AVR-ChipBasic2: BASIC-Referenz

(c) 2006-2008 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).

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 128-384 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-Verknpfung **&**
- Oder-Verknpfung **#**
- Vergleiche (=,<>,<,<=,>,>=) liefern 0 für falsch oder 1 für wahr

Dazu kommen noch öffnende und schließende Klammern sowie Funktionen. Funktionen können im Gegensatz zu Schlüsselwörtern nicht abgekürzt werden.

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,
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
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
PSTAT(	Sequenzer-Status, siehe Abschnitt Audio

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

- Die Zellen 0...767 sprechen den Bereich byteweise an
- Die Zellen 1024...1407 sprechen den Bereich im wortweise an

Bei den Byte-Zugriffen befindet sich das LOW-Byte an der geraden Adresse und das High-Byte an der darauffolgenden ungeraden Adresse. Wird auf eine Array-Zelle zugegriffen die nicht existiert, wird mit einer Fehlermeldung abgebrochen.

4 Schlüsselwörter

4.1 Wertzuweisung

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

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

4.1.1 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.

4.1.2 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.

4.1.3 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.

4.2 Programmsteuerung

4.2.1 FAST

Mit dem FAST-Befehl wird die Bildschirmdarstellung abgeschaltet. Synchronsignale werden weiterhin generiert.

4.2.2 SLOW

Mit dem SLOW-Befehl wird die Bildschirmdarstellung wieder eingeschaltet.

4.2.3 BREAK

Setzt einen Breakpoint, ruft den Monitor auf.

4.2.4 END

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

4.2.5 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.

4.2.6 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.

```
10 IF A>5 THEN A=5
20 IF A>5 A=5
```

Beide Ausdrücke bewirken exakt das Gleiche.

4.2.7 FOR - NEXT

Bei der Schleifenabarbeitung gibt es nur die Grundform **FOR A=1 TO C** ohne die Angabe der Schrittweite, die konstant 1 ist. Da der Stack auf 16 Einträge begrenzt ist, lassen sich nur 16 Schleifen bzw. Unterprogrammaufrufe schachteln.

```
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.

4.2.8 GOSUB - RETURN [GOS - RET]

GOSUB Expr ruft das Unterprogramm in der durch den Ausdruck definierten Zeile auf, mit **RETURN** wird wieder zurückgesprungen. Da der Stack auf 16 Einträge begrenzt ist, lassen sich nur insgesamt 16 Schleifen bzw. Unterprogrammaufrufe schachteln.

4.2.9 GOSUB mit zwei Parametern

Eine weitere, recht ungewöhnliche Anweisung ist **GOSUB 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. Der Rücksprung erfolgt wie gehabt mit **RETURN**.

4.3 Bildschirm-Ausgabe

4.3.1 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 7, dabei bedeutet:

Value	0	1	2	3	4	5	6	7
Farbe	schwarz	blau	rot	magenta	grün	cyan	gelb	weiss

```
10 COLOR 4,2
```

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

4.3.2 CLS

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

4.3.3 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.

4.3.4 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
Expr	gibt das Ergebnis des Ausdrucks mit dem eingestellten Format aus
;	Trenner zwischen Ausdürcken
,	Trenner zwischen Ausdürcken, 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 Schnittstelle
#2	Ausgabe auf die parallele Schnittstelle
#3	Ausgabe in das Array
#4...	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 auf GLCD’s feiner positionieren zu können.

4.3.5 EMIT

Gibt durch Komma getrennte Zeichen auf den Bildschirm/Seriell/Drucker aus, je nachdem was zuletzt eingestellt war. Es werden keine Zeichen, sondern Zahlenwerte der Zeichen erwartet. Beispiel:

```
10 EMIT 64,$40
```

gibt zwei Klammeraffen aus.

4.3.6 YEMIT

Gibt durch Komma getrennte Zeichen auf den Bildschirm/Seriell/Drucker aus, je nachdem was zuletzt eingestellt war. Es werden keine Zeichen, sondern Zahlenwerte der Zeichen erwartet. Beispiel:

```
11 YEMIT 64,$40
```

gibt wieder zwei Klammeraffen aus. Bei der Bildschirmausgabe stehen diese jetzt jedoch untereinander. Wenn die Ausgabe auf Seriell/Drucker erfolgt, entspricht die Ausgabe der von **EMIT**.

4.3.7 CBOX Y1,X1,Y2,X2

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

```
10 CBOX 3,3,5,5
```

löscht oben links ein Quadrat von 3x3 Zeichen.

4.3.8 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.

4.3.9 LCHAR n

Es werden (am oberen Bildrand beginnend) n Zeichenzeilen um ein Zeichen nach links verschoben. Am rechten Rand rücken Leerzeichen nach.

```
10 LCHAR 10
```

verschiebt die obersten 10 Zeichenzeilen um 1 Zeichen nach links.

4.3.10 GETCHAR variable,Y,X [GCH]

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

```
11 GCHAR F,0,0
```

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

4.3.11 GETATTR variable,Y,X [GAT]

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

```
12 GA F,0,0
```

Schreibt den Atributwert 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	0
4	Vordergrundfarbe Bit 0
5	Vordergrundfarbe Bit 1
6	Vordergrundfarbe Bit 2
7	Zeichensatz 0/1

4.4 Tastatur

4.4.1 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, Ausdrcke einzugeben die dannüberechnet werden. Das folgende Beispiel zeigt einen kleinen Rechner, das letzte Ergebnis ist in der Variable M gespeichert.

1. INPUT ”AUFGABE: ”,M
2. PRINT ”ERGEBNIS:”;M
3. GOTO 1

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

4.4.2 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 0,1024
03 X=1024
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.

4.4.3 RKEY V [RK]

Die aktuell gedrückte Taste wird in die Variable V geschrieben. Ist keine Taste gedrückt, wird eine 0 geschrieben.

```
10 RKEY P
```

der aktuelle Tastaturcode wird in die Variable P geschrieben.

4.4.4 WKEY V [WK]

Es wird auf einen Tastendruck gewartet und die gedrückte Taste wird in die Variable V geschrieben.

```
10 WKEY P
```

Es wird auf einen Tastendruck gewartet und der aktuelle Tastaturcode in die Variable P geschrieben.

4.4.5 Die Funktion KEY

Diese Funktion liefert verschiedene Tastaturabfragen als -1,0,1 Wert. Als Parameter wird die Art der Abfrage eingetragen. Ist keine der beiden Tasten betätigt, wird 0 als Funktionswert zurückgeliefert.

KEY(0)	linke Shift-Taste liefert 1, linke Control-Taste liefert -1, beide 0
KEY(1)	rechte Shift-Taste liefert 1, rechte Control-Taste liefert -1, beide 0
KEY(2)	Taste Cursor links liefert -1, Taste Cursor rechts liefert 1
KEY(3)	Taste Cursor nach unten liefert -1 Taste Cursor hoch liefert 1

4.5 Pseudografik

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.

4.5.1 PLOT Y,X(,C) [PL]

Mit dem PLOT-Befehl wird ein „Pixel" gesetzt. Dabei kann es passieren, dass benachbarte Punkte auch ihre Farbe wechseln. Zusätzlich zu den Koordinaten kann auch die Zeichenfarbe temporär gesetzt werden. Temporär deshalb, weil die Farbe nur für den aktuellen Zeichenvorgang verwendet wird.

```
10 PLOT 3,7+A
```

Zeichnet ein „Pixel" an der Position Y=3 und X=7+A mit der aktuellen Vordergrundfarbe.

```
10 PLOT 3,7+A,6
```

Zeichnet ein gelbes „Pixel" an der Position Y=3 und X=7+A.

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

Zeichnet eine Linie von Y1,X1 nach Y2,X2. Zusätzlich zu den Koordinaten kann auch die Zeichenfarbe temporär gesetzt werden.

