uBASIC

Ein plattformunabhängiger BASIC-Interpreter

Uwe Berger; 2011

```
Line       = number statement "\\n";
statement  = PRINT printlist |
             [LET] var "=" expression |
             GOTO expression |
             GOSUB (expression | characterstr) |
             RETURN |
             IF expression relop expression (THEN statement [ELSE statement]|[THEN] statement) |
             FOR var "=" expression ( TO | DOWNTO ) expression [STEP expression]|
             NEXT var |
             REM { any_character } |
             DIM var "(" expression ")" |
             SRAND |
             WAIT expression |
             OUT "(" characterstr "," expression ")" "=" expression |
             EPOKE "(" expression ")" "=" expression |
             VPOKE "(" characterstr ")" "=" expression |
             ABS "(" expression ")" |
             NOT "(" expression ")" |
             EPEEK "(" expression ")" |
             ADC "(" expression ")" |
             IN "(" characterstr "," expression ")" |
number     = decnumber | hexnumber | dualnumber;
decnumber  = decdigit {decdigit};
hexnumber  = "0" ("x"|"X") hexdigit {hexdigit};
dualnumber = "0" ("b"|"b") dualdigit {dualdigit};
separator  = "," | ";";
var        = ("A" | "..." | "Z" | "a" | "..." | "z") ["(" expression ")"];
decdigit   = "0" | "..." | "9";
hexdigit   = "0" | "..." | "9" | "A" | "..." | "F" | "a" | "..." | "f";
dualdigit  = "0" | "1";
relop      = "<" | ">" | "=" | "<=" | ">=" | "<>" | "<>";
operator   = "+" | "-" | "*" | "/" | "%" | "mod" | "|" | "or" | "&" | "and" | "xor" | "shl" | "shr";
characterstr    = "\""  { any_character } "\"";
```

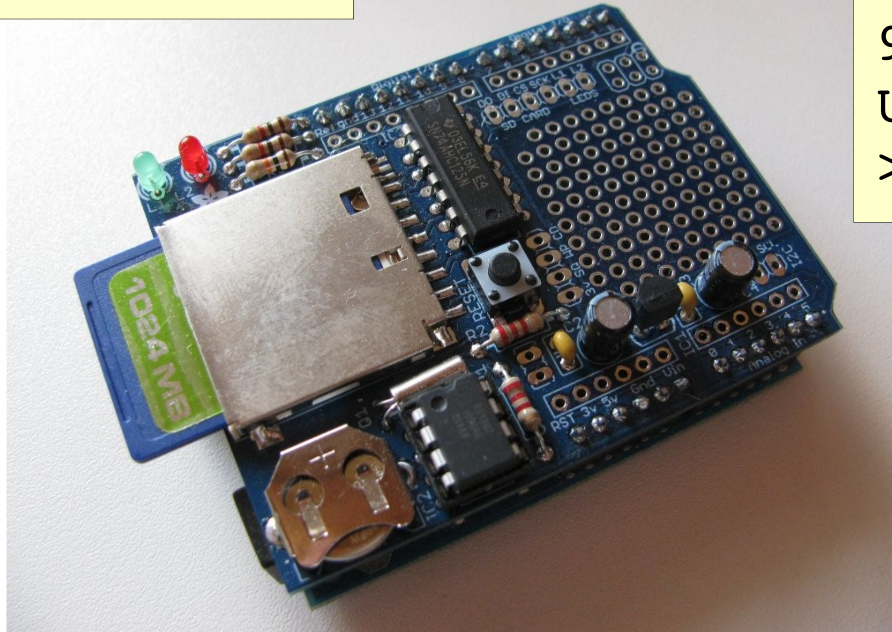
Uwe Berger

- Beruf: Softwareentwickler (PPS-Systeme)
- Linux seit ca. 1995
- Freizeit: Hard- und Softwarespielereien
- Brandenburger Linux User Group e.V. (BraLUG)
- Brandenburger Linux Infotag (BLIT)
  → 8.BLIT am 05.11.2011 in Potsdam
  → 9.BLIT am 03.11.2012(?)

