wird sie vor der Übertragung in Streifen zerschnitten (Funktion GRAPHIC_EXP$). Aus den 30 Bytespalten entstehen dabei 40 Bytespalten und es werden entsprechend mehr Bytes für eine Grafik übertragen.

Das LCD vom Typ 7 mit der Auflösung 128x64 hat im Modus ‚6x8-Zeichensatz‘ 21 Textspalten. Im Grafikmodus werden jedoch 22 Bytespalten gezählt, da 128 nicht durch 6 teilbar ist und so 2 Pixelspalten nicht ansteuerbar wären. Diese Tatsache kommt zum Tragen, wenn die Funktion GRAPHIC_EXP$ eingesetzt wird, um die Grafik an den 6x8-Modus anzupassen.

Hinweis: Grafik LCDs können sich in der Pixelform unterscheiden. So werden zum Beispiel Kreise auf manchem LCD zu Ellipsen, wenn die Pixel nicht quadratisch sind.
Die Anschlußbelegungen sind von Modell zu Modell unterschiedlich.

Besorgen Sie sich Unterlagen zu Ihrem LCD bevor Sie ein Modell fest einplanen.

Folgende Schritte sorgen in den Beispielprogrammen dafür, daß das LCD richtig eingebunden ist:

	Quelltext	**Bedeutung**
1.	#include LCD_4.INC oder #include LCD_6.INC	enthält Definitionen passend für 8x8-Zeichensatz mit 8 Bit pro Grafikpixel enthält Definitionen passend für 6x8-Zeichensatz mit 6 Bit pro Grafikpixel
2.	#include GR_TK1.INC	enthält Definitionen passend zum BASIC-Tiger® Graphic-Toolkit und das Unterprogramm ‚init_LCDpins'.
3.	call init_LCDpins	initialisiert LCD-Steuerpins entsprechend LCD_4.INC bzw. LCD_6.INC
4.	install_device „LCD-6963.TDD", Parameter...	installiert den Device-Treiber. Der Parameter LCD_TYPE ist in LCD_4.INC bzw. LCD_6.INC definiert.

LCD-Anschluß:

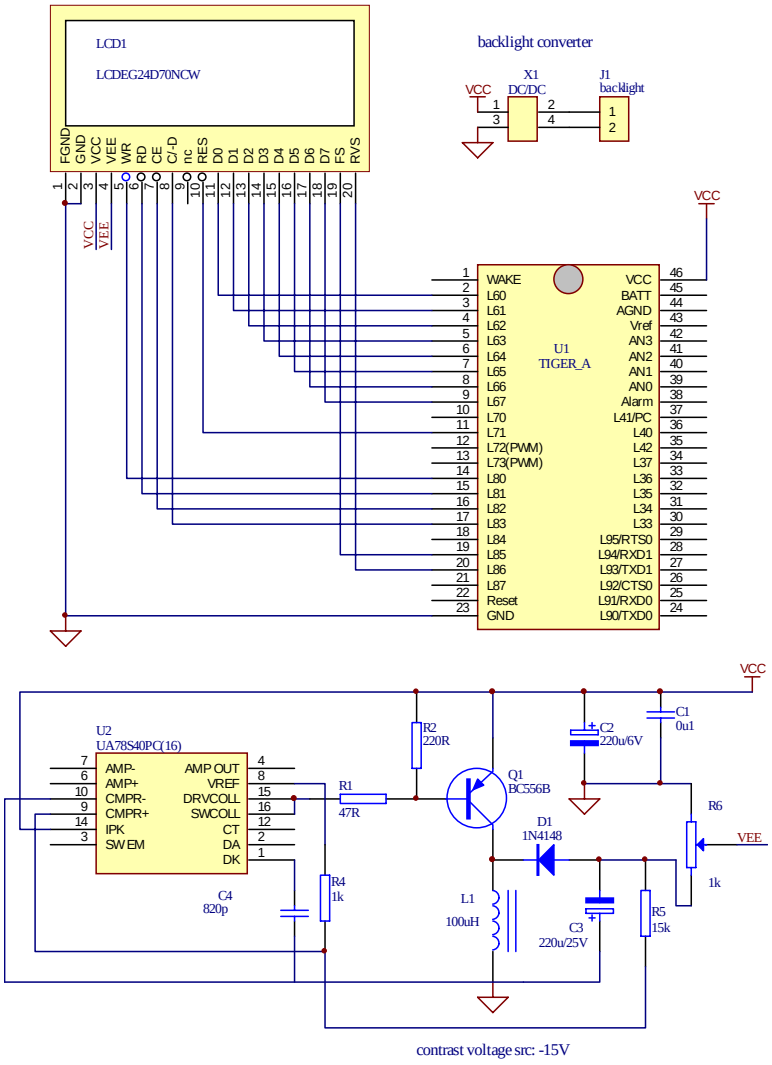
GND	1	2	VCC
Kontrast	3	4	C/-D
-WR	5	6	-RD
D0	7	8	D1
D2	9	10	D3
D4	11	12	D5
D6	13	14	D7
-CE	15	16	Reset
-15	17	18	ON
FS low: 8x8 high: 6x8	19	20	-Revers low: weiße high:schwarze Zeichen

LCD-Jumper

LCD	Pin	Pin	BASIC-Tiger®
backlight ON	1	2	L84
Font-Select	3	4	L85
Revers	5	6	L86
LCD-Reset	7	8	L71
LCD-Reset	9	10	Reset

Anmerkung: Wenn LCD-Reset auf das Resetnetz des BASIC-Tiger® Graphic-Toolkit gejumpert wird, dann ist zu beachten, daß das Resetnetz nur bei Tastendruck ein Reset erhält, nicht jedoch beim Einschalten des BASIC-Tiger® Graphic-Toolkit. Die Reset-Pins des BASIC-Tiger® A und BASIC-Tiger® T sind nur Eingänge, es wird kein Resetsignal nach außen gegeben. In den Beispielprogrammen gibt das Tigermodul über den Pin L71 dem LCD einen Reset.

Der Schaltplan zeigt alles, was zum Anschluß eines Grafik-LCD mit Hintergrundbeleuchtung an den BASIC-Tiger® nötig ist. Auf dem BASIC-Tiger® Graphic-Toolkit ist der Spannungskonverter und der DC/DC-Wandler für die Kontrastspannung auf der Platine unter dem LCD angeordnet.



Große Ziffern

Viele Anwendungen verwenden zur Darstellung große Zeichen auf dem Grafik-LCD. Die LCD-Module haben jedoch keinen internen Zeichensatz mit großen Zeichen, oder gar einen Zeichensatz, der in der Größe einstellbar ist. Handelt es sich um einen feststehenden Text, dann ist dieser einfach ein Bestandteil der Bildschirmmaske. Soll der Text jedoch veränderlich sein, zum Beispiel eine Ziffernanzeige, dann muß das Anwendungsprogramm die Ziffern zur Laufzeit zusammenstellen.

Das Programmbeispiel BIGDIGITS.TIG demonstriert, wie in einfacher Form große Ziffern dargestellt werden. Der Zeichensatz liegt als Grafik in einer Bitmapdatei vor. Die Größe der Zeichen ist bekannt und für alle Zeichen gleich. Das Unterprogramm ‚BigDigits' stellt eine Zahl zusammen, indem die 4 Ziffern mit der Instruktion GRAFIK_COPY in den Ausgabestring kopiert werden. Auf Wunsch kann die Zahl mit Vorzeichen ausgegeben werden. Das Unterprogramm verwendet einen Index in eine Liste von Fonts. Der Drehgeber wird verwendet, um die Zahl einzustellen. Ein Druck auf den Drehgeber wechselt den Zeichensatz, indem der Index in die Fontliste weitergezählt wird.

Folgende Erweiterungen zu dem Beispiel ‚BigDigits' sind denkbar:

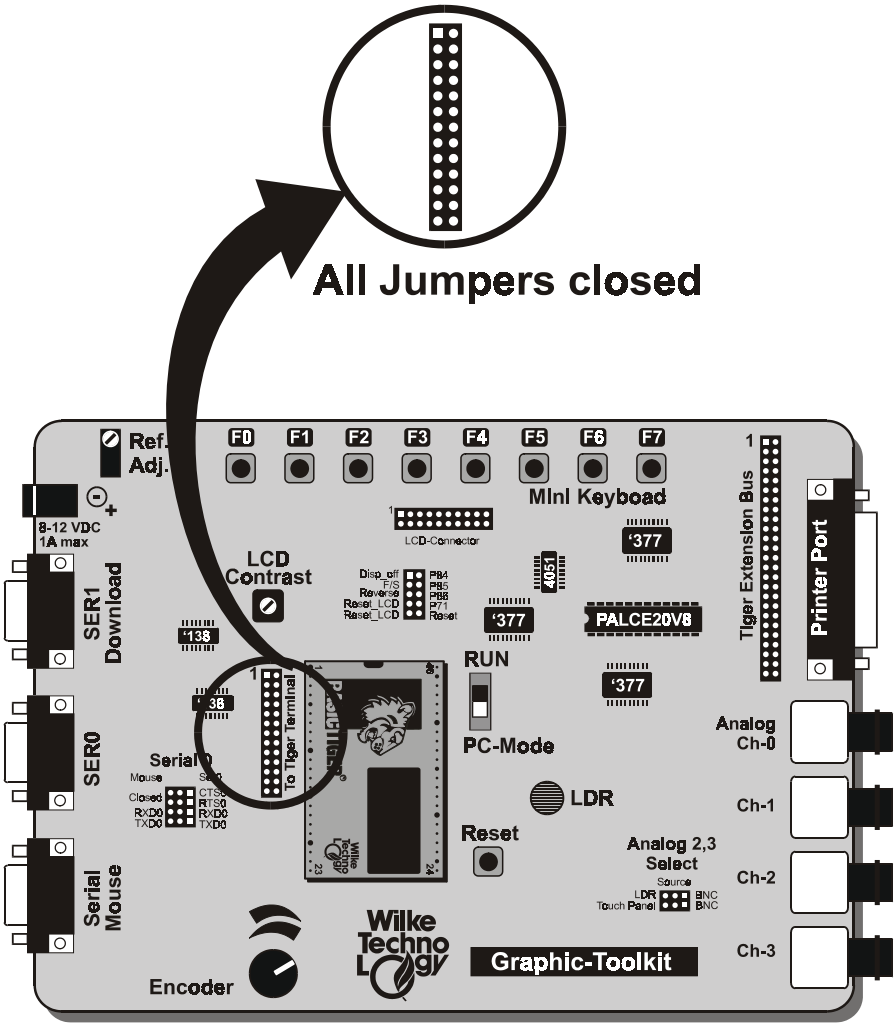
- mehr Zeichen (Buchstaben, Sonderzeichen)
- Unterprogramm verwendet Variable statt fester Zahlen für die Fontgrößen und die Kopierposition
- Bei Proportionalschrift hat jedes Zeichen eine individuelle Dikte (Breite)
- Skalierbare Fonts in Vektorgraphik

Programmbeispiel:

```
'----------------------------------------------------------------------
' Name: BIGDIGIT.TIG
' Zeigt auf dem grafischen Display den Zaehlstand des Encoders
' unter Verwendung eines grafischen Fonts in grossen Ziffern.
' Ein Druck auf die Encoderachse wechselt den Zeichensatz.
'----------------------------------------------------------------------
' L71 mit LCD-Reset verbinden
'----------------------------------------------------------------------
user_var_strict                   ' erzwinge Var-Deklaration
#include DEFINE_A.INC             ' allgemeine Definitionen
#include UFUNC3.INC               ' User-Function-Codes
#include LCD_4.INC                ' Definitionen fuer LCD Typ 4
#include GR_TK1.INC               ' Definitionen fuer Graphic Toolkit
#include BIGDIGIT.INC             ' grosse Ziffern fuer Grafik-LCD

STRING Screen$(GR_SIZE), tmp$(GR_SIZE)

'----------------------------------------------------------------------
TASK MAIN
  BYTE bEncKey
  LONG dwEnc
  LONG fontidx

  call Init_LCDpins               ' initialisiere LCD-Pins
                                          ' LCD-4=240x128, 150 KB/s
  install_device #LCD2, "LCD-6963.TDD",0,0,0EEH,4,150,11H
  install_device #TA, "TIMERA.TDD",3,156 ' Zeitbasis 1 kHz
  install_device #ENC, "ENC1_723.TDD" ' Encoder Port 7, Pins 2+3

  Screen$=FILL$ ("<0>",GR_SIZE)   ' String leer initialisieren

  dir_pin P_ENC_KEY, PIN_ENC_KEY, 1' Tastpin als Eingang
  put #LCD2, CURSOR_OFF           ' Text-Cursor aus
  put #ENC, 0                     ' starte Encoder

  fontidx = 0
  while 0 = 0                     ' Endlosschleife
    get #ENC, #0, 4, dwEnc        ' Drehschritte lesen
    print #LCD2, "<1Bh>A<3><2><0F0h>Encoder:"; dwEnc; "  ";
    tmp$ = fill$ ( "<0>", GR_SIZE )' Bereich fuer gr. Ziffern
                      ' DEST,  no., Font-Index, Digits,   X, Y  sign
    call printn_to_stri ( tmp$, dwEnc,   fontidx,      4,  20, 30, 1 )
    put #LCD2, #1, tmp$, 0, 0, GR_SIZE ' zeige Grafik an

    ll_iport_in P_ENC_KEY, bEncKey, M_ENC_KEY
    if bEncKey <> M_ENC_KEY then  ' wenn Encoderachse gedrueckt
      fontidx = fontidx + 1       ' Font wechseln
      if fontidx > 4 then
        fontidx = 0
      endif
      bEncKey = M_ENC_KEY + 1     ' <> M_ENC_KEY ist gedrueckt
      while bEncKey <> M_ENC_KEY  ' warte bis Encoderdruck losgelassen
        ll_iport_in P_ENC_KEY, bEncKey, M_ENC_KEY
      endwhile
    endif
    wait_duration 200             ' Schleifengeschwindigkeit
  endwhile
END
```


-1185

Mini Tastatur

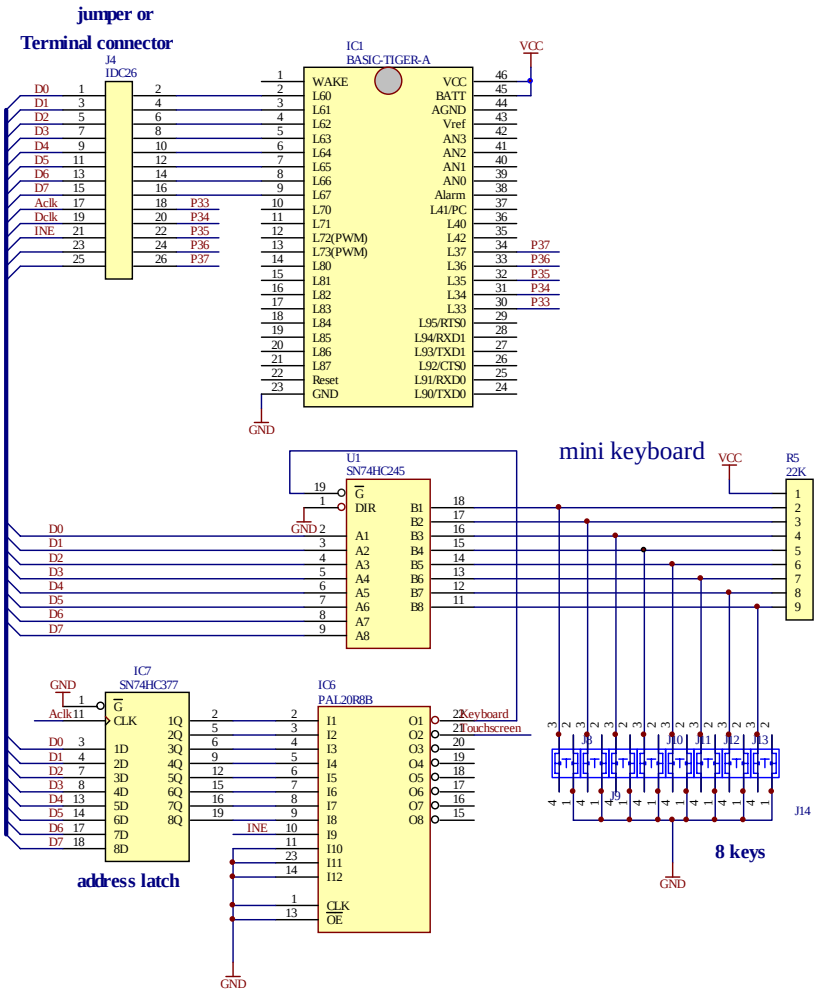


Jumper für erweiterte I/O, Minitastatur

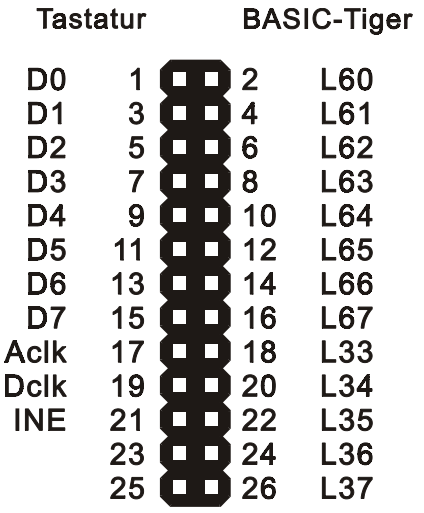
Auf dem BASIC-Tiger® Graphic-Toolkit befindet sich eine einspaltige Mini-Tastatur mit acht Tasten. Die Tasten können zum Beispiel Funktionen zugeordnet werden, die auf dem Grafik-LCD im unteren Bereich dargestellt werden.

Die Spaltenadresse dieser Tastatur ist durch das PAL auf die physikalische Adresse 88h gelegt. Bei einem voreingestellten USER_EPORT OFFSET von -10H ist die Tastatur softwareseitig ab der logischen Adresse 98h ansprechbar. Vom Standpunkt des BASIC-Tiger® besteht die Tastatur aus erweiterten Eingängen. Damit belegt die Tastatur die Pins L60...L67 (Datenbus) sowie die Leitungen L33 (Aclk) und L35 (INE).

Das Beispielprogramm **MINIKEY.TIG** liest die Scancodes der Tastatur. Dazu müssen folgende Jumper gesetzt sein:


jumper or
Terminal connector
J4
IDC26
IC1
BASIC-TIGER-A
VCC
WAKE
L60
L61
L62
L63
L64
L65
L66
L67
L70
L71
L72(PWM)
L73(PWM)
L80
L81
L82
L83
L84
L85
L86
L87
Reset
GND
VCC
BATT
AGND
Vref
AN3
AN2
AN1
AN0
Alarm
L41/PC
L40
L42
L37
L36
L35
L34
L33
L95/RTS0
L94/RXD1
L93/TXD1
L92/CTS0
L91/RXD0
L90/TXD0
D0
D1
D2
D3
D4
D5
D6
D7
Aclk
Dclk
INE
P33
P34
P35
P36
P37
GND
U1
SN74HC245
DIR
A1
A2
A3
A4
A5
A6
A7
A8
B1
B2
B3
B4
B5
B6
B7
B8
mini keyboard
R5
22K
IC7
SN74HC377
CLK
1D
2D
3D
4D
5D
6D
7D
8D
1Q
2Q
3Q
4Q
5Q
6Q
7Q
8Q
address latch
IC6
PAL20R8B
I1
I2
I3
I4
I5
I6
I7
I8
I9
I10
I11
I12
CLK
OE
O1
O2
O3
O4
O5
O6
O7
O8
2Keyboard
2Touchscreen
J3
J9
J10
J11
J12
J13
J14
8 keys

Jumperreihe für Tastatur / Terminalanschluß



Bei gesetzten Jumpern ist die interne Minitastatur mit dem BASIC-Tiger®-Modul verbunden. Wahlweise kann hier über ein Flachkabel das BASIC-Tiger®-Terminal angeschlossen werden.

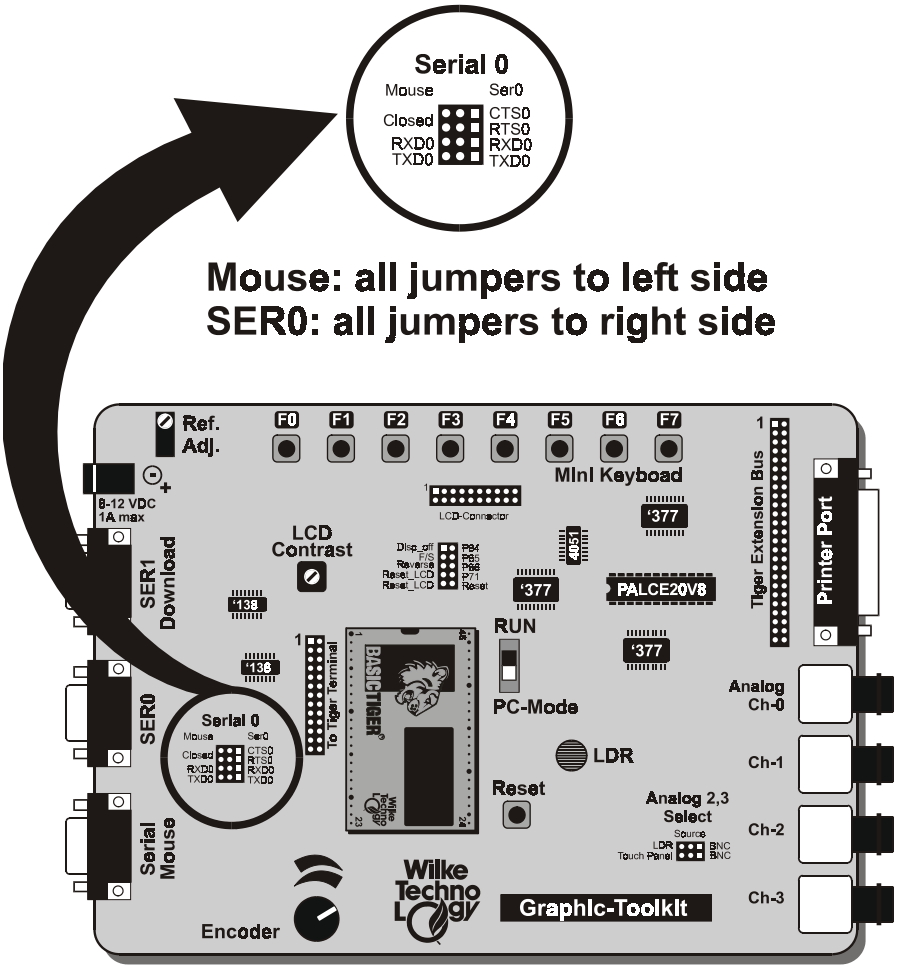
Achtung: wenn nur der Jumper ‚INE' gezogen ist, dann kann der Treiber der Tastatur einen aktiven Pegel auf die Leitungen des Port 6 schalten und so andere angeschlossene Komponenten stören. Mit VCC-Pegel wird der Treiber de-aktiviert.

Hinweis: der Device-Treiber LCD1.TDD wird umprogrammiert, um nur eine Tastaturspalte als Tasten zu scannen, alle andern (nicht vorhandenen) Spalten werden als DIP-Schalter gesehen. Bis zum Zeitpunkt der Umprogrammierung sind jedoch bereits eventuell Zeichen in den Tastaturpuffer gelangt. Daher muß dieser gelöscht werden, bevor das Programm die Tastatur verwendet.

Programmbeispiel:

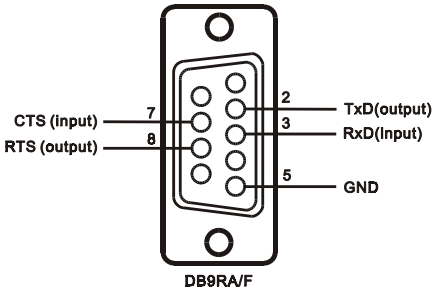
```
'---------------------------------------------------------------------
' Name: MINIKEY.TIG
' Gibt auf dem LCD den Scan-Code der gedrückten Taste aus.
'---------------------------------------------------------------------
user_var_strict                  ' Vars deklarieren!
#include DEFINE_A.INC            ' allgemeine Definitionen
#include UFUNC3.INC              ' User-Function-Codes
#include LCD_4.INC               ' Definitionen fuer LCD Typ 4
#include GR_TK1.INC              ' Definitionen fuer Grafik-Toolkit

TASK Main                        ' Beginn Task MAIN
  STRING Char$(1)

  call Init_LCDpins              ' initialisiere LCD-Pins
                                 ' LCD-4=240x128, 150 KB/s
  INSTALL DEVICE #LCD2,"LCD-6963.TDD",0,0,0EEH,4,150,11H
' Fuer Tastatur LCD1-Treiber installieren (BASIC-Tiger)
  INSTALL DEVICE #LCD,"LCD1.TDD", &
                        0,0,0,0,0,0,0eeh,0,0eeh,0,0,0eeh,0ffh,0ffh
' Fuer Tastatur LCD1-Treiber installieren (TINY-Tiger)
' INSTALL DEVICE #LCD, "LCD1.TDD", &
'                       0,0,0,0,0,0,80h,8,0eeh,0,0,0eeh,0ffh,0ffh

  put #lcd ,&                    ' nur eine Spalte als Tasten scannen
  "<1bh>D<16><1><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0><0f0H>"
                                 ' Formatierung fuer dezimal und HEX
  wait_duration 50               ' warte bis Scan neue Tabelle benutzt
  put #LCD, #0, #UFCO_IBU_ERASE, 0 ' bisherigen scan loeschen

  print #LCD2, "press one of 8 keys";
  using "UD<4><1>   0.0.0.0.4UH<2><2>   0.0.0.0.2"
  while 0 = 0                    ' Endlosschleife
    get #LCD, #0, 1, Char$       ' Warten auf Zeichen
    if Char$ <> "" then
      print_using #LCD2, "<1Bh>A<3><9><0F0h>Key = "; asc(Char$);&
                         " ($"; ASC(Char$); ")";"      "
    endif
  endwhile
END
```

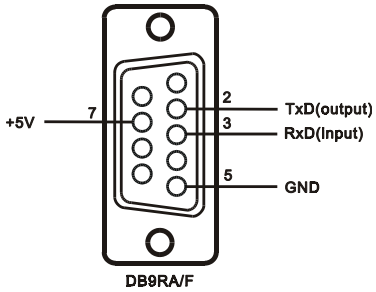
Serielle Schnittstellen

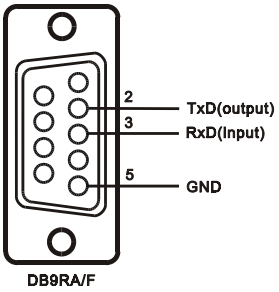


Jumper der seriellen Schnittstelle

Das BASIC-Tiger® Graphic-Toolkit führt beide RS232-Schnittstellen über Treiber auf DB-Steckverbinder. Die serielle Schnittstelle 0 kann durch Jumper wahlweise auf den DB9-Verbinder ‚SER0' oder den PC-Maus-Anschluß gelegt werden. In der Jumperstellung ‚Mouse' werden RTS0 und CTS0 verbunden.







