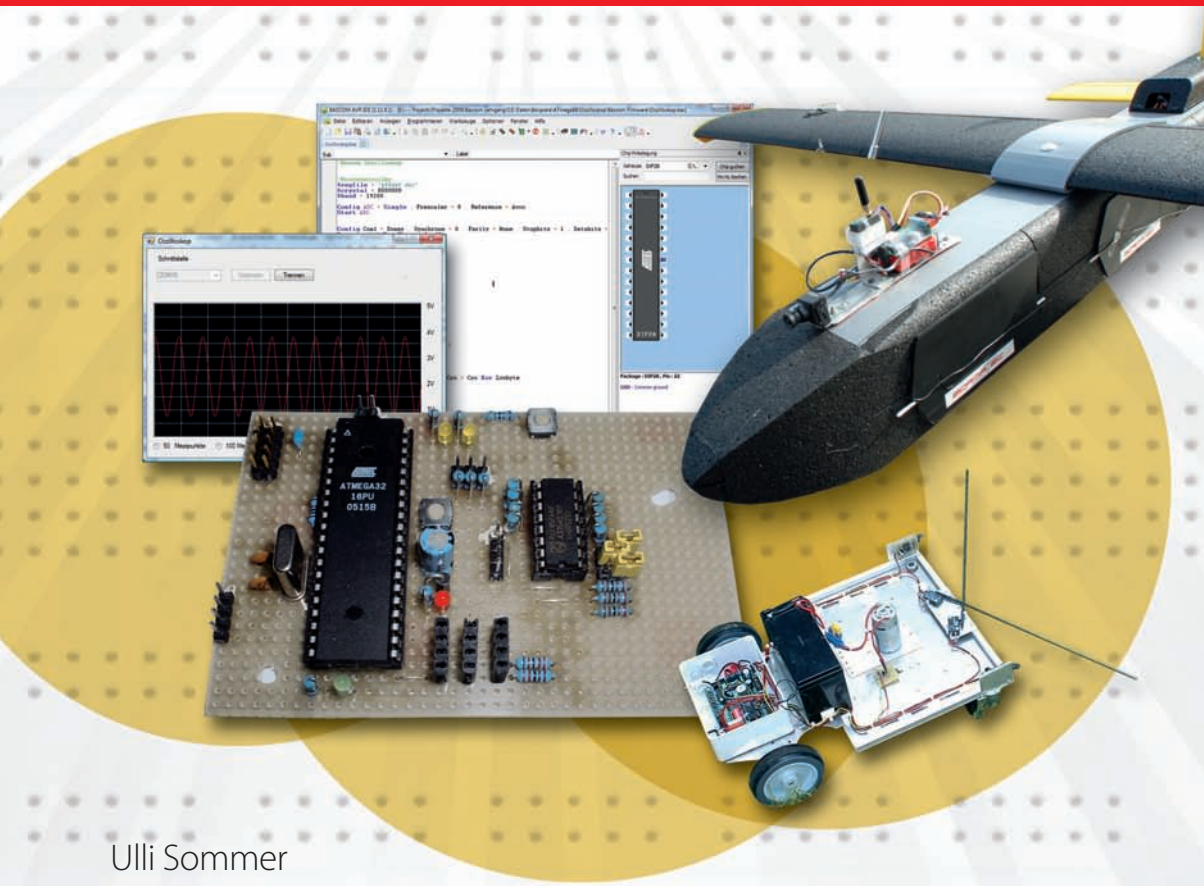
FRANZIS
PC & ELEKTRONIK



Ulli Sommer

Mikrocontroller programmieren mit Bascom Basic

Messen, Steuern, Regeln und Robotertechnik mit den AVR-Controllern

Auf CD-ROM:

- Komplette Software für den Einstieg
- Über 100 Quellcodes zu den Experimenten
- Open-Source-VB.NET-Programme zum Messen und Steuern

Ulli Sommer

Mikrocontroller programmieren mit Bascom Basic

FRANZIS
PC & ELEKTRONIK

Ulli Sommer

Mikrocontroller programmieren mit Bascom Basic

Messen, Steuern, Regeln und Robotertechnik mit den AVR-Controllern

Mit 161 Abbildungen

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie;
detaillierte Daten sind im Internet über http://dnb.ddb.de abrufbar.

Satz: DTP-Satz A. Kugge, München
art & design: www.ideehoch2.de
Druck: Bercker, 47623 Kevelaer
Printed in Germany

ISBN 978-3-645-65041-0

Vorwort

Mikrocontroller zu programmieren wird, wie man in den verschiedensten Foren und Fachzeitschriften beobachten kann, immer populärer. Das liegt daran, dass Mikrocontroller und zusätzliche Peripheriebausteine immer günstiger angeboten werden und an Schulen zunehmend in Mikrocontroller und Computertechnik unterrichtet wird. Schaltungen, die man früher mit mehreren ICs aufbauen musste, können nun in einem einzigen Mikrocontroller untergebracht werden. Das spart Zeit, Geld und Platz auf der Leiterplatte. Dieses Buch baut auf dem beliebten Basic-Compiler BASCOM-AVR mit integrierter Entwicklungsumgebung auf. Er ist für fast alle 8-Bit-AVR- und XMega-Mikrocontroller der Firma Atmel geeignet. BASCOM erfreut sich nicht nur bei Einsteigern, sondern auch bei Entwicklungsprofis, immer größerer Beliebtheit und stellt inzwischen schon fast einen Basic-Standard bei AVRs dar.

Viele Problemstellungen, die früher zeitaufwendig in Assembler oder C gelöst werden mussten, können durch diesen modernen Compiler blitzschnell mit wenigen Befehlen erledigt werden. Beispielsweise genügt ein einziger Befehl, um aus einem I/O-Port eine RS232-Schnittstelle, einen I²C-Bus oder einen Servoanschluss zu machen. Solche Dinge erfordern in anderen Programmiersprachen oft einen enormen Aufwand.

BASCOM erzeugt optimierten Maschinen-Code. Es werden alle AVR-RISC-Controller mit internem RAM der Serien AT90S, ATmega und ATTiny unterstützt. Mit einigen Einschränkungen sind jetzt auch ATTiny-Controller ohne SRAM mit BASCOM-AVR programmierbar. Dazu steht die $TINY-Funktion zur Verfügung.

Aus diesen Gründen ist der BASCOM Basic-Compiler ideal für den Einstieg in die Mikrocontroller-Programmierung geeignet. Er ist trotzdem sehr leistungsfähig und ermöglicht auch optimierte komplexe Software-Entwicklungen mit Profianforderungen. Ein weiterer großer Vorteil ist, dass diese Entwicklungsumgebung in hohem Tempo weiterentwickelt wird und die Updates kostenlos sind. So war BASCOM auch eine der ersten AVR-Entwicklungsumgebungen, die unter Vista und Windows 7 liefen.

Ich wünsche Ihnen viel Spaß beim Lesen und Experimentieren mit diesem Buch!

Waidhaus, Juli 2011

Ulli Sommer

Inhaltsverzeichnis

6 Programmiergeräte

Wenn Sie hauptsächlich mit BASCOM und 8-Bit-Atmel-Controllern arbeiten möchten, empfehle ich, den BASCOM USB-ISP-Programmer zu verwenden. Er kann direkt mit BASCOM verwendet werden und man kann zudem auch die Fuse- und Lock-Bits unter BASCOM einstellen. Mit der USB-Schnittstelle ist er nicht nur sehr schnell bei der Programmierung, sondern erlaubt auch den Betrieb an neueren Rechnern, die meist nur noch über USB-Schnittstellen verfügen.

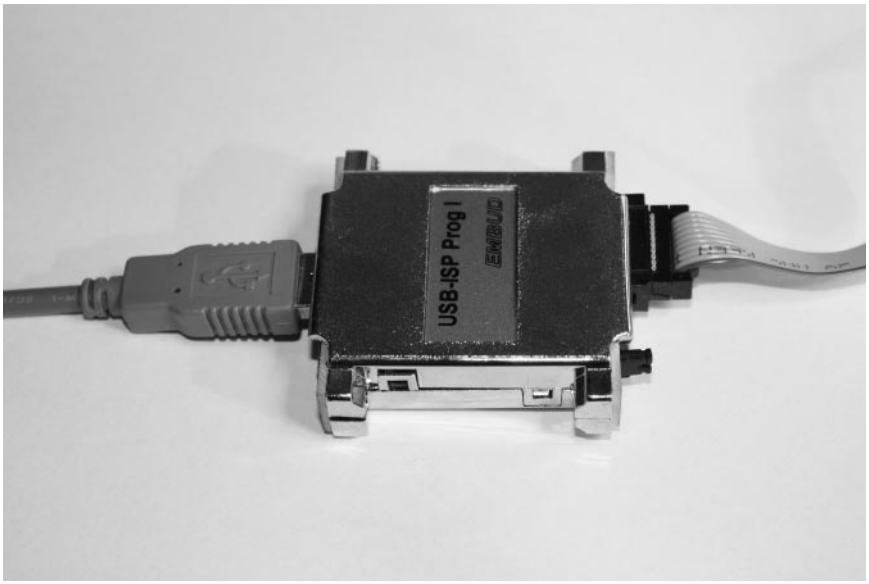


Bild 6.1: BASCOM USB-ISP-Programmer.

Beziehen lässt sich der Programmer direkt vom BASCOM-Hersteller oder in Deutschland über *www.Robotikhardware.de*.

Weit verbreitet ist auch der Atmel USB-ISP-Programmer MKII. Er wird nicht direkt unter BASCOM unterstützt, lässt sich aber auch dafür einrichten – jedoch nicht so komfortabel wie der BASCOM-eigene Programmer. Eine Aufstellung der unter BASCOM verwendbaren Programmiergeräte finden Sie unter *http://avrhelp.mcselec.com/index.html?supported_programmers.htm*.

