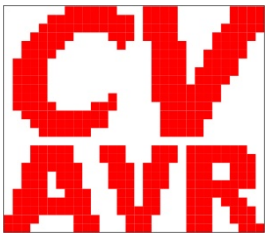
**AVR-8-bit-Mikrocontroller**
**Gruppe 500 - CodeVisionAVR C-Compiler**
**Inhaltsverzeichnis**



## Teil 601 - PB_LED



## Teil 602 - 2_Draht_LCD



## Teil 603 - IRDMS



## Teil 604 - Pegelsonde (Noch in Arbeit)

# 2 Ein LCD-Display anschalten und ansteuern

## 2.1 Zur Hardware des Shift-Registers und des LCD-Moduls 204B (funktionsgleich mit EA W204-NLED)

Dieses Projekt stellt eine Applikation zum Gebrauch des 2-Draht-LCD-Interface dar, das dazu geeignet ist, für alle weiteren Projekte, die ein LCD-Modul 204B (20x4-LCD-Anzeige) benötigen, eingesetzt zu werden.

**ACHTUNG:** Neu im Handel befindliche LCD-Displays sind etwas abgewandelt worden;
siehe **neues Datenblatt (EA W204-NLED)**

Der Treiber **lcd_2wire** ist für die Nutzung des LCD-Moduls **204B** mit dem Controller **HD44780** (oder kompatible) konzipiert. Die Schaltung ermöglicht durch die Zwischenschaltung eines **Shift-Registers 4094** die Ansteuerung eines LC-Displays im 4-Bit-Mode lediglich über 2 Port-Pins des Mikrocontrollers **ATmega88**.

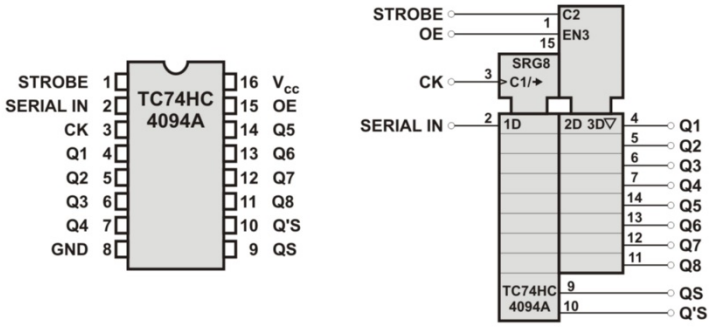



**Bild 2.1-01: Shift-Register TC74HC4094A** (Bildvergrößerung)

**Anmerkung:** Manche Shift-Register **4094** (z.B. das von Philips) haben eine andere Pin-Beschreibung als das hier verwendete 8-Bit-Shift-Register von Toshiba. Bei Philips beginnen die Q-Ausgänge bei **Q0** statt bei **Q1**. Hardwaremäßig sind die Pins jedoch identisch: Philips-**Q0** = Toshiba-**Q1** auf Pin4 usw.

In der vorliegenden Beschaltung des Shift-Registers **4094** wird ein an **SERIAL IN** (Pin 2) anstehendes **DATA**-Signal mit jedem **CLK**-Signal an **CK** (positive Flanke an Pin 3) der Reihe nach von **Q1** bis **Q7** verschoben. Die Eingänge **STROBE** und **OE** werden auf + 5 Volt gesetzt, **Q1**, **Q8**, **QS** und **Q'S** bleiben ohne Anschluss (No Connection - NC), so dass die Zusammenschaltung von Shift-Register und LCD auf das Wesentliche reduziert sich so darstellt:

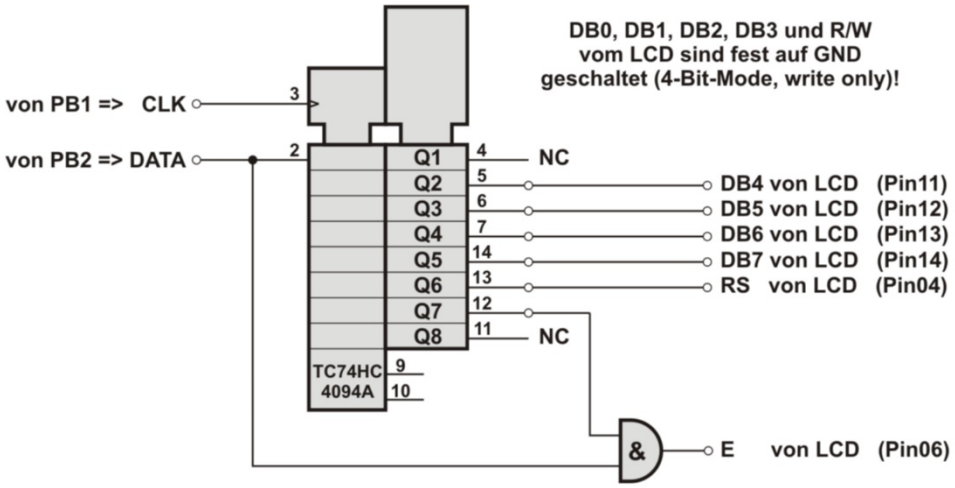



**Bild 2.1-02: Logische Zusammenschaltung von Shift-Register und LCD**
(Bildvergrößerung)

**Das Prinzip:** Während die Daten-Bits der Reihe nach vom Port-Bit (**PB2**) über die Leitung **DATA** an den Eingang **SERIAL-IN** (Pin2) des Shift-Registers gesendet werden, wird über ein weiteres Port-Bit (**PB1**) der Shift-Takt **CLK** an Pin3 des **4094** ausgesendet. Dabei wird stets das **MSB** (**M**ost **S**ignificant **B**it) zuerst an **SERIAL-IN** angelegt.

| Pin | Symbol | Pegel | Beschreibung |
|---|---|---|---|
| 01 | VSS | L | Versorgung 0 Volt, GND |
| 02 | VDD | H | Versorgung +5 Volt |
| 03 | VEE | - | Displayspannung 0 ... 0,5 Volt |
| 04 | RS | H/L | Register Select |
| 05 | R/W | H/L | H: Read / L: Write |
| 06 | E | H | Enable |
| 07 | D0 | H/L | Datenleitung 0 (LSB) |
| 08 | D1 | H/L | Datenleitung 1 |
| 09 | D2 | H/L | Datenleitung 2 |
| 10 | D3 | H/L | Datenleitung 3 |
| 11 | D4 | H/L | Datenleitung 4 |
| 12 | D5 | H/L | Datenleitung 5 |
| 13 | D6 | H/L | Datenleitung 6 |
| 14 | D7 | H/L | Datenleitung 7 (MSB) |
| 15 | LED + | - | LED-Versorgung Plus (über Widerstand) |
| 16 | LED - | - | LED-Versorgung Minus |

**Tabelle 2.1-01: Pinbelegung des EA W204-NLED**

Im 4-Bit-Mode werden bei der ersten negativen Impuls-Flanke an **E** des LCD (Pin06) die 4 Bits des oberen Nibbles übernommen und bei der zweiten negativen Impuls-Flanke an **E** die 4 Bits des unteren Nibbles. Dazu müssen natürlich die 4 Daten-Bits jeweils sicher an den Eingängen **DB4** bis **DB7** des LCD anstehen. An **Q6** (Pin13) des Shift-Registers steht zu diesem Zeitpunkt das **RS**-Signal an (`HIGH` für Daten-Übertragung oder `LOW` für Kommando-Übertragung).