```
10 DRAW 0,0,45,59
```

Hier wird eine Linie von der linken oberen in die rechte untere Ecke gezeichnet.

```
10 DRAW 0,0,45,59,4
```

Zeichnet eine grüne Linie von der linken oberen in die rechte untere Ecke.

4.5.3 DRAWTO Y2,X2(,C) [DTO]

Zeichnet eine Linie vom letzten Endpunkt nach Y2,X2. Zu Beginn sind die „letzten Koordinatenäuf 0,0 gesetzt, danach entsprechen sie immer den zuletzt angegebenen. Zusätzlich zu den Koordinaten kann auch die Zeichenfarbe temporär gesetzt werden.

```
10 PLOT 4,4
11 DRAWTO 10,8
```

Zeichnet eine Linie von Y=4,X=4 nach Y=10,X=8 in der aktuell eingestellten Farbe.

4.5.4 BOX Y1,X1,Y2,X2(,C)

Mit dem BOX-Befehl wird ein Rechteck gezeichnet. Zusätzlich zu den Koordinaten kann auch die Zeichenfarbe temporär gesetzt werden.

```
10 BOX 1,1,10,10
```

4.5.5 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. Zusätzlich zu den Koordinaten kann auch die Zeichenfarbe temporär gesetzt werden.

```
10 COLOR 2,0:FBOX 0,0,0,100,4
```

zeichnet eine waagerechte grüne Linie am oberen Bildschirmrand, der Wert 100 wird zur Laufzeit auf den Bildschirmbereich begrenzt. Auch wenn die Ausgabe auf den sichtbaren bereich begrenzt ist, verbrauchen auch nicht gesetzte Pixel Rechenzeit.

4.5.6 CIRCLE Y1,X1,RY,RX(,C) [CI]

Mit dem CIRCLE-Befehl können Kreise und Ellipsen gezeichnet werden. Zusätzlich zu den Koordinaten kann auch die Zeichenfarbe temporär gesetzt werden.

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

Zeichnet einen Kreis mit dem Mittelpunkt 20,20 und dem Radius 10

```
10 CIRCLE 20,20,10,6,4
```

Zeichnet eine grüne Ellipse mit dem Mittelpunkt 20,20 und den Radien 10 in Y-Richtung sowie 6 in X-Richtung.

4.5.7 FCIRCLE Y1,X1,RY,RX(,C) [FCI]

Mit dem CIRCLE-Befehl können ausgefüllte Kreise und Ellipsen gezeichnet werden. Zusätzlich zu den Koordinaten kann auch die Zeichenfarbe temporär gesetzt werden.

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

Zeichnet einen ausgefüllten Kreis mit dem Mittelpunkt 20,20 und dem Radius 10

```
10 CIRCLE 20,20,10,6,2
```

Zeichnet eine ausgefüllte rote Ellipse mit dem Mittelpunkt 20,20 und den Radien 10 in Y-Richtung sowie 6 in X-Richtung.

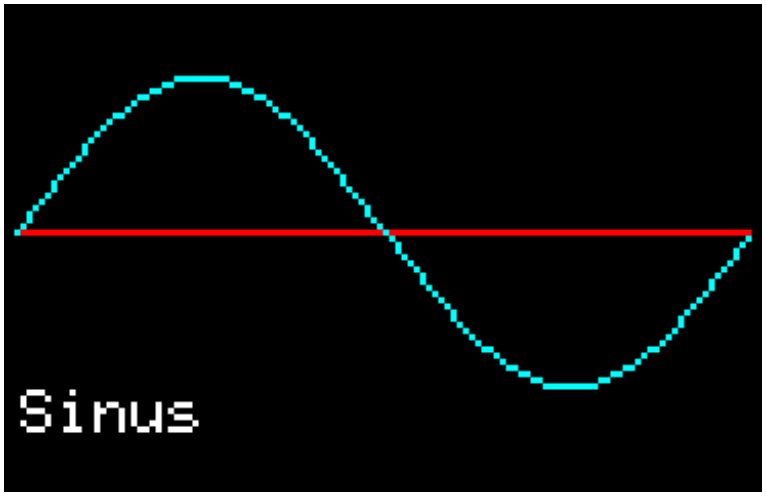
4.5.8 LPIX n

Es werden (am oberen Bildrand beginnend) n Zeichenzeilen um ein Pixel nach links verschoben. Am rechten Rand rücken nicht gesetzte Pixel nach. Es wird nur die Pixelinformation verschoben, nicht aber die Vordergrund- und Hintergrundfarbe.

