# AVR-ChipBasic2: BASIC-Referenz

## V1.45 (c) 2006-2012 Jörg Wolfram


## 1 Lizenz

Das Programm unterliegt der GPL (GNU General Public Licence) Version 3 oder höher, jede Nutzung der Software/Informationen nonkonform zur GPL oder ausserhalb des Geltungsbereiches der GPL ist untersagt!

Die Veröffentlichung dieses Programms erfolgt in der Hoffnung, daß es Ihnen von Nutzen sein wird, aber OHNE IRGENDEINE GARANTIE, auch ohne die implizite Garantie der MARKTREIFE oder der VERWENDBARKEIT FÜR EINEN BESTIMMTEN ZWECK.

Alle im Text genannten Marken sind Eigentum des entsprechenden Inhabers.

## 2 Allgemeines

Jedes Programm besteht aus maximal 95 Programmzeilen (1-95). Für längere Schlüsselwörter existieren teilweise Abkürzungen, diese stehen in rechteckigen Klammern neben der ausgeschriebenen Version. Nach jedem Schlüsselwort muss ein Leerzeichen stehen. Viele Befehle blenden nicht benutzte Bits bei den Parametern aus (z.B. COLOR) oder begrenzen auf den gültigen Wertebereich (z.B. PLOT). Leider werden in der aktuellen Version nicht alle Fehler abgefangen, in so einem Falle (meist wird der ganze Bildschirm mit Nullen oder unsinnigen Zeichen vollgeschrieben) sollte man mit CTRL+ALT+DEL den Computer neu starten und sich den BASIC-Programmtext nocheinmal ansehen.

# 3 Zahlen, Variablen und Funktionen

AVR-ChipBASIC kennt nur einen Datentyp, und das sind 16 Bit Integerzahlen. Dazu gibt es 26 Variablen (A-Z) und ein Array mit 768 oder 1024 Elementen AR(), dazu aber später mehr. Konstanten können sowohl in Dezimalform als auch in Hexadezimalform eingegeben werden, wobei bei letzterer keine negativen Werte erlaubt sind. Hexadezimalzahlen beginnen mit **$**. Folgende Operationen sind erlaubt:

- Addition **+**
- Subtraktion **-**
- Multiplikation *****
- Division **/**
- Modulo **%**
- Und-Verknüpfung **&**
- Oder-Verknüpfung **#**
- Exclusiv-Oder-Verknüpfung **^**
- Vergleiche (**=,<>,<,<=,>,>=**) liefern 0 für falsch oder 1 für wahr
- Bit-Shift nach links **«**
- Bit-Shift nach rechts **»**

Dazu kommen noch öffnende und schließende Klammern sowie Funktionen. Die Operatoren sind in unterschiedliche Prioritätsgruppen eingeteilt. Soll die Priorität eines Operators erhöht werden, müssen die dazugehörigen Operanden geklammert werden.

| **Priorität** | **Operatoren** |
|---|---|
| höchste | * , / , % , # , ^ |
| ... | « , » |
| ... | + , - , & |
| niedrigste | = , <> , < , <= , > , >= |

Funktionen können im Gegensatz zu Schlüsselwörtern nicht abgekürzt werden. Da die öffnende Klammer Bestandteil des Funktionsnamens ist, darf zwischen Funktionsname und Klammer kein Leerzeichen stehen.

| | |
|---|---|
| ABS( | Berechnet den Absolutwert des eingeklammerten Ausdruckes |
| SGN( | Berechnet das Vorzeichen (-1,0,1) des eingeklammerten Ausdruckes |
| NOT( | invertiert den eingeklammerten Ausdruck |
| SQR( | zieht die Quadratwurzel aus dem eingeklammerten Ausdruck |
| RND( | erzeugt eine Zufallszahl zwischen 0 und dem eingeklammerten Ausdruck |
| IN( | liefert den Pegelwert des angegebenen Portpins (0-7) an der parallelen Schnittstelle, ist der Parameterwert 255 werden die Portpins als Byte eingelesen, |
| SPI( | gibt ein Byte über die SPI-Schnittstelle aus und gibt den eingelesenen Wert zurück |
| TEMP( | liefert den Temperaturwert des angegebenen LM75-Sensors (0-7) |
| ADC( | liefert den Analogwert der Spannung des angegebenen Portpins (0-7) an der parallelen Schnittstelle |
| XPEEK( | liest ein Datenbyte aus dem optionalen Daten-EEPROM, der Klammerausdruck bestimmt die Adresse |
| EPEEK( | liest ein Datenbyte aus dem internen EEPROM, Adresse 0...999 |
| VPEEK( | liest ein Datenbyte aus dem Bildwiederholspeicher |
| LO( | liefert das niederwertige Byte des eingeklammerten Ausdruckes |
| HI( | liefert das höherwertige Byte des eingeklammerten Ausdruckes |
| KEY( | liefert verschiedene Tastaturabfragen, siehe Abschnitt Tastatur |
| AR( | Array-Wert (s.u.) |
| SIN( | liefert den Sinus des Winkels (in 1/10 Grad) mal 255 |
| COS( | liefert den Cosinus des Winkels (in 1/10 Grad) mal 255 |
| FTYPE( | siehe Abschnitt Dateifunktionen |
| FSIZE( | siehe Abschnitt Dateifunktionen |
| ERR( | Fehlerinformationen, siehe Abschnitt Fehlerhandling |
| XREC( | liest einen Datenblock via XModem-Protokoll ein |
| DBIT( | konvertiert Zeichensatzdaten |
| PSTAT( | Sequenzer-Status, siehe Abschnitt Audio |

Zusätzlich zu den normalen Variablen gibt es noch die Arrayvariable **AR()**.

- Die Zellen 0...767 sprechen den internen Bereich byteweise an
- Die Zellen 768...1023 sprechen die aktive Page des externen Bereiches byteweise an
- Die Zellen 1024...1407 sprechen den Bereich im wortweise an
- Die Zellen 1408...1535 sprechen die aktive Page des externen Bereiches wortweise an

Bei den Byte-Zugriffen befindet sich das LOW-Byte an der geraden Adresse und das High-Byte an der darauf folgenden ungeraden Adresse. Wird auf eine Array-Zelle zugegriffen die nicht existiert, wird mit einer Fehlermeldung abgebrochen. Über **AR()** lässt sich nur vom Array lesen, Schreibzugriffe müssen mit DATA erfolgen.
Um auf das externe Array zuzugreifen, kann mittels des Befehles **PAGE nn** die aktive Page im externen Speicher eingestellt werden. Einige Dinge wie Datei-I/O funktionieren nur mit dem internen Array-Speicher, um den externen Speicher nutzen zu können, müssen die Daten mittels **ACOPY** umkopiert werden. Voraussetzung dafür ist, dass auf Programmplatz 8 ein Treiber mit Unterstützung für XMEM geladen wurde.

# 4 Wertzuweisung

Wertzuweisungen beginnen nicht mit einem Schlüsselwort, sondern mit einem Variablennamen gefolgt von einem Gleichheitszeichen und einem Ausdruck.

```
01 A=-4
02 B=SQR(8*2)
```

## 4.1 ASHIFT v,n

Der Absolutwert der Variable v wird um n Bits verschoben. Für N>0 wird bitweise nach links verschoben, für N<0 nach rechts. Sinnvolle Werte für N liegen zwischen -15 und +15. Im Gegensatz zu den Shift-Operatoren wird nur der Absolutbetrag verschoben, damit liefert dieser Befehl auch bei negativen Werten ein richtiges Ergebnis.

```
01 A=-39
02 ASHIFT A,-1
03 PRINT A
```

Da A um 1 Bits nach rechts verschoben wird, wird der Wert -19 ausgegeben. Im Gegensatz dazu ergibt:

```
01 A=-39 >> 1
02 PRINT A
```

das Ergebnis 32749, da in die Vorzeichenstelle eine 0 geschoben wurde.

## 4.2 LIMIT v,min,max [LIM]

Der Wert der Variable v wird auf den Wertebereich min...max begrenzt. Die Funktion ist nur auf Variablen, nicht auf Arrayelemente erlaubt.

```
01 LIMIT A,1,6
02 LI B,C,C+4
```

im ersten Beispiel wird die Variable A auf den Bereich 1...6 begrenzt, im zweiten Beispiel die Variable B auf den Wertebereich, der durch den Inhalt der Variable C und die 4 darauffolgenden Zahlen begrenzt ist.

## 4.3 SCALE Variable,Y0,Y1,X0,X,X1

Dieser Befehl dient dazu, Werte von einem Bereich in einen anderen zu transformieren. X ist dabei der ”INPUT”-Bereich, Y liefert das Resultat. Ein Beispiel:

```
01 V=180
02 SCALE A,0,100,0,V,256
03 ? V
```

liefert z.B. als Resultat den wert **70**. In diesm Falle heißt das, der Wert von V (180) entspricht 70 Prozent vom Maximalwert 256. Man kann sich das so vorstellen, daß das Resultat so zwischen Y0 und Y1 liegt, wie der Wert X zwischen den Grenzen X0 und X1 liegt. Die Formel dafür ist:

```
 Y = Y0 + ((Y1-Y0) * (X-X0) / (X1-X0))
```

Damit das Ganze auch hinreichend genau ist, wird intern mit 32 Bit gerechnet. Aufgrund der begrenzten internen Programmspeicherkapazitäten werden aber nicht alle Bedingungen für das Zustandekommen eines gültigen Resultates überprüft. So sollte Y1 betragsmäßig immer größer als Y0 sein, X1 betragsmäßig größer als X0 sein und X sollte zwischen X0 und X1 liegen.

## 4.4 DATA offset,value1,value2... [DA]

Die Array Elemente werden ab Byte (offset) initialisiert. Als Werte kommen Konstanten, numerische Ausdrücke und Zeichenketten in Frage. Im Word-Bereich des Arrays (ab Offset 1024) werden immer 16 Bit geschrieben, das betrifft insbesondere Zeichenketten wo das HIGH-Byte immer 0 ist. Andersherum wird im Bytebereich des Arrays nur das Low-Byte der Werte geschrieben.

```
01 DATA 0,''Hallo'',0
```

Das Array wird ab Element 0 mit der nullterminierten Zeichenkette „Hallo” belegt.

## 4.5 ACOPY source,dest,num... [AC]

Eine Anzahl (num) von Array Elementen wird ab Byte (source) nach Byte (dest) kopiert. Das Kopieren erfolgt elementweise, so kann z.B. auch von 8 auf 16 Bit erweitert werden. Andersherum wird beim Kopieren vom Word- in den Bytebereich des Arrays nur das Low-Byte der Werte geschrieben.

```
02 ACOPY 0,1029,10
```

Das Array wird ab Byte-Element 0 in die Word-Elemente 1029...1038 (Byte-Elemente 10...29) kopiert.

Wichtig ist dabei, dass die Arrayzellen nur aufsteigend kopiert werden. Überlappen sich Quell- und Zielbereich und ist die Quelladresse kleiner als die Zieladresse, wird das Ergebnis nicht dem Gewünschten entsprechen, das Teile des Quellbereiches überschrieben werden bevor sie in den Zielbereich kopiert werden.

## 4.6 DBIT(N)

Diese Funktion vereinfacht das Erstellen von selbstdefinierten Zeichen im Videomodus 4. Vom Argument N werden nur die 6 unteren Bits berücksichtigt und in ein Bitmuster umgewandelt, welches bei der Darstellung dem Erwarteten entspricht. Grund dafür ist die Organisation des Zeichenspeichers: Hier werden Bit 0+1 nicht benutzt, Bit 7 entspricht dem Pixel ganz links und die nachfolgenden Bits (6...2) ensprechen jeweils der Differenz zum vorherigen Pixel. Sollen zum Beispiel alle 6 Pixel in einer Zeile gesetzt werden, muß 0x80 im Byte stehen. Bit 7 ist "1" und alle nachfolgenden "0", was "keine Änderung" bedeutet. Damit man sich diese Umrechnung ersparen kann, rechnet die Funktion **DBIT()** das anzuzeigende Bitmuster systemgerecht um. Vom Parameter N werden dabei nur die unteren 6 Bits benutzt.

```
01 VM 4:? @0,0;%0;:BORDER 4
02 DA 0,"88GG88GG88"
03 WAIT 10
04 FOR K=0 TO 9:C=DBIT(AR(K))
05 VPOKE 1448+K*128,C:NEXT
```

Das Programm ändert nach 1 Sekunde das Aussehen von Zeichen 0 welches links oben angezeigt wird, in eine Art "Schachbrettmuster".

# 5 Programmsteuerung

## 5.1 BREAK

Setzt einen Breakpoint, ruft den Monitor auf.

## 5.2 END

Mit dem END-Befehl wird das Programm an der aktuellen Stelle beendet. Das gleiche geschieht auch, wenn die letzte Programmzeile abgearbeitet ist.

## 5.3 GOTO n [GO]

Mit dem GOTO-Befehl kann die Programmabarbeitung mit einer anderen Zeile fortgesetzt werden. Argument ist ein beliebiger Ausdruck.

```
01 B=0:GOTO 2
02 GO B+1
```

Das ist eine Endlosschleife, die nur mit CTRL+C abgebrochen werden kann.

## 5.4 IF - THEN

Die bedingte Anweisung besteht aus **IF** gefolgt von einem Ausdruck. Ist das Ergebnis des Ausdrucks Null, wird zum Anfang der nächsten Zeile gesprungen, andernfalls wird die Zeile weiter abgearbeitet. Das **THEN** kann auch weggelassen werden. Schon wegen der Übersichtlichkeit und Lesbarkeit des Codes sollte dies aber nur dann erfolgen, wenn kein Platz mehr in der Zeile ist. Folgt als erstes eine Wertzuweisung, muss bei fehlenden **THEN** ein Doppelpunkt zwischen die beiden Anweisungen gesetzt werden.

```
01 IF A>5 THEN A=5
02 IF A>5:A=5
```

Beide Ausdrücke bewirken exakt das Gleiche.