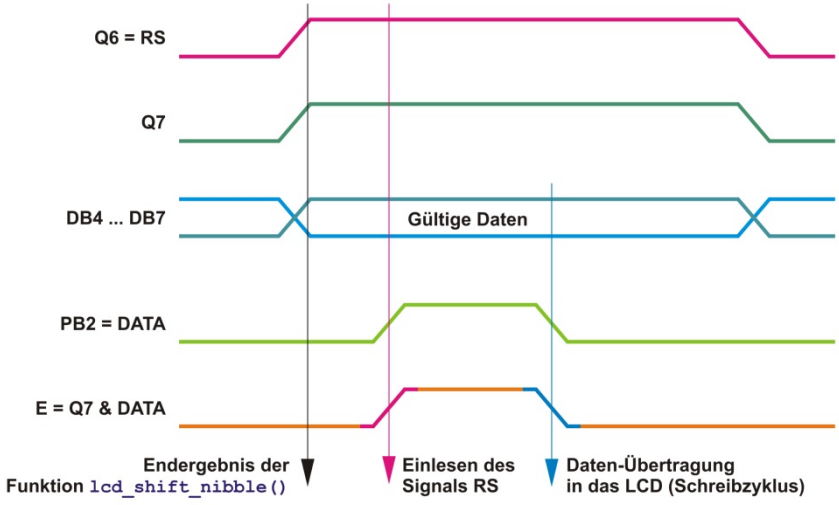



**Bild 2.1-03: Impulsdiagramm zur Übertragung eines Daten-Nibbles (Prinzip)**

Die andere Besonderheit der Schaltung ist die rechtzeitige Erzeugung des Enable-Signals **E** für das LCD. Damit die Übertragung zum LCD erst nach dem vollständigen Anliegen eines Nibbles an den Pins **Q2** bis **Q5** (Pin5, Pin6, Pin7 und Pin14 des IC's 4094) passiert, wird erst zu dem Zeitpunkt, an dem alle Bits im Shift-Register ihre korrekte Position erreicht haben, das Enable-Signal **E** aus dem in **Q7** (Pin12) abgelegten `HIGH` und einem gesonderten **DATA**-Impuls durch ein logisches **UND** gebildet und an **E** des LCD (Pin06) gesendet.

## 2.1.1 Adressierung des Textpuffers "DDRAM" im LCD

Das LC-Display kann in 4 Zeilen je 20 Zeichen darstellen (20x4). Für die Speicherung dieser 80 Zeichen ist ein Textpuffer (DDRAM) im LCD vorgesehen. Auf Grund des globalen Einsatzes des Controllers HD44780 (oder kompatiblen) für die unterschiedlichsten Displays, ist die Adressierung vordergründig sehr 'kryptisch'. Man muss bedenken, dass der Controller nur für 1- und 2-zeilige Displays konzipiert wurde. Das 4-zeilige Display beruht mithin auf einer 2-zeiligen Einstellung, bei der die physischen Display-Zeilen 0 und 2 sowie die Display-Zeilen 1 und 3 adressmäßig jeweils hintereinander zu denken sind:

**Zeile 0 => Zeile 2 => Zeile 1 => Zeile 3**

### Logische Adressierung des Displays:

**Spalten-Nummer** in der Zeile ist **x** = 0, 1, 2, 3, 4, 5, 6, . . . 14, 15, 16, 17, 18 oder 19

**Zeilen-Nummer** auf dem Display ist **y** = 0, 1, 2 oder 3

So sieht die programmtechnische Adressierung des **DDRAM** im Bezug auf die logische Adressierung des Displays aus (dezimal; die Adressen 40 bis 63 sind nicht ansprechbar !!):

| | Spalte 0 | Spalte 1 | Spalte 2 | . . . | Spalte 18 | Spalte 19 |
|---|---|---|---|---|---|---|
| **Zeile 0** | 0 | 1 | 2 | . . . | 18 | 19 |
| **Zeile 1** | 64 | 65 | 66 | . . . | 82 | 83 |
| **Zeile 2** | 20 | 21 | 22 | . . . | 38 | 39 |
| **Zeile 3** | 84 | 85 | 86 | . . . | 102 | 103 |

**Tabelle 2.1.1-01: Adressierung des LCD-DDRAM**

Für diese 'kryptische' Zählweise wird das Array `__flash U08 LCD_ROW_TABLE[4]` im FLASH-Speicher mit folgenden Werten angelegt (definiert):

```
__flash U08 LCD_ROW_TABLE[4] =
{
  0x80,         // LCD_ROW_TABLE[0] = 10000000 => Zeile 0 von/bis-Adr.:  0 .... 19
  0xC0,         // LCD_ROW_TABLE[1] = 11000000 => Zeile 1 von/bis-Adr.: 64 .... 83
  0x80 + 20,    // LCD_ROW_TABLE[2] = 10010100 => Zeile 2 von/bis-Adr.: 20 .... 39
  0xC0 + 20     // LCD_ROW_TABLE[3] = 11010100 => Zeile 3 von/bis-Adr.: 84 ... 103
};
```

Das ist ein Array mit den Anfangsadressen der Display-Zeilen im **DDRAM** des LCD. Aufgerufen wird das Array nur in der Funktion

`led_goto_xy(x,y)`

um die Position des Cursors nach Bedarf festzulegen. Zusätzlich wird bei der Definition der Werte das Bit7 gesetzt, um aus diesen Konstanten das LCD-Kommando **SET DDRAM ADDRESS** zu bilden (siehe weiter unten).

Wenn ein anderes LCD angeschlossen werden soll, müssen die spezifischen Werte ermittelt und hier geändert werden.

## 2.1.2 Kommandos des LCD-Moduls 204B

(und kompatible LCD's, die ebenfalls mit dem Chip **HD44780** ausgerüstet sind)

Bevor auf dem Display sinnvoll Zeichen angezeigt werden können, muss es per Programm initialisiert werden. Dazu werden eine Reihe verschiedener Steuer-Kommandos aus nachfolgender Tabelle in das LCD geschrieben. Aber auch nach der Initialisierung (wie in dem Beispiel `led_goto_xy(x,y)`) werden in den Funktionen Kommandos benötigt. Darüber hinaus kann es sinnvoll sein, in anderen Projekten die Darstellung per LCD-Kommandos zu variieren.

**R/W ist auf GND gesetzt und mithin immer `0`, d.h. das LCD kann NUR beschrieben werden!**

| Signal / Kommando | RS | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 | Bemerkungen |
|---|---|---|---|---|---|---|---|---|---|---|
| CLEAR DISPLAY | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Das gesamte Display (Textpuffer) wird gelöscht |
| RETURN HOME | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | Der Cursor wird auf das 1. Zeichenfeld gestellt |
| SET ENTRY MODE | 0 | 0 | 0 | 0 | 0 | 0 | 1 | I/D | SH | Cursor-Laufrichtung<br>I/D: 1 Increase, 0 Decrease<br>Anzeige-Shift<br>SH: 1 Shift rechts, 0 kein Shift |
| DISPLAY ON/OFF | 0 | 0 | 0 | 0 | 0 | 1 | D | C | B | Display<br>D: 1 Display EIN, 0 Display AUS<br>Cursor-Ansicht<br>C: 1 Ansicht EIN, 0 Ansicht AUS<br>Cusor-Blinken<br>B: 1 Blinken EIN, 0 Blinken AUS |
| SHIFT | 0 | 0 | 0 | 0 | 1 | S/C | R/L | 0 | 0 | Verschieben<br>S/C: 1 Display, 0 Cursor<br>Richtung<br>R/L: 1 1 Stelle rechts, 0 1 Stelle links |
| SET FUNCTION | 0 | 0 | 0 | 1 | DL | N | F | 0 | 0 | Interface-Modus<br>DL: 1 8-Bit-Mode, 0 4-Bit-Mode<br>Anzahl Zeilen<br>N: 1 2-zeilig, 0 1-zeilig<br>Punkt-Muster<br>F: 1 5x10-Matrix, 0 5x7-Matrix |
| SET CGRAM ADDRESS | 0 | 0 | 1 | CGRAM-Adresse | | | | | | Kommando zum Setzen der CGRAM-Adresse |
| SET DDRAM ADDRESS | 0 | 1 | DDRAM-Adresse | | | | | | | Kommando zum Setzen der DDRAM-Adresse |
| WRITE DATA | 1 | Daten-Byte | | | | | | | | Schreibe Daten-Byte in das CGRAM oder DDRAM |

**Tabelle 2.1.2-01: Kommandos zur Steuerung des LCD's**

## 2.1.3 Start-Initialisierung des LCD-Moduls 204B

**(1) 50 ms Verzögerung beim Einschalt- und Reset-Vorgang (`delay_ms(50)`)**

Es wird zwar ein Byte übertragen aber nur das hohe Nibble ausgewertet!

