AVR-Mikrocontroller in BASCOM programmieren, Teil 1

Dies ist ein sehr knapp gehaltenes Tutorial. Man sollte unbedingt zusätzlich die BASCOM-Hilfe zu Rate ziehen.

Empfehlenswerte Bücher:

Roland Walter
AVR-Mikrocontroller-Lehrbuch
http://www.rowalt.de/mc/index.htm

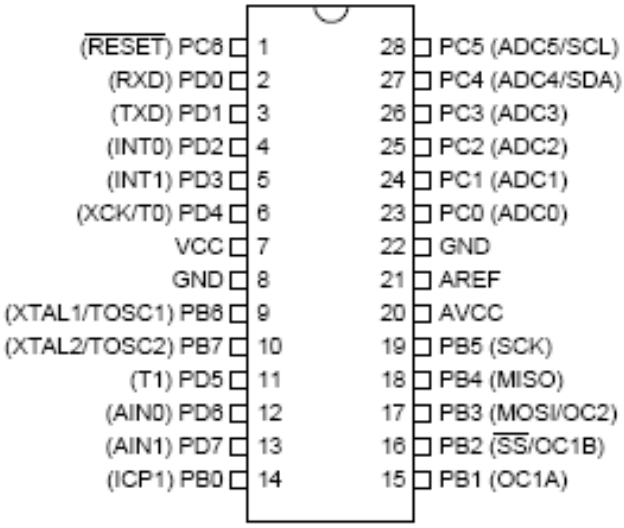
Claus Kühnel
Programmieren der AVR RISC Mikrocontroller
http://www.ckuehnel.ch/

Ulli Sommer
Roboter selbst bauen
Franzis Verlag
(ist eigentlich ein Buch über Roboter, lehrt aber auch BASCOM)

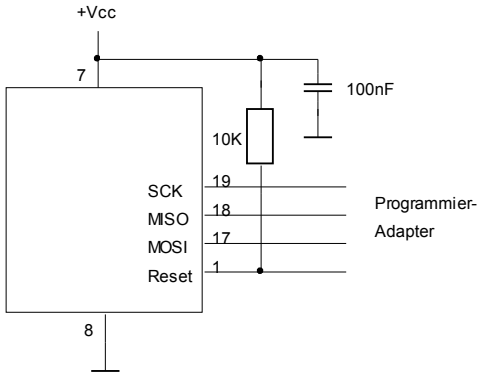
1. Was braucht man um anzufangen?

1.1 Hardware

Gut geeignet ist der **AtMega8**, da er überschaubar und klein ist, aber trotzdem sehr viele integrierte Peripherie hat.



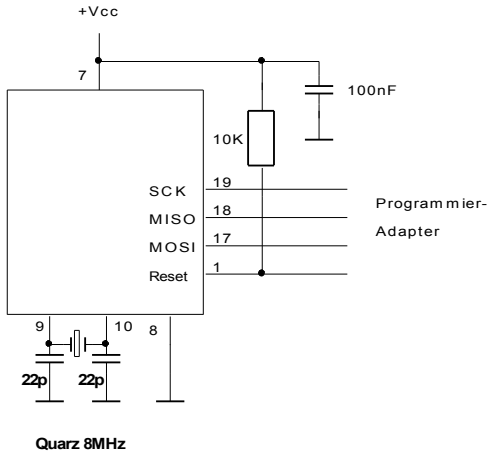
Frisch gekauft ist der AtMega8 so programmiert, dass er den internen RC-Oszillator benutzt (Taktfrequenz 1 MHz) und keinen externen Quarz braucht. In diesem Fall besteht die einfachste Schaltung aus dem Controller und einem Pullup-Widerstand am Reset-Pin:

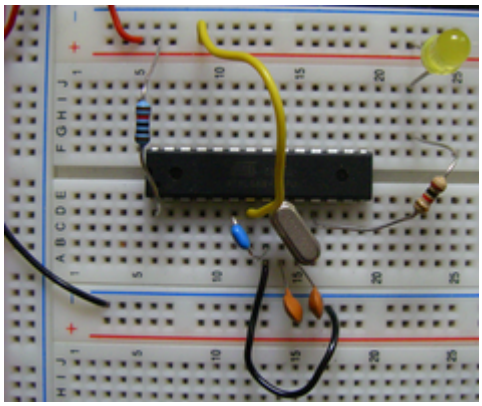


Der Entkoppelkondensator an der Betriebsspannung ist nicht immer notwendig, auf jeden Fall aber empfehlenswert.

Natürlich wird man mindestens eine LED mit Vorwiderstand oder irgendsonstwas anschliessen, damit der Controller etwas sichtbares tun kann.

Will man einen präziseren Takt haben oder andere Taktfrequenzen als 1MHz oder 8MHz, muss man den Controller mit einem Quarz betreiben:





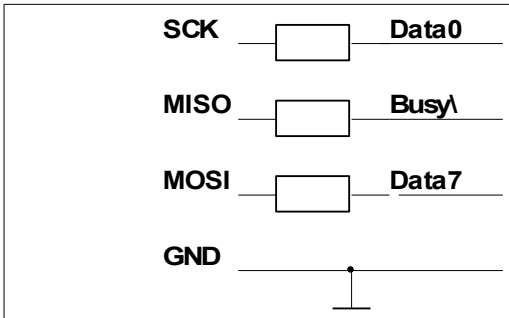
Eine solche Schaltung kann man schnell auf einem Breadboard (Experimentierplatine) oder auf Lochraster aufbauen.

Wenn ein Quarz benutzt wird muss Fuse Bit A987 auf 1111:1111 eingestellt werden.
(siehe Kapitel Fuses am Ende von Teil 1)

1.2 Programmieradapter

(auch „Brenner“ genannt)

Glücklich wer noch einen Parallelport am PC hat, in diesem Fall geht es besonders einfach.