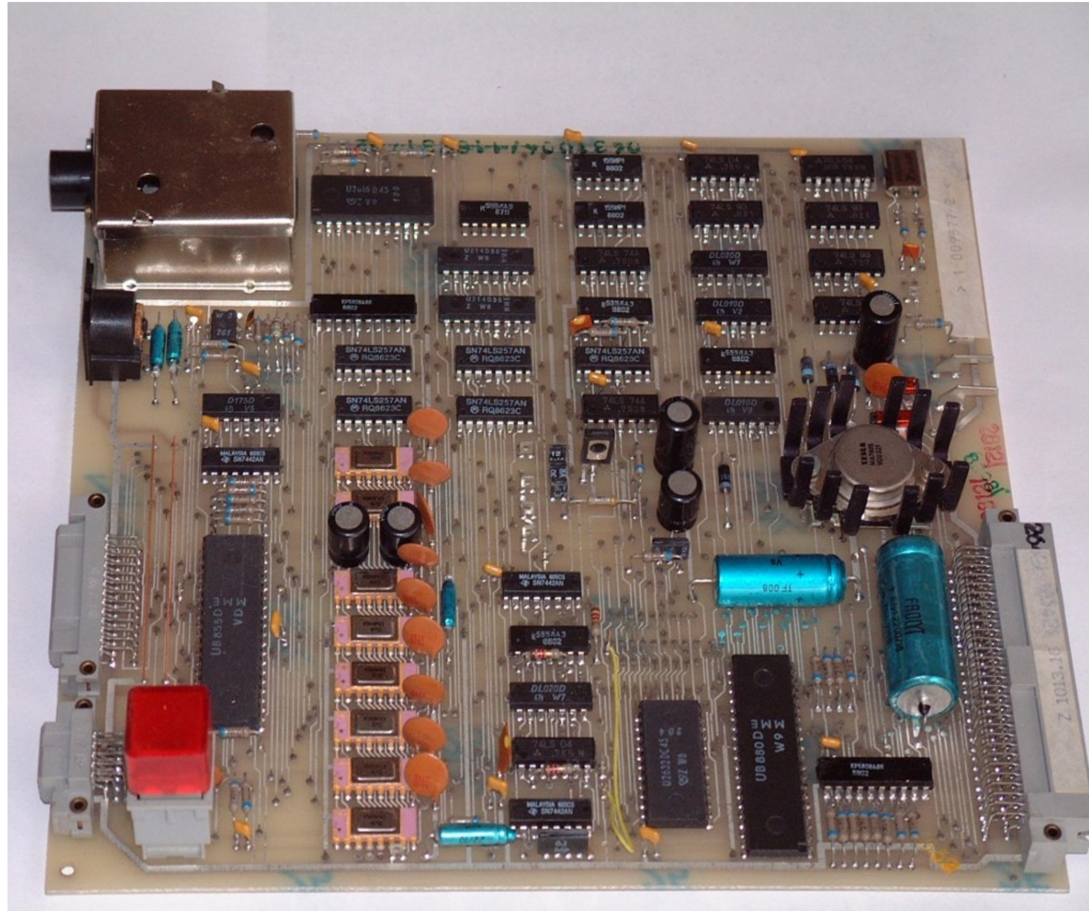
Inhalt

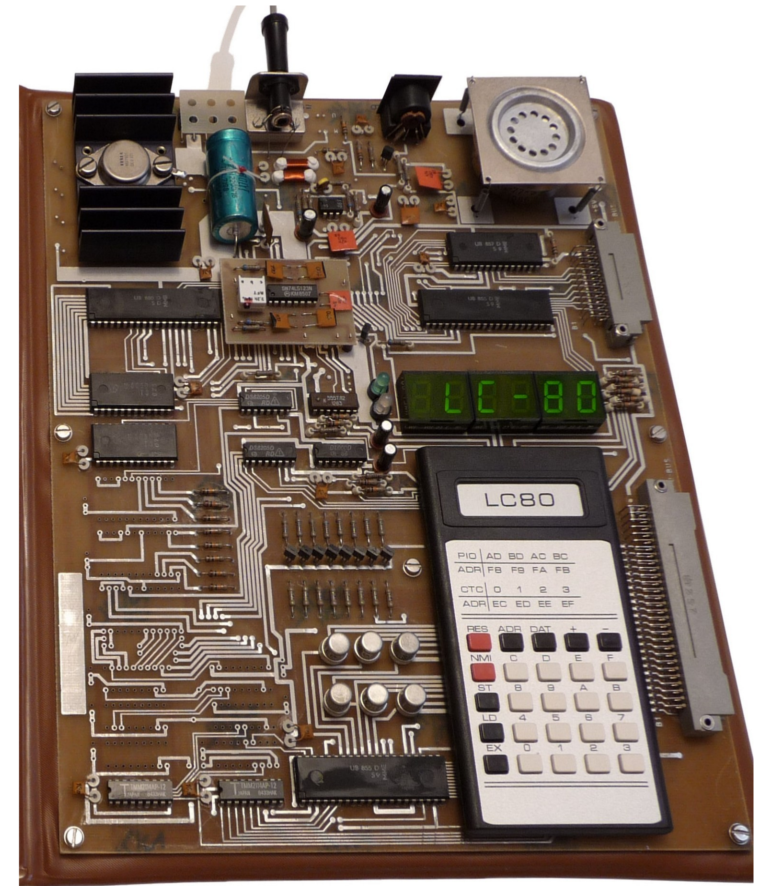
- Warum ein weiterer Interpreter?
- Etwas Theorie...
- uBASIC

```
10  PRINT “Hallo BLIT!“
20  FOR A=1 TO 10
30  IF A%2 THEN PRINT A
40  NEXT A
50  GOSUB 100
60  END
100 PRINT “UP“
110 RETURN
```



```
Hallo BLIT!
1
3
5
7
9
UP
>
```





Bildquellen: Wikipedia

Warum?

