# AVR-ChipBasic2: BASIC-Referenz

## Grundlagen


## 1 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).

## 2 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.

# 3 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)
```

## 3.1 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.

### 3.2 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.

### 3.3 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 Programmsteuerung

### 4.1 FAST

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

### 4.2 SLOW

Mit dem SLOW-Befehl wird die Bildschirmdarstellung wieder eingeschaltet.

### 4.3 BREAK

Setzt einen Breakpoint, ruft den Monitor auf.

### 4.4 END

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

### 4.5 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.

### 4.6 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.

### 4.7 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.

### 4.8 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.

### 4.9 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**.

## 5 Bildschirm-Ausgabe

### 5.1 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.

## 5.2 CLS

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

## 5.3 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.

## 5.4 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. Funktioniert nur im Bytezugriff auf das Array. |
| 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.

## 5.5 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.

## 5.6 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**.

## 5.7 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.

## 5.8 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.

## 5.9 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.

## 5.10 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.

### 5.11 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 |

# 6 Tastatur

## 6.1 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”.

## 6.2 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.

## 6.3 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.

## 6.4 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.

## 6.5 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 |

## 6.6 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 |

# 7 Zeit

## 7.1 WAIT n

Mit dem WAIT-Befehl wird N*0,1 Sekunden gewartet. N kann wieder ein beliebiger Ausdruck sein.

```
10 WA 10
```

wartet 1 Sekunde, bis das Programm fortgesetzt wird.

### 7.2 SYNC n

Mit dem SYNC-Befehl wird auf N Bildsynchronimpuse gewartet. N kann wieder ein beliebiger Ausdruck sein.

```
10 SYNC 300
```

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

### 7.3 TSET n

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

```
10 TSET 0
```

setzt den Timer auf 0.

### 7.4 TGET V

Der interne Timer (10Hz) wird ausgelesen und in die Varaiable V gespeichert.

```
10 TGET Z
```

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

## 8 Fehlermeldungen und Errorhandling

### 8.1 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.

### 8.2 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.

## 8.3 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 |