Spartanisch:

Passiver SP12-Brenner

BASCOM-Einstellungen (Programmer):

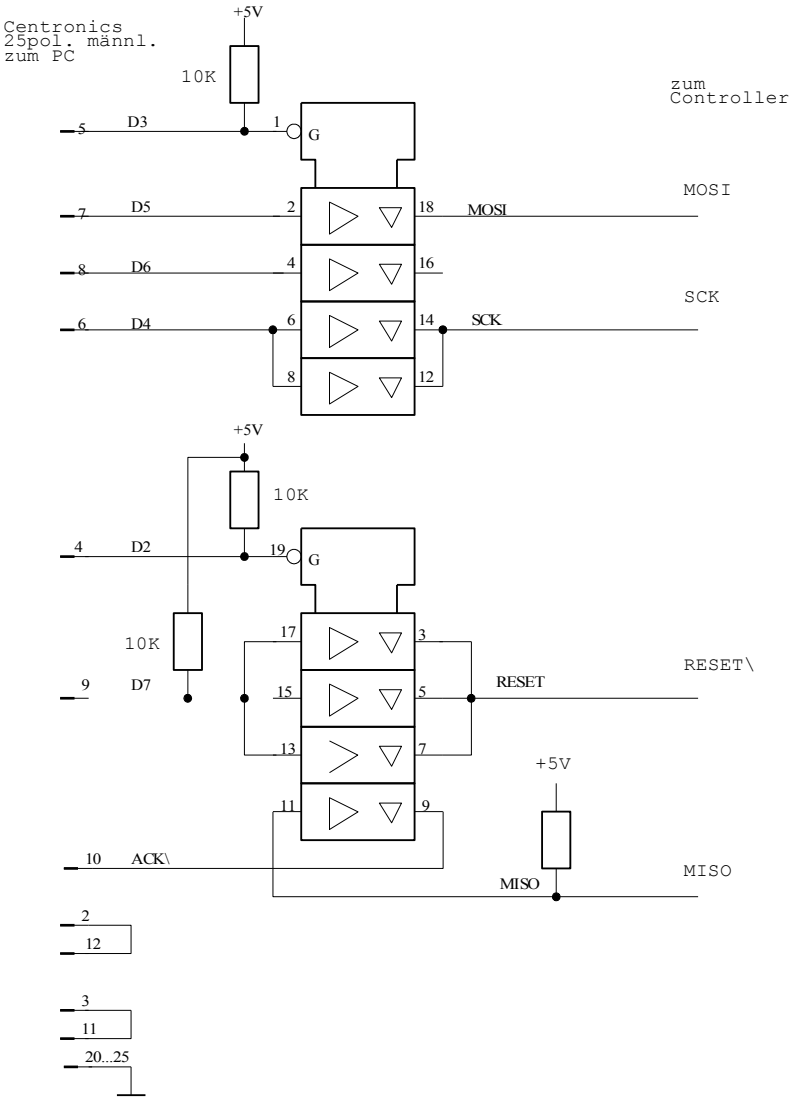
Universal MCS Interface,

WinAVR

Portadresse $378 für LPT1, $278 für LPT2

Aktiver Brenner

Sicherer ist eine Pufferung der Signale mit einem 74244:



BASCOM-Einstellungen (Programmer): STK200

Portadresse $378 für LPT1, $278 für LPT2

USB AVR Lab

Bei neueren PCs ist man auf ein USB-Programmiergerät angewiesen.

Von Atmel gibt es den AVRISP mkII.

Einen hervorragenden Ersatz als Fertiggerät oder im Selbstbau gibt es hier:

http://www.ullihome.de/index.php/USBAVR-ISP/de

Im Programmer sitzt ein Mega8-Controller

Der Nachteil beim Selbstbau ist, dass dieser bei der Inbetriebnahme zunächst programmiert werden muss, wofür man sich dann eventuell ein anderes Programmiergerät ausleihen muss, wenn man keines hat.

Einstellungen in BASCOM:

USBProg Programmer / AVRISP mkII, USB, AVRISPmkII.

Achtung: Clock muss auf einen Wert von weniger als ¼ der Taktfrequenz des zu programmierenden Controllers eingestellt werden!

Bei mir funktionierte das USBLab in der neuesten BASCOM-Version 1.11.9.5, nicht aber in 1.11.9.3. Also unbedingt updaten wenn nötig!

1.3 BASCOM

Eine kostenlose Demoversion (bis 4KB Code, das ist gar nicht so wenig) gibt es hier:

http://www.mcselec.com

Es lohnt sich aber auch BASCOM zu kaufen wenn man viel damit programmiert.

Optionen

Einige Optionen wie Chip, Quarzfrequenz, LCD, ... kann man voreinstellen.

Dies ist nicht unbedingt erfordert, wenn man die dafür nötigen Befehle ins Programm schreibt, was sehr von Vorteil ist da diese Optionen dann eindeutig dokumentiert sind.

Die einzige Option die unbedingt eingestellt werden muss, ist der Typ des Progrmmieradapters.

(siehe dort)

2. Grundlegende Vorgehensweise

Programm schreiben, compilieren und zum Controller schicken

- File - New
  Quelltext schreiben
- Program - Compile
- Program - Send to chip

  Bei umfangreicheren Programmen kann es nützlich sein, das Programm im eingebauten Simulator auszutesten.

3. Programmstruktur

(Beispiel):

```
'Deklarationen + Initialisationen:
        $CRYSTAL  = 4000000                 'Quarzfrequenz
        $Regfile = "m8def.dat"              'Registerdefinitionen Mega8
        Config PortD = Output
        Config PortB = Input
        Config PinC.3 = Input
        Config PinC.2 = Output
        Config ....

        DIM VariableA as byte
        DIM VariableB as integer
        ....

'Hauptprogramm ist immer eine Endlosschleife
        DO
                ......
                ......
        LOOP
        END
```

4. Controller mit externem Quarz

Ein „frischer" Mega8 ist so programmiert, dass der interne Taktgenerator mit 1MHz benutzt wird.

Ein aussen angeschlossener Quarz ist wirkungslos, solange nicht die entsprechenden Fusebits geändert werden.