In diesem Buch wird der BASCOM USB-ISP-Programmer verwendet. Die Art, wie bei der Programmierung vorgegangen wird, ist bei anderen Programmiergeräten ähnlich.

Eine kostengünstige Alternative zu käuflich erwerbbaren Programmiergeräten ist der Eigenbau. Es werden nicht allzu viele Bauteile benötigt, womöglich finden sich die Teile sogar in Ihrer Bastelkiste. Der zu verwendende Computer muss jedoch eine parallele Schnittstelle besitzen. Der folgende Schaltplan zeigt, wie es geht.

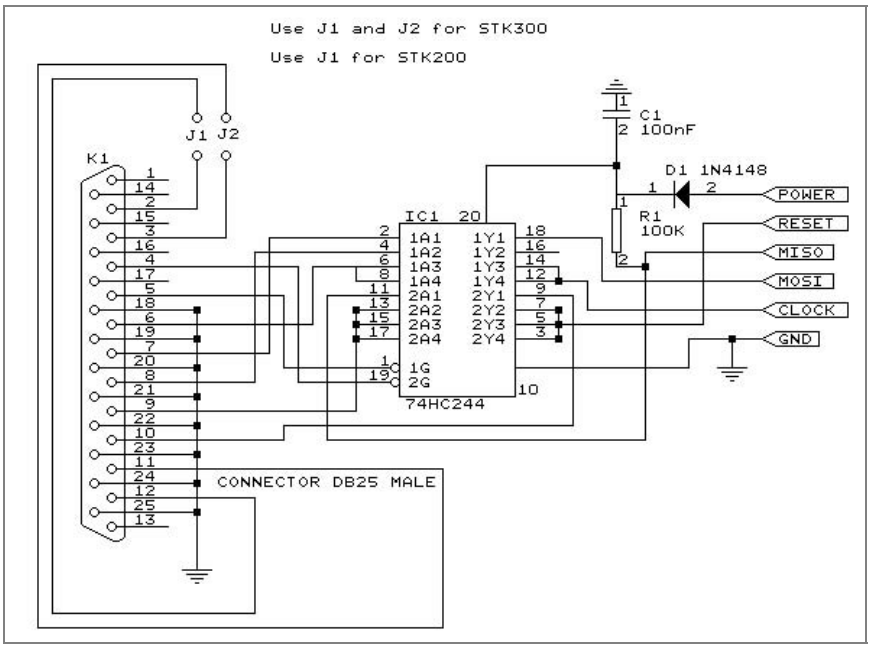


Bild 6.2: Parallel-Port ISP-Programmer. (Quelle: BASCOM-Hilfe)

Diese Version ist auch bei verschiedenen Elektronikversendern für meist unter 20 € käuflich zu erwerben.

Es gibt noch eine Sparversion eines Parallel-Port-ISP-Programmers. Sie ist aber mit Vorsicht zu genießen, da ein Kurzschluss der I/Os des Parallelports zu einem Defekt der Schnittstelle führen kann. Die 330-Ohm-Widerstände schützen den Port nur geringfügig. Es ist also immer ratsamer, die Version mit dem Puffer-IC 74HC244 aufzubauen.

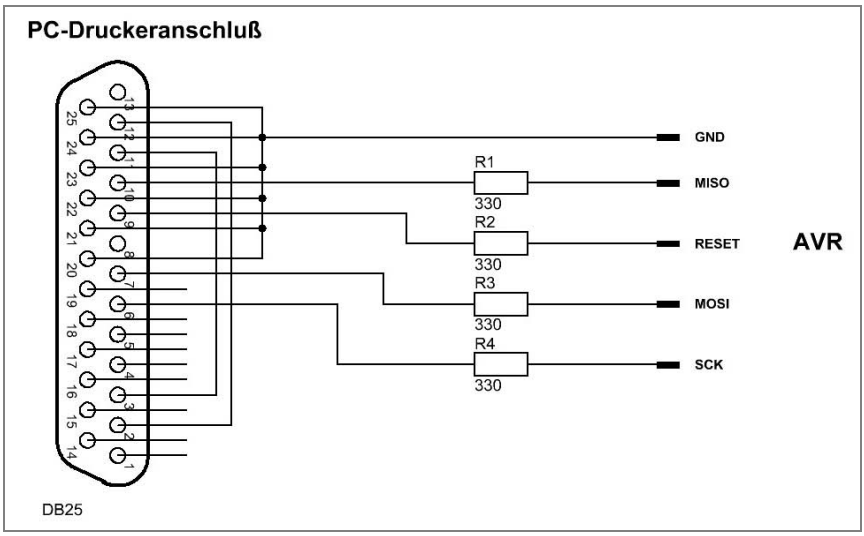


Bild 6.3: Der »Lowcost«-ISP-Programmer für die parallele Schnittstelle.

7 Interessante AVR-Boards für den Einstieg

Es gibt natürlich auch einige kostengünstige AVR-Boards zu kaufen, die den Einstieg in die AVR-Welt vereinfachen. Eine breite Palette diverser AVR-Platinen bietet die Firma Robotikhardware an. Die Module besitzen meist schon diverse Zusatzhardware wie DC-Motortreiber, Schrittmotortreiber, Funkmodule, Schaltausgänge mit Transistoren oder Relais. Programmiert werden die Boards wie die Eigenbauschaltung. Es folgt ein kleiner Überblick verschiedenster Mikrocontroller-Platinen aus dem Angebot der Firmen Robotikhardware und Atmel.

7.1 RN-CONTROL

Für den Einstieg, erste Mikrocontroller-Experimente aber auch für konkrete Projekte wie autonome Roboter, Steuerungen und vieles mehr gibt es das Board RN-CONTROL. Bei der Entwicklung wurde besonders auf ein gutes Preis-Leistungs-Verhältnis geachtet. Trotz günstigen Preises ist ein sehr flexibles Board für unzählige Anwendungsmöglichkeiten entstanden. Über den I²C-Bus stehen zahlreiche Erweiterungs-Boards zur Verfügung. So können beispielsweise die gleichen I²C-Erweiterungen kombiniert werden wie beim großen RNBFRA-Board (Relaiskarte, Sprachausgabe usw.).

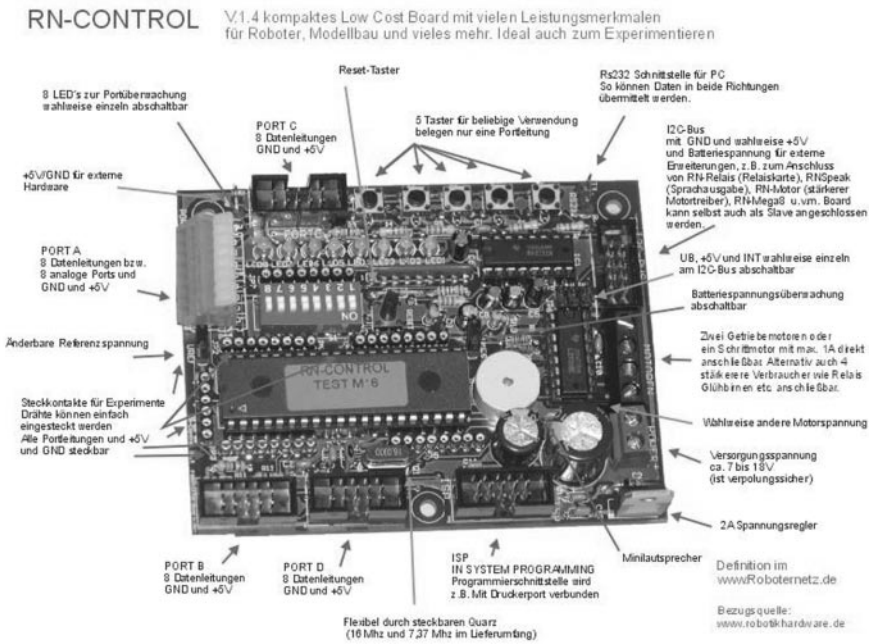


Bild 7.1: RN-Control 1.4. (Quelle: Robotikhardware)

Besonders viel Wert wurde auch auf den einfachen Aufbau und viele Experimentier- und Einsatzmöglichkeiten gelegt. Mit diesem Board lässt sich schon ein recht ausgereifter Roboter konstruieren. Ultraschallsensoren, Infrarot-Entfernungssensoren, Motoren u. v. m. können direkt angeschlossen werden. Da das Board auch in der Community Roboternetz recht beliebt ist, findet man dort viele Tipps und Programme.

7.2 RN-Mega8PLUS

Der Nachfolger des RN-Mega8-Boards, jetzt mit Funkmodulsteckplatz und weiteren Optimierungen. Dieses Nachfolge-Board wurde speziell zum Experimentieren mit den Mikrocontrollern Mega8 und Mega168 entworfen.