```
05 LPIX 10
```

verschiebt die obersten 20 Zeichenzeilen um 1 Pixel nach links.

4.6 Vollgrafik



Zusätzlich zur Pseudografik im Textmodus gibt es noch 3 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 Vorder- und Hintergrundfarbe gesetzt, zeigen diese auf den Palettenindex und geben nicht die Farbe direkt an.

4.6.1 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	8 Vordergrund und 8 Hintergrund
1	168	116	2 aus 8
2	120	76	4 aus 8
3	84	58	8 aus 8

Zusätzlich werden die im jeweiligen Modus genutzten Paletteneinträge vorbelegt.

4.7 PALETTE start,anzahl,c1(,c2,c3,c4) [PAL]

Mit diesem befehl wird die Farbpalette für die Videomodi 1 bis 3 eingestellt. Für Modus 1 werden nur die Einträge 0 und 1 verwendet, für Modus 2 die Einträge 0 bis 3. Die Anzahl der zu setzenden Paletteneinträge muss gleich der angegebenen Anzahl sein, ist jedoch auf 4 begrenzt, anderenfalls kommt es zu einer Fehlermeldung. Als Farbwerte müssen im Bereich (0...7) angegeben werden. Nach dem Umschalten des Videomodus sind die Paletteneinträge wie folgt vorbelegt:

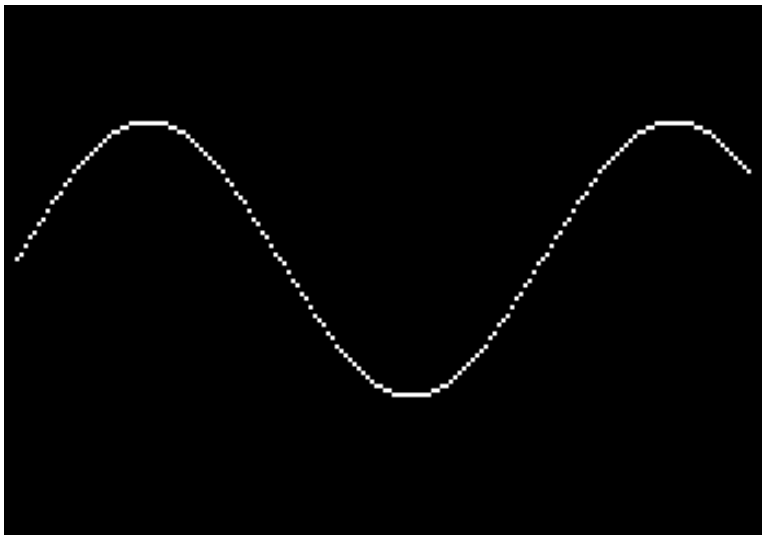
Index	Modus 1	Modus 2	Modus 3
0	schwarz (0)	schwarz (0)	schwarz (0)
1	weiss (7)	rot (2)	blau (1)
2		cyan (5)	rot (2)
3		weiss (7)	magenta (3)
4			grün (4)
5			cyan (5)
6			gelb (6)
7			weiss (7)

4.7.1 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(30*X)/6
04 PLOT 55-V,X
05 NEXT
#
```

Zeichnet eine Sinuskurve (Videomode 1).