Was wurde gesucht?

- eine Möglichkeit bestehende Applikationen einfach um weitere Funktionalität erweitern zu können ohne sie neu übersetzen zu müssen
- ein Interpreter, der auch auf ressourcenarmen Plattformen läuft (z.B. AVR-MCU)
- ein Interpreter der leicht erweiterbar und konfigurierbar ist
- eine Codebasis für die unterschiedlichsten Hardwareplatt-formen
- ein allgemein bekannter Sprachsyntax → BASIC
- eine „akademische“ Herausforderung...

Was gibt es schon für AVR-MCUs?

- AVR-ChipBasic: http://www.jcwolfram.de
- BasicBeetle: http://www.dieprojektseite.de
- ...?

Aber:

- es handelt sich um vollständige und abgeschlossene „BASIC-Computer“
- Firmware ist gänzlich in Assembler geschrieben
- Code ist nicht portierbar

...und was gibt es noch?

- gefunden wurde aber auch eine ideale Ausgangsbasis für eine Eigenentwicklung:

Adam Dunkle: „uBASIC – A really tiny BASIC interpreter“

(http://www.sics.se/~adam/ubasic/)

- auch neulich gefunden...: uBasic in CHDK (**C**anon **H**ack **D**evelopment **K**it) → http://chdk.wikia.com/wiki/UBASIC

Theorie...

Compiler, Interpreter

- Interpreter:
  - analysiert ein Quellprogramm unmittelbar zur Laufzeit und führt die einzelnen Befehle sofort aus
  - das Quellprogramm liegt in der Regel in einer lesbaren Form vor
- Compiler:
  - übersetzt ein Programm von einer Quellsprache in das semantische Äquivalent einer Zielsprache
  - in der Regel wird als Quellsprache eine lesbare Form und als Zielsprache meist direkt ausführbarer Code verwendet
  - die Ausführung des Programms erfolgt asynchron nach der Übersetzung

Compiler, Interpreter (Funktionsweise)

Interpreter

Programmquelltext

Quelltextanalyse

- Lexikalische Analyse
- Syntaktische Analyse
- Semantische Analyse

Programmausführung

Compiler

Sprachsynthese

Libs, Binder, Linker...

Zielsprache

Programmausführung

Vor-/Nachteile Interpreter

- Vorteile:
  - Quellprogramm plattformunabhängig, es wird lediglich ein plattformspezifischer Interpreter benötigt
  - effizientere Entwicklungsphase (auch Fehlersuche), da der Übersetzungsvorgang entfällt → das was im Quelltext steht, wird auch wirklich ausgeführt
- Nachteile:
  - geringere Ausführungsgeschwindigkeit, da Quelltext während der Ausführung (teilweise mehrmals → z.B. Schleifen) analysiert werden muss

Vor-/Nachteile Compiler

- Vorteile:
  - hohe Ausführungsgeschwindigkeit, da Zielsprache in der Regel maschinennah
  - zeitaufwendige Quellcodeanalyse erfolgt einmal zum Zeitpunkt der Übersetzung
  - plattformspezifische Optimierungsmöglichkeiten
- Nachteile:
  - erzeugter Code ist nicht plattformunabhängig, für jede Plattform wird eine eigene Toolchain benötigt
  - zur Fehlersuche werden immer plattformspezifische Werkzeuge benötigt

„Zwischen“ Compiler und Interpreter...

- JIT-Compiler:
  - Just-in-Time
  - Programmcode wird zur Laufzeit vollständig in Maschinencode übersetzt und danach sofort ausgeführt
- Bytecode-Interpreter:
  - Quellcode wird in einen einfacheren, optimierten und plattformunabhängigen Zwischencode übersetzt
  - die Ausführung des Bytecode erfolgt in einem plattformspezifischen Interpreter (VM)

Compiler-/Interpreterbau (Hilfsmittel)

- siehe diverse Standardliteratur der Informatik
- Syntaxbeschreibung → z.B. Backus-Naur-Form
- Werkzeuge zur Generierung von Quelltextparser:
  - Programmierung von Hand ;-)
  - automatische Parser-Generatoren: Lex, Flex, Yacc, Bison etc.
- Werkzeuge zur automatsichen Compilergenerierung (Sprachsynthese) sind derzeit noch Forschungsgegen-stand