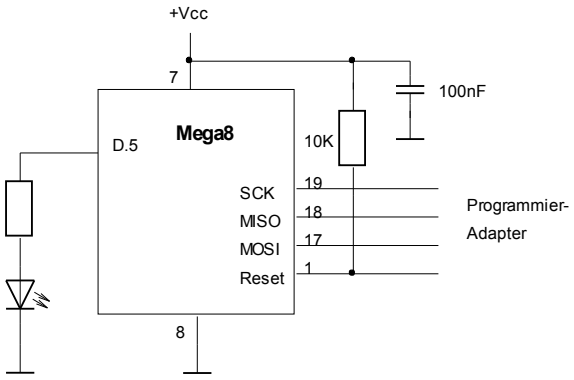
Fusebits für Mega8 und Quarzbeschaltung einstellen:

 Manuel Program - Lock and Fuse bits - Fuse bit A987 - 1111:1111 external crystal (siehe Kapitel „Fuse bits")

5. Port als Output: Hello World mit LED

Zusätzlich zur Minimalbeschaltung wird eine LED mit Vorwiderstand angeschlossen.

```
$crystal = 1000000
$regfile = "m8def.dat"


Config Pind.5 = Output

Do
   Portd.5 = 1
   Waitms 100
   Portd.5 = 0
   Waitms 200
Loop
```



Hier wird ein einziger Ausgang gesetzt und rückgesetzt.

Will man mehrere Portpins gleichzeitig setzen, geht das mit der so:

```
$crystal = 1000000
$regfile = "m8def.dat"

Config PortD = Output

Do
   Portd = &HFF          'alle LEDs an
   Waitms 100
   Portd = 0             'alle LEDs aus
   Waitms 200
Loop
```

Werden $CRYSTAL und $REGFILE nicht benutzt, so gelten die Einstellungen der Optionen!

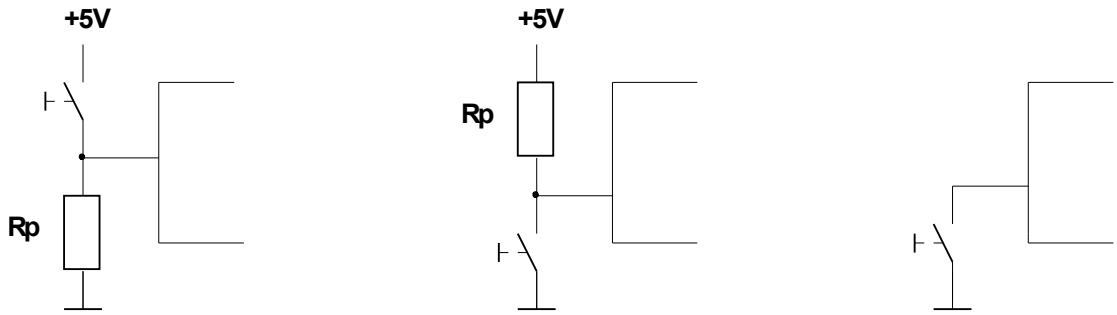
Konfiguration als Ein- oder Ausgang:
z.B.
Config **Port**B = Output / Input **für den ganzen Port** B
Config **Pin**C.3 = Output **für das einzelne Bit** 3 von PortC

Alternativ dazu kann man auch direkt in die Data Direction Register schreiben:
z.B.
DDRD = &B0001100 'D.2 und D.3 als Ausgang, die restlichen als Eingang
(1 = Ausgang, 0 = Eingang)

6. Port als Input: Anschluss von Tastern

Grundsätzliches:

Taster können gegen +UB oder Masse geschaltet werden. Im ersten Fall wird ein Pulldown-, im zweiten Fall ein Pullup-Widerstand benötigt, damit sich im Ruhezustand ein eindeutiges Potential am Eingang ergibt..



Der Wert von Rp ist unkritisch, 10kΩ ist ein guter Wert.

Taster nach	+UB	Masse	Masse
Zustand wenn betätigt	1	0	0
	Pulldown	Pullup	Interner Pullup

Die dritte Schaltung im Bild ist die am meisten verwendete, sie verwendet den internen Pullupwiderstand im Mikrocontroller.

Um diesen zu aktivieren, muss der Port als Eingang festgelegt und auf das Portregister an dieser Stelle eine 1 geschrieben werden (genaueres siehe Datenblatt!):

```
Config Pinb.0 = Input                    'Taster an B.0
Portb.0 = 1
```

Achtung: Der Eingang wird mit PinX.Y und nicht mit PortX.Y gelesen!

In BASCOM hat das Schlüsselwort Pin zwei unterschiedliche Bedeutungen:

Bei der Konfiguration als Ein- / Ausgang wird es benutzt für einen Pin im Gegensatz zum ganzen Port. Beim Lesen wird Pin für den Eingang benutzt.

Beispiel:

Das vorige Beispiel soll abgeändert werden , sodass die LED schneller blinkt, wenn ein Taster gedrückt wird.

Unser Programm könnte so aussehen:

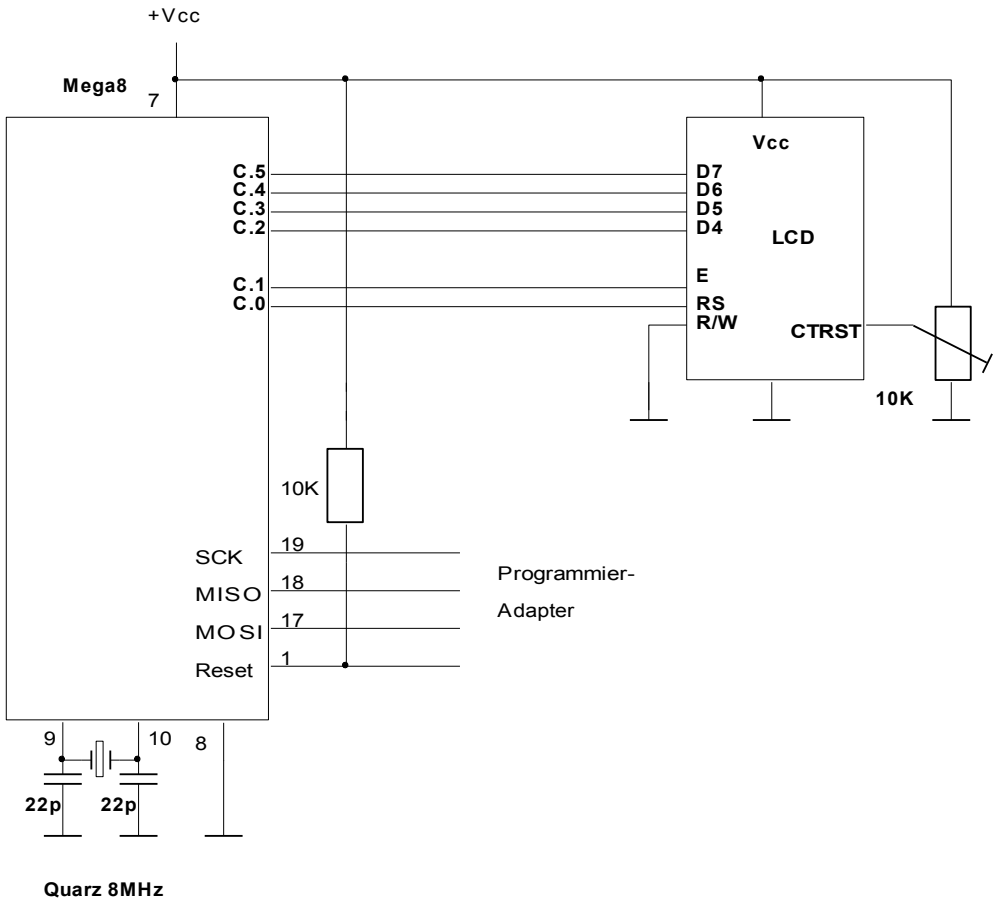
```
$crystal = 1000000
$regfile = "m8def.dat"

Config Pinb.0 = Input                    'Taster
Portb.0 = 1

Config Pind.5 = Output                   'LED

Do
   Portd.5 = 1
   Waitms 100
   Portd.5 = 0

   If Pinb.0 = 0 Then
     Waitms 100
   Else
     Waitms 500
   End If
Loop
```

