AVR-ChipBASIC: BASIC-Referenz

V0.87 (c) 2006/2007 Jörg Wolfram

1 Lizenz

2 Allgemeines

Da es reichlich Schlüsselwörter gibt, kann es sein, dass diese Referenz anfangs noch unvollständig ist. Viele Hinweise findet man auch in den Beispielprogrammen.
Jedes Programm besteht aus maximal 51 Programmzeilen (1-51). Alle Schlüsselwörter ausser Funtionen können auf zwei Zeichen abgekürzt werden, beim Laden werden abgekürzte Schlüsselwörter auch mit nur 2 Zeichen dargestellt. Nach jedem Schlüssselwort 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 16Bit Integerzahlen. Dazu gibt es 26 Variablen (A-Z) und ein Array mit 128 Elementen (@), 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 **#**
- Vergleiche (=,<>,<,<=,>,>=) liefern 0 oder 1

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
DIN(	liefert den Pegelwert des angegebenen Portpins (0-7) an der parallelen Schnittstelle
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
COL(	Kollisionsflag für Sprite 1...4 (0/1)

Zusätzlich zu den normalen Variablen gibt es noch die Arrayvariable **@**. Seit Version 0.87 ist das Array erheblich erweitert worden:

- Die Zellen 0...127 sprechen den Bereich im RAM wortweise an
- Die Zellen 128...255 sprechen den Puffer 1 im Dataflash wortweise an
- Die Zellen 256...383 sprechen den Puffer 2 im Dataflash wortweise an
- Die Zellen 1024...1279 sprechen den Bereich im RAM byteweise an
- Die Zellen 1280...1535 sprechen den Puffer 1 im Dataflash byteweise an
- Die Zellen 1536...1791 sprechen den Puffer 2 im Dataflash byteweise 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. Beispiele:

- A=-4
- B=SQR(8*2)
- @(B)=RND(6)+1

4.1.1 LIMIT v,min,max

Der Wert der Variable v wird auf den Wertebereich min...max begrenzt. Beispiele:

- LIMIT A,1,6
- 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...

Die Array Elemente werden ab Byte (offset) initialisiert. Beispiel:

- DATA 1024,”Hallo”,0

Das Array wird ab Element 1024 mit der nullterminierten Zeichenkette „Hallo” belegt. Im Word-Bereich des Arrays werden fehlende Bytes mit 0x00 aufgefüllt, das betrifft insbesondere Zeichenketten. Andersherum wird im Bytebereich des Arrays nur das Low-Byte geschrieben.

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

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

- GOTO 2
- GO B+1

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** kan auch weggelassen werden. Beispiele:

- IF A>5 THEN A=5
- IF A>5 A=5

4.2.7 FOR - NEXT

Bei der Schleifenabarbeitung gibt es nur die Grundform **FOR A=1 TO C** ohne die Angabe der Schrittweite. Da der Stack auf 9 Einträge begrenzt ist, lassen sich nur 9 Schleifen bzw. Unterprogrammaufrufe schachteln. Beispiel:

1. FOR A=0 TO 9
2. FOR B=0 TO 9
3. PLOT A,B,3
4. NEXT :NEXT

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

4.2.8 CALL - RETURN

Im Gegensatz zu den meisten BASIC-Dialekten heisst GOSUB hier CALL. Durch das verwendete Prinzip der Abkürzungen dürfen keine zwei Schlüsselwörter mit denselben beiden Buchstaben beginnen und GOTO beginnt schon mit „GO". **CALL 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 XCALL

Eine weitere, recht ungewöhnliche Anweisung ist **XCALL prognr,zeile** die Unterprogramme in einem der 3 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 Ausgabe

4.3.1 CLS

Mit dem CLS-Befehl wird der Bildschirm gelöscht. Beim Programmstart geschieht das automatisch.

4.3.2 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

4.3.3 PRINT

Der PRINT-Befehl dient zur Ausgabe auf den Bildschirm oder auf die serielle/parallele Schnittstelle. Zusätzlich kann die Ausgabe noch formatiert werden.

"TEXT"	der Text TEXT wird ausgegeben
#Expr	Festlegung des Ausgabekanals (0=Bildschirm, 1=seriell, 2=parallel)
!Expr	Stellt das Format ein (s.u.)
%Expr	Direkte Ausgabe eines Zeichens mit Zeichencode=Expr
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. 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=führende Leerzeichen 2=führende Nullen 3=führende Nullen mit Vorzeichen
Bit 0	2/4 Zeichen bei hexadezimaler Ausgabe

4.3.4 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:

- EMIT 64,$40

gibt zwei Klammeraffen aus.