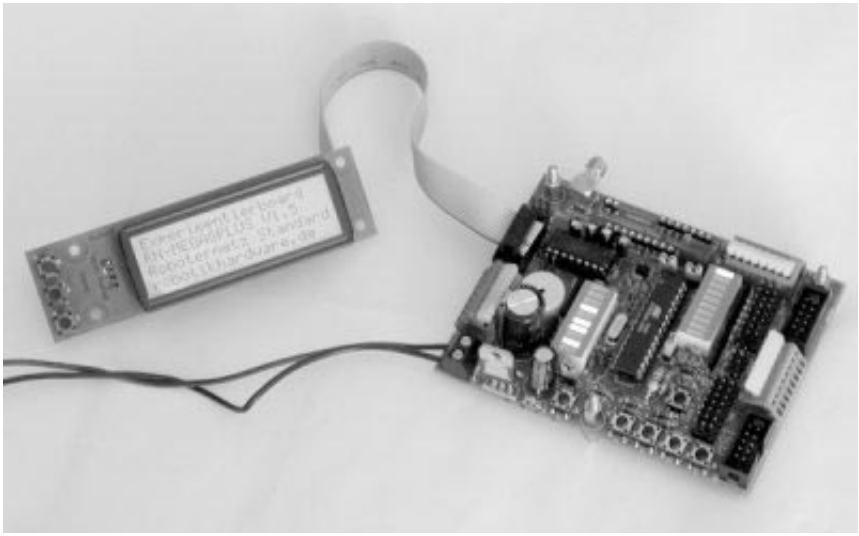


Bild 7.2: RN-Mega8PLUS. (Quelle: Robotikhardware)

Die Controller Mega8 und Mega168 eignen sich wegen ihres günstigen Preises und ihrer geringen Baugröße für zahlreiche Aufgaben, bei denen ein Mega16 oder ein Mega32 überdimensioniert wäre. Im Bereich Robotik kann dieser Controller ideal auch als Co-Controller für Servosteuerung, Motorsteuerung, Display-Ausgabe, Sensorüberwachung und vieles mehr eingesetzt werden. Oft sind kaum externe Bauteile notwendig. Um den Controller jedoch für eine spezielle Aufgabe programmieren zu können, bedarf es einer Entwicklungsumgebung die quasi alle Ports steckbar zugänglich macht, also auch die visuelle Überwachung der Port-Zustände erlaubt.

Für diese Aufgabe ist RN-Mega8PLUS ideal. Ganze 20 Ports können gleichzeitig visuell über Leuchtbalken überwacht werden. Nahezu alle Ports sind über einfache Steckklemmen erreichbar. Zudem verfügt das Board über jeweils einen genormten LCD-Display-, I²C-Bus-, RS-232-, Servo- und ISP-Anschluss.

Eine Besonderheit von RN-Mega8PLUS ist der Steckplatz für ein EasyRadio-Funkmodul. Dadurch wird das Board funkkompatibel zu RN-Mega128Funk, RN-Steuerung und RN-Funk. So können Daten mit anderen Boards oder PCs per Funk ausgetauscht werden. Das Funkmodul ist optional über *www.robotikhardware.de* beziehbar – es wird nur eingesteckt und man kann loslegen.

7.3 RN-MINICONTROL

Dieses Controllerboard zeichnet sich durch seine kompakte Größe (nur 5 cm x 7,8 cm) und sehr geringen Strombedarf aus. Besonders viel Wert wurde auch auf die vielseitigen Anschlüsse gelegt. Nahezu alle Ports stehen dem Anwender somit zur Verfügung. Besonders günstig sind die wichtigen AD-, Interrupt-, Timer- und PWM-Ports auf die Stecker verteilt worden. So lassen sich Servos, Drehgeber und RC-Empfänger, aber auch Motortreiber (Doppel-H-Brücken wie RN-VN2Dualmotor) oder LCDs direkt anschließen. Natürlich steht auch der I²C-Bus zur Verfügung. Alle Stecker sind zudem kompatibel zu den Roboternetz-Definitionen.

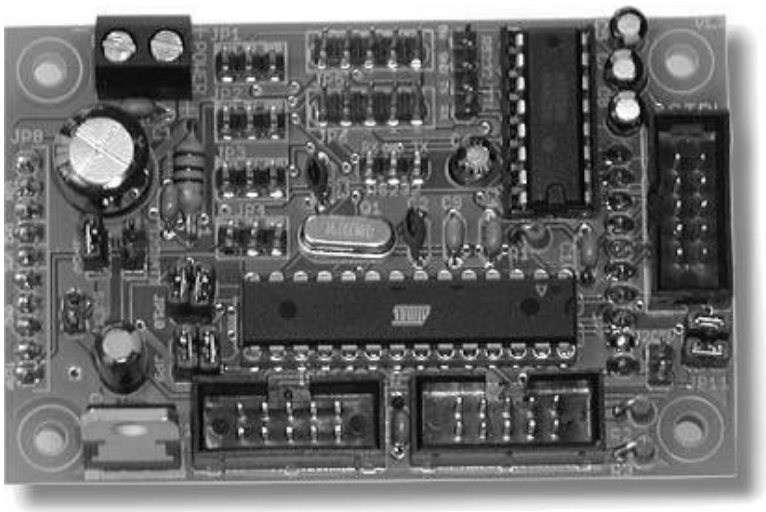


Bild 7.3: RN-MINICONTROL. (Quelle: Robotikhardware)

Obwohl das Board, wie auch der »große Bruder« RN-Control, zum Experimentieren als Haupt-Board verwendet werden kann, ist es in erster Linie als praktisch einsetzbares Zusatz-Board für echte Projekte gedacht.

RN-MINICONTROL ist als kostengünstiges Co-Controllerboard ideal als Zweit- oder Dritt-Board in Projekten (Roboter etc.). Ein besonderes Feature sind zwei Stiftleisten auf der Unterseite. Dadurch kann das Board auf ein anderes Board wie RN-VNH2Dualmotor oder Lochrasterplatinen aufgesteckt werden. Das Board ist mit dem leistungsstarken Controller ATmega168 ausgestattet. Er ist weitgehend kompatibel zum ATmega8, daher kann wahlweise auch ein ATmega8 bestückt werden. Der Mega168 bietet jedoch weit mehr Features: 6x PWM, drei Timer, 16 KB Speicher und Interrupt an jedem Pin.

12 BASCOM-AVR Basic – Programmierkurs

Wer mit BASCOM seine ersten Programmierschritte wagt, sollte diese Kapitel genauer studieren und die Beispiele ausprobieren, bis er sie versteht. Dieser Grundlagenkurs der BASCOM-Programmierung legt den Grundstein für weitere Programme. Lassen Sie nicht gleich den Kopf hängen, wenn etwas nicht auf Anhieb klappt. Auch im Profibereich funktioniert nicht immer alles beim ersten Mal. Die Freude ist umso größer, wenn ein Versuch nach mehreren Anläufen endlich klappt, und man kann dabei nur lernen.

12.1 Der Aufbau eines BASCOM-Programms

- Infotexte und Programmbeschreibung (Header)
- Prozessorangaben und Konfiguration
- Konfiguration der seriellen Schnittstelle
- Port, ADC, Timer, PWM-Konfiguration
- Variable anlegen, Alias für Ports
- Interrupts aktivieren
- Hauptschleife
- Eigene Prozeduren, Routinen und IRQ-Routinen
- Daten für LCD-Character oder Tabellen
- Include Files

12.2 Testaufbau mit MAX232

An dieser Stelle schreiben wir gleich das nächste BASCOM-Basic-Programm. Tippen Sie zur Übung den Quelltext einfach ab oder entnehmen Sie den Quellcode der CD-ROM unter *Beispiele*. Das Programm benötigt das interne Terminalprogramm von BASCOM und einen passenden Schnittstellenwandler (z. B. MAX232) oder einen USB-zu-UART-Brückenchip (z. B. FT232RL oder CP2102). Eine Beispielschaltung dazu zeigt der folgende Schaltplan.

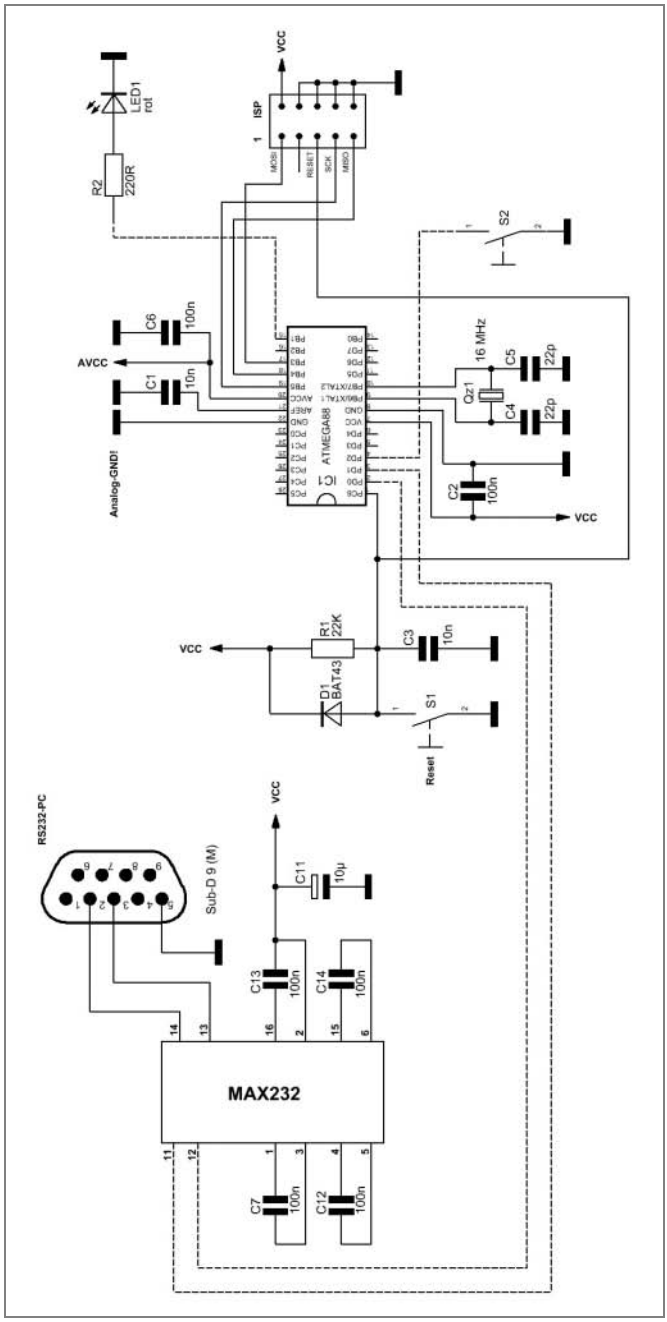


Bild 12.1: Der Schaltplan zeigt eine serielle Verbindung zum PC über einen Schnittstellenwandler MAX232.