Tasterprellen

Vorsicht: Taster prellen, sie liefern also mehrere Impulse beim Tastendruck. In unserem Beispiel ist das kein Problem. Wenn aber z.B. Tastendrücke gezählt werden sollen, muss man darauf aufpassen. Mehr dazu später.

7. Anschluss eines LCD-Displays

Beispiel:



(Der Quarz und die 22pF-Kondensatoren können weggelassen werden wenn der interne Oszillator benutzt wird)

Pinbelegung üblicher LCD:

Pin	14	13	12	11					6	5	4	3	2	1
	D7	D6	D5	D4					E	R/W	RS	CTRST	+5V	GND

In der obigen Schaltung wird das LCD-Display im 4-bit-Modus angesteuert.

BASCOM ist sehr flexibel, was den Anschluss angeht. Die Verbindung zum LCD kann im Prinzip über beliebige Portpins erfolgen, sie muss mit CONFIG LCDPIN festgelegt werden.

Im Programm muss das LCD mit 2 Befehlen konfiguriert werden:

- **Config LCD** legt die Grösse fest (20 * 2)
- **Config Lcdpin** legt die Anschlussbelegung fest
  (kann auch geändert werden, wenn nötig)

Vor dem Schreiben soll das Display mit **CLS** gelöscht werden. Dies bewirkt auch einen LCD-Reset.

Das Schreiben selbst erfolgt mit dem Befehl **LCD**

Mit **LOCATE** kann der Cursor positioniert werden, so dass man z.B. auch in die 2.Zeile schreiben kann.

Beispielprogramm:

```
$crystal = 8000000
$regfile = "m8def.dat"

Config Lcd = 16 * 2
Config Lcdpin = Pin , Db7 = Portc.5 , Db6 = Portc.4 , Db5 = Portc.3 , Db4 = Portc.2 , E = Portc.1, Rs= Portc.0

Cls

Do
    Lcd "Hello World"
    Waitms 500
    Cls
    Waitms 200
Loop
```

Das Programm gibt einen blinkenden Text aus.

Achtung: es gibt einzeilige LCD-Displays mit 16x1 Zeichen, die aber als 8x2 konfiguriert werden müssen!

8. Variablen

In vielen Programmen müssen Werte zwischengespeichert werden. Da das RAM des Mikrocontrollers knapp ist, muss man gut überlegen welche Art Variable man verwendet, um keinen Speicher zu verschwenden.

Beispiel:

Auf dem LCD-Display soll eine Zahl zwischen 0 und 59 angezeigt werden, die jede Sekunde um 1 hochgezählt wird.
(Dies könnte man für die Sekundenanzeige einer nicht sehr genauen Uhr benutzen, und leicht um Minuten und Stunden erweitern)

Da die Zahl kleiner als 255 ist, genügt eine Byte-Variable.

```
$crystal = 1000000
$regfile = "m8def.dat"

Config Lcd = 16 * 2
Config Lcdpin = Pin , Db7 = Portc.5 , Db6 = Portc.4 , Db5 =
Portc.3 , Db4 = Portc.2 , E = Portc.1 , Rs = Portc.0

Dim X As Byte

Cls

Do
    if X=60 then X=0
    Lcd X
    Wait 1
    Cls
    X = X + 1
Loop
```

BASCOM speichert die Variablen im RAM ab Adresse 60h ab.

Variablentyp	Speicherverbrauch	Wertebereich
8 Bit	1 Byte	
Byte	1 Byte	0...255
Integer	2 Byte	-32768...+32767
Word	2 Byte	0...65535
Long	4 Byte	-2 147 483 648...2 147 483 647
Single	4 Byte	$\pm 1.5 \times 10^{-45}$ ...3.4×10^{38}
String der Länge L	L+1 Byte	max. 254 Zeichen

9. Unterprogramme

Bei größeren Projekten ist es sinnvoll, das Programm in übersichtliche kleine Unterprogramme aufzuteilen. Dies erleichtert auch eine spätere Änderung im Programm, weil sich leichter zurechtfindet als in einem einzigen unübersichtlichen Programm.

Das obige Beispiel könnte man dann so schreiben:

```
.......
'Hauptprogramm:
Do
    If X = 60 Then X = 0
    Gosub Displayx
    Wait 1
    X = X + 1
Loop
'-----------------------------------------------------------------
'Unterprogramm für die Anzeige
Displayx:                          'Label = Name des UP (endet mit „:“)
   'Vorigen Wert löschen
   Locate 1 , 1                      'Cursor setzen
   Lcd Space(10)
   'Neuen Wert schreiben
   Locate 1 , 1
   Lcd X
Return                             'Rücksprung ins Hauptprogramm
```

Hier wurde auch eine etwas elegantere Art benutzt, um den Wert zu schreiben.

