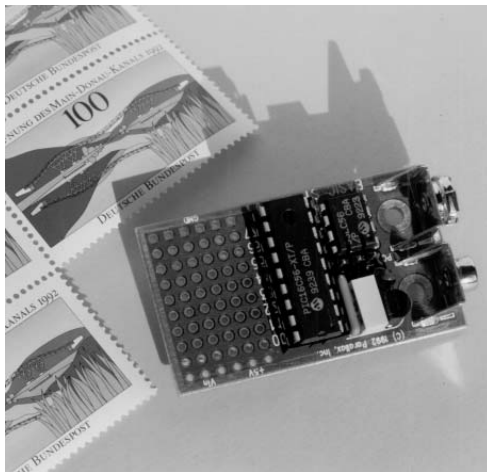


BASIC-Briefmarke®

BSI-2002-I



BASIC-Briefmarke®

BSI-2002-I

Benutzer-Handbuch

Copyright © Parallax, Inc.
© Wilke Technology GmbH

Version 1.1

Dieses Handbuch, sowie die Hard- und Software, die es beschreibt, ist urheberrechtlich geschützt und darf ohne ausdrückliche schriftliche Genehmigung von Wilke Technology GmbH in keiner Weise vervielfältigt, übersetzt oder in eine andere Darstellungsform gebracht werden.

Warenzeichen Diejenigen Bezeichnungen in dieser Publikation von Erzeugnissen und Verfahren, die zugleich Warenzeichen sind, wurden nicht besonders kenntlich gemacht. Solche Namen sind Warenzeichen der jeweiligen Warenzeichen-Inhaber. Aus dem Fehlen der Markierung ® kann nicht geschlossen werden, daß diese Bezeichnungen freie Warennamen sind.

PBASIC® ist ein Warenzeichen von Parallax, Inc.
BASIC-Briefmarke® ist ein Warenzeichen von Wilke Technology.

Hinweis Herausgeber, Übersetzer und Autoren dieser Publikation haben mit größter Sorgfalt die Texte, Abbildungen und Programme erarbeitet. Dennoch können Fehler nicht völlig ausgeschlossen werden. Wilke Technology übernimmt daher weder eine Garantie noch eine juristische Verantwortung oder Haftung für Folgen, die auf fehlerhafte Angaben zurückgehen. Mitteilungen über eventuelle Fehler werden jederzeit gerne entgegengenommen.

Alle Rechte vorbehalten.

Inhaltsverzeichnis

Bedienung des Editors	7
Starten des Editors	7
Programm-Format	7
Programme eingeben und editieren	11
Editor Funktions-Tasten	11
Starten Ihres Programms	13
Programm von Disk laden	13
Programm auf Disk speichern	13
Ausschneiden, Kopieren und Einfügen	14
Suchen & Ersetzen	15
I/O Port & Variablen-Speicherplatz	16
Übersicht Befehlssatz	18
BASIC Befehle	21
BRANCH	21
BUTTON	22
DEBUG	24
EEPROM	25
END	26
FOR...NEXT	27
GOSUB	29
GOTO	30
HIGH	31
IF...THEN	32
INPUT	33
LET	34
LOOKDOWN	35
LOOKUP	36
LOW	37
NAP	38
OUTPUT	39
PAUSE	40
POT	41
PULSIN	44
PULSOUT	45
PWM	46
RANDOM	47

Inhaltsverzeichnis

READ _____	48
RETURN _____	49
REVERSE _____	50
SERIN _____	51
SEROUT _____	54
SLEEP _____	55
SOUND _____	56
TOGGLE _____	57
WRITE _____	58
Literatur-Hinweise, Bezug _____	59
Aktuelle Applikations-Berichte _____	59
Bezug _____	59

Bedienung des Editors

Starten des Editors

Starten Sie die Editor-Software, indem Sie am DOS-Prompt eingeben:

STAMP

Angenommen Sie sind im richtigen Verzeichnis, wird die Software nach einigen Sekunden starten. Der Editor-Bildschirm ist dunkelblau, mit einer Menüzeile am oberen Bildschirmrand. Außer dieser Zeile kann der ganze Bildschirm zur Eingabe von BASIC Programmen verwendet werden.

Programm-Format

Es gibt einige Einschränkungen beim Eingeben von Programmen. Sie sollten die Regeln für die Eingabe von Konstanten, Labels und Kommentaren kennen, wie auf den folgenden Seiten beschrieben:

- **Konstanten:** konstante Werte können auf 4 Arten deklariert werden: dezimal, hex, binär und ASCII.

Hex-Zahlen werden durch ein vorgestelltes Dollar-Zeichen (\$) gekennzeichnet, Binär-Zahlen durch ein Prozent-Zeichen (%) und ASCII-Werte werden in Anführungszeichen eingeschlossen ("). Wird keine spezielle Kennung angegeben, nimmt der Editor einen Dezimalwert an. Einige Beispiele:

100	'Dezimal
\$64	'Hex
%01100100	'Binär
"A"	'ASCII "A" (65)
"Hello"	'ASCII "H", "e", "l", "l", "o"

Bedienung des Editors

- **Adress-Label:** der Editor verwendet Labels um auf Adressen (Positionen) in Ihrem Programm zu verweisen - im Unterschied zu einigen BASIC-Versionen, die Zeilennummern verwenden.

Allgemein können Labelnamen jede Kombination von Buchstaben, Zahlen und Unterstrichen (_) sein, aber das erste Zeichen des Namens darf keine Ziffer sein. Auch dürfen Labels keine reservierten Worte beinhalten, wie Befehls-Namen (serin, toggle, goto, etc.) und Variablen-Namen (port, w2, b13, etc.)

Bei erstmaliger Verwendung müssen die Labels mit einem Doppelpunkt enden (:). Werden Sie irgendwo im Programm aufgerufen, werden Sie ohne den Doppelpunkt bezeichnet. Das folgende Beispiel zeigt, wie mit einem Label auf eine Adresse verwiesen wird:

```
loop:    toggle 0                'Toggle pin 0

        for b0 = 1 to 10
        toggle 1                'Toggle pin 1 (10 x)
        next

        goto loop              'Wiederhole den Vorgang
```

- **Wert-Label:** Sie können sich mit Labels auch auf Variablen und Konstanten beziehen. Wert-Labels haben die gleichen Syntax-Regeln wie Adress-Labels, aber Wert-Labels enden niemals mit einem Doppelpunkt, und sie müssen mit der "Symbol"-Direktive definiert werden. Das folgende Beispiel zeigt einige Wert-Labels:

```
        symbol start = 1        'Definiere 2 Konstanten
        symbol end = 10        'Labels

        symbol count = B0      'Definiere ein Variablen-
                               'Label

loop:    for count = start to end
        toggle 1
        next
```

Bedienung des Editors

- **Kommentare:** Das Einfügen von Kommentaren macht Ihre Programme übersichtlicher.

Kommentare beginnen mit einem Hochkomma (!) und gehen bis zum Ende der Zeile. Wahlweise können Sie einen Kommentar mit der REM-Anweisung kennzeichnen. Ein Beispiel:

```
symbol relay = 3           'Definiere Konstante

REM dies ist die Hauptschleife

main:                      port = %0000000011111111
                           length = length + 10
                           gosub sub
                           goto main

sub:                        pulsout relay,length :
toggle 0 : return
```

- **Generelles Format:**

Mehrere Befehle und Label in einer Zeile werden durch Doppelpunkte (:) voneinander getrennt.

```
dirs = 255
for b2 = 0 to 100 : pins = b2 : next
```

Bedienung des Editors

- **Mathematische Operatoren:** die folgenden Operatoren können in mathematischen Ausdrücken verwendet werden:

+	addieren
-	subtrahieren
*	multiplizieren (gibt low word des Ergebnis zurück)
**	multiplizieren (gibt high word des Ergebnis zurück)
/	dividieren (gibt Quotient zurück)
//	dividieren (gibt Rest zurück)
MIN	behält Variable größer oder gleich Wert
MAX	behält Variable kleiner oder gleich Wert
&	logisch AND
	logisch OR
^	logisch XOR
&/	logisch AND NOT
/	logisch OR NOT
^/	logisch XOR NOT

Einige Beispiele:

```
count = count + 1      'count hochzählen
timer = timer * 2      'timer mal 2
b2 = b2 / 8            'b2 durch 8 teilen
w3 = w3 & 255          'low-byte von w3
```

Wichtiger Hinweis:

Mathematische Ausdrücke werden von der BASIC-Briefmarke streng von links nach rechts verarbeitet. Dies ist wichtig zu wissen, denn Sie könnten andere Ergebnisse erwarten. Zum Beispiel wird der Ausdruck $2 + 3 \times 4$ normalerweise als $2 + (3 \times 4)$ gelöst, da die Multiplikation eine höhere Priorität hat als die Addition. Das Ergebnis wäre 14.

Da die BASIC-Briefmarke Ausdrücke von links nach rechts auswertet, wird der Ausdruck als $(2 + 3) \times 4$ gelöst, mit dem Ergebnis 20.

Bedienung des Editors

Programme eingeben und editieren

Wie auf den vorherigen Seiten beschrieben gibt es einige Regeln für die Benutzung von Konstanten, Labels und Kommentaren. Größtenteils können Sie Ihre Programme aber nach eigenem Wunsch gestalten.

