Mikrocontroller-Seminar O28

Nicolas DL1DOW Sänger

29. Dezember 2007

There are 10 kinds of human: Those who understand binary and those who don't

1 Inhaltsübersicht

Die ganze Veranstaltung ist als absolute Anfänger-Einführungsveranstaltung gedacht, um der Welt der Mikrocontroller das Geheimnisvolle zu nehmen. „Alte Hasen“ sind natürlich als Hilfestellung jederzeit auch willkommen. Vorraussetzung sollten Basis-Programmierkenntnisse sein („Was sind Schleifen?“, „Wozu dienen Variablen?“...). Benutzt wird der ATMEGA8 oder der ATMEGA168 in einer eigenen Entwicklungsumgebung („Arduino“), Programmiersprache ist C (Basiskenntnisse hilfreich, aber nicht erforderlich). Theorie, Software und Hardware sollen dabei immer abwechselnd und ineinandergreifend behandelt werden, um mit ersten Resultaten und Übungen später die direkte Anwendung in eigenen Projekten zu ermöglichen.

Übersicht

- Aufbau Testplatine mit ATMEGA8 oder ATMEGA168
- Installation Entwicklungsumgebung
- ISP/High-Voltage-Programming, Bootloader, Fuse-Bytes
- Grundlagen C (Schleifen, Bedingungen, Datentypen)
- Grundlagen μC (In- und Outputfunktionen, Ports)
- Entwicklungsumgebung, GCC
- Hochladen des ersten Programmes („Hallo Welt“)
- Analogsensoren
- Ansteuerung „intelligenter“ LCDs
- Schieberegister
- Timer, Interrupts
- Entprellen von Tastern und Schaltern
- Schrittmotorsteuerung (bei Wunsch)
- ...

Eine Testplatine, die benötigten Elektronikkomponenten und der nötige USB- oder RS232-Adapter werden zur Verfügung gestellt.

2 Testplatine

Die äußere Beschaltung des Atmega8 gestaltet sich sehr einfach und beschränkt sich auf eine 5V-Spannungsversorgung, einen Quarz (1-16 MHz), einen Pull-Up-Widerstand für die Resetleitung und eine handvoll (also fünf) Kondensatoren (siehe Abbildung 2.2). Die im Rahmen dieses Seminars verwendete Testplatine verfügt darüberhinaus über LEDs, um einen direkten Überblick über den Status jeder Leitung zu haben und die entsprechenden Steckkontakte. Der Kondensator für die interne Referenzspannung fehlt und muß bei Bedarf auf der Leiterseite zwischen den Pins 21 und 22 oder unterhalb der IC-Fassung eingelötet werden.

Wie in Abbildung 2.1 zu erkennen ist, sind die Pins 6 und 7 des Atmega durch einen Fehler auf der Leiterplatte miteinander verbunden. Dieser Leiterbahnkontakt muß unbedingt entfernt werden!

Bei der Bestückung ist bei den sieben Lötbrücken zu beginnen. Die linken sechs Widerstände und LEDs (PortC) sind nicht zu bestücken.

Stückliste:

1×	Widerstand 10kΩ
1×	Diode
1×	IC-Fassung 28pol
1×	Quarz 16MHz
2×	LED 2mA rot
6×	LED 2mA gelb
7×	LED 2mA grün
1×	Sockelleiste 1×14
1×	Sockelleiste 1×6
1×	Sockelleiste 2×5

1×	Stiftleiste 1×5
1×	Stiftleiste 2×3
5×	100nF, RM2,55mm
2×	22pF, RM2,55mm
15×	Widerstand 1k5Ω
1×	Taster
1×	Hohlbuchse
2×	Elko $\geq 100\mu$F
1×	LM7805
1×	Atmega8 oder Atmega168

