­

**PROTOKOLL**zur Laborübung

***AVR LCD***



|  |  |  |
| --- | --- | --- |
| Gruppe / Klasse | Protokollführer | Unterschrift |
| 5 / **4BHELS** | HOFSTÄTTER A. |  |
| Übungs- / Abgabedatum | Mitarbeiter | Unterschrift |
| 14. April 2015  16. April 2015 | BIEHL S. |  |
| Lehrer | Mitarbeiter | Unterschrift |
| CRHA |  |  |
| Note | Mitarbeiter | Unterschrift |
|  |  |  |
| ***AVR LCD***  *ATmega32U4* | | |
| **Verwendete Geräte**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Nr. | Gerätebezeichnung | Hersteller | Typ | Platznummer | | - | - | - | - | - |   **Verwendete Programme**   |  |  |  | | --- | --- | --- | | Nr. | Name | Version | | 1. | CodeBlocks | 13.12 | | 2. | DFU-Programmer | 1.2.2 | | | |

ÜBUNGS-/ABGABE-DATUM

Klasse /Gruppe

NOTE

LEHRER

# Inhaltsverzeichnis

[1 Inhaltsverzeichnis 2](#_Toc416941103)

[2 Aufgabenstellung 3](#_Toc416941104)

[2.1 Messablauf 3](#_Toc416941105)

[3 Allgemeines 4](#_Toc416941106)

[3.1 Display 4](#_Toc416941107)

[3.1.1 Anzeige 4](#_Toc416941108)

[3.1.2 Verbindung zum µC 4](#_Toc416941109)

[4 Grundprogramm 5](#_Toc416941110)

[5 Programmierung der Aufgaben 9](#_Toc416941111)

[5.1 Zeichen ausgeben 9](#_Toc416941112)

[5.2 Wort ausgeben 9](#_Toc416941113)

[5.3 Wort blinken lassen 9](#_Toc416941114)

[5.4 Wort in einer Zeile scrollen 9](#_Toc416941115)

[5.5 Eigenes Zeichen kreieren und anzeigen 10](#_Toc416941116)

[5.6 Von 0 bis 100 zählen 10](#_Toc416941117)

[6 Program-Listing Verzeichnis 11](#_Toc416941118)

# Aufgabenstellung

Aufgabe dieser Laborübung war es mit Hilfe eines ATmega32U4 ein LCD-Display anzusteuern. Folgende Aspekte waren dabei zu erfüllen

* LCD im 4 Bit Modus konfigurieren
* Ausgabe eines einzelnen Zeichens
* Ausgabe eines beliebigen Wort
* Laufschrift in einer Zeile
* Ausgabe eines eigens erstelltem Zeichen
* Zählen von 0 bis 100

## Messablauf

Zuerst wurde das LCD-Display mit dem AVR verbunden. Die Pinbelegung wurde so gewählt und angeschlossen, dass der Bildschirm im 4 Bit Modus angesteuert werden konnte.

Anschließend wurde ein Programm im C-Code geschrieben und durch den avr-gcc kompiliert. Mit Hilfe des „DFU-Programmers“ wurde das kompilierte und umgewandelte HEX File auf den AVR geflasht.

**Compiler Aufrufe unter UNIX**

**avr-gcc** main.c -Os -std=c99 -o output.elf -mmcu=atmega32u4;

**avr-objcopy** -R .eeprom -R .fuse -R .lock -R .signature -O ihex output.elf output.hex

**Loadhex.sh** atmega32u4 erase flash output.hex start

**Inhalt des benutzerdefinierten Scripts “LoadHex.sh” zum flashen unter UNIX**

**dfu-programmer $1 $2**

**dfu-programmer $1 $3 $4**

**dfu-programmer $1 $5**

**exit**

# Allgemeines

## Display

### Anzeige

Das verwendete LCD-Display hat zwei Zeilen mit je 20 Zeichen. Ein Zeichen besteht aus 5\*8 Pixel. Das Display unterstützt die Zeichen des ASCII-Codes, wodurch sich die Ansteuerung wesentlich vereinfacht.

