Inhaltsverzeichnis

Was ist ein Mikrokontroller? 1
Hardware und Software 1
Das Seeeduinoboard 1
Die Programmiersoftware BASCOM 2
Die Anschlüsse des Mikroprozessors 7
Schnittstelle zur Außenwelt 8
Das Prototype-Shield 8
LEDs als Statusanzeige 9
Bargraph Anzeige (Balkenanzeige) 11
7-Segment-Anzeige 12
Taster als Eingabegerät 14
Wie rechnet eigentlich ein Mikrocomputer? 18
Das LCD-Display 19
Variablen, Schleifen und Bedingungen 21
  Variablen 21
  Schleifen mit Abbruchbedingungen 22
    For ➔ Next 22
    While ➔ Wend 22
  Sprung in ein Unterprogramm 23
  Conditional Instruction 24
    IF ➔ THEN 24
    IF ➔ THEN ➔ ELSE 24
    Select CASE 25
Analog-Digital Converter (ADC) 26
Multiplex 28

Was ist ein Mikrokontroller?

Das sind programmierbarer Chips, die wie ein Minicomputer (Prozessoren) arbeiten und heute in allen Handys, Taschenrechnern, Waschmaschinen, Fahrkartenautomaten, aber auch in Autos und Maschinen etc. eingebaut sind. In diesen Geräten wird ein Mikrokontroller programmiert, um z.B. zu erkennen, wann eine Taste gedrückt wird, oder um eine Digitalanzeige anzusteuern. Mikrokontroller werden dafür programmiert, Sensordaten zu lesen, die Daten zu verarbeiten, um dann Entscheidungen zu treffen, indem Texte auf Displays auszugeben werden. Ebenso kann aber auch eine LED oder ein Motor angesteuert werden. Wie das geht werden wir in den nächsten Wochen lernen. Als Hardware werden wir dazu das Open Source Projekt *Arduino* benutzen, für das es je nach Anwendung weitere Aufbaumodule gibt.

Hardware und Software

Mit der Programmierung eines Arduino-Projekts anzufangen ist ähnlich, als wenn Du einen neuen PC oder Laptop in Betrieb nehmen willst. Das erste, was die meisten Leute tun, wenn sie einen neuen Computer haben, ist ihn auszupacken, einzustecken, die Software zu installieren und auszuprobieren und vielleicht sogar ein paar eigene Programmzeilen in einer Programmiersprache zu schreiben. Wenn Du den Arduino zum ersten Mal einsetzt, wirst Du etwa dieselben Dinge tun. Im Folgenden wird Dir gezeigt, wie Du ein Arduino zum Laufen bringst:

- Installieren der Programmierungs – Software (**BASCOM**) und der Software für den Programmieradapter,
- Anschluss des Arduino Moduls an den PC für die Programmierung,
- Schreiben eines ersten BASIC Programms,
- Anschluss des Arduino an die (Batterie-) Stromversorgung,
- Start des ersten Programms.

Das Seeeduinoboard

Das Seeeduino-Board ist vergleichbar mit einem Motherboard bei einem PC (s. Abb. 1.1).

- Auf ihm sitzt der Prozessor, eine kleiner unabhängiger Rechner, der programmiert werden kann **[1]**.
- Dieses geschieht über einen Programmieradapter der am ICSP-Stecker **[2]** angeschlossen wird.
- Weiterhin besitzt der Seeedunio einen USB Anschschluss **[3]**, mit dem er Daten mit einem PC oder Notebook austauschen kann.
- Die Verbindung zur "Außenwelt" funktioniert über die Steckerleisten **[4]** und **[5]**, indem einzelne Ports abgefragt oder angesteuert werden. Das geschieht z.B. indem man weitere Shields (Boards) z.B. für ein Display oder eine Tastatur auf das Motherboard aufsteckt.

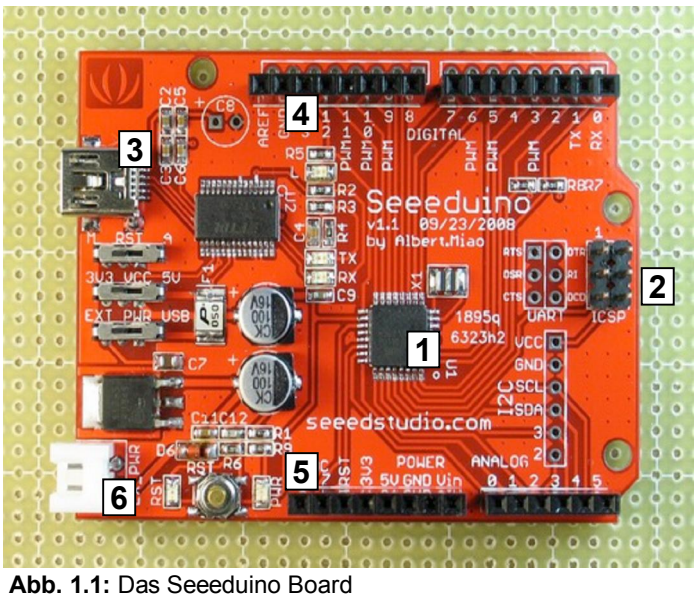


Abb. 1.1: Das Seeeduino Board

- Am Anschluss **[6]** kann eine externe Stromversorgung wie eine Batterie angeschlossen werden, damit das Arduino-Projekt eigenständig ohne PC läuft

Die Programmiersoftware BASCOM

BASCOM ist die Software mit dem Dein Arduino-Projekt in der Programmiersprache *Basic* programmiert wird (Alternative Sprachen sind C++, Assembler, etc.).
Diese Software ermöglicht es Dir, auf Deinem Computer Programme zu schreiben und diese in den Mikrokontroller auf den Seeeduino zu laden. Dieses Programm ist bereits auf den Rechnern installiert oder kann von folgender Website als Demo heruntergeladen werden ➔ http://www.mcselec.com

Bevor aber ein eigenes Projekt gestartet wird, müssen einige Einstellungen vorgenommen werden.

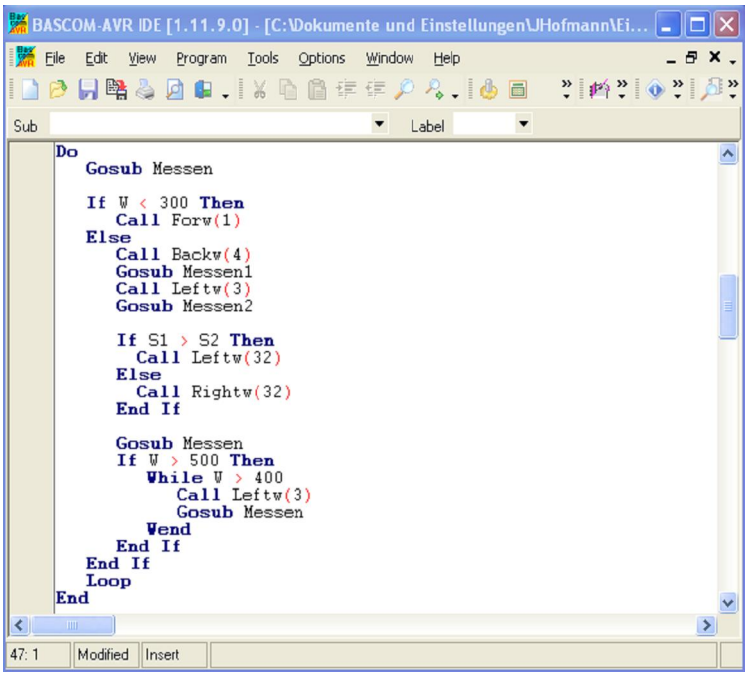


Abb. 1.2: Bascom Programmeditor

Dazu auf ➔ Options➔ Compiler

➔ Chip
Zuerst muss dem Programm erklärt werden, welcher Chip programmiert werden soll (s. Abb. 1.2).
Bei einem Seeeduino ist es der Prozessor **M 328** (m328pdef)

➔ LCD
Ebenfalls kennt der Controller noch nicht, welches Display angeschlossen ist und welche Ports (EN, RS, DB4-DB7) dazu verwendet werden.
Außerdem muss der LCD-Typ eingestellt werden.
(Anzahl Zeichen und Zeilen)

Alle Einstellungen müssen nach den Vorgaben der Abbildungen eingestellt werden

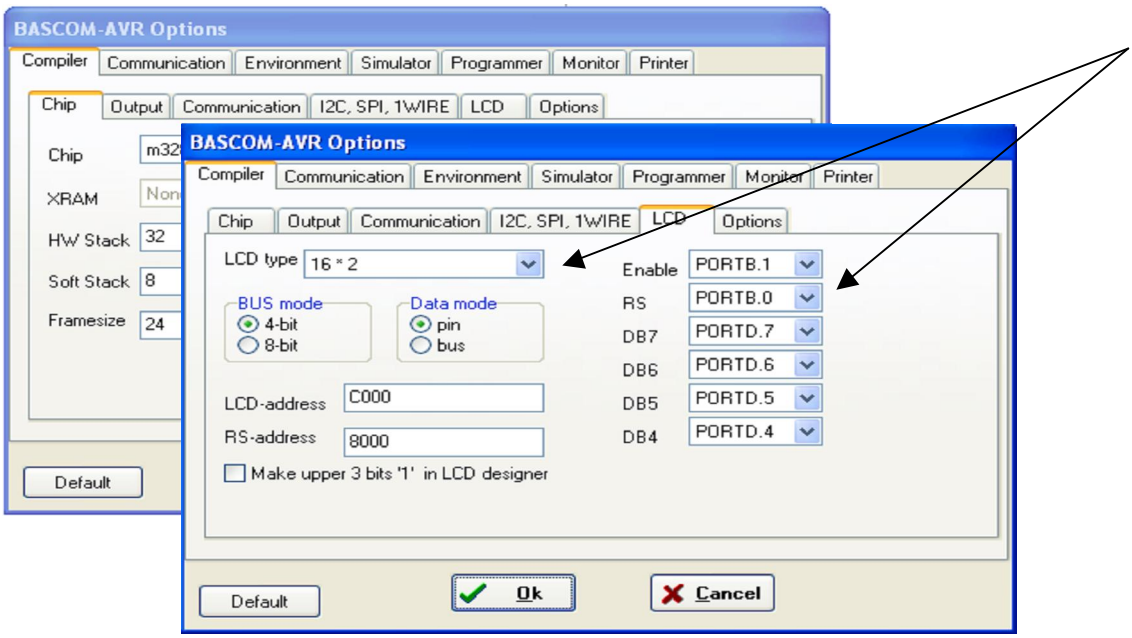


Abb. 1.3: Einstellung von Bascom AVR Options

Programmieradapter

Um den Mikrokontroller zu programmieren wird ein Programmieradapter benötigt, der das Programm auf den Mikrokontroller "brennt". Die Treiber für den Programmieradapter werden nach Herstellerangaben installiert.
Damit BASCOM nun weiß, welcher Programmieradapter verwendet wird, müssen auch hier Einstellungen vorgenommen werden.
Dazu wird als Programmer der STK500 gewählt, wobei die Software ebenfalls noch installiert werden muss. Diese findet man in Google, wenn man unter "stk500.exe sucht und sich das Programm z.B. bei *AVR Freaks* herunterlädt und installiert.