Wir verzichten auf das langsame CLS, das auch ein unangenehmes Flackern bewirkt, und löschen den vorigen Wert indem wir ihn mit Spaces überschreiben. Das geht schneller und ist flackerfrei.

In BASCOM gibt es noch eine andere Art Unterprogramme aufzurufen. Diese benutzt die Definition des UP mit SUB...

10. Taster – Tasterprellen

Tasten werden oft zum Auswählen von Menüpunkten oder ähnlichem verwendet.

In unserem einfachen Demo-Beispiel soll eine Zahl hochgezählt werden, wenn eine Taste mit internem Pullup-Widerstand an B.0 gedrückt wird.

Dies ist nicht ganz so trivial wie man meinen könnte, denn eine Lösung mit

```
IF PinB.0 THEN X = X+1
```

führt dazu, dass X während des Tastendrucks hochgezählt wird, und das geht mit unglaublicher Schnelligkeit (gebremst allerdings durch die LCD-Anzeige).

Um dies zu vermeiden, muss man, wenn die Taste betätigt wird, warten bis sie wieder losgelassen wird.

```
'Taster an B.0 : Hochzählen jedesmal wenn Taster gedrückt
'Version mit Tasterprellen

$crystal = 1000000
$regfile = "m8def.dat"

'Taster mit internem Pullup
Config Pinb.0 = Input
```

```
Portb.0 = 1

Config Lcd = 16 * 2
Config Lcdpin = Pin , Db7 = Portc.5 , Db6 = Portc.4 , Db5 = Portc.3 , Db4 = Portc.2 , E = Portc.1 , Rs =
Portc.0

Dim X As Byte

Cls
Lcd X
'-----------------------------------------------------------------------------------------------------------
'Hauptprogramm:
Do
    If Pinb.0 = 0 Then

      'Warten bis Taster losgelassen
      Do
      Loop Until Pinb.0 = 1

      'X erhöhen
      X = X + 1

      Gosub Displayx            'Wert anzeigen, UP wie oben
    End If

Loop
'-----------------------------------------------------------------------------------------------------------
.......
```

Aber auch hier gibt es noch ein Problem:
Taster prellen, d.h. pro Tastendruck ergeben sich eventuell mehrere Impulse, so dass X nicht nur einmal, sondern mehrmals inkrementiert wird.

BASCOM bietet hier eine elegante Lösung an, den **DEBOUNCE** (Entprell) Befehl.
Dieser macht intern eine Kombination aus Verzögerung und Warten auf das Loslassen der Taste.
Er muss immer zusammen mit einem Unterprogramm benutzt werden.

```
.............
'Hauptprogramm:
Do
  Debounce Pinb.0 , 0 , Reaktion_auf_taste , Sub
Loop
'-------------------------------------------------------------
Reaktion_auf_taste:
      'X erhöhen
       X = X + 1

       'Wert anzeigen: letzten löschen und neuen schreiben
       Locate 1 , 1
       Lcd Space(10)
       Locate 1 , 1
       Lcd X
Return
```

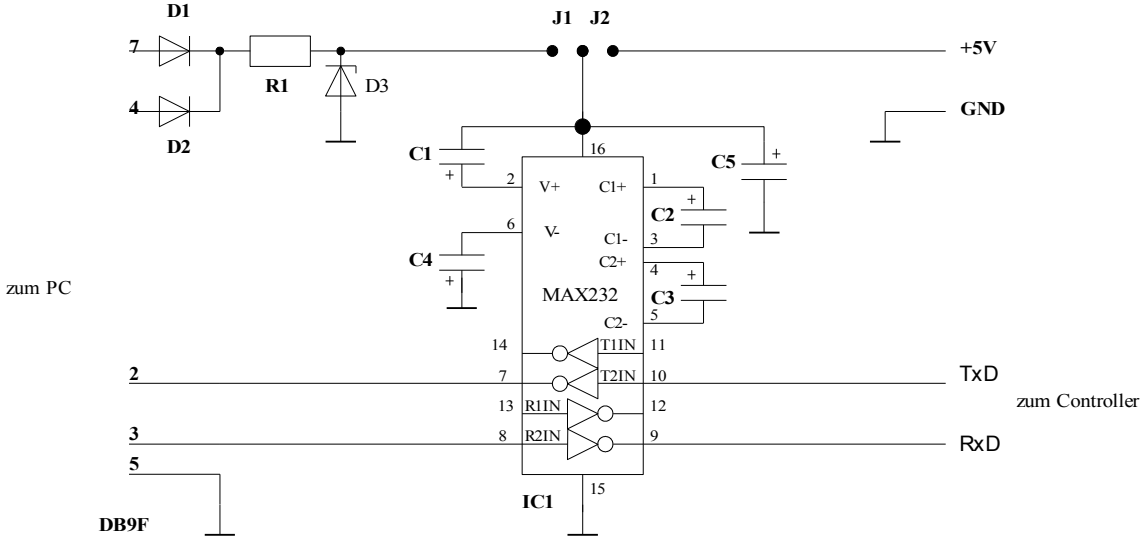
11. Serielle Schnittstelle

Diese bei PCs leider fast ausgestorbene Schnittstelle bringt viele Vorteile bei der Kommunikation mit dem Controller. Man kann sie zum Debuggen benutzen, indem man sich die Inhalte von Variablen an den PC schickt, oder man kann dem Controller Befehle schicken, um irgendetwas im Programm zu verändern.

11.1 Hardware

Die RS232 - Signale des PCs haben ±12V, während der Controller mit 5V-Pegel arbeitet. Ausserdem ist die Logik invertiert.

Es muss also eine Anpass-Schaltung zwischengeschaltet werden, z.B. die klassische MAX232-Schaltung, welche in der hier abgebildeten Form ihren "Saft" eleganterweise aus der RS232-Schnittstelle des PCs beziehen kann , wenn J1 gesetzt ist (Voraussetzung ist natürlich ein 1:1-Kabel, welches auch die Signale der Pins 4 (DTR) und 7 (RTS) führt).
Die Speisung kann aber auch vom Controller-Board aus erfolgen, wenn J2 gesetzt ist.



