|  |  |  |  |
| --- | --- | --- | --- |
| **Übungsprotokoll**  **SYTx – Systemtechnik <Teilbereich>** | | | |
|  | **Übungsdatum:**  KW 00/17 – 99/17 | **Klasse:**  5AHIT | **Name:**  Yi Liu |
| **Abgabedatum:**  13.3.2024 | **Gruppe:**  SYTI | **Note:** |
| **Leitung:**  DI (FH) Markus BRUNNER | **Mitübende:**  Gegebenenfalls hier anführen, muss mit Aufgabenteilung in der Aufgabenstellung korrespondieren! | | |
| **Übungsbezeichnung**:  Messomat 7k | | | |

**Inhaltsverzeichnis:**

[1 Aufgabenstellung 2](#_Toc193956689)

[2 Abstract (English) 3](#_Toc193956690)

[3 Theoretische Grundlagen 4](#_Toc193956691)

[3.1 Mikrocontroller ATMega328p 4](#_Toc193956692)

[3.2 Sensoren & Peripheriegeräte 4](#_Toc193956693)

[3.3 Kommunikation & Datenübertragung 4](#_Toc193956694)

[3.4 Embedded-C Programmierung 4](#_Toc193956695)

[3.5 Fehlerbehandlung 4](#_Toc193956696)

[4 Schaltplan 5](#_Toc193956697)

[5 Übungsdurchführung / Überlegungen beim Code-Design 6](#_Toc193956698)

[5.1 Timer-Initialisierung 6](#_Toc193956699)

[5.2 UART-Kommunikation 7](#_Toc193956700)

[5.3 EEPROM-Datenspeicherung 8](#_Toc193956701)

[5.4 LCD-Anzeige 9](#_Toc193956702)

[5.5 Protokollengineering 10](#_Toc193956703)

[5.6 ACK-Bestätigung 10](#_Toc193956705)

[6 Ergebnisse 10](#_Toc193956707)

[7 Code 11](#_Toc193956708)

[8 Ergebnis 22](#_Toc193956711)

[9 Kommentar 22](#_Toc193956712)

# Aufgabenstellung

Ziel ist die Entwicklung eines intelligenten Temperatur- und Feuchtigkeitsüberwachungssystems, das (Mess-)Daten erfasst und an einen Rechner übermittelt. Zur Visualisierung Daten und Steuerung der Messanwendung ist im ersten Schritt eine einfache Terminalanwendung einzusetzen.

Weitere genauere Anforderungen sind im Dokument „*SYTI5\_UE2-Messomat7k-v1.4.pdf“* zu sehen.

# Abstract (English)

The aim is to develop an intelligent temperature and humidity monitoring system that records (measurement) data and transmits it to a computer. The first step is to use a simple terminal application to visualize data and control the measurement application.

More detailed information can be found in the document *“SYTI5\_UE2-Messomat7k-v1.4.pdf”*.

# Theoretische Grundlagen

Die theoretischen Grundlagen für die Übung mit dem ATMega328p und C-Programmierung umfassen mehrere Bereiche:

## ****Mikrocontroller ATMega328p****

1. Architektur: 8-Bit RISC, 32 Register, interne Peripheriegeräte (ADC, Timer, UART etc.)
2. Speicher: Flash (für Code), SRAM (für Variablen), EEPROM (für nichtflüchtige Speicherung)
3. GPIOs: Steuerung von Sensoren, Aktoren und LCD
4. Energie-Modi: Normalbetrieb vs. Energiesparmodus

## ****Sensoren & Peripheriegeräte****

1. **Temperatur- und Feuchtigkeitssensor** (z. B. DHT11, DHT22 oder SHT3x)
2. **Hochleistungsventilator** (Steuerung per PWM oder einfacher GPIO-Schaltung)
3. **LCD-Display** (z. B. HD44780, I2C- oder SPI-Ansteuerung)
4. **Taster-Eingaben** (z. B. Interrupt-gesteuert oder per Polling)

## ****Kommunikation & Datenübertragung****

1. **Serielle Kommunikation (UART)** zur Datenübertragung an den Host-Computer
2. **Protokoll mit STX/ETX** zur Rahmenbildung der Datenpakete
3. **Fehlermanagement**: Erkennung von Verbindungsabbrüchen, ACK/NACK-Prüfungen