Wir haben versucht die Bedienung des Editors so intuitiv wie möglich zu machen: drücken Sie <CrsrOben> um eine Zeile nach oben zu kommen, drücken Sie <Shift-CrsrRechts> um ein Zeichen zu markieren usw.

Die meisten Funktionen des Editors sind einfach anzuwenden. Mit einzelнем Tastendruck können Sie folgende Funktionen ausführen:

- Laden, Speichern, und Ausführen von Programmen.
- Bewegen des Cursor in Schritten von Zeichen, Worten, Zeilen, Seiten oder zum Beginn oder Ende einer Datei.
- Markieren von Text in Blöcken von Zeichen, Worten, Zeilen, Seiten oder bis zum Beginn oder Ende einer Datei.
- Ausschneiden, Kopieren und Einfügen von markiertem Text.
- Suchen und/oder Ersetzen von Text.

Editor Funktions-Tasten

Die folgende Liste zeigt die Tasten, die eine Funktion auslösen:

Alt-R	Programm auf Briefmarke starten (<i>erst von Bildschirm downloaden, dann starten</i>)
Alt-L	Programm von Disk laden
Alt-S	Programm auf Disk speichern
Alt-Q	Editor verlassen/Rückkehr nach DOS
Enter	Information eingeben und eine Zeile weiter
Tab	Wie Enter
Cursor links	Ein Zeichen nach links
Cursor rechts	Ein Zeichen nach rechts

Bedienung des Editors

Cursor rauf	Eine Zeile nach oben
Cursor runter	Eine Zeile nach unten
Ctrl-Links	Ein Wort nach links
Ctrl-Rechts	Ein Wort nach rechts
Home/Pos1	Zum Anfang der Zeile
End	Zum Ende der Zeile
Page Up	Eine Seite zurück
Page Down	Eine Seite nach vorne
Ctrl-Page Up	Zum Anfang der Datei
Ctrl-Page Down	Zum Ende der Datei
Shift-Left	Ein Zeichen nach links markieren
Shift-Right	Ein Zeichen nach rechts markieren
Shift-Up	Eine Zeile nach oben markieren
Shift-Down	Eine Zeile nach unten markieren
Shift-Ctrl-Left	Ein Wort nach links markieren
Shift-Ctrl-Right	Ein Wort nach rechts markieren
Shift-Home	Bis Zeilenanfang markieren
Shift-End	Bis Zeilenende markieren
Shift-Page Up	Eine Seite rückwärts markieren
Shift-Page Down	Eine Seite vorwärts markieren
Shift-Ctrl-Page Up	Bis Datei-Anfang markieren
Shift-Ctrl-Page Down	Bis Datei-Ende markieren
Shift-Einfügen	Wort unter Cursor markieren
ESC	Markierung löschen
Backspace	Zeichen vor Cursor löschen
Delete	Zeichen unter Cursor löschen
Shift-Backspace	Von Zeilenanfang bis vor Cursor löschen
Shift-Delete	Bis Zeilenende löschen
Ctrl-Backspace	Zeile löschen
Alt-X	Markierten Text ausschneiden
Alt-C	Markierten Text kopieren
Alt-V	Markierten Text einfügen (an Cursorposition)
Alt-F	Suche Text
Alt-N	Suche nächstes Auftreten des Textes
Alt-P	Potentiometer-Skalierung einstellen (siehe POT -Befehl für mehr Information)

Bedienung des Editors

Starten Ihres Programms

Um das Programm im Editor zu starten, drücken Sie ALT-R. Der Editor durchsucht die parallelen Ports nach einer BASIC Briefmarke. Findet er eine, sendet er das Programm ab und startet es. Findet der Editor keine Briefmarke, zeigt er eine Fehlermeldung.

Angenommen Sie haben die Briefmarke korrekt an Ihren PC angeschlossen, zeigt der Editor einen Balken mit dem aktuellen Stand des Download. Dieser dauert normalerweise nur einige Sekunden, der Balken füllt sich also schnell.

Sie werden feststellen, daß sich ein Teil des Balkens weiß füllt, der Rest rot. Diese Farben zeigen die Speicherbelegung des EEPROM an. Weiß zeigt den verfügbaren Speicherplatz, Rot den vom Programm belegten Speicherplatz. Dadurch können Sie einfach feststellen, wieviel Speicher des EEPROM für zusätzliche Befehle oder Daten frei bleibt.

Wenn der Download beendet ist, wird Ihr Programm in der Briefmarke automatisch gestartet. Wenn Sie in Ihrem Programm die DEBUG-Direktive verwendet haben, zeigt es seine Daten beim Auftreten im Programm an.

Um den Balken vom Bildschirm zu entfernen und das Editieren fortzusetzen, drücken Sie eine beliebige Taste.

Programm von Disk laden

Um ein BASIC Programm von einer Disk zu laden, drücken Sie Alt-L. Eine kleine Box erscheint und fragt nach dem Dateinamen. Haben Sie den Namen korrekt eingegeben, wird das Programm in den Editor geladen. Andernfalls wird eine Fehlermeldung angezeigt.

Wollen Sie das Programm nicht laden, drücken Sie ESC.

Programm auf Disk speichern

Um ein Programm auf einer Disk zu speichern, drücken Sie Alt-S. Eine kleine Box erscheint und fragt nach einem Dateinamen. Nachdem der Dateiname eingegeben ist, speichert der Editor Ihr Programm.

Bedienung des Editors

Ausschneiden, Kopieren und Einfügen

Wie die meisten Textverarbeitungen kann der Editor Text leicht ausschneiden, kopieren und einfügen. Wollen Sie große Änderungen in Ihrem Programm vornehmen oder hat Ihr Programm sich wiederholende Sequenzen, können Ihnen diese Funktionen viel Zeit sparen.

Die Aufgabe der Ausschneide-, Kopier- und Einfügeroutinen ist es, markierten Text auszuschneiden oder in das *Clipboard* zu kopieren (das Clipboard ist ein Speicherbereich außerhalb des Editors). Text im Clipboard kann später irgendwo in Ihrem Programm eingefügt werden. Sowohl Ausschneiden wie auch Kopieren kopiert Text in das Clipboard, aber Ausschneiden entfernt den Text auch von seiner aktuellen Position.

Beachten Sie, daß Ausschneiden und Löschen von Text unterschiedlich sind. In beiden Fällen wird er aus seiner ursprünglichen Position gelöscht, aber Ausschneiden speichert ihn im Clipboard - Löschen entfernt ihn vollständig.

Als Übung schneiden wir einen Textbereich aus und fügen ihn an anderer Stelle wieder ein. Die folgenden Schritte helfen Ihnen dabei:

- Erstens: Markieren Sie etwas Text. Beispielsweise vom Cursor bis zum Ende der Zeile. Drücken Sie **Shift-End** (alles von Cursor bis zum Ende der Zeile wird markiert).
- Zweitens: Drücken Sie **Alt-X** (ausschneiden). Der Text sollte verschwinden.
- Drittens: Bewegen Sie den Cursor an eine andere Stelle und drücken Sie **Alt-V** (einfügen). Der Text sollte an der Cursor-Position erscheinen, den nachfolgenden Text nach hinten verschiebend.

Der zweite Schritt könnte ersetzt werden durch Kopieren (Alt-C). In diesem Fall würde der Text an seiner ursprünglichen Position erhalten bleiben.

Bedienung des Editors

Suchen & Ersetzen

Der Editor hat eine Funktion, die Ihnen das Suchen und/oder Ersetzen von Text erlaubt. In vielen Fällen kann diese Funktion sehr nützlich sein. Zum Beispiel wollen Sie im ganzen Programm einen Variablen-Namen ändern. Manuell würde das viel Zeit kosten, mit Suchen/Ersetzen dauert es nur Sekunden.

Um das Such-Kriterium zu setzen, drücken Sie **Alt-F** (suchen). Eine Eingabe-Box fragt nach dem Such-String und optional nach einem Ersetzungs-String. Führen Sie folgende Schritte durch:

- **Geben Sie den Such-String ein.** Soll dieser die *Tab* or *Return* Tasten enthalten, geben Sie diese durch *Ctrl-Tab* oder *Ctrl-Return* ein. "•" erscheint für jeden Tab, "↓" für jedes Return.
- **Geben Sie den Ersetzungs-String ein (falls nötig).** Wenn Sie diesen eingeben, wird er in das Clipboard kopiert. Während der Suche haben Sie die Möglichkeit, einzelne Auftreten des Such-Strings mit dem Ersetzungs-String zu ersetzen.

Wollen Sie nur suchen, geben Sie beim Ersetzungs-String nur *Enter* ein.

- Der Editor entfernt die Eingabe-Box und zeigt das erste Auftreten des Such-Strings an.

Um den markierten String zu ersetzen, drücken Sie **Alt-V** (ersetzen).

Um das nächste Auftreten des Such-Strings zu finden, drücken Sie **Alt-N**.