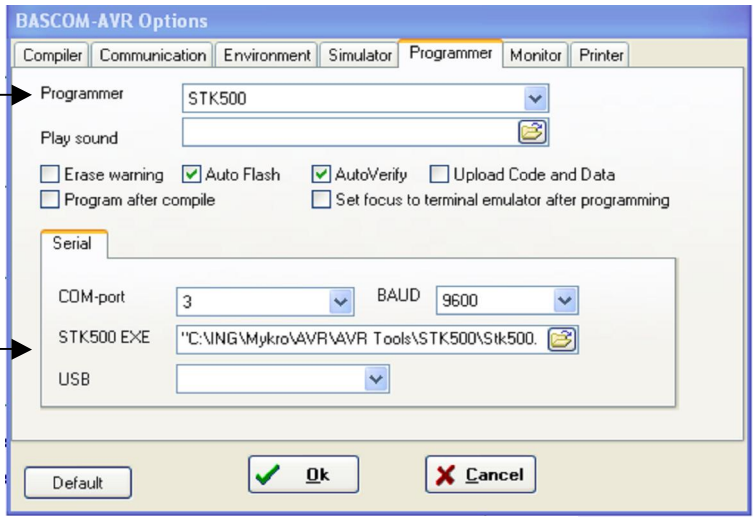


Abb. 1.4: Auswahl des Programmieradapters

Danach muss angegeben werden, an welcher Stelle die Software für den Stk500 installiert wurde.

Zum Schluss muss der Programmer nur noch über den USB Anschluss mit dem PC oder Notebook und dem Seeeduino verbunden werden.

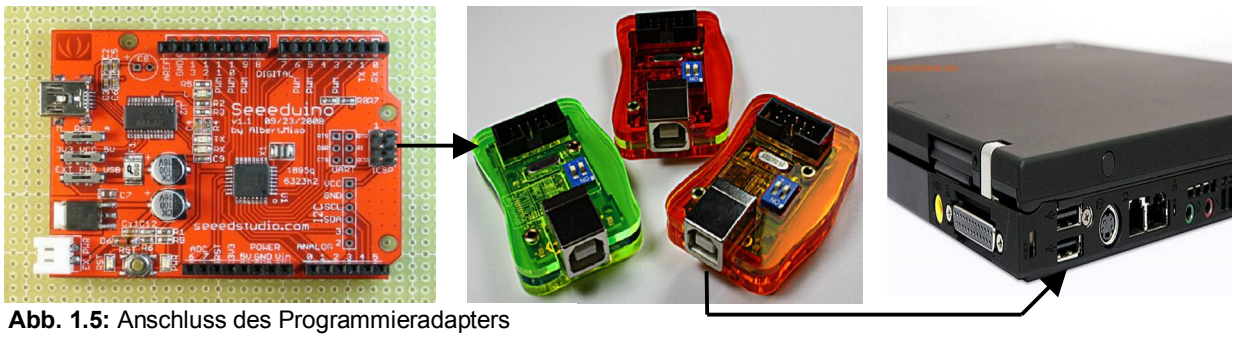


Abb. 1.5: Anschluss des Programmieradapters

So jetzt hast Du eine Menge über die Installation der Programme und verschiedene Softwareeinstellungen, die notwendig sind, erfahren. Klingt kompliziert, ist aber einfacher als Du Dir das jetzt vielleicht vorstellst.
Im Übrigen habe ich für Euch schon sämtliche notwendigen Programme installiert. Falls du aber auch zu Hause mal Programme schreiben willst, dann könnten diese Informationen sehr nützlich sein.

Viel Spass beim Programmieren !!!

Testprogramm "Hallo World"

Das erste Programm, das wir schreiben und auf den Mikrokontroller übertragen werden, soll uns zeigen, ob der Mikrokontroller einsatzbereit ist und ob die Übertragung durch den Programmieradapter funktioniert. Dazu nutzen wir eine winzig kleine LED, die auf dem Seeeduino zu finden ist, die wir mit dem Mikrokontroller ansteuern (s. Abb. 2.1).

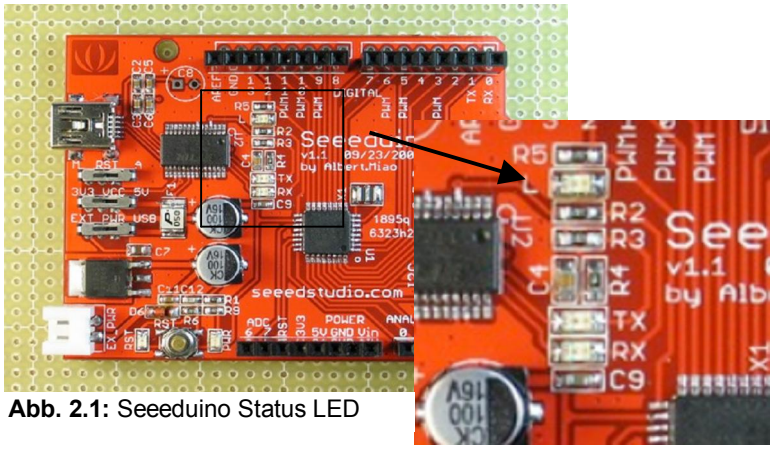


Abb. 2.1: Seeeduino Status LED

Dazu müssen man ein kleines Programm schreiben, dass aus einzelnen Befehlen besteht, die der Mikrokontroller versteht. Dazu verwendet man die Programmiersprache *BASIC*, eine von mehreren Sprachen, die der Mikrokontroller verstehen kann.

Ganz praktisch bei unserem kleinen Testprogramm ist, dass wir auch schon ein paar ganz einfache Befehle kennenlernen (s. Abb. 2.2).

Zuerst schreibt man ein Programm in einem sogenannten **Programmeditor**

Wenn das Programm fertig geschrieben ist, wird es von einem **Compiler** übersetzt, damit es der Microprozessor auch verstehen kann. (Hier Button drücken)

Damit das Programm auf den Mikrokontroller übertragen wird, muss dieser Button abschließend gedrückt werden.

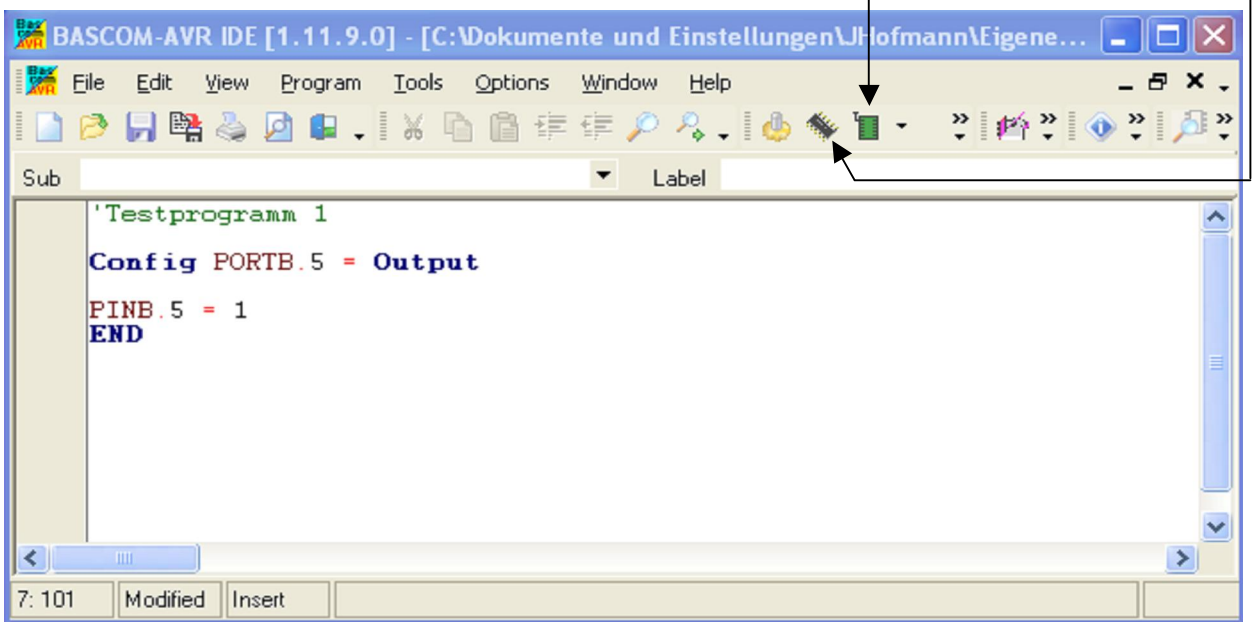


Abb. 2.2: Schreiben eines Testprogramms "Hallo World"

Los geht es !!!

Aufgabe

Schreibe folgendes Programm

Programm 1.a

```
Config Portb.5 = Output

Pinb.5 = 1
END
```

Was beobachtest du auf dem Seeeduino?

Erweitere das Programm um zusätzliche Befehle

Programm 1.b

```
Config Portb.5 = Output

Pinb.5 = 0
Waitms 2000
Pinb.5 = 1

END
```

Was beobachtest du nun auf dem Seeeduino?

Was könnte der Befehl **Waitms** bewirken und welche Bedeutung hat die **Zahl** dahinter?

Erweitere das Programm und rücke die Befehle zwischen **DO** und **LOOP** mit der Tabulatortaste ein.

Programm 1.c

```
Config Portb.5 = Output

DO
  Waitms 1000
  Pinb.5 = 1
  Waitms 1000
  Pinb.5 = 0
LOOP

END
```

Was geschieht nun mit der LED auf dem Seeeduino?

Was bewirkt der Befehl **DO** und **LOOP** und wann kann man ihn beim Programmieren einsetzen?

Jetzt kannst Du selber ein wenig experimentieren, indem Du in den Programmzeilen neue Werte ausprobierst.

Was hast Du rausgefunden?

Befehlsliste in Bascom

Um einen Microprozessor zu programmieren, hat Bascom mittlerweile über 80 Befehle. Natürlich können hier nur eine Auswahl von Befehlen behandelt werden. Sämtliche Befehle (statements) sind aber in der Hilfe von Bascom zu finden und werden in englischer Sprache erklärt.