**(2) übertrage 00110000 in das LCD LCD-Interface auf 8-Bit-Mode setzen**

**(3) 5 ms Verzögerung zwischen den Kommandos (`delay_ms(5)`)**

**(4) übertrage 00110000 in das LCD LCD-Interface auf 8-Bit-Mode setzen**

**(5) 5 ms Verzögerung zwischen den Kommandos (`delay_ms(5)`)**

**(6) übertrage 00110000 in das LCD LCD-Interface auf 8-Bit-Mode setzen**

**(7) 5 ms Verzögerung zwischen den Kommandos (`delay_ms(5)`)**

**(8) übertrage 00100000 in das LCD LCD-Interface auf 4-Bit-Mode setzen**

**(9) 5 ms Verzögerung zwischen den Kommandos (`delay_ms(5)`)**

Jetzt wird das Display im 4-Bit-Mode am Interface betrieben. **Alle** folgenden 2 Nibbles eines Bytes werden automatisch vom LCD zusammengesetzt:

**(10) übertrage `00101000` in das LCD 4-Bit-Mode, 2-zeilig und 5x7-Punkt-Darstellung**

**(11) `5` ms Verzögerung zwischen den Kommandos (`delay_ms(5)`)**

**(12) übertrage `00001100` in das LCD Display ein, Cursor-Anzeige aus**

**(13) `5` ms Verzögerung zwischen den Kommandos (`delay_ms(5)`)**

**(14) übertrage `00000001` in das LCD Display (Textpuffer) loeschen**

**(15) übertrage `00000010` in das LCD Cursor auf 1. Zeichen (Adresse 0)**

**(16) `5` ms Verzögerung nach dem letzten Kommando (`delay_ms(5)`)**

## 2.2 Aufbau des AVR-C-Projektes 2_Draht_LCD

Um die Übersicht nicht zu verlieren, soll der Aufbau des Projektes in Form eines Block-Diagramms vorangestellt werden. Das Programm besteht aus den Blöcken:

**(1) Hauptprogramm**

**(2) Initialisierung des Projektes**

**(3) Initialisierung des LC Displays**

**(4) Kommando- oder Daten-Byte-OUT**

**(5) Kommando-Funktionen**

**(6) Daten-Format-Funktionen**

**(7) Konvertierungs-Funktionen**

Jeder Block besteht in der Regel aus zahlreichen Funktionen und Funktionsaufrufen, die in vielschichtiger Weise miteinander kommunizieren. Zur Vereinfachung sind wiederholt auftretende Funktionsaufrufe (beispielsweise die wiederholte Ausgabe von Initialisierungs-Kommandos) weggelassen worden.

Funktionen und ihre Aufrufe sind deutlich durch Pfeile voneinander zu unterscheiden: der Pfeil zeigt immer auf die aufgerufene Funktion. Die Funktionen sind farblich markiert, um deutlich zu machen, welchen Zweck sie verfolgen und zu welchem Block bzw. zu welchem Modul sie gehören.

Viele von den Funktionen werden in diesem Projekt nicht benötigt. Sie sind aber bereits vorausschauend erstellt worden und in Modulen zusammengefasst worden, um weiteren Projekten vorsorglich zur Verfügung zu stehen. Das sind im vorliegenden Fall:

- **Kommando-Funktionen** zum Absetzen spezieller Kommandos auf das LCD
- **Daten-Format-Funktionen** zur LCD-Ausgabe verschieden formatierter Zeichen, Zahlen und Zeichenketten
- **Konvertierungs-Funktionen** zur Konvertierung von Zeichen, Zahlen und Zeichenketten in ASCII-Zeichen