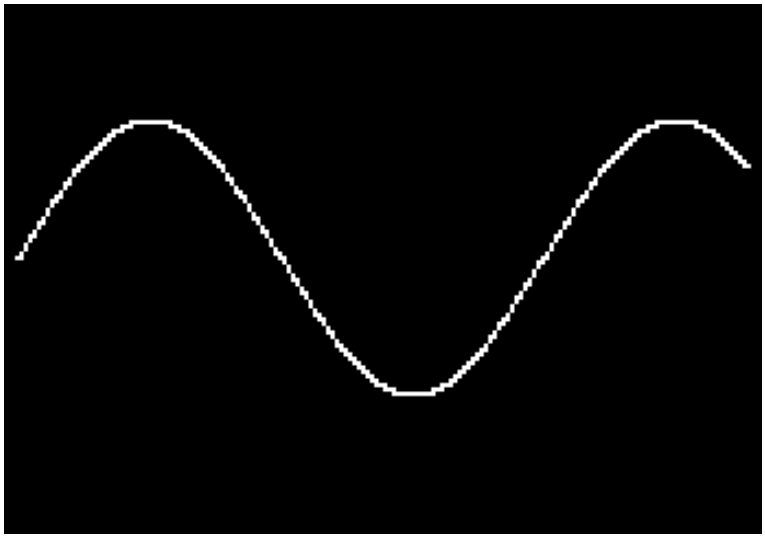


Wie man sieht, ist der Kurvenzug teilweise unterbrochen, da halt nur einzelne Punkte gesetzt werden. Um solche Unterbrechungen zu vermeiden gibt es den folgenden Befehl.

4.7.2 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.

```
01 CLS:VMODE 1,PLOT 55,0
02 FOR X=1 TO 167
03 V=SIN(30*X)/6
04 DRAWTO 55-V,X
05 NEXT
#
```



Diesmal gibt es keine Unterbrechungen. 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.

4.7.3 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(30*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.

4.7.4 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
```

4.7.5 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.

4.7.6 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.

4.7.7 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.

4.7.8 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.

4.7.9 BCOPY [BC]

Der BCOPY Befehl ermöglicht es, rechteckige Blöcke innerhalb des Bildschirms oder ins/vom Array zu kopieren. Damit lassen sich z.B. Sprites im Grafikmodus oder Scrolling realisieren. Blöcke können von und an (fast) beliebigen Koordinaten kopiert werden, lediglich die Größe ist Beschränkungen unterworfen. Es können nur ganze Teilblöcke kopiet 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,Y2,X2,DY,DY
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.

4.7.10 PRINT [?]

Der PRINT Befehl entspricht weitestgehnd 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 (richtig).

4.8 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.

4.8.1 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.

4.8.2 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
```

Bleibt das Stecnchen stehen und das Hinedrnis ist nicht mehr zu sehen.

4.9 Audio

4.9.1 NOTE n [NO]

Ein Ton mit der Tonhöhe n (Halbtonschritte ab 220 Hz aufwärts) wird ausgegeben. Bei n=0 bis 63 werden Noten in einer „dunklen" Klangfarbe ausgegeben, bei n=64 bis 127 Noten mit einer „hellen" Klangfarbe. Bei n=255 wird keine Note sondern Rauschen ausgegeben.

```
01 FOR N=0 to 63:NOTE N
02 WAIT 5:NEXT
```

gibt nacheinander alle spielbaren Noten aus.

4.9.2 PLAY start,stop,speed

Ab Version 0.69 hat AVR-Chipbasic2 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 und die mit der letzten zu spielenden Note sowie die die Anzahl der Halbbilder (1/50s bei PAL, 1/60s bei NTSC) zwischen zwei Noten benötigt. Die Noten zwischen Start- und End Array-Zelle werden zyklisch abgespielt, ein Aufruf mit weniger als 3 Parametern schaltet den Sequenzer (sofort) ab.