Zu finden unter: Help ➔ Index ➔ Index (hier gesuchten Befehl eingeben)

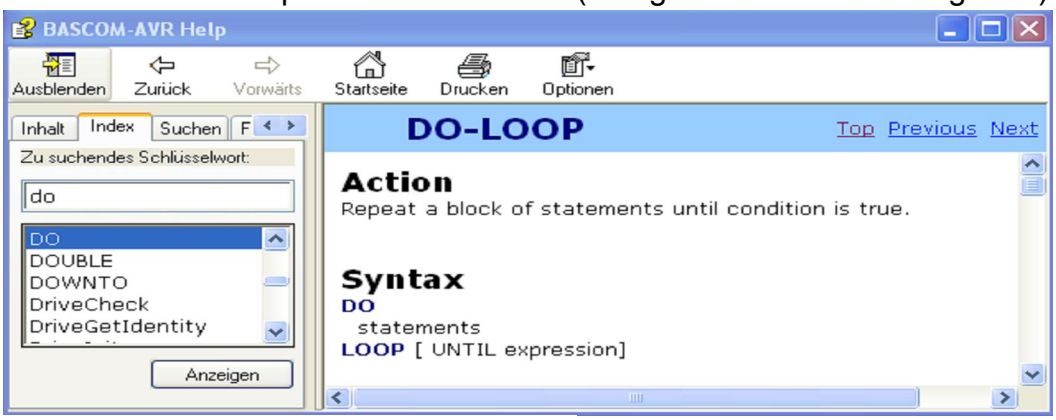


Abb. 3.1: BASCOM Befehlsliste

Es ist sinnvoll sich für jeden Befehl (statement) sein eigene Befehlsliste anzulegen, um sich so sein eigenes Nachschlagewerk zu konstruieren.

Als Beispiel wird dieses hier einmal an zwei Befehlen, die Ihr schon kennen gelernt habt gezeigt.

DO-LOOP

Action

Wiederholung von Befehlen in einer Schleife angefangen von DO bis LOOP

Syntax

```
DO
   statement 1
   statement 2
   …
LOOP
```

Abb. 3.2: Befehlssyntax Do-Loop

Waitms x

Action

Warten von x Millisekunden bis der nächste Befehl ausgeführt wird.

Syntax

Waitms 1000

Verwandte Befehle

Wait, **Waitus**, **Delay**

Abb. 3.3: Befehlssyntax Waitms

Die Anschlüsse des Mikroprozessors

Ein Mikroprozessor hat Anschlüsse, mit denen er entweder

- ➔ Dinge wie eine LED **steuern** oder
- ➔ Informationen wie z.B. von einem Temperatursensor **empfangen** kann.

Einen einzelnen Anschluss nennt man bei einem Mikroprozessor **Pin**. In der Abbildung rechts erkennst Du Bezeichnungen wie PB0 bis PB5, PC0 bis PC5 und PD0 bis PD7. Der zweite Buchstabe bei der Bezeichnung (B,C oder D) gibt an, welchem **Port** der Anschluss angehört.

- ➔ Bei einem Port werden nämlich mehrer Pins zusammengefasst und durchnummeriert.

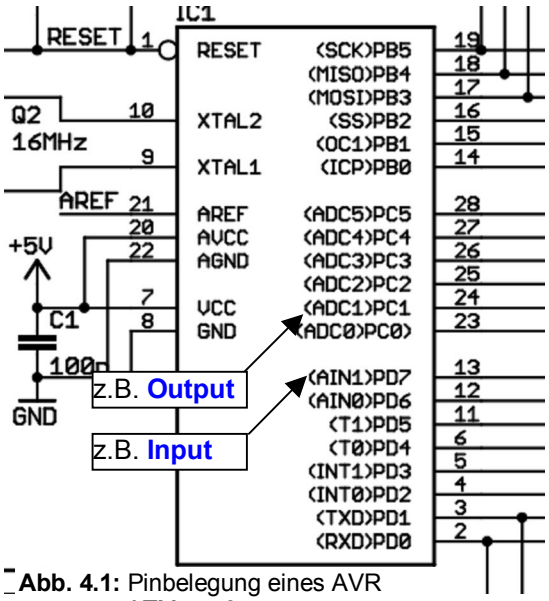


Abb. 4.1: Pinbelegung eines AVR ATMega 8

Je nachdem, ob ein ganzer Port oder nur ein Pin Eingang oder Ausgang sein soll, muss er im Programm vorher festgelegt werden.

Config Portx.y **=** [**Output**] oder [**Input**]

Mit dem Befehl **Config** wird bei der Programmierung immer etwas festgelegt (konfiguriert). In diesem Fall wird dem Prozessor mitgeteilt, ob einer seiner Anschlüsse ein **Output** (Ausgang) oder ein **Input** (Eingang) ist.
Der Anschlussname ist immer eine Adresse in der Form Pin**x.y**, wobei

- ➔ **x** den Portnamen angibt (B,C oder D) und
- ➔ **y** die Zugehörige Nummer des Pin (1,2,3 ...)

Beispiel:

```
Config Portc.0 = Output      ' Pinc.0 ist ein Output
Config Portd.6 = Input       ' Pind.6 ist ein Output
```

Abb. 4.2: Konfiguration eines Ports

Befehle:

Im BASCOM-Programmeditor werden **Befehle** immer automatisch **blau** markiert.

Kommentare:

Neben dem eigentlichen Programm, bei dem man sich ganz genau an die Schreibweise halten muss, damit es vom Prozessor "verstanden" wird, kann man auch eigene Kommentare einfügen. Diese müssen aber durch ein " ' " vor dem Kommentar gekennzeichnet werden.

Schnittstelle zur Außenwelt

Die Ports des Mikrokontroller sind die Schnittstelle zur Außenwelt und werden auf dem Seeeduino an drei Steckerleisten herausgeführt (s. Abb. 4.3). Zusätzlich gibt es noch eine Steckerleiste für die Spannungs-versorgung (Power).

Die Entwickler dieses Mikrokontroller Projekts (Arduino Projekt) haben genau festgelegt, an welcher Stelle sich die Steckerleisten befinden, so dass auf die Steckerleisten weitere Platinen (Shields) aufgesteckt werden können.

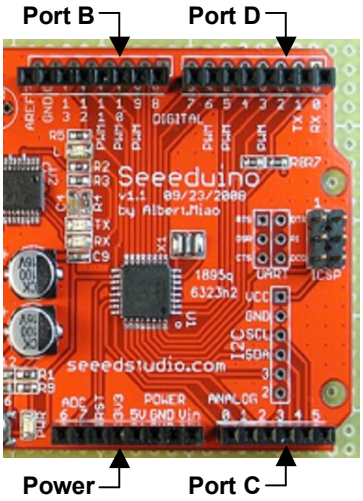


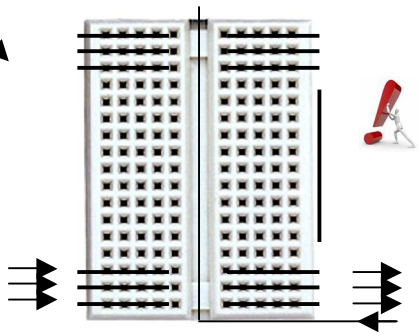
Abb. 4.3: Ports des Seeeduino

Abb. 4.4: Prototype-Shield, Sensor-Shield, LCD-Shield

Beim Aufstecken der Shields muss man unbedingt darauf achten, dass die Stecker nicht verbogen werden!

Das Prototype-Shield

Für eigenen Entwicklungen ist das sogenannte Prototype-Shield sehr interessant. Hier kann man eigene Schaltungen auf einem kleinen Steckbrett entwickeln und über die Ports mit Hilfe von kleinen Kabeln direkt mit dem Mikrokontroller verbinden.



Damit das Verbinden der elektronischen Bauelemente sehr einfach geht, sind jeweils die fünf Buchsen in einer Zeile - sowohl auf der rechten als auch auf der linken Seite - miteinander verbunden.

Abb. 4.5: Verdrahtung des Steckbrettes

Jede aufgebaute Schaltung wird vor der Inbetriebnahme vom Lehrer überprüft!

LEDs als Statusanzeige

Leuchtdioden (s. Abb. 4.6) werden in vielen Geräten der Unterhaltungselektronik eingesetzt. Ihr Aufleuchten zeigt dabei den Betrieb oder die Betriebsbereitschaft des jeweiligen Gerätes an. Der erste Teil ihres Namens ist damit klar : sie "leuchten" - geben Licht ab.

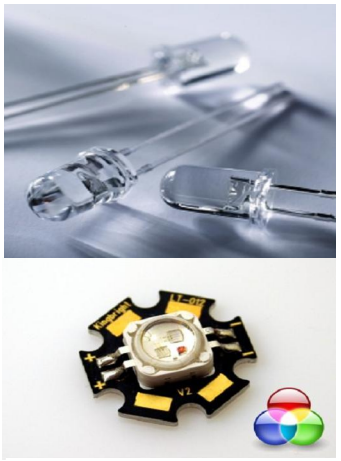


Abb. 4.6: Versch. LED -Typen

Leuchtdioden sind ebenfalls Dioden wie andere Halbleiter- oder Röhrendioden auch. Dioden sind so etwas wie ein "Ventil" für Elektronen: so wie das Ventil an einem Fahrradreifen, die Luftmoleküle nur hinein- aber nicht wieder herauslassen, lassen auch Dioden die Elektronen nur in einer Richtung passieren.

Aus diesem Grund muss man den Aufbau einer LED kennen, um diese richtig in die Schaltung einzusetzen. In Abb. 4.7. ist eine typische Standard-LED mit ihren Anschlussmerkmalen dargestellt.

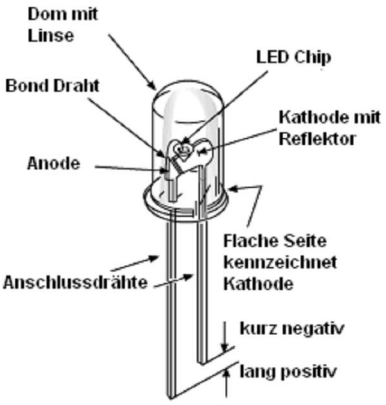


Abb. 4.7: Aufbau einer LED

Der **lange** Anschlussdraht ist die **Anode** und zeigt zum **Port** des Mikrokontrollers.
Der **kurze** Anschlussdraht ist die **Kathode** und wird mit **GND** verbunden.

LEDs werden immer mit einem Vorwiderstand betrieben (s. Abb. 4.8), da die Versorgungsspannung fast immer über der LED-Spannung liegt, was ohne Vorwiderstand zur Folge hat, dass ein zu hoher Strom fließt, der die LED zerstören würde.

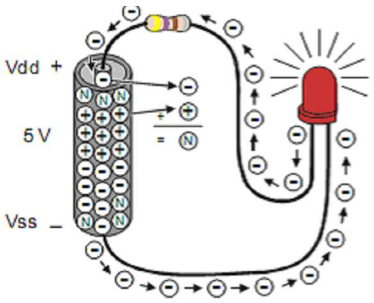


Abb. 4.8: LED mit Vorwiderstand an einer Batterie

LEDs dürfen **nie ohne Widerstand** betrieben werden, da sonst die **LED zerstört** werden könnte!

Als Vorwiderstand für eine **LED an einem Mikrokontroller** (s. Abb. 4.9) wählt man einen Wert zwischen 100 Ω und 470 Ω, um den **Mikrokontroller nicht** zu **zerstören**.

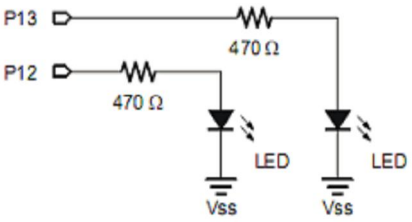


Abb. 4.9: LED an einem Port (Schaltplan)

Aufgabe

Baue die LED Schaltung nach Vorgabe des Fotos auf!