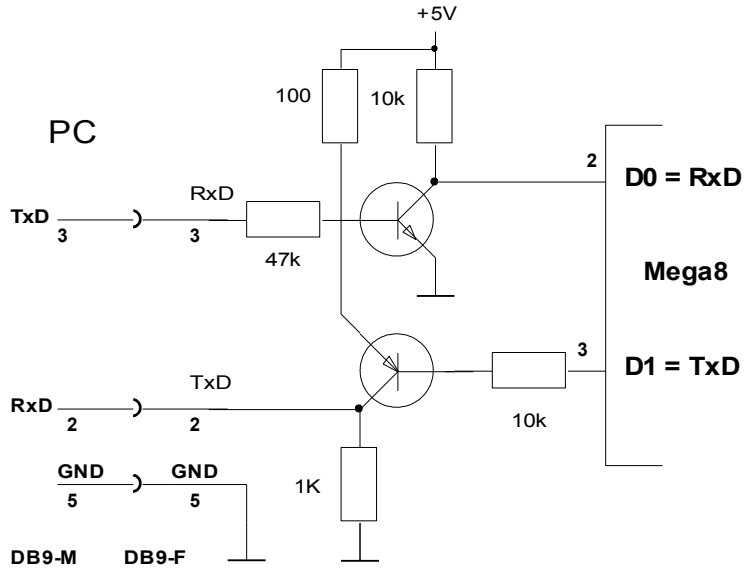
D1, D2 = 1N4148
D3 = ZD4v7
R1 = 100Ω
C1...C5 = 1µF

J1 = Stromversorgung vom PC
J2 = Stromversorgung von der Kontrollerplatine

Die Verbindung nach rechts zum Mikrokontroller ist beim Mega8 die folgende:

RxD	PortD.0 (RxD)
TxD	PortD.1 (TxD)

Die Anpass-Schaltung kann übrigens auch mit 2 Transistor-Invertern aufgebaut werden.
An RxD des PCs (Pin 2) genügt meist schon ein 5V-Pegel, so dass die Schaltung nur 5V Betriebsspannung braucht:



T1 (npn) an RxD: z.B BC547 o.ä.

T2 (pnp) an TxD: z.B. BC557 o.ä.

11.2 Senden eines Textes

1. Baudrate festlegen z.B. $BAUD=9600 (mit den Standardeinstellungen 8bit, no parity 1 Stop bit wie standardmäßig in Options – Communication festgelegt.)
   Vorsicht!
   Da die Baudrate aus der Taktfrequenz des Controllers abgeleitet wird, kann man bei niedrigen Taktfrequenzen keine hohen Baudraten wählen, ohne dass sich ein unzulässig hoher Fehler ergibt. Im Menüpunkt Program – Show results kann man sich unter anderem den Fehler in der Baudrate anzeigen lassen.

2. PRINT benutzen für die Ausgabe.
   Normalerweise fügt PRINT einen Zeilenumbruch hinzu, ausser wenn am Ende mit „;“ abgeschlossen wird (siehe Beispiel)

Testprogramm:

```
$crystal = 4000000
$regfile = "m8def.dat"
$baud = 9600

Dim X As Byte

Do
    Print "Hello";
    Print "    ";
    Print X
    Wait 1
    X = X + 1
Loop
```

Ausgabe im Terminalfenster von BASCOM (Tools – Terminal emulator):

```
Hello     1
Hello     2
Hello     3
Hello     4
```

```
Hello     5
Hello     6
.......
```

11.3 Empfangen eines Textes

Textzeilen vom PC, die mit <CRLF> (Carriage Return + Line Feed) abgeschlossen sind, wie es beim Betätigen von <Enter> im Terminalprogramm geschieht, werden mit INPUT empfangen.

Beispiel:

```
$crystal = 4000000
$regfile = "m8def.dat".dat "
$Baud = 9600
Dim S As String * 10

Do
        Input "Please type your name:" , S
        Print "Hello ";
        Print S
Loop
End
```

Terminal:

```
Please type your name:Jean-Claude
Hello Jean-Claude
```

Hier wurde eine Stringvariable S dimensioniert um den Text zu empfangen. Diese muss hinreichend groß sein, damit alle Zeichen hineinpassen. Der Text in „“ hinter INPUT wird vor dem Input an den PC ausgegeben.

11.4 Empfangen einzelner Zeichen

Oft ist es sinnvoll, statt eines Textes nur einzelne Zeichen (je 1 Byte) zu schicken.

Beispiel:

Eine LED soll durch Senden von „1“ eingeschaltet und durch „0“ ausgeschaltet werden.

```
$crystal = 4000000
$regfile = "m8def.dat".dat "
$Baud = 9600
Dim S As Byte

Config Portd.5 = Output                    'LED  gegen Masse

Do
  S = Inkey()                              'Ist ein Zeichen da?
  If S <> 0 Then

        Select Case S                      'Wenn ja, welches?
              Case Asc( "0")
                    Print "Aus"
                    portd.5=0
              Case Asc( "1")
                    Print "Ein"
                    Portd.5 = 1
        End Select

  End If
```

```
Loop
End
```

Der INKEY-Befehl gibt den ASCII-Wert zurück, wenn ein Zeichen anliegt, sonst null.

SELECT CASE erlaubt eine bequeme Wahl verscheidener Möglichkeiten.

11.5 Empfang mit Interrupt

Die hier vorgestellte Methode heißt „Polling“. Sie ist einfach, aber sehr ineffektiv, da das Hauptprogramm eigentlich nichts anderes tun kann als auf ein zu empfangendes Byte zu warten. Braucht es einmal länger für eine Berechnung oder sonstwas, besteht die Gefahr dass zu empfangende Bytes verpasst werden.

Das gleiche gilt natürlich auch für die Polling-Methode bei Tasterabfragen.

Günstiger ist es, mit INTERRUPTs zu arbeiten. Mehr dazu siehe Kapitel Interrupts.

11.6 Software-UART

Oft ist es sinnvoll, statt eines Textes nur einzelne Zeichen (je 1 Byte) zu schicken.

Diese Option ist interessant wenn man mehrere serielle Schnittstellen braucht, der Controller hardwaremässig aber nur eine anbietet.

Die Pins für TxD und RxD müssen jeweils einzeln mit dem OPEN - Befehl als “Gerät” Nummer *x* konfiguriert werden (angelehnt an die DOS-Syntax).
Anschliessend werden sie mit dem PRINT #*x* beschrieben oder mit INPUT #*x* bzw INKEY (#x) gelesen.