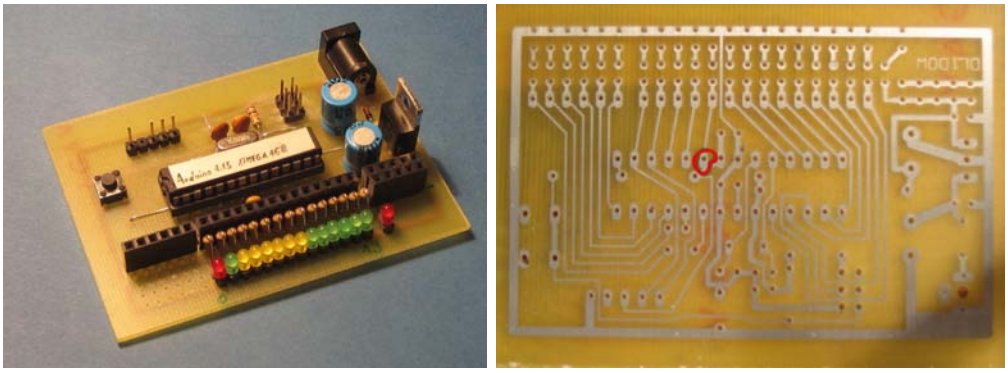

Abbildung 2.1: Fertig aufgebautes Testboard / Achtung, Fehler auf der Leiterplatte!

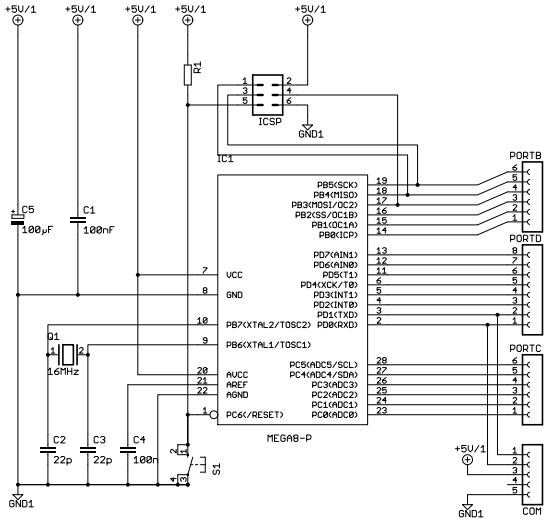

Abbildung 2.2: Vereinfachter Schaltplan

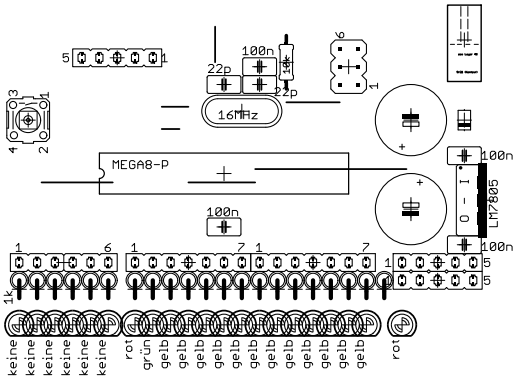

Abbildung 2.3: Bestückungsplan

Hinweis Sockelleisten: Erst Pin rausziehen, dann mit scharfem Messer teilen! Vor Einfügen der Hohlbuchse müssen zwei Schlitze gefräst werden.

3 Programmierung

3.1 „Hallo Welt“

```
// "Hallo Welt: Blinken von LED 0; Arduino

void setup()
{
  pinMode(0,OUTPUT);
  }

void loop()
{
  digitalWrite(0,HIGH);
  delay(200);
  digitalWrite(0,LOW);
  delay(200);
  }
```

Oben ist das „Hallo Welt“, bestehend aus einer blinkenden LED an Pin 0 in der „Arduino programming language“, einer Abart von C/C++. Vorteil dieser Programmiersprache ist die sehr einfache Erlernbarkeit und die fast selbstredenden Befehle.

Das gleiche Programm in „normalem“ für Mikrocontroller, wie es der GCC (Gnu Cross Compiler) verwendet, der auch in AVR-Studio eingebunden ist, sähe wie folgt aus:

```
// "Hallo Welt: Blinken von LED 0;
//
// GCC/AVR-Studio kompatibles C

#include <avr/delay.h>
#include <avr/io.h>

void main()
{
  DDRD=0x01;

  while(1)
  {
    PORTD=0x01;
```

```
    for (int i=0; i<20; i++)
    {
      _delay_ms(10);
    }
    PORTD=0x00;

    for (int i=0; i<20; i++)
    {
      _delay_ms(10);
    }
  }
}
```