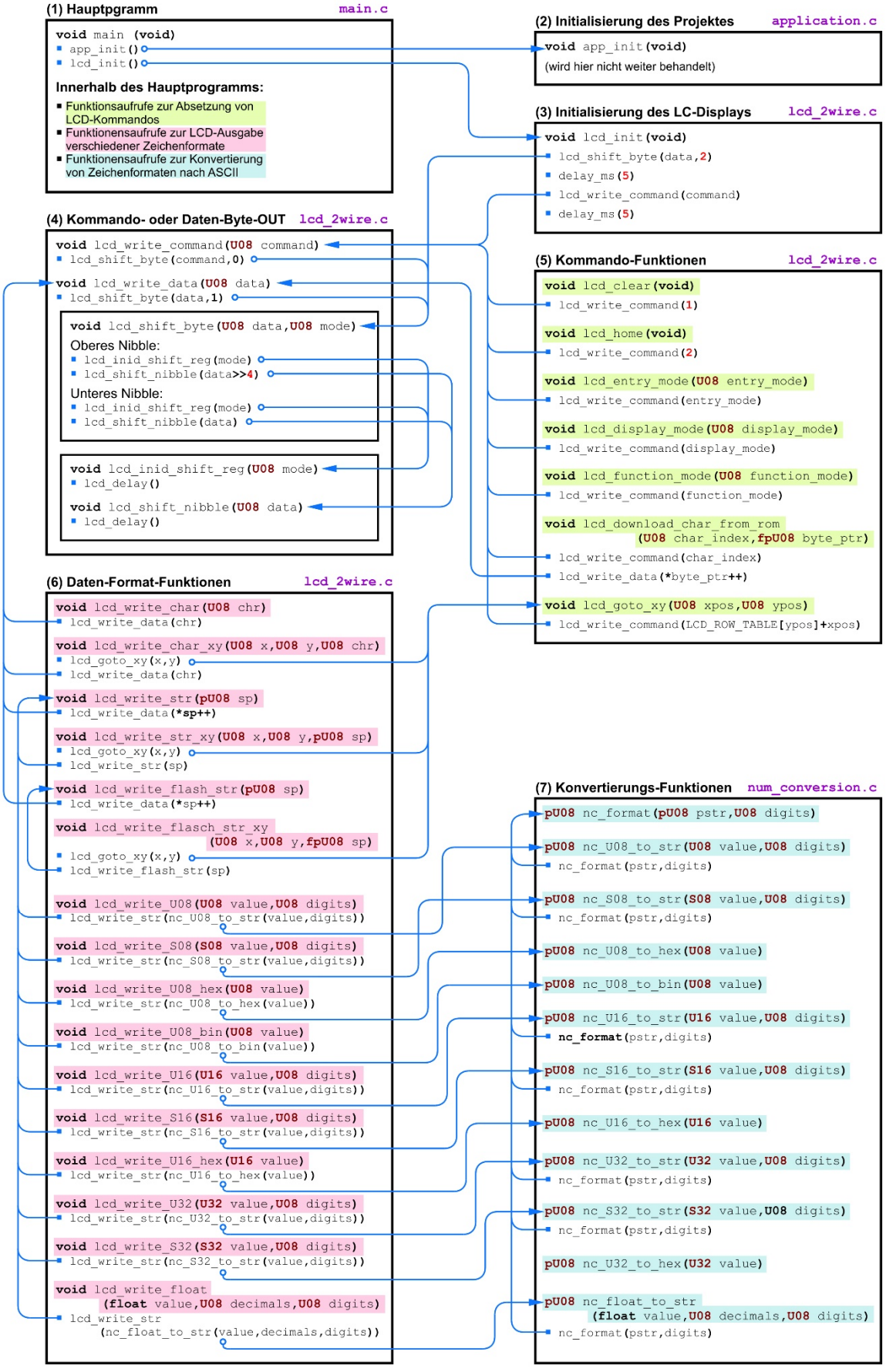



**Bild 2.2-01: Blockdiagramm des AVR-C-Projektes 2_Draht_LCD (neue Fassung)**
(Bildvergrößerung)

## 2.3 Erklärung zur Software (Steuerungs- und Übertragungs-Modi)

Wie oben dargestellt, können Kommandos ODER Daten in das LCD übertragen werden. Bei den Kommandos muss programmtechnisch noch unterschieden werden, ob es sich um solche der Initialisierung ODER solche für die individuelle Einstellung der LCD-Ansicht handelt. Für diese Unterscheidung werden 3 Modi (abgelegt in der **char**-Variable `mode`) eingeführt:

- **Modus 0** für die Übertragung eines Kommandos zur individuellen Steuerung des LCD.
  Für das Daten/Kommando-Signal **RS** muss **LOW** durch Shift auf **Q6** landen
  (Übertragung der Nibbles ausschließlich im 4-Bit-Mode).
- **Modus 1** für die Übertragung eines Daten-Bytes (in **ASCII**) zur Anzeige im LCD.
  Für das Daten/Kommando-Signal **RS** muss **HIGH** durch Shift auf **Q6** landen.
- **Modus 2** für die Übertragung eines Kommandos zur Initialisierung des LCD.
  Für das Daten/Kommando-Signal **RS** muss **LOW** durch Shift auf **Q6** landen
  (Berücksichtigung des 8-Bit-Modes).

**Hinweis:** Für das Enable-Signal **E** muss **vor** jeder Übertragung zum LCD ein **HIGH** auf **Q7** gesetzt sein!

Zur Daten-Bit-Erzeugung **DATA** für das Shift-Register werden die folgenden Befehle verwendet:

```
PORTB  |= 0x04;        // an PB2 => DATA = 1
lcd_delay();           // Verzoegerung (Dauer je nach eingestelltem
                       // System-Takt)
PORTB  &= ~0x04;       // an PB2 => DATA = 0
lcd_delay();           // Verzoegerung (Dauer je nach eingestelltem
                       // System-Takt)
```

Zur Takt-Erzeugung **CLK** für das Shift-Register werden stets die folgenden 3 Befehle benötigt:

```
PORTB  |= 0x02;        // an PB1 => CLK = 1 (steigende Impuls-Flanke,
                       // sie loest den Shift aus)
lcd_delay();           // Verzoegerung (Dauer je nach eingestelltem
                       // System-Takt)
PORTB  &= ~0x02;       // an PB1 => CLK = 0 (fallende Impuls-Flanke)
```

## 2.4 Typenfestlegung durch Synonym-Bildung

Es werden folgende Synonyme aus der Header-Datei **typedefs.h** zur Bildung abgekürzter Typen benutzt (ausnahmsweise sind auch die Kommentare bunt dargestellt):

**Farbe für Typen**
**Farbe für Synonyme**
**Farbe für Pointer als Synonyme**

```
typedef unsigned char U08;     // Bildung eines Synonyms: aus unsigned char wird U08
typedef U08 *pU08;             // Bildung eines Synonyms: pU08 steht fuer U08 *...
                               // oder als Synonym fuer unsigned char *...
                               // das heisst pU08 ist ein Pointer auf char-Variable
typedef U08 __flash *fpU08;    // Bildung eines Synonyms:
                               // fpU08 steht fuer U08 __flash *...
                               // oder als Synonym fuer unsigned char __flash *...
                               // das heisst fpU08 ist ein Pointer auf eine
                               // char-Variable im Flash-Speicher
typedef signed char S08;       // Bildung eines Synonyms: aus signed char wird S08
typedef unsigned int U16;      // Bildung eines Synonyms: U16 steht fuer
```

```
                                // unsigned int
typedef U16 *pU16;              // Bildung eines Synonyms: pU16 steht fuer U16 *...
                                // oder als Synonym fuer unsigned int *...
                                // das heisst pU16 ist ein Pointer auf eine
                                // int-Variable
typedef signed int S16;         // Bildung eines Synonyms: S16 steht fuer signed int
typedef unsigned long U32;      // Bildung eines Synonyms: U32 steht fuer
                                // unsigned long
typedef signed long S32;        // Bildung eines Synonyms: S32 steht fuer signed long
```

Daraus folgt z.B. im vorliegenden Projekt:

```
pU08 pstr;                      // diese Anweisung ist identisch mit char *pstr;
                                // oder *pstr ist ein Pointer auf eine char-Variable
```

## 2.5 Das Hauptprogramm

**Anmerkung** zur Benutzung des Moduls **LCD-Steuerung** (**lcd_2wire.c** / **lcd_2wire.h**) für dieses und für weitere (auch selbst gestrickte) Projekte :

Das Hauptprogramm **main.c** für dieses **602_Projekt_2_Draht_LCD** zur LCD-Ausgabe beinhaltet im 1. Kommentarblock einen globalen Überblick, der bei weiteren Programmen, die das LCD-Modul benutzen natürlich wegfallen kann. In den Programmteilen sind die am Ende im Zeilen-Kommentar mit **&&&&&** markierten Anweisungen gegen Anweisungen des neuen Projektes auszutauschen. Lediglich die Aufrufe der Initialisierungs-Funktionen `app_init()` und `lcd_init()` sowie die Preprozessor-Anweisungen **`#include XXXXX`** müssen übernommen werden.

Die Dateien der Module **application** (***.c** und ***.h**), **lcd_2wire** (***.c** und ***.h**) und **num_conversion** (***.c** und ***.h**) sowie die Header-Dateien **typedefs.h**, **iomx8.h**, **macros.h** und **switches.h** werden stets verwendet und sind mit dem Hauptprogramm **main.c** im zugehörigen Projekt-Ordner anzulegen. Anpassungen in der **application.h** sind von Fall zu Fall vorzunehmen.

**Zu diesem 602_Projekt_2_Draht_LCD:**

Ausführliche Erläuterungen werden im Quell-Programm **main.c** gegeben.

Der Programm-Quell-Code (**C**- und Header-Dateien) ist zu finden unter

**602_Projekt_2_Draht_LCD**

Zu Beginn werden alle Zeilen mit **Row** y beschrieben, wobei **y** die Zeilen-Nummer **0** bis **3** angibt. Am Ende jeder Zeile wird ein Zähler rasant dezimal und hexadezimal hoch gezählt. In der 1. Zeile beginnt der Zähler mit **0**, in der 2. mit **16384**, in der 3. mit **32768** und in der 4. mit **49152**, so dass das LCD zu Beginn folgendes anzeigen soll:

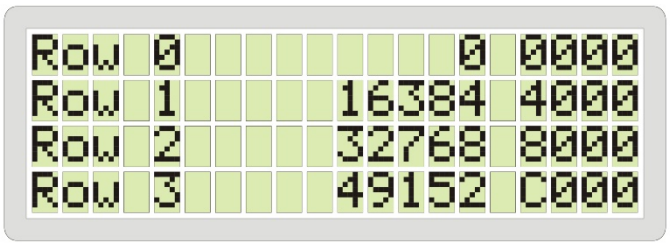



**Bild 2.5-01: LCD-Anzeige von 602_Projekt_2_Draht_LCD**

Durch Drücken der geschalteten **S1**-Taste wird das Hochzählen angehalten und mit der **RESET**-Taste wird der Zähler wieder in die Grundposition gebracht.