## ****Embedded-C Programmierung****

1. Verwendung von **Header-Dateien (.h)** für systemweite Definitionen
2. **Bedingte Kompilierung** (Mock-Mode für Testzwecke)
3. **Interrupts & Timer** für präzise Steuerung (z. B. 1s vs. 4s Messintervall)
4. **EEPROM-Speicherung** zur Sicherung der letzten 10 Messwerte

## ****Fehlerbehandlung****

1. **Fehlermanagement** durch Reset-Taste oder Software-Reset
2. **LED-Statusanzeige** für Verbindungsstatus

# Schaltplan

Verwendete Teile sind:

* ATMega328p Board
* 2 LEDs
* DHT-7k
* LCD

Ein Bild, das Text, Diagramm, Plan, technische Zeichnung enthält.

KI-generierte Inhalte können fehlerhaft sein.

# Übungsdurchführung / Überlegungen beim Code-Design

## ****Timer-Initialisierung****

Die Messungen sollen in festen Intervallen (1 oder 4 Sekunden) erfolgen.

Ein Timer-Interrupt ist effizienter als eine \_delay\_ms()-Schleife, da er den Prozessor nicht blockiert.

**Vorgang:**

Timer1 im CTC-Modus (Clear Timer on Compare Match) mit Prescaler 1024.

Compare Match Value:

**1 Sekunde:** OCR1A = 15624 (16.000.000 Hz / 1024 / 1 Hz - 1)

**4 Sekunden:** OCR1A = 62499.

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.}

## ****UART-Kommunikation****

Die Daten müssen zuverlässig an das Host-System übertragen werden.

Ein Framing-Protokoll (STX/ETX) sorgt für eine klare Struktur der Nachrichten.

ACK-Bestätigungen gewährleisten, dass die Daten korrekt empfangen wurden.

**Vorgang:**

**Framing:** Jede Nachricht wird mit <STX> (0x02) und <ETX> (0x03) gerahmt.

Beispiel: <STX>DATE22|HU55|SN1<ETX>.

**Retry-Logik:** Nach 3 fehlgeschlagenen Übertragungen:

LED leuchtet (**PB0**).

Messdaten werden im EEPROM gespeichert.

void send\_data() {

sprintf(txBuffer, "\x02DATE%d|HU%d|SN%d\x03", currentTemp, currentHumidity, seqNumber);

uart\_puts(txBuffer);

retryCount++;

if (retryCount >= 3) PORTB |= (1 << STATUS\_LED\_PIN); // Fehler-LED

}

## **EEPROM-Datenspeicherung**

Bei Verbindungsabbrüchen sollen die letzten 10 Messwerte gespeichert werden, um sie später erneut zu senden.

Ein Ringpuffer (Kreis) ist effizient, da er Speicherplatz wiederverwendet und keine Verschiebung der Daten erfordert.

**Vorgang:**

* **Ringpuffer-Struktur:**
  + Byte 0: Aktuelle Position im Puffer.
  + Byte 1: Letzte Sequenznummer.
  + Byte 2–21: Temperatur- und Feuchtigkeitsdaten (10 × 2 Byte).

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

**Vorteile des Ringpuffers:**

**Speichereffizienz:** Es werden nur 22 Byte benötigt.

**Einfache Implementierung:** Die Position wird durch Modulo-Operation automatisch zurückgesetzt.

**Robustheit:** Daten gehen nicht verloren, selbst wenn der Puffer voll ist.

## ****LCD-Anzeige****

**Zeile 1:** T:22C H:55%

**Zeile 2:** F:ON I:1s ERR! (bei Verbindungsabbruch).

Ein Bild, das Text, Screenshot, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

## ****Protokollengineering****

### ****Framing (STX/ETX)****

Ein Rahmen (STX/ETX) ermöglicht es dem Empfänger, den Anfang und das Ende einer Nachricht zu erkennen. Dies ist besonders wichtig bei serieller Kommunikation, da Datenströme kontinuierlich sind.

**Implementierung:**

* Jede Nachricht beginnt mit <STX> (0x02) und endet mit <ETX> (0x03).
* Beispiel: <STX>DATE22|HU55|SN1<ETX>.

## ****ACK-Bestätigung****

Der Sender muss sicherstellen, dass die Nachricht korrekt empfangen wurde. Ein ACK (0x06) bestätigt den erfolgreichen Empfang.