Als Zugeständnis an die verwendete Entwicklungsumgebung sieht der Quelltext in „Arduino-C“ wie folgt aus:

```
// "Hallo Welt: Blinken von LED 0;
//
// Arduino kompatibles C

#include <avr/delay.h>
#include <avr/io.h>

void setup()
{
  DDRD=0x01;
}

void loop()
{
  PORTD=0x01;
  for (int i=0; i<20; i++)
  {
    _delay_ms(10);
  }
  PORTD=0x00;

  for (int i=0; i<20; i++)
  {
    _delay_ms(10);
  }
}
```

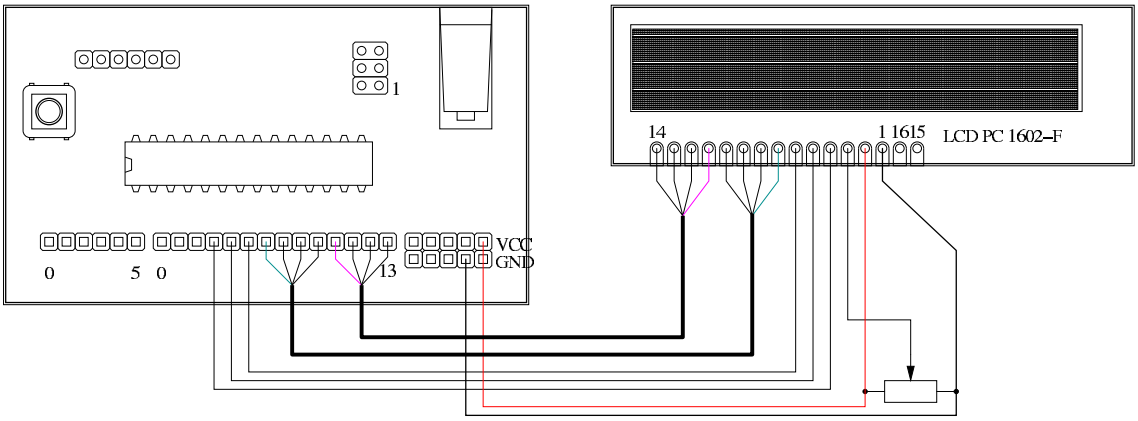

Abbildung 3.1: Anschlußschema des LCDs

3.2 LCDs

Viele Textbasierte LCDs verwenden einen „HD44780“ oder kompatiblen Controller. Im Großen und Ganzen ist die Pinbelegung immer recht ähnlich, für die exakte Pinbelegung ist in jedem Fall das Datenblatt zu konsultieren. Insbesondere die Beschaltung für die Kontrastspannung und für die Hintergrund-LEDs variiert.

Ansteuerung per Microcontroller

Die Ansteuerung eines HD44780-basierten Displays gestaltet sich sowohl was den Hardware- als auch den Softwareaufwand angeht extrem einfach und eignet sich somit sehr gut als Übungsprojekt für Anfänger.

Das HD44780-Interface besteht aus 8 Datenleitungen(D0-D7) sowie den 3 Steuerleitungen RS(Register Select), R/W(Read/Write) und E(Enable). Die Displays werden mit 5V Betriebsspannung versorgt, die Kontrastspannung V0 bekommt man, indem man ein 10k-Poti zwischen VCC und GND anschließt und den mittleren Anschluss als Kontrastspannung verwendet. Bei ca. 4,5 Volt werden dann die Pixel sichtbar. Optional haben viele Displays noch eine LED-Beleuchtung eingebaut, diese benötigt oft einen Vorwiderstand, um an 5V angeschlossen zu werden. Möchte man nur einen 8-Bit-Port zur Ansteuerung verwenden, so ist dies möglich, indem man lediglich die Pins D4-D7 sowie die 3 Steuerleitungen an den Controller anschliesst, das Display muss dann im 4-Bit-Modus betrieben werden.

3.2.1 Initialisierung

Initialisierung im 8 Bit Modus

Im 8 Bit Modus kann die Initialisierungssequenz in Form kompletter Bytes übertragen werden:

- Nach dem Anlegen der Betriebsspannung muss eine Zeit von mindestens ca. 15ms gewartet werden, um dem LCD-Kontroller Zeit für seine eigene Initialisierung zu geben
- `0x30` ins Steuerregister schreiben (RS = 0)
- mindestens 4.1ms warten
- `0x30` ins Steuerregister schreiben (RS = 0)
- mindestens 100μs warten
- 0x30 ins Steuerregister schreiben (RS = 0)
- Mit dem Konfigurier-Befehl `0x30` das Display konfigurieren (8-Bit, 1 oder 2 Zeilen, 5x7 Format) mit den restlichen Konfigurierbefehlen die Konfiguration vervollständigen: Display ein/aus, Cursor ein/aus, etc.