Maus

Um eine **Microsoft-Maus** an dem BASIC-Tiger® betreiben zu können, ist ein spezieller serieller Device-Treiber zu verwenden (SER1C_xxx.TDD). Dieser Treiber stellt die von der Maus gelieferten Daten aufbereitet zur Verfügung. Das folgende Beispielprogramm zeigt die Verwendung des seriellen Maus-Treibers. Die Bewegungen und Tastendrücke werden auf dem LCD in Textform angezeigt. Ein weiteres Beispiel zeigt danach die Verwendung der Maus in einer grafischen Anwendung.

Achtung: Während des Download-Vorgangs dürfen auf auf SER0 keine Daten gesendet werden. Bewegen Sie die Maus während des Downloads nicht oder ziehen Sie den Stecker ab.

Programmbeispiel:

```
'-------------------------------------------------------------------------
' Name: MOUSE1.TIG
' Demonstriert, wie Mausdaten gelesen werden.
' Zeigt Mausdaten an (auf Textbildschirm)
'-------------------------------------------------------------------------
' L71 mit LCD-Reset verbinden
'-------------------------------------------------------------------------
user_var_strict                  ' Vars deklarieren!
#include DEFINE_A.INC            ' allgemeine Definitionen
#include UFUNC3.INC              ' User-Function-Codes
#include LCD_4.INC               ' Definitionen fuer LCD Typ 4
#include GR_TK1.INC              ' Definitionen fuer Graphic Toolkit

LONG key                         ' Maustasten
LONG mx,my                       ' Mauskoordinaten

'-------------------------------------------------------------------------
' Hauptprogramm
'-------------------------------------------------------------------------
TASK Main
  LONG x, y

  call Init_LCDpins               ' init LCD
                                        ' LCD-4=240x128, 150 KB/s
  install_device #LCD2,"LCD-6963.TDD",0,0,0EEH,4,150,11H
                                  ' spezieller ser. Treiber fuer Maus
  install_device #SER,"SER1C_K1.TDD",&
    "M", DP_8N, YES, BD_38_400, DP_8E, YES

  print #LCD2, "<1bh>A<0><0><0f0h>please move mouse";
  print #LCD2, "<1bh>A<0><1><0f0h>and press mouse keys";

  while 0 = 0                     ' Endlosschleife
    get #SER, #12H, 4, key        ' Mausstatus lesen
    if key bitand 4 <> 0 then     ' wenn gueltiges erstes Mausbyte
      get #SER, #10H, 4, x        ' Maus-Increments
      get #SER, #11H, 4, y        ' Maus-Increments
      print #LCD2, "<1bh>A<5><6><0f0h>x:";x;" y:";y;"        ";

      if key bitand 1 <> 0 then   ' pruefe auf rechte Maustaste
        print #LCD2, "<1bh>A<15><8><0f0h>right key";
      else
        print #LCD2, "<1bh>A<15><8><0f0h>         ";
      endif
      if key bitand 2 <> 0 then   ' pruefe auf linke Maustaste
        print #LCD2, "<1bh>A<4><8><0f0h>left key";
      else
        print #LCD2, "<1bh>A<4><8><0f0h>        ";
      endif
    endif
    wait_duration 50
  endwhile
END

' zur Information
'-------------------------------------------------------------------------
' Mausdaten
'-------------------------------------------------------------------------
```

```
' Byte:           <---1--->  <---2--->  <---3--->  <---4--->  <---5--->
'
' Sek-ADR:
'   00 hex  ===>  <------X-Pos------->  <------Y-Pos------->  <--Keys->
'                 <---signed WORD---->  <---signed WORD---->  <Bit-1,0>
'                                                            000000LR
'
'   10 hex  ===>  <-----------------X-Pos----------------->
'                  signed numerical Result (LONG, WORD, BYTE)
'                  Wenn die X-Pos abgefragt wird, wird sofort eine
'                  Kopie der akt. Y-Pos angefertigt, die man dann
'                  anschliessend abfragen kann.
'
'   11 hex  ===>  <-----------------Y-Pos----------------->
'                  Hier wird die mit SAdr=10H angefertigte Kopie gezeigt
'
'   12 hex  ===>  <----00000000000000000000000000MCLR---->
'                  Bit-3 = Movement, Bit-2 = Change (key or movement),
'                  Bit-1 = LEFT,     Bit-0 = RIGHT Mouse-Key.
'                    "1" = Key pressed
'-------------------------------------------------------------------------
```


Die folgende grafische Mausanwendung baut aus 8 Grafikbildern eine große Grafik zusammen. Mit gedrückter linker Maustaste verschiebt jede Mausbewegung die Grafik. Die rechte Maustaste invertiert das Bild. Der Mauscursor selbst ist ein ‚Smilie' und wird im Modus ‚Verschieben' zu einem Symbol mit 4 Pfeilen.

Programmbeispiel:

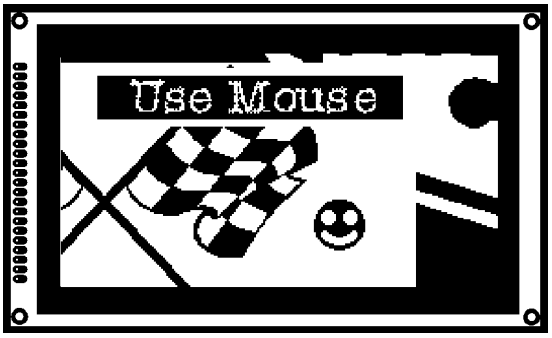
```
'-----------------------------------------------------------------------
' Name: MOUSE2.TIG
' Zeigt einen Mauscursor,
' mit gedrueckter linker Maustaste verschiebt sich der Hintergrund,
' gedrueckte rechte Maustaste invertiert das Bild
'-----------------------------------------------------------------------
' L71 mit LCD-Reset verbinden
'-----------------------------------------------------------------------
user_var_strict                  ' Vars deklarieren!
#include DEFINE_A.INC            ' allgemeine Definitionen
#include UFUNC3.INC              ' User-Function-Codes
#include LCD_4.INC               ' Definitionen fuer LCD Typ 4
#include GR_TK1.INC              ' Definitionen fuer Graphic Toolkit

LONG n, key, mscale
LONG x, y
LONG mx,my, mxp,myp, mx2n,my2n, mx2p,my2p' Mauskoordinaten
LONG deltax, deltay, deltax2, deltay2 ' effective Bewegungen auf Screen
LONG xabs,yabs, xabs2,yabs2      ' abs. Shifts in big$
LONG ptr
STRING big$(31K)                 ' Big Area for 8 screens 240 x 128
STRING Screen$(GR_SIZE)

DATALABEL  L_CURS_A1,L_CURS_A2,L_CURS_B1,L_CURS_B2
DATALABEL  PICT1, PIC_240_256, USE_MOUSE

'-----------------------------------------------------------------------
' Hauptprogramm
' Treiber laden, Variable initialisieren, Tasks starten
'-----------------------------------------------------------------------
TASK Main
L_CURS_A1::
  DATA FILTER "CURS_A1.BMP","GRAPHFLT",0        ' OR-Mask       32 x 32
L_CURS_A2::
  DATA FILTER "CURS_A2.BMP","GRAPHFLT",0        ' AND-Mask      32 x 32
L_CURS_B1::
  DATA FILTER "CURS_B1.BMP","GRAPHFLT",0        ' OR-Mask       32 x 32
L_CURS_B2::
  DATA FILTER "CURS_B2.BMP","GRAPHFLT",0        ' AND-Mask      32 x 32

' Bilder, die zum Hintergrund zusammengestellt werden
PICT1::
  DATA FILTER "SS1_002.BMP","GRAPHFLT",0        ' "Power"      240 x 128
  DATA FILTER "SS1_004.BMP","GRAPHFLT",0        ' "tiny"       240 x 128

  DATA FILTER "SS1_018.BMP","GRAPHFLT",0        ' "flags"      240 x 128
  DATA FILTER "SS1_025.BMP","GRAPHFLT",0        ' "Indu-buldg" 240 x 128

  DATA FILTER "SS1_013.BMP","GRAPHFLT",0        ' "#1"         240 x 128
  DATA FILTER "SS1_024.BMP","GRAPHFLT",0        ' "Industry"   240 x 128

  DATA FILTER "SS1_014.BMP","GRAPHFLT",0        ' "red.cost"   240 x 128
  DATA FILTER "SS1_015.BMP","GRAPHFLT",0        ' "dollar"     240 x 128
USE_MOUSE::
  DATA FILTER "MDEM_T02.BMP","GRAPHFLT",0 ' "Use Mouse"  176 x 32

  call Init_LCDpins              ' init LCD
                                 ' LCD-4=240x128, 150 KB/s
```

```
                                ' spezieller ser. Treiber fuer Maus
  install_device #SER,"SER1C_K1.TDD",&
    "M", DP_8N, YES, BD_38_400, DP_8E, YES

  put #LCD2, CURSOR_OFF         ' text-Cursor aus
  screen$ = fill$ ( "<0>", GR_SIZE ) ' initialisiere screen$
  big$ = fill$ ("<00>",31K)     ' Big$ mit weissen Pixeln besetzen
  ptr = pict1                   ' FLASH-Ptr auf eine Bild setzen

'-------------------------------------------------------------------------
' Baue einen grossen String mit 8 Bildschirmen
'-------------------------------------------------------------------------
'
'    !-----!-----!
'    !  0  !  1  !
'    !-----!-----!
'    !  2  !  3  !
'    !-----!-----!
'    !  4  !  5  !
'    !-----!-----!
'    !  6  !  7  !
'    !-----!-----!
  for y = 0 to 3*128 step 128
    for x = 0 to 240 step 240
      graphic_copy ( &
        big$, &                 ' Ziel
        ptr, &                  ' Quelle
        480,512, &              ' Zielformat
        X,Y, &                  ' Zielposition
        240,128, &              ' Quellformat
        0,0, &                  ' von Quellposition
        240,128, &              ' Quellfenster
        0)                      ' Modus
      ptr=PTR+240*128/8
    next
  next

  mscale =   1                  ' Maus-Skalierungsfaktor
  xabs   =  64                  ' abs-Shift von BIG in SCREEN
  yabs   = 128

  mx     = 120 * mscale         ' Maus-X2 vorige Position * Faktor
  my     =  64 * mscale
  mx2p = mx                     ' vorige Maus-X2-Position
  my2p = my

' zeige Anfangsbild
' zuerst ein Fenster aus big$ ausschneiden
  graphic_copy ( &
    screen$, &                  ' Ziel
    big$, &                     ' Quelle
    240,128, &                  ' Zielformat
    0,0, &                      ' Zielposition
    480,512, &                  ' Quellformat
    xabs,yabs, &                ' von Quellposition
    240,128, &                  ' Quellfenster
    0)                          ' Modus
' Text 'use-mouse' einfuegen
  graphic_copy ( &
    screen$, &                  ' Ziel
```

```
    240,128, &                    ' Zielformat
    16,8, &                       ' Zielposition
    176,32, &                     ' Quellformat
    0,0, &                        ' von Quellposition
    176,32, &                     ' Quellfenster
    0)                            ' Modus Punch
' Mauscursor einfuegen
    graphic_copy ( &
      screen$, &                  ' Ziel
      L_CURS_A2, &                ' Quelle
      240,128, &                  ' Zielformat
      mx,my, &                    ' Zielposition
      32,32, &                    ' Quellformat
      0,0, &                      ' von Quellposition
      32,32, &                    ' Quellfenster
      2)                          ' Modus AND
    graphic_copy ( &
      screen$, &                  ' Ziel
      L_CURS_A1, &                ' Quelle
      240,128, &                  ' Zielformat
      mx,my, &                    ' Zielposition
      32,32, &                    ' Quellformat
      0,0, &                      ' von Quellposition
      32,32, &                    ' Quellfenster
      1)                          ' Modus OR
  put #LCD2, #1, screen$, 0, 0, GR_SIZE

'-------------------------------------------------------------------
' Mausdaten
'-------------------------------------------------------------------
'                 lowest                                        highest
' Byte:          <---1--->  <---2--->  <---3--->  <---4--->  <---5--->
'
' Sek-ADR:
'   00 hex  ===>  <------X-Pos------->  <------Y-Pos------->  <--Keys->
'                 <---signed WORD---->  <---signed WORD---->  <Bit-1,0>
'                                                            000000LR
'
'   10 hex  ===>  <------------------X-Pos----------------->
'                  signed numerical Result (LONG, WORD, BYTE)
'                  Wenn die X-Pos abgefragt wird, wird sofort eine
'                  Kopie der akt. Y-Pos angefertigt, die man dann
'                  anschliessend abfragen kann.
'
'   11 hex  ===>  <------------------Y-Pos----------------->
'                  Hier wird die mit SAdr=10H angefertigte Kopie gezeigt
'
'   12 hex  ===>  <----00000000000000000000000000MCLR---->
'                  Bit-3 = Movement, Bit-2 = Change (key or movement),
'                  Bit-1 = LEFT,     Bit-0 = RIGHT Mouse-Key.
'                    "1" = Key pressed
'-------------------------------------------------------------------

  while 0 = 0                     ' Endlosschleife
    get #SER, #12H, 4, key        ' Mausstatus lesen
    if key bitand 4 <> 0 then
      get #SER, #10H, 4, x        ' Maus-Increments
      get #SER, #11H, 4, y        ' Maus-Increments

      mx2n = limit (mx2p+x,0,(240-32)*mscale)' neue X-Position
```

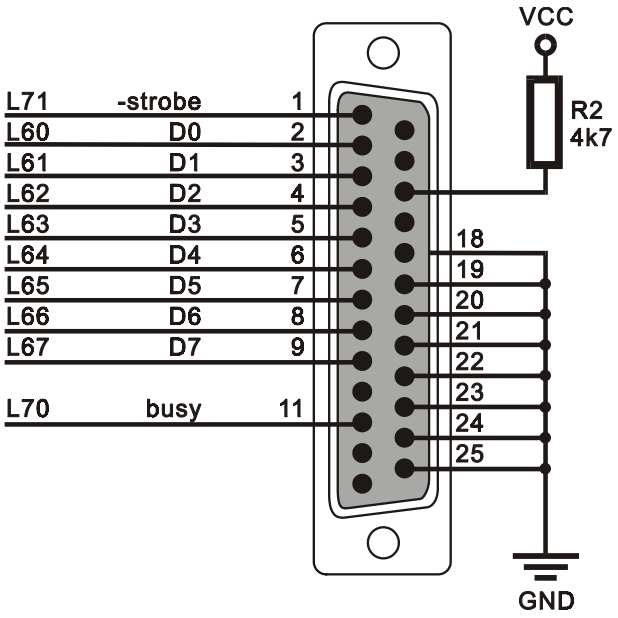
```
                                ' effective Delta2's:
      deltax2 = mx2n - mx2p     ' Delta-X2 <-- Mouse-X2 new - Mouse-X2
      deltay2 = my2n - my2p     ' Delta-Y2 <-- Mouse-Y2 new - Mouse-Y2

      mxp = mx                  ' vorige Maus-X-Position
      myp = my
      mx2p = mx2n               ' vorige Maus-X2-Position
      my2p = my2n

      mx = (mx2n-deltax2)/mscale  ' neue akt. Mouse-X Position on LCD
      my = (my2n-deltay2)/mscale

      deltax = mx - mxp         ' effective Delta-Y
      deltay = my - myp

      if key bitand 2 <> 0 then   ' linke Maustaste schiebt BIG$
        xabs = limit (xabs - deltax, 0, 240)
        yabs = limit (yabs - deltay, 0, 384)
      endif

'-------------------------------------------------------------------
' schneide ein Fenster aus big$
      graphic_copy ( &
        screen$, &              ' Ziel
        big$, &                 ' Quelle
        240,128, &              ' Zielformat
        0,0, &                  ' Zielposition
        480,512, &              ' Quellformat
        xabs,yabs, &            ' von Quellposition
        240,128, &              ' Quellfenster
        0)                      ' Modus
      if key bitand 1 <> 0 then   ' pruefe auf rechte Maustaste
        screen$ = invert ( screen$, 0, GR_SIZE) ' invertiere Bildschirm
      endif

'-------------------------------------------------------------------
' kopiere Message in Screen: "Use Mouse"
      graphic_copy ( &
        screen$, &              ' Ziel
        use_mouse, &            ' Quelle
        240,128, &              ' Zielformat
        16,8, &                 ' Zielposition
        176,32, &               ' Quellformat
        0,0, &                  ' von Quellposition
        176,32, &               ' Quellfenster
        0)                      ' Modus Punch

'-------------------------------------------------------------------
      if key bitand 2 = 0 then    ' pruefe auf linke Maustaste
' kopiere Cursor "smile", wenn linke Maustaste NICHT gedrueckt ist
        graphic_copy ( &
          screen$, &            ' Ziel
          L_CURS_A2, &          ' Quelle
          240,128, &            ' Zielformat
          mx,my, &              ' Zielposition
          32,32, &              ' Quellformat
          0,0, &                ' von Quellposition
          32,32, &              ' Quellfenster
          2)                    ' Modus AND
```

```
          screen$, &              ' Ziel
          L_CURS_A1, &            ' Quelle
          240,128, &              ' Zielformat
          mx,my, &                ' Zielposition
          32,32, &                ' Quellformat
          0,0, &                  ' von Quellposition
          32,32, &                ' Quellfenster
          1)                      ' Modus OR
'-------------------------------------------------------------------------
      else
' kopiere Cursor "arrows", wenn linke Maustaste GEDRUECKT ist.
        graphic_copy ( &
          screen$, &              ' Ziel
          L_CURS_B2, &            ' Quelle
          240,128, &              ' Zielformat
          mx,my, &                ' Zielposition
          32,32, &                ' Quellformat
          0,0, &                  ' von Quellposition
          32,32, &                ' Quellfenster
          2)                      ' Modus AND
        graphic_copy ( &
          screen$, &              ' Ziel
          L_CURS_B1, &            ' Quelle
          240,128, &              ' Zielformat
          mx,my, &                ' Zielposition
          32,32, &                ' Quellformat
          0,0, &                  ' von Quellposition
          32,32, &                ' Quellfenster
          1)                      ' Modus OR
      endif
      put #LCD2, #1, screen$, 0, 0, GR_SIZE ' zeige Fenster aus big$
    endif
    wait_duration 50
  endwhile
END
```



Drucker Port

An den Druckerport des BASIC-Tiger® Graphic-Toolkit kann ein paralleler Drucker angeschlossen werden. Neben dem Datenbus L60...L67 werden die Leitungen L70 für ‚-strobe' und L71 für ‚busy' verwendet.



Programmbeispiel:

```
'------------------------------------------------------------------------
' Name: PRINTER1.TIG
' Ausgabe auf Druckerschnittstelle
'------------------------------------------------------------------------
TASK Main
                                        ' LCD-4=240x128, 150 KB/s
  INSTALL DEVICE #9,"LCD-6963.TDD",0,0,0EEH,4,150,11H
  INSTALL_DEVICE #2,"PRN1_k4.TDD"       ' Druckerport-Treiber

  PRINT #9, "<1Bh>A<1><2><0F0h>print test on printer port"
  PRINT #2, "<10><13>Print test on parallel port"
  PRINT #2, "   Test finished<12>";
END
```

Analog Eingänge

Die Analogeingänge des BASIC-Tiger® Graphic-Toolkit führen über Operationsverstärkerschaltungen, die als Impedanzwandler geschaltetet sind und die Spannungen an den vier BASIC-Tiger® Analogeingängen begrenzen.

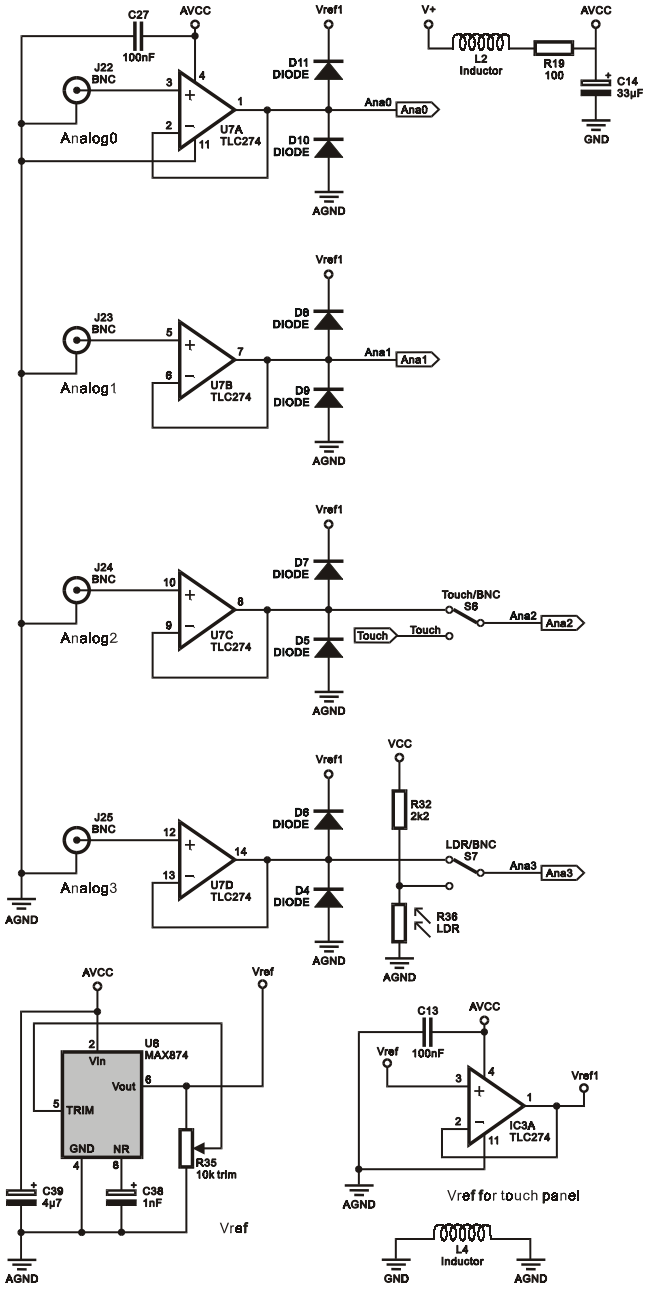
Der Analogkanal 2 ist entweder auf die BNC-Buchse oder auf das Touchpanel mittels Jumper schaltbar. Der Analogkanal 3 ist entweder auf die BNC-Buchse oder auf den Fotowiderstand (LDR) jumperbar.

Analogkanal	entweder	oder
Kanal 2	BNC	Touchpanel
Kanal3	BNC	Fotosensor (LDR)

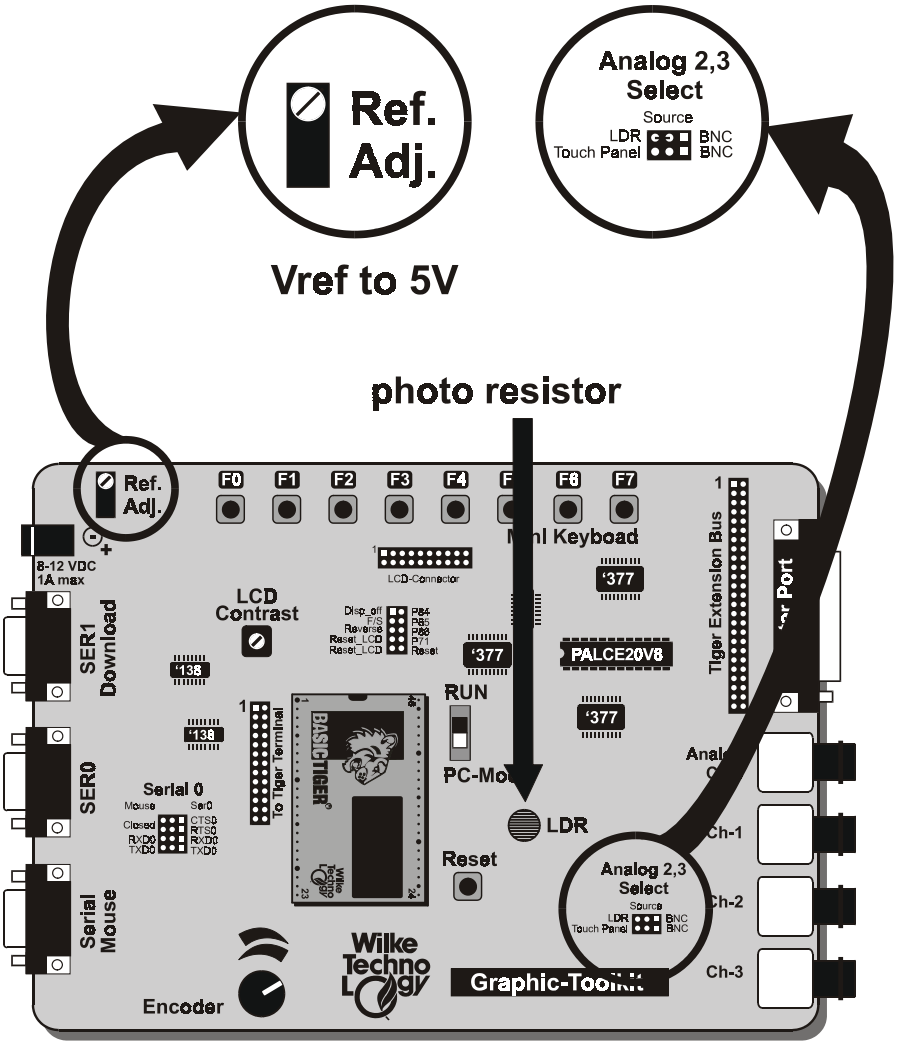
Es steht eine einstellbare Referenzspannungsquelle zur Verfügung, die direkt mit dem Vref-Pin des BASIC-Tiger® verbunden ist. Von dieser Referenzspannung wird ebenfalls die stabile Spannung für das Touchpanel abgeleitet.