Es gibt verschiedene MAX232-Typen. Neuere Typen benötigen nur noch kleine Kondensatoren wie hier 100 nF. Im Datenblatt zu Ihren verwendeten Schnittstellenwandlern finden Sie die passenden Werte.

Der Schnittstellenwandler setzt das RS-232-Signal vom PC (+/-12 V) in ein 0/5-V-Signal für den Mikrocontroller um.

Wichtig zu wissen ist, dass die Sender-/Empfängerleitungen gedreht sind (dem Nullmodemkabel ähnlich). Der Sender muss mit dem Empfänger (TxD -> RxD) verbunden werden.

12.3 Testaufbau mit FTDI FT232RL

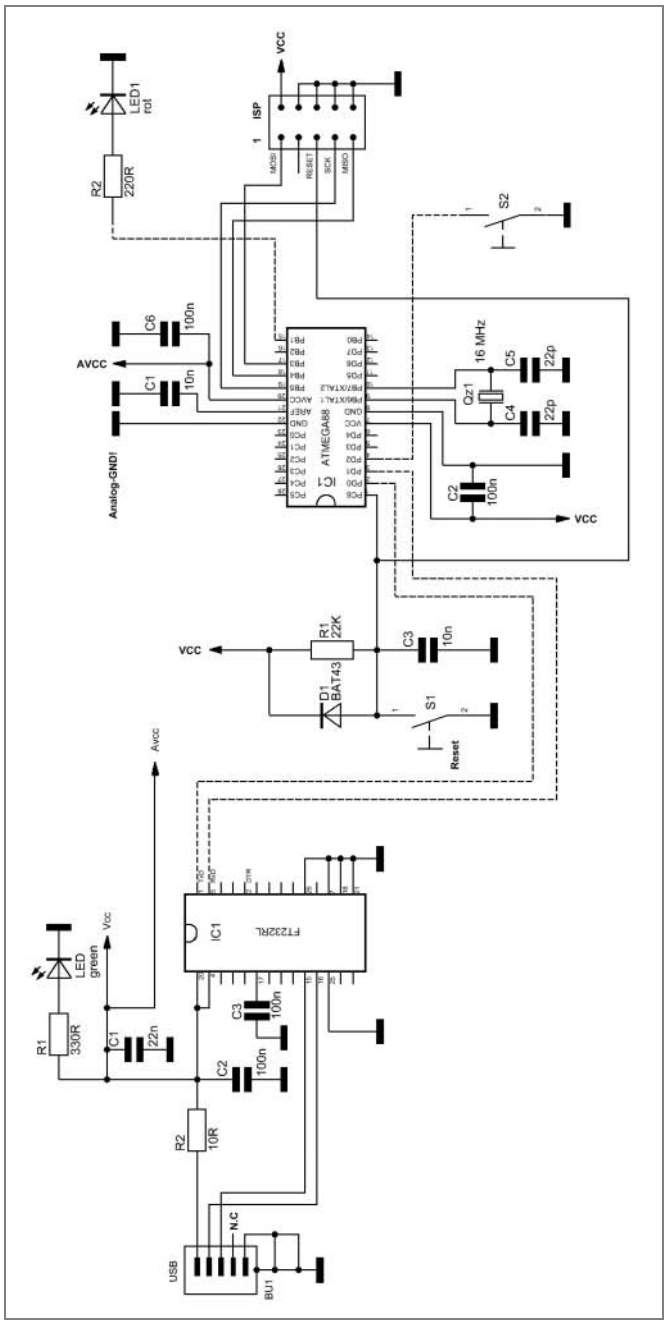


Bild 12.2: Dieser Schaltplan zeigt die Verwendung eines USB-zu-UART-Brückenchips FT232RL der Firma FTDI.

Alternativ gibt es diverse Konverter wie den Mini-USB-zu-UART-Converter der Firma Conrad zu kaufen. Diesen Konverter kann man einfach in eine Lochrasterplatine löten und mit dem Controller verbinden.

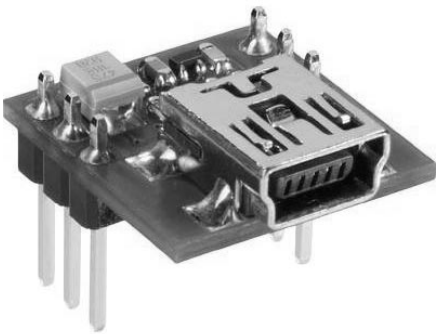


Bild 12.3: USB-zu-UART-Konverter der Firma Conrad Electronic SE; Bestellnummer: 197326. (Quelle: Conrad Electronic SE)

Mehr Informationen zur seriellen Schnittstelle (UART) folgen im Kapitel 18.

12.4 Test der seriellen Ausgabe

Nachdem Sie die Schaltung um einen Schnittstellenkonverter ergänzt haben, starten Sie nach dem Übertragen auf den Controller das Terminal unter *Werkzeuge* oder mit STRG+T und klicken zur Sicherheit noch mal auf *Reset* oder drücken Sie die Tastenkombination SHIFT+F4. Jetzt werden Sie aufgefordert, die erste Zahl (0 bis 255) einzugeben. Drücken Sie nach der Eingabe Return. Geben Sie nun die zweite Zahl ein und bestätigen Sie diese Eingabe erneut mit Return. BASCOM gibt das Ergebnis der Addition der beiden Zahlen aus. Nach der Ausgabe beginnt die LED an *Portb.1* zu blinken. Durch Drücken des Tasters S2 beginnt das Programm erneut.

Beispiel: erstes Programm.bas

```
' Das ist das erste Programm mit Bascom Basic
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Led Alias Portb.1
Config Led = Output
Led = 0

S2 Alias Pind.2
Config S2 = Input
```

```
Portd.2 = 1

Const Pi = 3.14
Dim Byte1 As Byte
Dim Byte2 As Byte
Dim Ich_bin_eine_variable As Word

Declare Function Testfunktion(byval A As Byte , Byval B As Byte)
As Word
Declare Sub Led_toggle()

' Hauptprogramm
Do

   ' Hier steht das eigentliche Programm
   ' In der Funktion wird der Wert berechnet und
   ' an "ich bin eine Variable" übergeben.

   Print "Zwei Zahlen mit Bascom addieren"
   Print

   Input "Ersten Wert eingeben: " , Byte1
   Input "Zeiten Wert eingeben: " , Byte2
   Ich_bin_eine_variable = Testfunktion(byte1 , Byte2)
   Print "Ergebnis = " ; Ich_bin_eine_variable ; Chr(10) ; Chr(13)
   Print "Taste S2 druecken um fortzufahren!" ; Chr(10) ; Chr(13)

   Do
     Led_toggle
   Loop Until S2 = 0

Loop
End

Function Testfunktion(a As Byte , B As Byte) As Word
     Testfunktion = A + B
End Function

Sub Led_toggle()
   Toggle Led
   Waitms 250
End Sub
```

Das Programm enthält bereits die wichtigsten Grundzüge eines sequenziellen Programms. Zuerst wird BASCOM mitgeteilt, welcher Controller verwendet wird.

Danach wird die Taktfrequenz auf 16 MHz festgelegt und die Geschwindigkeit der UART-Schnittstelle eingestellt.

Mit dem Befehl *Alias* wird den I/O-Pins ein Name zugeteilt (z. B. »LED«). Verwenden Sie auch in Ihren Programmen den *Alias*-Befehl. Das macht den Quellcode lesbarer.

Mit *Config <Name> = Input* oder *Output* teilt man BASCOM mit, ob wir den Pin als Aus- oder Eingang verwenden möchten. Wird der Pin als Ausgang verwendet, muss man ihn als *PORT* benennen. Möchte man den Pin als Eingang verwenden, muss man bei der Zuweisung *PIN* schreiben. Wird der I/O als Eingang benutzt, muss man (in unserem Fall) noch den internen Pull-up-Widerstand aktivieren, da die Taster gegen Masse schalten. Dies geschieht mit *Portd.2 = 1*. Hier darf man nicht *PIN* schreiben, auch wenn der Pin eigentlich ein Eingang ist.

Jetzt werden die Konstanten und Variablen angelegt und die Funktionen und Subroutinen bekannt gegeben.

Nun beginnt das eigentliche Programm. Es läuft in einer *Do-Loop*-Schleife, die nie verlassen wird. Der Befehl *Print* gibt zuerst eine Information über die UART-Schnittstelle aus, das zweite *Print* generiert eine Leerzeile. Bei den danach folgenden *Prints* ist ein *chr(10) ; chr(13)* angefügt. Das erfüllt den gleichen Zweck wie ein zusätzliches *Print*.

Der Befehl *Input* wartet nun so lange, bis eine Eingabe erfolgt ist und diese mit Return abgeschlossen wurde. Der eingegebene Wert wird in die Variable hinter dem String *Byte1* und *Byte2* abgelegt. Nun wird das Ergebnis berechnet, indem die beiden Werte von *Byte1* und *Byte2* in die Funktion mit dem Namen *Testfunktion* übergeben werden. Das Ergebnis steht nun in der Variablen mit dem Namen *Ich_bin_eine_Variable*.