**Implementierung:**

* Der Host sendet 0x06 nach erfolgreichem Empfang.
* Der Mikrocontroller setzt den Retry-Zähler zurück und erhöht die Sequenznummer.

#### ****C. Sequenznummern****

Sequenznummern ermöglichen es, Nachrichten in der richtigen Reihenfolge zu verarbeiten. Sie helfen auch bei der Identifikation von verlorenen oder doppelten Nachrichten.

**Implementierung:**

* Jede Nachricht enthält eine Sequenznummer (SNx).
* Die Sequenznummer wird nach jedem erfolgreichen ACK erhöht.

# Ergebnisse

Beschreibung und Darstellung der Ergebnisse in geeigneter Form, z. B. Screenshots, Tabellen, ...

# Code

## Main.c

/\*

\* Messomat.c

\*

\* Created: 2024/12/5 13:16:47

\* Author : Yi

\*/

#define *F\_CPU* 16000000UL

#define UART\_BAUD\_RATE 9600

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include <avr/eeprom.h>

#include "dht.h"

#include "lcd.h"

#include "uart.h"

//Reservierungen

#define STATUS\_LED\_PIN PORTB0

#define FAN\_PIN PORTB1

#define BTN\_T1\_PIN PORTD2

#define BTN\_T2\_PIN PORTD3

#define EEPROM\_SIZE 22 // 2 header + 10\*2 data

// Globale Statuse

volatile *uint8\_t* interval = 1;

volatile *uint8\_t* sendFlag = 0;

volatile *uint8\_t* seqNumber = 1;

volatile *uint8\_t* ackReceived = 0;

volatile *uint8\_t* retryCount = 0;

volatile *uint8\_t* measureTimeFlag = 0;

volatile *uint8\_t* fanStatus = 0; // 0=off, 1=on

volatile *uint8\_t* connectionLost = 0;

// EEPROM Specific

*uint8\_t* EEMEM eepromStorage[EEPROM\_SIZE];

//Sensor Daten

volatile *int8\_t* currentTemp;

volatile *int8\_t* currentHumidity;

volatile char displayBuffer[20];

volatile char txBuffer[32];

void timer1\_init(void) { // Set Timer1 für CTC-Modus

TCCR1B |= (1 << WGM12); // CTC-Modus

OCR1A = 15624; // 1 Sekunde bei 16 MHz und Prescaler 1024

TCCR1B |= (1 << CS12) | (1 << CS10); // Prescaler 1024

TIMSK1 |= (1 << OCIE1A); // Output Compare A Interrupt aktivieren

}

ISR(TIMER1\_COMPA\_vect) {

measureTimeFlag = 1;

}

void update\_display(*uint8\_t* stopped) {

lcd\_clrscr();

if(stopped) {

lcd\_puts("\*\*ME gestoppt\*\*");

} else {

// Measurements

*sprintf*(displayBuffer, "T:%dC H:%d%%", currentTemp, currentHumidity);

lcd\_puts(displayBuffer);

// Fan status & interval

lcd\_gotoxy(0,1);

*sprintf*(displayBuffer, "F:%s I:%ds %s",

fanStatus ? "ON " : "OFF",

interval,

connectionLost ? "ERR!" : " ");

lcd\_puts(displayBuffer);

}

}

void send\_data(){

*sprintf*(txBuffer, "DATE%d|HU%d|SN%d", currentTemp, currentHumidity, seqNumber);

uart\_putc(0x02);

uart\_puts(txBuffer);

uart\_putc(0x03);

retryCount++;

}

// EEPROM: erste Stelle sag die neuest gespeicherten Daten aus, die zweite die Seriennummer

// folgenden 10\*2 positionen bilden ein Kreis

void store\_in\_eeprom(){

*uint8\_t* pos = eeprom\_read\_byte(&eepromStorage[0]);

if(pos >= 10) pos = 0;

eeprom\_write\_byte(&eepromStorage[pos\*2 + 2], currentTemp);

eeprom\_write\_byte(&eepromStorage[pos\*2 + 3], currentHumidity);

eeprom\_write\_byte(&eepromStorage[0], pos + 1);

eeprom\_write\_byte(&eepromStorage[1], seqNumber);

}

void resend\_eeprom\_data(){

*uint8\_t* start\_pos = eeprom\_read\_byte(&eepromStorage[0]);