I/O Port & Variablen-Speicherplatz

Die BASIC-Briefmarke hat 16 Byte des RAM als Variablen-Speicherplatz zur Verfügung. Zwei Bytes werden für I/O benötigt (1 für Pins, 1 für Direction-Control), verbleiben 14 für Daten. Die 14 Daten-Byte können als Word-Variablen (W0-W6) und Byte-Variablen (B0-B13) verwendet werden. Zusätzlich können B0 and B1 (W0) als Bit-Variablen (Bit0-Bit15) verwendet werden. Die Aufteilung des Variablen-Speichers:

	Words	Bytes	Bits	Alternative Bit-Namen
0000 0000 0000 0000	Port	Pins Dirs	Pin0-Pin7 Dir0-Dir7	Pins.0-Pins.7, Port.0-Port.7 Dirs.0-Dirs.7, Port.8-Port.15
0000 0000 0000 0000	W0	B0 B1	Bit0-Bit7 Bit8-Bit15	B0.0-B0.7, W0.0-W0.7 B1.0-B1.7, W0.8-W0.15
0000 0000 0000 0000	W1	B2 B3		
0000 0000 0000 0000	W2	B4 B5		
0000 0000 0000 0000	W3	B6 B7		
0000 0000 0000 0000	W4	B8 B9		
0000 0000 0000 0000	W5	B10 B11		
0000 0000 0000 0000	W6	B12 B13		

Die BASIC-Sprache erlaubt eine ziemliche Flexibilität im Benennen von Variablen und I/O-Pins. Abhängig von Ihren Anforderungen können Sie Variablen-Speicher und I/O-Pins als Bytes oder 16-Bit Worte verwenden. Zusätzlich können die I/O-Pins und ersten beiden Variablen-Byte Bit-für-Bit verwendet werden. In vielen Fällen kann ein einzelnes Bit völlig ausreichen, z.B. für Flags oder zum Lesen eines einzelnen Input.

I/O Port & Variablen-Speicherplatz

Port ist das I/O-Wort, welches aus 2 Bytes, **Pins** and **Dirs**, besteht:

Pins und **Pin0-Pin7** sind die I/O-Port-Pins. Werden diese Variablen gelesen, werden die I/O-Pins direkt gelesen. Wird auf diese Variablen geschrieben, wird auch auf das zugehörige RAM geschrieben, welches dann vor jedem Befehl an den I/O-Port gesendet wird.

Dirs und **Dir0-Dir7** sind die I/O-Port Richtungs-Bits. Eine "0" in einem dieser Bits macht den zugehörigen I/O-Pin zu einem Input-Pin, eine "1" zu einem Output-Pin. Dieses Daten-Byte wird vor jedem Befehl an die I/O-Port Richtungs-Register gesendet.

Beim Schreiben eines BASIC Programms verwenden Sie die oben genannten Symbole zum Lesen und Beschreiben der 8 I/O-Pins.

Normalerweise starten Sie Ihr Programm mit der Definition welche Pins Input und welche Output sind. "dirs = %00001111" würde Bits 0-3 als Output, Bits 4-7 als Input festlegen (von rechts nach links).

Nach dieser Definition können Sie die Pins auslesen und beschreiben. Der Befehl "pins = 7" setzt die Bits 0-2 high. Der Befehl "b2 = pins" würde alle 8 Pins in die Byte-Variable *b2* einlesen.

Pins können bit-weise adressiert werden, was für einige Applikationen von Vorteil ist. Der Befehl "if pin3 = 1 then start" liest den I/O-Pin 3 und springt nach "start" (Adresse) wenn der Pin high (1) war.

Die restlichen Variablen (W0-W6) können für beliebige Speicherzwecke verwendet werden, mit einer wichtigen Ausnahme:

W6 wird als Stack benötigt wenn GOSUB's ausgeführt werden.

Die Editor-Software erkennt die Variablen-Namen auf der vorherigen Seite. Wollen Sie andere Namen verwenden, können Sie Ihr Programm mit Befehlen zur Definition neuer Namen beginnen:

```
symbol switch = pin0      'I/O-Pin-Label
symbol flag = bit0       'Bit-Label
symbol count = b2       'Byte-Label
```

Übersicht Befehlssatz

SPRUNGBEFEHLE

IF...THEN	<i>Vergleichen und bedarfsweise springen.</i>
BRANCH	<i>Springen auf durch Offset spezifizierte Adresse.</i>
GOTO	<i>Springen auf Adresse.</i>
GOSUB	<i>Springen auf Unterprogramm bei Adresse. Bis zu 16 GOSUB's sind erlaubt.</i>
RETURN	<i>Rücksprung aus Unterprogramm.</i>

SCHLEIFEN

FOR...NEXT	<i>Einrichten einer FOR-NEXT Schleife.</i>
------------	--

NUMERIK

{LET}	<i>Variablen-Manipulation, wie $A=5$, $B=A+2$, usw. Erlaubte Operationen sind addieren, subtrahieren, multiplizieren, dividieren, max. limit, min. limit, und logische Operationen AND, OR, XOR, AND NOT, OR NOT, und XOR NOT.</i>
LOOKUP	<i>Speichert den durch Offset angegebenen Wert einer Tabelle in einer Variablen.</i>
LOOKDOWN	<i>Findet Position des Zielwerts in einer Tabelle und speichert sie in einer Variable.</i>
RANDOM	<i>Generieren einer Pseudo-Zufallszahl.</i>

DIGITALER I/O

OUTPUT	<i>Macht Pin zu Output.</i>
LOW	<i>Macht Pin zu Output (low).</i>
HIGH	<i>Macht Pin zu Output (high).</i>
TOGGLE	<i>Macht Pin zu Output und wechselt Status.</i>
PULSOUT	<i>Ausgabe eines Impuls durch zeitliche Invertierung eines Pin.</i>

Übersicht Befehlssatz

INPUT	<i>Macht Pin zu Input</i>
PULSIN	<i>Mißt einen Input-Impuls.</i>
REVERSE	<i>Macht Input- zu Output-Pin und umgekehrt.</i>
BUTTON	<i>Entprelle Taste, führe Auto-Repeat durch, und springe zu Adresse falls Taste im Ziel-Zustand.</i>

SERIELLER I/O

SERIN	<i>Serieller Input mit optionalen Triggern und Variablen für Speicherung der empfangen Daten. Sind Trigger gesetzt, wartet der Befehl bis er sie empfängt bevor er Variablen besetzt oder mit dem nächsten Befehl fortfährt. Baudraten von 300, 600, 1200 und 2400 sind gestattet. Voraussetzung: No Parity, 8 Datenbit und 1 Stopbit.</i>
SEROUT	<i>Daten seriell senden. Daten werden mit 300, 600, 1200 oder 2400 Baud gesendet (No Parity, 8 Datenbit und 1 Stopbit).</i>

ANALOGER I/O

PWM	<i>Output PWM, dann Pin wieder auf Input setzen. Kann für Ausgabe analoger Spannungen (0-5V) mit Kondensator und Widerstand verwendet werden.</i>
POT	<i>Liest ein 5-50K Potentiometer und skaliert Ergebnis.</i>

SOUND

SOUND	<i>Erzeugt Töne. Ton 0 ist Ruhe, Töne 1-127 sind aufsteigende klare Töne. Töne im Bereich 128-255 sind weißes Rauschen.</i>
-------	---

EEPROM ZUGRIFF

EEPROM	<i>Speichert Daten im EEPROM vor dem downloaden eines BASIC Programms.</i>
--------	--

Übersicht Befehlssatz

READ *Liest EEPROM Byte in eine Variable.*

WRITE *Schreibe Byte ins EEPROM.*

ZEIT

PAUSE *Unterbreche Ausführung für 0–65536 Millisekunden.*

POWER KONTROLLE

NAP *"Schlummer-Modus". Strombedarf ist kurzzeitig reduziert.*

SLEEP *"Schlaf-Modus". Strombedarf ist dauerhaft reduziert auf ca. 20 μ A.*

END *"Schlaf-Modus" bis Stromversorgung AUS-EIN geschaltet wird oder eine Verbindung zum PC aufgenommen wird. Der Strombedarf ist reduziert auf ca. 20 μ A.*

PROGRAMM DEBUGGEN

DEBUG *Sendet während des Programm-Laufs Variablen zur Ansicht an den PC. Der Entwickler kann an Hand der Variablenwerte die richtige Programmfunktion überprüfen.*

BASIC Befehle

BRANCH

BRANCH Offset,(Adresse0,Adresse1,...AdresseN)

Sprung auf die in Offset angegebene Adresse und weitere Ausführung dort.

- **Offset** ist eine Variable/Konstante welche die Adresse bestimmt, auf die gesprungen werden soll (0-N).
- **Adressen** sind Labels die angeben wohin gesprungen werden soll.

Beispiel-Programm:

```
abc:  serin 0,n2400,("code"),b2 'Monitor serial
input and
                                     'wait for
"c","o","d","e".
                                     'Store next byte in b2.

      BRANCH b2,(xyz,iou,irs) 'If b2=0, gehe zu xyz;
                                     'If b2=1, gehe zu iou;
                                     'If b2=2, gehe zu irs.

      goto abc
                                     'Gehe zu abc falls b2
                                     'nicht in Bereich 0-2.

xyz:  ...
iou:  ...
irs:  ...
```

BASIC Befehle

BUTTON

BUTTON Pin,Downstate,Delay,Rate,Bytevar,Zielstatus,Adresse

entprelle Tasten-Eingabe, führe Auto-Wiederholung durch und springe, wenn die Taste im Ziel-Zustand ist. Die Tasten-Schaltung kann aktiv-low oder aktiv-high sein, wie im Schaltbild auf der nächsten Seite zu sehen.