4.3.5 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:

- 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**.

4.3.6 LCHAR n

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

- LCHAR 10

verschiebt die obersten 10 Zeichenzeilen um 1 Zeichen nach links.

4.3.7 GCHAR v,y,x

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

- GCHAR F,0,0

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

4.4 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 Hintergrundfarbe ist immer schwarz und die 4 Pixel eines Zeichens haben immer die gleiche Vordergrundfarbe.

4.4.1 COLOR

Mit dem COLOR-Befehl wird die Zeichenfarbe festgelegt. Diese wird bei PLOT, BOX und FBOX sowie bei der Ausgabe von großen Zeichen (Formatbit 6 gesetzt) ausgewertet. Akzeptiert werden Werte von 0 bis 7, dabei bedeutet 0=schwarz, 1=blau, 2=rot, 3=magenta, 4=grün, 5=cyan, 4=gelb und 7=weiss. Argument ist ein beliebiger Ausdruck. Beispiele:

- COLOR 2
- CO A

Ist die Farbe 0, werden die Pixel nur gelöscht, Farbinformationen werden nicht verändert.

4.4.2 PLOT Y,X

Mit dem PLOT-Befehl wird ein „Pixel” im Pseudografikmodus gesetzt. Dabei kann es passieren, dass benachbarte Punkte auch ihre Farbe wechseln. Beispiel:

- PLOT 3,2+X

4.4.3 DRAW Y1,X1,Y2,X2

Zeichnet eine Linie vo X1,Y1 nach X2,Y2 im Pseudografikmodus. Dabei kann es passieren, dass vorhandene Punkte in Liniennähe auch ihre Farbe wechseln. Beispiel:

- DRAW 0,0,45,59

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

4.4.4 BOX Y1,X1,Y2,X2

Mit dem BOX-Befehl wird ein Rechteck im Pseudografikmodus gezeichnet. Dabei kann es passieren, dass vorhandene Punkte auch ihre Farbe wechseln. Beispiel:

- BOX 1,1,10,10

4.4.5 FBOX Y1,X1,Y2,X2

Mit dem FBOX-Befehl wird ein gefülltes Rechteck im Pseudografikmodus gezeichnet. Dabei kann es passieren, dass vorhandene Punkte auch ihre Farbe wechseln. ist Y1=Y2 oder X1=X2 werden horizontale oder vertikale Linien gezeichnet. Beispiel:

- COLOR 1:FBOX 0,0,0,100

zeichnet eine waagerechte rote Linie am oberen Bildschirmrand, der Wert 100 wird zur Laufzeit auf den Bildschirmbereich begrenzt.

4.4.6 LPIX n

Es werden (am oberen Bildrand beginnend) n*2 Pixelzeilen um ein Pixel nach links verschoben. Bei verschiedenfarbigen Pixeln können Farbwechsel auftreten. Amrechten Rand rücken schwarze Pixel nach. Beispiel:

- LP 20

verschiebt die obersten 40 Pixelzeilen um 1 Pixel nach links.

4.5 Sprites

Die Sprites sind recht einfach „gestrickt”. Sie bestehen aus nur einem Zeichen, welches aber mehrfach nebeneinander oder übereinander dargestellt wird. 4 Sprites stehen zur Verfügung. Wird ein Sprite an einer neuen Stelle angezeigt, wird es vorher an der alten Stelle gelöscht, falls es sichtbar war.

4.5.1 SDEF n,t,c

Definiert ein Sprite, dabei ist n die Nummer des Sprites (1-4), t der Typ (s.u.) und c die Position des Zeichens in der Zeichentabelle. Der Typ t kann zwischen 0 und 15 liegen:

0,8	Ein einzelnes Zeichen
1-7	2 bis 8 Zeichen nebeneinander
9-15	2 bis 8 Zeichen untereinander

4.5.2 SSHOW n,y,x

Zeichnet das Sprite 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 zugehörige Kollisionsflag gesetzt und das Sprite nicht gezeichnet. Beispiel:

- SHOW 1,0,0

Zeichnet das Sprite 1 in die linke obere Ecke.

4.5.3 SHIDE n

Löscht das Sprite n. Beispiel:

- HIDE 1

Löscht das Sprite 1.