Initialisierung im 4 Bit Modus

Achtung: Im folgenden sind alle Bytes aus Sicht des LCD-Kontrollers angegeben! Da LCD-seitig nur die Leitungen DB4 - DB7 verwendet werden, ist daher immer nur das höherwertige Nibbel gültig. Durch die Art der Verschaltung (DB4 - DB7 wurde auf dem PORT an PD0 bis PD3 angeschlossen) ergibt sich dadurch eine Verschiebung, so dass das am Kontroller auszugebende Byte nibblemässig vertauscht ist!

Die Sequenz, aus Sicht des Kontrollers, sieht so aus:

- Nach dem Anlegen der Betriebsspannung muss eine Zeit von mindestens ca. 15ms gewartet werden, um dem LCD-Kontroller Zeit für seine eigene Initialisierung zu geben
- `0x3` ins Steuerregister schreiben (RS = 0)
- mindestens 4.1ms warten
- `0x3` ins Steuerregister schreiben (RS = 0)
- mindestens 100μs warten
- `0x3` ins Steuerregister schreiben (RS = 0)
- `0x2` ins Steuerregister schreiben (RS = 0), dadurch wird auf 4 Bit Daten umstellt

ab jetzt muss für die übertragung eines Bytes jeweils das höherwertige Nibble und dann das niederwertige Nibble übertragen werden, wie oben beschrieben Mit dem Konfigurier-Befehl `0x20` das Display konfigurieren (4-Bit, 1 oder 2 Zeilen, 5x7 Format) mit den restlichen Konfigurierbefehlen die Konfiguration vervollständigen: Display ein/aus, Cursor ein/aus, etc.

Abbildung 3.2: Speicheradressen gängiger LCD-Formate

3.3 Befehlsreferenz

delay.h

Der einfachste Fall des Wartens, das Zeitvertrödeln, kann in vielen Fällen und mit großer Genauigkeit anhand der avr-libc Bibliotheksfunktionen `_delay_ms()` und `_delay_us()` erledigt werden. Die Bibliotheksfunktionen sind einfachen Zählschleifen (Warteschleifen) vorzuziehen, da leere Zählschleifen ohne besondere Vorkehrungen sonst bei eingeschalteter Optimierung vom avr-gcc-Compiler wegoptimiert werden. Weiterhin sind die Bibliotheksfunktionen bereits darauf vorbereitet, die in `F_CPU` definierte Taktfrequenz zu verwenden. Ausserdem sind die Funktionen der Bibliothek wirklich getestet.

Diese Bibliotheksfunktionen funktionieren allerdings nur dann korrekt, wenn sie mit zur Übersetzungszeit (beim Compilieren) bekannten konstanten Werten aufgerufen werden. Die Wartezeit der Funktion `_delay_ms()` ist auf 262,14ms/`F_CPU` (in MHz) begrenzt, d.h. bei 16 MHz kann man nur max. 16,3ms warten. Die Wartezeit der Funktion `_delay_us()` ist auf 768μs/`F_CPU` (in MHz) begrenzt, d.h. bei 16 MHz kann man nur max. 48μs warten. Längere Wartezeiten mssen dann ber einen mehrfachen Aufruf in einer Schleife gelöst werden. Der Quellcode muss mit eingeschalteter Optimierung übersetzt werden, sonst wird sehr viel Maschinencode erzeugt und die Wartezeiten stimmen nicht mehr mit dem Parameter überein.

stdio.h

Die Bilbliothek „`stdio.h`“ stellt unter anderem die `printf`-Funktion zur Verfügung. Die Formatierung bei `printf` wird mittels Formatierungszeichen realisiert.

Häufig benutzte Zeichen sind:

%d, %i	Decimal signed integer
%o	Octal integer
%x ,%X	Hex integer
%u	Unsigned integer
%c	Character
%s	String (siehe unten)
%f	double
%e ,%E	double
%g , %G	double
%p	Zeiger
%n	Number of characters written by this printf No argument expected
%%	% . No argument expected

Als Parameter muß immer ein Wert, kein Zeiger "bergeben werden. (Ausnahme: `%`; bei einem String wird erwartet, daß ein Zeiger übergeben wird.