*uint8\_t* start\_seq = eeprom\_read\_byte(&eepromStorage[1]) > 10

? eeprom\_read\_byte(&eepromStorage[1]) - 10

: 0;

for(*uint8\_t* i=0; i<10; i++) {

*uint8\_t* pos = (start\_pos + i) % 10;

*int8\_t* temp = eeprom\_read\_byte(&eepromStorage[pos\*2 + 2]);

*int8\_t* hum = eeprom\_read\_byte(&eepromStorage[pos\*2 + 3]);

*sprintf*(txBuffer, "DATE%d|HU%d|SN%d", temp, hum, start\_seq + i);

uart\_putc(0x02);

uart\_puts(txBuffer);

uart\_putc(0x03);

}

}

void check\_input(void){

char command = uart\_getc();

switch (command) {

case 0x06:

ackReceived = 1;

if(connectionLost) {

connectionLost = 0;

resend\_eeprom\_data();

}

break;

case '1':

OCR1A = 15624;

interval = 1;

TCNT1 = 0;// Reset timer

update\_display(0);

break;

case '4':

OCR1A = 62499;

interval = 4;

TCNT1 = 0; // Reset timer

update\_display(0);

break;

case 'd':

sendFlag = 1;

retryCount = 0; // Reset retry counter

update\_display(0);

break;

case 'q':

sendFlag = 0;

PORTB &= ~(1 << FAN\_PIN);

fanStatus = 0;

update\_display(1);

break;

case 'e':

PORTB |= (1 << FAN\_PIN);

fanStatus = 1;

update\_display(0);

break;

case 'a':

PORTB &= ~(1 << FAN\_PIN);

fanStatus = 0;

update\_display(0);

break;

case 's':

uart\_putc(0x02);

uart\_puts(fanStatus ? "FAN1" : "FAN0");

uart\_putc(0x03);

break;

case 'r': // Reset command

retryCount = 0;

ackReceived = 0;

connectionLost = 0;

PORTB &= ~(1 << STATUS\_LED\_PIN);

break;

}

}

int main(void)

{

lcd\_init(LCD\_DISP\_ON);

lcd\_clrscr();

uart\_init(UART\_BAUD\_SELECT(UART\_BAUD\_RATE,*F\_CPU*));

timer1\_init();

//I/O-Konfigurationen

DDRB |=(1<< STATUS\_LED\_PIN)|(1<<FAN\_PIN);//DDB0/DDB1

PORTB &= ~((1 << PORTB0) | (1 << PORTB1));// Sicherstellen dass 2 LEDs am Anfang ausgeschaltet sind

//Buttons (unnoetig)

DDRD &= ~((1 << BTN\_T1\_PIN)|(1 << BTN\_T2\_PIN));

PORTD |= (1 << BTN\_T1\_PIN)|(1 << BTN\_T2\_PIN); // pull-ups einschalten

*sei*();

update\_display(1);

while(1){

check\_input();

if(ackReceived) {

PORTB &= ~(1 << STATUS\_LED\_PIN); // Turn off LED

seqNumber++;

retryCount = 0;

ackReceived = 0;

}

if(measureTimeFlag) {

measureTimeFlag = 0;

if(dht\_gettemperaturehumidity(&currentTemp, &currentHumidity) == DHT\_ERROR\_NOERR) {

store\_in\_eeprom(); //alle Messungen werden gespeichert

update\_display(0);

if (sendFlag) {

if (retryCount < 3) {

send\_data();

} else {

PORTB |= (1 << STATUS\_LED\_PIN); // Turn on LED after 3 retries

connectionLost = 1;

}

}

}

}

// Handle LED state

/\*if (retryCount >= 3 && !ackReceived) {

PORTB |= (1 << PORTB0);

}\*/

}

}

## Lcd.h

#ifndef LCD\_H

#define LCD\_H

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Title : C include file for the HD44780U LCD library (lcd.c)

Author: Peter Fleury <pfleury@gmx.ch> http://tinyurl.com/peterfleury

File: $Id: lcd.h,v 1.14.2.4 2015/01/20 17:16:07 peter Exp $

Software: AVR-GCC 4.x

Hardware: any AVR device, memory mapped mode only for AVR with

memory mapped interface (AT90S8515/ATmega8515/ATmega128)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*

@mainpage

Collection of libraries for AVR-GCC

@author Peter Fleury pfleury@gmx.ch http://tinyurl.com/peterfleury

@copyright (C) 2015 Peter Fleury, GNU General Public License Version 3

@file

@defgroup pfleury\_lcd LCD library <lcd.h>

@code #include <lcd.h> @endcode

@brief Basic routines for interfacing a HD44780U-based character LCD display

LCD character displays can be found in many devices, like espresso machines, laser printers.