## 5.5 FOR - NEXT

Bei der Zählschleifen-Abarbeitung gibt es nur die Grundform **FOR A=B TO C** ohne die Angabe der Schrittweite, die konstant 1 ist. Die Schleife wird auf jeden Fall mindestens 1 mal durchlaufen, wenn C keiner als B ist findet ein Überlauf statt und die Anzahl der Schleifendurchläufe ist N=C-B+65536.

```
01 CLS
02 FOR A=0 TO 9
03 FOR B=0 TO 9
04 PLOT A,B,3
05 NEXT :NEXT
```

Dieses Programm zeichnet ein Rechteck in die obere linke Ecke des Bildschirmes.

## 5.6 REPEAT - UNTIL [REP - UNT]

Dies ist eine weitere Schleifenkonstruktion, eine sogenannte Fußgesteuerte oder auch Nichtabweis-Schleife. Das heißt, der Programmteil zwischen **REPEAT** und **UNTIL** wird mindestens einmal durchlaufen.

```
01 A=0
02 REPEAT
03    ? A
04    A=A+3
05 UNTIL A<10
```

Schreibt die Zahlen 0,3,6 und 9 untereinander auf den Bildschirm.
Systembedingt gibt es kein EXIT, um eine Schleife abzubrechen. Ein einfaches GOTO aus der Schleife heraus ist auch keine gute Lösung, da damit ein Stackeintrag blockiert wird. Eine praktikable Lösung wäre:

```
01 REPEAT
02    ? "Hallo! ";
03    IF KEY(0)>0 UNT 0: GO 5
04 UNTIL 1
```

schreibt solange ”Hallo!” auf den Bildschirm, bis eine Taste gedrückt wird. Das UNTIL 0 bewirkt in diesem Fall, dass der reservierte Stackeintrag wieder freigegeben wird.

## 5.7 GOSUB [GOS]

**GOSUB Expr** ruft das Unterprogramm in der durch den Ausdruck definierten Zeile auf, auf die Zeilennummer können noch bis zu 5 Parameter folgen. Da der Stack auf 10 Einträge begrenzt ist, lassen sich nur insgesamt 10 Schleifen bzw. Unterprogrammaufrufe schachteln.

```
01 GOSUB 4,2,3
02 ? ~R
03 END
04 RETURN ~(1)+~(2)
```

Das Unterprogramm in Zeile 4 addiert die beiden Parameter, in diesem Fall 2 + 3 = 5. In der zweiten Zeile wird dann das Resultat ausgegeben.

## 5.8 RETURN [RET]

Mit **RETURN** wird aus einem Unterprogramm wieder zurückgesprungen, mit **RETURN returnwert** kann ein 16 Bit Wert zurückgegeben werden, der dann mit der Systemvariable ~R im aufrufenden Programm verfügbar ist.

## 5.9 CALL

Eine weitere Anweisung ist **CALL prognr,zeile** die Unterprogramme in einem der 7 anderen Programme aufrufen kann. Damit ist es möglich, den gesamten Programmspeicherbereich für eine einzige Anwendung zu nutzen. Auf die Programm- und Zeilennummer können noch bis zu 5 Parameter folgen, der Rücksprung erfolgt wie gehabt mit **RETURN**.

Da der Stack auf 10 Einträge begrenzt ist, lassen sich nur 10 Schleifen bzw. Unterprogrammaufrufe schachteln. Weiterhin führt das Verlassen von Schleifen oder Unterprogrammen zu blockierten Stack-Einträgen, die dann nicht mehr genutzt werden können.

Wenn das aufgerufene Programm vom Typ Binär ist, gibt es zwei Möglichkeiten:

- Der zweite Parameter ist eine Zahl und gibt den Offset zum Programmanfang in WORDS an. Möglich sind Werte zwischen 0...1535. Danach können noch bis zu 4 weitere Parameter an das Binärprogramm übergeben werden. Das T-Flag im Statusregister ist bei Übergabe zurückgesetzt
- Der zweite Parameter ist eine in Gänsefüsschen gesetzte Zeichenkette. Hier wird dann ein Zeiger auf die Zeichenkette im Y-Register übergeben. Einsprungadresse ist Programmanfang+9 WORDS, das T-Flag im Statusregister ist bei Übergabe gesetzt

## 5.10 LFIND V,n

Mit dieser Anweisung lässt sich feststellen, ob eine bestimmte Binärbibliothek geladen ist. Dazu hat jede dieser Bibliotheken eine (hoffentlich) eindeutige Kennung in Byte 14 und 15. Wenn das LOW-Byte des Suchwertes gleich 0 ist, wird das LOW-Byte der Bibliotheks-ID ignoriert.

Die Variable wird auf 1...8 gesetzt, wenn eine entsprechende Bibliothek gefunden wurde.

```
01 LFIND L,$1200
02 IF L>0 GOTO 4
03 ?"MATLIB1 NICHT GEFUNDEN":END
04 ?"MATLIB1 AUF PLATZ";L:
```

## 5.11 VID n

Dieser Befehl ersetzt seit Version 1.00 die Befehle **SLOW** und **FAST** um die Bildausgabe ein- und auszuschalten. Mit **VID 0** wird die Bidausgabe ausgeschaltet, Synchronsignale werden auch weiterhin erzeugt. **VID 1** schaltet die Bildausgabe wieder ein. Mit ausgeschalteter Bildausgabe ist die Geschwindigkeit höher, da ein großer Teil der Rechenleistung normalerweise für die Bilddarstellung verwendet wird. Es sollte dabei beachtet werden, dass damit auch der Aufruf geladener Videotreiber abgeschaltet wird, was z.B. bei einem PWM- oder LED-Multiplex Treiber ungewünschte Folgen haben kann.

## 5.12 Die System-Variablen

Die Sytem-Variablen (definiert durch vorangestellte Tilde, also die ”Schlangenlinie”) geben Auskunft über den System-Zustand und können nur gelesen werden. Zwischen das Tilde-Zeichen ~und dem Buchstaben für die Systemvariable darf sich KEIN LEERZEICHEN befinden.

### 5.12.1 Das Array ~(1) ... ~(5) (Funktionsparameter)

In diesem Array stehen die Parameter, die beim letzen **GOSUB** bzw. **CALL** übergeben wurden. Liegt das Argument außerhalb des Bereiches 1...5, wird 0 zurückgegeben.

### 5.12.2 ~M (Mikrocontroller)

In dieser Variable steht, ob es sich um einen Mega644 (1) oder Mega644P (2) handelt. Dabei wird ein Mega644P mit Input-Pin der ersten seriellen Schnittstelle an PD3 als Mega644 behandelt, da auch hier die zweite serielle Schnittstelle nicht nutzbar ist.

### 5.12.3 ~L (aktuelle Programmzeile)

Diese Variable liefert als Wert die gerade abgearbeitete Programmzeile. Damit lassen sich z.B. relative Sprünge realisieren. Der Wert liegt zwischen 1 und 95, die folgende Programmzeile ezeugt in jeder Zeile eine Endlosschleife, die einen ”bunten” Rand erzeugt:

```
xx A=(A+1)\&15:BORDER A: GOTO ~L
```

### 5.12.4 ~N (Parameter-Anzahl)

In dieser Variable steht, wieviele Parameter beim letzten **GOSUB** bzw. **CALL** übergeben wurden.

### 5.12.5 ~P (aktuelles Programm)

Diese Variable liefert als Wert das gerade abgearbeitete Programm. Der Wert liegt zwischen 1 und 8. Wenn mittels CALL P,N ein Unterproramm in einem anderen Programm aufgerufen wurde, wird nicht das aufrufende Programm zurückgegeben, sondern das im Moment aktive.

#### 5.12.6 ~R (Return-Wert)

Diese Variable enthält den zuletzt bei **RETURN** angegebenen Wert.

#### 5.12.7 ~S (Scanline)

Diese Variable liefert als Wert die aktuelle Bildschirmzeile, die gerade dargestellt wird. Bei NTSC bewegt sich der Wert zwischen 0 und 262, bei PAL zwischen 0 und 312. Anwendungsgebiete sind z.B. genauere Zeitmessungen, bei denen **TGET** zu grob auflöst. Dabei sollte aber die Videoausgabe abgeschaltet werden.

```
01 REPEAT :UNTIL ~S<230:BO 4
02 REPEAT :UNTIL ~S>100:BO 0:GO 1
```

Auf dem Bldschirm sollte jetzt oben und unten ein grüner Rand zu sehen sein, dazwischen ist der Bildschirm schwarz. Allerdings ist dabei ein starkes Zittern der Kanten zu beobachten, was einfach daher rührt, dass die Abarbeitung der Befehle länger dauert als eine Bildschirmzeile.

#### 5.12.8 ~V (Videosystem)

diese Variable liefert als Wert die Anzahl der Bildschirmzeilen je Halbbild. Eine mit **~S** ermittelte aktuelle Bildschirmzeile wird immer zwischen 0 und **~V**-1 liegen. Bei NTSC ist ~V = 263, bei PAL beträgt der Wert 313.

#### 5.12.9 ~X (X-Ausdehnung)

diese Variable liefert als Wert die Anzahl der Pixel in horizontaler Richtung. Dabei wird die aktuelle Auflösung berücksichtigt. In Video-Mode 0 hat die variable z.B. den wert 60 und im Mode 1 den Wert 168.

#### 5.12.10 ~Y (Y-Ausdehnung)

diese Variable liefert als Wert die Anzahl der Pixel in vertikaler Richtung. Dabei wird die aktuelle Auflösung berücksichtigt. In Video-Mode 0 hat die variable z.B. den wert 46 und im Mode 1 den Wert 116.

### 5.13 Erweiterungen

Wenn sich auf Programmplatz 6 eine Bibliothek mit Bibliothekscode 0x70...0x7f befindet, können dort zusätzliche Befehle ausgewertet und ausgeführt werden. Diese Befehle beginnen mit einem Unterstrich. Ist der Befehl dort nicht bekannt oder keine passende Bibliothek geladen, wird mit einem **UNKNOWN KEYWORD** Fehler abgebrochen. Wie bei den eingebauten Befehlen können auch Variablen oder Parameter folgen.

# 6 Bildschirm-Ausgabe

## 6.1 COLOR F(,B) [COL]

Mit dem COLOR-Befehl wird die Vorder- sowie die Hintergrundfarbe festgelegt. Diese wird im Gegensatz zu früheren Versionen bei allen Ausgaben bercksichtigt, da jetzt für jedes Zeichen Vorder- und Hintergrundfarbe einzeln festgelegt werden können. Wird nur ein Parameter angegeben, wird nur die Vordergrundfarbe gesetzt. Akzeptiert werden Werte von jeweils 0 bis 15, ohne die ”16-Farben Erweiterung” sind die Farben 0. . . 7 und 8. . . 15 identisch:

| **Wert** | 0/8 | 1/9 | 2/10 | 3/11 | 4/12 | 5/13 | 6/14 | 7/15 |
|---|---|---|---|---|---|---|---|---|
| **Farbe** | schwarz | blau | rot | magenta | grün | cyan | gelb | weiss |

Wenn die ”16-Farben Erweiterung” eingebaut wurde, stehen 8 zusätzliche Farben zur Verfügung, wobei die ersten 8 Farben insgesamt etwas dunkler werden.

| **Wert** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| **Farbe** | schwarz | blau | rot | magenta | grün | cyan | gelb | hellgrau |
| **Wert** | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| **Farbe** | dunkelgrau | hellblau | hellrot | hellmagenta | hellgrün | hellcyan | hellgelb | weiss |

```
01 COLOR 4,2
```

Hier wird die Zeichenfarbe „Grün auf rotem Grund” festgelegt, was allerdings nicht gerade besonders gut lesbar ist.

## 6.2 CLS

Mit dem CLS-Befehl wird der Bildschirm gelöcht. Beim Programmstart geschieht das automatisch. Es wird die eingestellte Hintergrundfarbe (beim Programmstart schwarz) verwendet.

## 6.3 BORDER B [BO]

Mit diesem Befehl kann die Randfarbe eingestellt werden, gültige Werte sind 0. . . 15.

## 6.4 POS Y,X

Mit dem POS-Befehl wird der Schreibcursor an die Stelle Y,X gesetzt. Nach jedem Löschen des Bildschirms wird der Cursor auf die Position 0,0 (links oben) gesetzt.

## 6.5 WRAP n

Mit dem WRAP-Befehl kann das Verhalten am Bildschirmende im Textmodus eingestellt werden. Mit **WRAP 0** wird der Bildschirm beim Erreichen des Bildschirmendes um eine Zeile nach oben gescrollt. Dies ist auch die Standardeinstellung beim Start. Mit **WRAP 1** wird beim Erreichen des Bildschirmendes der Cursor auf den Bildschirmanfang zurückgesetzt. BeimProgrammstart ist der Wert mit 0 vorbelegt.