In der Header-Datei **`application.h`** können die Ports und Pins neu definiert werden. Für die Testboard-Beschaltung mit der Applikation sieht es so aus:

**`CLOCK`** an **PB1**
**`DATA`** an **PB2**
**`S1`** an **PB3**

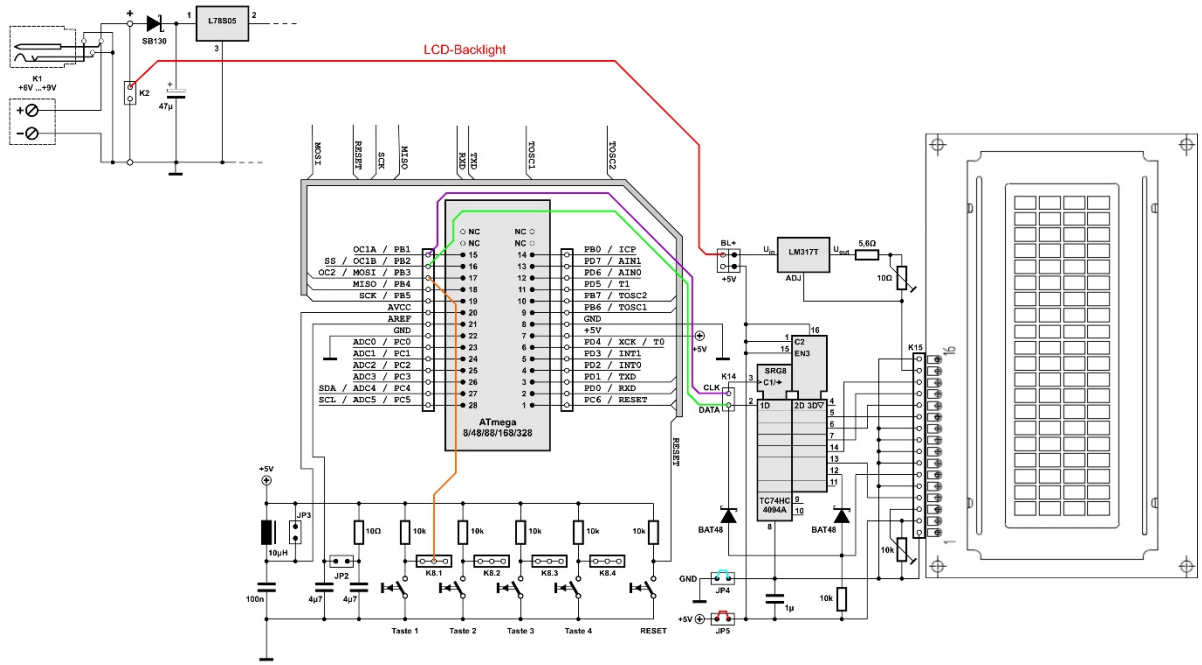



**Bild 2.5-02: Beschaltung des AVR-C-Projektes 2_Draht_LCD** (Bildvergrößerung)

## 2.6 Hinweise zu ausgewählten Funktionen

Besonderes Interesse sollte den folgenden beiden Funktionen entgegengebracht werden:

1. `void lcd_download_char_from_rom(U08 char_index, fpU08 byte_ptr)`
2. `void lcd_write_float(float value,U08 decimals,U08 digits)`

### 2.6.1. Generieren von selbst entworfenen Zeichen und ihre Anzeige am LCD

Das LCD-Modul 204B gibt die Möglichkeit, bis zu 8 selbst entworfene Zeichen im 5x7-Raster im sog. **CGRAM** abzulegen. Von hier können sie mit einem gewöhnlichen LCD-Kommando **WRITE DATA** abgerufen und zur Anzeige gebracht werden.