4.5.4 COL(n)

Kollisionsabfrage für Sprite n. Beispiel:

1. SHOW 1,0,0
2. IF COL(1)=0 THEN END
3. PRINT ”KOLLISION!”

Versucht Sprite 1 an die Position 0,0 zu zeichnen. Befindet sich dort bereits kein anderes Zeichen, wird das Programmm beendet. Andernfalls wird ein Text ausgegeben.

4.6 Audio

4.6.1 NOTE n

Ein Ton mit der Tonhöhe n (Halbtonschritte ab 220 Hz aufwärts) wird ausgegeben. Bei n=0 bis 63 werden Noten ausgegeben, bei n=255 Rauschen. Beispiel:

1. FOR N=0 to 63:NOTE N
2. WAIT 5:NEXT

gibt nacheinander alle spielbaren Noten aus.

4.7 Tastatur

4.7.1 INPUT

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.

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”.

4.7.2 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.

1. INPUT ”Text: ”,M
2. CTEXT 0,1024
3. X=1024
4. C=@(X):IF C=0 THEN END
5. PRINT %C;:X=X+1:GOTO 4

Der nach der Eingabeaufforderung eingegebene Text wird eine Zeile tiefer wiederholt.

4.7.3 RKEY V

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

- RKEY P

der aktuelle Tastaturcode wird in die Variable P geschrieben.

4.7.4 WKEY V

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

- WKEY P

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

4.7.5 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

4.8 Zeit

4.8.1 WAIT n

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

- WA 10

wartet 1 Sekunde, bis das Programm fortgesetzt wird.

4.8.2 SYNC n

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

- SYNC 300

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

4.8.3 TSET n

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

- TSET 0

setzt den Timer auf 0.

4.8.4 TGET V

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

- TGET Z

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

4.9 Ein-/Ausgabe

4.9.1 DIR n

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

- DIR $F0

Pin D0-D3 werden als Eingang, D4-D7 als Ausgang konfiguriert.

4.9.2 OUT n,b

Setzen (b=1) oder Rücksetzen (b=0) des Ausganges n. Beispiel:

- OUT 7,0

Setzt D0 auf 0-Pegel.

4.9.3 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 zu übertragenden Bytes an. Beispiel:

- 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 Adresse übertragen 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.

4.10 Serielles

4.10.1 PUMP n

Schaltet die Ladungspumpe aus (n=0) oder ein (n=1). Beispiel:

- PU 1

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

4.10.2 SPUT n

das Byten wird an die serielle Schnittstelle ausgegeben. Beispiel:

- SPUT 10

gibt einen Zeilenvorschub an die serielle Schnittstelle aus.

4.10.3 SGET V

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

- SGET C

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

4.11 Speicher

4.11.1 ESET dat

Speichert einen Wert im internen EEPROM. Dieser Wert ist programmbezogen. Nach dem Laden über die serielle Schnittstelle wird der Wert auf 0 gesetzt. Beispiel:

- ESET C

speichert den Wert der Variablen C in den internen EEPROM.

4.11.2 EGET V

Liest einen Wert aus dem internen EEPROM. Dieser Wert ist programmbezogen. Beispiel:

- EGET C

liest den Wert aus dem EEPROM in die Variable C.

4.11.3 XPOKE adr,dat

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

- XPOKE 100,C

speichert den Wert der Variablen C an die Adresse 100.

4.11.4 EPOKE adr,dat

Speichert ein Word im internen EEPROM. Als Adressen sind 0...999 möglich.

- EPOKE V,$12

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

4.11.5 FREAD nr,page

Liest die Page **page** aus dem Dataflash in den Puffer **nr** (1/2) ein. Beim ATAT45DB081 kann page zwischen 0 und 4095 liegen.

- FR 1,0

liest die Page 0 in den Puffer 1 (Array 128...255).

4.11.6 FWRITE nr,page

Schreibt den Puffer **nr** (1/2) in die Page **page** des Dataflash. Beim ATAT45DB081 kann page zwischen 0 und 4095 liegen.

- FR 1,0

schreibt den Puffer 1 (Array 128...255) in die Page 0.

5 Changelog

Version 0.84 vom 16.3.2007

- Erste öffentliche Mega32-Version

Osterei-Version 0.87 vom 5.4.2007

- Erweiterte PRINT Funktionalität (direkte Zeichenausgabe)
- Universelle I2C Funktion
- Auf 256 Bytes / 128 Words vergrössertes Array
- Einfache Array-Belegung mit DATA
- Ein paar neue Zeichen im Zeichensatz
- Dataflash-Routinen