## 6.6 PRINT [?]

Der PRINT-Befehl dient zur Ausgabe auf den Bildschirm oder auf die serielle/parallele Schnittstelle, in das Array oder auf die I2C-Schnittstelle. Zusätzlich kann die Ausgabe noch formatiert werden. Anstelle des PRINT Befehls kann auch (BASIC-üblich) ein Fragezeichen verwendet werden.

| | |
|---|---|
| ”TEXT” | der Text TEXT wird ausgegeben |
| #Expr | Festlegung des Ausgabekanals (s.u.) |
| !Expr | Stellt das Format ein (s.u.) |
| @Expr1,Expr2 | Cursorpositionierung (Y=Expr1, X=Expr2) |
| %Expr | Direkte Ausgabe eines Zeichens mit Zeichencode=Expr |
| &Expr | gibt Arrayelemente als Zeichen aus. Gestoppt wird bei einem Null-Byte oder wenn das Ende des Arrays erreicht ist. Funktioniert nur im Bytezugriff auf das Array. |
| Expr | gibt das Ergebnis des Ausdrucks mit dem eingestellten Format aus |
| ; | Trenner zwischen Ausdrücken |
| , | Trenner zwischen Ausdrücken, Leerzeichen bis zur nächsten durch 8 teilbaren Position |

Steht am Ende des PRINT-Befehls einer der beiden Trenner, wird kein Zeilenvorschub ausgeführt. Der Ausgabekanal legt fest, wohin die Zeichen ausgegeben werden. Bei Ausgabekanal >3 wird auf die I2C-Schnittstelle mit der Kanalnummer als Devicenummer ausgegeben. Dabei wird das niederwertigste Bit auf 0 gesetzt (Schreibmode).

| | |
|---|---|
| #0 | Ausgabe auf den Bildschirm, Defaulteinstellung |
| #1 | Ausgabe auf die serielle System-Schnittstelle |
| #2 | Ausgabe auf die parallele Schnittstelle |
| #3 | Ausgabe in das Array |
| #4 | Ausgabe auf die zweite serielle Schnittstelle (nur ATMega 644P) |
| #5... | Ausgabe über die I2C-Schnittstelle |

Das Format ist ein Wert zwischen 0 und 255, wobei die Bits folgende Bedeutung haben:

| | |
|---|---|
| Bit 7 | 0=dezimale Ausgabe, 1=hexadezimale Ausgabe |
| Bit 6 | 1 schaltet auf Großdarstellung um |
| Bit 4/5 | Kommaposition (0-3 Nachkommastellen), nur für dezimale Ausgabe |
| Bit 2/3 | Anzahl der ausgegebenen Ziffern (2-5), nur für dezimale Ausgabe |
| Bit 0/1 | 0=Kompakt, 1=fhrende Leerzeichen 2=führende Nullen 3=führende Nullen mit Vorzeichen |
| Bit 0 | 2/4 Zeichen bei hexadezimaler Ausgabe |

Bei Ausgabe auf den Bildschirm wird der Cursor auf die entsprechende Position gesetzt, bei Ausgabe auf die serielle oder parallele Schnittstelle wird die Positionierung ignoriert. Bei Ausgabe in das Array entspricht der erste Wert Array-Byteposition*256 und der zweite Wert der Array-Byteposition, ohne @ ist die Byteposition zu Beginn jedes PRINT-Befehls 0x0000. Wird über die I2C-Schnittstelle ausgegeben, so wird zuerst **0xff** und danach **xxxxxxxx** und **yyyyyyyy** gesendet. Die Änderung wude notwendig, um Zeichen z.B. auf extern angeschlossenen GLCD’s feiner positionieren zu können.

## 6.7 CBOX Y1,X1,Y2,X2

Mit dem CBOX-Befehl wird ein Rechteck gelöscht (mit der aktuellen Hintergrundfarbe).

```
01 CBOX 3,3,5,5
```

löscht oben links ein Quadrat von 3x3 Zeichen.

## 6.8 FRAME Y1,X1,Y2,X2

Zeichnet einen Textrahmen mit Umrandung.

```
01 COLOR 7,3:FRAME 0,0,22,29
```

Zeichnen einen Textrahmen mit voller Bildschirmgröße, violettem Untergrund und weißer Zeichenfarbe.

## 6.9 HFRAME Y1,X1,Y2,X2

Zeichnet einen Textrahmen mit Umrandung und einem ”Kopfbereich”, so wie z.B. auch das Hauptmenü aufgebaut ist.

```
01 COLOR 7,3:HFRAME 0,0,22,29
02 COLOR 3,7:?@1,1;"HFRAME"
```

Zeichnen einen Textrahmen wie im vorigen Beispiel, allerdings jetzt mit einem Kopfbereich und Text darin.

## 6.10 IBOX Y1,X1,Y2,X2

Mit dem IBOX-Befehl werden in einem Rechteck Vorder- und Hintergrundfarbe vertauscht. Die Zeichen selbst werden nicht verändert.

```
01 IBOX 0,0,0,0
```

invertiert das Zeichen in der linken oberen Ecke.

## 6.11 SCROLL n

In den Vieomodi 0, 4 und 6 gibt es ein Scrollfenster, in dem in alle 4 Richtungen gescrollt werden kann. Das Fenster beginnt bei Zeichen 2 in Zeile 2 und geht bis Zeichen 27 in Zeile 20. Somit gibt es um das Scrollfenster einen 2 Zeichen breiten Rand in dem z.B. Informationen stehen, die nicht mitgescrollt werden sollen.
Die freiwerdende Zeile wird mit Leerzeichen in der aktuellen Farbeinstellung gefüllt.

| n | Richtung |
|---|---|
| 0 | nach oben |
| 1 | nach rechts |
| 2 | nach unten |
| 3 | nach links |

```
10 SCROLL 2
```

verschiebt das Scrollfenster um 1 Zeichen nach unten.

## 6.12 GETCHAR variable,Y,X [GCH]

Ermittelt das Zeichen an der Position y,x und schreibt dieses in die Variable v.

```
11 GCH F,0,0
```

Schreibt den Zeichenwert von Position 0,0 in die Variable F.

## 6.13 GETATTR variable,Y,X [GAT]

Ermittelt das Attributbyte an der Position y,x und schreibt dieses in die Variable v.

```
12 GAT F,0,0
```

Schreibt den Attributwert von Position 0,0 in die Variable F. Dabei setzt sich das Attribut folgendermassen zusammen:

| Bit | Funktion |
|---|---|
| 0 | Hintergrundfarbe Bit 0 |
| 1 | Hintergrundfarbe Bit 1 |
| 2 | Hintergrundfarbe Bit 2 |
| 3 | Hintergrundfarbe Bit 3 |
| 4 | Vordergrundfarbe Bit 0 |
| 5 | Vordergrundfarbe Bit 1 |
| 6 | Vordergrundfarbe Bit 2 |
| 7 | Vordergrundfarbe Bit 2 |

Dabei ist zu beachten, daß die Bits wegen der Kompatibilität zu früheren Versionen und der 8-Farben Hardware im Bildspeicher physisch anders angeordnet sind:

| Bit | Funktion |
|---|---|
| 0 | Hintergrundfarbe Bit 3 |
| 1 | Hintergrundfarbe Bit 0 |
| 2 | Hintergrundfarbe Bit 1 |
| 3 | Hintergrundfarbe Bit 2 |
| 4 | Vordergrundfarbe Bit 3 |
| 5 | Vordergrundfarbe Bit 0 |
| 6 | Vordergrundfarbe Bit 1 |
| 7 | Vordergrundfarbe Bit 2 |

# 7 Tastatur

## 7.1 INPUT [INP]

Es können durch Kommata getrennt Zeichenketten und Variablen angegeben werden. Die Zeichenketten werden ausgegeben, die Variablen bewirken einen Eingabecursor. Falsch eingegebene Zeichen können mit der Backspace-Taste korrigiert werden. Es ist auch möglich, Ausdrücke einzugeben die dann berechnet werden. Das folgende Beispiel zeigt einen kleinen Rechner, das letzte Ergebnis ist in der Variable M gespeichert.

```
01 INPUT "AUFGABE: ",M
02 PRINT "ERGEBNIS:";M
03 GOTO 1
```

Gibt man als erstes „1+2" ein, erhält man „3". Gibt man danach „M*5" ein, erhält man „15".

## 7.2 CTEXT adr,anz

Kopiert den zuletzt bei Input eingegebenen Text in das Array byteweise ab Element **adr**. Als Endemarkierung wird ein Nullbyte angehängt. Mit dem 2.Parameter **anz** wird die Anzahl der maximal einzulesenden Bytes begrenzt. Dabei ist zu beachten, dass wegen dem angehängten Nullbyte effektiv **anz**+1 Bytes kopiert werden.

```
01 INPUT "Text: ",M
02 CTEXT 0,4
03 X=0
04 C=AR(X):IF C=0 THEN END
05 PRINT %C;:X=X+1:GOTO 4
```

Der nach der Eingabeaufforderung eingegebene Text wird eine Zeile tiefer wiederholt, wobei nur die ersten 4 Zeichen ausgegeben werden.

## 7.3 Die Keycodes

Neben den „normalen" ASCII-Werten für Ziffern, Zahlen und Satzzeichen liefern RKEY und WKEY auch Codes für Funktionstasten etc.

| **Code Hexadezimal** | **Code Dezimal** | **Taste** |
|---|---|---|
| $C0 | 192 | Power |
| $C1 | 193 | Sleep |
| $C2 | 194 | Wake |
| $E0 | 224 | Pos1 |
| $E1 | 225 | End |
| $E2 | 226 | Pfeiltaste nach links |
| $E3 | 227 | Pfeiltaste nach rechts |
| $E4 | 228 | Pfeiltaste nach oben |
| $E5 | 229 | Pfeiltaste nach unten |
| $E6 | 230 | Bild hoch |
| $E7 | 231 | Bild runter |
| $E8 | 232 | Einfg (Ins) |
| $E9 | 233 | Entf (Del) |
| $EA | 234 | Enter |
| $EB | 235 | Tabulator |
| $EC | 236 | Backspace |
| $ED | 237 | Esc |
| $F1…$FC | 241…252 | F1…F12 |

## 7.4 Die Funktion KEY

Diese Funktion liefert verschiedene Tastaturabfragen. Als Parameter wird die Art der Abfrage eingetragen. Bei KEY(0) bis KEY(2) zählen Shift, CTRL und ALT nicht als gedrückte Tasten.

| | |
|---|---|
| KEY(0) | Es wird die gedrückte Taste oder 0x00 zurückgegeben |
| KEY(1) | Es wird auf einen Tastendruck gewartet, Resultat ist der Code der gedrückten Taste. |
| KEY(2) | Es wird gewartet, bis keine Taste mehr gedrückt ist, zurückgegeben wird der Code der zuletzt gedrückten Taste. |
| KEY(3) | Es wird der Status der Shift- und CTRL Tasten zurückgeliefert. Belegung s.u. |
| KEY(4) | linke Shift-Taste liefert 1, linke Control-Taste liefert -1, beide zusammen 0 |
| KEY(5) | rechte Shift-Taste liefert 1, rechte Control-Taste liefert -1, beide zusammen 0 |
| KEY(6) | Taste Cursor links liefert -1, Taste Cursor rechts liefert 1 |
| KEY(7) | Taste Cursor nach unten liefert -1 Taste Cursor hoch liefert 1 |
| KEY(8) | Es wird der zuletzt gelesene Scancode zurückgeliefert |
| KEY(9…255) | Es wird gewartet, bis die Taste mit dem entsprechenden Keycode gedrückt wird |

Belegung der Bits beim Rückgabewert von KEY(3):

| Bit | Funktion |
|---|---|
| 0 | Linke Shift-Taste |
| 1 | Rechte Shift-Taste |
| 2 | Linke CTRL-Taste |
| 3 | Rechte CTRL-Taste |
| 4 | ALT-Taste |
| 5-7 | Interne Verwendung |

# 8 Grafik

## 8.1 Pseudografik im Textmodus

Für die Pseudografik wird jedes Zeichen in 4 „Pixel" aufgeteilt. Bei 23 Zeilen a 30 Zeichen ergibt sich so eine Arbeitsfläche von 60x46 Punkten. Die 4 Pixel eines Zeichens haben immer die gleiche Hinter- und Vordergrundfarbe.
Ist die Zeichenfarbe gleich der Hintergrundfarbe des zu ändernden Zeichens, so werden die Pixel nur gelöscht, Farbinformationen werden nicht verändert. Ist die Zeichenfarbe gleich der Vordergrundfarbe, werden nur Pixel gesetzt, Farbinformationen werden auch hier nicht verändert. Gibt es keine Übereinstimmung zwischen der Zeichenfarbe und der Vorder- und Hintergrundfarbe des betreffenden Zeichens, wird die Vordergrundfarbe neu gesetzt wobei bereits gesetzte Pixel zwangsläufig mit umgefärbt werden.
Zusätzlich zur Pseudografik im Textmodus gibt es noch 4 Grafikmodi, die sich in Auflösung und Anzahl der darstellbaren Farben unterscheiden. Allerdings lässt der verfügbare Speicher des Controllers keine sehr hohen Auflösungen zu. Ebenfalls aus Speicherplatzgründen ist es nicht möglich, den Monitor in den Grafikmodi aufzurufen.
Werden in den Modi 2,3 und 5 Vorder- und Hintergrundfarbe gesetzt, zeigen diese auf den Palettenindex und geben nicht die Farbe direkt an.

## 8.2 Videomode 1

Die Pixelauflösung beträgt 168x116 Pixel, wobei für jeweils 8x8 (am unteren Rand nur 4x8) Pixel Vorder- und Hintergrundfarbe eingestellt werden können. Ist die Zeichenfarbe gleich der Hintergrundfarbe an der zeichenposition, so werden die Pixel nur gelöscht, Farbinformationen werden nicht verändert. Ist die Zeichenfarbe gleich der Vordergrundfarbe, werden nur Pixel gesetzt, Farbinformationen werden auch hier nicht verändert. Gibt es keine Übereinstimmung zwischen der Zeichenfarbe und der Vorder- und Hintergrundfarbe des entsprechenden Bereiches, wird die Vordergrundfarbe neu gesetzt wobei bereits gesetzte Pixel zwangsläufig mit umgefärbt werden.

## 8.3 Videomode 2

Die Pixelauflösung beträgt 120x76 Pixel, jedes Pixel kann eine aus 4 über die Palette einstellbaren Farben annehmen.

## 8.4 Videomode 3

Die Pixelauflösung beträgt 84x58 Pixel, jedes Pixel kann eine aus 16 über die Palette einstellbaren Farben annehmen.

## 8.5 Videomode 5

Die Pixelauflösung beträgt 128x64 Pixel, jedes Pixel kann eine aus 2 über die Palette einstellbaren Farben annehmen. Dieser Modus dient hauptsächlich zur Emulation von Grafik-LCD mit der entsprechenden Auflösung.