### Verbindung zum µC

|  |  |
| --- | --- |
| AVR | Display |
| PB0 – PB3 | D4 – D7 |
| PB6 | RS |
| PB5 | RW |
| PB4 | Enable |

Da das Display im 4Bit Modus betrieben wurde, sind nur die Datenleitungen D4 bis D7 notwendig.

RS steht für Register Select. RS = 0 -> Einstellungsregister, RS = 1 -> Datenregister Bei einer steigenden Flanke am Enable-Pin werden die Daten übernommen.

Der Lese/Schreib-Modus wurde hardwaremäßig auf Write gestellt. Zusätzlich wurde das Display mit 5V (USB) versorgt.

# Grundprogramm

Im Folgenden ist das Grundprogramm zum Ansteuern des Displays ersichtlich. Es sind etliche Funktionen definiert, um die Ansteuerung im Hauptprogramm so einfach wie möglich zu gestalten. Die wichtigsten Funktionen sind cmd zum Befehle senden und data bzw. string zum Senden der Daten.

//////////////////////////////////////////////////////////////////////////////

//

// Headefile LCD.h zur LCD Ansteuerung am Port B für das HTL Demoboard

//

// Funktionen: initialisieren(), cmd(char data), senden(char data)

// string(char \*data)

//

// initialisieren(); initialisiert Port D

// und LCD im 4-Bit Mode, 2 Zeilen, 5x7 Dots

// - Bsp. Aufruf: initialisieren();

//

// cmd(char data); schickt Befehl ans LCD

// - Bsp. Aufruf: cmd(0xC5); //gehe zu 2. Zeile, 6. // Position

//

// senden(char data); schickt Daten ans LCD

// - Bsp. Aufruf: senden(0xEF); //sendet ein ö

//

// string(char \*data); schickt eine Zeichenkette ans LCD

// - Bsp. Aufruf: string("Hallo"); //sendet Hallo

//

// Pinbelegung am Board:

// LCD | Atmega32U4 | Bemerkung

// ----|------------|--------------------

// DB7 | PB3 |

// DB6 | PB2 |

// DB5 | PB1 |

// DB4 | PB0 |

// DB3 | - | wird nicht benötigt

// DB2 | - | wird nicht benötigt

// DB1 | - | wird nicht benötigt

// DB0 | - | wird nicht benötigt

// E | PB4 |

// RW | PB5 |

// RS | PB6 |

// VO | PB7 |

// VDD | GND |

// VSS | 5V |

//

////////////////////////////////////////////////////////////////////////////////

//-- Hier die Pinzuordnung bei Bedarf aendern (siehe Tabelle oben) --//

#define DB7 PB3

#define DB6 PB2

#define DB5 PB1

#define DB4 PB0

#define E PB4

#define RW PB5

#define RS PB6

#define VO PB7

// PORT Zuordnung muss im gesamten LCD.h angepasst werden!

// d.h. PORTB und DDRB durch gewünschten anderen Port ersetzen

//

//------------------- Ende Pinzuordnung ------------------------------//

#include <avr/io.h>

#include <util/delay.h> // \_delay\_ms() geht nur bis max. 262.14 ms / F\_CPU

void delay\_ms (unsigned int ms) //Hilfsfunktion: Zeitvernichtung

{ int i;

for (i=0; i<ms; i++)

{

\_delay\_ms(1);

}

}

void Enable(void) //Hilfsfunktion: H=>L Flanke der Enable Leitung (E)

{

PORTB = PORTB | (1<<E); //E = 1

delay\_ms(5);

PORTB = PORTB &~(1<<E); //E = 0

delay\_ms(5);