Analog zu `printf` funktioniert `snprintf`. Die Syntax lautet:

```
snprintf(%s,%n,Argumente)
```

Im Gegensatz zu `sprintf()` wird nur eine endliche Anzahl `%n` Zeichen in den String `%s` geschrieben.

Beispiele:

```
snprintf(s,40,"Der HEX-Code lautet: %#X", zahl);
snprintf(s,40,"Die erste Zahl ist %d, die zweite Zahl ist %d, zahl1, zahl2);
```

Zusätzliche Formatierungen (Auszug):

-	linksbündig
0	Felder mit 0 ausfüllen (an Stelle von Leerzeichen)
+	Vorzeichen einer Zahl immer ausgeben
blank	positive Zahlen mit Leerzeichen beginnen
#	verschiedene Bedeutung: %#o (Oktal): 0 Präfix wird eingefügt %#x (Hex): 0x Präfix wird bei Werten $\neq$ Null eingefügt %#X (Hex): 0X Präfix wird bei Werten $\neq$ Null eingefügt %#e : Dezimalpunkt immer anzeigen %#E : Dezimalpunkt immer anzeigen %#f : Dezimalpunkt immer anzeigen

4 Quelltexte

Lauflicht 1 (Arduino)

```
1  void setup()
2  {
3    for(int i=0; i<=13; i++)
4    {
5      pinMode(i,OUTPUT);
6    }
7  }
8
9
10 void loop()
11 {
12   for (int i=0; i<=13; i++)
13   {
14     digitalWrite(i,HIGH);
15     delay(50);
16   }
17
18   delay(200);
19
20   for(int i=0; i<=13; i++)
21   {
22     digitalWrite(i,LOW);
23     delay(100);
24   }
25 }
```

Lauflicht 1 (C)

```
#include <avr/delay.h>
#include <avr/io.h>

#define F_CPU 16000000L

void setup()
{
  DDRD=0xFF;
  DDRB=0x3F;

  while(1)
  {
    for (int i=0; i<=13; i++)
    {
      PORTD |= (1 << i);
      //PORTB |= (1 << (i-8)); // klappt nicht!
      PORTB |= (1 << i) >> 8; // so klappt das
      _delay_ms(50);
    }

    for (int i=0; i<=20; i++)
    {
    _delay_ms(50);
    }

    for(int i=0; i<=13; i++)
    {
      PORTD &= ~(1 << i);
      PORTB &= ~(1 << i) >> 8;
      _delay_ms(100);
    }
  }
}

void loop(){}
```

LCD 8-bit HD44780 (Arduino)

```
/*
 Ansteuerung eines zweizeiligen, HD44780-kompatiblen Textdisplays
 im 8-bit-Modus; Ausfuehrung der Befehle wird einfach gewartet
 (Arduino-Quelltext)
 */

// Definition der Daten-Anschlusspins (Variablen)
int lcdPins[] = {
  6, 7, 8, 9, 10, 11, 12, 13};

// Definition der Steuerpins (Konstanten!)
#define LCD_RS 3
#define LCD_RW 4
#define LCD_EN 5

// Definition der Wartezeiten (in Mikrosekunden!)
#define LONGDELAY 2000
#define SHORTDELAY 700

// Kommandos an LCD uebergeben
void lcd_putcom(char kommando)
{
  for (int i=0; i<8; i++)
  {
    digitalWrite(lcdPins[i],kommando & 0x01);
    kommando >>= 1;
  }

  delayMicroseconds(1);
  digitalWrite(LCD_EN,HIGH);
  delayMicroseconds(1);
  digitalWrite(LCD_EN,LOW);

  delayMicroseconds(SHORTDELAY);
}

// Zeichen an LCD ausgeben
void lcd_putdat(char zeichen)
{
  digitalWrite(LCD_RS,HIGH);
  lcd_putcom(zeichen);
  digitalWrite(LCD_RS,LOW);
}
```

```
44 void lcd_init()
45 {
46   digitalWrite(LCD_RW,LOW);
47   lcd_putcom(0x30);
48   delayMicroseconds(LONGDELAY);
49   lcd_putcom(0x30);
50   delayMicroseconds(LONGDELAY);
51   lcd_putcom(0x30);
52   delayMicroseconds(LONGDELAY);
53
54   lcd_putcom(B00111000);
55   delayMicroseconds(LONGDELAY);
56   lcd_putcom(0x0F);
57   delayMicroseconds(LONGDELAY);
58   lcd_putcom(0x01);
59   delayMicroseconds(LONGDELAY);
60   lcd_putcom(0x06);
61   delay(5);
62
63 }
64
65 void setup()
66 {
67   for (int i=2; i<=13; i++)
68   {
69     pinMode(i,OUTPUT);
70   }
71
72   lcd_init();
73   lcd_putdat('H');
74   lcd_putdat('a');
75   lcd_putdat('l');
76   lcd_putdat('l');
77   lcd_putdat('o');
78   lcd_putdat(0x33);
79
80 }
81
82 void loop()
83 {
84
85 }
```