Das errechnete Ergebnis wird nun mit *Print* über die serielle Schnittstelle ausgegeben. Jetzt springt das Programm in eine *Do-Loop-Until*-Schleife. In dieser Schleife wird so lange die Subroutine *Led_toggle* aufgerufen, bis der Taster S2 gedrückt wird. Erst wenn die Bedingung *S2 = 0* (Taster gegen Masse geschaltet) erfüllt ist, wird diese Schleife wieder verlassen. In der Subroutine *Led_toggle* wird mithilfe des *Toggle*-Befehls die LED zum Blinken gebracht.

12.5 Der Simulator

BASCOM besitzt einen sehr guten und vor allem einen sehr komfortablen Simulator, der es ermöglicht, die Programme auch ohne Hardware zu testen und zu simulieren.

Um in den Simulator zu gelangen, drücken Sie die Taste F2 oder gehen Sie unter *Programmieren* auf *Simulator*. Ein kleines Beispiel wird Ihnen nun den Umgang mit dem BASCOM-Simulator erklären.

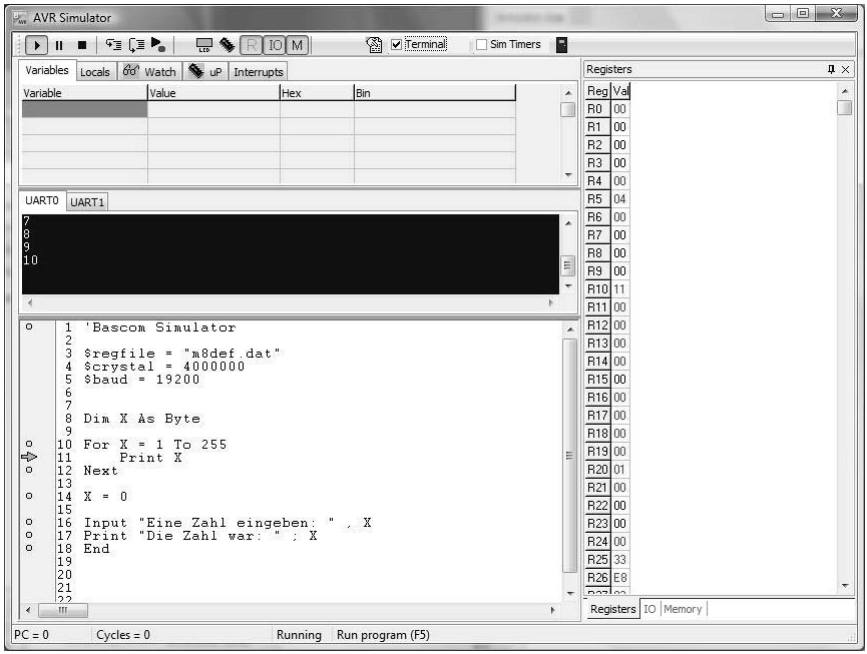


Bild 12.4: Der BASCOM-Simulator.

Beispiel: Simulator.bas

```
' Bascom Simulator
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Config Portb.1 = Output

Cls
Locate 1 , 1
Lcd "Bascom ist"
Locate 2 , 2
Lcd "super!"

Dim X As Byte

For X = 1 To 10
```

```
    Print X
Next
X = 0

Input "Eine Zahl eingeben: " , X
Print "Die Zahl war: " ; X
End
```

Wenn Sie das Programm nun kompilieren und den Simulator aufrufen, sehen Sie bereits, ob es funktioniert. Drücken Sie dazu auf *Play* oder F5, wird links im Programmcode *Fenster* ein kleiner Pfeil angezeigt, der die Stelle zeigt, an dem sich das Programm derzeit befindet.

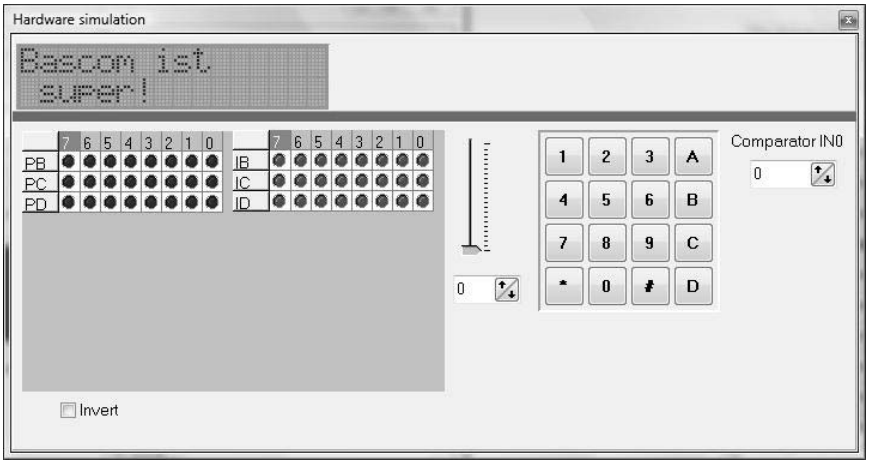


Bild 12.5: Die Hardware-Simulation.

12.6 Die Hardware-Simulation

Alle Variablen und Registerinhalte können im Simulator überwacht und simuliert werden. Spielen Sie ruhig ein wenig mit dem Simulator herum. Es kann auch die Hardware wie das LCD, ADC usw. simuliert werden. Probieren Sie hin und wieder die Beispiele auch im Simulator aus, um sich mit ihm vertraut zu machen.

Im mittleren Teil des Simulators sehen Sie ein kleines Terminal. Hier werden die Werte ausgegeben und auch die Eingaben können darüber erfolgen.

12.7 Kommentare im Quelltext

Wer sein Programm auch später noch richtig lesen und verstehen möchte, sollte seinen Quelltext exakt dokumentieren. Die Dokumentation kann man übersichtlich im Quellcode selbst erstellen. Dazu gibt es unterschiedliche Kommentarzeichen, die man dazu verwenden kann, normale Texte auszuklammern. In den Standardeinstellungen von BASCOM erscheint ein auskommentierter Text in Grün.

Beispiel: Kommentar.bas

```
' Ich bin ein einzeiliger Kommentar

REM Auch ich bin ein einzeiliger Kommentar

' (
    Ich bin ein
    mehrzeiliger Kommentar,
    der immer länger und länger wird …
' )
```

12.8 Datentypen und Variablen

Jedes Programm besteht aus verschieden Variablen – externen, die entweder von der Außenwelt wie von einem ADC oder I/O-Port stammen, oder internen, die man zur Verrechnung im Programm benötigt, um daraus wieder eine Ausgabe über ein LCD, einen Port, eine RS-232 etc. zu machen. Für die Programmierung stehen verschiedene Variablen-Typen wie Byte, Word, Integer usw. (siehe BASCOM-Hilfe) zur Verfügung. Diese müssen vor der Verwendung immer definiert werden. Das kann als Konstante oder als Variable geschehen.

Variablen-Namen

In BASCOM-Basic-Variablen-Namen wird nicht zwischen Groß- und Kleinschreibung unterschieden. Außerdem ist der Unterstrich _ zugelassen. Dieser wird oft benutzt, um lange Variablen-Namen lesbarer zu machen. Schlüsselwörter wie *If, While* usw. dürfen als Variablen-Namen nicht verwendet werden. Globale Variablen und Funktionen dürfen nicht denselben Namen besitzen. Außerdem können Funktionen innerhalb des Bezugsrahmens einer lokalen Variablen nicht benutzt werden, wenn sie denselben Namen wie die Variablen besitzen.

12.9 Lokale und globale Variablen

Wenn eine Variable innerhalb einer Funktion, einer Prozedur, oder als Argument einer Funktion deklariert wird, ist sie lokal eingebunden. Dies geschieht dann nicht mit *Dim,* sondern mit der Anweisung *Local.* Das bedeutet für uns, dass die Variable nur innerhalb dieser Funktionsdefinition existiert. Eine außerhalb einer Funktion deklarierte Variable wird als *globale Variable* bezeichnet. Sie ist für alle Funktionen innerhalb unseres Programms definiert, einschließlich der Funktionen, die in anderen Dateien definiert sind (Include Files).

12.10 Variablen-Zuweisung

Eine Variable in BASCOM kann als Hex-, Dezimal- oder Binärzahl, als String oder als String Array angegeben werden. In der folgenden Auflistung sehen Sie, welche Variablen-Typen möglich sind und wie viel Speicher dabei belegt wird.

- Bit (1/8 Byte) kann den Zustand 1 oder 0 speichern.
- Byte (1 Byte) kann Werte von 0 bis 255 aufnehmen.
- Integer (zwei Bytes) kann Werte von –32.768 bis +32.768 aufnehmen (16 Bit Integer).
- Word (zwei Bytes) kann Werte von 0 bis 65.535 annehmen.
- Long (4 Bytes) kann Werte von –2.147.483.648 bis 2.147.483.647 annehmen (32 Bit Long).
- Single-Variablen können Werte mit 32 Bit und Vorzeichen speichern. Der Bereich liegt zwischen $1{,}5 \times 10^{-45}$ und $3{,}4 \times 10^{38}$.
- Double können Werte mit 64 Bit speichern. Der Bereich liegt zwischen $5{,}0 \times 10^{-324}$ und $1{,}7 \times 10^{308}$
- String(bis zu 254 Bytes)-Variablen speichern Bytes mit einer 0 Terminierung ab. Wenn man 10 Zeichen in einen String schreiben möchte, muss man das String-Array mit einer Länge von 11 spezifizieren.