- **Pin** ist eine Variable/Konstante (0-7), die den benutzten I/O-Pin spezifiziert.
- **Downstate** ist eine Variable/Konstante (0 oder 1), die angibt, welcher logische Status gelesen wird, wenn die Taste gedrückt ist.
- **Delay** ist eine Variable/Konstante (0-255), die angibt wie lange die Taste gedrückt sein muß, bevor die Auto-Wiederholung startet. Die Verzögerung wird in Zyklen der Tasten-Routine gemessen.

Delay hat zwei spezielle Einstellungen: 0 und 255. Falls 0, gibt die Routine den Tasten-Status zurück ohne Entprellung oder Auto-Wiederholung. Falls 255, wird ein Entprellung, aber keine Auto-Wiederholung durchgeführt.

- **Rate** ist eine Variable/Konstante (0-255), die die Auto-Wiederholungsrate angibt. Die Rate wird angegeben in Zyklen der Tasten-Routine.
- **Bytevar** ist der Arbeitsspeicher für BUTTON. Er muß vor der ersten Verwendung von BUTTON auf 0 gesetzt werden.
- **Zielstatus** ist eine Variable/Konstante (0 oder 1), welche angibt, bei welchem Tasten-Status der Sprung ausgeführt wird. (0=nicht gedrückt, 1=gedrückt)
- **Adresse** ist ein Label, das angibt wohin gesprungen wird, wenn die Taste im Ziel-Zustand ist.

Fortsetzung nächste Seite

BASIC Befehle

BUTTON *(Fortsetzung)*

Beispiel-Programm:

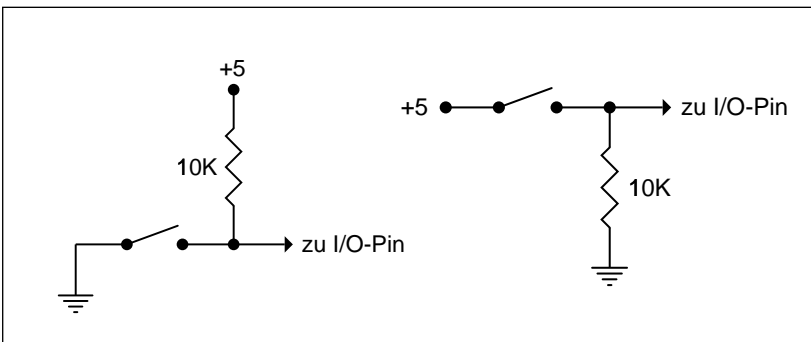
```
b2=0                                'Lösche b2 vor Verwen-
                                    'dung in der Tasten-
                                    'Routine.

loop:  BUTTON 3,0,90,10,b2,0,abc 'Lies Taste auf Pin 3.
                                    'Taste ist low (0) wenn
                                    'gedrückt. Auto-Repeat
                                    'startet nach 50 Routi-
                                    'nen-Zyklen.
                                    'B2 enthält die Anzahl
                                    'der Routinen-Zyklen.
                                    'Falls Taste nicht
                                    'gedrückt, Sprung zu abc.

                                    pulsout 5,1000          'Puls Pin 5 für 0.01 sec.

abc:  pause 10                       'Pause für 10 ms. Erstellt
                                    'ein Auto-Repeat-Delay
                                    'von 1000 ms (10x100)
                                    'und eine Repeat-Rate von
                                    '100 ms (10x10).

                                    goto loop                'Zurück zu BUTTON-Befehl.
```



BASIC Befehle

DEBUG

DEBUG

Sendet Variablen-Inhalte an den PC zur Ansicht. DEBUG arbeitet ähnlich wie PRINT in vielen BASIC-Dialekten. Statt Text auf dem Bildschirm auszugeben, sendet DEBUG Text und Variablen zurück an den PC.

Die folgenden Regeln beziehen sich auf den DEBUG Befehl:

- Die genannten Variablen generieren den Ausdruck "Variable = Wert" auf dem PC.
- Die als *#variable* genannten Variablen generieren Ihren Wert (ohne den "Variable =" Text).
- Text in Anführungszeichen erscheint am PC wie eingegeben.
- Variablen werden normalerweise dezimal angezeigt. Um eine Variable in Hex anzuzeigen, stellen Sie ein "\$" vor ihren Namen. Um eine Variable binär anzuzeigen, stellen Sie ein "%" vor ihren Namen.
- "cr" kann für einen Zeilenumbruch am PC eingefügt werden.
- "cls" kann für Bildschirm löschen am PC eingefügt werden.
- Anzuzeigende Variablen müssen durch Kommas getrennt werden.

Beispiele:

```
DEBUG b2           'Print "b2 = " + Wert von b2
DEBUG #b2          'Print Wert von b2
DEBUG "Wert b2=",b2 'Print "Wert b2=" + Wert b2
DEBUG #%b2         'Print Wert von b2 (binär)
DEBUG "Werte:",b2,b3,cr 'Print "Werte" + Wert b2 +
                        'Wert b3 + <CR>
```

BASIC Befehle

EEPROM

EEPROM {Position},{Data,Data,...}

Speichert Werte im EEPROM vor dem Downloaden des BASIC Programms. Dies ist hilfreich, um das EEPROM mit Werten vorzubeseetzen, die von Ihrem Programm verwendet werden.

Die EEPROM Direktive wird vor jedem Download ausgeführt, wird selbst aber nicht an die Briefmarke gesendet. Daher verbraucht Sie keinen Speicherplatz auf der Briefmarke.

- **Position** ist eine optionale Variable/Konstante (0-255), die die Startposition der Daten im EEPROM festlegt. Wird keine Position angegeben, werden die Daten an die nächste freie Position geschrieben.
- **Data** sind Variablen/Konstanten (0-255), die sequentiell im EEPROM gespeichert werden.

Beispiel-Program:

```
EEPROM 0,(5,23,17,158,2)  'Vorbeseetzen des EEPROM  
                          'mit später verwendeten  
                          'Daten.  
.  
.  
.
```

BASIC Befehle

END

END

Wechselt auf unbestimmte Zeit in den "Sleep Mode". Die Briefmarke wacht wieder auf wenn die Stromversorgung unterbrochen und wieder eingeschaltet wird oder der PC die Verbindung herstellt. Der Stromverbrauch ist im Sleep-Mode reduziert auf ca. 20 μA , angenommen an den Ausgangs-Pins liegt keine Belastung an.

END hat keine Parameter.

BASIC Befehle

FOR...NEXT

FOR Variable = Start TO End {STEP {-}Inkrement}

.
. .
. . .

NEXT {Variable}

Einrichten einer FOR-NEXT Schleife. *Variable* wird auf den Wert *Start* gesetzt. Die Anweisungen zwischen dem FOR und dem NEXT-Befehl werden ausgeführt. Die *Variable* wird um den Wert *Inkrement* erhöht oder vermindert (ist kein Inkrement angegeben, wird die *Variable* um 1 erhöht). Falls die *Variable* den Wert *End* noch nicht erreicht oder überschritten hat, werden die Befehle zwischen FOR und NEXT nochmals ausgeführt. Hat die *Variable* den Wert *End* erreicht oder überschritten, wird die Ausführung hinter dem NEXT-Befehl fortgesetzt. Die Schleife wird mindestens einmal ausgeführt, unabhängig von den Werten für *Start* und *End*.

Ihr Programm kann so viele For...Next-Schleifen enthalten wie nötig, aber nur bis Verschachtelungstiefe 8 (Im Programm können nicht mehr als 8 Schleifen ineinander verschachtelt werden).

- **Variable** ist eine Bit, Byte oder Wort-Variable als interner Zähler. *Start* und *End* sind eingeschränkt durch die Art der *Variable* (Bit-Variablen können von 0 bis 1 zählen, Byte-Variablen von 0 bis 255, und Wort-Variablen von 0 bis 65535).
- **Start** ist eine Variable/Konstante, die den Anfangswert von *Variable* festlegt.
- **End** ist eine Variable/Konstante, die den Endwert von *Variable* festlegt.
- **Inkrement** ist ein optionaler Wert, der den Zähler erhöht oder vermindert (falls negativ). Wird kein Inkrement-Wert angegeben, wird der Zähler um 1 erhöht.

Fortsetzung nächste Seite...

BASIC Befehle

FOR...NEXT *(Fortsetzung)*

Beispiel Programm:

```
FOR b2 = 0 TO 255           'For-Next-Schleife, die
                             'von 0 bis 255 zählt.

pins = b2                   'Ausgabe von 8-Bit-Wert
                             'auf die Pins 0-7.

NEXT
```

BASIC Befehle

GOSUB

GOSUB Adresse

Speichert die Adresse des auf GOSUB folgenden Befehls und springt dann nach *Adresse*, wo die Ausführung fortgesetzt wird. Ihr Programm kann die Ausführung dort fortsetzen, von wo aus der Sprung erfolgte, indem Sie ein RETURN an das Ende der gerufenen Routine schreiben.

Es sind bis zu 16 GOSUB's in einem Programm erlaubt.