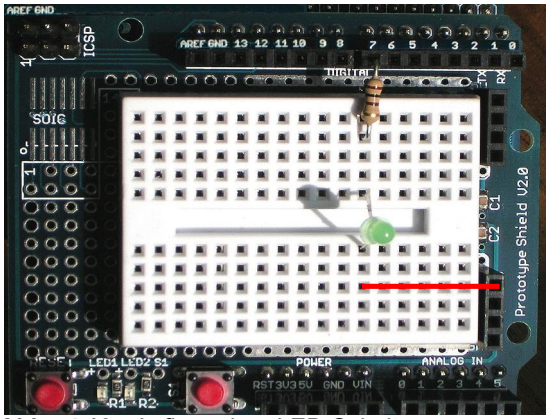


Abb. 4.10: Aufbau einer LED-Schaltung

Material:

- 1x LED
- 1x Widerstand 100 Ω
- 1x Kabel

Verwendete Pins:

Port D: D7

LEDs dürfen **nie ohne Widerstand** betrieben werden, da sonst der **Mikrokontroller zerstört** wird!

Schreibe das folgende Programm!

Programm 2.a

```
Config Portd.7 = Output

Portd.7 = 1
END
```

Was beobachtest du?

Schließe eine weiteren LED an einen anderen Pin an und erweitere das Programm so, dass beide LEDs leuchten!

Programm 2.b

```
Config Pind.7 = Output
...
Portd.7 = 1
...

END
```

Was beobachtest du?

Erweitere das Programm so, dass die LEDs abwechselnd blinken!

Programm 2.c

```
Config Portd.7 = Output
...

DO
  ...
LOOP

END
```

Zähle auf, welche Befehle du verwendet hast?

Bargraph Anzeige (Balkenanzeige)

Eine Bargraph-Anzeige ist eine Anzeigeinstrument, bei dem sich die Länge eines leuchtenden Balkens (engl.: *bar* für Stange) mit der Signalgröße verändert. In der Tontechnik z.B. benutzt man eine Bargraph-Anzeige, um den Lautstärkepegel von Boxen anzuzeigen.

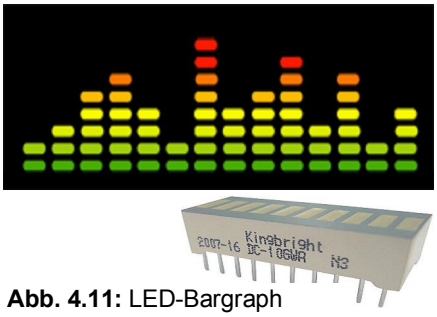


Abb. 4.11: LED-Bargraph

Aufgabe

Baue die Bargraph Schaltung nach Vorgabe des Fotos auf!!

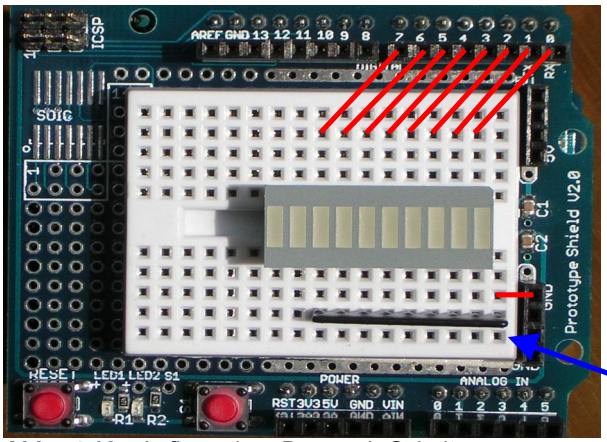


Abb. 4.12: Aufbau einer Bargraph-Schaltung

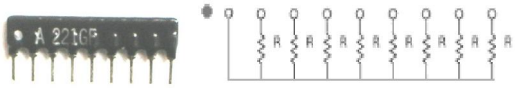
Material:

- 1x Bargraph Anzeige
- 1x Widerstands-Array 100 Ω
- 9x Kabel

Verwendete Pins:

Port D: D0, D1, D2, D3, D4, D5, , D6, D7

Der weiße Punkt muss auf die rechte Seite!



Schreibe das folgende Programm!

Programm 3.a

```
Config portd = Output

Do
  Portd = &B00000001
  Waitms 500
  Portd = &B00000010
  Waitms 500
  Portd = &B00000100
  Waitms 500
LOOP
END
```

Was beobachtest du?

Was bewirkt der Befehl Portd = &B00000001 ? Trage den Befehl in deine Befehlsliste ein!

Kann man auch zwei LED gleichzeitig ansteuern? (Erklärung)

Erweitere das Programm so, dass die LED einmal von rechts nach links und wieder zurück läuft!

Programm 3.b

```
Config portd = Output

Do
...
LOOP
END
```

Fallen dir noch andere Muster ein, die du programmieren kannst ?

Schau in der Hilfe nach, was die Befehle **shift** und **rotate** bewirken und schreibe dazu ein Programm!

7-Segment-Anzeige

Eine Segmentanzeige kann durch eine elektronische Ansteuerung aus einzelnen oder mehreren Segmenten („Strichen“) Buchstaben, Zahlen oder Zeichen darstellen.
Die bekannteste ist die 7-Segment-Anzeige, welche z.B. in Digitaluhren zum Einsatz kommt und alle Ziffern von 0-9 aus bis zu sieben einzelnen Segmenten zusammensetzt.

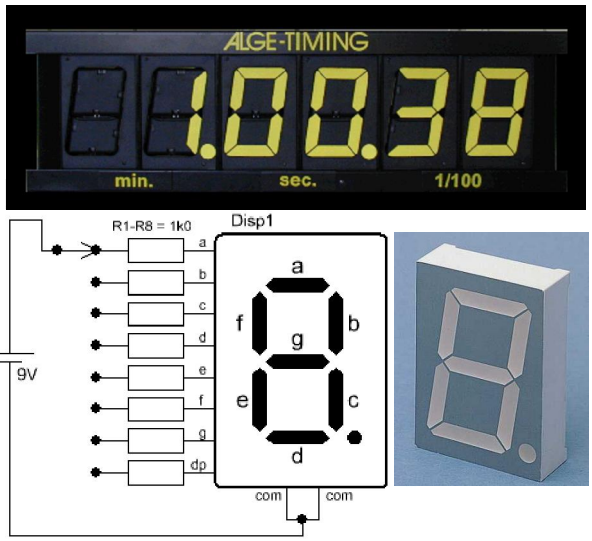


Abb. 4.13: 7Segment-Anzeige

Aufgabe

Erstelle eine Tabelle, welche Segmente für jede Zahl von 0 bis 9 angesteuert werden müssen!

Zahl	Benötigte Segmente	Programmierung
0	a, b, c, d, e, f	Portd = &B00111111
1	b, c	Portd = &B00000110
2	a, b, d, e, g	Portd = &B01011011
3	a, b, c, d, g	Portd = &B01001111
4	b, c, f, g,	Portd = &B01100110
5	a, c, d, e, f	Portd = &B01101101
6	c, d, e, f, g,	Portd = &B01111100
7	a, b, c,	Portd = &B00000111
8	a, b, c, d, e, f, g	Portd = &B01111111
9	a, b, c, d, f, g	Portd = &B01101111

Kann man auch Buchstaben darstellen? Wenn ja, welche?

Aufgabe

Baue die 7-Segment-Schaltung nach Vorgabe des Fotos auf!

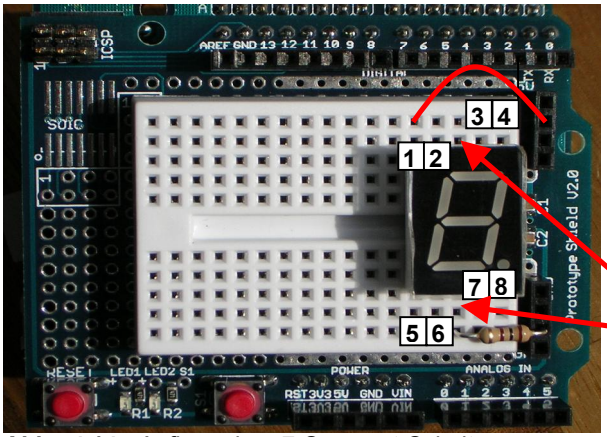


Abb. 4.14: Aufbau einer 7-Segment-Schaltung

Material:

- 1x 7-Segment-Anzeige
- 1x Widerstand 100 Ω
- 1x Kabel

Verwendete Pins:

Port D: D0, D1, D2, D3, D4, D5, D6, D7

Hier darf auf gar keinen Fall eine Leitung angeschlossen werden, da es sonst einen Kurzschluss gibt und das Board zerstört wird!

Leider steht uns für diese Anzeige kein Datenblatt zur Verfügung, so dass man erst einmal selber herausfinden muss, an welchen Anschlüssen die jeweiligen Segmente angesteuert werden.

Nehme das Kabel wie in der Abbildung oben und probiere nacheinander die nächsten Anschlüsse (2, 3, 4, ..., 8) aus und notiere die Belegung (a, b, ... , g) in der Abbildung unten!

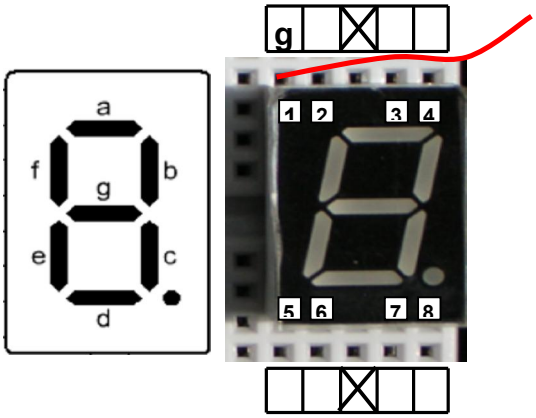


Abb. 4.15: Überprüfung der Anschlussbelegung der 7-Segment-Anzeige

Verbinde nun die Anschlüsse der 7-Segmentanzeige mit den Ports des Mikrocontrollers nach der folgenden Zuordnung **a ➔ D0 ; b ➔ D1 ; c ➔ D2**

Schreibe ein Countdown Programm!

Programm 4.a

```
Config portd = Output

  Portd = Portd = &B01101111
  ...
```

Taster als Eingabegerät

Abb. 4.16: Verschiedene Tastertypen

Bis jetzt haben wir uns ausschließlich mit der Ausgabe an Ports beschäftigt. Ebenso wichtig ist aber auch, den Mikrokontroller von außen zu steuern oder Informationen zur Verarbeitung bereitzustellen.
In der einfachsten Form geschieht dieses mit Hilfe eines Tasters.

Der Mikrokontroller kann an seinen Ports zwischen zwei unterschiedliche Zuständen unterscheiden:

- ➔ **High** (Spannung 5V) und
- ➔ **Low** (GND=0V)

High wird auch oft als "1" und low als "0" bezeichnet.