Beispiel: Variablen Zuweisung.bas

```
Const PI = 3.14       ' Konstante PI

Dim Variable As Byte  ' Byte-Variable, Sie kann Werte von 0 bis
                      ' 255 annehmen

Dim Var(9) As Byte    ' Byte Array, hat eine ähnliche Bedeutung
                      ' wie 10x ein Byte
```

```
                ' mit Dim Var X  As Byte zu erstellen.
                ' Das jeweilige Byte
                ' wird über den Index angesprochen: Var( x)
                ' Die Arrays der Pro werden von 0 aus gezählt!
```

12.11 Arrays

Ein Array (Feld) ist eine Anordnung von Variablen. Array bezeichnet in der Informatik eine Datenstruktur. Mithilfe eines Arrays können Daten eines üblicherweise einheitlichen Datentyps (Byte, Word usw.) geordnet so im Speicher eines Computers abgelegt werden, dass ein Zugriff auf die Daten über einen Index möglich wird. Der Index für ein Array startet bei BASCOM mit 1, bei anderen Compilern beginnt der Index mit 0. Zudem unterstützt BASCOM derzeit nur eindimensionale Arrays.

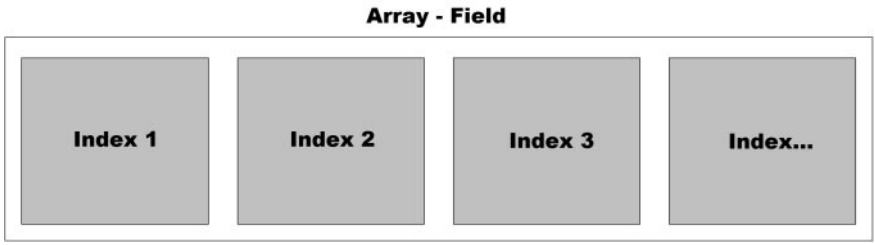


Bild 12.6: So kann man sich die Anordnung der Variablen im Speicher vorstellen.

Beispiel: Arrays.bas

```
' Bascom Arrays
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim X(10) As Byte
Dim Y As Byte

For Y = 1 To 10
   X(y) = Y
Next

For Y = 1 To 10
   Print X(y)
Next
End
```

Das obige Beispiel verdeutlicht die Verwendung von Arrays. Erst werden dem Array über den Index die Werte der *For-Next*-Zählschleife zugewiesen (1 bis 10), danach werden die gespeicherten Werte ausgegeben.

12.12 Operatoren

Jeder Datentyp verfügt über einige spezifische Operatoren, die angeben, welche Operationen auf den Typ angewendet werden können. Die folgende Auflistung zeigt eine Auswahl der BASCOM-Operatoren.

Arithmetik:	Grundrechenarten
Vergleich:	größer/kleiner usw.
Bitweise Arithmetik:	And, Or, Xor, usw.
Boolesche Arithmetik:	logische And-, Or-, Xor-Verknüpfung, usw., (wahr/falsch)
Ganze Zahlen:	Char, Byte, Word, Integer, Long
Fließkommazahlen:	Single, Double
Arithmetische Funktionen:	Sin, Cos, Tan, usw.

Sehen Sie dazu auch in der Hilfe von BASCOM nach.

12.13 Kontrollstrukturen

Jedes Programm benötigt, um auf Ereignisse reagieren zu können, Bedingungen, sogenannte *Kontrollstrukturen*. Diese werden mit *If … Then … Else … End If* angegeben. Die Ausgaben der Beispielprogramme erfolgen über das BASCOM-Terminalprogramm. Laden Sie die Beispiele auf das Mikrocontroller-Board und starten Sie danach das Terminalprogramm.

12.13.1 If Then – End if

```
If [Variable A] = [Variable B] Then
    ' Hier steht der Code, der bei der Bedingung A=B ausgeführt
    ' wird
End If
```

Beispiel: If Then.bas

```
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim X As Byte

Main:

   X = X + 1
   If X = 10 Then
      Print "X ist jetzt 10!"
      Goto Ende
   End If
   Goto Main

Ende:
Print "fertig!"
End
```

Der Programmcode durchläuft so lange die *Goto-Main*-Schleife, bis X gleich 10 ist. Erst dann wird der Programmteil zwischen *If* und *End If* ausgeführt. Sie sehen: Mit *If Then* kann man einfach Programmverzweigungen realisieren.

12.13.2 If Then – Else – End if

```
If [Variable A] > [Variable B] Then
   ' Code, der ausgeführt werden soll
   Else    ' Oder wenn A nicht größer B
   ' Code, der ausgeführt werden soll
End If
```

Beispiel: Else.bas

```
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim X As Byte

Main:

   X = X + 1
   If X <= 10 Then
      Print X
      Goto Main
```

```
   Else
      Print "X ist bereits 10, Juhu!"
   End If

End
```

Mit der *Else*-Anweisung kann man eine Alternative anbieten. Das Programm gibt so lange den Wert von X aus, bis X den Wert 10 erreicht hat. Erst wenn X größer/gleich 10 ist, wird der Programmteil zwischen *Else* und *End if* ausgeführt.

12.13.3 If und ElseIf

Eine weitere Möglichkeit der mehrfachen Verschachtelung von *If*-Anweisungen ist *ElseIf*. Hier können unterschiedliche Zustände der Variablen abgefragt werden. Je nach Wahrheit (true oder false) wird der entsprechende Abschnitt in der *ElseIf*-Anweisung ausgeführt.

```
If [Variable A] <> [Variable B] Then
   ' Code, der ausgeführt werden soll
   ElseIf [Variable A] > [Variable B] Then
   ' Code, der ausgeführt werden soll
   ElseIf [Variable A] < [Variable B] Then
   ' Code, der ausgeführt werden soll
End If
```

Beispiel: ElseIf.bas

```
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim X As Byte

Main:

   X = X + 1
   If X = 1 Then
      Print "X = 1"
   Elseif X = 2 Then
      Print "X = 2"
   Elseif X = 3 Then
      Print "X = 3"
   Else
      Print "X = 4, Fertig!"
      Goto Ende
   End If
```

```
   Goto Main
Ende:
End
```

Je nach Wert der Variablen X ist der ausgegebene Text anders.

12.13.4 Select Case

Ähnlich der *ElseIf*-Anweisung verhält sich der Befehl *Select Case*. Auch hier wird, je nachdem, welcher Wert true ist (also der Wahrheit entspricht), der entsprechende Codeabschnitt ausgeführt. Alternativ kann mit *Case Else* eine Alternative angeboten werden, sollte nichts innerhalb von *Case* zutreffen.

```
Select Case [Variable]

   Case 1
   ' Code, der ausgeführt werden soll, Variable = 1

   Case Is > 10
   ' Code, der ausgeführt werden soll, Variable >10

   Case Else

   ' Alternativ-Code, wenn alle anderen Bedingungen nicht
   ' zutreffen

End Select
```

Beispiel: Select Case.bas

```
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim Zahl As Byte

Main:

   Input "Geben Sie eine Zahl ein: " , Zahl

   Select Case Zahl

      Case 1
      Print "Zahl ist 1"

      Case 2 To 10
      Print "Die Zahl liegt zwischen 2 und 10"

      Case Is > 10
```

```
      Print "Die Zahl ist groesser 10"

   Case Else
      Print "Die Zahl ist Null"

   End Select
Goto Main
```

Geben Sie in diesem Programm eine Zahl zwischen 0 und 255 ein. Über *Select Case* wird Ihnen BASCOM eine Antwort darauf geben, in welchem Bereich sich die Zahl befindet. Der zusätzliche Befehl *Input* dient zur Eingabe über das Terminalprogramm. Er gibt den Textstring *Geben Sie eine Zahl aus:* aus und übergibt Ihre Eingabe der Variablen mit dem Namen *Zahl*.

12.14 Schleifen

Bei der Programmierung werden häufig Programmschleifen benötigt, z. B. um Dezimal- oder Binärzähler oder eine Hauptschleife zu realisieren. Es wird vielleicht auch die serielle Schnittstelle so lange ausgelesen, wie Zeichen im Puffer sind – um nur einige zu nennen. Es gibt dafür verschieden Schleifentypen. Jede von ihnen hat ihre Eigenheiten.

12.14.1 For Next

Die *For-Next*-Schleife zählt innerhalb eines angegeben Wertebereichs die Variable X hoch oder herunter. Dabei können bestimmte Schrittweiten (Steps) vorgegeben werden.

```
' Diese Schleife zählt von Var1 bis Var2 mit einer Schrittweite
' von 1
For X = [Var1] To [Var2]
   'Code, der 10-mal durchlaufen werden soll
Next

   ' Die Variable X wird jetzt immer um 2 erhöht
For X = [Var1] To [Var2] Step 2
   ' Code, der durchlaufen werden soll
Next

   ' Variable X wird von Var1 auf Var2 heruntergezählt
   ' (Schrittweite 1)
For X = Var1 To Var2 Step -1
   ' Code, der durchlaufen werden soll
Next
```

Beispiel: For Next.bas

```
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim Zahl As Byte
Dim X As Byte

Input "Wie weit soll ich zaehlen? " , Zahl
Print "Ich zaehle bis " ; Zahl
For X = 1 To Zahl
   Print X
Next

Input "Wie weit soll ich zaehlen? " , Zahl
Print "Ich zaehle bis " ; Zahl ; " Schrittweite = 2"
For X = 1 To Zahl Step 2
   Print X
Next

Input "Mit welcher Zahl soll ich beginnen? " , Zahl
Print "Ich beginne mit " ; Zahl
For X = Zahl To 1 Step -1
   Print X
Next

Print "fertig!"
End
```

Übertragen Sie das Beispiel auf den Controller und starten Sie wieder das Terminalprogramm. Das Programm wird Sie nach einer Zahl fragen. Geben Sie einen Wert zwischen 1 und 255 ein. Die Variable X wird bei jedem Durchlauf ausgegeben und zeigt Ihnen, was bei der *For-Next-Schleife* genau passiert. Sie werden sehen, dass wir bei einer *For-Next-Schleife* eine definierte Anzahl von n Durchläufen mit einer gewünschten Schrittweite realisieren können. Mit dem Befehl *Exit For* können Sie die Schleife auch vorzeitig verlassen.