Beispiel für das Senden:

```
Open "comd.3:9600,8,n,1" For Output As #1
Print #1 , "serial output"
```

Beispiel für Empfangen und Senden:

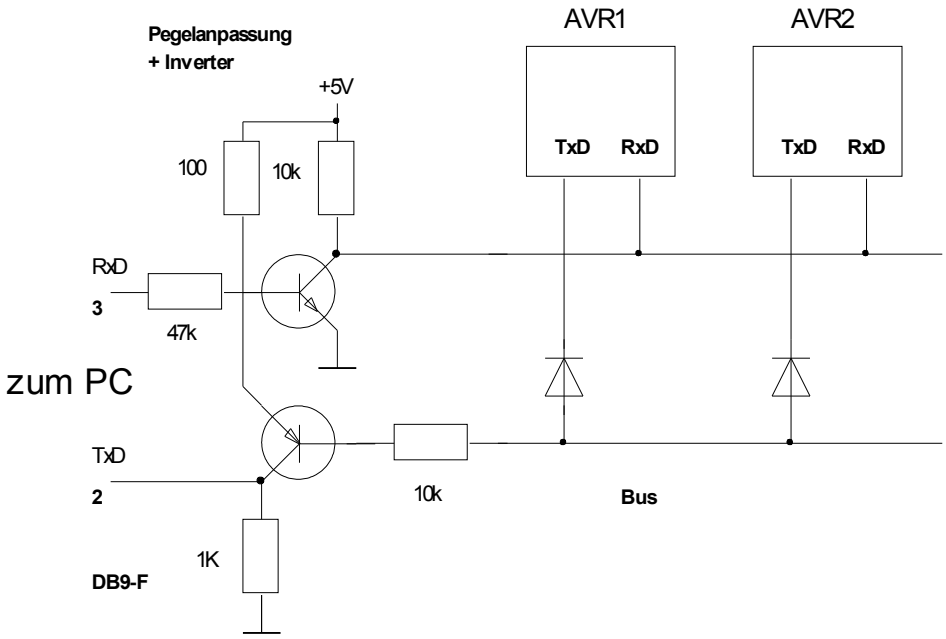
```
Open "comd.3:9600,8,n,1" For Output As #1
Open "comd.2:9600,8,n,1" For Input As #2
Print #1 , "serial output"

Do
   S = Inkey(#2)
   If S <> 0 Then
         S = S + 1
        Print #1 , S
    End If
Loop
```

Der CLOSE-Befehl sollte hier nicht benutzt werden.
Im obigen Programm würde er nur überflüssig sein. Wenn PRINT #.... in einem Unterprogramm benutzt wird welches (wie es normalerweise der Fall ist) hinter dem Hauptprogramm steht und in diesem CLOSE #.. zum Schliessen verwendet wird, dann kommt es zu Fehlermeldungen.

Weitere interessante Befehle:

```
INPUT, INPUTBIN, INPUTHEX
PRINT
ISCHARWAITING, WAITFORKEY
```

11.7 Mehrere Controller an einem RS232-Port

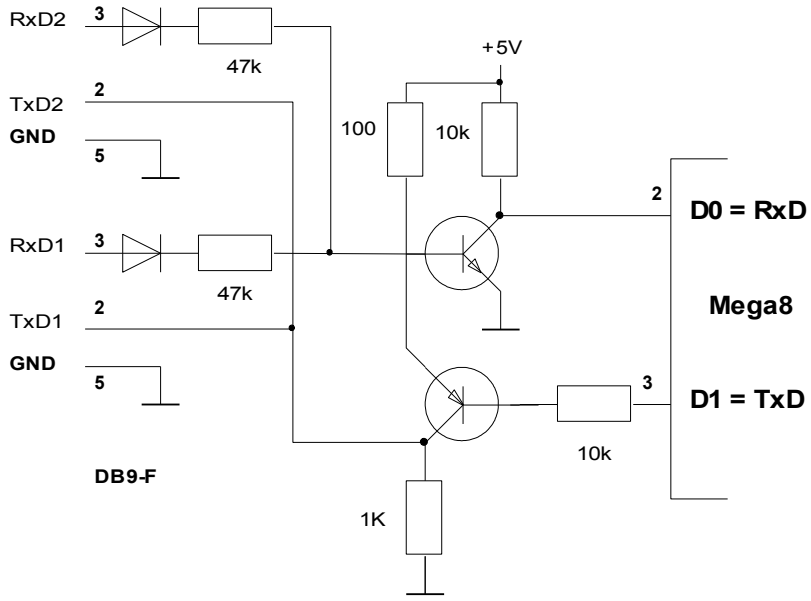


Mit dieser Schaltung kann der PC als Master Kommandos an mehrere AVRs schicken die dann als Slaves arbeiten.

- Der TxD-Ausgang des AVR hat im Ruhezustand +5V, die L-Pegel bei der Übertragung bewirken über die Diode Basisstrom für den pnp-Transistor und H am PC-Stecker.
- Die RxD-Eingänge liegen parallel so dass alle AVRs gleichzeitig “hören” können.
- Wichtig ist, dass keine 2 AVRs gleichzeitig senden.

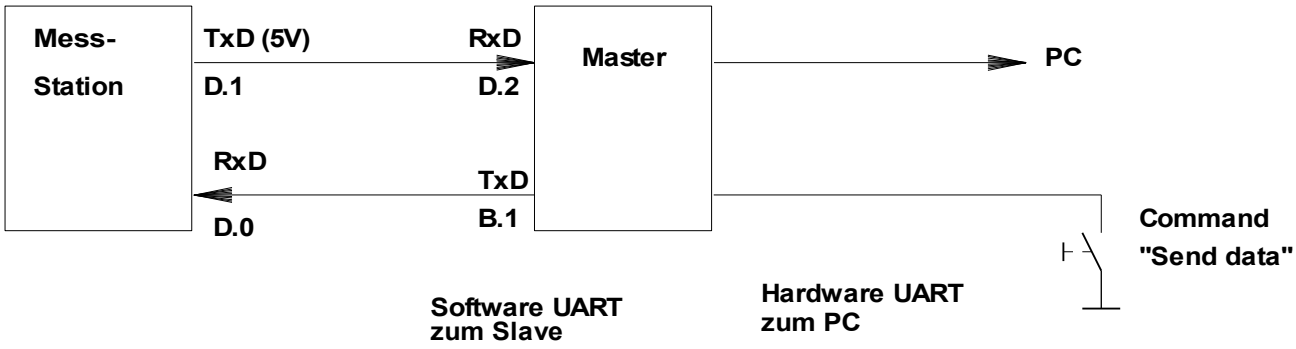
11.8 Mehrere RS232-Schnittstellen an einem Mega8

Manchmal soll ein Controller mit 2 anderen Geräten kommunizieren können, wobei aber nur eine Verbindung aktiv ist.