- **Adresse** ist ein Label das angibt, wohin gesprungen wird.

Beispiel Programm:

```
        for b4 = 0 to 10
        GOSUB abc                'Speichere Rücksprung-
                                'Adresse und springe
                                'nach abc
        next
abc:    pulsout 0,b4            'Ausgabe Puls auf Pin 0.
                                'Puls-Länge ist b4 x 10µs.

        toggle 1                'Invertiere Pin 1

        return                  'Rücksprung
```

BASIC Befehle

GOTO

GOTO Adresse

Springe nach *Adresse*, wo die Ausführung fortgesetzt wird.

- **Adresse** ist ein Label das angibt, wohin gesprungen wird.

Beispiel Programm:

```
abc:    pulsout 0,100           'Generiert einen 1000µs
                                     'Puls auf Pin 0

        GOTO abc              'Wiederhole dies
```

BASIC Befehle

HIGH

HIGH Pin

Setze den angegebenen Ausgabe-Pin auf High. Ist der Pin als Eingabe-Pin programmiert, wird er zum Ausgabe-Pin.

- **Pin** ist eine Variable/Konstante (0-7) die den zu setzenden I/O-Pin festlegt.

Beispiel Programm:

```
ibm:      HIGH 3          'Ausgabe-Pin 3=High
          low 2          'Ausgabe-Pin 2=Low
          HIGH 2        'Ausgabe-Pin 2=High
          low 3         'Ausgabe-Pin 3=Low

          goto ibm      'Wiederhole dies
```

BASIC Befehle

IF...THEN

IF Variable ?? Wert {AND/OR Variable ?? Wert...} THEN Adresse

Vergleiche Variable(n) mit Wert(en) und springe nach Adresse, falls Ergebnis positiv (wahr).

Anders als der IF...THEN Befehl in anderen BASIC-Dialekten kann die auf THEN folgende Adresse nicht durch andere Befehle ersetzt werden, wie "IF A=10 THEN READ 200,B". Es darf nur ein Adress-Label auf THEN folgen.

- **??** ist einer der folgenden Operatoren: =, <>, >, <, >=, <=.
- **Variable** ist eine Variable, die mit Wert(en) verglichen wird.
- **Wert** ist eine Variable/Konstante, mit der verglichen wird.
- **Adresse** ist ein Label das angibt, wohin gesprungen wird wenn die Vergleiche positiv (wahr) sind.

Beispiel Programm:

```
abc:      serin 0,n2400,b2          'Empfange serielles Byte
          IF b2 = 185 THEN xyz      'Falls 185 empfangen,
          high 2                    'springe nach xyz.
          goto abc                  'Ausgabe-Pin 2=High.
          goto abc                  'Springe nach abc.
xyz:      low 2                     'Ausgabe-Pin 2=Low.
          goto abc                  'Springe nach abc.
```

BASIC Befehle

INPUT

INPUT Pin

Setze den angegebenen Pin als Input-Pin. Dies schaltet alle Ausgabe-Treiber ab und erlaubt Ihrem Programm jeden Status zu lesen, der an diesem Pin anliegt.

- **Pin** ist eine Variable/Konstante (0-7) die angibt, welcher I/O-Pin angesprochen wird.

Beispiel Programm:

```
INPUT 5                                'Eingabe-Pin 5 =Input.
abc:  if pin5 = 1 then xyz              'Falls Pin 5=High,
                                           'springe nach xyz.
                                           goto abc                                'Springe nach abc.
xyz:  serout 3,n300,(65)                'Sende 65 (seriell).
nbc:  if pin5 = 1 then nbc              'Warte bis Pin 5=Low.
                                           goto abc
```

BASIC Befehle

LET

{LET} Variable = {-}Wert ?? Wert...

Weise einer Variable einen Wert zu und/oder manipulierte eine Variable. Alle internen Manipulationen werden auf Wort-Ebene ausgeführt (16 Bits).

Der Befehl "LET" ist optional. "A=10" ist beispielsweise identisch zu "LET A=10".

- ?? ist einer der folgenden Operatoren:

+	addieren
-	subtrahieren
*	multiplizieren (gibt Low-Wort des Ergebnis zurück)
**	multiplizieren (gibt High-Wort des Ergebnis zurück)
/	dividieren (gibt Quotient zurück)
//	dividieren (gibt Rest zurück)
MIN	behält Variable größer oder gleich Wert
MAX	behält Variable kleiner oder gleich Wert
&	logisch AND
	logisch OR
^	logisch XOR
&/	logisch AND NOT
/	logisch OR NOT
^/	logisch XOR NOT

- **Variable** wird ein Wert und/oder ein Ausdruck zugewiesen.
- **Wert(e)** ist eine Variable/Konstante, die die Variable beeinflusst.

Beispiel Program:

```
abc:      pot 0,100,b3          'Lies Potentiometer und
                                     'lege Ergebnis in b3.

          LET b3=b3/2          'Teile Ergebnis durch 2.

          b3=b3 max 100        'Begrenze Ergebnis auf
                                     '0-100."LET" ist nicht
                                     'notwendig.
```

BASIC Befehle

LOOKDOWN

LOOKDOWN Ziel,(Wert0,Wert1,...WertN),Variable

Durchsuche Werte nach dem Ziel-Wert. Falls *Ziel* einem der *Werte* entspricht, wird die Nummer dieses Wertes in *Variable* gespeichert. Sind beispielsweise die Werte 2, 13, 15, 28, 8 und der Zielwert ist 15, würde die Variable den Wert "2" enthalten, da "15" der dritte Wert in der Liste ist (der erste Wert ist #0, der zweite ist #1, usw.).

Wird keine Übereinstimmung gefunden, wird die Variable nicht verändert.

- **Ziel** ist die Variable/Konstante nach der Sie suchen.
- **Wert0, Wert1,...** ist eine Liste von Werten. Der Zielwert wird mit diesen Werten verglichen.
- **Variable** enthält das Ergebnis der Suche.

Beispiel Programm:

```
serin 0,n2400,b2          'Empfange serielles Byte.
LOOKDOWN b2,(65,88,93),b3 'Falls b2=65, b3=0;
                          'Falls b2=88, b3=1;
                          'Falls b2=93, b3=2;
.
.
.
```

BASIC Befehle

LOOKUP

LOOKUP *Offset*,(*Wert0*,*Wert1*,...*WertN*),*Variable*

Sieht nach einem Wert angegeben durch *Offset* und speichert ihn in *Variable*. Sind die Werte beispielsweise 2, 13, 15, 28, 8 und der *Offset* ist 1, würde die *Variable* den Wert "13" enthalten, da "13" der zweite Wert in der Liste ist (der erste Wert ist #0, der zweite ist #1, usw.).

Liegt der *Offset* oberhalb der Anzahl der gegebenen Werte, wird die *Variable* nicht verändert.

- **Offset** gibt an welcher Wert in die *Variable* kopiert wird.
- **Wert0, Wert1,...** ist eine Liste von Werten. Der *Offset*-Wert wird aus diesen Werten genommen.
- **Variable** enthält das Ergebnis des Befehls.

Beispiel Programm:

```
for b2 = 0 to 25  
  
LOOKUP b2,(65,66,67,..),b3 'Konvertiere Offset (0-  
                             '25) zu entsprechendem  
                             'ASCII-Zeichen (A-Z).  
  
next
```

BASIC Befehle

LOW

LOW Pin

Setze den angegebenen Ausgabe-Pin auf Low. Ist der Pin als Eingabe-Pin programmiert, wird er zum Ausgabe-Pin.

- **Pin** ist eine Variable/Konstante (0-7) die den zu setzenden I/O-Pin festlegt.

Beispiel Programm:

```
abc:      LOW 3           'Output-Pin 3=Low.
          pause 1000     'Pause für ca. 1 sec.
          high 3         'Output-Pin 3=High.
          goto abc       'Wiederhole dies.
```

BASIC Befehle

NAP

NAP Dauer

Wechselt für eine kurze Zeit in den "Sleep Mode". Der Stromverbrauch ist reduziert auf ca. 20 μ A, angenommen an den Ausgangspins liegt keine Belastung an.

- **Dauer** ist eine Variable/Konstante die die Dauer der "Schlummerphase" festlegt. Die Dauer ist $(2^{\text{Dauer}}) * \sim 18$ ms. *Dauer* kann zwischen 0 und 7 liegen, resultierend in "Schlummerphasen" von 18 ms bis 2.3 Sekunden.

Beispiel Programm:

```
      .  
      .  
      .  
      NAP 7                               'Briefmarke schläft für  
                                          '2304 ms (2.304 Sek).
```

Übersicht über die einstellbare Zeitdauer:

NAP 0:	18 ms
NAP 1:	36 ms
NAP 2:	72 ms
NAP 3:	144 ms
NAP 4:	288 ms
NAP 5:	576 ms
NAP 6:	1152 ms
NAP 7:	2304 ms

BASIC Befehle

OUTPUT

OUTPUT Pin

Macht angegebenen Pin zu Ausgabe-Pin.

- **Pin** ist eine Variable/Konstante (0-7) die angibt, welcher I/O-Pin angesprochen wird.