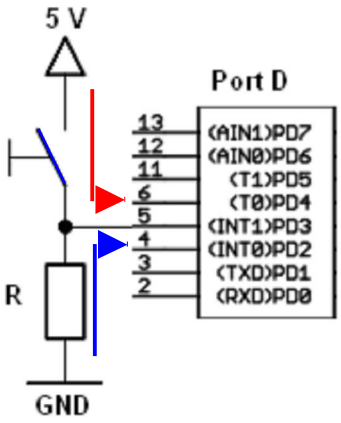


Abb. 4.17: Tasterschaltung (Schaltplan)

In der Abb. 4.16 kann man sehen, was passiert wenn der Taster gedrückt wird.
Im Ausgangszustand ist der Taster geöffnet und am Port PD3 liegt über einen Widerstand GND = 0V (low) an.
Wenn der Taster geschlossen ist, liegt am Port eine Spannung von 5V, also ein High-Signal an.

Was passiert, wenn man den Taster wieder loslässt?
Richtig am Port liegt wieder das Low-Signal an.

Mit diesem Wissen ist es nun möglich, mit einem Taster ein Programm zu steuern.
Wie das genau geht, werden wir anhand eines kleinen Programms sehen.

Baue die Tasterschaltung nach Vorgabe des Fotos auf!

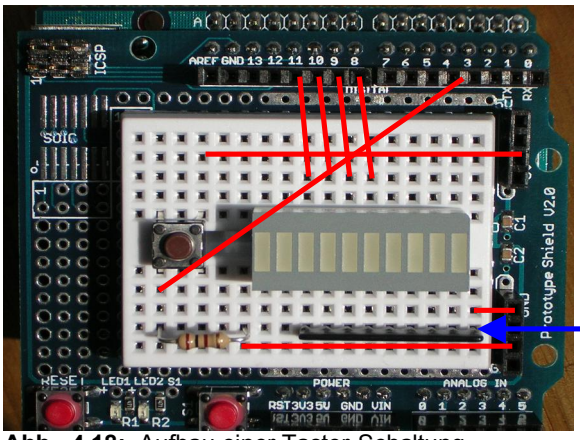


Abb. 4.18: Aufbau einer Taster-Schaltung

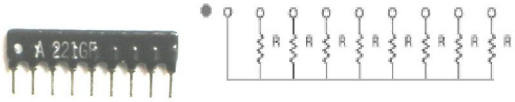
Material:

- 1x Bargraph-Anzeige
- 1x Widerstands-Array 100 Ω
- 7x Kabel

Verwendete Pins:

Port B: B0, B1, B2, B3
Port D: D3

Der weiße Punkt muss auf die rechte Seite!



Durch die Tasterschaltung wird jetzt ein High (1) oder ein Low (0) erzeugt, aber nun muss man dem Mikrokontroller beibringen, was er nun mit den Signalen machen soll. Auf jeden Fall kennt er ganz bestimmte Befehle (statments) für die Ports PD2 und PD3, die man **Interrupts** nennt, um mit dem Taster ein Programm zu steuern.

INTERRUPT aktivieren

Action

Ein INTERRUPT unterbricht sofort das Hauptprogramm und führt ein Unterprogramm aus. Dabei wird der Interrupt durch ein Signal entweder am Port PD2 (INT0) oder PD3 (INT1) z.B. durch einen Taster ausgelöst.
Um einen INTERRUPT festzulegen (konfigurieren) und zu aktivieren, sind mehrere Befehle notwendig.

Syntax

Enable Interrupts
➔ hier wird die Interruptfunktion aktiviert

Config IntX = Status
➔ hier wird der Trigger (Auslöser) festgelegt

X = 0 für **INT0** z.B. Taster an Port PD2 oder
X = 1 für **INT1** wenn Taster an Port PD3

Status = **Falling** für Signalwechsel von 1 ➔ 0
Status = **Rising** für Signalwechsel von 0 ➔ 1
Status = **Change** für Signalwechsel von 0 ➔ 1 oder 1 ➔ 0

Enable IntX
➔ hier wird der jeweilige Interrupt **INT0** oder **INT1 aktiviert**

X = 0 für **INT0**
X = 1 für **INT1**

Mit **Disable** kann man den jeweiligen Interrupt wieder abschalten

On Int0 label **Nosave**
➔ gibt an, welches Unterprogramm (Name hier Label) bei einem Interrupt aufgerufen werden soll.

Beispiel

```
' CONFIG Interrupt ----------------------------------------------------------------
  Enable Interrupts
  Config Int1  = rising
  Enable Int1
  On Int1 Susi Nosave
' Hauptprogramm -------------------------------------------------------------------
  Do
     Pinb.1 = 0
  Loop
  End
' Unterprogramm -------------------------------------------------------------------
  Susi:
     Pinb.1 = 1
  Return                    ' durch Return wird wieder ins Hauptprogramm gesprungen
```

Schreibe das folgende Programm!

Programm 5.a

```
Config Portb = Output
'CONFIG Interrupts-----------------------------------------------
Enable Interrupts          'Aktiviert die Interruptfunktion
Config Int1 = Falling      'Definiert den Trigger
Enable Int1                'Aktiviert den Interrupts Int0
On Int1 Susi Nosave        'Springt zum Unterprogramm
                           ' Susi, wenn Taste Gedrückt
'----------------------------------------------------------------------
'Hauptprogramm
Do
    Pinb.0 = 1
Loop
End                        'Ende Hauptprogramm Label
'----------------------------------------------------------------------
'Unterprogramm
Susi:                      'Beginn des Unterprogramms Susi
  Pinb.0 = 0
Return                     'Rücksprung ins Hauptprogramm
```

Was beobachtest du?

Erkläre, was das Haupt und das Unter- programm macht!

Was muss man ändern, dass die LED bei Tastendruck einge- schaltet wird

Was muss im Programm geändert werden, wenn der Taster an PD2 ange- schlossen wird?

Dieses Programm ist total langweilig, denn so ein Aufstand für nur eine einzige Lampe anzuschalten!?!

Hast Du vielleicht eine Idee, wie man ein Programm schreiben kann, womit man seine Reaktionen testen kann? Ich mache mal den Anfang!

Programm 5.b

```
'CONFIG Interrupts-----------------------------------------------
Enable Interrupts          'Aktiviert die Interruptfunktion
Config Int1 = Falling      'Definiert den Trigger
Enable Int1                'Aktiviert den Interrupts Int0
On Int1 Reaktion Nosav     'Springt zum Unterprogramm
                           ' Rektion, wenn Taste gedrückt
'----------------------------------------------------------------------
'Hauptprogramm
Do
  Portd = &B00000001
  Waitms 100
  Portd = &B00000010
  ...
Loop
End                        'Ende Hauptprogramm Label
'----------------------------------------------------------------------
'Unterprogramm
Reaktion:                  'Beginn des Unterprogramms
...
```

Man kann auch mal dieses Programm ausprobieren:

Programm 5.c

```
'CONFIG Interrupts----------------------------------------------------
Enable Interrupts          'Aktiviert die Interruptfunktion
Config Int1 = Falling      'Definiert den Trigger
Enable Int1                'Aktiviert den Interrupts Int0
On Int1 Susi Nosave        'Springt zum Unterprogramm
                           ' Susi, wenn Taste Gedrückt
'-------------------------------------------------------------------------------
'Hauptprogramm
Do
    Pinb.0 = 1
Loop
End                        'Ende Hauptprogramm Label
'-------------------------------------------------------------------------------
'Unterprogramm
Susi:                      'Beginn des Unterprogramms Susi
  Pinb.0 = 0
Return                     'Rücksprung ins Hauptprogramm
```

Was beobachtest du?

Was verursacht der Befehl shift

Wie rechnet eigentlich ein Mikrocomputer?

Bis jetzt haben wir gesehen, dass man mit dem Mikrocomputer über Ports LEDs, 7-Segment-anzeigen oder eine Bargraphanzeige ansteuern können. Das funktioniert immer über die Information Port oder Pin an (1) oder aus (0). Bei der Eingabe über Taster ist das genau so. Viele Leute sagen ein Computer wäre eigentlich dumm, denn er kann nur 1 oder 0.

Aber wie rechnet er dann? **Mit Einsen und Nullen ...?**

Genau so geht es! Die Zahlen, die der Computer kennt, sehen genau so aus, wie wir es schon einmal bei der Ansteuerung mit dem Portbefehl gesehen haben.

➔ Portd = &B00000101

Für den Computer sehen unsere Dezimalzahlen wie in Tab.5.1 aus und heißen Binärzahlen.

Tab. 5.1

Dezimalzahl	?	?	Binärzahl					
			32	16	8	4	2	1
1								1
2							1	0
3							1	1
? 4								
5						1	0	1
? 7								
8					1	0	0	0
12					1	1	0	0
? 15								
17				1	0	0	0	1
55			1	1	0	1	1	1
? 103								

Eine Knobelaufgabe: Wie werden die Binär-zahlen gebildet?

Wie wird die Reihe nach 32 fortgesetzt?

Wie heißen die fehlenden Binärzahlen 4, 7, 15 und 103?

Der Computer rechnet mit Eins und Null. Das ist ziemlich einfach und geht super schnell.

Rechne doch einfach einmal, wie ein Computer und wandel zwei Zahlen in Binärzahlen um (z.B. 7 und 11)

- Addiere beide Binärzahlen schriftlich
- Wandele das Ergebnis wieder in eine Dezimalzahl um!
- Stimmt das Ergebnis?

Was mit der Addition geht, sollte auch bei der Subtraktion funktionieren.

... und was ist mit der Multiplikation?

Das LCD-Display

Auf einem LCD-Display können durch die Ansteuerung mit Hilfe des Mikroprozessors Buchstaben, Zahlen oder Zeichen dargestellt werden.

Dadurch können sehr einfach Anzeigen z.B. für Messgeräte oder eine Statusanzeige bei Waschmaschinen realisiert werden.

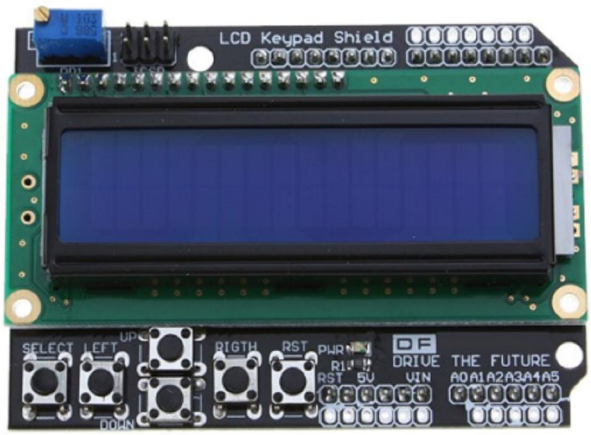


Abb. 5.1: LCD-Display mit Tastatur

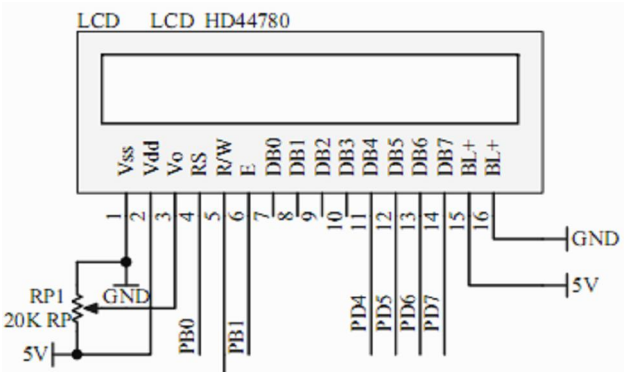


Abb. 5.2: Schaltplan eines LCD-Display HD44780 auf dem Arduino LCD-Shield

Ein typisches LCD Display ist in Abb.5.2 dargestellt. Vier Anschlüsse (DB4 – DB7) werden für die Datenübertragung zum Display benötigt.