}

////////////////////////////////////////////////////////////////////////////

// initialisieren(..) Initialisierung: Port D, 4-Bit Mode, 2 Zeilen, 5x7 Dots

void initialisieren(void)

{

DDRB = DDRB |(1<<E) |(1<<RS) |(1<<RW); //E,RS,RW als Ausgang

DDRB = DDRB |(1<<DB7)|(1<<DB6)|(1<<DB5)|(1<<DB4); //DB7..DB4 als Ausgang

delay\_ms(50); //lt. Datenblatt min. 15ms nach Power ON warten

PORTB = PORTB & (~(1<<E) & ~(1<<RS) & ~(1<<RW)); //RS=0,E=0,RW=0

// Function Set

//DB7..DB4 = 0011

PORTB = PORTB & (~(1<<DB7) & ~(1<<DB6)); //Interface auf 8 Bit

PORTB = PORTB | (1<<DB5) | (1<<DB4);

Enable();

//DB7..DB4 = 0011

PORTB = PORTB & (~(1<<DB7) & ~(1<<DB6)); //Interface auf 8 Bit

PORTB = PORTB | (1<<DB5) | (1<<DB4);

Enable();

//DB7..DB4 = 0011

PORTB = PORTB & (~(1<<DB7) & ~(1<<DB6)); //Interface auf 8 Bit

PORTB = PORTB | (1<<DB5) | (1<<DB4);

Enable();

//DB7..DB4 = 0010

PORTB = PORTB & (~(1<<DB7) & ~(1<<DB6) &~(1<<DB4));

PORTB = PORTB | (1<<DB5); //Interface auf 4 Bit

Enable();

// 2-zeilig, 5x8 Matrix //

//DB7..DB4 = 0010

PORTB = PORTB & (~(1<<DB7) & ~(1<<DB6) &~(1<<DB4));

PORTB = PORTB | (1<<DB5); //Upper Nibble

Enable();

//DB7..DB4 = 1000

PORTB = PORTB | (1<<DB7); //Lower Nibble

PORTB = PORTB & (~(1<<DB6) & ~(1<<DB5) & ~(1<<DB4));

Enable();

//Display Off //

//DB7..DB4 = 0000

PORTB = PORTB & (~(1<<DB7) & ~(1<<DB6) & ~(1<<DB5) & ~(1<<DB4)); //Upper Nibble

Enable();

//DB7..DB4 = 1000

PORTB = PORTB | (1<<DB7); //Lower Nibble

PORTB = PORTB & (~(1<<DB6) & ~(1<<DB5) & ~(1<<DB4));

Enable();

//Clear Display //

//DB7..DB4 = 0000

PORTB = PORTB & (~(1<<DB7) & ~(1<<DB6) & ~(1<<DB5) & ~(1<<DB4)); //Upper Nibble

Enable();

//DB7..DB4 = 0001

PORTB = PORTB & (~(1<<DB7) & ~(1<<DB6) & ~(1<<DB5)); //Lower Nibble

PORTB = PORTB | (1<<DB4);

Enable();

//No Display Shift //

//DB7..DB4 = 0000

PORTB = PORTB & (~(1<<DB7) & ~(1<<DB6) & ~(1<<DB5) & ~(1<<DB4)); //Upper Nibble

Enable();

//DB7..DB4 = 0011

PORTB = PORTB & (~(1<<DB7) & ~(1<<DB6)); //Lower Nibble

PORTB = PORTB | (1<<DB5) | (1<<DB4);

Enable();

// Display ON , Cursor ON, Blinken ON //

//DB7..DB4 = 0000

PORTB = PORTB & (~(1<<DB7) & ~(1<<DB6) & ~(1<<DB5) & ~(1<<DB4)); //Upper Nibble

Enable();

//DB7..DB4 = 1111

PORTB = PORTB | (1<<DB7) | (1<<DB6) | (1<<DB5) | (1<<DB4); //Lower Nibble

Enable();