## 8.6 VMODE n [VM]

Schaltet den Videomodus um. Dabei werden der Bildschirm gelöscht und die Textkoordinaten auf die linke obere Ecke gesetzt.

| Mode | Auflösung X | Auflösung Y | Darstellbare Farben |
|---|---|---|---|
| 0 | Text 30 | Text 23 | 16 Vordergrund und 16 Hintergrund |
| 1 | 168 | 116 | 2 aus 16 |
| 2 | 120 | 76 | 4 aus 16 |
| 3 | 84 | 58 | 16 aus 16 |
| 4 | (User Symbole) 30 | (User Symbole) 23 | 16 Vordergrund und 16 Hintergrund |
| 5 | 128 | 64 | 2 aus 16 |
| 6 | (User Symbole) 30 | (User Symbole) 23 | 16 für jedes Pixel |
| 7 | Text 30 | Text 2 | 16 Vordergrund und 16 Hintergrund |

Zusätzlich werden die im jeweiligen Modus genutzten Paletteneinträge vorbelegt.

## 8.7 PALETTE start,c1(,c2,c3,c4,c5,c6) [PAL]

Mit diesem Befehl wird die Farbpalette für die Videomodi 2 bis 3 und 5 eingestellt. Für den Modus 5 werden nur die Einträge 0 und 1 verwendet, für Modus 2 die Einträge 0 bis 3. Die Anzahl der zu setzenden Paletteneinträge ist auf 6 begrenzt, anderenfalls kommt es zu einer Fehlermeldung. Als Farbwerte müssen im Bereich (0...15) angegeben werden. Nach dem Umschalten des Videomodus sind die Paletteneinträge wie folgt vorbelegt:

| Index | Modus 1 | Modus 2 | Modus 3 |
|---|---|---|---|
| 0 | schwarz (0) | schwarz (0) | schwarz (0) |
| 1 | weiss (7) | rot (2) | blau (1) |
| 2 | — | cyan (5) | rot (2) |
| 3 | — | hellgrau (7) | magenta (3) |
| 4 | — | — | grün (4) |
| 5 | — | — | cyan (5) |
| 6 | — | — | gelb (6) |
| 7 | — | — | hellgrau (7) |
| 8 | — | — | dunkelgrau (8) |
| 9 | — | — | hellblau (9) |
| 10 | — | — | hellrot (10) |
| 11 | — | — | hellmagenta (11) |
| 12 | — | — | hellgrün (12) |
| 13 | — | — | hellcyan (13) |
| 14 | — | — | hellgelb (14) |
| 15 | — | — | weiss (15) |

## 8.8 PLOT Y,X(,C) [PL]

Mit dem PLOT-Befehl wir ein Punkt an der Stelle Y,X gesetzt. Bei Bedarf kann zusätzlich die Zeichenfarbe mit angegeben werden.

```
01 CLS:VMODE 1
02 FOR X=1 TO 167
03 V=SIN(3*X)/6
04 PLOT 55-V,X
05 NEXT
#
```

Zeichnet eine Sinuskurve (Videomode 1). Da nur einzelne Punkte gesetzt werden, ist die Linie nicht durchgängig.

## 8.9 DRAWTO Y2,X2(,C) [DTO]

Zeichnet eine Linie vom zuletzt gezeichneten Pixel nach Y2,X2 in der aktuellen Vordergrundfarbe. Bei Bedarf kann zusätzlich die Zeichenfarbe mit angegeben werden. Beim Programmstart ist das zuletzt gezeichnete Pixel als Y=0,X=0 definiert.

```
01 CLS:VMODE 1,PLOT 55,0
02 FOR X=1 TO 167
03 V=SIN(3*X)/6
04 DRAWTO 55-V,X
05 NEXT
#
```

Zeichnet eine Sinuskurve (Videomode 1). Da diesmal anstelle einzelner Punkte kurze Linienstücke gezeichnet werden, ist die Linie durchgängig. Der Plot-befehl in der ersten Zeile setzt die Anfangs-Koordinate für das erste Linienstück, fehlt sie, wird mit dem Zeichnen bei 0,0 begonnen.

## 8.10 DRAW Y1,X1,Y2,X2(,C) [DR]

Zeichnet eine Linie vo Y1,X1 nach Y2,X2 in der aktuellen Vordergrundfarbe. Bei Bedarf kann zusätzlich die Zeichenfarbe mit angegeben werden.

```
01 CLS:VMODE 1,PLOT 55,0
02 FOR X=1 TO 167
03 V=SIN(3*X)/6
04 DRAWTO 55-V,X
05 NEXT
06 DRAW 55,0,55,167
#
```

Hier wird die Null-Linie durch die gerade gezeichnete Sinuskurve gezogen.

## 8.11 BOX Y1,X1,Y2,X2(,C)

Mit dem BOX-Befehl wird ein Rechteck in der aktuellen Vordergrundfarbe gezeichnet. Bei Bedarf kann zusätzlich die Zeichenfarbe mit angegeben werden.

```
11 BOX 0,0,115,167
```

## 8.12 FBOX Y1,X1,Y2,X2(,C)

Mit dem FBOX-Befehl wird ein gefülltes Rechteck gezeichnet. Ist Y1=Y2 oder X1=X2 werden horizontale oder vertikale Linien gezeichnet. Bei Bedarf kann zusätzlich die Zeichenfarbe mit angegeben werden.

```
10 FBOX 0,0,0,100
```

zeichnet eine waagerechte Linie am oberen Bildschirmrand, der Wert 100 wird zur Laufzeit auf den Bildschirmbereich begrenzt, falls er auuserhalb des Bildschirms liegt.

## 8.13 CIRCLE Y,X,RY,RX(,C) [CI]

Zeichnet einen Kreis (eine Ellipse) mit dem Mittelpunkt Y,X und den Radien RY und RX in der aktuellen Vordergrundfarbe. Für Kreise muss RX=RY sein. Bei Bedarf kann zusätzlich die Zeichenfarbe mit angegeben werden.

```
10 CIRCLE 50,50,20,20
```

Zeichnet einen Kreis mit den Mittelpunktkoordinaten 50,50 und dem Radius 20.

## 8.14 FCIRCLE Y,X,RY,RX(,C) [FCI]

Zeichnet einen gefüllten Kreis (eine gefüllte Ellipse) mit dem Mittelpunkt Y,X und den Radien RY und RX in der aktuellen Vordergrundfarbe. Für Kreise muss RX=RY sein. Bei Bedarf kann zusätzlich die Zeichenfarbe mit angegeben werden.

```
10 FC 50,50,20,10
```

Zeichnet eine gefüllte Ellipse mit den Mittelpunktskoordinaten 50,50, dem Y-Radius 10 und dem X-Radius 20.

## 8.15 ARC Y,X,RY,RX,W1,W2(,C)

Zeichnet einen Kreis- oder Ellipsenausschnitt mit dem Mittelpunkt Y,X und den Radien RY und RX in der aktuellen Vordergrundfarbe. beginnend beim Winkel W1 bis zum Winkel W2. Die beiden Winkel werden in Grad angegeben, dabei sollte der Anfangswinkel kleiner als der Endwinkel sein, ansonsten braucht die Funktion recht lange. Für Kreisauschnitte muss RX=RY sein, bei Bedarf kann zusätzlich die Zeichenfarbe mit angegeben werden.

```
01 VM 1
02 ARC 50,50,20,20,0,270
```

Zeichnet ein Dreiviertel-Kreissegment mit den Mittelpunktkoordinaten 50,50 und dem Radius 20

## 8.16 PIE Y,X,RY,RX,W1,W2(,C)

Zeichnet einen abgeschlossenen Kreis- oder Ellipsenausschnitt (ein ”Tortenstück”)mit dem Mittelpunkt Y,X und den Radien RY und RX in der aktuellen Vordergrundfarbe. beginnend beim Winkel W1 bis zum Winkel W2. Die beiden Winkel werden in Grad angegeben, dabei sollte der Anfangswinkel kleiner als der Endwinkel sein, ansonsten braucht die Funktion recht lange. Im Gegensatz zu **ARC** werdennoch zusätzlich Linien von den beiden Endpunkten zum Mittelpunkt gezogen. Für Kreisauschnitte muss RX=RY sein, bei Bedarf kann zusätzlich die Zeichenfarbe mit angegeben werden.

```
01 VM 1
02 ARC 50,50,20,20,0,270,6
```

Zeichnet ein geschlossenes gelbes Dreiviertel-Kreissegment mit den Mittelpunktkoordinaten 50,50 und dem Radius 20

## 8.17 GETPIX variable,Y,X [GPX]

Ermittelt die Farbe des Pixels an der Stelle Y,X und speichert den Palettenindex (nicht die Farbe!) in der angegebenen Variablen.

```
10 GPIX P,0,0
```

Bestimmt den Palettenindex des Pixels in der oberen linken Ecke.

## 8.18 BCOPY [BC]

Der BCOPY Befehl ermöglicht es, rechteckige Blöcke oder Speicherbereiche innerhalb des Bildschirms oder ins/vom Array zu kopieren. Damit lassen sich z.B. Sprites im Grafikmodus oder Scrolling realisieren. Blöcke können in den Modi 1...3 von und an (fast) beliebigen Koordinaten kopiert werden, lediglich die Größe ist Beschränkungen unterworfen. Es können nur ganze Teilblöcke kopiert werden.

| **Videomode** | **Pixel je Teilblock** |
|---|---|
| 1 | 8x8 |
| 2 | 4x4 |
| 3 | 2x2 |

Im Videomodus 1 ist ein Teilblock 8x8 Pixel groß, um einen 16(V)x24(H) Pixel großen Block zu kopieren, müssen DX=2 und DY=3 sein, im Videomodus 2 DX=4 und DY=6 und im Videomodus 3 DX=8 und DY=12.
Der erste Wert gibt den Kopiermodus an, von diesem ist auch die Anzahl der restlichen Parameter abhängig.

| **Modus** | **Quelle** | **Ziel** | **zus.Parameter** |
|---|---|---|---|
| 1 | Bildspeicher | Bildspeicher | Y1,X1,DY,DX,Y2,X2 |
| 2 | Bildspeicher | Array | Y,X,DY,DX,ARRAYPOS |
| 3 | ARRAY | Bildspeicher | ARRAYPOS,Y,X |

Im Modus 2 wird geprüft, ob die Grenzen des Arrays überschritten werden, in diesem Fall wird dann sofort abgebrochen. Mit den folgenden Formeln lässt sich der erforderliche Speicherbedarf berechnen:

**Videomode 1** (8 * DX * DY) +2
**Videomode 2** (4 * DX * DY) +2
**Videomode 3** (2 * DX * DY) +2

DX und DY ist hierbei die Anzahl der Teilblöcke. Kopiermodus 3 sollte nur auf Arraypositionen angewendet werden, die auch sinnvolle Informationen enthalten. Diese können entweder durch Kopieren im Modus 2 oder auch durch „DATA” entstanden sein.
Die Pixelinformationen sind derart organisiert, dass in einem Byte je nach Videomode 8, 4 oder 2 Pixel codiert sind. dabei ist das niederwertigste Bit ganz links.

```
01 DATA 0,1,1,$FF,$81,$81,$81
02 DATA 6,$81,$81,$81,$FF
03 VMODE 1
04 BCOPY 3,0,0,0
05 WAIT 10
```

Das Programm zeichnet ein kleines Quadrat in die obere linke Bildschirmecke. Zuerst wird das Array gefüllt. In den beiden ersten Bytes stehen die Anzahl der Teilblöcke in Y und X-Richtung, danach folgen die Pixeldaten.
Zusätzlich gibt es noch die Modi 4-6, die auf die Orgaisation des Bildspeichers keine Rücksicht nehmen und zum schnellen Umkopieren gedacht sind.

| **Modus** | **Quelle** | **Ziel** | **zus.Parameter** |
|---|---|---|---|
| 4 | Bildspeicher | Bildspeicher | Quelladresse, Zieladresse, Anzahl der Bytes |
| 5 | Bildspeicher | Array | Quelladresse, Zieladresse, Anzahl der Bytes |
| 6 | ARRAY | Bildspeicher | Quelladresse, Zieladresse, Anzahl der Bytes |

Diese 3 Modi sind insbesondere in den Videomodi 4 und 6 mit den Benutzerdefinierten Zeichen interessant. Die Organisation des Bildspeichers in den verschiedenen Videomodi wird im Abschnitt **Der Bildspeicher** genauer beschrieben.

## 8.19 PRINT [?]

Der PRINT Befehl entspricht weitestgehend seinem Pendant im Textmodus. Unterschiedlich sind die Koordinatenangaben die hier anstelle von Zeichen in Pixeln angegeben werden. Und das Format-Attribut „Grossschrift" entspricht doppelter Zeichenhöhe und -breite. Ausserdem funktioniert der Zeilenumbruch nicht wie man es erwartet, es wird lediglich an den Anfang der nächsten Pixelzeile gesprungen.

## 8.20 FONT n

In den Grafikmodi 1…3 und 5 ist zusätzlich zum Standard-Font mit 6x10 Pixeln ein kleinerer Font mit 4x6 Pixeln verfügbar. Umgeschaltet wird mit **FONT 0** für den Standard Font und **FONT 1** für den Mini-Font.

# 9 Sprites

Sprites sind einzelne Zeichen oder einfache Zeichengruppen, die sich einfach an beliebiger Stelle auf dem Bildschirm anzeigen lassen. Dabei wird der ursprüngliche Bildschirminhalt an dieser Stelle „gemerkt" und beim Verschieben oder Löschen des Sprites wieder restauriert. In der aktuellen Version können (theoretisch) bis zu 85 Sprites definiert werden, die tatsächliche Anzahl hängt von der Größe und dem verfügbaren Platz im Array ab. Sprites funktionieren nur im Textmodus und nicht in den Grafikmodi.