Zusätzlich sind zwei weitere Anschlüsse (RS und E) notwendig, um das Display zu aktivieren, so dass die Datenleitungen zusätzlich auch für andere Anwendung zur Verfügung stehen.

Da die Entwickler ihre elektronische Schaltungen individuell entwerfen, ist es notwendig alle Ports und das Displays vorher in Bascom zu konfigurieren.

Die Einstellung findet man unter

➔ Options ➔ Compiler ➔ LCD

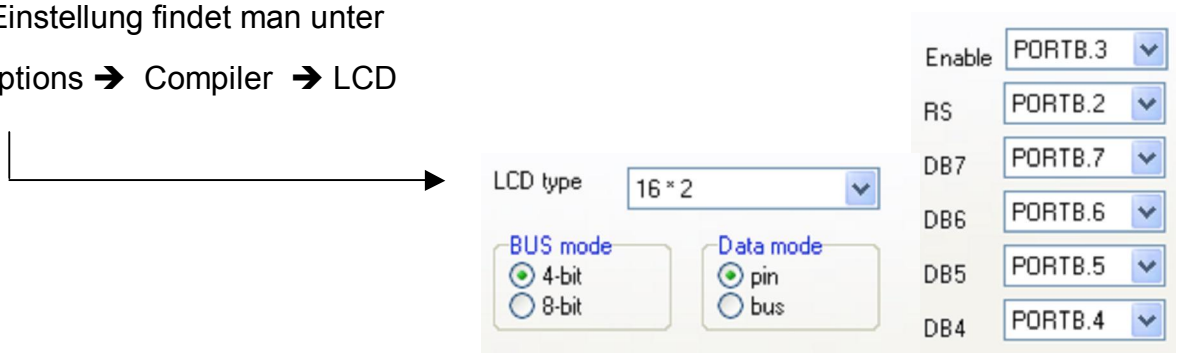


Abb. 5.2: Einstellungen in Bascom für ein Arduino LCD-Shield

Schreibe das folgende Programm!

Programm 6.a

```
Cls

Locate 1 , 1
Lcd "Hallo"
Locate 2 , 1
Lcd "World"

End
```

Trage in die Befehlsliste ein, was die verwendeten Befehle bewirken!

Gibt es noch weitere Befehle, die für die Programmierung von Displays nützlich sein können?
Probiere die einzelnen Befehle aus und trage sie in die Befehlsliste ein!

Schreibe ein Programm für den Abspann eines Filmes wie z.B bei Star Wars!

Einzelne Zeichen können in Bascom auch selbst definiert und in den Speicher des Displays übertragen werden.

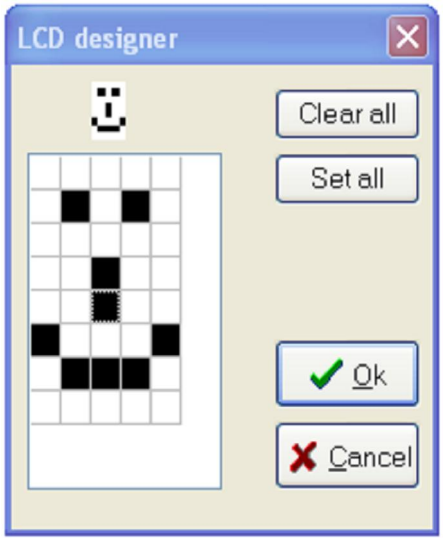


Abb. 5.3: Smiley entworfen mit Bascom LCD Designer

Hierzu wird der LCD Designer benötigt

Tools ➔ LCD designer

Hier werden einzelne Pixels (Punkte) in einer 5x8 Matrix definiert und in einen von 8 Speicherplätzen im Display übertragen.

Hierzu wird dann in unserem Beispiel (Smiley) folgende Programmzeile automatisch in das Programm übertragen.

```
Deflcdchar 1,32,10,32,4,4,17,14,32
```

Nummer des Speicherplatzes

Definiere selber unterschiedliche Zeichen und lade sie in dein Programm, um sie im Abstand von 5 Sekunden hintereinander darstellen zu können!

Informiere dich im Internet über den **American Standard Code for Information Interchange?** Wozu wird er benötigt?

Wozu werden die Befehle **chr()** und **asc()** verwendet?
Probiere sie aus und übertrage ihre Funktion anschließend in die Befehlsliste.

Schreibe ein Count-Down-Programm von 1 – 100!
Kannst du dieses Programm in 2 Minuten schreiben?

Variablen, Schleifen und Bedingungen

Im letzten Beispiel haben wir gesehen, das ein Countdown-Programm, so wie wir es bis jetzt kennen gelernt haben, viele Programmschritte benötigt.
Ein Mikrokontroller kann aber auch einfacher programmiert werden, da er bestimmte hard- und softwaretechnische Eigenschaften besitzt, um das Programm auch kürzer zu schreiben.

Dazu zählen

- ➔ Variablen
- ➔ Bedingungen
- ➔ Schleifen

Abb. 5.4: Display mit Jahres Countdown

Variablen

Variablen sind einzelne Platzhalter für verschiedene Zahlen, Buchstaben oder Zeichen. Hardwaretechnisch handelt es sich hier um einen Speicherplatz, der unter einem bestimmten **Namen** verschiedene **Werte** mit einer ganz bestimmten **Größe** abspeichern kann kann.
Der Speicher ist bei Computern immer begrenzt und kostete gerade in der Vergangenheit eine Menge Geld, so dass mit Speicher sehr ressourcenschonend umgegangen werden muss. Deshalb kann man sich in einem Speicherchip oder direkt im Mikrokontroller bestimmte Speicherbereiche je nach benötigter Größe für ein Programm reservieren. Dieses muss man im Programm, bevor man den Speicher im Programmablauf benutzt, jeweils festlegen (deklarieren).
Hierzu gibt es in Bascom, wie auch in anderen Programmiersprachen, verschiedene Arten von Variablen, die sich in ihrer Dimension (Größe) unterscheiden (Tab. 6.1).

Tab. 6.1: Art und Dimension von Variablentypen

Name	Zahlengröße (Dimension)
Bit	0 oder 1
Byte	0 bis 255
Integer	- 32767 bis 32767
Word	0 bis 65535
String	bis 254 bytes

Die Befehle für unterschiedliche Variablen in Bascom zur Deklaration lauten:

Tab. 6.2: Deklaration von Variablentypen

```
'-----------------------------------------------------------
'Declaration

Dim Hubert As Bit
Dim Klara As Byte
Dim Walpurga As Integer
Dim Winibald As Word
Dim Felicitas as  String * 8
'-----------------------------------------------------------
...
```

Trage die Anweisung zur Deklaration von Variablen in die Befehlsliste ein!

Schleifen mit Abbruchbedingungen

Bis jetzt kennen wir als Schleife den Do-Loop Befehl. Der Nachteil einer solchen Schleife ist, dass diese sie nie endet.

Abb. 5.5: Endlosschleife

Aus diesem Grund gibt es alternativ Schleifen die solange „laufen“ bis eine bestimmte *Abbruchbedingung* erfüllt ist. Dabei gibt es im Wesentlichen zwei Arten von Schleifen, die sehr nützlich sind.

For ➔ Next

Führt eine Block von Staments für eine festgelegte Anzahl von Wiederholungen aus

Programm 7.a

```
'-----------------------------------------------------------
'Declaration

Dim J As Bit
'-----------------------------------------------------------
‘Main

For J = 1 To 100
   LCD J
   Waitms 200
Next J

END
```

Schreibe das folgende Programm!

Was kann beobachtet werden?

Trage den FOR-NEXT Anweisung in die Befehlsliste ein!

Wie kann der Befehl durch das Step Statement erweitert werden und was ändert sich hierdurch?

While ➔ Wend

Führt verschiedene Statements in einer Schleife aus, solange eine Bedingung erfüllt ist.

Programm 7.b

```
'-----------------------------------------------------------
'Declaration

Dim A As Bit
'-----------------------------------------------------------
‘Main

While A < 100
   LCD A
   Incr A
   Waitms 200
Wend A

END
```

Schreibe das folgende Programm!

Was kann beobachtet werden?

Trage doe While-Wend Anweisung in die Befehlsliste ein!

Was bewirkt der Incr Befehl in der While-Schleife und wie könnte man den Decr Befehl einsetzen?

Schreibe nun das Countdown-Programm neu!
Maximal sind 12 Programmzeilen erlaubt!

Verzweigte Schleifen

Abb. 5.6: Designerdigitaluhr

Schleifen mit Abbruchbedingungen können sowohl hinter-einander angeordnet, als auch ineinander geschachtelt werden.

Damit ist es möglich mit Hilfe eines Mikrocontrollers auch eine Digitaluhr zu programmieren. Hierzu müssen die einzelnen Schleifen für Stunden, Minuten und Sekunden miteinander kombiniert werden.

Die Anzeige soll mit Hilfe eines LCD-Displays geschehen, wobei dieser Programmteil in einem Unterprogramm realisiert werden soll.

Programm 7.C

```
'-------------------------------------------------------
'Declaration
Dim h As …                  'Dim Stunde
…
'-------------------------------------------------------
‘Programm Digitaluhr
For h = 0 To …
  For  …
    For  …
      Gosub Anzeige:   ‘Sprung zum Unterprogramm Anzeige
    Next s
  Next m
Next h
END

Anzeige:                    ‘Anfang Unterprogramm Anzeige
…
Return                      ‘Zurück zum Hauptprogramm
```

Schreibe ein Programm für eine Digitaluhr, welche Stunden, Minuten und Sekunden anzeigt!

Informiere Dich in der Hilfe über den Aufruf von Unterprogrammen und trage die Gosub Anweisung in die Befehlsliste ein!

Sprung in ein Unterprogramm

Ein Unterprogramm oder eine Subroutine ist ein Teil eines Programms, die aus dem Hauptprogramm oder anderen Programmteilen heraus aufgerufen werden kann und nach Abschluss der Abarbeitung jeweils in das aufrufende Programm wieder zurückkehrt.
Ein Grund der Aufteilung ist die mehrfache Verwendung der gleichen Befehlsfolge, damit Einsparung von Speicherplatz und die Vermeidung von Programmwiederholungen.
Zudem werden Programme übersichtlicher, wenn man in sich abgeschlossene Programmteile programmiert und in das Hauptprogramm einbindet.