LCD – Hilfsfunktionen (Arduino)

Cursor im LCD auf eine vorgegebene Position bringen (siehe auch Abbildung 3.2 auf Seite 10):

```
1   #define DISPLAYLENGTH 16
2   #define DISPLAYHEIGHT 2
3   #define DISPADRESSABST 0x40
4
5  // Cursor auf Position X, Y bewegen
6  void lcd_pos(int posX, int posY)
7  {
8    posX %= DISPLAYLENGTH;
9    posY %= DISPLAYHEIGHT;
10   lcd_putcom((posX+posY*DISPADRESSABST) | 0x80);
11   delayMicroseconds(LONGDELAY);
12 }
```

Kommando zum Löschen des Displayinhalts (Siehe Datenblatt DIP162):

```
1   #define CLEARCOMMAND 0x01
2
3  // Inhalt des Displays komplett loeschen
4  void lcd_clear()
5  {
6    lcd_putcom(CLEARCOMMAND);
7    delayMicroseconds(LONGDELAY);
8  }
```

Zeichenketten (Strings) am LCD ausgeben:

```
1  // Strings auf LCD ausgeben
2  void lcd_putstr(char *zeichenkette)
3  {
4    while (*zeichenkette)
5    {
6      lcd_putdat(*zeichenkette);
7      zeichenkette++;
8    }
9  }
```

Benutzung im Hauptprogramm (`setup`):

```
1  void setup()
2  {
3    for (int i=2; i<=13; i++)
4    {
5      pinMode(i,OUTPUT);
6    }
7
8    lcd_init();
9    lcd_putstr("Hallo");
10
11   delay(500);
12   lcd_clear();
13   lcd_pos(0,0);
14   lcd_putstr("Nochmal");
15
16   lcd_pos(5,1);
17   char s[]=("Gespeichert");
18   lcd_putstr(s);
19 }
```

Benutzung der `printf()`-Funktion:

```
1  #include <stdio.h>
2  char cs[40];
3  int zaehler=0;
4
5  // Definition der Daten-Anschlusspins (Variablen)
6  int lcdPins[] = {
7    6, 7, 8, 9, 10, 11, 12, 13};
8
9  // Definition der Steuerpins (Konstanten!)
10 #define LCD_RS 3
11 #define LCD_RW 4
12 #define LCD_EN 5
13
14 // Definition der Wartezeiten (in Mikrosekunden!)
15 #define LONGDELAY 2000
16 #define SHORTDELAY 700
17
18 // Zusaetzliche Konstanten fuer Hilfsfunktionen
19 #define CLEARCOMMAND 0x01
20 #define DISPLAYLENGTH 16
21 #define DISPLAYHEIGHT 2
22 #define DISPADRESSABST 0x40
23
24
25 void setup()
26 {
27   for (int i=2; i<=13; i++)
28   {
29     pinMode(i,OUTPUT);
30   }
31
```