Die Sprache BASIC

- BASIC → Beginner's All-purpose Symbolic Instruction Code
- imperative Programmiersprache: eine Berechnung wird durch die Folge von Anweisungen beschrieben
- 1964: John George Kemeny, Thomas Eugene Kurtz
- Ansatz:
  - *„Programmieren lernen ohne Vorkenntnisse."*
  - *„Die Hardwarespezifika sind nicht Gegenstand des Programms."*
- sehr weite Verbreitung durch Heimcomputer der 80'er

uBASIC...

uBASIC: allgemeine Merkmale

- Interpreter ist vollständig in C geschrieben
- einfach in eigene Applikationen integrierbar
- Schnittstellen/Abstraktionsschichten zur einbettenden Applikation vorhanden
- modularer und verständlicher Aufbau; leicht erweiterbar
- extrem ressourcenschonend (z.B. ATmega168: 16kByte Flash, 1kByte SRAM)
- Basic-Sprachumfang mit TinyBASIC vergleichbar
- „intelligente“ Mechanismen zur Laufzeitverbesserung

uBASIC: Sprachumfang

- Variablen und eindimensionale Felder (DIM)
- Dezimal-, Dual-, Hexadezimal-Zahlenformat
- Arithmetische, Bit- und Vergleichs-Operatoren
- Schleifen (FOR-NEXT); auch DOWNTO und STEP
- bedingte Anweisung (IF-THEN-ELSE)
- Sprünge (GOTO)
- Unterprogramme (GOSUB)
- Ausgabe-/Eingabebefehle (PRINT, INPUT)

uBASIC: Sprachumfang

- DATA/READ/RESTORE-Anweisung
- diverse Funktionen zur String-Verarbeitung
- diverse weitere Funktionen (…, siehe Dokumentation!)
- BASIC-Programme ohne Zeilennummerierung möglich

```
Line          = number statement "\\n";
statement     = PRINT printlist |
                [LET] var "=" expression |
                GOTO expression |
                GOSUB (expression | characterstr) |
                RETURN |
                IF expression relop expression (THEN statement [ELSE statement]|[THEN] statement) |
                FOR var "=" expression ( TO | DOWNTO ) expression [STEP expression]|
                NEXT var |
                REM { any_character } |
                DIM var "(" expression ")" |
                SRAND |
                WAIT expression |
                OUT "(" characterstr "," expression ")" "=" expression |
                EPOKE "(" expression ")" "=" expression |
                VPOKE "(" characterstr ")" "=" expression |
                DIR "(" characterstr "," expression ")" "=" expression |
                CALL "(" characterstr "," exprlist ")";
printlist     = [printitem | printitem separator printlist];
printitem     = (expression | characterstr);
varlist       = var | var "," varlist;
exprlist      = expression | expression "," exprlist;
expression    = ["-"|"+"] (term | "(" term ")");
term          = factor | factor operator term;
factor        = var | ["+" | "-"] number | expression | function;
function      = RAND "(" expression ")" |
                ABS "(" expression ")" |
                NOT "(" expression ")" |
                EPEEK "(" expression ")" |
                ADC "(" expression ")" |
                IN "(" characterstr "," expression ")" |
                CALL "(" characterstr "," exprlist ")" |
                VPEEK "(" characterstr ")"      ;
number        = decnumber | hexnumber | dualnumber;
decnumber     = decdigit {decdigit};
hexnumber     = "0" ("x"|"X") hexdigit {hexdigit};
dualnumber    = "0" ("b"|"b") dualdigit {dualdigit};
separator     = "," | ";";
var           = ("A" | "..." | "Z" | "a" | "..." | "z") ["(" expression ")"];
decdigit      = "0" | "..." | "9";
hexdigit      = "0" | "..." | "9" | "A" | "..." | "F" | "a" | "..." | "f";
dualdigit     = "0" | "1";
relop         = "<" | ">" | "=" | "<=" | ">=" | "<>" | "<>";
operator      = "+" | "-" | "*" | "/" | "%" | "mod" | "|" | "or" | "&" | "and" | "xor" | "shl" | "shr";
characterstr  = "\""  { any_character } "\"";
```

Stand: 04/2011

uBASIC: Einschränkungen

- maximal 26 Variablen (signed integer); Variablennamen nur einzelne Buchstaben
- ein BASIC-Befehl pro Programmzeile
- endlicher (aber konfigurierbarer) Speicherplatz für diverse interne Caches und Stacks

uBASIC: Funktionsweise

- „Klassisches“ Konzept der Quelltextanalyse
- tokenizer.c (der Parser)
  - Überlesen von Whitespaces im BASIC-Programm
  - Token-Erkennung → *lexikalische Analyse*
- ubasic.c (der eigentliche Interpreter)
  - schrittweises Zusammensetzen der erkannten Token nach den jeweiligen Vorgaben → *syntaktische Analyse*
  - Überprüfung der Richtigkeit des Syntax und der Randbedingungen → *semantische Analyse*
  - Ausführung der erkannten Anweisung oder Teile davon, sobald sie einen Sinn ergeben

uBASIC: Funktionsweise (Tokenizer)