Beispiel Programm:

```
        bit0 = 0                'Setze Bit-Variablen
        bit1 = 1

        OUTPUT 2                'Pin 2=Ausgabe-Pin.

abc:    pin2 = bit0             'Ausgabe-Pin 2=Low.
        pin2 = bit1             'Ausgabe-Pin 2=High.

        goto abc                'Springe nach abc.
```

BASIC Befehle

PAUSE

PAUSE Millisekunden

Unterbricht die Programmausführung für ein paar Millisekunden. Die Dauer der Unterbrechung ist so genau wie die Zeitbasis des Resonators. Zusätzliche Zeit (vielleicht 1 ms) wird allerdings benötigt, um die anliegenden Befehle zu holen und auszuführen. Wird die Länge der Pause erhöht, fällt diese zusätzlich benötigte Zeit weniger ins Gewicht.

"Ziemlich genaue" Timer-Funktionen können implementiert werden, indem man eine längere Pause, z.B. 100 ms, einfügt und dann einen Zähler erhöht, auf eine End-Bedingung prüft und - falls nicht wahr - auf die Pause zurückspringt. Im Fall von 100 ms könnte der Fehler +1% sein. Das kann umgangen werden indem man die 100 in eine 99 ändert (basierend auf Tests), oder die 100 auf 1000 erhöht und in Schritten von 1 Sekunde zählt.

- **Millisekunden** ist eine Variable/Konstante (0-65535) die angibt, für wieviele Millisekunden unterbrochen wird.

Beispiel Programm:

```
abc:      low 2                'Output-Pin 2=Low.
          PAUSE 100           'Pause für 0.1 Sekunde.
          high 2              'Output-Pin 2=High.
          PAUSE 100           'Pause für 0.1 Sekunde.
          goto abc
```

POT

POT Pin,Scale,Variable

Liest ein 5-50K Potentiometer ein. Physikalisch kann es sich dabei um einen Thermistor, Photowiderstand, Weg- oder Winkelgeber oder anderen variablen Widerstand handeln. Der Eingabe-Pin muß mit einer Seite eines Widerstands verbunden sein, dessen verbleibende Seite durch einen Kondensator mit Masse verbunden ist. Die Widerstands-Messung erfolgt durch Messung der RC-Zeitkonstante.

- **Pin** ist eine Variable/Konstante (0-7) die den verwendeten I/O-Pin angibt.
- **Scale** ist eine Variable/Konstante (0-255) die verwendet wird um das interne 16-Bit Ergebnis des Befehls zu skalieren. Der 16-Bit-Wert wird mit (Scale/256) multipliziert, so daß ein Scale-Wert von 128 den Bereich um ca. 50% reduziert, eine Skalierung von 64 auf 25% usw. Die Alt-P Option (siehe unten) bietet eine Möglichkeit den besten Scale-Wert für einen bestimmten Widerstand zu finden.
- **Variable** speichert das endgültige Ergebnis der Messung.

Intern errechnet der POT Befehl einen 16-Bit Wert, der dann in einen 8-Bit Wert gewandelt werden muß. Der Faktor durch den der interne Wert umgesetzt wird hängt von der Größe des verwendeten Widerstands ab.

Um den besten Scale-Wert zu finden, drücken Sie Alt-P in der Editor-Software (Die Briefmarke muß dabei mit dem PC verbunden sein). Ein spezielles Kalibrierungs-Fenster erscheint, das Ihnen das Finden des besten Wertes ermöglicht. Die erforderlichen Schritte sind auf der nächsten Seite aufgeführt.

Fortsetzung nächste Seite...

BASIC Befehle

POT (Fortsetzung)

Findes des besten POT Scale-Wertes:

- Drücken Sie Alt-P im Editor. Die Briefmarke muß mit dem PC verbunden sein und der Widerstand für den POT-Befehl muß mit der Briefmarke verbunden sein.
- Ein Fenster erscheint und fragt nach der Nummer des I/O-Pin mit dem der Widerstand verbunden ist. Wählen Sie den entsprechenden Pin (0-7).
- Der Editor sendet ein kurzes Programm an die Briefmarke (überschreibt jedes in der Briefmarke gespeicherte Programm).
- Ein Fenster erscheint mit zwei Zahlen: *Scale* und *Wert*. Justieren Sie den Widerstand bis die Zahl für *Scale* so klein wie möglich ist. Sobald Sie den kleinsten Wert für *Scale* gefunden haben, sind Sie fertig. Diese Zahl sollte im POT Befehl für *Scale* eingesetzt werden.
- Als zusätzlichen Schritt können Sie die *Scale*-Nummer durch Druck auf die Space-Taste verifizieren. Dies blockiert den *Scale*-Wert und veranlaßt die Briefmarke fortlaufend den Widerstand auszulesen. Das Ergebnis wird als *Wert* im gleichen Fenster angezeigt. Ist der *Scale*-Wert gut, sollten Sie den Widerstand so justieren können, daß die Werte zwischen 0 und 255 liegen (oder so dicht wie möglich). Um den *Scale*-Wert zu ändern und diesen Schritt erneut auszuführen, drücken Sie noch einmal die Space-Taste. Machen Sie weiter bis Sie den besten *Scale*-Wert gefunden haben.

Fortsetzung nächste Seite...

BASIC Befehle

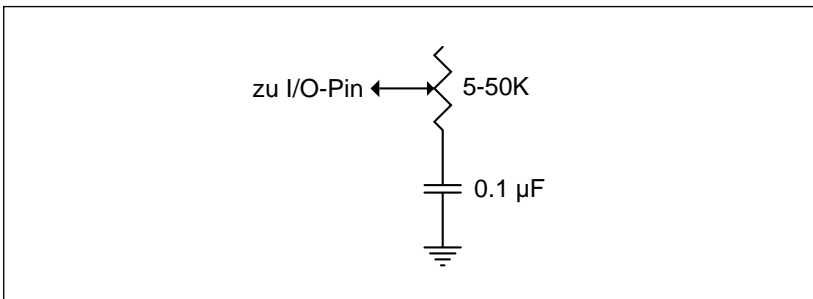
POT (Fortsetzung)

Beispiel Programm:

```
abc:      POT 0,100,b2           'Lese Potentiometer an
                                           'Pin 0.

          serout 1,N300,(b2)     'Sende Potentiometerwert
                                           'über seriellen Ausgang.

          goto abc              'Wiederhole das.
```



BASIC Befehle

PULSIN

PULSIN Pin,Status,Variable

Messen eines Eingangs-Puls in Schritten von 10 µs.

- **Pin** ist eine Variable/Konstante (0-7) die den verwendeten I/O-Pin festlegt.
- **Status** ist eine Variable/Konstante (0 oder 1) die angibt, welcher Pegel auftreten muß damit die Pulsdauer-Messung beginnt.
- **Variable** ist eine Variable die das Ergebnis der Messung speichert (1-65536). Die verwendete Variable kann vom Typ Byte oder Wort sein. Wenn eine Byte-Variable benutzt wird, ist der maximale Meßwert 2.55ms. Ist der Impuls länger als 2.55ms, ist das Ergebnis 0. Wenn eine Word-Variable benutzt wird, ist der maximale Meßwert 0.65536 Sekunden. Ist der Impuls länger als 0.65536 Sekunden, ist das Ergebnis 0. Ist die Zeit bis zu einem Impuls größer als 0.65536 Sekunden, bricht der Befehl ab und das Ergebnis ist 0.

Beispiel Programm:

```
PULSIN 4,0,w2           'Mißt einen Eingangs-Puls
                        'auf Pin 4. Startet die
                        'Messung wenn ein High-
                        'nach-Low Übergang
                        'auftritt. Stoppt die
                        'Messung beim Auftreten
                        'eines Low-nach-High
                        'Übergangs.

serout 1,n300,(b5)     'Sendet High-Byte der 16-
                        'Bit-Puls-Messung über
                        'seriellen Ausgang.

.
.
.
```

BASIC Befehle

PULSOUT

PULSOUT Pin,Zeit

Generiert einen Puls durch Invertieren eines Pins für eine bestimmte Zeitdauer.

- **Pin** ist eine Variable/Konstante (0-7) die den verwendeten I/O-Pin festlegt.
- **Zeit** ist eine Variable/Konstante (0-65535) die die Länge des Pulses in Einheiten von 10 μ s angibt.

Beispiel Programm:

```
abc:      PULSOUT 0,3           'Invert. Pin 0 für 30  $\mu$ s.  
          pause 1             'Pause für 1 ms.  
          goto abc           'Sprung nach abc.
```

BASIC Befehle

PWM

PWM Pin,Duty,Cycles

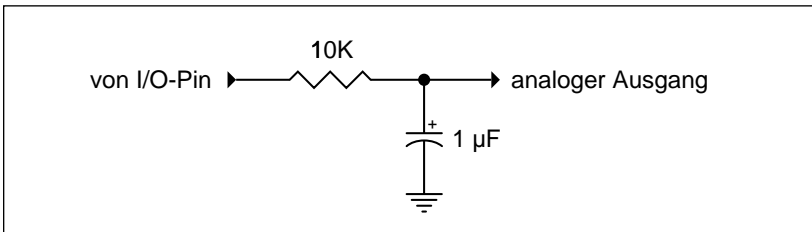
Gibt auf einem Pin ein puls-breiten-moduliertes Signal aus und kehrt dann in den Eingangs-Status des Pin zurück. Kann benutzt werden zum Generieren von analogen Spannungen (0-5V) über einen Pin verbunden mit einem Widerstand und Kondensator mit Masse; die Widerstand-Kondensator-Verbindung ist der analoge Ausgang (siehe Schaltung unten). Da sich der Kondensator langsam entlädt, sollte PWM periodisch ausgeführt werden um die analoge Spannung aufzufrischen.