12.14.2 Do Loop und Do Until

Weitere Varianten einer Schleife sind die *Do-Loop-* und die *Do-Loop-Until*-Version. Die *Do-Loop-Until*-Version wird einmal durchlaufen und prüft erst nach einem Durchlauf, ob die Bedingung erfüllt ist. D h., diese Schleife wird mindestens einmal durchlaufen. Die reine *Do-Loop*-Schleife wird gern für Endlosschleifen eingesetzt. Auch die *Do-Loop*-Schleife kann mit *Exit-Do*-Anweisung abgebrochen werden.

```
' Endlosschleife
Do
      ' Was auch immer wir hier tun wollen
Loop

' Endlosschleife mit bedingtem Abbruch
Do
      Variable = Variable + 1
      If Variable > 10 Then Exit Do
Loop

' Do Loop Until
Do
   Variable = Variable + 1
Loop Until Variable > X
```

Beispiel: Do Loop.bas

```
' Bascom Do Loop
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim X As Byte
X = 0

Do
   X = X + 1
   Print X
   If X > 9 Then Exit Do
Loop

X = 0

Do
   X = X + 1
   Print X
Loop Until X > 9

Print "fertig!"
End
```

Endlosschleifen werden z. B. für das Hauptprogramm verwendet, da sonst das Programm nach einem Durchlauf beendet wäre.

12.14.3 While Wend

Eine Schleifenvariante, die nur durchlaufen wird, wenn die Bedingung erfüllt ist (true), ist die *While-Wend*-Schleife. Sie prüft zu Beginn, ob die Bedingung erfüllt ist.

Beispiel: While Wend.bas

```
' Bascom While Wend
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim X As Byte
X = 0

While X < 10
   X = X + 1
   Print X
   If X > 5 Then Exit While
Wend

While X < 3
   X = X + 1
   Print X
Wend

Print "fertig!"
End
```

Hier wird nur die erste *While-Wend*-Schleife durchlaufen. Die zweite Schleife wird nicht durchlaufen, weil X bereits größer 3 ist. Auch dieser Schleifentyp kann vorzeitig mit einer *Exit*-Anweisung beendet werden. In unserem Beispiel dann, wenn X größer 5 ist.

12.15 Funktionen, Prozeduren und Labels

Funktionen benötigt man immer wieder. Sie machen das Programm wesentlich übersichtlicher, und auch eigene Befehle können damit verwirklicht werden. Man kann eigene Funktionen und Prozeduren (Unterprogramme ohne Rückgabewert), die man immer wieder benötigt, selbst schreiben und in kommenden Projekten wiederverwenden (Modularität). Der Unterschied zwischen *Funktion* und *Prozedur,* oder allgemein *Routine* (Sub Routine) genannt, ist, dass eine Funktion im Gegensatz zu einer Prozedur einen Rückgabewert besitzt. In einer Funktion kann man z. B. eine mathematische Berechnung durchführen, die das Ergebnis an eine Variable zurückgibt.

```
' Am Programmanfang muss die Prozedur bekanntgegeben werden.
Declare Sub Unterroutine_xyz(Variablen, die Sie übergeben
möchten)

' Die eigentliche Prozedur
Sub Unterroutine_xyz(Variablen, die Sie übergeben möchten)
      ' Hier könnten wir z. B. einen Port setzen
End Sub
```

Der Aufruf in BASCOM geschieht nur durch die Eingabe des Prozedurnamens, in unserem Fall *Unterroutine_xyz* oder, wenn man Variablen übergeben möchte: *Unterroutine(Variablen die übergeben werden sollen)*.

Achtung: Hier muss man auf den HW Stack, Soft Stack und die Frame Size achten!
Werden diese zu klein ausgelegt, können Funktionen und Routinen nicht richtig behandelt werden.
In der BASCOM-Hilfe findet man mehr darüber.

Original Konfiguration

```
$hwstack = 32                                                    '
default use 32 for the hardware stack
$swstack = 10
'default use 10 for the SW stack
$framesize = 40
'default use 40 for the frame space

besser…

$hwstack = 64
$swstack = 20
$framesize = 80
```

12.15.1 Subroutinen

Beispiel: Sub.bas

```
' Bascom Unterprogramme Subs
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim X As Byte
```

```
Declare Sub Ausgabe_1()
Declare Sub Ausgabe_2()

Ausgabe_1
Ausgabe_2
End

Sub Ausgabe_1()
    Print "Ausagbe 1"
End Sub

Sub Ausgabe_2()
    Print "Ausgabe 2"
End Sub
```

12.15.2 Funktionen

```
' Am Programmanfang muss die Funktion bekanntgegeben werden.
Declare Function Funktion_xyz(Variablen die übergeben werden)
Rückgabe Variable

' Die eigentliche Funktion
Function Funktion_xyz(Variablen die übergeben werden) Rückgabe
Variable
      ' Hier könnten wir z. B. eine Berechnung durchführen
End Function
```

Beispiel: Function.bas

```
‘ Bascom Funktionen
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim X As Word

Declare Function Addiere(byval A As Byte , Byval B As Byte) As
Word

X = Addiere(10 , 13)
Print "Das Ergebnis = " ; X
End

Function Addiere(byval A As Byte , Byval B As Byte) As Word
   Addiere = A + B
End Function
```

Innerhalb der Klammer stehen die Übergabeparameter, nach der Klammer wird definiert, um welchen Variablen-Typ die Rückgabe erfolgen soll.

Je nachdem, wie eine Variable in einer Prozedur oder Funktion übergeben werden soll, muss die Zuweisung als *Byval* oder *Byref* erfolgen. Dazu finden Sie noch mehr Informationen in der Hilfe zu BASCOM.

12.15.3 Gosub

Eine weitere Möglichkeit ist es, die Unterprogramme ohne Übergabe- oder Rückgabewert über *Gosub* anzuspringen. Hier muss die Prozedur nicht zuvor am Anfang des Programms bekannt gegeben werden.

```
' Der Aufruf der Routine
Gosub Test_Routine

' Die Routine
Test_Routine:
   ' auszuführender Programmcode
Return
```

Beispiel: Gosub.bas

```
' Bascom Gosub
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim X As Byte

Gosub Zaehlen
End

Zaehlen:
   For X = 1 To 10
      Print X
   Next
Return
```

12.15.4 Goto

Eine weitere praktische aber mittlerweile veraltete Anweisung ist *Goto* (= gehe zur Markierung xxx). Dieser Befehl stammt aus Zeiten, in denen Basic-Programme noch mit Zeilennummern und Sprüngen geschrieben wurden. Es gibt zwar in unserem Basic noch die Möglichkeit, Zeilennummern anzuzeigen, aber sie dienen nur noch zur Orientierung und haben mit dem Programmablauf

nichts zu tun. Doch kann es hin und wieder sinnvoll sein, den *Goto*-Befehl zu verwenden. Er benötigt nur sehr wenig Ausführungszeit, da er in Assembler direkt ausgeführt werden kann. In den obigen Beispielen haben wir den Befehl *Goto* immer wieder für Endlosschleifen und zum Beenden des Programms verwendet. Wenn Sie im Besitz eines Oszilloskops sind, können Sie mit dem folgenden Beispiel sehen, wie schnell der BASCOM und der kleine ATmega88 mit dem 16-MHz-Oszillator ist. Über eine *Goto*-Schleife wird der *Portb.1* ein- und ausgeschaltet. Dabei wird eine Frequenz von knapp 3 MHz erreicht, was ein sehr guter Wert für ein Basic-Programm ist.

```
Start:      ' Label anstatt Zeilennummer
            ' Code, der ausgeführt werden soll
Goto Start  ' Sprung zum Label Start
```

Beispiel: Goto.bas

```
' Bascom Goto
$regfile = "m88def.dat"
$crystal = 16000000

Config Portb.1 = Output
Portb.1 = 0

Main:
   Portb.1 = 1
   Portb.1 = 0
Goto Main
```

TIPP: *Goto* kann man sehr gut zum Abbrechen, für einen Programmneustart oder zur Fehlerbehandlung nutzen.

12.15.5 On

Eine weitere Verzweigung zu den bereits aufgeführten Varianten ist die *On*-Verzweigung. Je nachdem, welchen Wert die Variable angenommen hat, wird das dementsprechende Label angesprungen. Der Index der Variablen beginnt mit 0.

```
' Wenn eine Variable dem Wert entspricht (true), springe zu Label x.
On [Var] Goto [Label 0], [Label 2],... [Label n]
End if
```

Beispiel: On.bas

```
' Bascom On
$regfile = "m88def.dat"
$crystal = 16000000
```

```
$baud = 19200

Dim X As Byte

Do

   Input "Welches Label (0 bis 2): " , X
   On X Gosub Label0 , Label1 , Label2

Loop
End

Label0:
   Print "Label 0"
Return
Label1:
   Print "Label 1"
Return

Label2:
   Print "Label 2"
Return
```

12.16 String und String-Bearbeitung

Unter *Strings* versteht man eine Zeichenkette mit einem Character als Variableninhalt. In einem String können ASCII-Zeichen oder ganze ASCII-Zeichenketten abgespeichert werden. Kurz gesagt: Strings sind Character-Arrays.

12.16.1 Strings

```
' String mit der Länge 1, er kann nur 1 Zeichen enthalten
Dim Text as string  * 1

' String mit der Länge 10, er kann 10 Zeichen enthalten
Dim Text as string * 10
```

Beispiel: Strings.bas

```
' Bascom Strings
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim Text1 As String * 1
```

```
Dim Text2 As String * 10

Text1 = "A"
Text2 = "1234567890"

Print Text1
Print Text2
End
```

12.16.2 Ucase

Mit *Ucase* konvertiert man einen String von Klein- in Großbuchstaben.