## 9.1 Sprite definieren

Dafür gibt es keinen eigenständigen Befehl (mehr), da die Daten einfach im Array liegen (Byte-Bereich). Jedes Sprite besteht aus einem Header von 5 Bytes, den eigentlichen Daten und dem Backup-Bereich. Die Belegung der Bytes ist wie folgt:

| Header-Byte | Adresse | Funktion |
|---|---|---|
| 1 | N | das Kollisionsflag (0/2) |
| 2 | N+1 | Y-Position (am Anfang $ff oder -1) |
| 3 | N+2 | X-Position (am Anfang $ff oder -1) |
| 4 | N+3 | Y-Ausdehnung in Zeichen (dy) |
| 5 | N+4 | X-Ausdehnung in Zeichen (dx) |

Ist das Kollisionsflag mit 0 vorbelegt, wird das Sprite bei einer Kollision nicht gezeichnet. Ist es mit 2 vorbelegt, wird das Sprite bei einer Kollision gezeichnet. Danach folgen dx*dy Bytes für die darzustellenden Zeichen, gefolgt von dx*dy Bytes für die Attribute. Die Attribut-Bytes sind Vordergrundfarbe + 16 * Hintergrundfarbe, bei Addition von 128 bleibt beim Zeichnen die ursprüngliche Hintergrundfarbe erhalten. Danach folgen noch 2*dx*dy Bytes in denen der ursprüngliche Bildschirminhalt beim Zeichnen automatisch gespeichert wird.
Ein minimales Sprite (1 Zeichen) braucht also 5+4=9 Bytes, eins mit 2x2 Zeichen schon 5+16=21 Bytes. Die Werte für die X und Y Koordinaten sollten ausserhalb des Bildschirmbereiches liegen, da ansonsten beim erstmaligen Aufrufen des Sprites der Backup-Bereich des Sprites dorthin kopiert wird.

## 9.2 SPRITE n,y,x [SPR]

Zeichnet das Sprite von Array-Adresse n an die Position Y,X. War es bereits an einer anderen Stelle gezeichnet, wird es vorher gelöscht. Befindet sich an der Stelle wohin das Sprite gezeichnet wird bereits ein anderes Zeichen, dessen Position in der Zeichentabelle nicht ein Vielfaches von 16 ist, wird das Kollisionsflag gesetzt. Ist Bit 1 des Kollisionsflags nicht gesetzt, wird das Sprite nicht gezeichnet. Das Kollisionsflag wird nicht automatisch zurückgesetzt. Sprites dürfen nicht über den Bildschirmrand hinaus gezeichnet werden, sonst wird mit einer Fehlermeldung abgebrochen. Werden für X und oder Y Koordinaten ausserhalb des Bildschirms (X>29, Y>22) angegeben, wird das Sprite vom Bildschirm verschwinden.

```
01 DA 0,0,-1,-1,1,1,"*",$86
02 ?@10,10;"#"
03 X=RND(29): Y=RND(22)
04 SPRITE 0,Y,X
05 IF (AR(0)&1)=1 THEN END
05 WAIT 1
06 GOTO 2
```

Zuerst werden die Sprite-Daten in das Array geschrieben (in diesem fall ein gelbes Sternchen), dann wird noch ein Hindernis an Position (10,10) gezeichnet.
Im Anschluß hüpft das Sternchen hin und her, bis es auf das Hindernis trifft. Dieses bleibt stehen, das Sternchen ist "weg". Wird die erste Zeile abgeändert in:

```
01 DA 0,2,-1,-1,1,1,"*",$86
```

Dann bleibt das Sternchen stehen und das Hinedrnis ist nicht mehr zu sehen.

# 10 Audio

## 10.1 NOTE n [NO]

Ein Ton mit der Tonhöhe n (Halbtonschritte ab 220 Hz aufwärts) wird ausgegeben. Bei n=0 bis 62 werden Noten in einer „dunklen" Klangfarbe ausgegeben, bei n=64 bis 126 Noten mit einer „hellen" Klangfarbe. Bei n=63 und n=127 wird keine Note sondern Rauschen ausgegeben. Wird 128 zum Notenwert addiert, wird der Ton mit einer langsameren Hüllkurve ausgegeben.

```
01 FOR N=0 to 255:NOTE N
02 WAIT 5:NEXT
```

gibt nacheinander alle spielbaren Noten aus. Die folgende Tabelle ist eine Übersicht über die verschiedenen Möglichkeiten:

| Wert | Bedeutung |
|---|---|
| 0...62 | Noten in einer „dunklen" Klangfarbe mit schneller Hüllkurve |
| 63 | Rauschen mit schneller Hüllkurve |
| 64...126 | Noten in einer „hellen" Klangfarbe mit schneller Hüllkurve |
| 127 | Rauschen mit schneller Hüllkurve |
| 128...190 | Noten in einer „dunklen" Klangfarbe mit langsamer Hüllkurve |
| 191 | Rauschen mit langsamer Hüllkurve |
| 192...254 | Noten in einer „hellen" Klangfarbe mit langsamer Hüllkurve |
| 255 | Rauschen mit langsamer Hüllkurve |

## 10.2 NOTE n,v [NO]

Beim **NOTE** Befehl kann als zweiter Parameter die Lautstärke im Bereich von 0...15 angegeben werden. Der angegebene Lautstärkewert gilt dann nur für diese eine Note.

## 10.3 VOL n

Dieser Befehl dient zum steuern der Lautstärke, möglich sind Werte zwischen 0 und 15. Bei einem Wert von 0 wird die Tonausgabe nicht vollständig abgeschaltet, sollte aber kaum hörbar sein. Bei einem Wert von 15 wird die maximale Lautstärke erreicht. Der eingestellte Lautstärkewert gilt für Noten und für den Sequenzer, solange dort die Lautstärke nicht für die einzelnen Noten explizit vorgegeben wird.

## 10.4 PLAY start,stop,speed,loops

AVR-Chipbasic2 hat einen einfachen Sequenzer eingebaut. Dieser benutzt das Array als Notenspeicher und läuft im Hintergrund. Als Parameter werden die Nummer der Arrayzelle (Byte-bereich) mit der ersten zu spielenden Note, die Anzahl der zu spielenden Noten sowie die die Anzahl der Halbbilder (1/50s bei PAL, 1/60s bei NTSC) für einen "Tick" Die Noten zwischen Start- und End Array-Zelle werden zyklisch abgespielt, die Anzahl der Zyklen wird mit dem Wert von **loops** (Byte-Variable) bestimmt. Wenn dieser Wert größer als 127 ist wird der Zyklenzähler nicht nach jedem Durchlauf um 1 verringert, damit werden die Noten in einer Endlosschleife abgespielt. Jede Note besteht aus 2 Bytes im Array, im ersten Byte steht der Notenwert entsprechend dem **NOTE**-Befehl, im zweiten Byte die Anzahl der Ticks bis zur nächsten Note im unteren Nibble und die Lautstärke im oberen Nibble. Ist das obere Nibble 0, wird die aktuelle Lautstärke beibehalten.

Das folgende Beispiel spielt fortlaufend 3 Töne auf und ab...

```
01 DA 0,39,8,44,4,49,8,44,4
02 PLAY 0,4,10,128
03 GOTO 3
```

Wird die letzte Zeile weggelassen hört man nichts oder maximal einen einzigen Ton, da beim Programmende der Sequenzer sofort wieder gestoppt wird. Dieser Befehl sollte nur dann verwendet werden, wenn sich auch gültige Notenwerte im Array befinden.

## 10.5 PSTAT(mode)

Diese Funktion liefert Aufschluss darüber, was der Sequenzer gerade „macht”. Ist der Byte-Parameter mode<255, dann wird der Zyklenzähler auf den Wert von **mode** gesetzt, mode=255 liefert nur den aktuellen Zyklenzähler zurück.

| **Wert** | **Bedeutung** |
|---|---|
| 0 | Sequenzer ist gestoppt |
| 1 | Sequenzer läuft noch bis zum Ende des aktuellen Zyklus und wird dann stoppen |
| 2...127 | Sequenzer läuft, Noten werden gespielt, nach jedem Zyklus wird der Zyklenzähler um 1 verringert |
| 128...255 | Sequenzer läuft, Noten werden gespielt, Zyklenzähler ändert sich nicht |

# 11 Zeit

## 11.1 WAIT n

Mit dem WAIT-Befehl wird N*0,1 Sekunden gewartet. N kann wieder ein beliebiger Ausdruck sein. Im Videomode 7 kann es dabei durchaus sein, dass der Treiber die Zeit nicht weiterzählt und die Funktion nicht wieder zurückkehrt. Das hängt aber von der konkreten Treiberimplementation ab.

```
10 WA 10
```

wartet 1 Sekunde, bis das Programm fortgesetzt wird.

## 11.2 SYNC n

Mit dem SYNC-Befehl wird auf N Bildsynchronimpuse gewartet. N kann wieder ein beliebiger Ausdruck sein. Im Videomode 7 kann es dabei durchaus sein, dass der Treiber die Zeit nicht weiterzählt und die Funktion nicht wieder zurückkehrt. Das hängt aber von der konkreten Treiberimplementation ab.

```
10 SYNC 300
```

wartet 6 Sekunden, bis das Programm fortgesetzt wird. Das gilt nur für PAL, bei NTSC wird nur ca. 5 Sekunden gewartet.

## 11.3 TSET n

Der interne Timer (10Hz) wird auf den Wert n gesetzt.

```
10 TSET 0
```

setzt den Timer auf 0.

## 11.4 TGET V

Der interne Timer (10Hz) wird ausgelesen und in die Varaiable V gespeichert. Im Videomode 7 kann es dabei durchaus sein, dass der Treiber die Zeit nicht weiterzählt und die Funktion die bei **TSET** gesetzte Zeit zurückliefert. Das hängt aber von der konkreten Treiberimplementation ab.

```
10 TGET Z
```

Damit kann z.B. die Laufzeit von Programmen bestimmt werden.

## 11.5 ONSYNC L,N

Ähnlich dem Timer-Interrupt bei einem Mikrocontroller können mit **ONSYNC** zeitgesteuerte Unterprogrammaufrufe eingerichtet werden. Dabei ist L die Zeilennummer und N die Anzahl der Bildsynchronimpulse zwischen zwei Aufrufen. Der Wert N für die Wartezeit kann maximal 255 betragen, L sollte eine gültige Zeilennummer sein, bei L=0 werden die automatischen Aufrufe gestoppt. Das aufgerufene Unterprogramm muß mit **RETURN** enden und darf nicht länger dauern als die Zeit zwischen zwei Aufrufen. Intern wird bei jedem Bildsynchronimpuls ein Zähler mit dem Wert N verglichen und bei Gleichheit ein Flag gesetzt. Im BASIC-Interpreter wird dann vor der Ausführung jedes Statements dieses Flag überprüft und falls es gesetzt ist ein **GOSUB** zur angegebenen Zeilennummer ”eingeschoben”. Im Videomode 7 kann es dabei durchaus sein, dass der Treiber die Zeit nicht weiterzählt und angegebene Zeile nie aufgerufen wird. Das hängt aber von der konkreten Treiberimplementation ab.

```
01 ONSYNC 5,50
02 ? "Border:";A,
03 GOTO 2
04
05 A=A+1:IF A=8 THEN A=0
06 BORDER A:RETURN
```

Dieses Programm ändert alle Sekunde die Randfarbe, während der aktuelle Border-Wert ständig als Text ausgegeben wird.

# 12 Ein- und Ausgabe

## 12.1 DIR n

setzt die I/O-Richtung der 8 Portpins an der parallelen Schnittstelle. Eine 0 bedeutet Eingang, eine 1 Ausgang.

```
10 DIR $F0
```

Pin D0-D3 werden als Eingang, D4-D7 als Ausgang konfiguriert.
Die Funktion kann durch Treiber auf Programmplatz 8 deaktiviert werden, in diesem Fall wird mit der Meldung ”IO disabled” das Programm abgebrochen.

## 12.2 OUT n,b

Hier gibt es mehrere Funktionen, n=0..7 setzt einzelne Bits, n=$100 bis $1ff erlaubt es, mehrere Bits gleichzeitig zu setzen bzw. zu löschen. dabei dienen die unteren 8 Bits von n als Maske.
Beispiele:

```
10 OUT 7,0
```

Setzt D0 auf 0-Pegel.

```
10 OUT $1F0,$F0
```

Setzt D3-D7 auf 1-Pegel.
Die Funktion kann durch Treiber auf Programmplatz 8 deaktiviert werden, in diesem Fall wird mit der Meldung ”IO disabled” das Programm abgebrochen.

Mit n=512 ($200) wird die SPI-Schnittstelle konfiguriert. Ist b=255 ($ff), wird die SPI-Schnittstelle auf Standardwerte und zurückgesetzt. Bei anderen Werten wird die SPI-Schnittstelle nach folgendem Schema konfiguriert: Die Bits 0-5 werden in des SPCR-Register des Mikrocontrollers geschrieben, Bit 6 in das SPSR-Register.

| **Bit** | **Bit=0** | **Bit=1** |
|---|---|---|
| 0 | siehe SCK-clockteiler | siehe SCK-clockteiler |
| 1 | siehe SCK-clockteiler | siehe SCK-clockteiler |
| 2 | Datenübernahme mit Vorderflanke des Taktes | Datenübernahme mit Rückflanke des Taktes |
| 3 | Takt in Ruhelage LOW | Takt in Ruhelage HIGH |
| 4 | SPI Slave | SPI Master |
| 5 | Bit 7 wird zuerst gesendet | Bit 0 wird zuerst gesendet |
| 6 | Bitrate x 1 | Bitrate x 2 |

Über den SCK Clockteiler kann im Master-Mode die SPI-Clockfrequenz eingestellt werden. Dabei sind folgende Werte möglich:

| **Bit 6** | **Bit 1** | **Bit 0** | **Frequenz** |
|---|---|---|---|
| 0 | 0 | 0 | 5 MHz |
| 0 | 0 | 1 | 1,25 MHz |
| 0 | 1 | 0 | 312,5 KHz |
| 0 | 1 | 1 | 156,25 KHz |
| 1 | 0 | 0 | 10 MHz |
| 1 | 0 | 1 | 2,5 MHz |
| 1 | 1 | 0 | 625 KHz |
| 1 | 1 | 1 | 312,5 KHz |

Die Clockfrequenz von 10MHz kann nur im Master-Mode verwendet werden, wobei der Slave-Mode nur bedingt einsetzbar ist, da die CPU während der Bildausgabe (ca. 70% der Zeit) die SPI-Schnittstelle nicht bedienen kann.