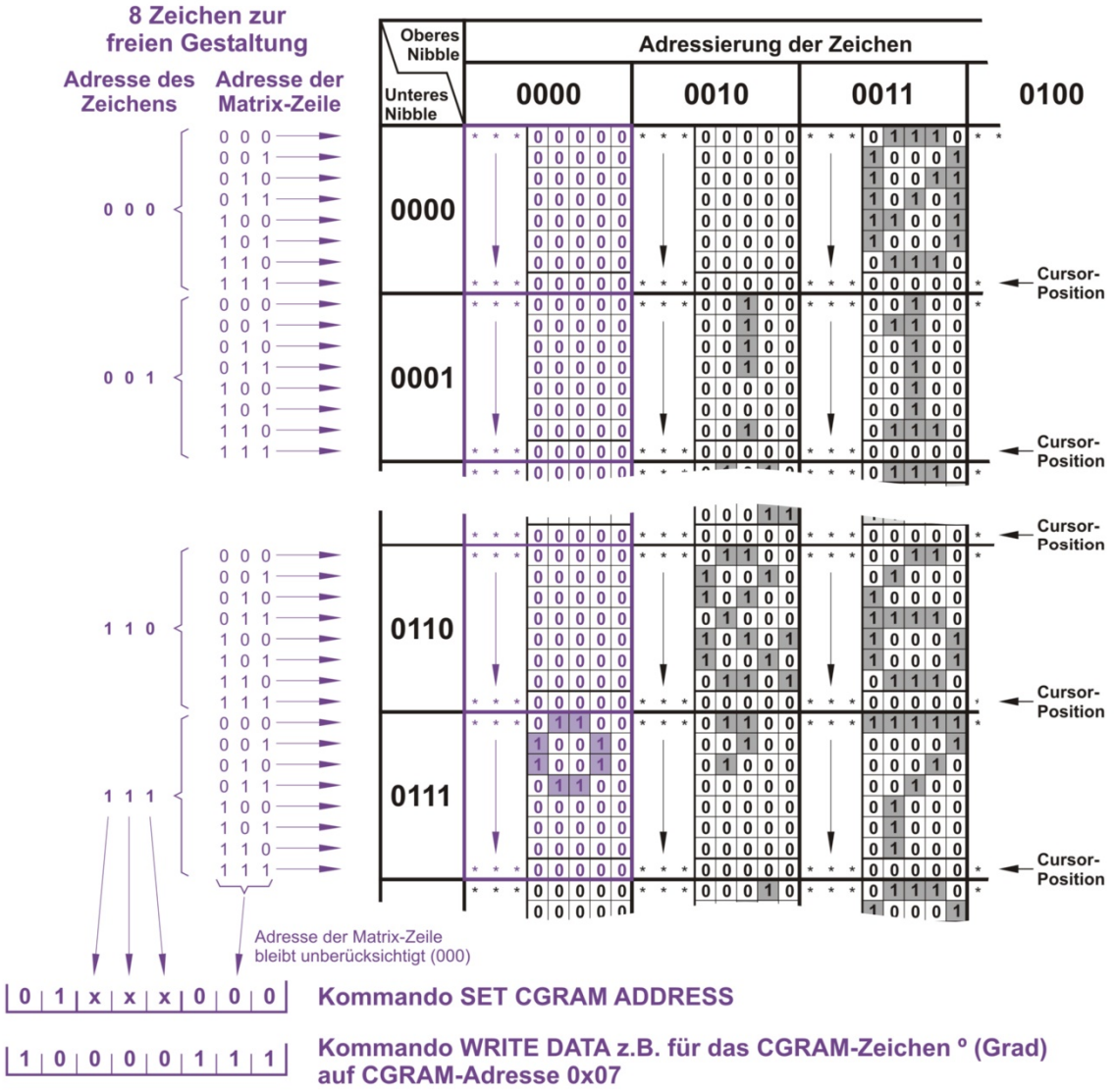



**Bild 2.6.1-01: Adress-Struktur des LCD (das Grad-Zeichen liegt hier auf Adresse 7)**
(Bildvergrößerung)

Das Bild zeigt einen Ausschnitt aus der Adress-Struktur des LCD. Der rechte "schwarze" Bereich zeigt einen Teil der nicht variablen Zeichen im **CGROM** (**C**haracter **G**enerator **R**ead **O**nly **M**emory), die mit ihrem "ASCII-Wert" über das **DDRAM** (**D**isplay **D**ata **R**andom **A**ccess **M**emory) adressiert werden. Zum Beispiel:

```
lcd_write_char(0x26) => 00100110 => '&'
```

Der linke "violette" Teil zeigt den veränderbaren Adress-Raum für die selbst entworfenen variablen Zeichen im **CGRAM** (**C**haracter **G**enerator **R**andom **A**ccess **M**emory). Diese werden direkt über das CGRAM mit **0**, **1**, ... **6** oder **7** adressiert (je nach Lage, wo man sie platziert hat). Zum Beispiel:

```
lcd_write_char(3) => 00000011 => hier nicht darstellbares Zeichen
```

Die Adressierung über DDRAM **oder** CGRAM auf das gewünschte abzubildende Zeichen wird mit den LCD-Kommandos **SET DDRAM ADDRESS** oder **SET CGRAM ADDRESS** voreingestellt. Die mit den nachfolgenden Kommandos **WRITE DATA** übermittelten Daten beziehen sich dann auf das jeweils eingestellte RAM.

Die Gestaltung eines Zeichens, d.h. sein Aussehen auf dem Display, wird durch Setzen einzelner Pixel in Form von gesetzten Bits in den Adressen der Matrix-Zeilen vollzogen. Im nachfolgenden Bild und Beispiel wird für vier verschiedene Zeichen die Pixel-Belegung gezeigt. Nach dieser Methode können bis zu 8 verschiedene Zeichen kreiert werden. Die Zeichen sind im LCD flüchtig (sie befinden sich im

RAM) und müssen, sofern man sie auch nach dem Ausschalten anzeigen will, im **FLASH**-Speicher des Mikrocontrollers abgelegt werden.

```
__flash U08 _SIGN0[8] =
{
  0x0C, 0 0 0 0 1 1 0 0
  0x12, 0 0 0 1 0 0 1 0
  0x12, 0 0 0 1 0 0 1 0
  0x0C, 0 0 0 0 1 1 0 0
  0x00, 0 0 0 0 0 0 0 0
  0x00, 0 0 0 0 0 0 0 0
  0x00, 0 0 0 0 0 0 0 0
  0x00  0 0 0 0 0 0 0 0
}
```

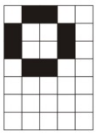


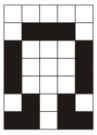
**Bild 2.6.1-02:** **Bildung von 4 selbst entworfenen Zeichen** (Bildvergrößerung)

**Das Grad-Zeichen**

```
__flash U08 _SIGN1[8] =
{
  0x0A, 0 0 0 0 1 0 1 0
  0x15, 0 0 0 1 0 1 0 1
  0x0A, 0 0 0 0 1 0 1 0
  0x15, 0 0 0 1 0 1 0 1
  0x0A, 0 0 0 0 1 0 1 0
  0x15, 0 0 0 1 0 1 0 1
  0x0A, 0 0 0 0 1 0 1 0
  0x00  0 0 0 0 0 0 0 0
}
```

**Das Schachbrett-Zeichen**

```
__flash U08 _SIGN2[8] =
{
  0x00, 0 0 0 0 0 0 0 0
  0x0E, 0 0 0 0 1 1 1 0
  0x11, 0 0 0 1 0 0 0 1
  0x11, 0 0 0 1 0 0 0 1
  0x11, 0 0 0 1 0 0 0 1
  0x0A, 0 0 0 0 1 0 1 0
  0x1B, 0 0 0 1 1 0 1 1
  0x00  0 0 0 0 0 0 0 0
}
```



**Das Ohm-Zeichen (Omega)**

```
__flash U08 _SIGN3[8] =
{
  0x17, 0 0 0 1 0 1 1 1
  0x14, 0 0 0 1 0 1 0 0
  0x16, 0 0 0 1 0 1 1 0
  0x14, 0 0 0 1 0 1 0 0
  0x17, 0 0 0 1 0 1 1 1
  0x10, 0 0 0 1 0 0 0 0
  0x1F, 0 0 0 1 1 1 1 1
  0x00  0 0 0 0 0 0 0 0
}
```

**Das LE-Zeichen**

**Anwendung** der Funktion:

```
void lcd_download_char_from_rom(U08 char_index,fpU08 byte_ptr)

// Aufruf: lcd_download_char_from_rom(x,FlashPointer);
// x => 3 Bits fuer CGRAM-Adressteil des Sonder-Zeichens
// x = 0 ... 7 ist gleichzeitig die Ansteuerungs-Nummer des Sonder-Zeichens
// (statt ASCII) mit dem Kommando WRITE DATA
// (Es können bis zu 8 Sonder-Zeichen moduliert werden)
// FlashPointer => Pointer auf Punkt-Matrix des Zeichens(8 Byte) im FLASH-Speicher
```

Der Programm-Quell-Code des folgenden Programms (**C**- und Header-Dateien) ist zu finden unter