Beispiel: Ucase.bas

```
' Bascom Strings (Ucase)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim A As String * 10
Dim B As String * 10

A = "franzis"
B = Ucase(a)
Print B
End
```

12.16.3 Lcase

Mit *Lcase* konvertiert man einen String von Groß- in Kleinbuchstaben.

Beispiel: Lcase.bas

```
' Bascom Strings (Lcase)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim A As String * 10
Dim B As String * 10

A = "FRANZIS"
B = Lcase(a)
Print B
End
```

12.16.4 Bin

Mit *Bin* wandelt man ein Byte in einen binären String um.

Beispiel: Bin.bas

```
' Bascom Strings (Bin)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim A As String * 10
Dim B As Byte

B = 42
A = Bin(b)
Print A
End
```

12.16.5 Hex

Mit *Hex* konvertiert man ein Byte in einen Hex-String.

Beispiel: Hex.bas

```
' Bascom Strings (Hex)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim A As Integer
Dim B As String * 4

A = 1023
B = Hex(a)
Print B
End
```

12.16.6 Hexval

Mit *Hexval* wandelt man einen Hex-String in eine Zahl um.

Beispiel: Hexval.bas

```
' Bascom Strings (Hexval)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200
```

```
Dim A As Integer
Dim B As String * 2

B = "42"
A = Hexval(b)
Print A
End
```

12.16.7 Val

Mit *Val* wandelt man eine String-Zahl in eine Zahl um.

Beispiel: Val.bas

```
' Bascom Strings (Val)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim A As String * 10
Dim B As Byte

A = "42"
B = Val(a)
Print B
End
```

12.16.8 Str

Mit *Str* konvertiert man eine Zahl in einen String (also das Gegenteil von *Val*).

Beispiel: Str.bas

```
' Bascom Strings (Str)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim A As String * 10
Dim B As Word

B = 65535
A = Str(b)
Print A
End
```

12.16.9 String

Mit *String* erzeugt man eine bestimmte Anzahl von einem ASCII-Zeichen.

Beispiel: StringConv.bas

```
' Bascom Strings (String)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim A As String * 10

A = String(3 , 65)
Print A
End
```

12.16.10 Space

Mit *Space* erzeugt man eine bestimmte Anzahl von Leerzeichen.

Beispiel: Space.bas

```
' Bascom Strings (Space)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim A As String * 10
Dim B As Byte

B = 10
A = Space(b)
Print "<" ; A ; ">"
End
```

12.16.11 Fusing

Mit *Fusing* wird eine Single-Variable in einen String gewandelt. Diese Funktion rundet das Ergebnis dabei gegebenenfalls.

Beispiel: Fusing.bas

```
' Bascom Strings (Fusing)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200
```

```
Dim A As Single
Dim B As String * 10

A = 128.1234567
B = Fusing(a , "#.###")
Print B
End
```

12.16.12 Format

Mit *Format* formatiert man den Ursprungsstring. Dabei kann auch ein mathematisches Vorzeichen (+ oder -) mit angegeben werden.

Beispiel: Format.bas

```
' Bascom Strings (Format)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim A As String * 6
Dim B As String * 6

A = "12345"
B = Format(a , "+")
Print B

A = "123"
B = Format(a , "00000")
Print B

A = "12345"
B = Format(a , "000.00")
Print B

A = "12345"
B = Format(a , " +000.00")
Print B
End
```

12.16.13 Len

Mit *Len* bestimmt man die Länge eines Strings.

Beispiel: Len.bas

```
' Bascom Strings (Len)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim A As String * 10
Dim B As Byte

A = "Franzis"
B = Len(a)
Print B
End
```

12.16.14 Instr

Mit *Instr* sucht man in einem String eine bestimmte Reihenfolge von Zeichen und gibt deren Position zurück. Es wird immer die zuerst gefundene Zeichenkette zurückgegeben.

Beispiel: Instr.bas

```
' Bascom Strings (Instr)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim A As String * 14
Dim B As String * 2
Dim C As Integer

A = "www.franzis.de"
B = "de"
C = Instr(a , B)
Print C
End
```

12.16.15 Mid

Mit *Mid* ermittelt man den mittleren Teil eines Strings.

Beispiel: Mid.bas

```
' Bascom Strings (Mid)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim A As String * 20
Dim B As String * 10
Dim C As Byte
Dim D As Byte

A = "Franzis ist super!"
C = 9
D = 3
B = Mid(a , C , D)
Print B
End
```

12.16.16 Split

Mithilfe von *Split* teilt man einen String in einzelne kleine Blöcke auf. So kann man z. B. die einzelnen Wörter eines Satzes herausfiltern.

Beispiel: Split.bas

```
' Bascom Strings (Split)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim A(5) As String * 10
Dim B As Byte
Dim C As Byte

B = Split( "Franzis ist super!" , A(1) , " ")

For C = 1 To B
   Print A(c)
Next
End
```

12.16.17 Left

Mit *Left* schneidet man einen bestimmten Ausschnitt des Strings von links aus heraus.

Beispiel: Left.bas

```
' Bascom Strings (Left)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim A As String * 10
Dim B As String * 3
Dim C As Integer

A = "Franzis"
C = 5
B = Left(a , C)
Print B
End
```

12.16.18 Right

Mit *Right* schneidet man einen bestimmten Ausschnitt des Strings von rechts aus heraus.

Beispiel: Right.bas

```
' Bascom Strings (Right)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim A As String * 10
Dim B As String * 3
Dim C As Integer

A = "Franzis"
C = 3
B = Right(a , C)
Print B
End
```

12.16.19 Ltrim

Mit *Ltrim* entfernt man die Leerzeichen von der linken Seite eines Strings.

Beispiel: Ltrim.bas

```
' Bascom Strings (Ltrim)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim A As String * 10
Dim B As String * 10

A = "   Franzis"
B = Ltrim(a)
Print B
End
```

12.16.20 Rtrim

Mit *Rtrim* entfernt man die Leerzeichen von der rechten Seite eines Strings.

Beispiel: Rtrim.bas

```
' Bascom Strings (Rtrim)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200

Dim A As String * 10
Dim B As String * 10

A = "Franzis   "
B = Rtrim(a)
Print B
End
```

12.16.21 Trim

Mit *Trim* entfernt man die Leerzeichen (Space) auf beiden Seiten eines Strings.

Beispiel: Trim.bas

```
' Bascom Strings (Trim)
$regfile = "m88def.dat"
$crystal = 16000000
$baud = 19200
```

```
Dim A As String * 15
Dim B As String * 15

A = "   Franzis "
B = Trim(a)
Print B
End
```

Ulli Sommer

Mikrocontroller programmieren mit Bascom Basic

Mikrocontroller steuern, messen und regeln die unterschiedlichsten Geräte, und jeder Controller ist für seine spezielle Aufgabe programmiert.

Wie Sie einen Mikrocontroller programmieren und was Sie alles benötigen, um Ihre eigenen Spezial-ICs zu erstellen, erfahren Sie in diesem Buch. Es zeigt Ihnen Schritt für Schritt auf verständliche Weise, was Sie für den Einstieg in die ATMEL-AVR-Mikrocontrollerwelt benötigen. Sie erfahren, wie Sie bei der Hardware vorgehen müssen und wie man das Programmieren erlernt, um später seine eigene Software zu entwickeln.

Dieses Buch baut auf dem Basic-Compiler Bascom und den ATMEL-AVRs auf. Bascom ist ein moderner Basic-Compiler mit integrierter Entwicklungsumgebung und eignet sich für fast alle 8-Bit-AVR- und XMega-Mikrocontroller der Firma ATMEL.

Viele Problemstellungen, die früher zeitaufwendig in Assembler oder C gelöst werden mussten, können durch diesen modernen Compiler blitzschnell mit wenigen Befehlen erledigt werden, und sollte dies einmal nicht ausreichen, so stellt Bascom noch die Möglichkeit des Inline-Assemblers zur Verfügung.

Die ersten Kapitel des Buches vermitteln Ihnen die Programmierung mit Bascom in einem ausführlichen Programmierlehrgang. Hier werden die Befehle anhand kleiner Beispiele verdeutlicht. Hard- und Software werden detailliert erklärt, und am Ende macht Ihnen keiner mehr ein Byte für ein Bit vor. Das erlernte Wissen aus dem Programmierkurs wird in den darauf folgenden Experimenten kreativ und spielerisch in Mess-, Steuer- und Regelanwendungen eingesetzt. Das Buch wird Ihnen auch danach als Referenz und Nachschlagewerk nützliche Dienste erweisen.

Aus dem Inhalt:

- AVR: Grundlagen und Programmierung
- Ausführlicher Bascom-Programmier-Einsteigerkurs
- Schaltplan und Erklärung zu jedem Experiment
- Sensoren und Aktoren anschließen und anwenden
- Von den Grundlagen bis zur eigenen Applikation
- Über 100 praktische Experimente, z. B. Ein-/Ausschaltverzögerung, Temperaturschalter, Kapazitätsmessgerät, GPS-Maus auslesen, Digitalvoltmeter mit AVR und VB.NET, Rasenmäh-Roboter, Steuern mit VB.NET, Ultraschallsensoren, Drehzahlmesser, Transistor-LED-Dimmer, Lüftersteuerung, Digitales Codeschloss, Daten aufzeichnen mit Stamp PLOT, Digitaler Datenplotter mit VB.NET selbst gemacht, Bewegungsmelder, Alarmanlage, Dämmerungsschalter, Morsezeichen-Encoder, Morsezeichen-Decoder, Modellflug: Drohnen-Telemetriesystem u. v. m.

ISBN 978-3-645-65041-0

39,95 EUR [D]