Einige der folgenden Beispielprogramme verwenden als Signalquelle die analogen Eingänge.

Analogteil des BASIC-Tiger® Graphic-Toolkits

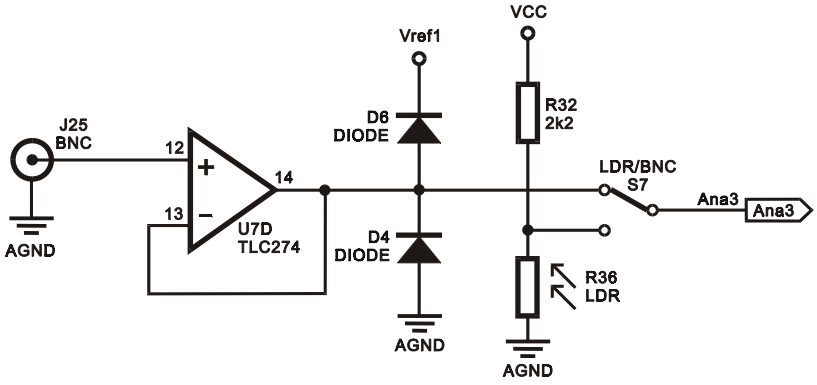


Fotowiderstand



Jumper und Einstellung Vref

Der Fotowiderstand ist über einen Jumper mit dem Analogkanal 3 des BASIC-Tiger® verbunden. Zur Demonstration dient hier das Programm **FOTO_R.TIG**, welches den Spannungsmeßwert in großen Ziffern darstellt.



Programmbeispiel:

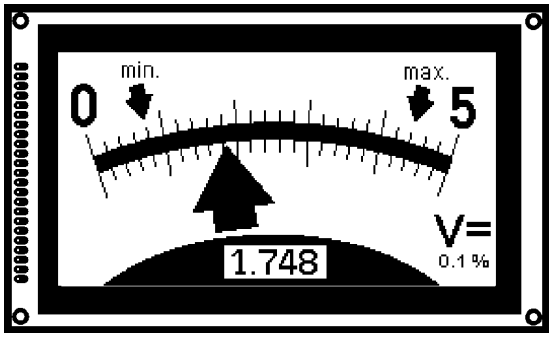
```
'------------------------------------------------------------------------
' Name: FOTO_R.TIG
' Der Wert des Fotosensors wird per AD-Wandler digitalisiert und in
' Grafikform in grossen Ziffern auf dem Grafikdisplay angezeigt.
'------------------------------------------------------------------------
' L71 mit LCD-Reset verbinden
' Vref auf 5V einstellen
'------------------------------------------------------------------------
user_var_strict                   ' erzwinge Var-Deklaration
#include DEFINE_A.INC             ' allgemeine Definitionen
#include UFUNC3.INC               ' User-Function-Codes
#include LCD_4.INC                ' Definitionen fuer LCD Typ 4
#include GR_TK1.INC               ' Definitionen fuer Graphic Toolkit

STRING Screen$(GR_SIZE), Scr$(GR_SIZE), tmp$(GR6_SIZE)

TASK Main                         ' Beginn Task MAIN
  LONG dwAnalog

  call Init_LCDpins               ' initialisiere LCD-Pins
                                  ' LCD-4=240x128, 150 KB/s
  install_device #LCD2, "LCD-6963.TDD",0,0,0EEH,4,150,11H
  install_device #AD1, "ANALOG1.TDD" ' Analog-In installieren
                                     '
  put #LCD2, CURSOR_OFF           ' Text-Cursor aus
  screen$ = fill$ ( "<0>", GR_SIZE )

  while 0 = 0                     ' Endlosschleife
    get #AD1,#3,1,dwAnalog        ' Foto-R an Analog 3 lesen
    dwAnalog = 5000 * dwAnalog / 255 ' skaliere auf 5V
    call BigDigits ( dwAnalog )   ' und graphisch darstellen
    wait_duration 200
  endwhile
END


'------------------------------------------------------------------------
' BigDigits
' stellt Zahlen von -9999 bis 9999 in grossen Ziffern graphisch dar.
'------------------------------------------------------------------------
SUB BigDigits ( LONG dwNumber )
DATALABEL DIGITS
Digits::
  DATA FILTER "Fnt16x20.BMP", "GRAPHFLT", 0
  LONG dwTmp, thousands, hundreds, tens, ones
  LONG xpos, ypos

  dwTmp = dwNumber
  thousands = dwTmp / 1000        ' Tausender
  dwTmp = dwTmp - thousands*1000
  hundreds = dwTmp / 100          ' Hunderter
  dwTmp = dwTmp - hundreds*100
  tens = dwTmp / 10               ' Zehner
  ones = dwTmp - tens*10

  xpos = 76                       ' X-Position in Pixel
  ypos = 50                       ' Y-Position in Pixel
```

```
  graphic_copy ( &                ' Tausender
    Scr$,Digits, &                ' Zielstring, Quelle (im Flash)
    COLUMNS, LINES, &             ' Zielgroesse
    xpos,ypos, &                  ' Zielposition
    16,260, &                     ' Quellgroesse
    0,thousands*20, &             ' Quellposition
    16,20, &                      ' kopierte Groesse
    0)                            ' Modus COPY
  graphic_copy ( &                ' Punkt
    Scr$,Digits, &                ' Zielstring, Quelle (im Flash)
    COLUMNS, LINES, &             ' Zielgroesse
    xpos+18,ypos, &               ' Zielposition
    16,260, &                     ' Quellgroesse
    0,11*20, &                    ' Quellposition
    16,20, &                      ' kopierte Groesse
    0)                            ' Modus COPY
  graphic_copy ( &                ' Hunderter
    Scr$,Digits, &                ' Zielstring, Quelle (im Flash)
    COLUMNS, LINES, &             ' Zielgroesse
    xpos+36,ypos, &               ' Zielposition
    16,260, &                     ' Quellgroesse
    0,hundreds*20, &              ' Quellposition
    16,20, &                      ' kopierte Groesse
    0)                            ' Modus COPY
  graphic_copy ( &                ' Zehner
    Scr$,Digits, &                ' Zielstring, Quelle (im Flash)
    COLUMNS, LINES, &             ' Zielgroesse
    xpos+54,ypos, &               ' Zielposition
    16,260, &                     ' Quellgroesse
    0,tens*20, &                  ' Quellposition
    16,20, &                      ' kopierte Groesse
    0)                            ' Modus COPY
  graphic_copy ( &                ' Einer                          one
    Scr$,Digits, &                ' Zielstring, Quelle (im Flash)
    COLUMNS, LINES, &             ' Zielgroesse
    xpos+72,ypos, &               ' Zielposition
    16,260, &                     ' Quellgroesse
    0,ones*20, &                  ' Quellposition
    16,20, &                      ' kopierte Groesse
    0)                            ' Modus COPY

  if LCD_MODE = LCD_MODE6X8 then  ' wenn LCD 6x8 Zeichensatz
    tmp$ = graphic_exp$ ( &       ' Pixel auf Bytes verteilen
      scr$, &                     ' Quellstring
      LCD_TXT_COL8X8,&            ' Quellbreite in Byte
      GR6_SIZE, &                 ' Ziellaenge in Pixel
      LCD_BYT_COL6X8, &           ' Zielbreite in Byte
      6, &                        ' Pixel/Byte im Zielstring
      0)                          ' Shifts nach links im Zielstring
  else
    tmp$ = let$ ( scr$ )
  endif
  put #LCD2, #1, tmp$, 0, 0, GR_SIZE ' zeige Grafik an
END
```


3.411

Meßinstrument

Derselbe Meßvorgang wird nun benutzt, um die Lichtmenge auf einem grafischen Zeigerinstrument anzuzeigen. Das Hintergrundbild ‚METER.BMP' enthält alle unveränderlichen Pixel des Instruments. Der bewegte Zeiger wird in Vektorgrafik erzeugt und wie die digitalen Ziffern in den Grafikstring mit dem Instrument eingeblendet.



Programmbeispiel:

```
'------------------------------------------------------------------------
' Name: METER.TIG
' Zeigt ein Mess-Instrument grafisch auf dem LCD
' Als Analoge Eingangsspannung dient der LDR auf dem Toolkit
'------------------------------------------------------------------------
' L71 mit LCD-Reset verbinden
' Vref auf 5V einstellen
'------------------------------------------------------------------------
user_var_strict                   ' erzwinge Var-Deklaration
#include DEFINE_A.INC             ' allgemeine Definitionen
#include UFUNC3.INC               ' User-Function-Codes
#include LCD_4.INC                ' Definitionen fuer LCD Typ 4
#include GR_TK1.INC               ' Definitionen fuer Graphic Toolkit
#include BIGDIGIT.INC             ' grosse Ziffern fuer Grafik-LD

#define AD 6                      ' Analog-Geraetenummer
#define VREF 5000                 ' Vref in millivolt
#define LIGHT 3                   ' LDR Analogkanal
#define MAR 31                    ' Pfeilradius des Instruments

'--------------------------------- globale Variable:
STRING Screen$(GR_SIZE)           ' Original-Bildschirminhalt
STRING Scr$(GR_SIZE)              ' Kopie des Bildschirminhalts
STRING tmp$(GR_SIZE)              ' fuer temporaere Operationen

DATALABEL METER_5V

'------------------------------------------------------------------------
' Hauptprogramm
'------------------------------------------------------------------------
TASK main
METER_5V::                        ' 240 x 128: "Voltmeter 5V range"
  data filter "METER_5V.BMP", "GRAPHFLT",0
  BYTE i, n, l, value
  LONG rot, volt, mvolt
  FIFO meter(16) of BYTE          ' speichert Analogwerte

  call Init_LCDpins               ' initialisiere LCD-Pins

  install_device #LCD2, "LCD-6963.TDD",0,0,0eeh,4,150,11h
  install_device #TA,  "TIMERA.TDD", 3,223 ' 700Hz (11kHz)
  install_device #AD,"ANALOG2.TDD"

  put #LCD2, CURSOR_OFF           ' Textcursor aus
  screen$ = peek_flash$ ( METER_5V, GR_SIZE )
  put #TA, 3, 223                 ' 700Hz

  put #AD, #0, #UFCO_AD2_RESO, 8  ' Aufloesung 8 Bit
  put #AD, #0, #UFCO_AD2_SCAN, 1  ' 1 Kanal abtasten
  put #AD, #0, #UFCO_AD2_CHAN, LIGHT ' diesen Kanal
  put #AD, #0, #UFCO_AD2_PSCAL, 120 ' Vorteiler fuer Analog-Device

  put #AD, meter                  ' starte A/D
  for i = 0 to 0 step 0           ' Endlosschleife
    l = 0
    while l = 0                   ' warte auf Messwert
      l = len_fifo ( meter )
    endwhile
```

```
    mvolt = (255-value) * (VREF/255)' millivolt (noch nicht fertig)
    volt  = mvolt / 1000           ' ganze Volt
    mvolt = mod ( mvolt, 1000 )    ' jetzt millivolt
    rot = (128-value) * (2600/128)' 2600-> 26 Grad

    tmp$ = fill$ ( "<0>", GR_SIZE )' Bereich fuer Pfeil
                                   ' Zeichne Pfeil fuer Instrument
    draw_line ( tmp$, 240, 128, & ' String, Groesse
             119, 270, &
             -2,2-MAR, &
             -2,0-MAR, &
             6000, 6000, &
             rot, &                ' Rotation des Pfeils
             3, &
             0)
    draw_next_line (-4,  0-MAR,  0)
    draw_next_line ( 0, -6-MAR,  0)
    draw_next_line ( 4,  0-MAR,  0)
    draw_next_line ( 2,  0-MAR,  0)
    draw_next_line ( 2,  2-MAR,  0)
    close_line (0)
    fill_area ( tmp$ )             ' fuelle Bildschirm mit "1"-Bits
    scr$ = or2$( screen$, tmp$ )   ' Verschmelze Hintergrund und Pfeil

    tmp$ = fill$ ( "<0>", GR_SIZE )' String fuer Ziffern
                        ' DEST,  no., Font-Index, Digits,   X, Y
    call printn_to_stri ( tmp$, volt,          4,      1,  95, 108, NO)
    call printn_to_stri ( tmp$,mvolt,          4,      3, 110, 108, NO)
    scr$ = xor1$ ( tmp$ )          ' blende Nummern ins Bild
    put #LCD2, #1, scr$, 0, 0, GR_SIZE ' zeige fertige Grafik an
  next
END
```

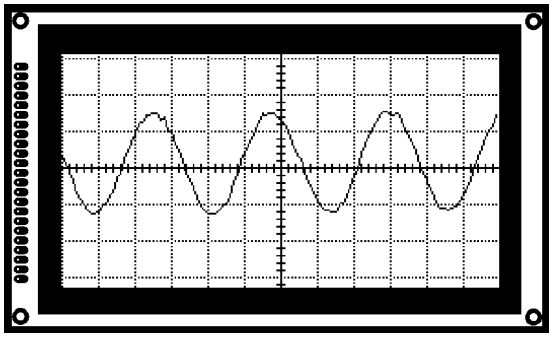
Oszilloskop

Die folgende Applikation zeigt ein Oszilloskop, welches die am Analogeingang 0 anliegende Spannung anzeigt. Dazu wird in diesem Beispiel abwechselnd ein Kurvenstück über einen Bildschirm aufgezeichnet und dann dargestellt. Die Geschwindigkeit des Aufzeichnens, also die Sample-Rate, ist begrenzt durch die maximale Frequenz des Device-Treibers TIMERA. Die Bildwiederholfrequenz ergibt sich aus den sich addierenden Zeiten, die zum Zeichnen der Kurve mit DRAW_LINE und DRAW_NEXT_LINE benötigt wird sowie der maximalen Ausgabegeschwindigkeit des LCD.

Programmbeispiel:


```
'------------------------------------------------------------------------
' Name: OSCAR1.TIG
' Oszilloskop mit den Werten des Analogkanals 0
'------------------------------------------------------------------------
' L71 mit LCD-Reset verbinden
'------------------------------------------------------------------------
user_var_strict                  ' Vars deklarieren!
#include DEFINE_A.INC            ' allgemeine Definitionen
#include UFUNC3.INC              ' User-Function-Codes
#include LCD_4.INC               ' Definitionen fuer LCD Typ 4
#include GR_TK1.INC              ' Definitionen fuer Graphic Toolkit

#define AD_QUANTITY 120
#define AD_STEP_X 2

STRING Screen$(GR_SIZE), sample$(AD_QUANTITY)
LONG line_x0, line_y0, line_x1, s
LONG value, ana_div
DATALABEL OscScrn

TASK MAIN
OscScrn::
  data filter "OscScrn.BMP", "GRAPHFLT", 0
  BYTE i

  lcd_mode = LCD_MODE30           ' 8x8 Zeichen
  call Init_LCDpins               ' initialisiere LCD-Pins
                                        ' LCD-4=240x128, 150 KB/s
  install_device #LCD2,"LCD-6963.TDD",0,0,0EEH,4,150,11H
  install_device #TA, "TIMERA.TDD", 2, 78 ' TimerA auf 8012 Hz
  install_device #AD1, "ANALOG2.TDD"

  line_y0 = 0
  put #LCD2, "<1Bh>c<20><0F0h>"   ' Text-Cursor aus
  ana_div = 2                     ' Sample mit 8 Bit, LCD: zeige 7 Bit

  put #AD1, #0, #UFCO_AD2_RESO, 8
  put #AD1, #0, #UFCO_AD2_SCAN, 1
  put #AD1, #0, #UFCO_AD2_CHAN, 0
  put #AD1, #0, #UFCO_AD2_PSCAL, 1

' lese x Samples von Kanal 0 in String
  for i = 0 to 0 step 0           ' Endlosschleife
```

```
    put #AD1, sample$, 0, AD_QUANTITY, 0 ' starte Sampling
    while sample$ = ""            ' warte bis String voll ist
    endwhile

' gebe Werte aus String mit DRAW-Line auf LCD
    peek_flash OscScrn, Screen$, GR_SIZE ' frische Bildschirmmaske
    line_x0 = 0                   ' erster Punkt
    line_y0 = (LINES/2) - (nfroms ( sample$, 0, 1 ) / ana_div)
                                  ' zweiter Punkt
    line_x1 = AD_STEP_X           ' X
    value = (LINES/2) - (nfroms ( sample$, 1, 1 ) / ana_div)
                                  ' setze Startpunkt
    draw_line ( Screen$, &        ' Zielstring
                COLUMNS, LINES, & ' Zielformat
                0, LINES/2, &     ' Referenzpunkt
                line_x0, line_y0, & ' erster Punkt
                line_x1, value, & ' zweiter Punkt
                1000, 1000, &     ' X-, Y-Scale
                0, &              ' Rotation
                0, &              ' schreibe schwarz
                0 )               ' Stift

    for s = 2 to ( len(sample$) - 1 ) ' zechne Linie fertig
      value = (LINES/2) - (nfroms ( sample$, s, 1 ) / ana_div)
      line_x1 = line_x1 + AD_STEP_X
      draw_next_line ( line_x1, value, 0 )
    next
    put #LCD2, #1, Screen$, 0, 0, GR_SIZE
  next
END
```



Oszilloskop-Schreiber

Die folgende Applikation zeigt ein Oszilloskop, welches die an den Analogeingängen 0 bis 3 anliegenden Spannung vierkanalig anzeigt. Dazu wird in diesem Beispiel lückenlos aufgezeichnet, die Kurve im Grafikstring ergänzt, horizontal verschoben und in Abständen auf dem Bildschirm dargestellt. Es entsteht der Eindruck einer wandernden Kurve, wie bei einem Schreiber. Die Geschwindigkeit des Aufzeichnens, also die Sample-Rate, muß so eingestellt sein, daß der FIFO-Puffer nie voll wird, da sonst die Messung abbricht. Die Bildwiederholfrequenz ergibt sich aus den sich addierenden Zeiten, die zum Zeichnen der Kurvenstücke mit DRAW_LINE und DRAW_NEXT_LINE und dem horizontalen Verschieben benötigt wird sowie der maximalen Ausgabegeschwindigkeit des LCD. Je schneller die Messung laufen soll, desto größer müssen die Kurvenstücke sein, die gezeichnet werden, bevor die Bildschirmdarstellung wieder aufgefrischt wird. Bei ganz langsamen Messungen kann jeder Pixelpunkt einzeln angezeigt werden. Das ist zum Beispiel bei Temperaturverläufen, Wettervorgängen oder der Fall.

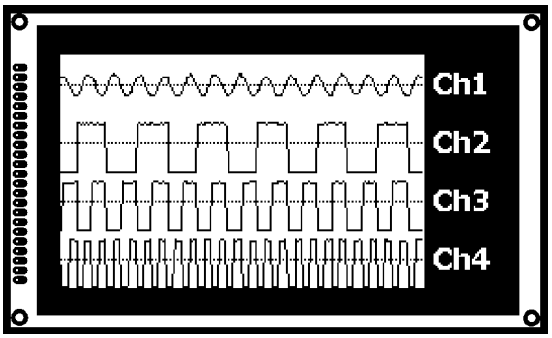
Programmbeispiel:

```
'------------------------------------------------------------------------
' Name: OSCAR3.TIG
' Zeigt einen Schreiber als wanderndes Bild auf dem LCD-Bildschirm.
' Es wird immer ein Kurvenfragment gezeichnet, dann  dargestellt,
' dann die Pixeldaten eine Fragmentgroesse weitergescrollt.
'------------------------------------------------------------------------
' L71 mit LCD-Reset verbinden
'------------------------------------------------------------------------
user_var_strict                   ' Vars deklarieren!
#include DEFINE_A.INC             ' allgemeine Definitionen
#include UFUNC3.INC               ' User-Function-Codes
#include LCD_4.INC                ' Definitionen fuer LCD Typ 4
#include GR_TK1.INC               ' Definitionen fuer Graphic Toolkit

                                  ' Bildschirmbereich f. Samplekurve
#define SCRN_XSIZ 200             ' X-Weite des Fensters der Anzeige
#define SCRN_YSIZ 128             ' Y-Weite des Fensters der Anzeige

                                  ' Null-Linie v. LCD-Top gezaehlt
#define PEN0POS SCRN_YSIZ/4       ' Y-Position Stift 0
#define PEN1POS SCRN_YSIZ/2       ' Y-Position Stift 1
#define PEN2POS 3*(SCRN_YSIZ/4)   ' Y-Position Stift 2
#define PEN3POS SCRN_YSIZ         ' Y-Position Stift 3             y pos

#define SCRN_DIV 2000             ' sichtbare msec auf LCD
#define TA_RANGE_INIT 3           ' TIMERA range             (156250Hz)
#define TA_DIV_INIT 217           ' TIMERA range           (/217=720Hz)
#define ANA_PSCAL 6               ' analog prescaler         (/6=120Hz)
#define DSP_FRQ 10                ' LCD display frequency

LONG smp_p_sec                    ' Samples/Sek.
LONG msec_p_scrn                  ' msec pro Bildschirm
LONG pix_act                      ' Pixel fuer Mess-Anzeige
LONG smp_p_dsp                    ' Samples zwischen Bildschirm-Refresh
```

```
LONG line_x0, line_y0, line_x1, s
LONG ana_div
LONG valch0, valch1, valch2, valch3
LONG d_xpos                      ' x-draw position start
LONG w_xpos                      ' x-draw position start
LONG ta_div
ARRAY ta_frq(3) of LONG
STRING Screen$(GR_SIZE)
STRING Scr$(GR_SIZE)
STRING tmp$(GR_SIZE)
STRING tmp1$(GR_SIZE)
STRING stripe$(384) '(3*LINES)          ' stripe of 3 Bytes

DATALABEL OscScrn

TASK MAIN
OscScrn::
  data filter "OscScrn4.BMP", "GRAPHFLT", 0
  BYTE i
  LONG fi, smp, l1, l2, l3
  FIFO sample(4096) of BYTE

  lcd_mode = LCD_MODE30           ' 8x8 Zeichen
  call Init_LCDpins               ' initialisiere LCD-Pins
                                          ' LCD-4=240x128, 150 KB/s
  install_device #LCD2,"LCD-6963.TDD",0,0,0EEH,4,150,11H
  install_device #TA, "TIMERA.TDD", TA_RANGE_INIT, TA_DIV_INIT
  install_device #AD1, "ANALOG2.TDD"

  put #LCD2, CURSOR_OFF           ' Text-Cursor aus
  stripe$ = fill$ ( "<0>", 3*LINES ) ' weisser Streifen f. Scroll
  peek_flash OscScrn, Screen$, GR_SIZE
  Scr$ = fill$ ( "<0>", GR_SIZE )
  ta_frq(0) = 2500000
  ta_frq(1) = 625000
  ta_frq(2) = 156250
  msec_p_scrn = SCRN_DIV          ' sichtbare msec per screen
  smp_p_sec = (SCRN_XSIZ * 1000) / msec_p_scrn ' samples per second
  ta_div = (ta_frq(TA_RANGE_INIT-1) / smp_p_sec) / ANA_PSCAL
  put #TA, TA_RANGE_INIT, ta_div
  smp_p_dsp = smp_p_sec / DSP_FRQ ' Samples pro Display-Refresh
  d_xpos = SCRN_XSIZ - smp_p_dsp - 1' x-Position f. neues Kurvenstueck
  w_xpos = SCRN_XSIZ - smp_p_dsp  ' x-Pos. weissen Streifen rechts an
  cop_siz = SCRN_XSIZ - smp_p_dsp ' Kopierbreite des weissen Streifens

  ana_div = (8000*LINES) / (SCRN_YSIZ*1000)' Sample Vertikalskalierung
  put #AD1, #0, #UFCO_AD2_RESO, 8 ' Aufloesung 8 Bit
  put #AD1, #0, #UFCO_AD2_SCAN, 4 ' 4 Kanaele
  put #AD1, #0, #UFCO_AD2_PSCAL, ANA_PSCAL
  put #AD1, sample                ' starte Analog-Kanal 0 in FIFO

' die folgende Zeile spaeter auskommentieren
' fuer Testphase
'  run_task Disp                  ' FIFO darf nicht voll werden

'# erster Wert ##########################################################
  s = len_fifo ( sample )         ' warte auf ersten Wert
  while s < 4                     ' von 4 Kanaelen
    s = len_fifo ( sample )
```

```
  get_fifo sample, valch0          ' hole alle Werte und scaliere
  get_fifo sample, valch1
  get_fifo sample, valch2
  get_fifo sample, valch3
  valch0 = -valch0 / ana_div       ' negativ, da Y von oben zaehlt
  valch1 = -valch1 / ana_div
  valch2 = -valch2 / ana_div
  valch3 = -valch3 / ana_div

'# loop ##########################################################################
  for i = 0 to 0 step 0            ' Endlosschleife
    s = len_fifo ( sample )
    s = s / 4                      ' 4 channels
    if s > smp_p_dsp then          ' wenn genug samples da sind
      line_x0 = d_xpos             ' setze Startpunkt auf LCD:
      draw_line ( Scr$, &          ' Zielstring
        COLUMNS, LINES, &          ' Zielformat
        0, PEN0POS, &              ' Referenzpunkt
        line_x0, valch0, &         ' erster Punkt
        line_x0, valch0, &         ' zweiter Punkt
        1000, 1000, &              ' X-, Y-Scale
        0, &                       ' Rotation
        0, &                       ' schreibe schwarz
        0 )                        ' Stift
      draw_line ( Scr$, &          ' Zielstring
        COLUMNS, LINES, &          ' Zielformat
        0, PEN1POS, &              ' Referenzpunkt
        line_x0, valch1, &         ' erster Punkt
        line_x0, valch1, &         ' zweiter Punkt
        1000, 1000, &              ' X-, Y-Scale
        0, &                       ' Rotation
        0, &                       ' schreibe schwarz
        1 )                        ' Stift
      draw_line ( Scr$, &          ' Zielstring
        COLUMNS, LINES, &          ' Zielformat
        0, PEN2POS, &              ' Referenzpunkt
        line_x0, valch2, &         ' erster Punkt
        line_x0, valch2, &         ' zweiter Punkt
        1000, 1000, &              ' X-, Y-Scale
        0, &                       ' Rotation
        0, &                       ' schreibe schwarz
        2 )                        ' Stift
      draw_line ( Scr$, &          ' Zielstring
        COLUMNS, LINES, &          ' Zielformat
        0, PEN3POS, &              ' Referenzpunkt
        line_x0, valch3, &         ' erster Punkt
        line_x0, valch3, &         ' zweiter Punkt
        1000, 1000, &              ' X-, Y-Scale
        0, &                       ' Rotation
        0, &                       ' schreibe schwarz
        3 )                        ' Stift
      for smp = 1 to smp_p_dsp     ' zeichne Analogkurve
        get_fifo sample, valch0
        get_fifo sample, valch1
        get_fifo sample, valch2
        get_fifo sample, valch3
        valch0 = -valch0 / ana_div
        valch1 = -valch1 / ana_div
        valch2 = -valch2 / ana_div
        valch3 = -valch3 / ana_div
```