Conditional Instruction

Neben den Abbruchbedingungen, welche direkt zum Schleifenbefehl gehören, gibt es auch Verzweigungen, die direkt in das Programm eingebaut werden können. Solche Conditional Instructions“ benutzt man zum Beispiel, um unter bestimmten Bedingungen bestimmte Anweisungen auszuführen oder von einem Hauptprogramm in ein Unterprogramm zu springen.

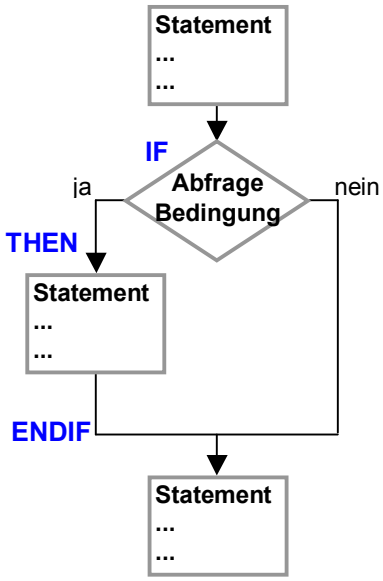


Abb. 5.7: IF-Then Anweisung

IF ➔ THEN

Führt ein oder mehrere Statements so oft aus, so lange die Bedingung erfüllt ist.

Programm 7.b

```
'-----------------------------------------------------------
'Declaration

Dim A As Bit
'-----------------------------------------------------------
‘Main

For A = 0 To 10
Lcd A
Waitms 100
  If A < 4 Then
     Lcd "Zahl ist kleiner Vier"
  End If
Next
End
```

Schreibe das folgende Programm!

Wie funktioniert die IF-Abfrage?

Trage die IF-Then Anweisung in die Befehlsliste ein!

IF ➔ THEN ➔ ELSE

ELSE führt ein oder mehrere Statements aus, wenn die Bedingung nicht erfüllt ist.

Erweitere das Beispiel oben durch ELSE so, dass bei Nicht-Erfüllung der Bedingung auf dem Display "Zahl größer Vier" ausgegeben wird (siehe Hilfe)!

Trage die IF-Then-Else-Anweisung in die Befehlsliste ein!

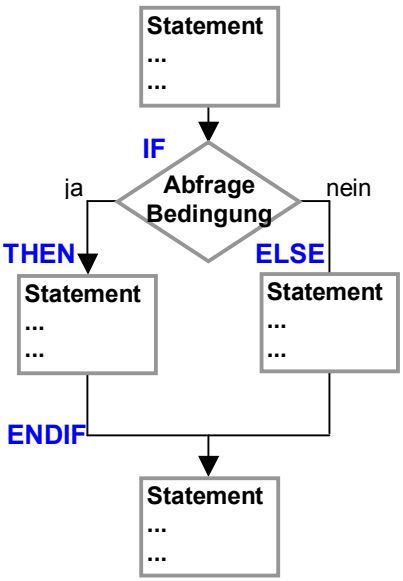
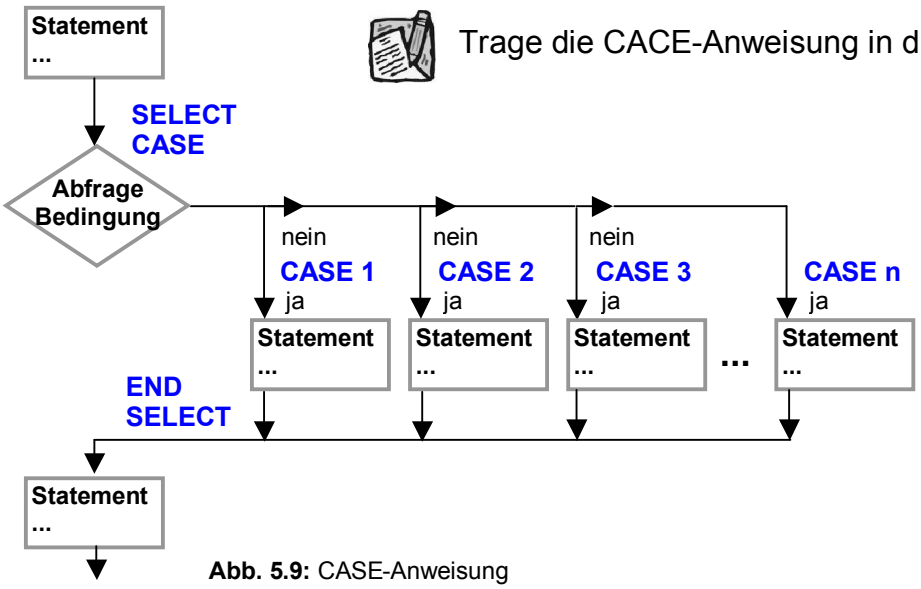


Abb. 5.8: IF-Then-Else Anweisung

Select CASE

Bei mehreren Verzweigungen wird die ELSE Anweisung sehr schnell unübersichtlich. In diesen Fällen bietet sich der CASE an. In der Hilfe gibt es ein sehr schönes Beispiel und eine Beschreibung, wie die CASE-Anweisung funktioniert.



Trage die CACE-Anweisung in die Befehlsliste ein!

Abb. 5.9: CASE-Anweisung

Programm xx

```
' Config Port
Config portd = Output
'-----------------------------------------------------
' Config VAR
Dim k as Byte
'-----------------------------------------------------
Do
   k = k+ 1
   Waitms 200
   Gosub Anzeige
Loop

Anzeige:
Select Case k
   Case 0 : LCD "Null"
   Case < 10 : LCD "kleiner 10"


End Select
```

Analog-Digital Converter (ADC)

Ein Analog-Digital-Converter (engl. ADC für Analog-to-Digital-Converter), auch Analog-Digital-Wandler oder A/D-Wandler genannt, setzt analoge Eingangssignale in digitale Daten bzw. einen Datenstrom um, der dann weiterverarbeitet oder gespeichert werden kann.

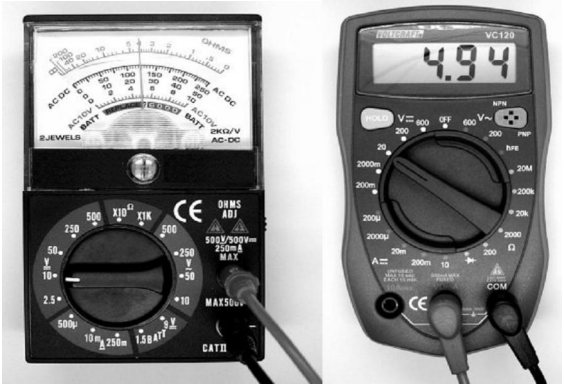


Abb. 5.1: Analog und Digitalmessgerät

Aufgabe

Baue die ADC Schaltung nach Vorgabe des Fotos auf!!

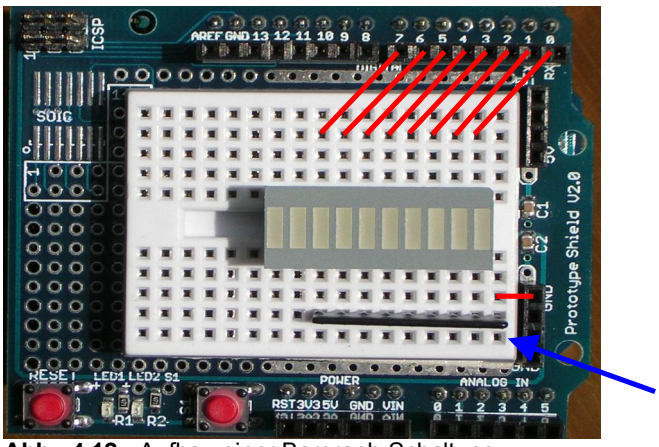


Abb. 4.12: Aufbau einer Bargraph-Schaltung

Material:

- 1x
- 1x
- 9x

Verwendete Pins:

Port:

Schreibe das folgende Programm!

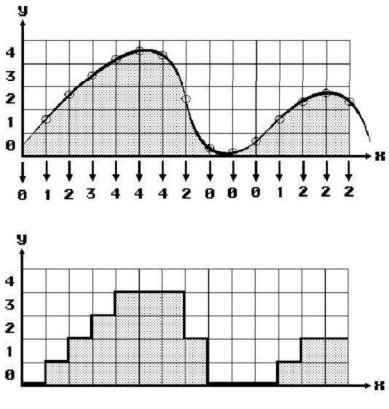
Programm 3.a

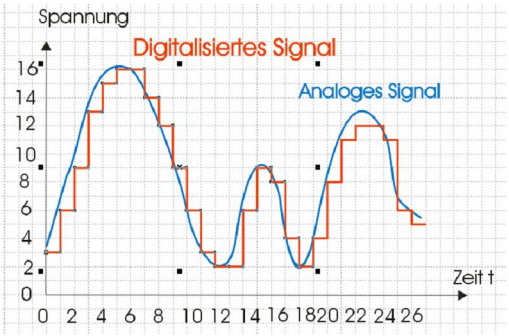
```
Config portd = Output

Do
  Portd = &B00000001
  Waitms 500
  Portd = &B00000010
  Waitms 500
  Portd = &B00000100
  Waitms 500
Loop
END
```

Was beobachtest du?

Was bewirkt der Befehl Portd = &B00000001 ? Trage den Befehl in deine Befehlsliste ein!

Kann man auch zwei LED gleichzeitig ansteuern? (Erklärung)


y
4
3
2
1
0
x
0 1 2 3 4 4 4 2 0 0 0 1 2 2 2
y
4
3
2
1
0
x


Spannung
Digitalisiertes Signal
Analoges Signal
16
14
12
10
8
6
4
2
0
Zeit t
0 2 4 6 8 10 12 14 16 18 20 22 24 26

Multiplex

In einem vorigen Projekt haben wir bereits eine einzelne 7-Segementanzeige kennengelernt. Aber damit kann man lediglich eine Ziffer ausgeben. Um z.B. die Uhrzeit oder Messerwerte auszugeben braucht man aber mehrere Ziffern. Leider gibt es da ein Problem! Man benötigt nämlich für 4 Ziffern 32 Pins (3 x 8 Pins), doch leider hat unser Mikrokontroller nur 20 Pins.
Das Zauberwort, um das Problem zu lösen heißt *Multiplexing*.
Dabei wird jede Stelle *einzeln* und *nacheinander* angezeigt. Wenn dieser Vorgang sehr schnell hintereinander abläuft, sieht das menschliche Auge alle Ziffern gleichzeitig.

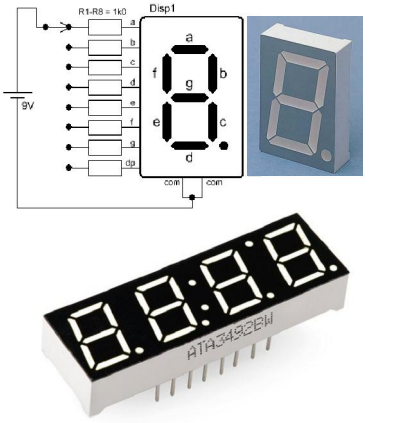


Abb. 5.1: 7-Seg-Anzeige mit 4 Digits

Wie das funktioniert, kann man sehr gut mit einer, zwei oder drei LEDs ausprobieren.