**602_Programm_Sonderzeichen**

**Beispiel 2.6.1-01: Selbst entworfene Zeichen auf dem LCD ausgeben**

```
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// Abbilden von Sonderzeichen auf dem LCD
// Zusaetzliche Header..: keine
// Module.(*.c und *.h).: application, lcd_2wire, num_conversion
// Benoetigte Hardware..: Testboard, 20x4-LCD, 2-Draht-LCD-Interface
// Version.............: 1.1
// Compiler............: CodeVisionAVR
// Chip................: ATmega88
// Datum...............: Juni 2009
// Autor...............: ALE23 basierend auf dem Projekt LCD von Udo Juerss
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// Erzeugen/Anzeigen von Sonderzeichen auf dem 20x4-LCD
//
// Anschluesse des 20x4-LCD:
```

```
// CLOCK.................: PB1 (kann geaendert werden in "application.h")
// DATA..................: PB2 (kann geaendert werden in "application.h")
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// Verwendete Typ-Synonyme (werden in typedefs.h deklariert)
// U08    unsigned char
// fpU08  unsigned char __flash *...// Pointer auf unsigned char im FLASH
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// Header-Dateien einfuegen:
#include "application.h"
#include "lcd_2wire.h"
#include "num_conversion.h"
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
fpU08 LCD_STR[4] =                              // Display-Text im FLASH:
{
  ("Grad-Zeichen:      a"),                     // a wird ersetzt durch 
  ("Schachbrett:       b"),                     // b wird ersetzt durch 
  ("Ohm-Zeichen:       c"),                     // c wird ersetzt durch 
  ("LE-Zeichen:        d")                      // d wird ersetzt durch 
};
//-----------------------------------------------------------------------------

U08 n;
__flash U08 _SIGN0[8] =
{
  0x0C,0x12,0x12,0x0C,0x00,0x00,0x00,0x00       // Grad-Zeichen
};
__flash U08 _SIGN1[8] =
{
  0x0A,0x15,0x0A,0x15,0x0A,0x15,0x0A,0x00       // Schachbrett-Zeichen
};
__flash U08 _SIGN2[8] =
{
  0x00,0x0E,0x11,0x11,0x11,0x0A,0x1B,0x00       // Ohm-Zeichen
};
__flash U08 _SIGN3[8] =
{
  0x17,0x14,0x16,0x14,0x17,0x10,0x1F,0x00       // LE-Zeichen
};
//-----------------------------------------------------------------------------

void main(void)
{
  app_init();                                   // Initialisierung der Applikation
  lcd_init();                                   // Initialisierung des LC-Displays

// Setzen der Sonder-Zeichen im LCD-Controller
  lcd_download_char_from_rom(0,(fpU08)&_SIGN0); // auf Grad-Zeichen
  lcd_download_char_from_rom(1,(fpU08)&_SIGN1); // auf Schachbrett
  lcd_download_char_from_rom(2,(fpU08)&_SIGN2); // auf Ohm-Zeichen
  lcd_download_char_from_rom(3,(fpU08)&_SIGN3); // auf LE-Zeichen

  while (true)                                  // Endlosschleife
  {
    for (n = 0; n < 4; n++)                     // Schreibe Display-Text
    {
      lcd_goto_xy(0,n);                         // Setze Cursor an Zeilen-Anfang
      lcd_write_flash_str((fpU08)LCD_STR[n]);   // Begleittext der Zeile
      lcd_goto_xy(19,n);                        // An das Ende der Zeile
      lcd_write_char(n);                        // Sonder-Zeichen (n=Adr. im CGRAM)
    }
  }
}
```


**Bild 2.6.1-03: LCD-Anzeige von 602_Programm_Sonderzeichen**

## 2.6.2 Einschränkungen bei der LCD-Ausgabe einer Gleitkomma-Zahl

Im Abschnitt **4.2.2 Datentypen** von **Teil 504 - Syntax der C-Programmiersprache** wurde ausführlich auf die **Besonderheit der Gleitkomma-Zahlen (`float` oder `double`)** eingegangen. Zur Wiederholung:

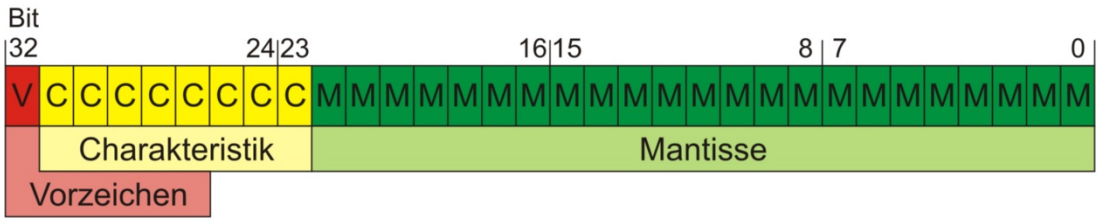



**Bild 2.6.2-01: Darstellung der Gleitkomma-Zahlen im AVR-Mikrocontroller ATmega88**

Bei der **Charakteristik** (auch **Biased Exponent** genannt) handelt es sich nicht um einen "tatsächlichen" Exponenten. Denn, um auch negative Exponenten darstellen zu können, ohne ein weiteres Vorzeichen-Bit zu opfern, wurde die Charakteristik in der Normung für Gleitkomma-Zahlen gem. **IEEE 754** besonders definiert (für die o. b. Darstellung der 8-Bit-Controller):

| | | | | | |
|---|---|---|---|---|---|
| 0 bis 126 oder **00000000** ... **01111110** für negative Exponenten | von | -126 | bis | -1 |
| 128 bis 255 oder **10000000** ... **11111111** für positive Exponenten | von | +1 | bis | +128 |

Gleichzeitig ist der zwischen beiden Bereichen liegende **Bias** auf dem Wert **127** oder **01111111** festgelegt worden, so dass Zahlen mit den 2er-Potenzen von $\pm 2^{-126}$ bis $\pm 2^{128}$ dargestellt werden können, d.h. es wird ein Wertebereich von ca. $\pm 1{,}175 \cdot 10^{-38}$ bis $\pm 3{,}402 \cdot 10^{38}$, abgedeckt. Das ist ein "riesiger" Zahlenbereich. **ABER:** Die Mantisse aus 23 Bits plus Hidden-Bit kann nur Werte zwischen **0** und **16777215** (das ist $2^{24}$-1) darstellen, was darüber hinaus geht wird abgeschnitten.

**Merke:** Wird eine Zahl behandelt, die eine höhere Mantisse als **16777215** aufweist (z.B. durch Vorgabe einer Konstanten), so sind die 8. und ggf. folgende Ziffern (unabhängig von der Stellung des Kommas) nicht mehr gesichert! Nur 7 aufeinander folgende Ziffern sind wirklich garantiert.

Bei der Darstellung einer Gleitkomma-Zahl auf dem LCD sollte man auf diese Beschränkung achten, wenn man nicht von "Phantasie-Zahlen" überrascht werden will. Die Funktion

```
void lcd_write_float(float value,U08 decimals,U08 digits)

// Aufruf: lcd_write_float(Gleitkomma-Zahl,d-Bruchstellen,n-Zeichen);
// Wertebereich: +/- 1.175e-35 ... +/- 3.402e38
// ABER nur 7 Stellen aufeinanderfolgender Ziffern sind garantiert.
// d-Bruchstellen: maximal 5 darzustellende Stellen nach dem Komma
// n-Zeichen: Anzahl darzustellender Stellen insgesamt
```

und die von ihr aufgerufene (Unter-)Funktion im Modul **`num_conversion.c`** / **`num_conversion.h`**

```
pU08 nc_float_to_str(float value,U08 decimals,U08 digits)

// Aufruf: nc_float_to_str(Gleitkomma-Zahl,d-Bruchstellen,n-Zeichen);
// Wandle eine 32-Bit-Gleitkomma-Zahl in ein ASCII-String um
// d-Bruchstellen: maximal 5 darzustellende Stellen nach dem Komma
// n-Zeichen: Anzahl darzustellender Stellen insgesamt
```

sehen nur eine Beschränkung der Anzahl Dezimalstellen nach dem Komma auf **5** vor. Die Gesamtzahl der Stellen kann auf die maximale Zeilenlänge von 20 Zeichen ausgedehnt werden, so dass bei vielen aufeinander folgenden Ziffern schnell ein falscher Eindruck der Genauigkeit der Anzeigegröße entstehen kann.