}

////////////////////////////////////////////////////////////////////////////

// senden(..) sendet 1 Byte im 4-Bit Mode

void senden(char data)

{

char temp = data;

PORTB = PORTB | (1<<RS); //SFR vom LCD mit RS auf Daten umschlten

//Upper Nibble senden

if (temp & 0b10000000) {PORTB = PORTB | (1<<DB7);}

else {PORTB = PORTB & ~(1<<DB7);}

if (temp & 0b01000000) {PORTB = PORTB | (1<<DB6);}

else {PORTB = PORTB & ~(1<<DB6);}

if (temp & 0b00100000) {PORTB = PORTB | (1<<DB5);}

else {PORTB = PORTB & ~(1<<DB5);}

if (temp & 0b00010000) {PORTB = PORTB | (1<<DB4);}

else {PORTB = PORTB & ~(1<<DB4);}

Enable();

delay\_ms(1);

//Lower Nibble senden

if (temp & 0b00001000) {PORTB = PORTB | (1<<DB7);}

else {PORTB = PORTB & ~(1<<DB7);}

if (temp & 0b00000100) {PORTB = PORTB | (1<<DB6);}

else {PORTB = PORTB & ~(1<<DB6);}

if (temp & 0b00000010) {PORTB = PORTB | (1<<DB5);}

else {PORTB = PORTB & ~(1<<DB5);}

if (temp & 0b00000001) {PORTB = PORTB | (1<<DB4);}

else {PORTB = PORTB & ~(1<<DB4);}

Enable();

delay\_ms(1);

}

////////////////////////////////////////////////////////////////////////////

// cmd(..) Befehl senden im 4-Bit Mode

void cmd(char data)

{

char temp = data;

PORTB = PORTB & ~(1<<RS); //SFR vom LCD mit RS auf Befehle umschalten

//Upper Nibble senden

if (temp & 0b10000000) {PORTB = PORTB | (1<<DB7);}

else {PORTB = PORTB & ~(1<<DB7);}

if (temp & 0b01000000) {PORTB = PORTB | (1<<DB6);}

else {PORTB = PORTB & ~(1<<DB6);}

if (temp & 0b00100000) {PORTB = PORTB | (1<<DB5);}

else {PORTB = PORTB & ~(1<<DB5);}

if (temp & 0b00010000) {PORTB = PORTB | (1<<DB4);}

else {PORTB = PORTB & ~(1<<DB4);}

Enable();

delay\_ms(1);

//Lower Nibble senden

if (temp & 0b00001000) {PORTB = PORTB | (1<<DB7);}

else {PORTB = PORTB & ~(1<<DB7);}

if (temp & 0b00000100) {PORTB = PORTB | (1<<DB6);}

else {PORTB = PORTB & ~(1<<DB6);}

if (temp & 0b00000010) {PORTB = PORTB | (1<<DB5);}

else {PORTB = PORTB & ~(1<<DB5);}

if (temp & 0b00000001) {PORTB = PORTB | (1<<DB4);}

else {PORTB = PORTB & ~(1<<DB4);}

Enable();

delay\_ms(1);

}

////////////////////////////////////////////////////////////////////////////

// string(..) sendet ganzen String im 4-Bit Mode

void string(char \*data)

{

while (\*data != '\0') //bis zum letzten Zeichen

{senden(\*data++);}

}

////////////////////////////////////////////////////////////////////////////

// Daten im 4-Bit Mode zum LCD senden

void data (char wert)

{

IOD = IOD | (1<<RS); //SFR vom LCD mit RS=1 auf Daten umschalten

LCD\_send(wert);

}

Listing 1. - Header File zur Ansteurung

# Programmierung der Aufgaben

## Zeichen ausgeben

cmd(0x0C); //Display ON, Cursor OFF, Blinking OFF

string("a");

Listing 2. - Zeichen ausgeben

Diese zwei Codezeilen stehen im Hauptprogramm nach der Initialisierungsroutine und sorgen dafür, dass das Zeichen „a“ am Display zu sehen ist. Für den genauen Code der Funktion cmd() oder string() siehe 4 Grundprogramm.