- zwei Algorithmen zur Tokenerkennung implementiert

„Standard-Parser“

- Kombinationen Schlüsselwort/Token stehen in einer internen Tabelle
- Tabelle wird für jedes Schlüsselwort zeilenweise durchsucht
- Vorteil: transparenter Quelltext
- Nachteil: langsam

„Fast-Parser“ (René Böllhoff)

- Schlüsselwörter sind zeichenweise in einem internen Binärbaum abgelegt
- Suchvorgang erfolgt zeichenweise entlang der Baumstruktur bis zu einem „Blatt“ → Token gefunden
- Vorteil: sehr schnell (Faktor 2-3)
- Nachteil: Extra-Tool zur Binärbaum-Generierung notwendig

uBASIC: Funktionsweise (Interpreter)

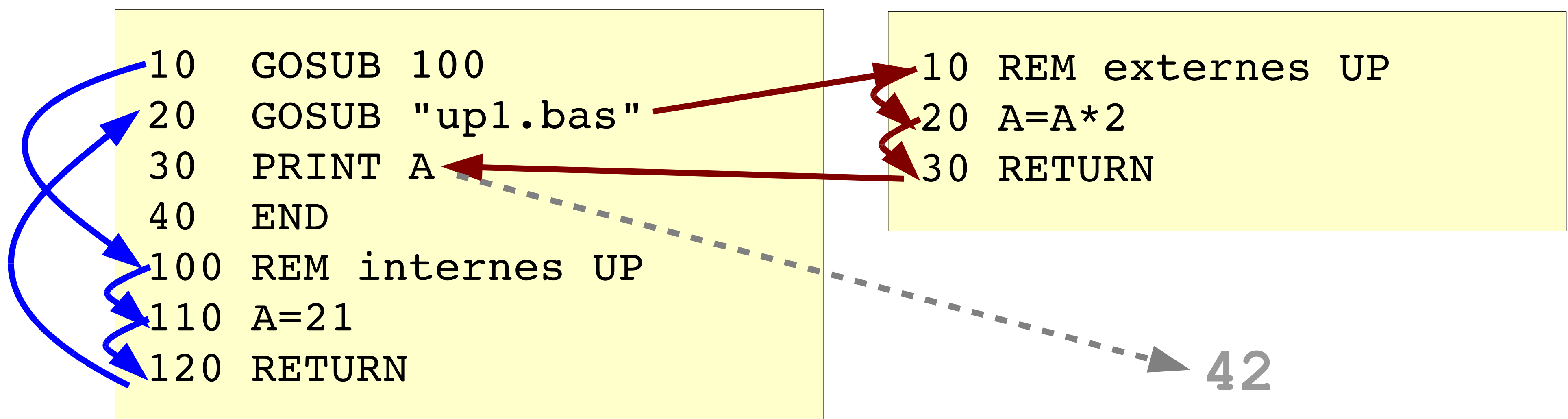
```
static void dim_statement(void) {
  int var, dim;
  accept(TOKENIZER_DIM);
  var = tokenizer_variable_num();
  accept(TOKENIZER_VARIABLE);
  accept(TOKENIZER_LEFTPAREN);
  dim=expr();
  variables[var].dim=dim;
  variables[var].adr=malloc(dim*sizeof(int));
  if (variables[var].adr == NULL  ) {
    tokenizer_error_print(current_linenum, 42);
    ubasic_break();
  }
  accept(TOKENIZER_RIGHTPAREN);
  accept(TOKENIZER_CR);
}
```

```
10 DIM A(N*10)
```

```
int expr(void) {
    int val;
    /* ... */
    return val;
}
```