Direkter Zugriff auf die I/O Register des ATMega644 steht im Bereich n=$400 bis $4ff zur Verfügung. Hier sollte man Vorsicht walten lassen, es ist damit auch möglich das System zu Stillstand zu bringen.

```
01 OUT $4B9,92
```

Setzt das TWBR Register (I2C-Bitrate) auf 92 (100KHz bei Vorteiler 1).

Bei Adressen im Bereich n=$0800 bis $ffff wird auf eine Funktion im Treiber (Programmplatz 8) zugegriffen. Ist kein entsprechender Treiber geladen ist dies nur eine Dummy-Funktion, die das Fehlerregister setzt.

## 12.3 Die Funktion IN(n)

Hier gibt es mehrere Möglichkeiten, n=0..7 liest einzelne Bits des I/O-Ports, n=$1xx erlaubt es, mehrere Bits gleichzeitig zu lesen wobei xx als Bitmaske dient.
Eine erweiterte Funktionalität steht im Bereich n=$400 bis $4ff zur Verfügung. Hier kann direkt auf die I/O Register des ATMega644 zugegriffen werden.

```
01 D=IN($426)
```

liest das PIN-Register (Port-Input) von Port C in die Variable D ein.

Bei Adressen im Bereich n=$0800 bis $ffff wird auf eine Funktion im Treiber (Programmplatz 8) zugegriffen. Ist kein entsprechender Treiber geladen oder benutzt dieser die angegebene Adresse nicht, wird ein Fehler (NO IO DRIVER) generiert.

## 12.4 Die Funktion ADC(n)

Diese Funktion dient zum Einlesen von Analogwerten über den integrierten Analog-Digital-Wandler. Auch hier gibt es mehrere Möglichkeiten, n=0..7 werden die I/O-Leitungen D0...D7 mit der internen Referenzspannung von 2,56 Volt eingelesen. Im Parameterbereich n=$1xx ist es möglich, das ADMUX Register direkt zu steuern. Damit lassen neben unterschiedlichen Referenzen auch die differentiellen Modes nutzen.

## 12.5 Die Funktion SPI(n)

Mit dieser Funktion wird ein Byte über die SPI-Schnittstelle ausgegeben und der eingelesene Wert zurückgegeben.

```
10 IF SPI(67)<>67 PRINT "ERROR"
```

Wenn MOSI mit MISO verbunden und die SPI-Schnittstelle aktiviert ist, sollte kein "ERROR" angezeigt werden.

## 12.6 SPISEL n

Mittels diesem Signal lässt sich der SPI-SS Portpin schalten. Ab Version 1.44 hat sich die Auswertung des Parameters geändert, damit die SPI-Multiselect Erweiterung genutzt werden kann. Dazu wird bei Werten < 255 das Byte mit deaktiviertem SS Signal ausgegeben und erst danach das SS Signal aktiviert.

- **SPISEL nn(0...254)** aktiviert das SS Signal (LOW-Pegel an PORTB.4)
- **SPISEL 255** deaktiviert das SS Signal (HIGH-Pegel an PORTB.4)

```
01 SPISEL $F7
02 A=SPI($FF)
03 SPISEL $FF
```

Mit diesem Befehl wird die Selectleitung aktiviert, 0xFF über die SPI-Schnittstelle ausgegeben und danach die Selectleitung wieder deaktivert. Mit SPI-Multiselect Erweiterung wird ser SS-Ausgang an Bit 3 aktiviert.

# 13 Kommunikation

## 13.1 SPUT a,b...

ein oder mehrere Bytes werden an die serielle Schnittstelle ausgegeben.

```
01 SPUT 65,10
```

gibt ein großes "A" und einen Zeilenvorschub an die serielle Schnittstelle aus.

## 13.2 SGET V

Ein Zeichen von der seriellen Schnittstelle wird eingelesen und in die Varaiable V gespeichert. Diese Funktion ist aber nur beim ATMega644P nutzbar, und nur wenn der Eingangspin der seriellen System-Schnittstelle auf PD1 liegt. Ansonsten wird mit einer Fehlermeldung abgebrochen. Die Funktion kehrt erst zurück wenn das Zeichen eingelesen wurde, alternativ kann mit der **ESC** Taste abgebrochen werden. In diesem Fall wird der Wert -1 zurückgegeben.

```
01 SGET C
```

wartet auf ein Zeichen von der seriellen Schnittstelle und speichert die in die Variable C

## 13.3 BAUD n

Hiermit kann die Bitrate der seriellen Schnittstelle eingestellt werden. Diese Einstellung gilt aber nur zur Laufzeit der Programmes und wird nach Programmende wieder durch die Systemeinstellung überschrieben. Es sind folgende Bitraten möglich:

| n | Bitrate (Bps) |
|---|---|
| 0 | 1200 |
| 1 | 2400 |

```
01 BAUD 1
```

setzt die Bitrate auf 2400Bps.

## 13.4 ESPUT a,b...

ein oder mehrere Bytes werden an die zweite serielle Schnittstelle ausgegeben. Diese Funktion ist aber nur beim ATMega644P nutzbar, und nur wenn der Eingangspin der seriellen System-Schnittstelle auf PD1 liegt. Ansonsten wird mit einer Fehlermeldung abgebrochen.

```
01 ESPUT 65,66,10
```

gibt einen Zeilenvorschub an die zweite serielle Schnittstelle aus.

## 13.5 ESGET V

Ein Zeichen von der zweiten seriellen Schnittstelle wird eingelesen und in die Varaiable V gespeichert. Diese Funktion ist aber nur beim ATMega644P nutzbar, und nur wenn der Eingangspin der seriellen System-Schnittstelle auf PD1 liegt. Ansonsten wird mit einer Fehlermeldung abgebrochen. Die Funktion kehrt erst zurück wenn das Zeichen eingelesen wurde, alternativ kann mit der **ESC** Taste abgebrochen werden. In diesem fall wird der Wert -1 zurückgegeben.

```
01 ESGET C
```

wartet auf ein Zeichen von der zweiten seriellen Schnittstelle und speichert die in die Variable C

## 13.6 EBAUD n

Hiermit können die Bitrate und weitere Parameter der zweiten seriellen Schnittstelle des ATMega644P eingestellt werden. Diese Funktion ist aber nur nutzbar, wenn der Eingangspin der seriellen System-Schnittstelle auf PD1 liegt. Ansonsten wird mit einer Fehlermeldung abgebrochen.
Die Bits 0...2 sind für die Bitrate verantwortlich, dabei sind folgende Bitraten möglich:

| bbb | Bitrate (Bps) |
|---|---|
| 000 | 1200 |
| 001 | 2400 |
| 010 | 4800 |
| 011 | 9600 (default) |
| 100 | 19200 |
| 101 | 31250 (MIDI) |
| 110 | 38400 |
| 111 | 57600 |

Mit Bit 4 kann die Anzahl der Stopp-Bits eingestellt werden:

| b | Stopp-Bits |
|---|---|
| 0 | 1 Stopp-Bit |
| 1 | 2 Stopp-Bits |

Mit den Bits 5...7 lässt sich eines von 8 verschiedenen Datenformaten wählen:

| bbb | Format |
|---|---|
| 000 | 8 Bit, kein Parity-Bit |
| 001 | 8 Bit, gerade Parity |
| 010 | 6 Bit, kein Parity-Bit |
| 011 | 8 Bit, ungerade Parity |
| 100 | 7 Bit, kein Parity-Bit |
| 101 | 7 Bit, gerade Parity |
| 110 | 5 Bit, kein Parity-Bit |
| 111 | 7 Bit, ungerade Parity |

Die Modi 0x00...0x07 sind dabei kompatibel zu früheren Versionen. Weitere Einstellmöglichkeiten ergeben sich natürlich, wenn über **OUT** Befehle direkt auf die Register des USART zugegriffen wird.

```
01 EBAUD 5
```

setzt die Bitrate der zweiten seriellen Schnittstelle auf 31250Bps (MIDI), Format 8/N/1.

## 13.7 PUMP n

Schaltet die Ladungspumpe für die serielle Schnittstelle aus (n=0) oder ein (n=1).

```
01 PU 1
```

schaltet die Ladungspumpe ein. Die Ladungspumpe ist auch per default nach dem Start eingeschaltet.

## 13.8 XSEND n,a

gibt einen Datenblock aus dem Array über das XMODEM Protokoll aus. N ist dabei die Blocknummer und a die Arrayposition (0...640). Um das ACK/NAK Handling muss man allerdings selbst kümmern.

```
01 SGET C:IF C<>21 THEN GOTO 1
02 FOR Z=0 TO 5:A=Z*128
03 XSEND Z,A
04 SGET C:IF C=21 THEN GOTO 3
05 NEXT:SPUT 4
```

Dieses Programm sendet den kompletten Array-Inhalt via X-Modem an ein anderes Gerät.

## 13.9 XREC(a)

liest einen Datenblock über das XMODEM Protokoll in das Array ein. A ist dabei die Arrayposition (0...640). Um das ACK/NAK und Error Handling muss man allerdings wieder selbst kümmern.

```
01 SPUT 21
02 FOR Z=0 TO 5:A=Z*128
03 C=XREC(A):SPUT 6:NEXT
04 SGET C: SPUT 6
```

Dieses Programm liest den kompletten Array-Inhalt via X-Modem von einem anderen Gerät. Bei einem Empfangsfehler wird einfach abgebrochen. Dies könnte aber mit **ONERR** abgefangen werden.

## 13.10 ICOMM adr,start,num [IC]

Generische I2C Routine. Der Erste Parameter **adr** gibt die Slave-Adresse an. Gleichzeitig wird mit Bit 0 festgelgt, ob geschrieben (0) oder gelesen (1) werden soll. Die beiden anderen Parameter geben die Startadresse im Array und die Anzahl der zu übertragenden Bytes an.

```
01 DATA HI(B),LO(B),LO(A),HI(A):ICOMM \$a0,0,4
02 SYNC 1
```

Schreibt den Wert der Variable A an die Adresse B eines I2C-EEPROMs (>24C16) mit der Adresse 0. Dabei ist auf die Reihenfolge der Bytes zu achten, da Beim I2C EEPROM zuerst das High-Byte der Adresse und danach das Low-Byte der Adresseübertragen werden muss. Um eventuelle Wartezeiten nach der Aktion muss man sich selbst kümmern, Ein Schreiben in den EEPROM mit darauffolgendem Lesen gibt mit hoher Sicherheit einen I2C-Fehler,**SYNC 2** schafft eine Pause von mindestens 20ms. Wird die Anzahl der zu schreibenden Bytes auf 0 gesetzt, werden solange Bytes aus dem Array ausgegeben, bis 0x00 erkannt wird. Im Lesemodus bedeutet eine 0 als Anzahl, dass als erstes die Anzahl der zu lesenden Bytes eingelesen wird. Ein I2C-Slave, der nichts senden möchte, muss also nur ein 0x00 zurücksenden, welches mit einem **ACK** beantwortet wird. Ein Dummy-Byte wie in vorherigen Versionen ist nicht mehr nötig.

# 14 Speicher

## 14.1 EPOKE adr,dat [EP]

Speichert ein Datenbyte in den internen EEPROM. Als Adressen sind 0...1999 möglich.

```
02 EPOKE V,$12
```

speichert den Wert 18 an die Adresse V im internen EEPROM.

## 14.2 EPEEK(adr)

Die Funktion liest ein Byte aus dem internen EEPROM. Als Adressen sind 0...1999 möglich.

```
02 W=EPEEK(V)
```

liest den Wert von EEPROM-Adresse V (internes EEPROM) und speichert diesen in der Variable W.

An die I2C-Schnittstelle kann ein zusätzlichs EEPROM (derzeit nur 24C64...24C512) angeschlossen werden. Bei anderen Typen lässt sich die gleiche Funktionalität (etwas umständlicher) über **ICOMM** realisieren.

## 14.3 XPOKE adr,dat [XP]

Speichert ein Byte im externen EEPROM, die Baustein-Adresse ist im Konfigurationsmenü festgelegt und nur dort einstellbar. Die Baustein-Adressen 6 und 7 sind für SRAM/FRAM vorbehalten, da hier nicht gewartet wird, bis der Schreibvorgang abgeschlossen ist.

```
01 XPOKE 100,C
```

speichert den Wert der Variablen C an die Adresse 100.

## 14.4 XPEEK(adr)

Die Funktion liest ein Byte aus dem externen EEPROM. Als Adressen sind 0...$FFFF möglich.

```
02 W=XPEEK(100)
```

liest den Wert von EEPROM-Adresse 100 /externes EEPROM) und speichert diesen in der Variable W.

## 14.5 VPOKE adr,dat [VP]

Dieser Befehl schreibt ein Byte in den Bildspeicher. Als Adressen sind 0...2759 möglich.

```
01 VPOKE 0,66:VPOKE 690,$08
```

schreibt ein schwarzes "B" auf grünem Grund in die linke obere Bildschirmecke.