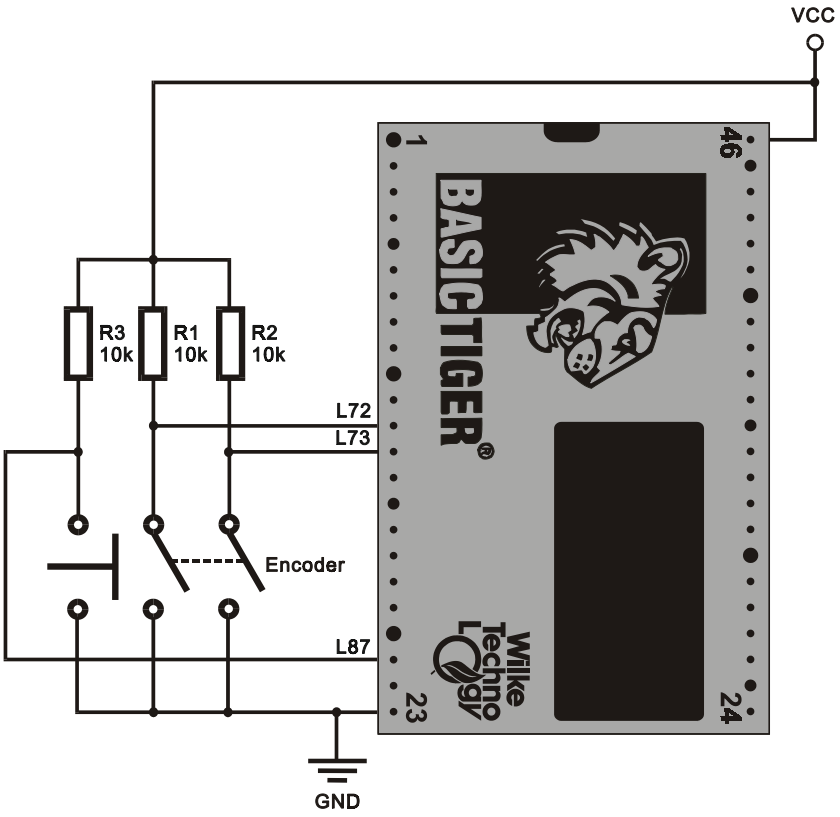
```
        draw_next_line ( line_x0, valch1, 1 )
        draw_next_line ( line_x0, valch2, 2 )
        draw_next_line ( line_x0, valch3, 3 )
        line_x0 = line_x0 + 1
      next

      tmp1$ = or2$ ( Screen$, Scr$ )
      put #LCD2, #1, tmp1$, 0, 0, GR_SIZE

' Mess-Bildschirminhalt nach links schieben
      graphic_copy (Scr$, &       ' Original LCD-Inhalt
        Scr$, &                   ' zu scrollen
        COLUMNS, LINES, &         ' Groesse Zielbereich
        0, 0, &                   ' Ziel der Kopie
        COLUMNS, LINES, &         ' Groesse Quellbereich
        smp_p_dsp, 0, &           ' von Quellposition
        cop_siz, SCRN_YSIZ+1, &     ' Groesse der zu kop. Grafik
        0)                        ' Modus Copy

' freigewordenes Stueck weiss fuellen fuer neues Messkurvenstueck
      graphic_copy (Scr$, &       ' Original LCD-Inhalt
        stripe$, &                ' weisser Streifen
        COLUMNS, LINES, &         ' Groesse Zielbereich
        w_xpos, 0, &              ' Ziel der Kopie
        24, LINES, &              ' Groesse Quellbereich
        0, 0, &                   ' von Quellposition
        smp_p_dsp, SCRN_YSIZ+1, &   ' Groesse der zu kop. Grafik
        0)                        ' Modus Copy
    endif
  next
END


'---------------------------------------------------------------------
' Fuer Testphase: zeigt FIFO-Fuellstand
'---------------------------------------------------------------------
Task Disp
  BYTE i

  for i = 0 to 0 step 0           ' Endlosschleife
    print #LCD2, "<1Bh>A<0><1><0F0h>len sample:";len_fifo ( sample );"
    wait_duration 200             ' gib der Grafik eine Chance
  next
END
```


Ch1
Ch2
Ch3
Ch4

Drehimpulsgeber

Auf dem BASIC-Tiger® Graphic-Toolkit befindet sich ein Drehimpulsgeber, der ohne Begrenzung in beide Richtungen drehbar ist. Die Drehung erzeugt zwei versetzte Pulsketten, woran sowohl die Drehrichtung als auch die Anzahl der Winkelschritte erkannt wird. Der Drehgeber wird durch einen speziellen Device-Treiber unterstützt. Programm-Menüs können mit dem Drehgeber durchrollt werden, und die zusätzliche Tastfunktion durch Drücken auf die Achse dient als Eingabetaste. Der Drehimpulsgeber ist an den Pins L72 und L73 fest angeschlossen. Der Taster ist mit L87 verbunden.

Ein einfaches Beispielprogramm zum Drehimpulsgeber gibt es im Verzeichnis EXMPLDEV unter dem Namen ENC1.TIG. Das Beispielprogramm zum BASIC-Tiger® Graphic-Toolkit verwendet den Drehgeber, um ein Beispielmenü zu bedienen. Die Menüpunkte sind frei erfunden. Es werden aus Untermenüs Parameter bei Tastendruck gesetzt.



Programmbeispiel:

```
'-----------------------------------------------------------------------
' Name: ENCMENU1.TIG
' Bedient ein Menue auf dem LCD-Bildschirm mit dem Encoder.
' Dieses Demo stellt ueber das Menue 3 (erfundene) Parameter ein.
' In echten Anwendungen kann das Einstellen der Parameterwerte
' durch andere Aktionen (weitere Unterprogramme, Untermenues, Tasks)
' ersetzt werden.
' Menuesystem:
' Alle Menues sind in einem Array gespeichert.
' In einem anderen Array sind die Positionen der ersten Zeile
' der Menues abgelegt.
' Ein drittes Array enthaelt den hoechsten Index jedes Menues.
' Die Bedienung des Menuesystems erfolgt mit dem Drehgeber.
' Ein Druck auf die Encoder-Achse entspricht EINGABE
'-----------------------------------------------------------------------
' L71 mit LCD-Reset verbinden
'-----------------------------------------------------------------------
user_var_strict                  ' Vars deklarieren!
#include DEFINE_A.INC            ' allgemeine Definitionen
#include UFUNC3.INC              ' User-Function-Codes
#include LCD_6.INC               ' Definitionen fuer LCD Typ 4
#include GR_TK1.INC              ' Definitionen fuer Graphic Toolkit


'-------------------------------- menu system
' Menues
#define MNU_MAIN 0                ' Hauptmenue
#define MNU_PARA10 1              ' menu for parameter 1
#define MNU_PARA20 2              ' menu for parameter 2
#define MNU_PARA30 3              ' menu for parameter 3
#define MNU_DEFAULT 4             ' menu for Menue 'default'
#define MNU_MAIN_LAST 5           ' Dimension des Menue-Arrays

                                  ' Menues mit max. 16 Zeilen
                                  ' jeder Eintrag max. 40 Zeichen lang
ARRAY menu_txt$(MNU_MAIN_LAST, 17) OF STRINGS(40)
ARRAY menu_pos(MNU_MAIN_LAST,2) OF LONG
ARRAY menu_last(MNU_MAIN_LAST) OF BYTE

' global variables
LONG m_ix                         ' Index fuer Menue
LONG main_ix                      ' Index fuer Main-Menue-Eintrag
LONG para10_ix                    ' Index fuer Para10-Menue-Eintrag
LONG para20_ix                    ' Index fuer Para20-Menue-Eintrag
LONG para30_ix                    ' Index fuer Para20-Menue-Eintrag
LONG mnu_max                      ' maximaler Menue-Eintrags-Index
BYTE mnu_mode_y

'-------------------------------- Encoder
LONG mov                          ' Encoder-Bewegung
BYTE bEncKey
BYTE enc_click

'-------------------------------- LCD
STRING mnuscr$(GR_SIZE)           ' Menue-Grafik
STRING tmpscr$(GR_SIZE)           ' Menue-Grafik
STRING black$(GR_SIZE)            ' schwarze Flaeche
STRING white$(GR_SIZE)            ' weisse Flaeche
```

```
LONG lcd_fnt_x, lcd_fnt_y

'--------------------------------- Anwendung
#define POS_PARA10 "<1bh>A<20><1><0f0h>"
#define FMT_PARA10 "UD<4><1>   0 0 0 3,3"
#define POS_PARA20 "<1bh>A<20><2><0f0h>"
#define FMT_PARA20 "UD<4><1>   0 0 0 3,3"
#define POS_PARA30 "<1bh>A<21><3><0f0h>"
#define FMT_PARA30 "UD<3><3>   0 0 0 1.2"

long para10, para20, para30

'--------------------------------------------------------------------
TASK MAIN
  call Init_LCDpins               ' initialisiere LCD-Pins
                                  ' LCD-4=240x128, 150 KB/s
  install_device #LCD2,"LCD-6963.TDD",0,0,0EEH,LCD_TYPE,150,11H
  install_device #TA, "TIMERA.TDD",3,156 ' Zeitbasis 1 kHz
  install_device #ENC, "ENC1_723.TDD" ' Encoder Port 7, Pins 2+3

  put #LCD2, TEXT_ON              ' Text und Grafik einschalten
  put #LCD2, GRAPHIC_ON
  put #LCD2, CURSOR_OFF           ' Text-Cursor aus
  put #LCD2, MODE_XOR             ' LCD-Modus
  black$ = fill$ ( "<255>", GR_SIZE )' schwarze Flaeche
  white$ = fill$ ( "<0>", GR_SIZE )  ' weisse Flaeche
  mnuscr$ = fill$ ( "<0>", GR_SIZE )
  lcd_txtclr$ = fill$ ( " ", 240 )

  dir_pin P_ENC_KEY, PIN_ENC_KEY, 1' Tastpin als Eingang
  put #ENC, 0                     ' setze Resetwert
  get #ENC, #1, 0, mov            ' und resette ihn
  call init_menu                  ' initialisiere Menue-System
  call ini_parameters
  main_ix = 0                     ' Menue-Inice
  para10_ix = 0
  para20_ix = 0
  para30_ix = 0

  para10 = 50                     ' init and display paramters
  para20 = 0
  para30 = 0
  using FMT_PARA10
  print_using #LCD2, POS_PARA10;para10;"Hz";
  using FMT_PARA20
  print_using #LCD2, POS_PARA20;para20;"%";
  using FMT_PARA30
  print_using #LCD2, POS_PARA30;para30;"V";

mnu0_init:                        ' menu main 0 ----------------------
  m_ix = MNU_MAIN                 ' Menue mit diesem Item selektiert
  call show_menu ( main_ix )      ' anzeigen

new_inp0:                         ' Schleifenanfang
  get #ENC, #1, 0, mov            ' Drehgeber bewegt?
  if mov <> 0 then
    main_ix = main_ix + sgn ( mov )' bewege 1 nach oben oder untern
    main_ix = limit ( main_ix, 0, mnu_max)
    call show_menu ( main_ix )    ' Menue mit diesem Item selektiert
```

```
  call m_input                    ' auf Encoderachse gedrueckt?
  if enc_click = YES then
    switch main_ix                ' bearbeite diesen Menuepunkt
      case 0:
        call menu_para10
        goto mnu0_init
      case 1:
        call menu_para20
        goto mnu0_init
      case 2:
        call menu_para30
        goto mnu0_init
      case 3:
        call ini_parameters
        using FMT_PARA10
        print_using #LCD2, POS_PARA10;para10;"Hz";
        using FMT_PARA20
        print_using #LCD2, POS_PARA20;para20;"%";
        using FMT_PARA30
        print_using #LCD2, POS_PARA30;para30;"V";
        goto mnu0_init
      default:
    endswitch
  endif

  goto new_inp0                   ' Schleifenende
END


'---------------------------------------------------------------------
' SUB menu_para10
'---------------------------------------------------------------------
SUB menu_para10
  BYTE k

  m_ix = MNU_PARA10
  call show_menu ( para10_ix )    ' Menue mit diesem Item selektiert

new_inp_para10:                   ' Schleifenanfang
  get #ENC, #1, 0, mov            ' Drehgeber bewegt?
  if mov <> 0 then                ' wenn ja
    para10_ix = para10_ix + sgn ( mov )' bewege nach oben oder untern
    para10_ix = limit ( para10_ix, 0, mnu_max)
    call show_menu ( para10_ix )  ' Menue mit diesem Item selektiert
  endif

  call m_input                    ' auf Encoderachse gedrueckt?
  if enc_click = YES then
    switch para10_ix              ' bearbeite diesen Menuepunkt
      case 0:
        para10 = 50
        goto end_menu_para10
      case 1:
        para10 = 100
        goto end_menu_para10
      case 2:
        para10 = 200
        goto end_menu_para10
      case 3:
```

```
        goto end_menu_para10
      case 4:
        para10 = 800
        goto end_menu_para10
      case 5:
        para10 = 1600
        goto end_menu_para10
      default:
    endswitch
  endif
  goto new_inp_para10             ' Schleifenende

end_menu_para10:                  ' Menue-Ausgang
  using FMT_PARA10
  print_using #LCD2, POS_PARA10;para10;"Hz";
  call hide_menu ( para10_ix )    ' Menue entfernen
END


'---------------------------------------------------------------------
' SUB menu_para20
'---------------------------------------------------------------------
SUB menu_para20
  BYTE k

  m_ix = MNU_PARA20
  mnu_mode_y = YES
  call show_menu ( para20_ix )    ' Menue mit diesem Item selektiert

new_inp_para20:                   ' Schleifenanfang
  get #ENC, #1, 0, mov            ' Drehgeber bewegt?
  if mov <> 0 then                ' wenn ja
    para20_ix = para20_ix + sgn ( mov )' bewege nach oben oder untern
    para20_ix = limit ( para20_ix, 0, mnu_max)
    call show_menu ( para20_ix )  ' Menue mit diesem Item selektiert
  endif

  call m_input                    ' auf Encoderachse gedrueckt?
  if enc_click = YES then
    switch para20_ix              ' bearbeite diesen Menuepunkt
      case 0:
        para20 = 0
        goto end_menu_para20
      case 1:
        para20 = 10
        goto end_menu_para20
      case 2:
        para20 = 20
        goto end_menu_para20
      case 3:
        para20 = 40
        goto end_menu_para20
      case 4:
        para20 = 60
        goto end_menu_para20
      case 5:
        para20 = 80
        goto end_menu_para20
      case 6:
        para20 = 100
```

```
      default:
    endswitch
  endif
  goto new_inp_para20             ' Schleifenende

end_menu_para20:
  using FMT_PARA20
  print_using #LCD2, POS_PARA20;para20;"%";
  call hide_menu ( para20_ix )    ' Menue entfernen
END


'---------------------------------------------------------------------
' SUB menu_para30
'---------------------------------------------------------------------
SUB menu_para30
  BYTE k

  m_ix = MNU_PARA30
  mnu_mode_y = YES
  call show_menu ( para30_ix )    ' Menue mit diesem Item selektiert

new_inp_para30:                   ' Schleifenanfang
  get #ENC, #1, 0, mov            ' Drehgeber bewegt?
  if mov <> 0 then                ' wenn ja
    para30_ix = para30_ix + sgn ( mov )' bewege nach oben oder untern
    para30_ix = limit ( para30_ix, 0, mnu_max)
    call show_menu ( para30_ix )  ' Menue mit diesem Item selektiert
  endif

  call m_input                    ' auf Encoderachse gedrueckt?
  if enc_click = YES then
    switch para30_ix              ' bearbeite diesen Menuepunkt
      case 0:
        para30 = 100
        goto end_menu_para30
      case 1:
        para30 = 225
        goto end_menu_para30
      case 2:
        para30 = 375
        goto end_menu_para30
      case 3:
        para30 = 500
        goto end_menu_para30
      default:
    endswitch
  endif
  goto new_inp_para30             ' Schleifenende

end_menu_para30:
  using FMT_PARA30
  print_using #LCD2, POS_PARA30;para30;"V";
  call hide_menu ( para30_ix )    ' Menue entfernen
END


'---------------------------------------------------------------------
' SUB show_menu
'---------------------------------------------------------------------
```

```
  BYTE mnu_x, mnu_y               ' X- und Y Positionen
  LONG item, mnuitemx, mnuitemy, mnuitxsiz, mnuitysiz
  STRING mpos$(5)

  mnu_x = menu_pos ( m_ix, 0 )    ' Position der ersten Zeile
  mnu_y = menu_pos ( m_ix, 1 )
  mnu_max = menu_last ( m_ix )    ' setze mnu_max fuer aufrufendes Prg
  item = 0                        ' beginne mit Menuepunkt 0

next_item:                        ' Schleifenbeginn
  if menu_txt$ ( m_ix, item ) = "" then
    goto show_menu_end            ' letzter Menuepunkt erreicht
  endif
  mpos$ = "<1Bh>A" + chr$(mnu_x) + chr$(mnu_y) + "<0F0h>"
  print #LCD2, mpos$;menu_txt$ ( m_ix, item );
                                  ' nun Grafik hinterlegen
                                  ' Menue-Eintrag X-Groesse in Pixel
  mnuitxsiz = 2 + (len (menu_txt$(m_ix, item)) * lcd_fnt_x)
  mnuitysiz = lcd_fnt_y + 2       ' Menue-Eintrag Y-Groesse in Pixel
  mnuitemx = (mnu_x * lcd_fnt_x)  ' Menue-Eintrags-X-Position in Pixel
  if mnu_y = 0 then               ' Y-Position an Pos. 0 anders
    mnuitemy = 0
  else
    mnuitemy = mnu_y * lcd_fnt_y - 1
  endif

  if item = me_ix then            ' wenn markierter Eintrag
    graphic_copy ( &              ' weisser Text auf schw. Hintergrund
      mnuscr$, &                  ' Ziel
      black$, &                   ' Quelle
      COLUMNS, LINES, &           ' Zielgroesse
      mnuitemx, mnuitemy, &       ' Position in Ziel
      COLUMNS, LINES, &           ' Quellgroesse
      0, 0, &                     ' Quellposition
      mnuitxsiz, mnuitysiz, &     ' kopierte Groesse
      0)                          ' Modus Kopieren
  else                            ' sonst (=nicht markiert)
    if item = 0 then
      graphic_copy ( &            ' weisser Text auf schw. Hintergrund
        mnuscr$, &                ' Ziel
        white$, &                 ' Quelle
        COLUMNS, LINES, &         ' Zielgroesse
        mnuitemx, mnuitemy, &     ' Position in Ziel
        COLUMNS, LINES, &         ' Quellgroesse
        0, 0, &                   ' Quellposition
        mnuitxsiz, mnuitysiz, &   ' kopierte Groesse
        0)                        ' Modus Kopieren
    else
      graphic_copy ( &            ' weisser Text auf schw. Hintergrund
        mnuscr$, &                ' Ziel
        white$, &                 ' Quelle
        COLUMNS, LINES, &         ' Zielgroesse
        mnuitemx, mnuitemy+1, &   ' Position in Ziel
        COLUMNS, LINES, &         ' Quellgroesse
        0, 0, &                   ' Quellposition
        mnuitxsiz, mnuitysiz-1, & ' kopierte Groesse
        0)                        ' Modus Kopieren
    endif
  endif
  mnu_y = mnu_y + 1               ' naechste Zeile
```

```
  goto next_item                    ' Schleifenende

show_menu_end:
  if LCD_MODE = LCD_MODE6X8 then    ' wenn LCD 6x8 Zeichensatz
    tmpscr$ = graphic_exp$ ( &
      mnuscr$, &                    ' source string
        LCD_TXT_COL8X8,&            ' source horizotal wide
        GR6_SIZE, &                 ' dest. length
        LCD_BYT_COL6X8, &           ' dest. horizotal wide
        6, &                        ' no. of dest. bits per byte
        0)                          ' shift left
  else
    tmpscr$ = let$ ( mnuscr$ )
  endif
  put #LCD2, #1, tmpscr$, 0, 0, GR6_SIZE ' zeige Grafik an
END


'---------------------------------------------------------------------
' SUB hide_menu
'---------------------------------------------------------------------
SUB hide_menu ( me_ix )
  BYTE mnu_x, mnu_y                 ' X- und Y Positionen
  LONG item, mnuitemx, mnuitemy, mnuitxsiz, mnuitysiz
  STRING mpos$(5), tmp$

  mnu_x = menu_pos ( m_ix, 0 )     ' Position der ersten Zeile
  mnu_y = menu_pos ( m_ix, 1 )
  item = 0

hide_next_item:                     ' Schleifenbeginn
  if menu_txt$ ( m_ix, item ) = "" then
    goto hide_menu_end              ' letzter Menuepunkt erreicht
  endif
  mpos$ = "<1Bh>A" + chr$(mnu_x) + chr$(mnu_y) + "<0F0h>"
  tmp$ = fill$ ( " ", len(menu_txt$ (m_ix, item)) )
  print #LCD2, mpos$;tmp$;          ' drucke Leerzeichen ueber Eintrag
                                    ' nun Grafik entfernen
                                    ' Menue-Eintrag X-Groesse in Pixel
  mnuitxsiz = 2 + (len (menu_txt$(m_ix, item)) * lcd_fnt_x)
  mnuitysiz = lcd_fnt_y + 2         ' Menue-Eintrag Y-Groesse in Pixel
  mnuitemx = (mnu_x * lcd_fnt_x)
  if mnu_y = 0 then
    mnuitemy = 0
  else
    mnuitemy = mnu_y * lcd_fnt_y - 1
  endif

  if item = me_ix then
    graphic_copy ( &                  ' weisser Text auf schw. Hintergrund
      mnuscr$, &                      ' Ziel
      white$, &                       ' Quelle
      COLUMNS, LINES, &               ' Zielgroesse
      mnuitemx, mnuitemy-1, &         ' Position in Ziel
      COLUMNS, LINES, &               ' Quellgroesse
      0, 0, &                         ' Quellposition
      mnuitxsiz, mnuitysiz+1, &       ' kopierte Groesse
      0)                              ' Modus Kopieren
  endif
  mnu_y = mnu_y + 1                 ' naechste Zeile
```