uBASIC: GOSUB

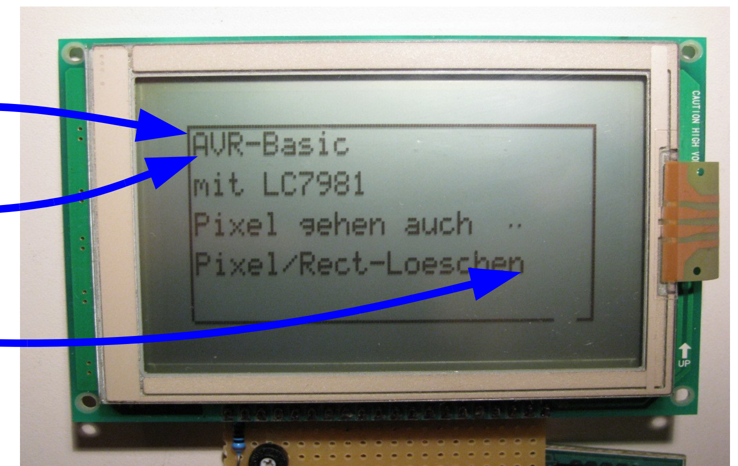
- Standard: GOSUB n → Aufruf eines Unterprogramms ab der Zeile n; Rücksprung bei Auftreten eines RETURN
- Erweiterung: das Unterprogramm kann auch in einem externen „Quelltext-Bereich“ (z.B. Datei) liegen



uBASIC: externe Routinen/Variablen

- BASIC-Befehl zum Aufruf von Routinen der einbettenden Anwendung: CALL()
- BASIC-Befehl zum Lesen/Schreiben von Variablen der einbettenden Anwendung: VPEEK(), VPOKE()
- Routinen/Variablen müssen entsprechend dem BASIC-Interpreter bekannt gemacht werden

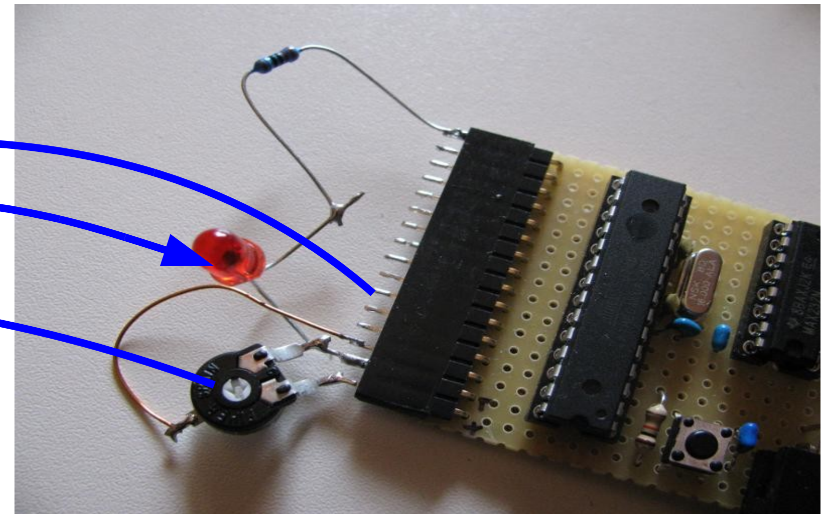
```
10 CALL("clear")
20 CALL("line", 10, 10, 149, 10)
30 CALL("puts", 12, 12, "AVR-Basic")
40 CALL("pset", 120, 40)
50 REM ...
60 END
```



uBASIC: AVR-MCU-spezifisches

- Ein-/Auslesen von I/O-Ports: DIR() , IN() , OUT()
- Ein-/Auslesen des EEPROM: EPOKE(), EPEEK()
- BASIC-Befehl zum Auslesen des ADC: ADC()

```
10 DIR(“b“,1)=1
20 DIR(“b“,2)=0
20 A=0
40 PRINT IN(“b“,2)
30 OUT(“b“,1)=A
40 PRINT ADC(0)
50 IF A=1 THEN A=0 ELSE A=1
60 WAIT 1000
70 GOTO 30
80 END
```



uBASIC: Programmmedium

- der BASIC-Quelltext ist während der Abarbeitung ständig lesend im Zugriff (Parser/Tokenizer)
- der Quelltext kann/soll auf den unterschiedlichsten Medien stehen (RAM, SD-Karte, Festplatte, PROM etc.)
- die Zugriffe auf den BASIC-Quelltext lassen sich auf einige wenige Dinge reduzieren