Für TxD ist keine besondere Maßnahme nötig, das Signal wird einfach auf beide Schnittstellen ausgegeben.
Bei RxD wird mit Dioden entkoppelt. Dies funktioniert natürlich nur, wenn nicht von den 2 Schnittstellen gleichzeitig Signale ankommen.

11.9 Tücken der Kommunikation zwischen zwei Controllern



In der obigen Konfiguration wurde ein Controller als Master benutzt, um Daten von einer Mess-Station abzurufen. Hier ist ein Problem die Synchronisation zwischen Master und Slave.

Im Beispiel sendet der Master ein bestimmtes Zeichen, woraufhin der Slave seine Daten schickt, jeweils mit einem PRINT-Befehl. Dieser schickt am Ende eines Datenpäckchens ein CRLF (Zeilenende und Zeilenvorschub: ASCII-Zeichen 13, 10) mit, was man zum Synchronisieren nutzen kann.

Der Slave liest die Daten mit dem INPUT-Befehl, welcher auf ein CR(LF?) wartet.

Diese Lösung funktionierte nicht auf Anhieb. Nach einigem Debugging-Aufwand stellte sich heraus warum: Der INPUT-Befehl lässt ein LF (chr(10)) im Puffer stehen, welches ab dem zweiten Datenpäckchen Probleme macht.

Zwei Lösungen sind denkbar:

- der Slave sendet z.B.
  PRINT x;
  Print chr(13);
  also nur CR statt CRLF

- der Master liest die Daten als String und entfernt bei Bedarf das überflüssige LF am Beginn. Erst dann werden sie in Zahlenwerte gewandelt.

Im Beispiel wurde die zweite Lösung benutzt, die Daten liessen sich damit problemlos empfangen.

.....

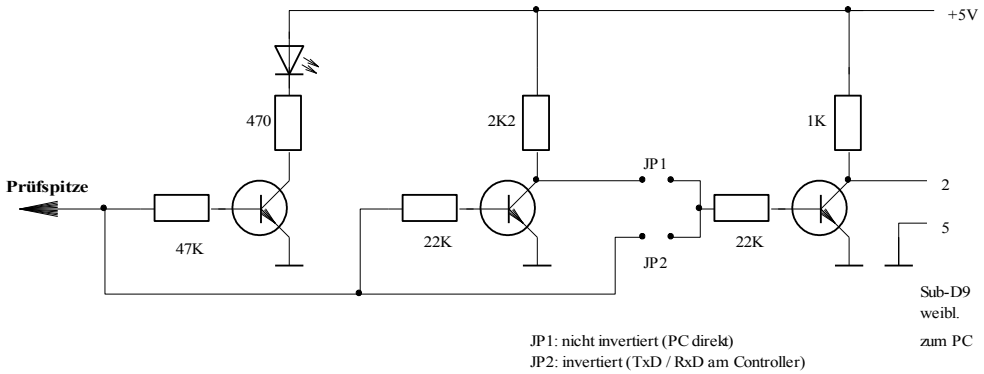
```
'Command to station
Print #1 , "*"                                  'b.1

'Read 7 pieces of data terminated by CR
For I = 1 To 7
   Input #2 , S(i)
Next I

'correction if LF (10) left in the buffer
' (this occurs on all but first item, as input cuts behind the CR)
For I = 1 To 7
   If Left(s(i) , 1) = Chr(10) Then S(i) = Mid(s(i) , 2)
Next I

'Assign data to variables for processing
Secs = Val(s(1))
Temperature_outside = Val(s(2))
```

.....

11.10 Datenspion zum Debuggen serieller Kommunikation

Die folgende Schaltung erlaubt es mithilfe eines Terminalprogramms (wie das von BASCOM) zu sehen was über die TxD / RxD – Leitungen geschickt wird.

Dies ist oft hilfreicher als die Benutzung eines Oszilloskops, mit dem man die Zeichen mühsam decodieren müsste.



Mit dem Jumper an JP1 / JP2 kann eingestellt werden ob das Signal invertiert wird oder nicht, je nachdem ob es direkt an einem Controller oder hinter einem Pegelwandler auf dem Weg zum PC gemessen werden soll.

Die erste Stufe mit der LED ist ein einfacher Logiktester, der auch zum Ansehen des logischen Zustandes an beliebigen Pins benutzt werden kann.

12 Fuse bits

Die Fuse bits erreicht man über „Program – Send to chip – Manual program“

Vorsicht:

Mit den Fuse bits herumzuspielen ist gefährlich! Es kann dazu führen dass der Controller nicht mehr ansprechbar ist.

Taktquelle einstellen: Fusebit KLA987

Am wichtigsten sind die Möglichkeiten

- **Int RC Osc** 1MHz / 2MHz / 4MHz / 8MHz für den Betrieb ohne Quarz, mit internem Oszillator
- Ext Crystal / Resonator High Freq. (111111) für den Betrieb mit Quarz.

Brown Out detection

BODEN enabled bewirkt dass ein Reset erfolgt, wenn die Betriebsspannung zusammenbricht. Dies kann unkontrollierte Reaktionen des Controllers verweiden helfen, wenn die Speisespannung mit Störsignalen „verseucht“ ist, wie es z.B. bei Robotern häufig der Fall ist, oder wenn sie beim Ausschalten langsam gegen null geht (langsam sich entladende Elkos).

Immer empfehlenswert ist :BODEN = 1 um diese Probleme zu vermeiden.

Nach dem Einstellen müssen die Fuse bits mit „Write FS“ geschrieben werden.

Von den anderen Fusebits lässt man am Besten die Finger, ausser wenn man das Datenblatt studiert hat und genau weiss, was man tut.

So, das wäre der erste Teil des Tutorials.

Im zweiten soll es über Interrupts und Timer gehen.