LCD 4-bit HD44780 (Arduino)

```
/*
 Ansteuerung eines zweizeiligen, HD44780-kompatiblen Textdisplays
 im 4-bit-Modus; Ausfuehrung der Befehle wird einfach gewartet
 (Arduino-Quelltext)
 */

#include <stdio.h>
char cs[40];
int zaehler=0;

// Definition der Daten-Anschlusspins (Variablen)
int lcdPins[] = {10, 11, 12, 13};
int emptyPins[] = {6, 7, 8, 9};

// Definition der Steuerpins (Konstanten!)
#define LCD_RS 3
#define LCD_RW 4
#define LCD_EN 5

// Definition der Wartezeiten (in Mikrosekunden!)
#define LONGDELAY 2000
#define SHORTDELAY 700

// Zusaetzliche Konstanten fuer Hilfsfunktionen
#define CLEARCOMMAND 0x01
#define DISPLAYLENGTH 16
#define DISPLAYHEIGHT 2
#define DISPADRESSABST 0x40


void setup()
{
  for (int i=2; i<=13; i++)
  {
    pinMode(i,OUTPUT);
  }

  for (int i=0; i<4; i++)
  {
    pinMode(emptyPins[i],INPUT);
    }

  lcd_init();
  lcd_putstr("Initialized");
}
```

```
44 void loop()
45 {
46   lcd_pos(0,1);
47   snprintf(cs,40,"Loop No.: %X", zaehler);
48   lcd_putstr(cs);
49   zaehler++;
50 
51   if (zaehler==0xFFFF)
52   {
53     zaehler=0;
54   }
55 }


56 // Kommandos an LCD uebergeben
57 void lcd_putcom(char kommando)
58 {
59   char temp=kommando;
60 
61   // Hoeherwertiges Halbbyte uebertragen
62   kommando >>=4;
63   for (int i=0; i<4; i++)
64   {
65     digitalWrite(lcdPins[i],kommando & 0x01);
66     kommando >>= 1;
67   }
68 
69   delayMicroseconds(1);
70   digitalWrite(LCD_EN,HIGH);
71   delayMicroseconds(1);
72   digitalWrite(LCD_EN,LOW);
73   delayMicroseconds(1);
74 
75   // Niederwertiges Halbbyte uebertragen
76   for (int i=0; i<4; i++)
77   {
78     digitalWrite(lcdPins[i],temp & 0x01);
79     temp >>= 1;
80     }
81 
82   delayMicroseconds(1);
83   digitalWrite(LCD_EN,HIGH);
84   delayMicroseconds(1);
85   digitalWrite(LCD_EN,LOW);
86 
87   delayMicroseconds(SHORTDELAY);
88 }
```

```
89  // Zeichen an LCD ausgeben (4 bit)
90  void lcd_putdat(char zeichen)
91  {
92    digitalWrite(LCD_RS,HIGH);
93    lcd_putcom(zeichen);
94    digitalWrite(LCD_RS,LOW);
95  }

96  // Halbbytes ans LCD uebergeben
97  void lcd_putini(char kommando)
98  {
99      // Niederwertiges Halbbyte uebertragen
100   for (int i=0; i<4; i++)
101   {
102     digitalWrite(lcdPins[i],kommando & 0x01);
103     kommando >>= 1;
104     }
105
106   delayMicroseconds(1);
107   digitalWrite(LCD_EN,HIGH);
108   delayMicroseconds(1);
109   digitalWrite(LCD_EN,LOW);
110
111   delayMicroseconds(SHORTDELAY);
112
113   }

114 // Initialisierung im 4-Bit-Modus
115 void lcd_init()
116 {
117   digitalWrite(LCD_RW,LOW);
118   lcd_putini(0x3);
119   delay(5);
120   lcd_putini(0x3);
121   delay(1);
122   lcd_putini(0x3);
123   delay(1);
124   lcd_putini(0x2);
125
126   lcd_putcom(0x0F);
127   delayMicroseconds(LONGDELAY);
128   lcd_putcom(0x01);
129   delayMicroseconds(LONGDELAY);
130   lcd_putcom(0x06);
131   delay(5);
132
133 }
```