## 14.6 VPOKE adr,arrayadr,num [VP]

Werden bei **VPOKE** drei Parameter angegeben, ist es möglich ganze Byte-Sequenzen auf einmal in den Bildspeicher zu schreiben. Als Adresse wieder sind 0...2759 möglich. Zweiter Parameter ist eine Arrayposition im Bytebereich, ab der die zu schreibenden Daten stehen. Als dritter Parameter wird angegeben, wie oft die Sequenz in den Bildspeicher geschrieben wird.
Im Array selbst stehen die Daten paarweise als Bytes. Das erste Byte gibt den zu schreibenden Wert an und das zweite Byte die danach zu überspringenden Bytes. Der zweite Wert kann zwischen 0 und 127 liegen, ist Bit 7 gesetzt (Wert > 127) wird 128 abgezogen, der Offset normal ausgeführt und danach die Sequenz beendet.

```
01 DA 0,65,0,66,0,67,27+128
02 VP 0,0,10
```

schreibt 10x untereinander "ABC".

## 14.7 VPEEK(adr)

Die Funktion liest ein Byte aus dem Bildspeicher. Als Adressen sind 0...2759 möglich.

```
01 ? "A"
02 ? VPEEK(0)
```

sollte in der ersten Zeile ein ”A” und in der zweiten Zeile 65 (das ist der ASCII-Wert von ”A”) ausgeben.

## 14.8 PAGE num

Mit dieser Funktionlässt sich die Position des Array-Bereiches 768...1023 im externen Speicher festlegen. Als Werte sind 0...255 möglich, das gibt einen theoretisch nutzbaren Speicherbereich von 64 Kilobytes.

```
01 PAGE 0
```

setzt das Fenster auf die ersten 256 Bytes im externen RAM.

## 14.9 CCHAR src-char,dest-char

Kopiert im Videomodus 4 einzelne Zeichen vom eingebauten ROM-Zeichensatz in den benutzerdefinierten Zeichensatz. **Src-char** ist dabei die Zeichennummer im ROM-Zeichensatz (0...255) und **dest-char** die Zeichennummer im RAM Zeichensatz (0...127). Sollen mehrere Zeichen kopiert werden, muß dies z.B. in einer Schleife geschehen.

## 14.10 CCHAR src-char,dest-char,color

Kopiert im Videomodus 6 einzelne Zeichen vom eingebauten ROM-Zeichensatz in den benutzerdefinierten Zeichensatz. **Src-char** ist dabei die Zeichennummer im ROM-Zeichensatz (0...255) und **dest-char** die Zeichennummer im RAM Zeichensatz (0...63). Der Wert **color** ergibt sich aus Vordergrundfarbe+16*Hintergrundfarbe für die Kopie. Sollen mehrere Zeichen kopiert werden, muß dies z.B. in einer Schleife geschehen.

# 15 Dateisystem-Funktionen

Wenn Sie bereits mit Personal Computern vertraut sind, werden Sie jetzt ein klein wenig umdenken müssen, da das Dateisystem von AVR-ChipBasic2 ein bisschen anders aufgebaut ist. Voraussetzung um mit Dateien arbeiten zu können ist ein Dataflash-Modul, welches in den Programmier-Port gesteckt wird. Je nach Typ können bis zu 0,5 oder 1 Megabyte Daten gespeichert werden. Natürlich ist es immer etwas weniger, denn die Daten wollen auch organisiert werden, was wiederum Speicherplatz auf dem Medium beansprucht. Um mit dem Dataflash-Modul arbeiten zu können, muss dieses „formatiert" werden. Das geschieht im DFLASH Menü, welches über das Hauptmenü erreichbar ist. Im Gegensatz zu flüchtigen Speichern (RAM) oder Festplatten sind Schreibzugriffe auf Flash oder EEPROM Speicher nur in begrenzter Anzahl möglich. Meistens liegt diese Zahl zwischen 10000 und 1 Million, je nach Speichertyp und Hersteller. Um die Speicherblöcke möglichst gleichmäßig „abzunutzen" enthalten sie einen Zähler, der bei jedem Schreibvorgang um 1 erhöht wird. Wenn eine Datei erzeugt wird, werden nun Blöcke mit möglichst niedrigem Zählerstand verwendet und auch beim häufigen Schreiben auf den gleichen Block kann es passieren, dass ein neuer Block mit niedrigerem Zählerstand gesucht wird. Darum muss man sich aber nicht kümmern, das Ganze passiert völlig transparent, kann aber zu kurzzeitigen Verzögerungen im Programmablauf führen. Die Dataflash-Module sollten nicht allzu häufig formatiert werden, da dabei die Informationen über den „Abnutzungsgrad" der Blöcke verloren gehen.

Der freie Speicherplatz ist in Blöcke (Pages) von 256 Bytes aufgeteilt, die wiederum maximal 128/256 Dateien (Files) zugeordnet werden können. Jede Datei kann minimal 1 und maximal 128 Datenblöcke enthalten was Dateigrößen von 256 Bytes bis 32 Kilobytes erlaubt. Dateinamen, wie man sie vom PC her kennt, gibt es hier aber nicht. Damit Dateien in der Fileselect-Box unterscheidbar sind, haben sie einen fest eincodierten Typ und die ersten 12 Bytes werden als Dateiname angezeigt. Wenn USR Dateien angelegt werden, sollte auch in die ersten 12 Bytes eine mehr oder weniger eindeutige Identifizierung geschrieben werden. Der Einfachkeit halber kann man auch den ersten Block der Datei nur für den Namen reservieren, auch wenn dadurch Speicherplatz verschenkt wird. Ansonsten wird einfach mit der Dateinummer gearbeitet.

Zusätzlich gibt es noch die Befehle RREAD und RWRITE, mit denen Pages im Dataflash direkt gelesen oder geschrieben werden können.

## 15.1 FTYPE(n)

Die Funktion FTYPE liefert entweder den Flashtyp (n=-1) oder der Dateityp der Datei n. Für n=-1 ist das Resultat

| **Wert** | **Bedeutung** |
|---|---|
| 0 | kein Dataflash angeschlossen |
| 2048 | 4MBit Dataflash angeschlossen |
| 4096 | 8MBit Dataflash angeschlossen |

für 0<=n<=255 ist das Resultat (derzeit)

| **Wert** | **Bedeutung** |
|---|---|
| 252 | Datei ist nicht belegt |
| 16 | BASIC-Programm |
| 18 | Backup-Datei |
| 20 | USR-Datei |
| 22 | Chip8-Datei |
| 24 | AVR-Binärdatei |
| 26 | GPS-Positionsliste |
| 28 | Sampling data (Messdaten) |
| 30 | Logikblock-Definition |
| 31 | Image-Datei |
| 32 | Screenshot-Datei |
| 34 | S12(X)-Binärdatei |
| -1 | Dateinummer existiert nicht |

## 15.2 FSIZE(n)

Die Funktion FSIZE(n) liefert entweder die Zahl der verfügbaren Files/Pages oder die Größe einer datei in 256Bytes-Pages.

| n | Rückgabewert |
|---|---|
| -1 | Anzahl der freien Files |
| -2 | Anzahl der freien Pages |
| 0...255 | Anzahl der Pages in der Datei oder 0 (nicht belegt) |

## 15.3 FCREATE f,p [FCR]

Erzeugt das USR-File f mit p Pages. Die Anzahl der Pages kann dabei zwischen 1 und 128 liegen was einer Dateigröße von 256 Bytes...32 Kbytes entspricht. Da FCREATE im Fehlerfall abbricht, sollte vorher mit FTYPE und FSIZE sichergestellt werden, dass die Datei auch erzeugt werden kann. Es werden immer Files vom Typ USR (Dateityp 0x14) erzeugt, ausser es wird als dritter Parameter ein anderer Dateityp angegeben.

```
40 'create file F with P pages
41 'R=F or -1 if failed
42 R=-1:IF P<1 THEN RETURN
43 IF P>128 THEN RETURN
44 IF FTYPE(-1)=0 THEN RETURN
45 IF FTYPE(F)<>0 THEN RETURN
46 IF FSIZE(-2)<P THEN RETURN
47 FCREATE F,P: R=F: RETURN
```

Die Subroutine erzeugt das File F mit P Pages, wenn dies möglich ist. In der Variable R steht dann das Resultat: entweder die Filenummer F oder -1, falls das Anlegen des Files nicht möglich ist. Zu beachten ist dabei, daß für diese Funktion das letzte Drittel des Array (Zelle 512-767) temporär genutzt wird und dort befindliche Daten überschrieben werden.

## 15.4 FWRITE f,p,a [FWR]

Schreibt 256 Bytes aus dem Array in die Page P von File F. Der dritte Parameter bestimmt die Arrayposition:

| a | Arraybereich |
|---|---|
| 0 | 0...255 |
| 1 | 256...511 |
| 2 | 512...767 |

Da FWRITE im Fehlerfall abbricht, sollte vorher mit FTYPE und FSIZE sichergestellt werden, dass die Page auch geschrieben werden kann.

```
50 'write array 0...255 to
51 'file F in page P
52 R=-1:IF P<1 THEN RETURN
53 IF FTYPE(F)<>20 THEN RETURN
54 IF P>FSIZE(F) THEN RETURN
55 FWRITE F,N,0: R=F: RETURN
```

Die Subroutine schreibt den Inhalt der Arrayzellen 0...255 in die Page P von File F. In der Variable R steht das Resultat: entweder die Filenummer F oder -1, falls das Schreiben der Page nicht möglich ist.

## 15.5 FREAD f,p,a [FRD]

Liest 256 Bytes aus Page P von File F und schreibt sie in das Array. Der dritte Parameter bestimmt wieder die Arrayposition:

| a | Arraybereich |
|---|---|
| 0 | 0...255 |
| 1 | 256...511 |
| 2 | 512...767 |

Da FREAD im Fehlerfall abbricht, sollte vorher mit FTYPE und FSIZE sichergestellt werden, dass die Page auch gelesen werden kann.

```
50 'read page P of file F to
51 'array 256...511
52 R=-1:IF P<1 THEN RETURN
53 IF FTYPE(F)<>20 THEN RETURN
54 IF P>FSIZE(F) THEN RETURN
55 FREAD F,N,1: R=F: RETURN
```

Die Subroutine liest den Inhalt der Page P von File F in die Arrayzellen 256...511. In der Variable R steht das Resultat: entweder die Filenummer F oder -1, falls das Lesen der Page nicht möglich ist.

## 15.6 FDELETE f [FDEL]

Löscht das File F, wenn es vorhanden und vom Typ USR ist. Da FDELETE im Fehlerfall abbricht, sollte vorher mit FTYPE und FSIZE sichergestellt werden, dass das File vorhanden ist.

```
56 'delete file F
57 R=-1
58 IF FTYPE(F)<>20 THEN RETURN
59 FDELETE F: R=F: RETURN
```

Die Subroutine löscht das File F, falls es vorhanden ist.

## 15.7 FSELECT v,a [FSEL]

Stellt die Fileselect-Box dar, falls ein formatierter Dataflash-Baustein angeschlossen ist. Die Nummer des gewählten Files liegt im Anschluss in der Variable v. Ist der Dataflash nicht ansprechbar oder wird die Auswahl mit der ESC-Taste abgebrochen, wird die Variable v auf -1 gesetzt. Der wert a ist die Array-Position, ab der die nullterminierte Boxüberschrift steht. Der Rahmen der Fileselct-Box wird mit der aktuell eingestellten Vordergrundfarbe gezeichnet, der Hintergrund ist immer schwarz. Der ursprüngliche Bildinhalt wird nach Verlassen der Fileselect-Box wiederhergestellt.

```
60 'draw fileselct box
61 DATA 0,"Select file:",0
62 FSELECT F,0: RETURN
```

Die Subroutine zeigt eine Filelesect-Box mit dem Titel „Select file:”.

## 15.8 FFIND(a)

Da das Dateisystem nicht mit Verzeichnissen arbeitet, braucht man die Dateinummer um in eine bestimmte Datei zu schreiben oder von ihr zu lesen. Dazu dient die Funktion **FFIND**. Der Parameter a ist wieder ein Zeiger auf den Bytebereich des Arrays. Das erste Byte gibt den gesuchten Dateityp an (siehe Tabelle oben), danach folgt die Anzahl der zu vergleichenden Bytes am Anfang der Datei und das Vergleichsmuster. Rückgabewert ist die erste gefundene Dateinummer oder -1, falls keine entsprechende Datei gefunden wurde.

```
10 DATA 0,252,0:F=FFIND(0)
```

Es wird nach der ersten freien Datei gesucht (Kennung 252) und liefert die Dateinummer bzw. -1 falls keine freie Datei mehr vorhanden ist.

# 16 Direktzugriff auf das Dataflash-Modul

In manchen Fällen kann es sinnvoll sein, anstelle der Dateisystemfunktionen Daten direkt auf das Dataflash-Modul zu schreiben oder von dort zu lesen. Dazu dienen die folgenden beiden Befehle:

## 16.1 RREAD p,a [RRD]

Liest 256 Bytes aus Page P und schreibt sie in das Array. P ist hierbei die Page-Nummer im Dataflash und bezieht sich nicht auf eine Datei. Der zweite Parameter bestimmt wieder die Arrayposition:

| **a** | **Arraybereich** |
|---|---|
| 0 | 0...255 |
| 1 | 256...511 |
| 2 | 512...767 |

Da RREAD im Fehlerfall abbricht, sollte sichergestellt sein, dass die Page auch gelesen werden kann.

## 16.2 RWRITE p,a [RWR]

Schreibt 256 Bytes aus dem Array in die Page P. P ist hierbei die Page-Nummer im Dataflash und bezieht sich nicht auf eine Datei.
**Achtung!** Diese Funktion kann das Dateisystem zerstören und sollte deshalb nur mit Vorsicht oder auf unformatierte Dataflash-Bausteine angewendet werden. Der zweite Parameter bestimmt die Arrayposition:

| **a** | **Arraybereich** |
|---|---|
| 0 | 0...255 |
| 1 | 256...511 |
| 2 | 512...767 |

Da RWRITE im Fehlerfall abbricht, sollte sichergestellt sein, dass die Page auch geschrieben werden kann.