The Hitachi HD44780 controller and its compatible controllers like Samsung KS0066U have become an industry standard for these types of displays.

This library allows easy interfacing with a HD44780 compatible display and can be

operated in memory mapped mode (LCD\_IO\_MODE defined as 0 in the include file lcd.h.) or in

4-bit IO port mode (LCD\_IO\_MODE defined as 1). 8-bit IO port mode is not supported.

Memory mapped mode is compatible with old Kanda STK200 starter kit, but also supports

generation of R/W signal through A8 address line.

@see The chapter <a href=" http://homepage.hispeed.ch/peterfleury/avr-lcd44780.html" target="\_blank">Interfacing a HD44780 Based LCD to an AVR</a>

on my home page, which shows example circuits how to connect an LCD to an AVR controller.

@author Peter Fleury pfleury@gmx.ch http://tinyurl.com/peterfleury

@version 2.0

@copyright (C) 2015 Peter Fleury, GNU General Public License Version 3

\*/

#include <inttypes.h>

#include <avr/pgmspace.h>

#if (\_\_GNUC\_\_ \* 100 + \_\_GNUC\_MINOR\_\_) < 405

#error "This library requires AVR-GCC 4.5 or later, update to newer AVR-GCC compiler !"

#endif

/\*\*@{\*/

/\*

\* LCD and target specific definitions below can be defined in a separate include file with name lcd\_definitions.h instead modifying this file

\* by adding -D\_LCD\_DEFINITIONS\_FILE to the CDEFS section in the Makefile

\* All definitions added to the file lcd\_definitions.h will override the default definitions from lcd.h

\*/

#ifdef \_LCD\_DEFINITIONS\_FILE

#include "lcd\_definitions.h"

#endif

/\*\*

\* @name Definition for LCD controller type

\* Use 0 for HD44780 controller, change to 1 for displays with KS0073 controller.

\*/

#ifndef LCD\_CONTROLLER\_KS0073

#define LCD\_CONTROLLER\_KS0073 0 /\*\*< Use 0 for HD44780 controller, 1 for KS0073 controller \*/

#endif

/\*\*

\* @name Definitions for Display Size

\* Change these definitions to adapt setting to your display

\*

\* These definitions can be defined in a separate include file \b lcd\_definitions.h instead modifying this file by

\* adding -D\_LCD\_DEFINITIONS\_FILE to the CDEFS section in the Makefile.

\* All definitions added to the file lcd\_definitions.h will override the default definitions from lcd.h

\*

\*/

#ifndef LCD\_LINES

#define LCD\_LINES 2 /\*\*< number of visible lines of the display \*/

#endif

#ifndef LCD\_DISP\_LENGTH

#define LCD\_DISP\_LENGTH 16 /\*\*< visibles characters per line of the display \*/

#endif

#ifndef LCD\_LINE\_LENGTH

#define LCD\_LINE\_LENGTH 0x40 /\*\*< internal line length of the display \*/

#endif

#ifndef LCD\_START\_LINE1

#define LCD\_START\_LINE1 0x00 /\*\*< DDRAM address of first char of line 1 \*/

#endif

#ifndef LCD\_START\_LINE2

#define LCD\_START\_LINE2 0x40 /\*\*< DDRAM address of first char of line 2 \*/

#endif

#ifndef LCD\_START\_LINE3

#define LCD\_START\_LINE3 0x14 /\*\*< DDRAM address of first char of line 3 \*/

#endif

#ifndef LCD\_START\_LINE4

#define LCD\_START\_LINE4 0x54 /\*\*< DDRAM address of first char of line 4 \*/

#endif

#ifndef LCD\_WRAP\_LINES

#define LCD\_WRAP\_LINES 0 /\*\*< 0: no wrap, 1: wrap at end of visibile line \*/

#endif

/\*\*

\* @name Definitions for 4-bit IO mode

\*

\* The four LCD data lines and the three control lines RS, RW, E can be on the

\* same port or on different ports.