LCD 4-bit in C

```
/*
 Ansteuerung eines zweizeiligen, HD44780-kompatiblen Textdisplays
 im 4-bit-Modus; Ausfuehrung der Befehle wird einfach gewartet;
 andere Pinbelegung
 (C-Quelltext)
 */

#include <avr/delay.h>
#include <avr/io.h>
#include <stdio.h>

char cs[40];
int zaehler=0;

int lcdPins[] = {10, 11, 12, 13};

// Definition des LCD-Anschlusses
#define LCDPORT PORTB

// Definition der Steuerpins (Konstanten!)
#define LCD_RS 8
#define LCD_EN 9
#define LCD_RSA 0
#define LCD_ENA 1

// Definition der Wartezeiten (in Mikrosekunden!)
#define LONGDELAY 2000
#define SHORTDELAY 700

// Zusaetzliche Konstanten fuer Hilfsfunktionen
#define CLEARCOMMAND 0x01
#define DISPLAYLENGTH 16
#define DISPLAYHEIGHT 2
#define DISPADRESSABST 0x40


void longdelay()
{
  for (int i=0; i<LONGDELAY/10; i++)
  {
    _delay_us(10);
  }
}

void shortdelay()
{
  for (int i=0; i<SHORTDELAY/10; i++)
  {
    _delay_us(10);
  }
}
```

```
49 // Kommandos an LCD uebergeben
50 void lcd_putcom(char kommando)
51 {
52   // Hoeherwertiges Halbbyte uebertragen
53 **
54
55   // Enable-Impuls
56 **
57 **
58 **
59 **
60   // Niederwertiges Halbbyte uebertragen
61 **
62
63
64   // Enable-Impuls
65 **
66 **
67 **
68 **
69   for (int i=0; i<SHORTDELAY/10; i++)
70   {
71     _delay_us(10);
72   }
73 }

74 // Zeichen an LCD ausgeben
75 void lcd_putdat(char zeichen)
76 {
77 **
78   lcd_putcom(zeichen);
79 **
80 }

81 // Halbbytes ans LCD uebergeben
82 void lcd_putini(char kommando)
83 {
84   // Niederwertiges Halbbyte uebertragen
85 **
86
87 **// Enable-Puls
88 **
89 **
90 **
91
92   for (int i=0; i<SHORTDELAY/10; i++)
93   {
94     _delay_us(10);
95   }
96
97 }
```

```
98  // 4-Bit-Modus
99  void lcd_init()
100 {
101 **
102   _delay_ms(5);
103 **
104   _delay_ms(1);
105 **
106   _delay_ms(1);
107 **
108
109 **
110   longdelay();
111   lcd_putcom(0x01);
112   longdelay();
113   lcd_putcom(0x06);
114   _delay_ms(5);
115
116 }

117 /*
118  Kleine Sammlung von Hilfsfunktionen, um die Ausgabe an
119  LCDs etwas komfortabler zu gestalten
120  */
121
122 // Cursor auf Position X, Y bewegen
123 void lcd_pos(int posX, int posY)
124 {
125   posX %= DISPLAYLENGTH;
126   posY %= DISPLAYHEIGHT;
127   lcd_putcom((posX+posY*DISPADRESSABST) | 0x80);
128   longdelay();
129 }

130 // Inhalt des Displays komplett loeschen
131 void lcd_clear()
132 {
133   lcd_putcom(CLEARCOMMAND);
134   longdelay;
135 }

136 // Strings auf LCD ausgeben
137 void lcd_putstr(char *zeichenkette)
138 {
139   while (*zeichenkette)
140   {
141     lcd_putdat(*zeichenkette);
142     zeichenkette++;
143   }
144 }
```

```
145 void setup()
146 {
147   DDRB=0xFF;
148 
149   lcd_init();
150   lcd_putstr("Initialized");
151 }


152 void loop()
153 {
154   lcd_pos(0,1);
155   snprintf(cs,40,"Loop No.: %X", zaehler);
156   lcd_putstr(cs);
157   zaehler++;
158 
159   if (zaehler==0xFFFF)
160   {
161     zaehler=0;
162   }
163 }


164 int main()
165 {
166  setup();
167 
168  for(;;)
169  {
170   loop();
171  }
172 
173 return 0;
174  }
```

5 Literaturverzeichnis

[1] *Helmut Erlenkötter*, C Programmieren von Anfang an, ISBN 9783499600746

[2] AVR-Tutorial, `http://www.mikrocontroller.net/articles/AVR-Tutorial` Online-Ressource, Abruf: 11.11.2007

[3] *ATMEL*, Datenblatt Atmega8(L), Version 08/07

[4] *Electronic Assembly*, Datenblatt DIPS162, Version 07/2003

[5] *Julian Ilett*, How to use intelligent L.C.D.s, Everyday Practical Elektronics, February 1997