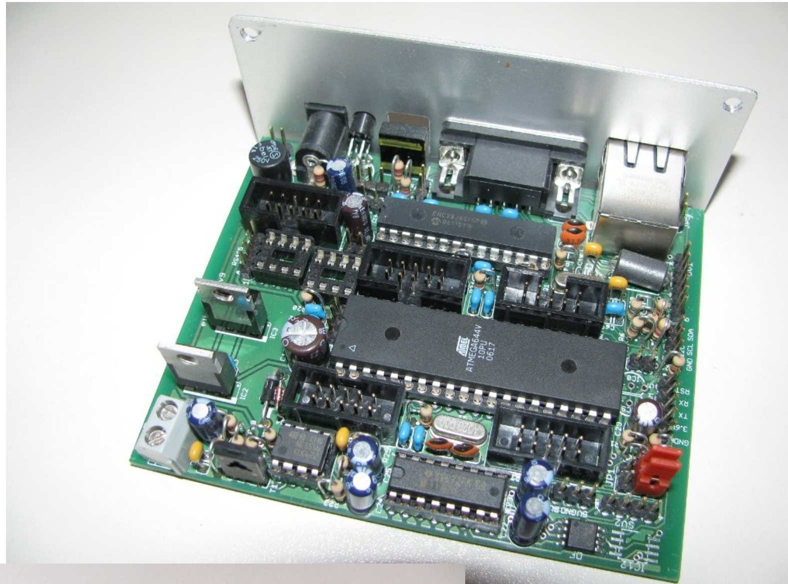
```
#define PTR_TYPE                       char const *
static  PTR_TYPE                       ptr;
#define PROG_PTR                       ptr
#define SET_PROG_PTR_ABSOLUT(param)    (PROG_PTR = (param))
#define GET_CONTENT_PROG_PTR           *ptr
#define INCR_PROG_PTR                  ++ptr
#define END_OF_PROG_TEXT               GET_CONTENT_PROG_PTR == 0
```

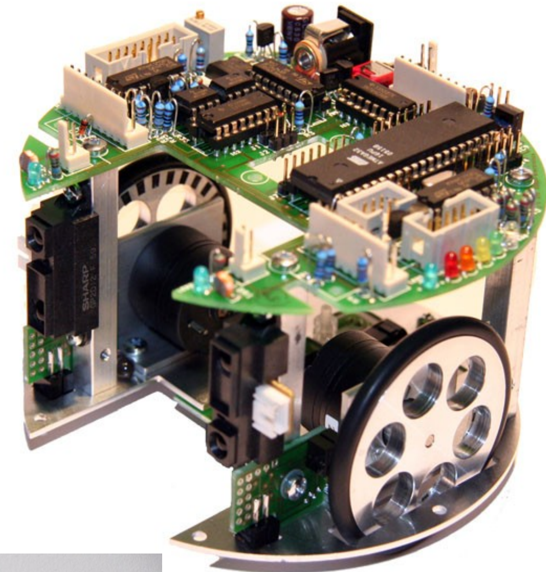
uBASIC: Einbindung in Applikationen

- Konfiguration an Gegebenheiten anpassen
- entsprechende Header-Dateien inkludieren
- … und ein paar Zeilen im Rahmenprogramm:

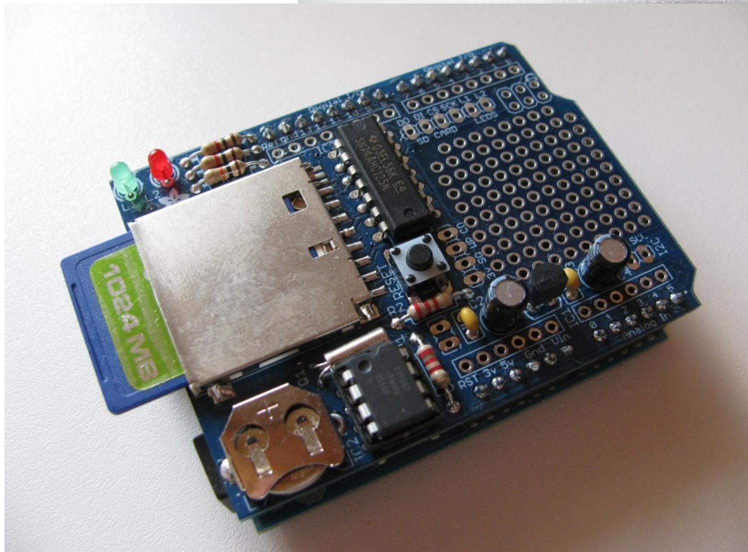
```
/* BASIC-Programm oeffnen */
/* …                      */

ubasic_init(zeiger_auf_programmanfang);
do {
      ubasic_run();
      /* … was auch immer … */
   } while(!ubasic_finished());
```

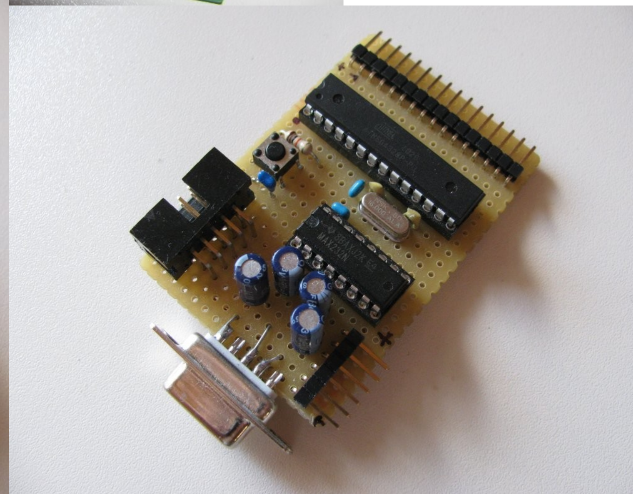
uBASIC: ein paar Plattformen...