Der Programm-Quell-Code des folgenden Programms (**C**- und Header-Dateien) ist zu finden unter

**602_Programm_Gleitkomma**.

**Beispiel 2.6.2-01: Konvertierung einer Gleitkomma-Zahl**

```
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// Konvertierung einer Gleitkomma-Zahl
// Zusaetzliche Header..: keine
// Module.(*.c und *.h).: application, lcd_2wire, num_conversion
// Benoetigte Hardware..: Testboard, 20x4-LCD und 2-Draht-LCD-Interface
// Version.............: 1.1
// Compiler............: CodeVisionAVR
// Chip................: ATmega88
// Datum...............: Juni 2009
// Autor...............: ALE23 basierend auf dem Projekt LCD von Udo Juerss
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// Darstellung einer Gleitkomma-Zahl auf dem LCD
//
// Anschluesse des 20x4-LCD:
// CLOCK...............: PB1 (kann geaendert werden in "application.h")
// DATA................: PB2 (kann geaendert werden in "application.h")
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// Verwendete Typ-Synonyme (werden in typedefs.h deklariert)
// U08    unsigned char
// U32    unsigned long int
// fpU08  unsigned char __flash *...// Pointer auf unsigned char im FLASH
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// Header-Dateien einfuegen:
#include "application.h"
#include "lcd_2wire.h"
#include "num_conversion.h"
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------

fpU08 LCD_STR[4] =                  // Vordefinierter Display-Text im FLASH
{
  ("String:   +12345.678"),         // Wiedergabe der Gleitkomma-Zahl vom String
  ("Vor:         xxxxxxx"),         // Wert vor dem Dezimalpunkt
  ("Nach:        zzzzzzz"),         // Wert nach dem Dezimalpunkt
  ("Float:     ffffffff")           // Darstellung der Gleitkomma-Zahl
};                                  // max. Vorzeichen, 7 Ziffern, Dezimalpunkt
//-----------------------------------------------------------------------------
// Zu wandelnder Zahlenwert - in der Mantisse max. darstellbarer Wert 16777215
float float_wert  =    -1677.7215;  // Beispiel: Zu wandelnde Gleitkomma-Zahl
U08 char_wert[13] = "  -1677.7215"; // Beispiel: Wiederholung als String

U08   n, i;                         // Zaehler
U32   int_wert;                     // Integer-Wert, d.h. Wert vor dem Punkt
float dec_wert;                     // Ziffernfolge nach dem Dezimalpunkt
//-----------------------------------------------------------------------------

void main(void)
{
  app_init();                       // Initialisierung der Applikation
  lcd_init();                       // Initialisierung des LC-Displays

  for (n = 0; n < 4; n++)           // Schreibe 4-zeiligen Display-Text
  {
```

```
    lcd_goto_xy(0,n);
    lcd_write_flash_str((fpU08)LCD_STR[n]);
  }

  for (n = 11; n > 0; n--)          // Ermitteln Anzahl Ziffern nach Dezimalpunkt
  {
    if(char_wert[n] == '.')         // Punkt gefunden?
    {                               // JA: hoechster Index vom Array minus n
      i = 11-n;                     // ist i Anzahl Ziffern nach Dezimalpunkt
      if(i > 5) i = 5;              // i darf nur max. 5 sein
      else;
    }
    else;                           // NEIN: Punkt suchen
  }
  while (1)                         // Unendliche Schleife
  {
    lcd_goto_xy(8,0);               // x = 8, y = 0 (1. Zeile)
    lcd_write_str(char_wert);       // Zahl als String darstellen

    if (float_wert < 0.0)           // Abfrage: Ist Gleitkomma-Zahl negativ?
    {                               // Wenn JA:
      int_wert = -float_wert;       // float in pos. U32 implizit wandeln
      dec_wert = (-float_wert) - int_wert; // Ziffern nach Punkt ermitteln
      lcd_goto_xy(8,2);             // x = 8, y = 2 (3. Zeile)
      lcd_write_float(dec_wert,i,12);      // i Ziffern nach Punkt darstellen
    }
    else                            // Wenn NEIN (nicht negativ):
    {
      int_wert = float_wert;        // float in U32 implizit wandeln
      dec_wert = float_wert - int_wert;    //Ziffern nach Punkt ermitteln
      lcd_goto_xy(8,2);             // x = 8, y = 2 (3. Zeile)
      lcd_write_float(dec_wert,i,12);      // i Ziffern nach Punkt darstellen
    }

    lcd_goto_xy(8,1);               // x = 8, y = 1 (2. Zeile)
    lcd_write_U32(int_wert,12);     // Ziffern vor dem Punkt darstellen

    lcd_goto_xy(8,3);               // x = 8, y = 3 (4. Zeile)
    lcd_write_float(float_wert,i,12);      // Gleitkomma-Zahl darstellen
  }
}
```

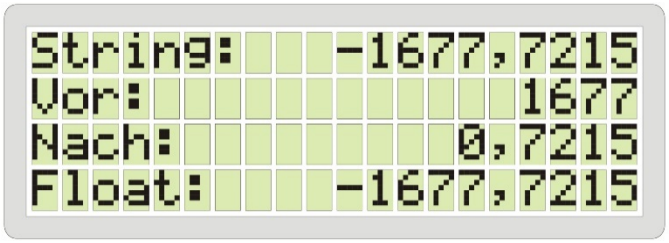



**Bild 2.6.2-02: LCD-Anzeige von 602_Programm_Gleitkomma**

## 2.6.3 Endlich: "Hello World!"

Jetzt soll es endlich dazu kommen, dass wir auch den vielzitierten Anfangsspruch der Programmier-Eleven erzeugen können. Wie aus den oben gezeigten Projekt-Beispielen erfahren, lässt sich auch das triviale "Begrüßungswort" ohne viele Worte auf das Display zaubern.


Der Programm-Quell-Code des Programms (**C**- und Header-Dateien) ist zu finden unter

**602_Programm_Hello_World**

**Beispiel 2.6.3-01: Textausgabe verteilt über 4 Zeilen**

```
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// Begruessung der "Eleven" auf dem LCD
// Zusaetzliche Header..: keine
// Module.(*.c und *.h).: application, lcd_2wire, num_conversion
// Benoetigte Hardware..: Testboard, 20x4-LCD und 2-Draht-LCD-Interface
// Version..............: 1.1
// Compiler.............: CodeVisionAVR
// Chip.................: ATmega88
// Datum................: Maerz 2020
// Autor................: ALE23 basierend auf dem Projekt
//                        2-Draht-LCD von Udo Juerss
//-----------------------------------------------------------------------------
//-----------------------------------------------------------------------------
// Darstellung von Text auf dem 20x4-LCD
//
// Anschluesse des 20x4-LCD:
// CLOCK................: PB1 (kann geaendert werden in "application.h")
// DATA.................: PB2 (kann geaendert werden in "application.h")

//-----------------------------------------------------------------------------
// Verwendete Typ-Synonyme (werden in typedefs.h definiert):
//-----------------------------------------------------------------------------
// fpU08 unsigned char __flash *...     // Pointer auf unsigned char im FLASH
//-----------------------------------------------------------------------------
// Header-Dateien einfuegen:
//-----------------------------------------------------------------------------
#include "application.h"                // Initialisierung der Anwendung
#include "lcd_2wire.h"                  // Initialisierung des 2-Draht_LCDs
#include "num_conversion.h"             // Initialisierung der Konvertierung

//-----------------------------------------------------------------------------
// Globale Variablen:
//-----------------------------------------------------------------------------
fpU08 LCD_STR[4] =                      // festgelegter Display-Text im FLASH
{
  ("        Wie         "),            // Reiner Text ohne Besonderheiten
  ("    versprochen:    "),            // und ohne Einfuegungen
  ("       Hello,       "),            // dto.
  ("       World!       ")             // (man darf sich nur nicht verzaehlen)
};

U08    n;                               // Zaehler

//-----------------------------------------------------------------------------
// Hauptprogramm main (nur festgelegte Textausgabe):
//-----------------------------------------------------------------------------
void main(void)
{
  app_init();                           // Initialisierung der Applikation
  lcd_init();                           // Initialisierung des LC-Interfaces

  for (n = 0; n < 4; n++)               // Schreibe 4-zeiligen Display-Text
  {
    lcd_goto_xy(0,n);                   // Setze Cursor an LCD-Anfang
    lcd_write_flash_str((fpU08)LCD_STR[n]);
  }
}
//-----------------------------------------------------------------------------
```