- **Pin** ist eine Variable/Konstante (0-7) die den verwendeten I/O-Pin festlegt.
- **Duty** ist eine Variable/Konstante (0-255) die den gewünschten analogen Level (0-5 Volt) angibt.
- **Cycles** ist eine Variable/Konstante (0-255) die die Anzahl der Zyklen für die Ausgabe angibt. Größere Kondensatoren benötigen mehrere Zyklen zum vollständigen Aufladen. Jeder Zyklus dauert ca. 5 ms.

Wenn der Analog-Ausgang einen nennenswerten Strom abgeben muß, ist ein Ausgangsverstärker vorzusehen (Op-Amp).

Beispiel Program:

```
abc:   serin 0,n300,b2      'Empfange serielles Byte.  
      PWM 1,b2,20         'Ausgabe analoger Spannung  
                        'entsprech. empfang. Byte.
```



BASIC Befehle

RANDOM

RANDOM Wortvariable

Generiert die nächste Pseudo-Zufallszahl in *Wortvariable*. Die Briefmarke verwendet eine Sequenz von 65535 essentiellen Zufallszahlen um diesen Befehl auszuführen. Wenn der Befehl ausgeführt wird, bestimmt der Wert in *Wortvariable* wo in die Sequenz von Zufallszahlen eingegriffen wird. Wird immer mit dem gleichen Wert initialisiert, erhalten Sie auch immer die gleiche Sequenz von Zufallszahlen. Obwohl diese Methode nicht absolut zufällig ist, reicht sie für die meisten Applikationen aus.

Um wirklich zufällige Ergebnisse zu erhalten, müssen Sie ein unsicheres Element in den Prozeß einbeziehen. Ein Beispiel wäre eine Echtzeit-Uhr, die über einen seriellen Eingang gelesen wird.

- **Wortvariable** ist eine Variable (0-65535) die als Arbeitsplatz für die Routine und ihr Ergebnis fungiert. Jeder Durchlauf durch RANDOM läßt die nächste Zahl der Pseudo-Zufalls-Sequenz zurück.

Beispiel Program:

```
loop:    RANDOM w1           'Generiert 16-Bit Zufalls-
                               'zahl.

        sound 1,(b2,10)     'Generiert zufälligen Ton
                               'auf Pin 1 mit dem Low-
                               'Byte der Zufallszahl (b2)
                               'of the random number (b2)
                               'als Nummer der Note.

        goto loop           'Wiederhole das
```

BASIC Befehle

READ

READ Position,Variable

Liest Position im EEPROM und speichert Wert in Variable.

Das EEPROM wird sowohl als Programmspeicher (von oben nach unten) als auch als Datenspeicher (von unten nach oben) verwendet. Um sicherzustellen, daß Ihr Programm sich nicht selbst überschreibt, lesen Sie die Position 255 im EEPROM bevor Sie Daten schreiben. Position 255 enthält die Adresse des letzten Befehls Ihres Programms. Folglich kann Ihr Programm jeden Platz unterhalb der Adresse in Position 255 verwenden. Enthält Position 255 den beispielsweise den Wert 100, kann Ihr Programm die Positionen 0-99 für Daten verwenden.

- **Position** ist eine Variable/Konstante (0-255) die angibt, welche Position im EEPROM gelesen werden soll.
- **Variable** empfängt den Wert vom EEPROM (0-255).

Beispiel Program:

<code>READ 255,b2</code>	<code>'Hole Position des letzten 'Befehls Ihres Programms.</code>
<code>loop: b2 = b2 - 1</code>	<code>'Nächste freie EEPROM- 'Position</code>
<code>serin 0,N300,b3</code>	<code>'Empfange serielles Byte 'in b3.</code>
<code>write b2,b3</code>	<code>'Speichere empfangenes 'Byte in nächster EEPROM- 'Position.</code>
<code>if b2 > 0 then loop</code>	<code>'Hole weiteres Byte falls 'noch Speicherplatz.</code>

BASIC Befehle

RETURN

RETURN

Rücksprung aus Unterprogramm. RETURN springt zurück auf die Adresse, die der letzten GOSUB-Anweisung folgt. Dort wird die Ausführung fortgesetzt.

RETURN hat keine Parameter.

Beispiel Program:

```
        for b4 = 0 to 10
gosub abc          'Speichere Rücksprung-
                   'Adresse und springe
                   'dann nach abc
        next
        .
        .
        end

abc:    pulsout 0,b4      'Ausgabe eines Puls auf
                   'Pin 0. Puls-Länge ist
                   'b4 x 10 µs.

        toggle 1        'Invertiere Pin 1.

        RETURN          'Rücksprung zur rufenden
                   'Routine.
```

BASIC Befehle

REVERSE

REVERSE Pin

Ändere die Richtung des angegebenen Pins. Ist der Pin ein Eingangs-Pin, wird daraus ein Ausgangs-Pin und umgekehrt.

- **Pin** ist eine Variable/Konstante (0-7) die den zu ändernden I/O-Pin festlegt.

Beispiel Program:

```
dir3 = 0                'Pin 3=Input.  
  
REVERSE 3              'Pin 3=Output.  
REVERSE 3              'Pin 3=Input.
```

BASIC Befehle

SERIN

SERIN Pin,Baudmode,(Trigger,Trigger,...)

SERIN Pin,Baudmode,{#}Variable,{#}Variable,...

SERIN Pin,Baudmode,(Trigger,Trigger,...),{#}Variable,{#}Var,...

Einrichten eines seriellen Eingangs und warten auf optionale Trigger und/oder Variablen. Sind Trigger angegeben, müssen Sie in der exakten Reihenfolge empfangen werden, bevor die Ausführung weiterläuft. Sind Variablen angegeben, werden Sie mit den empfangenen Daten aufgefüllt (*Achtung: nur auf Trigger folgende Daten werden zum Besetzen der Variablen verwendet*).

Variablen-Namen können optional mit vorgestelltem Nummern-Symbol (#) versehen werden. Dies sagt der Routine, nur ASCII-Daten als Daten für die Variablen anzunehmen. Zum Vergleich: Ist die Variable "abc" als nächste zu besetzen, würde das nächste Daten-Byte dies tun. Ist die Variable "#abc", wartet die Routine bis eine Zahl empfangen wird. Eine "Zahl" ist jede Byte-Folge von ASCII-Ziffern 0-9, gefolgt von einem nicht-Ziffern-Zeichen wie A-Z, Space, usw. Dies ist nützlich zum Lesen von Zahlen von einem Terminal oder anderen Gerät das ASCII-Format abgibt. Solche Geräte übertragen z.B. "108" in drei Bytes ('1', '0', '8'), gefolgt von Space oder einem anderen Zeichen.

- **Pin** ist eine Variable/Konstante (0-7) die den verwendeten I/O-Pin festlegt..
- **Baudmode** ist eine Variable/Konstante (0-7) die den Modus des seriellen Ports festlegt (durch # oder durch Symbol):

#	Symbol	Baudrate	Eingang wahr/invert
0	T2400	2400	wahr
1	T1200	1200	wahr
2	T600	600	wahr
3	T300	300	wahr
4	N2400	2400	invert
5	N1200	1200	invert
6	N600	600	invert
7	N300	300	invert

BASIC Befehle

SERIN (Fortsetzung)

- **Trigger** sind optionale Variablen/Konstanten (0-255) die in exakter Reihenfolge empfangen werden müssen.
- **Variablen** (optional) speichern die empfangenen Daten (0-255). Falls Trigger angegeben sind, müssen diese erfüllt sein bevor die Variablen besetzt werden.

Wenn Sie den seriellen Eingang verwenden, können Sie nur die Baudrate und den wahr/invert-Status anpassen. Die Daten müssen immer im Format **8 Datenbit, No Parity, 1 Stopbit** eintreffen.

Beispiel Programm:

```
read 255,b2           'Hole Position des letzten
                      'Programm-Befehls.

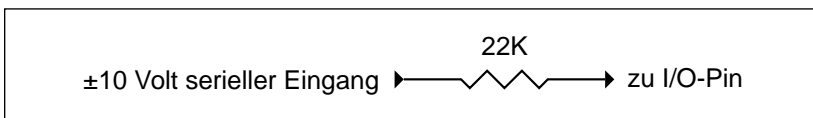
loop:  b2 = b2 - 1     'Nächste verfügbare
                      'EEPROM Position.

SERIN 0,N300,("ABC"),b3 'Warte bis "ABC" empfangen
                      'nächstes Byte in b3.

write b2,b3          'Speichere empfangenes
                     'Byte in nächster
                     'EEPROM Position.

if b2 > 0 then loop  'Hole weiteres Byte falls
                     'noch Speicherplatz.
```

Soll die Briefmarke ein RS-232 Signal empfangen, kann dies durch Hinzufügen eines 22K Widerstands erreicht werden. Schalten Sie den Widerstand in Reihe mit dem Eingangs-Signal. Dies begrenzt den Eingangswert auf einen verwertbaren Pegel. Die ± 10 Volt RS-232 Pegel, werden intern auf 0 bis 5 Volt begrenzt. Schaltbild:



BASIC Befehle

SEROUT

SEROUT Pin,Baudmode,({#}Data,{#}Data,...)