Wert	Bedeutung
0...63	Noten in einer „dunklen" Klangfarbe
64...127	Noten in einer „hellen" Klangfarbe
128...253	nicht genutzt entspricht den Werten 0...125
254	Stille, es wird keine Note gespielt
255	Rauschen

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

```
01 DA 0,39,44,49,44
02 PLAY 0,3,20
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.

4.9.3 PSTAT(mode)

Diese Funktion liefert Aufschluss darüber, was der Sequenzer gerade „macht". Ist der Parameter mode=1, dann wird die laufende Sequenz noch bis zum Ende gespielt und dann gestoppt. Ist der Parameter mode=2, dann wird die laufende Sequenz sofort gestoppt. Alle anderen Parameterwerte liefern nur den aktuellen Status, der wie folgt codiert ist:

Wert	Bedeutung
0	Sequenzer läuft, Noten werden gespielt
1	Sequenzer läuft noch bis zum Ende der Sequenz und wird dann stoppen
2	Sequenzer ist gestoppt

4.10 Zeit

4.10.1 NOTE n [NO]

Ein Ton mit der Tonhöhe n (Halbtonschritte ab 220 Hz aufwärts) wird ausgegeben. Bei n=0 bis 63 werden Noten in einer „dunklen" Klangfarbe ausgegeben, bei n=64 bis 127 Noten mit einer „hellen" Klangfarbe. Bei n=255 wird keine Note sondern Rauschen ausgegeben.

```
01 FOR N=0 to 63:NOTE N
02 WAIT 5:NEXT
```

gibt nacheinander alle spielbaren Noten aus.

4.10.2 PLAY start,stop,speed

Ab Version 0.69 hat AVR-Chipbasic2 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 und die mit der letzten zu spielenden Note sowie die die Anzahl der Halbbilder (1/50s bei PAL, 1/60s bei NTSC) zwischen zwei Noten benötigt. Die Noten zwischen Start- und End Array-Zelle werden zyklisch abgespielt, ein Aufruf mit weniger als 3 Parametern schaltet den Sequenzer (sofort) ab.

Wert	Bedeutung
0…63	Noten in einer „dunklen" Klangfarbe
64…127	Noten in einer „hellen" Klangfarbe
128…253	nicht genutzt entspricht den Werten 0…125
254	Stille, es wird keine Note gespielt
255	Rauschen

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

```
01 DA 0,39,44,49,44
02 PLAY 0,3,20
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.

4.10.3 PSTAT(mode)

Diese Funktion liefert Aufschluss darüber, was der Sequenzer gerade „macht". Ist der Parameter mode=1, dann wird die laufende Sequenz noch bis zum Ende gespielt und dann gestoppt. Ist der Parameter mode=2, dann wird die laufende Sequenz sofort gestoppt. Alle anderen Parameterwerte liefern nur den aktuellen Status, der wie folgt codiert ist:

Wert	Bedeutung
0	Sequenzer läuft, Noten werden gespielt
1	Sequenzer läuft noch bis zum Ende der Sequenz und wird dann stoppen
2	Sequenzer ist gestoppt

4.11 Ein- und Ausgabe

4.11.1 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.

4.11.2 OUT n,b

Hier gibt es zwei 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.

4.12 Kommunikation

4.12.1 SPUT n

das Byten wird an die serielle Schnittstelle ausgegeben.

```
01 SPUT 10
```

gibt einen Zeilenvorschub an die serielle Schnittstelle aus.

4.12.2 SGET V

Ein Zeichen von der seriellen Schnittstelle wird eingelesen und in die Varaiable V gespeichert.

```
01 SGET C
```

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

4.12.3 PUMP n

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

```
01 PU 1
```

schaltet die Ladungspumpe ein. Ist der AutoRun-Jumper gesetzt, ist die Ladungspumpe per default nach dem Start ausgeschaltet.