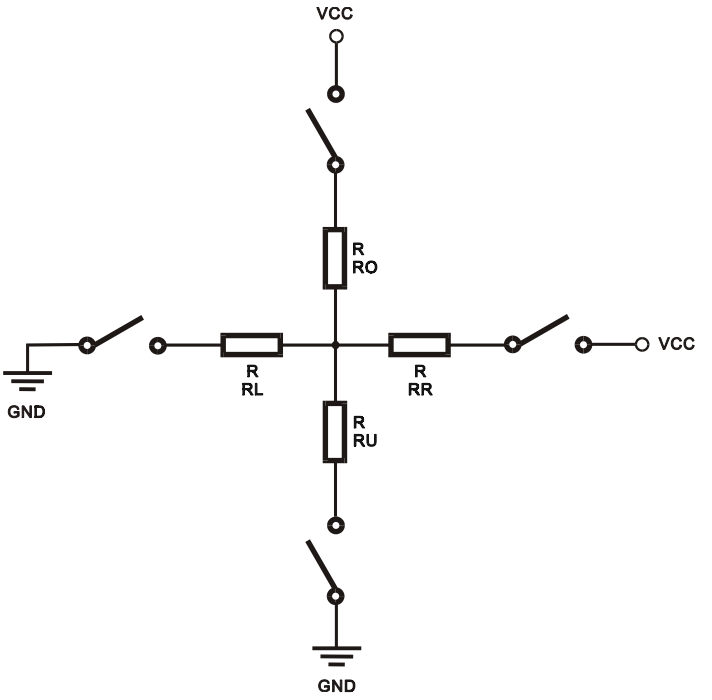
```
  goto hide_next_item             ' Schleifenende

hide_menu_end:
  if LCD_MODE = LCD_MODE40 then
    tmpscr$ = graphic_exp$ ( &
      mnuscr$, &                  ' source string
        30,&                      ' source horizotal wide
        GR6_SIZE, &               ' dest. length
        40, &                     ' dest. horizotal wide
        6, &                      ' no. of dest. bits per byte
        0)                        ' shift left
  else
    tmpscr$ = let$ ( mnuscr$ )
  endif
  put #LCD2, #1, tmpscr$, 0, 0, GR_SIZE ' zeige Grafik an
END


'---------------------------------------------------------------------
' SUB m_input
' pruefe, ob Encoderachse gedrueckt
' wenn ja, dann warte bis losgelassen
' setze Flagge entsprechend
'---------------------------------------------------------------------
SUB m_input
  ll_iport_in P_ENC_KEY, bEncKey, M_ENC_KEY
  if bEncKey <> M_ENC_KEY then    ' wenn Encoderachse gedrueckt
    enc_click = YES
    bEncKey = M_ENC_KEY + 1       ' <> M_ENC_KEY ist gedrueckt
    while bEncKey <> M_ENC_KEY    ' warte bis Encoderdruck losgelassen
      ll_iport_in P_ENC_KEY, bEncKey, M_ENC_KEY
    endwhile
  else
    enc_click = NO
  endif
END


'---------------------------------------------------------------------
' init_menu
' initialisiert das Menue-System
'---------------------------------------------------------------------
SUB init_menu
  menu_txt$(MNU_MAIN, 0) = "set parameter 10"
  menu_txt$(MNU_MAIN, 1) = "set parameter 20"
  menu_txt$(MNU_MAIN, 2) = "set parameter 30"
  menu_txt$(MNU_MAIN, 3) = "default values  "
  menu_txt$(MNU_MAIN, 4) = ""

  menu_pos (MNU_MAIN, 0) = 3     ' x pos.
  menu_pos (MNU_MAIN, 1) = 5     ' y pos.

  menu_txt$(MNU_PARA10, 0) = "  50Hz"
  menu_txt$(MNU_PARA10, 1) = " 100Hz"
  menu_txt$(MNU_PARA10, 2) = " 200Hz"
  menu_txt$(MNU_PARA10, 3) = " 400Hz"
  menu_txt$(MNU_PARA10, 4) = " 800Hz"
  menu_txt$(MNU_PARA10, 5) = "1600Hz"
  menu_txt$(MNU_PARA10, 6) = ""
```

```
  menu_pos (MNU_PARA10, 1) = 5    ' y pos.

  menu_txt$(MNU_PARA20, 0) = "  0%"
  menu_txt$(MNU_PARA20, 1) = " 10%"
  menu_txt$(MNU_PARA20, 2) = " 20%"
  menu_txt$(MNU_PARA20, 3) = " 40%"
  menu_txt$(MNU_PARA20, 4) = " 60%"
  menu_txt$(MNU_PARA20, 5) = " 80%"
  menu_txt$(MNU_PARA20, 6) = "100%"
  menu_txt$(MNU_PARA20, 7) = ""

  menu_pos (MNU_PARA20, 0) = 20   ' x pos.
  menu_pos (MNU_PARA20, 1) = 6    ' y pos.

  menu_txt$(MNU_PARA30, 0) = "1.00V"
  menu_txt$(MNU_PARA30, 1) = "2.25V"
  menu_txt$(MNU_PARA30, 2) = "3.75V"
  menu_txt$(MNU_PARA30, 3) = "5.00V"
  menu_txt$(MNU_PARA30, 4) = ""

  menu_pos (MNU_PARA30, 0) = 20   ' x pos.
  menu_pos (MNU_PARA30, 1) = 7    ' y pos.

  menu_last (MNU_MAIN) = 3
  menu_last (MNU_PARA10) = 5
  menu_last (MNU_PARA20) = 6
  menu_last (MNU_PARA30) = 3

  lcd_fnt_y = 8                   ' LCD Zeichenhoehe in Pixel
  if LCD_MODE = LCD_MODE30 then
    lcd_fnt_x = 8                 ' LCD Zeichenbreite in Pixel
  else
    lcd_fnt_x = 6
  endif
END


'---------------------------------------------------------------------
' SUB clear_text_screen
' loescht den Textbildschirm
' 16 x 30 -> 480 Leerzeichen
' 16 x 40 -> 640 Leerzeichen
' max 240 in einem PRINT
'---------------------------------------------------------------------
SUB clear_text_screen
  BYTE j
  LONG fi

  for j = 1 to 3                  ' loesche Textbildschirm
    print #LCD2, lcd_txtclr$;
                                  ' Fuellstand Ausgabepuffer
    get #LCD2, #0, #UFCI_OBU_FILL, 4, fi
    while fi > 0                  ' warte bis leer
      get #LCD2, #0, #UFCI_OBU_FILL, 4, fi
    endwhile
  next
  if lcd_mode = LCD_MODE40 then
    print #LCD2, lcd_txtclr$;
  endif
  put #LCD2, "<2>"                ' home
```

```
'--------------------------------------------------------------------
' SUB ini_parameters
'--------------------------------------------------------------------
SUB ini_parameters
  para10 = 50                    ' initialisiere Parameter
  para20 = 0
  para30 = 0
END
```

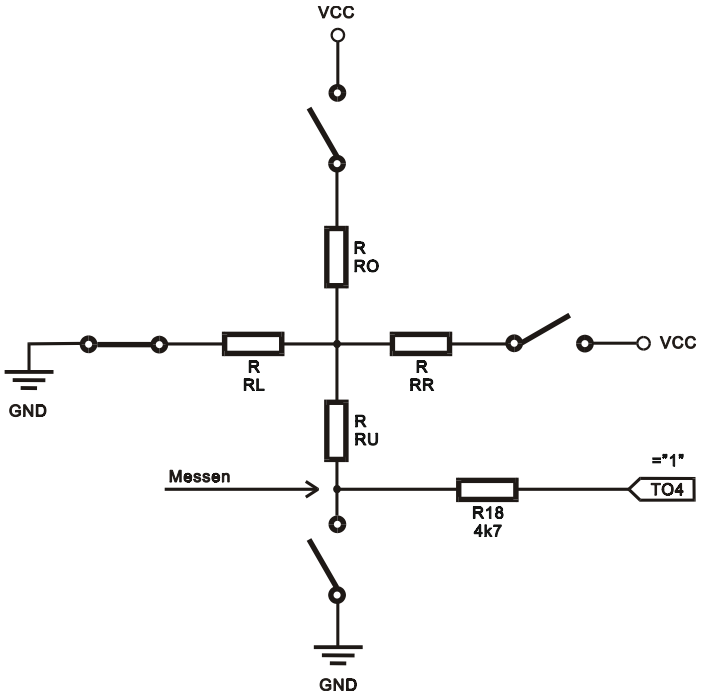
Touchpanel

Das analoge Touchpanel des BASIC-Tiger® Graphic-Toolkits besteht aus zwei übereinanderliegenden Widerstandsfolien, an die in horizontaler bzw. vertikaler Richtung eine Spannung über Transistoren zugeschaltet werden kann. Wird nun an einer Stelle auf das Touchpanel gedrückt, entsteht dort eine elektrische Verbindung und es entsteht ein Spannungsteiler in horizontaler und vertikaler Richtung. Die Druckposition wird ermittelt, indem nacheinander in horizontaler und in vertikaler Richtung eine Spannung angelegt wird und die Werte der Spannungsteilung mit einem analogen Eingang des BASIC-Tiger® gemessen werden.

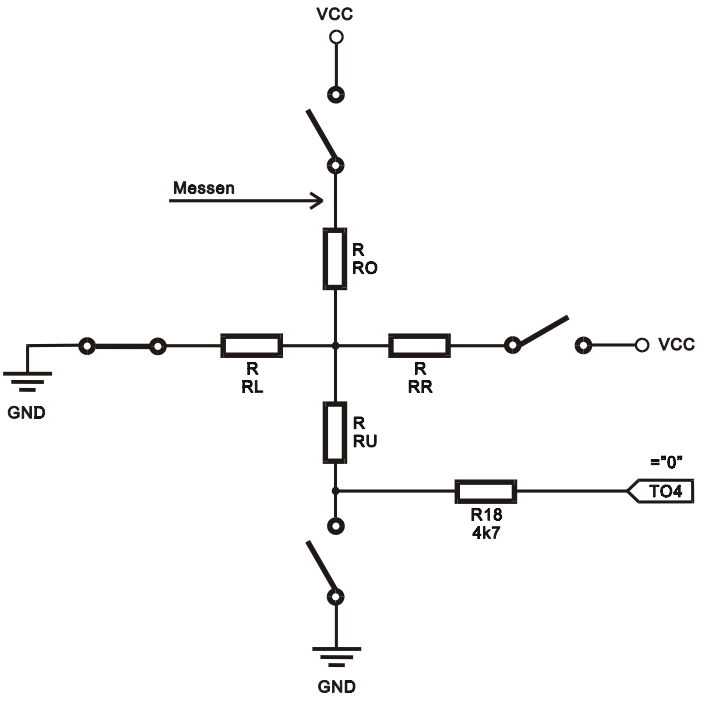


Die Erkennung eines Tastendruckes läuft in drei Schritten ab:

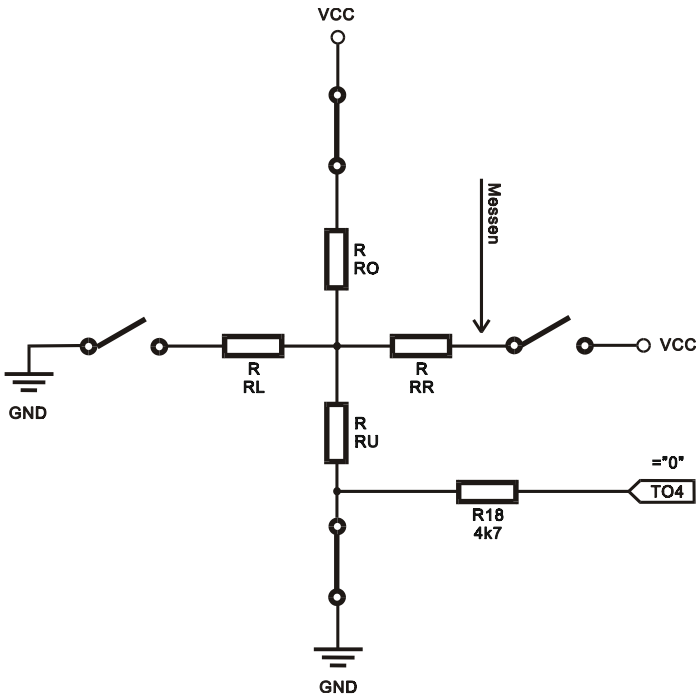
- Im ersten Schritt wird festgestellt ob das Touchpanel berührt wurde. Dazu wird, über den externen Widerstand R18 5V an das Touchpanel angelegt. Gleichzeitig werden die Schalter an RO, RR und RU geöffnet, während der Schalter an RL geschlossen wird. Wurde das Touchpanel nicht berührt, dann wird an RU 5V gemessen. Wenn jedoch das Touchpanel berührt wurde, dann fließt ein Strom über RU und RL und der Wert ist von 5V verschieden.



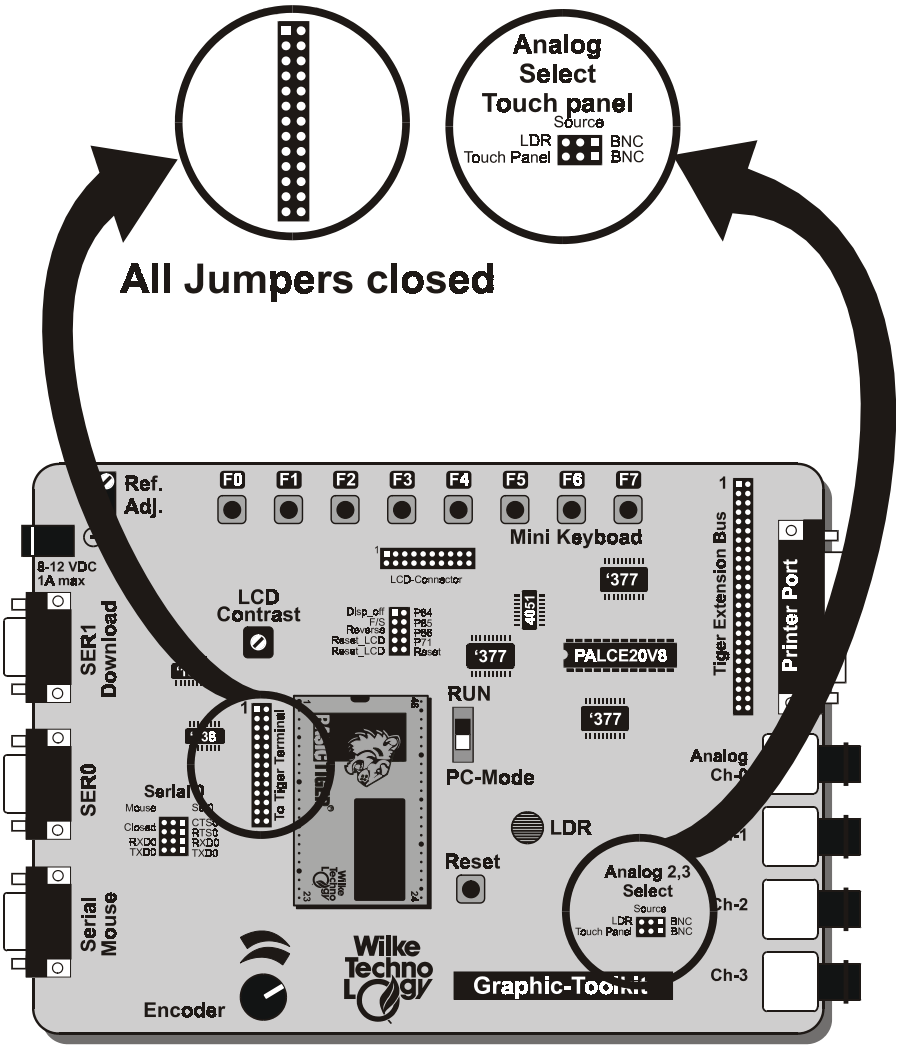
Im zweiten Schritt wird nun die X-Koordinate ermittelt. Dazu werden die Schalter an RO und RU geöffnet, während die Schalter an RL und RR geschlossen werden. Der Widerstand R18 ist mit 4.7kΩ mindestens 10 mal so groß wie die Widerstände des Touchpanels (300...500Ω) und fällt nicht störend ins Gewicht. Durch drücken auf das Touchpanel entsteht ein Spannungsteiler in X-Richtung. Der Spannungswert wird am Schalter von RO gemessen und ist proportional zur X-Position.



Im dritten und letzten Schritt wird die Y-Position ermittelt. Dazu werden die Schalter an RO und RU geschlossen, während die Schalter an RL und RR geöffnet werden. Durch messen an RR wird nun die Y-Position ermittelt.



Um nun nicht drei analoge Eingänge des BASIC-Tiger® zu verbrauchen, wurde auf dem BASIC-Tiger® Graphic-Toolkit eine Analog-Multiplexerschaltung integriert, die es ermöglicht, mit nur einen Analogeingang zu arbeiten.



Jumper für Touchpanel

Um das Touchpanel auf dem BASIC-Tiger® Graphic-Toolkit zu verwenden, müssen folgende Jumper gesetzt sein:

Die Analogwerte des Touchpanels werden über einen Analogport eingelesen. Die 3 Meßpunkte werden über einen Analogschalter nach Bedarf auf diesen Eingang geschaltet. Die Umschaltung erfolgt über einen erweiterten Ausgang, daher müssen die 13 Jumper gesteckt sein. Um Analogwerte einzulesen, stehen 2 Device-Treiber zur Verfügung:

- ANALOG1.TDD liest einen Analogwert von einem Kanal bei Bedarf, d.h. mit einer GET-Instruktion.
- ANALOG2.TDD liest Analogwerte von einem bis zu 4 Kanälen getriggert von TIMERA.TDD ein.

Erfordert die Anwendung nicht ANALOG2.TDD, dann ist das Touchpanel einfacher mit ANALOG1.TDD zu beherrschen.

Weiterhin gibt es zwei grundsätzlich Möglichkeiten, das Touchpanel einzusetzen:

- In den meisten Anwendungen will man das Panel wie eine Tastatur verwenden und die Tasten bzw. Berührstellen grafisch darstellen. Dann muß die Anwendung ein Unterprogramm haben, welche die Tastendrücke erkennt. Sinnvoll ist eine akustische oder optische Rückmeldung, die anzeigt, daß ein Tastendruck erkannt worden ist.
- Einige Anwendungen brauchen laufende Information darüber ob, und wenn ja, wo eine Berührung stattfindet. Dann muß die Anwendung eine Task haben, welche ständig Informationen über die Berührung liefert. In der Regel erfolgt eine optische Rückmeldung auf dem LCD.

Nach der Montage muß ein neuer Touchpanel-Typ eingemessen werden, um im Programm die Positionen auf dem Panel den Positionen auf dem LCD zuzuordnen.

Es folgen einige Beispielprogramme:

Programm	
TKB_TST1.TIG	Es werden direkt die analogen Meßwerte der Berührung angezeigt, und zwar 6 Werte nebeneinander, die reihum neu beschreiben werden. Sobald der Scanwert < 1023 ist, entstehen Werte für X und Y. Die Schwankungen der Meßwerte und ‚falsche' Werte aus dem Moment des Drückens bzw. Loslassens sind zu sehen. Mit diesem Testprogramm können die Randwerte des Touchpanels bestimmt werden.
TKB_DRAW.TIG	Während einer Berührung werden ständig neue Werte geliefert. Eine Mittelung in einem FIFO-Puffer reduziert Störungen. Die erkannte Position wird durch einen Cursor sichtbar gemacht, der in das Originalbild einkopiert wird. Dazu wird vor dem Einkopieren des Cursors eine Kopie des Orignalbildes erstellt.
TKB_KEY1.TIG	Wenn eine Anzahl Werte innerhalb einer bestimmten Zeit im gleichen Wertebereich lag, wird dies als Tastendruck gedeutet und ein Tastencode weitergegeben. Jede Berührung erzeugt nur einen Code. Um einen neuen Tastendruck zu erzeugen, muß zwischendurch losgelassen werden.
TKB_TST2.TIG	Wie TKB_TST1.TIG, jedoch unter Verwendung von ANALOG2.TDD. Es werden nur Werte für X und Y erzeugt. Der FIFO darf sich nicht füllen, auch bei Berührung des Panels nicht. Eventuell muß in den RUN-Modus geschaltete werden.
TKB_KEY2.TIG	Wie TKB_KEY1.TIG, jedoch unter Verwendung von ANALOG2.TDD. Die Anwendung muß die X- und Y-Werte zusammen mit den Analogmeßwerten der Anwendung liefern.

Touchpanel-Cursor

Im folgenden Beispiel werden die analogen Meßwerte, die bei der Berührung des Touchpanels entstehen, in einem FIFO-Puffer zwecks Mittelwertbildung gesammelt. Die Tiefe des Puffers ist per ‚#define' im Programm einstellbar. Im Unterschied zu TKB_KEY1.TIG werden hier ständig gemittelte Werte über die Berührposition über globale Variable an das Hauptprogramm geliefert.

Das Hauptprogramm blendet den Cursor als kleine 8x8-Grafik in den Original-LCD-Bildschirm ein. Dies geschieht in einer Kopie des Originals, um dieses nicht zu zerstören. Wenn gewünscht, kann diese Aufgabe auch in die Task ‚TPanel' verlegt werden.

Programmbeispiel:


```
'-----------------------------------------------------------------------
' Name: TKB_DRAW.TIG
' Das Hauptprogramm zeigt Beruehrungen auf dem Touchpanel an.
' Das Scan-Programm ist als eigene Task realisiert und kann in
' eigene Programme uebertragen werden.
' Integrierte Scanwerte werden staendig global uebergeben.
'-----------------------------------------------------------------------
' L71 mit LCD-Reset verbinden
'-----------------------------------------------------------------------
user_var_strict                 ' Vars deklarieren!
#include DEFINE_A.INC           ' allgemeine Definitionen
#include UFUNC3.INC             ' User-Function-Codes
#include LCD_4.INC              ' Definitionen fuer LCD Typ 4
#include GR_TK1.INC             ' Definitionen fuer Graphic Toolkit

LONG tp_x, tp_y                 ' global fuer Touchpanel X-, Y-Werte
STRING Screen$(3840)            ' orig. Pixeldaten
STRING Scr$(3840)               ' Kopie der Pixeldaten um Cursor
                                ' einzublenden

' Die folgenden 4 Werte sind von der Grösse
' und der Position des Touchpanel abhängig
#define TP_XMIN 106             ' Min-Max-Werte im Touchpanelbereich
#define TP_XMAX 830
#define TP_YMIN 174
#define TP_YMAX 840


'-----------------------------------------------------------------------
' Hauptprogramm
' Installiert Device-Treiber und initialisiert LCD
' Zeigt X-, Y-Scanwerte aus Task TPanel an
'-----------------------------------------------------------------------
TASK Main                       ' Beginn Task MAIN

  BYTE i
  LONG old_tp_x, old_tp_y
  LONG lcd_x, lcd_y             ' Cursor X-, Y-Werte auf LCD
  STRING GrCursor$

  call Init_LCDpins              ' init LCD
                                          ' LCD-4=240x128, 150 KB/s
```

```
  install_device #AD1, "ANALOG1.TDD" ' Analog-In installieren
                                     ' 8x8 Cursor Grafik
  GrCursor$ = "<10h><10h><10h><0ffh><10h><10h><10h><0>"
  put #LCD2, "<1Bh>c<20><0F0h>" ' text-Cursor aus
  print #LCD2, "<1Bh>A<10><10><0F0h>please touch"
  run_task TPanel, 20            ' aktiviere Touchpanel als Tastatur

  Screen$ = FILL$( "<00H>", 3840 ) ' hier: leerer Bildschirm

  tp_x = 0
  tp_y = 0
  old_tp_x = tp_x
  old_tp_y = tp_y
  for i = 0 to 0 step 0          ' Endlosschleife
    if (tp_x <> old_tp_x) or (tp_y <> old_tp_y) then
      print #LCD2, "<1bh>A<0><0><0F0h>Scanned:";tp_x;";";tp_y;"   ";
                                 ' Touch-Koord. in Bildschirmkoord.
      lcd_x = lin_approx (TP_XMIN, 0, TP_XMAX, 239, tp_x)
      lcd_y = (LINES-1) - (lin_approx (TP_YMIN, 0, TP_YMAX, 127, tp_y))
      print #LCD2, "<1bh>A<0><1><0F0h>    LCD:";lcd_x;";";lcd_y;"   ";
      scr$ = screen$               ' rette Original
      graphic_copy (Scr$, &        ' Original LCD-Inhalt
        GrCursor$, &               ' Cursorpixel
        COLUMNS, LINES, &          ' Groesse Zielbereich
        lcd_x-3, lcd_y-3, &        ' Ziel der Kopie (-3: Cursorzentrum)
        8, 8, &                    ' Groesse Quellbereich
        0, 0, &                    ' von Position
        7, 7, &                    ' Groesse der zu kopierenden Grafik
        0)                         ' Modus Copy
      put #LCD2, #1, Scr$, 0, 0, 3840 ' auf LCD ausgeben
      old_tp_x = tp_x
      old_tp_y = tp_y
    endif
  next
END


'---------------------------------------------------------------------
' Integriert Touchpanel-Scanwerte
'---------------------------------------------------------------------
#define TP_INTEG 32              ' Integrationstiefe (Wert=2^n)

TASK TPanel
  BYTE i
  WORD wR0, wX, wY               ' Scanwert, X, Y
  FIFO tpf_x (TP_INTEG) of WORD  ' X-Integration
  FIFO tpf_y (TP_INTEG) of WORD  ' Y-Integration
  start_fifo tpf_x, TP_INTEG, 0  ' init FIFOs
  start_fifo tpf_y, TP_INTEG, 0

  for i = 0 to 0 step 0          ' Endlosschleife
    wR0 = SCAN_IDLE + 1
    while wR0 >= SCAN_IDLE       ' warte auf Beruehrung
      out TOUCH, 255, TKB_S      ' Scan-Modus
      get #AD1, #2, 2, wR0       ' Scanwert holen
    endwhile

    out TOUCH, 255, TKB_X        ' Spannung in X-Richtung
    get #AD1, #2, 2, wX          '
    out TOUCH, 255, TKB_Y        ' Spannung in X-Richtung
```

```
    integral_fifo tpf_x, -TP_INTEG, wX
    integral_fifo tpf_y, -TP_INTEG, wY
    wX = limit ( wX, TP_XMIN, TP_XMAX )' beschraenke auf
    wY = limit ( wY, TP_YMIN, TP_YMAX )' gueltigen Wertebereich
    tp_x = wX
    tp_y = wY
  next
END
```

Touchpanel als Tastatur

Im Beispielprogramm TKB_KEY1.TIG zeigt die Main-Task ein Tastaturmuster auf dem LCD an. In einer echten Anwendung ist an dieser Stelle die Bildschirm-Maske der Anwendung. Es müssen übrigens nicht alle Stellen als Tasten ausgewertet werden. Da dieses Programm nur ein Test ist, werden von der Main-Task die übergebenen Tastencodes einfach in Textform angezeigt. Ein Code ist als Kennzeichen für ‚keine Taste gedrückt' reserviert (KEY_INVALID).

In der Task TPanel wird zunächst das Touchpanel abgetastet, um eine Berührung zu erkennen. Dieser Abtastmodus spart etwas Strom gegenüber der direkten Abtastung von X- oder Y-Werten zur Erkennung einer Berührung. Wird eine Berührung erkannt, dann werden die X-und Y-Werte entsprechend der Matrix die Scan-Codes berechnet. Ein gültiger Tastendruck muß mindestens eine gewisse Anzahl gleicher Codes liefern. Die Anzahl ist per ‚#define' im Programm einstellbar. Je höher diese Anzahl, desto länger muß die Berührung gehalten werden, bis daraus ein gültiger Tastendruck wird, desto höher wird aber auch die Sicherheit, keine falschen Codes zu erkennen. Die Task, die für die Erkennung der Tastendrücke zuständig ist, gibt die Tasten-Codes über globale Variable an das Hauptprogramm weiter. Abschließend wird gewartet, bis die Berührung aufgehoben wird, um eine unkontrollierte Auto-Repeat-Funktion zu verhindern.