Einrichten eines seriellen Ausgangs und senden von Daten.

Variablen/Konstanten können optional mit vorgestelltem Nummern-Symbol (#) versehen werden. Dies sagt der Routine die Daten als ASCII-Daten zu senden. Zum Vergleich: Soll die Variable "abc" gesendet werden, würde die Routine ein Byte senden. Ist die Variable "#abc", sendet die Routine soviele Byte wie Ziffern in der Zahl in "abc" sind. Eine "Zahl" ist eine Byte-Folge von ASCII-Ziffern 0-9, gefolgt von einer nicht-Ziffer wie A-Z, Space, usw. Ein Beispiel wäre "108", was in drei Bytes ('1', '0', '8') gesendet würde, gefolgt von Space oder einem anderen Zeichen. Dies ist nützlich zum Senden an ein Gerät, das Daten im ASCII-Format erwartet.

- **Pin** ist eine Variable/Konstante (0-7) die den verwendeten I/O-Pin festlegt..
- **Baudmode** ist eine Variable/Konstante (0-7) die den Modus des seriellen Ports festlegt (durch # oder durch Symbol):

#	Symbol	Baudrate	wahr/invert	Open/Driven
0	T2400	2400	wahr	stets getrieben
1	T1200	1200	wahr	stets getrieben
2	T600	600	wahr	stets getrieben
3	T300	300	wahr	stets getrieben
4	N2400	2400	invert	stets getrieben
5	N1200	1200	invert	stets getrieben
6	N600	600	invert	stets getrieben
7	N300	300	invert	stets getrieben
8	OT2400	2400	wahr	open drain
9	OT1200	1200	wahr	open drain
10	OT600	600	wahr	open drain
11	OT300	300	wahr	open drain
12	ON2400	2400	invert	open source
13	ON1200	1200	invert	open source
14	ON600	600	invert	open source
15	ON300	300	invert	open source

BASIC Befehle

SEROUT (Fortsetzung)

- **Data** sind Variablen/Konstanten (0-255) die von der seriellen Routine ausgegeben werden.

Wenn Sie den seriellen Eingang verwenden, können Sie nur die Baudrate und den wahr/invert-Status anpassen. Die Daten müssen immer im Format **8 Datenbit, No Parity, 1 Stopbit** eintreffen.

Beispiel Programm:

```
abc:    pot 0,100,b2           'Lies Potentiometer auf
                                     'Pin 0.

        SEROUT 1,N300,(b2)    'Sende Potentiometer-
                                     'Messung über
                                     'seriellen Ausgang.

        goto abc              'Wiederhole das.
```

Der serielle Ausgang der Briefmarke verwendet 0-5 Volt Pegel, könnte also u.U. RS-232 Geräte nicht ohne zusätzliche Schaltung betreiben. Die meisten seriellen PC-Ports funktionieren allerdings auch ohne zusätzliche Schaltung.

BASIC Befehle

SLEEP

SLEEP Sekunden

Wechselt für einige Sekunden in den "Sleep Mode". Die Auflösung dieses Befehls ist ca. 2.3 Sekunden, die generelle Genauigkeit ist ca. 99.9%. Der Stromverbrauch ist reduziert auf ca. 20 μ A, angenommen an den Ausgangs-Pins liegt keine Belastung an.

- **Sekunden** ist eine Variable/Konstante (1-65535) die die Dauer des "Sleep Mode" in Sekunden angibt. Die Länge kann zwischen 1 Sekunde und etwas über 18 Stunden liegen.

Beispiel Programm:

```
.  
.   
.   
SLEEP 3600           'Schlafe für ca. 1 Stunde.  
  
goto xyz            'Weiter nach dem "Schlaf".
```

BASIC Befehle

SOUND

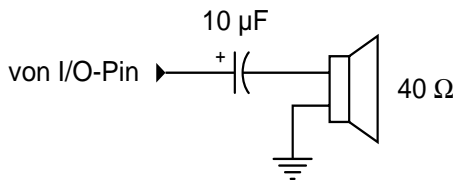
SOUND Pin,(Note,Dauer,Note,Dauer,...)

Spielt Noten mit vorgegebener Dauer. Der Ausgangs-Pin muß mit der positiven Seite eines Kondensators verbunden sein, dessen negative Seite mit einem Lautsprecher verbunden ist, dessen andere Seite wiederum mit Masse verbunden ist (siehe Schaltbild unten). Die Töne werden durch Rechteck-Wellen generiert.

- **Pin** ist eine Variable/Konstante (0-7) die den verwendeten I/O-Pin festlegt.
- **Note(n)** sind Variablen/Konstanten (0-255) die den Typ und die Frequenz angeben. Ton 0 bedeutet Ruhe für die angegebene Dauer. Die Töne 1-127 sind aufsteigende klare Töne. Töne im Bereich 128-255 sind weißes Rauschen.
- **Dauer** sind Variablen/Konstanten (0-255) die angeben wie lange jede Note gespielt wird.

Beispiel Programm:

```
for b2 = 0 to 255  
  
  SOUND 1,(25,10,b2,10) 'Konstanter Ton (25),  
                        'gefolgt von aufsteigendem  
                        'Tone (b2). Beide Töne mit  
                        'gleicher Dauer (10).  
  
next
```



*Kondensator entfernen bei Verwendung
eines piezo-elektrischen Lautsprechers*

BASIC Befehle

TOGGLE

TOGGLE Pin

Bringe Pin in Ausgangs-Status und invertiere ihn.

- **Pin** ist eine Variable/Konstante (0-7) die den verwendeten I/O-Pin festlegt.

Beispiel Programm:

```
for b2 = 1 to 25
  TOGGLE 5           'Invertiere Pin 5.
next
```

BASIC Befehle

WRITE

WRITE Position,Data

Speichere Daten an Position in EEPROM.

Das EEPROM wird sowohl als Programmspeicher (von oben nach unten) als auch als Datenspeicher (von unten nach oben) verwendet. Um sicherzustellen, daß Ihr Programm sich nicht selbst überschreibt, lesen Sie die Position 255 im EEPROM bevor Sie Daten schreiben. Position 255 enthält die Adresse des letzten Befehls Ihres Programms. Folglich kann Ihr Programm jeden Platz unterhalb der Adresse in Position 255 verwenden. Enthält Position 255 den beispielsweise den Wert 100, kann Ihr Programm die Positionen 0-99 für Daten verwenden.

- **Position** ist eine Variable/Konstante (0-255) die angibt, an welche Position im EEPROM geschrieben werden soll.
- **Data** ist eine Variable/Konstante (0-255) die an die Position im EEPROM geschrieben wird.

Beispiel Programm:

	<code>read 255,b2</code>	'Hole Position des letzten 'Befehls Ihres Programms.
<code>loop:</code>	<code>b2 = b2 - 1</code>	'Nächste freie EEPROM- 'Position
	<code>serin 0,N300,b3</code>	'Empfange serielles Byte 'in b3.
	<code>WRITE b2,b3</code>	'Speichere empfangenes 'Byte in nächster EEPROM- 'Position.
	<code>if b2 > 0 then loop</code>	'Hole weiteres Byte falls 'noch Speicherplatz.

Aktuelle Applikations-Berichte

Eine Reihe von Veröffentlichungen und Applikations-Berichten über die BASIC-Briefmarken Technologie sind bereits in der Fachpresse erschienen - Hinweise hierzu erhalten Sie unter der unten stehenden Adresse.

Ferner gibt es Beispiel-Applikationen mit Hardware-Schaltungen und kompletten Listings im Toolkit des BASIC-Briefmarken Entwicklungs-Systems.

Auch von Anwendern liegen einzelne Beispiel-Applikationen vor, die im Toolkit Handbuch wiedergegeben sind.

Wenn Sie selbst interessante Lösungen mit der BASIC-Briefmarke realisiert haben, und Sie möchten diese anderen Anwendern zugänglich machen, senden Sie uns Schaltung, Programm-Listing und einen erläuternden Text ein. Es ist geplant solche Applikationen in einer eigenen Broschüre zusammenzufassen.

Bezug

Chips, 1-Platinen Computer und das Entwicklungs-System "Die BASIC-Briefmarke" sind zu beziehen über:

Wilke Technology GmbH
Krefelder Str. 147
52070 Aachen, Germany

bzw. **Wilke Technology GmbH**
P.O.Box 1727
52018 Aachen, Germany

Tel: ++49 (241) 154071

Fax: ++49 (241) 158475

oder über kooperierende Partner.

Versionen: deutsch, englisch, französisch, italienisch.

Notizen

Notizen

Notizen

Wilke Technology GmbH
P.O. Box 1727
D-52018 Aachen, Germany
Phone: +49 (241) 91890-0
Fax: +49 (241) 91890-44
Mailbox: +49 (241) 91890-55
Internet: Wilke@RMI.DE