\* Change LCD\_RS\_PORT, LCD\_RW\_PORT, LCD\_E\_PORT if you want the control lines on

\* different ports.

\*

\* Normally the four data lines should be mapped to bit 0..3 on one port, but it

\* is possible to connect these data lines in different order or even on different

\* ports by adapting the LCD\_DATAx\_PORT and LCD\_DATAx\_PIN definitions.

\*

\* Adjust these definitions to your target.\n

\* These definitions can be defined in a separate include file \b lcd\_definitions.h instead modifying this file by

\* adding \b -D\_LCD\_DEFINITIONS\_FILE to the \b CDEFS section in the Makefile.

\* All definitions added to the file lcd\_definitions.h will override the default definitions from lcd.h

\*

\*/

#define LCD\_IO\_MODE 1 /\*\*< 0: memory mapped mode, 1: IO port mode \*/

#if LCD\_IO\_MODE

#ifndef LCD\_PORT

#define LCD\_PORT PORTA /\*\*< port for the LCD lines \*/

#endif

#ifndef LCD\_DATA0\_PORT

#define LCD\_DATA0\_PORT LCD\_PORT /\*\*< port for 4bit data bit 0 \*/

#endif

#ifndef LCD\_DATA1\_PORT

#define LCD\_DATA1\_PORT LCD\_PORT /\*\*< port for 4bit data bit 1 \*/

#endif

#ifndef LCD\_DATA2\_PORT

#define LCD\_DATA2\_PORT LCD\_PORT /\*\*< port for 4bit data bit 2 \*/

#endif

#ifndef LCD\_DATA3\_PORT

#define LCD\_DATA3\_PORT LCD\_PORT /\*\*< port for 4bit data bit 3 \*/

#endif

#ifndef LCD\_DATA0\_PIN

#define LCD\_DATA0\_PIN 4 /\*\*< pin for 4bit data bit 0 \*/

#endif

#ifndef LCD\_DATA1\_PIN

#define LCD\_DATA1\_PIN 5 /\*\*< pin for 4bit data bit 1 \*/

#endif

#ifndef LCD\_DATA2\_PIN

#define LCD\_DATA2\_PIN 6 /\*\*< pin for 4bit data bit 2 \*/

#endif

#ifndef LCD\_DATA3\_PIN

#define LCD\_DATA3\_PIN 7 /\*\*< pin for 4bit data bit 3 \*/

#endif

#ifndef LCD\_RS\_PORT

#define LCD\_RS\_PORT PORTC /\*\*< port for RS line \*/

#endif

#ifndef LCD\_RS\_PIN

#define LCD\_RS\_PIN 5 /\*\*< pin for RS line \*/

#endif

#ifndef LCD\_RW\_PORT

#define LCD\_RW\_PORT PORTC /\*\*< port for RW line \*/

#endif

#ifndef LCD\_RW\_PIN

#define LCD\_RW\_PIN 4 /\*\*< pin for RW line \*/

#endif

#ifndef LCD\_E\_PORT

#define LCD\_E\_PORT PORTC /\*\*< port for Enable line \*/

#endif

#ifndef LCD\_E\_PIN

#define LCD\_E\_PIN 3 /\*\*< pin for Enable line \*/

#endif

#elif defined(\_\_AVR\_AT90S4414\_\_) || defined(\_\_AVR\_AT90S8515\_\_) || defined(\_\_AVR\_ATmega64\_\_) || \

defined(\_\_AVR\_ATmega8515\_\_)|| defined(\_\_AVR\_ATmega103\_\_) || defined(\_\_AVR\_ATmega128\_\_) || \

defined(\_\_AVR\_ATmega161\_\_) || defined(\_\_AVR\_ATmega162\_\_)

/\*

\* memory mapped mode is only supported when the device has an external data memory interface

\*/

#define LCD\_IO\_DATA 0xC000 /\* A15=E=1, A14=RS=1 \*/

#define LCD\_IO\_FUNCTION 0x8000 /\* A15=E=1, A14=RS=0 \*/

#define LCD\_IO\_READ 0x0100 /\* A8 =R/W=1 (R/W: 1=Read, 0=Write \*/

#else

#error "external data memory interface not available for this device, use 4-bit IO port mode"

#endif

/\*\*

\* @name Definitions of delays

\* Used to calculate delay timers.