Programmbeispiel:

```
'---------------------------------------------------------------------
' Name: TKB_KEY1.TIG
' mit ANALOG1.TDD
' Das Hauptprogramm zeigt Beruehrungen auf dem Touchpanel als
' Tastendruecke an.
' Das Scan-Programm ist als eigene Task realisiert und kann in
' eigene Programme uebertragen werden.
' Fuer einen gueltigen Tastendruck muss eine Anzahl Scanwerte
' in einer festgelegten Zeit uebereinstimmen.
' Zu Testzwecken druckt das Hauptprogramm die Berechnung
' und den Tastecode in den Bildschirm.
'---------------------------------------------------------------------
' L71 mit LCD-Reset verbinden
'---------------------------------------------------------------------
user_var_strict                  ' erzwinge Var-Deklaration
#include DEFINE_A.INC             ' allgemeine Definitionen
#include UFUNC3.INC               ' User-Function-Codes
#include LCD_4.INC                ' Definitionen fuer LCD Typ 4
#include GR_TK1.INC               ' Definitionen fuer Graphic Toolkit

' Die folgenden 4 Werte sind von der Grösse
' und der Position des Touchpanel abhängig
#define TP_XMIN 106              ' Min-Max-Werte im Touchpanelbereich
#define TP_XMAX 830
#define TP_YMIN 174
#define TP_YMAX 840

#define KEY_INVALID 255          ' code für 'keine Taste gedrueckt'
BYTE key                         ' global fuer Taste
STRING Screen$(GR_SIZE)
DATALABEL keypatt1

'---------------------------------------------------------------------
' Hauptprogramm
' Installiert Device-Treiber und initialisiert LCD
' Zeigt erkannte Tastencodes aus Task TPanel an
'---------------------------------------------------------------------
TASK Main                        ' Beginn Task MAIN
keypatt1::
  data filter "KEY_10X5.BMP", "GRAPHFLT", 0 ' Tastaturmuster
  BYTE ever

  call Init_LCDpins               ' init LCD
                                           ' LCD-4=240x128, 150 KB/s
  install_device #LCD2,"LCD-6963.TDD",0,0,0EEH,4,150,11H
  install_device #AD1, "ANALOG1.TDD" ' Analog-In installieren

  peek_flash keypatt1, Screen$, 3840 ' lese Grafik aus Flash
  put #LCD2, #1, Screen$,0,0,3840 ' zeige Tastaturmuster an
  put #LCD2, "<1Bh>c<20><0F0h>" ' Text-Cursor aus
  print #LCD2, "<1Bh>A<10><10><0F0h>please touch"
  key = 255                      ' ungueltige Taste
  run_task TPanel, 20            ' aktiviere Touchpanel als Tastatur

  dir_pin 4, 2, 0                ' L42 mit Beeper verbunden
  for ever = 0 to 0 step 0
    while key = KEY_INVALID      ' ungueltig, warte auf Beruehrung
    endwhile
```

```
    ll_iport_out 4, 255
                                ' zeige Code der gueltigen Tasten an
    print #LCD2, "<1bh>A<0><4><0F0h>-> global key:";key;" ";
    key = KEY_INVALID           ' Benutze Taste nur einmal
  next
END


'--------------------------------------------------------------------
' TASK TPanel
' Ordnet Tastendrücken auf dem Touchpanel Codes zu.
' Das LCD mit 240x128 Pixeln ist unterteilt in Tastenbereiche:
' TP_KEY_ROWS x TP_KEY_COLUMNS (Reihen x Spalten)
' Eine Beruehrung wird als Tastendruck erkannt, wenn
'   eine Anzahl Loops (TP_RECOG_LOOPS) X- und Y-Werte aus dem
'   gleichen Bereich entstanden sind.
' Zusaetzlich gibt es ein Time-Out (TP_RECOG_TIMEOUT), nach dem
'   zunaechst ein Loslassen abgewartet wird.
'--------------------------------------------------------------------
#define TP_KEY_COLUMNS 10       ' Anz. Tasten in X-Richtung (Spalten)
#define TP_KEY_ROWS 5           ' Anz. Tasten in Y-Richtung (Reihen)
#define TP_KEY_XUNIT ((TP_XMAX - TP_XMIN) / TP_KEY_COLUMNS)
#define TP_KEY_YUNIT ((TP_YMAX - TP_YMIN) / TP_KEY_ROWS)
#define TP_RECOG_LOOPS 10       ' Schleifen bis eine Taste erkannt ist

TASK TPanel
  BYTE i, keytmp
  WORD wScan
  WORD wX, wY, old_wX, old_wY
  WORD tp_recog                 ' 'erkannt'-Flag
  LONG t                        ' Zeit

  key = KEY_INVALID             ' fange an mit ungueltiger Taste
  for i = 0 to 0 step 0         ' Endlosschleife
    old_wX = 255
    old_wY = 255
    wScan = SCAN_IDLE + 1
    while wScan >= SCAN_IDLE    ' warte auf Beruehrung
      out TOUCH, 255, TKB_S     ' Scan-Modus
      get #AD1, #2, 2, wScan    ' Scanwert holen
    endwhile
    tp_recog = 0
    while tp_recog < TP_RECOG_LOOPS ' Erkenn-Schleife
      out TOUCH, 255, TKB_X     ' Spannung in X-Richtung
      get #AD1, #2, 2, wX       '
      out TOUCH, 255, TKB_Y     ' Spannung in Y-Richtung
      get #AD1, #2, 2, wY

      if (wX > TP_XMIN) &       ' wenn im gueltigen Wertebereich
     and (wX < TP_XMAX) &
     and (wY > TP_YMIN) &
     and (wY < TP_YMAX) then    ' dann in Code umrechnen
        wX = (wX - TP_XMIN) / TP_KEY_XUNIT
        wY = (wY - TP_YMIN) / TP_KEY_YUNIT
        if (wX = old_wX) and (wY = old_wY) then
          tp_recog = tp_recog + 1 ' zaehle Vorkommen gleicher Werte
        endif
        old_wX = wX
        old_wY = wY
' wenn TP_KEY_COLUMNS > TP_KEY_ROWS
```

```
' wenn TP_KEY_ROWS > TP_KEY_COLUMS
'        keytmp = wX * TP_KEY_ROWS + wY
        print #LCD2, "<1bh>A<0><0><0F0h>code calculation:"
        print #LCD2, "<1bh>A<0><1><0F0h>";keytmp;"=";wX;"*";TP_KEY_COLUM
      endif
    endwhile ' Erkenn-Schleife
    key = keytmp                  ' key erkannt, Code zuweisen

    tp_recog = 0                  ' nun warte auf loslassen
    while (wScan < SCAN_IDLE)
      out TOUCH, 255, TKB_S       ' Scan-Modus
      get #AD1, #2, 2, wScan      ' Scanwert holen
    endwhile
  next
END
```

Touchpanel als Tastatur mit ANALOG2.TDD

Im vorigen Beispiel ist die Tastatur dadurch realisiert, daß bei Bedarf der Analogport mit den anliegenden Meßwerten des Touchpanels ausgelesen wird. Die freien Analogeingänge können zur Aufnahme von anderen Meßwerten verwendet werden. Wird jedoch eine Analogmessung mit dem Device-Treiber ANALOG2.TDD (zusammen mit TIMERA.TDD) benötigt, dann muß dieser ANALOG2.TDD sowohl die benötigten Meßwerte der Anwendung als auch die Meßwerte des Touchpanels liefern.

In diesem Fall sind wiederum 2 Konstellationen denkbar:

- Der ANALOG2 wird nicht gleichzeitig für die Meßwertaufnahme und das Touchpanel gebraucht. In diesem Fall kann die Einstellung des ANALOG2 passend für beide Meßaufgaben konfiguriert werden.
- Der ANALOG2 muß gleichzeitig Meßwerte aufnehmen und das Touchpanel ablesen. Da die Messung im Hintergrund immer für alle benötigten Kanäle erfolgt und auch in der Kanalwahl einige Einschränkungen bestehen, erhöht sich der Programmieraufwand. Das folgende Beispiel zeigt eine mögliche Konstellation und klärt gleichzeitig über die Schwierigkeiten auf.

Die Main-Task des Beispiels TKB_KEY2.TIG ist stellvertretend für irgendeine Anwendung, die ANALOG2 verwendet, um etwas zu messen und nebenher die Meßwerte des Touchpanels liefern soll. Zur Vereinfachung werden hier nur die Meßwerte des Touchpanels erzeugt. Auf dem BASIC-Tiger® Graphic-Toolkit kommen sowohl die X- als auch die Y-Werte des Touchpanels über einen Analogeingang, der entsprechend umgeschaltet wird. Das Problem für die Task TPanel ist nun, daß die Task nach dem Umschalten auf Messung von X-Werten bzw. Y-Werten nicht selbst die passenden Meßwerte abholen kann. Vielmehr kann sie nur die X- oder Y-Messung einschalten und ist dann darauf angewiesen, über eine globale Variable an die analogen Meßwerte des richtigen Kanals heranzukommen, denn die eigentliche Messung wird von der Main-Task gesteuert. In diesem Fall wird in einen FIFO gemessen. Damit nicht nach einer Umschaltung auf Y-Messung noch alte X-Werte angeliefert werden, darf der FIFO-Puffer sich nie füllen. In einem gefüllten FIFO liesse sich beim Auslesen nicht mehr feststellen, ob zum Zeitpunkt der Messung X- oder Y-Werte gemessen wurden.Eiin gefüllter FIFO-Puffer würde außerdem eine Verzögerung bedeuten, da zunächst ältere Werte gelesen werden.

Wenn die Voraussetzung erfüllt ist, daß der FIFO-Puffer sich nicht füllt, dann kann die Task TPanel kurz nach der Umschaltung z.B. von X- nach Y-Messung erwarten, daß die richtigen Werte geliefert werden. Ab diesem Moment funktioniert das Erkennen von Tastendrücken genauso wie bei TKB_KEY1.TIG.

Um in einer Anwendung mit ANALOG2.TDD ein Touchpanel einsetzen zu können, müssen also folgende Voraussetzungen erfüllt sein:

- die Messung und die Bedienung des Touchpanels fallen nie zusammen
- oder die Messung läuft langsam genug, so daß der FIFO-Puffer sich nicht füllt.

Sind die Bedingungen nicht erfüllt, bleibt die Erwägung, einen zweiten Prozessor einzusetzen oder bei großen Stückzahlen einen speziellen Device-Treiber entwickeln zu lassen.

Programmbeispiel:

```
'---------------------------------------------------------------------
' Name: TKB_KEY2.TIG
' mit ANALOG2.TDD
' Das Hauptprogramm zeigt Beruehrungen auf dem Touchpanel als
' Tastendruecke an.
' Das Scan-Programm ist als eigene Task realisiert und kann in
' eigene Programme uebertragen werden.
' Fuer einen gueltigen Tastendruck muss eine Anzahl Scanwerte
' in einer festgelegten Zeit uebereinstimmen.
'---------------------------------------------------------------------
' L71 mit LCD-Reset verbinden
'---------------------------------------------------------------------
user_var_strict                  ' erzwinge Var-Deklaration
#include DEFINE_A.INC             ' allgemeine Definitionen
#include UFUNC3.INC               ' User-Function-Codes
#include LCD_6.INC                ' Definitionen fuer LCD Typ 6
#include GR_TK1.INC               ' defines for Graphic Toolkit

' Die folgenden 4 Werte sind von der Grösse
' und der Position des Touchpanel abhängig
#define TP_XMIN 106              ' Min-Max-Werte im Touchpanelbereich
#define TP_XMAX 830
#define TP_YMIN 174
#define TP_YMAX 840

#define KEY_INVALID 255          ' code für 'keine Taste gedrueckt'
BYTE key                         ' global fuer Taste
STRING Screen$(GR8_SIZE)
STRING Scr$(GR_SIZE)
DATALABEL keypatt1

#define TA_RANGE_INIT 3          ' TIMERA range             (156250Hz)
#define TA_DIV_INIT 217          ' TIMERA range           (/217=720Hz)
#define ANA_PSCAL 6              ' analog prescaler         (/6=120Hz)

WORD wTouch, wX, wY, l1, l2

'---------------------------------------------------------------------
' Hauptprogramm
' Installiert Device-Treiber und initialisiert LCD
' Zeigt erkannte Tastencodes aus Task TPanel an
'---------------------------------------------------------------------
TASK Main                        ' Beginn Task MAIN
keypatt1::
  data filter "KEY_8X4.BMP", "GRAPHFLT", 0 ' Tastaturmuster
  BYTE ever
  FIFO ana ( 1024 ) OF WORD

  call Init_LCDpins               ' init LCD
                                          ' LCD-6=240x128, 150 KB/s
  install_device #LCD2,"LCD-6963.TDD",0,0,0EEH,LCD_TYPE,150,11H
  install_device #TA, "TIMERA.TDD", TA_RANGE_INIT, TA_DIV_INIT
  install_device #AD1, "ANALOG2.TDD"

  peek_flash keypatt1, Screen$, GR8_SIZE ' lese Grafik aus Flash
  l1 = len ( Screen$ )
  Scr$ = graphic_exp$ ( &         ' Pixel auf Bytes verteilen
    screen$, &                    ' Quellstring
```

```
    GR6_SIZE, &                    ' Ziellaenge in Pixel
    LCD_BYT_COL6X8, &              ' Zielbreite in Byte
    6, &                           ' Pixel/Byte im Zielstring
    0)                             ' Shifts nach links im Zielstring
  l2 = len ( Scr$ )
  put #LCD2, #1, Scr$,0,0,GR_SIZE ' zeige Tastaturmuster an
  put #LCD2, CURSOR_OFF            ' Text-Cursor aus
  print #LCD2, "<1Bh>A<10><10><0F0h>please touch"
  key = 255                        ' ungueltige Taste
  wX = 0
  wY = 0
  run_task TPanel                  ' aktiviere Touchpanel als Tastatur
  put #AD1, #0, #UFCO_AD2_RESO, 10' Aufloesung 10 Bit
  put #AD1, #0, #UFCO_AD2_SCAN, 1 ' Anz. Kanaele An2(touchkb)
  put #AD1, #0, #UFCO_AD2_CHAN, 2 ' Kanal An2(touchkb)
  put #AD1, #0, #UFCO_AD2_PSCAL, ANA_PSCAL
  put #AD1, ana                    ' starte Analog-Kanal 0 in FIFO

  ' die folgende Zeile spaeter auskommentieren
  run_task Disp                    ' FIFO darf nicht voll werden

  dir_pin 4, 2, 0
  for ever = 0 to 0 step 0         ' Endlosschleife
    while key = KEY_INVALID        ' warte auf Beruehrung
      if len_fifo ( ana ) > 1 then
        get_fifo ana, wTouch       ' Scanwert holen
      endif
    endwhile
    ll_iport_out 4, 0
    ll_iport_out 4, 255
    print #LCD2, "<1bh>A<0><8><0F0h>-> global key:";key;" ";
    key = KEY_INVALID              ' Benutze Taste nur einmal
  next ' ever
END


'---------------------------------------------------------------------
' TASK TPanel
' Ordnet Tastendrücken auf dem Touchpanel Codes zu.
' Das LCD mit 240x128 Pixeln ist unterteilt in Tastenbereiche:
' TP_KEY_ROWS x TP_KEY_COLUMNS (Reihen x Spalten)
' Eine Beruehrung wird als Tastendruck erkannt, wenn
'   eine Anzahl Loops (TP_RECOG_LOOPS) X- und Y-Werte aus dem
'   gleichen Bereich entstanden sind.
'   Vor der naechsten Taste wird zunaechst ein Loslassen abgewartet.
'---------------------------------------------------------------------
#define TP_KEY_COLUMNS 8          ' Anz. Tasten in X-Richtung (Spalten)
#define TP_KEY_ROWS 4             ' Anz. Tasten in Y-Richtung (Reihen)
#define TP_KEY_XUNIT ((TP_XMAX - TP_XMIN) / TP_KEY_COLUMNS)
#define TP_KEY_YUNIT ((TP_YMAX - TP_YMIN) / TP_KEY_ROWS)
#define TP_RECOG_LOOPS 3          ' Schleifen bis eine Taste erkannt ist

TASK TPanel
  BYTE ever, keytmp
  WORD wScan
  WORD wX, wY, old_wX, old_wY
  WORD tp_recog                    ' 'erkannt'-Flag
  LONG t, tt, ttt                  ' Zeit

  key = KEY_INVALID                ' fange an mit ungueltiger Taste
```

```
    old_wX = 0
    old_wY = 0
    tp_recog = 0                    ' Taste nicht erkannt
    while tp_recog < TP_RECOG_LOOPS ' Erkenn-Schleife
      out TOUCH, 255, TKB_X         ' Spannung in X-Richtung
      ttt = 0
      tt = ticks()
        while ttt < 30
        ttt = diff_ticks ( tt )
      endwhile
      wX = wTouch

      out TOUCH, 255, TKB_Y         ' Spannung in Y-Richtung
      ttt = 0
      tt = ticks()
        while ttt < 30
        ttt = diff_ticks ( tt )
      endwhile
      wY = wTouch

      if (wX > TP_XMIN) and &       ' wenn im gueltigen Wertebereich
         (wX < TP_XMAX) and &
         (wY > TP_YMIN) and &
         (wY < TP_YMAX) then        ' dann in Code umrechnen
        wX = (wX - TP_XMIN) / TP_KEY_XUNIT
        wY = (wY - TP_YMIN) / TP_KEY_YUNIT
        if (wX = old_wX) and (wY = old_wY) then
          tp_recog = tp_recog + 1 ' zaehle Vorkommen gleicher Werte
        endif
        old_wX = wX
        old_wY = wY
' wenn TP_KEY_COLUMNS > TP_KEY_ROWS
        keytmp = wX * TP_KEY_COLUMNS + wY
' wenn TP_KEY_ROWS > TP_KEY_COLUMS
'        keytmp = wX * TP_KEY_ROWS + wY

' 3 Kontroll-Ausgaben
        print #LCD2, "<1bh>A<0><0><0F0h>code calculation:"
        print #LCD2, "<1bh>A<0><1><0F0h>";keytmp;"=";wX;"*";TP_KEY_COLUM
        print #LCD2, "<1bh>A<0><2><0F0h>";tp_recog;"  ";
      endif
    endwhile ' Erkenn-Schleife
    key = keytmp                    ' key erkannt, Code zuweisen

    tp_recog = 0                    ' nun warte auf loslassen
    out TOUCH, 255, TKB_S           ' Scan-Modus
    while wTouch < SCAN_IDLE
' noch eine Kontroll-Ausgabe
      print #LCD2, "<1bh>A<0><4><0F0h>";"wTouch=";wTouch;"   ";
    endwhile

  next
END


'--------------------------------------------------------------------
' Fuer Testphase: zeigt FIFO-Fuellstand
'--------------------------------------------------------------------
Task Disp
  BYTE i
```

```
  for i = 0 to 0 step 0           ' Endlosschleife
    print #LCD2, "<1Bh>A<0><5><0F0h>len of fifo:";len_fifo ( ana );"
    wait_duration 200             ' gib der Grafik eine Chance
  next
END
```

Touchpanel als komplette Tastatur

In diesem Beispiel ist die Touchpanel-Tastatur fast komplett: es gibt alle Buchstaben und Ziffern sowie Leertaste, Eingabe (Return), Löschtaste (del) und sogar eine Shift-Taste, mit der zwischen Groß- und Kleinschreibung umgeschaltet werden kann.

Auf dem kleinen LCD mit 240 x 128 Pixeln ist es wahrscheinlich nur feineren Fingern möglich, die Tasten sofort richtig zu treffen. Eine noch feinere Aufteilung mit guter Trefferrate dürfte kaum möglich sein.

Das Programm besitzt eine Code-Conversionstabelle, um die Scan-Codes in die richtigen Zeichen umzusetzen. Die Funktion CONVERT$ macht das in einer Zeile. Die Tastatur selbst ist kein eingebundenes Bitmap, sondern sie wird per Programm erzeugt. Dazu werden die Buchstaben als Beschriftung der Tasten auf den Textbildschirm geschrieben und die ‚Tasten' werden in dem Grafikbereich als schwarze Vierecke mit GRAPHIC_COPY erzeugt.

Die Task Main tut nichts weiter als das Unterprogramm zur Eingabe ‚get_string' aufzurufen und nach erfolgter Eingabe (Abschluß mit RETURN) die Eingabezeile zu löschen. ‚Get_string' ermöglicht das Löschen eingegebener Zeichen und beachtet den Abschluß der Einagbe. ‚Get_key' erhält den Scan-Code von der Task ‚TPanel' und konvertiert je nach Groß- oder Kleinschreibung in die Zeichen-Codes. ‚Get_key' beachtet dabei die SHIFT-Funktion, die ein Ändern der Tastaturbeschriftung bewirkt.