Schreibe das folgende Programm!

Programm 3.a

```
Config portd = Output
Do
  Portd.0 = 1
  Waitms 200
  Portd.0 = 0
  Waitms 200
Loop
END
```

Was beobachtest du?

Wie müssen die Zeiten geändert werden, damit das Flackern verschwindet?

Erweitere die Schaltung um eine weitere LED und schreibe ein Blinklicht-programm!

Ändere auch hier die Zeiten! Bei welchen Zeiten leuchten beide LEDs ohne zu flackern?

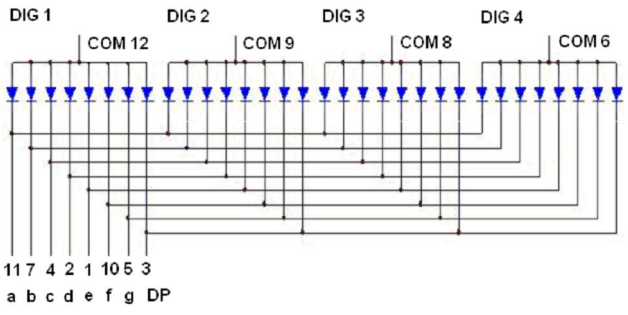


Abb. 5.1: Schaltplan 7-Seg-Anzeige mit 4 Digits

Der Schaltplan einer 7-Seg-Anzeige mit vier Digits (Stellen) zeigt, dass alle Verbindungen für die Darstellung der Zahlen (a-g) parallel geschaltet sind. Lediglich die Ansteuerung (COM) für die einzelnen Digits unterscheiden sich. Das erste Digit wird angesteuert, indem die Ansteuerung COM 12 auf „Eins“ gesetzt wird, während die anderen COM-Ansteuerungen auf „Null“ bleiben. Die einzelnen LED-Segmente leuchten, wenn die benötigten Pins (a-g) auf „Null“ gesetzt werden, also genau andersrum wie bei der einzelnen 7-Segmentanzeige.

Aufgabe

Baue die 7-Segment-Schaltung mit 4 Digits nach Vorgabe des Fotos auf!

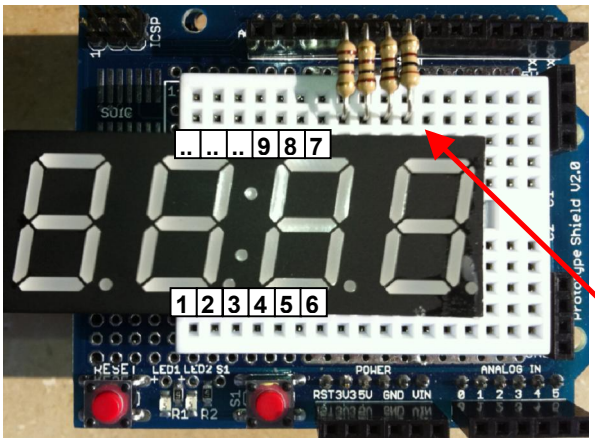


Abb. 4.14: Aufbau einer 7-Segment-Schaltung

Material:

- 1x 7-Segment-Anzeige 4 Digits
- 4x Widerstand 100 Ω
- 12x Kabel

Verwendete Pins:

Port D: D0, D1, D2, D3, D4, D5, D6, D7
Port B: B0, B1, B2, B3

Werden die Widerstände beim Verbinden der Com-Anschlüsse vergessen, wird der Mikro-controller definitiv zerstört

Verbinde nun die Anschlüsse der einzelnen *Segmente (a-g)* für die Zahlendarstellung mit den *Pins des Port D* des Mikrocontrollers nach der folgenden Zuordnung

a ➔ D0 ; b ➔ D1 ; c ➔ D2 ; ...

Die *COM-Anschlüsse* der einzelnen Digits werden jeweils über einen Widerstand mit den *Pins des Port B* verbunden

Com 12 ➔ B0 ; Com 9 ➔ B1 ; Com 8 ➔ B2 ; ...

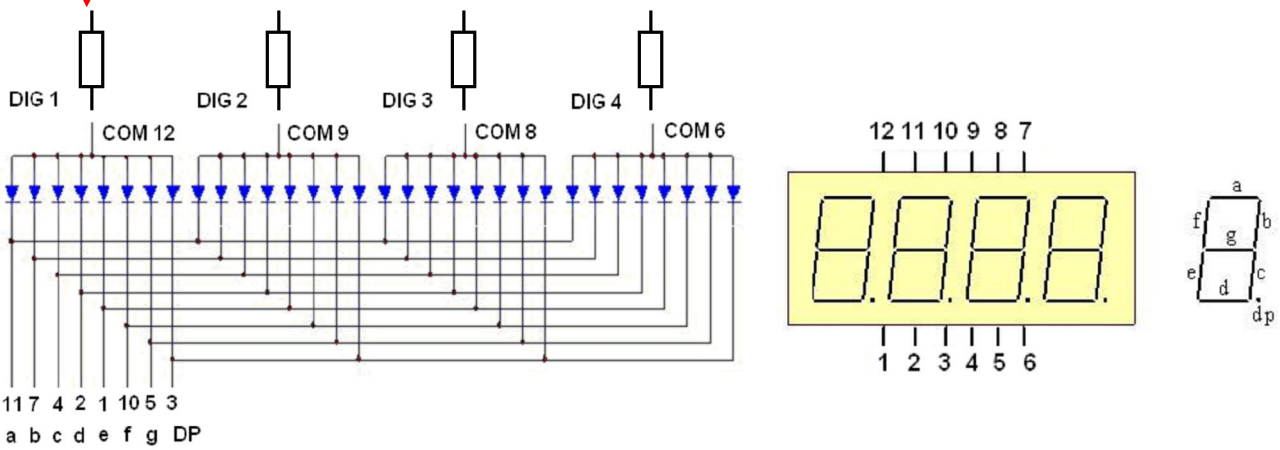


Abb. 5.4: Auszug aus einem Datenblatt einer 7 –Segment-Anzeige mit 4 Digits

Ein Digit ansteuern

Schreibe ein Countdown Programm (3, 2, 1, ...) für ein Digit!

Die Zuordnung für die einzelnen Segmente muss vorher *invertiert* werden, d.h. alle “Einsen“ und “Nullen“ müssen anders wie bei der einzelnen 7-Seg-Anzeige getauscht werden.

Z.B. **Drei'** b, c: Portd = & 10110000 ➔ Portd = &B01001111

Programm xx

```
Config portd ...
Config portb ...

Do
   Portb.0 = 1       'Aktiviert (/Deaktiviert) Digit 1
   Portb.1 = 0       'Aktiviert (/Deaktiviert) Digit 2
   Portb.2 = 0       'Aktiviert (/Deaktiviert) Digit 3
   Portb.3 = 0       'Aktiviert (/Deaktiviert) Digit 4

   Portd  = & 10110000
   Waitms 200
   ...
END
```

Ergänze das Programm!

Was beobachtest du?

Was ändert sich, wenn Portb.0 = 0 und Portb.0 = 1 gesetzt wird?

Was ändert sich, wenn Portb.2 = 1 und Portb.3 = 1 gesetzt wird?

Erkläre was mit dem Portb gesteuern wird!

Mehrer Digits hintereinander ansteuern

Nun wollen wir eine 2-stellige Zahl auf der Anzeige ausgeben. Dazu muss das Programm so erweitert werden, dass alle Digits nacheinander angesteuert werden.
Damit dieses funktioniert und die Anzeige ständig wiederholt wird, benötigt man ein spezielles Timerprogramm (Timer0).

Programm xx

```
' Config Port
Config portd = Output
Config portb = Output
' -----------------------------------------------------
' Config VAR
Dim Einer as Byte
Dim Zehner as Byte
Dim Ziffer as Byte
' -----------------------------------------------------
' Timer0
Config Timer0 = Timer , Prescale = 8
On Timer0 Anzeige        ' Wenn Timersignal,
                         ' gehe zum
                         ' Unterprogramm Anzeige
Const Timerinit = 400
Enable Timer0
Enable Interrupts
' -----------------------------------------------------
Einer = 7
Zehner = 5

Do
  Nop                    ' No Oparation
Loop
END
```

```
'-----------------------------------------------------
Anzeige:
   Portb.0 = 0
   Portb.1 = 1
   Ziffer = Einer
   Gosub Anzz
   Portb.0 = 1
   Portb.1 = 0
   Ziffer = Zehner
   Gosub Anzz
Return

Anzz:
Select Case Ziffer
   Case 0 : Portd = &B11000000   '0   a, b, c, d, e, f
    ...                          'Fehlende ergänzen
   Case 9 : Portd = &B10010000   '9   a, b, c, d, f, g
End Select

Waitus 80
Return
```

Abb. 5.4: Metronom

Der **Timer0** bewirkt, dass in einem vorgegebenen Rhythmus das Unterprogramm “Anzeige“ aufgerufen wird, um die Ziffern darzustellen. Dazu wird das eigentliche Programm unterbrochen. Würde der Timer0 nicht dafür sorgen, dass immer wieder in das Unterprogramm gesprungen wird, gäbe es keine fortlaufende Anzeige z.B. einer Uhrzeit oder eines Messergebnisses.

Schreibe das Programm xx!

Erkläre, was die einzelnen Statements im Programm bewirken!

```
Portb.0 = 0
Portb.1 = 1
Ziffer = Einer
Gosub Anzz
```

Was bewirkt Select Case?

Erweitere das Programm so, dass eine 4-stellige Zahl angezeigt wird!

Eine Zahl auf die einzelnen Digits aufteilen

Klar ist nun wie die einzelnen Digits angesteuert werden, aber wie wird eine Zahl mit 3 Ziffern auf die einzelnen Digits aufgeteilt?
Für die ersten drei Digits (Einer, Zehner, Hunderter) wird Programm xx folgendermaßen erweitert.

Programm xx

```
' Config VAR
...
DIM Zahl as Word
DIM Zahl_tmp as Word
DIM Hunderter as Word
'-----------------------------------------------------
Anzeige:
Zahl_tmp = Zahl
Einer = Zahl_tmp  Mod 10          '1234  ->  4 Einer

Zahl_tmp = Zahl_tmp - Einer       '1234  ->  1230
Zahl_tmp = Zahl_tmp / 10          '1230  ->  123
Zehner = Zahl_tmp  Mod 10         '123   ->  3 Zehner

Zahl_tmp = Zahl_tmp – Zehner      '123  ->  120
Zahl_tmp = Zahl_tmp / 10          '120  ->  12
Hunderter = Zahl_tmp  Mod 10      '12   ->  2 Hunderter

  Portb.0 = 0
  Portb.1 = 0
  Portb.2 = 1
  Ziffer = Einer
  Gosub Anzz
...
Return
```