Quelle:http://wiki.ctbot.de





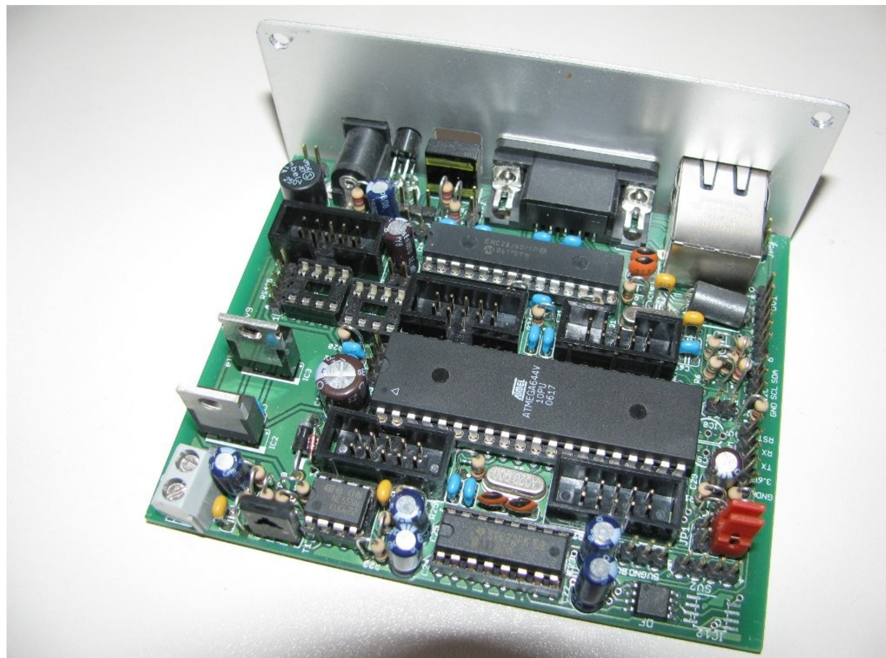
uBASIC: Umfeld

- smed: small editor
- ein kleiner Editor, welcher auf einer ressourcenarmen MCU läuft
- VT100- kompatibler Terminalclient notwendig
- Bibliothek `mcurses` von Frank Meier

```
~/work/mikrocontroller.net/avr-basic/uwe_berger
Datei  Bearbeiten  Ansicht  Terminal  Reiter  Hilfe
~/work/mikrocontroller.net/avr-basi...    ~/work/uart
 smed - Version 0.2                         bubble
10   b=20
20   rem ***Zufallszahen erzeugen***
30   dim a(b)
40   srand
50   for i=0 to b-1
60   a(i)=rand(100)
80   next i
90   gosub 1000
100  print
110  print "**********"
120  rem ***Bubblesort***
130  for i=0 to b-1
140  for j=0 to b-1
150  if a(j) > a(i) gosub 1100
160  next j
170  next i
180  gosub 1000
190  print
200  end
1000 rem ***Array ausgeben***
1010 for i=0 to b-1
1020 print a(i),
 ROW:5     COL:4     POS:76     ALL:498                    by Uwe Berger, 2011
```

uBASIC: Live-Vorführung

- innerhalb einer Linux-Applikation
- als Firmwarebestandteil auf einem AVR-Mikrocontroller



```
: ~/work/ubasic_avr
Datei  Bearbeiten  Ansicht  Terminal  Reiter  Hilfe
: ~/work/ubasic_avr        ~/work/ubasic_avr
AVR-uBasic-Interpreter; Uwe Berger, 2010
Compiliert am May  7 2010 um 17:19:35
Compiliert mit GCC Version 4.3.0
list
10 gosub 100
15 a=30
20 for i = 1 to a step 10
30 print "i=", i
40 next i
50 end
100 print "subroutine"
110 return
run
subroutine
i= 1
i= 11
i= 21
```

Weitere Informationen:

- http://www.sics.se/~adam/ubasic/
- http://www.mikrocontroller.net/topic/177030
- http://www.mikrocontroller.net/articles/AVR_BASIC
- http://www.mikrocontroller.net/svnbrowser/avr-basic/
- http://www.mikrocontroller.net/topic/227312
- http://www.mikrocontroller.net/articles/MCURSES
- „Visionäre der Programmierung“; Biancuzzi & Warden; O'Reilly
- http://www.ittybittycomputers.com/IttyBitty/TinyBasic/TBuserMan.htm
- http://www.ittybittycomputers.com/IttyBitty/TinyBasic/TBEK.txt

Danke...!