Programmbeispiel:

```
'---------------------------------------------------------------------
' Name: TKB_KEY3.TIG
' mit ANALOG1.TDD
' Das Hauptprogramm zeigt eine komplette Touchpanel-Tastatur
'---------------------------------------------------------------------
' L71 mit LCD-Reset verbinden
'---------------------------------------------------------------------
user_var_strict                  ' erzwinge Var-Deklaration
#include DEFINE_A.INC             ' allgemeine Definitionen
#include UFUNC3.INC               ' User-Function-Codes
#include LCD_4.INC                ' Definitionen fuer LCD Typ 4
#include GR_TK1.INC               ' Definitionen fuer Graphic Toolkit

' Die folgenden 4 Werte sind von der Grösse
' und der Position des Touchpanel abhängig
#define TP_XMIN 106              ' Min-Max-Werte im Touchpanelbereich
#define TP_XMAX 850
#define TP_YMIN 174
#define TP_YMAX 840

STRING screen$(GR8_SIZE)         ' 240x480
STRING mnuscr$(GR8_SIZE)
STRING tmpscr$(GR_SIZE)
STRING black$(GR_SIZE)

#define KEY_INVALID 255          ' code für 'keine Taste gedrueckt'
BYTE key                         ' global fuer Taste
BYTE shifted
STRING inp$(40)
LONG cx, cy                      ' Cursor-Startposition
LONG x                           ' Cursor X-Pos. relativ

#define _CR_ "<0dh>"             ' einige Zeichen fuer get_string
#define _BS_ "<08h>"

STRING kb_lower$(240)            ' Tastaturbeschriftung
STRING kb_upper$(240)
STRING conv_lower$(128)          ' Scan-Code-Konversionstabellen
STRING conv_upper$(128)

'---------------------------------------------------------------------
' Hauptprogramm
' Installiert Device-Treiber und initialisiert LCD
'---------------------------------------------------------------------
TASK Main                        ' Beginn Task MAIN
  BYTE ever
  STRING k$(1)

  call Init_LCDpins              ' init LCD
  install_device #LCD2,"LCD-6963.TDD",0,0,0EEH,LCD_TYPE,150,11H
  install_device #AD1, "ANALOG1.TDD" ' Analog-In installieren

  dir_pin 4, 2, 0                ' L42 mit Beeper verbunden

  call init_vars                 ' initialisiere einige Variable
  put #LCD2, CURSOR_OFF          ' Text-Cursor aus
  put #LCD2, MODE_XOR            ' LCD-Modus
  put #LCD2, TEXT_ON
```

```
  key = 255                         ' ungueltige Taste
  shifted = 255                     ' no
  call gen_kb_text                  ' Tastaturbeschriftung
  call gen_kb_graphic               ' Tastaturgrafik
  run_task TPanel, 20               ' aktiviere Touchpanel als Tastatur

  for ever = 0 to 0 step 0          ' Endlosschleife
    call get_string ( &
      0, 0, &                       ' Feldposition
      40 )                          ' Feldlaenge
    wait_duration 2000              ' nach 2 Sek.: loesche erste Zeile
    print #LCD2, &
    "<1bh>A<0><0><0f0h>                                          ";
  next                              ' erneute Eingabe
END


'---------------------------------------------------------------------
' SUB get_string
' liefert Eingabestring in globaler Variablen inp$ zurueck
' liest den String von 'i_devno' (hier Unterprogramm 'get_key')
' und gibt Echo auf Geraet mit der Nummer 'o_devno' aus.
' Bei RETURN wird das Unterprogram beendet
'---------------------------------------------------------------------
SUB get_string ( &
  LONG   cxx, cyy, &                ' Feldposition
         slen )                     ' Feldlaenge

  BYTE i, i_devno, o_devno
  STRING c$(1)

'  i_devno = INPUT_DEVICE
  o_devno = LCD2
  cx = cxx
  cy = cyy

  put #o_devno, CURSOR_ON8          ' Eingabecursor

  inp$ = fill$ ( " ", slen )
  print #LCD2, "<1bh>A"; chr$(cx) ; chr$(cy) ;"<0f0h>";
  x = 0
  inp$ = ""

next_char1:
  call get_key ( c$ )
  if c$ <> "" then
    switch c$
      case _CR_:                    ' bei RET
        goto end_get_string         ' RET beendet Eingabe
                                    ' ---------------------------------
      case _BS_:                    ' loesche ein Zeichen
        if x > 0 then               ' nur wenn eins da ist
          x = x - 1
          inp$ = left$ ( inp$, len(inp$)-1 )
          print #o_devno, "<1bh>A"; chr$(cx+x) ; chr$(cy) ;"<0f0h>";" ";
          print #o_devno, "<1bh>A"; chr$(cx+x) ; chr$(cy) ;"<0f0h>";
        endif
                                    ' ---------------------------------
      default:
        if x < slen then            ' solange nicht maximale Laenge
```

```
          inp$ = inp$ + c$        ' sammle es ein
          print #o_devno, c$;
        endif
    endswitch
  endif
  goto next_char1

end_get_string:
  print #o_devno, CURSOR_OFF
END


'---------------------------------------------------------------------
' SUB get_key
'---------------------------------------------------------------------
SUB get_key ( var k$ )
  STRING tmp$(1)

new_key:
  while key = KEY_INVALID         ' ungueltig, warte auf Beruehrung
  endwhile
  tmp$ = chr$ ( key )
  if shifted = 0 then
    tmp$ = convert$ ( tmp$, conv_upper$ ) ' Code der Taste
  else
    tmp$ = convert$ ( tmp$, conv_lower$ ) ' Code der Taste
  endif
  if tmp$ = "<00>" then
    goto new_key
    key = KEY_INVALID             ' Benutze Taste nur einmal
  endif
  if tmp$ = "<0ffh>" then
    shifted = bitnot ( shifted )
    call gen_kb_text
    ll_iport_out 4, 0             ' erzeuge akustische Rueckmeldung
    ll_iport_out 4, 255
    key = KEY_INVALID             ' Benutze Taste nur einmal
    print #LCD2, "<1bh>A"; chr$(cx+x) ; chr$(cy) ;"<0f0h>";
    goto new_key
  endif
  k$ = tmp$
  ll_iport_out 4, 0               ' erzeuge akustische Rueckmeldung
  ll_iport_out 4, 255
  key = KEY_INVALID               ' Benutze Taste nur einmal
END


'---------------------------------------------------------------------
' TASK TPanel
' Ordnet Tastendrücken auf dem Touchpanel Codes zu.
' Das LCD mit 240x128 Pixeln ist unterteilt in Tastenbereiche:
' TP_KEY_ROWS x TP_KEY_COLUMNS (Reihen x Spalten)
' Eine Beruehrung wird als Tastendruck erkannt, wenn
'   eine Anzahl Loops (TP_RECOG_LOOPS) X- und Y-Werte aus dem
'   gleichen Bereich entstanden sind.
' Zusaetzlich gibt es ein Time-Out (TP_RECOG_TIMEOUT), nach dem
'   zunaechst ein Loslassen abgewartet wird.
'---------------------------------------------------------------------
#define TP_KEY_COLUMNS 10         ' Anz. Tasten in X-Richtung (Spalten)
#define TP_KEY_ROWS 8             ' Anz. Tasten in Y-Richtung (Reihen)
```

```
#define TP_KEY_YUNIT ((TP_YMAX - TP_YMIN) / TP_KEY_ROWS)
#define TP_RECOG_LOOPS 10        ' Schleifen bis eine Taste erkannt ist

TASK TPanel
  BYTE ever, keytmp
  WORD wScan
  WORD wX, wY, old_wX, old_wY
  WORD tp_recog                  ' 'erkannt'-Flag
  LONG t                         ' Zeit

  key = KEY_INVALID              ' fange an mit ungueltiger Taste
  for ever = 0 to 0 step 0       ' Endlosschleife
    old_wX = 255
    old_wY = 255
    wScan = SCAN_IDLE + 1
    while wScan >= SCAN_IDLE     ' warte auf Beruehrung
      out TOUCH, 255, TKB_S      ' Scan-Modus
      get #AD1, #2, 2, wScan     ' Scanwert holen
    endwhile
    tp_recog = 0
    while tp_recog < TP_RECOG_LOOPS ' Erkenn-Schleife
      out TOUCH, 255, TKB_X      ' Spannung in X-Richtung
      get #AD1, #2, 2, wX        '
      out TOUCH, 255, TKB_Y      ' Spannung in Y-Richtung
      get #AD1, #2, 2, wY

      if (wX > TP_XMIN) &        ' wenn im gueltigen Wertebereich
     and (wX < TP_XMAX) &
     and (wY > TP_YMIN) &
     and (wY < TP_YMAX) then     ' dann in Code umrechnen
        wX = (wX - TP_XMIN) / TP_KEY_XUNIT
        wY = (wY - TP_YMIN) / TP_KEY_YUNIT
        if (wX = old_wX) and (wY = old_wY) then
          tp_recog = tp_recog + 1 ' zaehle Vorkommen gleicher Werte
        endif
        old_wX = wX
        old_wY = wY
' wenn TP_KEY_COLUMNS > TP_KEY_ROWS
        keytmp = wX * TP_KEY_COLUMNS + wY
' wenn TP_KEY_ROWS > TP_KEY_COLUMS
'        keytmp = wX * TP_KEY_ROWS + wY
'        print #LCD2, "<1bh>A<0><0><0F0h>code calculation:"
'        print #LCD2, "<1bh>A<0><1><0F0h>";keytmp;"=";wX;"*";TP_KEY_COLU
      endif
    endwhile ' Erkenn-Schleife
    key = keytmp                  ' key erkannt, Code zuweisen

    tp_recog = 0                  ' nun warte auf loslassen
    while (wScan < SCAN_IDLE)
      out TOUCH, 255, TKB_S       ' Scan-Modus
      get #AD1, #2, 2, wScan      ' Scanwert holen
    endwhile
  next
END


'-------------------------------------------------------------------------
' SUB gen_kb_text
' druckt den text der Tastatur
'-------------------------------------------------------------------------
```

```
  BYTE i

  if shifted = 0 then           ' wenn gross                          if
    print #LCD2, "<1bh>A<0><06><0f0h>";kb_upper$;
  else
    print #LCD2, "<1bh>A<0><06><0f0h>";kb_lower$;
  endif


#cm
  kb_r0$ = "0123456789"
  kb_r0a$ = "=!<22h>#$<25h><26h><2fh>+*"
  print #LCD2, "<1bh>A<1><06><0f0h>";
  for i = 0 to 9                ' drucke 9 Zeichen mit Abstand
    if shifted = 0 then         ' wenn gross
      print #LCD2, mid$ ( kb_r0a$, i, 1 );"  ";
    else                        ' sonst die Ziffern
      print #LCD2, mid$ ( kb_r0$, i, 1 );"  ";
    endif
  next

  kb_r1$ = "qwertzuiop"
  print #LCD2, "<1bh>A<1><08><0f0h>";
  for i = 0 to 9                ' drucke 9 Zeichen mit Abstand
    if shifted = 0 then         ' wenn gross
      print #LCD2, upper$ ( mid$ ( kb_r1$, i, 1 ));"  ";
    else                        ' sonst klein
      print #LCD2, mid$ ( kb_r1$, i, 1 );"  ";
    endif
  next

  kb_r2$ = "asdfghjkl?"
  print #LCD2, "<1bh>A<1><10><0f0h>";
  for i = 0 to 9
    if shifted = 0 then
      print #LCD2, upper$ ( mid$ ( kb_r2$, i, 1 ));"  ";
    else
      print #LCD2, mid$ ( kb_r2$, i, 1 );"  ";
    endif
  next

  kb_r3$ = "yxcvbnm,.-"
  kb_r3a$ = "YXCVBNM;:_"
  print #LCD2, "<1bh>A<1><12><0f0h>";
  for i = 0 to 9
    if shifted = 0 then
      print #LCD2, mid$ ( kb_r3a$, i, 1 );"  ";
    else
      print #LCD2, mid$ ( kb_r3$, i, 1 );"  ";
    endif
  next

  kb_r4$ = "shift  space    del    ret"
  print #LCD2, "<1bh>A<1><14><0f0h>";kb_r4$;
#endcm
END


'-------------------------------------------------------------------------
' SUB gen_kb_graphic
```

```
'---------------------------------------------------------------------
SUB gen_kb_graphic
  BYTE i, button_xsize, button_ysize, buttonx, buttony

  button_xsize = 2 * FONT_X + 2
  button_ysize = 2 * FONT_Y - 2

  for i = 0 to 9
    buttonx = FONT_X - 4 + (i*(button_xsize+6))
    buttony = -3 + (6 * FONT_Y)
    graphic_copy ( &               ' weisser Text auf schw. Hintergrund
      mnuscr$, &                   ' Ziel
      black$, &                    ' Quelle
      COLUMNS, LINES, &            ' Zielgroesse
      buttonx, buttony, &          ' Position in Ziel
      COLUMNS, LINES, &            ' Quellgroesse
      0, 0, &                      ' Quellposition
      button_xsize, button_ysize, &' kopierte Groesse
      0)                           ' Modus Kopieren
  next

  for i = 0 to 9
    buttonx = FONT_X - 4 + (i*(button_xsize+6))
    buttony = -3 + (8 * FONT_Y)
    graphic_copy ( &               ' weisser Text auf schw. Hintergrund
      mnuscr$, &                   ' Ziel
      black$, &                    ' Quelle
      COLUMNS, LINES, &            ' Zielgroesse
      buttonx, buttony, &          ' Position in Ziel
      COLUMNS, LINES, &            ' Quellgroesse
      0, 0, &                      ' Quellposition
      button_xsize, button_ysize, &' kopierte Groesse
      0)                           ' Modus Kopieren
  next

  for i = 0 to 9
    buttonx = FONT_X - 4 + (i*(button_xsize+6))
    buttony = -3 + (10 * FONT_Y)
    graphic_copy ( &               ' weisser Text auf schw. Hintergrund
      mnuscr$, &                   ' Ziel
      black$, &                    ' Quelle
      COLUMNS, LINES, &            ' Zielgroesse
      buttonx, buttony, &          ' Position in Ziel
      COLUMNS, LINES, &            ' Quellgroesse
      0, 0, &                      ' Quellposition
      button_xsize, button_ysize, &' kopierte Groesse
      0)                           ' Modus Kopieren
  next

  for i = 0 to 9
    buttonx = FONT_X - 4 + (i*(button_xsize+6))
    buttony = -3 + (12 * FONT_Y)
    graphic_copy ( &               ' weisser Text auf schw. Hintergrund
      mnuscr$, &                   ' Ziel
      black$, &                    ' Quelle
      COLUMNS, LINES, &            ' Zielgroesse
      buttonx, buttony, &          ' Position in Ziel
      COLUMNS, LINES, &            ' Quellgroesse
      0, 0, &                      ' Quellposition
      button_xsize, button_ysize, &' kopierte Groesse
```

```
  next

  button_xsize = 5 * FONT_X + 10
  for i = 0 to 3
    buttonx = FONT_X - 4 + (i*(button_xsize+10))
    buttony = -3 + (14 * FONT_Y)
    graphic_copy ( &              ' weisser Text auf schw. Hintergrund
      mnuscr$, &                  ' Ziel
      black$, &                   ' Quelle
      COLUMNS, LINES, &           ' Zielgroesse
      buttonx, buttony, &         ' Position in Ziel
      COLUMNS, LINES, &           ' Quellgroesse
      0, 0, &                     ' Quellposition
      button_xsize, button_ysize, &' kopierte Groesse
      0)                          ' Modus Kopieren
  next

  if LCD_MODE = LCD_MODE40 then   ' wenn LCD-Typ 6
  tmpscr$ = graphic_exp$ ( &      ' Pixel auf Bytes verteilen
    mnuscr$, &                    ' Quellstring
    LCD_TXT_COL8X8,&              ' Quellbreite in Byte
    GR6_SIZE, &                   ' Ziellaenge in Pixel
    LCD_BYT_COL6X8, &             ' Zielbreite in Byte
    6, &                          ' Pixel/Byte im Zielstring
    0)                            ' Shifts nach links im Zielstring
  else
    tmpscr$ = let$ ( mnuscr$ )
  endif
  put #LCD2, #1, tmpscr$, 0, 0, GR_SIZE ' zeige Grafik an
END


'-----------------------------------------------------------------------
' SUB init_vars
'-----------------------------------------------------------------------
SUB init_vars
  screen$ = fill$ ( "<0>", GR8_SIZE ) ' initialisiere alle Strings
  mnuscr$ = fill$ ( "<0>", GR8_SIZE )
  black$ = fill$ ( "<255>", GR8_SIZE )
  tmpscr$ = fill$ ( "<0>", GR_SIZE )

' no key = 00
' shift = ff
'0 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
  conv_upper$ = " &               ' Konversion von Scan- zu Tastencodes
ff 59 41 51 3d 00 00 00 00 00 ff 58 53 57 21 00 & ' 00...0F
00 00 00 00 00 43 44 45 22 00 00 00 00 00 20 56 & ' 10...1F
46 52 23 00 00 00 00 00 20 42 47 54 24 00 00 00 & ' 20...2F
00 00 08 4e 48 5a 25 00 00 00 00 00 00 08 4d 4a 55 & ' 30...3F
26 00 00 00 00 00 08 3b 4b 49 2f 00 00 00 00 00 & ' 40...4F
0d 3a 4c 4f 2b 00 00 00 00 00 0d 5f 3f 50 2a 00 & ' 50...5F
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 & ' 60...6F
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 & ' 70...7F
"%

  conv_lower$ = " &               ' Konversion von Scan- zu Tastencodes
ff 79 61 71 30 00 00 00 00 00 ff 78 73 77 31 00 & ' 00...0F
00 00 00 00 00 63 64 65 32 00 00 00 00 00 20 76 & ' 10...1F
66 72 33 00 00 00 00 00 20 62 67 74 34 00 00 00 & ' 20...2F
00 00 08 6e 68 7a 35 00 00 00 00 00 08 6d 6a 75 & ' 30...3F
```

```
0d 2e 6c 6f 38 00 00 00 00 00 0d 2d 3f 70 39 00 & ' 50...5F
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 & ' 60...6F
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 & ' 70...7F
"%

  kb_lower$ = "&                  ' Tastaturbeschriftung Kleinbuchst.
 0  1  2  3  4  5  6  7  8  9<13><10><10>&
 q  w  e  r  t  z  u  i  o  p<13><10><10>&
 a  s  d  f  g  h  j  k  l  ?<13><10><10>&
 y  x  c  v  b  n  m  ,  .  -<13><10><10>&
 shift  space    del    ret"

  kb_upper$ = "&                  ' Tastaturbeschriftung Grossbuchst.
 =  !  <22h>  #  $  <25h>  <26h>  <2fh>  +  *<13><10><10>&
 Q  W  E  R  T  Z  U  I  O  P<13><10><10>&
 A  S  D  F  G  H  J  K  L  ?<13><10><10>&
 Y  X  C  V  B  N  M  ;  :  _<13><10><10>&
 shift  space    del    ret"

END
```

Leere Seite

5 Häufig gestellte Fragen - Support

Wie wende ich die Instruktion PWM an?

Beim BASIC-Tiger® wird an Stelle einer speziellen BASIC-Instruktion ein entsprechender Device-Treiber verwendet. Durch die Wahl des Device-Treibers wird z.B. festgelegt, ob man die internen PWM-Pins des BASIC-Tigers® benutzt oder externe PWM-/DA-Bausteine anspricht. Nachdem der entsprechende Treiber eingebunden ist, wird einfach der gewünschte Wert, beim Treiber „PWM1.TDD" zwischen 0 und 255, auf den Ausgabekanal des PWM-Treibers ausgegeben. Da BASIC-Tiger® ein Multitasking-System ist, bleibt der einmal eingestellte Wert so lange am PWM-Ausgang erhalten, bis man ihn gezielt durch einen neuen Wert überschreibt. Weder bleibt der BASIC-Tiger® bei der PWM-Ausgabe stehen, noch endet der eingestellte PWM-Wert, wenn die Programm-Ausführung zur nächsten BASIC-Instruktion kommt.

Kann ich das BASIC-Tiger®-Modul auch in Maschinensprache programmieren?

Nein. Die einzig verfügbare Ebene ist die Programmierung in Tiger-BASIC®. Tiger-BASIC® erzeugt sehr schnell ablaufenden Programmcode der dem Wunsch nach hoher Geschwindigkeit entspricht. Je nach Anforderungen an Durchsatz oder Antwortverhalten, wird ein Tiger Modul der entsprechenden Leistungsklasse ausgewählt.

Die Abwesenheit einer niedrigeren Sprachebene hat den Vorteil, daß Tiger-BASIC® Programme sehr leicht zwischen verschiedenen Modultypen portierbar sind. Ebenso ist es praktisch unmöglich, ungewollt das Laufzeit-System zum Absturz zu bringen, was zu der hohen Zuverlässigkeit des BASIC-Tigers® entscheidend beiträgt.

Wie spreche ich die erweiterten Outputs auf dem Plug & Play Lab an?

Genauso wie die internen Ports, lediglich mit anderer Adresse. Der mögliche Adress-Bereich für erweitere Ports liegt zwichen: 10H....FFH (16...255). Auf dem Plug & Play Lab liegen die 8 ePort-Ausgänge im Adress-Bereich: 10H...17H (16...23). Oberhalb der Tastatur sind alle 8 x 8 Ausgangs-Bits dieser ePorts durch ein LED vertreten. Ein Befehl zur Ausgabe eines Bytes auf den untersten ePort (10H) könnte also lauten: OUT 10H,0FFH,data_byte.

Ich setze Port-Pins zum Beispiel mit der OUT-Instruktion, aber an den Pins tut sich nichts. Was ist der Grund?

Die Ports eines BASIC-Tiger® Moduls können grundsätzlich für ganz verschiedene Zwecke eingesetzt werden. Mit Instruktionen wie IN und OUT kann direkt auf die Ports eines Moduls zugegriffen werden - sofern diese Pins nicht schon anderweitig belegt sind. Die Mechanismen. die außerdem Ports verwenden sind: Device-Treiber sowie das ePort-System zur Erweiterung auf bis zu 1920 zusätzliche digitale Ein- oder Ausgänge.

Verwendet z.B. ein Device-Treiber wie „LCD1.TDD“ einen Port-Pin, so kann der Treiber bei Bedarf ihn für alle anderen Zugriffe (mit IN oder OUT) sperren. Es ist zwar noch erlaubt auf einen solchen Port mit OUT eine Ausgabe auszuführen, die reservietren Pins bleiben jedoch vollkommen unverändert. Dies ist ein Schutzmechanismus um die richtige Funktion der angeschlossenen Perihperie-Geräte zu sichern und es vereinfacht die Programmierung, da man selbst keine ausdrückliche Vorkehrung treffen muß um solche Pins unverändert zu lassen

Im Plug & Play Lab sind die Pins L60...L67 und L30...L37 zur Ansteuerung erweiterter Ports belegt und können somit nicht mit IN oder OUT angesprochen werden.

Sie können auf die Aktivität erweiterter Ports mit der Compiler-Anweisung ***USER_EPORT NOACTIVE*** *verzichten (Siehe Hardware-Handbuch). Dann sind diese Pins wie ursprünglich erwartet ansprechbar.*

Sind die Device-Treiber re-entrant?

Device-Treiber können wie auch Subroutinen grundsätzlich von beliebig vielen Tasks gleichzeitig verwendet werden. Im Unterschied zu Subroutinen steht hinter jedem Device-Treiber jedoch ein Ein- oder Ausgabe-Gerät. So lange das entsprechende Peripherie-Gerät sinnvoll angesprochen wird erhält man auch sinnvolle Resultate.

Ein Beispiel. Auf dem LC-Display des Plug & Play Labs können verschiedene Tasks gleichzeitig ihre Ausgaben machen. Man sollte dabei lediglich die Position des LCD-Cursors und die Puffer-Situation beachten. Eine Möglichkeit besteht darin, daß jede Task, die etwas auf dem LCD ausgeben will, in jeder PRINT-Instruktion zu Beginn eine absolute Cursor-Positionierung ausführt und danach die entsprechende Ausgabe. Da eine einzelne BASIC-Instruktion niemals mitten drin vom Multitasking-System unterbrochen werden, wird eine PRINT-Instruktion immer vollständig abgearbeitet. Die Cursor-Positionierung und die nachfolgende Ausgabe wird daher nicht gestört.

Weiterhin sollte man darauf achten,daß der Ausgabe Puffer des LCDs nicht ständig voll ist - was auch keinen Sinn machen würde. Ein vorwiegend voller Ausgangs-Puffer könnte folgenden Effekt erzeugen: Task X schreibt in den Ausgangs-Puffer bis er voll ist. Kurz danach will auch Task Y in diesen Puffer schreiben, kann es jedoch nicht, weil der Puffer immer noch zu voll ist.Task Y „wartet“ also. Etwas später, wenn Task X wieder ein PRINT auf diesen Device ausführt, kann der Puffer grade wieder genug Platz für weitere Daten haben, wodurch er wieder voll wird. Task X hat also erneut Daten abgesetzt. Die kurz darauf folgende Task Y würde ev. abermals einen vollen Puffer vorfinden .... Es entstehen Interferrenzen von zufälligem Character.

Warum werden nach dem Einschalten oder einem Reset die DIP-Schalter zunächst falsch gelesen?

Es wird stets das Ergebnis des letzten Scans gelesen. Nach einem Reset muss ein Scan abgewartet werden, also ca. 20msec.

Unterstützt BASIC-Tiger® die 1-wire-Chips von Dallas?

Der Treiber TMEM unterstützt TouchMemory-Chips der Serie DS199x. Ansonsten empfehlen wir zur Unterstützung der 1-wire-Chips den ebenfalls von Dallas gelieferten Chip DS2480, der eine Umwandlung von RS-232 nach 1-wire vornimmt.

Rat und Hilfe

Funktioniert einmal etwas nicht wie erwartet? – Dann nehmen Sie Kontakt auf zum Tiger Support Team, das Ihnen gerne mit Rat und Hilfe für Ihr Problem zur Verfügung steht.

Zur Selbsthilfe sei auch auf den Abschnitt ‚Tiger-BASIC® für Fortgeschrittene' im Programmierhandbuch hingewiesen, wo häufig gemachte Fehler kurz zusammengefasst sind.

Damit es möglichst schnell geht, senden uns Sie bei jeder Anfrage mindestens diese Informationen und Files zu:

- Lauffähiges Programm-Beispiel als „.TIG" File:
- Reduziert auf ein möglichst einfaches Programm-Beispiel bei dem der Fehler auftritt. Maximal sollte dabei eine Seite herauskommen, meist sogar nur ein paar Zeilen.
- Service-Informationen: Senden Sie uns auch die Datei, die Tiger-BASIC® mit dem Befehl ‚Service-Informationen' aus dem Menü ‚Hilfe' als Fenster öffnet.
- Welches Tiger-Modul setzen Sie ein (Typ und Version)?
- Beschreiben Sie den Fehler so genau wie möglich.
- In welchem Zusammenhang tritt das Problem auf?
- Tritt es immer nachvollziehbar auf oder nur gelegentlich?
- Wie sind Sie am besten erreichbar: E-mail, Fax- und Telefonnummern für ev. Rückfragen und Antwort nicht vergessen!

BASIC-Tiger®-Service-Hotline, Mo…Fr. 8.oo–17.oo Uhr:

0241 / 15 15 99

Wilke Technology GmbH
Heider-Hof-Weg 23 D
Postfach 1727

D-52080 Aachen / Germany

Tel: 02405 / 480 55 0
Fax: 02405 / 480 55 444
eMail: support@wilke.de

Leere Seite

6 Stichwortverzeichnis

Leere Seite

Leere Seite

7 Anhang

ASCII-Tabelle

ASCII, gesprochen ‘aski’, ist eine Abkürzung für ‚**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange’. Der ASCII-Code ist wahrscheinlich der am weitesten verbreitete Code zur Darstellung von Buchstaben, Ziffern, einigen Sonderzeichen und Steuerzeichen. Er findet im PC, Macintosh und Internet Verwendung.

CHAR	HEX	DEC	CHAR	HEX	DEC	CHAR	HEX	DEC	CHAR	HEX	DEC
NUL	00	000	SP	20	032	@	40	064		60	096
SOH	01	001	!	21	033	A	41	065	a	61	097
STX	02	002	"	22	034	B	42	066	b	62	098
ETX	03	003	#	23	035	C	43	067	c	63	099
EOT	04	004	$	24	036	D	44	068	d	64	100
ENQ	05	005	%	25	037	E	45	069	e	65	101
ACK	06	006	&	26	038	F	46	070	f	66	102
BEL	07	007	'	27	039	G	47	071	g	67	103
BS	08	008	(	28	040	H	48	072	h	68	104
HT	09	009	)	29	041	I	49	073	i	69	105
LF	0A	010	*	2A	042	J	4A	074	j	6A	106
VT	0B	011	+	2B	043	K	4B	075	k	6B	107
FF	0C	012	,	2C	044	L	4C	076	l	6C	108
CR	0D	013	-	2D	045	M	4D	077	m	6D	109
SO	0E	014	.	2E	046	N	4E	078	n	6E	110
SI	0F	015	/	2F	047	O	4F	079	o	6F	111
DLE	10	016	0	30	048	P	50	080	p	70	112
D1	11	017	1	31	049	Q	51	081	q	71	113
D2	12	018	2	32	050	R	52	082	r	72	114
D3	13	019	3	33	051	S	53	083	s	73	115
D4	14	020	4	34	052	T	54	084	t	74	116
NAK	15	021	5	35	053	U	55	085	u	75	117
SYN	16	022	6	36	054	V	56	086	v	76	118
ETB	17	023	7	37	055	W	57	087	w	77	119
CAN	18	024	8	38	056	X	58	088	x	78	120
EM	09	025	9	39	057	Y	59	089	y	79	121
SUB	1A	026	:	3A	058	Z	5A	090	z	7A	122
ESC	1B	027	;	3B	059	[	5B	091	{	7B	123
FS	1C	028	<	3C	060	\	5C	092	|	7C	124
GS	1D	029	=	3D	061	]	5D	093	}	7D	125
RS	1E	030	>	3E	062	^	5E	094	~	7E	126
US	1F	031	?	3F	063	_	5F	095	DEL	7F	127

EBCDIC Codes

EBCDIC, gesprochen ‘eb-si-dik’, ist eine Abkürzung für ‚**E**xtended **B**inary-**C**oded **D**ecimal **I**nterchange **C**ode’. EBCDIC ist ein IBM-Code zur Darstellung von Buchstaben, Ziffern und einigen Sonderzeichen. Obwohl der Code auf großen IBM-Computern weit verbreitet ist, verwenden die meisten anderen Computer (PCs, Macintosh) den ASCII-Code.

Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec	Char	Hex	Dec
blank	40	64	a	81	129	A	C1	193	0	F0	240
.	4B	75	b	82	130	B	C2	194	1	F1	241
<	4C	76	c	83	131	C	C3	195	2	F2	242
(	4D	77	d	84	132	D	C4	196	3	F3	243
+	4E	78	e	85	133	E	C5	197	4	F4	244
|	4F	79	f	86	134	F	C6	198	5	F5	245
&	50	80	g	87	135	G	C7	199	6	F6	246
!	5A	90	h	88	136	H	C8	200	7	F7	247
$	5B	91	i	89	137	I	C9	201	8	F8	248
*	5C	92	j	91	145	J	D1	209	9	F9	249
)	5D	93	k	92	146	K	D2	210			
;	5E	94	l	93	147	L	D3	211			
-	60	96	m	94	148	M	D4	212			
/	61	97	n	95	149	N	D5	213			
,	6B	107	o	96	150	O	D6	214			
%	6C	108	p	97	151	P	D7	215			
_	6D	109	q	98	152	Q	D8	216			
>	6E	110	r	99	153	R	D9	217			
?	6F	111	s	A2	162	S	E2	226			
:	7A	122	t	A3	163	T	E3	227			
#	7B	123	u	A4	164	U	E4	228			
@	7C	124	v	A5	165	V	E5	229			
'	7D	125	w	A6	166	W	E6	230			
=	7E	126	x	A7	167	X	E7	231			
"	7F	127	y	A8	168	Y	E8	232			
			z	A9	169	Z	E9	233			

Baudot-Code-Satz

Baudot, gesprochen ‘bodoh’, wurde 1874 erstmals in einem ‚Telegraph’ (Fernschreiber) eingesetzt, ist aber bis heute in einigen Bereichen zur datenübertragung in Gebrauch. Die Zeichen ‚LTRS=Letters’ und ‚FIGS=Figures’ schalten zwischen den beiden Zeichensätzen um. Das rechts-stehende niederwertige Bit (LSB) wird zuerst gesendet. Den Codes werden entweder Buchstaben oder Sonder- und Steuerzeichen zugeordnet.