\* Adapt the F\_CPU define in the Makefile to the clock frequency in Hz of your target

\*

\* These delay times can be adjusted, if some displays require different delays.\n

\* These definitions can be defined in a separate include file \b lcd\_definitions.h instead modifying this file by

\* adding \b -D\_LCD\_DEFINITIONS\_FILE to the \b CDEFS section in the Makefile.

\* All definitions added to the file lcd\_definitions.h will override the default definitions from lcd.h

\*/

#ifndef LCD\_DELAY\_BOOTUP

#define LCD\_DELAY\_BOOTUP 16000 /\*\*< delay in micro seconds after power-on \*/

#endif

#ifndef LCD\_DELAY\_INIT

#define LCD\_DELAY\_INIT 5000 /\*\*< delay in micro seconds after initialization command sent \*/

#endif

#ifndef LCD\_DELAY\_INIT\_REP

#define LCD\_DELAY\_INIT\_REP 64 /\*\*< delay in micro seconds after initialization command repeated \*/

#endif

#ifndef LCD\_DELAY\_INIT\_4BIT

#define LCD\_DELAY\_INIT\_4BIT 64 /\*\*< delay in micro seconds after setting 4-bit mode \*/

#endif

#ifndef LCD\_DELAY\_BUSY\_FLAG

#define LCD\_DELAY\_BUSY\_FLAG 4 /\*\*< time in micro seconds the address counter is updated after busy flag is cleared \*/

#endif

#ifndef LCD\_DELAY\_ENABLE\_PULSE

#define LCD\_DELAY\_ENABLE\_PULSE 1 /\*\*< enable signal pulse width in micro seconds \*/

#endif

/\*\*

\* @name Definitions for LCD command instructions

\* The constants define the various LCD controller instructions which can be passed to the

\* function lcd\_command(), see HD44780 data sheet for a complete description.

\*/

/\* instruction register bit positions, see HD44780U data sheet \*/

#define LCD\_CLR 0 /\* DB0: clear display \*/

#define LCD\_HOME 1 /\* DB1: return to home position \*/

#define LCD\_ENTRY\_MODE 2 /\* DB2: set entry mode \*/

#define LCD\_ENTRY\_INC 1 /\* DB1: 1=increment, 0=decrement \*/

#define LCD\_ENTRY\_SHIFT 0 /\* DB2: 1=display shift on \*/

#define LCD\_ON 3 /\* DB3: turn lcd/cursor on \*/

#define LCD\_ON\_DISPLAY 2 /\* DB2: turn display on \*/

#define LCD\_ON\_CURSOR 1 /\* DB1: turn cursor on \*/

#define LCD\_ON\_BLINK 0 /\* DB0: blinking cursor ? \*/

#define LCD\_MOVE 4 /\* DB4: move cursor/display \*/

#define LCD\_MOVE\_DISP 3 /\* DB3: move display (0-> cursor) ? \*/

#define LCD\_MOVE\_RIGHT 2 /\* DB2: move right (0-> left) ? \*/

#define LCD\_FUNCTION 5 /\* DB5: function set \*/

#define LCD\_FUNCTION\_8BIT 4 /\* DB4: set 8BIT mode (0->4BIT mode) \*/

#define LCD\_FUNCTION\_2LINES 3 /\* DB3: two lines (0->one line) \*/

#define LCD\_FUNCTION\_10DOTS 2 /\* DB2: 5x10 font (0->5x7 font) \*/

#define LCD\_CGRAM 6 /\* DB6: set CG RAM address \*/

#define LCD\_DDRAM 7 /\* DB7: set DD RAM address \*/

#define LCD\_BUSY 7 /\* DB7: LCD is busy \*/

/\* set entry mode: display shift on/off, dec/inc cursor move direction \*/

#define LCD\_ENTRY\_DEC 0x04 /\* display shift off, dec cursor move dir \*/

#define LCD\_ENTRY\_DEC\_SHIFT 0x05 /\* display shift on, dec cursor move dir \*/

#define LCD\_ENTRY\_INC\_ 0x06 /\* display shift off, inc cursor move dir \*/

#define LCD\_ENTRY\_INC\_SHIFT 0x07 /\* display shift on, inc cursor move dir \*/