4.12.4 ICOMM adr,start,num

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 zuübertragenden Bytes an.

```
01 BDATA HI(B),LO(B),LO(A),HI(A):ICOMM \$a0,0,4
```

Schreibt den Wert der Variable A an die Adresse B des Daten-EEPROMS. 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 nach 0x00 noch ein Dummy-Byte an den Master (BASIC-Controller) senden, ist mindestens 1 Byte zu senden braucht kein Dummy-Byte zu folgen.

4.13 Speicher

4.13.1 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.

4.13.2 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 und Adresse 1) angeschlossen werden. Bei anderen Typen lässt sich die gleiche Funktionalität über ICOMM realisieren.

4.13.3 XPOKE adr,dat [XP]

Speichert ein Byte im externen EEPROM (Adresse=1).

```
01 XPOKE 100,C
```

speichert den Wert der Variablen C an die Adresse 100.

4.13.4 XPEEK(adr)

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

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

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

4.14 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 reservíeren, auch wenn dadurch Speicherplatz verschenkt wird. Ansonsten wird einfach mit der Dateinummer gearbeitet.

4.14.1 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
-1	Dateinummer existiert nicht

4.14.2 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)

4.14.3 FCREATE n,p

Erzeugt das USR-File n 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 erzeugt.

```
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.

4.14.4 FWRITE n,p,a

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. Es kann nur in Files vom Typ USR geschrieben werden.

```
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.

4.14.5 FREAD n,p,a

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 geschrieben werden kann. Es kann nur von Files vom Typ USR gelesen werden.

```
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.

4.14.6 FDELETE n

Löscht das File N, 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 und von TYP USR 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 und vom Typ USR ist.

4.14.7 FSELECT v,a

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:”.

4.15 Meldungen und Fragen in der Box

4.15.1 ALERT n

Zeigt eine Alert-Box an. Die Daten dazu befinden sich im Array ab Position n. Zuerst stehen dort Vordergrund- und Hintergrundfarbe, gefolgt von Nullterminiertem Text. Fehlt dder wert 0x00, wird die Ausgabe nach 20 Zeichen abgebrochen.

```
01 DATA 0,1,6,"Hallo Welt",0
02 ALERT 0
```

Sieht dann so aus:

4.15.2 ASK V,n

Zeigt eine Frage-Box an. Die Daten dazu befinden sich im Array ab Position n. Zuerst stehen dort Vordergrund- und Hintergrundfarbe, gefolgt von Nullterminiertem Text. Fehlt dder wert 0x00, wird die Ausgabe nach 20 Zeichen abgebrochen.

```
01 DATA 0,1,6,"Weitermachen?",0
02 ASK P,0
03 ? P
04 WAIT 20
```

Sieht dann so aus:

Wenn Y(y) gedrückt wurde, bekommt P den Wert 1 zugewiesen, wurde N(n) gedrückt, erhält P den Wert 0

4.16 Fehlermeldungen und Errorhandling

4.16.1 Fehlermeldungen

Insgesamt gibt es 32 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, meist wird in diesen Fällen aber SYNTAX ERROR angezeigt, da ungültige Schlüsselwörter nicht tokenisiert werden.
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	TOO MANY FOR	Insgesamt dürfen nur 16 FOR und GOSUB zu gleichen Zeit „geöffnet" sein
14	TOO MANY GOSUB	Insgesamt dürfen nur 16 FOR und GOSUB zu gleichen Zeit „geöffnet" sein, Ursache kann z.B. rekursives Programmieren sein.
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, eventuell gibt es einen Bug im Interpreter. 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...3)
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) 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
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

4.16.2 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.

4.16.3 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.

5 Changelog

Version 0.65 vom 14.1.2008

- Erste öffentliche Mega664-Version

Version 0.69 vom 31.1.2008

- Fehler in der Dekompressionsroutine behoben
- Fehler in den Bcopy-Routinen behoben
- Neue Sprite-Routinen
- Zusätzliche Array-Funktionen
- Zweite Klangfarbe und integrierter Sequenzer