LTRS	FIGS	HEX	BITS
A	-	03	00011
B	?	19	11001
C	:	0E	01110
D	$	09	01001
E	3	01	00001
F	!	0D	01101
G	&	1A	11010
H	STOP	14	10100
I	8	06	00110
J	'	0B	01011
K	(	0F	01111
L	)	12	10010
M	.	1C	11100
N	,	0C	01100
O	9	18	11000
P	0	16	10110
Q	1	17	10111
R	4	0A	01010
S	BELL	05	00101
T	5	10	10000
U	7	07	00111
V	;	1E	11110
W	2	13	10011
X	/	1D	11101
Y	6	15	10101
Z	"	11	10001
n/a	n/a	00	00000
CR	CR	08	01000
LF	LF	02	00010
SP	SP	04	00100
LTRS	LTRS	1F	11111
FIGS	FIGS	1B	11011

ANSI-Steuersequenzen

Die in der Tabelle gelisteten ANSI-Steuersequenzen sind nützlich, wenn ein serielles ANSI-Terminal bzw. ein PC mit Terminalprogramm mit ANSI-Emulation angeschlossen ist. Auf dem Bildschirm lassen sich so die Ausgaben positionieren und eventuell farblich darstellen. Die Liste erhebt keinen Anspruch auf Vollständigkeit.

ANSI-Sequenzen beginnen mit ESC gefolgt von „[“. Es folgen weitere Bytes als Kommando, Parameter und eventuell ein Abschluß. In der Tabelle wird die Darstellung <0> für ein Byte verwendet, so wie in Strings in Tiger-BASIC® üblich. Folgende Abkürzungen gelten:

<zb> = Zeile als Binärwert
<sb> = Spalte als Binärwert
<nb> = binärer Wert.

Wird ein Wert nicht angegeben, wird der Wert ‚eins’ angenommen. In Tiger-BASIC® wird eine ANSI-Sequenz zum Beispiel so ausgegeben (Cursor in Zeile 2, Spalte 3):

PUT #1, “<27>[<2>;<3>H“

ANSI-Sequenz	Funktion
ESC [2J	Löscht den Bildschirm und setzt den Cursor auf die Home-Position.
ESC [<zb>;<sb>H	Bewegt den Cursor an die angegebene Position (Home ist <1>;<1>)
ESC [<zb>;<sb>f	Bewegt den Cursor an die angegebene Position (Home ist <1>;<1>)
ESC [<nb>A	Bewegt den Cursor um n Zeilen nach oben, ohne die Spalte zu beeinflussen, bis maximal Position 1.
ESC [<nb>B	Bewegt den Cursor um n Zeilen nach unten, ohne die Spalte zu beeinflussen, bis maximal zur letzten Zeile.
ESC [<nb>C	Bewegt den Cursor um n Zeilen nach rechts, ohne die Zeile zu beeinflussen, bis maximal zur letzten Position.
ESC [<nb>D	Bewegt den Cursor um n Zeilen nach links, ohne die Zeile zu beeinflussen, bis maximal zur Position 1.
ESC [6n	Gerät gibt den Cursorpositions-Report in dieser Form aus: ESC [<zb>;<sb>R
ESC [s	Gerät speichert die aktuelle Cursorposition.
ESC [u	Gerät bewegt den Cursor an die zuletzt gespeicherte Position.
ESC [K	Löscht den Inhalt einer Zeile ab der Cursorposition bis zum Ende der Zeile.
ESC [<nb>;...;<nb>m	Funktion ‘Set Graphic Rendition’ schaltet grafische Attribute. Siehe Tabelle im Anschluß

ESC-m-Parameter	Funktion
0	Alle Attribute aus
1	Fett ein
2	Schwach ein
3	Kursiv ein
5	Blinken ein
6	Schnelles Blinken ein
7	Invers ein
8	Verdeckt ein
30...47	Verschiedene Vorder- und Hintergrundfarben

Windows 95/98/NT Shortcuts

Die in der Tabelle stellt eine Übersicht über die wichtigsten Windows 95/98/NT Shortcuts dar. Die Liste erhebt keinen Anspruch auf Vollständigkeit.

Eingabe	Funktion
F1	Hilfe aufrufen
F10	Menüleiste (de)aktivieren
SHIFT + F10	Kontextmenü aufrufen
CTRL + C	Kopieren
CRTL + X	Ausschneiden
CTRL + V	Einfügen
CTRL + A	Alles markieren
CTRL + S	Speichern
CTRL + N	Neues Fenster öffnen
CTRL + Z	Letzte Aktion widerrufen
CTRL + ESC	Startmenü aufrufen
CTRL + F4	Aktuelles Unter-Fenster in Programm schließen
CTRL + TAB	Vorwärts Unter-Fenster / Registrierkarten wechseln
CTRL + SHIFT + TAB	Rückwärts Unter-Fenster / Registrierkarten wechseln
CTRL + O	Öffnen
CTRL + P	Drucken
CTRL + Pos1	Zum Anfang eines Dokuments
CTRL + Ende	Zum Ende eines Dokuments
CTRL + WIN + F	Suche Computer
CTRL + ALT + DEL	Öffnet Fenster „Anwendung schließen“

Eingabe	Funktion
WIN + TAB	Umschalten zwischen Programmen in Task-Leiste
WIN + E	Explorer starten
WIN + F1	Hilfe aufrufen
WIN	Startmenü aufrufen
WIN + BREAK	System-Eigenschaften aufrufen
WIN + D	Minimieren bzw. Wiederherstellen aller Fenster
WIN + M	Minimieren aller Fenster
WIN + SHIFT + M	Zurücknehmen minimieren aller Fenster
WIN + R	Dialogfeld „Ausführen“ aufrufen
WIN + F	Dialogfeld „Suchen“ aufrufen
ALT + TAB	Schaltet zum nächsten Programm in der Taskleiste
ALT + SHIFT + TAB	Schaltet zum vorigen Programm in der Taskleiste
ALT + F4	Programm(fenster) schließen
ALT + F10	Menüleiste einer Anwendung aktivieren
ALT + SPACE	Systemmenü der aktuellen Anwendung anzeigen
ALT + ESC	Wechselt zwischen Explorer und anderen anderen Anwendungen
ALT + -	Systemmenü des aktuellen Unterfensters anzeigen

Short-Cuts Tiger-BASIC® Version 5

Eingabe	Funktion
Pfeiltasten	Ein Zeichen vor oder zurück
Pfeiltasten	Eine Zeile vor oder zurück
STRG-Pfeiltasten	Ein Wort vor oder zurück
Bild-auf-, Bild-ab-Taste	Eine Bildschirmseite vor oder zurück
STRG-Pos1	An den Anfang des Textes
STRG-Ende	An das Ende des Textes
STRG-F7/F8	Zum nächsten/vorigen Fehler
Bildlaufleisten	Schnell vor oder zurück
Suchfunktion	An eine bestimmte Stelle im Text
Strg-S	Datei-Speichern
Strg-Z	Bearbeiten-Rückgängig
Strg-X	Bearbeiten-Ausschneiden
Strg-C	Bearbeiten-Kopieren
Strg-V	Bearbeiten-Einfügen
Strg-A	Bearbeiten-Alles markieren
Strg-F	Suchen-Suche nach
Strg-R	Suchen-Ersetzen
F3	Suche-Wiederholen
Strg-G	Gehe zu Zeile
Strg-F9	Anzeige Meldungen
Strg-M	Anzeige-Tigerstatus
ALT-F5	Anzeige-Ausdruck auswerten
Strg-F5	Anzeige-Überwachte Ausdrücke
Strg-W	Anzeige-Überwachte Ausdrücke aktualisieren
Strg-N	Anzeige-zu überwachten Ausdrücken hinzufügen
F4	Starten-Compilieren

Eingabe	Funktion
F5	Starten-Ausführen
Strg-L	Starten-Programm laden
Strg-D	Starten-Programm löschen
F6	Debug-nächste Anweisung (in Task)
ALT-F6	Debug-Anweisungen ausführen (in Task)
F7	Debug-Unterprogramm überspringen (in Task)
ALT-F7	Debug-Unterprogramme überspringen (in Task)
F8	Debug-nächste Anweisung (überall)
ALT-F8	Debug-Anweisungen ausführen (überall)
Strg-T	Debug-bis zum Cursor ausführen
Strg-H	Debug-Programm anhalten
Strg-E	Debug-Programm beenden (Reset)
F2	Debug-Haltepunkt umschalten
ALT-F2	Debug-Haltepunkt hinzufügen
Strg-K	Debug-Alle Haltepunkte löschen
Strg-B	Debug-Editierschutz ein-/ausschalten
Strg-F3	Optionen-Übertragung
F1	Hilfe

Kennzeichnung von Widerständen und Kondensatoren

Kennzeichnung von Widerständen / Kondensatoren mit Angabe des Temperatur-Koeffizienten (DIN-41429 / DIN-IEC-62 / IEC 115-1-4.5)

Farbcodes

Farbe	Ziffer	Multiplikator	Toleranz	Temp.-Koeff.
schwarz	0	10^0	-	± 250 * 10^{-6}/K
braun	1	10^1	±1%	± 100 * 10^{-6}/K
rot	2	10^2	±2%	± 50 * 10^{-6}/K
orange	3	10^3	-	± 15 * 10^{-6}/K
gelb	4	10^4	-	± 25 * 10^{-6}/K
grün	5	10^5	±0,5%	± 20* 10^{-6}/K
blau	6	10^6	±0,25%	± 10 * 10^{-6}/K
violett	7	10^7	±0,1%	± 5 * 10^{-6}/K
grau	8	10^8	-	± 1 * 10^{-6}/K
weiß	9	10^9	-	-
silber	-	10^{-2}	±10%	-
gold	-	10^{-1}	±5%	-

Wertkennzeichnung durch Buchstaben

(DIN 1301/12.93, EN 60 062/10.94)

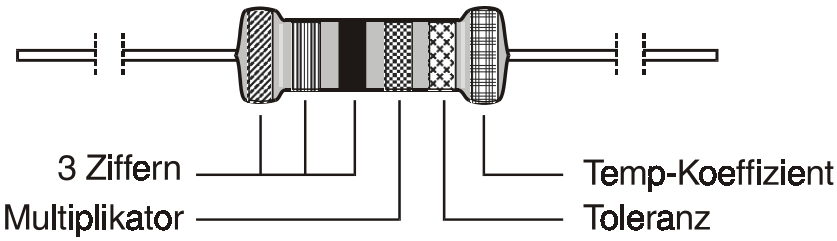
Kennbuchstabe	Name	Multiplikator
y	Yocto	10^{-24}
z	Zepto	10^{-21}
a	Atto	10^{-18}
f	Femto	10^{-15}
p	Pico	10^{-12}
n	Nano	10^{-9}
µ	Mikro	10^{-6}
m	Milli	10^{-3}
c	Centi	10^{-2}
d	Dezi	10^{-1}
R,F	-	10^{0}
da	Deka	10^{1}
h	Hekto	10^{2}
K	Kilo	10^{3}
M	Mega	10^{6}
G	Giga	10^{9}
T	Tera	10^{12}
P	Peta	10^{15}
E	Exa	10^{18}
Z	Zetta	10^{21}
Y	Yotta	10^{24}

Toleranzkennzeichnung durch Buchstaben

(EN 60 062/10.94)

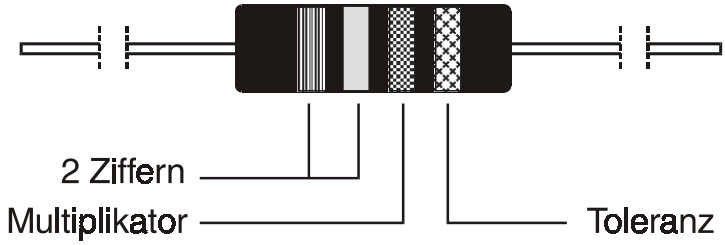
Abweichungen	
Toleranz	**Buchstabe**
Symmetrische Abweichung:	
±0,1%	B
±0,25%	C
±0,5%	D
±1%	F
±2%	G
±5%	J
±10%	K
±20%	M
±30%	N
Unsymmetrische Abweichung:	
+30...-10%	Q
+50...-10%	T
+50...-20%	S
+80...-20%	Z
Symmetrische Abweichung für Kapazitätswerte < 10pF:	
±0,1 pF	B
±0,25pF	C
±0,5pF	D
±1pF	F

Anbringung der Farbcodes an Widerständen:



Beispiel: rot-gelb-weiß-rot-braun-rot

24,9 kOhm, +/-1%, +/-50 ppm/°K



Beispiel: gelb-violett-orange-gold

47 kOhm, +/-5%

Mittlere Schrittweite für Widerstands-Wachstum aufeinanderfolgender Werte:

E3	E6	E12	E24	E48	E96	E192
+115%	+47,8%	+21,2%	+10,07%	+4,914%	+2,428%	+1,206%

Normreihen für Widerstandswerte

E3	E6 ±20%	E12 ±10%	E24 ±5%	E48 ±2%	E96 ±1%	E192 ±0,5%
100	100	100	100	100	100	100
						101
					102	102
						104
				105	105	105
						106
					107	107
						109
			110	110	110	110
						111
					113	113
						114
				115	115	115
						117
					118	118
		120	120			120
				121	121	121
						123
					124	124
						126
				127	127	127
						129
			130		130	130
						132
				133	133	133
						135
					137	137
						138

E3	E6 ±20%	E12 ±10%	E24 ±5%	E48 ±2%	E96 ±1%	E192 ±0,5%
				140	140	140
						142
					143	143
						145
				147	147	147
						149
	150	150	150		150	150
						152
				154	154	154
						156
					158	158
			160			160
				162	162	162
						164
					165	165
						167
				169	169	169
						172
					174	174
						176
				178	178	178
		180	180			180
					182	182
						184
				187	187	187
						189
					191	191
						193
				196	196	196
						198
			200		200	200
						203
				205	205	205
						208
					210	210
						213

E3	E6 ±20%	E12 ±10%	E24 ±5%	E48 ±2%	E96 ±1%	E192 ±0,5%
				215	215	215
						218
220	220	220	220		221	221
						223
				226	226	226
						229
					232	232
						234
				237	237	237
			240			240
					243	243
						246
				249	249	249
						252
					255	255
						258
				261	261	261
						264
					267	267
		270	270			271
				274	274	274
						277
					280	280
						284
				287	287	287
						291
					294	294
						298
			300	301	301	301
						305
					309	309
						312
				316	316	316
						320
					324	324
						328

E3	E6 ±20%	E12 ±10%	E24 ±5%	E48 ±2%	E96 ±1%	E192 ±0,5%
	330	330	330	332	332	332
						336
					340	340
						344
				348	348	348
						352
					357	357
			360			361
				365	365	365
						370
					374	374
						379
				383	383	383
						388
		390	290		392	392
						397
				402	402	402
						407
					412	412
						417
				422	422	422
						427
			430		432	432
						437
				442	442	442
						448
					453	453
						459
				464	464	464
470	470	470	470			470
					475	475
						481
				487	487	487
						493
					499	499
						505

E3	E6 ±20%	E12 ±10%	E24 ±5%	E48 ±2%	E96 ±1%	E192 ±0,5%
			510	511	511	511
						517
					523	523
						530
				536	536	536
						542
					549	549
						556
		560	560	562	562	562
						569
					576	576
						583
				590	590	590
						597
					604	604
						612
			620	619	619	619
						626
					634	634
						642
				649	649	649
						657
					665	665
						673
	680	680	680	681	681	681
						690
					698	698
						706
				715	715	715
						723
					732	732
						741
			750	750	750	750
						759
					768	768
						777

E3	E6 ±20%	E12 ±10%	E24 ±5%	E48 ±2%	E96 ±1%	E192 ±0,5%
				787	787	787
						796
					806	806
						816
		820	820	825	825	825
						835
					845	845
						856
				866	866	866
						876
					887	887
						898
			910	909	909	909
						920
					931	931
						942
				953	953	953
						965
					976	976
						998

Leere Seite

BASIC-Tiger® Modul A – Nutzung der Pins

Übersicht der I/O-Pin-Nutzung der wichtigsten Device-Treiber. Einige Funktionen sind fest an den entsprechenden Pin gebunden (PWM, PLSO1, PLSIN1), andere sind lediglich die Standardbelegung, lassen sich aber auf andere Pins verlegen (LCD, Strobe, Busy):

Funktion	Pin-Bez.	Pin-Nr	Pin-Nr	Pin-Bez.	Funktion
	res.	1	46	VCC	
D0	L60	2	45	Batt.	
D1	L61	3	44	AGND	
D2	L62	4	43	Vref	
D3	L63	5	42	An3	
D4	L64	6	41	An2	
D5	L65	7	40	An1	
D6	L66	8	39	An0	
D7	L67	9	38	Alarm	
Busy	L70	10	37	L41/PC	
Strobe	L71	11	36	L40	
PWM0	L72	12	35	L42	Beep
PWM1	L73	13	34	L37	LCD1-RS
LCD-WR	L80	14	33	L36	LCD1-E
LCD-RD	L81	15	32	L35	INE
LCD-CE	L82	16	31	L34	Dclk
LCD-CD	L83	17	30	L33	Aclk
PLSIN1	L84	18	29	L95	RTS
	L85	19	28	L94	Rx1
PLSO1	L86	20	27	L93	Tx1
	L87	21	26	L92	CT0
	Reset-in	22	25	L91	Rx0
	GND	23	24	L90	Tx0

Leere Seite

Freie Tabelle BASIC-Tiger®-Modul A als Kopiervorlage. Projekt:

Funktion	Pin-Bez.	Pin-Nr	Pin-Nr	Pin-Bez.	Funktion
	res.	1	46	VCC	
	L60	2	45	Batt.	
	L61	3	44	AGND	
	L62	4	43	Vref	
	L63	5	42	An3	
	L64	6	41	An2	
	L65	7	40	An1	
	L66	8	39	An0	
	L67	9	38	Alarm	
	L70	10	37	L41/PC	
	L71	11	36	L40	
	L72	12	35	L42	
	L73	13	34	L37	
	L80	14	33	L36	
	L81	15	32	L35	
	L82	16	31	L34	
	L83	17	30	L33	
	L84	18	29	L95	
	L85	19	28	L94	
	L86	20	27	L93	
	L87	21	26	L92	
	Reset-in	22	25	L91	
	GND	23	24	L90	

Leere Seite

TINY-Tiger® – Nutzung der Pins

Übersicht der I/O-Pin-Nutzung der wichtigsten Device-Treiber. Einige Funktionen sind fest an den entsprechenden Pin gebunden (PWM, PLSO1, PLSIN1), andere sind lediglich die Standardbelegung, lassen sich aber auf andere Pins verlegen (LCD, Strobe, Busy):

Funktion	Pin-Bez.	Pin-Nr	Pin-Nr	Pin-Bez.	Funktion
D0	L60	1	44	VCC	
D1	L61	2	43	Batt.	
D2	L62	3	42	Vref	
D3	L63	4	41	AGND	
D4	L64	5	40	An3	
D5	L65	6	39	An2	
D6	L66	7	38	An1	
D7	L67	8	37	An0	
Busy	L70	9	36	L41/PC	
Strobe	L71	10	35	res.	
PWM0	L72	11	34	Alarm	
PWM1	L73	12	33	L37	LCD1-RS
LCD-WR	L80	13	32	L36	LCD1-E
LCD-RD	L81	14	31	L35	INE
LCD-CE	L82	15	30	L34	Dclk
LCD-CD	L83	16	29	L33	Aclk
PLSIN1	L84	17	28	L95	RTS
	L85	18	27	L94	Rx1
PLSO1	L86	19	26	L93	Tx1
	L87	20	25	L92	CT0
	Reset-in	21	24	L91	Rx0
	GND	22	23	L90	Tx0

Leere Seite

Freie Tabelle TINY-Tiger® als Kopiervorlage. Projekt:

Funktion	Pin-Bez.	Pin-Nr	Pin-Nr	Pin-Bez.	Funktion
	L60	1	44	VCC	
	L61	2	43	Batt.	
	L62	3	42	Vref	
	L63	4	41	AGND	
	L64	5	40	An3	
	L65	6	39	An2	
	L66	7	38	An1	
	L67	8	37	An0	
	L70	9	36	L41/PC	
	L71	10	35	res.	
	L72	11	34	Alarm	
	L73	12	33	L37	
	L80	13	32	L36	
	L81	14	31	L35	
	L82	15	30	L34	
	L83	16	29	L33	
	L84	17	28	L95	
	L85	18	27	L94	
	L86	19	26	L93	
	L87	20	25	L92	
	Reset-in	21	24	L91	
	GND	22	23	L90	

Leere Seite

TINY-Tiger® Modul E – Nutzung der Pins

Übersicht der I/O-Pin-Nutzung der wichtigsten Device-Treiber. Einige Funktionen sind fest an den entsprechenden Pin gebunden (PWM, PLSO1, PLSIN1), andere sind lediglich die Standardbelegung, lassen sich aber auf andere Pins verlegen (LCD, Strobe, Busy):

Funktion	Pin-Bez.	Pin-Nr	Pin-Nr	Pin-Bez.	Funktion
D0	L60	1	28	VCC	
D1	L61	2	27	L37	LCD1-RS
D2	L62	3	26	L36/An3	LCD1-E
D3	L63	4	25	L35/An2	INE
D4	L64	5	24	L34/An1	Dclk
D5	L65	6	23	L33/An0	Aclk
D6	L66	7	22	L41/PC	
D7	L67	8	21	L85	
LCD-WR	L80	9	20	Reset-in	
LCD-RD	L81	10	19	L94	Rx1
LCD-CE	L82	11	18	L93	Tx1
LCD-CD	L83	12	17	L92/L86	CT0/PLSO1
PLSIN1	L84	13	16	L91/L87	Rx0
	GND	14	15	L90	Tx0

Leere Seite

Freie Tabelle TINY-Tiger®-Modul E als Kopiervorlage. Projekt:

Funktion	Pin-Bez.	Pin-Nr	Pin-Nr	Pin-Bez.	Funktion
	L60	1	28	VCC	
	L61	2	27	L37	
	L62	3	26	L36/An3	
	L63	4	25	L35/An2	
	L64	5	24	L34/An1	
	L65	6	23	L33/An0	
	L66	7	22	L41/PC	
	L67	8	21	L85	
	L80	9	20	Reset-in	
	L81	10	19	L94	
	L82	11	18	L93	
	L83	12	17	L92/L86	
	L84	13	16	L91/L87	
	GND	14	15	L90	

Leere Seite

BASIC-Tiger®-CAN-Modul – Nutzung der Pins

Übersicht der I/O-Pin-Nutzung der wichtigsten Device-Treiber. Einige Funktionen sind fest an den entsprechenden Pin gebunden (PWM, PLSO1, PLSIN1), andere sind lediglich die Standardbelegung, lassen sich aber auf andere Pins verlegen (LCD, Strobe, Busy). **Die B-Reihe ist nicht aufgeführt**, da die meisten Pins unbelegt sind und die belegten Pins keine Variable Funktion zulassen:

Funktion	Pin-Bez.	Pin-Nr	Pin-Nr	Pin-Bez.	Funktion
	res.	1	46	VCC	
D0	L60	2	45	Batt.	
D1	L61	3	44	AGND	
D2	L62	4	43	Vref	
D3	L63	5	42	An3	
D4	L64	6	41	An2	
D5	L65	7	40	An1	
D6	L66	8	39	An0	
D7	L67	9	38	Alarm	
Busy	L70	10	37	L41/PC	
Strobe	L71	11	36	L40	
PWM0	L72	12	35	L42	Beep
PWM1	L73	13	34	L37	LCD1-RS
LCD-WR	L80	14	33	L36	LCD1-E
LCD-RD	L81	15	32	L35	INE
LCD-CE	L82	16	31	L34	Dclk
LCD-CD	L83	17	30	L33	Aclk
PLSIN1	L84	18	29	L95	RTS
	L85	19	28	L94	Rx1
PLSO1	L86	20	27	L93	Tx1
		21	26	L92	CT0
	Reset-in	22	25	L91	Rx0
	GND	23	24	L90	Tx0

Leere Seite

Freie Tabelle BASIC-Tiger®-CAN-Modul als Kopiervorlage.
Die B-Reihe ist nicht aufgeführt, da die meisten Pins unbelegt sind und die belegten Pins keine Variable Funktion zulassen: Projekt:

Funktion	Pin-Bez.	Pin-Nr	Pin-Nr	Pin-Bez.	Funktion
	res.	1	46	VCC	
	L60	2	45	Batt.	
	L61	3	44	AGND	
	L62	4	43	Vref	
	L63	5	42	An3	
	L64	6	41	An2	
	L65	7	40	An1	
	L66	8	39	An0	
	L67	9	38	Alarm	
	L70	10	37	L41/PC	
	L71	11	36	L40	
	L72	12	35	L42	
	L73	13	34	L37	
	L80	14	33	L36	
	L81	15	32	L35	
	L82	16	31	L34	
	L83	17	30	L33	
	L84	18	29	L95	
	L85	19	28	L94	
	L86	20	27	L93	
		21	26	L92	
	Reset-in	22	25	L91	
	GND	23	24	L90	

7