/\* display on/off, cursor on/off, blinking char at cursor position \*/

#define LCD\_DISP\_OFF 0x08 /\* display off \*/

#define LCD\_DISP\_ON 0x0C /\* display on, cursor off \*/

#define LCD\_DISP\_ON\_BLINK 0x0D /\* display on, cursor off, blink char \*/

#define LCD\_DISP\_ON\_CURSOR 0x0E /\* display on, cursor on \*/

#define LCD\_DISP\_ON\_CURSOR\_BLINK 0x0F /\* display on, cursor on, blink char \*/

/\* move cursor/shift display \*/

#define LCD\_MOVE\_CURSOR\_LEFT 0x10 /\* move cursor left (decrement) \*/

#define LCD\_MOVE\_CURSOR\_RIGHT 0x14 /\* move cursor right (increment) \*/

#define LCD\_MOVE\_DISP\_LEFT 0x18 /\* shift display left \*/

#define LCD\_MOVE\_DISP\_RIGHT 0x1C /\* shift display right \*/

/\* function set: set interface data length and number of display lines \*/

#define LCD\_FUNCTION\_4BIT\_1LINE 0x20 /\* 4-bit interface, single line, 5x7 dots \*/

#define LCD\_FUNCTION\_4BIT\_2LINES 0x28 /\* 4-bit interface, dual line, 5x7 dots \*/

#define LCD\_FUNCTION\_8BIT\_1LINE 0x30 /\* 8-bit interface, single line, 5x7 dots \*/

#define LCD\_FUNCTION\_8BIT\_2LINES 0x38 /\* 8-bit interface, dual line, 5x7 dots \*/

#define LCD\_MODE\_DEFAULT ((1<<LCD\_ENTRY\_MODE) | (1<<LCD\_ENTRY\_INC) )

/\*\*

\* @name Functions

\*/

/\*\*

@brief Initialize display and select type of cursor

@param dispAttr \b LCD\_DISP\_OFF display off\n

\b LCD\_DISP\_ON display on, cursor off\n

\b LCD\_DISP\_ON\_CURSOR display on, cursor on\n

\b LCD\_DISP\_ON\_CURSOR\_BLINK display on, cursor on flashing

@return none

\*/

extern void lcd\_init(*uint8\_t* dispAttr);

/\*\*

@brief Clear display and set cursor to home position

@return none

\*/

extern void lcd\_clrscr(void);

/\*\*

@brief Set cursor to home position

@return none

\*/

extern void lcd\_home(void);

/\*\*

@brief Set cursor to specified position

@param x horizontal position\n (0: left most position)

@param y vertical position\n (0: first line)

@return none

\*/

extern void lcd\_gotoxy(*uint8\_t* x, *uint8\_t* y);

/\*\*

@brief Display character at current cursor position

@param c character to be displayed

@return none

\*/

extern void lcd\_putc(char c);

/\*\*

@brief Display string without auto linefeed

@param s string to be displayed

@return none

\*/

extern void lcd\_puts(const char \*s);

/\*\*

@brief Display string from program memory without auto linefeed

@param progmem\_s string from program memory be be displayed

@return none

@see lcd\_puts\_P

\*/

extern void lcd\_puts\_p(const char \*progmem\_s);

/\*\*

@brief Send LCD controller instruction command

@param cmd instruction to send to LCD controller, see HD44780 data sheet

@return none

\*/

extern void lcd\_command(*uint8\_t* cmd);

/\*\*

@brief Send data byte to LCD controller

Similar to lcd\_putc(), but without interpreting LF

@param data byte to send to LCD controller, see HD44780 data sheet

@return none

\*/

extern void lcd\_data(*uint8\_t* data);

/\*\*

@brief macros for automatically storing string constant in program memory

\*/

#define lcd\_puts\_P(\_\_s) lcd\_puts\_p(PSTR(\_\_s))

/\*\*@}\*/

#endif //LCD\_H

# Ergebnis

Blazor Visualisierung:

Ein Bild, das Text, Screenshot, Diagramm, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

Breadboard:

Ein Bild, das Elektronik, Elektrische Leitungen, Kabel, Elektrisches Bauelement enthält.

KI-generierte Inhalte können fehlerhaft sein.

# Kommentar

Die Übung war *frrrrrrrr* FIRE