## Wort ausgeben

cmd(0x0C);   
string ("cc – labor 4bhels");

Listing 3. - Wort ausgeben

Mit dieser Codezeile wird am Bildschirm „cc – labor 4bhels“ ausgegeben.

## Wort blinken lassen

cmd(0x0C);

string("cc – labor 4bhels ");

while(1)

{

cmd(0x08); // Display aus

delay\_ms(750); // ca. 750ms warten

cmd(0x0C); // Display an

delay\_ms(705);

}

Listing 4. - Blinken

Der senden des HEX Wertes 0x08 schaltet das Display aus, 0x0C schaltet es wieder ein.

## Wort in einer Zeile scrollen

cmd(0x0C);

string ("cc - labor");

while(1)

{

cmd(0x18); // display shift left

delay\_ms(750);

}

Listing 5. - Laufschrift

Das senden des HEX Wertes 0x18 schiebt alle Zeichen um eine Stelle nach links. Durch die Endlosschleife und dem Delay ergibt sich eine Laufschrift. Das delay ist so konzipiert, dass es eine Verzögerung von etwa 750ms zur Folge hat.

## Eigenes Zeichen kreieren und anzeigen

Ein eigenes Zeichen wird definiert, indem das Zeichen in den CGRAM (Character Generator RAM) geladen wird. Danach kann man es wie ein normal definiertes Zeichen aufrufen. Jeder Pixel wird einem Bit zugeordnet, daher sind pro Zeile 5Bit notwendig. In das CGRAM werden 8 Mal 5 Bit geladen, was ein Zeichen ergibt. Um die nötigen Werte zu ermitteln wurde ein Programm im Internet verwendet.

Der Link dazu: <http://www.8051projects.net/lcd-interfacing/lcd-custom-character.php>

Per Mausklick werden die gewünschten Pixel aktiviert bzw. deaktiviert.

Der Code um das eigene Zeichen zu erzeugen und anzuzeigen lautet nun wie folgt.

cmd(0x40); //Laden der Speicheradresse

data(0x15); //Lade Daten Zeile 1

data(0x15); //Lade Daten Zeile 2

data(0x15); //Lade Daten Zeile 3

data(0x15); //Lade Daten Zeile 4

data(0x15); //Lade Daten Zeile 5

data(0x15); //Lade Daten Zeile 6

data(0x15); //Lade Daten Zeile 7

data(0x15); //Lade Daten Zeile 8

cmd(0x0C);

data (0x00); //Eigener Character 0

Listing 6. - Eigener Character

Das Zeichen muss zuerst in den CCRAM (Character Generator RAM) geladen werden. Der Befehl 0x40 wählt die Stelle aus, in diesem Fall die Stelle 0. Danach werden zeilenweise die Pixel des Zeichens angegeben. Um das Zeichen der Stelle 0 auszugeben, wird der Befehl 0x00 verwendet.

## Von 0 bis 100 zählen

Da das Display den ASCII-Code zur Anzeige verwendet, muss ein programmtechnischer Integer-Zähler vor der Ausgabe in einen Character-Wert umgewandelt werden.

while(1)

{

temp = "000";

for(int i=0;i<=100;i++)

{ cmd(0xC0);

sprintf(temp,"%3d", i);

string(temp);

}

}

Listing 7. - Zählen von 0 bis 100

# Program-Listing Verzeichnis

Listing 1. - Header File zur Ansteurung 8

Listing 2. - Zeichen ausgeben 9

Listing 3. - Wort ausgeben 9

Listing 4. - Blinken 9

Listing 5. - Laufschrift 9

Listing 6. - Eigener Character 10

Listing 7. - Zählen von 0 bis 100 10