# 17 Laden und Speichern von Programmen

Da die BASIC Programme mit 95 Zeilen a 32 Zeichen nicht gerade sehr üppig ausgestattet sind, gibt es die Möglichkeit, Programme während der Laufzeit auf den Dataflash zu schreiben oder von dort zu laden.

## 17.1 LOADP f,p

Lädt das Programm mit der Dateinummer f an den Programmplatz p. Dabei ist es nicht möglich, das gerade laufende Programm zu überschreiben. Wird dies versucht, bricht das Programm mit einer Fehlermeldung ab.

## 17.2 SAVEP f,p

Schreibt das Programm von Programmplatz n in die Datei mit der Nummer f. Dazu muß die Datei entweder nicht vorhanden oder eine Programmdatei sein. **VORSICHT**, eine eventuell vorhandene Datei wird ohne Nachfrage überschrieben!

```
10 DATA 0,252,0:C=FFIND(0)
11 IF C<0 THEN E=1:RETURN
12 DATA 0,16,4,"Test"
13 D=FFIND(0)
14 IF D<0 THEN E=2:RETURN
15 SAVEP C,8:LOADP D,8
16 GOSUB 8,1:LOADP C,8
17 FDELETE C:E=0:RETURN
```

Dieses Unterprogramm sucht nach einem freien Dateiplatz und nach einer Programmdatei, die mit ”Hallo” beginnt. Ist beides gefunden, wird das aktuelle Programm 8 auf dem freien Speicherplatz gesichert, das Hallo-Programm in Programmplatz 8 geladen und dann mittelx **GOSUB** gestartet. Wenn das Programm dann mittels **RETURN** zurückkehrt, wird das ursprünglich an Programmplatz 8 befindliche Programm wiederhergestellt und die temporär erzeugte Datei gelöscht. Das aufgerufene Programm sollte dabei die Variable C nicht ändern.

# 18 Meldungen und Fragen in der Box

## 18.1 ALERT n

Zeigt eine Alert-Box an. Die Daten dazu befinden sich im Array ab Position n. Als Erstes steht dort ein Byte (Vordergrundfarbe + 16 * Hintergrundfarbe), gefolgt von nullterminiertem Text. Fehlt der Abschluss mit 0x00, wird die Ausgabe nach 20 Zeichen abgebrochen. Der aktuelle Bildschirminhalt wird vorher gesichert und nach Betätigen der **ENTER** Taste wieder restauriert.

```
01 DATA 0,97,"Hallo Welt",0
02 ALERT 0
```

Zeichnet eine Alertbox mit blauer Schrift **Hallo Welt** auf gelbem Grund. Wird die **Enter** Taste gedrückt, verschwindet die Box wieder.

## 18.2 ASK V,n

Zeigt eine Frage-Box an. Die Daten dazu befinden sich im Array ab Position n. Zuerst steht dort ein Byte (Vordergrundfarbe + 16 * Hintergrundfarbe), gefolgt von nullterminiertem Text. Fehlt der Wert 0x00, wird die Ausgabe nach 20 Zeichen abgebrochen.

```
01 DATA 0,97,"Weitermachen?",0
02 ASK P,0
03 IF P=0 WAIT 20: GOTO 2
04 ? "Ende"
```

Wenn **"Y"** oder **Enter** gedrückt wurde, bekommt P den Wert 1 zugewiesen, wurde "N" oder **ESC** gedrückt, erhält P den Wert 0. Groß- und Kleinschreibung spielt dabei keine Rolle. Der aktuelle Bildschirminhalt wird vorher gesichert und nach Betätigen einer gültigen Taste wieder restauriert.

# 19 Menüfunktionen

## 19.1 MENU(n)

Die Funktion **MENU(n)** zeigt ein User-definiertes Menu an. Dabei zeigt n auf eine Zeile im aktuellen Programm, ab der in 1 oder 2 Kommentarzeilen Arraytext hinterlegt sind. Das letzte Zeichen in der ersten Zeile enthält zwei Steuerbits. Diese lsssen sich z.B. mittels der Ziffern "0"…"3" einstellen:

| Zeichen | Menü-Eigenschaften |
|---|---|
| 0 | Eine Menü-Ebene, Funktion kehrt nur beim Drücken einer der in der nächsten Tabelle gelisteten Tasten zurück. |
| 1 | Zwei Menü-Ebenen, Funktion kehrt nur beim Drücken einer der in der nächsten Tabelle gelisteten Tasten zurück. |
| 2 | Eine Menü-Ebene, Funktion kehrt nur beim Drücken einer beliebigen Taste zurück (ausser Shift, CTrl und Alt). |
| 2 | Zwei Menü-Ebenen, Funktion kehrt nur beim Drücken einer beliebigen Taste zurück (ausser Shift, CTrl und Alt). |

Die folgende Tabelle listet die Rückgabewerte auf, die beim Betätigen der Tasten **ESC** und **F1…F3** auf. Für alle anderen tasten wird der "normale" Tastencode zurückgeliefert.

| Offset | Taste | Rückgabewert |
|---|---|---|
| 0…5 | ESC | 1 |
| 6…11 | F1 | 2 |
| 12…17 | F2 | 3 |
| 18…23 | F3 | 4 |
| 24…29 | F4 | 5 |
| 30…35 | linke CTRL + ESC | 6 |
| 36…41 | linke CTRL + F1 | 7 |
| 42…47 | linke CTRL + F2 | 8 |
| 48…53 | linke CTRL + F3 | 9 |
| 54…59 | linke CTRL + F4 | 10 |

Das folgende Programm zeigt ein Beispielmenü an. Die Daten dazu befinden sich in den Zeillen 1 und 2

```
01 '1111112222223333334444445555551
02 '6666667777778888888999999AAAAAA
03 A=MENU(1)
04 ? @0,0;"Menu:",A," "
05 B=KEY(2):IF A>1 GOTO 3
```

Mit der linken CTRL-Taste kann zwischen beiden Menüebenen hin- und hergeschaltet werden. Dabei wird gewartet, bis eine gültige Taste (ESC, F1...F4) gedrückt wurde und in der obersten Zeile wird der letzte Rückgabewert angezeigt.**ESC** bricht das Programm ab.

# 20 Fehlermeldungen und Errorhandling

## 20.1 ONERR N

Liegt N im Bereich der gültigen Zeilen (1...95), wird bei einem Fehler in diese Zeile gesprungen. Andernfalls wird dann (wie gewohnt) mit einer Fehlermeldung abgebrochen. Befinden sich an und hinter der angegebenen Zeile keine Befehle mehr, wird das Programm ohne Fehlermeldung verlassen.
Wenn mittels GOSUB eine Subroutine in einem anderen Programm aufgerufen wird, wird im Fehlerfall dort in die angegebene Zeile gesprungen.

```
01 ONERR 10
02 A=SQR(-1)

10 DATA 512,7,1,"Error!",0
11 ALERT 512
12 END
```

Da versucht wird, die Wurzel aus einer negativen Zahl zu ziehen, springt das Programm in Zeile 10 und zeigt dort eine Alertbox an.

## 20.2 ERR(n)

Über diese Funktion können Informationen über den aufgetretenen Fehler ausgelesen werden. Das funktioniert natürlich nur, wenn vorher mit ONERR eine eigene Fehlerroutine angesprungen wurde. Für n=1 wird die Zeile zurückgegeben, in der der Fehler aufgetreten ist, für n=2 das Statement. Für alle anderen Werte von n wird die Fehlernummer zurückgegeben.

```
01 ONERR 10
02 A=SQR(-1)

10 DATA 512,7,1,"Error          ",0
11 ? #3@2,8;ERR(0);
12 ALERT 512
13 END
```

Da versucht wird, die Wurzel aus einer negativen Zahl zu ziehen, gibt es wieder eine Alertbox. In Zeile 11 wird die Fehlernummer in das Array geschrieben und nun mit angezeigt.

## 20.3 Fehlermeldungen

Insgesamt gibt es 37 verschiedene Fehlermeldungen. Im Editor werden diese oben in der zweiten Zeile mit Programm-, Zeilen- und Statementnummer angezeigt. Im Allgemeinen lässt sich damit der Fehler recht schnell finden, manchmal kann er sich auch am Ende der vorherigen Zeile befinden.

| | | |
|---|---|---|
| 01 | BREAK | Das Programm wurde mit der Tastenkombination Control (Strg) + C abgebrochen. |
| 02 | OVERFLOW | Bei einer Rechenoperation liegt das Ergebnis ausserhalb der Grenzen von -32767…32767 |
| 03 | DIVIDE/0 | Es wurde versucht, durch Null zu dividieren |
| 04 | SQR FROM <0 | Es wurde versucht, die Quadratwurzel aus einer negativen Zahl zu ziehen |
| 05 | CONSTANT TOO BIG | Die angegebene Zahl liegt das Ergebnis ausserhalb der Grenzen von -32767…32767 |
| 06 | WRONG EXPRESSION | In der Formel befinden sich syntaktische Fehler |
| 07 | SYNTAX ERROR | Fehlende Parameter, ungültige Zeichen |
| 08 | UNKNOWN KEYWORD | Unbekanntes Schlüsselwort, der Programmcode enthält ungültige Bytes |
| 09 | WRONG FORMAT | Die Zahlenformatierung ist ungültig (mehr Nachkommastellen als sichtbare) |
| 10 | BAD LINENUMBER | Die Zeile mit der angegebenen Zeilennummer existiert nicht |
| 11 | NEXT W/O FOR | Zu diesem NEXT Statement gibt es kein korrespondierendes FOR |
| 12 | RETURN W/O GOSUB | Zu diesem RETURN Statement gibt es kein korrespondierendes GOSUB |
| 13 | STACK OVERFLOW | Insgesamt dürfen nur 16 FOR/GOSUB/REPEAT zu gleichen Zeit „geöffnet” sein, Ursache kann z.B. ein rekursiver Unterprogrammaufruf sein. |
| 14 | UNTIL W/O REPEAT | Zu diesem UNTIL Statement gibt es kein korrespondierendes REPEAT |
| 15 | I2C ERROR | An der angegebenen Adresse meldet sich kein Gerät oder es meldet keine Bereitschaft, serielle EEPROMs können z.B. mit dem Schreiben von Daten beschäftigt sein. |
| 16 | UNKNOWN ERROR | unbekannter Fehler. Leider lassen sich alle möglichen Kombinationen von Befehlen, Funktionen und Parametern nicht so einfach vollständig testen und es wäre schön, wenn Sie den Fehler melden würden. |
| 17 | DFLASH ERROR | Kein Dataflash angeschlossen oder der Baustein reagiert nicht. |
| 18 | OUT OF ARRAY | Es wurde versucht auf ein Arrayelement zuzugreifen, welches nicht existiert. Der Fehler tritt auch auf, wenn bei BCOPY der Platz im Array nicht ausreicht. |
| 19 | INCOMPLETE PAR | Es wurden zuwenige Parameter für den Befehl angegeben |
| 20 | KEYWORD IS MISSING | Das angegebene Statement beginnt nicht mit einem Schlüsselwort sondern z.B. mit einer Zahl |
| 21 | WRONG BCOPY | Der Kopiermodus liegt nicht im Bereich (1…6) |
| 22 | OUT OF SCREEN | Versuch, ausserhalb des Bildschirmbereiches zu zeichnen (nicht implementiert) |
| 23 | CANNOT CREATE FILE | Das File existiert bereits oder die Filenummer liegt ausserhalb des gültigen Bereiches |
| 24 | DFLASH FULL | Das File kann nicht erstellt werden, da der Platz auf dem Dataflash nicht ausreicht. |
| 25 | FILE NOT FOUND | Das File existiert nicht (ist frei) |
| 26 | PAGE NOT IN FILE | Die zu lesende/schreibende Page existiert nicht in diesem File |
| 27 | PAGES NOT IN RANGE | Es können nur Files mit 1…128 Pages erzeugt werden |
| 28 | NOT IN GRAPHICS MODE | Der angegebene Befehl kann nicht im Grafikmode (VMODE 1…3/5) ausgeführt werden |
| 29 | NOT IN TEXT MODE | Der angegebene Befehl kann nicht im Textmode (VMODE 0) ausgeführt werden |
| 30 | NO USR FILE | Es können nur Files vom Typ USR gelesen/geschrieben werden (obsolete) |
| 31 | SRC OUT OF SCREEN | Der Quellblock bei BCOPY befindet sich nicht vollständig im Bildbereich |
| 32 | DEST OUT OF SCREEN | Der Zielblock bei BCOPY befindet sich nicht vollständig im Bildbereich |
| 33 | WRONG SPRITE | Die Sprite-Definition hat einen Fehler (z.B. größer als 8x8 Zeichen) |
| 34 | WRONG EEPROM ADR | Es wurde versucht, auf eine interne EEPROM-Zelle >1999 zuzugreifen |
| 35 | NO PRG FILE | Es wurde versucht, mit LOADP ein Nicht-Programm-Datei zu laden |
| 36 | XMODEM ERROR | Beim X-Modem Empfang ist ein (Prüfsummen-) Fehler aufgetreten |
| 37 | CANT LOAD THIS | Es wurde versucht, das gerade laufende Programm mit LOADP zu überschreiben |
| 38 | IO DISABLED | Ausgabe auf den Parallelport wurde durch einen Treiber auf Programmplatz 8 deaktiviert |
| 39 | SERIAL1 NOT AVAILABLE | es wurde versucht, auf die zweite serielle Schnittstelle zuzugreifen, Controllertyp ist aber ein Mega644 oder Input-Pin der seriellen System-Schnittstelle liegt an PD3 |
| 40 | NO IO DRIVER | es wurde versucht, mittels IN/OUT auf eine nicht existierende Adresse ab 0x0800 zuzugreifen |